# SMS Magic-Coding Exercise

Name: Prajwal Rawate

Qualification: BE-Computer Engineering

Codenera: Java June 2023 Batch

## Coding Exercise

This is a simple coding exercise and You can choose any programming language of your choice. If you wish you add the code to github, please do so and share the repo URL for review. Also it would be good if you are able to do a simple recording of the usage of the APIs working, it would be great.

## Entities

### Client:

This Client entity represents a client in a system. Here's a brief description of its attributes and relationships:

id: This is the primary key of the Client entity, generated automatically using an identity strategy (GenerationType.IDENTITY).

name: Represents the name of the client.

company: This is a Many-to-One relationship with the Company entity. It represents the company to which the client belongs.

user: This is another Many-to-One relationship with the User entity. It represents the user associated with this client.

email: Represents the email address of the client

phone: Represents the phone number of the client. The Client entity also includes constructors to create instances with all attributes, getters and setters for all attributes to allow access to them, and a default constructor. This entity can be used to manage clients in the application, associating them with specific companies and users, and storing their contact information.

---------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------

**clientUser:**

This ClientUser entity represents a relationship between a client, a user, and a company in a system. Here's a brief description of its attributes and relationships:

id: This is the primary key of the ClientUser entity, generated automatically using an identity strategy (GenerationType.IDENTITY).

client: This is a Many-to-One relationship with the Client entity. It represents the client associated with this relationship.

user: This is another Many-to-One relationship with the User entity. It represents the user associated with this relationship.

company: This is also a Many-to-One relationship with the Company entity. It represents the company associated with this relationship.

createdAt: Represents the date and time when this relationship was created.

updatedAt: Represents the date and time when this relationship was last updated.

deletedAt: Represents the date and time when this relationship was deleted or marked as inactive.

active: Indicates whether this relationship is currently active or not.

------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------

**User:**

This User entity represents a user in the system. Here's a brief description of its attributes and relationships:

id: This is the primary key of the User entity, generated automatically using an identity strategy (GenerationType.IDENTITY).

username: Represents the username of the user.

clientUsers: This is a One-to-Many relationship with the ClientUser entity, indicating the list of client-user relationships associated with this user

------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------

**Company:**

This Company entity represents a company in the system. Here's a brief description of its attributes and relationships:

id: This is the primary key of the Company entity, generated automatically using an identity strategy (GenerationType.IDENTITY).

name: Represents the name of the company.

clientUsers: This is a One-to-Many relationship with the ClientUser entity, indicating the list of client-user relationships associated with this company.

------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------

**ENDPoints:**

**GET /api/users:**

Endpoint to list all users.

Returns a list of all users in the system.

------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------

**PUT /api/users/{id}:**

Endpoint to replace some fields of a specific user.

Accepts a user ID in the path and the updated user details in the request body.

Replaces the specified fields of the user with the new values.

Returns the updated user details.

------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------

**POST /api/clients:**

Endpoint to create a new client.

Accepts client details in the request body.

Saves the new client details to the database.

Retrieves and sets the associated company and user for the client.

Returns the created client details.

PATCH /api/clients/{id}:

---------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------

Endpoint to partially update a specific client's fields.

Accepts a client ID in the path and the updated client details in the request body.

Updates the specified fields of the client with the new values.

Also updates the associated user and company fields if provided in the request body.

Returns the updated client details.

These endpoints allow for CRUD (Create, Read, Update, Delete) operations on users and clients in the system. They handle requests to manipulate user and client data, interacting with the corresponding repositories and services to perform database operations.

---------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------

**SQL**

(3) Create some function with some custom query for the following

3.1. Search for Companies by employees range.

Employees range: i.e. list Companies that has employees range between 500 and 2000

Query:

```
SELECT * FROM companies
WHERE employees BETWEEN 500 AND 2000;
```

-------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------