DTS Developer Challenge Project Document

Project Trello Board: [DTS-Developer-Challenge | Trello]
Project Github Repository: [challenges-bear-2025/DTS-Developer-Challenge]

Table of Contents

Requirements Analysis

Problem Statement

HMCTS requires a new system to be developed so caseworkers can keep track of their tasks. Develop a new system that allows caseworkers to efficiently manage their tasks.

User Stories

Below are the key user stories aligned to the system's intended functionality, from the perspective of the caseworker:

1. **(Create)**
   As a caseworker, I want to create a new task with a title, optional description, status, and due date/time, so that I can effectively manage and organize my workload.

2. **(Retrieve - List View)**
   As a caseworker, I want to view a list of all tasks, so that I can quickly see an overview of my current workload and outstanding actions.

3. **(Retrieve - Specific Task)**
   As a caseworker, I want to retrieve a specific task by its ID, so that I can view and manage detailed information about that task.

4. **(Update)**
   As a caseworker, I want to update information related to a task (such as status or description), so that task progress and related details can be tracked and maintained.

5. **(Delete)**
   As a caseworker, I want to delete a task when it is no longer needed, so that my task list remains accurate, relevant, and manageable.

Functional Requirements

Below is a table of requirements derived from the user stories, these requirements are all of the same priority for this project so that column has not been shown:

ID	Requirement
FR1	The system must allow users to create new tasks, with a title, optional description, status and the due date/time
FR2	The system must allow users to view a list of all tasks.
FR3	The system must allow users to retrieve a specific task by its unique ID.
FR4	The system must allow users to update the status of a task.
FR5	The system must allow users to delete a task.

Non-Functional Requirements

Below is a table of the non-functional requirements for the main quality attributes that are needed to ensure this project is completed successfully:

ID	Requirement	Quality Attribute Areas
NFR1	The system must be responsive and load the task list within 2 seconds under normal conditions.	Performance, Usability
NFR2	The system must ensure data consistency between frontend and the backend.	Reliability, and Data Integrity
NFR3	The system must comply with WCAG 2.1 AA standards.	Usability
NFR4	The system must be compatible with the latest versions of major browsers (Chrome,	Compatibility

	Firefox, Edge).	
NFR5	The backend must expose secure APIs with proper authentication	Security
NFR6	The system should be easy to maintain and follow clean coding standards.	Maintainability
NFR7	The system will be designed in accordance with the GOV.UK Design System.	Usability, and Accessibility

System Design & Architecture

Technology Stack

For this project, the following technologies have been selected to develop the task management system for HMCTS, ensuring a robust and maintainable application architecture:

Frontend: React.js Framework

Backend: Java Spring Boot Framework (Maven)

Database: PostgreSQL

Technology Justification

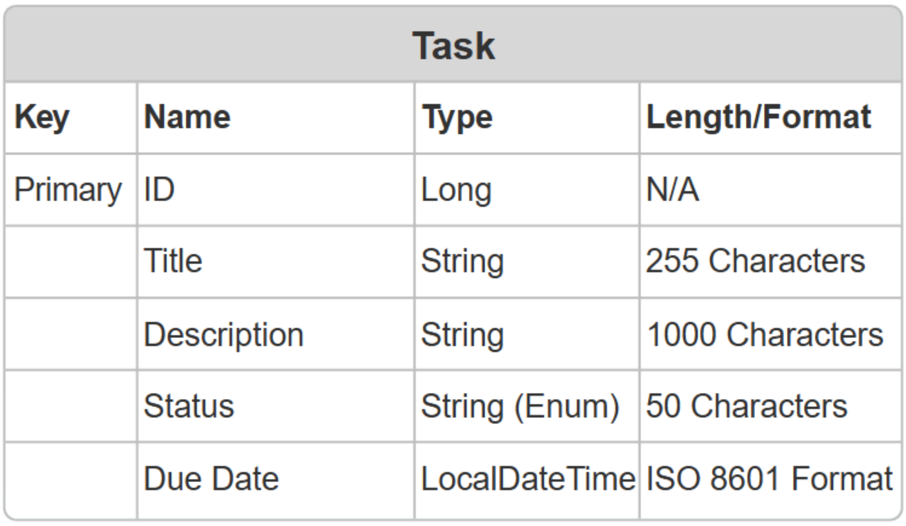
React.js: Enables creation of highly responsive, component-based user interfaces that are ideal for modern web applications. It supports fast development and easy maintenance through reusable components.

Spring Boot: Provides a lightweight, production-ready backend with built-in features such as dependency injection, security, and simplified REST API development. Maven will be used for the project build and dependency management.

PostgreSQL: A robust, open-source relational database management system known for reliability, data integrity, and a strong support for complex queries. It integrates well alongside Spring Data JPA.

Entity-Relationship Diagram

Given the relatively simple scope of this project, the entity-relationship diagram primarily focuses on modelling the **Task** entity and its structure within the database schema. The **Task** entity will be mapped with attributes such as ID, Title, Description, Status and Due Date. The primary key for this entity will be the ID, and this will be used to guarantee uniqueness of each entry.



Task

Key	Name	Type	Length/Format
Primary	ID	Long	N/A
	Title	String	255 Characters
	Description	String	1000 Characters
	Status	String (Enum)	50 Characters
	Due Date	LocalDateTime	ISO 8601 Format

Entity-Relationship Diagram

UML Class Diagram

Below is the UML class diagram representing the core structure of the system’s backend architecture.

The diagram models the primary components:

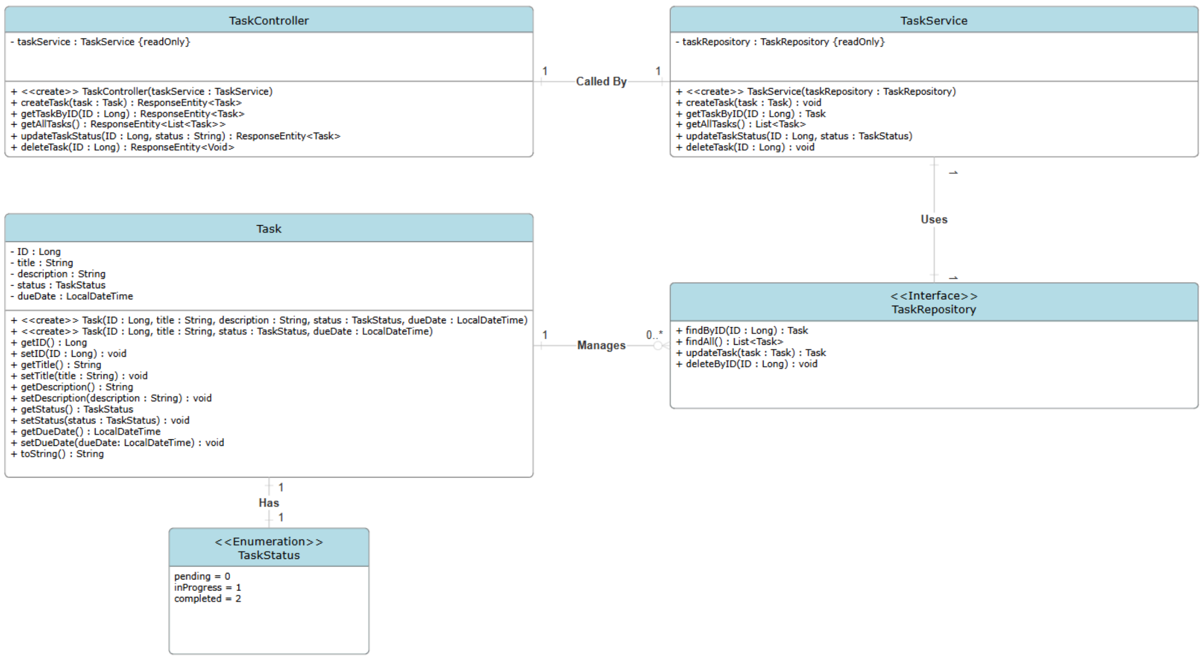
Task Entity Model Class: Represents the task entities data structure and its attributes

Task Repository Interface: Defines the database access methods for task persistence

Task Service Class: Encapsulates business logic for creating, retrieving, updating and deleting tasks.

Task Controller Class: Exposes RESTful endpoints to allow the frontend to interact with the backend services

This design promotes a clean separation of concerns, improving maintainability, and aligns with the best practices for Spring Boot application development.


TaskController
- taskService : TaskService {readOnly}
+ <<create>> TaskController(taskService : TaskService)
+ createTask(task : Task) : ResponseEntity<Task>
+ getTaskByID(ID : Long) : ResponseEntity<Task>
+ getAllTasks() : ResponseEntity<List<Task>>
+ updateTaskStatus(ID : Long, status : String) : ResponseEntity<Task>
+ deleteTask(ID : Long) : ResponseEntity<Void>
1
Called By
1
TaskService
- taskRepository : TaskRepository {readOnly}
+ <<create>> TaskService(taskRepository : TaskRepository)
+ createTask(task : Task) : void
+ getTaskByID(ID : Long) : Task
+ getAllTasks() : List<Task>
+ updateTaskStatus(ID : Long, status : TaskStatus)
+ deleteTask(ID : Long) : void
Uses
Task
- ID : Long
- title : String
- description : String
- status : TaskStatus
- dueDate : LocalDateTime
+ <<create>> Task(ID : Long, title : String, description : String, status : TaskStatus, dueDate : LocalDateTime)
+ <<create>> Task(ID : Long, title : String, status : TaskStatus, dueDate : LocalDateTime)
+ getID() : Long
+ setID(ID : Long) : void
+ getTitle() : String
+ setTitle(title : String) : void
+ getDescription() : String
+ setDescription(description : String) : void
+ getStatus() : TaskStatus
+ setStatus(status : TaskStatus) : void
+ getDueDate() : LocalDateTime
+ setDueDate(dueDate: LocalDateTime) : void
+ toString() : String
1
Manages
0..*
<<Interface>>
TaskRepository
+ findByID(ID : Long) : Task
+ findAll() : List<Task>
+ updateTask(task : Task) : Task
+ deleteByID(ID : Long) : void
1
Has
1
<<Enumeration>>
TaskStatus
pending = 0
inProgress = 1
completed = 2

Class Diagram

Development Methodology

This project will follow a lightweight Agile-inspired development process tailored over about a week's time frame. Agile principles such as iterative development, continuous feedback, and working software delivery are adapted to suit a condensed schedule.

The workflow is structured into short, focused iterations to ensure rapid progress while maintaining quality and flexibility. This project will be developed using test-driven development.

Note: This is a rough guideline plan to give the project some structure, and if there is successful completion of tasks earlier than expected, then I will move freely onto the next task.

Project Workflow Structure

Phase	**Duration**	**Key Tasks**
Planning & Setup	Day 1	● Finalise requirements ● Define technical approach ● Set up project repository and project structure
Sprint 1	Days 2-4	● Develop backend API ● Implement database integration ● Test core API endpoints using Postman
Sprint 2	Days 5-7	● Build the frontend with React.js ● Create task CRUD interfaces ● Connect frontend to backend APIs
Sprint 3	Days 8-9	● Refine functionality (validation, error handling, UX improvements) ● Conduct full

		system integration testing ● Fix defects and polish UI
Review & Delivery	Day 10	● Final testing ● Documentation preparation ● Project submission

Agile Practices

Below are the following agile practices I will apply to the project to attempt to simulate the development of a real project using the Agile methodology:

Daily Standups (Self-check-ins): At the start of each day I will conduct a personal review of completed tasks, current objectives, and any blockers. Any notes will be taken within my handwritten journal.

Sprints: The project has been broken into small roughly 2-3 day sprints, each delivering working increments of the system.

Continuous Integration: Code changes are committed regularly to Github to ensure the system remains in a deployable state.

Frequent Testing: Unit testing and manual functional testing will be performed continuously throughout development

Documentation: Key technical and user-facing documentation will be developed at the end of the development process to support the codebase.

Tools Used

Tool	**Purpose**
Git & Github	Source control and version management
Postman	Allows for testing of backend API
Trello	Will be used to track sprints and organise tasks to complete