IES FRANCISCO DE GOYA

Gestión de Centros Culturales de la Comunidad de Madrid

Proyecto fin de Ciclo Formativo DAM

César Luis Cabrero Romero (2º DAM)

Contenido

1 Objetivos del proyecto

1.1 Introducción

El documento aquí recogido es la memoria del Proyecto de Fin de Ciclo del CFGS DAM (Desarrollo de Aplicaciones Multiplataforma)

El proyecto presentado es una aplicación web alojada en un servidor local, en la que podemos gestionar la visita de personas a los diferentes centros culturales repartidos por la comunidad de Madrid.

1.2 Objetivos Iniciales

El proyecto surge como una necesidad o idea en referencia a las visitas a las salas de estudio de los centros culturales de la comunidad de Madrid, en las cuales a día de hoy hay que recoger por escrito en papel el nombre, los apellidos y el correo electrónico de la persona que quiera entrar a estas.

Para hacer esto posible, haremos una aplicación web combinando las tecnologías de Angular 8 y Spring JPA, así como NodeJS y una base de datos PostgreSQL.

Los objetivos iniciales propuestos a realizar son:

- Aprendizaje en profundidad de tecnologías punteras y muy utilizadas en el ámbito empresarial, como pueden ser Spring JPA y Angular 8, muy importante estos últimos años en el desarrollo web.
- Aprendizaje de un nuevo gestor de bases de datos, PostgreSQL, ya que en el curso realizado solo se trabajó con MySQL y SQLite.
- Pagina web funcional y con manejo de errores, intuitiva y lista para implantar de inmediato en el caso

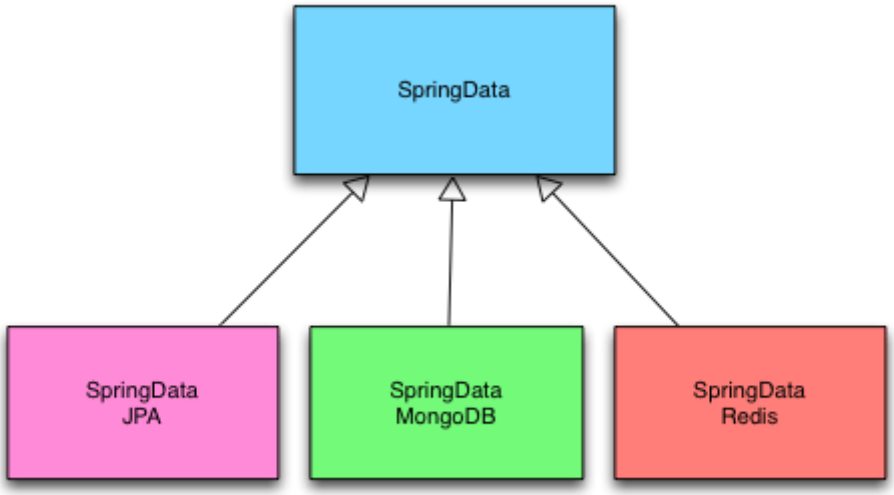
2 Tecnologías e instalación:

2.1 Información de las tecnologías usadas

Los lenguajes de programación/tecnologías usadas en la aplicación propuesta son:

2.1.1 SpringData / Spring JPA

Spring Data es uno de los frameworks que se encuentra dentro de la plataforma de Spring. Su objetivo es simplificar al desarrollador la persistencia de datos contra distintos repositorios de información.



La parte de SpringData que utilizaremos será la de Spring JPA, que sirve para realizar API de persistencia de Java. Cabe mencionar que todo esto se encuentra dentro de la vista de JAVA EE (Java Enterprise Edition).

2.1.2 NodeJS

NodeJS es un entorno en tiempo de ejecución (runtime) de JavaScript para servidor multiplataforma, de código abierto y enfocado a la capa de servidor, con una arquitectura orientada a eventos y corriendo en el motor V8 de Google. Todo el código se ejecuta en el lado del servidor. Su gestor de paquetes NPM (Node Package Manager) nos permite acceder a una enorme cantidad de librerías Open Source desarrolladas por la comunidad.

2.1.3 Angular 8

Angular es un framework de desarrollo para JavaScript creado por Google. La finalidad de Angular es facilitarnos el desarrollo de aplicaciones web SPA y además darnos herramientas para trabajar con los elementos de una web de una manera más sencilla y optima. Una aplicación web SPA creada con Angular es una web de una sola página, en la cual la navegación entre secciones y páginas de la aplicación, así como la carga de datos, se realiza de manera dinámica, casi instantánea, asincronamente haciendo llamadas al servidor (backend con un API REST) y sobre todo sin refrescar la página en ningún momento.

2.1.4 PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos relacional orientado a objetos y de código abierto. Algunas de sus principales características son, entre otras la alta concurrencia (Mientras se escribe en una tabla, otros pueden acceder a esta sin necesidad de bloqueos) y la amplia variedad de tipos nativos.

2.1.5 Apache Tomcat 9.0

Apache Tomcat (también llamado Jakarta Tomcat o simplemente Tomcat) funciona como un contenedor de servlets (El servlet es una clase en el lenguaje de programación Java, utilizada para ampliar las capacidades de un servidor). Incluye el compilador Jasper, que compila JSPs convirtiéndolas en servlets. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor web Apache.

2.1.6 Spring Tool Suite 4 (STS4)

SpringSource Tool Suite (STS) es un IDE basado en la versión Java EE de Eclipse, pero altamente customizado para trabajar con Spring Framework. Esta versión de Eclipse contiene todo lo necesario ya instalado para hacer una aplicación Spring JPA (Tomcat,Spring,Maven…)

2.1.7 Maven

Maven es una herramienta de software para la gestión y construcción de proyectos Java. Maven utiliza un Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.

2.1.8 Visual Studio Code

Visual Studio Code 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. También es personalizable, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias.

2.1.9 CSS3

CSS (Cascading Style Sheets) u "Hojas de Estilo en Cascada", es un lenguaje de diseño gráfico para la presentación visual de un documento web e interfaces como HTML.

2.1.10 BootStrap

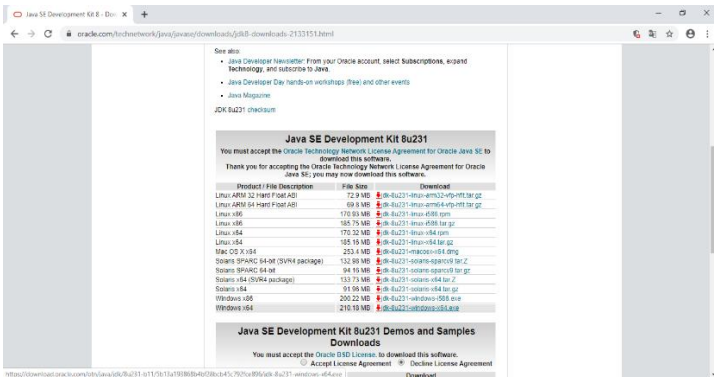
Es una biblioteca multiplataforma de código abierto para diseño de sitios y aplicaciones web. Se usa para diseño basado en HTML, CSS y JavaScript.

2.2 Instalación de cada tecnología

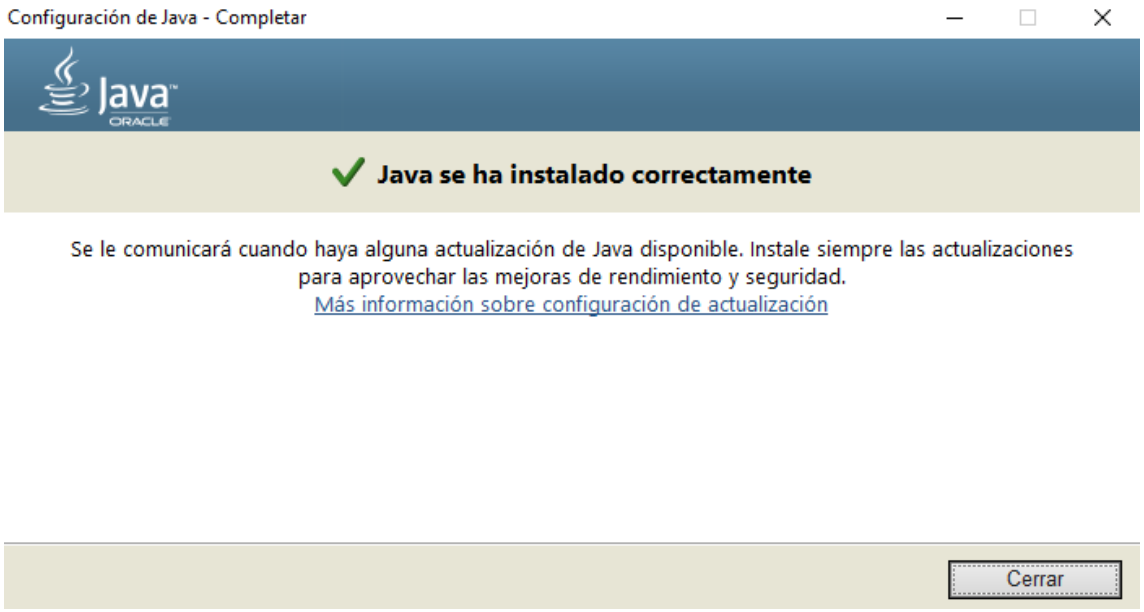
Para que este proyecto se ejecute correctamente, se necesitan varias instalaciones, tanto de entornos IDE como de librerías, lenguajes… Vamos a ver como se instalan los fundamentales:

2.2.1 Java Developement Kit 1.8

Para instalar JDK 1.8 debemos ir a la pagina de Oracle (https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html) y descargar el instalador que mas se adecue a nuestro sistema.

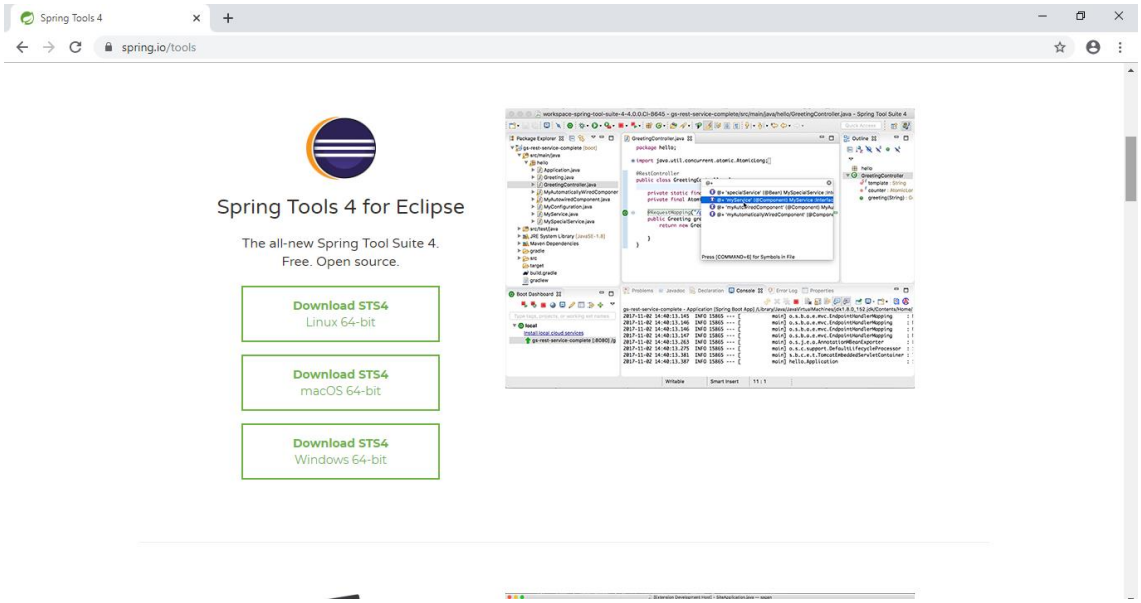


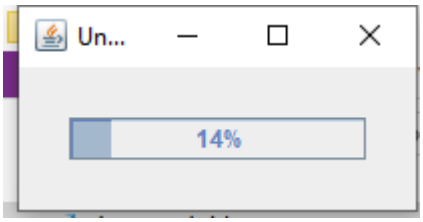
Una vez descargado instalaremos el programa sin problemas.



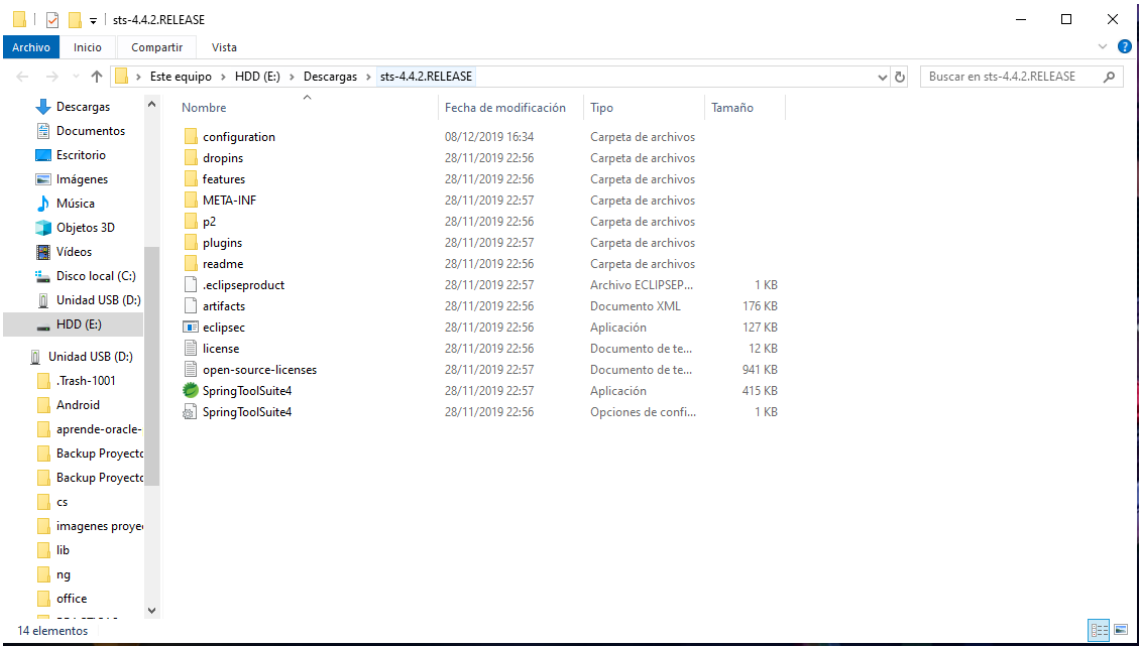
2.2.2 Spring Tools Suite 4 (STS4)

Para instalar el IDE Spring Tools Suite, deberemos irnos a su pagina de descarga (https://spring.io/tools) y descargar la versión de nuestro sistema operativo. En el caso de Windows, se nos descargará un archivo con extensión .jar, el cual deberemos abrir.

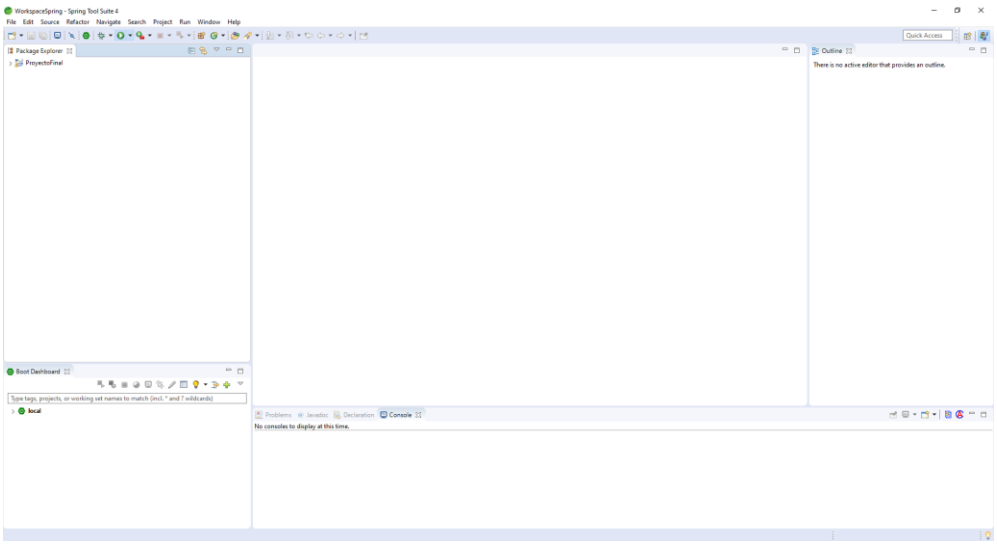




Una vez ejecutado el .jar, nos creará una capeta con todo lo necesario, ahora ya solo tendremos que ejecutarlo.



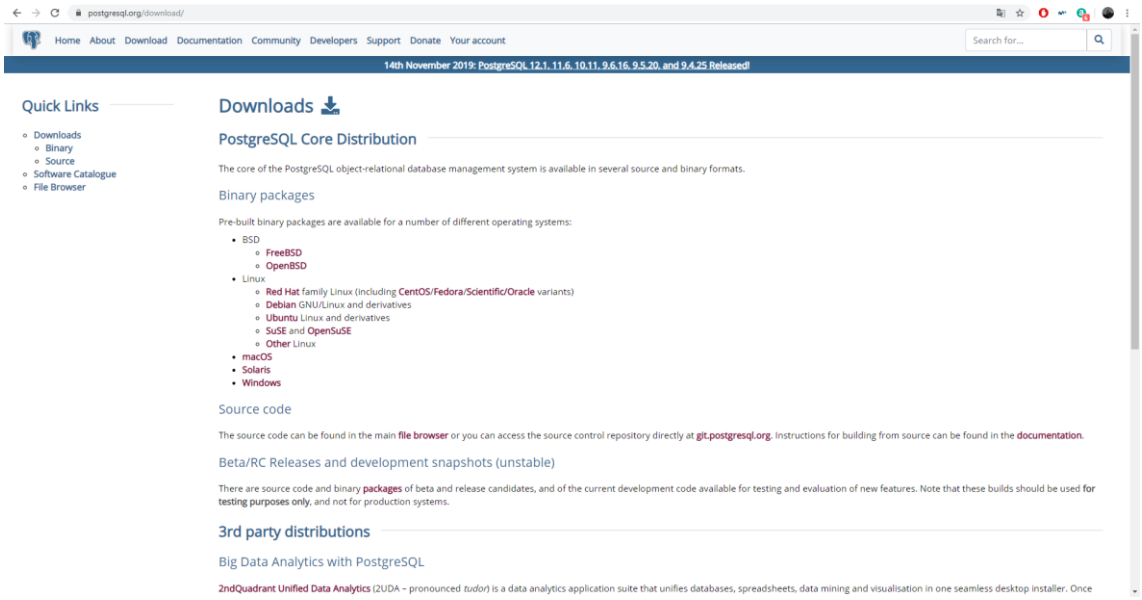
Una vez dentro del programa vemos que visualmente es similar a eclipse, y que tiene herramientas instaladas que este ultimo no tiene, como el boot dashboard.



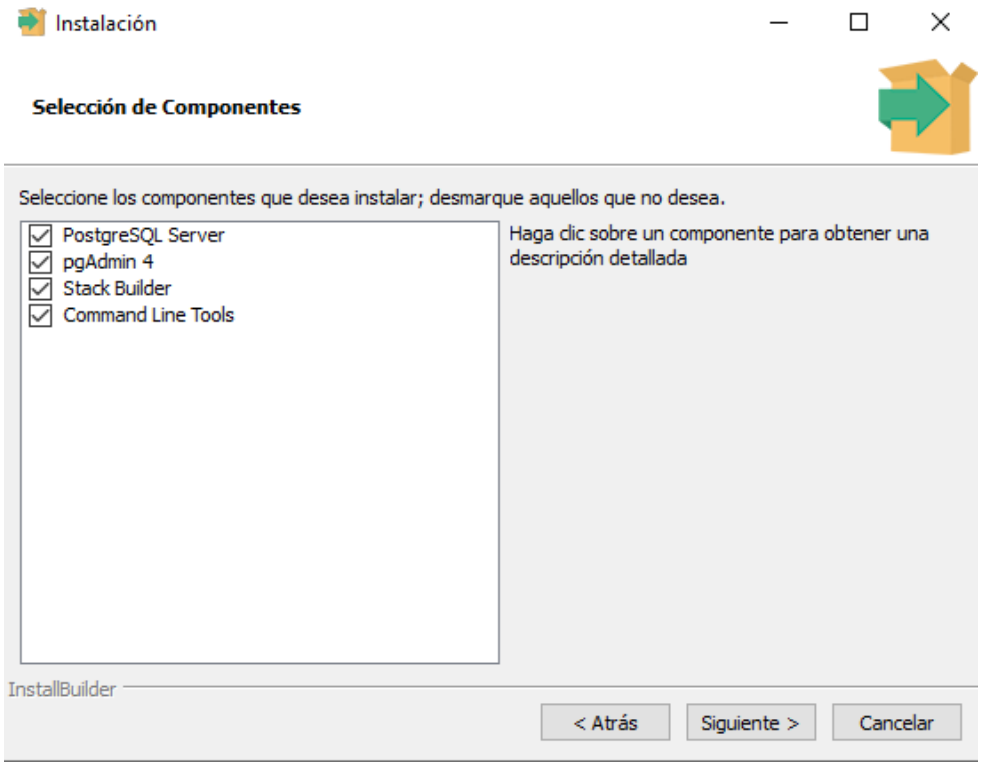
A partir de aquí ya podremos crear cualquier proyecto de Spring, o importar uno nuevo.

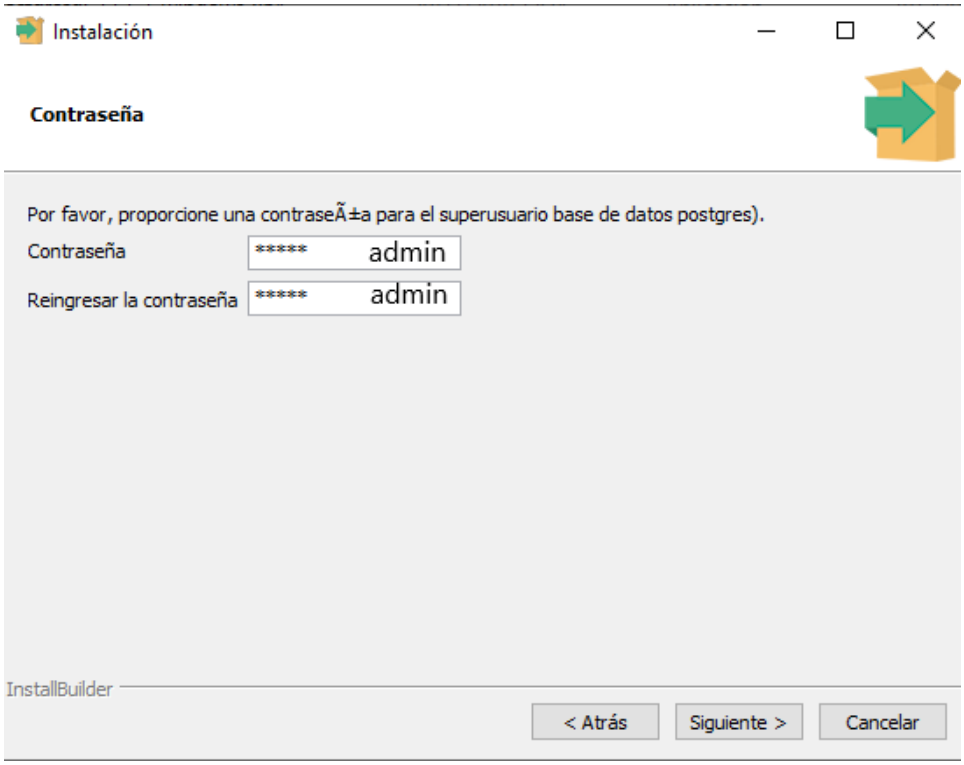
2.2.3 PostgreSQL

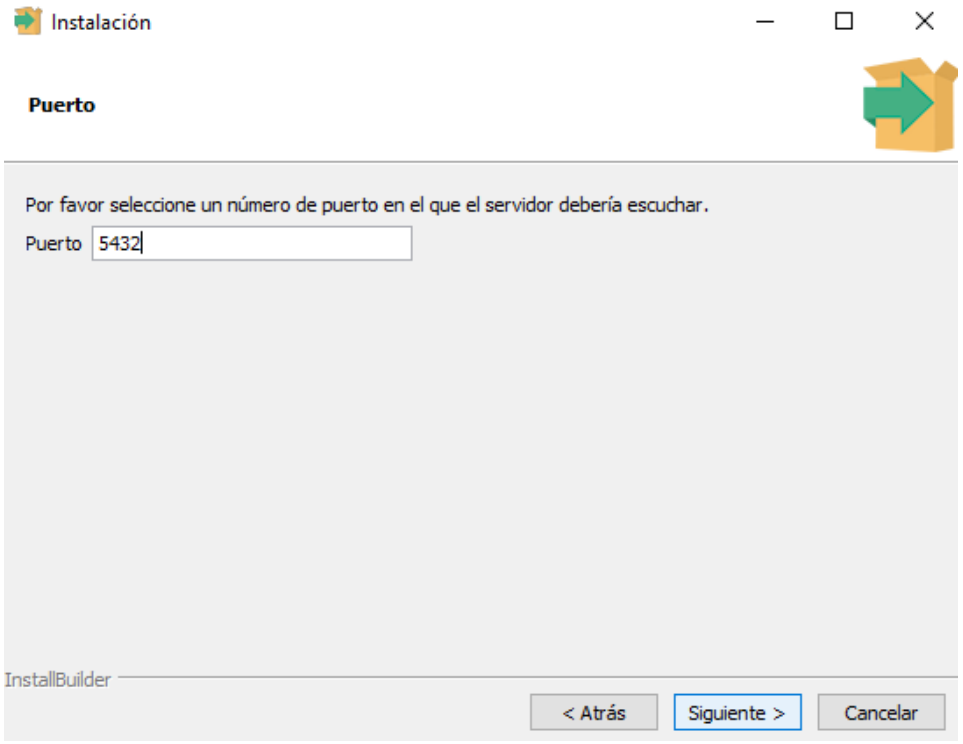
Para realizar la instalación de PostgreSQL en nuestra maquina, primero deberemos descargar la herramienta que mejor se adecue a nuestro Sistema operativo de su pagina web (https://www.postgresql.org/download/)



Para la instalación deberemos ejecutar el archivo que hemos descargado y darle a continuar hasta la siguientes imágenes,que deberemos marcar lo que indica en ellas:

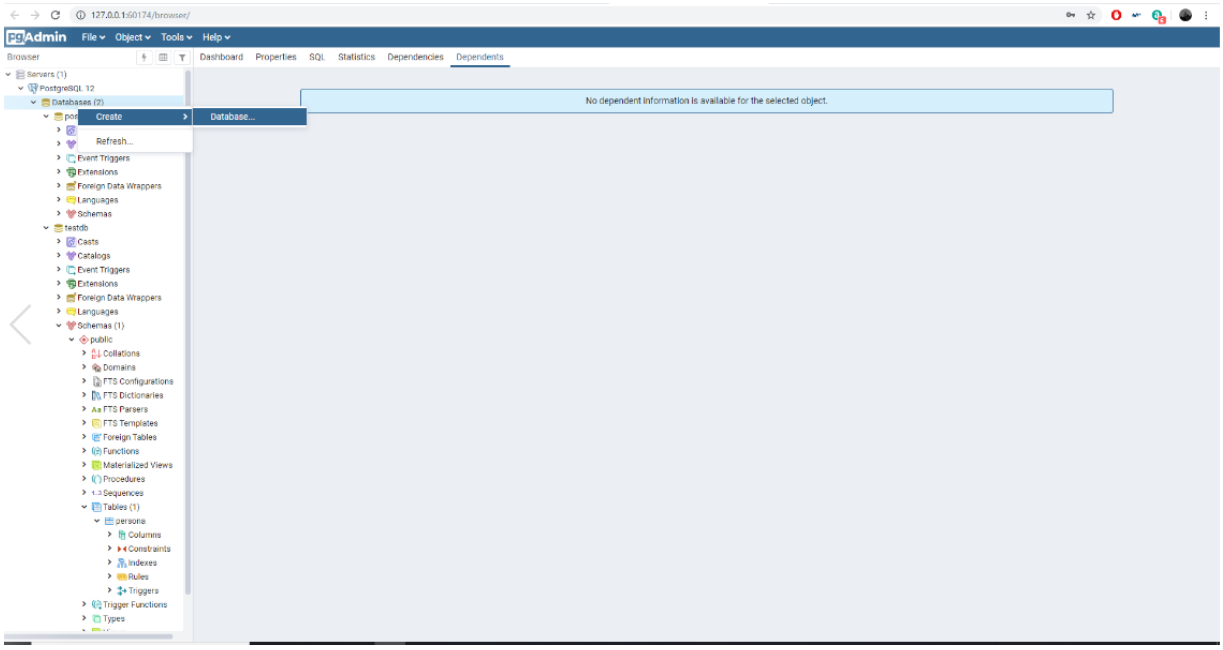


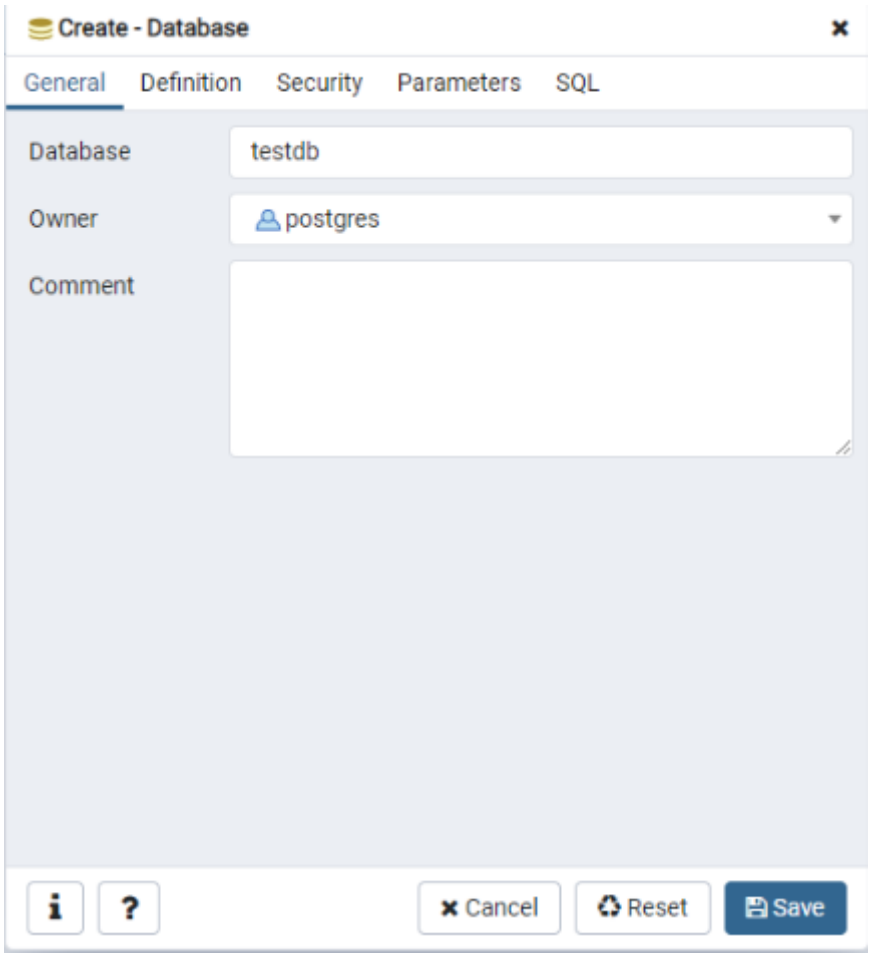




Y ya tendríamos listo para usar nuestro postgre desde la aplicación PGAdmin4.

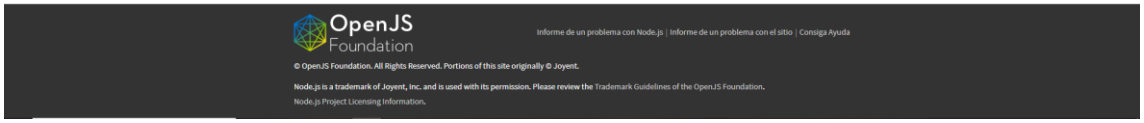
Para nuestro proyecto, deberemos ejecutar PGAdmin 4 e iniciar sesión con la contraseña que hemos puesto anteriormente (admin). Una vez dentro,deberemos crear una base de datos con el click derecho que se llame testdb (fundamental para que funcione)


pgAdmin
File
Object
Tools
Help
Browser
Dashboard
Properties
SQL
Statistics
Dependencies
Dependents
Servers (1)
PostgreSQL 12
Databases (2)
Create
Database...
Refresh...
No dependent information is available for the selected object.
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Schemas
testdb
Casts
Catalogs
Schemas (1)
public
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Procedures
Sequences
Tables (1)
persona
Columns
Constraints
Indexes
Rules
Triggers
Trigger Functions
Types


Create - Database
General
Definition
Security
Parameters
SQL
Database
testdb
Owner
postgres
Comment
Cancel
Reset
Save

2.2.4 NodeJS

Para instalar NodeJS en nuestra maquina (Indispensable para instalar paquetes NPM) debemos ir a su pagina web(https://nodejs.org/es/) y descargar la ultima versión estable (LTS) y descargar el instalador



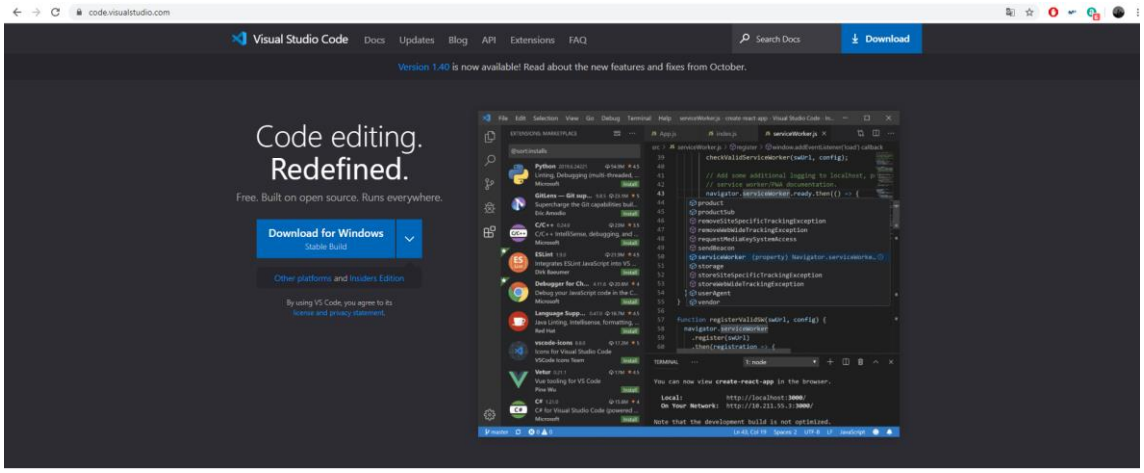
Una vez descargado, ejecutamos el instalador y clicamos en siguiente todo lo que nos dice hasta que se instale (por defecto).

Para saber si se ha instalado correctamente, nos vamos a Windows Powershell y ponemos el comando **npm version.** Si nos detecta una version esta instalado correctamente.

```
PS C:\Users\saluc> npm version
{
  npm: '6.12.1',
  ares: '1.15.0',
  brotli: '1.0.7',
  cldr: '35.1',
  http_parser: '2.8.0',
  icu: '64.2',
  llhttp: '1.1.4',
  modules: '72',
  napi: '5',
  nghttp2: '1.39.2',
  node: '12.13.1',
  openssl: '1.1.1d',
  tz: '2019c',
  unicode: '12.1',
  uv: '1.33.1',
  v8: '7.7.299.13-node.16',
  zlib: '1.2.11'
}
```

2.2.5 Visual Studio Code

Una vez instalado NodeJs, ya podremos instalar nuestra version de Visual Studio Code y nuestra librería de angular. Para ello necesitaremos primero de todo bajar la ultima version de su pagina web (https://code.visualstudio.com/)



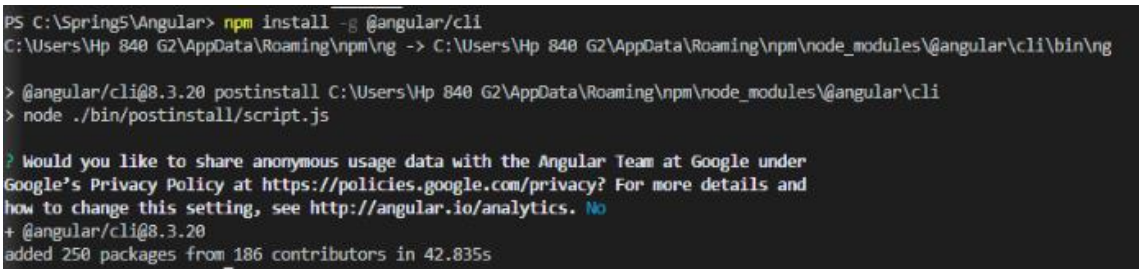
Una vez descargado, ejecutamos el instalador e instalamos con las opciones por defecto.

Una vez finalizada la instalación, ejecutamos el programa para configurar un par de parámetros y librerias:

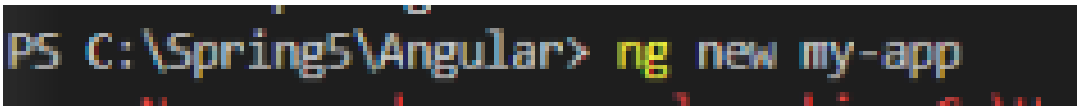
1. Por defecto en Windows, la ejecución de scripts esta deshabilitada,asique abriendo la consola de Visual Studio Code (CTRL + Ñ) nos abrirá una consola directa desde Windows PowerShell, donde deberemos añadir este comando para que no nos de errores:

2. Desde esta misma consola, debemos instalar las librerias de @Angular/cli con el siguiente comando:

Y se nos descargaran las librerias necesarias:



3. Para crear un nuevo proyecto de angular, basta con ubicarnos en la carpeta donde queramos crearlo (En mi caso C:/Spring5/Angular/) y ejecutar el siguiente comando:



En my-app iría el nombre del proyecto que le queramos dar (en este caso my-app)

2.3 Desplegando el proyecto

Para desplegar nuestro proyecto, tendremos dos carpetas comprimidas (persona-app para visual studio code y ProyectoFinal para Spring Tools Suite).

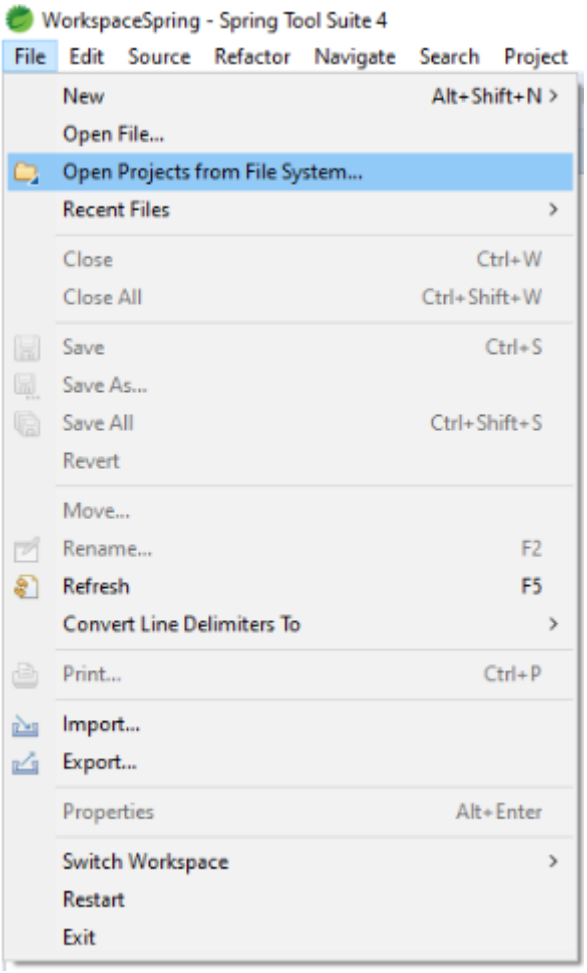
Vamos a ver como podemos desplegar el proyecto.

2.3.1 Desplegando en Spring Tools Suite

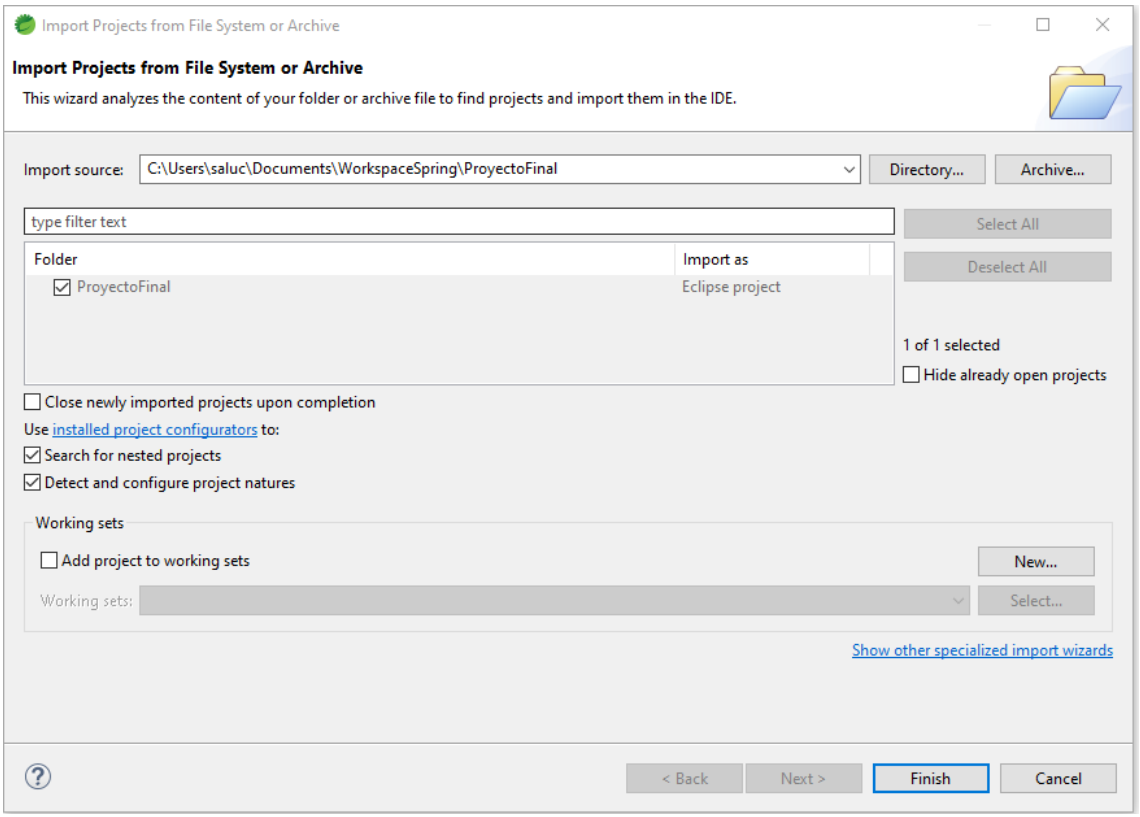
Para desplegar el proyecto en Spring Tools Suite, deberemos extraer la carpeta ProyectoFinal a nuestro workspace (lo tenemos que elegir en la instalación o al ejecutar el programa).

Una vez extraído y ejecutado el programa Spring Tools Suite, deberemos realizar los siguientes pasos:

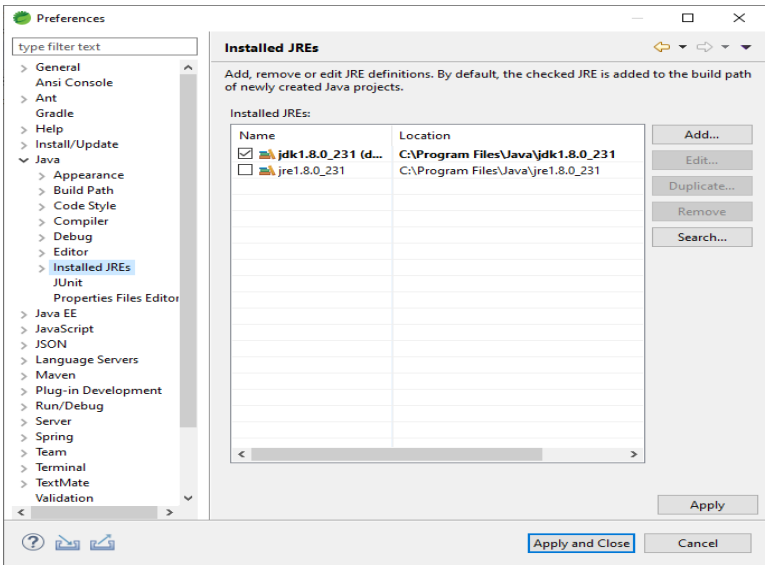
1. Debemos irnos en el menú a File y clicar en open projects from Filesystem

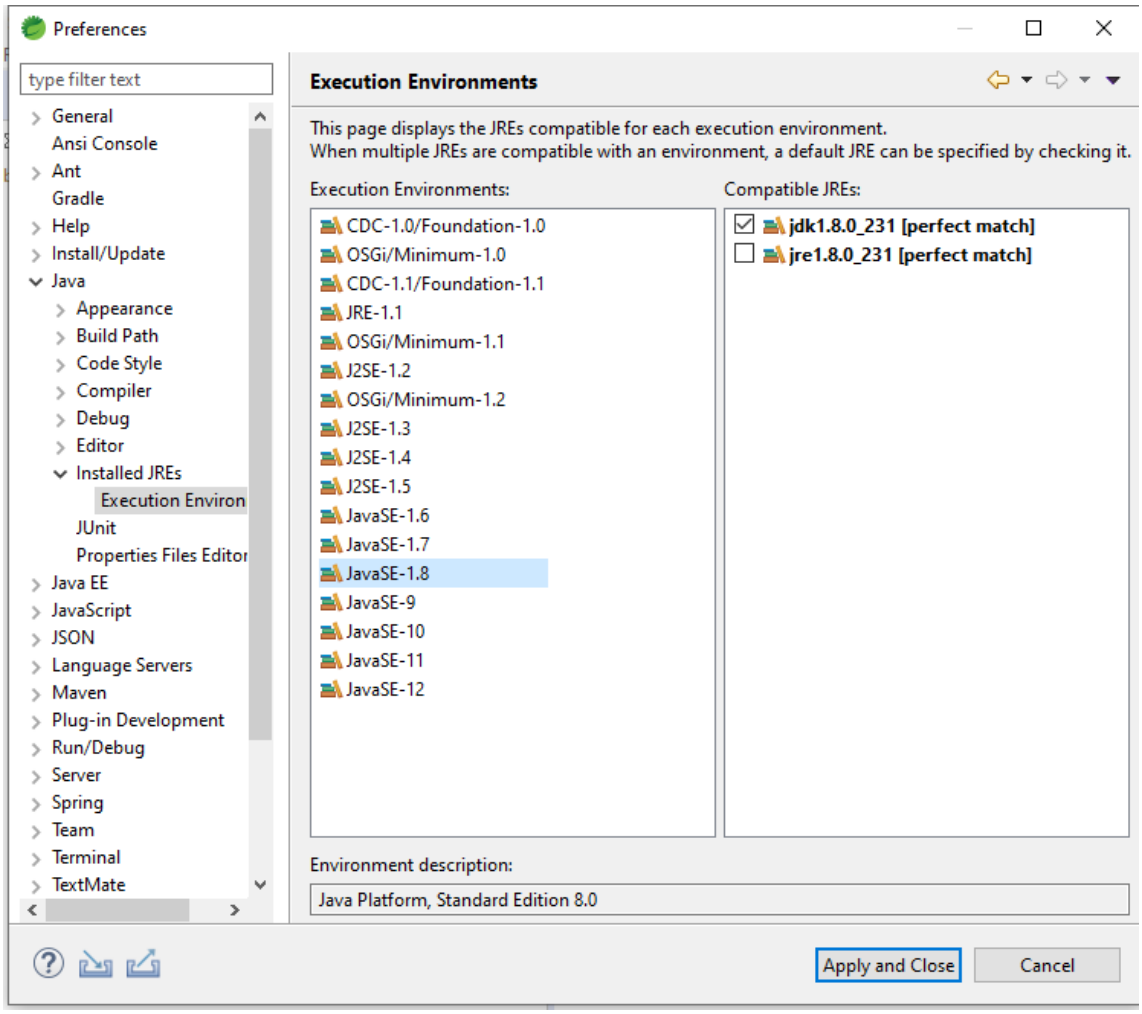


2. Una vez clicado, en import Source deberemos darle al botón de directory y seleccionar la carpeta extraida, para que nos quede asi:



3. Una vez importado, damos click en finalizar y nos lo importara automáticamente, actualizándonos el workspace. Una vez realizado y de o no fallos, debemos irnos a window>preferences, y debemos cambiar en JAVA>Installed JREs para dejarlo asi:





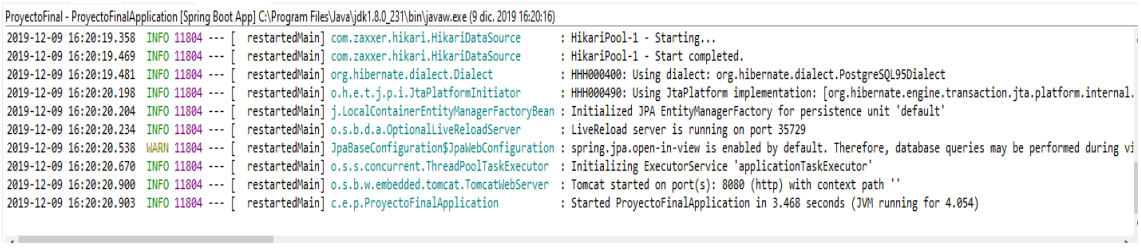
Si no hacemos esto, nos puede dar un error en el paso siguiente en el cual el mensaje será:

```
[ERROR] No compiler is provided in this environment. Perhaps you are running on a JRE rather than a JDK?
```

4. Una vez configurado todo, deberemos darle click derecho en el proyecto y donde pone Run as… debemos darle a Maven Install (opción 7). Una vez haga build success ya podemos ejecutarlo.

```
<terminated> C:\Program Files\Java\jdk1.8.0_231\bin\javaw.exe (9 dic. 2019 16:16:11)
[INFO]
[INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ EjemploPostgress ---
[INFO] Deleting C:\Users\saluc\Documents\WorkspaceSpring\ProyectoFinal\target
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.701 s
[INFO] Finished at: 2019-12-09T16:16:13+01:00
[INFO] ------------------------------------------------------------------------
[WARNING] The requested profile "pom.xml" could not be activated because it does not exist.
```

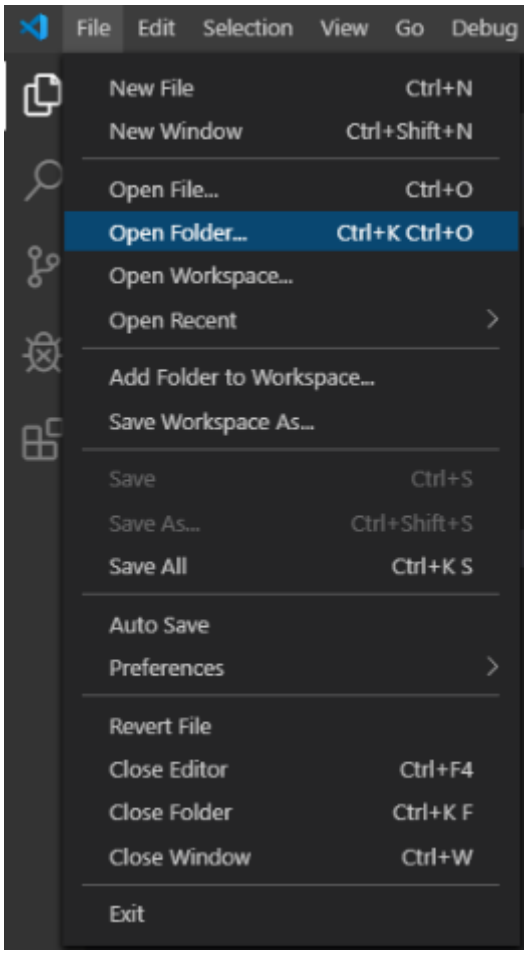
5. Para ejecutarlo basta con darle click derecho, Run as… y selecionamos la opción Spring Boot App hasta que nos aparezca lo siguiente:

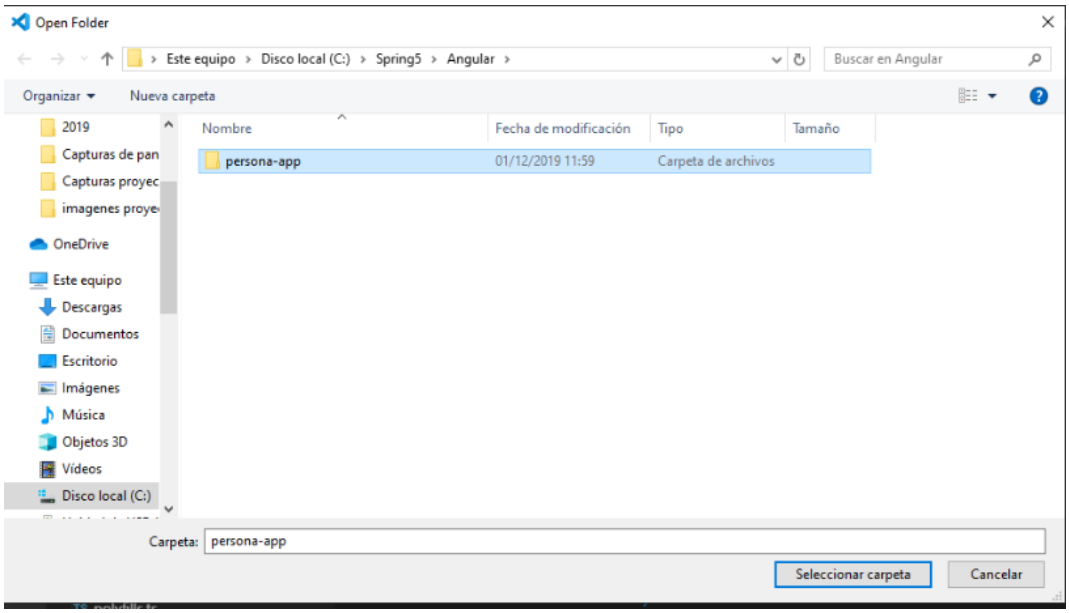


Y ya estaría listo para dejarlo ahí en segundo plano.

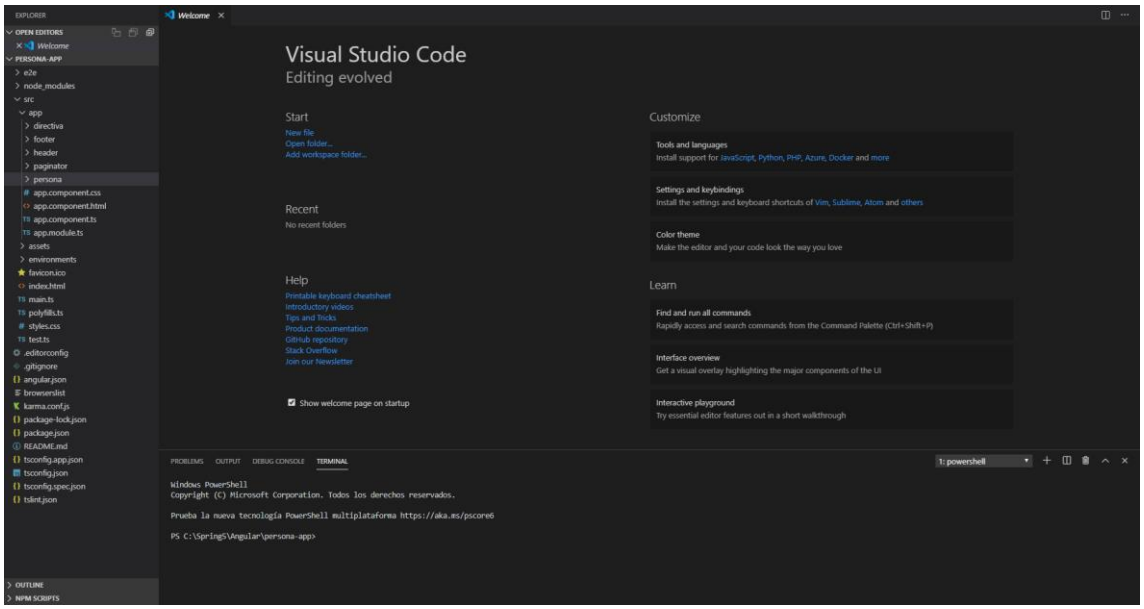
2.3.2 Desplegando en Visual Studio Code

Para desplegar en este IDE es mucho mas fácil que en el caso anterior. Basta con descomprimir la carpeta persona-app en la ubicación que queramos y abrirla ejecutando Visual Studio Code y pinchando en File>Open folder… y seleccionamos la carpeta.

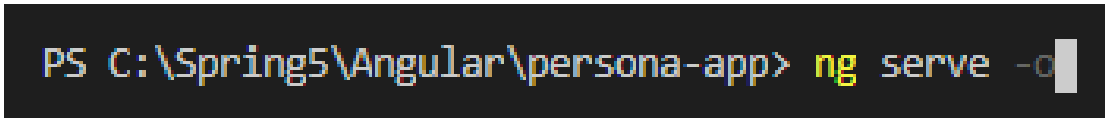


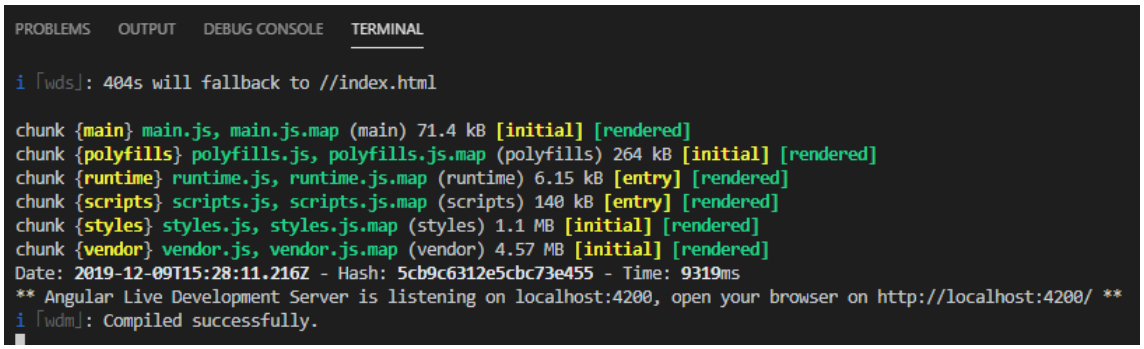


Una vez abierto, Visual Studio Code nos pondrá los archivos de ese proyecto a la izquierda y nos abrirá una consola directamente sobre ese proyecto.



Para iniciarlo ya solo basta con ejecutar el comando ng serve -o en la consola y este nos abrirá la web diseñada con angular, y al tener ejecutado también el proyecto de spring lo unirá.





Y listo,ya tendremos desplegado todo nuestro proyecto.

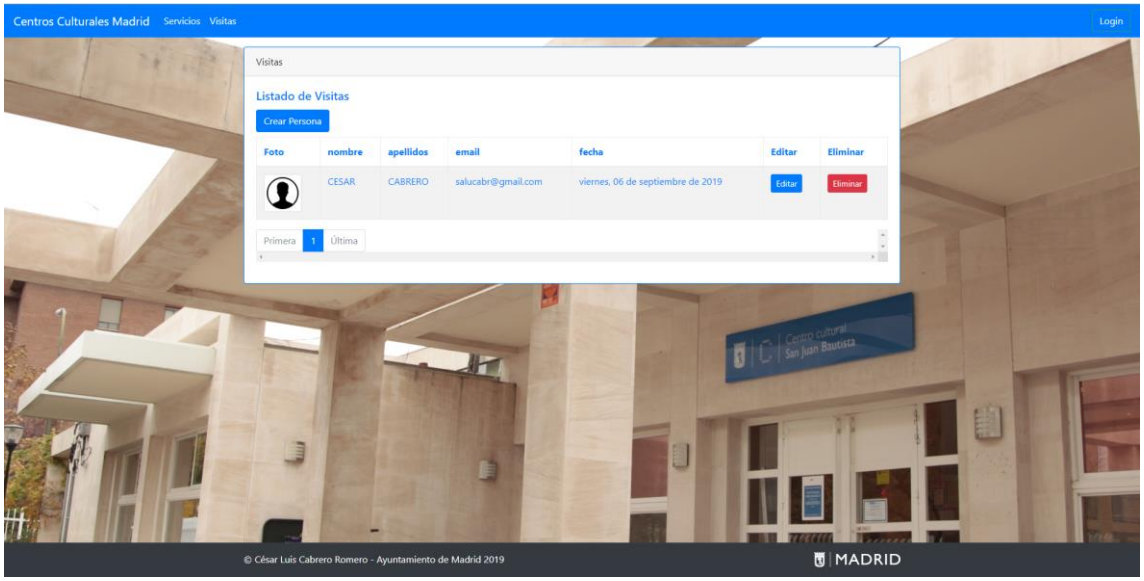
3 Análisis de Requisitos

El proyecto tiene como fin gestionar las visitas de personas a un centro cultural, asi como informar de sus actividades, asique lo plantearemos de la siguiente forma:

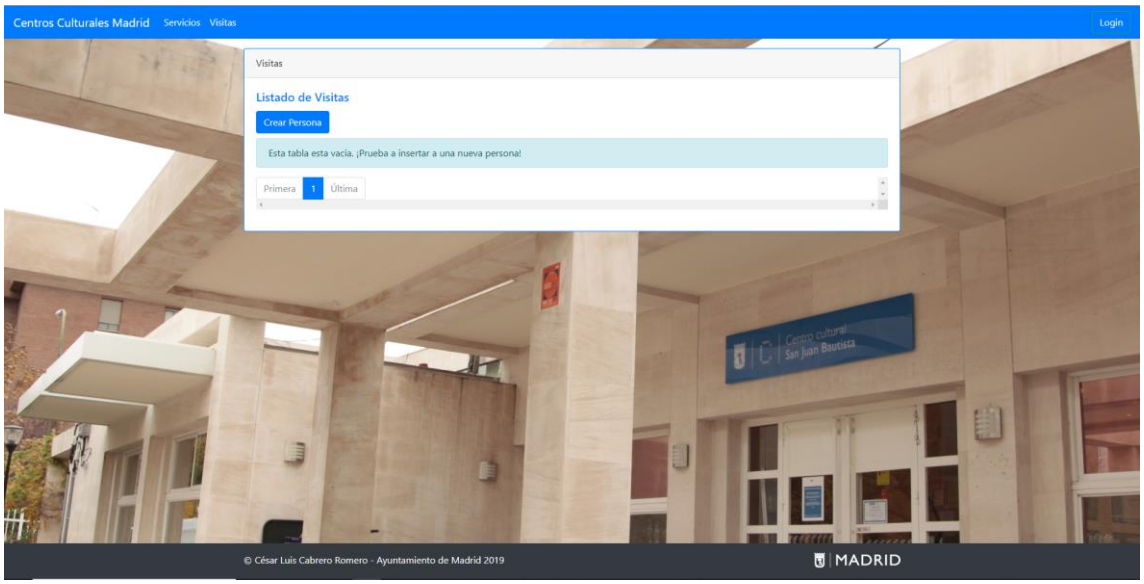
- La pagina al abrir el proyecto será directamente la tabla donde tendremos todos los datos de personas almacenados, además de un boton Crear Persona en donde podamos insertar personas con nombre, apellidos, email, fecha de creación del registro y una foto, así como un botón de eliminar o editar.
- Una pagina llamada servicios, donde haya un botón que despliegue una lista con los servicios del centro.
- Una pagina de registro al presionar el botón Crear Persona, que nos lleve a otra pagina diferente para la creación, donde tendremos manejados todos los errores, por ejemplo si insertamos un nombre en blanco se nos requerirá rellenar ese campo, o si metemos un email que no tenga formato email se nos avisará.
- Una pagina de subida de fotos al clicar en la foto, donde podamos ver la miniatura de la foto o los datos de la persona, con una barra de progreso para la subida. Todo esto se hará en un modal de Bootstrap.

Veamos mas en profundidad las paginas:

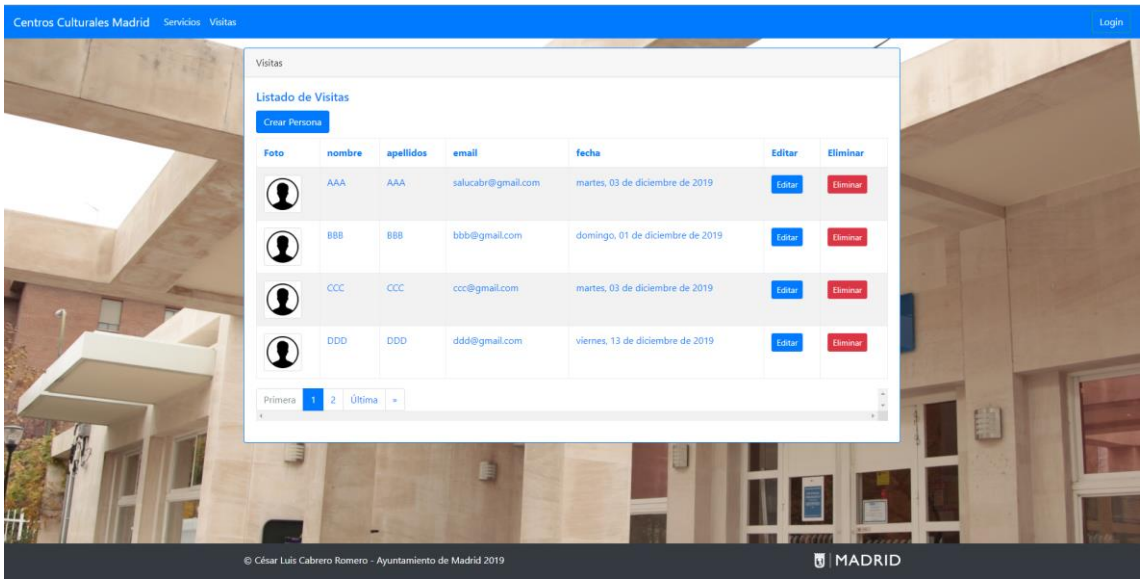
3.1 Pagina inicio



Esto es lo que aparecerá si tenemos al menos una persona dada de alta.



Si no existe ninguna persona en la base da datos, se nos mostrará un cartel de que no hay nadie registrado, y se nos incitará a dar de alta a alguien.

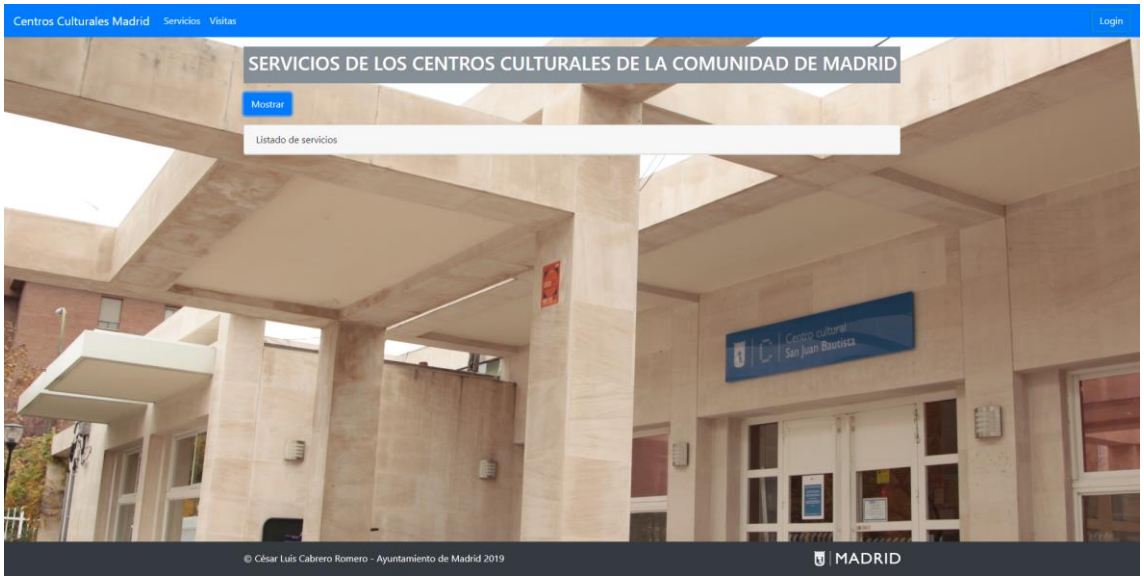


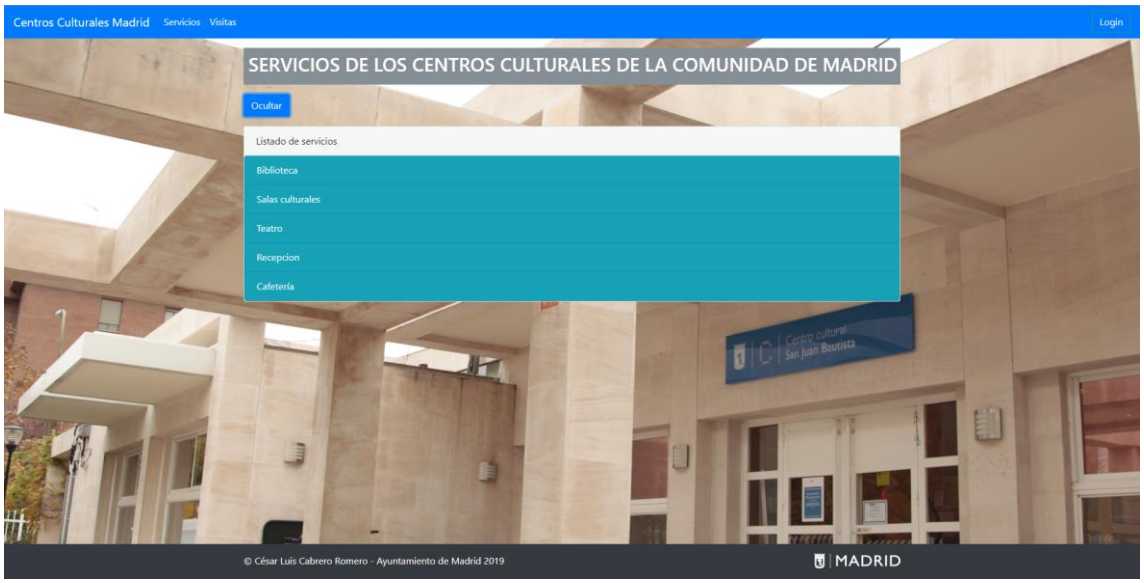
Si hay mas de 4 personas, la página se paginará, dándonos opción a la pagina que deseemos visitar con números paginados, además de un link para pasar directamente de la primera a la ultima página y viceversa.

En el footer, si pinchamos en el icono de Madrid, nos redirigirá a la pagina real del ayuntamiento.

3.2 Pagina Servicios

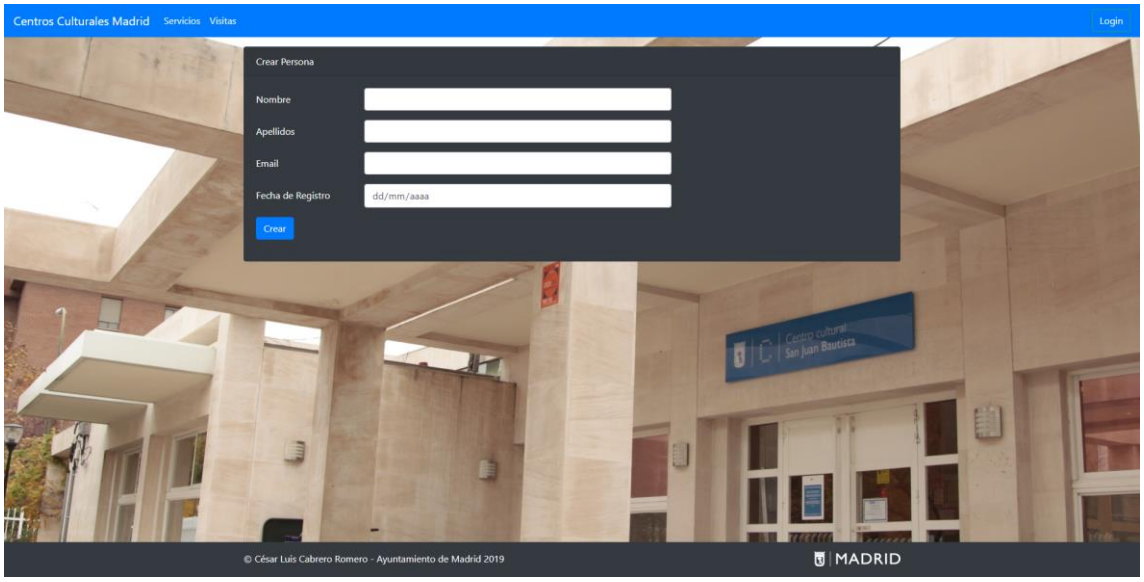
Cuando Pinchemos en la pagina Servicios, esta nos mostrará una pagina sencilla con un titulo y un botón que mostrará y ocultará la lista de servicios disponibles en ese centro.



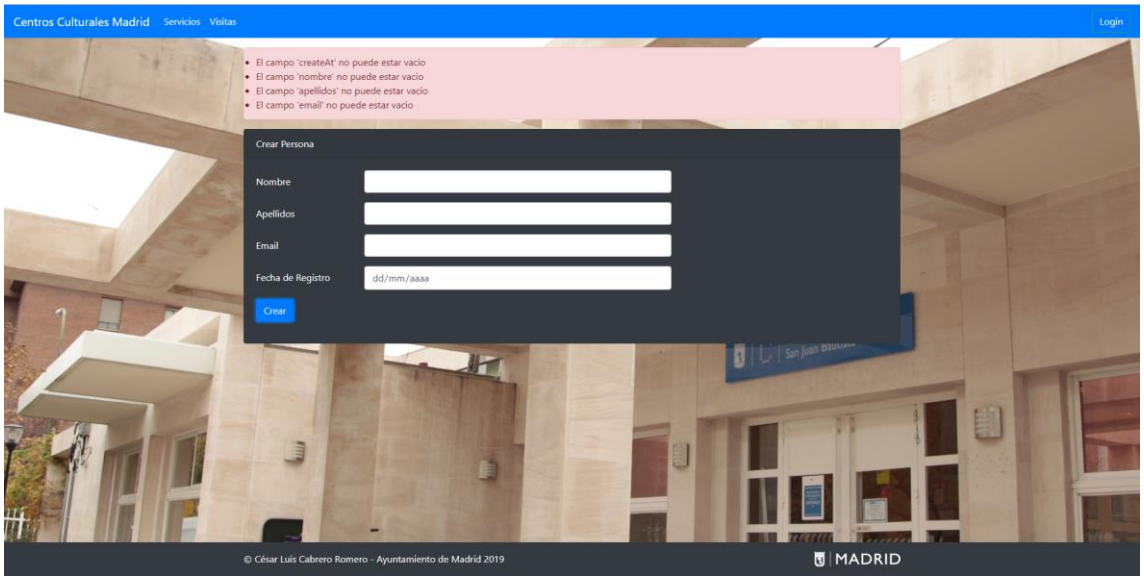


3.3 Pagina Crear Persona

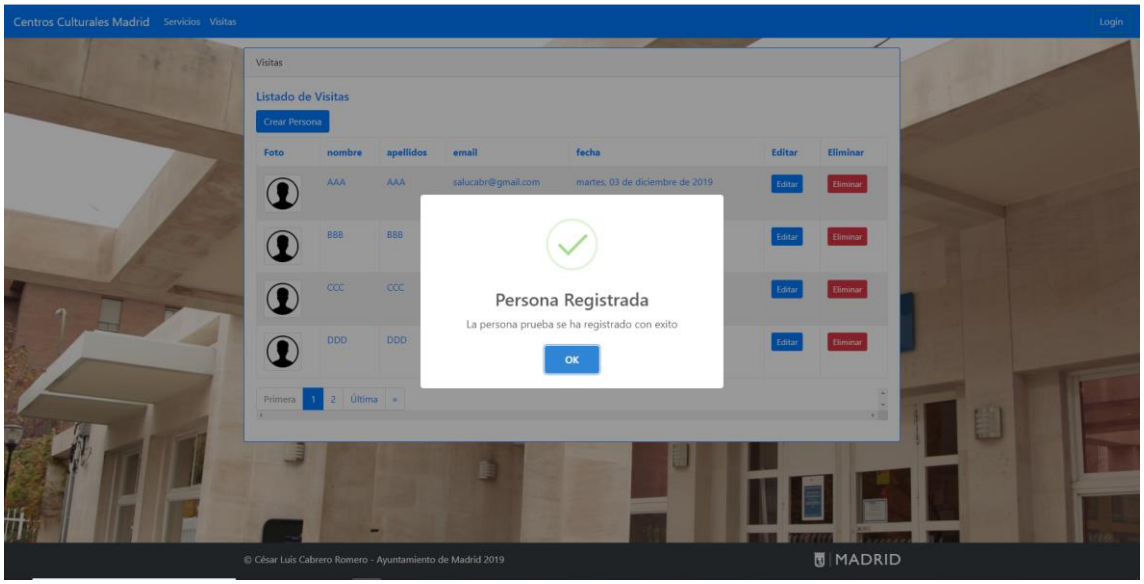
Cuando pulsemos en el botón Crear Persona dentro de la pagina principal, se nos mandará a un formulario donde deberemos rellenar los siguientes datos:



Si dejamos todos o algún campo sin rellenar, se nos mostrará un mensaje de alerta con los fallos pertinentes:

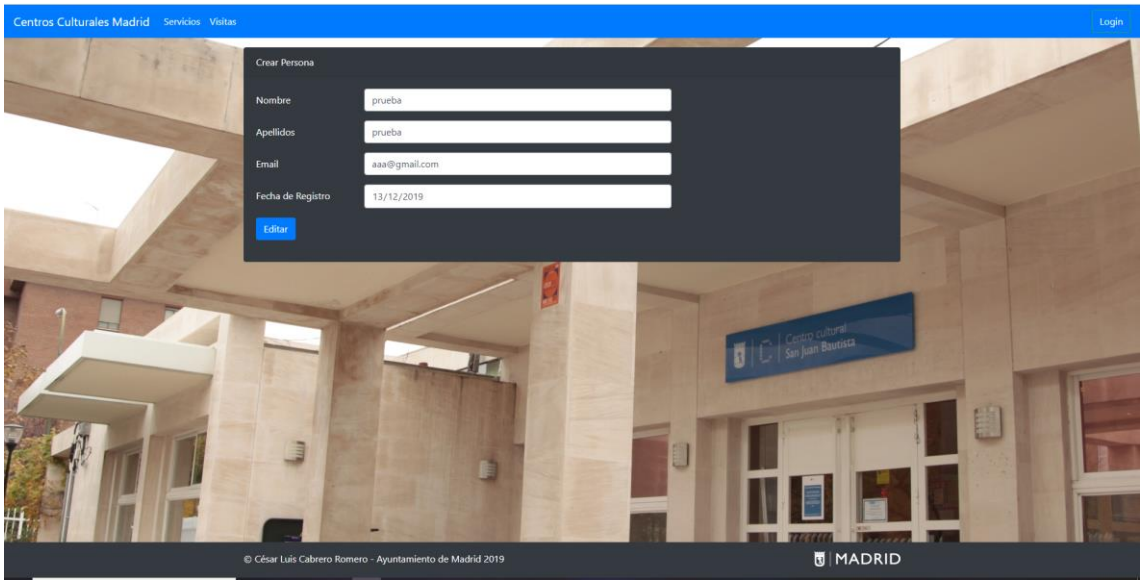


Cuando rellenemos todos los datos bien y pulsemos en el botón de crear, se nos mandará a la pagina inicial con un mensaje de éxito, además de habernos insertado el registro:



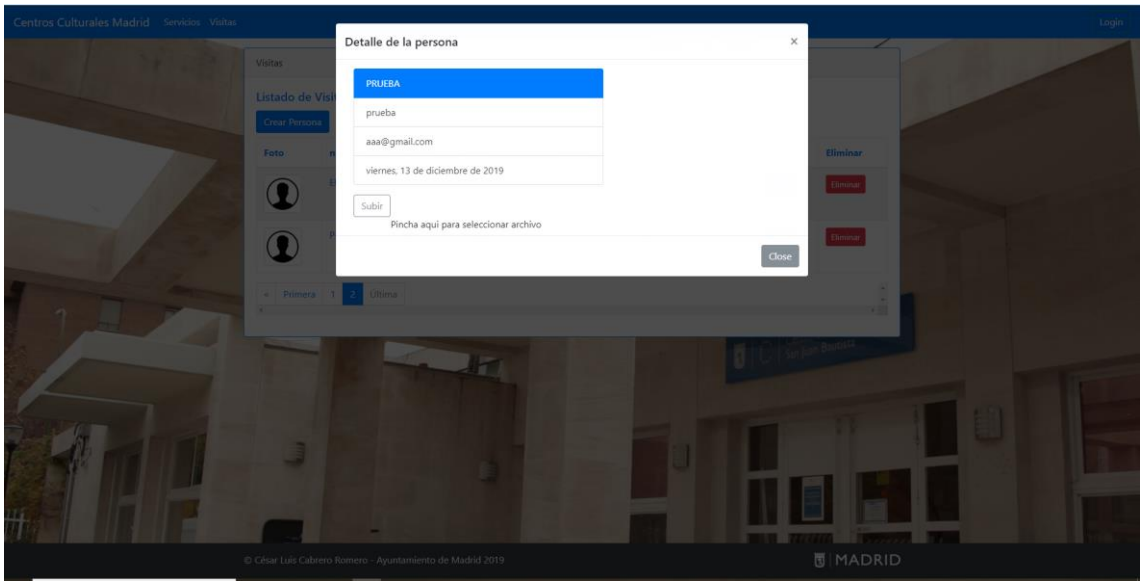
3.4 Pagina Editar Persona

La pagina editar persona será idéntica a la de crear persona, solo que cuando pinchemos en ella nos saldrán todos los datos rellenos.

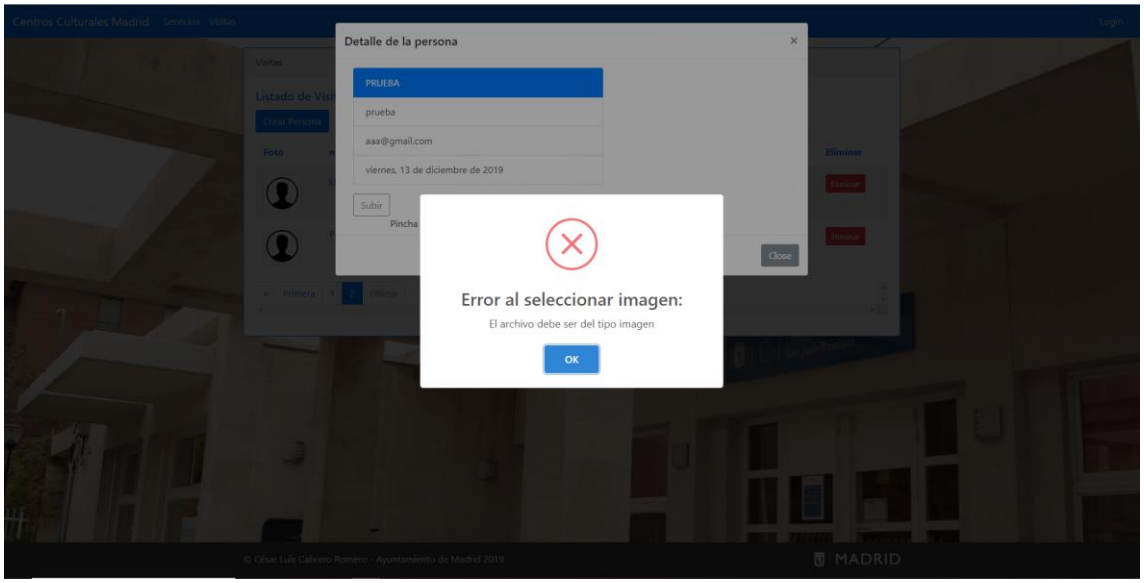


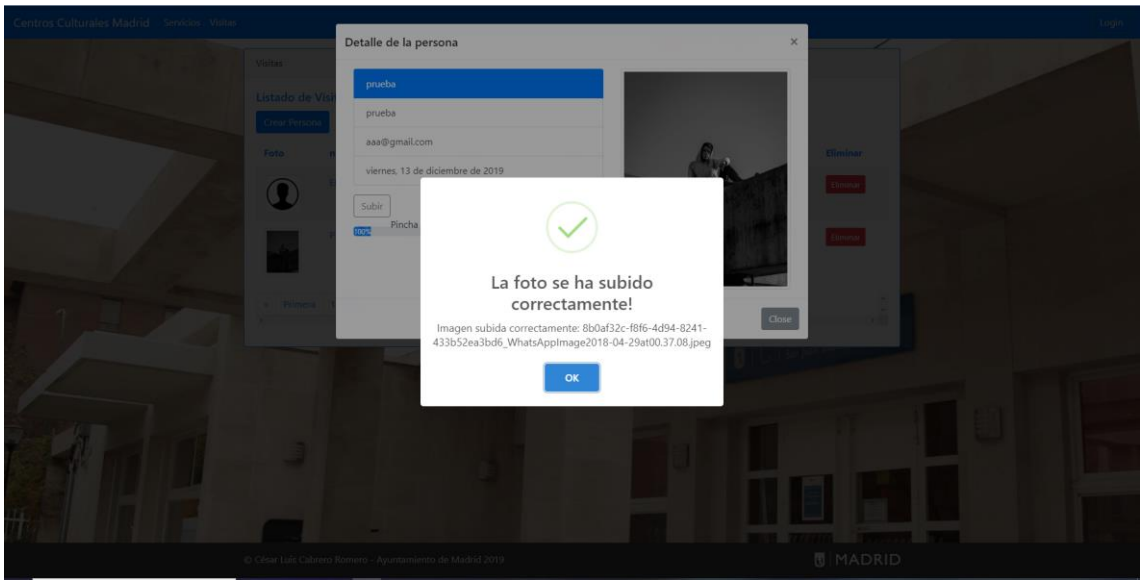
3.5 Pagina Foto

Al clicar en la foto de cada persona, se nos abrirá un modal, se nos mostraran los datos de la persona en la que hayamos clicado y se nos dejará subir o actualizar la foto actual:



Para insertar una foto tendremos que clicar sobre el texto “Pincha aquí para seleccionar un archivo”, coger un archivo tipo foto y clicar en el botón subir. Se nos notificará que la foto se ha actualizado correctamente. De lo contrario o al intentar subir un archivo que no sea de tipo foto,se nos mostrará un error:



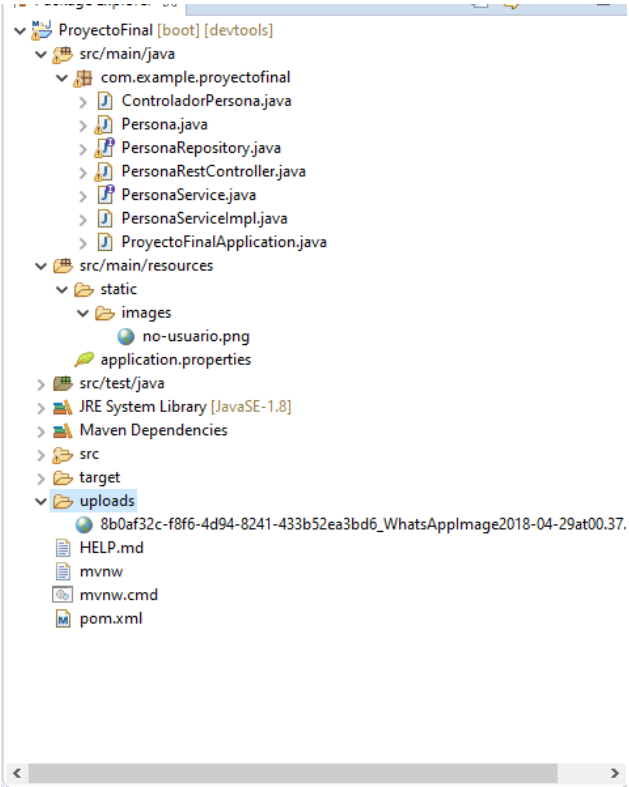


Aparte, el nombre de la foto contendrá al inicio unos números generados por la aplicación para que no haya dos fotos con el mismo nombre.

El apartado LOGIN esta deshabilitado en esta version

3.6 Estructura de código en Spring Tools Suite

En este apartado también hablaremos de como se estructura el código y lo que hace cada carpeta/clase. Comenzamos con la parte Java:



En el proyecto de Java tendremos un total de 2 interfaces y 5 clases. Ademas, en la carpeta uploads almacenaremos todas las imágenes de todos los usuarios, y en la carpeta resources/static/images almacenaremos una imagen global para todos los usuarios nuevos o que no tengan foto.

En las interfaces almacenaremos todos los métodos de las otras clases, por ejemplo en la interfaz PersonaService meteremos todos los métodos de la clase PersonaServiceImpl, que es la clase que implementa el servicio:

Interfaz PersonaService:

```java
package com.example.proyectofinal;

import java.util.List;


public interface PersonaService {

    public List<Persona> findAll();

    public Page<Persona> findAll(Pageable pageable);

    public Persona findById(Long id);

    public Persona save(Persona persona);

    public void delete(Long id);

}

```

Clase PersonaServiceImpl:

```java
package com.example.proyectofinal;

import java.util.List;

@Service
public class PersonaServiceImpl implements PersonaService {

    @Autowired
    private PersonaRepository personaDao;
    @Override
    @Transactional(readOnly = true)
    public List<Persona> findAll() {
        return (List<Persona>) personaDao.findAll();
    }

    @Override
    @Transactional(readOnly = true)
    public Page<Persona> findAll(Pageable pageable) {

        return personaDao.findAll(pageable);
    }

    @Override
    @Transactional(readOnly = true)
    public Persona findById(Long id) {
        return personaDao.findById(id).orElse(null);
    }

    @Override
    @Transactional
    public Persona save(Persona persona) {
        return personaDao.save(persona);
    }

    @Override
    @Transactional
    public void delete(Long id) {
        personaDao.deleteById(id);
    }


}
```

Por otra parte, en la clase Persona almacenaremos los datos de persona, tales como su ID, su nombre, sus apellidos, su email y su fecha de creación, asi como sus métodos get y set y el constructor, asi como las anotaciones de spring necesarias para el manejo de errores, declaraciones de variables, etc…

Clase Persona:

```java
package com.example.proyectofinal;
import java.io.Serializable;

@Entity
@Table(name="Persona")
public class Persona implements Serializable {

    @Id
    @GeneratedValue( strategy= GenerationType.IDENTITY )
    private long id;
    @NotEmpty(message="no puede estar vacio")
    @Size(min=3, max=20,message="el tamaño tiene que estar entre 3 y 20 caracteres")
    @Column(nullable=false)
    private String nombre;
    @NotEmpty(message="no puede estar vacio")
    private String apellidos;
    @NotEmpty(message="no puede estar vacio")
    @Email(message="no es una direccion de correo valida")
    @Column(nullable=false, unique=true)
    private String email;
    @NotNull(message="no puede estar vacio")
    @Column(name="create_at")
    @Temporal(TemporalType.DATE)
    private Date createAt;

    private String foto;


    public Date getCreateAt() {
        return createAt;
    }

    public void setCreateAt(Date createAt) {
        this.createAt = createAt;
    }

    public Persona(long id, String nombre,String apellidos, String email, Date createAt) {
        this.id = id;
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.email = email;
        this.createAt = createAt;

    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getApellidos() {
        return apellidos;
    }

    public void setApellidos(String apellidos) {
        this.apellidos = apellidos;
```

```
public Persona() {
}


public String getNombre() {
    return nombre;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}


public String getFoto() {
    return foto;
}

public void setFoto(String foto) {
    this.foto = foto;
}


private static final long serialVersionUID = 1L;
```

La interfaz PersonaRepository la usamos para extender el repositorio de JPA.

Clase PersonaRepository:

```
1  package com.example.proyectofinal;
2
3
4
5  import org.springframework.data.jpa.repository.JpaRepository;
10
11 @Repository
12 public interface PersonaRepository extends JpaRepository<Persona,Long> {
13 }
```

En la clase ControladorPersona incluiremos el controlador de toda la aplicación, también con anotaciones propias de Spring.

Clase ControladorPersona:

```java
package com.example.proyectofinal;

import java.util.List;

@RestController
@RequestMapping("/Persona")
public class ControladorPersona {

    @Autowired
    private PersonaRepository personaRepository;

    @RequestMapping("")
    public List<Persona> listPersona() {
        return (List<Persona>) personaRepository.findAll();
    }

    @RequestMapping(value="", method=RequestMethod.POST)
    public Persona crearPersona(@RequestBody Persona persona) {
        return personaRepository.save(persona);
    }
}
```

En la clase PersonaRestController tendremos todo el manejo de la aplicación,tanto si un método sale bien o mal y si sale mal el porque, como hacia donde va mapeado cada método con anotaciones de Spring. Esta clase es demasiado larga asique pongo un ejemplo pequeño con el método créate.

Clase PersonaRestController – Metodo Create

```java
    @PostMapping("/persona")
    public ResponseEntity<?> create(@Valid @RequestBody Persona persona, BindingResult result) {

        Persona personaNew = null;
        Map<String, Object> response = new HashMap<>();

        if(result.hasErrors()) {
            List<String> errors = result.getFieldErrors()
                    .stream()
                    .map(err -> "El campo '" + err.getField() + "' " + err.getDefaultMessage())
                    .collect(Collectors.toList());
            response.put("errors", errors);
            return new ResponseEntity<Map<String, Object>>(response, HttpStatus.BAD_REQUEST);
        }
        try {
            personaNew = personaService.save(persona);
        } catch(DataAccessException e) {
            response.put("mensaje", "Fallo al realizar la inserción en la base de datos");
            response.put("error", e.getMessage().concat(": ").concat(e.getMostSpecificCause().getMessage()));
            return new ResponseEntity<Map<String, Object>>(response, HttpStatus.INTERNAL_SERVER_ERROR);
        }
        response.put("mensaje","La persona se ha insertado con exito");
        response.put("persona", personaNew);
        return new ResponseEntity<Map<String, Object>>(response, HttpStatus.CREATED);
    }
```

Por último tenemos la clase ProyectoFinalApplication que solo contiene el main para que la aplicación pueda ser lanzada. Deberemos hacer el uso propio de la anotación @SpringBootApplication.

Clase ProyectoFinalApplication:

```java
package com.example.proyectofinal;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class ProyectoFinalApplication {

    public static void main(String[] args) {
        SpringApplication.run(ProyectoFinalApplication.class, args);
    }

}
```

Para finalizar dentro de la carpeta Resources tenemos el archivo application.properties. Este archivo sirve para configurar la conexión con la base de datos (En nuestro caso PostgreSQL). Tambien sirve para configurar parámetros tales como el tamaño máximo de la imagen a subir o el puerto por el que escucha la aplicación (Por defecto 8080).

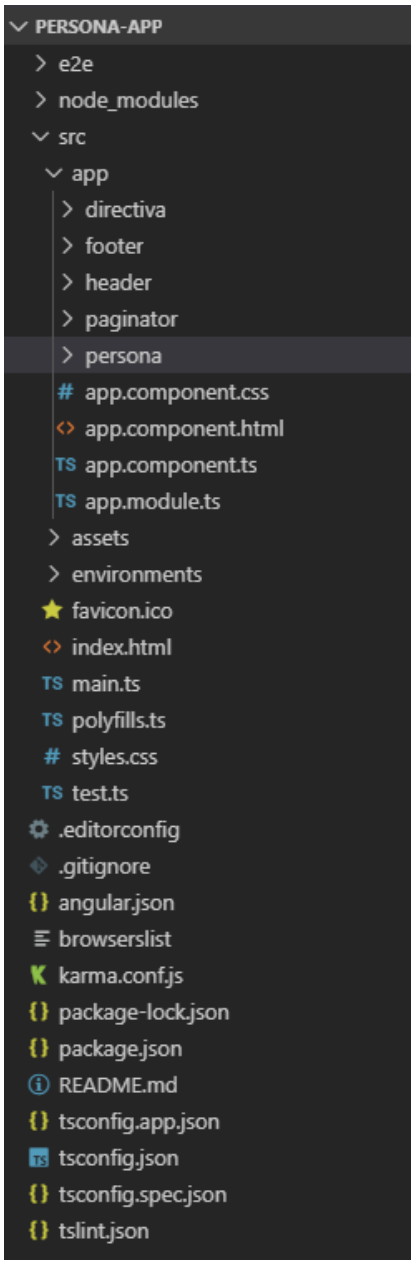
Application.properties:

```
spring.datasource.url=jdbc:postgresql://localhost/testdb
spring.datasource.username=postgres
spring.datasource.password=admin
spring.datasource.platform=postgres
spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.generate-ddl=true
spring.jpa.database=postgresql
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update

spring.rsocket.server.port=8080

spring.servlet.multipart.max-file-size=10MB
spring.servlet.multipart.max-request-size=10MB
```

3.7 Estructura en Visual Studio Code

En Visual Studio Code tendremos una serie de archivos y carpetas que son muy complejos pero una vez comprendidos se hace tarea fácil.

La estructura de archivos es la siguiente:

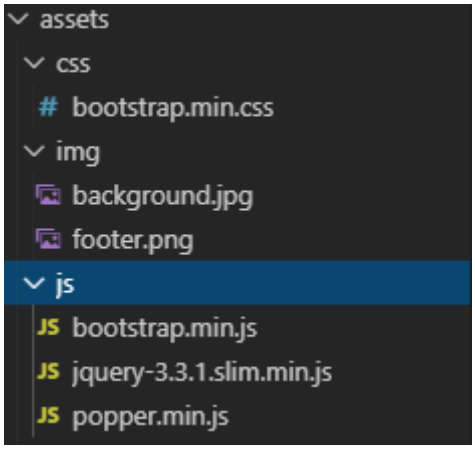


Las carpetas de e2e y node modules no son necesarias,al igual que todos los archivos por debajo de test.ts, solo nos centraremos en la carpeta src

Dentro de src tenemos las carpetas de app (donde se almacena toda la aplicación) la de assets (donde va el código css de Bootstrap o los js de Bootstrap) y la de environments, esta última sin uso.

Dentro de la carpeta de assets tenemos descargados los siguientes archivos.

Carpeta Assets:

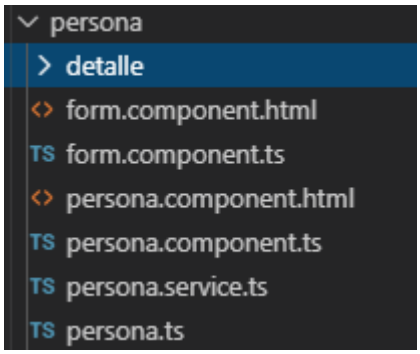


En css tenemos guardado todo el código de Bootstrap al igual que en JS, para que no nos tenga que redirigir a la pagina de Bootstrap y podamos usar la aplicación sin conexión. Tambien tenemos los javascript para las extensiones de jquery y Popper.

En la carpeta Src tendremos todo el proyecto, dividido por carpetas como pueden ser el header, el footer, el paginador, la pagina inicial o la pagina de servicios.

Cada carpeta tiene su propio estilo css, su html y su typescript (ts) donde se almacena todo el código a ejecutar.

Ejemplo de carpeta Persona:



Dentro de la carpeta persona, tenemos otra carpeta para mostrar en detalle cada persona, pero también tenemos el formulario de creación y el componente y servicio de persona, que reflejan con el proyecto en Spring. Vamos a ver un pequeño ejemplo tanto de HTML, como de TS.

Persona.component.html:

```
<detalle-persona *ngIf="personaSeleccionada" [persona] = "personaSeleccionada"></detalle-persona>
<div class="card border-primary mb-3">
  <div class="card-header">Visitas</div>
  <div class="card-body text-primary">
    <h5 class="card-title">Listado de Visitas</h5>
  <div class="my-2 text-left">
    <button class="btn btn-rounded btn-primary" type="button" [routerLink]="['/persona/form']">Crear Persona</button>
  </div>
  <div *ngIf="persona?.length==0" class="alert alert-info">
    Esta tabla esta vacia. ¡Prueba a insertar a una nueva persona!
  </div>
<table class="table table-bordered table-striped" *ngIf="persona?.length>0">
  <thead>
    <tr>
      <th>Foto</th>
      <th>nombre</th>
      <th>apellidos</th>
      <th>email</th>
      <th>fecha</th>
      <th>
        Editar
      </th>
      <th>
        Eliminar
      </th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let persona of persona">
      <td><img (click)="abrirModal(persona)" *ngIf="persona?.foto"
          src="http://localhost:8080/api/uploads/img/{{persona.foto}}" alt="{{persona.foto}}"
          class="img-thumbnail rounded" style="width: 64px; cursor:pointer;">
          <img (click)="abrirModal(persona)" *ngIf="!persona?.foto"
            src="http://localhost:8080/images/no-usuario.png" alt="Sin foto"
            class="img-thumbnail rounded" style="width: 64px; cursor:pointer;">
```

En este ejemplo mostrado,estamos creando la tabla que se ve al inicio de la aplicación,asi como que hace cada click.

Persona.component.ts – método delete:

```
delete(persona: Persona): void {
  const swalWithBootstrapButtons = Swal.mixin({
    customClass: {
      confirmButton: 'btn btn-success',
      cancelButton: 'btn btn-danger'
    },
    buttonsStyling: false
  })

  swalWithBootstrapButtons.fire({
    title: '¿Eliminar?',
    text: `¿Seguro que desea eliminar a la persona ${persona.nombre} ${persona.apellidos}?`,
    icon: 'warning',
    showCancelButton: true,
    confirmButtonText: 'Si,Eliminar',
    cancelButtonText: 'No,Cancelar',
    reverseButtons: true
  }).then((result) => {
    if (result.value) {
      this.personaService.delete(persona.id).subscribe(
        response => {
          this.persona = this.persona.filter(per => per !== persona)
          swalWithBootstrapButtons.fire(
            'Eliminado',
            `La persona ${persona.nombre} ha sido eliminada con exito`,
            'success'
          )
        }
      )
    }
  })
}
```

En este ejemplo podemos ver como haríamos un método delete en Typescript,que interactue con Spring y además el mensaje lanzado.

Toda la estructura de Angular es prácticamente similar, asique con un solo ejemplo podemos entender prácticamente el 90% del código (exceptuando imports propios de cada clase).

4 Pruebas

Para el desarrollo de la aplicación se han hecho algunas pruebas, teniendo éxito en casi todas ellas.Al tener los errores depurados en la aplicación,estos se pueden comprobar haciendo mal uso de esta, lo que desencadenara un mensaje de error para todos los casos gracias al manejo de errores implementado. Los errores antes citados en la instalación son los únicos errores que he podido experimentar. Otros errores comunes son:

- Puerto 8080 already in use: este error es un error común de Spring Tools Suite, el cual se da cuando el puerto 8080 esta siendo utilizado por otra aplicación. Para subsanar este error,podemos probar a detener todas las ejecuciones de spring que tengamos. Esto detiene el error en un 90% de los casos, bien sea porque tengamos 2 ejecuciones abiertas o por errores con el compilador. El método mas efectivo es irnos a nuestra terminal del sistema operativo y matar el proceso que este usando el puerto 8080. En Windows, el proceso seria el siguiente:

  1. Abrimos el cmd y escribimos el comando netstat -oan

  ```
  C:\Users\saluc>netstat -oan
  ```

  2. Nos saldrán un montón de procesos y puertos. A nosotros solo nos interesa el proceso que use el puerto 8080

  ```
  TCP    [::]:8080              [::]:0                 LISTENING       11804
  ```

  3. Ahora que tenemos localizado el proceso, basta con poner el siguiente comando taskkill /F /PID nro_pid, siendo nro_pid el ultimo numero de la anterior imagen.

  ```
  C:\Users\saluc>taskkill /F /PID 11804
  ```

  Asi detendremos el proceso y podremos volver a ejecutar nuestra aplicación sin errores.

Para este proyecto se ha creado varios Backups, cada uno en una carpeta con un archivo txt llamado changelist,con cada uno de los cambios realizados en esa version. Esto en un ámbito empresarial es muy importante para saber lo que se ha cambiado del proyecto en cada version. Si se necesitan dichas carpetas, se le pueden pedir al alumno.

5 Conclusiones

En este apartado me gustaría hablar sobre como he ido evolucionando con el proyecto, tanto en limitaciones como en experiencia propia desde 0 con estas tecnologías.

He trabajado con Angular y Spring, dos lenguajes altamente demandados en empresas actuales, y me quedo con el aprendizaje recibido y aplicado, ya que este proyecto me ha ayudado muchísimo a reforzar mi experiencia en Java (que es lo que mas problemas me ha causado en todo el grado) y para aprender que hay lenguajes mas allá de html, css o php para paginas web.

5.1 Limitaciones

La mayor limitación que he tenido en el proyecto ha sido el tiempo de entrega. Ya que decidí cambiar de proyecto y de tecnología a 3 semanas de entregar este documento, me he tenido que esforzar muchísimo y quizás la falta de tiempo se vea reflejada en algún apartado de la aplicación como por ejemplo en el progressbar.

A nivel de hardware, no he tenido ningún problema de limitación.

5.2 Futuras líneas de investigación

Esta aplicación desde mi punto de vista podría implementarse de manera real en todos los centros culturales de la comunidad de Madrid para, por ejemplo, informar de actividades o eventos.

Hay muchas funcionalidades que me hubiese gustado meter como, por ejemplo:

- Hacer un login tanto para usuarios como para personal administrativo, diferenciando a cada uno de estos sus funciones. Por ejemplo el personal podría dar de alta nuevos usuarios, y los usuarios podrían decidir a que boletines suscribirse o cambiar su foto.
- Disponer de otra tabla en la base de datos para seleccionar el distrito al que pertenecen los usuarios, o su centro favorito.
- Investigar el software y hardware necesario para hacer tarjetas físicas de contacto que interactúen con los datos de la aplicación para alquiler de libros, apuntarse a clases…

6 Bibliografía

Estos son todos los canales oficiales en los que he buscado información:

- Tutoriales propios dados por la empresa de practicas (ATOS)
- Bootstrap: https://getbootstrap.com/
- Angular Cli: https://angular.io/
- Angular Material: https://material.angular.io/
- W3Schools: https://www.w3schools.com/
- Sweetalert2: https://sweetalert2.github.io/
- NPM: https://www.npmjs.com/
- Spring Tools Suite: https://spring.io/tools
- StackOverflow: https://es.stackoverflow.com/
- Visual Code Studio: https://code.visualstudio.com/
- PostgreSQL: https://www.postgresql.org/
- Java API: https://docs.oracle.com/javase/7/docs/api/
- Spring API: https://docs.spring.io/spring/docs/current/javadoc-api/