Role-Based Access Control (RBAC) Overview:

Role-Based Access Control (RBAC) is a system architecture where users are assigned specific roles (such as Private User or Commercial Client), and these roles determine what features and sections of the app they can access. Both Private Users and Commercial Clients can exist on a single platform, but they are served different content and features based on their assigned role.

Example of Role-Based Access Control (RBAC):

- **Private User:**
  - Can access discounts, view offers, redeem vouchers, and make purchases from commercial clients.
  - Cannot access offer creation tools or analytics.
- **Commercial Client:**
  - Can create offers, view analytics (e.g., redemptions, customer behavior), and manage customer interactions.
  - Cannot access the discount redemption features meant for private users.

How the Login Screen Will Look:

To support RBAC, the Login Screen will remain simple and consistent for all users, but once authenticated, the system will determine the user's role and direct them accordingly.

Example of the Login Screen:

1. **Basic Form:**
   - Email or Phone Number field
   - Password field
   - Login Button
2. **Additional Features:**
   - Sign Up Link: A link that leads users to a registration page for both private users and commercial clients.
   - Forgot Password Link
3. There is no need to explicitly ask the user to identify themselves as a Private User or Commercial Client on the login screen. Instead, the system will determine their role after authentication.

Backend Logic for RBAC:

1. **Role Assignment During Registration:**
   - During user sign-up (whether for a private user or commercial client), the user's role is stored in the database.
   - Example:
     - A private user would be assigned a role of "private_user".
     - A commercial client would be assigned a role of "commercial_client".
   - This role is tied to the user's account, and it is referenced every time the user logs in.

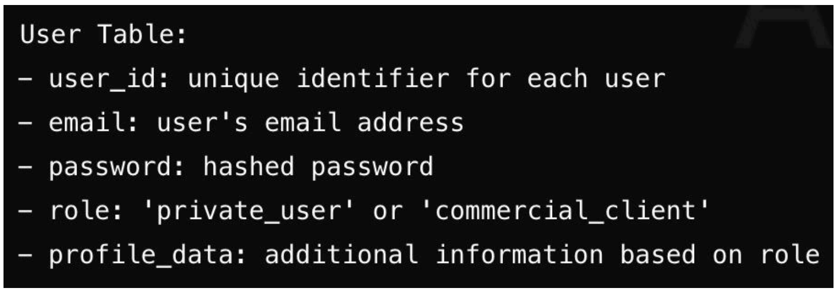
2. **Login Process:**
   - The user enters their credentials (email/password) on the login screen.
   - The backend checks the user's credentials against the database.
   - If the credentials are valid, the backend retrieves the user's role (either "private_user" or "commercial_client").
   - The role is embedded into the user's session token (JWT) or stored in the session state.
   - The front-end UI will dynamically adapt based on the user's role.
     - For a private user, the UI displays features such as browsing offers, viewing discounts, and redeeming vouchers.
     - For a commercial client, the UI displays features such as creating offers, managing redemptions, and viewing analytics.

UI Behavior After Login:

1. **Private User UI:**
   - The user sees the main dashboard with features like:
     - Offers from commercial clients.
     - Filtered shops and offers on Google Maps.
     - Redeem vouchers or discounts.
   - No access to offer creation or analytics.
2. **Commercial Client UI:**
   - The user is directed to the MyFirstAdmin dashboard where they see:
     - Offer management tools (e.g., create/edit offers).
     - Offer performance analytics.
     - Customer management features.
   - No access to redeem discounts or view offers like private users.

Backend Understanding of Each Account:

1. **User Database Schema:** The backend will have a unified user database where each user’s data is stored with an associated role.
2. **Example Schema:**



```
User Table:
- user_id: unique identifier for each user
- email: user's email address
- password: hashed password
- role: 'private_user' or 'commercial_client'
- profile_data: additional information based on role
```

1. **Session or JWT Tokens:**
   - When a user logs in, the system generates a session or JWT token that includes:
     - user_id
     - role
     - token expiration
   - This token is sent to the frontend, and all subsequent API requests include this token.
   - The backend checks the user’s role from the token before authorizing actions (such as creating an offer or redeeming a voucher).

Technical Stacks Overview:

Authentication:

- JWT Tokens: For securely passing user information (including their role) between the frontend and backend.
- OAuth 2.0 or Firebase Authentication can also be used for user authentication.

Database:

- PostgreSQL or MongoDB: For storing user data and associated roles.
- A well-structured database schema will allow you to track users, their roles, and any related data (such as offers for commercial clients).

Backend Role Management:

- Use libraries like Passport.js or JWT with Node.js to handle authentication and RBAC.
- In Nest.js, decorators and guards can be used for role-based permissions on specific routes.

Switching Accounts:

- Role-based toggling: If the user has both private user and commercial client roles, they can switch roles within the app by toggling roles within the UI.
- Backend simply refreshes their token or session with the updated role.