Development Plan

Table 1: Revision History

Date	**Developer(s)**	**Change**
April 4, 2025	Derek Li	Final Revision
October 28, 2024	Damien Cheung	Revision of 9
September 24, 2024	Emma Wigglesworth	Completion of 1, 2, 3, 4, 5, 6, 7, and final edits to all sections
September 21, 2024	Temituoyo Ugborogho	Completion of 8, Appendix
September 21, 2024	Damien Cheung	Completion of 9, Appendix
September 21, 2024	Derek Li	Completion of 10, 11

Capstone Project: McMaster Softball League Online Scheduling and League Management Platform

This development plan outlines the approach Team 6 will take to ensure the successful completion of our capstone project. The document covers key areas such as team roles, meeting schedules, and communication strategies to promote effective collaboration. It also details the tools, technologies, and workflows we will use to manage tasks, track progress, and maintain code quality. Additionally, the plan includes a breakdown of project milestones, scheduling, and potential risks. By following this plan, our team aims to stay organized, meet deadlines, and deliver a high-quality final product.

1 Confidential Information

The platform will handle some confidential information, such as players' email addresses and phone numbers. To protect user privacy, this information will only be visible to members of the same team and the league commissioner, who may need access for coordination and communication purposes. It is not accessible to other users or the public. We will ensure that sensitive data is stored securely and only shared when necessary to support team communication and platform functionality.

2 IP to Protect

The platform will not store or expose any sensitive IP (Internet Protocol) address information. There are no specific IP-related protections required beyond standard network security practices.

3 Copyright License

Our team will use the following MIT License.

4 Team Meeting Plan

The team will meet twice a week, with a strong emphasis on maintaining at least one in-person meeting whenever possible. Physical meeting locations and continuous discussion outside of meeting times will be communicated internally over text messaging or the team's designated communication platform (Discord).

Meetings will be organized as follows:

- **Progress Meeting**: Focused on updates, task delegation, and alignment on overall project goals (minimum 30 minutes).

- **Working Session**: Dedicated to collaborative tasks such as coding, debugging, and addressing technical challenges (minimum 1 hour).

We will meet with our industry advisor biweekly to discuss progress and challenges. In-person meetings with our advisor will always be prioritized. During weeks when a deliverable is due, this meeting shall take place at least 2 days before the deadline to allow for necessary feedback and revisions. In-person meetings will occur at Hatch when rooms are available for booking. Virtual meetings will be held over MS Teams. All meetings will be organized by email with the industry advisor with at least 24 hours of notice.

Meeting Structure:

- **Meeting Chair**: A rotating team member will serve as the chair for each meeting, responsible for maintaining structure and guiding discussions.
- **Agenda**: A predefined agenda will be prepared by the chair and shared with all team members at least 24 hours in advance. This will allow everyone to review and prepare for the meeting topics.
- **Minutes**: A designated notetaker (rotating role) will document key decisions, progress, and action items.

In case of emergencies or immediate concerns, impromptu meetings can be scheduled at short notice using text messaging or the team's designated communication platform (Discord). These meetings will be used to address urgent issues that cannot wait until the next scheduled session.

All team meetings will be planned and documented as GitHub Issues. These issues will include, at the minimum: the meeting agenda and attendees.

5 Team Communication Plan

Primary Communication Platforms:

Instant Messaging: The team will use an instant messaging group chat for casual discussions or time-sensitive communication.

Discord: The team will use Discord as the primary tool for daily communication, task updates, and file sharing. Different channels will be created for specific topics, such as:

- `#general`: For overall team communication.
- `#progress-updates`: For regular project updates and task completion.
- `#coding-issues`: For discussing code-related problems or bugs.
- `#advisor-meetings`: For coordinating and preparing for advisor meetings.

GitHub: GitHub will be the main platform for tracking code development and issues. Each team member will be responsible for regularly pushing code and using the issues board to log any bugs or features that need to be addressed.

Issue Tracking: All team members will log bugs, feature requests, and tasks as issues in GitHub. Each issue will include:

- A clear description of the problem or task.
- Labels to indicate priority (e.g., high, medium, low) and type (bug, documentation, etc.).
- A due date or milestone when applicable.
- Assigned member(s) responsible for its completion when applicable.

Meeting Summaries: After every meeting (team or with advisors), a summary of key points and decisions will be posted to Discord by the notetaker in `#advisor-meetings`. These summaries will also be stored in a shared Google Drive folder for easy reference.

Code Reviews: Pull Requests (PRs): Before any code is merged into the main branch on GitHub, team members will submit PRs for organization and traceability. Each PR must be reviewed and approved by two other members.

Decision Making: For decisions on the project (technical or otherwise), team members will communicate through Discord or instant messaging. If consensus isn't reached after discussion, a vote will be held, and majority decisions will move forward.

6 Team Member Roles

To ensure that responsibilities are clear and all aspects of the project are efficiently managed, the following roles will be defined for each stage of the project. These roles will rotate as needed to provide equal opportunities for team members to lead and contribute across different areas:

6.1 Project Manager

- Oversee overall project progress and ensure deadlines are met.
- Coordinate tasks and delegate responsibilities to team members.
- Monitor team morale and ensure smooth collaboration.

6.2 Team Liaison

- Act as the primary point of contact with the faculty.
- Distribute and document all communication between the faculty and team members via text messaging or in an appropriate channel on Discord.

- Respond to emails from faculty in a timely manner and ensure all statements made to the faculty are approved by the rest of the group.

6.3 Meeting Chair

- Lead and structure team meetings.
- Prepare and distribute the meeting agenda at least 24 hours before the meeting.
- Ensure meetings stay on topic and follow the agenda.
- Facilitate discussions and ensure all team members are heard.

6.4 Notetaker

- Take detailed notes during team and advisor meetings.
- Document key decisions, action items, and deadlines.
- Distribute meeting minutes promptly after each meeting in the appropriate Discord channel (within 24 hours).

6.5 Code Reviewer

- Review code submitted by team members via GitHub pull requests to ensure quality and consistency.
- Provide constructive feedback and suggest improvements to ensure best practices are followed.
- Ensure the code adheres to the team's coding standards and works as expected.

6.6 Documentation Lead

- Maintain and update project documentation, including the development plan, technical documentation, and user guides.
- Ensure that all team members contribute to the documentation as needed.
- Lead final reports and project summaries required for deliverables.

7 Workflow Plan

To ensure efficient collaboration and seamless project development, the following workflow processes will be adopted:

7.1 Git Workflow

The team will follow the **Git Feature Branch Workflow**, but instead of "feature" branches, we will refer to them as **"task branches"** to better reflect the variety of work being done (e.g., adding documentation, fixing bugs, or developing new features).

- **Main Branch**: The `main` branch will always contain production-ready code. Only tested and approved code will be merged into this branch.
- **Development Branch**: All new tasks, whether for coding, documentation, or bug fixes, will be developed in task branches created from the `development` branch.
- **Task Branches**: Each new task (feature, bug fix, or documentation section) will have its own dedicated branch named descriptively (e.g., `webpage-login-feature`, `webpage-signout-fix`). This allows team members to work independently without affecting the main codebase.
- **Pull Requests (PRs)**: Once a task is complete, the developer will submit a pull request (PR) to merge the code into the `development` branch. PRs must be reviewed by at least **two rotating team members** before merging to ensure code quality and consistency.

7.2 Issue Management

GitHub Issues will be used to track tasks, bugs, and feature requests throughout the project. Each issue will:

- Be clearly described, outlining the task, bug, or enhancement in detail.
- Include relevant labels to indicate priority (e.g., high, medium, low) and type (bug, task, improvement).
- Have a designated assignee to ensure accountability.
- Include milestones and deadlines where applicable.

Issue Templates will be created to standardize how issues are reported:

- **Bug Report Template**: A form to report bugs, with fields for reproduction steps, expected behavior, and actual behavior.
- **Task/Request Template**: A form to describe tasks, such as documentation sections or new features, detailing the goal and steps for completion.

7.3 Continuous Integration (CI)

The team will implement **Continuous Integration (CI)** to ensure that the code remains stable as new changes are introduced.

- A basic **CI pipeline** (using a tool like **GitHub Actions**) will be configured to automatically run tests whenever a new pull request is opened. This will ensure that the new code doesn't break existing functionality or introduce bugs.
- The CI pipeline will run simple unit tests and style checks, ensuring the code is clean and functional.
- Only code that passes all CI checks will be eligible for merging into the `development` branch.

7.4 Code Review Process

Every pull request will require at least **two reviews** from **rotating team members** before being merged.

- **Rotating Reviewers**: Team members will take turns reviewing pull requests to ensure that everyone contributes to maintaining code quality.
- **Review Feedback**: Reviewers will provide constructive feedback through GitHub, and any requested changes must be implemented before approval.

7.5 GitHub Projects & Kanban Board

The team will use **GitHub Projects** with a **Kanban board** to visually track progress. GitHub Projects allows automatic creation of tasks on the Kanban board from **GitHub Issues**, ensuring that both issue tracking and task tracking are seamlessly integrated. The Kanban board will include the following columns:

- **To Do**: Automatically populated with new issues created in GitHub.
- **In Progress**: Tasks/issues that are actively being worked on will be moved to this column.
- **In Review**: Pull requests awaiting review will be placed here.
- **Completed**: Once a task is fully completed and merged, it will be moved to the Completed column.

7.6 Task Assignment

Tasks will be assigned via **GitHub Issues** and reflected on the **Kanban board**. Team members can self-assign tasks or be assigned by the project manager. Larger tasks will be broken down into smaller sub-tasks to ensure steady progress. This process ensures that task assignment, status, and completion are always visible, helping the team stay on track and maintain accountability.

8 Project Decomposition and Scheduling

8.1 Scheduling

GitHub Projects is the software we will be using for project scheduling. It integrates our issues with our pull requests on GitHub to improve our effectiveness.

We will be using GitHub Projects for:

- **Task organizing and breakdown**: To ensure our progress is manageable, we will break down the project into smaller attainable tasks. This will be done with GitHub issues for each task and also creating sections for tasks (Frontend, Backend, etc.).
- **Tracking Milestones/progress**: The project will be initially broken down into larger phases (milestones). This will ensure that high-level goals are met and tracked properly at each phase of the project.
- **Assign team members to tasks**: To ensure accountability for project tasks and deliverables, each team member will be assigned a task that they will be responsible for. This will provide us with a more standard procedure to track and follow up on the progress of the project.
- **Collaboration**: This will provide a central platform to work on project milestones and issues as a team. It streamlines the development process through a variety of things, including team discussions, project feedback, etc.

This is the link to the GitHub Project: Capstone.

8.2 Decomposition

The project will be broken down into these major milestones:

- **Documentation**: This is an ongoing phase throughout the entire duration of the project. This includes keeping track of resources used, research, and other important project information.
- **Information Gathering**: This phase will include meeting with stakeholders/project instructors to understand and collect information on the goals and overall project plan. There will be an analysis of the current system in place, then specifications and requirements will be developed based on the team and stakeholders' examination of potential areas of improvement.
- **System Design**: At this phase, we will design the system's architecture with the necessary technology stack needed for the implementation. This will include technical design for the frontend, backend, database, and other required components needed for the functionality of the system.

- **Development**: This will be one of the longer milestones. This stage will be focused on setting up and creating the necessary environments (frontend, backend, database, etc.). At this stage, the connection between the different environments will be established.
- **Testing**: This phase will include testing the system's usability, functionality, and performance. The project will be tested to ensure it meets the requirements and specifications. At this milestone, the system will also be tested for bugs so necessary refinements can be implemented. This stage will improve the growth and confidence in the product.
- **Deployment/Conclusion**: This will be the final stage of the milestone. At this stage, the product will be launched and introduced formally to stakeholders and other end users. All supporting documents will be updated and finalized, then the team will hand over all the documents, accompanied with a session to showcase the final product and features.

8.3 Timeline

The timeline for this project will be as follows:

Task	Deadline
Problem Statement, POC Plan, Development Plan	September 24th
Requirements Document Revision 0	October 9th
Hazard Analysis	October 23rd
V&V Plan Revision 0	November 1st
Proof of Concept Demonstration	November 11th - 22nd
Design Document Revision 0	January 15th
Revision 0 Demonstration	February 3rd - 14th
V&V Report Revision 0	March 7th
Final Demonstration	March 24th - 30th
EXPO Demonstration	April
Final Documentation	April 2nd

9 Proof of Concept Demonstration Plan

9.1 Project Risks

The Scheduling component is the core component of the platform, managing game schedules across multiple divisions, which requires handling both standard scheduling and flexibility for rescheduling. The challenges include:

- **Complex Logic Requirements**: Generating a balanced, fair schedule for each team across divisions involves creating logic that factors in team availability, preferences, and fair game distribution while avoiding conflicts. Rescheduling due to conflicts, particularly on short notice, adds another layer of complexity.

- **User Interface Challenges**: Creating an intuitive and flexible interface where captains can request game reschedules and select preferred time slots poses additional challenges. The interface needs to accommodate various scenarios, including availability conflicts and division-based schedules, without overwhelming users with complexity.

9.2 Proof of Concept Demonstration

To address these risks, the PoC demonstration will focus on the following key areas to mitigate these concerns:

- Automatic schedule generation for a league with multiple divisions.
- Initial development of rescheduling functionality through a captain's interface.

1. **Create Demo Team Data**
    - Populate sample dataset for demo teams across multiple divisions
    - (Stretch Goal) Include basic metadata for each team, including preferred game days and time slots, to simulate real league scenarios.
2. **Implement Basic Scheduling Algorithm**
    - Develop an initial algorithm to auto-generate balanced schedules based on team availability, preferred time slots, and division constraints.
    - If short on time, for simplicity, start with a basic round-robin schedule within each division to ensure every team plays each other team equally.
3. **Develop the User Interface for Schedule Viewing**
    - Create a web interface that shows each team's schedule within their division.
    - Include a section for team captains to request a game reschedule by selecting their desired time slots.
    - Create a display that shows the current schedule for each team, highlighting upcoming games and any requested reschedules.
4. **(Stretch Goal) Enable Rescheduling Functionality**
    - Add an option in the UI for team captains to request rescheduling of a specific game.
    - Develop logic that handles reschedule requests, taking into account existing game slots to avoid overlaps.

10 Expected Technology

Since the McMaster Softball League Platform will be a full-stack web development project, it is best to use technologies that are popular in the industry and ones that we are already familiar with. We choose to use modern frameworks so that this project can be easily picked up by someone else in the future if updates or modifications are needed. Our project needs to be highly maintainable and able to scale in size for long-term use.

10.1 Frontend

React.js

A JavaScript library for building interactive user interfaces. It is advantageous when we need dynamic parts of the platform, such as user interactions. This library will make it easier to control team management, scorekeeping, and any other functional interaction required to make the platform meet expectations.

Another important concept in React is components. We can create components to separate logic and these components can be reused constantly for different purposes. For example, we can design a simple button component and reuse this component at multiple endpoints for the site, saving time and increasing work efficiency.

Tailwind CSS

A CSS framework that provides flexibility in styling the front-end components by using a utility-first concept. A utility-first concept is when you use pre-defined classes (utilities) directly in your HTML. Hence, we apply numerous pre-defined classes to achieve what we want. This framework saves time by eliminating the need to create new CSS classes for different designs, ensuring that everyone uses the same existing classes to maintain consistency across the website.

10.2 Backend

Node.js with Express.js

Node.js is a scalable JavaScript runtime for building server-side applications. Express.js will be used to simplify the management of API requests, handle authentication, and interact with the database.

Node.js is a runtime environment, which allows us to run JavaScript on our server. This tool is incredibly useful since it is practically mandatory for us to run out of backend logic for our application outside of the browser.

Express.js is a framework of Node.js, offering tools to simplify the web application development experience. This framework allows for an easier way to

handle communication between our browser and server, such as GET requests and POST requests.

MongoDB

A **NoSQL** database is ideal for a sports league platform like ours. **MongoDB** is known for being highly scalable, capable of handling large amounts of unstructured Data. Its flexibility is unmatched.

SQL stores rows in tables. In NoSQL, documents store information and can have different structures. Documents can be stored in the same collection, similar to a table. For example, one document might have a "captain" field, while another might not. However, they can both exist in the same collection.

10.3 Version Control

Version control such as git, GitHub, and GitHub projects is mandatory to avoid changes pushed to production without proper review.

10.4 Authentication

JWT (JSON Web Tokens) is the standard for secure user authentication and we can use this to vary access control depending on the user role. For our platform, permissions will be different for players, captains, and commissioners.

10.5 DevOps & Deployment

We will use GitHub Actions to set up Continuous Integration (CI) and Continuous Deployment (CD). GitHub provides documentation that we can follow to learn more about its features.

Since the current website is already being hosted in some domain, we can reuse the same domain for our production. We do not need other deployments.

10.6 Testing

We will use a combination of automated and manual testing to ensure the platform works reliably. Simple unit tests will be written to verify that core backend functionalities, such as user account creation, correctly handle and store inputs like name, email, and password. In addition to this, we will perform a lot of manual testing by navigating through the website ourselves, interacting with various features, and checking that everything behaves as expected from a user's perspective. This hands-on approach will help us quickly catch usability issues and edge cases that automated tests might miss.

11 Coding Standard

Similar to any engineering project, there are practices and rules that our web development project must adhere to. This list will contain the rules that we want to follow to make our project maintainable and scalable long-term for the present and future.

11.1 Consistent Code

Everyone must write code that is easy to read and understand for others. Examples of this include concise, descriptive variable and function names, inclusion of comments, and consistent formatting of code. Code should be broken down with an identifiable purpose so that it can be easily tested. Code duplication should be avoided as this helps keep the codebase consistent and maintainable, while also reducing the risk of errors.

11.2 SOLID Principles

Our code will follow the SOLID principles, which we learned in our previous software engineering classes. These principles are standard practices, ensuring that any code we write will be easily readable by others for interpretation or future usage.

Single Responsibility Principle

A class should only have one responsibility. Responsibilities should remain separate to avoid confusion and code duplication. This principle also makes it easier to locate specific functionalities.

Open/Close Principle

Open for extension, closed for modification. This principle means that any time we want to add a new functionality, there is no need to change or modify the original code. This principle avoids bugs and allows us to update the website smoothly without any loss of functionality.

Liskov Substitution Principle

A subclass should behave correctly according to its parent class. For example, if we have a `bird` class that has method `fly()`, we should not create a subclass called `penguin` because penguins cannot fly. If we ever used a `penguin` to substitute for a `bird`, our program would break. Any subclasses of the class `bird` should adhere to its rules, such as a `pigeon`.

Interface Segregation Principle

Smaller, more specific interfaces to avoid unnecessary methods. Our clients should only access the interfaces they need.

For example, instead of a single `Stats` interface, that includes methods for both player and team statistics, we should separate it into `PlayerStats` and `TeamStats` interfaces. This reduces unnecessary dependencies.

Dependency Inversion Principle

High-level modules should not depend on low-level modules; both should rely on abstractions.

For example, our platform (high-level) should not depend directly on a specific database (low-level), such as MongoDB. Instead, there should be an abstract data layer, allowing us to swap out the database for another if needed, without impacting the rest of the system. In this way, the platform can work with a database, but the specific implementation remains flexible.

11.3 Responsive and Accessible Design

Users have a variety of needs, as all users are different individuals. We want our features to be mobile-friendly and accessible. The standard here is to use ARIA roles, keyboard navigation support, semantic HTML, etc. We also do not want the colours to be negatively affecting user experience.

11.4 API Design & Error Handling

We want our design to follow REST API conventions, including descriptive endpoints that are well documented. We also need proper error handling throughout our functionalities to ensure that both the user and our system understand what error is being communicated, which will enhance the overall experience of our platform.

For example, if a user enters the wrong password when logging in, that message should be conveyed back to the user. This password error must also be logged into our system so that our developers can see the error.

11.5 Documentation

Everyone needs to document their work, so others can understand the process that leads to the result. It is insufficient to only look at the final result because without understanding we are unable to make modifications when needed. In the engineering world, changes are bound to happen and we must document our process to reflect our understanding of that. This concept will help future developers understand and improve upon what we have built here.

Appendix — Reflection

1. Why is it important to create a development plan prior to starting the project?

   - We believe it is important to create a development plan prior to the start of a project because it provide us a better idea of the scope of the project. When viewing a project from an outside persepctive, it is very easy to underestimate the magnitude and the responsibilities required by that project. This often results in a bunch of missed deadlines, increased stress throughout project duration, incomplete or unreliable final product, etc.
   - By creating a development plan, the team is able to ensure we have smaller more manageable tasks, and a clear timeline for the development of the project.
   - Developing a project plan is also very important to improve the collaboration and unity of the team and increase the communication at each stage of the project. This allows each team member to have more accountability and responsibility over their respective parts of the project.
   - A development plan could also help us identify potential issues that we could run into throughout the duration of the project.

2. In your opinion, what are the advantages and disadvantages of using CI/CD?

   Advantages

   - A major benefit of CI/CD is streamlining our development process.
   - By using CI/CD we are able to release code in a faster and more stable way which provides us with a more reliable development.
   - Integrating CI/CD into our project gives us a huge boost in our team collaboration. The team is able to merge smaller pieces of code more frequently without a consistent need for manual intervention. This allows us to test features in smaller units which in return, results in a higher level of confidence in our codebase and code releases.

   Disadvantages

   - Team members could sometimes be unfamiliar with CI/CD, and the changes needed to adopt that skill could result in some lost productivity due to major changes in their mindset, workflow, development process, etc.

- There could be some distrust in the automation with the CI/CD due to unreliable or false test. This could result in reduced confidence in the CI/CD system and wasted time from debugging tests.
- There could also be security risks with using CI/CD. Deploying code with project secrets or vulnerable information on accident could expose sensitive information. This adds in an extra layer of consideration and consistent attention/vigilance.

3. What disagreements did your group have in this deliverable, if any, and how did you resolve them?

   We have had a number of disagreements so far throughout the course of this problem. Initially, we had a problem selecting what project to work on for the capstone. To resolve this, we had each member vote for their 2 top picks and after that we all discussed and we were able to finalize on a project.

   Another issue we had was with the the convention for naming branches. Each individual in the team has had work experience at different companies, and we learn different conventions when it comes to branch names. To decide on this, we took down all the different perspective and weighed each option based which would be most efficient and useful for the team using a matrix, then we were able to decide on a convention for naming branches.

Appendix — Team Charter

External Goals

Our team's external goals include:

- Achieve a grade of 11 or greater in the capstone course.
- Produce a project that team members can discuss in future job interviews.
- Produce a project that team members can take pride in as a representation of 4 years of Software Engineering courses.
- Gain experience in project management and working as a cohesive team.
- Build strong, professional connections within the team, with faculty/supervisors.

Attendance

Expectations

- Team members must attend all scheduled meetings, both virtual and in-person, unless a valid reason is provided in advance.
- Team members are expected to arrive on time and stay for the entire duration of the meeting. Repeated lateness or early departures without prior notice or valid reasons are unacceptable.
- Team member that are missing a meeting must notify the team at least 24 hours in advance, or as early as possible, except in the case of emergencies.
- Team members should come prepared, having completed any assigned tasks and actively participating in the productivity of the meeting.

Acceptable Excuse

Acceptable excuses for missing a meeting include:

- Health-related issues.
- Family emergencies or urgent personal matters.
- Conflicts with other courses or prior academic or professional commitments.
- Unavoidable technical difficulties.

Unacceptable excuses:

- Forgetting the meeting time or oversleeping.
- Non-urgent personal commitments (e.g., social events, non-critical errands).
- Lack of preparation for the meeting.

- Laziness.

In Case of Emergency

Our teams process for emergencies:

- The team member must inform the team as soon as possible through the groupchat explaining their situation.
- If the team member cannot attend the meeting, they should provide a brief update on their progress or any relevant information so the team can continue without them (as soon as safely and conveniently possible).
- In the case that a deadline cannot be met, the team member must alert the team and the team will confer on the urgency of the task and reassign tasks accordingly.
- For major deliverables, the team may confer to try to accommodate a reasonable extension. Team members are expected to be willing to collaborate and contribute extra work to help cover for the lost time from the emergency and put the team first.

In case of prolonged emergencies, the team will review the situation and may reassign responsibilities to ensure the project remains on track.

Accountability and Teamwork

Quality

Our teams process and expectations for quality:

- Each team member is expected to come fully prepared for meetings, having completed any assigned tasks and ideation processes.
- Deliverables and tasks should be done to the best of the team member's abilities.
- At least one other team member will have reviewed each task. If a task is below the teams accepted quality, the reviewer is expected to respectfully bring it to the attention of the team and suggest improvements.
- Team members are expected to provide constructive criticism to other team members, and team members are expected to handle the feedback professionally.

Attitude

Our teams process and expectations for attitude:

- Each team member is expected to act professionally and cooperatively.

- Team members should be expected to be willing to help out on other member's tasks, providing feedback and support (within reason).
- Team members should follow the code of conduct and conflict resolution plans from Harvard University: https://hr.harvard.edu/staff-personnel-manual/employee-conduct

Stay on Track

To keep the team on track, we will:

- Have weekly progress check-ins to ensure tasks are being completed as expected.
- Set specific target metrics, such as attendance, commits, and task completion.

If any team member misses a meeting without reason, they have to bring coffee at the next meeting.

Team Building

To build team cohesion, we will organize at least 2 team-building activities over the course of the Capstone course and begin meetings with a personal check-in.

Decision Making

Our team will aim for decision-making by consensus, where every member's opinion is considered. In situations where consensus cannot be reached, we will vote, with the majority decision winning. For significant disputes, we will seek input from the project TA or instructor to guide the final resolution.