Design Document for ServiceHUB

Group TA 122

Leha Dutta: Frontend (Description/working of the app, Views:Splash Screen,Create User Account, Login,Home Service,Settings(User Profile),Checkout & Payment) , Block Diagram (Frontend: Views, GUI,Legend) , API doc generation

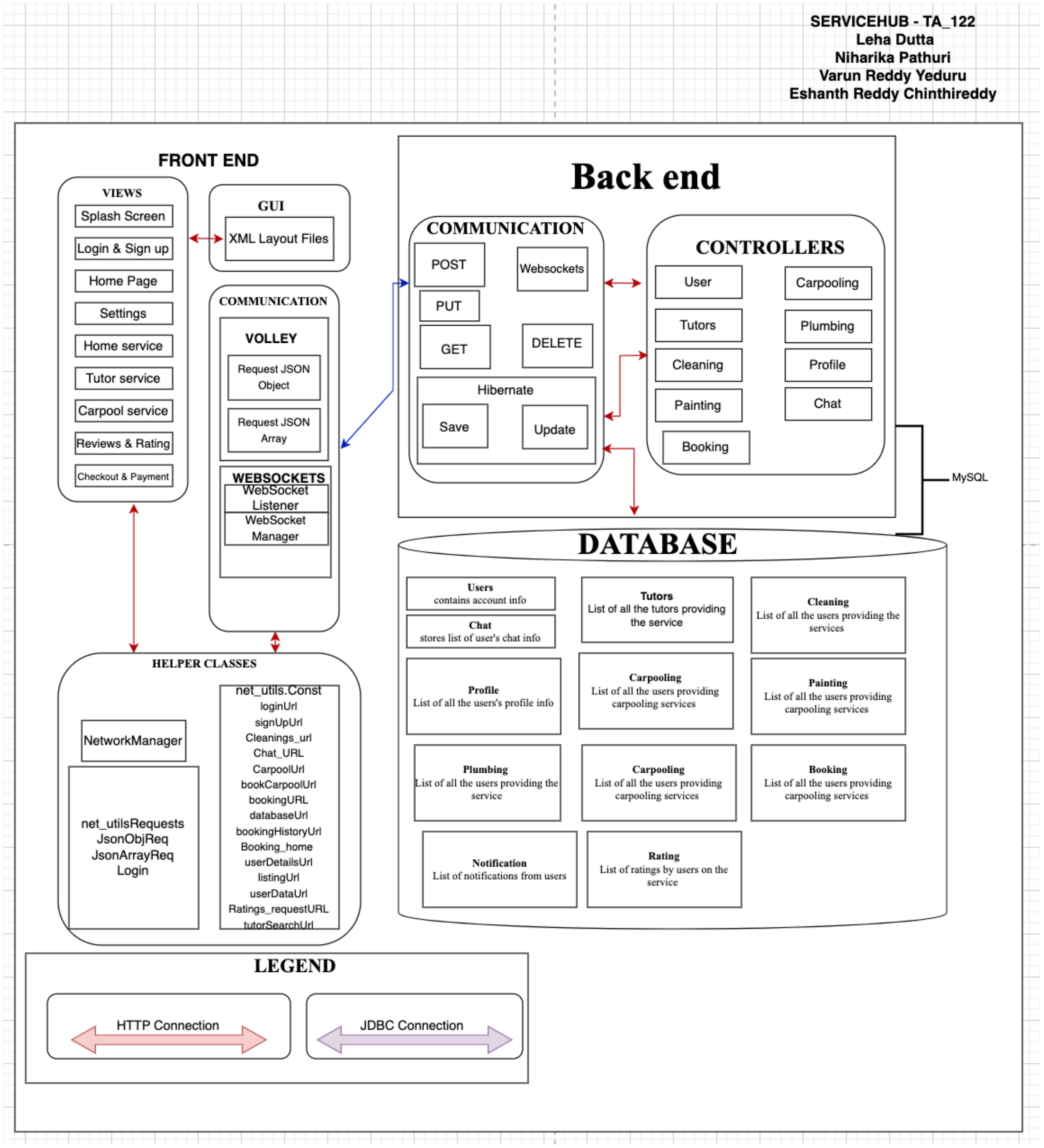
Niharika Pathuri: Block Diagram (Backend and Database), Relationship Diagram, Backend Design Descriptions(Users, Profile, Login, Painting, Plumbing, Cleaning, Message, Reservation), Relationship Diagram, API Docs

Varun Reddy Yeduru: Frontend (Education services, Carpool Services, Home Page,Types of users: Service Provider, Service Seeker, admin), Block Diagram (Helper Classes, Communication), API doc generation

Eshanth Reddy Chinthireddy: Block Diagram, (Backend and Database), Relationship Diagram, Backend Design Descriptions(Users, Tutors, Carpooling, Profile, Message, Booking), API Docs(Tester Swagger.IO)

Block Diagram

PUT THE BLOCK DIAGRAM PICTURE ON THIS PAGE! (Create the picture using pencil or drawIO)



Design Description

FRONTEND:

DESCRIPTION/WORKING OF THE APP:

- When the user starts the app they are presented with the splash screen , which has sign up and login buttons which the user can click on to navigate further in the app
    - If the user already has an account they can click on the login button and use the app
    - If the user is a new user, they will have to click on the sign up button , create an account and then navigate back to the splash screen and login to the app
- Once the user has successfully logged into the app, the user will be navigated to the home page which includes all the services and a search bar.
- Once the user is on the home page, they can use the navigation bar at the bottom to navigate to the different services the user needs.
    - The navigation bar includes home, tutor,carpool,home and settings.
- Once the user opts for any of the services they will be navigated to the service the user has opted for. For instance, if the user opts for home service, they will be navigated to the home service pages which includes all the home services the app is providing.
    - Once the user clicks on the specific service under the main service (for example, cleaning under home service) , they will be redirected to the specific service the user opted for (i.e, cleaning)
    - After the user has selected the specific service, they have an option to either list the service or book the service
        - If the user has opted to list the service, they will be asked to fill out information like name, price and a brief description.
    - When the user is in the specific service they have the option to read a brief description of the service, see a list of people providing the service, time slots available, price rate and reviews/rating.
    - After the user has selected the provider they want the service from they will be able to book the service based on the availability of the service and once they book, they

    will be redirected to the confirmation page which will include the service details, total price , option to apply a coupon code.
  - Once the user has reviewed the details under the confirmation page, they will be redirected to the payment page on clicking on the pay now button
  - Once the user clicks on the pay now button , they are redirected to the payment page where they can choose the payment method and process the payment.

VIEWS:

SPLASH SCREEN

- When the app starts, the user will be directed to a splash screen, which is a welcome page where the user will have the option to choose from two options: login or sign up

CREATE USER ACCOUNT (USERS)

- Users that are new to the app will have to create the account to use the app
- Create account will navigate us to the signup page with the following page with elements:
  - EditText: Name
  - EditText: Email Address
  - EditText: Password
  - EditText: Confirm Password
  - Button: Create Account
- The sign up page also includes features like:
  - Remember me
  - Forgot Password?
  - Sign up using Google, Facebook, Phone number
- Once the user has created an account, the values the user entered for the name, email address and password will be stored in the Users database by sending a POST request to the server

LOGIN (USERS)

- Once the user has created the account or if a user already has an account can input their credentials - email address and password to login to the app

  - If the credentials entered by the user are correct , the user will be navigated to the home page
  - If the credentials entered by the user are incorrect, the user will be notified using toast that will display Login Failed or Wrong credentials
- Once the user logs in the data, a GET request is sent to the USERS database to fetch the data and this matches the data the user entered from the data that is stored in the database.

SETTINGS (USERS)

- Once the user successfully logs into the app, the user will be navigated to the home page; the user has the option to navigate to the settings page and the details will be displayed in the settings page under the user profile.
- The user can use the settings page to update features of the app such as password as well as view other features of the app.
- The settings profile will also include the booking history of the user

HOME SERVICE (HOME)

- Once the user logs in to the app, the user can navigate to the home service page and choose from multiple services the app provides.
- The user will have the option to book or list the service
- Specific services under the home service will have a slot booking feature and an option to chat with the service provider
- The user will also be able to view the booking history of all the home services the user has booked

CHECKOUT & PAYMENT

- Once the user logs in to the app, the user can navigate to the service they want and when they book a service, they will be navigated to the checkout page which will include all the services they have added to the cart.
- The user will also have the ability to apply any coupon code they might have and the checkout page will also include the total price.

- Once the user has verified everything in the checkout page, they will be navigated to the payment page by clicking on the pay now button in the checkout page, which will lead them to a page where they will have multiple methods of payment they can choose from.
  Note: Our app will not include an official payment method but we will be showing the different payment methods the app would provide, the user will just not be able to have a successful payment.
  Signup Process:

  When a user first opens the app, they are presented with a welcoming splash screen, offering two primary options: "Login" or "Sign Up."
  To create an account, users select "Sign Up." This process allows them to access the full range of services offered by the app.

<u>USER ROLES</u>

SERVICE PROVIDERS:

- For users who intend to offer services, such as home services or carpooling, they can select the "Service Provider" option during the signup process.
- The signup page includes fields where service providers can enter their name, email address, password, and confirm their password.
- Once registered, their profile is created as a service provider, enabling them to list their services, set pricing, and interact with potential customers.

SERVICE SEEKERS:

- Users who are primarily seeking services, like booking home services or finding a tutor, can opt for the "Seeker" role during signup.
- The signup form for seekers includes fields for their name, email address, password, and password confirmation.

- Upon successful signup, seekers can browse services, make bookings, and utilize various features tailored to their needs.

ADMIN:

- The app includes an "Admin" role, which is reserved for administrators who have special privileges and responsibilities.
- Administrators can directly log in from the login page by selecting the "Admin Login" option.
- Admin accounts are typically created separately and are not part of the regular signup process.
- Admins have access to features that allow them to manage and oversee the app's operations, including user management, service listings, and other administrative tasks.

LOGIN PROCESS:

- After signing up and creating an account, users can easily log in by selecting the "Login" option on the splash screen.
- To log in, users need to provide their registered email address and password, ensuring secure access to their accounts.
- Once logged in, users are directed to the home page, where they can explore the various services and features based on their chosen role—service provider, seeker, or admin.

HOME PAGE

- When a user logins, they immediately notice the home page, which is thoughtfully divided into three distinct sections. The first section, labeled "Home Service," is their primary destination for discovering various home-related services.

- Clicking on the "Home Service" section, users are presented with a list of services such as cleaning, repairs, and more. Each service listing includes a brief description, pricing, and the option to either list a service or book a service.
- If a user wants to offer a home service, they can easily list it by providing details like the service name, price, and a brief description, enabling them to connect with potential customers.
- When a user needs a specific home service, they can explore the listings, select a service, and book it based on their preferred time slot.

EDUCATION SERVICE

- The second section, labeled "Education," on the home page provides users with a valuable resource for their academic needs. It's where they can access educational services like tutoring.
- Clicking on "Education" takes users to a page where they can browse through a list of tutors. Each tutor listing provides essential information, including qualifications, subjects of expertise, names, and brief descriptions.
- Users can select a tutor that aligns with their academic requirements, making personalized academic support easily accessible.

CARPOOL SERVICE

- Users can access the Carpool Service section from the home page, where they can either list their available carpool service or search for carpooling options.
- Listing a carpool service: Users who want to offer carpooling can create a listing by providing details such as the starting location, destination, date, and time. This listing is visible to other users looking for carpool services.
- Searching for carpool options: Users in need of a ride can browse through the available listings based on their preferred starting and ending locations.
- Each carpool service listing provides crucial information, including the departure point, destination, date, time, and the driver's average rating. To book a carpool service, users select a specific listing that aligns with their travel needs. They can choose a suitable time slot based on their needs. The app maintains a booking history feature, allowing users to

revisit their previous carpool bookings. This feature provides comprehensive information about each booking, including details about the driver, date, and time. After completing a carpool service, the drivers have the opportunity to rate and review the service. This rating and review system fosters trust and accountability within the carpooling community. Users can assign a rating and provide feedback about their carpool experience, which can be accessed by others considering the same service provider.

FRONTEND API

Leha:

- BookingHistory
- BookingService
- BookingSlotActivity
- ChatActivity
- FetchServices
- HomeCleaning
- HomeService
- ListingService
- LoginActivity
- LoginSuccess
- MainActivity
- SignUpActivity
- UpdatePasswordActivity
- UserActivity
- WebsocketListener
- WebsocketManager

Varun:

- booking_history
- carpool_home
- carpool_search
- CarpoolActivity
- CarpoolAdapter
- CarpoolListing
- first_page

- MapActivity
- MainActivity
- maps_first
- NetworkManager
- Ratings
- Ratings_review
- trash_check
- Tutor
- tutor_become
- TutorAdapter
- TutorSearchActivity
- TutorAdapter

BACKEND

COMMUNICATION:

This is the main component responsible for handling communication with the database.

- **POST**: This is used to send details about something that needs to be added to our database.
- **GET**: This is used to ask for information, usually with a specific code or name for the exact thing you want to know about.
- **PUT**: This is used to send details to update a specific thing in the database.
- **DELETE**: This is used to send a code or name to remove a specific thing from the database.

CONTROLLERS:

These are like instruction books that show how the front part of our system talks to the database.

Niharika

ReservationController

Class Description:

- The ReservationController class is a RESTful controller in the OneToOne application responsible for handling HTTP requests related to reservations. It is annotated with @RestController and is part of the Swagger2 documentation with the @Api annotation. The controller provides various methods for managing reservations, including retrieving all reservations, getting a reservation by ID, creating, updating, and deleting reservations.

Fields:

- ReservationRepository: An instance of the ReservationRepository class, allowing access to reservation data.

Methods:

- getAllBookings: Retrieves a list of all reservations.
- getBookingById: Retrieves a specific reservation by its unique ID.

- getBookingsByType: Retrieves reservations based on service type.
- createBooking: Creates a new reservation and returns a success message or an error message if any fields are missing or if there's a time slot conflict.
- getUserForBooking: Retrieves the user associated with a specific reservation.
- updateBooking: Updates an existing reservation based on the provided ID.
- deleteBooking: Deletes a reservation based on its unique ID.

Swagger Documentation:

- The @Api annotation indicates that this controller is part of the Swagger2 documentation, providing additional information about the REST APIs related to reservations.

MessageController

Class Description:

- The MessageController class is a RESTful controller in the OneToOne application responsible for managing HTTP requests related to messages. It is annotated with @RestController and includes Swagger2 annotations (@Api, @ApiOperation, @ApiResponses) for documenting the REST APIs related to messages.

Fields:

- messageRepository: An instance of the MessageRepository class, facilitating access to message data.

Methods:

- markMessageAsSeen: Marks a message as seen based on the provided message ID. Documented with Swagger2 annotations for API documentation.
- saveMessage: Saves a message, setting its bold property based on frontend information. Returns a ResponseEntity indicating success or an internal server error. Documented with Swagger2 annotations.

PaintingController

Class Description:

- The PaintingController class is a RESTful controller in the OneToOne application responsible for managing HTTP requests related to painting services. It is annotated with @RestController and includes Swagger2 annotations (@Api, @ApiOperation, @ApiResponses) for documenting the REST APIs related to painting services.

Fields:

- paintingRepository: An instance of the PaintingRepository class, allowing access to painting data.
- success: A success message string used for responses.
- failure: A failure message string used for responses.

Methods:

- getAllPaintings: Retrieves a list of all painting services. Documented with Swagger2 annotations for API documentation.
- getPaintingById: Retrieves a specific painting service by its unique ID. Documented with Swagger2 annotations.
- createPainting: Creates a new painting service and returns a success or failure message. Documented with Swagger2 annotations.
- updatePainting: Updates an existing painting service based on the provided ID. Documented with Swagger2 annotations.
- deletePainting: Deletes a painting service based on its unique ID. Documented with Swagger2 annotations.

PlumbingController

Class Description:

- The PlumbingController class is a RESTful controller in the OneToOne application responsible for managing HTTP requests related to plumbing services. It is annotated with @RestController and includes Swagger2 annotations (@Api, @ApiOperation, @ApiResponses) for documenting the REST APIs related to plumbing services.

Fields:

- plumbingRepository: An instance of the PlumbingRepository class, allowing access to plumbing service data.
- success: A success message string used for responses.
- failure: A failure message string used for responses.

Methods:

- getAllPlumbingServices: Retrieves a list of all plumbing services. Documented with Swagger2 annotations for API documentation.
- getPlumbingServiceById: Retrieves a specific plumbing service by its unique ID. Documented with Swagger2 annotations.
- createPlumbingService: Creates a new plumbing service and returns a success or failure message. Documented with Swagger2 annotations.
- updatePlumbingService: Updates an existing plumbing service based on the provided ID. Documented with Swagger2 annotations.
- deletePlumbingService: Deletes a plumbing service based on its unique ID. Documented with Swagger2 annotations.

CleaningController

Class Description:

- The CleaningController class is a RESTful controller in the OneToOne application responsible for managing HTTP requests related to cleaning services. It is annotated with @RestController and includes Swagger2 annotations (@Api, @ApiOperation, @ApiResponses) for documenting the REST APIs related to cleaning.

Fields:

- cleaningRepository: An instance of the CleaningRepository class, allowing access to cleaning data.
- success: A success message string used for responses.
- failure: A failure message string used for responses.

Methods:

- getAllCleanings: Retrieves a list of all cleaning services. Documented with Swagger2 annotations for API documentation.
- getCleaningById: Retrieves a specific cleaning service by its unique ID. Documented with Swagger2 annotations.
- createCleaning: Creates a new cleaning service and returns a success or failure message. Documented with Swagger2 annotations.
- updateCleaning: Updates an existing cleaning service based on the provided ID. Documented with Swagger2 annotations.
- deleteCleaning: Deletes a cleaning service based on its unique ID. Documented with Swagger2 annotations.

ProfileController

Class Description:

- The ProfileController class is a RESTful controller in the OneToOne application responsible for managing HTTP requests related to user profiles. It is annotated with @RestController and includes Swagger2 annotations (@Api, @GetMapping, @PutMapping, @DeleteMapping) for documenting the REST APIs related to user profiles.

Fields:

- userRepository: An instance of the UserRepository class, allowing access to user data.
- success: A success message string used for responses.
- failure: A failure message string used for responses.

Methods:

- getProfileById: Retrieves a user profile by its unique ID. Documented with Swagger2 annotations.
- updatePassword: Updates the password for a user based on the provided ID. Documented with Swagger2 annotations.

- updateEmail: Updates the email for a user based on the provided ID. Documented with Swagger2 annotations.
- deleteProfile: Deletes a user profile based on its unique ID. Documented with Swagger2 annotations.

Swagger Documentation:

- The @Api annotation indicates that this controller is part of the Swagger2 documentation, providing additional information about the REST APIs related to user profiles.
- The @GetMapping, @PutMapping, and @DeleteMapping annotations are used to document specific operations, including their purpose and the expected response.

UserController

Class Description:

- The UserController class is a RESTful controller in the OneToOne application responsible for handling HTTP requests related to user management. This class includes methods for user CRUD (Create, Read, Update, Delete) operations and additional functionality like updating passwords, updating email addresses, getting user details by email, assigning a tutor to a user, user login, and more.

Fields:

- userRepository: An instance of the UserRepository class, facilitating access to user data.
- tutorRepository: An instance of the TutorRepository class, facilitating access to tutor data.
- userService: An instance of the UserService class.

Constructors:

- UserController(UserRepository userRepository, TutorRepository tutorRepository, UserService userService): Constructor to initialize the UserController with required repositories and services.

Methods:

- getAllUsers: Retrieves a list of all users. It is annotated with @GetMapping to handle HTTP GET requests for the /users endpoint.
- getProfileById: Retrieves a user's profile by their unique ID. It is annotated with @GetMapping to handle HTTP GET requests for the /users/{id} endpoint.
- updatePassword: Updates a user's password based on the provided ID. It is annotated with @PutMapping to handle HTTP PUT requests for the /users/{id}/up/{newPassword} endpoint.
- updateEmail: Updates a user's email based on the provided ID. It is annotated with @PutMapping to handle HTTP PUT requests for the /users/{id}/ue/Email endpoint.
- getUserById: Retrieves a user by their unique ID. It is annotated with @GetMapping to handle HTTP GET requests for the /users/{id} endpoint.
- createUser: Creates a new user entity. It is annotated with @PostMapping to handle HTTP POST requests for the /users endpoint.
- updateUser: Updates an existing user based on the provided ID. It is annotated with @PutMapping to handle HTTP PUT requests for the /users/{id} endpoint.
- getUserDetails: Retrieves user details by email. It is annotated with @GetMapping to handle HTTP GET requests for the /getUserDetails endpoint.
- assignLaptopToUser: Assigns a tutor to a user based on user and tutor IDs. It is annotated with @PutMapping to handle HTTP PUT requests for the /users/{userId}/tutors/{tutorId} endpoint.
- login: Handles user login based on email and password. It is annotated with @PostMapping to handle HTTP POST requests for the /login endpoint.
- deleteUser: Deletes a user based on their unique ID. It is annotated with @DeleteMapping to handle HTTP DELETE requests for the /users/{id} endpoint.

Eshanth

bookingController

Class Description:

- The bookingController class is a component of the OneToOne application responsible for handling HTTP requests related to bookings in a carpooling system. It operates as a RESTful controller and is annotated with @RestController. The primary functionalities include creating new bookings and retrieving a list of carpool listings associated with a specific user.

Fields

- bookingRepository: An instance of the bookingRepository class, facilitating access to booking data.
- bookingService: An instance of the bookingService class, providing booking-related business logic.
- userRepository: An instance of the UserRepository class, enabling access to user data.
- carpoolListingRepository: An instance of the CarpoolListingRepository class, allowing access to carpool listing data.

Constructor

- The constructor initializes the class with instances of the repositories and services needed for booking operations.

Methods

- createBooking: Handles the creation of a new booking by accepting user and carpooling IDs as parameters. Returns an appropriate ResponseEntity based on the success or failure of the booking creation.
- getCarpoolListings: Retrieves carpool listings associated with a specific user ID and returns them as a list in a ResponseEntity.

CarpoolListingController

Class Description:

- The CarpoolListingController class is a RESTful controller in the OneToOne application responsible for managing HTTP requests related to carpool listings. It is annotated with @RestController and includes Swagger2 annotations (@Api,

@ApiOperation, @ApiResponses) for documenting the REST APIs related to carpooling.

Fields:

- carpoolListingRepository: An instance of the CarpoolListingRepository class, allowing access to carpool listing data.
- success: A success message string used for responses.
- failure: A failure message string used for responses.

Methods:

- getAllCarpoolListings: Retrieves a list of all carpool listings. Documented with Swagger2 annotations for API documentation.
- getlistersById: Retrieves a specific carpool listing by its unique ID. Documented with Swagger2 annotations.
- createCarpoolListing: Creates a new carpool listing and returns a success or failure message. Documented with Swagger2 annotations.
- updateCarpoolListing: Updates an existing carpool listing based on the provided ID. Documented with Swagger2 annotations.
- deleteCarpoolListing: Deletes a carpool listing based on its unique ID. Documented with Swagger2 annotations.

Swagger Documentation:

- The @Api annotation indicates that this controller is part of the Swagger2 documentation, providing additional information about the REST APIs related to carpooling.
- The @ApiOperation annotations provide details about each specific operation, including its purpose and the expected response.
- The @ApiResponses annotations specify the possible HTTP responses for each operation, including success (code 200) and not found (code 404).

MessageController

Class Description:

- The MessageController class is a RESTful controller in the OneToOne application responsible for managing HTTP requests related to messages. It is annotated with @RestController and includes Swagger2 annotations (@Api, @ApiOperation, @ApiResponses) for documenting the REST APIs related to messages.

Fields:

- messageRepository: An instance of the MessageRepository class, facilitating access to message data.

Methods:

- markMessageAsSeen: Marks a message as seen based on the provided message ID. Documented with Swagger2 annotations for API documentation.
- saveMessage: Saves a message, setting its bold property based on frontend information. Returns a ResponseEntity indicating success or an internal server error. Documented with Swagger2 annotations.

TutorController

Class Description:

- The TutorController class is a RESTful controller in the OneToOne application responsible for handling HTTP requests related to tutors. This class is annotated with @RestController and includes various methods for performing CRUD (Create, Read, Update, Delete) operations on tutor entities.

Fields:

- tutorRepository: An instance of the TutorRepository class, facilitating access to tutor data.
- success: A success message string used for responses.
- failure: A failure message string used for responses.

Methods:

- getAllTutors: Retrieves a list of all tutors. It is annotated with @GetMapping to handle HTTP GET requests for the /tutors endpoint.
- getTutorById: Retrieves a specific tutor by its unique ID. It is annotated with @GetMapping to handle HTTP GET requests for the /tutors/{id} endpoint.
- createTutor: Creates a new tutor entity. It is annotated with @PostMapping to handle HTTP POST requests for the /tutors endpoint.
- updateTutor: Updates an existing tutor based on the provided ID. It is annotated with @PutMapping to handle HTTP PUT requests for the /tutors/{id} endpoint.
- deleteTutor: Deletes a tutor based on its unique ID. It is annotated with @DeleteMapping to handle HTTP DELETE requests for the /tutors/{id} endpoint.

UserController

Class Description:

- The UserController class is a RESTful controller in the OneToOne application responsible for handling HTTP requests related to user management. This class includes methods for user CRUD (Create, Read, Update, Delete) operations and additional functionality like updating passwords, updating email addresses, getting user details by email, assigning a tutor to a user, user login, and more.

Fields:

- userRepository: An instance of the UserRepository class, facilitating access to user data.
- tutorRepository: An instance of the TutorRepository class, facilitating access to tutor data.
- userService: An instance of the UserService class.

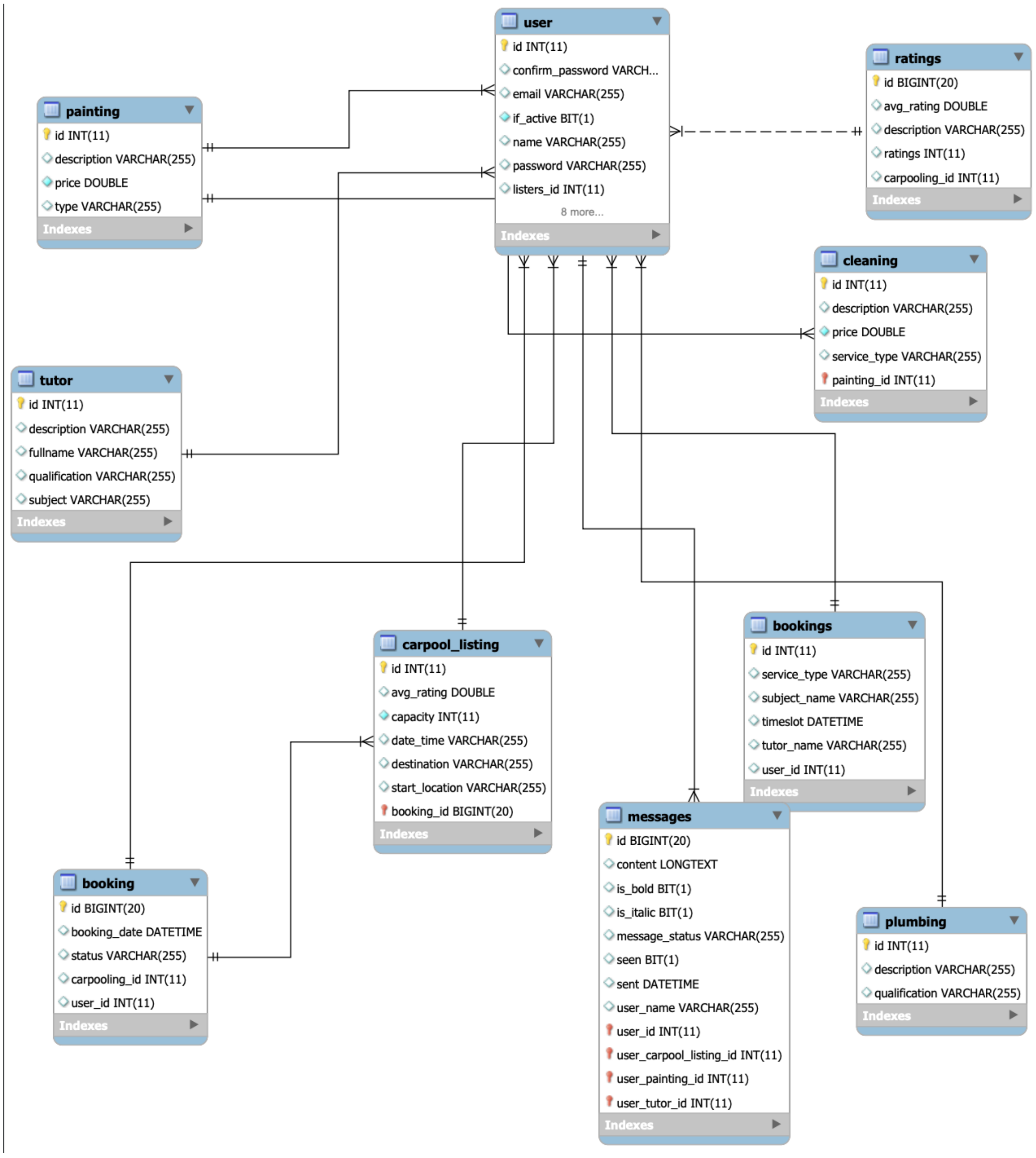
Constructors:

- UserController(UserRepository userRepository, TutorRepository tutorRepository, UserService userService): Constructor to initialize the UserController with required repositories and services.

Methods:

- getAllUsers: Retrieves a list of all users. It is annotated with @GetMapping to handle HTTP GET requests for the /users endpoint.
- getProfileById: Retrieves a user's profile by their unique ID. It is annotated with @GetMapping to handle HTTP GET requests for the /users/{id} endpoint.
- updatePassword: Updates a user's password based on the provided ID. It is annotated with @PutMapping to handle HTTP PUT requests for the /users/{id}/up/{newPassword} endpoint.
- updateEmail: Updates a user's email based on the provided ID. It is annotated with @PutMapping to handle HTTP PUT requests for the /users/{id}/ue/Email endpoint.
- getUserById: Retrieves a user by their unique ID. It is annotated with @GetMapping to handle HTTP GET requests for the /users/{id} endpoint.
- createUser: Creates a new user entity. It is annotated with @PostMapping to handle HTTP POST requests for the /users endpoint.
- updateUser: Updates an existing user based on the provided ID. It is annotated with @PutMapping to handle HTTP PUT requests for the /users/{id} endpoint.
- getUserDetails: Retrieves user details by email. It is annotated with @GetMapping to handle HTTP GET requests for the /getUserDetails endpoint.
- assignLaptopToUser: Assigns a tutor to a user based on user and tutor IDs. It is annotated with @PutMapping to handle HTTP PUT requests for the /users/{userId}/tutors/{tutorId} endpoint.
- login: Handles user login based on email and password. It is annotated with @PostMapping to handle HTTP POST requests for the /login endpoint.
- deleteUser: Deletes a user based on their unique ID. It is annotated with @DeleteMapping to handle HTTP DELETE requests for the /users/{id} endpoint.

Relationship Diagram



Swagger API

http://coms-309-063.class.las.iastate.edu:8080/swagger-ui/index.html#/