Project Plan Documentation
For

10/01/2024

Team Members: Danny Elhamalawy, Kevin Gawrinauth, Alex Pateroulakis, Elijah Philip, Gayathri Suresh

Version 1.0

TABLE OF CONTENTS

1. Introduction

This document presents a comprehensive project plan for the development of the GreenGrocer web-based application. It provides a clear outline of the tasks, timelines, and resources needed to complete the project within a 9-week timeframe. The project work begins on October 1st, 2024, and will finish before December 3rd, 2024. The schedule is set to finish by Week 8, with the last week primarily for slack/cushion in case of scheduling changes. It is important to note that all metrics, schedules, and team designations are subject to change throughout the project.

2. Project and Dependencies

2.1 Requirements Table:

The requirements table outlines the basic tasks to be completed alongside their alphabetical designation, duration, dependencies, risk level, completion status, and team members responsible.

Task	Task Name	Days	Dependencies	Delegation	Risk Level	Completed?
A	GitHub	1	None	Danny	Low	Yes
B	MySQL Lite	4	A	Kevin, Elijah	High	Yes
C	Website Creation	6	A	All	Medium	No
D	User	3	B	Alex	Medium	No
E	Kroger Product	5	C	Kevin	Medium	No
F	Login/Sign-Up	4	D	Gayathri	Medium	No
G	Location	4	E	All	High	No
H	Settings	3	F	Danny	Medium	No
I	Product Page	5	E, G	Alex	High	No

J	Search	3	I	Gayathri	High	No
K	Categories	4	I	Elijah	Low	No
L	FAQ	2	H	All	Low	No
M	User Details	3	H	Danny	Medium	No
N	Order History	4	H	Kevin	High	No
O	Favorites	3	I	Alex	High	No
P	Loyalty	5	L, M, N	Gayathri	Medium	No
Q	Cart	5	J, K, O	All	High	No
R	Checkout	6	P, Q	Kevin	High	No
S	Summary	3	R	Elijah	Low	No
T	Accessibility	4	S	Elijah	Low	No
U	Final Documentation & Testing	8	T	Danny, Gayathri	Low	No

Table 1: Requirements Table

2.2 Requirements Descriptions and References:

A. GitHub

- Create a GitHub repository with branches for version control. Ensure all team members can access and commit the initial project files.
- Requirements Section 2.4

B. MySQLite

- Design and implement the database schema using MySQL Lite.
- Requirements Section 2.2.3

C. Website Creation

- Develop the website and its overall layout, structure, and functionality using CSS, HTML, and JavaScript. Utilize the Figma design as a reference to build the

website.

- Requirements Section 2.2.2

D. User

- Develop the user interface and functionality, with registration and login via Flask, and use storage via MySQL Lite database.
- Requirements Section 2.2.3

E. Kroger Product

- Configure the Kroger API to the website. Use the Kroger API to fetch product details such as name, price, and category for displaying.
- Requirements Section 2.2.3

F. Login/Sign-Up

- Implement the login and sign-up functionality using Flask and forms, connecting to the database for user login.
- Requirements Section 3.1, *R-1* to *R-8*

G. Location

- Integrate 5 locations using the Kroger Location API to display store locations and allow users to choose their preferred store.
- Requirements Section 3.2, 3.4, *R-21* to *R-22*

H. Settings

- Create a sidebar menu that allows users to view their account and settings information.
- Requirements Section 3.2, 3.3, *R-13* to *R-20*

I. Product Page

- Develop a product page using HTML that fetches items from the Kroger API.

- Requirements Section 3.6, *R-31* to *R-33*

J. Search

- Implement a search bar functionality that enables users to find products quickly using name, brand, and product type.
- Requirements Section 3.6, *R-27* to *R-30*

K. Categories

- Create categories on the product page to allow users to navigate through products. Implement using Flask and Bootstrap to integrate APIs for real-time data.
- Requirements Section 3.5, *R-23* to *R-26*

L. FAQ

- Develop an FAQ page that addresses common questions about the website, services, and products.
- Requirements Section 2.6, 3.3

M. User Details

- Manage user information such as name, email, phone, and address. Registered users can view and edit these details. This is a crucial feature for account management, storing data in a MySQL Lite or JSON format.
- Requirements Section 2.2, 3.3

N. User History

- Create a User History page that allows registered users to view their past orders, including dates, methods of payment, and any applied discounts. This feature is useful for tracking purchases and is built using SQLite/JSON and Flask.
- Requirements Section 3.2, 3.3, *R-18*

O. Favorites

- Create a Favorites functionality for registered users to mark items as favorites for easy access in the future. This feature saves favorite items, allowing users to quickly find products they often buy.
- Requirements Section 3.2, *R-19*

P. Loyalty

- Create a Loyalty program with rewards for registered users using points for purchases. The feature tracks how many points are accumulated, which can later be used for promotions or discounts.
- Requirements Section 3.3, 3.9, *R-49* to *R-51*

Q. Cart

- Create a cart that allows users to manage items they intend to buy. Users can add or remove products and track the total cost of their cart. It's connected to the checkout process and built using Flask and MySQL Lite.
- Requirements Section 3.7, *R-34* to *R-35*

R. Checkout

- Develop a checkout page that manages the final purchase process. It allows users to review their cart, apply promotions or loyalty points, and select delivery or pick-up options. Users can enter payment details and finalize the order.
- Requirements Section 3.7, *R-36* to *R-44*

S. Summary

- Integrate an order summary functionality for registered users receiving an order summary after checkout, detailing the transaction, order number, and delivery information.
- Requirements Section 3.8, *R-45* to *R-48*

T. Accessibility

- Create an accessibility page that ensures the application can be used by individuals with disabilities, including options to change languages and visual settings (color blindness adjustments, for example).
- Requirements Section 3.10, *R-52* to *R-55*

U. Final Documentation and Testing

- Includes technical documentation for the app's development and usage, maintained through GitHub and Google Docs, providing instructions for developers and users.
- Requirements Section 2.2, 2.6

2.3 Activity Dependency Chart, Gantt Chart, Critical Path, and Slack:

The following Activity Dependency Chart and Gantt Chart below visually represent the project timelines and their dependencies.

Activity Dependency Chart: The activity dependency chart below visually represents the relationship and dependencies of each task, along with their early start, early finish, late start, and late finish (Figure 1). From these times, the critical path was determined, which highlights the sequence of tasks that must be completed on time to avoid delaying the project's overall schedule. The slack was also calculated from these times, which shows the flexibility of the tasks that are not on the critical path, allowing resources to be efficiently allocated and ensuring that potential delays in non-critical tasks do not affect the project deadline.

Gantt Chart: The Gantt chart below visually represents the project schedule, showing the start and end dates of each task and their dependencies (Figure 2). From the chart, the highest resource days are between weeks 3 and 4.

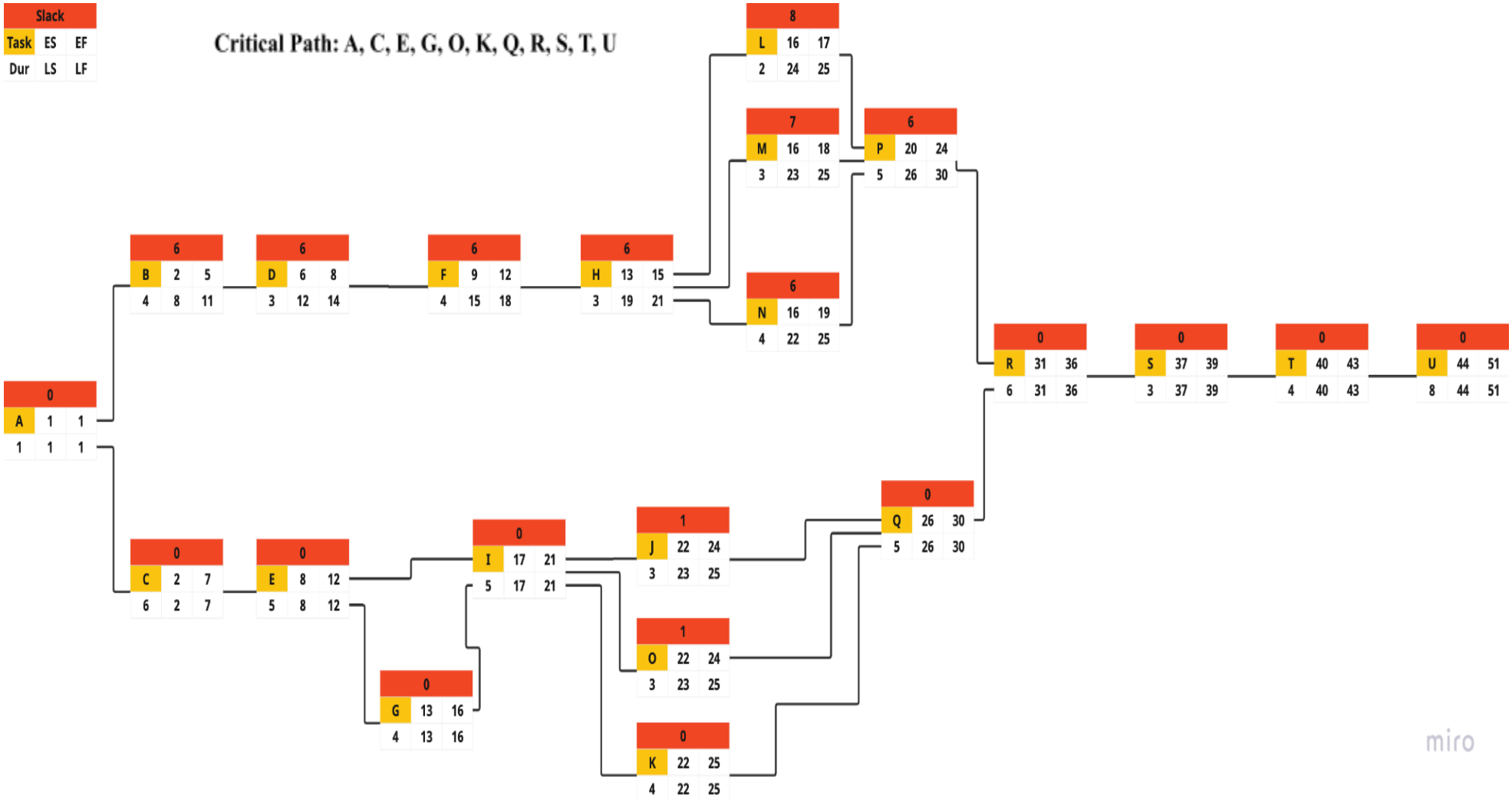

Figure 1. Activity Dependency Chart

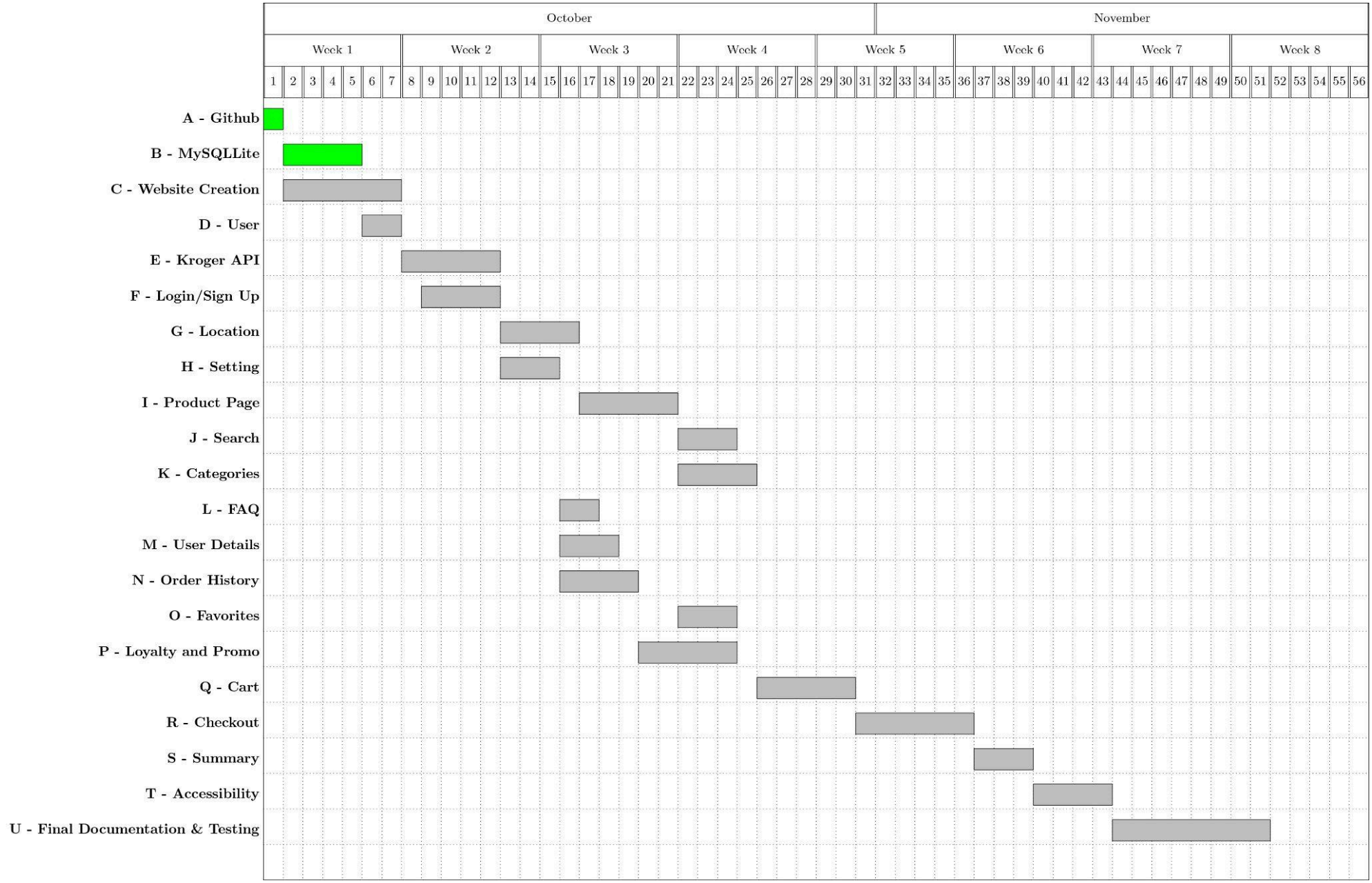

Figure 2. Gantt Chart

3. Risk

This section of the document highlights the risks, associated tasks, and management. By identifying these risks early and implementing the proposed mitigation strategies, the GreenGrocer project has a structured approach to handle challenges, ensuring that the development process stays on track and that the final product meets functional and security standards. Each risk is tied to specific tasks, and the team continues monitoring for any emerging risks as the project progresses.

3.1 Configuration Management:

- *Tasks at Risk:* All tasks that involve collaborative coding, particularly GitHub (Task A) and Website Creation(Task C).
- *Risk Description:* As multiple team members work on different modules, improper configuration management leads to issues such as code overwriting, merge conflicts, or misaligned software environments. This slows down the project and introduces bugs along with inconsistent versions of the application.
- **Mitigation**:
    - Use Git branching strategies (feature branches) to ensure that different tasks are developed in parallel without conflicts.
    - Enforce pull requests and peer code reviews before merging changes to the main branch, ensuring consistency.
    - Implement continuous integration (CI) to automatically detect and fix configuration issues.

3.2 Coding Issues:

- *Tasks at Risk:* MySQL Lite Database (Task B), Website Creation (Task C), Kroger Product API (Task E), Login/Sign-Up (Task F).
- *Risk Description*: Coding bugs and improper implementation cause delays and potentially break the app's functionality. For instance, misconfigurations in the integration of the Kroger API lead to inaccurate product listings, and poor coding practices in the login/sign-up process expose security vulnerabilities within the system.
- **Mitigation**:
    - Regular peer code reviews and debugging sessions to identify issues early in development.
    - Conduct unit testing and integration testing for each feature (e.g., API responses, and database interactions) to ensure they function correctly.
    - Assign pair programming or involve a second team member for critical modules such as API integration or login security.

3.3 Scalability of Database/Server:

- *Tasks at Risk:* MySQL Lite Database (Task B), Cart Management (Task Q), Kroger Product Integration (Task E).
- *Risk Description*: MySQL Lite, being lightweight, may not scale well with a growing number of users. As the app's user base increases, the database and server could slow down under heavy load, affecting user experience, especially in high-traffic components like the shopping cart or checkout system.
- **Mitigation**:
    - Performance testing for MySQL Lite, simulating high-traffic conditions, to

understand its limits.

- Plan for future scalability by upgrading to a more robust database solution (e.g., MySQL or PostgreSQL) if needed.
- Optimize SQL queries and database structure to ensure efficient data retrieval.

3.4 Poor Project Management:

- *Tasks at Risk*: All tasks, especially high-dependency tasks like Website Creation (Task C), Kroger Product Integration (Task E), and User Management (Task D).
- *Risk Description:* Poor project management can lead to missed deadlines, incomplete features, or project scope creep. If roles and responsibilities aren't clearly defined, or task progress is not tracked properly, the project faces delays and inefficiencies.
- **Mitigation**:
  - Use project management tools like Trello to create, delegate, and track progress on tasks, making sure to monitor key dependencies.
  - Hold regular meetings to discuss progress, blockers, and next steps, allowing for agile adjustments as needed.
  - Set clear ownership of each task to avoid overlap and ensure accountability for deliverables.

3.5 Unplanned Changes Risk:

- *Tasks at Risk:* Kroger Product Integration (Task E), Location Management (Task G).
- *Risk Description:* The Kroger API or business requirements may change unexpectedly during development, forcing the team to adapt. Unplanned changes can disrupt the project timeline and require the reworking of completed components.

- **Mitigation**:
    - Stay up to date with external dependencies like the Kroger API by monitoring updates or changes in documentation and API endpoints.
    - Maintain flexibility in the project schedule to accommodate potential rework or changes.
    - Establish a change management process where unplanned changes are discussed, prioritized, and approved by the team before implementation.

3.6 Dependency Risks:

- *Tasks at Risk:* Website Creation (Task C), Cart Management (Task Q), Login/Sign-Up (Task F), Order History(Task N).
- *Risk Description:* Several tasks are dependent on one another, meaning delays in one task could delay others. For instance, Cart Management depends on the completion of both the Product Pages and the Search feature. If one component is delayed, the overall system may not function as intended.
- **Mitigation**:
    - Identify all critical dependencies early in the project planning stage. Ensure that tasks on the critical path are prioritized.
    - Utilize Agile sprints to ensure continuous progress and allow flexibility to accommodate delays in high-dependency tasks.
    - Communicate interdependencies clearly during team meetings to manage expectations and resource allocation effectively.

3.7 User Authentication (Saving Data/Security):

- *Tasks at Risk:* Login/Sign-Up (Task F), User Management (Task D), Order History (Task N), Favorites (Task O).
- *Risk Description:* Improper handling of sensitive user data, such as login credentials, could lead to security vulnerabilities and data breaches. This is particularly concerning for tasks that manage user authentication and sensitive personal information such as email, phone number, and home address.
- **Mitigation**:
    - Implement secure storage for passwords using modern encryption standards (e.g., bcrypt or similar hashing algorithms) and secure data transfers via HTTPS.
    - Ensure Flask-Login sessions are secure and use HTTPS for communication.
    - Regularly review the application for security vulnerabilities and conduct security audits before deployment.

4. Quality Assessment Testing

Testing for quality assurance makes sure the GreenGrocer product as a whole satisfies the requirements specification. To ensure a consistent experience, user experience testing on multiple devices, browsers, and levels of components ensures the application is a high-quality and user-friendly product.

4.1 Unit Testing:

- The GreenGrocer team uses unit testing as the process of examining individual methods, functions, and components to make sure they operate as intended when

tested separately.

- Every Flask Route, form validation, and API call undergoes testing to verify their accuracy. Additionally, every button and added feature undergoes unit testing to ensure appropriate and intended function.

4.2 Component Testing:

- In component testing, the team makes sure that interconnected components, such as user authentication, cart systems, and API-based product listing work well together. Both individually and as part of the overall system, each component is evaluated.
- Component testing is done on both small and large scales to observe cohesive functions with a couple or a significant number of units of the system.

4.3 System Testing:

- To make sure that all of the system's parts function as a whole, system testing assesses the complete system. This involves testing the user experience from registration to checkout, to make sure the app works as planned. During this stage, integration with the Kroger API is a major focus.
- Another part of system testing includes use-case testing where the application is tested based on example usage by different types of users. System testing also certifies that errors are thrown where appropriate and eliminates any user bias of the interface. This testing ensures that the application works in the way it's intended for all features and functionalities.

5. Methodologies

5.1 Agile Framework

The GreenGrocer project adopts the Scrum framework, an agile methodology that emphasizes flexibility and iterative progress. This approach allows the team to adapt quickly to changes and deliver working increments of the application regularly.

- **Sprint Structure**
    - One-week sprints
    - Each sprint aims to produce a working increment of the app
    - Brief team check-ins twice a week to ensure progress and address challenges

5.2 Team Organization

The project employs a flexible team structure to promote shared leadership and effective communication.

- **Leadership Rotation**
    - The meeting facilitator role rotates weekly among group members
    - Ensures equal participation and shared responsibility
- **Communication Channels**
    - Primary communication through shared Discord (similar to Slack)
    - Twice-weekly check-ins
    - Additional meetings scheduled as needed

5.3 Project Management Tools

To facilitate efficient project management and collaboration, the team utilizes the following tools:

- **Miro**
    - Free version for task creation and assignment
    - Progress tracking
    - Activity chart for project overview
    - Identifies areas that may require project crashing
- **GitHub**
    - Version control
    - Team collaboration on code
    - Peer review process
    - Efficient management of app versions

Appendix A: Metrics

- Metrics are obtained from similar GitHub repositories for grocery applications, and data from an article stating the average lines of code produced per day by a programmer of similar experience to the GreenGrocer team. The average lines of code for tasks D, F, and H-T were determined using the GitHub repositories. To deduce the duration of each task, estimated lines of code for each task were divided by the average lines of code written per day (100 lines). This data was strictly used only for programming-related tasks. All other task's durations were estimated from previous projects and earlier documentation. Since this data was not obtained from the team's direct metrics, the project plan will be updated as the project progresses with more precise estimations to determine the schedule to be on track for the final submission by December 3rd, 2024.

Appendix B: Sources

- Lines of code written per day
- https://github.com/ParasGarg/Online-Grocery-Store
- https://github.com/ebarthur/grocery-shop
- https://github.com/kareem983/Grocery-Shopping-System-App
- https://github.com/offfahad/grocery-shopping-app
- https://github.com/hieuwu/android-groceries-store