**Bibendo API Documentation**

**1. General API Structure**

* Basic settings:
* Host: The API runs on serious-gaming-platform.appspot.com
* BasePath: All endpoints start with /api
* Schemes: HTTPS is used
* Content-Type: The API sends and receives JSON data

**2. Endpoints and Functionalities**

* Various Categories: The API is divided into different functional blocks, such as:
* Account management: Endpoints for retrieving, creating, updating, and deleting user accounts. Examples include /account/accountDetails, /account/create and /account/{fullId}.
* Game management: Endpoints for creating, modifying, cloning and deleting games. Think of /game/create, /game/clone/{gameId} and /game/{gameId}.
* Organization management: Endpoints for creating and managing organizations via /organization/create and /organization/{id}.
* Player management and Contacts: Endpoints to add or remove contacts, and manage invitations, such as /player/add and /player/myContacts.
* Media and Libraries: Endpoints for uploading, deleting and searching for media files in a media library via /media and /media/list.
* HTTP methods: The API uses standard RESTful methods:
* GET: For retrieving data
* POST: For creating or editing data
* PUT: For updates
* DELETE: For deleting data

**3. Security with Firebase**

* Authentication: All endpoints are secured with Firebase authentication. This means that every request must contain a valid Firebase token in the Authorization header (for example, Bearer <token>). This token ensures that only authenticated users can access the API.
* Audience Validation: The API checks if the token belongs to the correct "audience" (in your case "serious-gaming-platform"), which provides additional security.

**4. Data Models and Definitions**

* Definitions Section: In the "definitions" section, you will find the data models that describe the structure of the data:
* Account: Describes account level, email, name, organization ID, etc.
* Game: Describes the properties of a game, such as game ID, name, and other related data.
* Other models: Such as GameTheme, MediaLibraryFile, Response and more, each supporting a specific part of the API's functionality.

**5. How Everything Works Together**

* Interaction via Endpoints: When a client (for example, a mobile app or web application) sends a request, it is handled via one of the endpoints. The request contains the necessary parameters (in the URL, query or body) and the Firebase token for authentication.
* Response Structure: Each successful request yields a response with a status code (e.g., 200 for OK) and a JSON object that complies with the relevant data model. Error messages and codes are also defined, so the client knows what's going wrong if something fails.

**6. Testing and Documentation**

* Testing with Tools: As discussed earlier, you can use tools like Swagger UI, Postman or cURL to test the API. This allows you to see what the actual responses are and collect examples for your documentation.
* Documentation: Your documentation can include this structure and the test results as examples. This makes it clear to users how they should call the API and what they can expect as a response.

**Summary**

Your Bibendo API is a comprehensive, RESTful API covering a wide range of functionalities, from user management to game management and media management. The API is secured via Firebase, and the complete structure and data formats are defined in the Swagger specification. By testing the API with tools like Swagger UI or Postman, you can collect concrete examples that help you write clear and practical documentation.

**Test the Endpoints**

* Use Postman or cURL: Make GET requests to endpoints that provide data. This helps you see which JSON structures actually come from the API. For example, test the endpoint /account/accountDetails to see which fields are in the response.
* Collect Examples: Save the responses or copy them into a document. This way, you can easily reference the exact data that the API returns later.

**Automate the Overview**

* Script for Data Extraction: You can write a script (for example in Python) that:
* Reads the Swagger JSON.
* Reads all endpoints and associated parameters and response models.
* Generates an overview, for instance in the form of a table showing which endpoint provides what data.
* Example: In the previously shared script, you can adjust the output so that it logs the response structure (or a sample response) per endpoint.

**Document the Overview**

* Create a Table or Diagram: Put all endpoints in an overview (for example in a spreadsheet or a diagram). For each endpoint, indicate:
* The HTTP method (GET, POST, etc.)
* The expected parameters (path, query, body)
* The data model of the response (e.g., Account, Game)
* Use Visual Aids: Tools such as Lucidchart or even PowerPoint can help create visual flowcharts showing how data flows between the different endpoints.

**Skillsdojo**

**1. Basic Information**

* Swagger (OpenAPI) version: 2.0
* Host: ${endpoints.project.id}.appspot.com (placeholder for the eventual host, often skillsdojo.appspot.com)
* BasePath: /api
* Schemes: https
* Security: Most endpoints are secured via Firebase OAuth2 (securityDefinitions and security with firebase).

**2. Endpoints and Their Purpose**

The paths section in Swagger contains all available endpoints. A selection of the main groups:

1. Account Management

* /account/create, /account/create/demo: Create a new account, optionally as a demo account.
* /account/update, /account/update/{uid}: Update account information.
* /account/resetpw/{email}, /account/setPw/{token}: Reset or set password.
* /account/admin/{uid} and /admin/account/{uid}: Administrator actions (e.g., granting admin rights or deleting an account).

1. Asset Management (Assets are "mission elements" such as texts, images, assignments, etc.)

* /asset/create/...: Endpoints to create different types of "assets" (e.g., instructions, images, comments).
* /asset/update/{assetId}: Update existing assets.
* /asset/{assetId}: Retrieve a specific asset.
* /asset/{missionId}/{assetId}/delete: Delete an asset within a specific mission.

1. Community and Missions

* /community/mission/{missionId}/like: "Like" a mission (and delete to "unlike").
* /mission/create, /mission/{missionId}: Create, retrieve or delete a mission.
* /mission/clone/{missionId}: Clone a mission.
* /mission/update/{missionId}: Update mission information.
* /mission/{missionId}/asset/{assetId}/{visible}: Change the visibility of an asset within a mission.

1. Subscription Management

* /subscription/create/{uid}: Link a new subscription to a user.
* /subscription/{subscriptionId}/user/email/{email}: Add a user to a subscription.
* /subscription/{subscriptionId}/user/{uid}: Remove a user from a subscription.

1. Themes

* /theme/{themeId}: Create or delete new themes. (For example, a CSS style or color scheme.)

1. Various Other Endpoints

* /wakeup: Appears to be a simple "ping" or "wake up" endpoint (useful to prevent the app from going to sleep).

**3. Data Models (Definitions)**

In the definitions section, you'll find the structure of all objects used. Some examples:

* Account
* Contains e.g., email, uid, subscriptionId and flags such as demo or introduction.
* Uses Timestamp objects for accountCreationDate and lastLoginDate.
* Mission
* A mission with fields such as title, description, duration, featured, tags and assets.
* Can have various properties describing the content or status of the mission.
* AbstractAsset and derived types (e.g., ImageAsset, InstructionAsset, CommentAsset, etc.)
* These describe different "components" of a mission, such as an instruction text or image.
* Many of them share the same basic structure (a title, type, id, missions, etc.).
* Subscription
* Describes e.g., id, maxSeats, owner and a list of seats.
* Theme
* An object with e.g., color, css, id and name.

**4. Security via Firebase**

This means that a valid Firebase ID token is required to use the endpoint. You send this token in the Authorization header (e.g., Bearer <JWT>).

**5. How Does It Work in Practice?**

* User Management (Accounts): You can create and manage new users (accounts). There are also functions to create demo accounts or reset passwords.
* Mission Management: Using endpoints like /mission/create and /mission/update/{missionId}, you can set up new missions or update existing ones. By using /asset/create/..., you add specific components (assets) to a mission.
* Subscriptions: You can manage subscriptions, add or remove users, and change ownership.
* Administrator Actions: There are separate admin endpoints (e.g., /admin/account/{uid}) that allow administrators to delete accounts or set init status.

**Summary**

* API Type: RESTful, described with Swagger/OpenAPI 2.0.
* Functionality: Focuses on user management, missions (with associated assets) and subscription management.
* Security: Based on Firebase (requires a valid ID token).
* Data Structure: Consists of different models (Mission, Asset, Subscription, etc.) that together form the core of the platform.