IEEE-1016
System Design Description

1. Frontispiece

Date of Issue: May 2, 2025
Status: Draft v1.0
Authors:

- Stefan Mojesov - smojseov@constructor.university
- Bilal Waraich - bwaraich@constructor.university
- Felipe Ribadeneira - fribadenei@constructor.university
- Kamila Ziza - kziza@constructor.university
- Saim Malik - samalik@constructor.university
- Milica Tadic - mitadic@constructor.university
- Bisera Kjurchinska - bkjurchins@constructor.university
- Malakfatima Rzazade - mrzazade@constructor.university
- Kurabage Raveen Ranasinghe - kuranasing@constructor.university
- Madina Mazhenova - mmazhenova@constructor.university
- Anas Bakkoury - abakkoury@constructor.university

Change History:

- `v0.1`: Initial draft
- `v1.0`: Aligned with IEEE 1016-2009

2. Introduction

2.1 Purpose

To document the software design of a project and task management application, including architecture, rationale, data structures, and design viewpoints, for academic development and deployment on university servers/GitHub.

2.2 Scope

This project involves designing a web-based application for issue tracking, including:

- Project creation
- Task assignment

- User authentication
- Comments and file uploads
- Status tracking

2.3 Context

This software is designed as part of a university-level software engineering course. It will be developed collaboratively and deployed on a university-hosted server. The database will be hosted remotely.

2.4 Summary

The SDD follows IEEE 1016-2009 and provides views on the APIs used, deployment, functionality, traceability matrix, and rationale using diagrams and structured documentation.

3. API Overview

User Endpoints

- `POST /register` – Register new user

- `POST /login` – Authenticate user

- `GET /users/{id}` – Get user profile

Project Endpoints

- `POST /projects` – Create new project

- `GET /projects` – List all projects

- `GET /projects/{id}` – View specific project

Task Endpoints

- `POST /tasks` – Create new task

- `GET /tasks` – Get all tasks

- `PUT /tasks/{id}` – Update task

- `DELETE /tasks/{id}` – Delete task

Comment Endpoints

- `POST /tasks/{id}/comments` – Add comment

- `GET /tasks/{id}/comments` – View comments

4. Traceability Matrix

Requirement ID	Description	Design Element	API Endpoints
R1	**User Registration & Authentication**	**AuthService, DB Schema**	**POST /register, POST /login**
R2	**Task Management**	**TaskService, UI Component**	**POST /tasks, GET /tasks**
R3	**Project Overview & Assignment**	**ProjectService, Dashboard**	**POST /projects, GET /projects**
R4	**Commenting on Tasks**	**CommentService, Task UI**	**POST /tasks/{id}/comments**
R5	**Role-based Access Control**	**Middleware, AuthService**	**Auth Tokens, User Roles Table**

5. Viewpoints

Functional View: Task lifecycle, user flows

Deployment View:Hosted on university servers

Development View: Modular monorepo with Git branching

Data View: Prisma DB schema, object storage for attachments

6. Architecture Views

Context Diagram

```
User -->|Uses| Frontend
Frontend -->|API Calls| Backend
Backend -->|Reads/Writes| Database
Backend -->|Stores| ObjectStorage
Backend -->|Sends| NotificationService
```

Component Diagram

```
Frontend -->|REST API| Backend
Backend --> AuthService
Backend --> ProjectService
Backend --> TaskService
Backend --> NotificationService
Backend --> FileStorageService
AuthService --> Database
ProjectService --> Database
TaskService --> Database
NotificationService --> MessageQueue
FileStorageService --> ObjectStorage
```

Creating a Task: Sequence Diagram

```
participant U as User
participant FE as Frontend
participant BE as Backend
participant DB as Database

U->>FE: Fill task form
FE->>BE: POST /tasks
BE->>DB: INSERT INTO tasks
DB-->>BE: Success
```

BE-->>FE: 201 Created
FE-->>U: Show success message

7. Database Schema

For brevity purposes can be found in the attached ISO-IEC-IEEE-42010 compliant document under 4. Database Schema.

8. Stakeholders and Respective Concerns

Stakeholders	Concerns
Product Owner	Scalability
Team Members	Performance + Usability
Security Team	Security + Authentication

9. Architectural Decisions + Reasoning

This section contains the same information as the ISO-IEC-IEEE-42010 compliant document regarding rationale for why we choose the specific packages and frameworks.

2.4 Architectural Decisions and Reasoning

Frontend: React + TailwindCSS + Vite

Why choose React?

- **Industry standard** for modern frontend development.
- Huge ecosystem and community support.
- Excellent support for **component-based architecture**, reusable UI.
- Works well with libraries like Redux, React Router, etc.

Compared to alternatives:

- vs. Angular: React has a lighter learning curve and better flexibility.

- vs. Vue: React has broader job market and library ecosystem.

Why choose Tailwind CSS?

- Utility-first: No need to write custom CSS classes unless needed.
- Enforces **design consistency** across your UI.
- Extremely customizable and works great with dark/light mode, responsiveness, and themes.

Compared to alternatives:

- vs. Bootstrap: Tailwind gives more **design flexibility**, not locked into prebuilt styles.
- vs. plain CSS/SCSS: Much faster styling and better scalability.

Why choose Vite?

- Fast dev server and builds (thanks to native ES modules).
- Out-of-the-box support for **hot module replacement** and optimized React support.
- Modern tooling that's simpler and faster than Webpack.

Compared to alternatives:

- vs. CRA (Create React App): Vite is faster, lighter, and more customizable.

Backend: Node.js (NestJS) + NEXT.js + REST API

Why choose Node.js?

- Built on JavaScript → **shared language** between frontend and backend.
- Fast I/O and event-driven → great for handling many API requests efficiently.

Why choose NestJS?

- Scalable and **structured framework** built on top of Express.
- Inspired by Angular → **strong typing with TypeScript**, dependency injection, and modular architecture.
- Ideal for building enterprise-grade backends with **clean code and maintainability**.

Compared to alternatives:

- vs. Express.js alone: NestJS is opinionated and easier to scale.
- vs. Django/Rails: NestJS is better for JS/TS stacks, more flexible API design.

Why include Next.js?

- Can be used for **server-side rendering (SSR)** or API routes.
- You might use it **alongside or ahead of NestJS** if you want SEO-ready pages or server-rendered components.
- Helpful for **hybrid rendering**, dynamic content, and performance optimization.

Why choose REST API?

- Universally understood, easy to debug, and well-supported across platforms.
- Simple for CRUD operations and great if you're not doing real-time updates or complex graph traversals.

Compared to alternatives:

- vs. GraphQL: REST is easier to implement and test; less overhead for simple APIs.
- vs. gRPC: REST is more web-friendly and doesn't require extra tooling for most frontend use.

Database: Prisma

Why choose Prisma?

- **Type-safe ORM** with autocompletion and database schema syncing.
- Modern syntax and seamless integration with PostgreSQL, MySQL, SQLite, etc.
- Easy database migrations and clear data modeling with `schema.prisma`.

Compared to alternatives:

- vs. Sequelize: Prisma is more modern, less verbose, and fully TypeScript-native.
- vs. TypeORM: Prisma is faster, has more active development, and better DX (developer experience).

Authentication: JWT with Refresh Tokens

Why choose JWT + Refresh Tokens?

- **Stateless authentication**, great for REST APIs.
- Tokens can be stored client-side (e.g., in httpOnly cookies or localStorage).
- Refresh tokens improve **security and user experience** by allowing silent re-authentication.

Compared to alternatives:

- vs. Session-based auth: JWT is scalable, especially for APIs and microservices.
- vs. OAuth2 alone: JWT is simpler for internal apps or systems not relying on third-party login providers

Summary Table

Layer	Choice	Why it's better
Frontend	React + Tailwind + Vite	Fast dev, modern design, reusable components
Backend	Node.js + NestJS + (Optional Next.js SSR)	Type-safe, scalable, and well-structured
API	REST	Simpler and cleaner for CRUD apps
Database	Prisma	Type-safe ORM with fast migrations and better DX
Auth	JWT + Refresh Tokens	Secure, scalable, and stateless login management

10. Mapping of Sample User Stories to exact System Components and Flows

The User Stories detailed in the ISO-IEC-IEEE-42010 under 5. Sample User Stories are explained further with which system components are used + Views Affected.

User Story	Related Components	Views Affected
Create a task so that i can track my work	TaskService, TaskController, Task DB Schema, Frontend Task Form, TaskList.jsx	Task Creation View, Task List, Task Detail
Assign team members to projects as a project manager	ProjectService, UserService, AssignmentController, Project and User DB Schemas, TeamSelector.jsx	Project Detail View, Team Management View
Comment on tasks as a developer to communicate blockers	CommentService, TaskService, Comment DB Schema, CommentBox.jsx, TaskDetail.jsx	Task Detail View

11. Testing Strategy

Basic Unit Tests will be performed using Jest and Load Testing using JMeter.

12. Folder Structure

The relevant development folder structure can be found on the team github.