FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Hotel System

Architecture and Database

Project documentation for the purpose of BIE-SWI course.

Authors:

Table of Contents

1. System architecture

The Hotel Room Management System is architected around a centralized server infrastructure, hosted by a third-party service provider for scalability and maintainability.
The system is accessible via a responsive web application, which supports modern web browsers and can be used from any internet-connected device. Both hotel personnel and guests interact with the system through secure HTTPS connections, ensuring data confidentiality and integrity.

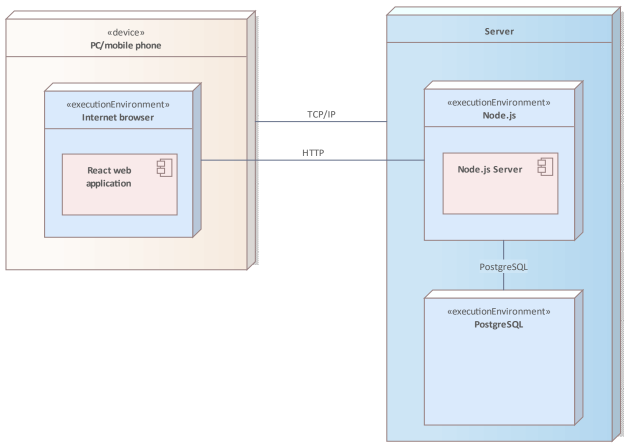

Figure 1 - System architecture

1.1 Server

The server acts as the central backend infrastructure, hosting both the application logic and the database to ensure continuous availability and secure data handling.

1.1.1 Node.js

Node.js is a JavaScript runtime built on Chrome's V8 engine, allowing JavaScript to be run server-side. Its asynchronous, non-blocking architecture makes it highly efficient for handling multiple simultaneous requests. Node.js is also often used with databases like PostgreSQL, which is useful in our case.

1.1.2 PostgreSQL

PostgreSQL is a powerful, open-source relational database that stores and manages all the structured data for the hotel management system, supporting reliable and efficient data transactions.

1.2 PC/mobile phone

The user can access the web application from any internet-enabled device, such as a PC, smartphone, tablet, or any other platform with a web browser and internet connectivity.

1.2.1 Internet browser

Access to the web application is possible through any up-to-date web browser that supports standard web technologies.

1.2.1.1 React web application

The front-end framework used is React, chosen for its component-based architecture and its efficiency in building dynamic, responsive user interfaces.

2. Logic Architecture

This chapter outlines the architecture of the Hotel Room Management System web application.

The application is built using the following technologies:

- React Framework
- Node.js Framework
- PostgreSQL Database

The architecture is organized into two distinct applications:

Frontend: Responsible for presenting application data and handling all associated logic.
- Presentation Layer: Displays data to the user and manages user interactions.
- Services Layer: Handles communication with the backend and provides utility functions for the frontend.

Backend: Manages data persistence and processes the application's core logic.
- Business Layer: Handles business logic and processes requests from the frontend.
- Data Layer: Responsible for interacting with the PostgreSQL database to store and retrieve data.

This separation ensures a clear division of responsibilities and supports scalability, maintainability, and efficient data management.

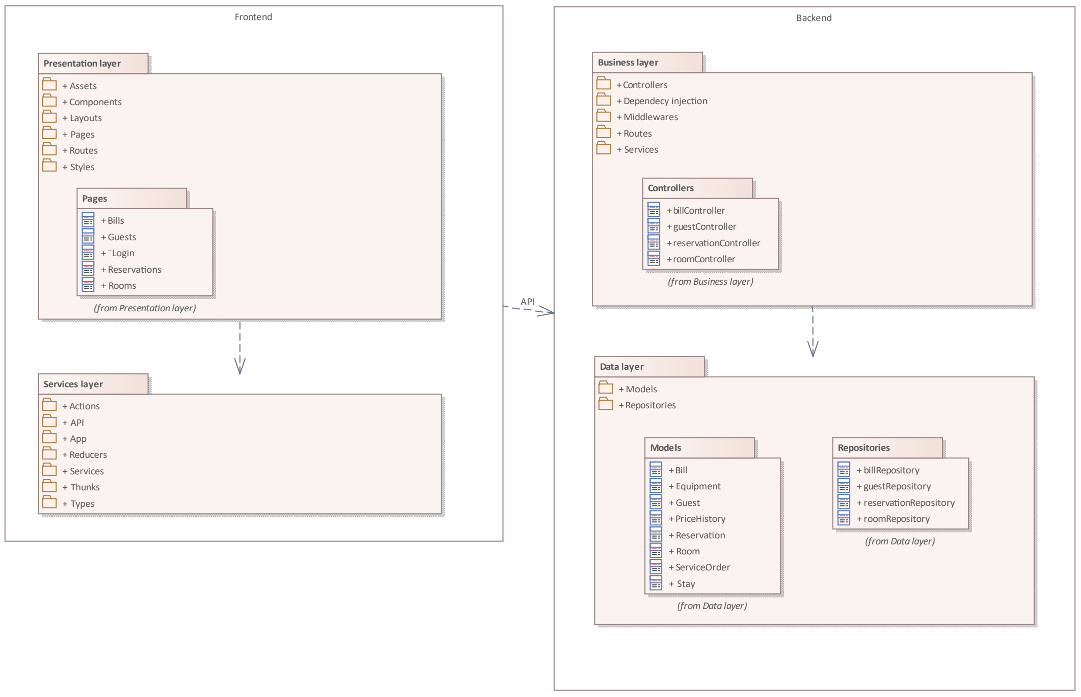

Figure 2 - Architecture

2.1 Presentation layer

The presentation layer is responsible for sharing the information with a user or other systems. It consists of a web user interface.

2.1.1 Assets

Contains static files used across the app, such as images, icons, fonts, and other media. These are not generated dynamically and support the visual presentation of the UI.

2.1.2 Components

React components will represent the user interface. For example, there is a component to display available rooms, one to manage reservations, and so on.

2.1.3 Layouts

Defines reusable page structures or templates that wrap around page content (e.g., headers, footers, sidebars). Helps maintain consistent UI across pages.

2.1.4 Pages

Contains top-level views mapped to routes. Each file typically represents a full page composed of components and layouts. There's one for Bills, Guests, Login, Reservation and Rooms.

2.1.4.1 ¨Login

Page where the user can securely log into their account.

2.1.4.2 Bills

Page where the user can access their bills.

2.1.4.3 Reservations

Page where the user can access to their reservations.

2.1.4.4 Rooms

Page where the user can see all the available rooms.

2.1.5 Routes

Defines the application's navigation structure and maps URL paths to page components.

2.1.6 Styles

Stores global styles, theme definitions, and shared CSS files used across the application.

2.2 Services layer

2.2.1 Actions

Defines Redux action creators and action types used to trigger state changes and asynchronous flows.

2.2.2 API

The api folder contains all the necessary code for managing interactions with the external backend API. The folder helps centralize all API-related logic for easier maintenance and scalability.

2.2.3 App

This package handles the setup of global state management using Redux. It configures the store and exports types for the application state and dispatch actions, centralizing state management logic.

2.2.4 Reducers

The reducers folder contains the Redux reducers responsible for managing state updates across the application. For example, there are separate reducers for handling guests, reservations, and other key features of the app.

2.2.5 Services

This package includes utility functions or helper methods used throughout the application for various common tasks.

2.2.6 Thunks

The thunks folder contains Redux thunk functions that handle asynchronous operations, such as API calls or side effects, and dispatch actions based on the results. These thunks help manage complex logic outside of the reducers.

2.2.7 Types

This package contains the TypeScript type definitions used throughout the application. It helps ensure type safety by defining interfaces, types, and enums for different data structures and components across the app, such as guest, reservation, amenity...

2.3 Business layer

The business layer is responsible for implementing the core business logic and system operations. It includes service managers that encapsulate the system's behavior, as well as value objects used to transfer data between the business and presentation layers.
This layer is implemented using Node.js.. On the frontend, the React framework is used for the presentation layer, enabling efficient data exchange with the business layer via RESTful APIs.

2.3.1 Controllers

2.3.1.1 billController

Handles HTTP requests related to bills in a web application:

- Getting all bills or filtering bills based on different criteria like status, guest_id, or stay_id.
- Getting a single bill by its ID.
- Creating a new bill by processing the data provided in the request.
- Updating a bill based on its ID and the provided data.
- Updating the status of a bill.

- Processing payments for bills, requiring both the payment amount and method.
- Checking for overdue bills.

2.3.1.2 guestController

Handles HTTP requests related to guest management in a web application:
- Getting all guests: Retrieves a list of all guests from the service.
- Getting a single guest by ID: Retrieves a specific guest's information by their ID.
- Creating a new guest: Creates a new guest record using data provided in the request.
- Updating a guest's information: Updates the guest's details (like email or other fields) based on their ID.

2.3.1.3 reservationController

Handles HTTP requests related to reservation management in a web application:
- Getting all reservations: Retrieves all reservations, with the option to filter by status if provided in the query parameters.
- Getting a single reservation by ID: Fetches details of a specific reservation by its ID.
- Creating a new reservation: Allows the creation of a new reservation with the provided data (such as check-in/check-out dates and room availability).
- Updating a reservation: Updates an existing reservation based on its ID and the provided new data.
- Updating reservation status: Changes the status of an existing reservation (e.g., confirmed, canceled).

2.3.1.4 roomController

Handles HTTP requests related to room management in a web application:
- Get all rooms: Retrieves all rooms with their details.
- Get a single room by ID: Fetches the details of a room using its ID.
- Get current reservation and guest for a room: Provides information about the current reservation and the guest in a specific room.
- Get price history for a room: Fetches the price history for a room by its ID.
- Get equipment/amenities for a room: Retrieves the list of equipment or amenities available for a specific room.
- Create a new room: Allows for the creation of a new room with the provided data.
- Update a room: Updates the room details for a specific room based on its ID.
- Delete a room: Deletes a room, with checks to ensure it doesn't have active reservations.
- Get available rooms for a date range: Returns the list of available rooms for a specified check-in and check-out date range.
- Update room status: Changes the status of a room (e.g., available, unavailable) based on the provided status.

2.3.2 Dependecy injection

This package is responsible for managing the application's dependencies through a container. It centralizes the registration and injection of services, repositories, and controllers to ensure that each component receives the necessary dependencies without tightly coupling them.
For example, repositories like RoomRepository, GuestRepository, and ReservationRepository are registered, as well as corresponding services and controllers for handling business logic and user interactions.

2.3.3 Middlewares

The middlewares folder contains functions that handle requests before they reach the controller. These middlewares perform tasks such as validating user input, authenticating users, logging requests, or handling errors.

2.3.4 Routes

The routes folder in the backend defines the application's API endpoints and maps them to their corresponding controller actions. Each route specifies the HTTP method (GET, POST, PUT, DELETE) and the path for the endpoint, as well as the controller and method to handle the request

2.3.5 Services

The services folder contains business logic for handling different aspects of the Hotel Room Management System.

2.4 Data layer

The Data Layer is responsible for data persistence. In this application, it handles communication with the PostgreSQL database to store and retrieve information related to hotel rooms, clients, reservations, and other domain entities.

2.4.1 Models

The models in the application contains classes representing the main entities of the system (e.g., Room, Reservation, Customer, etc.).

2.4.1.1 Bill

Model representing the bill:
- bill_id
- total_amount
- status
- stay_id

2.4.1.2 Equipment

Model representing the room equipment:
- id
- name
- price
- room_id

2.4.1.3 Guest

Model representing the guest user:
- id
- first_name
- last_name
- email
- phone_number

2.4.1.4 PriceHistory

Model representing the price history of the room:
- price_history_id
- start_date
- end_date
- price
- room_id

2.4.1.5 Reservation

Model representing the reservation and the associated values:

- reservation_id
- check_in_date
- check_out_date
- status
- room_id
- guest_id

2.4.1.6 Room

Model representing the hotel room:
- id
- type
- status
- capacity
- price_per_night

2.4.1.7 ServiceOrder

Model representing a service the guest used during their stay:
- service_order_id
- service_name
- price
- date_time
- stay_id

2.4.1.8 Stay

Model representing the stay of the guest:
- stay_id
- check_in_date
- check_out_date
- reservation_id

2.4.2 Repositories

The repositories folder contains data access logic for interacting with the database. Each repository is responsible for managing a specific data model, such as rooms, guests, or reservations.

2.4.2.1 billRepository

Interacts with the databas to handle data operations related to bills:
- Finding Bills with Details
- Finding a Single Bill with Details
- Finding Bills by Status
- Finding Bills for a Specific Stay
- Finding Bills for a Specific Guest
- Updating Bill Status
- Creating a Payment for a Bill
- Getting Total Payments for a Bill

2.4.2.2 guestRepository

Defines operations related to guest data management:

- Retrieve the list of all guests
- Find a guest by their ID
- Check if a guest has active reservations

2.4.2.3 reservationRepository

Handles reservation-specific data operations:

- Retrieve reservations with detailed information
- Find a reservation with detailed information by ID
- Find reservations by their status
- Find reservations made by a specific guest
- Find reservations associated with a specific room
- Update the status of a reservation
- Check if a room is available for a specified date range.
- Create a stay record for a reservation
- Update stay information with check-out details

2.4.2.4 roomRepository

Handles room-specific data operations in an application:

- Retrieve rooms with detailed information
- Find a specific room with detailed information by ID
- Find the current reservation and guest information for a room
- Retrieve price history for a specific room
- Find equipment and amenities associated with a room
- Create a new price history record for a room
- End the current price history for a room
- Find room IDs with active reservations during a date range
- Find available rooms for a given date range.
- Check if a room has any active reservations

3. Database Model

This chapter describes the structure of the database schema where all the information about the books, authors, loans and reservation is stored.

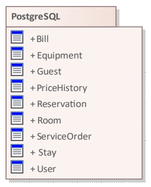

Figure 3 - Database Model

3.1 PostgreSQL

This package describes the structure of the PostgreSQL database used by the Library system.

The schema is not complete, it describes only the important part of the schema currently implemented.

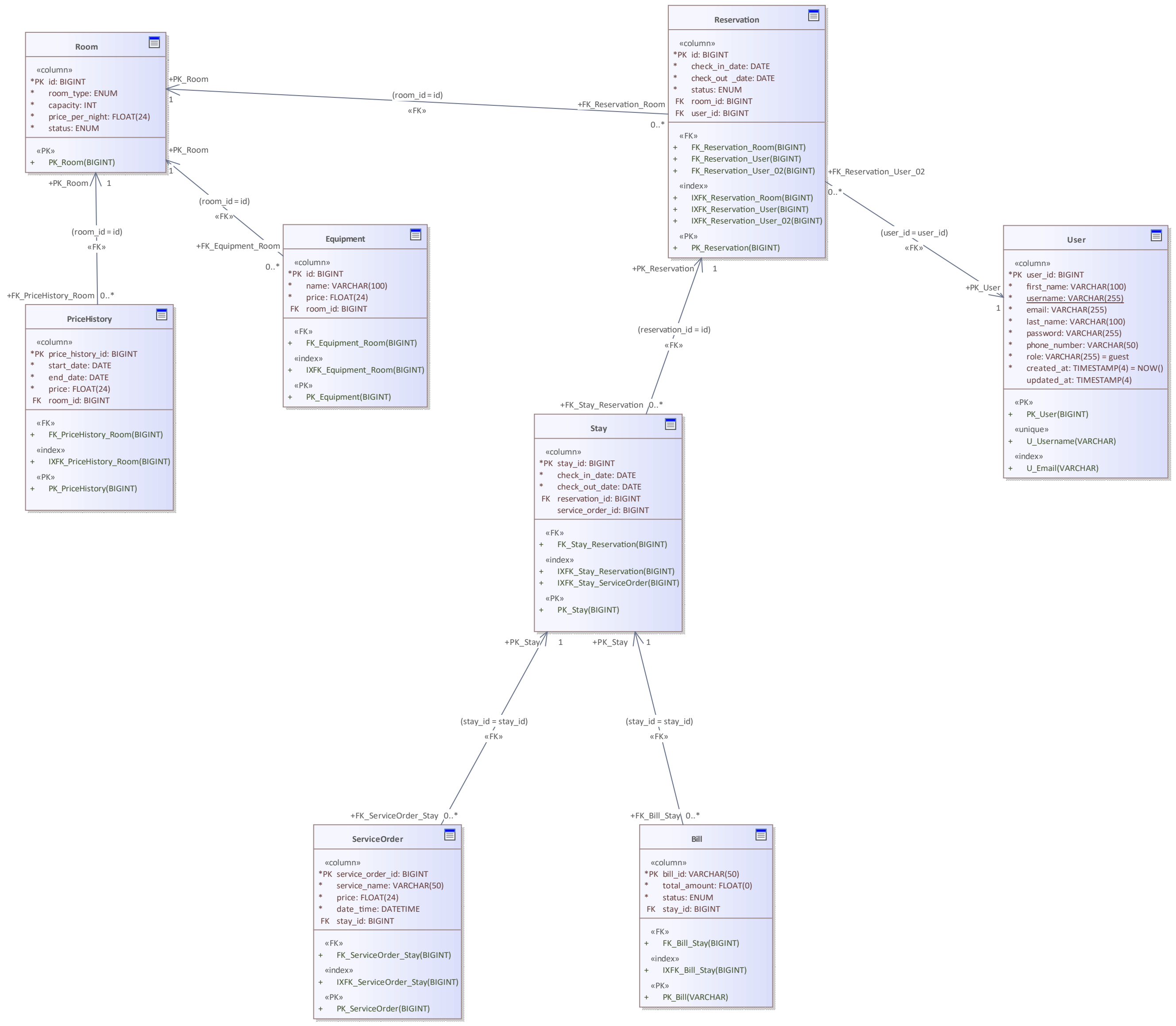

Figure 4 - PostgreSQL

The role of the user (guest or hotel staff).

3.1.1 Bill *«table»*

The Bill table contains information about the billing for a stay, including the total amount and the status of the bill.
Purpose: It is used to manage the financial aspect of a guest's stay by tracking the charges associated with their stay, including room charges and additional services.

Column name	Data type	Not null	Description
bill_id	VARCHAR(50)	True	Unique identifier for each bill (Primary Key)
total_amount	FLOAT(0)	True	The total amount charged for the stay, including room charges and any services.
status	ENUM	True	The current status of the bill (e.g., Paid, Unpaid).
stay_id	BIGINT	False	The stay that the bill is associated with (Foreign Key referencing Stay).

3.1.2 Equipment *«table»*

The Equipment table tracks the equipment available in rooms, such as furniture and appliances. Each item is linked to a specific room.

Purpose: It helps to manage and track the equipment present in each room and its associated price.

Column name	Data type	Not null	Description
id	BIGINT	True	Unique identifier for each piece of equipment (Primary Key).
name	VARCHAR(100)	True	The name of the equipment (e.g., TV, bed).
price	FLOAT(24)	True	
room_id	BIGINT	False	The room where the equipment is located (Foreign Key referencing Room).

3.1.3 Guest *«table»*

The Guest table contains details about the guests who stay at the hotel. This table holds personal information like the name, email, and phone number.

Purpose: It is used to store information about guests for managing reservations and stays.

Column name	Data type	Not null	Description
id	BIGINT	True	Unique identifier for each guest (Primary Key).
email	VARCHAR(100)	True	The email address of the guest (unique).
user_id	BIGINT	False	

3.1.4 PriceHistory *«table»*

The PriceHistory table tracks changes in the prices of rooms over time. Each entry records the price of a room during a specific period.

Purpose: It helps keep historical pricing data for each room, which can be used for analysis or reporting.

Column name	Data type	Not null	Description
price_history_id	BIGINT	True	Unique identifier for each price record (Primary Key).
start_date	DATE	True	The date when the price change starts.
end_date	DATE	True	The date when the price change ends.
price	FLOAT(24)	True	The price of the room during the period.
room_id	BIGINT	False	The room associated with the price change (Foreign Key referencing Room).

3.1.5 Reservation *«table»*

The Reservation table stores information about the reservations made by guests. It links a guest to a room for a specific duration.

Purpose: It helps track which room has been reserved by a specific guest and for what time period.

Column name	Data type	Not null	Description
id	BIGINT	True	Unique identifier for each reservation (Primary Key).
check_in_date	DATE	True	The date when the guest is expected to check-in.
check_out__date	DATE	True	The date when the guest is expected to check-out.

Column name	Data type	Not null	Description
status	ENUM	True	The current status of the reservation (e.g., Confirmed, Canceled).
room_id	BIGINT	False	The room reserved for the guest (Foreign Key referencing Room).
user_id	BIGINT	False	The ID of the guest who made the reservation (Foreign Key referencing Guest).

3.1.6 Room *«table»*

The Room table contains details about the rooms available in the hotel. Each room has a unique identifier (room_id) and is associated with attributes like its type, capacity, and price per night.

Purpose: It is used to manage the information of the rooms in the hotel, such as how many people it can accommodate and the price for each night.

Column name	Data type	Not null	Description
id	BIGINT	True	Unique identifier for the room (Primary Key).
room_type	ENUM	True	The type of room (e.g., Single, Double, Suite).
capacity	INT	True	The number of people the room can accommodate.
price_per_night	FLOAT(24)	True	The cost for one night's stay in the room.
status	ENUM	True	

3.1.7 ServiceOrder *«table»*

The ServiceOrder table records the services ordered by guests during their stay (e.g., room service).

Purpose: It helps track any additional services provided to the guest and the associated cost.

Column name	Data type	Not null	Description
service_order_id	BIGINT	True	Unique identifier for each service order (Primary Key).
service_name	VARCHAR(50)	True	The name of the service (e.g., Food, Laundry).
price	FLOAT(24)	True	The price of the service.
date_time	DATETIME	True	The date and time when the service was requested.
stay_id	BIGINT	False	The stay for which the service was ordered (Foreign Key referencing Stay).

3.1.8 Stay *«table»*

The Stay table stores information about the actual stay of a guest in a room, including check-in and check-out dates.

Purpose: It is used to track when a guest occupies a room and for how long.

Column name	Data type	Not null	Description
stay_id	BIGINT	True	Unique identifier for each stay (Primary Key).
check_in_date	DATE	True	The actual check-in date of the guest.
check_out_date	DATE	True	The actual check-out date of the guest.
reservation_id	BIGINT	False	ID linked to Reservations table
service_order_id	BIGINT	False	ID linked to ServiceOrder table

3.1.9 User *«table»*

The User table contains details about the users that use the system . This table holds information like username, email, password and role that can be either staff or guests.
Purpose: It is used to store information about guest and staff.

Column name	Data type	Not null	Description
user_id	BIGINT	True	Unique identifier for each user (Primary Key).
first_name	VARCHAR(100)	True	The first name of the guest.
username	VARCHAR(255)	True	The unique username of the user.
email	VARCHAR(255)	True	The email address of the user (unique)
last_name	VARCHAR(100)	True	The last name of the user.
password	VARCHAR(255)	True	The hashed password of the user.
phone_number	VARCHAR(50)	True	The phone number of the user.
role	VARCHAR(255)	True	
created_at	TIMESTAMP(4)	True	
updated_at	TIMESTAMP(4)	False	

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE