Fayoum University
Faculty of Science
Mathematics department
Mathematics and Computer Science

City Mall

(Application&&Security System)

A graduation project document submitted to the Dep. of Computer Science as partial fulfillment for the Requirement for the Degree of Bachelor in Computer Science

By

Ahmed Hamada Makhouf

Amr Khaled

Mohammed Ashraf

Mustafa Hamdy

Supervised by

Dr. Mohamed Abd-Elbaky

6/2023

Backend Part

Table of content

1- Introduction

Welcome to the Backend Documentation for city mall. This documentation aims to provide you with a comprehensive understanding of the backend system and its functionalities. It is intended for developers and technical stakeholders involved in working with the backend infrastructure and APIs.

The backend system serves as the backbone of our application, handling data storage, processing, and the communication between the frontend and various external services. It plays a crucial role in ensuring the smooth operation and functionality of the overall application.

Key Features:
- Robust API layer for interacting with the frontend and external services.
- Secure authentication and authorization mechanisms to protect user data.
- Efficient data storage and retrieval through the database layer.
- Integration with third-party services and APIs to enhance application capabilities.
- Scalable architecture to handle increasing traffic and workload.
- Automated deployment and continuous integration processes for seamless updates.
- Logging and monitoring systems to track system health and performance.

Target Audience:
This documentation is designed for supervisor doctor, judgmental commission, developers and technical stakeholders who are responsible for maintaining, extending, or integrating with the backend system. It assumes a basic understanding of web development concepts, databases, and APIs. Familiarity with programming languages such as JavaScript or Java would be beneficial.

Throughout this documentation, we will delve into the architecture, API references, database structure, security measures, deployment processes, and other essential aspects of the backend system. By following this guide, you will gain the knowledge and insights necessary to effectively work with the backend infrastructure and contribute to the success of our application.

Next, we will explore the system architecture in detail, providing an overview of how the backend system is structured and the technologies used.

2- Used Technology

This section provides an overview of the technology stack used in the backend system of City Mall. Understanding the technologies involved will give you a better understanding of the system's architecture and development environment.

Backend Technology Stack:

1. Node.js: Node.js is a JavaScript runtime built on the V8 JavaScript engine. It allows running JavaScript code outside of a web browser and is commonly used for building server-side applications.

2. Express.js: Express.js is a fast and minimalist web application framework for Node.js. It provides a robust set of features for building web APIs, handling routes, and managing middleware.

3. Database System: PostgreSQL. The backend system utilizes a database system for persistent data storage and retrieval.

4. Database System: PostgreSQL is a powerful and feature-rich open-source relational database management system (RDBMS). It provides a reliable and scalable solution for storing and retrieving structured data.

5. Sequelize: Sequelize is a popular Object-Relational Mapping (ORM) library for Node.js. It simplifies database interactions by providing an intuitive API for querying and manipulating data in PostgreSQL using JavaScript objects and models.

6. Authentication and Authorization: JSON Web Tokens (JWT) is an industry-standard for representing claims securely between two parties. It is commonly used for authentication and authorization in web applications. JWTs consist of encoded JSON objects that are digitally signed to ensure integrity.
This ensures secure access to protected routes and resources within the backend system.

7. Other Libraries and Packages:

- **Body-parser**: body-parser simplifies parsing request bodies, enabling easy access to JSON or URL-encoded data.
- **cors**: cors handles Cross-Origin Resource Sharing, ensuring proper security and access control for requests originating from different origins.
- **bcrypt**: bcrypt provides a secure way to hash and compare passwords, safeguarding user credentials.
- **multer**: multer simplifies handling file uploads, allowing users to submit files

Key Features of Node.js with express:

1. Asynchronous and Event-Driven: Node.js uses a non-blocking, event-driven architecture, making it highly efficient and scalable. It can handle a large number of concurrent connections without blocking the execution of other tasks.

2. Single-Threaded and Non-Blocking I/O: Node.js operates on a single thread, using non-blocking I/O operations. This allows it to handle multiple requests simultaneously, resulting in improved performance and responsiveness.

3. JavaScript Everywhere: Node.js enables the use of JavaScript on both the client-side and server-side, allowing developers to write server-side and client-side code using the same programming language. This simplifies development and promotes code reusability.

4. NPM Ecosystem: Node.js has a vast and vibrant ecosystem supported by npm (Node Package Manager). It offers a wide range of open-source libraries and modules that can be easily integrated into Node.js applications, accelerating development and enhancing functionality.

5. Scalability: Node.js is highly scalable due to its event-driven, non-blocking I/O model. It can efficiently handle thousands of concurrent connections, making it suitable for building real-time applications and microservices architectures.

Key Features of Express.js:

1. Minimal and Lightweight: Express.js is a minimalistic web application framework for Node.js. It provides essential features for building web applications without imposing unnecessary overhead, making it lightweight and easy to learn.

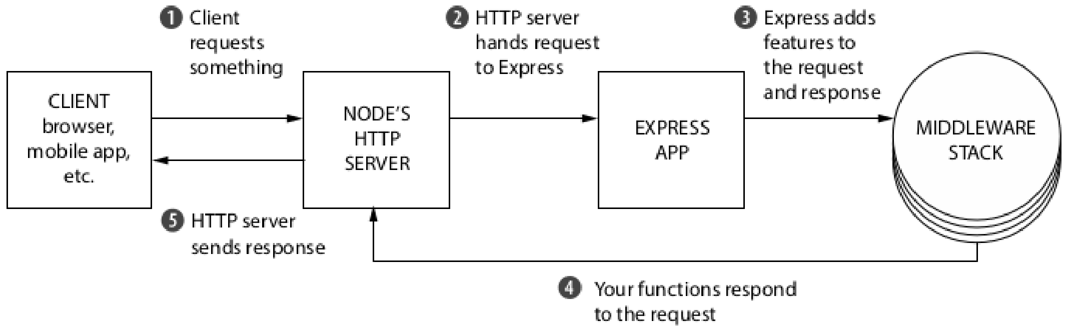
2. Routing: Express.js simplifies the handling of HTTP requests through its routing mechanism. Developers can define multiple routes and associate them with specific HTTP methods, making it easier to handle different types of requests and execute corresponding actions.

3. Middleware Support: Express.js offers robust middleware support, allowing developers to define custom middleware functions that can intercept and modify incoming requests or outgoing responses. This enables functionality such as authentication, logging, error handling, and request processing.

4. Extensibility: Express.js can be extended using middleware and third-party packages from the npm ecosystem. This allows developers to add additional functionality to their applications easily and leverage existing community-driven solutions.

5. RESTful API Development: Express.js provides a solid foundation for building RESTful APIs. It allows developers to define routes, handle HTTP methods, and perform CRUD operations on resources, making it a popular choice for developing API-driven applications.

These key features make Node.js and Express.js powerful tools for developing scalable, efficient, and feature-rich web applications. Their combination provides a robust environment for server-side JavaScript development, enabling developers to build high-performance applications and APIs.



1 Client requests something
2 HTTP server hands request to Express
3 Express adds features to the request and response
CLIENT browser, mobile app, etc.
NODE'S HTTP SERVER
EXPRESS APP
MIDDLEWARE STACK
5 HTTP server sends response
4 Your functions respond to the request

Key Features of Sequelize with PostgreSQL:

- Schema definition: Sequelize allows defining database tables, columns, and relationships using JavaScript models, making it easy to work with PostgreSQL schemas.
- Data validation: Sequelize provides built-in validation mechanisms to ensure data integrity and consistency before saving it to the PostgreSQL database.
- Query building: Sequelize offers a fluent API for constructing complex SQL queries, including joins, conditions, and aggregations, without the need to write raw SQL statements.
- Migrations: Sequelize simplifies database schema management through the use of migrations. Migrations enable seamless updates to the database schema while preserving existing data.
- Associations and relationships: Sequelize supports defining associations between models, such as one-to-one, one-to-many, and many-to-many relationships, making it convenient to work with related data.

3- Project Architecture

The backend application for the City Mall security project follows the Model-View-Controller (MVC) architectural pattern, which provides a logical and modular separation of concerns. This pattern divides the application into three interconnected components: the Model, View, and Controller, each with its specific responsibilities.

1. Model:
In the context of the City Mall security project, the Model component represents the data layer of the application. It encapsulates the logic for data storage, retrieval, and manipulation related to user data, visits, cars, shops, cinemas, offers, movies, issues, and employee attendance. The Model handles interactions with the PostgreSQL database, ensuring the integrity and consistency of the stored data. It incorporates data access code, query execution, and data validation specific to the entities within the City Mall environment.

2. View:
The View component focuses on presenting the application's user interface to the client. In the City Mall security system, the View involves generating and constructing JSON responses, or rendering data in various formats for display. It formats the data received from the Controller in a way that is suitable for presentation to security personnel, administrators, and other system users. The View is decoupled from the underlying data

and focuses solely on displaying information effectively.
3. Controller:
The Controller acts as the intermediary between the Model and View components. It receives and handles incoming requests related to managing user data, visits, cars, shops, cinemas, offers, movies, issues, and employee attendance. The Controller processes the data, interacts with the Model to fetch or modify data from the database, and determines the appropriate response to send back to the client. It performs business logic such as authentication, authorization, and data transformation specific to the City Mall security system. The Controller ensures that the correct data is retrieved or updated and prepares it for presentation by the View.

By adopting the MVC structure in the backend application of the City Mall security project, you achieve several benefits:

1. Separation of Concerns: The MVC pattern separates the responsibilities of data manipulation, user interface, and control flow. This separation allows each component to focus on its specific task, making the codebase more organized and easier to understand.

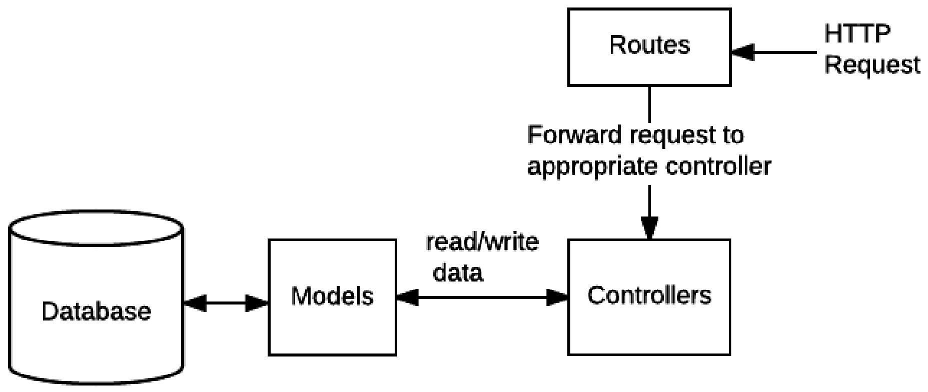
2. Code Reusability: The Models and Views developed for the City Mall security system can be reused across different parts of the application or even in future projects. The decoupling of the View from the data layer enables flexibility in changing the presentation layer without affecting the underlying data structures.

3. Maintainability: The modular structure of MVC makes it easier to maintain and update the City Mall security system. Changes made to one component have minimal impact on the others, promoting a more manageable and scalable codebase.

4. Collaboration: The MVC pattern facilitates collaboration among multiple developers working on the City Mall security system. Each developer can work on a specific component without interfering with others, resulting in parallel development and faster progress.

5. Testability: The separation of concerns in MVC makes it easier to write unit tests for each component individually. Testing the Models, Views, and Controllers independently helps ensure the correctness and reliability of the City Mall security system.

In conclusion, the adoption of the MVC pattern in the backend application of the City Mall security project provides a robust and scalable architecture. It separates the concerns of data manipulation, user interface, and control flow, leading to better organization, code reusability, maintainability, and collaboration among developers. By leveraging the advantages of the MVC structure, the backend system ensures efficient management of security-related aspects within the City Mall environment and allows for future enhancements and scalability.



4- Getting Started

This section will guide you through the initial steps required to set up and start working with the backend system of city mall. By following these steps, you will be able to quickly get our development environment up and running.

Prerequisites:

Before getting started, make sure you have the following prerequisites in place:

1. Operating System: Windows, macOS, Linux

2. Node.js: Install the latest stable version of Node.js. You can download it from the official Node.js website or use a package manager like `npm` or `yarn`.

3. Database: PostgreSQL. Install and configure the database on our local machine or have access to a remote database instance.

4. Text Editor or IDE: Choose a text editor or an integrated development environment

(IDE) that you are comfortable with. Some popular options include Visual Studio Code
Installation:

To get started with the backend system, follow these steps:

1. Clone the Repository: Clone the backend repository from City Mall (Backend) using Git or download the source code as a ZIP file and extract it to our desired location.

2. Install Dependencies: Open a terminal or command prompt, navigate to the project's root directory, and run the following command to install the required dependencies:

```
npm install
```

3. Start the Backend Server: After the dependencies are installed and the configuration is in place, start the backend server by running the following command:

```
npm start
```

This will start the backend server on the specified port, and you should see a message indicating that the server is running successfully.

5- API Documentation:

Once the backend server is up and running, you can explore the API documentation to understand the available endpoints, request/response formats, authentication methods, and any additional details specific to the backend system. The API documentation can be accessed at City Mall API.

This section provides detailed documentation for the APIs developed for the graduation project in the previous link. The APIs are designed to support various functionalities of the project, including user management, visits, cars, shops, cinemas, offers, movies, issues, and employee management.

Users API

- Get All Users: Retrieves a list of all users in the system.
- Create a User: Creates a new user in the system.
- Get User by ID: Retrieves a user based on our ID.
- Update User by ID: Updates the details of a specific user.
- Delete User by ID: Deletes a user from the system.

Visits API

- Get All Visits: Retrieves a list of all visits recorded in the system.
- Add Visit: Records a new visit in the system.
- Get Visit by ID: Retrieves details of a specific visit.
- Update Visit by ID: Updates the details of a specific visit.
- Delete Visit by ID: Deletes a visit from the system.

Cars API

- Get All Cars: Retrieves a list of all cars registered in the system.
- Create a Car: Registers a new car in the system.
- Get Car by Plate Number: Retrieves details of a car based on its plate number.
- Update Car by Plate Number: Updates the details of a specific car.
- Delete Car by Plate Number: Deletes a car from the system.

Shops API

- Get All Shops: Retrieves a list of all shops in the system.
- Create a Shop: Registers a new shop in the system.
- Get Shop by ID: Retrieves details of a shop based on its ID.
- Update Shop by ID: Updates the details of a specific shop.
- Delete Shop by ID: Deletes a shop from the system.

Cinemas API

- Get All Cinemas: Retrieves a list of all cinemas in the system.
- Create a Cinema: Registers a new cinema in the system.
- Get Cinema by ID: Retrieves details of a cinema based on its ID.
- Update Cinema by ID: Updates the details of a specific cinema.
- Delete Cinema by ID: Deletes a cinema from the system.

Offers API

- Get All Offers: Retrieves a list of all offers in the system.
- Create an Offer: Creates a new offer in the system.
- Get Offer by ID: Retrieves details of an offer based on its ID.
- Update Offer by ID: Updates the details of a specific offer.
- Delete Offer by ID: Deletes an offer from the system.

Movies API

- Get All Movies: Retrieves a list of all movies in the system.
- Create a Movie: Adds a new movie to the system.
- Get Movie by ID: Retrieves details of a movie based on its ID.
- Update Movie by ID: Updates the details of a specific movie.
- Delete Movie by ID: Deletes a movie from the system.

Issues API

- Get All Issues: Retrieves a list of all issues reported in the system.
- Create an Issue: Reports a new issue in the system.
- Get Issue by ID: Retrieves details of a specific issue.
- Update Issue by ID: Updates the details of a specific issue.
- Delete Issue by ID: Deletes an issue from the system.

Employees API

- Employee Login: Authenticates an employee with the system.
- Create Employee Attendance: Records the attendance of an employee.
- Get All Attendance: Retrieves a list of attendance records for all employees.
- Get Employee Attendance: Retrieves attendance details of a specific employee.
- Get Employee by ID: Retrieves details of an employee based on our ID.

6- Database Design

The backend of the security system for the mall utilizes a PostgreSQL database managed using Sequelize. The following tables have been designed to support the functionalities of the system:

5.1. Car

The `cars` table stores information about the cars entering and leaving the mall. It includes columns such as `plateNum` (primary key) and `color`.

5.2. Cinema

The `cinemas` table stores details about the cinemas within the mall. It includes columns such as `id` (primary key), `name`, `location`, `openAt`, `closeAt`, `phone`, and `imageUrl`.

5.3. EmployeeAttendance

The `employeeAttendance` table tracks the attendance of the mall employees. It includes columns such as `id` (primary key), `loggedIn`, and `loggedOut`.

5.4. Employee

The `employees` table stores information about the mall employees. It includes columns such as `id` (primary key), `name`, `email`, `password`, `imageUrl`, and `role`.

5.5. Issue

The `issues` table records the issues reported within the mall. It includes columns such as `id` (primary key), `type`, `details`, `imageUrl`, and `state`.

5.6. ModelIssue

The `modelIssues` table stores information about the model issues reported within the mall. It includes columns such as `id` (primary key), `type`, `details`, `imageUrl`, and `state`.

5.7. Movie

The `movies` table stores information about the movies being shown in the mall's cinema. It includes columns such as `id` (primary key), `name`, `duration`, `release`, `description`, `genre`, `time`, `ticketPrice`, and `imageUrl`.

5.8. Offer

The `offers` table stores details about the offers available at various shops. It includes columns such as `id` (primary key), `discount`, `startAt`, and `endAt`.

5.9. Shop

The `shops` table stores information about the shops within the mall. It includes columns such as `id` (primary key), `name`, `location`, `openAt`, `closeAt`, `phone`, `imageUrl`, and `shopType`.

5.10. User

The `users` table stores information about the system users, including customers. It includes columns such as `id` (primary key), `name`, `phone`, `email`, `password`, `passwordConfirm`, `passwordChangedAt`, `passwordResetCode`, and `passwordResetExpire`.

5.11. Visit

The `visits` table tracks the visits of cars to the mall. It includes columns such as `id` (primary key), `timeIn`, `timeOut`, `section`, and `cost`.

5.12. Checkout

The `checkouts` table represents the checkouts made by users for movies. It includes columns such as `id` (primary key), `status`, `ticketPrice`, `ticketNum`, and `cost`.

5.13. CinemaMovie

The `cinema_movie` table represents the relationship between cinemas and movies, indicating which movies are being shown in each cinema. It does not include any additional columns.

5.14. IssueEmployee

The `issue_employee` table represents the relationship between issues and employees, indicating which employees are assigned to each issue. It does not include any additional columns.

5.15. ModelIssueEmployee

The `modelIssue_employee` table represents the relationship between model issues and employees, indicating which employees are assigned to each model issue. It does not include any additional columns.

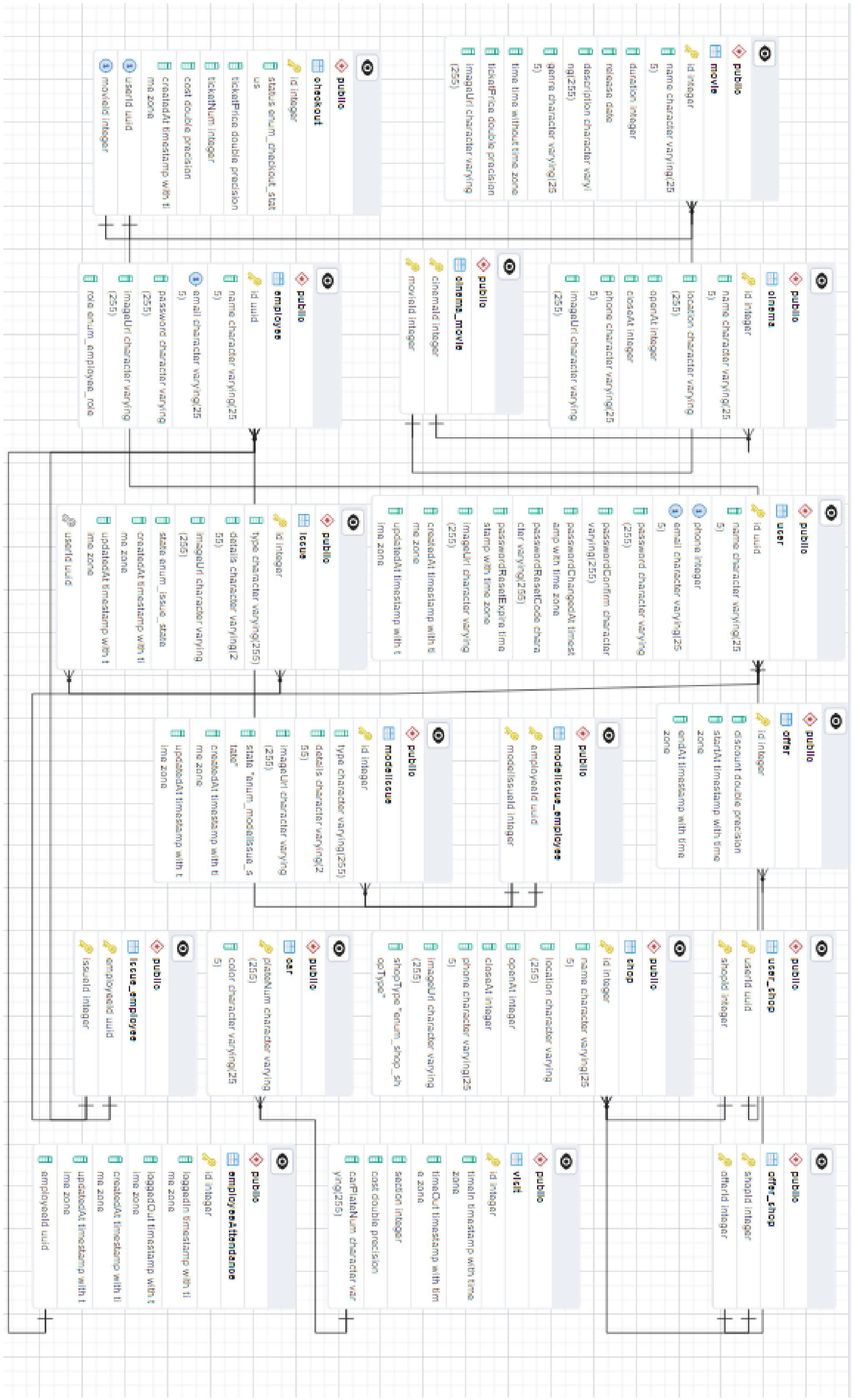
5.16. OfferShop

The `offer_shop` table represents the relationship between offers and shops, indicating which shops have specific offers. It does not include any additional columns.

5.17. UserShop

The `user_shop` table represents the relationship between users and shops, indicating which shops are visited by each user. It does not include any additional columns.

The ERD for the Database



public
movie
id integer
name character varying(255)
duration integer
release date
description character varying(255)
genre character varying(255)
time time without time zone
ticketPrice double precision
imageUrl character varying(255)
public
checkout
id integer
status enum_checkout_status
ticketPrice double precision
ticketNum integer
cost double precision
createdAt timestamp with time zone
userId uuid
movieId integer
public
cinema
id integer
name character varying(255)
location character varying(255)
openAt integer
closeAt integer
phone character varying(255)
imageUrl character varying(255)
public
cinema_movie
cinemaId integer
movieId integer
public
employee
id uuid
name character varying(255)
email character varying(255)
password character varying(255)
imageUrl character varying(255)
role enum_employee_role
public
user
id uuid
name character varying(255)
phone integer
email character varying(255)
password character varying(255)
passwordConfirm character varying(255)
passwordChangedAt timestamp with time zone
passwordResetCode character varying(255)
passwordResetExpire timestamp with time zone
imageUrl character varying(255)
createdAt timestamp with time zone
updatedAt timestamp with time zone
public
issue
id integer
type character varying(255)
details character varying(255)
imageUrl character varying(255)
state enum_issue_state
createdAt timestamp with time zone
updatedAt timestamp with time zone
userId uuid
public
offer
id integer
discount double precision
startAt timestamp with time zone
endAt timestamp with time zone
public
modelIssue_employee
employeeId uuid
modelIssueId integer
public
modelIssue
id integer
type character varying(255)
details character varying(255)
imageUrl character varying(255)
state "enum_modelIssue_state"
createdAt timestamp with time zone
updatedAt timestamp with time zone
public
user_shop
userId uuid
shopId integer
public
shop
id integer
name character varying(255)
location character varying(255)
openAt integer
closeAt integer
phone character varying(255)
imageUrl character varying(255)
shopType "enum_shop_shopType"
public
car
platNum character varying(255)
color character varying(255)
public
issue_employee
employeeId uuid
issueId integer
public
offer_shop
shopId integer
offerId integer
public
visit
id integer
timeIn timestamp with time zone
timeOut timestamp with time zone
section integer
cost double precision
carPlateNum character varying(255)
public
employeeAttendance
id integer
loggedIn timestamp with time zone
loggedOut timestamp with time zone
createdAt timestamp with time zone
updatedAt timestamp with time zone
employeeId uuid

7- Security

Ensuring the security of our project is essential to protect sensitive data, prevent unauthorized access, and mitigate potential security vulnerabilities. In this section, we will discuss the security measures implemented in the backend system, including authentication and authorization, input validation, and security best practices.

1) **Authentication and Authorization**

Authentication is the process of verifying the identity of a user or entity accessing the system. It ensures that only authenticated users can access protected resources. In our graduation project, authentication can be implemented using a combination of user credentials and tokens.

User Credentials

User credentials, such as username (or email) and password, are used to verify the identity of users during the login process. These credentials are securely stored in the `users` table of our database. When a user attempts to log in, the system checks if the provided credentials match the stored values.

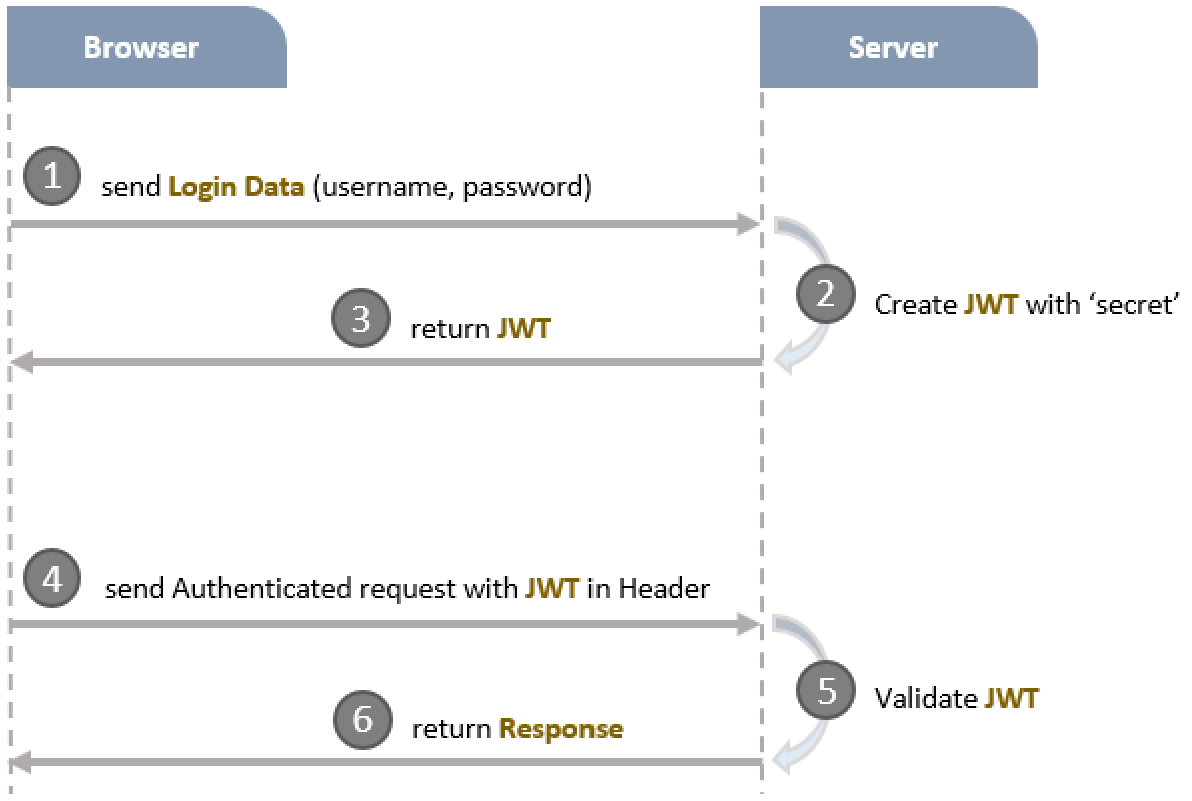
To enhance security, it is recommended to use secure password storage techniques such as hashing and salting. Hashing transforms the user's password into a fixed-length string of characters, making it computationally difficult to reverse-engineer the original password. Salting adds a random value to each password before hashing, further increasing security.

Token-based Authentication

Once a user's credentials are successfully verified, the system generates an authentication token, such as a JSON Web Token (JWT). This token contains information about the user's authentication status and other relevant details. The token is digitally signed by the server to ensure its integrity and authenticity.

The generated token is sent back to the user as part of the authentication response. Subsequently, the user includes this token in the headers (e.g., Authorization header) of our requests to access protected resources within the system.

Flow for Signup & Login with JWT Authentication
The diagram shows flow of User Registration, User Login and Authorization process.


Browser
Server
1 send Login Data (username, password)
2 Create JWT with 'secret'
3 return JWT
4 send Authenticated request with JWT in Header
5 Validate JWT
6 return Response

A legal JWT must be added to HTTP x-access-token Header if Client accesses protected resources.

Authorization

Authorization determines what actions an authenticated user is allowed to perform within the system. It ensures that users have the necessary permissions to perform specific operations. In our graduation project, authorization can be based on user roles and our associated permissions.

User Roles and Permissions
User roles define different levels of access and functionality within the system. The `employees` table in our database includes a `role` column, which specifies the role of each employee. Common roles could include 'admin', 'manager', 'employee', or 'customer'.

Each role is associated with a set of permissions that determine what actions a user with that role can perform. For example, an 'admin' role might have full access to all functionalities, while an 'employee' role might have restricted access.

Authorization Service

The system's authorization service receives requests from authenticated users and validates our authentication tokens. It then checks the user's role and associated permissions to determine if the requested action or resource access is allowed.

Based on the user's role, the authorization service consults a set of predefined rules or policies to determine access rights. These rules can be stored in a separate configuration file or database table.

2) Node.js Express Architecture with Authentication & Authorization

The architecture of a Node.js Express application that incorporates authentication and authorization. Node.js and Express are popular choices for building web applications due to our simplicity and scalability. Adding authentication and authorization features enhances the security and control over the application's resources.

Overview of the Architecture

The architecture of a Node.js Express application with authentication and authorization typically involves the following components:

1. **Client:** The client refers to the user interface or frontend of the application, which can be a web browser, mobile app, or any other client application.
2. **Server:** The server is responsible for handling incoming requests from clients, performing authentication and authorization checks, and serving appropriate responses.
3. **Express Framework:** Express is a minimal and flexible web application framework for Node.js. It provides a set of features and utilities to build robust and scalable web applications.
4. **Authentication Middleware:** Authentication middleware is used to verify the identity of the user making the request. It validates user credentials, such as username and password, against a user database or external authentication providers.
5. **Authorization Middleware:** Authorization middleware determines whether the authenticated user has the necessary privileges to access a particular resource or perform a specific action. It checks user roles, permissions, or any custom authorization logic.

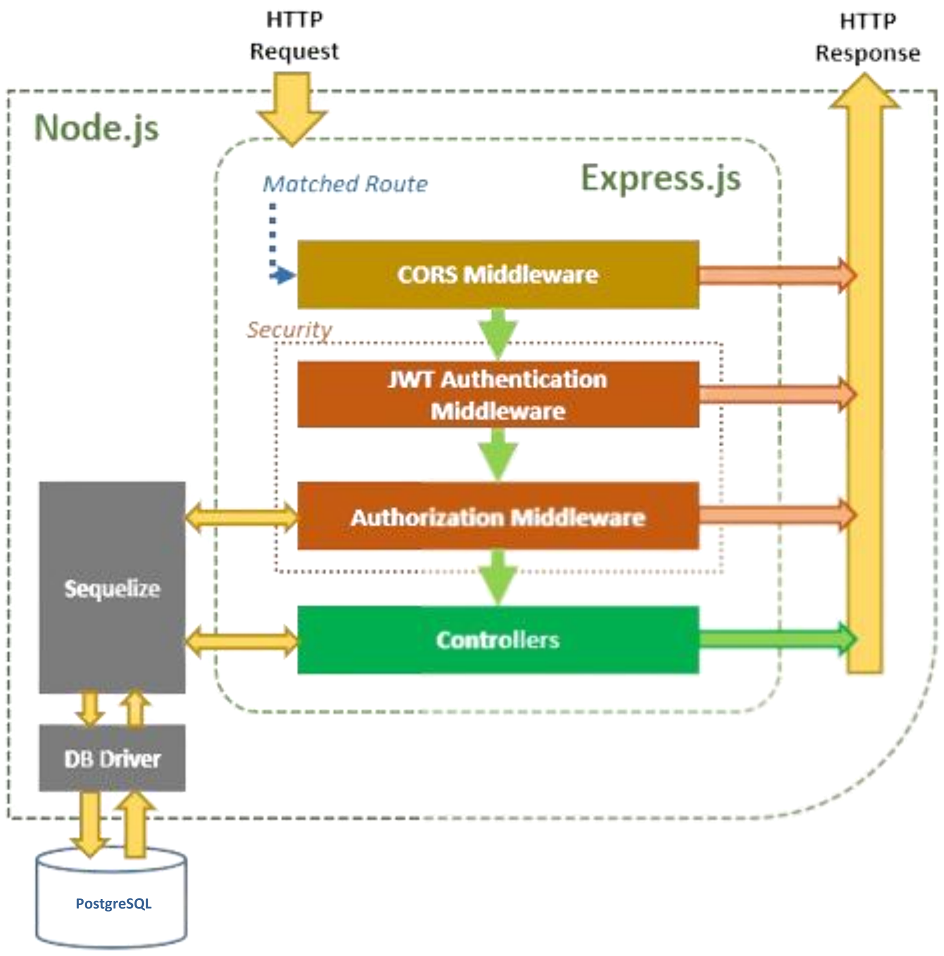
6. **User Database:** A user database stores user information, such as usernames, passwords (hashed), roles, and permissions. Common choices for user databases include relational databases like MySQL or PostgreSQL, NoSQL databases like MongoDB, or even external identity providers like OAuth or LDAP.

Flow of Authentication & Authorization

The flow of authentication and authorization in a Node.js Express application typically follows these steps:

1. **Client Requests Access:** The client initiates a request to the server to access a protected resource or perform a specific action.
2. **Authentication Middleware:** The server's authentication middleware intercepts the request and verifies the user's identity. It checks the provided credentials against the user database or external authentication providers.
3. **Authentication Success:** If the authentication is successful, the server generates a token (such as a JSON Web Token or JWT) and includes it in the response to the client.
4. **Client Includes Token:** The client receives the token from the server and includes it in subsequent requests as an Authorization header or another appropriate mechanism.
5. **Authorization Middleware:** For each subsequent request, the server's authorization middleware intercepts the request, extracts the token, and verifies its validity.
6. **Authorization Check:** The server performs authorization checks based on the user's role, permissions, or any custom authorization logic. It determines whether the user is allowed to access the requested resource or perform the requested action.
7. **Response to Client:** If the user is authorized, the server proceeds with the requested operation and sends the appropriate response back to the client. Otherwise, an error response or access denied message is returned.

Here's a diagram illustrating the flow of authentication and authorization in a Node.js Express application:



The diagram provides a visual representation of the steps involved in authenticating and authorizing users in a Node.js Express application.

3) Input Validation

Input validation is a critical security measure that helps prevent various types of attacks, such as injection attacks and cross-site scripting (XSS). The backend system implements strong input validation techniques to validate and sanitize user input.
Validation: User input is thoroughly validated against predefined rules and constraints to ensure its integrity and adherence to expected formats. Data types, length limits, format patterns, and business-specific validation rules are validated to ensure that the input is valid and secure. Proper validation prevents the system from processing malicious or malformed input that could lead to security vulnerabilities.
Sanitization: Input sanitization is performed to remove potentially harmful or malicious content from user input. The backend system employs techniques to sanitize user input, such as encoding special characters or scripts that could be used for attacks. This prevents the execution of malicious code and protects against common security vulnerabilities.

By incorporating these security measures and following best practices, our backend system will be more resilient against security threats, protecting the confidentiality, integrity, and availability of our data and ensuring a secure user experience

8- Deployment and Environment Management

The deployment and environment management process plays a crucial role in ensuring the successful deployment and operation of our backend system. In this section, we will discuss the deployment strategy used and the environment management practices implemented for our project, with a specific focus on the deployment on Render.

Deployment Strategy

For our project, the deployment strategy involves leveraging the Render platform. Render is a cloud provider that simplifies the deployment and management of web applications, providing scalability, reliability, and ease of use.

The deployment process typically involves the following steps:

1. Configuration: Set up the necessary configuration files, including the project dependencies, environment variables, and deployment settings. This ensures that the deployed application has all the required resources and configurations to function correctly.

2. Build Process: Prepare the project for deployment by building and packaging the application. This may involve compiling code, bundling assets, and generating necessary artifacts for deployment.

3. Deployment: Utilize Render's platform to deploy our backend system. Render supports various deployment methods, such as Docker containers, static site hosting, and serverless functions. The deployment process on Render is simplified, allowing us to focus on the application logic rather than infrastructure management.

4. Scaling and Load Balancing: Render offers automatic scaling and load balancing capabilities, ensuring that our backend system can handle increasing traffic and scale horizontally as needed. This ensures optimal performance and reliability, even during peak usage periods.

Environment Management

Efficient environment management is essential for maintaining consistency, security, and stability across different environments in our project's lifecycle. Here are some key practices employed for environment management:

1. Environment Variables: Utilize environment variables to store sensitive configuration values, such as API keys, database credentials, and other secrets. Render provides a secure and easy-to-use interface for managing environment variables, ensuring that sensitive information remains protected.

2. Separation of Environments: Maintain separate environments for development, staging, and production. This segregation allows for thorough testing and validation of changes before deploying them to the live production environment. Each environment should closely resemble the production environment to identify any potential issues early on.

3. Version Control and Deployment Pipelines: Utilize version control systems, such as Git, to manage our project's source code.

4. Monitoring and Error Reporting: Implement monitoring and error reporting mechanisms to track the health and performance of our deployed application. Utilize tools such as logging frameworks, error tracking services, and performance monitoring solutions to gain insights into application behavior and identify and resolve issues promptly.

By leveraging the Render platform for deployment and adhering to robust environment management practices, you can ensure smooth and reliable deployment of our backend system. Render's features and infrastructure capabilities provide scalability, security, and ease of management, enabling us to focus on building and improving our application while leaving the infrastructure complexities to the platform.

9- Conclusion (Backend):

In conclusion, the backend system that is presents a robust and secure foundation for the City Mall security system. This section provided a comprehensive overview of the backend architecture and its key components.
The backend system, built using Node.js and Express.js, offers a reliable and efficient platform for handling data storage, processing, and communication between the frontend and external services. It incorporates essential features such as a powerful API layer, secure authentication and authorization mechanisms, seamless integration with PostgreSQL for data storage, and compatibility with various third-party services.

The API documentation and reference guide included in this documentation provide developers and technical stakeholders with a clear understanding of the system's endpoints, data structures, and the expected behavior of each API route. It serves as a valuable resource for future development, maintenance, and expansion of the City Mall security system.

With a focus on security, the backend system ensures the protection of sensitive data and the privacy of users. Authentication and authorization are implemented using JSON Web Tokens (JWT), with user credentials securely stored through hashing and salting techniques. Additional security measures, such as input validation and adherence to best practices, contribute to the overall robustness of the system.

The database design, managed using Sequelize, efficiently stores and retrieves data related to users, visits, cars, shops, cinemas, offers, movies, issues, and employees. The defined relationships between these entities facilitate data manipulation and retrieval, ensuring optimal performance and scalability.

By successfully developing the backend system for the City Mall security project, the team has demonstrated our expertise in backend development, database management, and security implementation. The documentation provided offers a comprehensive guide for understanding and utilizing the backend components, enabling future enhancements and integrations.

Overall, the backend system developed for the City Mall security project showcases the team's technical competence, problem-solving abilities, and dedication to creating a secure and efficient application. It lays the foundation for a comprehensive security system that effectively manages user data, visits, cars, shops, cinemas, offers, movies, issues, and employee attendance within the City Mall environment.