Backend -CRUD Perfil, Crear y Eliminar Characteres y Seasons

Autenticación, enrutamiento -login logout y Creación de Post

Para iniciar la API back ejecutamos los siguientes pasos:

1. `npm  init` para crear el paquete.json

2.`npm i express mongoose passport passport-jwt jsonwebtoken body-parser bcryptjs validator`

Se descarga los siguientes modulos

3. `npm i -D nodemon`

4. Se ejecuta el node server.

5. En el paquete package.json generado cambiamos `"start": "node server.js"` por `"server": "nodemon server.js"`

Conexión a MongoDB con Mongoose

:

1. Se crea un archivo `keys.js` dentro del directorio `config` y se pega en el URI de MongoDB de **mLab** a keys.js. Ejemplo:

```javascript

module.exports = {  mongoURI: "mongodb://username:password@ds117545.mlab.com:17545/react-social-network"};

2.  Se agrega dentro del directorio de config: `const mongoose = require("mongoose" );` y `server.js`

//DB Config

const db = require("./config/keys").mongoURI;

// Connect to MongoDB thru Mongoose
```

```
mongoose
  .connect(db)
  //.then = if it connects successfully
  .then(() => console.log("MongoDB Connected"))
  //catches if login had error (wrong pw in keys.js or something)
  .catch(err => console.log(err));
```

Enrutamiento de archivos con Express Router

Se crean rutas separadas para cada uno de nuestros objetos.

1. Creamos una carpeta llamada router que contendrá cada una de las API.

Ejemplo:

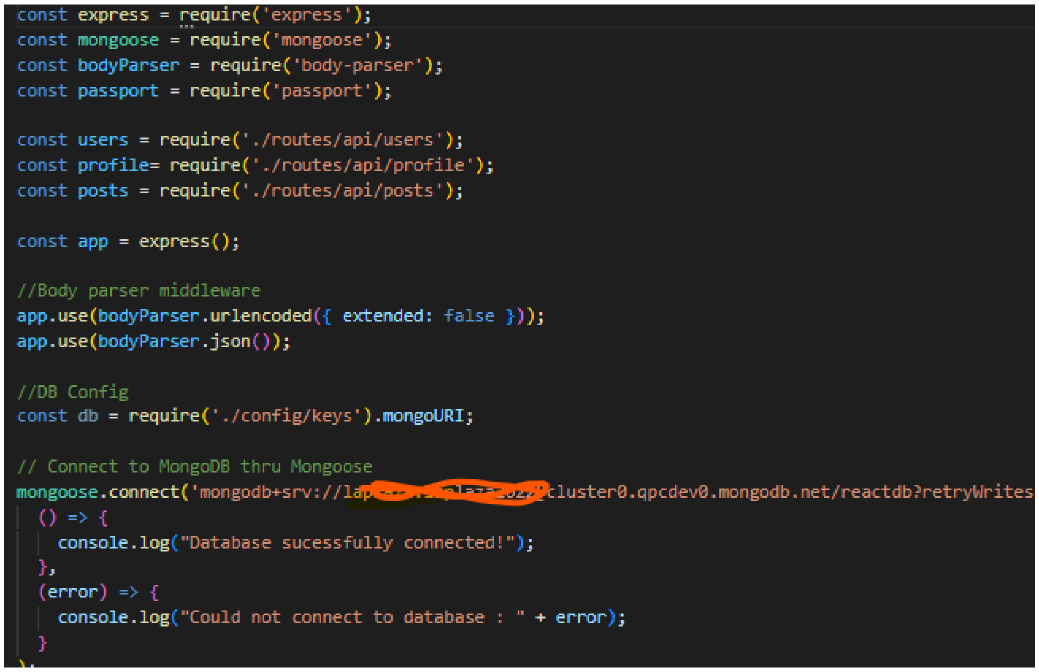
- `users.js` se encarga de la autenticación (nombre de usuario, correo electrónico, auténtico)
- `profile.js`(CRUD Perfil, CRUD Personajes, CRUD Temporadas)
- `posts.js` para publicaciones de personajes más destacados y/ o favoritos y comentarios de usuarios.

2. Se agrega en server.js

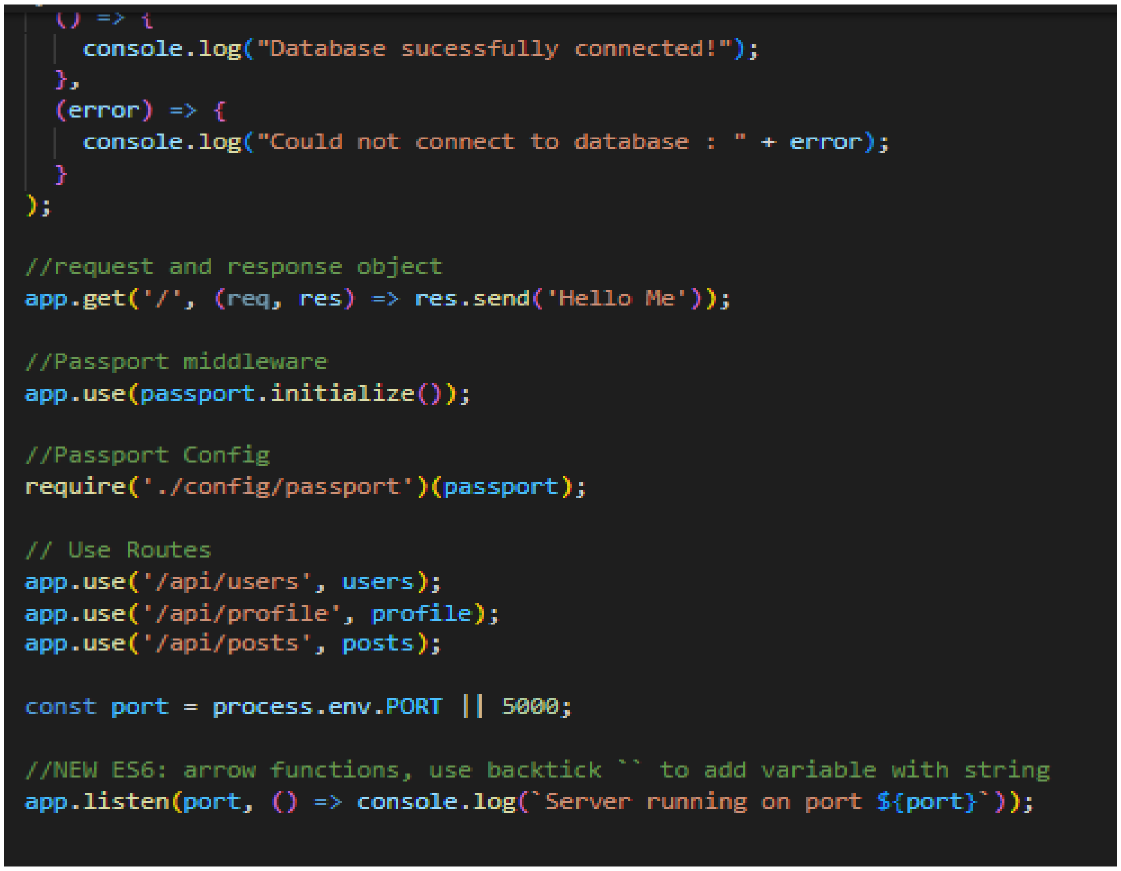
```
const users = require("./routes/api/users");

const profile = require("./routes/api/profile");

const posts = require("./routes/api/posts");
```

3. Se ustilizan las rutas app.use

```
// Use Routes

app.use('/api/users', users);

app.use('/api/profile', profile);

app.use('/api/posts', posts);
```



```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const passport = require('passport');

const users = require('./routes/api/users');
const profile= require('./routes/api/profile');
const posts = require('./routes/api/posts');

const app = express();

//Body parser middleware
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

//DB Config
const db = require('./config/keys').mongoURI;

// Connect to MongoDB thru Mongoose
mongoose.connect('mongodb+srv://lap[redacted]cluster0.qpcdev0.mongodb.net/reactdb?retryWrites
  () => {
    console.log("Database sucessfully connected!");
  },
  (error) => {
    console.log("Could not connect to database : " + error);
  }
```



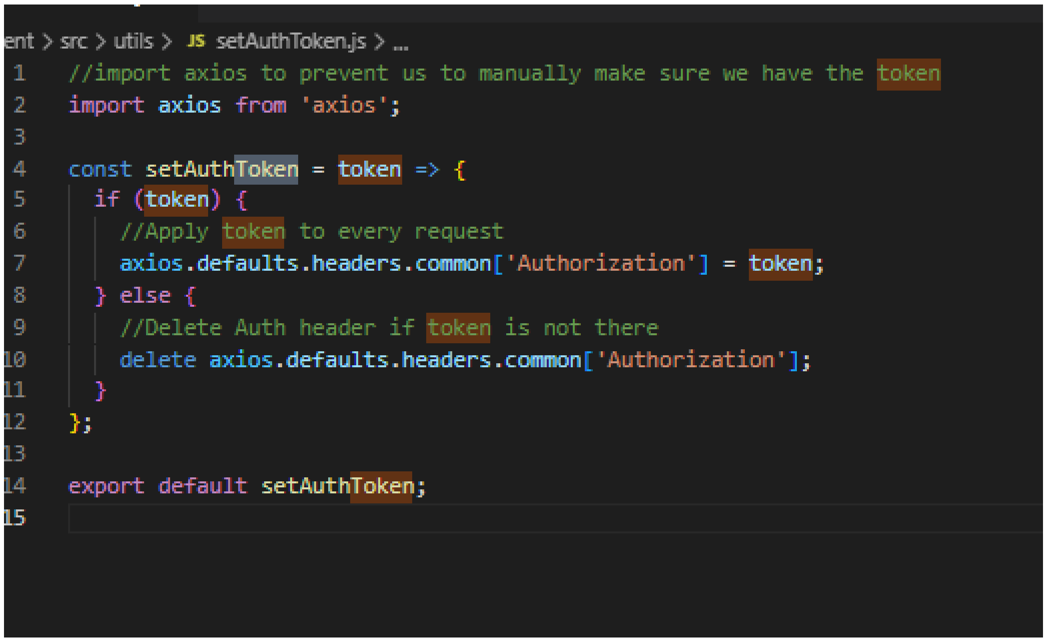
```
  () => {
    console.log("Database sucessfully connected!");
  },
  (error) => {
    console.log("Could not connect to database : " + error);
  }
);

//request and response object
app.get('/', (req, res) => res.send('Hello Me'));

//Passport middleware
app.use(passport.initialize());

//Passport Config
require('./config/passport')(passport);

// Use Routes
app.use('/api/users', users);
app.use('/api/profile', profile);
app.use('/api/posts', posts);

const port = process.env.PORT || 5000;

//NEW ES6: arrow functions, use backtick `` to add variable with string
app.listen(port, () => console.log(`Server running on port ${port}`));
```

Creación del modelo de usuario: autenticación, tokens web JSON, registro, inicio de sesión

1. Se crea el directorio models para User.js, profile.js y post.js, Los cuales definirán el tipo de variable y/o atributo del objeto

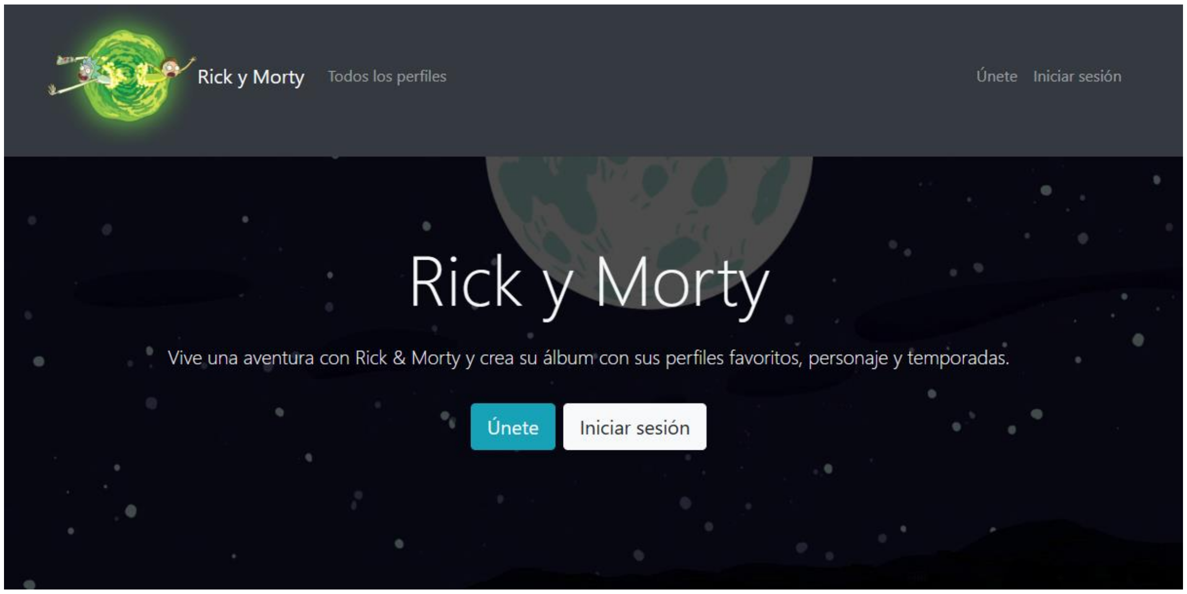
2. npm i gravatar para sacar el avatar del correo electrónico.

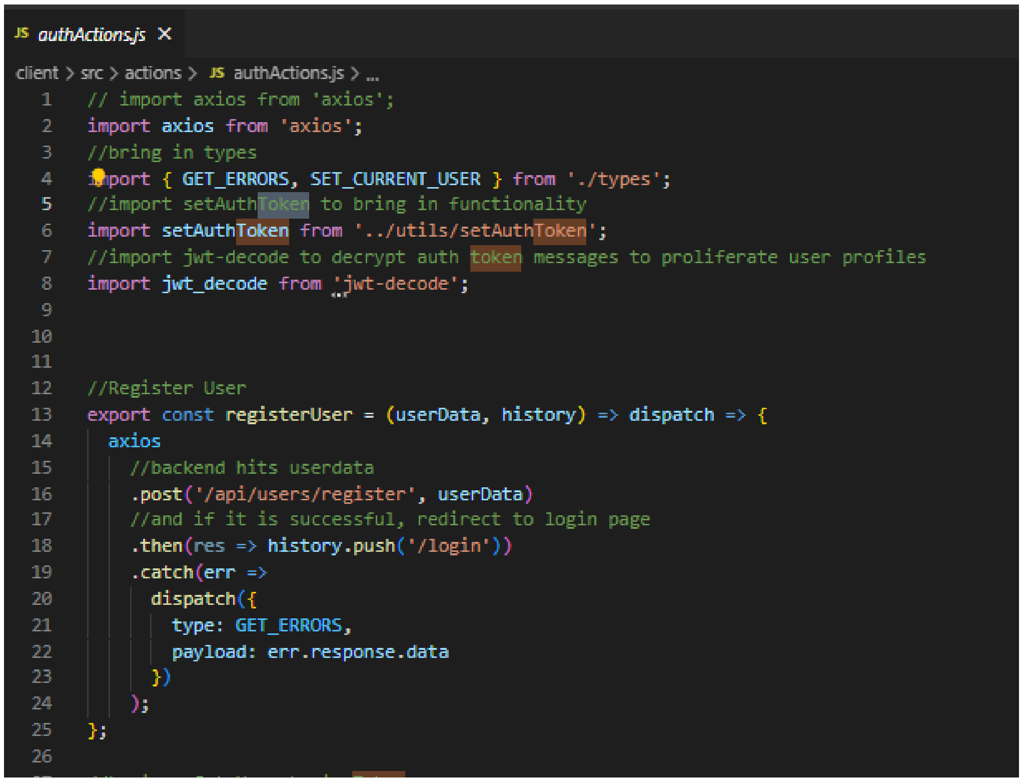
3. Se instala cryptjspara hash de contraseña



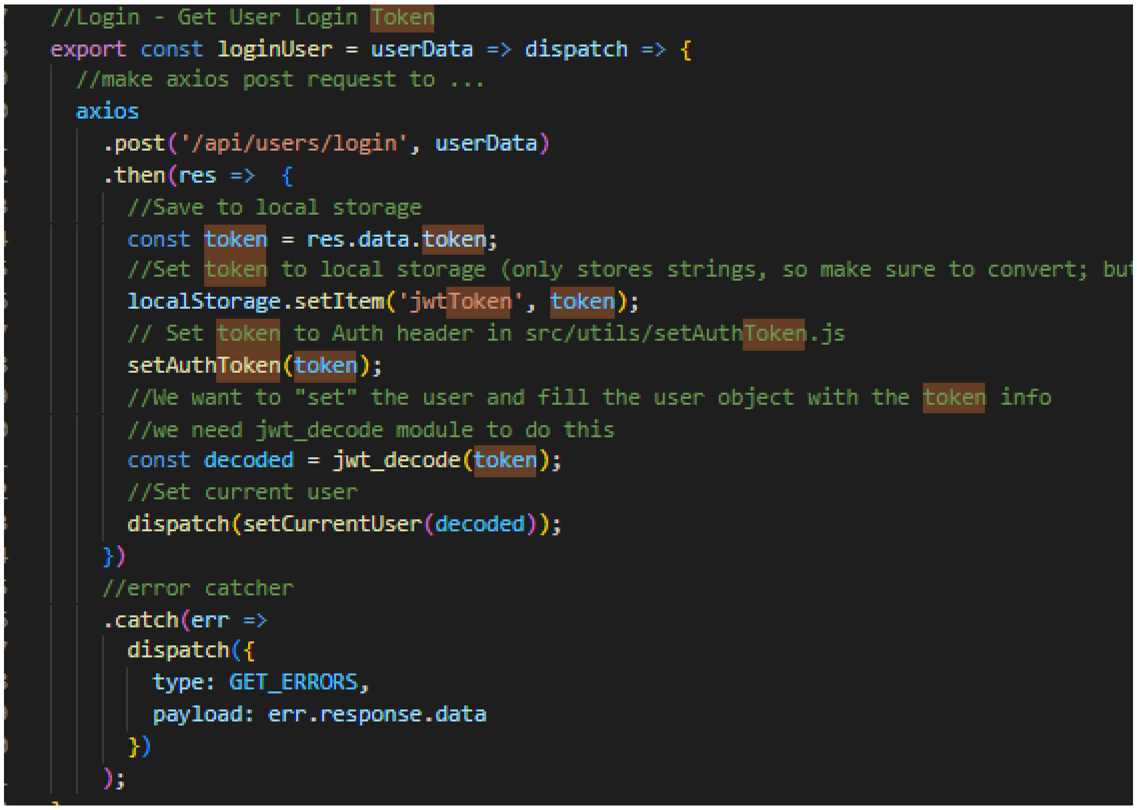
ent > src > utils > JS setAuthToken.js > ...

```javascript
//import axios to prevent us to manually make sure we have the token
import axios from 'axios';

const setAuthToken = token => {
  if (token) {
    //Apply token to every request
    axios.defaults.headers.common['Authorization'] = token;
  } else {
    //Delete Auth header if token is not there
    delete axios.defaults.headers.common['Authorization'];
  }
};

export default setAuthToken;
```

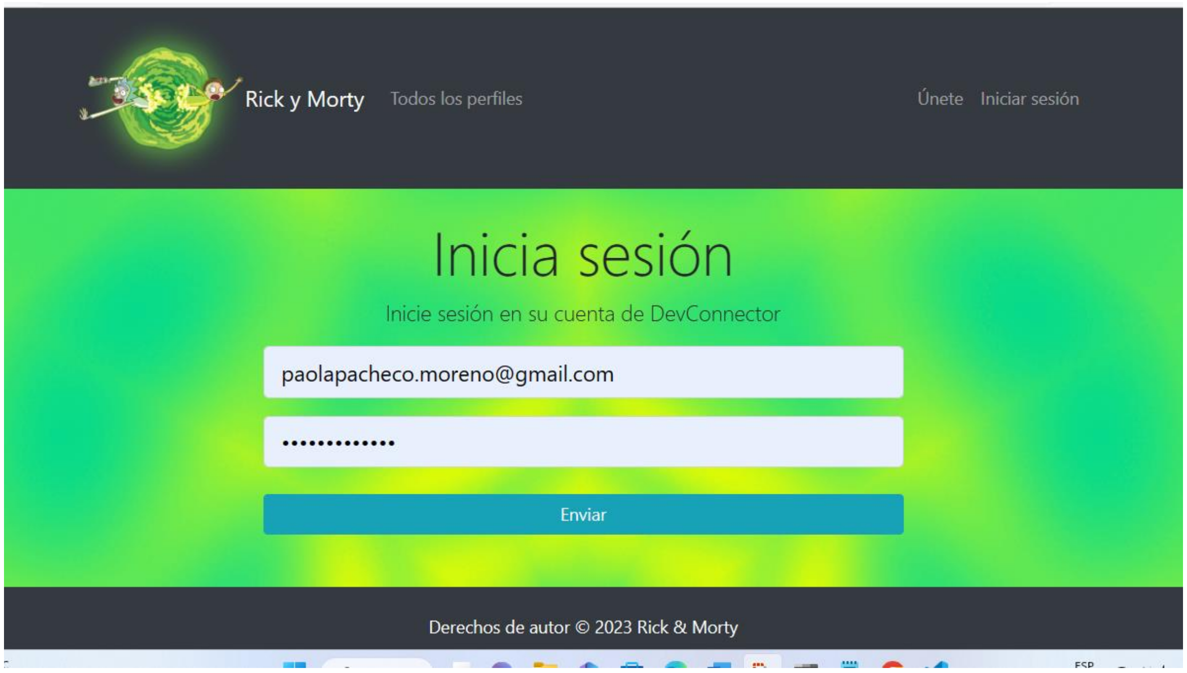

Rick y Morty
Todos los perfiles
Únete
Iniciar sesión
Rick y Morty
Vive una aventura con Rick & Morty y crea su álbum con sus perfiles favoritos, personaje y temporadas.
Únete
Iniciar sesión



```
authActions.js ×
client > src > actions > authActions.js > ...
// import axios from 'axios';
import axios from 'axios';
//bring in types
import { GET_ERRORS, SET_CURRENT_USER } from './types';
//import setAuthToken to bring in functionality
import setAuthToken from '../utils/setAuthToken';
//import jwt-decode to decrypt auth token messages to proliferate user profiles
import jwt_decode from 'jwt-decode';



//Register User
export const registerUser = (userData, history) => dispatch => {
  axios
    //backend hits userdata
    .post('/api/users/register', userData)
    //and if it is successful, redirect to login page
    .then(res => history.push('/login'))
    .catch(err =>
      dispatch({
        type: GET_ERRORS,
        payload: err.response.data
      })
    );
};

```



```
//Login - Get User Login Token
export const loginUser = userData => dispatch => {
  //make axios post request to ...
  axios
    .post('/api/users/login', userData)
    .then(res =>  {
      //Save to local storage
      const token = res.data.token;
      //Set token to local storage (only stores strings, so make sure to convert; but
      localStorage.setItem('jwtToken', token);
      // Set token to Auth header in src/utils/setAuthToken.js
      setAuthToken(token);
      //We want to "set" the user and fill the user object with the token info
      //we need jwt_decode module to do this
      const decoded = jwt_decode(token);
      //Set current user
      dispatch(setCurrentUser(decoded));
    })
    //error catcher
    .catch(err =>
      dispatch({
        type: GET_ERRORS,
        payload: err.response.data
      })
    );
```



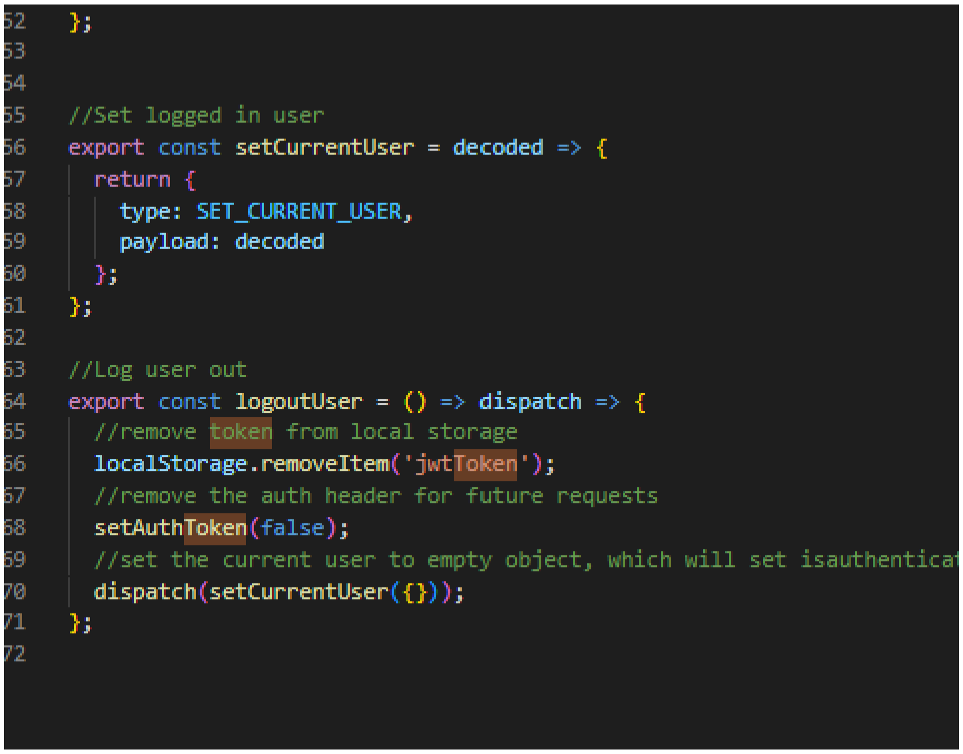
Inicia sesión

Inicie sesión en su cuenta de DevConnector

paolapacheco.moreno@gmail.com

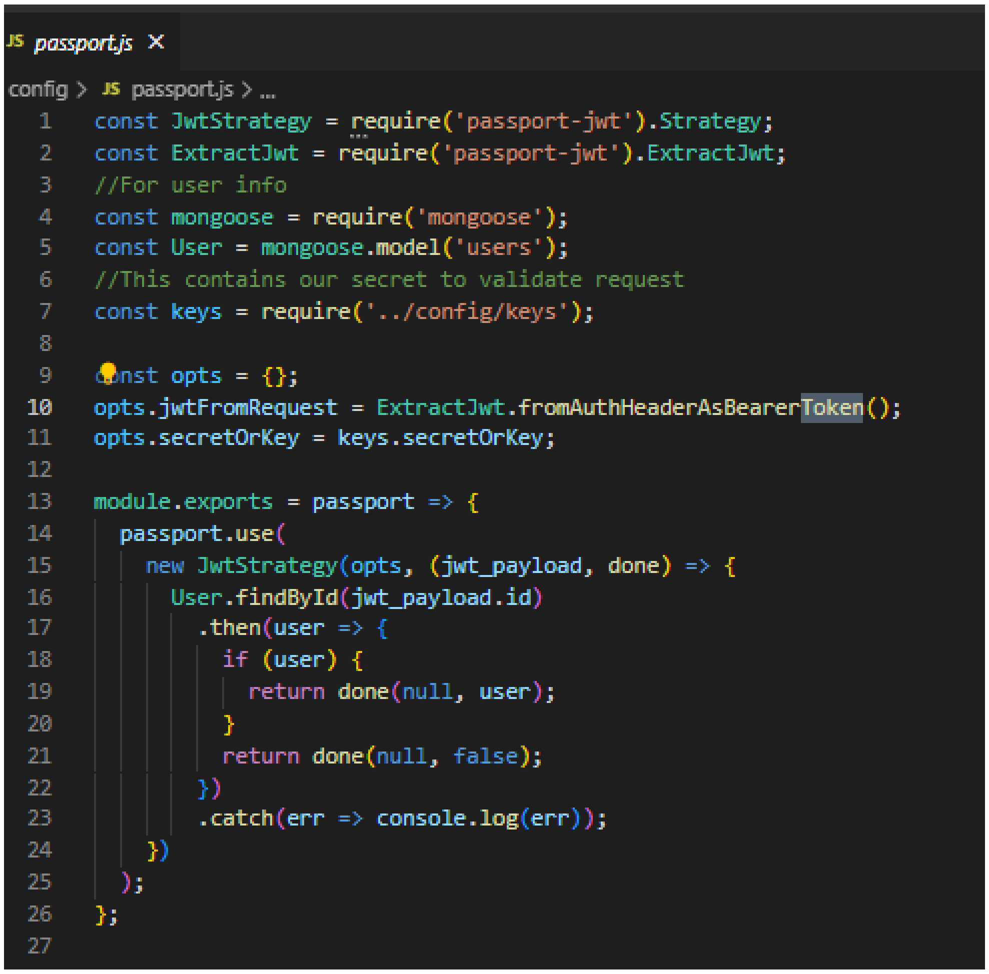
•••••••••••••

Derechos de autor © 2023 Rick & Morty

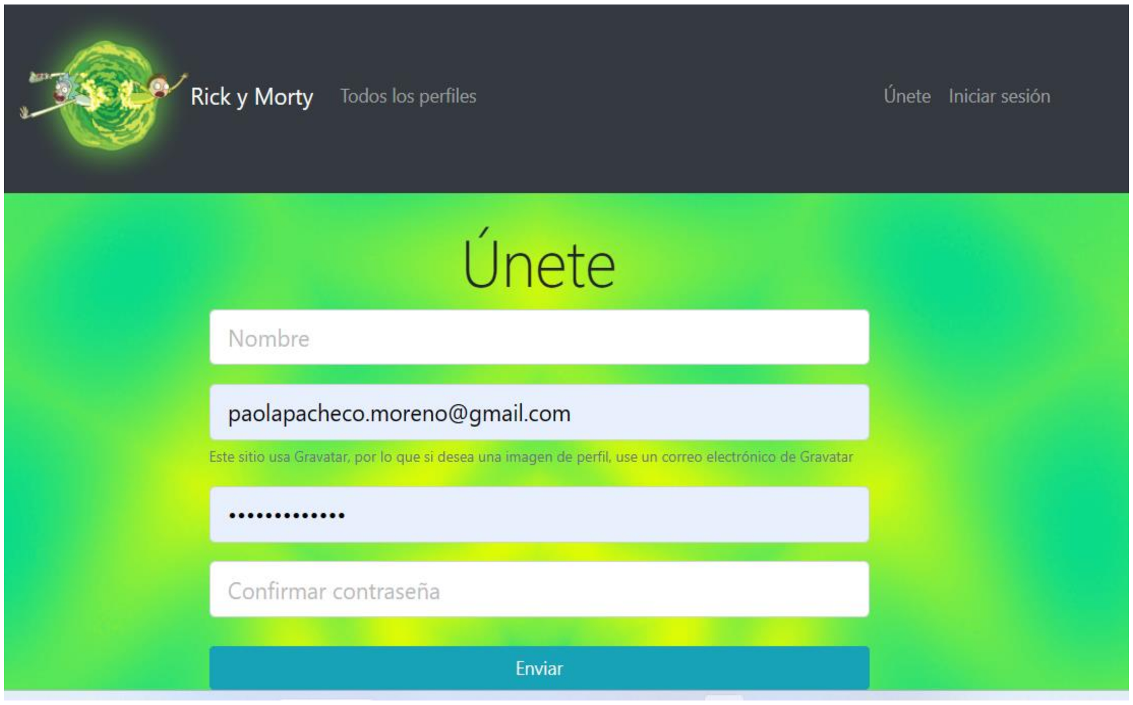


```
};


//Set logged in user
export const setCurrentUser = decoded => {
  return {
    type: SET_CURRENT_USER,
    payload: decoded
  };
};

//Log user out
export const logoutUser = () => dispatch => {
  //remove token from local storage
  localStorage.removeItem('jwtToken');
  //remove the auth header for future requests
  setAuthToken(false);
  //set the current user to empty object, which will set isauthenticat
  dispatch(setCurrentUser({}));
};

```

4. Utilizamos el correo electrónico y la contraseña para el inicio de sesión (Tokens)



```javascript
const JwtStrategy = require('passport-jwt').Strategy;
const ExtractJwt = require('passport-jwt').ExtractJwt;
//For user info
const mongoose = require('mongoose');
const User = mongoose.model('users');
//This contains our secret to validate request
const keys = require('../config/keys');

const opts = {};
opts.jwtFromRequest = ExtractJwt.fromAuthHeaderAsBearerToken();
opts.secretOrKey = keys.secretOrKey;

module.exports = passport => {
  passport.use(
    new JwtStrategy(opts, (jwt_payload, done) => {
      User.findById(jwt_payload.id)
        .then(user => {
          if (user) {
            return done(null, user);
          }
          return done(null, false);
        })
        .catch(err => console.log(err));
    })
  );
};
```



Únete

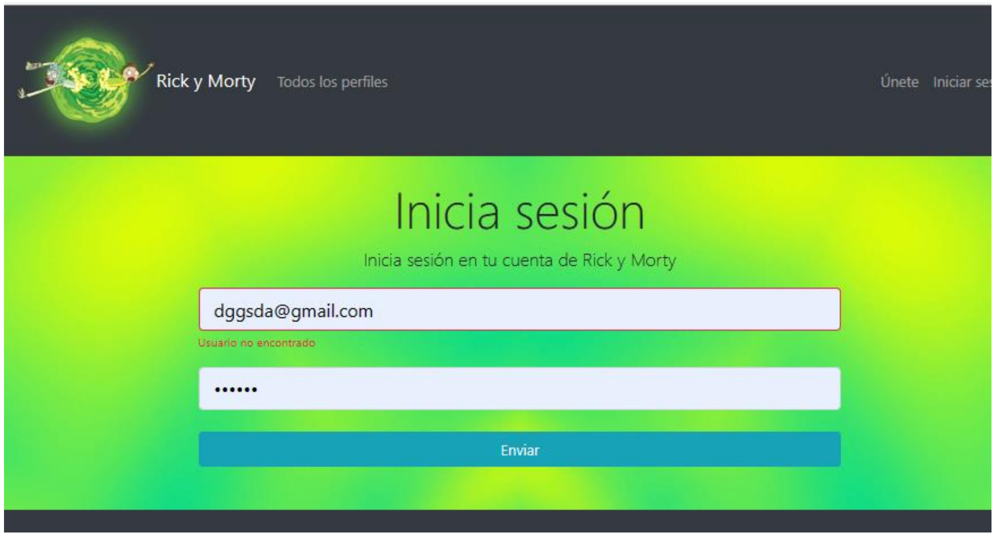
Nombre

paolapacheco.moreno@gmail.com

Este sitio usa Gravatar, por lo que si desea una imagen de perfil, use un correo electrónico de Gravatar

•••••••••••••

Confirmar contraseña



Inicia sesión

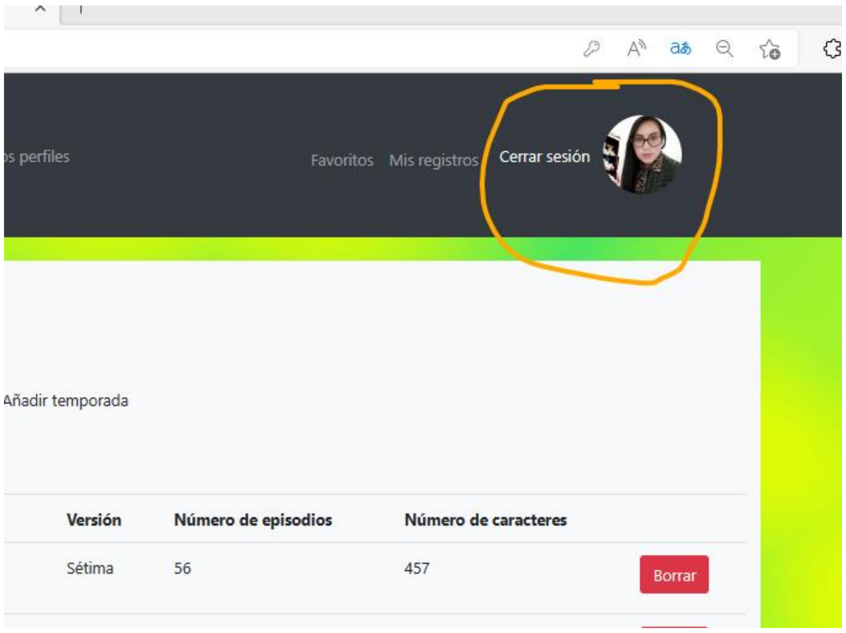
Inicia sesión en tu cuenta de Rick y Morty

dggsda@gmail.com

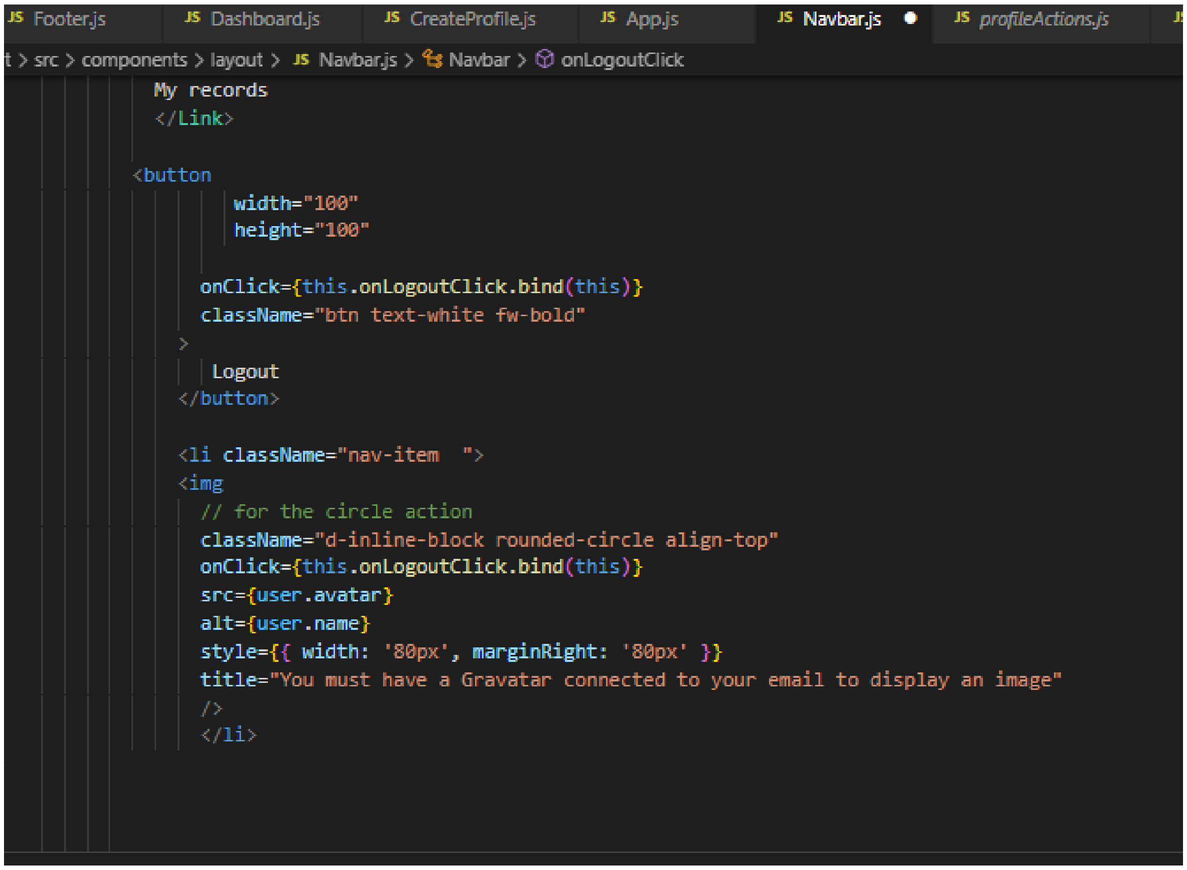
Usuario no encontrado

••••••

Cierre de sesión



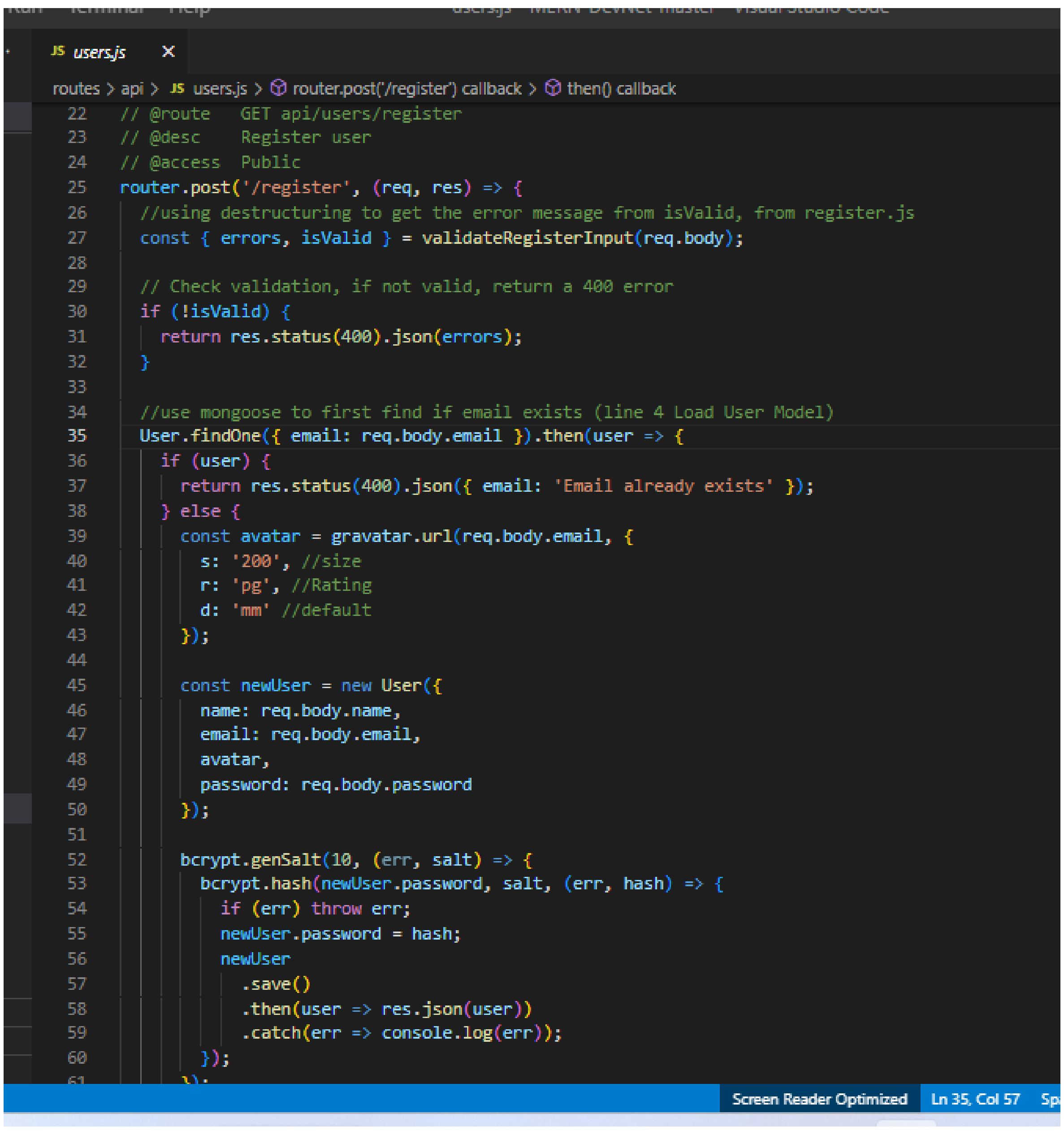
Versión	Número de episodios	Número de caracteres	
Sétima	56	457	Borrar



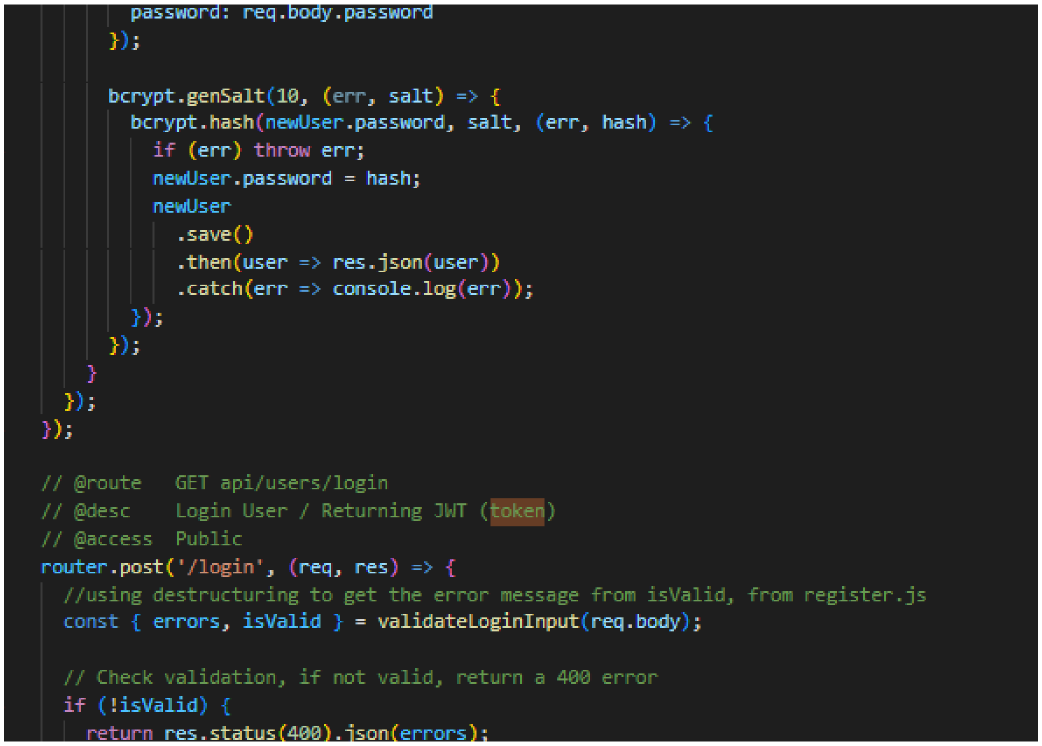
Footer.js Dashboard.js CreateProfile.js App.js Navbar.js profileActions.js

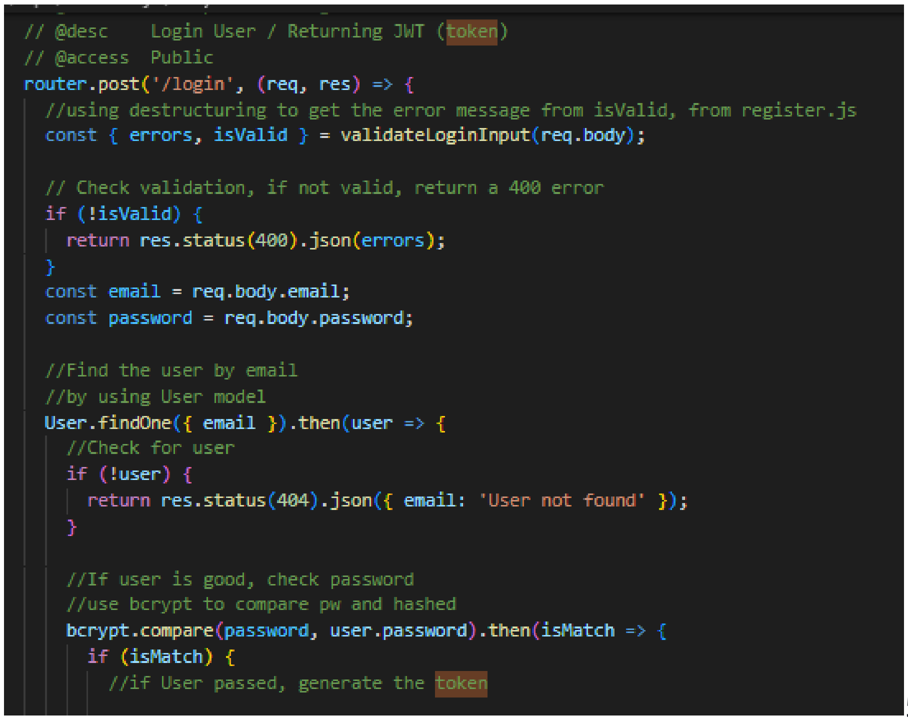
t > src > components > layout > Navbar.js > Navbar > onLogoutClick

```
            My records
            </Link>

          <button
                width="100"
                height="100"

              onClick={this.onLogoutClick.bind(this)}
              className="btn text-white fw-bold"
            >
              Logout
            </button>

            <li className="nav-item  ">
            <img
              // for the circle action
              className="d-inline-block rounded-circle align-top"
              onClick={this.onLogoutClick.bind(this)}
              src={user.avatar}
              alt={user.name}
              style={{ width: '80px', marginRight: '80px' }}
              title="You must have a Gravatar connected to your email to display an image"
              />
              </li>
```

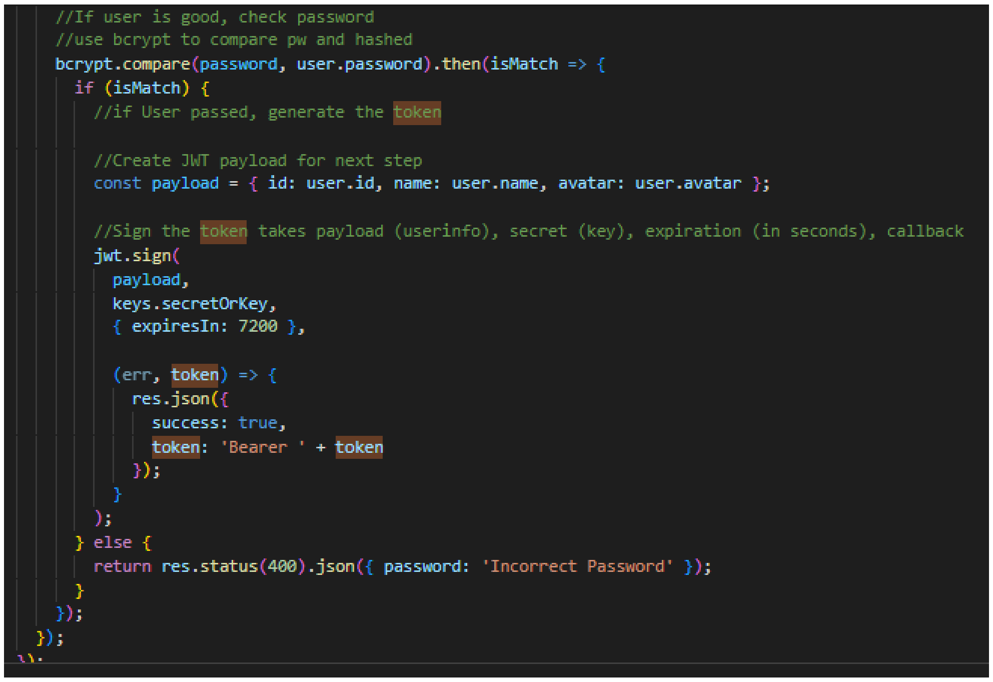

users.js
routes > api > users.js > router.post('/register') callback > then() callback
Screen Reader Optimized
Ln 35, Col 57

```
// @route   GET api/users/register
// @desc    Register user
// @access  Public
router.post('/register', (req, res) => {
  //using destructuring to get the error message from isValid, from register.js
  const { errors, isValid } = validateRegisterInput(req.body);

  // Check validation, if not valid, return a 400 error
  if (!isValid) {
    return res.status(400).json(errors);
  }

  //use mongoose to first find if email exists (line 4 Load User Model)
  User.findOne({ email: req.body.email }).then(user => {
    if (user) {
      return res.status(400).json({ email: 'Email already exists' });
    } else {
      const avatar = gravatar.url(req.body.email, {
        s: '200', //size
        r: 'pg', //Rating
        d: 'mm' //default
      });

      const newUser = new User({
        name: req.body.name,
        email: req.body.email,
        avatar,
        password: req.body.password
      });

      bcrypt.genSalt(10, (err, salt) => {
        bcrypt.hash(newUser.password, salt, (err, hash) => {
          if (err) throw err;
          newUser.password = hash;
          newUser
            .save()
            .then(user => res.json(user))
            .catch(err => console.log(err));
        });
```


```
            password: req.body.password
          });

          bcrypt.genSalt(10, (err, salt) => {
            bcrypt.hash(newUser.password, salt, (err, hash) => {
              if (err) throw err;
              newUser.password = hash;
              newUser
                .save()
                .then(user => res.json(user))
                .catch(err => console.log(err));
            });
          });
        }
      });
    });

    // @route   GET api/users/login
    // @desc    Login User / Returning JWT (token)
    // @access  Public
    router.post('/login', (req, res) => {
      //using destructuring to get the error message from isValid, from register.js
      const { errors, isValid } = validateLoginInput(req.body);

      // Check validation, if not valid, return a 400 error
      if (!isValid) {
        return res.status(400).json(errors);
```

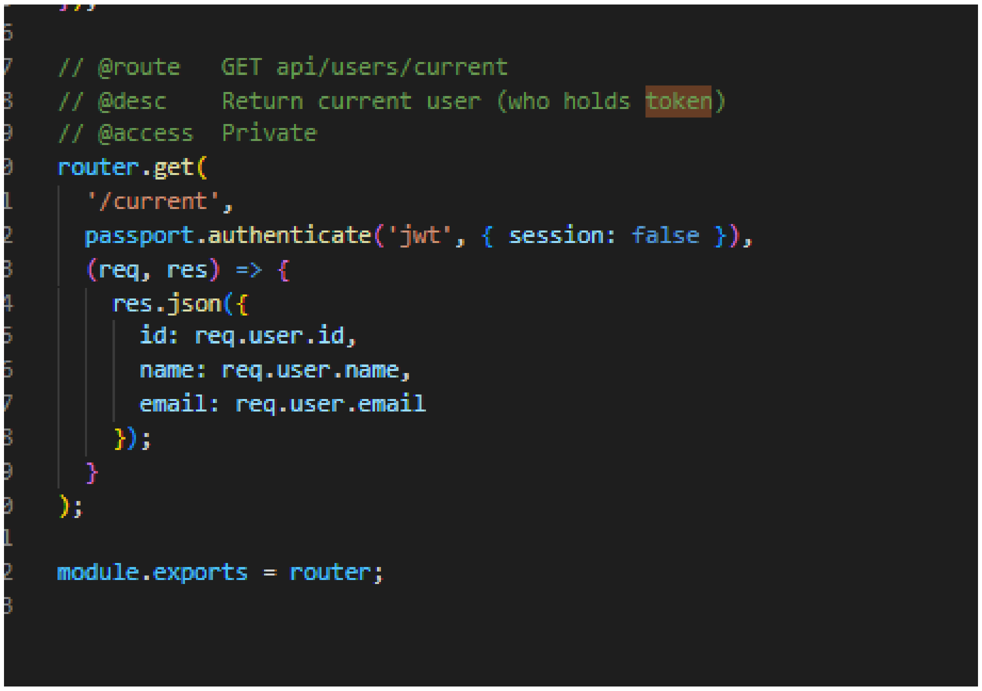

```
// @desc    Login User / Returning JWT (token)
// @access  Public
router.post('/login', (req, res) => {
  //using destructuring to get the error message from isValid, from register.js
  const { errors, isValid } = validateLoginInput(req.body);

  // Check validation, if not valid, return a 400 error
  if (!isValid) {
    return res.status(400).json(errors);
  }
  const email = req.body.email;
  const password = req.body.password;

  //Find the user by email
  //by using User model
  User.findOne({ email }).then(user => {
    //Check for user
    if (!user) {
      return res.status(404).json({ email: 'User not found' });
    }

    //If user is good, check password
    //use bcrypt to compare pw and hashed
    bcrypt.compare(password, user.password).then(isMatch => {
      if (isMatch) {
        //if User passed, generate the token
```

5. Se crea un JSON Webtoken (JWT) para iniciar sesión. Para esto creamos JWT importando dependencias y definiendo jwt.sing para aplicar el token:



```
      //If user is good, check password
      //use bcrypt to compare pw and hashed
      bcrypt.compare(password, user.password).then(isMatch => {
        if (isMatch) {
          //if User passed, generate the token

          //Create JWT payload for next step
          const payload = { id: user.id, name: user.name, avatar: user.avatar };

          //Sign the token takes payload (userinfo), secret (key), expiration (in seconds), callback
          jwt.sign(
            payload,
            keys.secretOrKey,
            { expiresIn: 7200 },

            (err, token) => {
              res.json({
                success: true,
                token: 'Bearer ' + token
              });
            }
          );
        } else {
          return res.status(400).json({ password: 'Incorrect Password' });
        }
      });
    });
```

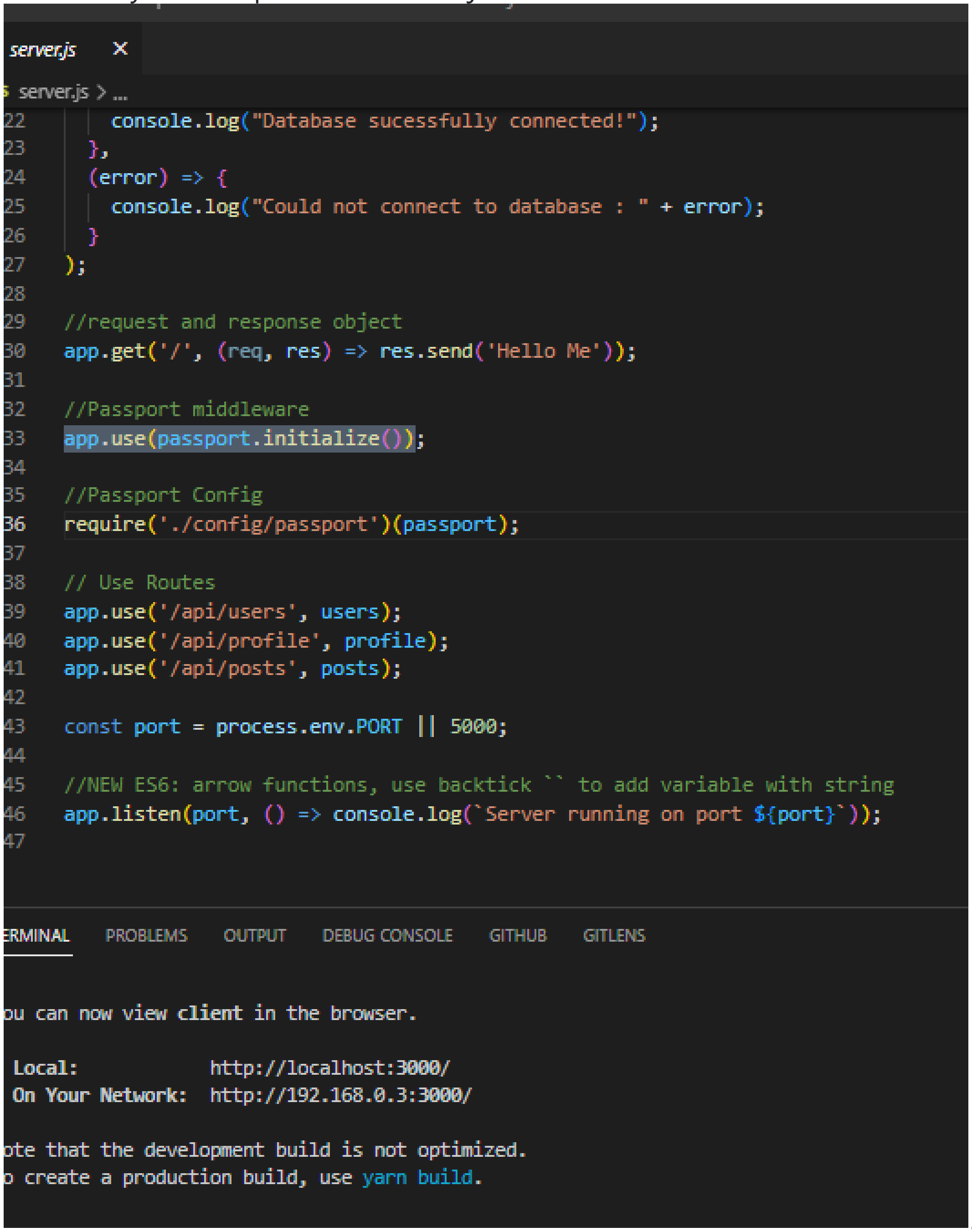
const jwt = require("jsonwebtoken");, se declara una clave dentro del directorio config, en la carpeta Key.js : secretOrKey: "secret"
y esta a su vez se importa en user.js



```
// @route   GET api/users/current
// @desc    Return current user (who holds token)
// @access  Private
router.get(
  '/current',
  passport.authenticate('jwt', { session: false }),
  (req, res) => {
    res.json({
      id: req.user.id,
      name: req.user.name,
      email: req.user.email
    });
  }
);

module.exports = router;
```

Implementación del Passport para autenticación JWT
Se verifica el token del paso anterior.

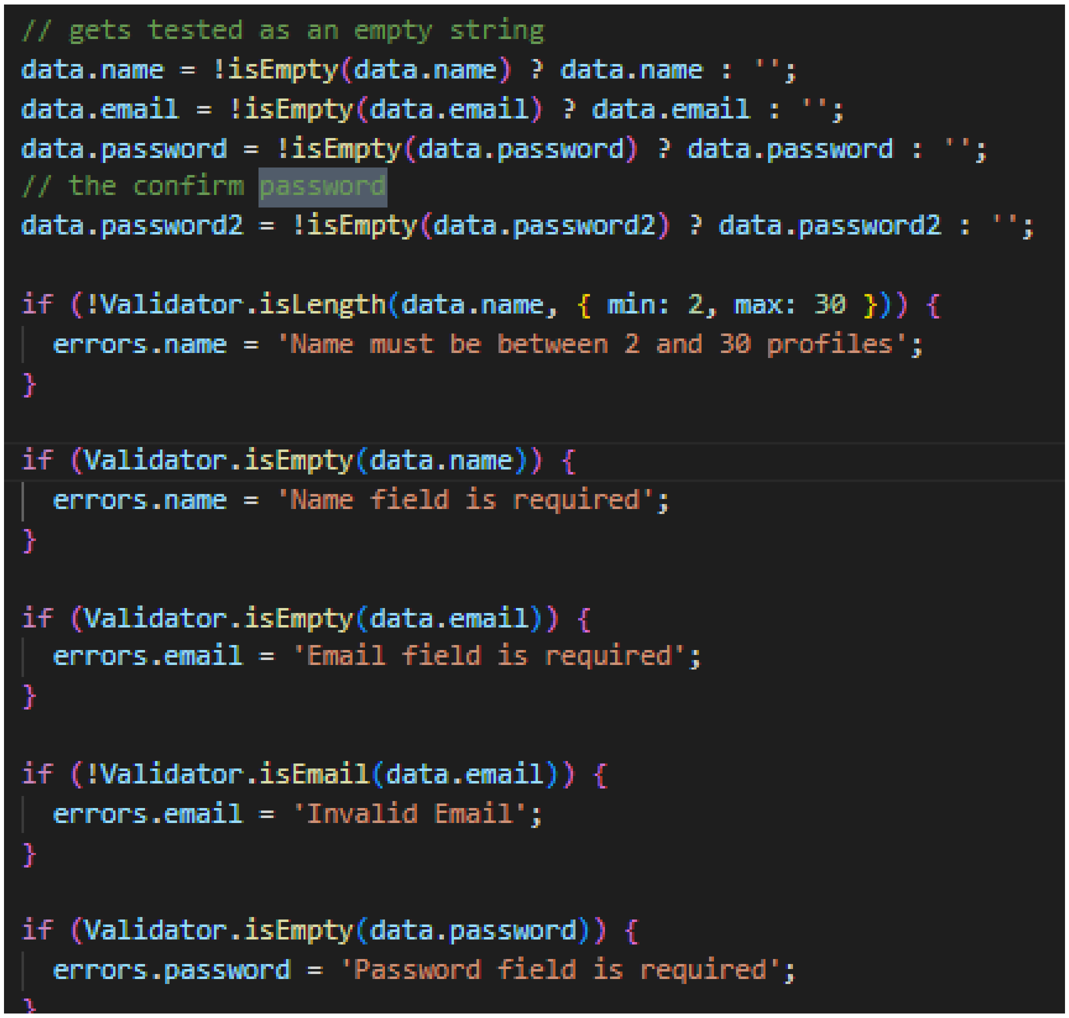
1. se incluye el Passport en el server.js



```
    console.log("Database sucessfully connected!");
  },
  (error) => {
    console.log("Could not connect to database : " + error);
  }
);

//request and response object
app.get('/', (req, res) => res.send('Hello Me'));

//Passport middleware
app.use(passport.initialize());

//Passport Config
require('./config/passport')(passport);

// Use Routes
app.use('/api/users', users);
app.use('/api/profile', profile);
app.use('/api/posts', posts);

const port = process.env.PORT || 5000;

//NEW ES6: arrow functions, use backtick `` to add variable with string
app.listen(port, () => console.log(`Server running on port ${port}`));
```

```
ou can now view client in the browser.

 Local:            http://localhost:3000/
 On Your Network:  http://192.168.0.3:3000/

ote that the development build is not optimized.
o create a production build, use yarn build.
```

2. Se actualiza el User.js y el Passport.js con el token JWT
3. Se usa el validator.js en el register.js, para validar los posibles errores y el is-empaty.js para comprobar si hay algo vacio.
const Validator = require("validator");



```
// gets tested as an empty string
data.name = !isEmpty(data.name) ? data.name : '';
data.email = !isEmpty(data.email) ? data.email : '';
data.password = !isEmpty(data.password) ? data.password : '';
// the confirm password
data.password2 = !isEmpty(data.password2) ? data.password2 : '';

if (!Validator.isLength(data.name, { min: 2, max: 30 })) {
  errors.name = 'Name must be between 2 and 30 profiles';
}

if (Validator.isEmpty(data.name)) {
  errors.name = 'Name field is required';
}

if (Validator.isEmpty(data.email)) {
  errors.email = 'Email field is required';
}

if (!Validator.isEmail(data.email)) {
  errors.email = 'Invalid Email';
}

if (Validator.isEmpty(data.password)) {
  errors.password = 'Password field is required';
```

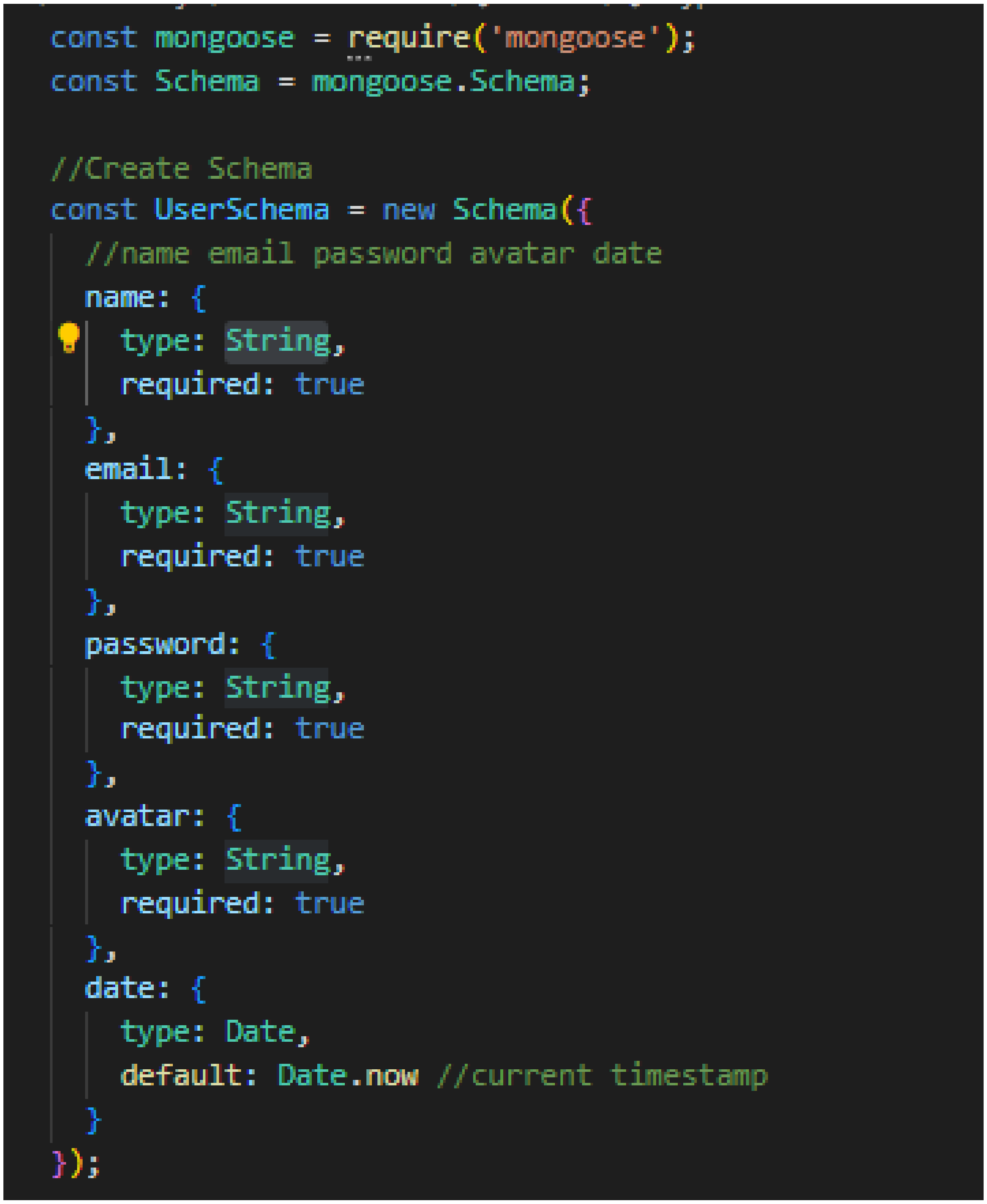
};

};

Definición de API s Rutas

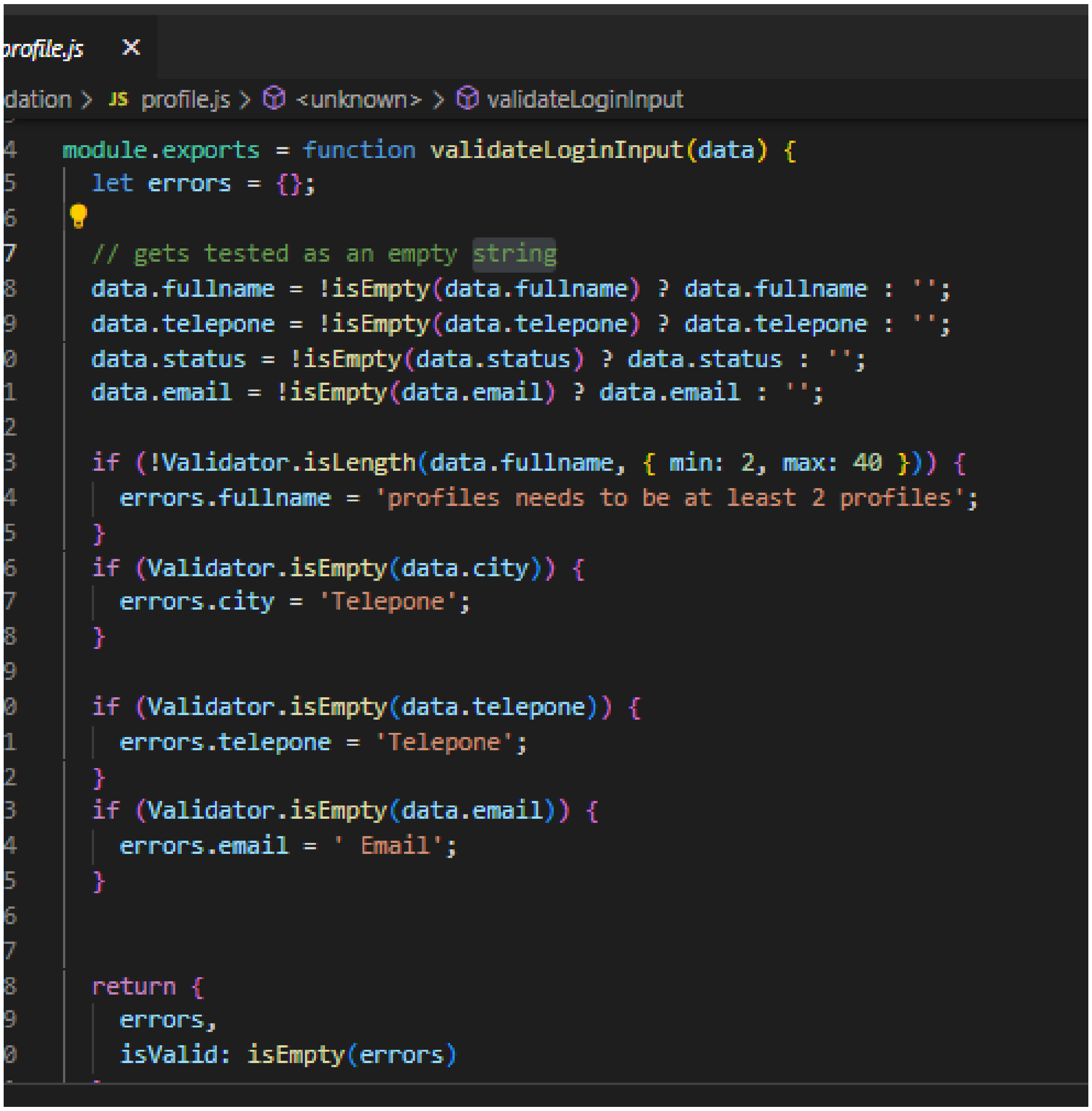
1. Se instala mongoose y se define dentro de cada uno de los archivos .js de models : Post.js, profile.js y user.js, Es decir se enrutan los modelos de perfil, usuario y post

const mongoose = require('mongoose');
const Schema = mongoose.Schema;



```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

//Create Schema
const UserSchema = new Schema({
  //name email password avatar date
  name: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true
  },
  password: {
    type: String,
    required: true
  },
  avatar: {
    type: String,
    required: true
  },
  date: {
    type: Date,
    default: Date.now //current timestamp
  }
});
```

2. A través del Validator hacemos validaciones para profile.js, charcter.js, season.js, login.js, post.js y register.js, se prueba como una cadena vacía cada una de las variables.



```javascript
module.exports = function validateLoginInput(data) {
  let errors = {};

  // gets tested as an empty string
  data.fullname = !isEmpty(data.fullname) ? data.fullname : '';
  data.telepone = !isEmpty(data.telepone) ? data.telepone : '';
  data.status = !isEmpty(data.status) ? data.status : '';
  data.email = !isEmpty(data.email) ? data.email : '';

  if (!Validator.isLength(data.fullname, { min: 2, max: 40 })) {
    errors.fullname = 'profiles needs to be at least 2 profiles';
  }
  if (Validator.isEmpty(data.city)) {
    errors.city = 'Telepone';
  }

  if (Validator.isEmpty(data.telepone)) {
    errors.telepone = 'Telepone';
  }
  if (Validator.isEmpty(data.email)) {
    errors.email = ' Email';
  }


  return {
    errors,
    isValid: isEmpty(errors)
```

3. Se obtiene el perfil por identificador y adicional todos lo perfiles

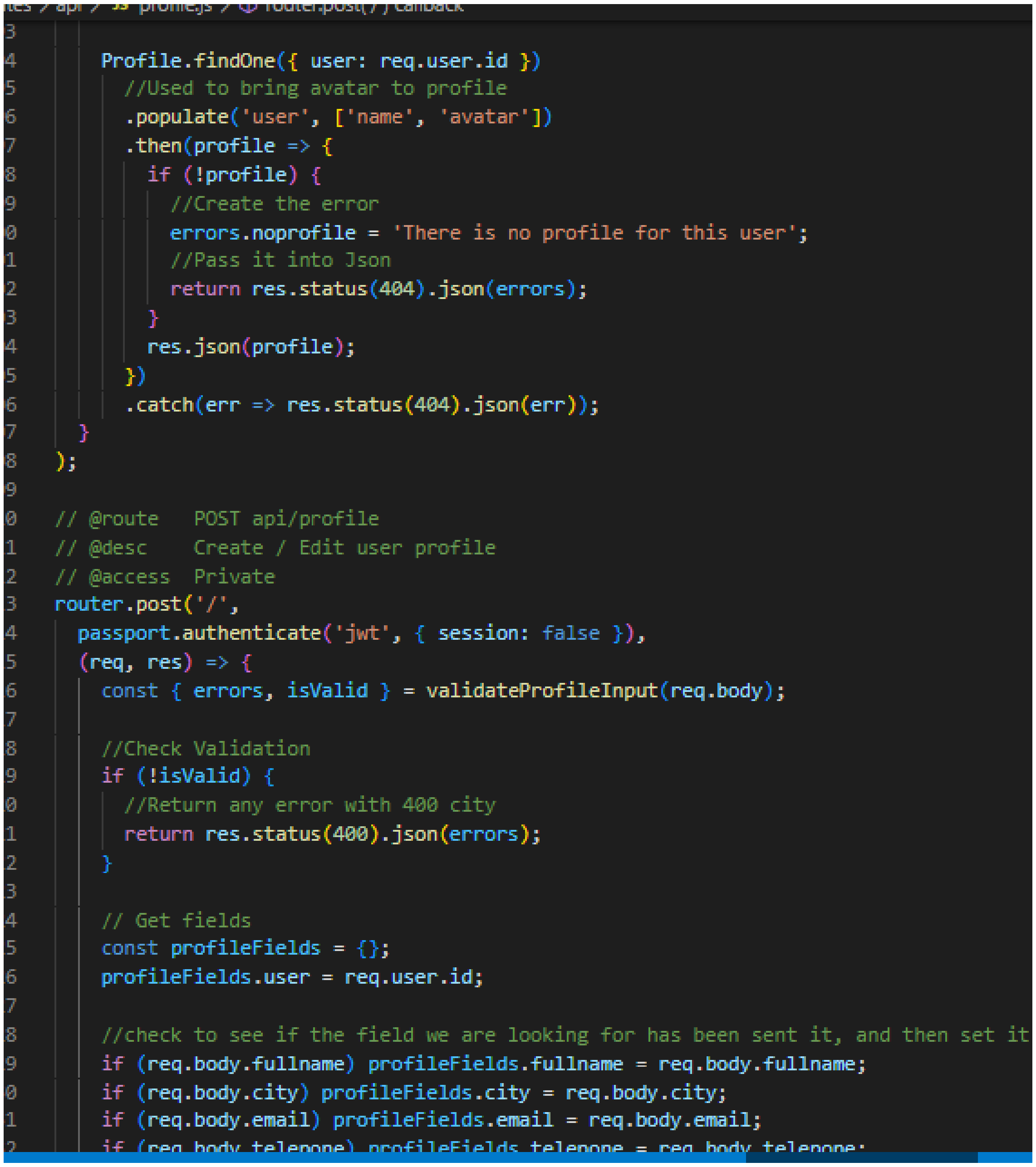
POST

1. A través del POST de generan nuevas solicitudes y verificar validaciones.

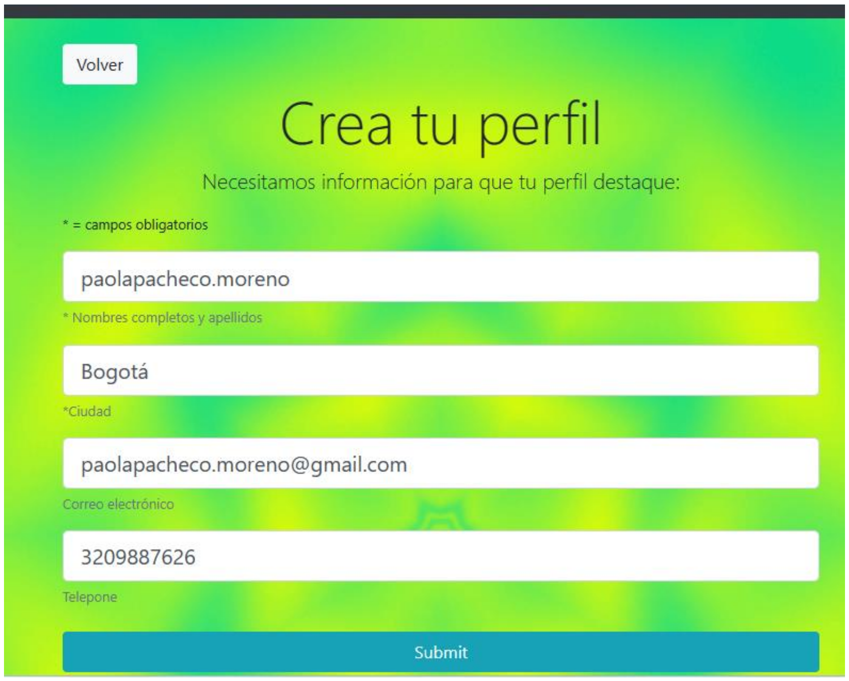
2. Se crean APIs para llevar a cabo la funcionalidad total del programa, esto según los requerimientos funcionales y no funcionales

`1. api/profile.js`

`Creación perfil`



```
      Profile.findOne({ user: req.user.id })
        //Used to bring avatar to profile
        .populate('user', ['name', 'avatar'])
        .then(profile => {
          if (!profile) {
            //Create the error
            errors.noprofile = 'There is no profile for this user';
            //Pass it into Json
            return res.status(404).json(errors);
          }
          res.json(profile);
        })
        .catch(err => res.status(404).json(err));
    }
);

// @route   POST api/profile
// @desc    Create / Edit user profile
// @access  Private
router.post('/',
  passport.authenticate('jwt', { session: false }),
  (req, res) => {
    const { errors, isValid } = validateProfileInput(req.body);

    //Check Validation
    if (!isValid) {
      //Return any error with 400 city
      return res.status(400).json(errors);
    }

    // Get fields
    const profileFields = {};
    profileFields.user = req.user.id;

    //check to see if the field we are looking for has been sent it, and then set it
    if (req.body.fullname) profileFields.fullname = req.body.fullname;
    if (req.body.city) profileFields.city = req.body.city;
    if (req.body.email) profileFields.email = req.body.email;
```



Volver

Crea tu perfil

Necesitamos información para que tu perfil destaque:

* = campos obligatorios

paolapacheco.moreno

* Nombres completos y apellidos

Bogotá

*Ciudad

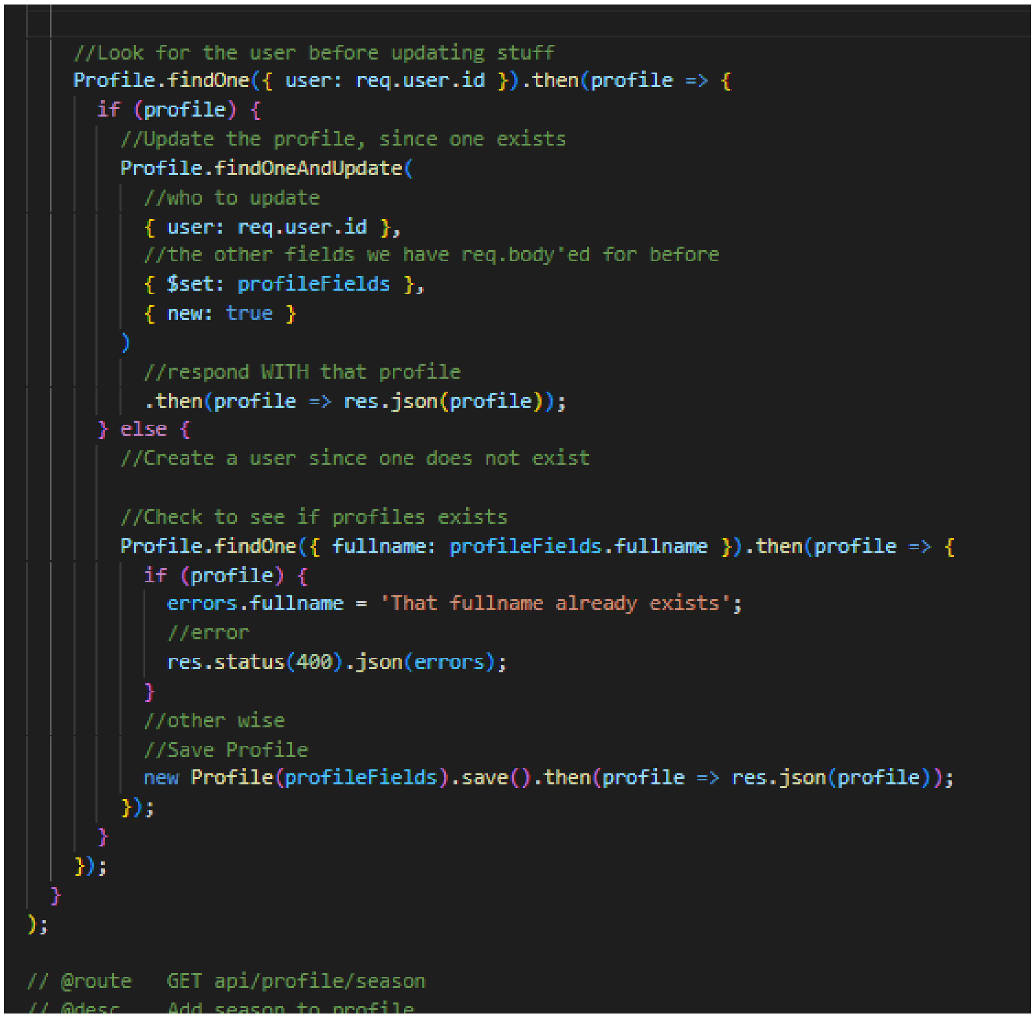
paolapacheco.moreno@gmail.com

Correo electrónico

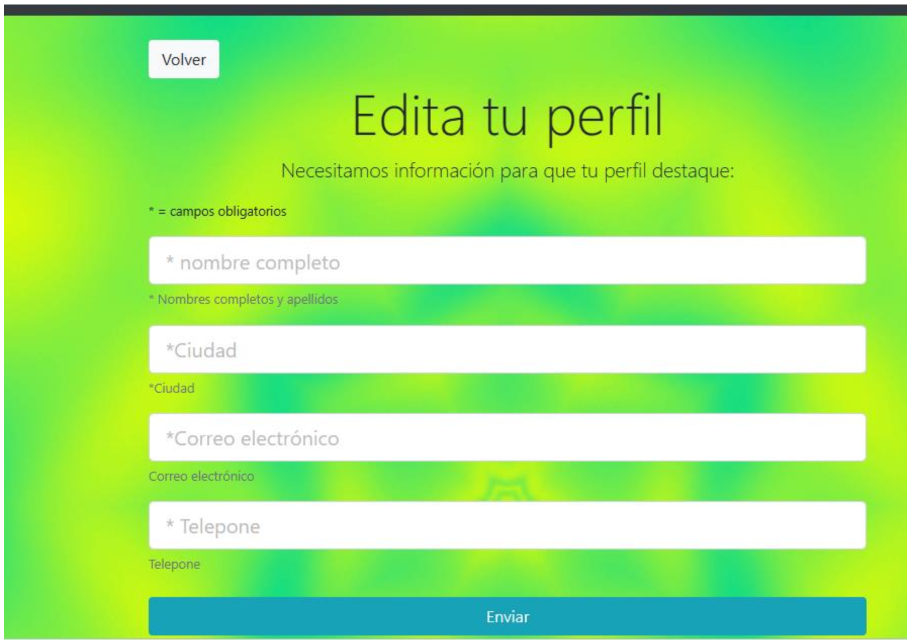
3209887626

Telepone

Submit



```
    //Look for the user before updating stuff
    Profile.findOne({ user: req.user.id }).then(profile => {
      if (profile) {
        //Update the profile, since one exists
        Profile.findOneAndUpdate(
          //who to update
          { user: req.user.id },
          //the other fields we have req.body'ed for before
          { $set: profileFields },
          { new: true }
        )
          //respond WITH that profile
          .then(profile => res.json(profile));
      } else {
        //Create a user since one does not exist

        //Check to see if profiles exists
        Profile.findOne({ fullname: profileFields.fullname }).then(profile => {
          if (profile) {
            errors.fullname = 'That fullname already exists';
            //error
            res.status(400).json(errors);
          }
          //other wise
          //Save Profile
          new Profile(profileFields).save().then(profile => res.json(profile));
        });
      }
    });
  }
);

// @route   GET api/profile/season
```



Volver

Edita tu perfil

Necesitamos información para que tu perfil destaque:

* = campos obligatorios

* nombre completo

* Nombres completos y apellidos

*Ciudad

*Ciudad

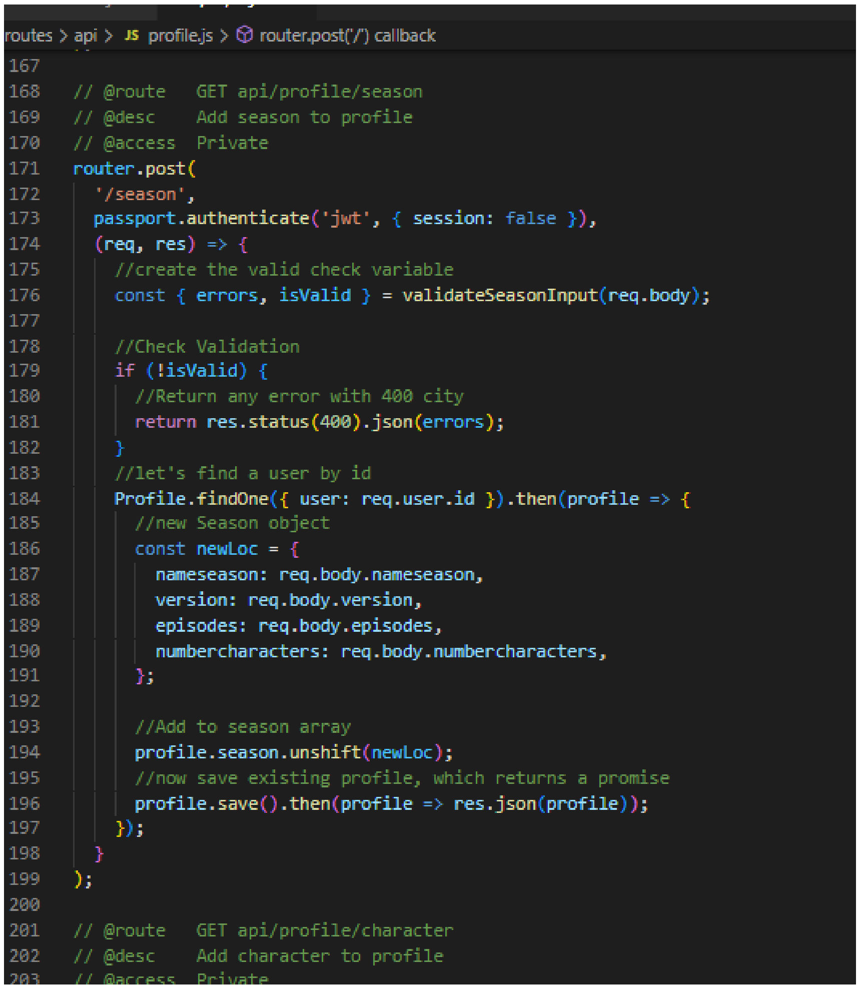
*Correo electrónico

Correo electrónico

* Telepone

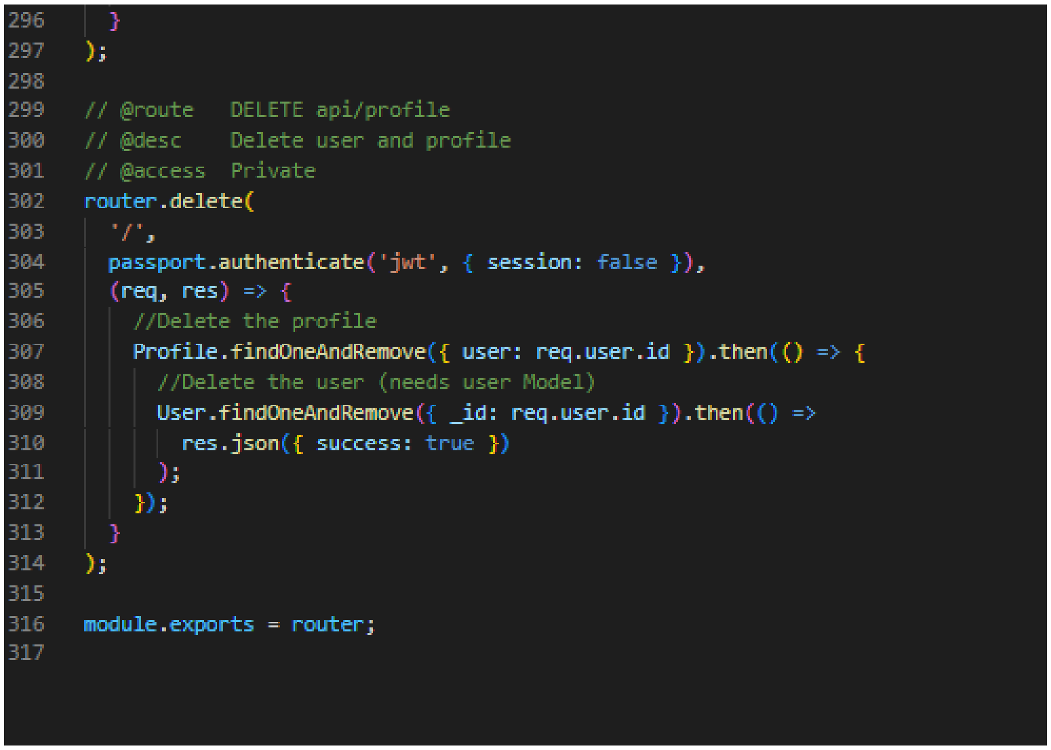
Telepone

Enviar



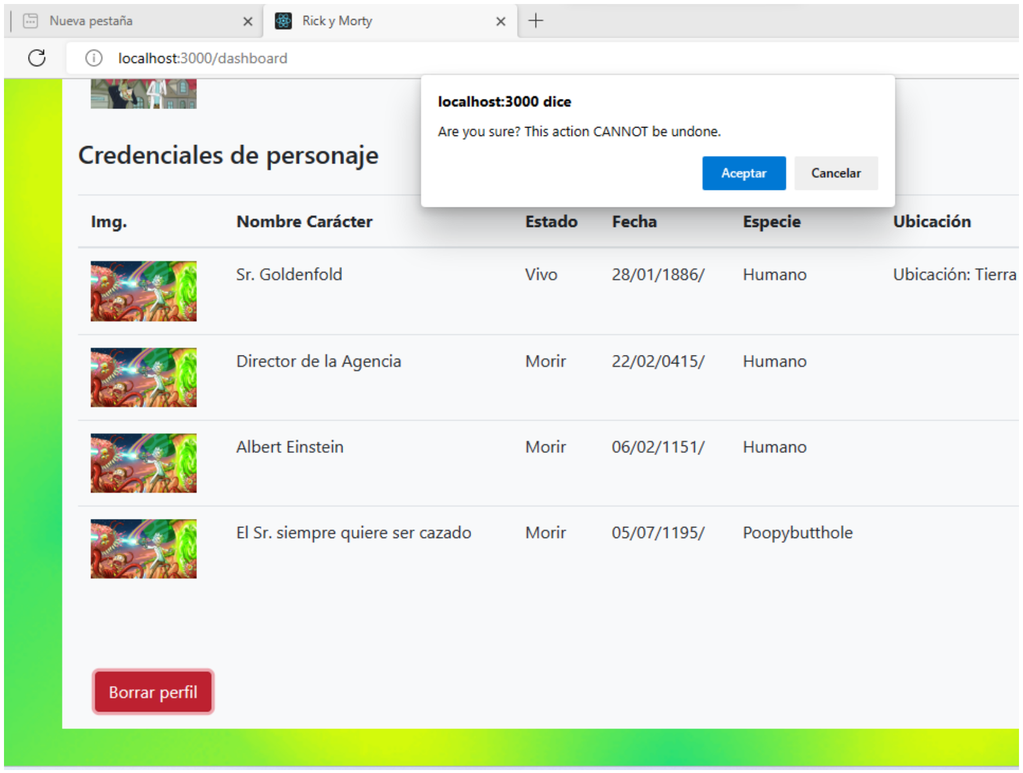
routes > api > JS profile.js > router.post('/') callback

```
// @route   GET api/profile/season
// @desc    Add season to profile
// @access  Private
router.post(
  '/season',
  passport.authenticate('jwt', { session: false }),
  (req, res) => {
    //create the valid check variable
    const { errors, isValid } = validateSeasonInput(req.body);

    //Check Validation
    if (!isValid) {
      //Return any error with 400 city
      return res.status(400).json(errors);
    }
    //let's find a user by id
    Profile.findOne({ user: req.user.id }).then(profile => {
      //new Season object
      const newLoc = {
        nameseason: req.body.nameseason,
        version: req.body.version,
        episodes: req.body.episodes,
        numbercharacters: req.body.numbercharacters,
      };

      //Add to season array
      profile.season.unshift(newLoc);
      //now save existing profile, which returns a promise
      profile.save().then(profile => res.json(profile));
    });
  }
);

// @route   GET api/profile/character
// @desc    Add character to profile
// @access  Private
```



```
  }
);

// @route   DELETE api/profile
// @desc    Delete user and profile
// @access  Private
router.delete(
  '/',
  passport.authenticate('jwt', { session: false }),
  (req, res) => {
    //Delete the profile
    Profile.findOneAndRemove({ user: req.user.id }).then(() => {
      //Delete the user (needs user Model)
      User.findOneAndRemove({ _id: req.user.id }).then(() =>
        res.json({ success: true })
      );
    });
  }
);

module.exports = router;
```

Borrar Perfil



localhost:3000 dice

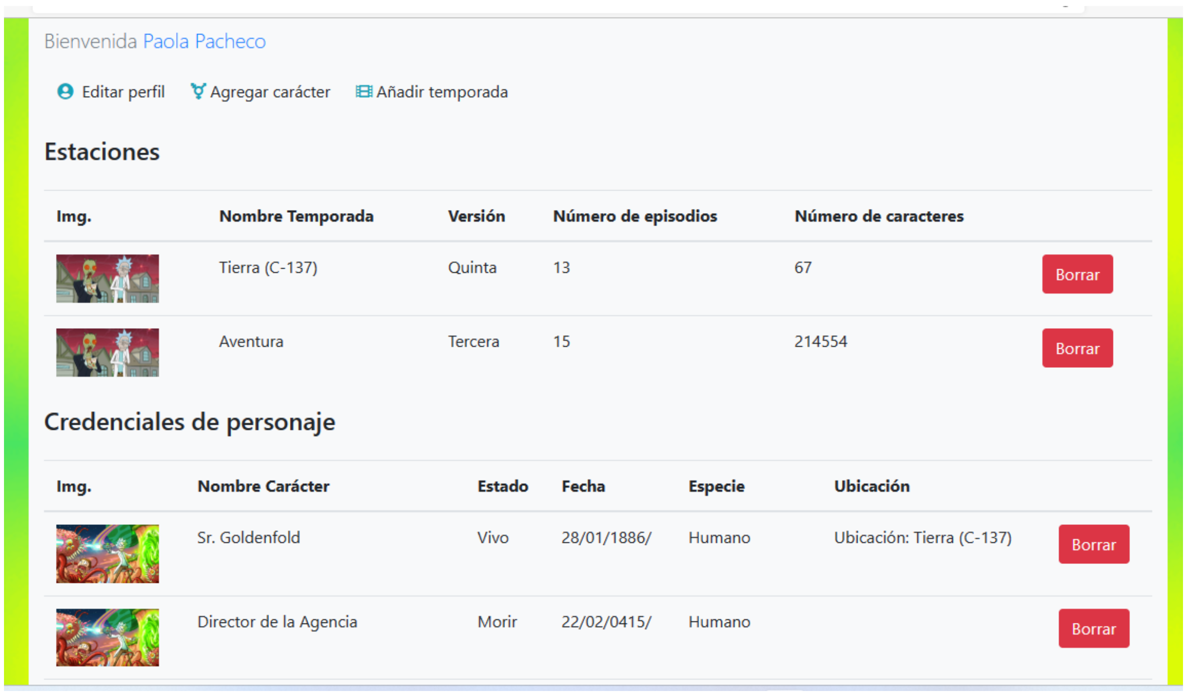
Are you sure? This action CANNOT be undone.

Aceptar Cancelar

Credenciales de personaje

Img.	Nombre Carácter	Estado	Fecha	Especie	Ubicación
	Sr. Goldenfold	Vivo	28/01/1886/	Humano	Ubicación: Tierra
	Director de la Agencia	Morir	22/02/0415/	Humano	
	Albert Einstein	Morir	06/02/1151/	Humano	
	El Sr. siempre quiere ser cazado	Morir	05/07/1195/	Poopybutthole	

Borrar perfil



Bienvenida Paola Pacheco

Editar perfil Agregar carácter Añadir temporada

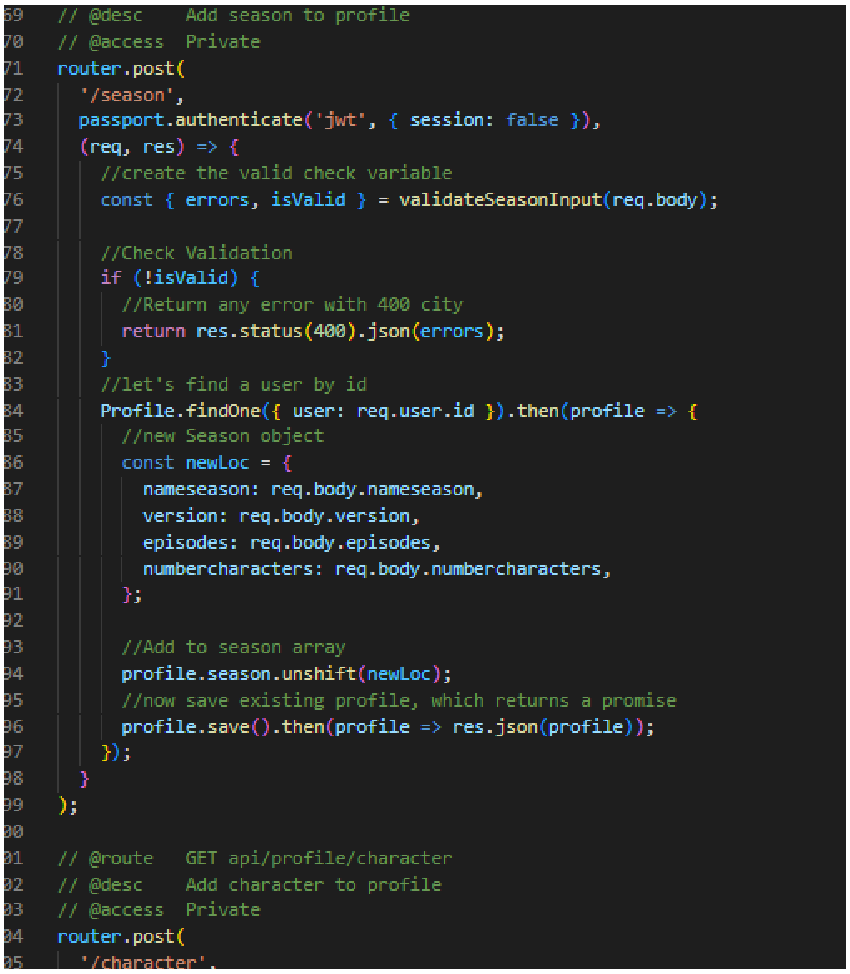
Estaciones

Img.	Nombre Temporada	Versión	Número de episodios	Número de caracteres	
	Tierra (C-137)	Quinta	13	67	Borrar
	Aventura	Tercera	15	214554	Borrar

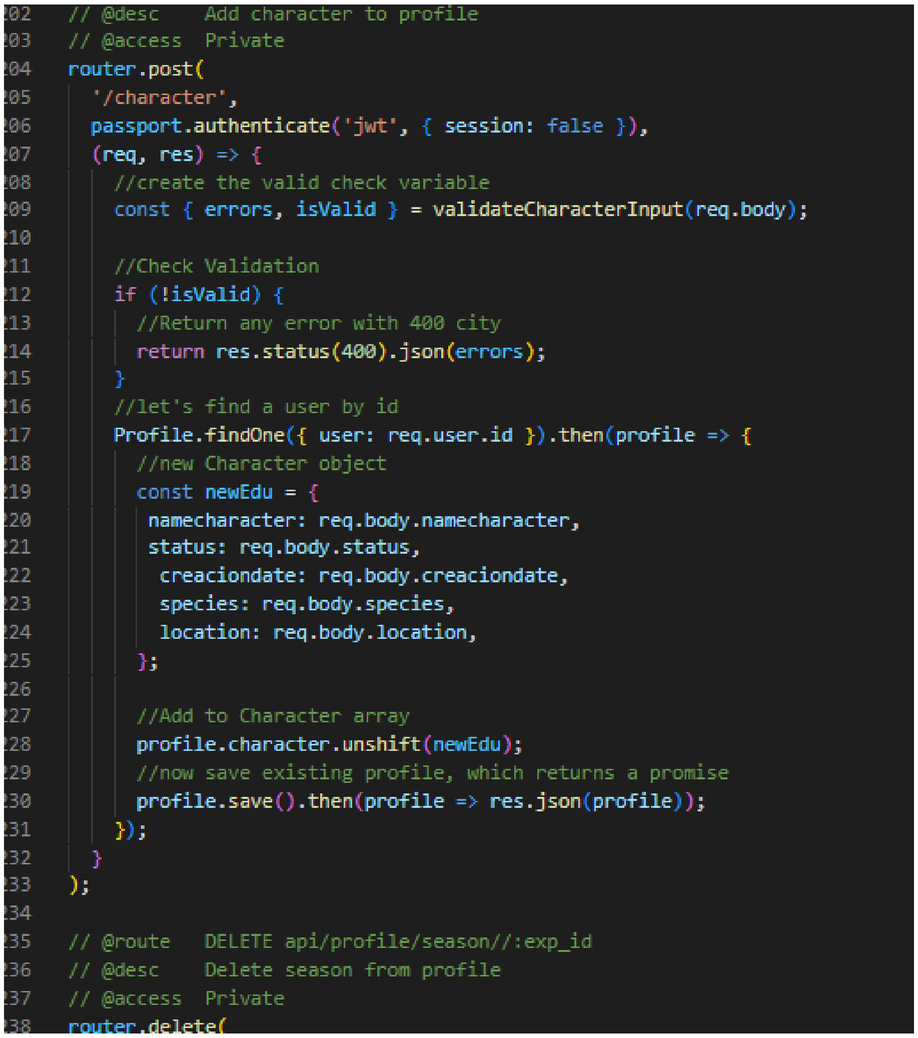
Credenciales de personaje

Img.	Nombre Carácter	Estado	Fecha	Especie	Ubicación	
	Sr. Goldenfold	Vivo	28/01/1886/	Humano	Ubicación: Tierra (C-137)	Borrar
	Director de la Agencia	Morir	22/02/0415/	Humano		Borrar

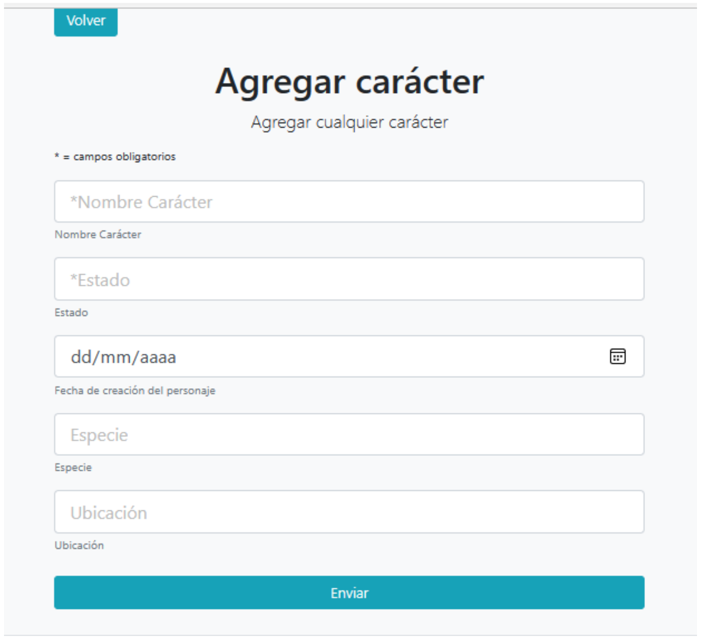
Api para crear Character y Season



```
// @desc    Add season to profile
// @access  Private
router.post(
  '/season',
  passport.authenticate('jwt', { session: false }),
  (req, res) => {
    //create the valid check variable
    const { errors, isValid } = validateSeasonInput(req.body);

    //Check Validation
    if (!isValid) {
      //Return any error with 400 city
      return res.status(400).json(errors);
    }
    //let's find a user by id
    Profile.findOne({ user: req.user.id }).then(profile => {
      //new Season object
      const newLoc = {
        nameseason: req.body.nameseason,
        version: req.body.version,
        episodes: req.body.episodes,
        numbercharacters: req.body.numbercharacters,
      };

      //Add to season array
      profile.season.unshift(newLoc);
      //now save existing profile, which returns a promise
      profile.save().then(profile => res.json(profile));
    });
  }
);

// @route   GET api/profile/character
// @desc    Add character to profile
// @access  Private
router.post(
  '/character',
```



```
// @desc    Add character to profile
// @access  Private
router.post(
  '/character',
  passport.authenticate('jwt', { session: false }),
  (req, res) => {
    //create the valid check variable
    const { errors, isValid } = validateCharacterInput(req.body);

    //Check Validation
    if (!isValid) {
      //Return any error with 400 city
      return res.status(400).json(errors);
    }
    //let's find a user by id
    Profile.findOne({ user: req.user.id }).then(profile => {
      //new Character object
      const newEdu = {
       namecharacter: req.body.namecharacter,
       status: req.body.status,
        creaciondate: req.body.creaciondate,
        species: req.body.species,
        location: req.body.location,
      };

      //Add to Character array
      profile.character.unshift(newEdu);
      //now save existing profile, which returns a promise
      profile.save().then(profile => res.json(profile));
    });
  }
);

// @route   DELETE api/profile/season//:exp_id
// @desc    Delete season from profile
// @access  Private
router.delete(
```



Volver

Agregar carácter

Agregar cualquier carácter

* = campos obligatorios

*Nombre Carácter

Nombre Carácter

*Estado

Estado

dd/mm/aaaa

Fecha de creación del personaje

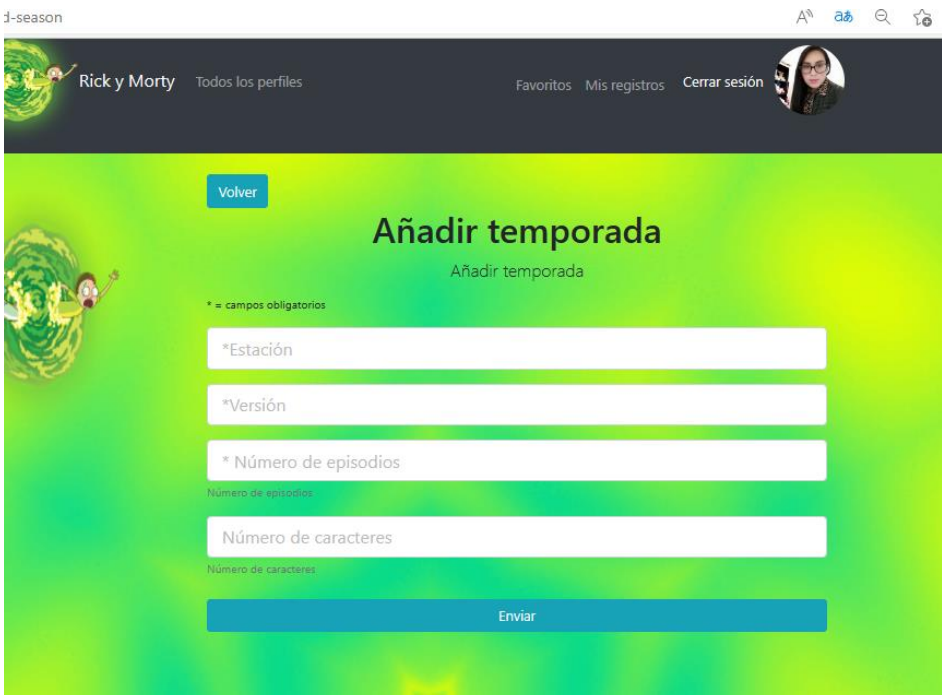
Especie

Especie

Ubicación

Ubicación

Enviar


d-season

Rick y Morty Todos los perfiles Favoritos Mis registros Cerrar sesión

Volver

Añadir temporada

Añadir temporada

* = campos obligatorios

*Estación

*Versión

* Número de episodios

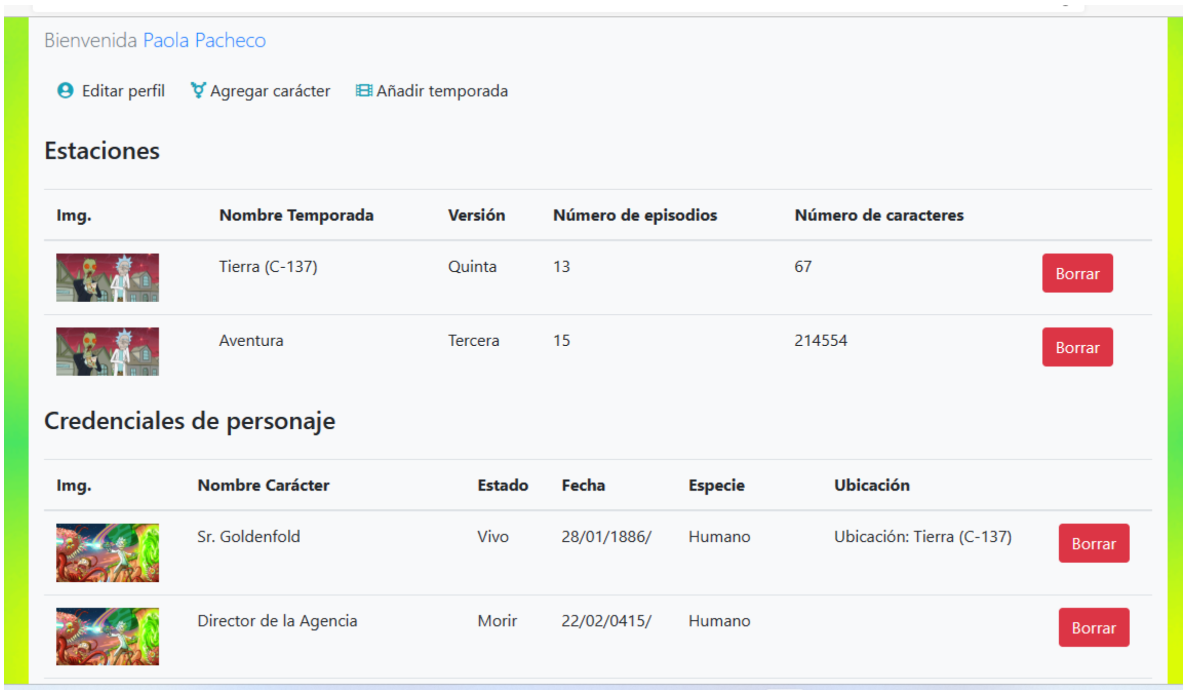
Número de episodios

Número de caracteres

Número de caracteres

Enviar

Api que permite borrar el componente de character y Season


Bienvenida Paola Pacheco

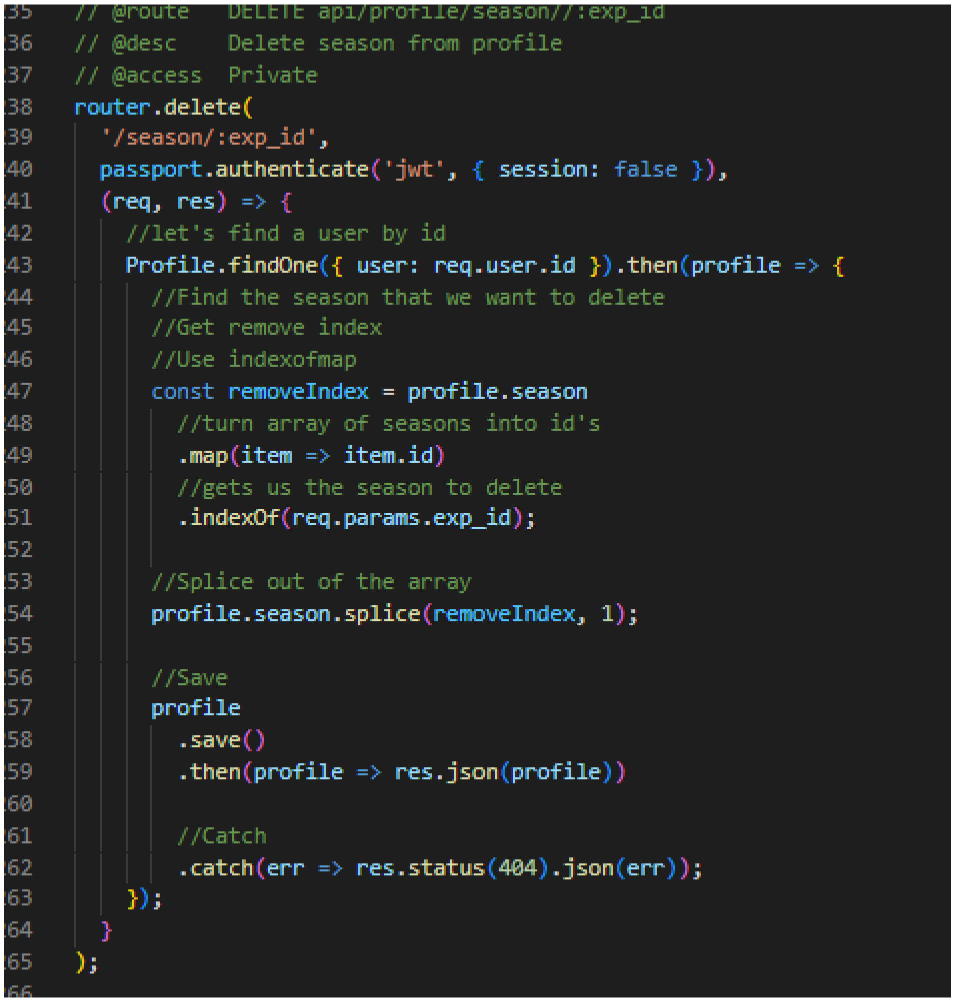
Editar perfil Agregar carácter Añadir temporada

Estaciones

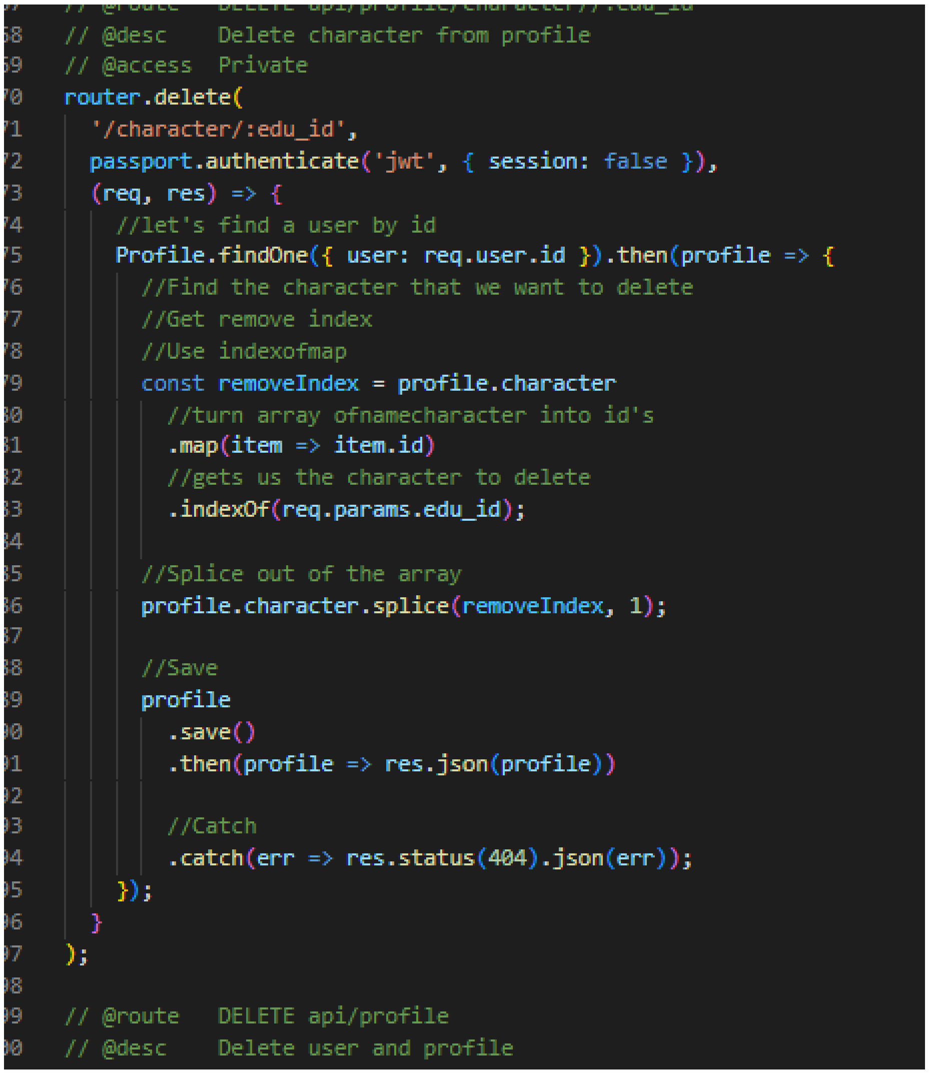
Img.	Nombre Temporada	Versión	Número de episodios	Número de caracteres	
	Tierra (C-137)	Quinta	13	67	Borrar
	Aventura	Tercera	15	214554	Borrar

Credenciales de personaje

Img.	Nombre Carácter	Estado	Fecha	Especie	Ubicación	
	Sr. Goldenfold	Vivo	28/01/1886/	Humano	Ubicación: Tierra (C-137)	Borrar
	Director de la Agencia	Morir	22/02/0415/	Humano		Borrar

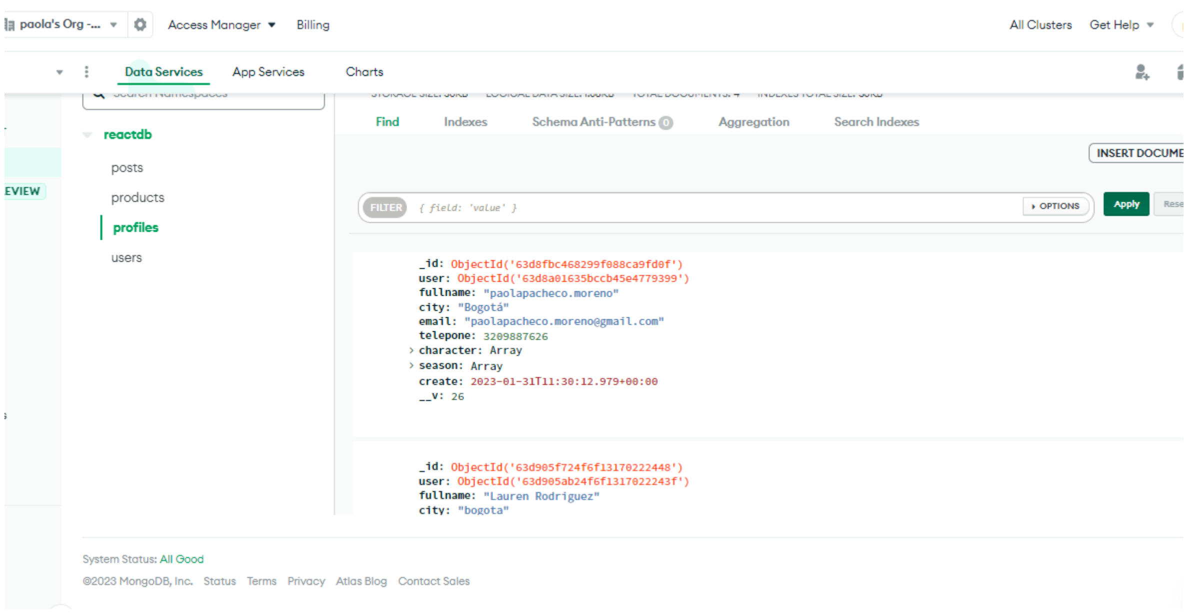


```
// @route   DELETE api/profile/season//:exp_id
// @desc    Delete season from profile
// @access  Private
router.delete(
  '/season/:exp_id',
  passport.authenticate('jwt', { session: false }),
  (req, res) => {
    //let's find a user by id
    Profile.findOne({ user: req.user.id }).then(profile => {
      //Find the season that we want to delete
      //Get remove index
      //Use indexofmap
      const removeIndex = profile.season
        //turn array of seasons into id's
        .map(item => item.id)
        //gets us the season to delete
        .indexOf(req.params.exp_id);

      //Splice out of the array
      profile.season.splice(removeIndex, 1);

      //Save
      profile
        .save()
        .then(profile => res.json(profile))

        //Catch
        .catch(err => res.status(404).json(err));
    });
  }
);
```



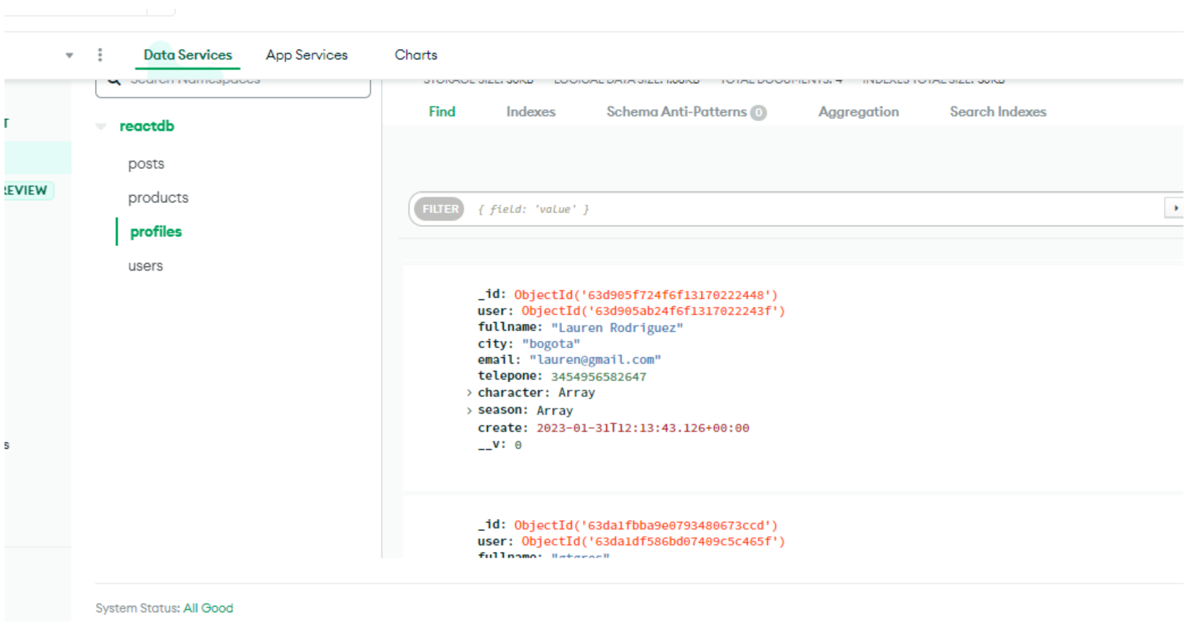
```javascript
// @route   DELETE api/profile/character/:edu_id
// @desc    Delete character from profile
// @access  Private
router.delete(
  '/character/:edu_id',
  passport.authenticate('jwt', { session: false }),
  (req, res) => {
    //let's find a user by id
    Profile.findOne({ user: req.user.id }).then(profile => {
      //Find the character that we want to delete
      //Get remove index
      //Use indexofmap
      const removeIndex = profile.character
        //turn array ofnamecharacter into id's
        .map(item => item.id)
        //gets us the character to delete
        .indexOf(req.params.edu_id);

      //Splice out of the array
      profile.character.splice(removeIndex, 1);

      //Save
      profile
        .save()
        .then(profile => res.json(profile))

        //Catch
        .catch(err => res.status(404).json(err));
    });
  }
);

// @route   DELETE api/profile
// @desc    Delete user and profile
```

Datos almacenados en MongoDB



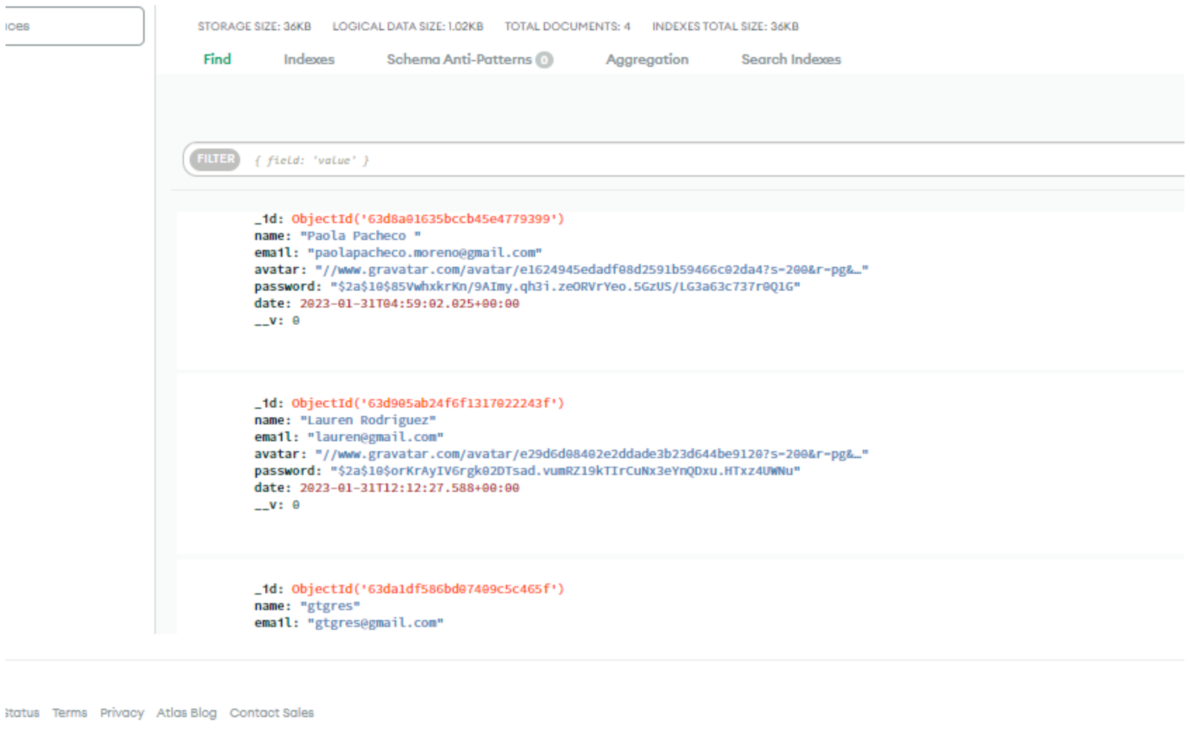
```
{ field: 'value' }
```

```
_id: ObjectId('63d8fbc468299f088ca9fd0f')
user: ObjectId('63d8a01635bccb45e4779399')
fullname: "paolapacheco.moreno"
city: "Bogotá"
email: "paolapacheco.moreno@gmail.com"
telepone: 3209887626
> character: Array
> season: Array
create: 2023-01-31T11:30:12.979+00:00
__v: 26

_id: ObjectId('63d905f724f6f13170222448')
user: ObjectId('63d905ab24f6f1317022243f')
fullname: "Lauren Rodriguez"
city: "bogota"
```



```
{ field: 'value' }
```

```
_id: ObjectId('63d905f724f6f13170222448')
user: ObjectId('63d905ab24f6f1317022243f')
fullname: "Lauren Rodriguez"
city: "bogota"
email: "lauren@gmail.com"
telepone: 3454956582647
> character: Array
> season: Array
create: 2023-01-31T12:13:43.126+00:00
__v: 0

_id: ObjectId('63da1fbba9e0793480673ccd')
user: ObjectId('63da1df586bd07409c5c465f')
```


ices
STORAGE SIZE: 36KB LOGICAL DATA SIZE: 1.02KB TOTAL DOCUMENTS: 4 INDEXES TOTAL SIZE: 36KB
Find
Indexes
Schema Anti-Patterns 0
Aggregation
Search Indexes
FILTER
{ field: 'value' }
Status Terms Privacy Atlas Blog Contact Sales

```
_id: ObjectId('63d8a01635bccb45e4779399')
name: "Paola Pacheco "
email: "paolapacheco.moreno@gmail.com"
avatar: "//www.gravatar.com/avatar/e1624945edadf08d2591b59466c02da4?s=200&r=pg&…"
password: "$2a$10$85VwhxkrKn/9AImy.qh3i.zeORVrYeo.5GzUS/LG3a63c737r0Q1G"
date: 2023-01-31T04:59:02.025+00:00
__v: 0

_id: ObjectId('63d905ab24f6f1317022243f')
name: "Lauren Rodriguez"
email: "lauren@gmail.com"
avatar: "//www.gravatar.com/avatar/e29d6d08402e2ddade3b23d644be9120?s=200&r=pg&…"
password: "$2a$10$orKrAyIV6rgk02DTsad.vumRZ19kTIrCuNx3eYnQDxu.HTxz4UWNu"
date: 2023-01-31T12:12:27.588+00:00
__v: 0

_id: ObjectId('63da1df586bd07409c5c465f')
name: "gtgres"
email: "gtgres@gmail.com"
```

DASBOARD

FRONTED-CARPETA CLIENT

Implementando React -

1. Se ejecuta `create-react-app client para crear la carpeta.`

2. `npm i concurrently`

3. `npm install --prefix client`

4. npm run dev

5. En la carpeta cliente ejecutamos npm i react-router-dom

6. npm i react-bootstrap bootstrap

7. npm i axios classnames jwt-decode react-redux react-router-dom redux redu

x-thunk

8. se crean los respectivos directorios y archivos tales como:

Components, components/auth, entre otros y un de los archivos más principales como el App.js, que hace el llamado de cada uno de los componentes y el router

9. Configuración de Redux y autenticación

Se utiliza para compartir datos entre los diversos componentes,

Para esto instalamos los siguientes paquetes a la carpeta de Client

- npm i redux react-redux redux-thunk

10. Se configura el usuario extrayendo información del token, para esto instalamos jwtcode

- npm i jwt-decode
- import jwt_decode from 'jwt-decode'; en authActions

10. Prepare& Deploy

- Finalmente verificamos que las claves del config sean correctas, para que el sistema no genere problemas en la conexión con el backend

keys_devs.js

//we do not want to push this file

module.exports = {

```
module.exports = {
  mongoURI:
    "mongodb://username:password@ds117545.mlab.com:17545/react-social-network"
};
```

11. Finalmente ejecutamos en la terminal de client, npm start para ver la ejecución del proyecto en el localhost y a su vez también ejecutamos el backend con nodemon server.js desde la raíz del proyecto.