

Proyecto FPBasics.

Sistema de Gestión de Ciclos Duales

Integrantes:

- Javier García Jiménez
- Francisco Javier Rosa Martín
- Luciano Saavedra Conejo



Indice

Descripción General del proyecto.....	3
Objetivos Generales.....	3
Análisis previo de ideas.....	3
Diseño Inicial.....	3
Derivación del Proyecto.....	4
Diseño Final.....	4
Justificación de las opciones.....	4
Desarrollo del Proyecto.....	5
Sobre las Bases de Datos.....	5
Diseño.....	5
Diagramas.....	7
Estructura Servicio-Repositorio-Modelo.....	9
Maven.....	9
Spring Framework.....	11
Anotaciones.....	12
MyBatis.....	13
Los Mappers.....	13
Los XML.....	14
Modelo Vista-Controlador(MVC).....	15
JavaServer Faces (JSF).....	17
PrimeFaces.....	18
La interfaz.....	20
Visión General.....	20
Dificultades encontradas.....	26
MyBatis y Accesos.....	26
Sistemas Gestores de Bases de Datos.....	26
Sobre la integración entre Spring y JSF.....	27
Inyección Automática e Inyección Manual.....	27
Los archivos de configuración.....	28
Problemas con Servidor TomCat.....	29
Conclusiones.....	29
Aplicaciones Reales del Proyecto.....	29
Futuras Mejoras.....	29
Ventajas y Desventajas de Java.....	30
Fuentes de Información y agradecimientos.....	31

Descripción General del proyecto.

Nuestro proyecto consiste en una aplicación multiplataforma destinada a la gestión de todo lo referente a la formación de los ciclos de formación dual, especialmente orientado al ciclo de Desarrollo de aplicaciones web del Instituto I.E.S Luis Vélez de Guevara.

En líneas generales funciona como un sistema compuesto por una interfaz que interactúa con un sistema de servicios y repositorios, los cuales interactúan con una base de datos a través de mappers, guardando los contenidos obtenidos en diversos vectores (Listas), los cuales son mostrados en la interfaz gráfica, permitiendo la consulta y la modificación de estos.

En este proyecto se utilizan diversos frameworks y tecnologías, que concretamente son los que se piden actualmente en el mercado laboral y están a la cabeza tecnológica en el campo del desarrollo con la tecnología Java.

Objetivos Generales.

Los objetivos generales de nuestro proyecto son diseñar una aplicación Web, destinada a la gestión de ciclos formativos de formación dual, concretamente a la gestión de la información del periodo de prácticas en empresa donde podremos gestionar formadores, bloques formativos, módulos, resultados de aprendizaje y actividades formativas.

Nuestro objetivo es crear esta aplicación utilizando el lenguaje de programación Java EE, así como sus frameworks más populares y las técnicas de programación más vanguardistas en torno a este lenguaje. Asimismo tenemos el objetivo de profundizar en estas tecnologías, puesto que buscamos un proceso de aprendizaje profundo, actualizado y al nivel de las exigencias laborales de nuestro sector.

Análisis previo de ideas.

Diseño Inicial.

Nuestro primer pensamiento sobre el diseño de nuestra aplicación, consistía en un proyecto java simple que mediante un limitado número de clases y haciendo uso directo del driver JDBC, realizar una conexión a la base de datos y hacer consultas y modificaciones sobre ella a través de una interfaz de escritorio.

Derivación del Proyecto.

A medida que avanzamos en nuestra formación, hemos ido descubriendo los beneficios que aportan varias técnicas de programación y frameworks. El primero de estos fue Maven (vease Desarrollo del Proyecto>Maven) con el que pudimos trabajar con mucha más rapidez y facilidad y tras este, Spring (vease Desarrollo del Proyecto>Spring Framework), con el que comprendimos las ventajas de trabajar sobre interfaces y nos llevó a trabajar con MyBatis (vease Desarrollo del Proyecto>MyBatis) y con la estructura Repositorio-Servicio. (vease Desarrollo del Proyecto>Estructura Servicio-Repositorio).

También, la interfaz de nuestro proyecto sufrió una derivación, por la cual decidimos utilizar una interfaz web en lugar de una interfaz de escritorio. Esto se produjo, principalmente por dos razones, la primera fue por el consejo de nuestros tutores de prácticas, pues estos nos comentaron que una interfaz web provoca muchos menos problemas y es considerablemente más simple que una de escritorio a la hora del desarrollo, la segunda razón se basa en el entorno de desarrollo utilizado (Eclipse), puesto que este entorno a pesar de ser mucho más potente que su principal competidor (NetBeans), no proporciona herramientas de apoyo al desarrollo de interfaces gráficas, al contrario que NetBeans. Por lo que finalmente decidimos utilizar una interfaz web basada en JSF.

Diseño Final.

En resumen, el diseño de nuestra aplicación consiste en una interfaz gráfica desarrollada con JSF y usando Primefaces, que se conecta a una estructura Servicio-Repositorio, la cual conecta con MyBatis, que realiza la conexión a la base de datos a través de unos mappers en xml para realizar operaciones de consulta y modificación sobre la base de datos, y una vez recogida esta información la muestra a través de la interfaz con ayuda de unas clases auxiliares que almacenan información de cada registro.

Todo esto lo veremos con más detalle a medida que avancemos en este documento.

Justificación de las opciones.

Las opciones que hemos escogido son justificadas a continuación:

- **Maven:** Hemos utilizado este tipo de proyecto, porque a la hora de utilizar librerías externas, nos resulta mucho más sencillo trabajar con un repositorio de dependencias ligado a un xml, y porque además la organización visual de los archivos es mucho más clarificadora.
- **Spring:** Este framework proporciona mucha facilidad a la hora de inyectar diferentes frameworks, puesto que nos permite trabajar instanciando en base a la interfaz y con sus anotaciones permite auto-inyectar los servicios y mappers.

- **MyBatis:** Este framework, que trabaja de la mano con Spring, nos permite trabajar con la base de datos asignando ciertas funciones a los mappers que usaremos en los repositorios.
- **JSF:** Este framework es una de las mejores opciones que existen hoy en día para desarrollar interfaces web. Este es de fácil implementación y permite crear interfaces de usuario con un acabado bastante profesional.

Desarrollo del Proyecto.

Sobre las Bases de Datos.

Las Bases de Datos se consideran una de las mayores aportaciones que ha dado la informática.

Las principales utilidades que ofrece una base de datos a la empresa son las siguientes:

- Agrupar y almacenar todos los datos de la empresa en un único lugar.
- Facilitar que se compartan los datos entre los diferentes miembros de la empresa.
- Evitar la redundancia y mejorar la organización de la agenda.
- Realizar una interlocución adecuada con los clientes.

Si una Base de Datos se gestiona adecuadamente, obtendremos diferentes ventajas:

- Aumentará su eficacia, habrá trabajos que se realicen con mayor rapidez y agilidad.
- Podremos mejorar la seguridad de los datos que almacenamos.
- Ahorraremos espacio en el disco eliminando los datos redundantes.
- Mantendremos la precisión e integridad de los datos.

Por tanto, se producirá una mejora en la productividad y estas funcionalidades aportarán un valor añadido a la empresa.

Diseño.

Para realizar un diseño útil y eficiente debemos definir el propósito de nuestra base de datos, para ello encontramos 4 etapas que nuestro diseño debe recorrer.

1- Identificación el propósito de tu base de datos:

En este punto debemos pensar qué vamos a almacenar en la base de datos.

Nuestra necesidad de crear una base de datos surge de la obligación de almacenar los datos relacionados con la Formación Dual del I.E.S Luis Vélez.

2- Organización de los datos en tablas.

En nuestro caso, vamos a crear las siguientes tablas: Modulos, Bloques, Formadores, Resultados de Aprendizaje y Actividades Formativas, y sus respectivos campos.

El cuerpo de las tablas mencionadas anteriormente se mostrará en el siguiente apartado con más detalle.

3- Especificación de las claves primarias y análisis de las relaciones.

En el siguiente punto debemos establecer las relaciones entre las tablas, es decir, definir los atributos que funcionarán como clave primaria para cada tabla y los atributos que funcionarán como clave foránea para relacionar las tablas entre sí.

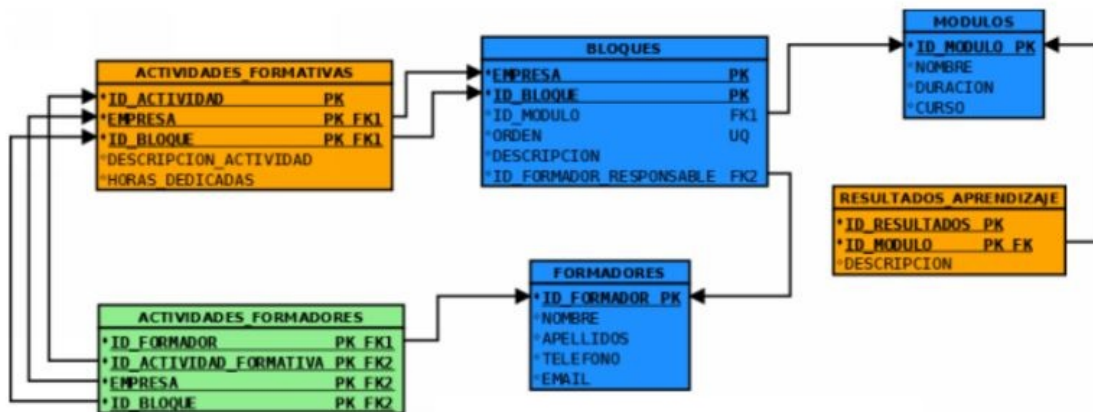
TABLA	CLAVE PK	CLAVE FK a CLAVE PK
<u>MODULOS</u>	<u>ID_MODULO</u>	
FORMADORES	ID_FORMADOR	
BLOQUES	ID_BLOQUE	fk1: <u>ID_MODULO - ID_MODULO</u> (tabla <u>MODULOS</u>) fk2: <u>ID_FORMADOR_RESPONSABLE - ID_FORMADOR</u> (tabla <u>FORMADORES</u>)
<u>RESULTADOS_APRENDIZAJE</u>	<u>ID_RESULTADOS</u>	fk: <u>ID_MODULO - ID_MODULO</u> (tabla <u>MODULOS</u>)
<u>ACTIVIDADES_FORMATIVAS</u>	ID_ACTIVIDAD	fk: <u>EMPRESA, ID_BLOQUE - EMPRESA, ID_BLOQUE</u> (tabla <u>BLOQUES</u>)

4- Normalización para estandarizar las tablas.

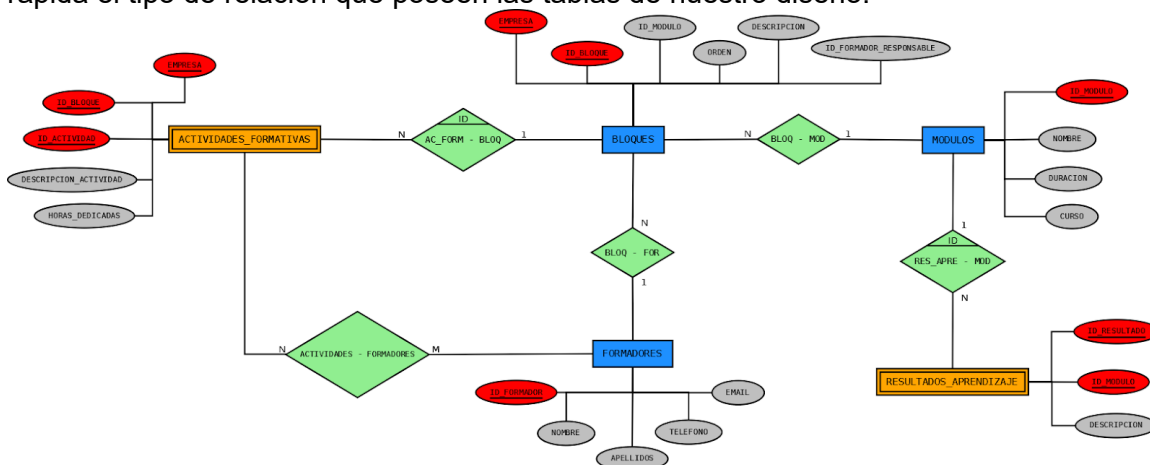
- La primera forma normal “1FN”: especifica que cada celda de la tabla solo puede tener un valor, nunca una lista de valores. *Requisito cumplido.*
- La segunda forma normal “2FN”: especifica que un atributo no puede depender de otro atributo, únicamente puede depender de la clave primaria. *Requisito cumplido.*
- La tercera forma normal “3FN”: especifica que cada una de las columnas que no sea clave primaria debe ser independiente de las demás columnas. *Requisito cumplido.*

Diagramas.

A continuación se muestra el diagrama Relacional de la base de datos que hemos diseñado para entender mejor cómo funciona.



En este caso, se muestra el diagrama Entidad-Relación para poder visualizar de una forma fácil y rápida el tipo de relación que poseen las tablas de nuestro diseño.



Para terminar con la fase de diseño de la base de datos creamos un usuario en nuestra base de datos y le dimos permisos totales con los siguientes comandos:

```
SQL> create user fpbasics identified by fpbasics
2 DEFAULT TABLESPACE users
3 TEMPORARY TABLESPACE temp
4 QUOTA 3M ON users;
```

```
User created.
```

```
SQL>
```

```
SQL> GRANT CREATE SESSION TO fpbasics;

Grant succeeded.
```

Una vez hecho esto, ejecutamos el script que preparamos con la creación de las tablas y sus respectivas claves (este script podemos encontrarlo en la carpeta raíz de nuestro proyecto).

```
1  CREATE TABLE FORMADORES
2  (
3      ID_FORMADOR    NUMERIC(3),
4      NOMBRE         VARCHAR(30),
5      APELLIDOS      VARCHAR(30),
6      TELEFONO       NUMERIC(9),
7      EMAIL          VARCHAR(30),
8      CONSTRAINT pk_formador PRIMARY KEY(ID_FORMADOR)
9  );
10
11 CREATE TABLE MODULOS
12 (
13     ID_MODULO    NUMERIC(3),
14     NOMBRE       VARCHAR(30),
15     DURACION     NUMERIC(3,1),
16     CURSO        VARCHAR(2),
17     CONSTRAINT pk_modulo PRIMARY KEY(ID_MODULO)
18 );
19
20 CREATE TABLE BLOQUES
21 (
22     EMPRESA        VARCHAR(20),
23     ID_BLOQUE      NUMERIC(3),
24     ID_MODULO      NUMERIC(3),
25     DESCRIPCION    VARCHAR(100),
26     ORDEN          VARCHAR(3),
27     ID_FORMADOR_RESPONSABLE NUMERIC(3),
28     CONSTRAINT pk_bloques PRIMARY KEY(EMPRESA, ID_BLOQUE),
29     CONSTRAINT fk1_bloques FOREIGN KEY(ID_MODULO) REFERENCES MODULOS(ID_MODULO),
30     CONSTRAINT fk2_formadores FOREIGN KEY(ID_FORMADOR_RESPONSABLE) REFERENCES FORMADORES(ID_FORMADOR),
31     CONSTRAINT uq_bloques UNIQUE (ORDEN)
32 );
33
34 CREATE TABLE RESULTADOS_APRENDIZAJE
35 (
36     ID_RESULTADOS  NUMERIC(3),
37     ID_MODULO      NUMERIC(3),
38     DESCRIPCION    VARCHAR(100),
39     CONSTRAINT pk_resultados PRIMARY KEY(ID_RESULTADOS, ID_MODULO),
40     CONSTRAINT fk_modulo FOREIGN KEY(ID_MODULO) REFERENCES MODULOS(ID_MODULO) ON DELETE CASCADE
41 );
42
43 CREATE TABLE ACTIVIDADES_FORMATIVAS
44 (
45     ID_ACTIVIDAD    NUMERIC(3),
46     EMPRESA        VARCHAR(20),
47     ID_BLOQUE      NUMERIC(3),
48     DESCRIPCION_ACTIVIDAD VARCHAR(100),
49     HORAS_Dedicadas NUMERIC(3,1),
50     CONSTRAINT pk_actividades PRIMARY KEY(ID_ACTIVIDAD, EMPRESA, ID_BLOQUE),
51     CONSTRAINT fk_actividades FOREIGN KEY(EMPRESA, ID_BLOQUE) REFERENCES BLOQUES(EMPRESA, ID_BLOQUE) ON DELETE CASCADE
52 );
53
54 CREATE TABLE ACTIVIDADES_FORMADORES
55 (
56     ID_FORMADOR    NUMERIC(3),
57     ID_ACTIVIDAD    NUMERIC(3),
58     EMPRESA        VARCHAR(20),
59     ID_BLOQUE      NUMERIC(3),
60     CONSTRAINT pk_actividades_formadores PRIMARY KEY(ID_FORMADOR, ID_ACTIVIDAD),
61     CONSTRAINT fk1_actividades_formadores FOREIGN KEY(ID_FORMADOR) REFERENCES FORMADORES(ID_FORMADOR) ON DELETE CASCADE,
62     CONSTRAINT fk2_actividades_formadores FOREIGN KEY(ID_ACTIVIDAD, EMPRESA, ID_BLOQUE)
63     REFERENCES ACTIVIDADES_FORMATIVAS(ID_ACTIVIDAD, EMPRESA, ID_BLOQUE) ON DELETE CASCADE
64 );
65
```


Estructura Servicio-Repositorio-Modelo.

La estructura Servicio-Repositorio-Modelo consiste en una metodología de trabajo que se encarga de implementar los métodos del back-end que va a necesitar nuestra aplicación en una clase servicio. Estos métodos sólo tendrán el cometido de llamar a otras funciones análogas en los repositorios, los cuales en su contenido tendrán todo el grueso del código necesario para hacer su función.

Esta metodología de trabajo nos aporta dos principales ventajas, la primera es que clarifica en gran medida las funciones que lleva a cabo nuestra aplicación, puesto que los servicios, al estar libre de todo el grueso del código, nos muestran con claridad cuáles son las funcionalidades que ofrece nuestro programa.

Y en segundo lugar, nos permite ampliar en gran medida la modularidad de nuestra aplicación, puesto que si queremos cambiar el funcionamiento de nuestro programa a nivel de backend, podremos cambiar los repositorios, sin que el servicio y la posible interfaz lo noten.

Pongamos un ejemplo, imaginemos que el creamos un programa de acceso y consulta a una base de datos, contamos con una interfaz conectada a un servicio, el cual a su vez está conectado a un repositorio que realiza la conexión y la consulta. Pero resulta que nos encontramos que nuestro jefe de operaciones ya no quiere que el programa interactúe con una base de datos, si no que en su lugar utilice un sistema de ficheros.

Utilizando la estructura Servicio-Repositorio, solo tendríamos que generar un nuevo repositorio, con los mismo métodos que el anterior, pero cambiando la consulta a la base de datos por una consulta a ficheros, y una vez creado solo habría que cambiar el puntero en el servicio, para que el programa funcionase de forma completamente distinta sin necesidad de refactorizar toda la estructura del programa y corriendo menos riesgos de provocar un fallo.

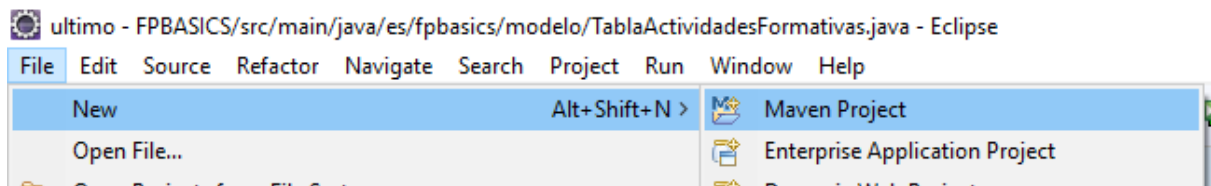
Maven.

Una de las herramientas más útiles a la hora de utilizar librerías de terceros es Maven. Maven hace posible la creación de software con dependencias incluidas dentro de la estructura del JAR. Es necesario definir todas las dependencias del proyecto (librerías externas usadas) en un fichero propio de todo proyecto Maven, el POM(Project Object Model), que es un archivo XML que contiene todo lo necesario para generar el fichero ejecutable de la aplicación.

La capacidad más importante de Maven es la capacidad de trabajar en red. Al usar librerías en el POM, el sistema se encargará de descargar y mantener actualizada dicha librería ya que los creadores mantienen totalmente actualizado un repositorio online, manteniendo la opción de volver a versiones anteriores.

Manual de creación de un proyecto Maven en Eclipse:

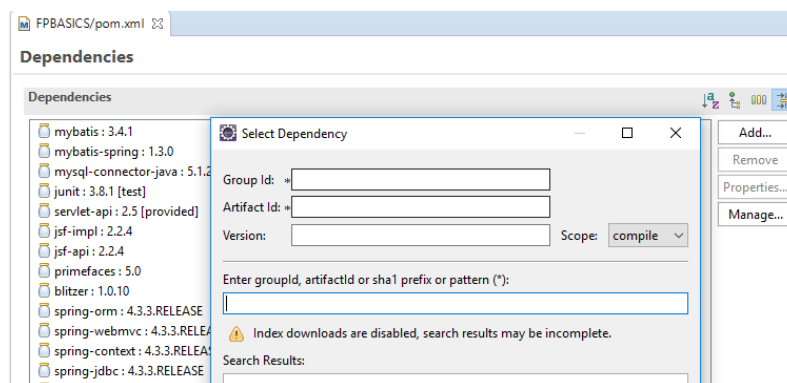
1. File -> New... -> Other -> Maven -> Maven Project



2. Para añadir dependencias, buscamos el archivo XML llamado POM.xml y en la pestaña "Source" tendremos un XML en el que añadiremos dentro de la raíz "Dependencies" las dependencias que deseemos integrar en nuestro proyecto.

```
21      <!-- https://mvnrepository.com/artifact/org.mybatis/mybatis -->
22      <dependency>
23          <groupId>org.mybatis</groupId>
24          <artifactId>mybatis</artifactId>
25          <version>3.4.1</version>
26      </dependency>
27      <!-- https://mvnrepository.com/artifact/org.mybatis/mybatis-spring -->
28      <dependency>
29          <groupId>org.mybatis</groupId>
30          <artifactId>mybatis-spring</artifactId>
31          <version>1.3.0</version>
32      </dependency>
33      <!-- db driver -->
34      <dependency>
35          <groupId>mysql</groupId>
36          <artifactId>mysql-connector-java</artifactId>
37          <version>5.1.21</version>
38      </dependency>
```

3. Otra forma de añadir dependencias, es en la pestaña "Dependencies" haciendo click en "Add..." y usando el buscador para localizar nuestra dependencia.



Spring Framework.

A la hora de programar en Java, llegamos a un punto en el que usamos diferentes Frameworks y cada uno de estos Frameworks generan sus propios conjuntos de objetos. Anteriormente la generación de estos objetos corría a cargo del desarrollador, pero, en la actualidad usamos Spring Framework.

Spring es el encargado, mediante anotaciones o ficheros xml, de generar los objetos necesarios para el funcionamiento de los demás frameworks usados en nuestro proyecto. Este framework es el encargado de inicializar todos los objetos de los distintos frameworks usados y también de asegurarnos que se integran de forma correcta unos con otros.

Un ejemplo de código de Spring framework es el siguiente, que se trata de un fragmento de nuestro propio código de configuración de Spring:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xmlns:context="http://www.springframework.org/schema/context"
5     xmlns:jdbc="http://www.springframework.org/schema/jdbc"
6     xsi:schemaLocation="http://www.springframework.org/schema/beans
7     http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
8     http://www.springframework.org/schema/context
9     http://www.springframework.org/schema/context/spring-context-3.1.xsd">
10
11     <context:component-scan base-package="es.fpbasics.servicio" />
12     <context:component-scan base-package="es.fpbasics.servicioimpl" />
13     <context:component-scan base-package="es.fpbasics.repositorio" />
14     <context:component-scan base-package="es.fpbasics.repositorioimpl" />
15
16     <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
17         <property name="driverClassName" value="net.sourceforge.jtds.jdbc.Driver" />
18         <property name="url" value="jdbc:jtds:sqlserver://localhost:1433/FPBASICS;instance=SQLEXPRESS" />
19         <property name="username" value="sa" />
20         <property name="password" value="Temporal22" />
21     </bean>
```

En este fragmento de código, podemos ver el ApplicationContext.xml, este archivo es uno de los archivos de configuración más importantes, puesto que en el declaramos los paquetes de modelos y repositorios que son necesario para el correcto funcionamiento de la integración de Spring. A parte de declarar estos componentes, también declaramos la configuración necesaria de los drivers de conexión a la base de datos y sus credenciales. (También podríamos declarar tras muchas cosas como por ejemplo los Handlers, que no han sido necesarios en este proyecto).

Otro archivo de configuración bastante interesante, el web.xml el cual nos permite, por ejemplo, definir cuál es la página de arranque del proyecto cuando accedemos a la interfaz entre otras cosas.

```

3    <display-name>MyBatis + Spring + JSF2 + Primefaces</display-name>
4    <display-name>Cursos_catalogo</display-name>
5    <context-param>
6        <param-name>contextConfigLocation</param-name>
7        <param-value>
8            /WEB-INF/applicationContext.xml
9        </param-value>
10    </context-param>
11    <context-param>
12        <param-name>javax.faces.PROJECT_STAGE</param-name>
13        <param-value>Development</param-value>
14    </context-param>
15    <context-param>
16        <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
17        <param-value>.xhtml</param-value>
18    </context-param>
19    <welcome-file-list>
20        <welcome-file>index.xhtml</welcome-file>
21    </welcome-file-list>
22    <servlet>
23        <servlet-name>Faces Servlet</servlet-name>
24        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
25        <load-on-startup>1</load-on-startup>
26    </servlet>

```

Anotaciones.

Las anotaciones se usan para reducir el tamaño de nuestro fichero de configuración XML. Algunos ejemplos de ellas, o quizás, las más utilizadas son:

- **@Autowired:** Esta anotación nos permite auto-inyectar ciertas clases de nuestra aplicación, sin necesidad de crear una instancia.
- **@Component:** Es de propósito general, indica que la clase es un componente de Spring.
- **@Repository:** Indica que la clase es un repositorio de datos.
- **@Service:** Indica que la clase define un servicio.

Conclusión sobre Spring framework:

Nos evita, a los desarrolladores, muchos problemas a la hora de integrar varios frameworks en un mismo proyecto, además, un punto a favor es que la comunidad lo mantiene continuamente actualizado.

MyBatis.

MyBatis es una herramienta de persistencia Java, que se encarga de mapear sentencias SQL y procedimientos almacenados, tales como, vistas, consultas de cualquier complejidad..., es decir, para tener el control total del SQL ejecutado.

Estas sentencias SQL se mapean en los denominados “mappers”, que son ficheros XML, llamados por clases de Java que interactúan haciendo mención a dichos ficheros. Estas clases Java permiten pasar parámetros a las sentencias alojadas en el XML mediante anotaciones “@Param”.

A continuación mostramos un ejemplo de una clase Java del mapper de la tabla “Modulos” y su respectivo fichero XML.

Los Mappers.

En este apartado mostramos un fragmento del código de nuestro programa, más concretamente la clase Java llamada “ModuloMapper.java” la cual se encarga de pasar los parámetros necesarios para ejecutar de forma correcta las sentencias SQL descritas en el fichero XML que le sucede. La llamada a esta clase viene dada desde el repositorio, anteriormente explicado.

En estos métodos se definen los parámetros mediante las anotaciones “@Param(“NombreDelParámetro”)”. Nos fijamos como ejemplo en el método señalado.

```
11 public interface ModuloMapper {
12
13     public int modificarRegistroDeLaTablaModulos(@Param("modulo")BotonModificarModulosVista registro);
14
15     public List<TablaModulos> consultarTodosLosRegistrosDeLaTablaModulos();
16
17     public TablaModulos consultarRegistrosDeLaTablaModulosPorIdModulo(@Param("idModulo")Integer idModulo) ;
18
19     public List<TablaModulos> consultarRegistrosDeLaTablaModuloFiltrado(@Param("filtro")String filtro);
20 }
```

Los XML.

El siguiente fragmento es un trozo de código de nuestro mapper llamado “ModulosMapper.xml” donde encontramos las sentencias SQL necesarias.

Esta primera captura muestra la sentencia general, que más adelante podremos invocar mediante su ID y podremos añadirle así algunas restricciones que consideremos necesarias sin tener que repetir continuamente la misma sentencia.

Es aconsejable incluir la restricción “WHERE 1=1” para poder concatenar con mayor facilidad las siguientes restricciones, ya que ésto siempre será verdadero y podremos concatenar únicamente poniendo la palabra “AND” en las siguientes restricciones.

```
12      <!-- General -->
13      <sql id="baseSelectDocumento">
14          SELECT * FROM MODULOS WHERE 1=1
15      </sql>
```

Como vemos en la siguiente captura, mediante el ID hacemos referencia al método de la clase Java que invocará esta sentencia SQL y al añadir “<include refid=“ID_DeLaSentenciaGeneral”>” podemos incorporar nuestra sentencia general y añadir lo que consideremos necesario para realizar correctamente esta sentencia SQL solamente concatenando con la palabra “AND”.

También debemos fijarnos que el parámetro que pasamos en la clase Java, mediante la anotación “@Param”, lo capturamos en esta sentencia mediante la terminología “#{filtro}”, siendo “filtro” el nombre que le dimos al parámetro en la clase Java(consultar apartado anterior).

```
<select id="consultarRegistrosDeLaTablaModuloFiltrado" resultMap="crearRegistro">
    <include refid="baseSelectDocumento"/>AND (NOMBRE LIKE #{filtro} OR CURSO LIKE #{filtro})
</select>
```

Como podemos comprobar, una vez realizada esta consulta, nos dirige a un “resultMap”.

El resultMap es sencillo de comprender, la consulta obtiene unos valores de la base de datos, y con estos valores nos crea en memoria un objeto por cada registro obtenido con los atributos correspondientes, es importante que los atributos tengan el mismo nombre que las columnas de nuestra base de datos.

El resultMap tiene un ID que es referenciado en nuestra sentencia SQL mediante el parámetro “resultMap=“ID_resultMap””,(captura anterior).

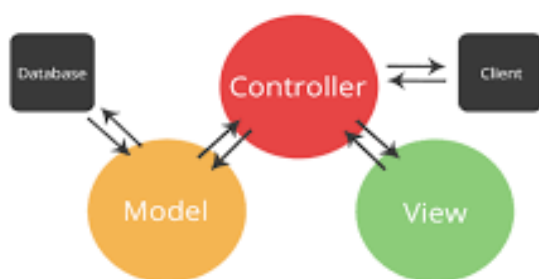
Con el parámetro “type=“Paquetes.NombreDeLaClase”” le indicamos al resultMap donde puede encontrar la clase y sus atributos para crear los objetos. Esta clase debe tener un constructor que reciba como parámetro cada uno de los atributos, ya que el resultMap lo usará para crear los objetos.

```
<resultMap id="crearRegistro" type="es.fpbasics.modelo.TablaModulos">
  <constructor>
    <idArg column="ID_MODULO" javaType="Integer" />
    <arg column="NOMBRE" javaType="String" />
    <arg column="DURACION" javaType="Integer" />
    <arg column="CURSO" javaType="String" />
  </constructor>
</resultMap>
```

Modelo Vista-Controlador(MVC).

Este modelo es un estilo de arquitectura de software que se caracteriza por separar los datos, la interfaz de usuario y la lógica de control en tres componentes distintos, es decir, estos están separados.

La eficacia de este modelo ha sido demostrada a lo largo de los años en distintas plataformas de desarrollo y multitud de lenguajes.



Decidimos aplicar esta arquitectura a nuestro proyecto porque se adecua perfectamente a este y sobre todo al problema planteado como caso práctico.

A grandes rasgos, esto es lo que caracteriza a cada parte:

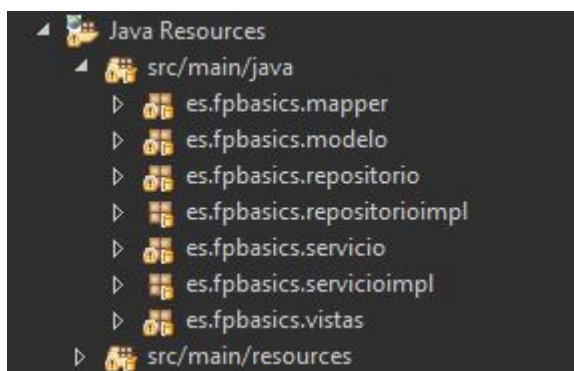
Por una parte el Modelo contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia. En este caso, nuestro proyecto posee varias capas a nivel de modelo, pues utiliza la estructura Servicio-Repositorio-Modelo.

Por otro lado, la Vista, o interfaz de usuario, compone la información que se envía al cliente y los mecanismos interacción con éste. Nuestra aplicación cuenta con una interfaz de usuario vía web que hemos desarrollado con el framework JSF (explicado más adelante).

Para finalizar el Controlador, se encuentra entre el Modelo y la Vista, este actúa como intermediario entre ambos, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

Podríamos decir que los controladores de nuestra aplicación son las “vistas”, pues estas trabajan con los servicios (que ya se encargan de obtener la información requerida) y se encargan de ofrecer la información a la interfaz, para poder mostrarla.

Aquí podemos ver un esquema general de la aplicación:



Dentro de “src/main/resources” encontramos todos los archivos .xhtml correspondiente la interfaz, y los recursos que esta necesita.

Por último debemos recalcar que este tipo de arquitectura hace que nuestra aplicación sea completamente modular, es decir, cualquiera de las partes de estas podrían ser sustituidas sin que fuese necesario reestructurar la aplicación completa. Esto es muy importante.

Además, es interesante recordar que la estructura Servicio-Repositorio-Modelo permite encapsular la aplicación. En lugar de tener un único “todo”, tenemos distintas piezas que se unen para formar un puzzle.

JavaServer Faces (JSF).

JavaServer Faces o JSF es un framework dedicado a aplicaciones Java web que simplifica el desarrollo de interfaces web. Este está basado en el MVC (mencionado anteriormente) de tipo 1, en el que las vistas conocen la acción que se va a invocar en su petición. Normalmente estas conocen las acciones que invocarán en su petición, y estas están definidas dentro de la vista.

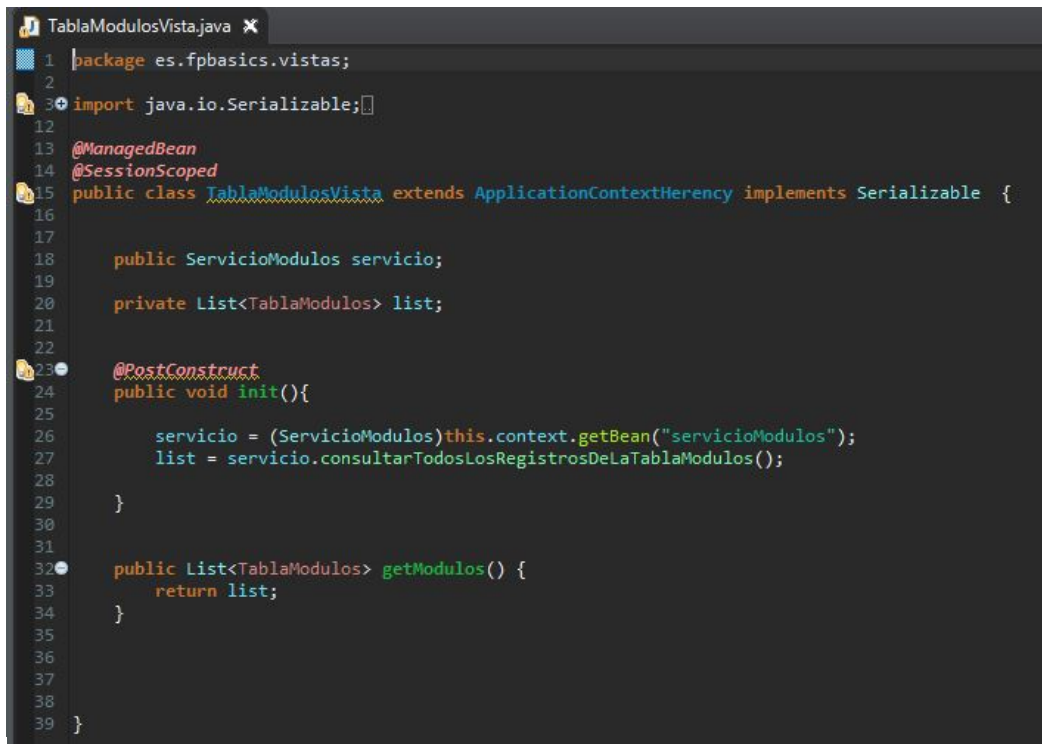
Está basado en su antecesor JSP y permite hacer el despliegue de las páginas, además de acomodarse a otras tecnologías como XUL (acrónimo de XML-based User-interface Language, lenguaje basado en XML para la interfaz de usuario). También permite introducir javascript en la página para acelerar la respuesta de la interfaz.

Está formado por un conjunto de APIs para representar componentes de una interfaz de usuario y administrar su estado, manejar eventos, validar la entrada de datos, definir un esquema de navegación de las páginas...

JSF utiliza el concepto de JavaBeans o Beans, un modelo de componentes que permiten encapsular varios objetos en uno único (al que llamamos Bean). Estos deben obedecer ciertas convenciones sobre nomenclatura de métodos, construcción y comportamiento.

Es decir, asocia a cada vista con formularios un conjunto de objetos Java manejados por el controlador (managed beans) que facilitan la recogida, manipulación y visualización de los valores mostrados en los diferentes elementos de los formularios.

Aquí podemos ver un ejemplo de una vista, observamos las anotaciones necesarias para configurarla:



```
1 package es.fpbasics.vistas;
2
3 import java.io.Serializable;
4
5
6
7
8
9
10
11
12 @ManagedBean
13 @SessionScoped
14 public class TablaModulosVista extends ApplicationContextHerency implements Serializable {
15
16
17     public ServicioModulos servicio;
18
19     private List<TablaModulos> list;
20
21
22
23 @PostConstruct
24 public void init(){
25     servicio = (ServicioModulos)this.context.getBean("servicioModulos");
26     list = servicio.consultarTodosLosRegistrosDeLaTablaModulos();
27
28 }
29
30
31
32 public List<TablaModulos> getModulos() {
33     return list;
34 }
35
36
37
38
39 }
```

Tiene como objetivo establecer un conjunto simple de clases base que puedan ser utilizadas como componentes de la interfaz de usuario. Estas clases tratarán los aspectos del ciclo de vida de la interfaz de usuario, controlando el estado de un componente durante el ciclo de vida de su página.

Este sería un fragmento de uno de los archivos asociados a esta vista:

```
<br/>
<p:dataTable var="modulo" value="#{tablaModulosVista.modulos}">
  <p:column headerText="Id Módulo">
    <h:outputText value="#{modulo.idModulo}" />
  </p:column>
  <p:column headerText="Nombre">
    <h:outputText value="#{modulo.nombre}" />
  </p:column>
  <p:column headerText="Duración">
    <h:outputText value="#{modulo.duracion}" />
  </p:column>
  <p:column headerText="Curso">
    <h:outputText value="#{modulo.curso}" />
  </p:column>
  <p:column headerText="Seleccionar registro a modificar:">
    <p:commandLink id="modificar"
      action="ActualizarRegistroTablaModulos.xhtml"
      actionListener="#{botonModificarModulosVista.botonParaUnaPrimaria(modulo.idModulo)}"
      ajax="false" onclick="PF('carga').show()">
      <h:outputText value="Actualizar" />
    </p:commandLink>
  </p:column>
</p:dataTable>
```

En este podemos ver como en a través del atributo “value” de la tabla se está accediendo al método getter de la lista de registros que el servicio le proporciona a la vista. De esta forma vemos cómo desde la interfaz podemos acceder al código java que recibirá la petición cuando la interfaz se cargue en el servidor.

A continuación pasaremos a hablar de una librería bastante de componentes muy extensa y que es vital para JSF. Esta es PrimeFaces.

PrimeFaces.

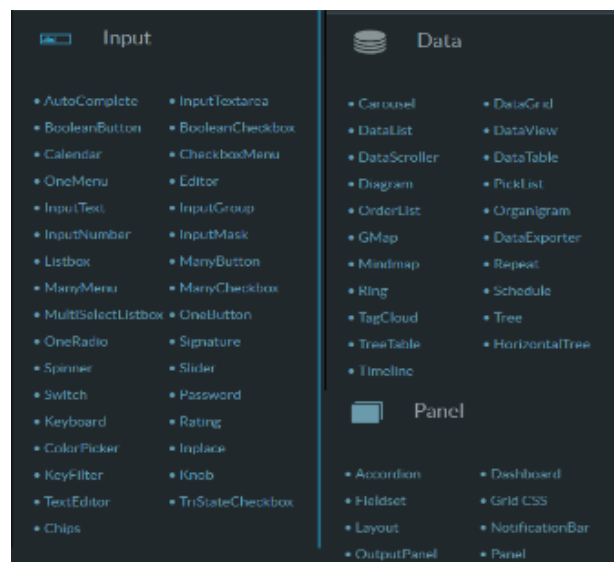
PrimeFaces es una librería de componentes de código abierto para JSF 2.0 con una gran cantidad de componentes a nuestra entera disposición. Es mucho mejor que otras bibliotecas de componentes JSF, debido a varias razones, podemos mencionar algunas:

PrimeFaces posee un amplio conjunto de componentes para la interfaz de usuario, componentes como, Inputs(AutoComplete, CheckboxMenu, InputNumber...), Data(DataGrid, DataView, DataTable...), Paneles(Panel Grid,Scroll panel...), Menús , Multimedias, y un largo etcétera... que permiten crear una interfaz con un acabado muy profesional y limpio.

Podemos encontrar todos estos componentes en el siguiente enlace:

<https://www.primefaces.org/showcase/>

A continuación se muestran algunos ejemplos de los componentes con los que podremos trabajar:



A diferencia de su antecesor JSP que posee una etiquetas muy básicas, PrimeFaces permite crear una interfaz muy customizada. Además, también podemos personalizar los propios complementos.

En resumen, PrimeFaces nos ofrece las herramientas necesarias para crear una interfaz web muy dinámica, novedosa y con un espectacular acabado.

Como mencionamos anteriormente, JSF permite enlazar instrucciones con eventos y todos estos componentes pueden acceder al código Java. Esto lo hace bastante más potente que JSP debido a que en este último el código Java es embebido en la propia página.

Otra punto a destacar de este framework es que introduce un procesamiento de la petición, como por ejemplo el de validación, reconstrucción de la vista o el recuperación de los valores de los elementos.

Pero sin duda uno de sus puntos más fuertes es que forma parte del estándar J2EE. Esto es cierto, pues aunque existen muchas alternativas para crear la capa de presentación y control de una aplicación web java, como Struts y otros frameworks, solo JSP y JSF forman parte del estándar.

La interfaz.

A continuación se muestra el funcionamiento de la aplicación y las acciones que se van encadenando a medida que vamos interactuando con la interfaz con una serie de casos prácticos.

Una vez es iniciado el servidor de aplicaciones Apache Tomcat, nuestra aplicación será accesible vía web a través de la siguiente dirección:

<http://localhost:8080/PruebaTecnica/index.xhtml>

Si accedemos a este recurso, se hará una petición al servidor, que se encargará de cargar la página de inicio de la aplicación. Como ya hemos mencionado, esta interfaz irá creando una serie de eventos e irá accediendo al código del back-end de la aplicación según el flujo de ejecución del programa.

Comenzaremos planteando una visión general de la ejecución de la aplicación.

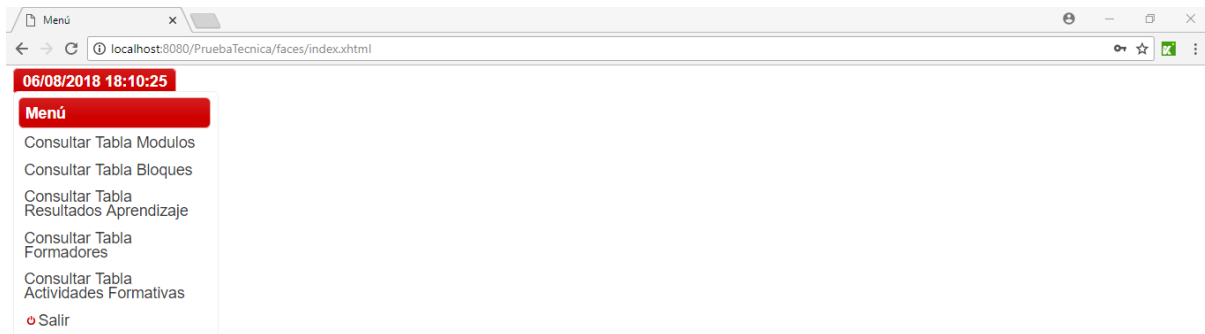
Visión General.

En primer lugar, justo al iniciar el programa se carga el contenido del “index.xhtml” que se trata de un “login”. Este login lo tenemos configurado mediante métodos de acceso en Java que solo aceptan “usuario=usuario;contraseña=usuario” (En el apartado de futuras mejoras retomamos este tema).

Una vez iniciamos sesión mediante el Login y pulsamos el botón “Login” nos redirige a otro archivo.xhtml, esta vez a “menu.xhtml”, en el cual encontramos un menú con las distintas opciones disponibles para su uso.

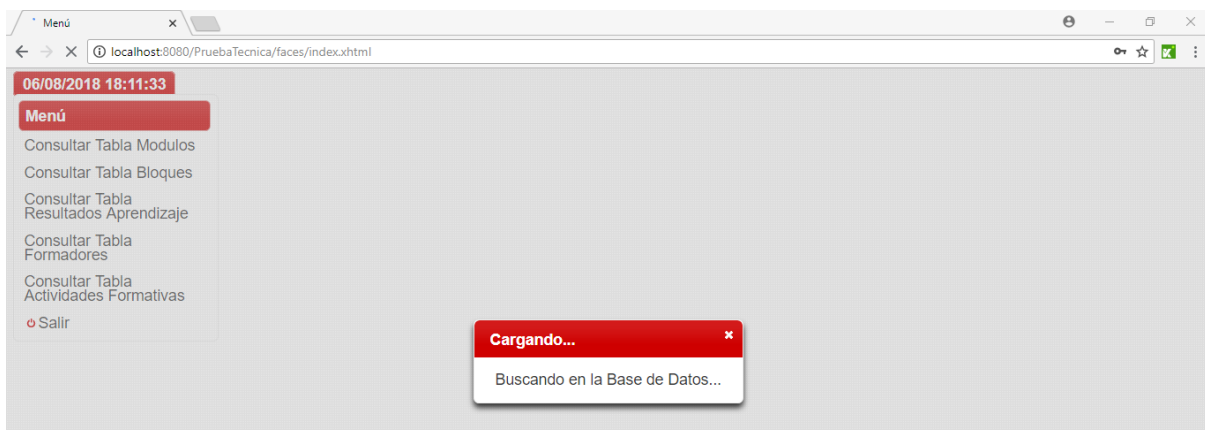
A screenshot of a web browser window. The address bar shows the URL "localhost:8080/PruebaTecnica/faces/index.xhtml". The page has a red header bar with the word "Login" in white. Below the header, there is a form with two input fields: "Usuario:" with the text "usuario" and "Password:". Below the password field is a blue button labeled "Login".

Como se aprecia en la siguiente captura, encontramos cinco botones de consulta y un sexto para salir y volver de nuevo a la ventana de Login.



Cuando hacemos clic en una de las opciones de consulta que se encuentran disponibles en el menú, lo primero que realiza el programa automáticamente es una consulta completa de la tabla seleccionada(`SELECT * FROM nombreTabla;`).

Como se aprecia en la siguiente imagen, hemos añadido un mensaje de espera, el cual indica al usuario que se está tramitando su petición a la base de datos.



Una vez recuperados los datos de la bases de datos, son mostrados en su correspondiente xhtml, en este caso se trata de “tablaModulos.xhtml” que contiene todo el código JSF necesario para mostrar los datos en una forma dinámica y agradable para el usuario. (Código JSF visto en el apartado anterior).



Registros de la tabla Módulos

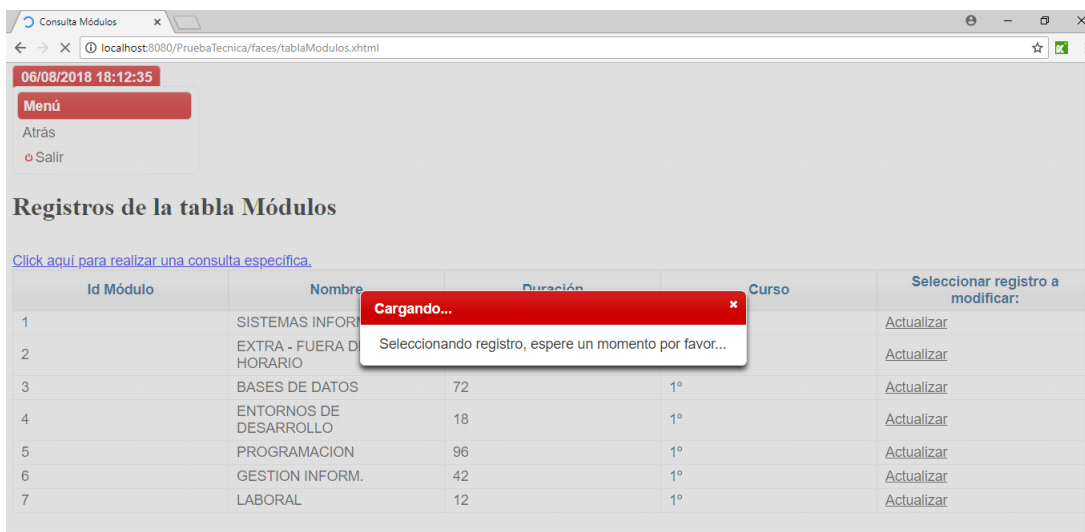
[Click aquí para realizar una consulta específica.](#)

Id Módulo	Nombre	Duración	Curso	Seleccionar registro a modificar:
1	SISTEMAS INFORMATICOS	48	1º	Actualizar
2	EXTRA - FUERA DE HORARIO	15	Co	Actualizar
3	BASES DE DATOS	72	1º	Actualizar
4	ENTORNOS DE DESARROLLO	18	1º	Actualizar
5	PROGRAMACION	96	1º	Actualizar
6	GESTION INFORM.	42	1º	Actualizar
7	LABORAL	12	1º	Actualizar

Como vemos en la imagen anterior, podemos hacer clic en el botón “Actualizar” para realizar modificaciones sobre un registro concreto de la tabla que estemos visualizando en ese momento.

Como ejemplo, hacemos clic sobre el registro con id=1, y comprobamos que esta vez nos redirige a otra página donde disponemos de los valores actuales que tiene dicho registro en la base de datos y la posibilidad de modificar todos los campos que no sean clave primaria(Primary Key).

Al hacer clic, nuevamente disponemos de una pequeña pantalla de carga mientras se extraen los datos correspondientes de la base de datos.



06/08/2018 18:13:31

Menú

Atrás

Salir

Actualizar registro tabla Módulos:

Id Módulo	Nombre	Duración	Curso
1	SISTEMAS INFORMATICOS	48	1º

Id Módulo	Nombre	Duración	Curso
1	INFORMATIC SYSTEMS	10	2º

[Enviar](#)

Una vez ingresemos los nuevos valores que deseamos pulsamos en “Enviar” y pasamos a la siguiente ventana, que no es más que una ventana de confirmación de cambios, donde se nos muestra los valores que vamos a introducir en los distintos campos y se nos pregunta si estamos seguro de realizar dichos cambios, ya que serán permanentes. Si todo está correcto hacemos clic en “Sí, estoy seguro” para continuar, en cambio, si no estamos de acuerdo pulsamos sobre “No, volver atrás” para volver a la pantalla de Menú.

En nuestro ejemplo, pulsaremos “Sí, estoy seguro” para continuar.

Justo al hacer clic en “Sí, estoy seguro” aparecerá de nuevo la pantalla de carga para indicar al usuario que el programa está tramitando su petición.

06/08/2018 18:13:52

Menú

Salir

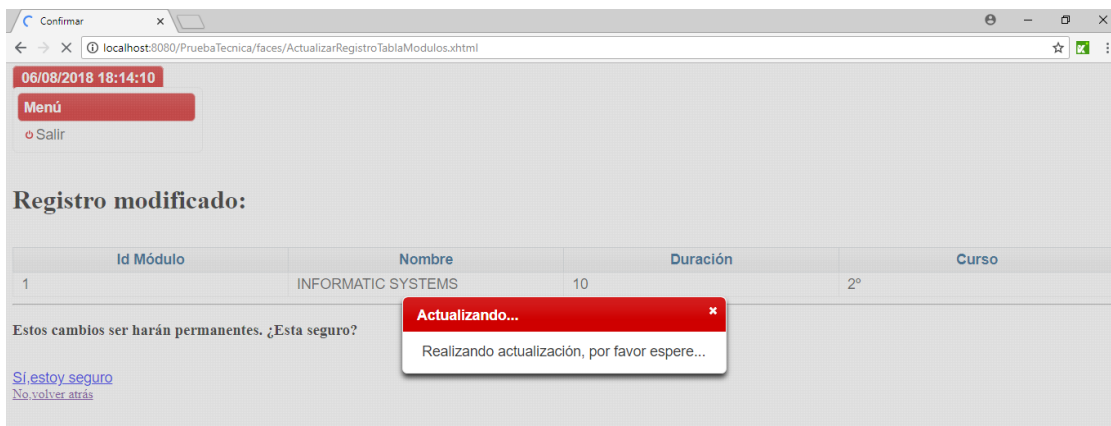
Registro modificado:

Id Módulo	Nombre	Duración	Curso
1	INFORMATIC SYSTEMS	10	2º

Estos cambios ser harán permanentes. ¿Esta seguro?

[Si estoy seguro](#)

[No volver atrás](#)



Para terminar, cuando esta pantalla desaparezca y verificar que los cambios que se han realizado son permanentes el programa automáticamente nos dirigirá a la última ventana de la modificación, que se trata de una consulta de todos los registros de dicha tabla para que podamos comprobar que los cambios se han realizado con éxito y nuestro viejo registro a sido modificado por los valores que anteriormente hemos introducido (esta sentencia vuelve a ser de nuevo un `SELECT * FROM nombreTabla;`).



Registros de la tabla Módulos

[Click aquí para realizar una consulta específica.](#)

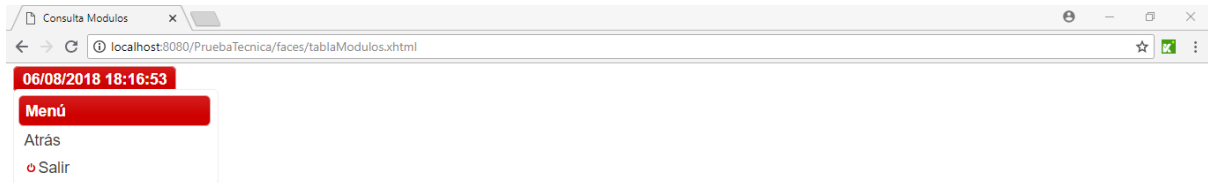
Id Módulo	Nombre	Duración	Curso	Seleccionar registro a modificar:
1	INFORMATIC SYSTEMS	10	2°	Actualizar
2	EXTRA - FUERA DE HORARIO	15	Co	Actualizar
3	BASES DE DATOS	72	1°	Actualizar
4	ENTORNOS DE DESARROLLO	18	1°	Actualizar
5	PROGRAMACION	96	1°	Actualizar
6	GESTION INFORM.	42	1°	Actualizar
7	LABORAL	12	1°	Actualizar

Por otro lado, debemos de explicar otra función de nuestra aplicación, y es la búsqueda mediante filtrado.

Cada vez que realicemos una búsqueda completa, nos encontraremos un enlace que dice “Click aquí para realizar una consulta específica”, que como su nombre indica, es para realizar una búsqueda de un registro concreto en nuestra base de datos.

Si hacemos clic en dicho enlace, nos dirigirá a la siguiente ventana.

En esta ventana, dependiendo de la tabla que estemos consultando, dispondremos de un campo para insertar un parámetro de búsqueda. Dicho parámetro vendrá determinado por el programa, en este ejemplo, que se trata de la tabla módulos, el programa nos permite realizar una búsqueda por “nombre” y/o “curso”.



Realizar consulta filtrada:

Inserte nombre del módulo y/o curso:

[Enviar](#)

Para realizar un ejemplo gráfico, vamos a realizar la búsqueda del registro anteriormente modificado. Como se muestra en la siguiente imagen, realizamos la búsqueda de todos los registros que comiencen por “Informa”.



Realizar consulta filtrada:

Inserte nombre del módulo y/o curso:

[Enviar](#)

Otra vez disponemos del detalle con el usuario de indicarle que la aplicación está tramitando su consulta.



Para terminar esta demostración, aquí mostramos cómo, efectivamente, la consulta se ha realizado con éxito y ha extraído todos los registros que comienzan por “Informa” (en nuestro caso únicamente disponemos de un registro que cumpla este requisito).



Registros de la tabla Módulos

Id Módulo	Nombre	Duración	Curso	Seleccionar registro a modificar:
1	INFORMATIC SYSTEMS	10	2º	Actualizar

Dificultades encontradas.

A lo largo del desarrollo de este proyecto nos hemos encontrado diversas dificultades, que nos han retrasado en mayor o menor medida. A continuación mostramos una lista de las principales dificultades que nos hemos encontrado.

MyBatis y Accesos.

Uno de los problemas que nos encontramos fue a la hora de definir de que forma íbamos a acceder a la base de datos, puesto que buscábamos una forma de acceder a ella que pudiera ser utilizada por todos los métodos de nuestra app.

Este problema fue solucionado modularizando los accesos a la base de datos en primer lugar según la tabla de la base de datos que se consulta y en segundo lugar creando una forma de acceso diferente para cada uno de los métodos de la aplicación.

Sistemas Gestores de Bases de Datos.

Otro problema que nos encontramos a lo largo del desarrollo fue las ligeras incompatibilidades entre los sistemas gestores de bases de datos, pero este problema se soluciona con pequeñas refactorizaciones de código usando la función “Replace Next” de Notepad++ o de Sublime Text.

También tuvimos el problema de que el desarrollo se llevó a cabo sobre una base de datos de SQL Server 2017, pero en las máquinas virtuales estábamos intentando ejecutar servidores Oracle. Por lo que el driver que utilizábamos para la conexión no era compatible.

Decidimos utilizar el mismo gestor de base datos que utilizábamos para el desarrollo (SQL Server 2017) pero al usar un sistema operativo Windows 7 en lugar de Windows 10 (debido a que el consumo de recursos de Windows 7 es mucho menor) no podíamos cumplir los requisitos mínimos (pues pide Windows 8 de requisitos mínimos)

Finalmente decidimos hacer una copia de la máquina virtual que usábamos para el desarrollo y emular la máquina virtual en el ordenador personal de uno de los participantes de este proyecto.

Sobre la integración entre Spring y JSF.

El mayor problema al que nos hemos enfrentado a lo largo de este proyecto consistió en la integración del framework Spring con JSF, este problema tenía muchas causas y pasamos bastante tiempo dando palos de ciego, puesto que en muchas ocasiones no podíamos ver si las correcciones que aplicamos surtían efecto o no.

Describiendo con mayor profundidad el problema, podemos comentar que cuando esperábamos que el servidor TomCat activara la interfaz gráfica en JSF, ésta inyectara la estructura Servicio-Repositorio-Modelo y todo el backend del que dependía la aplicación.

La solución a estos errores vino cuando uno de nuestros formadores de Aytos nos recomendó pedirle ayuda a su compañero en la formación, pues este había trabajado en desarrollo y tenía experiencia en este tipo de integraciones.

El origen de error (el cual se producía en mayor medida por nuestra falta de experiencia) tenía 3 pilares básicos: La inyección de recursos, los archivos de configuración y el driver de acceso a Base de Datos(problema comentado en el punto anterior).

Inyección Automática e Inyección Manual.

El problema de la inyección consistía en que a la hora de que las clases Vista inyectaran los recursos(repositorios, servicios y mappers), utilizábamos la anotación “@Autowired”, pero esta no funcionaba correctamente.

La solución fue inyectar manualmente los recursos necesarios.

Para ello usamos la anotación “@PostConstruct”, la cual indica a Spring, que cuando cargue el contexto de la aplicación al ser iniciada, el primer método que se ejecutará sea ese, el cual es un método sobrescrito llamado “init()”.

En este método se instancia el “context” que es extendido de “ApplicationContextHerency”, el cual incorpora la configuración necesaria para desplegar todo el contexto con Spring.

Este sería su contenido:

```
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class ApplicationContextHerency {

    ClassPathXmlApplicationContext context =
        new ClassPathXmlApplicationContext(new String[] {"applicationContext.xml"});
}
```

Si nos fijamos, esta clase hace uso de “applicationContext.xml”, que es donde se define el contexto a desplegar, es decir, donde indicamos a Spring dónde buscar lo que tiene que desplegar al iniciar la aplicación.

Su contenido lo hemos visto antes, en el apartado “Spring Framework”.

La clase llamada “ApplicationContextHerency” nos la proporcionó uno de nuestros formadores, ya que es una clase que escapaba a nuestros conocimientos.

Nos enseñó a implementarla para que funcionara correctamente como se muestra en la siguiente captura.

```
public class TablaModulosVista extends ApplicationContextHerency implements Serializable {

    public ServicioModulos servicio;

    private List<TablaModulos> list;

    @PostConstruct
    public void init(){

        servicio = (ServicioModulos)this.context.getBean("servicioModulos");
        list = servicio.consultarTodosLosRegistrosDeLaTablaModulos();

    }
```

Tener en cuenta que en “getBean()” se debe indicar el nombre del servicio que vamos a instanciar y usar en esta clase.

Los archivos de configuración.

A su vez, a parte del problema de la inyección tuvimos el problema de que desconocíamos que necesitábamos más archivos de configuración, que los que teníamos en ese momento.

La solución la encontramos cuando el formador que nos iba a ayudar más adelante con el problema de la inyección, nos dio una pista, al mencionar que nos hacía falta un archivo llamado applicationContext.xml. Con esta pista, localizamos un repositorio public con un proyecto muy parecido al nuestro, del cual pudimos ver la estructura y el uso que hacía de este archivo.

Una vez comprendido su funcionamiento, nos resultó muy sencillo crear un archivo similar, que hiciera las funciones que nosotros necesitábamos.

En este archivo, se señala a los componentes conocidos como “Beans” de los que tendrá que hacer uso el programa, así como indicar el driver y la ruta de la base de datos con la que trabajamos.

Problemas con Servidor TomCat

A lo largo del desarrollo del proyecto, nos encontramos con el problema de que el servidor Tomcat en el que emulábamos nuestra aplicación, en muchos casos dejaba de funcionar para alguno de nosotros, por lo que finalmente buscamos una solución sencilla, que consistió en exportar un modelo de configuración del servidor Tomcat estable a nuestro Drive del proyecto y en caso de que a alguno de nosotros nos fallara, solo tendríamos que borrar el anterior servidor y volver a cargar el estable.

Conclusiones.

Aplicaciones Reales del Proyecto.

La proyección futura del proyecto podría llegar a ser bastante útil, puesto que cubre una necesidad real surgida recientemente a raíz de la creación del Ciclo Dual de Desarrollo de Aplicaciones Web.

Si esta aplicación se ampliase lo suficiente, podría aplicarse a cualquier ciclo de formación dual, y permitiría la gestión de los datos referentes a la formación en las empresas.

Futuras Mejoras.

En cuanto a las futuras mejoras, podemos añadir diversas mejoras, a continuación mencionaremos algunas mejoras que pueden mejorar considerablemente nuestra aplicación:

1. La primera mejora que podemos añadir sería incluir botones de filtrado automático, de forma que podamos realizar, por ejemplo, una consulta completa de la tabla módulos, y que al hacer clic en un registro, podamos obtener todos los resultados de aprendizaje relacionados con dicho Módulo.

En definitiva, realizar una interfaz gráfica más dinámica, que no tengamos que volver atrás cada vez que queramos realizar una consulta, poder ir “pegando saltos” de tabla en tabla mediante sus relaciones.

2. Otra incorporación que puede mejorar bastante la seguridad de nuestra aplicación sería incluir un login, conectado a una base de datos donde guardar los usuarios y sus contraseñas cifradas, de modo que sea muy difícil falsear el inicio de sesión en nuestra aplicación.
3. También nos interesaría añadir la opciones de Insertar y Eliminar registros de la base de datos. Pues actualmente solo se trata de una aplicación de consulta y modificación.

De esta forma nuestra aplicación ofrecería un CRUD completo utilizando todas estas tecnologías tan novedosas e innovadoras.

4. Debido al gran problema que encontramos para realizar la integración entre spring y JSF, perdimos una gran cantidad de tiempo que podríamos haber invertido en otros puntos, como el desarrollo de la interfaz. Es por esto que podríamos incluir como futura mejora, modificar la forma en la que las vistas tratan la información que reciben de los servicios, para ofrecersela a la vistas de una forma más óptima.

En resumen, nuestra solución planteada es eficaz, pero no es óptima. Por tanto podríamos optimizarla reestructurando estos aspectos mencionados

Ventajas y Desventajas de Java.

Después de buscar información sobre la opinión de los usuarios de Java en varios foros encontrados en Internet, hemos llegado a extraer una serie de ventajas en las que todos los desarrolladores Java coinciden:

1. Lenguaje es Multi-plataforma: es lenguaje interpretado, por lo que el programa desarrollado en Java será funcional en cualquier plataforma que disponga de dicho intérprete.
2. Manejo automático de la memoria: a diferencia de multitud de lenguajes, Java maneja la memoria de forma automática utilizando su “garbage collector”, recolector de basura. Es decir, libera al desarrollador de la carga de estar pendiente de la memoria usada por el programa.
3. Es un lenguaje relativamente fácil de aprender, pues está diseñado para asemejarse a la forma de pensar de un humano.
4. Se puede usar para desarrollar tanto aplicaciones Web, como móviles o de escritorio.
5. Existen multitud de librerías y frameworks gratuitos, como por ejemplo los vistos en esta documentación.

Desde luego, no todo van a ser ventajas, también encontramos algunas desventajas:

1. Una mala implementación de un programa en Java, puede resultar algo muy lento a la hora de funcionar.
2. Algunas herramientas, bastante útiles, tienen un costo adicional.
3. Algunas implementaciones y librerías pueden tener un código bastante complejo.
4. Requiere de un intérprete.

Estas son las ventajas y desventajas más mencionadas por la comunidad de desarrolladores que hemos encontrado en la web.

Fuentes de Información y agradecimientos.

Durante la realización de este proyecto hemos recibido ayuda directa o indirecta por parte de muchos compañeros y profesores. Queremos hacer un pequeño paréntesis al final de este proyecto para mencionar y hacer un pequeño agradecimiento a estas personas.

A todos los formadores de Aytos, por conseguir que cada día aprendiéramos algo nuevo, conseguir mantener nuestra atención con clases dinámicas, casos prácticos reales y por no olvidarse de nosotros a pesar de estar al fondo, ser de otro módulo y a veces no salir ni en las listas a la hora de pasarlas. Y además de todo esto dar siempre una buena referencia de nosotros y demostrarnos que se valora el esfuerzo de los que intentamos hacernos hueco en este mundillo..

A Antonio Calle, por orientarnos en las primeras fases de nuestro proyecto cuando aún nos encontrábamos bastante perdidos y mostrarnos que una interfaz web da mucha menos lata que una interfaz de escritorio, además de ser mucho más útil hoy en día.

A Guille Morilla, por interesarse por el proyecto siempre que veía a su primo (Javi García) y por darnos el nombre del compañero que nos llevaría a la solución del mayor escollo que tuvimos durante este proyecto.

Definitivamente a Jesús Carmona, por ayudarnos a resolver el GRAN PROBLEMA que encontramos a mediados del proyecto, si el cual aún seguiríamos dando palos de ciego para arreglar el estropicio que teníamos entre manos.

A todos los profesores que nos han formado durante estos 2 años en ASIR, mostrando paciencia y mucha profesionalidad, pero especialmente a...

Jose Antonio Muñoz, por iniciarnos en el mundo de Java y como tutor de nuestro proyecto y de clase estar siempre a nuestra disposición para orientarnos en todo lo que pudiera incluso si el problema se le escapara un poco de las manos.

Juan Farfán, por mover los hilos y ofrecernos la posibilidad de hacer nuestro periodo de formación en una de las empresas más grandes y punteras como es Aytos Berger-Levrault, no solo de nuestro pueblo o comunidad, si no de nuestro país e incluso de Europa, demostrandonos que no importa tanto cual sea tu oficio, si no cual sea tu vocación.

A los compañeros del ciclo de DAW, por recibirnos con los brazos abiertos y esos momentos libres en los que podíamos desconectar y reinos como unos más del grupo.

Enlaces de Referencia:

Diseño de BD:

<https://www.lucidchart.com/pages/es/tutorial-de-estructura-y-dise%C3%B1o-de-bases-de-datos>

Maven:

<http://panamahitek.com/que-es-maven-y-para-que-se-utiliza/>

MyBatis:

<https://es.wikipedia.org/wiki/MyBatis>

MVC:

<https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html>

JSF y PrimeFaces:

https://es.wikipedia.org/wiki/JavaServer_Faces

<https://josemmsimo.wordpress.com/2012/07/30/jsf-caracteristicas-principales-ventajas-y-puntos-a-destacar/>

<https://www.primefaces.org/showcase/>