Design Document for FindRight

Group SD-327

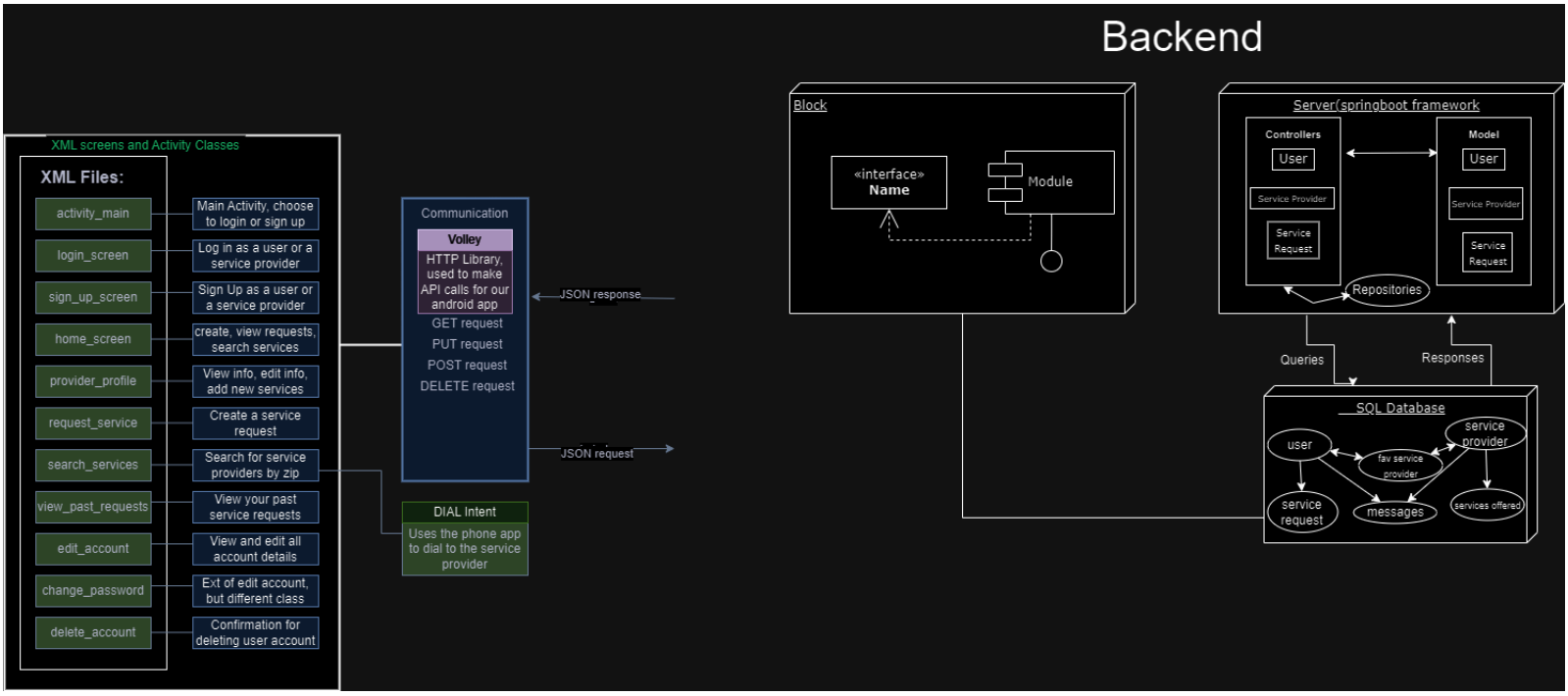
Subham Bhattacharya: 25% contribution

Chiran Subedi: 25% contribution

Dhairya Kachalia: 25% contribution

Jashwanth Kumar Komanabelli: 25% contribution

BackEnd: http://coms-309-060.class.las.iastate.edu:8080/swagger-ui/index.html#/


Backend
XML screens and Activity Classes
XML Files:
activity_main
Main Activity, choose to login or sign up
login_screen
Log in as a user or a service provider
sign_up_screen
Sign Up as a user or a service provider
home_screen
create, view requests, search services
provider_profile
View info, edit info, add new services
request_service
Create a service request
search_services
Search for service providers by zip
view_past_requests
View your past service requests
edit_account
View and edit all account details
change_password
Ext of edit account, but different class
delete_account
Confirmation for deleting user account
Communication
Volley
HTTP Library, used to make API calls for our android app
GET request
PUT request
POST request
DELETE request
JSON response
JSON request
DIAL Intent
Uses the phone app to dial to the service provider
Block
«interface»
Name
Module
Server(springboot framework
Controllers
User
Service Provider
Service Request
Model
User
Service Provider
Service Request
Repositories
Queries
Responses
SQL Database
user
service provider
fav service provider
service request
messages
services offered

Frontend

Our app has two primary actors: 'service providers' (who provide the service) and 'users' (who request services). The initial interaction occurs on the signup and login pages, where actor selects their role—'user' or 'service provider.' Depending on their choice, actors are directed to different screens, each tailored to offer role-specific features and functionalities.

Both users and service providers interact with the system through purpose-built Android views that adeptly handle user input and provide a responsive user interface. Our app boasts a modular structure featuring multiple views corresponding to the diverse screens actors can seamlessly navigate. These views, serving as the visual interface for users and service providers, initiate actions by calling upon their respective presenters.

Upon selecting their role during the signup or login process, users and service providers are seamlessly directed to role-specific screens, unlocking many tailored features. For instance, users, once identified, gain access to functionalities crafted for efficient service request management. Users can effortlessly create, edit, and explore a spectrum of service requests through an intuitive interface. Conversely, service providers are equipped with tools designed for refining service offerings, managing account details, and swiftly searching for pertinent service requests submitted by users. We use the Volley library to streamline network requests and connect the backend with the frontend. An example of this interaction between the frontend and backend is when a user clicks the log-in button in the loginScreen view, the code orchestrates the login function, which uses a GET request and checks the backend to see if there is a combination of the username and password.

Backend

Communication

The backend uses mappings to update the database based on information sent to the given mappings' URLs. These include:

• Post: send information on an item to be added to the database. For example adding a new user or service provider when signing up.

• Get: request information, often with an identifier for the specific item requested from the database. For example, retrieving the email address and other attributes for users and service providers. Also retrieving relations between different entities such as users and service providers.

• Put: send information to update a specific item in the database. Features like edit account use this to change the attribute values of particular entities. This also can be used to edit the relations like service provider and the services they can provide with.

• Delete: send an identifier to delete a specific item from the database. Currently this is used to remove instances of entities.

Controllers

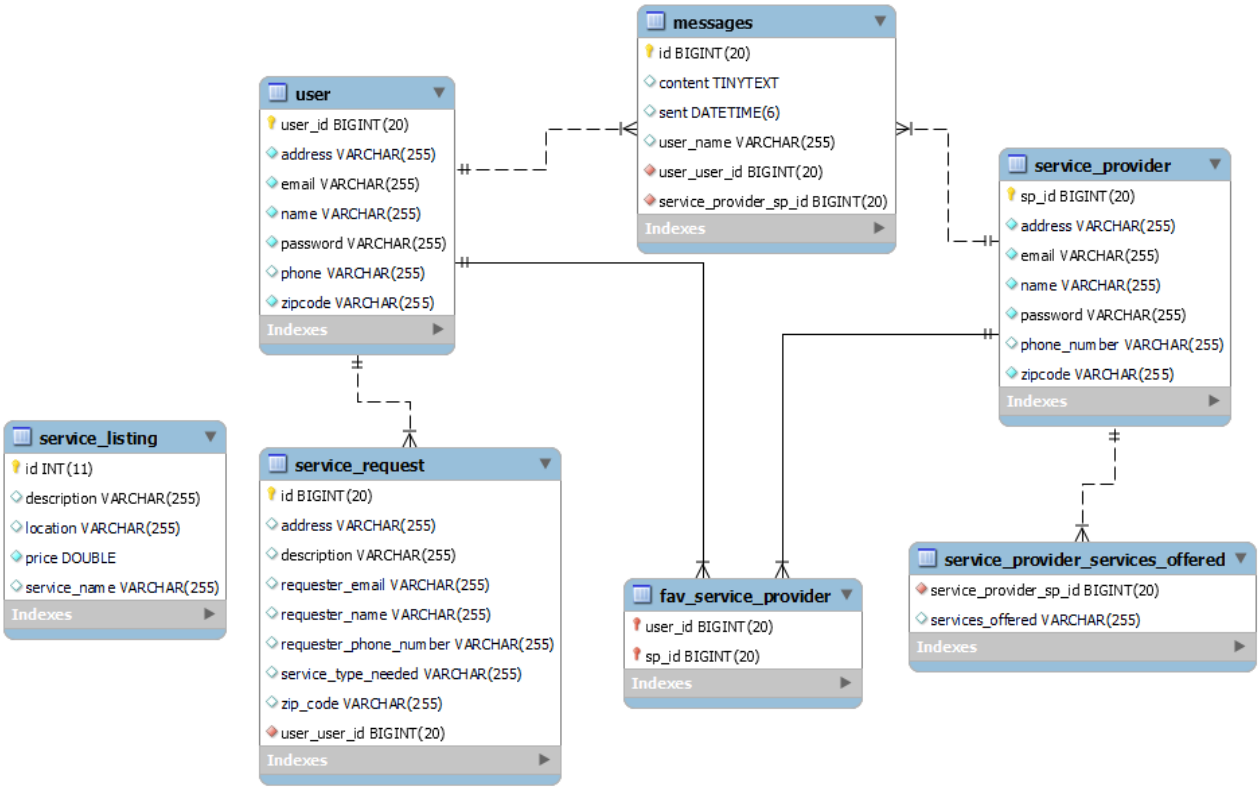
The controllers are the components that handle the communication between the frontend and the database. They define the logic and the rules for how the data is processed and displayed. The controllers you have given are:

User: This controller is responsible for creating and managing the user entity, which represents a person who uses the system. The user entity has various attributes, such as name, email, password, phone number, address, etc. The user controller provides methods to perform CRUDL (create, read, update, delete, list) operations on these attributes. The user controller also defines the relationships between the user entity and other entities, such as service provider and service request. A user can have many service providers as their favorites. This is a many to many relationship. A user can also have many service requests, meaning that they can request services from the providers they have selected. This forms a one to many relationship.

Service Provider: This controller is responsible for creating and managing the service provider entity, which represents a person or an organization that offers services to the users. The service provider entity has attributes such as name, contact information, rating, reviews, etc. The service provider controller provides methods to perform CRUDL operations on these attributes. The service provider controller also defines the relationships between the service provider entity and

other entities, such as user and service type. A service provider can have many users to be favorited by, meaning that they can serve multiple customers who request their services. A service provider can also have many service types, meaning that they can offer different kinds of services to the users. The service types are stored as a list per instance of a service provider.

Service Request: This controller is responsible for creating and managing the service request entity, which represents a request for a service made by a user to a service provider. The service request entity has attributes such as user, service provider, service type, date, time, location, status, etc. The service request controller provides methods to perform CRUDL operations on these attributes. The service request controller also defines the relationship between the service request entity and other entities, such as user. A service request has one user, meaning that it belongs to a specific customer who made the request. A service request also has one service provider and one service type, meaning that it specifies the provider and the kind of service that the user wants.


messages
id BIGINT(20)
content TINYTEXT
sent DATETIME(6)
user_name VARCHAR(255)
user_user_id BIGINT(20)
service_provider_sp_id BIGINT(20)
Indexes
user
user_id BIGINT(20)
address VARCHAR(255)
email VARCHAR(255)
name VARCHAR(255)
password VARCHAR(255)
phone VARCHAR(255)
zipcode VARCHAR(255)
Indexes
service_provider
sp_id BIGINT(20)
address VARCHAR(255)
email VARCHAR(255)
name VARCHAR(255)
password VARCHAR(255)
phone_number VARCHAR(255)
zipcode VARCHAR(255)
Indexes
service_listing
id INT(11)
description VARCHAR(255)
location VARCHAR(255)
price DOUBLE
service_name VARCHAR(255)
Indexes
service_request
id BIGINT(20)
address VARCHAR(255)
description VARCHAR(255)
requester_email VARCHAR(255)
requester_name VARCHAR(255)
requester_phone_number VARCHAR(255)
service_type_needed VARCHAR(255)
zip_code VARCHAR(255)
user_user_id BIGINT(20)
Indexes
fav_service_provider
user_id BIGINT(20)
sp_id BIGINT(20)
Indexes
service_provider_services_offered
service_provider_sp_id BIGINT(20)
services_offered VARCHAR(255)
Indexes