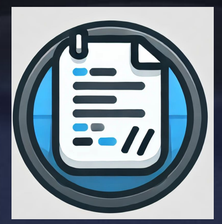
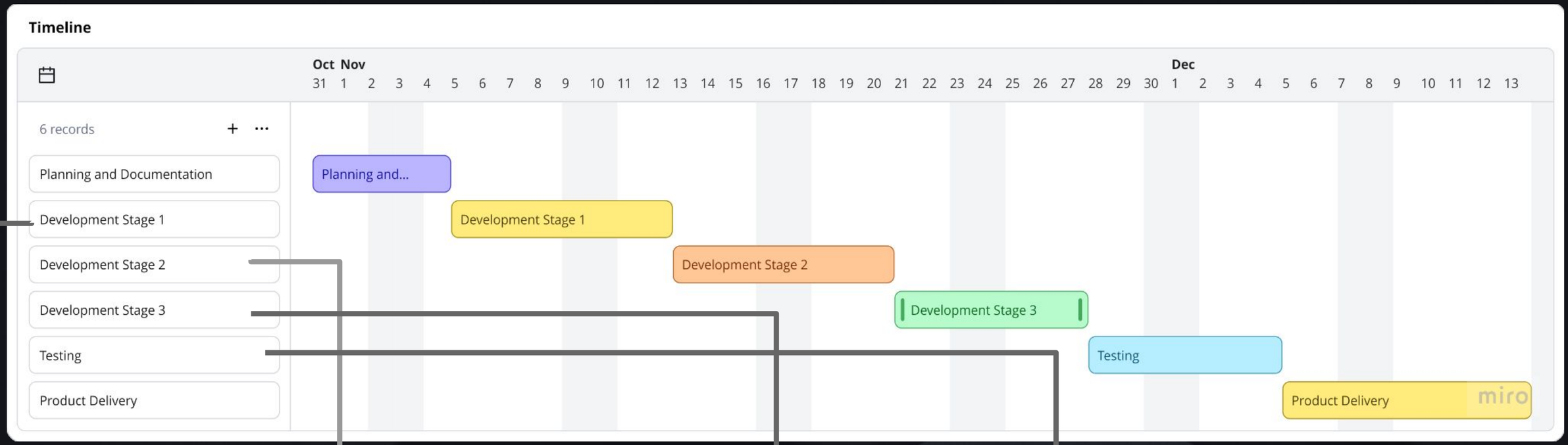
# Team-3 Pitch Document

Devlog: Developer focused journaling tool



# Timeline




Frontend Dashboard

Frontend for Templates and Backend for storage

Journal features and theme

CI/CD Workflows and Feature Testing

# Statement of Purpose

## Purpose

Streamline development documentation with a lightweight, template-driven platform using vanilla JavaScript & MySQL

## Who

Development teams (5-20 members) needing quick, organized technical documentation

## Why

- Preserve context behind technical decisions
- Improve knowledge retention and transfer
- Speed up onboarding of new team members

## When

- During architectural decision-making
- After solving complex bugs
- Post-code review sessions
- During sprint retrospectives

## Tech Stack

- JavaScript
- HTML CSS
- MySQL

## Key Features

- Ready-to-use templates (TODO, MOM, Bug Reviews)
- Tag-based organization & quick search (tentative)
- GitHub commit linking
- Dark/light mode for developers

# User Personas

## Persona 1: College Student - Emma

**Background:**

- Emma is a third-year college student majoring in Biology.
- She is highly organized and prefers digital tools for schoolwork and collaboration.

**Goals:**

- Manage and excel in multiple courses simultaneously.
- Keep track of assignments and prepare for exams efficiently.

**Behavioral Traits:**

- Tech-savvy and highly adaptable to new software.
- Prefers visually organized data and quick access to needed information.

**Use of Platform:**

- **Notion:** Emma uses Notion to create and manage a digital notebook for each of her courses. She organizes these notebooks by subject matter, including lecture notes, research papers, and lab results.
- **Trello:** She utilizes Trello to track her assignments and exam dates, setting up boards for each class with cards for assignments, study sessions, and project deadlines. Tags help her prioritize and manage her workload based on urgency and subject.

## Persona 2: Project Manager - Alex

**Background:**

- Alex is a seasoned project manager in an IT company specializing in software development.
- He has extensive experience in project planning, team management, and product delivery.

**Goals:**

- Efficiently manage multiple projects with varying scopes.
- Ensure on-time delivery of high-quality software products.

**Behavioral Traits:**

- Detail-oriented, with a focus on big-picture outcomes.
- Strong leadership skills and effective communicator.

**Use of Platform:**

- **Notion:** Alex uses Notion to maintain comprehensive documentation for each product, including detailed feature descriptions and user manuals, which serve as a central knowledge base for the team.
- **Jira:** He leverages Jira to oversee the project roadmap, set milestones, and distribute tasks across his development team. This includes monitoring progress and adjusting timelines as needed to meet project deliverables.

## Persona 3: Freelance Graphic Designer - Mia

**Background:**

- Mia is a freelance graphic designer who works with various clients on creative projects, from branding to digital marketing campaigns.
- She needs to manage projects and tasks efficiently to meet client expectations and deadlines.

**Goals:**

- Organize client projects and personal tasks to maximize productivity.
- Maintain clear communication and project transparency with clients.

**Behavioral Traits:**

- Creatively driven and visually oriented.
- Independent worker who values structure and clear timelines.

**Use of Platform:**

- **Notion:** Mia uses Notion to create pages for each client, where she organizes project briefs, design ideas, and final assets. She links related content and incorporates visual elements to keep her creative flow accessible and structured.
- **Trello:** She sets up boards for each project phase, like concept development, design revisions, and final approvals, using cards to manage specific tasks and deliverables. This helps her track progress and set expectations with clients clearly.
- **Jira:** Although not her main tool, Mia occasionally uses a simplified Jira setup to map out longer-term personal projects and career milestones, helping her plan her growth and skills development effectively.

# Risks

## 1. Template System Risks

**Rabbit Holes:**

- Spending too much time building a complex template editor
- Over-engineering template customization
- Creating a WYSIWYG editor from scratch

**What Could Go Wrong:**

- Team gets stuck building advanced formatting features
- Templates become too rigid or too flexible
- Excessive time spent on editor features instead of core functionality

**Keep it Simple:**

- Start with basic HTML templates
- Use contentEditable for basic editing
- Stick to predefined templates without customization initially
- Avoid building complex formatting tools

## 2. State Management Risks

**Rabbit Holes:**

- Building a complex state management system
- Over-using localStorage
- Creating unnecessary event systems

**What Could Go Wrong:**

- Data inconsistency between client and server
- localStorage size limits hit unexpectedly
- Complex state updates causing bugs
- Race conditions in data updates

**Keep it Simple:**

- Use simple CRUD operations
- Store minimal data in localStorage
- Rely on server for data persistence
- Avoid real-time updates initially

## 3. UI/UX Complications

**Rabbit Holes:**

- Perfect pixel-perfect designs
- Complex animations and transitions
- Over-engineering responsive design
- Too many theme options

**What Could Go Wrong:**

- Inconsistent appearance across browsers
- Mobile version takes too long to implement
- Performance issues from animations
- CSS becomes unmanageable

**Keep it Simple:**

- Start with mobile-first approach
- Use simple CSS transitions only
- Stick to two themes (light/dark)
- Focus on functionality over aesthetics initially

## 4. Performance Traps

**Rabbit Holes:**

- Premature optimization
- Over-caching
- Complex DOM manipulation
- Handling large documents

**What Could Go Wrong:**

- Slow page loads
- Memory leaks
- Browser crashes with large documents
- Poor mobile performance

**Keep it Simple:**

- Start with basic optimization
- Implement pagination early
- Limit document size
- Use simple DOM updates

## 5. Team Coordination Risks

**Rabbit Holes:**

- Over-complicated git workflows
- Too many coding standards
- Excessive documentation requirements
- Complex review processes

**What Could Go Wrong:**

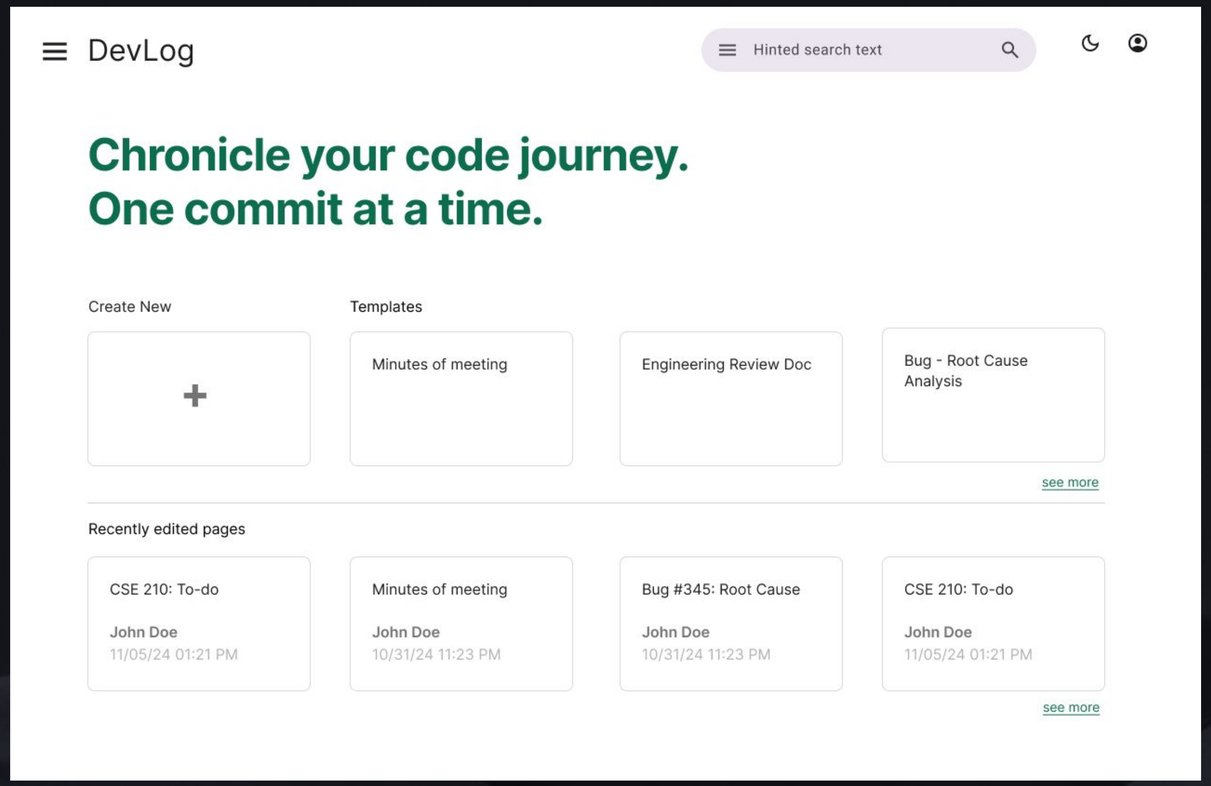
- Merge conflicts
- Inconsistent code
- Time wasted on process
- Delayed releases

**Keep it Simple:**

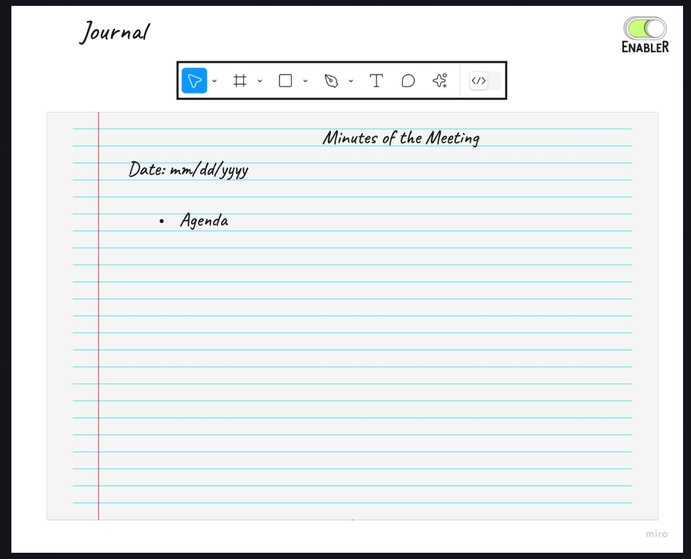
- Basic git branching strategy
- Essential coding standards only
- Documentation for critical features only
- Quick code reviews

# Wireframes

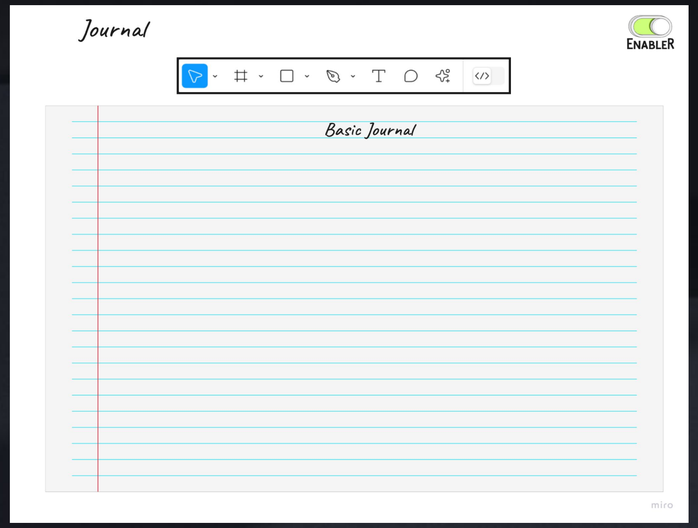
## Homepage




## Minutes of meeting template




## Basic journal page



# Template: Feature Specifications

DevLog

Hinted search text

Create New +

Templates

Todo list

Blank page

Bug reviews

Engineering doc

Minutes of meeting

Recently edited

CSE 210: Todo list

Sprint retrospective: MOM

**Feature Specifications: [Feature_Name]**

Last Updated: [YYYY-MM-DD]

Author: [Name]
Status: [Draft/In Review/Approved]
Target Release: [Version/Sprint]

1. Overview

1.1 Feature Summary
[Brief description of the feature in 2-3 sentences]

1.2 Business Context
- Problem Statement: [What problem does this solve?]
- Success Metrics: [How will we measure success?]
- User Value: [Why is this important to users?]

1.3 Scope
- In Scope: [List of what's included]
- Out of Scope: [List of what's explicitly not included]
- Future Considerations: [Potential future enhancements]

2. Technical Architecture
2.1 System Design
- Architecture Diagram: [High-level system diagram]
- Component Interaction: [How new components interact with existing systems]
- Data Flow: [Description of data flow between components]

2.2 Data Model
- Schema Changes: [Required database changes]
- New Tables/Fields: [Detailed schema design]
- Data Migration: [Migration strategy if needed]

2.3 API Design
- Endpoints: [New/modified API endpoints]
- Request/Response Format: [API contracts]
- Error Handling: [Error scenarios and responses]

3. Implementation Details
3.1 Technical Requirements
- Dependencies: [External services/libraries needed]
- Infrastructure Changes: [Required infrastructure updates]
- Security Considerations: [Security requirements/concerns]
- Performance Requirements: [Performance targets/constraints]

3.2 Algorithm/Logic
- Business Logic: [Core algorithms/processing logic]
- Edge Cases: [Known edge cases and handling]
-State Management: [How state is managed]

3.3 User Interface
- UI Components: [Required UI changes]
- User Flows: [User interaction flows]
- Mockups: [Links to design mockups]

4. Testing Strategy
4.1 Test Plan
- Unit Tests: [Areas requiring unit tests]
- Integration Tests: [Integration test scenarios]
- Performance Tests: [Performance test cases]
- User Acceptance: [UAT criteria]

# Template: Bug Reviews (Root cause analysis)

DevLog

Hinted search text

Create New +

Templates

Todo list

Blank page

Bug reviews

Engineering doc

Minutes of meeting

Recently edited

CSE 210: Todo list

Sprint retrospective: MOM

**Bug Review ##**

Last Updated: [YYYY-MM-DD]

Bug ID: [Tracking number from your system]
Title: [Short, descriptive title of the bug]
Priority: [Critical/High/Medium/Low]
Status: [Open/In Progress/Fixed/Closed]
Reported By: [Name]
Reported Date: [YYYY-MM-DD]

Environment Details
- Platform: [OS/Browser/Device]
- Version: [Application version number]
- Environment: [Development/Staging/Production]
- User Role: [Admin/Regular User/etc.]

Bug Description

Current Behavior
[Detailed description of what is happening]

Expected Behavior
[Description of what should happen]

Steps to Reproduce
1. [First step]
2. [Second step]
3. [Continue as needed]

Impact
- Users Affected: [Number or percentage of users]
- Business Impact: [Description of business implications]
- Functionality Impact: [Which features are affected]

Technical Analysis

Root Cause
[Detailed technical explanation of what caused the bug]

Code Location
- File(s): [Relevant file paths]
- Function/Component: [Specific function or component names]
- Line Numbers: [If applicable]

Solution

Code Changes
[Code snippet or summary of changes made]

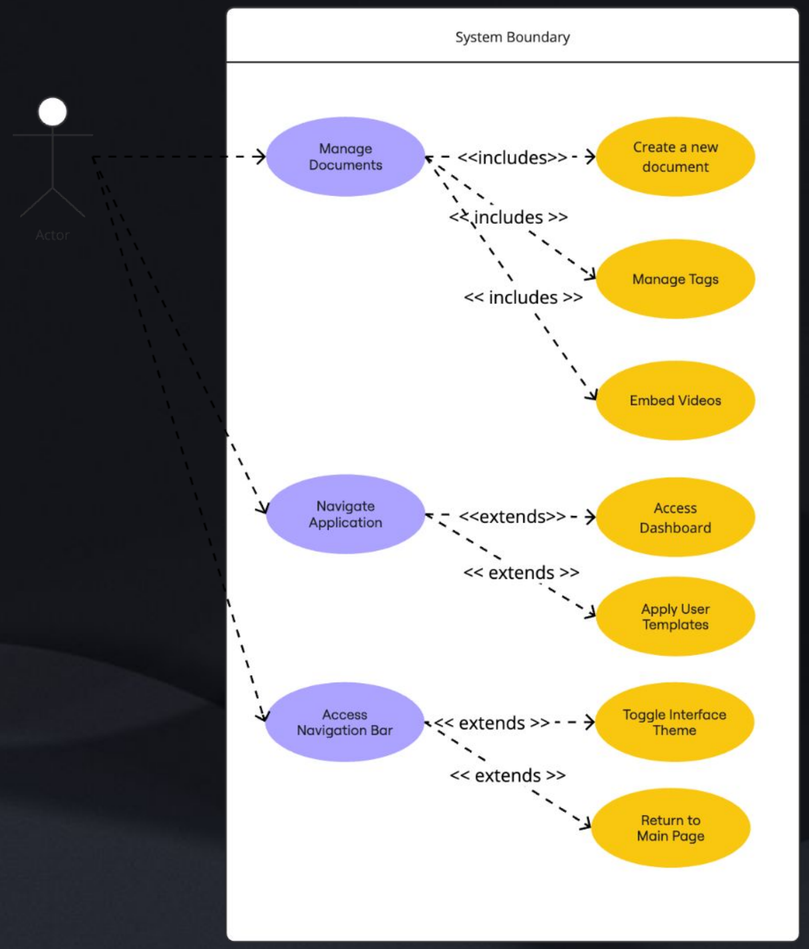
Lessons Learned

Prevention Measures
[Steps to prevent similar bugs in the future]

Documentation Updates
[Required changes to documentation]

Sign-off
- Fixed By: [Developer name]
- Reviewed By: [Reviewer name]
- QA Verified By: [QA engineer name]
- Fix Deployed Date: [YYYY-MM-DD]

# Use Case Diagram (UML)




BETA

## UML Use Case Diagram

Actors

- User

Use Cases

1. Manage Documents
   - Includes viewing, downloading, and linking documents to each other.
2. Create New Document
   - User can create a new document from scratch or using predefined templates.
3. Manage Tags
   - User can create, assign, and filter tasks and documents by tags.
4. Embed Videos
   - User can embed YouTube videos for development productivity.
5. Navigate Application
   - User can navigate through the application using the navbar to access different pages.
6. Access Dashboard
7. Apply User Templates
   - User can use different templates like todo list, journaling, minutes of meetings, etc.
8. Access Navigation Bar
9. Toggle Interface Theme
   - User can switch between dark and light mode.
10. Return to Main Page

Relationships

- “Manage Documents” use case includes “Create New Document”, "Manage Tage", "Embed Videos".
- “Navigate Application” extends to specific functionalities like accessing the dashboard/home which can lead to other actions (e.g., creating documents, using templates).
- “Access Navigation Bar” extends to specific functionalities like toggling interface themes or returning to the main page.

miro

# System Design

## Architecture Components

| Component | Description | Key Features |
|---|---|---|
| **Frontend** | Browser-based UI | - Vanilla JavaScript- Local storage caching- Simple state management |
| **Backend** | Basic HTTP server | - REST API endpoints- Template engine- Authentication service |
| **Database** | MySQL | - Document storage- User management- Tags and templates |

## Core Database Tables

| Table | Purpose | Key Fields |
|---|---|---|
| `users` | User management | - id, username, email, password_hash |
| `documents` | Document storage | - id, title, content, template_type, author_id |
| `tags` | Document organization | - id, name |
| `templates` | Pre-built layouts | - id, name, content |

## Key API Endpoints

| Endpoint | Method | Purpose |
|---|---|---|
| `/api/documents` | GET/POST | Document management |
| `/api/templates` | GET | Template retrieval |
| `/api/tags` | GET/POST | Tag management |
| `/api/auth` | POST | Authentication |

## File Structure

| Directory | Contents |
|---|---|
| `/public` | - CSS files- JavaScript modules- Template files |
| `/server` | - API routes- Services- Config files |

## Caching Strategy

| Type | Purpose | Storage |
|---|---|---|
| Client | - Recent documents- User preferences | LocalStorage |
| Server | - Template cache- Query results | Memory |

## Security Measures

| Area | Implementation |
|---|---|
| Authentication | Session-based |
| Data Safety | SQL prepared statements |
| Input | XSS sanitization |
| CSRF | Token validation |