Routes Documentation:-

User Management Endpoints

`GET  /api/users`
Retrieves a list of all users in the system.
Access restricted to users with the `ADMIN` role.

`GET  /api/users/{id}`
Fetches the user record for the specified `id`, clearing sensitive fields before returning.
Accessible to roles `USER`, `EVENT_ORGANIZER`, or `ADMIN`.

`DELETE  /api/users/{id}`
Removes the user with the given `id` from the in-memory store.
Access restricted to users with the `ADMIN` role.

`GET  /api/users/all`
Public endpoint that returns a simple “Public Content.” string.
No authentication or role required.

`GET  /api/users/user`
Returns “User Content.” for authenticated users.
Accessible to roles `USER`, `EVENT_ORGANIZER`, or `ADMIN`.

`GET  /api/users/organizer`
Returns “Event Organizer Board.” for event organizers.
Accessible only to users with the `EVENT_ORGANIZER` role.

`GET  /api/users/admin`
Returns “Admin Board.” for administrators.
Accessible only to users with the `ADMIN` role.

User–Event Association Endpoints

`POST   /api/users/{userId}/events/{eventId}`
Adds the event identified by `eventId` to the user’s list of events.
Returns success or error message in JSON.

`DELETE /api/users/{userId}/events/{eventId}`
Removes the event identified by `eventId` from the user’s list.
Returns success or error message in JSON.

Wallet Service Endpoints

`GET   /api/wallet/balance-alt?userId={userId}`
Alias for the balance endpoint; returns wallet balance in a JSON map.
Requires a valid Bearer token in the `Authorization` header.

`GET   /api/wallet/balance?userId={userId}`
Retrieves the current wallet balance for the given userId.
Returns a JSON object with `userId`, `balance`, and optional `message`.

`POST  /api/wallet/add-funds?userId={userId}&amount={amount}`
Adds funds to the user’s wallet and returns updated balance.
Requires a valid Bearer token in the `Authorization` header.

`POST  /api/wallet/deduct-funds?userId={userId}&amount={amount}`
Deducts funds from the user's wallet and returns updated balance.
Requires a valid Bearer token in the `Authorization` header.

`POST  /api/wallet/create-wallet?userId={userId}[&initialBalance={balance}]`
Creates a new wallet for the user with an optional initial balance (default 1000).
Requires a valid Bearer token in the `Authorization` header.

`GET   /api/wallet/transactions?userId={userId}`
Fetches the transaction history for the specified user.
Requires a valid Bearer token in the `Authorization` header.

`GET   /api/wallet/check-wallet?userId={userId}`
Checks whether a wallet exists for the specified user.
Requires a valid Bearer token in the `Authorization` header.

`GET   /api/wallet/test-token`
Echoes back the received `Authorization` header and extracted token.
Useful for debugging token extraction logic; no auth enforced.

`POST  /api/wallet/init-db`
Initializes (mocks) the wallet database for testing purposes.
Returns a JSON success message.

Web Page Controller Endpoints

`GET /google-login`
Serves the Google login HTML page view.
No JSON; returns a rendered template.

`GET /auth-success`
Serves the authentication success page view after OAuth.
No JSON; returns a rendered template.

`GET /auth-error`
Serves the authentication error page view for failed logins.
No JSON; returns a rendered template.

AuthController (disabled; no base path)
Note: This controller is disabled in favor of SupabaseAuthController.

POST /signin
Description: Authenticates a user by email and password via Supabase.
Method: POST

POST /signup
Description: Registers a new user account using email and password via Supabase.
Method: POST

POST /signout
Description: Signs the user out of the current session and returns a confirmation message.
Method: POST

POST /reset-password
Description: Initiates a password reset flow for the given email address.
Method: POST

POST /email/signup
Description: Registers a new user using a raw EmailAuthRequest payload.
Method: POST

POST /email/signin
Description: Authenticates a user using a raw EmailAuthRequest payload.
Method: POST

POST /phone/signup
Description: Registers a new user using phone number authentication.
Method: POST

POST /phone/signin
Description: Sends an OTP to the given phone number and returns an empty 200 response on success.
Method: POST

POST /phone/verify-otp
Description: Verifies a one-time password (OTP) and completes phone-based sign-in.
Method: POST

OAuthController (/api/oauth)
GET|POST /api/oauth/google
Description: Returns a JSON object containing the Google OAuth sign-in URL.
Method: GET, POST

GET /api/oauth/callback?code={code}
Description: Handles the OAuth callback, exchanges the code for tokens, and returns an AuthResponse.
Method: GET

GET /api/oauth/success?token={token}&refresh_token={refresh_token}[&expires_in={expiresIn}]
Description: Redirects the browser to the front-end success page with the issued tokens.
Method: GET

GET|POST /api/oauth/google/redirect
Description: Redirects the user's browser to the Google OAuth consent screen.
Method: GET, POST

SupabaseAuthController (/api/auth)
POST /api/auth/signup

Description: Creates a new user account with email and password, returning an AuthResponse.
Method: POST

POST /api/auth/signin
Description: Authenticates a user by email and password and returns an AuthResponse.
Method: POST

POST /api/auth/signout
Description: Signs out the user identified by the bearer token in the Authorization header.
Method: POST

GET /api/auth/validate
Description: Validates the session token from the Authorization header and returns session status.
Method: GET

POST /api/auth/reset-password
Description: Sends a password reset email to the address in ResetPasswordRequest.
Method: POST

NotificationController (/api/notifications)
POST /api/notifications/send
Description: Sends a new notification using the provided NotificationRequest in the request body.
Method: POST

GET /api/notifications/user/{userId}
Description: Retrieves all notifications associated with the user identified by the given userId.
Method: GET

POST /api/notifications/command/{action}/{id}
Description: Executes an action (e.g., mark as read, delete) on a notification with the specified id.
Method: POST

SearchController (/api/search)
GET /api/search
Description: Searches for events by optional parameters such as keyword, location, and date range (fromDate, toDate).
Method: GET

BookingController Endpoints

`POST /bookings?roomType={roomType}&nights={nights}`
Attempts to book a room of the specified type for the given number of nights by consulting the availability client.
Returns the generated booking ID on success or HTTP 400 if availability check fails.

`POST /bookings/events/{eventId}?userId={userId}`
Creates a new booking for the specified event and user, returning a JSON object with `bookingId`, `eventId`, `userId`, and `status`.
Returns HTTP 200 on success or HTTP 400 with an error message on failure.

`DELETE /bookings/{bookingId}`
Cancels the booking identified by `bookingId` and returns HTTP 204 No Content on success.
Always returns HTTP 204, even if the booking did not previously exist.

`POST /bookings/{bookingId}/refund`
Processes a refund for the booking identified by `bookingId`.
Returns HTTP 200 OK on success.

`POST /bookings/events/payment?userId={userId}&eventId={eventId}`
Processes payment for the specified user and event, returning a JSON object with `success` and `message`.
Returns HTTP 200 on success or HTTP 500 with an error message on failure.

`POST /bookings/events/refund?userId={userId}&eventId={eventId}`
Processes a refund for the specified user and event, returning a JSON object with `success` and `message`.
Returns HTTP 200 on success or HTTP 500 with an error message on failure.

`CategoryController`:

---

* **GET `/api/categories`** – Fetch all categories.
  Response: List of `Category` objects.

* **GET `/api/categories/{id}`** – Fetch a category by its ID.
  Response: A single `Category` object.

* **POST `/api/categories`** – Create a new category.
  Request: `Category` object in the body, **Response:** Created `Category`.

* **PUT `/api/categories/{id}`** – Update an existing category.
  Request: Updated `Category` in body, **Response:** Updated `Category`.

* **DELETE `/api/categories/{id}`** – Delete a category by its ID.
  Response: Confirmation map with `deleted = true`.

`EventController`:

---

* **GET `/api/events`** – Fetch all events with pagination and sorting.
  Query Params: `page`, `size`, `sortBy`, `direction`
  Response: Paginated list of `Event` objects with metadata.

* **GET `/api/events/{id}`** – Fetch an event by its ID.
  Response: Single `Event` object.

* **POST `/api/events`** – Create a new event.
  Request: `Event` object in body, optional `categories` as query params
  Response: Created `Event`.

* **PUT `/api/events/{id}`** – Update an existing event.
  Request: Updated `Event` object in body, optional `categories` as query params
  Response: Updated `Event`.

* **DELETE `/api/events/{id}`** – Delete an event by its ID.
  Response: Confirmation map with `deleted = true`.

* **GET `/api/events/organizer/{organizerId}`** – Get events by organizer with pagination.
  Query Params: `page`, `size`
  Response: Paginated list of events.

* **GET `/api/events/public`** – Get all public events with pagination.
  Query Params: `page`, `size`
  Response: Paginated list of public events.

* **GET `/api/events/upcoming`** – Get upcoming events with pagination.
  Query Params: `page`, `size`
  Response: Paginated list of upcoming events.

* **GET `/api/events/category/{categoryId}`** – Get events by category with pagination.
  Query Params: `page`, `size`
  Response: Paginated list of events in that category.

* **GET `/api/events/search`** – Search events by keyword with pagination.
  Query Params: `keyword`, `page`, `size`
  Response: Paginated list of matching events.

* **PATCH `/api/events/{id}/publish`** – Publish an event by ID.
  Response: Updated `Event`.

* **PATCH `/api/events/{id}/unpublish`** – Unpublish an event by ID.
  Response: Updated `Event`.

* **PATCH `/api/events/{id}/type`** – Change event type.
  Query Params: `eventType` (enum)
  Response: Updated `Event`.

* **POST `/api/events/{id}/signup`** – Sign up a user to an event.
  Query Params: `userId`
  Response: Updated `Event` or error message.

* **GET `/api/events/participant/{userId}`** – Get events a user is participating in with pagination.
  Query Params: `page`, `size`

  Response: Paginated list of events.

* **GET `/api/events/user/{userId}`** – Get all events related to a user (organizer & participant) with pagination.
  Query Params: `page`, `size`
  Response: Paginated list of events.

* **GET `/api/events/available`** – Get events with available tickets with pagination.
  Query Params: `page`, `size`
  Response: Paginated list of events.

* **GET `/api/events/{id}/available-tickets`** – Get available tickets count for a specific event.
  Response: Map with `eventId` and `availableTickets`.

* **PUT `/api/events/{id}/available-tickets`** – Adjust available tickets count by delta.
  Query Params: `delta` (int)
  Response: No content (status 200).

* **POST `/api/events/{id}/book`** – Book tickets for an event.
  Query Params: `userId`, `numberOfTickets` (default 1)
  Response: Booking details or error message.

* **POST `/api/events/{eventId}/participants/{userId}`** – Add participant to event.
  Response: Success message or error.

* **DELETE `/api/events/{eventId}/participants/{userId}`** – Remove participant from event.
  Response: Success message or error.

* **GET `/api/events/{eventId}/participants/{userId}`** – Check if user is registered for event.
  Response: Map with `isRegistered` boolean or error.