

# Playtime Swagger



Full Stack Web Development

# hapi-swagger

- A plugin for HAPI
- Leverages Hapi Routes + Joi Schemas to automatically generate Swagger / Open API 2.0 Specification

The screenshot shows a GitHub repository page for `glennjones/hapi-swagger`. The repository has 26 watchers, 405 forks, and 877 stars. The `Code` tab is selected, showing the `usageguide.md` file. The file was last updated by `mrjono1` on May 15, 2021. It has 624 lines (505 sloc) and is 21.8 KB in size. The file content is titled "Usage Guide" and includes a "Content" section with a list of topics:

- JSON body
- Form body
- Params query and headers
- Naming
- Grouping endpoints by path or tags
- Extending group information with tag objects
- Ordering the endpoints within groups
- Rewriting paths and groupings
- Response Object
- Status Codes
- Caching
- File upload
- Prevent JOI properties from being included in the Swagger schema
- Headers and `.unknown()`
- Additional Hapi data using `x-*`
- JSON without UI
- Simplifying the JSON
- Debugging
- Features from Hapi that cannot be ported to Swagger

# Install Plugin

- Install
- Configure
- Initialise

```
npm install hapi-swagger
```

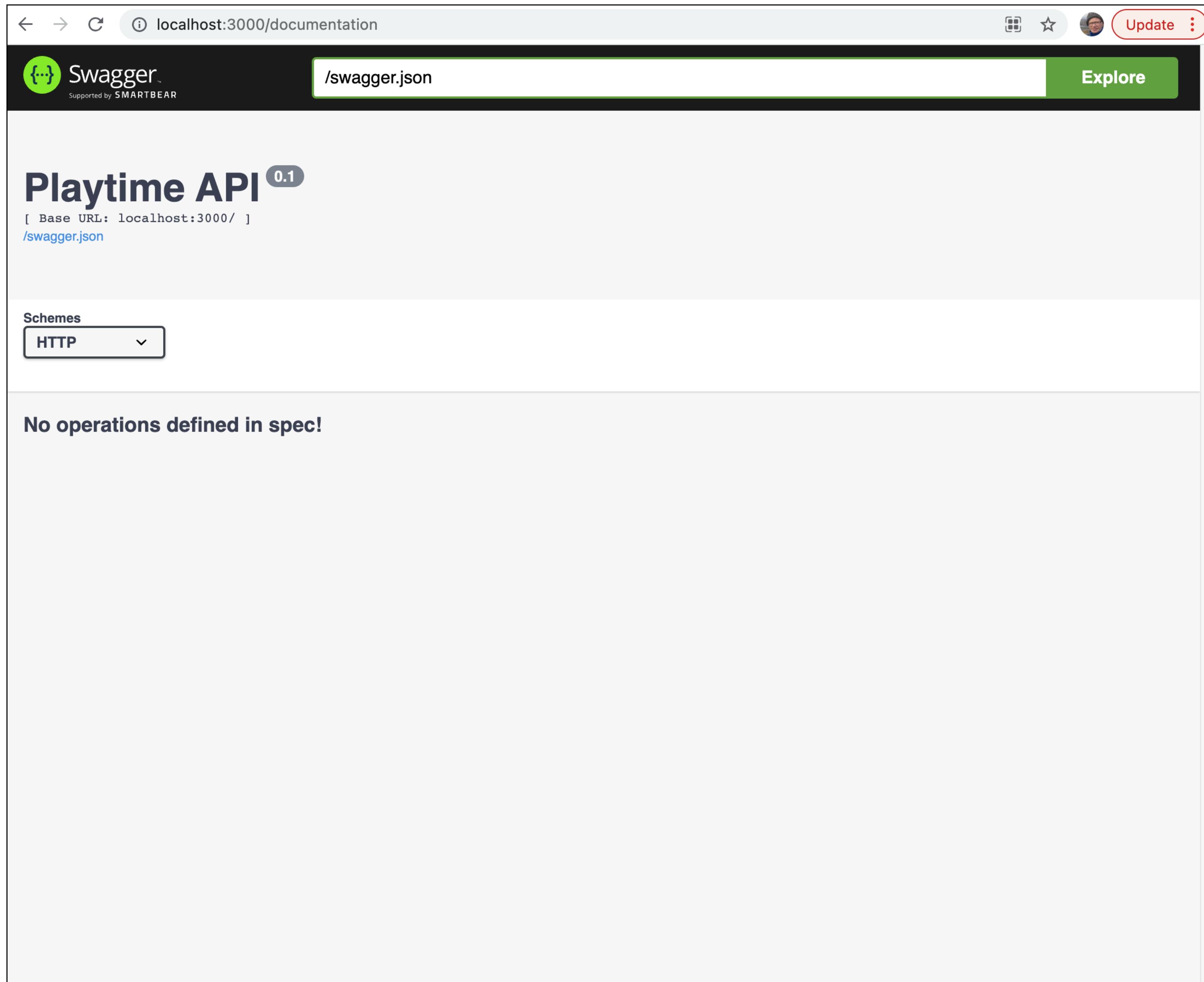
```
import HapiSwagger from "hapi-swagger";

const swaggerOptions = {
  info: {
    title: "Playtime API",
    version: "0.1",
  },
};
```

```
await server.register([
  Inert,
  Vision,
  {
    plugin: HapiSwagger,
    options: swaggerOptions,
  },
]);
```

# API Hub Site

- Plugin generates a Web Site Compatible with the SwaggerHub
- Web site hosts
  - Generated Spec
  - Forms for exploring the API
  - API Invocation workbench



# API Controller Annotations

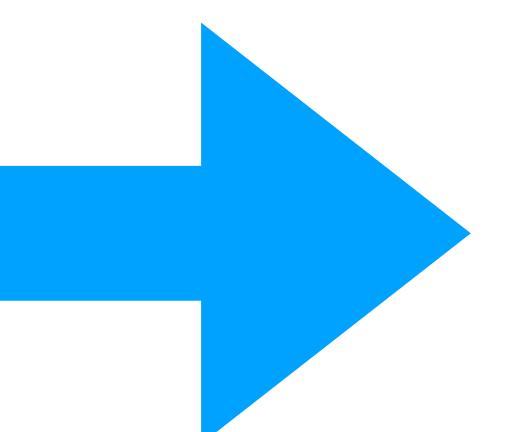
- Directly in the api controllers, specify:
  - tag
  - Description
  - Notes

## user-api.js

```
export const userApi = {  
  find: {  
    auth: false,  
    handler: async function(request, h) {  
      try {  
        const users = await db.userStore.getAllUsers();  
        return users;  
      } catch (err) {  
        return Boom.serverUnavailable("Database Error");  
      }  
    },  
    tags: ["api"],  
    description: "Get all userApi",  
    notes: "Returns details of all userApi",  
  },
```

## user-api.js

```
export const userApi = {  
  find: {  
    auth: false,  
    handler: async function(request, h) {  
      try {  
        const users = await db.userStore.getAllUsers();  
        return users;  
      } catch (err) {  
        return Boom.serverUnavailable("Database Error");  
      }  
    },  
    tags: ["api"],  
    description: "Get all userApi",  
    notes: "Returns details of all userApi",  
  },
```

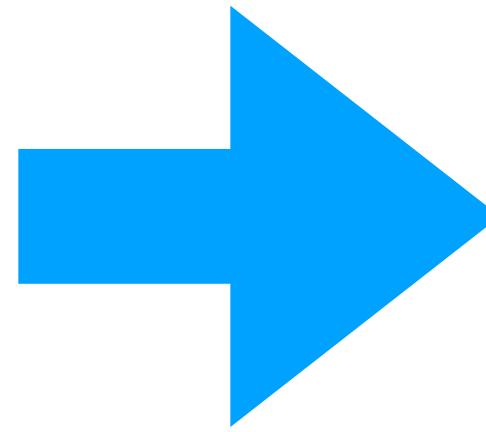


- Plug in generates API specification

The screenshot shows a web browser window displaying the Playtime API documentation. The title bar says "Playtime API" and the address bar shows "localhost:3000/documentation#/api/getApiUsers". The main content area is titled "Playtime API 0.1" with the base URL "localhost:3000/". It features a "Swagger" logo and a "Supported by SMARTBEAR" link. A navigation bar includes "Explore" and other icons. Below the title, there's a "Schemes" dropdown set to "HTTP". The main section is titled "api" and contains a "GET /api/users Get all userApi" endpoint. This endpoint is described as "Returns details of all userApi". It has a "Parameters" section stating "No parameters" and a "Responses" section. The "Responses" section shows a single entry for "default" with the status code "Successful" and an example value of "string". A "Try it out" button is located in the top right of the response section.

- Response from GET /api/users

# Execute



The screenshot shows a web browser window titled "Playtime API" with the URL "localhost:3000/documentation#/api/getApiUsers". The page displays a "GET /api/users" endpoint for "Get all userApi". It describes the endpoint as returning details of all userApi and notes that there are no parameters. Below this, there are two buttons: "Execute" (highlighted in blue) and "Clear". The "Responses" section shows a "Curl" command and a "Request URL" (http://localhost:3000/api/users). Under "Server response", a "Code" dropdown is set to "200" and the "Details" tab is selected. The response body is displayed as a JSON array of three objects, each representing a user with fields like \_id, firstName, lastName, email, password, and \_\_v. At the bottom, there are "Response headers" and "Download" buttons.

```
curl -X 'GET' \
  'http://localhost:3000/api/users' \
  -H 'accept: application/json'
```

Request URL

http://localhost:3000/api/users

Server response

Code Details

200 Response body

```
[{"_id": "6200cf9478dd0e38b9216de0", "firstName": "Homer", "lastName": "Simpson", "email": "homer@simpson.com", "password": "secret", "__v": 0}, {"_id": "6200cf9478dd0e38b9216de3", "firstName": "Marge", "lastName": "Simpson", "email": "marge@simpson.com", "password": "secret", "__v": 0}, {"_id": "6200cf9478dd0e38b9216de6", "firstName": "Bart", "lastName": "Simpson", "email": "bart@simpson.com", "password": "secret", "__v": 0}]
```

Response headers

Download

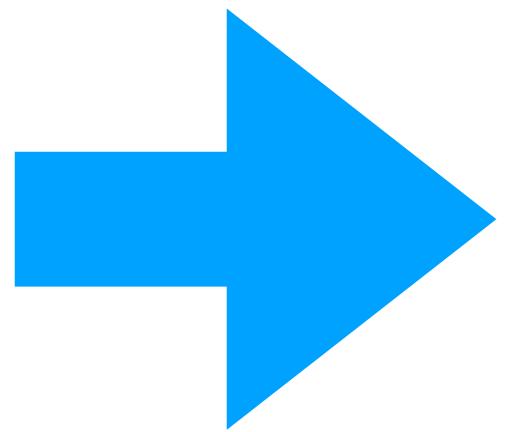
Swagger  
Supported by SMARTBEAR

/swagger.json

# Playtime API 0.1

[ Base URL: localhost:3000 / ]

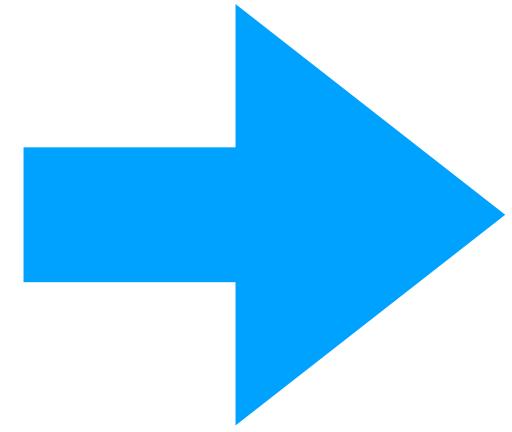
</swagger.json>



- swagger.json available from the generated swagger site

```
{  
  "swagger": "2.0",  
  "host": "localhost:3000",  
  "basePath": "/",  
  "schemes": [  
    "http"  
,  
  "info": {  
    "title": "Playtime API",  
    "version": "0.1"  
,  
  "tags": [  
    ],  
  "paths": {  
    "/api/users": {  
      "get": {  
        "summary": "Get all userApi",  
        "operationId": "getApiUsers",  
        "description": "Returns details of all userApi",  
        "tags": [  
          "api"  
,  
        "responses": {  
          "default": {  
            "schema": {  
              "type": "string"  
,  
            "description": "Successful"  
          }  
        }  
      }  
    },  
  "definitions": {  
    }  
  }  
}
```

```
{  
  "swagger": "2.0",  
  "host": "localhost:3000",  
  "basePath": "/",  
  "schemes": [  
    "http"  
,  
  "info": {  
    "title": "Playtime API",  
    "version": "0.1"  
,  
  "tags": [  
,  
  "paths": {  
    "/api/users": {  
      "get": {  
        "summary": "Get all userApi",  
        "operationId": "getApiUsers",  
        "description": "Returns details of all userApi",  
        "tags": [  
          "api"  
,  
        "responses": {  
          "default": {  
            "schema": {  
              "type": "string"  
,  
            "description": "Successful"  
          }  
        }  
      }  
    }  
,  
  "definitions": {  
}|
```



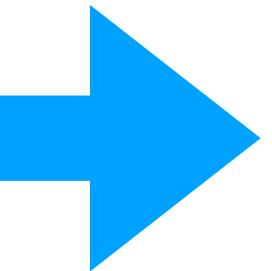
```
swagger: '2.0'  
host: localhost:3000  
basePath: "/"  
schemes:  
- http  
info:  
  title: Playtime API  
  version: '0.1'  
tags: []  
paths:  
  "/api/users":  
    get:  
      summary: Get all userApi  
      operationId: getApiUsers  
      description: Returns details of all userApi  
      tags:  
      - api  
      responses:  
        default:  
          schema:  
            type: string  
            description: Successful  
definitions: {}
```

Yaml version via

<https://www.json2yaml.com/>

# Joi/Swagger

- Current UserSpec Joi Schema employed for form validation.
- Joi has additional attributes, including **example & label**



```
export const UserSpec = {
  firstName: Joi.string().required(),
  lastName: Joi.string().required(),
  email: Joi.string().email().required(),
  password: Joi.string().required(),
};
```

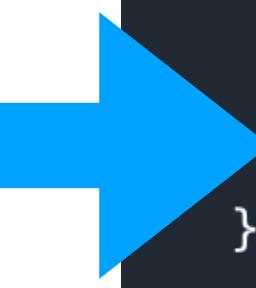
```
export const UserSpec = Joi.object().keys({
  firstName: Joi.string().example("Homer").required(),
  lastName: Joi.string().example("Simpson").required(),
  email: Joi.string().email().example("homer@simpson.com").required(),
  password: Joi.string().example("secret").required(),
}).label("User Details");

export const UserArray = Joi.array().items(UserSpec).label("User Array");
```

# Response Validation

```
export const UserSpec = Joi.object().keys({  
  firstName: Joi.string().example("Homer").required(),  
  lastName: Joi.string().example("Simpson").required(),  
  email: Joi.string().email().example("homer@simpson.com").required(),  
  password: Joi.string().example("secret").required(),  
}).label("User Details");  
  
export const UserArray = Joi.array().items(UserSpec).label("User Array");
```

```
export const userApi = {  
  find: {  
    auth: false,  
    handler: async function(request, h) {  
      try {  
        const users = await db.userStore.getAllUsers();  
        return users;  
      } catch (err) {  
        return Boom.serverUnavailable("Database Error");  
      }  
    },  
    tags: ["api"],  
    description: "Get all userApi",  
    notes: "Returns details of all userApi",  
    response: { schema: UserArray }  
  },  
};
```



- **Response:** Joi validation will be engaged to validate the response

# Model Documentation

The screenshot shows the Playtime API documentation interface. At the top, it displays the title "Playtime API 0.1" and the base URL "[ Base URL: localhost:3000/ ] /swagger.json". Below this, there is a dropdown menu labeled "Schemes" with "HTTP" selected. The main content area is titled "api" and contains a "Models" section. This section lists two models: "UserDetails" and "UserArray".

**UserDetails** (Object)

- firstName\***: string, example: Homer
- lastName\***: string, example: Simpson
- email\***: string, example: homer@simpson.com
- password\***: string, example: secret
- \_id**: string
- \_v**: number

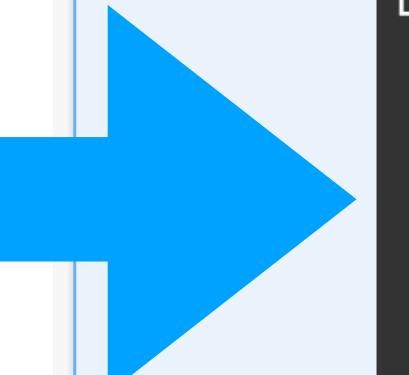
**UserArray** (Array)

- firstName\***: string, example: Homer
- lastName\***: string, example: Simpson
- email\***: string, example: homer@simpson.com
- password\***: string, example: secret
- \_id**: string
- \_v**: number

- Models section generated in documentation

# Response Documentation

- Example successful response generated from Joi schema



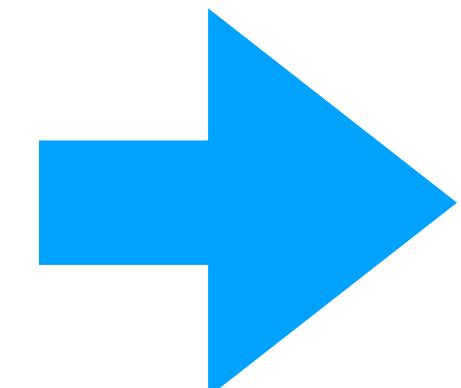
The screenshot shows a detailed API documentation page for a 'GET /api/users' endpoint. The top navigation bar says 'api'. Below it, the method 'GET' and the endpoint '/api/users' are listed, along with the description 'Get all userApi'. A large blue button labeled 'Try it out' is visible on the right. The 'Parameters' section indicates 'No parameters'. In the 'Responses' section, a table lists a single row for status code 200, which is described as 'Successful'. Below this, there is a 'Example Value | Model' link and a JSON code block:

```
[  
  {  
    "firstName": "Homer",  
    "lastName": "Simpson",  
    "email": "homer@simpson.com",  
    "password": "secret",  
    "_id": "string",  
    "__v": 0  
  }  
]
```

# Swagger/OpenAPI Specification

```
export const UserSpec = Joi.object().keys({  
    firstName: Joi.string().example("Homer").required(),  
    lastName: Joi.string().example("Simpson").required(),  
    email: Joi.string().email().example("homer@simpson.com").required(),  
    password: Joi.string().example("secret").required(),  
}).label("User Details");  
  
export const UserArray = Joi.array().items(UserSpec).label("User Array");
```

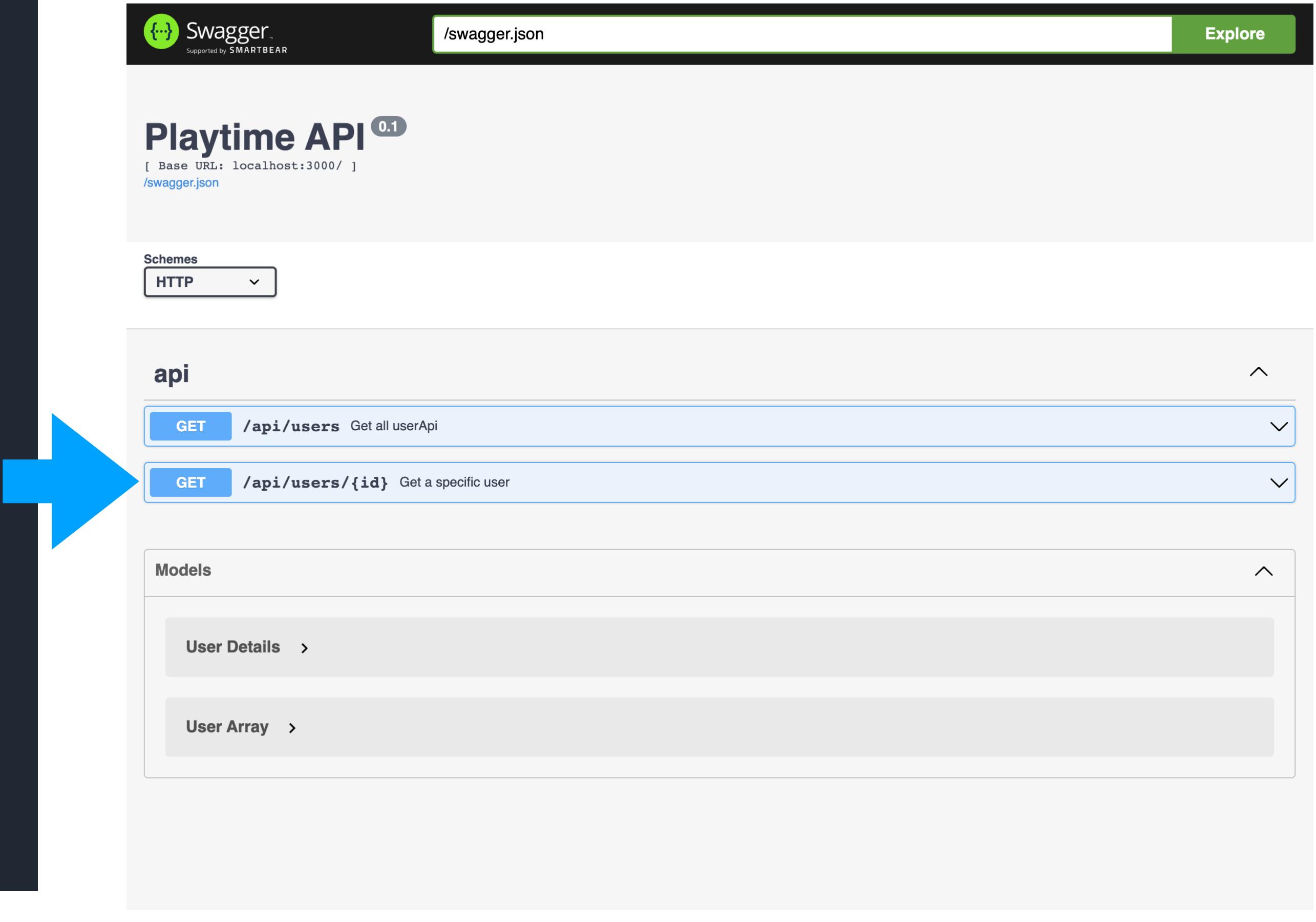
```
export const userApi = {  
    find: {  
        auth: false,  
        handler: async function(request, h) {  
            try {  
                const users = await db.userStore.getAllUsers();  
                return users;  
            } catch (err) {  
                return Boom.serverUnavailable("Database Error");  
            }  
        },  
        tags: ["api"],  
        description: "Get all userApi",  
        notes: "Returns details of all userApi",  
        response: { schema: UserArray }  
    },  
};
```



```
swagger: '2.0'  
host: localhost:3000  
basePath: "/"  
schemes:  
- http  
info:  
    title: Playtime API  
    version: '0.1'  
tags: []  
paths:  
    "/api/users":  
        get:  
            summary: Get all userApi  
            operationId: getApiUsers  
            description: Returns details of all userApi  
            tags:  
            - api  
            responses:  
                '200':  
                    schema:  
                    "$ref": "#/definitions/UserArray"  
                    description: Successful  
definitions:  
    UserDetails:  
        type: object  
        properties:  
            firstName:  
                type: string  
                example: Homer  
            lastName:  
                type: string  
                example: Simpson  
            email:  
                type: string  
                example: homer@simpson.com  
                x-format:  
                    email: true  
            password:  
                type: string  
                example: secret  
            _id:  
                type: string  
                x-alternatives:  
                - type: string  
                - "$ref": "#/x-alt-definitions/_id"  
            __v:  
                type: number  
        required:  
        - firstName  
        - lastName  
        - email  
        - password  
    UserArray:  
        type: array  
        items:  
        "$ref": "#/definitions/UserDetails"  
        x-alt-definitions:  
        _id:  
            type: object
```

# Validate Response

```
import { UserSpec, UserArray } from "../models/joi-schemas.js";
...
findOne: {
  auth: false,
  handler: async function (request, h) {
    try {
      const user = await db.userStore.getUserById(request.params.id);
      if (!user) {
        return Boom.notFound("No User with this id");
      }
      return user;
    } catch (err) {
      return Boom.serverUnavailable("No User with this id");
    }
  },
  tags: ["api"],
  description: "Get a specific user",
  notes: "Returns user details",
  response: { schema: UserSpec, failAction: validationError },
},
```



```
export function validationError(request, h, error) {
  console.log(error.message);
}
```

# Validate Request

```
findOne: {  
  auth: false,  
  handler: async function(request, h) {  
    try {  
      const user = await db.userStore.getUserById(request.params.id);  
      if (!user) {  
        return Boom.notFound({ message: "No User with this id" });  
      }  
      return user;  
    } catch (err) {  
      return Boom.serverUnavailable({ message: "No User with this id" });  
    }  
  },  
  tags: ["api"],  
  description: "Get a specific user",  
  notes: "Returns user details",  
  validate: { params: { id: IdSpec }, failAction: validationError },  
  response: { schema: UserSpec, failAction: validationError },  
},
```

The screenshot shows a Swagger UI interface for a REST API. At the top, it displays a GET request for the endpoint `/api/users/{id}`. The description for this endpoint is "Get a specific user" and it "Returns user details". Below the endpoint, there is a section for "Parameters" with a table. The table has two columns: "Name" and "Description". There is one row for the parameter "id", which is marked as required (\*). The "Name" column contains "id" and the "Description" column contains "string (path)". To the right of the table is a "Try it out" button. Further down, there is a "Responses" section with a table for code 200. The "Code" column contains "200" and the "Description" column contains "Successful". Below this, there is a "Example Value | Model" section containing a JSON object:

```
{  
  "firstName": "Homer",  
  "lastName": "Simpson",  
  "email": "homer@simpson.com",  
  "password": "secret",  
  "_id": "string",  
  "__v": 0  
}
```

- Validate input parameter

# Validate Request/Response

```
create: {  
  auth: false,  
  handler: async function(request, h) {  
    try {  
      const user = await db.userStore.addUser(request.payload);  
      if (user) {  
        return h.response(user).code(201);  
      }  
      return Boom.badImplementation("error creating user");  
    } catch (err) {  
      return Boom.serverUnavailable("Database Error");  
    }  
  },  
  tags: ["api"],  
  description: "Create a User",  
  notes: "Returns the newly created user",  
  validate: { payload: UserSpec, failAction: validationError },  
  response: { schema: UserSpec, failAction: validationError },  
},
```

POST /api/users Create a User ^

Returns the newly created user

Parameters Try it out

Name	Description
body object (body)	Example Value   Model <pre>{   "firstName": "Homer",   "lastName": "Simpson",   "email": "homer@simpson.com",   "password": "secret",   "_id": "string",   "__v": 0 }</pre> Parameter content type application/json

Responses Response content type application/json

Code	Description
200	Successful Example Value   Model <pre>{   "firstName": "Homer",   "lastName": "Simpson",   "email": "homer@simpson.com",   "password": "secret",   "_id": "string",   "__v": 0 }</pre>



Swagger

Supported by SMARTBEAR

/swagger.json

Explore

# Playtime API 0.1

[ Base URL: localhost:3000/ ]

[/swagger.json](#)

## Schemes

HTTP ▾

## api

^

**GET** /api/users Get all userApi

▼

**POST** /api/users Create a User

▼

**DELETE** /api/users Delete all userApi

▼

**GET** /api/users/{id} Get a specific user

▼

## Models

^

**UserDetails** >**UserArray** >

```
swagger: '2.0'
host: localhost:3000
basePath: "/"
schemes:
- http
info:
  title: Playtime API
  version: '0.1'
tags: []
paths:
  "/api/users":
    get: ...
    post: ...
    delete: ...
  "/api/users/{id}":
    get: ...
definitions:
  UserDetails:
    type: object
    properties:
      firstName:
        type: string
        example: Homer
      lastName:
        type: string
        example: Simpson
      email:
        type: string
        example: homer@simpson.com
      x-format:
        email: true
      password:
        type: string
        example: secret
      _id:
        type: string
        x-alternatives:
          - type: string
          - "$ref": "#/x-alt-definitions/_id"
      __v:
        type: number
    required:
      - firstName
      - lastName
      - email
      - password
  UserArray:
    type: array
    items:
      "$ref": "#/definitions/UserDetails"
x-alt-definitions:
  _id:
    type: object
```

# Playtime Swagger



Full Stack Web Development