12-2019

Building Security Aware E-Commerce Web Applications

Sai Teja Reddy Kommuri
skommuri@stcloudstate.edu

Building Security Aware E-Commerce Web Applications

by

Sai Teja Reddy Kommuri

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

December, 2019

Abstract

In the past decade, there has been a rapid increase in Electronic commerce (e-commerce) activity via web applications. With this increase, there is a need for building a good quality application. One of the major factors to achieve quality is the application’s security. This paper discusses about how alarming the threats posed to the e-commerce applications are, how can one counteract against the threats, what are the mistakes made by the current e-commerce applications which enabled them to get affected by the security attacks, how they recovered from the incidents, takes the opinions from the consumers with the help of a survey, proposes a methodology to be followed while developing an e-commerce web application to make the application aware about the threats and take countermeasures accordingly.

Acknowledgement

The effective finish of this paper could not have been conceivable without the direction of my professors Dr. Dennis Guster and Dr. Lynn Collen. I also would like to thank Dr. Sneh Kalia for being part of the committee.

Table of Contents

List of Figures

Chapter I: Introduction

Introduction

With the low cost and high availability of the Internet, there has been a spike in the usage of web applications, may it be with the mobile phones or with the computers. To make use of this, many businesses are making their way into providing e-commerce services. E-Commerce, which stands for Electronic Commerce is a way of making commerce online with the use of internet. This business model has shown a massive growth in the past couple of decades all around the world. There are some businesses which are providing exclusive e-commerce services with zero physical retail business. From the past few years, there have also been debates that there will be a significant loss of business if not concentrated correctly on e-commerce trend. The success or failure of an e-commerce application depends upon numerous numbers of factors of which the business model, the products, customers, investors, data security, transactional security, etc., are some of them.

With the fast growth of internet and e-commerce along with it, there have been a wide variety of criminal attacks and intrusions because of which it is very essential that security of the application must be a part of the application design. If it is not included in the design, then the application team should be constantly addressing new security issues every now and then which make the application untrustworthy. A customer does not use an application if it is not trustworthy. Besides, we should also consider the financial losses when there is a successful launch of an attack on the e-commerce application. Hence, security should be an integral part of an e-commerce application. In a broader perspective, here, security also refers to Confidentiality, Integrity and Authenticity (the CIA triad). To place an order, there will be a numerous number of requests and transactions between the customer and the merchant most of which is private information. So, maintaining confidence should be the utmost priority of the application.

Problem Statement

There have been many studies on the security of e-commerce applications, but they have failed to address the methodology to be followed to implement the security aspects in the applications. It is true that most of the e-commerce applications nowadays are being developed providing security but the cost of implementation is somewhat not economical. Hence, there is a need of a methodology, using which the cost of security in the applications becomes cost effective and time effective.

Nature and Significance of the Problem

Making use of the e-commerce applications makes our busy life easy. But these e-commerce applications should also be reliable in order to use them extensively. All the security threats that are to be discussed in this study poses a big threat to that reliability. It is very important that those threats are addressed in a cost-effective way. So, this study is an effort to analyze those threats, finding a way to address them and suggesting a methodology to be followed while developing an e-commerce application which is intended to counteract with almost every security threat out there in the real world.

Objectives

The major objective of this study is to find the current posing security threats against e-commerce applications, to suggest a way to counteract them and to propose a methodology such that a given e-commerce application which is developed making use of this methodology is aware of almost every security threat against it.

Study Questions

1. What are the different security threats in concerned to the e-commerce applications?
2. What is the difference between internal and external threats?
3. How do you provide secure online payments?
4. How do you prevent fraud?
5. How do you minimize the payment risk?
6. How do you limit financial losses?

Summary

In this chapter, we have seen what e-commerce is and how important it is to provide security to e-commerce applications. It also covered about the problem statement, objectives of the study and study questions. Now let us take discuss about the background of the problem and related literature review in the next chapter.

Chapter II: Background and Review of Literature

Introduction

In this chapter let us discuss about the background of e-commerce, its posing threats in today's world and the different ways to mitigate them, and the literature review from the studies done by various researchers.

Background

E-Commerce application (sometimes also known as e-tail or e-business (Rouse, 2018)) is a virtual place where the commercial transactions like buying and selling takes place in exchange for money. The emergence of e-commerce was possible because of three key developments: emergence of Electronic Data Interchange (EDI) in the 1960s, Electronic Funds Transfer (EFT) in the late 1970s and the major breakthrough of 1991, the Internet (Rouse, 2018). EDI and EFT allowed the companies to transfer the commercial documents electronically in the late 1970s, but the transfer was limited to the local network. With the help of internet, it became possible to transfer those documents globally (History of Ecommerce, n.d.). In late 1990s, the rise of the e-commerce giants eBay and Amazon revolutionized the industry and a lot of e-commerce companies had emerged. Because of the dotcom crash on March 11, 2000, many e-commerce businesses had to get disappeared (History of Ecommerce, n.d.). Fortunately, new e-commerce businesses have emerged after the crash make it possible for the e-commerce industry to make billions of dollars in transactions. The e-commerce services provided by the businesses can be broadly divided into types based on their business models.

1. Business to Consumer (B2C):

In Business to Consumer model, a business organization sells its goods or services to a consumer. For example, buying textiles from an online retailer.

2. Business to Business (B2B):

In Business to business model, a business organization sells its goods or services to another business organization. For example, Amazon gives its cloud services to other businesses for usage purposes.

3. Consumer to Consumer (C2C):

In Consumer to Consumer model, a consumer sells his/her goods or services to another consumer. eBay is the best example for this model where the consumers list their products to sell to the other consumers.

4. Consumer to Business (C2B):

In Consumer to Business model, a consumer sells their own products or services to a business organization. For example, a software developer freelances himself to give his services to a business organization (Sharma, 2016).

For any business model, there are a lot of things which remain common, but for the products or services on which they are doing the commerce by which we can say that the security threats for all the business models remain the same. There is a threat only when there is a vulnerability in the application which the attacker exploits. The possibility of the exploitation of vulnerability is called a threat. There are three places where the vulnerabilities occur.

1. Client-side vulnerabilities
2. Server vulnerabilities
3. Communication Channel vulnerabilities

Client-side vulnerabilities occur with the mistakes or ignorance of the clients. It is the duty of a security professional to anticipate such vulnerabilities and design the system accordingly. These vulnerabilities are often due to the design flaws at the server-side. Some such vulnerabilities are:

1. Concurrency Flaws
2. Cross Site Scripting
3. Weak Authentication Vulnerability
4. SQL Injection
5. Buffer Overflow
6. Broken Session Management
7. Insecure Direct Object References

Communication Channel vulnerabilities plays a major role in the security of any application. Insecure Communications makes the applications lose confidentiality and integrity (Hardin, 2009).

Literature Related to the Problem

In order for an application to be a trustworthy and reliable application in the customers' minds, there are four security aspects in e-commerce applications.

1. Unauthorized Access:

Unauthorized Access is a state when the people who are not intended to access a particular resource gets access to it. In the recent years, most of the data breaches were led caused by unauthorized access. One such data breach is Acer e-commerce website's 2015-2016 data breach. This breach exposed the contact information and payment information of around 34,500 customers. The exposed information included the customer's name, address, credit card number, credit card expiry date, and the three-digit CVV code (Loshin, 2016). With this information exposed to the hackers, the customers could lose their money within no time. As explained in the article by Loshin (2016), this breach went unnoticed for almost an year which might be because of the Acer’s audit time duration which might be once an year.

There is a saying which could be relatable to security that “No one should be trusted no matter what”. Supporting that saying, there are two types of threats caused by unauthorized access.

Internal Threats: Internal threats are very difficult to detect as they operate within the network and are authorized to access the resources. These threats can be the employees, contractors or the freelancers who are given access to the application’s resources (Hau, 2012). These threats are very difficult to identify unless there is a good audit in place.

External Threats: External threats are performed if the network is compromised or because of the application vulnerabilities like weak authentication, broken session management or SQL Injection. Of these, the most common vulnerability exposed by the hackers is weak authentication. Once the hacker gets into the system, they can store, retrieve, edit and remove the data from the system or monitor the transactions being performed between the client and the server or the server and the database.

To prevent unauthorized access, there is a need to incorporate one of the security policies of access control models. There a various access control models but the significant ones are:

Discretionary Access Control: Commonly referred to as DAC, this model allows the subject to change the access types to the objects it owns. This model makes use of the Access Control Lists (ACL) for this purpose. Because of its advantage of being able to change access types whenever needed, this model is widely used in commercial and industrial environments (Hau, 2012).

Mandatory Access Control: Commonly referred to as MAC, this model allows the administrator rather than the subject to change the access types to the objects based on the given security labels. This model is usually implemented in military and hospital environments where the access restrictions are very strict (Hau, 2012).

Role Based Acces Control: Referred to as RBAC, this model allows set of access control rules that are authorized centrally based on the roles of the individuals who are involved (Hau, 2012). Gambhir (n.d.) of Hitachi Soultions discussed about their access control mechanism to reduce security threats in e-commerce applications. He said that their company uses Role Based Acces Control (RBAC) management to achieve it. He gave a classic example of Content Management team who are responsible and limited to the content management who wouldn't need the orders' information, the customers' information, etc. This model can be implemented where there are a lot of teams with different roles are involved.

2. Confidentiality:

Confidentiality is an assurance that the information is not being accessed by the personnel who do not have a legitimate permission and need. The issues related to confidentiality are considered to be the subsets of unauthorized access (Kaur, 2015). To ensure the confidentiality across the system, the information flow direction and the security levels should be addressed with a security model. There are various models for this purpose of which the most significant one is the Bell-LaPadula model. This model uses security levels to decide the access rights based on the following rules:

- The flow of information with higher security level to lower security level is forbidden
- The subjects with higher security levels cannot own (i.e., write) the objects with lower security levels

The changes to the objects' security level can be changed only by the administrator (Hau, 2012) and this model is used in the environments where there is a high need of confidentiality like military.

3. Data Integrity:

Data integrity is an assurance that the data is consistent and correct. In the field of e-commerce, the data that is transferred between two entities (In B2B, between the two businesses, in B2C, between business and the customer and in C2C, between the customers) is subject to integrity checks. There are a numerous number of ways how the data integrity can be compromised. Some of them are:

- Human errors when data is entered
- Errors that occur when data is transmitted between two entities
- Hardware malfunctions, such as disk crashes
- Natural disasters, such as floods and earthquakes
- Exploiting application vulnerability

Here, our intention is to prevent the attackers from the exploiting the application vulnerability affecting the data integrity. To ensure the data integrity, there are various security policies for integrity model, but the significant ones are Biba Model and Clark-Wilson Model (Hau, 2012).

Biba model is the opposite of Bell-LaPadula model focusing on the integrity of the application. The access rights are based on the following rules:

- The flow of information with lower security level to higher security level is forbidden (Hau, 2012)

- The subjects with lower security levels cannot own (i.e., write) the objects with higher security levels (Hau, 2012)

The Clark-Wilson model enforces and certifies the data items and processes to ensure the integrity of the data (Hau, 2012). The core of the model based on the following:

- Well Formed Transactions
- The principle of separation of duties

This model is used to address the security issues in the commercial applications.

4. Non-Repudiation:

To repudiate means to deny. Repudiation is the phenomenon of denying the validity of the resource. Non-Repudiation is the opposite of it i.e., not being able to deny the validity of the resource. In online transactions, non-repudiation makes it possible for a service or an entity in communication to not deny the origin and authenticity of the transaction.

Now, let us discuss about the costs of implementing the security in e-commerce. Many businesses do not consider security while developing the applications. They realize its necessity once they got affected by an attack. For an application, security can be integrated in two ways – design time security, runtime security. Design time security is when we consider the aspects of security right from the beginning of the application. Runtime security is when we provide security to the application when it is in production phase. When the application is in the production phase, it is very difficult and complex to make considerable changes to the application. Also it is not recommended to do so. While in production, if we make changes to the application, there is a chance of data loss as well.

Literature Related to the Methodology

In general, any e-commerce application’s flow is as shown in the following figure:

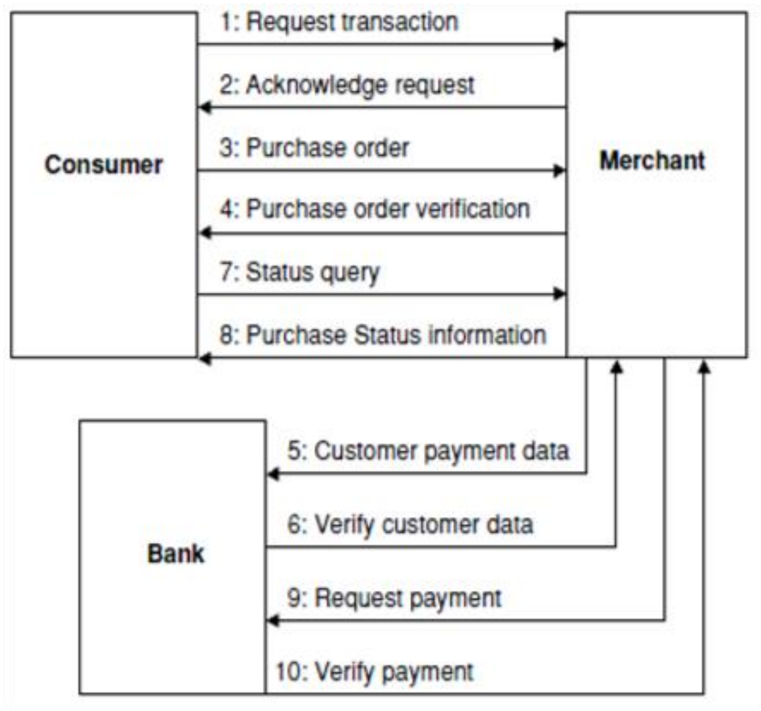


Figure 1. E-Commerce Application's Flow Diagram, reprinted from Transaction Security for Internet E-commerce Application, by Khandare, 2015 Retrieved from http://ijarcsse.com/Before_August_2017/docs/papers/Volume_5/2_February2015/V5I2-0207.pdf

1. In step-1, the consumer logs into his/her account of the e-commerce application or creates a new account and then logs in if the consumer is new, requests for the product/service for transaction
2. In step-2, the merchant acknowledges the request by returning the list of products/services
3. In step-3, the consumer selects the products/services required and requests the merchant for purchase order with the payment information included like the name, credit/debit card number, credit/debit card expiry date, and the three-digit CVV code
4. In step-4, the merchant verifies the purchase order, saves the payment information in the database and acknowledges the consumer

5. In step-5, the merchant transmits the consumer’s payment information for the bank to start the financial transaction
6. In step-6, the bank verifies the consumer’s payment information with the bank’s database and acknowledges the merchant
7. In step-7, the consumer queries for the payment status
8. In step-8, the merchant replies with the verification of the information
9. In step-9, the merchant requests the bank for the payment
10. In step-10, the bank verifies the payment, deducts the money from the consumer’s account and adds it in the merchant’s account and acknowledges the merchant

Let us look at the security concerns beginning with step-1. In this step, there is a chance that the hacker might exploit the weak authentication vulnerability by numerous numbers of methods. Authentication is the process of verifying the user’s information for validation which requires the username/email/user id and password. If this authentication process is weak, hacker can launch a series of attacks to get into the system with false identification. Some of the major concerned authentication attacks are:

Dictionary Attack: In this attack, a program is developed with built in dictionaries which would use all the words in the dictionary in an attempt to find the correct password. Here, the attack will only be successful if the user’s password is a proper dictionary word. In the other words, the attack will fail if the user’s password is not a dictionary word (Authentication and access control attacks, n.d.).

Brute Force Attack: In this attack, the hacker tries to break the password by entering all the possible combinations. This attack makes sense if the number of letters is less than 3. If the length of the password is more than that even though this attack is effective, it proves out to be non-efficient. For the cases where the password is more than 3 letters, the attackers uses a brute force program which tries every possible combination. In the modern days, with processing power of the computers and the cloud resources, the attacker can launch the brute force attack within no time using the software program (Authentication and access control attacks, n.d.).

One way to mitigate with the dictionary attacks and brute force attack, the application can be programmed to limit the number of login attempts. If the user exceeds the limit, we can program the application to lock the user’s account. But there is a bad side for it. A hacker can intentionally make the application to lock the user’s account thus, successfully launching Denial of Service (DoS) attack. The other way of mitigating with dictionary attack is to provide a strong password policy.

To mitigate with DoS, we can program the application to send an e-mail to the user with a URL which is generated randomly and exclusively for this purpose with a preset expiry period (Sawma & Probert).

In 2015, Starbucks’s mobile application was used by the hackers to hack the users’ accounts launching Brute Force attack on the credentials. Then with the feature of autoload which enables the mobile app to reload the money from the users’ bank account or PayPal account, the hackers added money into a gift card which can be used whenever needed by the hacker. This attack was made possible because of a flaw in the application’s architecture and also because of the lack of the implementation of two-factor authentication (Weise, 2015).

Even after these counter measures, if the communications channel is compromised the hacker can launch a complete different set of network attacks. Some of the significant network attacks are

Sniffing attack: Also known as packet sniffing or Man-in-the-middle attack, this attack allows the hacker to eavesdrop on the information which is flowing between the client and the server and the ability to capture the TCP/IP transmissions which contain the username and password information. Then, the hacker can retrieve the credentials from the TCP/IP packet and use them to falsify the authentication. One way to mitigate with this attack is to provide encryption for the communications channel (Authentication and access control attacks, n.d.).

Replay Attack: In this attack, the hacker traps the authentication sequence and of an authorized user and replays it in the same sequence with same set of values. This attack can be mitigated by providing encryption to the communications channel and setting a time-stamp to all the sensitive transmissions which are sent across the network.

Then there is SQL Injection vulnerability. This is a database vulnerability. When a user inputs the username and password, the application sends it to the database server to verify the information. This verification is done using the following SQL query

SELECT * FROM user_table WHERE user_name= ‘’ and password = ‘’;

The input which the users enters comes into the single quotes. So, when a malicious user who either knows the username or doesn’t know it enters the below strings, he/she could break in to the application by tricking the database server.

correct_username’ AND ‘1’ = ‘1’ -- //To get into the system impersonating a user

‘ OR ‘1’ = ‘1’ -- //If the malicious user does not know any username

SQL injection can be mitigated in two ways both of which are validations. One is client-side validation and the other is server-side validation. In client-side validation, the input text should be validated such that it does not contain any meta characters. In server-side validation, the input string which is received from the client-side will be validated with a set of validation methods which may be predefined, or user defined. If the validation mechanism throws any errors, then the user must be notified with an appropriate message.

In the beginning of March 2014, eBay which is one of the C2C giants only next to Alibaba, noticed a database session which was scanning all the password files which was later announced by the company officials that close to 120 million users' credentials and personal information might have been compromised. This breach was caused by a phishing scam where the hacker scammed one of the company's employee whose account has the access to this information. The hacker catching hold of the employee's credentials, thus entered into the system and created the breach. This breach wouldn't have occurred if the company had two-factor authentication imposed on its employees' accounts or if they had encrypted the users' information using some of the best algorithms out there (Kh, 2014).

In step 3, when a particular user selects a product/service which is very limited in quantity and another user who also selects the same product/service simultaneously and one of the users selects and orders the whole quantity but still the other user will be able to place an order. When it come to the delivery of goods, one of the consumers could not get the product/service. In this entire process, the fault lies in the server which could not keep track of the inventory due to the concurrency flaw. This vulnerability can be exploited by malicious users. Once such exploitation is shopping cart exploitation. In this exploitation, the malicious user opens the applications in two different pages simultaneously, selects the quantity of a product/service, updates the shopping cart, goes to another page where the quantity is not updated yet, selects another quantity and checks out without updating the cart this time where the malicious user has successfully placed the order of greater quantity for lesser price. We can mitigate with concurrency flaws by incorporating the concurrency control techniques such as multithreading, locks, thread synchronization, etc. (Vogel, 2016)

Again in steps 3 and 5, if the network is compromised, there is a possibility of attackers launching sniffing and replay attacks. To mitigate these attacks we can encrypt the transactions and set the timestamps as discussed above. But for the protection of Payment Gateway, there is a compliance standard which we can incorporate into the e-commerce application called PCI DSS compliance which stands for Payment Card Industry Data Security Standard. PCI DSS consists of twelve requirements which are categorized into six goals which are as follows:

Goal-1: Build and Maintain a Secure Network

1. Install maintain a firewall configuration to protect cardholder data
2. Do not use vendor-supplied defaults for system passwords and other security parameters

Goal-2: Protect Cardholder Data

3. Protect stored cardholder data
4. Encrypt transmission of cardholder data across open, public networks

Goal-3: Maintain a Vulnerability Management Program

5. Use and regularly update anti-virus software
6. Develop and maintain secure system and applications

Goal-4: Implement Strong Access Control Measures

7. Restrict access to cardholder data by business need-to-know
8. Assign a unique ID to each person with computer access
9. Restrict physical access to cardholder data

Goal-5: Regularly Monitor and Test Networks

10. Track and monitor all access to network resources and cardholder data

11. Regularly test security systems and processes

Goal-6: Maintain an Information Security Policy

12. Maintain a policy that addresses information security

By maintaining PCI DSS compliance, the e-commerce applications can make sure that they are handling the payment gateway vulnerabilities (Maintaining Payment Security, n.d.)

Apart from these, there are coding vulnerabilities too which could turn out be biggest the threats. There are two significant coding vulnerabilities which the developer should keep in mind while developing the applications. They are

Buffer Overflow: It is a condition when the program tries to store more data in the buffer than it can hold. When the program overflows the buffer, the data is stored forcefully into the memory past the buffer. Doing so could corrupt the data of another program or crash the program of which the memory belongs to. There is also a possibility to execute malicious code using this memory. To mitigate this vulnerability, we have check for the upper boundary and lower boundary and send an error message if the input is out of the boundary values (Buffer Overflow, 2016).

Cross Site Scripting: Commonly referred as XSS, this is one of the top vulnerabilities of web applications. It occurs when malicious scripts are injected from an untrusted source into a trusted source. This usually occurs when the attacker sends the malicious code in the form of a browser side script. To mitigate XSS attacks, the sensitive information should not be transmitted with get requests and also validating the input texts helps reducing the probability of XSS attacks (Cross-site Scripting (XSS), 2018).

Summary

In this chapter, we have discussed about the background of e-commerce and the information which we must know about e-commerce security like different security vulnerabilities of an e-commerce application, different aspects of e-commerce security, different security attacks in each step of the application flow and various security policies in order to create a methodology which enables the e-commerce applications to mitigate with almost every security attack.

Chapter III: Methodology

Introduction

In this chapter we will be discussing the methodology about how we make an e-commerce application aware of most of the security risks. We shall also discuss about the hardware and software requirements in order to run this security aware e-commerce application.

Design of the Study

The approach that we are going to use is a quantitative study. To begin with, we shall learn about how the vulnerabilities mentioned in the previous chapter affect an e-commerce application and how we mitigate them in our implementation. We would be needing some tools and technologies using which we shall be able to do the desired implementation. Once we acquire them, we can start our implementation.

Hardware and Software Requirements

- Operating System: Linux (Ubuntu 16.04)
- Hard Disk Volume: a minimum of 30 GB
- RAM: a minimum of 8 GB
- Server-Side Technologies: JDK 1.8, Spring Boot, Gradle
- Client-Side Technologies: ReactJS, Material UI
- Hosting Platform: AWS EC2
- Storage Platform: AWS S3
- Databases: PostgreSQL, MongoDB
- Version Control and Source Code Management: GitHub
- IDEs: Visual Studio Code, IntelliJ (IDEA)
- Load Balancer: NGINX

Budget

Most of the technologies used are Open Source Technologies. So, there is no need of investing too much to build the application in context. The application will be hosted on AWS cloud using its free tier instance EC2. The IDE IntelliJ which will be used is the Enterprise version which costs around $150. But as student, it can be acquired with a free subscription. The PostgreSQL database service which will be used in the application is hosted on GCP's free tier service which is free up to an amount of $300 in spending for a year and MongoDB Atlas's M0 Sandbox cluster which is also a free tier hosted in GCP. The only real expense would be the domain names. Two domains name servers are required for the development of the application. One for the Front-End of our application and one for the API.

1. cargaraage.now.sh - $0.99
2. saiteja.dev - $12.00

System Design

As an e-commerce application cannot be made as a static application, it would need two implementations; Client-side implementation for the Front-end (UI) and the server-side implementation which does all the heavy lifting of the application, including the business logic because of which server-side implementation may sometimes be referred to as business layer. For this project, we will be using ReactJS for front-end and Spring Boot Java for back-end.

The reasons for choosing ReactJS over the other front-end technologies are that it facilitates the writing for reusable components which can be used in or alongside the other components. This concept yields productivity and facilitates quick maintenance by isolating the root cause of a problem. Moreover, it is a light weight library which renders on a virtual DOM unlike Angular which renders on the regular DOM which increases the performance of the UI.

One of the biggest advantages of using ReactJS is that it is SEO friendly, i.e., it enables the search engines like Google, Yahoo, Bing, etc., to find your application content. For e-commerce applications, it necessary that these search engines find your content to attract the potential customers.

The reasons for choosing Java are its Object-Oriented Programming (OOPs) features which avoids most of the coding risks just with their concepts. One such risk is **buffer overload**. This risk is mitigated by one of its features of garbage collection. The reasons for choosing Spring Boot Java for the server-side implementation is that Spring Boot makes it easy to create stand-alone applications which have production quality builds, it is easy to configure the 3rd party libraries and it enables the avoidance of writing lots of boilerplate code. The packaging will be done using Gradle.

High Level Design

There are several components involved in the application which are aligned as shown in the below high-level diagram:

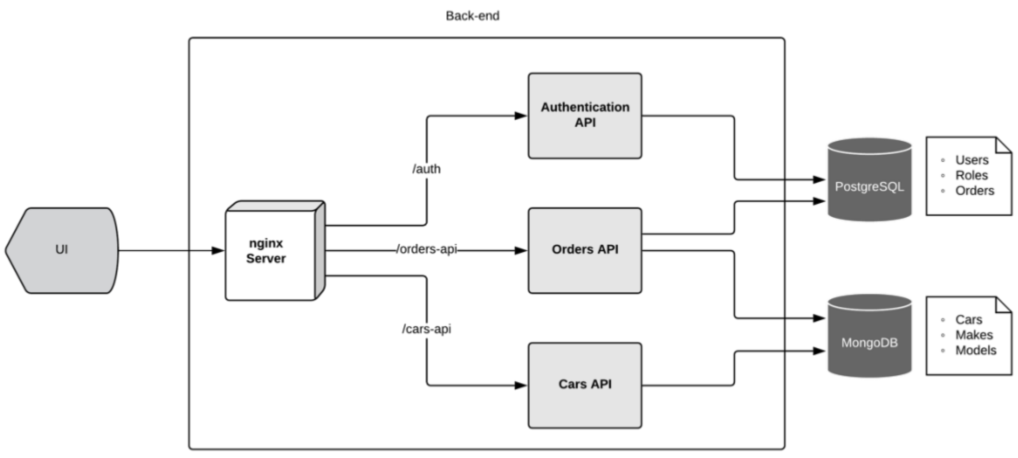


Figure 2. High level design of the system

UI

It facilitates the interaction between the customer and the application. This is the place where a potential customer can sign up, login, view the product(s), add the product(s) to the cart, and place order(s).

Nginx Server

It acts as a proxy server, load balancer, redirects all the HTTP requests to HTTPS to make it a secure connection, and redirects the requests to appropriate services running in the server.

Authentication API

This service authenticates the user based on the user's username and password, generates a JWT token with the user's information such as username, user id, email, first name, last name, full name, status of the user's account, roles of the user and token expiry time. It also creates a new user and returns a JWT token with the above-mentioned information.

Orders API

This service takes the request for a new order with payment information in it, verifies the payment details, places the order and returns the order id as a response.

Cars API

This service is a repository of cars, its makes and body styles. You can create a new car, update and existing car details, create a new make, create a new body style, etc.

PostgreSQL

PostgreSQL is a free open source RDBMS. In this application, the components Authentication API and Orders API use this database which has the following tables in it

- User

- Roles
- Order
- Order_Details
- Roles_Users
- Shipping_Address

The tables user and roles have many-to-many relationship i.e., a single user can have multiple roles and a single role can have multiple users in it. In order to map this relationship, the table roles_users is created. The mapping will be based on the column user_id (foreign key to the column user.id) and role_id (foreign key to the column roles.id).

The tables order and user have many-to-one relationship i.e., a single user can have multiple orders but a single order can have only one user. They are mapped based on the column order.user_id (foreign key to user.id).

The tables order and shipping have one-to-one unidirectional relationship i.e., a single order can have only one shipping address but the shipping address can be repeated for another order. They are mapped based on the column order. shipping_address_id (foreign key to shipping_address.id).

The tables order_details and order have many-to-one relationship i.e., one order can have multiple order details. They are mapped with the column order_details.order_id (foreign key to order.id).

The tables order_details and user have many-to-one relationship i.e., one user can have multiple order details. They are mapped with the column order_details.user_id (foreign key to user.id).

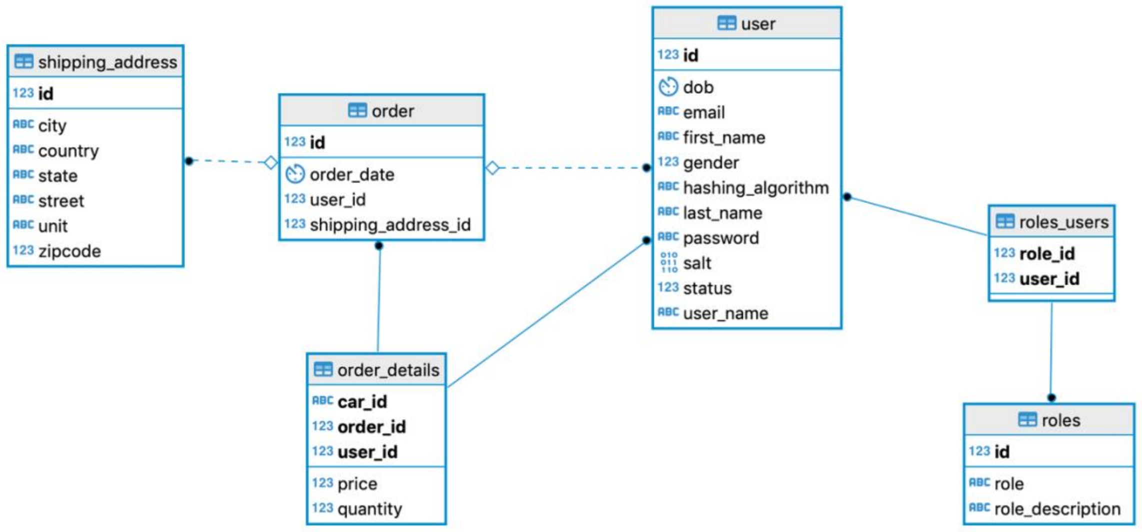


Figure 3: Entity-Relationship diagram of all the tables used from RDBMS

MongoDB

MongoDB is an open source non-relational database classified as NoSQL database. It stores the data in the form of JSON-like documents. The collections in MongoDB are like tables and documents are like records in a traditional RDBMS. It does not have a defined schema. So, one can freely add or remove fields to a document; there is no restriction. MongoDB by default provides a primary key for a document which is of type String. In this application, the components Orders API and Cars API use MongoDB which has the following collections in it:

- Cars
- Makes
- Body_Styles

Setting up the AWS cloud services

To begin with, AWS cloud services could be used either as a free tier for the first-time users for one year or paid tier. Once an AWS account is created from the URL https://portal.aws.amazon.com/billing/signup, an IAM user must be created using the following steps:

1. Navigate to https://console.aws.amazon.com/iam/
2. In the navigation pane, select *Users* < *Add User*
3. Enter a *User Name* (Administrator is a recommended user name)
4. Select *AWS Management Console Access* < *Custom Password* and enter a password
5. Uncheck the box *User must create a new password at next sign-in,* as a password is already created
6. Select *Next: Permissions* > *Set permissions* > *Add user to group* > *Create group*
7. In the dialog box, enter a group name (Administrators is recommended)
8. Select *Filter policies* > *AWS managed -job function* to filter the table contents.
9. In the policy list, select *AdministratorAccess* > *Create group*

After creating the IAM user, a key pair should be generated in order to access the EC2 instances from the remote terminal. The following steps can be followed to create a new key pair. This would be the private key of the user who is just created.

1. From the AWS dashboard, select EC2 to open the Amazon EC2 management console.

2. From the navigation bar, select a region for the key pair

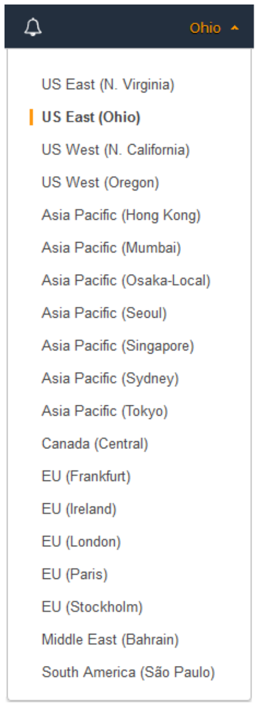


Figure 4. List of selectable regions available in the EC2 instance

3. In the navigation pane, go to *NETWORK & SECURITY* > *Key Pairs*

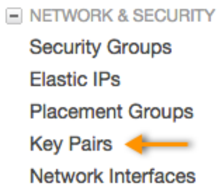


Figure 5. Network & Security's navigation pane in EC2 instance

4. Select *Create Key Pair*

5. Enter a name for the new key pair in the *Key pair name* field and select *Create*

6. The private key file will be downloaded automatically by your browser (The format of the private key file will be .pem)

7. Once the file gets download, open the terminal, go to the file location and perform the command "chmod 400 <*user_name*>-key-pair-<*selected_region_name*>. pem"

Once the IAM user and the key pair is created, proceed to launch a new EC2 instance. It can be created by the following steps

1. Navigate to https://console.aws.amazon.com/ec2/
2. Select *Launch instance*
    a. *Choose an Amazon Machine Image:* Select a desired image (for example, Ubuntu Server 18.04 LTS
    b. *Choose Instance Type:* Select the desired instance type from the given list and proceed to *Next: Configure Instance Details*
    c. *Configure Instance Details:* Enter the desired configuration details and select *Review and Launch* or *Add Storage*
3. In the Review page click *Launch* which will launch the instance

This created instance can be accessed from a remote terminal or an SSH client like PuTTY using the private key which is downloaded. From the remote terminal the instance can be accessed by executing the command “ssh -i "<file-location>/<private-key-name>.pem" <username>@<instance-public-address>”

Now, set up an S3 storage bucket by following the steps below

1. Navigate to the URL https://console.aws.amazon.com/s3/
2. Select *Create bucket*
3. Enter the name of the bucket in the *Bucket name* field (should be a DNS-compliant name)
4. Select the desired *region* where the bucket should be created and then select *Create*

Setting up the PostgreSQL server on GCP

Create a new GCP account either a trial version or a paid version. Once the account is created, navigate to the GCP console and select *SQL* in the navigation pane. Select *Create an*

Instance. Select one of the database engines displayed (at this moment, the available database engines are MySQL, PostgreSQL, SQL Server). Enter the database instance information and click *Create*.

Setting up MongoDB server

To set up a MongoDB server, create an account in MongoDB Atlas. Once an account is created, create a new project. After creating the project, follow the steps below

1. Select *Build a New Cluster*
2. Select a *Cloud provider* from the given list (at this moment there are 3 providers available; namely AWS, GCP, Azure)
3. Select a *Region*
4. Select a *Cluster Tier* from the list of available tiers (for the first-time user, M0 sandbox cluster offers a free tier for the lifetime)
5. Enter the *Cluster name*
6. Select *Create Cluster*

Implementing the backend Code

All the resources in the backend implementation are exposed to the world using REST web services. Unlike the other web services such as SOAP, REST is not a protocol; it is an architecture. These web services are stateless. So, the client must keep the session state alive and send all the necessary information in the request and wait for the response to proceed further. REST adheres to the HTTP standards, thus implements all the concepts of HTTP. All the HTTP methods are implemented and GET, POST, DELETE, PUT, PATCH, etc., are the most commonly used methods. GET is used when a resource is to be retrieved i.e., READ, POST is used when a resource has to be created,

Cars API

The Cars API deals with the repository of cars. The cars are usually differentiated based on the body style. Some of the examples of body style are Sedan. SUV, Coupe, etc. This is considered as a resource and it has an end point “/bodyStyles”.

Authentication API

Authentication API mainly has two functionalities; namely, authenticating a user, and registering a new user. In the process of authenticating a user, the API retrieves the username and password. It then queries the database table ‘USER’ with that username. If the username is found, it will verify the password with the one the user has provided. If anyone of them do no match, it will say that the user is not found. In the process of registering a new user, it will first persist the user in ‘USER’ table, then assigns the user to the role ‘User’. This user to role relationship will be kept maintained in the table ‘USER_ROLES’.

Orders API

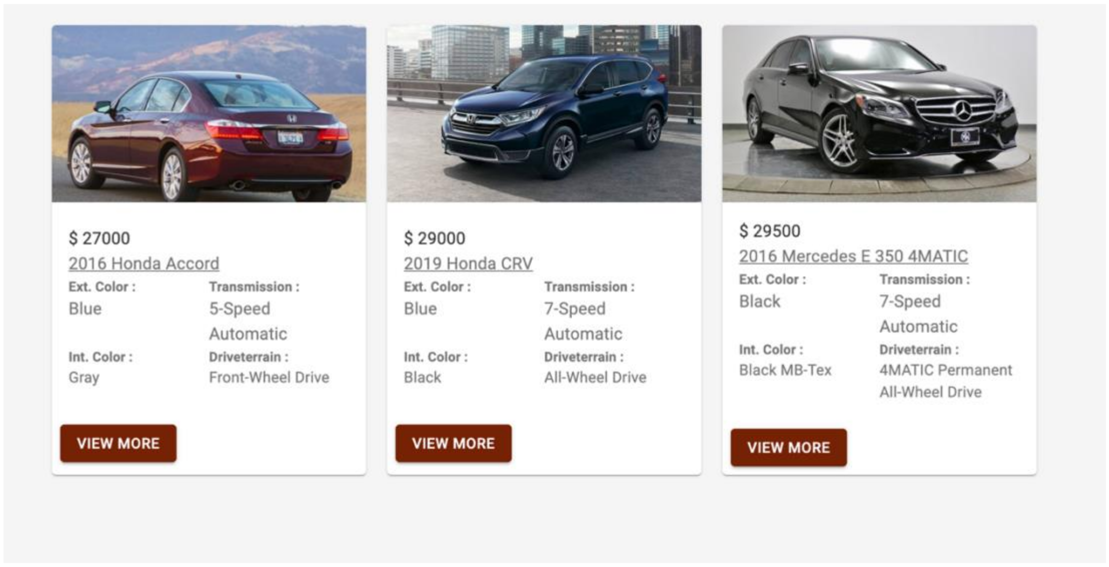
Orders API deals with two functionalities. One of them being creating an order and the other one is retrieving an order. While creating an order, the API retrieves the payment details, shipping details and the order details. First, it will verify the payment details. If the payment details are good, then it will proceed to place the order. In the process of creating the order, it will add the shipping details to the order and then persist it. Once the order is saved, the API will respond back with the order number. This order number can be used to retrieve the order.

Implementing the UI

The UI has several modules which include the following

Cars for Sale

In this module, the end-user can view a list of cars which are available for sale. Each car is made with a component called Card. This Card has several child components such as picture of the car, price in USD, exterior color and interior color of the car, transmission, drive terrain and an option to view more details of the car.



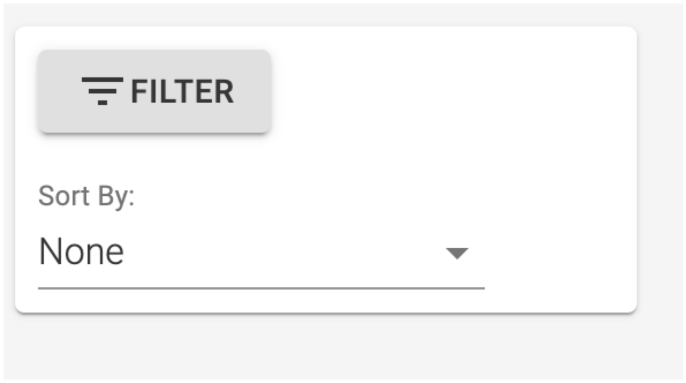
$ 27000
2016 Honda Accord
Ext. Color : Blue
Transmission : 5-Speed Automatic
Int. Color : Gray
Driveterrain : Front-Wheel Drive

$ 29000
2019 Honda CRV
Ext. Color : Blue
Transmission : 7-Speed Automatic
Int. Color : Black
Driveterrain : All-Wheel Drive

$ 29500
2016 Mercedes E 350 4MATIC
Ext. Color : Black
Transmission : 7-Speed Automatic
Int. Color : Black MB-Tex
Driveterrain : 4MATIC Permanent All-Wheel Drive

Figure 6. List of Cars in the built application

There is one more module where the end-user can filter the cars as well as sort them.



Sort By: None

Figure 7. Options to Filter and Sort the cars in the application

In the filter, the end-user can filter out the cars based on its make, body style, and price.

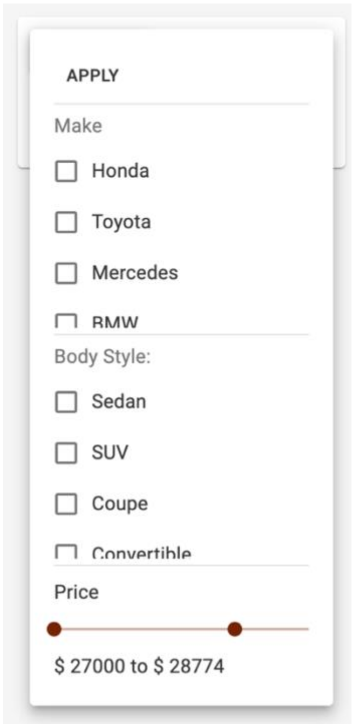


Figure 8. Filter options in the application

The user can sort the cars based on the order or makes or body styles, newest car, oldest car, highest to lowest price, and lowest to highest price.

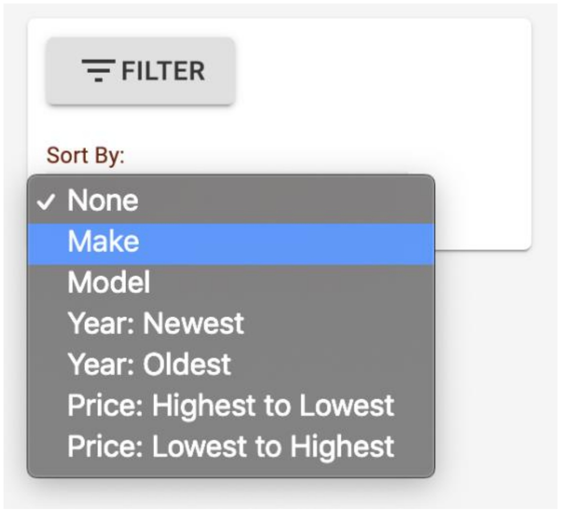


Figure 9. Sort options in the application

View Car

The end-user can navigate to this module by clicking on the *VIEW MORE* option inside the card in the cars for sale module where he/she can view the selected car in detail, multiple images of the car, add the car to the cart, and checkout.

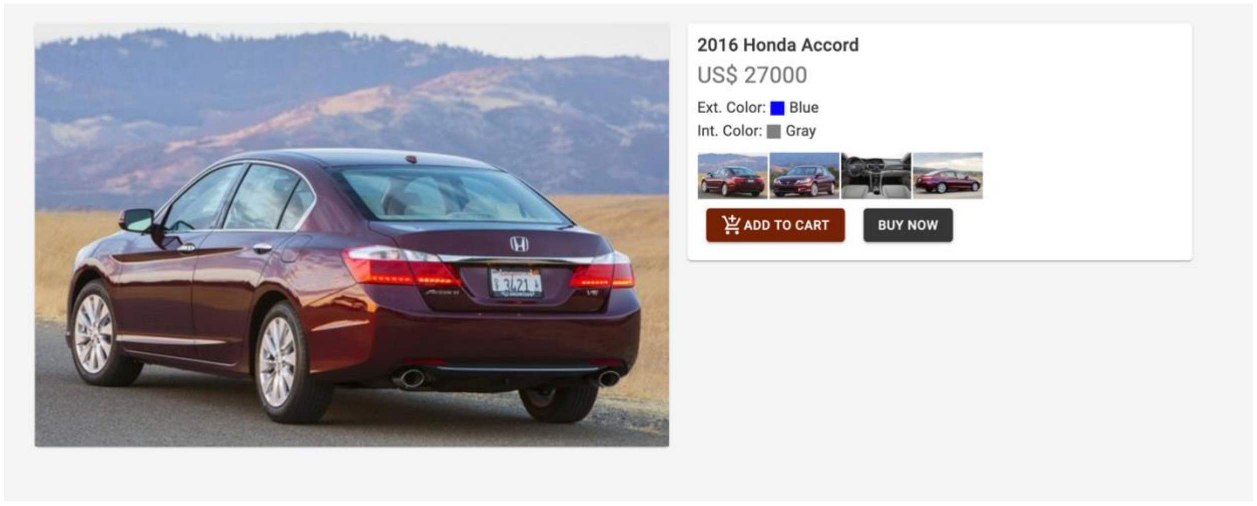


Figure 10. Details of a selected car in the application

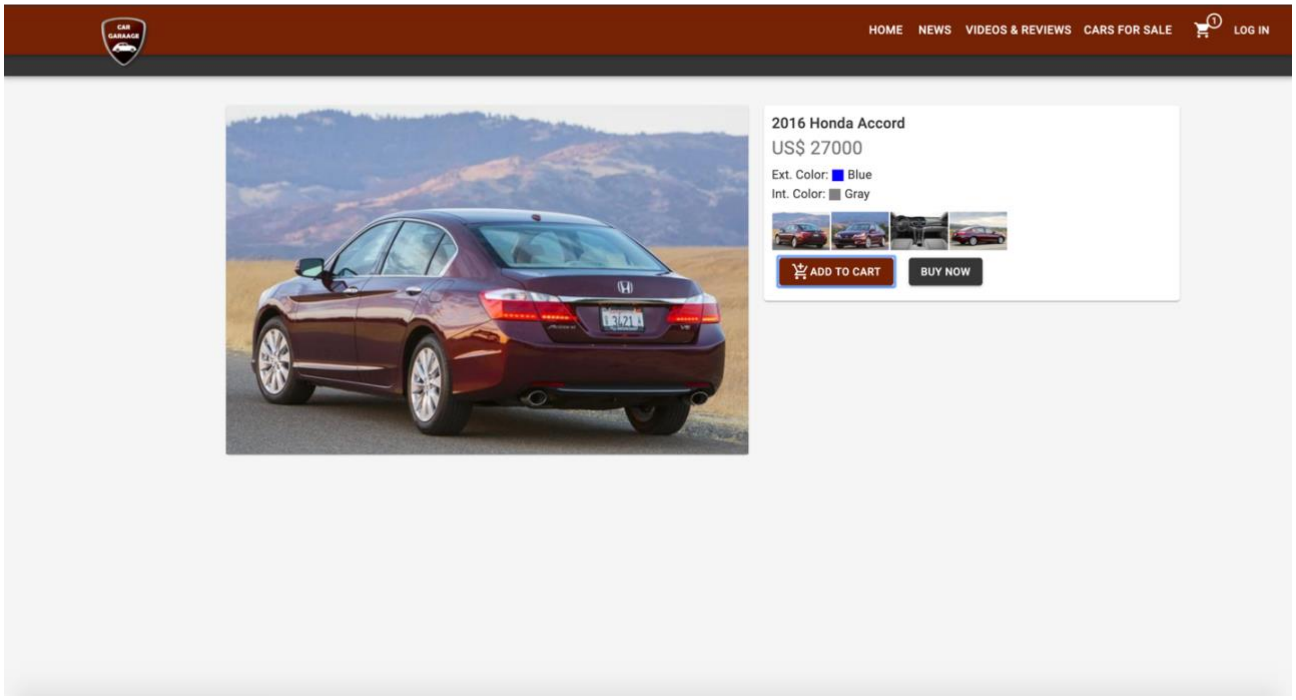


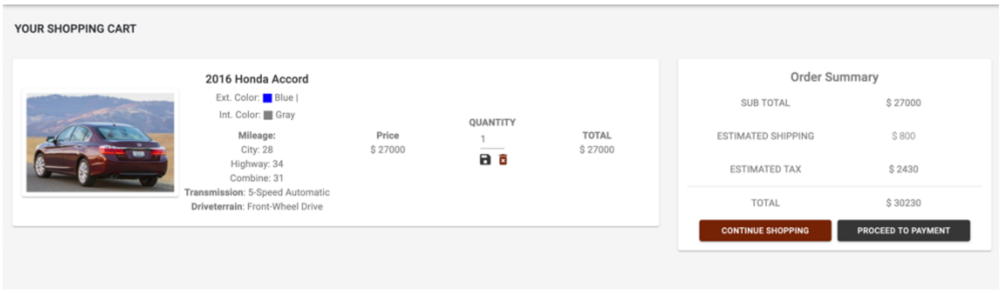
Figure 11. Car added to the cart

If the end-user tries to check out directly from this module, it first adds the car to the cart and takes the user to the checkout module.

Cart

The end-user can navigate to this module by clicking on the cart icon in the navigation bar where all the items which the user have added in the intention to buy are displayed in this module. There is also an option to change the quantity of items, and remove the item from the cart.

The Order summary shows the total price of all the items in the cart, estimated shipping cost, estimated tax, and the grand total at the bottom. The user can review this information and proceed to the check-out module.



2016 Honda Accord
Ext. Color: 🟦 Blue |
Int. Color: ⬛ Gray
Mileage:
City: 28
Highway: 34
Combine: 31
Transmission: 5-Speed Automatic
Driveterrain: Front-Wheel Drive

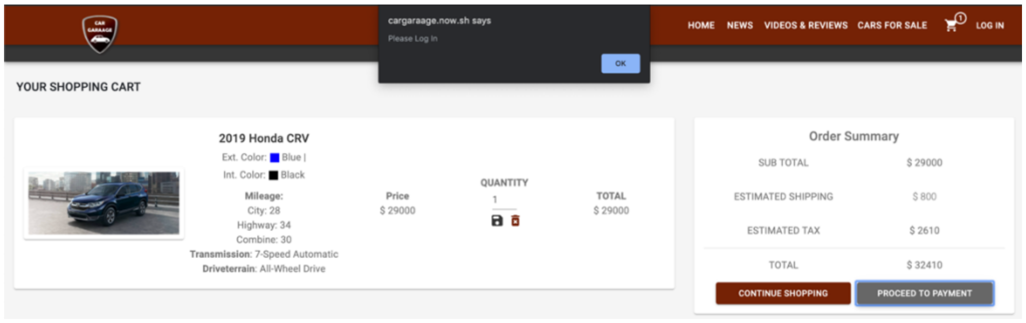
Price	QUANTITY	TOTAL
$ 27000	1	$ 27000

Order Summary

SUB TOTAL	$ 27000
ESTIMATED SHIPPING	$ 800
ESTIMATED TAX	$ 2430
TOTAL	$ 30230

Figure 12. Cart module

If the end-user is not logged in, it will prompt to Log in, thus restricting the unauthenticated end-user to check out the item.



cargaraage.now.sh says
Please Log In

2019 Honda CRV
Ext. Color: 🟦 Blue |
Int. Color: ⬛ Black
Mileage:
City: 28
Highway: 34
Combine: 30
Transmission: 7-Speed Automatic
Driveterrain: All-Wheel Drive

Price	QUANTITY	TOTAL
$ 29000	1	$ 29000

Order Summary

SUB TOTAL	$ 29000
ESTIMATED SHIPPING	$ 800
ESTIMATED TAX	$ 2610
TOTAL	$ 32410

Figure 13. Unauthenticated checkout attempt

Check Out

The end-user can navigate to the checkout from either the Cart module or from the view car module. It has three important components in it.

1. Payment Details

   In this component, the end-user can enter the payment information like Credit/Debit card information, Expiry Date, three-digit CVC number, and Name on Card.

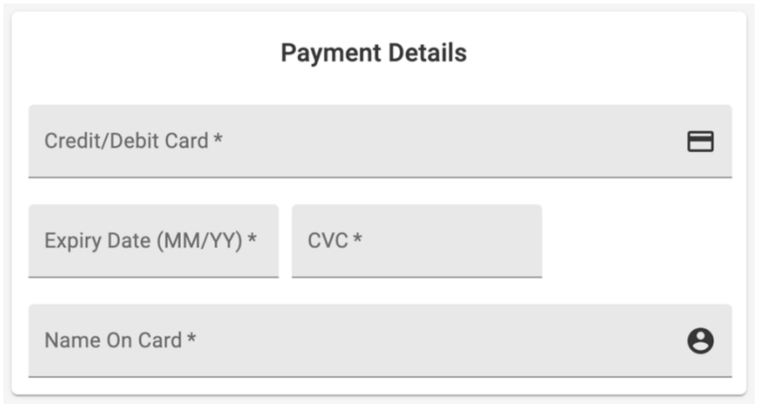


Figure 14. Payment details form

2. Shipping Address

   In this component, the end-user can enter the shipping address details to where he/she intend the order to be shipped to.

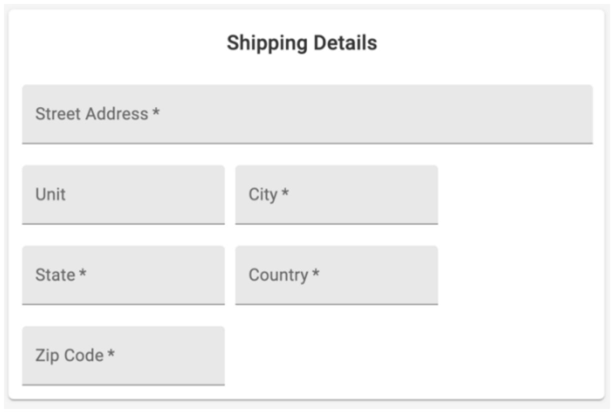
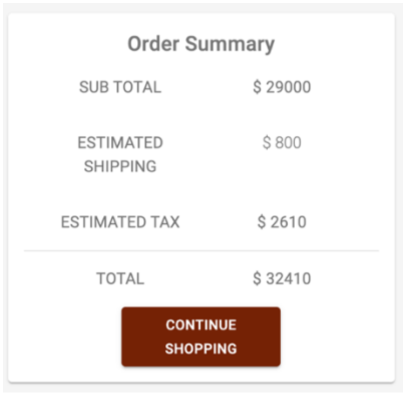


Figure 15. Shipping details form

3. Order Summary

   Order summary has the details of the total price of all the items in the cart, estimated shipping cost, estimated tax, and the grand total. If the end-user decides to continue the shopping, it provides that option.



Order Summary

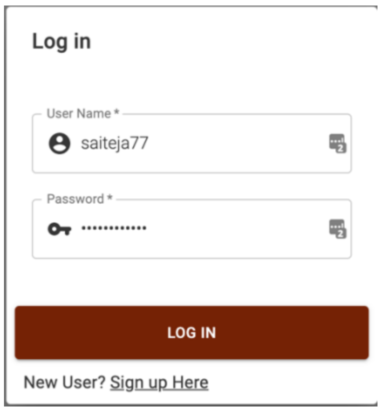
SUB TOTAL	$ 29000
ESTIMATED SHIPPING	$ 800
ESTIMATED TAX	$ 2610
TOTAL	$ 32410

Figure 16. Order summary

If the user is not logged in, it will redirect the user to the Unauthorized page.

Log In

The end-user can get authenticated by logging in to the application from here by enter the username and password. There is also an option provided to new users to get registered by redirecting them to the registration module by clicking on “Sign up Here”.



Log in

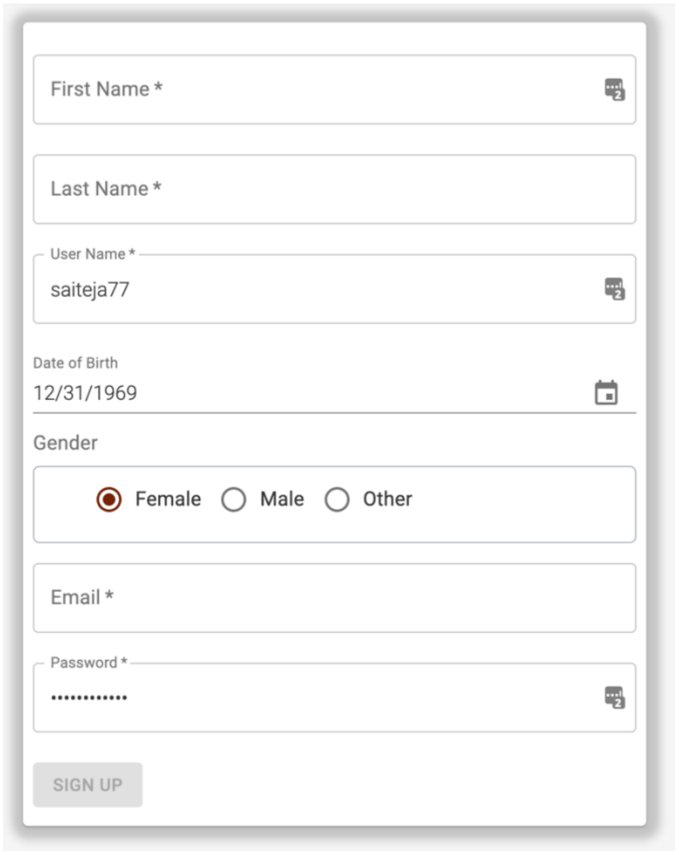
User Name * saiteja77

Password * ••••••••••••

Figure 17. Log in module of the application

Registration

If the end-user is not registered with the application, this module provides an interface to get registered. In order to get registered successfully, the end-user should enter the first name, last name, user name, gender, email address, and password. The end-user can also enter the date of birth but it is optional.


First Name *

Last Name *

User Name *
saiteja77

Date of Birth
12/31/1969

Gender
[✓ Female] Male Other

Email *

Password *
••••••••••••

Figure 18. Registration module

Summary

In this chapter, I have discussed about the study design, what are the software and hardware requirements to build the application, cost of implementing, installations, how to develop each module and its implementation.

Chapter IV: Data Analysis

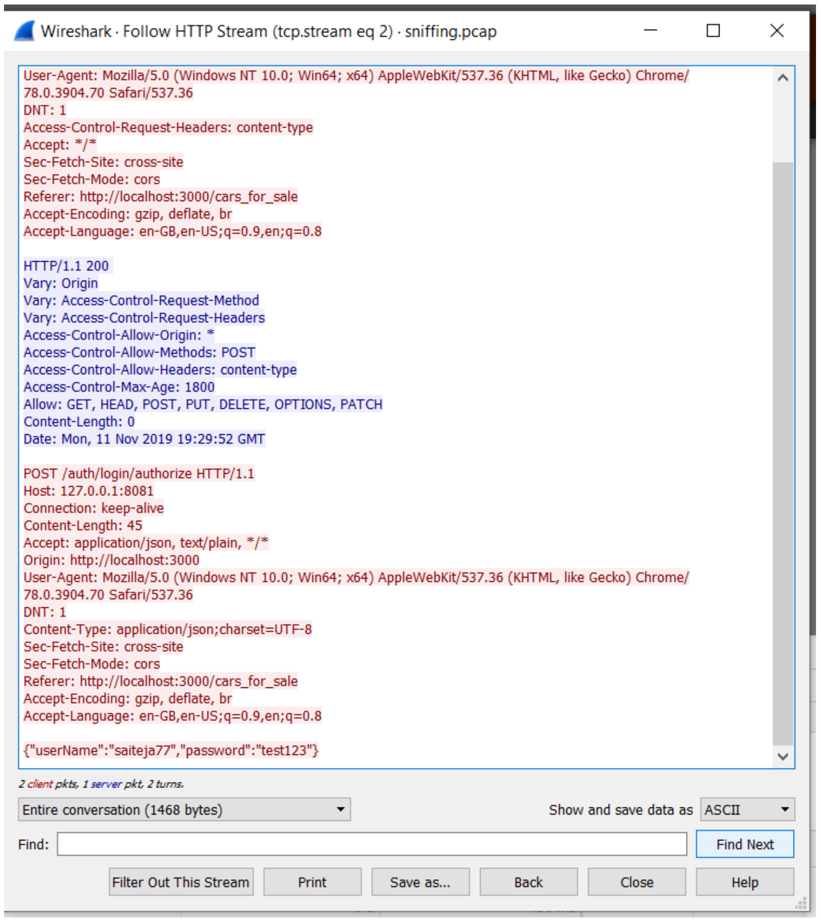
Introduction

In this chapter, the security of the built application will be discussed.

Analysis

Secure Communication channel

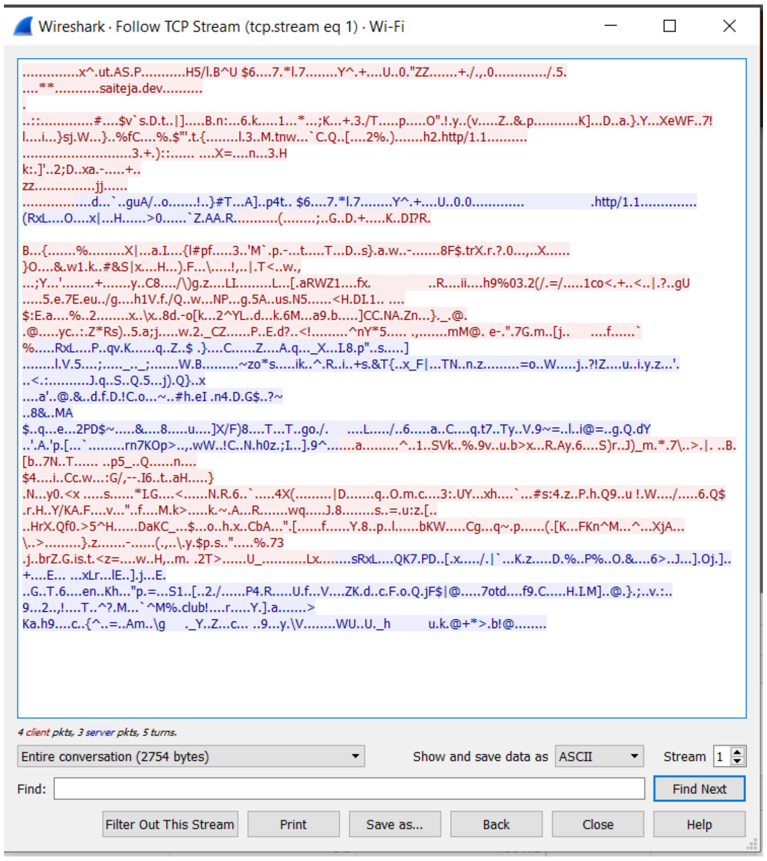
Starting off, let us see how secure the communication channel of the application is. For that purpose, first let us see what would happen if the communication channel is not secured. From the UI, I have tried to log in to the application. The UI makes a REST call to the Authentication API through an insecure channel exposing the payload as shown in the figure 19.


Wireshark · Follow HTTP Stream (tcp.stream eq 2) · sniffing.pcap
2 client pkts, 1 server pkt, 2 turns.
Entire conversation (1468 bytes)
Show and save data as
ASCII
Find:
Find Next
Filter Out This Stream
Print
Save as...
Back
Close
Help

```
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
78.0.3904.70 Safari/537.36
DNT: 1
Access-Control-Request-Headers: content-type
Accept: */*
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: cors
Referer: http://localhost:3000/cars_for_sale
Accept-Encoding: gzip, deflate, br
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8

HTTP/1.1 200
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: POST
Access-Control-Allow-Headers: content-type
Access-Control-Max-Age: 1800
Allow: GET, HEAD, POST, PUT, DELETE, OPTIONS, PATCH
Content-Length: 0
Date: Mon, 11 Nov 2019 19:29:52 GMT

POST /auth/login/authorize HTTP/1.1
Host: 127.0.0.1:8081
Connection: keep-alive
Content-Length: 45
Accept: application/json, text/plain, */*
Origin: http://localhost:3000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
78.0.3904.70 Safari/537.36
DNT: 1
Content-Type: application/json;charset=UTF-8
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: cors
Referer: http://localhost:3000/cars_for_sale
Accept-Encoding: gzip, deflate, br
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8

{"userName":"saiteja77","password":"test123"}
```

Figure 19. Packet capture of communication through insecure channel

The data from figure 19 is retrieved by capturing the tcpdump. If we notice the last line of the captured data, we can see that the payload is visible in plain text. With this information, the attacker can mimic the user to perform malicious activities. This threat to the application is mitigated by securing the communication channel with the help of Secure Socket Layer (SSL). For this, I have generated an SSL certificate in the server with the respective DNS names of UI and API and got them signed with a Certification Authority (CA) called “Let’s Encrypt”. All the communications made through SSL is secure as it encrypts all of them. Once the server receives the encrypted data, it knows how to decrypt it with its private key. Once again, the authentication request is made by the UI to Authentication API, but this time in a secure channel. In figure 20, we can see how the communication is encrypted in intelligible form.



```
..............x^.ut.AS.P...........H5/l.B^U $6....7.*l.7........Y^.+....U..0."ZZ.......+./.,.0............./.5.
....**...........saiteja.dev..........
.
..::.............#....$v`s.D.t..|].....B.n:...6.k.....1...*...;K...+.3./T.....p.....O".!.y..(v.....Z..&.p...........K]...D..a.}.Y...XeWF..7!
l....i...}sj.W...}..%fC....%.$'".t.{........l.3..M.tnw...`C.Q..[....2%.).......h2.http/1.1..........
..........................3.+.)::...... ....X=....n...3.H
k:.]'..2;D..xa.-.....+..
zz...............jj......
................d...`..guA/..o.......!..}#T...A]..p4t.. $6....7.*l.7........Y^.+....U..0.0.............          .http/1.1.............
(RxL....O....x|...H......>0......`Z.AA.R..........(........;..G..D.+.....K..DI?R.

B...{.......%.........X|...a.I....{l#pf.....3..'M`.p.-...t.....T...D..s}.a.w..-.......8F$.trX.r.?.0...,..X......
}O....&.w1.k..#&S|x....H...).F...\.....!,..|.T<..w.,
...;Y...'........+.......y..C8..../\)g.z....LI.........L...[.aRWZ1....fx.          ..R....ii....h9%03.2(/.=/.....1co<.+..<..|.?..gU
.....5.e.7E.eu../g....h1V.f./Q..w...NP...g.5A..us.N5......<H.DI.1.. ....
$:E.a....%..2........x..\x..8d.-o[k...2^YL..d...k.6M...a9.b.....]CC.NA.Zn...}._.@.
.@.....yc..:.Z*Rs)..5.a;j.....w.2._CZ......P..E.d?..<!.........^nY*5..... .,.......mM@. e-.".7G.m..[j..     ....f......`
%.....RxL....P..qv.K......q..Z..$ .}....C......Z....A.q..._X...I.8.p"..s.....]
........l.V.5....;....._.._;.......W.B.........~zo*s.....ik..^.R..i..+s.&T{..x_F|...TN..n.z.........=o..W.....j..?!Z....u..i.y.z...'.
..<.:..........J.q..S..Q.5...j).Q}..x
....a'..@.&..d.f.D.!C.o...~..#h.eI .n4.D.G$..?~
..8&..MA
$..q...e...2PD$~.....&....8.....u....]X/F)8....T...T..go./.    ....L...../..6.....a..C....q.t7..Ty..V.9~=..l..i@=..g.Q.dY
..'.A.'p.[...`.........rn7KOp>..,.wW..!C..N.h0z.;I...].9^.......a.........^..1..SVk..%.9v..u.b>x...R.Ay.6....S)r..J)_m.*.7\..>.|. ..B.
[b..7N..T...... ..p5_..Q......n....
$4....i..Cc.w...:G/,--.I6..t..aH.....}
.N...y0.<x .....s......*I.G....<......N.R.6..`.....4X(.........|D.......q..O.m.c....3:.UY...xh....`...#s:4.z..P.h.Q9..u !.W..../.....6.Q$
.r.H..Y/KA.F....v...."..f....M.k>.....k.~.A...R.......wq.....J.8........s..=.u:z.[..
..HrX.Qf0.>5^H......DaKC_...$...o..h.x..CbA...".[......f......Y.8..p..l......bKW.....Cg...q~.p......(.[K...FKn^M...^...XjA...
\..>.........}.z.......-......(.,..\.y.$p.s.."....%.73
.j..brZ.G.is.t.<z=....w..H,..m. .2T>......U_...........Lx.........sRxL....QK7.PD..[.x...../.|`...K.z.....D.%..P%..O.&....6>..J...].Oj.]..
+....E... ...xLr...lE..].j...E.
..G..T.6....en..Kh..."p.=...S1..[..2./......P4.R.....U.f...V....ZK.d..c.F.o.Q.jF$|@.....7otd....f9.C.....H.I.M]..@.}.;..v.:..
9...2..,!....T..^?.M...`^M%.club!....r.....Y.].a.......>
Ka.h9....c..{^..=..Am..\g     ._Y..Z...c... ..9...y.\V........WU..U._h        u.k.@+*>.b!@........
```

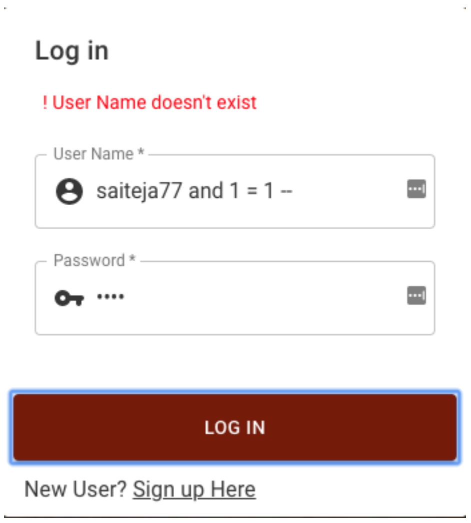
4 client pkts, 3 server pkts, 5 turns.

Entire conversation (2754 bytes)   Show and save data as ASCII   Stream 1

Figure 20. Packet capture of communication through secure channel

SQL Injection

To mitigate SQL injection, I have used Spring Data JPA. Most of the applications while making a query with the information received from the user, just replace a particular string pattern in the query with the string received. The Spring Data JPA library deals with this differently. Instead of replacing the string, it will insert the string into the placeholder. For example, let us say that the Authentication API is called for authentication a user. The query for that is 'SELECT * FROM USER WHERE USERNAME=?'. When the user tries to inject SQL with the string "saiteja77' AND '1' = '1' --", JPA queries the database for the username "saiteja77' AND '1' = '1' --" excluding all the SQL escape characters. Figure 21 confirms that.


Log in

! User Name doesn't exist

User Name *

saiteja77 and 1 = 1 --

Password *

••••

LOG IN

New User? Sign up Here

Figure 21. Successful mitigation of SQL injection

Cross Site Request Forgery

The threat of cross site request forgery is mitigated by following the REST architecture. According to REST architecture, all the sensitive information should be passed through a request payload. For example, to make a log in request, we must send the username and password details to the API. Instead of send these details in the request parameters, we should send them in the request body. Following this method ensures that the request is not forged.

Cross Site Scripting

The threat of cross site scripting is mitigated by the application's information security policy. According to the policy, only users who are authorized will be able to change the application data thus ensuring the data integrity. Even if the authorized user is compromised, the UI will not allow scripts from the data. If any such data is found, it will be converted into a text and will be displayed as shown in the figure 22.

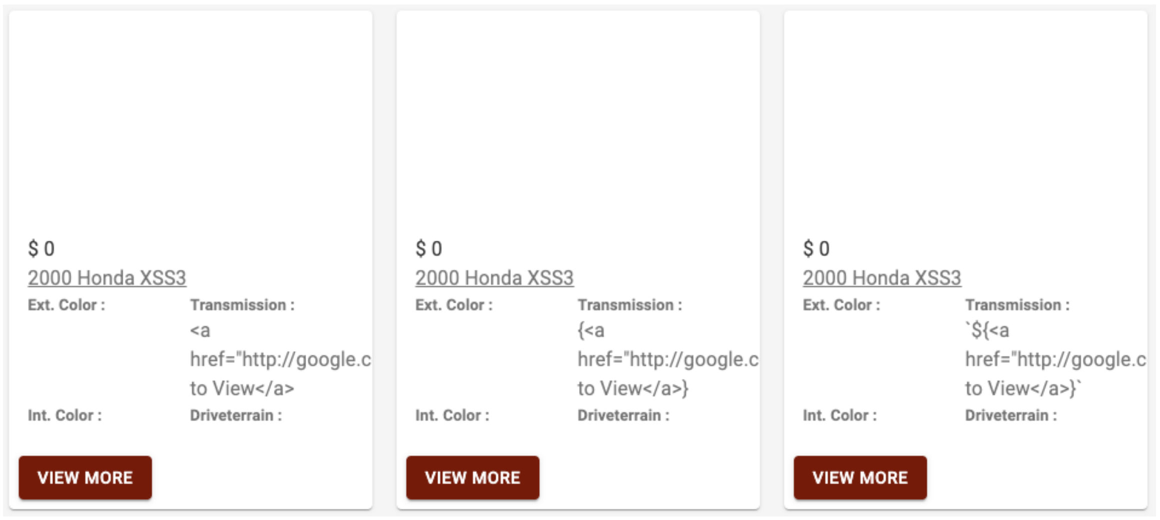
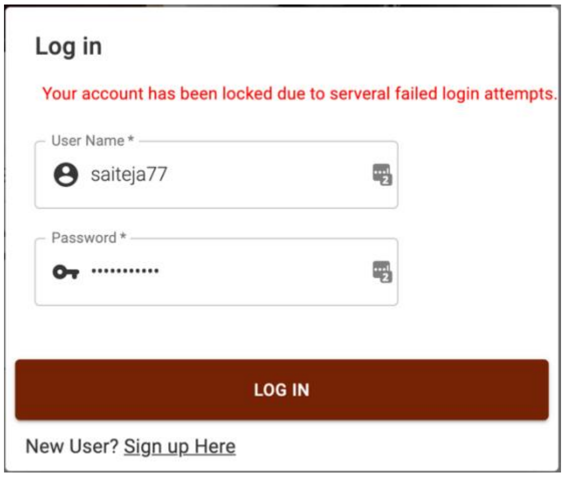


Figure 22. Successful mitigation of Cross Site Scripting

Brute Force Attack

Brute force attack is always made easy with weak authentication. For this purpose, the user is prompted to have a password with some predefined patterns like it should not start or end with a non-alphanumeric character, should be 8-32 characters length, at least one lowercase letter, one upper case letter, one number and, one of the symbols .!@#$%^&*. This would make it harder for the attackers to guess the password, yet the attacker can get through by using a script file or running a program. To mitigate this risk, the application is designed in such a way that if there are more than five invalid log in attempts, that account will be locked as shown in the figure 23.


Log in

Your account has been locked due to serveral failed login attempts.

User Name * saiteja77

Password * •••••••••••

Figure 22. Successful mitigation of Brute Force attack

Insecure Direct Object References

This threat is mitigated by securing all the resources which have sensitive information. The sensitivity of a resource can be defined by the information it holds. Any resource which has the personal information of a user, payment information, etc., can be as sensitive resource. These resources are secured in such a way that only an authenticated or authorized person can access it. For example, defined the roles of a user is a private information. This is a privileged operation which should be performed by an admin.

For this purpose, the APIs related to those operations would require an authorization header whose value is a JWT token, which includes the user information like fist name, last name, email address, username, roles, token issued time, token expiry time, etc. This information would be used to determine the roles of the user. If a user has a role of "ADMIN", then the request will be proceeded further or else the API responds back saying that the user is not authorized to request that particular resource.

Summary

In this chapter, we discussed about how the built ecommerce application mitigated most of the security risks. Even after mitigating those risks, there are still some improvements and better approaches to some threats. Those improvements and approaches will be continued in the future.

Chapter V: Conclusions and Future Work

Introduction

This chapter concludes all the discussions made and methods implemented along with discussing how this is a better solution over the existing ones and the works which are to be carried in the future.

Conclusion

As discussed earlier in the problem statement, there have been many studies about the security in ecommerce application, but all of them failed to define a methodology how to incorporate security into the application from the beginning of the application development. In this research, I have shown how I have secured all the resources in the APIs which are sensitive, how to store the passwords, how to mitigate the security threats right from the beginning of the application development, how to use cloud to increase the cost efficiency of the application, what are latest trends of application development, etc.

With all these discussions and provided evidences, I conclude that my study is better than the many existing studies related to ecommerce security.

Future Work

Even though the cross site request forgery is mitigated, there are more better approaches for it. One such approach is to maintain CSRF tokens across the application. The CSRF tokens are a unique, secret, and unpredictive values. These tokens will be unique for each module of the application and will be generated dynamically by the server which shall be included in each API request made by the client to the server.

We have seen how the brute force attack is mitigated by improving the password policy and minimizing the number of invalid tries. In this process, the user account is being locked which can be leading to denial of service. This can be better approached by giving an option to the user to unlock the account.

References

Authentication and access control attacks. (n.d.). Retrieved from Internet-Computer-Security.com : http://www.internet-computer-security.com/Security-Terminology/Authentication-Attacks.html

Buffer Overflow. (2016, 06 29). Retrieved from OWASP: https://www.owasp.org/index.php/Buffer_Overflow

Cross-site Scripting (XSS). (2018, 06 05). Retrieved from OWASP: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

Gambhir, J. (n.d.). Reduce Security Threats with Role-Based Access Control in Ecommerce. Retrieved from Hitachi Soultions: https://us.hitachi-solutions.com/blog/role-based-access-control-and-security-in-ecommerce/

Hardin, B. (2009, October 12). Insecure Communications. Retrieved from bretthard: http://bretthard.in/post/insecure-communications

Hau, D. (2012). Key fingerprint = AF19 FA27 2F94 998D FDB5 DE3D F8. Global Information Assurance Certifications, 1-10. Retrieved from https://www.giac.org/paper/gsec/3161/unauthorized-access-threats-risk-control/105264

History of Ecommerce. (n.d.). Retrieved from Ecommerce Land: https://www.ecommerce-land.com/history_ecommerce.html

Kaur, K. (2015). E-Commerce Privacy and Security System. International Journal of Engineering Research and Applications, 63-73.

Kh, R. (2014, 09 10). 4 E-commerce Security Failure Case Studies. Retrieved from Benzinga: https://www.benzinga.com/14/09/4840193/4-e-commerce-security-failure-case-studies

Khandare, N. (2015). Transaction Security for Internet E-commerce Application. International Journal of Advanced Research in Computer Science and Software Engineering, 629-640.

Loshin, P. (2016, June 21). Acer's e-commerce website hit by a customer data breach. Retrieved from TechTarget: https://searchsecurity.techtarget.com/news/450298892/Acers-ecommerce-website-hit-by-a-customer-data-breach

Maintaining Payment Security. (n.d.). Retrieved from PCI Security Standards: https://www.pcisecuritystandards.org/pci_security/maintaining_payment_security

Rouse, M. (2018, June). e-commerce (electronic commerce or EC). Retrieved from Techtarget: https://searchcio.techtarget.com/definition/e-commerce

Sawma, V. D., & Probert , R. L. (n.d.). E-COMMERCE AUTHENTICATION An Effective Countermeasures Design Model. Ottawa, Ontario, Canada.

Sharma, A. (2016). A Survey on E-Commerce Security Issues and Solutions. International Journal of Advanced Research in, 300-302.

Vogel, L. (2016, 06 07). Java concurrency (multi-threading) - Tutorial. Retrieved from Vogella: http://www.vogella.com/tutorials/JavaConcurrency/article.html

Weise, E. (2015, 05 16). Starbucks customers' mobile accounts breached by thieves. Retrieved from USA Today: https://www.usatoday.com/story/tech/2015/05/15/starbucks-gift-card-hack/27370491/

Appendix A: Backend Code of Cars API

BodyStylesController.java

```
package com.bitbyte.cargaraage.controller;
import com.bitbyte.cargaraage.enitities.BodyStylesEntity;
import com.bitbyte.cargaraage.service.BodyStyleService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import javax.validation.constraints.NotNull;
import java.util.Optional;

@RestController
@CrossOrigin
public class BodyStylesController {

   @Autowired
   private BodyStyleService service;

   @GetMapping("/bodyStyles")
   public ResponseEntity<Iterable<BodyStylesEntity>> getBodyStyles() {
      return new ResponseEntity<>(service.getAll(), HttpStatus.OK);
   }

   @GetMapping("/bodyStyles/{id}")
   public ResponseEntity<BodyStylesEntity> getBodyStyles(@NotNull @PathVariable String
id) {
      BodyStylesEntity make = service.getById(id);
      if (make != null)
         return new ResponseEntity<>(make, HttpStatus.OK);
      else
         return new ResponseEntity<>(new BodyStylesEntity(null), HttpStatus.NOT_FOUND);
   }

   @PostMapping("/bodyStyles")
   public ResponseEntity<BodyStylesEntity> saveBodyStyles(@NotNull @RequestBody
BodyStylesEntity make) {
      BodyStylesEntity makesEntity = service.save(make);
      return new ResponseEntity<>(makesEntity, HttpStatus.CREATED);
   }

   @PutMapping("/bodyStyles")
   public ResponseEntity<BodyStylesEntity> updateBodyStyles(@NotNull @RequestBody
BodyStylesEntity make, @NotNull @PathVariable String id) {
      BodyStylesEntity makesEntity = service.update(make, id);
      if (makesEntity != null)
         return new ResponseEntity<>(makesEntity, HttpStatus.OK);
      else
         return new ResponseEntity<>(make, HttpStatus.BAD_REQUEST);
   }

   @DeleteMapping("/bodyStyles")
   public ResponseEntity<BodyStylesEntity> deleteBodyStyles(@NotNull @PathVariable
String id) {
      Optional<BodyStylesEntity> make = service.delete(id);
      if (make.isPresent())
         return new ResponseEntity<>(make.get(), HttpStatus.OK);
      else
         return new ResponseEntity<>(new BodyStylesEntity(null),
HttpStatus.BAD_REQUEST);
   }

   @GetMapping("/bodyStyles/pageable")
   public ResponseEntity<Page<BodyStylesEntity>> getPageableBodyStyles(@NotNull
@RequestParam Integer pageNumber, @NotNull @RequestParam Integer pageSize) {
      return new ResponseEntity<>(service.getPageableBodyStyles(pageNumber, pageSize),
HttpStatus.OK);
   }
}
```

CarsController.java

```
package com.bitbyte.cargaraage.controller;
import com.bitbyte.cargaraage.enitities.CarsEntity;
import com.bitbyte.cargaraage.service.CarsService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.server.ResponseStatusException;

@RestController
@CrossOrigin
public class CarsController {

   @Autowired
   private CarsService service;
```

```
    @GetMapping("/cars")
    public ResponseEntity<Iterable<CarsEntity>> getCars(){
        return new ResponseEntity<>(service.getCars(), HttpStatus.OK);
    }

    @GetMapping("/cars/{id}")
    public ResponseEntity<CarsEntity> getCarsById(@PathVariable String id){
        CarsEntity car = service.getCarsById(id);
        if( car != null)
            return new ResponseEntity<>(car, HttpStatus.OK);
        else
            throw new ResponseStatusException(HttpStatus.NOT_FOUND);
    }

    @PostMapping("/cars")
    public ResponseEntity<CarsEntity> saveCar(@RequestBody CarsEntity car){
        return new ResponseEntity<>(service.saveCar(car), HttpStatus.CREATED);
    }

    @PutMapping("/cars")
    public ResponseEntity<CarsEntity> updateCar(@RequestBody CarsEntity car) {
        return  new ResponseEntity<>(service.updateCar(car), HttpStatus.OK);
    }
}
```

MakesController.java

```
package com.bitbyte.cargaraage.controller;
import com.bitbyte.cargaraage.enitities.MakesEntity;
import com.bitbyte.cargaraage.service.MakesService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import javax.validation.constraints.NotNull;
import java.util.Optional;

@RestController
@CrossOrigin
public class MakesController {

    @Autowired
    private MakesService service;

    @GetMapping("/makes")
    public ResponseEntity<Iterable<MakesEntity>> getMakes(){
        return new ResponseEntity<>(service.getAll(), HttpStatus.OK);
    }

    @GetMapping("/makes/{id}")
    public ResponseEntity<MakesEntity> getMakesEntity(@NotNull @PathVariable String id){
        MakesEntity make = service.getById(id);
        if(make!=null)
            return new ResponseEntity<>(make, HttpStatus.OK);
        else
            return new ResponseEntity<>(new MakesEntity(null), HttpStatus.NOT_FOUND);
    }

    @PostMapping("/makes")
    public ResponseEntity<MakesEntity> saveMakes(@NotNull @RequestBody MakesEntity
make){
        MakesEntity makesEntity = service.save(make);
        return new ResponseEntity<>(makesEntity, HttpStatus.CREATED);
    }

    @PutMapping("/makes")
    public ResponseEntity<MakesEntity> updateMakes(@NotNull @RequestBody MakesEntity
make, @NotNull @PathVariable String id){
        MakesEntity makesEntity = service.update(make, id);
        if (makesEntity != null)
            return new ResponseEntity<>(makesEntity, HttpStatus.OK);
        else
            return new ResponseEntity<>(make, HttpStatus.BAD_REQUEST);
    }

    @DeleteMapping("/makes")
    public ResponseEntity<MakesEntity> deleteMakes(@NotNull @PathVariable String id){
        Optional<MakesEntity> make = service.delete(id);
        if(make.isPresent())
            return new ResponseEntity<>(make.get(), HttpStatus.OK);
        else
            return new ResponseEntity<>(new MakesEntity(null), HttpStatus.BAD_REQUEST);
    }
}
```

UsersController.java

```
package com.bitbyte.cargaraage.controller;
import com.bitbyte.cargaraage.enitities.UserEntity;
import com.bitbyte.cargaraage.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.security.NoSuchAlgorithmException;
import java.util.List;

@RestController
@CrossOrigin
public class UsersController {

    @Autowired
    private UserService service;

    @GetMapping("/users")
    public ResponseEntity<Iterable<UserEntity>> getAll(){
        return new ResponseEntity<>(service.getAll(), HttpStatus.OK);
    }

    @GetMapping("/users/{id}")
    public ResponseEntity<UserEntity> getUserById(@PathVariable String id){
        return new ResponseEntity<>(service.getUserById(id), HttpStatus.OK);
    }

    @GetMapping("users/validate_credentials")
    public ResponseEntity<Boolean> checkPassword(@RequestParam String emailAddress,
@RequestBody String password) throws NoSuchAlgorithmException {
        return new ResponseEntity<>(service.checkPassword(password, emailAddress),
HttpStatus.OK);
    }

    @PostMapping("/users")
    public ResponseEntity<UserEntity> saveUser(@RequestBody UserEntity user){
        return new ResponseEntity<>(service.save(user), HttpStatus.CREATED);
    }

    @DeleteMapping("/users/{id}")
    public ResponseEntity<UserEntity> deleteUser(@PathVariable String id){
        return new ResponseEntity<>(service.delete(id), HttpStatus.OK);
    }
}
```

BodyStylesEntity.java

```
package com.bitbyte.cargaraage.enitities;
import com.fasterxml.jackson.annotation.JsonProperty;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "body_styles")
public class BodyStylesEntity {

   @Id
   private String id;

   @JsonProperty(value = "value")
   private String styleName;

   public BodyStylesEntity() {
   }

   public String getId() {
      return id;
   }

   public void setId(String id) {
      this.id = id;
   }

   public String getStyleName() {
      return styleName;
   }

   public void setStyleName(String styleName) {
      this.styleName = styleName;
   }

   public BodyStylesEntity(String styleName) {
      this.styleName = styleName;
   }
}
```

CarsEntity.java

```
package com.bitbyte.cargaraage.enitities;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import java.util.List;

@Document(collection = "cars")
public class CarsEntity {

   @Id
```

```
private String id;
private String carName;
private Integer carPrice;
private Properties specs;
private List<String> pictures;

public CarsEntity() {
}

public CarsEntity(String carName, Integer carPrice, Properties specs, List<String> pictures) {
   this.carName = carName;
   this.carPrice = carPrice;
   this.specs = specs;
   this.pictures = pictures;
}

public String getId() {
   return id;
}

public void setId(String id) {
   this.id = id;
}

public String getCarName() {
   return carName;
}

public void setCarName(String carName) {
   this.carName = carName;
}

public Properties getSpecs() {
   return specs;
}

public void setSpecs(Properties specs) {
   this.specs = specs;
}

public List<String> getPictures() {
   return pictures;
}

public void setPictures(List<String> pictures) {
        this.pictures = pictures;
    }

    public Integer getCarPrice() {
        return carPrice;
    }

    public void setCarPrice(Integer carPrice) {
        this.carPrice = carPrice;
    }
}
```

FuelConsumption.java

```
package com.bitbyte.cargaraage.enitities;
public class FuelConsumption {
    private Integer city;
    private Integer highway;
    private Integer combined;

    public FuelConsumption() {
    }

    public Integer getCity() {
        return city;
    }

    public void setCity(Integer city) {
        this.city = city;
    }

    public Integer getHighway() {
        return highway;
    }

    public void setHighway(Integer highway) {
        this.highway = highway;
    }

    public Integer getCombined() {
        return combined;
    }

    public void setCombined(Integer combined) {
        this.combined = combined;
    }
}
```

MakesEntity.java

```
package com.bitbyte.cargaraage.enitities;
import com.fasterxml.jackson.annotation.JsonProperty;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.index.IndexDirection;
import org.springframework.data.mongodb.core.index.Indexed;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "makes")
public class MakesEntity {

   @Id
   private String id;

   @Indexed(direction = IndexDirection.ASCENDING)
   @JsonProperty(value = "value")
   private String makeName;

   public MakesEntity(String makeName) {
      this.makeName = makeName;
   }

   public String getId() {
      return id;
   }

   public String getMakeName() {
      return makeName;
   }

   public void setMakeName(String makeName) {
      this.makeName = makeName;
   }

   public void setId(String id) {
      this.id = id;
   }

   public MakesEntity() {
   }
}
```

Properties.java

```
package com.bitbyte.cargaraage.enitities;
public class Properties {

    private BodyStylesEntity bodyStyle;
    private MakesEntity make;
    private String fuelType;
    private FuelConsumption mileage;
    private Integer fuelCapacity;
    private Integer accelerationTime;
    private Integer seatingCapacity;
    private Integer horsePower;
    private Integer torque;
    private String engineType;
    private Integer year;
    private String interiorColor;
    private String exteriorColor;
    private String transmission;
    private String driveterrain;

    public Properties(BodyStylesEntity bodyStyle, MakesEntity make, String fuelType,
FuelConsumption mileage, Integer fuelCapacity, Integer accelerationTime, Integer
seatingCapacity, Integer horsePower, Integer torque, String engineType, Integer year, String
interiorColor, String exteriorColor, String transmission, String driveterrain) {
        this.bodyStyle = bodyStyle;
        this.make = make;
        this.fuelType = fuelType;
        this.mileage = mileage;
        this.fuelCapacity = fuelCapacity;
        this.accelerationTime = accelerationTime;
        this.seatingCapacity = seatingCapacity;
        this.horsePower = horsePower;
        this.torque = torque;
        this.engineType = engineType;
        this.year = year;
        this.interiorColor = interiorColor;
        this.exteriorColor = exteriorColor;
        this.transmission = transmission;
        this.driveterrain = driveterrain;
    }

    public BodyStylesEntity getBodyStyle() {
        return bodyStyle;
    }

    public void setBodyStyle(BodyStylesEntity bodyStyle) {
```

```
    this.bodyStyle = bodyStyle;
}

public MakesEntity getMake() {
    return make;
}

public void setMake(MakesEntity make) {
    this.make = make;
}

public String getFuelType() {
    return fuelType;
}

public void setFuelType(String fuelType) {
    this.fuelType = fuelType;
}

public FuelConsumption getMileage() {
    return mileage;
}

public void setMileage(FuelConsumption mileage) {
    this.mileage = mileage;
}

public Integer getFuelCapacity() {
    return fuelCapacity;
}

public void setFuelCapacity(Integer fuelCapacity) {
    this.fuelCapacity = fuelCapacity;
}

public Integer getAccelerationTime() {
    return accelerationTime;
}

public void setAccelerationTime(Integer accelerationTime) {
    this.accelerationTime = accelerationTime;
}

public Integer getSeatingCapacity() {
    return seatingCapacity;
```

```
}

public void setSeatingCapacity(Integer seatingCapacity) {
   this.seatingCapacity = seatingCapacity;
}

public Integer getHorsePower() {
   return horsePower;
}

public void setHorsePower(Integer horsePower) {
   this.horsePower = horsePower;
}

public Integer getTorque() {
   return torque;
}

public void setTorque(Integer torque) {
   this.torque = torque;
}

public String getEngineType() {
   return engineType;
}

public void setEngineType(String engineType) {
   this.engineType = engineType;
}

public Integer getYear() {
   return year;
}

public void setYear(Integer year) {
   this.year = year;
}

public String getInteriorColor() {
   return interiorColor;
}

public void setInteriorColor(String interiorColor) {
   this.interiorColor = interiorColor;
}
```

```
    public String getExteriorColor() {
        return exteriorColor;
    }

    public void setExteriorColor(String exteriorColor) {
        this.exteriorColor = exteriorColor;
    }

    public String getTransmission() {
        return transmission;
    }

    public void setTransmission(String transmission) {
        this.transmission = transmission;
    }

    public String getDriveterrain() {
        return driveterrain;
    }

    public void setDriveterrain(String driveterrain) {
        this.driveterrain = driveterrain;
    }
}
```

Status.java

```
package com.bitbyte.cargaraage.pojo;

public enum Status {
    ACTIVE,INACTIVE,DELETED;

    @Override
    public String toString() {
        return super.toString();
    }
}
```

BodyStylesRepository.java

```
package com.bitbyte.cargaraage.repository;
import com.bitbyte.cargaraage.enitities.BodyStylesEntity;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;
import java.util.Optional;

@Repository
public interface BodyStylesRepository extends CrudRepository<BodyStylesEntity, String> {

   Page<BodyStylesEntity> findAll(Pageable page);

   Optional<BodyStylesEntity> findByStyleName(String styleName);
}
```

CarsRepository.java

```
package com.bitbyte.cargaraage.repository;
import com.bitbyte.cargaraage.enitities.CarsEntity;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface CarsRepository extends CrudRepository<CarsEntity, String> {
}
```

MakesRepository.java

```
package com.bitbyte.cargaraage.repository;
import com.bitbyte.cargaraage.enitities.MakesEntity;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository
public interface MakesRepository extends CrudRepository<MakesEntity, String> {
   Optional<MakesEntity> findByMakeName(String makeName);
}
```

BodyStyleService.java

```
package com.bitbyte.cargaraage.service;
import com.bitbyte.cargaraage.enitities.BodyStylesEntity;
import com.bitbyte.cargaraage.repository.BodyStylesRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.stereotype.Service;

import javax.validation.constraints.NotNull;
import java.util.Optional;
import static org.springframework.data.domain.PageRequest.*;

@Service
public class BodyStyleService {

   @Autowired
   private BodyStylesRepository repository;

   public Iterable<BodyStylesEntity> getAll() {
      return repository.findAll();
   }

   public BodyStylesEntity getById(@NotNull String id){
      Optional<BodyStylesEntity> make = repository.findById(id);
      if(make.isPresent()) {
         return make.get();
      }
      else{
         return null;
      }
   }

   public BodyStylesEntity save(@NotNull BodyStylesEntity make) {
      BodyStylesEntity newEntity = repository.save(new
BodyStylesEntity(make.getStyleName()));
      return newEntity;
   }

   public BodyStylesEntity update(@NotNull BodyStylesEntity bodyStyle, @NotNull String id)
{
      Optional<BodyStylesEntity> entity = repository.findById(id);
      if(entity.isPresent()){
         entity.get().setStyleName(bodyStyle.getStyleName());
         return repository.save(entity.get());
      } else
         return null;
   }

   public Optional<BodyStylesEntity> delete(String id) {
      Optional<BodyStylesEntity> entity = repository.findById(id);
      if(entity.isPresent())
         repository.delete(entity.get());
      return entity;
   }
```

```
    public Page<BodyStylesEntity> getPageableBodyStyles(int pageNumber, int pageSize){
        return repository.findAll(of(pageNumber,pageSize));
    }
}
```

CarsService.java

```
package com.bitbyte.cargaraage.service;
import com.bitbyte.cargaraage.enitities.BodyStylesEntity;
import com.bitbyte.cargaraage.enitities.CarsEntity;
import com.bitbyte.cargaraage.enitities.MakesEntity;
import com.bitbyte.cargaraage.repository.BodyStylesRepository;
import com.bitbyte.cargaraage.repository.CarsRepository;
import com.bitbyte.cargaraage.repository.MakesRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.Optional;

@Service
public class CarsService {

    @Autowired
    private CarsRepository repository;

    @Autowired
    private BodyStylesRepository bodyStylesRepository;

    @Autowired
    private MakesRepository makesRepository;

    public Iterable<CarsEntity> getCars() {
        return repository.findAll();
    }

    public CarsEntity getCarsById(String id) {
        Optional<CarsEntity> car = repository.findById(id);
        return car.isPresent() ? car.get() : null;
    }

    public CarsEntity saveCar(CarsEntity car) {
        Optional<BodyStylesEntity> bodyStyle =
bodyStylesRepository.findByStyleName(car.getSpecs().getBodyStyle().getStyleName());
        Optional<MakesEntity> make =
makesRepository.findByMakeName(car.getSpecs().getMake().getMakeName());
        if(bodyStyle.isPresent() && make.isPresent()){
            car.getSpecs().getBodyStyle().setId(bodyStyle.get().getId());
            car.getSpecs().getMake().setId(make.get().getId());
        } else if(bodyStyle.isPresent() || make.isPresent()){
            if(bodyStyle.isPresent())
                car.getSpecs().getBodyStyle().setId(bodyStyle.get().getId());
            else
                car.getSpecs().getMake().setId(make.get().getId());
        }
        return repository.save(car);
    }

    public CarsEntity updateCar(CarsEntity car) {
        return repository.save(car);
    }
}
```

MakesService.java

```
package com.bitbyte.cargaraage.service;
import com.bitbyte.cargaraage.enitities.MakesEntity;
import com.bitbyte.cargaraage.repository.MakesRepository;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import javax.validation.constraints.NotNull;
import java.util.Optional;

@Service
public class MakesService {

    private static final Logger LOGGER = LoggerFactory.getLogger(MakesService.class);

    @Autowired
    private MakesRepository repository;

    public Iterable<MakesEntity> getAll() {
        return repository.findAll();
    }

    public MakesEntity getById(@NotNull String id){
        Optional<MakesEntity> make = repository.findById(id);
        if(make.isPresent()) {
            LOGGER.info("Found One Make");
            return make.get();
        }
        else{
            LOGGER.error("No makes were found for the given ID: {}", id);
            return null;
        }
    }

    public MakesEntity save(@NotNull MakesEntity make) {
        MakesEntity newEntity = repository.save(new MakesEntity(make.getMakeName()));
        return newEntity;
    }

    public MakesEntity update(@NotNull MakesEntity make, @NotNull String id) {
        Optional<MakesEntity> entity = repository.findById(id);
        if(entity.isPresent()){
            entity.get().setMakeName(make.getMakeName());
            return repository.save(entity.get());
        } else
            return null;
    }

    public Optional<MakesEntity> delete(String id) {
        Optional<MakesEntity> entity = repository.findById(id);
        if(entity.isPresent())
            repository.delete(entity.get());
        return entity;
    }
}
```

Applicaton.properties

```
spring.data.mongodb.host=cluster1-shard-00-00-xhqwl.gcp.mongodb.net
spring.data.mongodb.port=27017
spring.data.mongodb.database=car_garaage
spring.data.mongodb.username=app_user
spring.data.mongodb.password=ENC(9BjwpfuG2CyJk2POOz36JKcZFP5t1TwVTsP7gBl0XI0=)
spring.data.mongodb.uri = mongodb+srv://app_user:rbtQ3a4T22eZkUvR@cluster1-
xhqwl.gcp.mongodb.net/car_garaage?retryWrites=true
password.hashing-algorithm = MD5
spring.application.name=cars-api
server.servlet.context-path=/cars-api
server.port=8082
```

build.gradle

```
plugins {
    id 'org.springframework.boot' version '2.1.8.RELEASE'
    id 'io.spring.dependency-management' version '1.0.8.RELEASE'
    id 'java'
}

group = 'com.bitbyte'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '1.8'

configurations {
    developmentOnly
    runtimeClasspath {
        extendsFrom developmentOnly
    }
}

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-actuator'
    implementation 'org.springframework.boot:spring-boot-starter-data-mongodb'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    compile group: 'com.github.ulisesbocchio', name: 'jasypt-spring-boot-starter', version: '2.1.2'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

Appendix B: Backend Code of Authentication API

AuthController.java

```
package com.bitbyte.cargaraag.auth.controllers;

import com.bitbyte.cargaraag.auth.entities.User;

import com.bitbyte.cargaraag.auth.exceptionhandlers.WrongCredentialsException;

import com.bitbyte.cargaraag.auth.models.AuthorizationResponse;

import com.bitbyte.cargaraag.auth.models.Credentials;

import com.bitbyte.cargaraag.auth.services.AuthenticationService;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;


import org.springframework.web.bind.annotation.*;


import java.security.NoSuchAlgorithmException;


@RestController

@CrossOrigin

public class AuthController {


    private static final Logger LOG = LoggerFactory.getLogger(AuthController.class);
    @Autowired
    private AuthenticationService authenticationService;


    @PostMapping(value = "/login/authorize", produces = "application/json")
    public ResponseEntity<AuthorizationResponse> authorizeUser(@RequestBody Credentials
userCreds){
        try{
            String token = authenticationService.authenticateUser(userCreds);
            LOG.info("Successfully Authenticated the the user {}", userCreds.getUserName());
            return new ResponseEntity<>(new AuthorizationResponse("Log in Successful!,", token),
HttpStatus.OK);
        } catch (WrongCredentialsException wce){
            LOG.info("{}: {}", wce.getMessage(), userCreds.getUserName());
            return new ResponseEntity<>(new AuthorizationResponse(wce.getMessage(), null),
HttpStatus.UNAUTHORIZED);
        } catch (NoSuchAlgorithmException nsae){
            LOG.info("{}: {}", nsae.getMessage(), userCreds.getUserName());
            return new ResponseEntity<>(new AuthorizationResponse("It's not you. It's. We're trying
to fix some things up. Please try later", null), HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
    @PostMapping("/sign_up")
    public ResponseEntity<AuthorizationResponse> signUpUser(@RequestBody User user){
        try{
            String token = authenticationService.signUpNewUser(user);
            LOG.info("Successfully created the user: {}", user.getUserName());
            return new ResponseEntity<>(new AuthorizationResponse("Sign up successful. Please
use your user name: " + user.getUserName() + " to Log In for the next time", token),
HttpStatus.CREATED);
        } catch (NoSuchAlgorithmException nsae){
            LOG.info("{}: {}", nsae.getMessage(), user.getUserName());
            return new ResponseEntity<>(new AuthorizationResponse("It's not you. It's. We're trying
to fix some things up. Please try later", null), HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
}
```

Roles.java

```
package com.bitbyte.cargaraag.auth.entities;
import javax.persistence.*;
import java.util.HashSet;
import java.util.Set;

@Entity
```

```
public class Roles {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id")
    int id;

    @Column(unique = true)
    private String role;

    @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    @JoinTable(inverseJoinColumns = @JoinColumn(name = "userId", referencedColumnName
= "id"),
            joinColumns = @JoinColumn(name = "roleId", referencedColumnName = "id"))
    private Set<User> users;

    public Roles(String role) {
        this.role = role;
    }

    public Roles() {
    }

    public int getId() {
```

```
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getRole() {
    return role;
}

public void setRole(String role) {
    this.role = role;
}

public Set<User> getUsers() {
    return users;
}

public void setUsers(Set<User> users) {
    this.users = users;
}
```

```
    public void addUser(User user) {

        if (users == null) {

            users = new HashSet<>();

        }

        users.add(user);

    }

}
```

User.java

```
package com.bitbyte.cargaraag.auth.entities;
import com.bitbyte.cargaraag.auth.models.Gender;
import com.bitbyte.cargaraag.auth.models.Status;
import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import javax.persistence.*;
import java.util.Date;
import java.util.Set;

@Entity
@JsonIgnoreProperties(value = "password", allowSetters = true)
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String firstName;
    private String lastName;
    @Column(unique = true)
    private String userName;
    private String password;
    private Gender gender;
    private String email;
    private Date dob;
    @JsonIgnore
    private byte[] salt;
    @JsonIgnore
```

```
private String hashingAlgorithm;
private Status status;

@ManyToMany(mappedBy = "users",fetch = FetchType.EAGER)
private Set<Roles> roles;

public Long getId() {
   return id;
}

public void setId(Long id) {
   this.id = id;
}

public String getFirstName() {
   return firstName;
}

public void setFirstName(String firstName) {
   this.firstName = firstName;
}

public String getLastName() {
   return lastName;
}

public void setLastName(String lastName) {
   this.lastName = lastName;
}

public String getUserName() {
   return userName;
}

public void setUserName(String userName) {
   this.userName = userName;
}

public String getPassword() {
   return password;
}

public void setPassword(String password) {
   this.password = password;
}
```

```
public Gender getGender() {
   return gender;
}

public void setGender(Gender gender) {
   this.gender = gender;
}

public String getEmail() {
   return email;
}

public void setEmail(String email) {
   this.email = email;
}

public Date getDob() {
   return dob;
}

public void setDob(Date dob) {
   this.dob = dob;
}

public byte[] getSalt() {
   return salt;
}

public void setSalt(byte[] salt) {
   this.salt = salt;
}

public String getHashingAlgorithm() {
   return hashingAlgorithm;
}

public void setHashingAlgorithm(String hashingAlgorithm) {
   this.hashingAlgorithm = hashingAlgorithm;
}

public Status getStatus() {
   return status;
}
```

```
    public void setStatus(Status status) {
        this.status = status;
    }

    public Set<Roles> getRoles() {
        return roles;
    }

    public void setRoles(Set<Roles> roles) {
        this.roles = roles;
    }
}
```

WrongCredentialsException.java

```
package com.bitbyte.cargaraag.auth.exceptionhandlers;

public class WrongCredentialsException extends RuntimeException {

    private String message;

    public WrongCredentialsException(String exceptionMessage){
        this.message = exceptionMessage;
    }

    @Override
    public String getMessage() {
        return message;
    }
}
```

AuthorizationResponse.java

```
package com.bitbyte.cargaraag.auth.models;

public class AuthorizationResponse {
    private String message;
    private String token;

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public String getToken() {
        return token;
    }

    public void setToken(String token) {
        this.token = token;
    }

    public AuthorizationResponse(String message, String token) {
        this.message = message;
        this.token = token;
    }
}
```

Credentials.java

```
package com.bitbyte.cargaraag.auth.models;

public class Credentials {
    String userName;
    String password;

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

Gender.java

```
package com.bitbyte.cargaraag.auth.models;

public enum Gender {
    Male, Female, Other
}
```

Role.java

```
package com.bitbyte.cargaraag.auth.models;

public enum Role {
   USER, ADMIN
}
```

Status.java

```
package com.bitbyte.cargaraag.auth.models;

public enum Status {
   ACTIVE, INACTIVE, DELETED
}
```

RolesRepository.java

```
package com.bitbyte.cargaraag.auth.repository;

import com.bitbyte.cargaraag.auth.entities.Roles;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository
public interface RolesRepository extends CrudRepository<Roles, Long> {
   Optional<Roles> findByRole(String role);
}
```

UserRepository.java

```
package com.bitbyte.cargaraag.auth.repository;
import com.bitbyte.cargaraag.auth.entities.User;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;
import java.util.Optional;

@Repository
public interface UserRepository extends CrudRepository<User, Long> {

   Optional<User> findByUserName(String userName);
}
```

PasswordUtils.java

```
package com.bitbyte.cargaraag.auth.securityutils;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
```

```
public class PasswordUtils {

    private static final char[] HEX_ARRAY = "0123456789ABCDEF".toCharArray();

    public static byte[] createSalt() {
        byte[] salt = new byte[20];
        SecureRandom random = new SecureRandom();
        random.nextBytes(salt);
        return salt;
    }

    public static byte[] generateSaltedHash(String password, String hashingAlgorithm, byte[] salt)
throws NoSuchAlgorithmException {
        MessageDigest digest = MessageDigest.getInstance(hashingAlgorithm);
        digest.reset();
        digest.update(salt);
        byte[] hash = digest.digest(password.getBytes());
        return hash;
    }

    public static String convertByteArrayToHexString(byte[] passwordHash) {
        char[] hexChars = new char[passwordHash.length * 2];
        for (int j = 0; j < passwordHash.length; j++) {
            int v = passwordHash[j] & 0xFF;
            hexChars[j * 2] = HEX_ARRAY[v >>> 4];
            hexChars[j * 2 + 1] = HEX_ARRAY[v & 0x0F];
        }
        return new String(hexChars);
    }

    public static boolean comparePasswords(String password, String hash, byte[] salt, String
hashingAlgorithm) throws NoSuchAlgorithmException {
        String generatedHash = convertByteArrayToHexString(generateSaltedHash(password,
hashingAlgorithm, salt));
        return hash.equals(generatedHash);
    }
}
```

TokenGenerator.java

```
package com.bitbyte.cargaraag.auth.securityutils;
import com.bitbyte.cargaraag.auth.entities.Roles;
import com.bitbyte.cargaraag.auth.entities.User;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.apache.commons.lang3.time.DateUtils;
import java.util.Date;
import java.util.stream.Collectors;

public class TokenGenerator {

   public static String generateToken(User authenticatedUser) {
      Claims claims = Jwts.claims()
            .setSubject(authenticatedUser.getUserName())
            .setExpiration(DateUtils.addHours(new Date(), 24))
            .setIssuedAt(new Date())
            .setIssuer("");
      claims.put("clientId", "");
      claims.put("userId", authenticatedUser.getId());
      claims.put("userName", authenticatedUser.getUserName());
      claims.put("email", authenticatedUser.getEmail());
      claims.put("firstName", authenticatedUser.getFirstName());
      claims.put("lastName", authenticatedUser.getLastName());
      claims.put("fullName", authenticatedUser.getFirstName() + " " +
authenticatedUser.getLastName());
      claims.put("sts", authenticatedUser.getStatus());
      claims.put("group",
authenticatedUser.getRoles().stream().map(Roles::getRole).collect(Collectors.toList()));

      return Jwts.builder()
            .setClaims(claims)
            .signWith(SignatureAlgorithm.HS512, "clientSecret")
            .compact();
   }
}
```

AuthenticationService.java

```
package com.bitbyte.cargaraag.auth.services;
import com.bitbyte.cargaraag.auth.entities.Roles;
import com.bitbyte.cargaraag.auth.entities.User;
import com.bitbyte.cargaraag.auth.exceptionhandlers.WrongCredentialsException;
import com.bitbyte.cargaraag.auth.models.Credentials;
import com.bitbyte.cargaraag.auth.models.Role;
import com.bitbyte.cargaraag.auth.repository.RolesRepository;
import com.bitbyte.cargaraag.auth.repository.UserRepository;
import com.bitbyte.cargaraag.auth.securityutils.PasswordUtils;
import com.bitbyte.cargaraag.auth.securityutils.TokenGenerator;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import java.security.NoSuchAlgorithmException;
import java.util.HashSet;
import java.util.Optional;
import java.util.Set;

@Service
public class AuthenticationService {

   @Autowired
   private UserRepository userRepository;

   @Autowired
   private RolesRepository rolesRepository;

   @Value("${hashing.algorithm}")
   private String hashingAlgorithm;

   public String authenticateUser(Credentials userCreds) throws NoSuchAlgorithmException {
      User authenticatedUser = validateUserCredentials(userCreds);
      String token = TokenGenerator.generateToken(authenticatedUser);
      return token;
   }

   public String signUpNewUser(User user) throws NoSuchAlgorithmException {
      byte[] salt = PasswordUtils.createSalt();
      byte[] saltedHashPassword = PasswordUtils.generateSaltedHash(user.getPassword(),
hashingAlgorithm, salt);
      user.setPassword(PasswordUtils.convertByteArrayToHexString(saltedHashPassword));
      user.setSalt(salt);
      user.setHashingAlgorithm(hashingAlgorithm);

      Roles role = rolesRepository
            .findByRole(Role.USER.name())
            .orElseGet(() -> rolesRepository.save(new Roles(Role.USER.name())));

      role.addUser(userRepository.save(user));
      rolesRepository.save(role);
      User savedUser = userRepository.findByUserName(user.getUserName()).get();
      return TokenGenerator.generateToken(savedUser);
   }

   private User validateUserCredentials(Credentials userCreds) throws
NoSuchAlgorithmException {
        Optional<User> optionalUser =
userRepository.findByUserName(userCreds.getUserName());
        if (optionalUser.isPresent()) {
            User user = optionalUser.get();
            boolean isValidUser = PasswordUtils.comparePasswords(userCreds.getPassword(),
user.getPassword(), user.getSalt(), user.getHashingAlgorithm());
            if (isValidUser)
                return user;
            else
                throw new WrongCredentialsException("Invalid credentials");
        } else
            throw new WrongCredentialsException("User Name doesn't exist");
    }
}
```

Application.properties

```
# Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
spring.datasource.url=jdbc:postgresql://35.184.163.17:5432/cargaraage
spring.datasource.username=
spring.datasource.password=

# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
spring.jpa.properties.hibernate.show_sql=false
spring.jpa.properties.hibernate.jdbc.batch_size=1000
spring.servlet.multipart.max-file-size=6000KB
spring.servlet.multipart.max-request-size=6000KB
spring.jpa.properties.hibernate.default_schema=public
```

build.gradle

```
plugins {
    id 'org.springframework.boot' version '2.2.0.RELEASE'
    id 'io.spring.dependency-management' version '1.0.8.RELEASE'
    id 'java'
}

group = 'com.bitbyte.cargaraag'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '1.8'

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    runtimeOnly 'org.postgresql:postgresql'
    testImplementation('org.springframework.boot:spring-boot-starter-test') {
        exclude group: 'org.junit.vintage', module: 'junit-vintage-engine'
    }
}

test {
    useJUnitPlatform()
}
```

Appendix C: Backend Code of Orders API

AdminController.java

```
package com.bitbyte.cargaraage.api.controllers;

import com.bitbyte.cargaraage.api.entities.Role;

import com.bitbyte.cargaraage.api.services.AdminService;

import com.bitbyte.cargaraage.api.services.AuthorizationService;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.*;

import org.springframework.web.server.ResponseStatusException;


@RestController("/admin")

@CrossOrigin

public class AdminController {


    private static final Logger LOG = LoggerFactory.getLogger(AdminController.class);


    @Autowired

    private AdminService adminService;
    @Autowired

    private AuthorizationService authorizationService;



    @PostMapping("/roles")

    public ResponseEntity<Role> createRole(@RequestHeader String token, @RequestBody

Role role){

        if(authorizationService.isAuthorized(token, "ADMIN")){

            LOG.info("AdminController -> createRole -> start");

            Role createdRole = adminService.createRole(role);

            LOG.info("AdminController -> createRole -> end");

            return new ResponseEntity<>(createdRole, HttpStatus.CREATED);

        } else {

            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);

        }

    }

}
```

OrdersController.java

```
package com.bitbyte.cargaraage.api.controllers;
import com.bitbyte.cargaraage.api.entities.Order;
import com.bitbyte.cargaraage.api.services.AuthorizationService;
import com.bitbyte.cargaraage.api.services.OrderService;
import com.bitbyte.cargaraage.api.services.PaymentService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.server.ResponseStatusException;

@RestController
@CrossOrigin
public class OrdersController {

   private static final Logger LOG = LoggerFactory.getLogger(OrdersController.class);

   @Autowired
   private OrderService orderService;

   @Autowired
   private AuthorizationService authorizationService;

   @Autowired
   private PaymentService paymentService;

   @PostMapping("/orders")
   public ResponseEntity<Long> saveOrder(@RequestHeader("access-token") String token,
@RequestBody Order order){
       try {
          if(authorizationService.isAuthorized(token, "USER")){
             if(paymentService.authorizePayment(order.getPayment())){
                Long orderId = orderService.saveNewOrder(order);
                return new ResponseEntity<>(orderId, HttpStatus.CREATED);
             } else
                throw new ResponseStatusException(HttpStatus.NOT_FOUND);
          } else {
             LOG.error("Unauthorized request for new Order");
             throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
          }
       } catch (Exception e){
          LOG.error(e.getMessage());
          throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR);
       }
   }
}
```

Order.java

```
package com.bitbyte.cargaraage.api.entities;
import com.bitbyte.cargaraage.api.models.Payment;
import com.fasterxml.jackson.annotation.JsonSetter;
import javax.persistence.*;
import java.util.Date;
import java.util.Set;
```

```
@Entity
public class Order {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   private Long userId;

   @Transient
   @JsonSetter
   private Payment payment;

   @ManyToOne(optional = false)
   @JoinColumn(name = "userId", referencedColumnName = "id", insertable = false, updatable
= false)
   private User user;

   @Temporal(value = TemporalType.DATE)
   private Date orderDate;

   @OneToMany(cascade = CascadeType.REMOVE, mappedBy = "order", targetEntity =
OrderDetails.class, fetch = FetchType.EAGER)
   private Set<OrderDetails> orderDetails;

   @OneToOne(cascade = CascadeType.ALL)
   @JoinColumn(name = "shippingAddressId", referencedColumnName = "id")
   private ShippingAddress shippingAddress;

   public Long getId() {
      return id;
   }

   public void setId(Long id) {
      this.id = id;
   }

   public Long getUserId() {
      return userId;
   }

   public void setUserId(Long userId) {
      this.userId = userId;
   }
```

```
    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }

    public Date getOrderDate() {
        return orderDate;
    }

    public void setOrderDate(Date orderDate) {
        this.orderDate = orderDate;
    }

    public Set<OrderDetails> getOrderDetails() {
        return orderDetails;
    }

    public void setOrderDetails(Set<OrderDetails> orderDetails) {
        this.orderDetails = orderDetails;
    }

    public ShippingAddress getShippingAddress() {
        return shippingAddress;
    }

    public void setShippingAddress(ShippingAddress shippingAddress) {
        this.shippingAddress = shippingAddress;
    }

    public Payment getPayment() {
        return payment;
    }

    public void setPayment(Payment payment) {
        this.payment = payment;
    }
}
```

OrderDetailsId.java

```
package com.bitbyte.cargaraage.api.entities;
import java.io.Serializable;
import java.util.Objects;

public class OrderDetailsId implements Serializable {

    private Long orderId;
    private Long userId;
    private String carId;

    public OrderDetailsId() {
    }

    public OrderDetailsId(Long orderId, Long userId, String carId) {
        this.orderId = orderId;
        this.userId = userId;
        this.carId = carId;
    }

    public Long getOrderId() {
        return orderId;
    }

    public void setOrderId(Long orderId) {
        this.orderId = orderId;
    }

    public Long getUserId() {
        return userId;
    }

    public void setUserId(Long userId) {
        this.userId = userId;
    }

    public String getCarId() {
        return carId;
    }

    public void setCarId(String carId) {
        this.carId = carId;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        OrderDetailsId that = (OrderDetailsId) o;
        return orderId.equals(that.orderId) &&
                userId.equals(that.userId) &&
                carId.equals(that.carId);
    }

    @Override
    public int hashCode() {
        return Objects.hash(orderId, userId, carId);
    }
}
```

Role.java

```
package com.bitbyte.cargaraage.api.entities;
import com.fasterxml.jackson.annotation.JsonInclude;
import javax.persistence.*;
import java.util.HashSet;
import java.util.Set;

@Entity(name = "roles")
@JsonInclude(JsonInclude.Include.NON_NULL)
public class Role {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id")
    int id;

    @Column(unique = true)
    private String role;

    private String roleDescription;

    @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    @JoinTable(inverseJoinColumns = @JoinColumn(name = "userId", referencedColumnName
= "id"),
            joinColumns = @JoinColumn(name = "roleId", referencedColumnName = "id"))
    private Set<User> users;

    public Role(String role) {
        this.role = role;
    }

    public Role() {
    }
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getRole() {
        return role;
    }

    public void setRole(String role) {
        this.role = role;
    }

    public Set<User> getUsers() {
        return users;
    }

    public String getRoleDescription() {
        return roleDescription;
    }

    public void setRoleDescription(String roleDescription) {
        this.roleDescription = roleDescription;
    }

    public void setUsers(Set<User> users) {
        this.users = users;
    }

    public void addUser(User user) {
        if (users == null) {
            users = new HashSet<>();
        }
        users.add(user);
    }
}
```

ShippingAddress.java

```
package com.bitbyte.cargaraage.api.entities;
import javax.persistence.*;

@Entity
```

```
public class ShippingAddress {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
   private String street;
   private String unit;
   private String city;
   private String state;
   private String country;
   private Integer zipcode;

   @OneToOne(mappedBy = "shippingAddress")
   private Order order;

   public Long getId() {
      return id;
   }

   public void setId(Long id) {
      this.id = id;
   }

   public String getStreet() {
      return street;
   }

   public void setStreet(String street) {
      this.street = street;
   }

   public String getUnit() {
      return unit;
   }

   public void setUnit(String unit) {
      this.unit = unit;
   }

   public String getCity() {
      return city;
   }

   public void setCity(String city) {
      this.city = city;
```

```
    }

    public String getState() {
        return state;
    }

    public void setState(String state) {
        this.state = state;
    }

    public String getCountry() {
        return country;
    }

    public void setCountry(String country) {
        this.country = country;
    }

    public Integer getZipcode() {
        return zipcode;
    }

    public void setZipcode(Integer zipcode) {
        this.zipcode = zipcode;
    }

    public Order getOrder() {
        return order;
    }

    public void setOrder(Order order) {
        this.order = order;
    }
}
```

User.java

```
package com.bitbyte.cargaraage.api.entities;
import com.bitbyte.cargaraage.api.models.Gender;
import com.bitbyte.cargaraage.api.models.Status;
import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import javax.persistence.*;
import java.util.Date;
import java.util.Set;
```

```
@Entity
@JsonIgnoreProperties(value = "password", allowSetters = true)
public class User {

   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;
   private String firstName;
   private String lastName;

   @Column(unique = true)
   private String userName;
   private String password;
   private Gender gender;
   private String email;
   private Date dob;

   @JsonIgnore
   private byte[] salt;

   @JsonIgnore
   private String hashingAlgorithm;
   private Status status;

   @OneToMany(mappedBy = "user", targetEntity = Order.class, fetch = FetchType.LAZY)
   private Set<Order> orders;

   @ManyToMany(mappedBy = "users", fetch = FetchType.EAGER)
   private Set<Role> roles;

   @OneToMany(mappedBy = "user", targetEntity = OrderDetails.class, fetch =
FetchType.LAZY)
   private Set<OrderDetails> orderDetailsSet;

   public Long getId() {
      return id;
   }

   public void setId(Long id) {
      this.id = id;
   }

   public String getFirstName() {
      return firstName;
   }
```

```
public void setFirstName(String firstName) {
   this.firstName = firstName;
}

public String getLastName() {
   return lastName;
}

public void setLastName(String lastName) {
   this.lastName = lastName;
}

public String getUserName() {
   return userName;
}

public void setUserName(String userName) {
   this.userName = userName;
}

public String getPassword() {
   return password;
}

public void setPassword(String password) {
   this.password = password;
}

public Gender getGender() {
   return gender;
}

public void setGender(Gender gender) {
   this.gender = gender;
}

public String getEmail() {
   return email;
}

public void setEmail(String email) {
   this.email = email;
}
```

```
public Date getDob() {
   return dob;
}

public void setDob(Date dob) {
   this.dob = dob;
}

public byte[] getSalt() {
   return salt;
}

public void setSalt(byte[] salt) {
   this.salt = salt;
}

public String getHashingAlgorithm() {
   return hashingAlgorithm;
}

public void setHashingAlgorithm(String hashingAlgorithm) {
   this.hashingAlgorithm = hashingAlgorithm;
}

public Status getStatus() {
   return status;
}

public void setStatus(Status status) {
   this.status = status;
}

public Set<Order> getOrders() {
   return orders;
}

public void setOrders(Set<Order> orders) {
   this.orders = orders;
}

public Set<Role> getRoles() {
   return roles;
}

public void setRoles(Set<Role> roles) {
        this.roles = roles;
    }

    public Set<OrderDetails> getOrderDetailsSet() {
        return orderDetailsSet;
    }

    public void setOrderDetailsSet(Set<OrderDetails> orderDetailsSet) {
        this.orderDetailsSet = orderDetailsSet;
    }
}
```

DecodedToken.java

```
package com.bitbyte.cargaraage.api.models;

public class DecodedToken {
    private final String header;
    private final String payload;
    private boolean validExpiration;
    private boolean validIssuer;
    private boolean authorized;

    public DecodedToken(String header, String payload) {
        this.header = header;
        this.payload = payload;
    }

    public String getHeader() {
        return header;
    }

    public String getPayload() {
        return payload;
    }

    public boolean isValidExpiration() {
        return validExpiration;
    }

    public void setValidExpiration(boolean validExpiration) {
        this.validExpiration = validExpiration;
    }

    public boolean isValidIssuer() {
        return validIssuer;
    }

    public void setValidIssuer(boolean validIssuer) {
        this.validIssuer = validIssuer;
    }

    public boolean isAuthorized() {
        return authorized;
    }

    public void setAuthorized(boolean authorized) {
        this.authorized = authorized;
    }
}
```

Gender.java

```
package com.bitbyte.cargaraage.api.models;

public enum Gender {
    Male, Female, Other
}
```

Payment.java

```
package com.bitbyte.cargaraage.api.models;

public class Payment {
    private Integer card;
    private Integer exp;
    private Integer cvc;
    private String name;
    private Integer total;

    public Integer getCard() {
        return card;
    }

    public void setCard(Integer card) {
        this.card = card;
    }

    public Integer getExp() {
        return exp;
    }

    public void setExp(Integer exp) {
        this.exp = exp;
    }

    public Integer getCvc() {
        return cvc;
    }

    public void setCvc(Integer cvc) {
        this.cvc = cvc;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getTotal() {
        return total;
    }

    public void setTotal(Integer total) {
        this.total = total;
    }
}
```

Status.java

```
package com.bitbyte.cargaraage.api.models;

public enum Status {
    ACTIVE, INACTIVE, DELETED
}
```

OrderDetailsRepository.java

```
package com.bitbyte.cargaraage.api.repositories;
import com.bitbyte.cargaraage.api.entities.OrderDetails;
import com.bitbyte.cargaraage.api.entities.OrderDetailsId;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface OrderDetailsRepository extends CrudRepository<OrderDetails, OrderDetailsId>
{
}
```

OrderRepository.java

```
package com.bitbyte.cargaraage.api.repositories;
import com.bitbyte.cargaraage.api.entities.Order;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface OrderRepository extends CrudRepository<Order, Long> {
}
```

RoleRepository.java

```
package com.bitbyte.cargaraage.api.repositories;
import com.bitbyte.cargaraage.api.entities.Role;
import org.springframework.data.repository.CrudRepository;

public interface RoleRepository extends CrudRepository<Role, Long> {
}
```

ShippingAddressRepository.java

```
package com.bitbyte.cargaraage.api.repositories;
import com.bitbyte.cargaraage.api.entities.ShippingAddress;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ShippingAddressRepository extends CrudRepository<ShippingAddress, Long>
{
}
```

UserRepository.java

```
package com.bitbyte.cargaraage.api.repositories;
import com.bitbyte.cargaraage.api.entities.User;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends CrudRepository<User, Long> {
}
```

AdminService.java

```
package com.bitbyte.cargaraage.api.services;
import com.bitbyte.cargaraage.api.entities.Role;
import com.bitbyte.cargaraage.api.repositories.RoleRepository;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class AdminService {

    private static final Logger LOG = LoggerFactory.getLogger(AdminService.class);

    @Autowired
    private RoleRepository roleRepository;

    public Role createRole(Role role){
        LOG.info("AdminService -> createRole -> start");
        Role createdRole = roleRepository.save(role);
        LOG.info("AdminService -> createRole -> end");
        return createdRole;
    }
}
```

AuthorizationService.java

```
package com.bitbyte.cargaraage.api.services;
import com.bitbyte.cargaraage.api.entities.User;
import com.bitbyte.cargaraage.api.models.DecodedToken;
import com.bitbyte.cargaraage.api.models.Status;
import com.bitbyte.cargaraage.api.repositories.UserRepository;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import static org.springframework.security.jwt.JwtHelper.decode;
import org.springframework.stereotype.Service;
import java.util.Optional;

@Service
public class AuthorizationService {

    @Autowired
    private UserRepository userRepository;

    private static final Logger LOG = LoggerFactory.getLogger(AuthorizationService.class);

    public boolean isAuthorized(String token, String role){
        DecodedToken decodedToken = new DecodedToken(decode(token).toString(),
decode(token).getClaims());
        validateTokenExpiration(decodedToken);
        validateIssuer(decodedToken);
        try {
            JsonObject jsonObject = new
JsonParser().parse(decodedToken.getPayload()).getAsJsonObject();
            Long userId = jsonObject.get("userId").getAsLong();
            Optional<User> user = userRepository.findById(userId);
            if (user.isPresent() && user.get().getRoles().stream().filter(r ->
r.getRole().equals(role)).findAny().isPresent() && user.get().getStatus().equals(Status.ACTIVE)
&& decodedToken.isValidExpiration() && decodedToken.isValidIssuer()){
                return true;
            } else
                return false;
        } catch (Exception e){
            LOG.error(e.getMessage());
            return false;
        }
    }

    private void validateIssuer(DecodedToken decodedToken) {
        try {
            JsonObject jsonObject = new
JsonParser().parse(decodedToken.getPayload()).getAsJsonObject();
            if(jsonObject.get("iss").getAsString().equals("")){
                decodedToken.setValidIssuer(true);
            } else
                decodedToken.setValidIssuer(false);
        } catch (Exception e){
            LOG.error(e.getMessage());
            decodedToken.setValidIssuer(false);
        }
    }

    private void validateTokenExpiration(DecodedToken decodedToken) {
        try {
            JsonObject jsonObject = new
JsonParser().parse(decodedToken.getPayload()).getAsJsonObject();
            if(isExpirationValidated(jsonObject.get("exp").getAsLong())){
                decodedToken.setValidExpiration(true);
            } else {
                decodedToken.setValidExpiration(false);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private boolean isExpirationValidated(Long dateToCheckInMillis) {
        return System.currentTimeMillis() < (dateToCheckInMillis.longValue() * 1000);
    }
}
```

OrderService.java

```
package com.bitbyte.cargaraage.api.services;
import com.bitbyte.cargaraage.api.entities.Order;
import com.bitbyte.cargaraage.api.entities.OrderDetails;
import com.bitbyte.cargaraage.api.entities.User;
import com.bitbyte.cargaraage.api.repositories.OrderDetailsRepository;
import com.bitbyte.cargaraage.api.repositories.OrderRepository;
import com.bitbyte.cargaraage.api.repositories.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.Optional;
import java.util.Set;

@Service
public class OrderService {

    @Autowired
    private OrderRepository orderRepository;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private OrderDetailsRepository orderDetailsRepository;

    public Long saveNewOrder(Order order){
        Long userId = order.getUserId();
        Optional<User> user = userRepository.findById(userId);
        order.setUser(user.get());
        Order savedOrder = orderRepository.save(order);
        Set<OrderDetails> orderDetailsSet = order.getOrderDetails();
        orderDetailsSet.forEach(orderDetails -> {
            orderDetails.setOrderId(savedOrder.getId());
            orderDetails.setUserId(savedOrder.getUserId());
        });
        orderDetailsRepository.saveAll(orderDetailsSet);
        return savedOrder.getId();
    }
}
```