# Auth API documentation

This is the documentation for the auth API, this is meant to be a guide for the API usage, this documentation should be read before using the API to avoid mistakes.

## How does this API work?

This API uses JWTs for authentication, we use two types of tokens the “refresh tokens”, and the “action tokens”.

### **Refresh Token**

This type of token serves the propose of generating access tokens (and thus the refresh tokens are generated at login and deleted at logout), and they are used for some user’s operations such as changing name and last name.

This kind of tokens never automatically expire, they only expire when requested to (like in the logout and logoutAll route).

### **Access Token**

This type of token serves the main propose of the app, they are meant to be used to credit that the user is already authenticated.

They are meant to be used as one-time tokens, since they expire soon after they are created (30s).

### **Database**

A mongo database is used to save the registered users and all the important user related data, the schema used to store the users is the following.

{

    name: { type: String, required: true },

    lastName: { type: String, required: true },

    email: { type: String, required: true, unique: true },

    password: { type: String, required: true },

    createdAt: { type: Date, default: Date.now },

    updatedAt: { type: Date, default: Date.now },

    refreshTokens: [{ type: String }]

}

**Explanation:**

name: contains the name of the user (to use it on apps using the API).

lastName: contains the name of the user (to use it on apps using the API).

email: it must be unique, it doesn’t need to be a real email, it is going to act as identifier for the user.

password: it is the user’s password, which is hashed and added 10 bcrypt salt cycles.

createdAt: it stores the exact time at which a user was stored to database.

updatedAt: it stores the exact time of the last: name, lastName or password change.

refreshTokens: it is a list of all the valid refresh tokens for that user, a new one is stored at a login, and deleted at a logout.

## Routes:

### **/register - POST**

This route registers a user in the database, it receives a JSON containing the following info:

|  |  |
| --- | --- |
| Key | Value |
| Name | The user’s name. |
| lastName | The user’s last name. |
| Email | The user’s email (it doesn’t have to be real). |
| Password | The user’s password. |

Request.body:

{

    "name": "Clark",

    "lastName": "Kent",

    "email": "superman@gmail.com",

    "password": "ImSuperman"

}

And that, when passed to the /register, will generate a user, which will be stored in the database, and the server will reply with response that looks like this:

Res.body:

{

    "refreshTokens": [],

    "\_id": "60ff6267393c8a1e4cb72966",

    "name": "Clark",

    "lastName": "Kent",

    "email": "superman@gmail.com",

    "password": "$2b$10$h1QwtYJXK7UPp9B7FyEOL.wtRAjA/.h9OqNZOgLkRa0BxG/AiDksm",

    "createdAt": "2021-07-27T01:33:27.202Z",

    "updatedAt": "2021-07-27T01:33:27.202Z",

    "\_\_v": 0

}

This is the user stored at database, as you see, the refresh tokens are empty, that’s because those are going to be generated and registered on the /login route.

### **/login - POST**

This route generates a refresh token by receiving the user’s email and password, the request body should contain the following parameters (in a json):

|  |  |
| --- | --- |
| Key | Value |
| Email | The user’s already registered email. |
| Password | The password corresponding to that email. |

Req.body:

{

    "email": "superman@gmail.com",

    "password": "ImSuperman"

}

And the server should reply (if everything is valid) with a response that looks like this:

{

    "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoiNjBmZjYyNjczOTNjOGExZTRjYjcyOTY2IiwiZW1haWwiOiJzdXBlcm1hbkBnbWFpbC5jb20iLCJpYXQiOjE2MjczNTAyMDl9.WHSTXR0X142IwuJ5i2023HR7ZTDfy8JHQbL4nBctlWY"

}

That is a refresh token, it contains the user’s id in the database and his email, this token was registered in that user’s document in the database and thus it will be valid.

### **/logout - DELETE**

This route invalidates a refresh token by receiving the token itself, and that will be called as a logout. The parameters are:

|  |  |
| --- | --- |
| Key | Value |
| Refresh Token | The refresh token we want to invalidate |

Req.body:

{

    "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoiNjBmZjYyNjczOTNjOGExZTRjYjcyOTY2IiwiZW1haWwiOiJzdXBlcm1hbkBnbWFpbC5jb20iLCJpYXQiOjE2MjczNTAyMDl9.WHSTXR0X142IwuJ5i2023HR7ZTDfy8JHQbL4nBctlWY"

}

And the server will delete that token from the user’s corresponding document in database, therefore making it invalid, a success response should look like this:

Res.body:

"You have been logged out, that refresh token will no longer be valid"

And that refresh token will no longer work.

### **/logoutAll - DELETE**

This route will invalidate all the refresh tokens that have been created. To accomplish this, the user is going to have to send He's email and his password. And then the server is going delete all the valid refresh tokens from the database.

The following are the parameters needed to in order to accomplish this:

|  |  |
| --- | --- |
| Key | Value |
| Email | The user’s already registered email. |
| Password | The password corresponding to that email. |

Req.body:

{

        "email": "superman@gmail.com",

        "password": "ImSuperman"

}

And a successful response should look something like this:

Res.body:

"All refresh tokens have been invalidated"

### **/token - POST**

This route is intended to generate an action token, it gets a refresh token and responds with an action token. The parameter needed is the refresh token:

|  |  |
| --- | --- |
| Key | Value |
| Refresh Token | The refresh token we want to invalidate |

Req.body:

{

    "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoiNjBmZjYyNjczOTNjOGExZTRjYjcyOTY2IiwiZW1haWwiOiJzdXBlcm1hbkBnbWFpbC5jb20iLCJpYXQiOjE2MjczNTAyMDl9.WHSTXR0X142IwuJ5i2023HR7ZTDfy8JHQbL4nBctlWY"

}

A successful response should look like this:

Res.body:

{

    "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoiNjBmZjYyNjczOTNjOGExZTRjYjcyOTY2IiwiZW1haWwiOiJzdXBlcm1hbkBnbWFpbC5jb20iLCJuYW1lIjoiQ2xhcmsiLCJsYXN0TmFtZSI6IktlbnQiLCJpYXQiOjE2MjczNTI3MjYsImV4cCI6MTYyNzM1Mjc1Nn0.lvfmtYVDvDF-yayh4IxWLript0JId5ea9XOqnA3X8MA"

}

And that is a 30s valid action token.

### **/decode - POST**

It receives an action token and returns the data contained in it. This route is only included for testing proposes, a route or middleware like this, should be built in the server this auth API serves to. It receives an access token and returns the info. Contained on it.

|  |  |
| --- | --- |
| Key | Value |
| Access Token | A valid Access Token |

Req.body:

{

    "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoiNjBmZjYyNjczOTNjOGExZTRjYjcyOTY2IiwiZW1haWwiOiJzdXBlcm1hbkBnbWFpbC5jb20iLCJuYW1lIjoiQ2xhcmsiLCJsYXN0TmFtZSI6IktlbnQiLCJpYXQiOjE2MjczNTI3MjYsImV4cCI6MTYyNzM1Mjc1Nn0.lvfmtYVDvDF-yayh4IxWLript0JId5ea9XOqnA3X8MA"

}

And a successful response looks like this:

Res.body:

{

    "user\_id": "60ff6267393c8a1e4cb72966",

    "email": "superman@gmail.com",

    "name": "Clark",

    "lastName": "Kent",

    "iat": 1627353311,

    "exp": 1627353341

}

### **/update - PUT**

This route has the propose of changing the user’s name and last name, it requires the client to send a valid refresh token.

|  |  |
| --- | --- |
| Key | Value |
| Refresh Token | A valid Refresh Token |

Req.body:

{

    "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoiNjBmZjYyNjczOTNjOGExZTRjYjcyOTY2IiwiZW1haWwiOiJzdXBlcm1hbkBnbWFpbC5jb20iLCJpYXQiOjE2MjczNTAyMDl9.WHSTXR0X142IwuJ5i2023HR7ZTDfy8JHQbL4nBctlWY"

}

And a successful response looks like this:

Res.body

"The user info has been updated"

And now the user information on database has been changed.

### **/updatePassword - PUT**

This route serves the propose of allowing the user to change his password, client must provide valid email and password.

|  |  |
| --- | --- |
| Key | Value |
| Email | The user’s already registered email. |
| Password | The password corresponding to that email. |

Req.body:

{

    "email": "superman@gmail.com",

    "password": "ImSuperman",

    "newPassword": "ImTheSuperman"

}

And a successful response looks like this:

"The password has been changed"

### **/delete – DELETE**

With this route the user is going to be able to delete the user in the server, in order to do that the client has to provide a valid email and password.

|  |  |
| --- | --- |
| Key | Value |
| Email | The user’s already registered email. |
| Password | The password corresponding to that email. |

Req.body:

{

        "email": "superman@gmail.com",

        "password": "ImTheSuperman"

}

And a successful response looks like this:

"The user has been deleted"

And after this message is gotten, the user will no longer exist.

## Validations:

These are the validations the fields must pass when they are sent to the server, not passing them correctly, will result in the server responding with the status 400 – Bad Request:

Name: The name must be a string, have at least 2 characters long and at most 20 characters long, it must not contain <,>, &, ', " and \ characters

Last Name:

The lastName must be a string, have at least 2 characters long and at most 30

characters long, it must not contain <,>, &, ', " and \ characters

Email:

The email must be a valid email address, it must match this regular expression:

/^(([^<>()[\]\\.,;:\s@\"]+(\.[^<>()[\]\\.,;:\s@\"]+)\*)|(\".+\"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/

Password:

The password must be a string, have at least 6 characters long and at most 20

characters long and must not contain spaces.