**Functionality Description**

**For**

**[**[**ay2324s2-capstone-assignment-khai-chuen-jeremy-yan-yi**](https://github.com/SP-CET-Capstone/ay2324s2-capstone-assignment-khai-chuen-jeremy-yan-yi)**]**

**Date: [09/02/24]**

Contents

[1. Document Management 3](#_Toc158402281)

[1.1 Contributors 3](#_Toc158402282)

[1.2 Version Control 3](#_Toc158402283)

[2. Overview 4](#_Toc158402284)

[2.1 Functionality Overview 4](#_Toc158402285)

[2.2 Full Stack Communication Architecture 6](#_Toc158402286)

[3.1 Function Description 7](#_Toc158402287)

[3.1.1 Login Feature 7](#_Toc158402288)

[3.1.2 Register Feature 9](#_Toc158402289)

[3.1.3 Add Entity Feature 11](#_Toc158402290)

[3.1.4 List Entity Feature 16](#_Toc158402291)

[3.1.5 Delete Entity Feature 19](#_Toc158402292)

[3.1.6 Update Entity Feature 23](#_Toc158402293)

[3.1.7 Display of Entity History Pre-Update Feature 27](#_Toc158402294)

[3.2 Technology 28](#_Toc158402295)

[3.3 Development Tools and Processes 30](#_Toc158402296)

[3.4 Interfaces and services 31](#_Toc158402297)

# Document Management

## 1.1 Contributors

|  |  |
| --- | --- |
| **Function Integrated** | **Name** |
| Registration | LEE KHAI CHUEN |
| Login | LEE KHAI CHUEN |
| List travel Listing | HO YAN YI |
| Delete travel Listing | HO YAN YI |
| Add travel Listing | JEREMY LONG SHI WEI |
| Update travel Listing | JEREMY LONG SHI WEI |
| Display Entity History Pre-Update | JEREMY LONG SHI WEI |

## 1.2 Version Control

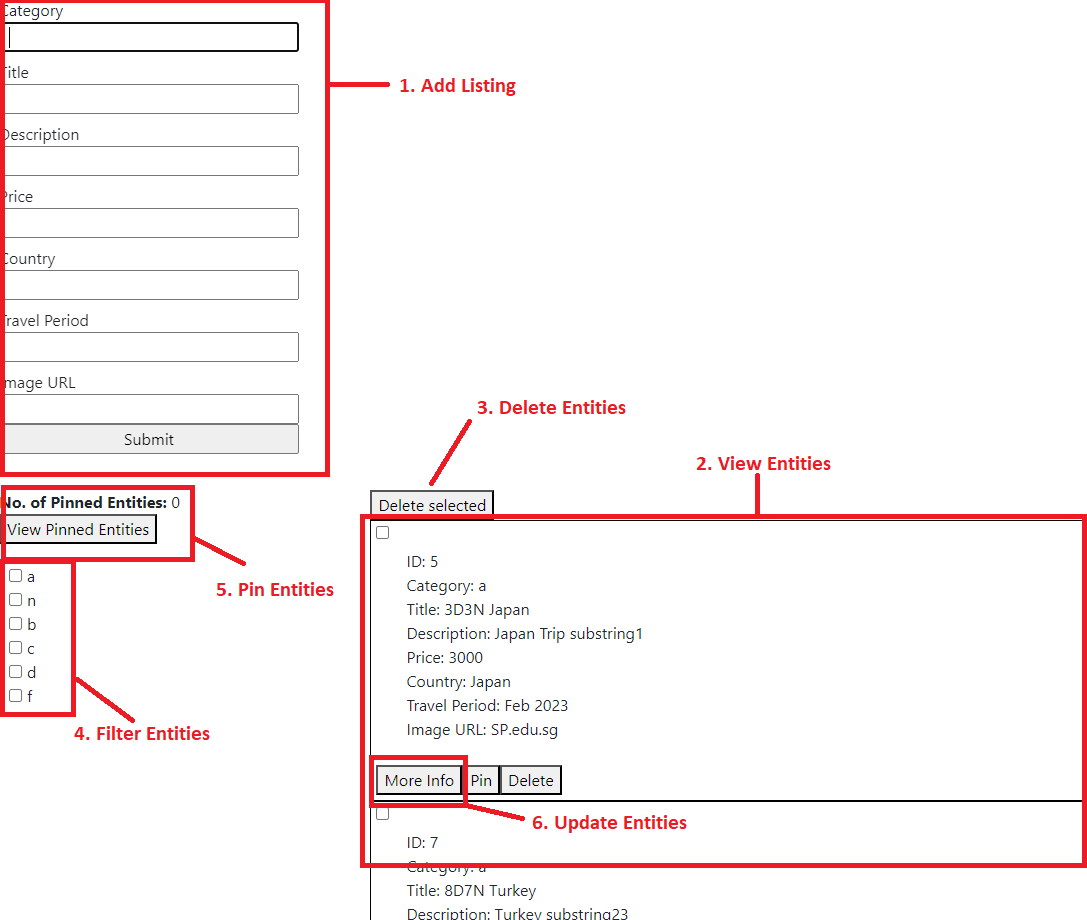
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date** | **Version** | **Author** | **Section** | **Amendment** |
| 08/02/24 | 1.0 | - | - | Initial Creation |
| 09/02/24 | 1.1 | Jeremy | 2.1, 2.2, 3,2, 3.3 | Initial Draft |
| 09/02/24 | 1.2 | Jeremy | 3.1.3, 3.1.6 | SSDD for Add and Update Entity Feature |
| 09/02/24 | 1.3 | Jeremy | 3.1.7 | SSDD for Display of Update Archives |
| 09/02/24 | 1.4 | Khai Chuen | 3.1.1, 3.1.2, 3.3 | Login and Register Feature  Add Point 4 |
| 09/02/24 | 1.5 | Ho Yan Yi | 3.1.1, 3.1.2, 3.4 | Listing and Delete Feature  Interfaces and Services |
|  |  |  |  |  |
|  |  |  |  |  |

# Overview

## 2.1 Functionality Overview

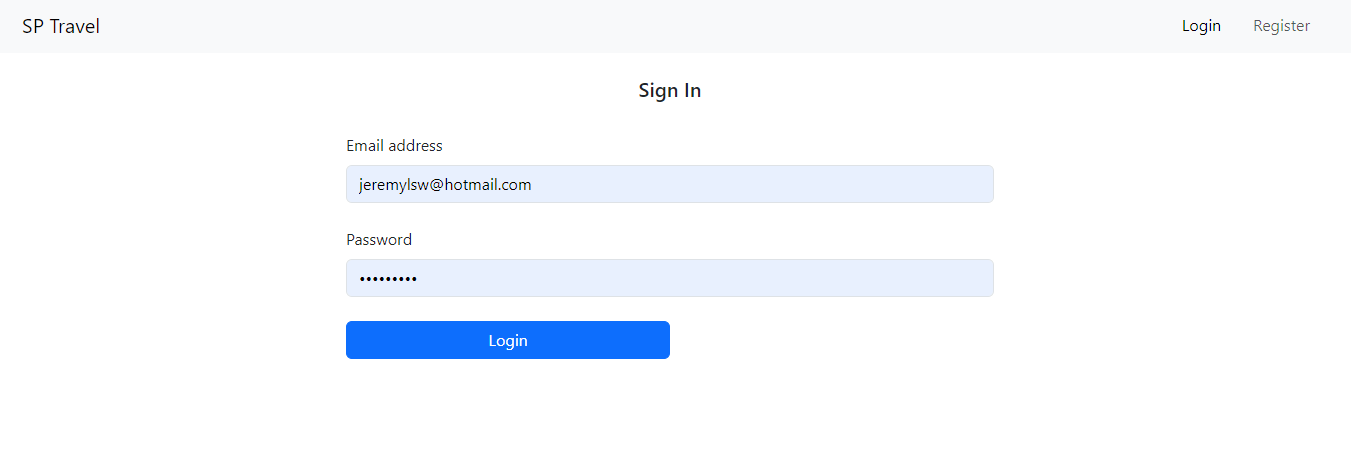
[[ay2324s2-capstone-assignment-khai-chuen-jeremy-yan-yi](https://github.com/SP-CET-Capstone/ay2324s2-capstone-assignment-khai-chuen-jeremy-yan-yi)] is a travel listing website where companies are able to keep track of the travel listings they offer. From this website, users are able to execute one of any of the following tasks:

1. Add a travel listing
2. View existing travel listings
3. Delete (individual / in bulk) unwanted travel listings
4. View entities filtered by their categories
5. Pin favourite entities
6. Update existing entities



***Figure 1: Snapshot of Front-End Listing Page***

To use the website’s services, users have to be a registered user and first login from the login page.



***Figure 2: Snapshot of Front-End Login Page***

Upon logging in, the users will then be able to access additional resources such as viewing pinned favourites



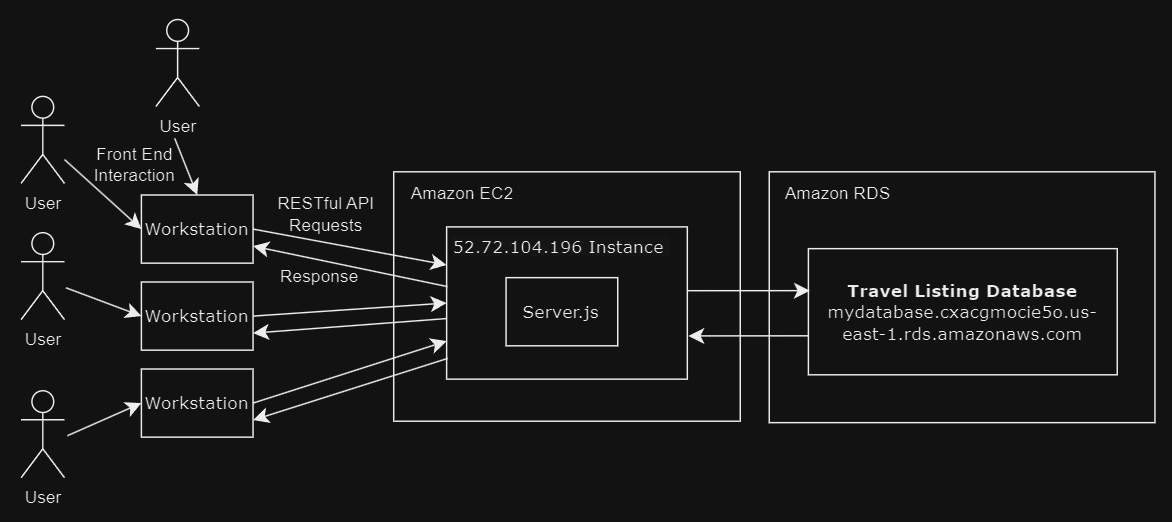
***Figure 3: Snapshot of Front End Pinned Page***

## 2.2 Full Stack Communication Architecture

The user first interacts with the front-end website using the implemented functionalities described in section 2.1. The front-end website will then send HTTP requests to a server hosted on an Amazon EC2 instance that is configured with a fixed elastic IP.

If the request requires modifications or retrieval of data, it will communicate with the database hosted on Amazon RDS to make the necessary changes or retrieval.

The following diagram shows the components involved in the full stack integration between the front and back end and how the user interacts with the backend database using the front-end webpages.



***Figure 4: State Diagram of full stack communication***

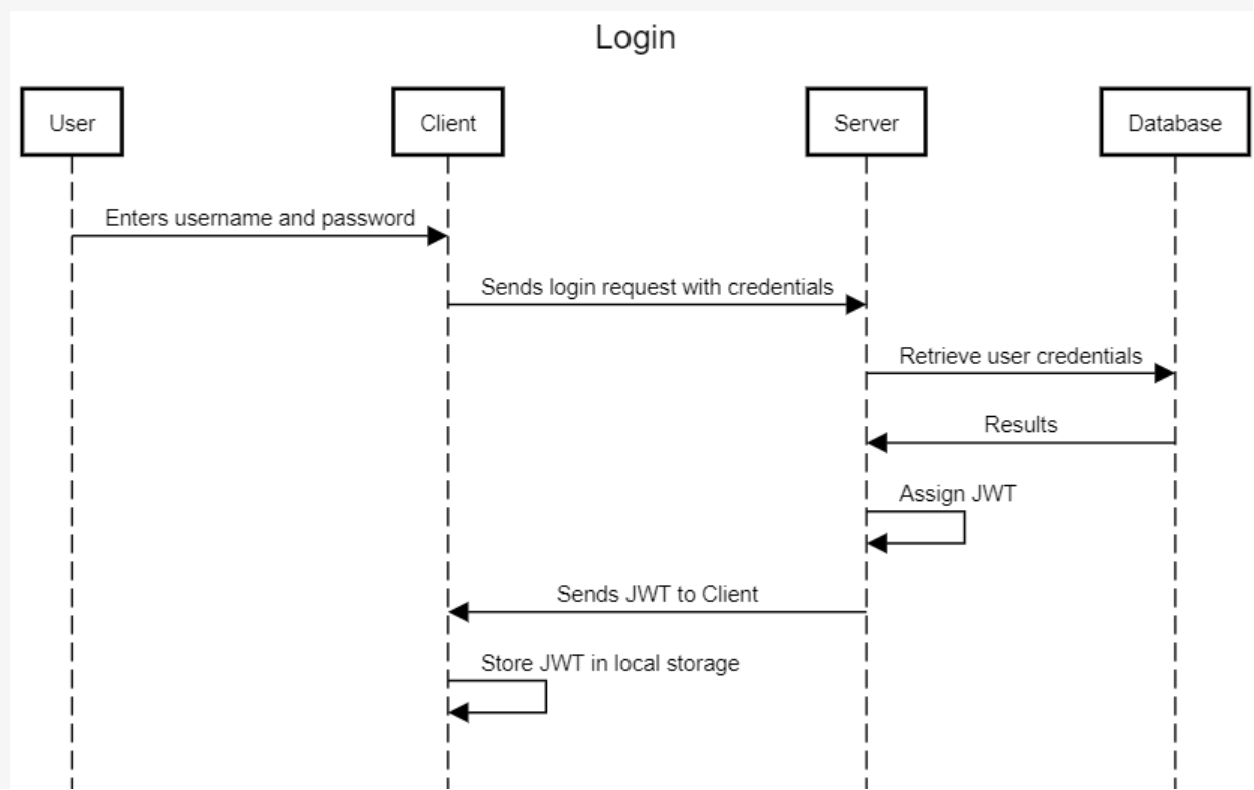
# Software System Design Description

## 3.1 Function Description

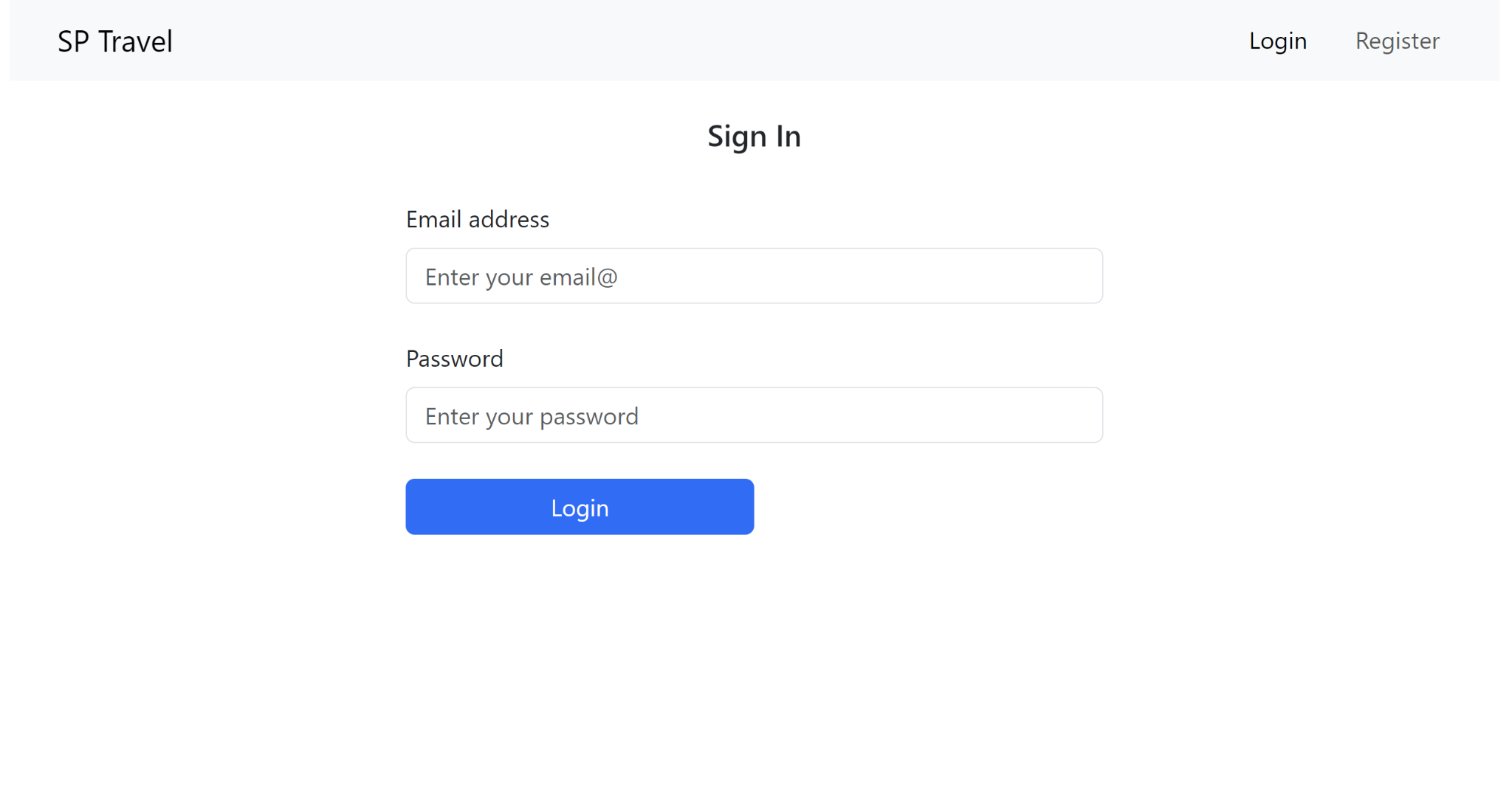
### 3.1.1 Login Feature

**Sequence Diagram**

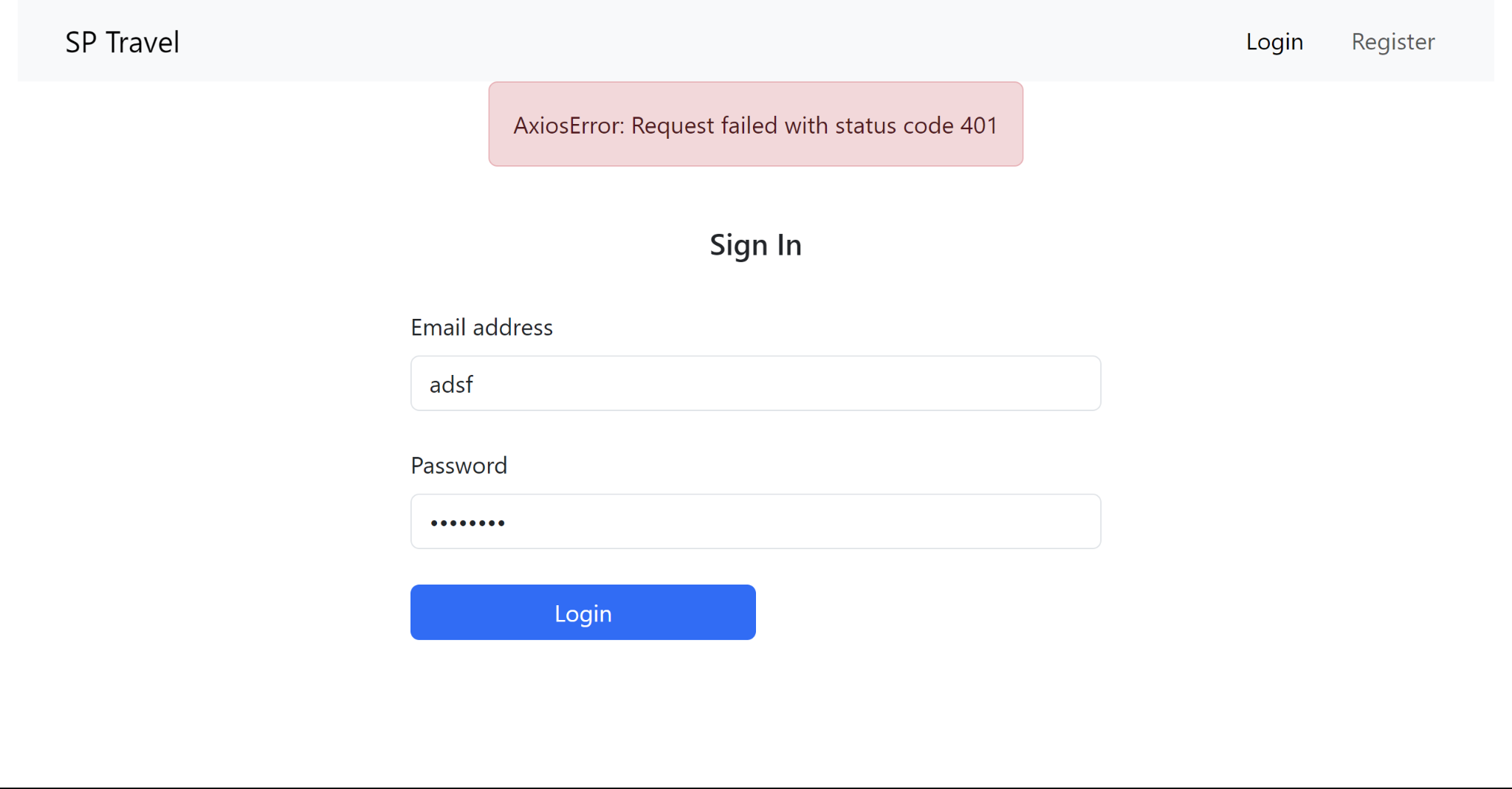
The login process begins when the user enters their username and password into the Client application. This information is then sent as a login request to the Server. The Server interacts with the Database to retrieve the user's credentials and receives a response indicating the success or failure of the operation. If the credentials are valid, the Server generates a JSON Web Token (JWT) and sends it back to the Client. The Client securely stores the JWT in its local storage for future use in authentication and authorization. This sequence ensures a secure and seamless login experience for the user, with data handling and security measures implemented at each step.



**End-to-end Screenshot**



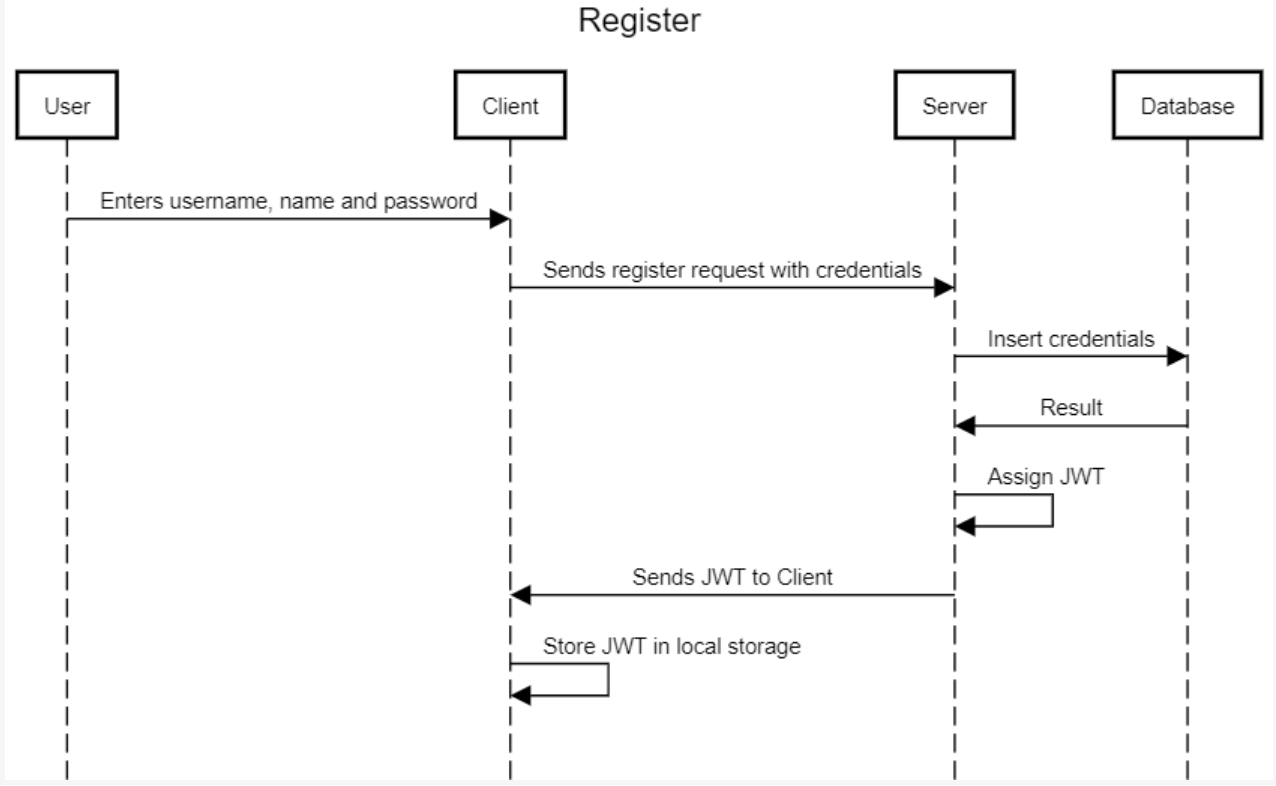
Error alert pops up to tell user if login failed



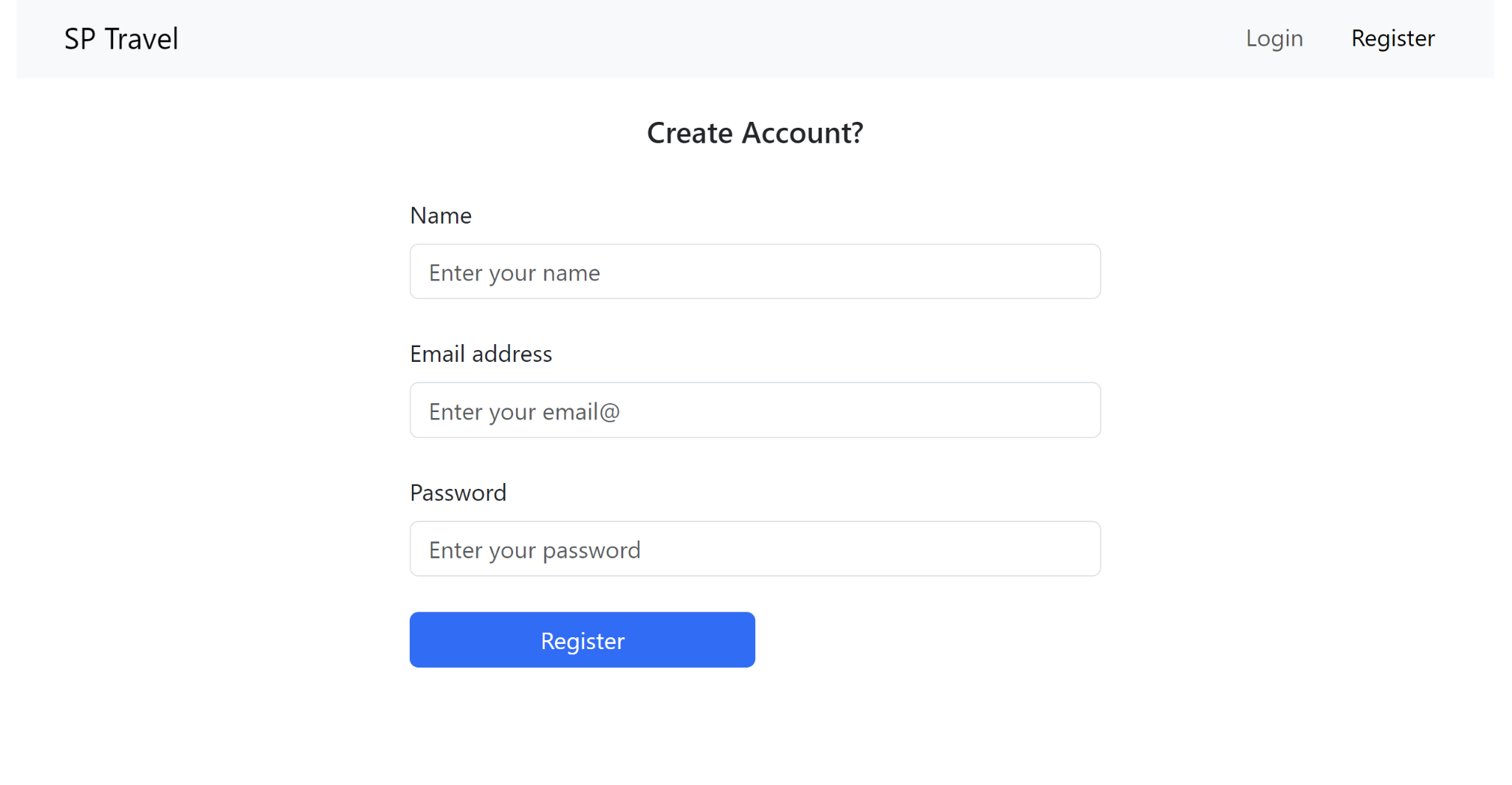
### 3.1.2 Register Feature

**Sequence Diagram**

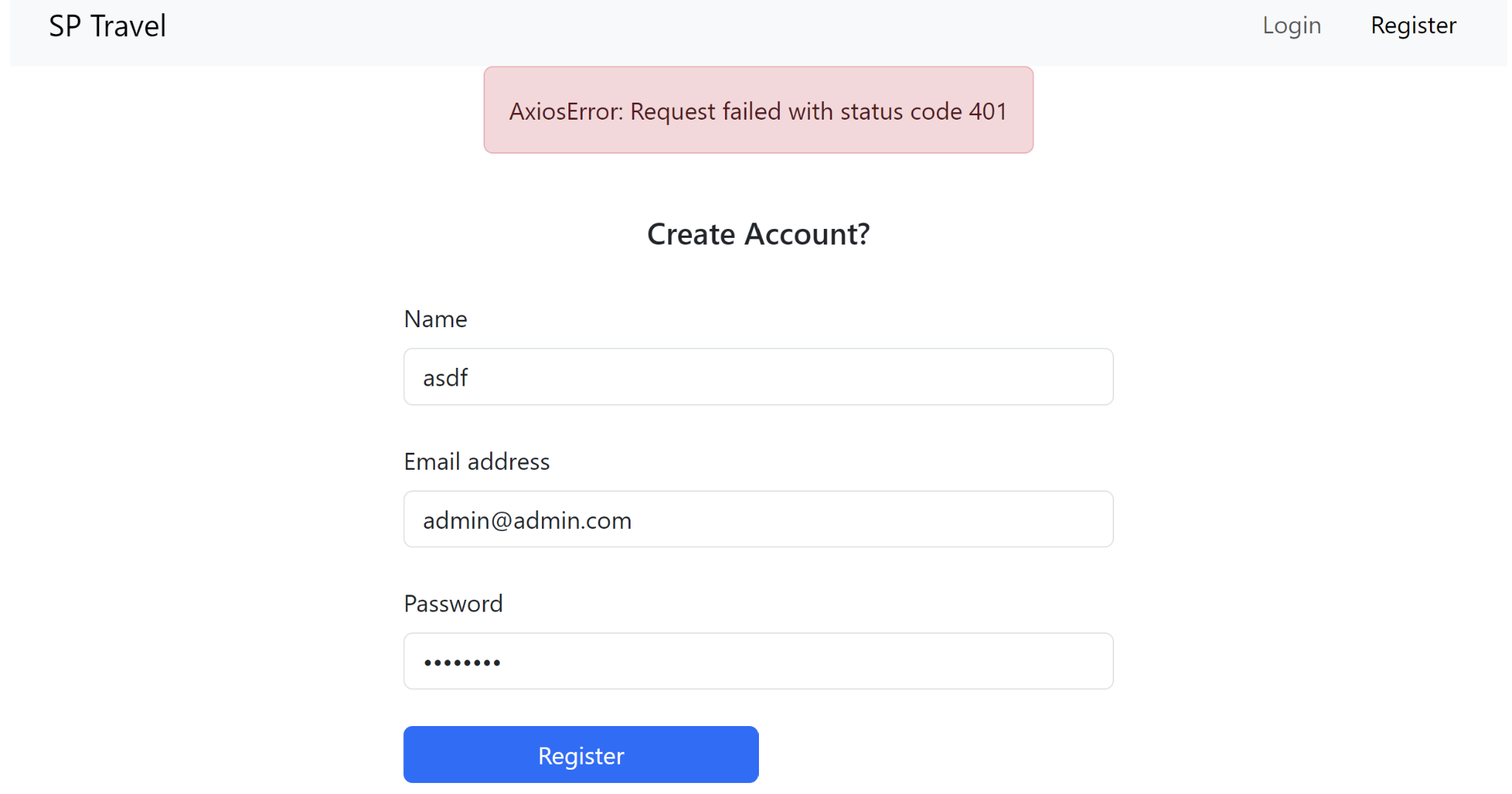
The process begins with the user entering their username, name, and password into the Client application. This information is then sent as a registration request to the Server. The Server interacts with the Database to insert the user's credentials and receives a response indicating the success or failure of the operation. Upon successful insertion, the Server proceeds to assign a JSON Web Token (JWT) to the user. This token is then sent back to the Client, which securely stores it in its local storage for future authentication and authorization purposes. This entire sequence ensures a seamless registration process for the user, with necessary data handling and security measures in place at each step.



**End-to-end Screenshot**



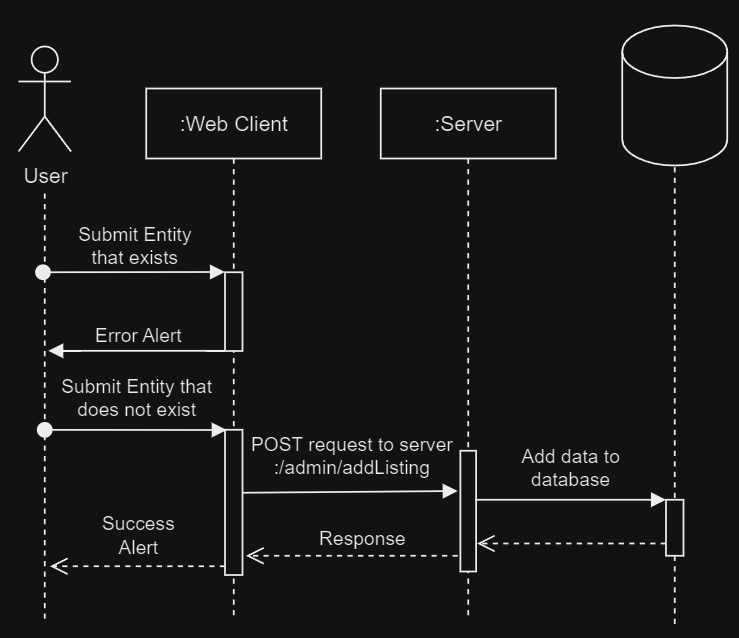
Error alert pops up to tell user if register failed



### 3.1.3 Add Entity Feature

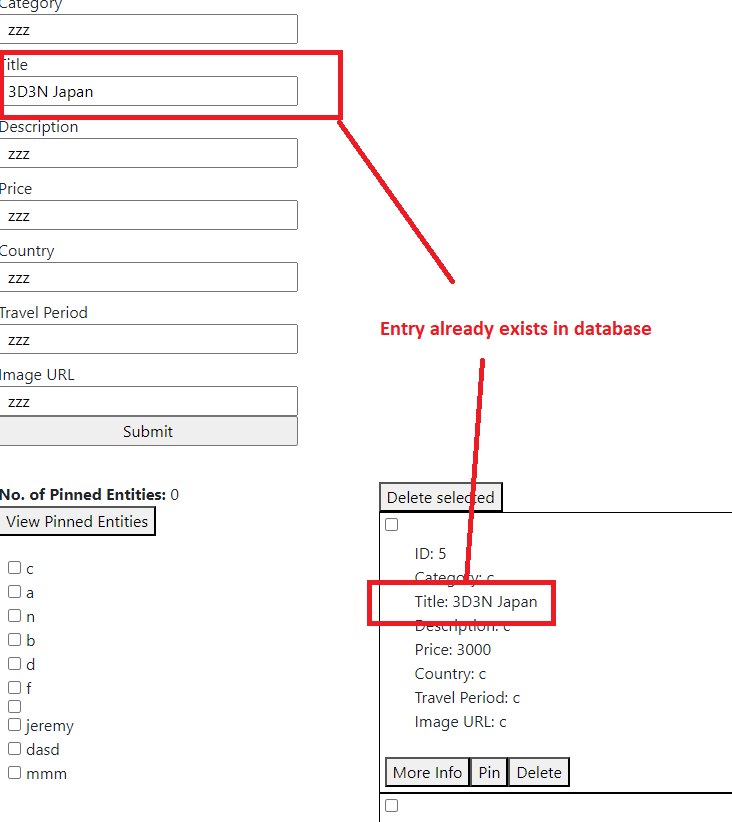
**Sequence Diagram**

If the user submits a request to add an entity that already exists, the web client will return an alert that tells the user that the entry exists in the database. If the entry does not already exist in the database, the data will be pushed to the backend database and the user will receive a alert indicating a successful transaction.

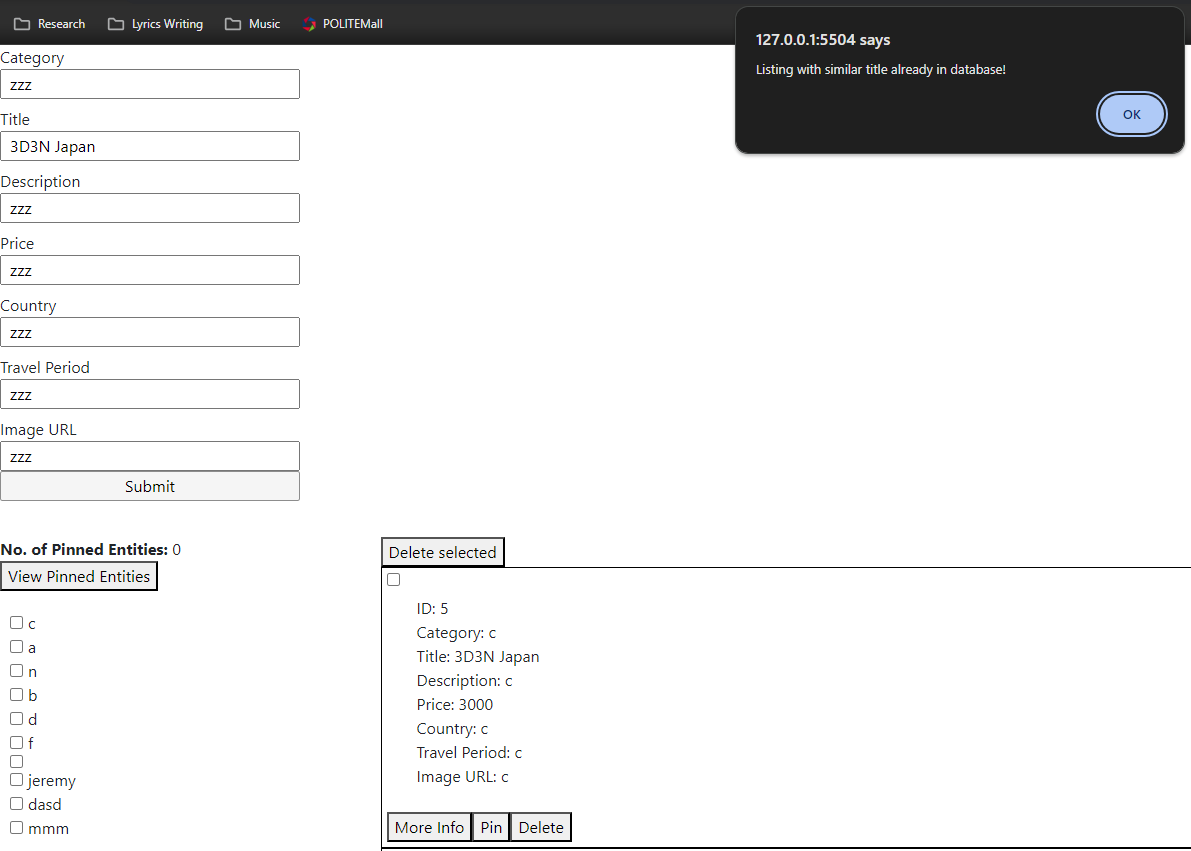


**End-to-end Screenshot**

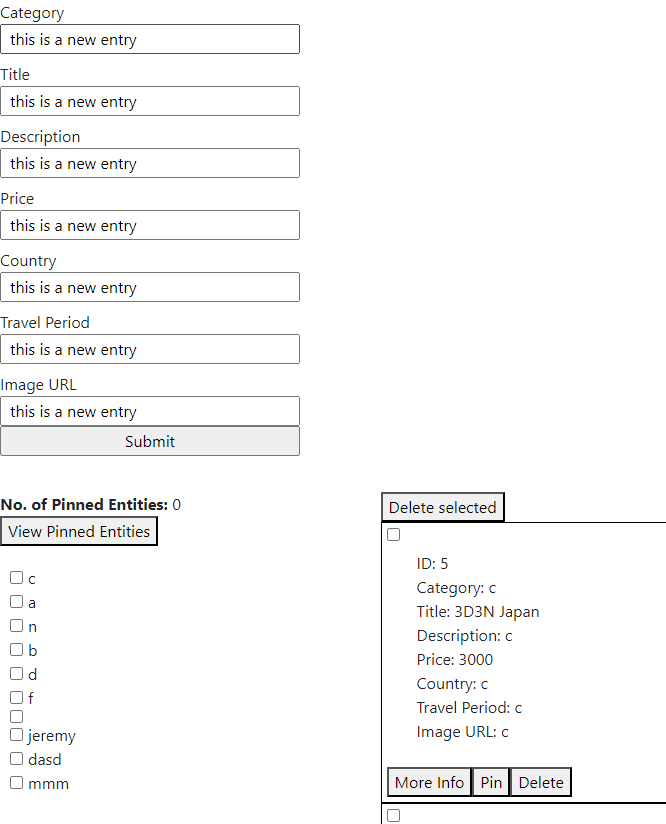
Assume user has already logged in, but tries to submit an entry that already exists



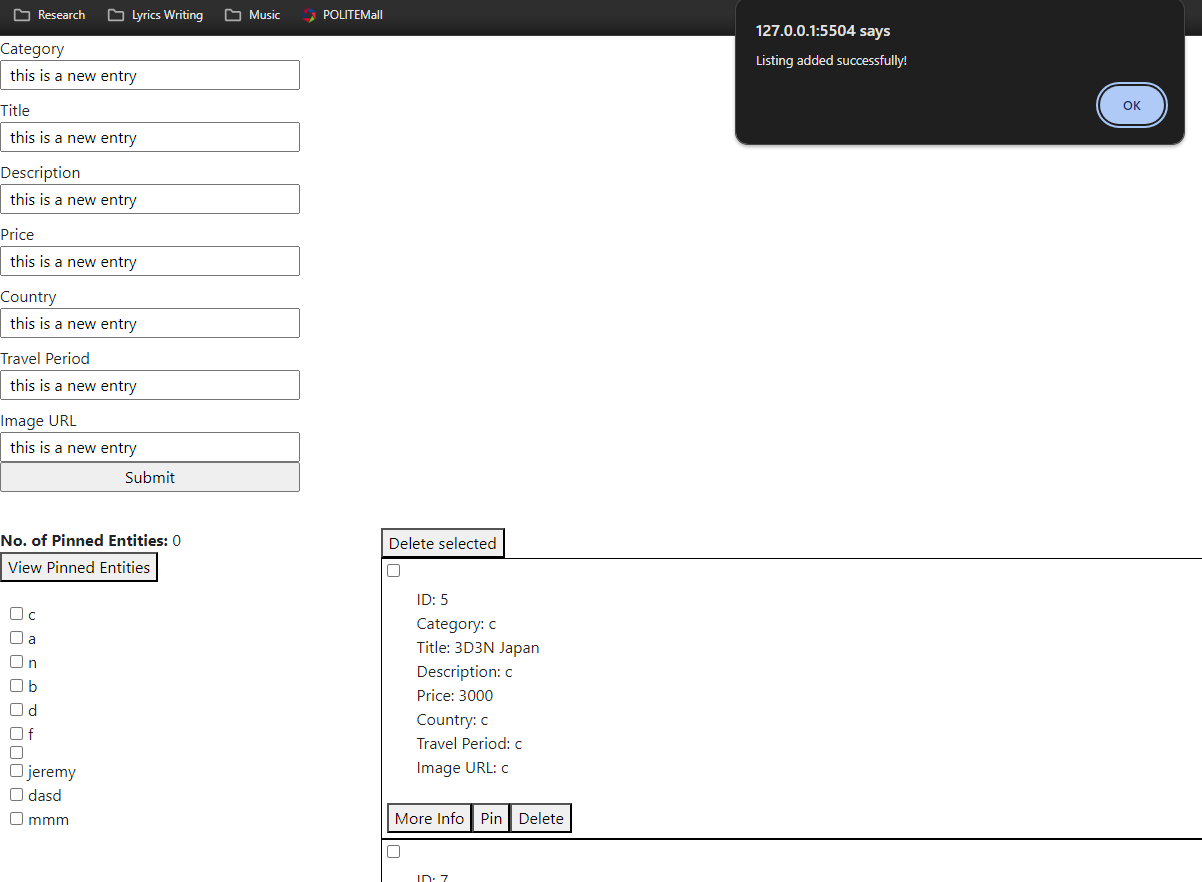
Error alert pops up to tell user that entry already exists in database



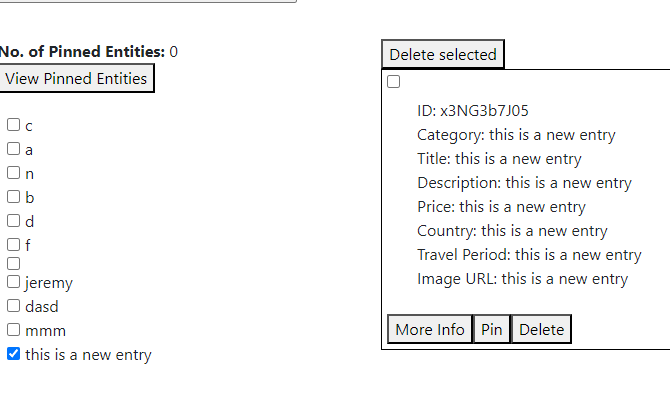
If user submits a new entry not in database,



Success alert will pop up

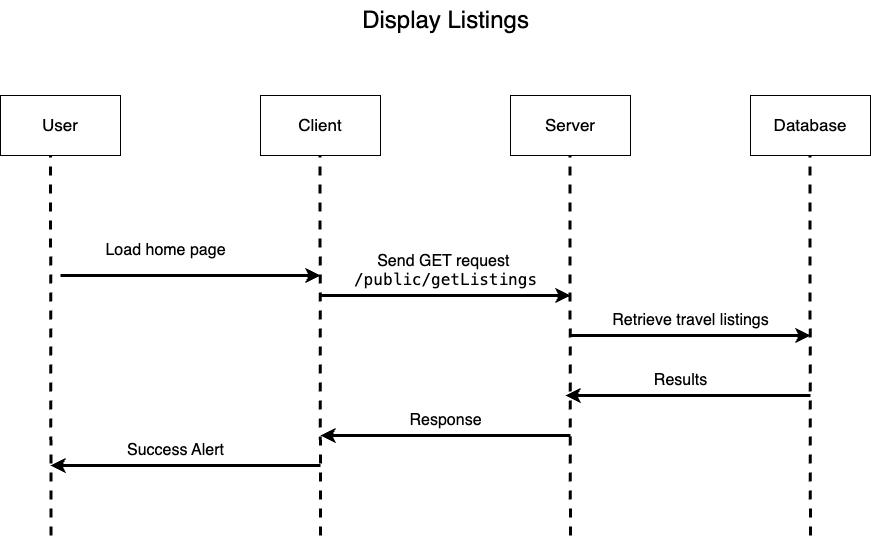


Upon clicking ok, new entry will be reflected on webpage. Filter can be used to filter out other entities for easy viewing.

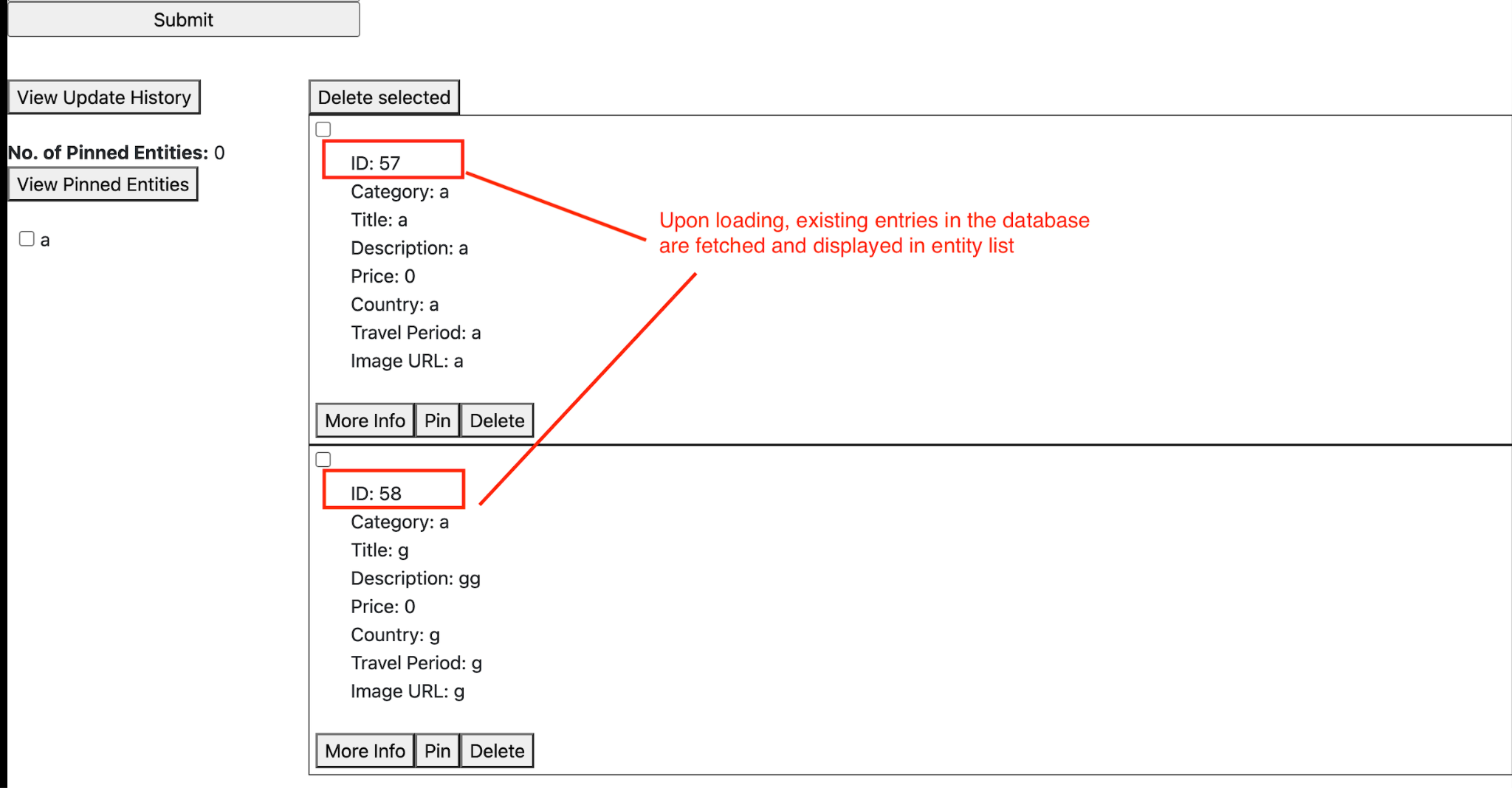


### 3.1.4 List Entity Feature

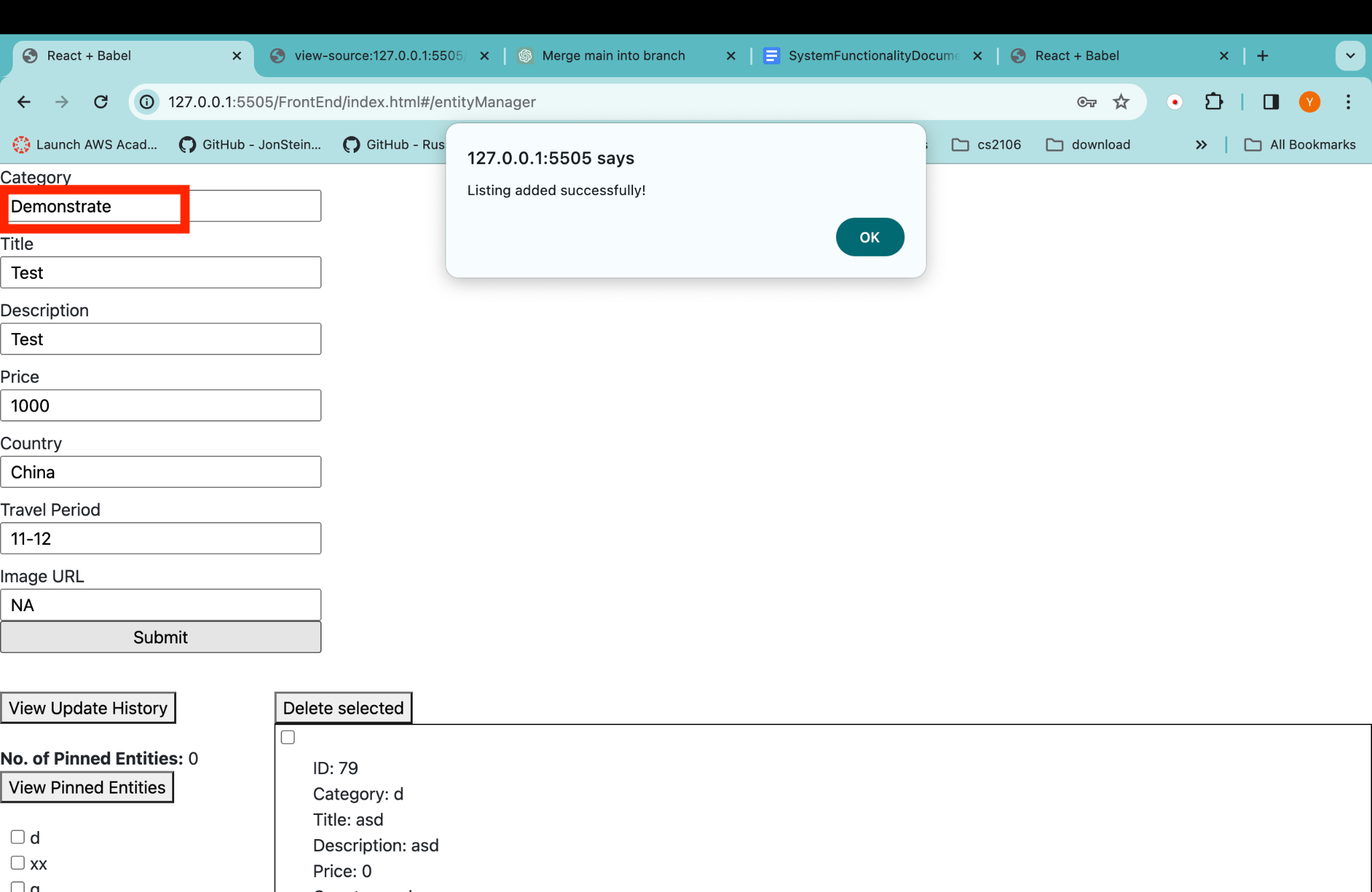
**Sequence Diagram**

The process starts when the user loads the homepage. The list component will send a get request to the backend server to retrieve all existing listings and render it on the homepage. 

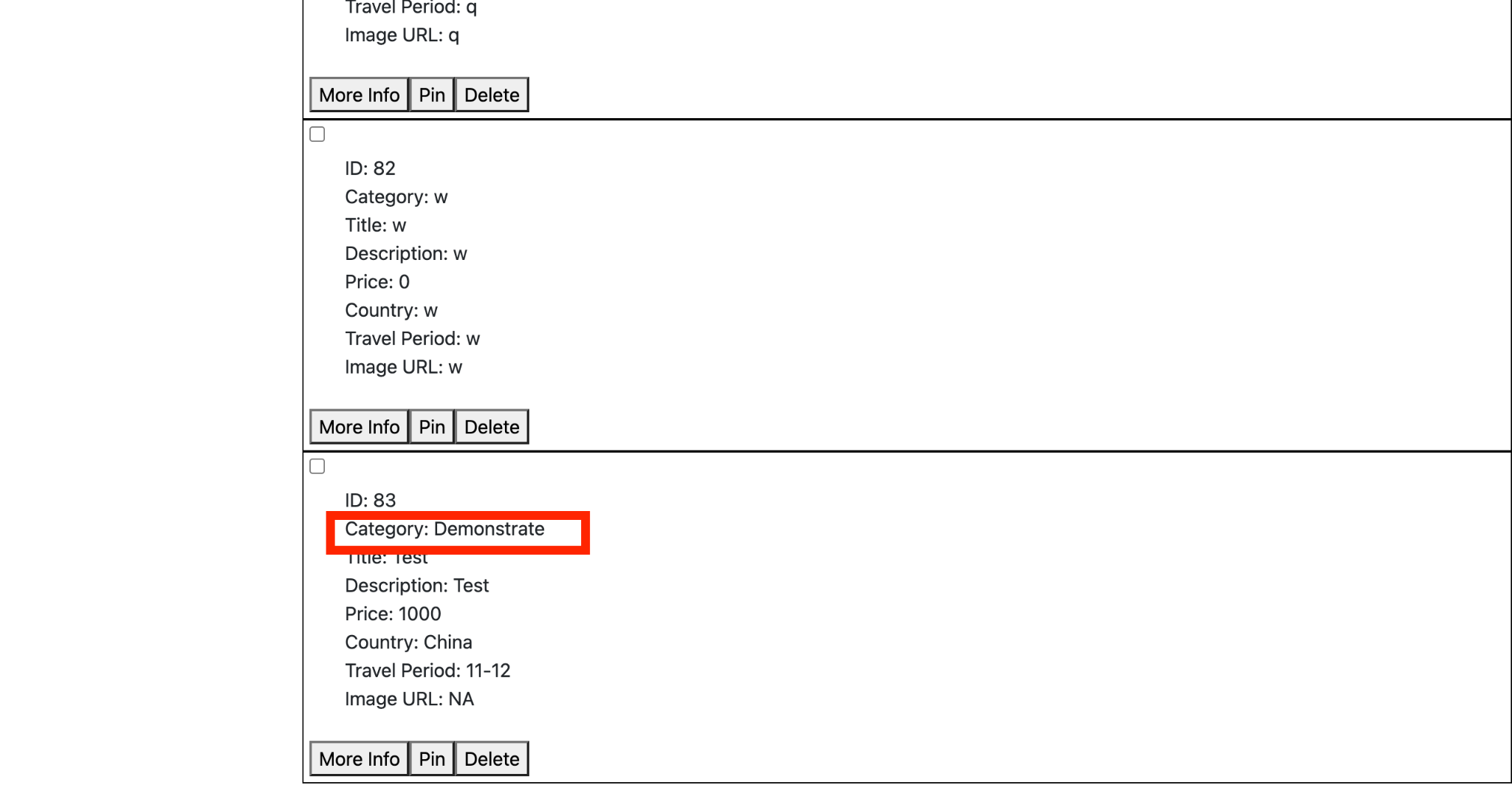
**End-to-end Screenshot**

Upon loading the homepage, all existing travel listing entries in the database are fetched and displayed in a list.

After adding a new entry, the new travel listing will be displayed in the list component.



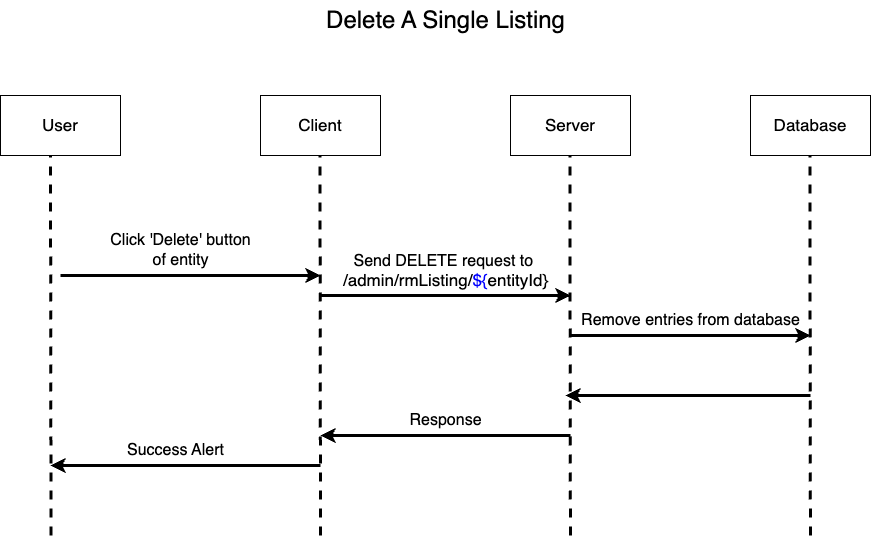
The updated list component containing the newly added entry is shown below



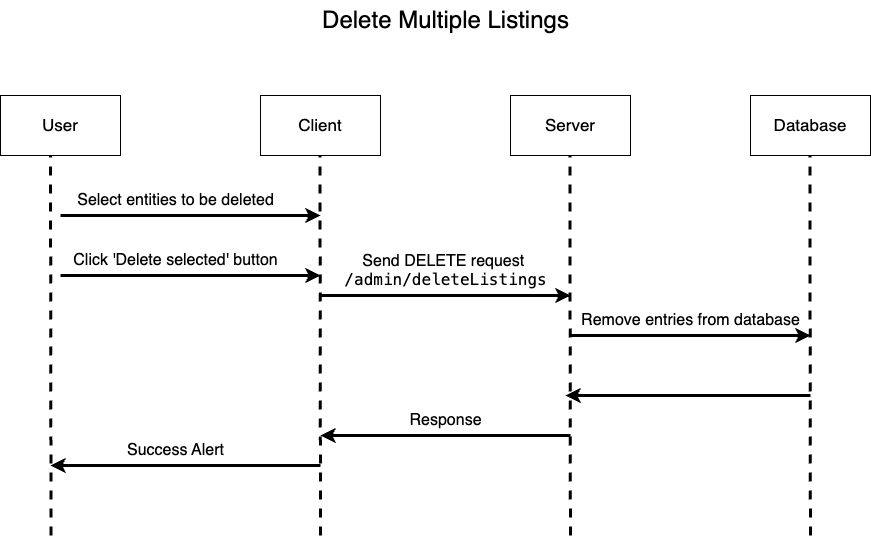
### 3.1.5 Delete Entity Feature

**Sequence Diagram**

The user can click on the ‘Delete’ button of any listing to delete the entity from the database and remove it from the homepage. Upon clicking the delete button, a delete request is sent to backend server along with the user’s JSON Web Token. If the token is verified to be valid and removal of the entry from the database is successful, the user will receive a success alert and the deleted entity will no longer displayed in the list component of the front end.

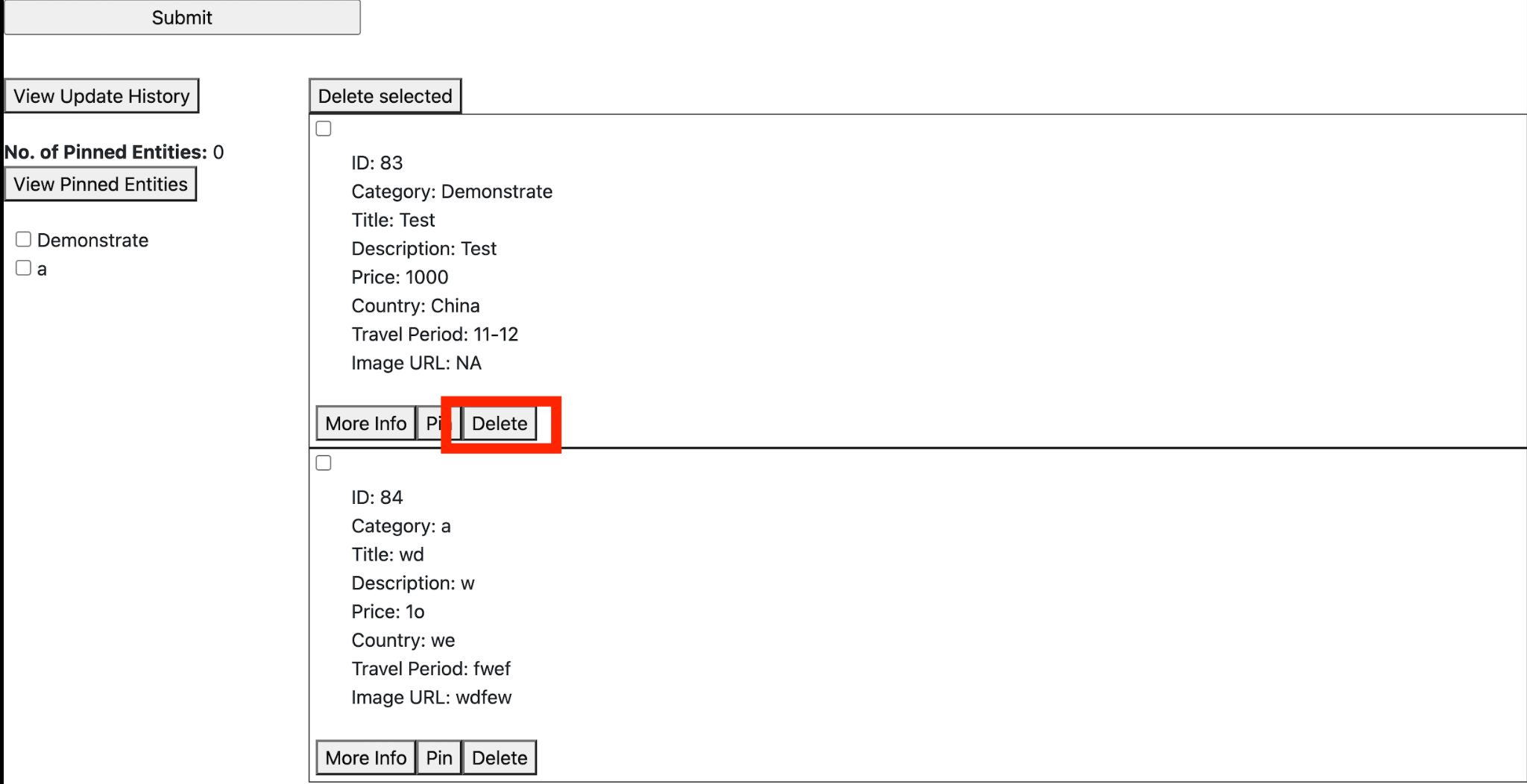
****

The user can click select multiple listings to be deleted by selecting their respective checkboxes and clicking the ‘Delete selected’ button. Upon clicking the button, a delete request is sent to backend server along with the user’s JSON Web Token and array of selected entity IDs. If the token is verified to be valid and all selected entities are successfully remove from the database, the user will receive a success alert and the deleted entities will no longer displayed in the list component of the front end.

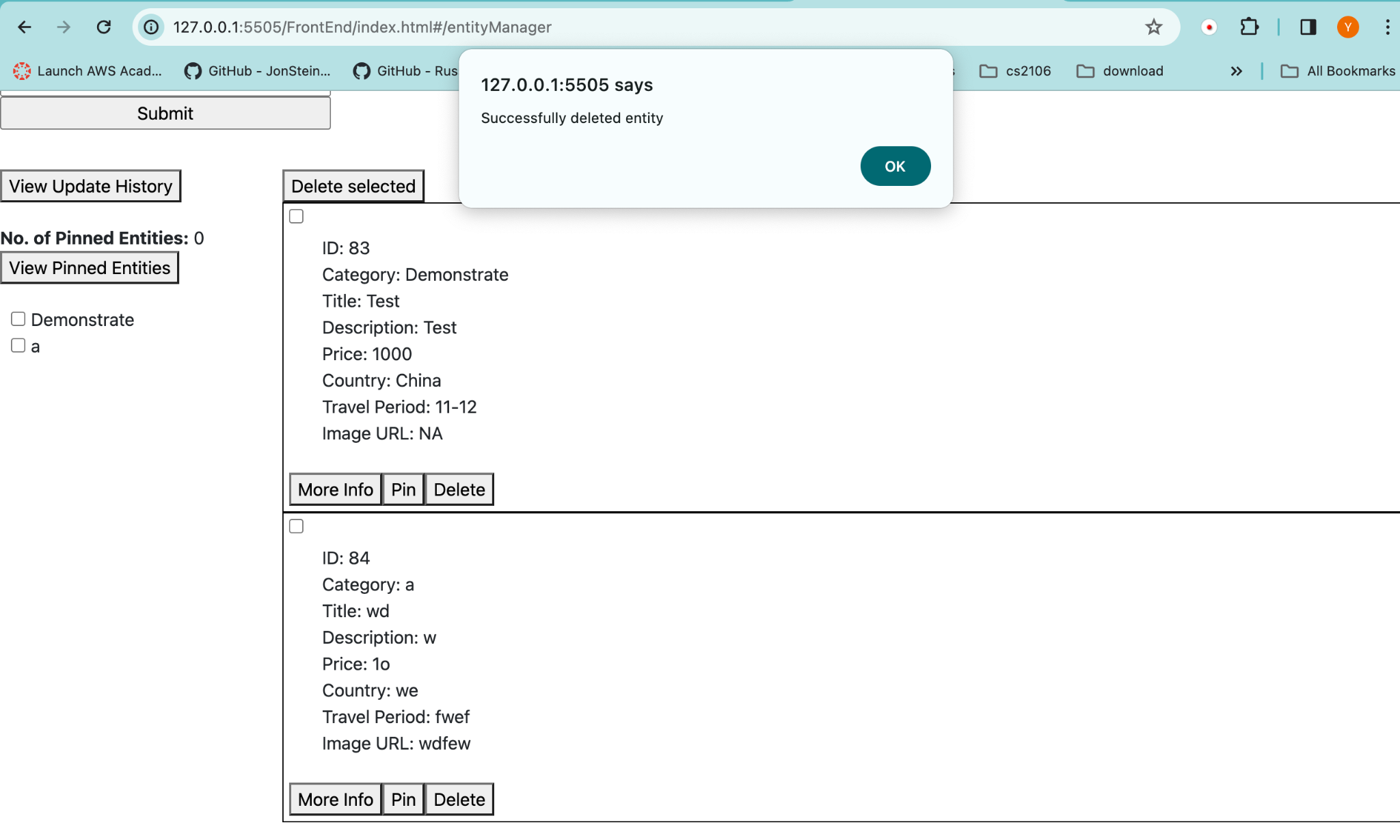


**End-to-end Screenshot**

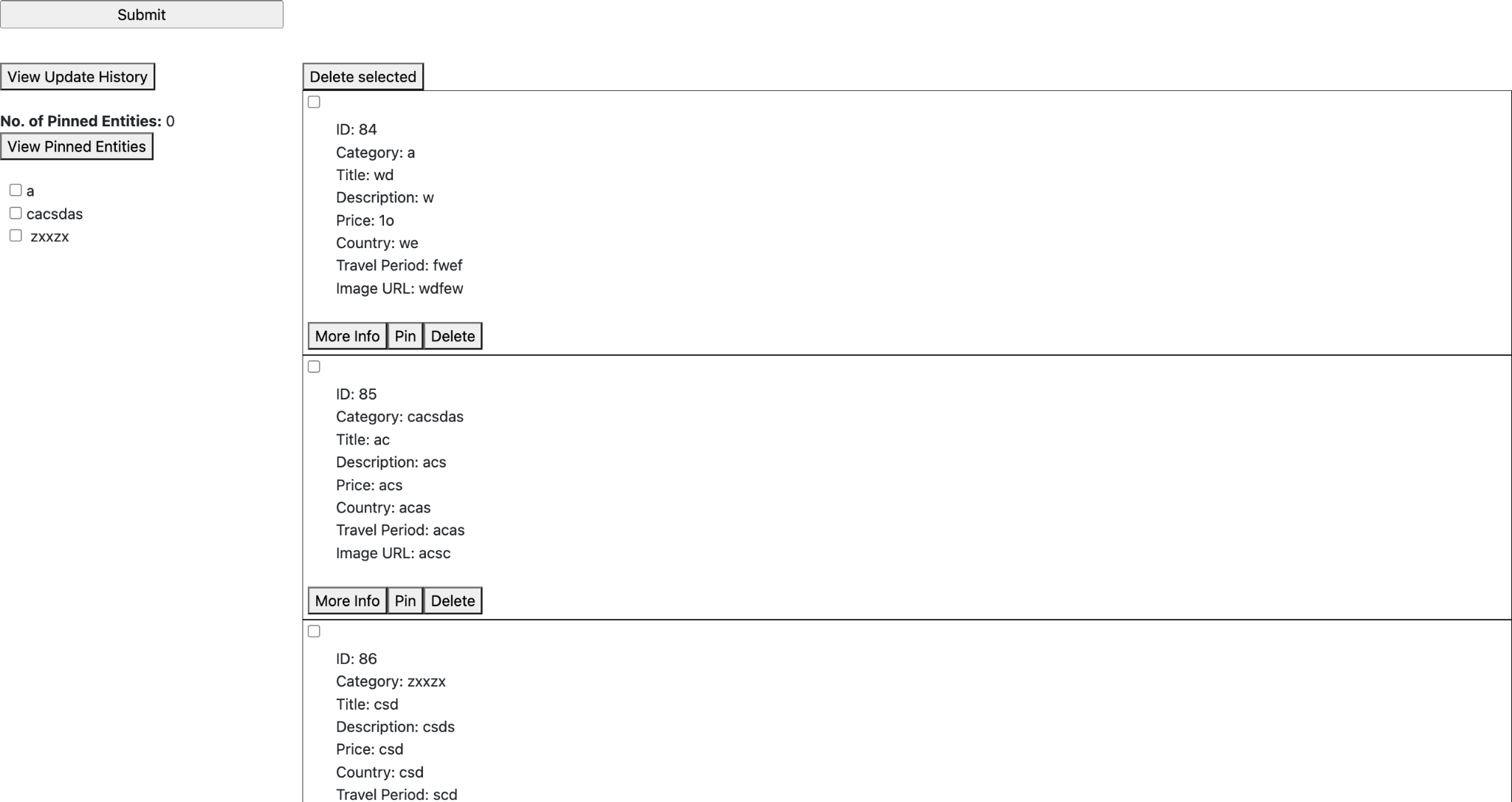
Users can delete a single entity by clicking on its delete button.



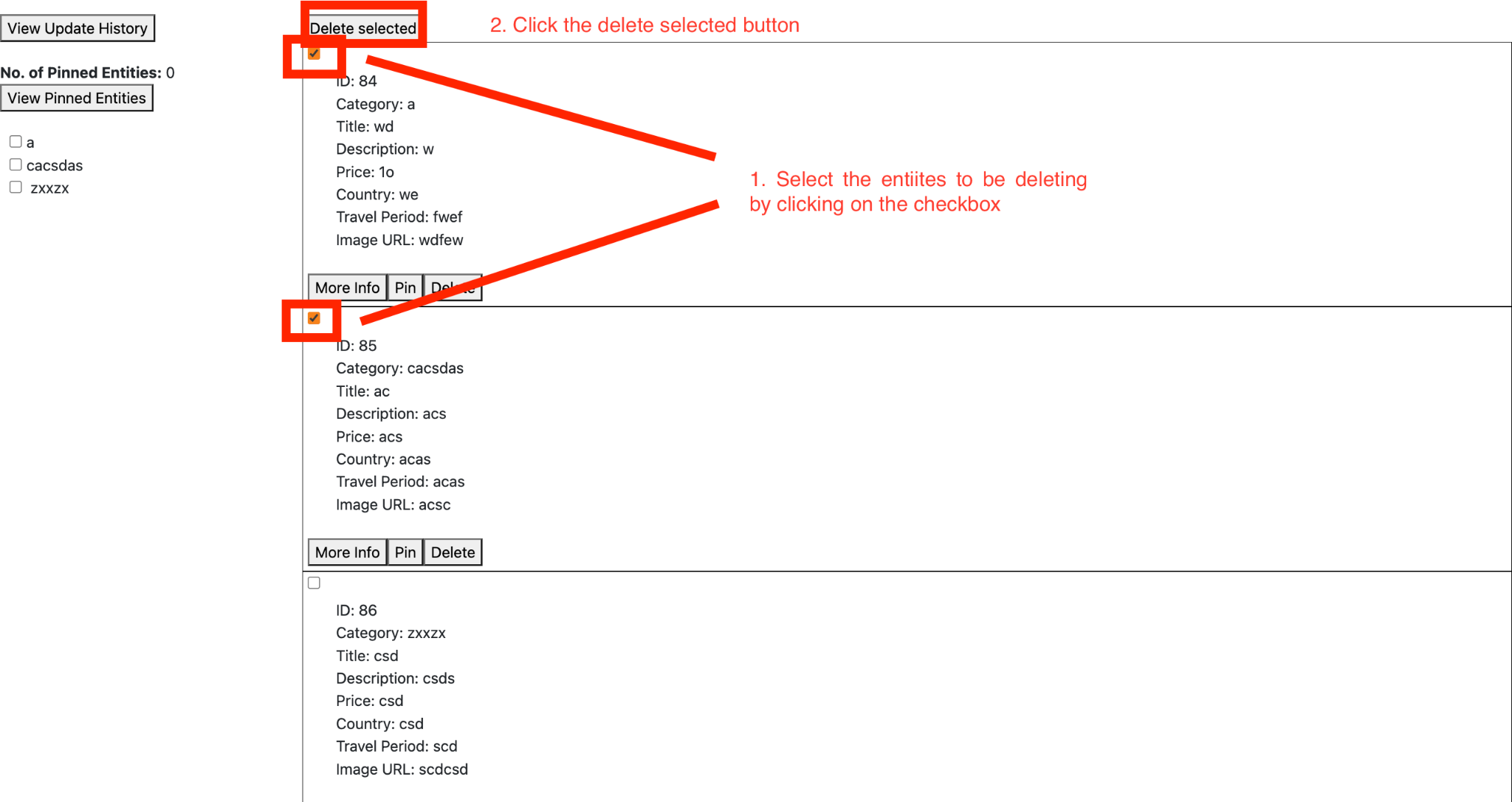
Upon successfully deleting the single entity entries from the database a success alert will pop up.



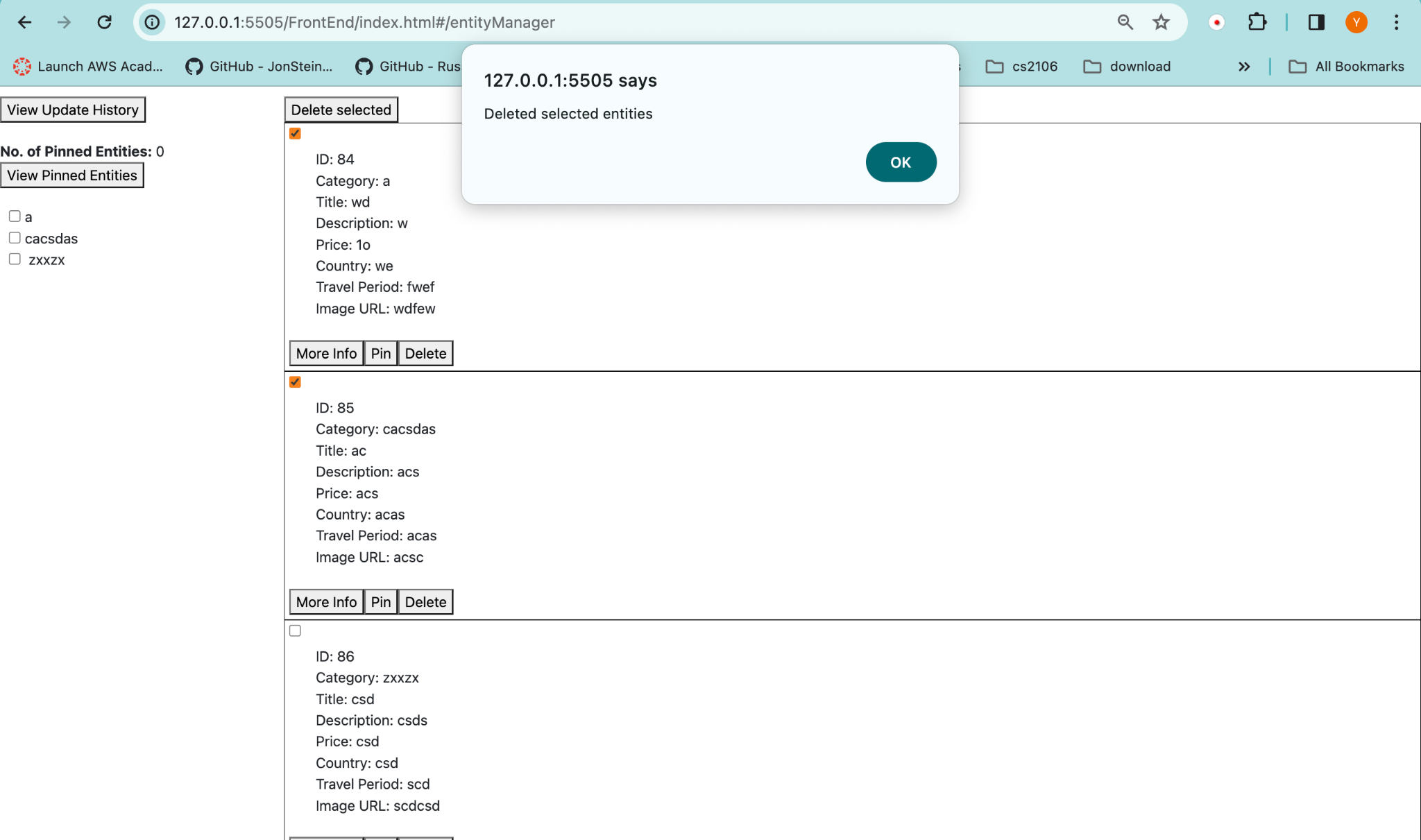
The deleted entity is removed from the list component in the home page



Users can also delete multiple entities at once by selecting the checkboxes of the respective listings and clicking on the ‘Delete selected’ button.

****

Upon successfully deleting the multiple entity entries from the database a success alert will pop up.



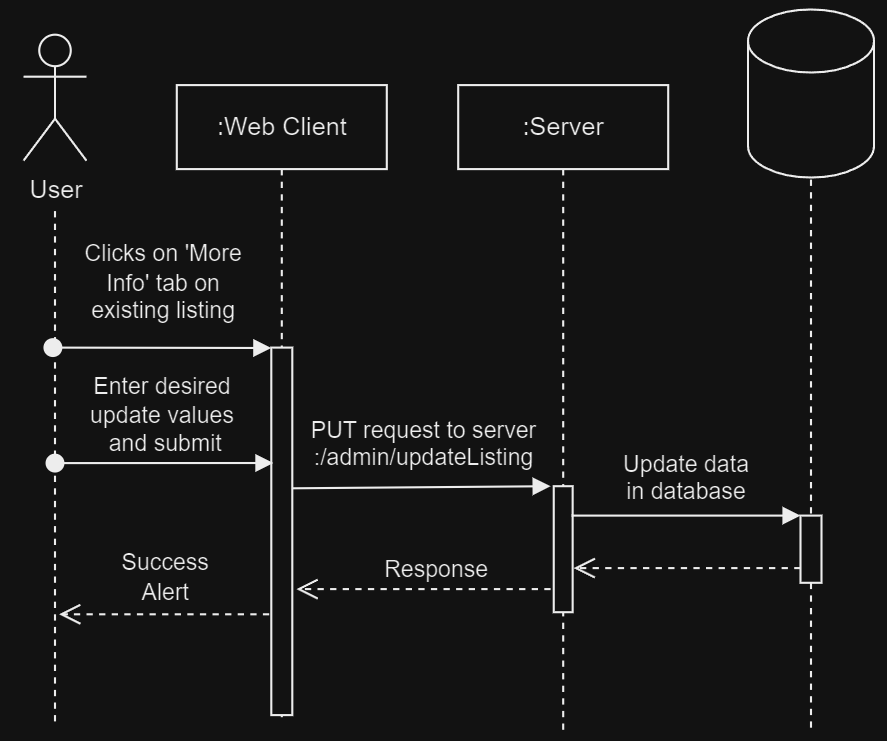
The selected entities are removed from the list component in the home page



### 3.1.6 Update Entity Feature

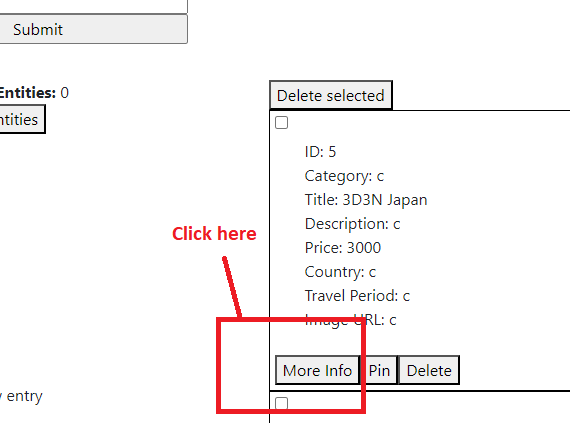
**Sequence Diagram**

The user can click on the ‘More Info’ button on any listing to edit the entity information in the database. Upon clicking the button, the user will be brought to a page that shows the entity’s existing information. The user has to fill up the new Category, Description, Price, Country, Travel Period and Image URL information. Upon clicking the submit button, if the update to the database is successful, the user will receive a success alert indicating a successful transaction.

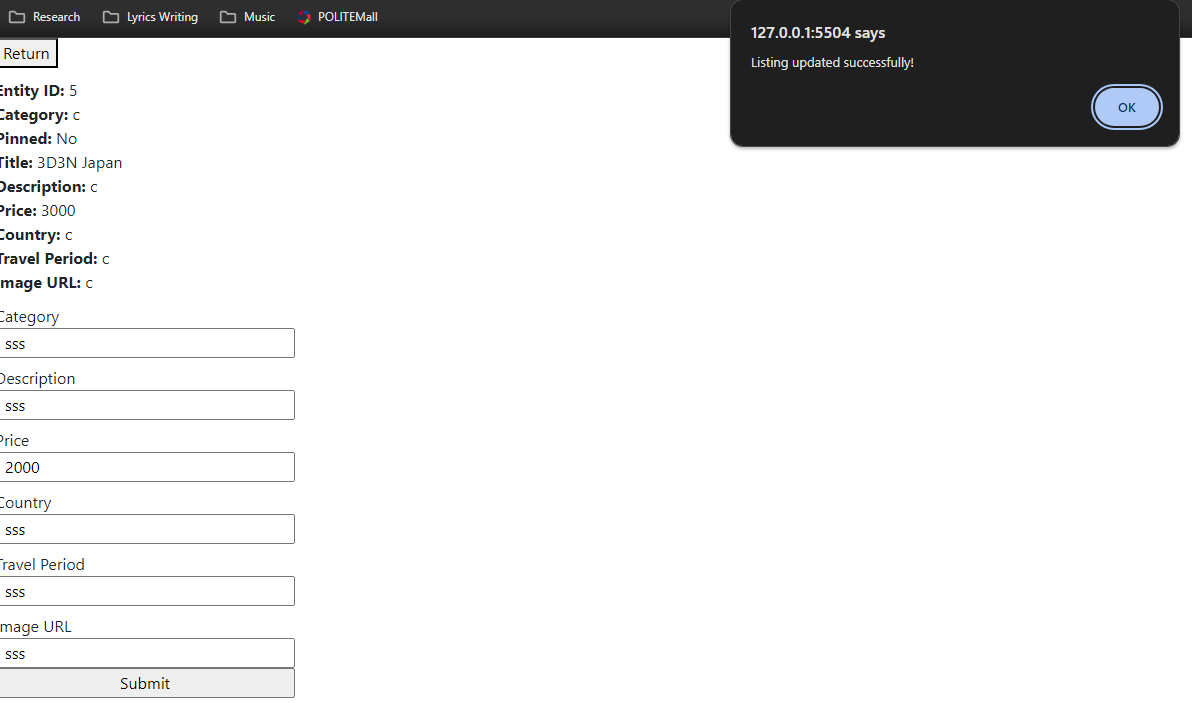


**End-to-end Screenshot**

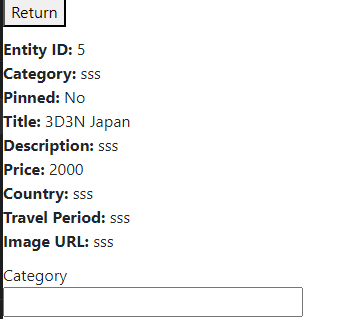
Assume user has already logged in. User can click on the ‘More Info’ tab to edit existing listing.



The user can then update the fields to the desired values and click submit.Upon success, the user will see a success alert.



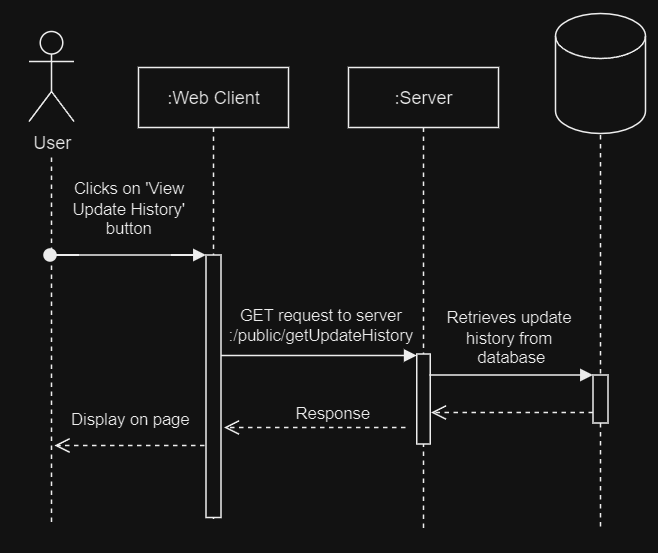
Upon clicking Ok, the user will remain on the same page and see the updated values. The user can continue to try and update the entity again while remaining on the same page or return to the main page by clicking return.



### 3.1.7 Display of Entity History Pre-Update Feature

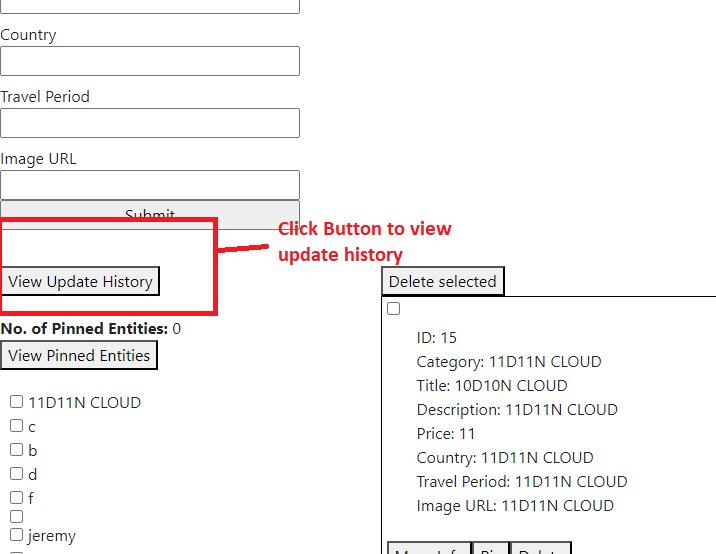
**Sequence Diagram**

When users update an entity using the methodology described in Section 3.1.6, the server will log the existing data in the database before the update. This display of update archives feature provides the user with the option of viewing past entity information on another page. By clicking on the ‘View Update History’ button, the user will be taken to another page to view the archives.

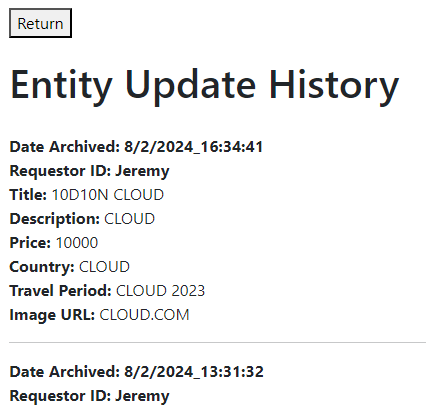


**End-to-end Screenshot**

Users are able to now view update history by clicking the button shown below

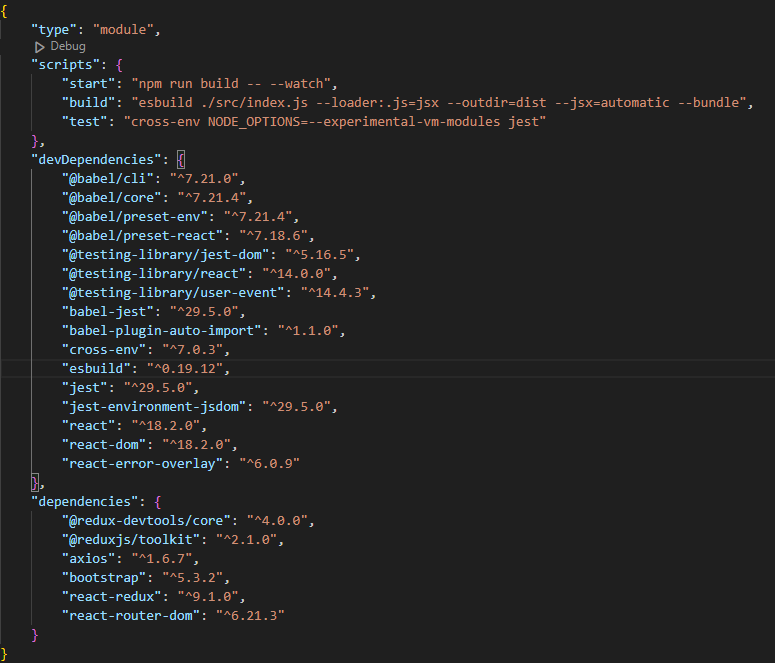


Clicking the button will bring them to a page which lists all entity history with the latest history displayed at the top of the page.



## 3.2 Technology

JavaScript is used for this project. The relevant libraries and modules used are as follows:

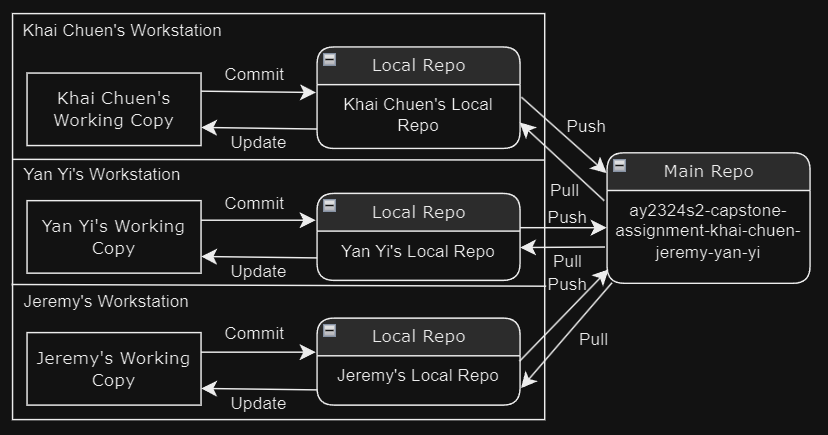
App.js (lines 275 to 290)

## 

## 3.3 Development Tools and Processes

A distributed version control methodology approach is used for version control.

1. A branch will be created for each new feature to be implemented.
2. Changes for the new feature are made locally within the developer’s system and using git add and commit, committed to the individual local repo.
3. Upon the completion of the feature, the developer will git push the changes to the remote main repo by raising a pull request.
4. Other developers will review and approve the pull request.
5. Upon merging the changes into the main branch, other developers will git pull and update their local repository.

**

## 

## 3.4 Interfaces and services

| **Function Integrated** | **Details of Front End Codes** |  | **Details of Back End Codes** |
| --- | --- | --- | --- |
| Login | **SignInSignUpForm** (lines 99 to 104) | <-> | App.js (lines 27 to 45) |
| Register | **SignInSignUpForm** (lines 105 to 119) | <-> | App.js (lines 48 to 67) |
| Add Entity | **AddEntityForm.js:** Fill in of Add Entity Form and submit (lines 49 to 70) POST web service to server | <-> | App.js (lines 71 to 101) |
| List Entity | **EntityList.js** (lines 12 to 37) | <-> | App.js (lines 296 to 306) |
| Delete Entity | **EntityList.js** (lines 39 to 63)  **Entity.js** (lines 19 to 41) | <-> | App.js (lines 275 to 290) |
| Update Entity | **EntityInfo.js:** PUT Update request (lines 45 to 59) to server | <-> | App.js (lines 103 to 160) |
| Pre-Update Entity History Display | **EntityUpdateHistory.js:** GET request to get update history (lines 11 to 19) | <-> | App.js (lines 336 to 345) |