6-2018

EPICCONFIGURATOR COMPUTER CONFIGURATOR AND CMS PLATFORM

IVO A. TANTAMANGO
tantamai@coyote.csusb.edu

EPICCONFIGURATOR COMPUTER CONFIGURATOR AND CMS PLATFORM

A Project

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

by

Ivo Alejandro Tantamango

June 2018

EPICCONFIGURATOR COMPUTER CONFIGURATOR AND CMS PLATFORM

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Ivo Alejandro Tantamango

June 2018

Approved by:

David Turner, Advisor, Computer Science & Engineering

Ernesto Gomez, Committee Member

Kay Zemoudeh, Committee Member

ABSTRACT

Very often when we are looking to buy new IT equipment in an online store, we face the problem that certain parts of our order are not compatible with others or sometimes one part needs additional components. From another point of view in this process, when an online store owner wants to manage the products available in stock, assign prices, set conditions to make an order, or manage customer information, he or she must often rely on information from different systems, physical files, or other sources.

EpicConfigurator simplifies and solves the issues mentioned above. EpicConfigurator makes it easy for User Customers to configure computer products by making the process of product selection more straightforward. It can actively gather customer requirements and map them to a set of products and service options. These capabilities will guide users towards an optimal solution for their needs.

EpicConfigurator also allows User Customers to keep track and edit saved product configurations. This system also includes a user administrator perspective that allows Store Owners to act as User Admins helping them to manage and load new products, set configuration rules for products and manage all users.

Following open source technologies, EpicConfigurator is an application easy to enhance, expand and integrate with newer technologies.

This is a configurator tool and does not provide any purchasing feature. To purchase, the configuration results should be provided to the local reseller or sales representative to get an official quote.

ACKNOWLEDGEMENTS

I would like to first thank my Project Adviser, Dr. Turner, and my Project committee, formed by Dr. Gomez and Dr. Kay Zemoudeh, for supporting me on this project. I also want to thank my School Coordinator, Dr. Josephine Mendoza, for her readiness to help and advise me on any query related to enrollment and in choosing the right classes, as well as all of the professors that were part of this MS program. I appreciate all their efforts to teach me and help me on this journey; I will always be indebted to them because of that. Last but not least, I want to thank my wife and family for their support, and for always being there when I need them the most.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1 Background

For this Master's project, I developed a computer configurator tool called EpicConfigurator. The idea for this project was born while working as a Software Engineer for a computer manufacturing company; I noticed that most of their systems had a configuration engine that validated the products that customers were adding in order to get a quote of the order. This system intends to cover that aspect of the process and also provides a management perspective that allows a user admin to manage products, set configuration rules, and manage customer information.

1.2 Purpose

The purpose of this project is to create a comprehensive platform that can grow in functionality and be implemented for other computer resellers as a tool to manage their clients, products and to configure computer products. Also a personal goal of mine is to increase my experience using mature open source technologies and learn more on how to architect a new application, provide solutions to requirements, and use this project as a base to add new functionalities using newer technologies.

1.3 Project Scope

The report will detail main functionalities and describe the flows of the application for different users of the system. It will also explain the reasoning behind the technologies being used and implementation details.

This system has 2 user types:

Customer User and Admin Users.

The Customer User can:

- Register
- Search products
- Create Configurations
- Save Configurations
- Edit Configurations
- Delete Configurations
- Print a Configuration or List of Configurations

The Admin User can:

- Manage Users
- Manage Products
- Load Configuration Rules

1.4 Project Limitations

This application will not cover any purchase flow.

1.5 Development Tools

This project was developed using Java programming language, Spring Framework (including its modules Spring Core, Spring MVC and Spring Security) and Apache Solr to perform indexing and search of different products. For the Front End side of the application I used technologies such as JavaScript, JQuery (to handle AJAX invocations and animation of HTML elements) and Bootstrap for a responsive design of HTML and CSS. In the Back end of the application I used Java Server Pages (JSP), Spring MVC Controllers and Hibernate ORM to have a mapping between Java Objects and Database Tables.

1.5.1 J2EE

Java Platform, Enterprise Edition (Java EE) is a programming language used to develop enterprise software. Java EE follows the Java Community Process, which is an open community that gets contributions from industry experts, commercial and open source organizations, Java User Groups, and many global users. This programming language add constantly new features that satisfy industry needs and application portability in every new release. [1]

1.5.2 Hibernate

Hibernate ORM is a framework that helps developers to create applications that use a persistent layer, this means application which data created will persist even after the execution of the program. Hibernate framework is a great Object/Relational Mapping (ORM) tool, it also provides ready to use functions that easy the implementation of common requirements when interacting with the Data Base, instead of accessing it just via JDBC. Hibernate uses JDBC internally and also allows the developer to perform native SQL queries if required. [2]

1.5.3 Spring MVC

Spring Web MVC framework defines a Model-View-Controller (MVC) pattern that provides a ready to use framework that simplifies the implementation of this pattern by the developer. Spring MVC framework components help to develop very flexible and loose coupled web applications. The MVC design pattern separates different application aspects that will include the reception and handling of the input performed by the user, the business logic in the backend of the application that will process this input and the return of the resulting User Interface (UI). A description of Spring MVC components is as follow:

- The Model encapsulates the application data which consists of plain old Java object (POJO) classes.
- The View renders the model data and generates HTML output that can be interpreted by the client's browser.

- The Controller processes the user requests, builds an appropriate model, and passes it to the View for rendering.

The Spring Web MVC framework has as a main element the servlet DispatcherServlet. This one handles all HTTP requests and responses that interact with the application. The interaction between the elements of the Spring Web MVC DispatcherServlet is illustrated in the Figure 1.1.

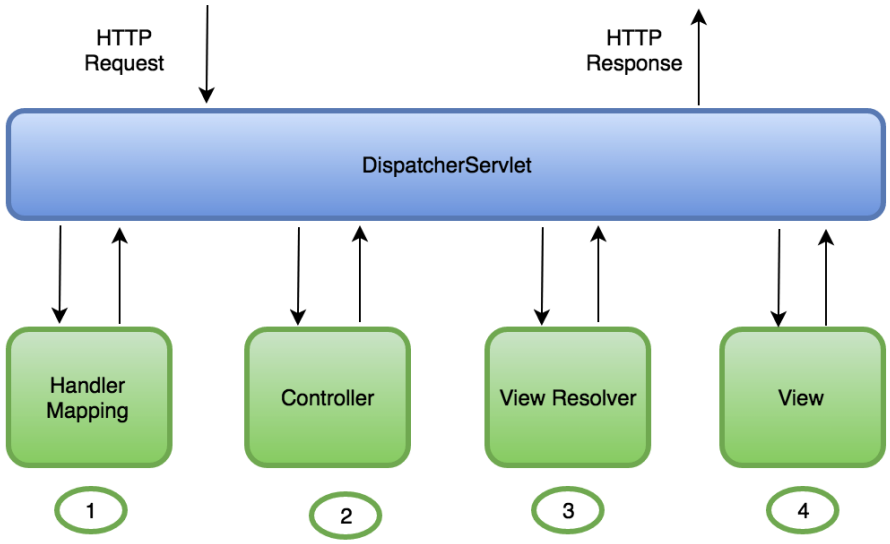

Figure 1.1. Spring MVC Architecture

The sequence of events corresponding to the interaction of an incoming HTTP request to DispatcherServlet are as follows:

After receiving an HTTP request, the DispatcherServlet consults the HandlerMapping to call the appropriate Controller. Depending on the request method (GET or POST), the Controller calls the appropriate service methods.

These service methods will generate or receive model data and will return a specific view name to the DispatcherServlet.

The DispatcherServlet will rely on the ViewResolver interface which maps view names to actual views and picks up the defined view for the request.

Once view is defined by the ViewResolver, the DispatcherServlet passes the model data to the View, which is finally rendered in HTML.

All the components mentioned such as HandlerMapping, Controller, and ViewResolver are parts of the WebApplicationContext. This is an extension of the plain ApplicationContext interface with some extra features necessary for web applications. [3]

1.5.4 Spring Security

Spring Security provides a very wide-ranging security solution for Java web based software applications. Spring Security provides support to the application security layer, as well as different implementations that satisfy assorted requirements for numerous business problem domains.

Spring Security targets two major areas of application security: "authentication" and "authorization" (or "access-control"). "Authentication" is the process of validating if an actor is who he or she claims to be, and "authorization" is the process of verifying if an actor can perform an action within the application. Spring Security supports integration with the following technologies:

- HTTP BASIC authentication
- HTTP X.509 client certificate exchange

- LDAP authentication
- Form-based authentication (for simple user interface needs)
- Java Authentication and Authorization Service (JAAS)
- Out of the Box "remember-me" authentication (depending on configuration this avoids user re-authentication for a predetermined period of time).

Spring Security is very customizable and can provide a seamless integration with other systems if needed.

The elements related to authentication and authorization in Spring Security are described in the Spring Security namespace. The major building blocks of Spring Security are the following components:

- UserDetails: This object is an implementation of the UserDetailsService Interface and provides the necessary information to the authentication provider of the security framework. The UserDetails implementation will store a string for the username as well as a string for the password, and it will have a collection of objects called GrantedAuthority.
- GrantedAuthority: This object keeps track of the different permissions granted to a principal.
- AuthenticationManager: This interface obtains the authorizations contained in the UserDetails object, and will delegate this task to an AuthenticationProvider instance.
- AuthenticationProvider: The implementation of this interface will call an object that implements the UserDetailsService interface and return

a UserDetails object fully populated. After the AuthenticationProvider has returned this object, it will check if the credentials entered are valid and in a web-aware application like the one presented in this report. Any interaction with the application will be filtered by a servlet filter known as SpringSecurityFilterChain, which is responsible for all the security (protecting application URLs, validating submitted username and passwords, redirecting to the login form, etc.).

The setup used for Spring Security is a Namespace configuration; this is XML- based and has an <http> element that contains the URL to be intercepted, the form login, and the success or failure page, this can be seen in Figure 1.2.

```
<http use-expressions="true" pattern="/jsp/admin**" >
    <intercept-url pattern="/jsp/admin**" access="hasRole('ROLE_ADMIN')" />
    <access-denied-handler error-page="/403" />
    <form-login login-processing-url="/j_spring_security_check"
    login-page='/jsp/loginadmin.html'
        authentication-failure-url="/jsp/loginadmin.html?error"
        authentication-success-handler-ref="myAuthenticationSuccessHandler"/>
    <logout/>
     <csrf/>
</http>
```

Figure 1.2. Spring Security Configuration

Spring Security implementation also prevents Cross-Site Request Forgery (CSRF) attacks [4], by the addition of a randomly generated token passed to the user request as an HTTP parameter.

1.5.5. Bootstrap

Bootstrap is one of the most widespread UI frameworks for CSS, HTML, and JavaScript; it is free and open source. Bootstrap is known as one of the best CSS frameworks for building responsive web applications. Bootstrap is a mobile-first framework; this means that anything you design or create will be mobile compatible or responsive.

Bootstrap requires HTML5 doctype to handle HTML elements and CSS properties. This must be included at the beginning of all Bootstrap projects.

Bootstrap work with containers these ones include the elements used by the framework [5]. It only uses two types of containers as seen in Figure 1.3.

Figure 1.3. Bootstrap Containers

Bootstrap also provides a wide number of reusable components, including iconography, dropdowns, input groups, navigation, alerts, and more.

Bootstrap comes with several JavaScript components in the form of jQuery plugins. These provide additional user interface elements such as dialog boxes, tooltips, and carousels. They also extend the functionality of some existing interface elements, including autocomplete function for input fields. [6]

1.5.6. Apache Solr

Solr is a standalone enterprise search engine with a REST-like API that allows querying Solr indexed data. Querying this indexed data via HTTP GET will produce a response in JSON, XML, CSV, or binary format. [7].

Apache Solr treats data as documents and falls in the category of NoSQL databases. Apache Solr provides the following features:

- Advanced full-text search capabilities.
- Great performance for high volume traffic.
- It is based on Open Interfaces - XML, JSON and HTTP.
- Comprehensive administration interfaces, with a UI web administration console.
- Highly scalable and fault tolerant.
- Near real-time indexing using Apache Lucene high-performance, text search engine [8].

By performing different queries to the indexed data, Solr can make use of the following features:

- Pagination and sorting
- Faceting
- Autosuggest
- Spell-checking
- Geospatial search

An example of the Solr query syntax is shown in Figure 1.4.

```
http://localhost:8983/solr/query?debug=query&q=hello

{
  "responseHeader":{
    "status":0,
    "QTime":0,
    "params":{
      "q":"hello",
      "debug":"query"}},
  "response":{"numFound":0,"start":0,"docs":[]
  },
  "debug":{
    "rawquerystring":"hello",
    "querystring":"hello",
    "parsedquery":"text:hello",
    "parsedquery_toString":"text:hello",
    "QParser":"LuceneQParser"}
}
```

Figure 1.4. Solr Query Syntax

1.6. Definitions, Acronyms, and Abbreviations

- AJAX: Asynchronous JavaScript And XML.

- ApplicationContext: Central interface to provide configuration for an application.
- Class: A custom data structure that defines properties and methods of an object.
- CSS: Cascading Style Sheets.
- CSRF: Cross Site Request Forgery, is a type of malicious attack that uses unauthorized commands to a web application using saved data from a user's session.
- DAO: Data Access Object.
- DI: Dependency Injection, this is a technique used by Spring Core that allows the ApplicationContext to create and assign instances (dependencies) of to another Object that requires them, instead of this last Object creating the dependencies inside of itself [10].
- GUI: Graphical User Interface.
- IDE: Integrated Development Environment.
- Interface: Similar than a Java class but they are mostly used to define methods without implementation.
- JRE: Java Runtime Environment
- JDBC: Java Database Connectivity. Java Interface, which defines to access a database
- JVM: Java Virtual Machine.
- JSON: Java Script Object Notation.

- JPA: Java Persistence API.
- JSP: Java Server Page.
- Metadata: set of data that describes and gives information about other data.
- Metadata: Data that describes other data. The prefix Meta means definition or description.
- Namespace: In computer science this is a region or container that holds different special functions, names and identifiers.
- Object: An instance of a class.
- Property: A variable that belongs to a class.
- POJO: Plain Old Java Object.
- Solr Indexing: Uploading Data to Solr Server.
- UI: User Interface.
- XML: Extensible Markup Language.

2. SYSTEM DESIGN AND ARCHITECTURE

2.1 Project Design

EpicConfigurator is a web based application that uses Java programming language for the backend layer and JSPs, CSS, and JavaScript libraries in the front-end layer. The persistent layer is implemented using Hibernate framework, and is deployed in an Apache Tomcat server. The database engine chosen for this application is MySQL.

2.2 Actors

There are 3 main actors in the system: Guest User, Logged-in User, and Admin User.

2.2.1 Guest User

A user new to the system has the option to browse through the categories available and search for a specific product, but he or she cannot create and save configurations. A Guest User can create an account. Guest User Use Cases are shown in Figure 2.1.

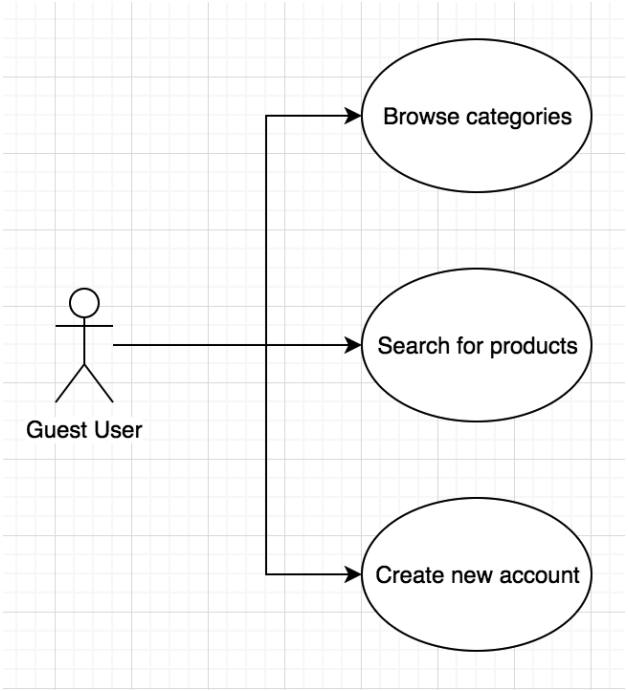

Figure 2.1. Guest User Use Cases

2.2.2 Logged-in User

A Logged-in User has the same permissions to search and browse for products and categories as a Guest User, but this one can also manage (create, edit, and remove) a Product Configuration. Additionally, this user can download a report of all the saved configurations and a detailed report of a single configuration. Logged in User Use Cases are shown in Figure 2.2.

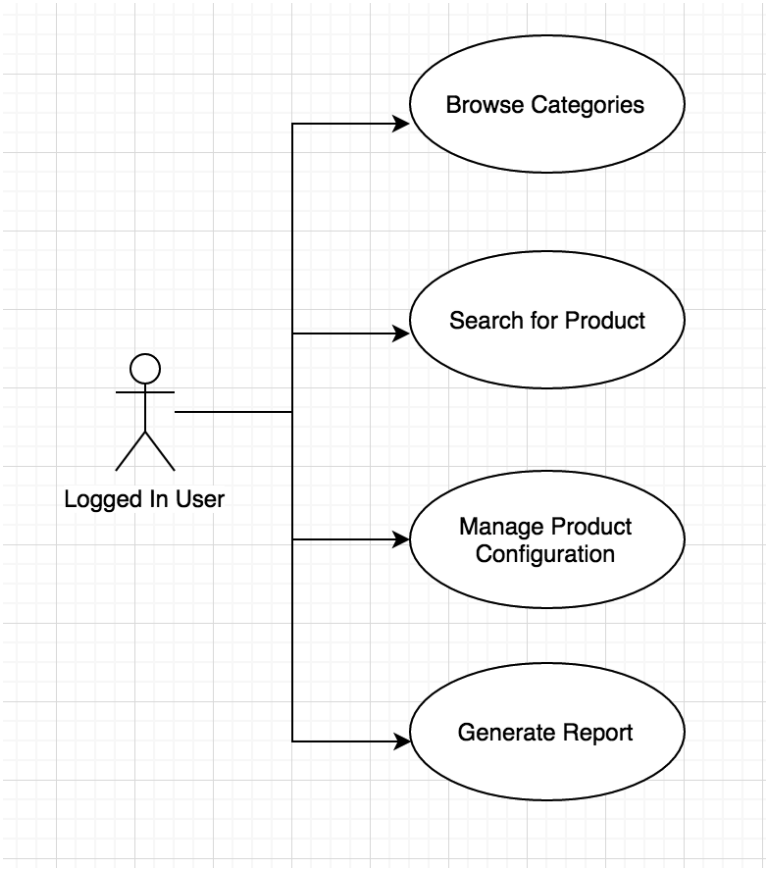

Figure 2.2 Logged in User Use Cases

2.2.3 Admin User

The Admin User can manage products and users. The Admin User can add new products to the system in bulk by uploading a .csv file or by adding one single product at a time. The Admin User can also add configuration rules for products, this is achieved by uploading a .csv file into the system. This .csv file contains the configuration restrictions or rules for adding parts to a certain product. Admin User can also remove, and edit users and products in the

system, and also perform different search options. Admin User Use Cases are shown in Figure 2.3.

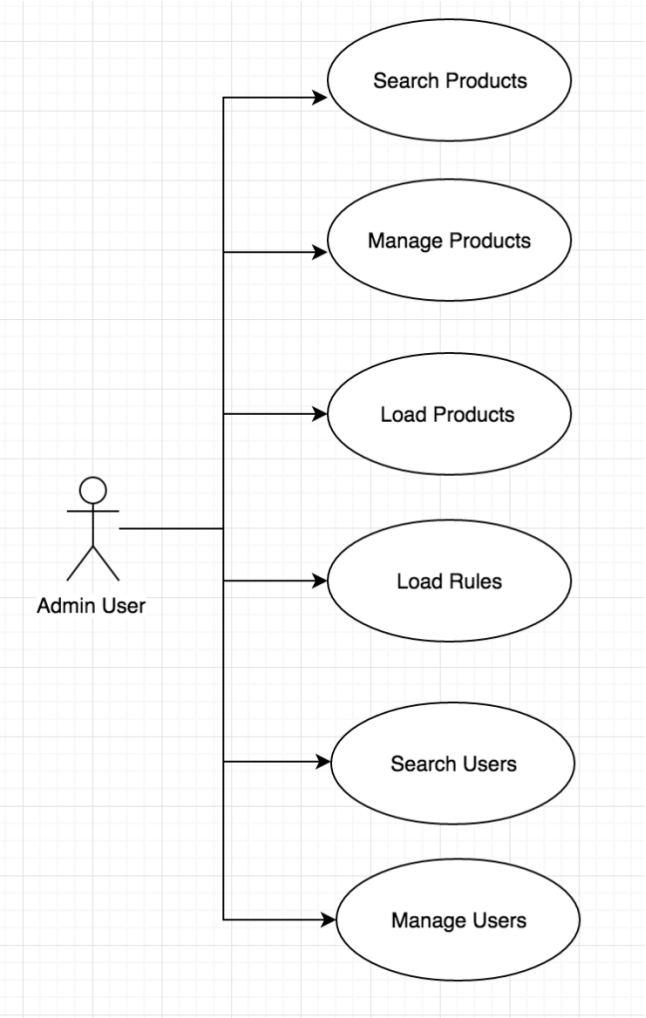

Figure 2.3. Admin User Use Cases

2.3 Design Patterns

The design patterns used in this application are the following:

- MVC Pattern

- DAO Pattern
- Factory Pattern

2.3.1 MVC

MVC Pattern stands for Model-View-Controller Pattern. This design pattern provides a good approach for separation of concerns in the application.

- Model – This layer contains the objects or POJOs used in the application. It is also responsible for updating the controller if data changes.
- View – This visualizes the data that the model provides. It is implemented in the application by JSPs.
- Controller – This layer “controls” the interactions between the model and the view layer.

2.3.2 DAO Pattern

The DAO design pattern is used in this project to provide a layer that includes data operations in the application. This pattern is based on abstraction and encapsulation principles, allowing specific calls to the persistence layer, this design pattern does not expose database implementation details.

To ensure better separation of concerns in the application, there is an additional service layer that interacts to the DAO layer to perform operations with persistent objects, the DAO implementation used in this project can be appreciated in Figure 2.4.

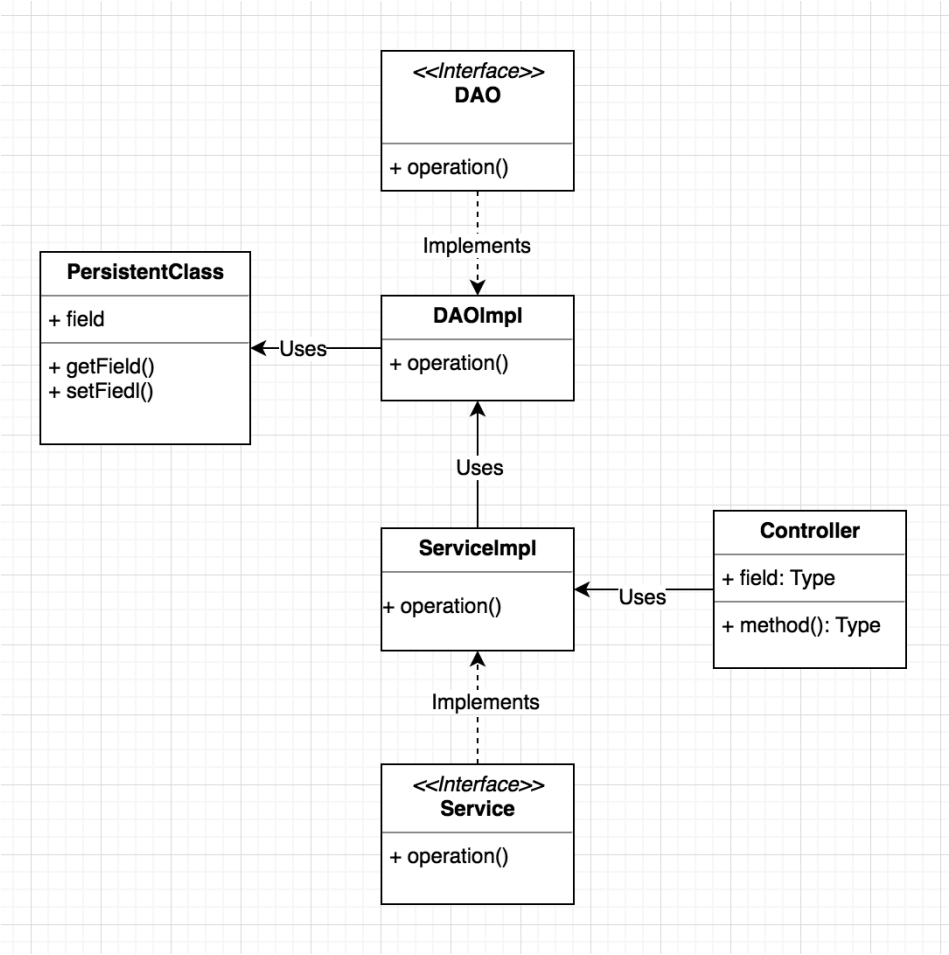

Figure 2.4. DAO Design Pattern

2.3.3 Factory Pattern

This Design pattern is implement by Spring Framework in its Core module. The creation of objects is not exposed to the requestor, the requestor can only access to the instances through a common interface, which in Spring are the ApplicationContext and BeanFactory interfaces, both have similarities and differences and one can be used instead of the other depending on the

developer needs, but both contain all the definitions of the beans used in the application and also define the DI to other beans.

2.4. Data Base Mapping and Metadata

As stated previously this project uses Hibernate ORM to map existing SQL tables to Java Objects. By the use of Annotations Hibernate provides a powerful way of mapping SQL tables metadata into Java files, this helps to understand the table structure and also makes easier the development process. The list of entities used in this project are displayed in Figure 2.5.

```
<bean id="hibernate4AnnotatedSessionFactory"
class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
<property name="dataSource" ref="dataSource" />
<property name="annotatedClasses">
    <list>
        <value>com.epic.ecommerce.core.model.Category</value>
        <value>com.epic.ecommerce.core.model.Country</value>
        <value>com.epic.ecommerce.core.model.Customer</value>
        <value>com.epic.ecommerce.core.model.Product</value>
        <value>com.epic.ecommerce.core.model.PartValidation</value>
        <value>com.epic.ecommerce.core.model.Menu</value>
        <value>com.epic.ecommerce.core.model.User</value>
        <value>com.epic.ecommerce.core.model.UserRole</value>
        <value>com.epic.ecommerce.core.model.ConfigurationProduct</value>
        <value>com.epic.ecommerce.core.model.QuoteConfiguration</value>
    </list>
</property>
<property name="hibernateProperties">
    <props>
        <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
        <prop key="hibernate.show_sql">true</prop>
    </props>
</property>
</bean>
```

Figure 2.5. Model Configuration and Object Relation Mapping

3. DATABASE DESIGN

The database model for this application has been created using MySQL Data Base.

3.1. Database Analysis

EpicConfigurator Data Base is composed of the following tables:

- CATEGORY
- CONFIGURATION_PRODUCT
- COUNTRY
- CUSTOMER
- MENU
- PART_PRODUCT
- PART_VALIDATION
- PRODUCT
- QUOTE_CONFIGURATION
- SUBCAT_CAT
- USER_ROLES
- USERS

The Entity Relational (ER) diagram for the EPIC_CONFIGURATOR system is shown in Figure 3.1.

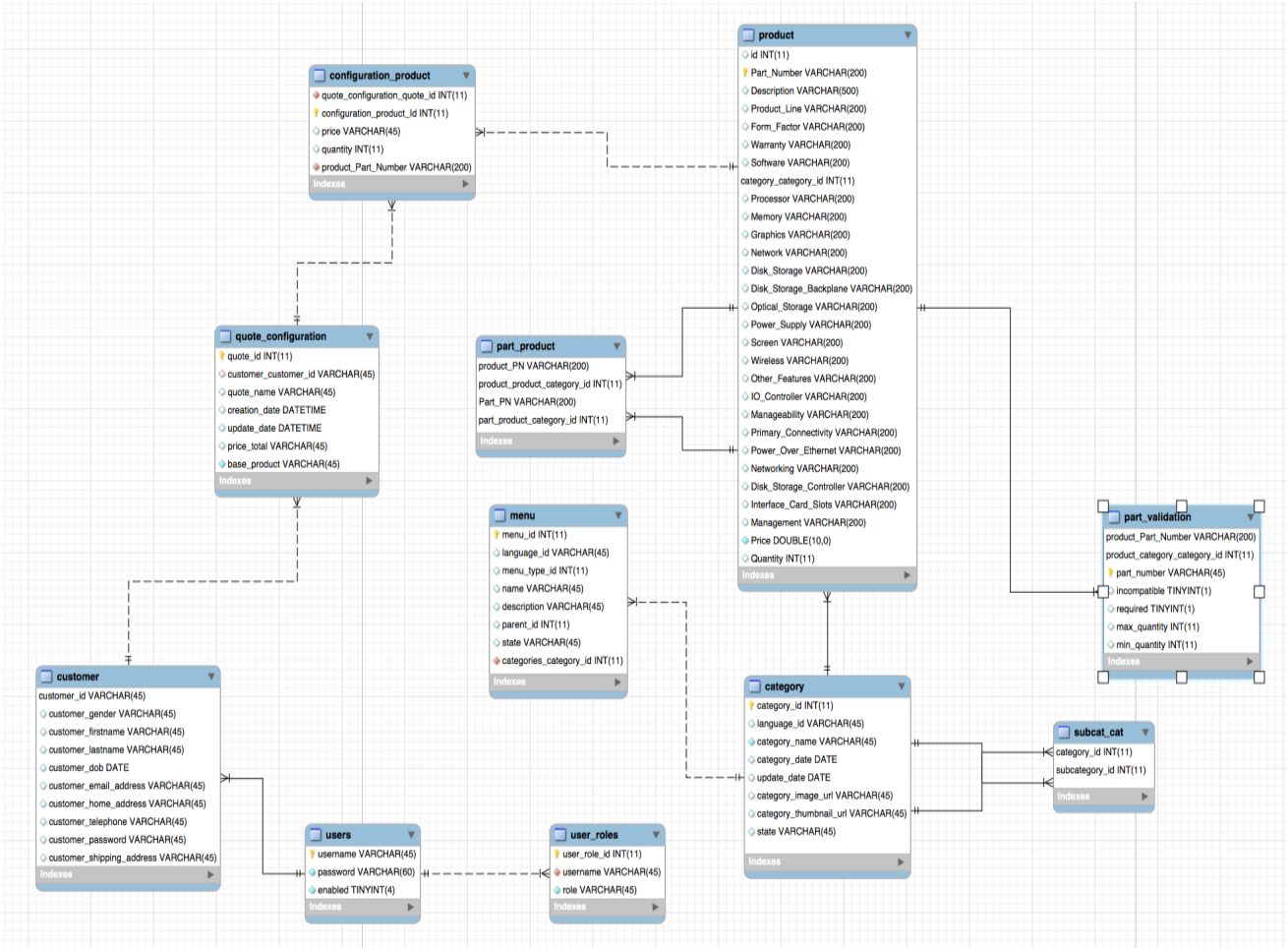

Figure 3.1. Entity Relational Diagram

3.1.1. CATEGORY Table

The CATEGORY table contains all the categories used in the creation of configurations. Every product belongs to a category, and a category can have other categories; the relationship between these categories is defined in the SUBCAT_CAT table, creating a self-Many-to-Many relationship. See Table 3.1 for details.

Table 3.1. CATEGORY Table

CATEGORY			
CATEGORY_ID	INT(11)	NOT NULL AUTO_INCREMENT	PK
LANGUAGE_ID	VARCHAR(45)	NULL DEFAULT NULL	
CATEGORY_NAME	VARCHAR(45)	NOT NULL	
CATEGORY_DATE	DATE	DATE NULL DEFAULT NULL,	
UPDATE_DATE	DATE	DATE NULL DEFAULT NULL,	
CATEGORY_IMAGE_URL	VARCHAR(45)	NULL DEFAULT NULL	
CATEGORY_THUMBNAIL_URL	VARCHAR(45)	NULL DEFAULT NULL	
STATE	VARCHAR(45)	NULL DEFAULT NULL	

3.1.2. SUBCAT_CAT Table

The SUBCAT_CAT table defines the self-reference relationship between elements that belong to the CATEGORY table, Figure 3.2. represents the relationship. See Table 3.2 for details.

Table 3.2. SUBCAT_CAT Table

SUBCAT_CAT			
CATEGORY_ID	INT(11)	NOT NULL	FK
SUBCATEGORY_ID	INT(11)	NOT NULL	FK

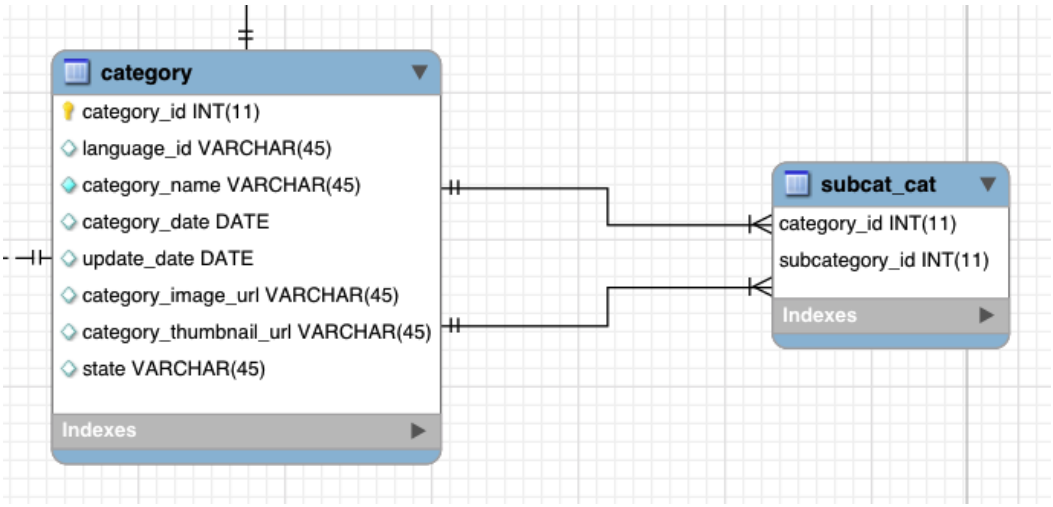

Figure 3.2. Category Table Self Many To Many Relationship

3.1.3. MENU Table

The MENU table contains a list of the menus which belong to one menu type. This is defined by the attribute MENU_TYPE_ID. This table has a dependent relationship with the Category Table, and allows a category to have one or multiple menus. See Table 3.3 and Figure 3.3 for details.

Table. 3.3. MENU Table

MENU			
MENU_ID	INT(11)	NOT NULL AUTO_INCREMENT	PK
LANGUAGE_ID	VARCHAR(45)	NULL DEFAULT NULL	
MENU_TYPE_ID	INT(11)	NULL DEFAULT NULL	

NAME	VARCHAR(45)	NULL DEFAULT NULL	
DESCRIPTION	VARCHAR(45)	NULL DEFAULT NULL	
PARENT_ID	INT(11)	NULL DEFAULT NULL	
STATE	VARCHAR(45)	NULL DEFAULT NULL	
CATEGORIES_CATEGORY_ID	INT(11)	NULL DEFAULT NULL	FK

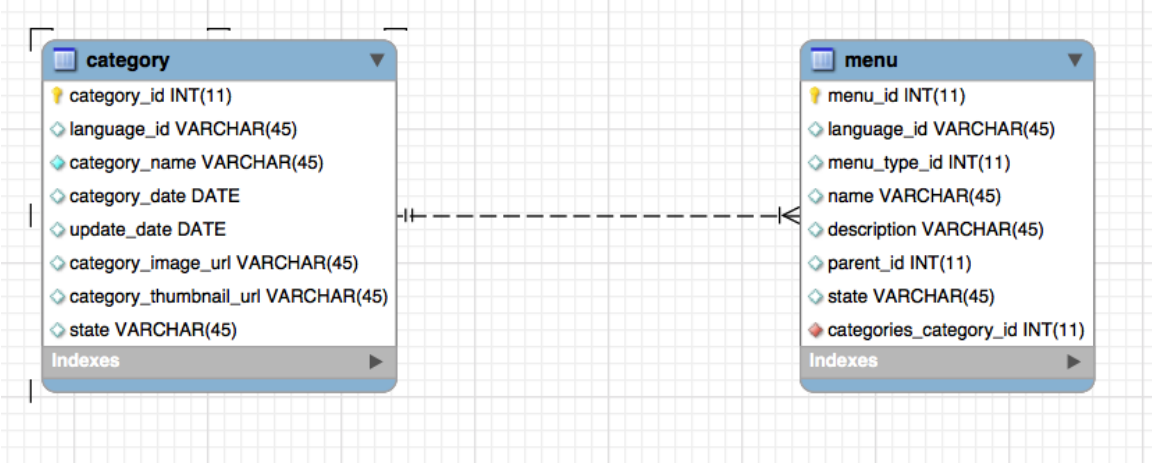

Figure 3.3. Category Menu Table Relationship

3.1.4. PRODUCT Table

Product Table is the main table in the system. It has a many-to-one relationship with the CATEGORY table, a self-many-to-many relationship implemented with PART_PRODUCT table, a one-to-many relationship with PART_VALIDATION table, and a one-to-many relationship with CONFIGURATION_PRODUCT. See Table 3.4 and Figure 3.4 for details.

Table 3.4. PRODUCT Table

PRODUCT			
ID	INT(11)	NOT NULL AUTO_INCREMENT	PK
PART_NUMBER	VARCHAR(45)	NULL DEFAULT NULL	
DESCRIPTION	VARCHAR(45)	NULL DEFAULT NULL	
PRODUCT_LINE	VARCHAR(45)	NULL DEFAULT NULL	
FORM_FACTOR	VARCHAR(45)	NULL DEFAULT NULL	
WARRANTY	DATE	NULL DEFAULT NULL	
SOFTWARE	VARCHAR(45)	NULL DEFAULT NULL	
CATEGORY_CATEGORY_ID	INT(11)	NOT NULL	FK
PROCESSOR	VARCHAR(45)	NULL DEFAULT NULL	
MEMORY	VARCHAR(200)	NULL DEFAULT NULL	
GRAPHICS	VARCHAR(200)	NULL DEFAULT NULL	
NETWORK	VARCHAR(200)	NULL DEFAULT NULL	
DISK_STORAGE	VARCHAR(200)	NULL DEFAULT NULL	
DISK_STORAGE_BACKPLANE	VARCHAR(200)	NULL DEFAULT NULL	
OPTICAL_STORAGE	VARCHAR(200)	NULL DEFAULT NULL	
POWER_SUPPLY	VARCHAR(200)	NULL DEFAULT NULL	
SCREEN	VARCHAR(200)	NULL DEFAULT NULL	
WIRELESS	VARCHAR(200)	NULL DEFAULT NULL	
OTHER_FEATURES	VARCHAR(200)	NULL DEFAULT NULL	
IO_CONTROLLER	VARCHAR(200)	NULL DEFAULT NULL	
MANAGEABILITY	VARCHAR(200)	NULL DEFAULT NULL	

PRIMARY_CONNECTIVITY	VARCHAR(200)	NULL DEFAULT NULL	
POWER_OVER_ETHERNET	VARCHAR(200)	NULL DEFAULT NULL	
NETWORKING	VARCHAR(200)	NULL DEFAULT NULL	
DISK_STORAGE_CONTROLLER	VARCHAR(200)	NULL DEFAULT NULL	
INTERFACE_CARD_SLOTS	VARCHAR(200)	NULL DEFAULT NULL	
MANAGEMENT	VARCHAR(200)	NULL DEFAULT NULL	
PRICE	DOUBLE(10,0)	NOT NULL DEFAULT '0'	
QUANTITY	INT(11)	NULL DEFAULT NULL	

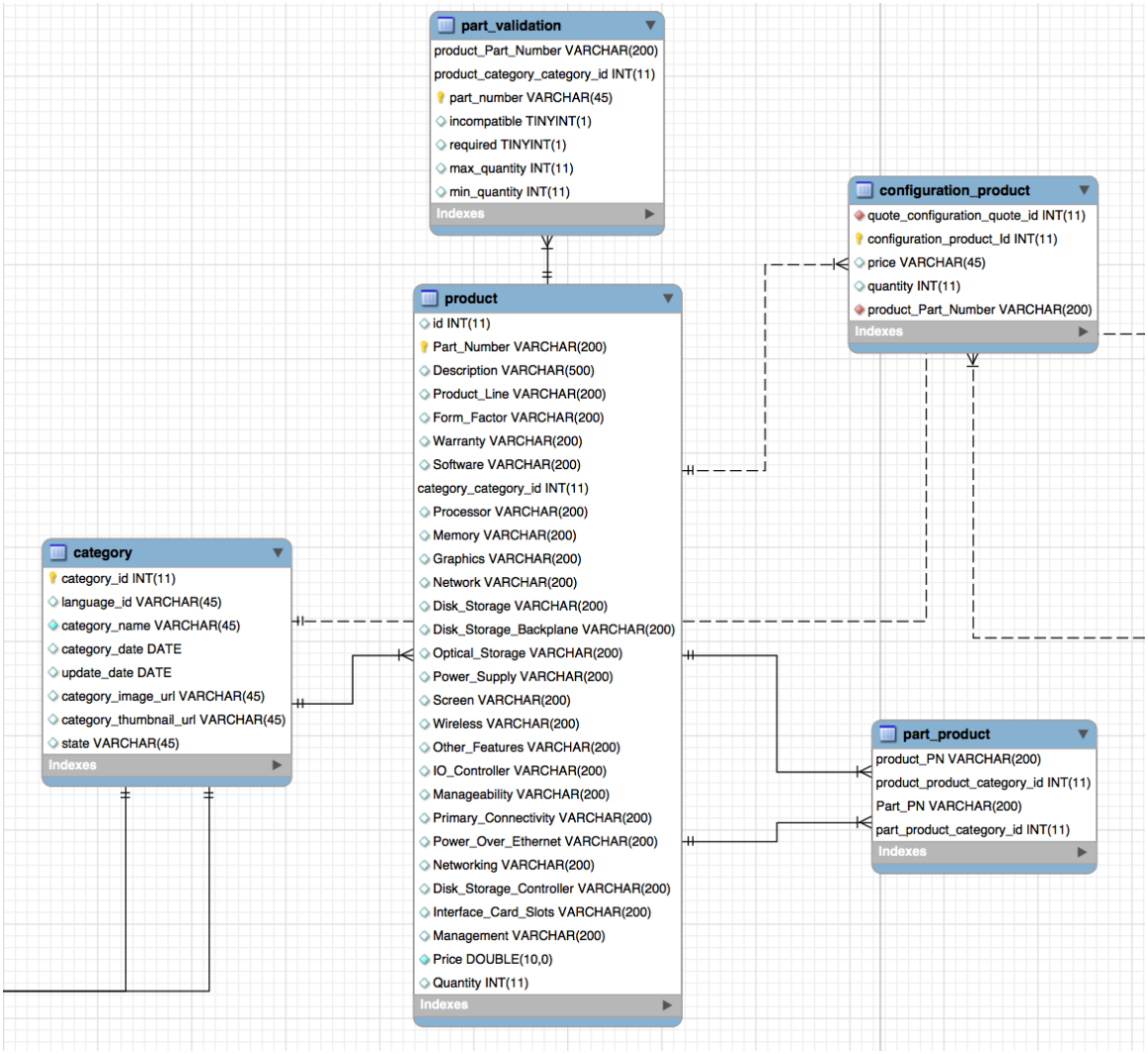

Figure 3.4. Product Table Relationships

3.1.5. PART_PRODUCT Table

PART_PRODUCT table is used to implement Product Self-Many to Many relationships. Every part is a product, and a product can have different parts, which means that a part can have other parts. See Table 3.5 and Figure 3.5 for details.

Table 3.5. PART_PRODUCT Table

PART_PRODUCT			
PRODUCT_PN	VARCHAR(200)	NOT NULL	FK
PRODUCT_PRODUCT_CATEGORY_ID	INT(11)	NOT NULL	FK
PART_PN	VARCHAR(200)	NOT NULL	FK
PART_PRODUCT_CATEGORY_ID	INT(11)	NOT NULL	FK

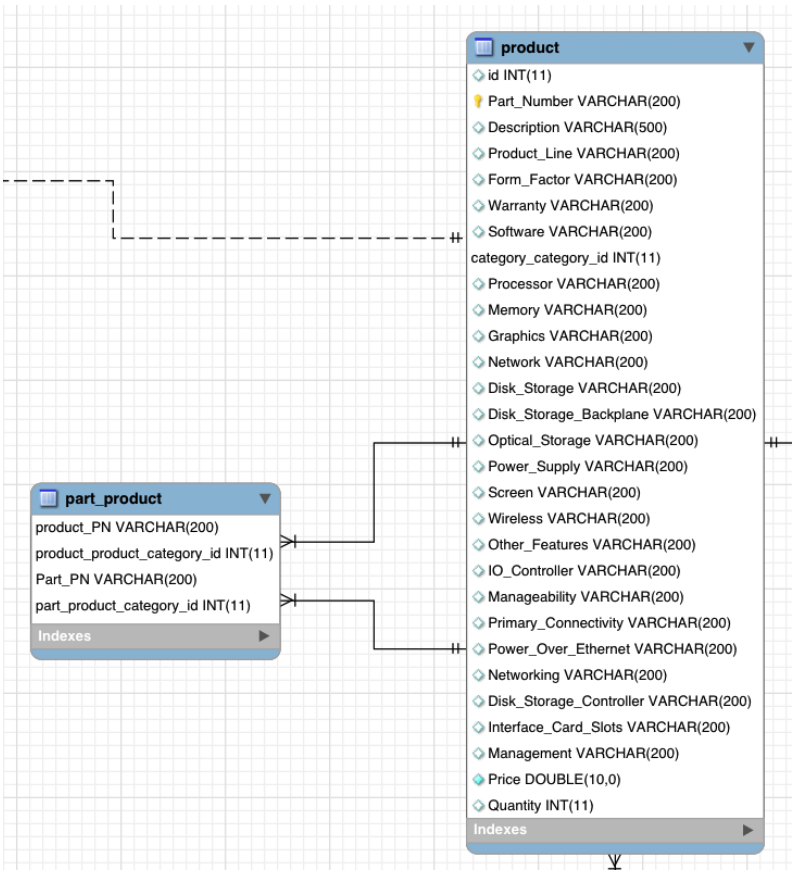

Figure 3.5. Self Many To Many Product Relationship

3.1.6. PART_VALIDATION Table

PART_VALIDATION Table contains all the validations for products, which describe the maximum and minimum instances of a product that can be used in a configuration. The PRODUCT_PART_NUMBER field describes the main product (server, laptop, storage, etc.) we are configuring; the PART_NUMBER field describes the part we want to add to our configuration; the fields INCOMPATIBLE and REQUIRED show the compatibility of a part to a product in our configuration. See Table 3.6 for details.

Table 3.6. PART_VALIDATION Table

PART_VALIDATION			
PRODUCT_PART_NUMBER	VARCHAR(200)	NOT NULL	PK
PRODUCT_CATEGORY_CATEGORY_ID	INT(11)	NOT NULL	FK
PART_NUMBER	VARCHAR(45)	NOT NULL	FK
INCOMPATIBLE	TINYINT(1)	NULL DEFAULT NULL	
REQUIRED	TINYINT(1)	NULL DEFAULT NULL	
MAX_QUANTITY	INT(11)	NULL DEFAULT NULL	
MIN _QUANTITY	INT(11)	NULL DEFAULT NULL	

3.1.7. QUOTE_CONFIGURATION Table

QUOTE_CONFIGURATION Table contains the quotes for different product configurations. It has a relationship with CONFIGURATION_PRODUCT table and a many-to-one relationship with CUSTOMER table. See Table 3.7 for details.

Table 3.7. QUOTE_CONFIGURATION Table

CONFIGURATION_PRODUCT			
QUOTE_CONFIGURATION_QUOTE_ID	INT(11)	NOT NULL AUTO_INCREMENT	FK
CONFIGURATION_PRODUCT_ID	INT(11)	NULL DEFAULT NULL	PK
PRICE	VARCHAR(45)	NULL DEFAULT NULL	
QUANTITY	INT(11)	NULL DEFAULT NULL	
PRODUCT_PART_NUMBER	VARCHAR(200)	NOT NULL	FK

3.1.8. CONFIGURATION_PRODUCT Table

CONFIGURATION_PRODUCT Table contains all the parts of a product configuration. See Table 3.8 for details.

Table 3.8. CONFIGURATION_PRODUCT Table

QUOTE_CONFIGURATION			
QUOTE_ID	INT(11)	NOT NULL AUTO_INCREMENT	PK
CUSTOMER_CUSTOMER_ID	VARCHAR(45)	NULL DEFAULT NULL	FK
QUOTE_NAME	VARCHAR(45)	NULL DEFAULT NULL	
CREATION_DATE	DATETIME	NULL DEFAULT NULL	
UPDATE_DATE	DATETIME	NULL DEFAULT NULL	
PRICE_TOTAL	VARCHAR(45)	NULL DEFAULT NULL	
BASE_PRODUCT	VARCHAR(45)	NOT NULL	

3.1.9. CUSTOMER Table

CUSTOMER Table has a description of an external user of the system. It also has a one-to-one relationship with the Users table. Every customer is a User. See Table 3.9 for details.

Table 3.9. CUSTOMER Table

CUSTOMER			
CUSTOMER_ID	VARCHAR(45)	NOT NULL	FK
CUSTOMER_GENDER	VARCHAR(45)	NULL DEFAULT NULL	
CUSTOMER_FIRSTNAME	VARCHAR(45)	NULL DEFAULT NULL	

CUSTOMER_LASTNAME	VARCHAR(45)	NULL DEFAULT NULL	
CUSTOMER_DOB	DATE	NULL DEFAULT NULL	
CUSTOMER_EMAIL_ADDRESS	VARCHAR(45)	NULL DEFAULT NULL	
CUSTOMER_HOME_ADDRESS	VARCHAR(45)	NULL DEFAULT NULL	
CUSTOMER_TELEPHONE	VARCHAR(45)	NULL DEFAULT NULL	
CUSTOMER_PASSWORD	VARCHAR(45)	NULL DEFAULT NULL	
CUSTOMER_SHIPPING_ADDRESS	VARCHAR(45)	NULL DEFAULT NULL	

3.1.10. USERS Table

USERS Table has the user name, password, and activation status of a user, and it also has a relationship with the table USER_ROLES. See Table 3.10 for details.

Table 3.10. USERS Table

USERS

USERNAME	VARCHAR(45)	NOT NULL AUTO_INCREMENT	PK
PASSWORD	VARCHAR(45)	NOT NULL	
ENABLED	TINYINT(4)	NOT NULL DEFAULT '1'	

3.1.11. USER_ROLES Table

USER_ROLES Table describes the type of users of the system: USER and ADMIN. See Table 3.11 for details.

Table 3.11. USER_ROLES Table

USER_ROLES			
USER_ROLE_ID	INT(11)	NOT NULL	PK
USERNAME	VARCHAR(45)	NOT NULL	FK
ROLE	VARCHAR(45)	NOT NULL	

4. PROJECT IMPLEMENTATION

EpicConfigurator is an application that can be accessed through a web browser. The preferred browser for this application is Google Chrome, due to its speed in loading content and opening tabs, as well as its security, since chrome is always updated and has less bugs than other web browsers. Additionally, Chrome is platform independent in comparison with Internet Explorer (IE) which only works on Windows environments. The application is also compatible with Mozilla Firefox, IE, and Safari. All the interfaces were developed using JSPs (Java Server Pages) and JQuery to provide support for AJAX invocations. To support the responsiveness of the application in multiple devices, Bootstrap framework was used. The back-end of the application was developed with Java programming language.

4.1. Administration Profile

The Administration Profile consists of a set of modules for different administration tasks. To access the Administration Page, the user must be logged in the system with ADMIN permissions.

4.1.1. Admin Login Page

This page validates the user permissions to access Admin management modules. The user can access the Admin Login Page by adding admin.html to the url: *http://localhost:8080/epic/jsp/admin.html.* See Figure 4.1 for details.

Figure 4.1. Admin Login Page

4.1.2. Admin Error Login Page

If the user enters a wrong username or password, he or she will receive a message requiring the correct username and password be entered. See Figure 4.2 for details.

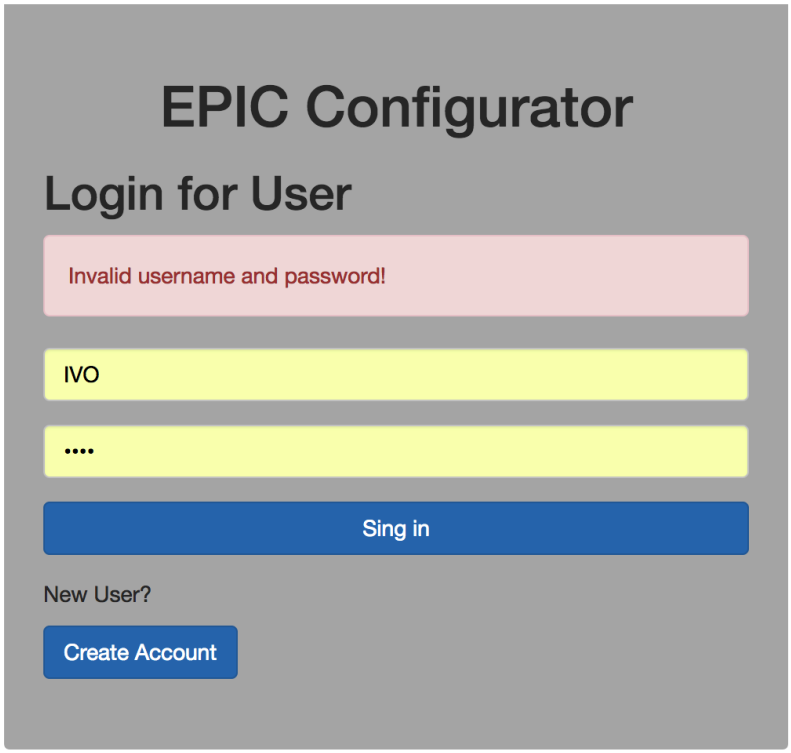

Figure 4.2. Fail Admin Login Page

4.1.3. Admin Page

The Admin Page has a single-access layout that allows access to Product, User, Price and Rule management pages. The content of each section is asynchronously reloaded, this means only the content of each selected option is refreshed and not the entire screen, which is achieved by using Ajax. See Figure 4.3 for details.

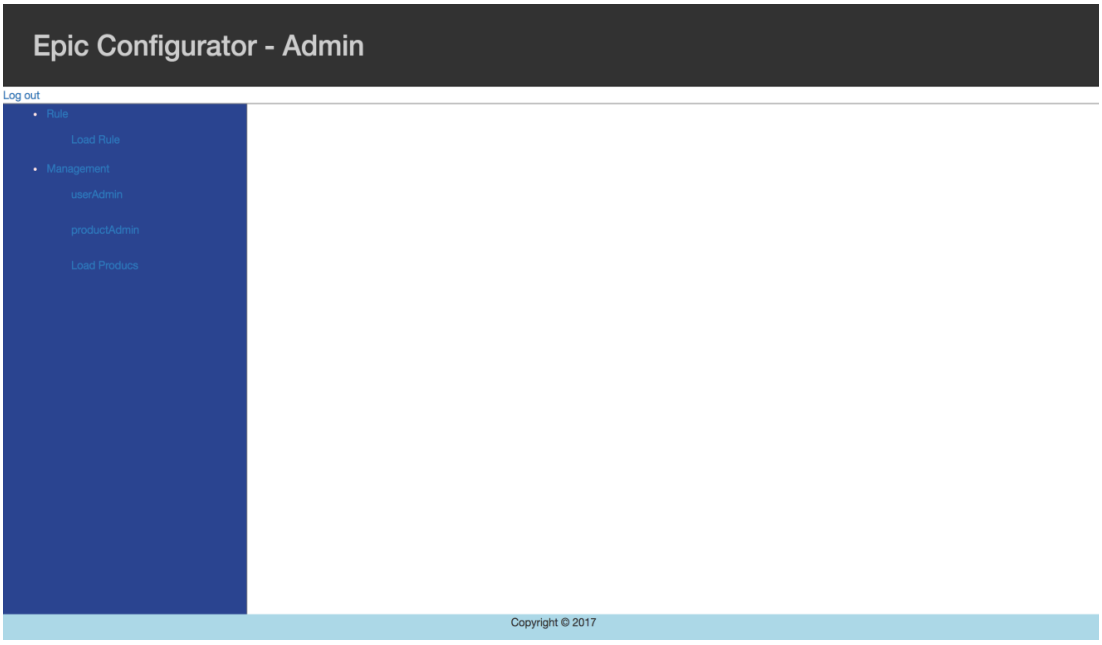

Figure 4.3. Admin Page

4.1.4. Product Management Page

The Product Management Page allows the system's Admin to search for different products, edit their attributes, remove them, and create new Products. By default, this option shows a list of all the products in the system. See Figure 4.4 for details.

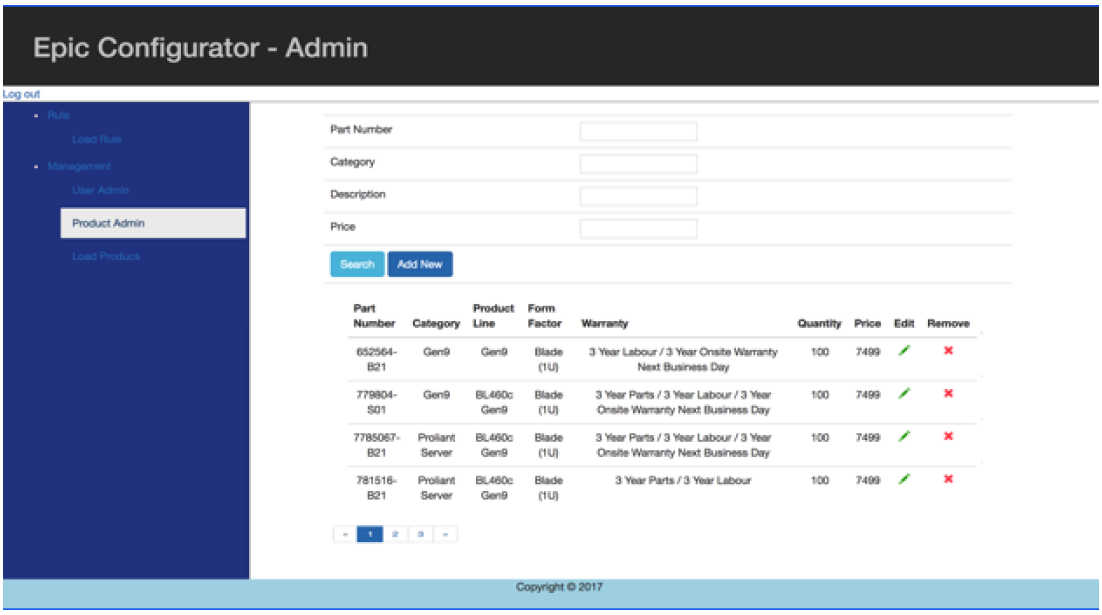

Figure 4.4. Product Management Page

4.1.4.1. Search Product Page. The Admin user can search for one specific product by entering either a part number, category, description or price. See Figures 4.5, 4.6, 4.7 and 4.8 for details.

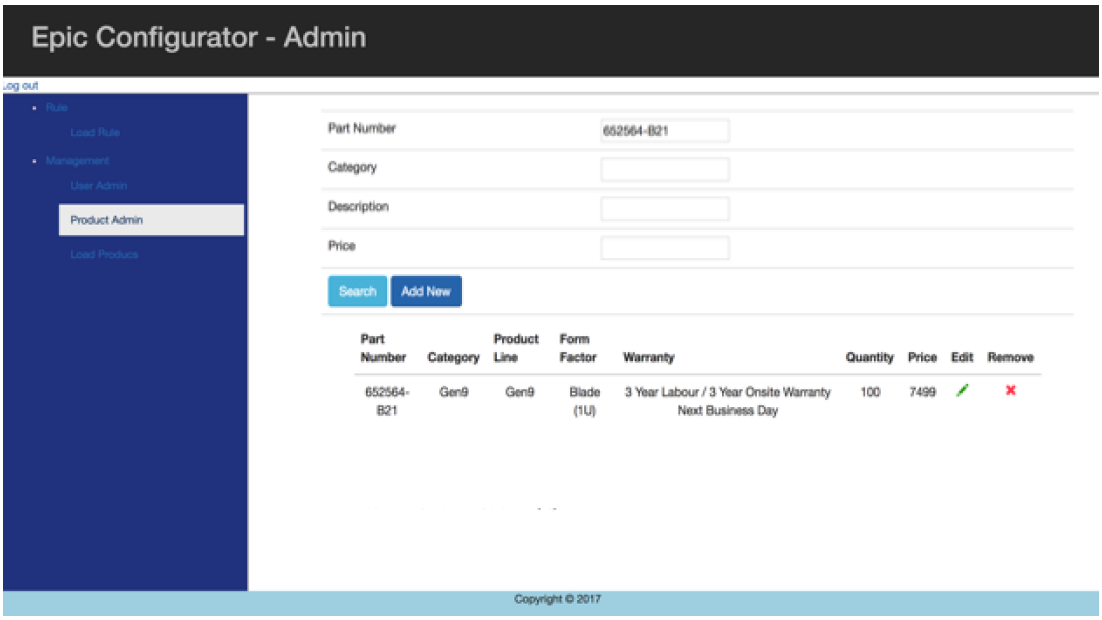

Figure 4.5. Product Search by Part_Number

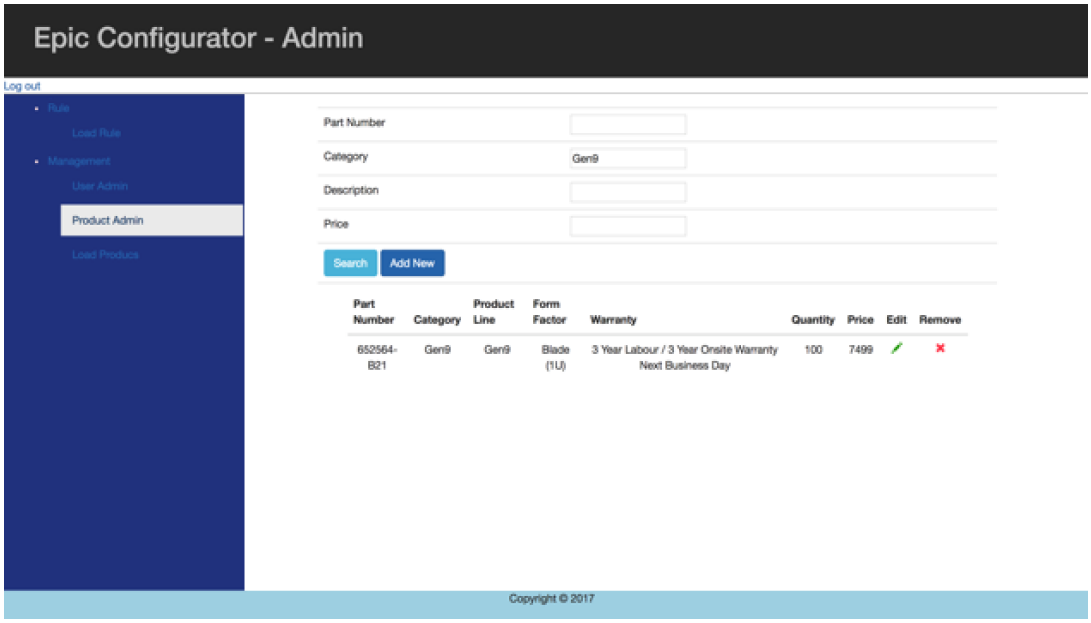

Figure 4.6. Product Search by Category

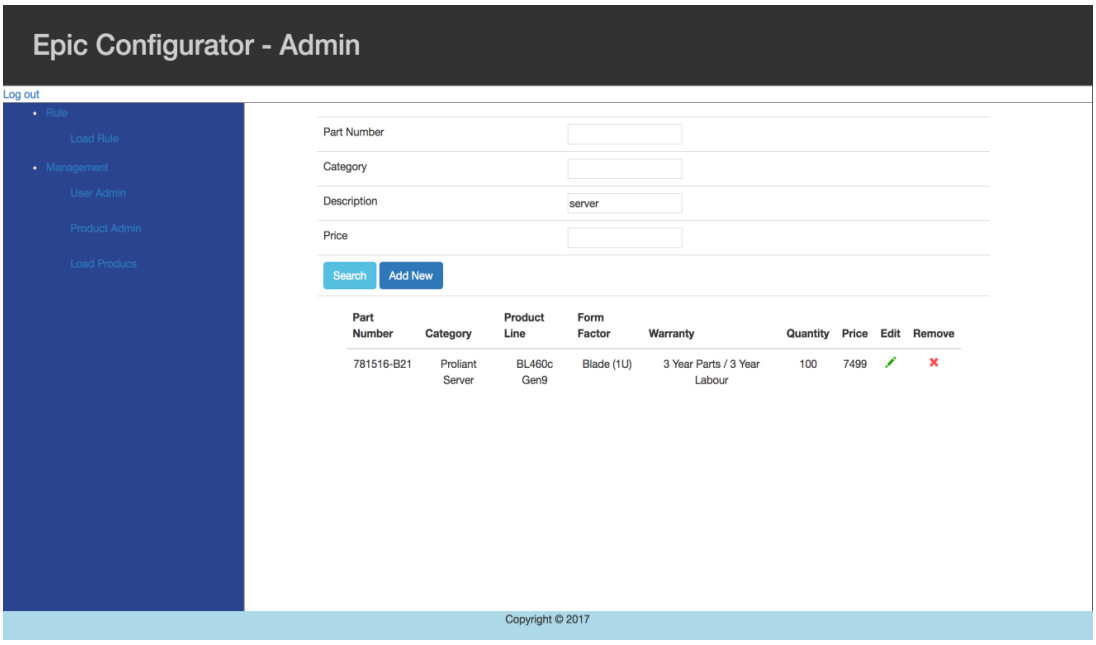

Figure 4.7. Product Search by Description

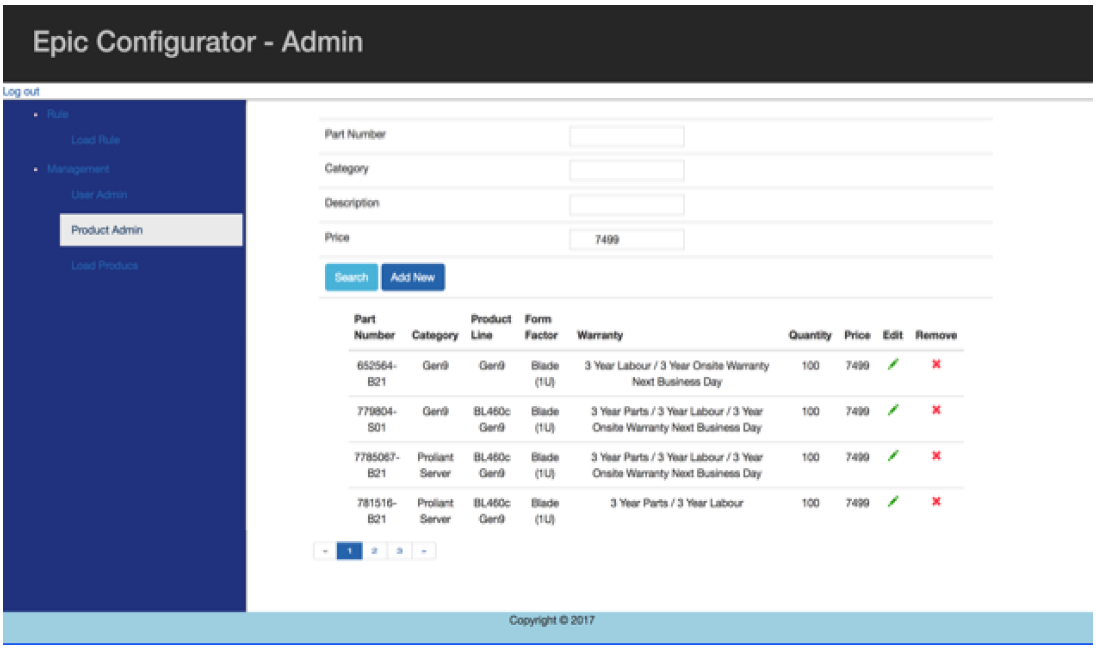

Figure 4.8. Product Search by Price

4.1.4.2. Edit Product Page. Every product shown in the resulting table can be edited. The system will display an Edit Product Page, where the Admin user can update different values of the selected product. The Admin user can then save changes or cancel the edition of that product, after which he or she will be redirected to the Product Management Page. See Figure 4.9 for details.

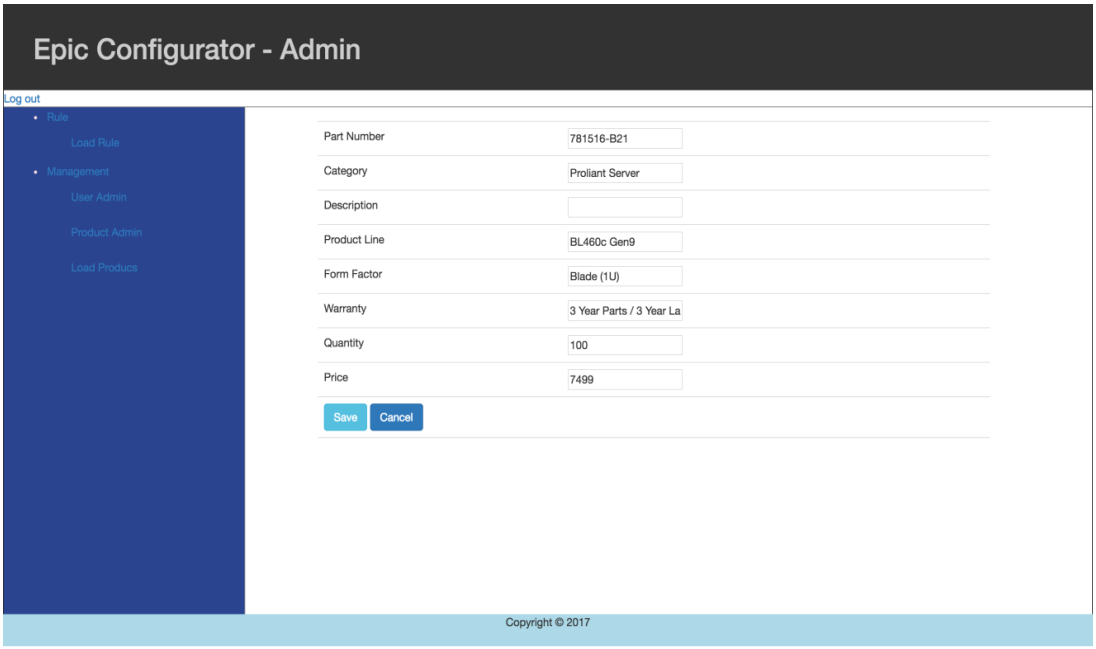

Figure 4.9. Product Edit Page

4.1.4.3. Add Product Page. The User Admin has the option to add a new product to the system. On the Product Admin page, there is an option to add a new product. After selecting this option, the system will show an “Add New Product” page, and the user can either save the product or to cancel the action. See Figure 4.10 for details.

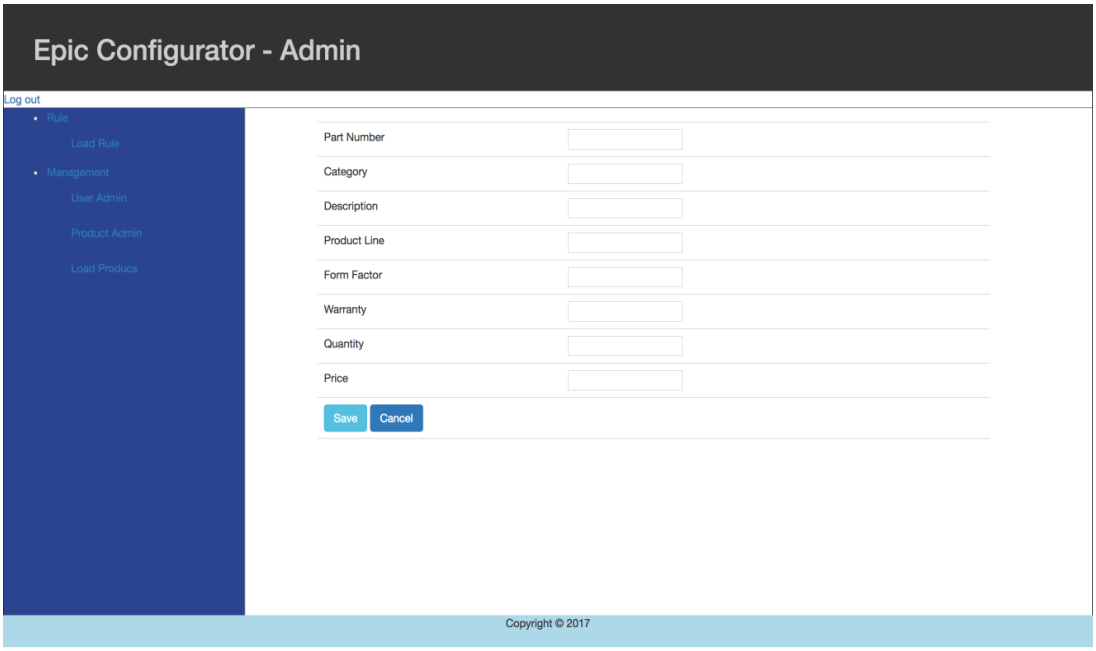

Figure 4.10. Add New Product Page

4.1.4.4. Remove Product Option. The User Admin has the option to remove a product from the system. In the Product Admin page, there is an option next to the Edit Product icon to remove a Product. After selecting this option, the system will confirm if you want to remove the product; if so, the product will be removed from the list of products. See Figure 4.11 for details.

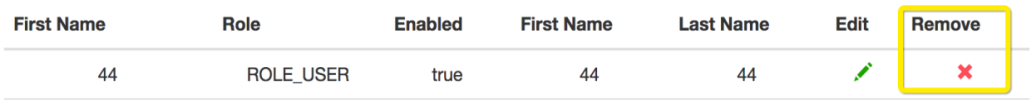

Figure 4.11. Remove Product Option

4.1.5. User Management Page

The User Management Page allows the system's Admin to search for any Users, edit their attributes, remove them, and create new Users. By default, this option shows a list of all the users in the system. See Figure 4.12 for details.

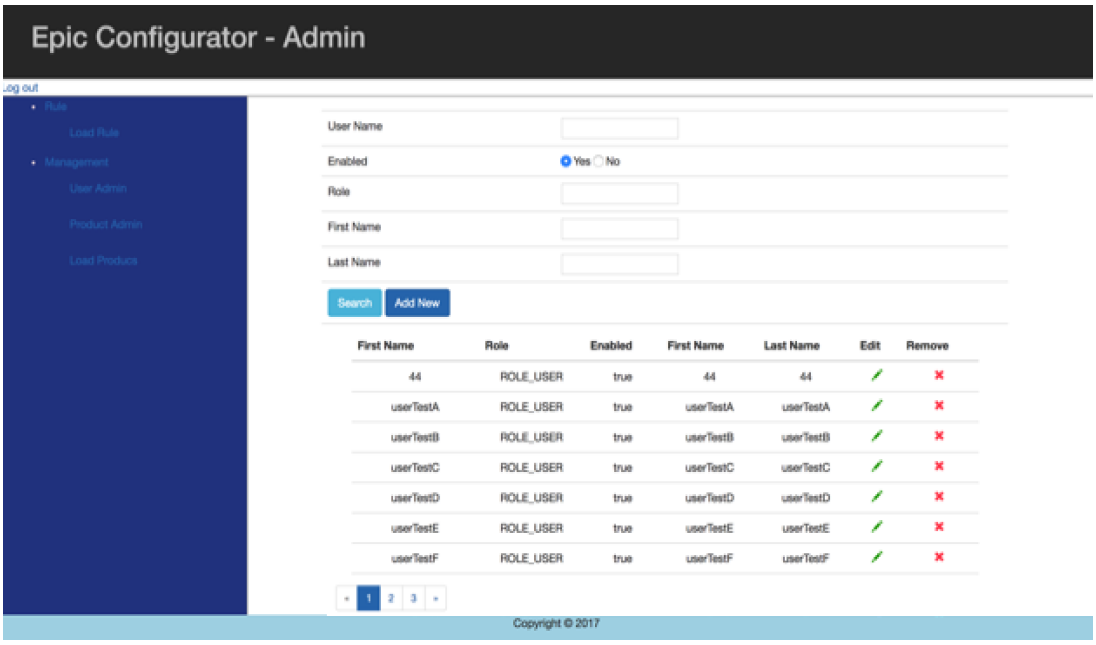

Figure 4.12. User Management Page

4.1.5.1. Search User Page. The Admin User can search for one specific User by filling out the fields User Name, Enabled/Disabled, Role, First Name or Last Name - one at a time or in combination. See Figures 4.13, 4.14, 4.15, 4.16 and 4.17 for details.

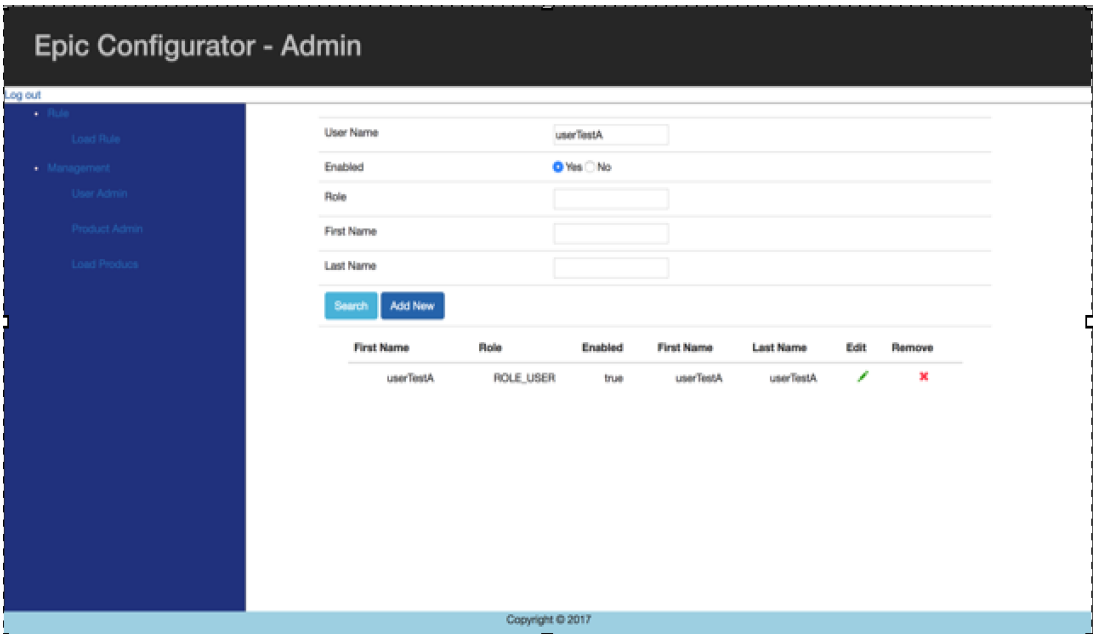

Figure 4.13. User Search by User Name

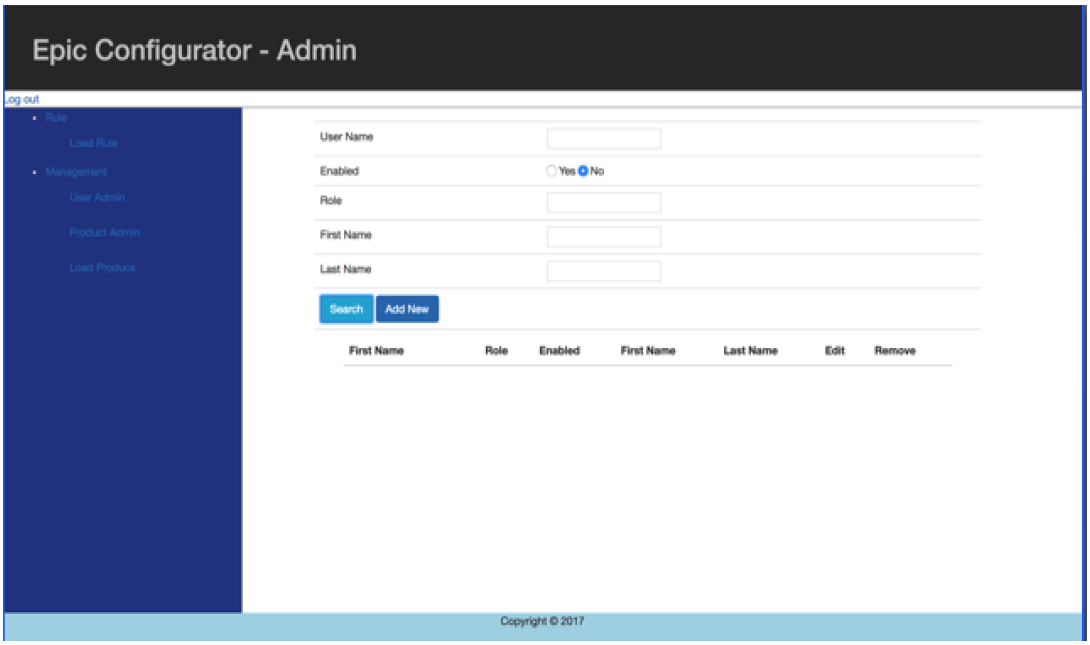

Figure 4.14. User Search by Enabled/Disabled

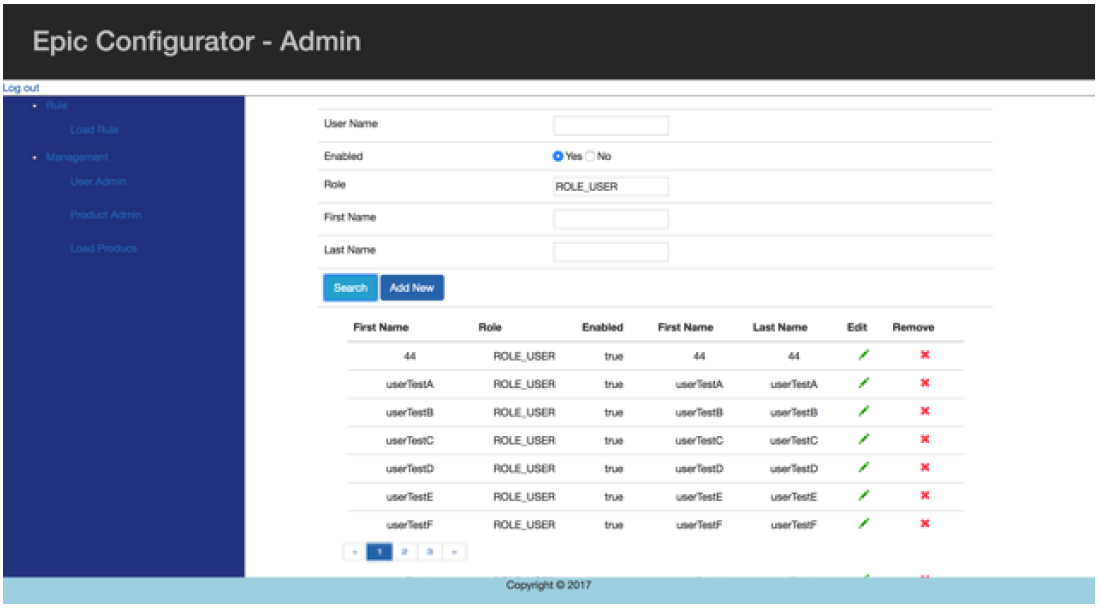

Figure 4.15. User Search by Role

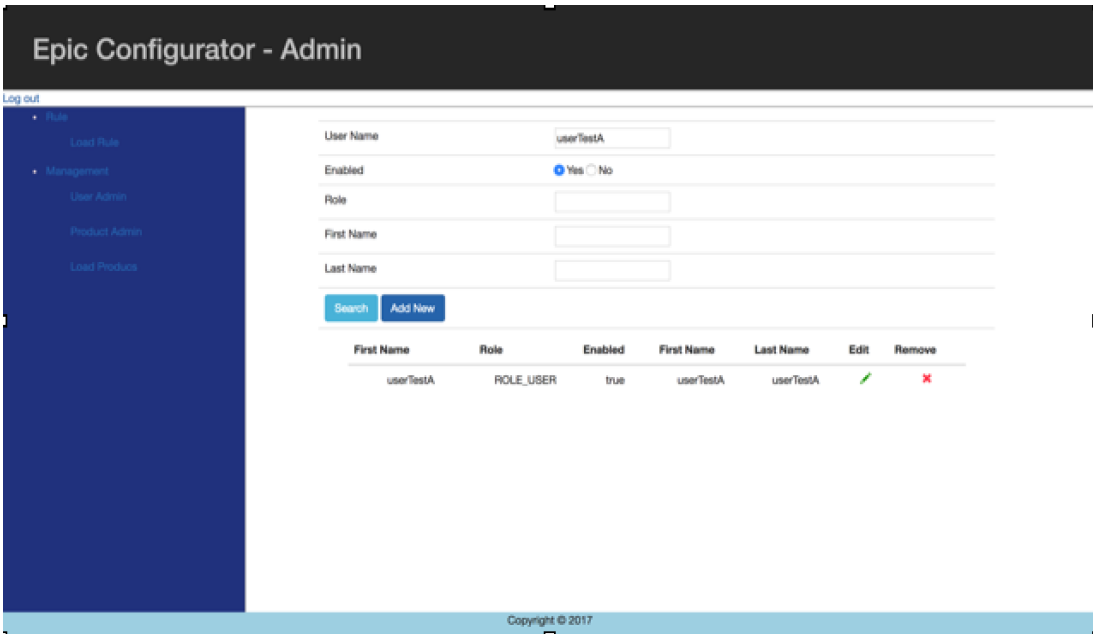

Figure 4.16. User Search by First Name

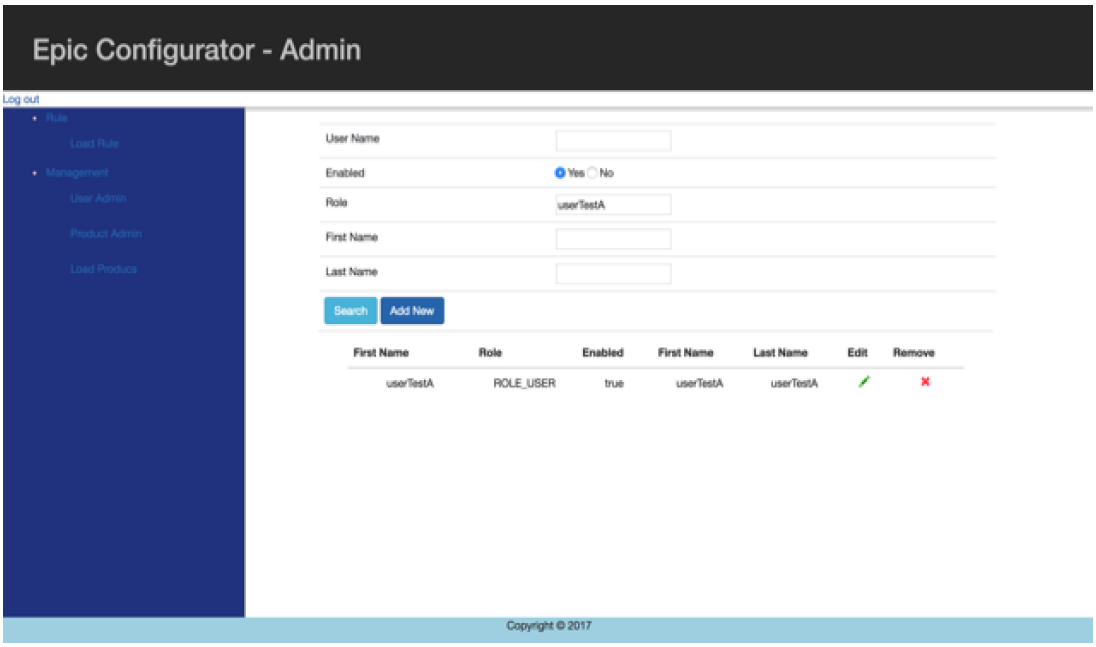

Figure 4.17. User Search by Last Name

4.1.5.2. Edit User Page. Every user in the User Result table can be edited by the Admin user. After selecting that option, the system will show an Edit Product page, where the Admin user can update different values of the selected user. The Admin user can save the changes or cancel the update of that product, after which the user will be redirected to the User Management page. See Figure 4.18 for details.

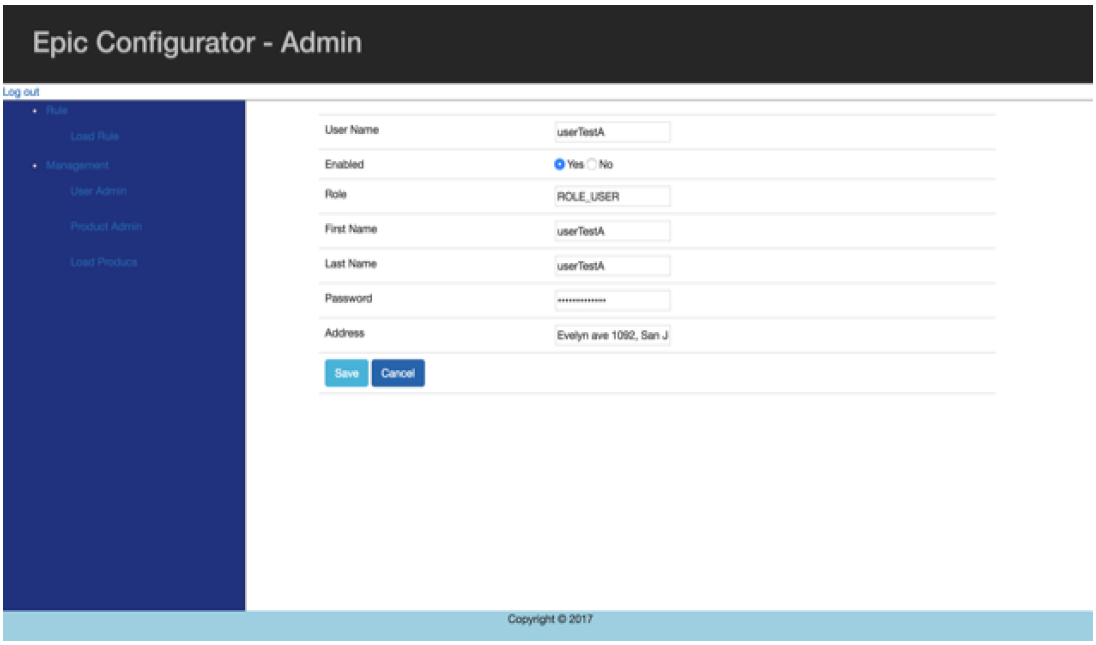

Figure 4.18. User Edit Page

4.1.5.3. Add User Page. The User Admin has the option to add a new user to the system. On the User Admin page is an option to “Add New User”. After clicking this option, the system will show the “Add a New User” page, and the user can either save the user or to cancel the action. See Figure 4.19 forUser Admin has the option to add a new user to the system. On the User Admin page is an option to “Add New User”. After clicking this option, the system will show the “Add a New User” page, and the user can either save the user or to cancel the action. See Figure 4.19 for details.

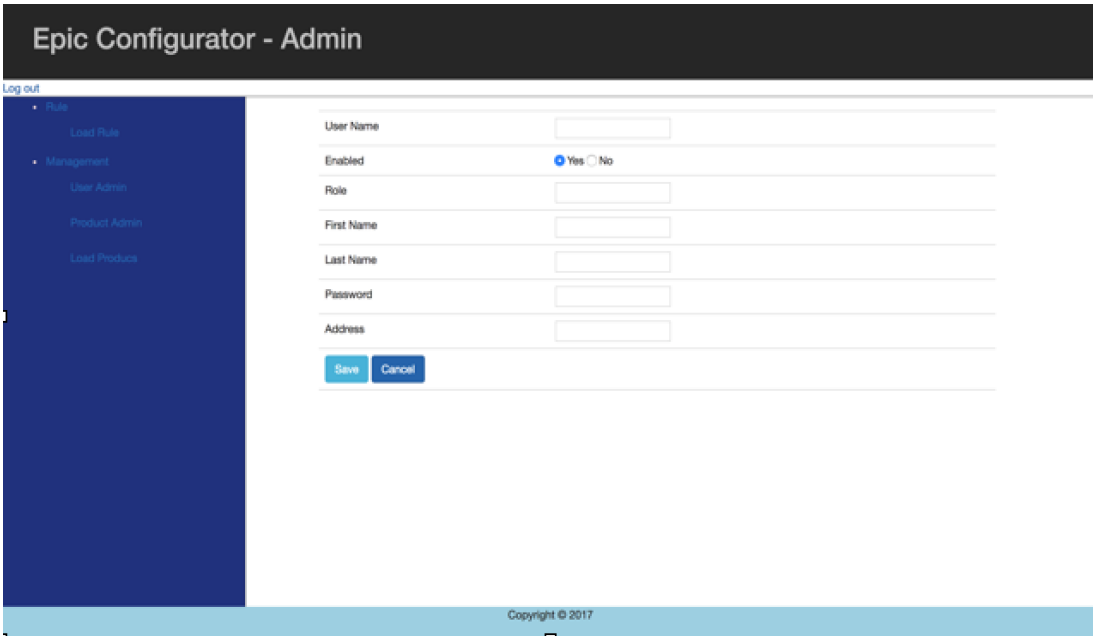

Figure 4.19. Add New User Page

4.1.5.4. Remove User Option. The User Admin has the option to remove a user from the system. In the “User Admin page”, there is an option to “Remove a user” next to the “Edit User icon.” After selecting this option, the system will confirm if you want to remove the user; if the admin user accepts, the user will be removed from the list of users. See Figure 4.20 for details.

First Name	Role	Enabled	First Name	Last Name	Edit	Remove
44	ROLE_USER	true	44	44		

Figure 4.20. Remove User Option

4.1.6. Load Products Page

The Load Products Page allows the Admin User to load products in bulk using a .csv file. When Admin selects “Choose File”, a window will pop up and the user will be able to load a file, which should have as its name productload.csv. If the file does not have the correct name or is corrupt or malformed, the system will show an error message. See Figure 4.21 for details.

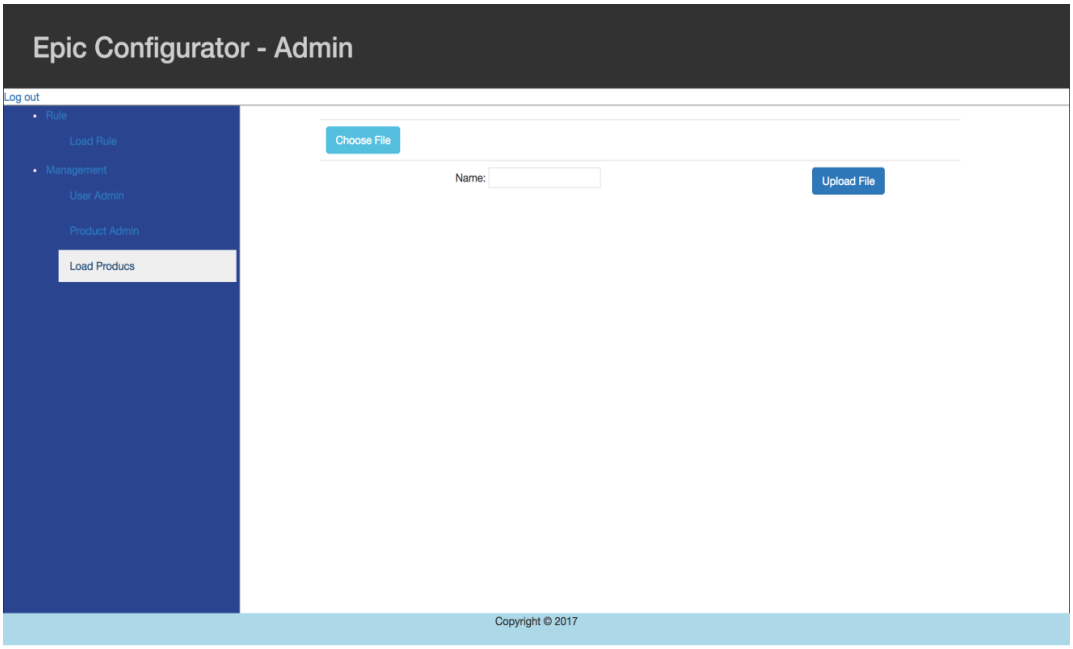

Figure 4.21. Load Product Page

4.1.7. Rule Page

The Rule Page allows the Admin User to load Rules for Product Validations in bulk using a .csv file. When Admin selects “Choose File”, a window will pop up and the user will be able to select the file to be loaded, which should

have as its name productload.csv. If the file doesn't have the correct name or is corrupt or malformed, the system will show an error message. See Figure 4.22 for details.

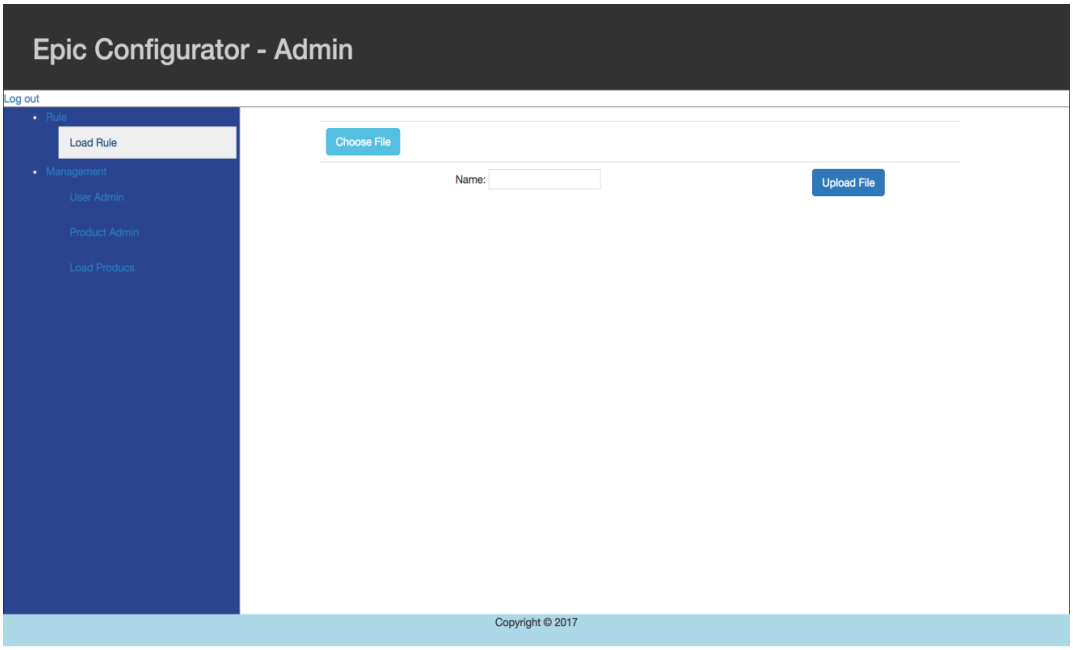

Figure 4.22. Load Rule Page

4.1.8. Help Page

EpicConfigurator provides a Help page to provide clarifications on how to load product rules and how to load new products in the system. This page is located under the Resources menu and it displays samples on how to load new rules or load products. See Figure 4.23 for details.

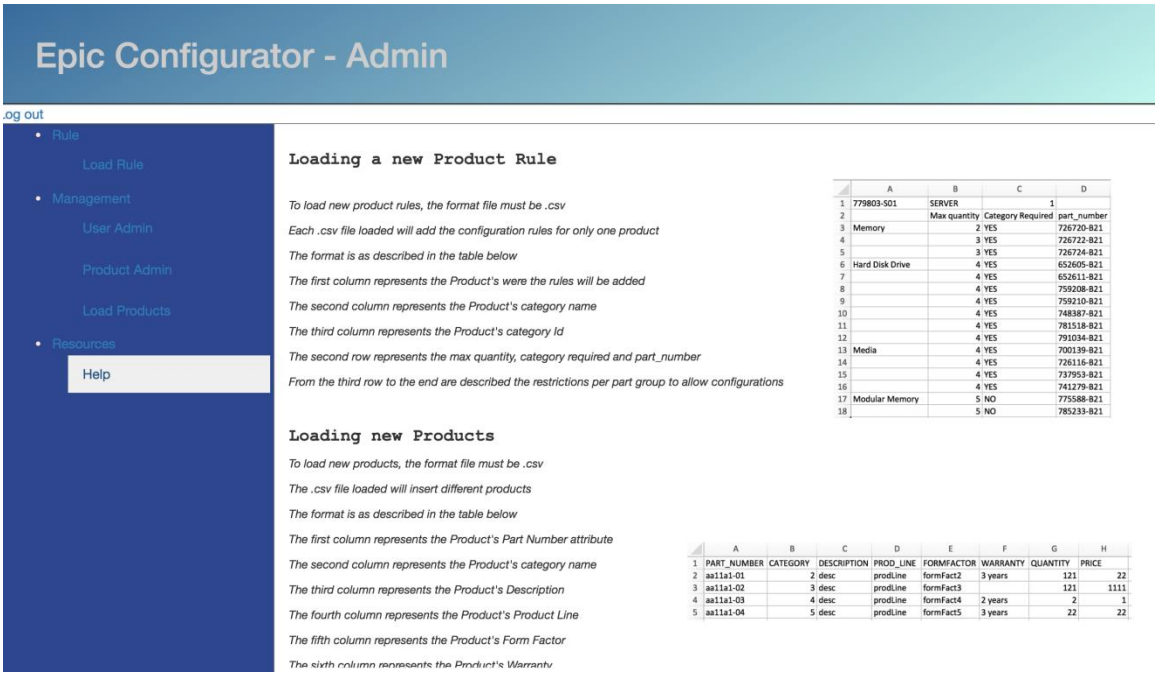

Figure 4.23. Help Page

4.2. User Profile

Under this profile, the user can access all modules for non-Admin user interaction with the system that are available. In this profile, the user will be able to create, edit, and remove product configurations.

4.2.1. Welcome Page

The Welcome Page is the default page when the system is deployed, and is the starting point for accessing all other functionalities a non-Admin user can access. See Figure 4.24 for details.

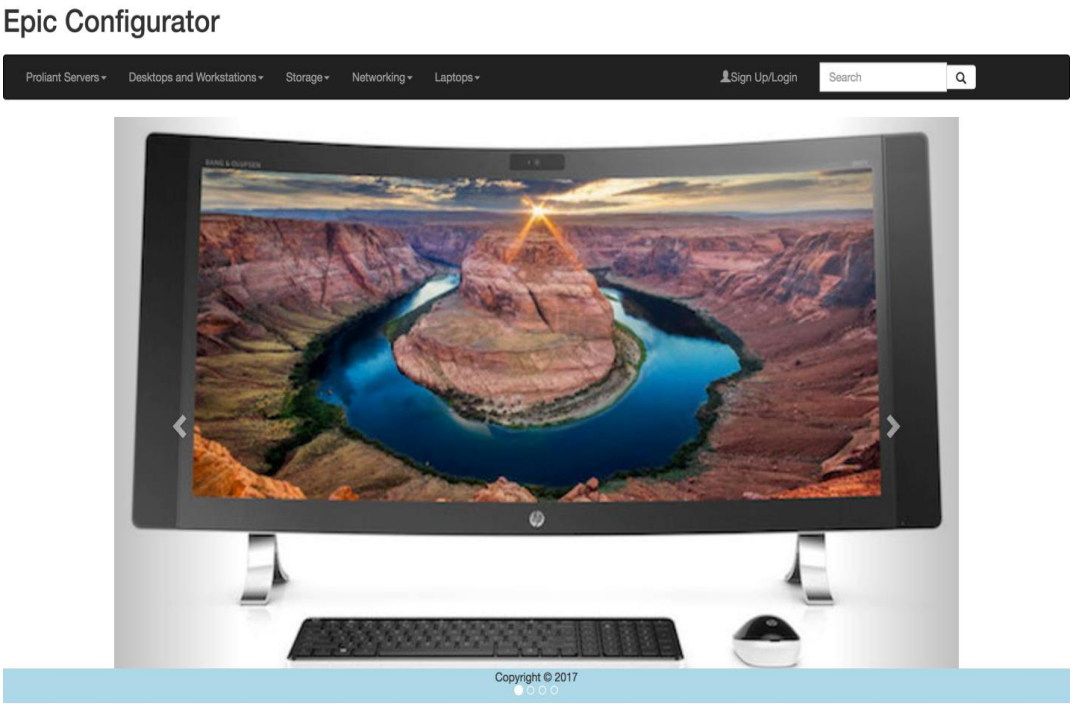

Figure 4.24. Welcome Page

4.2.2. User Login Page

An existing user can create and edit configurations only if he or she is logged into the system. To do so, the user can log in via the “Sign Up/Login” button in the menu header of the system, which will take the user to a login page. After filling out the user name and password, the user will be redirected to the “Welcome Page” as a logged in user. The user name will be shown in the menu header of the system. See Figures 4.25 and 4.26 for details.

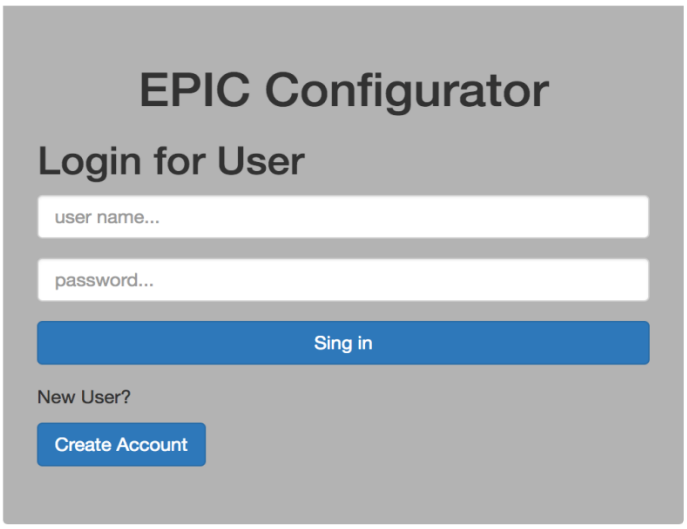

Figure 4.25. Login User Page

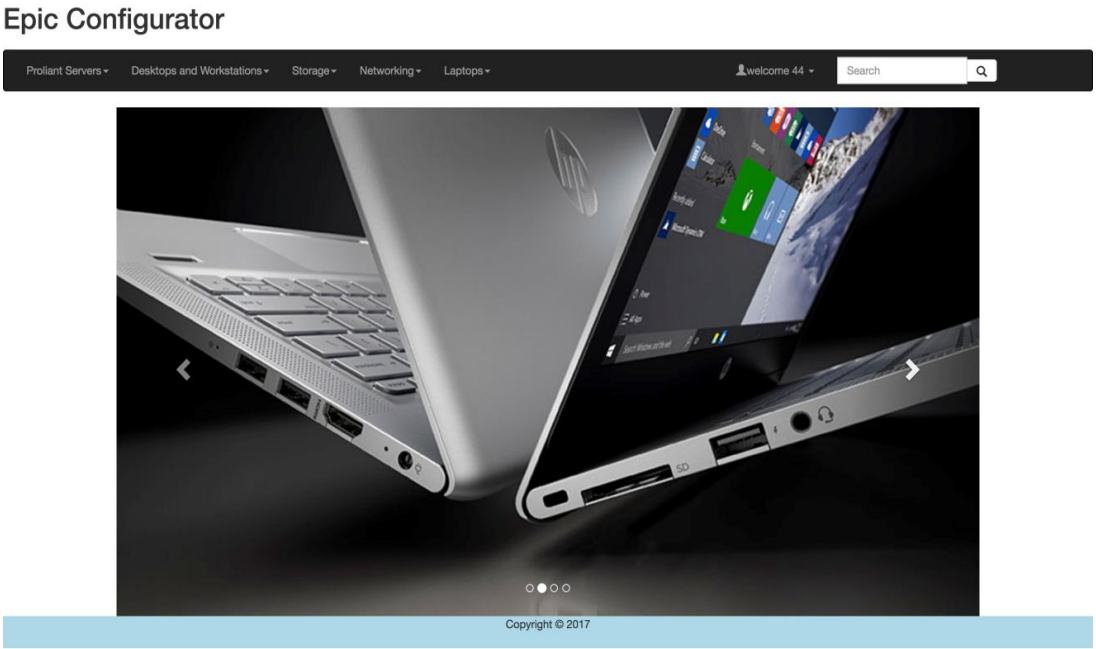

Figure 4.26. Logged User Welcome Page

If the user enters incorrect credentials, an error message will be shown. See Figure 4.27 for details.

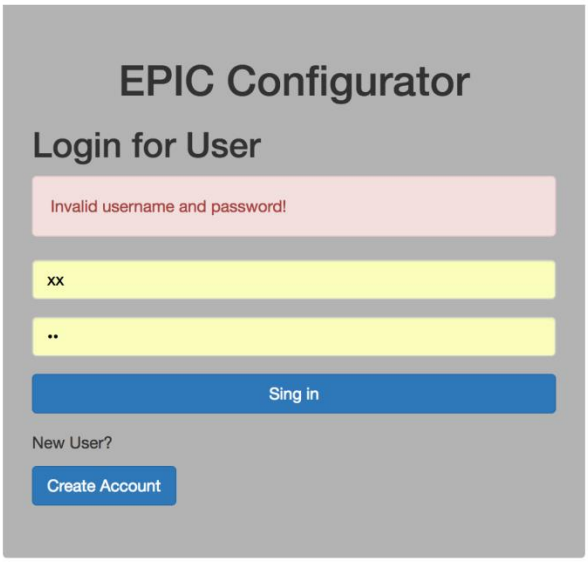

Figure 4.27. User Wrong Credentials Page

4.2.3. Create User Account Page

EpicConfigurator allows creation of a user account via the “Sign Up/Login” button and the subsequent “Create Account” option. This will take the user to the user Sign up page. See Figures 4.28 and 4.29 for details.

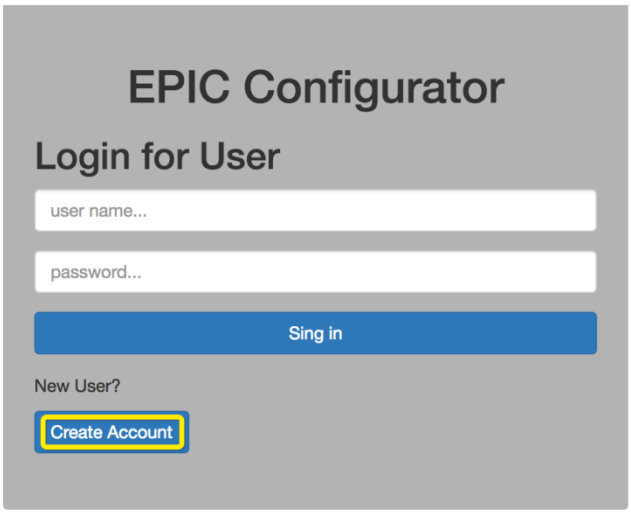

Figure 4.28. Create Account Option

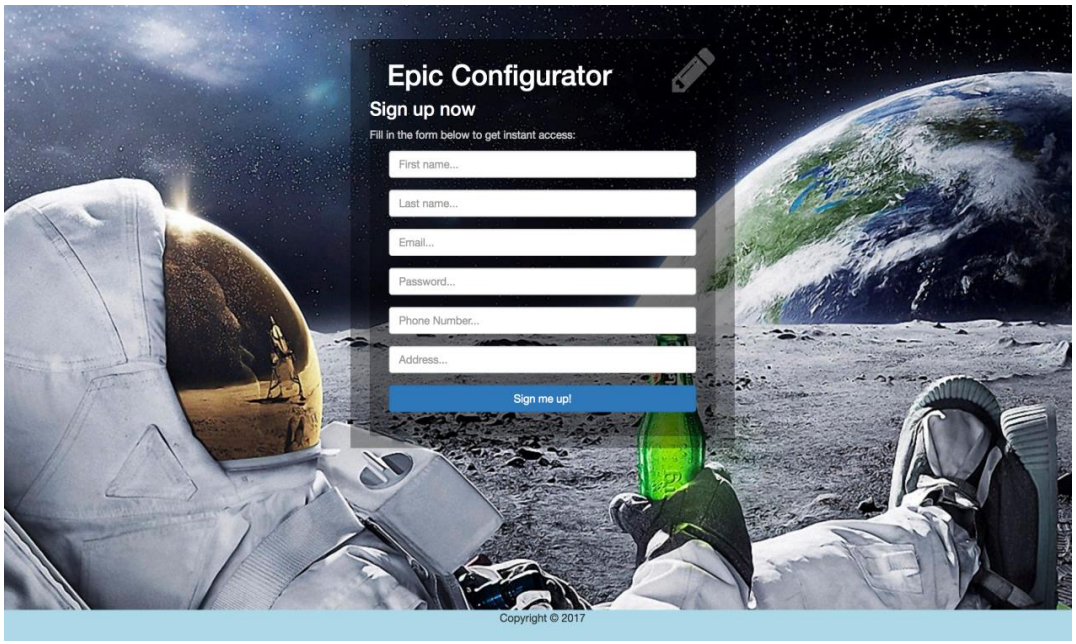

Figure 4.29. Sign Up Page

4.2.4. Saved Configuration Page

After a user is successfully logged in, he or she can access the saved configurations page containing a list of all the configurations saved by the user. Each configuration can be removed or edited. This list has a pagination that will show a different list of configurations depending on the selected page. See Figures 4.30 and 4.31 for details.

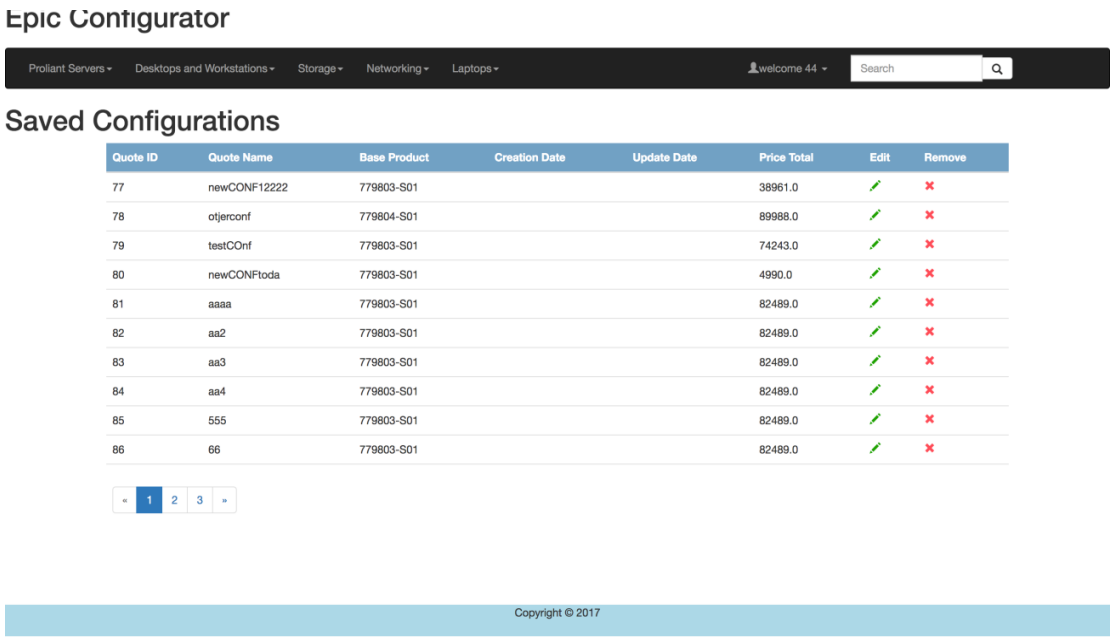

Figure 4.30. Saved Configurations Page

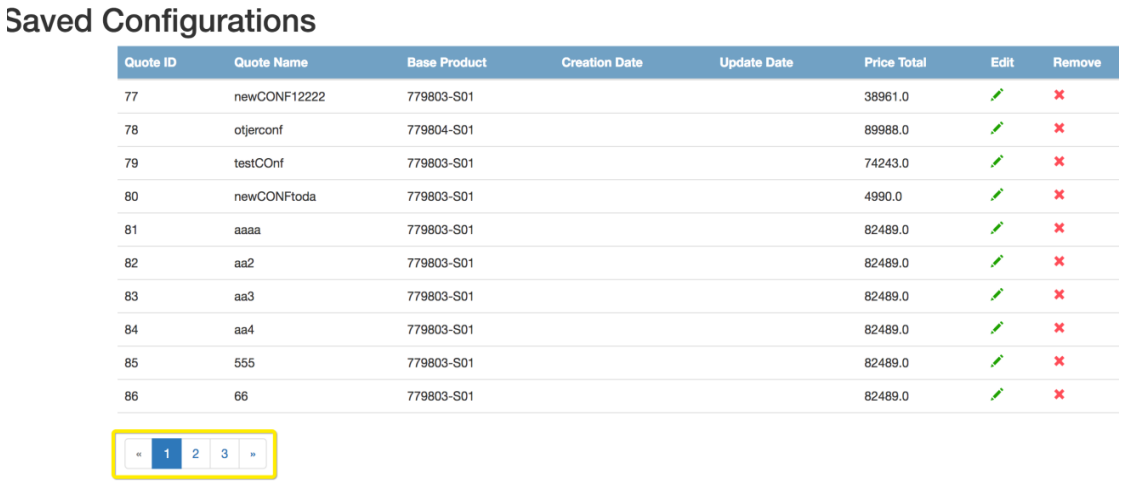

Figure 4.31. Saved Configuration Pagination Option

A logged-in user has the option of editing a specific Configuration by clicking the “Edit” icon. After selecting the “Edit” icon, the user will be taken to the configuration page, where the user can see the product being configured, the category, the number of parts added and the price of each part, along with the validations for the parts included. See Figure 4.32 for details.

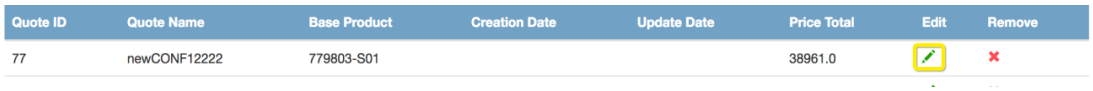

Figure 4.32. Saved Configuration Edit Option

The logged in User has the option to remove a configuration by clicking the remove option. The system will show a confirmation pop out before removing the configuration from the user saved configurations. See Figure 4.33 for details.

Figure 4.33. Saved Configuration Remove Option

The user can update a saved configuration after entering new values to the saved configuration and selecting “Update Configuration”. Afterwards, a successful update message will be shown to the user. See Figures 4.34 and 4.35 for details.

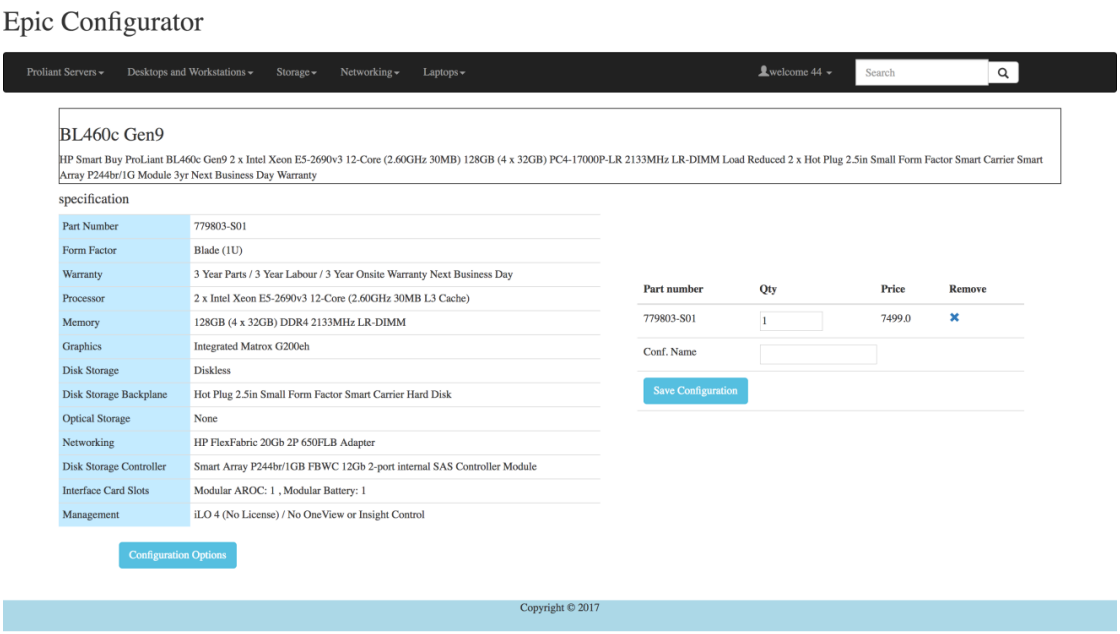

Figure 4.34. Edit Configuration Page

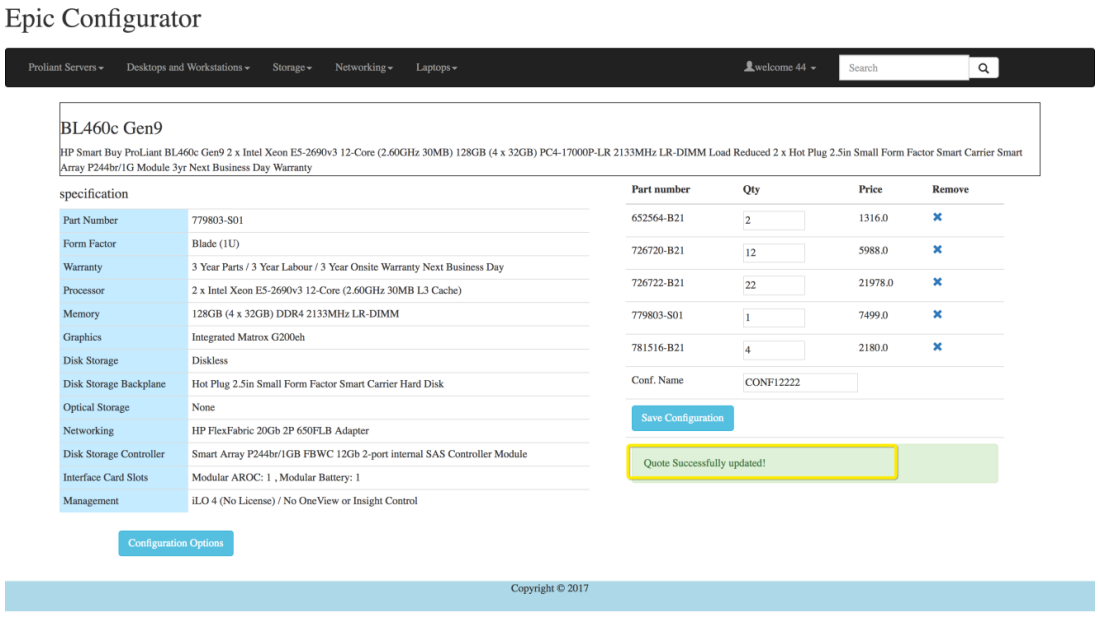

Figure 4.35. Update Configuration Message

4.2.5. New Configuration Page

The user can create a new configuration after selecting a product from the categories available in the menu header. The New Configuration Page will show up. See Figure 4.36 for details.

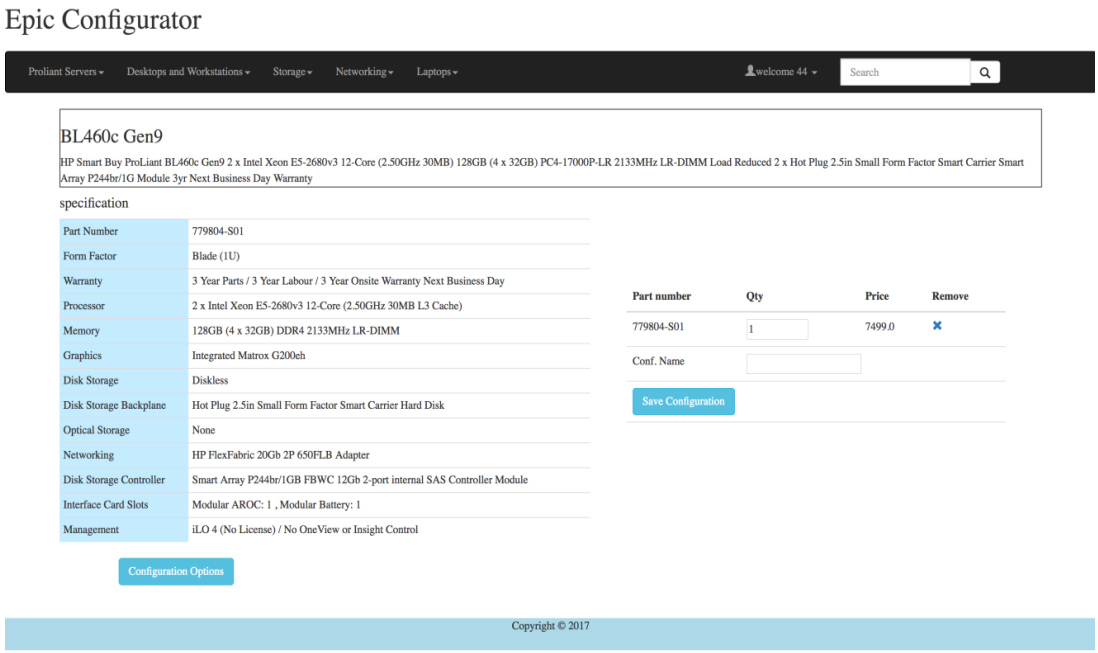

Figure 4.36. New Configuration Page

The configuration options for a product are displayed when the user selects the “Configuration Option” button. See Figure 4.37 for details.

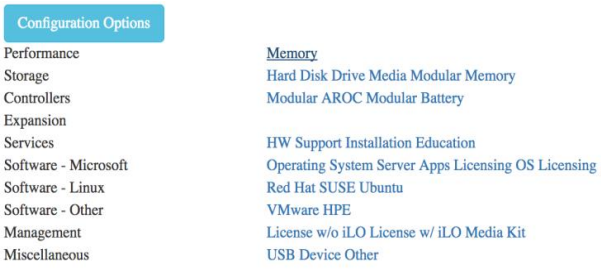

Figure 4.37. Product Configuration Options Section

To display the parts available for each configuration option, the user can click the types of configurations links for each option, after which a table with the parts belonging to that option is shown. The user can enter the quantity of every part required. Afterwards, the new added parts will be displayed in the summary section of the product configuration, along with any validation rule that could be triggered by the configuration. See Figures 4.38 and 4.39 for details.

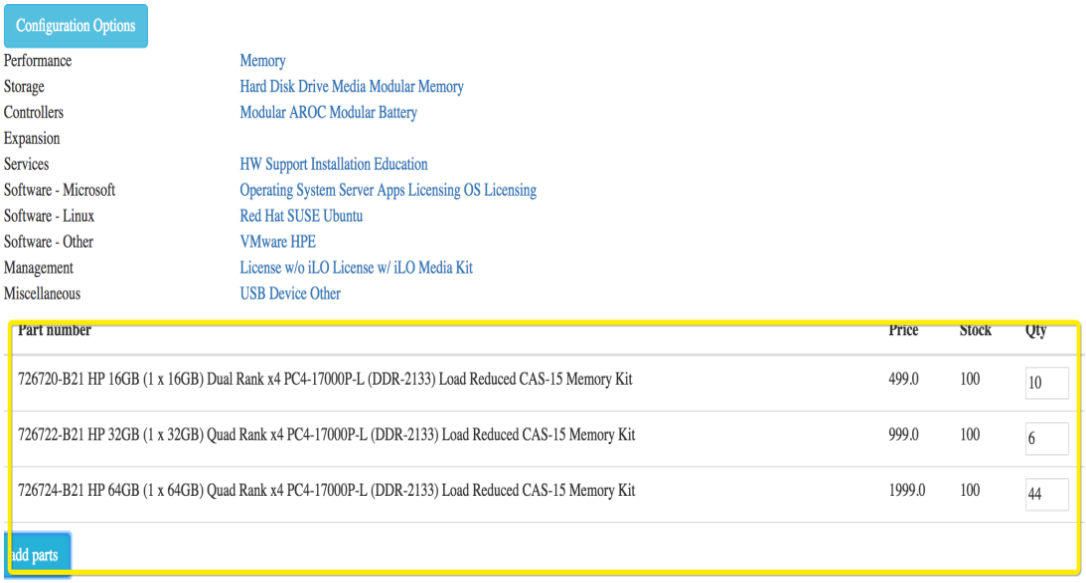

Figure 4.38. Add Parts Configuration Options Table

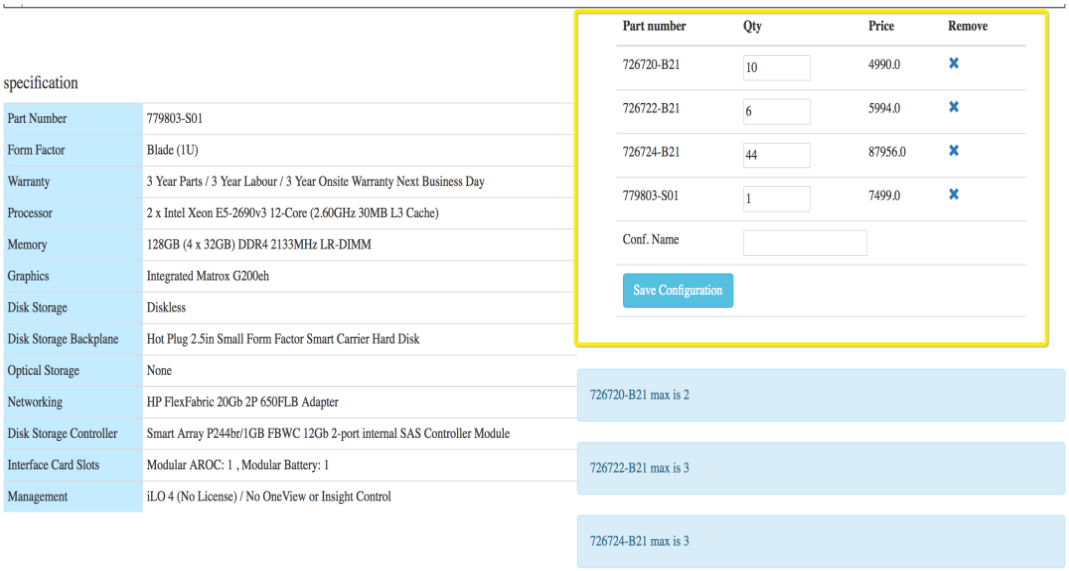

Figure 4.39. Summary Configuration

4.2.6. Category Page

Every product belongs to a category, and these categories are displayed in the header menu of the application. See Figure 4.40 for details.

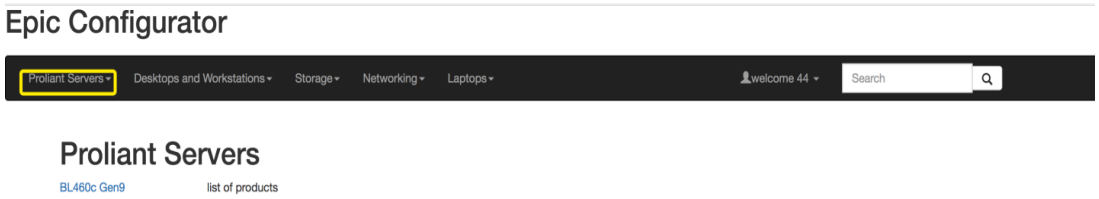

Figure 4.40. Category Page

4.2.7. Product Listing Page

In the “Product Listing page”, all the products belonging to the selected category are displayed. See Figure 4.41 for details.

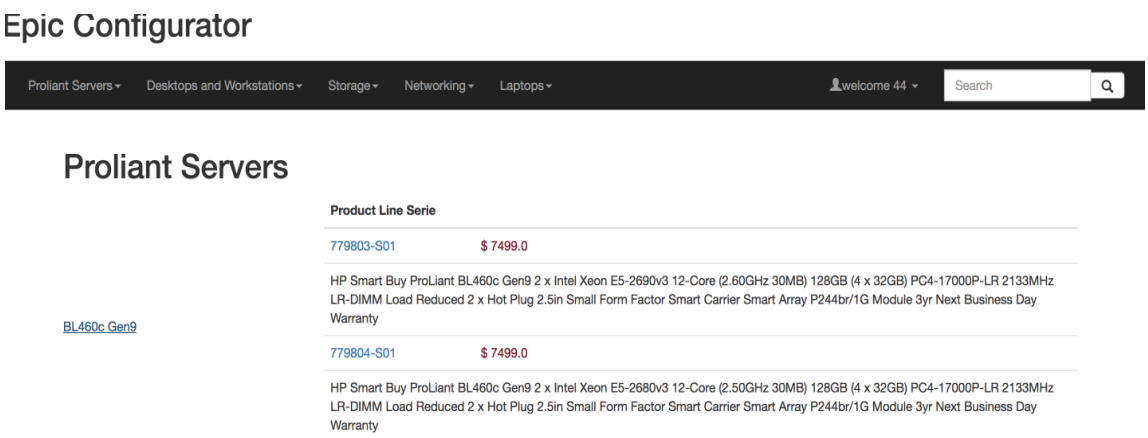

Figure 4.41. Product Listing Page

4.2.8. Product Configuration Page

The “Product page” can be accessed by clicking the product displayed in the “Product listing page” or by searching in the search box. See Figure 4.42 for details.

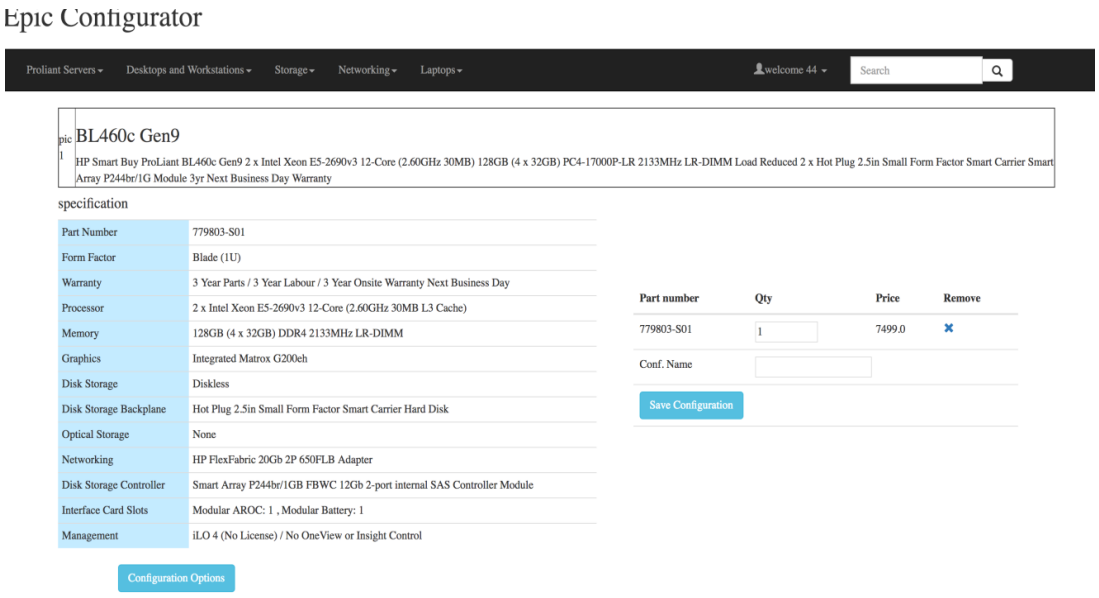

Figure 4.42. Product Page

4.2.9. Product Search Option

A product can be searched directly using the search box in the menu header. Suggested Search functionality in the application is obtained with the use of Apache Solr. See Figure 4.43 for details.

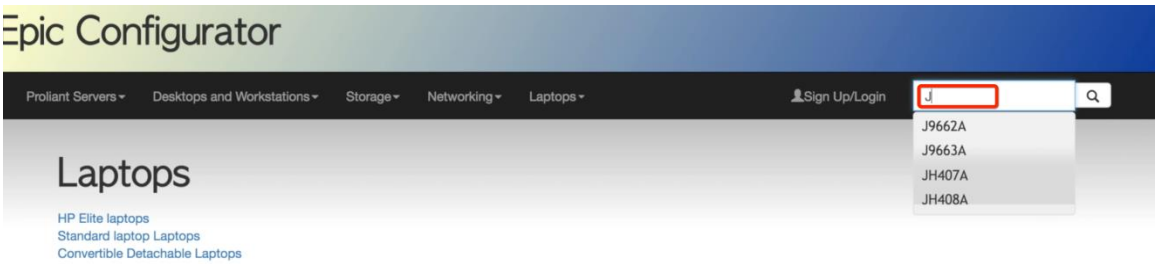

Figure 4.43. Suggested Search

5. SYSTEM IMPLEMENTATION AND DEPLOYMENT

In this chapter, I will explain the minimum requirements for deploying EpicConfigurator. The deployment depends on the type of Operating System (OS) preferred by the System Administrator. In this case, I will explain how this is done in a Linux OS, though it can be deployed in other OS's as well.

5.1. Java Installation

The JDK used for this application is JDK 1.8 and can be found in the downloads section of the official Oracle website [9]. Select the RPM distribution and download. After downloading, this distribution we run the following command from the location where this file will be installed:

```
$ chmod +x jdk-8u131-linux-x64.rpm.bin
$ ./ jdk-8u131-linux-x64.rpm
```

Accept the terms and complete the installation, the next step is to set the environment variables, and create /etc/profile/javaInstall.sh file with following contents:

```
export JAVA_HOME=/user/java/latest
export PATH=$PATH:$JAVA_HOME/bin
```

Then run the following command to execute the program:

```
$ chmod +x /etc/profile.d/javaInstall.sh
```

5.2. Tomcat Setup

EpicConfigurator is an application developed using Spring MVC Framework, which requires a servlet container. I am using Apache Tomcat Web Server version 8 to deploy the application. Tomcat requires that the JRE be installed in the host OS; this step is already covered in the Java installation in Chapter 5.1.

Tomcat binaries can be downloaded from the repository by running the following commands:

 # cd /opt/

#wget http://ftp.itu.edu.tr/Mirror/Apache/tomcat/tomcat-8/v8.0.9/bin/apache-tomcat-8.0.9.tar.gz

 # tar -xvf apache-tomcat-8.0.9.tar.gz

Now we can start Tomcat server with the following command:

 # cd /opt/apache-tomcat-8.0.9/bin

 # ./startup.sh

5.3. MySQL Installation

MySQL provides a repository for several Linux distributions that contain the latest stable MySQL release. We need to add a new MySQL repository to the system in order to proceed.

Add the new MySQL repository to the CentOS server by running this yum command:

#yum localinstall https://dev.mysql.com/get/mysql57-community-release-el7-9.noarch.rpm

You will be asked to add a new repository; type 'y' and press 'Enter' to confirm.

A new repository has been added to the system.

The new MySQL repository is available on the system now, and we are ready to install MySQL 5.7’s latest stable version from the repository. The package name is 'mysql-community-server'.

On CentOS, install 'mysql-community-server' with yum.

 # yum -y install mysql-community-server

After installing MySQL, start it and add MySQL to start at boot time automatically with the systemctl command.

For CentOS server use 'mysqld' service.

 # systemctl start mysqld

 # systemctl enable mysqld

MySQL started, it's using port 3306 for the connection.

5.4. Database Migration

After the MySQL has been installed we need to run the EpicConfiguratorDB Schema script defined in Table 3.12. This script contains only the tables used by the application.

Some data is required for Menu and Category tables, and to create an Admin User, run insert statements like the following:

INSERT INTO ‘epicConfiguratorDb’.’users’ (username,password,enabled) VALUES ('ADMIN','open', true);

INSERT INTO ‘epicConfiguratorDb’.’user_roles’ (username, role) VALUES ('ADMIN', 'ROLE_ADMIN');

INSERT INTO ‘epicConfiguratorDb’.’category’ (`language_id`, `category_name`, `state`) VALUES ('EN', 'SERVER', '1');

INSERT INTO ‘epicConfiguratorDb’.’subcat_cat’
(category_id,category_name,subcategory_id,subcategory_name) VALUES
(1,"SERVER",8,"Performance");

INSERT INTO ‘epicConfiguratorDb’.’menu’
(language_id,menu_type_id,name,description,parent_id,state,categories_categor
y_id) VALUES ("en",1,"Servers","Proliant Servers",null,1,1);

Product data will be inserted by the User Admin and Logged-in User.

5.5. Application Deployment

To build the application and generate the WAR required to deploy it, I used the export utility that Eclipse IDE provides. After getting this WAR file, Tomcat server needs to be stopped, and we place this WAR under the “webapps” directory in Tomcat, after which we restart the server. This will deploy the application and make it ready and available in the browser.

6. CONCLUSION AND FUTURE DIRECTION

6.1 Conclusion

Working on this project was a very fulfilling experience in the technical and business aspects of the system. Translating business requirements into technical functionalities and implementing the right solution have substantially increased my knowledge of and experience in software engineering.

6.2 Future Direction

As the next step, I will add new functionalities and integrate more technologies based on Cloud Computing and Continuous Integration and Continuous Delivery (CI/CD). I will also provide the full support of the application for mobile devices. My intention is for this system to be a starting point for developing a Management System that can be implemented by diverse businesses, and provide the capability to add custom modules appropriate for a particular business.

APPENDIX

Object Database Representation

The metadata of each table is represented in the following classes, without considering getter and setter methods:

```
@Entity
@Table(name="CATEGORY")
public class Category{

	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	@Column(name="category_id")
	private Integer categoryId;

	@Column(name="language_id")
	private char languageId;

	@Column(name="category_name")
	private String categoryName;

	@Column(name="category_date")
	private Date creationDate;

	@Column(name="update_Date")
	private Date updateDate;

	@Column(name="category_Image_Url")
	private String categoryImageUrl;

	@Column(name="category_Thumbnail_Url")
	private String categoryThumbnailUrl;

	@Column(name="state")
	private String state;

	@OneToMany(mappedBy="category", cascade=CascadeType.ALL)
	private Set<Product> product;

}
```

Category Entity

```
@Entity
@Table(name="COUNTRY")
public class Country{
```

```
        @Id
        @GeneratedValue(strategy=GenerationType.IDENTITY)
        @Column(name="country_id")
        private Integer countryId;

        @Column(name="iso")
        private char iso;

        @Column(name="iso3")
        private String iso3;

        @Column(name="name")
        private String name;

        @Column(name="nicename")
        private String nicename;

        @Column(name="numcode")
        private Integer numcode;

        @Column(name="phonecode")
        private Integer phonecode;

}
```

Country Entity

```
@Entity
@Table(name="CUSTOMER")
public class Customer {

   @Id
   @Column(name="customer_id")
   @GeneratedValue(generator="gen")
   @GenericGenerator(name="gen",
strategy="foreign",parameters=@Parameter(name="property", value="user"))
   private String id;

        @OneToOne
        @PrimaryKeyJoinColumn
        private User user;

        @Column(name="customer_gender")
        private String customerGender;

        @Column(name="customer_firstname")
```

```
	private String customerFirstname;

	@Column(name="customer_lastname")
	private String customerLastname;

	@Column(name="customer_dob")
	private Date customerDob;

	@Column(name="customer_email_address")
	private String customerEmailAddress;

	@Column(name="customer_home_address")
	private String customerHomeAddress;

	@Column(name="customer_telephone")
	private String customerTelephone;

	@Column(name="customer_password")
	private String customerPassword;

	@Column(name="customer_shipping_address")
	private String customerShippingAddress;

}
```

Customer Entity

```
@Entity
@Table(name="PRODUCT")
public class Product {

	@ManyToOne
	@JoinColumn(name = "category_category_id")
	private Category category;

	@Column(name = "Price")
	private double price;

	@Column(name="id")
	private int productId;

	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	@Column(name="Part_Number")
	private String partNumber;

	@Column(name="Description")
	private String description;
```

```
@Column(name="Product_Line")
private String ProductLine;

@Column(name="Form_Factor")
private String FormFactor;

@Column(name="Warranty")
private String warranty;

@Column(name="Software")
private String software;

@Column(name="Processor")
private String processor;

@Column(name="Memory")
private String memory;

@Column(name="Graphics")
private String graphics;

@Column(name="Network")
private String network;

@Column(name="Disk_Storage")
private String diskStorage;

@Column(name="Disk_Storage_Backplane")
private String diskStorageBackplane;

@Column(name="Optical_Storage")
private String opticalStorage;

@Column(name="Power_Supply")
private String powerSupply;

@Column(name="Screen")
private String screen;

@Column(name="Wireless")
private String wireless;

@Column(name="Other_Features")
private String otherFeatures;

@Column(name="IO_Controller")
private String IOcontroller;

@Column(name="Manageability")
private String manageability;

@Column(name="Primary_Connectivity")
```

```
	private String primaryConnectivity;

	@Column(name="Power_Over_Ethernet")
	private String powerOverEthernet;

	@Column(name="Networking")
	private String networking;

	@Column(name="Disk_Storage_Controller")
	private String diskStorageController;

	@Column(name="Interface_Card_Slots")
	private String interfaceCardSlots;

	@Column(name="Management")
	private String management;

	@Column(name="Quantity")
	private String quantity;

}
```

Product Entity

```
@Entity
@Table(name="part_validation")
public class PartValidation {

	@EmbeddedId
	private PartValidationId partValId;

	@Column(name = "incompatible")
	@Type(type = "org.hibernate.type.NumericBooleanType")
	private boolean incompatible;

	@Column(name = "required")
	@Type(type = "org.hibernate.type.NumericBooleanType")
	private boolean required;

	@Column(name = "max_quantity")
	private int maxQuantity;

	@Column(name = "min_quantity")
	private int minQuantity;
}
```

PartValidation Entity

```
@Entity
@Table(name="menu")
public class Menu {

	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	@Column(name="menu_id")
	private Integer menuId;

	@Column(name="language_id")
	private String languageId;

	@Column(name="menu_type_id")
	private Integer menuTypeId;

	@Column(name="name")
	private String name;

	@Column(name="description")
	private String description;

	@Column(name="parent_id")
	private Integer parentId;

	@OneToMany(mappedBy="parent")
	private Set<Menu> menus;

	@Column(name="categories_category_id")
	private Integer categoryId;

	@ManyToOne(fetch=FetchType.EAGER, cascade={CascadeType.ALL})
	@JoinColumn(name="parent_id", insertable = false, updatable = false)
	private Menu parent;

}
```

Menu Entity

```
@Entity
@Table(name="USERS")
```

```
public class User {

	@Id
	@Column(name = "username", unique = true,nullable = false, length = 45)
	private String username;

	@Column(name = "password", nullable = false, length = 60)
	private String password;

	@Column(name = "enabled", nullable = false)
	private boolean enabled;

	@OneToOne(mappedBy="user", cascade = CascadeType.ALL)
	private Customer customer;

	@OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
	private Set<UserRole> userRole = new HashSet<UserRole>(0);

}
```

User Entity

```
@Entity
@Table(name="USER_ROLES")
public class UserRole {

	@Id
	@GeneratedValue(strategy = IDENTITY)
	@Column(name = "user_role_id",unique = true, nullable = false)
	private Integer userRoleId;

	@ManyToOne(fetch = FetchType.LAZY)
	@JoinColumn(name = "username", nullable = false)
	private User user;

	@Column(name = "role", nullable = false, length = 45)
	private String role;

}
```

UserRole Entity

```
@Entity
@Table(name="configuration_product")
public class ConfigurationProduct implements java.io.Serializable{

	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy = IDENTITY)
	@Column(name = "configuration_product_Id",unique = true, nullable = false)
	private Integer confProductId;

	@ManyToOne(cascade=CascadeType.ALL)
	@JoinColumn(name = "quote_configuration_quote_id")
	private QuoteConfiguration quoteConfiguration;

	@Column(name = "price")
	private String price;

	@Column(name = "quantity")
	private Integer quantity;

	@Column(name = "product_Part_Number",nullable=false)
	private String productPartNumber;

	public ConfigurationProduct(){
	}
	public Integer getConfProductId() {
		return confProductId;
	}
	public void setConfProductcId(Integer confProductId) {
		this.confProductId = confProductId;
	}

}
```

ConfigurationProduct Entity

```
@Entity
@Table(name="QUOTE_CONFIGURATION")
public class QuoteConfiguration implements java.io.Serializable{

	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy = IDENTITY)
	@Column(name = "quote_id",unique = true, nullable = false)
```

```
        private Integer quoteId;

        @Column(name = "base_product", nullable = false)
        private String baseProduct;

        @Column(name = "quote_name")
        private String quoteName;

        @Column(name = "customer_customer_id")
        private String customerId;

        @Column(name = "creation_date")
        private Date creationDate;

        @Column(name = "update_date")
        private Date updateDate;

        @Column(name = "price_total")
        private String priceTotal;

        @OneToMany(cascade=CascadeType.ALL,mappedBy="quoteConfiguration")
        private Set<ConfigurationProduct> configurationProduct= new
HashSet<ConfigurationProduct>(0);

}
```

QuoteConfiguration Entity

Database Schema

The schema definition of the database is the following:

EpicConfiguratorDB Schema script

```
DROP SCHEMA IF EXISTS `epicConfiguratorDB` ;

-- -----------------------------------------------------
-- Schema epicConfiguratorDB
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `epicConfiguratorDB` DEFAULT CHARACTER SET latin1
;

USE `epicConfiguratorDB` ;

-- -----------------------------------------------------
-- Table `epicConfiguratorDB`.`category`
```

```
-- -----------------------------------------------------
DROP TABLE IF EXISTS `epicConfiguratorDB`.`category` ;

CREATE TABLE IF NOT EXISTS `epicConfiguratorDB`.`category` (
  `category_id` INT(11) NOT NULL AUTO_INCREMENT,
  `language_id` VARCHAR(45) NULL DEFAULT NULL,
  `category_name` VARCHAR(45) NOT NULL,
  `category_date` DATE NULL DEFAULT NULL,
  `update_date` DATE NULL DEFAULT NULL,
  `category_image_url` VARCHAR(45) NULL DEFAULT NULL,
  `category_thumbnail_url` VARCHAR(45) NULL DEFAULT NULL,
  `state` VARCHAR(45) NULL DEFAULT NULL,
  PRIMARY KEY (`category_id`))
ENGINE = InnoDB
AUTO_INCREMENT = 42
DEFAULT CHARACTER SET = latin1;


-- -----------------------------------------------------
-- Table `epicConfiguratorDB`.`product`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `epicConfiguratorDB`.`product` ;

CREATE TABLE IF NOT EXISTS `epicConfiguratorDB`.`product` (
  `id` INT(11) NULL DEFAULT NULL,
  `Part_Number` VARCHAR(200) NOT NULL,
  `Description` VARCHAR(500) NULL DEFAULT NULL,
  `Product_Line` VARCHAR(200) NULL DEFAULT NULL,
  `Form_Factor` VARCHAR(200) NULL DEFAULT NULL,
  `Warranty` VARCHAR(200) NULL DEFAULT NULL,
  `Software` VARCHAR(200) NULL DEFAULT NULL,
  `category_category_id` INT(11) NOT NULL,
  `Processor` VARCHAR(200) NULL DEFAULT NULL,
  `Memory` VARCHAR(200) NULL DEFAULT NULL,
  `Graphics` VARCHAR(200) NULL DEFAULT NULL,
  `Network` VARCHAR(200) NULL DEFAULT NULL,
  `Disk_Storage` VARCHAR(200) NULL DEFAULT NULL,
  `Disk_Storage_Backplane` VARCHAR(200) NULL DEFAULT NULL,
  `Optical_Storage` VARCHAR(200) NULL DEFAULT NULL,
  `Power_Supply` VARCHAR(200) NULL DEFAULT NULL,
  `Screen` VARCHAR(200) NULL DEFAULT NULL,
  `Wireless` VARCHAR(200) NULL DEFAULT NULL,
  `Other_Features` VARCHAR(200) NULL DEFAULT NULL,
  `IO_Controller` VARCHAR(200) NULL DEFAULT NULL,
  `Manageability` VARCHAR(200) NULL DEFAULT NULL,
  `Primary_Connectivity` VARCHAR(200) NULL DEFAULT NULL,
  `Power_Over_Ethernet` VARCHAR(200) NULL DEFAULT NULL,
  `Networking` VARCHAR(200) NULL DEFAULT NULL,
  `Disk_Storage_Controller` VARCHAR(200) NULL DEFAULT NULL,
  `Interface_Card_Slots` VARCHAR(200) NULL DEFAULT NULL,
  `Management` VARCHAR(200) NULL DEFAULT NULL,
  `Price` DOUBLE(10,0) NOT NULL DEFAULT '0',
  `Quantity` INT(11) NULL DEFAULT NULL,
```

```
  PRIMARY KEY (`Part_Number`, `category_category_id`),
  INDEX `fk_product_categories1_idx` (`category_category_id` ASC),
  CONSTRAINT `fk_product_categories1`
    FOREIGN KEY (`category_category_id`)
    REFERENCES `epicConfiguratorDB`.`category` (`category_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;


-- -----------------------------------------------------
-- Table `epicConfiguratorDB`.`users`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `epicConfiguratorDB`.`users` ;

CREATE TABLE IF NOT EXISTS `epicConfiguratorDB`.`users` (
  `username` VARCHAR(45) NOT NULL,
  `password` VARCHAR(60) NOT NULL,
  `enabled` TINYINT(4) NOT NULL DEFAULT '1',
  PRIMARY KEY (`username`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;


-- -----------------------------------------------------
-- Table `epicConfiguratorDB`.`customer`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `epicConfiguratorDB`.`customer` ;

CREATE TABLE IF NOT EXISTS `epicConfiguratorDB`.`customer` (
  `customer_id` VARCHAR(45) NOT NULL,
  `customer_gender` VARCHAR(45) NULL DEFAULT NULL,
  `customer_firstname` VARCHAR(45) NULL DEFAULT NULL,
  `customer_lastname` VARCHAR(45) NULL DEFAULT NULL,
  `customer_dob` DATE NULL DEFAULT NULL,
  `customer_email_address` VARCHAR(45) NULL DEFAULT NULL,
  `customer_home_address` VARCHAR(45) NULL DEFAULT NULL,
  `customer_telephone` VARCHAR(45) NULL DEFAULT NULL,
  `customer_password` VARCHAR(45) NULL DEFAULT NULL,
  `customer_shipping_address` VARCHAR(45) NULL DEFAULT NULL,
  PRIMARY KEY (`customer_id`),
  CONSTRAINT `customer_user`
    FOREIGN KEY (`customer_id`)
    REFERENCES `epicConfiguratorDB`.`users` (`username`)
    ON DELETE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;


-- -----------------------------------------------------
-- Table `epicConfiguratorDB`.`quote_configuration`
-- -----------------------------------------------------
```

```
DROP TABLE IF EXISTS `epicConfiguratorDB`.`quote_configuration` ;

CREATE TABLE IF NOT EXISTS `epicConfiguratorDB`.`quote_configuration` (
  `quote_id` INT(11) NOT NULL AUTO_INCREMENT,
  `customer_customer_id` VARCHAR(45) NULL DEFAULT NULL,
  `quote_name` VARCHAR(45) NULL DEFAULT NULL,
  `creation_date` DATETIME NULL DEFAULT NULL,
  `update_date` DATETIME NULL DEFAULT NULL,
  `price_total` VARCHAR(45) NULL DEFAULT NULL,
  `base_product` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`quote_id`),
  INDEX `fk_quote_configuration_customer1_idx` (`customer_customer_id` ASC),
  CONSTRAINT `fk_quote_configuration_customer1`
    FOREIGN KEY (`customer_customer_id`)
    REFERENCES `epicConfiguratorDB`.`customer` (`customer_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;


-- -----------------------------------------------------
-- Table `epicConfiguratorDB`.`configuration_product`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `epicConfiguratorDB`.`configuration_product` ;

CREATE TABLE IF NOT EXISTS `epicConfiguratorDB`.`configuration_product` (
  `quote_configuration_quote_id` INT(11) NOT NULL,
  `configuration_product_Id` INT(11) NOT NULL AUTO_INCREMENT,
  `price` VARCHAR(45) NULL DEFAULT NULL,
  `quantity` INT(11) NULL DEFAULT NULL,
  `product_Part_Number` VARCHAR(200) NOT NULL,
  PRIMARY KEY (`configuration_product_Id`),
  INDEX `fk_configuration_product_quote_configuration1_idx` (`quote_configuration_quote_id`
ASC),
  INDEX `fk_configuration_product_product1_idx` (`product_Part_Number` ASC),
  CONSTRAINT `fk_configuration_product_product1`
    FOREIGN KEY (`product_Part_Number`)
    REFERENCES `epicConfiguratorDB`.`product` (`Part_Number`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_configuration_product_quote_configuration1`
    FOREIGN KEY (`quote_configuration_quote_id`)
    REFERENCES `epicConfiguratorDB`.`quote_configuration` (`quote_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

-- -----------------------------------------------------
-- Table `epicConfiguratorDB`.`menu`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `epicConfiguratorDB`.`menu` ;
```

```
CREATE TABLE IF NOT EXISTS `epicConfiguratorDB`.`menu` (
  `menu_id` INT(11) NOT NULL AUTO_INCREMENT,
  `language_id` VARCHAR(45) NULL DEFAULT NULL,
  `menu_type_id` INT(11) NULL DEFAULT NULL,
  `name` VARCHAR(45) NULL DEFAULT NULL,
  `description` VARCHAR(45) NULL DEFAULT NULL,
  `parent_id` INT(11) NULL DEFAULT NULL,
  `state` VARCHAR(45) NULL DEFAULT NULL,
  `categories_category_id` INT(11) NOT NULL,
  PRIMARY KEY (`menu_id`),
  INDEX `fk_menu_categories1` (`categories_category_id` ASC),
  CONSTRAINT `fk_menu_categories1`
    FOREIGN KEY (`categories_category_id`)
    REFERENCES `epicConfiguratorDB`.`category` (`category_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
AUTO_INCREMENT = 13
DEFAULT CHARACTER SET = latin1;


-- -----------------------------------------------------
-- Table `epicConfiguratorDB`.`part_product`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `epicConfiguratorDB`.`part_product` ;

CREATE TABLE IF NOT EXISTS `epicConfiguratorDB`.`part_product` (
  `product_PN` VARCHAR(200) NOT NULL,
  `product_product_category_id` INT(11) NOT NULL,
  `Part_PN` VARCHAR(200) NOT NULL,
  `part_product_category_id` INT(11) NOT NULL,
  PRIMARY KEY (`product_PN`, `product_product_category_id`, `Part_PN`,
`part_product_category_id`),
  INDEX `fk_part_product_product2` (`Part_PN` ASC, `part_product_category_id` ASC),
  CONSTRAINT `fk_part_product_product1`
    FOREIGN KEY (`product_PN` , `product_product_category_id`)
    REFERENCES `epicConfiguratorDB`.`product` (`Part_Number` , `category_category_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_part_product_product2`
    FOREIGN KEY (`Part_PN` , `part_product_category_id`)
    REFERENCES `epicConfiguratorDB`.`product` (`Part_Number` , `category_category_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;


-- -----------------------------------------------------
-- Table `epicConfiguratorDB`.`part_validation`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `epicConfiguratorDB`.`part_validation` ;
```

```
CREATE TABLE IF NOT EXISTS `epicConfiguratorDB`.`part_validation` (
  `product_Part_Number` VARCHAR(200) NOT NULL,
  `product_category_category_id` INT(11) NOT NULL,
  `part_number` VARCHAR(45) NOT NULL,
  `incompatible` TINYINT(1) NULL DEFAULT NULL,
  `required` TINYINT(1) NULL DEFAULT NULL,
  `max_quantity` INT(11) NULL DEFAULT NULL,
  `min_quantity` INT(11) NULL DEFAULT NULL,
  PRIMARY KEY (`product_Part_Number`, `product_category_category_id`, `part_number`),
  CONSTRAINT `fk_part_compatible_product1`
    FOREIGN KEY (`product_Part_Number` , `product_category_category_id`)
    REFERENCES `epicConfiguratorDB`.`product` (`Part_Number` , `category_category_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;


-- -----------------------------------------------------
-- Table `epicConfiguratorDB`.`subcat_cat`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `epicConfiguratorDB`.`subcat_cat` ;

CREATE TABLE IF NOT EXISTS `epicConfiguratorDB`.`subcat_cat` (
  `category_id` INT(11) NOT NULL,
  `subcategory_id` INT(11) NOT NULL,
  PRIMARY KEY (`category_id`, `subcategory_id`),
  INDEX `fk_subcat_cat_category2` (`subcategory_id` ASC),
  CONSTRAINT `fk_subcat_cat_category1`
    FOREIGN KEY (`category_id`)
    REFERENCES `epicConfiguratorDB`.`category` (`category_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_subcat_cat_category2`
    FOREIGN KEY (`subcategory_id`)
    REFERENCES `epicConfiguratorDB`.`category` (`category_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;


-- -----------------------------------------------------
-- Table `epicConfiguratorDB`.`user`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `epicConfiguratorDB`.`user` ;

CREATE TABLE IF NOT EXISTS `epicConfiguratorDB`.`user` (
  `user_id` INT(11) NOT NULL AUTO_INCREMENT,
  `user_name` VARCHAR(60) NOT NULL,
  `password` VARCHAR(60) NOT NULL,
  `enabled` TINYINT(4) NOT NULL DEFAULT '1',
```

```
  `role` VARCHAR(45) NULL DEFAULT NULL,
  PRIMARY KEY (`user_id`, `user_name`))
ENGINE = InnoDB
AUTO_INCREMENT = 2
DEFAULT CHARACTER SET = latin1;


-- -----------------------------------------------------
-- Table `epicConfiguratorDB`.`user_roles`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `epicConfiguratorDB`.`user_roles` ;

CREATE TABLE IF NOT EXISTS `epicConfiguratorDB`.`user_roles` (
  `user_role_id` INT(11) NOT NULL AUTO_INCREMENT,
  `username` VARCHAR(45) NOT NULL,
  `role` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`user_role_id`),
  UNIQUE INDEX `uni_username_role` (`role` ASC, `username` ASC),
  INDEX `fk_username_idx` (`username` ASC),
  CONSTRAINT `fk_username`
    FOREIGN KEY (`username`)
    REFERENCES `epicConfiguratorDB`.`users` (`username`))
ENGINE = InnoDB
AUTO_INCREMENT = 9
DEFAULT CHARACTER SET = latin1;
```

REFERENCES

[1] J2EE. [Online]. Viewed 2017 February 2. Available: http://www.oracle.com/technetwork/java/javaee/overview/index.html

[2] Hibernate. [Online]. Viewed 2017 February 16. Available: http://hibernate.org/orm/

[3] Spring MVC. [Online]. Viewed 2017 March 12. Available: https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html

[4] Spring Security. [Online]. Viewed 2017 April 10. Available: https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/

[5] Bootstrap. [Online]. Viewed 2017 April 10. Available: http://getbootstrap.com/

[6] Bootstrap. [Online]. Viewed 2017 April 10. Available: https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)

[7] Apache Solr. [Online]. Viewed 2017 April 20. Available: http://lucene.apache.org/solr/features.html

[8] Apache Lucene. [Online]. Viewed 2017 April 20. Available: https://lucene.apache.org/core/

[9] Oracle. Java platform, standard edition 1.8. Viewed 2017 April 20. Available: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

[10] Spring Core Technologies. Viewed 2018 February 27. Available:

https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html