**API Documentation for Speaker Session Booking Platform**

**Vimal M | RA2211028010217 | vm5470@srmist.edu.in**

**Introduction**

Welcome to the API documentation for the Speaker Session Booking Platform. This API enables users to create profiles, browse available speakers, book sessions, and receive notifications via email and Google Calendar invites. The API is built using Node.js and Express.js, with a SQL database for data persistence.

**Endpoints**

**1. User Signup**

* **URL:** /user/signup
* **Method:** POST
* **Description:** Register a new user or speaker.
* **Parameters:**
  + firstName (string)
  + lastName (string)
  + email (string)
  + password (string)
  + userType (string: "user" or "speaker")
* **Response:**
  + HTTP 201 Created
  + Body: User object with OTP for verification.
* **Notes:** User must verify email via OTP before account activation.

**2. User Login**

* **URL:** /user/login
* **Method:** POST
* **Description:** Authenticate a user or speaker and obtain JWT token.
* **Parameters:**
  + email (string)
  + password (string)
* **Response:**
  + HTTP 200 OK
  + Body: JWT token.
* **Notes:** Token must be included in Authorization header for protected routes.

**3. Verify User Email**

* **URL:** /user/verify
* **Method:** POST
* **Description:** Verify user account with OTP.
* **Parameters:**
  + email (string)
  + otp (string)
* **Response:**
  + HTTP 200 OK
  + Body: Verification success message.

**4. Create Speaker Profile**

* **URL:** /protected/createspeakerprofile
* **Method:** POST
* **Description:** Speakers create their profile with expertise and session price.
* **Parameters:**
  + expertise (string)
  + pricePerSession (number)
* **Response:**
  + HTTP 200 OK
  + Body: Speaker profile details.
* **Notes:** Only speakers can access this route.

**5. List Speakers**

* **URL:** /public/speakers
* **Method:** GET
* **Description:** Retrieve list of available speakers.
* **Response:**
  + HTTP 200 OK
  + Body: Array of speaker profiles.

**6. Book a Session**

* **URL:** /protected/book
* **Method:** POST
* **Description:** Book a session with a speaker.
* **Parameters:**
  + speakerId (string)
  + date (date)
  + timeSlot (string: e.g., "10:00 AM - 11:00 AM")
* **Response:**
  + HTTP 200 OK
  + Body: Booking confirmation details.
* **Notes:** Time slot is blocked for the selected date.

**7. Delete User Account**

* **URL:** /user/deleteuser
* **Method:** DELETE
* **Description:** Delete a user account.
* **Parameters:**
  + email (string)
* **Response:**
  + HTTP 200 OK
  + Body: Deletion success message.

**Models**

* **User:**
  + id (UUID)
  + firstName (string)
  + lastName (string)
  + email (string)
  + password (string)
  + userType (string: "user" or "speaker")
  + isVerified (boolean)
  + otp (string)
  + otpExpiration (date)
* **SpeakerProfile:**
  + id (UUID)
  + userId (UUID, foreign key to User)
  + expertise (string)
  + pricePerSession (decimal)
* **Booking:**
  + id (UUID)
  + userId (UUID, foreign key to User)
  + speakerId (UUID, foreign key to SpeakerProfile)
  + date (date)
  + timeSlot (string)

**Authentication**

* **JWT Token:**
  + Obtained from /user/login endpoint.
  + Include token in Authorization header as Bearer <token>.
  + Token expires in 1 hour.
* **Role-Based Access Control:**
  + Middleware checks userType in JWT payload.
  + Certain routes are protected for specific roles.

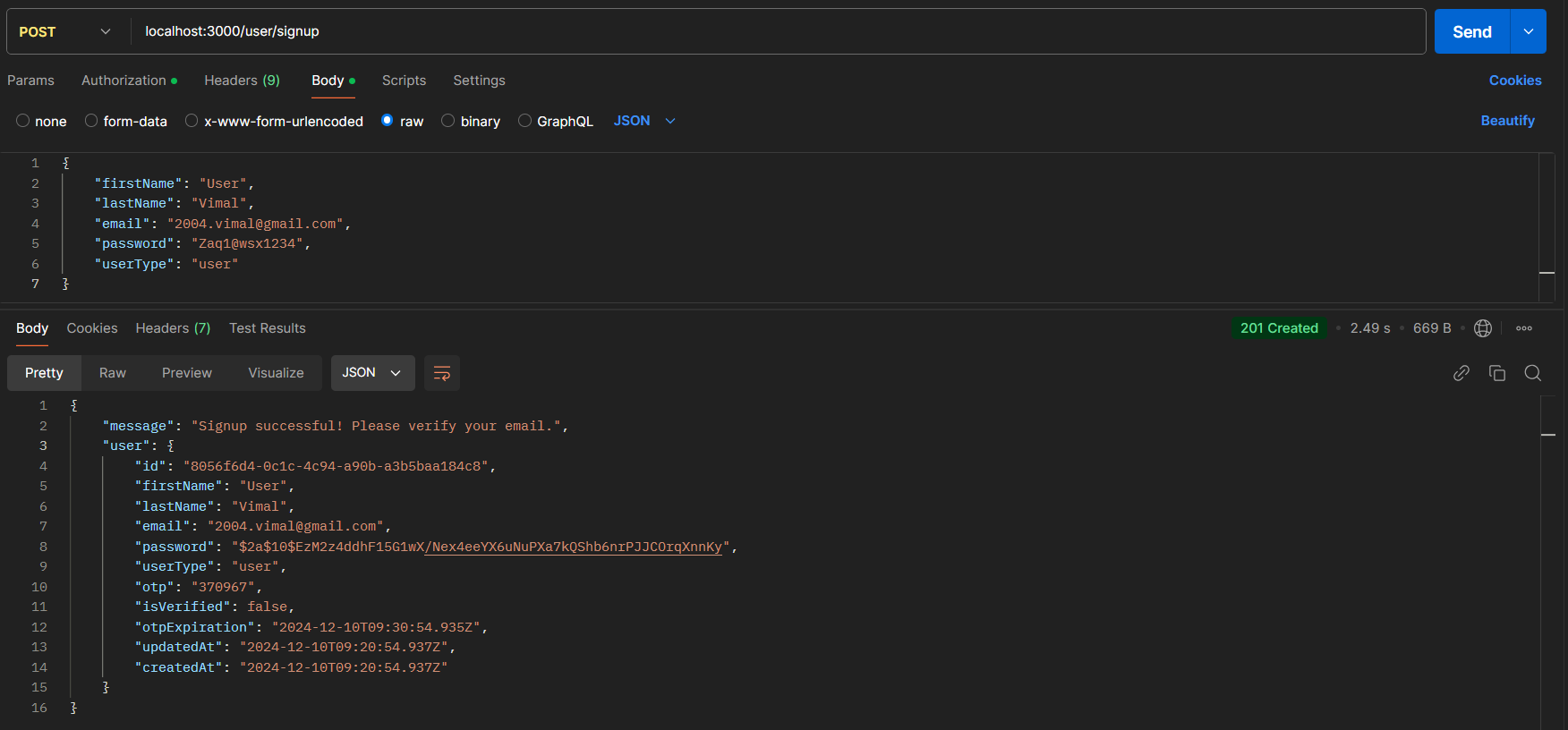
**Error Handling**

* **400 Bad Request:** Invalid input parameters.
* **401 Unauthorized:** Missing or invalid JWT token.
* **403 Forbidden:** Insufficient permissions.
* **404 Not Found:** Resource not found.
* **500 Internal Server Error:** Server-side error.

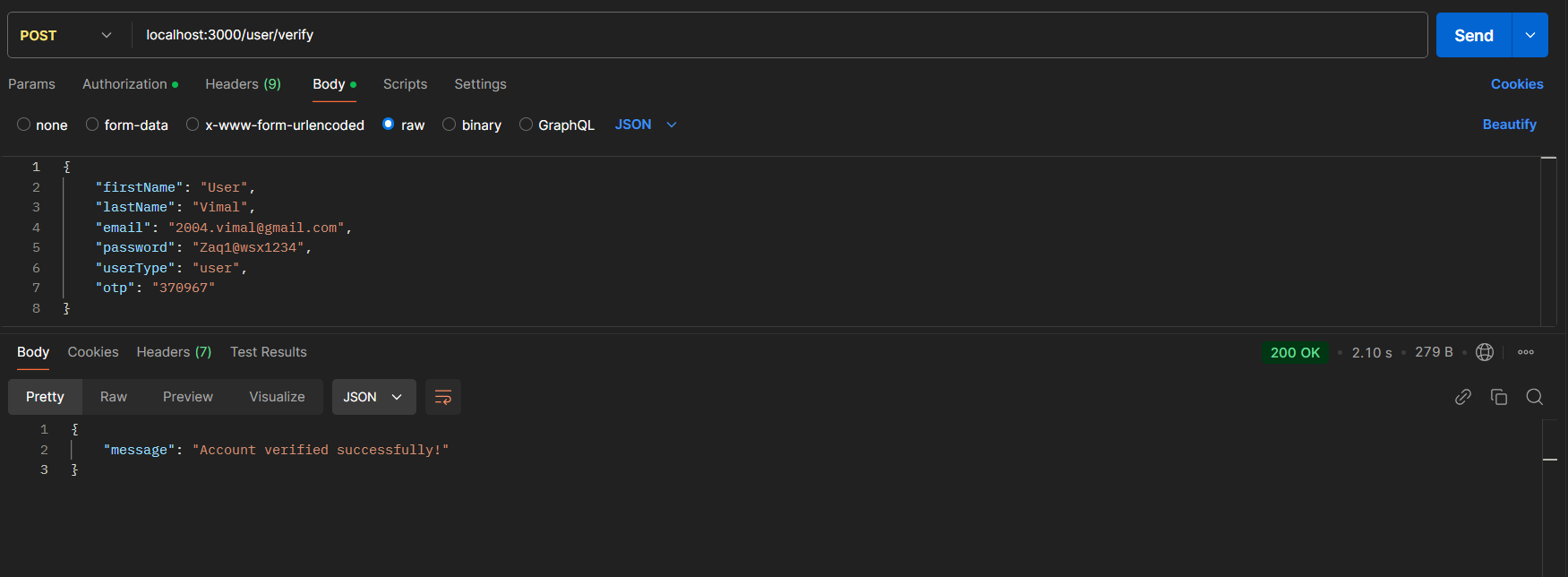
**Appendices**

**Sample Request and Response**

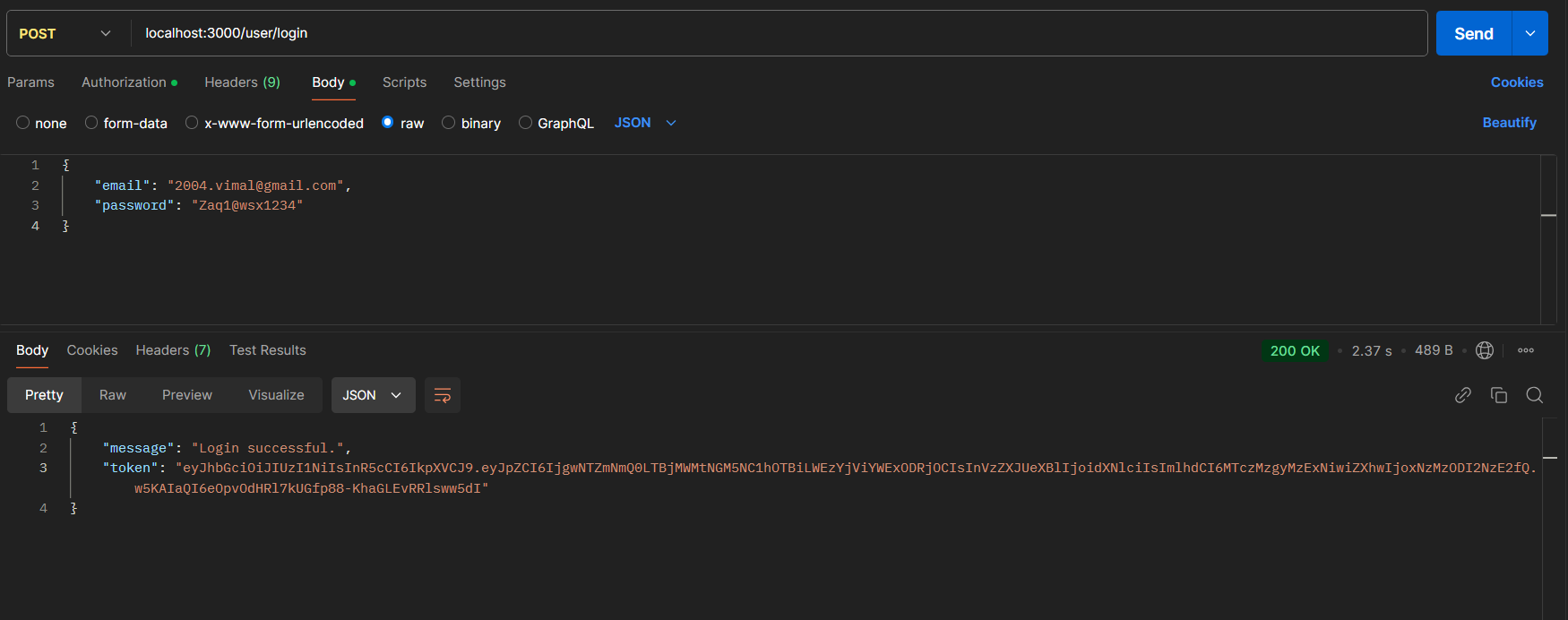
1. **User Signup:**



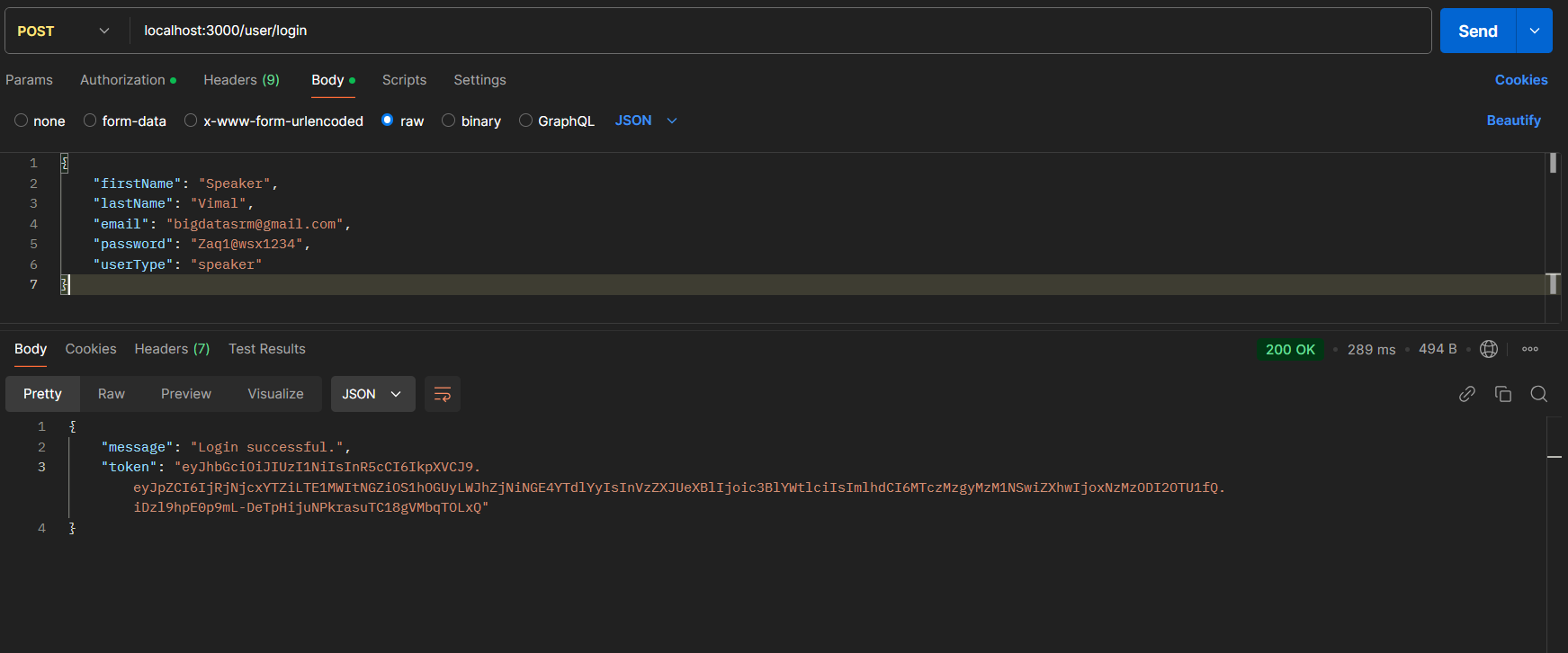
1. **User Verification**



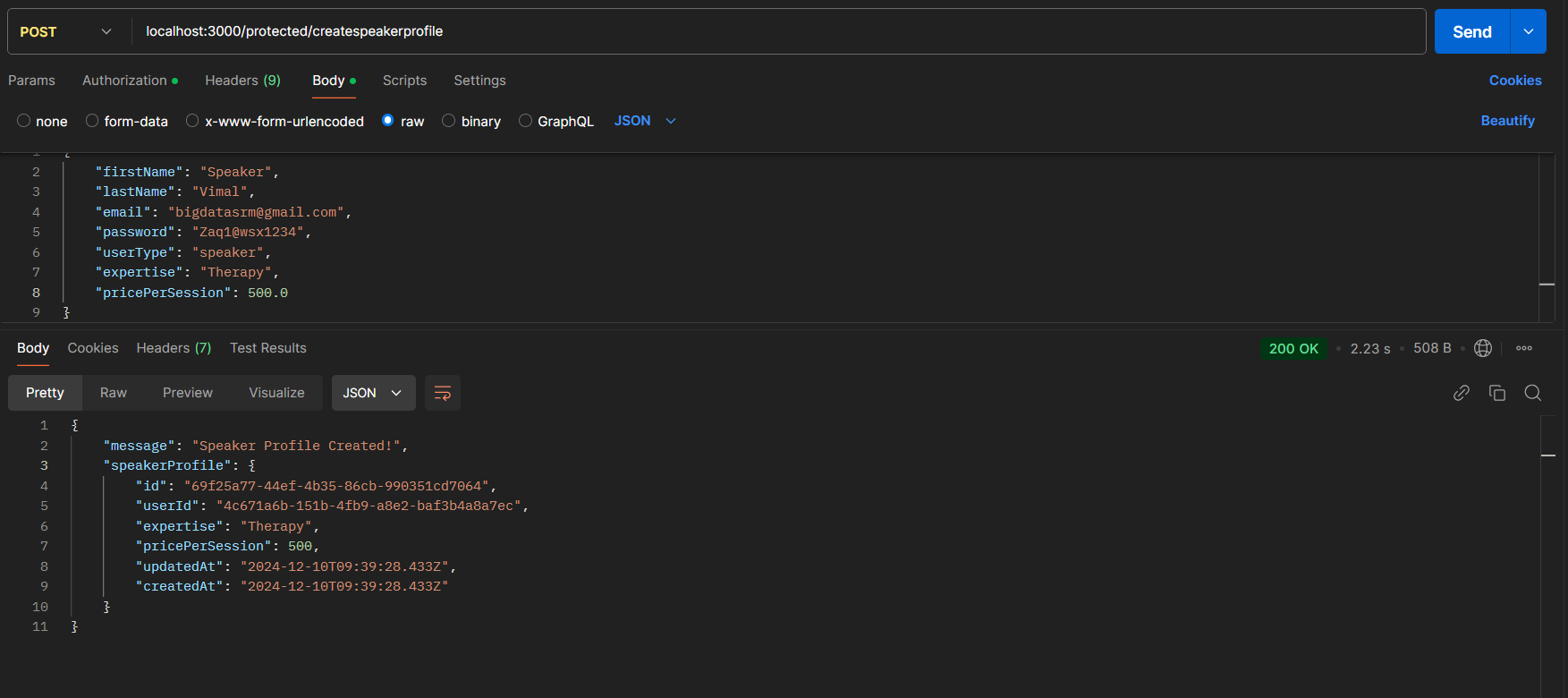
1. **User Login**



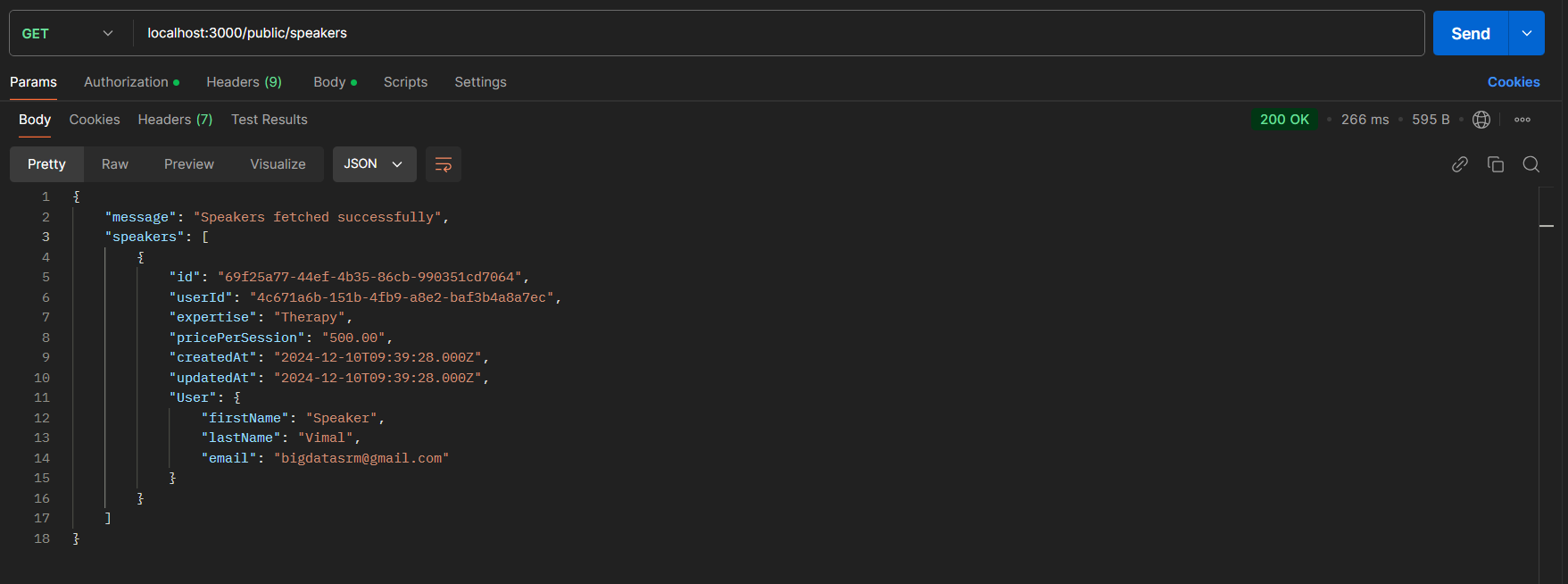
1. **Creating Speaker Profile**



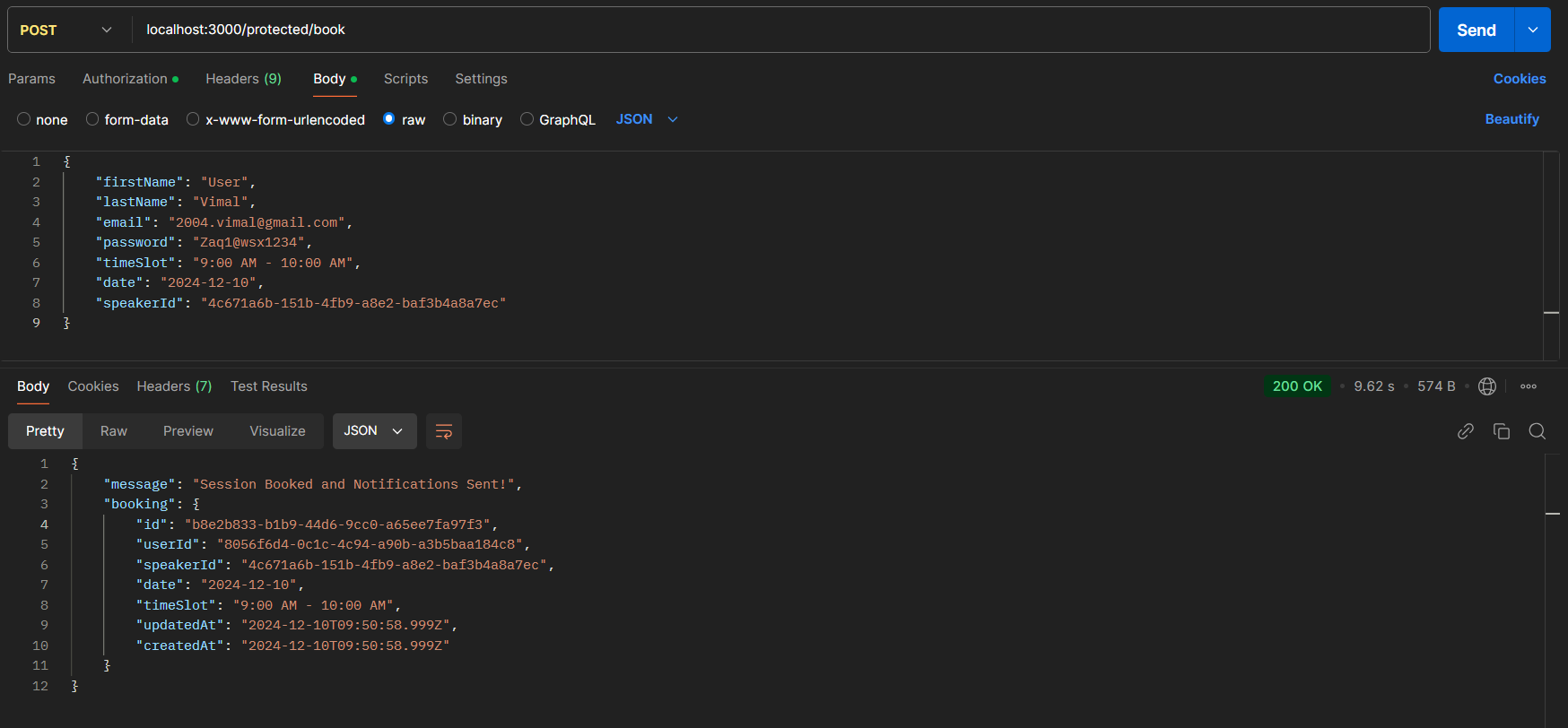
**Use Authorization token provided during login to access the protected route “/protected/createspeakerprofile”**



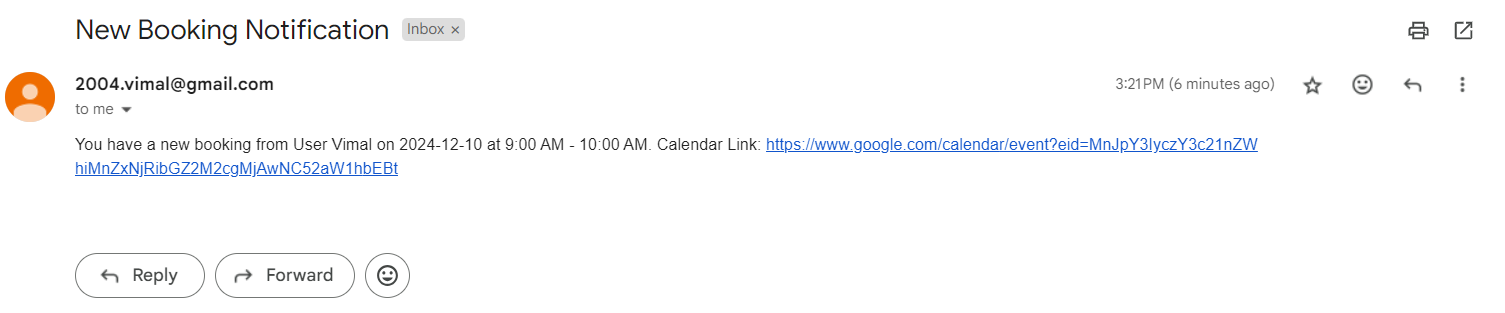
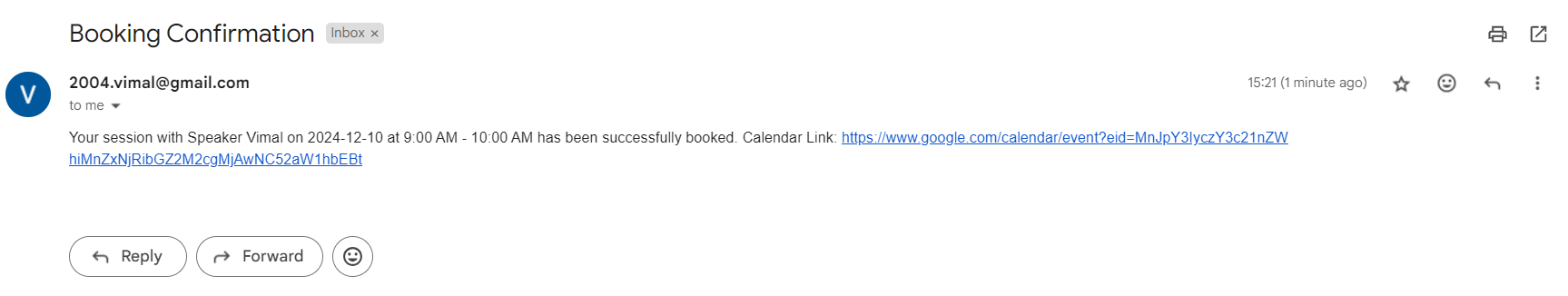
1. **Public route to get all speaker profiles**

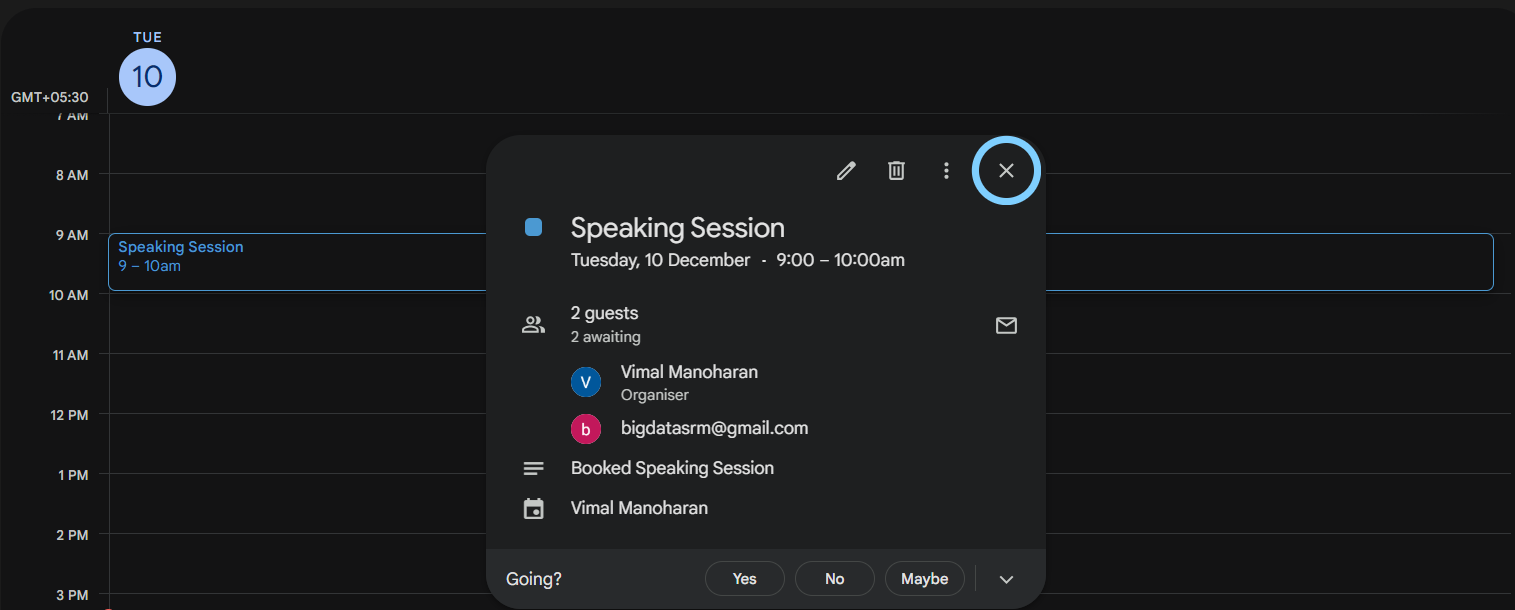


1. **Booking a slot with a speaker. Protected route which only users can access.**



Emails and Google Calender Events are set for both user and speaker





**API Documentation for Speaker Session Booking Platform**

**Table of Contents**

1. Introduction
2. Endpoints
   * 2.1 User Signup
   * 2.2 User Login
   * 2.3 Verify User Email
   * 2.4 Create Speaker Profile
   * 2.5 List Speakers
   * 2.6 Book a Session
   * 2.7 Delete User Account
3. Models
   * 3.1 User Model
   * 3.2 SpeakerProfile Model
   * 3.3 Booking Model
4. Authentication
   * 4.1 JWT Token Management
   * 4.2 Role-Based Access Control
5. Error Handling
   * 5.1 Error Codes and Messages
6. Security Best Practices
   * 6.1 Input Validation
   * 6.2 Data Encryption
7. Performance and Scalability
   * 7.1 API Performance Considerations
   * 7.2 Scalability Plans
8. Appendices
   * 8.1 Sample Requests and Responses
   * 8.2 Error Response Examples

1. Introduction

Welcome to the API documentation for the Speaker Session Booking Platform. This API enables users to create profiles, browse available speakers, book sessions, and receive notifications via email and Google Calendar invites. The API is built using Node.js and Express.js, with a SQL database for data persistence.

2. Endpoints

2.1 User Signup

* URL: /user/signup
* Method: POST
* Description: Register a new user or speaker.
* Parameters:
  + firstName (string)
  + lastName (string)
  + email (string)
  + password (string)
  + userType (string: "user" or "speaker")
* Response:
  + HTTP 201 Created
  + Body: User object with OTP for verification.
* Edge Cases:
  + Email already registered.
  + Password does not meet complexity requirements.
  + Invalid userType.
  + OTP generation or email sending failure.
* Additional Information:
  + OTP validity period.
  + User activation requires email verification.

2.2 User Login

* URL: /user/login
* Method: POST
* Description: Authenticate a user or speaker and obtain JWT token.
* Parameters:
  + email (string)
  + password (string)
* Response:
  + HTTP 200 OK
  + Body: JWT token.
* Edge Cases:
  + Unverified email login attempt.
  + Multiple incorrect password attempts.
  + Token expiration.
* Additional Information:
  + Token expiration time.
  + JWT token format.

2.3 Verify User Email

* URL: /user/verify
* Method: POST
* Description: Verify user account with OTP.
* Parameters:
  + email (string)
  + otp (string)
* Response:
  + HTTP 200 OK
  + Body: Verification success message.
* Edge Cases:
  + Incorrect OTP.
  + Expired OTP.
  + Unknown email.
  + Already verified user.
* Additional Information:
  + OTP validity period.

2.4 Create Speaker Profile

* URL: /protected/createspeakerprofile
* Method: POST
* Description: Speakers create their profile with expertise and session price.
* Parameters:
  + expertise (string)
  + pricePerSession (number)
* Response:
  + HTTP 200 OK
  + Body: Speaker profile details.
* Edge Cases:
  + Non-speaker user access.
  + Existing speaker profile.
  + Invalid expertise or pricePerSession.
* Additional Information:
  + Allowed expertise values.
  + Price currency.

2.5 List Speakers

* URL: /public/speakers
* Method: GET
* Description: Retrieve list of available speakers.
* Response:
  + HTTP 200 OK
  + Body: Array of speaker profiles.
* Edge Cases:
  + No speakers available.
  + Incomplete speaker data.
* Additional Information:
  + Sorting order.
  + Pagination parameters.

2.6 Book a Session

* URL: /protected/book
* Method: POST
* Description: Book a session with a speaker.
* Parameters:
  + speakerId (string)
  + date (date)
  + timeSlot (string)
* Response:
  + HTTP 200 OK
  + Body: Booking confirmation details.
* Edge Cases:
  + Past date booking.
  + Already booked time slot.
  + Invalid speaker availability.
* Additional Information:
  + Available time slots.
  + Conflict handling.

2.7 Delete User Account

* URL: /user/deleteuser
* Method: DELETE
* Description: Delete a user account.
* Parameters:
  + email (string)
* Response:
  + HTTP 200 OK
  + Body: Deletion success message.
* Edge Cases:
  + Non-existent account.
  + Deleting another user's account.
  + Account with active bookings.
* Additional Information:
  + Deletion policy (permanent or soft delete).

3. Models

3.1 User Model

* id (UUID)
* firstName (string)
* lastName (string)
* email (string)
* password (string)
* userType (string: "user" or "speaker")
* isVerified (boolean)
* otp (string)
* otpExpiration (date)
* Constraints:
  + String length limits.
  + OTP validity.

3.2 SpeakerProfile Model

* id (UUID)
* userId (UUID, foreign key to User)
* expertise (string)
* pricePerSession (decimal)
* Relationships:
  + User-SpeakerProfile relationship.

3.3 Booking Model

* id (UUID)
* userId (UUID, foreign key to User)
* speakerId (UUID, foreign key to SpeakerProfile)
* date (date)
* timeSlot (string)
* Constraints:
  + Valid date and timeSlot formats.

4. Authentication

4.1 JWT Token Management

* Token Expiration: 1 hour.
* Token Refresh: Mechanism for token renewal.
* Security Measures: Token revocation and blacklisting.

4.2 Role-Based Access Control

* UserTypes: "user", "speaker".
* Access Control Examples:
  + Routes accessible by "user" or "speaker".

5. Error Handling

5.1 Error Codes and Messages

* 400 Bad Request: Invalid input parameters.
* 401 Unauthorized: Missing or invalid JWT token.
* 403 Forbidden: Insufficient permissions.
* 404 Not Found: Resource not found.
* 500 Internal Server Error: Server-side error.
* Specific Error Messages: Detailed error responses for each scenario.

6. Appendices

Sample Requests and Responses