Project Plan for Pollaris™

Version 1.0

02.27.2024

Team Members: Mofoluwake Adesanya, Ethan Dettmann, Rizwan Khan, Rohit Poolust, Brandon Swirsky

TABLE OF CONTENTS

1. Project Activities and Dependencies

1.1 Requirements Table

The requirements table highlights each task from start to finish of the project development, including the number of days each task would take to be completed, their dependencies, the resources assigned to each task and a brief description of it.

Task	Days	Dependencies	Resources	Task Description	Completed?
T1	1	None	Rizwan	Set up version control - GitHub	Yes
T2	1	None	Rizwan	Set up server	Yes
T3	1	T1, T2	Rizwan	Install dependencies on server	Yes
T4	2	None	Brandon, Foluke, Ethan	Create Landing Page for all Users	Yes
T5	1	T4	Brandon, Foluke, Ethan	Test Landing Page	Yes
T6	2	None	Rohit	Firebase Setup	Yes
T7	2	T6	Rohit	Data tables Creation (Data Types): All data to be tracked	Yes
T8	2	T4, T5	Brandon, Foluke, Ethan	Create Working page for Instructor user class	Yes
T9	2	T8	Brandon, Foluke, Ethan	Test working page	Yes
T10	3	T5,T9	Brandon, Foluke, Ethan	Create Login and Signup pages for Instructor user class	Yes
T11	3	T7, T10	Rizwan, Rohit	Set up Firebase Authentication (Connection between database with	Yes

				Framework)	
T12	2	T3, T11	Rizwan, Rohit	Setting up real-time sockets	
T13	2	T12	Rizwan, Rohit	Test authentication with Firebase with database	Yes
T14	3	T13	All	Sync database with UI/UX	Yes
T15	1	T14	All	Testing Sync	Yes
T16	3	T9	Brandon, Foluke, Ethan	Create question types (multiple choice, written) and Create Test	No
T17	2	T16	Brandon, Foluke, Ethan	Edit/delete test	
T18	1	T17	Brandon, Foluke, Ethan	Test Quiz Creation	Yes
T19	1	T18	Brandon, Foluke, Ethan	Generate test PIN	Yes
T20	1	T19	Brandon, Foluke, Ethan	Create publish and toggle buttons	
T21	2	T5	Brandon, Foluke, Ethan	Create Student Login Page	Yes
T22	3	T21	Brandon, Foluke, Ethan	Create Student Quiz Interface	Yes
T23	2	T12, T15, T22	Rizwan	Test Real Time Sockets	
T24	4	T15, T22	All	Sync database with UI/UX II	Yes
T25	1	T20, T24	All	Link Instructor code to Students = Take a Quiz	Yes
T26	2	T25	All	Create 'Generate	

				report" for test taken by Students (email to Instructor)	
T27	4	T23, T24, T26	All	Final Testings (Combining all frameworks)	

Table 1: Requirements Table

Pollaris's team begins development on Wednesday, February 28, 2024, working 2 to 3 hours for 5 days per week till Tuesday, April 26, 2024.

1.2 Activity Dependency Chart

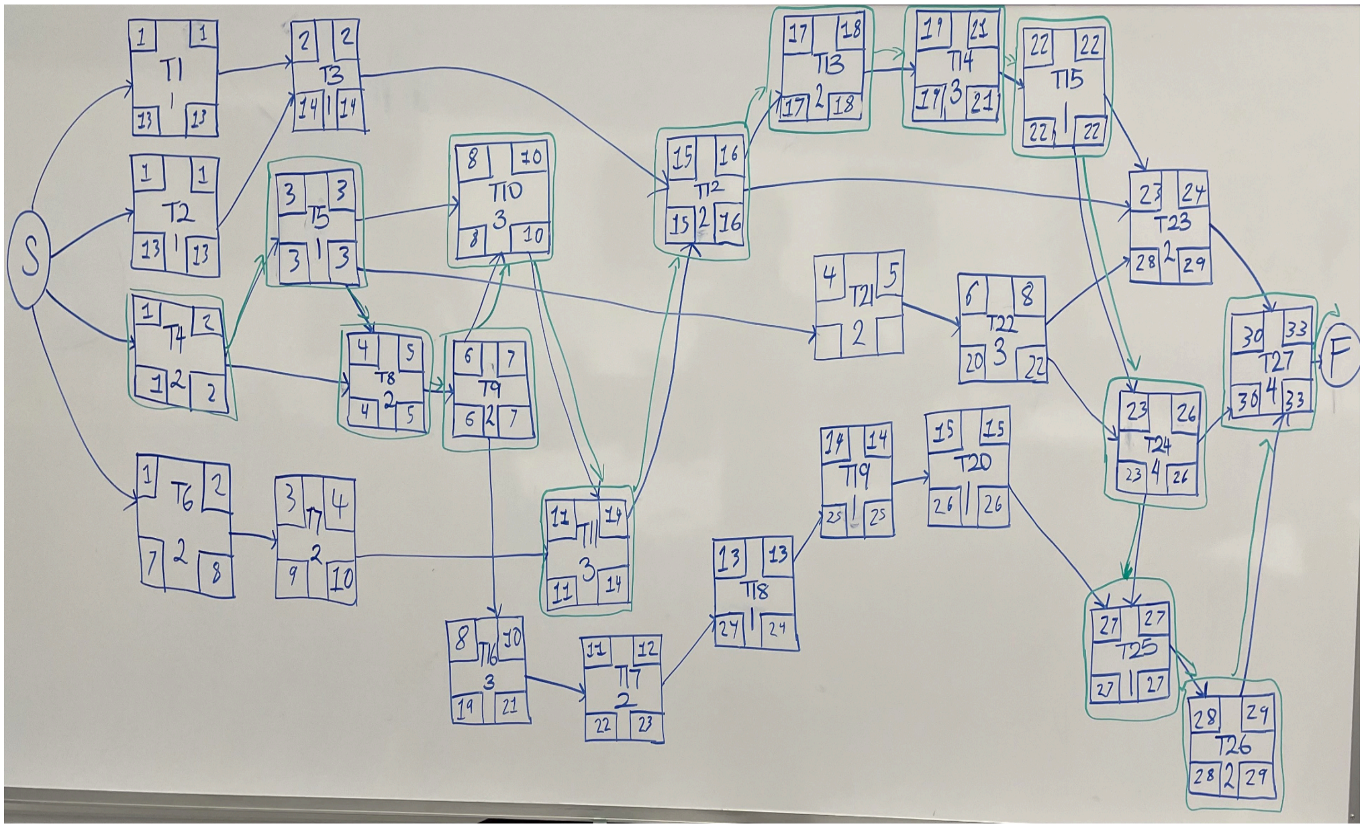

Figure 1 : Activity Dependency Chart with Critical Path highlighted in green

1.3 Requirements Description

T1 - Setup version control - GitHub:
- Establish a GitHub repository for code versioning.

T2 - Set up server:
- Configure the hosting server environment.

T3 - Install dependencies on the server:
- Install necessary software dependencies.

T4 - Create a Landing Page for all Users:
- Develop a universal landing page.

T5 - Test Landing Page:
- Conduct comprehensive landing page testing.

T6 - Firebase Setup:
- Implement Firebase services.

T7 - Data tables Creation (DataTypes) All data we need to keep track of:
- Define and create required data tables.

T8 - Create a Working page for the Instructor user class:
- Design a functional instructor page.

T9 - Test working page:
- Perform instructor page testing.

T10 - Create Login and Signup pages for Instructor user class:
- Develop secure login/signup pages.

T11 - Set up Firebase Authentication (Connection between database with Framework):
- Configure Firebase authentication.

T12 - Setting up real-time sockets:
- Implement real-time socket connections.

T13	- Test authentication with Firebase with database: - Verify Firebase authentication.
T14	- Sync database with UI/UX (Look at the chart in discord) - Synchronize database with UI/UX.
T15	- Testing Sync: - Test database/UI/UX synchronization.
T16	- Create question types (multiple choice, written) and Create Test: - Develop various question types and test creation.
T17	- Edit/delete test: - Implement test editing/deletion features.
T18	- Test Quiz Creation: - Test quiz creation functionality.
T19	- Generate test PIN: - Implement PIN generation for tests.
T20	- Create publish and toggle buttons: - Develop publish and toggle buttons.
T21	- Create Student Login Page: - Design student login page.
T22	- Create Student Quiz Interface: - Develop a student quiz interface.
T23	- Test Real-Time Sockets: - Conduct real-time socket testing.
T24	- Sync database with UI/UX II: - Further, refine database/UI/UX sync.
T25	- Link Instructor code to Students = Take a Quiz:

- Enable students to access instructor quizzes.

T26 - Create ‘Generate report’ for the test taken by Student (email):
- Develop test report generation feature.

T27 - Final Testings (Combining all frameworks):
- Conduct final comprehensive testing.

1.4 Gantt Chart

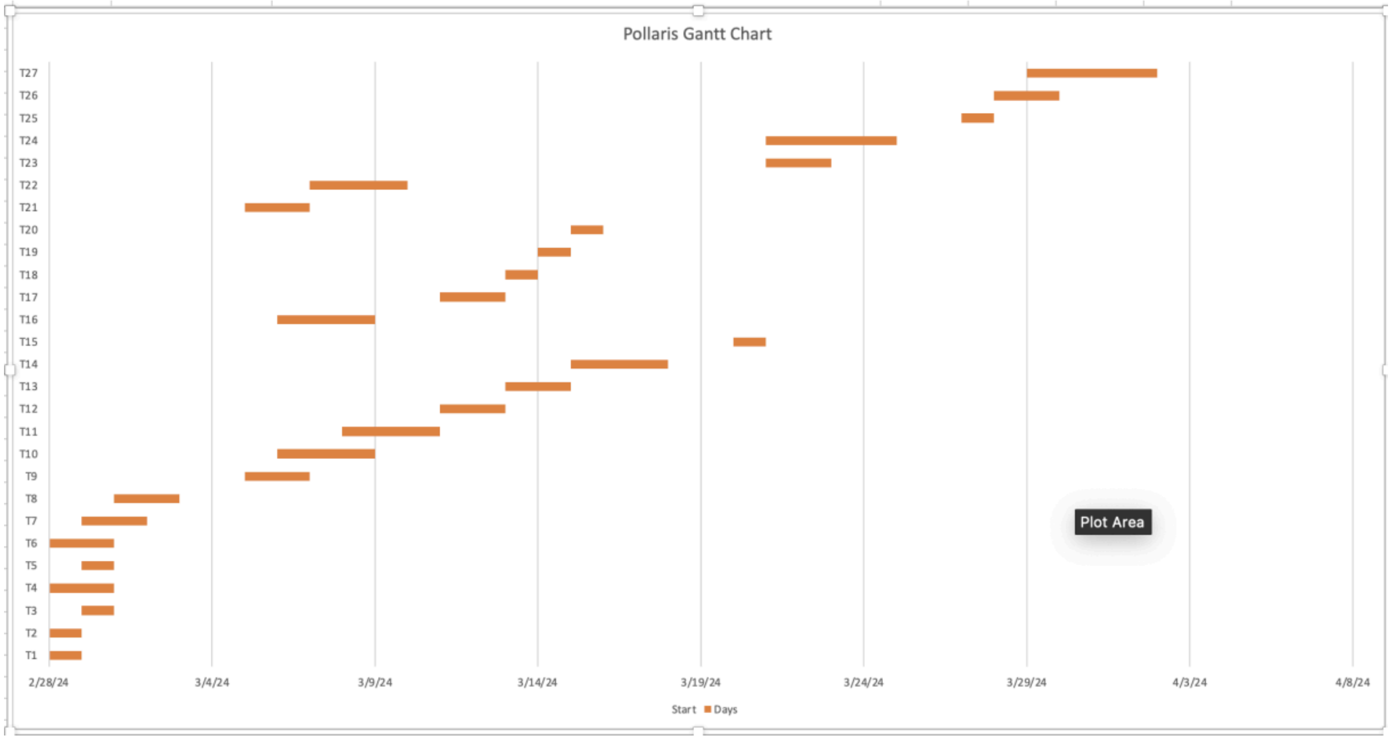

Figure 2 : Gantt chart generated from Figure 1 (Y-axis represents Requirements; X-axis represents Time in weeks)

2. Risks

2.1 Risk Identification and Classification

The risks associated with website development are classified according to high, moderate and low risks depending on if they are risks associated with critical services or features and how they interact with data.

- **High Risk:** Risks are considered to be *high risk* if they are associated with processes that store private data or/and provide a critical service. The compromise of such processes have a significant negative impact on the overall safety of all users and their systems, and on Pollaris's system and reputation. High risks related to the development of Pollaris are:
    - ***Coding Issues***: Bugs in the frontend code of a website produce risks like data leaks, loss of information and cyber attacks. Improper planning, prototyping and testing leaves coding issues unchecked and mitigated. A workforce with limited skills and development can only contribute to coding issues and a failing website. This is a high risk since Pollaris's services are expected to be secure to users and administrators. Pollaris seeks to avoid harm to users and maintain the wellbeing of the computing community.
    - ***Broken authentication***: This refers to insufficient security during authentication. This manifests in various ways including access control issues, insufficient encryption of personal data in transmission and storage, injection of malicious code, use of insufficient user verification techniques, request forgery and many more.

- **Moderate Risk:** Risks are classified as *moderate risk* if they are related to processes that provide a fairly important service that is not *low risk*. These processes relate with internal data and can be internally trusted by other processes.
    - ***Poor project management***: This risk produces an unstable project or failure to meet deadlines. This is due to poor team coordination and communication, lack of analytical skills, and poor risk management.
- **Low Risk:** Risks are classified as *low risk* if they are related to processes that operate on public data and do *not* provide critical services.
    - ***External risks***: These are risks that stem from market changes, competitors, new or changing government regulations, changing consumer needs and requirements and other risks from external factors outside the control of Pollaris. This is a low risk because it is considered a fairly rare occurrence and unpredictable.

2.2 Risk Management and Mitigation

These includes the strategies that are applied to mitigate the risks highlighted in 6.1:

- To mitigate the risk of ***coding issues***, Pollaris identifies the specific needs of each service it provides and the existing tools and engineering techniques that provide the best solutions for each service. Pollaris is built using HTML, CSS and Javascript to ensure the interactivity, accessibility and accuracy of its services. It also utilizes Google's Firebase database to preserve the cybersecurity of end users.

- There are numerous vulnerabilities in website development and security is the highest priority. Techniques to eradicate all cyber vulnerabilities are still being researched so some vulnerabilities can not be completely removed. To manage ***broken authentication***, Pollaris leverages Firebase's database (*Firestore*) which is a no-SQL database that prevents risks like SQL injection. HTTPS ensures secure communication and privacy between a server and a client. Pollaris's website is HTTPS-protected to prevent *eavesdropping* during transmission of data between Pollaris's users and its server. Moreso, Pollaris's workforce ensures that its deployment code is robust and that best practices are applied to its full stack development. Code is reviewed for efficiency and congruence with Internet safety standards.
- To improve ***project management***, Pollaris's team strives to maintain high standards of strategic planning using the CMMi framework. Sufficient documentation is used to manage team changes, and prevent delays.
- To manage ***external risks*** like market changes, or changing consumer needs, market research and worldwide tendencies are considered for patterns that are beneficial for Pollaris. Active learning and engagement in classrooms is highly valued across all stages of student education and Pollaris provides a solution to enhancing classroom learning.

3. Quality Assurance Planning

3.1 Testing Strategies

Various testing strategies are used in order to ensure Pollaris is a high quality product.

- **Use-Case Testing** is used in order to make sure each feature works as intended as if a normal user was using it. Each new feature is tested thoroughly for all possible cases, throwing errors where appropriate. Later in Pollaris' development, unbiased users outside of its creation are used to ensure the average user is capable of operating on the web page. Use-Case Testing is employed whenever a major task or set of tasks is completed.
- **Direct Testing** is employed in order to ensure that, as minor features are added, the program still works as intended. Smaller-scale tests are done to make sure Pollaris is in working condition for a majority of its development, and any errors can be addressed as soon as possible.
- **Interface Testing** is important, as Pollaris relies heavily on its web pages. Each page's interface is thoroughly tested to make sure buttons perform the features they should, the user is sent to the correct page when navigating the website, and other aspects of the interface that affect the functionality. Interface testing also ensures that there aren't any discrepancies between elements on the page and the database.
- **Regression Testing** is done whenever changes are made to Pollaris. This is to ensure that, even if a section was considered functional, it doesn't become

nonfunctional after updates are made to the code. The original code is well documented and recorded, such as on GitHub, so that it is not lost if still needed.

4. Configuration Management Strategy

Pollaris's file management and version control are handled by Github. Each team member has their own version of code files within their machine. To work on a new version of the project, team members pull from the Github repository. Team members can also edit files locally before pushing them back to the repository for validation by the team. To set this up, each team member has access to the repository with push and pull permissions from their device. From certain requirements within the project plan, multiple team members are to edit the same files. Team members are to collaborate to resolve merge conflicts between differing files and setups. Team members are subjected to commit to a specified layout for programming files to accommodate commits to Github.

Branching is set to be utilized to foster parallel developments. The process of branching encompasses the creation of new code lines from pre-existing ones, which fosters independent growth for both the new and original code lines. Such an approach allows us to engage in work on fresh features or fixes without compromising the stability of our primary codebase.

Every addition or alteration in Pollaris signifies a new version of the configuration item. These unique instances are differentiated by introducing fresh features, resolving bugs, or upgrading existing functionalities. Each version carries a distinct identification that offers a unique reference point which is crucial for robust change tracking and maintaining consistency across the project.

In parallel, to ensure the reliability of the project, regular backups of the server are to be scheduled. These backups are stored in an offsite location in an event to protect against data loss in case of server failure. GitHub and the server supports the team's backup strategy. Regular testing of the backup must occur regularly to ensure integrity.

Team members have their independent workspace where software parts can be modified without affecting other members' existing work. This ensures that individuals can work independently and experiment with changes without impacting the main codebase.

Appendix A: Glossary

- HTML, CSS and Javascript: This set of languages is used for the frontend website development of Pollaris's interface.
- HTTPS: Hypertext Transfer Protocol is an application layer protocol that governs transmission of data between devices on a network e.g. the World Wide Web. HTTPS is the secure version of HTTP.
- CMMi: Capability Maturity Model Integration is a level improvement program to improve software engineering processes and the quality of software.
- Github: This is a platform that allows developers to manage and process software versions.
- Firestore: This is a no-SQL document database that is scalable and serverless, providing security, robustness, and ease of implementation.

Appendix B: Work Split

Name	Contributions
Brandon	Quality Assurance Management, Requirements Table
Rizwan	Configuration Management Strategy
Foluke	Cover Page, Table of Contents, Risks
Ethan	Configuration Management Strategy, Activity Dependency Chart
Rohit	Requirements Description, Gantt Chart

Table 2 : Table shows the work contributions of each member of the team to this document