**API Documentation for Speaker Session Booking Platform**

**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 - User Signup:**

{

"firstName": "John",

"lastName": "Doe",

"email": "john.doe@example.com",

"password": "password123",

"userType": "user"

}

**Sample Response - User Login:**

{

"message": "Login successful.",

"token": "eyJhbGciOiJIUzI1NiIs..."

}