# 🛠️ Task: Build a Three-Web-App System for File and Data Management

Imagine you are building a **"Digital Property Management System"** for a real estate agency.

- **App 1** allows realtors to upload **property-related documents** (e.g., floor plans, contracts, images).
- **App 2** allows realtors to upload **JSON files** or fill out a form containing **property metadata** (e.g., address, price, property type).
- **App 3** is the agency's **central portal** that can manage both files and data.

You are assigned to build **three interconnected web applications** using **Angular** (frontend) and **Spring Boot** (backend), with **MySQL** as the database.

The goal is to:

1. Upload and store files.
2. Upload and store structured data (JSON to DB).
3. Create a **central portal** that communicates with both systems through **REST APIs** to manage all uploads and view the data centrally.

## Contents

## 📦 App 1: "File Storage Service"

- **Tech Stack:** Angular + Spring Boot
- **Functionality:**
    - User uploads a **file** (e.g., PDF, DOCX, JPG, etc.).
    - Backend stores the file **physically** inside the server, specifically inside the **/resources/uploads** folder.
- **Endpoints (Spring Boot):**
    - POST /api/files/upload: Accepts a multipart/form-data request to upload a file.
    - GET /api/files: Lists all uploaded files (filenames only).
- **Angular Frontend:**
    - Simple form with file input + upload button.
    - Page that lists uploaded files.
- **Backend concepts:**
    - Use @RestController for APIs.
    - Save files in resources/uploads/.
    - Set spring.servlet.multipart.max-file-size and spring.servlet.multipart.max-request-size in application.properties.
- **Frontend concepts:**
    - Use Material UI.
    - Use Angular Reactive Forms for file uploads.
    - Display a table/grid for uploaded files.

## App 2: "JSON Data Ingestion Service"

- **Tech Stack:** Angular + Spring Boot + MySQL
- **Functionality:**
    - User uploads a **JSON file** or fills out a form that contains structured data.
    - Backend reads the JSON and **saves** the data into a **MySQL database**.
- **Endpoints (Spring Boot):**
    - POST /api/json/upload: Accepts a JSON file.
    - GET /api/json/records: Fetches all saved records from the DB.
- **Database:**
    - Create a table based on the JSON structure (you'll see an example below).
- **Angular Frontend:**
    - Form with file input + upload button.
    - Form with the required fields.
    - Table view to display parsed and stored records.
- **Backend concepts:**
    - Use Spring Data JPA.
    - Create properties table automatically with Hibernate DDL generation.
- **Frontend concepts:**
    - Allow users to upload JSON or fill out a form.
    - Display property data in a table using Angular Material Table.
- **Example JSON Upload:**

**File:** property_apartmentA.json:

{

"propertyId": "APT001",

"address": "123 Main St, Springfield",

"price": 250000,

"propertyType": "Apartment",

"bedrooms": 2,

"bathrooms": 1,

"squareFeet": 850,

"listedDate": "2025-04-01"

}

- **Corresponding Database Table:**

| Field | Type |
|---|---|
| id | AUTO_INCREMENT PRIMARY KEY |
| property_id | VARCHAR(50) |
| address | VARCHAR(255) |
| price | INT |
| property_type | VARCHAR(50) |
| bedrooms | INT |
| bathrooms | INT |
| square_feet | INT |
| listed_date | DATE |

## 🌐 App 3: "Central Management Portal"

- **Tech Stack:** Angular + Spring Boot + MySQL
- **Functionality:**
    - Get files → calls App 1 via API.
    - Get JSON data → calls App 2 via API.
    - Keeps data up to date.
    - Allows users to:
        - View:
            - List of uploaded files (from App 1).
            - List of ingested data records (from App 2).
- **Communication:**
    - **Consume** the APIs of App 1 and App 2 using **REST API calls**.
- **Endpoints (Spring Boot, if needed internally):**
    - Might include simple proxy endpoints if needed.
- **Angular Frontend:**
    - Dashboard with:
        - **Section 1**: List Uploaded Files.
        - **Section 2**: List Saved Data.
- **Backend concept:**
    - Find an optimal way to draw data from the other apps, store them and keep them up to date.

## Additional Requirements / Considerations

- Backend should validate:
    - Maximum file size (e.g., 5MB).
    - JSON structure validation (mandatory fields must exist).
- Clear error handling (e.g., if upload fails or JSON format is invalid).
- Proper folder creation/checking before saving files if the resource folder doesn't exist.
- Include CORS configuration so that apps can communicate across different ports locally.
- Document:
    - How to run each app locally (port numbers, frontend/backend).
    - API Endpoints list in a simple README.

### Run and Test Locally

1. Start **MySQL** and create a database.
2. Run each backend project (mvn spring-boot:run or from your IDE).
3. Serve each Angular frontend (ng serve --port 4201/4202/4203).
4. Use different ports for each Angular app to avoid conflicts.
5. Test APIs with **Postman** if needed before connecting them.

### Example Target

| App | Port (Backend) | Port (Frontend) | Description |
|---|---|---|---|
| File Storage Service | 8081 | 4201 | Upload/View files |
| JSON Data Ingestion | 8082 | 4202 | Upload/View JSON records |
| Central Portal | 8083 | 4203 | Upload/View both |

## 📚 Outcomes (What will be learned from this project)

By completing this project, you will gain strong experience in:

### 🔵 Backend Skills (Spring Boot)

- Setting up Spring Boot projects for REST APIs.
- File upload handling (MultipartFile) and saving files to the server filesystem.
- Parsing JSON files and mapping them to Java entities.
- Using JPA and Hibernate to persist data into a **MySQL** database.
- Database CRUD operations with Spring Data JPA.
- Handling exceptions (upload errors, parsing errors, database errors).
- Application configuration (application.properties, environment setup).
- Inter-service communication using **RestTemplate**.
- Structuring scalable and modular backend codebases.

### 🔵 Frontend Skills (Angular)

- Creating forms and uploading files and JSON files.
- Connecting Angular frontend with RESTful Spring Boot APIs.
- Handling file uploads using FormData.
- Displaying lists of files and database entries in **tables** (using *ngFor, simple UI tables).
- Basic UI/UX: form validation, user feedback (error/success messages).
- Angular service-layer architecture for API communication (HttpClient usage).
- Managing state across components (upload → refresh lists).

### 🔵 System Architecture and Integration

- Designing a **multi-app modular system** where services communicate via APIs.
- Understanding the separation of concerns: file storage, data parsing, centralized management.
- Working with **different services** and coordinating frontend and backend separately.

- Deployment considerations: running multiple Spring Boot apps and Angular apps locally.
- Basic understanding of REST API best practices.

🔵 **Bonus soft skills**

- Planning: structuring work into logical milestones (first local storage, then DB storage, then integration).
- Debugging: handling common issues (connection refused, CORS, JSON parsing failures, upload size limits).
- Documentation: preparing READMEs, writing clean API instructions.

## 🛠️ Prerequisites (Knowledge needed to successfully complete the project)

To work efficiently and independently on this project, you should already have:

- **Backend**
  - Good understanding of Java programming basics (classes, objects, collections).
  - Experience with Spring Boot (controllers, services, dependency injection).
  - Familiarity with handling HTTP requests and responses (especially POST with @RequestBody and @RequestParam).
  - Basic experience with file I/O in Java (InputStream, File classes).
  - Basic knowledge of using MySQL (setting up schemas, tables, user privileges).
- **Frontend**
  - Solid understanding of Angular fundamentals (components, modules, services).
  - Experience working with Angular forms (Template-driven or Reactive Forms).
  - Experience using HttpClient in Angular to communicate with APIs.
  - Basic Material UI usage for styling forms and tables.
- **General**
  - Experience running multiple projects locally (different ports).
  - Git basics (pull, push, branching if needed).
  - Ability to read documentation (e.g., Spring Boot annotations, Angular HttpClient guide).

## 🚀 Useful resources

Some useful resources to start:

**1. Java Fundamentals – Fast Track**

You need **just enough Java** to understand classes, exceptions, and simple data structures.

- 📄 **Java Tutorial for Beginners** (Java 17 or later, crash course):
  - https://www.w3schools.com/java/
  - *(focus mainly on: Variables, Methods, Classes/Objects, Lists, Exception handling)*
- 🎥 **Java Full Course for Beginners (YouTube - BroCode)**:
  - https://www.youtube.com/watch?v=23HFxAPyJ9U&list=PLZPZq0r_RZOOj_NOZYq_R2PECIMglLemc
  - (Good style for people who want fast, applied learning.)

---

**2. Spring Boot Essentials**

Core target: **Learn to build REST APIs with file upload and database save.**

- 📄 **Spring Boot Official Getting Started Guide**:
  - https://spring.io/guides/gs/rest-service/
  - (Good for understanding how to make a REST API)
- 🎥 **Spring Boot Crash Course (YouTube - Amigoscode)**:
  - https://www.youtube.com/watch?v=Cw0J6jYJtzw
  - (Covers everything from setting up Spring Boot to connecting to MySQL)
- 📄 **File Upload with Spring Boot (Guide)**:
  - https://www.baeldung.com/spring-file-upload
  - *(Exactly what you need to upload files to backend resources folder.)*

---

**3. Angular + Spring Boot Full Example (Optional)**

In case you want a **full end-to-end example** to understand how frontend communicates with backend.

- 🎥 **Angular + Spring Boot CRUD Full Stack App (YouTube - Amigoscode)**:
- https://www.youtube.com/watch?v=Gx4iBLKLVHk