user-srv Webservice Master Reference

Paulo Gouveia

March 31, 2022

Contents

1 Introduction

Webservice to provide user management and authentication.

The goal is to provide a service that is flexible and customizable. This should allow it to be used in different contexts (e.g., different websites). Customization is mostly performed using environment variables.

1.1 Start Up

When the service starts it will look into the database to check if the *root* user exists. If it does not then it will be created with the following properties:

- email: root@yepkit.com
- password: YepkitTemp1
- role: super-admin

The password should be changed as soon as possible, for obvious security reasons.

2 Endpoints

Available endpoints:

- /create-user
- /read-user
- /update-user
- /delete-user
- /sign-in
- /token-refresh

2.1 /create-user

Endpoint to create a new user. The user document will be persisted in a MondoDB database.

2.1.1 Authorization

Authorization is through JWT.

To create user with role “user” the JWT must have a role of “guest”, “admin” or “super-admin”.

To create user with role “admin” or “super-admin” the JWT must have a role of “super-admin”.

2.1.2 Request

A POST request with a body containing a JSON with the following structure:

```
{
  email: "user@email.com",
  pwd: "somepassword",
  firstName: "Paulo",
  lastName: "Gouveia",
  role: "user" || "admin" || "super-admin"
}
```

The above structure is the minimal required set of properties to create a user. Additional properties can be added to a user at creation time or when updating.

2.1.3 Response

Successful response Status code with value 200 and a JSON in the body with the following structure:

```
{
  jwt: "jwt token",
  id: "user doc id",
  firstname: "Paulo",
  expireSeconds: 7200
}
```

Error response Status code with value 4xx and a JSON in the body with the following structure:

```
{
  error: "",
  message: ""
}
```

2.2 /read-user

Read a user document from the database and return it to the client that requested it.

2.2.1 Authorization

JWT of the requested user or a JWT with admin or higher role.

2.2.2 Request

POST request with a JSON body with the following structure:

```
{
  id: ""
}
```

Just one of these two properties are required. The document can be fetched from the database using the id or the email.

2.2.3 Response

In case of a successful request the response will have the status code 200. Otherwise in case of error the status code will be ≥ 400. In both cases a JSON body will be returned.

In the case of **success** the body JSON is the following:

```
{
  email: null,      // string, required
  firstName: null,  // string, required
  lastName: null,   // string, required
  role: "user",     // string, required
  company: null,    // string
  vatNumber: null,  // string
  billingDetails: {
    contactName: null,    // string
    company: null,        // string
    addLine1: null,       // string
    addLine2: null,       // string
    postCode: null,       // string
    city: null,           // string
    state: null,          // string
    country: null,        // string
    phone: null,          // string
    vatNumber: null
  },
  shippingDetails: {
    contactName: null,    // string
    company: null,        // string
    addLine1: null,       // string
    addLine2: null,       // string
    postCode: null,       // string
    city: null,           // string
    state: null,          // string
    country: null,        // string
    phone: null,          // string
  },
  status: "unconfirmed"     // string, "unconfirmed" | "active" | "inactive"
                            // | "blacklisted"
}
```

In the case of **error** the JSON will be like the following:

```
{
  error: "",
  message: ""
}
```

2.3 /update-user

Update a user document in the database.

2.3.1 Authorization

The authorization mechanism is based on a JWT on the *Authorization* request header. For most of the user properties can be updated by the specific user or by a higher role user. The exception to this rule are listed bellow.

The following properties may only be performed by a **super-user**:

- role
- password

JWT of the specific user or with a admin (or higher) role.

2.3.2 Request

POST request with a JSON body. This JSON has the following structure:

```
{
  id: "",
  email: "",
  updateProperties: {
    email: 'user2@teste.com',
    firstName: 'Sérgio',
    lastName: 'Santos',
    password: 'somepwd!!!',

  }
}
```

Both id or email can be used to filter the target user document to be updated. Only one of them is required, any of them. If both are present in the JSON the id will be used.

2.3.3 Response

If the request is executed successfully the response status code will have the value 200. If an error occurred the status code will a 4xx.

In the case of an error, in addition to the status code, a JSON body will be sent with the following structure.

```
{
  error: "",
  message: ""
}
```

2.4 /delete-user

Delete a user document on the data base. This action is destructive and not recoverable. Only a super-user is allowed to perform this action.

2.4.1 Authorization

JWT in the authorization header with *super-admin* user role.

2.4.2 Request

POST request with a JSON body with the following structure:

```
{
  id: ""
}
```

2.4.3 Response

HTTP response with status code **200** in case of success. In case of error the status code is ≥ 400.

2.5 /sign-in

Endpoint to sign-in a user or a guest.

Signing-in a guest is a way of providing a session mechanism to anonymous website visitors.

Being a sign-in it obviously doesn't require authorization.

2.5.1 Request

POST request with the following JSON:

```
{
  email: "",
  pwd: "",
  guest: false
}
```

The *guest* property is a flag to identify if it's a request to authenticate a registered (signed-up) user or a guest (anonymous user). If the *guest* property is set to *true* the email and password properties are not considered (or necessary) and a JWT with role "guest" is issued.

2.5.2 Response

For a valid request the response has a **status code 200** and a JSON in the body with the following structure:

```
{
  jwt: "",
  userId: "12334555",
  firstName: "",
  role: "", // user | admin | super-admin
  expiresIn: 3600
}
```

The above response is for an authenticated user. For a *guest* user the response is the following:

```
{
  jwt: "",
  role: "guest",
  expiresIn: 3600
}
```

In case of a error signing in a response with **status code** $\geq$ **400** and a JSON body with the structure bellow is replied:

```
{
  error: "sign in error",
  message: "Unable to sign you in."
}
```

2.5.3 JWT structure

ES512 is used for the algorithm and the payload has the structure bellow.

```
{
  "iss": "user-srv",
  "sub": "yepkit",
  "aud": "yepkit.com",
  "exp": expiretime,
  "user": {
    "id": "",
    "role": "admin"
  }
}
```

Where the *user.id* is the MongoDB *_id* Object converted to string. For example:

```
let _id = new ObjectId("617d9a9d278ca1ce11440d23");

user.id = _id.toString();  // user.id is set with value
                    //"617d9a9d278ca1ce11440d23"
```

2.6 /token-refresh

Endpoint to refresh a aging JWT. Clients should keep track of the JWT's age and request a refresh before it expires.

2.6.1 Authorization

Authorization is given to requests that have the JWT to be refreshed.

2.6.2 Request

POST request with the JWT to be refreshed in the header.

2.6.3 Response

For a token successfully refreshed a response with status code 200 is sent together with a JSON body. On the other hand if an error occurred the response will have a status code $\geq$ 400 and no body. If successful the request is responded with the new JWT in a JSON.

In the case of success the JSON body is the following:

```
{
  jwt: "",
  expire: 3600
}
```

3 Service Configuration

Configuration is performed using environment variables.

3.1 Environment variables

Environment variables can be used to configure the service. If a environment variable is not set a default value will be used.

3.1.1 General configuration variables

Environment variables used for general service configuration.

SERVICE_PORT Port which the service will listen to. Default value: **3004**.

3.1.2 JWT related variables

Environment variables related with the issuing of a JWT.

JWT_ISS Issuer for the JWT. String. Default value: **user-srv**.

JWT_SUB Subject for the JWT. String. Default value: **yepkit**.

JWT_AUD Audience for the JWT. String. Default value: **yepkit**.

JWT_EXP Time to expire in seconds. Number. Default value: **86400** (one day).

3.1.3 yepkit-authorization module configuration variables

KEY_MANAGER_HOST Host of the *key-manager* service.

KEY_MANAGER_PORT

3.1.4 Database configurations

MONGODB_URI Default: "mongodb://127.0.0.1:27017".

3.1.5 Kafka configurations

KAFKA_BROKER Default: "127.0.0.1:9092".

4 Keys

Key name: userSrvKey

Issuer: yepkit

Audience: yepkit

5 Users database

Database name: **users**

Collections:

- details

5.1 details collection

Details collection contains the documents with the details of a user. These include: Personal information; Billing details; and Shipping details.

5.1.1 Document structure

```
{
  email: null,      // string, required
  firstName: null,  // string, required
  lastName: null,   // string, required
  pwdHash: null,    // string, required
  role: "user",     // string, required
  company: null,    // string
  vatNumber: null,  // string
  billingDetails: {
    contactName: null,    // string
    company: null,        // string
    addLine1: null,       // string
    addLine2: null,       // string
    postCode: null,       // string
    city: null,           // string
    state: null,          // string
    country: null,        // string
    phone: null,          // string
    vatNumber: null
},
  shippingDetails: {
    contactName: null,    // string
    company: null,        // string
    addLine1: null,       // string
    addLine2: null,       // string
    postCode: null,       // string
    city: null,           // string
    state: null,          // string
    country: null,        // string
    phone: null,          // string
  },
  status: "unconfirmed"     // string, "unconfirmed" | "active" | "inactive"
```

```
                                             // | "blacklisted"
}
```

6 Cookbook

7 Dependencies

This service will make requests to the following services:

- key-manager