SW Engineering
CSC648-848-05 Spring 2025

Serial

Team 02

- Victoria Barnett rbarnett@sfsu.edu Team Lead
- Tushin Kulshreshtha tkulshreshtha@sfsu.edu Back End Lead
- Ansh Patel apatel18@sfsu.edu Front End Lead
- Pritham Sandhu psandhu3@sfsu.edu Software Architect
- Alison John ajohn3@sfsu.edu Database Administrator
- Nidhey Patel npatel20@sfsu.edu Technical writer
- Yikang Xu yxu26@sfsu.edu Github Master

URL: https://serialpm.tech

Milestone - 5

05/08/2025

History Table

Milestone	**Version**	**Date**
Milestone 5	No version	05-18-2025
Milestone 4	Version 1	05-08-2025
Milestone 3	Version 2	05-01-2025
Milestone 3	Version 1	04-17-2025
Milestone 2	Version 2	04-13-2025
Milestone 2	Version 1	03-20-2025
Milestone 1	Version 2	03-06-2025
Milestone 1	Version 1	02-20-2025

Table Of Contents

2. Product Summary:

We are seeking backers for Serial, a new piece of project management software being developed by Team 02. Our goal is to create a uniquely visual-first project management framework that allows technical and administrative workers to track their tasks and progress side by side in one application and thus, one enterprise payment plan. We believe our software can vastly simplify the way product managers and developers keep track of their progress in a revolutionary new way, collapsing all project management abstractions into one visual project timeline that keeps everyone on task, focused, and working with the same information. [product name] enables new focus to drive team efficiency, it will enable smaller, more agile teams to manage their project from humble beginnings all the way to an IPO in a more coordinated way by collapsing the confusing jumble of Product and Project management tools, Kanban boards, and Developer task tracking into one chronological-first organization method, and then filter the information as needed for whichever team is looking at it.

SerialPMs most unique feature is our visual timeline for work – rather than a Kanban board or sprints, you see tasks like a calendar along a timeline of work, perfect for elevating deadlines and reducing confusion around priorities. Our next unique feature is we aren't attempting to corner one segment of a product's specialists, SerialPM is interested in becoming the whole of a businesses project management structure – for it's developers, for it's administrators, for it's project managers, and for it's clients. This flexibility for role allows us to eat into the space of what may be 2-3 competitors being paid for by a business, their organizational calendar, and some internal tooling. All of this combines into an incredible value proposition for a business interested in staying thin.

3. Milestone Documents:

SW Engineering

CSC648-848-05 Spring 2025

Serial

Team 02

- Victoria Barnett rbarnett@sfsu.edu Team Lead

- Tushin Kulshreshtha tkulshreshtha@sfsu.edu Back End Lead
- Ansh Patel apatel18@sfsu.edu Front End Lead
- Pritham Sandhu psandhu3@sfsu.edu Software Architect
- Alison John ajohn3@sfsu.edu Database Administrator
- Nidhey Patel npatel20@sfsu.edu Technical writer
- Yikang Xu yxu26@sfsu.edu Github Master

Milestone 1

02/20/2025

3. Executive Summary

We are seeking backers for Serial, a new piece of project management software being developed by Team 02. Our goal is to create a uniquely visual-first project management framework that allows technical and administrative workers to track their tasks and progress side by side in one application and thus, one enterprise payment plan. We believe our software can vastly simplify the way product managers and developers keep track of their progress in a revolutionary new way, collapsing all project management abstractions into one visual project timeline that keeps everyone on task, focused, and working with the same information. [product name] enables new focus to drive team efficiency, it will enable smaller, more agile teams to manage their project from humble beginnings all the way to an IPO in a more coordinated way by collapsing the confusing jumble of Product and Project management tools, Kanban boards, and Developer task tracking into one chronological-first organization method, and then filter the information as needed for whichever team is looking at it.

This application will provide a visual project management timeline, collapsing different levels of project abstraction from organization-wide initiatives down to individual team tasks. It aims to consolidate timelines, due dates, sprints and sprint points, organizational considerations, and the specific needs of product managers, development teams, and QA teams, all within a single, opinionated, chronological-first Project management framework. The app will offer multiple views of the same underlying project data, allowing for dynamic updates across teams. For example, a development task due on the 20th will automatically generate a corresponding review task for the product manager due on the 22nd, and vice versa with a new feature request. Potential features include calendar synchronization (possibly via .ics file generation), a team meeting scheduler, task generation and assignment, and GitHub integration for developers. Importantly, the application will support various project management

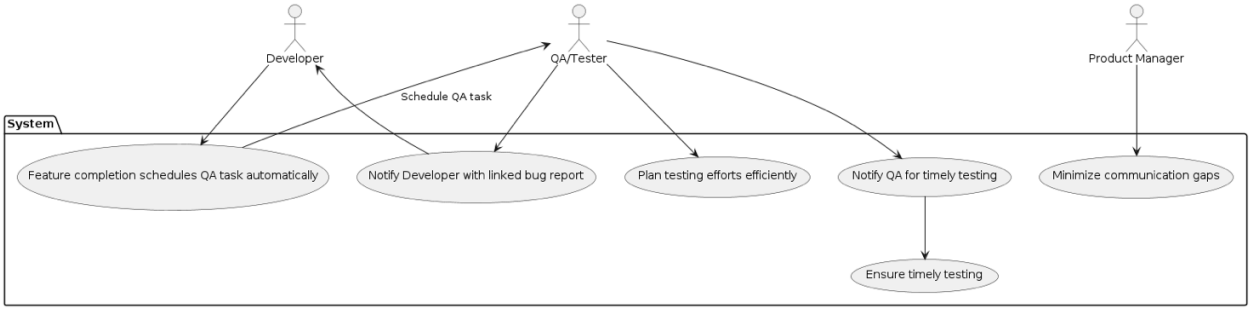
methodologies, including Kanban, Agile, and sprint, all for a single price. This application offers several key benefits to users.

The value of Serial is that it's a cost-effective alternative to other project management software, providing a robust feature set without a premium price tag. By streamlining workflows and keeping tasks clearly defined, it boosts efficiency and helps users stay on track. The visual nature of the timeline-based approach caters to users who benefit from a strong visual separation of tasks, eliminating the need to constantly recall priorities. Furthermore, the timeline focus actively combats "backlog" clutter by requiring due dates for all assigned tasks, ensuring consistent reminders and preventing items from being forgotten. The unique design of this application centers around its visual-first approach, prioritizing a chronological project view that provides a clear and intuitive understanding of project progression. Another key differentiator is its straightforward pricing model. All features are available for a single price, eliminating the confusion and limitations of tiered subscriptions for personal or enterprise use. This accessible pricing structure makes the application particularly well-suited for smaller teams.

4. Use Cases

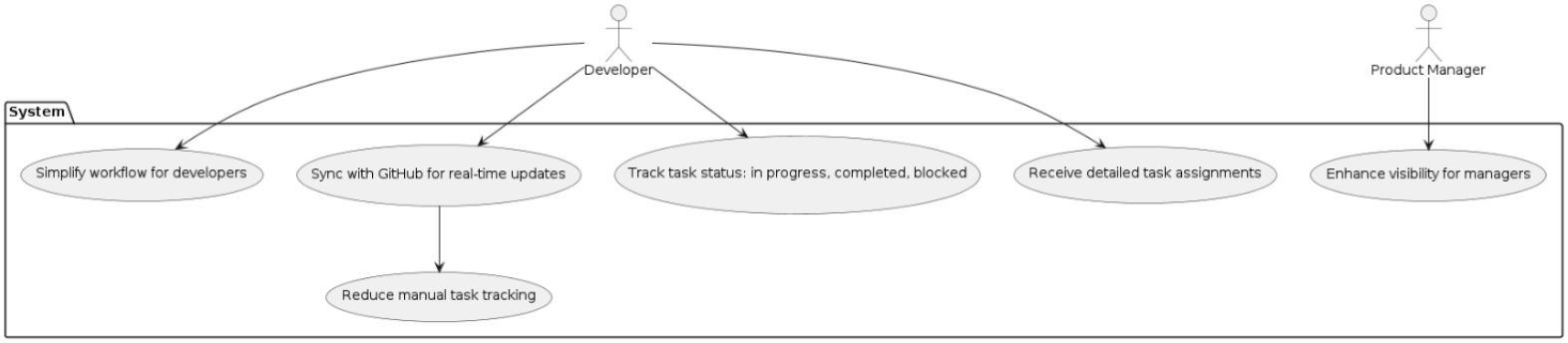
1. Create and Manage a Project Timeline

- **Actors**: Developers, Product Managers
- **Assumptions**: Developers, PrMs, PdMs need an easy way to track tasks & deadlines without switching between multiple platforms.
- **Use Case**: The project timeline provides teams with detailed task assignments, which they designate as in progress, completed, or blocked as they work. The system automatically syncs with GitHub to guarantee that merges and commits match actual task revisions. This interface reduces administrative overhead so that developers may concentrate on their work and gives product managers an open picture of continuous development initiatives.
- **Benefits**:
    - Reduces manual task tracking
    - Enhances visibility for managers
    - Simplifies workflow for developers.



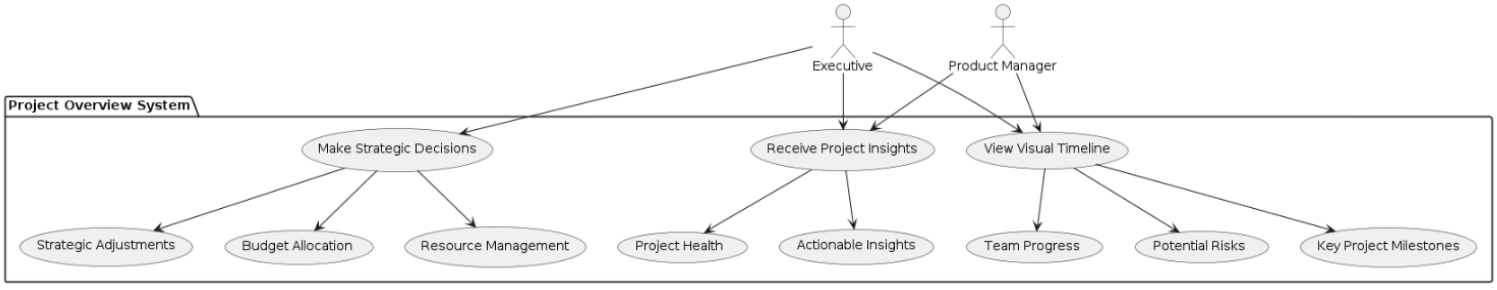
2. Automated Cross team Task Synchronization

- **Actors:** All, key interoperability for all actors
- **Assumptions**: Developers and QA teams work in separate cycles but need seamless handoffs.
- **Use Case**:In order to guarantee timely review without any human interaction, the system automatically schedules a QA assignment within a customizable time-window of a developer completing a feature. Notifications sent to the QA team help them to effectively coordinate testing initiatives. Should a problem arise, the developer is informed via a connected bug report, therefore fostering smooth cooperation and lowering project delays.
- **Benefits**:
    - Improves workflow efficiency
    - Ensures timely testing
    - Minimizes communication gaps.



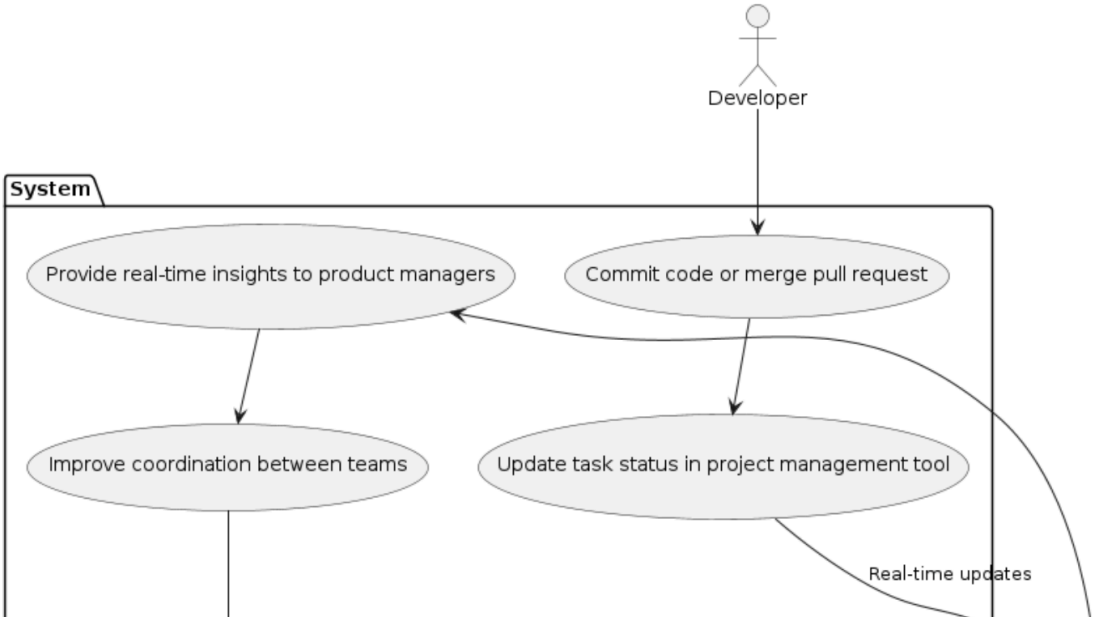
3. Visual Project Overview for Executives

- **Actors**: Executives, Project Managers, Team Leads
- **Assumptions**: Executives require high-level overviews rather than granular task details.
- **Use Case**: Executives have access to a visual timeline that is concise and displays key project milestones, team progress, and potential risks – a loading bar that captures all finer detail in the effort. Rather than delving into minutiae, they acquire practical understanding of the general state of the project, which enables quick decisions about resource management, budget allocation, and top-level strategy changes.
- **Benefits**:
    - Offers quick decision-making insights
    - Enhances resource allocation
    - Improves risk management.



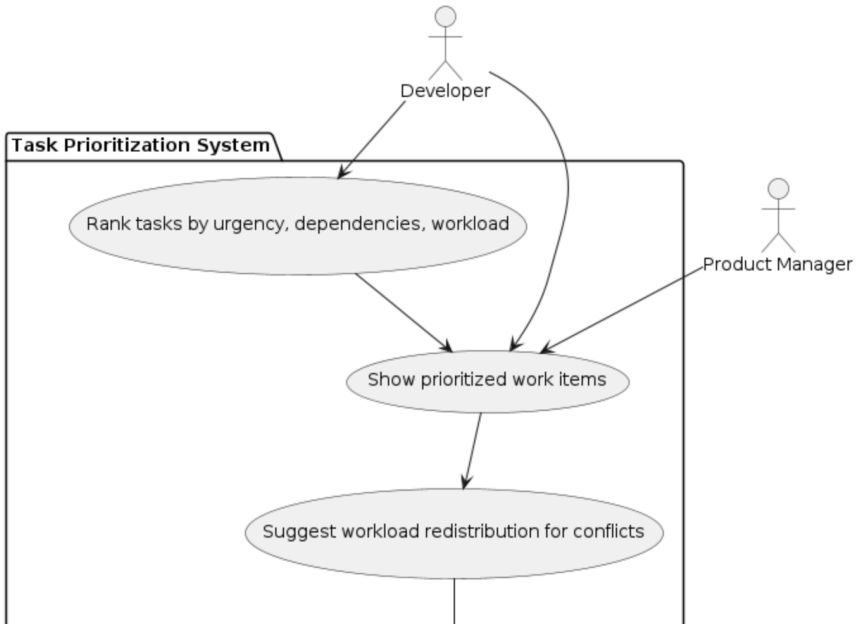
4. GitHub Integration for Real-Time Progress Updates

- **Actors**: Developers, Product Managers
- **Assumptions**: Developers use GitHub for code management.
- **Use Case**: The system changes the project management user interface when a developer commits code, submits a pull request, or when a pull request is approved. Product managers can get real-time information without having to check GitHub by hand. This makes it easier for the development and management teams to work together and keeps project tracking fully automated.
- **Benefits**:
    - Reduces manual tracking efforts
    - Provides real-time insights
    - Enhances collaboration



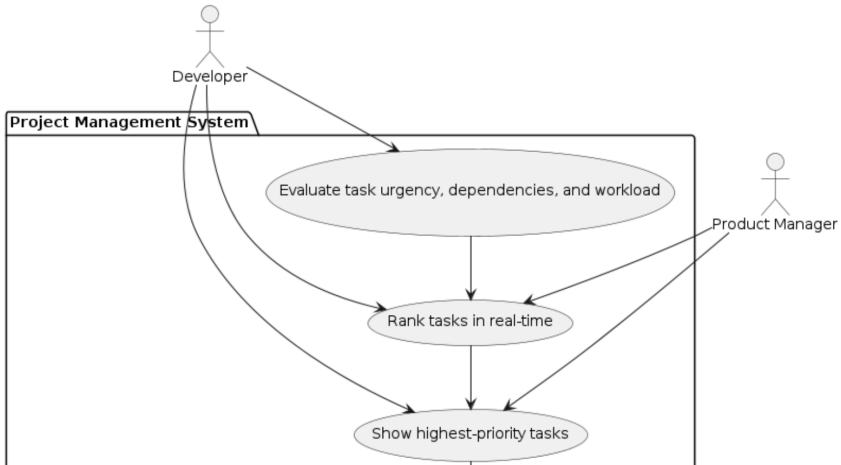
5. Task Prioritization Based on Deadlines & Dependencies

- **Actors**: Developers, Product Managers
- **Assumptions**: Projects have multiple dependencies that must be handled efficiently.
- **Use Case**: Task urgency, dependencies, and team workload all help the algorithm automatically rank activities. Clear prioritising of work items by developers and management enables them to concentrate on the most important. Should competing deadlines develop, the system recommends task reassignment to avoid congestion and guarantee consistent performance – Serial is opinionated about time, forcing you to make choices to avoid deadlock.
- **Benefits**:
    - Ensures efficient task execution
    - Prevents missed deadlines
    - Optimizes workload management.



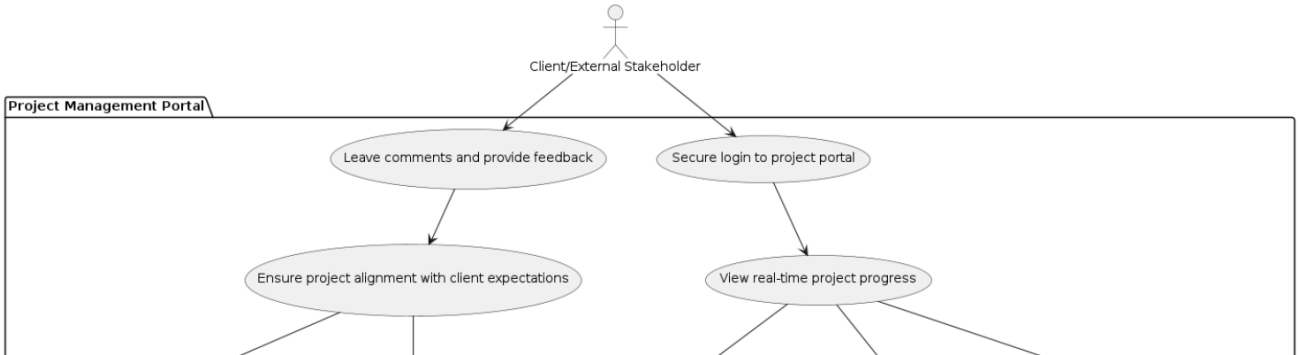
6. Schedule Meetings and Manage Calendar

- **Actors:** Developers, Product Managers
- **Assumptions:** Projects have multiple individuals who must sometimes meet regarding their tasks.
- **Use Case:**The project management software automatically evaluates task urgency, project dependencies, and current team workload using an intelligent algorithm. This system ranks activities, ensuring that both developers and product managers focus on the highest-priority tasks first, and facilitates easy ways for them to meet if needed. If competing deadlines arise, the app proactively suggests task reassignment or deadline adjustments to prevent bottlenecks, ensuring smooth and continuous progress. This dynamic prioritization helps teams manage their workload effectively and ensures critical tasks are completed on time.
- **Benefits:**
    - Ensures efficient task execution by always focusing on the most critical work items.
    - Prevents missed deadlines by identifying and resolving scheduling conflicts early.
    - Optimizes workload management across the team, reducing burnout and improving productivity.



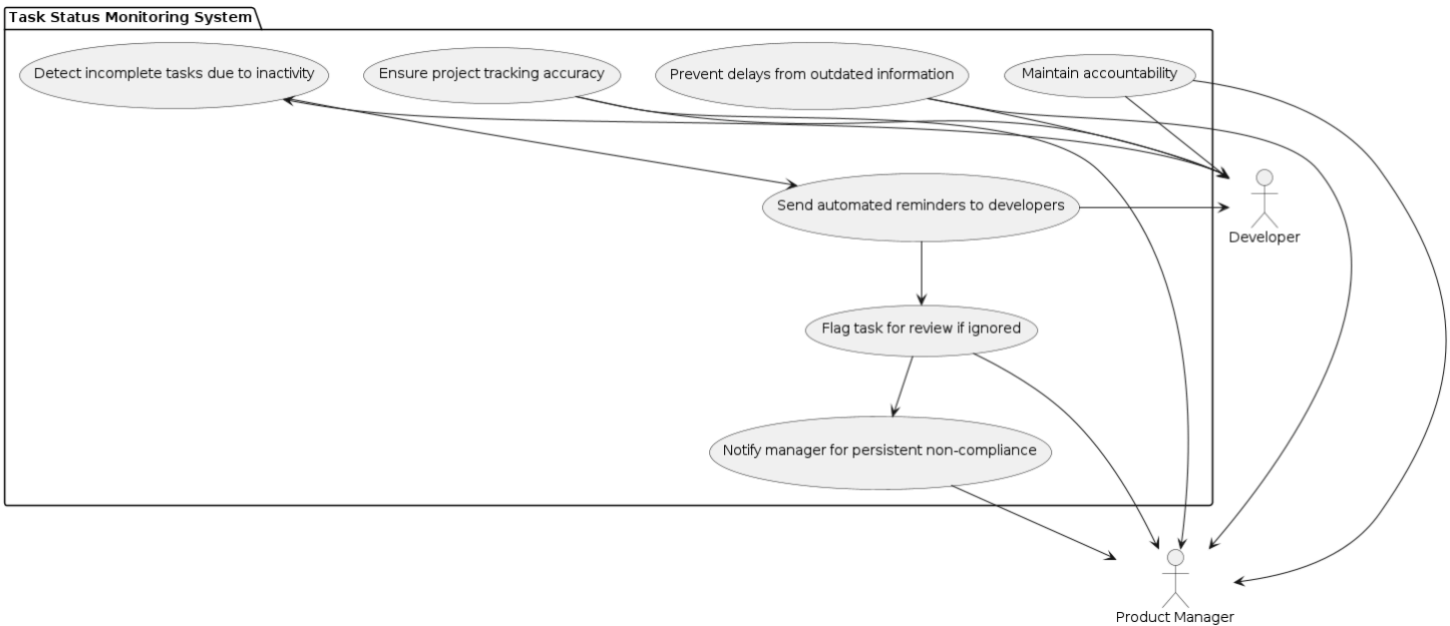
7. Review Project Progress and Provide Feedback

- **Actor:** Client/rnal Stakeholder
- **Assumptions:** Clients need real-time access to project status, upcoming deliverables, and a direct channel for feedback without constant back-and-forth communication.
- **Use Case:**
  Clients or outside stakeholders can access the project management app through a secure portal and see the real-time progress of the project, keep track of future deliverables, and look over goals that have already been met. Including finished activities, ongoing work, and any possible delays, the app offers a clear, graphic chronology of project status. Clients can immediately offer comments, questions, and feedback on certain aspects of the project, all configurable by the team. This direct feedback loop guarantees that the project remains in line with client expectations and helps to lower the necessity of regular status meetings and long-standing email exchanges.
- **Benefits:**
  - Enhances transparency by giving clients real-time project visibility.
  - Reduces the need for constant status updates through emails or meetings.
  - Facilitates quicker feedback, enabling teams to address client needs promptly.
  - Strengthens client relationships by keeping them informed and involved throughout the project lifecycle.



8. User Not Following App Protocol

- **Actors**: Developers, Product Managers
- **Assumptions**: Users sometimes neglect to update task status, causing workflow issues.
- **Use Case**: Confusion in project tracking results from a developer forgetting to mark a finished task as completed. The system notes inactivity and generates an automatic reminder. If not addressed, the work will be marked for review. Constant non-compliance forces a manager to be notified, therefore ensuring the problem is resolved. If an item falls off entirely, it will be flagged for later review: maybe this is a task your dont need to generate anymore?
- **Benefits**:
    - Ensures project tracking accuracy
    - Prevents delays due to outdated information
    - Maintains accountability.



5. Data items and Entities

Main Terms:

- **Project**: The active timeline Serial is displaying, with all associated tasks and other features. Projects group related tasks and are used for managing work that requires multiple steps or milestones

- **Task**: The most basic unit in Serial, an item that has at minimum a description, and a date attached – Tasks are the core unit of work in the software and can be assigned to users.

- Organization: The largest possible unit in Serial, the group of all users. Organizations create projects, and teams.

- **Teams**: Smaller sections of Organizations. Teams can attach to either projects or organizations depending on their scope. Teams create, are assigned, and complete tasks, Teams are assigned to specific projects, and team members have access to tasks within those projects.

- **Timeline**: A chronological “loading bar” of your active project, the default project view in Serial.

- **Chunk**: A section of the loading bar, typically delineating a release window, review deadline, or some other important delineation between work ending and passing to another team. Significant points or stages in a project that represent the completion of key tasks or deliverables, Chunks are used to track major project achievements and ensure alignment with deadlines.

- **Views**: A set of user interface filters that diverges from the timeline, reconstituting project data in different forms as needed, for different team management strategies or inquiries.
- Privileges: Permissibility to use specific feature sets, configurable by role or custom roles can be created.

- **User**: Represents an individual who interacts with the software. The user could be a manager, or a developer, or a custom role: each with different privileges and roles.

- **Role**: predefined sets of privileges for ease of using Serial the first time.

- **Developer**: A user role that, by default, has view options for VCS enabled.

- **Manager**: A user role that, by default, has granular task progress views and can assign to teams.

- **Client**: A user that experiences Client views of the project via a guest account at minimum.

- **Comment**: Allows users to communicate about a task, ask questions, or provide updates. Users can add, edit, and delete comments based on their privileges. Comments are displayed in a task's activity feed.

- **Privileges**:
    - Various feature sets in Serial can be protected behind customizable Roles, which are assigned the feature sets as privileges. Some privileges are:
        - Reassigning tasks
        - Creating teams
        - Editing Roles
        - Editing task deadlines
        - Creating Projects and editing their parameters

6. Functional Requirements

1. User and Access Management Functionalities

1.1 Users can register and log in using a username and password.

1.2 Users can reset their password via email verification.

1.3 Users can invite new team members to projects, be promoted to Admins, and manage access for other users.

1.4 Users can update their profile information.

1.5 User activity, such as task completion and project edits, will be logged.

2. Organization-Level Functionalities

2.1 Organizations can create and manage multiple projects within a single platform.

2.2 Organizations will have role-based access control, with permissions defined for administrators, product managers, developers, and QA teams.

2.3 Organization's users may view an enterprise-wide project timeline that visualizes all active projects.

2.4 Organizations can align their goals and deadlines through organization-wide sprint planning.

2.5 Organizations will have real-time data synchronization across all projects and all views of projects.

3. Project Management Functionalities

3.1 Product managers can create, update, and track project roadmaps.

3.2 Projects can be visualized as timelines to track tasks, deadlines, and dependencies.

3.3 Project managers can define sprint cycles and assign sprint points.

3.4 Project Task dependencies will be automatically generated(e.g., a development task due on the 20th triggers a review task due on the 22nd).

3.5 Projects will have filters to switch between different views of the same project data (e.g., development view, product manager view, QA view).

4. Task Management Functionalities

4.1 Tasks can be created, assigned, and updated.

4.2 Tasks can have priority levels (e.g., high, medium, low).

4.3 Tasks can be categorized by department (e.g., Development, QA, Product).

4.4 Tasks can have due dates and automatic archive dates to prevent backlog clutter.

4.5 Tasks can have assigned users who receive notifications about approaching task deadlines via email or in-app alerts.

5. Workflow Automation Functionalities

5.1 Automated workflows are available for users to streamline task assignments and approvals.

5.2 Automatic task assignments based on predefined rules and conditions (eg, QA admin creates a task -> goes to QA team, creates a feedback meeting with development.

5.3 Automated notifications for task updates, approaching deadlines, and project changes.

5.4 Workflow automation available for task dependencies to ensure that certain tasks cannot begin until prerequisites are met.

5.5 Automation for recurring tasks and reminders for routine project activities.

6. Calendar and Scheduling Functionalities

6.1 Shared project calendar for tracking team meetings and deadlines.

6.2 Calendar synchronization will be supported with external applications (Google Calendar, Outlook) via .ics file generation/import.

6.3 Scheduling is available for PrM/PdMs to assign team meetings directly from the application.

6.4 The calendar shall automatically generate meeting reminders for all invited participants.

6.5 Calendar tooling shall enable users to propose, vote on, accept, and decline meeting times to improve scheduling efficiency.

7. Client and Stakeholder Collaboration Functionalities

7.1 External clients and stakeholders will have custom views to view project progress with controlled access.

7.2 Clients shall view specific milestones and tasks, as defined by organization teams making them available.

7.3 Clients will be able to share feedback directly within Serial, by comments and flags on their view..

7.4 Client views shall have functionality to be real-time updated, or show a client only “completed” Blocks.

7.5 Clients shall be able to real-time communicate with Team leads and Organization managers.

8. Collaboration and Communication Functionalities

8.1 Commenting on tasks and projects shall be supported.

8.2 @Mentions shall notify specific team members.

8.3 Collaboration on tasks via file attachments shall be supported.

8.4 Internal communications by a team chat system shall be provided for discussions.

8.5 Project activity history shall be available to track changes and progress.

9. Reporting and Analytics Functionalities

9.1 Real-time project progress reports shall be generated.

9.2 Workload distribution across team members shall be viewable by managers.

9.3 Sprint burndown charts for Agile projects shall be provided.

9.4 Report exporting in PDF and CSV formats shall be supported.

9.5 Historical project analytics shall be available to track past performance.

10. Customization and User Preferences Functionalities

10.1 Dashboard layout customization shall be supported.

10.2 Light mode and dark mode themes shall be available.

10.3 Notification preferences shall be configurable.

10.4 Custom project template creation shall be supported.

10.5 Task color label selection shall be available.

7. Non Functional Requirements

Performance:

- The entire web application response time should be <5 seconds
- Client-side web page render time should be <3 seconds
- UI/UX element changes should be synchronized <2 seconds
- Changes to files >10MB should be synchronized <5 seconds
- Database queries should be <3 seconds
- The provided API response time should be <5 seconds
- The web application can handle 100 concurrent users

Reliability:

- The web application can handle 100 users and 25 Projects without degradation
- The web application can ensure data consistency during data transactions
- The web application can handle concurrent data editions
- Partial functionality without internet connection
- The system shall cache on the user's machine and send timely updates for safe backups of work

Security:

- AES-256, BCrypt, or equivalent encryptions for servers and databases
- Audit logs to see history of access and setting changes
- API key rotation or use authorization protocol
- Encrypted user information
- User password specifications
- User permissions and privileges
- Two-Factor Authentication
- HTTPS connection
- Bot detection/CAPTCHA to limit DDOS exposure
- Rate limits to limit damage in case of compromise

Usability:

- Intuitive and concise UI
- Responsive design that works on many sizes of workspace
- UI/UX elements should have a latency within 5 seconds
- Undo/Redo/Version control for users
- Light/Dark/Custom themes for readability in many environments
- Integration with other services for collaboration and adoptability
- User guide is provided to explain more complex features
- Multi-input support
- Multi-media support
- Accessibility/WCAG

Maintainability:

- Comprehensive documentation so developers can pick up where another left off
- Version control for developers, connecting directly to branches of different tasks
- Modular features and microservices to reduce application complexity
- Tests for new functionalities

Portability:

- Cross-platform support - ie, as many browsers as possible
- Compatibility with multiple APIs
- Containerized development

Scalability:

- Vertical scaling with optimization
- Potential for horizontal scaling with additional servers
- Load balancing

Compatibility:

- Cross-Browser/OS/Device support
- Localization
- Backward compatibility

Compliance:

- GDPR/CCPA and other personal data collection consents
- NIST Cybersecurity Framework
- WCAG for accessibility

Supportability:

- Bug track and issues list
- Regular updates/patches
- User reviews
- Customer Service

Efficiency:

- Client-side memory usage should be <4GB
- Server-side memory usage should be <16GB

Coding Standards:

- Using GitHub for version control
- Master and Development branches
- Feature branches & PRs
- Modular and repeatable code
- Input Validation
- Algorithm optimization
- Readability
- Error Handling
- Comments and technical Documentation

Environmental Sustainability:

- Reduce power and bandwidth consumption – smallest resource footprint possible
- Backward compatibility
- Optimize the code to increase efficiency

8. Competitive Analysis

Competitor's Table

Competitor	Strengths	Weaknesses	Pricing	Social Media Focus	Onboarding Experience
ClickUp	Tons of features, super customizable, works with other tools	Too many options can be confusing, takes time to learn	Free plan, paid starts at $7/user/month	Active on YouTube, Twitter, and LinkedIn	Step-by-step tutorials, but kinda overwhelming
Asana	Clean UI, easy task management, good for teamwork	Not great for tracking time, limited features on free plan	Free version, paid starts at **$10.99/user/month**	Engages a lot on LinkedIn and Twitter	Simple setup, has in-app guides
Trello	Super easy to use, great for visual organization	Not ideal for complex projects, weak reporting tools	Free plan, paid starts at **$5/user/month**	Big on Instagram, Twitter, and YouTube	Quick start, drag-and-drop style
Notion	Combines docs, tasks, and databases all in one	Not great for Agile teams, missing advanced project views	Free personal use, paid starts at **$8/user/month**	Very active on Twitter, YouTube, and TikTok	Flexible, but setup takes effort
Airtable	Clean UI with data visualization; Automation scripting available; Mobile support; API available; AI integration	Limited functions for free plan; Can only import from spreadsheet apps; No integration with external apps; Too many options	Free plan with 5 users; Paid start at **$20/user/month**	Active on LinkedIn and X; Presence on Facebook, Instagram and Youtube	Intuitive but may be overwhelmingly detailed; Spreadsheet-like experience

Competitive features table

Feature	ClickUp	Asana	Trello	Notion	Airtable	Serial
Automated cross-team dependancy	-	-	-	-	-	+
Customizable Organization Setups	+	Limited	Limited	Limited	+	+
Chronological First Visual Interactive UX	-	-	-	-	-	+
Automations	+	+	-	-	+	+
Self-hosting/ Self-modification	-	-	-	-	-	+

Summary

Our competitive edge will be both in our featureset and our planned pricing model. Presently, none of the competitors do everything: some like Notion and Trello have imposed collaboration frameworks for a stronger built-in experience, and some like Airtable rely on heavy integration on a paid plan to make full use of the application. Serial will be flexible where it matters: organizationally, visually, and with integrations for both free and paid users. However, it will have a solid backbone of an imposed framework where needed: in its chronological-first basis, tasks will need concrete dates attached to then flexibly fit in whatever views your organization uses. By distilling the primary unit of information all Project Management software uses into its most basic form, Serial will be able to rebuild it in a customizable way as needed by the user, allowing us to be more flexible than the competition.

9. Checklist

Task	ISSUE	ON TRACK	DONE
The Team has found a time slot to meet outside of class			✅
GitHub Master has been chosen			✅
The team has collectively decided on and agreed to use the listed software tools and deployment server.			✅
The team is ready to use the chosen front-end and back-end frameworks, and those who need to learn are actively working on it.			✅
The Team Lead has ensured that all members have read and understand the final M1 before submission.			✅
GitHub is organized as discussed in class (e.g., master branch, development branch, folder for milestone documents, etc.).			✅

10. High-Level System Architecture & Technologies

Primary Technologies:

- Cloud server host: AWS t2micro (1vcpu, 1gb RAM)
- OS: Ubuntu Server 24.04 LTS
- Database: MySQL 9.2
- Web Server: Express (4.21.2 unless we see an issue, then 5.0.1)
- Backend Languages, frameworks: JavaScript ES2023, Node 22.13.1
- Frontend Languages, frameworks: JavaScript ES2023, React 19.0

Other important technologies or packages:

- anime.js
- NPM
- GSAP
- Caddy
- Namecheap via GitHub Student for free domain & SSL
- Tailwind CSS
- Vite
- Docker: Docker Container on EC2

11. List Of Team Contributions

Team Member	Actions (Checkpoint 2)	Actions (Checkpoint 3)	Rating
Victoria Barnett	Worked on #3, 5, 8 Finalized Technology Stack, Created AWS account	Configured domain name, static IP, & DNS on AWS, Committed to About Page	8
Alison John	Primary for #5, Worked on #8	Initial setup of Database on instance as DB admin, Committed to About Page	7
Ansh Ankit Patel	Worked on #4, 8	Committed to About Page, Contributed to About page design, Corrected page routes	8
Nidhey Patel	Primary for #3, Worked on #8	Working on Credentials Readme, Committed to About Page	7
Pritham Singh Sandhu	Primary on #4 as Software Architect	Committed to About Page, Contributed to Credentials Readme	7
Tushin Kulshreshtha	Worked on #4, Primary on EC2 instance creation as Backend Lead, Key contributions to Technology Stack	Initial Setup of technology stack on instance as Backend Lead (Cloud server configuration), Contributed to About Page design, configured HTTPS	9
Yikang Xu	Primary for #6	Contributed to About page	5

SW Engineering

CSC648-848-05 Spring 2025

Serial

Team 02

- Victoria Barnett rbarnett@sfsu.edu Team Lead
- Tushin Kulshreshtha tkulshreshtha@sfsu.edu Back End Lead
- Ansh Patel apatel18@sfsu.edu Front End Lead
- Pritham Sandhu psandhu3@sfsu.edu Software Architect
- Alison John ajohn3@sfsu.edu Database Administrator
- Nidhey Patel npatel20@sfsu.edu Technical writer
- Yikang Xu yxu26@sfsu.edu Github Master

Milestone - 2

04/13/2025

3.1 Executive Summary

We are seeking backers for Serial, a new piece of project management software being developed by Team 02. Our goal is to create a uniquely visual-first project management framework that allows technical and administrative workers to track their tasks and progress side by side in one application and thus, one enterprise payment plan. We believe our software can vastly simplify the way product managers and developers keep track of their progress in a revolutionary new way, collapsing all project management abstractions into one visual project timeline that keeps everyone on task, focused, and working with the same information. [product name] enables new focus to drive team efficiency, it will enable smaller, more agile teams to

manage their project from humble beginnings all the way to an IPO in a more coordinated way by collapsing the confusing jumble of Product and Project management tools, Kanban boards, and Developer task tracking into one chronological-first organization method, and then filter the information as needed for whichever team is looking at it.

This application will provide a visual project management timeline, collapsing different levels of project abstraction from organization-wide initiatives down to individual team tasks. It aims to consolidate timelines, due dates, sprints and sprint points, organizational considerations, and the specific needs of product managers, development teams, and QA teams, all within a single, opinionated, chronological-first Project management framework. The app will offer multiple views of the same underlying project data, allowing for dynamic updates across teams. For example, a development task due on the 20th will automatically generate a corresponding review task for the product manager due on the 22nd, and vice versa with a new feature request. Potential features include calendar synchronization (possibly via .ics file generation), a team meeting scheduler, task generation and assignment, and GitHub integration for developers. Importantly, the application will support various project management methodologies, including Kanban, Agile, and sprint, all for a single price. This application offers several key benefits to users.

The value of Serial is that it's a cost-effective alternative to other project management software, providing a robust feature set without a premium price tag. By streamlining workflows and keeping tasks clearly defined, it boosts efficiency and helps users stay on track. The visual nature of the timeline-based approach caters to users who benefit from a strong visual separation of tasks, eliminating the need to constantly recall priorities. Furthermore, the timeline focus actively combats "backlog" clutter by requiring due dates for all assigned tasks, ensuring consistent reminders and preventing items from being forgotten. The unique design of this application centers around its visual-first approach, prioritizing a chronological project view that provides a clear and intuitive understanding of project progression. Another key differentiator is its straightforward pricing model. All features are available for a single price, eliminating the confusion and limitations of tiered subscriptions for personal or enterprise use. This accessible pricing structure makes the application particularly well-suited for smaller teams.

3.2 Data definitions

Main Terms

Project

Description: The active timeline Serial is displaying, with all associated tasks and other features. Projects group related tasks and are used for managing work that requires multiple steps or milestones.

Attributes:

- project_id (INT, Primary Key)
- title (VARCHAR(255), required)
- description (TEXT, optional)
- start_date (DATE)
- end_date (DATE)
- status (ENUM: active, completed, on_hold)
- owner (INT, Foreign Key → User)
- org_id (INT, Foreign Key -> Organization)

Metadata:

- status, owner, start_date, end_date

Raw Data:

- title, description

Supporting Data:

- timeline, chunks, tasks

Task

Description: The most basic unit in Serial, an item that has at minimum a description and a date attached. Tasks are the core unit of work in the software and can be assigned to users.

Attributes:

- task_id (INT, Primary Key)
- title (VARCHAR(255), required)
- description (TEXT)
- assigned_user_id (INT, Foreign Key → User)
- priority (ENUM: low, medium, high)
- status (ENUM: todo, in_progress, completed)
- due_date (DATE)
- created_at (TIMESTAMP)
- updated_at (TIMESTAMP)
- tags (VARCHAR(255), optional)
- parent_task_id (INT, Foreign Key → Task, optional)

Constraints:

- due_date must be >= created_at

UI Naming Consistency:

- Matches task_id and status field names in frontend components

---

Organization

Description: The largest possible unit in Serial, the group of all users. Organizations create teams and mint admins.

Attributes:

- organization_id (INT, Primary Key)
- name (VARCHAR(255), unique)

- Admin(VARCHAR(255), foreign key from User table)
- Teams(INT, foreign key)

Metadata:

- Org ID

Supporting Data:

- users, teams

Teams

Description: Smaller sections of Organizations. Teams can attach to either projects or organizations depending on their scope. Teams create, are assigned, and complete tasks. Team members have access to tasks within those projects.

Attributes:

- team_id (INT, Primary Key)
- name (VARCHAR(255))
- organization_id (INT, Foreign Key → Organization)
- members (ARRAY of User IDs or via join table)
- project_ids (ARRAY of INTs or via join table)

Constraints:

- Must belong to one organization

Timeline

Description: A chronological “loading bar” of your active project, the default project view in Serial.

Attributes:

- timeline_id (INT, Primary Key)
- project_id (INT, Foreign Key → Project)
- start_date (DATE)
- end_date (DATE)
- milestones (VARCHAR(255))

Metadata:

- start_date, end_date, milestones

Chunk

Description: A section of the loading bar, typically delineating a release window, review deadline, or some other important delineation between work ending and passing to another team. Chunks track significant project achievements and deadlines.

Attributes:

- chunk_id (INT, Primary Key)
- project_id (INT, Foreign Key → Project)
- title (VARCHAR(255))
- description (TEXT)
- due_date (DATE)
- status (ENUM: not_started, in_progress, completed)

Constraints:

- due_date must fall within the timeline of the project

Views

Description: A set of user interface filters that diverge from the timeline, reconstituting project data in different forms as needed for different team management strategies or inquiries.

Attributes:

- view_id (INT, Primary Key)
- user_id (INT, Foreign Key → User)
- filters (JSONB, storing filter config)
- view_type (ENUM: timeline, kanban, calendar)

Privileges

Description: Permissibility to use specific feature sets, configurable by role. Custom roles can be created with specific privileges.

Privileges include:

- Reassigning tasks
- Creating teams
- Editing roles
- Editing task deadlines
- Creating projects and editing parameters

User

Description: Represents an individual who interacts with the software. The user could be an admin, developer, team lead, manager, or custom role, each with different privileges and roles.

Attributes:

- user_id (INT, Primary Key)
- name (VARCHAR(255))

- email (VARCHAR(255), unique)
- password_hash (VARCHAR(255))
- role (VARCHAR(255)) — links to predefined roles
- profile_pic_url (VARCHAR(255), optional)
- status (ENUM: active, inactive, suspended)

Metadata:

- status, role

Role

Description: Predefined sets of privileges for ease of using Serial for the first time.

Attributes:

- role_id (INT, Primary Key)
- name (ENUM: admin, developer, manager, client)
- privileges (ARRAY of permissions or JSONB)

Custom Roles: Supported with editable privileges

Developer Role

Description: A user role that, by default, has view options for VCS enabled.

Attributes:

- user_id (INT, Primary Key)
- role_name (VARCHAR(255), required) - "Developer"
- view_vcs (BOOLEAN, default: true)
- can_assign_tasks (BOOLEAN, default: false)

- can_create_projects (BOOLEAN, default: false)
- can_comment (BOOLEAN, default: true)

Metadata:

- role_name, view_vcs, can_comment

Raw Data:

- user_id, view_vcs, can_assign_tasks

Supporting Data:

- Associated projects/tasks
- User permissions

Manager Role

Description: A user role that, by default, has granular task progress views and can assign tasks to teams.

Attributes:

- user_id (INT, Primary Key)
- role_name (VARCHAR(255), required) - "Manager"
- can_assign_tasks (BOOLEAN, default: true)
- view_task_progress (BOOLEAN, default: true)
- manage_teams (BOOLEAN, default: true)
- can_create_projects (BOOLEAN, default: false)

Metadata:

- role_name, view_task_progress, can_assign_tasks

Raw Data:

- user_id, manage_teams, view_task_progress

Supporting Data:

- Associated teams and projects
- Task progress views

Client Role

Description: A user that experiences Client views of the project via a guest account at minimum.

Attributes:

- user_id (INT, Primary Key)
- role_name (VARCHAR(255), required) - "Client"
- view_project_progress (BOOLEAN, default: true)
- can_leave_comments (BOOLEAN, default: true)
- can_assign_tasks (BOOLEAN, default: false)
- can_edit_projects (BOOLEAN, default: false)

Metadata:

- role_name, view_project_progress, can_leave_comments

Raw Data:

- user_id, view_project_progress

Supporting Data:

- Associated project progress
- Comment history

Comment

Description: Allows users to communicate about a task, ask questions, or provide updates. Users can add, edit, and delete comments based on their privileges. Comments are displayed in a task's activity feed.

Attributes:

- comment_id (INT, Primary Key)
- author_user_id (INT, Foreign Key → User)
- task_id (INT, Foreign Key → Task)
- content (TEXT)
- created_at (TIMESTAMP)

Supporting Data:

- Used for task communication threads

Privileges:

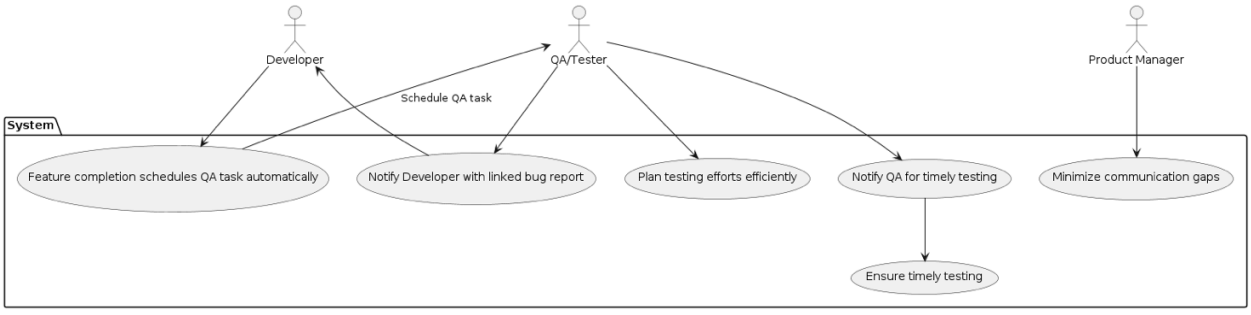
Various feature sets in Serial can be protected behind customizable Roles, which are assigned the feature sets as privileges. Some privileges are:

- Reassigning tasks
- Creating teams
- Editing Roles
- Editing task deadlines
- Creating Projects and editing their parameters

4.1 Use cases

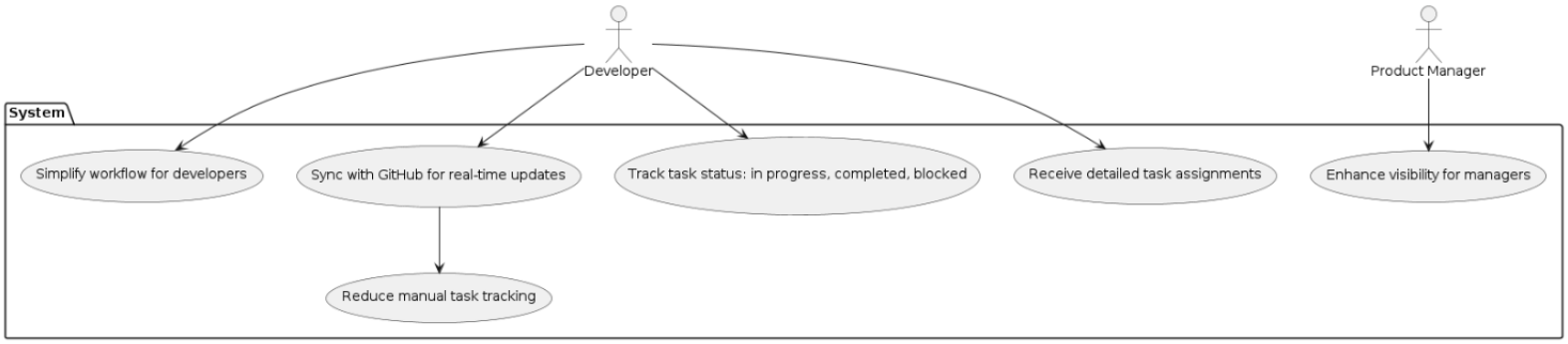
1. Create and Manage a Project Timeline

- **Actors**: Developers, Product Managers
- **Assumptions**: Developers, PrMs, PdMs need an easy way to track tasks & deadlines without switching between multiple platforms.
- **Use Case**: The project timeline provides teams with detailed task assignments, which they designate as in progress, completed, or blocked as they work. The system automatically syncs with GitHub to guarantee that merges and commits match actual task revisions. This interface reduces administrative overhead so that developers may concentrate on their work and gives product managers an open picture of continuous development initiatives.
- **Benefits**:
    - Reduces manual task tracking
    - Enhances visibility for managers
    - Simplifies workflow for developers.



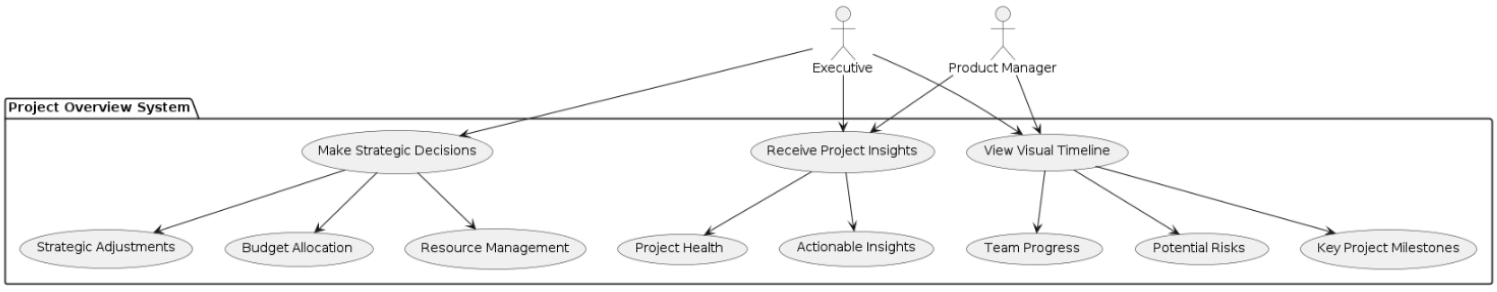
2. Automated Cross team Task Synchronization

- **Actors:** All, key interoperability for all actors
- **Assumptions**: Developers and QA teams work in separate cycles but need seamless handoffs.
- **Use Case**:In order to guarantee timely review without any human interaction, the system automatically schedules a QA assignment within a customizable time-window of a developer completing a feature. Notifications sent to the QA team help them to effectively coordinate testing initiatives. Should a problem arise, the developer is informed via a connected bug report, therefore fostering smooth cooperation and lowering project delays.
- **Benefits**:
    - Improves workflow efficiency
    - Ensures timely testing
    - Minimizes communication gaps.



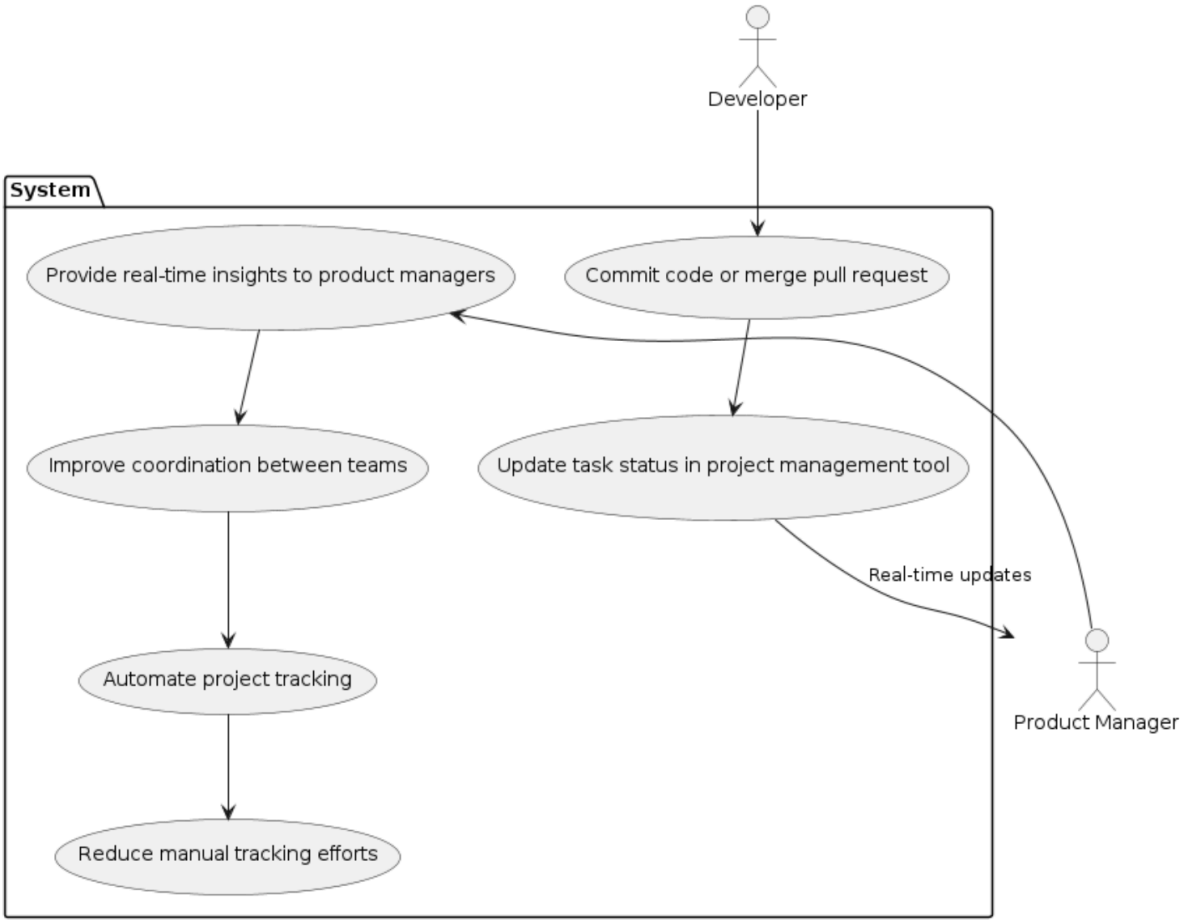
3. Visual Project Overview for Executives

- **Actors**: Executives, Project Managers, Team Leads
- **Assumptions**: Executives require high-level overviews rather than granular task details.
- **Use Case**: Executives have access to a visual timeline that is concise and displays key project milestones, team progress, and potential risks – a loading bar that captures all finer detail in the effort. Rather than delving into minutiae, they acquire practical understanding of the general state of the project, which enables quick decisions about resource management, budget allocation, and top-level strategy changes.
- **Benefits**:
    - Offers quick decision-making insights
    - Enhances resource allocation
    - Improves risk management.



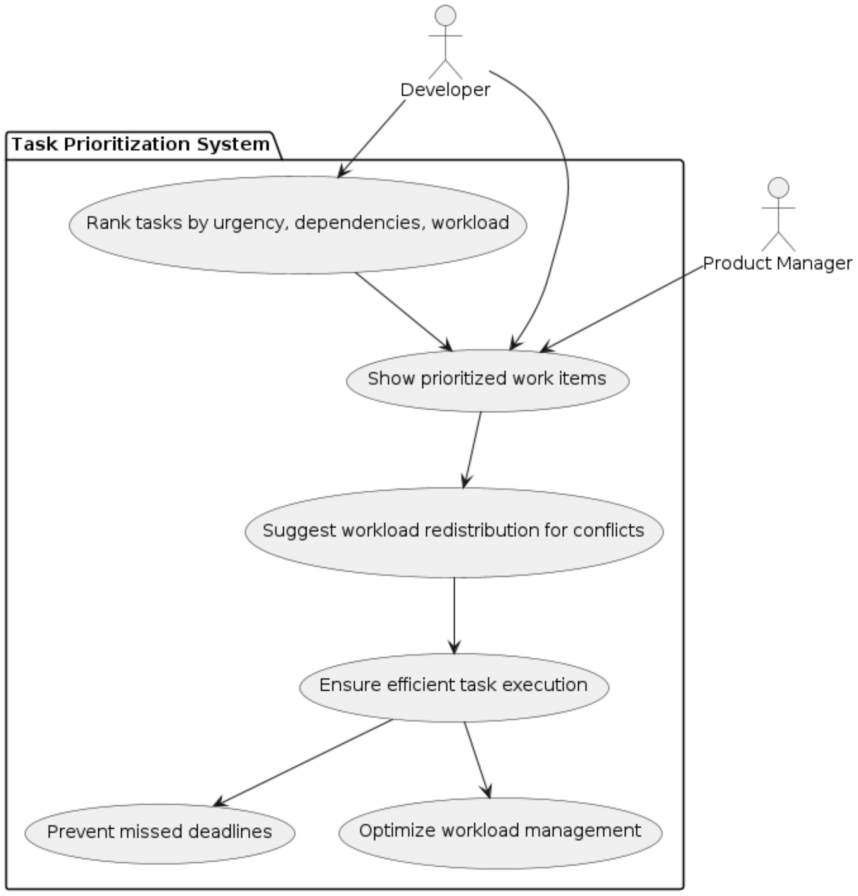
4. GitHub Integration for Real-Time Progress Updates

- **Actors**: Developers, Product Managers
- **Assumptions**: Developers use GitHub for code management.
- **Use Case**: The system changes the project management user interface when a developer commits code, submits a pull request, or when a pull request is approved. Product managers can get real-time information without having to check GitHub by hand. This makes it easier for the development and management teams to work together and keeps project tracking fully automated.
- **Benefits**:
    - Reduces manual tracking efforts
    - Provides real-time insights
    - Enhances collaboration



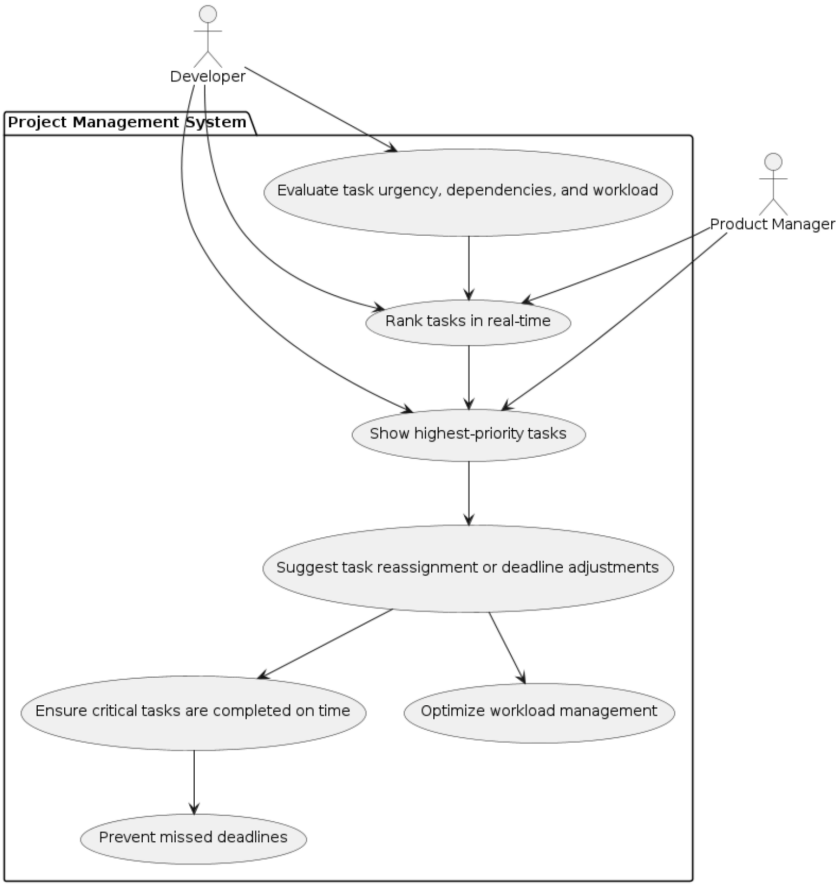
5. Task Prioritization Based on Deadlines & Dependencies

- **Actors**: Developers, Product Managers
- **Assumptions**: Projects have multiple dependencies that must be handled efficiently.
- **Use Case**: Task urgency, dependencies, and team workload all help the algorithm automatically rank activities. Clear prioritising of work items by developers and management enables them to concentrate on the most important. Should competing deadlines develop, the system recommends task reassignment to avoid congestion and guarantee consistent performance – Serial is opinionated about time, forcing you to make choices to avoid deadlock.
- **Benefits**:
    - Ensures efficient task execution
    - Prevents missed deadlines
    - Optimizes workload management.



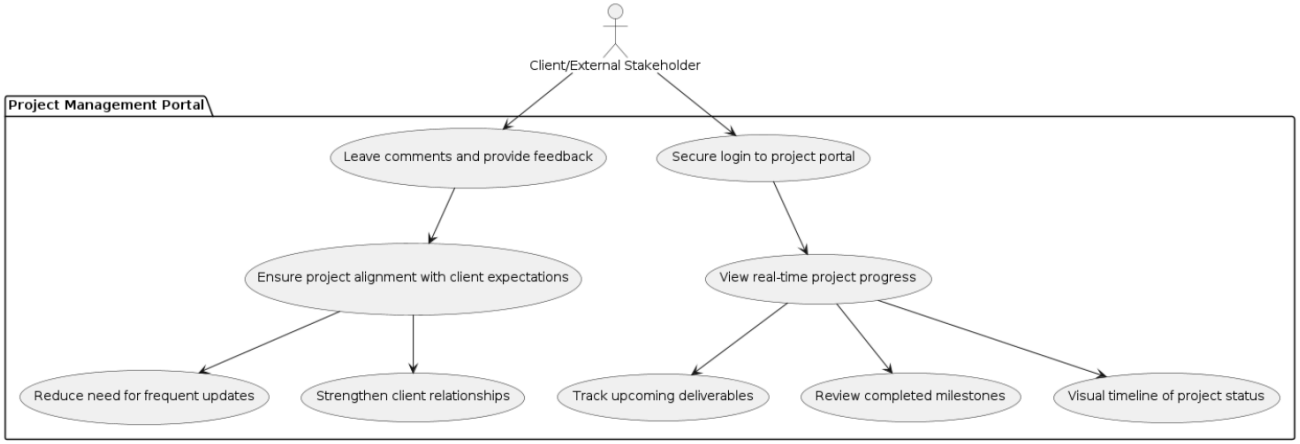
6. Schedule Meetings and Manage Calendar

- **Actors:** Developers, Product Managers
- **Assumptions:** Projects have multiple individuals who must sometimes meet regarding their tasks.
- **Use Case:**The project management software automatically evaluates task urgency, project dependencies, and current team workload using an intelligent algorithm. This system ranks activities, ensuring that both developers and product managers focus on the highest-priority tasks first, and facilitates easy ways for them to meet if needed. If competing deadlines arise, the app proactively suggests task reassignment or deadline adjustments to prevent bottlenecks, ensuring smooth and continuous progress. This dynamic prioritization helps teams manage their workload effectively and ensures critical tasks are completed on time.
- **Benefits:**
    - Ensures efficient task execution by always focusing on the most critical work items.
    - Prevents missed deadlines by identifying and resolving scheduling conflicts early.
    - Optimizes workload management across the team, reducing burnout and improving productivity.



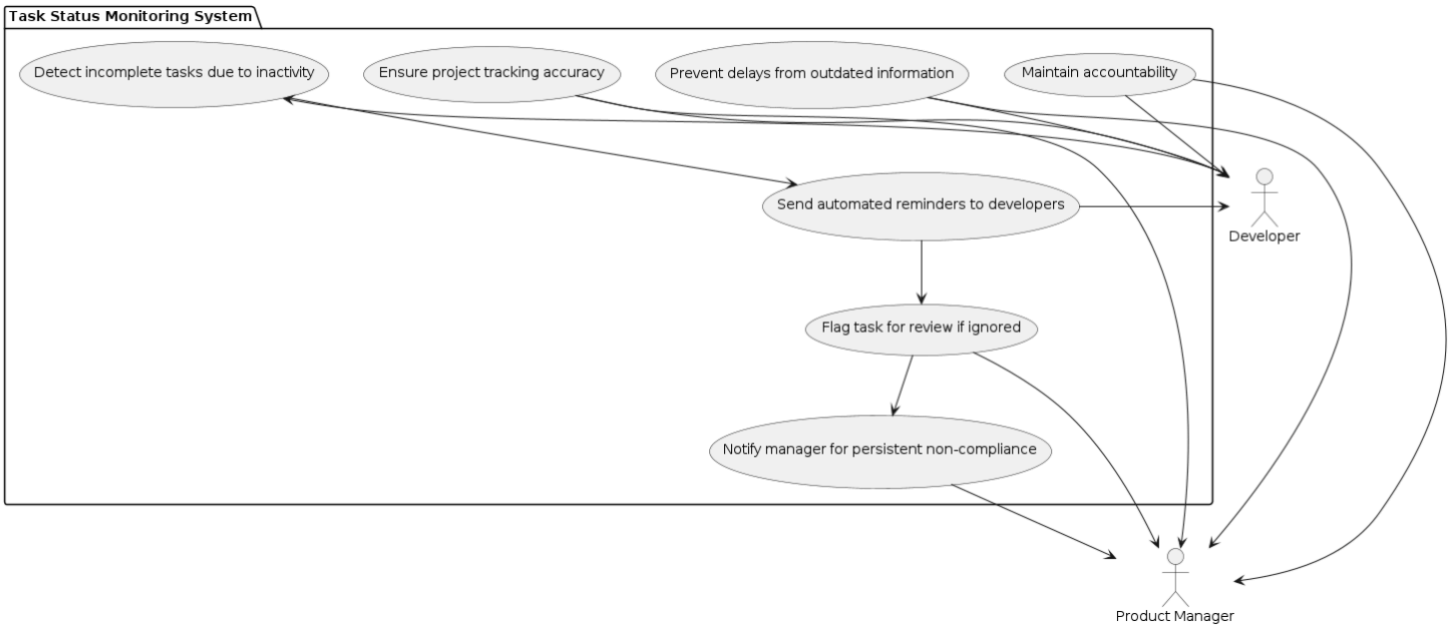
7. Review Project Progress and Provide Feedback

- **Actor:** Client/rnal Stakeholder
- **Assumptions:** Clients need real-time access to project status, upcoming deliverables, and a direct channel for feedback without constant back-and-forth communication.
- **Use Case:**
  Clients or outside stakeholders can access the project management app through a secure portal and see the real-time progress of the project, keep track of future deliverables, and look over goals that have already been met. Including finished activities, ongoing work, and any possible delays, the app offers a clear, graphic chronology of project status. Clients can immediately offer comments, questions, and feedback on certain aspects of the project, all configurable by the team. This direct feedback loop guarantees that the project remains in line with client expectations and helps to lower the necessity of regular status meetings and long-standing email exchanges.
- **Benefits:**
  - Enhances transparency by giving clients real-time project visibility.
  - Reduces the need for constant status updates through emails or meetings.
  - Facilitates quicker feedback, enabling teams to address client needs promptly.
  - Strengthens client relationships by keeping them informed and involved throughout the project lifecycle.



8. User Not Following App Protocol

- **Actors**: Developers, Product Managers
- **Assumptions**: Users sometimes neglect to update task status, causing workflow issues.
- **Use Case**: Confusion in project tracking results from a developer forgetting to mark a finished task as completed. The system notes inactivity and generates an automatic reminder. If not addressed, the work will be marked for review. Constant non-compliance forces a manager to be notified, therefore ensuring the problem is resolved. If an item falls off entirely, it will be flagged for later review: maybe this is a task your dont need to generate anymore?
- **Benefits**:
    - Ensures project tracking accuracy
    - Prevents delays due to outdated information
    - Maintains accountability.



4.2 Prioritized High-Level Functional Requirements

Priority 1 - Critical

User and Access Management
1.1 Users can register and log in using a username and password.
1.2 Users can reset their password via email verification.
1.3 Users can invite new team members to projects, be promoted to Admins, and manage access for other users.

Organization-Level Functionalities
2.1 Organizations can create and manage multiple projects within a single platform.
2.2 Organizations will have role-based access control, with permissions defined for administrators, product managers, developers, and QA teams.

Project Management Functionalities
3.1 Project managers can create, update, and track project roadmaps.
3.2 Projects can be visualized as timelines to track tasks, deadlines, and dependencies.
3.3 Project managers can define sprint cycles and assign sprint points.

Task Management Functionalities
4.1 Tasks can be created, assigned, and updated.
4.2 Tasks can have priority levels (e.g., high, medium, low).
4.4 Tasks can have due dates and automatic archive dates to prevent backlog clutter.
4.5 Tasks can have assigned users who receive notifications about approaching task deadlines via email or in-app alerts.

Workflow Automation Functionalities
5.1 Automated workflows are available for users to streamline task assignments and approvals.
5.3 Automatic task assignments based on predefined rules and conditions (eg, QA admin creates a task -> goes to QA team, creates a feedback meeting with development.
5.4 Workflow automation available for task dependencies to ensure that certain tasks cannot begin until prerequisites are met.

Collaboration and Communication Functionalities
8.1 Commenting on tasks and projects shall be supported.
8.2 @Mentions shall notify specific team members.
8.5 Project activity history shall be available to track changes.

Reporting and Analytics Functionalities
9.1 Real-time project progress reports shall be generated.

Priority 2 - Important

User and Access Management
1.4 Users can update their profile information.
1.5 User activity, such as task completion and project edits, will be logged.

Organization-Level Functionalities
2.3 Organization's users may view an enterprise-wide project timeline that visualizes all active projects.
2.4 Organizations can align their goals and deadlines through organization-wide sprint planning.
2.5 Organizations will have real-time data synchronization across all projects and all views of projects.

Project Management Functionalities
3.4 Project Task dependencies will be automatically generated(e.g., a development task due on the 20th triggers a review task due on the 22nd).
3.5 Projects will have filters to switch between different views of the same project data (e.g., development view, product manager view, QA view).

Task Management Functionalities
4.3 Tasks can be categorized by department (e.g., Development, QA, Product).

Workflow Automation Functionalities
5.2 Automatic task assignments based on predefined rules and conditions (eg, QA admin creates a task -> goes to QA team, creates a feedback meeting with development.
5.5 Automation for recurring tasks and reminders for routine project activities.

Calendar and Scheduling Functionalities
6.1 Shared project calendar for tracking team meetings and deadlines.
6.3 Scheduling is available for PrM/PdMs to assign team meetings directly from the application.
6.4 The calendar shall automatically generate meeting reminders for all invited participants.

Client and Stakeholder Collaboration Functionalities
7.1 External clients and stakeholders will have custom views to view project progress with controlled access.
7.2 Clients shall view specific milestones and tasks, as defined by organization teams making them available.
7.3 Clients will be able to share feedback directly within Serial, by comments and flags on their view.

Collaboration and Communication Functionalities
8.3 File attachments on tasks shall be allowed.

Reporting and Analytics Functionalities
9.2 Workload distribution across team members shall be viewable by managers.

9.3 Sprint burndown charts for Agile projects shall be provided.
9.4 Report exporting in PDF and CSV formats shall be supported.

Customization and User Preferences Functionalities
10.1 Dashboard layout customization shall be supported.
10.3 Notification preferences shall be configurable.

Priority 3 - Nice-to-Have

Calendar and Scheduling Functionalities
6.2 Calendar synchronization will be supported with external applications (Google Calendar, Outlook) via .ics file generation/import.
6.5 Calendar tooling shall enable users to propose, vote on, accept, and decline meeting times to improve scheduling efficiency.

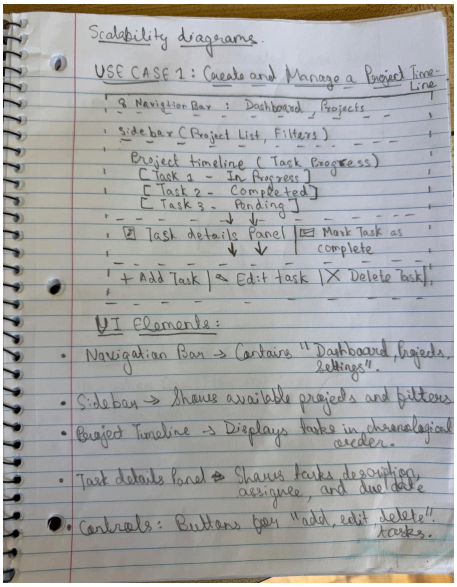
Client and Stakeholder Collaboration Functionalities
7.4 Client views shall have functionality to be real-time updated, or show a client only “completed” Chunk.
7.5 Clients shall be able to real-time communicate with Team leads and Organization managers.

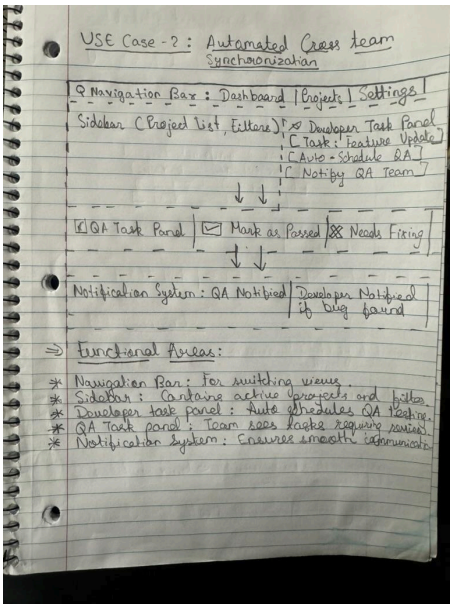
Collaboration and Communication Functionalities
8.4 A simple team chat system shall be provided for discussions.

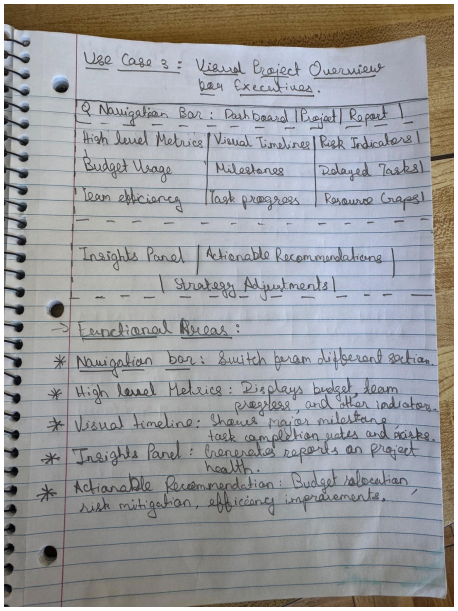
Reporting and Analytics Functionalities
9.5 Historical project analytics shall be available to track past performance.

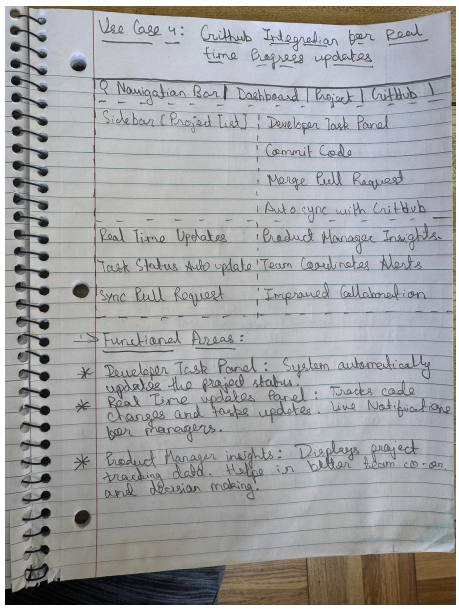
Customization and User Preferences Functionalities
10.2 Light mode and dark mode themes shall be available.
10.4 Custom project template creation shall be supported.
10.5 Task color label selection shall be available.

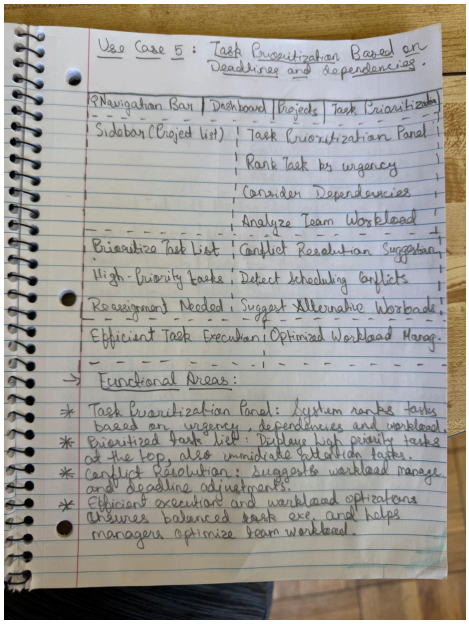
5.1 Mockups


Scalability diagrams.
USE CASE 1 : Create and Manage a Project Timeline
Navigation Bar : Dashboard, Projects
Sidebar (Project List, Filters)
Project timeline (Task Progress)
[Task 1 - In Progress]
[Task 2 - Completed]
[Task 3 - Pending]
Task details Panel
Mark Task as complete
+ Add Task
Edit task
Delete task
UI Elements:
Navigation Bar → Contains "Dashboard, Projects, Settings".
Sidebar → Shows available projects and filters.
Project Timeline → Displays tasks in chronological order.
Task details Panel → Shows tasks description, assignee, and due date
Controls: Buttons for "add, edit, delete" tasks.


USE Case - 2 : Automated Cross team Synchronization
Navigation Bar : Dashboard | Projects | Settings
Sidebar (Project List, Filters)
Developer Task Panel
[Task : Feature Update]
[Auto - Schedule QA]
[Notify QA Team]
QA Task Panel
Mark as Passed
Needs Fixing
Notification System : QA Notified
Developer Notified if bug found
Functional Areas:
Navigation Bar : For switching views.
Sidebar : Contains active projects and filters.
Developer task panel : Auto schedules QA testing.
QA Task panel : Team sees tasks requiring review.
Notification system : Ensures smooth communication.


Use Case 3 : Visual Project Overview for Executives.
Navigation Bar : Dashboard | Project | Report
High level Metrics
Visual Timelines
Risk Indicators
Budget Usage
Milestones
Delayed Tasks
Team efficiency
Task progress
Resource Gaps
Insights Panel
Actionable Recommendations
Strategy Adjustments
Functional Areas :
Navigation bar : Switch beam different section.
High level Metrics : Displays budget, team progress and filter indicators.
Visual timeline : Shows major milestone, task completion rates and risks.
Insights Panel : Generates reports on project health.
Actionable Recommendation : Budget allocation, risk mitigation, efficiency improvements.


Use Case 4: Github Integration for Real time Progress updates
Navigation Bar | Dashboard | Project | GitHub
Sidebar (Project List)
Developer Task Panel
Commit Code
Merge Pull Request
Auto sync with GitHub
Real Time Updates
Product Manager Insights
Task Status Auto update
Team Coordinates Marks
Sync Pull Request
Improved Collaboration
Functional Areas :
Developer Task Panel : System automatically updates the project status.
Real Time updates Panel : Tracks code changes and tasks updates. Live Notifications for managers.
Product Manager insights : Displays project tracking data. Helps in better team co-ordination and decision making.

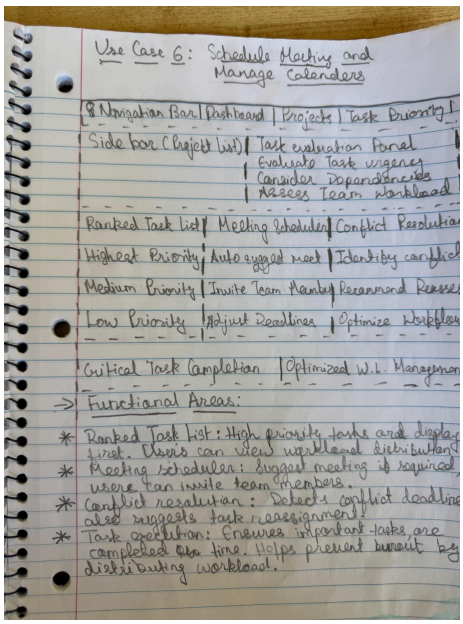


Use Case 5: Task Prioritization Based on Deadlines and Dependencies.

Navigation Bar	Dashboard	Projects	Task Prioritization
Sidebar (Project list)	Task Prioritization Panel	Rank Task by urgency	Consider Dependencies	Analyze Team Workload
Prioritize Task List	Conflict Resolution Suggestion
High-Priority Tasks	Detect Scheduling Conflicts
Reassignment Needed	Suggest Alternative Workloads
Efficient Task Execution	Optimized Workload Manag.

→ Functional Areas:

* Task Prioritization Panel: System ranks tasks based on urgency, dependencies and workload.
* Prioritized Task List: Displays high priority tasks at the top, also immediate attention tasks.
* Conflict Resolution: Suggests workload manage and deadline adjustments.
* Efficient execution and workload optimizations ensures balanced task exe, and helps managers optimize team workload.

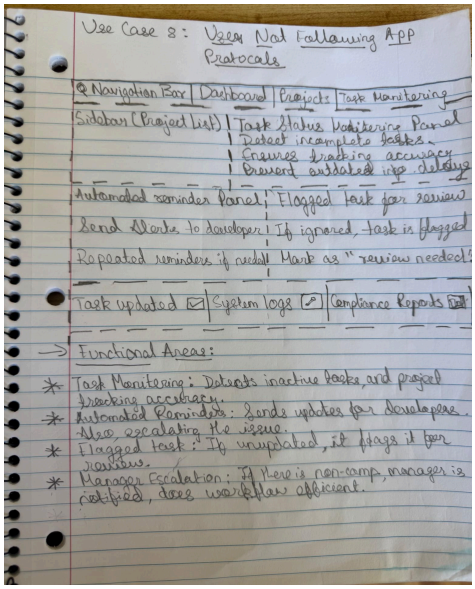


Use Case 6: Schedule Meeting and Manage Calendars

Navigation Bar	Dashboard	Projects	Task Priority
Side bar (Project List)	Task evaluation Panel	Evaluate Task urgency	Consider Dependencies	Assess Team Workload
Ranked Task List	Meeting Scheduler	Conflict Resolution
Highest Priority	Auto suggest meet	Identify conflict
Medium Priority	Invite Team Member	Recommend Reassi
Low Priority	Adjust Deadlines	Optimize Workflow
Critical Task Completion	Optimized W.L. Managmen

→ Functional Areas:

* Ranked Task list: High priority tasks and display first. Users can view workload distribution.
* Meeting scheduler: Suggest meeting if required, users can invite team members.
* Conflict resolution: Detects conflict deadline also suggests task reassignment.
* Task execution: Ensures important tasks are completed on time. Helps prevent burnout by distributing workload.

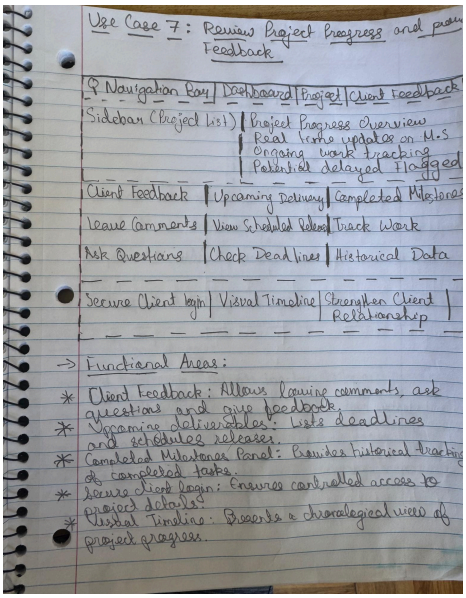


Use Case 8: Users Not Following App Protocols

Navigation Bar	Dashboard	Projects	Task Monitoring
Sidebar (Project List)	Task Status Monitoring Panel	Detect incomplete tasks.	Ensures tracking accuracy	Prevent outdated info delays
Automated reminder Panel	Flagged task for review
Send Alerts to developer	If ignored, task is flagged
Repeated reminders if needed	Mark as "review needed"
Task updated	System logs	Compliance Reports

→ Functional Areas:

* Task Monitoring: Detects inactive tasks, and project tracking accuracy.
* Automated Reminders: Sends updates for developers. Also escalating the issues.
* Flagged task: If unupdated, it flags it for reviews.
* Manager Escalation: If there is non-comp, manager is notified, does workflow efficient.



Use Case 7: Review Project Progress and pro Feedback

Navigation Bar	Dashboard	Project	Client Feedback
Sidebar (Project list)	Project Progress Overview	Real time updates on M.S	Ongoing work tracking	Potential delayed Flagged
Client Feedback	Upcoming Delivery	Completed Milestones
Leave Comments	View Scheduled Releases	Track Work
Ask Questions	Check Deadlines	Historical Data
Secure Client login	Visual Timeline	Strengthen Client Relationship

→ Functional Areas:

* Client Feedback: Allows leaving comments, ask questions and give feedback.
* Upcoming deliverables: Lists deadlines and schedules releases.
* Completed Milestones Panel: Provides historical tracking of completed tasks.
* Secure client login: Ensures controlled access to project details.
* Visual Timeline: Presents a chronological view of project progress.

5.2 High Level Database Architecture

Project

- **Name:** Project
- **Meaning:** A collection of tasks organized around a specific goal.
- **Usage:** Projects group related tasks and are used for managing work that requires multiple steps or milestones.

Key Fields:

- **Project ID** (unique identifier) (Primary Key) INT
- **Title** VARCHAR(255)
- **Description** TEXT
- **Start Date** DATE
- **End Date** DATE
- **Status** (active, completed, on hold) ENUM
- **Owner** (User ID) INT
- **Budget** (optional) DECIMAL(10, 2)
- **Milestones** (linked to task progress) VARCHAR(255)

Privileges:
Project owners or managers can edit and assign tasks, while team members can view tasks within the project and update their status.

Task

- **Name:** Task
- **Meaning:** A specific work item or action that needs to be completed.
- **Usage:** Tasks are the core unit of work in the software and can be assigned to users.

Key Fields:

- **Task ID** (unique identifier) (Primary Key) INT
- **Title** (brief description) VARCHAR(255)
- **Description** (detailed explanation) TEXT
- **Assigned User** (User ID) INT
- **Priority** (low, medium, high) ENUM
- **Status** (to-do, in-progress, completed) ENUM
- **Due Date** DATE
- **Created Date** DATE
- **Updated Date** DATE

- **Tags** (for categorization, e.g., bug, feature) VARCHAR(255)
- **Parent Task** (if it's part of a larger task, could be a task ID) INT

Behavior:
Tasks can be reassigned, updated with status changes, or marked as complete. They are tracked for progress and can have deadlines.

Team

- **Name:** Team
- **Meaning:** A group of users working together on a project.
- **Usage:** Teams are formed to manage collaboration and task assignments.

Key Fields:

- **Team ID** (unique identifier) (Primary Key) INT
- **Team Name** VARCHAR(255)
- **Members** (list of User IDs) VARCHAR(255)
- **Project Assignment** (linked to one or more projects) VARCHAR(255)

Behavior:
Teams are assigned to specific projects, and team members have access to tasks within those projects.

Time Log

- **Name:** Time Log
- **Meaning:** Tracks the time spent on tasks or projects.
- **Usage:** Time logs help measure productivity and ensure that tasks are completed on schedule.

Key Fields:

- **Log ID** (unique identifier) (Primary Key) INT
- **Task ID** (linked to a specific task) (Foreign Key) INT
- **User ID** (who logged the time) (Foreign Key) INT
- **Hours Spent** DECIMAL(5,2)
- **Date Logged** DATE

Behavior:
Users can log hours worked on tasks. Time logs are visible to task owners and project managers for tracking progress.

Comment

- **Name:** Comment
- **Meaning:** A message posted in relation to a task or project to provide feedback or updates.
- **Usage:** Comments allow users to communicate about a task, ask questions, or provide updates.

Key Fields:

- **Comment ID** (unique identifier) (Primary Key) INT
- **Author** (User ID) (Foreign Key) INT
- **Task ID** (linked to a specific task) (Foreign Key) INT
- **Content** (text of the comment) TEXT
- **Date Created** TIMESTAMP

Behavior:
Users can add, edit, and delete comments based on their privileges. Comments are displayed in a task's activity feed.

Chunk

- **Name:** Chunk
- **Meaning:** Significant points or stages in a project that represent the completion of key tasks or deliverables.
- **Usage:** Chunks are used to track major project achievements and ensure alignment with deadlines.

Key Fields:

- **Chunk ID** (unique identifier) (Primary Key) INT
- **Project ID** (linked to a specific project) (Foreign Key) INT
- **Title** VARCHAR(255)
- **Description** TEXT
- **Due Date** DATE
- **Status** (not started, in progress, completed) ENUM

Behavior:
Chunks are associated with project progress. When tasks linked to a chunk are completed, the chunk's status is updated.

User Roles & Privileges

- **Name:** Privileges & Roles
- **Meaning:** Defines access permissions for different user roles.
- **Usage:** Custom roles can be created with specific privileges.

Key Fields:

- **Role ID** (unique identifier) (Primary Key) INT
- **Name** (e.g., Admin, Developer, QA, Client) VARCHAR(255)
- **Permissions** (list of actions user can perform) TEXT

Organization

- **Meaning:** The largest possible unit in Serial, the group of all users. Organizations create projects and teams.

Key Fields:

- **Organization ID** (Primary Key) INT
- **Organization Name** VARCHAR(255)

Notification

- **Name:** Notification
- **Meaning:** Alerts and updates sent to users regarding changes in tasks, projects, or their assigned responsibilities.

Key Fields:

- **Notification ID** (unique identifier) (Primary Key) INT
- **Recipient User ID** (Foreign Key) INT
- **Notification Type** VARCHAR(50)
- **Content** TEXT
- **Status** (read/unread) ENUM
- **Timestamp** TIMESTAMP

Behavior:
Notifications can be triggered by task changes, comments, project status updates, etc. Users can mark notifications as read or dismiss them.

User

- **Name:** User
- **Meaning:** Represents an individual who interacts with the software.
- **Usage:** The user could be an administrator, manager, or team member, each with different privileges and roles.

Key Fields:

- **User ID** (unique identifier) (Primary Key) INT
- **Name** VARCHAR(255)
- **Email** VARCHAR(255)
- **Password** (encrypted) VARCHAR(255)
- **Role** (e.g., admin, manager, team member) VARCHAR(255)
- **Profile picture** VARCHAR(255)
- **Account status** (active, inactive, suspended) ENUM
- **Permissions** (read, write, delete) VARCHAR(255)

Privileges:
- Admin: Full access (create, edit, delete, assign tasks).

- Manager: Can assign tasks and view progress but cannot delete tasks.

- Team Member: Limited to task viewing and completion, no editing rights.

Timeline

- **Name:** Timeline
- **Meaning:** A chronological “loading bar” of your active project.

Key Fields:

- **Timeline ID** (unique identifier) (Primary Key) INT
- **Project ID** (linked to a specific project) (Foreign Key) INT
- **Start Date** DATE
- **End Date** DATE
- **Milestones** VARCHAR(255)

Behavior:
As tasks are completed, the timeline updates to reflect progress. It helps in tracking deadlines and ensuring timely completion of work.

Client

- **Name:** Client
- **Meaning:** A user role that experiences client views of the project via a guest account at minimum.
- **Usage:** Clients have limited access to project data, primarily to review progress, provide feedback, and track milestones.

Key Fields:

- **Client ID** (unique identifier) (Primary Key) - INT
- **Name** - VARCHAR(255)
- **Email** - VARCHAR(255)
- **Assigned Projects** (list of Project IDs) (Foreign Key) - VARCHAR(255)

Privileges:
Clients can view projects they are assigned to and leave comments but cannot edit tasks or assign roles.

Manager

- **Name:** Manager
- **Meaning:** A user role responsible for overseeing project progress and team performance.
- **Usage:** Managers have a higher level of control than regular team members, allowing them to track progress and manage assignments.

Key Fields:

- **Manager ID** (unique identifier) (Primary Key) - INT
- **Name** - VARCHAR(255)
- **Assigned Projects** (list of Project IDs) (Foreign Key) - VARCHAR(255)
- **Teams Supervised** (list of Team IDs) (Foreign Key) - VARCHAR(255)

Privileges:
Can assign tasks, view detailed progress reports, manage team roles, and approve project changes.

To keep the backend clean and easy to manage, we used a few design patterns that helped organize our code better.

Repository Pattern

We used this to keep our database stuff separate from the rest of the app. Instead of putting SQL queries all over the place, we made files like TaskRepository and UserRepository to handle all the DB actions. This made things way more organized.

Observer Pattern

This one helped with notifications. Like, when someone updates a task or leaves a comment, the system automatically lets the right people know. The cool part is that the task code doesn't need to worry about who gets notified — it happens in the background.

Strategy Pattern

This was perfect for handling roles and permissions. Since Admins, Managers, and others have different abilities, we used different strategies to define what each role can do. That way, if we ever need to change or add a new role, we can do it without messing up everything else.

Factory Pattern

We used this when creating users or tasks depending on their type. So, if a user is an Admin or a Manager, the setup might be a little different. The factory pattern helped us keep that logic clean and easy to manage.

These patterns weren't too complicated, but they made our code way easier to read, change, and scale in the future.

DBMS Selection:

We will use MySQL as the database system because it is reliable, secure, and handles large amounts of data efficiently, while also being relatively simple and having a high amount of overlap in our own personal or academic use compared to any other DBMS. It supports complex searches, ensures data is always accurate (ACID compliance), and can grow with the project by adding more servers if needed. Plus, it has features like JSON support, which helps store flexible data when required and will be a boon in a NodeJS project.

Database Organization

User and Access Management
Tables:

- Users (`user_id`, `username`, `email`, `password_hash`, `profile_info`, `created_at`, `updated_at`)
- User_Invitations (`invitation_id`, `inviter_id`, `invitee_email`, `status`, `created_at`)
- User_Activity_Log (`log_id`, `user_id`, `activity_type`, `timestamp`)
- Roles (`role_id`, `role_name`)
- User_Roles (`user_id`, `role_id`)

Organization-Level Management
Tables:
- Organizations (`org_id`, `org_name`, `created_at`)
- Organization_Users (`org_id`, `user_id`, `role_id`)
- Organization_Projects (`org_id`, `project_id`)

Project Management
Tables:
- Projects (`project_id`, `org_id`, `project_name`, `start_date`, `end_date`, `status`)
- Project_Timeline (`timeline_id`, `project_id`, `event_type`, `event_description`, `timestamp`)
- Sprints (`sprint_id`, `project_id`, `start_date`, `end_date`, `sprint_points`)
- Project_Views (`view_id`, `project_id`, `view_type`)

Task Management
Tables:
- Tasks (`task_id`, `project_id`, `task_name`, `task_description`, `assigned_to`, `priority`, `status`, `due_date`, `archive_date`)
- Task_Dependency (`task_id`, `dependent_task_id`)
- Task_Notifications (`notification_id`, `task_id`, `user_id`, `notification_type`, `timestamp`)

Workflow Automation
Tables:
- Automated_Workflows (`workflow_id`, `name`, `description`, `rule_conditions`, `created_at`)
- Task_Automation (`automation_id`, `task_id`, `workflow_id`, `status`)

Calendar and Scheduling
Tables:
- Project_Calendar (`event_id`, `project_id`, `event_name`, `event_date`, `event_type`, `participants`)
- Calendar_Integration (`integration_id`, `user_id`, `calendar_service`, `sync_status`)

Client and Stakeholder Collaboration
Tables:
- Client_Views (`view_id`, `client_id`, `project_id`, `view_type`, `allowed_milestones`)
- Client_Feedback (`feedback_id`, `client_id`, `project_id`, `comments`, `created_at`)

Collaboration and Communication

Tables:
- Comments (`comment_id`, `user_id`, `task_id`, `comment_text`, `timestamp`)
- Mentions (`mention_id`, `comment_id`, `mentioned_user_id`)
- File_Attachments (`file_id`, `task_id`, `file_name`, `file_path`, `uploaded_by`, `uploaded_at`)
- Chat_Messages (`message_id`, `sender_id`, `receiver_id`, `message_text`, `timestamp`)

Reporting and Analytics
Tables:
- Project_Reports (`report_id`, `project_id`, `report_type`, `generated_by`, `generated_at`)
- Workload_Analytics (`analytics_id`, `team_member_id`, `task_count`, `effort_estimate`)
- Historical_Analytics (`history_id`, `project_id`, `metric_type`, `metric_value`, `recorded_at`)

Customization and User Preferences
Tables:
- User_Settings (`setting_id`, `user_id`, `setting_key`, `setting_value`)
- Custom_Templates (`template_id`, `user_id`, `template_name`, `template_data`)

Security Measures
Tables:
- Audit_Logs (`log_id`, `user_id`, `action_type`, `timestamp`)
- Two_Factor_Auth (`user_id`, `auth_method`, `auth_status`)

Performance and Scalability Considerations
- Indexing for frequently queried fields (e.g., `user_id`, `project_id`, `task_id`).
- Partitioning of large tables (e.g., `User_Activity_Log`, `Audit_Logs`).
- Caching mechanisms for repetitive queries.
- Regular database backups for reliability.
- Horizontal scalability support.

Relationships

1. One-to-One (1:1)
   - Each user has a unique profile.
   - Each user has personalized dashboard settings.

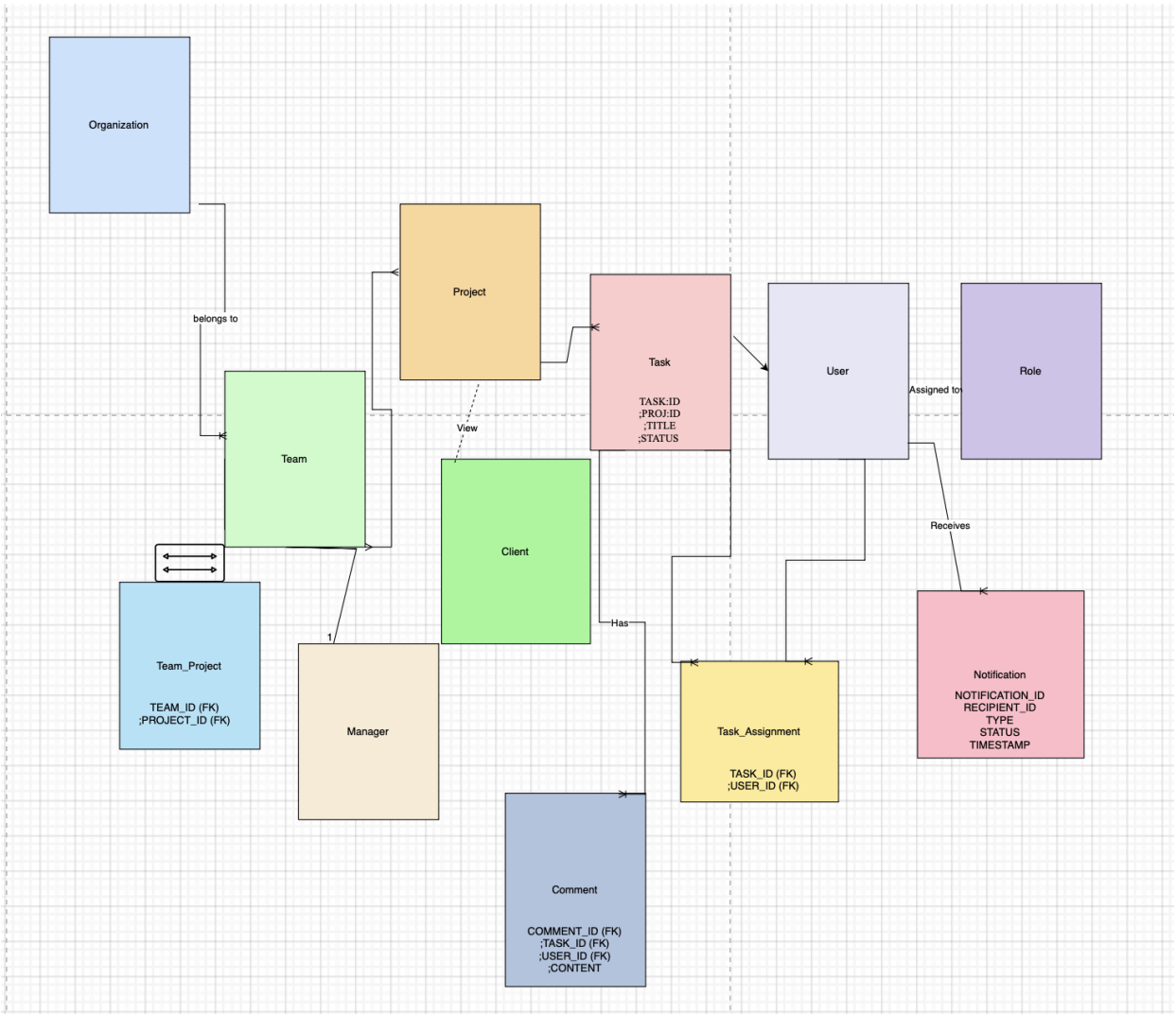
2. One-to-Many (1:M)
   - One organization can have multiple users.
   - One organization can have multiple projects.
   - One project can have multiple tasks.
   - One project manager can manage multiple projects.
   - One workflow rule can automate multiple task assignments.
   - One project calendar can contain multiple scheduled meetings.
   - One project can generate multiple reports.

- One user can perform multiple activities that are logged.
- One task or project can have multiple comments.

3. Many-to-One (M:1)
- Many users belong to one organization.
- Many projects belong to one organization.
- Many tasks belong to one project.
- Many scheduled meetings belong to one project calendar.
- Many reports belong to one project.

4. Many-to-Many (M:N)
- Users can be assigned to multiple projects, and projects can have multiple users.
- Users can belong to multiple organizations, and organizations can have multiple users.
- Tasks can be assigned to multiple users, and users can be assigned multiple tasks.
- A task can be part of multiple workflows, and workflows can contain multiple tasks.
- A meeting can have multiple participants, and a participant can be in multiple meetings.
- Clients can be associated with multiple projects, and projects can involve multiple clients.
- Users can mention multiple other users in comments, and each user can be mentioned in multiple comments.
- A report can contain data from multiple projects, and a project can appear in multiple reports.

> Entity Relation diagram :



Media Storage

For this project, we decided to store images, videos, and audio files in the file system using Redis instead of the database, however for the vertical prototype we still use LocalStorage for these functions. We will also utilize local storage via JWT to cache media, json data, etc until it is placed in Redis, for consistent UX if the connection ever gets shaky. We do not have any particular needs, this is mostly for profile photos and potentially media in comments later.

Why File System Over Database?

- Better Performance: Storing large files in the database can slow it down. The file system is faster for reading/writing media.
- Scalability: Easier to expand storage or switch to cloud storage (like AWS S3) if needed.

- Simpler Backups: Keeps the database lightweight since it only stores file paths instead of actual media files.

How We'll Store Media

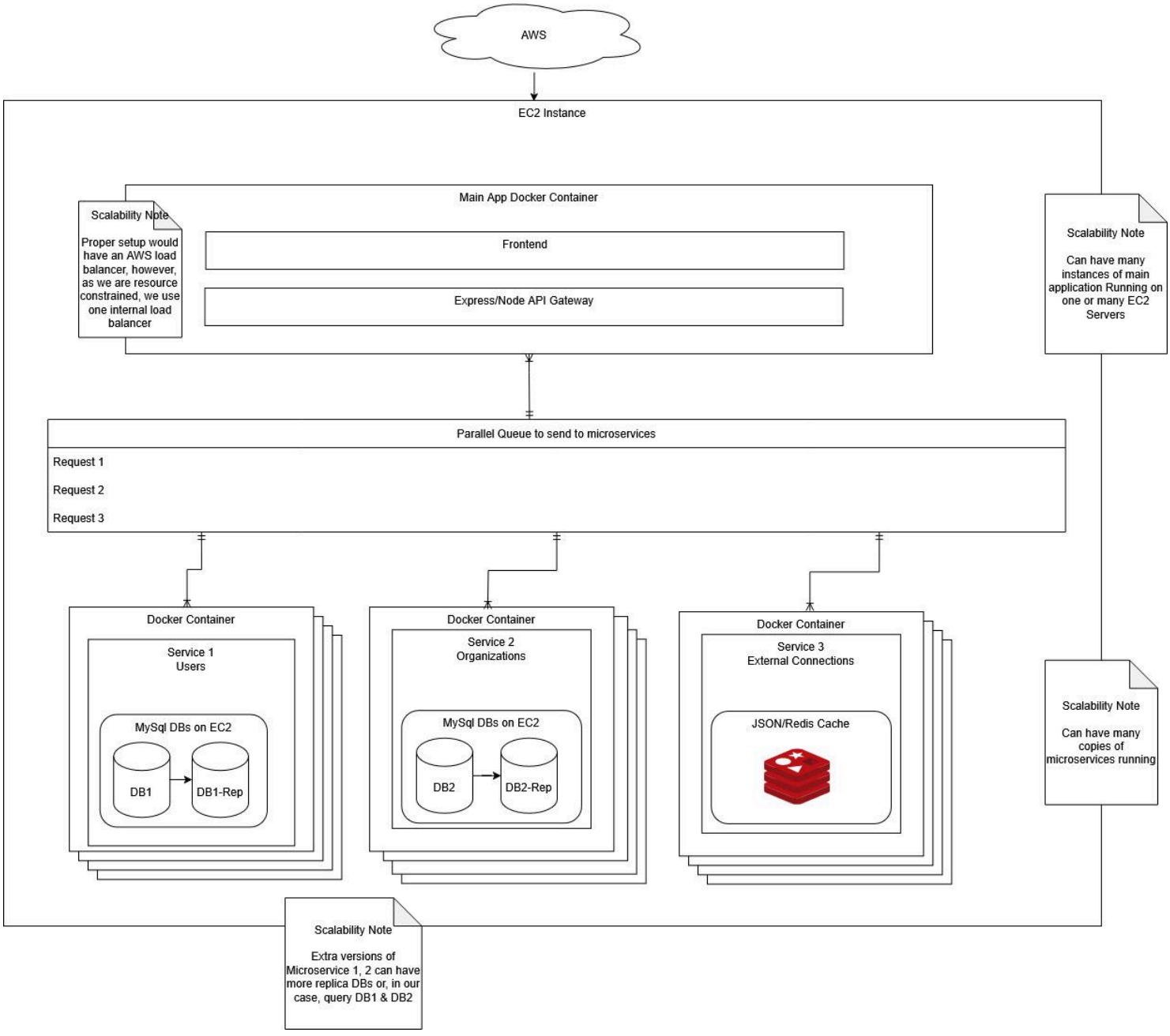
- Files will be saved in `/uploads/` with subfolders for images, videos, and audio.
- The database will store only file info (file name, path, uploaded by, etc.).

File Types Supported

- **Images:** JPG, PNG
- **Videos:** MP4, WebM
- **Audio:** MP3

5.3 Backend Architecture

Scalability Diagrams



Link to full size Scalability diagram (Google Drive/Draw.io):
https://drive.google.com/file/d/1K6o8ppVO0kDjtHlOqj91j6DidNiu8QAD/view?usp=drive_link

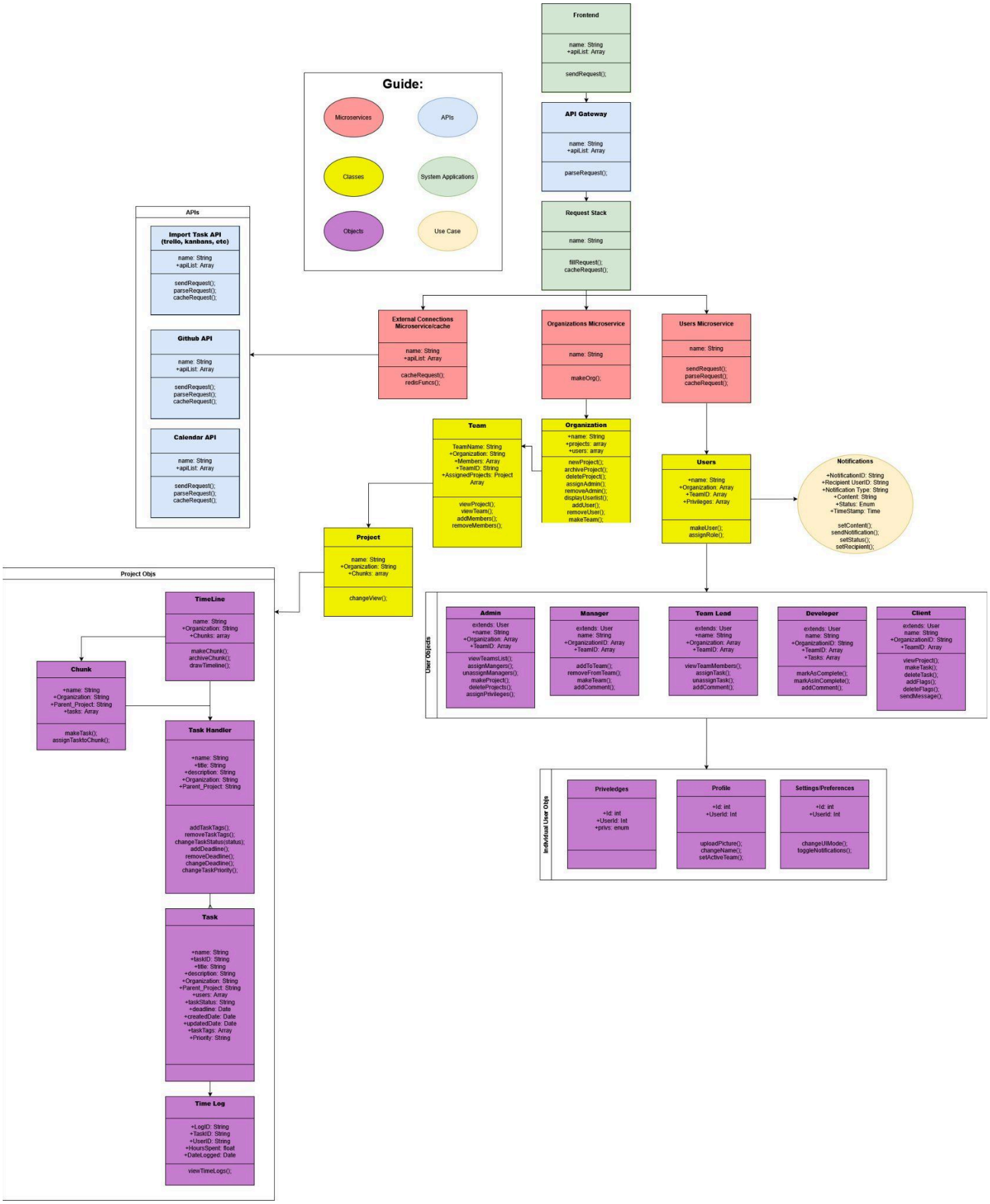
Architecture Summary

Serial's architecture is designed with scalability, ease of use, and minimal cost in mind. It utilized a microservice architecture managed by an Express API gateway for: users, organizations, and external connections. All instructions for these microservices are stored in a parallel queue - a stack-like piece of middleware that can ensure an instruction isn't skipped even if the service temporarily fails and has further request protection built in. We do load-balancing using Caddy, and cache many aspects of the application on the user-end and within Redis so that any short-term disconnections or processing isn't a disruption to a user.

Our parallel queue for requests to the microservices ensures some degree of fault tolerance, as requests are stored separately and repeatedly made until filled (within reasonable technical limit), we can allow users to work while a request finishes separate from them and that one bad packet doesn't ruin their request. Likewise, reliability is boosted by having databases tied to each microservice - no single error will hurt the entire application.

Each microservice and our main application & middleware are containerized using Docker, to reduce any potential deployment errors or lack of dependencies and standardize the operation such that it will work without a VM on any machine. So far, this looks like two different node applications: our backend, and the main vertical prototype logic. Our databases will be run as-is, but with replicas so that our middleware can make many requests to many replica versions of the database. These will be on one AWS server due to constraints of our project, but as it is only a series of MySQL connections, users with more robust infrastructure can benefit from the way this is designed and have copies and replicas of databases deployed wherever they need on any number of servers. Finally, Serial's architecture protects against potential erroneous infinite loops or accidental security breaks from bad actors: our API gateway middleware examines the plausibility of any request made, and will deny those that are unfeasible and not within the predefined limits of the middleware.

UML Class Diagram



Link to Full Resolution UML (Google Drive/Draw.io):
https://drive.google.com/file/d/1_Zk4aACFGBgaUj2zjUvdVl2biVd_qUJ9/view?usp=drive_link

UML Description

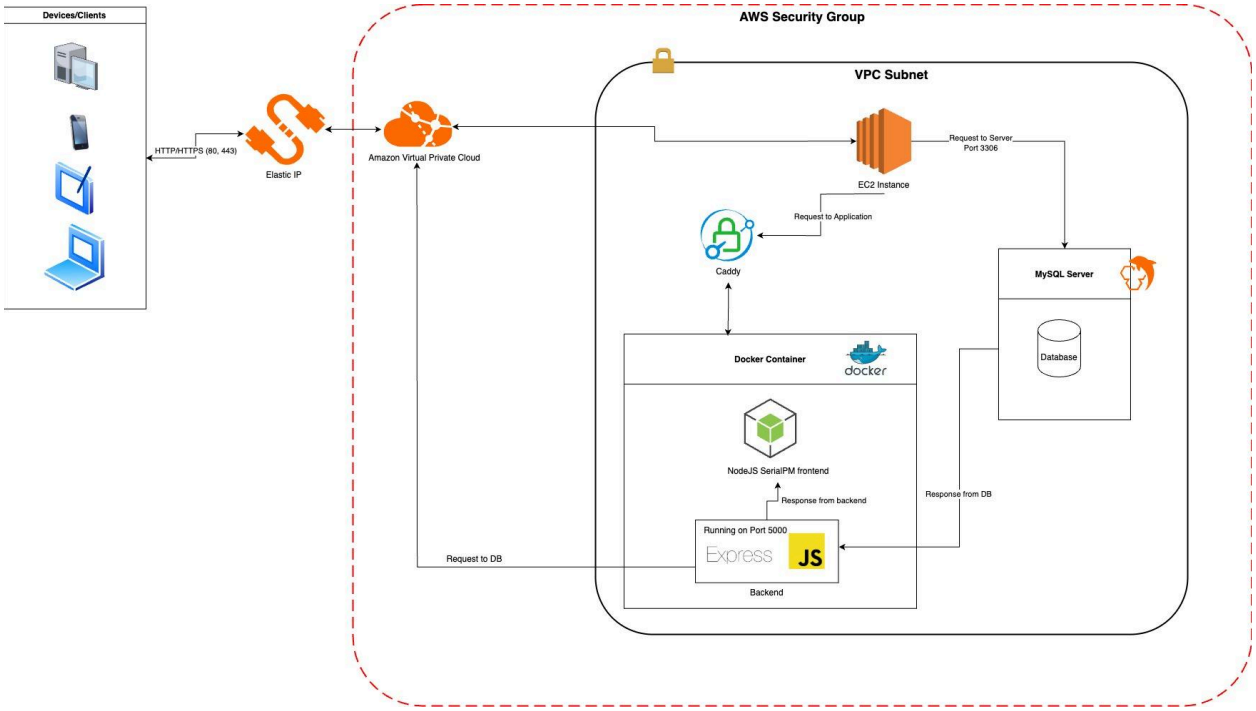
Our basic UML splits the application up into 6 different types of entities: APIs, Microservices, Classes, Objects, Use Cases, and System Applications. We have divided the core workings of the application into four essential aspects as described in our architecture summary, all pivoting around getting data to a “request stack” that is like a runtime stack for the interworking microservices and middleware:

- our two primary applications, the frontend and the API gateway in our backend, and
- The two types of things the API gateway communicates with via the request stack: microservices, and external APIs.

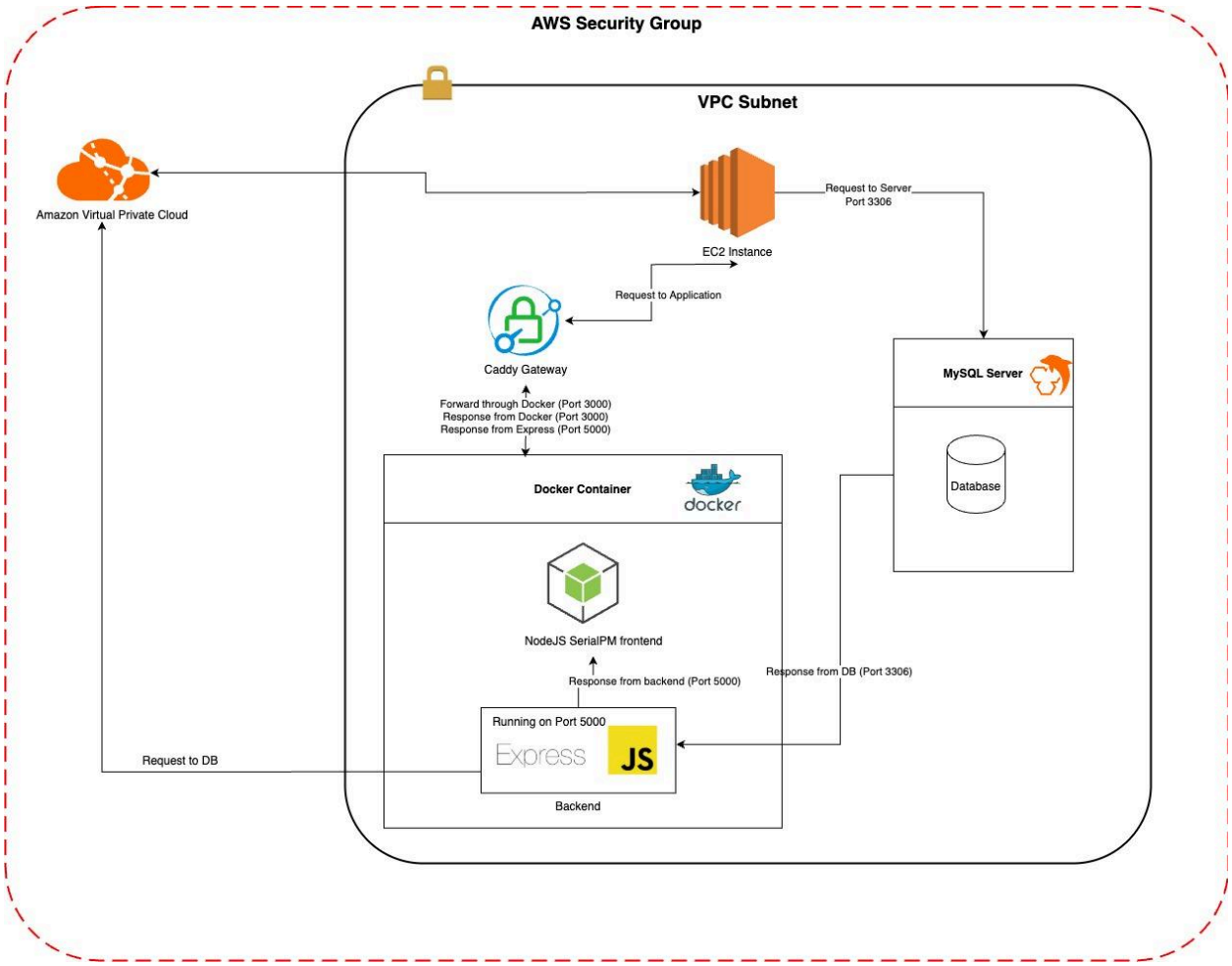
Both microservices utilize a variety of objects to efficiently do their main task (the management of Users and Tasks, respectively), and can have further refined objects attached to them that they supervise. Additionally, microservices can also have “use-cases” attached, which are classes with specific functions that interact at many levels of the application, such as the Notification system which needs to communicate back up to the API gateway to get between the Tasks and the Users.

5.4 High Level Application Network Protocols and Deployment Design

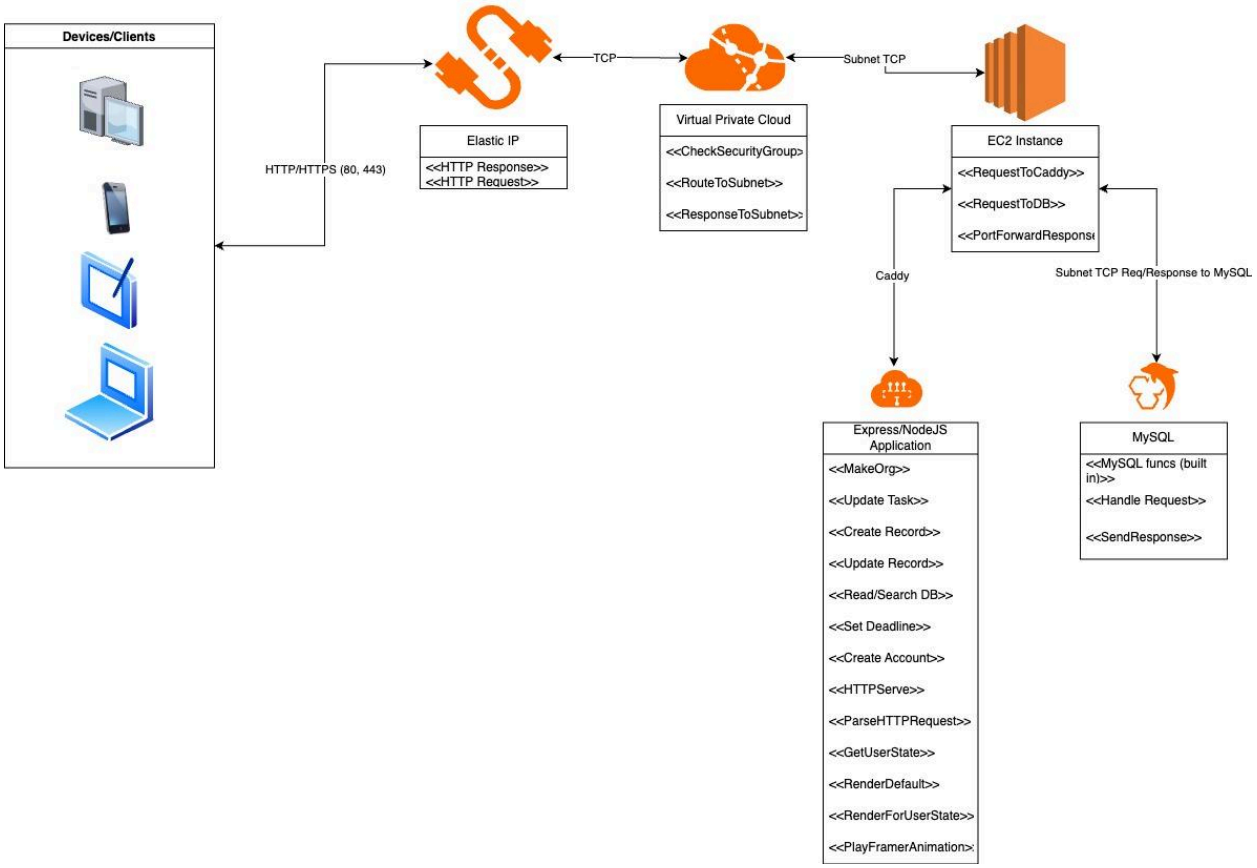
Combined Network and Deployment Diagram



Application Network Diagram



Deployment Diagram



Integration With External Components

There are currently no external libraries, only packages internally stored and accessed via NodeJS (Lucide, Framer-Motion, etc)

5.5 High Level APIs and Main Algorithms

Our system will feature multiple APIs to manage users, organizations, projects, tasks, and external integrations. The User Microservice API will handle all user-related data, including role-based access control, authentication (login/logout), and user preferences. The Organizations Microservice API will manage team structures, project assignments, and organizational settings. The Task Management API will facilitate task creation, assignment, and tracking, integrating with GitHub and Calendar APIs for enhanced workflow automation. Additionally, the External Microservice API will handle data retrieval from third-party services to support external dependencies.

A search algorithm will be implemented to allow users to efficiently search for tasks and projects within their organization. To optimize performance, we will utilize Fuzzy Search Algorithm, ensuring quick retrieval of relevant data. Additionally, caching mechanisms will be employed to store and serve frequently accessed data, such as organizational details and commonly queried project information, improving system responsiveness.

6. Functional Requirements

1. User and Access Management Functionalities

1.1 Users can register and log in using a username and password.

1.2 Users can reset their password via email verification.

1.3 Users can invite new team members to projects, be promoted to Admins, and manage access for other users.

1.4 Users can update their profile information.

1.5 User activity, such as task completion and project edits, will be logged.

2. Organization-Level Functionalities

2.1 Organizations can create and manage multiple projects within a single platform.

2.2 Organizations will have role-based access control, with permissions defined for administrators, product managers, developers, and QA teams.

2.3 Organization's users may view an enterprise-wide project timeline that visualizes all active projects.

2.4 Organizations can align their goals and deadlines through organization-wide sprint planning.

2.5 Organizations will have real-time data synchronization across all projects and all views of projects.

3. Project Management Functionalities

3.1 Project managers can create, update, and track project roadmaps.

3.2 Projects can be visualized as timelines to track tasks, deadlines, and dependencies.

3.3 Project managers can define sprint cycles and assign sprint points.

3.4 Project Task dependencies will be automatically generated(e.g., a development task due on the 20th triggers a review task due on the 22nd).

3.5 Projects will have filters to switch between different views of the same project data (e.g., development view, product manager view, QA view).

4. Task Management Functionalities

4.1 Tasks can be created, assigned, and updated.

4.2 Tasks can have priority levels (e.g., high, medium, low).

4.3 Tasks can be categorized by department (e.g., Development, QA, Product).

4.4 Tasks can have due dates and automatic archive dates to prevent backlog clutter.

4.5 Tasks can have assigned users who receive notifications about approaching task deadlines via email or in-app alerts.

5. Workflow Automation Functionalities

5.1 Automated workflows are available for users to streamline task assignments and approvals.

5.2 Automatic task assignments based on predefined rules and conditions (eg, QA admin creates a task -> goes to QA team, creates a feedback meeting with development.

5.3 Automated notifications for task updates, approaching deadlines, and project changes.

5.4 Workflow automation available for task dependencies to ensure that certain tasks cannot begin until prerequisites are met.

5.5 Automation for recurring tasks and reminders for routine project activities.

6. Calendar and Scheduling Functionalities

6.1 Shared project calendar for tracking team meetings and deadlines.

6.2 Calendar synchronization will be supported with external applications (Google Calendar, Outlook) via .ics file generation/import.

6.3 Scheduling is available for PrM/PdMs to assign team meetings directly from the application.

6.4 The calendar shall automatically generate meeting reminders for all invited participants.

6.5 Calendar tooling shall enable users to propose, vote on, accept, and decline meeting times to improve scheduling efficiency.

7. Client and Stakeholder Collaboration Functionalities

7.1 External clients and stakeholders will have custom views to view project progress with controlled access.

7.2 Clients shall view specific milestones and tasks, as defined by organization teams making them available.

7.3 Clients will be able to share feedback directly within Serial, by comments and flags on their view..

7.4 Client views shall have functionality to be real-time updated, or show a client only “completed” Chunk.

7.5 Clients shall be able to real-time communicate with Team leads and Organization managers.

8. Collaboration and Communication Functionalities

8.1 Commenting on tasks and projects shall be supported.

8.2 @Mentions shall notify specific team members.

8.3 File attachments on tasks shall be allowed.

8.4 A simple team chat system shall be provided for discussions.

8.5 Project activity history shall be available to track changes.

9. Reporting and Analytics Functionalities

9.1 Real-time project progress reports shall be generated.

9.2 Workload distribution across team members shall be viewable by managers.

9.3 Sprint burndown charts for Agile projects shall be provided.

9.4 Report exporting in PDF and CSV formats shall be supported.

9.5 Historical project analytics shall be available to track past performance.

10. Customization and User Preferences Functionalities

10.1 Dashboard layout customization shall be supported.

10.2 Light mode and dark mode themes shall be available.

10.3 Notification preferences shall be configurable.

10.4 Custom project template creation shall be supported.

10.5 Task color label selection shall be available.

7.1 Non Functional Requirements

Performance:

- The entire web application response time should be <5 seconds
- Client-side web page render time should be <3 seconds
- UI/UX element changes should be synchronized <2 seconds
- Changes to files >10MB should be synchronized <5 seconds
- Database queries should be <3 seconds
- The provided API response time should be <5 seconds
- The web application can handle 100 concurrent users

Reliability:

- The web application can handle 50 users without degradation
- The web application can ensure data consistency during data transactions
- The web application can handle concurrent data editions
- Partial functionality without internet connection
- Backups

Security:

- AES-256, BCrypt, or equivalent encryptions for servers and databases
- Audit logs
- API key rotation or use authorization protocol
- Encrypted user information
- User password specifications
- User permissions and privileges
- Two-Factor Authentication
- HTTPS connection
- Bot detection/CAPTCHA
- Rate limit

Usability:

- Intuitive and concise UI
- Responsive design
- UI/UX elements should have a latency within 5 seconds
- Undo/Redo/Version control for users
- Dark/Custom themes
- Integration with other services
- User guide is provided
- Multi-input support
- Multi-media support
- Accessibility/WCAG

Maintainability:

- Comprehensive documentation
- Version control for developers

- Modular features
- Tests for new functionalities

Portability:

- Cross-platform support - ie, as many browsers as possible
- Compatibility with multiple APIs
- Containerized development

Scalability:

- Vertical scaling with optimization
- Potential for horizontal scaling with additional servers
- Load balancing

Compatibility:

- Cross-Browser/OS/Device support
- Localization
- Backward compatibility

Compliance:

- GDPR/CCPA and other personal data collection consents
- NIST Cybersecurity Framework
- WCAG for accessibility

Supportability:

- Bug track and issues list
- Regular updates/patches
- User reviews
- Customer Service

Efficiency:

- Client-side memory usage should be <4GB
- Server-side memory usage should be <16GB

Coding Standards:

- Using GitHub for version control
- Master and Development branches
- Feature branches & PRs
- Modular and repeatable code
- Input Validation
- Algorithm optimization
- Readability
- Error Handling
- Comments and technical Documentation

Environmental Sustainability:

- Reduce power and bandwidth consumption – smallest resource footprint possible
- Backward compatibility
- Optimize the code to increase efficiency

7.2 Key Project Risks

1. Skill Risks
    a. The team consists of members with different skill sets and levels. This may include the technical aspects, such as the knowledge of the technology stack used for the project, as well as soft skills such as communication and coordination.
2. Schedule Risks
    a. The team has scheduled team meetings during and between class times. It may still be insufficient to establish a robust understanding for all team members. Even then, it is not guaranteed that all members can attend all meetings, as everyone has their own life.
3. Technical Risks
    a. The host servers may be prone to unauthorized access. The cloud services may be unable to handle malicious attacks such as DDoS.
    b. The selection of technologies may generate compatibility issues later in development.
4. Teamwork Risks
    a. Some of the designated tasks have dependencies that require the completion of other tasks.
    b. The workload distribution for team members may be uneven due to the difference in skill sets and levels.
5. Legal/Content Risks
    a. The team must be aware of copyright infringement, licensing requirements, intellectual property, privacy, data protection, or any other regulations.
    b. The project may require well-defined terms of services and user agreements.
    c. The team shall ensure that the project does not contain harmful or inappropriate content within its scope of usage.
6. Risk Mitigations
    a. Skill Risks: The team members can shrink their skill gaps by allocating extra time for learning, letting members educate others through mentoring, and collaborating for designated tasks to meet the skill requirements.
    b. Schedule Risk: The team can reduce it by prioritizing necessary tasks, planning ahead for contingencies, creating checkpoints to assess progression, and dynamically assigning members based on the current state.
    c. Technical Risks: The team can enforce stronger security policies and practices for data integrity; for compatibility, the team can create possible scenarios of the project to predict future integration issues.
    d. Teamwork Risks: The team can organize the individual tasks chronologically to reduce dependency and distribute them according to the team member’s skill and workload level.
    e. Legal/Content Risks: The team can ensure the project complies with laws and regulations, implements necessary standards, and reduces liability.

8.1 Competitive Analysis

Competitor's Table

Competitor	Strengths	Weaknesses	Pricing	Social Media Focus	Onboarding Experience
ClickUp	Tons of features, super customizable, works with other tools	Too many options can be confusing, takes time to learn	Free plan, paid starts at $7/user/month	Active on YouTube, Twitter, and LinkedIn	Step-by-step tutorials, but kinda overwhelming
Asana	Clean UI, easy task management, good for teamwork	Not great for tracking time, limited features on free plan	Free version, paid starts at **$10.99/user/month**	Engages a lot on LinkedIn and Twitter	Simple setup, has in-app guides
Trello	Super easy to use, great for visual organization	Not ideal for complex projects, weak reporting tools	Free plan, paid starts at **$5/user/month**	Big on Instagram, Twitter, and YouTube	Quick start, drag-and-drop style
Notion	Combines docs, tasks, and databases all in one	Not great for Agile teams, missing advanced project views	Free personal use, paid starts at **$8/user/month**	Very active on Twitter, YouTube, and TikTok	Flexible, but setup takes effort
Airtable	Clean UI with data visualization; Automation scripting available; Mobile support; API available; AI integration	Limited functions for free plan; Can only import from spreadsheet apps; No integration with external apps; Too many options	Free plan with 5 users; Paid start at **$20/user/month**	Active on LinkedIn and X; Presence on Facebook, Instagram and Youtube	Intuitive but may be overwhelmingly detailed; Spreadsheet-like experience

Competitive features table

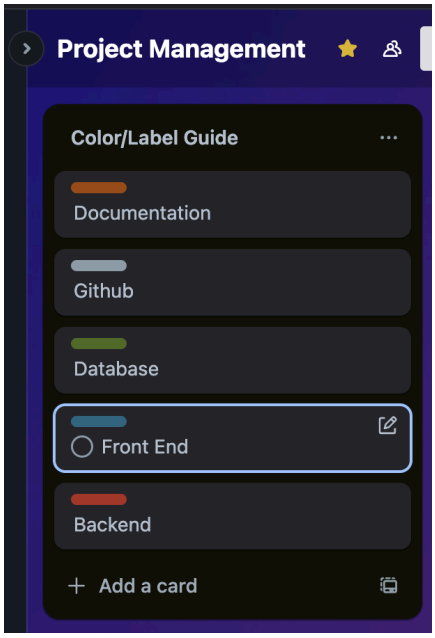
Feature	ClickUp	Asana	Trello	Notion	Airtable	Serial
Automated cross-team dependency	-	-	-	-	-	+
Customizable Organization Setups	+	Limited	Limited	Limited	+	+
Chronological Visual-Interactive UX	-	-	-	-	-	+
Automations	+	+	-	-	+	+
Self-hosting & User-modification	-	-	-	-	-	+

Summary

Our competitive edge will be both in our featureset and our planned pricing model. Presently, none of the competitors do everything: some like Notion and Trello have imposed collaboration frameworks for a stronger built-in experience, and some like Airtable rely on heavy integration on a paid plan to make full use of the application. Serial will be flexible where it matters: organizationally, visually, and with integrations, self-hosting, and modification available for both free and paid users. However, it will have a solid backbone of an imposed framework where needed: in its chronological-first basis, tasks will need concrete dates attached to then flexibly fit in whatever views your organization uses. By distilling the primary unit of information all Project Management software uses into its most basic form, Serial will be able to rebuild it in a customizable way as needed by the user, allowing us to be more flexible than the competition.

8.2 Project Management

We managed M2 tasks by switching from our M1 method of describing tasks to one-another and having the Team Lead check in on progress over discord, to utilizing a Trello [link to join is here] board for all tasks. We started by establishing a classification system for each of our tasks:



And then, by choosing which categories would be on our Trello Kanban board, settling on “Backlog, In-progress, Blocked, Review, and Done” as our first Board setup. Additionally, we are using the due dates on cards to pick end-dates for tasks in between Milestone due-dates, that way we can accomplish tasks appropriately in series and have them done as a reference for later-on portions of the milestone.

9.1 Checklist

Task	ISSUE	ON TRACK	DONE
The Team has found a time slot to meet outside of class			✅
GitHub Master has been chosen			✅
The team has collectively decided on and agreed to use the listed software tools and deployment server.			✅
The team is ready to use the chosen front-end and back-end frameworks, and those who need to learn are actively working on it.			✅
The Team Lead has ensured that all members have read and understand the final M1 before submission.			✅
GitHub is organized as discussed in class (e.g., master branch, development branch, folder for milestone documents, etc.).			✅

10. High-Level System Architecture & Technologies

Primary Technologies:

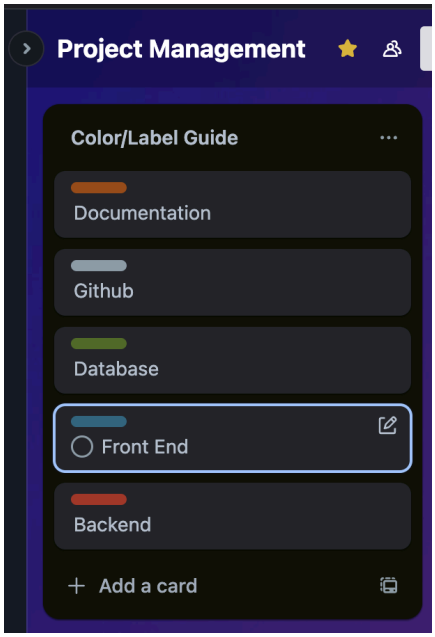
- Cloud server host: AWS t2 micro (1vcpu, 1gb RAM)
- OS: Ubuntu Server 24.04 LTS
- Database: MySQL 9.2
- Web Server: Express (4.21.2 unless we see an issue, then 5.0.1)
- Backend Languages, frameworks: JavaScript ES2023, Node 22.13.1
- Frontend Languages, frameworks: JavaScript ES2023, React 19.0, building via VIte 2.9

Other important technologies or packages:

- anime.js
- NPM
- GSAP
- Caddy
- Namecheap via GitHub Student for free domain & SSL
- Tailwind CSS
- Vite
- JWT token
- BcryptJS
- Lucide icons package
- Dotenv package for security concerns
- Multer for form data handling
- React-scroll and react-zoom-pan-pinch for Project timeline interaction UX
- Docker: Docker Container on EC2

12. Project Management

We managed M2 tasks by switching from our M1 method of describing tasks to one-another and having the Team Lead check in on progress over discord, to utilizing a Trello [link to join is here] board for all tasks. We started by establishing a classification system for each of our tasks:



And then, by choosing which categories would be on our Trello Kanban board, settling on “Backlog, In-progress, Blocked, Review, and Done” as our first Board setup. Additionally, we are using the due dates on cards to pick end-dates for tasks in between Milestone due-dates, that way we can accomplish tasks appropriately in series and have them done as a reference for later-on portions of the milestone.

13. List Of Team Contributions

Team Member	Actions (Checkpoint 1)	Actions (Checkpoint 2)	Rating
Victoria Barnett	Project management section draft, backend planning assist, UML Diagram primary	Profile page, project page, sessions jwt/bcrypt setup between signup/in/profile, created devcontainer for dependency ease	8
Alison John	Backend planning assist, expanded data defs, database requirements	Backend/database connection assist, sql creation script, database setup based on UML/other docs	7
Ansh Ankit Patel	Frontend mockups primary, Entity Relationship diagram	Sign-in, sign-up pages, dashboard page	8
Nidhey Patel	Updated title page, M2v1 revisions, primary technical wrter	Assisted with ranking system, created task component, technical docs review	7
Pritham Singh Sandhu	Helped Frontend mockups, prioritized functional reqs assist, expanded data defs, DBMS selection	Signup page, signin page	7
Tushin Kulshreshtha	Search section, UML diagram assist, backend architecture planning assist,	Landing page, backend/database connection, CRUD setup via api.js, fixed login functionality, many tweaks, env setup	9
Yikang Xu	Key project risks, prioritized functional reqs primary, media storage	Ranking/rating component, assisted profile,	7

SW Engineering

CSC648-848-05 Spring 2025

Serial

Team 02

- Victoria Barnett rbarnett@sfsu.edu Team Lead
- Tushin Kulshreshtha tkulshreshtha@sfsu.edu Back End Lead
- Ansh Patel apatel18@sfsu.edu Front End Lead
- Pritham Sandhu psandhu3@sfsu.edu Software Architect
- Alison John ajohn3@sfsu.edu Database Administrator
- Nidhey Patel npatel20@sfsu.edu Technical writer
- Yikang Xu yxu26@sfsu.edu Github Master

Milestone - 3

05/01/2025

3. Data definitions

Main Terms

Project

Description: The active timeline Serial is displaying, with all associated tasks and other features. Projects group related tasks and are used for managing work that requires multiple steps or milestones.

Attributes:

- project_id (INT, Primary Key)
- title (VARCHAR(255), required)
- description (TEXT, optional)
- start_date (DATE)
- end_date (DATE)
- status (ENUM: active, completed, on_hold)
- owner (INT, Foreign Key → User)
- org_id (INT, Foreign Key -> Organization)

Metadata:

- status, owner, start_date, end_date

Raw Data:

- title, description

Supporting Data:

- timeline, chunks, tasks

Task

Description: The most basic unit in Serial, an item that has at minimum a description and a date attached. Tasks are the core unit of work in the software and can be assigned to users.

Attributes:

- task_id (INT, Primary Key)
- title (VARCHAR(255), required)
- description (TEXT)
- assigned_user_id (INT, Foreign Key → User)
- priority (ENUM: low, medium, high)
- status (ENUM: todo, in_progress, completed)
- due_date (DATE)
- created_at (TIMESTAMP)
- updated_at (TIMESTAMP)
- tags (VARCHAR(255), optional)
- parent_task_id (INT, Foreign Key → Task, optional)

Constraints:

- due_date must be >= created_at

UI Naming Consistency:

- Matches task_id and status field names in frontend components

---

Organization

Description: The largest possible unit in Serial, the group of all users. Organizations create teams and mint admins.

Attributes:

- organization_id (INT, Primary Key)
- name (VARCHAR(255), unique)
- Admin(VARCHAR(255), foreign key from User table)
- Teams(INT, foreign key)

Metadata:

- Org ID

Supporting Data:

- users, teams

Teams

Description: Smaller sections of Organizations. Teams can attach to either projects or organizations depending on their scope. Teams create, are assigned, and complete tasks. Team members have access to tasks within those projects.

Attributes:

- team_id (INT, Primary Key)
- name (VARCHAR(255))
- organization_id (INT, Foreign Key → Organization)
- members (ARRAY of User IDs or via join table)
- project_ids (ARRAY of INTs or via join table)

Constraints:

- Must belong to one organization

Timeline

Description: A chronological “loading bar” of your active project, the default project view in Serial.

Attributes:

- timeline_id (INT, Primary Key)
- project_id (INT, Foreign Key → Project)
- start_date (DATE)
- end_date (DATE)
- milestones (VARCHAR(255))

Metadata:

- start_date, end_date, milestones

Chunk

Description: A section of the loading bar, typically delineating a release window, review deadline, or some other important delineation between work ending and passing to another team. Chunks track significant project achievements and deadlines.

Attributes:

- chunk_id (INT, Primary Key)
- project_id (INT, Foreign Key → Project)
- title (VARCHAR(255))
- description (TEXT)
- due_date (DATE)
- status (ENUM: not_started, in_progress, completed)

Constraints:

- due_date must fall within the timeline of the project

Views

Description: A set of user interface filters that diverge from the timeline, reconstituting project data in different forms as needed for different team management strategies or inquiries.

Attributes:

- view_id (INT, Primary Key)
- user_id (INT, Foreign Key → User)
- filters (JSONB, storing filter config)
- view_type (ENUM: timeline, kanban, calendar)

Privileges

Description: Permissibility to use specific feature sets, configurable by role. Custom roles can be created with specific privileges.

Privileges include:

- Reassigning tasks
- Creating teams
- Editing roles
- Editing task deadlines
- Creating projects and editing parameters

User

Description: Represents an individual who interacts with the software. The user could be an admin, developer, team lead, manager, or custom role, each with different privileges and roles.

Attributes:

- user_id (INT, Primary Key)
- name (VARCHAR(255))
- email (VARCHAR(255), unique)
- password_hash (VARCHAR(255))
- role (VARCHAR(255)) — links to predefined roles
- profile_pic_url (VARCHAR(255), optional)
- status (ENUM: active, inactive, suspended)

Metadata:

- status, role

Role

Description: Predefined sets of privileges for ease of using Serial for the first time.

Attributes:

- role_id (INT, Primary Key)
- name (ENUM: admin, developer, manager, client)
- privileges (ARRAY of permissions or JSONB)

Custom Roles: Supported with editable privileges

Developer Role

Description: A user role that, by default, has view options for VCS enabled.

Attributes:

- user_id (INT, Primary Key)

- role_name (VARCHAR(255), required) - "Developer"
- view_vcs (BOOLEAN, default: true)
- can_assign_tasks (BOOLEAN, default: false)
- can_create_projects (BOOLEAN, default: false)
- can_comment (BOOLEAN, default: true)

Metadata:

- role_name, view_vcs, can_comment

Raw Data:

- user_id, view_vcs, can_assign_tasks

Supporting Data:

- Associated projects/tasks
- User permissions

Manager Role

Description: A user role that, by default, has granular task progress views and can assign tasks to teams.

Attributes:

- user_id (INT, Primary Key)
- role_name (VARCHAR(255), required) - "Manager"
- can_assign_tasks (BOOLEAN, default: true)
- view_task_progress (BOOLEAN, default: true)
- manage_teams (BOOLEAN, default: true)
- can_create_projects (BOOLEAN, default: false)

Metadata:

- role_name, view_task_progress, can_assign_tasks

Raw Data:

- user_id, manage_teams, view_task_progress

Supporting Data:

- Associated teams and projects
- Task progress views

Client Role

Description: A user that experiences Client views of the project via a guest account at minimum.

Attributes:

- user_id (INT, Primary Key)
- role_name (VARCHAR(255), required) - "Client"
- view_project_progress (BOOLEAN, default: true)
- can_leave_comments (BOOLEAN, default: true)
- can_assign_tasks (BOOLEAN, default: false)
- can_edit_projects (BOOLEAN, default: false)

Metadata:

- role_name, view_project_progress, can_leave_comments

Raw Data:

- user_id, view_project_progress

Supporting Data:

- Associated project progress
- Comment history

Comment

Description: Allows users to communicate about a task, ask questions, or provide updates. Users can add, edit, and delete comments based on their privileges. Comments are displayed in a task's activity feed.

Attributes:

- comment_id (INT, Primary Key)
- author_user_id (INT, Foreign Key → User)
- task_id (INT, Foreign Key → Task)
- content (TEXT)
- created_at (TIMESTAMP)

Supporting Data:

- Used for task communication threads

Privileges:

Various feature sets in Serial can be protected behind customizable Roles, which are assigned the feature sets as privileges. Some privileges are:

- Reassigning tasks
- Creating teams
- Editing Roles
- Editing task deadlines
- Creating Projects and editing their parameters

4. Prioritized High-Level Functional Requirements

Priority 1 - Critical

User and Access Management
1.1 Users can register and log in using a username and password.
1.2 Users can reset their password via email verification.
1.3 Users can invite new team members to projects, be promoted to Admins, and manage access for other users.

Organization-Level Functionalities
2.1 Organizations can create and manage multiple projects within a single platform.
2.2 Organizations will have role-based access control, with permissions defined for administrators, product managers, developers, and QA teams.

Project Management Functionalities
3.1 Project managers can create, update, and track project roadmaps.
3.2 Projects can be visualized as timelines to track tasks, deadlines, and dependencies.
3.3 Project managers can define sprint cycles and assign sprint points.

Task Management Functionalities
4.1 Tasks can be created, assigned, and updated.
4.2 Tasks can have priority levels (e.g., high, medium, low).
4.4 Tasks can have due dates and automatic archive dates to prevent backlog clutter.
4.5 Tasks can have assigned users who receive notifications about approaching task deadlines via email or in-app alerts.

Workflow Automation Functionalities
5.1 Automated workflows are available for users to streamline task assignments and approvals.
5.3 Automatic task assignments based on predefined rules and conditions (eg, QA admin creates a task -> goes to QA team, creates a feedback meeting with development.
5.4 Workflow automation available for task dependencies to ensure that certain tasks cannot begin until prerequisites are met.

Collaboration and Communication Functionalities
8.1 Commenting on tasks and projects shall be supported.
8.2 @Mentions shall notify specific team members.
8.5 Project activity history shall be available to track changes.

Reporting and Analytics Functionalities

9.1 Real-time project progress reports shall be generated.

Priority 2 - Important

Reporting and Analytics Functionalities
9.5 Historical project analytics shall be available to track past performance.

User and Access Management
1.4
1.5 User activity, such as task completion and project edits, will be logged.

Organization-Level Functionalities
2.3 Organization's users may view an enterprise-wide project timeline that visualizes all active projects.
2.4 Organizations can align their goals and deadlines through organization-wide sprint planning.
2.5 Organizations will have real-time data synchronization across all projects and all views of projects.

Project Management Functionalities
3.4 Project Task dependencies will be automatically generated(e.g., a development task due on the 20th triggers a review task due on the 22nd).
3.5 Projects will have filters to switch between different views of the same project data (e.g., development view, product manager view, QA view).

Task Management Functionalities
4.3 Tasks can be categorized by department (e.g., Development, QA, Product).

Workflow Automation Functionalities
5.2 Automatic task assignments based on predefined rules and conditions (eg, QA admin creates a task -> goes to QA team, creates a feedback meeting with development.
5.5 Automation for recurring tasks and reminders for routine project activities.

Calendar and Scheduling Functionalities
6.1 Shared project calendar for tracking team meetings and deadlines.
6.3 Scheduling is available for PrM/PdMs to assign team meetings directly from the application.
6.4 The calendar shall automatically generate meeting reminders for all invited participants.

Client and Stakeholder Collaboration Functionalities
7.1 External clients and stakeholders will have custom views to view project progress with controlled access.
7.2 Clients shall view specific milestones and tasks, as defined by organization teams making them available.
7.3 Clients will be able to share feedback directly within Serial, by comments and flags on their view.

Collaboration and Communication Functionalities
8.3 File attachments on tasks shall be allowed.

Reporting and Analytics Functionalities
9.2 Workload distribution across team members shall be viewable by managers.
9.3 Sprint burndown charts for Agile projects shall be provided.
9.4 Report exporting in PDF and CSV formats shall be supported.

Customization and User Preferences Functionalities
10.1 Dashboard layout customization shall be supported.
10.3 Notification preferences shall be configurable.

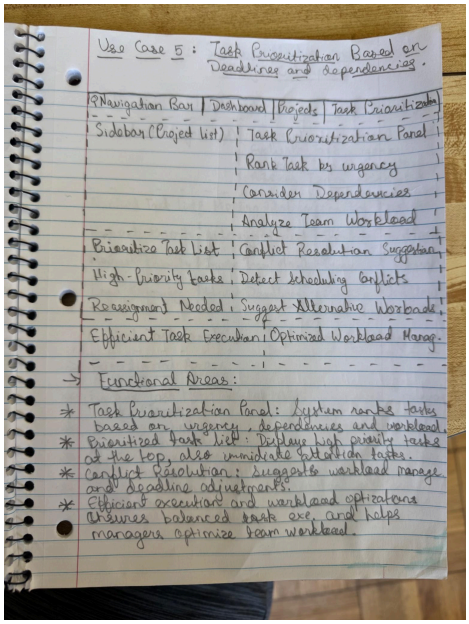
Priority 3 - Nice-to-Have

Calendar and Scheduling Functionalities
6.2 Calendar synchronization will be supported with external applications (Google Calendar, Outlook) via .ics file generation/import.
6.5 Calendar tooling shall enable users to propose, vote on, accept, and decline meeting times to improve scheduling efficiency.

Client and Stakeholder Collaboration Functionalities
7.4 Client views shall have functionality to be real-time updated, or show a client only “completed” Chunk.
7.5 Clients shall be able to real-time communicate with Team leads and Organization managers.

Collaboration and Communication Functionalities
8.4 A simple team chat system shall be provided for discussions.

Customization and User Preferences Functionalities
10.2 Light mode and dark mode themes shall be available.
10.4 Custom project template creation shall be supported.
10.5 Task color label selection shall be available.

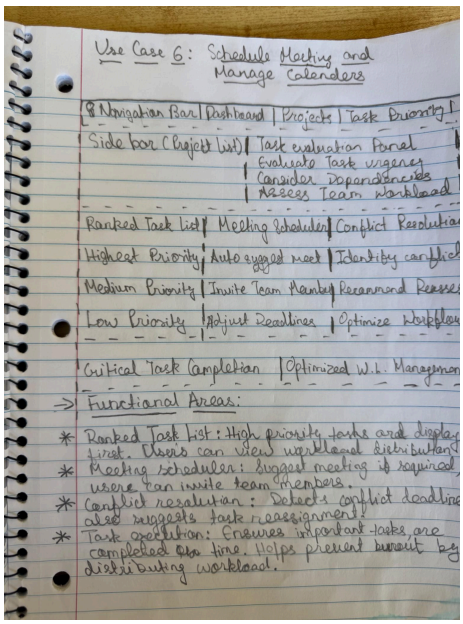


Use Case 5: Task Prioritization Based on Deadlines and Dependencies.

Navigation Bar	Dashboard	Projects	Task Prioritization
Sidebar (Project list)	Task Prioritization Panel	Rank Task by urgency	Consider Dependencies	Analyze Team Workload
Prioritize Task List	Conflict Resolution Suggestion
High-Priority Tasks	Detect Scheduling Conflicts
Reassignment Needed	Suggest Alternative Workloads
Efficient Task Execution	Optimized Workload Manag.

→ Functional Areas:

* Task Prioritization Panel: System ranks tasks based on urgency, dependencies and workload.
* Prioritized Task List: Displays high priority tasks at the top, also immediate attention tasks.
* Conflict Resolution: Suggests workload manage and deadline adjustments.
* Efficient execution and workload optimizations ensures balanced task exe, and helps managers optimize team workload.

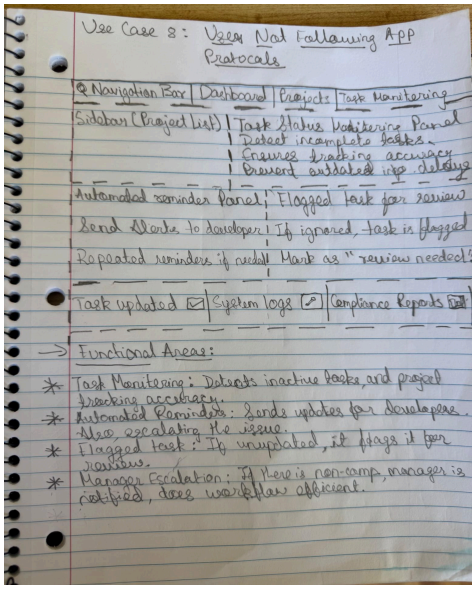


Use Case 6: Schedule Meeting and Manage Calendars

Navigation Bar	Dashboard	Projects	Task Priority
Side bar (Project List)	Task evaluation Panel	Evaluate Task urgency	Consider Dependencies	Assess Team Workload
Ranked Task List	Meeting Scheduler	Conflict Resolution
Highest Priority	Auto suggest meet	Identify conflict
Medium Priority	Invite Team Member	Recommend Reassi
Low Priority	Adjust Deadlines	Optimize Workflow
Critical Task Completion	Optimized W.L. Managmen

→ Functional Areas:

* Ranked Task list: High priority tasks and display first. Users can view workload distribution.
* Meeting scheduler: Suggest meeting if required, users can invite team members.
* Conflict resolution: Detects conflict deadline also suggests task reassignment.
* Task execution: Ensures important tasks are completed on time. Helps prevent burnout by distributing workload.

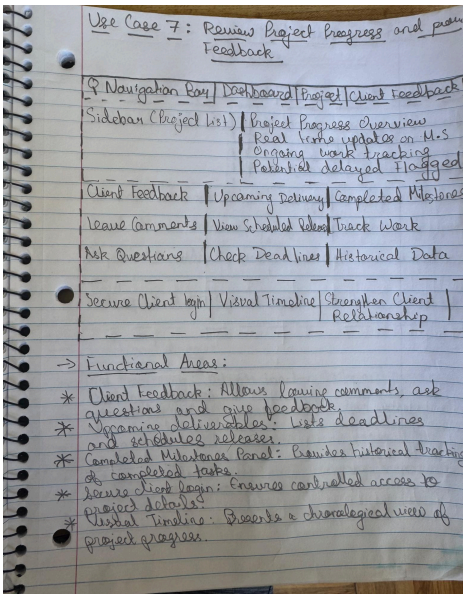


Use Case 8: Users Not Following App Protocols

Navigation Bar	Dashboard	Projects	Task Monitoring
Sidebar (Project List)	Task Status Monitoring Panel	Detect incomplete tasks.	Ensures tracking accuracy	Prevent outdated info delays
Automated reminder Panel	Flagged task for review
Send Alerts to developer	If ignored, task is flagged
Repeated reminders if needed	Mark as "review needed"
Task updated	System logs	Compliance Reports

→ Functional Areas:

* Task Monitoring: Detects inactive tasks, and project tracking accuracy.
* Automated Reminders: Sends updates for developers. Also escalating the issues.
* Flagged task: If unupdated, it flags it for reviews.
* Manager Escalation: If there is non-comp, manager is notified, does workflow efficient.



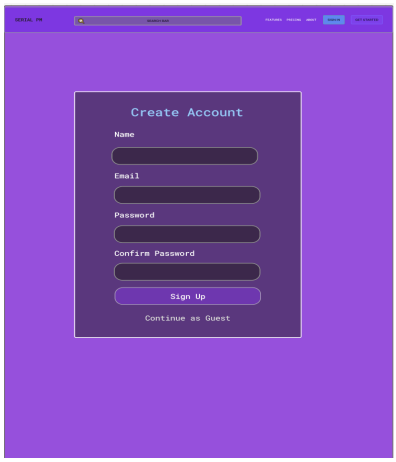
Use Case 7: Review Project Progress and pro Feedback

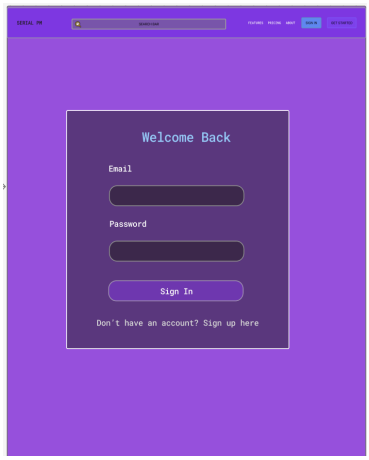
Navigation Bar	Dashboard	Project	Client Feedback
Sidebar (Project list)	Project Progress Overview	Real time updates on M.S	Ongoing work tracking	Potential delayed Flagged
Client Feedback	Upcoming Delivery	Completed Milestones
Leave Comments	View Scheduled Releases	Track Work
Ask Questions	Check Deadlines	Historical Data
Secure Client login	Visual Timeline	Strengthen Client Relationship

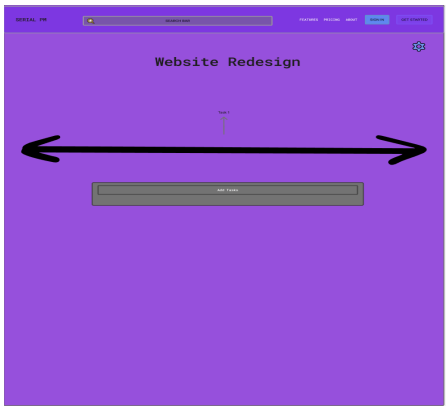
→ Functional Areas:

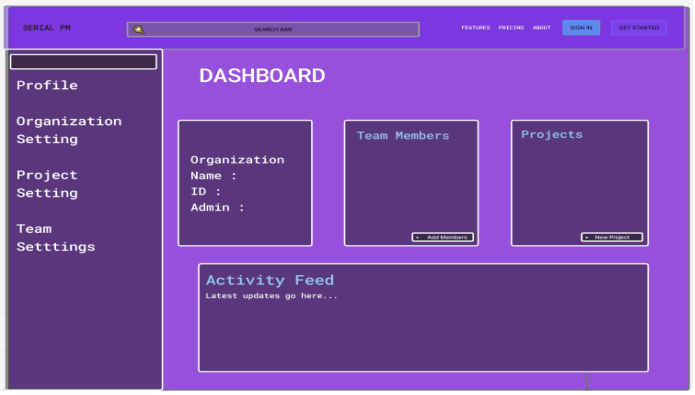
* Client Feedback: Allows leaving comments, ask questions and give feedback.
* Upcoming deliverables: Lists deadlines and schedules releases.
* Completed Milestones Panel: Provides historical tracking of completed tasks.
* Secure client login: Ensures controlled access to project details.
* Visual Timeline: Presents a chronological view of project progress.

5. UI/UX Wireframes:







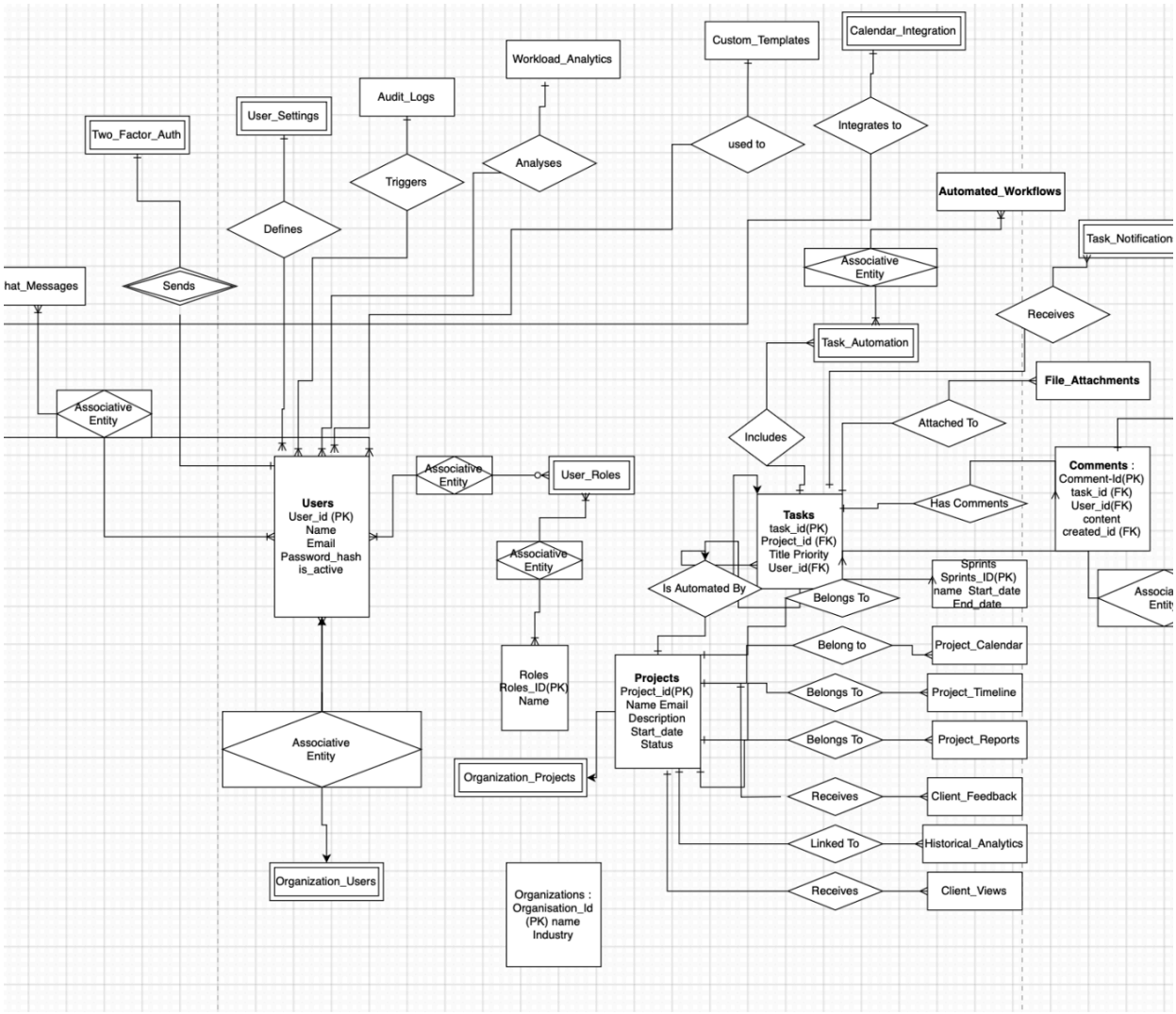


[Figma Wireframe]

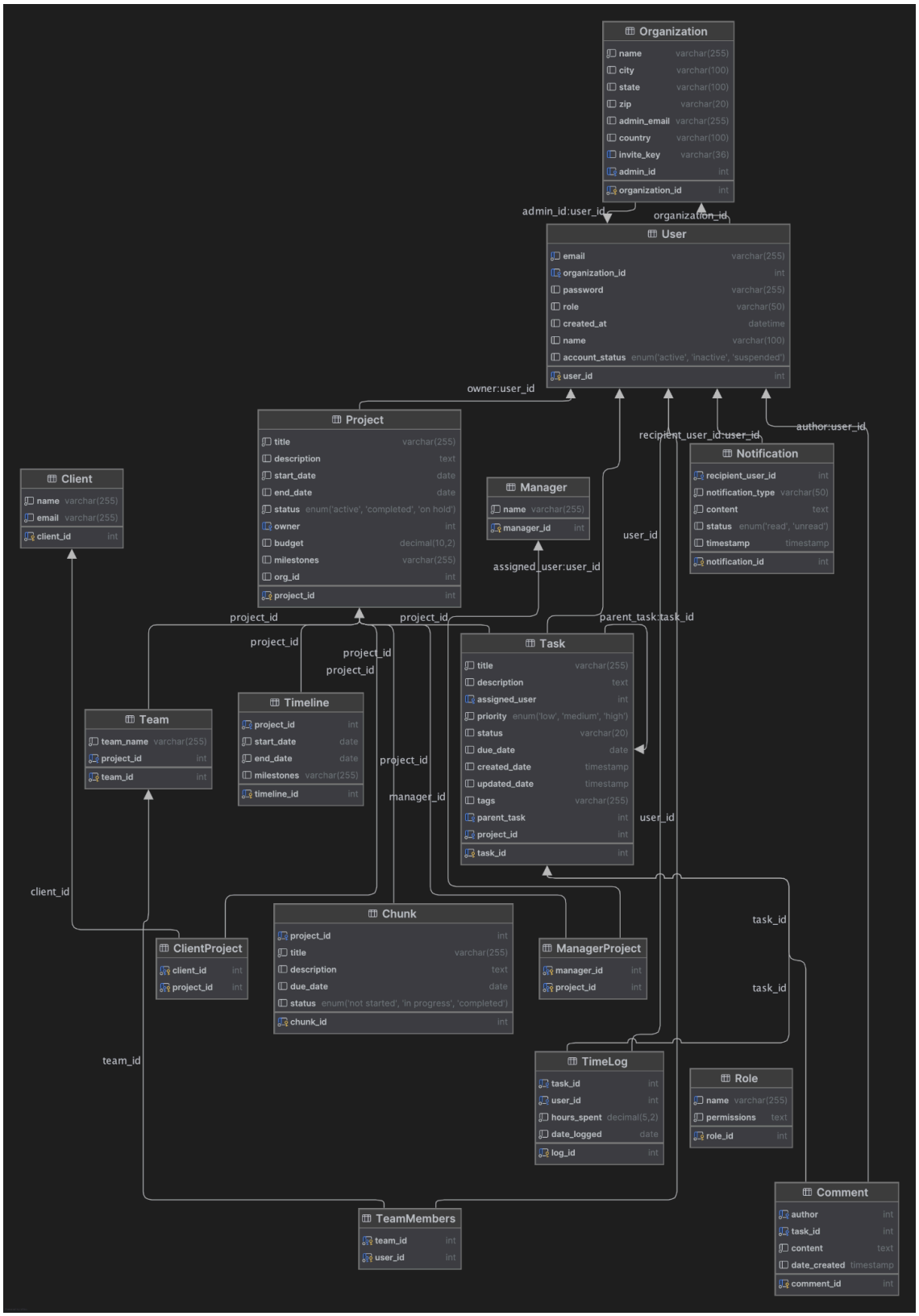
6. High Level System Design

6.1 High Level Database Architecture

Entity Relation diagram:

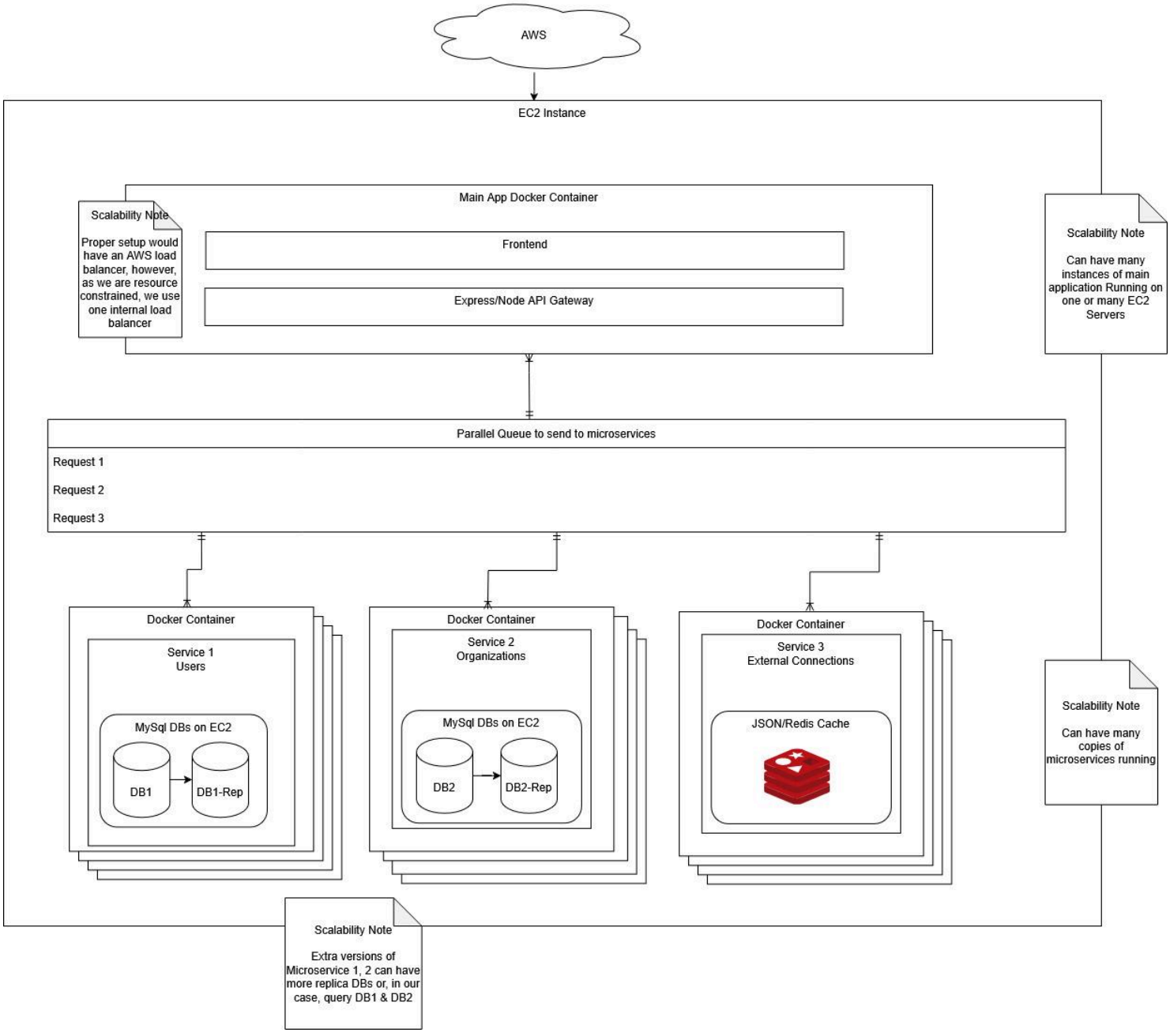


EER:


Organization
name varchar(255)
city varchar(100)
state varchar(100)
zip varchar(20)
admin_email varchar(255)
country varchar(100)
invite_key varchar(36)
admin_id int
organization_id int
admin_id:user_id
organization_id
User
email varchar(255)
organization_id int
password varchar(255)
role varchar(50)
created_at datetime
name varchar(100)
account_status enum('active', 'inactive', 'suspended')
user_id int
owner:user_id
author:user_id
recipient_user_id:user_id
Project
title varchar(255)
description text
start_date date
end_date date
status enum('active', 'completed', 'on hold')
owner int
budget decimal(10,2)
milestones varchar(255)
org_id int
project_id int
Notification
recipient_user_id int
notification_type varchar(50)
content text
status enum('read', 'unread')
timestamp timestamp
notification_id int
Client
name varchar(255)
email varchar(255)
client_id int
Manager
name varchar(255)
manager_id int
assigned_user:user_id
user_id
project_id
parent_task:task_id
Task
title varchar(255)
description text
assigned_user int
priority enum('low', 'medium', 'high')
status varchar(20)
due_date date
created_date timestamp
updated_date timestamp
tags varchar(255)
parent_task int
project_id int
task_id int
Team
team_name varchar(255)
project_id int
team_id int
Timeline
project_id int
start_date date
end_date date
milestones varchar(255)
timeline_id int
manager_id
client_id
Chunk
project_id int
title varchar(255)
description text
due_date date
status enum('not started', 'in progress', 'completed')
chunk_id int
ClientProject
client_id int
project_id int
ManagerProject
manager_id int
project_id int
task_id
team_id
TimeLog
task_id int
user_id int
hours_spent decimal(5,2)
date_logged date
log_id int
Role
name varchar(255)
permissions text
role_id int
Comment
author int
task_id int
content text
date_created timestamp
comment_id int
TeamMembers
team_id int
user_id int

6.2 Backend Architecture

Scalability Diagrams



Link to full size Scalability diagram (Google Drive/Draw.io):
https://drive.google.com/file/d/1K6o8ppVO0kDjtHlOqj91j6DidNiu8QAD/view?usp=drive_link

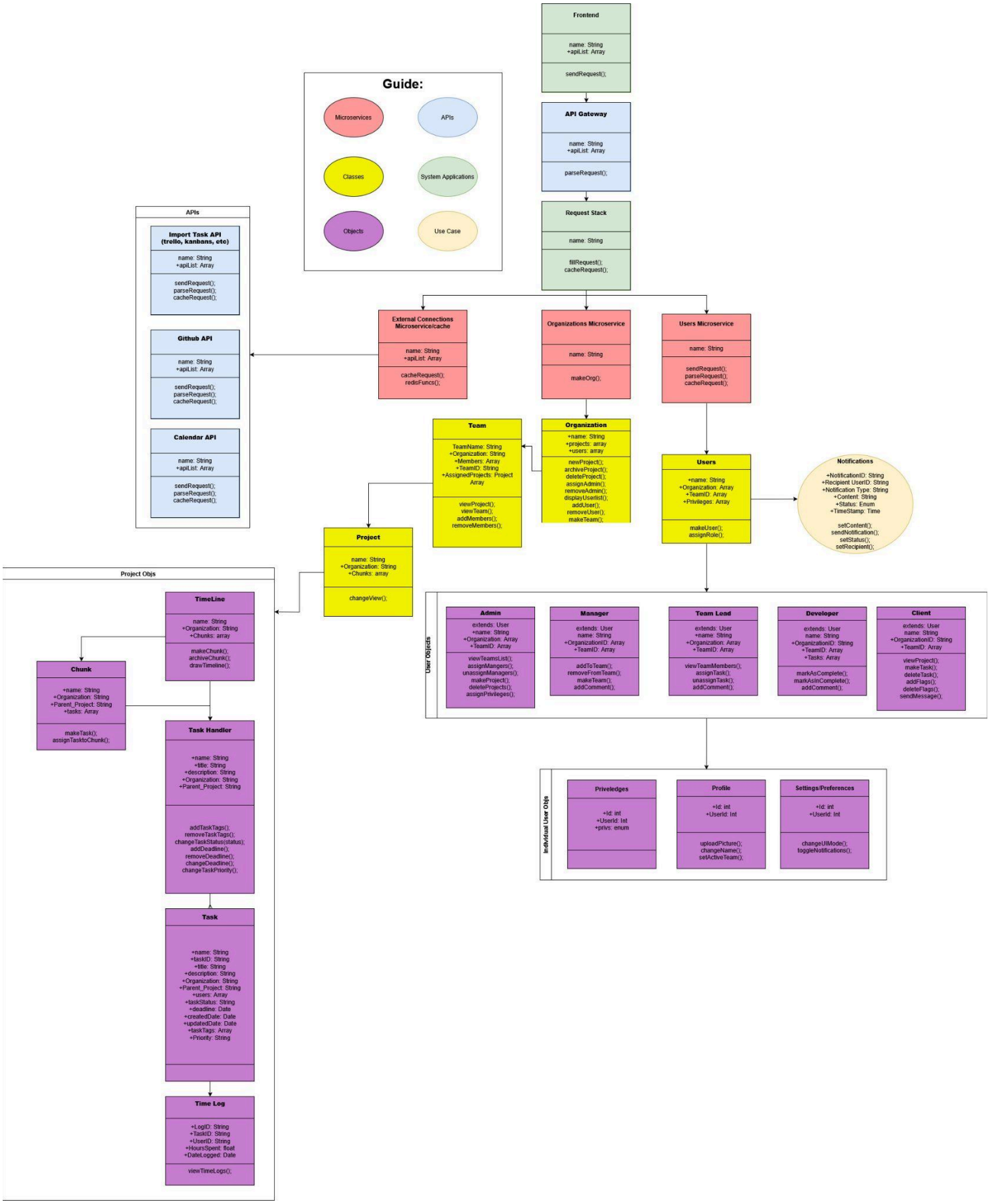
Architecture Summary

Serial's architecture is designed with scalability, ease of use, and minimal cost in mind. It utilized a microservice architecture managed by an Express API gateway for: users, organizations, and external connections. All instructions for these microservices are stored in a parallel queue - a stack-like piece of middleware that can ensure an instruction isn't skipped even if the service temporarily fails and has further request protection built in. We do load-balancing using Caddy, and cache many aspects of the application on the user-end and within Redis so that any short-term disconnections or processing isn't a disruption to a user.

Our parallel queue for requests to the microservices ensures some degree of fault tolerance, as requests are stored separately and repeatedly made until filled (within reasonable technical limit), we can allow users to work while a request finishes separate from them and that one bad packet doesn't ruin their request. Likewise, reliability is boosted by having databases tied to each microservice - no single error will hurt the entire application.

Each microservice and our main application & middleware are containerized using Docker, to reduce any potential deployment errors or lack of dependencies and standardize the operation such that it will work without a VM on any machine. So far, this looks like two different node applications: our backend, and the main vertical prototype logic. Our databases will be run as-is, but with replicas so that our middleware can make many requests to many replica versions of the database. These will be on one AWS server due to constraints of our project, but as it is only a series of MySQL connections, users with more robust infrastructure can benefit from the way this is designed and have copies and replicas of databases deployed wherever they need on any number of servers. Finally, Serial's architecture protects against potential erroneous infinite loops or accidental security breaks from bad actors: our API gateway middleware examines the plausibility of any request made, and will deny those that are unfeasible and not within the predefined limits of the middleware.

UML Class Diagram



Link to Full Resolution UML (Google Drive/Draw.io):
https://drive.google.com/file/d/1_Zk4aACFGBgaUj2zjUvdVl2biVd_qUJ9/view?usp=drive_link

UML Description

Our basic UML splits the application up into 6 different types of entities: APIs, Microservices, Classes, Objects, Use Cases, and System Applications. We have divided the core workings of the application into four essential aspects as described in our architecture summary, all pivoting around getting data to a “request stack” that is like a runtime stack for the interworking microservices and middleware:

- our two primary applications, the frontend and the API gateway in our backend, and
- The two types of things the API gateway communicates with via the request stack: microservices, and external APIs.

Both microservices utilize a variety of objects to efficiently do their main task (the management of Users and Tasks, respectively), and can have further refined objects attached to them that they supervise. Additionally, microservices can also have “use-cases” attached, which are classes with specific functions that interact at many levels of the application, such as the Notification system which needs to communicate back up to the API gateway to get between the Tasks and the Users.

7. List Of Team Contributions

Addendum, M3V2:
Thanks to Alison John & Ansh Patel for new ERD
Thanks to Victoria B. & Datagrip Diagrams for EER, document tweaks

Team Member	Actions (Checkpoint 1)	Actions (Checkpoint 2)	Rating
Victoria Barnett	Contributed to Backend architecture planning. Led system requirements review.	Document review, HP codebase bug fixing, corrected userstate errors. Tweaked projects to be owned by organizations in DB.	8/10
Alison John	Expanded database definitions, added attributes. Assisted system requirements review.	Tweaked database to fit HP requirements	7/10
Ansh Ankit Patel	Assisted system requirements review. Worked on figma wireframe. Main document reviewer/tech writer.	Lead Document review. Finalized figma wireframes.	8/10
Nidhey Patel	Assisted system requirements review. Created EER for high level system check.	Lead dashboard page major overhaul.	7/10
Pritham Singh Sandhu	Worked on Sign-in/sign-up flow, error validation. Assisted system requirements review. Worked on figma wireframe. Contributed to data definitions.	Added signup validation features, added guest signin/signup. Added toast notifications. Bugfixed for Horizontal prototype as frontend lead. Contributed to organizations rework.	9/10
Tushin Kulshreshtha	Primary on high level apis, main algorithms. Created	Primary on API route tweaks. Assisted in database tweaks for	9/10

	org creation component.Worked on figma wireframe.	HP to match vision. Dashboard page ui tweaks.	
Yikang Xu	Assisted system requirements review, assisted backend review meeting.	Primary on organization page rework.	7/10

SW Engineering

CSC648-848-05 Spring 2025

Serial

Team 02

- Victoria Barnett rbarnett@sfsu.edu Team Lead
- Tushin Kulshreshtha tkulshreshtha@sfsu.edu Back End Lead
- Ansh Patel apatel18@sfsu.edu Front End Lead
- Pritham Sandhu psandhu3@sfsu.edu Software Architect
- Alison John ajohn3@sfsu.edu Database Administrator
- Nidhey Patel npatel20@sfsu.edu Technical writer
- Yikang Xu yxu26@sfsu.edu Github Master

Milestone - 4

05/08/2025

3. Project Summary

Product Name:

Serial

Final P1 & P2 Functional Requirements:

User and Access Management	- Users can register and log in using a username and password. - Users can reset their password. - Users can invite new team members to projects, be promoted to Admins, and manage access for other users. - Users can update their profile information. - User activity, such as task completion and project edits, will be logged.
Organization-Level Functionalities	- Organizations can create and manage multiple projects. - Organizations have role-based access control, with permissions defined for administrators, developers, and clients. - Organization's users may view an enterprise-wide project timeline that visualizes all active projects. - Organizations can align their goals and deadlines through organization-wide sprint planning. - Organizations will have real-time data synchronization across all projects and multiple views of projects.
Project Management Functionalities	- Users can create, update, and track project roadmaps and tasks. - 3.5 Projects will have filters to switch between different views of the same project data (e.g., development view, client view).

	- Projects can be visualized as timelines to track tasks, deadlines, and dependencies.
Task Management Functionalities	- Tasks can be created, assigned, and updated. - Tasks can have priority levels (e.g., high, medium, low). - Tasks can have due dates. - Tasks can have assigned users who receive notifications about approaching task deadlines via in-app alerts. - Tasks can be categorized by department (e.g., Development, QA, Product).
Collaboration and Communication Functionalities	- Messaging between organization users shall be supported. - @Mentions shall notify specific team members. - Task activity history shall be available to track changes.
Client and Stakeholder Collaboration Functionalities	- External clients and stakeholders will have custom views to view project progress with controlled access. - Clients shall view specific milestones and tasks, as defined by organization teams making them available.

Unique Features:

SerialPMs most unique feature is our visual timeline for work – rather than a Kanban board or sprints, you see tasks like a calendar along a timeline of work, perfect for elevating deadlines and reducing confusion around priorities. Our next unique feature is we aren't attempting to corner one segment of a product's specialists, SerialPM is interested in becoming the whole of a businesses project management structure – for it's developers, for it's administrators, for it's project managers, and for it's clients. This flexibility for role allows us to eat into the space of what may be 2-3 competitors being paid for by a business, their organizational calendar, and

some internal tooling. All of this combines into an incredible value proposition for a business interested in staying thin.

Deployment URL:

https://serialpm.tech

4. Usability Test Plan

Test Objectives

The objective of this usability test is to evaluate the usability, efficiency, and user satisfaction of five critical features in a project management platform tailored for collaborative software teams. These features were selected for their importance in supporting productivity, transparency, and cross-functional collaboration. By observing real users interacting with each feature, we aim to assess how intuitively they perform key tasks, how effectively the system supports these workflows, and how satisfied users feel throughout the process. The results will help inform interface refinements, reduce friction in the user experience, and ensure the product aligns with real-world user needs in high-stakes project management environments.

The test will focus on:

1. Creating and managing a project timeline
2. Automated cross-team task synchronization
3. Executive visual project overview
4. GitHub integration for real-time updates
5. System response when users don't follow task protocol

Test Description

System Setup

The usability test will be conducted using a browser-accessible prototype hosted at:
URL: `https://serialpm.com`

The system will be deployed in a test environment pre-loaded with sample data including dummy projects, tasks, teams, and GitHub commit history. Email notifications and automation workflows will be simulated using test accounts and mock services.

Note: Some features (e.g., task synchronization and GitHub integration) are simulated during testing due to database and automation limitations. Facilitators will manually trigger updates or show mock results where needed.

Starting Point

Participants will start from the project dashboard after logging in with credentials provided to them. They will be given a brief scenario for each function, describing their role and what they need to do. No prior tutorial or training will be given, to better evaluate natural usability.

---

1. Project Timeline Management

Test Description:
Participants are given a scenario where they are assigned as a team lead responsible for planning a sprint. They must create new tasks, assign deadlines, adjust the timeline view (e.g., switching between weeks and months), and mark some tasks as in progress or completed. No instructions are given beyond the goal.
Objective: Evaluate how intuitively users can manipulate the timeline and track progress using the interface.

---

2. Automated Cross-Team Task Synchronization

Test Description:
Participants act as a developer finishing a feature. Upon marking a development task as completed, they are told to observe whether the QA task appears automatically. The facilitator simulates backend automation and explains the expected real-world effect.
Objective: Test users' understanding of automation flow between teams and whether the hand-off feels seamless and logical.

---

3. Executive Visual Project Overview

Test Description:
Users take the role of a product manager reviewing overall project health. From the dashboard, they must identify current progress, delays, and key milestones across teams.
Objective: Assess how clearly the interface communicates high-level project status, including risks and completion forecasts.

---

4. GitHub Integration for Real-Time Updates

Test Description:
Participants simulate a developer workflow where a code commit or pull request should trigger an update to task status. The system displays a mock GitHub action and corresponding task update.
Objective: Evaluate if users understand the connection between GitHub activity and project tracking, and if the interface reflects this accurately and reliably.

---

5. Task Protocol Enforcement

Test Description:
Participants are instructed to submit task progress without updating the task status. The system is expected to prompt the user with a warning or flag the task. Users must respond to the prompt and correct the issue.
Objective: Test how well the system enforces compliance with update protocols and how easy it is for users to resolve omissions.

Intended Users

Participants will include 5–8 individuals who are not involved in the development of the platform but are similar to the target audience:

- Developers
- QA Testers
- Product Managers
- Executives (project stakeholders)

Each participant will test all five features to ensure consistency across results.

Effectiveness Table

Function	Success Criteria	Success Rate (% of users completing task)

Project Timeline Management	User can create and label tasks, adjust timeline view, and mark progress	***83%*** *(5 out of 6 completed)*
Automated Task Synchronization	QA task is auto-assigned after dev task is marked completed	***Simulated*** *(Confirmed flow)*
Executive Project Overview	User finds key milestones and project health via dashboard	***100%*** *(All users succeeded)*
GitHub Integration	User sees task status updated after commit or PR	***Simulated*** *(Mock commit used)*
Task Protocol Enforcement	User receives prompt for unmarked task, and system flags it for review	*83% (5 out of 6 completed)*

Efficiency Table

Function	Expected Time to Complete (min)	Actual Avg. Time Taken	Observations
Project Timeline Management	3	3.5	Some users took extra time to locate the "edit task" icon
Automated Task Synchronization	2	2.2 (simulated)	Simulated by facilitator triggering QA task creation after Dev task
Executive Project Overview	2	1.6	Users found dashboard layout intuitive
GitHub Integration	2	2.5 (simulated)	Commit effect simulated; users understood the interaction clearly
Task Protocol Enforcement	2	2.1	Prompt triggered after user submitted without status update

User Satisfaction (Likert Questionnaire)

Use a 5-point Likert scale (Strongly Disagree to Strongly Agree). Each function gets 3 statements:

1. Project Timeline Management

- The timeline view was easy to understand and navigate. 4.2
- Creating and updating tasks on the timeline felt intuitive. 3.8
- I was confident that the timeline reflected the project's real status. 4.0

2. Automated Task Synchronization

- I felt the system reduced the need for manual coordination. 4.5
- I clearly understood how tasks were passed between teams. 4.2
- Notifications were timely and useful for keeping me informed. 4.3

3. Executive Visual Project Overview

- The project overview gave me a clear understanding of progress. 4.7
- The layout of the overview dashboard was easy to scan quickly. 4.6
- I could easily identify risks or delays from the visual indicators. 4.5

4. GitHub Integration

- I liked how the task system updated automatically based on GitHub actions. 0
- The integration saved me from switching between tools frequently.0
- The sync between code commits and tasks was accurate and reliable. 0

5. Task Protocol Enforcement

- I appreciated the reminders when I forgot to update my task status. 4.4
- The system made it easy to resolve uncompleted or outdated tasks. 4.0
- This feature helped keep the project on track and accountable. 4.6

5. QA Test Plan

1. Performance – Application Response Time (<5s)

Test Objectives:
Ensure that all key pages and features in the SerialPM web application load and respond in under 5 seconds under normal conditions.

HW and SW Setup:

- Device: Laptop (8GB RAM, SSD, i5 processor)
- OS: Windows 11
- Browsers: Chrome 123, Firefox 124, Edge 122
- URL: https://serialpm.tech

Feature to be Tested:
Initial dashboard load, project timeline access, and task editing.

QA Test Cases:

Test #	Title	Task Description	Input	Expected Output	Results
1.1	Dashboard Load Time	Measure time from login to dashboard	Login credentials	Load complete < 5s	PASS
1.2	Project Timeline Access	Load timeline for existing project	Click “View Timeline	Timeline view loads <5s	PASS
1.3	Edit Task Modal	Open and edit a task	Click “Edit Task”	Modal opens and responds <5s	PASS

Testing Execution:
✅ All supported browsers passed test cases under normal network load (50Mbps).

2. Reliability – Data Consistency During Transactions

Test Objectives:
Verify that task creation, updates, and deletions are consistently reflected across users and modules without data loss or corruption.

HW and SW Setup:

- Device: Laptop (8GB RAM, SSD, i5 processor)
- OS: Windows 11
- Browsers: Chrome 123, Firefox 124, Edge 122
- URL: https://serialpm.tech

Feature to be Tested:
Task management system: creation, update, deletion.

QA Test Cases:

Test #	Title	Task Description	Input	Expected Output	Results
2.1	Create Task Consistency	Create a task, reload page	New task data	Task persists after reload	PASS
2.2	Update Task Sync	Update task status, verify	Change status to “Done”	Updated status visible to all users	PASS
2.3	Delete Task Propagation	Delete task, confirm removal	Click “Delete Task”	Task deleted across all views/modules	PASS

Testing Execution:
✅ Data integrity maintained across Chrome, Firefox, Edge. Simultaneous access by multiple test accounts yielded consistent results.

3. Usability – UI/UX Element Latency (<5s)

Test Objectives:
Validate that UI components (dropdowns, modals, menus) respond within 5 seconds, ensuring smooth interactions.

HW and SW Setup:

- Device: Laptop (8GB RAM, SSD, i5 processor)
- OS: Windows 11
- Browsers: Chrome 123, Firefox 124, Edge 122
- URL: https://serialpm.tech

Feature to be Tested:
Dashboard menu, timeline interactions, notifications panel.

QA Test Cases:

Test #	Title	Task Description	Input	Expected Output	Results
3.1	Open home page	Click on home button	Click “SerialPM” icon	Back to home page	PASS
3.2	Expand Timeline Task	Expand a task for details	Click on timeline task box	Detail view expands in <2s	PASS
3.3	View Pricing info	Display the pricing for the application	Click “Pricing” icon	Pricing is displayed in <2s	PASS

Testing Execution:
✅ All tested browsers responded smoothly, latency remained below 2.5s in all cases.

4. Efficiency – Memory Usage (<4GB Client-side)

Test Objectives:
Ensure that SerialPM does not exceed 4GB memory usage during normal operation.

HW and SW Setup:

- Device: Laptop (8GB RAM, SSD, i5 processor)
- OS: Windows 11
- Browsers: Chrome 123, Firefox 124, Edge 122
- URL: https://serialpm.tech
- Browser memory profiling tools (DevTools → Performance tab)

Feature to be Tested:
Typical user session with project navigation, editing, and timeline usage.

QA Test Cases:

Test #	Title	Task Description	Input	Expected Output	Results
4.1	Baseline Load Memory	Log in and idle at dashboard	None	Memory < 500MB	PASS
4.2	Active Use Memory	Navigate, edit tasks, open views	Simulated usage for 10min	Memory < 3.5GB	PASS
4.3	Stress Memory Cap	Load 10 projects, switch frequently	Rapid view switching	Memory < 4GB, no crash/hang	PASS

Testing Execution:
✅ No instance exceeded 3.7GB across browsers even under simulated load.

5. Coding Standards – GitHub Version Control

Test Objectives:
Verify that all code commits follow proper branching, commit message conventions, and pull request (PR) reviews before merge.

HW and SW Setup:

- Device: Laptop (8GB RAM, SSD, i5 processor)
- OS: Windows 11
- Browsers: Chrome 123, Firefox 124, Edge 122
- URL: https://serialpm.tech
- Git CLI & GitHub web UI

Feature to be Tested:
Version control workflow adherence.

QA Test Cases:

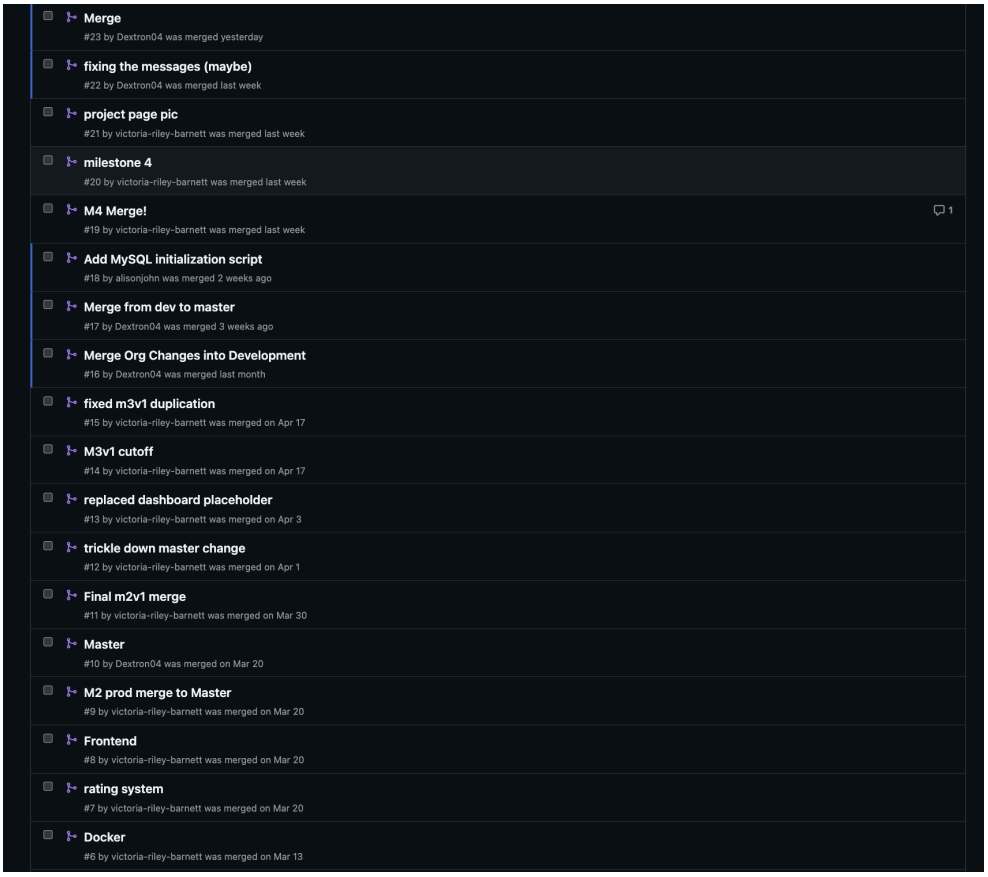
Test #	Title	Task Description	Input	Expected Output	Results
5.1	Commit Format Compliance	Push a new commit	Commit message	Matches required format (e.g. feat/fix)	PASS
5.2	Branch Policy Enforcement	Try to push to main branch	Direct push	Blocked, PR required	PASS
5.3	PR Review Process	Submit PR	Code changes	Requires at least one reviewer approval	PASS

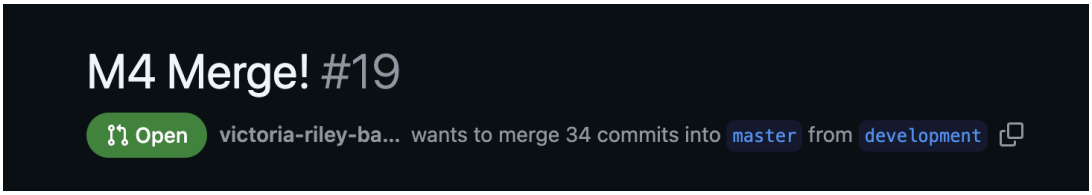
Testing Execution:
✅ All GitHub policies enforced as expected. PR checks, reviews, and commit standards confirmed.

6. Code Review

- **Coding Standards:**
  - Our coding practices were not too advanced, but ended up working okay. First of all, we often combined for pair coding, which helped everyone become familiar with the stack:

  - Next, our Github Manager or Team Lead reviewed commits before pull requesting them into the Master branch, and all commits went into “Feature” branches, or the development branch before being merged:





- Finally, we extensively documented our code and started out or project with introducing ourselves to our technology stack, catching us all up as much as possible on the tools we'd need for success. Most functions and api routes have comments.
- While we didn't extensively discuss in PR comments, we did extensively discuss changes between versions via Discord between big merges to plan the release and coordinate changes:

@Victoria Barnett did you commit this somewhere? Can play with it some more

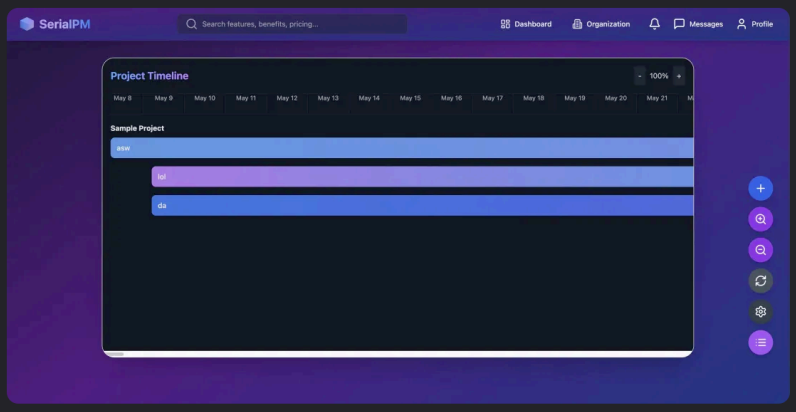
11:12 PM Tushin Kulshreshtha I'll do that right now I have some progress its not much tho

Tushin Kulshreshtha Did you add the network diagram in m3v2

May 8, 2025

12:04 AM Tushin Kulshreshtha I just added mine on top

12:35 AM Tushin Kulshreshtha

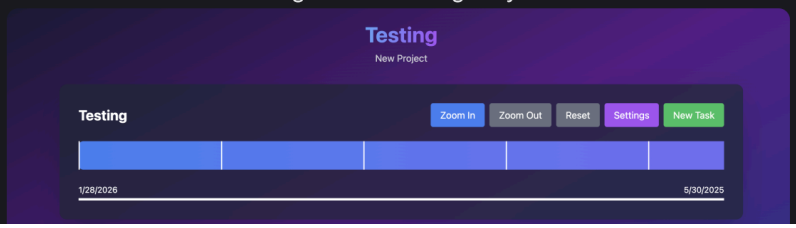


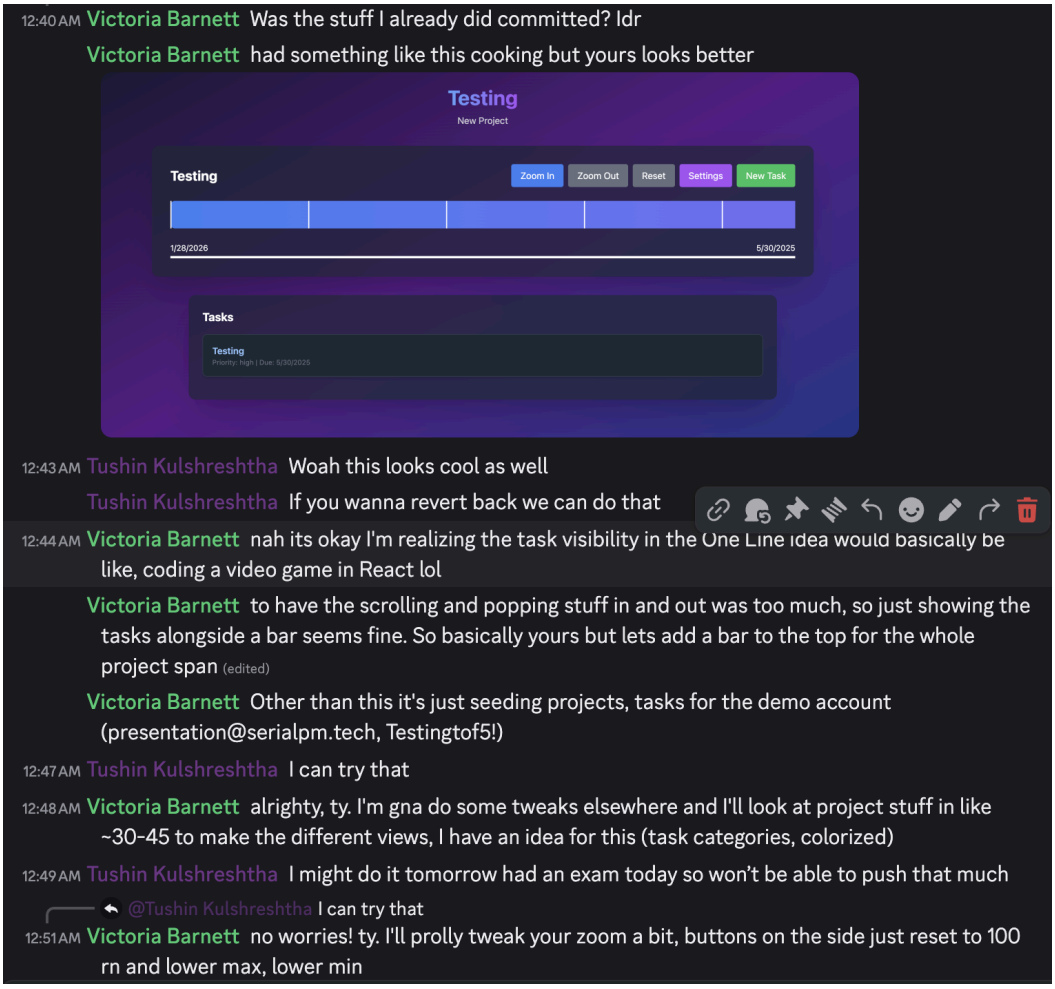
12:39 AM Victoria Barnett Sweet I'll pull

@Tushin Kulshreshtha *Click to see attachment*

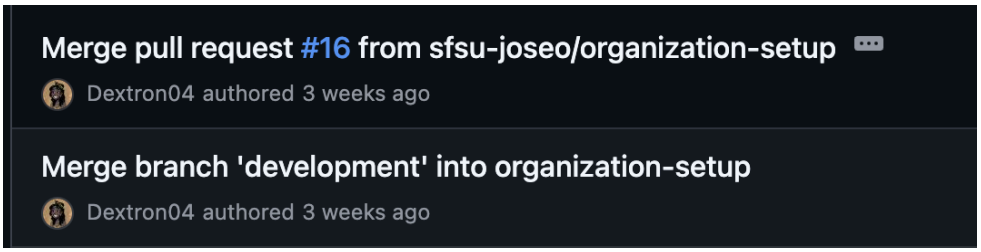
12:40 AM Victoria Barnett Was the stuff I already did committed? Idr

Victoria Barnett had something like this cooking but yours looks better

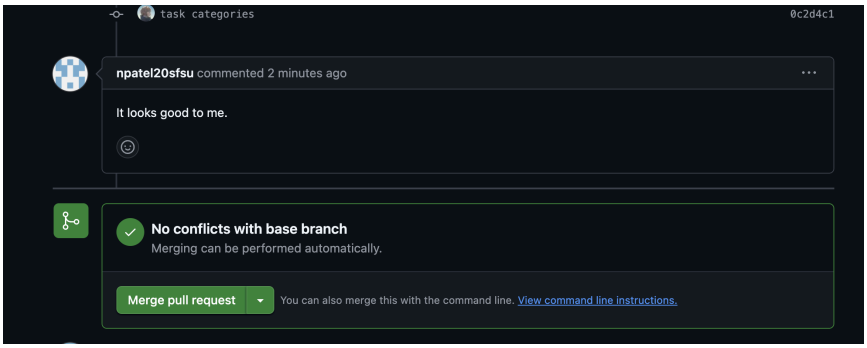




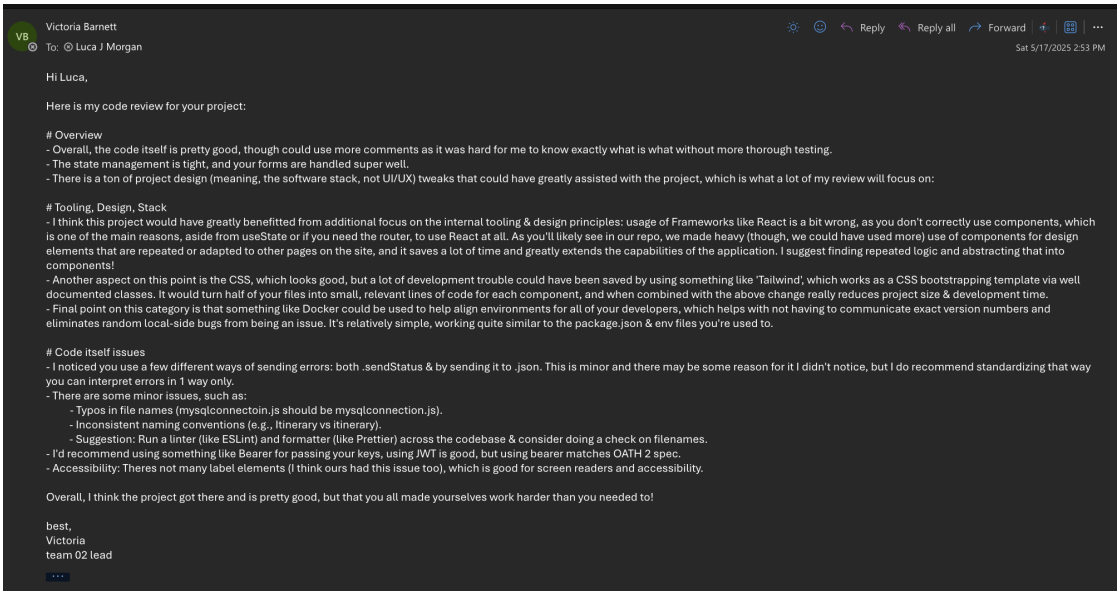
- **GitHub Code Review:**
  - Some example merges:

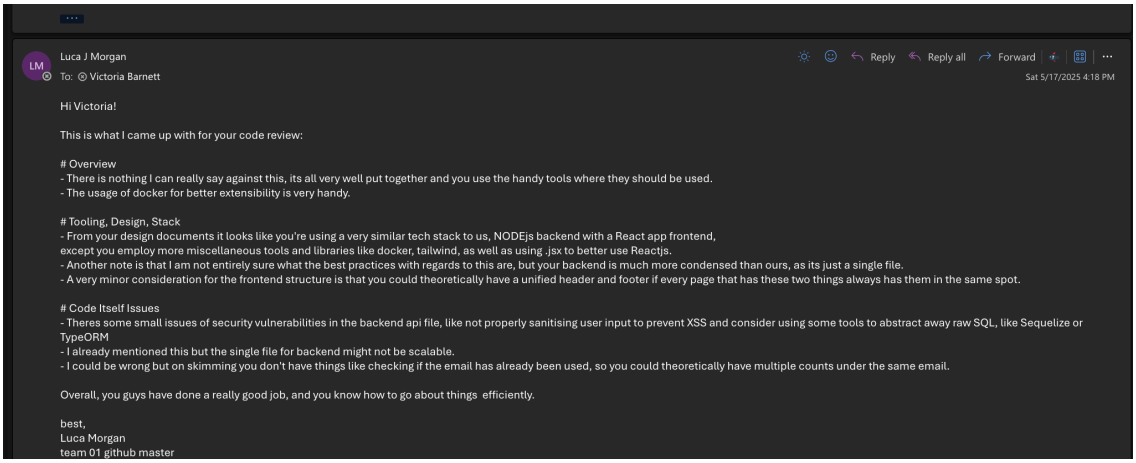


- & inside a Pull Request, for major updates another user tested code on their machine before the final merge:



- **External Code Review:**
  - We conducted an external Code review with Team 1, here are the comments from both teams:





  - Transcript:

Hi Luca,

Here is my code review for your project:

Overview
- Overall, the code itself is pretty good, though could use more comments as it was hard for me to know exactly what is what without more thorough testing.
- The state management is tight, and your forms are handled super well.
- There is a ton of project design (meaning, the software stack, not UI/UX) tweaks that could have greatly assisted with the project, which is what a lot of my review will focus on:

Tooling, Design, Stack
- I think this project would have greatly benefitted from additional focus on the internal tooling & design principles: usage of Frameworks like React is a bit wrong, as you don't correctly use components, which is one of the main reasons, aside from useState or if you need the router, to use React at all. As you'll likely see in our repo, we made heavy (though, we could have used more) use of components for design elements that are repeated or adapted to other pages on the site, and it saves a lot of time and greatly extends the capabilities of the application. I suggest finding repeated logic and abstracting that into components!
- Another aspect on this point is the CSS, which looks good, but a lot of development trouble could have been saved by using something like *'Tailwind'*, which works as a CSS bootstrapping template via well documented classes. It would turn half of your files into small, relevant lines of code for each component, and when combined with the above change really reduces project size & development time.
- Final point on this category is that something like Docker could be used to help align environments for all of your developers, which helps with not having to communicate exact version numbers and eliminates random local-side bugs from being an issue. It's relatively simple, working quite similar to the package.json & env files you're used to.

Code itself issues

- I noticed you use a few different ways of sending errors: both .sendStatus & by sending it to .json. This is minor and there may be some reason for it I didn't notice, but I do recommend standardizing that way you can interpret errors in 1 way only.
- There are some minor issues, such as:
    - Typos in file names (mysqlconnectoin.js should be mysqlconnection.js).
Inconsistent naming conventions (e.g., Itinerary vs itinerary).
Suggestion: Run a linter (like ESLint) and formatter (like Prettier) across the codebase & consider doing a check on filenames.
- I'd recommend using something like Bearer for passing your keys, using JWT is good, but using bearer matches OATH 2 spec.
- Accessibility: Theres not many label elements (I think ours had this issue too), which is good for screen readers and accessibility.

Overall, I think the project got there and is pretty good, but that you all made yourselves work harder than you needed to!

best,
Victoria
team 02 lead

Received:

Hi Victoria!

This is what I came up with for your code review:

Overview
- There is nothing I can really say against this, its all very well put together and you use the handy tools where they should be used.
- The usage of docker for better extensibility is very handy.

Tooling, Design, Stack
- From your design documents it looks like you're using a very similar tech stack to us, NODEjs backend with a React app frontend, except you employ more miscellaneous tools and libraries like docker, tailwind, as well as using .jsx to better use Reactjs.

```
- Another note is that I am not entirely sure what the best practices with regards to this are, but your backend is much more condensed than ours, as its just a single file.
- A very minor consideration for the frontend structure is that you could theoretically have a unified header and footer if every page that has these two things always has them in the same spot.

# Code Itself Issues
- Theres some small issues of security vulnerabilities in the backend api file, like not properly sanitising user input to prevent XSS and consider using some tools to abstract away raw SQL, like Sequelize or TypeORM
- I already mentioned this but the single file for backend might not be scalable.
- I could be wrong but on skimming you don't have things like checking if the email has already been used, so you could theoretically have multiple counts under the same email.

Overall, you guys have done a really good job, and you know how to go about things  efficiently.

best,
Luca Morgan
team 01 github master
```

7. Self-Check on Security Practices

- **Major Assets:**
    - **Source codes in GitHub**
        - Frontend elements
            - React components and webpages
        - API endpoints with token
            - JWT token generation, user creation, fetch organization, etc.
        - Database credentials and scripts
            - With SSH pem key and instructions to access the hosting server and databases

        - Public files
        - Libraries and package information

    - **Infrastructure**
        - Deploying version of the application on an Amazon EC2 server
        - Database instances containing user and application data.

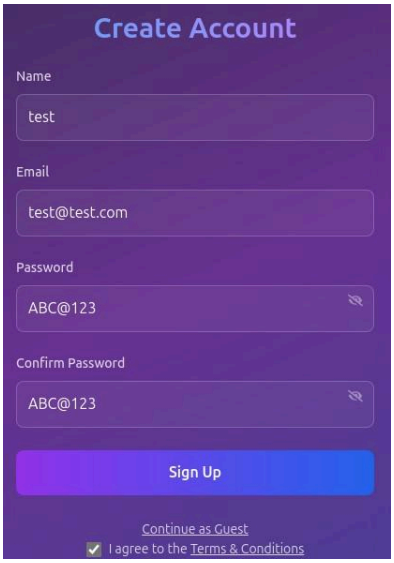
    - **Data**
        - User-specific: email, password, full name, roles, etc
        - Messages sent and received by users.
        - Application-specific: Organizations, Projects, Tasks, Timelines, and Team members.

- **Password Encryption:**
  - The password encryption occurs when an API call happens for a user creation:

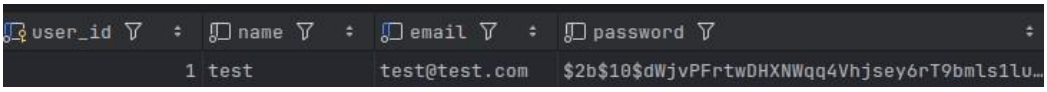
```
try {
  const hashedPassword = await bcrypt.hash(password, 10);
  const role = "client"; // Default role
  const account_status = "active"; // Default account status
  const organization_id = null;

  const [result] = await db.promise().query(
    `INSERT INTO User (name, email, password, role, account_status, organization_id)
       VALUES (?, ?, ?, ?, ?, ?)`,
    [name, email, hashedPassword, role, account_status, organization_id]
  );
```

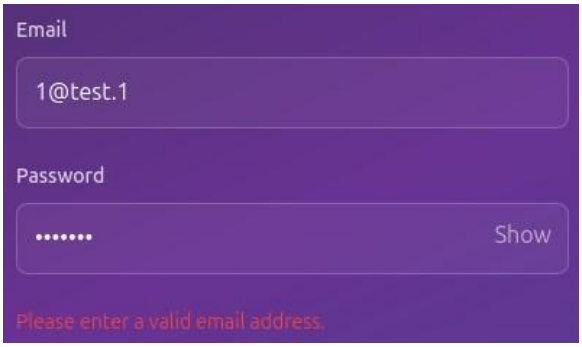
  - The passwords users enter at registration are hashed using the Bcrypt algorithm before inserting it into the database.



  - After success, the password is stored in the table *User* of the database, along with other user-specific information.



user_id	name	email	password
1	test	test@test.com	$2b$10$dWjvPFrtwDHXNWqq4Vhjsey6rT9bmls1lu…

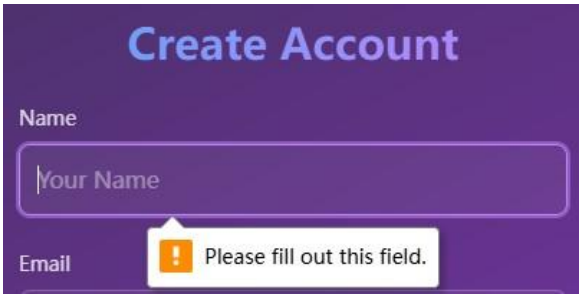
- # Input Validation:
  - **SignIn page**
    - Email
      - The input must be non-empty.
      - Input format: Non-whitespace characters + a single @ symbol + non-whitespace characters + a single period + non-whitespace characters.
      - The completed email must exist in the database.
      - 

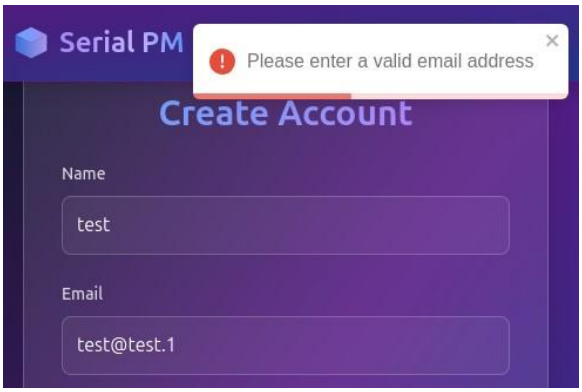
    - Password
      - The input must be non-empty.
      - The input must have a length > 6 characters.
      - The completed password must match the one associated with the email in the database.
      - 

    - Example:

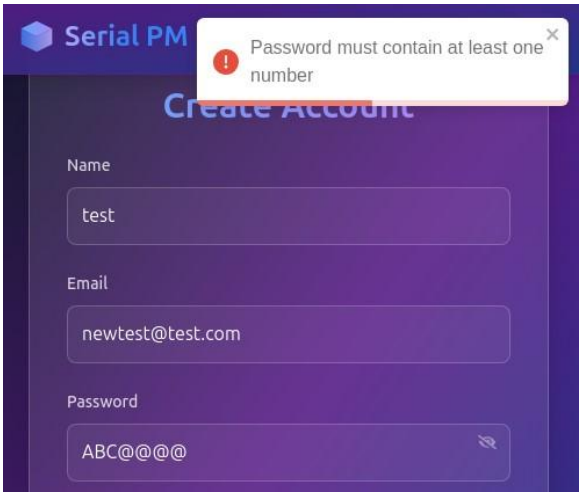
      ```
      // Check email format with regex
      if (!/\S+@\S+\.\S+/.test(formData.email)) {
          setError("Please enter a valid email address.");
      return false;
      }
      ```

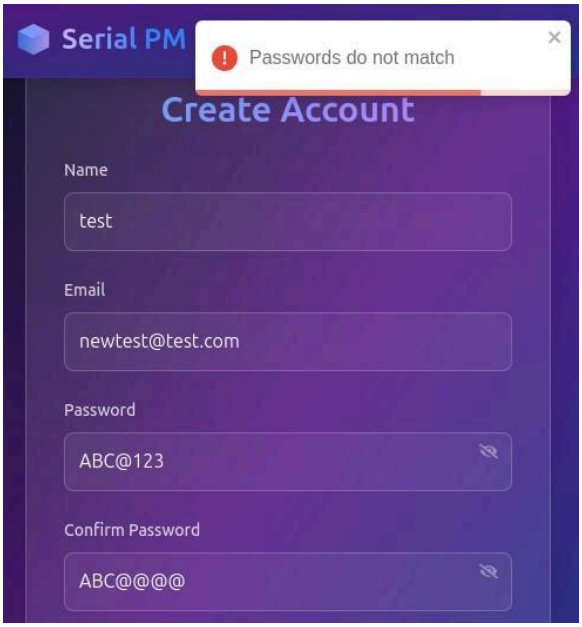
  - **SignUp page**
    - Name
      - The input must be non-empty.

- 

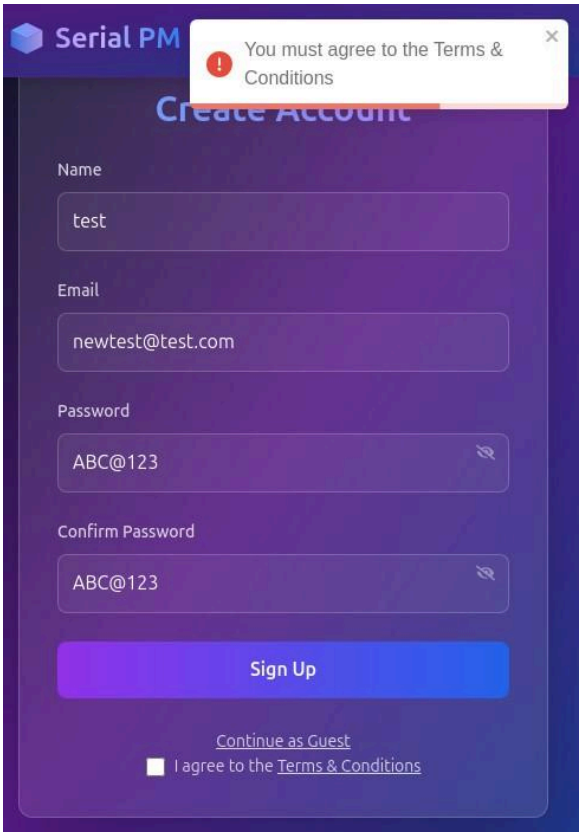
- Email
  - The input must be non-empty.
  - Input format: A string of digits, period, underscore, hyphen, upper and lower case letters + a single @ symbol + a string of digits, underscore, hyphen, upper and lower case letters + a single period + a string of upper and lower case letters.
  - The completed email must not be a duplicate of one in the database.
  - 

- Password
  - The input must be non-empty.
  - The input must contain a digit.
  - The input must contain a special character from the following: !@#$%^&*(),.?":{}|<>.
  - The input must have a length > 6 characters.

- 

- Confirm Password
  - The input must match the input in the Password field.
  - 

- Terms & Conditions
  - A Button that the users must click on it in order to register a new account.

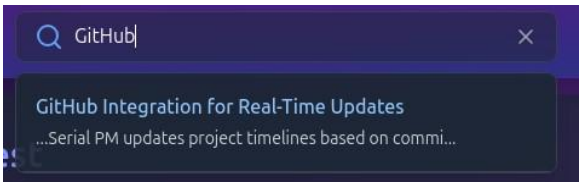


- Example:

```
// Ensure password length
if (password.length < 6) {
  setError("Password must be at least 6 characters long");
return false;
}
```

- **Search Bar component**
  - Input
    - Is Trimmed to remove whitespace.
    - Search queries will not occur with less than 2 characters.
    - Search query stops when the user has changed the input within 300 ms using debounce.
    - Check for API responses from queries to the database before showing the results.

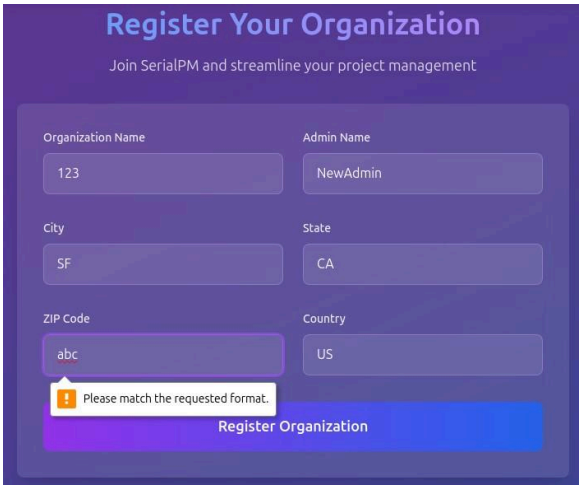
- Results
  - A mouse click navigates the user to the corresponding project.
  - Clicking outside the result list closes it.



- Example:

```
debounce(async (query) => {
   if (query.trim().length > 2) {
      // API call to fetch data
   }
}, 300)
```

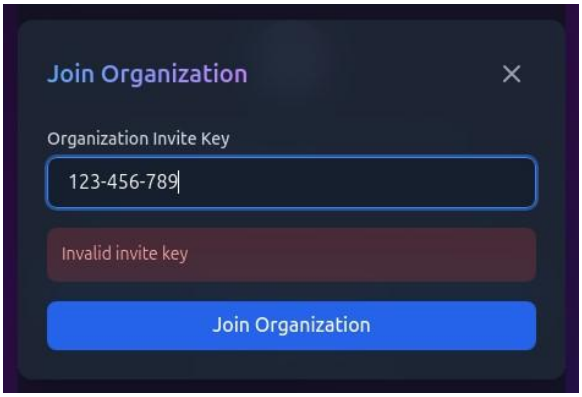
- **Organization Creation page**
  - It can only be accessed by already logged-in users.
  - All fields must be non-empty.
  - ZIP code
    - Input format: A string of exactly 5 digits + a hyphen + an optional string of exactly 4 digits.

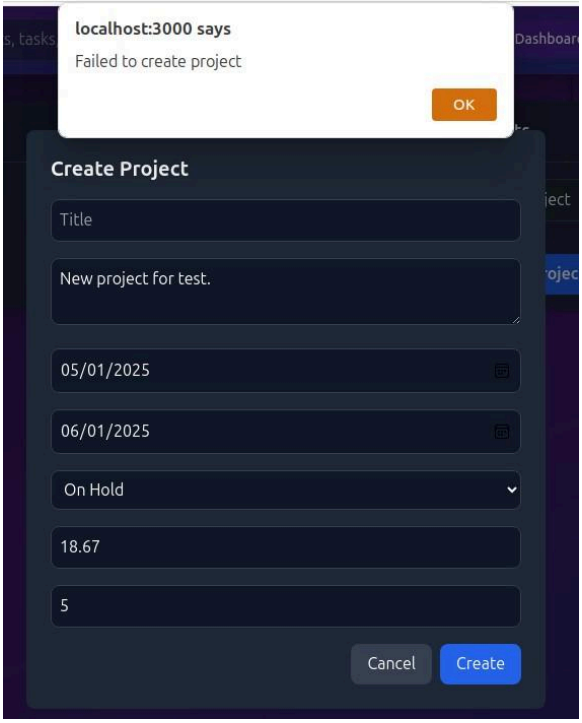


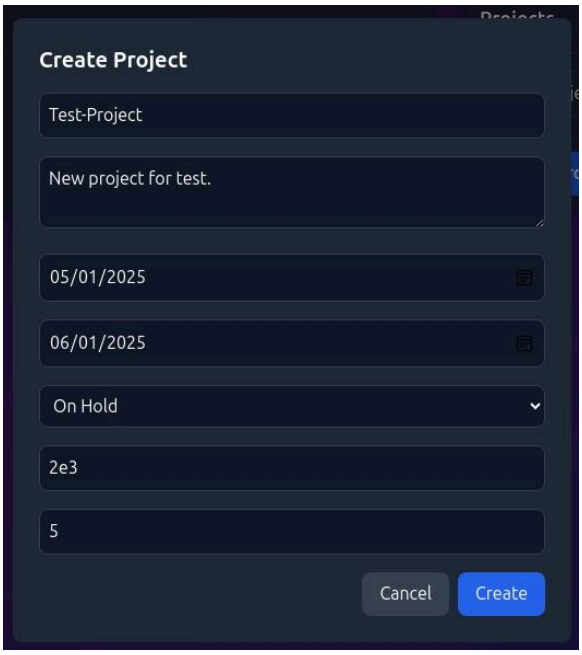
- Example:

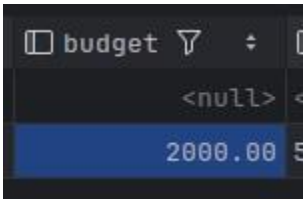
```
const zipRegex = /^\d{5}(-\d{4})?$/;
```

```
if (!zipRegex.test(formData.zip)) {
   setError('Please enter a valid ZIP code');
   return;
}
```

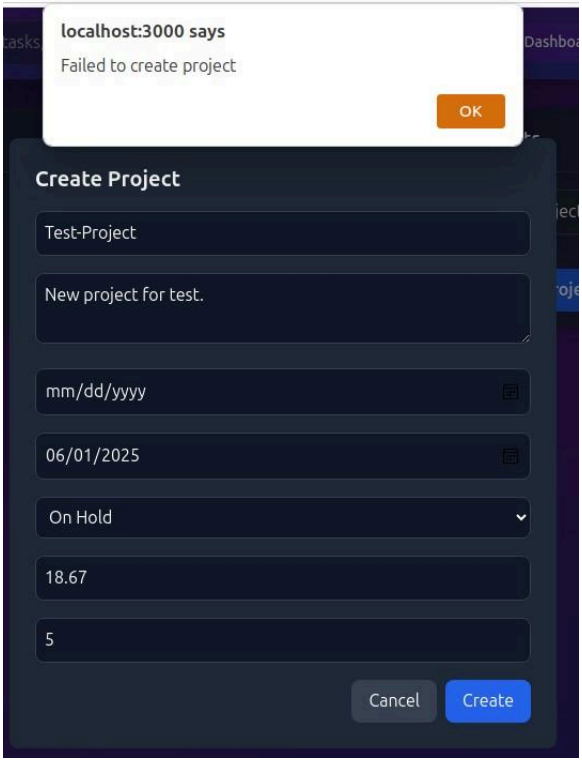
- **Organization Join page**
  - Can only be accessed by users who are not associated with an organization
  - Organization Invite Key
    - The input must match a valid invite key associated with an existing organization in the database.
    - 

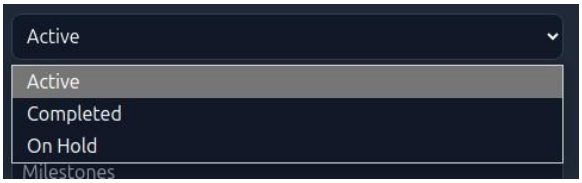
- **Project Creation component**
  - Task title
    - The input must be non-empty.
    - 

- Budget
  - The input is limited to digits and scientific notations.
  - After successful task creation, the input is converted to float type before storing it in the database.
    - 

    - 

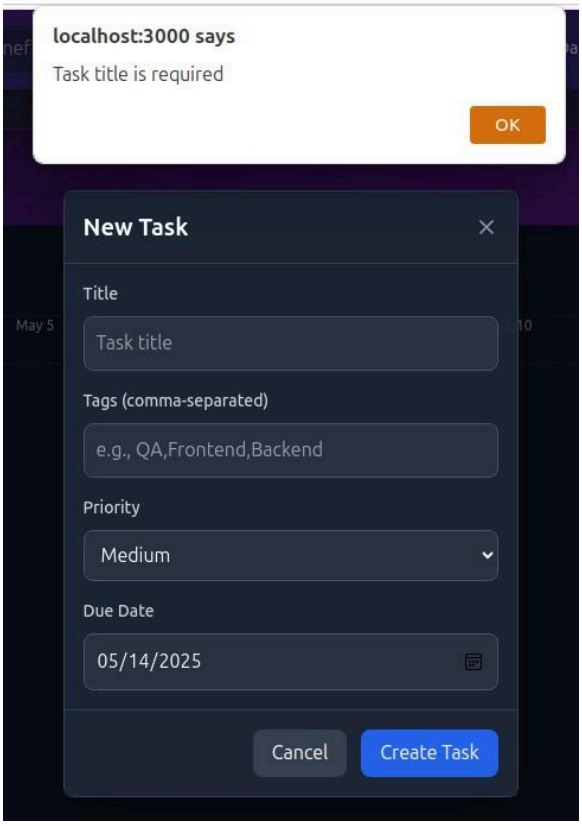
- Start and End date
  - The input must be non-empty for day, month, and year.
  - The input is limited to digits in date format.



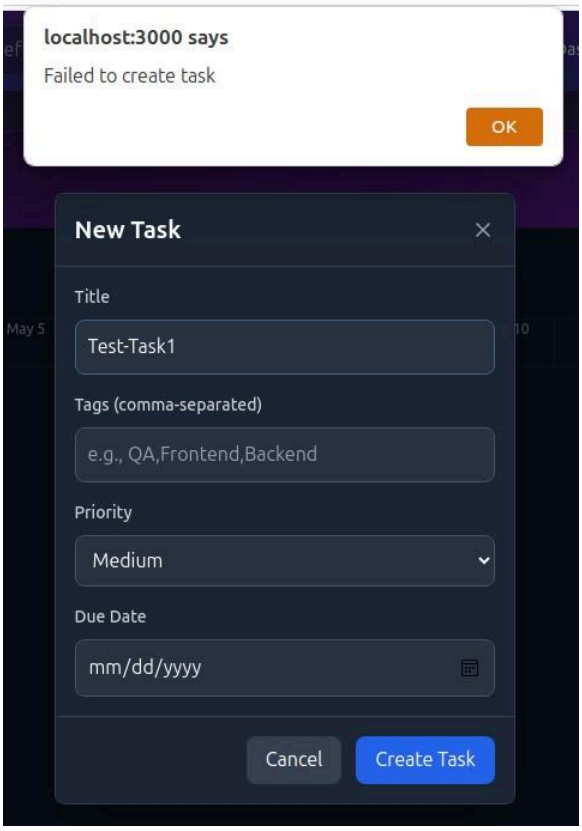
- Status
  - Only 3 options are available.
  - 

  - Example:
    // Convert budget into a valid form
    budget: formData.budget ? parseFloat(formData.budget) : null

- **Task component**
  - Title
    - During creation, the task must have a non-empty title.

- 

- Date
  - Task must have a non-empty due date.



- Example:

```
if (taskTitle && dueDate && assignedUsers.length > 0) {
    // create a new task
}
```

8. Self-Check: Adherence to Non-Functional Specs

- **Status Update:**

  Copy all original non-functional specs from M1V2 and mark each as DONE, ON TRACK, or ISSUE, with explanations if there are problems.

Non-functional Specs	Status	Explanation
The entire web application response time should be <5 seconds	✅	
Client-side web page render time should be <3 seconds	✅	
UI/UX element changes should be synchronized <2 seconds	✅	
Changes to files >10MB should be synchronized <5 seconds		Did not end up with file uploads implemented
Database queries should be <3 seconds	✅	
The provided API response time should be <5 seconds	✅	
The web application can handle 100 concurrent users	✅	
The web application can handle 100 users and 25 Projects without degradation	✅	

The web application can ensure data consistency during data transactions	✅	
The web application can handle concurrent data editions	✅	
Partial functionality without internet connection	Partial	No cache layer outside of a page data being based on a local-stored token, but some functions work with just this.
The system shall cache on the user's machine and send timely updates for safe backups of work	✅	
AES-256, BCrypt, or equivalent encryptions for servers and databases	✅	
Audit logs to see history of access and setting changes		Did not implement logging
API key rotation or use authorization protocol		JWT keys do expire, but the API is presently exposed.
Encrypted user information	✅	
User password specifications	✅	
User permissions and privileges	✅	
Two-Factor Authentication		Did not find/implement any free 2FA.
HTTPS connection	✅	
Bot detection/CAPTCHA to limit DDOS exposure		Did not implement CAPTCHA

Rate limits to limit damage in case of compromise		Did not implement a rate limit function for database
Intuitive and concise UI	✅	
Responsive design that works on many sizes of workspace	✅	
UI/UX elements should have a latency within 5 seconds	✅	
Undo/Redo/Version control for users	❌	
Light/Dark/Custom themes for readability in many environments	❌	
Integration with other services for collaboration and adaptability	❌	Could only use free APIs, so no Google Calendar for example yet.
User guide is provided to explain more complex features	❌	
Multi-input support	❌	All keyboard/moues
Multi-media support	❌	Somewhat, only images for profiles.
Accessibility/WCAG	❌	Could use better descriptors for headers and so on for screen readers.
Comprehensive documentation so developers can pick up where another left off	✅	
Version control for developers, connecting directly to branches of different tasks	✅	

Modular features and microservices to reduce application complexity	✅	
Tests for new functionalities	✅	
Cross-platform support - ie, as many browsers as possible	✅	
Compatibility with multiple APIs	❌	No significant external APIs
Containerized development	✅	
Vertical scaling with optimization	✅	
Potential for horizontal scaling with additional servers	✅	
Load balancing	❌	Can implement via AWS easily, but did not wish to pay.
Cross-Browser/OS/Device support	✅	
Localization	❌	Only setup for US English at present
Backward compatibility	✅	
GDPR/CCPA and other personal data collection consents		
NIST Cybersecurity Framework	✅	
WCAG for accessibility	✅	
Bug track and issues list		External

Regular updates/patches	✅	
User reviews	❌	
Customer Service	❌	
Client-side memory usage should be <4GB	✅	
Server-side memory usage should be <16GB	✅	
Using GitHub for version control	✅	
Master and Development branches	✅	
Feature branches & PRs	✅	
Modular and repeatable code	✅	
Input Validation	✅	
Algorithm optimization	✅	
Readability	✅	
Error Handling	✅	
Comments and technical Documentation	✅	
Reduce power and bandwidth consumption – smallest resource footprint possible	✅	Entire service is on one AWS t2.micro
Optimize the code to increase efficiency	✅	

9. Localization Testing

● Localization Plan:

The purpose of localization testing is to ensure that Serial PM displays language, formatting, and UI elements correctly across the following supported locales:

- **English (US)** – `en-US`
- **English (UK)** – `en-GB`
- **Spanish (Spain)** – `es-ES`
- **Spanish (Mexico)** – `es-MX`

Focus areas include:

- Correct translations for static and dynamic content
- Locale-specific terminology and grammar
- UI adaptability to text length and direction
- Proper number, date, and currency formatting

● Test Cases:

TEST CASE ID	TITLE	DESCRIPTION
TC-LI1	Missing translation keys	Ensure all locale files contain necessary translation keys
TC-LI2	UI layout integrity	Check for layout issues caused by text expansion in localized strings
TC-LI3	Pluralization and grammar	Verify proper use of singular/plural forms and locale grammar rules
TC-LI4	Date/Time/Currency Formatting	Ensure region-specific formatting is applied using `Intl` API
TC-LI5	Terminology Accuracy	Validate the use of culturally appropriate and region-specific vocabulary

TC-LI6	Locale Persistence	Verify the selected locale persists across sessions
TC-LI7	Cultural Sensitivity	Check for any culturally inappropriate content or references

Provide test cases for verifying that all elements (text, UI, etc.) are properly localized.

- ## Results:

Record and analyze the results of localization tests, noting any issues found.

LOCALE	MISSING KEYS	UI LAYOUT	FORMATTING ERROR	TERMINOLOGY ISSUES
en-US	None	None	None	None
en-GB	None	None	None	None
es-es	1	None	None	1
es-MX	None	None	None	None

9. List Of Team Contributions - M4v2 & M5

Team Member	Actions (Checkpoint 1)	Actions (Checkpoint 2)	Rating
Victoria Barnett	Code review, Project Summary	Project page redo, task categories, project views, prepared Demo account, added new project creation flow, extensive final tweaking on Project page, project categorization	9
Alison John	Finished roles system & assisted in M3v2 fixes, ERD	Finished roles system & huge database rebuild, saved SQL tasks to github for tracking	8
Ansh Ankit Patel	Visual appeal documentation	Prepared script for presentation, assisted localization (landing page)	8
Nidhey Patel	Led Usability & QA testing, did non-functional self-check	Finished Usability & QA testing, assisted localization (signin/signup)	8
Pritham Singh Sandhu	Led Usability & QA testing	Finished Usability & QA testing, added terms and conditions, assisted localization	8
Tushin Kulshreshtha	Assisted with Code review	Finished roles system & database rebuild, project page redo, updated API endpoints, led localization	9
Yikang Xu	Security Best practices	Finalized organization page, extensive m4v2/m5 tweaks, assisted localization	9

4. Team Contributions (Semester):

Team Member	Overview	Rating
Victoria Barnett	Led the team, contributed a lot of overall vision and code to the project, tried to work tasks out into the team but this always worked best in Documentation rather than code contribution. Overall getting good feedback, but unsure I executed the task of team leadership as well as could have been. Removed more lines of code than I added, ensured realistic heights based on team performance.	8
Alison John	Great on the database, though did focus on this lane rather than expanding to others when given opportunity. Overall performed all tasks at and above expected level though, and especially increased contribution after discussion of this. Work was extremely competent: our database was rock-solid and needed very little tweaking even as functionality grew more complicated than initial scope.	7
Ansh Ankit Patel	Was always willing to step up, was a large influence on the UI and rotated through many roles exceptionally. Took great leadership on mockups, wireframes, and eventually leading the presentation. Had solid	8

	technical basis but was often given the responsibility that lined up with this “design” role, which I think is acceptable.	
Nidhey Patel	A good role-player all semester. Nidhey did great documentation work and our best milestones had his assistance as technical writer. I do wish Nidhey had been on the repo more and contributing, though he did well when explicitly assigned tasks of this nature.	7
Pritham Singh Sandhu	A solid contributor and had the #3 added lines and time on repo, pritham was able to take entire sections or pages on himself when others needed to focus on higher-level interactions or documentation and deliver.	8
Tushin Kulshreshtha	The foundation of the technical side of the project and an absolutely essential contributor. I (victoria) committed and refined the project greatly, but this was minor and reactive to Tushin’s incredible output and dedication to fixing issues with the project and helping other team members get up to speed. A key and irreplaceable member, if anyone in the course has every been worthy of a 10, I bet it is Tushin.	10
Yikang Xu	Yikang was initially a bit lacking in communication and coordination, though this and his confidence improved greatly over the course of the project, so much so that by the end he was able to be	9

	solely responsible for whole features and documentation sections and delivery on them well. I would evaluate Yikang as the most improved team member throughout the course of the semester. Like others as noted, I do wish in the final standings he had been responsible for more commits.	

5. Post-Analysis – Lessons Learned:

The biggest takeaway I can think of for this project is that technical coordination is *extremely difficult*: it was easy to assign team members documentation tasks and see acceptable results, but communicating a somewhat abstract technical requirement was a different task entirely. Even after adopting Trello, adopting frequent out of class task check ins, and doing some introductory training sessions on the technical stack, few team members were able to hop right in and make the work their own. I now understand why new programmers on the job are often paired or given weeks and weeks to familiarize themselves with a codebase *before* contributing: doing excellent software engineering is not about knowing how to create one specific piece or being an expert in one 'vertical' of the field – it is about knowing just enough about as much of the process as possible that you can be effective in the first place, about knowing why certain tools are being used and having the ability to go look at that documentation. In short, it is a tall order, and the transformation of a Computer Science student into one of these exceptional engineers a taller one, though some arrive with that experience. With that said though, as we recognized this, we did do more pairing-up for large requirements and something like our M4v2 Localization was a near full-team effort on both the documentation and technical ends, which was great to see. So, it seems like despite the difficulty, we eventually were capable of figuring out this great question of the division of the technical work of the project at the very end, which I am proud of. It also marked a task where the team lead needed to do none of the technical implementation at all, which was a great growth indicator for the team.

Overall, I think the team ended up with a good product, but that we could have done better at transforming ourselves into a whole team of engineers. For the project itself, our biggest difficulty was trying to translate a complicated visual design into reality: we reduced the scale and scope of a feature idea as we went and reality became more clear, though it's sure that with a lot of technical work a Project Timeline of the original vision *could* be implemented, it would incur an exceptional amount of technical debt that we were not actually capable of handling with student developers on a short timeline. Some future improvements would include: a strong technical core (3-4 members) focused on the logic of the hardest feature from the very beginning, rather than letting 1-2 team members try to string it together. I think this would eliminate bottlenecks and more quickly reveal the viability or non-viability of big ideas. Other improvements we're interested in are increasing the usability of the application for the technical user: github integration with tasks tying PM-level descriptions to commits, documents and branches directly sharable via the chat and comments on tasks & projects, and a greater amount of the "automatic" project functionality that we had pitched, where for example creating a feature branch and deadline also creates the QA & PR review tasks for that feature.

Finally, I think our project ended up looking exceptional among some of the others, though our imposed realism did limit it's uniqueness somewhat, but this was by design and choice as we were most interested in practical methods of competition such as how they function, meaning a consolidation of specialization and being able to win on price, thus a platform for both

developers & project management was the idea and I think it was executed well though there is a lot of room for future growth if it were to be continued.