Assignment 6 Database UI

Paula Gearon

Table of contents

Database UI

This project expands on using the contents of the Shopping Trends file from Assignment 3, and the SQL access demonstrated in Assignment 5. The schema and data files from those projects have been duplicated into this project.

This report is available at https://quoll.github.io/608_frontend/report.html

Schema Description

This reiterates the description given in Assignment 5. See that report for more detailed design information about individual fields.

The `Customer` entity is treated as a unique individual, independent of any purchases.

The `Item` entity is the description of a commodity type.

The `Purchase` entity records a transaction in which a `Customer` obtains an instant of an `Item`.

While the original dataset has a one-to-one connection between customers and purchases, this database has been designed to allow a new `Customer` with no `Purchases` yet, and for `Customers` to make multiple `Purchases`.

Many of the attributes of each of these entities are theoretically configurable. For instance, a `Customer`'s `location` can be any of the US states, but a future version may need to expand this to US territories or other countries. Similarly for shipping types, Item categories, Seasons (for instance, "Christmas" and "Halloween"), payment methods, and shopping frequencies.

ER Diagrams

The ER Diagrams from Assignment 5 are included here for convenience.

Shopping Trends Chen ER Diagram

Shopping Trends Crows Foot ER Diagram

UI Description

This UI has been built using Flask and the MySQL-connector Python library. All required packages are described in the requirements.txt file. Because the UI is built using Flask, it is not available online, though it can be run locally.

To run the application:

- Load the Table Definitions into MySQL.
- Load the data (transformed from the original CSV) into MySQL.
- In the Flask directory, load the required packages using `pip install -r requirements.txt`
- Create a `.env` file in the flask directory.
- Start the Flask server using `flask --app app run`
  - This starts a web server on `localhost:5000`
- Open a web browser and navigate to `http://localhost:5000/`

`.env` file

While environment variables can be set in the terminal, it is easier to set them in a `.env` file. The following variables are required:

```
MYSQL_USER = <your_mysql_username>
MYSQL_PASS = <your_mysql_password>
MYSQL_HOST = localhost                # your MySQL server name
MYSQL_PORT = 3306
MYSQL_DATABASE = <your_database_name>  # your database
```

Listing Pages

Each of the following pages displays large lists of data. This data is paged to 50 items per page, with paging controls at the bottom of the list.

Customers

Upon starting, the UI will show a list of customers. The Customers page can be accessed at ant time by Selecting the "Customers" link in the navigation bar. The initial customer data is extracted from the original CSV file and displays:

- `ID`: The customer ID. Select this button to view that customer's information.
- `Age`: The age of the customer.
- `Gender`: The gender of the customer.
- `Subscription Status`: Indicates whether the customer is a subscriber or not.
- `Previous Purchases`: The number of iprevious purchases made by the customer.
- `Payment Method`: The *preferred* payment method of the customer.
- `Frequency Name`: The frequency of purchases made by the customer.
- `Location Name`: The location of the customer.
- `Purchase`: This is a button that navigates to the purchases page for that customer. The original dataset contains a single purchase per customer, but additional purchases can be added.

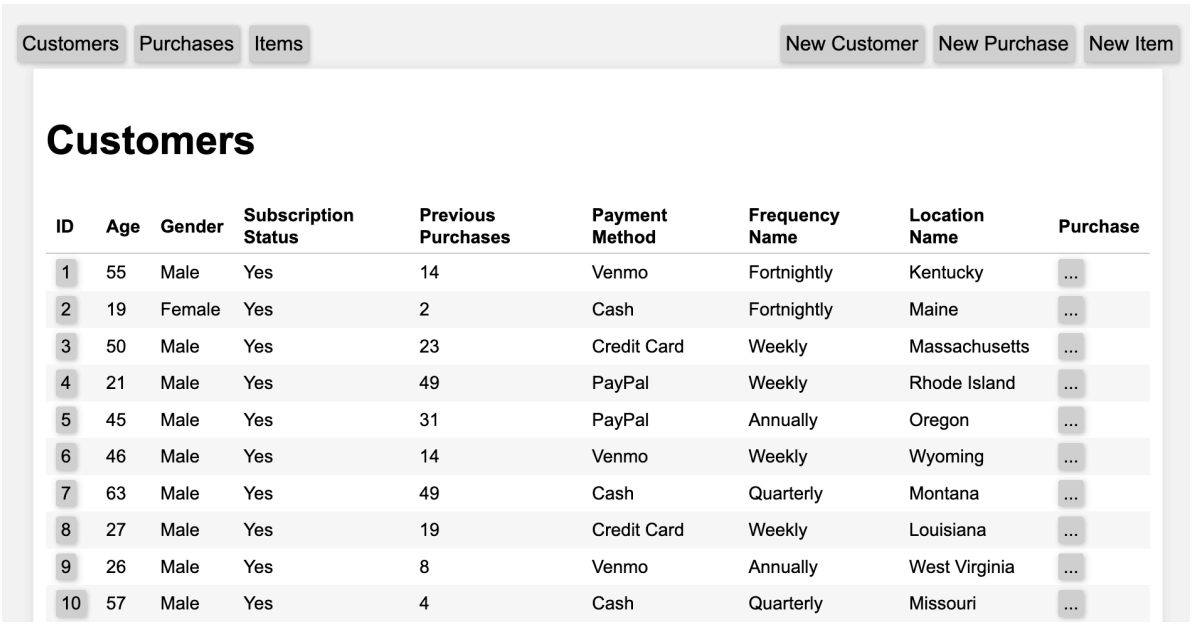


Figure 1: Customer Listing

Purchases

The Purchases page can be accessed by selecting the "Purchases" link in the navigation bar. The initial purchase data is extracted from the original CSV file. This page displays:

- `ID`: The purchase ID. Select this button to view the information for that purchase.
- `Customer`: The ID of the customer who made the purchase. Select this button to view the information for that customer.
- `Item`: Th name of the item purchased. *Hover over the item to see the category, size, and color of the item.*
- `Amount`: The amount of the purchase.
- `Season`: The season in which the purchase was made.
- `Rating`: The numerical rating given to the purchase by the customer.
- `Payment Method`: The payment method used for the purchase.
- `Shipping Type`: The shipping type used for the purchase.
- `Discount`: Indicates whether a discount was applied to the purchase.
- `Promo Code Used`: Indicates whether a promo code was used for the purchase.

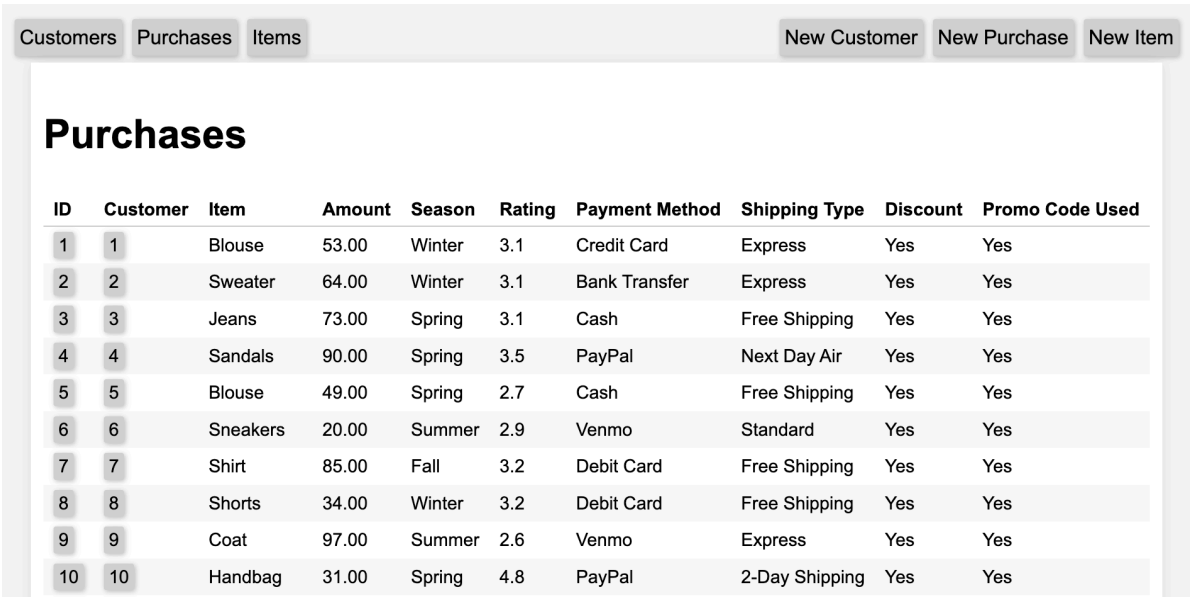


Figure 2: Purchase Listing

Items

The Items page can be accessed by selecting the "Items" link in the navigation bar. The initial item data is extracted from the original CSV file. This page displays:

- `ID`: The item ID. Select this button to view the information for that item.
- `Name`: The name of the item.
- `Category`: The category of the item.
- `Size`: The size of the item.
- `Color`: The color of the item.

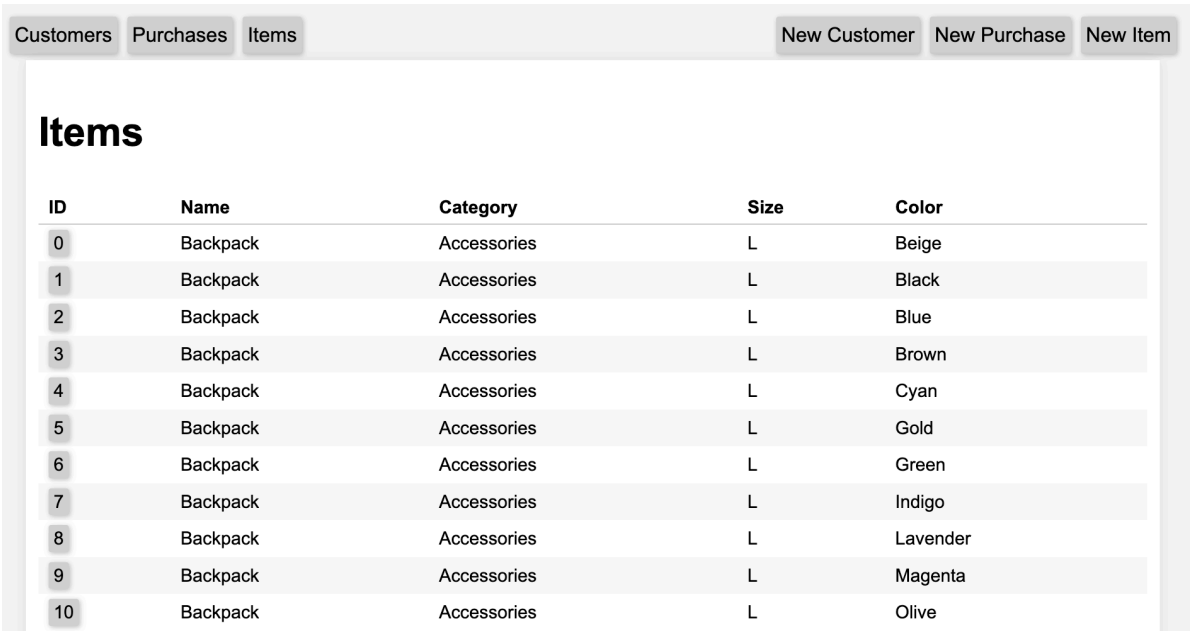


Figure 3: Item Listing

Paging Controls

As mentioned above, each of the preceding listings are shown in lists of 50 elements at a time. An example of the paging controls (from the bottom of Customers) is shown here.

Paging Controls

View and Editing Pages

Customer View

The Customer View page can be accessed by selecting the customer's ID button in the Customers page. This page shows the Customer ID in the header, and displays all of the information for that customer:

- `Age`: The age of the customer.
- `Gender`: The gender of the customer.
- `Subscription Status`: Indicates whether the customer is a subscriber or not.
- `Previous Purchases`: The number of previous purchases made by the customer.
- `Payment Method`: The preferred payment method of the customer.
- `Frequency Name`: The frequency of purchases made by the customer.
- `Location Name`: The location of the customer.

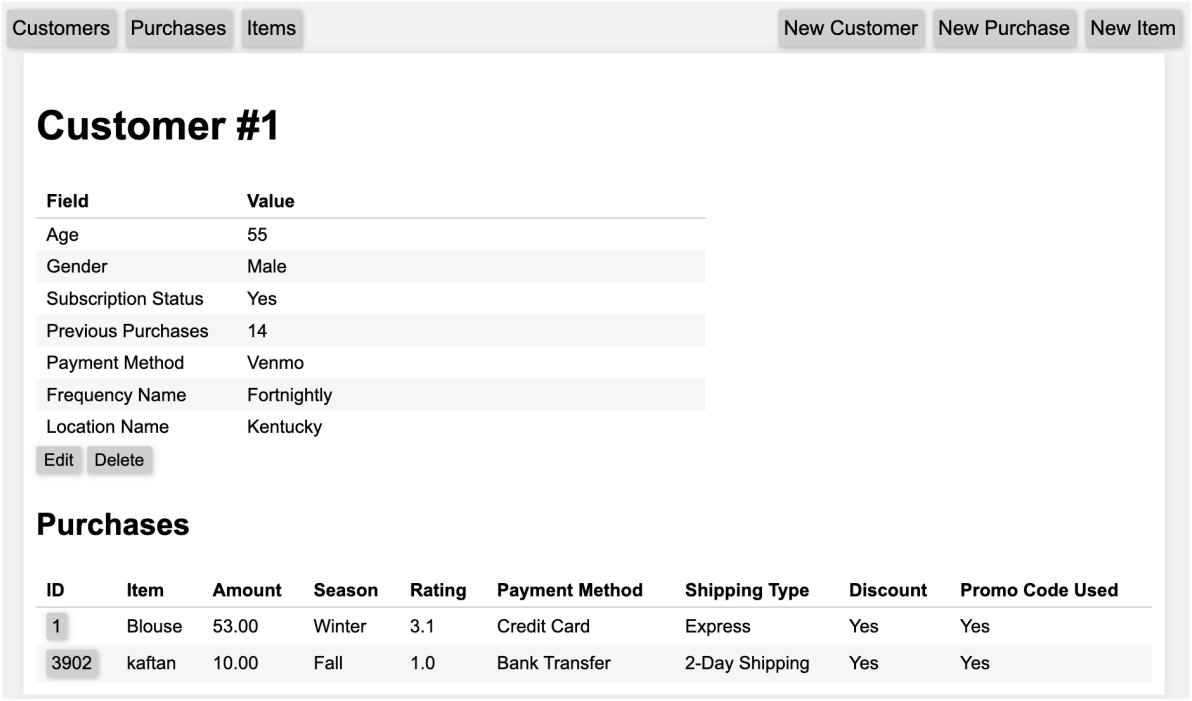


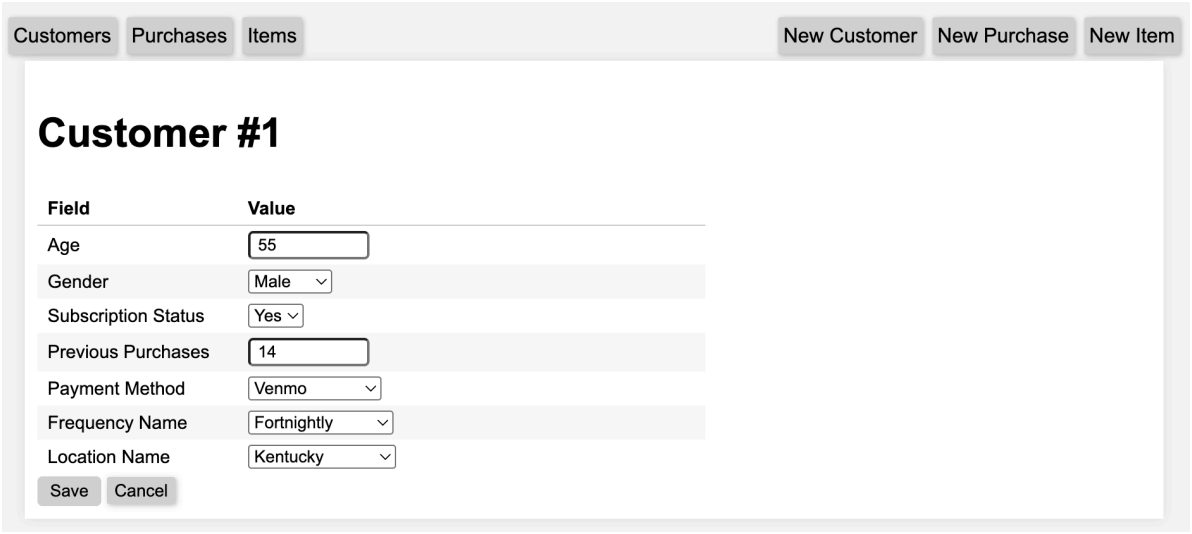
Figure 4: Customer View

Under the customer information, the purchases made by that customer are displayed. These contain the same information as the Purchases page, but only for the current Customer. The information for each purchase can be viewed by selecting the purchase ID button.

Note that the Customer view also has a **Delete** button. This can be used to remove the customer from the database.

Selecting the "Edit" button will take you to the Customer Edit page, where the information can be editing using the same layout as the Customer View page. The "Save" button will save the changes to the database, and the "Cancel" button will take you back to the Customer View page without saving any changes.

Examples of the Customer Edit and New Customer screens are shown here:



New Customer Screen

Purchase View

The Purchase View page can be accessed by selecting the purchase ID button in the Purchases page. This page shows the Purchase ID in the header, and displays all of the information for that purchase:

- `Customer`: The ID of the customer who made the purchase. Select this button to view the information for that customer.
- `Item`: The name of the item purchased. The category, size, and color of the item are displayed to the right.
- `Amount`: The amount of the purchase.
- `Season`: The season in which the purchase was made.
- `Rating`: The numerical rating given to the purchase by the customer.
- `Payment Method`: The payment method used for the purchase.
- `Shipping Type`: The shipping type used for the purchase.
- `Discount`: Indicates whether a discount was applied to the purchase.
- `Promo Code Used`: Indicates whether a promo code was used for the purchase.

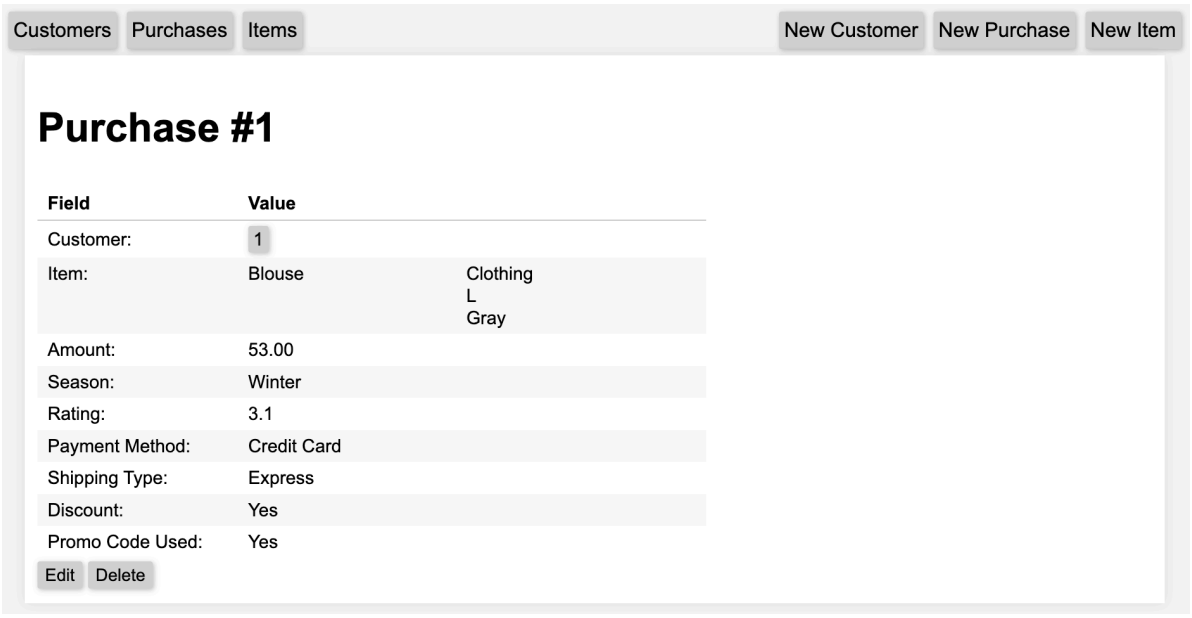


Figure 5: Purchase View

This screen also has a **Delete** button. This can be used to remove the purchase from the database. Note that if the purchase is referenced by a customer, then attempting to remove it will lead to an error page.

Selecting the "Edit" button will take you to the Purchase Edit page, where the information can be editing using the same layout as the Purchase View page. Note that most of these values are dropdowns, and the values are limited to those in the database.

The "Save" button will save the changes to the database, and the "Cancel" button will take you back to the Purchase View page without saving any changes.

Purchase Edit Screen

New Purchase Screen

Item View

The Item View page can be accessed by selecting the item ID button in the Items page. This page shows the Item ID in the header, and displays all of the information for that item:

- `Name`: The name of the item.
- `Category`: The category of the item.
- `Size`: The size of the item.
- `Color`: The color of the item.

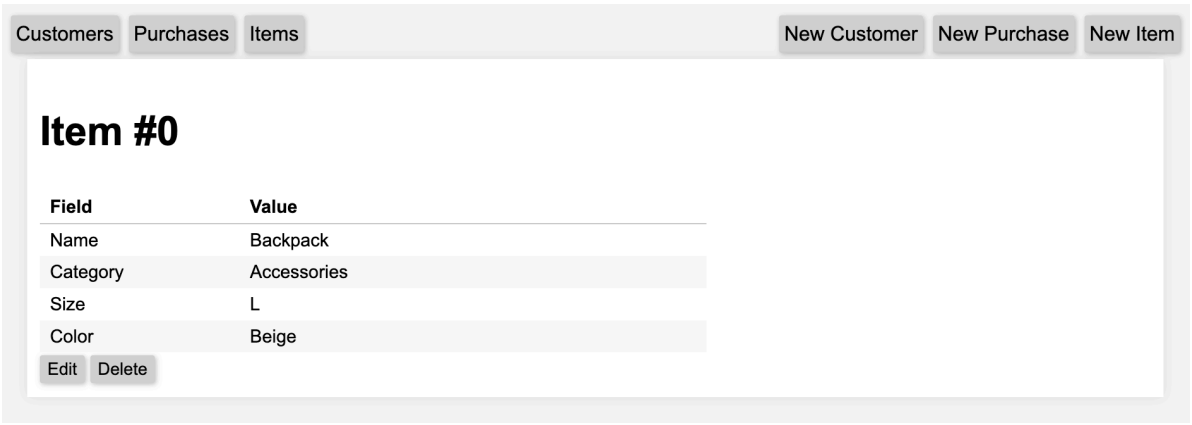


Figure 6: Item View

This screen also has a **Delete** button. This can be used to remove the item from the database. Note that attempting to remove an item that is referenced by any purchases, will lead to an error page.

Selecting the "Edit" button will take you to the Item Edit page, where the information can be editing using the same layout as the Item View page. Selecting the "Save" button will save the changes to the database, and the "Cancel" button will take you back to the Item View page without saving any changes.

Item Edit Screen

New Item Screen

Data

The schema is in `data/definitions.sql`

The data is found in `data/data.sql`

Developer Comments

UI Requirements

It is difficult to design a user interface without requirements. The approach I took was to implement standard operations around the major entities, and expanding these as I progressed.

Given that the original data is a static CSV file, having a site that simply displayed the data may have been considered adequate. But a normal system would expect users to enter new data as it became available, so it seemed reasonable to include this functionality. Similarly, if

data can be added, it should be modifyable, to correct errors. Similarly, it should be possible to remove data.

While it is possible to expand this system to make each of the tables editable (so users can add things like shipping types or locations) this level of expansion would significantly expand on the scope of a project which did not imply that it needed to go that far. The chosen feature set is typical of an initial release.

Client/Server Model

A project like this may often make heavy use of Javascript for validation, and for appealing effects and user interactions. This is not an area in which I excel, so I have chosen a utilitarian UI, though I do believe it is pragmatic.

Modern day software will also have a much more capable client that changes its own rendering and interacts with the server almost entirely through microservices. This contrasts with the Flask application presented here, where the client pages are static, and workflow is managed by the server. This is not to say that the submitted application is inadequate, but it does not reflect contemporary deployment styles.

Deployment

Finally, I am uncomfortable with providing only the source code and description without presenting a running system. However, most available architectures would require a SQL database for the UI to interact with, which I do not have free access to.

Some alternatives do exist. One is to use a free service like Google Sheets to manage the database. This is possible, but does not provide a SQL API. Another is to run a database within the web page. SQLite is available as a Web Assembly module, and page initialization could load the schema and data files. An alternative is Absurd SQL which reimplements SQLite in Javascript, using IndexedDB for persistence.

Running a database within the page changes the characteristics of the system significantly. The data would suddenly become private, and would need an explicit export mechanism to share changes. The implementation would also need to move entirely into Javascript, which introduces new complexity, and may not have been feasible in the provided timeframe.

The existing Flask application speaking to a MySQL service may require more work on the part of the user to set it up, but it offers a more flexible and extensible architecture. For instance, the database could be changed quite easily to SQLite, Postgres, or many others. Also, the Flask pages are simple and template driven. This appears to be suitable for the assignment, even with the frustrating lack of a live deployment option.