

Course Project: e-Hotels

Deliverable 2

by

Li, Xinghao, 300233504

Moominzada, Shabnam, 300325897

Daoudi, Souhail, 300135458

CSI2132 : Databases I

University of Ottawa

Faculty of Engineering

School of Electrical and Computer Science

March 31st, 2025

1. The DBMS and the programming languages used in the implementation of the application.

The DBMS used for this project is PostgreSQL.

The backend of the application is developed using Java and the Spring Boot framework, which simplifies the creation of REST APIs and manages application configuration. For build and dependency management, the project uses Maven, which automates the process of downloading libraries and compiling the project.

The frontend is built using standard web technologies. HTML has been used for the structure of the pages. CSS has been used in this project to style and polish the look of the interface. And JavaScript is used to handle the frontend logic, such as making API calls and interacting with the backend.

The development environment used was Visual Studio Code because of its lightweight nature, its rich plugin ecosystem, its Git integration and its support for multiple-language development.

Version control was handled using Git, with collaboration and backup on GitHub. This allowed easy tracking of changes, as well as branching, and collaboration amongst the team members.

1. Specific steps to install the application

The e-Hotel Booking System is a full-stack web application that allows users to book hotels, view their bookings, manage reservations, and handle employee authentication. It consists of a Spring Boot backend connected to a PostgreSQL database, and a frontend built with HTML/CSS/JS.

Prerequisites

1. Install Java Development Kit (JDK) Download and install JDK 17 or later: Download JDK Verify installation: java -version
2. Install PostgreSQL (Database) Download and install PostgreSQL 17: Download PostgreSQL Create a database named e\_hotel. Open pgAdmin and set up the database
3. Install Maven (Build Tool) Download and install Apache Maven: Download Maven Verify installation: mvn -version
4. Ensure port 8080 is free

Backend Setup:

1. Navigate to backend/e-hotel
2. Run ./mvnw spring-boot:run
3. Confirm server is running at <http://localhost:8080/>

Frontend Setup:

1. Navigate to frontend/
2. Open customer.html in browser **or** Use VS Code Live Server

Database Setup:

1. Open psql
2. Connect to e-hotel Database
3. Run the SQL file

Required data:

1. To run the project and access its full functionality, at least one customer must exist in the database, with a valid fullName for login.
2. Hotels, rooms, and bookings must be present in the database so that the customers can search, book, and interact with the website features.
3. SQL DDL statements used to create the database schema

1. CreateTables.sql

-- Create Tables

-- Hotel Chains

CREATE TABLE HotelChain(

HotelChainId INT PRIMARY KEY,

Address VARCHAR(200) NOT NULL,

NumberOfHotels INT DEFAULT 0,

Email VARCHAR(100),

PhoneNumber VARCHAR(50)

);

-- hotel

CREATE TABLE Hotel(

HotelID INT PRIMARY KEY,

HotelChainId INT NOT NULL,

Email VARCHAR(100),

PhoneNumber VARCHAR(50),

Address VARCHAR(200) NOT NULL,

NumberOfRooms INT,

Rating INT CHECK ( Rating BETWEEN 0 AND 5),

CONSTRAINT fk\_HotelChain

FOREIGN KEY (HotelChainId)

REFERENCES HotelChain (HotelChainId)

ON DELETE CASCADE

);

--Room

CREATE TABLE Room(

RoomID INT PRIMARY KEY,

HotelID INT NOT NULL,

Price DECIMAL(18, 2) NOT NULL CHECK (price >= 0),

Capacity INT NOT NULL CHECK (capacity > 0),

Amenities VARCHAR(200),

SeaView BOOLEAN DEFAULT FALSE,

MountainView BOOLEAN DEFAULT FALSE,

isExtendable BOOLEAN DEFAULT FALSE,

Problems VARCHAR(500),

CONSTRAINT fk\_Room\_Hotel

FOREIGN KEY (HotelID)

REFERENCES Hotel (HotelID)

ON DELETE CASCADE

);

--Employee

CREATE TABLE Employee (

EmployeeID INT PRIMARY KEY,

HotelID INT NOT NULL,

Name VARCHAR(100) NOT NULL,

Address VARCHAR(200),

Role VARCHAR(20) ,

CONSTRAINT fk\_employee\_hotel

FOREIGN KEY (HotelID)

REFERENCES Hotel(HotelID)

ON DELETE CASCADE

);

--Customer

CREATE TABLE Customer(

CustomerID INT PRIMARY KEY,

Name VARCHAR(100) NOT NULL ,

Address VARCHAR(200) NOT NULL ,

RegisterDate DATE NOT NULL

);

--Booking

CREATE TABLE Booking(

BookingID INT PRIMARY KEY ,

CustomerID INT,

RoomID INT,

StartDate Date NOT NULL ,

EndDate Date NOT NULL ,

BookDate DATE NOT NULL ,

Status VARCHAR(20) NOT NULL CHECK ( Status IN ('booked', 'transformed', 'cancelled') ),

Constraint fk\_Booking\_Customer

FOREIGN KEY (CustomerID)

REFERENCES Customer(CustomerID)

ON DELETE SET NULL,

Constraint fk\_Booking\_Room

FOREIGN KEY (RoomID)

REFERENCES Room(RoomID)

ON DELETE SET NULL,

CONSTRAINT chk\_BookingDates CHECK ( Booking.StartDate < Booking.EndDate )

);

--Renting

CREATE TABLE Renting(

RentID INT PRIMARY KEY ,

CustomerID INT,

RoomID INT,

StartDate DATE NOT NULL ,

EndDate DATE NOT NULL ,

Payment DECIMAL(18,2),

Constraint fk\_Renting\_Customer

FOREIGN KEY (CustomerID)

REFERENCES Customer(CustomerID)

ON DELETE SET NULL ,

CONSTRAINT fk\_Renting\_Room

FOREIGN KEY (RoomID)

REFERENCES Room(RoomID)

ON DELETE SET NULL ,

CONSTRAINT chk\_RentingDates check ( Renting.StartDate < Renting.EndDate )

);

2. Indexes.sql

-- It's a common case that we find all rooms for a given hotel

-- Improve speed for that

CREATE INDEX idx\_room\_hotel ON room(hotelid);

--Finding booking for a given room or looking bookings for a specific range of time

--are common cases in life.

CREATE INDEX idx\_booking\_room\_dates ON booking(roomid, startdate, enddate);

--Looking for hotel by geographical perspective is important and common.

CREATE INDEX idx\_hchain\_address ON hotelchain(address);

3.Triggers.sql

--Update number of hotels when inserting and deleting

CREATE OR REPLACE FUNCTION trg\_increase\_room()

RETURNS TRIGGER AS $$

BEGIN

UPDATE Hotel

SET NumberOfRooms = NumberOfRooms + 1

WHERE HotelID = NEW.HotelID;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_room\_insert

AFTER INSERT ON Room

FOR EACH ROW

EXECUTE PROCEDURE trg\_increase\_room();

CREATE OR REPLACE FUNCTION trg\_decrease\_room()

RETURNS TRIGGER AS $$

BEGIN

UPDATE Hotel

SET NumberOfRooms = NumberOfRooms - 1

WHERE HotelID = OLD.HotelID;

RETURN OLD;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_room\_delete

AFTER DELETE ON Room

FOR EACH ROW

EXECUTE PROCEDURE trg\_decrease\_room();

--Update booking status

CREATE OR REPLACE FUNCTION trg\_cancel\_bookings()

RETURNS TRIGGER AS $$

BEGIN

-- If the startDate has passed, make it cancelled.

IF (OLD.status = 'booked') AND (OLD.StartDate < CURRENT\_DATE) THEN

NEW.status := 'cancelled';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_check\_booking\_past

BEFORE UPDATE ON Booking

FOR EACH ROW

WHEN (OLD.status = 'booked') -- Only if old status was 'booked'

EXECUTE PROCEDURE trg\_cancel\_bookings();

--One Manger per hotel

CREATE OR REPLACE FUNCTION trg\_one\_manager\_per\_hotel()

RETURNS TRIGGER AS $$

DECLARE

existing\_managers INT;

BEGIN

IF NEW.role = 'Manager' THEN

SELECT COUNT(\*) INTO existing\_managers

FROM EMPLOYEE

WHERE hotelid = NEW.hotelid

AND role = 'Manager';

IF existing\_managers > 0 THEN

RAISE EXCEPTION 'Hotel % already has a manager!', NEW.hotelid;

END IF;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_employee\_insert\_manager

BEFORE INSERT ON EMPLOYEE

FOR EACH ROW

EXECUTE PROCEDURE trg\_one\_manager\_per\_hotel();

-- no multiple bookings for one room

CREATE OR REPLACE FUNCTION trg\_no\_double\_booking()

RETURNS TRIGGER AS $$

DECLARE

conflict\_count INT;

BEGIN

IF NEW.status = 'booked' THEN

SELECT COUNT(\*)

INTO conflict\_count

FROM booking

WHERE roomid = NEW.roomid

AND status = 'booked'

AND (

NEW.startdate < enddate

AND NEW.enddate > startdate

);

IF conflict\_count > 0 THEN

RAISE EXCEPTION 'Room % is already booked in the overlapping date range', NEW.roomid;

END IF;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_booking\_no\_overlap

BEFORE INSERT OR UPDATE ON booking

FOR EACH ROW

EXECUTE PROCEDURE trg\_no\_double\_booking();

4. Views.sql

--The number of available rooms per area.

CREATE OR REPLACE VIEW v\_available\_rooms\_per\_area AS

SELECT h.address AS area,

COUNT(r.roomid) AS available\_rooms

FROM HOTEL h

JOIN ROOM r ON h.hotelid = r.hotelid

WHERE r.roomid NOT IN (

SELECT b.roomid

FROM BOOKING b

WHERE b.status In ('booked','transformed')

AND CURRENT\_DATE BETWEEN b.startdate AND b.enddate

UNION

SELECT rt.roomid

FROM RENTING rt

WHERE CURRENT\_DATE BETWEEN rt.startdate AND rt.enddate

)

GROUP BY h.address;

--Aggregated capacity of all the rooms of a specific hotel

CREATE OR REPLACE VIEW v\_hotel\_capacity AS

SELECT h.hotelid,

h.address,

SUM(r.capacity) AS total\_capacity,

COUNT(r.roomid) AS total\_rooms

FROM hotel h

JOIN ROOM r ON h.hotelid = r.hotelid

GROUP BY h.hotelid, h.address;