MITA

Mini Issue Tracker Application

Implementation Workshop

Frontend: React

Choose ONE backend: Java | .NET | PHP

Database: MySQL

1. Workshop Overview

In this workshop, you will build **MITA** — a Mini Issue Tracker Application.

The project consists of **multiple repositories**:

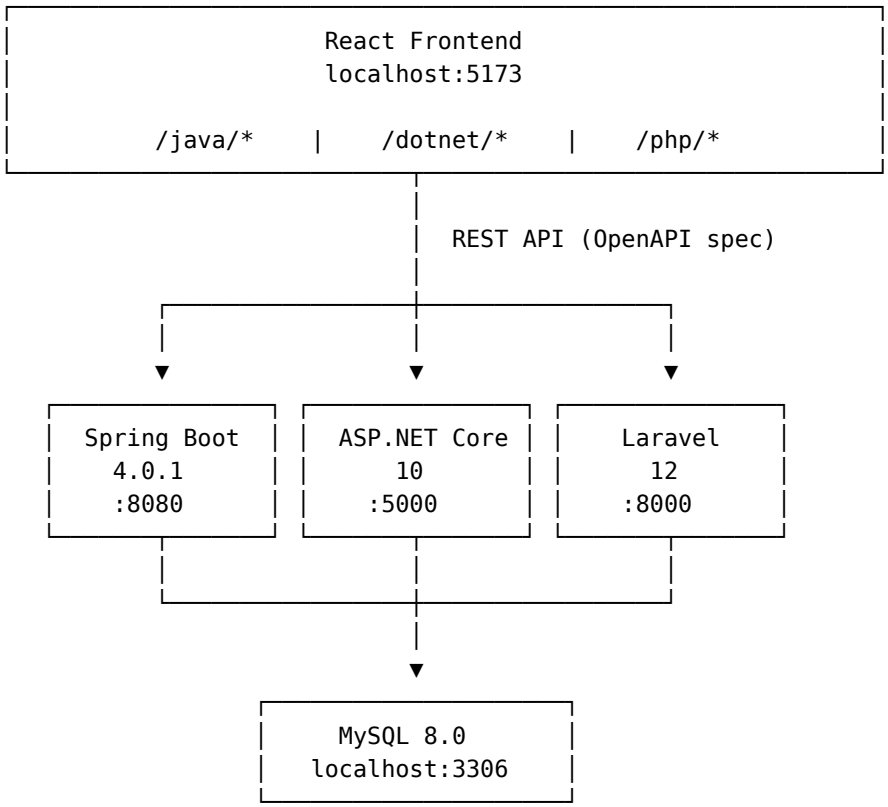
- One **React frontend** (skeleton with login stub)
- Three **backend skeletons** (Java, .NET, PHP)
- One shared **API contract** (OpenAPI specification)

Your task:

- Clone the **frontend** repository and implement all UI features
- Choose **ONE backend** repository and implement all API endpoints

Both frontend and backend contain only a basic login stub. You will implement the full functionality according to the requirements below.

2. Architecture



3. Repositories

API Contract	https://github.com/vmatviichuk-epam/mita-api-contract
Frontend	https://github.com/vmatviichuk-epam/mita-frontend
Java Backend	https://github.com/vmatviichuk-epam/mita-backend-java
.NET Backend	https://github.com/vmatviichuk-epam/mita-backend-dotnet
PHP Backend	https://github.com/vmatviichuk-epam/mita-backend-php

Getting Started

```
1. Clone the frontend repository:
   git clone --recursive https://github.com/vmatviichuk-epam/mita-frontend.git

2. Clone ONE backend repository of your choice:
   2.1 Java:  git clone --recursive https://github.com/vmatviichuk-epam/mita-backend-
java.git
   2.2 .NET:  git clone --recursive https://github.com/vmatviichuk-epam/mita-backend-
dotnet.git
   2.3 PHP:   git clone --recursive https://github.com/vmatviichuk-epam/mita-backend-
php.git

3. Ensure MySQL is running

4. Start the frontend:  npm install && npm run dev

5. Start your backend (see backend repo README)

6. Open browser: http://localhost:5173/java/login   (for Java)
                 http://localhost:5173/dotnet/login (for .NET)
                 http://localhost:5173/php/login    (for PHP)
```

Note: All repos include the API contract as a git submodule in `api-contract/` folder.

Full API specification available in `openapi.yaml`

4. Requirements

4.1. Goal

Provide a minimal issue-tracking application where authenticated users manage their own issues. The application must support user registration, login, issue CRUD operations, controlled status transitions, priority changes, filtering, and optional activity tracking. All data must be stored in a MySQL database.

4.2. Description

Implement a strictly scoped issue tracker that enables each authenticated user to create, update, view, and delete their own issues. The system must enforce user isolation, require authentication for all issue operations, and follow predefined workflows for status and priority changes.

4.3. 1. User Authentication & Management

- Users must be able to register with a unique username and a password.
- Users must be able to log in using their registered credentials.
- After login, users must obtain a session or token that is required for all subsequent operations.
- Users must be able to log out, invalidating their session or token.
- Users must only be able to access their own issues. Access to issues owned by other users must be denied.

4.4. 2. Issue Creation & Management

- Users must be able to create issues with the following required fields: title, description, status, priority, creation timestamp.
- The default value for status must be **Open**.
- The default value for priority must be **Medium**.
- Users must be able to update all issue fields except the creation timestamp.
- Users must be able to delete issues they own.
- Title and description must not be empty.

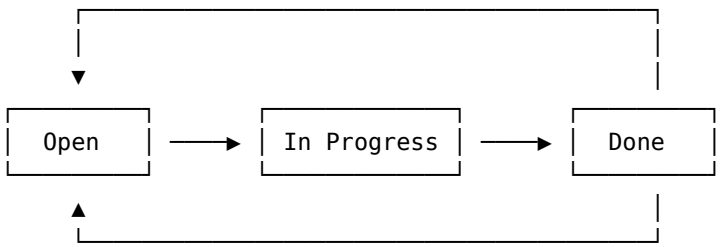
4.5. 3. Issue Workflow

Allowed status values: **Open**, **In Progress**, **Done**

Allowed priority values: **Low**, **Medium**, **High**

Allowed status transitions:

- Open → In Progress
- In Progress → Done
- Done → Open



Users must not set the status to a value outside the allowed list.

Users must not set the priority to a value outside the allowed list.

4.6. 4. Issue Listing & Filtering

- Users must be able to retrieve a list of their own issues.
- Users must be able to filter issues by status.
- Users must be able to filter issues by priority.
- If multiple filters are provided, the system must apply all provided filters simultaneously.
- The default ordering must be by creation timestamp ascending.

4.7. 5. Activity Tracking (Optional)

If activity tracking is implemented:

- Every status change must create an activity entry.
- Every priority change must create an activity entry.
- An activity entry must contain: issue reference, timestamp, change type, old value, new value.
- Activity entries must be immutable.
- Activity entries must be retrievable as part of the issue details.

4.8. 6. Storage & Persistence

- The application must store all data in a **MySQL database**.
- The schema must include at minimum: `users`, `issues`, `activity_log` (if activity tracking is implemented).
- Passwords must be stored in hashed form.
- Foreign key constraints must enforce user-to-issue ownership.

4.9. 7. UI

The UI must contain at minimum:

- A login page
- A registration page
- A page listing user issues with filtering controls
- A page for creating and editing issues

All UI actions must map directly to the defined backend operations.

4.10. Success Criteria

- The application must implement exactly the features described above.
- All core operations must require authentication.
- User data and issue data must remain isolated at all times.