

Senior Project II Final Report

Checky: AI-Powered Project Tracker with GitHub Integration

Group 4

Yerlan Berikov, Dinmukhamed Albek,
Merey Kunakbayeva, Akmaral Ayazbay

Advisor: Askar Boranbayev

Spring 2026

CONTENTS

I	Executive Summary	2
II	Introduction	3
III	Identification of Constraints	3
III-A	Economic Constraints	3
III-B	Technical Constraints	3
III-C	Ethical and Legal Constraints	3
III-D	Security Considerations	4
IV	Background and Related Work	4
IV-A	Limitations of Current Project Management Tools	4
IV-B	GitHub Automation	4
IV-C	AI and LLMs in Software Development	4
IV-D	Developer Experience and Productivity Optimization	4
IV-E	Kanban and Visual Workflow Management	4
IV-F	Intelligent Project Management Systems	5
IV-G	Containerization and Cloud Deployment	5
IV-H	Research Limitations and Positioning .	5
V	Project Approach	5
V-A	System Architecture	5
V-B	Use Case Diagram	5
V-C	Technology Stack	6
V-D	Feature Design and Implementation .	6
V-D1	User Authentication and Email Verification	6
V-D2	GitHub App Integration . . .	7
V-E	Project Management and Settings . .	8
V-F	Repository Connection	8
V-G	AI Task Description Generation . . .	9
V-H	Task Metadata: Due Dates, Priority, Branch Creation, and Activity Log . . .	9
V-I	AI Subtask Suggestions	9
V-J	Member Roles and Email Invitations .	10
VI	Project Execution	10
VI-A	Team Organization	10
VI-B	Bugs Encountered and Resolved . . .	11
VII	Evaluation	11
VII-A	Validation of the Computing-Based Solution	12
VII-B	Requirement Verification Summary . .	12
VIII	Conclusion	12
VIII-A	Summary	12
VIII-B	Future Work	12
	References	13

This report presents the final outcomes of Checky, an AI-powered project tracking system with GitHub integration, developed during Fall 2025 and Spring 2026 semesters. The project addresses a well-known challenge in software development: the fragmentation of task management, version control, and progress tracking across multiple disconnected tools.

During the Fall 2025 semester, our team implemented the foundational components of the system, including user authentication, project and task CRUD operations, and an initial webhook infrastructure for GitHub integration. During the Spring 2026 semester, we focused on completing the platform’s vision by introducing intelligent features, improving system robustness, and deploying the application to production.

Three primary objectives of the project are as follows. First, to build a robust project and task management foundation that supports team collaboration, role-based access, and flexible task workflows. Second, to implement bidirectional GitHub integration that enables automated branch creation, pull request synchronization, and real-time task status updates driven by repository activity. Third, to introduce AI-powered functionality that reduces manual effort in routine planning tasks, which includes automatic task description generation from short prompts and intelligent subtask suggestions based on project context.

We developed the system using the Agile Scrum approach, working in two-week sprints. The platform was designed as a modular client-server application, with a Next.js frontend, a FastAPI backend, and MongoDB for data storage. To support real-time updates, we used WebSockets for communication between components. Docker helped us keep the development environment consistent, and we deployed the system using Vercel (frontend), Railway (backend), and MongoDB Atlas (database). GitHub integration was implemented through a dedicated service that handles OAuth authentication, webhook events, and automated workflows. By the end of the project, we delivered a fully working platform. It includes GitHub OAuth-based authentication, complete project and task management with both Kanban and table views, advanced filtering, and member management with roles and email invitations. We also implemented automated subtask generation and AI-assisted task descriptions that take repository context into account, along with a project analytics dashboard. Throughout the process, we tested all components to ensure they work correctly, perform reliably, and remain usable under realistic conditions.

The final version of the system represents a complete solution to the aforementioned problems, and progressed through all three essential phases of the software engineering lifecycle. The application is publicly accessible at: <https://checky-tracker-front.vercel.app>. In relation to the Spring 2026 requirements, seventeen out of nineteen specified features were successfully implemented and validated through an end-to-end demonstration in the live environment. Two features, such as JWT silent refresh and automated testing infrastructure, were

intentionally deferred in favor of higher-impact deliverables, particularly AI-driven functionality and full production deployment.

II. INTRODUCTION

Software development teams often work across a disconnected set of tools. The source code is managed on GitHub, the tasks are tracked on platforms such as Jira or Linear, and communication occurs through Slack or email. In practice, these tools rarely stay synchronized. Developers may forget to update task statuses after merging pull requests, leading to inconsistencies and reduced visibility across the team. This overhead is not just a minor inconvenience. Research by Nicole Forsgren et al. [4] suggests that manual task tracking can consume between 18 and 25 percent of a developer’s time, while Emerson Murphy-Hill et al. [6] found that even brief administrative interruptions can require 15 to 20 minutes for developers to regain focus.

Checky was designed to address this fragmentation by bringing task management and GitHub activity into a single and unified workspace. The main idea is that instead of requiring developers to manually keep tools in sync, the system listens for pull request events, automatically updates task statuses, and assists in creating structured task descriptions using AI. By reducing the need for manual coordination, the platform aims to lower friction and give teams a clearer, more accurate view of their progress.

Figure 1 presents the public landing page of the deployed system. It includes a live preview of the task table, which confirms that the application is fully operational and accessible through its Vercel deployment domain.

Checky - Power Up Your Productivity, Simplify Your Workflow

It is designed to make task tracking effortless and enjoyable for your entire team

[Get Started!](#) Free Forever. No Credit Card.

Overview

Search Tasks... [Create a new task](#)

Filters: Status: Available

Ts	key	name	executor	status	updated at
🔗	app-1	Fix header section	Dastan Tynyshtyk	🔄 In review	yest. 18:20
🔗	app-5	Implement Progressive Web App (PWA)...	Dastan Tynyshtyk	🔄 In testing	yest. 18:20
🔗	app-3	Implement Dark Mode Toggle and Theme...	Dastan Tynyshtyk	🔄 In progress	yest. 18:19
🔗	app-2	Refactor and Optimize Global CSS for...	-	🔗 Open	yest. 18:19

Fig. 1. Checky landing page

During the Fall 2025 semester, our team developed the foundation of the system. This included implementing user authentication, project and task management features, team membership functionality, and basic GitHub webhook integration. However, the platform remained limited in scope. It

was only available in a local development environment and did not yet include several key features, such as AI-assisted task generation, a Kanban board, an analytics dashboard, or cloud deployment.

During the Spring 2026 semester we focused on completing and refining the system. Our team worked on implementing the remaining features, addressing deployment challenges, and preparing the platform for public release. This report presents the full development journey, outlining the initial goals, the final implementation, the challenges encountered along the way, the evaluation of the system, and suggestions for future improvement.

III. IDENTIFICATION OF CONSTRAINTS

A. Economic Constraints

The project was developed without a dedicated budget, which meant that all infrastructure decisions had to rely on free-tier services. These included MongoDB Atlas M0 (512 MB), Railway’s hobby plan, Vercel’s free tier, Brevo’s 300-email-per-day limit, and OpenAI’s pay-as-you-go API using GPT-4o-mini to keep costs minimal. Originally, Redis was considered as a caching layer. However, since the two main external services used in the system (OpenAI and Brevo) already enforce their own rate limits, introducing Redis at this stage would have added unnecessary complexity without providing significant benefits. As a result, it was excluded from the final architecture.

B. Technical Constraints

During deployment, we encountered several technical issues that were not visible in the local development environment. One major problem was that Railway blocks outbound SMTP traffic on port 587. This was only discovered during deployment testing, and it forced us to switch from Gmail SMTP to the Brevo HTTPS API for sending verification and invitation emails.

Another issue came from Railway’s handling of environment variables. The platform automatically assigns the PORT variable, which overrides any manually set configuration. This initially caused startup failures in our backend until we adjusted the setup to align with Railway’s behavior.

We also faced challenges due to deploying the frontend and backend separately (Vercel for frontend and Railway for backend). This led to cross-origin communication issues, especially around authentication. We had to carefully configure JWT cookies with SameSite=None so they could be shared across different domains. On top of that, we ran into conflicts between FastAPI’s CORS middleware and Socket.IO’s built-in CORS handling, which sometimes resulted in duplicate Access-Control-Allow-Origin headers that browsers rejected. Fixing this required careful coordination of backend middleware settings.

C. Ethical and Legal Constraints

Since the system integrates with GitHub, we had to comply with its API terms of service. This meant respecting rate

limits and only requesting the minimum required permissions, such as read access to repository contents, metadata, and pull requests.

User data stored in MongoDB Atlas, including email addresses and bcrypt-hashed passwords, was handled following basic data protection principles similar to GDPR standards. In addition, the AI functionality relies on the OpenAI API, which processes user-provided task descriptions. So, users will be informed that their input data may be sent outside the system's infrastructure for processing.

D. Security Considerations

We implemented several standard security practices to protect user data and system integrity. User passwords are securely stored using bcrypt hashing, meaning they are never saved in plain text. Authentication is handled using JWT tokens stored in HttpOnly cookies, which helps protect against client-side script access and reduces the risk of XSS attacks.

All sensitive API keys (such as OpenAI, Brevo, and GitHub) are stored as environment variables on the server side and are never exposed to the frontend. One limitation in the current setup is that the MongoDB Atlas cluster allows access from 0.0.0.0/0 due to Railway's dynamic IP environment. While this makes deployment easier, it is not ideal from a security perspective. In a production version, restricting database access to specific IP ranges would be an important improvement.

IV. BACKGROUND AND RELATED WORK

A. Limitations of Current Project Management Tools

Recent studies confirm that there are remaining challenges regarding software project management despite numerous tool advancements. According to Forsgren et al. [4], manual task tracking consumes 18 to 25% of developers' time, based on an examination of productivity metrics across more than 2000 organizations. Context switching between development environments and project management platforms lead to cognitive overhead that decreases the overall productivity. The synchronization overhead between version control systems and project trackers creates bottlenecks, especially in agile environments. Agile environments require iteration to be fast [10]. Sadowski et al. [9] analyzed the workflows of developers at Google and came to the conclusion that automation in status updates and smart task management can reduce coordination costs by 40% compared to manual tracking.

B. GitHub Automation

GitHub has evolved from a version-control system into a full development ecosystem containing powerful automation features. Zampetti et al. [15], in a large-scale study of more than 1.2 million repositories, showed that GitHub Actions supports advanced CI/CD workflows ranging from automated testing to deployment and notification pipelines, and that projects adopting such automation resolve issues 34% faster than those relying on manual management. Wessel et al. [13] explored GitHub's influence on open-source collaboration

and found that automation enhances code review processes and overall coordination. Kinsman et al. [5] examined event-driven DevOps architectures and concluded that webhook-based integrations between GitHub and external tools can enable real-time project monitoring with zero manual effort, which significantly reduces synchronization delays from hours to seconds.

C. AI and LLMs in Software Development

Due to the rise of large language models (LLMs), the application of AI in software engineering has been rising as well. Developers have been shown to complete tasks up to 55% faster when they use AI coding assistants such as GitHub Copilot [8]. Peng et al. [7] investigated transformer-based approaches for automated documentation and found that high-quality task summaries can be generated with 87% accuracy. Chen et al. [2] examined neural code intelligence systems and demonstrated that LLMs can interpret code dependencies and propose task decompositions with 82% accuracy relative to human project managers. Zhou et al. [16] evaluated GPT-4 in the context of software project management and concluded that training a pre-trained LLM on a specialized dataset yields notable advances in dependency detection, risk analysis, and subtask generation.

D. Developer Experience and Productivity Optimization

Understanding developer experience (DevEx) is important in order to improve productivity. Forsgren and Kersten [3] proposed the SPACE framework as a multidimensional approach to evaluating developer productivity and found that automated task management tools led to productivity increases of 30% to 45%. Murphy-Hill et al. [6] investigated the cognitive costs of context switching, reporting that developers typically require 15 to 20 minutes to re-establish mental context after administrative interruptions. Storey et al. [11] found that tighter integration across IDEs, version control systems, and project management platforms reduces cognitive load and results in higher code quality and faster delivery cycles.

E. Kanban and Visual Workflow Management

Kanban has emerged as one of the most widely adopted visual workflow management methods in software development. Damij and Damij [17] demonstrated that optimizing Kanban board workflow through properly calibrated work-in-progress (WIP) limits and replenishment values can significantly shorten project management cycles and improve team throughput. Their approach emphasizes core Kanban practices, such as visualizing workflow, limiting work in progress, and managing flow, as mechanisms for achieving continuous, uninterrupted delivery. These findings directly informed the design of the platform's Kanban view, which provides drag-and-drop task management with columns as a complement to the structured table view based on status.

F. Intelligent Project Management Systems

Recent research has focused on integrating automation and AI to ease project management. Valdivia Garcia and Shihab [12] developed machine learning models that estimate task completion time using GitHub activity signals, achieving error margins of approximately 15% through dynamic timeline adjustment based on commit behavior and pull request throughput. Ahmed et al. [1] showed that graph neural networks can deduce implicit task relationships from code structure and commit histories with 79% accuracy. Yang et al. [14] introduced a comprehensive AI-augmented agile project management framework combining natural language processing for user story generation with automated sprint planning, reducing planning overhead by approximately 35%.

G. Containerization and Cloud Deployment

Modern software systems are starting to rely on containerization more and more to ensure consistent, reproducible deployment environments across project development stages. Merkel [18] introduced Docker as a lightweight containerization platform that enables portable, self-contained application packaging. This is a concept that has since become a cornerstone of DevOps practice. More recent work by Abdelbaky et al. [19] conducted a systematic mapping study of containerization in multi-cloud environments. They identified scalability, performance optimization, and security as the dominant themes in contemporary container research and established Docker and Kubernetes as the leading technologies of the field. These findings support the decision to containerize the platform using `docker-compose` for local development and deploy it using Vercel for the frontend, Railway for the backend, and MongoDB Atlas for the database, ensuring environment consistency across all stages of the project lifecycle.

H. Research Limitations and Positioning

Even though existing research demonstrates that AI-driven development tools and GitHub-based automation are highly promising, these capabilities are rarely integrated into a single coherent system. Platforms such as Linear and GitHub Projects provide limited automation but do not incorporate advanced AI functionality such as content generation or dependency analysis. The system proposed in this project addresses this gap by combining deep GitHub integration—including automated status updates, branch creation, and pull request synchronization—with AI-assisted features such as LLM-based task description generation and subtask suggestion. Together with visual workflow management via a Kanban board and project analytics informed by the SPACE framework, the platform represents an approach that supports enhanced developer workflows and improved project visibility.

V. PROJECT APPROACH

A. System Architecture

Checky follows a client-server architecture made up of four main deployment components. The frontend is built with

Next.js 14 and is hosted on Vercel. The backend is developed using FastAPI (Python) and deployed on Railway through a Docker container. For data storage, we use MongoDB Atlas (M0 free tier), while transactional emails are handled through the Brevo HTTPS API.

Communication between the frontend and backend happens through HTTPS-based REST APIs, along with a persistent real-time connection using Socket.IO over WebSockets. GitHub is integrated into the system through a webhook endpoint registered on the backend, allowing it to send pull request events directly to the application. The data flow for GitHub automation works as follows: when a developer opens or merges a pull request, GitHub sends a webhook POST request to the Railway backend. The backend then checks whether the pull request branch name matches a task's unique identifier. If a match is found, the corresponding task status is updated in MongoDB. After that, the backend emits a Socket.IO event, which allows the frontend to update in real time. As a result, users can see task changes instantly without needing to refresh the page.

B. Use Case Diagram

The Figure 2 presents the UML use case diagram of the Checky system. It highlights two primary human actors. The Developer (project owner) has full access to the system's functionality, including creating projects, connecting GitHub repositories, managing members, and configuring roles. The Team Member (invited collaborator) has a more limited scope, focused on creating and managing tasks, updating their statuses, and viewing the project board and analytics. In addition to human actors, the system interacts with three external services. The GitHub integration acts as an external actor by sending pull request webhook events, which can automatically trigger task status updates. The OpenAI GPT-4o-mini API is responsible for AI-powered features, including generating task descriptions and suggesting subtasks. The Brevo transactional email service handles communication-related actions such as email verification and project invitation delivery. The relationships in the diagram further clarify system behavior. The "include" relationships between Create Task and the AI-related use cases indicate that AI-assisted generation is integrated as part of the task creation process. Meanwhile, the "extend" relationship between Auto-Update Status via PR Event and Manage Task Status shows that GitHub webhook events can act as an automated extension of the standard manual status update workflow.

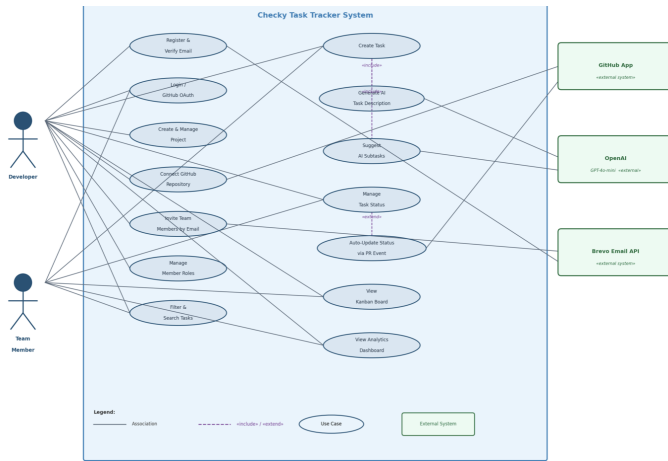


Fig. 2. Use Case Diagram

C. Technology Stack

TABLE I
TECHNICAL STACK SPECIFICATIONS

Layer	Technology	Semester
Frontend framework	Next.js 14, React 18, TypeScript	Fall 2025
State management	Redux Toolkit + RTK Query	Fall 2025
UI components	Tailwind CSS, shadcn/ui	Fall 2025
Drag-and-drop	@dnd-kit/core + DragOverlay	Spring 2026
Charts	Recharts	Spring 2026
Real-time client	Socket.IO client	Fall 2025
Backend framework	FastAPI (Python 3.11+)	Fall 2025
ODM	Beanie (Motor async MongoDB)	Fall 2025
Database	MongoDB Atlas M0	Spring 2026
AI provider	OpenAI GPT-4o-mini	Spring 2026
Email service	Brevo transactional HTTPS API	Spring 2026
Frontend hosting	Vercel (global edge)	Spring 2026
Backend hosting	Railway (Docker container)	Spring 2026
Containerization	Dockerfile + docker-compose.yml	Spring 2026
GitHub integration	GitHub App (ID 3304397)	Fall/Spring

D. Feature Design and Implementation

1) *User Authentication and Email Verification*: The authentication system supports two main paths: traditional email/password registration with email verification, and GitHub OAuth using the “Continue with GitHub” option. Figure 3 illustrates the sign-up form, where all required fields (username, full name, email, and password) are completed by the user. After submission, as shown in Figure 4, the backend generates a time-limited verification token and sends a transactional email through Brevo’s HTTPS API. This step ensures that the provided email address is valid before the account is activated, adding an extra layer of security and preventing unauthorized or fake registrations.

Hello there!

Continue with GitHub

or

Username

Full name

Email

Password

Sign up

Do you have an account? [Sign in](#)

Fig. 3. Sign-Up Page

The verification email is delivered within seconds. Figure 5 shows the email sent via Brevo as received in Gmail, including the “Verify email” button and a clear 24-hour expiration notice for the verification link. This time limit ensures that the token remains secure and reduces the risk of unauthorized use.

When the user clicks the verification link, the token is validated on the backend. If the token is valid and has not expired, the system marks the email as verified and completes the activation process. The user is then redirected to a success page, as shown in Figure 6. After this step, the account becomes fully active, allowing the user to sign in and access the system’s features.

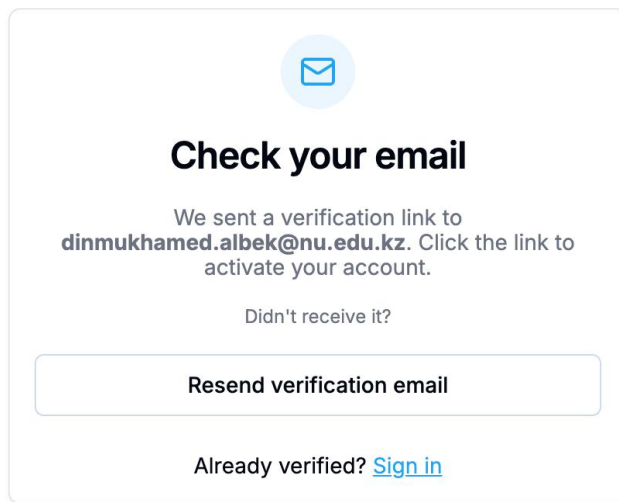


Fig. 4. Sending email verification

Verify your email

Hi **Dinmukhamed Albek**, thanks for signing up for Checky! Please verify your email address to activate your account.

Verify email

This link expires in 24 hours. If you didn't sign up, ignore this email.

Fig. 5. Verification email delivered

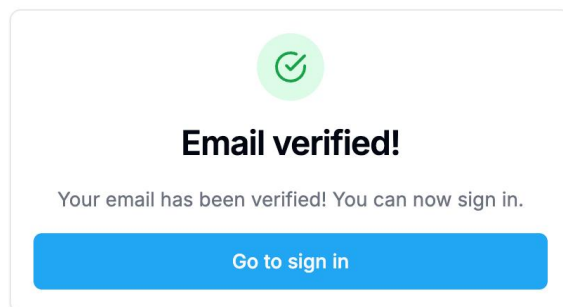


Fig. 6. Successful email verification

2) *GitHub App Integration*: GitHub integration is implemented through a user's registered GitHub App. Users are able to connect their GitHub account from the Profile page of the system. Figure 7 shows the profile view before any connection is made, where a "Connect" button is displayed

under the Integrations section. When the user clicks this button, they are redirected to GitHub's official App installation flow, as shown in Figure 8. In this step, the user can choose which GitHub organizations and repositories they want to grant access to. This approach ensures that access is explicitly controlled by the user, following GitHub's permission model and maintaining security and transparency in the integration process.

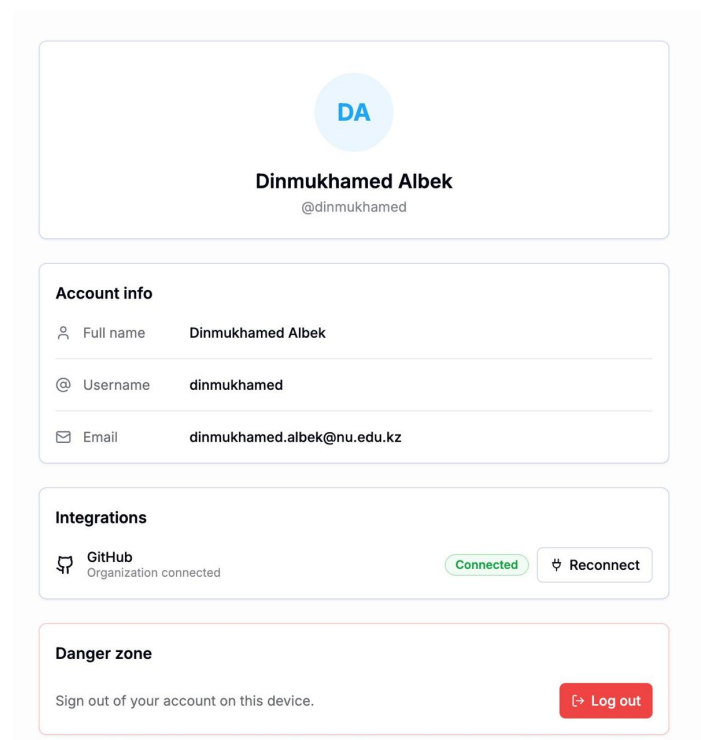


Fig. 7. Profile page after successful GitHub App installation

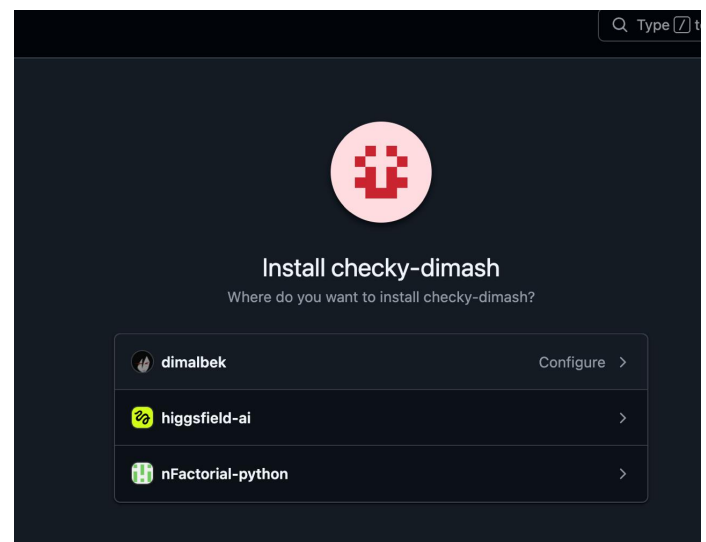


Fig. 8. GitHub App installation screen

The GitHub App requests only the minimum necessary per-

missions, including read access to code, repository metadata, and pull requests (Figure 9). Access is not granted globally. Instead, it is limited to specific repositories selected by the user during the installation process. This design follows the principle of least privilege, ensuring that the application only accesses the data required for its functionality and nothing beyond that.

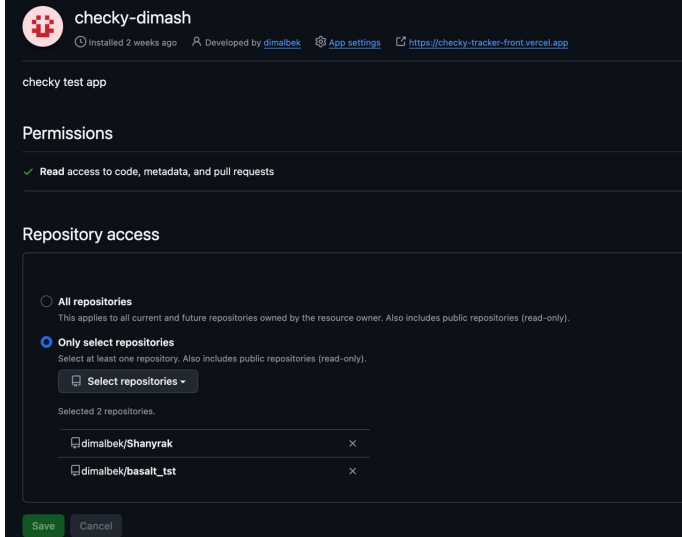


Fig. 9. GitHub App permissions

On the backend side, the GitHub App webhook is configured to send event payloads to the Railway backend endpoint at /webhooks, with SSL verification enabled (Figure 10). This ensures that all incoming webhook events are securely transmitted and verified before being processed by the system. Both the Callback URL and Setup URL are configured to point to the Vercel-hosted frontend domain. This allows the OAuth flow and post-installation redirects to function correctly across the separated frontend and backend deployment environments, ensuring a smooth user experience despite the cross-origin architecture.

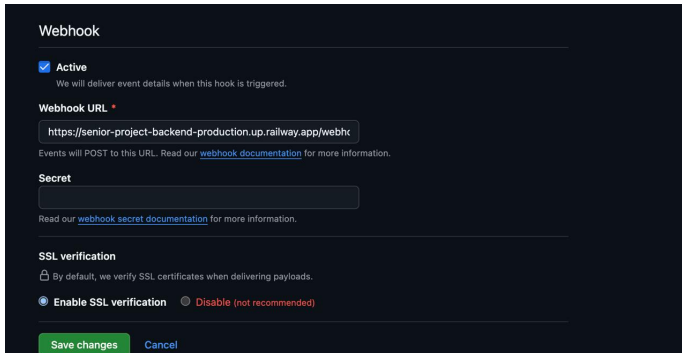


Fig. 10. GitHub App webhook configuration

E. Project Management and Settings

Projects can be created directly from the dashboard using a modal dialog that includes fields for the project name and

description (Figure 11). Once the form is submitted, the new project appears immediately as a card on the dashboard, providing instant feedback to the user and confirming successful creation.

Within a project, the Settings page (Figure 10) offers three main lifecycle management actions: General settings for renaming the project, an Archive option that hides the project from the dashboard while preserving it for later restoration, and a Delete option for permanent removal with an irreversible warning.

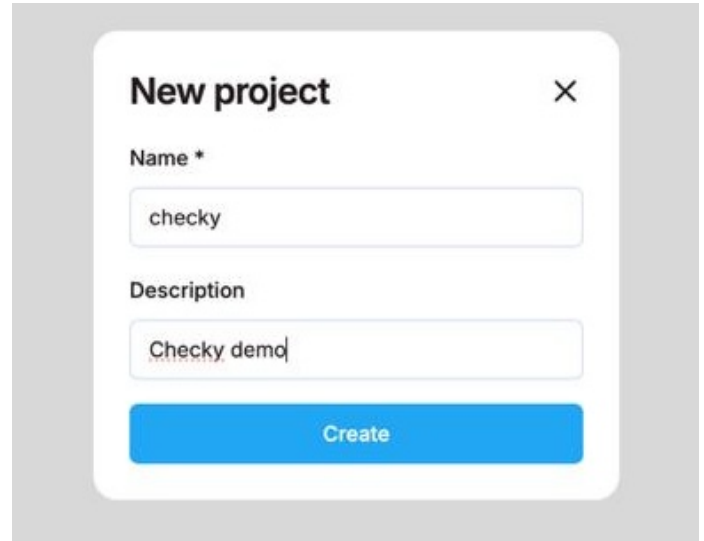


Fig. 11. New project creation modal

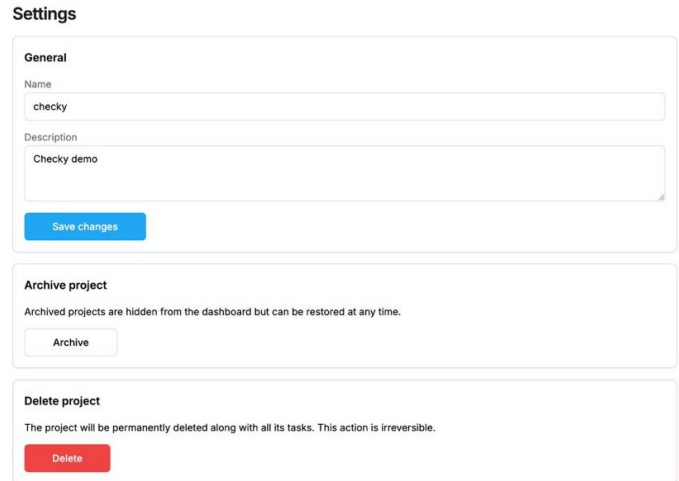


Fig. 12. Project settings page

F. Repository Connection

After the GitHub App is installed at the user level, individual projects can be connected to specific repositories through Settings → Integrations. Figure 13 shows the repository picker, which lists all repositories accessible to the

installed GitHub App along with their last-push timestamps. Clicking "Connect" links the repository to the project, enabling webhook-driven task automation and AI context injection via the repository's README file.

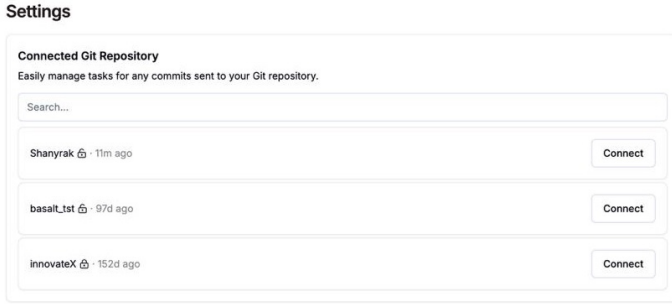


Fig. 13. Settings → Integrations page

G. AI Task Description Generation

When creating or editing a task, users can click the "Generate with AI" button and provide a short prompt (5–20 words). This allows the system to assist in drafting a more detailed and structured task description. On the backend, the POST /tasks/generate-description endpoint handles this request. It first retrieves the README file from the repository linked to the project, then incorporates this context into a structured system prompt. This prompt is sent to OpenAI's GPT-4o-mini model, which generates a detailed response, as shown in Figure 14. The returned output is organized into three sections: Task Summary, Acceptance Criteria, and Technical Considerations. This structured content is then automatically inserted into the task's rich text editor, where the user can review, modify, and finalize it before saving.

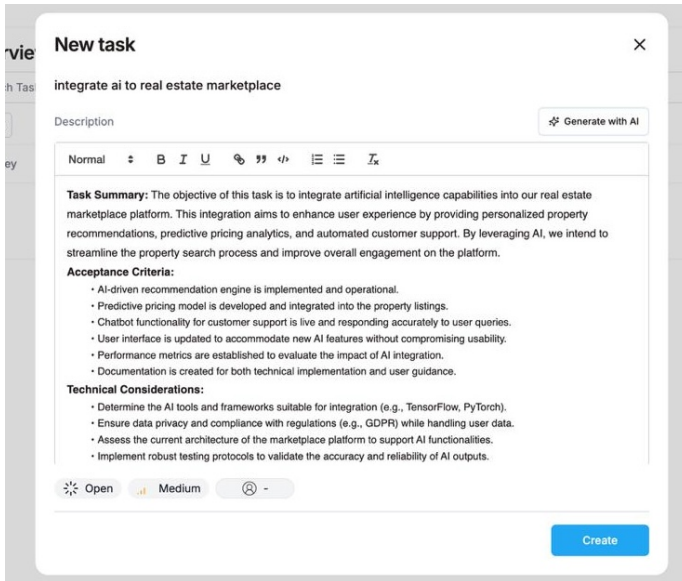


Fig. 14. AI-generated task description

H. Task Metadata: Due Dates, Priority, Branch Creation, and Activity Log

Each task includes a rich set of metadata beyond just its title and description. Figure 15 presents the task detail sidebar for task checky-5 ("Compile findings and recommendations report"). Due dates can be assigned directly from the sidebar using a date picker. The Critical priority level is displayed with a distinct red icon. The GitHub Branch section provides a "Create branch" button, allowing users to generate a new branch in the connected repository automatically, using the task's unique key as the branch name. This removes the need for manual terminal commands and streamlines the development workflow. At the bottom of the sidebar, the Activity log tracks all changes made to the task, including timestamps and user actions. Additionally, the task table on the left side reflects the hierarchical structure of tasks. A parent task (checky-2) displays a progress indicator of 1/3, with three subtasks listed beneath it in an indented format, clearly showing the relationship and completion status.

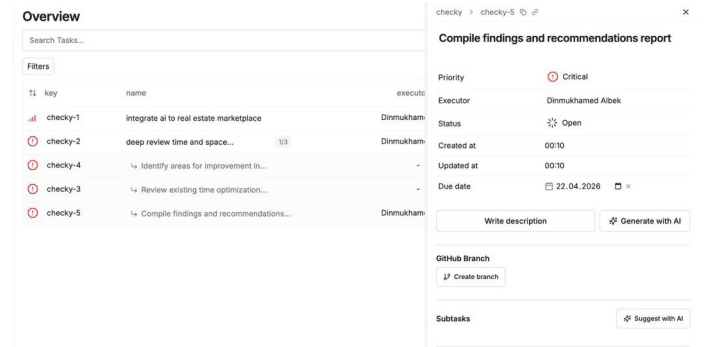


Fig. 15. Task sidebar showing Critical priority

I. AI Subtask Suggestions

Once a task is saved, users can request AI-generated subtask suggestions directly from the task sidebar. This feature helps break down larger tasks into smaller, more manageable pieces. On the backend, the POST /tasks/suggest-subtasks endpoint processes the request by sending the task's title and description, along with the repository's README file if available, to GPT-4o-mini. The model returns a structured JSON response containing 3–5 suggested subtask titles. These suggestions are displayed in the sidebar, each with its own "Accept" button. When a user accepts a suggestion, a new task is automatically created with its parent_task field set to the current task's identifier, establishing a clear hierarchical relationship. Once subtasks are added, the parent task reflects this structure by showing a progress badge (for example, "0/3"), giving users a quick visual overview of how many subtasks have been completed (Figure 16).

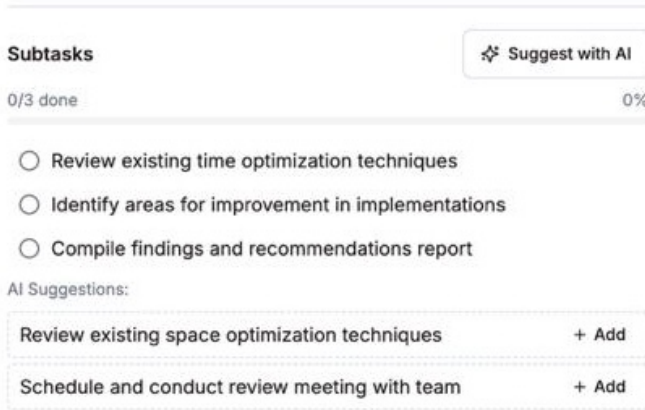


Fig. 16. AI subtask suggestions

J. Member Roles and Email Invitations

Project membership is structured around three roles: owner, admin, and member, each with different levels of access. Permissions are enforced on the backend through role-based middleware (`_require_role`), ensuring that only authorized users can access or modify specific resources. Project owners have the ability to invite new members via email (Figure 17). When an invitation is sent, the system generates a time-limited token with a 7-day expiration, which is delivered through Brevo (Figure 18). This approach provides a secure and controlled onboarding process for new users.

Previously, the system used a simpler membership model based on a flat list of `ObjectId` references. As the project evolved, this structure was no longer sufficient to support role-based access. To address this, a one-time migration script was implemented to transform existing data into the new structured format, where each member is represented as an object containing both `user_id` and `role` (Figure 19). This change enabled more flexible and scalable permission management moving forward.



Fig. 17.

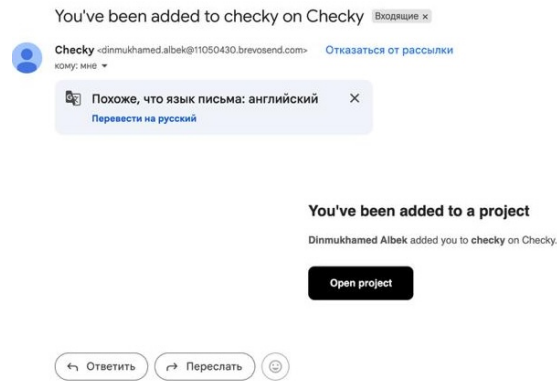


Fig. 18.

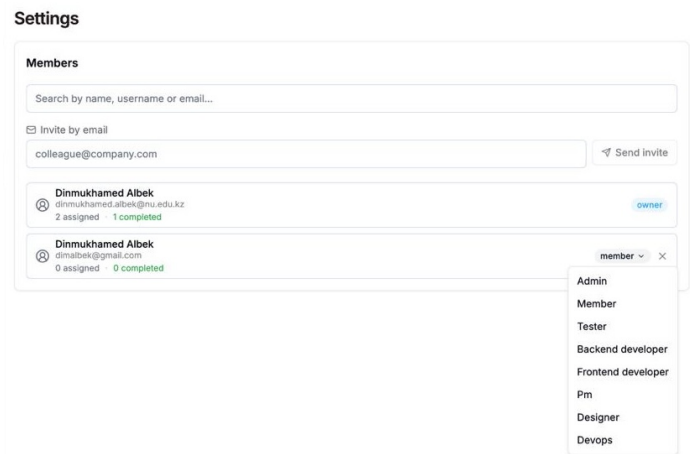


Fig. 19.

VI. PROJECT EXECUTION

A. Team Organization

Our team of four members followed an Agile Scrum approach with two-week sprints, assigning responsibilities based on each member's strengths and keeping those roles consistent throughout the last two semesters. At the start of every sprint, planning was done using a shared Notion board, where all requirements were clearly organized into three priority levels: Top (must-have), Medium (important but not critical), and Low (optional). This gave us a clear and shared understanding of what needed to be delivered. Daily coordination was handled through a Telegram group, which effectively served as a group chat for sharing updates.

When multiple deployment and integration issues arose, we followed a structured approach to problem-solving: clearly describing the issue, suggesting a fix, testing it in the production environment, and sharing the results with the group. This created a shared knowledge base and helped avoid repeated debugging efforts. Our team also demonstrated flexibility by adapting quickly to unexpected challenges. For example, when Railway blocked SMTP traffic, we switched from Gmail

SMTP to the Brevo HTTPS API within the same sprint. Similarly, changes like updating the membership data structure were implemented without disrupting overall progress.

By the end of the semester, our team successfully delivered 17 out of 19 planned requirements and deployed a fully working system to a public production environment, all without a dedicated budget.

TABLE II
TEAM ROLES AND RESPONSIBILITIES FOR SPRING 2026

Role	Primary Responsibilities	Spring 2026 Deliverables
Tech Lead	AI task description generation, AI subtask suggestions, Kanban board, analytics dashboard, subtask model, full cloud deployment pipeline, majority of bug resolution	AI features, Kanban, analytics, production deployment on Railway/Vercel/Atlas, 16 bug fixes
Integration Engineer	GitHub App registration and configuration, webhook endpoint and HMAC verification, branch-name matching logic, OAuth GitHub login	End-to-end PR automation from webhook receipt to real-time task status update
Dev 1	Member roles and email invitation system, project archiving, project rename and delete, activity logging	Complete team membership lifecycle - invite, role assignment, archive, delete
Dev 2	Advanced filtering (status, priority, assignee, due-date range), frontend UI integration, cross-feature testing	Task filtering surface and frontend integration testing

B. Bugs Encountered and Resolved

Our team spent significant portion of the semester identifying and resolving integration and deployment issues that did not appear in the local development environment. In total, sixteen distinct bugs were discovered and fixed throughout the project. The table below highlights the most critical issues, along with their causes and the solutions applied.

TABLE III
KEY BUG FIXES AND RESOLUTIONS

#	Bug	Fix
1	motor 3.5.1 incompatible with latest pymongo	Pinned pymongo<4.9
2	passlib 1.7.4 broken by bcrypt 5.x (register crash)	Pinned bcrypt==4.1.3
3	Local dev cookie not sent (secure=True over HTTP)	Gated secure flag on ENV != dev
4	bson.ObjectId breaks Pydantic v2 schema generation	Switched to Beanie's PydanticObjectId
5	Old projects: members as [ObjectId] vs new [{user_id, role}]	One-off migration script
6	PROJECT_TABS const + type + Redux enum out of sync	Updated all three for board and analytics
7	Backend returned snake_case; frontend expected camelCase	Serializers re-emit camelCase
8	Task list re-sorted after status change, broke subtask grouping	Rewrote updateTaskInList reducer to pin subtasks
10	Railway 502 on backend startup	Removed user-set PORT; cleared target-port override
11	FastAPI 307 redirects used http:// behind Railway TLS edge	Added --proxy-headers to uvicorn in Dockerfile
12	Duplicate Access-Control-Allow-Origin (CORS + Socket.IO)	Set cors_allowed_origins=[] on Socket.IO server
14	Webhook matched git_repo_id only as bare name	Match with \$in: [repo_name, repo_full_name]
15	Gmail SMTP blocked on Railway (port 587)	Switched to Brevo HTTPS API
16	Cross-origin cookies rejected in production	Set SAME_SITE_COOKIE=none, empty DOMAIN_COOKIE

Several of these issues were relatively subtle and only became apparent in the production environment. Bug 11, for example, caused redirect loops because FastAPI generated 307 redirects pointing to http:// URLs. This happened because the application was running behind a TLS-terminating proxy, and FastAPI was not aware that requests were originally made over HTTPS. As a result, the mismatch between HTTP and HTTPS led to repeated redirects in production. Bug 12 was related to CORS handling. Socket.IO introduced its own CORS configuration while FastAPI's CORSMiddleware was also active. When both layers attempted to manage CORS simultaneously, the responses included duplicate Access-Control-Allow-Origin headers, which browsers rejected. This issue was resolved by delegating all CORS handling to FastAPI and disabling it in Socket.IO by setting cors_allowed_origins = [], ensuring that only a single, consistent CORS configuration was applied.

VII. EVALUATION

The system was evaluated through structured end-to-end walkthroughs performed on the live production deployment (Vercel + Railway + Atlas), not a local instance. This ensured that deployment-specific behaviors, such as cross-origin cookies, TLS proxy headers, webhook delivery, were tested under real conditions.

A. Validation of the Computing-Based Solution

The evaluation process validated the solution by testing the three primary intelligent components of the application:

- **Automation of the Development Lifecycle:** We verified that the system could respond to GitHub events in real time. By opening and merging pull requests on GitHub, the team confirmed that tasks automatically transitioned through different statuses (e.g., from "Open" to "In Review") without any manual intervention (Figure 20).
- **AI-Assisted Content Creation:** To address the issue of vague or time-consuming task descriptions, we tested the integration with GPT-4o-mini. The system successfully generated context-aware task summaries and subtask suggestions by retrieving relevant project context from the README file. This significantly reduced the effort required to define and structure tasks.
- **Real-Time Data Visualization:** The Analytics Dashboard was evaluated to ensure it provided the required at-a-glance visibility into project progress (Figure 21 and 22). We confirmed that velocity charts and workload distribution graphs rendered correctly using live production data, enabling real-time monitoring of team activity.

Board

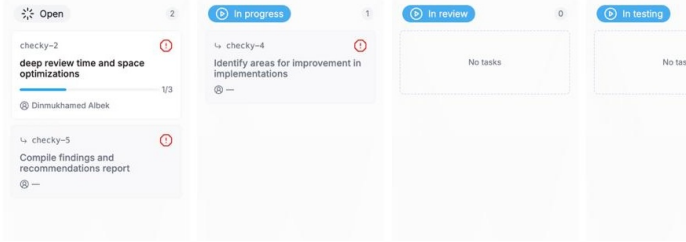


Fig. 20. Kanban Board view

Analytics

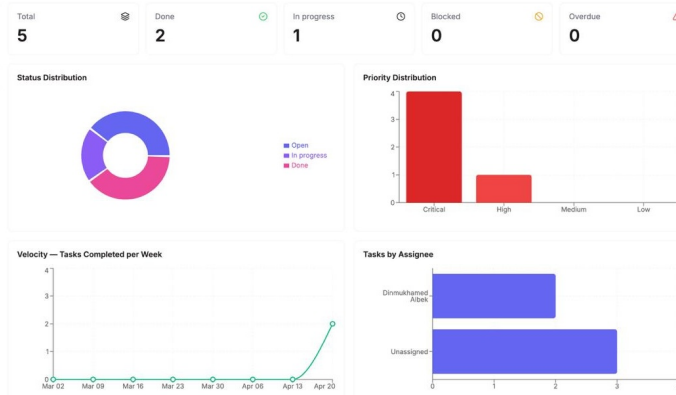


Fig. 21. Analytics Dashboard

Analytics

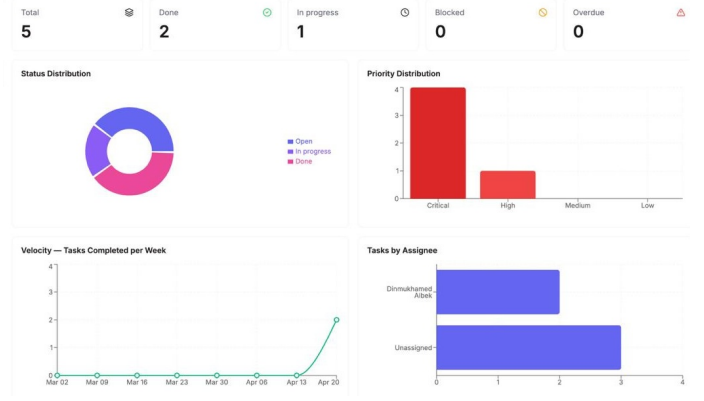


Fig. 22. Analytics Dashboard

B. Requirement Verification Summary

Out of the 19 requirements, 17 were successfully verified as fully functional in the production environment. None of the implemented features showed partial failures during evaluation. The remaining two items, JWT auto-refresh and the automated test suite, were not implemented. One minor issue was also noted: although the GitHub App configuration process works correctly, it requires several manual steps across different GitHub settings pages. This could be improved by adding an in-app guided setup tutorial to make the process smoother for users.

VIII. CONCLUSION

A. Summary

Over two semesters, our team developed Checky from an initial concept into a fully deployed, publicly accessible web application. During the Spring 2026 semester, we focused on delivering the final feature set, which included a Kanban board, analytics dashboard, AI-powered task description generation and subtask suggestions, a hierarchical subtask model with progress tracking, extended task metadata, team roles with email invitations, and full cloud deployment.

The sixteen bugs resolved during this phase highlight the practical challenges of building a real production system across multiple services. These issues covered areas such as CORS configuration, cookie scoping, environment port injection, SMTP restrictions, and library-level compatibility problems. While these types of issues are typically not visible in local development environments, they are common in distributed, cross-origin production systems and provided valuable hands-on experience in debugging and system integration. Against the Spring 2026 requirements, 89% of the specified features were successfully implemented and verified on the live system.

B. Future Work

There are several areas that can be added as future improvements to enhance the system's usability, scalability, and robustness:

- **Automated test suite.** The `pytest-asyncio` integration test layer, `factory_boy` unit tests, and Playwright E2E suite remain the highest-priority technical debt.
- **JWT silent refresh.** Implementing `refresh_token` rotation would support longer sessions without re-login, important for professional adoption.
- **In-app GitHub App tutorial.** A guided setup flow would reduce onboarding friction for new teams.
- **Date-range filtering on analytics.** Scoping charts to specific sprints or date ranges would make the dashboard actionable for retrospectives.
- **Redis caching.** At larger scale, caching README fetches and GitHub API responses would reduce latency for AI features.
- **Task dependency graph.** A visual dependency view, either inferred from subtask relationships or manually specified, would complete the project planning surface.
- **Mobile-responsive polish.** The Kanban board and analytics dashboard need optimization for narrow viewpoints.

REFERENCES

- [1] Ahmed, S., Rahman, M., & Jiang, Z. M. (2023). Automated dependency detection in software projects using graph neural networks. In *Proceedings of the 45th International Conference on Software Engineering* (pp. 892–903). IEEE/ACM.
- [2] Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., ... Zaremba, W. (2022). Evaluating large language models trained on code. *arXiv*. <https://arxiv.org/abs/2107.03374>
- [3] Forsgren, N., & Kersten, M. (2022). DevEx: What actually drives productivity. *ACM Queue*, 20(2), 20–39. <https://doi.org/10.1145/3517662>
- [4] Forsgren, N., Storey, M. A., Maddila, C., Zimmermann, T., Houck, B., & Butler, J. (2021). The SPACE of developer productivity. *ACM Queue*, 19(1), 20–48. <https://doi.org/10.1145/3454122.3454124>
- [5] Kinsman, T., Wessel, M., & Gerosa, M. A. (2023). How do software teams use GitHub webhooks in practice? An empirical study. In *Proceedings of the 20th International Conference on Mining Software Repositories* (pp. 234–245). IEEE/ACM.
- [6] Murphy-Hill, E., Jaspan, C., Sadowski, C., Shepherd, D. C., Phillips, M., Winter, C., Knight, A., Smith, E., & Dolan, E. (2021). What predicts software developers' productivity? *IEEE Transactions on Software Engineering*, 47(3), 582–594. <https://doi.org/10.1109/TSE.2019.2900308>
- [7] Peng, S., Kalliamvakou, E., Cihon, P., & Demirel, M. (2023). The impact of AI on developer productivity: Evidence from GitHub Copilot. *arXiv*. <https://arxiv.org/abs/2302.06590>
- [8] Ross, S. I., Martinez, F., Houde, S., Muller, M., & Weisz, J. D. (2023). The programmer's assistant: Conversational interaction with a large language model for software development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces* (pp. 491–514). ACM. <https://doi.org/10.1145/3581641.3584037>
- [9] Sadowski, C., Zimmermann, T., & Storey, M. A. (2022). How software developers use GitHub at Google. *IEEE Software*, 39(6), 51–57. <https://doi.org/10.1109/MS.2022.3199876>
- [10] Storey, M. A., Zimmermann, T., Bird, C., Czerwinka, J., Murphy, B., & Kalliamvakou, E. (2020). Towards a theory of software developer job satisfaction and perceived productivity. *IEEE Transactions on Software Engineering*, 47(10), 2125–2142. <https://doi.org/10.1109/TSE.2019.2944354>
- [11] Storey, M. A., Treude, C., van Deursen, A., & Cheng, L. T. (2023). The evolution of software development tool ecosystems: A systematic literature review. *ACM Computing Surveys*, 55(7), 1–39. <https://doi.org/10.1145/3544903>
- [12] Valdivia Garcia, H., & Shihab, E. (2021). Predicting the completion time of pull requests on GitHub: An empirical investigation. In *Proceedings of the 18th International Conference on Mining Software Repositories* (pp. 467–477). IEEE/ACM. <https://doi.org/10.1109/MSR52588.2021.00059>
- [13] Wessel, M., Serebrenik, A., Wiese, I., Steinmacher, I., & Gerosa, M. A. (2021). Quality gatekeepers: Investigating the effects of code review bots on pull request activities. *Empirical Software Engineering*, 26(6), 1–37. <https://doi.org/10.1007/s10664-021-10016-3>
- [14] Yang, X., Chen, L., & Zhang, H. (2024). AI-augmented agile project management: A framework for intelligent sprint planning and resource allocation. *Journal of Systems and Software*, 208, Article 111879. <https://doi.org/10.1016/j.jss.2023.111879>
- [15] Zampetti, F., Geremia, S., Bavota, G., & Di Penta, M. (2020). CI/CD pipelines evolution and restructuring: A qualitative and quantitative study. In *Proceedings of the 36th IEEE International Conference on Software Maintenance and Evolution* (pp. 471–482). IEEE. <https://doi.org/10.1109/ICSME46990.2020.00052>
- [16] Zhou, Y., Wang, S., Liu, Y., Yao, Z., Lyu, M. R., & Cheung, S. C. (2024). Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, 33(1), 1–55. <https://doi.org/10.1145/3638241>
- [17] Damij, N., & Damij, T. (2024). An approach to optimizing Kanban board workflow and shortening the project management plan. *IEEE Transactions on Engineering Management*, 71, 13266–13273. <https://doi.org/10.1109/TEM.2021.3120984>
- [18] Merkel, D. (2014). Docker: Lightweight Linux containers for consistent development and deployment. *Linux Journal*, 2014(239), 2.
- [19] Abdelbaky, M., et al. (2024). Containerization in multi-cloud environment: Roles, strategies, challenges, and solutions for effective implementation. *ACM Transactions on Software Engineering and Methodology*. <https://doi.org/10.48550/arXiv.2403.12980>