CIS 1512: Software Engineering

Gig App

(A Local Gig Jobs Platform)

Software Requirements Specification (SRS)

& Software Project Management & Planning (SPMP)

Prepared by: Gig App Team

Professor: Hadi Nasser

Table of Contents

- 6. SPMP - Project Management Plan
    - 6.1 Project Estimation
    - 6.2 Project Schedule
    - 6.3 Risk Management
    - 6.4 Staff Organization
- 7. Appendices

Software Requirements Specification (SRS) and Software Project Management Plan (SPMP)

1. Introduction

This document provides the Gig Work App's Software Requirements Specification (SRS) and the Software Project Management Plan (SPMP) to provide a comprehensive overview of the project. The app will allow users to post short-term gigs and workers to apply for them. The goal is to create a simple, responsive web application prototype within a 1–2 month timeframe.

1.1 Purpose

This application's purpose is to connect people offering short-term work opportunities (clients) with people seeking such work (workers). This version will focus on core functionality, user accounts, gig posting, browsing, and applying.

1.2 Scope

The Gig Work App is a prototype web application. The scope includes:

- User registration and login
- Post a gig
- Browse available gigs
- Apply for a gig
- Basic responsive design for desktop and mobile browsers

1.3 Overview of Document

SRS Section:

- Introduction
- Functional requirements
- Non-functional requirements
- System models
- Used cases

SPMP Section:

- Covers project planning
- Schedule
- Resources
- Risk analysis
- Process management

1.4 Intended audience

This document is intended for project stakeholders, including developers, project managers, quality assurance teams, clients, and workers who will interact with or review the Gig Work App.

2. Overall Description

The Gig Work App will have two main user roles: Clients and Workers. Clients can create gigs, while Workers can browse and apply. The app will use a simple relational database to store user and gig data.

2.1 Product Perspective

The Gig Work App is a web-based system that follows a client-server model. The frontend will be implemented using HTML, CSS, and JavaScript, while the backend will use Flask or Node.js. A relational database (SQLite for lightweight deployments or MySQL for production environments) will store persistent data, such as gig postings, applications, reviews, user accounts, and notifications. The Gig App will be designed as a stand-alone system but may integrate with third-party services for notifications, email verification, and, in future iterations, payment processing.

2.2 Product Functions

At a high level, the application will support the following functions:

- **User Registration/Login:** Create/manage user accounts
- **Job Posting:** Clients can create and publish new gigs
- **Job Browsing and Application:** Workers can search, filter, and apply for gigs
- **Review and Badge System:** Users can leave reviews; badges will highlight high-performing workers
- **Notifications and Messaging:** Users will receive alerts for new gigs, applications, and status changes
- **Responsive Design:** The system will adapt to desktop and mobile web browsers for broad accessibility

2.3 User Classes and Characteristics

- **Clients (Job Posters):** Post jobs, review applications, select workers, and leave feedback. Typically motivated by ease of posting and reliability of workers
- **Workers (Job Seekers):** Browse available gigs, apply for roles, and build credibility through reviews and badges. Will be motivated by opportunity, fair evaluation, and visibility
- **Administrators:** Monitor system activity and ensure compliance with community standards.

2.4 Operating Environment

- **Frontend:** Web browsers such as Chrome, Firefox, Safari, and Edge
- **Backend:** Flask (Python) or Node.js
- **Database:** SQLite (development/testing) or MySQL (production)
- **Supported Platforms:** Windows, macOS, Linux, and mobile web browsers on IOS/Android
- **Connectivity:** Requires stable internet access

2.5 Design and Implementation Constraints

- **Frontend -** must be implemented using HTML, CSS, and JavaScript
- **Backend -** must use Flask or Node.js
- **Data** to be stored in a relational database (SQLite/MySQL)
- **Passwords -** must be secure (hashed and salted)
- App must comply with basic data security and privacy standards

2.6 Assumptions and Dependencies

- Notifications may rely on third-party APIs (email/SMS/etc)
- The system depends on the backend server and database being operational
- Users will have consistent internet connectivity

3. Functional Requirements

The system shall include the following functions:

3.1 User Registration/Login

- Users can create accounts with an email and a password.
- Users can log in securely.

3.2 Job Posting and Browsing

- Clients can post gigs with title, description, location, and payment info.
- Workers can apply for gigs.
- Applications will be stored in the database.

3.3 Review & Badge System

- Allow clients to provide feedback on gigs they've booked.
- Reward service providers with badges for achievements or positive feedback

3.4 Notifications & Messaging

- Keep clients and service providers informed about gig-related activities.
- Enable direct communication between clients and service providers.

3.5 Responsive Design

- The app will be usable on both desktop and mobile browsers.

4. Non-Functional Requirements

4.1 Performance

- **Page Response Time:** Core pages (Home, Search, Booking, Profile) load in less than or equal to five seconds under a light to moderate load. The prototype is ready and prevents user drop-off.

- **API Processing:** Login, booking, and review submissions return in less than two seconds, ensuring smooth user interaction without requiring heavy server optimization.
- **Scalability:** Architecture designed to support up to 200 concurrent users. Demonstrates growth planning without the need for full enterprise-level stress testing.
- **Database Throughput:** Sustain greater than or equal to 20 transactions per second during peak usage, which supports basic bursts of activity.

4.2 Security

- **Data Encryption:** HTTPS (TLS 1.2+) for all traffic; passwords hashed with bcrypt greater than or equal to 12. Protects credentials and user data using standard, library-supported methods.
- **Authentication:** JWT for sessions. Provides a simple, secure way to verify users and prevent unauthorized access.
- **Access Control:** Role-based access for Client, Worker, and Admin. Enforces least-privilege principles.
- **Privacy Compliance:** A clear consent message modeled on GDPR/CCPA. This builds user trust and demonstrates awareness of privacy regulations.

4.3 Usability

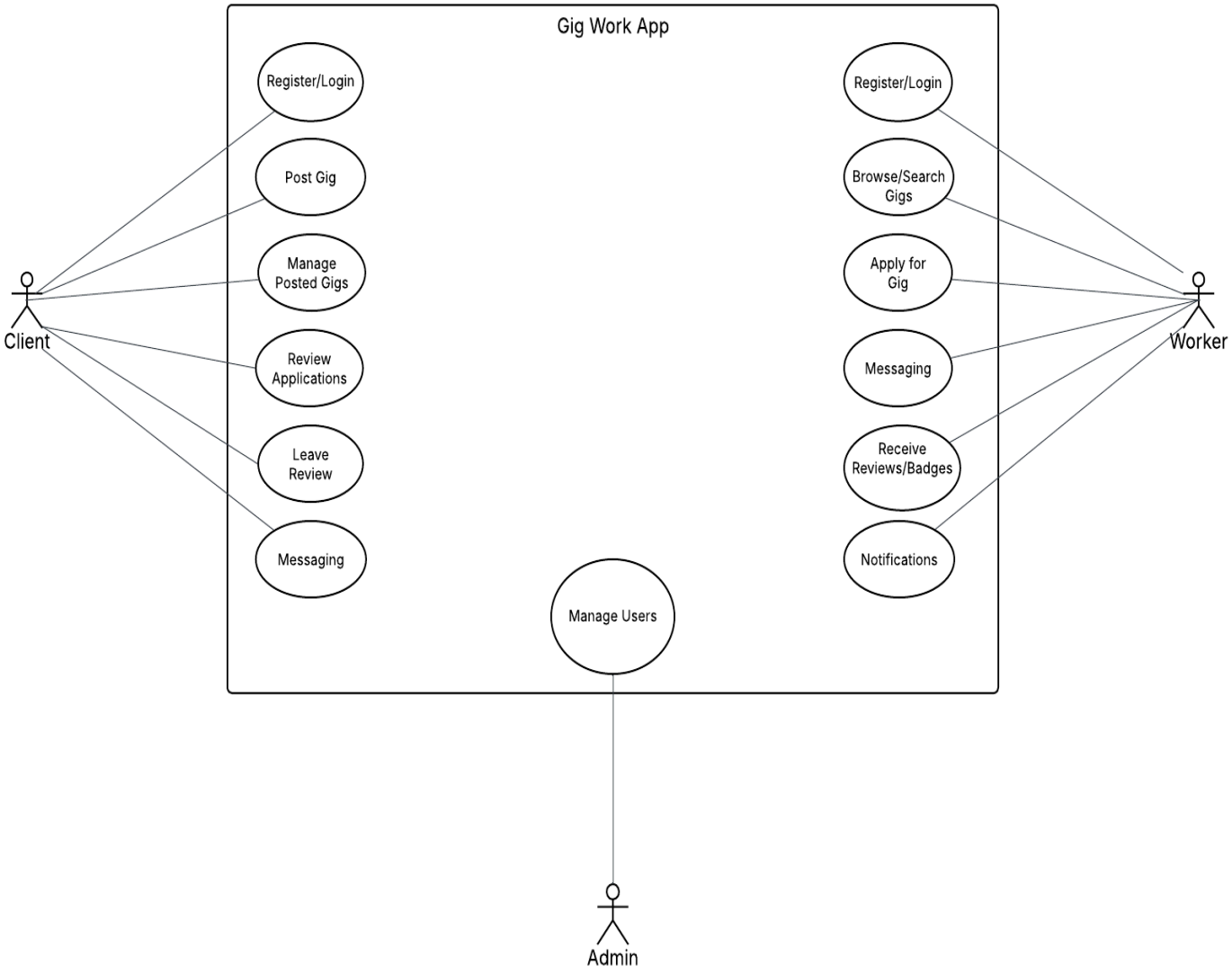
- **User Onboarding:** Complete registration in less than or equal to five minutes across less than or equal to three screens. Reduces abandonment and keeps the sign-up process manageable.
- **Navigation:** Key tasks (Post, Search, Book) are reachable in less than or equal to three clicks, improving efficiency without the need for complex menu logic.
- **Accessibility:** Provide keyboard navigation and a readable color contrast.
- **Cross-Browser:** Fully functional on the latest Chrome and Firefox.

4.4 Reliability

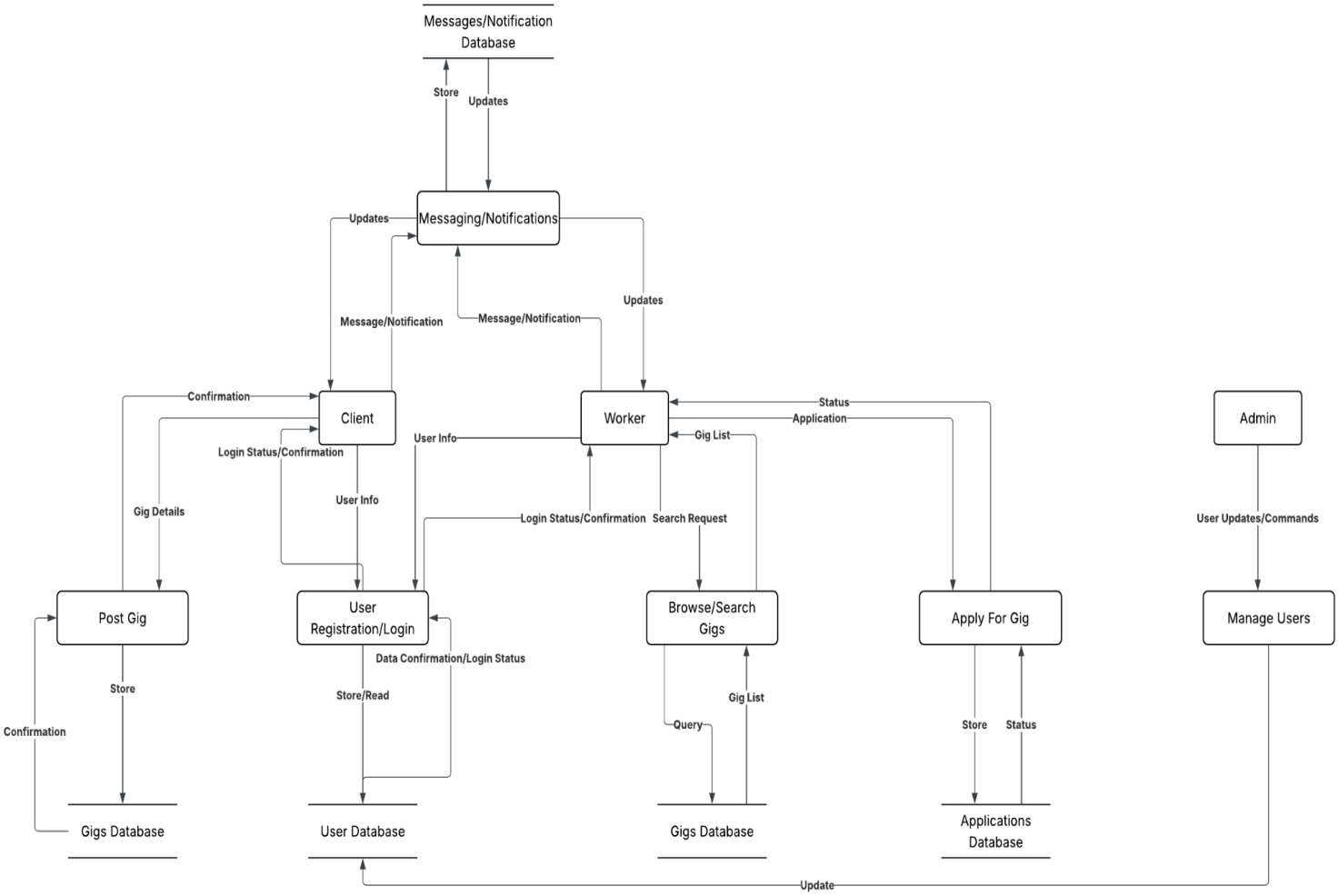
- **Uptime:** Greater than or equal to 98% availability during the project and demo period, providing professional reliability using student-level hosting.
- **Error Handling:** Display user-friendly error messages and roll back failed actions, preserving data integrity and preventing user confusion during testing.
- **Backups:** Automated daily database backup retained for 7 days. This enables quick recovery with simple local or cloud scripts.
- **Recovery Time Objective:** Restore core service within ≤ 4 hours of critical failure, minimizing downtime.

5. System Models

5.1 Use Case Diagrams (The app will be deployed locally or on a small cloud host for testing.)



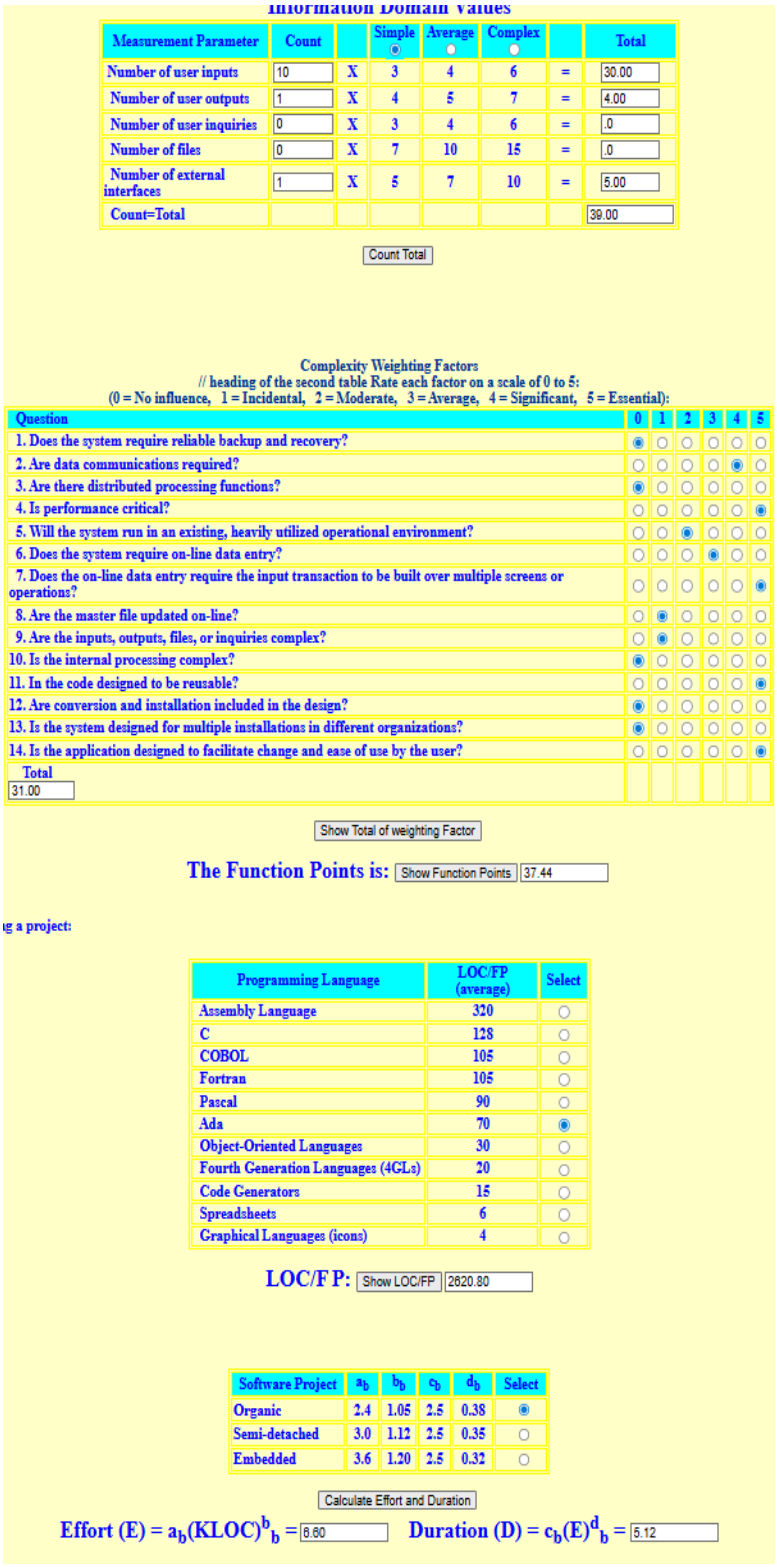
5.2 Data Flow Diagrams (No payment processing will be included in this prototype.)



6. SPMP - Project Management Plan

6.1 Project Estimates

- COCOMO CALCULATIONS
  - Effort (E) = ab(KLOC)bb = 6.60
  - Duration (D) = cb(E)db = 5.12



Information Domain Values

Measurement Parameter	Count		Simple	Average	Complex		Total
Number of user inputs	10	X	3	4	6	=	30.00
Number of user outputs	1	X	4	5	7	=	4.00
Number of user inquiries	0	X	3	4	6	=	.0
Number of files	0	X	7	10	15	=	.0
Number of external interfaces	1	X	5	7	10	=	5.00
Count=Total							39.00

Count Total

Complexity Weighting Factors
// heading of the second table Rate each factor on a scale of 0 to 5:
(0 = No influence, 1 = Incidental, 2 = Moderate, 3 = Average, 4 = Significant, 5 = Essential):

Question	0	1	2	3	4	5
1. Does the system require reliable backup and recovery?	◉	○	○	○	○	○
2. Are data communications required?	○	○	○	○	◉	○
3. Are there distributed processing functions?	◉	○	○	○	○	○
4. Is performance critical?	○	○	○	○	○	◉
5. Will the system run in an existing, heavily utilized operational environment?	○	○	◉	○	○	○
6. Does the system require on-line data entry?	○	○	○	◉	○	○
7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?	○	○	○	○	○	◉
8. Are the master file updated on-line?	○	◉	○	○	○	○
9. Are the inputs, outputs, files, or inquiries complex?	○	◉	○	○	○	○
10. Is the internal processing complex?	◉	○	○	○	○	○
11. In the code designed to be reusable?	○	○	○	○	○	◉
12. Are conversion and installation included in the design?	◉	○	○	○	○	○
13. Is the system designed for multiple installations in different organizations?	◉	○	○	○	○	○
14. Is the application designed to facilitate change and ease of use by the user?	○	○	○	○	○	◉
Total 31.00						

Show Total of weighting Factor

The Function Points is: Show Function Points 37.44

g a project:

Programming Language	LOC/FP (average)	Select
Assembly Language	320	○
C	128	○
COBOL	105	○
Fortran	105	○
Pascal	90	○
Ada	70	◉
Object-Oriented Languages	30	○
Fourth Generation Languages (4GLs)	20	○
Code Generators	15	○
Spreadsheets	6	○
Graphical Languages (icons)	4	○

LOC/F P: Show LOC/FP 2620.80

Software Project	a_b	b_b	c_b	d_b	Select
Organic	2.4	1.05	2.5	0.38	◉
Semi-detached	3.0	1.12	2.5	0.35	○
Embedded	3.6	1.20	2.5	0.32	○

Calculate Effort and Duration

Effort (E) = $a_b(KLOC)^b_b$ = 6.60 Duration (D) = $c_b(E)^d_b$ = 5.12

6.2 Project Schedule

Phase / Task	Assigned To	Start Date	End Date	Status
Phase 1.1: Project Proposal	Entire Team	09/02/2025	09/08/2025	Completed
Phase 1.2: SRS/SPMP (Specification & Planning)	Entire Team	09/09/2025	09/22/2025	Completed
Phase 1.3: Prototype Demo	Cordero & Sean	09/23/2025	09/29/2025	Completed
Phase 2: Design	Luca & Melissa	09/30/2025	10/06/2025	Completed
Phase 3: Implementation	Cordero, & Sean (Backend) Melissa & Tamara (Frontend)	10/07/2025	11/03/2025	In Progress
Phase 4.1: V&V (Testing)	Entire Team	11/04/2025	11/10/2025	Not Started
Phase 4.2: Maintenance (Future Enhancements)	Luca & Tamara	11/11/2025	11/17/2025	Not Started
Phase 4.3: Post-Mortem (Individual)	All Team Members	11/18/2025	11/24/2025	Not Started
Final Project Presentation	Entire Team	12/01/2025	12/01/2025	Not Started

6.3 Risk management

Risk Name	Risk Description	Impact	Probability	Mitigation	Contingency Plan
DATA BREACH	Sensitive user data could be leaked	High	Medium	Use encryption, secure authentication	Shut down servers, reset passwords
MISSED DEADLINES	Development tasks take longer than expected	Medium	High	Break work into sprints, weekly meetings	Reallocate tasks, extend schedule
LACK OF ACTIVE USERS	Fewer users sign up or use the app than expected.	High	Medium	Invest in marketing, offer referral incentives	Reconfigure business models, target new markets, increase promotional campaigns.
SCOPE CREEP	New features needed mid-project	Medium	Medium	Use change request approvals, update testing	Adjust schedule and resources accordingly

6.4 Staff Organization

Team Member	Roles
CORDERO	*SME, UI/UX Designer, Backend, Testing*
MELISSA	*Documentation, Frontend, Testing*
LUCA	*DevOps/Database, Testing, Scrum Master*
TAMARA	*Documentation, Frontend, Analyst*
SEAN	*UI/UX Designer, Backend, Analyst*
PROF. NASSER	*Business Owner, Stakeholder*

7. Appendices

Configuration Management

- Using Google Docs for collaboration.
- GitHub repository for code and documentation.
- Discord as primary communication

Project Resources

- Online tutorials, class slides, and developer guides.
- **Minimal Hardware Requirements**
    - User: Smartphone (Android/iOS), 4GB RAM.
    - Developer: Laptop/PC, 8GB RAM, 256GB storage.
- **Minimal Software Requirements**
    - User: Latest iOS/Android OS, stable internet.
    - Developer: VS Code, Git, Node.js, Database (e.g. MySQL).

References

- Class notes/lectures
- SRS Template http://www.rspa.com/docs/Reqmspec.html
- SPMP Template http://www.rspa.com/docs/Projectplan.html
- COCOMO Calculator http://groups.umd.umich.edu/cis/course.des/cis525/js/f00/gamel/cocomo.html