

Proyecto Final

Tutor proyecto: Joan Gerard Camarena

Tutor grupo: Jose Alfredo Murcia

Valentin Pankov Fichev

09/04/2022

Proyecto Final

Contenido

Datos del alumno	5
Datos del proyecto	5
1. Presentación.....	6
2. Contenido.....	6
3. Análisis/herramientas	7
3.1 Base de datos	7
3.2 Backend	7
3.3 Frontend.....	9
4.01 Desarrollo - Back	11
4.1 Estructura general	11
4.2 Model	13
4.2.1 Alumno	14
4.2.2 Asignatura	15
4.2.3 Curso.....	15
4.2.4 Examen	16
4.2.5 Modulo	16
4.2.6 NotaExamen	17
4.2.7 NotaExamenPK.....	17
4.2.8 Profesor	18
4.2.9 ProfesorAsignatura.....	18
4.2.10 ProfesorAsignaturaPK	19
4.2.11 Rol.....	19
4.2.12 Usuario	20

4.3	Controller	21
4.3.1	AlumnoController.....	21
4.3.2	AsignaturaController	22
4.3.3	CursoController	22
4.3.4	ExamenController.....	22
4.3.5	ProfesorController.....	23
4.3.6	RolController	23
4.3.7	UsuarioController	23
4.4	Service	24
4.5	Repository	25
4.5.1	AlumnoRepository.....	25
4.5.2	AsignaturaRepositoy	26
4.5.3	CursoRepository	26
4.5.4	ExamenRepository	26
4.5.5	ModuloRepository.....	26
4.5.6	ProfesorAsignaturaRepository	26
4.5.7	ProfesorRepository.....	27
4.5.8	RolRepository	27
4.5.9	UsuarioRepository.....	27
4.6	Configuration.....	28
4.7	Security.....	29
4.7.1	AuthenticationRestController	29
4.7.2	JwtAuthenticationEntryPoint	30
4.7.3	JwtToken	30
4.7.4	JwtTokenUtil.....	31
4.7.5	JwtTokenFilter	32
4.7.6	SecurityConfiguration.....	33
4.7.7	UsuarioCredenciales.....	35
4.7.8	UsuarioDetails	35
4.7.9	UsuarioDetailsService.....	36
4.8	Application.properties.....	37
4.02	Desarrollo – Base de datos.....	38
4.03	Desarrollo – Front	41

4.1 Login	44
4.2 Dashboard	47
4.3 Curso	50
4.4 Alumno	52

Datos del alumno

Nom i cognoms	Valentin Pankov Fichev
NIF/NIE	18508056W
Curs i CF	2n DAM-SP

Datos del proyecto

Títol del projecte	
Nom del tutor individual	Joan Gerard Camarena Estruch
Nom del tutor del grup	Jose Alfredo Murcia Andrés
Resum	Aplicación web para visualizar de forma clara los datos principales de un alumno y sus notas. Backend hecho en Java (Spring Boot) y el frontend con Vue.
Abstract	
Mòduls implicats	<ol style="list-style-type: none">1. Acceso a datos2. Desarrollo de interfaces3. Programación4. Base de datos
Data de presentació	Viernes 10 de Junio de 2022

1. Presentación

La idea de este proyecto es visualizar y acceder de una forma simple y fácil a los datos de un alumno y a sus calificaciones. Todo esto con el objetivo de poder llevar un seguimiento fácil de cada alumno y de un vistazo poder ver en que asignaturas puede ir mas flojo, en cuales más preparado.

El **backend** se ha hecho en **Java**, ya que es el lenguaje con el que hemos trabajado principalmente en el superior y en mi caso, recibí una formación en la empresa de practicas de **Spring boot**, y es por eso que decidí utilizar este framework.

En la parte de **frontend** he utilizado **Vue**, ya que consideré que era un buen framework para empezar a experimentar en el mundo del front, ya que lo máximo que he hecho ha sido html5 y css, por lo cual ha sido un gran reto hacer esto.

En cuanto a la **base de datos**, he utilizado JPA para autogenerar la base de datos en mysql, ya que me ha parecido una herramienta super útil que te ahorra muchísima faena.

2. Contenido

Como resumen, tenemos:

- Base de datos: Mysql generada con JPA desde el backend
- Backend: Java (Spring boot)
- Frontend: Vue

3. Análisis/herramientas

En este apartado quiero entrar un poco más en detalle con todas las herramientas que se han utilizado y como se ha estructurado todo, antes de pasar al siguiente apartado de desarrollo que ahí ya se veremos el código con muchísimo más detalle.

3.1 Base de datos

Vamos a empezar primero con la base de datos.

La idea de la base de datos es, por una parte, que tenga una tabla módulos, donde vamos a registrar los diferentes módulos de FP, DAM, DAW, por ejemplo, y la ESO

Cada uno de estos módulos va a tener “cursos” o años, los cuales van a contener asignaturas, por ejemplo, el curso 1 de DAM tiene, base de datos, programación... ponemos siempre la foreign key en la parte de los “muchos”, en estos casos la relación es uno a muchos.

Luego, tenemos una relación ternaria para decir que profesor imparte qué asignaturas y a que curso pertenecen estas. Tenemos como clave primaria la asignatura y el curso, que al mismo tiempo son foreign keys, para poder asignar varias asignaturas a un curso y que una asignatura pueda pertenecer a diferentes cursos, como en el caso de DAM y DAW, el primer año. Faltaría solamente hacer una constraint para indicar que cuando curso vale “x” y asignatura “y” solo pueda haber un profesor dando clase exactamente esa asignatura de ese curso.

Cada asignatura va a tener unos exámenes que la tabla simplemente va a ser una “convocatoria”, vamos a registrar el examen en si. Por otra parte tenemos a los alumnos que tienen una relación muchos a muchos con exámenes, por eso surge la tabla del medio que son las notas. Clave primaria va a ser alumno y examen, de esta manera podemos tener a muchos alumnos para el mismo examen y muchos exámenes para el mismo alumno y luego a parte la nota.

Para acabar tenemos la relación de profesor con usuario que es de uno a uno, lo que significa que un profesor puede tener un usuario y este solo puede pertenecer a un profesor y este mismo usuario puede tener un rol.

3.2 Backend

Como hemos mencionado anteriormente, el backend está hecho con Spring, que ha sido muy cómodo y nos ha facilitado muchísimo la faena. Hemos utilizado la librería que tiene Spring, Lombok, que simplemente con anotar una clase con las etiquetas `@AllArgsConstructor`, `@NoArgsConstructor` y `@Data`, nos crea el constructor con todos los atributos, un constructor vacío y todos los setter y getters, pero estos no se ven en el código. El código se ve muchísimo mas limpio y como justo la parte de los setter y getters es siempre la misma, con saber cuales son los atributos de la clase, ya sabemos que cada uno de ellos tiene get y set.

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
public class Rol {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String nombre;
    @OneToMany(mappedBy = "rol")
    @JsonBackReference
    private List<Usuario> usuarios;

    public Rol(Integer id, String nombre){
        this.id = id;
        this.nombre = nombre;
    }
}
```

Otra librería muy útil es la de seguridad de Spring y la de jsonwebtoken que nos ha permitido llevar todo el tema de seguridad a través de un usuario y contraseña y a parte de un JWT, que mas adelante veremos con detalle.

3.3 Frontend

En la parte del frontend tenemos Vue, que tiene una parte de vistas, que es donde crearemos cada una de nuestras pantallas, que estas se componen por una parte HTML, otra que es JavaScript y otra, opcional, que es el CSS, por si queremos dar algún diseño.

```
views / pages / Page404.vue / Page404.vue / template
<template>
  <div class="bg-light min-vh-100 d-flex flex-row align-items-center">
    <CContainer>
      <CRow class="justify-content-center">
        <CCol :md="6">
          <div class="clearfix">
            <h1 class="float-start display-3 me-4">404</h1>
            <h4 class="pt-3">Oops! You're lost.</h4>
            <p class="text-medium-emphasis float-start">
              The page you are looking for was not found.
            </p>
          </div>
          <CInputGroup class="input-prepend">
            <CInputGroupText>
              <CIcon icon="cil-magnifying-glass" />
            </CInputGroupText>
            <CFormInput type="text" placeholder="What are you looking for?" />
            <CButton color="info">Search</CButton>
          </CInputGroup>
        </CCol>
      </CRow>
    </CContainer>
  </div>
</template>

<script>
export default {
  name: 'Page404',
}
</script>
```

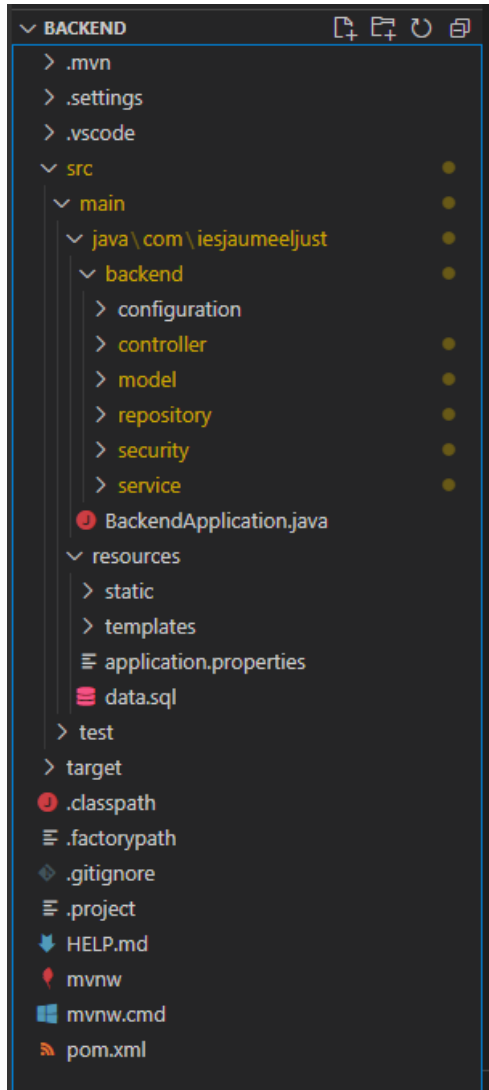
Por otra está el router que es el encargado de gestionar todo el tema de rutas dentro de lo que es nuestro front. Asigna un componente a la ruta que le indiquemos, entonces nosotros al pulsar un botón, le indicamos que queremos que nos redireccione a X ruta y de esta manera el router sabe que componente/vista cargar.

```
src > router > JS index.js > router > scrollBehavior
1  import { h, resolveComponent } from 'vue'
2  import { createRouter, createWebHashHistory } from 'vue-router'
3
4  import DefaultLayout from '@/layouts/DefaultLayout'
5
6  const routes = [
7    {
8      path: '/',
9      name: 'Home',
10     component: DefaultLayout,
11     redirect: '/dashboard',
12     children: [
13       {
14         path: '/dashboard',
15         name: 'Dashboard',
16         meta: {
17           requiresAuth: true,
18         },
19       },
20     ],
21   },
22 ]
```

4.01 Desarrollo - Back

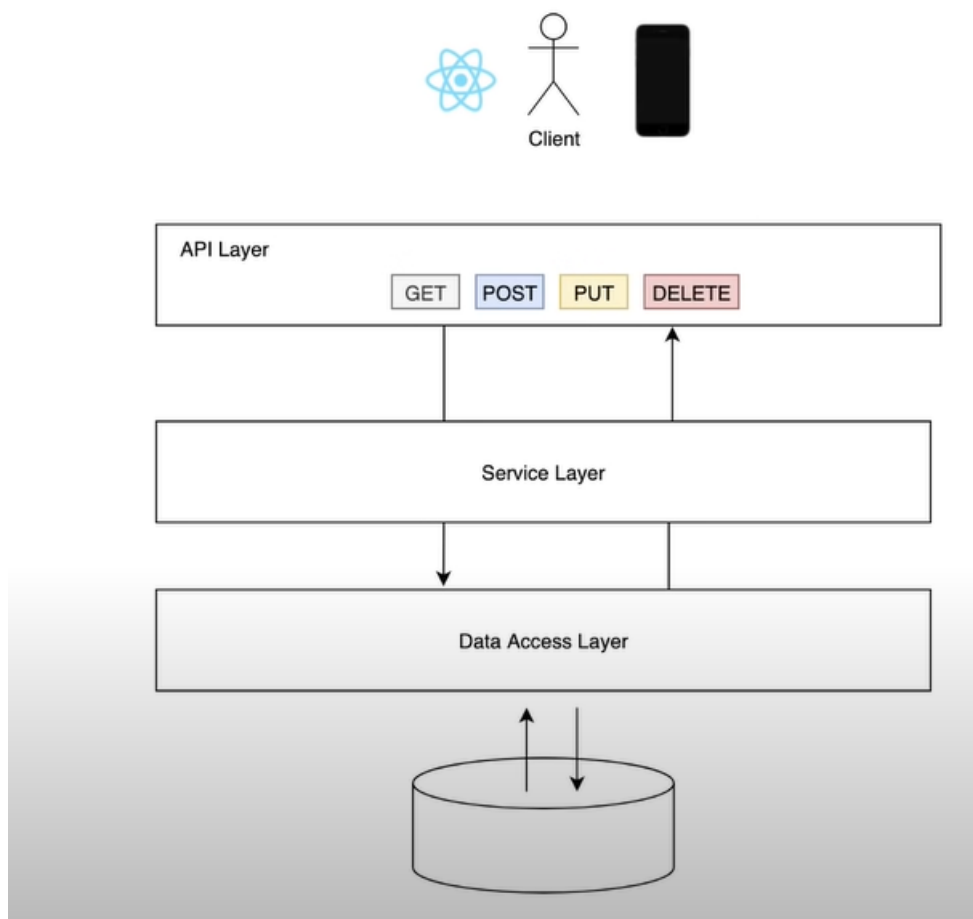
4.1 Estructura general

Vamos a empezar viendo el backend y a la par la base de datos.



Esta es la estructura principal del proyecto.

- Primero tenemos la carpeta configuración, que es donde esta el fichero de configuración web, que habilitamos el CORS, para poder realizar la comunicación entre el back y el front, que por defecto viene bloqueado.
- A continuación están los controller, que es donde definimos el endpoints, las consultas a nuestra API REST.
- Luego, los modelos, que son las clases principales de nuestro proyecto y al mismo tiempo las entidades que se crean en la base de datos.
- Los repositorios, que es desde donde accedemos a la base de datos
- La carpeta de seguridad, que es donde esta todo lo relacionado con la seguridad y la configuración del JWT.
- Por último los servicios, que es la parte lógica de nuestra API REST.



En esta imagen podemos ver de forma muy simple y resumida como esta estructurada una API REST.

En el primer bloque tenemos el controller, al que se le pueden hacer peticiones GET, POST, PUT, DELETE. La primera para obtener datos de la base de datos, la segunda para insertar datos, actualizar datos y borrar datos, respectivamente.

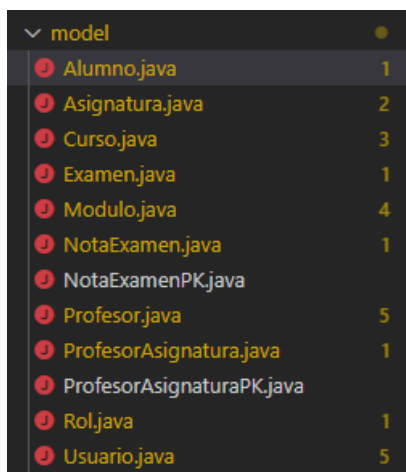
El segundo bloque que vemos es el service, es donde tendremos toda la lógica de nuestra aplicación. Si antes de devolver un dato, tenemos que comprobar que cumple X requisitos, es en esta capa donde lo haremos.

Por último tenemos el repositorio, que es la capa que comunica directamente con la base de datos.

El orden siempre es el mismo. El usuario hace una petición al controller, este llama al servicio, el servicio llama al repositorio, el repositorio pide los datos a la base de datos y estos datos siguen el mismo camino que el de ida, de base de datos pasa a repositorio, a servicio, controller y cliente respectivamente.

4.2 Model

Vamos a empezar por lo modelos



model	
Alumno.java	1
Asignatura.java	2
Curso.java	3
Examen.java	1
Modulo.java	4
NotaExamen.java	1
NotaExamenPK.java	
Profesor.java	5
ProfesorAsignatura.java	1
ProfesorAsignaturaPK.java	
Rol.java	1
Usuario.java	5

Intentaré ser lo mas breve posible ya que es algo repetitivo.

4.2.1 Alumno

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
public class Alumno {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String nombre;
    @Column(unique = true)
    private String dni;
    private String direccion;
    private String telefono;
    private Date fechaNacimiento;

    @ManyToOne
    @JoinColumn(name = "curso_alumno")
    @JsonManagedReference
    private Curso curso;
```

Tenemos al alumno con los siguientes atributos y abajo del todo vemos que tiene un curso y como se marca la relación que tiene con dicho curso. Es una relación de muchos a uno, ya que un curso tiene muchos alumnos y un alumno pertenece solo a un curso al mismo tiempo. Al estar en la parte de los muchos, que son los alumnos, se pone el `@ManyToOne`. El `JoinColumn` indica la columna en la base de datos. Por último tenemos la anotación `JsonManagedReference` que lo que evita es una recursión infinita al tener una relación bidireccional entre clases. Veremos mas adelante que las relaciones marcadas con `JsonManagedReference` son las que se serializan y aparecen en la base de datos, las otras que van con la anotación `JsonBackReference`, aparecen en la clase en java pero no aparecerán en la base de datos ya que no son serializables.

4.2.2 Asignatura

```

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Asignatura {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String nombre;
    @OneToMany(mappedBy = "asignatura")
    private List<Examen> examenes;
    @ManyToOne
    @JoinColumn(name = "curso_asignatura")
    @JsonManagedReference
    private Curso curso;
}

```

Por aquí tenemos asignatura, que tiene una relación @OneToMany con Exámenes ya que una asignatura tiene muchos exámenes, pero un examen en concreto solo puede pertenecer a una asignatura. El resto de anotaciones hasta ahora ya las hemos visto.

4.2.3 Curso

```

23
24 @Entity
25 @Data
26 @AllArgsConstructor
27 @NoArgsConstructor
28 public class Curso {
29     @Id
30     @GeneratedValue(strategy = GenerationType.IDENTITY)
31     private Integer id;
32     private Integer año;
33     @ManyToOne
34     @JoinColumn(name = "modulo_curso")
35     @JsonManagedReference
36     private Modulo modulo;
37     @OneToMany(mappedBy = "curso")
38     @JsonBackReference
39     private List<Asignatura> asignaturas;
40     @OneToMany(mappedBy = "curso")
41     @JsonBackReference
42     private List<Alumno> alumnos;
43 }
44

```

4.2.4 Examen

```
16
17 @Entity
18 @Data
19 @AllArgsConstructor
20 @NoArgsConstructor
21 public class Examen {
22     @Id
23     @GeneratedValue(strategy = GenerationType.IDENTITY)
24     private Integer id;
25     private Date fecha;
26     @ManyToOne
27     @JoinColumn(name = "asignatura")
28     private Asignatura asignatura;
29
30 }
```

4.2.5 Modulo

```
21
22 @Entity
23 @Data
24 @AllArgsConstructor
25 @NoArgsConstructor
26 public class Modulo {
27     @Id
28     @GeneratedValue(strategy = GenerationType.IDENTITY)
29     private Integer id;
30     private String nombre;
31     private Integer duracionAños;
32     @OneToMany(mappedBy = "modulo")
33     @JsonBackReference
34     private List<Curso> cursos;
35
36 }
```


4.2.6 NotaExamen

```
14  @Data
15  @AllArgsConstructor
16  @NoArgsConstructor
17  @Entity
18  public class NotaExamen {
19      @EmbeddedId
20      private NotaExamenPK id;
21
22      @ManyToOne
23      @MapsId("idExamen")
24      @JoinColumn(name = "id_examen")
25      private Examen examen;
26      @ManyToOne
27      @MapsId("idAlumno")
28      @JoinColumn(name = "id_alumno")
29      private Alumno alumno;
30
31      private Float nota;
32  }
33
```

Aquí podemos ver que tenemos una `EmbeddedId` que es una clave compuesta, que se indica en la clase que veremos a continuación.

4.2.7 NotaExamenPK

```
11
12  @Embeddable
13  @Data
14  @AllArgsConstructor
15  @NoArgsConstructor
16  public class NotaExamenPK implements Serializable{
17      private static final long serialVersionUID = -1963351309820398400L;
18
19      @Column(name = "id_examen")
20      private Integer idExamen;
21      @Column(name = "id_alumno")
22      private Integer idAlumno;
23  }
24
```

Aquí es donde definimos que dos atributos van a ser las ID's de la clase anterior, `NotaExamen` e indicamos el nombre de la columna que van a tener en la base de datos.

4.2.8 Profesor

```
23 @Data
24 @AllArgsConstructor
25 @NoArgsConstructor
26 @Entity
27 public class Profesor {
28     @Id
29     @GeneratedValue(strategy = GenerationType.IDENTITY)
30     private Integer id;
31     private String nombre;
32     private String dni;
33     private String direccion;
34     private String telefono;
35     private Date fechaNacimiento;
36     @Exclude
37     @OneToOne(mappedBy = "profesor")
38     @JsonBackReference
39     private Usuario usuario;
40 }
```

Aquí lo único que tenemos de diferente es la anotación Exclude que la tuve que poner porque daba un error al crear el hashCode con el lombok automáticamente y entraba en bucle, entonces tuve que excluirlo en una de las dos clases, de Profesor o Usuario.

4.2.9 ProfesorAsignatura

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
public class ProfesorAsignatura {
    @EmbeddedId
    private ProfesorAsignaturaPK id;

    @ManyToOne
    @MapsId("idCurso")
    @JoinColumn(name = "id_curso")
    private Curso curso;

    @ManyToOne
    @MapsId("idAsignatura")
    @JoinColumn(name = "id_asignatura")
    private Asignatura asignatura;

    @ManyToOne
    @JoinColumn(name = "id_profesor")
    private Profesor profesor;
}
```

Esta es la tabla que se genera de la relación ternaria que hay entre profesor curso y asignatura, en la que tenemos una clave compuesta entre asignatura y curso

4.2.10 ProfesorAsignaturaPK

```

12  @Data
13  @AllArgsConstructor
14  @NoArgsConstructor
15  @Embeddable
16  public class ProfesorAsignaturaPK implements Serializable {
17
18      private static final long serialVersionUID = -7073467377734704051L;
19
20      @Column(name = "id_curso")
21      private Integer idCurso;
22      @Column(name = "id_asignatura")
23      private Integer idAsignatura;
24
25  }

```

4.2.11 Rol

```

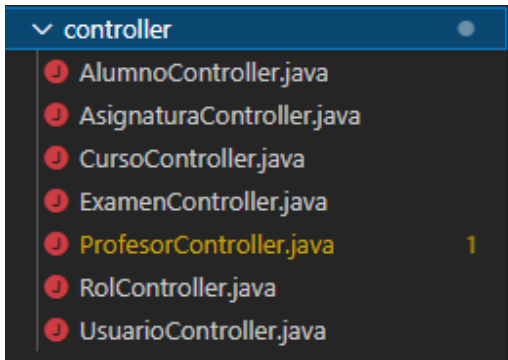
17
18  @Data
19  @AllArgsConstructor
20  @NoArgsConstructor
21  @Entity
22  public class Rol {
23
24      @Id
25      @GeneratedValue(strategy = GenerationType.IDENTITY)
26      private Integer id;
27      private String nombre;
28      @OneToMany(mappedBy = "rol")
29      @JsonBackReference
30      private List<Usuario> usuarios;
31
32      public Rol(Integer id, String nombre){
33          this.id = id;
34          this.nombre = nombre;
35      }
36

```

4.2.12 Usuario

```
22  @Data
23  @AllArgsConstructor
24  @NoArgsConstructor
25  @Entity
26  public class Usuario {
27      @Id
28      @Column(updatable = false, nullable = false)
29      private String nombre;
30      private String password;
31      @OneToOne
32      @JoinColumn(name = "profesor_id", referencedColumnName = "id")
33      @JsonManagedReference
34      private Profesor profesor;
35      @ManyToOne
36      @JoinColumn(name = "rol_usuario")
37      @JsonManagedReference
38      private Rol rol;
39
40      public Usuario(String nombre, String password, Profesor profesor){
41          this.nombre = nombre;
42          this.password = password;
43          this.profesor = profesor;
44      }
45      public Usuario(String nombre, String password, Rol rol){
46          this.nombre = nombre;
47          this.password = password;
48          this.rol = rol;
49      }
50  }
```

4.3 Controller



4.3.1 AlumnoController

```
@RestController
@RequestMapping("/api")
public class AlumnoController {

    @Autowired
    AlumnoService alumnoService;

    @GetMapping("/alumno")
    public Alumno getAlumnoById(@RequestParam("idAlumno") Integer id){
        return alumnoService.findAlumnoById(id);
    }

    @GetMapping("/alumnos")
    public List<Alumno> getAlumnos(){
        return alumnoService.findAll();
    }

    @GetMapping("/alumnos/curso")
    @ResponseBody
    public List<Alumno> findAllByCurso(@RequestParam("idCurso") Integer id){
        return alumnoService.findAllByCurso(id);
    }
}
```

Aquí tenemos los endpoints de la API REST relacionados con Alumno

Primero indicamos que después de localhost:8080 tenemos /api y a continuación va el la petición que queremos hacer, por ejemplo el GET de alumno, que sería localhost:8080/api/alumno?idAlumno=1, el ?idAlumno indica que es un parámetro, que es requerido para esta petición.

Al principio podemos ver el alumnoService que es una interfaz la cual se inyecta, que es lo que indica el @Autowired, con lo cual “une” al controller con el service, de esta manera podemos llamarlo desde aquí.

4.3.2 AsignaturaController

```
@RestController
@RequestMapping("/api")
public class AsignaturaController {

    @Autowired
    AsignaturaService asignaturaService;

    @GetMapping("asignaturas/curso")
    public List<Asignatura> findAllByCurso(@RequestParam("idCurso") Integer id){
        return asignaturaService.findAllByCurso(id);
    }
}
```

4.3.3 CursoController

```
@RestController
@RequestMapping("/api")
public class CursoController {

    @Autowired
    CursoService cursoService;

    @GetMapping("/cursos/modulo")
    @ResponseBody
    public List<Curso> getCursosByModulo(@RequestParam("idModulo") Integer idModulo){
        return cursoService.getCursosByModulo(idModulo);
    }
}
```

4.3.4 ExamenController

```
13
14 @RestController
15 @RequestMapping("/api")
16 public class ExamenController {
17
18     @Autowired
19     ExamenService examenService;
20
21     @GetMapping("examenes/asignatura")
22     public List<Examen> getExamenesByAsignatura(@RequestParam("idAsignatura") Integer id){
23         return examenService.findAllByAsignatura(id);
24     }
25 }
26
```

4.3.5 ProfesorController

```
15
16 @RestController
17 @RequestMapping("/api")
18 public class ProfesorController {
19     @Autowired
20     ProfesorService profesorService;
21
22     @GetMapping("/profesor/modulos")
23     @ResponseBody
24     public List<Modulo> getModulosByProfesor(@RequestParam("idProfesor") Integer idProfesor){
25         return profesorService.findModulosByProfesor(idProfesor);
26     }
27 }
28
```

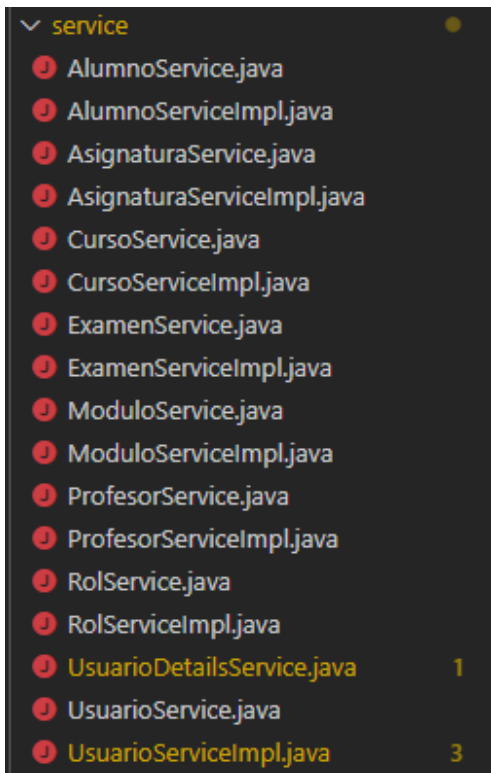
4.3.6 RolController

```
12 @RestController
13 @RequestMapping("/api")
14 public class RolController {
15
16     @Autowired
17     RolService rolService;
18
19     @GetMapping("/rol")
20     public Rol getRolByName(@RequestParam("nombre") String nombre){
21         return rolService.findRolByName(nombre);
22     }
23 }
24
```

4.3.7 UsuarioController

```
11
12 @RestController
13 @RequestMapping("/api")
14 public class UsuarioController {
15
16     @Autowired
17     UsuarioService usuarioService;
18
19     @GetMapping("/usuario")
20     public Usuario getUsuarioById(@RequestParam("idUsuario") String nombre){
21         return usuarioService.findUsuarioById(nombre);
22     }
23 }
24
```

4.4 Service



Por una parte tenemos la interfaz

```
public interface AlumnoService {  
    public List<Alumno> findAll();  
    public Alumno findAlumnoById(Integer id);  
    public List<Alumno> findAllByCurso(Integer id);  
}
```

Y por otra, el servicio con la interfaz implementada donde llamamos al repositorio y hacemos la consulta.

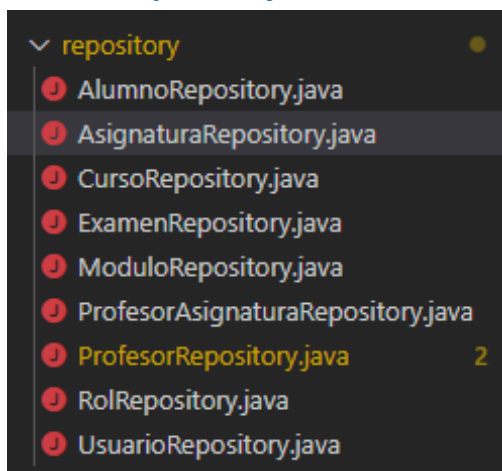

```

10
11 @Service
12 public class AlumnoServiceImpl implements AlumnoService {
13
14     @Autowired
15     AlumnoRepository alumnoRepository;
16
17     @Override
18     public List<Alumno> findAll() {
19         return (List<Alumno>) alumnoRepository.findAll();
20     }
21
22     @Override
23     public Alumno findAlumnoById(Integer id) {
24         return alumnoRepository.findById(id).get();
25     }
26
27     @Override
28     public List<Alumno> findAllByCurso(Integer id) {
29         return alumnoRepository.findAllByCurso(id);
30     }
31

```

No voy a poner la captura de todos los servicios ya que es bastante repetitivo y se puede ver en la primera captura de este apartado, son muchas interfaces y clases. No hay nada relevante respecto a esta captura.

4.5 Repository



4.5.1 AlumnoRepository

```

public interface AlumnoRepository extends CrudRepository<Alumno, Integer> {
    @Query("select a from Alumno a where a.curso.id = ?1")
    List<Alumno> findAllByCurso(Integer id);
}

```

Tenemos el repositorio de Alumno que extiende de CRUD, que viene de Create, Read, Update y Delete que hace referencia a la capa de persistencia. Esto permite que

podamos guardar en nuestra base de datos objetos, pasar un objeto a datos mediante la serialización y viceversa. En el CRUD tenemos las consultas básica a la base de datos, en caso de querer hacer algo diferente a lo que este no proporciona tenemos que hacerlo una custom query como en el caso de alumno. Como se puede ver no es una query común de mysql sino que utiliza los objetos.

En este caso seleccionamos los alumnos con un curso en especifico

4.5.2 AsignaturaRepository

```
public interface AsignaturaRepository extends CrudRepository<Asignatura, Integer> {
    @Query("select a from Asignatura a where a.curso.id = ?1")
    List<Asignatura> findAllByCurso(Integer id);
}
```

Se seleccionan las asignaturas de un curso

4.5.3 CursoRepository

```
public interface CursoRepository extends CrudRepository<Curso, Integer> {
    @Query("select c from Curso c where c.modulo.id = ?1")
    List<Curso> getCursosByModulo(Integer idModulo);
}
```

Seleccionamos los cursos que pertenecen a un modulo

4.5.4 ExamenRepository

```
public interface ExamenRepository extends CrudRepository<Examen, Integer> {
    @Query("select e from Examen e where e.asignatura.id = ?1")
    List<Examen> findExamenesByAsignatura(Integer idAsignatura);
}
```

Seleccionamos los exámenes de una asignatura

4.5.5 ModuloRepository

```
public interface ModuloRepository extends CrudRepository<Modulo, Integer> {
}
```

En este caso con el CRUD tenemos suficiente por lo cual, no tenemos que implementar ninguna búsqueda adicional

4.5.6 ProfesorAsignaturaRepository

```
public interface ProfesorAsignaturaRepository extends CrudRepository<ProfesorAsignatura, ProfesorAsignaturaPK> {
}
```

4.5.7 ProfesorRepository

```
12
13 public interface ProfesorRepository extends CrudRepository<Profesor, Integer> {
14     @Query("select c.modulo from Curso c where c.id in (select pa.curso from ProfesorAsignatura pa where pa.profesor.id = ?1)")
15     List<Modulo> findModulosByProfesor(Integer idProfesor);
16 }
17
```

Aquí seleccionamos los módulos a los que imparte un profesor, por lo que tenemos que hacer dos selects ya que no tenemos toda la información en una única tabla

4.5.8 RolRepository

```
7
8 public interface RolRepository extends CrudRepository<Rol, Integer> {
9     @Query("select r from Rol r where r.nombre = ?1")
10     Rol findRolByName(String nombre);
11 }
12
```

4.5.9 UsuarioRepository

```
8 public interface UsuarioRepository extends CrudRepository<Usuario, String> {
9     @Query("select u from Usuario u where u.nombre = ?1")
10     public Usuario findByName(String nombre);
11 }
12
```

4.6 Configuration

```
@Configuration
public class backendWebConfiguration {

    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {

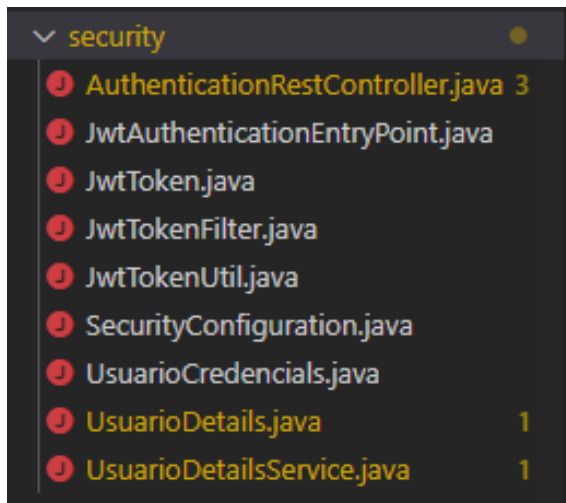
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                WebMvcConfigurer.super.addCorsMappings(registry);
                registry.addMapping(pathPattern: "**")
                    .allowedMethods(...methods: "GET", "POST", "PUT", "DELETE", "OPTIONS")
                    .allowedOrigins(...origins: "**");
            }

        };
    }

}
```

Esto es la configuración Web para habilitar el CORS, que es el intercambio de recursos de origen cruzado, que lo que viene a decir es que restringe las solicitudes HTTP desde un dominio diferente, como es en nuestro caso, peticiones desde el front al back y viene deshabilitado por defecto. Aquí habilitamos el CORS y los métodos que se ven a continuación, de cualquier origen.

4.7 Security



4.7.1 AuthenticationRestController

```
@RestController
@RequestMapping("/api")
public class AuthenticationRestController {

    @Autowired
    AuthenticationManager authenticationManager;

    @Autowired
    JwtTokenUtil jwtTokenUtil;

    @Autowired
    UserDetailsService userDetailsService;

    @Autowired
    PasswordEncoder encoder;

    @Autowired
    JwtToken jwtToken;

    @PostMapping("/login")
    public JwtToken login(@RequestBody UsuarioCredenciales usuarioCredenciales) {

        authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                usuarioCredenciales.getUsername(), usuarioCredenciales.getPassword()
            )
        );

        UserDetails userDetails = userDetailsService.loadUserByUsername(usuarioCredenciales.getUsername());
        String token = jwtTokenUtil.generateToken(userDetails);

        return new JwtToken(token, usuarioCredenciales.getUsername());
    }
}
```

Esto es el endpoint al que enviamos el usuario y contraseña, verifica si estos son correctos y le asigna al usuario un token.

4.7.2 JwtAuthenticationEntryPoint

```
@Component
public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint {
    @Override
    public void commence(HttpServletRequest request, HttpServletResponse response,
        AuthenticationException authException) throws IOException, ServletException {
        response.sendError(HttpServletResponse.SC_UNAUTHORIZED, authException.getMessage());
    }
}
```

Es es la clase que se encarga de verificar si el usuario tiene token y sino devuelve un 401 unauthorized.

4.7.3 JwtToken

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class JwtToken implements Serializable {
    private static final long serialVersionUID = 5615608530797516602L;

    private String token;

    private String username;
}
```

Esta es la clase que vamos a utilizar para almacenar el token que nos asignen y el usuario

4.7.4 JwtTokenUtil

```

15
16 @Component
17 public class JwtTokenUtil {
18
19     public static final Integer TOKEN_EXPIRATION = 2 * 60 * 60 * 1000;
20
21     @Value("${jwt.secret:mysecret}")
22     private String secret;
23
24     public String getUsernameFromToken(String token) {
25         return getClaimFromToken(token, Claims::getSubject);
26     }
27
28     public Date getExpirationDateFromToken(String token) {
29         return getClaimFromToken(token, Claims::getExpiration);
30     }
31
32     private Boolean isTokenExpired(String token) {
33         Date expiration = getExpirationDateFromToken(token);
34         return expiration.before(new Date());
35     }
36
37     private Claims getAllClaimsFromToken(String token) {
38         return Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody();
39     }
40
41     public <T> T getClaimFromToken(String token, Function<Claims, T> claimsResolver) {
42         Claims claims = getAllClaimsFromToken(token);
43         return claimsResolver.apply(claims);
44     }
45
46     public String generateToken(UserDetails userDetails) {
47         Map<String, Object> claims = new HashMap<>();
48         return Jwts.builder()
49             .setClaims(claims)
50             .setSubject(userDetails.getUsername())
51             .setIssuedAt(new Date())
52             .setExpiration(new Date(System.currentTimeMillis() + TOKEN_EXPIRATION))
53             .signWith(SignatureAlgorithm.HS512, secret)
54             .compact();
55     }
56
57     public Boolean validateToken(String token, UserDetails userDetails) {
58         String username = getUsernameFromToken(token);
59         return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
60     }
61 }

```

Esta es la clase que genera el token y valida que este bien formado y no haya expirado.

4.7.5 JwtTokenFilter

```
@Component
public class JwtTokenFilter extends OncePerRequestFilter {

    @Autowired
    JwtTokenUtil jwtTokenUtil;

    @Autowired
    UsuarioDetailsService usuarioDetailsService;

    @Autowired
    JwtToken jwtToken;

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
        throws ServletException, IOException {

        final String authHeader = request.getHeader(HttpHeaders.AUTHORIZATION); //mira header
        if(authHeader == null || !authHeader.startsWith("Bearer ")) { // si no lleva token
            filterChain.doFilter(request, response);
            return;
        }
    }
```

```
        final String token = authHeader.split(" ")[1].trim(); //separa token de la peticion

        String username = jwtTokenUtil.getUsernameFromToken(token); //crea variable username con el user sacado del token

        UserDetails userDetails = usuarioDetailsService.loadUserByUsername(username); //crea userdetails con el nombre del usuario

        if(jwtTokenUtil.validateToken(token, userDetails)) {
            UsernamePasswordAuthenticationToken authenticationToken =
                new UsernamePasswordAuthenticationToken(userDetails, credentials: null, userDetails.getAuthorities()); //crea
            authenticationToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

            SecurityContextHolder.getContext().setAuthentication(authenticationToken);
        }

        jwtToken.setToken(token);
        jwtToken.setUsername(username);

        filterChain.doFilter(request, response);
    }
```

Es la clase que utiliza el JwtTokenUtil para comprobar que el token es valido y permitir el acceso al usuario.

4.7.6 SecurityConfiguration

```
20
21 @Configuration
22 @EnableWebSecurity
23 @EnableGlobalMethodSecurity(
24     securedEnabled = true,
25     jsr250Enabled = true,
26     prePostEnabled = true
27 )
28 public class SecurityConfiguration extends WebSecurityConfigurerAdapter {
29
30     @Autowired
31     private UsuarioDetailsService usuarioDetailsService;
32
33     @Autowired
34     JwtTokenFilter jwtTokenFilter;
35
36     @Autowired
37     JwtAuthenticationEntryPoint jwtAuthenticationEntryPoint;
38
39     @Bean
40     @Override
41     public AuthenticationManager authenticationManagerBean() throws Exception {
42         return super.authenticationManagerBean();
43     }
44
45     @Override
46     protected void configure(AuthenticationManagerBuilder auth) throws Exception {
47         auth.userDetailsService(usuarioDetailsService);
48     }
49 }
```

```

@Override
protected void configure(HttpSecurity http) throws Exception {
    //Habilitar CORS y deshabilitar CSRF
    http = http.cors().and().csrf().disable();

    //Establecer el Session Management a STATELESS
    http = http.sessionManagement()
        .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        .and();

    //Establecer un Exception Handler para los REQUESTS no permitidos
    http = http.exceptionHandling()
        .authenticationEntryPoint(
            jwtAuthenticationEntryPoint
        )
        .and();

    //Establecer permisos a los endpoints
    http.authorizeRequests()
        .antMatchers(HttpMethod.POST, ...antPatterns: "/api/login").permitAll();

    //Añadir el filtro para procesar el token JWT
    http.addFilterBefore(jwtTokenFilter, beforeFilter: UsernamePasswordAuthenticationFilter.class);
}

@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder(strength: 12);
}

```

```

@Bean
@Scope(value = WebApplicationContext.SCOPE_REQUEST, proxyMode = ScopedProxyMode.TARGET_CLASS)
public JwtToken jwtToken() {
    return new JwtToken();
}

```

En esta clase es donde tenemos la configuración de seguridad que extiende de `WebSecurityConfigurerAdapter` y este ya trae los métodos necesarios que tendremos que implementar en nuestra clase, como por ejemplo permitir solo la entrada a la aplicación por `/api/login` a los usuarios sin token, añadir el filtro que mencionamos anteriormente antes de la autenticación, de esta manera, si tienen token entran directamente, sin necesidad de volver a validar el usuario y contraseña.

4.7.7 UsuarioCredenciales

```
7  @Data
8  @AllArgsConstructor
9  @NoArgsConstructor
10 public class UsuarioCredenciales {
11     private String username;
12     private String password;
13 }
14
```

Esta es la clase para almacenar los datos del usuario

4.7.8 UsuarioDetails

```
10
17 @Slf4j
18 public class UsuarioDetails extends Usuario implements UserDetails{
19
20     private static final long serialVersionUID = -3443505686188951096L;
21
22
23     public UsuarioDetails() {}
24
25     public UsuarioDetails(Usuario usuario) {
26         super.setNombre(usuario.getNombre());
27         super.setPassword(usuario.getPassword());
28         super.setRol(usuario.getRol());
29     }
30
31
32     @Override
33     public String getPassword() {
34         return super.getPassword();
35     }
36
37     @Override
38     public String getUsername() {
39         return super.getNombre();
40     }
41
42     @Override
43     public boolean isAccountNonExpired() {
44         return true;
45     }
46
```

```

@Override
public boolean isAccountNonLocked() {
    return true;
}

@Override
public boolean isCredentialsNonExpired() {
    return true;
}

@Override
public boolean isEnabled() {
    return true;
}

@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    Set<Rol> roles = new HashSet<Rol>();
    roles.add(super.getRol());
    return roles.stream().map((rol) -> new SimpleGrantedAuthority(rol.getNombre())).collect(Collectors.toList());
}

```

Esta clase es la que enlaza a nuestro usuario con el userDetails que nos proporciona los datos necesarios que nos pide spring que tenga un usuario.

4.7.9 UsuarioDetailsService

```

13
14 @Service
15 public class UsuarioDetailsService implements UserDetailsService {
16
17     @Autowired
18     UsuarioRepository usuarioRepository;
19
20     @Override
21     public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
22         Usuario usuario = usuarioRepository.findById(username)
23             .orElseThrow(() -> new UsernameNotFoundException("El usuario " + username + " no se ha encontrado"));
24
25         UserDetails usuarioDetails = new UserDetails(usuario);
26
27         return usuarioDetails;
28     }
29 }
30

```

UsuarioDetailsService es la clase que se encarga de buscar al usuario con el nombre de usuario.

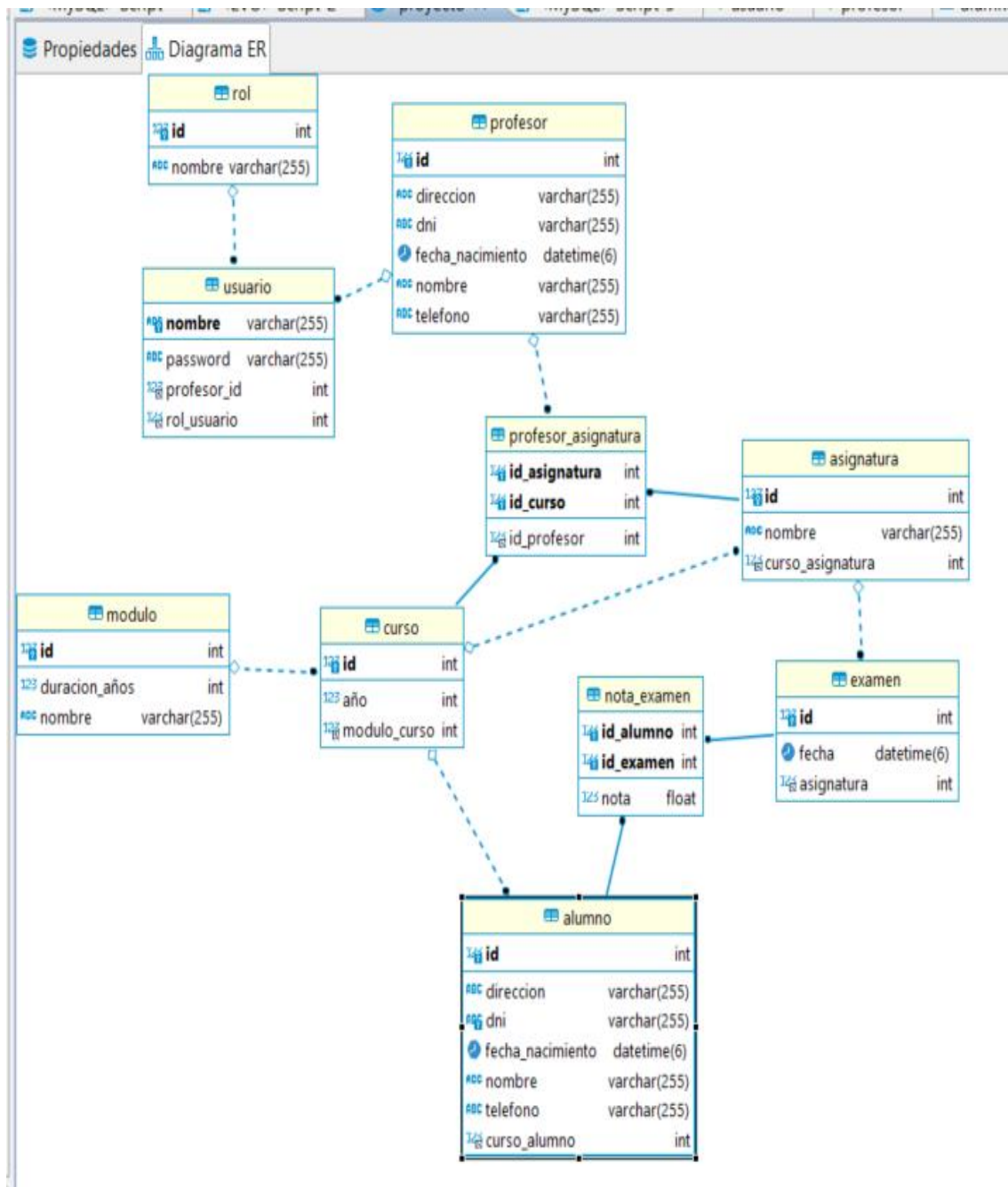
4.8 Application.properties

```
1  server.port=8081
2  spring.datasource.url=jdbc:mysql://localhost:3306/proyecto
3  spring.datasource.username=root
4  spring.datasource.password=admin
5
6  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
7  spring.jpa.hibernate.ddl-auto=create-drop
8  spring.jpa.defer-datasource-initialization=true
9  spring.sql.init.mode=always
10
11
12  jwt.secret = secret
13
```

En este archivo es donde se indican todas las propiedades que hacen referencia al funcionamiento de la aplicación, como el puerto en el que va a funcionar, la configuración de la base de datos, el driver...

4.02 Desarrollo – Base de datos

Vamos a ver como ha quedado la base de datos después de haber visto las anotaciones y las relaciones en la parte del back



Para cargar los datos mínimos para poder probar el funcionamiento de la base de datos he creado un archivo sql que tenemos que ejecutar en el DBeaver en mi caso o en un workbench por ejemplo.

```
⑥ INSERT INTO profesor (direccion,dni,fecha_nacimiento,nombre,telefono) VALUES
("Calle La Paz 13", "20040956L", "2011-12-28", "Jose Luis", "+34654345670"),
("Calle La Paz 13", "20040967L", "2011-12-28", "Mario", "+34654345670"),
("Calle La Paz 13", "20077956L", "2011-12-28", "Pepito", "+34654345670"),
("Calle La Paz 13", "20009956L", "2011-12-28", "Andres", "+34654345670");

⑥ INSERT INTO modulo (duracion_años, nombre) VALUES
(2, "DAM"),
(2, "DAW"),
(4, "ESO");

⑥ insert into rol (nombre) values
("administrador");

⑥ INSERT INTO curso (año, modulo_curso) VALUES
(1,1),
(2,1),
(1,2),
(2,2),
(1,3),
(2,3),
(3,3),
(4,3);

⑥ INSERT INTO asignatura (nombre, curso_asignatura) VALUES
("Programacion",1),
("Base de datos",1),
("FOL",1),
("Historia",5),
("EIE",3);
```



```

INSERT INTO alumno (direccion,dni,fecha_nacimiento,nombre,telefono,curso_alumno) values
-- Alumnos DAM año1 --
("Calle Ramon", "25550056L", "1980-12-28", "Jose", "+34654344570",1),
("Calle Ramon", "25224056L", "1980-12-28", "Paco", "+34654344570",1),
("Calle Ramon", "25553022L", "1980-12-28", "Pepe", "+34654344570",1),
("Calle Ramon", "25551056V", "1980-12-28", "Xavi", "+34654344570",1),
("Calle Ramon", "25559056L", "1980-12-28", "Ramon", "+34654344570",1),
-- Alumnos DAM año2 --
("Calle Ramon", "25550056A", "1980-12-28", "Jose", "+34654344570",2),
("Calle Ramon", "25553356L", "1980-12-28", "Paco", "+34654344570",2),
("Calle Ramon", "25554856L", "1980-12-28", "Pepe", "+34654344570",2),
-- Alumnos DAW año1 --
("Calle Ramon", "25550056M", "1980-12-28", "Felipe", "+34654344570",3),
("Calle Ramon", "25551056L", "1980-12-28", "Paco", "+34654344570",3),
("Calle Ramon", "25553056L", "1980-12-28", "Pepe", "+34654344570",3),
-- Alumnos DAW año2 --
("Calle Ramon", "25550056Z", "1980-12-28", "Jose", "+34654344570",4),
("Calle Ramon", "25554056L", "1980-12-28", "Paco", "+34654344570",4),
("Calle Ramon", "25551156L", "1980-12-28", "Pepe", "+34654344570",4),
-- Alumnos ESO año1 --
("Calle Ramon", "22550056L", "1980-12-28", "Jose", "+34654344570",5),
("Calle Ramon", "23554056L", "1980-12-28", "Paco", "+34654344570",5),
("Calle Ramon", "24553056L", "1980-12-28", "Pepe", "+34654344570",5),
-- Alumnos ESO año2 --
("Calle Ramon", "26550056L", "1980-12-28", "Jose", "+34654344570",6),
("Calle Ramon", "27554056L", "1980-12-28", "Paco", "+34654344570",6),
("Calle Ramon", "28553056L", "1980-12-28", "Pepe", "+34654344570",6),
-- Alumnos ESO año3 --
("Calle Ramon", "29550056L", "1980-12-28", "Jose", "+34654344570",7),
("Calle Ramon", "15554056L", "1980-12-28", "Paco", "+34654344570",7),
("Calle Ramon", "35553056L", "1980-12-28", "Pepe", "+34654344570",7),
-- Alumnos ESO año4 --
("Calle Ramon", "45550056L", "1980-12-28", "Jose", "+34654344570",8),
("Calle Ramon", "65554056L", "1980-12-28", "Paco", "+34654344570",8),
("Calle Ramon", "52553056L", "1980-12-28", "Pepe", "+34654344570",8);

```

```

insert into profesor_asignatura (id_asignatura,id_curso,id_profesor) VALUES
(1,1,1),
(5,5,2),
(4,3,1);

insert into usuario (nombre,password,profesor_id,rol_usuario) values
("admin", "$2a$12$5uf5z6MEXkN8ptCVVAgYs.LKNN51JKpJVhVCiDKvhG2mexNNk71PW",1,1),
("admin2", "$2a$12$5uf5z6MEXkN8ptCVVAgYs.LKNN51JKpJVhVCiDKvhG2mexNNk71PW",2,1);

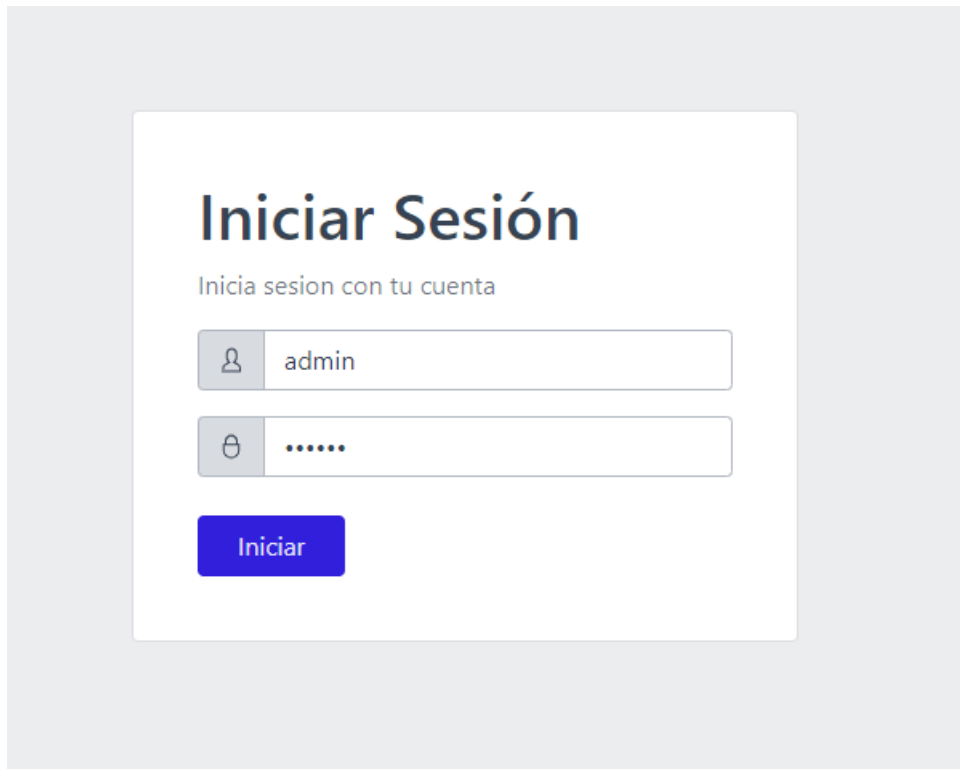
```

El campo password es la contraseña encriptada con bcrypt con una fuerza de 12 rondas.

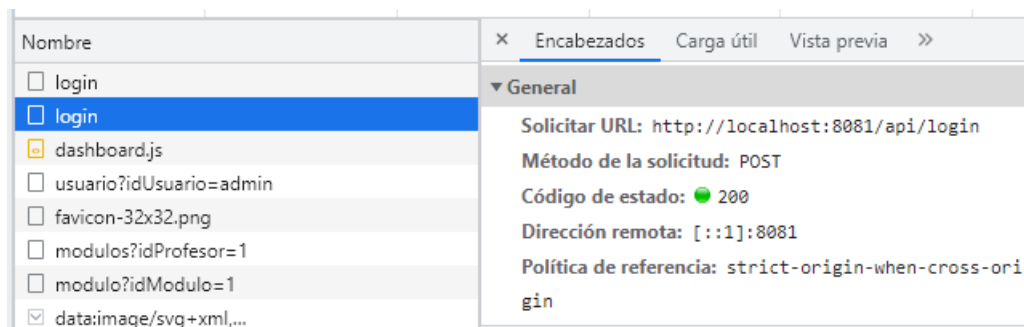
4.03 Desarrollo – Front

Hemos visto una parte del front en apartados anteriores, pero vamos a ver lo que nos queda.

Para empezar, tenemos la página de Login



Al darle a Iniciar, hace una petición al back



Devuelve un 200 OK y la respuesta:

```
▼ {, ...}
  token: "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbGliImV4cCI
  username: "admin"
```

[Home](#) / [Dashboard](#)

MODULOS:

DAM



DAW



El usuario admin es el usuario del profesor 1, el cual imparte a DAM y DAW

MODULOS:

DAM

año 1

año 2

En el desplegable vemos los 2 años de DAM, si entramos en el año 1:

[Home](#) / [Curso](#)

Curso 1 de DAM

ID	Nombre	Modulo	Curso
1	Jose	DAM	1
2	Paco	DAM	1
3	Pepe	DAM	1
4	Xavi	DAM	1
5	Ramon	DAM	1

C2 – Uso Restringido

Nos carga todos los alumnos de este curso, si entramos a ver al usuario 1 por ejemplo:

Vemos los datos detallados



La idea es que aquí aparezcan todas las asignaturas del alumno con sus respectivas medias

4.1 Login

```
<template>
  <div class="bg-light min-vh-100 d-flex flex-row align-items-center">
    <CAlert :color="alertType" :show="alertShow" fade>
      {{ alertInfo }}
    </CAlert>
    <CContainer>
      <CRow class="justify-content-center">
        <CCol :md="4">
          <CCardGroup>
            <CCard class="p-4">
              <CCardBody>
                <CForm>
                  <h1>Iniciar Sesión</h1>
                  <p class="text-medium-emphasis">
                    Inicia sesion con tu cuenta
                  </p>
                  <CInputGroup class="mb-3">
                    <CInputGroupText>
                      <CIcon icon="cil-user" />
                    </CInputGroupText>
                    <CFormInput
                      placeholder="Usuario"
                      autocomplete="usuario"
                      v-model="username"
                    />
                  </CInputGroup>
                  <CInputGroup class="mb-4">
                    <CInputGroupText>
                      <CIcon icon="cil-lock-locked" />
                    </CInputGroupText>
                    <CFormInput
                      type="password"
                      placeholder="Password"
                      autocomplete="current-password"
                      v-model="password"
                    />
                  </CInputGroup>
                </CForm>
              </CCardBody>
            </CCard>
          </CCardGroup>
        </CCol>
      </CRow>
    </CContainer>
  </div>
</template>
```

```
37         </CInputGroup>
38         <CRow>
39             <CCol :xs="6">
40                 <CButton
41                     @click="fetchLogin()"
42                     color="primary"
43                     class="px-4"
44                     :disabled="!disabledButton"
45                 >
46                     Iniciar
47                 </CButton>
48             </CCol>
49         </CRow>
50     </CForm>
51 </CCardBody>
52 </CCard>
53 </CCardGroup>
54 </CCol>
55 </CRow>
56 </CContainer>
57 </div>
58 </template>
59
```

```
<script>
import Config from '../../config.js'
export default {
  name: 'Login',
  data() {
    return {
      username: '',
      password: '',
      validUsername: false,
      validPassword: false,
      alertShow: false,
      alertType: 'primary',
      alertInfo: '',
    }
  },
  methods: {
    updateAlert({ message, type = 'primary', timeout = 3000 }) {
      this.alertShow = timeout
      this.alertType = type
      this.alertInfo = message
    },
    fetchLogin() {
      const response = fetch(`${Config.BASE_PATH}${Config.API_PATH}/login`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          username: this.username,
          password: this.password,
        }),
      }),
    }
  }
}
```

```
90     response
91     .then((res) => {
92       if (res.ok) {
93         let body = res.json()
94         body
95         .then((res) => {
96           localStorage.setItem('userToken', res.token)
97           localStorage.setItem('username', res.username)
98           //localStorage.setItem('role', body.role)
99           this.$router.push('/')
100         })
101       }
102       .catch((err) => {
103         this.updateAlert({
104           message: `Ha ocurrido un error: ${err}`,
105           type: 'danger',
106         })
107       })
108     } else {
109       this.updateAlert({
110         message: `Ha ocurrido un error`,
111         type: 'danger',
112         timeout: 10000,
113       })
114     })
115     .catch((err) => {
116       this.updateAlert({
117         message: `Ha ocurrido un error: ${err}`,
118         type: 'danger',
119         timeout: 10000,
120       })
121     })
122   },
123 },
```

```
124   computed: {
125     disabledButton: function () {
126       return Boolean(this.username && this.password)
127     },
128   },
129   mounted() {},
130 }
131 </script>
132
```

En esta parte, cogemos los datos introducidos en usuario y contraseña y hacemos una petición al back a /api/login y si la respuesta es 200 OK, guardamos el usuario y el token en localStorage del navegador.

4.2 Dashboard

```
1  <template>
2    <div v-if="this.modulos">
3      MODULOS:
4      <CAccordion>
5        <CAccordionItem
6          v-for="modulo in modulos"
7          :item-key="modulo.id"
8          :key="modulo.id"
9        >
10         <CAccordionHeader> {{ modulo.nombre }} </CAccordionHeader>
11         <CAccordionBody>
12           <CButton
13             class="d-grid gap-2 d-md-flex justify-content-md-end"
14             id="boton"
15             color="info"
16             v-for="curso in modulo.cursos"
17             :key="curso.id"
18           >
19             <router-link
20               :to="{
21                 path: '/curso',
22                 name: 'Curso',
23                 query: {
24                   nombreModulo: modulo.nombre,
25                   añoCurso: curso.año,
26                 },
27                 params: { idCurso: curso.id },
28               }"
29             >año {{ curso.año }}</router-link>
30           </CButton>
31         </CAccordionBody>
32       </CAccordionItem>
33     </CAccordion>
34   </div>
35 </template>
```

```
38 <script>
39 export default {
40   name: 'Dashboard',
41   components: {},
42   data() {
43     return {
44       modulos: [],
45       usuario: null,
46     }
47   },
48   mounted() {
49     this.getProfesorUsuario()
50   },
51   methods: {
52     async getProfesorUsuario() {
53       console.log('hacemos un get del profesor asignado a este usuario')
54       await fetch(
55         `http://localhost:8081/api/usuario?idUsuario=${localStorage.getItem(
56           'username',
57         )}`,
58       )
59       .then((response) => response.json())
60       .then((response) => (this.usuario = response))
61       .then(() => this.getModulos())
62     },
63     async getModulos() {
64       console.log('hacemos un get de modulos')
65       await fetch(
66         `http://localhost:8081/api/profesor/modulos?idProfesor=${this.usuario.profesor.id}`,
67       )
68       .then((response) => response.json())
69       .then((response) => (this.modulos = response))
70       .then(() => this.saveCursos())
71     },

```



```
72     async saveCursos() {
73         for (let i = 0; i < this.modulos.length; i++) {
74             const data = await fetch(
75                 `http://localhost:8081/api/cursos/modulo?idModulo=${this.modulos[i].id}`,
76             )
77             this.modulos[i].cursos = await data.json()
78         }
79     },
80 },
81 }
82 </script>
83
84 <style scoped>
85 #boton {
86     margin-top: 10px;
87     border: 1px solid #556b2f;
88     background-color: #d9ead3;
89     color: #556b2f;
90 }
91 </style>
92
```

En la primera ventana, una vez entramos a la aplicación, se hace una petición al back con el nombre de usuario para obtener el objeto entero del usuario, a partir de este obtenemos el profesor y hacemos una petición para sacar los módulos a los que da clase el profesor y los cursos.

4.3 Curso

```
1 <template>
2   <div v-if="this.alumnos">
3     <h1>
4       Curso {{ $route.query.añoCurso }} de {{ $route.query.nombreModulo }}
5     </h1>
6     <CTable striped>
7       <CTableHead>
8         <CTableRow color="dark">
9           <CTableHeaderCell scope="col">ID</CTableHeaderCell>
10          <CTableHeaderCell scope="col">Nombre</CTableHeaderCell>
11          <CTableHeaderCell scope="col">Modulo</CTableHeaderCell>
12          <CTableHeaderCell scope="col">Curso</CTableHeaderCell>
13        </CTableRow>
14      </CTableHead>
15      <CTableBody>
16        <CTableRow color="info" v-for="alumno in alumnos" :key="alumno.id">
17          <CTableHeaderCell scope="row"
18            ><router-link
19              :to="{
20                path: '/alumno',
21                name: 'Alumno',
22                params: { idAlumno: alumno.id },
23              }"
24            >
25              {{ alumno.id }}
26            </router-link></CTableHeaderCell>
27          <CTableDataCell>{{ alumno.nombre }}</CTableDataCell>
28          <CTableDataCell>{{ $route.query.nombreModulo }}</CTableDataCell>
29          <CTableDataCell>{{ $route.query.añoCurso }}</CTableDataCell>
30        </CTableRow>
31      </CTableBody>
32    </CTable>
33  </div>
34 </template>
```

```
37 <script>
38 export default {
39   name: 'Curso',
40   data() {
41     return {
42       alumnos: [],
43     }
44   },
45   mounted() {
46     this.getAlumnos()
47   },
48   methods: {
49     async getAlumnos() {
50       console.log('hacemos un get de alumnos')
51       await fetch(
52         `http://localhost:8081/api/alumnos/curso?idCurso=${this.$route.params.idCurso}`,
53       )
54       .then((response) => response.json())
55       .then((response) => (this.alumnos = response))
56     },
57   },
58 }
59 </script>
60
```

En esta ventana recibimos por parámetros del router link, el nombre del modulo y el id del curso y su año

Con el curso obtenemos todos los alumnos y con un for rellenamos toda la tabla con sus datos.

4.4 Alumno

```
1  <template>
2    <div v-if="loaded">
3      <h1 id="nombre">{{ alumno.nombre }}</h1>
4      <CTable color="success" striped>
5        <CTableHead>
6          <CTableRow>
7            <CTableHeaderCell scope="col">ID</CTableHeaderCell>
8            <CTableHeaderCell scope="col">Dni</CTableHeaderCell>
9            <CTableHeaderCell scope="col">Dirección</CTableHeaderCell>
10           <CTableHeaderCell scope="col">Teléfono</CTableHeaderCell>
11           <CTableHeaderCell scope="col">Fecha Nacimiento</CTableHeaderCell>
12         </CTableRow>
13       </CTableHead>
14       <CTableBody>
15         <CTableRow>
16           <CTableHeaderCell scope="row">{{ alumno.id }}</CTableHeaderCell>
17           <CTableDataCell>{{ alumno.dni }}</CTableDataCell>
18           <CTableDataCell>{{ alumno.direccion }}</CTableDataCell>
19           <CTableDataCell>{{ alumno.telefono }}</CTableDataCell>
20           <CTableDataCell>
21             <div id="fecha">{{ alumno.fechaNacimiento }}</div></CTableDataCell>
22           </CTableDataCell>
23         </CTableRow>
24       </CTableBody>
25     </CTable>
26   </div>
27 </template>
```

```
26     <div>
27       <CChart
28         type="radar"
29         :data="{
30           labels: [
31             'Programación',
32             'Base de Datos',
33             'Sistemas informáticos',
34             'Lenguajes de marcas',
35             'Entornos Desarrollo',
36             'FOL',
37           ],
38           datasets: [
39             {
40               label: 'NotasMedias',
41               backgroundColor: 'rgba(220, 220, 220, 0.2)',
42               borderColor: 'rgba(150, 220, 220, 1)',
43               pointBackgroundColor: 'rgba(220, 220, 220, 1)',
44               pointBorderColor: '#fff',
45               pointHighlightFill: '#fff',
46               pointHighlightStroke: 'rgba(150, 220, 220, 1)',
47               data: [8, 4, 5, 9, 7, 10],
48             },
49           ],
50         }"
51       />
52     </div>
53   </div>
54 </template>
```

```

56 <script>
57 export default {
58   name: 'Alumno',
59   data() {
60     return {
61       loaded: false,
62       alumno: '',
63       asignaturas: [],
64     }
65   },
66   mounted() {
67     this.getAlumno()
68   },
69   methods: {
70     async getAlumno() {
71       console.log('hacemos un get de alumno')
72       await fetch(
73         `http://localhost:8081/api/alumno?idAlumno=${this.$route.params.idAlumno}`,
74       )
75       .then((response) => response.json())
76       .then((response) => (this.alumno = response))
77       .catch((error) => console.error(error))
78       this.getAsignaturas()
79     },
80     async getAsignaturas() {
81       console.log('hacemos un get de asignaturas')
82       await fetch(
83         `http://localhost:8081/api/asignaturas/curso?idCurso=${this.alumno.curso.id}`,
84       )
85       .then((response) => response.json())
86       .then((response) => (this.asignaturas = response))
87       .then(() => (this.loaded = true))
88     },

```

```

89   },
90 }
91 </script>
92 <style scoped>
93 #fecha {
94   text-align: center;
95   width: 9.5ch;
96   overflow: hidden;
97   white-space: nowrap;
98 }
99 #nombre {
100   text-align: center;
101 }
102 </style>
103

```

En esta última parte del proyecto ya damos información mas detallada del alumno junto con las asignaturas y las notas medias de estas.

No me ha dado tiempo a hacer esta parte dinámica como el resto, he añadido la grafica fija para que se pudiera ver cual era realmente la idea.