Capstone Project

Link to source code on GitHub:

<https://github.com/MattStaxx/Capstone>

Link to Heroku deployment:

<https://capstone-musicstore.herokuapp.com/>

The capstone project goal was to create a website for a music store using a spring boot application. The project had multiple requirements including a customer facing pages such as catalog and shopping and admin facing pages such as inventory and order management. In order to achieve this functionality, multiple technologies were used including spring security, JSP, Bootstrap, SQL, and Heroku web deployment.

The first steps taken was to build a spring boot application using and implement spring security. Spring security is a powerful authentication and access-control framework for use with Java Spring applications. It is customizable to the specific requirements of numerous project configurations and allows developers to implement protection against most of the security threats that target web-based applications, while allowing users a seamless UI experience. Spring security works on four principal elements of security to achieve this result. Authentication, Authorization, Password storage, and Servlet filters.

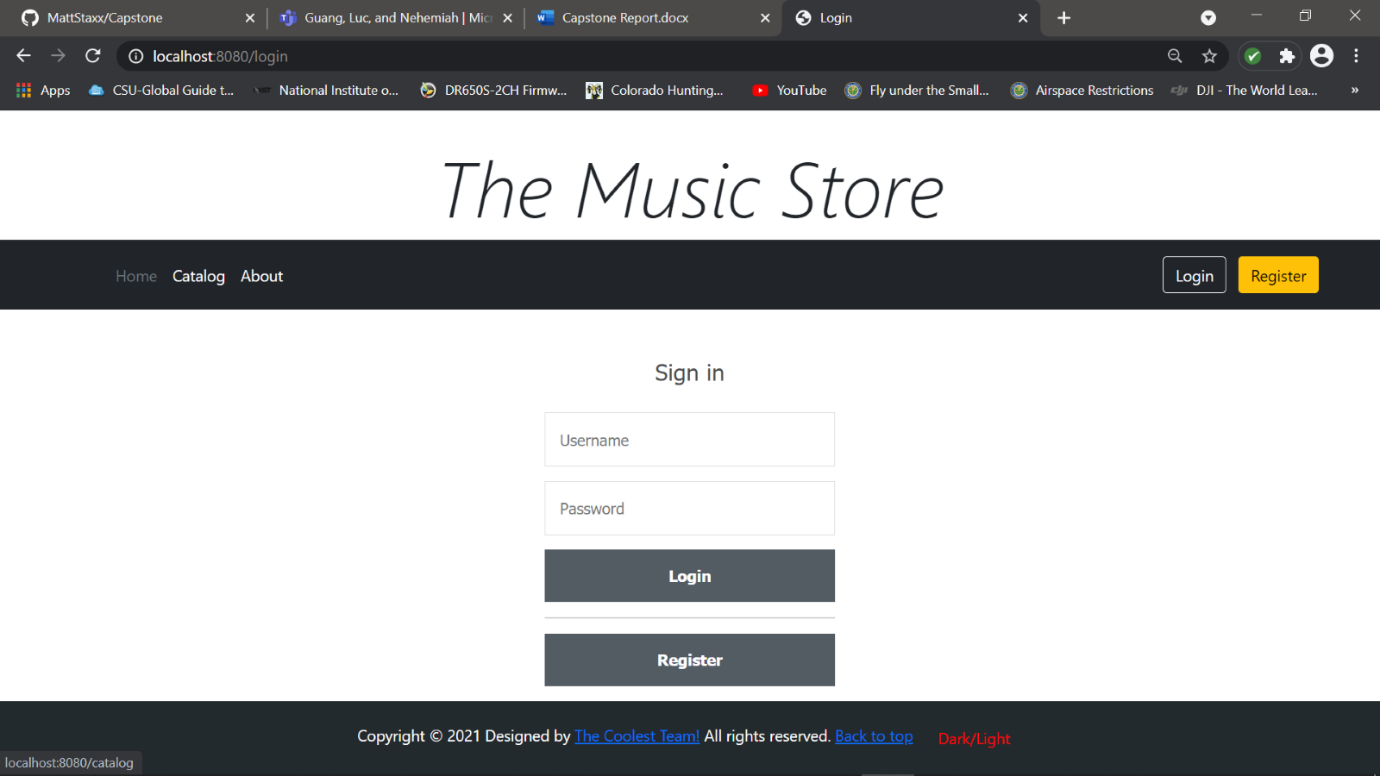
At the core of these principles is the password encoder. For this project we've utilized the BCryptPasswordEncoder, available withing the Spring Security framework. This grants the ability for passwords to be stored using a salted and hashed version of a given password with the Blowfish block cipher. This grants a great balance of defense from brute force password cracking attempts, while still providing a pleasant “honest” user experience while logging in.

Once the encoded password is stored, it is then used in combination with assigned user roles to determine what a user can access. This is where authentication and authorization come in. Once a user is authenticated against the stored password, the various site-specific roles that have been configured by system administrators will determine which resources within a site a user can access.

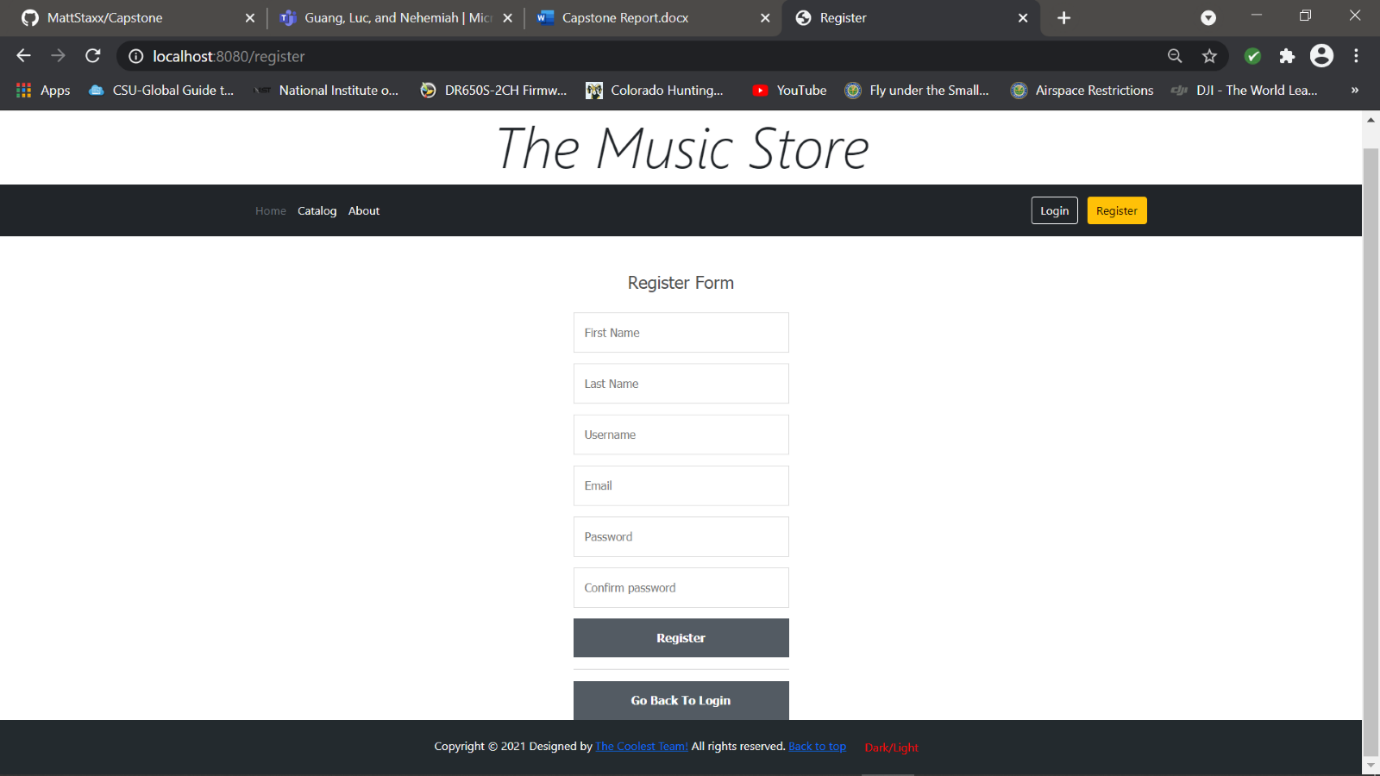
The authorization describe above is then enforced through the implementation of servlet filters. Within Spring projects servlets are handled behind-the-scenes, rather than explicit construction and configuration of servlet classes, and utilize the use of annotations to “wire” everything together and inform Spring as to the desired behavior expected from the servlet.

All together these four elements work in concert to ensure that web applications maintain availability to approved users, while maintaining confidentiality of system resources, which helps to guarantee the integrity of the application and the data stored within, all within an easy to use (for developers) framework.

 The login and register detail pages can be seen below.



**Figure 1. Screenshot of Login Page**



**Figure 2. Screenshot of Register Page**

The next steps were to build the backend layer which would be done using JPA entities. There were three entities that mapped to database tables. Each entity has a repository interface which extended the JPA Repository for developing methods to be auto wired into the service classes. The logic for the backend used various relationships in order to accomplish the business logic. The user table had the user details which included name, username, password, role, and etc. The product table had all the product details such as artist name and genre. The order table had all the order details such as customer and status. The product had a many to one relationship with the user so that a customer could have multiple products associated with them in a cart. The order also had one-to-many relationship with the products so that multiple products can be in one order. The last relationship was the one-to-many relationship between the user and order so that customer could have multiple orders at one time. The logic flowed when a customer puts a product into the cart then the product is then duplicated in the table with that user attached to it. Once the customer has placed the products are then mapped to an order and the order is also mapped to a user. The entity relationship diagram can be seen below.

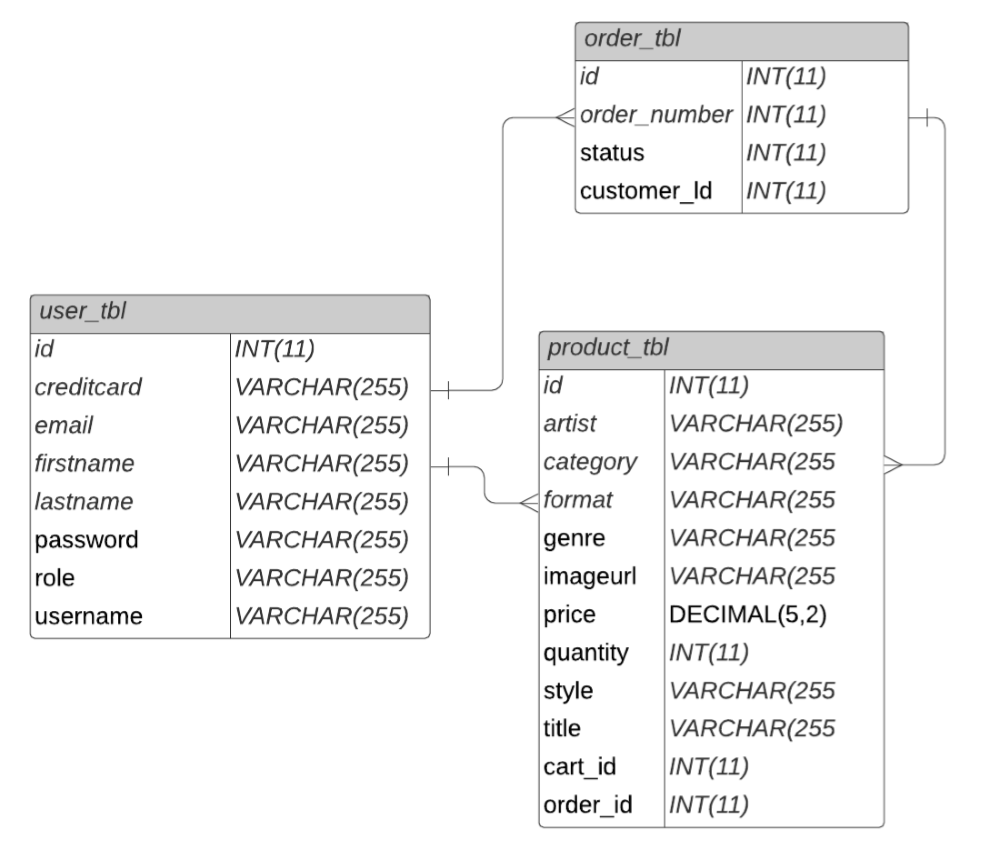


Figure 3. Entity Relationship Diagram

After the backend was developed, the controller was created in order to act as the medium between front end and backend. The first part to implement was the admin controller since they would be in control of the customer view pages such as manage the inventory which would reflect in the catalog. The admin controller would control three main JSP pages with specific functions, inventory management, user management, and order management. For the inventory management, the JSP was created to display a table of all the existing inventory and a form for adding and editing the inventory. The admin controller requested the values of the different forms and would pass the values to service method which would then persist the changes to the database. The next JSP for the controller to operate would be the manage orders, the only feature on this page was to update the status of the order sch as shipped or other status. The final JSP was the user management this is used to add, edit, or delete a user; however, the password would display to the admin encrypted so as not to jeopardize security.

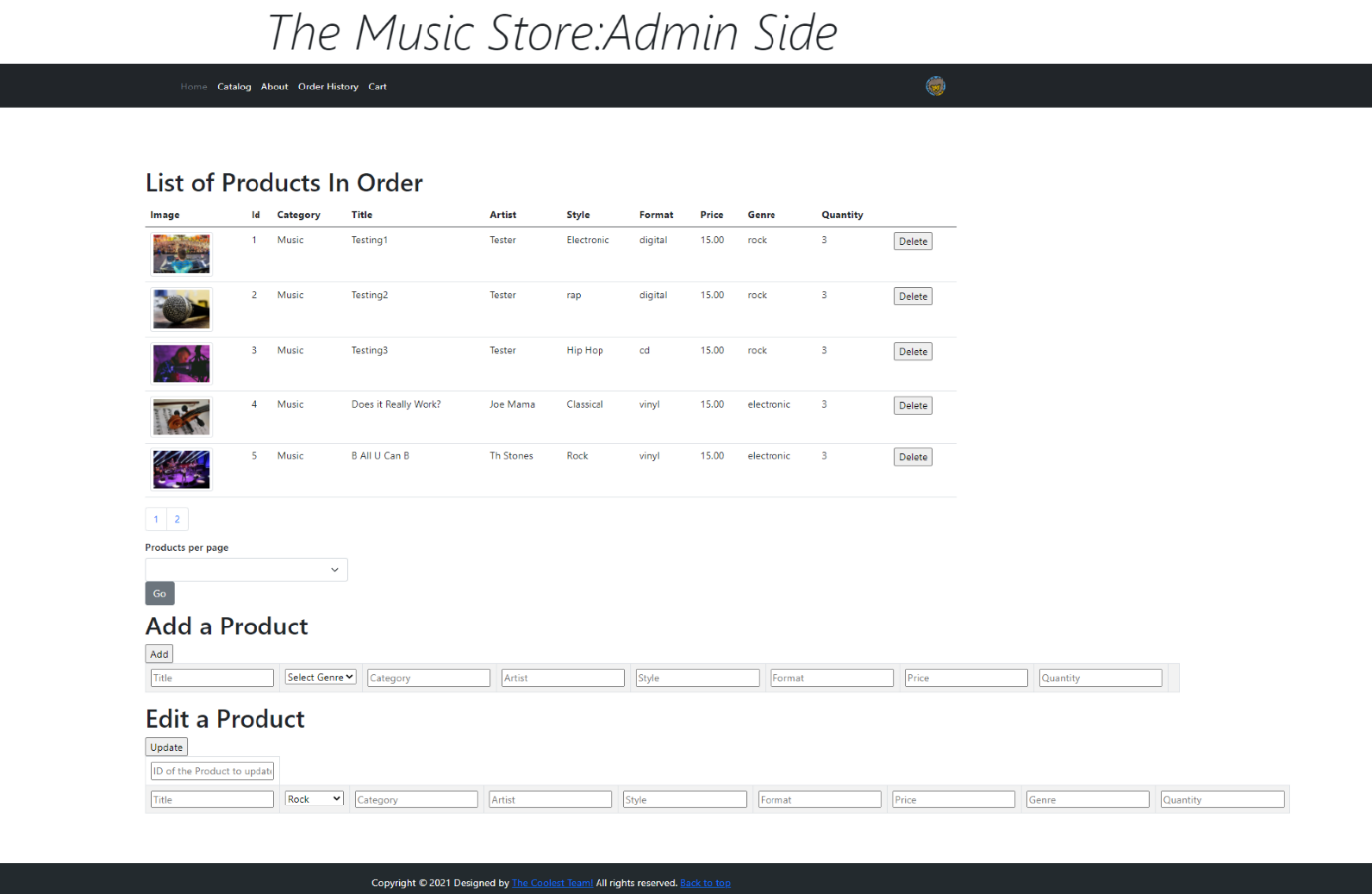


Figure 4.  Manage Inventory Page

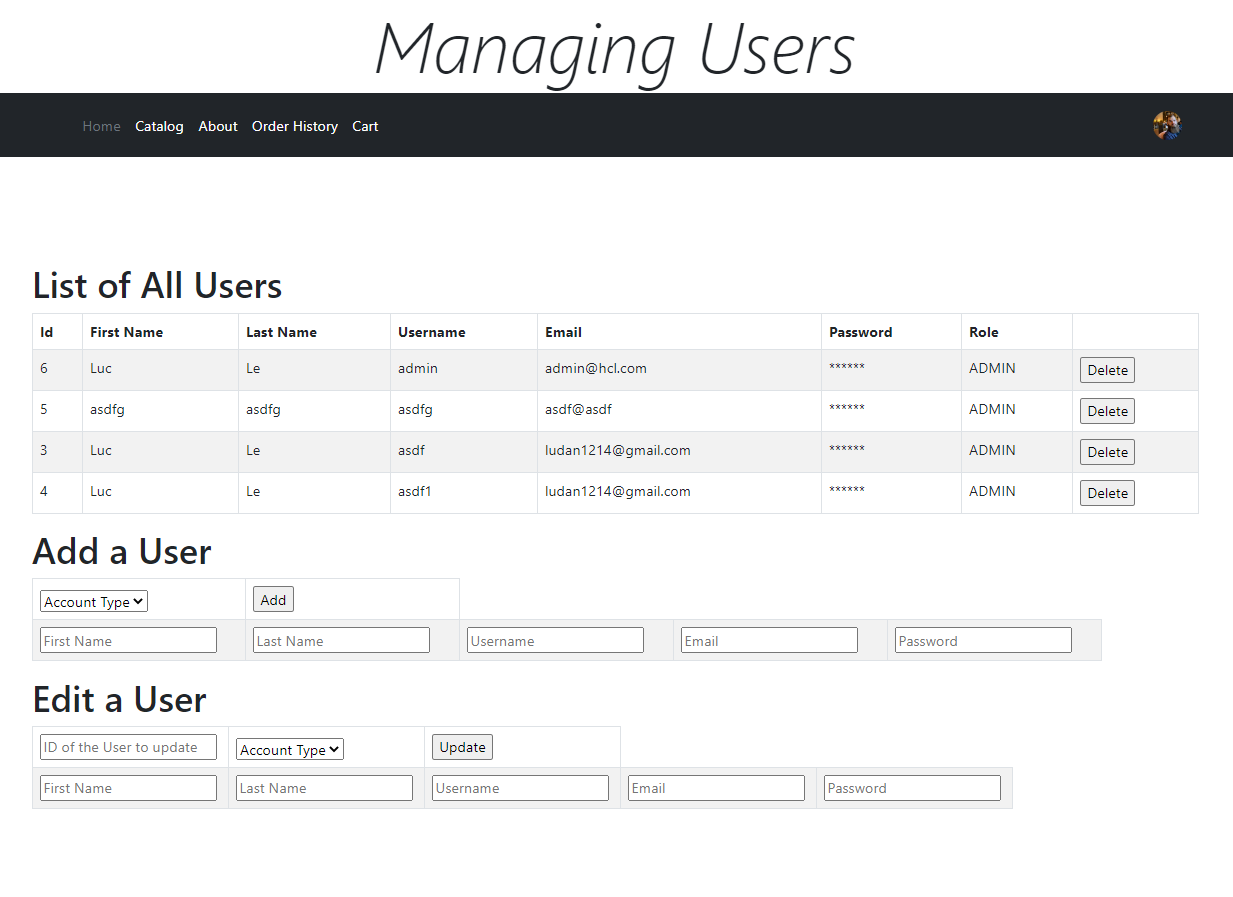


Figure 5. Manage User Page

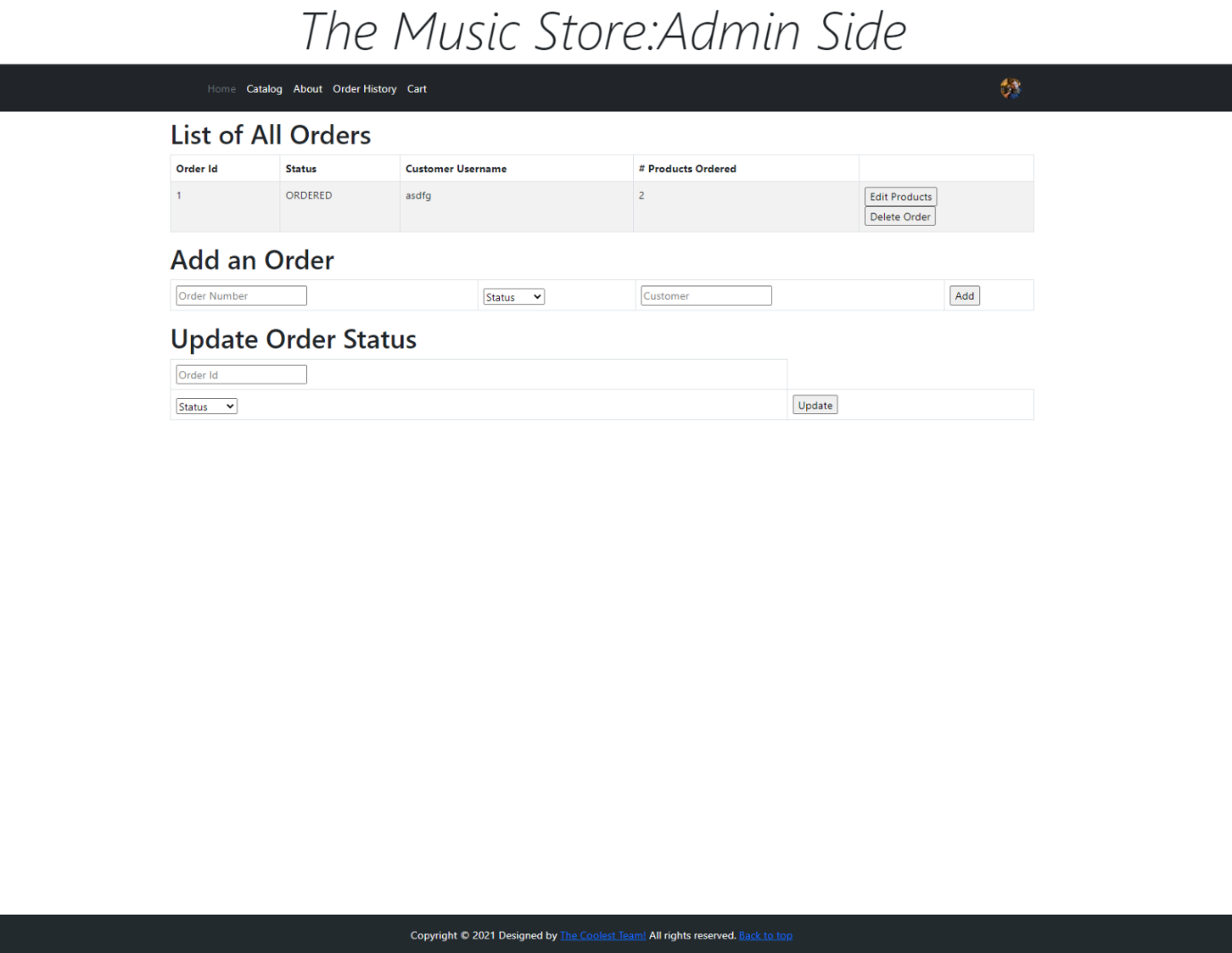


Figure 6. Manage order Page

Next, the customer controller was added to operate all the customer end functionality.  This controller had a few JSPs to control such as catalog, shopping cart, and order history. The catalog feature used two html forms which were displayed based on the search criteria such as artist name or price. All input except for price was passed in a string to the controller which then called a service method that used criteria API to return a list of the matching products; the predicate took in both the variable name and the desired output to find a list of the matching results. However, the default was to display all products. Since the products would could be mapped to both a user or an order a display method was created in the service in order to only display the products that had both those columns as null. Each item in the catalog had a link to display more details. Both the product and the catalog had a button that would add the product selected to the cart. Once the customer finished adding products to cart, the checkout feature would forward the customer to the payment JSP. This JSP took all the information to be associated to the order. After an order was processed the products would then be mapped to an order instead of a customer in order for them to make another order. The final JSP was the order history which would display all the orders by the user which would be accomplished by the service that would find all orders by that customer.

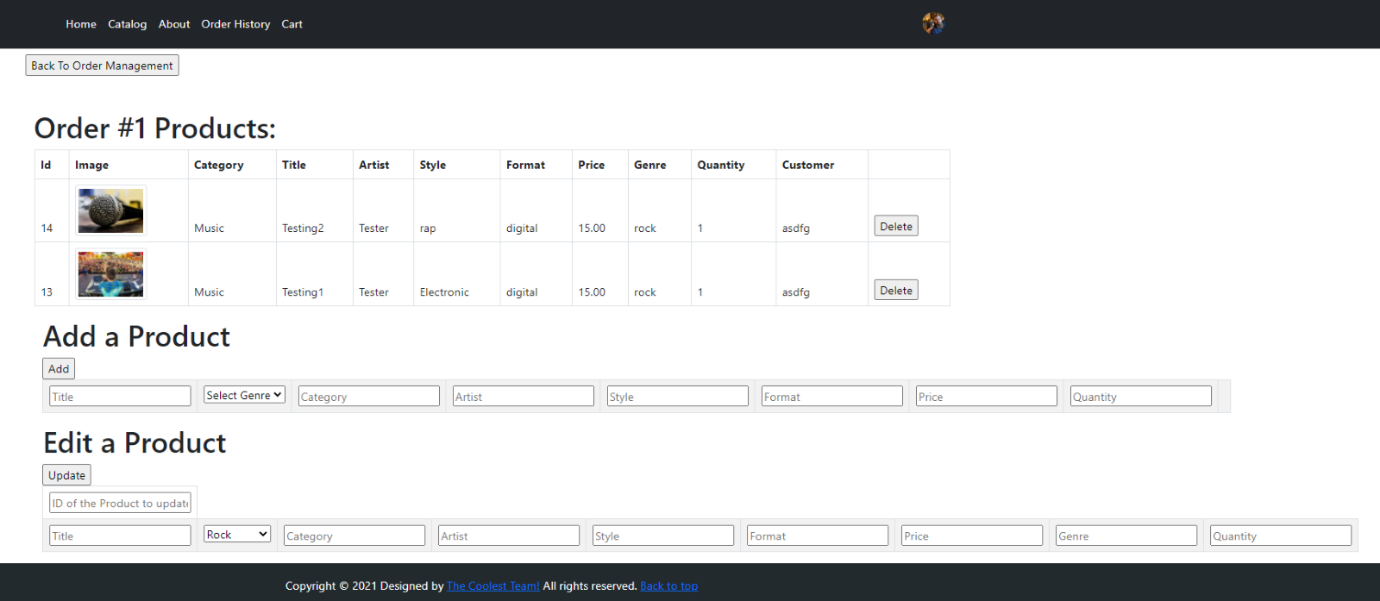


Figure 7. Order Edit Page

From the order management page, a separate edit page tailored to each order can be generated, an admin user has the ability to view, add, edit, or delete the products in each individual order. The endpoints used are the same as in the product manager, however an additional order ID parameter is sent to the controller to distinguish between the two operations. Both are essentially identical operations, but error and success messages will now be attached and sent back to this order edit page to allow easy editing of an individual order.

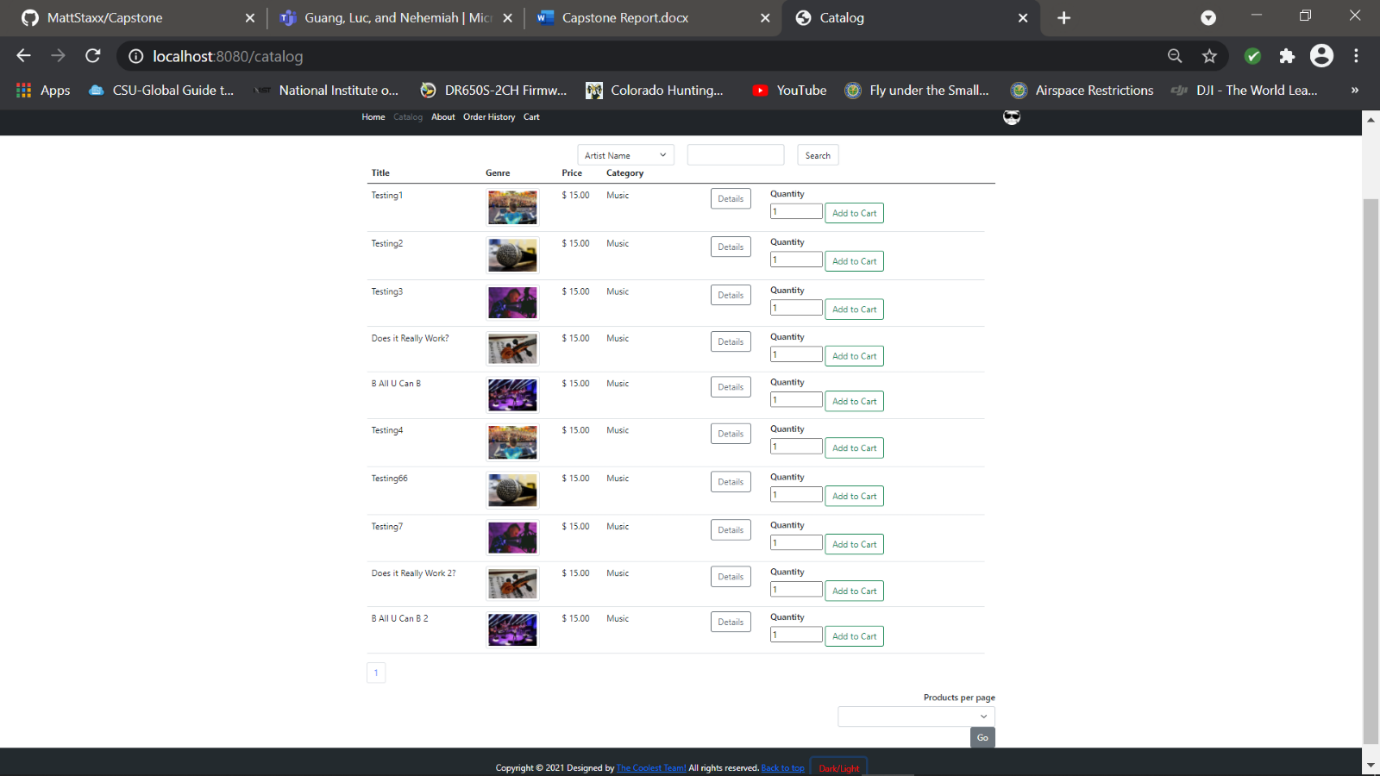


Figure 8. Catalog Page

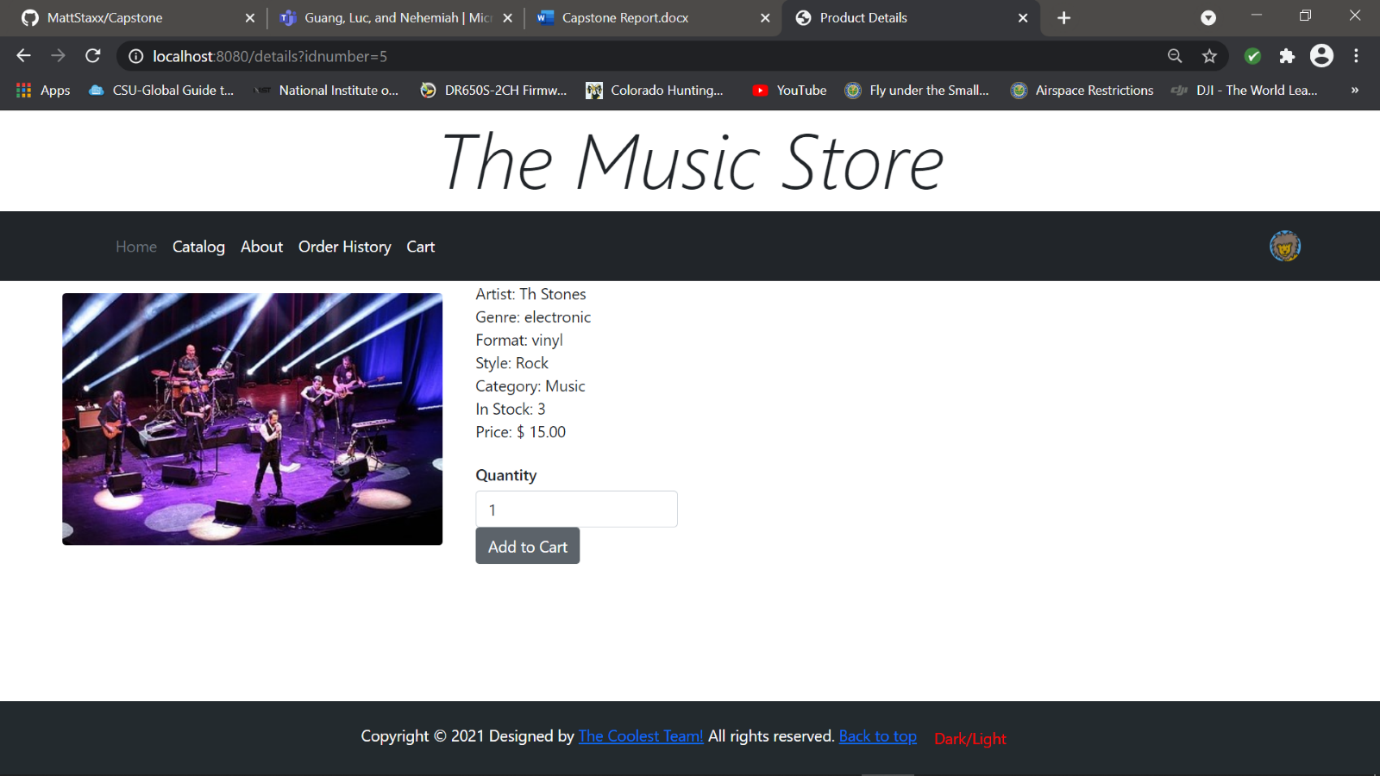


Figure 9. Product Details Page

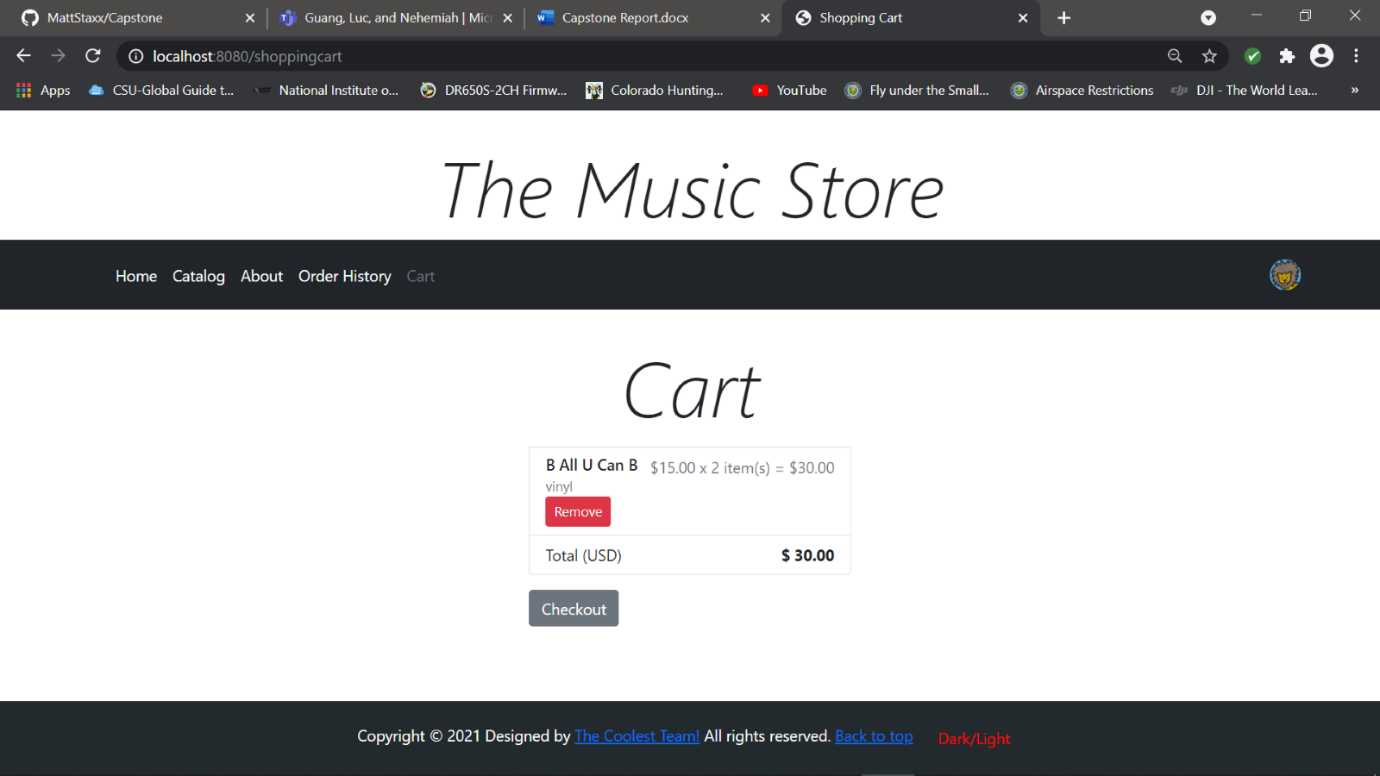


Figure 10. Shopping Cart Page

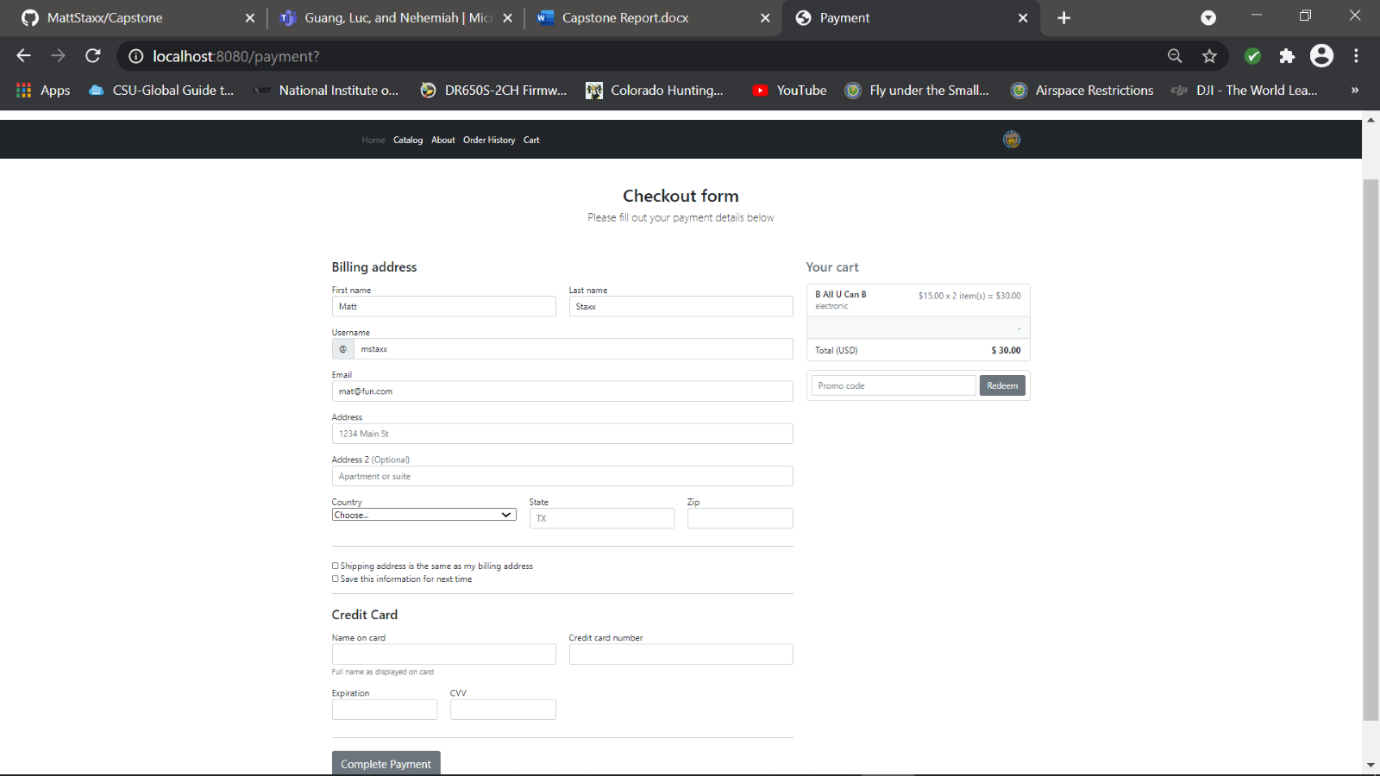


Figure 11. Payment Page

The last part of implementation was developing the frontend of the application to create a professional look to the website using Bootstrap. Some Bootstrap features that were used in order to make attractive website was adding a carousel to the homepage and specially formatted tables. In additions, the images were displayed to the catalog using the thumbnail tag and on the details page they were displayed as figures in order highlight the image more. A header and footer were added to increase consistency through webpages except for the homepage which was designed to be more eye-catching just the first thing to be seen. Another added feature was the use of pagination so that once catalog is fully populated with a significant amount of data it would not show up on page but would have a limit. In addition, JavaScript and CSS files were added to increase the styling and functionality. The overall design and flowchart and home page can be seen below.

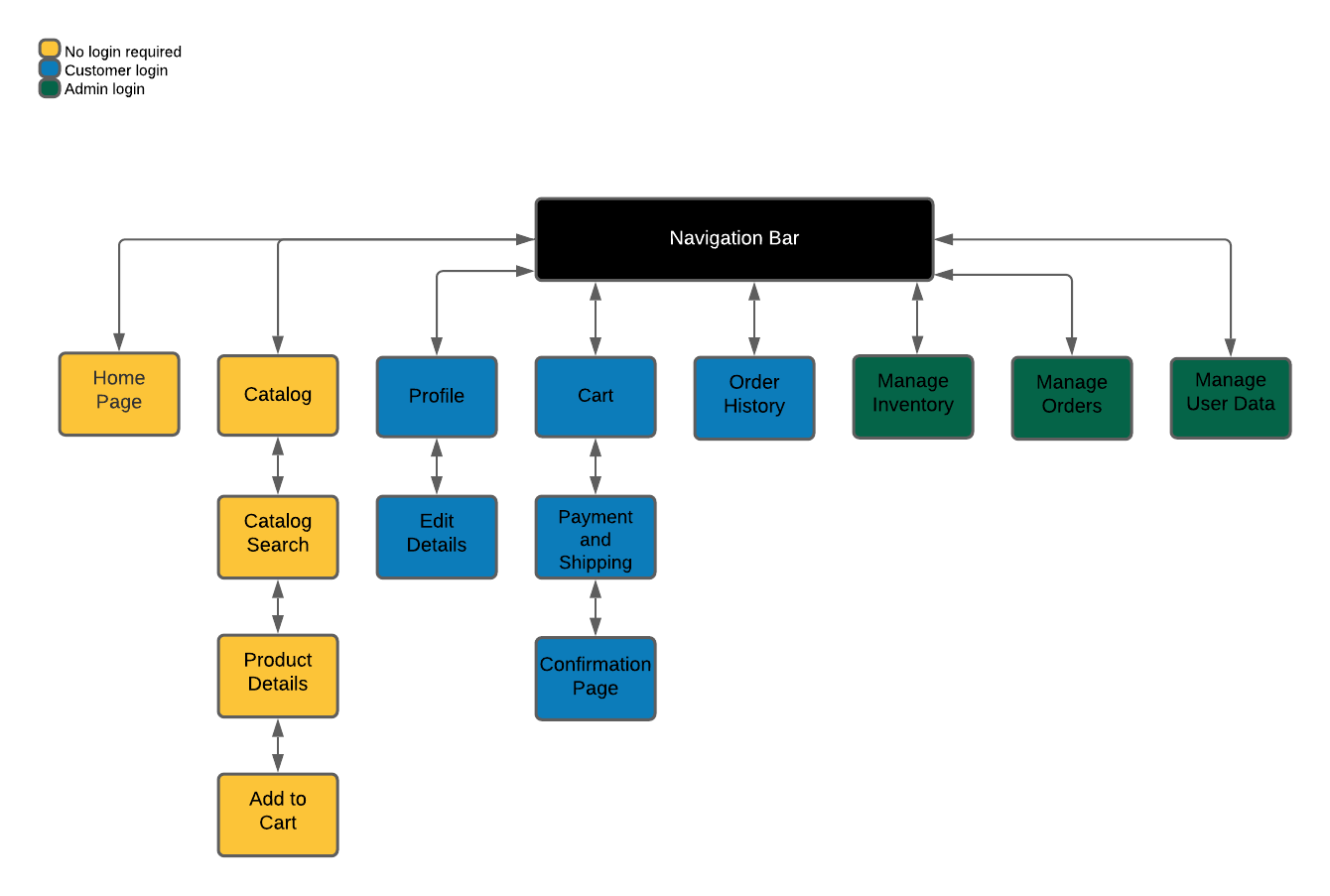


Figure 12. Flowchart

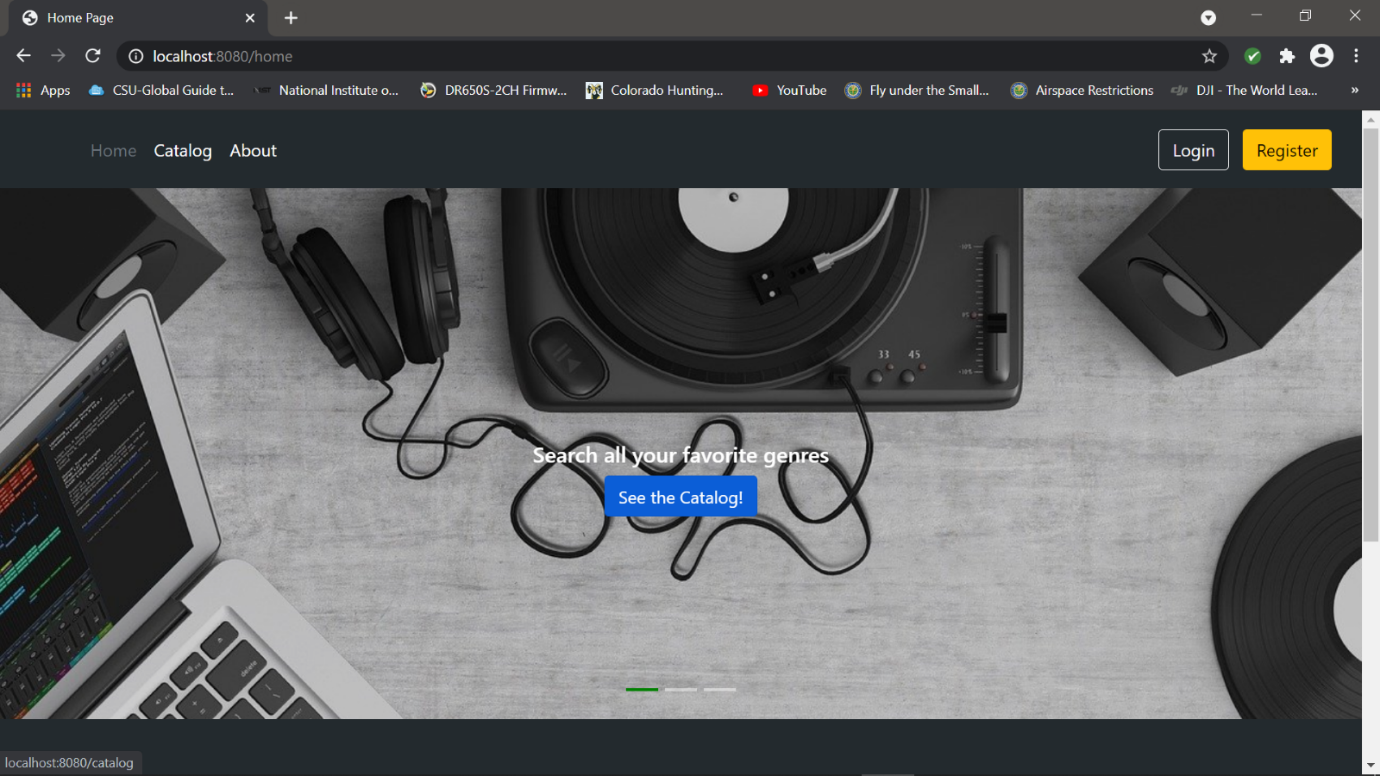


Figure 13. Home Page

In conclusion, the website implemented all the needed functionality. Some special added features were making a carousel for the home page and having a generic avatar showing when someone was logged in. In addition, to optimize security all passwords were stored used encryption.  However, some improvements could be made to optimize the program. The back-end mapping could have optimized by not adding duplicate products but by having separate entities for the user, product, order, and cart.