Development Plan
Software Engineering

Team #12, Streamliners
Mahad Ahmed
Abyan Jaigirdar
Prerna Prabhu
Farhan Rahman
Ali Zia

Table 1: Revision History

Date	**Developer(s)**	**Change**
Date1	Name(s)	Description of changes
Date2	Name(s)	Description of changes
...	...	...

[Put your introductory blurb here. Often the blurb is a brief roadmap of what is contained in the report. —SS]

[Additional information on the development plan can be found in the lecture slides. —SS]

1 Confidential Information?

This project does not involve the use or disclosure of confidential information from any industry partner. All requirements, specifications, and implementation details have been developed by the student team. As there is no industry-provided proprietary data or trade secrets involved, no Non-Disclosure Agreement (NDA) is required.

2 IP to Protect

All intellectual property (IP) generated as part of this project is owned by the student team, in accordance with McMaster University's policies on ownership of student work. Since the project is not sponsored by an external industry partner, no Intellectual Property Guide Acknowledgement form is required. To enable the intended use of the system, the student team will grant the McMaster Engineering Society a non-exclusive license to operate and maintain the software for its events and activities. Ownership of the underlying source code remains with the student developers.

3 Copyright License

All software developed by the student team is protected under copyright law. To clarify permitted use, the team has adopted the MIT License, which is included in the project repository here. This license allows reuse and modification of the code, provided that the original copyright notice and license terms are preserved. This ensures proper attribution to the development team.

4 Team Meeting Plan

The team will meet twice a week. Once online on Monday, from 9:30pm to 10:30pm, to discuss details on work to complete during the week. The second time will be from 5:00pm to 6:30pm to discuss progress, roadblocks, or any issues where assistance is required. This meeting will be online on the weeks there are no in-person progress check-ins and in-person otherwise.

The team will meet with the industry advisor online every week on Wednesday during the 4G06 tutorial time from 4:30pm to 5:00pm. This meeting will be to discuss project updates, clear up confusion, and ask for help if needed.

Before all meetings, an agenda will be provided. The agenda will cover the main discussion points to guide the conversation in meetings. All meetings will be structured with project updates provided by all team members first, then an update on any roadblocks or issues that need to be dealt with. The end of the meeting will focus on developing a plan for next steps, assigning tasks to each member, and defining an agenda for the next meeting. At the end of the meeting, a record of the meeting minutes will be provided by the chair of the meeting to keep the team accountable for their tasks.

5 Team Communication Plan

Main Objectives: The main objective of the team communication plan is to ensure the team is updated on the tasks they have to complete, the current project progress and the deadlines for completing their tasks. This is to ensure all members are accountable for the progress of the project.

Communication Platform: The team's main form of communication will be through Discord for quick updates, project reminders, discussions on division of labour for milestones and getting help for issues faced.

Response Times: Maximum 12 hours response time for any form of communication from accepting github issue assignments to message responses on Discord.

GitHub Issue Tracking:

- Creation of Issues: Issues will be created for every project task, defect or feature and will be linked to the Kanban board.
- Issue Assignment: Issues will be assigned to team members based on interest and group consensus.
- Issue Updates: Issue progress will need to be updated in the Kanban board by dragging into "In Progress" state or "Review" state.
- Issue Reviews and Closure: Issues will be reviewed by the team to ensure quality standards are met and will then be merged and closed after consensus is reached.

Conflict Resolution: Disagreements on responsibilities, division of tasks or ideas escalated to the entirety of the team for a vote to decide on next steps. Final escalation to project supervisor if no consensus reached.

6 Team Member Roles

1. **Leader: Farhan**
   Responsibilities: Leading team discussions with open-ended questions and guiding team direction to achieve goals.

2. **Organizer/Reviewer: Ali**
   Responsibilities: Keeping the team on track for project milestones and deadlines and making sure the quality of work meets team standards.

3. **Editor: Abyan** Responsibilities: Preparing final versions of deliverables and proofreading documents to ensure consistency and accuracy.

4. **Notetaker/Meeting chair: Prerna** Responsibilities: Gathering important information from lectures and tutorials and recording meeting notes to record and summarize project progress.

5. **Trouble-shooter: Mahad** Responsibilities: Help resolve issues and challenges faced by the team and analyze problems and criteria to figure out solutions.

7 Workflow Plan

7.1 Git Usage

- Github will be used as the primary version control system (VCS).
- All commits must be made to a non-master branch and submitted via a pull request before they can be merged.
- Branching Strategy:
  - Main branch: Stable production-ready code waiting for a release.
  - Documentation branches: Used to maintain and update project documentation and requirements.
    - Notation: `docs/SL-<Issue#>`
  - Development branch: Integration branch where all new features and bug fixes are merged. This branch will contain the latest features with the most up-to-date code.
    - Notation: `dev`
  - Feature branches: Used to develop and test new features before merging into the development branch.
    - Notation: `feature/SL-<Issue#>`
  - Bugfix branches: Used to fix defects.
    - Notation: `bugfix/SL-<Issue#>`

7.2 Issues and Project Management

- Github issues will serve as the primary issue/ticket system
- All subsections of each document/report will have its own issue
- All new tasks and/or features will have its own issue
- Issues will be classified to using labels (e.g., `documentation`, `feature`, `bug`).
- Github's Kanban board will be used to monitor progress

7.3 CI/CD Usage

- A CI/CD pipeline will be implemented using Github Actions.
- Continuous Integration **(CI)**:
  - Automatically build the project and run tests on every pull request.
- Continuous Deployment **(CD)**:
  - Deploy staging builds after merging to development branch.
  - Deploy production builds after merging to main branch.

8 Project Decomposition and Scheduling

- How will you be using GitHub projects?
- Include a link to your GitHub project

[How will the project be scheduled? This is the big picture schedule, not details. You will need to reproduce information that is in the course outline for deadlines. —SS]

9 Proof of Concept Demonstration Plan

What is the main risk, or risks, for the success of your project? What will you demonstrate during your proof of concept demonstration to convince yourself that you will be able to overcome this risk?

10 Expected Technology

Languages & Frameworks

The team will use **JavaScript** as the primary language across the stack. The web application will be built with **Next.js (React)**, the mobile application with **React Native**, and the backend will be developed using **NestJS** (chosen for its opinionated structure atop Express).

Database & ORM

The database will be **PostgreSQL**, self-managed via a Docker container. For database access, the team is considering **Drizzle ORM** as the shared standard across groups.

Payments

Payments will be handled through **Stripe**, integrated using official SDKs. Processor-specific logic will be isolated within the backend service.

Selected Libraries (initial)

- **Frontend:** a React component library (e.g., Tailwind or Material UI) to accelerate UI development.
- **Backend:** NestJS modules for routing/validation; Stripe SDK for payments; Drizzle ORM for database access.

Custom Components Expected to Be Implemented

Despite the existence of some libraries, the team anticipates implementing core logic to align with project requirements:

- **Feature-Based Access Control (FBAC)** with role templates and per-user overrides.
- **Sign-up capacity/hold logic** (e.g., bus/table/RSVP) to safely manage contention and inventory.

Dev Workflow & Tooling

- **Git** for version control; **GitHub** for repository management; **GitHub Projects** for task tracking.
- Production hosting will be on **DigitalOcean** (school-hosted), with Postgres running in Docker.

Testing, CI/CD, and Performance

The team intends to adopt a layered testing strategy:

- **Unit tests** for backend modules (NestJS services/controllers) and React components.
- **Integration tests** to validate backend endpoints against a local Postgres instance.
- **End-to-end (E2E) tests** for critical flows such as checkout, sign-ups, and access control.

Continuous Integration will be run through **GitHub Actions**, with pipelines for linting, type-checking, running automated tests, and validating database migrations.

11 Coding Standard

The team will adopt consistent coding standards to ensure readability, maintainability, and effective collaboration. For JavaScript code, the team will follow the Airbnb JavaScript Style Guide, which provides well-established conventions for formatting and best practices. Code formatting will be automated with **Prettier**, and linting will be enforced with **ESLint** to reduce inconsistencies across contributions.

Naming conventions will remain uniform across the stack:

- `camelCase` for variables and functions
- `PascalCase` for React components and NestJS classes
- `snake_case` for PostgreSQL tables and column names

Beyond style rules, the team will apply **Clean Code principles** to keep the codebase simple and easy to understand. This means writing small and focused functions, choosing descriptive names, avoiding duplication, and keeping modules cohesive. Comments will be added only where necessary to clarify complex logic, with the preference being self-explanatory code.

For version control, all work will be done in feature branches, with pull requests reviewed before merging into `staging` or `main`. Commit messages will follow the imperative style (for example, "Add payment processing logic") to maintain a clear project history.

Appendix — Reflection

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

   Creating a development plan helped the team align on goals, expectations, and responsibilities before beginning development. Because this project spans multiple groups, documenting decisions early established a common foundation and reduced the risk of misunderstandings later. It also encouraged us to think ahead about technologies, workflows, and risks, which gave the team more confidence moving into implementation.

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

   The advantages of CI/CD include automated testing, faster feedback, and reduced integration issues. It helps ensure that code changes are validated continuously, which improves reliability and makes collaboration smoother. On the other hand, the disadvantages are the upfront setup effort and the need for ongoing maintenance of pipelines. For a student project with limited time, the main challenge is balancing the investment in CI/CD with the need to deliver features quickly.

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

   The team did not have any major disagreements, but there were discussions around which ORM should be used. Some members had experience with Prisma, while others were interested in Drizzle for its modern features and close mapping to SQL. After reviewing the trade-offs together and following up with the mentors for additional guidance, the team decided to move forward with Drizzle as the preferred option. This process allowed the team to weigh different perspectives while still keeping flexibility in case future integration across groups requires adjustments.

Overall, this deliverable reinforced the importance of careful planning, open communication, and collaborative decision-making. By documenting our expectations early, reflecting on the role of CI/CD, and carefully weighing options for the technology stack, the team not only produced a development plan but also practiced the skills needed to sustain long-term teamwork. These lessons will continue to shape how we approach challenges in the later stages of the project.

Appendix — Team Charter

External Goals

- Secure a grade of A+ in 4G06.
- Gain real-world experience in areas of project management, teamwork and shipping projects.
- Create a strong portfolio project that has a large user base and can be discussed in interviews.

Attendance

Expectations

- All members are expected to attend all scheduled meetings either in-person or online.
- All members are expected to be on time for meetings and remain present for the duration of the meeting.
- All members are expected to notify the team 24 hours beforehand if they are unable to attend meetings.

Acceptable Excuse

Acceptable excuses for missing meetings or deadlines include the following:

- Family emergency
- Health issues
- Medical appointments

Unacceptable excuses for missing meetings or deadlines include the following:

- Forgetting about meetings
- Oversleeping

In Case of Emergency

- If the emergency affects the members ability to attend a meeting, they must notify the group immediately or as soon as possible after the emergency.
- If the emergency affects completion of work, the member must communicate explicitly what work is done and what remains so the team can decide on the division of labour or a plan to catch-up.

Accountability and Teamwork

Quality

- All members must come to meetings prepared with progress updates on their tasks.
- Deliverables should be well-organized and reviewed before they are shared with the team.
- Contributions in the code repository must follow team standards for readability and documentation.

Attitude

- All members must listen actively to all other members and avoid passive or dismissive behavior.
- All members should respect all ideas and viewpoints without judgment.
- All members are encouraged to share their ideas openly.
- All members must adopt a simple code of conduct revolving on principles of respect, communication and collaboration.
- Conflicts must be resolved respectfully and if unresolved will be escalated to the TA.

Stay on Track

- Weekly progress check-ins where members discuss what they completed and next steps will be used to keep the team on track.
- GitHub Issues and commits will be used to ensure members contribute as expected.
- Metrics:
  - An attendance rate of 100% is expected from all members unless an excuse is deemed valid and acceptable by the team.
  - Consistent GitHub activity is expected per week with issues in progress or items being committed for review.
  - On-time delivery of tasks on assigned deadlines.
- Rewards:
  - Members who reach targets early or who do well overall will be rewarded with acknowledgments of being strong contributors.
- Consequences:

  - First time incident will be a verbal reminder.
  - Second time will result in a discussion with the TA.
  - Third time and more will result in escalation to the course instructor.

Team Building

- Start meetings with small talk and conversations to get to know each other and maintain team connection.
- Doing a group ritual of small activities like board games or lunch to celebrate milestone deliveries.

Decision Making

- Initial plan will be to reach consensus when possible.
- If consensus cannot be reached, opt for a majority vote.
- If disagreements still exist within the team if consensus and majority vote leaves members unsatisfied, involve the TA for handling disagreements.