Development Plan
Grocery Spending Tracker

Team 1, JARS
Jason Nam
Allan Fang
Ryan Yeh
Sawyer Tang

Table 1: Revision History

Date	**Developer(s)**	**Change**
19/09/23	Ryan Yeh	Wrote initial Team Communication Plan
20/09/23	Allan Fang	Added team member roles
20/09/23	Ryan Yeh	Revised Team Communication Plan based on examples
20/09/23	Sawyer Tang	Workflow Plan
20/09/23	Jason Nam	Added Team Meeting Plan Section for Development Plan Doc
20/09/23	Allan Fang	Added document introduction
21/09/23	Ryan Yeh	Wrote initial Project Scheduling
24/09/23	Ryan Yeh	Wrote Proof of Concept Demo Plan
24/09/23	Sawyer Tang	Code Style Standards
24/09/23	Jason Nam	Added Technology section of Development Plan Doc
18/10/23	Allan Fang	Formatting and style, role adjustments from feedback
24/10/23	Ryan Yeh	Reduced scope of POC Demo Plan
08/11/23	Ryan Yeh	Clarified wording for POC Demo Plan

This document describes the necessary aspects for the effective execution of the 'Grocery Spending Tracker' project including team organization, communication, technical details, and project timelines.

1 Team Meeting Plan

- Standup meetings are scheduled weekly from **12:30 PM to 1:30 PM every Wednesday**. The primary meeting location is the *H.G. Thode Library of Science & Engineering*. However, in specific circumstances and within certain limitations, the meeting location may be shifted to *Discord*.
- Sprint review meetings are scheduled weekly from **5:00 PM to 6:00 PM every Sunday**. The meeting location is *Discord*.
- The roles and agendas for the next meeting will be decided in the final minutes of each meeting.
- Requests for additional meetings outside of regularly scheduled meetings must be made and communicated at least **1 day in advance**.
- The Chair of the meeting is responsible for preparing the task backlog and identifying any relevant issues before the meeting.

2 Team Communication Plan

2.1 Discord

The primary means for communication will be the team *Discord* server. To ensure team members are up-to-date with messages and discussion, everyone is expected to check *Discord* at least once per day and respond to direct messages within 24 hours. Additionally, the server has been organized in such a way that different topics are separated into different channels. For general discussion and planning, the *general* channel will be used. When requesting reviewers on pull requests, the *pull-request* channel should be used. Discussion surrounding bugs and issues will be brought up in the *bugs-issues* channel. Discussion of progress updates for upcoming deliverables will go into the *status-updates* channel and lastly, any requests for help regarding work items will go into the *help* channel.

2.2 SMS

Should the team require urgent or time-sensitive contact with any members, *SMS* will be used as the method of communication. *SMS* should facilitate faster and more efficient communication compared to *Discord* and will be used in events such as unforeseen issues with meeting availability or last-minute changes for deadlines.

2.3 GitHub

GitHub will be used to facilitate more technical code-related discussions via pull requests in addition to keeping track of work items using the issue tracker. The issue tracker will also be used to store notes from all group meetings.

3 Team Member Roles

Individuals will take leadership in their respective assigned areas. In development, all members are responsible for adjusting and filling roles as the project evolves. All members will participate in all tasks including but not limited to development, testing, and documentation. Sprint tasks will be assigned and reviewed bi-weekly at meetings. All team meetings will have a scrum master and a scribe, rotated on a per-meeting basis.

3.1 Allan Fang - Developer

- Lead on mobile and user interface development
- Lead on optical character recognition research and development
- Graphics design and creation specialist

3.2 Jason Nam - Developer

- Lead on research and development concerning language models
- Lead on backend development
- Lead data engineer

3.3 Sawyer Tang - Developer

- Responsible for server setup
- Responsible for DevOps and version control
- Graphics design and creation specialist
- Lead on backend development

3.4 Ryan Yeh - Developer

- Lead on project documentation
- Lead on testing and quality assurance
- Primary contact for external communications

4 Workflow Plan

We will be using Git as a version control software with GitHub as our remote repository. The root branch will be `master` and the development branch based on `master` will be `dev`. The developer will branch off from `dev` to work on issues. There will be a new branch for each issue and will be named containing the issue number and name detailed below. Continuous Integration is set up between all branches `dev`, `master`, and Issues. Once the milestone is finished to the degree we are happy to be graded, the `dev` branch will be merged into `master`.

Issues are used to track tasks and meetings related to the project. Meeting issues will be created before each meeting according to the **MeetingIssueTemplate.md** template. These include attendance, minutes, and action items from the meeting. These notes can be referenced from other issues and PRs for traceability. Feature issues are used to track tasks. They can be referenced in PRs and meetings to outline the tasks that the developer must fulfill in their PR. Issues can be assigned to a Milestone which tracks issues related to deliverables. These are useful for filtering issues and tracking due dates. Issues can be assigned labels that give information on which part of the project the issue is related to. These can denote issue types as well as issue severity or complexity and allow filtering.

4.1 Some significant labels:

- **documentation**: used for all issues related to building or maintaining the documentation for our project (including, but not limited to, LaTeX related issues).
- **dev-ops**: used for all issues related to code production, integration, or conventions.
- **meeting**: used to denote an issue is for a meeting.
- **lecture**: used to denote an issue is for a lecture.
- **bug**: used for issues that describe a bug that needs to be fixed.
- **feature**: used for issues that describe a new code feature to be implemented.

4.2 Version Control Developer Step-By-Step

1. Granted issue ticket with the issue number.
2. Open branch with name {IssueNumber}/{issue-name}. (eg. `135/add-user-endpoint`)
3. Commit with format: {IssueNumber}:{commit message}. (eg. `"135: updated yaml for add-user-endpoint"`)

4. Open Pull Request on GitHub following the **PRTemplate.md** template, thus mapping the PR to the Issue.
5. Await 2 reviewers.
6. Fix reviewer comments.
7. Developer will merge their *own* branch to dev.

5 Proof of Concept Demonstration Plan

The main functionality necessary for this application to succeed is the OCR of receipts. In addition to simply recognizing the characters, a big part of the input process is also correctly identifying the relevant data such as which prices belong to which entries and any additional processing that may need to be done in order to clean up the output. The OCR and input function of the application is the core feature of the application and is essential for the final product. Therefore, it is the biggest risk in this project to consider. To show that this risk can be overcome, the Proof of Concept Demo will involve manually recreating the backend pipelining for the OCR portion of the application. A receipt will be scanned and passed through the different components before the output is displayed to show the feasibility of the OCR and input system. Ideally, the result of the demo will be an accurate representation of data that was captured from the receipt.

6 Technology

6.1 Programming Language

- The team will utilize **VSCode** source-code editor.
  - VSCode is very flexible and well-documented.
  - The team has plenty of experience working with VSCode.
- The team will utilize **NodeJS** as their back-end programming language.
  - NodeJS is very popular and well-documented.
  - NodeJS has a reliable and flexible coding environment.
- The team will utilize **Flutter** (Dart) as their front-end programming language.
  - Flutter is a mainstream and modern open-source UI software development kit created by Google.
  - Flutter ports well with web, Android, and IOS operating systems.

6.2 Linter Tool

- The team will utilize **flutter_lints** for front-end programming language which is built on top of Dart's recommended sets of lints from Official Dart lint rules.
- The team will utilize **ESLint** tool for identifying and reporting on patterns found in ECMAScript/JavaScript code.

6.3 Unit Testing Framework

- The team will use Flutter's integrated unit testing framework for front-end unit testing purposes.
- The team will use **Postman** and **Mocha** frameworks for back-end unit testing purposes.
  - The team has extensive experience working with Postman.
  - Postman is a popular choice for testing back-end database queries.

6.4 Code Coverage Measuring Tools

- The team will utilize **coverage-node** to run and report code coverage for back-end NodeJS code.
  - coverage-node allows code coverage report with zero configurations.
  - Package is lean and simple to use.
- The team will utilize Flutter's coverage extension in VSCode to measure front-end code coverage.

6.5 Plans for Continuous Integration (CI)

The team will utilize **Github** for Continuous Integration with automated testing and linting being integrated into the pipeline.

6.6 Performance Measuring Tools

- The team will utilize **Flutter Performance Profiling**, part of Flutter DevTools, for debugging front-end performance issues.

6.7 Libraries and Other Tools

- The team will utilize **node-tesseract-ocr** to handle OCR on back-end.
  - node-tesseract-ocr compiles well with node.
  - node-tesseract-ocr is popular and well documented.

- The team will utilize **PostgreSQL** relational database to persist and aggregate data on demand.
  - PostgreSQL is a flexible database with high levels of control.
- The team will utilize **node-rsa** to encrypt data and provide security for the system.
  - node-rsa is widely used for encrypting back-end databases using node.js.

7 Coding Standards

- NodeJS code will follow the ECMAScript scripting conventions. eslint package will be used to keep code in adherence with ECMAScript.
- Flutter code will follow the Dart Style Guide. flutter_lints package will be used to keep code in adherence with the Dart Style Guide.

8 Project Scheduling

The general scheduling of the project will follow a weekly sprint format. At the beginning of each week, a plan will be made based on the deliverables due and progress made from previous sprints. The work will then be distributed afterward such that all members have tasks they have to work on. This structure will ensure feasibility regarding weekly objectives and maintain focus on certain goals rather than scope creep over a longer development period. This will also allow more flexibility among the team to adjust workload based on other responsibilities.

Regarding course deliverables specifically, scheduling will follow closely to the deadlines written in the course outline. The goal for each deliverable will be to have them finished a couple of days before the actual deadline to account for any problems or changes that may need to be made. As requirements are created and planning gets done for different project features, more technical tasks will be added via the *GitHub* Issue Tracker and these too will be integrated into the weekly sprints as they become available. This will especially be true during extended deadlines such as the period between the Proof of Concept Demonstration and the Design Document Revision 0 deadline.

Overall, this scheduling structure should promote a steady output of work allowing the team to meet all deadlines and produce a better end product.