SFU

# CMPT 276 - Project Report
# GitGood

015 - Mountains
Amar Koonar - 301600620
Thomas Brigham - 301637377
Edward Lee - 301637580
Matthew Tsui - 301597949

SFU

# Finalized APIs

1. **GitHub REST API** - GitHub's REST API will allow our website to display and utilize data from GitHub's massive database of open-source information. This data will be used on our website in many ways, including readme lookup and profile stat comparisons. It will also be used to search GitHub repos for good first issues to contribute to. This API will serve as an all-in-one hub for GitHub's information, empowering users of our website with access to relevant data. These uses include:
    - **Good first issue finder** - By using the GitHub API to search for issues tagged with “good first issue” and “help wanted” in a specified coding language, users can find issues to contribute to in languages they know. This will enable them to begin making GitHub contributions.
    - **ReadMe Viewer -** By searching for a user and a repo, a live, real-time, fully rendered markdown file will be rendered on the website from this repo. This will allow users to view good readmes and make local changes to them to practice editing readmes.
    - **Profile Comparison** - Gamifying learning is great for enhancing learning. Git-Goods profile comparison will allow friends to compare profile stats, encouraging friendly competition between peers. This will work by allowing the user to query two separate GitHub user names; their stats will then be returned and rendered to the client.
2. **GitHub-hosted AI Inference API** - GitHub's hosted AI API is an AI service hosted by GitHub that allows users to experiment with different AI models.  This will allow our group to utilize popular AI models like ChatGPT on our website. This AI will be used to help users pick out issues depending on their needs. It will also be utilized through a chatbot that will provide help on solving GitHub issues. As well as this it will help generate markdown for our users' GitHub repositories. With GitHub's hosted AI, users will be completing GitHub issues faster than ever.
    - **Issue Chatbot -** Using GitHub's hosted version of the ChatGPT 4 AI model, we will provide a simple chatbot that helps users with any GitHub issues they are struggling to solve. This will help people who are new to problem-solving and are unsure of where to begin.
    - **AI MD File Maker -** We will use AI to generate markdown for GitHub issues or readmes. This will work by allowing users to input information, then using the AI API to return a full, clear markdown file. If the user is not satisfied, they can request edits.
    - **AI Issue Selector -** One of our previous features allows users to generate a list of issues to choose from. We will add on top of this feature by adding an AI issue selector that will take in the list of options along with some information about the current user, and it will return an issue it believes is the best choice.

SFU

# Mid-Fidelity Prototype

## Static Version:

mid-fidelity-prototype.pdf

## Interactive Version (Prototype Mode Link):

https://www.figma.com/proto/12eJFtoqHAIAuohy4ztahp/Untitled?node-id=0-1&t=EQJgdUaVYQzpB4t7-1

## Interactive Version (Dev Mode Link):

https://www.figma.com/design/12eJFtoqHAIAuohy4ztahp/Untitled?node-id=0-1&m=dev&t=EQJgdUaVYQzpB4t7-1

**READER'S NOTE:**

Due to some unforeseen limitations of Affinity Designer, the Mid-Fidelity Prototype was developed twice. The first version is static, but includes detailed annotations describing the user flow. The second version is interactive click-through functionality. There are some minor design differences resulting from the Affinity → Figma redesign, and some small final additions (i.e., the second version includes some finalizations like the logo & about page). But the two versions are supposed to be viewed together to get a complete idea of the prototype.

# SDLC Model

**SDLC Model Chosen**: Agile - Kanban
**Link to Kanban**: Here
**Reason:** Considering the time window of the project to implement and test, and our level of proficiency with our tech stacks, we have decided to choose the Agile SDLC Model with the Kanban framework. Agile is flexible and has the ease to apply the feedback in the upcoming iteration, and we have determined to develop 6 distinct and unrelated features with 2 APIs, which would require immense feedback to be applied to the current or upcoming task as soon as possible. Therefore, we have concluded using Agile methodology with Kaban for our SDLC Model. Also, our familiarity with using GitHub Project Kanban Board throughout CMPT 276 has contributed to come up with this decision.

SFU

# Work Breakdown Structure (WBS)

1. **Planning**
   - 1.1. Analyse requirements
   - 1.2. Find API's and test them
   - 1.3. Setup repository
2. **Design**
   - 2.1. Design a low-fidelity prototype
   - 2.2. Brainstorm higher ideas for a higher fidelity prototype
   - 2.3. Design a high-fidelity prototype
3. **Coding/Development**
   - 3.1. Setup linter
   - **3.2. Frontend**
     - **3.2.1. Landing Page**
       - 3.2.1.1. Create a fixed navbar with links to other pages
       - 3.2.1.2. Create Footer
       - 3.2.1.3. Create a basic landing page
     - **3.2.2. Issue Selector Page**
       - 3.2.2.1. Create an input menu for the user
       - 3.2.2.2. Create an AI selector interface
     - **3.2.3. Profile Comparison Page**
       - 3.2.3.1. Create two input menus
       - 3.2.3.2. Create a graphic to display stats
     - **3.2.4. AI Chatbot Page**
       - 3.2.4.1. Create a chatbot input menu
       - 3.2.4.2. Create a chatbot output display
       - 3.2.4.3. Verify that the text will be displayed nicely inside the chatbox
     - **3.2.5. Readme Viewer Page**
       - 3.2.5.1. Create an input box for text
       - 3.2.5.2. Create a box to display the rendered MD
     - **3.2.6. MD Generator Page**
       - 3.2.6.1. Create an input box for text
       - 3.2.6.2. Create a box to display the rendered MD
   - **3.3. API Development**
     - **3.3.1. Github REST API**
       - 3.3.1.1. Create endpoints for functions
       - **3.3.1.2. Create Issue Finder Endpoint**
         - 3.3.1.2.1. Collect query information
         - 3.3.1.2.2. Query the GitHub API with specified info
         - 3.3.1.2.3. Return Info
         - 3.3.1.2.4. Display info on the frontend
       - **3.3.1.3. User Stat Endpoint**
         - 3.3.1.3.1. Collect user data.
         - 3.3.1.3.2. Query the GitHub API for user data in different ways
         - 3.3.1.3.3. Group up the data and return
         - 3.3.1.3.4. Display data on the frontend
       - **3.3.1.4. README Finder Endpoint**
         - 3.3.1.4.1. Collect data

SFU

3.3.1.4.2. Query GitHub API for the README of the specified repository
3.3.1.4.3. Return the markdown of ReadMe
3.3.1.4.4. Display MD on the frontend

**3.3.2. GitHub-hosted AI Inference API**

3.3.2.1. Create an endpoint boilerplate for functions

**3.3.2.2. Create an AI Chatbot Endpoint**

3.3.2.2.1. Collect input and context
3.3.2.2.2. Send data to the AI model
3.3.2.2.3. Clean response from the AI model and return
3.3.2.2.4. Display response on the frontend

**3.3.2.3. Create AI Issue Selector Endpoint**

3.3.2.3.1. Collect a list of GitHub issues and context on the user
3.3.2.3.2. Send data to AI model and get it to return best issue
3.3.2.3.3. Clean response and return
3.3.2.3.4. Display response on the frontend

**3.3.2.4. Create an AI-Powered Markdown Endpoint**

3.3.2.4.1. Create an endpoint that takes input
3.3.2.4.2. Send data to the AI model
3.3.2.4.3. Clean response and return it
3.3.2.4.4. Display response on the frontend

**4. Testing**

**4.1. Function Testing**

4.1.1. Verify APIs are returning correct data
4.1.2. Check that client-side components are working

**4.2. Usability Testing**

**4.2.1. Conduct User Testing**

4.2.1.1. Collect feedback on the usability of the application

**4.2.2. User Feedback**

4.2.2.1. Analyze feedback to find common issues or suggestions
4.2.2.2. Assign priority to issues based on severity

**4.2.3. Integrate Feedback**

4.2.3.1. Implement issues based on previously assigned priorities
4.2.3.2. Retest after issues have been fixed/solved

**4.3. Automatic Testing**

4.3.1. Set up integration tests for the frontend
4.3.2. Set up unit tests for API functions

**5. Deployment**

**5.1. Automate the build and testing process**

5.1.1. Ensure tests are run on the pull request and are passed before deployment

**5.2. Set up CI/CD pipeline**

5.2.1. Connect any API keys
5.2.2. Confirm that test cases pass before deployment

**5.3. Deploy**

5.3.1. Deploy code to production

**5.4. Documentation**

5.4.1. Create Release notes

SFU

# Project Schedule with Milestones and Deadlines

**July 1-5: First Sprint & Develop UI**
Internal Deadlines:

- July 2: Set up project file structure in Next.js & TailwindCSS with linter.
- July 3: Review and finalize **Milestone 1 - Report** for submission.
- July 4: Complete the basic UI layout for the website.
- July 5: Buffer day for debugging and improving basic UI layout.

July 6-10: **Beginning Development: Set up CI/CD and API Keys**
Internal Deadlines:

- July 7: Review and finalize **Milestone 1 - Video** for submission.
- July 8: Create .yml file in /workflows and connect Vercel tokens in the GitHub repository for CI/CD pipeline setup.
- July 9: Successfully logged API request responses.
- July 10: Buffer day to debug API request problems and CI/CD pipeline setup.

July 11-15: **API Functions & Integration With UI**
Internal Deadlines:

- July 11: Ensure API data is shown in the frontend with tests.
- July 12: Successfully connected API endpoints with the frontend.
- July 13: Integrate UI with API endpoints to finalize features.
- July 14: Buffer day to complete or debug features with API + UI integration.

July 16-21: **User Testing (unit testing, integration testing)**
Internal Deadlines:

- July 16: Create and ensure unit testing for each respective feature is successful.
- July 17: Create integration and ensure testing for each respective feature is successful.
- July 18: All automated tests run correctly.
- July 19: All features run successfully with API connections.
- July 20: Website UI successfully follows the 10 usability heuristics.
- July 21: Buffer day to debug and fix any potential bugs/API issues.

July 22-25: **TA Testing + Milestone 1.5**
Internal Deadlines:

- July 22: Review and finalize **Milestone 1.5** for submission.
- July 23: Fix and enhance the website based on TA feedback.
- July 24: Demo day for testing our website, save and store user feedback for next week.

- July 25: Buffer day for fixing any remaining bugs and UI issues from TA feedback.

July 26-29: **Implementing User Feedback + Finalization**
Internal Deadlines:

- July 26: Review user feedback from demo day and implement all necessary changes.
- July 27: Finalize frontend and backend of the website, ensure entire website layout is completed and ready.
- July 28: Successfully passed all automated tests and unit tests after website finalization.
- July 29: Buffer day for emergency fixes/improvements.

July 30 - August 1: **Documentation, cleanup, and deployment**
Internal Deadlines:

- July 30: Write documentation on the entire code.
- July 31: Clean up code, fix final bugs, and successfully deploy website without any issues.
- August 1: Buffer day for any remaining issues with code, documentation, cleanup, testing, and deployment.

August 2-6: **Milestone 2 and Final Presentation**
Internal Deadlines:

- August 2: Complete the first draft of **Milestone 2 - Report + Code.**
- August 3: Complete the first draft of **Milestone 2 - Presentation.**
- August 4: Review and finalize **Milestone 2 - Report + Code** for submission.
- August 5: Review and finalize **Milestone 2 - Presentation** for submission.
- August 6: Buffer day to review and finalize any part of Milestone 2 for submission.

August 7-8: **Submission Day**
Internal Deadlines:

- August 7: Final Presentation video completed and uploaded.
- August 8: Submission - Project complete.

SFU

# Risk Assessment

## Low-Risk:

**Risk:**
Stranger Danger
**Explanation:**
Working with new people can be awkward due to a lack of trust and not being comfortable with each other.
**Mitigation:**
- Encourage communication with weekly meetings and a group chat.
- Clearly define roles and responsibilities so everyone knows their tasks.

**Risk:**
Styling Inconsistencies
**Explanation:**
When there are so many people working on a project with different ideas, it may be difficult to have the same styling designs for each page.
**Mitigation:**
- Have the Project Manager double-check the work to see if it meets the requirements.

**Risk:**
Merge Conflicts
**Explanation:**
Having some of our members who are new to GitHub can lead to merge conflicts or other issues.
**Mitigation:**
- Good communication and following proper Git hygiene.

**Risk:**
Grammar Mistakes
**Explanation:**
When working an extended period of time on a project, it is easy to have minor grammar issues.
**Mitigation:**
- Quality inspections are conducted to ensure that everything is up to standard.

**Risk:**
API Limited Calls
**Explanation:**
When testing or using the API with limited uses per day, we may run out of calls.
**Mitigation:**
- Test early, so even if the limit is reached, we have time to test the next day.
- Plan tests to utilize the API so we don’t make unnecessary calls and waste calls.

SFU

## Medium-Risk:

**Risk:**

Error in Code

**Explanation:**

The need to constantly check if the code is error-free can be very difficult.

**Mitigation:**

- Utilize GitHub Actions and implement a .yml file to catch errors.
- Have a QA test and catch bugs.

**Risk:**

Team Scheduling Conflicts

**Explanation:**

Everyone has different schedules and responsibilities, and managing and efficiently using our time will be a challenge.

**Mitigation:**

- Have weekly checkup meetings (scrum), ask for help when needed, and keep each other accountable.

**Risk:**

Team Conflict

**Explanation:**

Teams can sometimes disagree, and this may lead to conflict due to different concerns.

**Mitigation:**

- Utilize different conflict styles to try to come to an agreement.
- If conflict can not be solved internally, ask a third-party person (TA, someone with knowledge, or others) for help, depending on the scenario.

**Risk:**

Poor GitHub Data

**Explanation:**

When comparing GitHub data or the README, there could be errors if the user has no data to compare.

**Mitigation:**

- Use try/catch error handling if the user does not have a README file.
- Add messages for errors.

**Risk:**

Variable Inconsistencies

**Explanation:**

With multiple software developers, each person can use different variable names.

**Mitigation:**

- Establish a naming convention
- Document variable names

SFU

## High-Risk:

**Risk:**

Incomplete Project

**Explanation:**

Our project could be overambitious with all the ideas and features. This could result in not having enough time to finish and produce a working project with all the promised features and ideas.

**Mitigation:**

- Come up with a minimum viable product, so we know what to prioritize.
- Schedule tasks with self-imposed deadlines to meet deadlines.

**Risk:**

Poor Time Management

**Explanation:**

Everyone has their schedules and responsibilities; if the little time we do have is not managed properly, we may not finish on time.

**Mitigation:**

- Have weekly check-ins to see where everyone is at.
- Schedule out each task with a deadline so that everything can be finished on time.

**Risk:**

No Prior Experience

**Explanation:**

Some members in our group may not have experience with AI or API integration, so they will need to spend time learning.

**Mitigation:**

- Spend one week on YouTube tutorials to learn.
- Add one feature at a time, instead of doing them all at once.

**Risk:**

AI/API downtime or access restriction

**Explanation:**

If there are any issues with the AI or API, like downtime, our site would not function.

**Mitigation:**

- Test API responses earlier to know if the features work.

**Risk:**

Insufficient testing

**Explanation:**

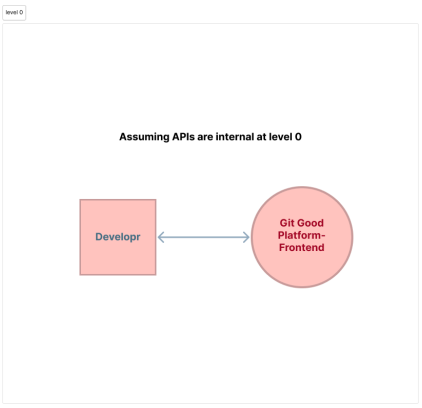
Without proper testing, bugs and errors could occur in the final product.

**Mitigation:**

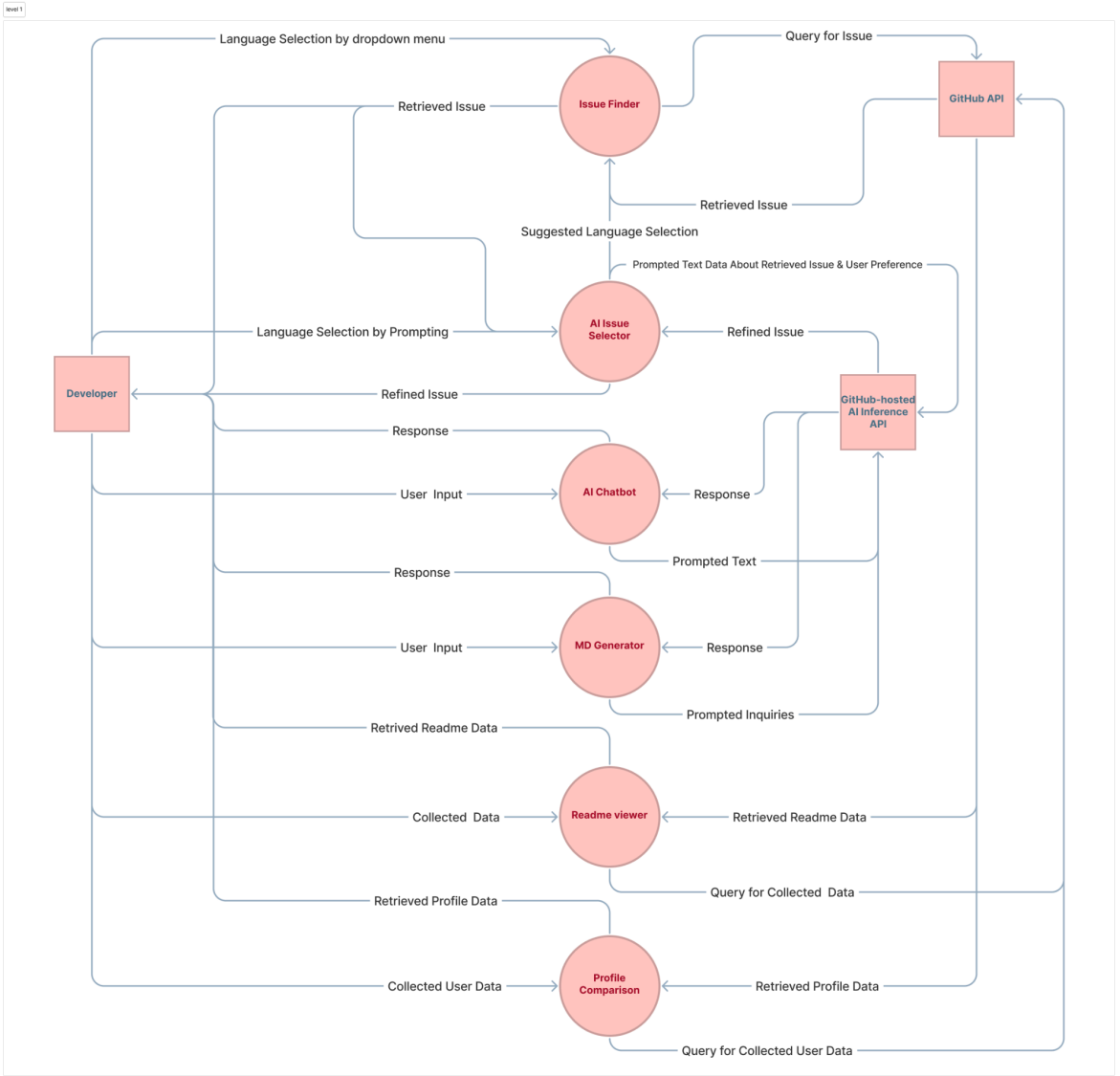
- Have an assigned QA to test for bugs and errors.

# Data Flow Diagrams

## Level 0




## Level 1




**Link:**
https://www.figma.com/design/T107EtUdwt6HUX9XT19Y4Z/Data-Flow-Diagram-Template--Community-?node-id=202-133&t=cuVWBMyDNB8ftivV-1

SFU

# MVC model diagram

SFU

# Appendix

## Group Contribution List

### Amar:

- Collectively decided on a project idea, APIs and tech stack by working on the project proposal.
- Organized and coordinated tasks with other members by creating GitHub issues and setting up a GitHub repository.
- Created the project Schedule.
- Wrote the description of the APIs for the project report
- Developed the Work Breakdown Structure.
- Tested APIs to make sure they will work as intended.

### Edward:

- Collectively decided on a project idea, APIs and tech stack by working on the project proposal.
- Wrote User Stories to identify and fully understand the concerns of the users for each feature.
- Identified personas for each user group.
- Organized Data Flow & Implemented Data Flow Diagram.
- Initiated discussion with teammates on choosing the SDLC model.
- Interpreted about the SDLC mode we chose and its brief explanation.

### Matthew:

- Collectively decided on a project idea, API's and tech stack by working on the project proposal.
- Identified the User Groups that would be utilizing the project.
- Created the Changelog.
- Documented everyone's contributions.
- Identified potential risks of the project and mitigation strategies for each risk.

### Thomas:

- Collectively decided on a project idea, API's and tech stack by working on the project proposal.
- Designed the Low-Fidelity Storyboard
- Designed a static Mid-Fidelity Prototype (Affinity/Adobe)
- Designed an interactive Mid-Fidelity Prototype (Figma)
- Created the MVC diagram
- Hand-designed an original GitGood logo from scratch

SFU

# Change Log

| Change No. | Change Name | Description of Change | Date of Change |
|---|---|---|---|
| 1 | ReadMe Viewer | Changed the ReadMe Viewer feature to no longer include AI | 06/22/2025 |
| 2 | Work Breakdown Structure | Broke the project down into smaller tasks to clearly identify tasks based on importance | 06/25/2025 |
| 3 | Schedule | Created a schedule outlining the timeline of the project with clear deadlines | 06/25/2025 |
| 4 | SDLC | Choose the SDLC model our group would use and wrote a description on why it is suitable for the project | 06/26/2025 |
| 5 | Identified Risks | Identified minor, major, and critical risks that will need to be addressed | 07/03/2025 |
| 6 | MVC Model | Designed a MVC Model for the project's structure | 07/03/2025 |
| 7 | Mid-Fidelity Prototype | Converted the low-fidelity storyboard into a mid-fidelity prototype utilizing Adobe and Affinity | 07/03/2025 |
| 8 | Created Changelog | Created and filled in the changelog to see changes made since milestone 0 | 07/04/2025 |
| 9 | Apendix | identified contributors and contributions, made a changelog, and addressed other information not in the report | 07/04/2025 |

# Additional Items

GitGood Logo