**MICROSERVICES**

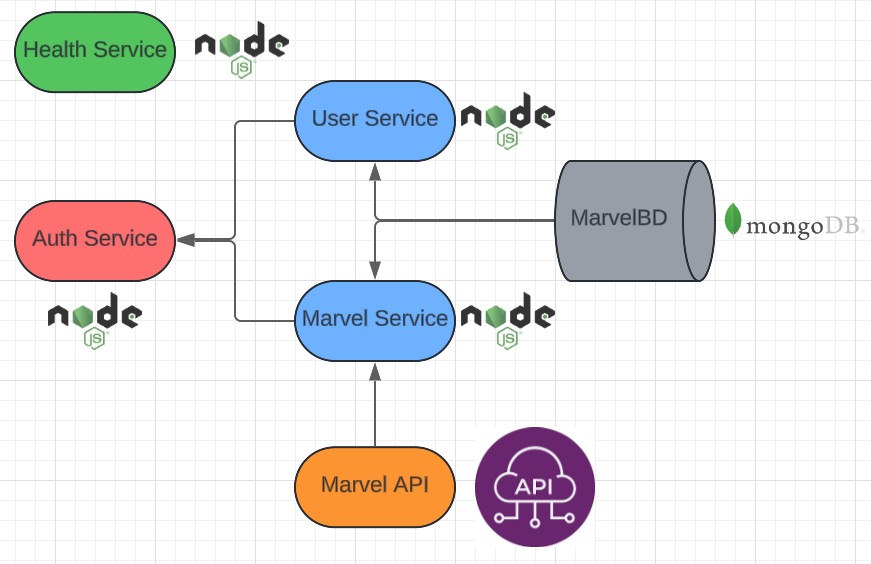
Planning

Objective

The objective of developing a project using a microservices architecture is scalability and maintainability, allowing for more efficient management of very large applications, and enabling each project to scale individually without directly affecting other services.

Requirements and Limitations To understand the decisions made for the project, the requirements and limitations must be taken into account, which are as follows:

1. Use microservices architecture
2. Use NodeJs
3. Use an API, in this case, the Marvel API was used
4. 3 days to complete it
5. Dockerize the application
6. Deploy the application in the cloud
7. Upload the application to a remote repository
8. Generate documentation

 Once this is understood, the necessary services for the application can be structured, which look as follows.

We can observe that the Auth service functions as the core of the others as it has the list of these, the Health service is responsible for providing us with information about the status of all other services, indicating whether they are active or inactive, the User service provides us access to the database to be able to obtain a list of all users as well as to register them, the Marvel service which interacts directly with the Marvel API to return 100 Marvel characters or obtain a character by their ID and finally the MongoDB database which stores the users as well as their passwords.

Structure

The structure of each of the services is a version similar to MVC, which is as follows:

Service

-node\_modules

-src

-controllers

-middleware

-models

-routes

-test

This structure was chosen for the simplicity that the project will have, which can also perfectly serve for large projects, allowing the use of code editor tools for the readability of the project when it is very large.

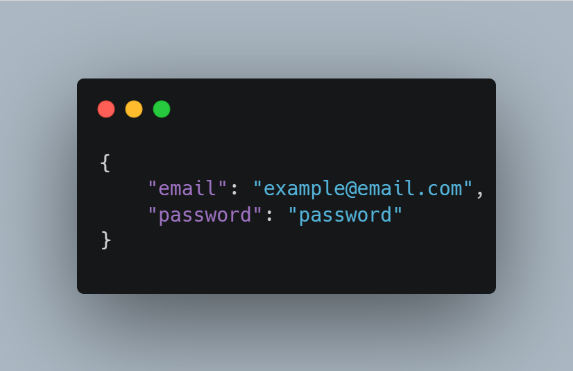
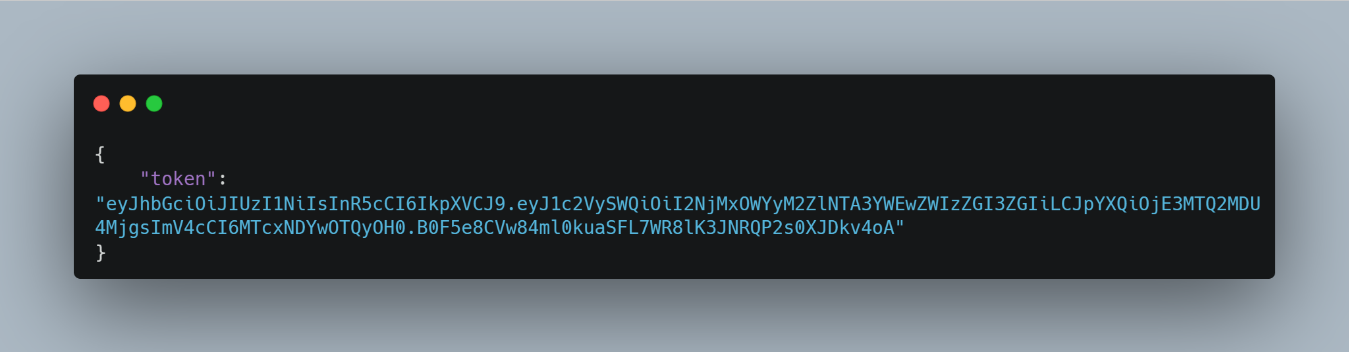
Principal Scripts

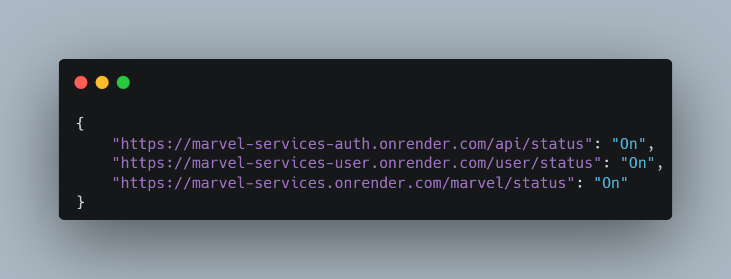
As main server files we have 4, which are as follows:

1. Index.js: This is the file that starts the application with which we bring all the server.
2. Server.js: In this file we have all the configurations, as well as necessary libraries for the server
3. Config.js: In this file we get the environment variables and in case any of them do not exist, we generate a default value for each of them.
4. Database.js: Here we have the configuration for the connection to the MongoDB database.

Endpoints

The endpoints are located in the routes folder, which are invoked in the server.js file to work, the list of current routes of the application are as follows:

* POST /api/auth/login:  
   Returns the JWT which serves us to be able to use the other endpoints Receives a JSON like this:
* GET /api/health/getStatusServices:

Returns a JSON of the status of all other service

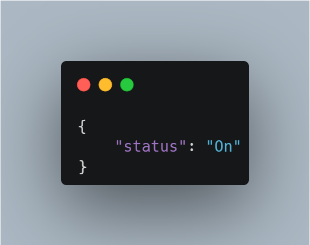
* GET /api/marvel/getAll:

Returns all Marvel characters

* GET /api/marvel/getOneById/:id:

Returns the character chosen by its ID

* GET /api/status:

It is an endpoint that is available in all services, which returns “On” if the service is turned on

* GET /api/users/getUsers:

Returns all users, without counting passwords

Methods

Marvel service methods

 Now, to interact with the API, you need the public key, private key, and the current date to create a hash and add it to the URL.

1. getCharacters

Gets all Marvel characters (maximum 100) from the Marvel API and returns them.

1. getCharacterById

Gets the character by the Marvel id returning the first result. User service methods

User service methods

1. signup

Allows to register users, verifying the integrity of the data, where an email is mandatory and the passwords must match each other, where it is stored in an array of errors as they are generated, otherwise the data is saved and the password is encrypted.

1. getUsers

Gets from all registered users their \_id, email and date.

Health service methods

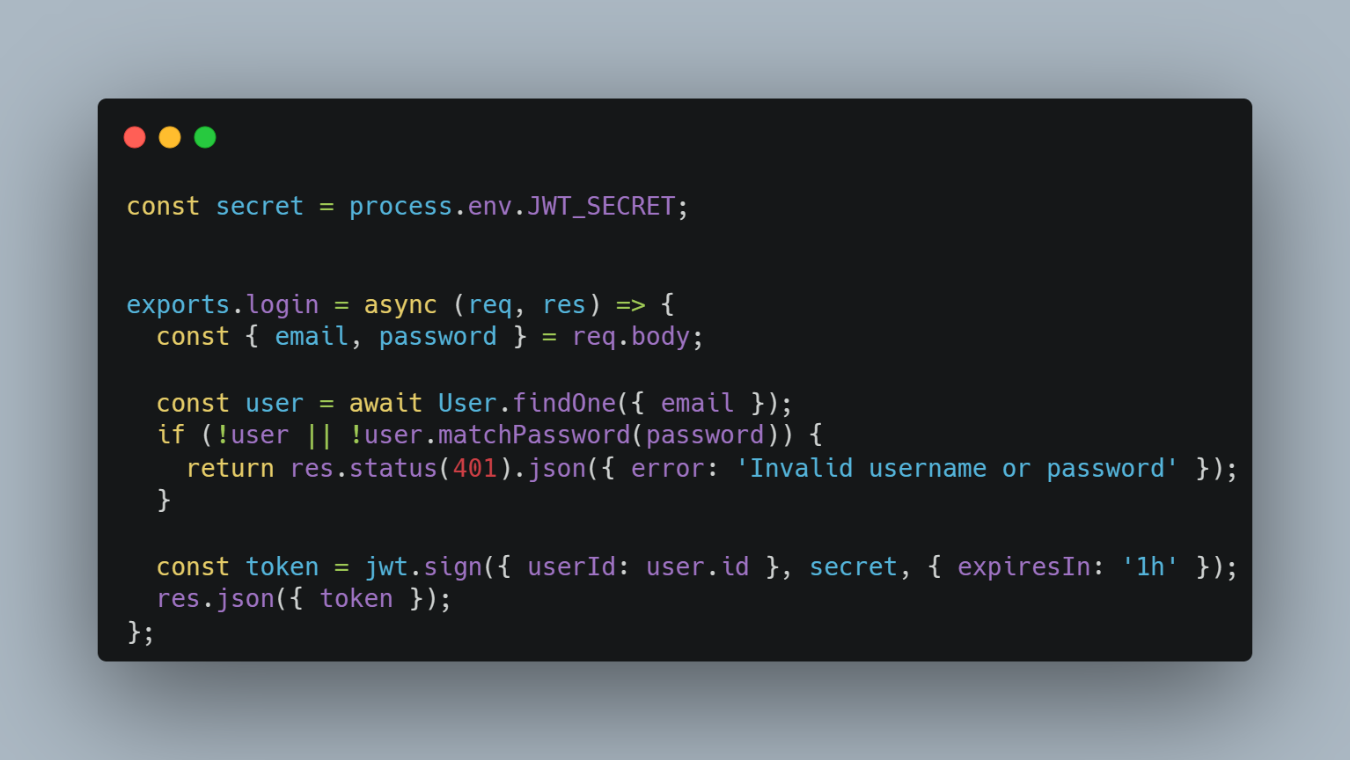
1. checkServices

This method first has to have the urls of the 3 services to make requests to another endpoint that exists in all services, which indicates the status of the service and gives us on or off depending on whether the service is active or inactive.

Auth service methods

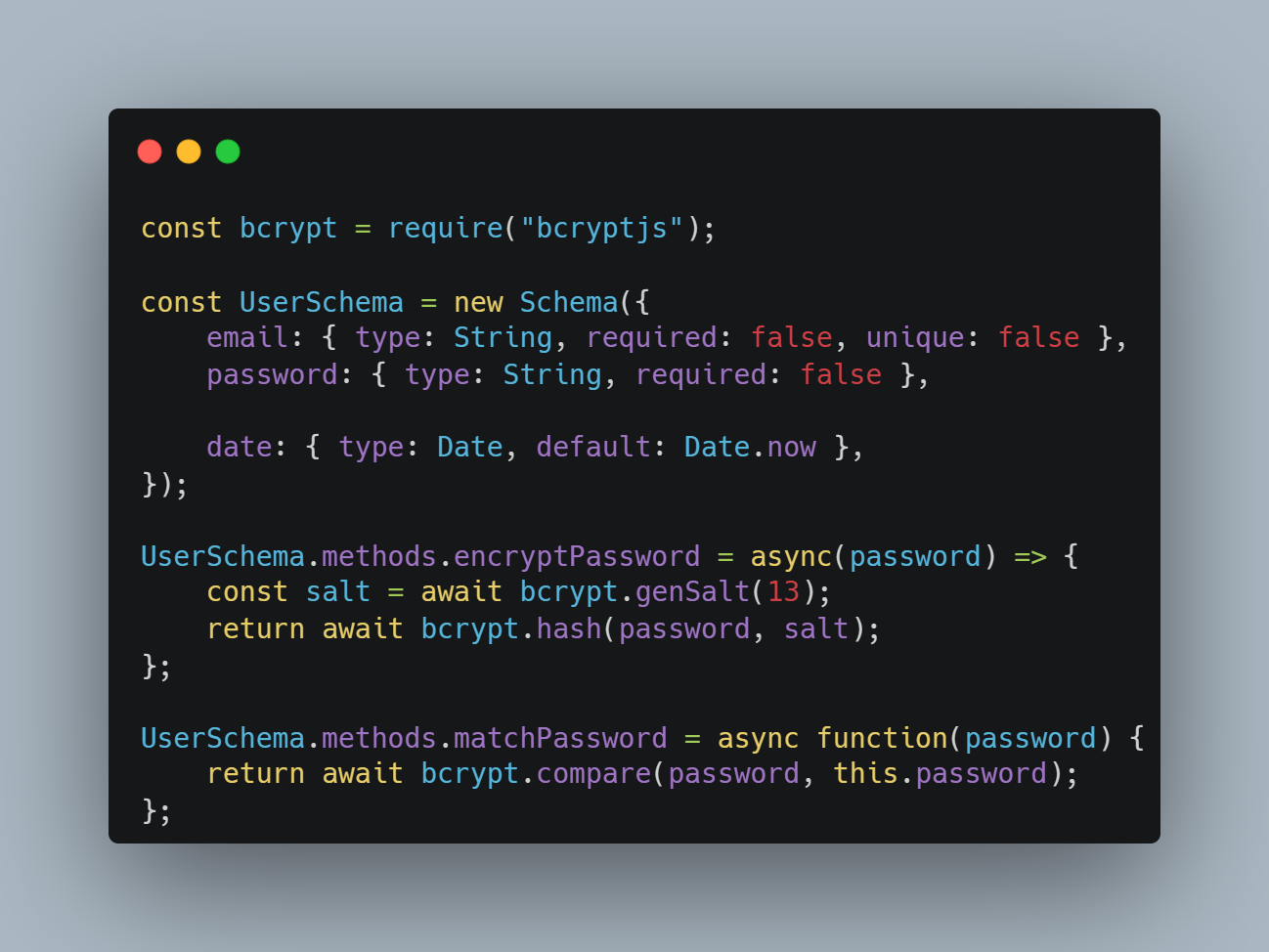
1. login

In this method we first have the verification of the data, where we obtain by email the user from the database to verify its existence and then verify that the encrypted password is correct, to finally send the generated token which has to be signed with the secret token generated with the following command.

Which we paste into the .env so that the token can be signed.

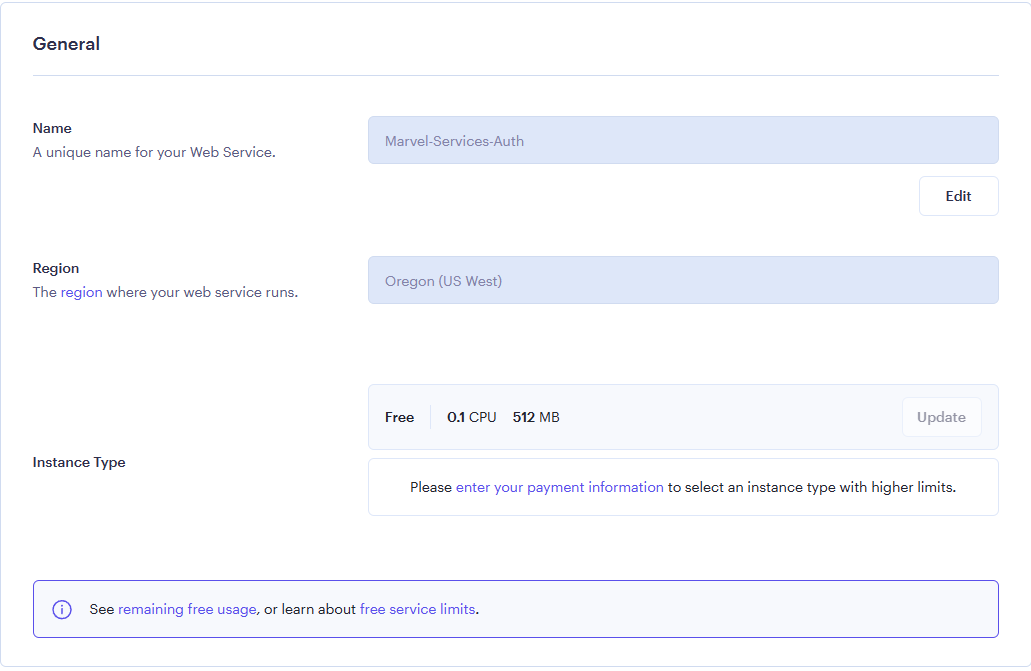
Models

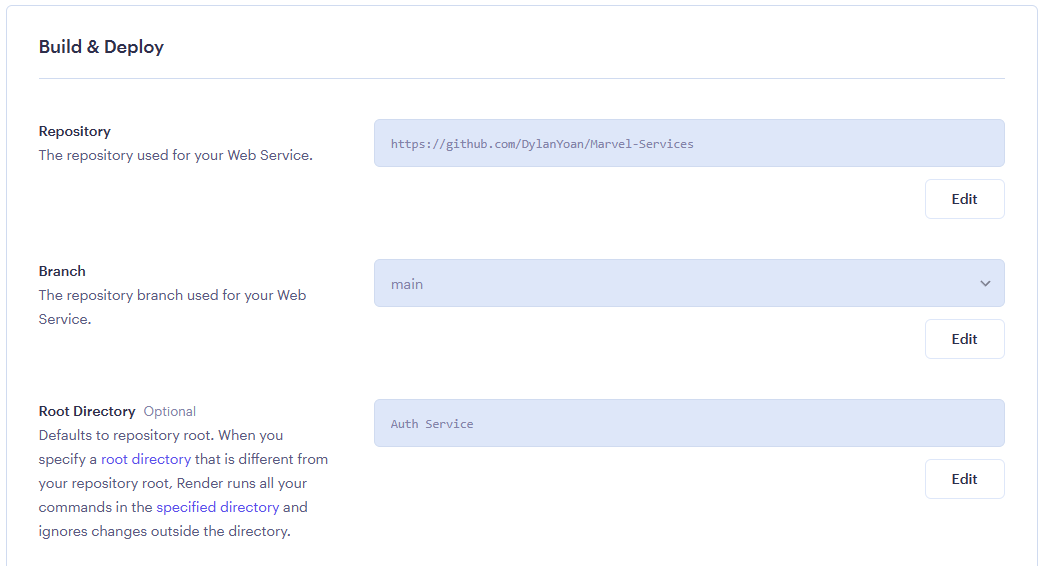
1. User

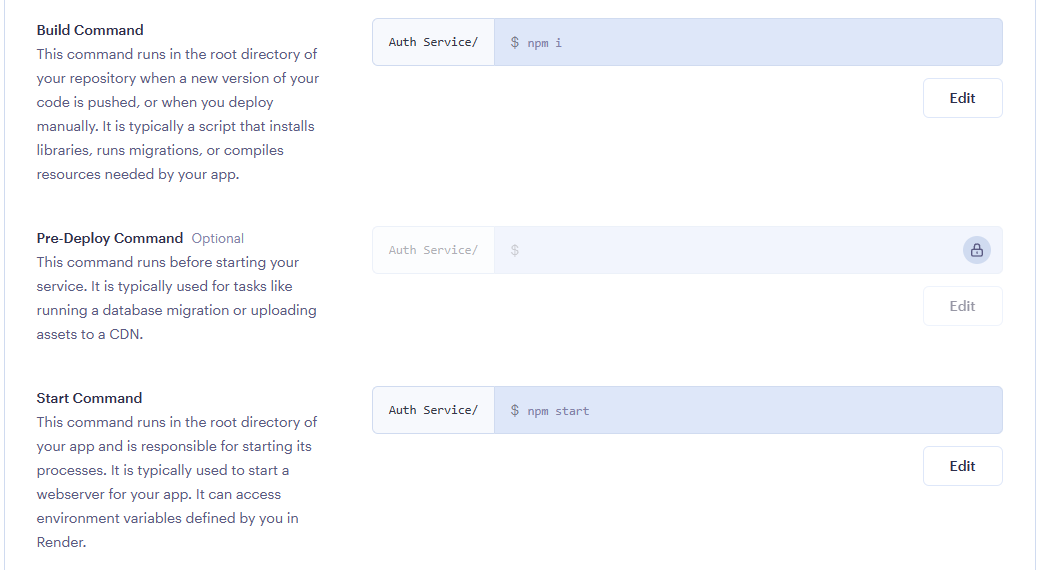
This model exists to be able to register users, generate the method to encrypt the password using a salt which generates a random string that is added to the hash to make it more complex and to be able to encrypt the password in a better way, as well as another method that verifies that the password matches the hash.

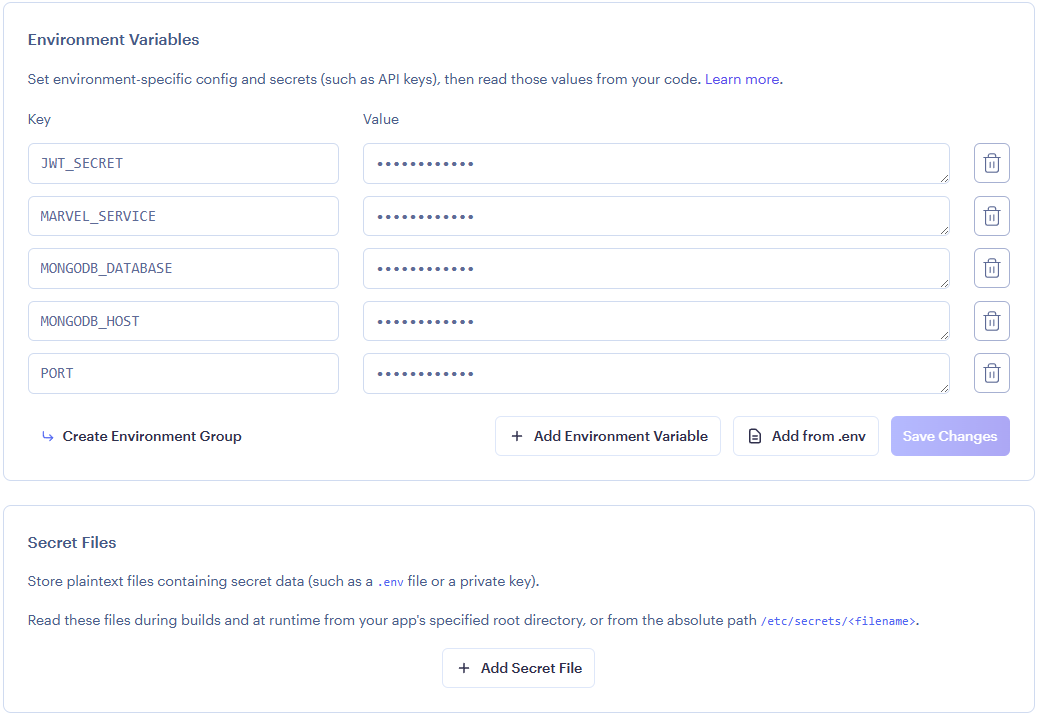
Deploy

To deploy all services, the Render cloud was used, which uses Docker as a way to deploy projects, however, the way they can be uploaded is through the github repository and configuring the necessary steps to work, such as a Dockerfile, however, through a graphical interface, as well as the configuration of the environment variables.

Despite the fact that the free plan has less access to functionalities we have the necessary ones, starting with the name and location of the server.

We can also configure the repository, the branch and the directory, being the last one that helped me to deploy my 4 services with a single repository.

In the same way the commands to install and deploy the project, in this case, to install the libraries and start the project.

Finally, we can configure or import the environment variables.

Final URL

Auth Service: <https://marvel-services-auth.onrender.com>

User Service: <https://marvel-services-user.onrender.com>

Health Service: <https://marvel-services-health.onrender.com>

Marvel Service: <https://marvel-services.onrender.com>