Railway Reservation (CS301)

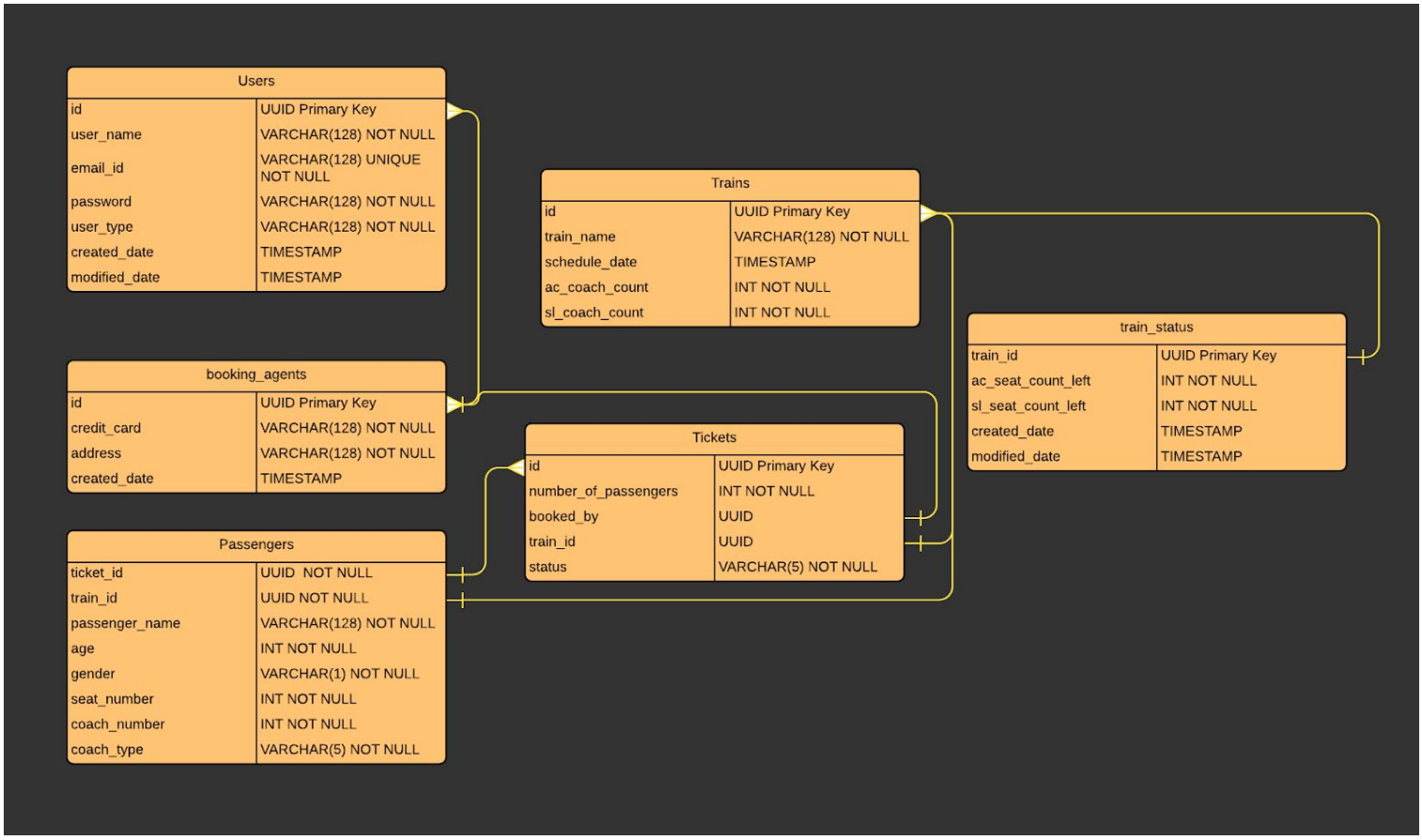
Praful Gupta (2018CSB1112) **Rohit Tuli (2018CSB1116)**

Technology Stack -

Front-end - React.js Language - Javascript
Back-end - Node.js, Express, PgSQL Language - Javascript, SQL
Hosting Service - Heroku
API Testing Toolkit - Postman
Current Website Endpoint - https://railway-reservation-project.herokuapp.com/
Postman Collection - https://www.getpostman.com/collections/03f24eedbb204816e5cf

Live Database Connection Parameters-
PG_USER=obuezweniacpnp
PG_PASSWORD=38b309d186d16f5b8bbfda30a00faa4371851fee0ebd640b050ee9abb2556da
d
PG_HOST=ec2-35-168-77-215.compute-1.amazonaws.com
PG_PORT=5432
PG_DATABASE=d1l8aafsi20c9g

Schema Diagram -



Schema Database Code -

```
CREATE TABLE IF NOT EXISTS
    users(
      id UUID PRIMARY KEY,
      email VARCHAR(128) UNIQUE NOT NULL,
      password VARCHAR(128) NOT NULL,
                user_type VARCHAR(128) NOT NULL,
      created_date TIMESTAMP,
      modified_date TIMESTAMP
    );



CREATE TABLE IF NOT EXISTS
      trains(
        id UUID PRIMARY KEY,
        train_name VARCHAR(128) UNIQUE NOT NULL,
        ac_coach_count INT NOT NULL,
        sl_coach_count INT NOT NULL,
        schedule_date TIMESTAMP,
                create_date TIMESTAMP
      );



CREATE TABLE IF NOT EXISTS
      booking_agents(
        id UUID PRIMARY KEY,
        credit_card VARCHAR(128) NOT NULL,
        address VARCHAR(128) NOT NULL,
                created_date TIMESTAMP,
                FOREIGN KEY (id) REFERENCES users(id)
      );
```

```
CREATE TABLE IF NOT EXISTS
	tickets(
	  id UUID PRIMARY KEY,
	  number_of_passengers INT NOT NULL,
	  booked_by UUID,
		train_id UUID,
		status VARCHAR(5) NOT NULL,
		created_date TIMESTAMP,
		FOREIGN KEY (train_id) REFERENCES trains(id),
		FOREIGN KEY (booked_by) REFERENCES booking_agents(id)
	);

CREATE TABLE IF NOT EXISTS
	passengers(
		id UUID PRIMARY KEY,
	  ticket_id UUID NOT NULL,
	  train_id UUID NOT NULL,
	  passenger_name VARCHAR(128) NOT NULL,
		age INT NOT NULL,
		gender VARCHAR(1) NOT NULL,
		seat_number INT NOT NULL,
		coach_number INT NOT NULL,
		coach_type VARCHAR(5) NOT NULL,
		FOREIGN KEY (ticket_id) REFERENCES tickets(id),
		FOREIGN KEY (train_id) REFERENCES trains(id)
	);

CREATE TABLE IF NOT EXISTS
	train_status(
		train_id UUID PRIMARY KEY,
	  ac_seat_count_left INT NOT NULL,
		sl_seat_count_left INT NOT NULL,
	  created_date TIMESTAMP,
	  modified_date TIMESTAMP,
		FOREIGN KEY (train_id) REFERENCES trains(id) ON DELETE
CASCADE
	);
```

Node Js Security Implementations -

1. Password in users table is stored after hashing using bcrypt library with 8 levels of salting rounds.
2. JWT token is implemented which generated a jwt token encrypted by a secret hashing function and contains details about the user's id and expires in 7 days.
3. Admin and agents have to send their jwt token as a parameter from the frontend in the header of every API call and a middleware verifies it and proceeds accordingly.

Node.js APIs -

1. **app.post('/api/v1/users/login',UserWithDb.login);**
   Receives email and passwords as parameters and verifies from the users table in the database and returns a jwt encrypted token for further use of other APIs.

2. **app.post('/api/v1/users/create', UserWithDb.create);**
   Creates a new booking agent by taking all information about booking agents like credit card number, email, password, name, address, and stores in user + booking_agents tables respectively.

3. **app.post('/api/v1/admin/create_train', Auth.verifyToken, AdminWithDb.createTrain);**
   Creates a new train only after getting all the details about a train.

4. **app.get('/api/v1/get_all_train', Auth.verifyToken, AdminWithDb.getAllTrains);**
   Returns all available future running trains;

5. **app.get('/api/v1/users/get_all_my_tickets', Auth.verifyToken, Ticket.getAllTickets);**
   Return all the ticket booked by that specific booking_agent

6. **app.post('/api/v1/users/create_ticket', Auth.verifyToken, Ticket.createTicket);**
   Creates a ticket for a specific train with unique PNR(id) and assigns passengers their seat numbers of booking is successful else sets ticket status as failed.

7. **app.get('/api/v1/users/get_all_passenger_by_ticket', Auth.verifyToken, Ticket.getAllPassengerByTicket);**

Return the list of all passengers on a specific ticket.

8. **app.get('/api/v1/users/get_all_passenger_by_train', Auth.verifyToken, Ticket.getAllPassengerByTrain);**
   Return the list of all passengers in a specific train to generate pnr list of train

Train Release Procedure -

Authorization - JWT token specific to admin only
Input - train_name , ac_coach_count , sl_coach_count , schedule_date
Logic -

```
[
     uuid(),  // unique id for train
     req.body.train_name,
     req.body.ac_coach_count,
     req.body.sl_coach_count,
     req.body.schedule_date,
     moment(new Date())  // current timestamp
];
```

Adds these values to the trains table.
After that, instead of using a trigger, we are using the server code to

```
[
     trainID,   // current train id
     req.body.ac_coach_count * acBerthCount,
     req.body.sl_coach_count * slBerthCount,
     moment(new Date()),   // current date
     moment(new Date())    // modified date
];
```

Adds these values to the train_status table where acBerthCount and slBerthCount is the total number of seats in an ac or sleeper coach.

Hence a train is added to the database from admin and can be shown from get_all_trains API.

Ticket Booking Procedure -

Authorization - JWT token specific to a booking agent
Input - train_id, number_of_passengers, coach_type ,passenger[list {name,age,gender}]
Logic -

1. Checks the train status, if possible to book in that specific coach for that train proceeds to step 2 else sets that ticket status to Failed and returns.
2. Updates ticket table with Sucess and inserts these parameters into the tickets table (id(Pnr unique uuid), number_of_passengers, agents_id, train_id, Success, created_date) and goes to step 3.
3. Adds all the passenger's details to the passenger table with parameters (id, ticket_id, train_id, name, age, gender, seat_number, coach_number, coach_type) and go to step 4.
4. Updates the train_status of that train by decreasing the count of available seats from that train.

Seat number and coach number are generated sequentially depending upon the number of available seats and berth count in a specific coach.

Trigger and Stored Procedures - Triggers are implemented as server code in all the APIs as described above and no stored procedures are used as we are using Javascript in the server (Node.js) and its libraries to implement everything like id generation, output parsing, or Query modifications accordingly.