

## README.md

# Manual técnico - Aplicación de administración de la base de datos de los miembros de Ingeniería de Sistemas

## Antes de empezar

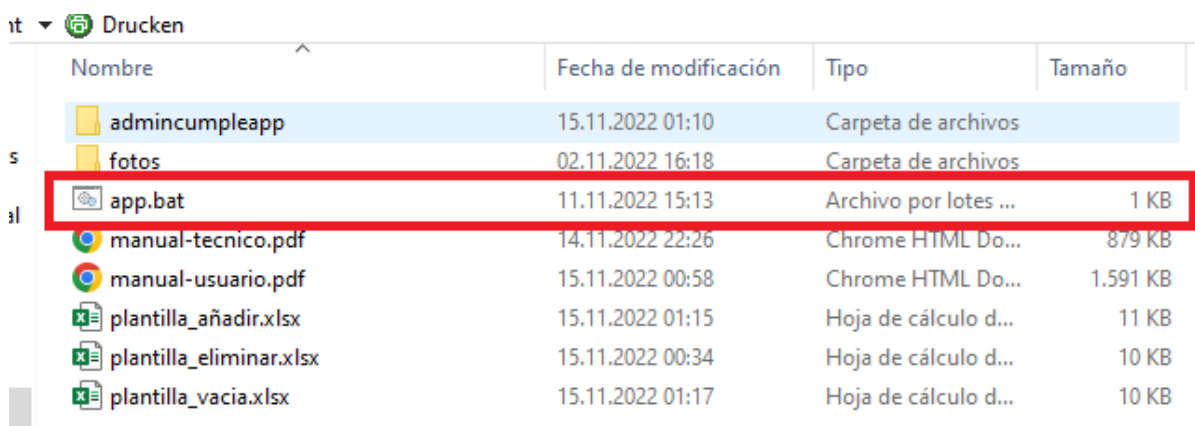
1. Por favor revise primero el [manual de usuario](#) para entender cómo se utiliza la aplicación

## Estructura de la aplicación

2. La aplicación está hecha con :
  - Java como lenguaje de programación backend
  - Spring Boot y Gradle como frameworks de desarrollo
  - HTML Y Javascript como lenguaje de programación frontend

## Ejecución de la aplicación

2. Para ejecutar la aplicación ubíquese en la raíz del proyecto y ejecute .\app en la CMD:



Nombre	Fecha de modificación	Tipo	Tamaño
admincumpleapp	15.11.2022 01:10	Carpeta de archivos	
fotos	02.11.2022 16:18	Carpeta de archivos	
app.bat	11.11.2022 15:13	Archivo por lotes ...	1 KB
manual-tecnico.pdf	14.11.2022 22:26	Chrome HTML Do...	879 KB
manual-usuario.pdf	15.11.2022 00:58	Chrome HTML Do...	1.591 KB
plantilla_añadir.xlsx	15.11.2022 01:15	Hoja de cálculo d...	11 KB
plantilla_eliminar.xlsx	15.11.2022 00:34	Hoja de cálculo d...	10 KB
plantilla_vacia.xlsx	15.11.2022 01:17	Hoja de cálculo d...	10 KB

3. Este script se ejecuta intermanente el siguiente comando : .\gradlew bootrun, el cual nos permite correr la aplicación la cual será accesible desde 127.0.0.1:8080

## Desarrollo backend

2. El backend de la aplicación está ubicado dentro de :

\src\main\java\app

Este equipo > Escritorio > proyecto > admincumpleapp > src > main > java > app					
Notas	Drucken				
	Nombre	Fecha de modificación	Tipo	Tamaño	
Contenido	controllers	09.11.2022 22:54	Carpeta de archivos		
Cloud Files	models	10.11.2022 10:28	Carpeta de archivos		
Personal	repositories	09.11.2022 22:54	Carpeta de archivos		
	services	10.11.2022 10:32	Carpeta de archivos		
o	CumpleAdministradorAppApplication.java	09.11.2022 22:54	IntelliJ IDEA	1 KB	
is					

3. Allí, encontraremos en primer lugar la clase principal :

CumpleAdministradorAppApplication.java

4. Esta es la clase principal de nuestra aplicación y se encargará de ejecutarla

```

1 package app;
2
3 import ...
4
5 @SpringBootApplication
6 public class CumpleAdministradorAppApplication {
7
8     public static void main(String[] args) { SpringApplication.run(CumpleAdministradorAppApplication.class, args); }
9
10 }

```

## Controladores

5. Ingresamos a la carpeta :

\src\main\java\app\controllers

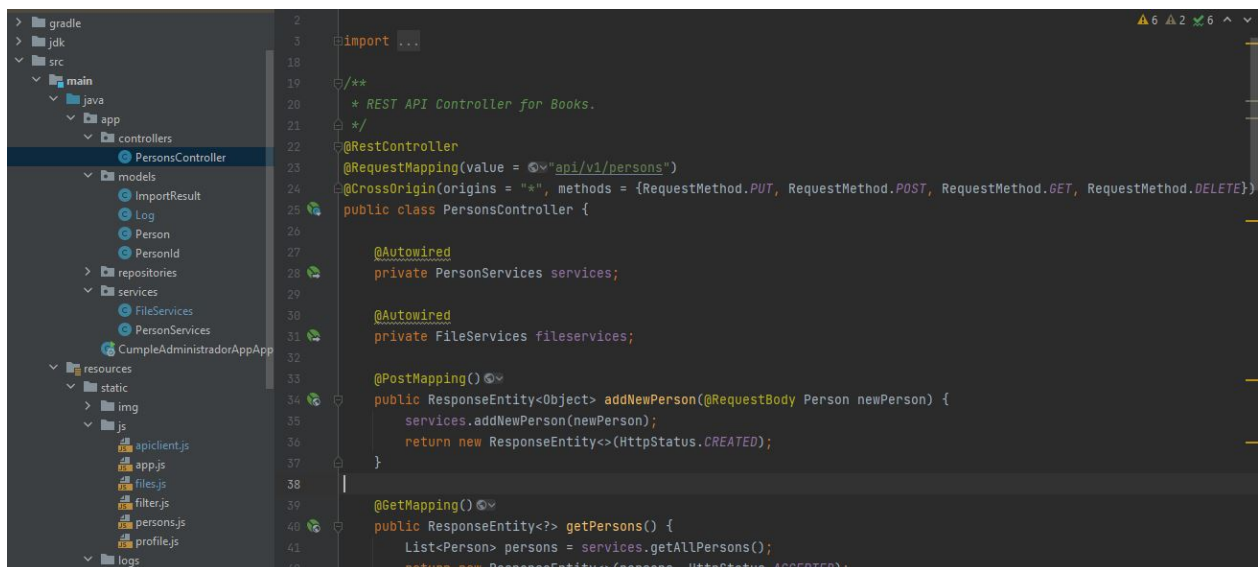
6. Allí encontramos la clase `PersonsController.java` . Esta clase se encargará de recibir las peticiones http que se hagan a la aplicación y responderlas.

7. Este controlador serán accesibles desde `127.0.0.1:8080/api/v1/persons/` en caso de querer probarlos.

8. Esta clase posee los siguientes metodos :

- `addNewPerson(@RequestBody Person newPerson)` : Se encargará de añadir una nueva persona
- `getPersons()` : Se encargará de obtener todas las personas miembros al programa de Ing. de Sistemas
- `getPersonById(@RequestBody PersonId personId)` : Se encargará de obtener una persona por su id. Su id es una llave combinada con su nombre, apellido y perfil

- contenida en una clase llamada PersonId
- deletePerson(@RequestBody Person personToDelete) : Elimina a una persona recibida como Body
- deletePersons(@RequestBody List personsToDelete) : Elimina una serie de personas recibida como Body
- putPersons(@RequestBody List persons) : Actualiza la información de una serie de personas recibida como Body
- uploadFile(@RequestParam("excelFile") MultipartFile file) : Carga la información de varias personas contenida en un archivo excel (csv,xls,xlsx). Este archivo debe ser una serie de filas con la siguiente estructura :  
carnet,nombre,apellido,dia,mes,correo,perfil,observacion,linkedin,ruta foto
- deleteFile(@RequestParam("excelFile") MultipartFile file) : Elimina la información de varias personas contenida en un archivo excel (csv,xls,xlsx). Este archivo debe ser una serie de filas con la siguiente estructura :  
carnet,nombre,apellido,dia,mes,correo,perfil,observacion,linkedin,ruta foto  
Elimina un archiv
- cleanLogs() : elimina los logs de la aplicacion



```

2  import
3
18
19  /**
20   * REST API Controller for Books.
21   */
22  @RestController
23  @RequestMapping(value = "/api/v1/persons")
24  @CrossOrigin(origins = "*", methods = {RequestMethod.PUT, RequestMethod.POST, RequestMethod.GET, RequestMethod.DELETE})
25  public class PersonsController {
26
27      @Autowired
28      private PersonServices services;
29
30      @Autowired
31      private FileServices fileservices;
32
33      @PostMapping()
34      public ResponseEntity<Object> addNewPerson(@RequestBody Person newPerson) {
35          services.addNewPerson(newPerson);
36          return new ResponseEntity<>(HttpStatus.CREATED);
37      }
38
39      @GetMapping()
40      public ResponseEntity<?> getPersons() {
41          List<Person> persons = services.getAllPersons();
42          return new ResponseEntity<>(persons, HttpStatus.ACCEPTED);
43      }
44  }

```

## Modelos

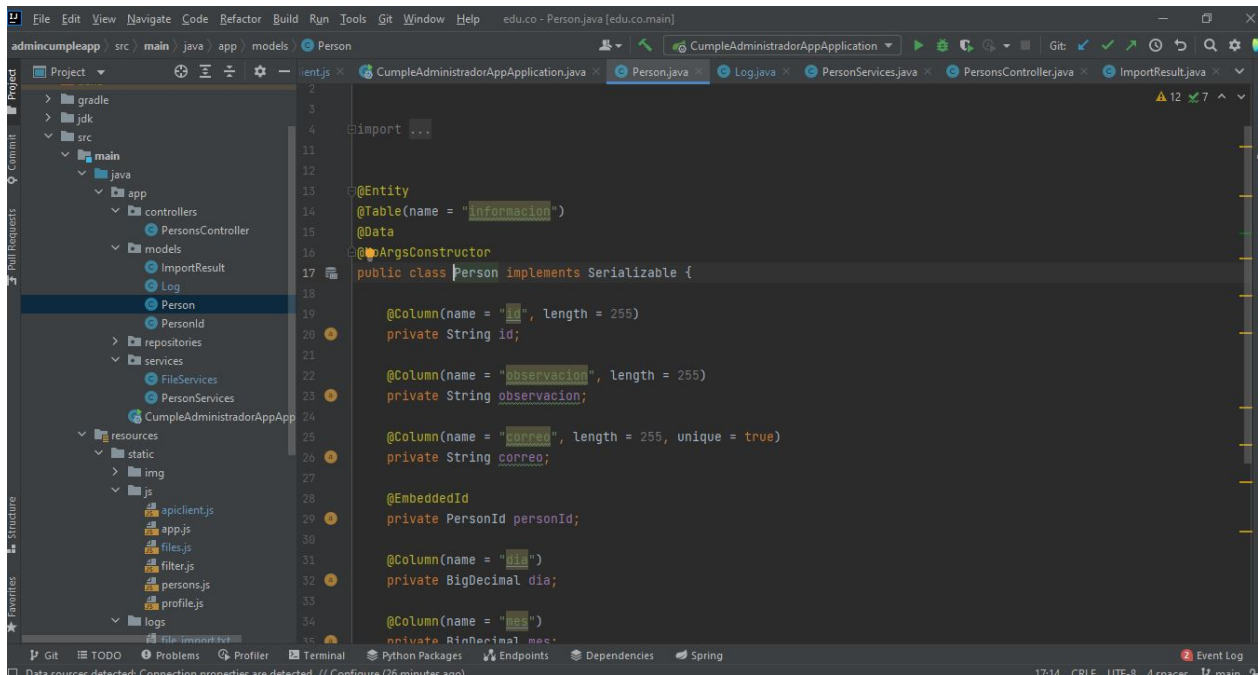
7. Ingresamos a la carpeta :

`\src\main\java\app\models`

6. Allí encontraremos los modelos de nuestra aplicación

7. Por un lado tenemos las clases `Person.java` y `PersonId.java` .

- `Person.java` representa el miembro perteneciente a la aplicación.
- Por otro lado, `PersonId.java` representa la el ID del usuario, que es una llave compuesta representada por el nombre, el perfil y el apellido de la persona.



8. Por otro lado la clase `log.java` representa los logs que son generados luego de importar algún archivo en la aplicación. Allí encontraremos los siguientes métodos :
- `writeLogs(List messages)` : Escribe varios mensajes al log
  - `setFileHandler(String fileName)` : Permite conectar el log con un archivo de texto. El nombre del archivo se recibe como parámetro y será guardado en `src/main/resources/static/logs`

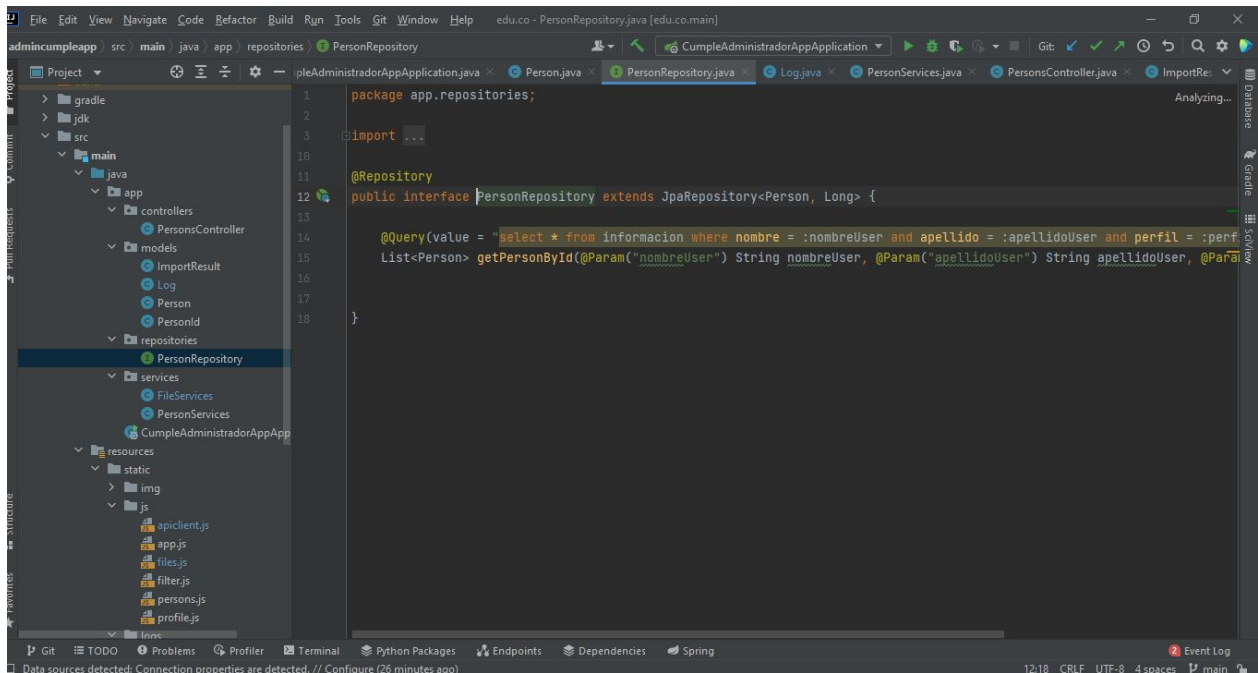


## Repositorios

9. Ingresamos a la carpeta :

```
\src\main\java\app\repositories
```

10. Dentro de esta carpeta encontramos el archivo `PersonRepository.java` . Esta clase utiliza JPA (Java Persistence API) para realizar la conexión entre nuestra aplicación y la base de datos.



## Servicios

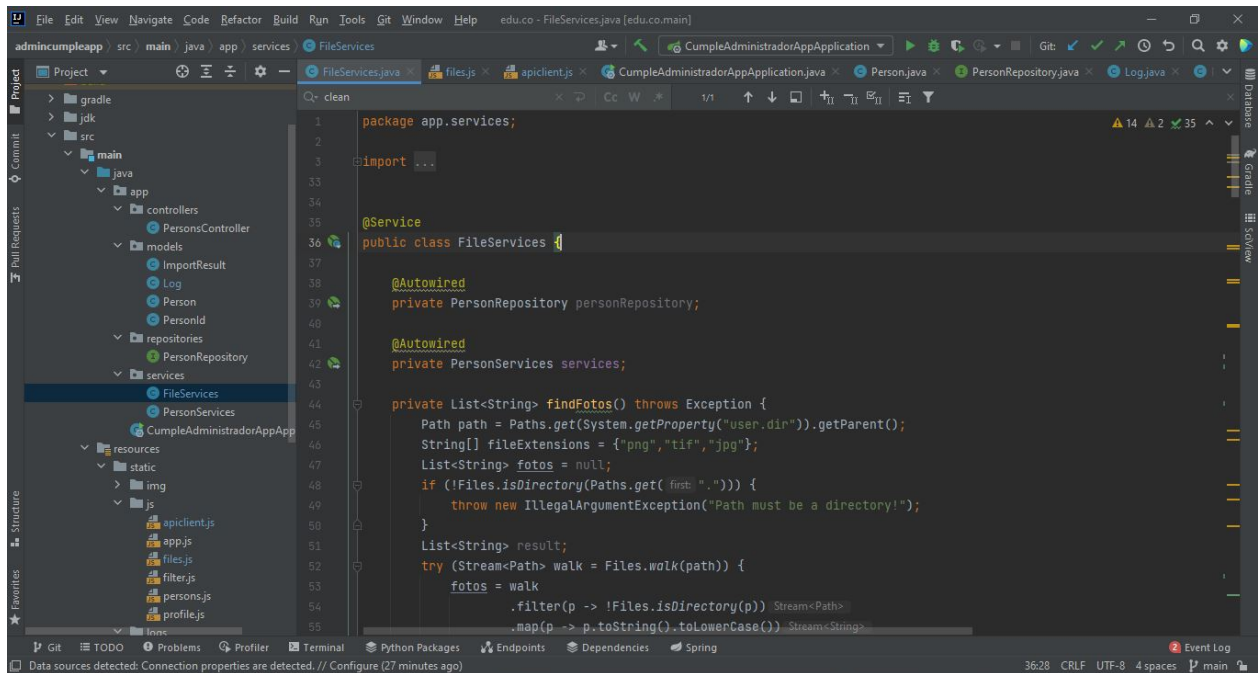
11. Ingresamos a la carpeta :

```
\src\main\java\app\
```

12. Allí encontramos dos archivos : `FileServices.java` y `PersonServices.java`

13. `FileServices.java` tiene la lógica de la importación de la información proveniente de archivos CSV, xls y xlsx. Esta clase tiene los siguientes metodos :

- `checkFileExtension(MultipartFile file)` : Verifica la extension de un archivo. Las extensiones validas son CSV,xls y xlsx
- `importPersonsFromCsv(MultipartFile file)` : Importa la información de nuevas personas proveniente de un CSV. La estructura de este archivo deben ser varias filas con la siguiente estructura : carnet,nombre,apellido,dia,mes,correo,perfil,observacion,linkedin
- `importPersonsFromExcel(MultipartFile file)` : Importa la información de nuevas personas proveniente de un Excel (xls, xlsx). La estructura de este archivo deben ser varias filas con la siguiente estructura : carnet,nombre,apellido,dia,mes,correo,perfil,observacion,linkedin
- `deletePersonsFromCsv(MultipartFile file)` : Elimina la información de nuevas personas proveniente de un CSV. La estructura de este archivo deben ser varias filas con la siguiente estructura : carnet,nombre,apellido,dia,mes,correo,perfil,observacion,linkedin
- `deletePersonsFromExcel(MultipartFile file)` : Importa la información de nuevas personas proveniente de un Excel (xls, xlsx). La estructura de este archivo deben ser varias filas con la siguiente estructura : carnet,nombre,apellido,dia,mes,correo,perfil,observacion,linkedin
- `cleanLogs()` : Elimina los logs generados de las importaciones y eliminaciones de archivos.



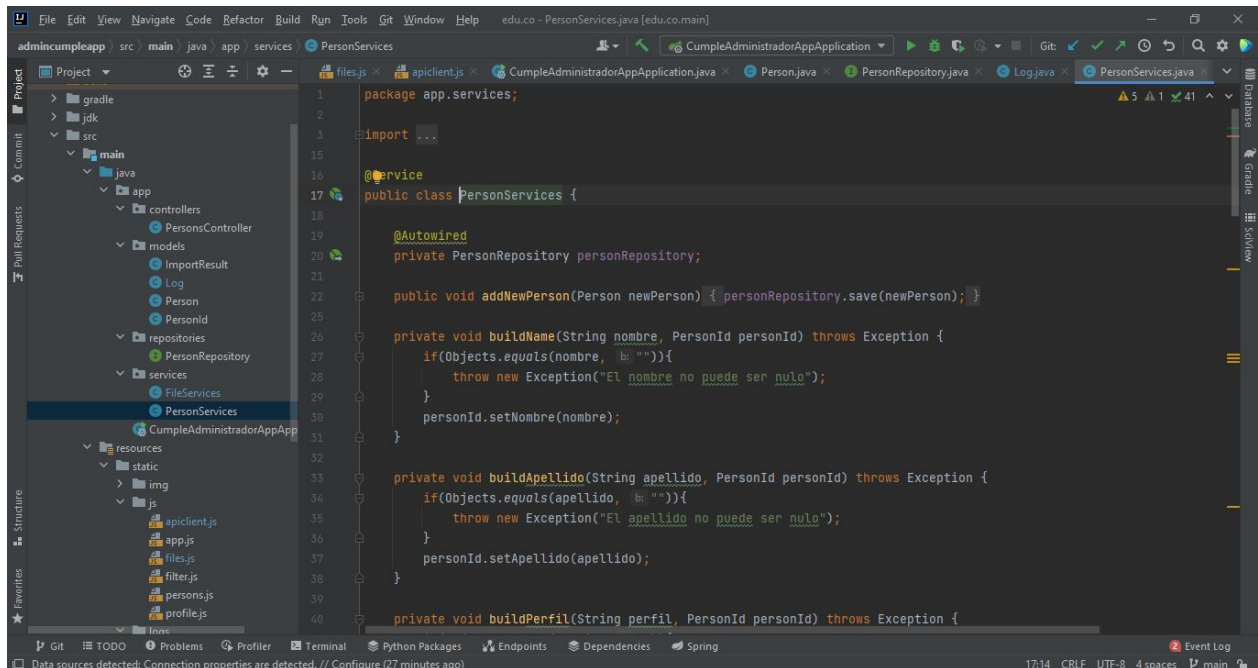
- El archivo excel a importar se ve como el siguiente

	A	B	C	D	E	F	G	H	I	J	K
1	carnet	nombre	apellido	dia	mes	correo	perfil	observacion	linkedin	foto	
2	2156625	andrés	gomez	6	10	andres@gmail.com	Estudiante	semestre: 10		fotos/foto.png	
3	2156625	Laura	Guardado	6	10	laura@gmail.com	Administrador	semestre: 11		fotos/foto.png	
4	2156625	Daniel	Ramirez	6	10	carlos@gmail.com	Estudiante	semestre: 12		fotos/foto.png	
5	2156625	Nic	Aguilera	6	10	nicolas@m	Estudiante	semestre: 13		fotos/foto.png	
6	2156625	Daniel	Contreras	6	10	daniel@gmail.com	Estudiante	semestre: 14		fotos/foto.png	
7	2156625	Felipe	Cardona	6	10	felipe@gmail.com	Graduado	semestre: 15		fotos/foto.png	
8	2156625	Gonzalo	Ortiz	6	10	gonzalo@gmail.com	Estudiante	semestre: 16		fotos/foto.png	
9	2156625	Jorge	Arango	6	10	jorge@gmail.com	Graduado	semestre: 17		fotos/foto.png	
10	2156625	Daniel	Gonzales	6	10	daniel@gmail.com	Estudiante	semestre: 18		fotos/foto.png	
11	2156625	Pablo	Mata	6	10	pablo@gmail.com	Estudiante	semestre: 19		fotos/foto.png	
12											

14. PersonServices.java tiene la lógica que construye el modelo de la Persona apartir de la información proveniente de los archivos. Su metodo más importante es:

- buildPersonfromListOfValues(List values) : Construye un objeto de tipo Persona apartir de una lista de valores . La lista estará compuesta por [carnet,nombre,apellido,dia,mes,correo,perfil,observacion,linkedin, ruta foto]





## Desarrollo FrontEnd

15. El frontend se encuentra dentro de los archivos estáticos de nuestra aplicación. Para encontrarlo, nos dirigimos a :

`\src\main\resources\static`

Este equipo > Escritorio > proyecto > admincumpleapp > src > main > resources > static

Nombre	Fecha de modificación	Tipo	Tamaño
img	09.11.2022 22:54	Carpeta de archivos	
js	10.11.2022 11:13	Carpeta de archivos	
logs	14.11.2022 19:26	Carpeta de archivos	
sweetalert2	09.11.2022 22:54	Carpeta de archivos	
index.html	10.11.2022 11:11	Chrome HTML Do...	3 KB
log.html	09.11.2022 22:54	Chrome HTML Do...	1 KB
persons.html	09.11.2022 22:54	Chrome HTML Do...	2 KB
profile.html	09.11.2022 22:54	Chrome HTML Do...	4 KB

16. Allí encontraremos cuatro archivos estáticos .html que son :

- `index.html` : representa la página principal de nuestra aplicación
- `persons.html` : representa la página de búsqueda de los miembros
- `log.html` : representa la página de búsqueda de los miembros
- `profile.html` : representa la página de edición de un miembro de nuestra aplicación

17. Además, dentro de la carpeta `js` encontraremos los archivos javascript de nuestra aplicación. Allí encontraremos lo siguiente :

- `apiclient.js` : maneja las peticiones http hacia el backend de nuestra aplicación

- `app.js` : en ella está la lógica de la funcionalidad de la página principal de la aplicación (`index.html`)
- `files.js` : representa la lógica del manejo de la carga de archivos de la aplicación. También se conecta con la página principal (`index.html`)
- `filter.js` : maneja la logica del filtro de búsqueda de los miembros del programa. Se conecta con la página de miembros del programa (`persons.html`)
- `persons.js` : maneja la logica de la búsqueda de los miembros del programa. Se conecta con la página de miembros del programa (`persons.html`)
- `profile.js` : maneja la logica de la edición de los miembros del programa. Se conecta con la página de miembros del programa (`persons.html`)

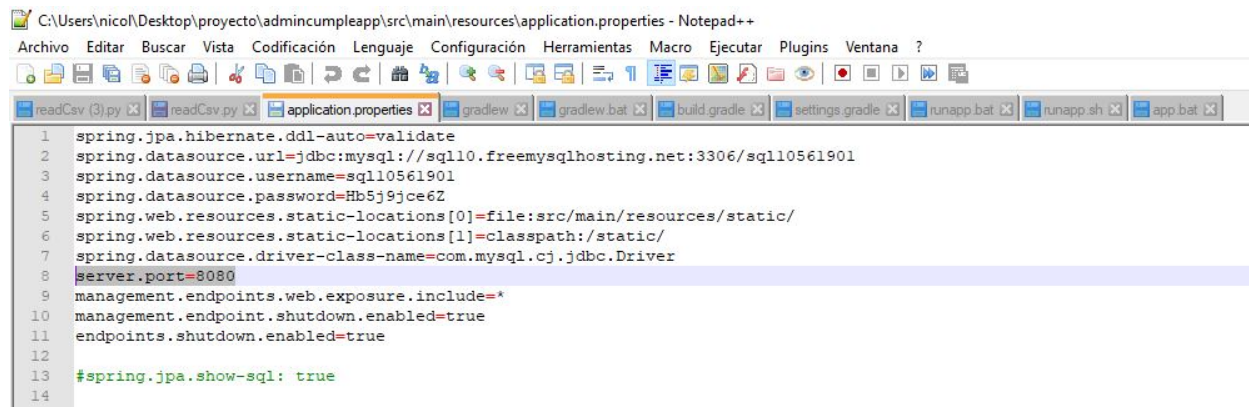
18. Dentro de la carpeta `logs` estarán ubicados los logs de la aplicación

19. Dentro de la carpeta `sweetalert2` está ubicada una libreria que se utiliza para mostrar alertas al usuario final

## Archivos de configuración

### Puerto de la aplicación

20. En el archivo `\src\main\resources\application.properties` cambiando la variable `server.port` podremos cambiar el puerto por el que corre la aplicación. Actualmente lo hace desde el puerto 8080



```

1 spring.jpa.hibernate.ddl-auto=validate
2 spring.datasource.url=jdbc:mysql://sql10.freemysqlhosting.net:3306/sql10561901
3 spring.datasource.username=sql10561901
4 spring.datasource.password=Hb5j9jce6Z
5 spring.web.resources.static-locations[0]=file:src/main/resources/static/
6 spring.web.resources.static-locations[1]=classpath:/static/
7 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
8 server.port=8080
9 management.endpoints.web.exposure.include=*
10 management.endpoint.shutdown.enabled=true
11 endpoints.shutdown.enabled=true
12
13 #spring.jpa.show-sql: true
14

```

### Credenciales de la base de datos

21. En el archivo `\src\main\resources\application.properties` está localizada la configuración de las credenciales de la base de datos. Allí tenemos la oportunidad de configurar el host, puerto, usuario y contraseña de la base de datos.

22. Existe una base de datos de prueba de la decanatura. Sus credenciales son las siguientes :

- `spring.datasource.url=jdbc:mysql://decanatura.is.escuelaing.edu.co:3306/test`
- `spring.datasource.username=fechacumple`
- `spring.datasource.password=Fecha20161Cumple`



## Librerías usadas de la aplicación

22. En el archivo `\build.gradle` están localizadas las dependencias que importan las librerías usadas por la aplicación