

Anirudh Kaluri

Date: 8 March 2022

LinkedIn: <https://linkedin.com/in/anirudhkaluri>

Github: <https://github.com/anirudhkaluri>

Leet Code: [https://leetcode.com/anirudh\\_k97](https://leetcode.com/anirudh_k97)

## **PROJECT STRUCTURE DOCUMENTATION**

There are four important directories:

- 1) models
- 2) Dao
- 3) Services
- 4) Routes

### **MODELS/ENTITIES/TABLES**

The models directory consists of 4 Entities with following attributes:

- 1) Users: #user\_id, name, phone, email
- 2) Credentials: \*user\_id, password
- 3) PersonalContacts: \*user\_id, name, phone, email
- 4) SpamNumbers: \*user\_id, phone

### **DATA ACCESS**

The Dao directory consists of 4 JS files.

Each file performs different Dao operations on their respective Models/tables.

- 1) userDao.js – Performs Dao operations on Users table
- 2) credentialDao.js - Performs Dao operations on Credentials table
- 3) contactsDao.js – Performs Dao operations on PersonalContacts table
- 4) spamDao.js – Performs Dao operations on SpamNumbers table

### **FUNCTIONALITY**

Functionality categorized under 3 different heads.

They are implemented in Services directory.

#### **1) Profile Functionality (/profileService)**

- a) Register new User
- b) Login existing user
- c) Logout
- d) Get details of Searched result along with Email from (2a) and (2b)
  - Returns above search results
  - Will return Email if (i) Searched user registered (ii) Searching user in PersonalContacts of Searched user

#### **2) Search Functionality (/searchService)**

- a) Search with Name:
  - Returns (name, phone, spam likelihood)
  - Searches on Users and PersonalContacts
- b) Search with Phone Number
  - Returns (name, phone, spam likelihood)
  - Searches first on User. If not found searches in PersonalContacts

### 3) Spam Functionality (/spamService)

- a) Add Spam Number

## API ENDPOINTS

Functionality	END POINTS	Request Method	Request Data	Response Data
Register New User	/profileService/registerUser	POST	name, phone, password, email (optional)	Text
Login	/profileService/loginUser	POST	phone, password	Text, “userid” cookie set on the client
Logout	/profileService/logout	GET	-	Text, “userid” cookie cleared
Search With Name	/searchService/searchName/:name	GET	Request URL having name in it	JSON. Array of sorted users.
Search with Number	/searchService/searchPhone/:phone	GET	Request URL having phone number in it	JSON. Array of users.
Get user details with Email	/profileService/getUserDetails	POST	JSON body with Search result	JSON. User details along with Email.
Add Spam Numbers	/spamService/addNumber	POST	JSON body with spam_number	Text.

## ROUTES

Routes have been categorized based on functionality. (Routes Directory)

- 1) searchRoutes.js : for search functionality
- 2) userRoutes.js : for profile functionality
- 3) spamRoutes.js : for spam functionality

## DATA POPULATION

Data Directory consists of populateData.js File which is executed when the app launches to populated all tables with data.

## **SPAM LIKELIHOOD FORMULA**

When a number (say X) is searched for spam likelihood

1. spamHits=Number of times X marked as spam by Users
2. (spamHits/Spam table size) gives the spamming tendency relative to other spam numbers. It is scored out of 100.
3. (spamHits/Number of Users registered) gives the spamming popularity among registered users . It is scored out of 100.
4. Total scored out of 200.

## **SCALABILITY/FUTURE REQUIREMENTS**

1. Spam Numbers stored along with userid of person who stores it so that users can check if they already marked a number as spam.
2. UserId used as Primary and Foreign Key instead of Phone Numbers despite Phone Numbers being unique so that User can change Phone Number in the future
  - a) No need to update PersonalContacts table when Phone Number changes
  - b) No need to update SpamNumbers table when Phone Number changes
3. Passwords stored in separate table instead of Users table because it is used only for authentication (Login and Registration)
4. Login and Logout Functionality implemented to make the project wholesome for testing(not explicitly mentioned in the document)

## **SECURITY FEATURES**

- 1) Services work only if the user is logged in. No unauthorized access.
- 2) Accessing routes other than API endpoints is restricted.
- 3) Passwords stored in Hashed format in the database. Any leakage wont give away credentials.

## SOME SCREENSHOTS WITH POSTMAN TOOL

These requests have been exported into instahyre.postman\_collections.json

### CREATE NEW USER

The screenshot shows the Postman interface with a collection named "Instahyre". A POST request titled "Create new user" is selected. The URL is "localhost:8000/profileService/registerUser". The "Body" tab is active, showing a JSON payload:

```
1 {  
2   ... "name": "ramana",  
3   ... "phone": "0885988500",  
4   ... "email": "ramanarao@gmail.com",  
5   ... "password": "rootuserlocks"  
6 }
```

The response status is 200 OK, and the body of the response is "user added successfully".

### LOGOUT

The screenshot shows the Postman interface with the same "Instahyre" collection. A GET request titled "Logout" is selected. The URL is "localhost:8000/profileService/logout". The "Authorization" tab is active, showing a note: "The authorization header will be automatically generated when you send the request. Learn more about [authorization](#)". The status bar indicates "This request is using No Auth from collection Instahyre".

The response status is 200 OK, and the body of the response is "Please login to continue".

## LOGIN

The screenshot shows the Postman interface for a POST request to `localhost:8000/profileService/loginUser`. The request body is set to JSON and contains the following data:

```
1 {  
2   "phone": "1234567890",  
3   "password": "root"  
4 }
```

The response status is 200 OK with a message: "You have successfully logged in".

## SEARCH WITH NUMBER

The screenshot shows the Postman interface for a GET request to `localhost:8000/searchService/searchPhone/1231231234`. The request includes a query parameter "phone": "1231231234".

The response status is 200 OK and the JSON data returned is:

```
1 [  
2   {  
3     "user_id": 2,  
4     "name": "somesh",  
5     "phone": "1231231234",  
6     "spam_likelihood": 83.33333333333333  
7   },  
8   {  
9     "user_id": 3,  
10    "name": "somesh",  
11    "phone": "1231231234",  
12    "spam_likelihood": 83.33333333333333  
13  }  
14 ]
```

## SEARCH WITH NAME

The screenshot shows the Postman application interface. The left sidebar has a tree view with a node 'Instahyre' expanded, showing various API endpoints like 'Create new user', 'Logout', 'Login', etc. The main workspace is focused on a 'GET /searchService/searchName/som' request. The 'Params' tab is selected, showing a single parameter 'Key' with value 'Value'. The 'Body' tab shows a JSON response with three objects, each containing 'name', 'phone', and 'spam\_liklihood' fields. The status bar at the bottom indicates a 200 OK response with 18 ms time and 497 B size.

```
[{"id": 1, "name": "somu", "phone": "0987654321", "spam_liklihood": 0}, {"id": 2, "name": "somesh", "phone": "1231231234", "spam_liklihood": 106.66666666666666}, {"id": 3, "name": "somesh", "phone": "1231231234", "spam_liklihood": 106.66666666666666}
```

## GET USER DETAILS ALONG WITH MAILS IF POSSIBLE FROM THE SEARCH RESULT

The screenshot shows the Postman application interface. On the left, there's a sidebar with a tree view of collections: 'space' (selected), 'Instahyre' (expanded), and 'SEQUELIZE'. Under 'Instahyre', several endpoints are listed: 'POST Create new user', 'GET Logout', 'POST Login', 'POST Add spam number', 'GET Search Number', 'POST Get user mail' (selected and expanded), and 'GET Search with name'. The main workspace shows the 'Get user mail' endpoint details. The URL is 'localhost:8000/profileService/getUserDetails'. The 'Tests' tab is active, containing a single test script: '1'. To the right of the tests, a note says: 'Test scripts are written in JavaScript, and are run after the response is received. Learn more about [test scripts](#)'. Below the tests, there are snippets for environment variables. At the bottom, the response body is displayed in 'Pretty' format, showing a JSON object with fields: name, phone, spam\_likelihood, and email. The status bar at the bottom indicates: Status: 200 OK, Time: 18 ms, Size: 377 B, and Save Response.

space

New Import < view POST Create POST Login POST Add Instahyre GET Search POST Get u GET Search GET Logout > + ... No Environment

Instahyre / Get user mail

Save ...

Send Cookies

POST Create new user

GET Logout

POST Login

POST Add spam number

GET Search Number

POST Get user mail

GET Search with name

> SEQUELIZE

Params Authorization Headers (8) Body Pre-request Script Tests Settings

1

Test scripts are written in JavaScript, and are run after the response is received. Learn more about [test scripts](#)

Snippets

Get an environment variable

Get a global variable

Get a variable

Get a collection variable

Set an environment variable

Set a global variable

Body Cookies (1) Headers (9) Test Results

Status: 200 OK Time: 18 ms Size: 377 B Save Response

Pretty Raw Preview Visualize JSON

```
1
2   "name": "anirudh",
3   "phone": "8919312196",
4   "spam_likelihood": 0,
5   "email": "random123@gmail.com"
6
```

Find and Replace Console Cookies Capture requests Runner Trash

## ADD SPAM NUMBER

The screenshot shows the Postman application interface. On the left, there's a sidebar with a tree view of API endpoints under 'Instahyre'. The selected endpoint is 'POST Add spam number'.

The main area shows a POST request to 'localhost:8000/spamService/addNumber'. The 'Body' tab is selected, showing the JSON payload:

```
1 {  
2   "spam_number": "4545454545"  
3 }
```

Below the request, the response status is 'Status: 200 OK' with a time of '9 ms' and a size of '312 B'. The response body is displayed as:

```
1 Saved spam number successfully
```

At the bottom, there are various navigation and utility buttons like 'Find and Replace', 'Console', 'Cookies', 'Capture requests', 'Runner', 'Trash', etc.