**Dulanji Cooray -Full Stack Engineer Take Home Assessment Answers**

**System Architecture Overview**

Your application will consist of three main components running in separate Docker containers:

|  |  |  |
| --- | --- | --- |
| **Component** | **Technology Choice** | **Responsibilities** |
| **Database** | PostgreSQL | Persist task data in a relational table |
| **Backend API** | Java with Spring Boot | Provide REST API endpoints for task operations |
| **Frontend UI** | HTML, CSS, Vanilla JavaScript | Single-Page Application for user interactions |

The communication flow is as follows:

1. **User** interacts with the Frontend UI (clicks, forms)
2. **Frontend** sends HTTP requests to the Backend API (fetch, XMLHttpRequest)
3. **Backend** processes requests and interacts with the Database (JDBC, JPA)
4. **Database** stores and returns persistent task data

**Database Design**

You'll need a single task table. Here is the recommended schema:

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** | **Description** |
| **id** | BIGSERIAL or SERIAL | PRIMARY KEY | Unique auto-incrementing task identifier |
| **title** | VARCHAR(255) | NOT NULL | The main title/headline of the task |
| **description** | TEXT | - | Detailed description of the task (optional) |
| **completed** | BOOLEAN | NOT NULL DEFAULT FALSE | Completion status (TRUE if done) |
| **created\_at** | TIMESTAMP | NOT NULL DEFAULT NOW() | Auto-recorded creation timestamp |

This design supports all user requirements: storing title/description, tracking completion status, and using created\_at to fetch the most recent tasks.

**Backend API Design (Java Spring Boot)**

The backend will be a RESTful API with the following endpoints:

|  |  |  |  |
| --- | --- | --- | --- |
| **HTTP Method** | **Endpoint** | **Request Body** | **Description** |
| **GET** | /api/tasks | - | Fetches the 5 most recent **incomplete** tasks |
| **POST** | /api/tasks | {"title": "string", "description": "string"} | Creates a new task |
| **PUT** | /api/tasks/{id}/complete | - | Marks a specific task as completed |

**Applying SOLID Principles in Your Backend Code**

Following SOLID principles is a key evaluation criterion. Here's how to apply them in your Java code:

* **Single Responsibility Principle (SRP)**: Each class has one reason to change.
  + TaskController: Handles HTTP requests and responses.
  + TaskService: Contains the core business logic (e.g., ensuring only 5 tasks are shown).
  + TaskRepository: Manages all data access and interaction with the database.
* **Dependency Inversion Principle (DIP)**: Depend on abstractions, not concrete implementations.
  + **Create a**TaskService**interface**: Define methods like List<Task> getRecentIncompleteTasks(), Task createTask(Task task), etc.
  + **Create a**TaskServiceImpl**class**: The concrete implementation of the interface.
  + **Inject the interface**: Your TaskController should depend on the TaskService interface, not the TaskServiceImpl class. This makes your code more flexible and easier to test.