Group 10 Design Document – DevSync (Project Tracker with GitHub Integration)

By Ahmed Ikram 2571642, Martin Litvin 2422734, Theo Short 2573299, Andre Ndong Nto 2621844, Huzaifa Ijaz 2541544

1. Problem Definition

Overview

Modern development teams often use multiple tools for project management and code hosting leading to scattered information and confusion. Out web app **DevSync** aims to centralise task management whilst integrating with GitHub to:

1. Link tasks to GitHub repositories, pull requests, and issues.
2. Track code changes and task progress.
3. Provide real-time updates and progress reports for team leaders.
4. Allow developers to comment on tasks for better collaboration
5. Deliver notifications for task assignments and updates

Key Features

1. **Client App (Team Members)**
    - View assigned tasks, update task progress.
    - See related GitHub pull requests or issues for each task.
    - Comment on tasks for collaboration.
    - Receive notifications when tasks are assigned or updated.

2. **Admin App (Project Managers)**
    - Assign tasks, set deadlines and track task progress.
    - Generate progress reports and monitor GitHub activity.
    - View and respond to task comments.
    - Configure notification preferences.

3. **Backend API**
    - Stores project data, tasks, user info.
    - Integrates with the GitHub API to fetch repository details.
    - Supports task commenting and real-time notifications.

4. **External API (GitHub)**
    - Link tasks to GitHub repositories for easy cross-referencing.
    - Optionally creates or updates GitHub issues automatically when tasks are created or updated.

2. Technology Stack

Frontend

1. **Framework**
    - **React.js** – Chosen for its component-based structure ensuring efficient UI updates and reusable elements making task management and user interaction seamless.

2. **Styling:**
    - **Tailwind CSS** – Provides a clean and modern responsive design with minimal custom styling.

3. **Routing**
    - **React Router** – Enables smooth navigation between task lists task details and the admin dashboard, ensuring an intuitive user experience.

4. **State Management**
    - **Context API** – lightweight state sharing between components allowing for efficient management of authentication and task states without the complexity of Redux.

5. **HTTP Requests:**
    - **Axios** – Used for API communication enabling structured and error-handled interactions between the frontend and backend.

6. **Real-Time Features**
    - **Socket.IO** – Enables real-time updates for notifications and task comments

Backend

1. **Framework**
    - **Flask** – A lightweight and scalable python framework that simplifies the development of RESTful APIs, making backend implementation faster and more maintainable.
    - **Flask-Socket.IO –** Provides WebSocket support for real-time notifications and live task updates.

2. **Database**
    - **PostgreSQL** – Chosen for its reliability and scalability ensuring structured data storage and supporting complex queries needed for project tracking, including storing task comments and notifications.

3. **Authentication:**
    - **JWT** (JSON Web Tokens) – Ensures secure stateless user sessions preventing unauthorised access to project data.

4. **Hosting:**
    - **Render** (free tier with autoscaling) – Provides flexible, scalable hosting solutions, ensuring smooth deployment and uptime.

External API Integration

- **GitHub API** - Facilitates repository and issue tracking within DevSync.

- **GitHub OAuth** – Enables secure user authentication using GitHub credentials ensuring access control.

Technology Stack Justification

Component	Technology	Alternative	Why We Chose It
Backend	**Flask** (Python)	**Django** (Python)	Flask is **lightweight**, more **flexible** for APIs, and has **fewer built-in constraints** compared to Django, which is more opinionated and monolithic. Flask allows for faster API development without unnecessary overhead.
Frontend	React.js	Angular (TypeScript)	React is lighter, more flexible, and has a larger ecosystem compared to Angular, which has a steeper learning curve and is often better for enterprise apps. React's component-based architecture makes it ideal for a scalable UI.
Database	**PostgreSQL**	**Firebase** (NoSQL)	PostgreSQL is a **relational database**, ideal for structured data like tasks, users, and projects. Firebase is NoSQL and lacks **strong relational integrity**, making it less suitable for project management applications.
Real-Time Communication	**Socket.IO (**WebSockets**)**	**Polling (AJAX Requests)**	WebSockets via **Socket.IO** allow **real-time updates** (notifications, live task status) with minimal latency. Polling is less efficient because it requires frequent HTTP requests, increasing server load.
Authentication	**GitHub OAuth**	**Custom Login (Email/Password)**	GitHub OAuth is **more secure**, removes the need to store passwords, and simplifies **user onboarding** for developers. A custom login would require additional security measures like password hashing & multi-factor authentication (MFA).
Cloud Host Platform	**Render (Deployment)**	**AWS EC2 / Heroku**	Render provides **simpler auto-deployment** with **free-tier scaling**, whereas AWS EC2 requires **manual infrastructure setup** and is more complex for small projects. Heroku is a good alternative but has pricing limits.
Version Control	**Git/GitHub**	**GitLab/Bitbucket**	GitHub offers superior integration with our chosen stack, extensive CI/CD capabilities, and seamless OAuth implementation. It also provides better collaboration features and a much larger developer community.

3. System Architecture

Overview

This system consists of **four** key components:

1. **Client App (React)** – User interface for tracking tasks, updating progress and linking with GitHub.
2. **Admin App (React)** – User Interface for task assignment, progress monitoring and reporting.
3. **Backend API (Flask)** – Centralised service for task management, authentication and API integrations.
4. **PostgreSQL Database** – Stores persistent data including tasks, user roles and project details.

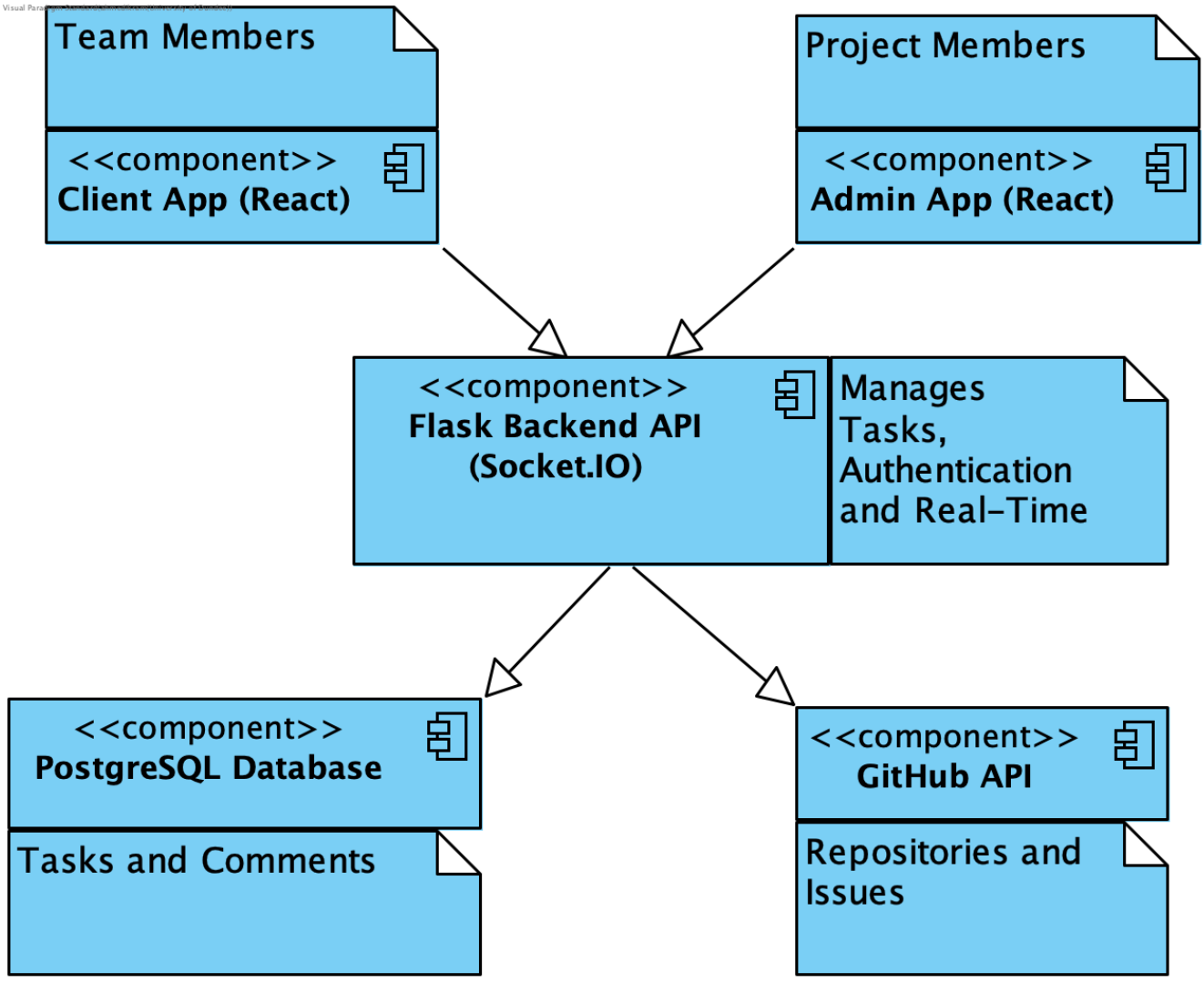
Frontend-Backend Communication

- All GitHub API calls go through the Flask backend to handle authentication and rate limits securely.

Real-Time Updates

- WebSockets facilitates live task updates and notifications.

System Architecture Diagram



4. API Design

Authentication Endpoints

Method	Endpoint	Description	Request Body	Response Example
POST	/api/auth/register	Register a new user	{ "name": "John Doe", "email": "john@example.com", "password": "secure123", "role": "developer" }	{ "message": "User registered successfully", "userId": 1 }
POST	/api/auth/login	Authenticate user, return JWT	{ "email": "john@example.com", "password": "secure123" }	{ "token": "eyJhbGciOiJIUz...", "userRole": "developer" }

Task Management Endpoints

Method	Endpoint	Description
GET	/api/tasks	Fetch all tasks (Admin only)
POST	/api/tasks	Create a new task (Admin only)
PUT	/api/tasks/{id}	Update an existing task
DELETE	/api/tasks/{id}	Delete a task (Admin only)
PATCH	/api/tasks/{id}/progress	Update task progress (Developer)

GitHub Integration Endpoints

Method	Endpoint	Description
GET	/api/github/repos	Fetch user repositories from GitHub
GET	/api/github/issues	Fetch GitHub issues linked to tasks
POST	/api/github/link-task	Link a task to a GitHub issue or pull request

Comment Management Endpoints

Method	Endpoint	Description
POST	/api/tasks/{id}/comments	Add a comment to a task
GET	/api/tasks/{id}/comments	Fetch comments for a task

5. Integration with External APIs

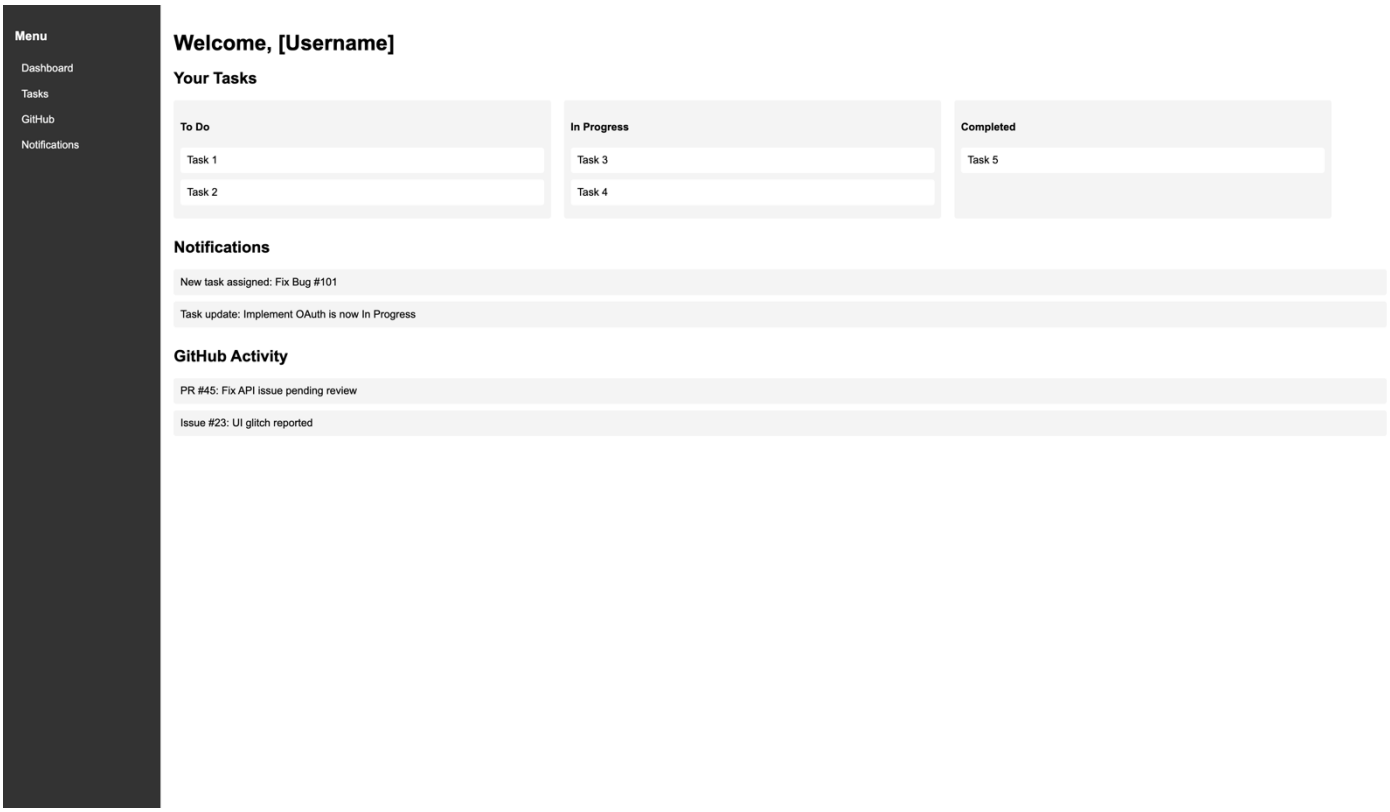
1. **GitHub OAuth Flows**
   - When a user chooses to link their GitHub account, they'll be redirected to GitHub's OAuth consent screen.
   - Upon Successful authentication, GitHub provides an OAuth token, which is securely secured on the server side (Flask).
2. **GitHub API Calls**
   - All GitHub-related API calls (fetching repos, issues, linking tasks) happen server-side using the user's OAuth token.
   - Flask manages these calls to avoid exposing tokens on the frontend. - We handle rate limiting by caching responses or using backoff strategies if GitHub's rate limit is reached.
3. **Error Handling**
   - If GitHub is down or tokens expire, the backend returns an appropriate error message to the frontend.
   - Users can re-authenticate if needed, renewing their OAuth tokens.
4. **Security Measures**
   - Tokens are encrypted or stored securely in the database.
   - Role-based checks ensure only Admins can perform tasks like linking tasks to repos or creating issues.
5. **Future Extensions**
   - We may integrate webhooks from GitHub to receive push/pull request events in real time.
   - Additional APIs (e.g., Slack) could be added similarly if needed.

6. Wireframes

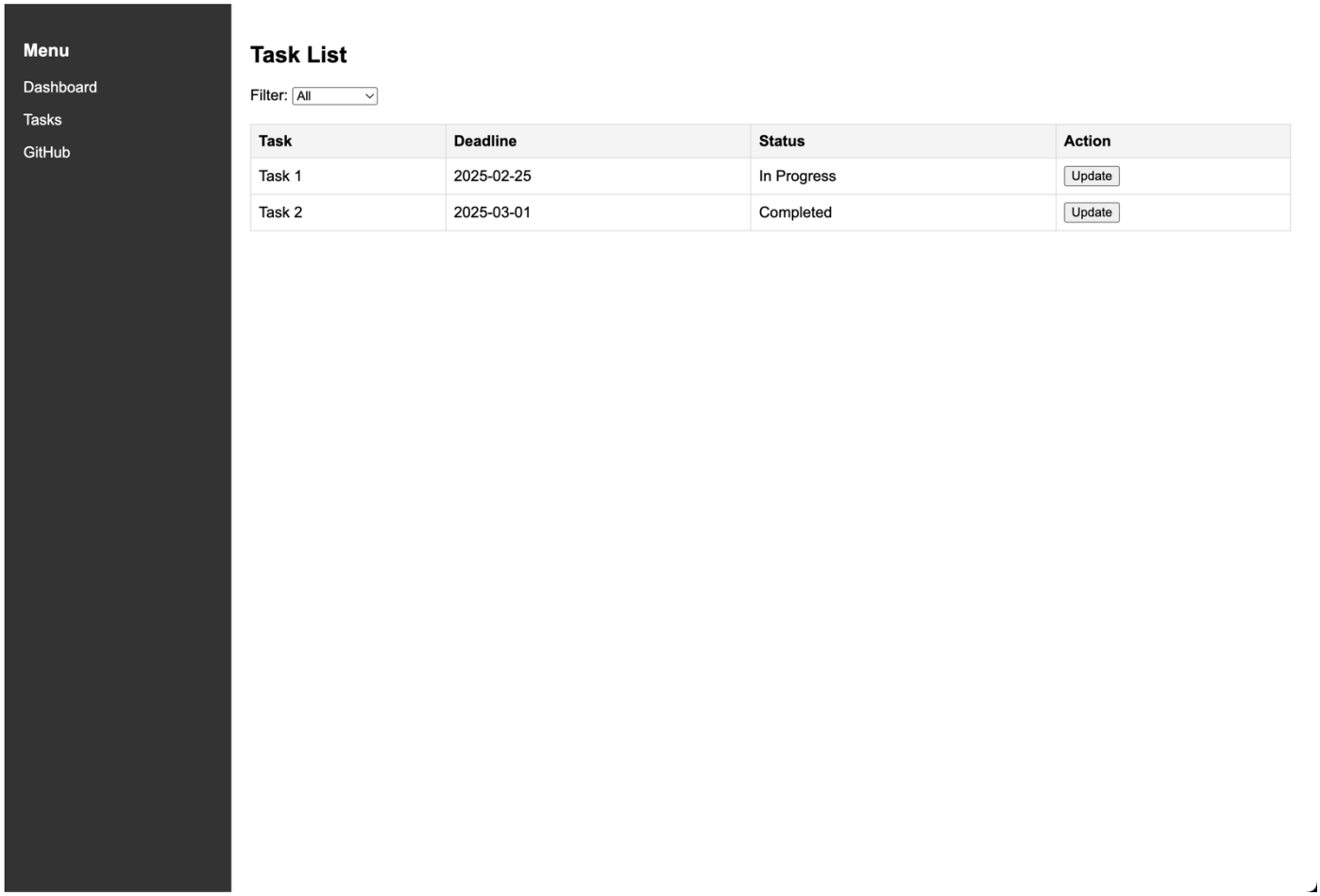
Client App (Team Members)

1. **Dashboard Page:** Shows assigned tasks, progress bars, relevant GitHub pull requests/issues
2. **Task List Page:** Shows a list of tasks with filters and general details
3. **Task Details Page:** Ability to mark progress, link to GitHub code or pull requests
4. **GitHub Integration Page:** Shows connect to GitHub button, Dropdowns for linked repos and issues and an unlink issue button

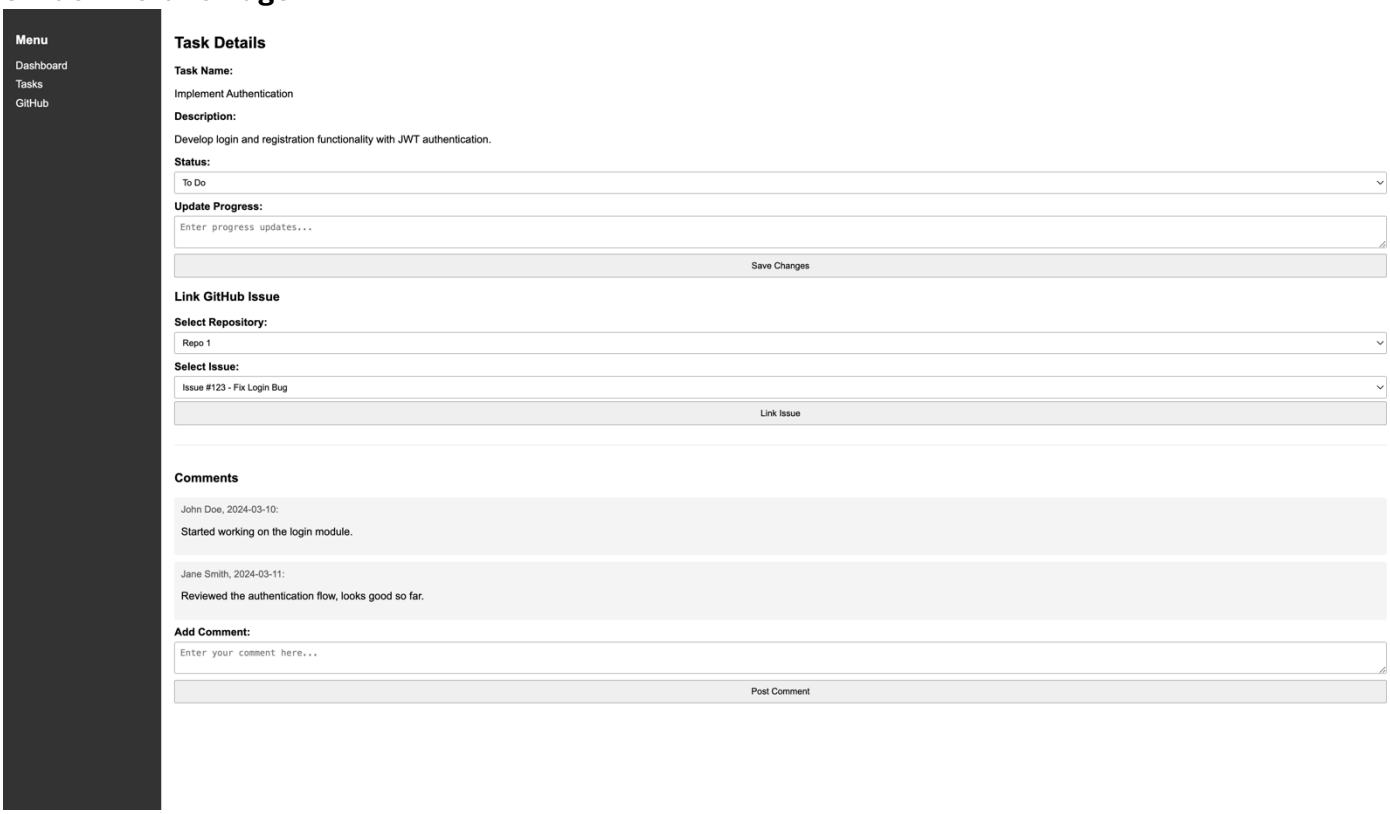
1. Dashboard Page



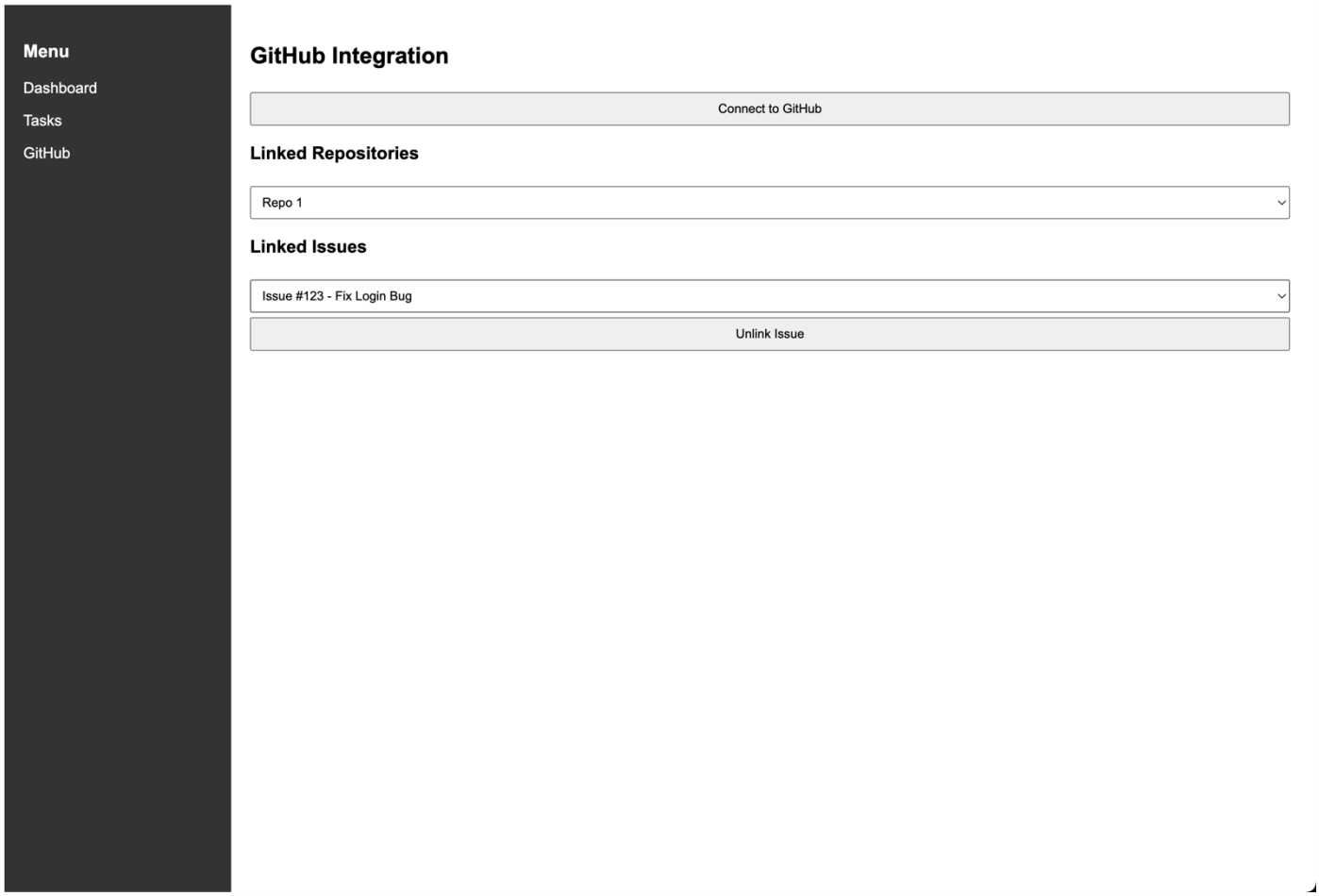
2. Task List Page



3. Task Details Page



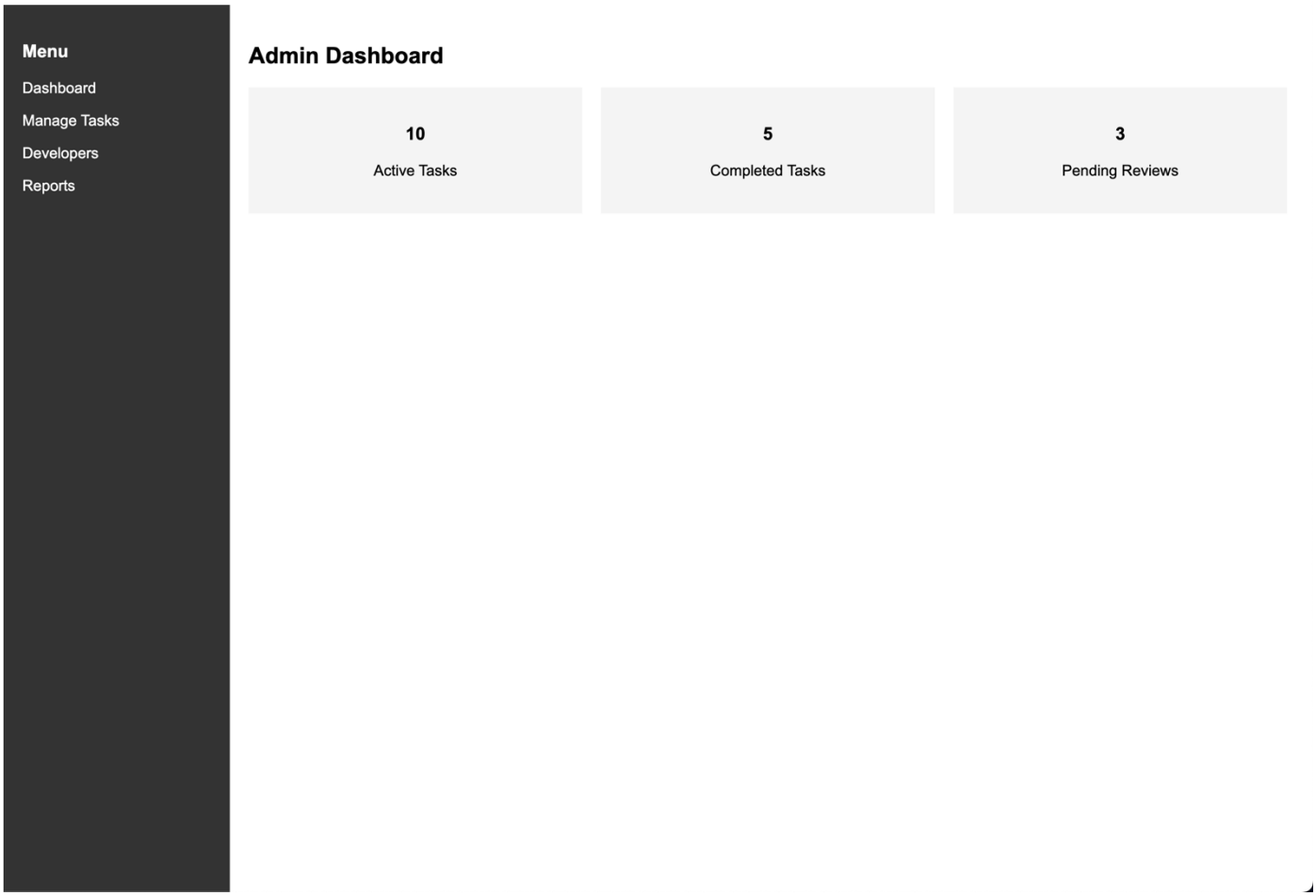
4. GitHub Integration Page



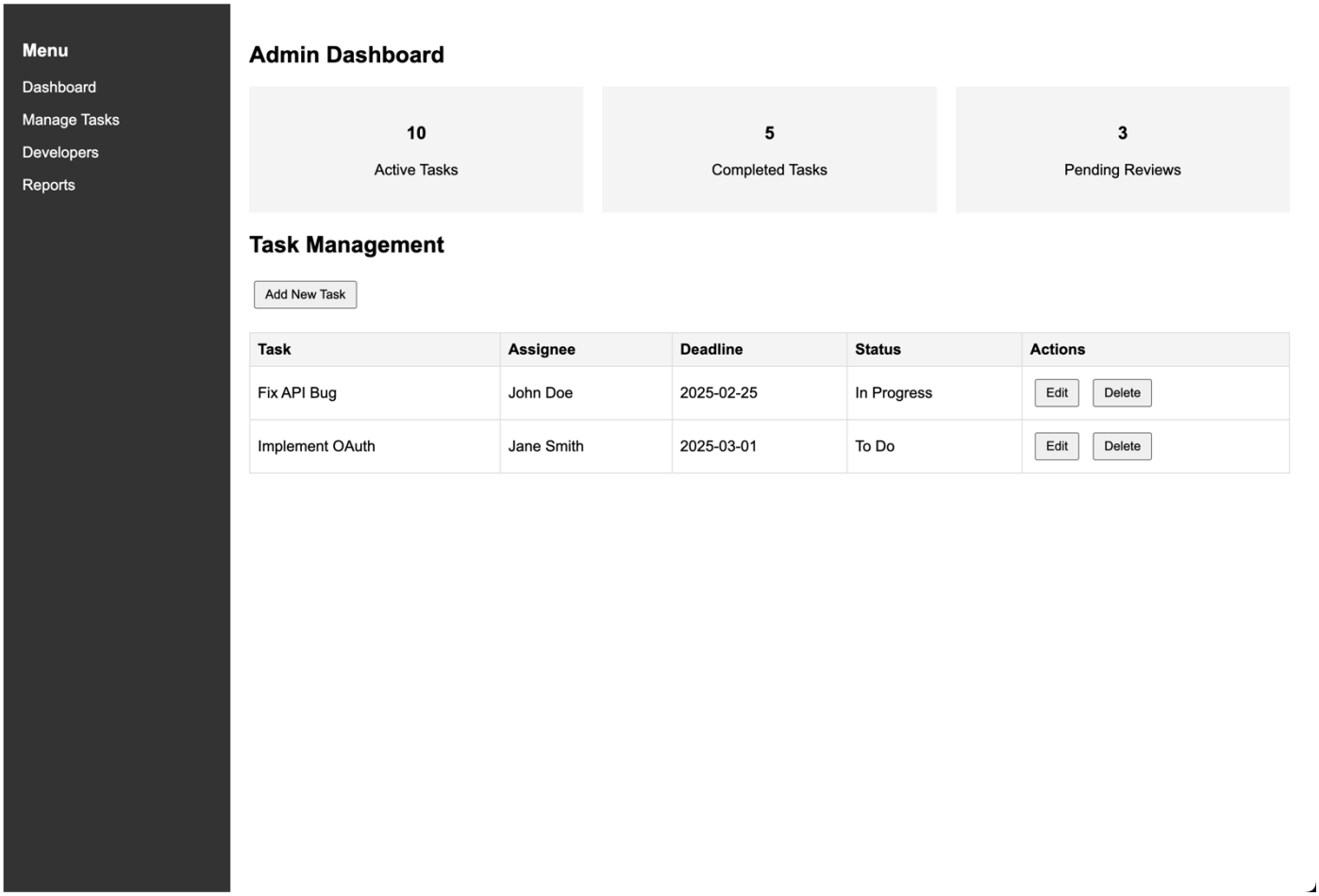
Admin App (Project Managers)

1. **Admin Dashboard Page:** Overview of all tasks, filters by status or team member
2. **Task Creation/Assignment Page:** Form to create tasks and link to GitHub repository
3. **Developer Progress Page –** Monitors developers' progress on assigned tasks.
4. **Reports Page:** Basic analytics – tasks completed, open pull requests, overdue tasks, etc

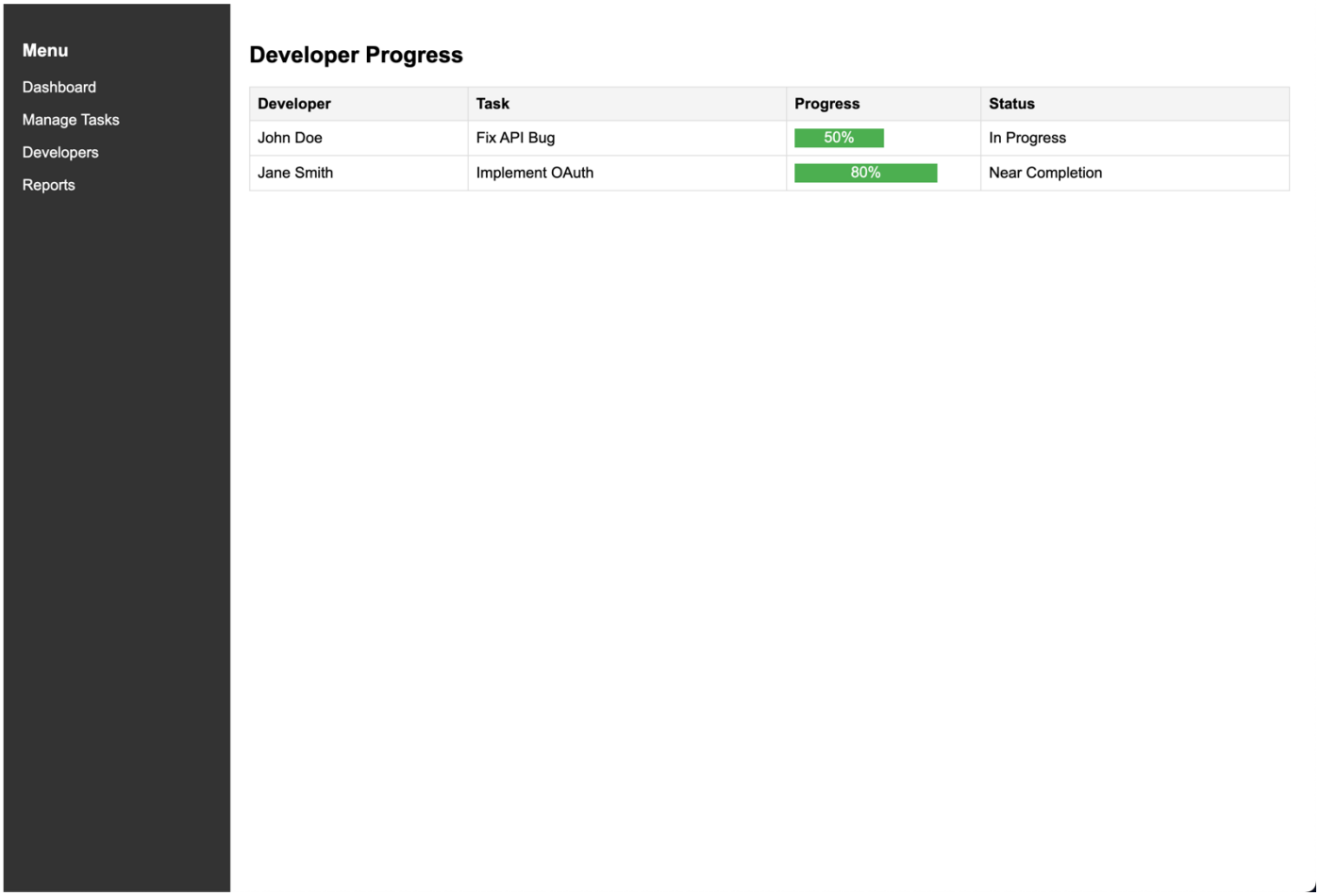
1. Admin Dashboard Page



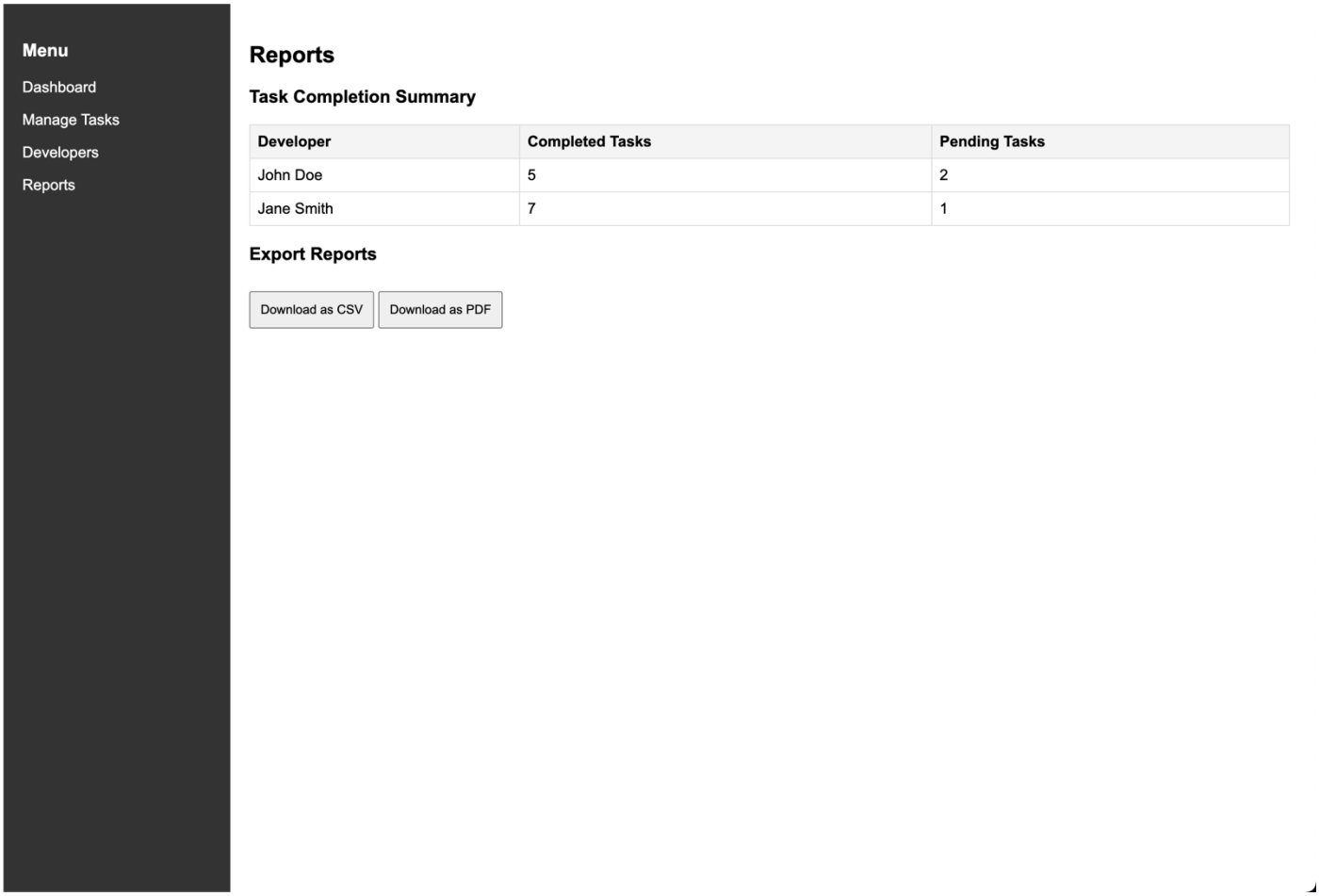
2. Task Creation/Assignment Page



3. Developer Progress Page



4. Reports Page



7. Security Considerations

1. **JWT Authentication**
   - Tokens expire in 24 hours with a refresh mechanism.
   - Secure HTTP-only cookies for storage.

2. **Rate Limiting & Caching**
   - Limits 100 requests per 15 min/IP.
   - Uses Redis for caching and rate tracking.

3. **Role-Based Access Control (RBAC)**
   - Admins: Full access.
   - Team Leads: Task creation & assignment.
   - Developers: Task updates & comments.
   - Middleware enforces permissions.

4. **Input Validation & Sanitization**
   - Flask-Validator for server-side checks.
   - CSP headers and parameterized queries.
   - HTML sanitization for comments.
   - MIME type validation for uploads.

5. **OAuth Security**
   - Uses PKCE for OAuth flow.
   - Encrypts and rotates OAuth tokens.
   - Validates state parameters and limits scopes.

8. Legal, Ethical, and Social Issues

Legal Considerations

- GDPR compliance ensures explicit user consent, right to data portability, and deletion requests.
- Licensing adherence for open-source dependencies**.**

Ethical Considerations

- Fair usage policies for data collection and user privacy.
- Avoidance of bias in task assignments and performance tracking.

Professional Considerations

- Secure handling of user authentication and sensitive data.
- Audit logging for administrative actions and security monitoring.

Social Considerations

- Accessibility features (WCAG 2.1 AA compliance) ensure inclusivity.
- Promotes collaboration through transparent task tracking and notifications.

9. Team Roles & Development Timeline

Team Roles

Role	Responsibility	Assigned To
Frontend Developer	Implement UI for Client App	Huzaifa Ijaz
Frontend Developer	Implement UI for Admin App	Martin Litvin
Backend & API Developer	Develop backend logic & API	Andre Ndong Nto
Database & Security Manager	Manage database and security	Ahmed Ikram
Full-Stack Integrator	Handle API connections & bug fixes	Theo Short
Documentation & Testing	Handle making final and code testing	Everyone

Development Timeline

Week	Task
Week 1 (w/c 24/02)	Research, Planning & Set Up Environment
Week 2 (w/c 03/03)	Develop Backend API & Database
Week 3 (w/c 10/03)	Implement Frontend Applications
Week 4 (w/c 17/03)	Integrate API & GitHub Services + Begin Testing
Week 5 (w/c 24/03)	Final Testing, Security Audits, Deployment & Documentation