**Introduction:**

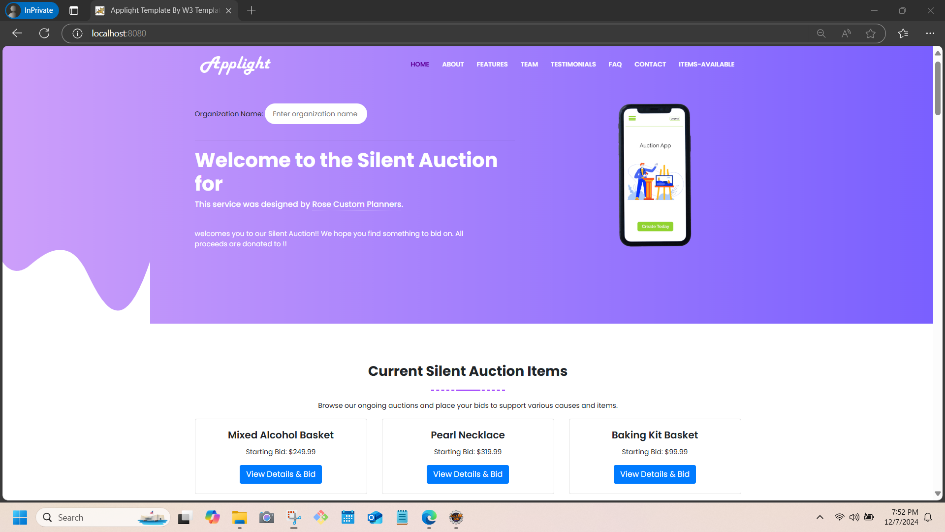
This project specifically focuses on AngularJS integration and implementing basic CRUD functionality, as outlined in the two previous project requirements. However, for a more secure and robust application, adding user logins and authentication could be a valuable enhancement. With user authentication, the application could ensure secure access and management of product data, allowing different users with varying roles (e.g., admin, regular user) to interact with the product data according to their permissions.

The **Product Management Application** is a web-based system designed to manage products, including the ability to create, update, delete, and display product information. This application demonstrates the implementation of CRUD (Create, Read, Update, Delete) functionality using the **Spring Boot** framework. Originally, the plan was to deploy the application using Docker, but for this iteration, I chose to use an **H2 in-memory database** instead. H2 is a lightweight, in-memory database ideal for development and testing, simplifying the setup and making it easier to manage for smaller projects.

To deepen my understanding of both backend development and templating, I incorporated **Thymeleaf** as the templating engine. Thymeleaf is a powerful tool for rendering dynamic content on the frontend, enhancing the user interface and making it more interactive. The combination of **H2** and **Thymeleaf** allowed me to integrate backend and frontend functionality seamlessly, improving my skills in both database management and frontend/backend interaction.

Additionally, **AngularJS** was integrated into the frontend to provide a dynamic and engaging user experience. Specifically, AngularJS is used to display a welcoming message to users, making the application feel more interactive and user-friendly. The combination of these technologies has resulted in a smooth, responsive product management system.

The application is accessible through a clean and user-friendly web interface.



It is divided into several key sections:

* **Product Management Interface** (http://localhost:8080/products/html): This page provides the CRUD operations for managing products.
* **Product JSON Data** (http://localhost:8080/products): This endpoint shows the product data in raw JSON format for easy viewing.
* **Welcome Message:** Displayed dynamically on the homepage via AngularJS.

The application is designed with simplicity in mind, making it ideal for small-scale product management systems. By interacting with the Product Management System, users can:

* **Add new products** by specifying the name and price.
* **Update existing products** to modify their details.
* **Delete products** that are no longer needed.

The intuitive design ensures that users can perform these operations with minimal effort, making it an essential tool for any small-scale product management workflow.

**2. Technical Documentation**

**2.1 System Overview**

The **Product Management Application** consists of two main components:

* **Frontend**: The user interface that displays product information and forms for managing products.
* **Backend**: The server-side logic handling data storage and CRUD operations via Spring Boot.

**Frontend Technologies:**

**A screenshot of a computer

Description automatically generated**

* **HTML**: Used to structure the content and layout of the web pages.
* **Thymeleaf**: A Java templating engine that renders dynamic content on the frontend, connecting to Spring Boot models and providing user-specific data.
* **Bootstrap**: A responsive framework to ensure that the application is mobile-friendly and visually consistent.
* **AngularJS**: A JavaScript framework used to enhance the user interface with dynamic data binding. Specifically, AngularJS is used to display a welcome message on the homepage.
  + The script provided initializes a simple welcome message and ensures that the application dynamically welcomes users to the silent auction.

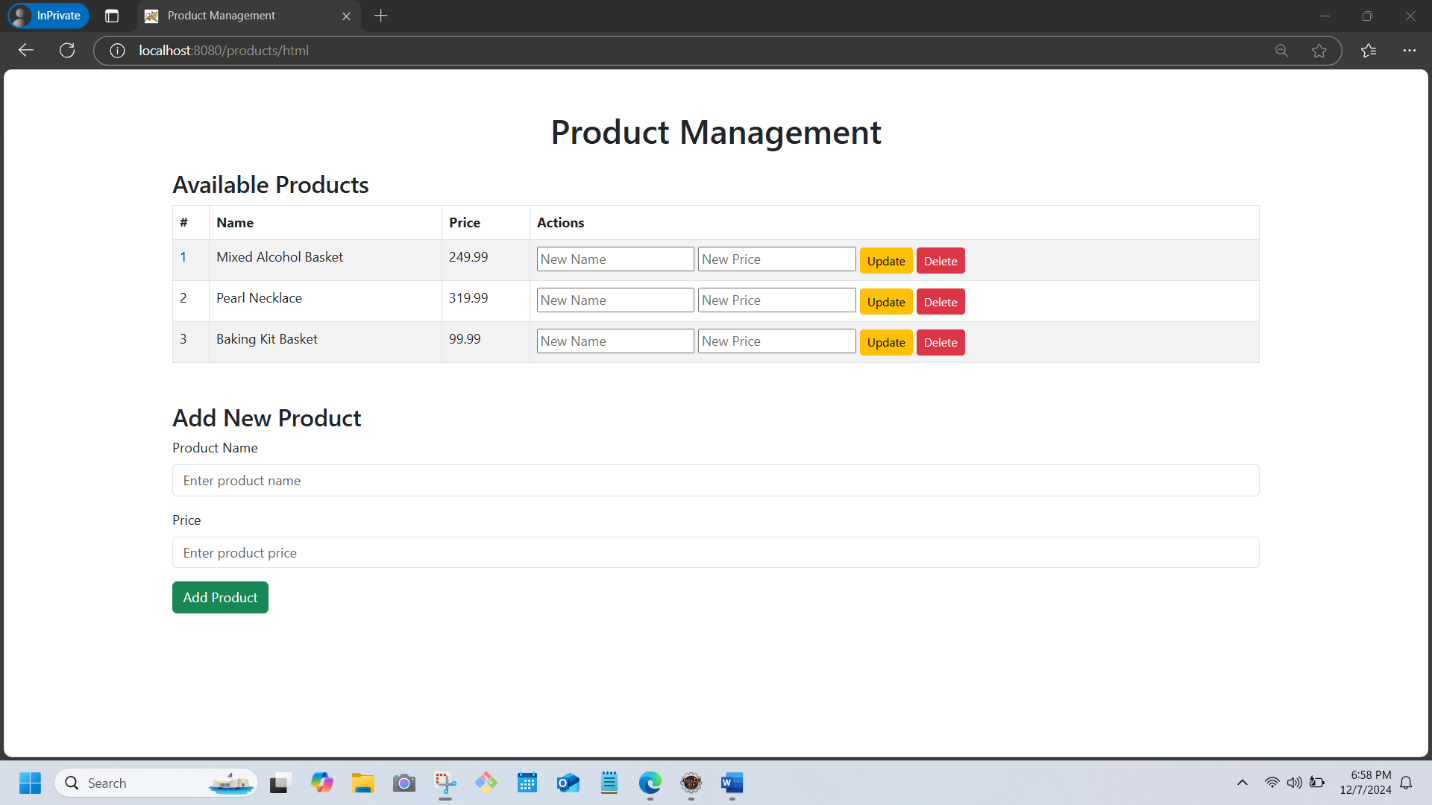
**Backend Technologies:**

* **Spring Boot**: A Java framework used to build the backend services that manage the product data, including performing CRUD operations.
* **Spring MVC**: Handles the HTTP requests for the application and routes them to appropriate controllers.
* **In-Memory Database (H2 or similar)**: Stores product information temporarily for testing purposes. For production, it could be replaced with a more permanent database solution.

**2.2 Application Flow**

The application allows users to interact with products via a table displaying all available products and forms to add, update, or delete products. Here's the flow of operations:

1. **Display Product List**:
   * Upon visiting the application, the user is presented with a table of products, each including details like the product name, price, and action buttons (Update, Delete).
2. **Add a Product**:



* + The user fills in a form with the product name and price and submits it.
  + A POST request is sent to the /products endpoint to add the new product to the database.
  + The page reloads to display the newly added product.

1. **Update a Product**:

A screenshot of a computer

Description automatically generated

* + The user can update a product by entering new details (name, price) in the corresponding fields for that product and clicking the Update button.
  + A POST request with a hidden \_method field set to "PUT" is sent to the /products/{id} endpoint, where {id} is the ID of the product to update.
  + After processing, the page reloads to show the updated product list.

1. **Delete a Product**:

A screenshot of a computer

Description automatically generated

* + The user can remove a product by clicking the Delete button next to the product.
  + A POST request with a hidden \_method field set to "DELETE" is sent to the /products/{id} endpoint.
  + After the product is deleted, the page reloads to reflect the change.

**2.3 AngularJS Integration**

In addition to Spring Boot and Thymeleaf, **AngularJS** has been used to enhance the user experience. Specifically, the **script.js** file includes an AngularJS module (module1) and a controller (myController) that manages a dynamic welcome message for users:

A purple screen with white text

Description automatically generated

* **Module and Controller**: The AngularJS module module1 is defined, and the controller myController is used to set the welcomeMessage on the frontend. This message will be displayed dynamically when users access the homepage.
* **Dynamic Data Binding**: AngularJS ensures that the welcome message is automatically updated in the user interface whenever the value of $scope.welcomeMessage changes, providing a dynamic and responsive user experience.

**2.4 Data Flow Recap**

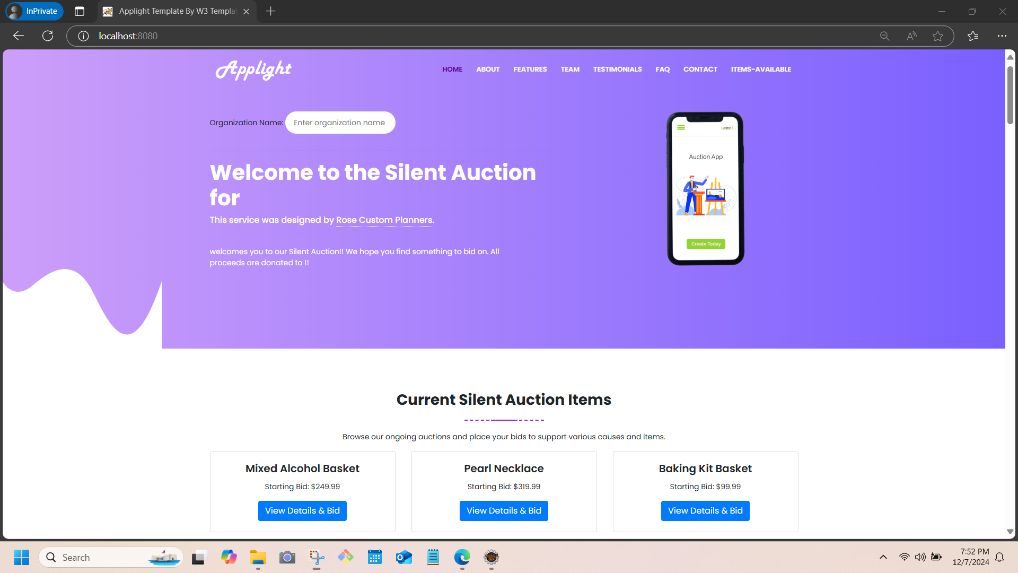
* **Frontend**: The user interface interacts with the backend through HTTP requests to manage products. This includes sending requests for adding, updating, and deleting products.
* **Backend**: The backend processes the received requests, interacts with the database (H2 in this case), and sends a response to the frontend with the updated product list.
* **Page Reloading**: After each CRUD operation (add, update, delete), the page reloads automatically, ensuring the user sees the most up-to-date product data.
* **AngularJS Integration**: The dynamic welcome message is rendered using AngularJS, providing a more interactive and personalized user experience.

**3. User Documentation**

**3.1 Getting Started**

To access the Product Management Application, navigate to the application’s website at:

* **Website URL**: <http://localhost:8080>



* + This is the main entry point where you can interact with the application and see the available product management features.
* **Product JSON Data**: http://localhost:8080/products

A screenshot of a computer

Description automatically generated

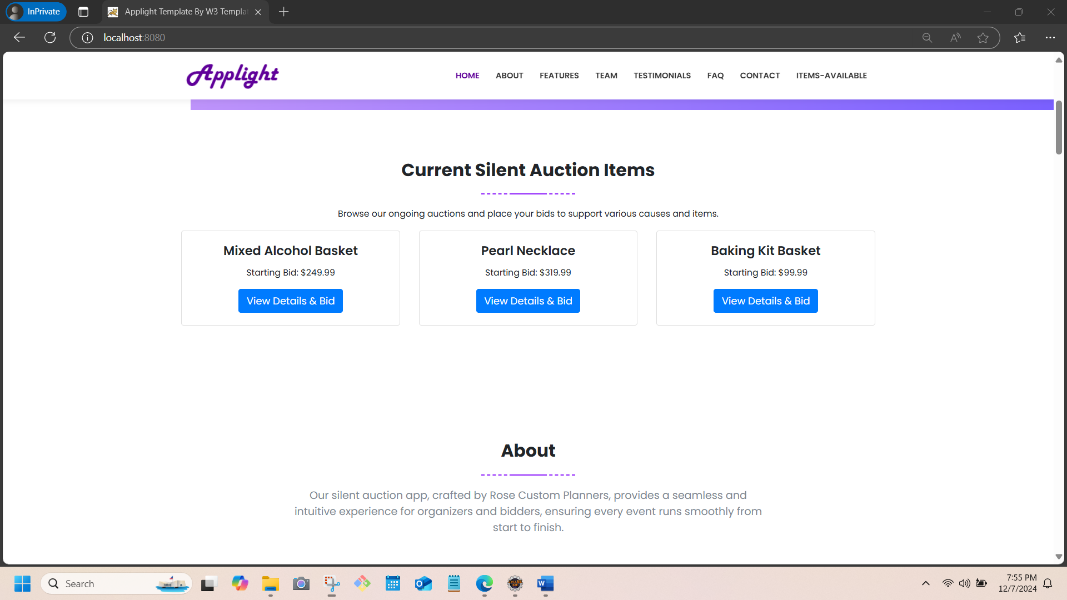
* + This endpoint displays product data in JSON format, allowing you to view the raw product information stored in the database.
* **CRUD Interface**: http://localhost:8080/products/html

A screenshot of a computer

Description automatically generated

* + This is the page where you can interact with the products through a user-friendly interface that allows you to create, read, update, and delete products. It is the main interface for managing your product listings.

**3.2 Viewing Products**

****

**Product Table**: Upon accessing the **CRUD Interface** (http://localhost:8080/products/html), you will see a table displaying a list of existing products. Each row in the table contains:

**A screenshot of a computer

Description automatically generated**

* + **Product ID:** A unique identifier for each product.
  + **Product Name:** The name of the product.
  + **Product Price:** The price of the product.
  + **Actions**: Buttons to update or delete each product.

**3.3 Adding a Product**

To add a new product:

1. **Navigate to the "Add Product" Form**:

A screenshot of a product

Description automatically generated

* + This form is located below the product table and includes fields for entering a product’s name and price.

1. **Fill in the form**:
   * Enter the product name and price in the respective fields.
   * Click the "Add Product" button to submit the form.
2. **Submission**:
   * The application will send a POST request to the /products endpoint with the provided information.
   * After the form is submitted, the page will reload and display the newly added product in the product table.

**3.4 Updating a Product**

To update an existing product:

1. **Navigate to the product row you wish to update**:

A screenshot of a computer

Description automatically generated

* + Each product has an "Update" button associated with it.

1. **Fill in the new details**:
   * In the corresponding update form, enter the new product name and/or price.
2. **Submit the updated information**:
   * Click the "Update" button to submit the updated data.
   * The form sends a POST request with the \_method hidden field set to "PUT", targeting the /products/{id} endpoint where {id} is the ID of the product you wish to update.
   * After submission, the page will reload, and you will see the updated product list.

**3.5 Deleting a Product**

To delete a product:

1. **Locate the product row**:
   * Each product row contains a "Delete" button.
2. **Click the "Delete" button**:

A screenshot of a computer screen

Description automatically generated

* + Clicking this button sends a POST request with the \_method hidden field set to "DELETE" to the /products/{id} endpoint.

1. **Confirm Deletion**:

A screenshot of a computer

Description automatically generated

* + Once the request is processed, the product will be removed from the database, and the page will reload to reflect the updated list, no longer displaying the deleted product.

**3.6 AngularJS Welcome Message**

When you first visit the application, you will see a dynamic welcome message that introduces you to the product management system. The message is generated by **AngularJS** and looks like this:

"Welcome to our Silent Auction for {Angular JS Code}”

A purple screen with white text

Description automatically generated

This message is displayed as soon as you land on the **CRUD Interface** (http://localhost:8080/products/html) and is managed by AngularJS using the script from script.js.

**3.7 Product Management Page Features**

* **Product Table**:
  + The product table displays all products with their details, such as the product name, price, and associated actions for updating or deleting.
  + Each row in the table includes forms for updating and deleting products.
* **Add Product Form**:
  + The "Add Product" form is located at the bottom of the product table. It allows users to input a product name and price and submit them to the backend for creation.
* **Update Product Form**:
  + Every product listed in the table has an update form where users can modify the product name and price. When submitted, the updated product is sent to the backend for processing and updating in the database.
* **Delete Product Form**:
  + Each product also has a delete form. Clicking the delete button removes the product from the system. Afterward, the page refreshes to show the updated product list without the deleted item.