UNIVERSITAT POLITÈCNICA DE VALÈNCIA

TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

Desarrollo de una app multiplataforma para la generación y almacenamiento de contraseñas seguras

Vidal Domínguez, Mario

Tutor: López Patiño, José Enrique

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingeniería de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2020-21

Valencia, 11 de Junio de 2021

Resumen

El proyecto consiste en el desarrollo de una app multiplataforma para la generación y almacenamiento de contraseñas seguras. En dicha app, el usuario será capaz de generar una contraseña después de registrarse y confirmar el correo de registro. La contraseña será generada a partir de la inserción, por parte del usuario, de 4 campos: 'nombre', 'longitud', 'dificultad', 'pista' y su respuesta. Tras la generación, un hash de la contraseña será almacenado en la base de datos junto a otros valores necesarios para poderla regenerar. El usuario para poder volver a generar la contraseña necesitará introducir las respuestas a las pistas que ha introducido previamente.

La aplicación se ha desarrollado sobre una plataforma Web, en el que el software del cliente está desarrollado en HTML5, JavaScript, CSS y el servidor en Node.js, utilizando el sistema de gestión de paquetes de npm para instalar distintas librerías necesarias para el desarrollo. La base de datos será proporcionada por MongoDB y será de tecnología NoSQL.

Resum

El projècte consisteix en el desenvolupament d'una app multiplataforma para la generació i emmagatzematge de contrasenyes segures. En aquesta app, l'usuari serà capaç de generar una contrasenya després de registrar-se i confirmar el correu de registre. La contrasenya serà generada a partir de la inserció, per part de l'usuari, de 4 camps: 'nom', 'longitud', 'dificultat', 'pista' i la seua resposta. En acabada la generació, un hash de la contrasenya serà emmagatzenat en la base de dades al costat d'altres valors necessaris per a poder-la regenerar. L'usuari per a poder tornar a generar la contrasenya necessitarà introduir les respostes a les pistes que ha introduït prèviament.

L'aplicació s'ha desenvolupat sobre una plataforma Web, en el qual el software del client està desenvolupat en HTML5, JavaScript, CSS i el servidor de Node.js, utilitzant el sistema de gestió de paquets de npm per a instal·lar diferents llibreries necessàries per al desenvolupament. La base de dades serà proporcionada per MongoDB i serà de tecnología NoSQL.

Abstract

The project involves the development of a multiplatform app for the generation and storage of secure passwords. In this app, the user will be able to generate a password after registering and confirming the registration email. The password will be generated from the insertion, by the user, of 4 fields: 'name', 'length', 'difficulty', 'hint' and the answer. After generation, a hash of the password will be stored in the database along with other values necessary to be able to regenerate it. To be able to generate the password again, the user will need to enter the answers to the clues that they have previously entered.

The application has been developed on a Web platform, in which the client software is developed in HTML5, JavaScript, CSS and the server in Node.js, using the npm package management system to install different libraries necessary for development. The database will be provided by MongoDB and will be NoSQL technology.

Índice general

Índice de figuras

Índice de tablas

Listado de siglas empleadas

API Application Programming Interface.

BDD Behavior Driven Development.

BSON Binary javaScript Object Notation.

CSRF Cross-Site Request Forgery.

CSS Cascading Style Sheets.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

JSON JavaScript Object Notation.

MEAN MongoDB, ExpressJS, AngularJS, NodeJS.

NoSQL No Structured Query Language.

npm Node Package Manager.

REST Representational State Transfer.

SQL Structured Query Language.

TDD Test Driven Development.

TFG Trabajo Fin de Grado.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

VSCode Visual Studio Code.

XML eXtensible Markup Language.

Parte I

Memoria

Capítulo 1

Introducción y objetivos

1.1. Introducción

Hoy en día las contraseñas forman parte de nuestra vida diaria. Para poder utilizar una red social es necesario iniciar sesión previamente, para poder hacer una compra online con Paypal es necesario iniciar sesión, incluso para poder firmar un documento es necesario que introduzcas una contraseña. A pesar de ser algo tan común, es cierto que muchas personas siguen teniendo contraseñas poco seguras y fáciles de hackear. Esta poca seguridad hace que muchos de sus datos queden al descubierto ante programas maliciosos que lo único que buscan es conseguir información bancaria o personal para poderla vender más adelante.

Actualmente, las contraseñas de 8 caracteres no son seguras, muy a pesar de que contengan símbolos, números y letras mayúsculas o minúsculas. Sitios web de gran importancia, como Google o Amazon, recomiendan en sus guías de como generar una contraseña de que la longitud mínima de la contraseña sea de 12 caracteres. Para que la contraseña sea considerada segura, además de poseer una longitud mínima de 12 caracteres, debe ir acompañada de una sucesión de símbolos, números y letras mayúsculas y minúsculas.

1.2. Objetivos

El principal objetivo de este TFG es el desarrollo de una aplicación web que permita la generación y almacenamiento de contraseñas, así como de aprender lenguajes de programación nuevos como JavaScript y tecnologías como Node.js. A su vez, este proyecto me permitirá incrementar mis conocimientos sobre el desarrollo de páginas web, dándome una visión sencilla pero amplia tanto del Back End como del Front End.

1.3. Metodología

Para la realización del TFG se utilizó una metodología de trabajo en la que las tareas se separaron en intervalos de tiempo. Cada tarea tenía un objetivo que era necesario para poder realizar la siguiente tarea.

1.3.1. Distribución de las tareas realizadas

A continuación, se mostrarán las tareas realizadas a lo largo de la realización del TFG.

- **Tarea 1:** Captura de requerimientos.
- **Tarea 2:** Aprendizaje del entorno de trabajo.
- **Tarea 3:** Aprendizaje de los lenguajes de programación; JavaScript, HTML5 (Pug), CSS.
- **Tarea 4:** Aprendizaje de las funcionalidades de Express.js.
- **Tarea 5:** Aprendizaje de las bases de datos NoSQL.
- **Tarea 6:** Diseño de la base de datos.
- **Tarea 7:** Diseño y creación de las interfaces de la aplicación.
- **Tarea 8:** Implementación del registro de usuario y sus distintas funcionalidades.
- **Tarea 9:** Implementación de los menús principales e interfaces de error.
- **Tarea 10:** Programación del algoritmo del generado de contraseñas y su almacenamiento.
- **Tarea 11:** Fase de pruebas y corrección de errores, bugs y arreglos visuales.
- **Tarea 12:** Despliegue de la aplicación.

Tareas	Duración	Fecha de Inicio	Fecha de Fin
Tarea 1	3 Días	30/03/2021	01/04/2021
Tarea 2	3 Días	02/04/2021	04/04/2021
Tarea 3	7 Días	05/04/2021	11/04/2021
Tarea 4	3 Días	12/04/2021	14/04/2021
Tarea 5	2 Días	15/04/2021	16/04/2021
Tarea 6	2 Días	17/04/2021	18/04/2021
Tarea 7	7 Días	19/04/2021	25/04/2021
Tarea 8	4 Días	26/04/2021	29/04/2021
Tarea 9	3 Días	30/04/2021	02/05/2021
Tarea 10	7 Días	03/05/2021	09/05/2021
Tarea 11	10 Días	10/05/2021	19/05/2021
Tarea 12	1 Día	20/05/2021	20/05/2021

Tabla 1.1: Organización temporal de las tareas.

1.4. Estructura de la memoria

La memoria se divide en capítulos que a su vez pueden dividirse en subcapítulos. A continuación, se mostrará una breve explicación del contenido de cada capítulo.

- **Capítulo 1:** En este capítulo se ha presentado el proyecto con una breve introducción, sus objetivos y la metodología llevada a cabo.
- **Capítulo 2:** El objetivo de este capítulo es el de explicar las tecnologías y herramientas utilizadas a lo largo del desarrollo de la aplicación.
- **Capítulo 3:** En este capítulo se mostrará y explicará el análisis previo realizado antes de desarrollar la aplicación.
- **Capítulo 4:** Este capítulo contiene todas las fases que se llevaron a cabo a la hora de desarrollar la aplicación.
- **Capítulo 5:** En este capítulo se explicarán las pruebas llevadas a cabo y como se ha realizado el despliegue de la aplicación.
- **Capítulo 6:** Este capítulo contiene una breve explicación de la seguridad implementada en la aplicación frente a ataques.
- **Capítulo 7:** En este último capítulo se extraen una lista de conclusiones y las posibles adiciones futuras a la aplicación.

Capítulo 2

Tecnologías y Herramientas

2.1. Tecnologías

2.1.1. Node.js

Node.js es un entorno de ejecución de JavaScript orientado a eventos asíncronos y diseñado para crear aplicaciones network escalables. El uso más popular para este entorno es el de desarrollo web, sin embargo, no es el único que tiene, ya que se pueden crear scripts, herramientas y muchas más funcionalidades con este entorno. [1]

A la hora de realizar este TFG se utilizó el framework de Express.js [2]. Podemos dividir el trabajo realizado por Node.js en 3 partes:

- **Ejecutar el servidor:** Node.js se encarga de crear un servidor y mantenerlo activo a las peticiones que le van entrando. En el caso de este proyecto, las peticiones serán HTTP GET y POST, aunque podrían utilizarse PUT, DELETE, PATCH...

  La figura 2.1 muestra una captura con la iniciación del servidor en el proyecto.

```
135  // SERVER & DATABASE CONNECTION
136  mongoose.connect(
137      MONGODB_URI,
138      {
139          useNewUrlParser: true,
140          useUnifiedTopology: true
141      }
142  )
143  .then((result) => {
144      app.listen(process.env.PORT || 3000);
145  })
146  .catch((err) => {
147      console.log(err);
148  })
```

Figura 2.1: Iniciación del servidor y la base de datos, 'app.js'.

- **Ejecutar lógica de la aplicación:** En el entorno de desarrollo será necesario manejar solicitudes, validar los inputs y conectarse a la base de datos.

Para poder manejar las solicitudes se utilizó el framework de Express.js, que lo que hace es direccionar las solicitudes según la URI y el método HTTP con el que se hace la petición al servidor, o también llamados EndPoints. [3]

La figura 2.2 muestra un ejemplo de EndPoint.

```
1  // NODE VANILLA PACKAGES DECLARATIONS
2  const express = require('express');
3
4  // NPM PACKAGES DECLARATIONS
5
6  // CONTROLLERS, MODELS, MIDDLEWARES DECLARATIONS
7  const menusController = require('../controllers/menus');
8  const isAuth = require('../middleware/is-auth');
9  const isNotAuth = require('../middleware/is-not-auth');
10
11 // INITIALIZATION
12 const router = express.Router();
13
14 // ROUTES ..
15 router.get('/', isNotAuth, menusController.getIndex);
16 router.get('/menu', isAuth, menusController.getMenu);
17 router.get('/guide', isAuth, menusController.getGuide);
18
19 module.exports = router;
```

Figura 2.2: Ejemplo de EndPoints, 'menus.js'.

En el caso de validación de los inputs, se utilizó una dependencia llamada 'express-validator'. Dicha dependencia permite validar los datos enviados por parte del cliente antes de ser tratados. [4]

En este proyecto, 'express-validator' ha sido muy útil, ya que a la hora de hacer el login, registro, generar la contraseña o modificar datos sobre tu cuenta, era necesario tener un control de la información de entrada. La figura 2.3 muestra un ejemplo de uso de dicha dependencia.

```
17 // ROUTES ..
18 router.get('/login', isNotAuth, authController.getLogin);
19 router.post(
20     '/login',
21     [
22         body('email')
23             .isEmail()
24             .withMessage((value, {req}) => {
25                 return req.t('loginView.validationErrors.email');
26             })
27             .toLowerCase()
28             .trim(),
29         body('password', (value, {req}) => {
30                 return req.t('loginView.validationErrors.pass');
31             })
32             .matches(regex)
33             .trim()
34     ],
35     isNotAuth,
36     authController.postLogin
37 );
```

Figura 2.3: Ejemplo de validación, 'auth.js'.

Con respecto a la conexión con la base de datos, se puede hacer de diversas maneras dependiendo del tipo que se utilice, SQL o NoSQL.

En este proyecto se ha utilizado una base de datos NoSQL con la ayuda de MongoDB Atlas, el cual nos permite crear un cluster en la nube de manera gratuita. En dicho cluster podremos crear múltiples bases de datos y manejarlas con las limitaciones de la versión gratuita [5].

Para realizar la conexión de dicha base de datos con nuestro servidor se utilizó una dependencia llamada 'mongoose'. Dicha dependencia nos permite conectarnos de manera rápida y sencilla a nuestra base de datos en la nube [6]. La conexión a la base de datos se muestra en la figura 2.1.

- **Responder:** Con Node.js se puede responder a una petición de múltiples maneras, por ejemplo, enviando información en formato JSON, enviando un fichero HTML renderizado, redireccionando... En esta aplicación la manera principal de respuesta será la de un fichero HTML renderizado o redireccionando.

  La figura 2.4 muestra un ejemplo de respuesta de un fichero HTML renderizado.

```
13  // EXPORTS
14  exports.getIndex = async (req, res, next) => {
15      const error = errorMessage(req);
16      const message = infoMessage(req);
17      return res
18          .status(200)
19          .render('menus/index', {
20              path: '/',
21              pageTitle: req.t('pageTitles.menusTitles.index'),
22              navNames: req.t('nav'),
23              indexNames: req.t('indexView'),
24              errorMessage: error,
25              validationErrors: [],
26              message: message
27          });
28  };
```

Figura 2.4: Ejemplo de respuesta, 'menus.js'.

2.1.2. HTML (Pug)

HTML es el componente más básico de la Web. Define el significado y la estructura del contenido web. Además de HTML, generalmente se utilizan otras tecnologías para describir la apariencia/presentación de una página web (CSS) o la funcionalidad/comportamiento (JavaScript). [7]

A lo largo del proyecto, se ha utilizado un motor de plantilla de Node.js, Pug. La razón de utilizar Pug en vez HTML puro, es que seremos capaces de escribir código HTML con una sintaxis mucho más sencilla, clara y directa, tanto a la hora de escribir como de leer y modificar. [8]

En la figura 2.5 se muestra la creación del HTML de la vista 'Mi cuenta'. En dicha figura, podemos observar que a la hora de asignarle una clase a los inputs se ejecuta un pequeño código JavaScript.

Por otra parte, se puede observar que en la primera línea de código de la figura 2.5, el fichero extiende de otro fichero '.pug'. Dicho fichero, es en el que se programa el esquema que compartirán

el resto de ficheros '.pug'. Además, se definen los bloques, que en este proyecto serán *styles*, *content* y *scripts*.

```
extends ../layouts/main-layout.pug

block styles
    link(rel="stylesheet", href="/css/forms.css")
    link(rel="stylesheet", href="/css/account.css")

block content
    main
        include ../layouts/message-popup.pug
        form.account-form(action="/account", method="POST")
            .form-control-account
                label(for="email") #{accountNames.form.email}
                input(type="text", name="email", value=input.email, disabled="true")#email
            .form-control-account
                label(for="userAlias") #{accountNames.form.alias}
                input(class=validationErrors.find(e => e.param === 'userAlias') ? 'invalid' : '', type="text", name="userAlias", value=input.userAlias, disabled="true")#userAlias
            .form-control-account
                label(for="userName") #{accountNames.form.name}
                input(class=validationErrors.find(e => e.param === 'userName') ? 'invalid' : '', type="text", name="userName", value=input.userName, disabled="true")#userName
            .form-control-account
                label(for="userAge") #{accountNames.form.age}
                input(class=validationErrors.find(e => e.param === 'userAge') ? 'invalid' : '', type="number", name="userAge", step="0" value=input.userAge, disabled="true")#userAge
            .form-control-account
                label(for="userCity") #{accountNames.form.city}
                input(class=validationErrors.find(e => e.param === 'userCity') ? 'invalid' : '', type="text", name="userCity", value=input.userCity, disabled="true")#userCity
            input(type="hidden", name="_csrf", value=csrfToken)
            button.btn(type="button")#edit #{accountNames.edit}
            button.btn(type="submit", disabled="true")#save #{accountNames.form.save}
        form.account-form(action="/account/change-password", method="GET")
            button.btn(type="button", disabled="true")#revert #{accountNames.revert}
            button.btn(type="submit", disabled="true")#changePassword #{accountNames.changePassword}
block scripts
    script(src="/js/account.js")
```

Figura 2.5: Ejemplo de creación de un HTML con Pug, 'account.pug'.

En el esquema que comparten todos los ficheros, se definirán la inicialización del fichero y la barra de navegación. En la figura 2.6 se muestra una parte del fichero.

- **Block styles:** En este bloque se definirán los archivos CSS que se utilizarán en el fichero.
- **Block content:** En este bloque se definirá todo el contenido del HTML, las listas, los form, los inputs...
- **Block scripts:** Al igual que en bloque de *styles* se definirán archivos, pero en este caso JavaScript. Dichos archivos serán necesarios a la hora de ejecutar en el navegador.

```
doctype html
html(lang="en")
    head
        meta(charset="UTF-8")
        meta(http-equiv="X-UA-Compatible", content="IE=edge")
        meta(name="viewport", content="width=device-width, initial-scale=1.0")
        title #{pageTitle}
        link(rel="stylesheet", href="/css/main.css")
        block styles
```

Figura 2.6: Inicialización de los ficheros '.pug', 'main-layout.pug'.

2.1.3. CSS

CSS es el lenguaje de estilos utilizado para describir la presentación de documentos HTML o XML. CSS describe como debe ser renderizado el elemento estructurado en la pantalla, en papel, en el habla o en otros medios. [9]

En este proyecto, CSS ha sido utilizado para dar color, aspecto y forma personalizada a los elementos HTML que componen las páginas. Por otra parte, CSS ha sido utilizado para posicionar los elementos HTML correctamente en la página.

En la figura 2.7 se muestra un ejemplo de un fichero CSS. En dicho fichero, se muestra como se da estilo a distintas clases, que más tarde serán asociadas a elementos HTML dentro del fichero '.pug'.

```
.index-button {
    padding: 0.75rem 1.5rem;
    font-size: large;
    font-weight: bold;
    border: 2px solid #009159;
    color: #009159;
    margin-right: 1.5rem;
    margin-top: 1rem;
    margin-bottom: 1rem;
    border-radius: 12px;
}

.user-message {
    margin: auto;
    font-weight: bold;
    width: 25rem;
    max-width: 90%;
    border: 1px solid #2e62a7;
    padding: 0.5rem;
    border-radius: 3px;
    background: #a6e3ff;
    text-align: center;
}

.user-message--error {
    border-color: red;
    background: rgb(255, 176, 176);
    color: red;
}

.text {
    font-size: medium;
    font-weight: bold;
}

```

Figura 2.7: Ejemplo de código CSS, 'main.css'.

2.1.4. JavaScript

JavaScript es un lenguaje de programación ligero, interpretado, o compilado justo-a-tiempo con funciones de primera clase. Si bien es más conocido como un lenguaje de scripting (secuencias de comandos) para páginas web, y es usado en muchos entornos fuera del navegador, tal como Node.js. [10]

A lo largo de este TFG, JavaScript ha sido utilizado como el lenguaje principal y en el que se basa el proyecto. Tal y como se explica anteriormente, JavaScript es el lenguaje usado en el entorno de Node.js (server-side), pero además se puede utilizar en el lado del cliente (client-side).

A la hora de resolver algunos problemas visuales o de interfaz, se han creado distintos scripts en JavaScript para que puedan ser ejecutados desde el lado del cliente. Dicha funcionalidad, permite

hacer páginas dinámicas sin necesidad de recargar la página cada vez que se hace un cambio.

La figura 2.8 muestra un ejemplo de un script del lado del cliente. En dicho script se hace una función para que a la hora de mover la barra de longitud en la vista de 'Generar contraseña' se cambie el número junto a la barra.

```
const slider = document.querySelector("#lengthSlider");
const number = document.querySelector("#lengthNumber");

function sliderClickHandler() {
    let sliderValue = document.querySelector("#lengthSlider").value;
    number.value = sliderValue;
}

function numberClickHandler() {
    let numberValue = document.querySelector("#lengthNumber").value;
    slider.value = numberValue;
}

slider.addEventListener('change', sliderClickHandler);
slider.addEventListener('mousemove', sliderClickHandler);
number.addEventListener('change', numberClickHandler);
```

Figura 2.8: Ejemplo de un script en el client-side, 'slider.js'.

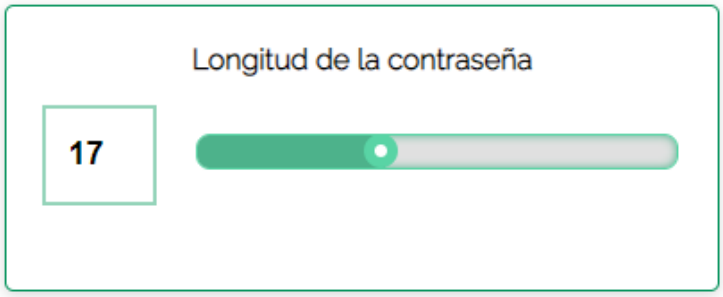

Figura 2.9: Barra de longitud en 'Generar Contraseña'.

2.1.5. JSON

JSON (notación de objetos javascript) es un formato de intercambio de datos. Es muy parecido a un subconjunto de sintaxis JavaScript, aunque no es un subconjunto en sentido estricto. Aunque muchos lenguajes de programación lo soportan, JSON es especialmente útil al escribir cualquier tipo de aplicación basada en JavaScript, incluyendo sitios web y extensiones del navegador. [11]

A lo largo de esta aplicación, JSON será utilizado como el formato principal de envío y recepción de datos entre ficheros '.pug' y Node.js. Además, por el hecho de utilizar una base NoSQL con MongoDB Atlas, los documentos que se guardan en la base de datos son documentos BSON, haciendo más simple la interacción de la base de datos.

En la figura 2.10 se muestra un ejemplo en el cual se envía al fichero '.pug' un objeto JSON. Dicho objeto, será interpretado para dar valor a algunos objetos HTML.

```
39          return res
40              .status(200)
41              .render('menus/menu', {
42                  path: '/menu',
43                  pageTitle: req.t('pageTitles.menusTitles.menu'),
44                  navNames: req.t('nav'),
45                  menuNames: req.t('menuView'),
46                  errorMessage: error,
47                  validationErrors: [],
48                  message: message
49              });
```

Figura 2.10: Ejemplo de envío de un JSON a un fichero '.pug', 'menus.js'.

2.1.6. MongoDB

MongoDB es un sistema de base de datos NoSQL orientado a documentos de código abierto, que en lugar de guardar los datos en tablas lo hace en estructuras de datos BSON (similar a JSON) con un esquema dinámico. Una de las principales características a destacar de MongoDB, sin duda sería la velocidad, que alcanza un balance perfecto entre rendimiento y funcionalidad gracias a su sistema de consulta de contenidos. [12]

La razón por la cual se ha elegido esta base de datos, es que junto a Node.js pertenece al MEAN Stack. Al pertenecer a este conjunto de subsistemas, la compatibilidad entre ambas tecnologías es muy alta y el soporte proporcionado por la comunidad igual.

Por todas estas razones y que su mayor característica es la escalabilidad, hacen de MongoDB la mejor opción para utilizar en este proyecto junto a Node.js.

2.2. Herramientas

2.2.1. VSCode

VSCode es un editor de código fuente desarrollado por Microsoft. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. Además, VSCode es compatible con varios lenguajes de programación y un conjunto de características que pueden o no estar disponibles para un lenguaje dado. [13]

En esta aplicación, VSCode ha sido utilizado como el entorno de trabajo principal. Desde el editor se ha generado la estructura de ficheros, se ha escrito la mayoría del código de la aplicación, se ha manejado el control de versiones...

En la figura 2.11 se muestra la estructura de ficheros del proyecto. A continuación, se describirá brevemente cada carpeta mostrada en la figura anteriormente citada.

- **ROOT:** El directorio raíz contendrá el fichero JavaScript principal de la aplicación, un fichero con las variables de entorno, ficheros para el control de versiones y ficheros relacionados con npm.
- **.vscode:** Contiene información con respecto al debugger de VSCode.

- **controllers:** En esta carpeta se guardan los ficheros JavaScript que contendrán la lógica de respuesta a una petición.
- **images:** Contiene distintas imágenes estáticas que podrán ser vistas en la aplicación.
- **locales:** En este directorio se guardarán los ficheros JSON necesarios para las traducciones a distintos idiomas. En el caso de este proyecto serán el inglés y castellano.
- **middleware:** En esta carpeta se guardan los middlewares. Un middleware es un bloque de código que se ejecuta entre la petición que hace el usuario (request) hasta que la petición llega al servidor.
- **models:** Este directorio contendrá los modelos de las tablas de la base de datos.
- **node_modules:** Contiene todos los ficheros de las dependencias instaladas vía npm.
- **public:** En esta carpeta se guardan todos los ficheros '.css' y todos los scripts del lado del cliente.
- **routes:** En este directorio se guardan todos los ficheros relacionados con el routing de la aplicación.
- **test:** Esta carpeta contiene todos los test creados para probar la aplicación.
- **util:** Esta carpeta contiene distintas funciones comunes a varios lugares de la aplicación.
- **views:** En este directorio se guardan todos las vistas utilizadas en la aplicación.

2.2.2. GitHub

GitHub es una forja (plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones Git. Se utiliza principalmente para la creación de código fuente de programas de ordenador. [14]

En este proyecto, Github ha sido utilizado como el repositorio en la nube para guardar todos los ficheros del proyecto. La metodología de funcionamiento que se llevó a cabo fue la que se muestra en la figura 2.12.

- **Local:** Es el directorio raíz del proyecto.
- **Repositorio Local:** Es el repositorio donde se guardan todos los commits realizados. Este repositorio se encuentra en el propio ordenador donde se esté realizando las acciones.
- **Repositorio Remoto:** Es el repositorio donde se guardan todos los commits realizados, pero a diferencia que en el repositorio local, es accesible por distintos usuarios debido a que se encuentra en la nube (GitHub).

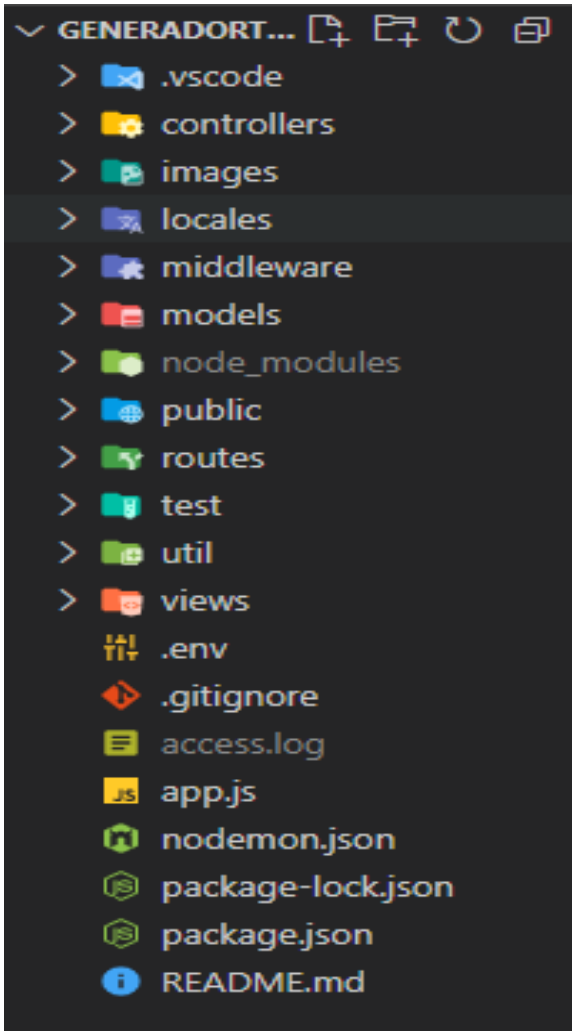

Figura 2.11: Estructura de ficheros del proyecto.

2.2.3. npm

npm es un administrador de paquetes para el lenguaje de programación JavaScript que proporciona alojamiento para el desarrollo de software y el control de versiones con el uso de Git. [15]

En este proyecto, npm fue utilizado como herramienta principal para instalar distintos tipos de paquetes necesarios a la hora de desarrollar la aplicación. A continuación, se explicará brevemente los distintos paquetes instalados:

- **Production Dependencies**
  - **@sendgrid/mail:** Este paquete nos permitirá enviar correos.
  - **bcryptjs:** Con este paquete seremos capaces de crear un hash o comprobar si un input coincide con el hash que se haya guardado. [16]
  - **compression:** Este paquete comprime los ficheros enviados al cliente. [17]

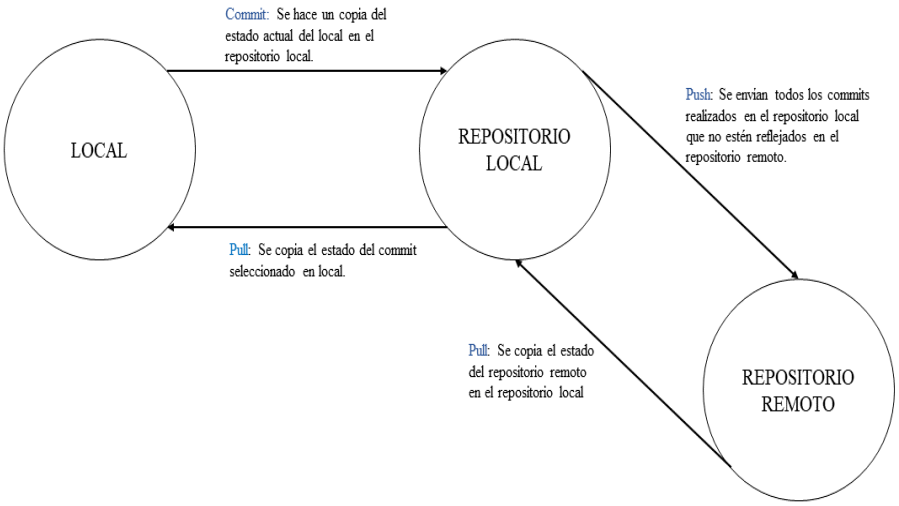

Figura 2.12: Ejemplo de funcionamiento de GitHub.

- **connect-flash:** Con este paquete se podrá guardar un mensaje de error o de información que se acabará mostrando al cliente. [18]
- **connect-mongodb-session:** Este paquete nos permitirá guardar las sesiones en la base de datos. [19]
- **csurf:** Nos proporcionará seguridad frente a un ataque CSRF. [20]
- **dotenv:** Este paquete cargará las variables de entorno guardadas en el fichero '.env'. [21]
- **express:** Este paquete ha sido explicado anteriormente. [2]
- **express-session:** Con este paquete seremos capaces de manejar las sesiones. [22]
- **express-validator:** Este paquete nos permitirá validar la información que le llega al servidor. [4]
- **helmet:** Nos proporcionará seguridad añadiendo distintas cabeceras HTTP. [23]
- **i18next, i18next-fs-backend, i18next-http-middleware:** Con estos tres paquetes se podrá internacionalizar los textos de la aplicación. [24]
- **mongodb:** Es el driver oficial de MongoDB para Node.js. [25]
- **mongoose:** Este paquete nos facilitará la conexión con la base de datos dentro del código. [6]
- **morgan:** Con este paquete seremos capaces de crear un fichero '.log' en el que guardar todos los logs de la aplicación. [26]
- **multer:** Es utilizado para subir ficheros desde el cliente al servidor. [27]

- **pug:** Este paquete ha sido explicado anteriormente. [8]

- **Development Dependencies**
  - **chai:** Este paquete contiene una biblioteca de aserciones BDD/TDD para el nodo y el navegador que se puede combinar con cualquier marco de prueba de JavaScript. [28]
  - **mocha:** Con este paquete podremos ejecutar en serie los test, permitiendo informes flexibles y precisos, al tiempo que asigna las excepciones no detectadas a los casos de prueba correctos. [29]
  - **nodemon:** Este paquete reinicia automáticamente la aplicación cuando se detectan cambios de archivo en el directorio. [30]
  - **sinon:** Con este paquete podremos manipular la funcionalidad de distintas funciones o métodos para que nos devuelvan algo preciso. [31]

Por otro lado, npm se puede utilizar para hacer pequeños scripts. En las figuras 2.13 y 2.14 se muestra el fichero en el que npm se basa para instalar las dependencias y ejecutar los scripts.

```
{
  "name": "generadortfgupv2021",
  "version": "1.0.0",
  "description": "Aplicación web para la generación y almacenamiento de contraseñas",
  "main": "app.js",
  "engines": {
    "node": "15.4.0"
  },
  "scripts": {
    "start": "export NODE_ENV=production || SET NODE_ENV=\"production\" && node app.js",
    "start-server": "SET NODE_ENV=\"development\" && node app.js",
    "start:dev": "nodemon app.js",
    "test": "SET NODE_ENV=\"test\" && mocha --timeout 5000"
  },
```

Figura 2.13: Scripts npm, 'package.json'.

2.2.4. MongoDB Compass

MongoDB Compass es la GUI de MongoDB. Compass permite analizar y comprender el contenido de los datos sin un conocimiento formal de la sintaxis de consulta de MongoDB. Además de explorar los datos en un entorno visual, también se puede utilizar Compass para optimizar el rendimiento de las consultas, administrar índices e implementar la validación de documentos. [32]

A lo largo de este proyecto, MongoDB Compass ha sido utilizado para comprobar que la información se guardaba, editaba o borraba correctamente en la base de datos adecuada.

En la figura 2.15 se muestra una captura de Compass mostrando una tabla de la base de datos.

2.2.5. SendGrid

SendGrid es un servicio de correo electrónico basado en la nube que ofrece un sistema confiable de entrega de correo electrónico transaccional, escalabilidad y análisis en tiempo real junto, con API flexibles que facilitan la integración personalizada. [33]

En este TFG, SendGrid ha sido utilizado principalmente para el envío de correo electrónico automáticamente, por ejemplo, a la hora de registrarte, recuperar la contraseña o confirmar tu cuenta.

En la figura 2.16 se muestra un ejemplo de correo electrónico.

```
25    "dependencies": {
26      "@sendgrid/mail": "^7.4.2",
27      "bcryptjs": "^2.4.3",
28      "compression": "^1.7.4",
29      "connect-flash": "^0.1.1",
30      "connect-mongodb-session": "^2.4.1",
31      "csurf": "^1.11.0",
32      "dotenv": "^8.2.0",
33      "express": "^4.17.1",
34      "express-session": "^1.17.1",
35      "express-validator": "^6.10.1",
36      "helmet": "^4.5.0",
37      "i18next": "^20.2.1",
38      "i18next-fs-backend": "^1.1.1",
39      "i18next-http-middleware": "^3.1.1",
40      "mongodb": "^3.6.6",
41      "mongoose": "^5.12.5",
42      "morgan": "^1.10.0",
43      "multer": "^1.4.2",
44      "pug": "^3.0.2"
45    },
46    "devDependencies": {
47      "chai": "^4.3.4",
48      "mocha": "^8.3.2",
49      "nodemon": "^2.0.7",
50      "sinon": "^10.0.0"
51    }
```

Figura 2.14: Dependencias instaladas vía npm, 'package.json'.

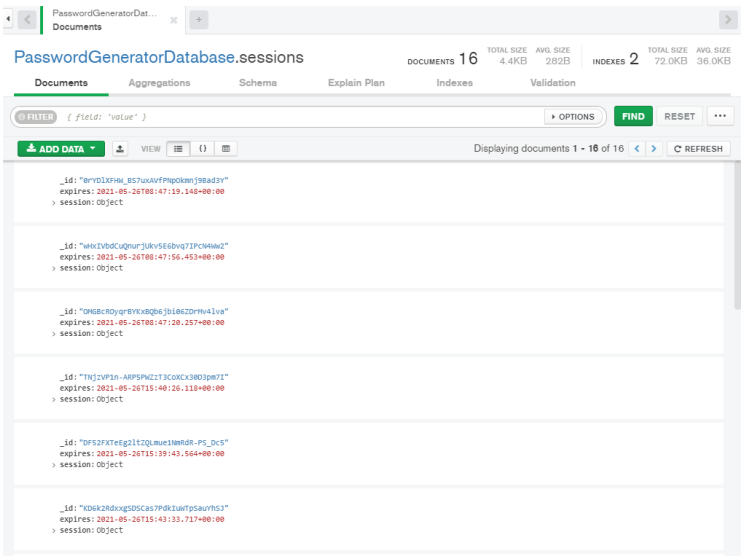

Figura 2.15: Captura de MongoDB Compass de la tabla 'sessions'.

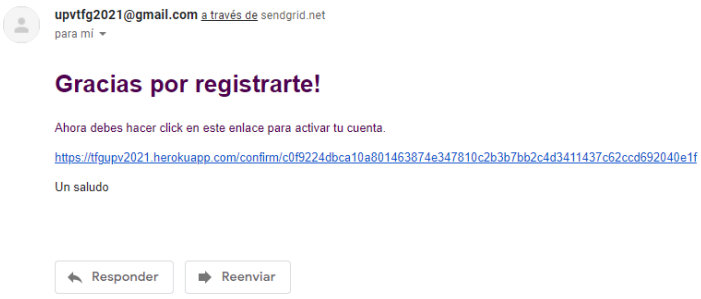

Figura 2.16: Captura de un correo electrónico enviado vía SendGrid.

Capítulo 3

Análisis

3.1. Requisitos

A continuación se listará los requisitos establecidos para el desarrollo de la aplicación:

- La aplicación debe proporcionar contraseñas seguras.
- Las contraseñas deben ser únicas.
- El cliente debe introducir ciertos datos para generar una contraseña.
- Los datos a introducir por el cliente serán; nombre de la contraseña, longitud de la contraseña, dificultad de la contraseña y 3 pistas con su correspondiente respuesta.
- Podrá volverse a generar una contraseña ya creada.
- No se guardará ningún valor en plano en la base de datos, en cambio se guardará un hash de cada valor y cada contraseña.
- El cliente podrá visualizar todas las contraseñas que tenga creadas.
- Se podrá borrar una contraseña pero no editarla.
- Existirá un registro y login y será necesario registrarse para usar la aplicación.
- Para generar una contraseña será necesario confirmar el correo electrónico.
- Existirá un mecanismo de recuperación de contraseña de la cuenta.
- Estará adaptado para todo tipo de tamaños de pantalla.
- La aplicación estará en castellano e inglés y se podrá implementar de manera simple otro idioma.

3.2. Base de datos

Para que la aplicación pueda mantener los datos entre distintas sesiones se ha utilizado una base de datos. El tipo de base de datos que se ha utilizado ha sido NoSQL. Este tipo de base de datos te permite guardar la información de una manera menos estricta con respecto a SQL. La base de datos utilizada ha sido MongoDB Atlas. En dicha base, se guardará los datos de la cuenta del usuario, los datos necesarios para regenerar la contraseña y las sesiones. En ningún momento se guardará ninguna contraseña en plano en la base de datos.

3.3. Interfaces

Las interfaces necesarias para desarrollar la aplicación serán:

- **Pre-login:**
  - **Inicio:** Será la interfaz principal de la aplicación, en ella se mostrará un texto explicando cierta información sobre la seguridad e importancia de las contraseñas.
  - **Conectarse:** En esta pantalla el cliente tendrá la posibilidad de iniciar sesión con un usuario ya registrado. En el caso de que el cliente no se acordase de la contraseña, existirá la opción de recuperar la contraseña.
  - **Registrarse:** Esta interfaz será en la que el cliente podrá registrarse y crear una cuenta nueva.
- **Post-login:**
  - **Menú:** Es la misma interfaz que 'Inicio' pero cambiando los botones.
  - **Guía:** En esta pantalla se mostrará una breve guía de uso de la aplicación.
  - **Generador de contraseñas:** Esta interfaz será en la que el cliente será capaz de generar contraseñas.
  - **Contraseñas guardadas:** Será la pantalla en la que el cliente podrá ver las contraseñas que tiene generadas en la aplicación.
  - **Visualizador de contraseña:** Con esta interfaz el cliente podrá ver la contraseña que ha generado. Dicha contraseña solo será visible mientras esta interfaz esté abierta.
  - **Mi cuenta:** Esta pantalla mostrará algunos datos del cliente y podrá editarlos todos a excepción del correo electrónico.
- **Ambos casos:**
  - **Errores:** Serán las interfaces en las que se muestre que ha ocurrido un error, por ejemplo 403, 404 o 500.

3.4. Aplicación

La aplicación se podrá dividir en dos secciones, la pre-autenticación y la post-autenticación. En esta primera sección de pre-autenticación, el usuario solo tendrá disponibles las interfaces de conectarse y registrarse.

A la hora de autenticarse, el usuario necesitará conectarse o registrarse. En el caso de registrarse, se tendrá que introducir un correo electrónico, un alias, la contraseña y la confirmación de la contraseña. Si los datos introducidos fuesen correctos, el usuario sería redirigido al menú principal de la aplicación con una sesión autenticada. Por el otro lado, al momento de conectarse, el usuario solo tendrá que introducir un correo electrónico registrado y la contraseña correspondiente. Al igual que al registrarse, si los datos introducidos fuesen correctos, el usuario sería redirigido al menú principal de la aplicación.

En la figura 3.1 se mostrará un esquema de autenticación de la aplicación.

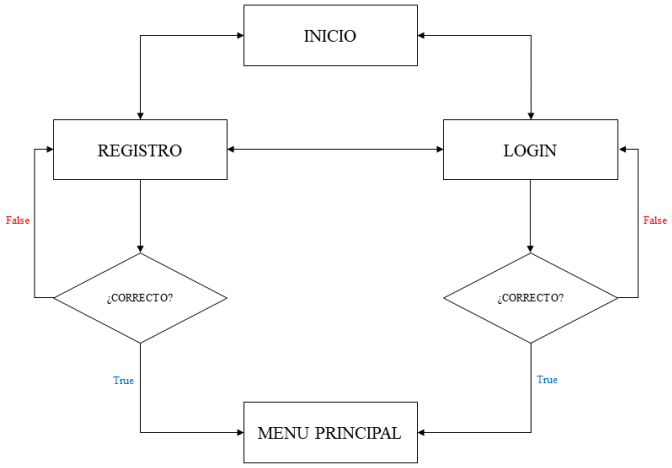

Figura 3.1: Esquema de autenticación.

En la segunda sección, la post-autenticación, el usuario podrá acceder a todas las funcionalidades de la aplicación. Para que el usuario tenga acceso a todas las funcionalidades debe haber confirmado su correo electrónico. En el caso de que no haya confirmado su correo electrónico, solo se podrá acceder a las interfaces de la aplicación, pero no se podrá crear ninguna contraseña.

- **Cuenta:** Se podrán modificar los datos de la cuenta; el alias, el nombre, la edad y la ciudad.
- **Contraseñas Generadas:** En este menú el usuario tendrá una lista de todas las contraseñas que ha generado hasta el momento. Además, tendrá la posibilidad de eliminarlas o regenerarlas. A la hora de regenerar la contraseña, el usuario tendrá que introducir las respuestas a las pistas que haya introducido cuando creo la contraseña. Si los datos introducidos fuesen correctos, el usuario sería redirigido a una interfaz en la que podría ver la contraseña mientras no cerrase la pestaña o cambiase de menú.
- **Generar Contraseña:** Para poder utilizar este menú el usuario debe haber confirmado su correo electrónico. Después de haber confirmado el correo, el usuario podrá introducir ciertos datos para poder generar la contraseña. Dichos parámetros serán; el nombre, la longitud, la dificultad y hasta 3 pistas con sus correspondientes respuestas. Una vez que todos los datos han sido introducidos correctamente se generará una contraseña. La contraseña generada

solo será visible mientras no se cierre la pestaña o se cambie de menú.

En la figura 3.2 se muestra un esquema general de uso de la aplicación. En dicho esquema se presupone que el usuario ya ha confirmado su correo electrónico.

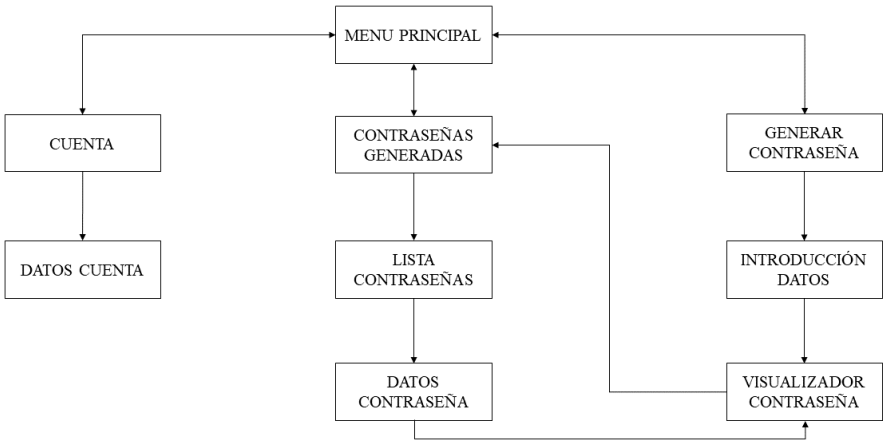

Figura 3.2: Esquema general de uso de la aplicación.

Capítulo 4

Diseño de la aplicación

4.1. Diseño base de datos

En la figura 4.1 se muestra una imagen con el esquema general de la base de datos.

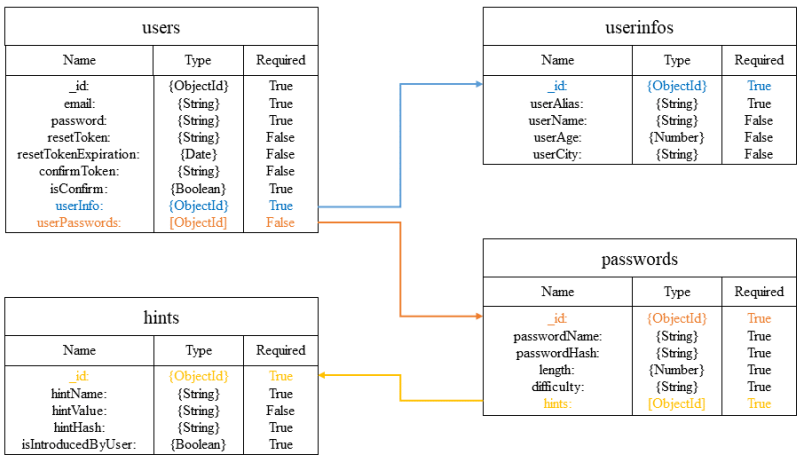

Figura 4.1: Base de datos.

A continuación, se explicará cada tabla mostrada en la figura 4.1:

- **users:**
  - *_id:* Es el identificador del usuario.
  - *email:* Es el email del usuario.
  - *password:* En esta columna se guardará el hash de la contraseña del usuario.

  - *resetToken:* Será un string generado aleatoriamente para que el usuario puede restablecer su contraseña.
  - *resetTokenExpiration:* Será una fecha límite para que el usuario pueda utilizar el 'resetToken' y restablecer su contraseña.
  - *confirmToken:* Al igual que 'resetToken' será un string generado aleatoriamente. En este caso, servirá para que el usuario pueda confirmar su cuenta.
  - *isConfirm:* Es un boolean en el que se indica si el usuario ha confirmado su cuenta.
  - *userInfo:* Es un identificador que hace referencia a una tabla de 'userinfos'.
  - *userPasswords:* Es un array de identificadores en el que cada uno hace referencia a una tabla de 'passwords'.

- **userinfos:**
  - *_id:* Es el identificador de la información del usuario.
  - *userAlias:* Es el alias del usuario.
  - *userName:* Es el nombre del usuario.
  - *userAge:* En esta columna se indica la edad del usuario.
  - *userCity:* En esta columna se indica la ciudad del usuario.

- **passwords:**
  - *_id:* Es el identificador de la contraseña.
  - *passwordName:* Es el nombre de la contraseña.
  - *passwordHash:* En esta columna se guardará el hash del valor de la contraseña.
  - *length:* Es la longitud que tiene la contraseña.
  - *difficulty:* Es la dificultad que de la contraseña.
  - *hints:* Es un array de identificadores en el que cada uno hace referencia a una tabla de 'hints'.

- **hints:**
  - *_id:* Es el identificador de la pista.
  - *hintName:* Es el nombre de la pista.
  - *hintValue:* En esta columna se guardan los valores que no es peligroso guardar en plano, como por ejemplo, el email y el nombre de la contraseña.
  - *hintHash:* En esta columna se guardará el hash del valor de la pista.
  - *isIntroducedByUser:* Es un boolean en el que se indica si el usuario ha sido el que ha introducido dicho dato. Por ejemplo, el email no es introducido por el usuario.

- **sessions:** A pesar de no mostrarse en la figura anteriormente mencionada, esta tabla existe en la base de datos. Los paquetes de 'express-session' y 'connect-mongodb-session' crearán y guardarán automáticamente las sesiones de cada usuario en la base de datos. Debido a estos paquetes, no es necesario diseñar ni crear las tablas manualmente.

4.2. Diseño de la aplicación

4.2.1. Front End

El Front End es la parte de una aplicación que interactúa con los usuarios, es conocida como el lado del cliente. En esta aplicación, el Front End se generará en el servidor mediante el paquete 'Pug'.

Pug nos permite generar en el lado del servidor un HTML que podrá ser enviado como respuesta a una petición hecha por el cliente. Esta funcionalidad nos permite tener un mayor control sobre los HTML de la aplicación y por consiguiente, sobre el Front End.

En la figura 4.2 se muestra un ejemplo completo de uso de Pug. En dicho ejemplo, se muestra el HTML que se envía a la hora de mostrar la contraseña.

```
1  extends ../layouts/main-layout.pug
2
3  block styles
4      link(rel="stylesheet", href="/css/generator.css")
5
6  block content
7      main
8          include ../layouts/message-popup.pug
9          div.centered
10             div.show-password-centered
11                 h1 #{showPasswordNames.textPrePassword}
12                 div.show-password-wrap
13                     div.show-password-wrap-password
14                         h1.show-password-password#password #{password}
15                 button.btn.btn-copy(type="button")#copy #{showPasswordNames.copy.button}
16                     div.popup
17                         span.popuptext#myPopup #{showPasswordNames.copy.text}
18                 h2 #{showPasswordNames.text.firstLine}
19                 h2 #{showPasswordNames.text.secondLine}
20                 form(action="/generator", method="GET")
21                     button.btn.index-button(type="submit") #{showPasswordNames.form.generate}
22                 form(action="/storage", method="GET")
23                     button.btn.index-button(type="submit") #{showPasswordNames.form.saved}
24 block scripts
25     script(src="/js/copyPassword.js")
```

Figura 4.2: Ejemplo completo de uso de Pug, 'show-password.pug'.

Uno de los requisitos del proyecto, era el de que la aplicación estuviese adaptada a todo tipo de tamaños de pantalla. Para lograr esto se utilizó el lenguaje de programación CSS. CSS, además de dar color, forma y aspecto a los elementos de un HTML es el encargado de posicionar los elementos. Para poder hacer la aplicación adaptable a todo tipo de pantalla, se intentó en su mayor medida hacer los elementos flexibles y que las unidades estuviesen en ' %' en vez de en píxeles o rem.

En la figura 4.3 se muestra el estilado que se le dio a distintas clases utilizadas en el HTML generado en la figura 4.2.

Otro de los requisitos del proyecto, era el de que la aplicación estuviese en distintos idiomas. Para lograr este requisito, se utilizaron distintos paquetes de internacionalización, 'i18next', 'i18next-fs-backend' e 'i18next-http-middleware'. Con dichos paquetes lo único que se tenía que hacer era crear un archivo JSON por idioma en el que guardar todas las traducciones. Después de crear dichos archivos, era necesario inicializar los paquetes en la aplicación. Esta inicialización era

```css
.show-password-centered {
    max-width: 90%;
    margin: auto;
    display: -webkit-box;
    display: -ms-flexbox;
    display: inline-block;
    -ms-flex-wrap: wrap;
    flex-wrap: wrap;
}

.show-password-wrap {
    display: -webkit-box;
    display: -ms-flexbox;
    margin: 0.5rem auto 1rem;
    display: flex;
    width: 100%;
    -ms-flex-wrap: wrap;
    flex-wrap: wrap;
}

.show-password-wrap-password {
    border: 3px solid #009159;
    border-radius: 5px;
    background-color: hsla(160, 100%, 75%, 0.5);
    font-size: larger;
    font-weight: bolder;
    -webkit-box-shadow: 0 2px 8px rgba(0,0,0,.15);
    box-shadow: 0 2px 8px rgba(0,0,0,.15);
    margin: auto;
    display: block;
    width: 25rem;
    height: 5rem;
    text-align: center;
}
```

Figura 4.3: Estilado de varias clases con CSS, 'generator.css'.

la encargada de detectar el idioma en el que estaba siendo usado el navegador, y como consiguiente elegía el archivo JSON respectivo a dicho idioma. En el caso de que no existiese el idioma, el paquete elegía el idioma por defecto, que en este TFG será el castellano.

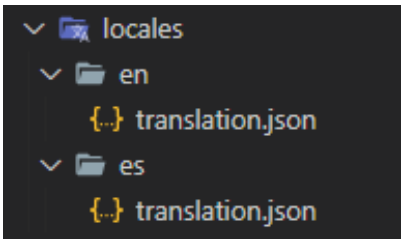

Figura 4.4: Localización de los ficheros de traducciones.

En las figuras 4.5 y 4.6 se muestra los ficheros JSON en los que se guardaban las traducciones del castellano e inglés utilizadas en el HTML de la figura 4.2.

Uno de los problemas que surgen a la hora de crear el HTML en el servidor es que es estático. Una mejora para evitar que sea tan estático es la implementación de ficheros JavaScript. A lo largo de aplicación se han utilizado múltiples ficheros JavaScript para hacer más dinámica las páginas HTML.

```
372     "showPasswordView": {
373         "form": {
374             "generate": "Generar otra contraseña",
375             "saved": "Contraseñas guardadas"
376         },
377         "textPrePassword": "La contraseña generada es:",
378         "copy": {
379             "button": "Copiar",
380             "text": "¡Copiado!"
381         },
382         "text": {
383             "firstLine": "Recuerde que solo podrá ver la contraseña en este momento.",
384             "secondLine": "Si quisieras volverla a ver, tendrías que introducir los datos de nuevo."
385         }
386     },
```

Figura 4.5: Traducciones de 'show-password' en castellano, 'translation.json'.

```
372     "showPasswordView": {
373         "form": {
374             "generate": "Generate another password",
375             "saved": "Saved passwords"
376         },
377         "textPrePassword": "The generated password is:",
378         "copy": {
379             "button": "Copy",
380             "text": "Copied!"
381         },
382         "text": {
383             "firstLine": "Remember that you will only be able to see the password at this time.",
384             "secondLine": "If you wanted to see it again, you would have to enter the data again."
385         }
386     },
```

Figura 4.6: Traducciones de 'show-password' en inglés, 'translation.json'.

En la figura 4.7 se muestra un fichero JavaScript utilizado para hacer más dinámica la página HTML creada en la figura 4.2.

```
1   const copy = document.querySelector('#copy');
2
3   function copyToClipboard(text) {
4       const type = 'text/plain';
5       const blob = new Blob([text], {type});
6       let data = [new ClipboardItem({[type]: blob})];
7
8       navigator.clipboard.write(data);
9
10      var popup = document.getElementById("myPopup");
11      popup.classList.toggle("show");
12  }
13
14  copy.addEventListener('click', function() {
15      copyToClipboard(document.getElementById('password').innerHTML.replace(/&amp;/g, '&'));
16  });
```

Figura 4.7: Función para copiar la contraseña, 'copyPassword.js'.

4.2.2. Back End

El Back End es el interior de las aplicaciones que viven en el servidor, es conocido como el lado del servidor. Consiste en un servidor, una aplicación y una base de datos. En este proyecto, el Back End será toda la lógica de la aplicación incluidas las interacciones con la base de datos.

Debido a que la aplicación consta de múltiples rutas y controllers, se podría hacer tedioso y largo el explicar todos. Para explicar el funcionamiento de la aplicación se utilizará como ejemplo el mostrado en la figura 4.13, el login.

A continuación, se explicará con detalle los pasos que se siguen en la figura 4.13:

- **Petición:** Está es la petición que realiza el cliente. En esta aplicación solo se utilizan GET y

POST.

- **app.js:** En esta parte, se procesa la petición y se redirige a la ruta correspondiente. En el caso del login la ruta será ../login (GET o POST). Más tarde se volverá a procesar la respuesta dada por el controller a la aplicación se enviará una respuesta al cliente.

- **Routes:** Cuando la petición pasa por la parte de direccionamiento se realiza una validación de los parámetros que lleva la misma. Aparte de realizarse la validación, la petición pasa por distintos middlewares. Si uno de los middlewares falla, la petición será denegada. Para el caso del login el usuario deberá no estar autenticado. Por último, la petición se redireccionará hacia un controller.

  En la figura 4.8, se muestra como la petición es procesada en el direccionamiento.

```
18  router.get('/login', isNotAuth, authController.getLogin);
19  router.post(
20      '/login',
21      [
22          body('email')
23              .isEmail()
24              .withMessage((value, {req}) => {
25                  return req.t('loginView.validationErrors.email');
26              })
27              .toLowerCase()
28              .trim(),
29          body('password', (value, {req}) => {
30                  return req.t('loginView.validationErrors.pass');
31              })
32              .matches(regex)
33              .trim()
34      ],
35      isNotAuth,
36      authController.postLogin
37  );
```

Figura 4.8: Ejemplo del direccionamiento del login, 'auth.js'.

- **Controllers:** En el controller es donde ocurre toda la lógica de la aplicación. En ella se envía una respuesta hacía el cliente y se conecta con la base de datos.

  En la figura 4.9, se muestra como desde el controller se conecta a la base de datos.

```
64          const user = await User.findOne({
65              email: email
66          });
```

Figura 4.9: Ejemplo de conexión con la base de datos, 'auth.js'.

Para poderse conectar con la base de datos es necesario crear un esquema de la tabla a la que se quiere acceder. Para poder crear el modelo se ha utilizado el paquete de 'mongoose'. Un ejemplo de creación de modelo es el mostrado en la figura 4.10.

En esta aplicación se han utilizado dos tipos de respuesta, el redireccionamiento o el envío de un HTML mediante Pug. En las figuras 4.11 y 4.12 se muestra un ejemplo de cada una respectivamente. Para el caso de envío de un HTML se puede observar como a la hora de responder se envían distintos datos. Dichos datos, serán utilizados para crear el HTML en cuestión.

```
// SCHEMA DECLARATION
const Schema = mongoose.Schema;

// SCHEMA
const userSchema = new Schema({
    email: {
        type: String,
        required: true
    },
    password: {
        type: String,
        required: true
    },
    resetToken: {
        type: String,
        required: false
    },
    resetTokenExpiration: {
        type: Date,
        required: false
    },
    confirmToken: {
        type: String,
        required: false
    },
    isConfirm: {
        type: Boolean,
        required: true,
        default: false
    },
    userInfo: {
        type: Schema.Types.ObjectId,
        ref: 'UserInfo',
        required: true
    },
    userPasswords: [
        {
            type: Schema.Types.ObjectId,
            ref: 'Password',
            required: false
        }
    ]
});

// EXPORTS
module.exports = mongoose.model('User', userSchema);
```

Figura 4.10: Modelo de ususario, 'user.js'.

```
                return res
                    .status(200)
                    .redirect('/menu');
            });
```

Figura 4.11: Ejemplo de redireccionamiento con la petición 'POST ../login', 'auth.js'.

4.2.2.1. Algoritmo de generación de contraseñas

A continuación, se explicará el funcionamiento del algoritmo a la hora de generar una contraseña.

El algoritmo consta de 4 partes; creación de los hashes, elección del paso, construcción del mix y construcción de la contraseña. Las tres primeras partes son la elección y creación de los caracteres y números que se van a utilizar a la hora de generar la contraseña. En cambio, el último paso consiste en la creación de la contraseña mediante los caracteres seleccionados en los pasos anteriores.

- **Creación de lo hashes:** La primera parte de todas es crear los hashes necesarios para la elección de los caracteres y números. Para poder realizar esta tarea, se ha utilizado la ayuda del paquete 'bcryptjs'. Con dicho paquete, haremos un hash del email del usuario, del nombre de la contraseña y del valor de las respuestas a las pistas que el usuario haya introducido.

```
23      return res
24          .status(200)
25          .render('auth/login', {
26              path: '/login',
27              pageTitle: req.t('pageTitles.authTitles.login'),
28              navNames: req.t('nav'),
29              loginNames: req.t('loginView'),
30              errorMessage: error,
31              validationErrors: [],
32              message: message,
33              oldInput: {
34                  email: '',
35                  password: ''
36              }
37          });
```

Figura 4.12: Ejemplo de envío de HTML con la petición '../login', 'auth.js'.

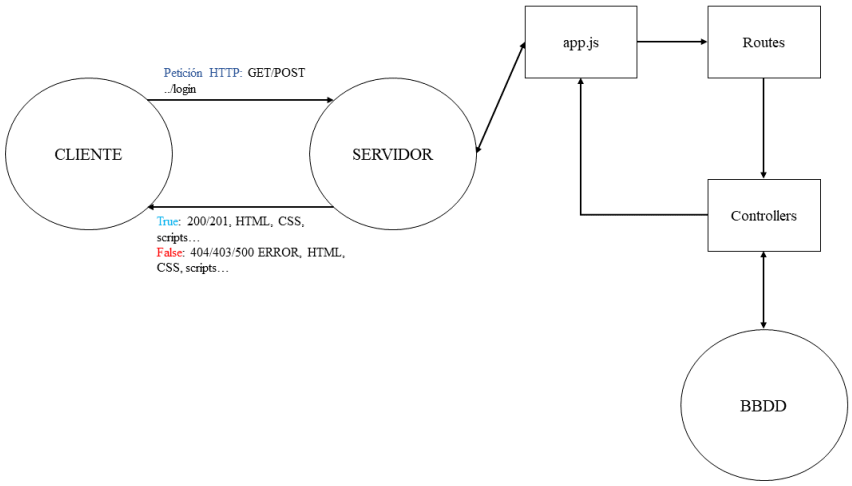

Figura 4.13: Ejemplo de funcionamiento del Back End de la aplicación.

- **Elección del paso:** Como segunda parte, se elegirá un paso de elección de carácter o número de los hashes creados anteriormente. Se pueden separar en 3; el paso de los datos introducidos directamente por el usuario (respuesta a las pistas), el paso de los datos que no son introducidos por el usuario (email, nombre de la contraseña) y el paso final.

  En la figura 4.14 se muestra el procedimiento a seguir en esta segunda parte. Dependiendo de si el usuario ha introducido 1, 2 o 3 datos, se utilizaran distintos números primos para hacer el resto de la operación.

- **Construcción del mix:** En esta parte, el algoritmo necesita crear un mix con el que poder seleccionar los caracteres y números finales de la contraseña. Este mix es creado a partir de los hashes y pasos creados en los partes anteriores. En el punto anterior, se explicó que hay distintos tipos de pasos para distintos hashes. El mix constará de 60 caracteres de longitud, por lo tanto se cogerá de cada hash un valor equitativo de caracteres. Este último valor dependerá del número de pistas que el usuario haya introducido. A continuación, se pasará a explicar separadamente el funcionamiento de creación del mix dependiendo del dato:

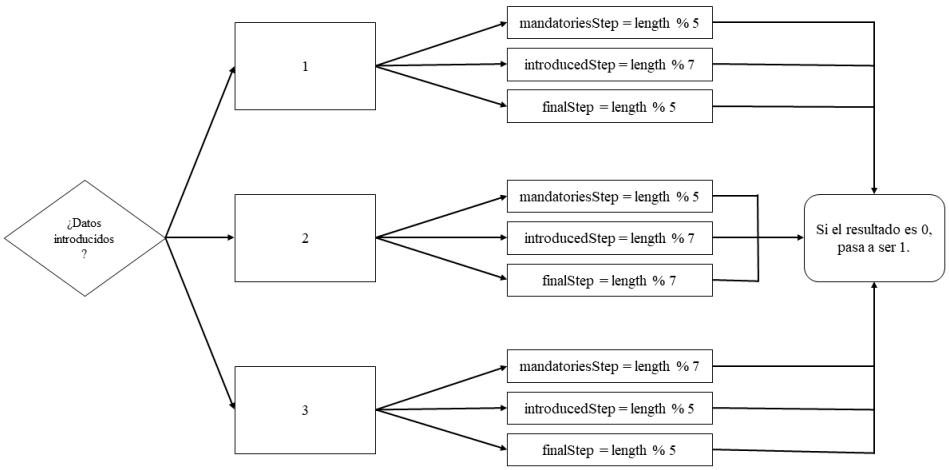

Figura 4.14: Funcionamiento de elección de paso.

- *Introducidos por el usuario:* En este campo, se encuentran las respuestas a las pistas introducidas por el usuario. Para seleccionar los caracteres de este tipo de datos, se utilizará el paso creado explícitamente para ello (introducedStep).
- *Introducidos por el servidor:* En este campo, se encuentran el email del usuario y el nombre de la contraseña. Para seleccionar los caracteres de este tipo de datos, se utilizará el paso creado explícitamente para ello (mandatoriesStep).

Cuando se hayan seleccionado los 60 caracteres, se creará un hash del mix. Dicho hash, será el utilizado en el último paso para construir la contraseña. En la figura 4.15, se muestra un ejemplo final de un mix.

```
"$2a$04$leWO/OrWWDI9NLudLv8rYeE.B9vQC14k5wfKwgf72huSKxwkKMkIe"
```

Figura 4.15: Ejemplo de un mix.

- **Construcción de la contraseña:** En esta última parte, se creará la contraseña a partir del mix del apartado anterior y del paso final. Para crear la contraseña se utilizaron dos tablas de equivalencias. Dichas tablas de equivalencias serán las mostradas en las figuras 4.16 y 4.17.

  El paso será utilizado para elegir un carácter dentro del mix. Dicho carácter dependiendo de la dificultad será cambiado o no. A continuación, se explicará el funcionamiento dependiendo de la dificultad de la contraseña:

  - *Fácil:* Con esta dificultad no se utilizará ninguna tabla. Se añadirá directamente el carácter a la contraseña si es una letra mayúscula o minúscula.
  - *Media:* En esta dificultad se utilizarán ambas tablas. A la hora de crear la contraseña, se hará que el 12.5 % de la contraseña sea números, otro 12.5 % sea símbolos y el restante, letras mayúsculas y minúsculas.

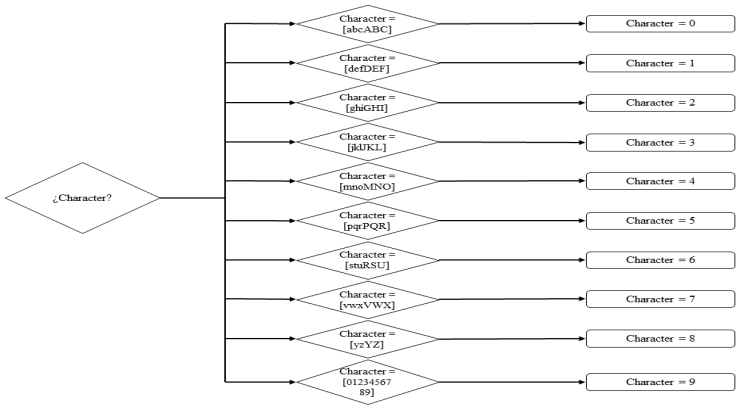

Figura 4.16: Tabla de equivalencias para números.

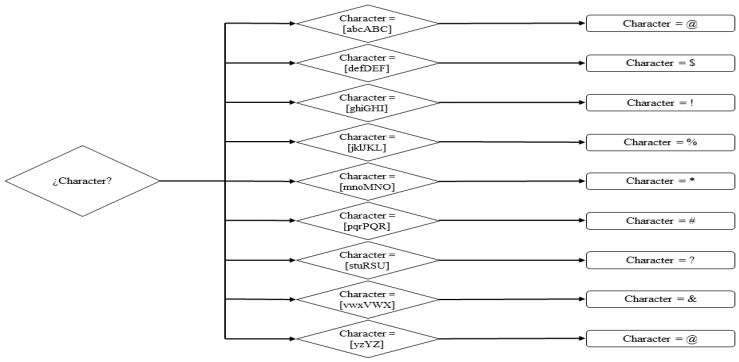

Figura 4.17: Tabla de equivalencias para símbolos.

- *Díficil:* Al igual que en la dificultad media, se utilizarán ambas tablas. A la hora de crear la contraseña, se hará que el 25 % de la contraseña sea números, otro 25 % sea símbolos y el restante, letras mayúsculas y minúsculas.

Capítulo 5

Pruebas y despliegue

5.1. Pruebas

Antes de realizar el despliegue de la aplicación, se hizo un plan de pruebas. El objetivo de este plan de pruebas era el de crear distintos test para probar las funcionalidades principales de la aplicación. Si dichos test eran pasados correctamente, significaría que la aplicación funcionaba. Haciendo estas pruebas se consigue que muchos de los errores a la hora de modificar el código a posteriori, sean fácilmente identificables y localizables.

Para poder hacer los test se utilizaron los paquetes de 'sinon', 'mocha' y 'chai'. Con dichos paquetes, se hizo más simple y menos laboriosa la tarea de realizar los test. Además, la utilización de los paquetes anteriormente mencionados, permitió la utilización de una base de datos mientras se testeaba distintas funciones.

En la figura 5.2, se muestra un ejemplo de test en la aplicación. En dicho ejemplo, se testea el controller de 'account.js', haciendo llamadas a la función de 'getAccount'.

```
141     describe('getAccount', function() {
142         it('should return statusCode 403 and renderValue true if has no user session', async function () {
143             req.user._id = undefined;
144             const getAccount = await AccountController.getAccount(req, res, next);
145             expect(getAccount[0])
146                 .to
147                 .be
148                 .equal(403);
149             expect(getAccount[1])
150                 .to
151                 .be
152                 .true;
153             return;
154         });
155         it('should return statusCode 200 and renderValue true if has user session', async function() {
156             req.user._id = reqUser._id;
157             const getAccount = await AccountController.getAccount(req, res, next);
158             expect(getAccount.statusValue)
159                 .to
160                 .be
161                 .equal(200);
162             expect(getAccount.renderValue)
163                 .to
164                 .be
165                 .true;
166             return;
167         });
168     });
```

Figura 5.1: Tests 'getAccount', 'account.js'.

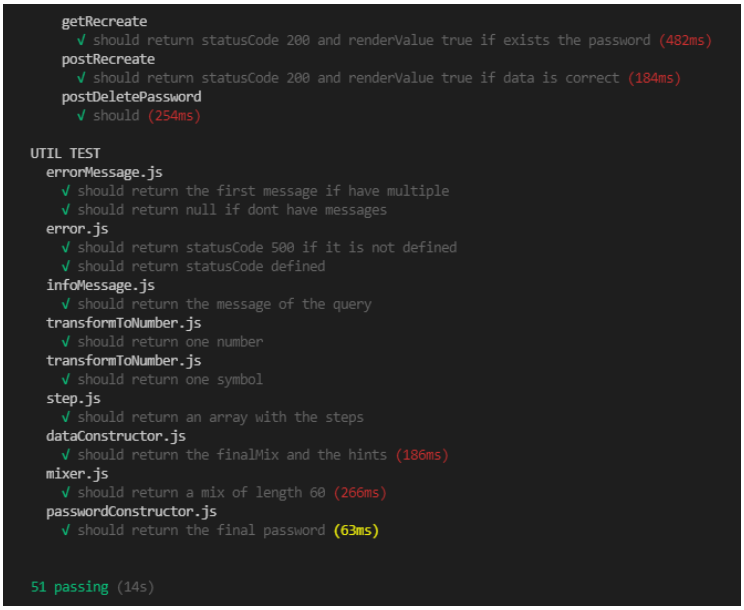

Figura 5.2: Tests pasados correctamente.

5.2. Despliegue

Para finalizar la aplicación, se realizó un despliegue utilizando Heroku. Heroku nos permite subir una aplicación y desplegarla de manera gratuita en la nube. El despliegue puede realizarse de varias maneras, pero la manera que se utilizó en este proyecto fue la de sincronizar GitHub con Heroku.

Heroku facilita la sincronización con GitHub. El proceso de sincronización es simple, primero hay que elegir el repositorio GitHub donde se encuentra la aplicación. Después, hay que elegir la rama que se quiere desplegar. Para acabar, se introducen las variables de entorno necesarias y se configura la aplicación. En este proyecto, no ha sido necesario introducir las variables de entorno, debido a que en la aplicación ya existe un fichero '.env' en el que se guardan dichas variables.

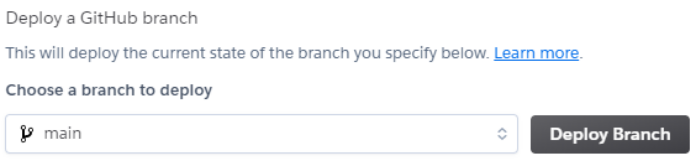

Figura 5.3: Despliegue con Heroku.

Capítulo 6

Seguridad frente ataques

6.1. Base de datos

En este TFG la base de datos se encuentra en la nube gracias al servicio de MongoDB Atlas. La seguridad es gestionada principalmente por MongoDB. A pesar de lo anteriormente mencionado, MongoDB nos permite personalizar dicha seguridad, creando roles en los clústeres y dando acceso a la base de datos mediante una Lista de Acceso IP, un Peering o un Private EndPoint.

En las figuras 6.1 y 6.2 se muestra dos capturas de la página de MongoDB Atlas donde se ha creado un rol y una lista de acceso IP respectivamente.

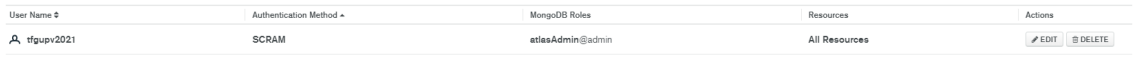

Figura 6.1: Roles de un clúster en MongoDB Atlas.

Figura 6.2: Listas de acceso IP de un clúster en MongoDB Atlas.

Además, si se diese el caso de que esta seguridad fallase y los datos fuesen robados, en la base de datos nunca se guardará ningún valor de una contraseña en texto plano, en cambio se guardará su respectivo hash.

6.2. Aplicación

Al igual que con la base de datos, la aplicación se encontrará en la nube y será gestionada principalmente por Heroku. Heroku nos permite gestionar los colaboradores del proyecto, los certificados ssl, el dominio de la aplicación, los logs... Algunas de las anteriores opciones son de pago, pero para este proyecto no ha sido necesario el pago de ningún servicio.

En la figura 6.3 se muestra una captura de la página de Heroku en la que se muestran los colaboradores del despliegue de la aplicación.

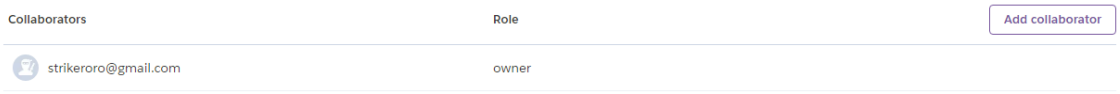

Figura 6.3: Colaboradores del despliegue en Heroku.

Además, la aplicación tiene seguridad ante el ataque CSRF gracias a la dependencia de 'csurf'. Este ataque es un tipo de exploit malicioso de un sitio web en el que comandos no autorizados son transmitidos por un usuario en el cual el sitio web confía.

En la figura 6.4 se muestra un ejemplo en el cual ha sucedido un ataque CSRF. [34]

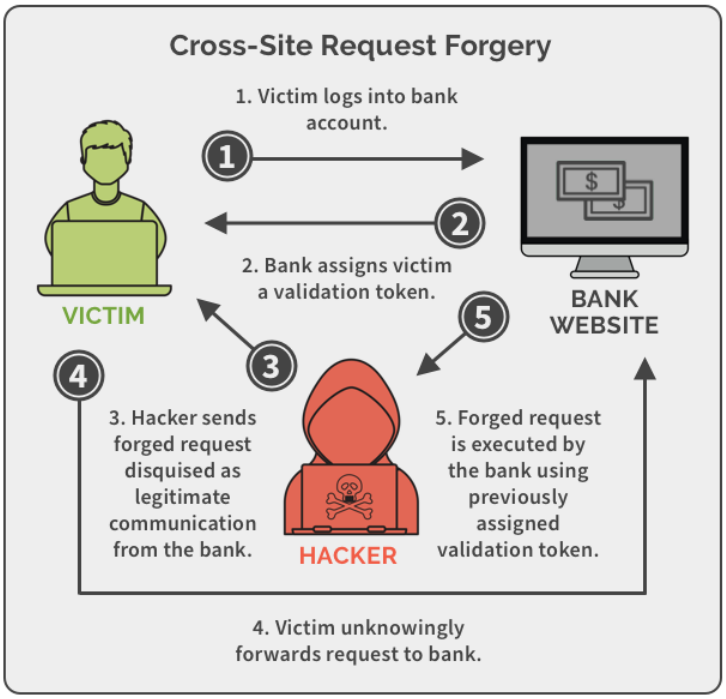

Figura 6.4: Ejemplo de ataque CSRF.

Capítulo 7

Conclusiones y propuesta de futuro

7.1. Conclusiones

El objetivo de este proyecto ha sido el de crear una aplicación web de un generador de contraseñas seguras. Además, dichas contraseñas podrían volverse a generar y tendrían que ser únicas. Para poder realizar la aplicación, se han utilizado distintas herramientas y tecnologías.

Para realizar este proyecto, han sido necesarios los conocimientos adquiridos a lo largo de la carrera, en especial asignaturas relativas a la gestión de la información, la programación y la seguridad. También, ha sido necesario adquirir nuevos conocimientos como el aprendizaje de Node.js, JavaScript, npm...

Para finalizar, se puede decir que el desarrollo de este TFG ha sido de gran interés. Además, ha hecho surgir un interés personal de aprender más respecto al desarrollo de aplicaciones web.

7.2. Propuestas de futuro

A continuación, se listarán las posibles futuras ampliaciones del proyecto:

- Separación completa del Front End y Back End, utilizando React.js para el Front End y una API REST con Node.js para el Back End.
- Adición de nuevos idiomas como el valenciano o francés.
- Posibilidad de registrarse mediante un certificado digital.
- Añadir más personalización a las contraseñas; número de siglas, número de números, número de letras...
- Añadir una foto de perfil.
- Crear un chat en tiempo de real de ayuda.
- Mejorar visualmente los correos que se envían a través de SendGrid.

Bibliografía

[1] OpenJS Foundation. *Documentación Node.js*. URL: https://nodejs.org/es/docs/.

[2] OpenJS Foundation. *Web Express.js*. URL: https://expressjs.com/.

[3] OpenJS Foundation. *Direccionamiento Express.js*. URL: https://expressjs.com/es/guide/routing.html.

[4] express-validator. *Documentación express-validator*. URL: https://express-validator.github.io/docs/.

[5] MongoDB Inc. *Documentación MongoDB Atlas*. URL: https://docs.atlas.mongodb.com/.

[6] LearnBoost. *Documentación Mongoose*. URL: https://mongoosejs.com/docs/guide.html.

[7] Mozilla. *Documentación HTML*. URL: https://developer.mozilla.org/es/docs/Web/HTML.

[8] Pug. *Documentación Pug*. URL: https://pugjs.org/api/getting-started.html.

[9] Mozilla. *Documentación CSS*. URL: https://developer.mozilla.org/es/docs/Web/CSS.

[10] Mozilla. *Documentación JavaScript*. URL: https://developer.mozilla.org/es/docs/Web/JavaScript.

[11] Mozilla. *Glosario JSON*. URL: https://developer.mozilla.org/es/docs/Glossary/JSON.

[12] MongoDB. *Manual MongoDB*. URL: https://docs.mongodb.com/manual/.

[13] Wikipedia. *Wikipedia VSCode*. URL: https://es.wikipedia.org/wiki/Visual_Studio_Code.

[14] Wikipedia. *Wikipedia GitHub*. URL: https://es.wikipedia.org/wiki/GitHub.

[15] npm. *Documentación npm*. URL: https://docs.npmjs.com/.

[16] bcryptjs. *Documentación bcryptjs*. URL: https://www.npmjs.com/package/bcryptjs.

[17] compression. *Documentación compression*. URL: https://www.npmjs.com/package/compression.

[18] Jared Hanson. *Documentación connect-flash*. URL: https://www.npmjs.com/package/connect-flash.

[19] connect-mongodb-session. *Documentación connect-mongodb-session*. URL: https://www.npmjs.com/package/connect-mongodb-session.

[20] csurf. *Documentación csurf*. URL: https://www.npmjs.com/package/csurf.

[21] dotenv. *Documentación dotenv*. URL: https://www.npmjs.com/package/dotenv.

[22] express-session. *Documentación express-session*. URL: https://www.npmjs.com/package/express-session.

[23] helmet. *Documentación helmet*. URL: https://www.npmjs.com/package/helmet.

[24] i18next. *Documentación i18next*. URL: https://www.i18next.com/.

[25] Mongo DB Inc. *Documentación mongoDB*. URL: http://mongodb.github.io/node-mongodb-native/.

[26] morgan. *Documentación morgan*. URL: https://www.npmjs.com/package/morgan.

[27] multer. *Documentación multer*. URL: https://www.npmjs.com/package/multer.

[28] Chai. *Documentación Chai*. URL: https://www.chaijs.com/api/.

[29] Mocha. *Documentación Mocha*. URL: https://mochajs.org/.

[30] nodemon. *Documentación nodemon*. URL: https://github.com/remy/nodemon#nodemon.

[31] Sinon. *Documentación Sinon*. URL: https://sinonjs.org/releases/v10.0.1/.

[32] MongoDB. *Documentación MongoDB Compass*. URL: https://docs.mongodb.com/compass/current/.

[33] Twilio. *Documentación SendGrid*. URL: https://sendgrid.com/docs/api-reference/.

[34] Creativity. *What is Cross-site Request Forgery (CSRF)?* URL: https://creativegroundtech.com/what-is-cross-site-request-forgery-csrf/.

Parte II

Anexos

Apéndice A

Manual de usuario

En este capítulo, se abordará la explicación de la estructura y funcionamiento de la aplicación. Para empezar se deberá acceder al sitio web en el que está desplegada la aplicación. La URL al sitio web será la siguiente https://tfgupv2021.herokuapp.com/.

Para poder comenzar a utilizar la aplicación es necesario registrarse o conectarse. En las figuras A.1 y A.2 se muestra un ejemplo de cada una respectivamente.

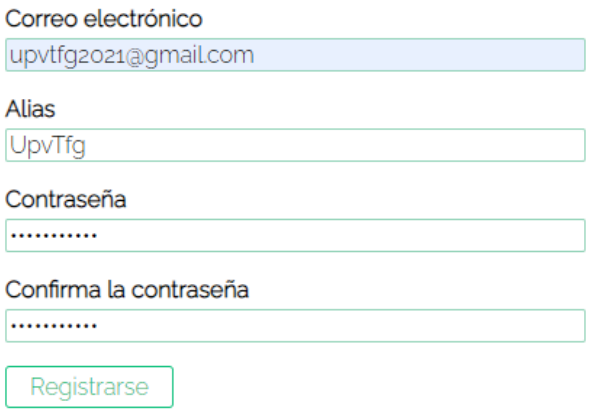

Figura A.1: Ejemplo de registro.

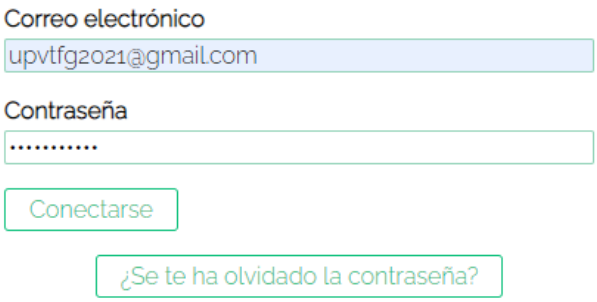

Figura A.2: Ejemplo de login.

Después de registrarse o conectarse, el usuario será redirigido al menú principal de la aplicación. Dicho menú, será igual al mostrado en la figura A.3.

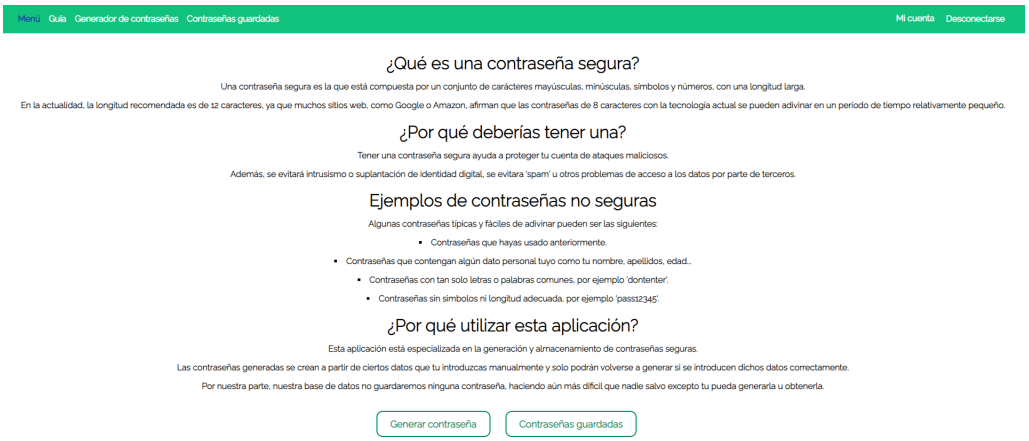

Figura A.3: Menú principal.

A continuación, se pasará a explicar apartado por apartado la barra de navegación que se muestra en la figura A.3. Se empezará por el apartado de 'Mi Cuenta'.

En el apartado de 'Mi Cuenta', se tendrá la opción de modificar ciertos datos asociados a la cuenta que se está utilizando, como el nombre, el alias, la edad o la ciudad. También, se podrá cambiar la contraseña de la cuenta. En las figuras A.4 y A.5, se muestra unos ejemplos de uso de esta página. En la figura A.4, se muestra como se vería el formulario previamente a darle al botón editar. En cambio en la figura A.5, se muestra el formulario siendo editado.

Figura A.4: Mi cuenta previamente a ser editado.

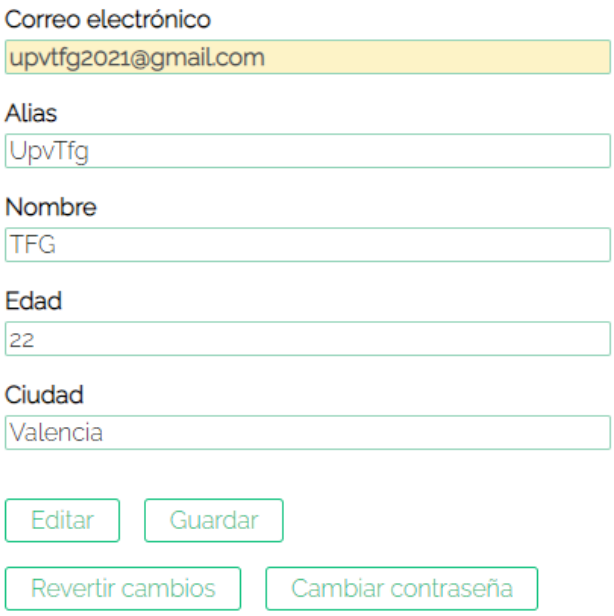

Figura A.5: Mi cuenta siendo editado.

Se continuará explicando el apartado de ’Guía’. En este apartado, simplemente se muestra una guía de uso de la aplicación. En la figura A.6, se muestra un ejemplo de este menú.

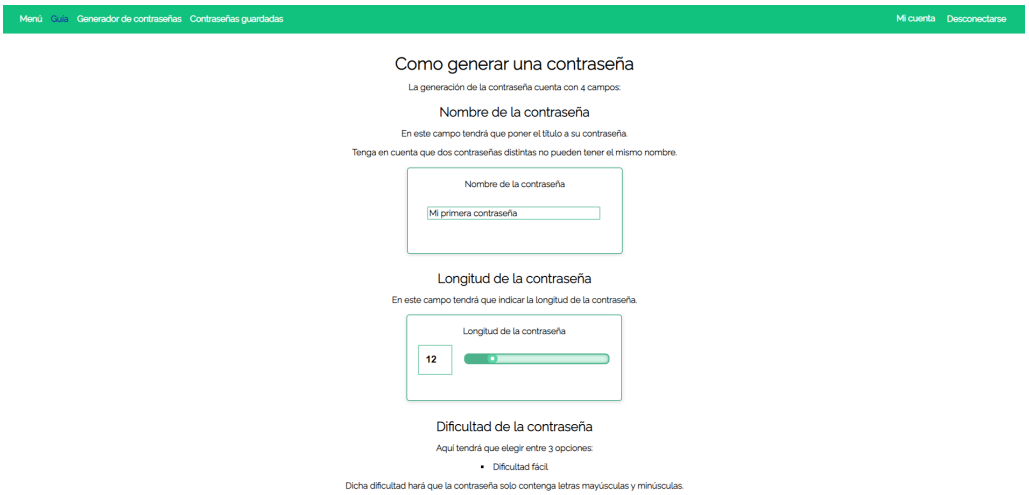

Figura A.6: Guía.

En el apartado de ’Generador de contraseñas’, se podrá generar una contraseña tras introducir ciertos datos respecto a ella. Los datos a introducir son; un nombre de contraseña, una longitud, una dificultad y hasta 3 pistas con sus respectivas respuestas. En las figuras A.7 y A.8, se muestra esta página vacía y con los datos rellenados respectivamente.

En la figura A.9, se muestra la contraseña generada a partir de la introducción de los datos de la figura A.8.

Figura A.7: Generador de contraseñas vacío.

Figura A.8: Generador de contraseñas con datos rellenados.

Por último, está el apartado de ’Contraseñas guardadas’. En este apartado, se podrá ver o eliminar las contraseñas generadas. En la figura A.10, se muestra dicha página con la contraseña generada en la figura A.9.

Además de ver o eliminar se podrá regenerar la contraseña. Para poder regenerar la contraseña es necesario darle al botón ver. Una vez realizada esta acción, se podrá ver una página similar a la de la figura A.11.

Para continuar, se procederá a introducir los datos al igual que cuando se generó por primera vez (véase figura A.12).

Una vez introducidos los datos y generada la contraseña, se mostrará una página igual a de la figura A.9 mostrando la contraseña correspondiente.

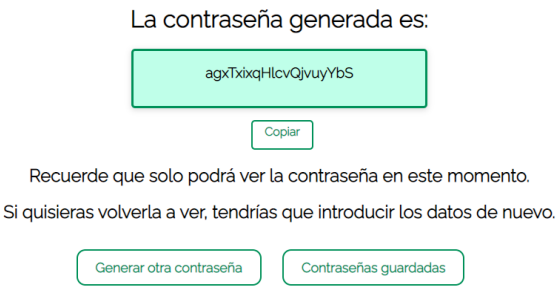

Figura A.9: Contraseña generada.

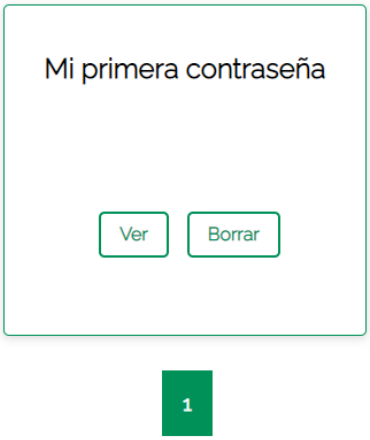

Figura A.10: Contraseña guardadas.

Figura A.11: Ver contraseña vacío.

Figura A.12: Ver contraseña con datos rellenados.