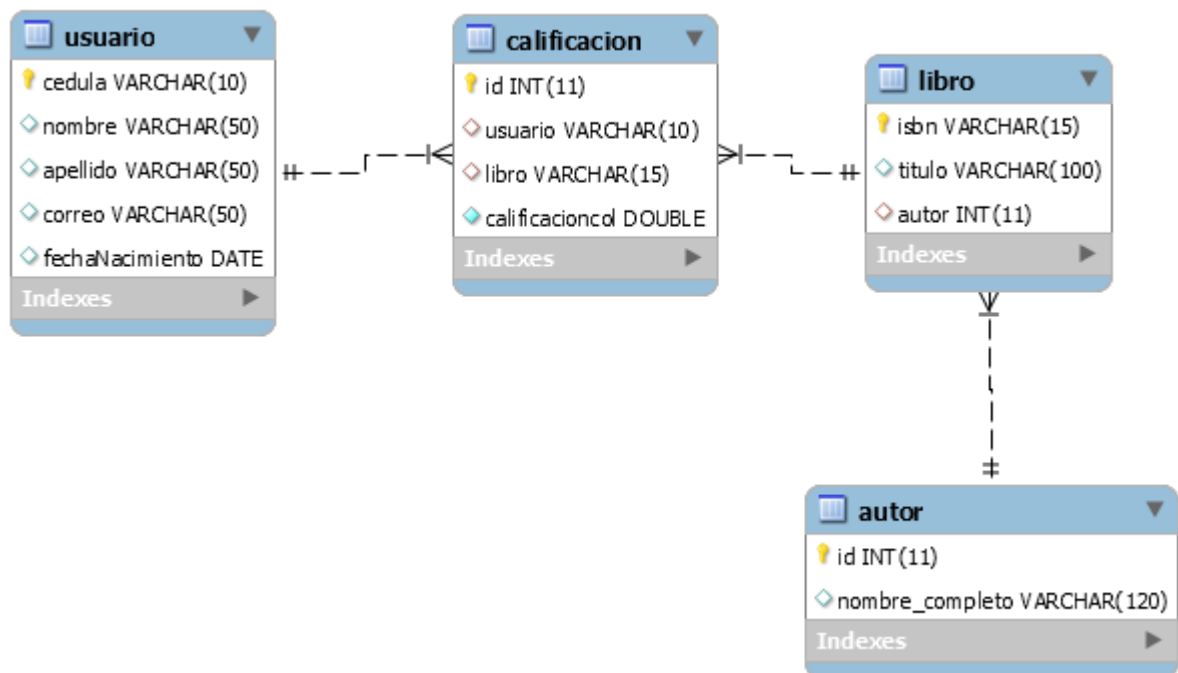


FACULTAD DE INGENIERIA EN ELECTRICIDAD Y COMPUTACION
DESARROLLO DE APLICACIONES WEB
I TÉRMINO 2019
TALLER 5

Grupo:	Master Developers
Integrantes:	Karla Burgos, Carlos Sesme, Tony Veas, Jonathan Quintana
Paralelo:	2

Reporte

1. Modelo lógico



2. Descripción del API REST

Se creó una api básica para poder obtener la data almacenada en la base de datos, las rutas de las cuales consta son:

- 1) "/", ruta principal en la cual se muestra formulario de login en esta se usa el método get.
- 2) "/login", utilizando el método post es la encargada de realizar la autenticación en el sistema y verificar que el usuario sea válido.

- 3) "/logout", utilizando el método post, esta ruta se encarga de cerrar la sesión iniciada previamente.
- 4) "/panel", método get, muestra la página principal en la cual se enlaza ambas bases y renderiza el template de panel de administración de la página.
- 5) "/loadautores", método get, se utiliza para pasar la información de la base de datos no relacional a la relacional.
- 6) "/autores", método get, devuelve todos los autores que están almacenados en la base para poder llenar el combo de la página principal.
- 7) "/libautor/:autor", método get, devuelve los registros de libros del autor que recibe por parámetro en url.
- 8) "/notas/:cedula", método get, obtiene todos los libros calificados por el autor y renderiza el template asignado.
- 9) "/calificar", método post, recibe la nota del usuario y la guarda en la base de datos.

3. Descripción de la Base de Datos no relacional: estructura, motor de la noRBDMS.

Para la base de datos no relacional se usó el motor MongoDB Atlas, la estructura de la colección utilizada está relacionada al archivo histórico.txt provisto para esto se pasó la data de este archivo en un csv y se realizó la importación a MongoDB:

The screenshot shows the MongoDB Atlas web interface. On the left, there's a sidebar with navigation options: 'CONTEXTO' (Project 0), 'ATLAS' (Racimos, Lago de datos BETA, SEGURIDAD, PROYECTO, SERVICIOS), and 'SERVICIOS' (Gráficos, Puntada, Disparadores). The main panel displays the 'taller5_daw.registros_historicos' collection. It shows 'TAMARO DE COLECCIÓN: 1.91MB', 'DOCUMENTOS TOTALES: 13714', and 'TAMARO TOTAL DE LOS INDICES: 132KB'. Below this, there's a search bar with a filter '{\"filter\": \"example\"}' and a table of results. The table shows document details like '_id', 'titulo', 'autores', 'isbn', and 'calificacion_promedio'.

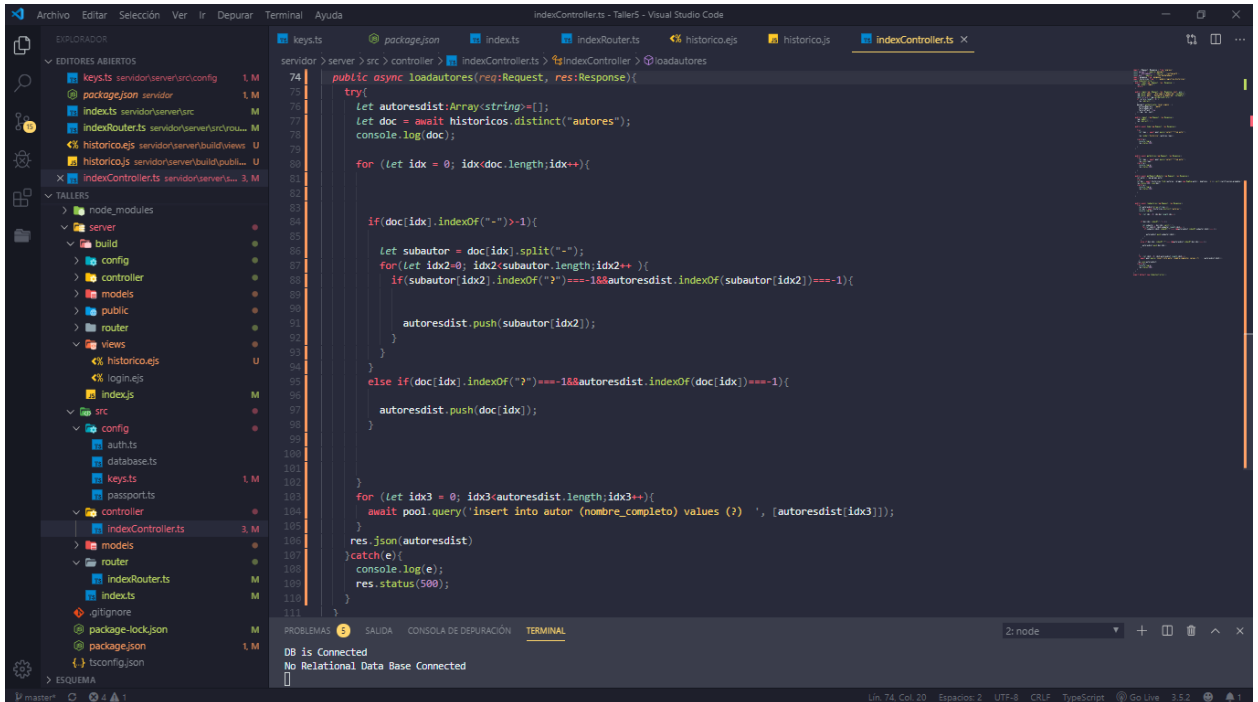
4. Evidencia de la implementación de los requerimientos Req1, Req2 y Req3.

Evidencia del requerimiento 1:

Para cumplir con el primer requerimiento lo primero que se realizó fue la importación de los registros históricos a la base de datos no relacional mostrados anteriormente.

Después se realizó la extracción de los autores de los libros desde la base de datos no relacional y se los registro en la base de datos relacional en su respectiva tabla.

Para esto se creó un servicio dentro del api con el cual se podría realizar este proceso teniendo en cuenta que cada libro podría llegar a tener más de 1 autor por lo cual se debía tener cuidado al realizar los registros en la base de datos relacional para evitar nombres repetidos.

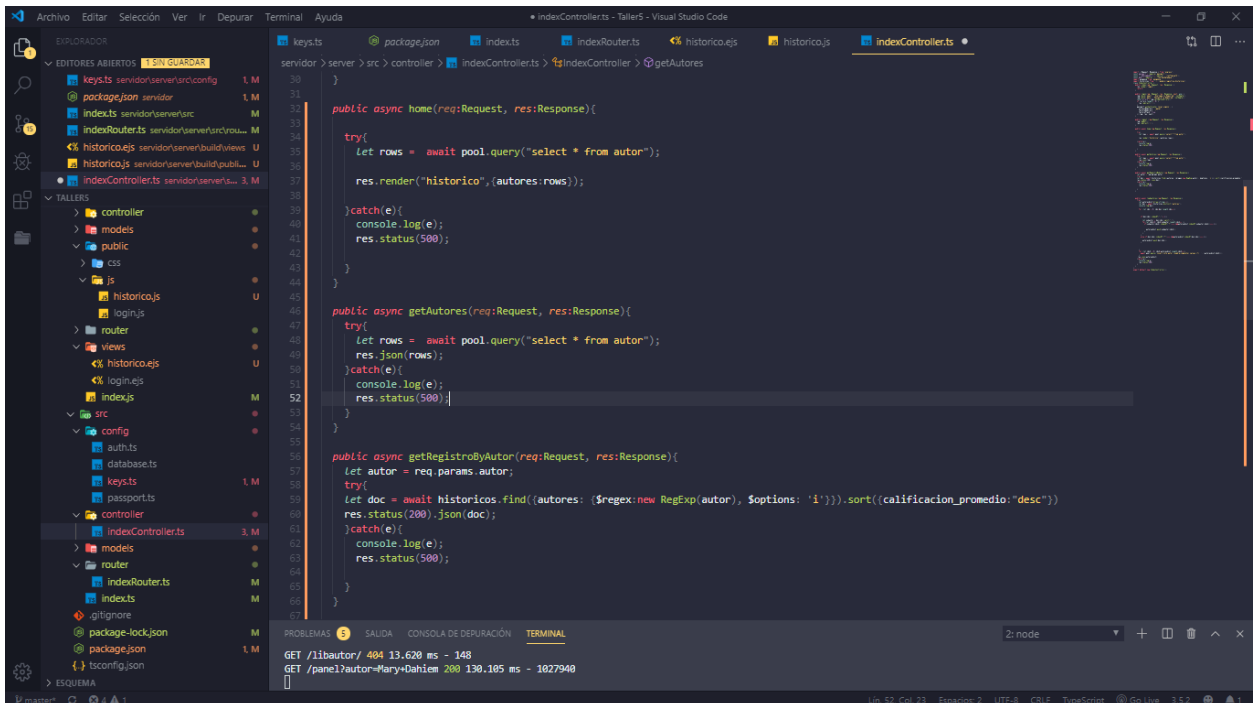


```

74 public async loadAutores(req: Request, res: Response) {
75   try {
76     let autoresdist: Array<string> = [];
77     let doc = await historicos.distinct("autores");
78     console.log(doc);
79
80     for (let idx = 0; idx < doc.length; idx++) {
81
82       if (doc[idx].indexOf("-") > -1) {
83
84         let subautor = doc[idx].split("-");
85         for (let idx2 = 0; idx2 < subautor.length; idx2++) {
86           if (subautor[idx2].indexOf("?") === -1 && autoresdist.indexOf(subautor[idx2]) === -1) {
87
88             autoresdist.push(subautor[idx2]);
89           }
90         }
91       }
92       else if (doc[idx].indexOf("?") === -1 && autoresdist.indexOf(doc[idx]) === -1) {
93         autoresdist.push(doc[idx]);
94       }
95     }
96
97     for (let idx3 = 0; idx3 < autoresdist.length; idx3++) {
98       await pool.query('insert into autor (nombre_completo) values (?) ', [autoresdist[idx3]]);
99     }
100     res.json(autoresdist);
101   } catch (e) {
102     console.log(e);
103     res.status(500);
104   }
105 }
106
107 }
108
109 }
110
111 }

```

Una vez cargados los nombres de los autores a la base de datos relacional se planteó los servicios con los cuales el usuario podría seleccionar el autor y obtener todos los libros relacionales a el ordenados de acuerdo a su calificación:

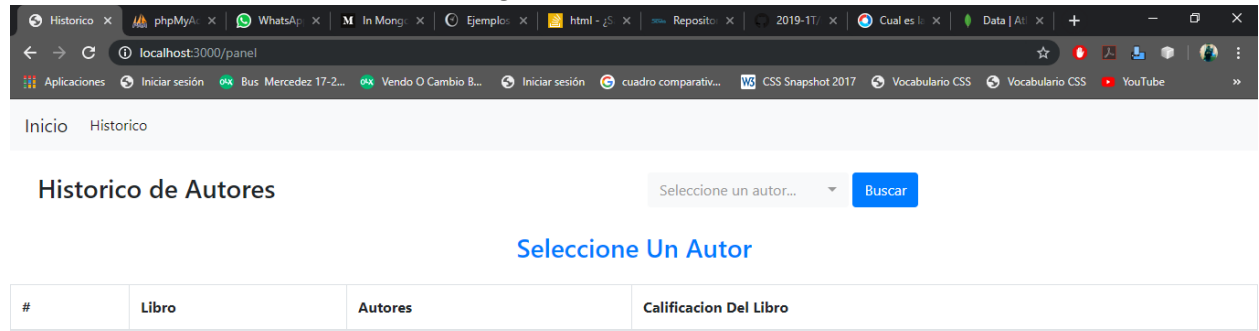


```

30 }
31
32 public async home(req: Request, res: Response) {
33   try {
34     let rows = await pool.query("select * from autor");
35     res.render("historico", {autores: rows});
36   } catch (e) {
37     console.log(e);
38     res.status(500);
39   }
40 }
41
42 public async getAutores(req: Request, res: Response) {
43   try {
44     let rows = await pool.query("select * from autor");
45     res.json(rows);
46   } catch (e) {
47     console.log(e);
48     res.status(500);
49   }
50 }
51
52 public async getRegistroByAutor(req: Request, res: Response) {
53   let autor = req.params.autor;
54   let doc = await historicos.find({autores: {$regex: new RegExp(autor)}, $options: 'i'}).sort({calificacion_promedio: "desc"});
55   res.status(200).json(doc);
56 } catch (e) {
57   console.log(e);
58   res.status(500);
59 }
60 }
61
62 }
63
64 }
65
66 }
67

```

Como resultado visual se mostrar de la siguiente manera:



Inicio Historico

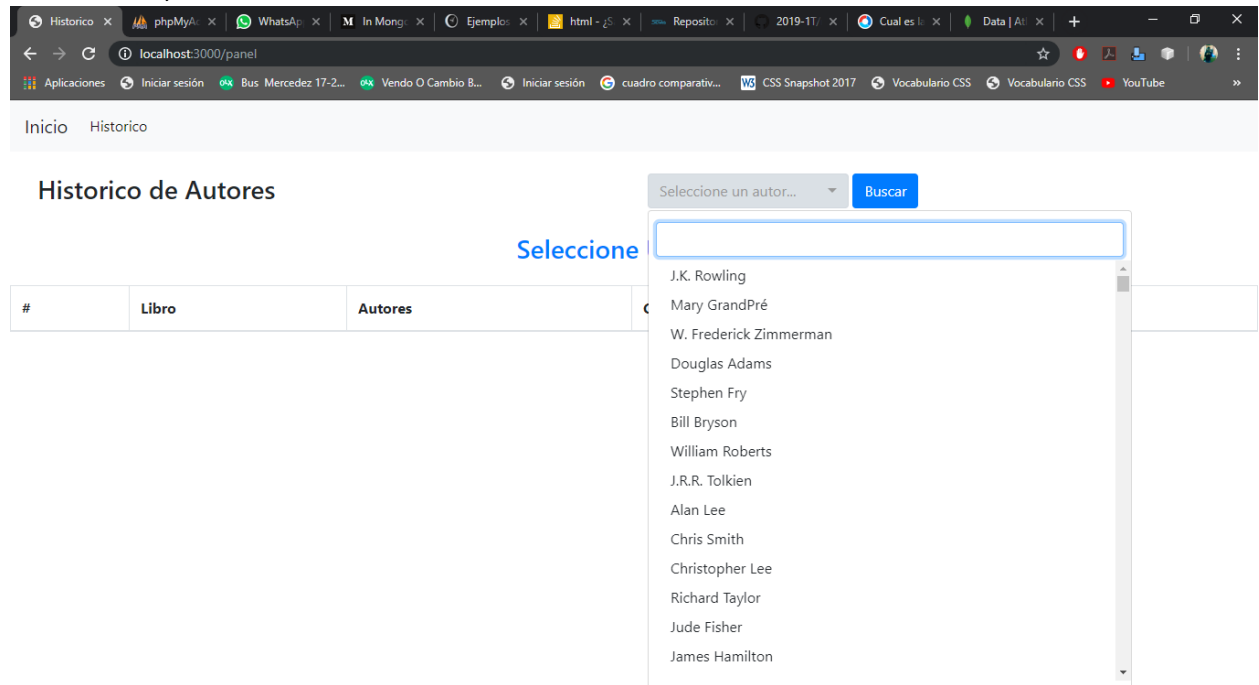
Historico de Autores

Seleccione un autor... Buscar

[Seleccione Un Autor](#)

#	Libro	Autores	Calificacion Del Libro
---	-------	---------	------------------------

Los usuarios podrán seleccionar o buscar desde un search select box el nombre del autor y al realizar la búsqueda se mostrarán todos los libros de dicho autor contenidos en el historial (base no relacional) ordenados de acuerdo a su calificación en dicho historial:



Inicio Historico

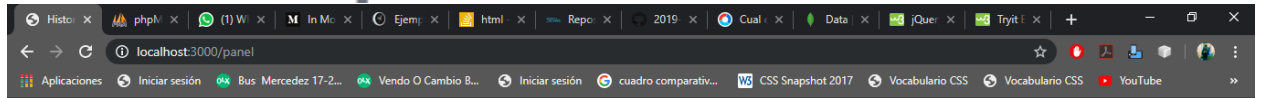
Historico de Autores

Seleccione un autor... Buscar

[Seleccione](#)

#	Libro	Autores
---	-------	---------

- J.K. Rowling
- Mary GrandPré
- W. Frederick Zimmerman
- Douglas Adams
- Stephen Fry
- Bill Bryson
- William Roberts
- J.R.R. Tolkien
- Alan Lee
- Chris Smith
- Christopher Lee
- Richard Taylor
- Jude Fisher
- James Hamilton



Inicio Historico

Historico de Autores

Seleccione un autor...

do

Buscar

do

Douglas Adams

Donna Ickes

Kate Douglas

Mark D. Widome

Don DeLillo

Carol Dommermuth

Steve Bodow

Donald A. Norman

Donald J. Trump

Donald T. Phillips

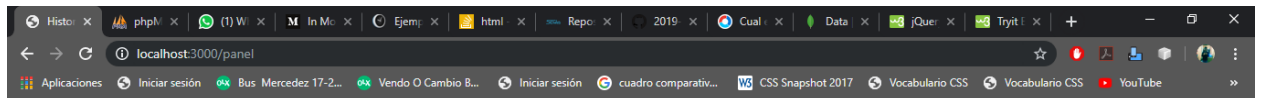
Rudolph W. Giuliani

Donada Peters

Doug Walsh

Herodotus

#	Libro	Autores
---	-------	---------



Inicio Historico

Historico de Autores

Douglas Adams

Buscar

Los Libros De: Douglas Adams

#	Libro	Autores	Calificacion Del Libro
1	The More Than Complete Hitchhiker's Guide (Hitchhiker's Guide #1-4 + short story)	Douglas Adams	458
2	The Ultimate Hitchhiker's Guide: Five Complete Novels and One Story (Hitchhiker's Guide to the Galaxy #1-5)	Douglas Adams	438
3	The Ultimate Hitchhiker's Guide to the Galaxy	Douglas Adams	438
4	The Ultimate Hitchhiker's Guide (Hitchhiker's Guide to the Galaxy #1-5)	Douglas Adams	438
5	The Hitchhiker's Guide to the Galaxy: The Quintessential Phase (Hitchhiker's Guide: Radio Play #5)	Douglas Adams	437
6	The Illustrated Hitchhiker's Guide To The Galaxy	Douglas Adams	432
7	The Hitchhiker's Guide to the Galaxy (Hitchhiker's Guide to the Galaxy #1)	Douglas Adams	422
8	The Hitchhiker's Guide to the Galaxy (Hitchhiker's Guide to the Galaxy #1)	Douglas Adams, Stephen Fry	422
9	The Restaurant at the End of the Universe (The Hitchhiker's Guide to the Galaxy #2)	Douglas Adams, Martin Freeman	422

```

server > server > src > router > indexRouter.ts > @config
1 import { Router } from 'express';
2 import indexController from '../controller/indexController';
3
4 class IndexRoutes {
5   public router: Router = Router();
6
7   constructor() {
8     this.config();
9   }
10
11   config():void {
12     this.router.get('/', indexController.index);
13     this.router.post('/login', indexController.login);
14     this.router.post('/logout', indexController.logout);
15     this.router.get('/panel', indexController.home);
16     this.router.get('/loadautores', indexController.loadautores);
17     this.router.get('/autores', indexController.getAutores);
18   }
19 }
20
21 export default IndexRoutes;

```

```

GET /panel 304 153.719 ms --
GET /libautor/J.K.X20Rowling 304 270.234 ms --
GET /libautor/J.K.X20Rowling 304 153.363 ms --
GET /libautor/J.K.X20Rowling 304 135.035 ms --
[nodemon] restarting due to changes...
[nodemon] starting 'node server/build/index.js'
server on port: 3000
DB is Connected
No Relational Data Base Connected
GET /panel 304 161.132 ms --
GET /libautor/J.R.R.X20Tolkien 200 321.525 ms - 13817
[nodemon] restarting due to changes...
[nodemon] starting 'node server/build/index.js'
server on port: 3000
DB is Connected
No Relational Data Base Connected
GET /panel 304 178.873 ms --
GET /libautor/BillX20Bryson 304 231.187 ms --
GET /libautor/MaryX20GrandPrX20MAG 304 130.838 ms --
GET /libautor/J.K.X20Rowling 304 151.653 ms --
GET /libautor/DouglasX20Adams 304 110.406 ms --
GET /libautor/DouglasX20Adams 304 121.466 ms --
GET /panel 304 195.649 ms --
GET /libautor/DouglasX20Adams 304 133.540 ms --

```

Requerimiento 2

Para proceder con este requerimiento se implementó una opción más en la pantalla principal, en este caso un botón que enlace hacia la sección de calificaciones.

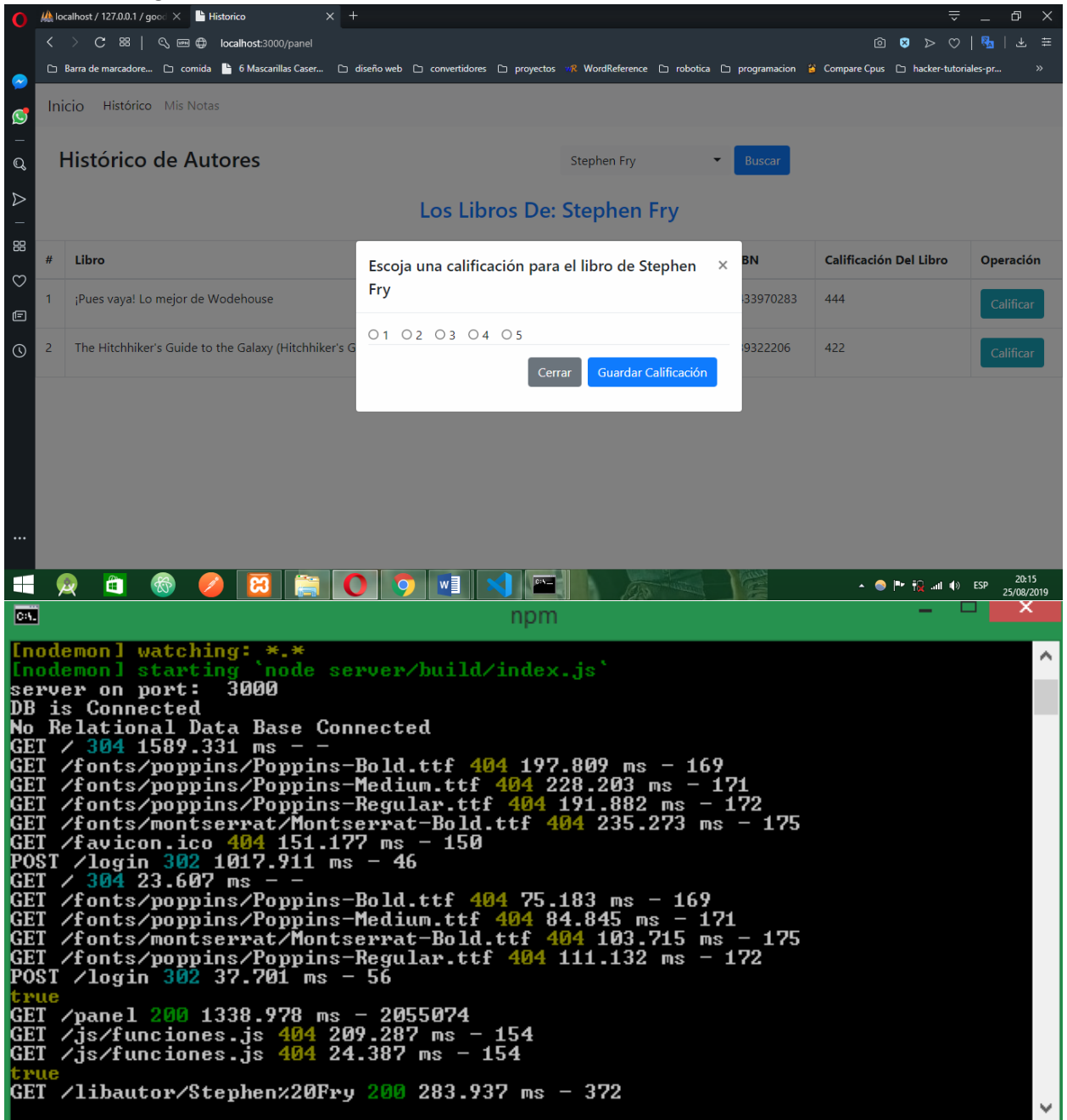
Inicio Histórico Mis Notas

Histórico de Autores

Los Libros De: Stephen Fry

#	Libro	Autores	ISBN	Calificación Del Libro	Operación
1	¡Pues vaya! Lo mejor de Wodehouse	P.G. Wodehouse, Stephen Fry	8433970283	444	<input type="button" value="Calificar"/>
2	The Hitchhiker's Guide to the Galaxy (Hitchhiker's Guide to the Galaxy #1)	Douglas Adams, Stephen Fry	739322206	422	<input type="button" value="Calificar"/>

Y al presionar el botón de calificar se presenta un modal en el cual se procederá a calificar el libro en un rango el 1 al 5



The screenshot shows a web application interface for rating books. A modal is open for rating a book by Stephen Fry. The modal contains a title, a rating scale from 1 to 5, and buttons for 'Cerrar' (Close) and 'Guardar Calificación' (Save Rating). The background shows a table of books and a terminal window with API logs.

#	Libro	BN	Calificación Del Libro	Operación
1	¡Pues vaya! Lo mejor de Wodehouse	33970283	444	<button>Calificar</button>
2	The Hitchhiker's Guide to the Galaxy (Hitchhiker's G	9322206	422	<button>Calificar</button>

```

[nodemon] watching: *.*
[nodemon] starting 'node server/build/index.js'
server on port: 3000
DB is Connected
No Relational Data Base Connected
GET / 304 1589.331 ms - -
GET /fonts/poppins/Poppins-Bold.ttf 404 197.809 ms - 169
GET /fonts/poppins/Poppins-Medium.ttf 404 228.203 ms - 171
GET /fonts/poppins/Poppins-Regular.ttf 404 191.882 ms - 172
GET /fonts/montserrat/Montserrat-Bold.ttf 404 235.273 ms - 175
GET /favicon.ico 404 151.177 ms - 150
POST /login 302 1017.911 ms - 46
GET / 304 23.607 ms - -
GET /fonts/poppins/Poppins-Bold.ttf 404 75.183 ms - 169
GET /fonts/poppins/Poppins-Medium.ttf 404 84.845 ms - 171
GET /fonts/montserrat/Montserrat-Bold.ttf 404 103.715 ms - 175
GET /fonts/poppins/Poppins-Regular.ttf 404 111.132 ms - 172
POST /login 302 37.701 ms - 56
true
GET /panel 200 1338.978 ms - 2055074
GET /js/funciones.js 404 209.287 ms - 154
GET /js/funciones.js 404 24.387 ms - 154
true
GET /libautor/Stephen%20Fry 200 283.937 ms - 372
  
```

Código de la calificación

```

42     res.status(500);
43   }
44 }
45
46
47
48 public async ingresarCalificacion(req:Request, res:Response){
49   const {nota,libro,usuario} = req.body;
50   try{
51     let rows = await pool.query("INSERT INTO calificacion (libro, usuario, calificacion) values(?,?,?)",[libro,usuario,nota]);
52   }catch(e){
53     console.log(e);
54     res.status(500);
55   }
56   //console.log(req.body);
57   res.status(200).json(req.body);
58 }
59
60
61
62 public async getAutores(req:Request, res:Response){
63   try{
64     let rows = await pool.query("select * from autor");
65     res.json(rows);
66   }catch(e){
67     console.log(e);
68     res.status(500);
69   }
70 }
71
72 public async getRegistroByAutor(req:Request, res:Response){
73   let autor = req.params.autor;
74   try{

```

Requerimiento 3

Para este se implementó una interfaz minimalista en la cual solo se mostrará los datos en una tabla con los detalles de todos los libros que el usuario haya calificado.

ISBN	User	Libro
0439358078	0910088796	Harry Potter and the Order of the Phoenix (Harry Potter #5)
0439358078	0910088796	Harry Potter and the Order of the Phoenix (Harry Potter #5)
0439554934	0910088796	Harry Potter and the Sorcerer's Stone (Harry Potter #1)
0439785960	0910088796	Harry Potter and the Half-Blood Prince (Harry Potter #6)
0439554934	0910088796	Harry Potter and the Sorcerer's Stone (Harry Potter #1)
0439785960	0910088796	Harry Potter and the Half-Blood Prince (Harry Potter #6)

Esta sección necesita recibir por la url la cedula del usuario que se desea consultar, es algo poco seguro, pero se lo implemento debido a que son fines académicos y para tratar de hacerlo de una manera más sencilla esto se puede solucionar mediante una cookie de sesión extra ya que

nuestro sistema de login se basa en un middleware de express el cual almacena las sesiones en la base de datos relacional.

Funcionamiento del login.

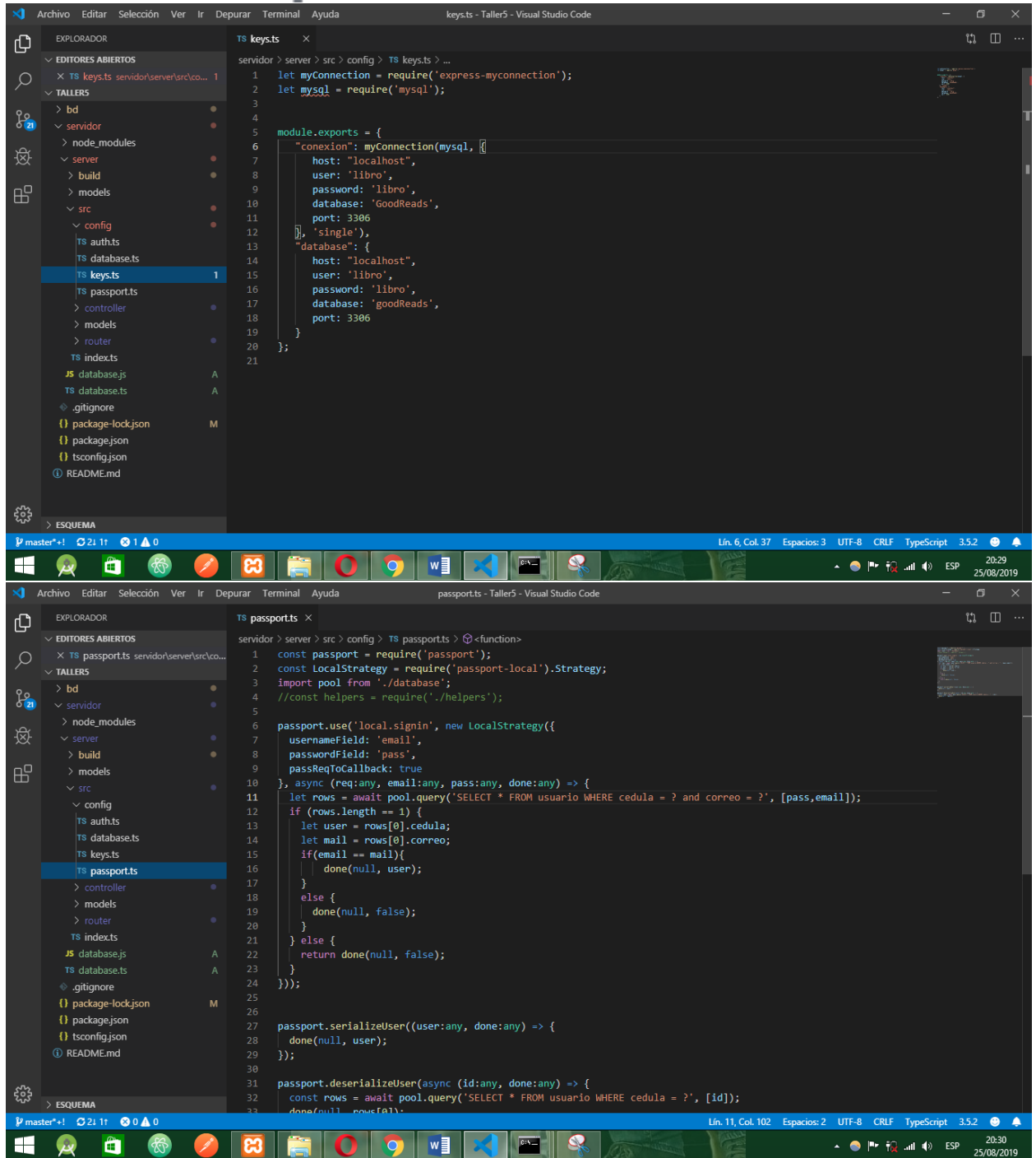
The image displays two screenshots of the Visual Studio Code editor, showing the implementation of a login system using Express.js and a relational database.

Top Screenshot: `auth.ts`

```
1 module.exports = {
2   isLoggedIn(req:any, res:any, next:any) {
3     console.log(req.isAuthenticated());
4     if (req.isAuthenticated()) {
5       return next();
6     }
7     return res.redirect('/');
8   },
9   isLoggedIn(req:any, res:any, next:any) {
10    if (!req.isAuthenticated()) {
11      return next();
12    }
13    return res.redirect('/');
14  }
15 };
16
```

Bottom Screenshot: `database.ts`

```
1 let mysql = require('mysql');
2 const { promisify } = require('util');
3
4 const { database } = require('./keys');
5
6 let pool = mysql.createPool(database);
7
8 pool.getConnection((err:any, connection:any) => {
9   if (err) {
10     if (err.code === 'PROTOCOL_CONNECTION_LOST') {
11       console.error('Database connection was closed.');
```



keys.ts

```

1 let myConnection = require('express-myconnection');
2 let mysql = require('mysql');
3
4
5 module.exports = {
6   "conexion": myConnection(mysql, {
7     host: 'localhost',
8     user: 'libro',
9     password: 'libro',
10    database: 'GoodReads',
11    port: 3306
12  }, 'single'),
13   "database": {
14     host: 'localhost',
15     user: 'libro',
16     password: 'libro',
17     database: 'goodReads',
18     port: 3306
19   }
20 };
21

```

passport.ts

```

1 const passport = require('passport');
2 const LocalStrategy = require('passport-local').Strategy;
3 import pool from './database';
4 //const helpers = require('./helpers');
5
6 passport.use('local.signin', new LocalStrategy({
7   usernameField: 'email',
8   passwordField: 'pass',
9   passReqToCallback: true
10 }, async (req:any, email:any, pass:any, done:any) => {
11   let rows = await pool.query('SELECT * FROM usuario WHERE cedula = ? and correo = ?', [pass,email]);
12   if (rows.length == 1) {
13     let user = rows[0].cedula;
14     let mail = rows[0].correo;
15     if(email == mail){
16       done(null, user);
17     }
18     else {
19       done(null, false);
20     }
21   } else {
22     return done(null, false);
23   }
24 }));
25
26 passport.serializeUser((user:any, done:any) => {
27   done(null, user);
28 });
29
30 passport.deserializeUser(async (id:any, done:any) => {
31   const rows = await pool.query('SELECT * FROM usuario WHERE cedula = ?', [id]);
32   done(null, rows[0]);
33 }

```

url del repositorio: <https://github.com/kbburgos/Taller5>