9/28/2024

Backend Server

Server for Multi-App support

Peter Sameh
DYNAMO MEA

Table of Contents

1. Database Design
2. Elasticsearch
3. Authentication Solution
4. API Endpoints

A. Database design (ERD)

PDF - Image

Note: Files are inserted in the same folder

Considerations:

1. **Soft Delete and Blocked Users**
   While not explicitly included in the ERD or listed in the API endpoints, two fields should be present in the collections:
   - **isBlocked**: Applies to user models to prevent access when necessary (e.g., for violating terms).
   - **isDeleted**: Implemented for soft deletes, ensuring that records are marked as deleted without being removed from the database.
2. **Base User Model**
   The database involves seven user roles, each sharing common attributes such as emails and usernames. To avoid redundancy and ensure unique constraints (e.g., preventing duplicate emails or usernames), we can:
   - Create a **BaseUser model** containing all the shared properties across the user roles (e.g., email, username, password, etc.).
   - Have each role-specific model (e.g., Customer, Vendor, Admin, etc.) inherit from this base model, with additional attributes specific to that role stored in smaller tables. This structure will optimize design and simplify validation across roles.
3. **Product Model Structure**
   Products are divided into three distinct models to suit the business requirements. However, creating a **BaseProduct model** is unnecessary. Introducing such a base model would add more references, which could degrade performance. The separation of product models is intentional to cater to specific use cases, and combining them into a shared base would not offer significant value.
4. **Indexing for Faster Searching**
   To enhance search performance, specific fields should be indexed, For example::
   - **Single-Field Index on Email:** This index will allow for quick lookups of users by their email addresses, significantly improving the speed of user authentication and profile retrieval.

B. Elasticsearch

Elasticsearch Integration

Elasticsearch will be integrated into the system to enhance search capabilities and improve the performance of data retrieval processes. Here's how it will be implemented and used:

1. **Data Synchronization**:
   - The application will utilize an ETL (Extract, Transform, Load) process to synchronize data between the primary database (e.g., MongoDB) and Elasticsearch. This ensures that any changes made to the database are reflected in Elasticsearch, maintaining data consistency.
2. **Indexing Strategy**:
   - Relevant collections, such as products and users, will be indexed in Elasticsearch to enable full-text search capabilities. Custom indexing strategies will be employed to optimize search relevance based on business needs.
3. **Search Functionality**:
   - The system will leverage Elasticsearch's powerful querying capabilities to perform complex searches, including full-text search, filtering, and aggregation. This will enhance user experience by providing faster and more relevant search results compared to traditional database queries.
4. **Analytics and Reporting**:
   - Elasticsearch will be used for real-time analytics, allowing the system to gather insights from user interactions, product searches, and other activities. This data can inform business decisions and improve overall system performance.
5. **Scalability**:
   - As the system grows, Elasticsearch's distributed architecture will allow for easy scaling to handle increased data volume and search traffic, ensuring that the system remains responsive under load.

C. Authentication System

1. Overview

The system is designed to handle authentication for three separate app categories using a shared backend server:

- **Customer App**: `/api/customers`
- **Rider App**: `/api/riders`
- **Vendor, Worker, Admin, Tech Support, and Sales Apps**: `/api`

The system is implemented with OAuth 2.0 principles, JWT tokens (Access and Refresh Tokens), and an OTP-based MFA for higher roles like Admins, Tech Support, and Sales.

2. Signup and Login Flow

Roles with Signup Capability:

1. **Customer** (via Customer App)
2. **Rider** (via Rider App)
3. **Worker** (via Worker interface under `/api`)

For other roles like **Admin**, **Tech Support**, **Sales**, and **Vendor**, the accounts are created by an **Admin** or **SuperAdmin**.

---

3. Token Management: Access & Refresh Tokens

- **Access Token**: Short-lived token used for API requests. Contains user information and role to authorize access.
- **Refresh Token**: Longer-lived token used to obtain new access tokens without requiring a login.

Both tokens will be issued upon a successful login/signup.

Access Token Structure:

```
{
  "user_id": "<user_id>",
  "role": "<role>",
  "expires_in": "<expiry_time>"
}
```

Refresh Token Structure:

```
{
  "user_id": "<user_id>",
  "role": "<role>",
  "expires_in": "<expiry_time>"
}
```

Refresh Token Endpoints:

- **Customer**:
  - `POST /api/customers/auth/refresh-token`
- **Rider**:
  - `POST /api/riders/auth/refresh-token`
- **Worker, Vendor, Admin, Sales, Tech Support**:
  - `POST /api/users/auth/refresh-token`

In each case, a valid refresh token is required to generate a new access token.

---

4. MFA (Multi-Factor Authentication)

OTP System:

- **Optional MFA**:
  1. **Customer**
  2. **Rider**
  3. **Vendor**
  4. **Worker**

  These users can enable MFA for additional security (e.g., during account setup).

- **Mandatory MFA**:
  1. **Sales**
  2. **Admin / Superadmin**
  3. **Tech Support**

  MFA is required for higher-privileged users during login for added security.

OTP Endpoints:

- **Customer/Rider/Worker/Vendor**:
  - `POST /api/customers/auth/verify-otp`
  - `POST /api/riders/auth/verify-otp`
  - `POST /api/users/auth/verify-otp`

OTP will be generated during:

- Signup (for customers, riders, workers)
- Login if MFA is enabled

---

5. Endpoints by Role and Use Case

5.1 Customer App Endpoints (`/api/customers`)

- **Signup**: `POST /api/customers/auth/signup`
- **Login**: `POST /api/customers/auth/login`
- **Social Login**: `POST /api/customers/auth/social-login`
- **Verify OTP (Signup or MFA)**: `POST /api/customers/auth/verify-otp`
- **Resend OTP**: `POST /api/customers/auth/resend-otp`
- **Forgot Password**: `POST /api/customers/auth/forget-password`
- **Reset Password**: `POST /api/customers/auth/reset-password`
- **Refresh Token**: `POST /api/customers/auth/refresh-token`

5.2 Rider App Endpoints (`/api/riders`)

- **Signup**: `POST /api/riders/auth/signup`
- **Login**: `POST /api/riders/auth/login`
- **Verify OTP (Signup or MFA)**: `POST /api/riders/auth/verify-otp`
- **Resend OTP**: `POST /api/riders/auth/resend-otp`
- **Forgot Password**: `POST /api/riders/auth/forget-password`
- **Reset Password**: `POST /api/riders/auth/reset-password`
- **Refresh Token**: `POST /api/riders/auth/refresh-token`

5.3 Vendor, Worker, Admin, Sales, Tech Support App Endpoints (`/api`)

- **Worker Signup**: `POST /api/workers/auth/signup`
- **Login (Admin, Worker, Sales, Vendor, Tech Support)**: `POST /api/users/auth/login`
- **Verify OTP (MFA)**: `POST /api/users/auth/verify-otp`
- **Forgot Password**: `POST /api/users/auth/forget-password`
- **Reset Password**: `POST /api/users/auth/reset-password`
- **Refresh Token**: `POST /api/users/auth/refresh-token`

---

6. Role-Based Token Structure

The **Access Token** for all roles will contain the role in the payload in JWT, enabling role-based authorization in the backend.

Based on the role in the token, specific permissions will be enforced for each API request.

---

7. Database and Models

To ensure separation of concerns and avoid conflicts during login and signup, each role will have its own table or collection in the database:

1. **Customers**: Stored in `customers` table/collection.
2. **Riders**: Stored in `riders` table/collection.
3. **Vendors, Workers, Admins, Sales, Tech Support**: Stored in `users` table/collection, with an additional field for role differentiation.

This model division ensures that when users log in, the system searches only in the relevant model based on the API endpoint being accessed.

---

8. SuperAdmin Role

The **SuperAdmin** is created when the system is initialized and is responsible for creating accounts for the following roles:

- **Admins**
- **Sales**
- **Tech Support**
- **Vendors**

The SuperAdmin has exclusive access to the following endpoints:

- `GET /api/admins (Get all admins)`
- `POST /api/admins (Create a new admin)`
- `PATCH /api/admins (Update admin status)`
- `DELETE /api/admins (Delete an admin)`

SuperAdmin and Admins have access to the following endpoints:

- `POST /api/admins/users (create new: Vendor, Sales, Tech-support`

9. Security Considerations

1. **JWT Expiration**:
   - Access tokens have a short expiry (e.g., 15-30 minutes) to reduce the risk of token misuse.
   - Refresh tokens have a longer expiry (e.g., 7 days) and are securely stored.
2. **MFA Enforcement**:
   - Mandatory MFA for Admins, Tech Support, and Sales ensures an additional security layer, reducing unauthorized access risks.
3. **Rate Limiting**:
   - Implement rate limiting on OTP endpoints to prevent abuse (e.g., `POST /auth/resend-otp`).
4. **Password Hashing**:
   - Store passwords securely using industry-standard hashing algorithms such as bcrypt.

---

10. Social Login (OAuth 2.0)

For customers, the system supports **Social Login** (e.g., Google, Facebook) using OAuth 2.0. The backend verifies the token from the social provider and generates JWT tokens (access and refresh) for the user in the same manner as standard login.

- Endpoint: `POST /api/customers/auth/social-login`
- The system will handle token verification with the social provider and user creation in the database.

D. Endpoints List

NOTE: detailed endpoint located in .txt file

A. Customer App API

BaseURL: /api/customers

1. Auth

- POST /auth/signup
- POST /auth/verify-otp
- POST /auth/resend-otp
- POST /auth/login
- POST /auth/social-login
- POST /auth/forget-password
- POST /auth/reset-password
- POST /auth/refresh-token

2. Profile

- GET /profile
- PATCH /profile (username, email, phone, DOB, firstname, lastname)
- POST /profile/upload-image
- PATCH /profile/change-password
- DELETE /profile
- GET /profile/wallet
- GET /profile/notifications

3. Home Flow

- GET /categories
- GET /categories/:categoryId?tag=xyz&search=abc&sort-by=name&sort-order=asc|desc
- GET /categories/:categoryId/collections/:collectionId?tag=xyz&ordered-before=[1|0]
- GET /categories/:categoryId/collections/:collectionId/products/:productId

- GET /categories/:categoryId/collections/:collectionId/reviews
- GET /categories/:categoryId/services/:serviceId/reviews
- GET /workers/:workerId

4. Cart Management

- GET /cart
- POST /cart
- PATCH /cart/:itemId?action=[delete|comment]
- DELETE /cart/:itemId

5. Favourite List

- GET /fav-list?type=[products|collections]&tag=xyz
- PATCH /fav-list

6. Orders

- GET /orders
- GET /orders/:orderId
- POST /orders
- PUT /orders/:orderId/cancel
- POST /orders/:orderId/return

7. Coupons

- GET /coupons
- POST /coupons/claim
- POST /coupons/redeem

8. Reviews

- POST /categories/:categoryId/collections/:collectionId/reviews
- POST /categories/:categoryId/collections/:collectionId/products/:productId/reviews
- POST /categories/:categoryId/services/:serviceId/reviews
- GET /reviews
- GET /reviews/:reviewId
- GET /reviews/undone
- PATCH /reviews/:reviewId

- DELETE /reviews/:reviewId

9. Addresses

- GET /addresses
- POST /addresses
- PATCH /addresses/:addressId

10. Chats

- POST /api/customers/chats
- GET /api/customers/chats/:chatId
- POST /api/customers/chats/:chatId

B. Rider App API **BaseURL**: /api/riders

1. Auth

- POST /auth/signup
- POST /auth/verify-otp
- POST /auth/resend-otp
- POST /auth/login
- POST /auth/forget-password
- POST /auth/reset-password
- POST /auth/refresh-token

2. Profile & Availability

- GET /profile
- PATCH /profile
- POST /profile/upload-image
- PATCH /profile/status
- PATCH /profile/change-password
- GET /notifications

3. Orders

- GET /orders?status=[pending|assigned]
- GET /orders/:orderId
- PUT /orders/:orderId/status

4. Vehicle Management

- GET /vehicle
- PUT /vehicle

Flow: (must check accountStatus)

1-Reciving order from notification

2-Go for status=pending and Rider can accept or deny

4-If approve then change order status to shipped && change status to assigned

5-After deliver change order status to delivered

C. Vendor, Worker, Admin, Tech Support, and Sales API

1- BaseURL: /api

2- after auth (MUST: send role in headers (inside JWT) or error**)**

3- IMPORTANT for each /users/ will be divided into /vendors /workers /admins /tech-supports /sales

1. Auth (All Roles)

- POST /workers/auth/signup (for workers only)
- POST /users/auth/login
- POST /users/auth/verify-otp
- POST /users/auth/forget-password
- POST /users/auth/reset-password
- POST /users/auth/refresh-token

2. Profile & Settings (All Roles)

- GET /users/profile
- POST /users/profile/upload-image
- PATCH /users/profile
- PATCH /users/change-password
- GET /users/notifications

3. Vendor Management

- GET /vendors/collections?tag=xyz&search=abc
- POST /vendors/collections
- GET /vendors/collections/:collectionId?tag=xyz
- PATCH /vendors/collections/:collectionId
- DELETE /vendors/collections/:collectionId
- POST /vendors/collections/:collectionId/products
- PATCH /vendors/collections/:collectionId/products/:productId
- DELETE /vendors/collections/:collectionId/products/:productId
- GET /vendors/orders?collection=abc&status=[any]
- GET /vendors/orders/:orderId

- PATCH /vendors/orders/:orderId/status
- GET /vendors/reviews
- GET /vendors/revenue?collection=abc

4. Worker Management

- GET /workers/services
- POST /workers/services/:serviceId/reservations
- GET /workers/services/:serviceId/reservations
- PATCH /workers/services/:serviceId/reservations/:reservationId/status
- GET /workers/:workerId/working-hours-days
- PUT /workers/:workerId/working-hours-days

5.SuperAdmin Management:

- GET /admins (Get admins)
- POST /admins (Create a new admin)
- PATCH /admins (Update admin statis)
- DELETE /admins (Delete admin)

5. **Admin Management**

- GET /admins/dashboard (Get overview stats)
- POST /admins/users (Create new user: vendor, sales, techSupport)
- GET /admins/users (List all users)
- GET /admins/users/:userId (Get user details)
- PATCH /admins/riders/:rider/status (Update rider status) //for rider status
- DELETE /admins/users/:userId (Delete user)
- GET /admins/categories (Get categories)
- POST /admins/categories (Create category)
- PATCH /admins/categories/:categoryId (Update category)
- DELETE /admins/categories/:categoryId (Delete category)
- GET /admins/collections (Get collections)
- DELETE /admins/collections/:collectionId (Delete collection)
- PATCH /admins/collections/:collectionId/block (Block and unblock collection) (send in body)
- GET /admins/services (Get services)

- POST /admins/services (Create service)
- PATCH /admins/services/:serviceId (Update service)
- DELETE /admins/services/:serviceId (Delete service)
- GET /admins/tags (Manage tags)
- POST /admins/tags (Create tag)
- PATCH /admins/tags/:tagId (Update tag)
- DELETE /admins/tags/:tagId (Delete tag)

6. Sales Management

- POST /sales/coupons (Create new coupon)
- GET /sales/coupons (List all coupons)
- GET /sales/coupons/:couponId (Get coupon details)
- PATCH /sales/coupons/:couponId (Update coupon)
- DELETE /sales/coupons/:couponId (Delete coupon)

7. Tech Support

- GET /techsupports/chats?status=[active|resolved] (List all active chats)
- GET /techsupports/chats/:chatId (Get specific chat)
- POST /techsupports/chats/:chatId (Send a message)
- PUT /techsupports/chats/:chatId/status (Update chat status: Close chat)