



DESARROLLO Y ARQUITECTURA DE APLICACIÓN WEB

Raúl García Pablos
Ruth Hernández Quitián
Iván Zheng Fu
Mario Pérez Alfaraz

PROGRAMACIÓN. DAM 1ºC

Contenido

Modelado UML.....	2
Diagrama casos de uso.....	2
Diagrama de clases.....	4
Arquitectura de la aplicación web.....	5
Modelo	5
DAO. FactoryDAO	6
FactoryDAO	7
SQL FactoryDAO	8
Interfaces DAO	10
Repository	11
Service	12
Servlet	12
DBConnection.....	14
Maquetado del Proyecto.....	15
Junit	22

Modelado UML

Diagrama casos de uso

En el siguiente diagrama de casos de uso, se encuentran representadas las funcionalidades a las que pretende dar respuesta el desarrollo de nuestra aplicación web.

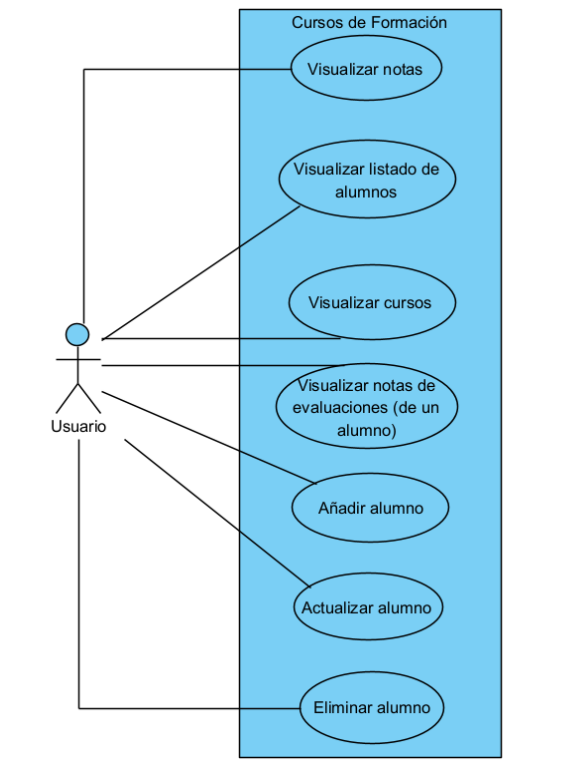


Imagen 1. Diagrama de casos de uso

Las funcionalidades consisten en las siguientes partes:

A partir de un usuario se podrán llevar a cabo distintas funciones como visualizar las notas del alumno, se podrá acceder en caso de ser alumno o tutor. También podemos encontrar la función de visualizar el listado de los alumnos pertenecientes al centro.

Otra función que podremos llevar a cabo será visualizar los cursos los cuales formarán parte del desarrollo de la distribución de los alumnos. Se podrán observar las notas por trimestre de un alumno específico, las cuales solo podrán tener uso el propio alumno o el tutor legal del mismo.

Posteriormente estas funcionalidades conllevan un gran papel como puede ser, tanto añadir alumnos al centro, como puede ser actualizar dicha cuenta a lo largo del curso o finalmente se podría llevar a cabo la eliminación de ese alumno.

Diagrama de clases

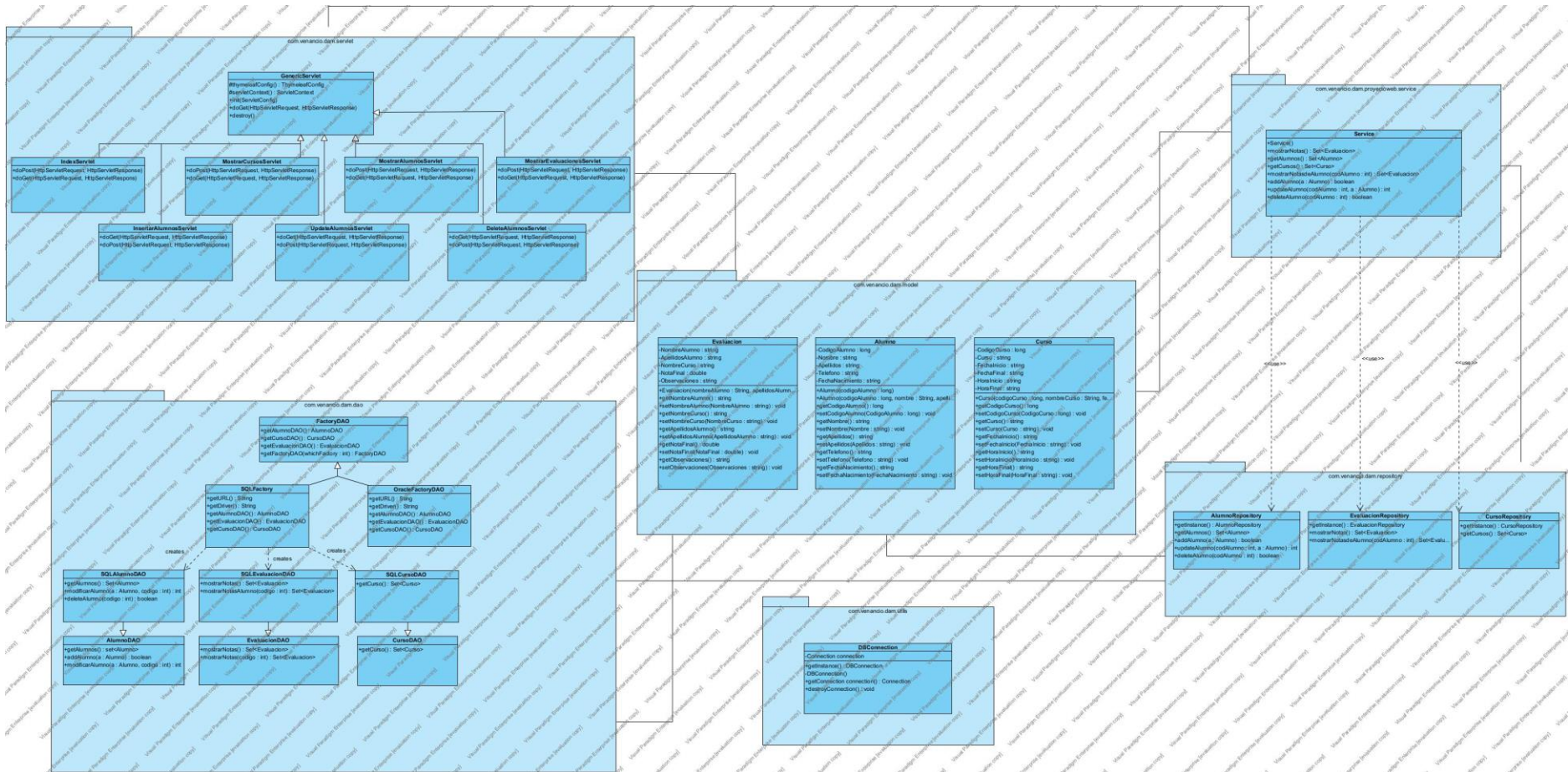


Imagen 2. Diagrama de clases

Arquitectura de la aplicación web

Modelo

Las clases que componen el modelo serán una representación de los datos que maneja el sistema. En nuestra aplicación, las clases que utilizaremos serán Alumno, Evaluación y Curso, análogas a las tablas que componen la base de datos del sistema, los atributos de las clases no son una representación directa de las tuplas de las tablas de la base de datos, sino que responden a las necesidades funcionales de la aplicación.

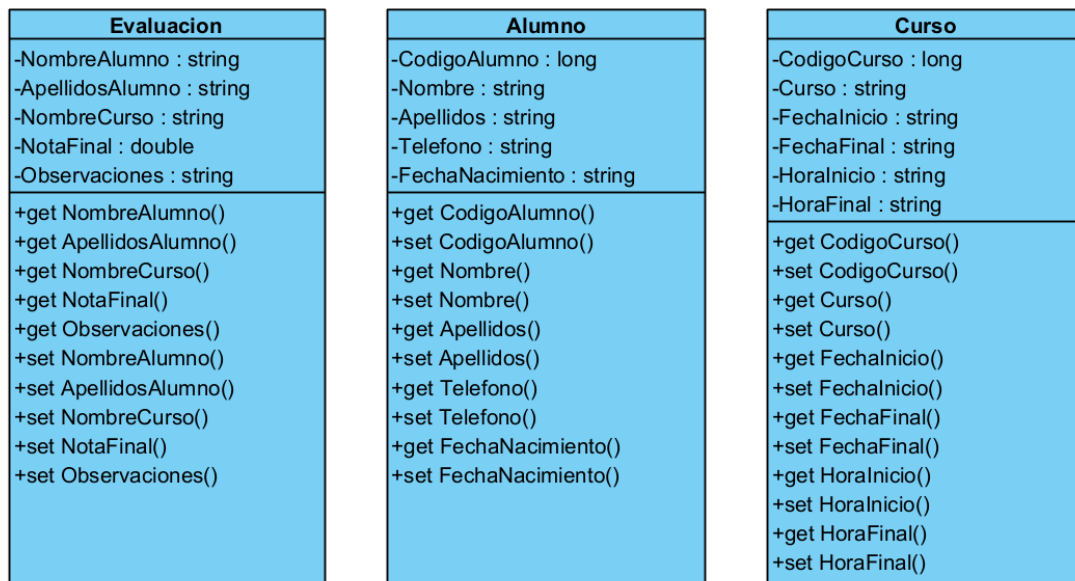


Imagen 3. Diagrama de clases del modelo

A nivel de programación, generaremos las clases representadas en el diagrama en el paquete modelo.

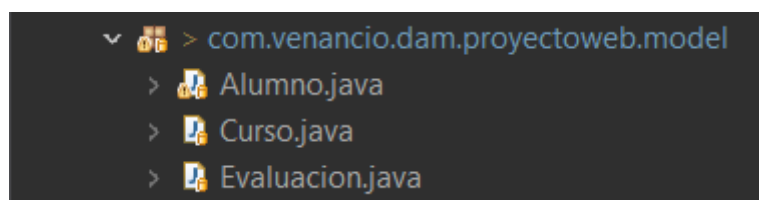


Imagen 4. Clases del paquete modelo

DAO. FactoryDAO

Los DAO o Data Access Object gestionan la conexión con la fuente de los datos para obtener y almacenar la información. Con el objetivo de desacoplar el código y facilitar una futura ampliación del programa, se ha implementado el esquema del FactoryDAO.

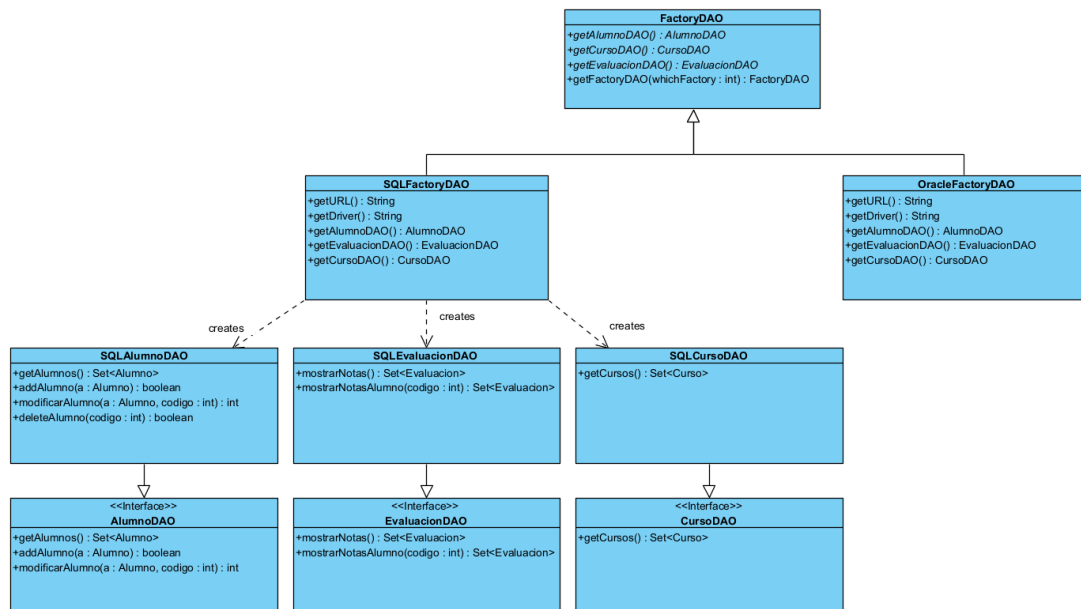


Imagen 5. Diagrama de clases del paquete dao

Este esquema está compuesto por una clase abstracta, **FactoryDAO**, que se encargará de facilitar el DAO apropiado al ser invocada.

Se generará además una clase por cada sistema gestor de base de datos o fuente de datos que se emplee en el programa; estas clases heredarán de la clase abstracta **FactoryDAO**, heredando la obligación de generar métodos que devuelvan los DAO apropiados, esta vez, los específicos de su sistema gestor. En nuestro programa tendremos dos clases, **SQLFactoryDAO** y **OracleFactoryDAO**, si bien solo la primera está operativa y es funcional para el programa.

Esta clase además será la encargada de generar la conexión con la base de datos correspondiente, si bien en el caso de nuestro programa, la clase **SQLFactoryDAO** facilita los datos necesarios a la clase **DBConnection** para que sea esta última la que genere la conexión.

Cada una de las clases del modelo tendrá su clase de acceso a datos o DAO para cada uno de los sistemas gestores que se manejen en el programa, por ello para facilitar la generación de estos DAO, se implementará una interfaz para cada una de las clases del modelo con los métodos que lanzarán consultas a la BBDD.

A nivel de programación, generaremos las siguientes clases en el paquete DAO:

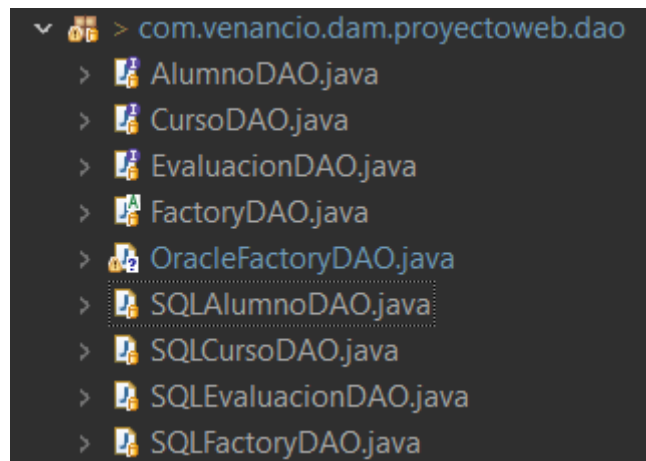


Imagen 6. Clases del paquete dao

FactoryDAO

```
package com.venancio.dam.proyectoweb.dao;

public abstract class FactoryDAO {

    public static final int SQL = 1;
    public static final int ORACLE = 2;

    public abstract AlumnoDAO getAlumnoDAO();

    public abstract CursoDAO getCursoDAO();

    public abstract EvaluacionDAO getEvaluacionDAO();

    public static FactoryDAO getFactoryDAO(int whichFactory) {
        switch (whichFactory) {
            case 1:
                return new SQLFactoryDAO();
            case 2:
                return new OracleFactoryDAO();
            default:
                return null;
        }
    }
}
```

Imagen 7. Código clase FactoryDAO

SQL FactoryDAO

Obtendrá los datos necesarios para realizar la conexión con la base de datos de un fichero de conexión y contará con métodos que devuelvan dicha información al ser invocados (en la clase DBConnection), en adición a los métodos que devuelven los DAO requeridos.

```

public class SQLFactoryDAO extends FactoryDAO {

    private final String RUTA = "Ficheros/";
    private final String DELIMITER = "@";

    private String DRIVER;
    private String URL;

    private String IP;
    private String PORT;
    private String BDNAME;
    private String CADENA_CONEXION = URL + IP + PORT + BDNAME;

    public void fillConfig() {
        Path file = Paths.get(RUTA, "conexion.txt");
        try (BufferedReader br = Files.newBufferedReader(file, StandardCharsets.UTF_8)) {
            String line = null;
            while (br.readLine() != null) {
                line = br.readLine();
                String[] aux = line.split(DELIMITER);
                switch (aux[0]) {
                    case "URL":
                        this.URL = aux[1];
                        break;
                    case "IP":
                        this.IP = aux[1];
                        break;
                    case "PORT":
                        this.PORT = aux[1];
                        break;
                    case "BDNAME":
                        this.BDNAME = aux[1];
                        break;
                    case "DRIVER":
                        this.DRIVER = aux[1];
                        break;
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public String getCadenaConexion() {
        return CADENA_CONEXION;
    }

    public String getDriver() {
        return DRIVER;
    }

    public SQLAlumnoDAO getAlumnoDAO() {
        return new SQLAlumnoDAO();
    }

    public SQLCursoDAO getCursoDAO() {
        return new SQLCursoDAO();
    }

    public SQLEvaluacionDAO getEvaluacionDAO() {
        return new SQLEvaluacionDAO();
    }
}

```

Imagen 8. Código clase SQLFactoryDAO

Interfaces DAO

```
package com.venancio.dam.proyectoweb.dao;

import java.util.Set;

public interface AlumnoDAO {

    public Set<Alumno> getAlumnos();

    public boolean addAlumno(Alumno a);

    public int modificarAlumno(Alumno a, int codigo);

    public boolean deleteAlumno(int cod);

}
```

Imagen 9. Código interfaz AlumnoDAO

```
package com.venancio.dam.proyectoweb.dao;

import java.util.Set;

public interface CursoDAO {

    public Set<Curso> getCursos();

}
```

Imagen 10. Código interfaz CursoDAO

```
package com.venancio.dam.proyectoweb.dao;

import java.util.Set;

public interface EvaluacionDAO {

    public Set<Evaluacion> mostrarNotas();

    public Set<Evaluacion> mostrarNotasAlumno(int codigo);

}
```

Imagen 11. Código interfaz EvaluacionesDAO

Repository

El repository es la capa intermedia entre el objeto de acceso a datos o DAO y la capa de negocio. Desde la capa repository llamaremos al FactoryDAO indicando con qué sistema gestor de bases de datos queremos realizar la conexión de manera que este nos facilite el DAO apropiado.

Así, en nuestro programa tendremos tres clases de repositorio correspondientes a las tres clases del modelo, Alumno, Evaluación y Curso.

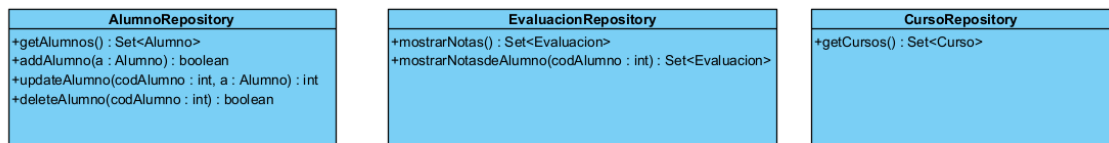


Imagen 12. Diagrama de clases repository

A nivel de programación, se generarán las clases ya explicitadas en el diagrama en el paquete repository:

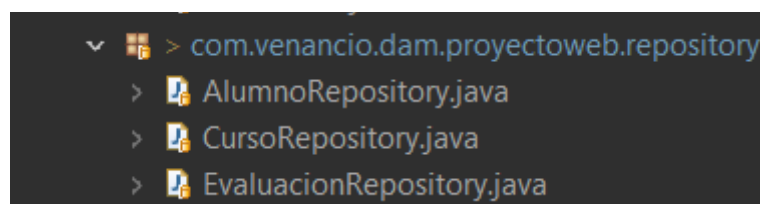


Imagen 13. Clases del paquete repository

Los repository tendrán un singleton para generar un único flujo de conexión.

```
private static CursoRepository instance;

public static synchronized CursoRepository getInstance() {
    if (instance == null) {
        instance = new CursoRepository();
    }
    return instance;
}
```

Imagen 14. Singleton de una clase del repositorio

Las llamadas a los DAO, se realizarán a través del FactoryDAO, que proporcionará el DAO correcto.

```

private CursoDAO dao;

public CursoRepository() {
    dao = FactoryDAO.getFactoryDAO(FactoryDAO.SQL).getCursoDAO();
}

```

Imagen 15. Código llamada al FactoryDAO desde el repositorio

Service

El service o capa de negocio, reunirá toda la lógica de programación. Se encargará de conectar con el servlet y de llamar a las capas intermedias en caso de ser necesario lanzar una consulta a la base de datos.

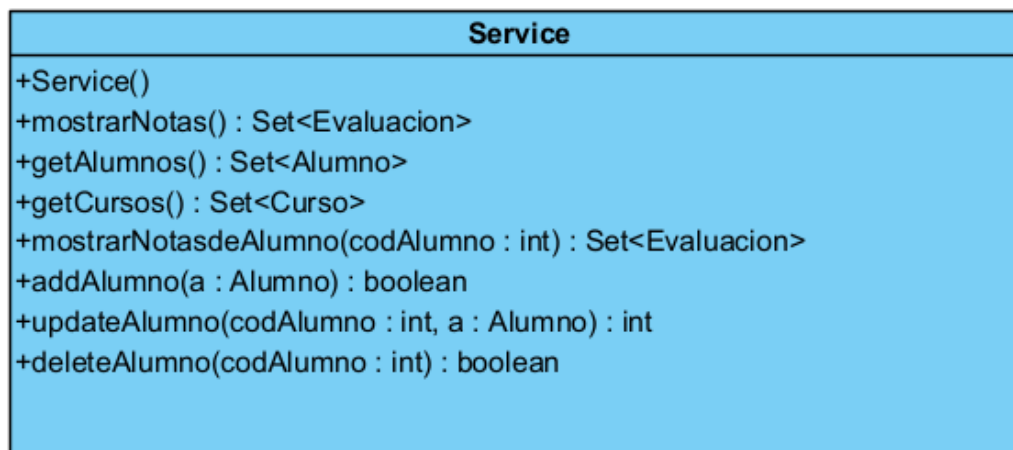


Imagen 16. Diagrama clase service

El service de nuestra aplicación recoge los métodos necesarios para llevar a cabo las siete funcionalidades de nuestro programa representadas en el diagrama de casos de uso.

Servlet

El servlet conectará el service con los documentos HTML y CSS que componen el front-end de la aplicación. Contaremos con un servlet genérico del que heredarán el resto de servlets

específicos que creemos para facilitar la generación del código. En nuestro programa, crearemos un servlet para cada una de las siete funcionalidades que ofrece.

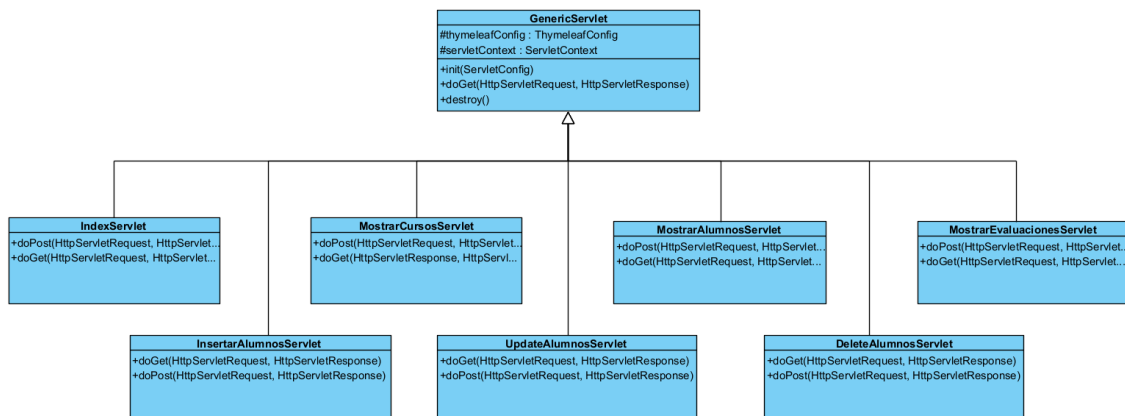


Imagen 17. Diagrama de clases del paquete servlet

A nivel de programación, se generarán las clases previamente mencionadas en el paquete correspondiente.

```

package com.venancio.dam.proyectoweb;

import java.io.IOException;

public class MostrarAlumnosServlet extends GenericServlet{

    private static final long serialVersionUID = 1L;

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        this.doGet(request, response);
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        super.doGet(request, response);

        WebContext ctx = new WebContext(request, response, servletContext, request.getLocale());

        Service service = new Service();
        Set<Alumno> allAlumnos = service.getAlumnos();

        ctx.setVariable("alumnos", allAlumnos);

        TemplateEngine engine = configThymeleaf.getTemplateEngine();
        engine.process("alumnos", ctx, response.getWriter());
    }
}
  
```

Imagen 18. Código de una clase del servlet

DBConnection

Clase de utilidad que genera la conexión con la base de datos.

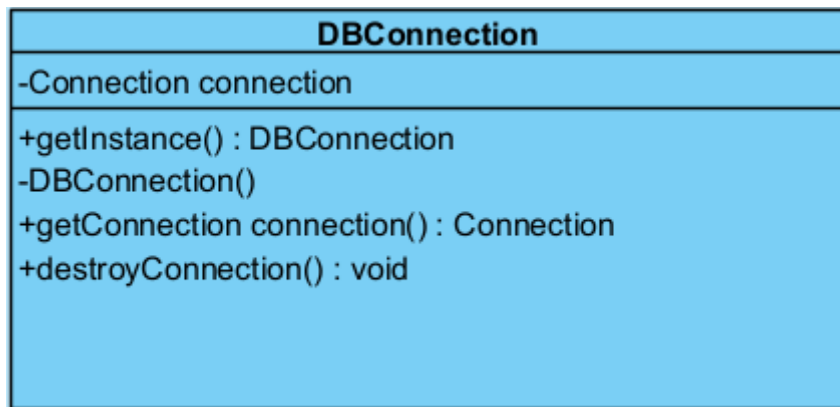


Imagen 19. Diagrama de clase DBConnection

Maquetado del Proyecto

Para poder hacer el diseño de nuestra página web hemos utilizado 2 herramientas principales: Figma y el modelado a papel

La gran ventaja que tenemos al hacer primero un modelado es que tenemos una visión exacta de lo que queremos y lo tenemos de referencia. Para así durante el proceso de maquetado en html, css no tengamos que dudar de nuestro diseño mental y tener una coherencia estética.

Figma: una de las ventajas es su fácil curva de aprendizaje y lo sencillo que es crear patrones de diseño como degradados para los fondos. Pero como desventaja, no puedes utilizarlo sin una conexión a internet.

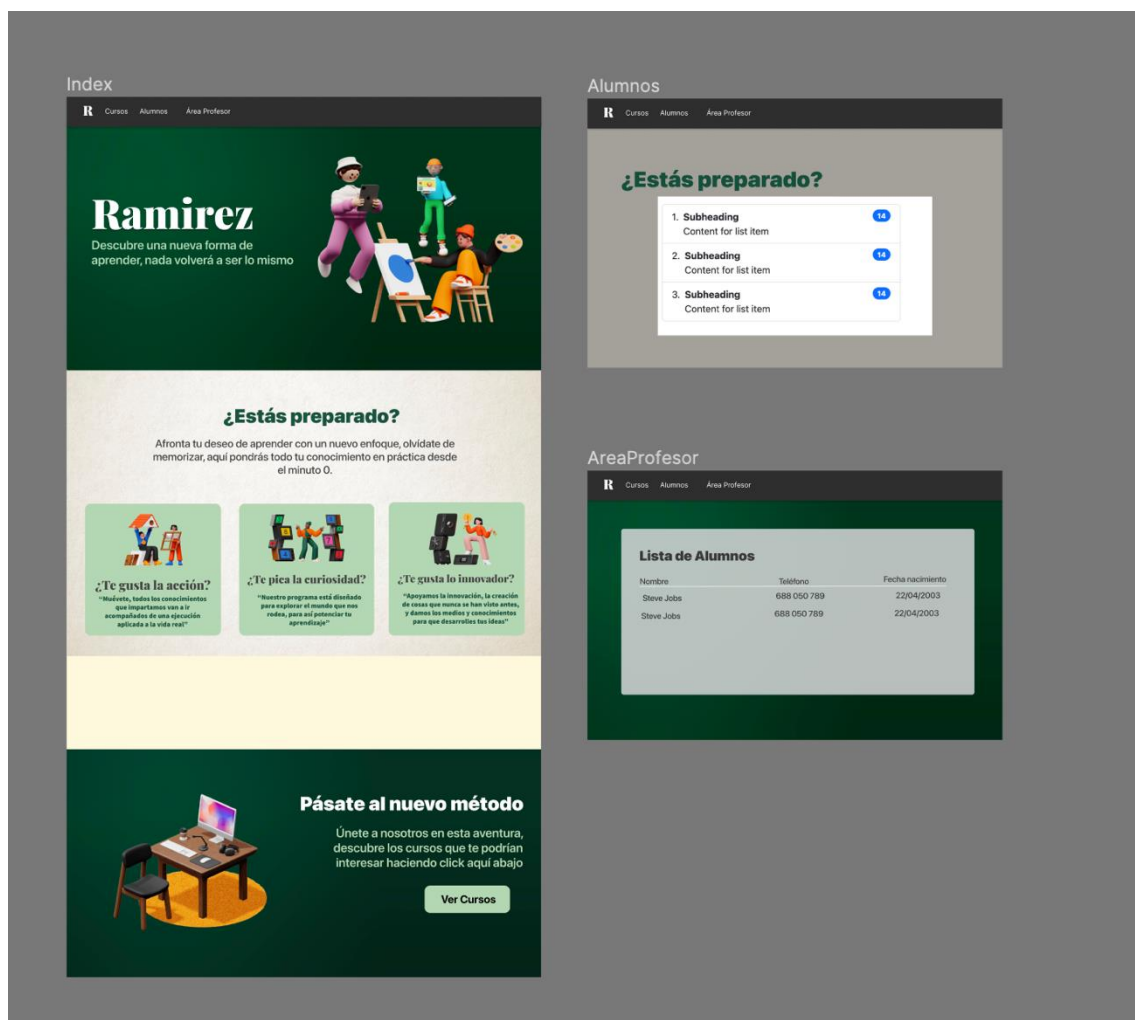


Imagen 20. Diseño figma

Y durante el período de tiempo que no tuvimos internet optamos por la vieja usanza, hacer el modelado a papel

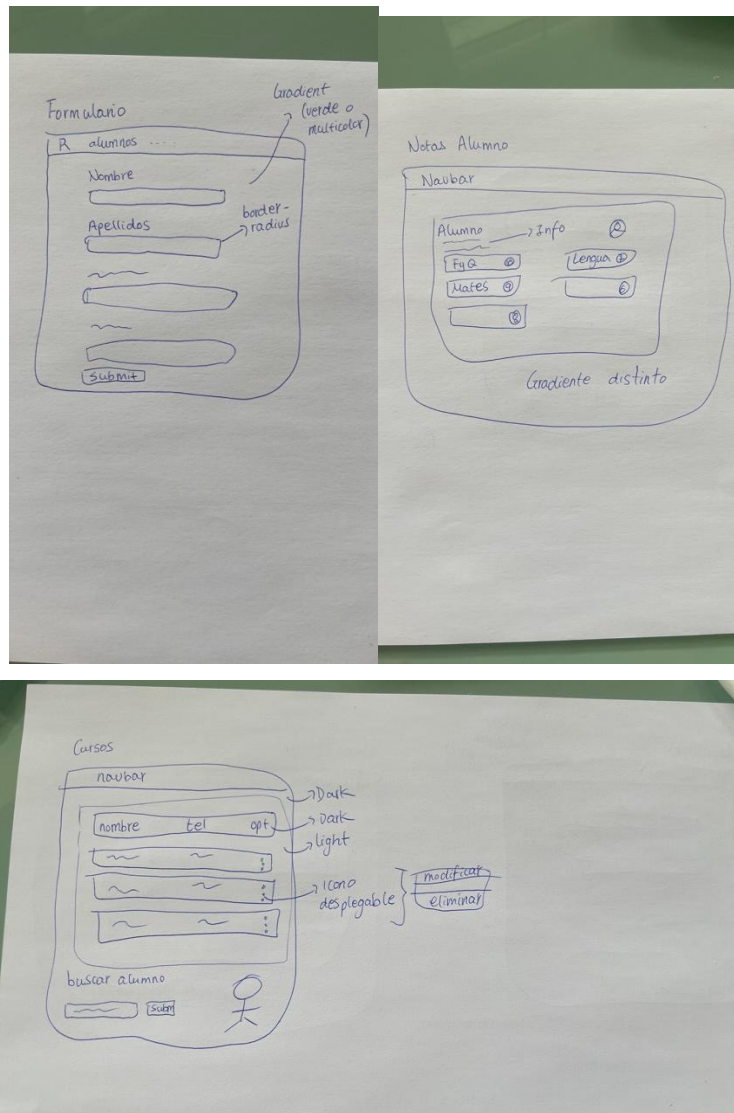


Imagen 21. Diseño a papel

Páginas que nos sirvieron de ayuda para realizar el maquetado:

[Storytale](#) para los diseños 3D que utilizamos, disponen de otros muchos personajes y líneas de diseño, como el scribble, cartoon y demás.

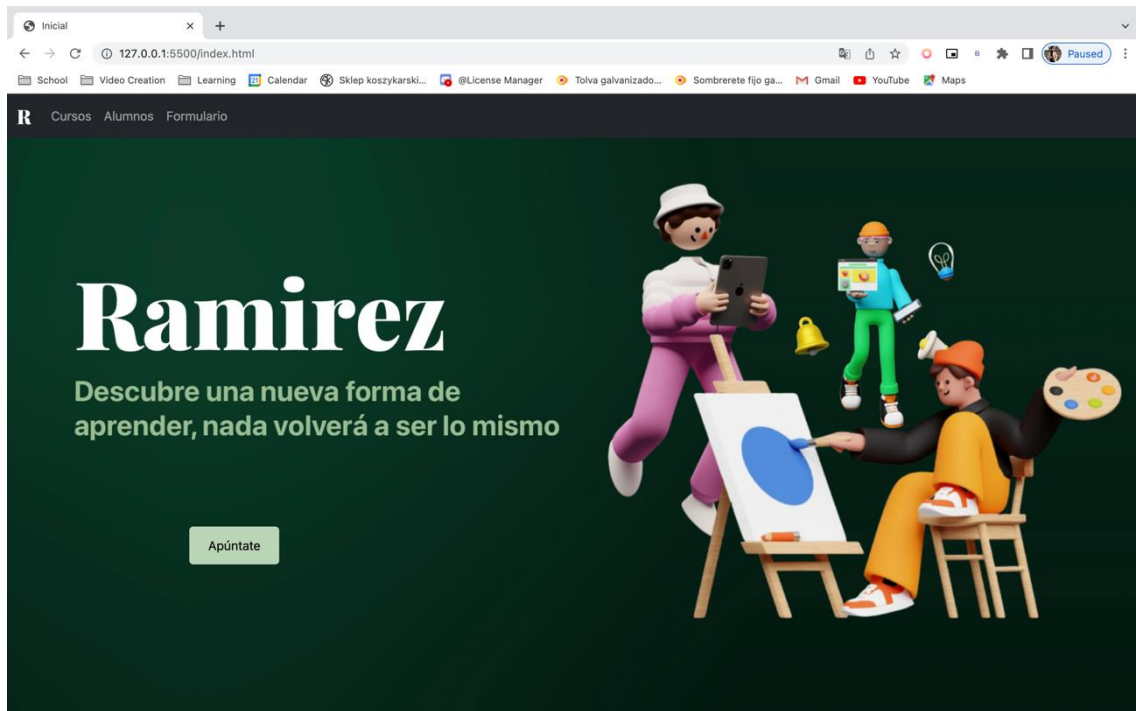
[icons8](#) es parecida a la de Storytale, pero no nos terminó de convencer lo que nos ofrecían

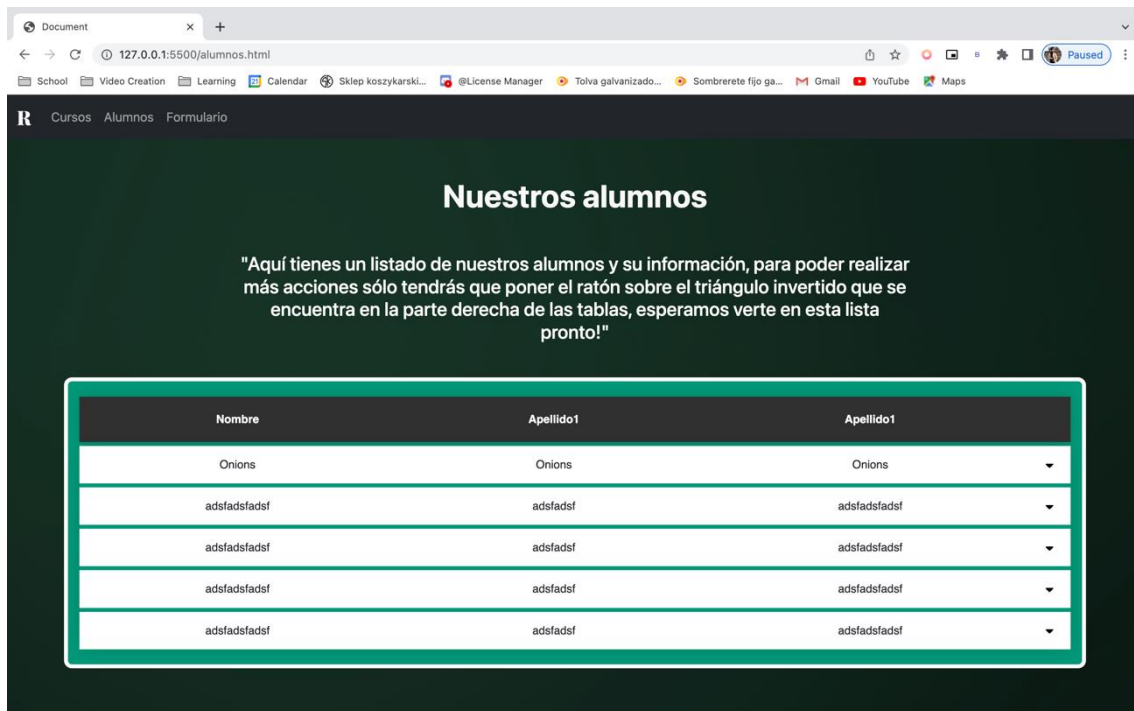
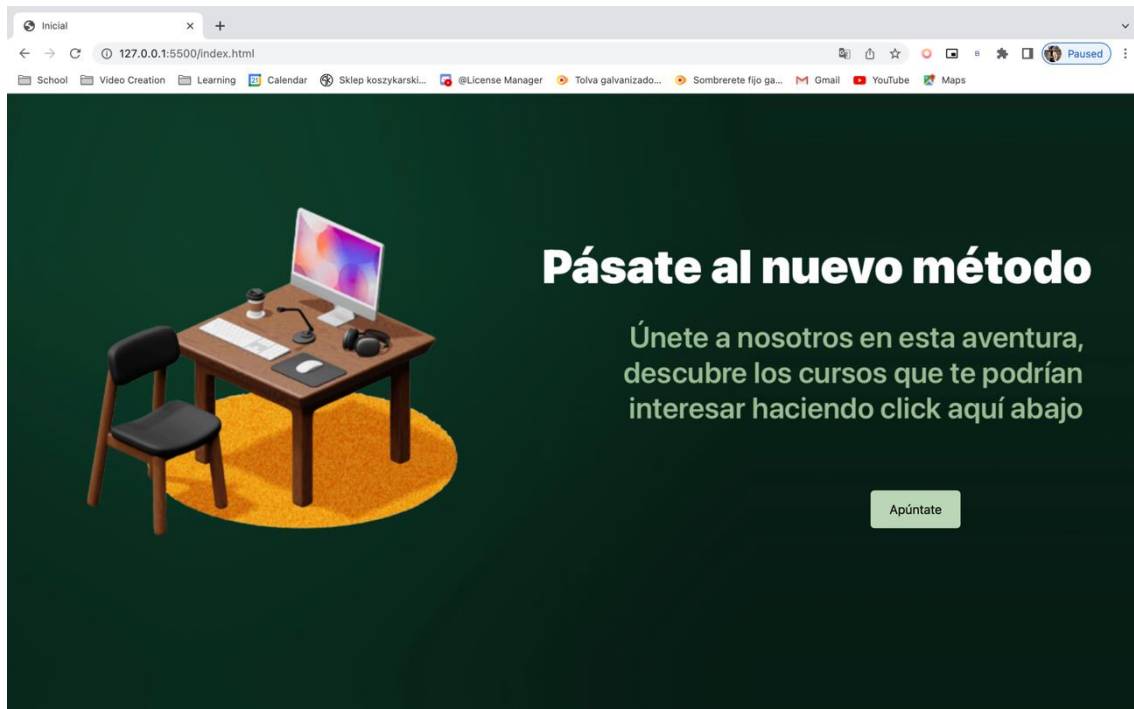
[Neumorphism](#) para crear sombreados en objetos.

[cssgradient.io](#) para previsualizar los gradientes que vas creando.

[Bootstrap](#) para previsualizar los diseños que te muestra Bootstrap, hablaremos de ello más adelante.

Una vez teniendo el maquetado hemos empezado a picar el código, buscamos maneras de convertir el proyecto Figma en código pero no hubo manera sencilla ni efectiva de hacerlo, así que tuvimos que picar código a la vieja usanza. Y este fue el resultado final





Document

127.0.0.1:5500/alumnos.html


School Video Creation Learning Calendar Sklep koszykarski... @License Manager Tolva galvanizado... Sombrero fijo ga... Gmail YouTube Maps

Onions	Onions	Onions
adfsadfsadfs	adfsadfs	adfsadfsadfs
adfsadfsadfs	adfsadfs	adfsadfsadfs
adfsadfsadfs	adfsadfs	adfsadfsadfs
adfsadfsadfs	adfsadfs	adfsadfsadfs

¿Estás Stalkeando a alguien en concreto?

"No te preocupes, sabemos que estás buscando a tu crush. Para encontrarlo tienes una función de búsqueda a tu derecha"

Encontrar!



Document

127.0.0.1:5500/alumnos.html

School Video Creation Learning Calendar Sklep koszykarski... @License Manager Tolva galvanizado... Sombrero fijo ga... Gmail YouTube Maps


Nuestros alumnos

"Aquí tienes un listado de nuestros alumnos y su información, para poder realizar más acciones sólo tendrás que poner el ratón sobre el triángulo invertido que se encuentra en la parte derecha de las tablas, esperamos verte en esta lista pronto!"

Nombre	Apellido1	Apellido1
Onions	Onions	Onions
adfsadfsadfs	adfsadfs	adfsadfsadfs
adfsadfsadfs	adfsadfs	adfsadfsadfs
adfsadfsadfs	adfsadfs	adfsadfsadfs
adfsadfsadfs	adfsadfs	adfsadfsadfs

Modificar
Eliminar
Mostrar notas

¿Estás Stalkeando a alguien en



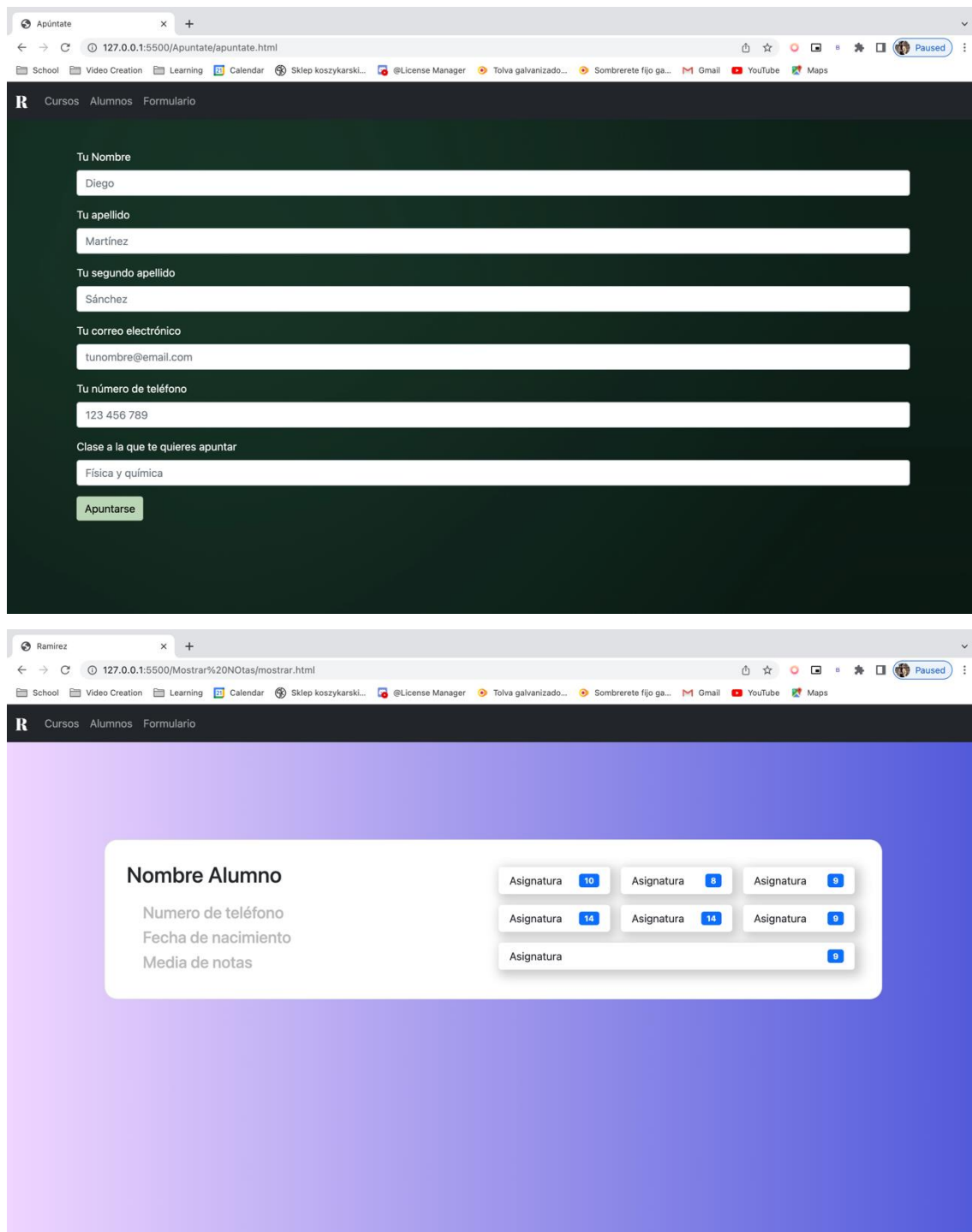


Imagen 22. Interfaz gráfica de nuestra aplicación

Para muchas partes del proyecto lo que hemos utilizado ha sido Bootstrap, en palabras simples es un autocompletado en esteroides, aunque hay varias funcionalidades como la del menú desplegable que no lo puedes conseguir sin JavaScript.

Algunos de los elementos hechos con Bootstrap han sido la navbar, la página de formulario y los badges de notas en la página de nombre alumno.

Y una vez ya tenemos importados los .css que necesitamos de Bootstrap, lo único que hay que hacer para poder utilizar es añadir estas dos líneas de código a tu html (el script va justo antes de acabar el body </body>)

```
<link rel="stylesheet" href="css/bootstrap.min.css"/>  
<script src="js/bootstrap.bundle.min.js"></script>
```

Imagen 23. Enlaces de documentos html a documentos css y bootstrap

Junit

Para el testeo del programa, se han empleado pruebas unitarias que comprueben la correcta ejecución de los métodos.

El objetivo era dar cobertura a la clase service.

Element	Coverage	covered Instructions	missed Instructions	Total Instructions
▼ projectoweb	41,6 %	865	1.212	2.077
▼ src/main/java	38,5 %	745	1.188	1.933
> com.venancio.dam.projectoweb	0,0 %	0	500	500
> com.venancio.dam.projectoweb	51,3 %	290	275	565
> com.venancio.dam.projectoweb	55,2 %	299	243	542
> com.venancio.dam.projectoweb	33,3 %	46	92	138
▼ com.venancio.dam.projectoweb	40,8 %	29	42	71
> Init.java	0,0 %	0	42	42
> Service.java	100,0 %	29	0	29
> com.venancio.dam.projectoweb	0,0 %	0	36	36
> com.venancio.dam.projectoweb	100,0 %	81	0	81
> src/test/java	83,3 %	120	24	144

Imagen 24. Coverage del proyecto