Software Design Document

for

CityMeet

Version 5 approved

Prepared by Darshan Patel, Elliot Gong, Alan Mai, Alejandro Vargas, Vivian Casas, Isidoro Flores, Deion Stapleton, Siying Chen

CS 3337 Group 5 Projet

30 April 2025

Revision History

Name	Date	Reason For Changes	Version
Darshan Patel, Elliot Gong, Alan Mai, Alejandro Vargas, Vivian Casas, Isidoro Flores, Deion Stapleton, Siying Chen	3/12/2025	Fill in the initial sections, minus sections 6 and 7.	1
Darshan Patel, Elliot Gong, Alan Mai, Alejandro Vargas, Vivian Casas, Isidoro Flores, Deion Stapleton, Siying Chen	3/19/2025	Update sections with new content from features and stories.	2
Darshan Patel, Elliot Gong, Alan Mai, Alejandro Vargas, Vivian Casas, Isidoro Flores, Deion Stapleton, Siying Chen	3/26/2025	Update sections with new content from features and stories.	3
Darshan Patel, Elliot Gong, Alan Mai, Alejandro Vargas, Vivian Casas, Isidoro Flores, Deion Stapleton, Siying Chen	4/8/2025	Update sections with new content from features and stories. Add new diagrams and graphs for architecture/design sections.	4
Darshan Patel, Elliot Gong, Alan Mai, Alejandro Vargas, Vivian	4/17/2025	Update sections with new content from features and stories. Add new diagrams and graphs for architecture/design sections.	5

Casas, Isidoro Flores, Deion Stapleton, Siying Chen			
Darshan Patel, Elliot Gong, Alan Mai, Alejandro Vargas, Vivian Casas, Isidoro Flores, Deion Stapleton, Siying Chen	4/23/2025	Update sections with new content from features and stories. Add new diagrams and graphs for architecture/design sections.	6
Darshan Patel, Elliot Gong, Alan Mai, Alejandro Vargas, Vivian Casas, Isidoro Flores, Deion Stapleton, Siying Chen	4/30/2025	Update sections with new content from features and stories. Add new diagrams and graphs for architecture/design sections.	7

1. Introduction

1.1 Purpose

This document will review the general software architecture and layout of CityMeet, a web-based social media platform. A majority of this document will cover the project's first version as a whole, highlighting both abstract and specific details of the system design and implementation. Additionally, this document will refer to the decisions and philosophies that influence how the system will fulfill the established user and system requirements.

1.2 Document Conventions

This document was written in the Times New Roman typeface with a font size of 12.

1.3 Intended Audience and Reading Suggestions

This document is intended for those possessing a basic to intermediate-level understanding of software engineering and user design. This software design document contains information about the project features/scope and the tools/techniques needed to produce these components. The document intro and system overview can be found in sections 1 and 2, which are then followed by the architectural strategies and models in sections 3 and 4. Section 5 goes over the policies and tactics that have influenced CityMeet's software architecture. Sections 6 and 7 offer a more detailed look at the system design and components. In section 8, readers can learn about the database architecture, which will explain how information is organized and layered. Section 9 covers the product's user interface while section 10 lists the user and system requirements that must be satisfied. Lastly, sections 11 and 12 highlight the glossary and references, which provide information about any relevant internal and external information that make up this document.

Below are some reading suggestions based on possible user backgrounds:

- Project Managers, Marketing - 1. Introduction, 5. Policies and Tactics, 10. Requirements Validation
- Documentation - 2. Design Considerations, 3. Architectural Strategies
- Developers - 4. System Architecture, 6. Detailed System Design, 7. Detailed Lower level Components, 8. Database Design
- UX/UI - 2. Design Considerations, 9. User Interface, 10. Requirements Validation
- Users - 1. Introduction, 9. User Interface, 10. Requirements Validation

1.4 System Overview

The CityMeet website will be organized using the Model-View-Control Architectural pattern to handle client and server interactions on a web-based browser. Using a web browser, users will be able to log in with username/password credentials to gain access to the main services provided by CityMeet. Once the server approves the login request and loads up the content, the user can choose to complete a variety of social media activities such as joining groups, posting content, or

changing their account settings. Such actions will allow them to manipulate the existing databases involved with CityMeet. However, requesting, adding, and changing data is not the only activity that can be done. Users can also open links to third-party/official services that provide services outside of the application. The product will utilize open-source/licensed software and tools such as the Mapbox API, Supabase, and Node.js.

Below is a simplified list of the functional and nonfunctional aspects of CityMeet.

Functionality:

- Create, edit, and delete user accounts.
- Upload, edit, and delete user content for recreational or promotional purposes.
- Find content via search and filters
- Create, edit, and join groups.
- Utilize mapping and search features to locate content.
- Apply to and post job openings.
- Report and flag content if necessary.
- Access official content/resources from the Los Angeles City government.
- Send individual/group messages
- Enable 2-factor authentication

Design and Architecture:

- User-centric layout with an emphasis on intuitive navigation and simplicity.
- Switch between map and list view for content.
- Prioritization of user privacy and security.

2. Design Considerations

This section describes the factors that must be considered and resolved before proceeding with any potential design solution.

2.1 Assumptions and Dependencies

Assumptions:

- Users will use CityMeet on standard web browsers(Google Chrome, Microsoft Edge, and Safari) that support HTML, CSS, and Javascript.
- Users will have access to CityMeet primarily through desktops and mobile devices that support the latest web tech.
- Users will have a basic understanding of navigating social media platforms as well as using web applications
- Users will enable geolocation services on their desktops or mobile devices to use the geospatial features effectively that CityMeet will offer

Dependencies:

- CityMeet will require a stable internet connection or cellular data.
- CityMeet will require third-party APIs for geospatial data, authentication, and social media sharing
- CityMeet will need to implement security protocols to protect user data, for example, HTTPS
- CityMeet will need regular updates and maintenance for both frontend and backend to make sure that it remains compatible and evolves with web standards and user needs

2.2 General Constraints

- Hardware or software environment
  - The *constraint* is that CityMeet needs to be consistent with all the modern web browsers such as Edge, Chrome, Safari, and smartphone browsers
  - The *impact* would be that our design's features have to work seamlessly with all the different web browsers on desktops or smartphones
- End-user environment
  - The *constraint* is that CityMeet will be accessed by users through different devices such as smartphones, tablets, laptops, and desktops
  - The *impact* would be that the user interface has to be smart and responsive to adapt to the different screen sizes and their corresponding resolutions in order to have a consistent user experience across all these devices.
- Interoperability requirements
  - The *constraint* is that CityMeet needs Third-Party services for the geospatial data and social media sharing
  - The *impact* would be that our design has to be compatible with these APIs and handle any changes or updates to them without interrupting the service for users
- Security requirements (or other such regulations)
  - The *constraint* is that CityMeet has to implement security protocols to protect user data and protect against unauthorized access
  - The *impact* would be that our design has to include things like encryption, HTTPs, etc.
- Network communications
  - The *constraint* is that CityMeet requires a stable internet connection to receive real-time updates and real-time interactions

    - The *impact* would be that our design has to include some offline modes and data synchronization when the connection is restored should there be network interruptions
- Verification and validation requirements (testing)
    - The *constraint* is that CityMeet has to go through testing to make sure that functionality, security, and performance are up-to-date and work well
    - The *impact* would be that our design needs regular testing, and regular updates to identify and fix issues promptly

You will not need to include all of these. Only the ones that will influence the design of your software

2.3 Goals and Guidelines

Goals

- To have ready-to-deploy software by the end of the semester
- To have CityMeet work seamlessly with different screen sizes and browsers
- To have a smart and user-friendly interface for our software, CityMeet, as well as make it accessible to our targeted user base

Development Guidelines

- Let everyone know about pull requests and merge so nobody will be blindsided.
-

2.4 Development Methods

Development Methods:

- Scrum/Agile Method is a development method that we committed to for our CityMeet software due to its flexibility and iterative approach. We applied Sprint Planning, where we divided the project into multiple weeks, each lasting one week. In Sprint planning, we divide and designate tasks to members as well as discuss clear goals for each of the sprints. We also have daily scrums to help assist with each of the tasks and have open communications with other members in order to meet our Sprint goals. We also have Sprint reviews and retrospectives where we discuss, at the end of each sprint, and review our goals and if they were met. We also go over any feedback we may have and reflect on it so that we may implement and improve for the following Sprint.

3. Architectural Strategies

3.1 Technology Stack

- **Frontend:** React.js for dynamic and responsive UI components.
- **Backend:** Node.js with Express.js for handling API requests and business logic.
- **Database:** PostgreSQL for structured data storage, combined with Redis for caching frequently accessed data.
- **Cloud Hosting:** Vercel/AWS/Firebase for hosting and storage solutions.
- **Authentication:** OAuth 2.0 for secure user authentication and third-party integrations.

3.2 Reuse of Existing Software Components

To speed up development and enhance reliability, the project integrates third-party libraries and APIs:

- **Google Maps API** for interactive mapping.
- **Firebase Authentication** for secure and scalable authentication management.
- **Stripe API** (optional) for handling potential future transactions and donations.

3.3 Extensibility and Future Enhancements

The system is designed with future growth in mind. Potential enhancements include:

- AI-driven job recommendations.
- A chatbot for guiding users through available resources.
- Mobile application integration with React Native.

3.5 Hardware and Software Interface Paradigms

CityMeet runs as a **cloud-based web application**, accessible via modern web browsers. APIs facilitate communication between frontend and backend services.

3.6 Error Detection and Recovery

- API Gateway manages request validation and error handling.
- Logging and monitoring via tools like Datadog or Sentry.
- Auto-recovery mechanisms through cloud-based server monitoring.

3.7 Memory Management and Data Storage

- PostgreSQL for persistent data storage.
- Redis for caching frequently accessed queries to enhance performance.
- Object storage (AWS S3 or Firebase Storage) for user-uploaded content.

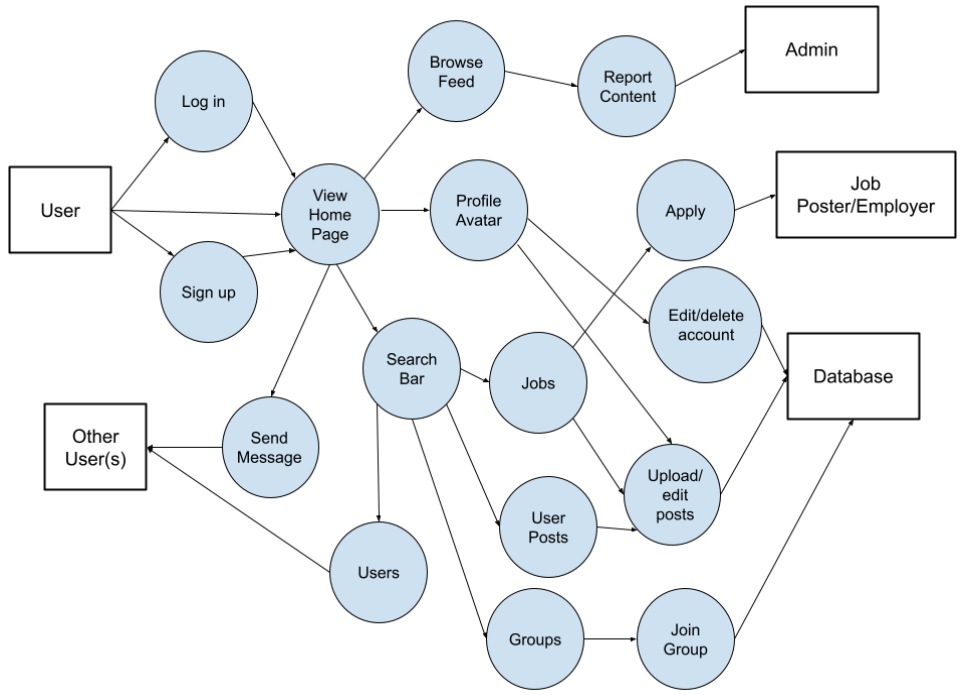
4. System Architecture

4.1 Logical View

Here, we list some class diagrams that would represent the software components that would be used to develop this product.

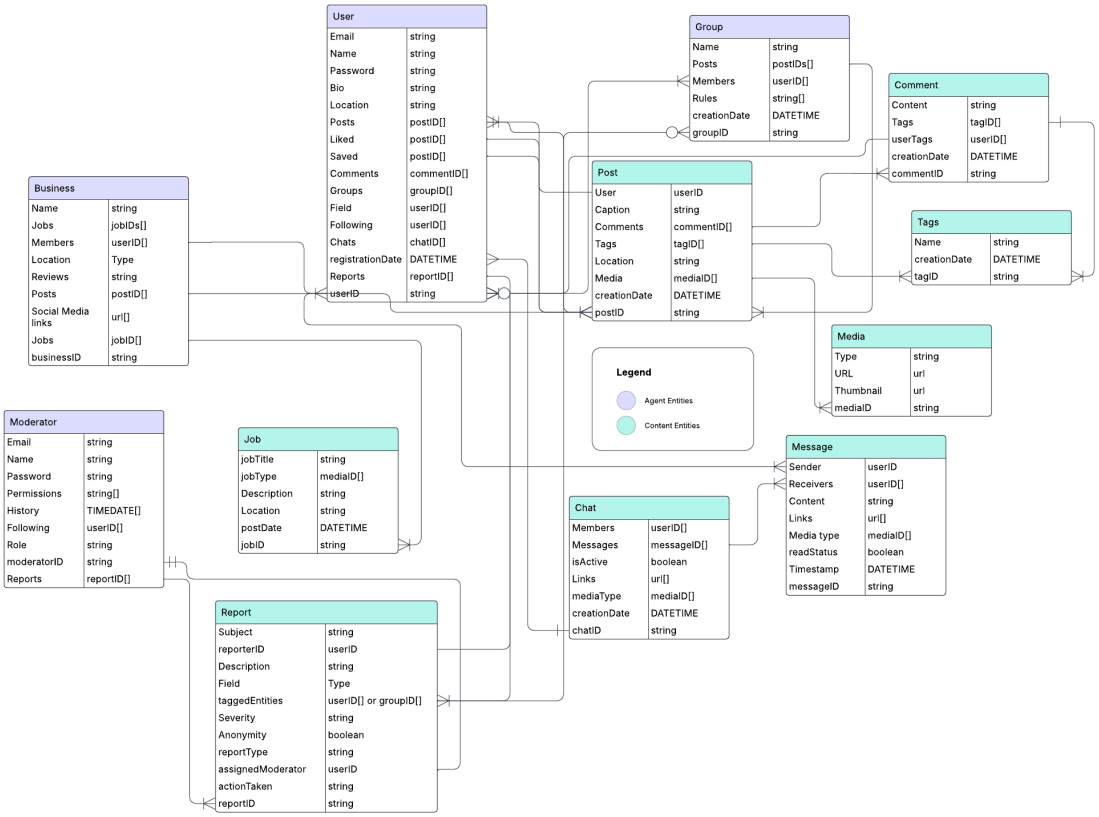
4.2 Development View

Level 0 Data Flow Diagram

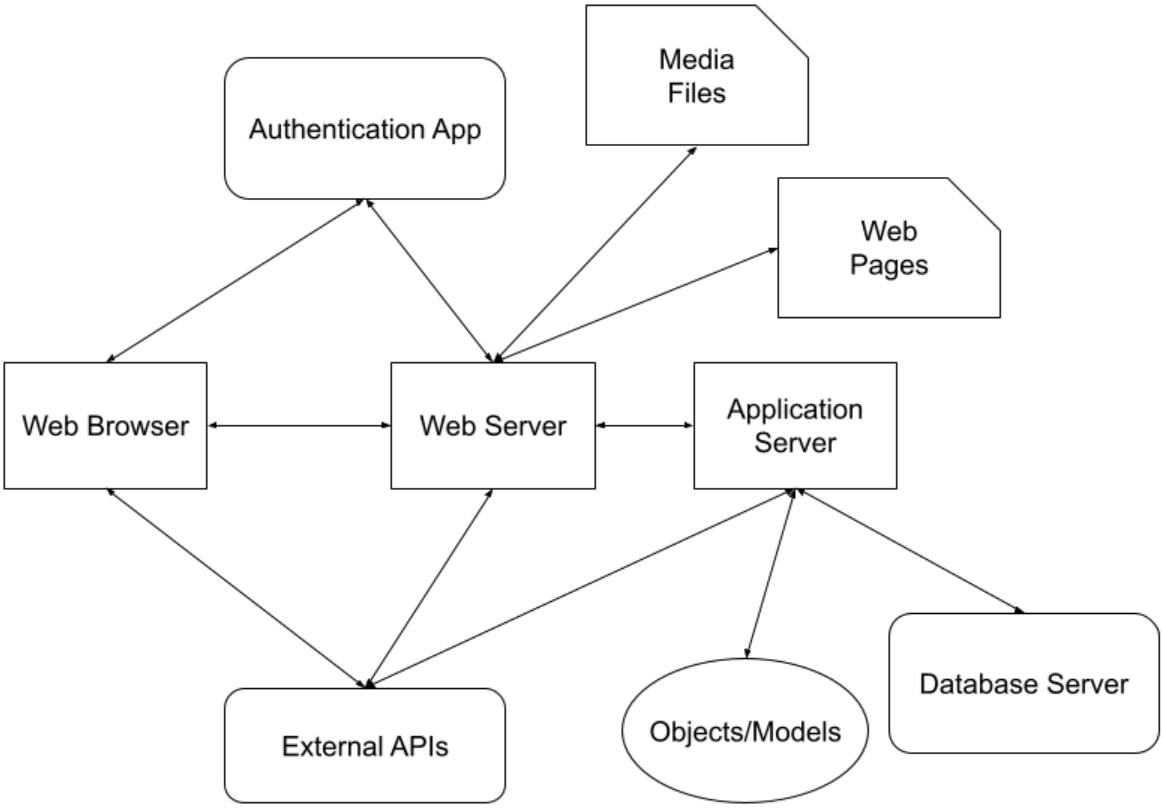


4.3 Process View

Database Entity-Relationship Diagram



4.4 Physical view



5. Policies and Tactics

5.1 Specific Products Used

- **IDE:** Visual Studio Code (VS Code) shall be the base Integrated Development Environment since it supports JavaScript, TypeScript, and Node.js which are key technologies in our stack.
- **Database:** PostgreSQL will be used as the primary database management system. It has been chosen for its high reliability and scalability, coupled with the provision of complex query support. Supabase, an environment for backend-as-a-service founded on PostgreSQL, will manage the database, authentication, and real-time features.
- **Libraries/Frameworks:**
    - Node.js and Express.js will handle server-side logic and API integration. They are chosen based on their capabilities in handling asynchronous operations and scalability.
    - Geospatial Integration: Google Maps API will be used for geospatial features like location tagging and map-based search.
- **Version Control:** GitHub will be used for version control and collaboration between the team members. It provides solid tools for code review, issue tracking, and continuous integration.

5.2 Requirements Traceability

To make sure that all the requirements are met and traceable throughout the development process, we will follow the following plan:

- **Traceability Matrix:** A traceability matrix will be created to trace each requirement from the Software Requirements Specification (SRS) to the corresponding design elements, test cases, and implementation details. The matrix will be updated and maintained throughout the project life cycle.
- **GitHub Issues:** All of the requirements shall be tracked with GitHub issues. The issues shall be tagged for specific tasks, pull requests, and test cases such that each of the requirements is addressed in development.
- **Code Reviews:** Through code reviews, the team members shall verify that the features included meet the laid-down requirements. Any differences will be recorded and corrected before code merging.

5.3 Testing the Software

Testing will be performed at multiple levels to ensure the quality and reliability of CityMeet:

- **Unit Testing:** The components and functionalities individually will be tested in isolation to ensure they are working as expected. Jest will be used for unit testing for both frontend and backend. Unit testing will be used to validate individual methods for account creation, posting, and group management.
- **Integration Testing:** Groups of components will be tested together to ensure they work with each other correctly. This will include testing API endpoints, database interaction, and frontend-backend communication.
- **System Testing:** The entire system will be end-to-end tested to ensure that all the system components are functioning in harmony with each other. It will comprise end-to-end testing of user flows such as account creation, posting, and group management.
- **Performance Testing:** The system will be load-tested to ensure that it performs well without any performance degradation when the number of users is high concurrently. Apache JMeter type of tools will be used for load testing.
- **User Acceptance Testing (UAT):** End-user team will run the application in a production environment to provide feedback on usability and functionality. This will help in ensuring that all issues not met during earlier testing phases are revealed.

5.4 Engineering Trade-offs

Trade-offs in Design and Implementation

- **Scalability and Complexity:** We have chosen microservices and modular architecture to make it scalable. This does increase the complexity of the system. We have proceeded with this trade-off to make sure that the system will be able to handle future growth.
- **Performance vs. Cost:** Using cloud infrastructure like Supabase and AWS assures high scalability and performance at an affordable cost. We chose to focus more on scalability and performance since they are most essential for user satisfaction.

- **Security vs. Usability:** Strong security features such as multi-factor authentication and encryption may add some friction to the user experience. We believe that the trade-off is worth making in order to protect user data and provide trust.

5.5 Guidelines and Conventions

Coding Guidelines and Conventions

- **Code Style:** We will apply the JavaScript Style Guide for frontend and backend code to ensure consistency across the codebase.
- **Naming Conventions:** camelCase for variables, functions, and classes, and UPPER_CASE for constants. Database tables and columns will be snake_case.
- **Documentation:** All code will be adequately documented with JSDoc for JavaScript/TypeScript and inline comments where necessary. API endpoints will be documented with Swagger/OpenAPI.
- **Version Control:** Git Flow will be used for version control, feature branches, pull requests, and code review before merging into the main branch.

5.6 Protocols

Protocols of Subsystems, Modules, or Subroutines

- **API Communication:** RESTful APIs will be utilized to communicate with JSON as the data format for all frontend-to-backend communication. HTTPS will be utilized for communication with security.
- **Authentication:** User authentication will be implemented using JSON Web Tokens and OAuth 2.0 for third-party login. Supabase Auth will handle user sessions and token generation.
- **Error Handling:** Uniform error messages with correct HTTP status codes will be returned from all API endpoints. Errors will be logged to assist debugging and monitoring.

5.7 Maintaining the Software

Plans for Maintaining the Software

- **Continuous Integration/Continuous Deployment (CI/CD):** We'll use GitHub Actions to automate build, testing, and deployment. This will assist in ensuring new code is deployed and tested quickly and reliably.
- **Monitoring and Logging:** We will use tools like Sentry for error tracking and monitoring. Logs will be stored in a central location for easy retrieval and analysis.
- **Regular Updates:** The software will be updated from time to time to fix security loopholes, remove bugs, and add new features. A release schedule will be adopted to enable timely updates.

5.8 Interfaces

Interfaces for End-Users, Software, Hardware, and Communications

- **User Interface:** The user interface will be user-friendly and accessible, following Material Design guidelines. It will be responsive and functional across devices.

- **API Interfaces:** The backend will expose APIs to interact with the frontend and third-party services using RESTful interfaces. API documentation will be published using Swagger/OpenAPI.
- **Hardware Interfaces:** The application will be coded to operate on typical consumer hardware (PCs, laptops, mobile phones) with no specific hardware requirements.
- **Communication Interfaces:** All client and server communication shall be through HTTPS. Push messages will be used for real-time mobile notifications.

5.9 System Deliverables

How to Construct and Develop the System's Deliverables

- **Build Process:** The build process will be automated through npm scripts for both the backend and frontend. Webpack will be utilized to build the front end, and Node.js will be used to build the backend.
- **Deployment:** Deployment shall be carried out via Docker containers, orchestrated via Kubernetes. This will allow seamless scaling and management of the application in the production environment.
- **Documentation:** All kinds of deliverables like source code, build guidelines, and deployment guidelines shall be documented and stored in the project repository.

5.10 Abstraction

Tactics for Abstraction

- **Database Abstraction:** We shall abstract database interaction using an ORM (Object-Relational Mapping) library like Sequelize. This will allow us to switch databases in the future with minimal code change.
- **API Abstraction:** The frontend will interact with the backend through a collection of abstracted API services. This will allow us to make changes to the backend implementation without affecting the frontend.
- **Component Abstraction:** We will use React components at the frontend as an abstraction of UI components. This will simplify the code and make it modular to manage.

6. Detailed System Design

6.1 Photos Page

6.1.1 Responsibilities

Provide a place to display, upload, and edit a user’s uploaded photos.

6.1.2 Constraints

Requires the usage of several custom components

6.1.3 Composition

The photos page consists of a grid where a user’s uploaded photos are displayed. There is also an upload photo button that lets users upload content from their machine when clicked.

6.1.4 Uses/Interactions

The user will use this component to view everything they’ve uploaded to the application. They can also upload photos, which should then pop up in the photo grid/display. Users can also mark photos as favorites or delete them from their account.

6.1.5 Resources

Photos are the main resource displayed on this page. Supabase is where all the uploaded photos are stored in the cloud. Memory is required to display the actual images on the web page.

6.1.6 Interface/Exports

This component provides a display for the user's content. Nav elements allow for page navigation. The sign-out button allows the user to safely log out of the application. There is also an upload photo button that lets users upload content from their machine when clicked.

6.2 Favorites Page

6.2.1 Responsibilities

Provide a place to display and edit a user’s photos that are marked as favorites.

6.2.2 Constraints

Requires the usage of several custom components

6.2.3 Composition

The photos page consists of a grid where a user’s uploaded photos are displayed.

6.2.4 Uses/Interactions

The user will use this component to view everything they've marked as favorite. They can also remove a photo from their favorites album or remove it from their account entirely.

6.2.5 Resources

Photos are the main resource displayed on this page. Supabase is where all the uploaded photos are stored in the cloud.Memory is required to display the actual images on the web page.

6.2.6 Interface/Exports

This component provides a display for the user's content. Nav elements allow for page navigation. The sign-out button allows the user to safely log out of the application.

6.3 Profile Page

6.3.1 Responsibilities

Display a user's account details, profile, and bio.

6.3.2 Constraints

Requires the usage of several custom components

6.3.3 Composition

On the left is the user's known friends and groups. The middle section displays the content that a user has uploaded and interacted with. On the right, we have the main user bio including their username, description, etc.

6.3.4 Uses/Interactions

The user will use this component to view everything their account. They may use it to change some settings/details about themselves. They can also quickly identify other users/groups to interact with. Lastly, they may manage the content that they have uploaded/interacted with.

6.3.5 Resources

Supabase is where all user's content is stored. Memory is required to display the actual images on the web page.

6.3.6 Interface/Exports

This component provides a display for the user's profile. Nav elements allow for page navigation. The sign-out button allows the user to safely log out of the application.

The set of services (classes, resources, data, types, constants, subroutines, and exceptions) that are provided by this component. The precise definition or declaration of each such element should be present, along with comments or annotations describing the

meanings of values, parameters, etc. For each service element described, include (or provide a reference) in its discussion a description of its important software component attributes (Classification, Definition, Responsibilities, Constraints, Composition, Uses, Resources, Processing, and Interface).

A description of any and all resources that are managed, affected, or needed by this entity. Resources are entities external to the design such as memory, processors, printers, databases, or a software library. This should include a discussion of any possible race conditions and/or deadlock situations, and how they might be resolved.

6.4 Map Page

6.3.1 Responsibilities

Display a user's choice of social service institutions .

6.3.2 Constraints

Requires the usage of several custom components

6.3.3 Composition

On the center the map is posted, with highlights different locations which reveals details about the location

6.3.4 Uses/Interactions

The user will use this component to find important links

wnt is stored. Memory is required to display the actual images on the web page.

6.5 Resources Page

6.1.1 Responsibilities

Provide a page for users to access helpful links as well as contacting .

6.1.2 Constraints

6.1.3 Composition

6.1.4 Uses/Interactions

6.1.5 Resources

6.1.6 Interface/Exports

7. Detailed Lower-Level Component Design

7.1 Nav

7.1.1 Classification

This component is a class, which is essentially a custom HTML navigation bar built with React.js.

7.1.2 Processing Narrative (PSPEC)

When users click on a tab in the Nav, they request a page to be displayed.

7.1.3 Interface Description

The Nav is made of interactive Link elements that switch out the main page's content depending on the element that was clicked.

7.1.4 Processing Detail

This component lets users navigate between different tabs of the CityMeet website

7.1.4.1 Design Class Hierarchy

This is a parent class.

7.1.4.2 Restrictions/Limitations

This class requires the use of React.js and Next.js.

7.1.4.3 Performance Issues

Issues are that this has a slow load time. When things are changed it does take time for things to update on the website.

7.1.4.4 Design Constraints

The necessary pages/file paths must be created first before they can be added to the component.

7.1.4.5 Processing Detail For Each Operation

Clicking on the links will trigger a GET request, so the server must load up the appropriate content when that happens.

7.2 Signout

7.1.1 Classification

This component is a button built with javascript

7.1.2 Processing

It'll handling signout out by ending the session with supabase and take them back into the sign in page

7.1.3 Interface Description

The signout button will be in the top right of the page

7.3 Authform

7.1.1 Classification

This component is a class, which is essentially a custom HTML form built with React.js.

7.1.2 Processing Narrative (PSPEC)

Users use this form to sign up or login into CityMeet.

7.1.3 Interface Description

The Nav is made of a form that switches between sign up/login if a specific button is clicked.

7.1.4 Processing Detail

This component takes in an email and password both both sign in and sign up actions.

7.1.4.1 Design Class Hierarchy

This is a parent class.

7.1.4.2 Restrictions/Limitations

This class requires the use of React.js and Next.js.

7.1.4.3 Performance Issues

Issues are that this has a slow load time. When things are changed it does take time for things to update on the website.

7.1.4.4 Design Constraints

This form doesn't really convey user errors/warnings in specific cases.

7.1.4.5 Processing Detail For Each Operation

Signing up and logging in would be a post request.

8. Database Design

This section will go into our database design, the tools we will be using, and the relationships between tables.

Database: Supabase/Postgresql

Supabase is the database we will be using for our CityMeet web app. Supabase is a PostgreSQL database. Supabase also contains a simple user interface that simplifies the usage of PostgreSQL without losing any core functionality PostgreSQL provides. We decided to use PostgreSQL because it allows us to manage large data sets and the relationships between the large data sets efficiently. This is especially important when considering the relationships between users and the accounts they may follow. Postgres also handles transactions really well and has better data integrity than most database management systems.

Tables and relationships:

In this section ,we will be looking into the tables we plan on establishing and the relationships between the tables**.**

Users:

Our user table will include data like Name, Username, Birthdate, and Age and may also include follower count, and the count of followers they have as well. This data will be shared with other users only if they are following each other.

Posts:

This table will include post interactions like, likes, comments, as well as post data like images, text, or audio, and location data.

Groups and Communities:

We will be storing data pertaining to groups or communities and then sharing it with people that request access to such communities.

Resume

Resume data will be stored and shared with companies that the user wants to share with.

Notifications:

Notification data like new followers, new posts, or new comments will be stored and pushed to the user. .

9. User Interface (AM)

9.1 Overview of User Interface

Describe the functionality of the system from the user's perspective. Explain how the user will be able to use your system to complete all the expected features and the feedback Information that will be displayed for the user. This is an overview of the UI and its use. The user manual will contain extensive detail about the actual use of the software.

- The system provides functionalities like login, data input, report generation, and feedback visualization.

Key Features:

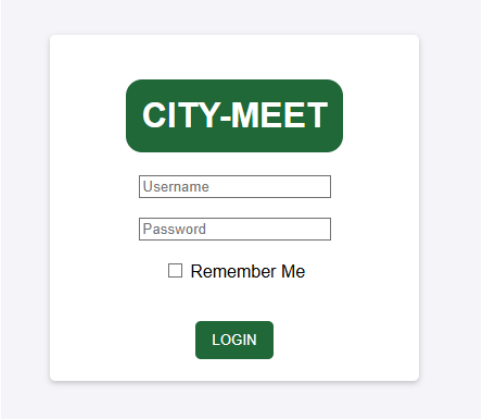
1. Home Page: Displays dashboard with navigation links.
2. Data Input: Allows user input for processing.
3. Reports Section: Provides data summaries or visuals.
4. Feedback Mechanism: Displays success/error messages after actions.

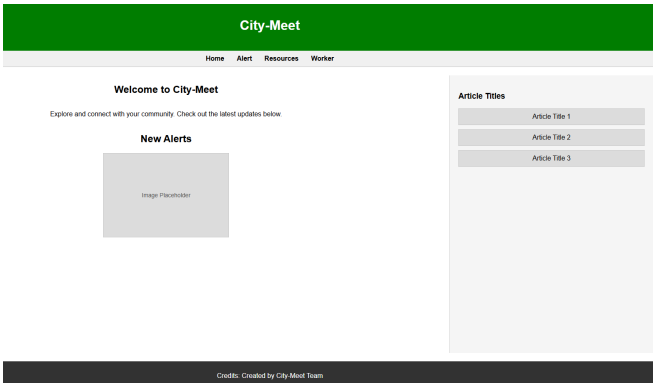
9.2 UX Standards

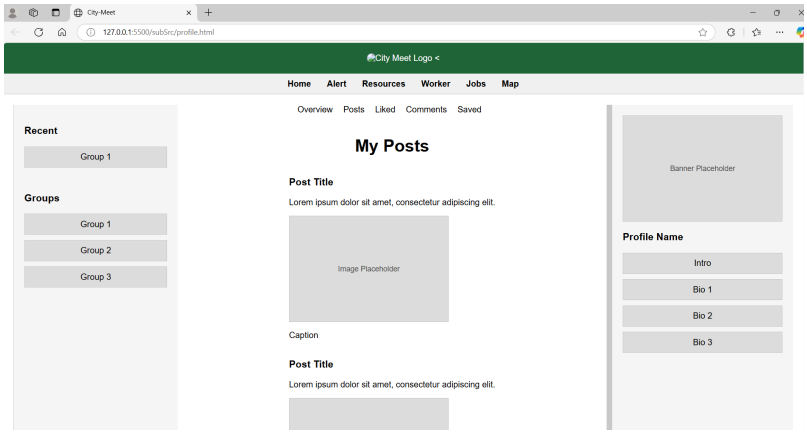
Consistency: Use of unified color schemes and layouts.
Responsiveness: Design for cross-device compatibility.
Clarity: Avoiding clutter by grouping related information and actions.

9.2 Screen Frameworks or Images

These can be mockups or actual screenshots of the various UI screens and popups.

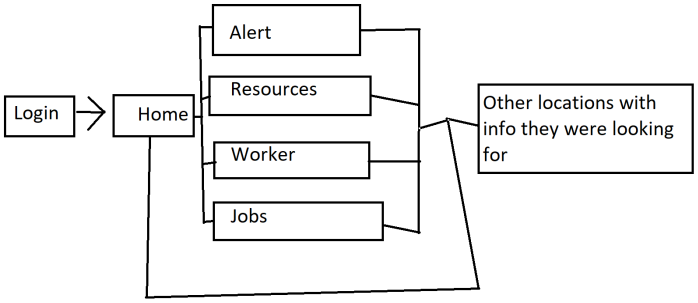






9.3 User Interface Flow Model

A discussion of screen objects and actions associated with those objects. This should include a flow diagram of the navigation between different pages.



10. Requirements Validation and Verification

Functional Requirements

Requirements	**Component Modules**	**Testing Methods**
Maps(4.1.1)	User Interface Module	Functional Testing Search and GeoLocation
Community Engagement(4.1.2)	User Interface Module	Unit Testing
Job Postings and Applications/ Welfare Locator (4.1.3)	User Interface Module	Unit Testing Integration Testing
User Authentication (4.1.4)	Data Holding Module	Unit Testing
Content Moderation and Reporting(4.1.5)	Request Handling	Unit Testing
Handling Abnormal Cases(4.1.6)	Processing/ Handling Module	Stress Testing Error Handling
Messaging/ Communications(4.1.7)	User Interface Module	Unit Testing
Two-Factor Authentication(4.1.8)	User Authentication Module	Functional Testing Unit Testing
User Accounts (4.1.9)	User Authentication Module	Functional Testing
User Posting/Content uploads(4.1.10)	User Interface Module	Unit Testing
Content Reporting and Flagging(4.1.11)	Processing/ Request Handling	Unit Testing Stress Handling

External Interface Requirements

Requirements	**Component Modules**	**Testing Methods**
User Authentication Interface	Data Holding Module	Functional Testing Integration Testing

Mapping API Interface	Data Holding Module	Functional Testing Integration Testing
Job Listing API Interface	Data Holding Module	Functional Testing Integration Testing
Email API Interface	Data Holding Module	Functional Testing Integration Testing
Database Interface	Data Holding Module	Functional Testing Integration Testing

Non-Functional Requirements

Requirements	**Component Modules**	**Testing Methods**
Performance Requirements(5.1)	Quality Control	Performance Testing
Safety Requirements(5.2)	Security Module	Security Testing Performance Testing
Security Requirements(5.3)	Security Module	Security Testing Performance Testing
Software Quality Attributes(5.4)	Quality Control	Performance Testing

11. Glossary

- **Software Architecture**- high level structure of a software system, defining how different components interact, their organization, and the principles behind design and development
- **MVC**- Model View Control Architectural Pattern

12. References

Gong, Elliot, et al. *CS 3337 Group 5 Project Proposals*. https://docs.google.com/presentation/d/1zL862LbHzwp00Qv5E4HOdjGxCMRTBaEpJW9zUCYLK-k.

- The CityMeet project's proposal document.

Gong, Elliot, et al. "CityMeet Software Test Plan." *Google Docs*, 21 Feb. 2025, https://docs.google.com/document/d/1rr_VO-qoN1aIkb05d-1kpxe2-Py6Dr9wahwB9l4Xbvs/edit?usp=sharing&usp=embed_facebook. - CityMeet's software test plan document.

Gong, Elliot, et al. *CityMeet Software Requirements Document Group 5*. Google, 24 Feb. 2025, https://docs.google.com/document/d/1w4Hws-BBQ0I96lDGaJopdF6AAx60qzuodhh7nCwixkM/edit?usp=sharing&usp=embed_facebook. - CityMeet's software requirements document.

N/A. *Department of Public Social Services*. https://dpss.lacounty.gov/en.html. Accessed 10 Mar. 2025.

- The official website of the Los Angeles County Department of Public Social Services

Node.Js — Run JavaScript Everywhere. https://nodejs.org/en. Accessed 14 Mar. 2025. - The official website for the Node.js javascript library.