Centro Profesional
Universidad Europea Madrid
LAUREATE INTERNATIONAL UNIVERSITIES

UNIVERSIDAD EUROPEA

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

CICLO FORMATIVO DE GRADO SUPERIOR DESARROLLO DE APLICACIONES MULTIPLATAFORMA

PROYECTO FIN DE CICLO

PÁDEL RAC

Rafaela Barrena
Carlos Dizy
Alfredo Rodríguez

CURSO 2020-2022

TÍTULO: **PÁDEL RAC**

AUTOR: Rafaela Barrena Pajuelo, E. Alfredo Rodríguez Pozo, Carlos Dizy Herrero

TUTOR DEL PROYECTO: Carlos Elvira Gómez

FECHA DE LECTURA: 15 de Junio de 2022

CALIFICACIÓN:

Fdo: Rafaela Barrena Pajuelo,
E. Alfredo Rodríguez Pozo,
Carlos Dizy Herrero

Tutor/a del Proyecto : Carlos Elvira Gómez

RESUMEN:

El objetivo del presente proyecto es desarrollar una aplicación web para la gestión de un centro deportivo de pádel.

Para que el diseño de la aplicación sea lo más real posible y en un futuro poder producirla y venderla, para el uso en centros deportivos, hemos contactado con un centro deportivo ubicado en el polígono de Vallecas.

En dicho centro deportivo realizamos un estudio detallado de cuáles son las necesidades del centro, y el funcionamiento de este, y así poder describir todas las tareas que se quiere llevar a cabo a través de la aplicación.

Una vez obtenido el estudio pormenorizado de cuales son los requerimientos, hemos decidido utilizar algunas de las tecnologías mas competitivas, estables y seguras, como son, Spring Framework, Hibernate Framework, Maven, Java, Apache Tomcat y varios patrones de diseño como son MVC o Strategy.

Ya que en el transcurso del desarrollo de un proyecto es muy común encontrarse con las siguientes situaciones:

- Los requerimientos cambian constantemente; el cliente, al cual hemos presentado el proyecto actual quiere cambios que debemos aplicar.
- Queremos una arquitectura flexible; hace unos años no pensábamos en tener clientes móviles y la pregunta que nos hacemos todos es que nos deparará el futuro. Nuestra aplicación debe estar preparada para adoptar esos cambios.
- Aunque en este proyecto no pretendemos diseñar el proyecto utilizando esta metodología, pensamos que los componentes deben ser testeables, siguiendo la metodología TDD para probar cada componente en el momento de su desarrollo y no esperar a la prueba de la aplicación final.

Nuestro reto, por tanto, fue encontrar una tecnología que nos facilite el desarrollo y sobre todo evitar un problema, el cual era el acoplamiento, y lograr que la aplicación sea escalable.

No vamos a profundizar en ninguno de estos temas, pero si podemos decir que utilizando el Framework Spring, solucionamos estos aspectos.

ABSTRACT:

The objective of this project is to develop a web application for the management of a padel sports centre.

In order to make the design of the application as real as possible and in the future to be able to produce and sell it for use in sports centres, we have contacted a sports centre located in the Vallecas industrial estate.

In this sports centre we carried out a detailed study of the needs of the centre and how it works, in order to describe all the tasks to be carried out through the application.

Once the detailed study of what the requirements are obtained, we have decided to use some of the most competitive, stable and secure technologies, such as Spring Framework, Hibernate Framework, Maven, Java, Apache Tomcat and several design patterns such as MVC or Strategy.

Since in the course of developing a project it is very common to encounter the following situations:

- Requirements are constantly changing; the client, to whom we have presented the current project, wants changes that we must apply.
- We want a flexible architecture; A few years ago we didn't think about having mobile clients and the question we all ask ourselves is what the future holds. Our application must be prepared to adopt those changes.
- Although in this project we do not intend to design the project using this methodology, we think that the components must be testable, following the TDD methodology to test each component at the time of its development and not wait for the test of the final application.

Our challenge, therefore, was to find a technology that makes development easier and, above all, to avoid a problem, which was coupling, and to make the application scalable.

We are not going to delve into any of these issues, but we can say that using the Spring Framework, we solve these aspects.

AGRADECIMIENTOS

Para estos agradecimientos tratamos de encontrar una frase que resumiera el esfuerzo de estos dos años que termina con este proyecto.

Tras analizar varias situaciones nos dimos cuenta que el esfuerzo individual ha sido importante, el sacrificio y constancia muy necesarios pero sobre todo hemos aprendido que en esta profesión sin un buen equipo no hay futuro.

Por ello nos quedamos con la que más nos representa y resume esta etapa formativa que culmina con este proyecto.

"***EL TALENTO GANA PARTIDOS, PERO EL TRABAJO EN EQUIPO Y LA INTELIGENCIA GANA CAMPEONATOS"***

Michel Jordan

Gracias a todos los que de alguna manera han hecho posible esta aventura formativa.

#Familia #profesores #parejas #compañeros

Gracias.

INDICE

INDICE DE FIGURAS

1. INTRODUCCIÓN

PÁDEL RAC es una aplicación de gestion de pistas de padel, en formato web que esta orientada para centros públicos y privados.

Con esta aplicación se podrá realizar reservas, anulación y modificación de las pistas de Pádel

Contará con un diseño fácil y funcional.

1.1.1. Objetivos

Los objetivos de **PÁDEL RAC**, hacer una apliacion que pueda ayudar a los usuarios de este deporte a realizar reservas y consultas con mas rapidez.
Aplicar todos los conocimientos adquiridos en el grado, aportando nuevas tecnoliagias y mejoras en los dispositivos

1.1.2. Motivaciones

PÁDEL RAC, se crea ante la necesidad de utilizar tecnologias de vanguardia para poder adaptarlas a las necesidades de los usuarios y con ello mejorar la gestion en las entidades publicas y privades de centros deportivos.

La motivación del equipo de **PÁDEL RAC** es desarrollar por nuestros medios una aplicación funcional con tecnologías avanzadas de uso actual en las empresas.

1.1.3. Antecedentes

El equipo PÁDEL RAC, como usuarios, aficionados y jugadores de pádel, hemos sufrido la mala gestión de las reservas de pistas que tienen algunos centros deportivos.

Ante la pasividad de estas instituciones y miedo al cambio, nos vemos obligados a la creación de una aplicación web, donde, tanto el usuario como el propio centro puedan interactuar de manera fácil y efectiva.

Para ello, después de un estudio pormenorizado y detallado de cuales son los aspectos a implementar en la aplicación, hemos decidido utilizar herramientas para el diseño de una aplicación que funcione vía web, y que ademas pueda albergar todos los datos necesarios de manera local en una base de datos.
En dichos datos va incluida la información de los propios usuarios, los detalles técnicos del centro y de las pistas y lo mas importante las reservas que cada usuario realizara, en función del horario, el tamaño de la pista o la cantidad de jugadores. Además el usuario tendrá la opción de crear su propio partido, a la cual podrá añadir jugadores, u otros jugadores podrán añadirse a ese partido.

2. DESARROLLO DE LA PRÁCTICA

La metodologia de trabajo aplicada en este proyecto se basa una buena planificacion , con un desarrollo integrado de los lenguajes aprendidos y la utilizacion de las nuevas tecnologias aprendidas.

2.1. Material

El desarrollo y codificación de la aplicación se realizará bajo las siguientes tecnologías, cada una de ellas serán definidas en este apartado:

- Como marco de trabajo utilizaremos el framework Spring para la inyección de dependencias, bajo la plataforma Java EE. Utilizaremos el IDE Eclipse Enterprise para desarrollo web.

- Usaremos el patrón MVC (Modelo – Vista - Controlador), para lograr separar la interfaz de usuario de la lógica de la aplicación.

- La gestión de los datos se realizará con: o MySQL como gestor de base de datos.
- Hibernate para el mapeo objeto-relacional, y la persistencia de datos. Técnica que se usa para mapear una representación de datos desde un modelo de objeto, que en un principio serán POJO, a un modelo de datos relacional.

- Apache Tomcat 9, como servidor de la aplicación.
- JSP, CSS y Javascript, para la resolución de las vistas, estilos y scripts del lado de cliente.

2.1.1. Spring Framework

- Spring es un framework de código abierto.
- Fué creado para simplificar el desarrollo de aplicaciones empresariales usando POJO's para conseguir objetivos que antes sólo eran posibles con EJB.
- Simplificar el desarrollo en Java. Spring es un framework no intrusivo, esto quiere decir que nuestras clases serán simples clases Java (POJO´s) no teniendo que heredar de otras clases, ni implementar otras interfaces propias de Spring.
- Evitar el acoplamiento. Spring utiliza la inyección de dependencias. No crearemos el objeto sino que esperaremos a que nos lo faciliten.
- Reutilización de objetos. Mas adelante veremos como los beans creados por Spring se mantienen en un contenedor y de esta forma podremos reutilizarlos.
- Reducir el código reutilizable mediante aspectos y plantillas. Tomando como ejemplo JDBC vemos la cantidad de código que se repite para lanzar dos queries muy parecidas. Spring soluciona esto utilizando plantillas.

Componentes de Spring

Spring es un marco de trabajo modular que cuenta con una arquitectura organizada en 20 módulos diferentes, que se pueden separar en seis categorías de funcionalidad (véase la figura 2-1).
En conjunto, estos módulos proporcionan todo lo necesario para desarrollar una aplicación empresarial. Además, no es necesario basar la aplicación al completo en Spring, basta con hacer uso de aquellos módulos que se ajusten a ésta. (Spring incluso ofrece puntos de integración con otros marcos de trabajo y bibliotecas).

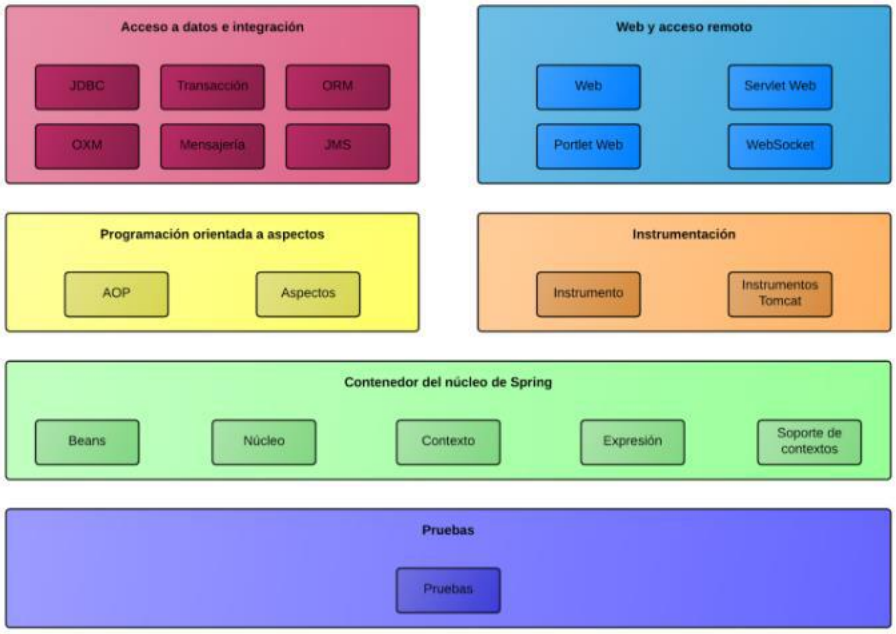


Figura 1: Estructura Spring Framework

Inyección de dependencias

La inyección de dependencias (DI) es una forma de asociar objetos de aplicación, de forma que no tengan por qué saber de dónde proceden sus dependencias o la forma en que se implementan. En lugar de adquirir las dependencias por ellos mismos, se proporciona a los objetos dependientes aquellos de los que dependen.

Cualquier aplicación está formada por una serie de objetos que deben trabajar de forma conjunta para conseguir un objetivo de negocio.

Estos objetos deben ser conscientes de la existencia de los otros y comunicarse entre sí para llevar a cabo el trabajo. Sin embargo, el enfoque tradicional al crear asociaciones entre objetos de aplicaciones (mediante construcción o búsqueda) genera código complicado, difícil de reutilizar y de probar como unidades.

En Spring, los objetos (bean) no son responsables de encontrar o crear el resto de objetos que necesitan para llevar a cabo su trabajo. En su lugar, el contenedor les asigna referencias a los objetos con los que tienen que colaborar.

En una aplicación basada en Spring, sus objetos de aplicación van a residir dentro del contenedor de Spring. El contenedor va a crear los objetos, los va a conectar, a configurar y a administrar su ciclo de vida completo.

Como se indicó en el apartado anterior, el contenedor de Spring se encuentra en el núcleo del marco de trabajo. Utiliza inyección de dependencias (DI) para administrar los componentes que forman una aplicación. Esto incluye la creación de asociaciones entre componentes que colaboran entre sí. De esta forma, los objetos están más limpios y son más fáciles de comprender, permiten su reutilización y son más fáciles de probar.

Ciclo de vida de un bean

En una aplicación Java tradicional, el ciclo de vida de un bean es sencillo. La palabra clave new de Java se utiliza para instanciar el bean. A continuación, el bean está listo para utilizarlo. Una vez deje de utilizarse, puede eliminarse.

Frente a esto, el ciclo de vida de un bean en un contenedor Spring es más complejo, pasa por varios pasos entre su creación y su eliminación. Cada paso es una oportunidad para personalizar la forma en que el bean se administra en Spring.

Los pasos son:

1. Spring instancia el bean.
2. Spring inyecta valores y referencias de bean en las propiedades de éste.
3. Si el bean implementa BeanNameAware, Spring proporciona el ID del bean al método setBeanName().
4. Si el bean implementa BeanFactoryAware, Spring invoca el método setBeanFactory(), proporcionando él mismo la fábrica de *bean*.
5. Si el *bean* implementa *ApplicationContextAware*, Spring invoca el método *setApplicationContext()*, proporcionándolo en una referencia al contexto de aplicación contenedor.
6. Si el *bean* implementa la interfaz *BeanPostProcessor*, Spring invoca su método *postProcessBeforeInitialization()*.
7. Si el *bean* implementa la interfaz *InitializingBean*, Spring invoca su método *afterPropertiesSet()*. De forma similar, si el *bean* se ha declarado con un método *init*, se invoca el método de inicialización especificado.
8. Si el *bean* implementa la interfaz *BeanPostProcessor*, Spring invoca su método *postProcessAfterInitialization()*.
9. Llegados a este punto, el *bean* estará listo para que la aplicación lo utilice, y va a permanecer en el contexto de la aplicación hasta que se elimine.
10. Si el *bean* implementa la interfaz *DisposableBean*, Spring invoca sus métodos *destroy()*. Del mismo modo, si se ha declarado con un método *destroy*, se invoca el método especificado.

Contenedores de Beans

La filosofía de Spring es la reutilización de objetos (en adelante beans). Se trata de crear un objeto y almacenarlo en un contenedor para su posterior uso.

En una aplicación desarrollada con Spring, los beans (objetos de aplicación) van a residir dentro del contenedor de beans.

Dicho contenedor se encuentra en el núcleo (Core).

El contenedor utiliza DI (Inyección de Dependencias) para crear los beans, conectarlos con otros, configurarlos y administrar su ciclo de vida.

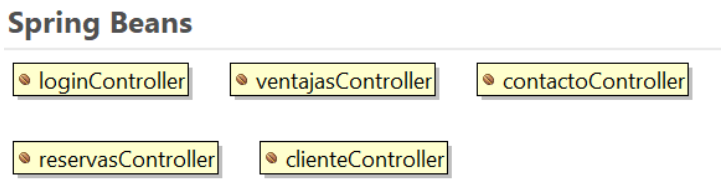


Figura 2: Estructura Beans

En adelante, se utiliza indistintamente el nombre contenedor, core-container, Spring IoC container o contexto de Spring para referirse a éste.

Marco de trabajo Spring MVC

El marco de trabajo web de Spring, Spring MVC, está basado en el *patrón* Modelo-Vista-Controlador (MVC). Ayuda a crear aplicaciones basadas en la web que son flexibles y cuentan con acoplamiento débil, al igual que el propio marco de trabajo Spring.

En este capítulo se va a tratar sobre cómo se ha creado gran parte de la capa web de la aplicación, utilizando para ello Spring MVC.

El *patrón* modelo–vista–controlador (MVC) es un patrón de arquitectura de software que separa los diferentes aspectos de una aplicación. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado, define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

De manera genérica, los componentes de MVC se podrían definir como sigue:

- El Modelo: El Modelo es el objeto que representa los datos del programa. Maneja los datos y controla todas sus transformaciones. El Modelo no tiene conocimiento específico de los Controladores o de las Vistas, ni siquiera contiene referencias a ellos. Es el propio sistema el que tiene encomendada la responsabilidad de mantener enlaces entre el Modelo y sus Vistas, y notificar a las Vistas cuando cambia el Modelo. (En el capítulo anterior se trató de forma exhaustiva este componente.)
- La Vista: La Vista es el objeto que maneja la presentación visual de los datos representados por el Modelo. Genera una representación visual del Modelo y muestra los datos al usuario. Interactúa con el Modelo a través de una referencia al propio Modelo. (En el próximo capítulo se tratará en detalle este componente.)
- El Controlador: Es el intermediario entre la vista y el modelo. Es quien controla las interacciones del usuario solicitando los datos al modelo y entregándolos a la vista para que ésta, lo presente al usuario, de forma “humanamente legible”. (En el presente capítulo se tratará en detalle este componente.)

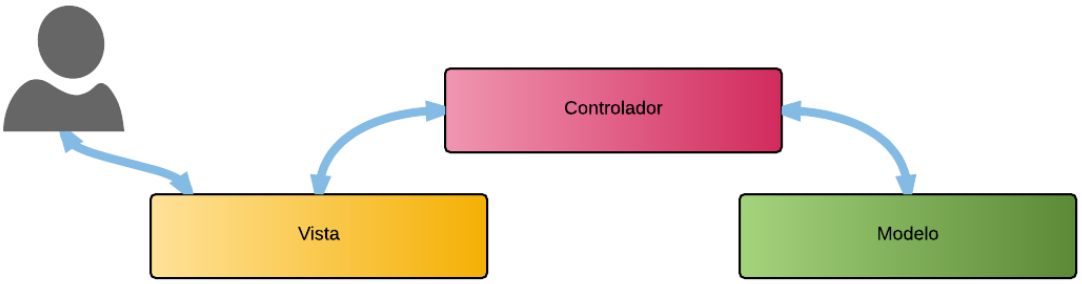


Figura 3: Patrón MVC

Ciclo de vida de una solicitud en Spring MVC

A continuación, se describe el ciclo de vida de una solicitud que, desde el cliente, recorre los componentes de Spring MVC para generar, en última instancia, una respuesta que se devuelve al cliente.

Cada vez que un usuario hace clic en un enlace o envía un formulario en su navegador Web, se pone en marcha una solicitud.

La solicitud siempre está ocupada. Desde que deja el navegador hasta que vuelve con una respuesta realiza varias paradas, y en cada una deja información y recoge otra. En la siguiente figura se pueden observar todas las paradas que realiza la solicitud.

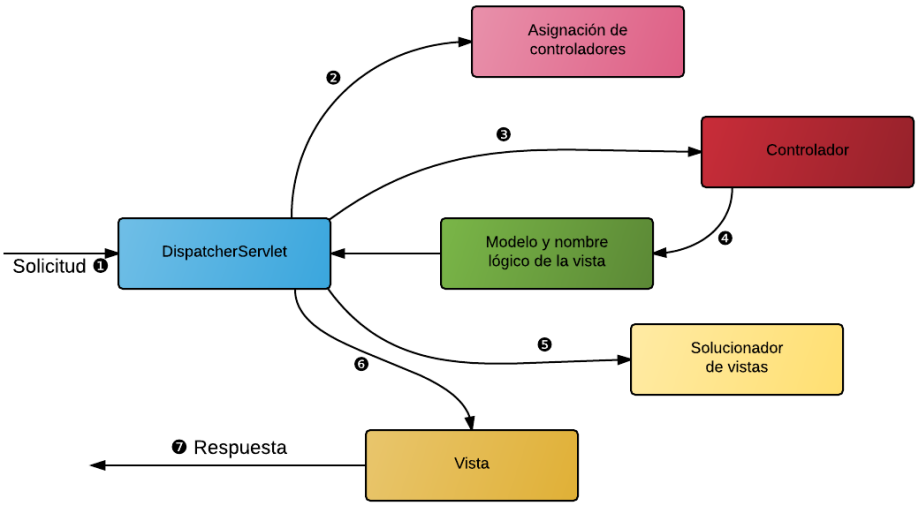


Figura 4: Ciclo de vida llamada Spring MVC

Cuando la solicitud abandona el navegador ❶, lleva información sobre lo que el usuario necesita. Como mínimo, la solicitud va a incluir la URL solicitada. Sin embargo, también puede llevar consigo información adicional, como por ejemplo la enviada por el usuario a través de un formulario.

La primera parada de la solicitud tiene lugar en DispatcherServlet de Spring. Spring MVC canaliza las solicitudes a través de este único servlet7 controlador. Un controlador frontal es un patrón común de aplicación web en el que un único servlet delega la responsabilidad de una solicitud a otros componentes de la aplicación para llevar a cabo el procesamiento. En el caso de Spring MVC, DispatcherServlet sería el controlador frontal.

La tarea de DispatcherServlet es enviar la solicitud a un controlador de Spring MVC. Un controlador es un componente de Spring que procesa la solicitud. Sin embargo, una aplicación

típica puede tener varios controladores y DispatcherServlet tiene que decidir cuál elegir. Para ello, consulta con una o más asignaciones de controlador para decidir cuál va a ser la siguiente parada de la solicitud ❷. La asignación del controlador va a prestar especial atención a la URL que transporta la solicitud a la hora de tomar la decisión.

Una vez se ha seleccionado un controlador adecuado, DispatcherServlet envía la solicitud en su camino hacia el controlador elegido ❸. En éste, la solicitud va a soltar su carga (la información enviada por el usuario) y a esperar mientras el controlador la procesa (en realidad, un controlador bien diseñado no lleva a cabo tareas de procesamiento y, en su lugar, delega la responsabilidad de la lógica de negocio a uno o más objetos de servicio).

A menudo, la lógica llevada a cabo por un controlador implica que parte de la información tiene que enviarse de nuevo al usuario y mostrarse en el navegador. Esta información recibe el nombre de modelo. Sin embargo, enviar información sin procesar al usuario no es suficiente: necesita un formato para que éste pueda consultarla. Para ello, a la información se le tiene que asignar una vista, normalmente una JSP.

Una de las últimas tareas que lleva a cabo un controlador es empaquetar los datos del modelo e identificar el nombre de la vista que debe generar el resultado. A continuación, envía la solicitud, junto con el modelo y el nombre de la vista, de vuelta a DispatcherServlet ❹.

Hasta el momento, el controlador no se acopla a ninguna vista y el nombre de la vista devuelto a DispatcherServlet no identifica una JSP específica (en concreto, ni siquiera tiene que sugerir que la vista sea una JSP). En su lugar, solo cuenta con un nombre lógico que se va a utilizar para examinar la vista que va a generar el resultado. DispatcherServlet va a consultar a un Solucionador de vistas para asignar el nombre de la vista lógica a una implementación de vista específica, que puede ser o no una JSP ❺.

Ahora que DispatcherServlet sabe qué vista va a procesar el resultado, el trabajo de la solicitud está prácticamente terminado. Su última parada es la implementación de la vista ❻, por lo general una JSP, donde entrega los datos del modelo. La vista va a utilizarlos para generar el resultado que el objeto de respuesta va a devolver al cliente ❼.

2.1.2. Acceso a Datos. My SQL

Un requisito que tienen prácticamente todas las aplicaciones es la persistencia de datos. La persistencia es un atributo de los datos que asegura que estarán disponibles incluso más allá de la vida de una aplicación, y aquí entran en juego las bases de datos. Se puede decir que una base de datos es un conjunto de datos persistentes que es utilizado por los sistemas de aplicación de alguna empresa dada.

En este capítulo se va a tratar sobre el sistema de gestión de base de datos utilizado, MySQL; el mapeo objeto-relacional, y la persistencia en aplicaciones basadas en Spring.

Para la aplicación "PadelCAR" se ha hecho uso de MySQL.

MySQL es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual GPL/Licencia comercial por Oracle Corporation y está considerada como la base datos open source más popular del mundo, y una de las más populares en general junto a Oracle y Microsoft SQL Server, sobre todo para entornos de desarrollo web.

Figura 5: MySQL

Características de MySQL

Entre las características de MySQL destacan:

- Escalabilidad y flexibilidad: Está disponible en gran cantidad de plataformas y sistemas. Ofrece la posibilidad de selección de mecanismos de almacenamiento que ofrecen diferentes velocidades de operación, soporte físico, capacidad, … Además, la naturaleza de código abierto de MySQL permite una personalización completa.
- Alto rendimiento
- Alta disponibilidad
- Apoyo transaccional robusto
- Fuerte protección de datos: Ofrece un sistema de contraseñas y privilegios seguro. Además, viene con soporte incorporado para SSL.
- Facilidad de gestión
- APIs disponibles para múltiples lenguajes de programación

2.1.3. Servidor Apache Tomcat

El servidor Apache Tomcat es un contenedor de código abierto de aplicaciones web basadas en Java. Implementa las especificaciones de los servlets y de JavaServer Pages (JSP) de Oracle Corporation (anteriormente de Sun Microsystems). En concreto, la versión 9 (que es la que se va a instalar para este proyecto), implementa las especificaciones de Servlet 3.0 y de JSP 2.2.

Fue creado en el marco del subproyecto Apache-Jakarta; sin embargo, debido a su popularidad, constituye ahora un proyecto Apache independiente, donde es soportado y mejorado por un grupo de voluntarios de la comunidad Java de código abierto. Apache Tomcat es muy estable y tiene todas las características de un contenedor de aplicaciones web comercial - sin embargo, se presenta bajo la licencia Open Source Apache License.

Figura 6: Apache Tomcat

Es un servidor HTTP y un contenedor de servlets. De forma predeterminada, se ejecuta en el puerto 8080.

Tomcat implementa los servlets java y las especificaciones de las páginas del servidor Java. Proporciona un entorno de servidor web Java para que se ejecute el código Java. Apache Tomcat incluye herramientas de configuración y gestión. También se puede configurar directamente editando el archivo de configuración XML.

2.2. Planificación

El equipo de Padel CAR se ha organizado con reuniones semanales todos los martes para plantear y diseñar y desarrollar la aplicación.

Para el desarrollo de la aplicación hemos trabajado sobre un repositorio remoto en GitHub, donde los integrantes del equipo han podido añadir, modificar y actualizar las partes de la aplicación para que esta funcione correctamente.

Padel Car hace uso de la metodología **Scrum** para llevar a cabo el conjunto de tareas de forma regular con el objetivo principal de trabajar de manera colaborativa, para fomentar el trabajo en equipo.

2.3. Descripción del trabajo realizado

Diseño del proyecto

Actualmente Spring se ha convertido en el estándar para el desarrollo de numerosos proyectos Java, por este motivo se utiliza para el desarrollo del presente proyecto.
Spring nos permite desarrollar aplicaciones de manera más rápida y eficaz. Tiene una amplia compatibilidad para la integración con otros frameworks y librerías de uso común para la creación de aplicaciones web, como Tiles, APIs en capa de persistencia como Hibernate y otros muchos recursos.

En nuestro caso, Spring se integra al proyecto como dependencia a través de Maven, por lo tanto, los módulos de Spring que se deseen agregar en el proyecto se añadirán en el pom.xml (Project Object Model) para que Maven lo integre. Las dependencias se añaden entre las etiquetas <dependencies></dependencies>.

Por ejemplo:

```
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-webmvc</artifactId>
<version>4.3.0.RELEASE</version>
</dependency>
```

Con el objetivo de simplificar el desarrollo de aplicaciones Spring posee la característica de Inyección de Dependencias (DI), que es una forma de asociar objetos de la aplicación. Para explicarlo de forma sencilla y rápida, en Spring, los objetos (bean) no son responsables de encontrar o crear el resto de los objetos que necesitan para llevar a cabo su trabajo, en su lugar, el contenedor del núcleo de Spring (fábricas de beans), les asigna referencias a los objetos con los que tienen que colaborar.

Spring ofrece tres mecanismos para configurar los beans y sus dependencias, que no son excluyentes y se pueden mezclar:

- Configuración explícita de XML
- Configuración explícita en Java
- Detección implícita y conexión automática de bean.

Para este proyecto se opta por la configuración en Java, con el uso de anotaciones, ya que es más potente, ofrece seguridad de tipos y permite refactorización.

Las clases anotadas con @Component (del paquete org.springframework.stereotype) identifica a una clase como clase de componente y sirve para indicar a Spring que debe crear un bean para la clase. Por defecto, el identificador del bean creado va a ser el nombre de la clase anotada con la primera letra en minúscula: @Component ("idDelBean”).

Por ejemplo, con la anotación

@ComponentScan: `@ComponentScan(basePackages = "padelcar")`

La conexión automática es una forma de permitir que Spring satisfaga automáticamente las dependencias de un bean buscando otros bean de la aplicación que coincidan con sus necesidades.

Para indicar que debe realizarse la conexión automática se utiliza la anotación @Autowired del paquete org.springframework.beans.factor.annotation.

Por ejemplo, en el proyecto se conecta el bean de IClienteService en ClienteController.java:
@Autowired IClienteService clienteService

Esquema de la aplicación en base a la estructura del framework Spring 4:

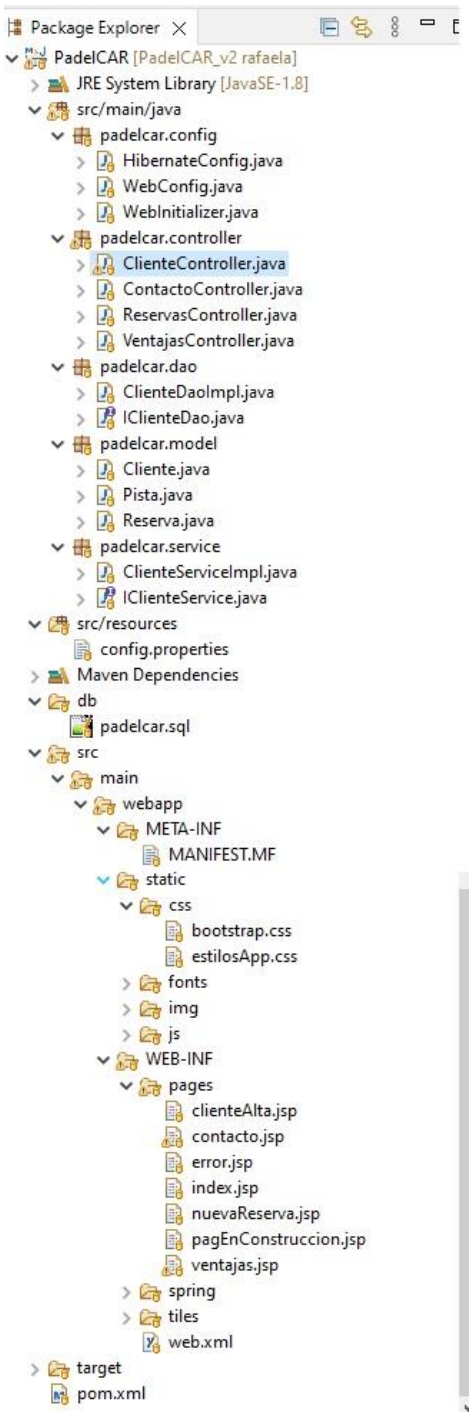


Figura 7: Estructura de paquetes

Arquitectura del sistema

Para el desarrollo de la aplicación se ha usado el entorno Eclipse Java EE IDE for Web Developers, la aplicación web se ha desarrollado en Java EE 8. Las dependencias de librerías se resuelven con Apache Maven.
Se utiliza Spring MVC 4 para implementar el modelo MVC y la inyección de dependencias. Para la base de datos se utiliza MySQL y para el acceso a la base de datos se usará JPA+Hibernate 4.

En la parte de la vista, se hará uso de las tecnologías Apache Tiles, para la resolución de vistas; JSP, para las vistas; CSS y BootStrap, para el estilo; y Javascript para los scripts del lado del cliente.

El servidor de aplicación es Apache Tomcat 9.
En la siguiente figura se muestran las diferentes tecnologías que se han utilizado de forma esquemática:

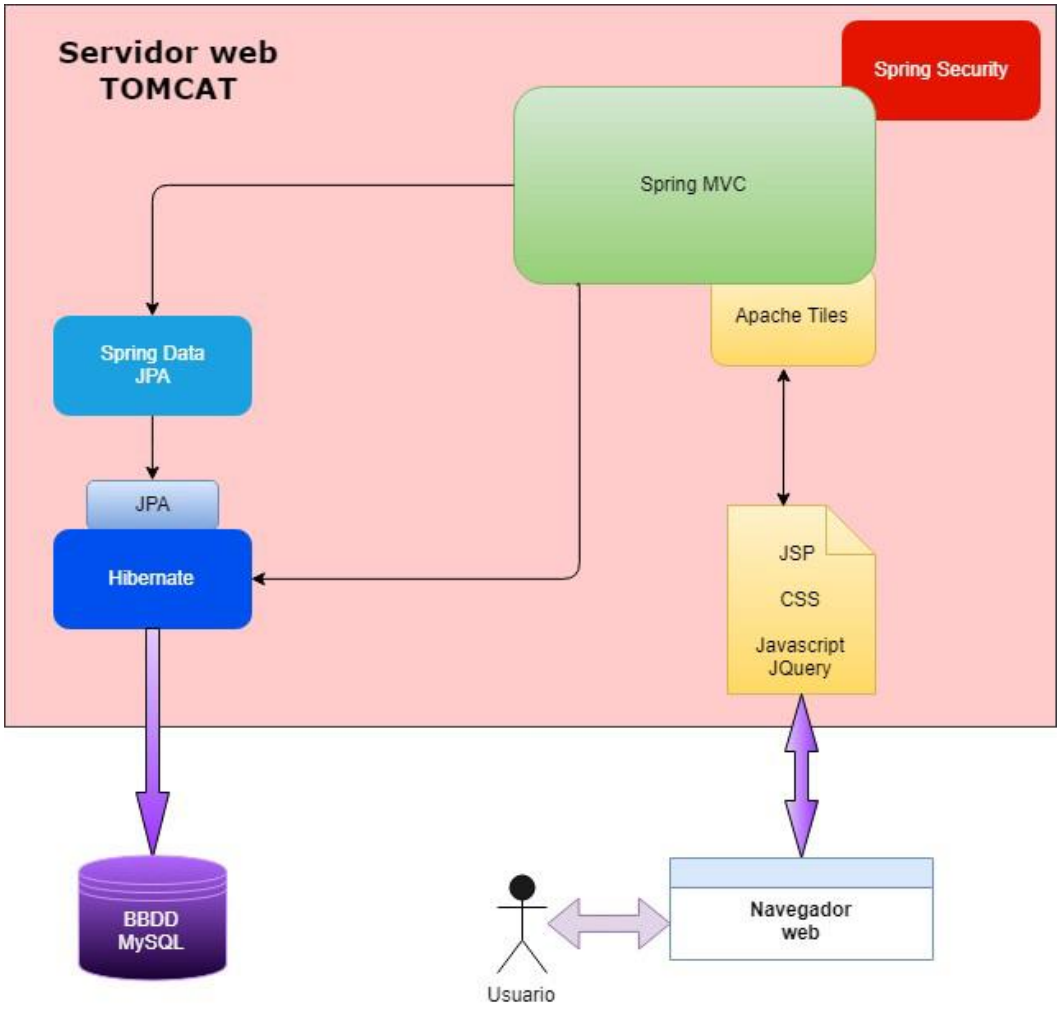


Figura 8: Arquitectura de la aplicación

Modelo de datos

El modelo de datos se ha implementado sobre una base de datos MySQL Server llamada padelcardb. El diseño del modelo de datos con las tablas que componen la aplicación, los atributos de las tablas y las relaciones establecidas entre éstas, son descritas en el siguiente diagrama:

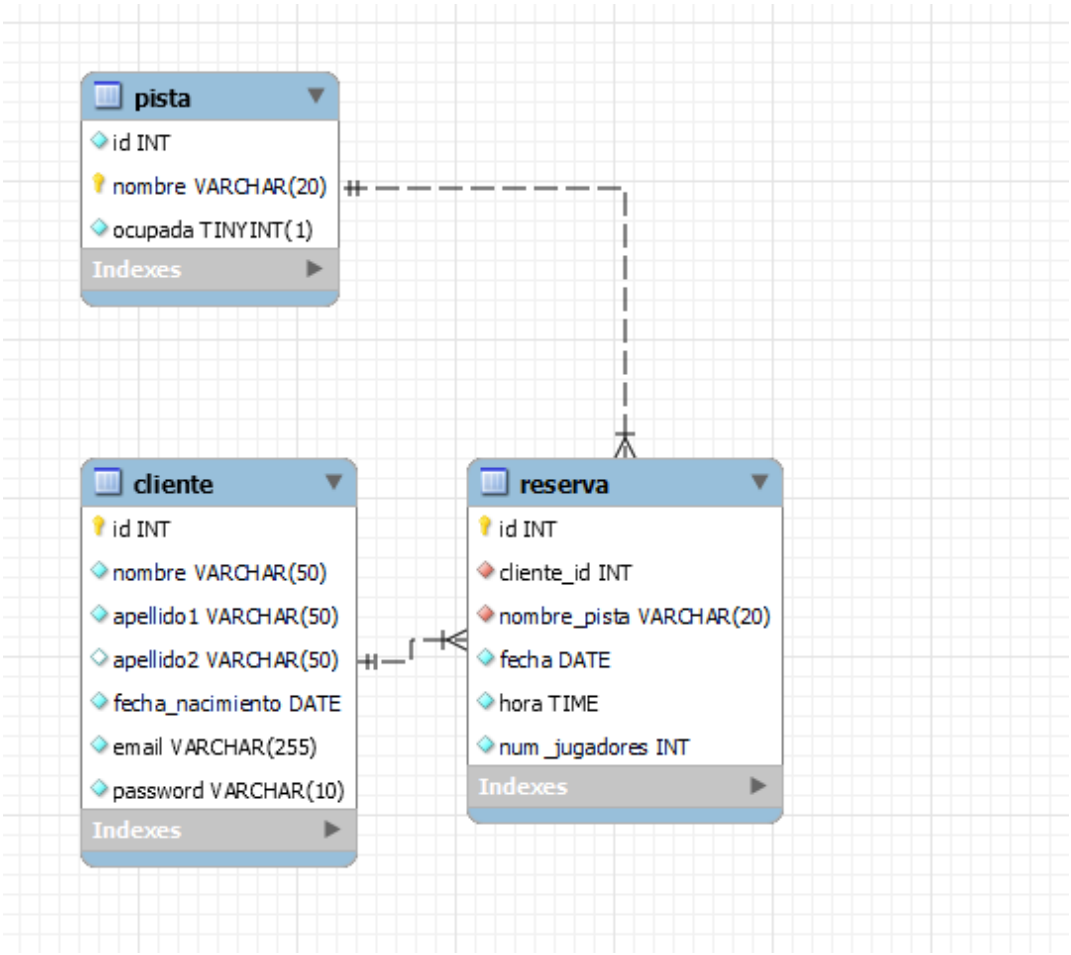


Figura 9: Modelo de datos

A continuación, se muestra una breve descripción de las tablas utilizadas:

Tabla Cliente:
En esta tabla se van a almacenar los datos referentes a los clientes
Existentes:

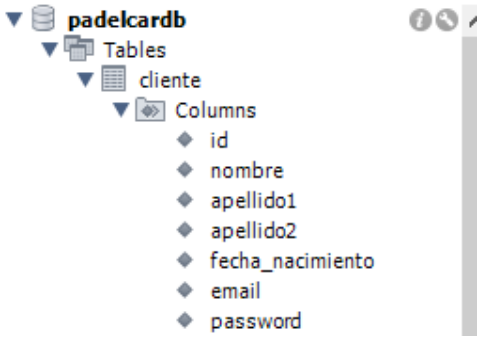


Figura 20: Estructura base de datos MySQL. Tabla cliente

Siguiente Tabla, se corresponde con un catalogo de las pistas existentes que se pueden seleccionar:
PISTAS:

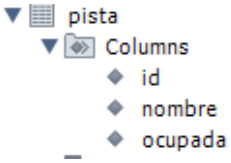


Figura 31: Estructura base de datos MySQL. Tabla pista

La tabla Reserva:
Se almacenan las reservas que hacen los clientes con las informaciones para dicha reserva

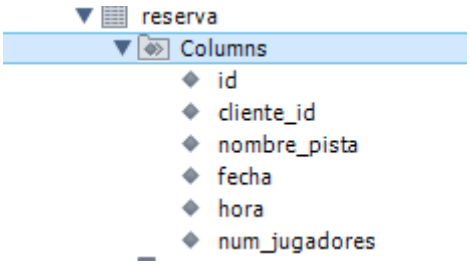


Figura 42: Estructura base de datos MySQL. Tabla reserva

Modelo de capas

El diseño de la aplicación va a seguir el siguiente modelo de capas:

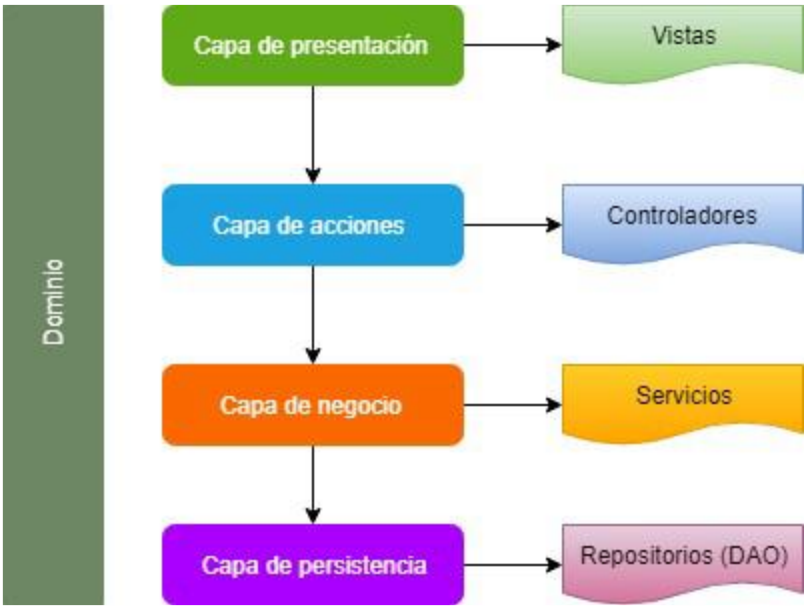
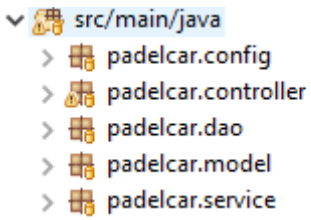


Figura 53: Modelo de capas

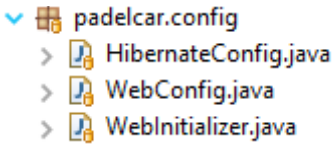
Estructura de paquetes

Las clases Java van a ir dentro de la carpeta src/main/java en los siguientes paquetes:

- Configuración:padelcar.config
- Capa de Dominio (Entidades): padelcar.model
- Capa de Persistencia (Repositorios): padelcar.dao
- Capa de Acciones (Controladores):padelcar.controller
- Capa de Negocio (Servicios): padelcar.service



CONFIGURACIÓN: en este paquete van a ir las diferentes clases de configuración, definidas mediante la anotación @Configuration:



HibernateConfig.java: Clase para configurar el contexto raíz, en concreto los componentes del nivel de datos. Se define la fábrica de administradores de entidades, las propiedades de Hibernate y la base de datos de la aplicación. Para conectar con la base de datos de la aplicación se añade la anotación @PropertySource(value =
{ "classpath:config.properties"}).

En el archivo config.properties se indica, entre otras cosas, el nombre de la base de datos y sus usuario y contraseña de acceso.

```
jdbc.driverClassName = com.mysql.jdbc.Driver
jdbc.url = jdbc:mysql://localhost:3306/padelcardb
jdbc.username = root
jdbc.password = root
hibernate.dialect = org.hibernate.dialect.MySQLDialect
hibernate.show_sql = true
hibernate.format_sql = false
```

WebConfig.java: Clase para configurar el contexto servlet.
WebInitializer.java: Clase para configurar el DispatcherServlet

***CAPA DE DOMINIO*:** En la capa de dominio se encuentran las entidades del sistema. A cada tabla del modelo de datos se le asigna una entidad.

Cuando se configura la fábrica de entidades, en la clase HibernateConfig se indica este paquete. Las clases de este paquete deben tener las siguientes características:

La clase va a llevar la anotación @Entity
El nombre de la clase coincidirá con el de la tabla, la primera letra en mayúscula y el resto en minúscula. La clase llevará la siguiente anotación con el nombre de la table correspondiente en cada caso: @Table(name="cliente")

Por cada propiedad de la clase se va a tener un get/set asociado.

CAPA DE PERSISTENCIA: En esta capa se encuentran las interfaces y clases usadas para el acceso a la base de datos, es decir lo repositorios, y se anotarán mediante @Repository.
La convención seguida para nombrar cada repositorio es utilizar el nombre de la entidad con el sufijo “Dao”, por ejemplo, “ClienteDao”.

Las interfaces siempre irán nombradas con una I de prefijo y las clases que implementan una interfaz llevarán el sufijo Impl, por ejemplo “ClienteDaoImpl”, para el caso de las clases e “IClienteDao” para el caso de las interfaces.

CAPA DE ACCIONES: En el paquete padelcar.controller van a ir las clases que realicen la función de controlador, es decir, las clases encargadas de interceptar las solicitudes HTTP del cliente y de traducir dichas solicitudes en operaciones específicas de negocio a ser realizadas.

Se sigue la siguiente convención:

- Utilizan el sufijo “Controller” en su nombre.
- Llevan la anotación @Controller

- En la medida de lo posible, no realizarán funciones de procesamiento, sino que se realizará la lógica del negocio en los objetos del servicio.
-

CAPA DE NEGOCIO: es una capa intermedia que está encargada de coger los datos de la capa de persistencia (los repositorios) y proporcionar la funcionalidad necesaria a la capa de acción (los controladores). Para cada entidad va a haber una clase de negocio que va a:

- Utilizar el sufijo "Service" en su nombre, por ejemplo"ClienteServiceImpl", y llevará la anotación @Service
- Implementar una interfaz cuyo nombre será el mismo que el de la clase y con el prefijo "I", por ejemplo "IClienteService".

Capa de presentación

La capa de presentación la constituyen las vistas que se van a integrar con el resto de capas.

A modo de ejemplo se añade una vista del proyecto

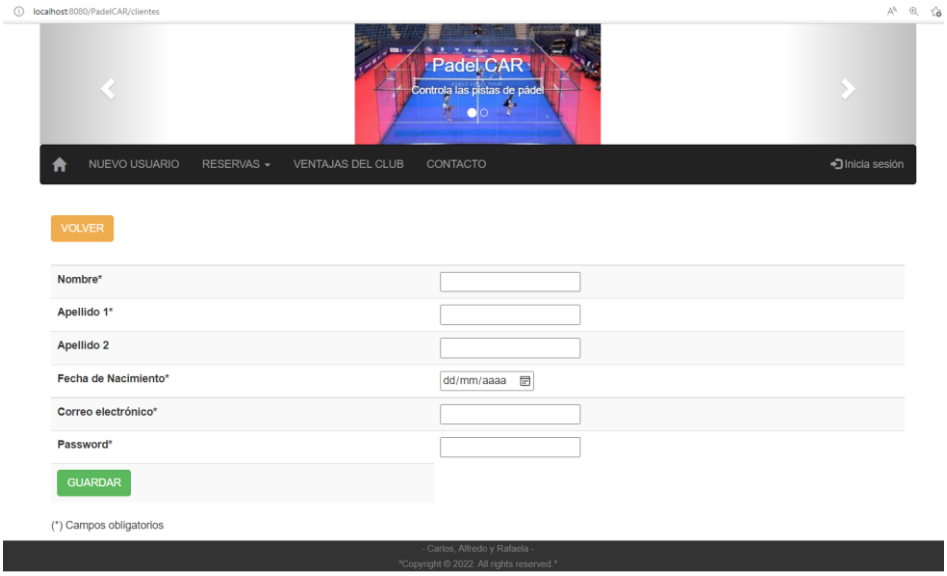


Figura 64: Pantalla Nuevo Usuario

Los ficheros correspondientes a la capa de presentación van a ir dentro de la carpeta src/main/webapp.

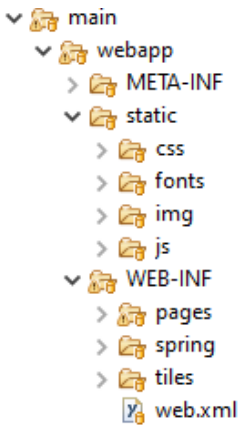


Figura 75: Estructura paquete main

Páginas JSP: En la aplicación se va a utilizar JSP para la tecnología de la vista. Para el diseño se va a hacer uso de Apache Tiles como motor de diseño para reducir la duplicación de elementos comunes. En la carpeta WEB-INF está ubicado:

En WEB-INF/tiles se encuentra el fichero tiles.xml que va a contener las especificaciones de mosaico.
En WEB-INF/tiles/layouts está la plantilla que va a usar el mosaico base.
En WEB-INF/tiles/template están las plantillas de cada apartado (encabezado, menú y pie de la web)
En WEB-INF/page se encuentran el resto de los ficheros .jsp que se corresponden con la sección central de las diferentes páginas de la aplicación.
Otros elementos de la capa de presentación: Dentro de la carpeta
webapp/static se encuentran el resto de los elementos como son los ficheros
css (static/css) para aplicar un estilo propio a la aplicación, los ficheros
javascript (static/js) y las imágenes ubicadas en la carpeta static/img.

2.4. Resultados y validación

En cuanto a la ejecución del programa, la aplicación contiene un encabezado que muestra fotografías a modo de carrusel, un menú con las opciones de navegación para el usuario y un pie con información.

El punto de partida es una pantalla de bienvenida con el nombre de la aplicación y una descripción.

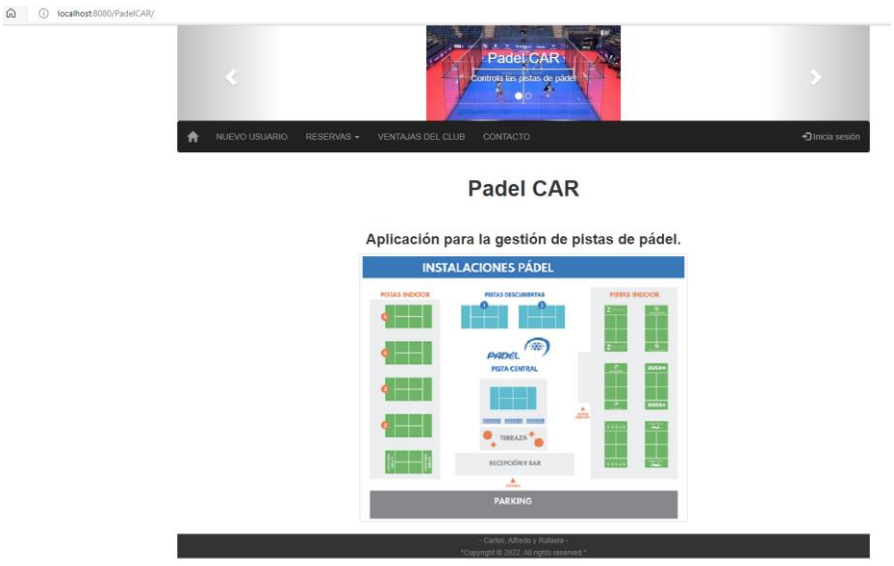


Figura 86: Pantalla principal

En la pantalla de nuevo usuario, si no estas registrado y quieres hacer una nueva reserva debes crear un nuevo registro.

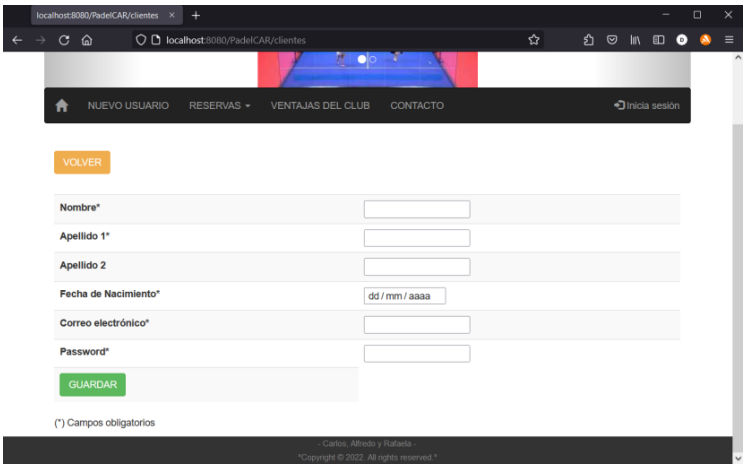


Figura 97: Pantalla nuevo usuario

Siendo obligatorio rellenar los campos de nombre, apellido 1, fecha de nacimiento, correo electrónico y la contraseña, solo tienes que darle a guardar para incluirte en nuestra base de datos.

Si alguno de los campos obligatorios no se rellena muestra un mensaje debajo del mismo indicándolo.

Figura 107: Validaciones

****Boton guardar****

*****RESERVAS****
En ventajas del club tienes una lista de ventajas que puedes disfrutar si usas la aplicación.

Figura 118: Pantalla ventajas del club

Contacto tiene mas información

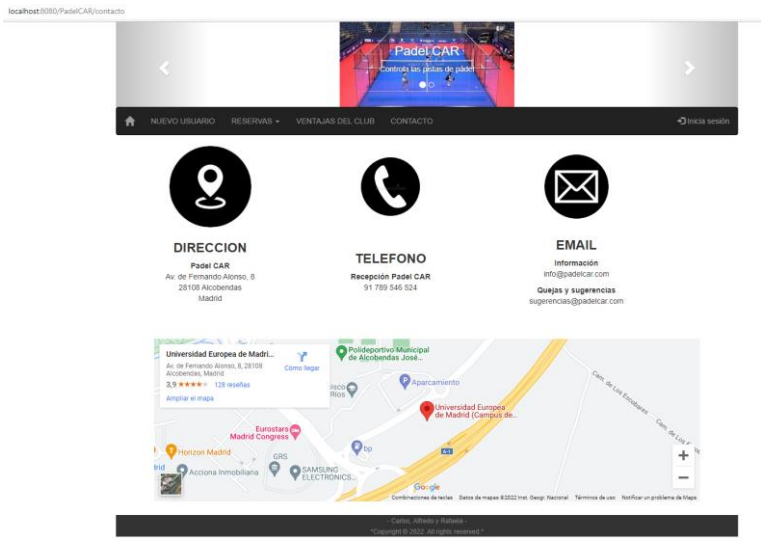


Figura 129: Pantalla Contacto

Información desde la dirección hasta la web, numero de teléfono, email y dirección.

Inicia sesión

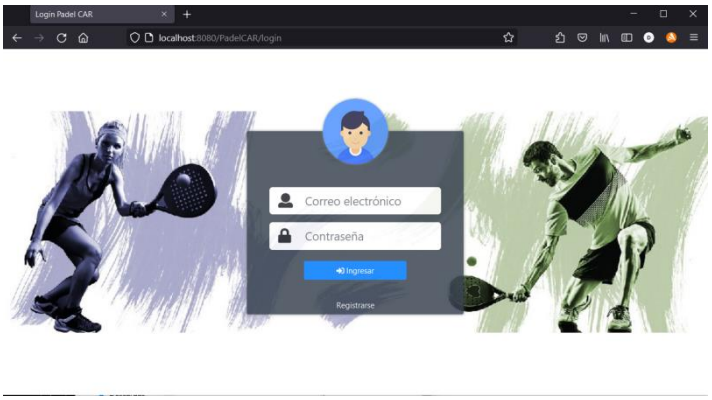


Figura 20: Pantalla inicio de sesión

El botón ingresar te lleva a reservas, y el botón registrarse te manda a nuevo usuario.

En cuanto a los problemas de la aplicación necesita algo mas de facilidad de uso para que no se haga pesada.

Las simulaciones se han echo en local utilizando el worbench para comprobar si el guardado se hacia correctamente, y el propio front de la aplicación para comprobar que la base de datos manda la información.
Es un programa sencillo que consta de dos partes, la parte del usuario, que te autentifica para poder generar un id único para cada uno, y la parte de reservas, que guarda, modifica y borra a trabes de ese id.

3. CONCLUSIONES

Despues de un largo trabajo, hemos logrado cumplir con el objetivo de este proyecto, el cual era llevar a cabo la creación de una aplicación web utilizando el Framework Spring y Java. Dicha aplicación serviria para gestionar la reservas que realiza un usuario en un centro deportivo de pádel.

El verdadero reto fue utilizar una tecnologia que no habiamos aprendido durante el curso. Pero, ya que sabiamos que era algo que teniamos que aprender, ya que, en el mundo laboral es lo que se utiliza, esta era nuestra oportunidad para conocerla y aprenderla, aunque haya sido de manera muy basica, ya que conforme ibamos investigando, nos dimos que cuenta que era una tecnologia muy extensa y compleja, a pesar de hacer mucho mas facil la programación.

Desde la primera idea, el diseño y el desarrollo, hemos trabajado de manera grupal, exponiendo de manera personal cada uno su idea, para despues fusionarlas en una. Con reuniones cada semana para tratar los problemas o errores encontrados y para plantear los nuevos objetivos durante los siguientes días, hemos logrado así, un trabajo constante.

Estamos seguros que gracias a la unión entre compañeros que supimos cultivar durante estos casi dos años, nos ha ayudado a saber superar cada uno de los aspectos mas dificiles que se nos presentaban durante el diseño del programa, dandonos apoyo mutuo en todo momento.

Como resultado final, hemos desarrollado una aplicación funcional, pero el verdadero resultado es la capacidad adquirida para trabajar tanto en Backend como en Frontend, gracias al uso de las distintas tecnologias como son CSS, JavaScript, html, Java y por supuesto, nuestro principal aporte, Spring.

3.1. Aportaciones

La parte del proyecto que puede considerarse como novedosa es la programación con spring.

Spring es un framework de código abierto que sirve como controlador de inversión de control que se encarga de instanciar inicializar y conectar los objetos de una aplicación.

Utilizar este framework como herramienta para programar tiene grandes ventajas, como su modulo de spring-core permite reducir las lineas de código, o estar basado en la programación orientada a interfaces que hace que se pueda cambiar la implementación de una clase en tiempo de ejecución.

Todo esto hace de spring una herramienta que facilita y clarifica el código de una aplicación.

3.2. Trabajo futuro

Al final del desarrollo y después de una larga investigación, nos dimos cuenta que se puede diseñar una aplicación web con otro tipo de tecnologias como por ejemplo Django, Angular o Gradle.

Y siguiendo la tecnologia Spring, el siguiente paso sería utilizar Spring Boot, el cuál ya implenta un contenedor Tomcat, dependencias y muchas otras funcionalidades que con Spring Framework tuvimos que implementarlas manualmente, sindo esto mas complicado de realizar.

Y si hablamos de nuestro proyecto, sabemos que se pueden ampliar y mejorar, en los siguientes aspectos:

- Cancelación de reservas
- Modificación del perfil del usuario
- Creación de equipos y partidos
- Añadir participantes a un partido
- Diseñar formas de pago real.
- Promociones e incentivos a los usuarios.

4. BIBLIOGRAFÍA Y WEBGRAFÍA

4.1.1. Biliografía

- Patrones de Diseño en Java. Eni Ediciones. Laurent Debrauwer. Fecha de consulta: abril 10, 2022
- Documentación Spring Core + MVC. Edicion Libre. Sin autor. Fecha de consulta: abril 15, 2022

4.1.2. Webgrafía

- Spring Framework
  https://www.uv.es/grimo/teaching/SpringMVCv5PasoAPaso/index.html.
  Francisco Grimaldo Moreno
  Fecha de consulta: abril 2, 2022
- Hibernate.
  Capítulo 6. Mapeos de colección (jboss.org)
  Red Hat, Inc.
  Fecha de consulta: abril 5, 2022
- stackoverflow
  java - Spring MVC jsp c:if tag doesn't work - Stack Overflow
  Stack Exchange, Inc.
  Fecha de consulta: mayo 10, 2022
- programador clic
  Spring MVC Hibernate MySQL Integration (integración) Tutorial de ejemplo de CRUD [extracto] - programador clic (programmerclick.com)
  Programadorclick.com
  Fecha de consulta: mayo 10, 2022
- https://www.adictosaltrabajo.com/2008/02/15/spring-hibernate-anotaciones/
- https://www.uv.es/grimo/teaching/SpringMVCv4PasoAPaso/index.html

Spring MVC Hibernate MySQL Integration (Integración) Tutorial de ejemplo de CRUD:

- https://programmerclick.com/article/21431004605/
- https://www.journaldev.com/3531/spring-mvc-hibernate-mysql-integration-crud-example-tutorial
- https://programacion.net/articulo/integracion_de_jsf-_spring_e_hibernate_para_crear_una_aplicacion_web_del_mundo_real_307/3

Estilos de letras HTML

- https://www.w3schools.com/bootstrap4/bootstrap_typography.asp
- Unipython
  Qué es el Framework Spring y las ventajas de utilizarlo - ▷ Cursos de Programación de 0 a Experto © Garantizados (unipython.com)