**Planifacation Project**

1. 🔍 Requirement Gathering (فهم المتطلبات)

### 🎯 Project Objective:

Build a web application (Backend with **Spring Boot** and Frontend with **Angular**) that allows users to:

* Add movies to the system
* Rate movies with a star rating and optional comment
* View a list of movies along with their ratings
* Calculate and display the average rating for each movie

### 🧱 Entities:

#### ✅ User

* id (Long) – Unique identifier
* username (String)
* email (String)
* password (String) – securely hashed
* roles (Set) – such as ADMIN or USER

#### ✅ Movie

* id (Long)
* title (String) – movie title
* description (String) – movie description
* releaseDate (LocalDate)
* posterImage (byte[] or file path) – movie poster image
* averageRating (Double) – average rating (calculated automatically)

#### ✅ Rating

* id (Long)
* stars (int) – from 1 to 5
* comment (String) – optional review comment
* user (User) – user who rated
* movie (Movie) – movie being rated
* createdAt (Timestamp)

### 📌 Core Features:

* 🔐 **Authentication & Authorization**:
  + User registration
  + User login with JWT authentication
  + Secure access control to protected endpoints
* 🎞️ **Movies**:
  + Add, update, delete movies (Admin only)
  + Upload movie poster images
  + Display movies with their posters and ratings
* ⭐ **Ratings**:
  + Allow registered users to rate movies
  + Users can update or delete their own ratings
  + Automatically calculate the average rating per movie
* 🔍 **Public Access**:
  + Provide read-only access to movie listings and ratings for unregistered visitors

### 📁 File Upload:

* When creating a new movie, the poster image is uploaded as a MultipartFile and stored either:
  + In an uploads/ directory inside the project
  + Or directly in the database as a byte[]

2. 🧱 System Design (تصميم النظام)

### 🗄️ Database Design:

* **User Table:**
  + id (Primary Key)
  + username (Unique)
  + email (Unique)
  + password (hashed)
  + roles (e.g., ADMIN, USER)
* **Movie Table:**
  + id (Primary Key)
  + title
  + description
  + release\_date
  + poster\_image (file path or byte array)
  + average\_rating (calculated or updated on rating changes)
* **Rating Table:**
  + id (Primary Key)
  + stars (1-5)
  + comment (nullable)
  + user\_id (Foreign Key to User)
  + movie\_id (Foreign Key to Movie)
  + created\_at (timestamp)

### 🔄 Relationships:

* One **User** can have many **Ratings** (One-to-Many)
* One **Movie** can have many **Ratings** (One-to-Many)
* Each **Rating** belongs to one **User** and one **Movie**

### 🏗️ Backend Architecture:

* Use **Spring Boot** to build REST APIs
* Secure endpoints using **JWT authentication**
* Create services to manage users, movies, and ratings
* Handle file uploads for movie posters — store locally or in the database
* Calculate average movie rating dynamically or update it whenever ratings change

### ⚛️ Frontend Architecture:

* Use **Angular** for building the UI
* Components for login, registration, movie list, movie details, and rating form
* Use HTTP client to communicate with the backend APIs
* Display movie posters, ratings, and average ratings

3. 📦 Work Breakdown Structure - WBS (تقسيم المشروع)  
1. Project Setup

* Initialize the Spring Boot backend project
* Initialize the Angular frontend project
* Set up the PostgreSQL database

### 2. User Management

* Design the User entity and create its table in the database
* Implement APIs for user registration
* Implement login API using JWT authentication
* Set up role-based access control (e.g., ADMIN, USER)

### 3. Movie Management

* Design the Movie entity and create its table in the database
* Implement APIs to add, update, and delete movies (usually admin only)
* Implement functionality to upload movie poster images
* Implement APIs to display movies with details and average ratings

### 4. Ratings Management

* Design the Rating entity and create its table in the database
* Implement APIs for users to rate movies
* Implement APIs for users to update or delete their own ratings
* Implement automatic calculation of the average rating per movie

### 5. Frontend Development

* Build components for user registration and login
* Build components to display the list of movies with posters and ratings
* Build components for movie details and rating submission
* Connect the frontend with the backend APIs

### 6. Testing

* Write unit tests for the backend
* Perform integration tests for the APIs
* Test the frontend and fix any issues

### 7. Deployment

* Prepare the project for deployment (backend and frontend)
* Configure PostgreSQL on the production server
* Deploy the application and verify it’s working properly

4. ⏱ Estimation (تقدير الوقت)

🟢 1. Project Setup (1.5 days)

* Initialize Spring Boot and Angular projects
* Set up PostgreSQL database
* Configure JWT and CORS  
  🕒 **Estimated Time: 1.5 days**

### 🔐 2. User Management (2 days)

* Create User entity and table
* Implement user registration and login
* Hash passwords securely
* JWT Authentication with role-based access (ADMIN/USER)  
  🕒 **Estimated Time: 2 days**

### 🎬 3. Movie Management (2.5 days)

* Create Movie entity and table
* Add/update/delete movies (Admin only)
* Upload movie poster image
* List all movies with details  
  🕒 **Estimated Time: 2.5 days**

### ⭐ 4. Rating Management (2 days)

* Create Rating entity and table
* Allow users to rate, edit, and delete their ratings
* Calculate and update average rating per movie  
  🕒 **Estimated Time: 2 days**

### 💻 5. Frontend Development (3 days)

* Build UI for login and registration
* Movie listing and details view
* Rating form and movie poster upload
* Integrate with backend via HTTP  
  🕒 **Estimated Time: 3 days**

### 🧪 6. Testing (1.5 days)

* Unit tests for backend services
* Functional testing of Angular components  
  🕒 **Estimated Time: 1.5 days**

### 🚀 7. Deployment (0.5 day)

* Deploy Spring Boot backend
* Deploy Angular frontend
* Set up PostgreSQL in production environment  
  🕒 **Estimated Time: 0.5 day**

### 🧮 ****Total Estimated Time:****

**🕓 Approximately 13 working days**

5. 🗓 Scheduling & Timeline (إعداد الجدول الزمني)

This timeline outlines how to organize your tasks over 13 working days for the **Movie Ratings App** project:

| Day | Task | Details |
| --- | --- | --- |
| ✅ **Day 1** | Project Setup | - Initialize Spring Boot & Angular projects- Connect to PostgreSQL- Configure JWT & CORS |
| 🔐 **Day 2** | User Registration & Login | - Create User entity- Implement user signup & login with JWT |
| 🔐 **Day 3** | Access Control | - Secure endpoints with roles- Set up ADMIN and USER permissions |
| 🎬 **Day 4** | Movie Management – Part 1 | - Create Movie entity- Add & update movie info |
| 🎬 **Day 5** | Movie Management – Part 2 | - Implement file upload for movie posters- Display movie list with images |
| ⭐ **Day 6** | Rating System – Part 1 | - Create Rating entity- Allow users to rate movies |
| ⭐ **Day 7** | Rating System – Part 2 | - Enable edit/delete rating- Calculate & display average movie rating |
| 💻 **Day 8** | Frontend – Auth UI | - Build login and registration components in Angular |
| 💻 **Day 9** | Frontend – Movie Display | - Show list of movies with posters and info |
| 💻 **Day 10** | Frontend – Rating UI | - Create form to add/edit ratings- Show stars and comments |
| 🧪 **Day 11** | Backend Testing | - Write unit and integration tests for backend |
| 🧪 **Day 12** | Frontend Testing | - Test UI functionality and API integration |
| 🚀 **Day 13** | Deployment & Final Touches | - Deploy backend & frontend- Prepare production DB and hosting |

🛠️ **Tip:** You can adjust the timeline based on your progress. Some tasks may take less time, giving you room for testing or deployment.

Would you like a Trello-style board or weekly calendar view based on this timeline?

6. 🧪 Testing Plan (تخطيط التستات)

### ✅ 1. Unit Testing

**🔧 Backend (Spring Boot):**

* UserServiceTest: Tests for user registration, login, and password hashing.
* MovieServiceTest: Tests for adding, updating, and deleting movies.
* RatingServiceTest: Tests for creating, updating, and deleting ratings.
* AverageRatingCalculatorTest: Verifies the correctness of average rating computation.

🛠 Tools:

* **JUnit**
* **Mockito**

### ✅ 2. Integration Testing

* Test full flows like user registration and login.
* Test rating submission for a movie.
* Ensure movie details include all associated ratings.

🛠 Tools:

* **Spring Boot Test**
* **TestRestTemplate** or **MockMvc**

### ✅ 3. Frontend Unit Testing (Angular)

* AuthComponent: Test login and registration forms.
* MovieListComponent: Test display of movies.
* RatingComponent: Test rating form functionality and validation.

🛠 Tools:

* **Karma**
* **Jasmine**

### ✅ 4. End-to-End (E2E) Testing

* Simulate real user flows:
  + User registration
  + Login
  + View movie list
  + Submit and edit a rating

🛠 Tools:

* **Cypress** or **Protractor**

### ✅ 5. Manual Testing

* Manual browser-based tests for:
  + Authentication flows
  + CRUD operations on movies
  + Submitting and editing ratings
  + Image upload & display
  + General user interface and responsiveness

### 🔁 Best Practices:

* Run unit tests during each development step.
* Automate integration and E2E tests for CI/CD.
* Cover edge cases (e.g., invalid input, unauthorized access).

7. 🚀 Deployment & Monitoring Plan (خطة الإطلاق والمراقبة)

### ✅ 1. Deployment Plan

#### 🛠 Backend (Spring Boot):

* Build the project and generate a .jar file using **Maven** or **Gradle**.
* Deployment options:
  + Use **Docker** for portability and easy deployment.
  + Deploy on cloud servers like **AWS EC2**, **DigitalOcean**, or similar.
  + Run as a standalone app with embedded **Tomcat** server.

#### 🌐 Frontend (Angular):

* Build a production-ready version with: ng build --prod.
* Host on:
  + Web servers like **Nginx** or **Apache**.
  + Hosting platforms such as **Netlify**, **Vercel**, or **Firebase Hosting**.

#### 🗄 Database (PostgreSQL):

* Use local DB for development.
* For production, use managed services like **Render**, **ElephantSQL**, or **Supabase**.

#### 🔐 Security:

* Enable HTTPS using **SSL certificates** (e.g., **Let's Encrypt**).
* Configure CORS, JWT authentication, and rate limiting.
* Manage sensitive data (passwords, tokens) with environment variables.

### ✅ 2. CI/CD (Continuous Integration and Deployment)

Use automated pipelines to build, test, and deploy the app whenever code is pushed.

**Tools:**

* **GitHub Actions**, **GitLab CI**, or **Jenkins**.

**Pipeline steps:**

1. Code push triggers pipeline.
2. Run unit and integration tests.
3. Build Angular frontend and Spring Boot backend.
4. Deploy to staging or production environment.

### ✅ 3. Monitoring & Logging

#### 📊 Monitoring:

* Uptime monitoring tools like **UptimeRobot**, **Pingdom**.
* Performance monitoring with **Prometheus + Grafana**, or commercial services like **New Relic**, **Datadog**.

#### 📋 Logging:

* Use **Spring Boot Actuator** for health checks and metrics.
* Log management with **Logback** or **Log4j**.
* Forward logs to files or external services like **ELK Stack**, **Loggly**, or **Graylog**.

### ✅ 4. Backup Strategy

* Schedule regular backups (daily or weekly) of the database.
* Store backups securely in cloud storage (e.g., AWS S3).
* Periodically test restoration procedures to ensure backups work.

### ✅ 5. Rollback Strategy

* Keep previous versions/tags of the app for rollback.
* Use Docker image tags or Git tags for version control.
* Maintain a staging environment to test before deploying to production.