University of Colorado,
Colorado Springs

# TaskTrackr Developer Documentation

Fall 2024

Prepared by :

**Monet Manning**

**Jimmy O'Hara**

**Jenna O'Hara**

**Avalee Cruz**

CS 3300 - 001

Professor:

Dr. Armin Moin

Teaching Assistants:

Marios Petrov, Michal Conner

## Overview

TaskTrackr is an intuitive and user-friendly task management application designed specifically for students to streamline the way they organize and manage their academic responsibilities. The platform enables users to efficiently track and manage assignments, projects, and other academic tasks by categorizing them based on class, task type, due dates, priorities, and progress status.

By offering visual progress indicators such as calendars, pie charts, and bar charts, TaskTrackr transforms task tracking into a seamless and engaging experience. Students can easily identify high-priority tasks, track their completion rates, and plan their schedules to stay on top of their academic workload.

One of TaskTrackr's standout features is its ability to promote productivity by helping students prioritize assignments and avoid the pitfalls of procrastination. Through customizable notifications and reminders, users are alerted to upcoming deadlines, enabling them to manage their time effectively and reduce stress.

With its clean design, customizable dashboard, and reliable performance, TaskTrackr is more than just a task manager—it’s a productivity companion that empowers students to achieve their academic goals with confidence and efficiency.


Overview ..... 2
Introduction ..... 3
Architecture Overview ..... 4
Requirements ..... 4
Functional Requirements ..... 4
Non-Functional Requirements ..... 5
User Requirements ..... 5
System Requirements ..... 5
Prerequisites and Installation ..... 5
Backend Setup (Spring Boot) ..... 5
Frontend Setup (Angular) ..... 6
System Design and Components ..... 6

Backend Components ..... 6
Frontend Components ..... 7
User Stories and Scenarios ..... 7
Testing ..... 7
Backend Testing ..... 7
Frontend Testing ..... 8
Risk Analysis ..... 8
Deployment ..... 9
Backend Deployment ..... 9
Frontend Deployment ..... 9
Development Workflow ..... 9
API Documentation ..... 9
User Authentication ..... 9
Class Management ..... 10
Task Management ..... 10
Notification Settings ..... 10
Miscellaneous ..... 11
Diagram of Functionality ..... 11
Initial Interface Sketch ..... 11

# Introduction

TaskTrackr provides college students with an accessible solution to manage multiple classes and assignments, prioritizing tasks and helping avoid stress. The application enables users to:

- Assign tasks to specific classes.
- Add custom parameters (task name, type, priority, due date, progress, etc.).
- Visualize task status and prioritize assignments.

# Architecture Overview

TaskTrackr is a **client-server** application:

- **Backend**: Spring Boot (Java) - handles data processing, storage, and task management.
- **Frontend**: Angular - provides an interactive, user-friendly interface for managing classes and assignments.

---

# Requirements

## Functional Requirements

1. **User Account Management**
    a. Users must be able to register, log in, and reset their password securely.
    b. Each user is uniquely identified by an email and password.
2. **Class Management**
    a. Users can add, edit, and delete classes with customizable names and colors.
    b. Each class will have associated assignments.
3. **Task Management**
    a. Users can add tasks with parameters like name, due date, task type, priority, progress status, and time allocation.
    b. Users can edit or delete tasks and update parameters.
4. **Notifications and Reminders**
    a. Users can set reminders for upcoming or high-priority tasks.
5. **Progress Visualization**
    a. Tasks and progress are displayed via visualizations (calendar, pie chart, bar chart).
6. **Dashboard Customization**
    a. Users can customize the dashboard to show preferred elements.

### Non-Functional Requirements

1. **Usability**
    a. Simple, intuitive interface with a consistent color scheme (Creamsicle theme).
2. **Performance**
    a. Efficient handling of tasks without performance degradation for large data sets.
3. **Scalability**
    a. Designed to support a large user base without compromising functionality.
4. **Reliability**
    a. Accurate and consistent task and class data storage.
5. **Security**
    a. Passwords and personal information are securely stored.

### User Requirements

1. Account creation and login
2. Class and task management with options for custom parameters
3. Visual organization with color-coded classes
4. Notifications for upcoming tasks
5. Customizable dashboard layout

### System Requirements

1. **Backend System**
    a. Java (JDK 11 or higher), Maven, Spring Boot
    b. Database for storing user, class, and task information
2. **Frontend System**
    a. Node.js, npm, Angular CLI, HTML, CSS (Tailwind)
3. **Development Environment**
    a. Any IDE supporting Java (IntelliJ IDEA, Eclipse) and Angular

## Prerequisites and Installation

### Backend Setup (Spring Boot)

1. **Install Maven**:
    a. Download and configure Maven, ensuring it's added to the system PATH.

2. **Run Spring Boot Application**:
    a. Navigate to the backend project directory and run:

```
mvn spring-boot:run
```

    b. Access the backend at http://localhost:8080.

### Frontend Setup (Angular)

1. **Install Node.js and npm**:
    a. Download the LTS version from Node.js and verify with:

```
node -v
npm -v
```

2. **Install Angular CLI**:
    a. Install Angular CLI globally:

```
npm install -g @angular/cli
```

3. **Run Angular Application**:
    a. Navigate to the frontend directory and run:

```
ng serve
```

    b. Access the frontend at http://localhost:4200.

---

## System Design and Components

### Backend Components

1. **User Module**:
    a. Manages account creation, login, and user data.
2. **Class Module**:
    a. Handles adding, editing, and deleting classes, color-coded for easy identification.
3. **Task Module**:
    a. Manages task creation, modification, and deletion with custom parameters.
4. **Notification Module**:

    a. Manages reminders and alerts for high-priority or upcoming tasks.

### Frontend Components

1. **Dashboard**:
    a. Centralized view for task and class summaries, visualizations, and customization options.
2. **Class Management**:
    a. Interface for creating and editing classes with color selection.
3. **Task Management**:
    a. Interface for adding tasks with editable parameters.
4. **Calendar/Progress Visualization**:
    a. Visual representation of tasks with filter options for classes and task priorities.

---

## User Stories and Scenarios

1. **User Story 1**: A new user wants to create an account and log in.
    a. **Requirements**: Registration and login must be secure and user-friendly.
2. **User Story 2**: A user wants to add classes with unique colors and names.
    a. **Requirements**: Users can add, edit, and delete classes.
3. **User Story 3**: A user wants to add tasks for a class with parameters like name, priority, and due date.
    a. **Requirements**: Task management must support customizable parameters.
4. **User Story 4**: A user needs to track task progress on a calendar or dashboard.
    a. **Requirements**: Progress visualization with filtering by task type, due date, and class.
5. **User Story 5**: A user wants notifications for due dates and priority tasks.
    a. **Requirements**: Configurable notification system for alerts.

## Testing

### Backend Testing

- **JUnit** is used for backend testing.
- Run tests:

```
mvn test
```

### Frontend Testing

- **Jasmine** and **Karma** are used for unit testing.
- Run tests:

```
ng test
```

---

## Risk Analysis

| Risk | Strategy | Impact |
|---|---|---|
| **Changes to requirements requiring major design rework** | Maintain clean, modular, and well-documented code to support easy changes and updates. | Delays in project timelines and increased development necessary. |
| **Developer has illness or unavailability at critical times** | Ensure team members are cross trained in different components of the project. | Prevents project bottlenecks and ensures continuity in development. |
| **Underestimated development time** | Use agile methodologies to track progress and adjust scope as needed. | Risk of missing deadlines, leading to incomplete or rushed functionality |
| **Version control conflicts during team collaboration** | Enforce coding standards, code reviews, and resolve conflicts immediately with clear communication. | Delays in code integration and potential overwriting of critical work. |
| **Inadequate testing coverage** | Mandate unit, integration, and user acceptance testing with detailed test plans and automated tests. | Increase in bugs and production issues, reducing application reliability. |
| **Dependency version incompatibility or updates** | Regularly update dependencies and monitor compatibility dependency management tools like Maven and npm. | Application instability and functionality issues. |

# Deployment

## Backend Deployment

- Package and deploy the Spring Boot application.
- Configure environment variables for the database and API endpoint.

## Frontend Deployment

- Build the Angular application for production:

```
ng build --prod
```

- Serve the static files using a web server (e.g., Nginx).

---

# Development Workflow

1. **Clone the Repository**:
    a. Clone the codebase and install backend and frontend dependencies.
2. **Feature Development**:
    a. Create branches for new features, implement, and test.
3. **Code Review**:
    a. Submit a pull request for code review.
4. **Testing**:
    a. Run automated tests before merging.
5. **Deployment**:
    a. Follow deployment steps to push changes to the live environment.

---

# API Documentation

## User Authentication

1. **POST /register** - Registers a new user.
2. **POST /login** - Authenticates a user.

3. **POST /logout** - Ends the user session.

## Class Management

1. **GET /classes** - Retrieves all classes for a user.
2. **POST /classes** - Creates a new class.
3. **DELETE /classes/{id}** - Deletes a specified class.

## Task Management

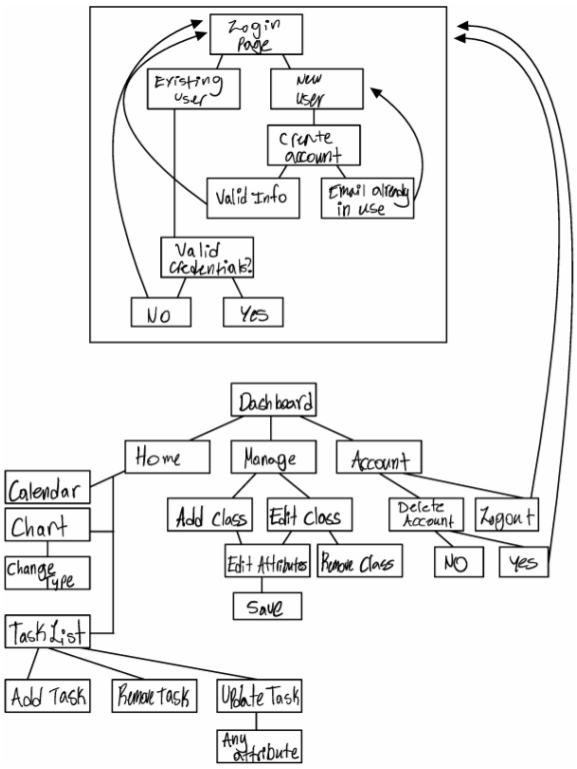
1. **GET /tasks** - Retrieves all tasks for a user.
2. **POST /tasks** - Adds a new task.
3. **PUT /tasks/{id}** - Updates a task's details.
4. **DELETE /tasks/{id}** - Deletes a specified task.

## Notification Settings

1. **GET /notifications** - Retrieves notification preferences.
2. **POST /notifications** - Updates notification preferences.

# Miscellaneous

## Diagram of Functionality




## Initial Interface Sketch