12-2016

Implementing Mongo DB with Spring MVC to Support Shared Student Web Space

Ashish Kastury
St Cloud State University, akastury@stcloudstate.edu

Implementing Mongo DB with Spring MVC to Support Shared Student Web Space

by

Ashish Kastury

A Starred Paper

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in Information Assurance

November, 2016

Starred Paper Committee:
Dennis Guster, Chairperson
Lynn Collen
Kasi Balasubramanian

Abstract

The data storage and retrieval process has always been a challenge for I.T. professionals. Many organizations have created different strategies to persist data. Yet, in this era of big data, retrieving information from the data which is stored, affects the efficiency of the system. Many algorithms have been developed to increase the speed of data retrieval. But still, algorithms don't change the structure of the data and the schema. The traditional way of storing data is the RDBMS (Relational Database Management System), which has been the de facto standard across many sectors, such as banking, insurance, retail, etc. But, unstructured data cannot be handled efficiently by the RDBMS. This is because there may not be a relation (such as a primary orforeign key) between different sets of data. NoSQL databases fill this gap of storing the data without any such relation. But as the name implies, NoSQL means "NOT ONLY SQL" i.e. it also behaves like SQL and handle the SQL queries. The NoSQL concept is preferably used in big data and real-time web applications. Many organizations have designed their databases with the NoSQL concept but are central to their database implementations. This paper is intended to use Mongo DB, which is a NoSQL database, which stores data in the form of "JSON" documents (not as rows and columns as in a relational database). Also, the database is implemented using the Spring MVC framework to check how the queries can be initiated from a web application to store and retrieve data.

Table of Contents

List of Tables

Chapter I: Introduction

The data retrieval process is a pressing issue which needs to be synchronized with the exponential growth of data. Unstructured data cannot be stored in a relational database management systems (RDBMS). Maintaining rows and columns based structure for the data is suitable for only a few domains; for example, to store the account summary of a particular customer in a banking domain. In this instance, the data is structured as it is entered in such a way that it makes a relation for each customer to their account id, balance, withdrawals, etc. But, in the case of Google or Facebook, the data is unstructured as the users enter the search engine or social media site in an anonymous way. All the entered data will be stored in the database with the help of algorithms, and it will be produced again when required. In the case of Facebook, when a user enters a post, it can be commented/liked by other users. Further, this post will be visible to other users who are in the friends list of the people who earlier commented or liked on the post.

In this project, Mongo DB (NoSQL database) is implemented with the Spring MVC framework to build a website where students can post data and communicate with each other. Mongo DB stores the data in the form of JSON documents. It is also a schema-less database, which means the documents may not be in a similar structure. This paper shows how a NoSQL database can be accessed from a JAVA point of view, and how the data is pushed and retrieved into MongoDB using simple queries.

Problem Statement

As there is no common and secure website for students to communicate with each other, this paper proposes to fill the gap by which students can easily communicate with one another. Facebook and other social networking websites can be used, but it is not safe as many unauthorized people will be able to access the site. The purpose of this project is to provide an integrated website which offers students to find study materials, transport, post questions and answers and more services in a secure way.

Nature and Significance of the Problem

This website is secure as students use their college ID to login to the website. Unknown people without a college ID cannot access this website. Also, this website is implemented with a NoSQL database which offers a more efficient way of retrieving the data created.

Objective of the Study

The objective of this study is to build a website by implementing MongoDB with Spring MVC framework. This website shows the ease of accessing the database with a minimum of queries, unlike an RDBMS.

Hypotheses

Implementing this project raises a few questions such as:

1. Is MongoDB really more useful in retrieving and storing data compared to an RDBMS?
2. Is it efficient to use MongoDB where the student database does not have unstructured data?
3. Does a high-level programming language offer support to the recently stabilized MongoDB?

Summary

Hence, the objective of this paper is to develop a web application by implementing MongoDB and Spring MVC framework. Also, I will use a NoSQL database instead of RDBMS as I am interested in dealing with unstructured data. MongoDB is a schema-less database and it supports documents with different structures which can be easily inserted and retrieved. By the end of the paper, we will know how to implement MongoDB with the Spring MVC framework.

Chapter II: Background and Review of Literature

Introduction

The university student portal offers various services for students. It has many modules which are used to separate the various services. Students with a valid Student ID can only login to the website. Once they log in, they can use the services offered. Different services included are finding study materials, transportation, finding accommodations, knowing about the university, selling and borrowing books from students, posting questions, and clarifying doubts about events, as well as sharing ideas.

Background Related to the Problem

Many students from different nations and various cultures come to pursue their education in a new place. As they are not familiar with the place, they need guidance to get used to their new surroundings. In such a scenario, an integrated website makes it easy to get services within a click. For example, a student can post to the university website to find out who is traveling from the airport to the university area on the day they arrive. They can get many responses either to offer them a ride or any other useful information about the transport available. Also, a student can find out a living place and roommates by posting a similar ad on the website. Interested people will then reply to their post. Students who have completed their courses can sell their books, furniture, or any accessories at reasonable prices so that it benefits both the seller and the buyer. They can also plan travel with a group of students to go to a certain place so that they can save money on transportation and travel safe as well. Students can also find out about events happening at the university. They will also become more familiar to other students by meeting each other through this website. All of the above services can be done through other social networking sites, but those sites can also be accessed by outsiders. This website, in its design, can only be accessed by students of the university.

Existing System

There is no such website which is dedicated to the students of the university where they can log in and avail themselves of the above-discussed services securely.

Disadvantages of the current system:

1. Students log into a social networking site and post their views and ideas. This can be seen by others and can mislead the students.
2. Students will post their plans of going out on the website. As this can be seen by outsiders, students can be trapped or otherwise influenced.
3. If a student posts about a transport facility, any unauthorized person can know about the schedule and easily approach it and it could be unsafe.
4. When a student wants to purchase books or accessories, the prices posted on third party websites cannot always be trustworthy.

Therefore, there are many security issues when a student uses a common social networking site.

Proposed System

In the proposed system, only students with a valid college ID can login to the website, and they can post their ideas without having fear of losing their information to outsiders. Through this, students can interact with each other and resolve many problems and make their college life potentially easier.

Advantages of the Proposed System

1. All the information that students post will be safe as no other people will have access to the site other than the students.

2. Students can easily trust the responses they receive from other students.
3. Students can find all sorts of services at one place, as this is an integrated website having various modules.

Conclusion

After careful analysis, this website has been identified to present itself with the following modules:

1. Login Module
2. Homepage (where a student can see different services).
   a. Study Materials
   b. Transport/Carpooling service
   c. Finding Accommodation/Roommates
   d. Group Discussions
   e. Conducting Events
3. Logout Module.
4. Registration Module.
5. Password Authentication

Chapter III: Methodology

Introduction

The basic model being followed is the Water Fall model. This has a sequential design, through which progress is seen in a linear way. The main approach is finding out the business requirements all at once, and then to start implementing according to the requirements.
The main stages of this methodology are (a) Requirements Gathering (Analysis), (b) Designing (Architecture), (c) Implementing (Coding), (d), Testing, and (e) Maintaining.

The first step is the gathering of requirements, which means knowing what needs to be done. The second step is planning the structure of the project; it serves as a blueprint. The third stage is the main step where the implementation begins. In this stage, the main development starts. All the necessary setup is done and the environment is created before the coding starts. For example, the operating system platform is made ready by installing required software and applications on the servers, to run the web application. The next stage is to test the whole website for any bugs. This is an important stage because, without proper testing, an application should never be deployed in the production environment. Finally, the application should be maintained until it is required.

Design of the Study

This study uses a mix of both qualitative and quantitative approaches. This website aims at providing services which are of use to the students. With this objective, it is necessary that the behavior or the tendency of students is studied and an application is developed for a useful purpose. The qualitative study depends on the investigations made on different scenarios and comes to one conclusion based on them. This website also follows the same pattern of analyzing the need to provide many kinds of services and integrate into a single web application. From a quantitative point of view, this study analyzes the number of students using social networking sites as a form of communication in a less secure way. As a result the outcome of this study is providing the services only to students on a small scale as compared to the social networking sites, which provide it to hundreds of millions of people worldwide. For this project, both approaches are equally important as one deals with the natural tendency of students sharing their information with others, and the second approach deals with the number of students being affected.

Data Collection

This is the stage where the data needed to develop this website is gathered. The natural way of collecting the data is by researching the services used by students when they arrive at a new place. Every such student would definitely look for accommodation, transport, dining, living, and stationery needs. Hence, the data has been collected keeping the needs of students as a primary source. Another common source is by observing the way students post on social networking sites. They ask about all these services before arriving at a new place. Thus, this also acts as a tool or technique to gather data.

Hardware and Software Environment

Software and hardware requirement specifications play a vital role in developing a quality and reliable product.

Hardware Requirement:

• Processor	Intel core i3 or above
• RAM	2 GB (min)
• Hard Disk	500 GB

Software Requirements:

• Operating System	Windows 7 or higher
• Programming Language	JAVA

• Web Framework	Spring MVC 4.0
• Backend Database	Mongo DB
• Front end	JSP, HTML, CSS
• Application Server	Apache Tomcat 8.0
• IDE	Eclipse, IntelliJIDEA

Chapter IV: System Design and Installations

Introduction

System design is the first step in developing a software project or product. This gives a blueprint which defines all the necessary activities to be carried. With the help of a blueprint, planning is made and tasks are easily assigned based on the qualifications of the team members. The development starts once the blueprint is evaluated.

Software Design

This project is implemented with JAVA programming language and Spring 4.0 MVC framework is used for web application. Java can be used in many IDE's such as Eclipse, Net Beans, spring tool suite, etc. I have used Eclipse Mars version. JDK 1.8 is used as this is the latest version of JAVA.

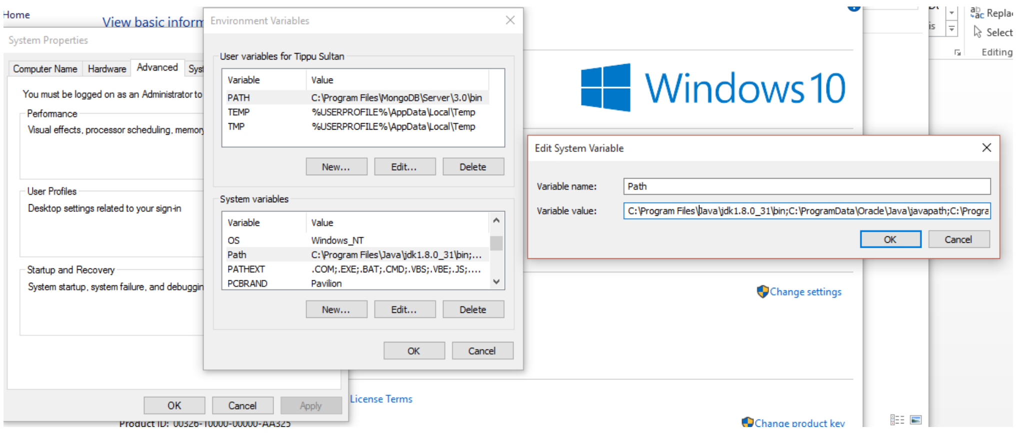
Mongo DB is used as the back end to store and retrieve the data. Being a NoSQL database, Mongo DB stores the data in the form of JSON documents. It does not have the rows and columns structure found in a relational database. The "Collection" is the term equivalent to the tables in SQL databases. Mongo DB stores the JSON documents in the collections. IntelliJ IDEA is used as an IDE (Integrated Development Environment) to implement Mongo DB. With the help of this, I can then create the database and the collections and store the data in it.

To connect Java and Mongo DB, the mongo-java-driver driver is used. Once connected, the database can then be accessed.

Installation

Java is an open source programming language and can be downloaded from the Oracle site. The first step is to download the JDK 1.8 for the Windows 64-bit operating system, as this project is implemented on 64-bit architecture. After installing Java, then go to the environment variables and set the PATH variable to the “bin” location of the JDK

Ex: PATH = C:\Program Files\Java\jdk1.8.0_31\bin;


Environment Variables

User variables for Tippu Sultan

Variable	Value
PATH	C:\Program Files\MongoDB\Server\3.0\bin
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

System variables

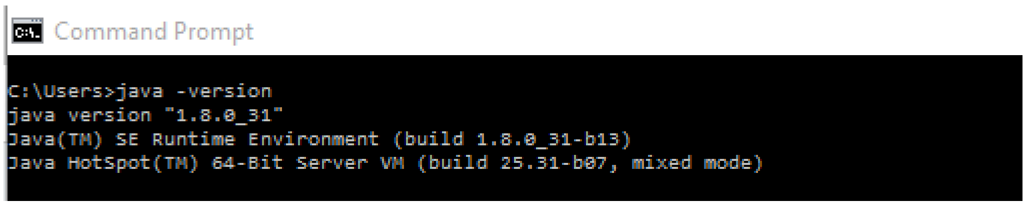
Variable	Value
OS	Windows_NT
Path	C:\Program Files\Java\jdk1.8.0_31\bin;...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....
PCBRAND	Pavilion

Edit System Variable

Variable name: Path

Variable value: C:\Program Files\Java\jdk1.8.0_31\bin;C:\ProgramData\Oracle\Java\javapath;C:\Progra

To check whether the path is set correctly, go to the command prompt and type “java –version” and it shows the JDK version installed.


Command Prompt

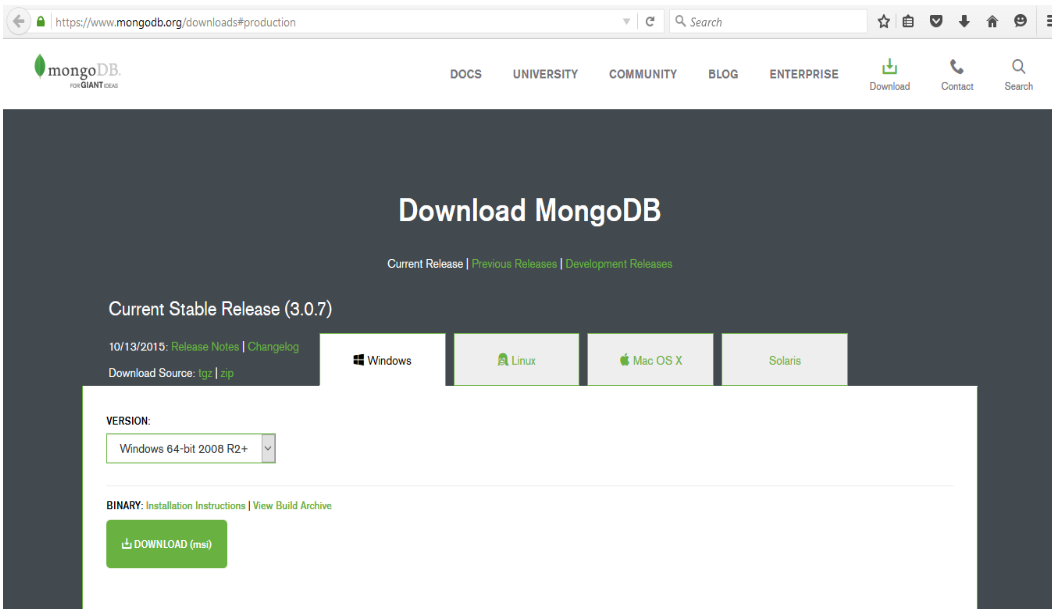
```
C:\Users>java -version
java version "1.8.0_31"
Java(TM) SE Runtime Environment (build 1.8.0_31-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.31-b07, mixed mode)
```

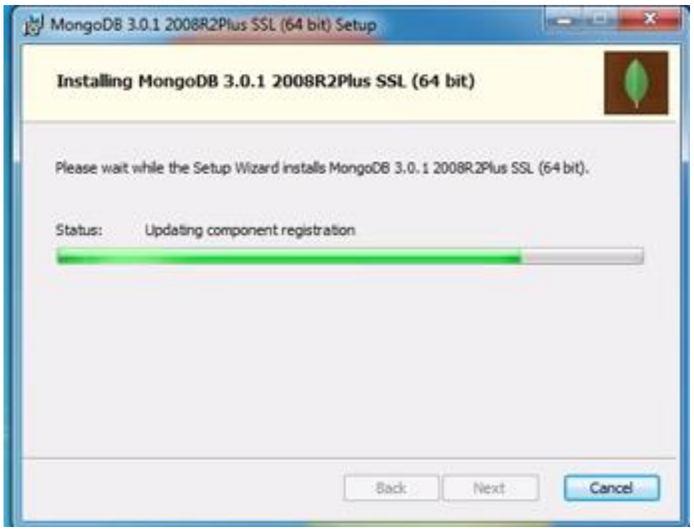
Once the java is installed, the next step is to install Eclipse IDE to develop the application. Eclipse is also an open source product which can be downloaded from https://eclipse.org/downloads/ (Eclipse Foundation, 2015). Download “Eclipse IDE for Java EE Developers,” the MARS version is used for this project as this is the latest version of Eclipse. Eclipse helps create different types of projects depending on the requirements set forth—whether it is a Java project, a dynamic web project, or a Maven project. A Maven project is used to implement this application. When a Maven project is used, it is not necessary to download the JAR files for the project, instead it is only necessary to add the dependency for the services used for the project in the "pom.xml" file, and this will automatically download the required JAR files.

With the above installation, it is then set for creating a web application. But it is still necessary to install Mongo DB to serve as the database.

MongoDB Installation

Mongo DB is also an open source product and can be downloaded from https://www.mongodb.org/downloads#production (MongoDB Inc., 2015a). Choose the operating system version and click on it to download.

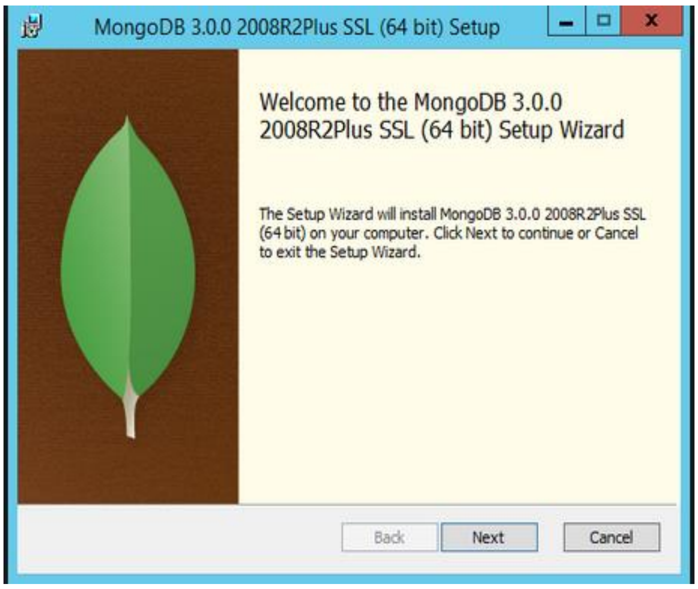

https://www.mongodb.org/downloads#production
DOCS
UNIVERSITY
COMMUNITY
BLOG
ENTERPRISE
Download
Contact
Search
Download MongoDB
Current Release | Previous Releases | Development Releases
Current Stable Release (3.0.7)
10/13/2015: Release Notes | Changelog
Download Source: tgz | zip
Windows
Linux
Mac OS X
Solaris
VERSION:
Windows 64-bit 2008 R2+
BINARY: Installation Instructions | View Build Archive
DOWNLOAD (msi)


MongoDB 3.0.1 2008R2Plus SSL (64 bit) Setup

Installing MongoDB 3.0.1 2008R2Plus SSL (64 bit)

Please wait while the Setup Wizard installs MongoDB 3.0.1 2008R2Plus SSL (64 bit).

Status: Updating component registration


MongoDB 3.0.0 2008R2Plus SSL (64 bit) Setup

Welcome to the MongoDB 3.0.0 2008R2Plus SSL (64 bit) Setup Wizard

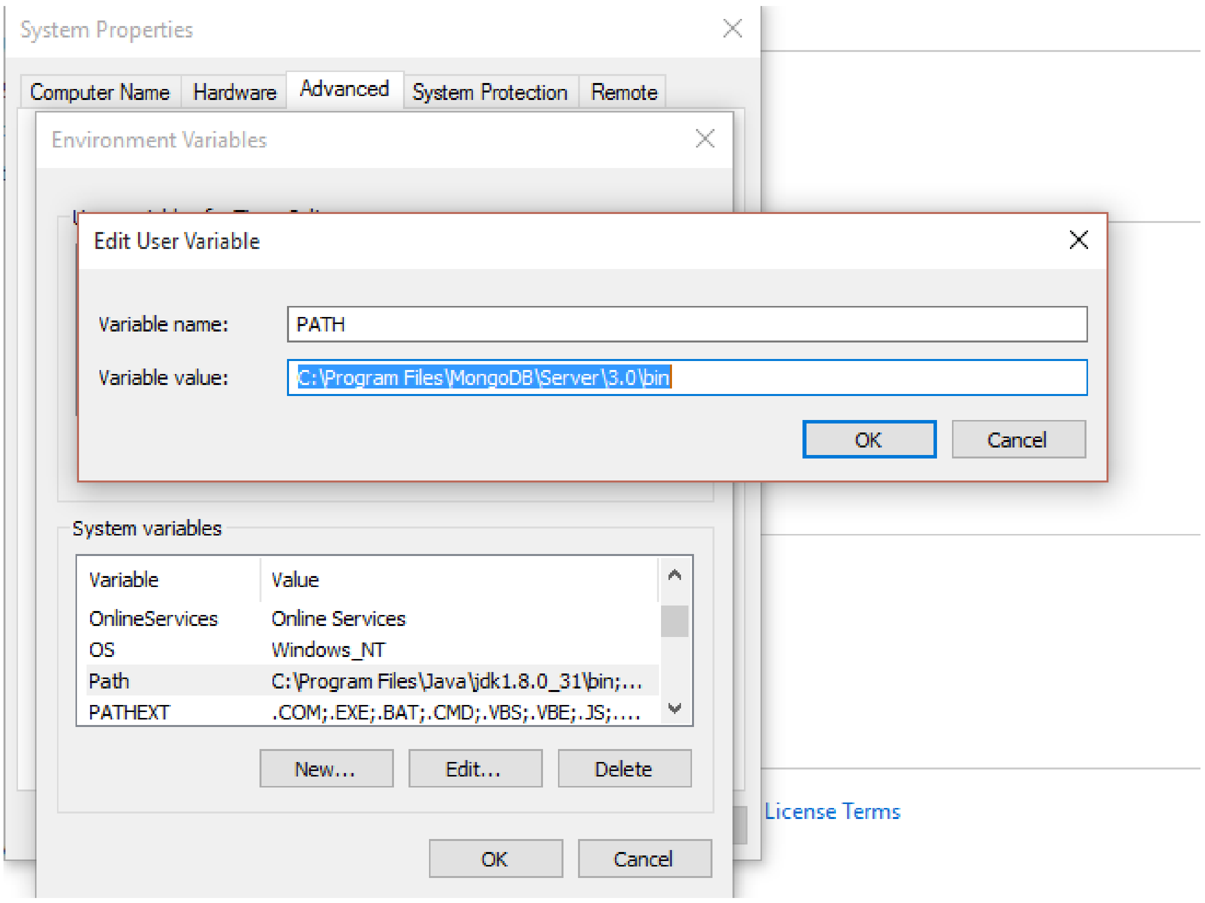
The Setup Wizard will install MongoDB 3.0.0 2008R2Plus SSL (64 bit) on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Once the installation is complete, you can then see the Mongo DB folder in C:\Program Files\MongoDB. The main location of the database is C:\Program\Files\MongoDB\Server\3.0\bin. There will be many files in the bin folder, but the main two files to be considered is mongod.exe and mongo.exe. The “mongod.exe” is the actual database, once it isstarted, it will run in the background. And “mongo.exe” is the application which is used for typing the commands, and creating the database. So “mongo.exe” is the command line interface to operate the MongoDB database.

After installing MongoDB, create a folder with a name such as “data” on the hard disk (c: drive) and create a folder “db” inside data (thenewboston, 2015).

An example path would be: C:\data\db

This is because all the data stored in MongoDB is saved at this location. After creating the above folders, it is necessary to make an entry of the “bin” location in the PATH variable just as was done for the JDK.

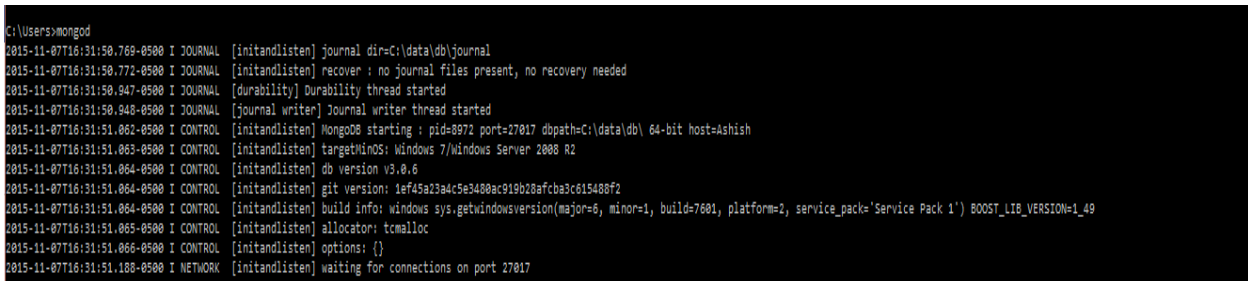


Variable name: PATH

Variable value: C:\Program Files\MongoDB\Server\3.0\bin

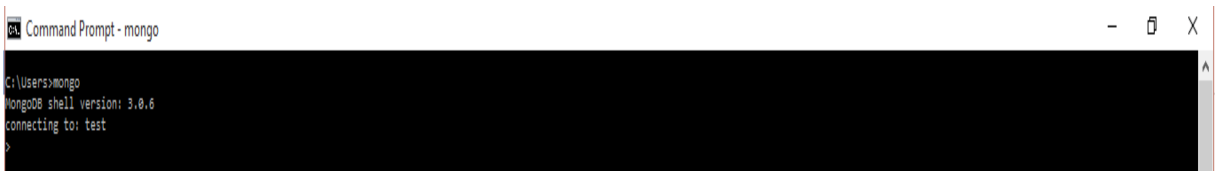
Variable	Value
OnlineServices	Online Services
OS	Windows_NT
Path	C:\Program Files\Java\jdk1.8.0_31\bin;...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....

After setting the path, then go to the command prompt and start the MongoDB database by typing “mongod.” It starts running and waits for a connection on port number 27017. This is the default port of MongoDB (thenewboston, 2015).

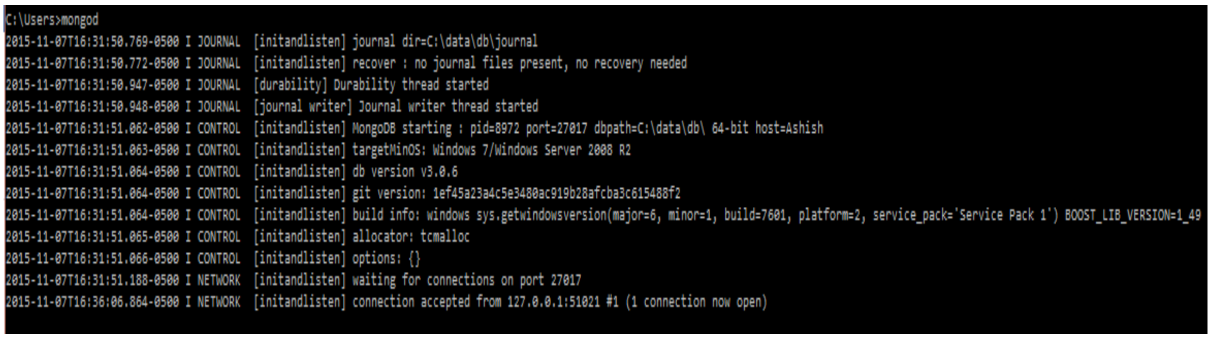


```
C:\Users>mongod
2015-11-07T16:31:50.769-0500 I JOURNAL  [initandlisten] journal dir=C:\data\db\journal
2015-11-07T16:31:50.772-0500 I JOURNAL  [initandlisten] recover : no journal files present, no recovery needed
2015-11-07T16:31:50.947-0500 I JOURNAL  [durability] Durability thread started
2015-11-07T16:31:50.948-0500 I JOURNAL  [journal writer] Journal writer thread started
2015-11-07T16:31:51.062-0500 I CONTROL  [initandlisten] MongoDB starting : pid=8972 port=27017 dbpath=C:\data\db\ 64-bit host=Ashish
2015-11-07T16:31:51.063-0500 I CONTROL  [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2015-11-07T16:31:51.064-0500 I CONTROL  [initandlisten] db version v3.0.6
2015-11-07T16:31:51.064-0500 I CONTROL  [initandlisten] git version: 1ef45a23a4c5e3480ac919b28afcba3c615488f2
2015-11-07T16:31:51.064-0500 I CONTROL  [initandlisten] build info: windows sys.getwindowsversion(major=6, minor=1, build=7601, platform=2, service_pack='Service Pack 1') BOOST_LIB_VERSION=1_49
2015-11-07T16:31:51.065-0500 I CONTROL  [initandlisten] allocator: tcmalloc
2015-11-07T16:31:51.066-0500 I CONTROL  [initandlisten] options: {}
2015-11-07T16:31:51.188-0500 I NETWORK  [initandlisten] waiting for connections on port 27017
```

As “mongod” is just a database, I cannot enter any queries at this point. To start entering queries, open a new command prompt window without closing the existing window and type mongo. This starts a new connection as shown below.

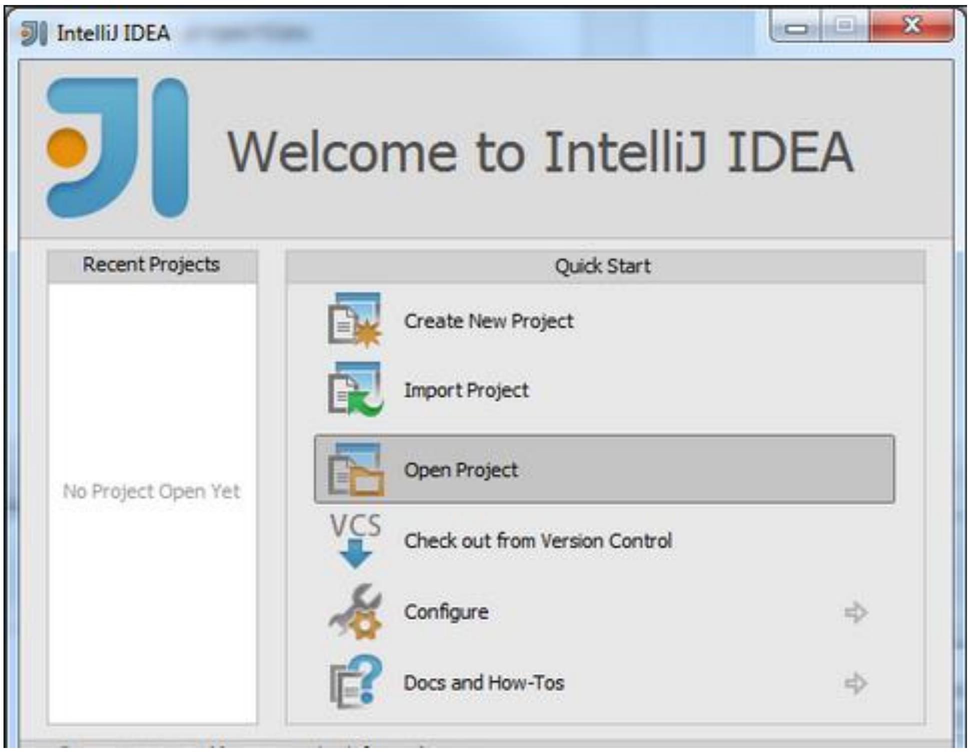


```
C:\Users>mongo
MongoDB shell version: 3.0.6
connecting to: test
>
```

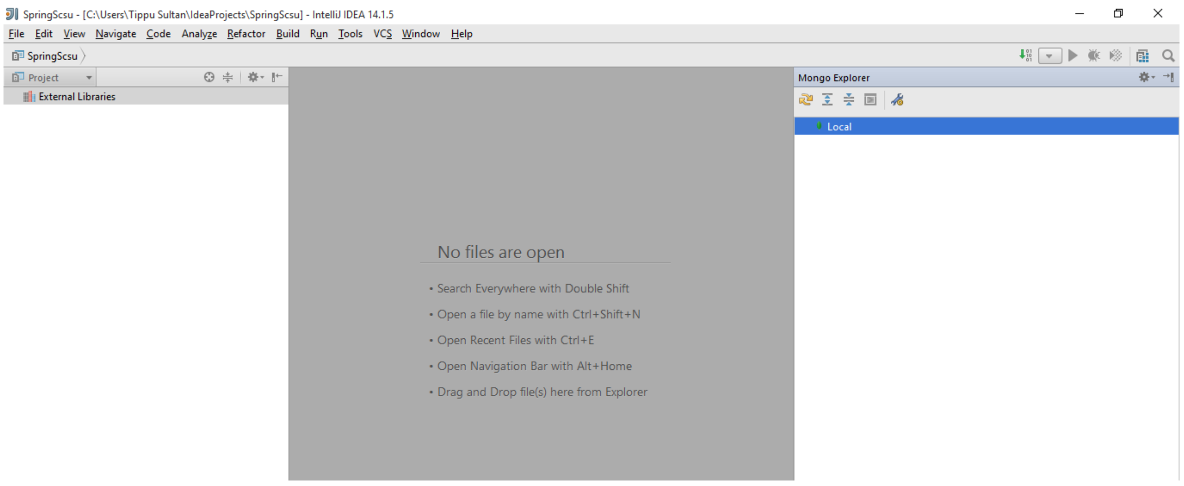


```
C:\Users>mongod
2015-11-07T16:31:50.769-0500 I JOURNAL  [initandlisten] journal dir=C:\data\db\journal
2015-11-07T16:31:50.772-0500 I JOURNAL  [initandlisten] recover : no journal files present, no recovery needed
2015-11-07T16:31:50.947-0500 I JOURNAL  [durability] Durability thread started
2015-11-07T16:31:50.948-0500 I JOURNAL  [journal writer] Journal writer thread started
2015-11-07T16:31:51.062-0500 I CONTROL  [initandlisten] MongoDB starting : pid=8972 port=27017 dbpath=C:\data\db\ 64-bit host=Ashish
2015-11-07T16:31:51.063-0500 I CONTROL  [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2015-11-07T16:31:51.064-0500 I CONTROL  [initandlisten] db version v3.0.6
2015-11-07T16:31:51.064-0500 I CONTROL  [initandlisten] git version: 1ef45a23a4c5e3480ac919b28afcba3c615488f2
2015-11-07T16:31:51.064-0500 I CONTROL  [initandlisten] build info: windows sys.getwindowsversion(major=6, minor=1, build=7601, platform=2, service_pack='Service Pack 1') BOOST_LIB_VERSION=1_49
2015-11-07T16:31:51.065-0500 I CONTROL  [initandlisten] allocator: tcmalloc
2015-11-07T16:31:51.066-0500 I CONTROL  [initandlisten] options: {}
2015-11-07T16:31:51.188-0500 I NETWORK  [initandlisten] waiting for connections on port 27017
2015-11-07T16:36:06.864-0500 I NETWORK  [initandlisten] connection accepted from 127.0.0.1:51021 #1 (1 connection now open)
```

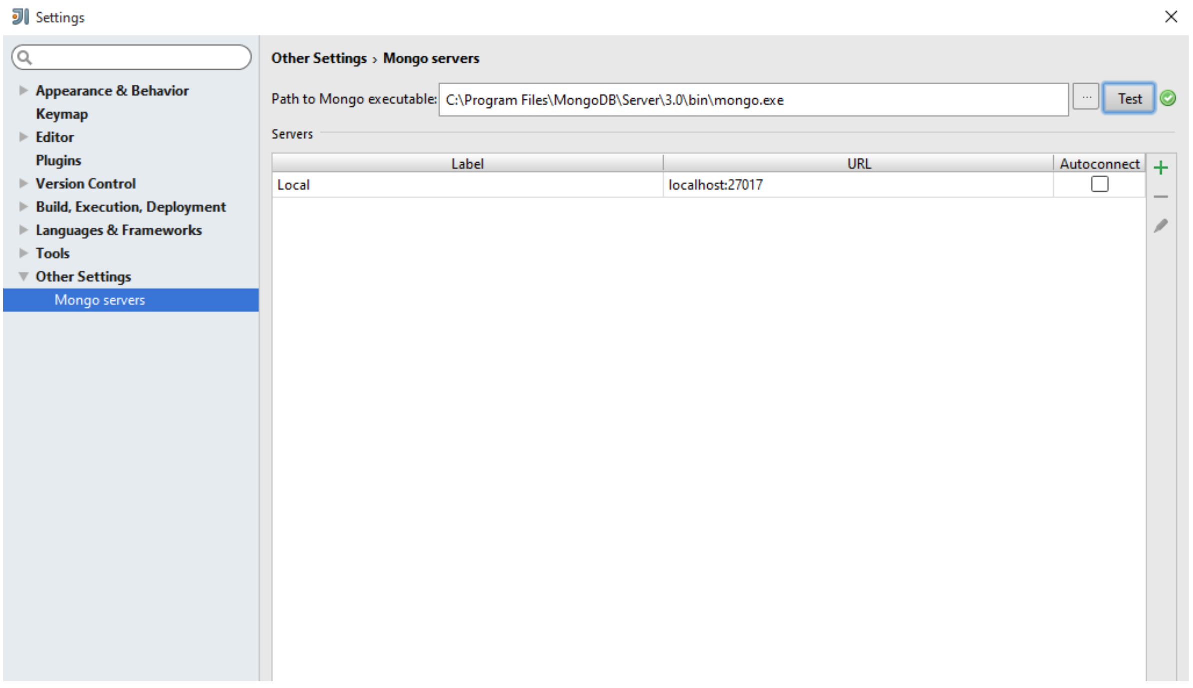
But, for implementing the queries and creating the database, I can use IntelliJ IDEA. This is a Java integrated environment to develop applications. It can also handle the database explorers through which I can operate MongoDB. It can be downloaded from https://www.jetbrains.com/idea/download/ (Jet Brains, 2015).



Community version should be downloaded as it is a free open source version. After installing IntelliJ, go to "View" and click on the "Tools" window and choose "Mongo Explorer." In the right pane, Mongo Explorer appears.


SpringScsu - [C:\Users\Tippu Sultan\IdeaProjects\SpringScsu] - IntelliJ IDEA 14.1.5
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
SpringScsu
Project
External Libraries
Mongo Explorer
Local
No files are open
• Search Everywhere with Double Shift
• Open a file by name with Ctrl+Shift+N
• Open Recent Files with Ctrl+E
• Open Navigation Bar with Alt+Home
• Drag and Drop file(s) here from Explorer

Then click on the "Settings" icon, a window opens and the path of "monog.exe" should be entered to connect to the MongoDB. Click on the "Test" button to check if the connection is successful. Then click "OK."



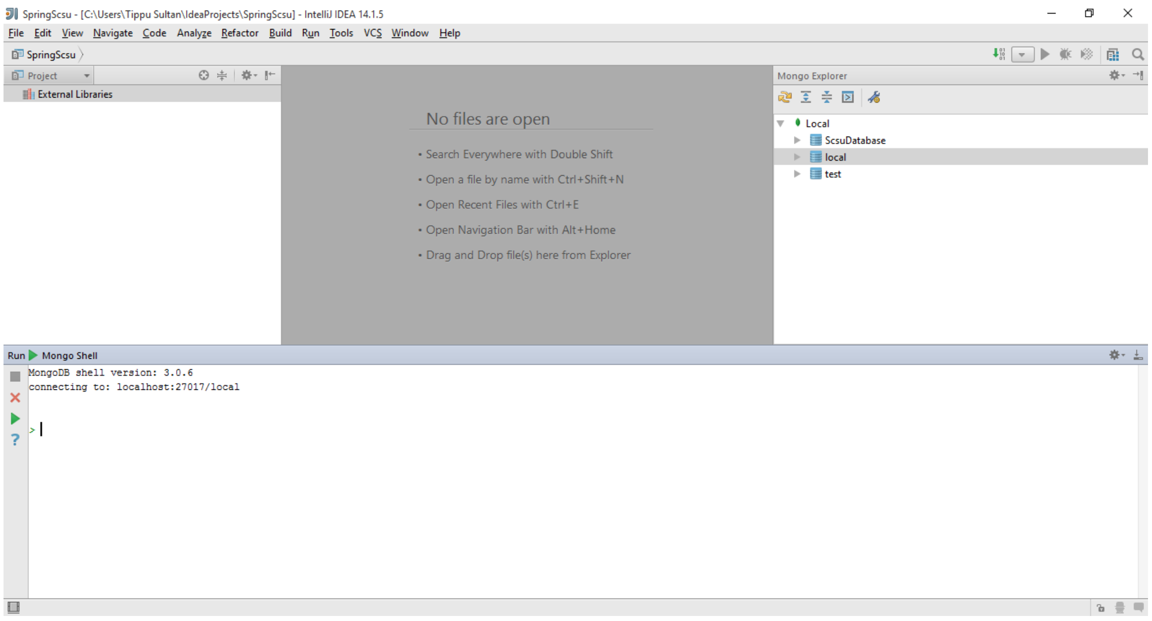
Path to Mongo executable: C:\Program Files\MongoDB\Server\3.0\bin\mongo.exe

Servers

Label	URL	Autoconnect
Local	localhost:27017	☐

Now, on the button “Local,” right click and connect to the server. Once connected it shows the databases present on the server. Click on the “Mongo shell” icon to open the console to enter queries.

A console appears at the bottom of the window and that is the place where queries can be entered.


SpringScsu - [C:\Users\Tippu Sultan\IdeaProjects\SpringScsu] - IntelliJ IDEA 14.1.5
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
SpringScsu
Project
External Libraries
Mongo Explorer
Local
ScsuDatabase
local
test
Run
Mongo Shell

No files are open

- Search Everywhere with Double Shift
- Open a file by name with Ctrl+Shift+N
- Open Recent Files with Ctrl+E
- Open Navigation Bar with Alt+Home
- Drag and Drop file(s) here from Explorer

```
MongoDB shell version: 3.0.6
connecting to: localhost:27017/local

> |
```

Now I am all set to operate MongoDB.

Mongo DB

Mongo DB is implemented in C++. It is an open source and document based database. It provides good performance and offers a high degree of data availability. It can also be easily scalable to allow for further growth. MongoDB runs on the concept of “Documents” and “Collections.” A group of documents is stored in a collection, and a group of collections is stored in a database (Tutorials Point, n.d.a.).

Technology. The “Collection,” or a group of “Documents,” is equivalent to a table in an RDBMS. A collection is present in a single database and it does not have a particular schema. MongoDB is a schema-less database. This means, the documents entered may not have a similar structure. For example, if the student1 record has attributes "First Name" and "Last Name," the student2 can have attributes "First Name," "Last Name" as well as "Age" in their record.

The relationship between MongoDB and RDBMS terminology is shown in Table 1 (Tutorials Point, n.d.a).

Table 1

Relationship between MongoDB and RDBMS

MongoDB	**RDBMS**
Database	Database
Collection	Table
Document	Tuple/Row
Field	Column
Embedded Documents	Table Join
Primary Key (Default key _id provided by MongoDB itself)	Primary Key

Advantages of MongoDB

1. As MongoDB is a schema-less database, one collection can hold various types of documents, each having its own structure.
2. There are no complex joins as in RDBMS.
3. Easy to scale.
4. Enables faster access of data by storing the data using the internal memory.
5. Use document-based query language that supports dynamic queries.
6. Tuning.

Sharding. Sharding is a database having a large volume of data can challenge the efficiency of a single server. Storage capacity will not be enough to handle so much data, and the increase in the number of queries will bring a server down. This will also affect the performance of RAM (MongoDB, Inc. 2015b). There are two approaches to address this problem—Vertical Scaling and Sharding.

Vertical scaling. This is just adding on more CPUs and storage to increase the capacity. This has some disadvantages as adding CPUs and RAM will be expensive compared to smaller systems (MongoDB, Inc. 2015b.)

Sharding.This is also called as horizontal scaling. This divides the data set and distributes it over different servers, also called "shards." Now, each shard will act as an independent database, but collectively all the shards are a single logical database (MongoDB, Inc. 2015b).

Advantages of sharding. Advantages of sharding include:

1. As the cluster grows, each shard processes fewer operations.
2. The number of operations each shard handles can be reduced.
3. The amount of data to be stored on each server is reduced.
4. As the cluster grows, each shard stores less data.
5. Capacity and throughput of the cluster are increased horizontally.

Design and Architecture

The design and architecture of the application defines the main structure of implementing MongoDB with Spring MVC. The two main designs for this project are MongoDB Database Design and Java Code Design.

Let us first discuss the MongoDB database design and how different it is from using an RDBMS.

Database design. As already discussed, a **Collection** refers to a table in RDBMS. So a collection is a place where the data is stored. As in an RDBMS, data is stored in the form of rows and columns, but in MongoDB, data is stored in the form of documents.

There are two ways of designing the RDBMS. Let us make a design of **RDBMS** based on the requirements of the Web Application.

RDBMS Design 1

1. There should be a student table to store the details of the students.

   For this requirement, it is necessary to create a student table with the following columns.

   a. First Name (User Name)

   b. Last Name

   c. Student ID

   d. Password

2. As there are three different services such as "Accommodation Module," "Car Pooling Module" and "Knowledge sharing Module," it is necessary to create three tables for the same data to be maintained separately. Therefore, it is necessary to create the tables below:

   a. Accommodation Table

   b. Car Pooling Table

   c. Knowledge Table

Each of these tables will have similar columns to fit the posts and comments related to the individual postings.

For Example, the Accommodation Table will have the columns shown in Table 2.

Table 2

RDBMS Design 1—Accommodation Table

Column Name	Data_type
accommodation_service_type	Varchar(30)
accommodation_post_id	Double
accommodation_post_title	Varchar(max)
accommodation_post_author	Varchar(30)
accommodation_comments	Varchar(max)
accommodation_comments_author	Varchar(30)

Similarly, for the Car Pooling Table, the columns would be as shown in Table 3.

Table 3

RDBMS Design 1—Carpooling Table

Column Name	Data_type
carpooling_service_type	Varchar(30)
carpooling_post_id	Double
carpooling_post_title	Varchar(max)
carpooling_post_author	Varchar(30)
carpooling_comments	Varchar(max)
carpooling_comments_author	Varchar(30)

Similarly, for the Knowledge Table, the columns would be as shown in Table 4.

Table 4

RDBMS Design 1—Knowledge Table

Column Name	Data_type
knowledge_service_type	Varchar(30)
knowledge_post_id	Double
knowledge_post_title	Varchar(max)
knowledge_post_author	Varchar(30)
knowledge_comments	Varchar(max)
knowledge_comments_author	Varchar(30)

The “service_type” columns define the exact service provided in the module, below are different services available in the three modules.

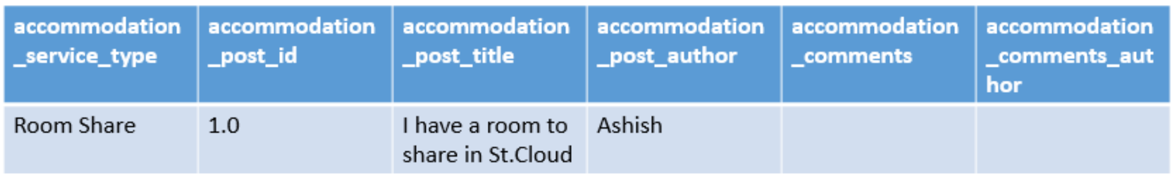
1. Accommodation Module
    a. “Need a room to share“
    b. “Have a room to rent”
    c. “Have a room to share”
2. Car Pooling Module
    a. “Need a Ride”
    b. “Have a Car to rent”
3. Knowledge Sharing Module
    a. “Study Materials”
    b. “Job References”
    c. “Miscellaneous”

As per the requirement, a student should be able to comment on any post. This means each post can have any number of comments under it.

Now the challenge is, with the above-defined table structure, how can I store multiple comments for a particular post? Can I increase the "comments" columns as comment1, comment2 . . . etc.? This cannot be done as I don't know how many comments will be created for individual post. Also, it is not a good practice to increase the columns dynamically. Even if I increase for all the rows, some posts may have fewer comments than others, leaving the columns blank and making the database less efficient. Also, increasing the number of columns will create a greater load on the database. As a result the "JDBC Result Set" should be iterated in a loop to retrieve the increasing number of columns.

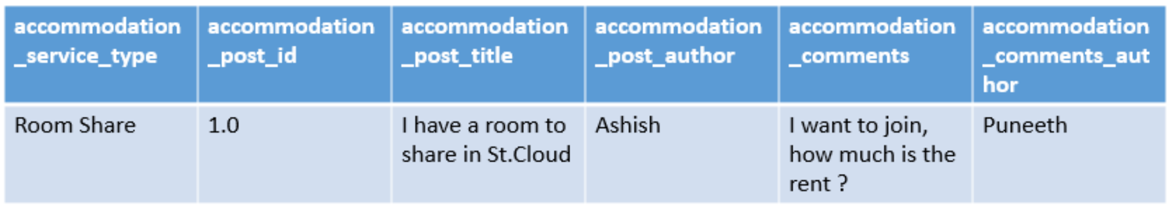
To address this issue, it is necessary to insert a new row with the new comment and "comment_author" and leaving the rest of the columns the same.

For example, if a student posts that he needs a room to share, this will be stored in the Accommodation Table as follows.



accommodation_service_type	accommodation_post_id	accommodation_post_title	accommodation_post_author	accommodation_comments	accommodation_comments_author
Room Share	1.0	I have a room to share in St.Cloud	Ashish		

Now, if someone wants to comment on the above post, I can insert it in the same row as the columns are available.



accommodation_service_type	accommodation_post_id	accommodation_post_title	accommodation_post_author	accommodation_comments	accommodation_comments_author
Room Share	1.0	I have a room to share in St.Cloud	Ashish	I want to join, how much is the rent ?	Puneeth

If there are a few more comments to the same post, then the comments should be added in a new row as shown below.

accommodation_service_type	accommodation_post_id	accommodation_post_title	accommodation_post_author	accommodation_comments	accommodation_comments_author
Room Share	1.0	I have a room to share in St.Cloud	Ashish	I want to join, how much is the rent ?	Puneeth
Room Share	1.0	I have a room to share in St.Cloud	Ashish	I am interested too, will join ASAP	Sandeep
Room Share	1.0	I have a room to share in St.Cloud	Ashish	How far is it from university.	Hari

Therefore, there is redundancy of data in the first four columns. As per this design, to accommodate new comments, the common data like “**accommodation_service_type**,” “**accommodation_post_id**,” “**accommodation_post_title**” and “**accommodation_post_author**” are repeated. This will result in larger storage requirements and brings down the efficiency of the database. While retrieving the data through JDBC with the above design, there will be a lot of SQL queries and Java “DAO”(Data Access Object) objects to be handled, which would create a cumbersome coding problem.

Disadvantages of Design 1. Disadvantages of Design 1 inclue:

1. Redundancy of data, as a new row is inserted for a new comment in the same post.
2. Different comments are inserted in the same column, hence requiring a timestamp to keep the comments in order.
3. As the number of comments increases, more rows will be inserted, which will increase the size of the database.

RDBMS Design 2

There is another way of designing the RDBMS. In this way, I can separate the comments from the Post table and join the two tables based on “Post_id.”

Following is the example for the Accommodation Table. Two tables are required in this case—Accommodation_post_table and Accommodation_comments_table.

Accommodation_post_table has the following columns as shown in Table 5.

Table 5

RDBMS Design 2—Accommodation Post Table

accommodation_service_type	Varchar(30)
accommodation_post_id	Double
accommodation_post_title	Varchar(max)
accommodation_post_author	Varchar(30)

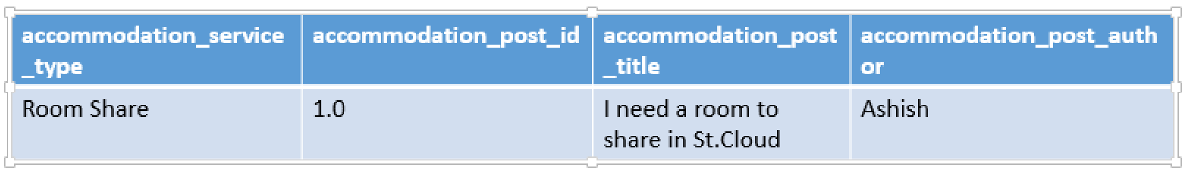
Accommodation_comments_table has the following columns as shown in Table 6.

Table 6

RDBMS Design 2—Accommodation Comments Table

accommodation_post_id	Double
a**ccomm**odation_comments	**Varchar(max)**
a**ccomm**odation_comments_author	**Varchar(30)**

With the above approach, the comments within the same post can be saved in the "comments" table with the same "post id" as in the "Accommodation post" table. It would look like the example below:



accommodation_service_type	accommodation_post_id	accommodation_post_title	accommodation_post_author
Room Share	1.0	I need a room to share in St.Cloud	Ashish

And the comments for this post can be then inserted into the Comments Table, as seen in the following example. This would reduce the redundancy of the same columns as was seen in Design 1.

accommodation_post_id	accommodation_comments	accommodation_comments_author
1.0	I want to join, how much is the rent ?	Puneeth
1.0	I am interested too, will join ASAP	Sandeep
1.0	How far is it from university.	Hari

With this design, it is possible to reduce the redundancy which occurred in Design 1. Also with this design, it is necessary to create two tables for each module which impacts the Java code (as more complex queries would need to be written). It is necessary to create two Model objects for each module, one for the "post" table and another for the "comments" table. Also, the Data Access Object will be complex, as I need to retrieve the data from two different tables by joining on the "post id." Hence, it would be necessary to write many queries for each service, which would create a lot of work for the programmer.

Disadvantages of Design 2. Disadvantages of Design 2 include:

1. Two tables for each service should be created, one for the "Posts" Table and another for the "Comments" Table. Hence there will be six tables for three services.
2. An increase in the number of tables will increase the model objects in Java.
3. The "Post" Table and the "Comments" Table should be joined based on the "post id" in the "Post" Table. This makes more effort in querying and retrieving the data from the database.

Hence, the two designs provided by an RDBMS have a few disadvantages, which needs to be addressed. Let us then see the design in the case of MongoDB.

MongoDB Design

With MongoDB, I can have a unique design which addresses all the requirements for this project with a minimum number of “collections” (referred to as tables in an RDBMS), less model objects in JAVA, and simpler queries to fetch the data from the collection.

As discussed earlier, MongoDB works on “collections” and “documents.” Documents are the actual data which reside in the collection. Each document can have its own structure. For example, if there are two students whose data should be stored and Student 1 has enrolled in sports whereas Student 2 is not enrolled. In this case, it is possible to have the documents with a different structure as seen below:

For Student 1:

```
{   FirstName: 'Ashish',
  LastName: 'Kastury',
  Major: 'Information Assurance’,
  Sports: 'Cricket'
 }
```

For Student 2:

```
{   FirstName: 'Sandeep',
  LastName: 'Singh',
  Major: 'Information Assurance'
 }
```

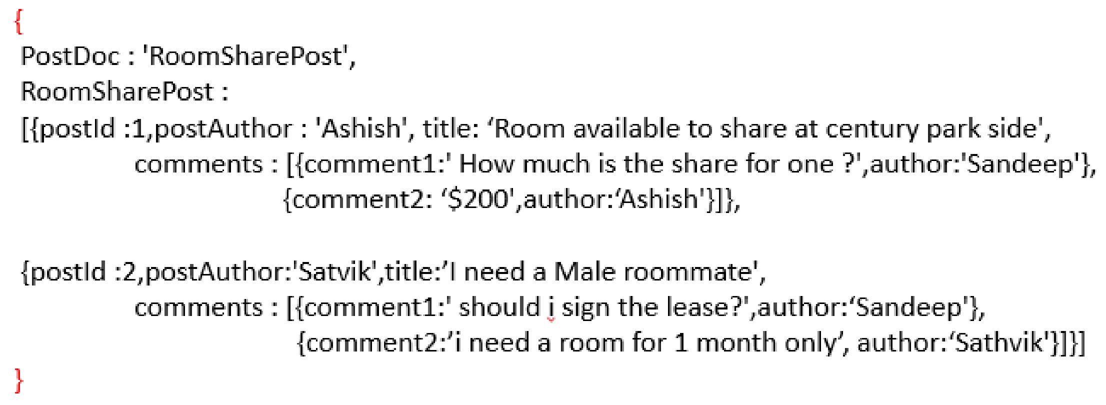
Hence two students’ records in the same collection have a different structure. This kind of flexibility will increase the efficiency of the database and reduces the redundancy. With the help of this flexibility, it is possible to build the design and to implement the requirements of this project efficiently.

As I have three different modules and each module have different services, I can keep documents for different services in the same “collection.” For example, as discussed earlier the “accommodation module” has three services: “Need a room to share,” “Have a room to rent.” and “ Have a room to share.” So, it is possible to create three documents for these three services and store all the comments within this document. Each document should have two main fields:

1. PostDoc: This field tells us the type of posts (or service) this document holds.
2. *Post: This field will store the actual posts and comments in an embedded document. This field is a type of array which increases dynamically when new comments are pushed to it.

The name of this field resembles the type of service provided.

Let us see the document for the “Have a room to share“ service.



```
{
 PostDoc : 'RoomSharePost',
 RoomSharePost :
 [{postId :1,postAuthor : 'Ashish', title: ‘Room available to share at century park side',
            comments : [{comment1:' How much is the share for one ?',author:'Sandeep'},
                        {comment2: ‘$200',author:‘Ashish'}]},

 {postId :2,postAuthor:'Satvik',title:’I need a Male roommate',
            comments : [{comment1:' should i sign the lease?',author:‘Sandeep'},
                         {comment2:’i need a room for 1 month only’, author:‘Sathvik'}]}]
}
```

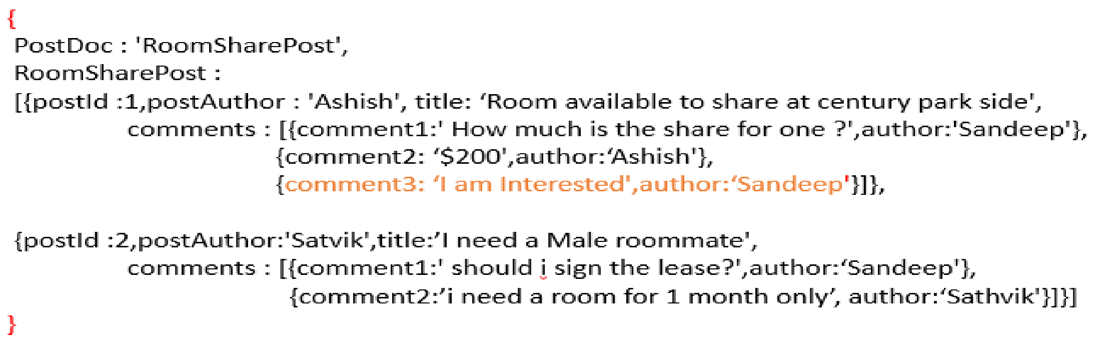
In the above document, the PostDoc field has the value as “RoomSharePost” as it is the type of service available. And the second field also resembles the same and stores all the posts and comments related to this service. The “RoomSharePost” field is an array type and stores embedded documents inside it in the form of arrays. In the above document, “RoomSharePost” has two embedded documents, one with “postId” as the first and the other with “postId” as the second. As this is an array, when someone enters a new post, it will be stored as “postId” as the third. Therefore, any new posts will be stored with its own “postId” making each post unique.

On the other hand, for each “postId,” there is a “post title” field which holds the actual data which is being posted. Also, there is a “postAuthor” field which stores the name of the student who actually made the post. There is a “comments” field associated to this “postId” which stores the comments commented on this particular post. This field has an embedded document of comments stored again as an array. As shown above, there are two comments in the postId1. If someone enters a new comment, it will be stored as “comment3.” The comments field has two fields, one to store the actual comment, which is denoted by the position of the comment such as comment1, comment2 . . . etc. The other field is the author, which stores the name of the student who commented on the post. The comments field is also an array of embedded documents having fields as “comment#” and “author.”

To summarize, “RoomSharePost” field has three inner fields:

1. PostId
2. PostAuthor
3. Comments—and this field has two inner fields: Comment# and Author.

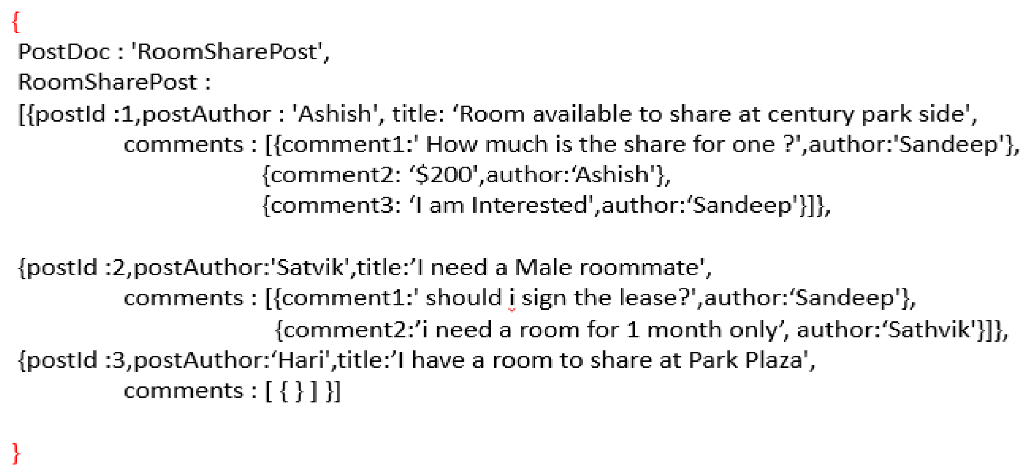
If someone comments to the postId 1, it will store as below.



```
{
 PostDoc : 'RoomSharePost',
 RoomSharePost :
 [{postId :1,postAuthor : 'Ashish', title: ‘Room available to share at century park side',
           comments : [{comment1:' How much is the share for one ?',author:'Sandeep'},
                       {comment2: ‘$200',author:‘Ashish'},
                       {comment3: ‘I am Interested',author:‘Sandeep'}]},

 {postId :2,postAuthor:'Satvik',title:’I need a Male roommate',
           comments : [{comment1:' should i sign the lease?',author:‘Sandeep'},
                        {comment2:’i need a room for 1 month only’, author:‘Sathvik'}]}]
}
```

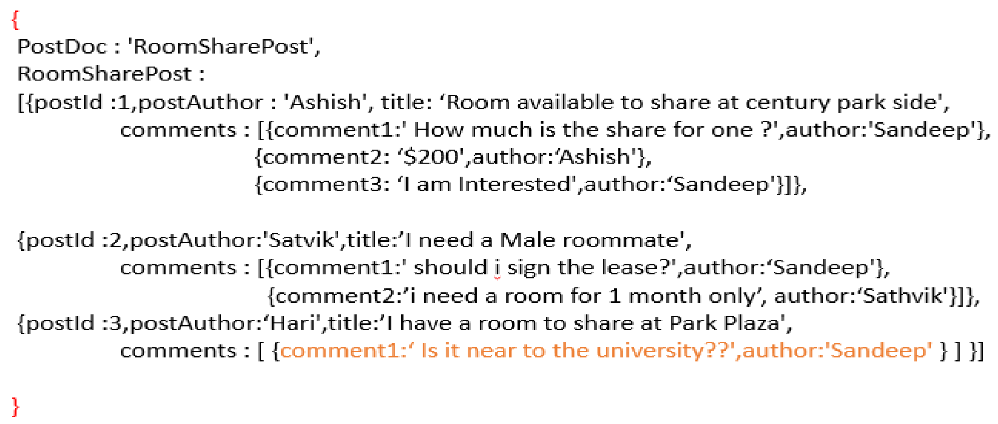
In the same way, “postId 2” also has the same fields and embedded documents in it. And if someone creates a new post it will be appended to the “RoomSharePost” field below the “postId 2.” This is because the field is also an array type, as stated earlier, and stores all the data in the form of an array.

Therefore, if someone creates a new post, it will look like the following example:



```
{
PostDoc : 'RoomSharePost',
RoomSharePost :
[{postId :1,postAuthor : 'Ashish', title: ‘Room available to share at century park side',
            comments : [{comment1:' How much is the share for one ?',author:'Sandeep'},
                        {comment2: ‘$200',author:‘Ashish'},
                        {comment3: ‘I am Interested',author:‘Sandeep'}]},

{postId :2,postAuthor:'Satvik',title:’I need a Male roommate',
            comments : [{comment1:' should i sign the lease?',author:‘Sandeep'},
                         {comment2:’i need a room for 1 month only’, author:‘Sathvik'}]},
{postId :3,postAuthor:‘Hari',title:’I have a room to share at Park Plaza',
            comments : [ { } ] }]

}
```

As can be seen, there is a new post created by “Hari,” which has a postId of “3,” and the comments field is empty as no one has commented on it yet. Therefore, the comments field is created with no inner fields. As the comments field is an array type of embedded documents, whenever there is a comment, it is pushed to the “comments” array, with fields as “comment#” and “author” as shown below.



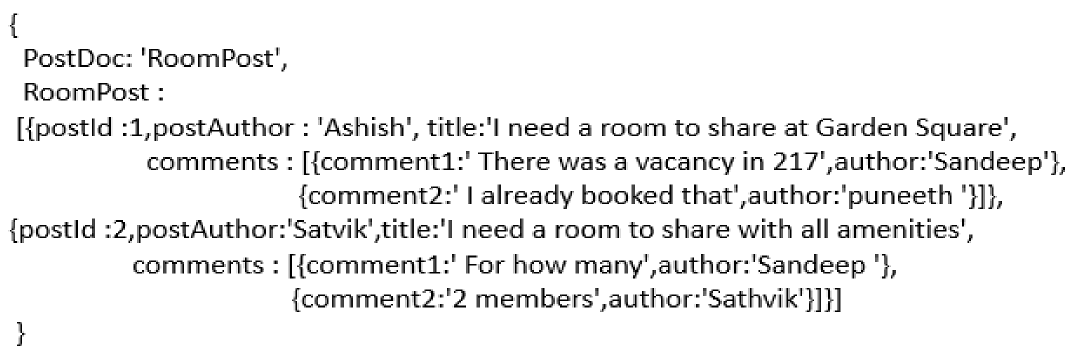
```
{
PostDoc : 'RoomSharePost',
RoomSharePost :
[{postId :1,postAuthor : 'Ashish', title: ‘Room available to share at century park side',
            comments : [{comment1:' How much is the share for one ?',author:'Sandeep'},
                        {comment2: ‘$200',author:‘Ashish'},
                        {comment3: ‘I am Interested',author:‘Sandeep'}]},

{postId :2,postAuthor:'Satvik',title:’I need a Male roommate',
            comments : [{comment1:' should i sign the lease?',author:‘Sandeep'},
                         {comment2:’i need a room for 1 month only’, author:‘Sathvik'}]},
{postId :3,postAuthor:‘Hari',title:’I have a room to share at Park Plaza',
            comments : [ {comment1:‘ Is it near to the university??',author:'Sandeep' } ] }]

}
```

If there is a second comment to the same post, a “comment2” field will be created and pushed into the “comments” array. There will then be an author field associated with the “comments2” field. In this way, the posts and comments will be appended one after the other in the document.

Similarly, it is possible to create other documents for the remaining services offered on the web portal. Therefore, there will be a total of eight documents in the collection. All of these documents would be have the same default fields as discussed above, but may vary in the number of posts and the number of comments created within them. On another note, the “PostDoc” field will have different values because it depends on the types of services it is associated with.

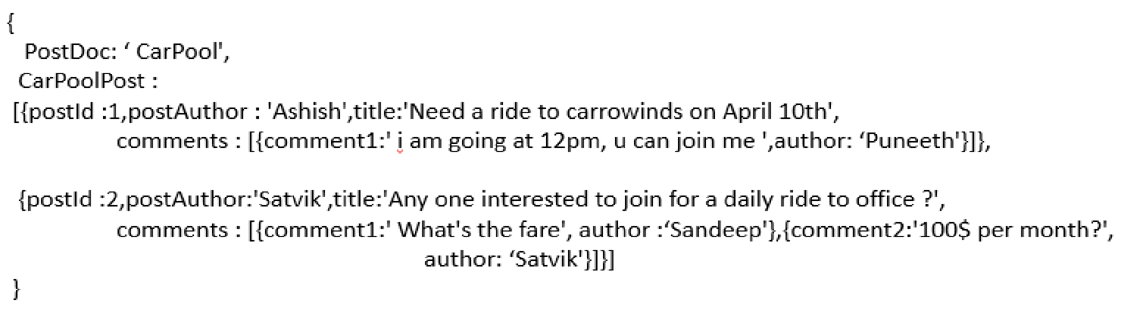
Following are the documents for different services available with some sample posts and comments.

Service type: “Need a room to share”



```
{
  PostDoc: 'RoomPost',
  RoomPost :
 [{postId :1,postAuthor : 'Ashish', title:'I need a room to share at Garden Square',
              comments : [{comment1:' There was a vacancy in 217',author:'Sandeep'},
                          {comment2:' I already booked that',author:'puneeth '}]},
{postId :2,postAuthor:'Satvik',title:'I need a room to share with all amenities',
             comments : [{comment1:' For how many',author:'Sandeep '},
                         {comment2:'2 members',author:'Sathvik'}]}]
 }
```

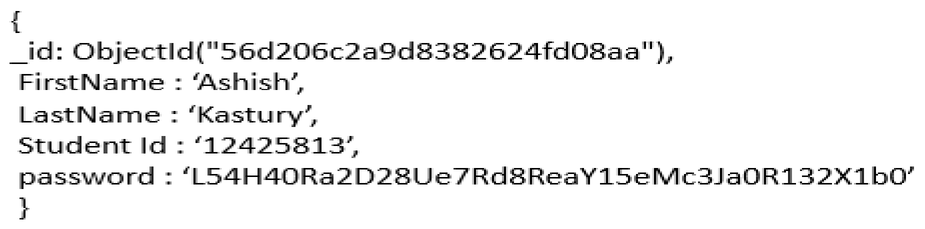
Service Type: “Need a Ride”



```
{
  PostDoc: ‘ CarPool',
 CarPoolPost :
 [{postId :1,postAuthor : 'Ashish',title:'Need a ride to carrowinds on April 10th',
            comments : [{comment1:' i am going at 12pm, u can join me ',author: ‘Puneeth'}]},

 {postId :2,postAuthor:'Satvik',title:'Any one interested to join for a daily ride to office ?',
            comments : [{comment1:' What's the fare', author :‘Sandeep'},{comment2:'100$ per month?',
                                                author: ‘Satvik'}]}]
 }
```

Similarly, for other posts, there will be respective documents for the type of service.

As I have the flexibility of creating the documents with different structure in the collection, I can also create the student details in the same collection. The following document structure stores the basic information of the student when one registers on the website.



```
{
 _id: ObjectId("56d206c2a9d8382624fd08aa"),
  FirstName : 'Ashish',
  LastName : 'Kastury',
  Student Id : '12425813',
  password : 'L54H40Ra2D28Ue7Rd8ReaY15eMc3Ja0R132X1b0'
  }
```

Hence, the above document can also be stored in the same collection where "posts" are stored. This design will make it easy to query the data from Java, as the connection will be made only to one specific "collection."

Summarizing the above design, there will be one collection (Student) in which all the documents related to the services and the details of the students can be maintained with different fields as per the requirement.

Advantages of MongoDB Design. Advantages of the MongoDB Design include:

1. Only one collection is sufficient for all the data. Thus, querying from Java will be easy as the connection will be made only to one collection.
2. There will be eight different documents holding the data (Posts and Comments) as there are only eight services available on the website.
3. The same collection can store the student details, which will have a different structure compared to the "Post" documents.
4. There will not be any "Joins" in the SQL query as there is only one collection, thereby reducing the Java code that is required.

5. There is no need to have a primary/foreign key concept as all the fields (Posts and associated comments) will reside in the same document.

How MongoDB Works

As discussed earlier, MongoDB works on the concept of “Collections” and “Documents.” Below are a few examples showing how to perform the CRUD (Create, Read, Update, Delete) operations such as creating, inserting, updating, or deleting the data.

1. **Creating**
   a. To create a database, just type “**use databaseName**.” MongoDB checks if the database exists or not, if it doesn’t exist, it will create one.

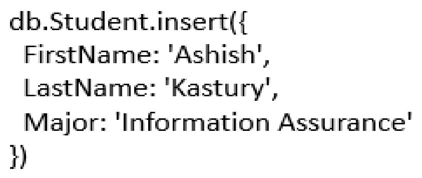
      Ex: To create the database for this project: “**use ScsuDatabase”**

   b. To create a collection, **db.createCollection(“CollectionName”)**.

      Ex: To create the collection for this project, below is the syntax.

      db.createCollection(“Student”).

2. **Inserting a document:** (Marchioni, 2015)



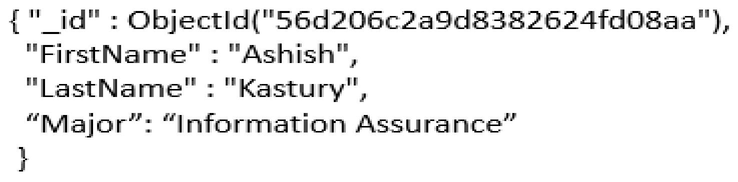
```
db.Student.insert({
  FirstName: 'Ashish',
  LastName: 'Kastury',
  Major: 'Information Assurance'
})
```

3. **Querying a document:** (Marchioni, 2015)

   To find documents in the collection.

   db.student.find ()

   The response would be as follows.



```
{ "_id" : ObjectId("56d206c2a9d8382624fd08aa"),
  "FirstName" : "Ashish",
  "LastName" : "Kastury",
  “Major”: “Information Assurance”
 }
```

4. **To find a document with a condition**.

   This is similar to the “Where” condition used in an RDBMS.

   db.student.find({“FirstName”:”Ashish”})

   Returns the above document.

5. **Updating a document** (Marchioni, 2015)

   db.student.update({“FirstName”:”Ashish”}, {$set: {“Age” : 26}})

   The above statement finds the document having, “FirstName” as “Ashish” and sets the “Age” field as 26. If the field is not available, it creates the “Age” field and sets the value to 26. This is a very useful concept of MongoDB as it is not necessary to alter or recreate the document with new fields.

6. **Deleting a Document** (Marchioni, 2015)

   db.student.remove(), this code removes all the documents from the collection. This is the same as the “TRUNCATE” command in SQL. But, if it is necessary to remove a particular document, then it may be necessary to use the “Where” clause as illustrated below.

   db.student.remove({“Age”: 26})

   The above statement removes the document having the “Age” field of 26.

7. **Inserting Arrays** (Marchioni, 2015)

   This is the most important concept which is used in this project. This will insert the data created in the form of an array.

   For example, if I want to insert comments in an array.

   db.student.insert({“Courses”: [“Mathematics,” “Physics,” “Chemistry”]})

   The above statement will insert three courses (Mathematics, Physics and Chemistry) in the “courses” field in the form of an array. When the comments field is queried, it will give three values. Hence, it is possible to iterate these values in Java and to display the comments.

JAVA Code Design

Another main design is Java code design, through which data from MongoDB is queried and displayed on the website. I am using the Spring MVC Annotation based framework. Hence, I can use the annotations @Bean, @Controller, and @RequestMapping. to reduce the XML configuration code. A Controller class is the one which interacts with the URL pattern to get the parameters from the URL and send the data from this class to the "**JSP**" file. The JSP page will then display the data passed from the controller class.

DAO configuration. This class is responsible for connecting to the database. Hence, this is the basic requirement of the project to make the connection to the "MongoDB" database. This is then named as "BaseDAO."

Transfer object class. There is a transfer object which denotes the fields in the MongoDB collection. This helps in storing the data fetched from the database, and sends the data posted on the website to the respective documents into the database. The flexibility with MongoDB is such that, there only needs to be one Transfer Object, which can then be used by all three of the controller classes associated with the respective modules and services. Hence, this will reduce the effort of designing different transfer object classes for the programmer.

Controller classes. As there are three different modules such as the "Accommodation Module," "Carpooling Module," and "Knowledge sharing Module," it is necessary to have three different controller classes to handle the data and process it to the website. Hence, below are the three controllers are (a) AccommodationController, (b) CarPoolingController, and (c) KnowledgeController.

There are two other controller classes which are used in validating the login and displaying the welcome page to the student who has successfully logged in. These are HelloController and WelcomeController.

The **AccommodationController** deals with the data used in the “Accommodation module.” As there are three services in this module such as “Need a room to share,”“ Have a room to rent,” and “ Have a room to share,” this class will interact with the similar documents present in MongoDB in the student collection. There are four methods for each of the three services available. Following is the functioning of these methods for the service “Need a room to share.”

1. **retriveData():** This method gets the data from the “student” collection. After getting the data it sets it to the “PostTo” object and passes it to the respective “JSP” page. For example, this method sends data to the “Default Post” page as this is referred to the “Need a room to share” service. All the posts and comments related to this service are then displayed on the “defaultPost.JSP” page. This method interacts with the collection by having the filed “PostDoc” with the value as “RoomPost.”
2. **pushComments():** As the name says, this will push posted comments by the students onto the web page. This will then push the comments to the same document as discussed above. After pushing the comments to the document, it will redirect to the same page refreshing the web page with the newly added comments.
3. **getnumberOfComments():** This method gets the number of comments available in the “RoomPost” document. This helps in pushing the next comment with the specific comment number. For Example, if there are three comments in the

database, this method returns the value three. With the help of this information, the program can push the next comment with the field name as "comment4." Hence, the program can maintain a track of these fields while iterating the comments in Java and it will be easy to send this data to the JSP page.

4. **newRoomPost():** This method will insert a new post into the document. When a student posts on the web page, this method is invoked and it will insert the new post into the document. This method also gets the previously available number of posts, which helps in assigning a "Post ID" of the newly inserted post. For example, if there are three posts in the document, this method will insert the new post with the "Post ID" of four.

Similarly, there are four methods for each of the services which do the same functionality as above. Following is another similar example for "Have a room to rent" service. The four methods present for this service are:

a. **retrieveRoomRentData()**: This has the same functionality as **retriveData()** . The only difference is, this method will interact with the document having the "PostDoc" field as "RoomRentPost." Hence, it separates the dataset of other services.

b. **pushRoomRentComments()**: This will push the new comments to the same document discussed in the above point. After pushing, it will refresh the existing page, and will display the newly added comments.

c. **getnumberOfRoomRentComments()**: This will get the number of comments present in the "RoomRentPost" document.

d. **newRoomRentPost()**: This will insert the new post to the "RoomRentPost" document.

As the name of the method says, the functionality is same for all the services. The only difference is the document which is accessed by the method as per the service utilized. But the logic is the same for all the services. The same pattern is repeated for “Carpooling Controller” and “Knowledge Sharing Controller” as well.

The **HelloController** is used to check the login validation and registering of a student. If the student enters the correct data, this controller validates the data and allows the student to enter the welcome page. If the student enters incorrect data, he will not be able to log in. This controller also checks for valid data while registering.

The **WelcomeController** is the main page as soon as the student logs in, this controller enables the student to navigate to different modules and services as per the links available on this page.

JSP Pages are present under the “WEB-INF” folder “/ScsuWebApp/src/main/webapp/WEB-INF/jsp” as this is configured in the “spring-dispatcher-servlet.xml.” So, when a URL is accessed, Spring MVC first invokes the **@RequestMapping** method present in the Controller class which is associated with the URL pattern and executes that method. START

Interaction between JAVA and MongoDB

The “mongo-java-driver” is the dependency code to be added in the “pom.xml” file which downloads the respective JAR. This JAR helps in connecting MongoDB with JAVA. There are a few steps to completely interact with the database and query the data. They are: (a) Connect to MongoDB, (b) access the database, (c) access the collection, (d) access the document, and (e) perform CRUD operations on the document.

Connect to MongoDB. Connection to MongoDB is made by “MongoClient.” It is a MongoDB client with internal connection pooling. It is necessary to have one “MongoClient” instance for the entire JVM, then it creates a Mongo instance based on a (single) MongoDB node (Marchioni, 2015).

Below is the syntax for connecting to the database.

```
MongoClient mongoClient = new MongoClient( "localhost" , 27017 );

DB db = mongoClient.getDB( "ScsuDatabase" );
```

In the above syntax, “localhost” is the name of the server where MongoDB is configured and 27017 is the port number. By default, MongoDB runs on the port number 27017.

Access the database. After connecting to the MongoDB, it is necessary to access the database. In the above syntax, “ScsuDatabase” is the name of the database where the data is stored and where the query is retrieving the information from.

Access the collection. After accessing the database, it is necessary to access the collection in which the documents are stored.

```
DBCollection coll = db.getCollection("student");
```

The above code will successfully access the student collection.

Accessing the documents.

```
BasicDBObject query =  new BasicDBObject("PostDoc", "CarRent");
DBCursor cursor = coll.find();
cursor = coll.find(query);
DBObject next = cursor.next();
```

The above code will access the document which has “PostDoc” field as “CarRent.” After accessing the document, it should be obtained into a “Cursor.” The “BasicDBObject” creates an object with the given key/value. The “DBCursor” is an iterator over database results, it iterates for all the elements present in the query. The “cursor.next()” gives the next element in the result set. Thus, it is possible to get all the data available for the particular document.

Performing CRUD operations on the documents.

Inserting a document:

```
BasicDBObject doc = new BasicDBObject("FirstName", “Ashish”)
        .append("LastName", “Kastury”).append("Student Id", “123”);

coll.insert(doc);
```

The above code inserts a document with the fields

FirstName: Ashish

LastName: Kastury

Student Id: 123

Similarly, I can insert the documents as per the requirement.

Updating a document: (Marchioni, 2015)

```
BasicDBObject doc = new BasicDBObject();

doc.append("$set", new BasicDBObject().append("Student Id", “12345”));

BasicDBObject searchQuery = new BasicDBObject().append ("FirstName",“Ashish”)

coll.update(searchQuery, doc);
```

The above code will update the “student Id” field to 12345 for the document having “FirstName” as “Ashish.”

Deleting a document:

```
DBObject doc = new BasicDBObject();

doc.put("FirstName", “Ashish”);

coll.remove(doc);
```

The above syntax removes the document where the “FirstName” is “Ashish” from the collection.

Pushing arrays into the document:

This is an important concept as I need to push the comments in the form of an array into the documents.

```
BasicDBObject docToInsert = new BasicDBObject("comment"+n, comments);
docToInsert.put("author", session.getAttribute("user"));

BasicDBObject updateCommand = new BasicDBObject("$push", new
BasicDBObject("CarPoolPost.$.comments", docToInsert));

coll.update(query, updateCommand);
```

In the above code, **“comment”+n** is the ID of the comment. Ex: comment1, comment2 . . .Etc. Comments are the actuals responses which the student has replied in the “String” format. In the second line, I am also appending the author who commented on the post. “Session.getAttribute()” will get the name of the student who is currently logged in. This is an inbuilt feature in spring framework.

In the third line, the update command is prepared by using the “PUSH” keyword. This will push the “docToInsert” (which has the comments and author fields) into the comments field of “CarPoolPost document.”

Spring Configurations

While using Spring framework (Spring, n.d.), there are a few configurations to be followed as a part of the framework. There are two main configurations: (a) Web.xml and (b) webApplicaiton Context.xml.

Web.xml. Web.xml is also called a dispatcher servlet of the framework. It contains the servlet name and servlet class (Tutorials Point, n.d.b). This file is present in the "webapp/WEB-INF/" folder. When the project is initialized, the spring framework tries to load an application context from "webApplicationContext.xml." Web.xml contains a reference to webApplicationContext.xml.

```
<servlet>
        <servlet-name>spring-dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
                <init-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>classpath*:webApplicationContext.xml</param-value>
                </init-param>
                <load-on-startup>1</load-on-startup>
</servlet>
```

The servlet mapping in web.xml specifies a pattern for the URL to be entered in the web page. Based on this URL, the Spring framework invokes the controller methods. In this project, the "/" is used as the pattern, as shown below (Tutorials Point n.d.b).

```
<Servlet-mapping>
                <servlet-name>spring-dispatcher</servlet-name>
                <url-pattern>/</url-pattern>
</servlet-mapping>
```

webApplicationContext.xml. webApplicationContext.xml acts as a front controller. Whenever there is a request, the dispatcher servlet will delegate the work to this file. To enable annotation capability, it is necessary to add **<context:component-scan>** tag and give the package name. When a request comes in, spring knows to search in the defined package.

"InternalResourceViewResolver" is used with a set of predefined rules to match the URL request. "ApplicationContext" will resolve the URL pattern by adding suffix and prefix and returns the JSP pages (Tutorials Point n.d.b).

```
<context:annotation-config/>
 <context:component-scan base-package="com.scsu.controller"/>
<mvc:annotation-driven/>
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>
<property name="prefix" value="/WEB-INF/jsp/"/>
<property name="suffix" value=."jsp"/>
</bean>
```

In the above code, **base-package=*"com.scsu.controller"*** indicates spring framework to check for controller classes in the "*com.scsu.controller*" package. All the controller classes are present in the above package, additionally, "PasswordEncryption" class is also present in the same package. "DAO" class, responsible for database connection, is present in the ***com.scsu.dao*** package. Model classes "PostTo" and "CommentAuthor" are present in ***com.scsu.postTo*** package

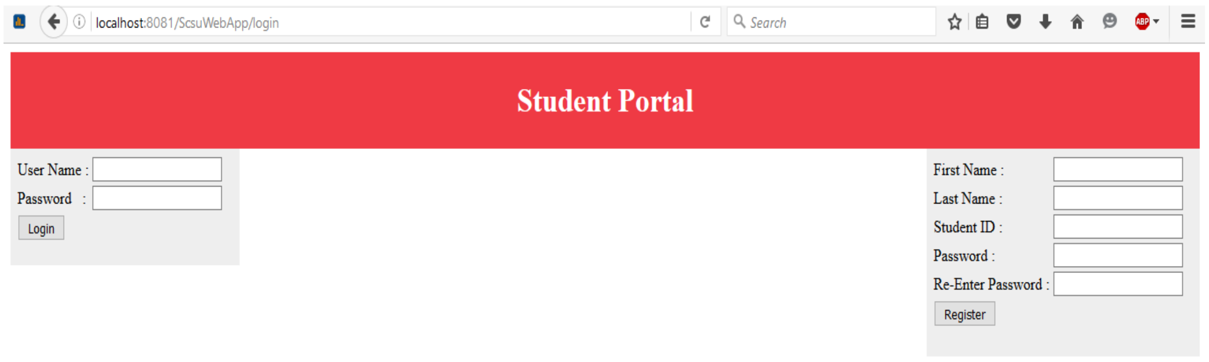
Chapter V: Implementation and Results

Introduction

The Web application developed for this project has several modules. Each module has its own services and complexity level. Modules present in the Application are as follows:

Registration module. A user should first register himself/herself on the website to avail the services provided by the application. He or she should enter their details on the page and set their password.

This module is present on the login page. The student should give their First Name, Last Name, Password and Student ID at the time of registering.

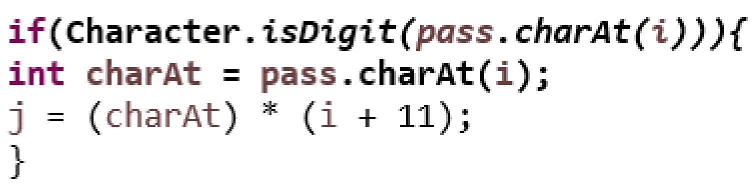


These details are validated and stored in the “student” collection. The validation includes (a) checks whether all the fields are entered and (b) checks that the passwords entered are matching. If these two rules are satisfied, then the student will get a message that they are successfully registered on the same page. At this time they can log into the website. In the above picture, the registration module is on the right side of the window.

Password encryption. When a user registers on the site, they should give a password for their account. The password is not directly sent to the database. It will be encrypted using Java logic and then stored in the database. The next time when the student tries to log in, the password they enter will be encrypted again and is matched with the stored password which was already encrypted at the time of registration.

Below is the algorithm used in the encryption:

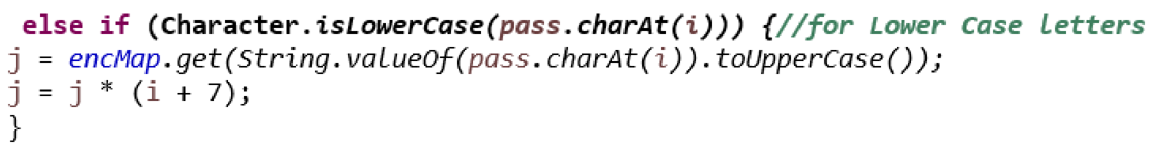
1. First, set all the Capital letters to the corresponding numbers starting from 1, Ex. A=1, B=2…Z=26.
2. Initialize an integer variable j = 0.
3. If the password contains a number, get the position of the number and add 11. Then multiply this result by the character value of the number and store in



```
if(Character.isDigit(pass.charAt(i))){
int charAt = pass.charAt(i);
j = (charAt) * (i + 11);
}
```

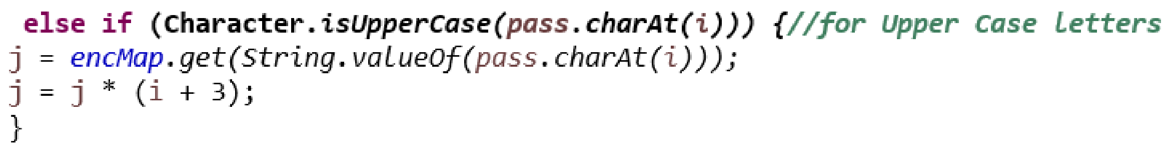
j.

4. If the password contains lowercase letters, convert to upper case letter and get the corresponding number it is mapped into j. Then multiply this number with the "sum of position number of the character" and 7.



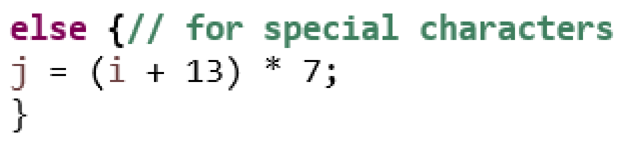
```
 else if (Character.isLowerCase(pass.charAt(i))) {//for Lower Case Letters
j = encMap.get(String.valueOf(pass.charAt(i)).toUpperCase());
j = j * (i + 7);
}
```

5. If the password contains uppercase letters, get the corresponding number mapped to it. Then multiply this with the "sum of position number of the character" and 3. Store the obtained value in j.



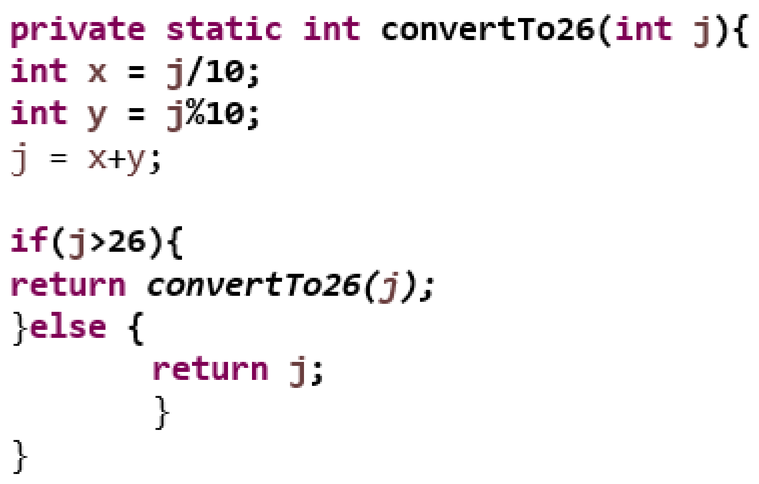
```
 else if (Character.isUpperCase(pass.charAt(i))) {//for Upper Case Letters
j = encMap.get(String.valueOf(pass.charAt(i)));
j = j * (i + 3);
}
```

6. For special characters, when none of the above condition matches, add the position number with 13, and multiply by 7.



```
else {// for special characters
j = (i + 13) * 7;
}
```

7. Now check for the value of “**j,**” if it is above 26, then convert it to a value less than 26 as below.
    a. First get the quotient and remainder to variables x and y by dividing “**j**” with 10.
    b. Then add **x** and **y**. Repeat this process until the value is less than 26.



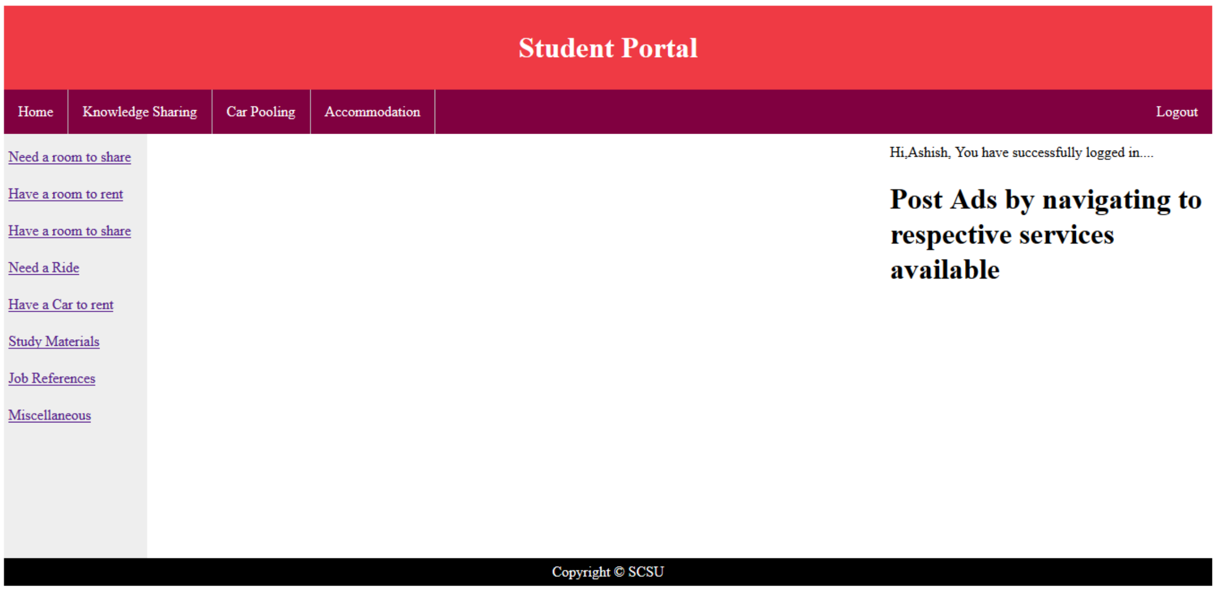
```
private static int convertTo26(int j){
int x = j/10;
int y = j%10;
j = x+y;

if(j>26){
return convertTo26(j);
}else {
        return j;
        }
}
```

8. After getting the value of “**j**,” get the corresponding letter from the map.
9. Multiply this number “**j**” with the sum of position number and 7. And get the hexadecimal value of this number.
10. Append this hexadecimal value to the letter I got from the map. Repeat the same for all the characters in the password.

Login module. This module allows users to enter into the website. A user has to provide their valid credentials to access the services provided by the application. When a user enters their login details and submits them, the details are matched with the data present in the database. If the details match exactly with the database, the user is allowed to log into the application. In the above picture, the login module is on the left side of the window.

When a student enters their credentials and hits the submit button, the spring MVC architecture redirects the user to "/ScsuWebApp/WelcomePage" if the details entered are valid. First, it comes to the "loginSuccess" method as it has the "@RequestMapping" annotation of "/WelcomePage." Then it checks whether the user name and password match with the one in the database. If it matches, it will display the welcome page, otherwise it remains on the login page and shows an error message.

Welcome page. Once the user successfully logs in, they can see the welcome page, which is the home page where they can find various services. Now they will be able to use the services provided.



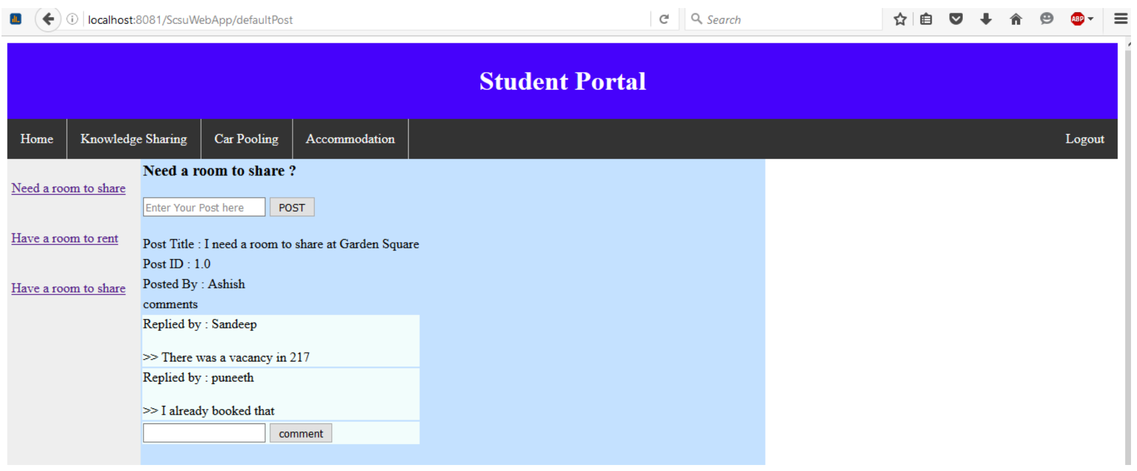
Hi,Ashish, You have successfully logged in....

Post Ads by navigating to respective services available

In the preceding picture, I can see that there are navigation links on the left side and on the menu bar. A student who has logged in can choose any of the links to avail themselves of the services. This content is written in the “welcome.jsp” page.

Accommodation module. This module helps students in finding out an accommodation at a specific location. They can check the available rooms by entering this module. They can also post the requirements of a roommate at a particular place and the price he or she is comfortable in paying. By seeing this request, interested people can join with them and plan for an accommodation.

When a user clicks on the accommodation link, spring MVC triggers the “retriveData” method as there is request mapping on this method **“@RequestMapping(value = “/defaultPost” ,method=RequestMethod.GET).”** This page shows the data entered for “Need a room to share” service. By default, it shows this page, and it also has the navigation links for other services in the accommodation module as shown in the image below.



Need a room to share ?

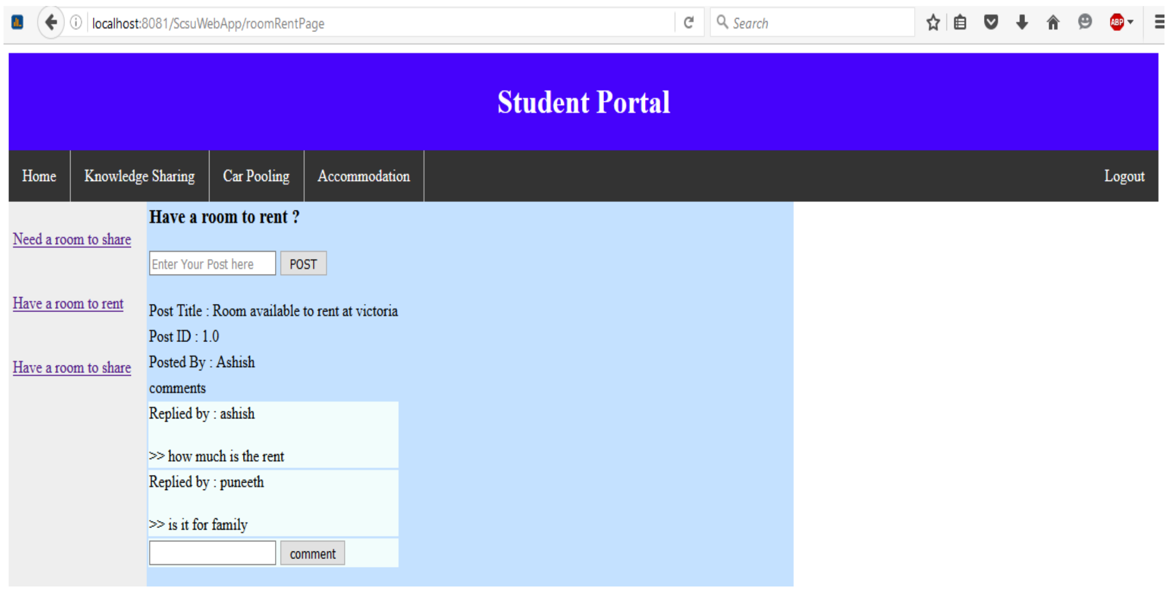
Post Title : I need a room to share at Garden Square
Post ID : 1.0
Posted By : Ashish
comments
Replied by : Sandeep

>> There was a vacancy in 217
Replied by : puneeth

>> I already booked that

As seen above in the left pane, a user can click any of the three services available and post their needs and comment on another’s posts. Below are the screenshots of the other two services.

Following is for “Have a room to rent” service:



localhost:8081/ScsuWebApp/roomRentPage

Student Portal

Home | Knowledge Sharing | Car Pooling | Accommodation | Logout

Need a room to share

Have a room to rent

Have a room to share

Have a room to rent ?

Enter Your Post here POST

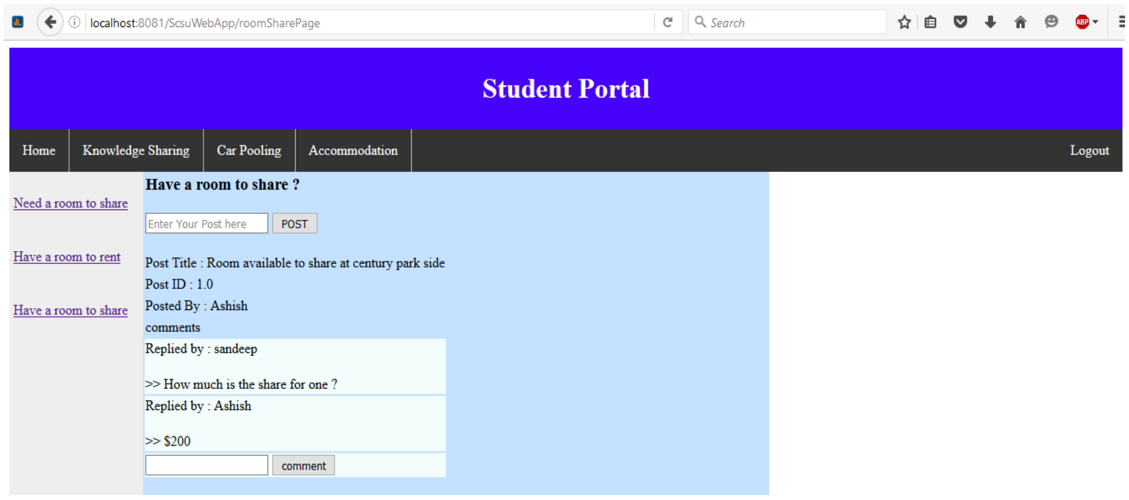
Post Title : Room available to rent at victoria
Post ID : 1.0
Posted By : Ashish
comments
Replied by : ashish

>> how much is the rent
Replied by : puneeth

>> is it for family

comment

Following is for “Have a room to share” service:



localhost:8081/ScsuWebApp/roomSharePage

Student Portal

Home | Knowledge Sharing | Car Pooling | Accommodation | Logout

Need a room to share

Have a room to rent

Have a room to share

Have a room to share ?

Enter Your Post here POST

Post Title : Room available to share at century park side
Post ID : 1.0
Posted By : Ashish
comments
Replied by : sandeep

>> How much is the share for one ?
Replied by : Ashish

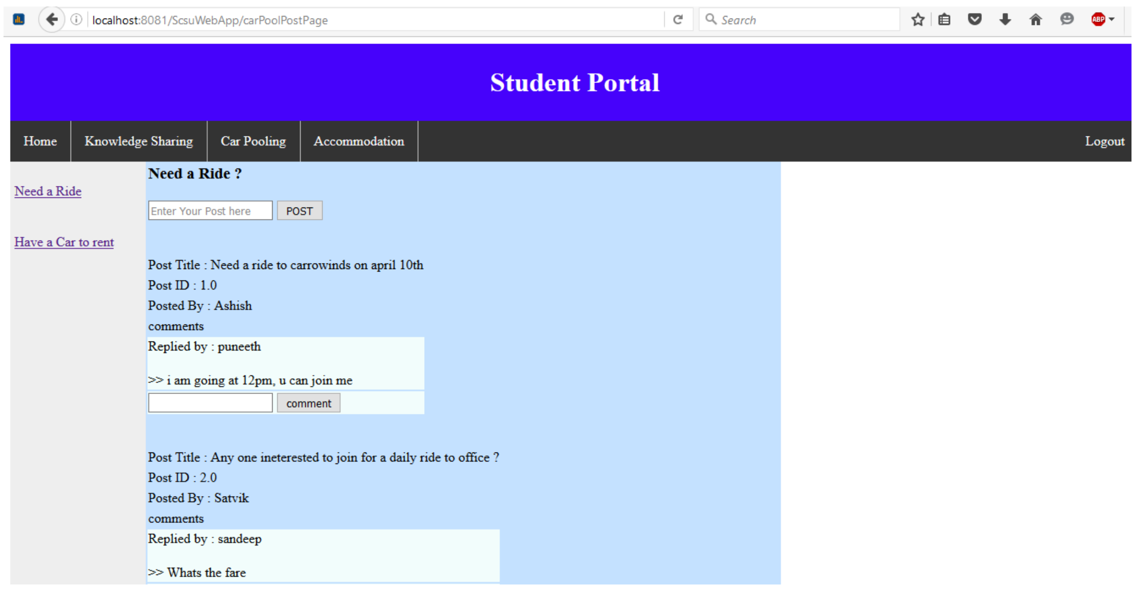
>> $200

comment

Car pooling module. In this module, students can request for any transportation or can follow other students who are planning to commute from one place to another. For example, a student can find out who is going to downtown on Sunday at a particular time so that he or she can join them. Or else they can ask other students who are interested in joining them.

When a user clicks on the carpooling module, the default action will take them to the “carPoolPostPage,” where they can search for any rides available.

In the backend system, Spring will invoke the method "retriveData" which will query the data (posts and comments) from the database which has the "posDoc" field as "CarPool." This content is displayed in "carPoolPostPage.jsp."


localhost:8081/ScsuWebApp/carPoolPostPage

Student Portal

Home | Knowledge Sharing | Car Pooling | Accommodation | Logout

Need a Ride

Have a Car to rent

Need a Ride ?

Enter Your Post here | POST

Post Title : Need a ride to carrowinds on april 10th
Post ID : 1.0
Posted By : Ashish
comments
Replied by : puneeth

>> i am going at 12pm, u can join me

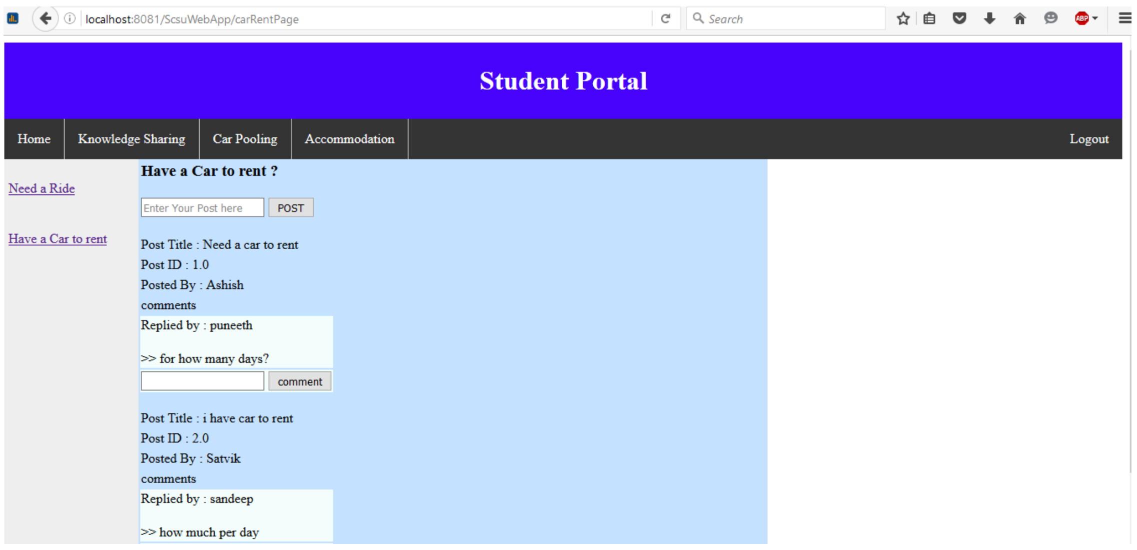
comment

Post Title : Any one ineterested to join for a daily ride to office ?
Post ID : 2.0
Posted By : Satvik
comments
Replied by : sandeep

>> Whats the fare

Similarly, when a user clicks on the "Have a Car to rent" link, spring controller invokes the "retriveCarRentData" method as this has the **@RequestMapping(value = "/carRentPage")**.

Following is the picture of "Have a Car to rent" web page:


localhost:8081/ScsuWebApp/carRentPage

Student Portal

Home | Knowledge Sharing | Car Pooling | Accommodation | Logout

Need a Ride

Have a Car to rent

Have a Car to rent ?

Enter Your Post here | POST

Post Title : Need a car to rent
Post ID : 1.0
Posted By : Ashish
comments
Replied by : puneeth

>> for how many days?

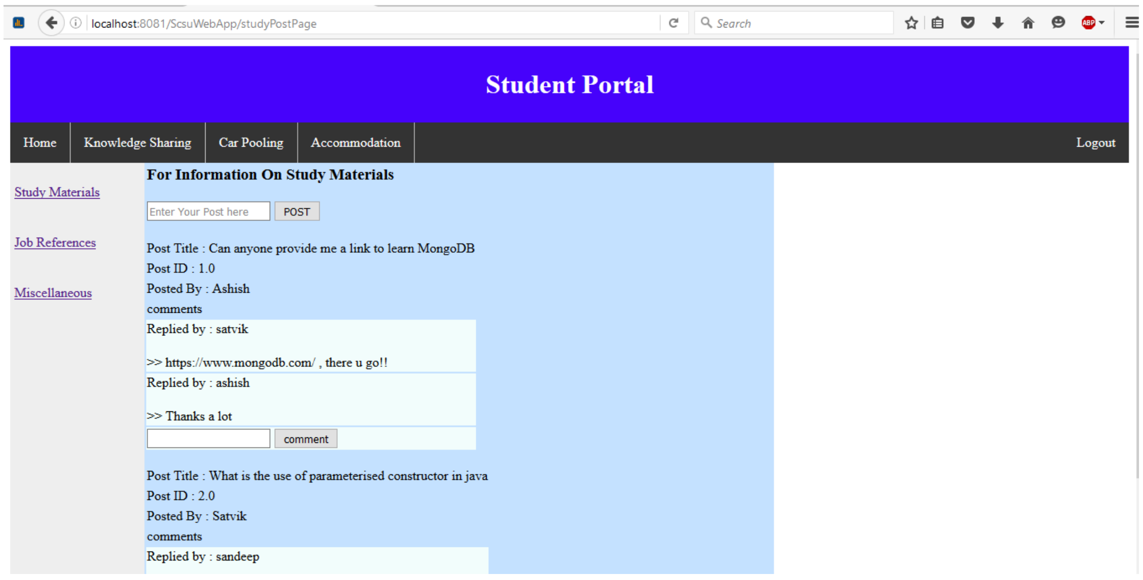
comment

Post Title : i have car to rent
Post ID : 2.0
Posted By : Satvik
comments
Replied by : sandeep

>> how much per day

Knowledge sharing module. This module enables students to post their ideas and share their knowledge in different areas. Students can select their interested topics and start gaining knowledge. They can also comment on another student's post and several students can view the comments and reply accordingly. This helps students to easily familiarize with the topics as wells as with other students.

When a student clicks on the "Knowledge Sharing module," spring controller displays the Study Materials page as this is the default method linked to the module. This means, when a user clicks on the Knowledge sharing module, "retriveData()" method of "KnowledgeController" class is invoked as this is annotated with **@RequestMapping** with **value = "/studyPostPage."** Hence the "study post page" is displayed to the student as shown below.



For Information On Study Materials

Post Title : Can anyone provide me a link to learn MongoDB
Post ID : 1.0
Posted By : Ashish
comments
Replied by : satvik

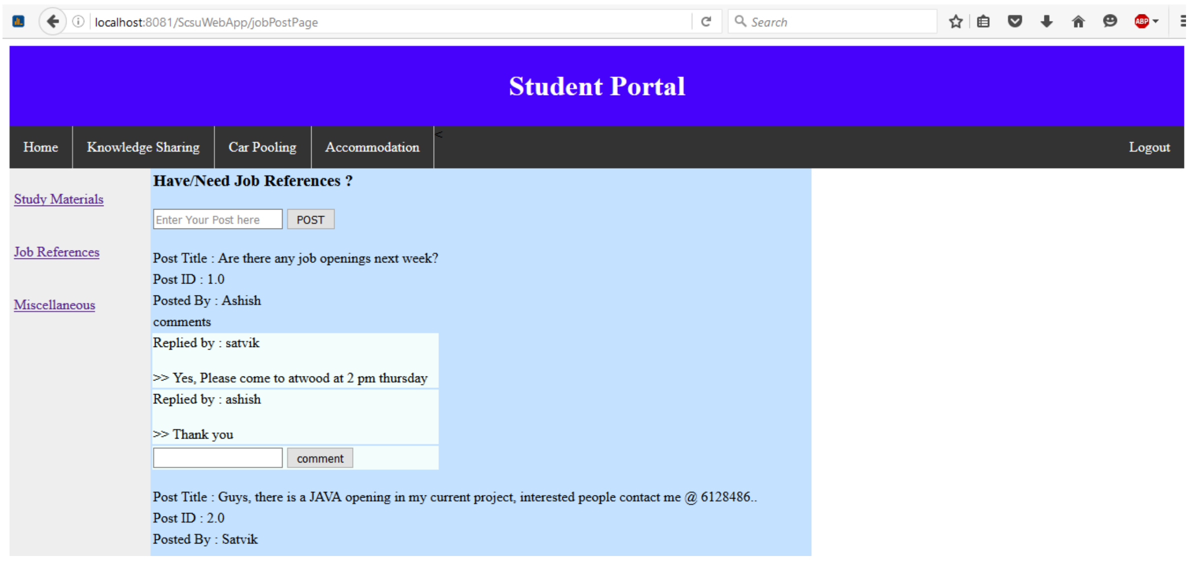
>> https://www.mongodb.com/ , there u go!!
Replied by : ashish

>> Thanks a lot

Post Title : What is the use of parameterised constructor in java
Post ID : 2.0
Posted By : Satvik
comments
Replied by : sandeep

In the preceding page, students can post the links related to the study materials.

Similarly, if a student clicks on the Job References link, he or she will be directed to that page. This page is displayed using the method "retriveJobPostData()" which is annotated with **@RequestMapping** with **value = "/jobPostPage."** In this page, students can ask and post any job references or internship positions if they know any at their workplace. This helps other students to more easily find and apply for jobs.



Have/Need Job References ?

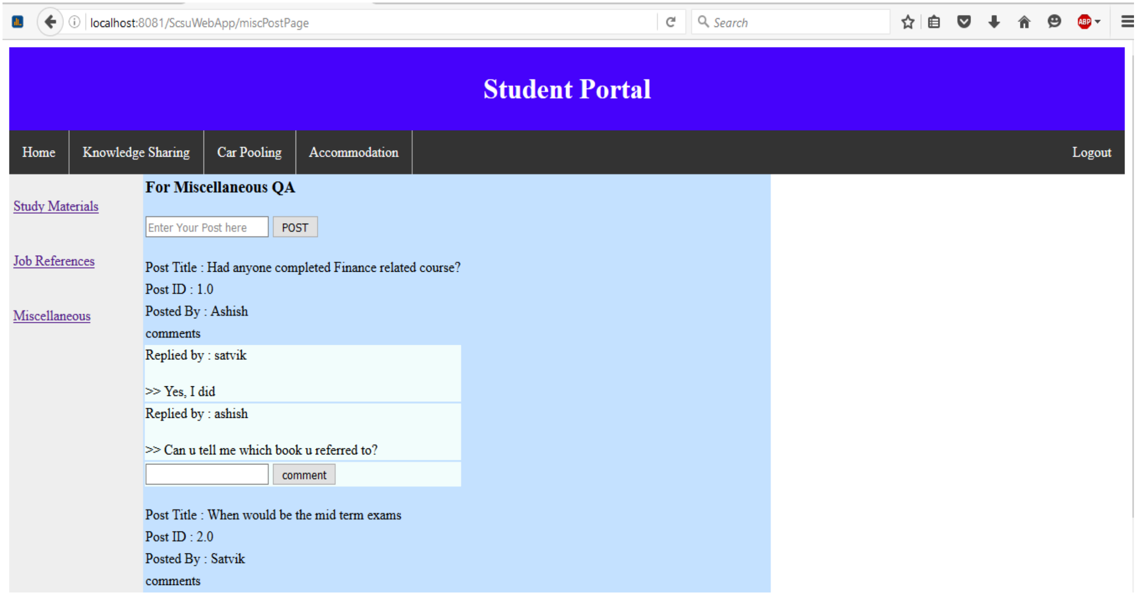
Post Title : Are there any job openings next week?
Post ID : 1.0
Posted By : Ashish
comments
Replied by : satvik

>> Yes, Please come to atwood at 2 pm thursday
Replied by : ashish

>> Thank you

Post Title : Guys, there is a JAVA opening in my current project, interested people contact me @ 6128486..
Post ID : 2.0
Posted By : Satvik

If a student clicks on the miscellaneous link, he or she will be able to ask any random topics and gain knowledge about the School, or class timings, courses, and any other general knowledge topics. This page is displayed by the "retriveMiscPostData()" method which is annotated with **@RequestMapping** with **value = "/miscPostPage."**



For Miscellaneous QA

Post Title : Had anyone completed Finance related course?
Post ID : 1.0
Posted By : Ashish
comments
Replied by : satvik

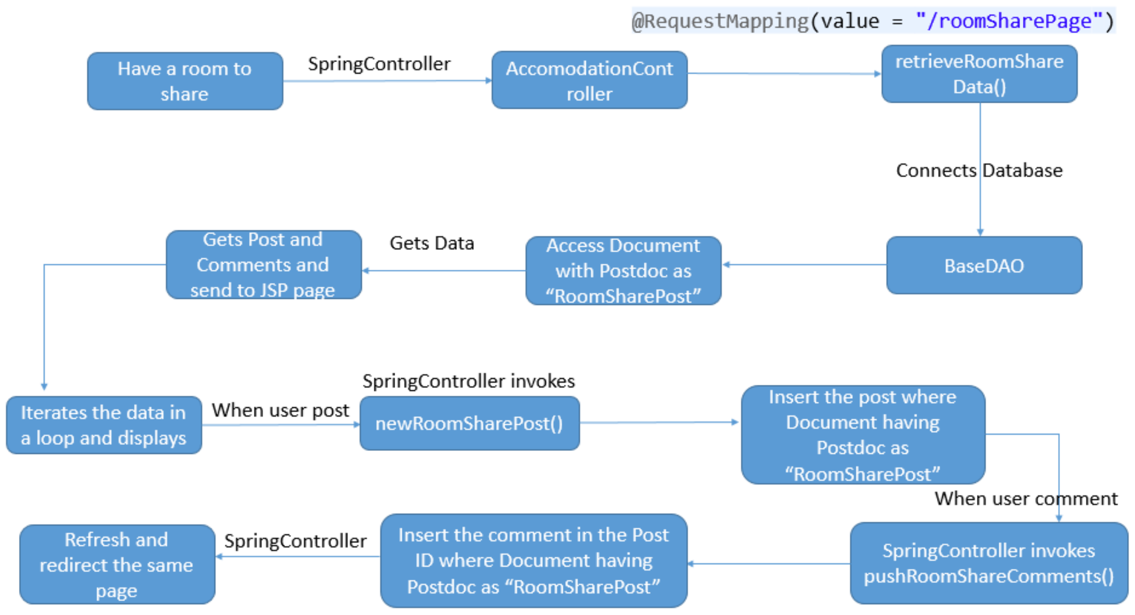
>> Yes, I did
Replied by : ashish

>> Can u tell me which book u referred to?

Post Title : When would be the mid term exams
Post ID : 2.0
Posted By : Satvik
comments

Technical Flow in the Background

Following is the technical flow when a user clicks on a link. This is to show the processing of the website technically using JAVA and MongoDB; for example, in the Accommodation Module, if a user clicks on "Have a room to share" link.


@RequestMapping(value = "/roomSharePage")
Have a room to share
SpringController
AccomodationController
retrieveRoomShareData()
Connects Database
BaseDAO
Access Document with Postdoc as "RoomSharePost"
Gets Data
Gets Post and Comments and send to JSP page
Iterates the data in a loop and displays
When user post
SpringController invokes
newRoomSharePost()
Insert the post where Document having Postdoc as "RoomSharePost"
When user comment
SpringController invokes pushRoomShareComments()
Insert the comment in the Post ID where Document having Postdoc as "RoomSharePost"
SpringController
Refresh and redirect the same page

Future Enhancements

1. Students should be able to edit the posts and comments written by them
2. Students should be able to delete the Posts and comments.
3. Uploading documents.
4. Private Texting.

Conclusion

Hence, as per the preceding diagram, it is possible to build a website using MongoDB and Spring MVC and navigate to different services present in the application. Also, it has been shown that the difference between the RDBMS approach and the MongoDB approach with a few design possibilities. There are two possible designs in an RDBMS which have a few disadvantages like redundancy of data and an increase in the number of tables. However, the MongoDB design proves that redundancy can be controlled with its unique behavior of storing the data in the form of arrays. It also has the minimum number of collections (referred to tables in an RDBMS) as it can store any type of documents because of its schema-less structure.

In addition, Spring MVC interacts efficiently with MongoDB having a minimum number of controller classes and is able to query and store the data in the database. In this test, I have only used one model object “PostTo” which is able to store the data in memory while performing operations on different documents. There are also no complicated queries written in JAVA to interact with MongoDB because I have maintained only one collection for all the services and student details as well, reducing the work of the programmer. Password authentication is also done while registering to the website. When the student tries to log in, the password they enter will again be encrypted and matched to the existing encrypted password in the database. Hence, all above points conclude that the student portal is easily and efficiently developed with the Spring MVC and MongoDB.

References

Eclipse Foundation. (2015). *Eclipse (Version Mars).* Retrieved from https://eclipse.org/downloads/

Jet Brains. (2015). IntelliJ IDEA (Version 14.1.5.) Retrieved from https://www.jetbrains.com/idea/download/

Marchioni, F. (2015). *MongoDB for Java developers*. Birmingham—Mumbai: PACKT.

MongoDB Inc. (2015a). *MongoDB* (Version 3.0.1). Retrieved from https://www.mongodb.org/downloads#production

MongoDB, Inc. (2015b). *Sharding*. Retrieved from https://docs.mongodb.org/manual/core/sharding-introduction/

Spring. (n.d.). Spring framework. Retrieved from https://projects.spring.io/spring-framework/

thenewboston. (2015, April 11). *MongoDB tutorial for beginners-1. Installing Mongo*. Retrieved from https://www.youtube.com/watch?v=1uFY60CESlM

Tutorials Point. (n.d.a). *MongoDB—Overview.* Retrieved from http://www.tutorialspoint.com/mongodb/mongodb_overview.htm

Tutorials Point (n.d.b). Spring - MVC Framework Tutorial. Retrieved from http://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm

Appendix

HelloController

```
package com.scsu.controller;


import static org.springframework.util.StringUtils.isEmpty;

import javax.servlet.http.HttpSession;

import org.bson.Document;
import org.springframework.stereotype.Controller;
import org.springframework.util.StringUtils;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.SessionAttributes;
import org.springframework.web.servlet.ModelAndView;

import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.scsu.dao.BaseDao;

@Controller
@SessionAttributes(“user”)
public class HelloController{
	DBCollection collection ;
	DB db = null;
	@RequestMapping(value =“/login” , method=RequestMethod.GET)
	public ModelAndView login() {
		System.out.println(“method /login”);
		ModelAndView modelAndView = new ModelAndView(“login”);
		BaseDao bd= new BaseDao();
		collection = bd.getConnection();
		//modelAndView.addObject(“welcomeMessage,”““);
		return modelAndView;
	}

	@RequestMapping(value =“/logout” , method=RequestMethod.GET)
	public ModelAndView logout(HttpSession session) {
		System.out.println(“method /logout”);
		 session.invalidate();
```

```
            ModelAndView modelAndView = new ModelAndView("login");
             return modelAndView;
    }

    //returns in the form of string and spring will render the page.No use of Model View
/*  @RequestMapping(value ="/login.jsp" , method=RequestMethod.GET)
    public String login(){
            System.out.println("method");
            return "login";
    }*/

    @RequestMapping(value = "/WelcomePage" ,method=RequestMethod.POST)
    public ModelAndView loginSuccess(@RequestParam("userName") String name ,

    @RequestParam("password")String password,HttpSession value){

            System.out.println("method -Login Success");
            ModelAndView modelAndView = null;
            //check from db
            BaseDao bd= new BaseDao();
            collection = bd.getConnection();
            Document document = new Document();
            document.get(name);

            PasswordEncryption pe = new PasswordEncryption();
            String encPass = pe.encPassword(password);

            BasicDBObject query = new BasicDBObject("FirstName," name);

            DBCursor cursor = collection.find();
            cursor = collection.find(query);
            String dbPass = "";
            try {
                    while(cursor.hasNext()) {
                            dbPass =   cursor.next().get("password").toString();
                             System.out.println(" here is the value password" + dbPass);
                    }
                } finally {
                    cursor.close();
                }

            if (dbPass.equals(encPass)){
                    modelAndView = new ModelAndView("WelcomePage");
                    modelAndView.addObject("welcomeMessage,""Hi,"+name);
                    value.setAttribute("user," name);
                    return modelAndView;
```

```
		}else {
			modelAndView = new ModelAndView("/login");
			modelAndView.addObject("welcomeMessage,""Invalid username or
password");
		}

/*		if(name.equals("ashish") && password.equals("ashish")){
			System.out.println("Inside success");
			modelAndView = new ModelAndView("WelcomePage");
			modelAndView.addObject("welcomeMessage,""Hi,"+name);
			return modelAndView;
		}else{
			modelAndView = new ModelAndView("login");
			modelAndView.addObject("welcomeMessage,""Invalid username or
password");
		}*/

		return modelAndView;
	}

	@RequestMapping(value = "/homePage" ,method=RequestMethod.GET)
	public ModelAndView homePage(HttpSession session){

		ModelAndView modelAndView =  new ModelAndView("/homePage");

		modelAndView.addObject("welcomeMessage," session.getAttribute("user"));

		return modelAndView;
	}
	@RequestMapping(value = "/register" ,method=RequestMethod.POST)
	public ModelAndView register(@RequestParam("FName") String fName ,

	@RequestParam("LName")String lName,@RequestParam("SId")String sId,

	@RequestParam("password")String password,@RequestParam("repassword")String
rePassword){
		ModelAndView modelAndView = null;
		System.out.println("method -register Success");

		if(isEmpty(fName) || isEmpty(lName) || isEmpty(sId) || isEmpty(password) ||
isEmpty(rePassword)){
			modelAndView = new ModelAndView("login");
			modelAndView.addObject("registerMsg,""Please Enter all the
details");
			return modelAndView;
		}
```

```
            if(!password.equals(rePassword)){
                    modelAndView = new ModelAndView(“login”);
                    modelAndView.addObject(“registerMsg,”“Passwords Do Not
Match”);
                    return modelAndView;
            }
            PasswordEncryption pe = new PasswordEncryption();
            String encPass = pe.encPassword(password);
            //check from db
            BaseDao bd= new BaseDao();
            DBCollection coll = bd.getConnection();

             BasicDBObject doc = new BasicDBObject(“FirstName,” fName)
          .append(“LastName,” lName).append(“Student Id,” sId)
          .append(“password,” encPass);
             coll.insert(doc);
    /*
            if(name.equals(“ashish”) && password.equals(“ashish”)){
                    System.out.println(“Inside success”);
                    modelAndView = new ModelAndView(“WelcomePage”);
                    modelAndView.addObject(“welcomeMessage,”“Hi,”+name);
                    return modelAndView;
            }else{
                    modelAndView = new ModelAndView(“login”);
                    modelAndView.addObject(“welcomeMessage,”“Invalid username or
password”);
            }*/
            modelAndView = new ModelAndView(“login”);
            modelAndView.addObject(“registerMsg,”“Succesfully Registered”);
            return modelAndView;
    }

    @RequestMapping(value =“/insert” , method=RequestMethod.POST)
    public ModelAndView insertDocument(@RequestParam(“userName”) String name ,
                    @RequestParam(“password”)String password) {
            System.out.println(“Insert Doc Method”);
            String insertName = name;
            String insertPassword = password;

            System.out.println(“----Name :” + insertName + “---pass + “ + password);
            ModelAndView modelAndView = new ModelAndView(“insert”);

            BaseDao bd= new BaseDao();
            collection = bd.getConnection();

            BasicDBObject doc = new BasicDBObject(“name,” insertName)
   .append(“LastName,” insertPassword);
```

```
				collection.insert(doc);
				modelAndView.addObject(“insertMessage,” “Details of “ + name + “Inserted
succesfully : “
							+ “Total documents :” + collection.getCount());
				return modelAndView;
		}

		@RequestMapping(value =“/update” , method=RequestMethod.POST)
		public ModelAndView updateDocument(@RequestParam(“userName”) String name ,
						@RequestParam(“Age”)String age) {
				System.out.println(“update Doc Method”);
				String insertName = name;
				String insertAge = age;

				System.out.println(“----Name :” + insertName + “---Age + “ + insertAge);
				ModelAndView modelAndView = new ModelAndView(“update”);

				BaseDao bd= new BaseDao();
				collection = bd.getConnection();

				BasicDBObject newDocument = new BasicDBObject();
				newDocument.append(“$set,” new
BasicDBObject().append(“age,”insertAge));
				//newDocument.append(“Age,” insertAge);
				DBObject searchQuery = new BasicDBObject().append(“name,” name);
				collection.update(searchQuery, newDocument);

				modelAndView.addObject(“updateMessage,” “Details of “ + name + “updated
succesfully : “
							+ “Total documents :” + collection.getCount());
				return modelAndView;
		}

		@RequestMapping(value =“/delete” , method=RequestMethod.POST)
		public ModelAndView deleteDocument(@RequestParam(“userName”) String name)
{
				System.out.println(“delete Doc Method”);

				ModelAndView modelAndView = new ModelAndView(“delete”);

				BaseDao bd= new BaseDao();
				collection = bd.getConnection();

				DBObject searchQuery = new BasicDBObject().append(“name,” name);
				collection.remove(searchQuery);
```

```
                modelAndView.addObject("deleteMessage," "Details of " + name + "deleted
succesfully : "
                                + "Total documents after deleteing:" + collection.getCount());
                return modelAndView;
        }
}

// not used when using annotations.
/*public class HelloController extends AbstractController{

        @Override
        protected ModelAndView handleRequestInternal(HttpServletRequest request,
                        HttpServletResponse response) throws Exception {

                ModelAndView modelAndView = new ModelAndView("HelloPage");
                modelAndView.addObject("welcomeMessage,""Hi,this is the first spring
App");

                return modelAndView;
        }

}*/
```

WelcomeController

```
package com.scsu.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.view.RedirectView;

import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.scsu.dao.BaseDao;

@Controller
public class WelcomeController {
	DBCollection collection ;
	DB db = null;

	@RequestMapping(value = “/insert” )
	public ModelAndView insertTest(){

		System.out.println(“method -Login Success”);
		ModelAndView modelAndView = null;
		//check from db

			modelAndView = new ModelAndView(“insert”);

		return modelAndView;
	}

	@RequestMapping(value = “/accomodation” )
	public ModelAndView accomodation(){

		System.out.println(“method -accomodation Success”);
		ModelAndView modelAndView = null;
		//check from db

			modelAndView = new ModelAndView(new
RedirectView(“/ScsuWebApp/defaultPost”));

		return modelAndView;
	}

	@RequestMapping(value = “/showAccPost”)
```

```
   public ModelAndView showAcc() throws Exception {

                        System.out.println("Show Posts after clicking get button");


      ModelAndView mav = new ModelAndView("showAccPost");
      return mav;
   }

         @RequestMapping(value = "/afterShowAccPost" ,method=RequestMethod.POST)
   public ModelAndView testShowAcc(@RequestParam("place") String name) throws
Exception {
               if(name.isEmpty()){
                        System.out.println("Display nothing");
               }else{
                        BaseDao bd= new BaseDao();
                        collection = bd.getConnection();
                        System.out.println(collection);
                        System.out.println("Display ");
               }

      ModelAndView mav = new ModelAndView("afterShowAccPost");
      return mav;
   }








/*
//       @RequestMapping(value = "/defaultPost" ,method=RequestMethod.GET)
   public ModelAndView showDefaultPost() throws Exception {
               System.out.println("Display default posts");
               String postId="" ;
                String title = "";
               DBCollection coll = null;
               BaseDao bd = new BaseDao();
               coll =   bd.getConnection();
               BasicDBObject query =  new BasicDBObject("FirstName," "Puneeth");
         //new BasicDBObject("Post.postId," "room");//
               //query.put("Post.postId," "room");
               DBCursor cursor = coll.find();
               cursor = coll.find(query);
```

```
                    try {
                                while(cursor.hasNext()) {
                                 System.out.println(cursor.next());
                                   // System.out.println(" here is the value " +
cursor.curr().get("type").toString());
                              //  System.out.println(" here is the value " +
cursor.curr().get("Post").toString());

System.out.println(cursor.curr().get("Post.postId.title").toString());
                                }
                              } finally {
                                cursor.close();
                              }

                    DBObject next = cursor.next();
                    BasicDBList listEditions = (BasicDBList)next.get("Post");
          //        System.out.println(listEditions.toString());
                    for(Object element: listEditions) {
                      BasicDBList comments =
(BasicDBList)((BasicDBObject)element).get("comments");
                      for(Object comment: comments) {//unable to get single comment
                                String comment1 =
(String)((BasicDBObject)comment).get("comment1");
                                System.out.println("comment1 in looop : "+ comment1);
                      }
                      String comment1 = (String)((BasicDBObject)element).get("comment1");
          //          System.out.println("comment1 : "+ comment1);

                       for(Object lie: comments) {
                             System.out.println("A");//to test how many items in list
           //      System.out.println(lie);
                 //System.out.println(((BasicDBObject)lie).get("fromDate"));
             }



                       //single strings
                              postId = (String)((BasicDBObject)element).get("postId");
                               title = (String)((BasicDBObject)element).get("title");
          //                  System.out.println(postId);
          //                  System.out.println(title);
                       for(Object lie: listInsertions) {
                          System.out.println(lie);
                          //System.out.println(((BasicDBObject)lie).get("fromDate"));
                       }
                    }
                    retriveData();//CHECK - not sure of this
                    List<String> postList = new ArrayList<String>();
```

```
                  RoomPostTo roomPost = new RoomPostTo();
                  roomPost.setPostTitle(title);
                  //roomPost.set

         postList.add(“Comment1”);
         postList.add(“Comment2”);
         postList.add(“Comment3”);
      ModelAndView mav = new ModelAndView(“defaultPost”);
      mav.addObject(“postList,”postList);
      mav.addObject(“defaultsPosts,”“TODO - display default posts here,”);
      return mav;
   }*/
}
```

PasswordEncryption:

```
package com.scsu.controller;

import java.util.HashMap;
import java.util.Map;

public class PasswordEncryption {

	private static Map<String, Integer> encMap = new HashMap<>();

	static {
		encMap.put(“A,” 1);
		encMap.put(“B,” 2);
		encMap.put(“C,” 3);
		encMap.put(“D,” 4);
		encMap.put(“E,” 5);
		encMap.put(“F,” 6);
		encMap.put(“G,” 7);
		encMap.put(“H,” 8);
		encMap.put(“I,” 9);
		encMap.put(“J,” 10);
		encMap.put(“K,” 11);
		encMap.put(“L,” 12);
		encMap.put(“M,” 13);
		encMap.put(“N,” 14);
		encMap.put(“O,” 15);
		encMap.put(“P,” 16);
		encMap.put(“Q,” 17);
		encMap.put(“R,” 18);
		encMap.put(“S,” 19);
		encMap.put(“T,” 20);
		encMap.put(“U,” 21);
		encMap.put(“V,” 22);
		encMap.put(“W,” 23);
		encMap.put(“X,” 24);
		encMap.put(“Y,” 25);
		encMap.put(“Z,” 26);
	}

	public String encPassword(String pass) {
		String encPass = ““;

		StringBuilder sb = new StringBuilder();
		for (int i = 0; i < pass.length(); i++) {
				int j = 0;
				if (Character.isDigit(pass.charAt(i))) {//for numbers
					int charAt = pass.charAt(i);
					j = (charAt) * (i + 11);
				} else if (Character.isLowerCase(pass.charAt(i))) {//for Lower Case
letters
				//	char a = pass.charAt(i);
					j = encMap.get(String.valueOf(pass.charAt(i)).toUpperCase());
					j = j * (i + 7);
				} else if (Character.isUpperCase(pass.charAt(i))) {//for Upper Case
letters
					j = encMap.get(String.valueOf(pass.charAt(i))); // (i+1);
					j = j * (i + 3);
				} else {// for special characters
					j = (i + 13) * 7;
				}

				if (j > 26) {
					j = convertTo26(j);
				}
				boolean isFound = false;
				for (String key : encMap.keySet()) {
					if (encMap.get(key).equals(j) && !isFound) {
						// Integer.toHexString(i * j);
						sb.append(key + Integer.toHexString((i + 7) * j));
						isFound = true;
					}
				}
			}
			encPass = sb.toString();
			return encPass;
		}

		private static int convertTo26(int j) {
			int x = j / 10;
			int y = j % 10;
			j = x + y;

			if (j > 26) {
				return convertTo26(j);
			} else {
				return j;
			}

		}
}
```

Accommodation Controller

```
package com.scsu.controller;

import java.util.ArrayList;
import java.util.List;

import javax.servlet.http.HttpSession;
import javax.websocket.server.PathParam;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.view.RedirectView;

import com.mongodb.BasicDBList;
import com.mongodb.BasicDBObject;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.scsu.dao.BaseDao;
import com.scsu.postTo.CommentAuthor;
import com.scsu.postTo.PostTo;


@Controller
public class AccommodationController {


        /**Shows the default posts when accommodation link is clicked.
         * @return
         * @throws Exception
         */
        @RequestMapping(value = "/defaultPost" ,method=RequestMethod.GET)
   public ModelAndView retriveData() throws Exception {
                System.out.println("--------------Retrieve Data---------");
                List<PostTo> roomPostList = new ArrayList<PostTo>();


                Double postId=null ;
                 String title = "";
                 String postAuthor = "";
                DBCollection coll = null;
```

```
            BaseDao bd = new BaseDao();
            coll =   bd.getConnection();
            BasicDBObject query =  new BasicDBObject(“PostDoc,” “RoomPost”);
        //new BasicDBObject(“Post.postId,” “room”);//
            //query.put(“Post.postId,” “room”);
            DBCursor cursor = coll.find();
            cursor = coll.find(query);

            DBObject next = cursor.next();
            BasicDBList listEditions = (BasicDBList)next.get(“RoomPost”);
            System.out.println(listEditions.toString());
            System.out.println(“size : + “ + listEditions.size());
            for(Object element: listEditions) {

System.out.println(“********************************************************”
);
                PostTo roomPost = new PostTo();
                //single strings
                    postId = (Double)((BasicDBObject)element).get(“postId”);
                    title = (String)((BasicDBObject)element).get(“title”);
                    postAuthor = (String)((BasicDBObject)element).get(“postAuthor”);

                    System.out.println(“postID” + postId);
                    System.out.println(“title : “+title);
                    System.out.println(“postAuthor : “+postAuthor);

                    roomPost.setPostTitle(title);
                    roomPost.setPostId(postId);
                    roomPost.setPostAuthor(postAuthor);
                    //Get comments
                BasicDBList comments =
(BasicDBList)((BasicDBObject)element).get(“comments”);
                System.out.println(“element get comments” + comments.toString());
                int i=1;
                List<String> commentList = new ArrayList<String>();


                //Gets induvidual comments from list of comments
                for(Object comment: comments) {//unable to get single comment
                        List<CommentAuthor> advancedCommentList = new
ArrayList<CommentAuthor>();
                        CommentAuthor ca = new CommentAuthor();
                    //  CommentAuthor ca = new CommentAuthor();
                        String comment1 =
(String)((BasicDBObject)comment).get(“comment”+i);
                        commentList.add(comment1);
                     //   advancedCommentList.add(comment1);
                       String author = (String)((BasicDBObject)comment).get("author");
                       System.out.println(" Author : " + author);
                        System.out.println("comment1 in looop : "+ comment1);
                        i++;
                        /*roomPost.setPostAuthor(author);
                        roomPost.setComments(commentList);*/
                        ca.setComment(comment1);
                        ca.setCommentAuthor(author);
                        advancedCommentList.add(ca);
                        roomPost.getAdvancedComments().addAll(advancedCommentList);
                }

                    roomPostList.add(roomPost);
            }
            /* roomPost.setComments(commentList);
                roomPostList.add(roomPost);*/
                ModelAndView mav = new ModelAndView("defaultPost");
            mav.addObject("postList,"roomPostList);
            mav.addObject("defaultsPosts,""TODO - display default posts here,");
            return mav;
}



        /**Push the comments entered in the text box and refreshes the page.
         * @param comments
         * @param id
         * @return
         */
        @RequestMapping(value = "/pushComments/{id}" )
        public ModelAndView pushComments(@PathParam("comments") String
comments,@PathVariable("id") Double id,HttpSession session){ //,@PathVariable("id")
Double id

            System.out.println(" Push the comments to db" + comments);
            System.out.println(" ID from JSP" + id);
            ModelAndView modelAndView = null;

            if(comments.isEmpty()){
                      modelAndView = new ModelAndView(new
RedirectView("/ScsuWebApp/defaultPost"));
                      System.out.println("Commments are blank");
                      return modelAndView;
            }
            DBCollection coll = null;
            BaseDao bd = new BaseDao();
```

```
		coll =   bd.getConnection();


		BasicDBObject query =  new BasicDBObject("PostDoc," "RoomPost");
		query.put("RoomPost.postId," id);

		// get number of comments to append next comment
		int n = getnumberOfComments(query,coll,id);

		//increment the comment to push as next comment number
		n =n+1;
		//Below code is to push the comment
		BasicDBObject docToInsert = new BasicDBObject("comment"+n,
comments);
		docToInsert.put("author," session.getAttribute("user"));

		BasicDBObject updateCommand = new BasicDBObject("$push," new
BasicDBObject("RoomPost.$.comments," docToInsert));

		coll.update(query, updateCommand);
		System.out.println(coll.findOne().toString());


		modelAndView = new ModelAndView(new
RedirectView("/ScsuWebApp/defaultPost"));

		return modelAndView;
	}


	//returns the number of comments in a post
	private int getnumberOfComments(BasicDBObject query,DBCollection coll, Double
id){

		DBCursor cursor = coll.find(query);

		DBObject next = cursor.next();
		BasicDBList listEditions = (BasicDBList)next.get("RoomPost");
		System.out.println(listEditions.toString());
		System.out.println("size : + " + listEditions.size());
		 BasicDBList comments =null;
		for(Object element: listEditions) {
		   //single strings
			double postId = (Double)((BasicDBObject)element).get("postId");
			 System.out.println("Post id :" + postId + " id :" + id );
			// if postid = id, get the number of comments of specific post id
			if(postId == id){
```

```
                                        comments =
(BasicDBList)((BasicDBObject)element).get(“comments”);
                                        System.out.println(“element get comments” +
comments.toString() + “coments size :” + comments.size());
                              }

                    }
                    return comments.size();
          }

          @RequestMapping(value = “/accPost” )
          public ModelAndView newRoomPost(@RequestParam(“accPost”) String
newPost,HttpSession session){

                    System.out.println(“New post -----------” + newPost);
                    //first get number of post in the room post and push the new post  with next
post id
                    BasicDBObject query =  new BasicDBObject(“PostDoc,”
“RoomPost”);//TODO - user name from session
                    BaseDao bd = new BaseDao();
                    DBCollection coll =    bd.getConnection();
                    DBCursor cursor = coll.find(query);
                    DBObject next = cursor.next();
                    BasicDBList postList = (BasicDBList)next.get(“RoomPost”);

                    System.out.println(“Number of posts : “ + postList.size());

                    Double nextPost = (double) (postList.size() + 1);

                    //push new post with post id as nextPost
          //        query.put(“RoomPost.postId,” nextPost);

                    List<DBObject> comments = new ArrayList<DBObject>();
                    /* comments.add(new BasicDBObject(“Author Ashish 1,”“second”));
                     comments.add(new BasicDBObject(“Author Ashish 2,”“third”));*/

                    //Below code is to push the post
                    BasicDBObject docToInsert = new BasicDBObject(“postId,” nextPost);
                    docToInsert.put(“postAuthor,” session.getAttribute(“user”));//TODO - user
name from session
                    docToInsert.put(“title,” newPost);
                    docToInsert.put(“comments,” comments);

                    BasicDBObject updateCommand = new BasicDBObject(“$push,” new
BasicDBObject(“RoomPost,” docToInsert));
                    coll.update(query, updateCommand);
```

```
                ModelAndView modelAndView = new ModelAndView(new
RedirectView(“/ScsuWebApp/defaultPost”));

                return modelAndView;
        }


        //------------below is for ROOM RENT POST-------------------------

        @RequestMapping(value = “/roomRentPage” )
        public ModelAndView retrieveRoomRentData(){

                System.out.println(“roomRentPage Success”);
                List<PostTo> roomPostList = new ArrayList<PostTo>();



                Double postId=null ;
                 String title = ““;
                 String postAuthor = ““;
                DBCollection coll = null;
                BaseDao bd = new BaseDao();
                coll =   bd.getConnection();
                BasicDBObject query =  new BasicDBObject(“PostDoc,” “RoomRentPost”);
        //new BasicDBObject(“Post.postId,” “room”);//
                //query.put(“Post.postId,” “room”);
                DBCursor cursor = coll.find();
                cursor = coll.find(query);

                DBObject next = cursor.next();
                BasicDBList listEditions = (BasicDBList)next.get(“RoomRentPost”);
                System.out.println(listEditions.toString());
                System.out.println(“size : + “ + listEditions.size());
                for(Object element: listEditions) {

System.out.println(“***********************************************************”
);
                   PostTo roomPost = new PostTo();
                   //single strings
                         postId = (Double)((BasicDBObject)element).get(“postId”);
                         title = (String)((BasicDBObject)element).get(“title”);
                         postAuthor = (String)((BasicDBObject)element).get(“postAuthor”);

                         System.out.println(“postID” + postId);
                         System.out.println(“title : “+title);
                         System.out.println(“postAuthor : “+postAuthor);

                         roomPost.setPostTitle(title);
```

```
                              roomPost.setPostId(postId);
                              roomPost.setPostAuthor(postAuthor);
                              //Get comments
                        BasicDBList comments =
(BasicDBList)((BasicDBObject)element).get("comments");
                        System.out.println("element get comments" + comments.toString());
                        int i=1;
                        List<String> commentList = new ArrayList<String>();


                        //Gets induvidual comments from list of comments
                        for(Object comment: comments) {//unable to get single comment
                               List<CommentAuthor> advancedCommentList = new
ArrayList<CommentAuthor>();
                               CommentAuthor ca = new CommentAuthor();
                              //  CommentAuthor ca = new CommentAuthor();
                               String comment1 =
(String)((BasicDBObject)comment).get("comment"+i);
                               commentList.add(comment1);

                             //   advancedCommentList.add(comment1);
                              String author = (String)((BasicDBObject)comment).get("author");
                              System.out.println(" Author : " + author);
                               System.out.println("comment1 in looop : "+ comment1);
                               i++;
                               /*roomPost.setPostAuthor(author);
                               roomPost.setComments(commentList);*/
                               ca.setComment(comment1);
                               ca.setCommentAuthor(author);
                               advancedCommentList.add(ca);
                               roomPost.getAdvancedComments().addAll(advancedCommentList);
                        }

                              roomPostList.add(roomPost);
                      }
                      /* roomPost.setComments(commentList);
                       roomPostList.add(roomPost);*/
                       ModelAndView mav = new ModelAndView("roomRentPage");
                  mav.addObject("postList,"roomPostList);
                  mav.addObject("defaultsPosts,""TODO - display default posts here,");
                  return mav;

        }

        @RequestMapping(value = "/pushRoomRentComments/{id}" )
        public ModelAndView pushRoomRentComments(@PathParam("comments") String
comments,@PathVariable("id") Double id,HttpSession session){
```

```
                    System.out.println(“ Push Room Rent comments to db” + comments);
                    System.out.println(“ ID from JSP” + id);
                    ModelAndView modelAndView = null;

                    if(comments.isEmpty()){
                              modelAndView = new ModelAndView(new
RedirectView(“/ScsuWebApp/roomRentPage”));
                              System.out.println(“Commments are blank”);
                              return modelAndView;
                    }
                    DBCollection coll = null;
                    BaseDao bd = new BaseDao();
                    coll =   bd.getConnection();


                    BasicDBObject query =  new BasicDBObject(“PostDoc,” “RoomRentPost”);
                    query.put(“RoomRentPost.postId,” id);

                    // get number of comments to append next comment
                    int n = getnumberOfRoomRentComments(query,coll,id);

                    //increment the comment to push as next comment number
                    n =n+1;
                    //Below code is to push the comment
                    BasicDBObject docToInsert = new BasicDBObject(“comment”+n,
comments);
                    docToInsert.put(“author,”session.getAttribute(“user”));

                    BasicDBObject updateCommand = new BasicDBObject(“$push,” new
BasicDBObject(“RoomRentPost.$.comments,” docToInsert));

                    coll.update(query, updateCommand);
                    System.out.println(coll.findOne().toString());


                    modelAndView = new ModelAndView(new
RedirectView(“/ScsuWebApp/roomRentPage”));

                    return modelAndView;
          }

          //returns the number of comments in a post
                    private int getnumberOfRoomRentComments(BasicDBObject
query,DBCollection coll, Double id){

                              DBCursor cursor = coll.find(query);
                              DBObject next = cursor.next();
                              BasicDBList listEditions = (BasicDBList)next.get("RoomRentPost");
                              System.out.println(listEditions.toString());
                              System.out.println("size : + " + listEditions.size());
                               BasicDBList comments =null;
                              for(Object element: listEditions) {
                                 //single strings
                                        double postId =
(Double)((BasicDBObject)element).get("postId");
                                         System.out.println("Post id :" + postId + " id :" + id );
                                        // if postid = id, get the number of comments of specific post id
                                        if(postId == id){
                                                    comments =
(BasicDBList)((BasicDBObject)element).get("comments");
                                                    System.out.println("element get comments" +
comments.toString() + "coments size :" + comments.size());
                                        }

                              }
                              return comments.size();
                    }


                    @RequestMapping(value = "/roomRentPost" )
                    public ModelAndView newRoomRentPost(@RequestParam("roomRentPost")
String newPost,HttpSession session){

                              System.out.println("New post -----------" + newPost);
                              //first get number of post in the room post and push the new post  with
next post id
                              BasicDBObject query =  new BasicDBObject("PostDoc,"
"RoomRentPost");//TODO - user name from session
                              BaseDao bd = new BaseDao();
                              DBCollection coll =    bd.getConnection();
                              DBCursor cursor = coll.find(query);
                              DBObject next = cursor.next();
                              BasicDBList postList = (BasicDBList)next.get("RoomRentPost");

                              System.out.println("Number of roomRentPost  : " + postList.size());

                              Double nextPost = (double) (postList.size() + 1);

                              //push new post with post id as nextPost
                    //        query.put("RoomPost.postId," nextPost);

                              List<DBObject> comments = new ArrayList<DBObject>();
```

```
                        /* comments.add(new BasicDBObject("Author Ashish 1,""second"));
                         comments.add(new BasicDBObject("Author Ashish 2,""third"));*/

                        //Below code is to push the post
                        BasicDBObject docToInsert = new BasicDBObject("postId,"
nextPost);
                        docToInsert.put("postAuthor," session.getAttribute("user"));//TODO -
user name from session
                        docToInsert.put("title," newPost);
                        docToInsert.put("comments," comments);

                        BasicDBObject updateCommand = new BasicDBObject("$push," new
BasicDBObject("RoomRentPost," docToInsert));
                        coll.update(query, updateCommand);

                        ModelAndView modelAndView = new ModelAndView(new
RedirectView("/ScsuWebApp/roomRentPage"));

                        return modelAndView;
                }


                // ----------------Below code is for room to share--------

                @RequestMapping(value = "/roomSharePage" )
                public ModelAndView retrieveRoomShareData(){

                        System.out.println("roomSharePage Success");
                        List<PostTo> roomPostList = new ArrayList<PostTo>();


                        Double postId=null ;
                         String title = "";
                         String postAuthor = "";
                        DBCollection coll = null;
                        BaseDao bd = new BaseDao();
                        coll =   bd.getConnection();
                        BasicDBObject query =  new BasicDBObject("PostDoc,"
"RoomSharePost");       //new BasicDBObject("Post.postId," "room");//
                        //query.put("Post.postId," "room");
                        DBCursor cursor = coll.find();
                        cursor = coll.find(query);

                        DBObject next = cursor.next();
                        BasicDBList listEditions = (BasicDBList)next.get("RoomSharePost");
                        System.out.println(listEditions.toString());
                        System.out.println("size : + " + listEditions.size());
```

```
                              for(Object element: listEditions) {
System.out.println("****************************************************************"
);
                                PostTo roomPost = new PostTo();
                                //single strings
                                        postId = (Double)((BasicDBObject)element).get("postId");
                                        title = (String)((BasicDBObject)element).get("title");
                                        postAuthor =
(String)((BasicDBObject)element).get("postAuthor");

                                        System.out.println("postID" + postId);
                                        System.out.println("title : "+title);
                                        System.out.println("postAuthor : "+postAuthor);

                                        roomPost.setPostTitle(title);
                                        roomPost.setPostId(postId);
                                        roomPost.setPostAuthor(postAuthor);
                                        //Get comments
                               BasicDBList comments =
(BasicDBList)((BasicDBObject)element).get("comments");
                               System.out.println("element get comments" + comments.toString());
                               int i=1;
                               List<String> commentList = new ArrayList<String>();


                               //Gets induvidual comments from list of comments
                               for(Object comment: comments) {//unable to get single comment
                                         List<CommentAuthor> advancedCommentList = new
ArrayList<CommentAuthor>();
                                         CommentAuthor ca = new CommentAuthor();
                                        //  CommentAuthor ca = new CommentAuthor();
                                         String comment1 =
(String)((BasicDBObject)comment).get("comment"+i);
                                         commentList.add(comment1);

                                       //   advancedCommentList.add(comment1);
                                        String author =
(String)((BasicDBObject)comment).get("author");
                                        System.out.println(" Author : " + author);
                                         System.out.println("comment1 in looop : "+ comment1);
                                         i++;
                                         /*roomPost.setPostAuthor(author);
                                         roomPost.setComments(commentList);*/
                                         ca.setComment(comment1);
                                         ca.setCommentAuthor(author);
                                         advancedCommentList.add(ca);
roomPost.getAdvancedComments().addAll(advancedCommentList);
                                }

                                        roomPostList.add(roomPost);
                                }
                                /* roomPost.setComments(commentList);
                                 roomPostList.add(roomPost);*/
                                 ModelAndView mav = new ModelAndView("roomSharePage");
                            mav.addObject("postList,"roomPostList);
                            mav.addObject("defaultsPosts,""TODO - display default posts here,");
                            return mav;

                        }

                        @RequestMapping(value = "/pushRoomShareComments/{id}" )
                        public ModelAndView
pushRoomShareComments(@PathParam("comments") String
comments,@PathVariable("id") Double id,HttpSession session){

                                System.out.println(" Push Room share comments to db" + comments);
                                System.out.println(" ID from JSP" + id);
                                ModelAndView modelAndView = null;

                                if(comments.isEmpty()){
                                        modelAndView = new ModelAndView(new
RedirectView("/ScsuWebApp/roomSharePage"));
                                        System.out.println("Commments are blank");
                                        return modelAndView;
                                }
                                DBCollection coll = null;
                                BaseDao bd = new BaseDao();
                                coll =   bd.getConnection();

                                BasicDBObject query =  new BasicDBObject("PostDoc,"
"RoomSharePost");
                                query.put("RoomSharePost.postId," id);

                                // get number of comments to append next comment
                                int n = getnumberOfRoomShareComments(query,coll,id);

                                //increment the comment to push as next comment number
                                n =n+1;
                                //Below code is to push the comment
                                BasicDBObject docToInsert = new BasicDBObject("comment"+n,
comments);
                                docToInsert.put("author," session.getAttribute("user"));
```

```
				BasicDBObject updateCommand = new BasicDBObject(“$push,” new
BasicDBObject(“RoomSharePost.$.comments,” docToInsert));

				coll.update(query, updateCommand);
				System.out.println(coll.findOne().toString());


				modelAndView = new ModelAndView(new
RedirectView(“/ScsuWebApp/roomSharePage”));

				return modelAndView;
			}

			//returns the number of comments in a post
				private int getnumberOfRoomShareComments(BasicDBObject
query,DBCollection coll, Double id){

					DBCursor cursor = coll.find(query);

					DBObject next = cursor.next();
					BasicDBList listEditions =
(BasicDBList)next.get(“RoomSharePost”);
					System.out.println(listEditions.toString());
					System.out.println(“size : + “ + listEditions.size());
					 BasicDBList comments =null;
					for(Object element: listEditions) {
					   //single strings
						double postId =
(Double)((BasicDBObject)element).get(“postId”);
						 System.out.println(“Post id :” + postId + “ id :” + id );
						// if postid = id, get the number of comments of specific
post id
						if(postId == id){
							comments =
(BasicDBList)((BasicDBObject)element).get(“comments”);
							System.out.println(“element get comments” +
comments.toString() + “coments size :” + comments.size());
						}

					}
					return comments.size();
				}

				@RequestMapping(value = “/roomSharePost” )
				public ModelAndView
newRoomSharePost(@RequestParam(“roomSharePost”) String newPost,HttpSession
session){
				System.out.println(“New post -----------” + newPost);
				//first get number of post in the room post and push the new
post  with next post id
				BasicDBObject query =  new BasicDBObject(“PostDoc,”
“RoomSharePost”);//TODO - user name from session
				BaseDao bd = new BaseDao();
				DBCollection coll =    bd.getConnection();
				DBCursor cursor = coll.find(query);
				DBObject next = cursor.next();
				BasicDBList postList =
(BasicDBList)next.get(“RoomSharePost”);

				System.out.println(“Number of roomSharePost  : “ +
postList.size());

				Double nextPost = (double) (postList.size() + 1);

				//push new post with post id as nextPost
			//	query.put(“RoomPost.postId,” nextPost);

				List<DBObject> comments = new ArrayList<DBObject>();
				/* comments.add(new BasicDBObject(“Author Ashish
1,”“second”));
				 comments.add(new BasicDBObject(“Author Ashish
2,”“third”));*/
				//Below code is to push the post
				BasicDBObject docToInsert = new BasicDBObject(“postId,”
nextPost);
				docToInsert.put(“postAuthor,”
session.getAttribute(“user”));//TODO - user name from session
				docToInsert.put(“title,” newPost);
				docToInsert.put(“comments,” comments);

				BasicDBObject updateCommand = new
BasicDBObject(“$push,” new BasicDBObject(“RoomSharePost,” docToInsert));
				coll.update(query, updateCommand);

				ModelAndView modelAndView = new ModelAndView(new
RedirectView(“/ScsuWebApp/roomSharePage”));

				return modelAndView;
			}

}
```

CarPoolingController

```
package com.scsu.controller;

import java.util.ArrayList;
import java.util.List;

import javax.servlet.http.HttpSession;
import javax.websocket.server.PathParam;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.view.RedirectView;

import com.mongodb.BasicDBList;
import com.mongodb.BasicDBObject;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.scsu.dao.BaseDao;
import com.scsu.postTo.CommentAuthor;
import com.scsu.postTo.PostTo;

@Controller
public class CarPoolingController {



        /**Shows the default posts when Car pool link is clicked.
         * @return
         * @throws Exception
         */
        @RequestMapping(value = "/carPoolPostPage" ,method=RequestMethod.GET)
   public ModelAndView retriveData() throws Exception {
                System.out.println("--------------Retrieve CarPoolPost Data---------");
                List<PostTo> carPoolPostList = new ArrayList<PostTo>();


                Double postId=null ;
                 String title = "";
                 String postAuthor = "";
                DBCollection coll = null;
                BaseDao bd = new BaseDao();
                coll =   bd.getConnection();
                BasicDBObject query =  new BasicDBObject(“PostDoc,” “CarPool”);
        //new BasicDBObject(“Post.postId,” “room”);//
                //query.put(“Post.postId,” “room”);
                DBCursor cursor = coll.find();
                cursor = coll.find(query);

                DBObject next = cursor.next();
                BasicDBList listEditions = (BasicDBList)next.get(“CarPoolPost”);
                System.out.println(listEditions.toString());
                System.out.println(“size : + “ + listEditions.size());
                for(Object element: listEditions) {

System.out.println(“**************************************************************”
);
                   PostTo roomPost = new PostTo();
                   //single strings
                         postId = (Double)((BasicDBObject)element).get(“postId”);
                         title = (String)((BasicDBObject)element).get(“title”);
                         postAuthor = (String)((BasicDBObject)element).get(“postAuthor”);

                         System.out.println(“postID” + postId);
                         System.out.println(“title : “+title);
                         System.out.println(“postAuthor : “+postAuthor);

                         roomPost.setPostTitle(title);
                         roomPost.setPostId(postId);
                         roomPost.setPostAuthor(postAuthor);
                         //Get comments
                  BasicDBList comments =
(BasicDBList)((BasicDBObject)element).get(“comments”);
                  System.out.println(“element get comments” + comments.toString());
                  int i=1;
                  List<String> commentList = new ArrayList<String>();


                  //Gets induvidual comments from list of comments
                  for(Object comment: comments) {//unable to get single comment
                          List<CommentAuthor> advancedCommentList = new
ArrayList<CommentAuthor>();
                          CommentAuthor ca = new CommentAuthor();
                         //  CommentAuthor ca = new CommentAuthor();
                          String comment1 =
(String)((BasicDBObject)comment).get(“comment”+i);
                          commentList.add(comment1);
```

```
                    //   advancedCommentList.add(comment1);
                      String author = (String)((BasicDBObject)comment).get("author");
                      System.out.println(" Author : " + author);
                       System.out.println("comment1 in looop : "+ comment1);
                       i++;
                       /*roomPost.setPostAuthor(author);
                       roomPost.setComments(commentList);*/
                       ca.setComment(comment1);
                       ca.setCommentAuthor(author);
                       advancedCommentList.add(ca);
                       roomPost.getAdvancedComments().addAll(advancedCommentList);
                 }

                     carPoolPostList.add(roomPost);
               }
               /* roomPost.setComments(commentList);
                roomPost.List.add(roomPost);*/
                ModelAndView mav = new ModelAndView("carPoolPostPage");
             mav.addObject("postList,"carPoolPostList);
             mav.addObject("defaultsPosts,""TODO - display default posts here,");
             return mav;
   }

        /**Push the comments entered in the text box and refreshes the page.
         * @param comments
         * @param id
         * @return
         */
        @RequestMapping(value = "/pushCarPoolComments/{id}" )
        public ModelAndView pushComments(@PathParam("comments") String
comments,@PathVariable("id") Double id,HttpSession session){

                System.out.println(" Push the comments to db" + comments);
                System.out.println(" ID from JSP" + id);
                ModelAndView modelAndView = null;

                if(comments.isEmpty()){
                        modelAndView = new ModelAndView(new
RedirectView("/ScsuWebApp/carPoolPostPage"));
                        System.out.println("Commmments are blank");
                        return modelAndView;
                }
                DBCollection coll = null;
                BaseDao bd = new BaseDao();
                coll =   bd.getConnection();
                BasicDBObject query =  new BasicDBObject(“PostDoc,” “CarPool”);
                query.put(“CarPoolPost.postId,” id);

                // get number of comments to append next comment
                int n = getnumberOfComments(query,coll,id);

                //increment the comment to push as next comment number
                n =n+1;
                //Below code is to push the comment
                BasicDBObject docToInsert = new BasicDBObject(“comment”+n,
comments);
                docToInsert.put(“author,” session.getAttribute(“user”));

                BasicDBObject updateCommand = new BasicDBObject(“$push,” new
BasicDBObject(“CarPoolPost.$.comments,” docToInsert));

                coll.update(query, updateCommand);
                System.out.println(coll.findOne().toString());


                modelAndView = new ModelAndView(new
RedirectView(“/ScsuWebApp/carPoolPostPage”));

                return modelAndView;
        }



        //returns the number of comments in a post
                private int getnumberOfComments(BasicDBObject query,DBCollection coll,
Double id){

                        DBCursor cursor = coll.find(query);

                        DBObject next = cursor.next();
                        BasicDBList listEditions = (BasicDBList)next.get(“CarPoolPost”);
                        System.out.println(listEditions.toString());
                        System.out.println(“size : + “ + listEditions.size());
                         BasicDBList comments =null;
                        for(Object element: listEditions) {
                           //single strings
                                double postId =
(Double)((BasicDBObject)element).get(“postId”);
                                 System.out.println(“Post id :” + postId + “ id :” + id );
                                // if postid = id, get the number of comments of specific post id
                                if(postId == id){
                                        comments =
(BasicDBList)((BasicDBObject)element).get(“comments”);
```

```
                                        System.out.println("element get comments" +
comments.toString() + "coments size :" + comments.size());
                                }

                        }
                        return comments.size();
                }


                @RequestMapping(value = "/carPoolNewPost" )
                public ModelAndView newCarPoolPost(@RequestParam("carPoolPost")
String newPost,HttpSession session){

                        System.out.println("New post -----------" + newPost);
                        //first get number of post in the room post and push the new post  with
next post id
                        BasicDBObject query =  new BasicDBObject("PostDoc," "CarPool");
                        BaseDao bd = new BaseDao();
                        DBCollection coll =    bd.getConnection();
                        DBCursor cursor = coll.find(query);
                        DBObject next = cursor.next();
                        BasicDBList postList = (BasicDBList)next.get("CarPoolPost");

                        System.out.println("Number of posts : " + postList.size());

                        Double nextPost = (double) (postList.size() + 1);

                        //push new post with post id as nextPost
                //      query.put("RoomPost.postId," nextPost);

                        List<DBObject> comments = new ArrayList<DBObject>();
                        /* comments.add(new BasicDBObject("Author Ashish 1,""second"));
                         comments.add(new BasicDBObject("Author Ashish 2,""third"));*/

                        //Below code is to push the post
                        BasicDBObject docToInsert = new BasicDBObject("postId,"
nextPost);
                        docToInsert.put("postAuthor," session.getAttribute("user"));
                        docToInsert.put("title," newPost);
                        docToInsert.put("comments," comments);

                        BasicDBObject updateCommand = new BasicDBObject("$push," new
BasicDBObject("CarPoolPost," docToInsert));
                        coll.update(query, updateCommand);

                        ModelAndView modelAndView = new ModelAndView(new
RedirectView("/ScsuWebApp/carPoolPostPage"));
```

```
                        return modelAndView;
                }

                //-------------Below Code is for Car Rent Post--------------------------------

                @RequestMapping(value = "/carRentPage" ,method=RequestMethod.GET)
            public ModelAndView retriveCarRentData() throws Exception {
                        System.out.println("-------------Retrieve carRentPostPage Data---------
");
                        List<PostTo> carPoolPostList = new ArrayList<PostTo>();



                        Double postId=null ;
                         String title = "";
                         String postAuthor = "";
                        DBCollection coll = null;
                        BaseDao bd = new BaseDao();
                        coll =   bd.getConnection();
                        BasicDBObject query =  new BasicDBObject("PostDoc," "CarRent");

                        //query.put("Post.postId," "room");
                        DBCursor cursor = coll.find();
                        cursor = coll.find(query);

                        DBObject next = cursor.next();
                        BasicDBList listEditions = (BasicDBList)next.get("CarRentPost");
                        System.out.println(listEditions.toString());
                        System.out.println("size : + " + listEditions.size());
                        for(Object element: listEditions) {

System.out.println("*********************************************************"
);
                          PostTo roomPost = new PostTo();
                          //single strings
                                postId = (Double)((BasicDBObject)element).get("postId");
                                title = (String)((BasicDBObject)element).get("title");
                                postAuthor =
(String)((BasicDBObject)element).get("postAuthor");

                                System.out.println("postID" + postId);
                                System.out.println("title : "+title);
                                System.out.println("postAuthor : "+postAuthor);

                                roomPost.setPostTitle(title);
                                roomPost.setPostId(postId);
                                roomPost.setPostAuthor(postAuthor);
```

```
                                        //Get comments
                                BasicDBList comments =
(BasicDBList)((BasicDBObject)element).get("comments");
                                System.out.println("element get comments" + comments.toString());
                                int i=1;
                                List<String> commentList = new ArrayList<String>();


                                //Gets induvidual comments from list of comments
                                for(Object comment: comments) {//unable to get single comment
                                        List<CommentAuthor> advancedCommentList = new
ArrayList<CommentAuthor>();
                                        CommentAuthor ca = new CommentAuthor();
                                       //  CommentAuthor ca = new CommentAuthor();
                                        String comment1 =
(String)((BasicDBObject)comment).get("comment"+i);
                                        commentList.add(comment1);

                                      //   advancedCommentList.add(comment1);
                                       String author =
(String)((BasicDBObject)comment).get("author");
                                       System.out.println(" Author : " + author);
                                        System.out.println("comment1 in looop : "+ comment1);
                                        i++;
                                        /*roomPost.setPostAuthor(author);
                                        roomPost.setComments(commentList);*/
                                        ca.setComment(comment1);
                                        ca.setCommentAuthor(author);
                                        advancedCommentList.add(ca);

roomPost.getAdvancedComments().addAll(advancedCommentList);
                                }

                                        carPoolPostList.add(roomPost);
                              }
                              /* roomPost.setComments(commentList);
                               roomPostList.add(roomPost);*/
                               ModelAndView mav = new ModelAndView("carRentPage");
                          mav.addObject("postList,"carPoolPostList);
                          mav.addObject("defaultsPosts,""TODO - display default posts here,");
                          return mav;
        }

                /**Push the comments entered in the text box and refreshes the page.
                 * @param comments
                 * @param id
                 * @return
```

```
		 */
		@RequestMapping(value = "/pushCarRentComments/{id}" )
		public ModelAndView pushCarRentComments(@PathParam("comments")
String comments,@PathVariable("id") Double id,HttpSession session){

			System.out.println(" Push the comments to db" + comments);
			System.out.println(" ID from JSP" + id);
			ModelAndView modelAndView = null;

			if(comments.isEmpty()){
				modelAndView = new ModelAndView(new
RedirectView("/ScsuWebApp/carRentPage"));
				System.out.println("Commments are blank");
				return modelAndView;
			}
			DBCollection coll = null;
			BaseDao bd = new BaseDao();
			coll =   bd.getConnection();


			BasicDBObject query =  new BasicDBObject("PostDoc," "CarRent");
			query.put("CarRentPost.postId," id);

			// get number of comments to append next comment
			int n = getnumberOfCarRentComments(query,coll,id);

			//increment the comment to push as next comment number
			n =n+1;
			//Below code is to push the comment
			BasicDBObject docToInsert = new BasicDBObject("comment"+n,
comments);
			docToInsert.put("author," session.getAttribute("user"));

			BasicDBObject updateCommand = new BasicDBObject("$push," new
BasicDBObject("CarRentPost.$.comments," docToInsert));

			coll.update(query, updateCommand);
			System.out.println(coll.findOne().toString());


			modelAndView = new ModelAndView(new
RedirectView("/ScsuWebApp/carRentPage"));

			return modelAndView;
		}
		//returns the number of comments in a post
			private int getnumberOfCarRentComments(BasicDBObject
query,DBCollection coll, Double id){

				DBCursor cursor = coll.find(query);

				DBObject next = cursor.next();
				BasicDBList listEditions =
(BasicDBList)next.get("CarRentPost");
				System.out.println(listEditions.toString());
				System.out.println("size : + " + listEditions.size());
				 BasicDBList comments =null;
				for(Object element: listEditions) {
				   //single strings
					double postId =
(Double)((BasicDBObject)element).get("postId");
					 System.out.println("Post id :" + postId + " id :" + id );
					// if postid = id, get the number of comments of specific
post id
					if(postId == id){
						comments =
(BasicDBList)((BasicDBObject)element).get("comments");
						System.out.println("element get comments" +
comments.toString() + "coments size :" + comments.size());
					}

				}
				return comments.size();
			}


			@RequestMapping(value = "/carRentNewPost" )
			public ModelAndView
newCarRentPost(@RequestParam("carRentPost") String newPost,HttpSession session){

				System.out.println("New post -----------" + newPost);
				//first get number of post in the room post and push the new
post  with next post id
				BasicDBObject query =  new BasicDBObject("PostDoc,"
"CarRent");
				BaseDao bd = new BaseDao();
				DBCollection coll =   bd.getConnection();
				DBCursor cursor = coll.find(query);
				DBObject next = cursor.next();
				BasicDBList postList =
(BasicDBList)next.get("CarRentPost");
```

```
                    System.out.println("Number of posts : " + postList.size());

                    Double nextPost = (double) (postList.size() + 1);

                    //push new post with post id as nextPost
            //      query.put("RoomPost.postId," nextPost);

                    List<DBObject> comments = new ArrayList<DBObject>();
                    /* comments.add(new BasicDBObject("Author Ashish
1,""second"));
                     comments.add(new BasicDBObject("Author Ashish
2,""third"));*/

                    //Below code is to push the post
                    BasicDBObject docToInsert = new BasicDBObject("postId,"
nextPost);
                    docToInsert.put("postAuthor,"session.getAttribute("user"));
                    docToInsert.put("title," newPost);
                    docToInsert.put("comments," comments);

                    BasicDBObject updateCommand = new
BasicDBObject("$push," new BasicDBObject("CarRentPost," docToInsert));
                    coll.update(query, updateCommand);

                    ModelAndView modelAndView = new ModelAndView(new
RedirectView("/ScsuWebApp/carRentPage"));

                    return modelAndView;
            }
}
```

KnowledgeController

```
package com.scsu.controller;

import java.util.ArrayList;
import java.util.List;

import javax.servlet.http.HttpSession;
import javax.websocket.server.PathParam;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.view.RedirectView;

import com.mongodb.BasicDBList;
import com.mongodb.BasicDBObject;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.scsu.dao.BaseDao;
import com.scsu.postTo.CommentAuthor;
import com.scsu.postTo.PostTo;

@Controller
public class KnowledgeController {


        /**Shows the default posts when Knowledge link is clicked.
         * @return
         * @throws Exception
         */
        @RequestMapping(value = "/studyPostPage" ,method=RequestMethod.GET)
  public ModelAndView retriveData() throws Exception {
                System.out.println("--------------Retrieve studyPost Data---------");
                List<PostTo> carPoolPostList = new ArrayList<PostTo>();


                Double postId=null ;
                 String title = "";
                 String postAuthor = "";
                DBCollection coll = null;
                BaseDao bd = new BaseDao();
                coll =   bd.getConnection();
                BasicDBObject query =  new BasicDBObject(“PostDoc,” “StudyPost”);
        //new BasicDBObject(“Post.postId,” “room”);//
                //query.put(“Post.postId,” “room”);
                DBCursor cursor = coll.find();
                cursor = coll.find(query);

                DBObject next = cursor.next();
                BasicDBList listEditions = (BasicDBList)next.get(“StudyPostData”);
                System.out.println(listEditions.toString());
                System.out.println(“size : + “ + listEditions.size());
                for(Object element: listEditions) {

System.out.println(“*******************************************************”
);
                  PostTo roomPost = new PostTo();
                  //single strings
                        postId = (Double)((BasicDBObject)element).get(“postId”);
                        title = (String)((BasicDBObject)element).get(“title”);
                        postAuthor = (String)((BasicDBObject)element).get(“postAuthor”);

                        System.out.println(“postID” + postId);
                        System.out.println(“title : “+title);
                        System.out.println(“postAuthor : “+postAuthor);

                        roomPost.setPostTitle(title);
                        roomPost.setPostId(postId);
                        roomPost.setPostAuthor(postAuthor);
                        //Get comments
                  BasicDBList comments =
(BasicDBList)((BasicDBObject)element).get(“comments”);
                  System.out.println(“element get comments” + comments.toString());
                  int i=1;
                  List<String> commentList = new ArrayList<String>();


                  //Gets induvidual comments from list of comments
                  for(Object comment: comments) {//unable to get single comment
                         List<CommentAuthor> advancedCommentList = new
ArrayList<CommentAuthor>();
                         CommentAuthor ca = new CommentAuthor();
                        //  CommentAuthor ca = new CommentAuthor();
                         String comment1 =
(String)((BasicDBObject)comment).get(“comment”+i);
                         commentList.add(comment1);

                       //   advancedCommentList.add(comment1);
```

```
                        String author = (String)((BasicDBObject)comment).get("author");
                        System.out.println(" Author : " + author);
                         System.out.println("comment1 in looop : "+ comment1);
                         i++;
                         /*roomPost.setPostAuthor(author);
                         roomPost.setComments(commentList);*/
                         ca.setComment(comment1);
                         ca.setCommentAuthor(author);
                         advancedCommentList.add(ca);
                         roomPost.getAdvancedComments().addAll(advancedCommentList);
                }

                        carPoolPostList.add(roomPost);
               }
               /* roomPost.setComments(commentList);
                roomPostList.add(roomPost);*/
                ModelAndView mav = new ModelAndView("studyPostPage");
             mav.addObject("postList,"carPoolPostList);
             mav.addObject("defaultsPosts,""TODO - display default posts here,");
             return mav;
   }


        /**Push the comments entered in the text box and refreshes the page.
         * @param comments
         * @param id
         * @return
         */
        @RequestMapping(value = "/pushStudyPostComments/{id}" )
        public ModelAndView pushComments(@PathParam("comments") String
comments,@PathVariable("id") Double id,HttpSession session){

                System.out.println(" Push the comments to db" + comments);
                System.out.println(" ID from JSP" + id);
                ModelAndView modelAndView = null;

                if(comments.isEmpty()){
                        modelAndView = new ModelAndView(new
RedirectView("/ScsuWebApp/studyPostPage"));
                        System.out.println("Commments are blank");
                        return modelAndView;
                }
                DBCollection coll = null;
                BaseDao bd = new BaseDao();
                coll =   bd.getConnection();
```

```
                    BasicDBObject query =  new BasicDBObject("PostDoc," "StudyPost");
                    query.put("StudyPostData.postId," id);

                    // get number of comments to append next comment
                    int n = getnumberOfComments(query,coll,id);

                    //increment the comment to push as next comment number
                    n =n+1;
                    //Below code is to push the comment
                    BasicDBObject docToInsert = new BasicDBObject("comment"+n,
comments);
                    docToInsert.put("author," session.getAttribute("user"));

                    BasicDBObject updateCommand = new BasicDBObject("$push," new
BasicDBObject("StudyPostData.$.comments," docToInsert));

                    coll.update(query, updateCommand);
                    System.out.println(coll.findOne().toString());



                    modelAndView = new ModelAndView(new
RedirectView("/ScsuWebApp/studyPostPage"));

                    return modelAndView;
          }

          //returns the number of comments in a post
                         private int getnumberOfComments(BasicDBObject
query,DBCollection coll, Double id){

                                        DBCursor cursor = coll.find(query);

                                        DBObject next = cursor.next();
                                        BasicDBList listEditions =
(BasicDBList)next.get("StudyPostData");
                                        System.out.println(listEditions.toString());
                                        System.out.println("size : + " + listEditions.size());
                                         BasicDBList comments =null;
                                        for(Object element: listEditions) {
                                           //single strings
                                             double postId =
(Double)((BasicDBObject)element).get("postId");
                                              System.out.println("Post id :" + postId + " id :" + id );
                                             // if postid = id, get the number of comments of specific
post id
                                             if(postId == id){
```

```
                                        comments =
(BasicDBList)((BasicDBObject)element).get(“comments”);
                                        System.out.println(“element get comments” +
comments.toString() + “coments size :” + comments.size());
                                }

                        }
                        return comments.size();
                }

                @RequestMapping(value = “/studyNewPost” )
                public ModelAndView newstudyPost(@RequestParam(“studyPost”)
String newPost,HttpSession session){

                        System.out.println(“New post -----------” + newPost);
                        //first get number of post in the “StudyPost” and push the new
post  with next post id
                        BasicDBObject query =  new BasicDBObject(“PostDoc,”
“StudyPost”);
                        BaseDao bd = new BaseDao();
                        DBCollection coll =   bd.getConnection();
                        DBCursor cursor = coll.find(query);
                        DBObject next = cursor.next();
                        BasicDBList postList =
(BasicDBList)next.get(“StudyPostData”);

                        System.out.println(“Number of posts : “ + postList.size());

                        Double nextPost = (double) (postList.size() + 1);

                        //push new post with post id as nextPost
                //      query.put(“RoomPost.postId,” nextPost);

                        List<DBObject> comments = new ArrayList<DBObject>();
                        /* comments.add(new BasicDBObject(“Author Ashish
1,”“second”));
                         comments.add(new BasicDBObject(“Author Ashish
2,”“third”));*/

                        //Below code is to push the post
                        BasicDBObject docToInsert = new BasicDBObject(“postId,”
nextPost);
                        docToInsert.put(“postAuthor,” session.getAttribute(“user”));
                        docToInsert.put(“title,” newPost);
                        docToInsert.put(“comments,” comments);
```

```
                                BasicDBObject updateCommand = new
BasicDBObject(“$push,” new BasicDBObject(“StudyPostData,” docToInsert));
                                coll.update(query, updateCommand);

                                ModelAndView modelAndView = new ModelAndView(new
RedirectView(“/ScsuWebApp/studyPostPage”));

                                return modelAndView;
                        }

        //----------------Below code is for Job Postings------------------------------


                        @RequestMapping(value = “/jobPostPage”
,method=RequestMethod.GET)
                    public ModelAndView retriveJobPostData() throws Exception {
                                System.out.println(“--------------Retrieve JobPost Data---------
”);
                                List<PostTo> jobPostList = new ArrayList<PostTo>();


                                Double postId=null ;
                                 String title = ““;
                                 String postAuthor = ““;
                                DBCollection coll = null;
                                BaseDao bd = new BaseDao();
                                coll =   bd.getConnection();
                                BasicDBObject query =  new BasicDBObject(“PostDoc,”
“JobPost”);
                                DBCursor cursor = coll.find();
                                cursor = coll.find(query);

                                DBObject next = cursor.next();
                                BasicDBList listEditions =
(BasicDBList)next.get(“JobPostData”);
                                System.out.println(listEditions.toString());
                                System.out.println(“size : + “ + listEditions.size());
                                for(Object element: listEditions) {

System.out.println(“*******************************************************”
);
                                  PostTo roomPost = new PostTo();
                                  //single strings
                                      postId =
(Double)((BasicDBObject)element).get(“postId”);
                                      title = (String)((BasicDBObject)element).get(“title”);
```

```
                                        postAuthor =
(String)((BasicDBObject)element).get("postAuthor");

                                        System.out.println("postID" + postId);
                                        System.out.println("title : "+title);
                                        System.out.println("postAuthor : "+postAuthor);

                                        roomPost.setPostTitle(title);
                                        roomPost.setPostId(postId);
                                        roomPost.setPostAuthor(postAuthor);
                                        //Get comments
                                BasicDBList comments =
(BasicDBList)((BasicDBObject)element).get("comments");
                                System.out.println("element get comments" +
comments.toString());
                                int i=1;
                                List<String> commentList = new ArrayList<String>();


                                //Gets induvidual comments from list of comments
                                for(Object comment: comments) {//unable to get single
comment
                                         List<CommentAuthor> advancedCommentList = new
ArrayList<CommentAuthor>();
                                         CommentAuthor ca = new CommentAuthor();
                                        //  CommentAuthor ca = new CommentAuthor();
                                         String comment1 =
(String)((BasicDBObject)comment).get("comment"+i);
                                         commentList.add(comment1);

                                       //   advancedCommentList.add(comment1);
                                        String author =
(String)((BasicDBObject)comment).get("author");
                                        System.out.println(" Author : " + author);
                                         System.out.println("comment1 in looop : "+
comment1);
                                         i++;
                                         /*roomPost.setPostAuthor(author);
                                         roomPost.setComments(commentList);*/
                                         ca.setComment(comment1);
                                         ca.setCommentAuthor(author);
                                         advancedCommentList.add(ca);

roomPost.getAdvancedComments().addAll(advancedCommentList);
                                }

                                        jobPostList.add(roomPost);
                                        }
                                        /* roomPost.setComments(commentList);
                                         roomPostList.add(roomPost);*/
                                         ModelAndView mav = new ModelAndView("jobPostPage");
                                    mav.addObject("postList,"jobPostList);
                                    mav.addObject("defaultsPosts,""TODO - display default posts
here,");
                                    return mav;
                        }


                              /**Push the comments entered in the text box and refreshes the page.
                               * @param comments
                               * @param id
                               * @return
                               */
                              @RequestMapping(value = "/pushJobPostComments/{id}" )
                              public ModelAndView
pushJobPostComments(@PathParam("comments") String comments,@PathVariable("id")
Double id,HttpSession session){

                                        System.out.println(" Push the comments to db" + comments);
                                        System.out.println(" ID from JSP" + id);
                                        ModelAndView modelAndView = null;

                                        if(comments.isEmpty()){
                                                  modelAndView = new ModelAndView(new
RedirectView("/ScsuWebApp/jobPostPage"));
                                                  System.out.println("Commments are blank");
                                                  return modelAndView;
                                        }
                                        DBCollection coll = null;
                                        BaseDao bd = new BaseDao();
                                        coll =   bd.getConnection();



                                        BasicDBObject query =  new BasicDBObject("PostDoc,"
"JobPost");
                                        query.put("JobPostData.postId," id);

                                        // get number of comments to append next comment
                                        int n = getnumberOfJobPostComments(query,coll,id);

                                        //increment the comment to push as next comment number
                                        n =n+1;
                                        //Below code is to push the comment
                                        BasicDBObject docToInsert = new
BasicDBObject("comment"+n, comments);
                                        docToInsert.put("author,"session.getAttribute("user"));

                                        BasicDBObject updateCommand = new
BasicDBObject("$push," new BasicDBObject("JobPostData.$.comments," docToInsert));

                                        coll.update(query, updateCommand);
                                        System.out.println(coll.findOne().toString());


                                        modelAndView = new ModelAndView(new
RedirectView("/ScsuWebApp/jobPostPage"));

                                        return modelAndView;
                              }

                              //returns the number of comments in a post
                                                  private int
getnumberOfJobPostComments(BasicDBObject query,DBCollection coll, Double id){

                                                            DBCursor cursor = coll.find(query);

                                                            DBObject next = cursor.next();
                                                            BasicDBList listEditions =
(BasicDBList)next.get("JobPostData");
                                                            System.out.println(listEditions.toString());
                                                            System.out.println("size : + " +
listEditions.size());
                                                             BasicDBList comments =null;
                                                            for(Object element: listEditions) {
                                                               //single strings
                                                                      double postId =
(Double)((BasicDBObject)element).get("postId");
                                                                       System.out.println("Post id :" + postId +
" id :" + id );

                                                                      // if postid = id, get the number of
comments of specific post id
                                                                      if(postId == id){
                                                                                comments =
(BasicDBList)((BasicDBObject)element).get("comments");
                                                                                System.out.println("element get
comments" + comments.toString() + "coments size :" + comments.size());
                                                                      }

                                                            }
                                                            return comments.size();
```

```
					}

					@RequestMapping(value = “/jobNewPost” )
					public ModelAndView
newJobPost(@RequestParam(“jobPost”) String newPost,HttpSession session){

						System.out.println(“New post -----------” +
newPost);
						//first get number of post in the “StudyPost” and
push the new post  with next post id
						BasicDBObject query =  new
BasicDBObject(“PostDoc,” “JobPost”);
						BaseDao bd = new BaseDao();
						DBCollection coll =    bd.getConnection();

						DBCursor cursor = coll.find(query);
						DBObject next = cursor.next();
						BasicDBList postList =
(BasicDBList)next.get(“JobPostData”);

						System.out.println(“Number of posts : “ +
postList.size());

						Double nextPost = (double) (postList.size() + 1);

						List<DBObject> comments = new
ArrayList<DBObject>();

						//Below code is to push the post
						BasicDBObject docToInsert = new
BasicDBObject(“postId,” nextPost);
						docToInsert.put(“postAuthor,”
session.getAttribute(“user”));
						docToInsert.put(“title,” newPost);
						docToInsert.put(“comments,” comments);

						BasicDBObject updateCommand = new
BasicDBObject(“$push,” new BasicDBObject(“JobPostData,” docToInsert));
						coll.update(query, updateCommand);

						ModelAndView modelAndView = new
ModelAndView(new RedirectView(“/ScsuWebApp/jobPostPage”));

						return modelAndView;
					}

//---------------Below code is for Misc Posts-------------------------------
```

```
                              @RequestMapping(value = “/miscPostPage”
,method=RequestMethod.GET)
                          public ModelAndView retriveMiscPostData() throws
Exception {
                                    System.out.println(“--------------Retrieve
MiscPost Data---------”);
                                    List<PostTo> miscPostList = new
ArrayList<PostTo>();

                                    Double postId=null ;
                                     String title = ““;
                                     String postAuthor = ““;
                                    DBCollection coll = null;
                                    BaseDao bd = new BaseDao();
                                    coll =   bd.getConnection();
                                    BasicDBObject query =  new
BasicDBObject(“PostDoc,” “MiscPost”);
                                    DBCursor cursor = coll.find();
                                    cursor = coll.find(query);

                                    DBObject next = cursor.next();
                                    BasicDBList listEditions =
(BasicDBList)next.get(“MiscPostData”);
                                    System.out.println(listEditions.toString());
                                    System.out.println(“size : + “ +
listEditions.size());
                                    for(Object element: listEditions) {

System.out.println(“*****************************************************”
);
                                      PostTo roomPost = new PostTo();
                                      //single strings
                                          postId =
(Double)((BasicDBObject)element).get(“postId”);
                                          title =
(String)((BasicDBObject)element).get(“title”);
                                          postAuthor =
(String)((BasicDBObject)element).get(“postAuthor”);

                                          System.out.println(“postID” + postId);
                                          System.out.println(“title : “+title);
                                          System.out.println(“postAuthor :
“+postAuthor);

                                          roomPost.setPostTitle(title);
                                          roomPost.setPostId(postId);
```

```
                                              roomPost.setPostAuthor(postAuthor);
                                              //Get comments
                                          BasicDBList comments =
(BasicDBList)((BasicDBObject)element).get(“comments”);
                                          System.out.println(“element get comments” +
comments.toString());
                                          int i=1;
                                          List<String> commentList = new
ArrayList<String>();


                                          //Gets induvidual comments from list of
comments
                                          for(Object comment: comments) {//unable to
get single comment
                                               List<CommentAuthor>
advancedCommentList = new ArrayList<CommentAuthor>();
                                               CommentAuthor ca = new
CommentAuthor();
                                              //  CommentAuthor ca = new
CommentAuthor();
                                               String comment1 =
(String)((BasicDBObject)comment).get(“comment”+i);
                                               commentList.add(comment1);

                                             //
advancedCommentList.add(comment1);
                                              String author =
(String)((BasicDBObject)comment).get(“author”);
                                              System.out.println(“ Author : “ +
author);
                                               System.out.println(“comment1 in
looop : “+ comment1);
                                               i++;
                                               /*roomPost.setPostAuthor(author);

roomPost.setComments(commentList);*/
                                               ca.setComment(comment1);
                                               ca.setCommentAuthor(author);
                                               advancedCommentList.add(ca);

roomPost.getAdvancedComments().addAll(advancedCommentList);
                                          }

                                              miscPostList.add(roomPost);
                                         }
                                        /* roomPost.setComments(commentList);
                                        roomPostList.add(roomPost);*/
                                        ModelAndView mav = new
ModelAndView("miscPostPage");
                                      mav.addObject("postList,"miscPostList);
                                      mav.addObject("defaultsPosts,""TODO - display
default posts here,");
                                      return mav;
                              }


                                  /**Push the comments entered in the text box and
refreshes the page.
                                   * @param comments
                                   * @param id
                                   * @return
                                   */
                                  @RequestMapping(value =
"/pushMiscPostComments/{id}" )
                                  public ModelAndView
pushMiscPostComments(@PathParam("comments") String comments,@PathVariable("id")
Double id,HttpSession session){

                                        System.out.println(" Push the comments to db"
+ comments);
                                        System.out.println(" ID from JSP" + id);
                                        ModelAndView modelAndView = null;

                                        if(comments.isEmpty()){
                                             modelAndView = new
ModelAndView(new RedirectView("/ScsuWebApp/miscPostPage"));
                                             System.out.println("Commments are
blank");
                                             return modelAndView;
                                        }
                                        DBCollection coll = null;
                                        BaseDao bd = new BaseDao();
                                        coll =   bd.getConnection();



                                        BasicDBObject query =  new
BasicDBObject("PostDoc," "MiscPost");
                                        query.put("MiscPostData.postId," id);

                                        // get number of comments to append next
comment
                                        int n =
getnumberOfMiscPostComments(query,coll,id);
```

```
                                        //increment the comment to push as next
comment number
                                        n =n+1;
                                        //Below code is to push the comment
                                        BasicDBObject docToInsert = new
BasicDBObject("comment"+n, comments);
                                        docToInsert.put("author,"
session.getAttribute("user"));

                                        BasicDBObject updateCommand = new
BasicDBObject("$push," new BasicDBObject("MiscPostData.$.comments," docToInsert));

                                        coll.update(query, updateCommand);
                                        System.out.println(coll.findOne().toString());


                                        modelAndView = new ModelAndView(new
RedirectView("/ScsuWebApp/miscPostPage"));

                                        return modelAndView;
                                  }

                                  //returns the number of comments in a post
                                            private int
getnumberOfMiscPostComments(BasicDBObject query,DBCollection coll, Double id){

      DBCursor cursor = coll.find(query);

      DBObject next = cursor.next();
      BasicDBList listEditions = BasicDBList)next.get("MiscPostData");

      System.out.println(listEditions.toString());
      System.out.println("size : + " + listEditions.size());
       BasicDBList comments =null;
      for(Object element: listEditions) {
       //single strings
      double postId = (Double)((BasicDBObject)element).get("postId");
                                                            System.out.println("Post
id :" + postId + " id :" + id );
// if postid = id, get the number of comments of specific post id
if(postId == id){
 comments = (BasicDBList)((BasicDBObject)element).get("comments");

System.out.println("element get comments" + comments.toString() + "coments size :" +
comments.size());
                                                           }
```

```
				}
				return comments.size();
			}

			@RequestMapping(value =
"/miscNewPost" )
			public ModelAndView
newMiscPost(@RequestParam("miscPost") String newPost,HttpSession session){

				System.out.println("New post ----
-------" + newPost);
				//first get number of post in the
"StudyPost" and push the new post  with next post id
				BasicDBObject query =  new
BasicDBObject("PostDoc," "MiscPost");
				BaseDao bd = new BaseDao();
				DBCollection coll =
	bd.getConnection();
				DBCursor cursor =
coll.find(query);
				DBObject next = cursor.next();
				BasicDBList postList =
(BasicDBList)next.get("MiscPostData");

				System.out.println("Number of
posts : " + postList.size());

				Double nextPost = (double)
(postList.size() + 1);

				List<DBObject> comments =
new ArrayList<DBObject>();

//Below code is to push the post
BasicDBObject docToInsert = new BasicDBObject("postId," nextPost);
				docToInsert.put("postAuthor,"
session.getAttribute("user"));
	docToInsert.put("title," newPost);
				docToInsert.put("comments,"
comments);

				BasicDBObject updateCommand
= new BasicDBObject("$push," new BasicDBObject("MiscPostData," docToInsert));
				coll.update(query,
updateCommand);
```

```
                                                        ModelAndView modelAndView
= new ModelAndView(new RedirectView(“/ScsuWebApp/miscPostPage”));

                                                        return modelAndView;
                                                }

}
```

BaseDAO

```
package com.scsu.dao;

import org.bson.Document;
import org.springframework.data.mongodb.core.MongoTemplate;

import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.Mongo;
import com.mongodb.MongoClient;
import com.mongodb.client.MongoCollection;

public class BaseDao {

        private MongoTemplate mt;
        private Mongo mongo = null;
        private DB db = null;
        private MongoCollection<Document> collection;


        public  DBCollection getConnection() {
                /*MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
                 MongoDatabase db1 = mongoClient.getDatabase("ScsuDatabase");

                 collection = db1.getCollection("student");
                */

                MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
                DB db = mongoClient.getDB( "ScsuDatabase" );
                DBCollection coll = db.getCollection("student");
                // mongoClient.close();

                 return coll;
        }


        public  DB getDB() {
                /*MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
                 MongoDatabase db1 = mongoClient.getDatabase("ScsuDatabase");

                 collection = db1.getCollection("student");
                */

                MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
                DB db = mongoClient.getDB( "ScsuDatabase" );
```

```
                    DBCollection coll = db.getCollection(“student”);
                    // mongoClient.close();

                     return db;
          }
}
```

PostTo

```
package com.scsu.postTo;

import java.util.ArrayList;
import java.util.List;

public class PostTo {

	private String postTitle;
	private Double postId;
	private String postAuthor;
	private List<String> comments;
	private List<CommentAuthor> advancedComments;

	// getters and setters

	public String getPostTitle() {
		return postTitle;
	}
	public Double getPostId() {
		return postId;
	}
	public void setPostId(Double postId2) {
		this.postId = postId2;
	}
	public void setPostTitle(String postTitle) {
		this.postTitle = postTitle;
	}
	public List<String> getComments() {
		return comments;
	}
	public void setComments(List<String> comments) {
		this.comments = comments;
	}
	public String getPostAuthor() {
		return postAuthor;
	}
	public void setPostAuthor(String author) {
		this.postAuthor = author;
	}
	public List<CommentAuthor> getAdvancedComments() {
		if(this.advancedComments == null){
			this.advancedComments = new ArrayList<CommentAuthor>();
		}
		return advancedComments;
	}
	public void setAdvancedComments(List<CommentAuthor> advancedComments) {
		this.advancedComments = advancedComments;
	}

}

/*

 class CommentAuthor {
	private String comment;
	private String commentAuthor;
	public String getComment() {
		return comment;
	}
	public void setComment(String comment) {
		this.comment = comment;
	}
	public String getCommentAuthor() {
		return commentAuthor;
	}
	public void setCommentAuthor(String commentAuthor) {
		this.commentAuthor = commentAuthor;
	}

}*/
```

CommentAuthor

```
package com.scsu.postTo;



public class CommentAuthor {
	private String comment;
	private String commentAuthor;
	public String getComment() {
		return comment;
	}
	public void setComment(String comment) {
		this.comment = comment;
	}
	public String getCommentAuthor() {
		return commentAuthor;
	}
	public void setCommentAuthor(String commentAuthor) {
		this.commentAuthor = commentAuthor;
	}

}
```

JSP Pages

CarPoolPostPage

```
<%@ taglib prefix ="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<html>

<body>
<div id="header">
<h1>Student Portal</h1>
</div>
<ul>
  <li><a href="<c:url value="homePage"/>">Home</a></li>
  <li><a href="<c:url value="studyPostPage"/>">Knowledge Sharing</a></li>
  <li><a href="<c:url value="carPoolPostPage"/>">Car Pooling</a></li>
  <li><a href="<c:url value="accomodation"/>">Accommodation</a></li>
  <ul style="float:right;list-style-type:none;">
    <li><a  href="<c:url value="logout"/>">Logout</a></li>
  </ul>
</ul>


<div id="nav">

                        <a class = "button" href="<c:url value="carPoolPostPage"/>" >Need a
Ride</a><br>
                        <a class = "button" href="<c:url value="carRentPage"/>">Have a Car to
rent</a><br>

</div>


<div id="postSection">
<!--This form is for new post button  -->
<form action="/ScsuWebApp/carPoolNewPost" method="post" >
<table>
            <tr><td><h3>Need a Ride ?</h3><input type="text" name="carPoolPost"
title="First Name" style="color:#888;"
    value="Enter Your Post here" onfocus="inputFocus(this)" onblur="inputBlur(this)">
            <input type ="submit" value="POST"></td></tr>
</table>
</form>

            <c:set var="textCount" value="0" scope="page"/>
```

```
        <c:forEach items="${postList}" var="CarPostTo">
        <form action="/ScsuWebApp/pushCarPoolComments/${CarPostTo.postId}"
method="post" >
        <table >
                <tr >
                        <td colspan="4">Post Title : <c:out value="${CarPostTo.postTitle}"
/></td>
                        </tr ><br>
                        <tr >
                        <td colspan="4">Post ID : <c:out value="${CarPostTo.postId}"
/></td>
                </tr>
                <tr>
                        <td>Posted By  : <c:out value="${CarPostTo.postAuthor}" /></td>

                </tr>
                <tr>
                        <td>comments</td>
                </tr>
                <c:forEach items="${CarPostTo.advancedComments}" var="element">
                        <tr>
                        <td style="background-color: rgb(242,253,252)" colspan="4" >
                                <c:out value=" Replied by : ${element.commentAuthor}"
/><br><br>    <c:out value=">> ${element.comment}" />
                        </td>
                        </tr>
                </c:forEach>
                <%-- <tr><td><a class = "button" href="<c:url
value="comment"/>">Comment</a></td></tr> --%>
        <tr><td style="background-color: rgb(242,253,252)"  ><input type="text"
name="comments">
        <input type ="submit" value="comment"></td></tr>
        </table>
   </form>
        </c:forEach>
</div>


<!--
<div id="footer">
Copyright © SCSU
</div>
 -->

  </body>
</html>
```

```
<style>
#nav {
    line-height:60px;
    background-color:#eeeeee;
    height:100%;
    width:150px;
    float:left;
    padding:5px;
}

/* .postSection {
    width:750px;
    float:left;
    padding:10px;
            height:300px;
            background-color : rgb(196,225,255);
}
 */
#postSection {
    display: table;
    height: 100%;
    overflow: hidden;
    position: relative;
    float:left;
     width:750px;
            background-color : rgb(196,225,255);
}

#header {
    background-color:rgb(70,2,251);
    color:white;
    text-align:center;
    padding:5px;
}

#footer {
    background-color:black;
    color:white;
    clear:both;
    text-align:center;
            bottom: 0%;
    width:100%;
    height:40px;
}

#menu {
    background-color:white;
    color:white;
    clear:both;
    text-align:center;
    padding:5px;

}

#logout {
    background-color:white;
    color:white;
    clear:both;
    float : left;
}


ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    overflow: hidden;
    background-color: #333;
}

li {
    float: left;
    border-right:1px solid #bbb;
}

li a:last-child {
    border-right: none;
}

li a {
    display: block;
    color: white;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
}

li a:hover:not(.active) {
    background-color: #111;
}

</style>
```

```
<script type="text/javascript">
function inputFocus(i){
   if(i.value==i.defaultValue){ i.value=""; i.style.color="#000"; }
}
function inputBlur(i){
   if(i.value==""){ i.value=i.defaultValue; i.style.color="#888"; }
}
</script>
```

CarRentPage

```
<%@ taglib prefix =“c” uri=“http://java.sun.com/jsp/jstl/core” %>
<%@ taglib uri=“http://www.springframework.org/tags/form” prefix=“form”%>
<html>

<body>
<div id=“header”>
<h1>Student Portal</h1>
</div>

<ul>
  <li><a href=“<c:url value=“homePage”/>“>Home</a></li>
  <li><a href=“<c:url value=“studyPostPage”/>“>Knowledge Sharing</a></li>
  <li><a href=“<c:url value=“carPoolPostPage”/>“>Car Pooling</a></li>
  <li><a href=“<c:url value=“accomodation”/>“>Accommodation</a></li>
  <ul style=“float:right;list-style-type:none;”>
    <li><a  href=“<c:url value=“logout”/>“>Logout</a></li>
  </ul>
</ul>


<div id=“nav”>

        <a class = “button” href=“<c:url value=“carPoolPostPage”/>“ >Need a Ride</a><br>
        <a class = “button” href=“<c:url value=“carRentPage”/>“>Have a Car to
rent</a><br>


</div>


<div id=“postSection”>
<!--This form is for new post button  -->
<form action=“/ScsuWebApp/carRentNewPost” method=“post” >
<table>
        <tr><td><h3>Have a Car to rent ?</h3><input type=“text” name=“carRentPost”
title=“First Name” style=“color:#888;”
    value=“Enter Your Post here” onfocus=“inputFocus(this)” onblur=“inputBlur(this)”>
        <input type =“submit” value=“POST”></td></tr>
</table>
</form>

        <c:set var=“textCount” value=“0” scope=“page”/>
        <c:forEach items=“${postList}” var=“CarPostTo”>
        <form action=“/ScsuWebApp/pushCarRentComments/${CarPostTo.postId}”
method=“post” >
```

```
        <table >
                <tr >
                        <td>Post Title : <c:out value="${CarPostTo.postTitle}" /></td>

                </tr>
                        <tr >

                        <td>Post ID : <c:out value="${CarPostTo.postId}" /></td>
                </tr>
                <tr>
                        <td>Posted By  : <c:out value="${CarPostTo.postAuthor}" /></td>

                </tr>
                <tr>
                        <td>comments</td>
                </tr>
                <c:forEach items="${CarPostTo.advancedComments}" var="element">
                        <tr>
                <%--    <td BGCOLOR = rgb(242,253,252) colspan="4" >
                                <c:out value="${element.comment}" />
                        </td>
                        <td >
                                <c:out value="--------------------${element.commentAuthor}" />
                        </td> --%>
                        <td style="background-color:  rgb(242,253,252)" colspan="4" >
                                <c:out value=" Replied by : ${element.commentAuthor}"
/><br><br>      <c:out value=">> ${element.comment}" />
                        </td>
                        </tr>
                </c:forEach>
                <%-- <tr><td><a class = "button" href="<c:url
value="comment"/>">Comment</a></td></tr> --%>
        <tr><td style="background-color:  rgb(242,253,252)" colspan="4" ><input
type="text" name="comments">
        <input type ="submit" value="comment"></td></tr>
        </table>
   </form>
        </c:forEach>
</div>


<!--
<div id="footer">
Copyright © SCSU
</div>
 -->
```

```
  </body>
</html>


<style>
#nav {
   line-height:60px;
   background-color:#eeeeee;
   height:100%;
   width:150px;
   float:left;
   padding:5px;
}

/* .postSection {
   width:750px;
   float:left;
   padding:10px;
         height:300px;
         background-color : rgb(196,225,255);
}
 */
#postSection {
   display: table;
   height: 100%;
   overflow: hidden;
   position: relative;
  width:750px;
   float:left;
   background-color : rgb(196,225,255);
}

#header {
   background-color:rgb(70,2,251);
   color:white;
   text-align:center;
   padding:5px;
}

#footer {
   background-color:black;
   color:white;
   clear:both;
   text-align:center;
         bottom: 0%;
   width:100%;
   height:40px;
}

#menu {
   background-color:white;
   color:white;
   clear:both;
   text-align:center;
   padding:5px;

}

#logout {
   background-color:white;
   color:white;
   clear:both;
   float : left;
}


ul {
   list-style-type: none;
   margin: 0;
   padding: 0;
   overflow: hidden;
   background-color: #333;
}

li {
   float: left;
   border-right:1px solid #bbb;
}

li:last-child {
   border-right: none;
}

li a {
   display: block;
   color: white;
   text-align: center;
   padding: 14px 16px;
   text-decoration: none;
}

li a:hover:not(.active) {
   background-color: #111;
}
```

```
</style>

<script type="text/javascript">
function inputFocus(i){
   if(i.value==i.defaultValue){ i.value=““; i.style.color=“#000”; }
}
function inputBlur(i){
   if(i.value==““){ i.value=i.defaultValue; i.style.color=“#888”; }
}
</script>
```

DefaultPost

```
<%@ taglib prefix =“c” uri=“http://java.sun.com/jsp/jstl/core” %>
<%@ taglib uri=“http://www.springframework.org/tags/form” prefix=“form”%>
<html>

<body>
<div id=“header”>
<h1>Student Portal</h1>
</div>

<ul>
  <li><a href=“<c:url value=“homePage”/>“>Home</a></li>
  <li><a href=“<c:url value=“studyPostPage”/>“>Knowledge Sharing</a></li>
  <li><a href=“<c:url value=“carPoolPostPage”/>“>Car Pooling</a></li>
  <li><a href=“<c:url value=“accomodation”/>“>Accommodation</a></li>
  <ul style=“float:right;list-style-type:none;”>
    <li><a  href=“<c:url value=“logout”/>“>Logout</a></li>
  </ul>
</ul>


<div id=“nav”>

            <a class = “button” href=“<c:url value=“defaultPost”/>“ >Need a room to share</a>
            <a class = “button” href=“<c:url value=“roomRentPage”/>“>Have a room to
rent</a><br>
            <a class = “button” href=“<c:url value=“roomSharePage”/>“>Have a room to
share</a><br>


</div>



<div id=“postSection”>
<!--This form is for new post button  -->
<form action=“/ScsuWebApp/accPost” method=“post” >
<table>
            <tr><td><h3>Need a room to share ?</h3><input type=“text” name=“accPost”
title=“First Name” style=“color:#888;”
    value=“Enter Your Post here” onfocus=“inputFocus(this)” onblur=“inputBlur(this)”>
            <input type =“submit” value=“POST”></td></tr>
</table>
</form>

            <c:set var=“textCount” value=“0” scope=“page”/>
```

```
        <c:forEach items="${postList}" var="RoomPostTo">
        <form action="/ScsuWebApp/pushComments/${RoomPostTo.postId}"
method="post" >
        <table >
                <tr >
                        <td>Post Title : <c:out value="${RoomPostTo.postTitle}" /></td>

                </tr>
                        <tr >
                        <td>Post ID : <c:out value="${RoomPostTo.postId}" /></td>

                </tr>
                <tr>
                        <td>Posted By  : <c:out value="${RoomPostTo.postAuthor}" /></td>

                </tr>
                <tr>
                        <td>comments</td>
                </tr>
                <c:forEach items="${RoomPostTo.advancedComments}" var="element">
                        <tr>
                <%--    <td BGCOLOR = rgb(242,253,252) colspan="4" >
                                <c:out value="${element.comment}" />
                        </td>
                        <td >
                                <c:out value="--------------------${element.commentAuthor}" />
                        </td> --%>
                        <td style="background-color:  rgb(242,253,252)" colspan="4" >
                                <c:out value=" Replied by : ${element.commentAuthor}"
/><br><br>      <c:out value=">> ${element.comment}" />
                        </td>
                        </tr>
                </c:forEach>
                <%-- <tr><td><a class = "button" href="<c:url
value="comment"/>">Comment</a></td></tr> --%>
        <tr><td style="background-color:  rgb(242,253,252)" colspan="4" ><input
type="text" name="comments">
        <input type ="submit" value="comment"></td></tr>
        </table>
   </form>
        </c:forEach>
</div>


<!--
<div id="footer">
Copyright © SCSU
```

```
</div>
 -->

  </body>
</html>


<style>
#nav {
    line-height:60px;
    background-color:#eeeeee;
    height:100%;
    width:150px;
    float:left;
    padding:5px;
}

/* .postSection {
    width:750px;
    float:left;
    padding:10px;
            height:300px;
            background-color : rgb(196,225,255);
}
 */
#postSection {
    display: table;
    height: 100%;
    overflow: hidden;
    position: relative;
   width:750px;
    float:left;
    background-color : rgb(196,225,255);
}

#header {
    background-color:rgb(70,2,251);
    color:white;
    text-align:center;
    padding:5px;
}

#footer {
    background-color:black;
    color:white;
    clear:both;
    text-align:center;
```

```
        bottom: 0%;
    width:100%;
    height:40px;
}

#menu {
    background-color:white;
    color:white;
    clear:both;
    text-align:center;
    padding:5px;

}

#logout {
    background-color:white;
    color:white;
    clear:both;
    float : left;
}


ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    overflow: hidden;
    background-color: #333;
}

li {
    float: left;
    border-right:1px solid #bbb;
}

li:last-child {
    border-right: none;
}

li a {
    display: block;
    color: white;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
}
```

```
li a:hover:not(.active) {
   background-color: #111;
}

</style>

<script type="text/javascript">
function inputFocus(i){
   if(i.value==i.defaultValue){ i.value=""; i.style.color="#000"; }
}
function inputBlur(i){
   if(i.value==""){ i.value=i.defaultValue; i.style.color="#888"; }
}
</script>
```

HomePage

```
<%@ taglib prefix =“c” uri=“http://java.sun.com/jsp/jstl/core” %>
<%@ taglib prefix=“spring” uri=“http://www.springframework.org/tags”%>
<html>

<body>



<div id=“header”>
<h1>Student Portal</h1>
</div>

<ul>
  <li><a href=“<c:url value=“homePage”/>“>Home</a></li>
  <li><a href=“<c:url value=“studyPostPage”/>“>Knowledge Sharing</a></li>
  <li><a href=“<c:url value=“carPoolPostPage”/>“>Car Pooling</a></li>
  <li><a href=“<c:url value=“accomodation”/>“>Accommodation</a></li>
  <ul style=“float:right;list-style-type:none;”>
    <li><a  href=“<c:url value=“logout”/>“>Logout</a></li>
  </ul>
</ul>

<%-- <div id=“menu”>
<tr>
<td><a href=“<c:url value=“accommodation”/>“>Home</a></td>
<td><a href=“<c:url value=“studyPostPage”/>“>Knowledge Sharing</a></td>
<td><a href=“<c:url value=“carPoolPostPage”/>“>Car Pooling</a></td>
<td><a href=“<c:url value=“accomodation”/>“>Accommodation</a></td>
</tr>
</div> --%>

<div id=“section”>
Hi, ${welcomeMessage}
<h1>Post Ads by navigating to respective services available</h1>
</div>

<div id=“nav”>

<tr>

        <a class = “button” href=“<c:url value=“defaultPost”/>“ >Need a room to
share</a><br>
        <a class = “button” href=“<c:url value=“roomRentPage”/>“>Have a room to
rent</a><br>
```

```
	<a class = “button” href=“<c:url value=“roomSharePage”/>“>Have a room to
share</a><br>
	<a class = “button” href=“<c:url value=“carPoolPostPage”/>“ >Need a Ride</a><br>
	<a class = “button” href=“<c:url value=“carRentPage”/>“>Have a Car to
rent</a><br>
	<a class = “button” href=“<c:url value=“studyPostPage”/>“ >Study
Materials</a><br>
	<a class = “button” href=“<c:url value=“jobPostPage”/>“>Job References</a><br>
	<a class = “button” href=“<c:url value=“miscPostPage”/>“>Miscellaneous</a><br>
</tr>

</div>

<div id=“footer”>
Copyright © SCSU
</div>
<%-- <tr>
	<td><a href=“<c:url value=“insert.jsp”/>“>Insert</a></td>
	<td><a href=“<c:url value=“update.jsp”/>“>Update</a></td>
	<td><a href=“<c:url value=“delete.jsp”/>“>Delete</a></td>
</tr>
 --%>


</body>
</html>
<style>
.navigation {

width: 180px;
font-family: Arial, Helvetica, sans-serif;
   font-size: 90%;
   font-weigth: bold;
}


#header {
   background-color:rgb(239,58,68);
   color:white;
   text-align:center;
   padding:5px;
}
#nav {
   line-height:40px;
   background-color:#eeeeee;
   height:450px;
   width:150px;
   float:left;
   padding:5px;
}
#section {
   width:350px;
   float:right;
   padding:10px;
}
#footer {
   background-color:black;
   color:white;
   clear:both;
   text-align:center;
   padding:5px;
}

#menu {
   background-color:white;
   color:white;
   clear:both;
   text-align:center;
   padding:5px;
}
#logout {
   background-color:white;
   color:white;
   clear:both;
   float : left;
}


ul {
   list-style-type: none;
   margin: 0;
   padding: 0;
   overflow: hidden;
   background-color: rgb(128,0,64);
}

li {
   float: left;
   border-right:1px solid #bbb;
}

li:last-child {
   border-right: none;
}
```

```
li a {
   display: block;
   color: white;
   text-align: center;
   padding: 14px 16px;
   text-decoration: none;
}

li a:hover:not(.active) {
   background-color:rgb(176,176,255);
}

</style>
```

JobPostPage

```
<%@ taglib prefix =“c” uri=“http://java.sun.com/jsp/jstl/core” %>
<%@ taglib uri=“http://www.springframework.org/tags/form” prefix=“form”%>
<html>

<body>
<div id=“header”>
<h1>Student Portal</h1>
</div>

<ul>
  <<li><a href=“<c:url value=“homePage”/>“>Home</a></li>
  <li><a href=“<c:url value=“studyPostPage”/>“>Knowledge Sharing</a></li>
  <li><a href=“<c:url value=“carPoolPostPage”/>“>Car Pooling</a></li>
  <li><a href=“<c:url value=“accomodation”/>“>Accommodation</a></li>
  <ul style=“float:right;list-style-type:none;”>
    <li><a  href=“<c:url value=“logout”/>“>Logout</a></li>
  </ul>
</ul>


<div id=“nav”>

            <a class = “button” href=“<c:url value=“studyPostPage”/>“ >Study
Materials</a><br>
            <a class = “button” href=“<c:url value=“jobPostPage”/>“>Job References</a><br>
            <a class = “button” href=“<c:url value=“miscPostPage”/>“>Miscellaneous</a><br>


</div>



<div id=“postSection”>
<!--This form is for new post button  -->
<form action=“/ScsuWebApp/jobNewPost” method=“post” >
<table>
            <tr><td><h3>Have/Need Job References ?</h3><input type=“text” name=“jobPost”
title=“First Name” style=“color:#888;”
    value=“Enter Your Post here” onfocus=“inputFocus(this)” onblur=“inputBlur(this)”>
            <input type =“submit” value=“POST”></td></tr>
</table>
</form>

            <c:set var=“textCount” value=“0” scope=“page”/>
            <c:forEach items=“${postList}” var=“JobPostTo”>
```

```
        <form action="/ScsuWebApp/pushJobPostComments/${JobPostTo.postId}"
method="post" >
        <table >
                <tr >
                        <td>Post Title : <c:out value="${JobPostTo.postTitle}" /></td>


                </tr>
                        <tr >

                        <td>Post ID : <c:out value="${JobPostTo.postId}" /></td>
                </tr>
                <tr>
                        <td>Posted By  : <c:out value="${JobPostTo.postAuthor}" /></td>

                </tr>
                <tr>
                        <td>comments</td>
                </tr>
                <c:forEach items="${JobPostTo.advancedComments}" var="element">
                        <tr>
                <%--    <td BGCOLOR = rgb(242,253,252) colspan="4" >
                                <c:out value="${element.comment}" />
                        </td>
                        <td >
                                <c:out value="--------------------${element.commentAuthor}" />
                        </td> --%>
                        <td style="background-color:  rgb(242,253,252)" colspan="4" >
                                <c:out value=" Replied by : ${element.commentAuthor}"
/><br><br>      <c:out value=">> ${element.comment}" />
                        </td>
                        </tr>
                </c:forEach>
                <%-- <tr><td><a class = "button" href="<c:url
value="comment"/>">Comment</a></td></tr> --%>
        <tr><td style="background-color:  rgb(242,253,252)" colspan="4" ><input
type="text" name="comments">
        <input type ="submit" value="comment"></td></tr>
        </table>
   </form>
        </c:forEach>
</div>


<!--
<div id="footer">
Copyright © SCSU
```

```
</div>
 -->

  </body>
</html>


<style>
#nav {
    line-height:60px;
    background-color:#eeeeee;
    height:100%;
    width:150px;
    float:left;
    padding:5px;
}

/* .postSection {
    width:750px;
    float:left;
    padding:10px;
            height:300px;
            background-color : rgb(196,225,255);
}
 */
#postSection {
    display: table;
    height: 100%;
    overflow: hidden;
    position: relative;
   width:750px;
    float:left;
    background-color : rgb(196,225,255);
}

#header {
    background-color:rgb(70,2,251);
    color:white;
    text-align:center;
    padding:5px;
}

#footer {
    background-color:black;
    color:white;
    clear:both;
    text-align:center;
```

```
        bottom: 0%;
   width:100%;
   height:40px;
}

#menu {
   background-color:white;
   color:white;
   clear:both;
   text-align:center;
   padding:5px;

}

#logout {
   background-color:white;
   color:white;
   clear:both;
   float : left;
}


ul {
   list-style-type: none;
   margin: 0;
   padding: 0;
   overflow: hidden;
   background-color: #333;
}

li {
   float: left;
   border-right:1px solid #bbb;
}

li:last-child {
   border-right: none;
}

li a {
   display: block;
   color: white;
   text-align: center;
   padding: 14px 16px;
   text-decoration: none;
}
```

```
li a:hover:not(.active) {
   background-color: #111;
}

</style>

<script type="“text/javascript”>
function inputFocus(i){
   if(i.value==i.defaultValue){ i.value=““; i.style.color=“#000”; }
}
function inputBlur(i){
   if(i.value==““){ i.value=i.defaultValue; i.style.color=“#888”; }
}
</script>
```

Login

```
<html>
<div id=“header”>
<h1>Student Portal</h1>
</div>
<style>
#header {
   background-color:rgb(239,58,68);
   color:white;
   text-align:center;
   padding:5px;
}
</style>
<body>
<div id=“login”>
<form action=“/ScsuWebApp/WelcomePage” method=“post”>
<table>
<tr><td><p> User Name  :</p></td><td><p><input type=“text” name=“userName”>
</p></td></tr>
<tr><td><p> Password &nbsp;  :</p></td><td><p><input type=“password”
name=“password”></p></td></tr>
<tr><td><input type =“submit” value=“Login”></td></tr>
<tr><td>${welcomeMessage}</td></tr>
</table>

</form>
</div>

<div id=“register”>
<form action=“/ScsuWebApp/register” method=“post”>
<table>
<tr><td><p> First Name : </p></td><td><p><input type=“text” name=“FName”>
</p></td></tr>
<tr><td><p> Last Name : </p></td><td><p><input type=“text” name=“LName”>
</p></td></tr>
<tr><td><p> Student ID :</p></td><td><p><input type=“text” name=“SId”>
</p></td></tr>
<tr><td><p> Password  : </p></td><td><p><input type=“password”
name=“password”></p></td></tr>
<tr><td><p> Re-Enter Password  : </p></td><td><p><input type=“password”
name=“repassword”></p></td></tr>

<tr><td><input type =“submit” value=“Register”></td></tr>
<tr><td>${registerMsg}</td></tr>
</table>
```

```
</form>
</div>
</body>

</html>

<style>
#login {
   line-height:60px;
   background-color:#eeeeee;
   height:100px;
   width:250px;
   float:left;
   padding:5px;
}

#register {
   line-height:60px;
   background-color:#eeeeee;

   width:300px;
   float:right;
   padding:5px;
}
</style>
```

miscPostPage

```
<%@ taglib prefix =“c” uri=“http://java.sun.com/jsp/jstl/core” %>
<%@ taglib uri=“http://www.springframework.org/tags/form” prefix=“form”%>
<html>

<body>
<div id=“header”>
<h1>Student Portal</h1>
</div>

<ul>
  <li><a href=“<c:url value=“homePage”/>“>Home</a></li>
  <li><a href=“<c:url value=“studyPostPage”/>“>Knowledge Sharing</a></li>
  <li><a href=“<c:url value=“carPoolPostPage”/>“>Car Pooling</a></li>
  <li><a href=“<c:url value=“accomodation”/>“>Accommodation</a></li>
  <ul style=“float:right;list-style-type:none;”>
    <li><a  href=“<c:url value=“logout”/>“>Logout</a></li>
  </ul>
</ul>
<div id=“nav”>

            <a class = “button” href=“<c:url value=“studyPostPage”/>“ >Study
Materials</a><br>
            <a class = “button” href=“<c:url value=“jobPostPage”/>“>Job References</a><br>
            <a class = “button” href=“<c:url value=“miscPostPage”/>“>Miscellaneous</a><br>


</div>



<div id=“postSection”>
<!--This form is for new post button  -->
<form action=“/ScsuWebApp/miscNewPost” method=“post” >
<table>
            <tr><td><h3>For Miscellaneous QA</h3><input type=“text” name=“miscPost”
title=“First Name” style=“color:#888;”
    value=“Enter Your Post here” onfocus=“inputFocus(this)” onblur=“inputBlur(this)”>
            <input type =“submit” value=“POST”></td></tr>
</table>
</form>

            <c:set var=“textCount” value=“0” scope=“page”/>
            <c:forEach items=“${postList}” var=“MiscPostTo”>
```

```
        <form action="/ScsuWebApp/pushMiscPostComments/${MiscPostTo.postId}"
method="post" >
        <table >
                <tr >
                        <td>Post Title : <c:out value="${MiscPostTo.postTitle}" /></td>


                </tr>
                        <tr >

                        <td>Post ID : <c:out value="${MiscPostTo.postId}" /></td>

                </tr>
                <tr>
                        <td>Posted By  : <c:out value="${MiscPostTo.postAuthor}" /></td>

                </tr>
                <tr>
                        <td>comments</td>
                </tr>
                <c:forEach items="${MiscPostTo.advancedComments}" var="element">
                        <tr>
                <%--    <td BGCOLOR = rgb(242,253,252) colspan="4" >
                                <c:out value="${element.comment}" />
                        </td>
                        <td >
                                <c:out value="--------------------${element.commentAuthor}" />
                        </td> --%>
                        <td style="background-color:  rgb(242,253,252)" colspan="4" >
                                <c:out value=" Replied by : ${element.commentAuthor}"
/><br><br>      <c:out value=">> ${element.comment}" />
                        </td>
                        </tr>
                </c:forEach>
                <%-- <tr><td><a class = "button" href="<c:url
value="comment"/>">Comment</a></td></tr> --%>
        <tr><td style="background-color:  rgb(242,253,252)" colspan="4" ><input
type="text" name="comments">
        <input type ="submit" value="comment"></td></tr>
        </table>
   </form>
        </c:forEach>
</div>


<!--
<div id="footer">
```

```
Copyright © SCSU
</div>
 -->

  </body>
</html>


<style>
#nav {
    line-height:60px;
    background-color:#eeeeee;
    height:100%;
    width:150px;
    float:left;
    padding:5px;
}

/* .postSection {
    width:750px;
    float:left;
    padding:10px;
            height:300px;
            background-color : rgb(196,225,255);
}
 */
#postSection {
    display: table;
    height: 100%;
    overflow: hidden;
    position: relative;
   width:750px;
    float:left;
    background-color : rgb(196,225,255);
}

#header {
    background-color:rgb(70,2,251);
    color:white;
    text-align:center;
    padding:5px;
}

#footer {
    background-color:black;
    color:white;
    clear:both;
   text-align:center;
         bottom: 0%;
   width:100%;
   height:40px;
}

#menu {
   background-color:white;
   color:white;
   clear:both;
   text-align:center;
   padding:5px;

}

#logout {
   background-color:white;
   color:white;
   clear:both;
   float : left;
}


ul {
   list-style-type: none;
   margin: 0;
   padding: 0;
   overflow: hidden;
   background-color: #333;
}

li {
   float: left;
   border-right:1px solid #bbb;
}

li:last-child {
   border-right: none;
}

li a {
   display: block;
   color: white;
   text-align: center;
   padding: 14px 16px;
   text-decoration: none;
}
```

```
li a:hover:not(.active) {
   background-color: #111;
}

</style>

<script type="text/javascript">
function inputFocus(i){
   if(i.value==i.defaultValue){ i.value=““; i.style.color=“#000”; }
}
function inputBlur(i){
   if(i.value==““){ i.value=i.defaultValue; i.style.color=“#888”; }
}
</script>
```

roomRentPage

```
<%@ taglib prefix =“c” uri=“http://java.sun.com/jsp/jstl/core” %>
<%@ taglib uri=“http://www.springframework.org/tags/form” prefix=“form”%>
<html>

<body>
<div id=“header”>
<h1>Student Portal</h1>
</div>

<ul>
  <li><a href=“<c:url value=“homePage”/>“>Home</a></li>
  <li><a href=“<c:url value=“studyPostPage”/>“>Knowledge Sharing</a></li>
  <li><a href=“<c:url value=“carPoolPostPage”/>“>Car Pooling</a></li>
  <li><a href=“<c:url value=“accomodation”/>“>Accommodation</a></li>
  <ul style=“float:right;list-style-type:none;”>
    <li><a  href=“<c:url value=“logout”/>“>Logout</a></li>
  </ul>
</ul>

<div id=“nav”>

            <a class = “button” href=“<c:url value=“defaultPost”/>“ >Need a room to share</a>
            <a class = “button” href=“<c:url value=“roomRentPage”/>“>Have a room to
rent</a><br>
            <a class = “button” href=“<c:url value=“roomSharePage”/>“>Have a room to
share</a><br>


</div>



<div id=“postSection”>
<!--This form is for new post button  -->
<form action=“/ScsuWebApp/roomRentPost” method=“post” >
<table>
            <tr><td><h3>Have a room to rent ?</h3><input type=“text” name=“roomRentPost”
title=“First Name” style=“color:#888;”
    value=“Enter Your Post here” onfocus=“inputFocus(this)” onblur=“inputBlur(this)”>
            <input type =“submit” value=“POST”></td></tr>
</table>
</form>

            <c:set var=“textCount” value=“0” scope=“page”/>
            <c:forEach items=“${postList}” var=“RoomPostTo”>
```

```
        <form action="/ScsuWebApp/pushRoomRentComments/${RoomPostTo.postId}"
method="post" >
        <table >
                <tr >
                        <td>Post Title : <c:out value="${RoomPostTo.postTitle}" /></td>


                </tr>
                        <tr >

                        <td>Post ID : <c:out value="${RoomPostTo.postId}" /></td>

                </tr>
                <tr>
                        <td>Posted By  : <c:out value="${RoomPostTo.postAuthor}" /></td>

                </tr>
                <tr>
                        <td>comments</td>
                </tr>
                <c:forEach items="${RoomPostTo.advancedComments}" var="element">
                        <tr>
                <%--    <td BGCOLOR = rgb(242,253,252) colspan="4" >
                                <c:out value="${element.comment}" />
                        </td>
                        <td >
                                <c:out value="--------------------${element.commentAuthor}" />
                        </td> --%>
                        <td style="background-color:  rgb(242,253,252)" colspan="4" >
                                <c:out value=" Replied by : ${element.commentAuthor}"
/><br><br>      <c:out value=">> ${element.comment}" />
                        </td>
                        </tr>
                </c:forEach>
                <%-- <tr><td><a class = "button" href="<c:url
value="comment"/>">Comment</a></td></tr> --%>
        <tr><td style="background-color:  rgb(242,253,252)" colspan="4" ><input
type="text" name="comments">
        <input type ="submit" value="comment"></td></tr>
        </table>
   </form>
        </c:forEach>
</div>


<!--
<div id="footer">
```

```
Copyright © SCSU
</div>
 -->

  </body>
</html>


<style>
#nav {
    line-height:60px;
    background-color:#eeeeee;
    height:100%;
    width:150px;
    float:left;
    padding:5px;
}

/* .postSection {
    width:750px;
    float:left;
    padding:10px;
            height:300px;
            background-color : rgb(196,225,255);
}
 */
#postSection {
    display: table;
    height: 100%;
    overflow: hidden;
    position: relative;
   width:750px;
    float:left;
    background-color : rgb(196,225,255);
}

#header {
    background-color:rgb(70,2,251);
    color:white;
    text-align:center;
    padding:5px;
}

#footer {
    background-color:black;
    color:white;
    clear:both;
   text-align:center;
         bottom: 0%;
   width:100%;
   height:40px;
}

#menu {
   background-color:white;
   color:white;
   clear:both;
   text-align:center;
   padding:5px;

}

#logout {
   background-color:white;
   color:white;
   clear:both;
   float : left;
}


ul {
   list-style-type: none;
   margin: 0;
   padding: 0;
   overflow: hidden;
   background-color: #333;
}

li {
   float: left;
   border-right:1px solid #bbb;
}

li:last-child {
   border-right: none;
}

li a {
   display: block;
   color: white;
   text-align: center;
   padding: 14px 16px;
   text-decoration: none;
}
```

```
li a:hover:not(.active) {
   background-color: #111;
}
</style>

<script type=“text/javascript”>
function inputFocus(i){
   if(i.value==i.defaultValue){ i.value=““; i.style.color=“#000”; }
}
function inputBlur(i){
   if(i.value==““){ i.value=i.defaultValue; i.style.color=“#888”; }
}
</script>
```

RoomsharePage

```
<%@ taglib prefix =“c” uri=“http://java.sun.com/jsp/jstl/core” %>
<%@ taglib uri=“http://www.springframework.org/tags/form” prefix=“form”%>
<html>

<body>
<div id=“header”>
<h1>Student Portal</h1>
</div>

<ul>
 <li><a href=“<c:url value=“homePage”/>“>Home</a></li>
  <li><a href=“<c:url value=“studyPostPage”/>“>Knowledge Sharing</a></li>
  <li><a href=“<c:url value=“carPoolPostPage”/>“>Car Pooling</a></li>
  <li><a href=“<c:url value=“accomodation”/>“>Accommodation</a></li>
  <ul style=“float:right;list-style-type:none;”>
    <li><a  href=“<c:url value=“logout”/>“>Logout</a></li>
  </ul>
</ul>
<div id=“nav”>

            <a class = “button” href=“<c:url value=“defaultPost”/>“ >Need a room to share</a>
            <a class = “button” href=“<c:url value=“roomRentPage”/>“>Have a room to
rent</a><br>
            <a class = “button” href=“<c:url value=“roomSharePage”/>“>Have a room to
share</a><br>


</div>



<div id=“postSection”>
<!--This form is for new post button  -->
<form action=“/ScsuWebApp/roomSharePost” method=“post” >
<table>
            <tr><td><h3>Have a room to share ?</h3><input type=“text”
name=“roomSharePost” title=“First Name” style=“color:#888;”
    value=“Enter Your Post here” onfocus=“inputFocus(this)” onblur=“inputBlur(this)”>
            <input type =“submit” value=“POST”></td></tr>
</table>
</form>

            <c:set var=“textCount” value=“0” scope=“page”/>
            <c:forEach items=“${postList}” var=“RoomPostTo”>
```

```
        <form action="/ScsuWebApp/pushRoomShareComments/${RoomPostTo.postId}"
method="post" >
        <table >
                <tr >
                        <td>Post Title : <c:out value="${RoomPostTo.postTitle}" /></td>


                </tr>
                        <tr >

                        <td>Post ID : <c:out value="${RoomPostTo.postId}" /></td>

                </tr>
                <tr>
                        <td>Posted By  : <c:out value="${RoomPostTo.postAuthor}" /></td>

                </tr>
                <tr>
                        <td>comments</td>
                </tr>
                <c:forEach items="${RoomPostTo.advancedComments}" var="element">
                        <tr>
                <%--    <td BGCOLOR = rgb(242,253,252) colspan="4" >
                                <c:out value="${element.comment}" />
                        </td>
                        <td >
                                <c:out value="--------------------${element.commentAuthor}" />
                        </td> --%>
                        <td style="background-color:  rgb(242,253,252)" colspan="4" >
                                <c:out value=" Replied by : ${element.commentAuthor}"
/><br><br>      <c:out value=">> ${element.comment}" />
                        </td>
                        </tr>
                </c:forEach>
                <%-- <tr><td><a class = "button" href="<c:url
value="comment"/>">Comment</a></td></tr> --%>
        <tr><td style="background-color:  rgb(242,253,252)" colspan="4" ><input
type="text" name="comments">
        <input type ="submit" value="comment"></td></tr>
        </table>
   </form>
        </c:forEach>
</div>


<!--
<div id="footer">
```

```
Copyright © SCSU
</div>
 -->

  </body>
</html>


<style>
#nav {
    line-height:60px;
    background-color:#eeeeee;
    height:100%;
    width:150px;
    float:left;
    padding:5px;
}

/* .postSection {
    width:750px;
    float:left;
    padding:10px;
            height:300px;
            background-color : rgb(196,225,255);
}
 */
#postSection {
    display: table;
    height: 100%;
    overflow: hidden;
    position: relative;
   width:750px;
    float:left;
    background-color : rgb(196,225,255);
}

#header {
    background-color:rgb(70,2,251);
    color:white;
    text-align:center;
    padding:5px;
}

#footer {
    background-color:black;
    color:white;
    clear:both;
   text-align:center;
         bottom: 0%;
   width:100%;
   height:40px;
}

#menu {
   background-color:white;
   color:white;
   clear:both;
   text-align:center;
   padding:5px;

}

#logout {
   background-color:white;
   color:white;
   clear:both;
   float : left;
}


ul {
   list-style-type: none;
   margin: 0;
   padding: 0;
   overflow: hidden;
   background-color: #333;
}

li {
   float: left;
   border-right:1px solid #bbb;
}

li:last-child {
   border-right: none;
}

li a {
   display: block;
   color: white;
   text-align: center;
   padding: 14px 16px;
   text-decoration: none;
}
```

```
li a:hover:not(.active) {
   background-color: #111;
}
</style>

<script type="“text/javascript”>
function inputFocus(i){
   if(i.value==i.defaultValue){ i.value=““; i.style.color=“#000”; }
}
function inputBlur(i){
   if(i.value==““){ i.value=i.defaultValue; i.style.color=“#888”; }
}
</script>
```

studyPostPage

```
<%@ taglib prefix =“c” uri=“http://java.sun.com/jsp/jstl/core” %>
<%@ taglib uri=“http://www.springframework.org/tags/form” prefix=“form”%>
<html>

<body>
<div id=“header”>
<h1>Student Portal</h1>
</div>

<ul>
 <li><a href=“<c:url value=“homePage”/>“>Home</a></li>
  <li><a href=“<c:url value=“studyPostPage”/>“>Knowledge Sharing</a></li>
  <li><a href=“<c:url value=“carPoolPostPage”/>“>Car Pooling</a></li>
  <li><a href=“<c:url value=“accomodation”/>“>Accommodation</a></li>
  <ul style=“float:right;list-style-type:none;”>
    <li><a  href=“<c:url value=“logout”/>“>Logout</a></li>
  </ul>
</ul>
<div id=“nav”>

        <a class = “button” href=“<c:url value=“studyPostPage”/>“ >Study
Materials</a><br>
        <a class = “button” href=“<c:url value=“jobPostPage”/>“>Job References</a><br>
        <a class = “button” href=“<c:url value=“miscPostPage”/>“>Miscellaneous</a><br>


</div>



<div id=“postSection”>
<!--This form is for new post button  -->
<form action=“/ScsuWebApp/studyNewPost” method=“post” >
<table>
        <tr><td><h3>For Information On Study Materials</h3><input type=“text”
name=“studyPost” title=“First Name” style=“color:#888;”
   value=“Enter Your Post here” onfocus=“inputFocus(this)” onblur=“inputBlur(this)”>
        <input type =“submit” value=“POST”></td></tr>
</table>
</form>

        <c:set var=“textCount” value=“0” scope=“page”/>
        <c:forEach items=“${postList}” var=“StudyPostTo”>
        <form action=“/ScsuWebApp/pushStudyPostComments/${StudyPostTo.postId}”
method=“post” >
```

```
        <table >
                <tr >
                        <td>Post Title : <c:out value="${StudyPostTo.postTitle}" /></td>


                </tr>
                        <tr >

                        <td>Post ID : <c:out value="${StudyPostTo.postId}" /></td>

                </tr>
                <tr>
                        <td>Posted By  : <c:out value="${StudyPostTo.postAuthor}" /></td>

                </tr>
                <tr>
                        <td>comments</td>
                </tr>
                <c:forEach items="${StudyPostTo.advancedComments}" var="element">
                        <tr>
                <%--    <td BGCOLOR = rgb(242,253,252) colspan="4" >
                                <c:out value="${element.comment}" />
                        </td>
                        <td >
                                <c:out value="--------------------${element.commentAuthor}" />
                        </td> --%>
                        <td style="background-color:  rgb(242,253,252)" colspan="4" >
                                <c:out value=" Replied by : ${element.commentAuthor}"
/><br><br>      <c:out value=">> ${element.comment}" />
                        </td>
                        </tr>
                </c:forEach>
                <%-- <tr><td><a class = "button" href="<c:url
value="comment"/>">Comment</a></td></tr> --%>
        <tr><td style="background-color:  rgb(242,253,252)" colspan="4" ><input
type="text" name="comments">
        <input type ="submit" value="comment"></td></tr>
        </table>
   </form>
        </c:forEach>
</div>


<!--
<div id="footer">
Copyright © SCSU
</div>
```

```
 -->

  </body>
</html>


<style>
#nav {
    line-height:60px;
    background-color:#eeeeee;
    height:100%;
    width:150px;
    float:left;
    padding:5px;
}

/* .postSection {
    width:750px;
    float:left;
    padding:10px;
            height:300px;
            background-color : rgb(196,225,255);
}
 */
#postSection {
    display: table;
    height: 100%;
    overflow: hidden;
    position: relative;
   width:750px;
    float:left;
    background-color : rgb(196,225,255);
}

#header {
    background-color:rgb(70,2,251);
    color:white;
    text-align:center;
    padding:5px;
}

#footer {
    background-color:black;
    color:white;
    clear:both;
    text-align:center;
            bottom: 0%;
   width:100%;
   height:40px;
}

#menu {
   background-color:white;
   color:white;
   clear:both;
   text-align:center;
   padding:5px;

}

#logout {
   background-color:white;
   color:white;
   clear:both;
   float : left;
}


ul {
   list-style-type: none;
   margin: 0;
   padding: 0;
   overflow: hidden;
   background-color: #333;
}

li {
   float: left;
   border-right:1px solid #bbb;
}

li:last-child {
   border-right: none;
}

li a {
   display: block;
   color: white;
   text-align: center;
   padding: 14px 16px;
   text-decoration: none;
}

li a:hover:not(.active) {
   background-color: #111;
}

</style>

<script type="text/javascript">
function inputFocus(i){
   if(i.value==i.defaultValue){ i.value=""; i.style.color="#000"; }
}
function inputBlur(i){
   if(i.value==""){ i.value=i.defaultValue; i.style.color="#888"; }
}
</script>
```

welcomePage

```
<%@ taglib prefix =“c” uri=“http://java.sun.com/jsp/jstl/core” %>
<%@ taglib prefix=“spring” uri=“http://www.springframework.org/tags”%>
<html>

<body>



<div id=“header”>
<h1>Student Portal</h1>
</div>

<ul>
  <li><a href=“<c:url value=“homePage”/>“>Home</a></li>
  <li><a href=“<c:url value=“studyPostPage”/>“>Knowledge Sharing</a></li>
  <li><a href=“<c:url value=“carPoolPostPage”/>“>Car Pooling</a></li>
  <li><a href=“<c:url value=“accomodation”/>“>Accommodation</a></li>
  <ul style=“float:right;list-style-type:none;”>
    <li><a  href=“<c:url value=“logout”/>“>Logout</a></li>
  </ul>
</ul>

<%-- <div id=“menu”>
<tr>
<td><a href=“<c:url value=“accommodation”/>“>Home</a></td>
<td><a href=“<c:url value=“studyPostPage”/>“>Knowledge Sharing</a></td>
<td><a href=“<c:url value=“carPoolPostPage”/>“>Car Pooling</a></td>
<td><a href=“<c:url value=“accomodation”/>“>Accommodation</a></td>
</tr>
</div> --%>

<div id=“section”>
${welcomeMessage}, You have successfully logged in....
<h1>Post Ads by navigating to respective services available</h1>
</div>

<div id=“nav”>

<tr>

                        <a class = “button” href=“<c:url value=“defaultPost”/>“ >Need a room to
share</a><br>
            <a class = “button” href=“<c:url value=“roomRentPage”/>“>Have a room to
rent</a><br>
```

```
          <a class = “button” href=“<c:url value=“roomSharePage”/>“>Have a room to
share</a><br>
          <a class = “button” href=“<c:url value=“carPoolPostPage”/>“ >Need a Ride</a><br>
          <a class = “button” href=“<c:url value=“carRentPage”/>“>Have a Car to
rent</a><br>
          <a class = “button” href=“<c:url value=“studyPostPage”/>“ >Study
Materials</a><br>
          <a class = “button” href=“<c:url value=“jobPostPage”/>“>Job References</a><br>
          <a class = “button” href=“<c:url value=“miscPostPage”/>“>Miscellaneous</a><br>
</tr>

</div>

<div id=“footer”>
Copyright © SCSU
</div>
<%-- <tr>
          <td><a href=“<c:url value=“insert.jsp”/>“>Insert</a></td>
          <td><a href=“<c:url value=“update.jsp”/>“>Update</a></td>
          <td><a href=“<c:url value=“delete.jsp”/>“>Delete</a></td>
</tr>
 --%>


</body>
</html>
<style>
.navigation {

width: 180px;
font-family: Arial, Helvetica, sans-serif;
   font-size: 90%;
   font-weigth: bold;
}



#header {
   background-color:rgb(239,58,68);
   color:white;
   text-align:center;
   padding:5px;
}
#nav {
   line-height:40px;
   background-color:#eeeeee;
   height:450px;
   width:150px;
   float:left;
   padding:5px;
}
#section {
   width:350px;
   float:right;
   padding:10px;
}
#footer {
   background-color:black;
   color:white;
   clear:both;
   text-align:center;
   padding:5px;
}

#menu {
   background-color:white;
   color:white;
   clear:both;
   text-align:center;
   padding:5px;
}
#logout {
   background-color:white;
   color:white;
   clear:both;
   float : left;
}


ul {
   list-style-type: none;
   margin: 0;
   padding: 0;
   overflow: hidden;
   background-color: rgb(128,0,64);
}

li {
   float: left;
   border-right:1px solid #bbb;
}

li:last-child {
   border-right: none;
}
```

```
li a {
   display: block;
   color: white;
   text-align: center;
   padding: 14px 16px;
   text-decoration: none;
}

li a:hover:not(.active) {
   background-color:rgb(176,176,255);
}

</style>
```

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://java.sun.com/xml/ns/javaee"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
        id="WebApp_ID" version="3.0">
        <display-name>SCSUSpringProject</display-name>

        <servlet>
                <servlet-name>spring-dispatcher</servlet-name>
                <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
                <init-param>
                        <param-name>contextConfigLocation</param-name>
                        <param-value>classpath*:webApplicationContext.xml</param-value>
                </init-param>
                <load-on-startup>1</load-on-startup>
        </servlet>

        <servlet-mapping>
                <servlet-name>spring-dispatcher</servlet-name>
                <url-pattern>/</url-pattern>
        </servlet-mapping>


</web-app>
```

webApplicationContext.xml

```
<beans xmlns=“http://www.springframework.org/schema/beans”
          xmlns:context=“http://www.springframework.org/schema/context”
          xmlns:mvc=“http://www.springframework.org/schema/mvc”
          xmlns:xsi=“http://www.w3.org/2001/XMLSchema-instance”
          xsi:schemaLocation=“http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
          http://www.springframework.org/schema/context
          http://www.springframework.org/schema/context/spring-context.xsd
          http://www.springframework.org/schema/mvc
          http://www.springframework.org/schema/mvc/spring-mvc.xsd”>

          <!-- Used when not working with annotations

          <bean id=“HandlerMapping”
class=“org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping”/>
          <bean name=“/welcome.html”
class=“com.practice.spring.hellocontroller.HelloController”/>
           -->

           <context:annotation-config/>
           <context:component-scan base-package=“com.scsu.controller”/>
           <mvc:annotation-driven/>
          <!--  <bean id=“viewResolver”
class=“org.springframework.web.servlet.view.UrlBasedViewResolver”>-->
          <bean
class=“org.springframework.web.servlet.view.InternalResourceViewResolver”>
          <property name=“viewClass” value
=“org.springframework.web.servlet.view.JstlView”/>
                    <property name=“prefix” value=“/WEB-INF/jsp/”/>
                    <property name=“suffix” value=.”jsp”/>
          </bean>
          </beans>
```