Software Engineering and Development

M1: DIA, OCC, CCC
2025-2026

Lab session 6: Build REST APIs and Connect them to MongoDB Atlas

Objectives:

In this lab, you will:

- strengthen and complete the REST APIs you started in the previous session,
- implement full CRUD operations (GET, POST, PUT, DELETE),
- adapt every route to **your own project**, based on the Jira tasks, epics, and use cases you already created,
- use the hospital example only as a reference model,
- install MongoDB Atlas and connect your backend to a real cloud database,
- migrate one entity from in-memory arrays to an Atlas collection,
- test your endpoints with Vitest and Supertest,
- follow the DevOps workflow: branch → commit → push → Pull Request → Continuous Integration.

1 Complete Your REST APIs

1.1 Identify Your Project Entities

Using your Jira board:

- select the **core entities** of your project (e.g., Book, Vehicle, Session, Member),
- identify the fields each entity must contain,
- ensure they match your use cases and acceptance criteria.

The hospital example below is only a model, adapt everything to your project.

1.2 Define the CRUD Endpoints

Each entity must expose the following routes:

Action	Method	Route
List all	GET	/api/<entity>
Retrieve by ID	GET	/api/<entity>/:id
Create	POST	/api/<entity>
Update	PUT	/api/<entity>/:id
Delete	DELETE	/api/<entity>/:id

1.3 Implementing CRUD

Below is an example for a **Doctor** entity. You must implement the same CRUD pattern for your entities.

```
// routes/doctor.routes.js
import express from "express";
const router = express.Router();

let doctors = [
  { id: 1, name: "Dr. Sarah Lee", specialty: "Cardiology" },
  { id: 2, name: "Dr. Amir Khan", specialty: "Pediatrics" }
];

// GET all
router.get("/", (req, res) => {
  res.status(200).json(doctors);
});

// GET by ID
router.get("/:id", (req, res) => {
  const id = Number(req.params.id);
  const doctor = doctors.find(d => d.id === id);
  if (!doctor) return res.status(404).json({ error: "Doctor not
     found" });
  res.json(doctor);
});

// POST
router.post("/", (req, res) => {
  const { name, specialty } = req.body;
  if (!name || !specialty) {
    return res.status(400).json({ error: "Missing required fields"
       });
  }
  const newDoctor = { id: doctors.length + 1, name, specialty };
  doctors.push(newDoctor);
  res.status(201).json(newDoctor);
});

// PUT
router.put("/:id", (req, res) => {
  const id = Number(req.params.id);
  const index = doctors.findIndex(d => d.id === id);
  if (index === -1) {
    return res.status(404).json({ error: "Doctor not found" });
  }
  doctors[index] = { ...doctors[index], ...req.body };
  res.json(doctors[index]);
});

// DELETE
router.delete("/:id", (req, res) => {
  const id = Number(req.params.id);
  const before = doctors.length;
  doctors = doctors.filter(d => d.id !== id);
  if (doctors.length === before) {
    return res.status(404).json({ error: "Doctor not found" });
  }
  res.status(204).end();
});

export default router;
```

1.4 Error Handling Middleware

```
// app.js
app.use((err, req, res, next) => {
  console.error(err);
  res.status(500).json({ error: "Internal server error" });
});
```

2 API Testing (Vitest + Supertest)

You must write tests for each entity (based on the examples below).

```
import request from "supertest";
import app from "../src/app.js";

describe("Doctors API", () => {
  it("GET /api/doctors returns array", async () => {
    const res = await request(app).get("/api/doctors");
    expect(res.status).toBe(200);
    expect(Array.isArray(res.body)).toBe(true);
  });

  it("POST /api/doctors creates a doctor", async () => {
    const res = await request(app)
      .post("/api/doctors")
      .send({ name: "Dr. Riahi", specialty: "Neurology" });
    expect(res.status).toBe(201);
    expect(res.body).toHaveProperty("id");
  });
});
```

3 Connecting One Entity to MongoDB Atlas

3.1 Atlas Setup

- Create a MongoDB account.
- Create a free-tier MongoDB Atlas cluster.
- Create a database user.
- Allow your IP address.

3.2 Install Dependencies

```
npm install mongodb dotenv
```

3.3 Database Connection Module

```
// db/mongo.js
import { MongoClient } from "mongodb";
import dotenv from "dotenv";

dotenv.config();
const client = new MongoClient(process.env.MONGO_URI);

let db;

export async function connectToDb() {
  await client.connect();
  db = client.db();
}
```

```
export function getDb() {
  return db;
}
```

3.4 Migrate One Entity to MongoDB

Choose one of your entities and replace its in-memory logic with real database operations. Example for “Doctor” (adapt the name and fields to your project):

```
// repositories/doctor.repository.js
import { getDb } from "../db/mongo.js";
import { ObjectId } from "mongodb";

export async function getAllDoctors() {
  return await getDb().collection("doctors").find().toArray();
}

export async function getDoctorById(id) {
  return await getDb()
    .collection("doctors")
    .findOne({ _id: new ObjectId(id) });
}

export async function createDoctor(data) {
  const result = await getDb().collection("doctors").insertOne(data)
    ;
  return { _id: result.insertedId, ...data };
}
```

Adapt your routes to call the repository functions.

4 Deliverables

You must submit:

- full CRUD APIs for your project entities,
- the entities migrated to MongoDB Atlas,
- automated tests (Vitest + Supertest) for all CRUD routes,
- Pull Request linked to your Jira story with passing CI.