UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

MÁSTER UNIVERSITARIO EN DESARROLLO ÁGIL DE SOFTWARE PARA LA WEB

Trabajo Fin de Máster

DESARROLLO WEB CON SPRING BOOT Y VAADIN

Luis Alberto Cobo Salgado

2021

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

MÁSTER UNIVERSITARIO EN DESARROLLO ÁGIL DE SOFTWARE PARA LA WEB

Trabajo Fin de Máster

DESARROLLO WEB CON SPRING BOOT Y VAADIN

Autor: **Luis Alberto Cobo Salgado**

Director: **Salvador Otón Tortosa**

Tribunal:

Presidente: .....

Vocal 1º: ..... Vocal 2º: .....

Calificación: .....

Fecha: ..... de ..... de .....

En agradecimiento a mis padres, esposa y especialmente a mis abuelos, porque gran parte de la persona que soy se lo debo a ellos, gracias y mil gracias por todo lo que aportaron y aportan a mi vida.

ÍNDICE RESUMIDO

ÍNDICE DETALLADO

ÍNDICE DE ILUSTRACIONES

1. INTRODUCCIÓN

En la actualidad existen diferentes Frameworks para el desarrollo de aplicaciones web en la parte del Frontend, pero en su mayoría basan su desarrollo e implementación en tecnologías como JavaScript y HTML. Por lo tanto, para utilizarlos sería necesario tener conocimientos en dichas tecnologías , pero también existen otros Frameworks como Vaadin que permiten crear una aplicación web sin tener dichos conocimientos ya que permite implementarlas solo con el lenguaje de programación Java que se utiliza principalmente en el desarrollo de la parte Backend.

Por lo tanto, todos los desarrolladores que tienen conocimientos en Java también pueden desarrollar la parte visual de las aplicaciones web haciendo uso de Vaadin y todos los componentes web de los que dispone.

2. OBJETIVOS DEL PROYECTO

El objetivo principal consiste en conocer el funcionamiento del Framework Vaadin con Spring Boot, para poder construir la capa visual de una aplicación web sin necesidad de utilizar JavaScript y HTML, solo con conocimientos de Java.

Se desarrollarán las funcionalidades de una típica aplicación empresarial, es decir, inicio de sesión de usuarios con diferentes roles, registros de usuarios, visualización de datos en tablas, alta, modificación, eliminación de datos existentes y también se incluirá gráficos estadísticos que se pueden utilizar como cuadro de mandos o KPIs.

El enfoque será a modo de tutorial, es decir se explicará desde la creación de la aplicación hasta la configuración de cada componente visual de Vaadin utilizado.

Para demostrar lo anteriormente dicho se va a desarrollar una aplicación de gestión de cursos, alumnos y matrículas.

3. ESTADO DEL ARTE

El desarrollo de aplicaciones web en la actualidad ha avanzado mucho y existen diferentes tecnologías y frameworks que permiten y facilitan el trabajo del desarrollador. El desarrollo web está dividido en dos partes, el Frontend y el Backend.

Ilustración 1. Partes y tecnologías de una aplicación web

- Frontend es la capa visual con la que interactúa el usuario y a la cual accede directamente porque se ejecuta en el navegador web por eso también se dice que se ejecuta en el lado cliente.

  Las principales tecnologías para el desarrollo en el Frontend son HMTL y CSS para definir la estructura y los estilos. JavaScript es la tecnología que se encarga de la lógica de la página y la comunicación con el Backend, es decir recibir las peticiones del usuario, procesarlas, enviarlas al Backend y procesar también la respuesta de este.

  Existen varias librerías o Framework basados en JavaScript que implementa determinados patrones y directrices de diseño para agilizar la implementación de la lógica, cabe destacar las tres principales que son Angular, React y Vue.

- Backend es la capa que se encarga de acceder y gestionar los datos, así como su lógica. Por lo tanto, no es accesible directamente por el usuario, esta parte se ejecuta en el lado del servidor.

  Los principales lenguajes con los que se puede desarrollar son Java, Node.js, PHP, Ruby y C#. También existen numerosos Frameworks que permiten mejorar tiempos y productividad de desarrollo como lo son Spring, Laravel, ASP.Net, Express.js, etc.

Existen también Frameworks que permiten la creación de aplicaciones web incluida la parte del Frontend utilizando puramente un lenguaje del lado servidor, es decir no es necesario escribir código en HTML y JavaScritps. Aquí se encuentra Vaadin Flow, que permite crear aplicaciones web utilizando solamente Java para desarrollar el Frontend y el Backend.

Vaadin traduce el código Java a JavaScript y HTML y lo presenta directamente en el lado cliente.

4. REQUISITOS Y DISEÑO

La aplicación por desarrollar consistirá en una aplicación para gestionar los cursos, alumnos y matrículas de una escuela.

Para ello se tendrán los siguientes requisitos basados en tres roles de usuarios.

- **Rol administrador**, será el encargado de gestionar la información y datos existentes en la aplicación y cuenta con las siguientes funcionalidades.
  - Visualizar, dar de alta, modificar y eliminar los cursos.
  - Visualizar, modificar y eliminar los alumnos.
  - Visualizar, dar de alta, modificar y eliminar los usuarios.
  - Visualizar estadísticas de cursos, usuarios y matriculas.

- **Rol profesor**, no tiene permisos para gestionar datos ni información de la aplicación, contará con las siguientes funcionalidades.
  - Visualizar los cursos existentes.
  - Visualizar los alumnos matriculados en los cursos que imparte.
  - Visualizar solo las estadísticas con los cursos que imparte.

- **Rol alumno**, es el que menos permisos tiene y solo podrá interactuar con los cursos, teniendo las siguientes funcionalidades.
  - Registrarse en la aplicación.
  - Visualizar los cursos existentes, matricularse en uno de ellos y eliminar también una matrícula.

En cuanto al diseño se implementarán las siguientes páginas y/o vistas.

  - Página de registro de usuario y login.
  - Página para gestionar los cursos.
  - Página para gestionar los alumnos.
  - Página para gestionar los usuarios.
  - Página para visualizar estadísticas.

Se visualizará en un menú lateral las distintas opciones de páginas a las que se puede acceder dependiendo del rol del usuario.

5. VAADIN FLOW

Vaadin Flow es un Framework Web open-source para crear aplicaciones web modernas en Java que se ejecutan en el servidor, abordando tanto la capa del Frontend y el Backend.

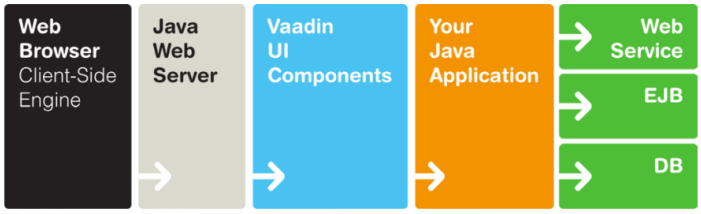


Ilustración 2. Arquitectura de aplicación Vaadin

Las características principales de Vaadin Flow son:

- Un conjunto de componentes de IU cuidadosamente diseñados que se enfocan tanto en la experiencia del usuario final como del desarrollador, por lo tanto, no es necesario tener conocimiento de HTML o JavaScript para poder utilizar dichos componentes en la aplicación a desarrollar.
- Poderosas capas de abstracción para crear sus propios componentes de interfaz de usuario reutilizables con plantillas Java.
- API de enlace de datos para conectar los componentes de la interfaz de usuario a cualquier Backend utilizando Java.
- API de enrutador para crear estructuras de página jerárquicas para que el usuario navegue.

5.1. Componentes disponibles en Vaadin.

Los componentes de Vaadin se basan en los estándares de componentes web. Ofrecen rendimiento nativo del navegador, vienen con un aspecto moderno y se pueden utilizar y configurar fácilmente.

Los componentes gratuitos tienen licencia Apache 2.0 y con licencia comercial. Todos los componentes de código abierto están en GitHub.

Vaadin cuenta con los siguientes componentes oficiales, algunos de ellos solo se pueden utilizar si se tiene una licencia de pago.

- **Entrada de datos**.

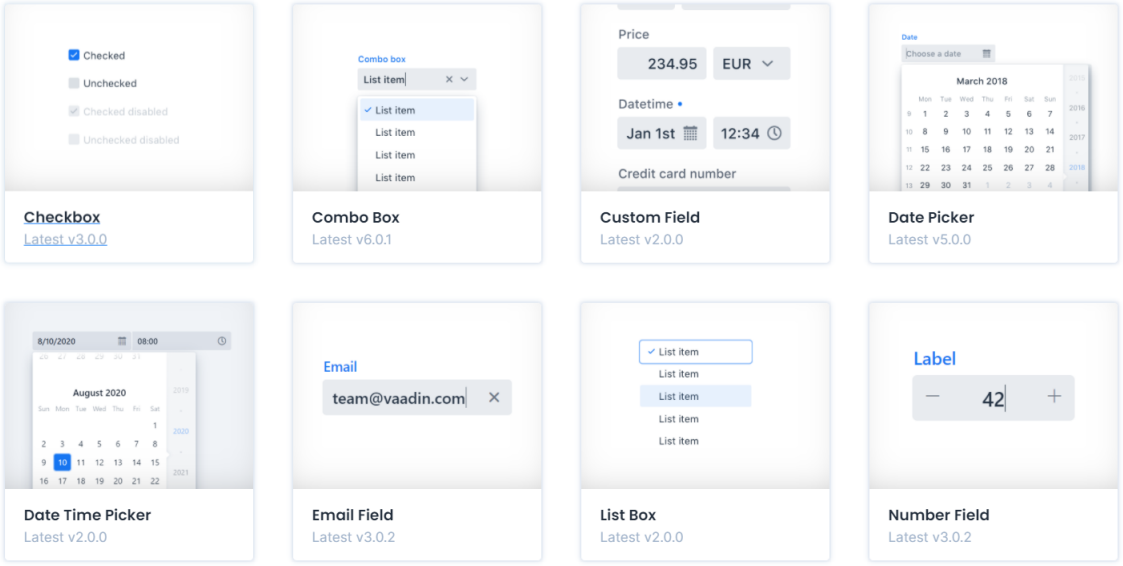
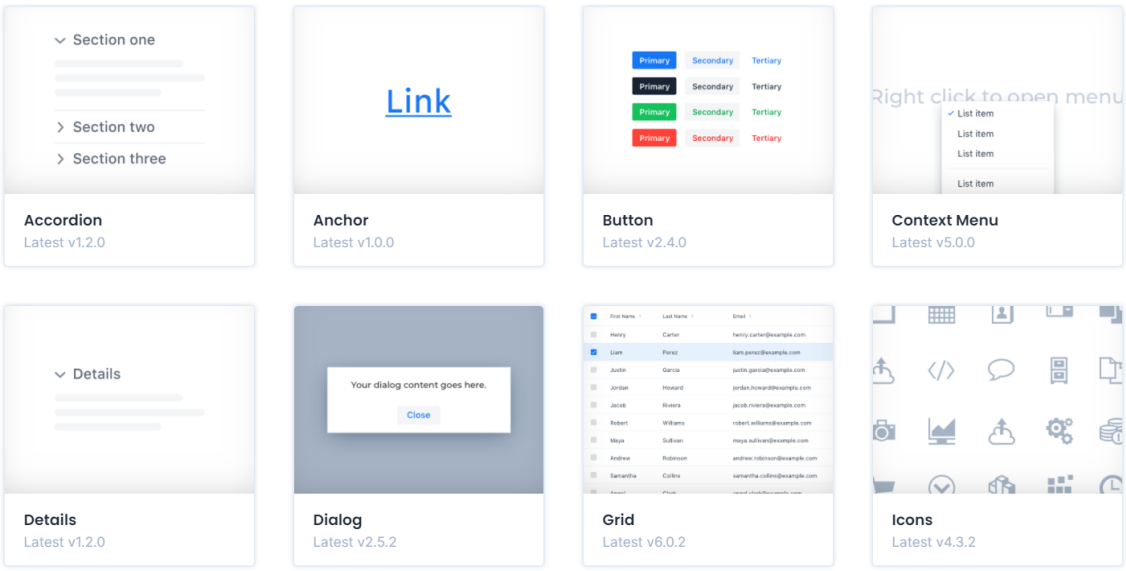


Ilustración 3. Componentes de entrada datos

- **Visualización e interacción.**



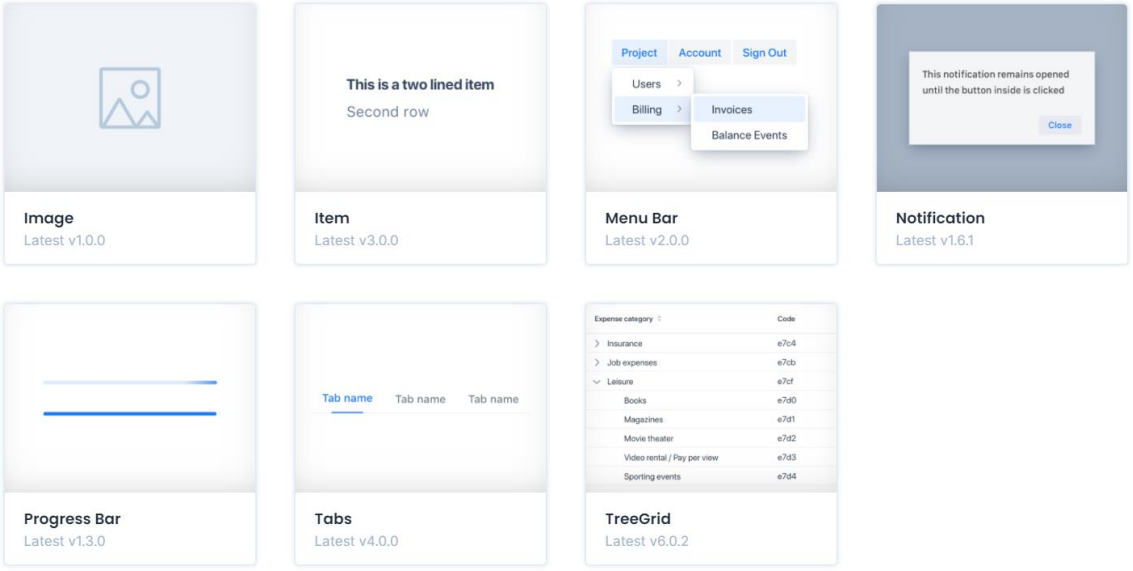


Ilustración 4. Componentes de visualización e interacción Vaadin

- **Layouts.**

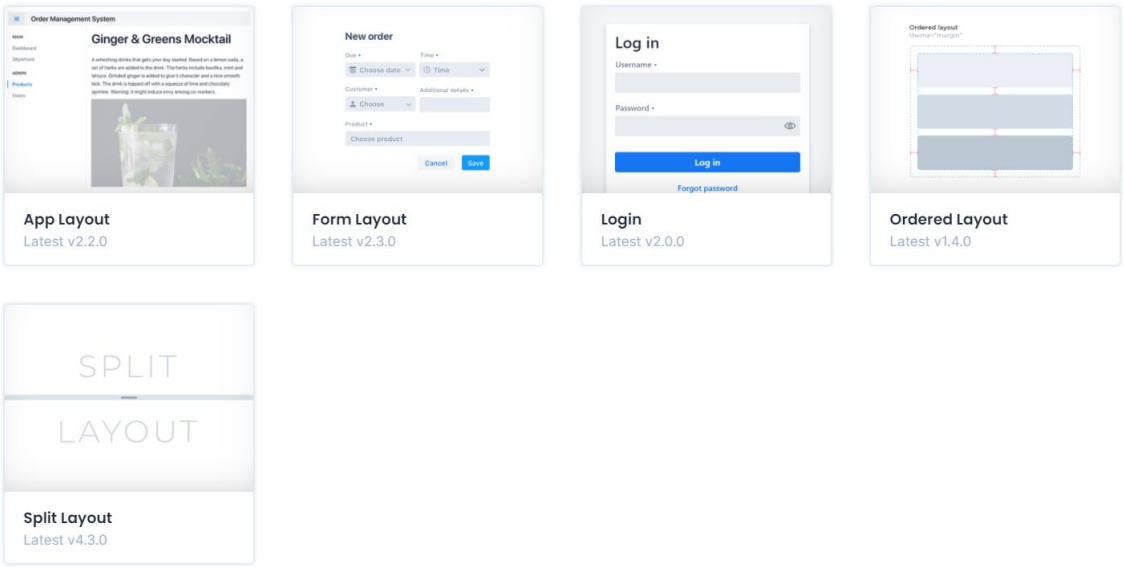


Ilustración 5. Componentes Layouts Vaadin

- **Componentes PRO (versión de pago).**

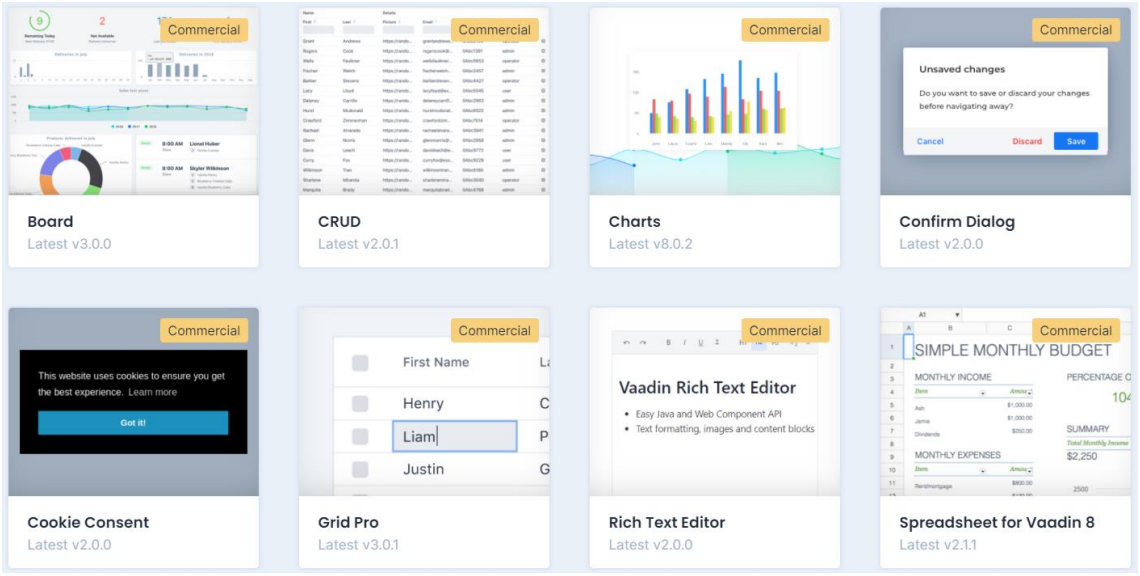


Ilustración 6. Componentes PRO Vaadin

Existe también componentes no oficiales los cuales son desarrollados por terceros.

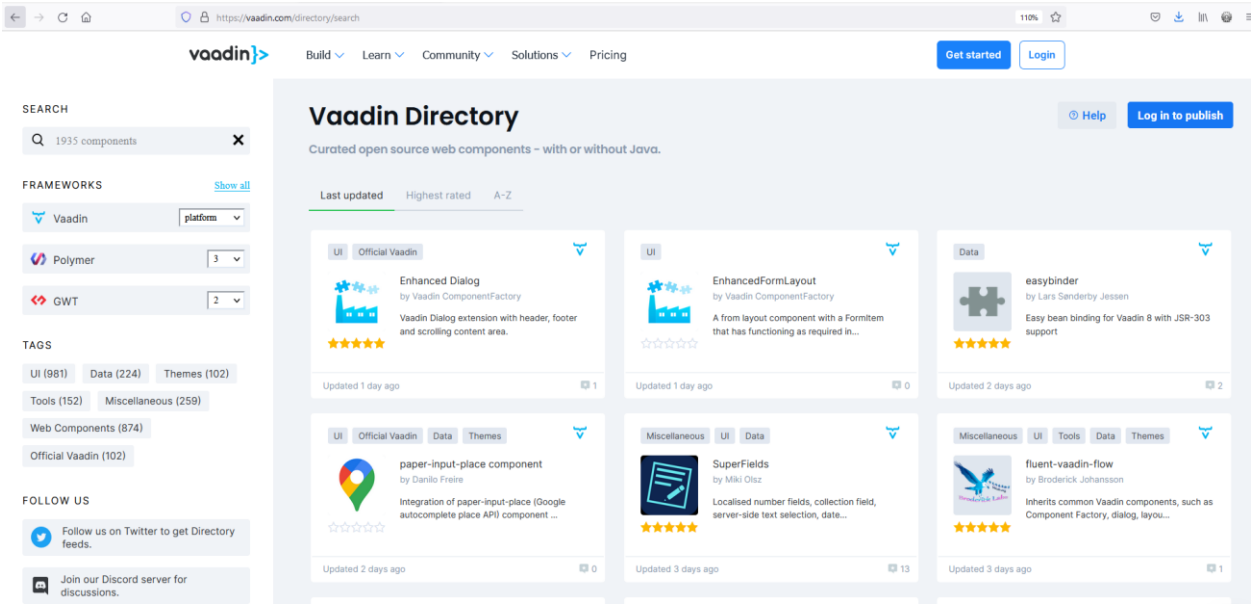


Ilustración 7. Componentes de terceros Vaadin

En cuanto a la documentación e información de Vaadin, en la página oficial se pueden encontrar diferentes tutoriales y guías para implementación de funcionalidades especificas hasta prácticamente diseñar una aplicación completa con todos los aspectos necesarios para que sea funcional y se pueda desplegar en un entorno real de producción.

Dicha documentación se puede encontrar clasificada por versión de Vaadin.

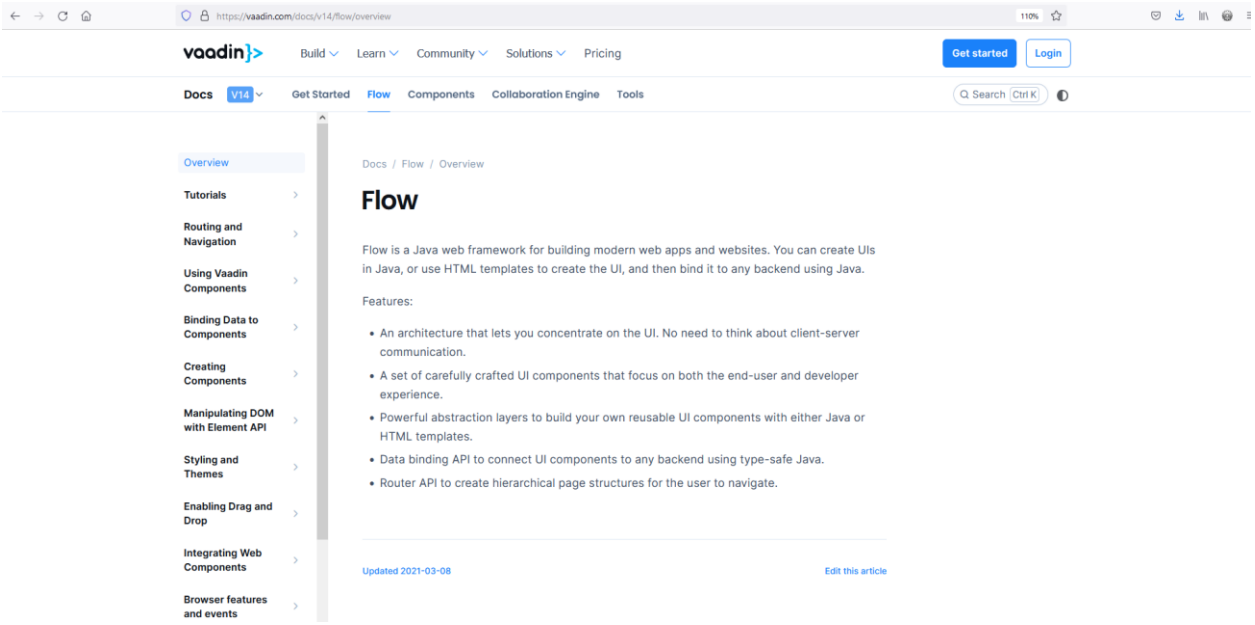


Ilustración 8. Documentación Vaadin

6. CREACIÓN DE APLICACIÓN EN VAADIN FLOW

6.1. Configuración inicial del proyecto.

La forma más sencilla de crear un nuevo proyecto de Vaadin desde cero es generándolo en https://vaadin.com/start.

Se genera un proyecto basado en Sprint Boot y Maven, se puede añadir en un repositorio de GitHub o directamente descargar el ZIP para abrirlo con cualquier IDE que permita trabajar con Java.

En el inicializador de aplicaciones de Vaadin se pueden añadir vistas con plantillas que después se pueden adaptar o tomar como base.

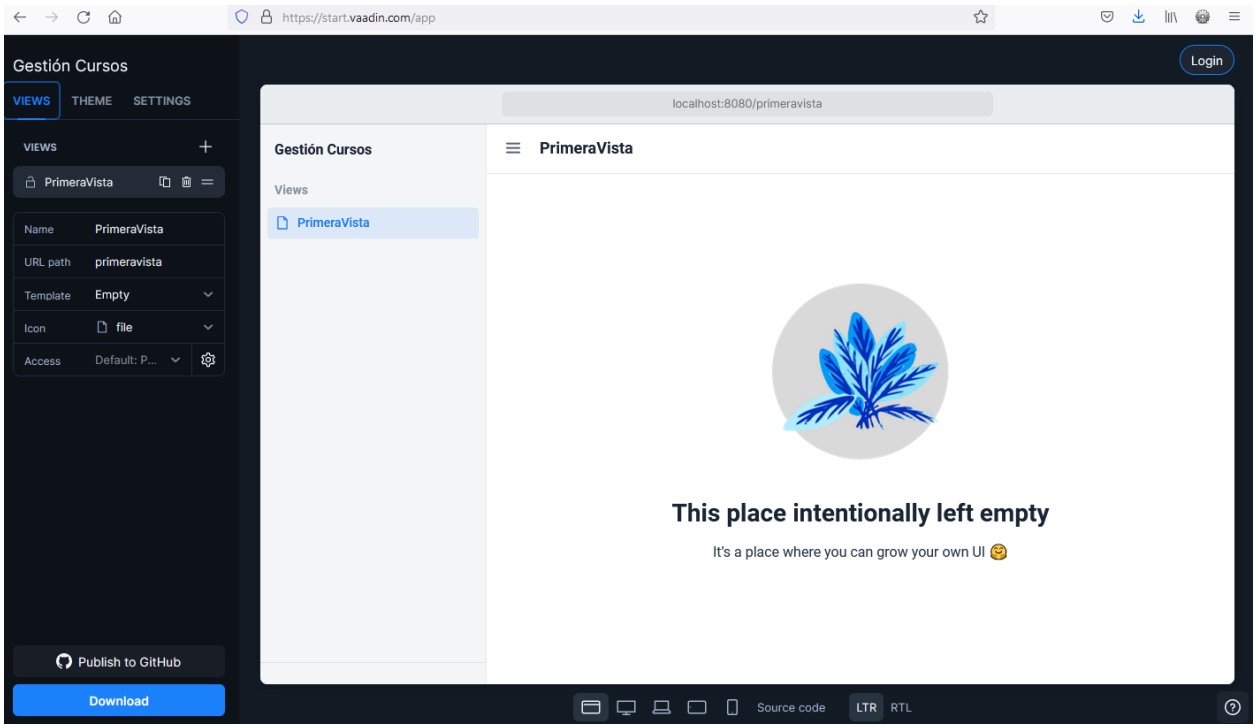


Ilustración 9. Configurador y creador de aplicaciones Vaadin 1

También se puede configurar el tema y estilos a utilizar en la aplicación además de configurar nombre del proyecto, layout y las tecnologías a utilizar. En este caso se va a utilizar la versión de 14 LTS de Vaadin con java 8.

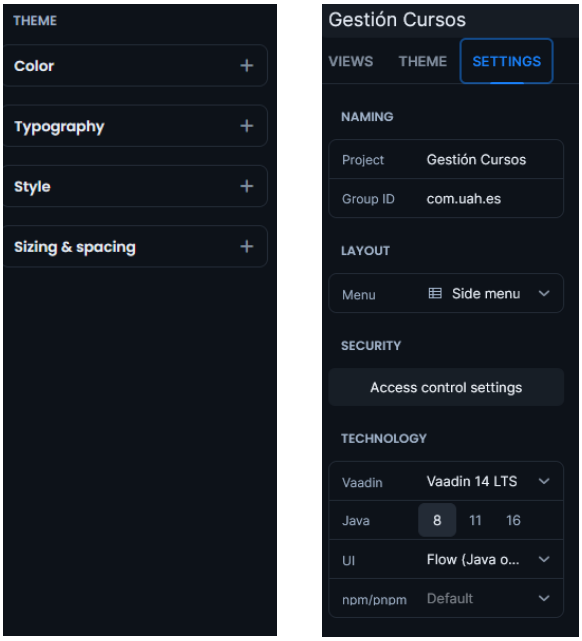


Ilustración 10. Configurador y creador de aplicaciones Vaadin 2

Cuando se ha generado el proyecto ya se puede abrir desde el IDE, en este caso se va a utilizar IntelliJ IDEA y tiene la siguiente estructura de paquetes y ficheros.

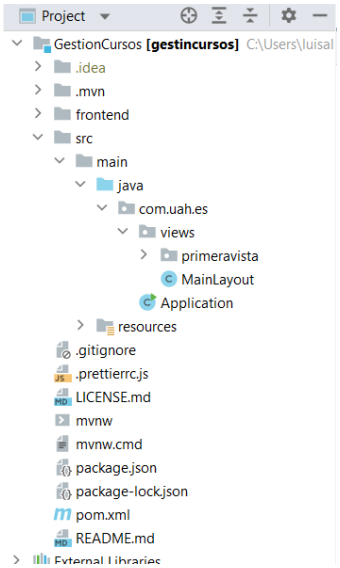


Ilustración 11. Paquetes y ficheros aplicación Vaadin

• **Frontend**: contiene los ficheros CSS para personalizar el aspecto visual de la aplicación.

• **Src/main/java/com/uah/es/views**: contienes las vistas, será aquí donde se irán añadiendo los componentes visuales de la aplicación.

• **Src/main/resources**: contiene el fichero de propiedades de Spring Boot además de otros recursos como imágenes, iconos, etc.

Con el proyecto descargado ya se tendría una mínima aplicación funcional con Vaadin, para comprobar se tendría que ejecutar y abrirla en el navegador en el puerto configurado.

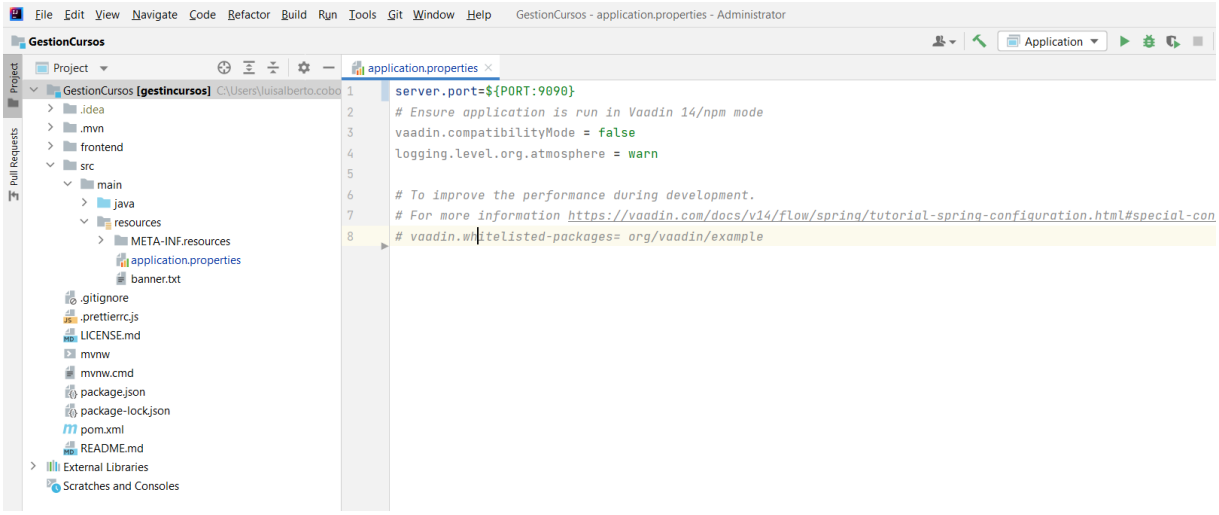


Ilustración 12. Fichero de propiedades

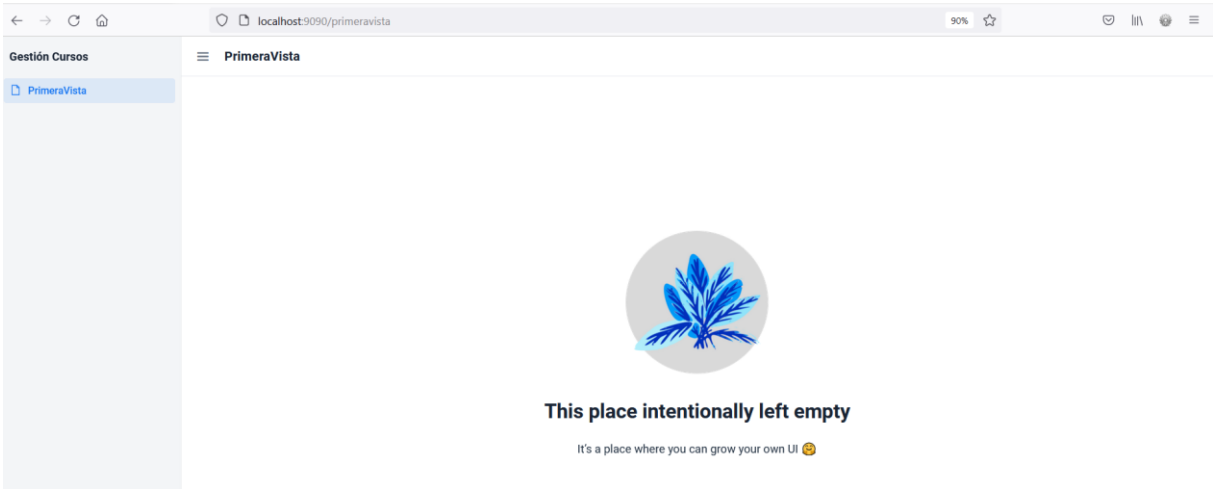


Ilustración 13. Ejecución de la aplicación inicial

A partir de este punto se pueden añadir vistas y funcionalidades de Vaadin a la aplicación. Lo primero que se va a realizar es una personalización de menú lateral y la cabecera de la aplicación.

La clase MainLayout.java es la principal y punto de partida de Vaadin, es la que contiene los componentes mencionados anteriormente y en la que se añadirán los enlaces a las vistas a crear.

Se asigna el estilo **sidemenu-header** mediante el método **setClassName** para cambiar el estilo de la cabecera.

```
private Component createHeaderContent() {
    HorizontalLayout layout = new HorizontalLayout();
```

```
    layout.setClassName("sidemenu-header");
    layout.getThemeList().set("dark", true);
    layout.setWidthFull();
    layout.setSpacing(false);
    layout.setAlignItems(FlexComponent.Alignment.CENTER);
    layout.add(new DrawerToggle());
    viewTitle = new H1();
    layout.add(viewTitle);

    return layout;
}
```

Se asigna el estilo **sidemenu-menu** mediante el método **setClassName** para cambiar el estilo del menú lateral.

```
private Component createDrawerContent(Tabs menu) {

    VerticalLayout layout = new VerticalLayout();
    layout.setClassName("sidemenu-menu");
    layout.setSizeFull();
    layout.setPadding(false);
    layout.setSpacing(false);
    layout.getThemeList().set("spacing-s", true);
    layout.setAlignItems(FlexComponent.Alignment.STRETCH);
    VerticalLayout usuarioLayout = new VerticalLayout();

    Avatar usuarioAvatar = new Avatar();

    usuarioLayout.add(usuarioAvatar);
    usuarioLayout.add(new Label(usuario.getNombre()));
    usuarioLayout.add(createLogoutLink());
    usuarioLayout.setAlignItems(FlexComponent.Alignment.CENTER);
    layout.add(usuarioLayout,menu);
    return layout;
}
```

Los estilos antes mencionados se encuentran en la carpeta **frontend** y en la hoja de estilos **main-layout.css.**

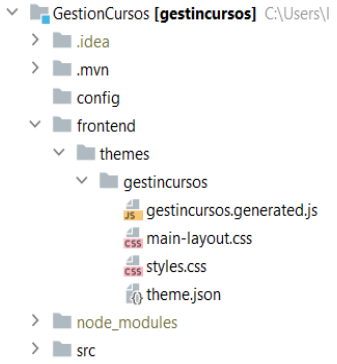


Ilustración 14. Ficheros Frontend Vaadin

```
.sidemenu-header {
  height: var(--lumo-size-xl);
  box-shadow: var(--lumo-box-shadow-s);
}
.sidemenu-header h1 {
  font-size: var(--lumo-font-size-l);
  margin: 0;
}
```

```
.sidemenu-menu #logo {
  box-sizing: border-box;
  box-shadow: inset 0 -1px var(--lumo-contrast-10pct);
  padding: var(--lumo-space-s) var(--lumo-space-m);
}

.sidemenu-menu #logo img {
  height: calc(var(--lumo-size-l) * 1.5);
}

.sidemenu-menu #logo h1 {
  font-size: var(--lumo-font-size-xl);
  font-weight: 600;
  margin: 0 var(--lumo-space-s);
}

.sidemenu-menu vaadin-tab {
  font-size: var(--lumo-font-size-s);
  height: var(--lumo-size-l);
  font-weight: 600;
  color: var(--lumo-body-text-color);
}

.sidemenu-menu vaadin-tab:hover {
  background-color: var(--lumo-contrast-5pct);
}

.sidemenu-menu vaadin-tab[selected] {
  background-color: var(--lumo-primary-color-10pct);
  color: var(--lumo-primary-text-color);
}

.link-margin {
  margin-top: 0px;
}
```

Ilustración 15. Modificación de cabecera y menú lateral

6.2. Comunicación con el backend.

Para la comunicación con los microservicios encargados de gestionar los datos y almacenarlos en la base de datos se necesitará utilizar **RestTemplate** para realizar las peticiones. Para ello se crearán tres paquetes con las clases necesarias.

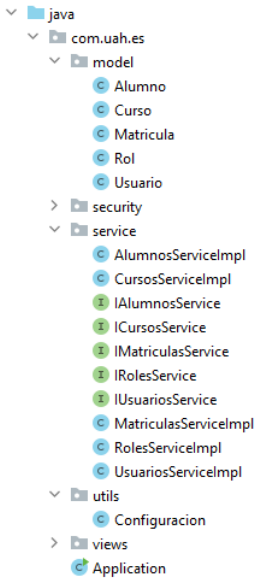


Ilustración 16. Paquetes y ficheros de los servicios

6.2.1. Paquete model.

Este paquete contienes las clases para la representación de los datos de negocio requeridos en la aplicación, a continuación, se detallará cada clase:

- **Alumno**: representa los alumnos con los atributos idAlumno, nombre, correo, cursos.

```
public class Alumno {

    private Integer idAlumno;
    private String nombre;
    private String correo;
    private List<Curso> cursos;

    public Alumno(String nombre, String correo) {
        this.idAlumno = 0;
        this.nombre = nombre;
        this.correo = correo;
        this.cursos = new ArrayList<Curso>();
    }

    public Alumno() {
    }

    // Getter and Setter ....

}
```

- **Curso**: representa los cursos con los atributos idCurso, nombre, duración, profesor, precio, categoría imagen y alumnos.

```
public class Curso {

    private Integer idCurso;
    private String nombre;
    private Integer duracion;
    private String profesor;
    private Double precio;
    private String categoria;
    private String imagen;
    private List<Alumno> alumnos;

    public Curso(Integer idCurso, String nombre, Integer duracion, String profesor, Double
precio, String categoria, String imagen, List<Alumno> alumnos) {
        this.idCurso = idCurso;
        this.nombre = nombre;
        this.duracion = duracion;
        this.profesor = profesor;
        this.precio = precio;
        this.categoria = categoria;
        this.imagen = imagen;
        this.alumnos = alumnos;
    }

    public Curso() {
    }

    // Getter and Setter ....
}
```

- **Matricula**: representa las matrículas (relación entre curso y alumno) con los atributos idMatricula, idCurso, precio, fecha y usuarios.

```
public class Matricula {

    private Integer idMatricula;
    private Integer idCurso;
    private Double precio;

    @JsonFormat(pattern = "yyyy-MM-dd", locale = "es-ES", timezone = "Europe/Madrid")
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private Date fecha;
    private Usuario usuario;

    public Matricula() {
    }

    public Matricula(Integer idCurso, Usuario usuario) {
        this.idCurso = idCurso;
        this.usuario = usuario;
    }

    // Getter and Setter
}
```

- **Rol**: representa los roles que pueden tener los usuarios con los atributos idRol y authority.

```
public class Rol {

    private Integer idRol;
    private String authority;

    public Rol() {
    }

    public Rol(Integer idRol, String authority) {
        this.idRol = idRol;
        this.authority = authority;
    }
```

```
    public Rol(String idRolAndName) {
        if (idRolAndName != null && idRolAndName.length() > 0) {
            String[] fieldPositions = idRolAndName.split("-");
            this.idRol = Integer.parseInt(fieldPositions[0]);
            this.authority = fieldPositions[1];
        }
    }

    // Getter and Setter
}
```

- **Usuario**: representa los usuarios con los atributos idUsuario, nombre, clave, correo, enable, roles y matriculas.

```
public class Usuario {

    private Integer idUsuario;
    private String nombre;
    private String clave;
    private String correo;
    private boolean enable;
    private List<Rol> roles;
    private List<Matricula> matriculas;

    public Usuario(Integer idUsuario, String nombre, String clave, String correo, boolean enable,
List<Rol> roles, List<Matricula> matriculas) {
        this.idUsuario = idUsuario;
        this.nombre = nombre;
        this.clave = clave;
        this.correo = correo;
        this.enable = enable;
        this.roles = roles;
        this.matriculas = matriculas;
    }

    public Usuario() {
    }

    // Getter and Setter ....
}
```

6.2.2. Paquete utils.

Este paquete solo contiene la clase Configuracion.java en la que se parametrizan las url de acceso a los servicios y los roles de los usuarios.

```
public class Configuracion {

    public static String ROL_ADMIN = "Admin";
    public static String ROL_ALUMNO = "Alumno";
    public static String ROL_PROFESOR = "Profesor";

    public static String URL = "http://localhost:8090/api";
    public static String URL_SERVICIO_USUARIOS = URL+"/zusuarios/usuarios";
    public static String URL_SERVICIO_ROLES = URL+"/zusuarios/roles";
    public static String URL_SERVICIO_MATRICULAS = URL+"/zusuarios/matriculas";
    public static String URL_SERVICIO_CURSOS = URL+"/zcursos/cursos";
    public static String URL_SERVICIO_ALUMNOS = URL+"/zcursos/alumnos";

}
```

6.2.3. Paquete service.

Este paquete contiene tanto las interfaces como su implementación para las peticiones, solo se detallará la implementación de ICursosService, ya que los demás son similares y siguen la misma estructura.

- **ICursosService:** este servicio cuenta con las operaciones necesarias para recuperar el listado de todos los cursos, crear, modificar y eliminar un curso existente en el sistema.

```
public interface ICursosService {

    Curso[] buscarTodos();

    Curso buscarCursoPorId(Integer idCurso);

    Curso[] buscarCursosPorNombre(String nombre);

    Curso[] buscarCursosPorCategoria(String categoria);

    Curso[] buscarCursosPorProfesor(String profesor);

    boolean guardarCurso(Curso curso);

    boolean actualizarCurso(Curso curso);

    boolean eliminarCurso(Integer idCurso);
}
```

Las primereas peticiones son GET para obtener los Cursos.

```
@Service
public class CursosServiceImpl implements ICursosService {

    String url = Configuracion.URL_SERVICIO_CURSOS;

    @Autowired
    RestTemplate template;

    @Override
    public Curso[] buscarTodos() {
        return template.getForObject(url, Curso[].class);
    }

    @Override
    public Curso buscarCursoPorId(Integer idCurso) {
        return template.getForObject(url + "/" + idCurso, Curso.class);
    }

   @Override
    public Curso[] buscarCursosPorNombre(String nombre) {
       return template.getForObject(url + "/nombre/" + nombre, Curso[].class);
    }

    @Override
    public Curso[] buscarCursosPorCategoria(String categoria) {
        return template.getForObject(url + "/categoria/" + categoria, Curso[].class);
    }

   @Override
    public Curso[] buscarCursosPorProfesor(String profesor) {
        return template.getForObject(url + "/profesor/" + profesor, Curso[].class);
    }
```

Para crear o editar un curso se utiliza POST y PUT respectivamente.

```
@Override
public boolean guardarCurso(Curso curso) {

    boolean result = false;
    curso.setIdCurso(0);
    try {
        template.postForEntity(url, curso, String.class);
        result = true;
    } catch (Exception ex) {
        System.out.println(ex.getMessage());
    }

    return result;
}

@Override
public boolean actualizarCurso(Curso curso) {

    boolean result = false;
    try {
        template.put(url, curso);
        result = true;
    } catch (Exception ex) {
        System.out.println(ex.getMessage());
    }

    return result;
}
```

Por último, para la eliminación de un curso se utiliza DELETE.

```
@Override
public boolean eliminarCurso(Integer idCurso) {
    boolean result = false;

    try {
        template.delete(url + "/" + idCurso);
        result = true;
    } catch (Exception ex) {
        System.out.println(ex.getMessage());
    }

    return result;
}
```

- **IAlumnosService:** este servicio cuenta con las operaciones necesarias para recuperar el listado de todos los alumnos, crear, modificar y eliminar un alumno existente en el sistema.

```
public interface IAlumnosService {

    Alumno[] buscarTodos();

    Alumno buscarAlumnoPorId(Integer idAlumno);

    Alumno buscarAlumnoPorCorreo(String correo);

    Alumno[] buscarAlumnosPorNombre(String nombre);

    boolean guardarAlumno(Alumno alumno);

    boolean actualizarAlumno(Alumno alumno);

    boolean eliminarAlumno(Integer idAlumno);

    boolean inscribirCurso(Integer idAlumno, Integer idCurso);

    boolean desinscribirCurso(Integer idAlumno, Integer idCurso);
}
```

- **IMatriculasService:** este servicio cuenta con las operaciones necesarias para recuperar el listado de todas las matrículas, guardar y eliminar una matrícula existente en el sistema.

```
public interface IMatriculasService {

    Matricula[] buscarTodas();

    Matricula buscarMatriculaPorId(Integer idMatricula);

    Matricula buscarMatriculaPorIdCursoIdUsuario(Integer idCurso, Integer idUsuario);

    boolean guardarMatricula(Matricula matricula);

    boolean eliminarMatricula(Curso curso, Alumno alumno);
}
```

- **IUsuariosService:** este servicio cuenta con las operaciones necesarias para recuperar el listado de todos los usuarios, crear, modificar y eliminar un usuario existente en el sistema.

```
public interface IUsuariosService {

    Usuario[] buscarTodos();

    Usuario[] buscarUsuariosPorRol(Integer idRol);

    Usuario buscarUsuarioPorId(Integer idUsuario);

    Usuario buscarUsuarioPorNombre(String nombre);

    Usuario buscarUsuarioPorCorreo(String correo);

    Usuario login(String correo, String clave);

    boolean guardarUsuario(Usuario usuario);

    boolean actualizarUsuario(Usuario usuario);

    boolean eliminarUsuario(Integer idUsuario);
}
```

- **IRolesService:** este servicio cuenta con las operaciones necesarias para recuperar el listado de todos los roles, crear, modificar y eliminar un rol existente en el sistema.

```
public interface IRolesService {

    Rol[] buscarTodos();

    Rol buscarRolPorId(Integer idRol);

    void guardarRol(Rol rol);

    void eliminarRol(Integer idRol);
}
```

6.3. Vista cursos.

Para gestionar los cursos disponibles en la aplicación se ha creado la vista **CursosView.java**. Dispone de un buscador en la parte superior, un botón para crear nuevos cursos, una tabla con la información de los cursos, además dicha tabla dispone de acciones como ver el listado de alumnos del curso, editar y eliminar un curso, por último, también se dispone de un enlace para la generación y descarga de un fichero CSV con la información de los cursos que se visualiza en la tabla.

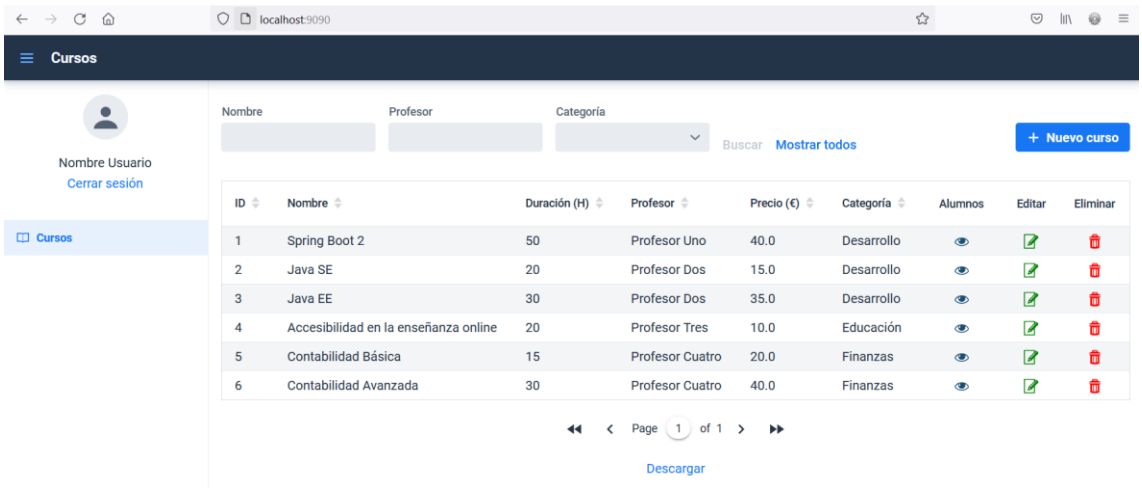


Ilustración 17. Vista cursos

A continuación, se explicará el diseño e implementación de cada uno de los componentes que integran la vista de cursos.

6.3.1. Buscador de cursos.

El buscador está compuesto por dos campos de texto y una lista desplegable para introducción el nombre, profesor y categoría por las que se desea filtrar los cursos, además dispone de dos botones para realizar la búsqueda por el filtro seleccionado o buscar todos los cursos disponibles en la aplicación.

Ilustración 18. Buscador de cursos

En primer lugar, se declaran e instancian los componentes de Vaadin que se van a utilizar en este caso son **TextFiled**, **Select** y **Button**.

```
TextField nombreFiltro = new TextField();
TextField profesorFiltro = new TextField();
Select<String> categoriaFiltro = new Select<>();
Button buscarBtn = new Button("Buscar");
Button mostrarTodosBtn = new Button("Mostrar todos");
```

Se configuran dichos componentes indicándoles el ancho que van a utilizar los filtros y en el caso del filtro de categoría se asignan los valores que va a mostrar en la lista desplegable.

```
private Component configurarBuscador() {

    HorizontalLayout buscadorLayout = new HorizontalLayout();

    // Configuracion del filtro para buscar por nombre
    nombreFiltro.setLabel("Nombre");
    nombreFiltro.setWidth("30%");
    nombreFiltro.setClearButtonVisible(true);
    nombreFiltro.setValueChangeMode(ValueChangeMode.EAGER);
    // Configuracion del filtro para buscar por profesor
    profesorFiltro.setLabel("Profesor");
    profesorFiltro.setWidth("30%");
    profesorFiltro.setClearButtonVisible(true);
    profesorFiltro.setValueChangeMode(ValueChangeMode.EAGER);
    // Configuracion del filtro para buscar por categoría
    categoriaFiltro.setLabel("Categoría");
    categoriaFiltro.setItems("","Desarrollo", "Educación","Finanzas");
```

Solo se habilitará el botón de buscar cuando alguno de los filtros tenga valor, además solo se puede filtrar por uno de los filtros, por ello añade un **ValueChangeListener** a cada filtro y se utiliza el método **setEnabled** para desactivar los demás filtros y el botón buscar en caso de que el filtro cambie de valor.

```
// Se habilita el btn buscar solo cuando el nombre tenga valor y los demás filtros se inhabilitan.
nombreFiltro.addValueChangeListener(e -> {
    buscarBtn.setEnabled(!Objects.equals(nombreFiltro.getValue(), ""));
    profesorFiltro.setEnabled(Objects.equals(nombreFiltro.getValue(), ""));
    categoriaFiltro.setEnabled(Objects.equals(nombreFiltro.getValue(), ""));
});
// Se habilita el btn buscar solo cuando el filtro profesor tenga valor y los demás filtros se inhabilitan
profesorFiltro.addValueChangeListener(e -> {
```

```
    buscarBtn.setEnabled(!Objects.equals(profesorFiltro.getValue(), ""));
    nombreFiltro.setEnabled(Objects.equals(profesorFiltro.getValue(), ""));
    categoriaFiltro.setEnabled(Objects.equals(profesorFiltro.getValue(), ""));
});
// Se habilita el btn buscar solo cuando el filtro categoría tenga valor y los demás filtros se inhabilitan
categoriaFiltro.addValueChangeListener(e -> {
    buscarBtn.setEnabled(!Objects.equals(categoriaFiltro.getValue(), ""));
    nombreFiltro.setEnabled(Objects.equals(categoriaFiltro.getValue(), ""));
    profesorFiltro.setEnabled(Objects.equals(categoriaFiltro.getValue(), ""));
});
```

Para los botones también se tienen que configurar el estilo y añadir el **ClickListener** para realizar la acción cuando se haga click sobre ellos. En el caso del botón **mostrarTodosBtn** lo que hará es limpiar los valores de los filtros y obtener todos los cursos.

```
//Se configuran los botones del buscador y sus listeners
buscarBtn.getStyle().set("cursor", "pointer");
buscarBtn.addThemeVariants(ButtonVariant.LUMO_TERTIARY_INLINE);
buscarBtn.addClickListener(e -> filtrar());

mostrarTodosBtn.getStyle().set("cursor", "pointer");
mostrarTodosBtn.addThemeVariants(ButtonVariant.LUMO_TERTIARY_INLINE);
mostrarTodosBtn.addClickListener(e -> {
    //Limpiar los buscadores
    nombreFiltro.clear();
    profesorFiltro.clear();
    categoriaFiltro.setValue("");
    // Habilitar los buscadores
    nombreFiltro.setEnabled(true);
    profesorFiltro.setEnabled(true);
    categoriaFiltro.setEnabled(true);
    obtenerTodosCursos();
});
```

Por último, se añaden los componentes antes mencionados al layout de los botones y del buscador.

```
HorizontalLayout layoutBtns = new HorizontalLayout();
layoutBtns.setDefaultVerticalComponentAlignment(FlexComponent.Alignment.END);
layoutBtns.add(buscarBtn,mostrarTodosBtn);

//Se añaden los componentes visuales
buscadorLayout.add(nombreFiltro,profesorFiltro,categoriaFiltro,layoutBtns);
return buscadorLayout;
```

6.3.2. Formulario para crear o editar datos del curso.

Para la creación o edición de un curso existente se ha implementado un formulario en donde el usuario tiene que introducir los datos del curso, dicho formulario utiliza componentes de Vaadin **(Texfiled, IntegerField, Select, Upload, Button y H2)**.

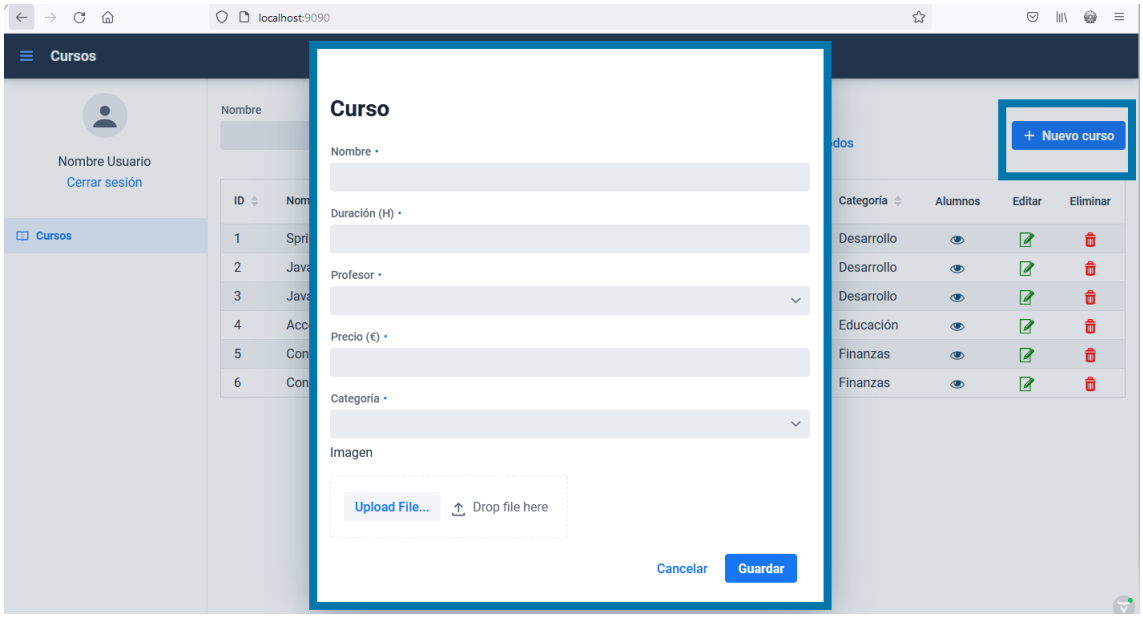


Ilustración 19. Formulario nuevo curso

Se ha creado la clase **CursoForm.java** que extiende de la clase **FormLayout** para la implementación de formulario. Para ello lo primero que se hace es declarar e instancias los componentes antes mencionados.

```
public class CursoForm extends FormLayout {

    MemoryBuffer buffer = new MemoryBuffer();
    Binder<Curso> binder = new BeanValidationBinder<>(Curso.class);
    private Curso curso = new Curso();

    // Inputs y btns del formulario
    TextField nombre= new TextField("Nombre");
    IntegerField duracion = new IntegerField("Duración (H)");
    TextField precio = new TextField ("Precio (€)");
    Select<String> categoria = new Select<>();
    Select<String> profesor = new Select<>();
    Upload imagen = new Upload(buffer);
    Button cancelarBtn = new Button("Cancelar");
    Button guardarBtn = new Button("Guardar");
    H2 titulo = new H2("Curso");
```

En primer lugar, se tienen que configurar y asignar los valores para los componentes **Select** de categoría y profesor.

```
public CursoForm(IUsuariosService usuariosService){

    // Se configura el listado de categorias en el componente Select
    categoria.setLabel("Categoría");
    categoria.setItems("Desarrollo", "Educación","Finanzas");

    // Se configura el listado de profesores en el componente Select
    profesor.setLabel("Profesor");
    List<Usuario> usuariosProfesor = Arrays.asList(usuariosService.buscarUsuariosPorRol(3));
    List<String> nombresProfesores = new ArrayList<>();
    usuariosProfesor.forEach(u ->{
        nombresProfesores.add(u.getNombre());
    });
    profesor.setItems(nombresProfesores);
```

Se mapean los demás componentes del formulario con el objeto **Curso** en el que se guardará la información, aplicando validaciones y conversiones de datos en caso de ser necesario.

```
// Relacionamos los atributos del objeto Curso con los campos del formulario
binder.forField(nombre)
        .asRequired("Campo requerido")
        .bind(Curso::getNombre,Curso::setNombre);
binder.forField(duracion)
        .asRequired("Campo requerido")
        .bind(Curso::getDuracion,Curso::setDuracion);
binder.forField(profesor)
        .asRequired("Campo requerido")
        .bind(Curso::getProfesor,Curso::setProfesor);
binder.forField(precio)
        .withNullRepresentation("")
        .withConverter(new StringToDoubleConverter("No es un precio válido"))
        .asRequired("Campo requerido")
        .bind(Curso::getPrecio,Curso::setPrecio);
binder.forField(categoria)
        .asRequired("Campo requerido")
        .bind(Curso::getCategoria,Curso::setCategoria);
```

Se asigna el tamaño máximo que puede tener el formulario y se añaden los componentes al formulario.

```
setMaxWidth("600px");
//Se añaden los componentes a la vista
add(titulo,nombre,duracion,profesor,precio,categoria,configurarCargaImagen(),configurarBtnsLayout
());
```

Para utilizar el componente **Upload** de Vaadin se tiene que configurar el tipo de archivos que se pueden cargar mediante el método **setAccepteFileTypes** y el listener para tratar el fichero cuando se haya cargado en el formulario utilizando el método **addSucceededListener**.

```
private Component configurarCargaImagen() {

    VerticalLayout verticalLayout = new VerticalLayout();
    verticalLayout.setPadding(false);

    // Se configura el componente uploap para la carga de la imagen en el formulario
    Label nombreCampoImagen = new Label("Imagen");
    imagen.setAcceptedFileTypes("image/jpeg", "image/png", "image/gif");
    imagen.addSucceededListener(event -> {
        curso.setImagen(event.getFileName());
    });

    verticalLayout.add(nombreCampoImagen,imagen);
    return verticalLayout;
}
```

Por último, se configuran los botones de cancelar y guardar añadiendo los listener y los estilos.

```
private Component configurarBtnsLayout() {

    //Se configura los btns del formulario
    guardarBtn.addThemeVariants(ButtonVariant.LUMO_PRIMARY);
    cancelarBtn.addThemeVariants(ButtonVariant.LUMO_TERTIARY);
    guardarBtn.addClickListener(click -> validarYGuardar());
    cancelarBtn.addClickListener(click -> fireEvent(new CerrarEvent(this)));
    binder.addStatusChangeListener(evt -> guardarBtn.setEnabled(binder.isValid()));

    HorizontalLayout btnsLayout =  new HorizontalLayout();
    btnsLayout.setPadding(true);
```

```
    btnsLayout.setJustifyContentMode(FlexComponent.JustifyContentMode.END);
    btnsLayout.add(cancelarBtn,guardarBtn);
    return btnsLayout;
}
```

Cuando ya se ha creado y configurado la clase **CursoForm.java** ya se puede referenciar desde **CursosView.java** para poder utilizar el formulario en la vista de gestión de cursos. Para ello solo se tiene que crear e instancias un objeto de dicha clase y añadir los listener para guardar el curso o cerrar el formulario dependiendo del botón presionado. Además, se tiene que configurar **nuevoCursoBtn** para poder abrir el formulario.

```
private Component configurarFormulario(){

    CursoForm cursoForm = new CursoForm(usuariosService);
    cursoForm.addListener(CursoForm.GuardarEvent.class, this::guardarCurso);
    cursoForm.addListener(CursoForm.CerrarEvent.class, e -> cerrarFormulario());
    formularioDg.add(cursoForm);

    //Se configura el boton para añadir un nuevo Curso
    nuevoCursoBtn.addThemeVariants(ButtonVariant.LUMO_PRIMARY);
    nuevoCursoBtn.addClickListener(event -> {
        formularioDg.open();
    });

    HorizontalLayout layoutBtn = new HorizontalLayout();
    layoutBtn.setDefaultVerticalComponentAlignment(FlexComponent.Alignment.END);
    layoutBtn.getElement().getStyle().set("margin-left", "auto");
    layoutBtn.add(nuevoCursoBtn);

    return layoutBtn;
}
```

6.3.3. Tabla con datos los cursos.

Para la visualización y gestión de los cursos se genera una tabla en las que poder ver los datos de los cursos en columnas, además de realizar acciones sobre ellos.

Hasta este momento solo se han utilizado componentes oficiales, pero una de las ventajas que ofrece Vaadin es que la comunidad de usuarios puede desarrollar y publicar componentes, de esta forma los demás usuarios pueden hacer uso de dichos componentes.

Debido a que el componente **Grid** de Vaadin no dispone de paginación, se ha encontrado el componente **PaginatedGrid** que si permite configurar una visualización de datos en páginas. Para poder utilizarlo es necesario agregar la dependencia al **POM**.

```
<dependency>
    <groupId>org.vaadin.klaudeta</groupId>
    <artifactId>grid-pagination</artifactId>
    <version>2.0.10</version>
</dependency>
```

Una vez añadida la dependencia ya se puede crear e instanciar nuestro objeto **PaginatedGrid**.

```
PaginatedGrid<Curso> grid = new PaginatedGrid<>();
```

La forma de añadir columnas a la tabla es muy sencilla, se tiene que utilizar el método **AddColumn,** con el cual se puede configurar el atributo del objeto Curso que se va a visualizar en dicha columna, la cabecera, el id, el tamaño y si permite o no permite ordenación.

En primer lugar, solo se van a configurar las columnas de visualización de datos.

```
private Component configurarGrid() {


// Se añaden las columnas al grid
grid.addColumn(Curso::getIdCurso).setHeader("ID").setKey("id").setSortable(true).setAutoWidth(true);
grid.addColumn(Curso::getNombre).setHeader("Nombre").setKey("nombre").setSortable(true).setAutoWidth(true);
grid.addColumn(Curso::getDuracion).setHeader("Duración (H)").setKey("duracion").setSortable(true).setAutoWidth(true);
grid.addColumn(Curso::getProfesor).setHeader("Profesor").setKey("profesor").setSortable(true).setAutoWidth(true);
grid.addColumn(Curso::getPrecio).setHeader("Precio (€)").setKey("precio").setSortable(true).setAutoWidth(true);
grid.addColumn(Curso::getCategoria).setHeader("Categoría").setKey("categoria").setSortable(true).setAutoWidth(true);
```

Ilustración 20. Tabla cursos 1

Para las columnas de acciones (Alumnos, Editar y Eliminar) su configuración es diferente, ya que se tiene que configurar el icono y añadir el Listener para realizar la acción cuando se haga click sobre ella en un determinado curso.

```
grid.addComponentColumn(item -> {
    Icon alumnosBtn = new Icon(VaadinIcon.EYE);
    alumnosBtn.setColor("#1B4F72");
    alumnosBtn.getStyle().set("cursor", "pointer");
    alumnosBtn.setSize("18px");
    alumnosBtn.addClickListener(e -> verListadoAlumnos(item));
    return alumnosBtn;
})
.setKey("alumnos")
.setHeader("Alumnos")
.setTextAlign(ColumnTextAlign.CENTER)

grid.addComponentColumn(item -> {
    Icon editarIcon = new Icon(VaadinIcon.EDIT);
    editarIcon.setColor("green");
    editarIcon.getStyle().set("cursor", "pointer");
    editarIcon.setSize("18px");
    editarIcon.addClickListener(e -> editarCurso(item));
    return editarIcon;
})
.setKey("editar")
.setHeader("Editar")
.setTextAlign(ColumnTextAlign.CENTER)
.setAutoWidth(true)
```

```
grid.addComponentColumn(item -> {
    Icon editarIcon = new Icon(VaadinIcon.TRASH);
    editarIcon.setColor("red");
    editarIcon.getStyle().set("cursor", "pointer");
    editarIcon.setSize("18px");
    editarIcon.addClickListener(e -> eliminarCurso(item));
    return editarIcon;
})
.setKey("eliminar")
.setHeader("Eliminar")
.setTextAlign(ColumnTextAlign.CENTER)
.setAutoWidth(true)
```

Ilustración 21. Tabla cursos 2

6.3.4. Dialogo de confirmación.

Para cada acción que implique un cambio en los datos se le requiere al usuario una confirmación y para ello se ha utilizado el componente **Dialog**, es un modal en el cual se le puede añadir más componentes. Para este caso solo se va a añadir un mensaje y los botones para cancelar o confirmar la acción.

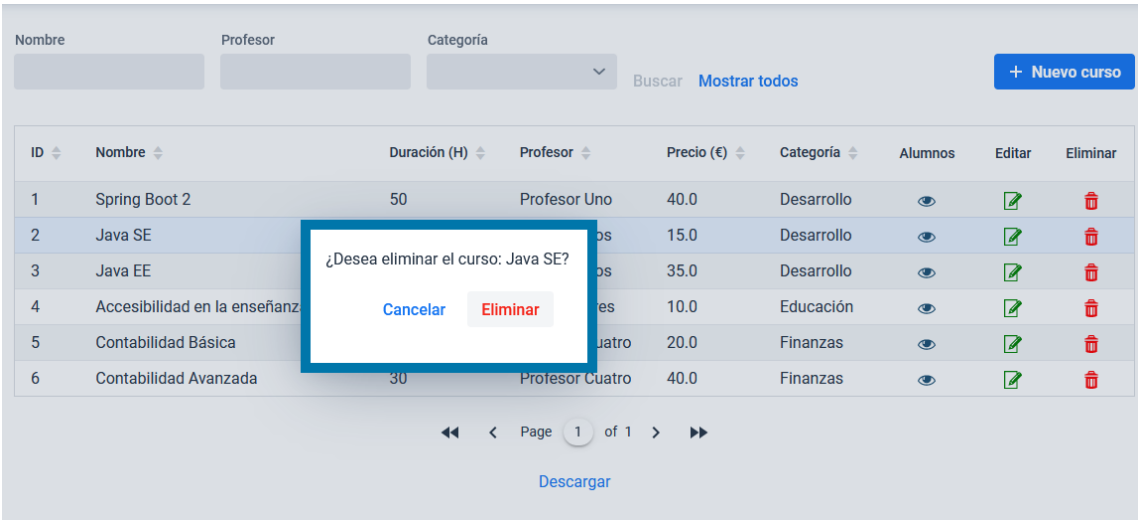


Ilustración 22. Dialogo de confirmación eliminación

Primero se crea e instancia el **Dialog**, se configura el mensaje de confirmación y los botones.

```
private void eliminarCurso(Curso curso) {

    // Se configura el Dialog para confirmar la eliminación
    Dialog confirmacionDg = new Dialog();
    Label msjConfirmacion = new Label();
    msjConfirmacion.setText("¿Desea eliminar el curso: "+curso.getNombre()+"?");

    HorizontalLayout btnsLayout = new HorizontalLayout();
    Button cancelarBtn = new Button("Cancelar");
    Button eliminarBtn = new Button("Eliminar");
    eliminarBtn.addThemeVariants(ButtonVariant.LUMO_ERROR);
    cancelarBtn.addThemeVariants(ButtonVariant.LUMO_TERTIARY);
```

Se configuran los Listener para los dos botones y se añaden los componentes al Dialog.

```
eliminarBtn.addClickListener(click -> {
    // Se muestra la notificación indicando el restultado
    if(cursosService.eliminarCurso(curso.getIdCurso())){
        notificacionOK.setText("Se ha eliminado correctamente el curso");
        notificacionOK.open();
    } else {
        notificacionKO.setText("Error al eliminar el curso");
        notificacionKO.open();
    }
    // Se actualiza el grid con todos los cursos
    confirmacionDg.close();
    obtenerTodosCursos();

});
cancelarBtn.addClickListener(click -> {
    confirmacionDg.close();
});
```

```
//Se añaden los componentes visuales al Dialog
btnsLayout.setPadding(true);
btnsLayout.setJustifyContentMode(FlexComponent.JustifyContentMode.END);
btnsLayout.add(cancelarBtn,eliminarBtn);
confirmacionDg.add(msjConfirmacion,btnsLayout);
confirmacionDg.open();
}
```

6.3.5. Listado de alumnos.

En la visualización del listado de alumnos del curso se utiliza un **Dialog** y un **Grid**.

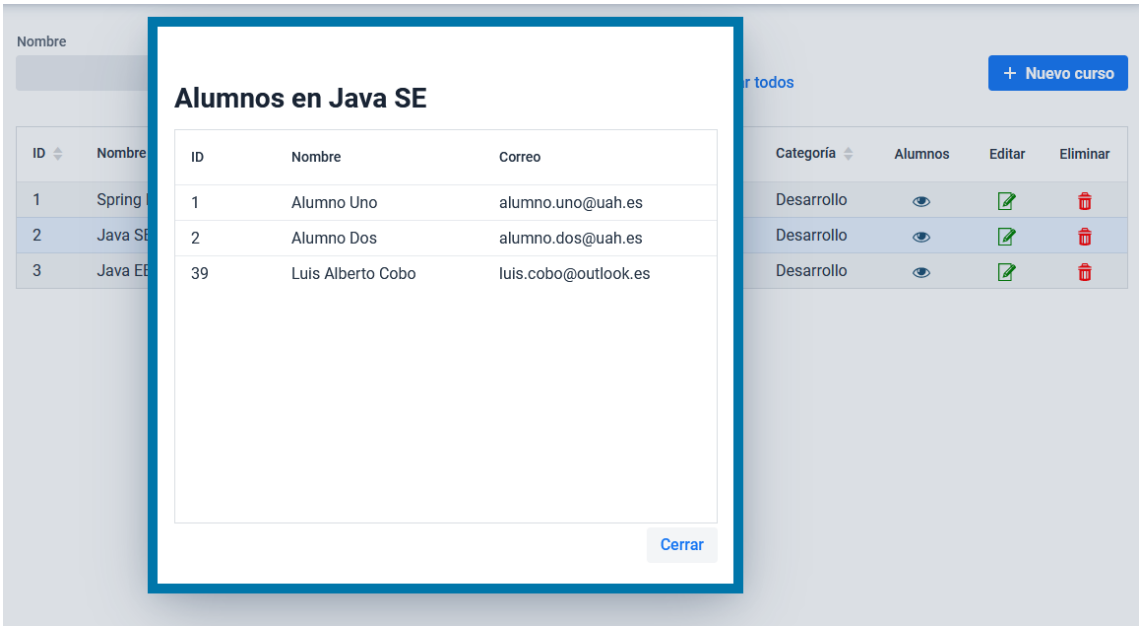


Ilustración 23. Listado de alumnos del curso

Primero se crean y configuran los componentes para después añadir al componente **Dialog**.

```
private void verListadoAlumnos(Curso curso) {

    // Se configura el Dialog para visualizar el listado de alumnos de un curso
    Dialog listadoAlumnosDg = new Dialog();
    H2 titulo = new H2("Alumnos en " + curso.getNombre());
    Grid<Alumno> gridAlumnos= new Grid<>();
    gridAlumnos.addColumn(Alumno::getIdAlumno).setHeader("ID").setKey("id").setAutoWidth(true);
    gridAlumnos.addColumn(Alumno::getNombre).setHeader("Nombre").setKey("nombre").setAutoWidth(true);
    gridAlumnos.addColumn(Alumno::getCorreo).setHeader("Correo").setKey("correo").setAutoWidth(true);
    gridAlumnos.setItems(curso.getAlumnos());
    HorizontalLayout btns = new HorizontalLayout();
    Button cerrarBtn = new Button("Cerrar");

    cerrarBtn.addClickListener(click -> {
        listadoAlumnosDg.close();
    });
    //Se añaden los componentes visuales al Dialog
    btns.setJustifyContentMode(FlexComponent.JustifyContentMode.END);
    btns.add(cerrarBtn);
    listadoAlumnosDg.add(titulo,gridAlumnos,btns);
    listadoAlumnosDg.setWidth("600px");
    listadoAlumnosDg.open();
}
```

6.3.6. Exportar datos en CSV.

En la descarga del fichero CSV se ha añadido el componente **Anchor** para visualizar el enlace de descarga.

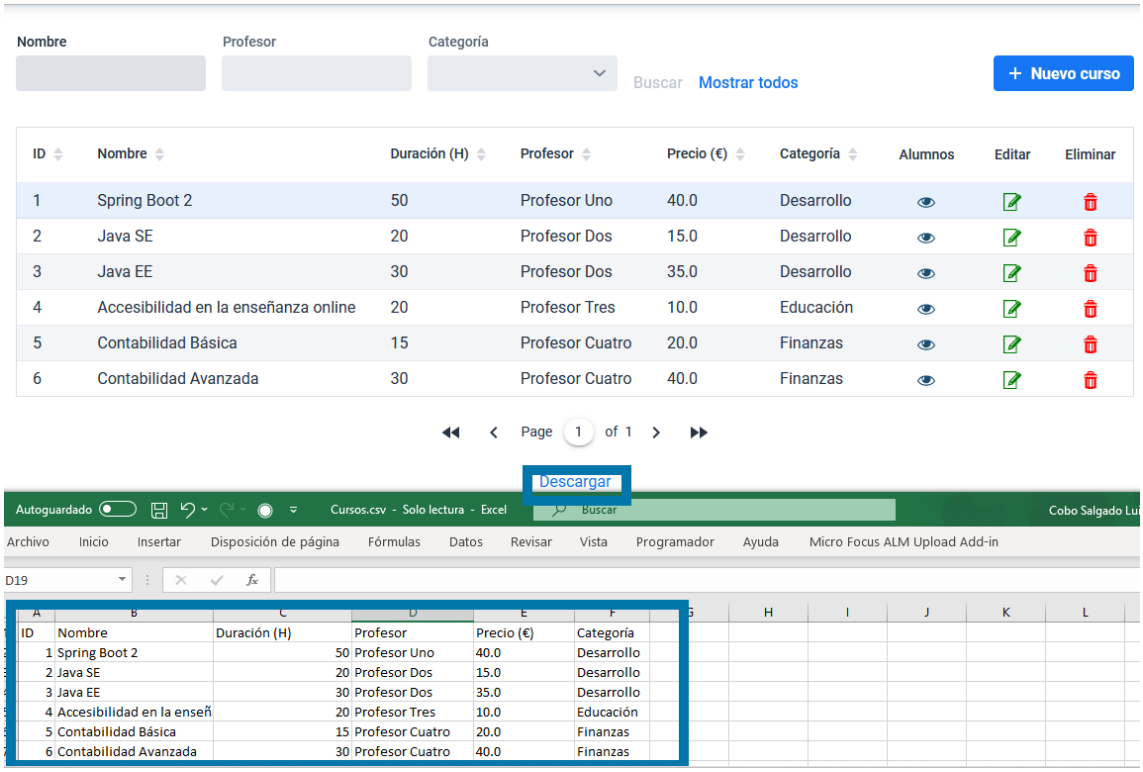


Ilustración 24. Descarga fichero CSV

Se crea tanto el componente **HorizontalLayout** como el **Anchor** para la visualización del enlace de descarga.

```
private Component configurarExportarCsv() {

    HorizontalLayout layoutLink = new HorizontalLayout();
    linkDescargaCsv = new Anchor(new StreamResource("Cursos.csv", this::generarCsv),
"Descargar");
    layoutLink.setJustifyContentMode(FlexComponent.JustifyContentMode.CENTER);
    layoutLink.add(linkDescargaCsv);
    return layoutLink;
}
```

Para generar el fichero, primero se obtiene el listado de cursos, se genera la primera fila con las cabeceras de las columnas y después se generan tantas líneas como cursos.

```
private InputStream generarCsv() {
    try {
        List<Curso> cursosCsv = new ArrayList<Curso>();
        if(isListadoMisCursos){
            cursosCsv = listaMisCursos;
        } else {
            cursosCsv = listaCursos;
        }
        StringWriter stringWriter = new StringWriter();
        CSVWriter csvWriter = new CSVWriter(stringWriter);
        csvWriter.setSeparatorChar(';');
        csvWriter.writeNext("ID", "Nombre", "Duración (H)", "Profesor", "Precio (€)", "Categoría");
        cursosCsv.forEach(c -> csvWriter.writeNext(String.valueOf(c.getIdCurso()),
c.getNombre(),c.getDuracion().toString(),c.getProfesor(),c.getPrecio().toString(),c.getCategoria(
))
        );
        return IOUtils.toInputStream(stringWriter.toString(), "UTF-8");
```

```
    } catch (IOException e) {
        e.printStackTrace();
        return null;
    }
}
```

6.4. Vista alumnos y vista usuarios.

Para gestionar los alumnos y usuarios se crean dos vistas **AlumnosView.java** y **UsuariosView.java**, estas vistas tienen una implementación similar a la vista de los cursos explicada en el apartado anterior. Por lo tanto, en este apartado no se explicará dicha implementación.

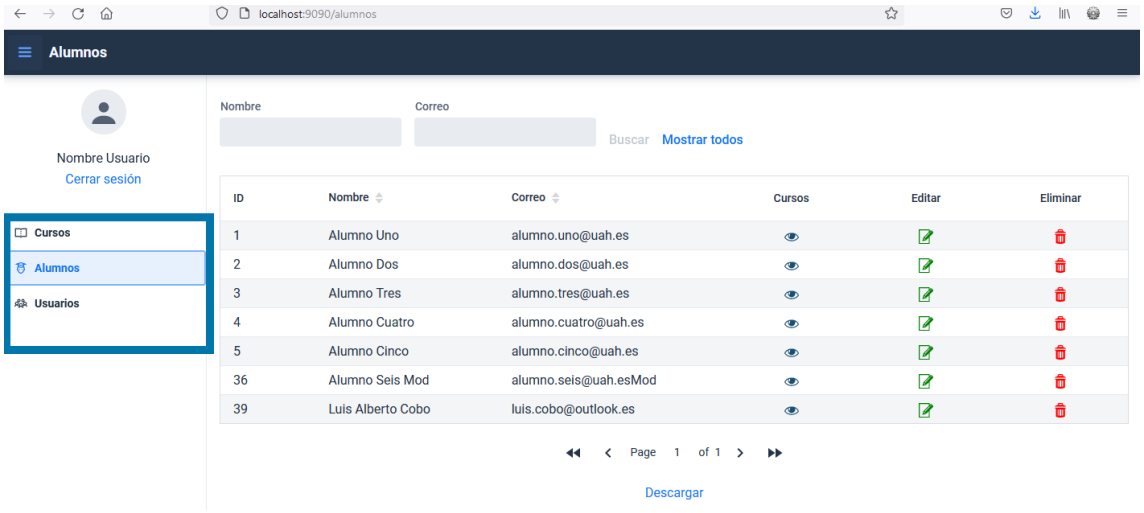


Ilustración 25. Vista alumnos

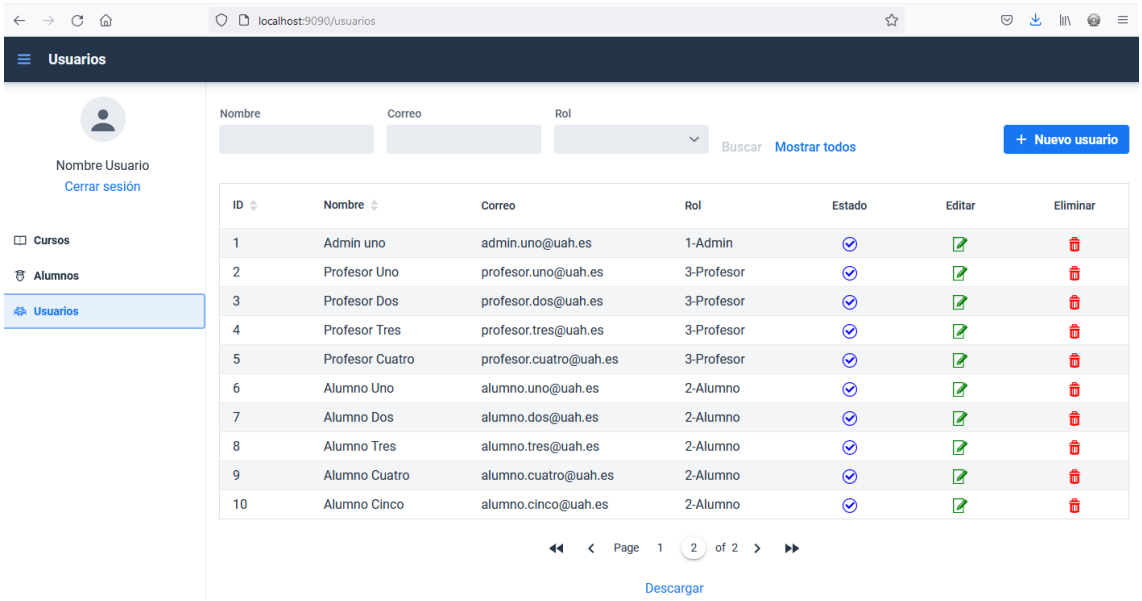


Ilustración 26. Vista usuarios

6.5. Vistas estadísticas.

Para hacer uso de los componentes que solo se pueden utilizar en la versión de pago se ha creado la vista **EstadistivasView.java** para visualizar gráficos con estadísticas de los cursos, roles y matrículas.

En primer lugar, se utilizará el componente **Board** para crear una visualización de componentes distribuida en filas y columnas tal y como se puede apreciar en la siguiente imagen.

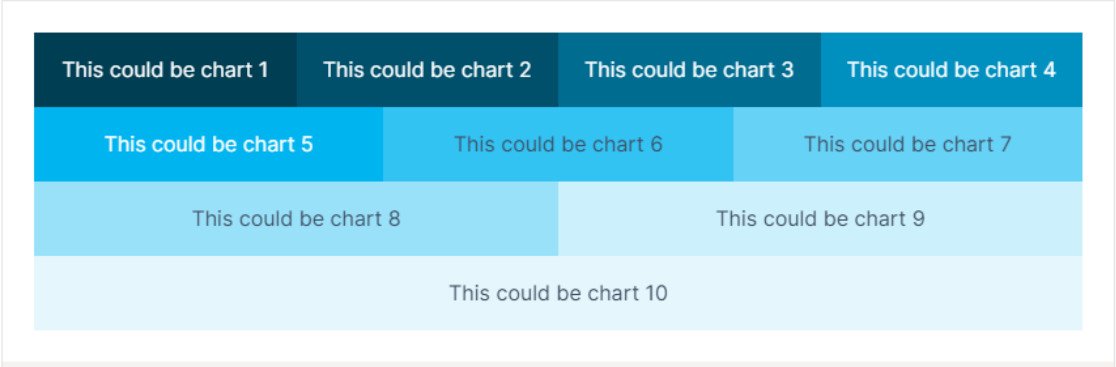


Ilustración 27. Configuración componente Board

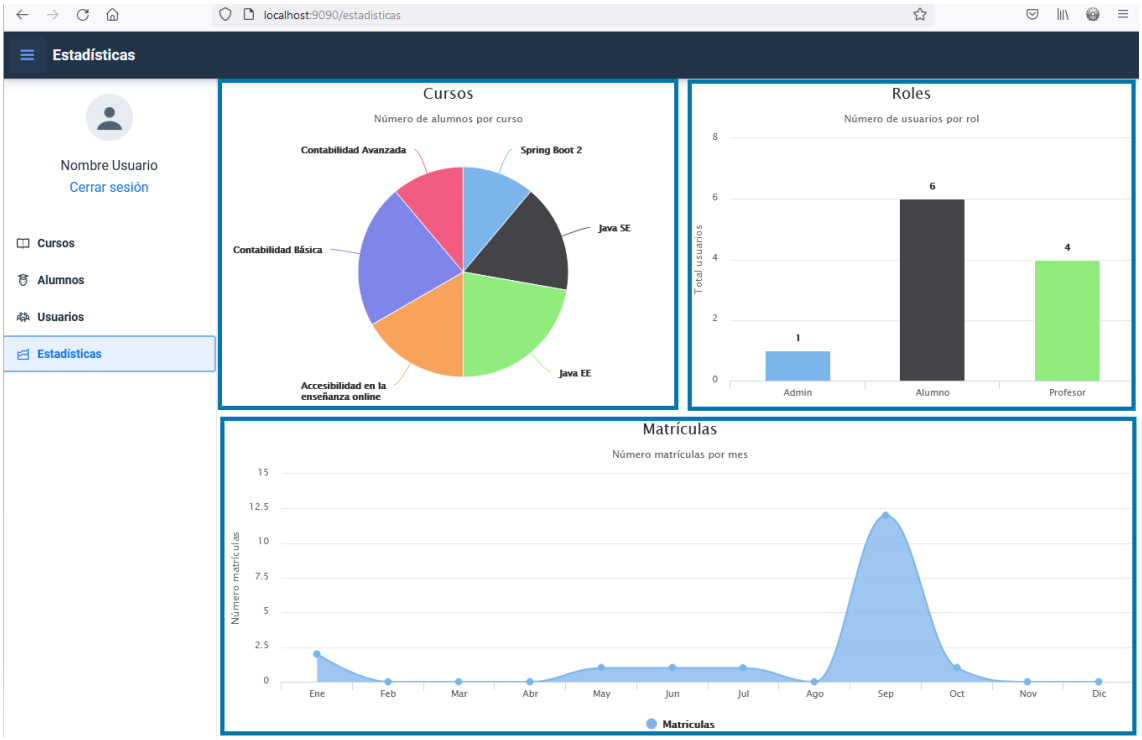


Ilustración 28. Vista estadísticas

Lo primero es realizar la inicialización de los servicios para recuperar los datos a mostrar y después crear y configurar el componente **Board** con los gráficos de la imagen anterior.

```
EstadisticasView(ICursosService cursosService, IUsuariosService usuariosService, IRolesService
rolesService, IMatriculasService matriculasService){
```

```
    //Se inicializan los servicios
    this.cursosService = cursosService;
    this.usuariosService = usuariosService;
    this.rolesService =  rolesService;
    this.matriculasService = matriculasService;

    //Configuracion de componente Board para visualizar los distintos gráficos
    Board board = new Board();
    board.addRow(graficoCursosAlumnos(),graficoRolesUsuarios());
    board.addRow(graficoMatriculasPorMes());
    add(board);
}
```

En el gráfico de número de alumnos por curso se utilizará el componente **Chart** con el tipo **PIE**. Primero se configurar el título y subtítulo del gráfico y por último se genera un **DataSeries** con cada uno de los cursos disponibles y el total de alumnos que tiene.

```
private Component graficoCursosAlumnos() {

    Chart graficoCursos = new Chart(ChartType.PIE);
    Configuration conf = graficoCursos.getConfiguration();

    // Configuracion del título y subtitulo
    conf.setTitle("Cursos");
    conf.setSubTitle("Número de alumnos por curso");

    //Configuracion del tooltip para visualizar el número de alumnos
    Tooltip tooltip = new Tooltip();
    conf.setTooltip(tooltip);

    //Datos a visualizar
    DataSeries cursos = new DataSeries("Alumnos");
    //Se obtienen todos los cursos.
    listaCursos = Arrays.asList(cursosService.buscarTodos());
    listaCursos.forEach(curso -> {
        //Para cada curso se recupera el nombre y el total de alumnos que tiene
        DataSeriesItem dato = new DataSeriesItem(curso.getNombre(),curso.getAlumnos().size());
        cursos.add(dato);
    });
    conf.setSeries(cursos);
    graficoCursos.setVisibilityTogglingDisabled(true);
    return  graficoCursos;
}
```

En el gráfico de número de usuarios por rol, se utilizará el componente **Chart** con el tipo **COLUMN**, lo primero es configurar el título y subtítulo y después configurar los valores el eje X y el eje Y.

```
private Component graficoRolesUsuarios() {

    Chart graficoRoles = new Chart(ChartType.COLUMN);
    Configuration conf = graficoRoles.getConfiguration();

    //Configuracion del grafico
    conf.setTitle("Roles");
    conf.setSubTitle("Número de usuarios por rol");
    conf.getLegend().setEnabled(false);
    XAxis x = new XAxis();
    x.setType(AxisType.CATEGORY);
    conf.addxAxis(x);
    YAxis y = new YAxis();
    y.setTitle("Total usuarios");
    conf.addyAxis(y);
```

Se crea un **DataSeries** para almacenar el nombre del rol y el número de usuarios que tiene cada uno, para ello se hace uso del método **obtenerNumeroUsuariosRol** para calcularlos.

```
    // Datos a visualizar
    DataSeries roles = new DataSeries("Roles");
    PlotOptionsColumn plotOptionsColumn = new PlotOptionsColumn();
    plotOptionsColumn.setColorByPoint(true);
    roles.setPlotOptions(plotOptionsColumn);
    //Se obtienen todos los usuarios y roles
    listaUsuarios = Arrays.asList(usuariosService.buscarTodos());
    listaRoles = Arrays.asList(rolesService.buscarTodos());
    listaRoles.forEach(rol -> {
        //Para cada rol recuperamos su nombre y el número de usuarios
        DataSeriesItem dato = new
DataSeriesItem(rol.getAuthority(),obtenerNumeroUsuariosRol(listaUsuarios,rol));
        roles.add(dato);
    });

    conf.setSeries(roles);
    return graficoRoles;
}
```

```
private int obtenerNumeroUsuariosRol(List<Usuario> usuarios, Rol rol) {
    AtomicInteger numeroUsuario = new AtomicInteger();
    usuarios.forEach( usuario -> {
        if (usuario.getRoles().contains(rol)){
           numeroUsuario.getAndIncrement();
        }
    });
    return numeroUsuario.get();
}
```

En el gráfico de número de matrículas por mes, se utilizará el componente **Chart** con el tipo **AREASPLINE**, lo primero es configurar el título y subtitulo y después configurar los valores el eje X y el eje Y.

```
private Component graficoMatriculasPorMes(){

    //Se obtienen todas las matrículas
    listaMatriculas = Arrays.asList(matriculasService.buscarTodas());
    Chart graficoMatriculas = new Chart(ChartType.AREASPLINE);
    Configuration conf = graficoMatriculas.getConfiguration();

    //Configuracion del grafico
    conf.setTitle("Matrículas");
    conf.setSubTitle("Número matrículas por mes");
    XAxis x = new XAxis();
    x.setType(AxisType.CATEGORY);
    x.setCategories("Ene", "Feb", "Mar", "Abr", "May", "Jun", "Jul", "Ago", "Sep", "Oct", "Nov",
"Dic");;
    conf.addxAxis(x);
    YAxis y = new YAxis();
    y.setTitle("Número matrículas");
    conf.addyAxis(y);
```

Se crea un **DataSeries** para almacenar el número de matrículas por mes, para ello se hace uso del método **obtenerNumeroMatriculasPorMes** para calcularlos.

```
//Para cada mes se recuperan las matrículas
DataSeries matriculas = new DataSeries("Matriculas");
matriculas.setData(
    obtenerNumeroMatriculasMes(0),//Enero
    obtenerNumeroMatriculasMes(1),
    obtenerNumeroMatriculasMes(2),
    obtenerNumeroMatriculasMes(3),
```

```
    obtenerNumeroMatriculasMes(4),
    obtenerNumeroMatriculasMes(5),
    obtenerNumeroMatriculasMes(6),
    obtenerNumeroMatriculasMes(7),
    obtenerNumeroMatriculasMes(8),
    obtenerNumeroMatriculasMes(9),
    obtenerNumeroMatriculasMes(10),
    obtenerNumeroMatriculasMes(11)//Diciembre
);

conf.addSeries(matriculas);
return graficoMatriculas;
```

```
private int obtenerNumeroMatriculasMes(int mes) {
    AtomicInteger numeroMatriculas = new AtomicInteger();
    listaMatriculas.forEach( matricula -> {
        if (matricula.getFecha().getMonth()==mes){
            numeroMatriculas.getAndIncrement();
        }
    });
    return numeroMatriculas.get();
}
```

6.6. Spring Security en Vaadin y Login.

Spring Security es el módulo del proyecto Spring para incorporar seguridad de acceso a las aplicaciones hechas de Spring Boot. Permite controles de acceso por URL y roles de usuarios. A continuación, se explicará la configuración de Spring Security en Vaadin.

6.6.1. LoginView.

Lo primero que se tiene que hacer para gestionar el registro e inicio de sesión de los usuarios en la aplicación es crear el formulario de registro e inicio de sesión para ello se creara la vista **LoginView.java**.

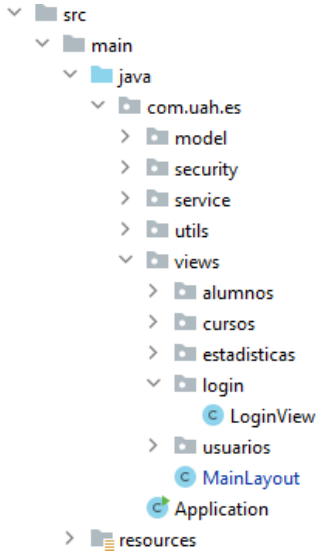


Ilustración 29. Paquetes y ficheros

Para el login se hace uso del componente **LoginForm** y para el registro de usuarios se utiliza el componente **FormLayout**. Primero que todo se inicializan y configuran los componentes.

```
public class LoginView extends VerticalLayout implements BeforeEnterObserver {

    // Servicio para comunicación con el backend
    IUsuariosService usuariosService;
    IRolesService rolesService;

    //Componentes visuales
    LoginForm login = new LoginForm();
    FormLayout registroForm = new FormLayout();
    Button registrarseBtn = new Button("Registrarse");
    Dialog registroDg = new Dialog();
    Notification notificacionOK = new Notification("", 3000);
    Notification notificacionKO = new Notification("", 3000);
    H2 tituloFormulario = new H2("Registro Usuario");

    // Inputs y btns del formulario de registro
    TextField nombre = new TextField("Nombre");
    EmailField correo = new EmailField ("Correo");
    PasswordField contrasena = new PasswordField("Contraseña");
    Button cancelarBtn = new Button("Cancelar");
    Button guardarBtn = new Button("Guardar");
    Binder<Usuario> binder = new Binder<>();

    Usuario usuario;
```

Debido a que el componente **LoginForm** tiene por defecto los textos en inglés, se tiene que hacer uso **LoginI18n** para configurarlo en español.

```
public LoginView(IUsuariosService usuariosService, IRolesService rolesService) {

    //Se inicializan los servicios
    this.usuariosService = usuariosService;
    this.rolesService = rolesService;

    //Se configuran el tamaño y posición el componente LoginView
    setSizeFull();
    setAlignItems(Alignment.CENTER);
    setJustifyContentMode(JustifyContentMode.CENTER);
    // Se configura el LoginView en idioma español
    login.setI18n(configurarLoginEsp());
    login.setAction("login");
```

```
private LoginI18n configurarLoginEsp() {

    final LoginI18n i18n = LoginI18n.createDefault();

    i18n.setHeader(new LoginI18n.Header());
    i18n.getHeader().setTitle("Gestión Cursos");
    i18n.getForm().setUsername("Usuario");
    i18n.getForm().setTitle("Inicio de sesión");
    i18n.getForm().setSubmit("Entrar");
    i18n.getForm().setPassword("Contraseña");
    i18n.getForm().setForgotPassword("Recordar contraseña");
    i18n.getErrorMessage().setTitle("Usuario/contraseña incorrecta");
    i18n.getErrorMessage().setMessage("Por favor vuelva a intentarlo");
    return i18n;
}
```

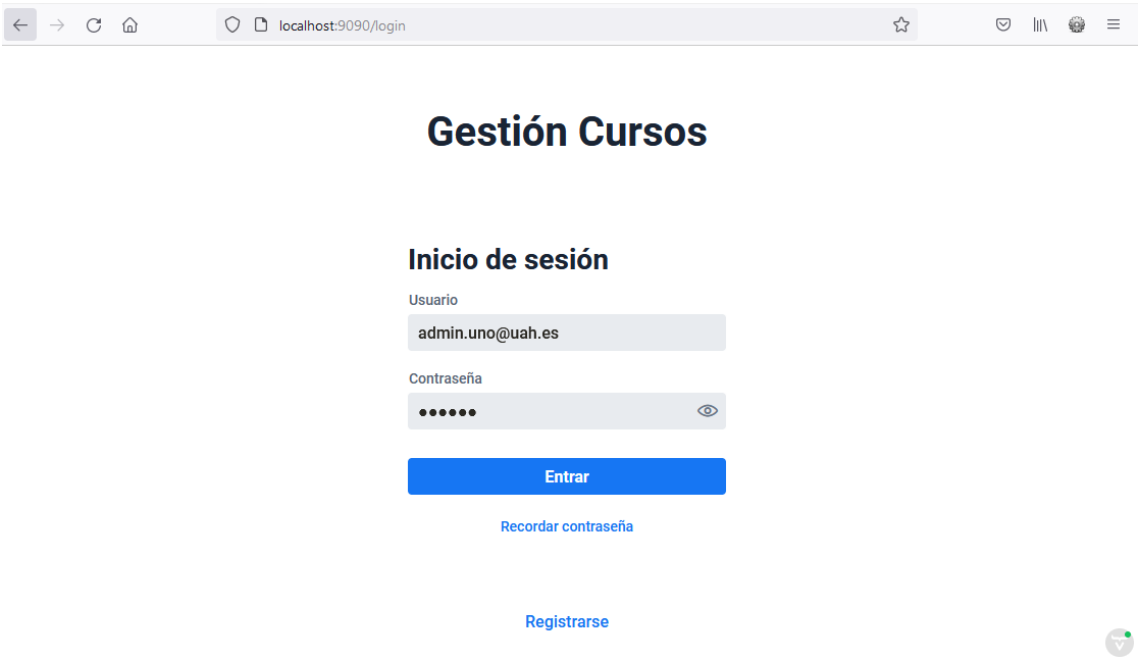


Ilustración 30. Vista Login

En el formulario de registro solo se va a disponer de tres datos de entrada, el nombre, correo y contraseña, primero se mapea el formulario con los atributos del objeto Usuario aplicando las validaciones necesarias en cada campo.

```
private void confirgurarFormRegistro () {

    usuario = new Usuario();
    guardarBtn.addThemeVariants(ButtonVariant.LUMO_PRIMARY);

    // Relacionamos los atributos del objeto Usuario con los campos del formulario
    binder.forField(nombre)
            .asRequired("Campo requerido")
            .bind(Usuario::getNombre,Usuario::setNombre);
    binder.forField(correo)
            .withValidator(new EmailValidator("Correo no válido"))
            .asRequired("Campo requerido")
            .bind(Usuario::getCorreo,Usuario::setCorreo);
    binder.forField(contrasena)
            .asRequired("Campo requerido")
            .bind(Usuario::getClave,Usuario::setClave);
```

El formulario tiene dos botones, el botón para cancelar y el botón para guardar el usuario y registrarse, para cada uno se añaden los **Listener**.

```
// Se configuran los listener para los botones del formulario
guardarBtn.addClickListener(event -> {
    if (binder.writeBeanIfValid(usuario)) {
        guardarUsuario(usuario);
    }
});
cancelarBtn.addClickListener(event -> {
    setUsuario();
    registroDg.close();
});
```

Se añaden los componentes al **Dialog** de registro.

```
//Se añaden los componentes visuales al formulario
    registroForm.add(nombre,correo,contrasena);
    registroForm.setResponsiveSteps(new FormLayout.ResponsiveStep("1px", 1));

    HorizontalLayout btnsLayout = new HorizontalLayout();
    btnsLayout.add(cancelarBtn, guardarBtn);
    btnsLayout.setPadding(true);
    btnsLayout.setJustifyContentMode(JustifyContentMode.END);

    //Se añaden los componentesn visuales al Dialog
    registroDg.add(tituloFormulario,registroForm,btnsLayout);
    registroDg.setMaxWidth("600px");
}
```

6.6.2. Configuración SpringSecurity.

El primer paso es añadir las dependencias de Spring Security en el POM.

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.security</groupId>
```

```
    <artifactId>spring-security-config</artifactId>
</dependency>
```

Para configurar Spring Security se crea un nuevo paquete con las siguientes clases.

- SecurityUtils.
- CustomAuthenticationProvider
- SecurityConfiguration.
- ConfigureUIServiceInitListener.

6.6.2.1. SecurityUtils.

En la clase **SecurityUtils.java** se implementan los siguientes métodos auxiliarles para hacer las verificaciones de usuarios y peticiones.

Método para verificar si la vista(clase) necesita un rol para poder visualizarla.

```
public final class SecurityUtils {

    public static boolean isAccessGranted(Class<?> securedClass) {

        Secured secured = AnnotationUtils.findAnnotation(securedClass, Secured.class);
        if (secured == null) {
            return true;
        }

        List<String> allowedRoles = Arrays.asList(secured.value());
        Authentication userAuthentication =
SecurityContextHolder.getContext().getAuthentication();
        return userAuthentication.getAuthorities().stream()
                .map(GrantedAuthority::getAuthority)
                .anyMatch(allowedRoles::contains);
    }
```

Método para verificar si el usuario que ha iniciado sesión tiene determinados roles.

```
public static boolean userHasRole(List<String> roles){

    Authentication userAuthentication = SecurityContextHolder.getContext().getAuthentication();
    return userAuthentication.getAuthorities().stream()
            .map(GrantedAuthority::getAuthority)
            .anyMatch(roles::contains);
}
```

Método para obtener el email del usuario.

```
public static String getEmailUser(){
    Authentication userAuthentication = SecurityContextHolder.getContext().getAuthentication();
    return userAuthentication.getName();
}
```

Método para verificar si es una petición interna de Vaadin.

```
static boolean isFrameworkInternalRequest(HttpServletRequest request) {
    final String parameterValue =
request.getParameter(ApplicationConstants.REQUEST_TYPE_PARAMETER);
    return parameterValue != null
            && Stream.of(RequestType.values())
            .anyMatch(r -> r.getIdentifier().equals(parameterValue));
}
```

Método para verificar si el usuario actual ha iniciado sesión.

```
static boolean isUserLoggedIn() {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    return authentication != null
            && !(authentication instanceof AnonymousAuthenticationToken)
```

```
            && authentication.isAuthenticated();
}
```

6.6.2.2. CustomAuthenticationProvider.

Esta clase es la encargada de tomar un objeto de solicitud de autenticación y decidir si es o no válida para ellos se implementa el método **authenticate** para consultar el **UsuarioService** y verificar si existe un usuario con esos datos y así devolver el objeto **Authentication**.

```
@Component
public class CustomAuthenticationProvider implements AuthenticationProvider {

    @Autowired
    private IUsuariosService usuariosService;
    public CustomAuthenticationProvider() {
        super();
    }

    @Override
    public Authentication authenticate(final Authentication authentication) throws AuthenticationException {

        final String usuario = authentication.getName();
        String password = authentication.getCredentials().toString();

        Usuario usuarioLogueado = usuariosService.login(usuario, password);
        if (usuarioLogueado != null) {
            final List<GrantedAuthority> grantedAuths = new ArrayList<GrantedAuthority>();
            for (Rol rol : usuarioLogueado.getRoles()) {
                grantedAuths.add(new SimpleGrantedAuthority(rol.getAuthority()));
            }
            final UserDetails principal = new User(usuario, password, grantedAuths);
            final Authentication auth = new UsernamePasswordAuthenticationToken(principal, password, grantedAuths);
            return auth;
        }
        return null;
    }

    @SuppressWarnings("rawtypes")
    @Override
    public boolean supports(final Class authentication) {
        return authentication.equals(UsernamePasswordAuthenticationToken.class);
    }

}
```

6.6.2.3. SecurityConfiguration.

En esta clase se configura el **AuthenticationManagerBuilder** que se encarga de manejar la autenticación de usuarios.

```
@EnableWebSecurity
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    private static final String LOGIN_PROCESSING_URL = "/login";
    private static final String LOGIN_FAILURE_URL = "/login?error";
    private static final String LOGIN_URL = "/login";
    private static final String LOGOUT_SUCCESS_URL = "/login";

    @Autowired
    private CustomAuthenticationProvider authProvider;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.authenticationProvider(authProvider);
    }
```

Configurando **HttpSecurity** se permite solo peticiones de usuarios no autenticados a la url de login.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
```

```
    http.csrf().disable()
            // Restringir el acceso a la aplicación
            .authorizeRequests()
            // Permitir todas las solicitudes internas.
            .requestMatchers(SecurityUtils::isFrameworkInternalRequest).permitAll()
            // Permitir todas las solicitudes de usuarios registrados
            .anyRequest().authenticated()
            // Configurar la página de login
            .and().formLogin()
            .loginPage(LOGIN_URL).permitAll()
            .loginProcessingUrl(LOGIN_PROCESSING_URL)
            .failureUrl(LOGIN_FAILURE_URL)
            // Configurar la página de logout
            .and().logout().logoutSuccessUrl(LOGOUT_SUCCESS_URL);
}
```

A continuación, se configura **WebSecurity** para que los recursos internos de Vaadin que se necesitan no se vean afectados por la configuración de seguridad anterior.

```
@Override
public void configure(WebSecurity web) {

    web.ignoring().antMatchers(
            "/VAADIN/**",
            "/favicon.ico",
            "/robots.txt",
            "/manifest.webmanifest",
            "/sw.js",
            "/offline.html",
            "/icons/**",
            "/images/**",
            "/styles/**",
            "/h2-console/**");
}
```

6.6.2.4. ConfigureUIServiceInitListener.

Finalmente, se configura la navegación del enrutador registrando un **Listener** que verifique los permisos. Esto es muy importante ya que **Spring Security** no conoce el comportamiento de una aplicación de una sola página como Vaadin.

```
@Component
public class ConfigureUIServiceInitListener implements VaadinServiceInitListener {

    @Override
    public void serviceInit(ServiceInitEvent event) {
        event.getSource().addUIInitListener(uiEvent -> {
            final UI ui = uiEvent.getUI();
            ui.addBeforeEnterListener(this::beforeEnter);
        });
    }

    /**
     * Redirige al usuario si no está autorizado para acceder a la vista..
     */
    private void beforeEnter(BeforeEnterEvent event) {
        if(!SecurityUtils.isAccessGranted(event.getNavigationTarget())) {
            if(SecurityUtils.isUserLoggedIn()) {
                event.rerouteToError(NotFoundException.class);
            } else {
               event.rerouteTo(LoginView.class);
            }
        }
    }
}
```

6.6.3. Configurar acceso a las vistas y componentes.

Hasta ahora las vistas eran visibles sin importar el rol que tenga el usuario es decir todos los usuarios autenticados pueden ver todas las vistas. Para ello se configura mediante @Secure los roles que pueden acceder a la vista.

Si no se indica la etiqueta **@Secure** cualquier usuario autenticado puede acceder.

```
@PageTitle("Cursos")
@Route(value = "cursos", layout = MainLayout.class)
@RouteAlias(value = "", layout = MainLayout.class)
public class CursosView extends Div {
```

Los usuarios con rol "Admin" puede acceder a la vista **UsuariosView** y **AlumnosView**.

```
@PageTitle("Usuarios")
@Route(value = "usuarios", layout = MainLayout.class)
@Secured("Admin")
public class UsuariosView extends Div {
```

```
@PageTitle("Alumnos")
@Route(value = "alumnos", layout = MainLayout.class)
@Secured("Admin")
public class AlumnosView extends Div {
```

Los usuarios con rol "Admin" y "Profesor" pueden acceder a la vista **EstadisticasView**.

```
@PageTitle("Estadísticas")
@Secured({"Admin","Profesor"})
@Route(value = "estadisticas", layout = MainLayout.class)
public class EstadisticasView extends Div {
```

Para no mostrar los enlaces de acceso a las vistas dentro del menú lateral se incluye una validación para comprobar el permiso de acceso a la vista del usuario que ha iniciado sesión. Para ello se utiliza el método **isAccessGranted** definido en **SecurityUtils.java.**

```
private List<Tab> createMenuItems() {
    MenuItemInfo[] menuItems = new MenuItemInfo[]{
            new MenuItemInfo("Cursos", "la la-book-open", CursosView.class),
            new MenuItemInfo("Alumnos", "la la-user-graduate", AlumnosView.class),
            new MenuItemInfo("Usuarios", "la la-users", UsuariosView.class),
            new MenuItemInfo("Estadísticas", "la la-chart-area", EstadisticasView.class),
    };

    List<Tab> tabs = new ArrayList<>();
    for (MenuItemInfo menuItemInfo : menuItems) {
        //Verificamos si el usuario tiene el rol que permita visualizar la vista
        if (SecurityUtils.isAccessGranted(menuItemInfo.getView())){
            tabs.add(createTab(menuItemInfo));
        }
    }
    return tabs;
}
```

Además de controlar el acceso a las vistas también se puede controlar que componentes o partes de componentes se pueden visualizar dependiendo el rol del usuario.

En la vista de Cursos solo el usuario Administrador puede añadir, modificar o eliminar un curso, para controlarlo se utiliza el método **userHasRole** definido en **SecurityUtils.java.**

Utilizando el método **setVisible** de los componentes **Button** y **PaginatedGrid** se puede mostrar u ocultar basándose en el rol del usuario.

```
//Boton solo será visible para el rol Admin
nuevoCursoBtn.setVisible(userHasRole(Collections.singletonList(Configuracion.ROL_ADMIN)));
```

```
grid.addComponentColumn(item -> {
    Icon editarIcon = new Icon(VaadinIcon.EDIT);
    editarIcon.setColor("green");
    editarIcon.getStyle().set("cursor", "pointer");
    editarIcon.setSize("18px");
    editarIcon.addClickListener(e -> editarCurso(item));
    return editarIcon;
})
.setKey("editar")
.setHeader("Editar")
.setTextAlign(ColumnTextAlign.CENTER)
.setAutoWidth(true)
.setVisible(userHasRole(Collections.singletonList(Configuracion.ROL_ADMIN)));//Solo visible para
el rol Admin
```

```
grid.addComponentColumn(item -> {
    Icon editarIcon = new Icon(VaadinIcon.TRASH);
    editarIcon.setColor("red");
    editarIcon.getStyle().set("cursor", "pointer");
    editarIcon.setSize("18px");
    editarIcon.addClickListener(e -> eliminarCurso(item));
    return editarIcon;
})
.setKey("eliminar")
.setHeader("Eliminar")
.setTextAlign(ColumnTextAlign.CENTER)
.setAutoWidth(true)
.setVisible(userHasRole(Collections.singletonList(Configuracion.ROL_ADMIN)));//Solo visible para
el rol Admin
```

Si accede un usuario con rol administrador tiene la siguiente visibilidad.

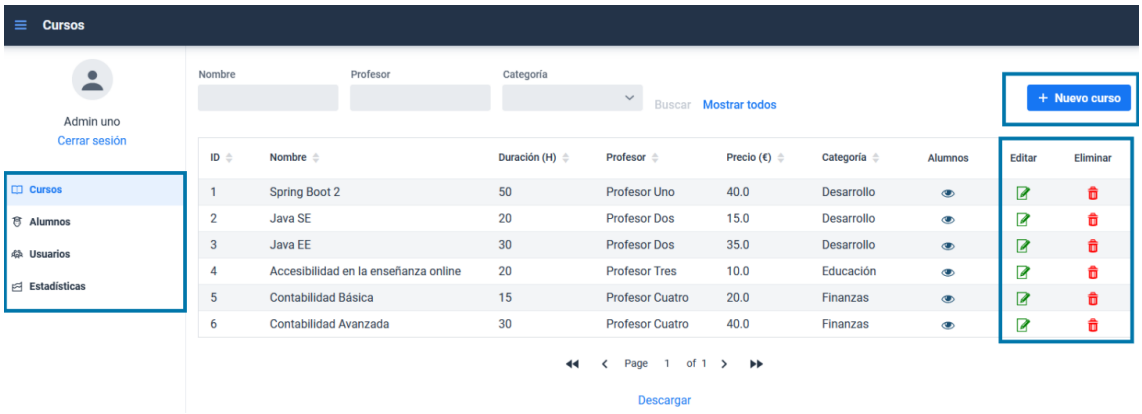


Ilustración 31. Visualización del rol administrador

Si accede un usuario con rol profesor tiene la siguiente visibilidad.

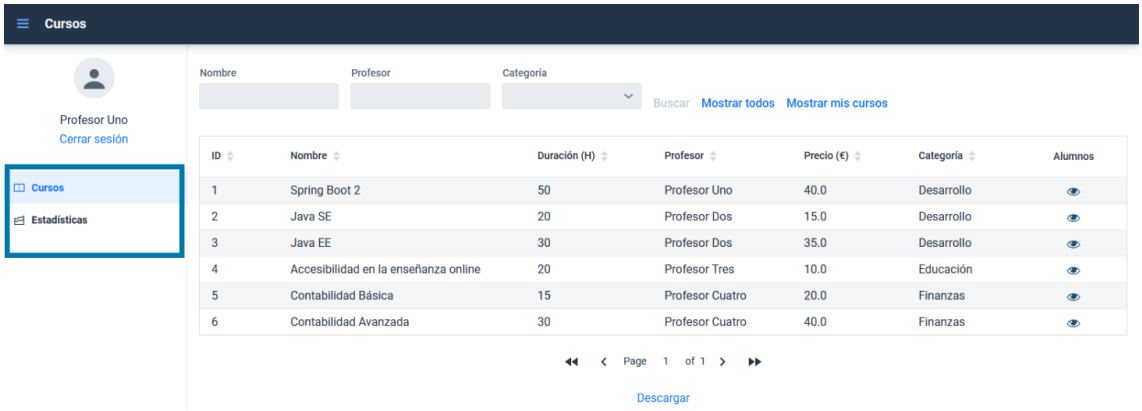


Ilustración 32. Visualización del rol profesor

Si accede un usuario con rol profesor tiene la siguiente visibilidad.

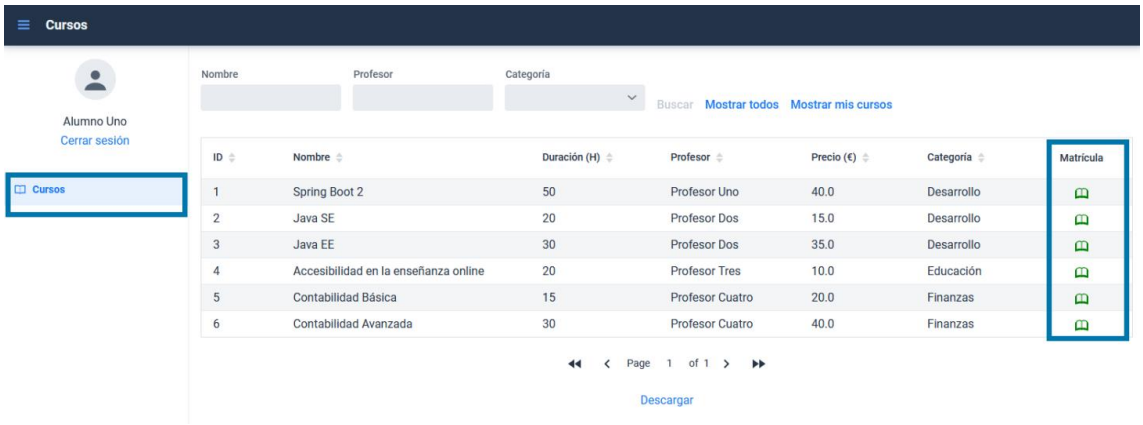


Ilustración 33. Visualización del rol alumno

7. MANUAL DE USUARIO

En la aplicación existen usuarios con tres roles diferentes, a continuación, se explicará el funcionamiento para cada rol.

7.1. Rol administrador.

Se inicia sesión con un usuario con rol administrador.

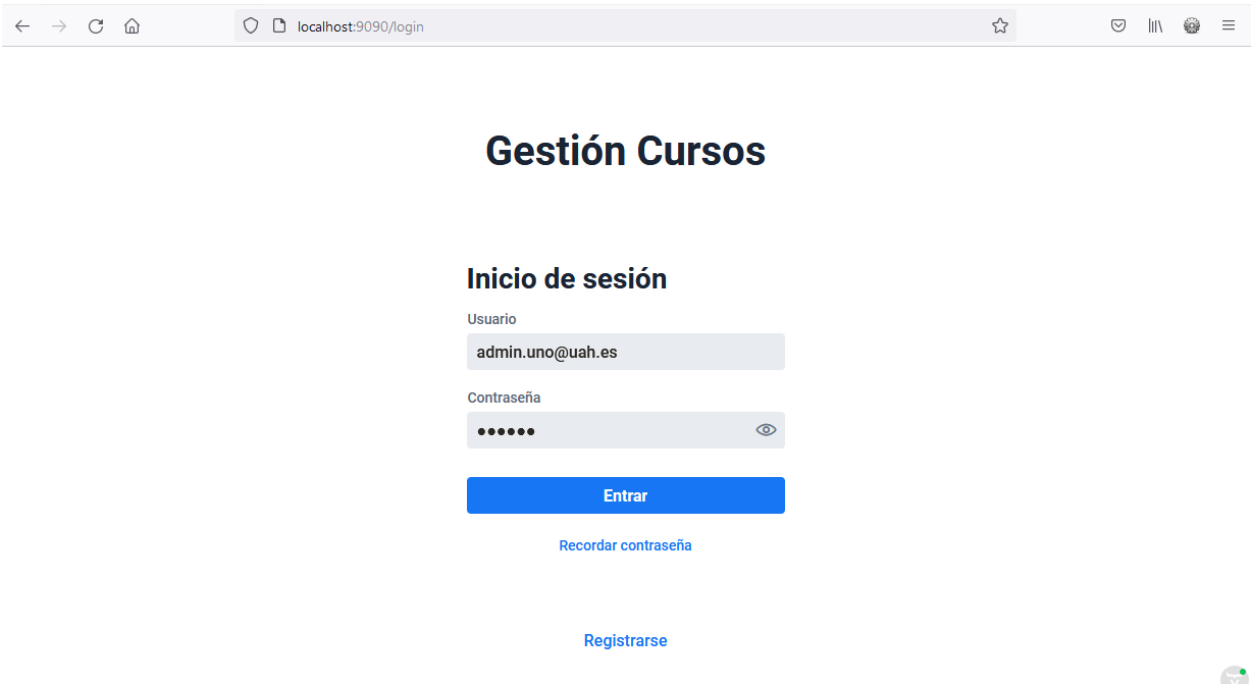


Ilustración 34. Login de usuario administrador

Con el rol administrador se pueden ver todas las vistas en el menú lateral.

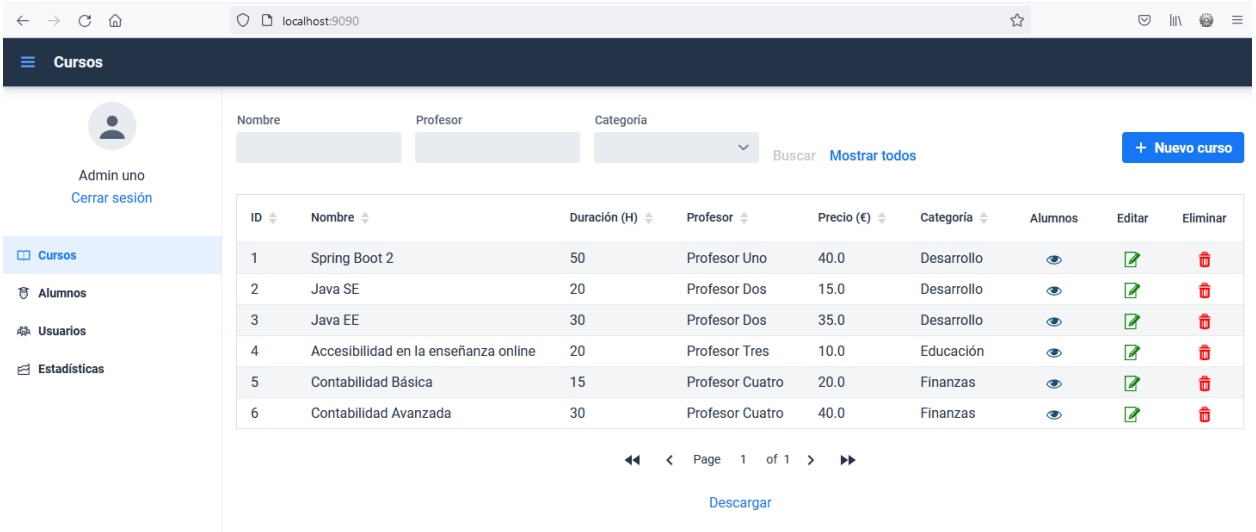


Ilustración 35. Visualización de cursos

Con el buscador situado en la parte superior se puede aplicar diferentes filtros para acotar la búsqueda de cursos, en este caso se va a filtrar por categoría.

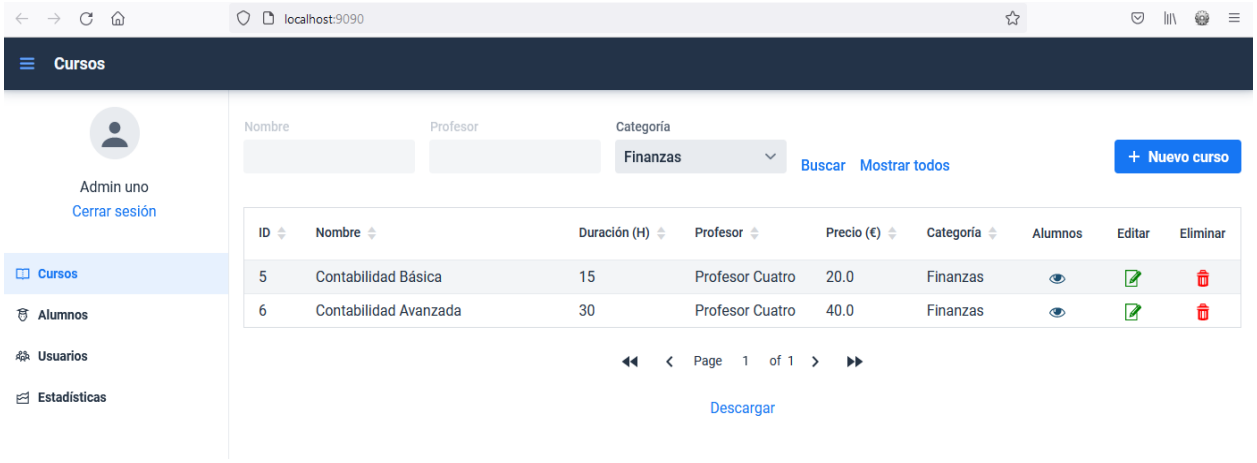


Ilustración 36. Cursos filtrados por categoría

Este rol también puede dar de alta un nuevo curso rellenando el formulario con los datos necesarios.

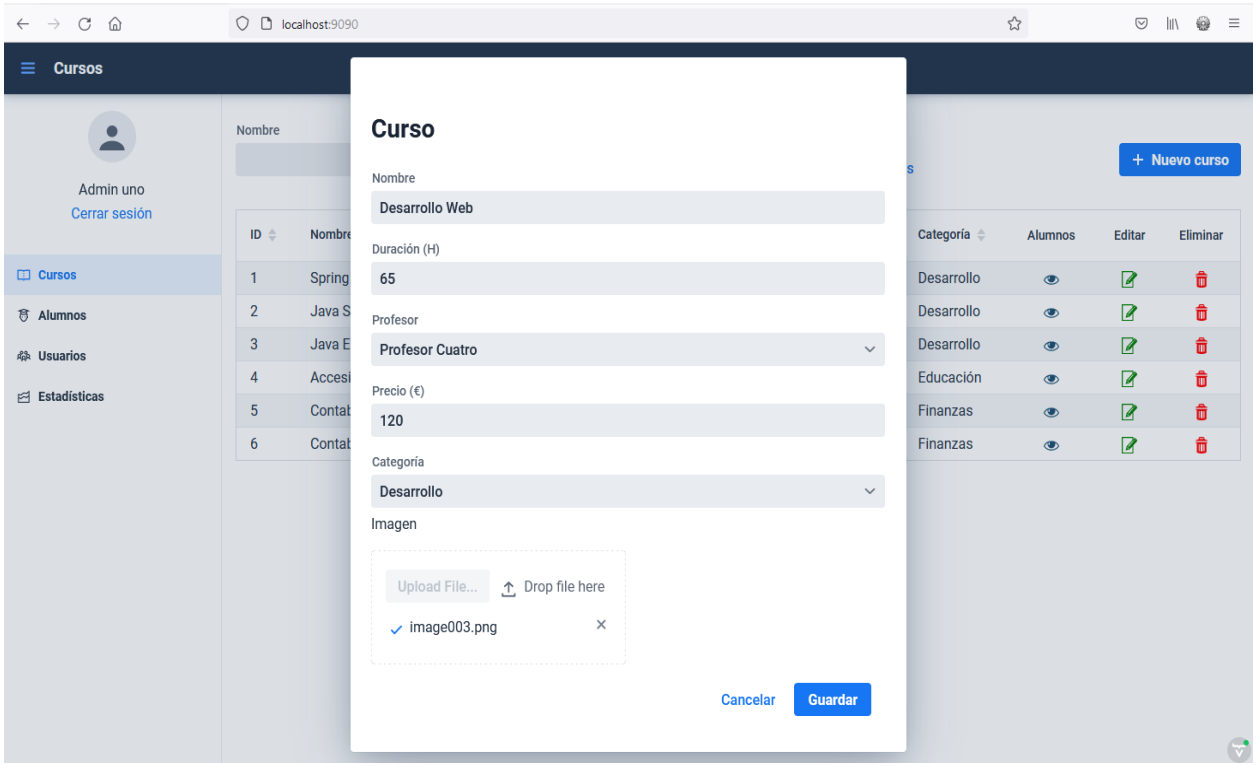


Ilustración 37. Formulario nuevo curso

En la parte inferior izquierda se puede visualizar el resultado del alta del curso, esa notificación se verá en pantalla durante 3 segundos.

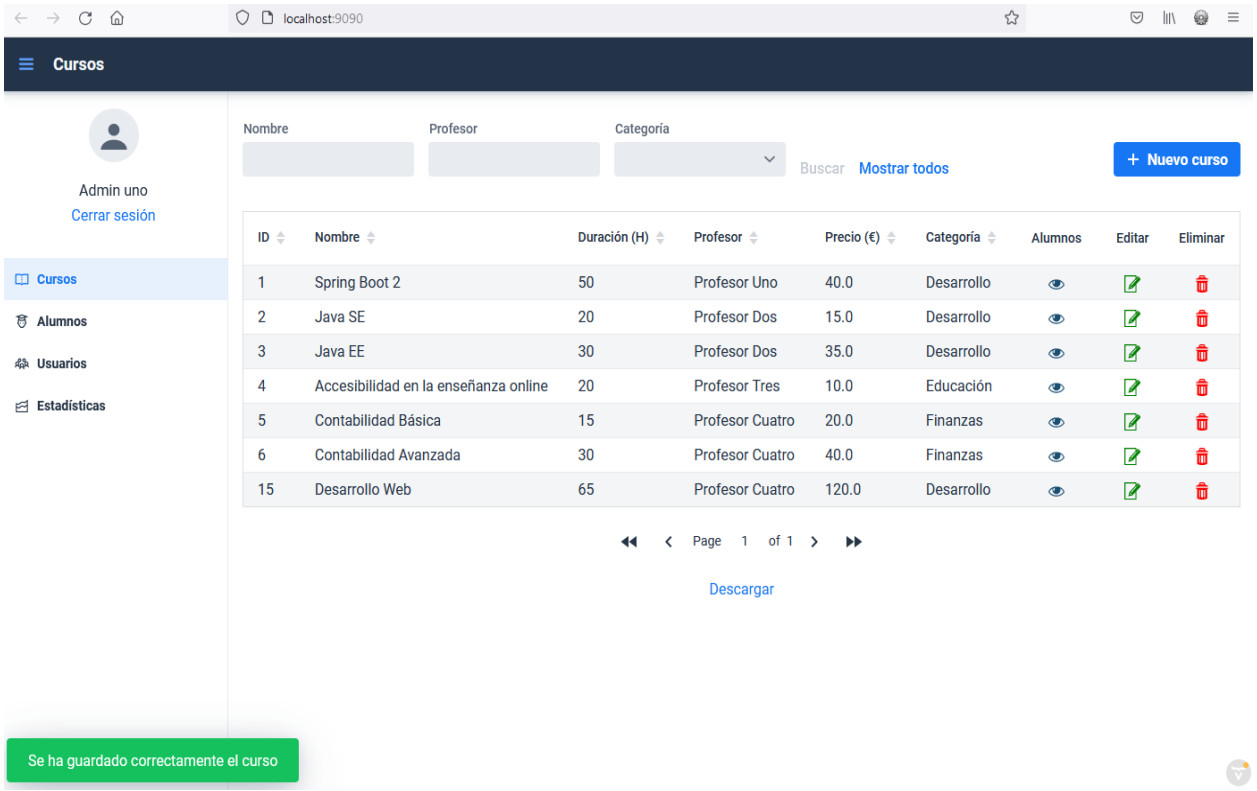


Ilustración 38. Confirmación de operación realizada correctamente

Para modificar los datos de un curso se tiene que presionar el icono de la columna editar y cambiar los datos en el formulario.

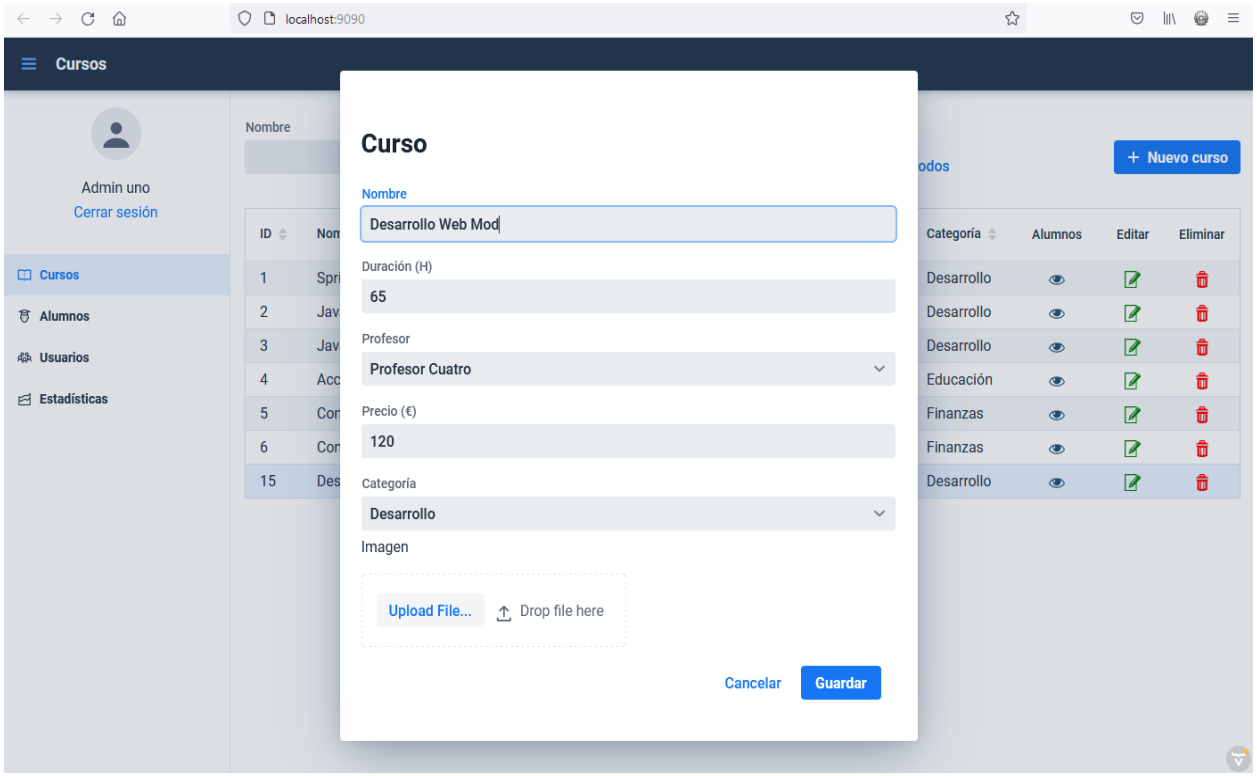


Ilustración 39. Modificación de datos de curso

Para eliminar un curso solo se tiene que presionar el icono de la columna Eliminar y confirmarlo en el modal.

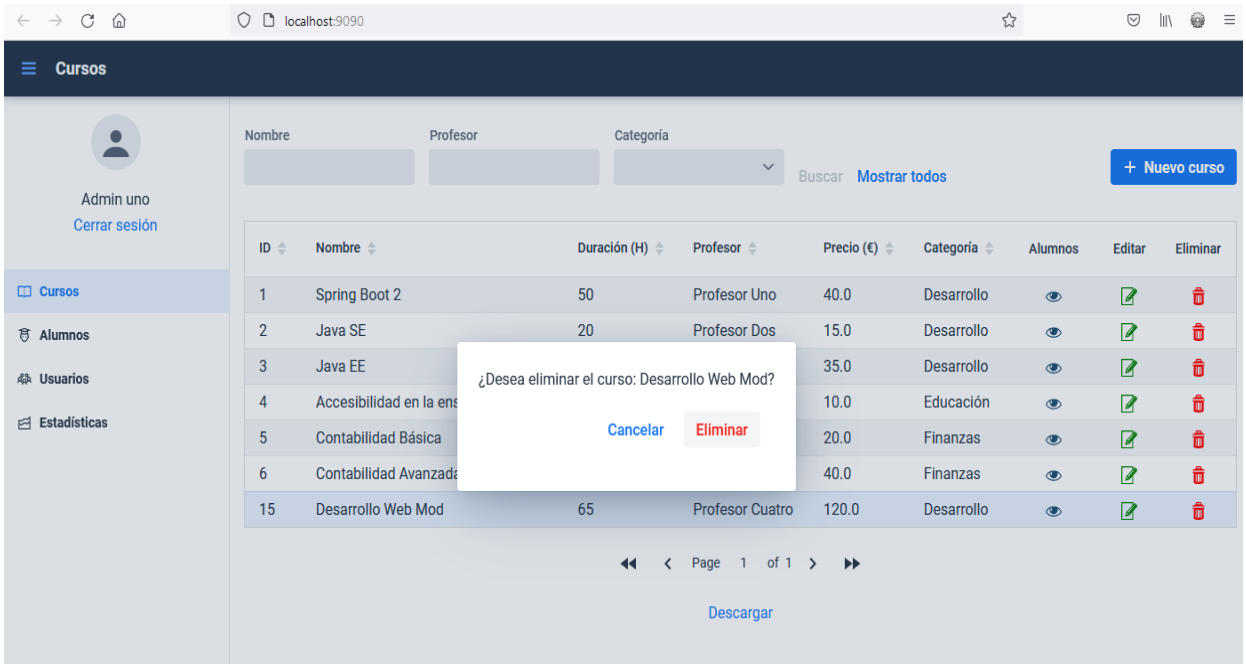


Ilustración 40. Confirmación eliminación de un curso

Para visualizar los alumnos matriculados en un curso se tiene que presionar el icono de la columna Alumnos.

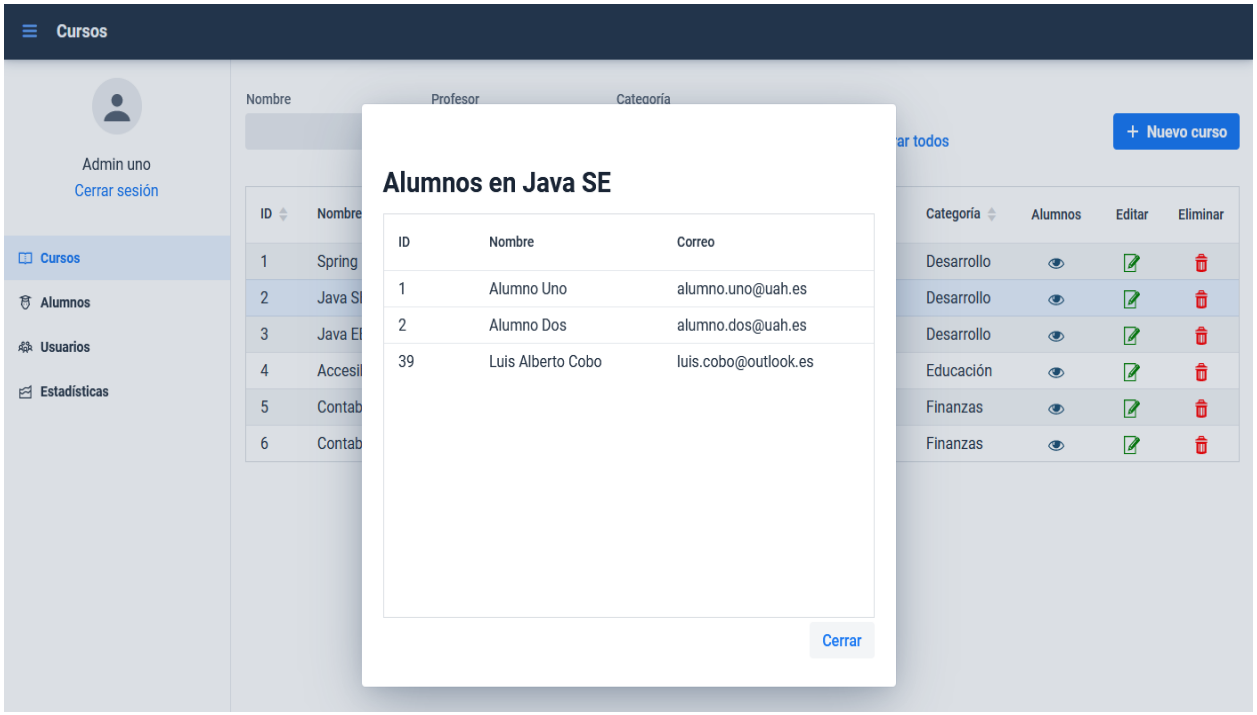


Ilustración 41. Listado de alumnos

En la vista alumnos también se puede aplicar filtros, ver los cursos del alumno, editar y eliminar un alumno.

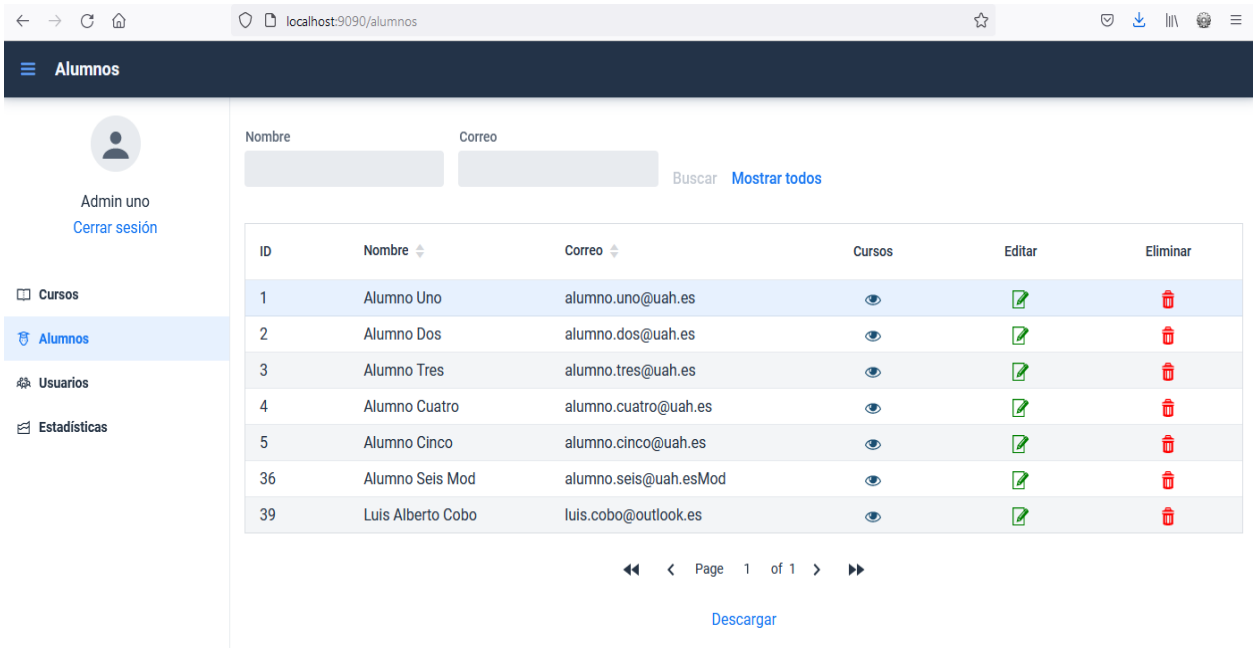


Ilustración 42. Vista alumnos

Para editar un alumno se tiene que presionar el icono de la columna Editar y modificar los datos en el formulario.

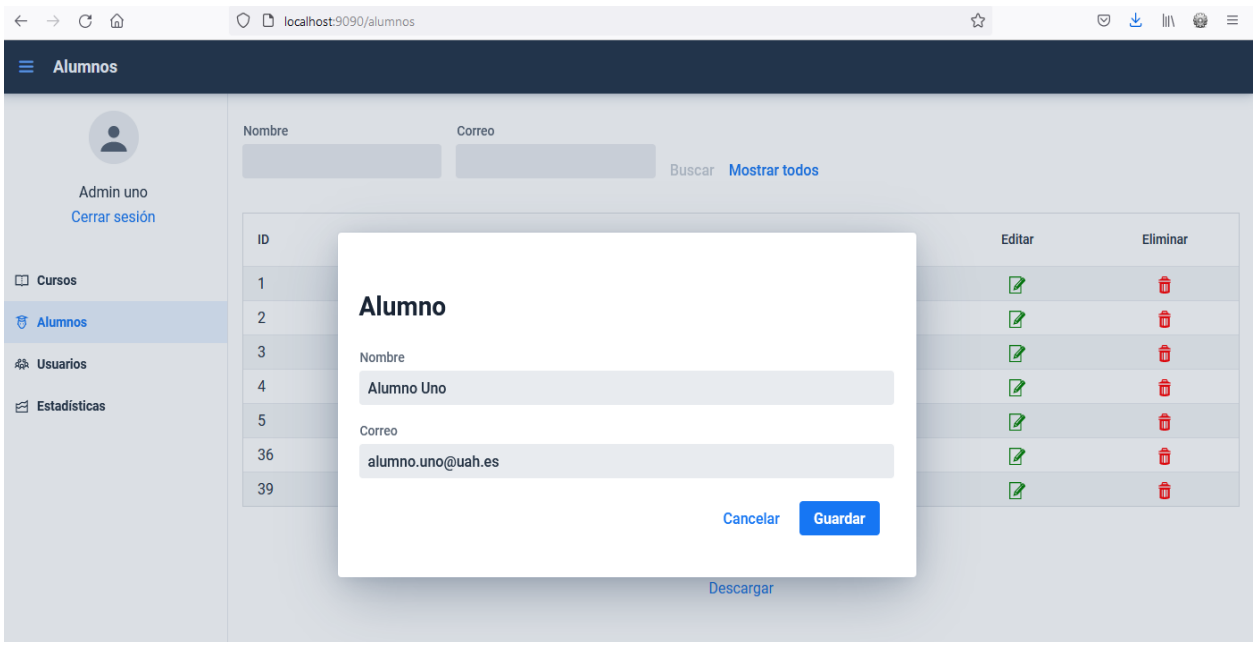


Ilustración 43. Modificación datos alumno

Presionando el icono de la columna Cursos se puede visualizar los cursos en los que está matriculado el alumno.

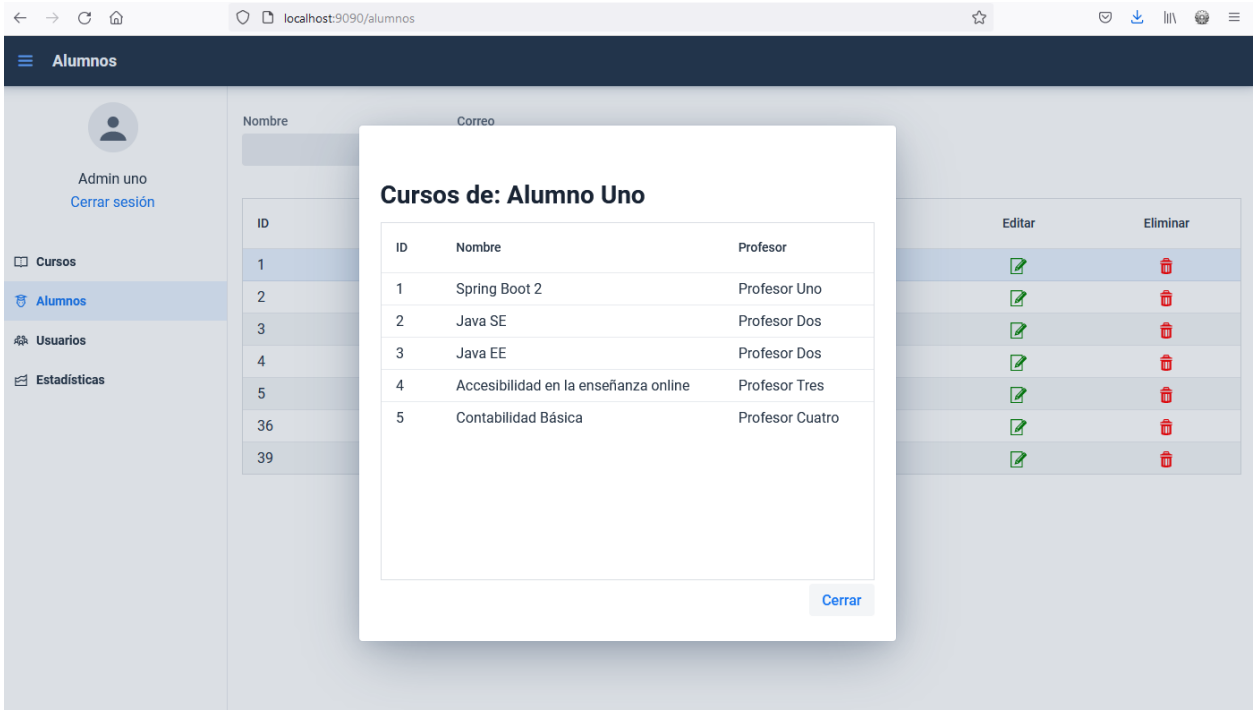


Ilustración 44. Listado de cursos de un alumno

En la vista de usuarios se puede filtrar, dar de alta, editar y eliminar un usuario.

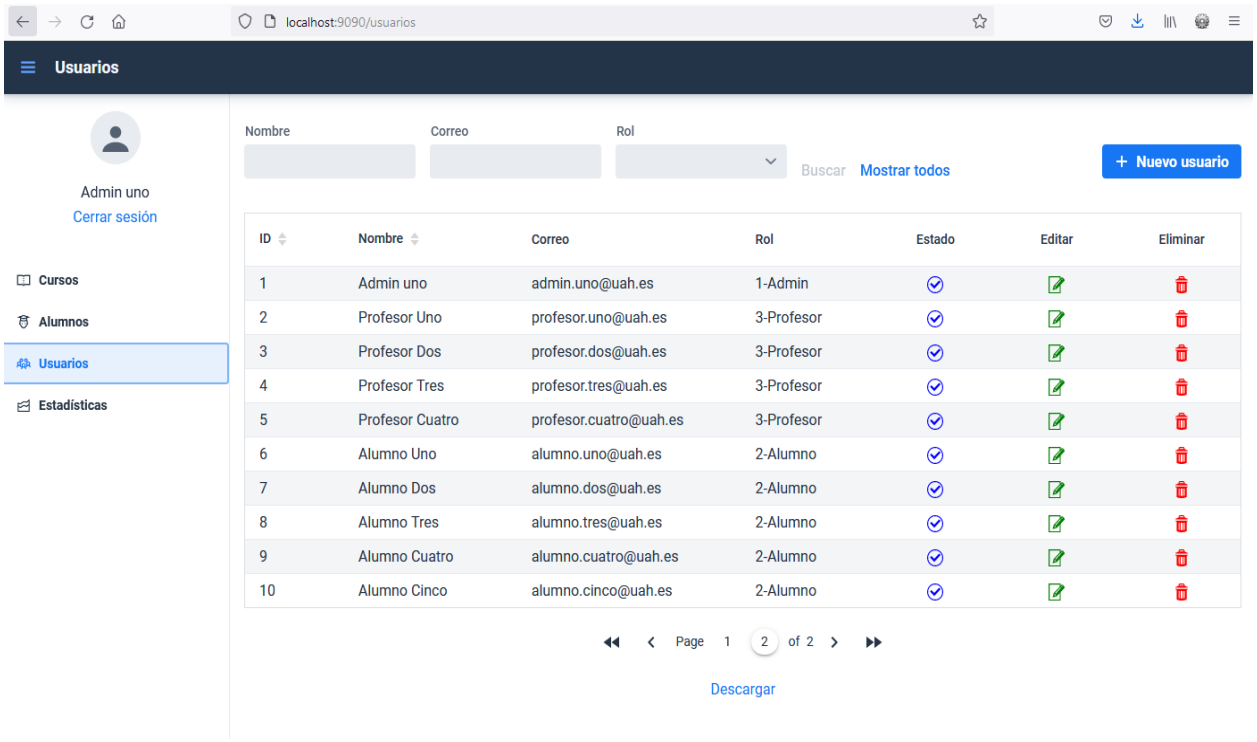


Ilustración 45. Vista usuarios

Para dar de alta un nuevo usuario se tiene que rellenar los datos del formulario, y en este caso se tiene que seleccionar el rol que tendrá asignado el nuevo usuario.

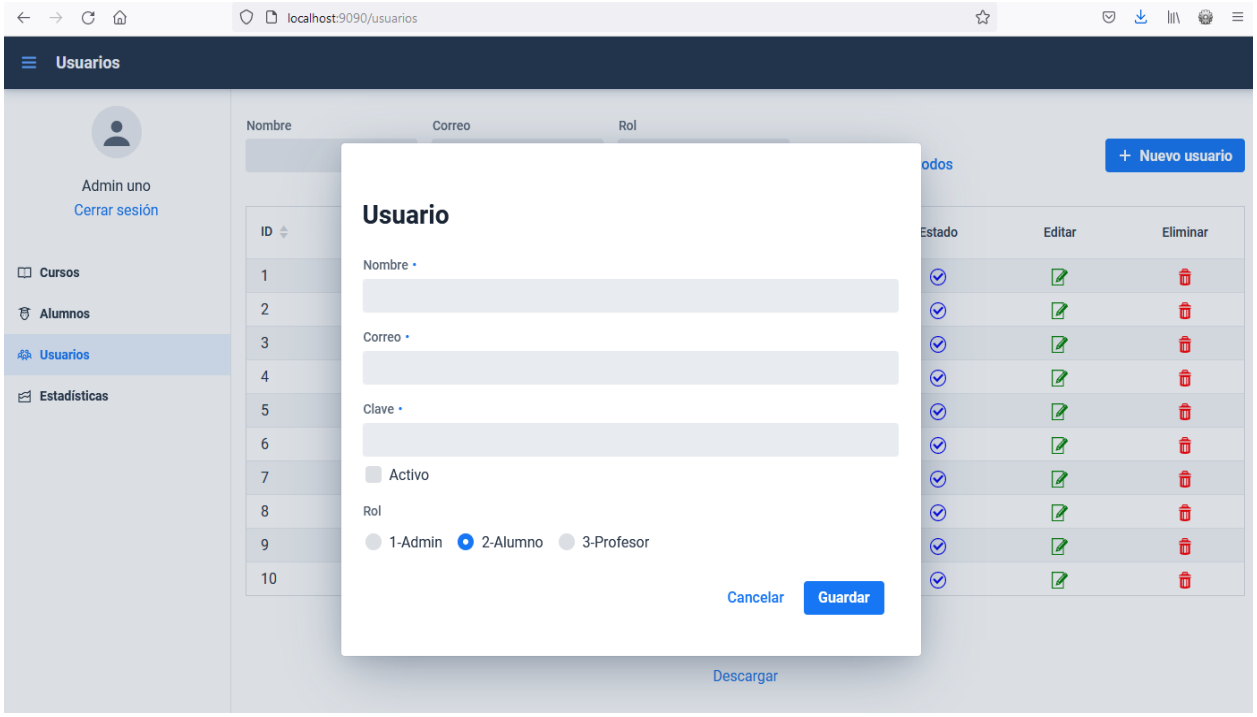


Ilustración 46. Formulario nuevo usuario

En la modificación de un usuario ya existente, el rol es el único dato que no se puede modificar.

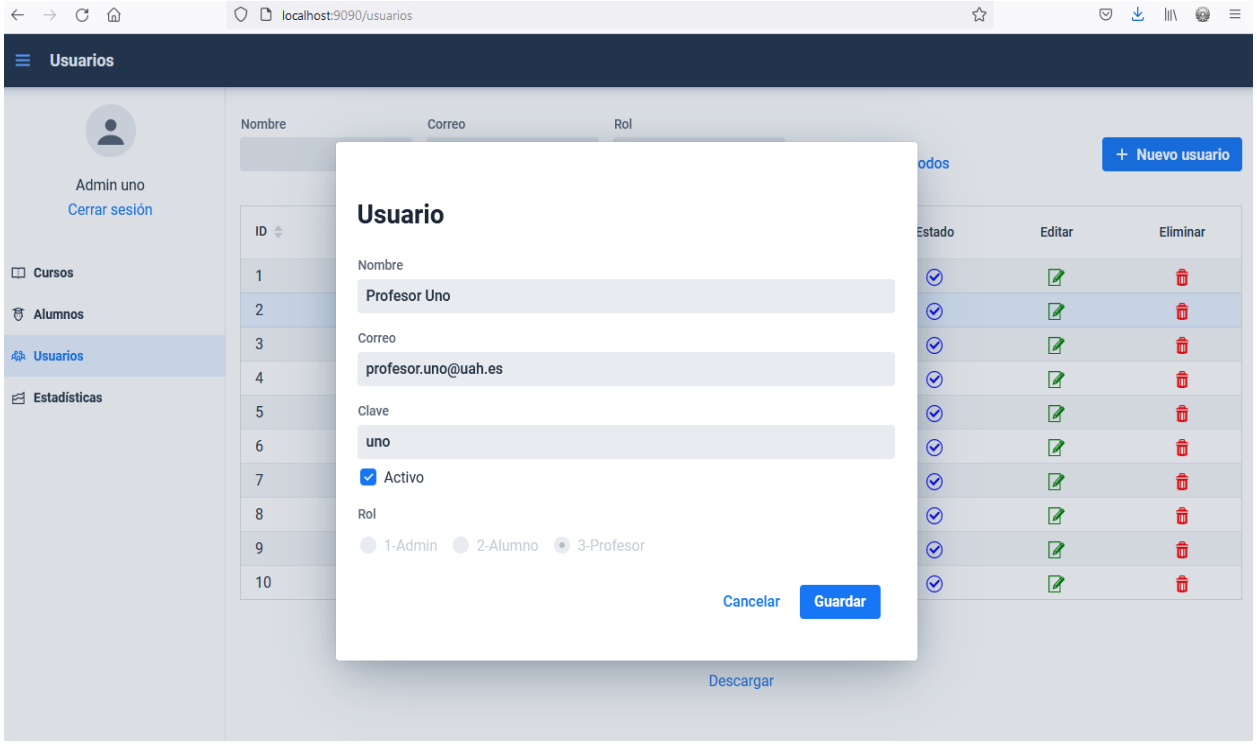


Ilustración 47. Formulario modificación usuario

En la vista de estadísticas se visualizan los tres gráficos con los datos de número de alumnos por curso, número de usuarios por rol y número de matrículas por mes.

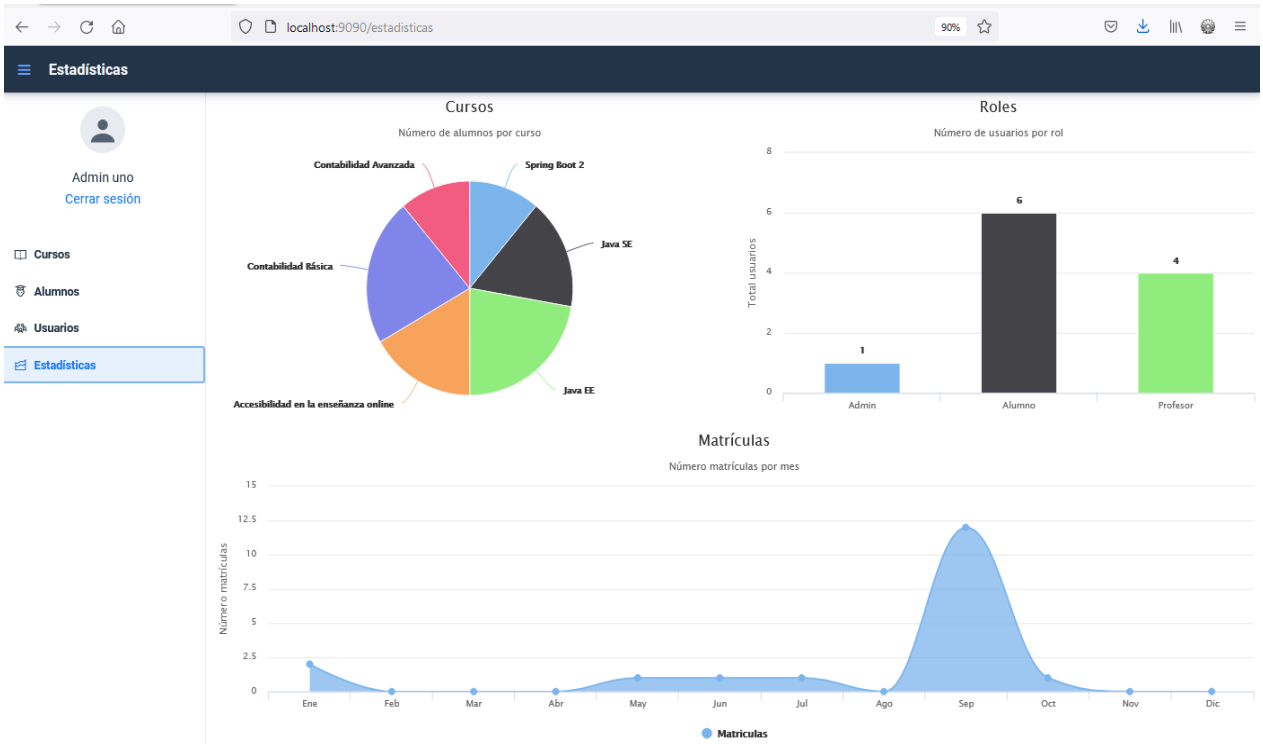


Ilustración 48. Vista estadísticas

7.2. Rol profesor.

Se inicia sesión con un usuario con rol profesor.

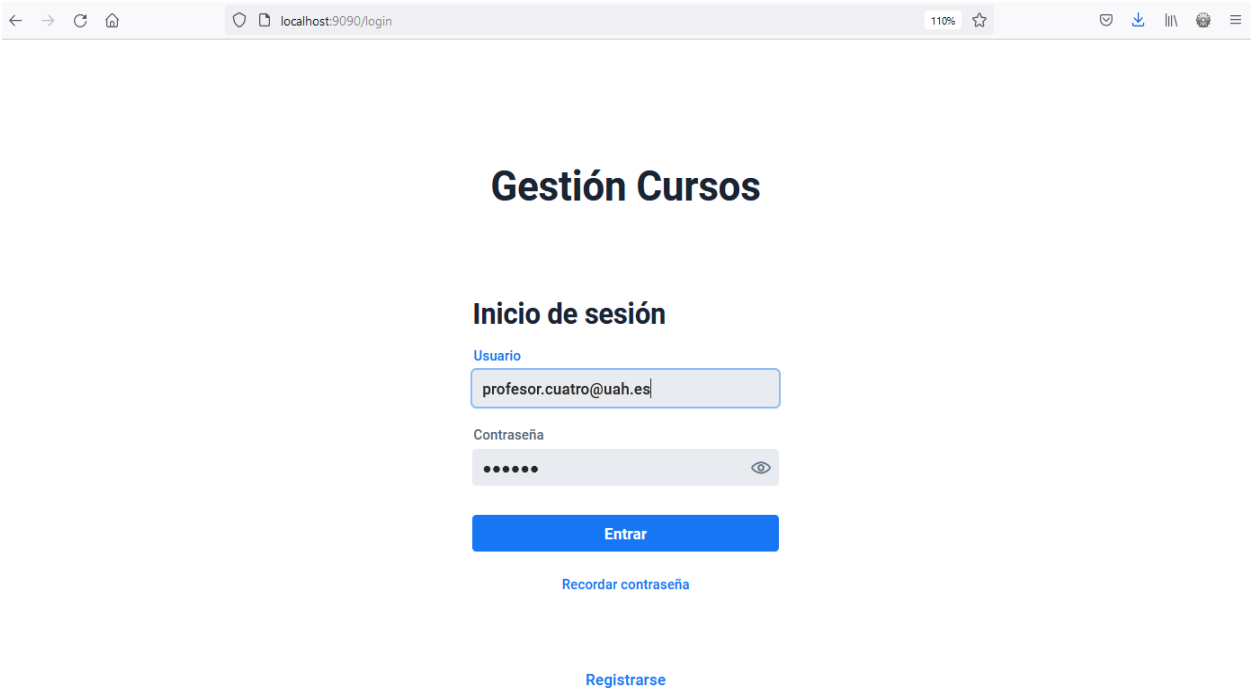


Ilustración 49. Login usuario profesor

El rol profesor solo puede visualizar las vistas de cursos y estadísticas. En la vista de estadísticas a diferencias del rol administrador tiene botón para filtrar por los cursos que imparte y no puede ni editar ni eliminar un curso.

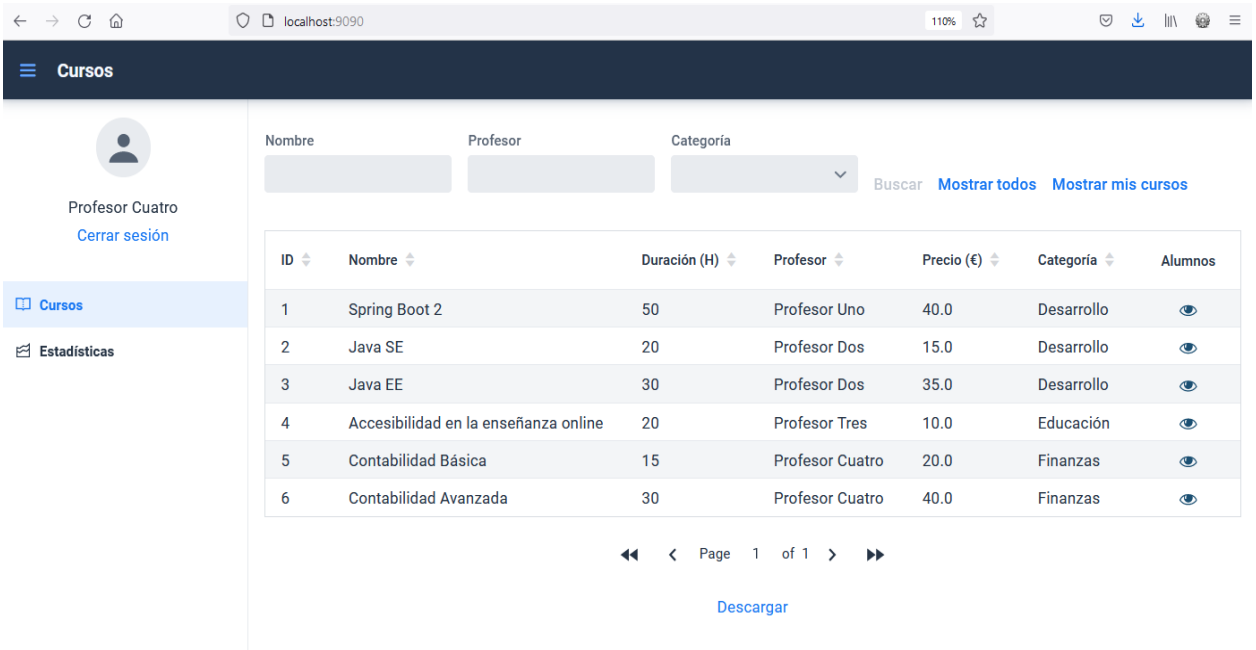


Ilustración 50. Vista cursos con rol profesor

Se filtra solo por los cursos que imparte el profesor.

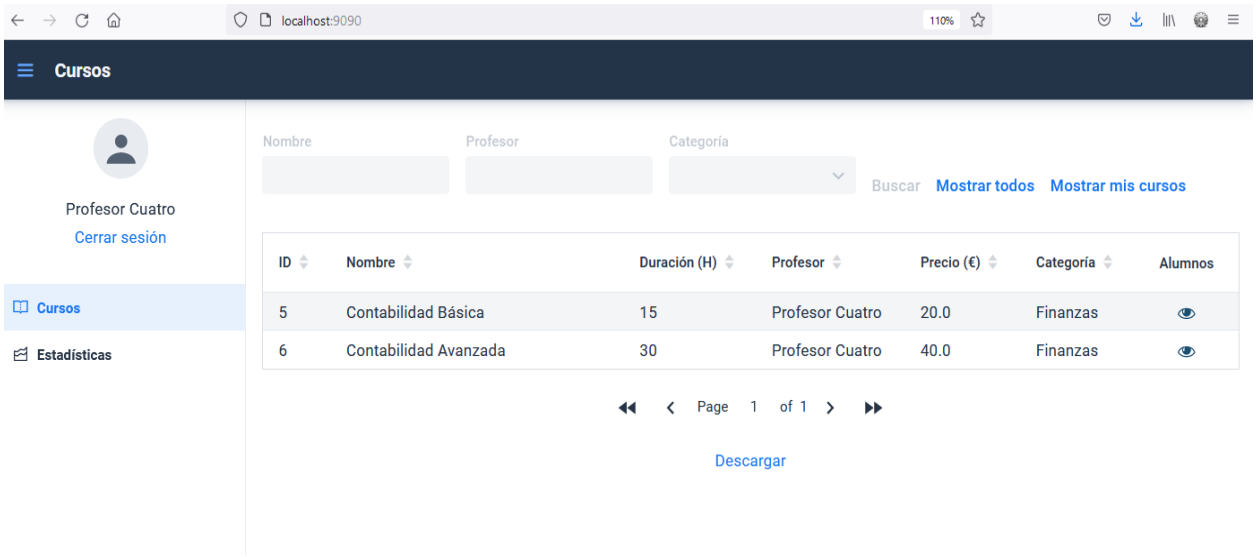


Ilustración 51. Filtro cursos del profesor

Se visualizan los alumnos matriculados en un curso.

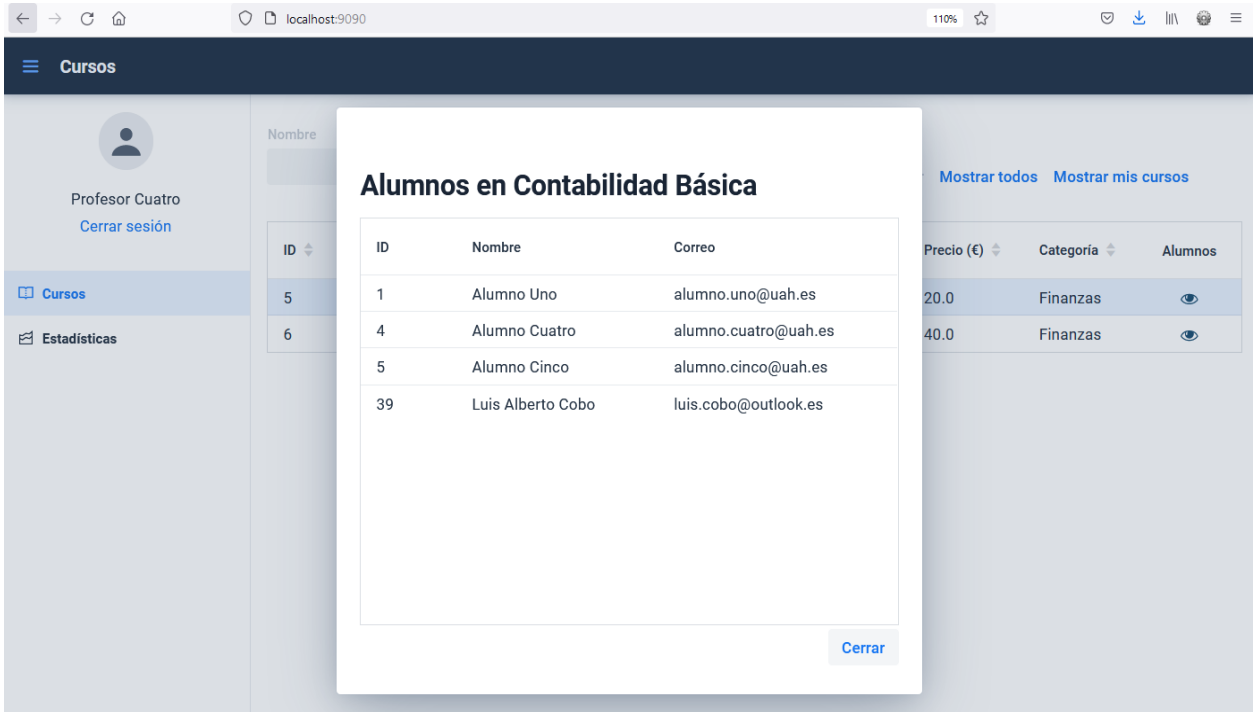


Ilustración 52. Alumnos de un curso

En la vista estadísticas el rol profesor solo puede visualizar el gráfico de número de alumnos por curso y en este caso solo aparecerán los cursos que imparte el profesor.

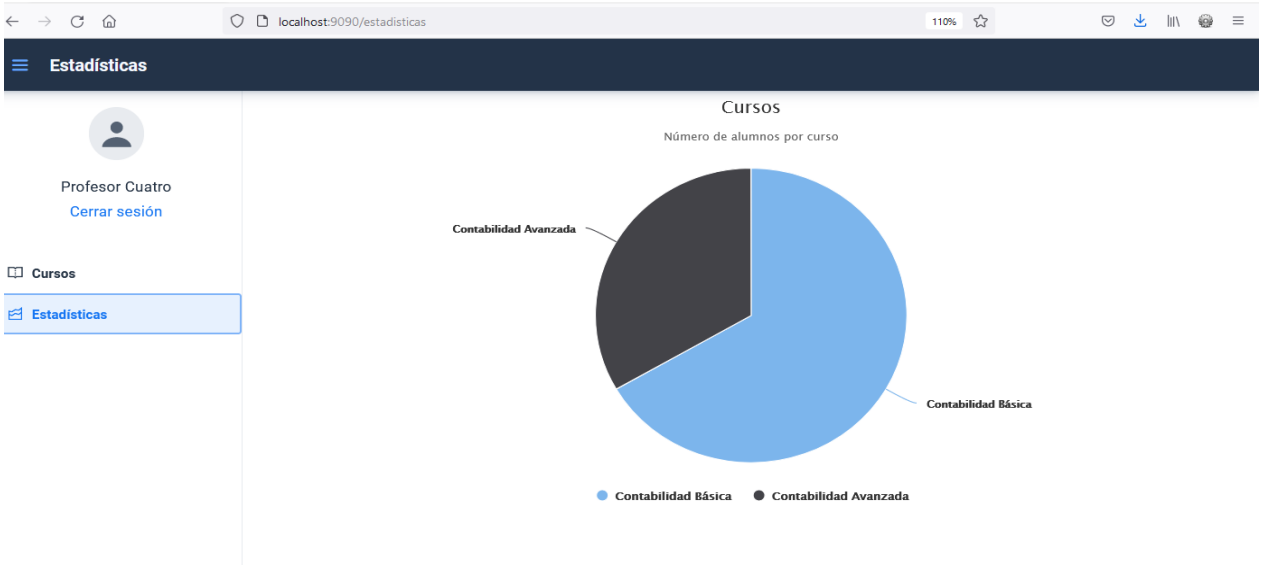


Ilustración 53. Vista de estadísticas con rol profesor

7.2. Rol alumno.

Utilizando la opción de registro de nuevo usuario se asignará automáticamente el rol de alumno, para ello se tiene que rellenar los datos del nuevo usuario.

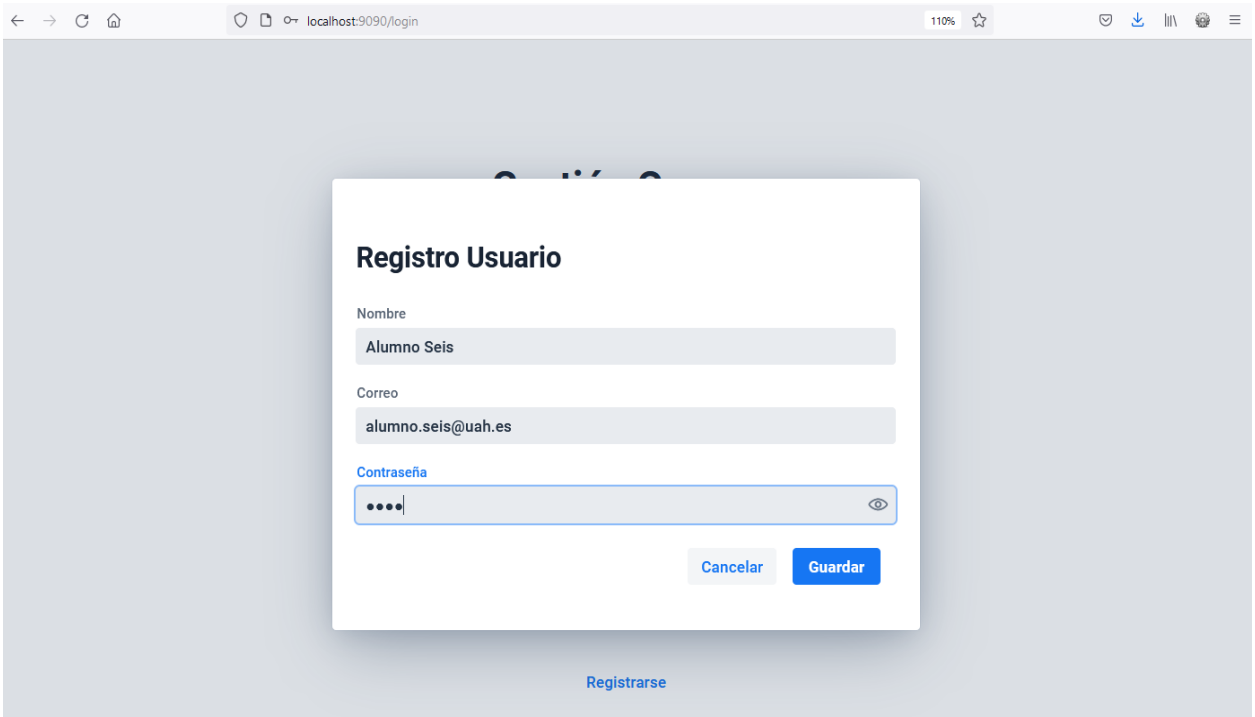


Ilustración 54. Registro de nuevo usuario

Cuando se ha registrado correctamente el usuario ya puede iniciar sesión

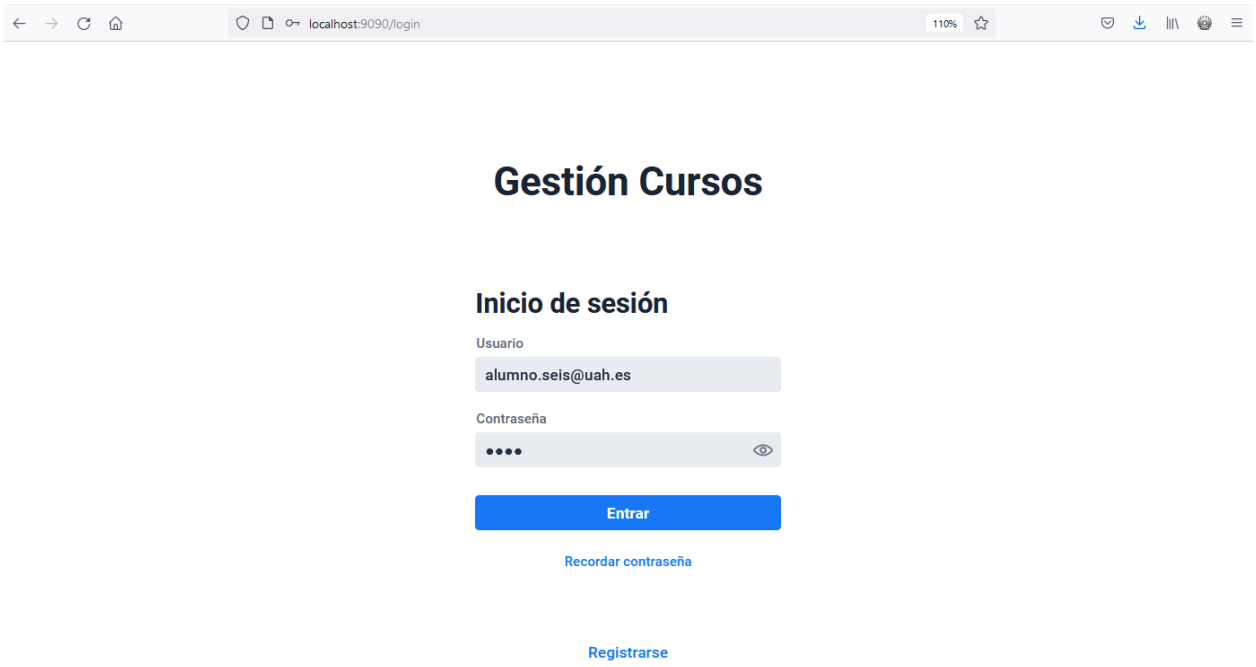


Ilustración 55. Login de usuario con rol alumno

El usuario con rol alumno solo puede visualizar la vista cursos y la única acción que tiene disponible es la de matricular y eliminar matricula.

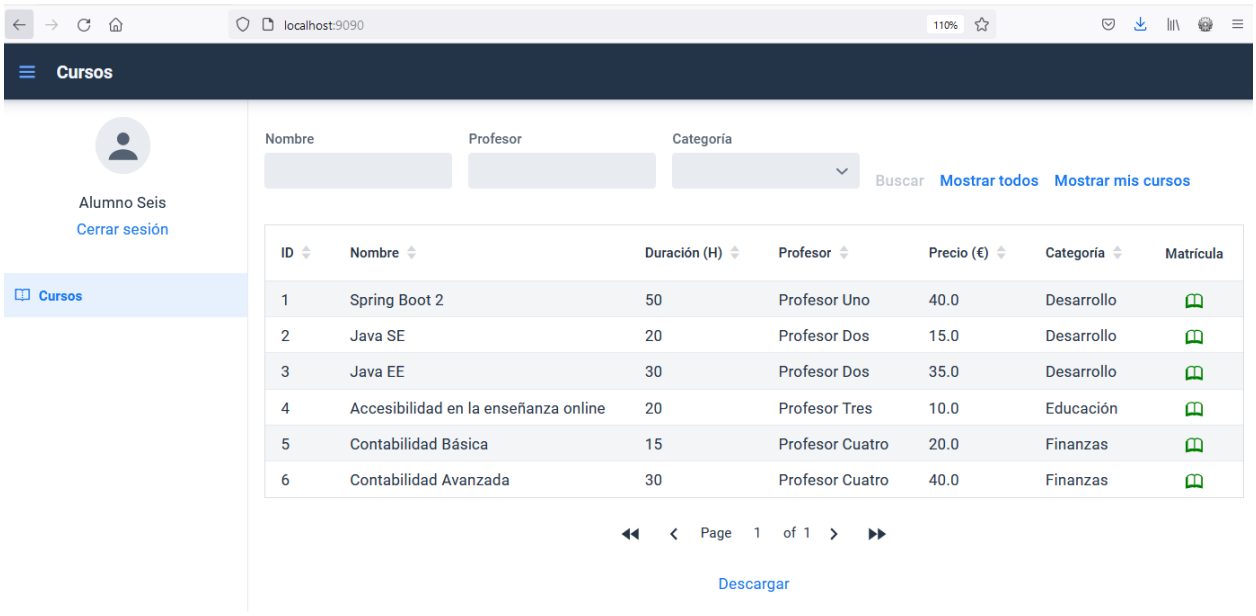


Ilustración 56. Vista de cursos con rol alumno

Si el alumno no está matriculado en el curso el icono de la columna Matricula se visualiza en color verde.

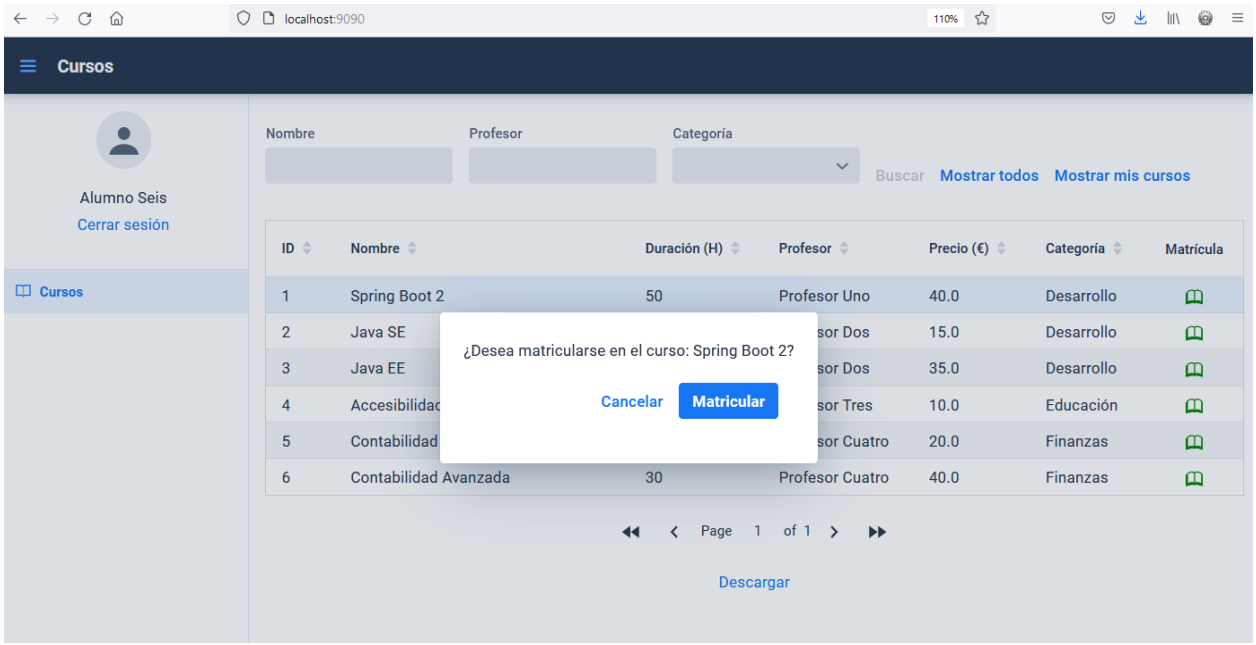


Ilustración 57. Confirmación de matriculación en curso

En el caso de que el alumno esté matriculado en el curso entonces el icono aparece en color rojo y presionándolo se puede eliminar la matricula del alumno en dicho curso.

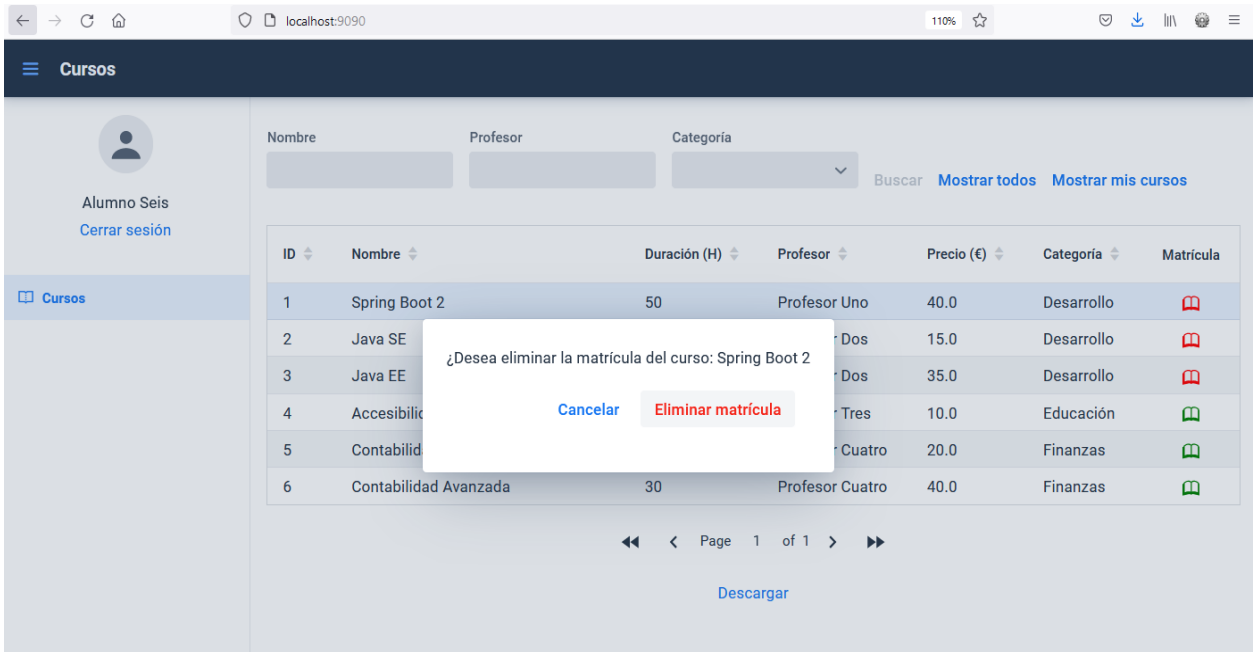


Ilustración 58. Confirmación eliminación matrícula

8. RESUMEN Y CONCLUSIÓN

8.1. Resumen.

Como se ha podido ver a lo largo de este trabajo con Vaadin se pueden crear aplicaciones empresariales que siguen los estándares de diseño y usabilidad de una manera muy ágil, permitiendo al desarrollador disponer de un amplio número de componentes visuales que se pueden configurar y utilizar fácilmente en la aplicación.

Además, al estar integrado con Spring Boot podemos utilizar los módulos de Spring para configurar la parte de seguridad de la aplicación como es el caso de Spring Security, RestTemplate para consumir servicios y también módulos de acceso a datos como JPA.

Palabras clave:

- Vaadin.
- Aplicación web.
- Framework.
- Spring Boot.
- Roles usuario.
- Backend
- Frontend

Abstract.

As has been seen throughout this work with Vaadin, business applications that follow design and usability standards can be created in a very agile way, allowing the developer to have a large number of visual components that can be easily configured and used in the app.

In addition, being integrated with Spring Boot we can use Spring modules to configure the security part of the application such as Spring Security, RestTemplate to consume service and also data access modules such as JPA.

Keywords:

- Vaadin.
- Web application.
- Framework.
- Spring Boot.
- User roles.
- Backend.
- Frontend.

8.2. Conclusiones y Futuras Líneas de Trabajo

La principal conclusión obtenida con la realización de este trabajo ha sido a que gracias al gran auge y avance de las distintas tecnologías y Frameworks para el desarrollo de aplicaciones web, hoy en día se pueden desarrollar aplicaciones completas conociendo solo un lenguaje de programación como Java, permitiendo también que una persona con perfil de Backend pueda desarrollar el Frontend. Se puede dedicar mayor esfuerzo en implementar la lógica de negocio, pero sin dejar de implementar la parte visual de la aplicación de una manera óptima y que se adapte a los estándares de diseño actuales.

Como futura línea de trabajo a nivel funcional se podría incluir el módulo de calificaciones, para que el profesor pueda asignar una calificación para cada alumno de la asignatura que imparte y que del mismo modo el alumno pueda visualizar dicha calificación y la de todos los cursos en la que está matriculado.

A nivel técnico se podría realizar la implementación de pruebas automáticas de la interfaz gráfica con la herramienta TestBench de Vaadin. Dicha herramienta permite crear y ejecutar test basados en la navegación de la aplicación simulando el uso de un usuario real.

9. BIBLIOGRAFÍA

[1] Documentation of vaadin flow version 14 [online]. https://vaadin.com/docs/v14/flow/overview

[2] Documentation of Spring Boot [online]. https://spring.io/projects/spring-boot

[3] Vaadin Components [online]. https://vaadin.com/components

[4] Tuturial of Spring Security in Vaadin [online]. https://vaadin.com/learn/tutorials/securing-your-app-with-spring-security

[5] Spring Security reference [online]. https://docs.spring.io/spring-security/site/docs/current/reference/html5/

[6] Grid Pagination Vaadin [online]. https://vaadin.com/directory/component/grid-pagination

[7] Stackoverflow Vaadin [online]. https://stackoverflow.com/questions/tagged/vaadin