ITDS560 - Web and Mobile Application Development

Project Report: EIU Student Task Manager

Student: Samuel Afolabi

Email: samuelafolabi48@gmail.com

Course: ITDS560 - Web and Mobile Application Development

Institution: European International University

Submission Date: January 2026

1. Introduction

1.1 Background

As a student juggling multiple courses, I often found myself losing track of assignment deadlines and project milestones. While there are many task management apps available, most are either too complex for simple academic use or require paid subscriptions for basic features.

This motivated me to build EIU Student Task Manager - a straightforward web application specifically designed for students to organize their coursework, assignments, and projects.

1.2 Objectives

The main goals for this project were:

1. Create a user-friendly task management system tailored for students

2. Technology Choices

2.1 Why Next.js?

I chose Next.js 16 as the framework for several reasons:

- **Full-stack capability**: Next.js handles both frontend React components and backend API routes in one project, which simplified development considerably.
- **App Router**: The new App Router provides a cleaner file-based routing system.
- **Built-in optimizations**: Automatic code splitting, image optimization, and fast refresh during development.
- **Industry relevance**: Next.js is widely used in the industry, so learning it has practical career value.

2.2 Database: SQLite with Prisma

For the database, I used SQLite with Prisma ORM:

3. System Design

3.1 Architecture Overview

The application follows a typical full-stack architecture:

```
┌──────────────────┐     ┌──────────────────┐     ┌──────────────────┐
│    Frontend      │ ───▶│   API Routes     │ ───▶│    Database      │
│  (React/Next)    │ ◀───│   (Next.js)      │ ◀───│   (SQLite)       │
└──────────────────┘     └──────────────────┘     └──────────────────┘
```

3.2 Database Schema

I designed two main tables:

Users Table:

- id (unique identifier)
- email (unique)

4. Implementation Details

4.1 User Authentication Flow

1. **Registration**: User submits name, email, and password. The password is validated for strength (8+ characters, uppercase, lowercase, number, special character), then hashed with bcrypt before storing.
2. **Login**: Email and password are verified. If correct, a JWT token is created and stored in an HTTP-only cookie.
3. **Session Management**: Each request includes the cookie. The middleware checks the token and allows or denies access accordingly.
4. **Logout**: The token cookie is cleared from the browser.

4.2 Task Management

Tasks support full CRUD operations:

5. Testing

5.1 Testing Approach

I implemented tests using Jest and React Testing Library:

1. **Unit Tests**: Testing individual functions like password hashing and validation schemas
2. **Component Tests**: Testing React components render correctly and respond to user interactions

5.2 Test Results

```
Test Suites: 3 passed, 3 total
Tests:       25 passed, 25 total
```

Key tests include:

- Password hashing produces different hashes for same input (salt verification)

6. Challenges and Solutions

6.1 Zod Version Compatibility

Problem: The Zod validation library changed its API between versions. Error messages weren't displaying correctly.

Solution: After debugging, I found that Zod v4 uses `.issues` instead of `.errors` for accessing validation errors. Updated all API routes accordingly.

6.2 ESM Module Issues with Jest

Problem: The `jose` library uses ES modules, which Jest doesn't handle by default.

Solution: Added `transformIgnorePatterns` to Jest config to transform the jose module, and restructured tests to avoid importing modules with ESM dependencies where possible.

6.3 SQLite Enum Limitation

7. Security Considerations

7.1 Implemented Security Measures

1. **Password Hashing**: bcrypt with 12 salt rounds
2. **JWT in HTTP-only Cookies**: Prevents XSS token theft
3. **Input Validation**: Zod schemas validate all user input
4. **User Isolation**: Users can only access their own tasks
5. **Strong Password Policy**: Minimum 8 characters with complexity requirements

7.2 Future Security Improvements

- Rate limiting on login attempts
- CSRF token protection
- Email verification for new accounts
- Password reset functionality

8. Deployment

8.1 Docker Configuration

I created Docker files for containerized deployment:

- **Dockerfile**: Multi-stage build for optimized production image
- **docker-compose.yml**: Orchestration for easy deployment

8.2 Running in Production

```
docker-compose up --build
```

This builds the Next.js application in standalone mode and serves it on port 3000.

8.3 Environment Variables

The application uses environment variables for configuration:

9. Reflection

9.1 What I Learned

This project taught me:

1. **Full-stack development**: Building both frontend and backend in a cohesive application
2. **Authentication**: Implementing secure JWT-based auth from scratch
3. **Database design**: Modeling relationships and using an ORM effectively
4. **Testing**: Writing meaningful tests and understanding test patterns
5. **Problem-solving**: Debugging compatibility issues between libraries

9.2 What I Would Do Differently

If starting over, I would:

10. Conclusion

The EIU Student Task Manager successfully meets its objectives of providing a simple, secure, and user-friendly task management solution for students. The project demonstrates competency in modern web development technologies including Next.js, React, TypeScript, Prisma, and authentication best practices.

The application is fully functional with user registration, authentication, and complete task CRUD operations. While there's room for additional features, the current implementation provides a solid foundation that could be extended in the future.

Building this project has been valuable for applying theoretical knowledge to a practical application and gaining real-world development experience.

References

1. Next.js Documentation - https://nextjs.org/docs
2. Prisma Documentation - https://www.prisma.io/docs
3. Tailwind CSS Documentation - https://tailwindcss.com/docs
4. Jose JWT Library - https://github.com/panva/jose
5. Zod Validation - https://zod.dev
6. React Testing Library - https://testing-library.com/docs/react-testing-library/intro

Appendix A: Running the Application

```
# Install dependencies
npm install

# Set up database
npm run db:push

# Run development server
npm run dev

# Run tests
npm run test

# Build for production
npm run build
```

Appendix B: Project Repository Structure

```
task-manager/
```