Group 13 Bays: Project Milestone 1 Report

SFU Toolbox

Egemen Guney - 301560582

Yifei Jia - 301440620

Meiirim Zholmukhan - 301635123

Moc Wang - 301450080

Link to GitHub repository: https://github.com/CMPT-276-FALL-2024/project-13-bays

TABLE OF CONTENTS

APIs

Translink API

- **Trip Update**: https://gtfsapi.translink.ca/v3/gtfsrealtime?apikey=[ApiKey]
- **Position Update:** https://gtfsapi.translink.ca/v3/gtfsposition?apikey=[ApiKey]
- **Service Alerts:** https://gtfsapi.translink.ca/v3/gtfsalerts?apikey=[ApiKey]

SFU API

- **SFU Library Hours**: https://api.lib.sfu.ca/api/hours/3/summary
- **SFU Library New Books**: https://api.lib.sfu.ca/api/newbooks/browse
- **SFU Weather**: https://api.lib.sfu.ca/api/weather/current
- **SFU Course**: https://www.sfu.ca/bin/wcm/course-outlines?2024/fall/cmpt/276/d200

Users will be able to view their course schedules and details in one place. The site will allow students to easily check upcoming classes, find classroom locations, and read course descriptions without navigating multiple platforms.

SFU Toolbox will display the **current weather conditions** and forecasts, allowing users to plan their day, dress accordingly, and decide whether to bring rain gear. Users will be able to **check library hours** directly within SFU Toolbox, so they know exactly when the library is open for studying, borrowing books, or using other services. We plan to showcase **new arrivals in the library**, allowing users to discover the latest books and resources.

We will be using the Translink API, the features will include,

- **The Trip Update** feature provides real-time updates for active transit routes, including any changes to bus schedules or delays.
- **Position Update** delivers live tracking of bus locations, showing users the exact position of buses on their route.
- **Service Alerts** notify users of any significant issues affecting transit services, such as cancellations, route disruptions, or other alerts that impact transit routes. SFU Toolbox

will send alerts to users if there are disruptions on their chosen routes, allowing them to plan alternative routes if necessary.

Features

SFU APIs

Course Schedule Overview: Displays a user's course schedules, including class times, classroom locations, and course descriptions. Students can quickly view their schedule and upcoming classes in one place without needing to navigate multiple platforms. Expanded functionality to include brief course descriptions for added context on each course.

Weather Forecast: Provides current weather conditions and forecasts for the day or week. Helps users plan their day, dress appropriately, and decide if they need rain gear, especially given Vancouver's rainy climate. Initially planned as a basic forecast feature, it now includes real-time weather updates and multi-day forecasts for more accurate planning.

Library Hours and New Arrivals: Displays current library hours and highlights new arrivals or recently added resources. Students can quickly check if the library is open and browse new materials, keeping them updated on the latest resources. The feature now includes both hours and new arrivals, emphasizing a more dynamic resource discovery for students.

Translink API

Trip Update: Provides real-time updates on transit routes, showing any delays or changes to bus schedules. Helps students plan their trips to and from campus with accurate and up-to-date transit information. No changes from the initial proposal, this feature remains essential for timely updates.

Position Update: Delivers live tracking of bus locations, showing users where buses are in real time. Allows students to monitor bus locations on their route, minimizing waiting times and enabling efficient planning. No changes from the initial proposal.

Service Alerts: Notifies users of any major transit disruptions, such as route cancellations or delays. Provides alerts for significant service disruptions, allowing students to plan alternative routes and avoid delays. Enhanced to allow users to set preferred routes for receiving customized alerts, making it more user-specific.

Mid-fidelity Prototype

Link to the prototype:
https://www.figma.com/proto/sk7unm0Sp6suhKsmeIgKP1/Mid-fidelity-prototype?node-id=16-54&node-type=frame&t=DnAUAQ7vBmfT5X1i-1&scaling=scale-down&content-scaling=fixed&page-id=0%3A1&starting-point-node-id=1%3A2&show-proto-sidebar=1
The prototype was developed using Figma. It demonstrates the preliminary design of the application interface. On the main page there is a toolbox that contains the options "Room Finder" and "Transit Information" as the "drawers" of the toolbox. When opened, each of the drawers provides an overview of what the user can expect while using the corresponding functions. Furthermore, there are links provided to specific pages that will be developed soon, and accessible in the near future. There is a "Return" button if the user wants to go back to the main page, and open another drawer.

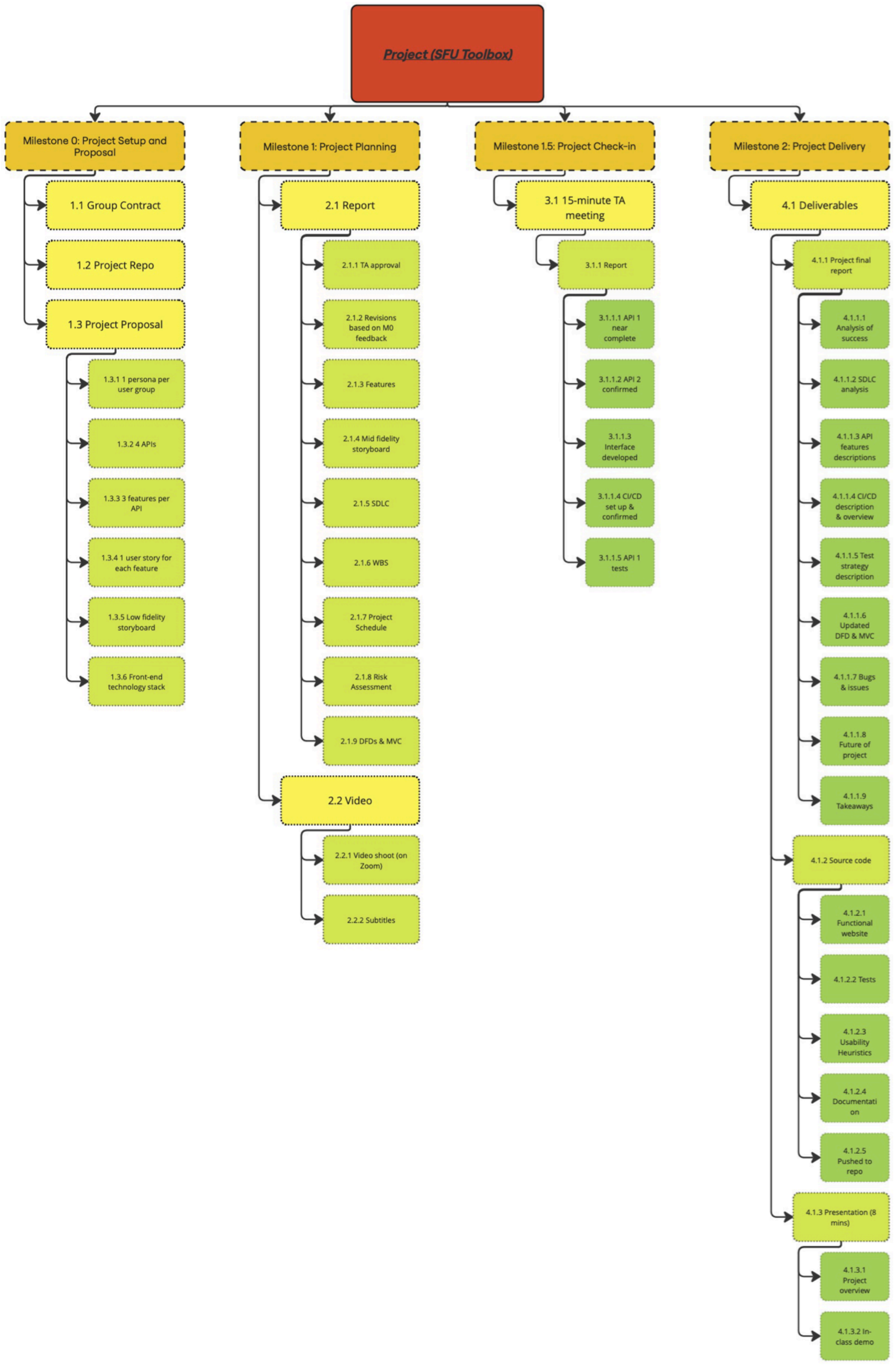
Our SDLC Methodology

We chose a combination of Scrum (Agile)-Kanban (Agile) hybrid (sometimes referred to as 'Scrumban') for our software development life cycle methodology approach. This decision is based on 2 important reasons:

1. We want a predictable and iterative process of development (Scrum). We have been and will be accomplishing this by:
    a. Having a well planned work structure at the beginning of each sprint/cycle and reviewing for completeness and quality at the end of each sprint/cycle.
    b. Having daily stand-ups on Discord and having at least one weekly in-person meeting(s).

2. We will be mainly using GitHub Projects for tracking the progress of the project, among other scheduling applications. We will be collaboratively managing each task, helping each other out. We also want to prevent the project from having bottlenecks.

WBS (Work Breakdown Structure)


Project (SFU Toolbox)
Milestone 0: Project Setup and Proposal
1.1 Group Contract
1.2 Project Repo
1.3 Project Proposal
1.3.1 1 persona per user group
1.3.2 4 APIs
1.3.3 3 features per API
1.3.4 1 user story for each feature
1.3.5 Low fidelity storyboard
1.3.6 Front-end technology stack
Milestone 1: Project Planning
2.1 Report
2.1.1 TA approval
2.1.2 Revisions based on M0 feedback
2.1.3 Features
2.1.4 Mid fidelity storyboard
2.1.5 SDLC
2.1.6 WBS
2.1.7 Project Schedule
2.1.8 Risk Assessment
2.1.9 DFDs & MVC
2.2 Video
2.2.1 Video shoot (on Zoom)
2.2.2 Subtitles
Milestone 1.5: Project Check-in
3.1 15-minute TA meeting
3.1.1 Report
3.1.1.1 API 1 near complete
3.1.1.2 API 2 confirmed
3.1.1.3 Interface developed
3.1.1.4 CI/CD set up & confirmed
3.1.1.5 API 1 tests
Milestone 2: Project Delivery
4.1 Deliverables
4.1.1 Project final report
4.1.1.1 Analysis of success
4.1.1.2 SDLC analysis
4.1.1.3 API features descriptions
4.1.1.4 CI/CD description & overview
4.1.1.5 Test strategy description
4.1.1.6 Updated DFD & MVC
4.1.1.7 Bugs & issues
4.1.1.8 Future of project
4.1.1.9 Takeaways
4.1.2 Source code
4.1.2.1 Functional website
4.1.2.2 Tests
4.1.2.3 Usability Heuristics
4.1.2.4 Documentati on
4.1.2.5 Pushed to repo
4.1.3 Presentation (8 mins)
4.1.3.1 Project overview
4.1.3.2 In-class demo

Link to work breakdown structure:

https://miro.com/app/board/uXjVLMSsPCY=/?share_link_id=750178237722

Project Schedule, Milestones, and Deadlines

Milestone 1:

1. Report
    a. TA Approval
    b. Revisions based on Milestone 0 feedback
    c. API features decided
    d. Medium fidelity prototype
    e. SDLC approach
    f. Work breakdown structure
    g. Project schedule
    h. Risk assessment
    i. Data flow diagrams and MVC
2. Video
    a. Video shoot on Zoom (individually)
    b. Subtitles

Milestone 1.5:

1. 15 minute TA meeting (check-in)
    a. Report (one pager)
        i. API #1 near complete (due Friday November 8)
        ii. API #1 tests (due Tuesday November 12)
        iii. API #2 confirmed (due Monday November 11)
        iv. Interface developed (due Friday November 8)
        v. CI/CD pipeline set up & confirmed (due Friday November 8)

Milestone 2 (Delivery) **[Tentative]**:

1. Project final report
    a. Analysis of success (due Monday November 25)

    b. SDLC analysis (due Sunday November 24)
    c. API features descriptions (due Thursday November 21)
    d. CI/CD description & overview (due Sunday November 24)
    e. Test strategy description (due Sunday November 24)
    f. Updated DFD & MVC (due Thursday November 21)
    g. Known bugs & issues and documentation (due Sunday November 24)
    h. Future of project (due Tuesday November 26)
    i. Takeaways from the project (due Tuesday November 26)
2. Source code
    a. Functional website (due Thursday November 21)
    b. Tests (due Sunday November 24)
    c. 10 usability heuristics (due Thursday November 21)
    d. Documentation (due Tuesday November 26)
    e. Pushed to repository (due Tuesday November 26)
3. In class presentation (8 minutes)
    a. Project overview (due Tuesday November 26)
    b. In class demo (due Tuesday November 26)

Risk Assessment

Low-Risk

API Rate Limits

- *Mitigation:* Implement cache to reduce the frequency of API calls, minimizing the risk of reaching rate limits.

Unfamiliarity with Javascript/Next.js

- *Mitigation:* Create a shared knowledge base and conduct pair programming sessions to help team members quickly gain proficiency with Next.js.

Complex UI Components or Animations

- *Mitigation:* Use libraries or modules that can handle these tasks more efficiently. For instance, we could integrate Vue for isolated micro-interactions.

Inconsistent UI Styles

- *Mitigation:* When using multiple libraries or frameworks, differences in UI styles can make the application look disjointed. To address this, we'll create a shared design specification with standardized styles, colors, and spacing that each component can reference.

Visual Clutter

- *Mitigation:* With multiple features displaying various types of information, there's a risk of overwhelming users with too much content on a single screen. Prioritize a minimalist design by strategically grouping related information, using collapsible sections, and employing whitespace.

Medium-Risk

Data Latency

- *Mitigation:* To avoid delays caused by fetching data from multiple sources, load data asynchronously, allowing different parts of the app to retrieve information at the same time. Prioritize loading essential information first, so users can see the most critical updates right away, even if other data is still on its way.

Inconsistent Data Formats

- *Mitigation:* Each API may deliver data in a different format. Set up standardized parsers and validators within the API Integration Layer. Make sure all incoming data is handled uniformly, reducing any formatting issues and making data processing smoother across the app.

Time Management

- *Mitigation:* The time allocated for coding and documenting this project can overlap with exams, assignments, and projects from other courses, and we have already encountered time conflicts for Milestone 1. To manage this, we decided to implement numerous short sprints using the agile method, allowing us to make the most of our fragmented time. Our primary focus will be on completing core functionalities rather than spending too much time on visuals and design in the initial stages. With this approach, we have established a priority queue for each broken-down task.

High-Risk

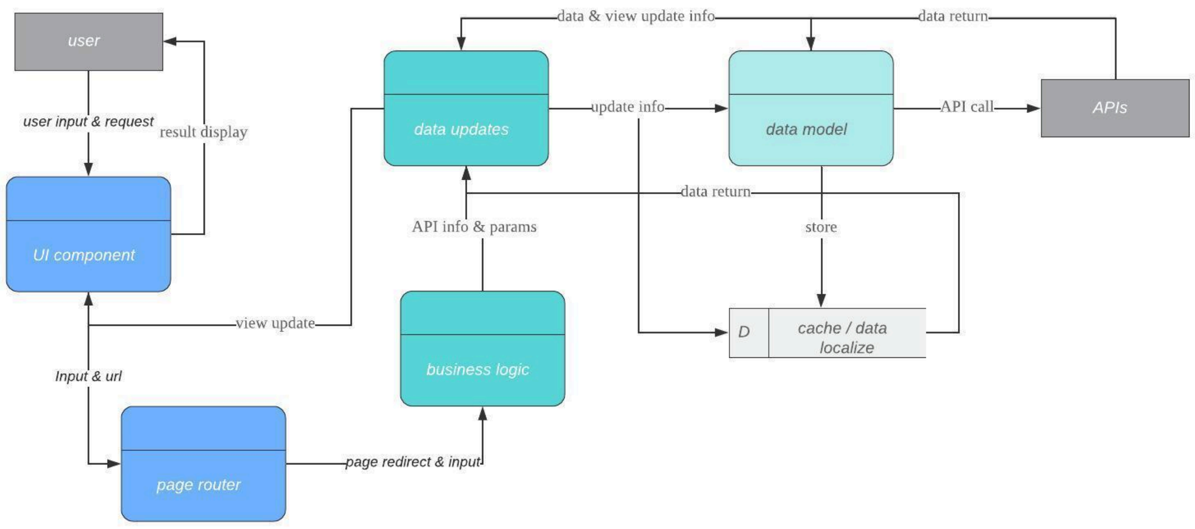
Third-Party API Malfunction

- *Mitigation:* Implement fallback messages or use cached data to display in case of real-time API data being unavailable. We also have a backup list of scraper APIs to serve as data source when the APIs we plan to use are problematic or the information is too insufficient.

Incompatible Technologies

- *Mitigation:* We've selected a variety of technologies for this project, each with its own dependencies. This can sometimes lead to dependency conflicts or incompatible code. Additionally, collaborating remotely in different environments increases the risk of configuration conflicts. To address these potential issues, we have backup frameworks in place, such as React.js as an alternative to Next.js, in case the more advanced and complex tools prove challenging to integrate. We’ve also created a specification file that outlines all required modules, packages, and dependencies for the project, standardizing the setup to minimize configuration conflicts and ensure consistency across development environments.

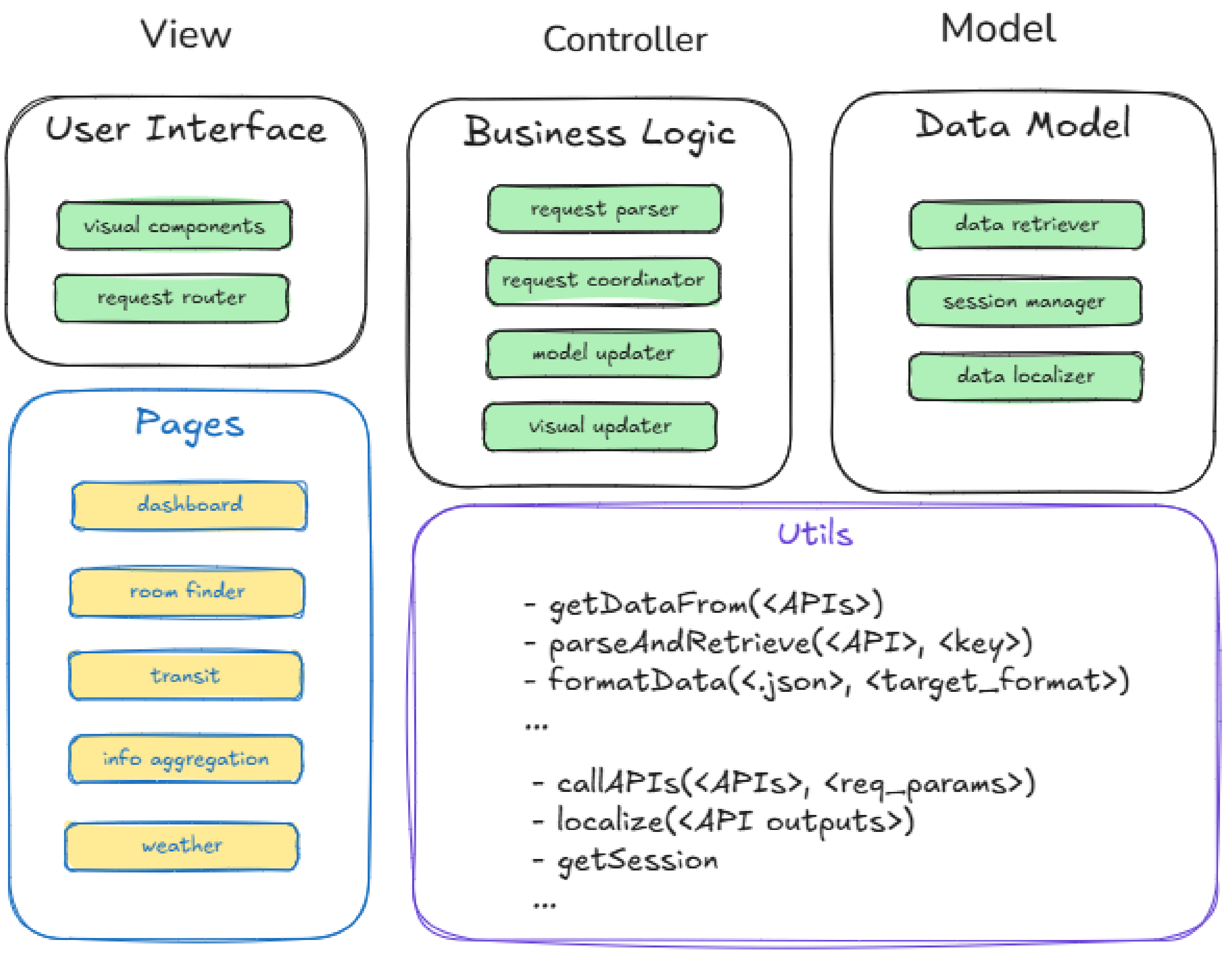
DFDs (Data Flow Diagrams)



This Data Flow Diagram illustrates how data moves through the SFU Toolbox application, from user interactions to API responses and back to the UI. It starts with the user entering a request, like finding an available study room or checking transit times, through the main **UI component**. The input is then directed by the **page router**, which ensures users are navigated to the relevant section of the app. The **business logic** module processes this request, determining which data sources are required and preparing parameters for the **data model**. Acting as the application's core, the data model coordinates API calls to external sources, such as SFU's Room Finder, Translink, or weather APIs, and receives the data in return.

Once the data is gathered, it moves to the **cache/data localizer** module, which manages caching and localization, allowing frequently accessed data to load quickly and adjusting information based on the user's context (like location or recent queries). The processed and cached data then flows into the **data updates** module, where it is organized and synchronized with the view layer to ensure real-time, accurate displays. Finally, the updated data returns to the **UI component**, where it is presented to the user in a clear and organized format, completing the cycle.

MVC Model



Model

Data handling is centralized through modules like the *data retriever*, *session manager*, and *data localizer*. The *data retriever* module is responsible for fetching information from various APIs (e.g., SFU APIs, Translink, Google Maps) and ensuring that each piece of data is available for the application. The *session manager* maintains the state of user interactions during a session, such as recent searches or preferences, without requiring a backend. The *data localizer* localizes key information about the users. Together, these modules provide an accurate and personalized data feed to the Controller, ready for processing.

Control

The Controller contains business logic modules, including *request parser*, *request coordinator*, *model updater*, and *visual updater*. The *request parser* translates user actions, like selecting a bus stop or finding a study room, into specific requests for the Model. The *request coordinator* then determines which data sources to query and manages interactions with the multiple Models, utilizing tools such as *getDataFrom(<APIs>)* or *callAPIs(<APIs>, <req_params>)* in the Utils Package (self-defined tool class) for performing data manipulation. The *model updater* dynamically updates the Model as new data is received, ensuring that the application's state reflects real-time changes. Finally, the *visual updater* communicates changes to the View, triggering updates to display new data to users.

View

The view presents information in an accessible, intuitive format via *visual components* and the *request router*, which directs users to specific pages such as dashboard, room finder, transit, info aggregation, and weather. Each page contains visual components tailored to specific user needs: for example, the dashboard consolidates essential information, and the room finder displays available study spaces. The request router manages user navigation across these pages. Visual components within each page are responsive to updates from the Controller's visual updater, ensuring that users see the latest information without needing to refresh or navigate away.