Enhancing Data Accessibility in Windsor: A User-Centric Web Application for the Open Data Catalogue

Keerat Bajwa, Miran Beshir, Moussa Jaafar-Sannan, Eli Pardalis, Rocio Rueda, Mark Zayat

401 Sunset Avenue, Windsor, Ontario N9B 3P4, Canada

bajwa12@uwindsor.ca
beshirm@uwindsor.ca
jaafarm@uwindsor.ca
pardali1@uwindsor.ca
ruedar@uwindsor.ca
zayatm@uwindsor.ca

Project repository found at: github.com/zayatMark/3220-group-project

***Abstract*— This web application was created with the aim to facilitate the access and useability of data on the City of Windsor Open Data Catalogue website. Since this catalogue may be accessed for several purposes and by various types of users (researchers, city planners, tourists, etc.), user experience was a crucial factor for us during these past iterative phases of development. When completed, our web application would be a fine companion site to the Open Data Catalogue, one that emphasizes interactivity as well as the accessibility and filtering of data.**

***Keywords*— Open Data Accessibility, User Experience Design, Web Application Development, Data Management Innovation, Civic Data Platforms**

I. Introduction

During the early stages of the development process, we sought to answer a couple key questions, such as: how might we integrate the datasets into our web app? And how might we enhance user interaction? We initially investigated a few third-party JavaScript libraries, but ultimately opted against using them due to their complexity. Instead, we display a subset of a given dataset from its .csv file, which is publicly available for download on the Open Data Catalogue page pertaining to that data. Likewise, the .csv file containing the full dataset is readily available for download on our site as well. Regarding the matter of user interaction, we decided to handle the front end using React.js, which aligned with our goal of creating an intuitive and scalable app since its architecture is component-based. React proved to be an effective tool while we were in the process of making the prototype real and interactive, and in the future, it will be the continued framework of choice while we work on fleshing out and testing our application's features.

While there exists a wealth of data in the Open Data Catalogue, the challenge we undertook here was figuring out how to make this data more accessible and easier to manage by a wide variety of users. Thus, the problem we sought to tackle was the current lack of an intuitive platform that meets the needs of these users, as not all have the computer literacy to parse through raw data sets. What our solution offers is more efficient data exploration, manipulation, and ease-of-use. By transforming how this data is accessed and interacted with, our web application can play a pivotal role when it comes to enhancing transparency in data-driven municipal decision-making in various sectors. Groups that may benefit from accessing this portal are taxpayers, city staff, administrators, and visitors, and as such, this web application is a real asset to the City of Windsor.

Likewise, a couple high level goals that we had in mind while developing this application are:

1. To create a scalable and adaptable platform that grows with the addition of new datasets and changes in existing data.
2. To foster economic growth by enabling data-driven innovations and reducing the

overhead costs associated with data management.

Since our portal is anticipated to make city operations more cost-effective, this should stimulate Windsor's economic growth and, ultimately, will foster a culture of openness and civic engagement.

II. Related Works

The development of our web application started with a close examination of the City of Windsor's Open Data Catalogue. The portal offers a variety of information, from geographical systems to municipal operations, but its user interface provides a barrier for use to the average user. Furthermore, due to the lack of curated data filters and a user-friendly interface, it is hard to interpret and navigate the available raw data sets. Our web application works on enhancing the user experience, simplifying data accessibility and discovery, uniquely for every type of user that has needs for such data. The original portal served as a critical starting point for our project, helping us understand the types of data available and the potential use cases for each data set. With the changes we strive to implement, we will allow more transparency and better user engagement, making municipal data more accessible to a broader audience.

III. Approach

A. Phase 1: Inception

In the inception phase of the project, we established high-level goals focused on developing an open data portal for Windsor. Our primary objectives revolved around enhancing accessibility and promoting transparency in data management practices. To achieve these goals, we identified and addressed various constraints, including strict adherence to data privacy laws, ensuring scalability for future growth, and operating within budgetary limitations. Formulating a robust business case emphasized the cost-effectiveness and advantages of the open data portal. Additionally, we developed a comprehensive use-case model outlining key functionalities such as data viewing, uploading, editing, distribution, and API access. This ensured alignment with project objectives and user requirements. Supplementary specifications were outlined to address concerns regarding user-friendliness, scalability, data privacy, and reliability, aiming to cater to the diverse needs of stakeholders and end-users. Identifying potential risks such as data privacy compliance issues, API documentation challenges, and budgetary concerns. We then proceed to make a prototype. The prototype development phase involved creating a mock UI prototype using Figma, showcasing user interactions and providing stakeholders with a tangible representation of the proposed solution. Additionally, we developed a use case diagram to identify primary functionalities and interactions, laying the foundation for understanding user requirements and defining the project scope.

B. Phase 2: Elaboration Iteration I

During the second phase, our main focus was on developing the user interface of the open data portal using React.js. We emphasized a modular and reusable approach through React class components, aiming to enhance scalability and ease of maintenance. As part of this effort, we expanded our UML documentation to include class and sequence diagrams, providing a high-level overview of the application's architecture and interaction flow. These diagrams served to guide both the implementation process and architecture planning effectively. Additionally, we established a modular architecture wherein each major element was represented as a class component, further enhancing the system's scalability and maintainability. We also placed significant emphasis on implementing a clear flow within the website, starting with the App class component and seamlessly navigating through various pages such as Home, Login, MultiDataView, and Data Overview. To ensure smooth page navigation and rendering, we implemented the App class as the main controller, responsible for managing these tasks and ensuring a seamless user experience. During this phase, we utilized tools such as Jira for project management and GitHub for version control. We found both invaluable tools for facilitating our team collaboration and quality assurance processes. We also used React testing library to test if components are rendered properly and that classes have

correctly functioning methods and correctly set fields.

C. Phase 3: Elaboration Iteration II

In the third phase, our focus was on refining and enhancing the open data portal's usability. We identified various design patterns such as Controller, Creator, and Information Expert, which were then implemented in the codebase to increase maintainability and extensibility, streamlining the development process and ensuring the codebase remained flexible as the project evolved. Additionally, we prioritized enhancing UI/UX elements to optimize the user experience, focusing on visual appeal, navigation flow, and responsiveness, thereby creating a more user-friendly interface. We then added additional features like a download button functionality, enabling users to download CSV files directly from the portal, thus enhancing data accessibility and usability. Furthermore, we integrated functionality to display CSV files within the portal, eliminating the need for external applications and improving the user experience and convenience. We then updated our UML diagrams to accurately reflect the current system's design and structure.

IV. Architecture

The open data portal was created using React.js. This framework allowed us to use object oriented design even when creating the UI. We used React class components for all the major UI elements, which allowed for overriding a render method to create the look of the element. All UI classes extend the Component class provided in React, since this provides a framework for creating UI elements. We used a few libraries in the project, which were the React Router Library for page navigation, the Universal Cookie Library for cookies, Lucide React for some icons, and Papa Parse for .csv processing. We also used the React Testing Library for unit testing, which made sure that the UI was correct and that the classes were returning and storing the correct information.

A. Page Routing

The flow of the website starts with the App class component. It contains the page routing logic that allows for the correct page to be displayed when the URL is changed. Because the App class is responsible for rendering the correct page, it is the first component to be called when the website is loaded. Since the navigation bar is on every screen, the Navbar class component is used directly in the App which will always render it. The Navbar class is responsible for displaying the navigation buttons at the top of the screen, with all the formatting. It is also responsible for changing the page URL to the page that corresponds to the clicked button. This allows the App component to then receive the change in URL and render the correct components. Each page on our website is its own class component. So the home page, login page, multidata view page, and data overview pages each have their own classes to specify their logic and rendering. The login page is relatively straightforward since it does not have functionality yet, as that would be completed in a future iteration. Currently, the LoginForm class is only responsible for laying out the various HTML elements for the screen.

B. Home Page

A more complex page like the home page demonstrates how React allows for reusable components and for the functionality to be built into the UI. The HomePage class is responsible for rendering the HomeCarousel and the VerticalDataList classes. The HomeCarousel component loads the images to display in the carousel and handles the formatting of the space to ensure the image carousel is the correct size. It then uses the ImageCarousel class to render the carousel. The ImageCarousel class is responsible for rendering the various elements of the carousel, like the images, scroll buttons, and selection buttons. It is also responsible for tracking the current image index and providing methods that change the current image, which are set to run when their respective buttons are clicked. This allows for the ImageCarousel class component to be reused on other pages in future iterations if there is ever a need. The HomePage class also renders the

VericalDataList which is responsible for displaying the list of recently viewed data. What it does is load the last visited pages from a cookie and then it creates and renders up to five VerticalDataListItems. It will provide each of these objects with the required shortened information from a DataItem object, which is stored in a list. The VerticalDataListItem will render the box and all of its contents and provide the functionality to click on the component and change the screen to that piece of data. The DataItem class is responsible for encapsulating the data id, icon, title, and short description and making them easily accessible. All of these classes come together to create the home page, while allowing for various elements to be usable in other parts of the site.

C. Data Filtering and Selection Page

Another important page is displayed through the MultiDataView class. It is the page that is responsible for displaying all the data options and providing buttons to filter the data by category. It does this by working together with the Classification class. This Classification class is not a UI element but is rather responsible for storing information about all the file types as well as providing methods to access this information. It provides methods such as getIdContent, which allows for getting the information related to the data with a certain ID. It also provides methods to retrieve all the data and to return a list of data that only matches the specified filter. The MultiDataView class instantiates Classification and uses this filtering functionality to render the currently selected data options. It has a method to update the data to display when the button is clicked and another method that sets all the data visible initially. The MultiDataView component handles rendering the correct data, while the Classification class handles retrieving the data.

D. Data Overview Page

The final page that we created was a data overview page. This page shows the detailed view of the data and will change its contents based on which piece of data is being viewed. It is responsible for rendering the Image and Detail classes, to which it provides the id of the data. It also updates a cookie on the user's machine that tracks the last five visited pieces of data. The Image class displays the icon and renders the Desc class. The Desc class is responsible for displaying the data's title, tags and for downloading the data. It uses the Classification class to retrieve the title, tags, and file path. It has a method that is called from a button press that will download the data and a helper method to get the tags of the data. The other component rendered on this page by the Overview class is the Detail class. This class is responsible for rendering the three dropboxes and providing them with an ID for the dropbox and the data ID. The Dropbox class renders the dropdown and tracks its state. It loads the necessary text using the Classification class for the descriptions and renders the Data class if it is a visualization. It has a method to toggle if it is expanded or not, as well as a method to get the content of the data. The Data class gets the file path using the Classification class and renders the DisplayCSVData class if there is data, or a message about no data being found if there is not. The DisplayCSVData class loads the specified file and uses Papa Parse to parse the .csv data to be displayed. It does this when the class is first displayed. It then renders up to a maximum number of lines into a table, informing the user that the data can be downloaded to view the rest if it is very long. It is the coordination of these classes that creates the data overview page.

E. Result of the Class Components

All of these class components come together to create the cohesive data portal that we ended up with. The use of separate classes for all these elements allows for reuse to occur and for the separation of the various elements into manageable chunks. It allowed us to rapidly iterate on our design and to easily make changes to the project throughout. This mind towards reuse would also benefit us in future iterations since as more functionality gets built out, more overlap in requirements will occur.

V. Discussion

A. Overall Behaviour of Prototype

When entering the website, the user is directed to the homepage. The homepage shows a slideshow of the different attractions in Windsor as well as a list of recent data the user has visited. At the top is a navigation bar, where users can click to different directories. For example, there is a person icon on the top left where the user can login. As well, there is a button called “Data” where users can search for datasets in the City of Windsor. With the easy-to-use, simple design built on React.js, this website offers many different uses, including searching for a particular dataset that government officials can use to make decisions,for tourists who would like to know the list of city attractions in the area, and for families trying to find the list of schools and recreational facilities in the area.

B. Original versus Updated Prototype

Our original prototype consists of a home page, a login page, a detailed data overview page, and a multi-data view page. The multi-data view page contained categories like sports, food, health, robotics, and law, as well as types of media files like image, audio, video, and text files. There was also an option to filter the data by year between 2016-2024. Although these categories cover very important fields for different types of datasets, it did not align closely with the City of Windsor’s open data portal. We changed the categories of datasets while still keeping the type of media files (text, audio, video, and image file filtering), but changed the topic categories to recreational/educational, transportation, legal/issues, environmental, and infrastructure. In addition, while our current prototype does not have yearly data filtering, this is something we plan on developing in future iterations.

For each clickable box displayed in the multi-data view page, the original design contained white boxes with a black border. While the design was very aesthetically pleasing, we decided to use light-gray boxes that darken when you hover over them. This allowed for a more simple design that matched the gray hues in the filter box of categories. In addition, instead of each box having a stethoscope image for each box, we replaced this image with a logo of the City of Windsor.

C. Using GRASP Patterns

There were many GRASP patterns used in the overall object-oriented design for our proposed open data portal website. The main pattern allowing for a main page to be linked to other pages is the controller. This pattern handles user interactions and system events by deciding which components to display and what actions to take in an application. An example where this is used is the App class in App.js. Within its render() method, it manages system events and user interactions, centralizes the control logic in a single entity, and renders anything the user wants to display. In other words, this helps keep the app organized and easy to change or grow, making updates and testing simpler.

Another relevant class is the Creator. This pattern involves a class creating other objects or instances of that class when it has the information necessary to do so. It also promotes encapsulation. For example, the MultiDataView class in multidata.js, since it creates instances of the Classification class. Essentially, this method has the required information to create Classification objects, is the class that stores the object, and uses them for data processing tasks (for instance, when fetching or filtering data) upon the user’s requests.

Thirdly, the Information Expert pattern is used to assign responsibilities to the class that has the specific knowledge needed to perform a given task. We used this design pattern predominantly in our Classification class with the filterData() method in filecomponents.js. This is because it is responsible for filtering data based on criteria provided by the values parameter. It stores the information on the data that needs to be filtered, making it an information expert on this data. Putting all data filtering and handling in the Classification class makes the app less complicated and easier to manage.

An important design pattern we also used is Low Coupling. This pattern aims to minimize

dependencies between different parts of an application, making it easier to change one part without affecting others. Methods such as getListOfFiles() in the Classification class demonstrate an example of the Low Coupling approach since it reminds us that the MultiDataView class and other UI classes do not actually need to interact with the logic in Classification, which contains the getListOfFiles() method, to display information. To put it differently, the responsibility of retrieving and filtering the data is neatly contained in an independent class that is accessible by others when required, rather than each class needing to know how this implementation works.

Lastly, High Cohesion is used in this open data portal's object-oriented design. High cohesion ensures that a class is focused on a single purpose or closely related activities, making the class more understandable and manageable. The DisplayCSVData class in DisplayCSVData.js demonstrates High Cohesion since it's dedicated to a main task. This class can parse, load, and display the data from a comma-separated values (CSV) file for a given item, making the class very focused on showing the CSV on screen. This has many benefits, like more organized and modular components, more readable code, and easier modification.

D. OOP Design Changes

Our class diagram slightly changed as our requirements changed and we had more classes involved. For instance, in phase II, our class diagram consisted of the following classes: App, Navbar, Homepage, MultiDataView, LoginScreen, HomeCarousel, VerticalDataList, Classification, Overview, Detail, Image, and Dropbox. While these classes constructed the base implementation for our website, we decided for phase III to create other classes for more specific applications. A new class, DisplayCSVData, class was used for its ability to visualize the data in tabular format on the screen. While this class can display the visualization in an empty html based page, in order to embed it into a dropdown menu, we have created another class called Data. This class is able to create a connection between the Dropbox and DisplayCSVData, ensuring an extra level of validation to check if there is an associated file with that dataset. If there is not, an alternative message will be available, and if there is, it will create an instantiation to DisplayCSVData passing parameters fileName and filePath.

VI. Conclusion

In terms of our overall design for our platform, the website offers streamlined access to the City of Windsor's datasets. Our user-friendly website built with React.js allows the user to filter data by selecting preferred categories, display and filter data, and download data. Currently, the core architecture and most main functionalities are implemented, with routing implemented for users to access all pages from the home page. The current deployment of the website includes most of its essential functions, although some limitations exist. These limitations involve incomplete functionality for the login screen, a non-operational search engine, and restricted options for selecting categories.

Implementing the functionalities mentioned above offers several benefits. A search engine enables users to directly search for datasets by name, providing quick access to specific datasets, especially as the dataset collection grows. This becomes particularly useful when browsing through categories becomes insufficient for the user's needs. With a larger dataset collection, offering more filtering options, such as filtering datasets by specific years through checkboxes, would expedite the search process. Additionally, implementing a properly functional login screen would allow users to keep track of their recent data, enhancing security by encapsulating their data within their account.

Despite these limitations, the website effectively achieves its main objective of enabling users to efficiently filter and find relevant information. Despite these challenges, we have largely succeeded in accomplishing what we set out to do in our iteration plan.

To recap our website, we built an open data portal using React.js allowing for an object oriented design that handles dynamic and interactive elements. We used many different libraries for the creation of our platform. Some included, Universal library for cookies, Lucide React for icons, React Router library for page navigation, Papa Parse for csv processing, and React Testing Library. In addition, the website includes the following pages: home, login, multi data view, and data overview pages. Each page has their respective task; for example, the multi data view page is responsible for filtering and displaying the list of datasets on the screen. Each of these respective pages have their own classes to specify their logic and rendering. Also, all of these class components come together to create the cohesive data portal that we ended up with.

Our object oriented design also utilises GRASP patterns to classify classes for certain roles in the system design. For example, the App class acts as the controller as it centralizes the control logic in a single entity and renders anything the user wants to display. The Classification class itself utilizes many GRASP patterns, like acting as Information Expert and provides low coupling. Lastly, High Cohesion is used in the DisplayCSVData class since it is dedicated to only the task of parsing, loading, and displaying the data.

Overall, our platform offers efficient access to City of Windsor's datasets, but there are areas requiring improvement, such as the search engine and filtering options. Nevertheless, our design efficiently achieves its goal of enabling efficient data interaction for users.

VII. Future Works

Going forward, our team is excited to refine and expand on our open data portal. The next iteration of our web application will focus on the following key areas:

- Enhancing Search Capabilities: Creating a more advanced search engine within our application allows for users to find data more efficiently. Introducing keyword search and filtering data based on dataset attributes.
- User Accounts: Introducing user accounts helps offer a unique experience for every user, catering to their needs and capabilities. Here, users will be able to save datasets to view later, access their search history, and receive recommended data sets based on the user's historical interests.
- Expanding Data Filtering Options: By expanding on the current filtering options, we can incorporate a wider range of datasets. Ensuring that the further growth or complexity of new data sets has the resources available for their implementation in our web application.
- Mobile Responsiveness: We will ensure the current web application we have developed presents the same capabilities and accessibility across all devices. This will start with optimizing for a mobile-friendly user interface that complies with web accessibility standards.
- Engagement With City of Windsor Government: By presenting our web application to City of Windsor government officials, we can better align our goal with the city's needs. This will help us get feedback, which will help refine our platform for the usage that is required. Ultimately, allowing for city-wide implementation as a user-friendly replacement to the current data portal.

By addressing these key areas of future work, we can solidify our web application as a key resource for the city. A comprehensive and adaptable data portal that meets current and future city needs.

VIII. Appendix A - Group work

A. Project Management Techniques

We utilized Jira as the primary project management tool for task allocation, tracking progress, and managing sprints. This one was accomplished by publishing our user activity (referred to as "epics" in Jira) in order to keep track of our tasks and those of our team members. We conducted sprint planning sessions to prioritize tasks from the backlog and decided on achievable goals for each sprint. This allowed for a more efficient dynamic between group members, as tasks could be completed regularly and any issues arising handled accordingly. Paired with Jira, we also used

Discord as our main means of communication. We held regular meetings, and messaged the group chat frequently in order to ensure task quality and completion.

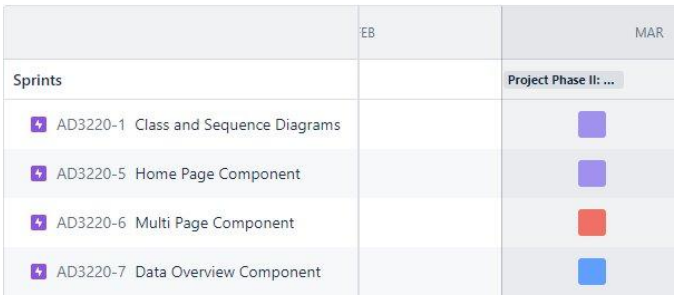

Fig. 1 Outline of Jira sprints for iterations

B. *Version Control*

As our software version control, we leveraged Git and hosted the project repository on GitHub. We utilized branches for feature development, bug fixes, and experimentation. Also, we regularly committed changes to the repository and used pull requests for code review before merging changes into the main branch.

C. *Testing Methods*

Using React's testing library facilitated unit testing, as we were able to test individual React components in isolation, mocking any dependencies they might have had. Essentially, it allowed us to test if components had rendered correctly and handled different props and states as expected. This library also allowed us to perform isolation testing, as we are able to test how components interact with each other within the application. Additionally, it allowed us to simulate user interactions and verify that the application behaved correctly in response.

```
PASS src/multi-data-view/filecomponents.test.js
PASS src/App.test.js
PASS src/HomePage/VerticalDataList/VerticalDataList.test.js
PASS src/multi-data-view/multidataview.test.js
PASS src/HomePage/ImageCarousel/HomeCarousel.test.js
PASS src/HomePage/VerticalDataList/VerticalDataListItem.test.js
PASS src/HomePage/HomePage.test.js

Test Suites: 7 passed, 7 total
Tests:       10 passed, 10 total
Snapshots:   0 total
Time:        5.465 s, estimated 6 s
Ran all test suites.

Watch Usage: Press w to show more.
```

Fig. 2 Screenshot of test results from React Testing Library

IX. Appendix A - UML Diagrams

In the inception and iteration phases, we created and refined our UML diagrams to understand what we were trying to achieve and how. It started with a use-case diagram, which we made during the Inception phase. This allowed us to understand how the different use-cases connected and to visualize which users could do what.

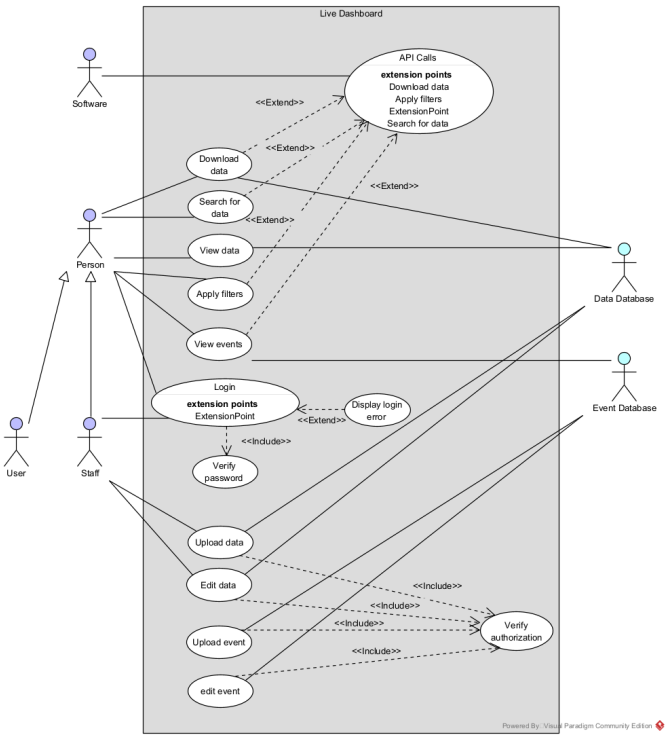

Fig. 1 Use case diagram from inception

Then, in the iterations, we created and refined class and sequence diagrams. This allowed us to understand what the different parts of our system were and how they interacted, which we used as a starting point for implementation. Below is the progression of our class diagrams and then our sequence diagrams.

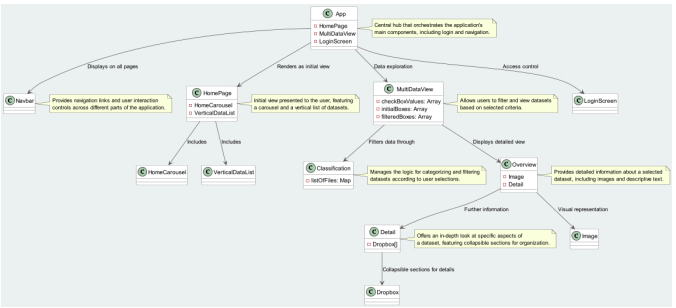

Fig. 2 Early class diagram

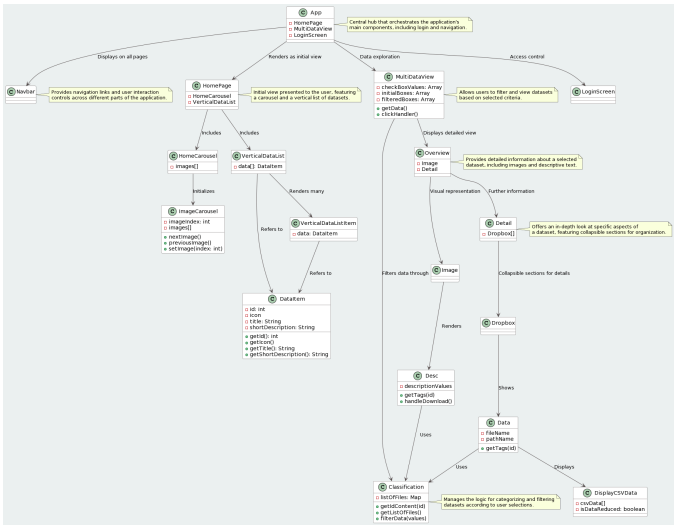

Fig. 3 Class diagram from the end of the project

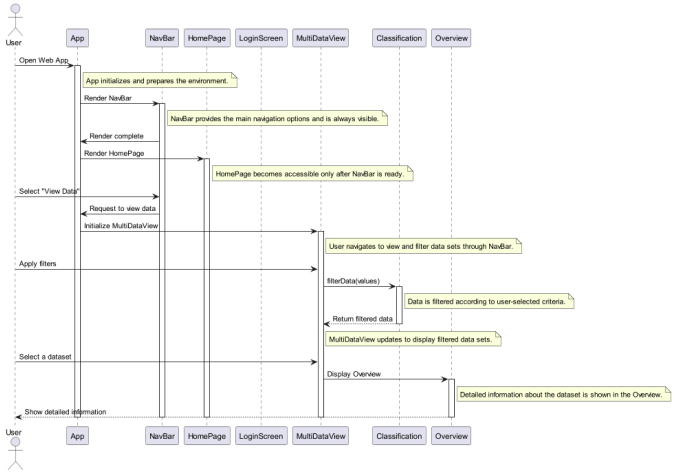

Fig. 4 Early sequence diagram

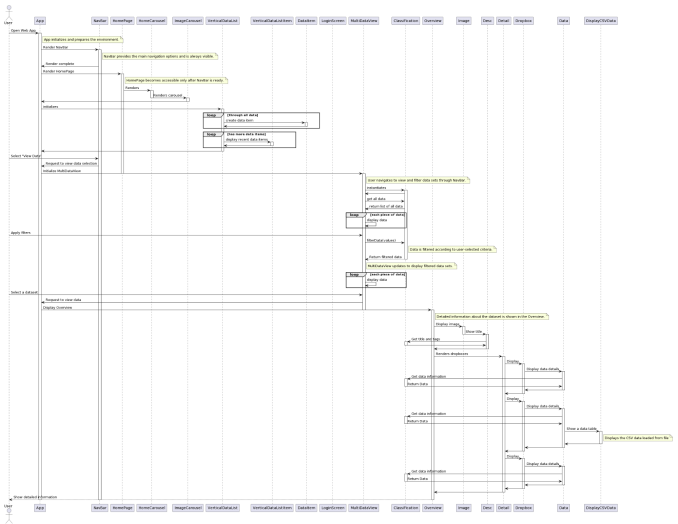

Fig. 5 Sequence diagram from the end of the project

Acknowledgment

We extend our gratitude to the City of Windsor for developing the open data portal, which provides accessible datasets that we have utilized for our website.

We would also like to acknowledge every member who worked on this project. The team's tasks were divided as follows, with Rocio focusing on implementing the multidata view functionality; Eli focusing on the home page, app routing, and repository management; Miran displaying .csv files and downloading functionality; Mark focusing on the implementation and development of documentation, UML diagrams, and maintaining project management tools; Keerat focusing on the implementation and development of the login page; and Moussa on the implementation and development of the navbar.

We also would like to acknowledge Meghan Cadarette, who graciously provided photography of Windsor for the project.

References

[1] Lucide, "Lucide React," Lucide. https://lucide.dev/guide/packages/lucide-react

[2] City of Windsor, "Open Data Catalogue," City of Windsor. https://opendata.citywindsor.ca/

[3] npm, "Papa Parse," npm. https://www.npmjs.com/package/papaparse

[4] React, "React Documentation," React. https://react.dev/

[5] React Router, "React Router Documentation," React Router. https://reactrouter.com/en/main

[6] Testing Library, "React Testing Library," Testing Library.

[7] https://testing-library.com/docs/react-testing-library/intro/

[8] npm, "universal-cookie," npm. https://www.npmjs.com/package/universal-cookie