UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Diseño e implementación de una aplicación web de apuestas deportivas con bitcons

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Moncho Ferrer, David

Tutor: Posadas Yagüe, Juan Luís

2016-2017

Resumen

El trabajo que se propone consiste en el diseño e implementación de una plataforma web donde el usuario pueda realizar diferentes tipos de apuestas con una moneda digital como el bitcoin.

Los usuarios podrán elegir entre varios deportes y diferentes tipos de apuestas y solo se aceptarán monedas virtuales, esta es la principal diferencia entre las casas de apuestas mas conocidas del mercado.

La web mostrará los eventos deportivos disponibles donde los usuarios podrán apostar, estos eventos se irán mostrando automáticamente, solo se mostrarán eventos que no hayan empezado y dejarán de mostrarse una vez no sea posible apostar en dichos eventos.

Los usuarios podrán elegir entre apuestas simples o combinadas. En las apuestas simples el usuario elegirá un evento al cual quiera apostar y introducirá la cantidad que saldrá de su cartera virtual. En las apuestas combinadas tiene que seleccionar mas de un evento y estas cuotas se combinaran para crear una cuota final en la que el usuario deberá introducir la cantidad a apostar de la misma forma que en las apuestas simples. Una vez terminados los partidos la aplicación pasara a pagar a los usuarios ganadores.

Palabras clave: web, apuestas, bitcoin

Abstract

The work that is proposed is the design and implementation of a web platform where the user can make different types of images with a digital coin such as bitcoin.

Users choose from various sports and different types of bets and solos are accepted virtual currencies, is the main difference between the bookmakers and known market.

The website will show the available sports events in which the users are shown, these events are automatically displayed, only the events that have not been started are displayed and stopped showing once there is no possibility to bet on these events.

Users can choose between single or combined bets. In simple bets the user will choose an event to which he wants to bet and enter the amount that will come out of his virtual wallet. In combined bets you have to select more than one event and these odds are combined to create a final quota in which the user must enter the amount of the same bet in the same way as in simple bets. Once finished the application parties will pay the winning users.

Keywords: web, bets, bitcoin

Tabla de contenidos

1 TABLA DE CONTENIDO

Índice de gráficos

1 INTRODUCCIÓN

El presente documento es la memoria del trabajo final de grado de ingeniería informática. En el vamos a desarrollar todas las etapas de la vida del software, objetivos, análisis, diseño, implementación, pruebas, validación y mantenimiento.

1.1 MOTIVACIÓN

El uso de Internet cada día está más extendido entre la población, cada día son más las personas que eligen este medio para realizar cosas cotidianas, por esto cada día cobra más importancia que los negocios que hace unos años solo se desarrollaban de la forma tradicional necesitan tener visibilidad en la red y ofrecer sus servicios a través de esta.

A su vez, aunque más recientemente es clara la tendencia en el uso de las criptomonedas, y en concreto del bitcoin que desde hace un tiempo a esta parte está ganando muchos usuarios.

Otra realidad es que el mercado de las apuestas online cada vez es mayor y descubrimos nada nueva al decir que estas plataformas tienen muchos beneficios, aunque también es verdad que entrar en el negocio de las apuestas es muy difícil debido a las grandes exigencias de las instituciones y la dificultad de conseguir licencias.

Al mismo tiempo son pocas las que ofrecen entre sus divisas las criptomonedas y es este punto de innovación o diferenciación del resto del mercado el que sumado a las características anteriormente descritas nos ha llevado a desarrollar esta aplicación web.

1.2 OBJETIVOS

El objetivo de este proyecto es el desarrollo de una plataforma de apuestas on-line con bitcoins que permita a los usuarios de forma intuitiva realizar sus apuestas anónimamente.

Los usuarios tienen que poder realizar sus apuestas desde cualquier dispositivo ya se ordenador, Tablet o teléfono móvil en unos pocos clics.

En la web se clasificarán los eventos deportivos por deportes y seguidamente por competiciones, tendremos dos tipos de apuestas diferentes las apuestas simples y apuestas combinadas, todas las apuestas realizadas en la plataforma serán antes del partido.

La plataforma tiene que estar preparada para para ser actualizada con facilidad ya que se irán modificando constantemente los eventos deportivos.

2 ESTADO DEL ARTE

2.1 CRIPTOMONEDAS

La criptomoneda es una nueva forma de ver el dinero, también es conocida como moneda virtual o dinero electrónico.

La criptomoneda incorpora los principios de la criptografía para ofrecer una economía segura, anónima y descentralizada.

La primera criptomoneda que empezó a operar fue el Bitcoin en el año 2009 aproximadamente. Desde entonces han aparecido muchas criptomonedas alternativas, cada cual con sus diferenciaciones.

La que más importancia tiene en este momento es el Bitcoin y es en la que nos hemos centrado para el desarrollo de este proyecto.

2.1.1 Bitcoin

Esta criptomoneda nació en 2009 de la mano de Satoshi Nakamoto, y lo que al principio parecía una moda pasajera con el tiempo se ha convertido en una realidad.

Desde sus inicios Bitcoin no ha parado de crecer hasta llegar a valores insospechados. A principios de 2011 consigue ponerse a la par con el dólar americano, un año más tarde se consolida como moneda de intercambio virtual y en 2013 el valor de todos los bitcoins del planeta asciende a 1 billón de dólares, a fecha de hoy, julio de 2017 el valor de un bitcoin es de 2600 dólares.

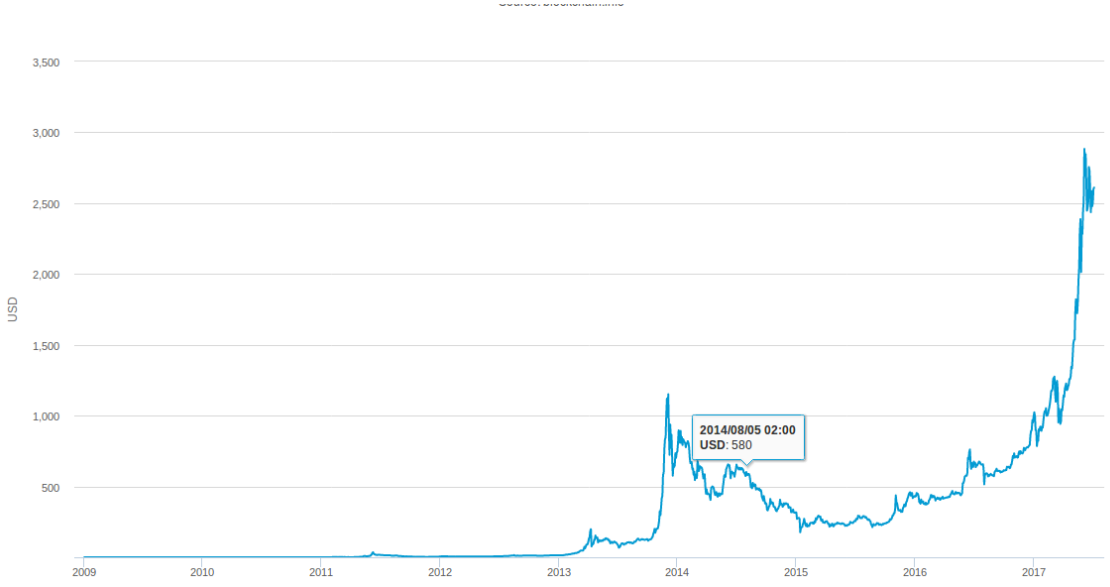


Figura 1: gráfico histórico del valor del bitcoin

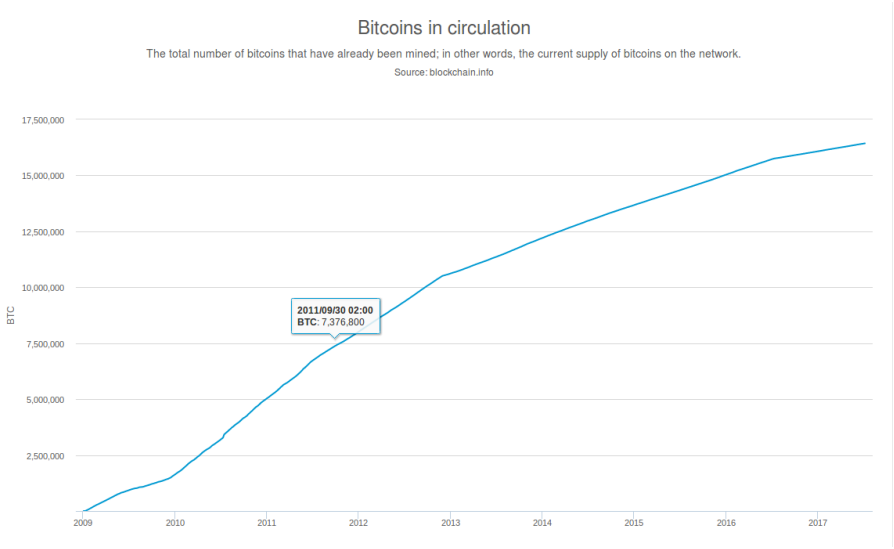


Figura 2: total de bitcoins en circulación

Toda esta evolución es debido a las características que la diferencian del dinero fiduciario, a continuación, vamos a exponer algunas de estas características.

- Descentralizado: el Bitcoin no está controlado por ningún estado, banco o institución financiera.
- Anonimato: permite hacer transacciones anónimas.
- Internacionalidad: se puede utilizar en cualquier parte del mundo sin necesidad de hacer cambio de divisa.
- Seguridad: tus Bitcoins te perteneces solo a ti y nadie te los puede intervenir.
- Rapidez: las transacciones son mucho más rápidas que a través de las entidades financieras.
- Uso: el uso de esta moneda es voluntario.
- Inflación: el bitcoin tiene una Inflación controlada a diferencia del dinero fiduciario ya que en este ultimo las instituciones pueden generar más dinero de manera artificial y en el Bitcoin se conoce de antemano la cantidad que existen en total.

Está claro que cada vez son más las personas que usan este tipo de monedas y la realidad que la oferta existente de sitios donde se pueda usar el Bitcoin es escasa, aunque está en alza.

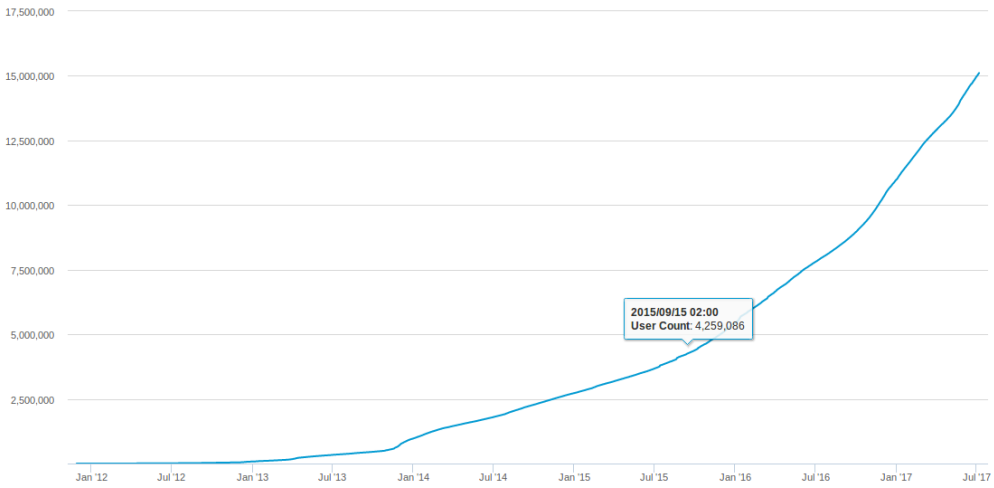


Figura 3: usuarios con carteras de bitcoins

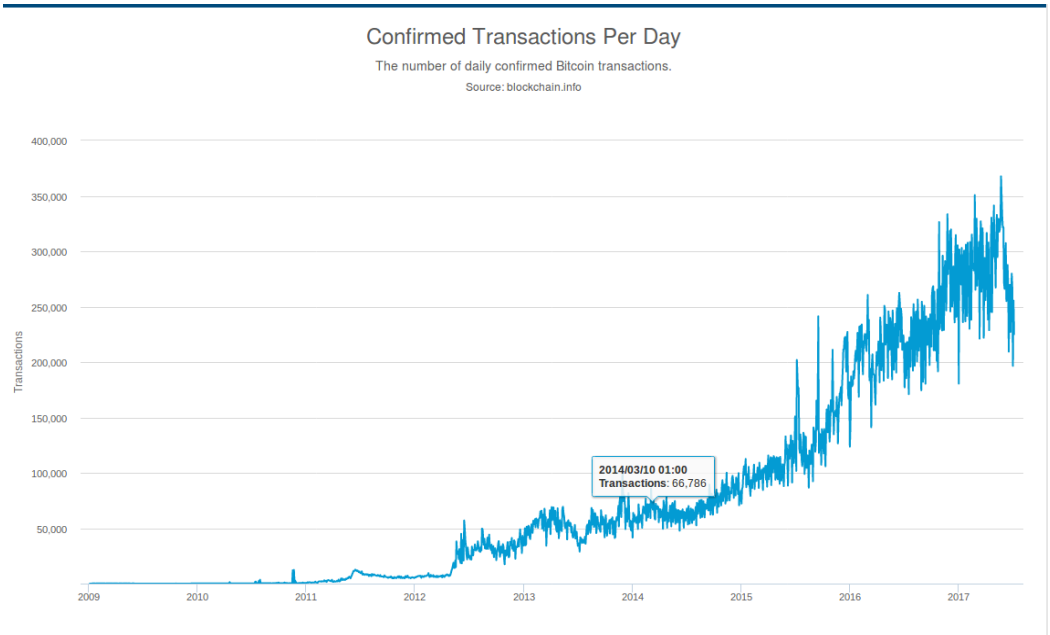


Figura 4: transacciones por día

Por estos dos motivos, crecimiento y las características unido a la escasa oferta de sitios donde usar el Bitcoin ha hecho que hayamos elegido el Bitcoin para nuestro proyecto, ya que tenemos el convencimiento que este tipo de monedas y entre ellas el Bitcoin es el futuro.

2.2 APLICACIONES RELACIONADAS

Existen algunas plataformas que ya nos permiten realizar nuestras apuestas en bitcoin, que desde nuestro punto de vista tienen puntos mejorables y por los cuales queremos diferenciarnos.

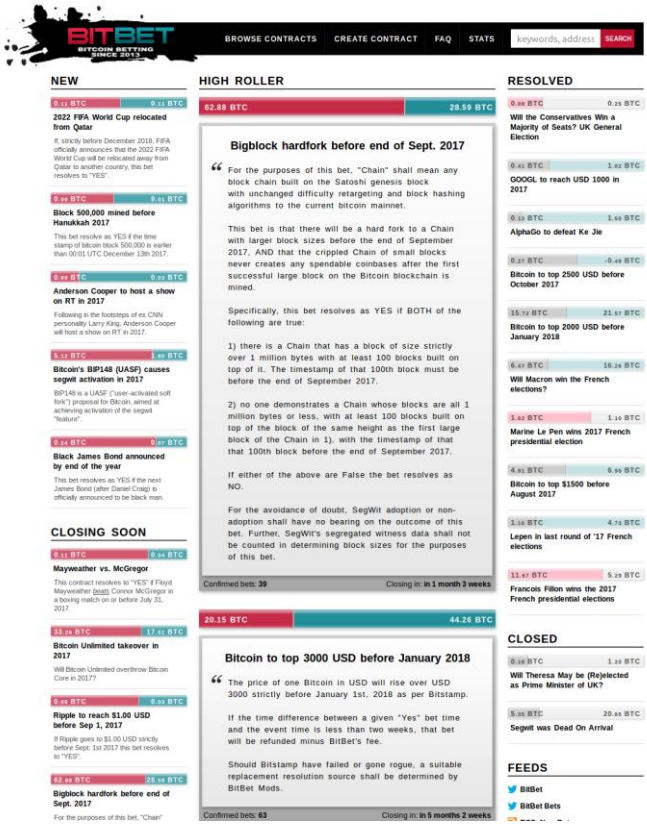


Figura 5: web bitbet.us

En este caso vemos una web del año 2013, lo que podemos observar es que el diseño se ha quedado un poco obsoleto y navegando en ella se aprecia la poca usabilidad de la misma.

Las apuestas no están bien estructuradas y no es fácil encontrar alguna apuesta en concreto.

Por otro lado, este portal deja realizar apuestas de forma anónima, sin ningún tipo de registro de usuario, cosa que nos parece muy interesante y es uno de sus puntos fuertes.

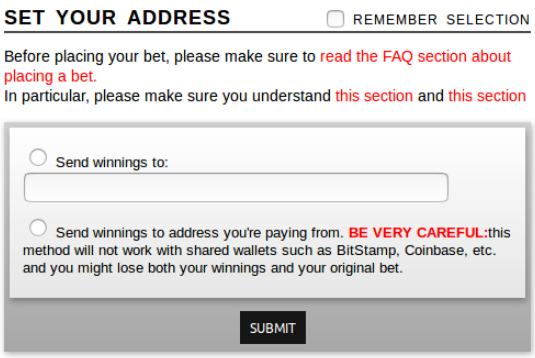


Figura 6: realizar apuesta en bitbet.us

La figura anterior nos muestra el momento de realizar la apuesta en la aplicación de bitbet.us. Para poder realizar la solo nos pide una cartera bitcoin donde enviar las ganancias no nos pide en ningún momento abrirnos una cuenta con ellos.

El resto de páginas que hemos visto estéticamente son mucho más actuales y mejor organizadas, pero presentan el problema de que no dejan realizar apuestas si no tienes una cuenta con ellos, cosa que nos parece un poco contradictoria con la filosofía de bitcoin que su gran fuerza radica en ese anonimato y es algo que queremos potenciar.

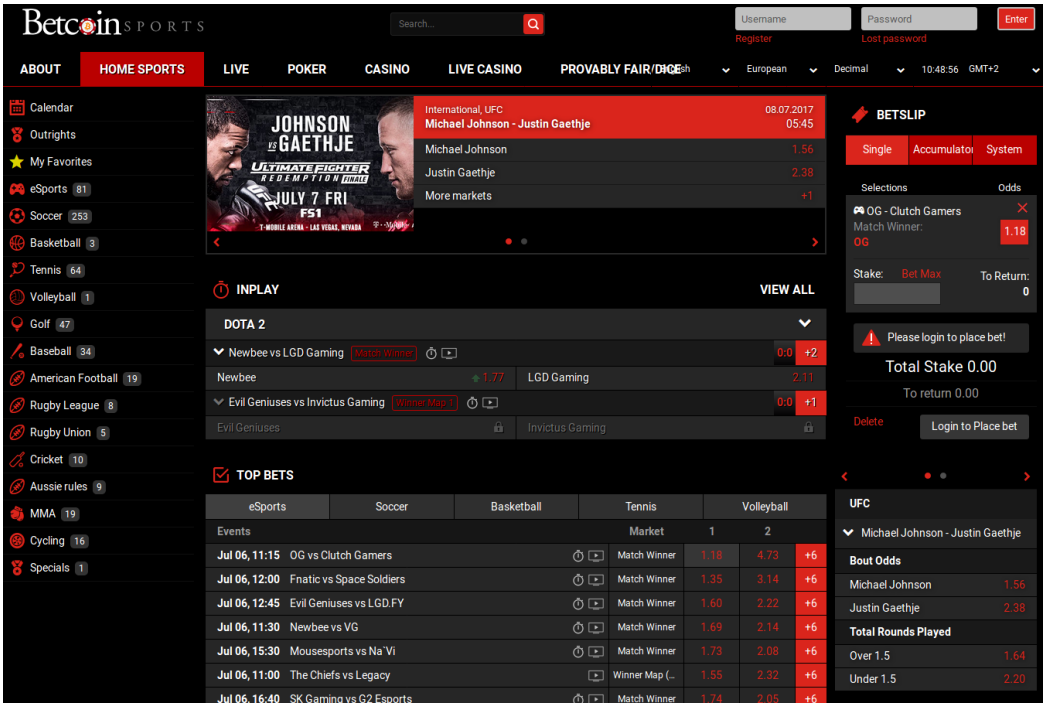


Figura 7: web betcoin.ag

En este caso podemos ver la web betcoin.ag la cual tiene un diseño más cuidado y una mejor organización, pero si nos damos cuenta nos vemos con la obligación de tener una cuenta con ellos para poder realizar las apuestas, lo cual desde nuestro punto de vista es un inconveniente.

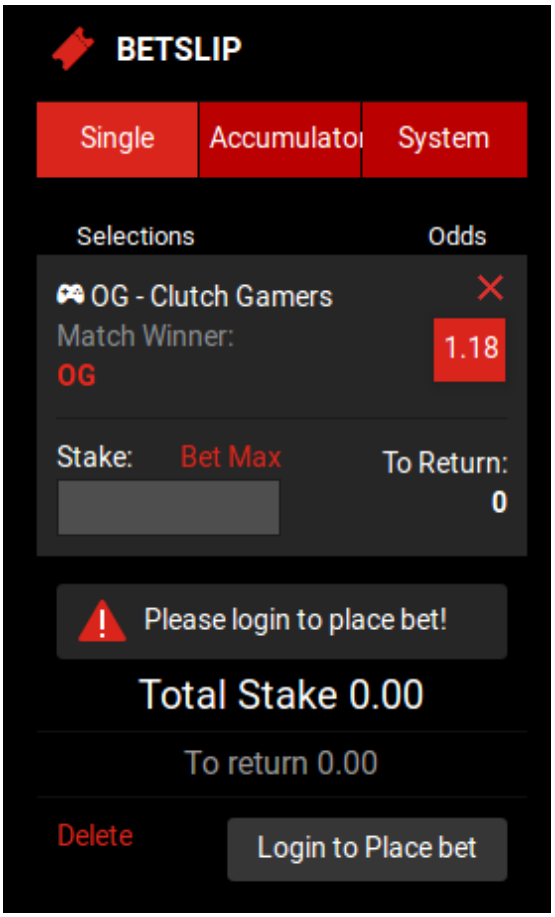


Figura 8: realización apuesta betcoin.ag

Como podemos observar nos pide iniciar sesión para realizar la apuesta.

3 ESPECIFICACIÓN DE REQUISITOS

3.1 INTRODUCCIÓN

En esta sección delimitaremos el alcance funcional de Enjoybets, se listarán los requisitos tanto funcionales como no funcionales.

3.1.1 Propósito

El propósito de la especificación de requisitos es definir una serie de requisitos que se puedan validar después del desarrollo de la aplicación, que estos requisitos sirvan como una base para el desarrollo y el diseño de la misma y atender las necesidades y requisitos por parte del usuario.

3.1.2 Alcance

Diseño y desarrollo de la plataforma Enjoybets.

Enjoybets será una aplicación web que permitirá realizar apuestas en bitcoins en eventos deportivos.

Para la definición del alcance dividiremos el proyecto en dos bloques, la parte del administrador y la parte del usuario.

La parte del administrador permitirá en un futuro añadir nuevos deportes competiciones y mercados desde un panel de administración entre otras cosas.

La parte del usuario este podrá realizar:

- desplazarse por los diferentes deportes y competiciones.
- seleccionar entre diversos mercados y ganadores.
- añadir sus ganadores o selecciones al cupón de apuestas.
- eliminar las selecciones del cupón de apuestas.
- realizar una apuesta.
- consultar sus apuestas.
- consultar su saldo disponible.

3.1.3 Definiciones acrónimos y abreviaturas

En este apartado intentaremos describir los conceptos clave del proyecto para hacerlos más entendibles.

3.1.3.1 De negocio

- Selección: es el ganador que selecciona el usuario de un evento deportivo.
- Realizar apuesta: acción de introducir bitcoins en la apuesta.
- Combinar apuesta: realizar una apuesta de más de un evento deportivo simultáneamente.
- Cupón: seleccione de eventos deportivos de los cuales aún no ha realizado una apuesta.
- Cancelar selección: eliminar del cupón un evento deportivo previamente seleccionado.
- Apuesta prepartido: son la apuesta que solo se pueden realizar antes del inicio del evento deportivo al que queremos apostará.
- Cartera bitcoin: es donde se almacenan las claves privadas para poder acceder a los bitcoins. Las carteras tienen una dirección única para poder transferir bitcoins.
- Bitcoin: Criptomoneda que se usara en las apuestas.

3.1.3.2 Del sistema

- Usuario: persona que navega o utiliza Enjoybets.
- Administrador del Sistema: persona encargada de ofrecer el soporte técnico y operativo a Enjoybets.
- Pruebas: proceso median te el cual se realizan actividades para verificar la óptima función del sistema.
- Refactorización: técnica para reestructurar código sin alterar su comportamiento.

3.1.3.3 De la tecnología

- MVC: modelo vista controlador, es el patrón de diseño que siguen la mayoría de las aplicaciones web actuales, se trata de separar la parte de la vista de la parte de la lógica y la persistencia de datos para de esta forma hacer más fácil de mantener y reutilizar las partes del código de la aplicación.
- PHP (acrónimo recursivo de PHP: Hypertext Preprocessor): lenguaje de código abierto adecuado para el desarrollo web y que puede ser incrustado en HTML.
- Apache: servidor HTTP de código abierto.
- GIT: software de control de versiones, sirve para facilitar el desarrollo y mantenimiento de proyectos.
- Laravel: framework de código abierto para desarrollar aplicaciones y servicios web con PHP.

3.1.4 Visión de conjunto

- Se desarrollará es una aplicación web donde los usuarios puedan hacer sus apuestas en bitcoins.
- Los usuarios podrán seleccionar entre todos los eventos deportivos que estén en la plataforma.
- Para poder apostar en la plataforma solo será necesario tener bitcoins en una cartera de bitinios.

3.2 DESCRIPCIÓN GENERAL

3.2.1 Perspectiva del producto

Se pretende desarrollar una plataforma web donde los usuarios de bitcoin tengan la posibilidad de realizar sus apuestas deportivas.

El usuario podrá realizar tanto apuestas simples como combinadas pre-partido y elegir entre la gran oferta de deportes.

3.2.2 Funciones del producto

- Selección de apuestas.
- Realizar apuestas.
- Realizar pago.
- Recibir pago.

3.2.3 Características del usuario

En esta aplicación solo habrá dos tipos de usuario:

Administrador del sistema: que se encargara de gestionar la plataforma web.

Usuario: al no tener la necesidad de registro todos los tipos de usuario serán invitados y podrán realizar las mismas acciones.

3.2.4 Restricciones

El usuario necesitara tener un dispositivo con el que se pueda conectar a Internet para así acceder a Enjoybets.

Otra de las restricciones es que todos los usuarios necesariamente necesitan tener bitcoins ya que en la plataforma es la única divisa aceptada.

3.2.5 Suposiciones y dependencias

Por una parte, necesitaremos un servidor web donde alojar la web para que pueda estar disponible 24/7.

Por otra parte, hay una dependencia con la API del sistema de transferencias de bitcoins.

3.3 REQUISITOS ESPECÍFICOS

Este apartado está dedicado a nombrar los requisitos funcionales que deberá cumplir el sistema.

3.3.1 Requisitos de la interfaz externa

3.3.1.1 Interfaz de usuario

Una de las características que queremos que nos diferencie de otras aplicaciones similares y ya existentes tiene que ser la interfaz del usuario.

Como ya hemos nombrado en alguno de los apartados anteriores buscamos que la experiencia del usuario sea muy buena y para esto nuestra aplicación tiene que ser intuitiva y fácil de usar y en un principio estará implementada en castellano, pero en el momento de la internacionalización también deberá estar en inglés.

Debido a que cada vez el uso de los dispositivos portátiles está ganando terreno al uso del ordenador personal también es de vital importancia que la web sea responsive para que el usuario tenga una experiencia muy agradable desde cualquier dispositivo.

3.3.1.2 Interfaz hardware

Para este desarrollo tendremos que tener en cuenta el servidor donde alojemos la propia web y los servidores de donde estén las APIS utilizadas para la gestión de pagos en bitcoins.

Nuestro servidor tiene que ser flexible a la demanda ya que tendremos momentos con muchas peticiones simultaneas debido a los grandes eventos deportivos combinados con otros momentos de menos demanda donde no se estén disputando tantos eventos.

3.3.1.3 Interfaz software

El usuario podrá acceder desde cualquier navegador y cualquier tipo de dispositivo, por lo que cualquier navegador estándar tendría que ser válido para ejecutar nuestra aplicación.

En la parte del servidor será un requisito imprescindible que este PHP 5 o superior y MySQL, así como GIT para el control de versiones.

3.3.2 Requerimientos funcionales

Número del requisito	1
Nombre del requisito	Seleccionar evento deportivo
Tipo	Requisito
Fuente del requisito	Aplicación web de apuestas en bitcoins
Prioridad del requisito	Alta/Esencial
Descripción del requisito	Se podrá seleccionar de entre la lista de eventos deportivos el evento deportivo al que el usuario quiera apostar

Número del requisito	2
Nombre del requisito	Combinar eventos deportivos
Tipo	Requisito
Fuente del requisito	Aplicación web de apuestas en bitcoins
Prioridad del requisito	Alta/Esencial
Descripción del requisito	Al hacer más de una selección se combinarán las cuotas para poder hacer apuestas de más de un evento deportivo

Número del requisito	3
Nombre del requisito	Realizar apuesta simple
Tipo	Requisito
Fuente del requisito	Aplicación web de apuestas en bitcoins
Prioridad del requisito	Alta/Esencial
Descripción del requisito	Cuando hay una única selección en el cupón poder realizar una apuesta introduciendo la cantidad deseada.

Número del requisito	4
Nombre del requisito	Realizar apuesta múltiple
Tipo	Requisito
Fuente del requisito	Aplicación web de apuestas en bitcoins
Prioridad del requisito	Alta/Esencial
Descripción del requisito	Cuando hay más de una selección en el cupón poder realizar una apuesta introduciendo la cantidad deseada.

Número del requisito	5
Nombre del requisito	Combinar dos selecciones del mismo evento deportivo
Tipo	Restricción
Fuente del requisito	Aplicación web de apuestas en bitcoins
Prioridad del requisito	Alta/Esencial
Descripción del requisito	El usuario no podrá poner en un mismo cupón dos selecciones del mismo evento para combinarlas. Ej.: no se puede seleccionar la victoria y al mismo tiempo la derrota de un mismo equipo

Número del requisito	6
Nombre del requisito	Realizar apuesta múltiple
Tipo	Restricción
Fuente del requisito	Aplicación web de apuestas en bitcoins
Prioridad del requisito	Alta/Esencial
Descripción del requisito	El usuario no podrá apostar más Bitcoins de los que tiene.

3.3.3 Requisitos de Rendimiento

El rendimiento de esta plataforma tiene que ser alto ya que para las apuestas al ser las cuotas cambiantes dependiendo la gente que entre no podemos permitirnos retrasos entre las acciones y las ejecuciones de las ordenes.

3.3.4 Requisitos de la base de datos lógica

La base de datos tiene que ser relacional, este tipo de base de datos nos garantiza evitar la duplicidad de registros y con la integridad referencial nos permite el borrado en cascada de datos relacionados entre si.

3.3.5 Limitaciones de diseño

El diseño tiene que ser responsive para que en todo tipo de dispositivo el usuario se encuentre cómodo y le sea sencillo e intuitivo realizar sus apuestas y así maximizar la experiencia de usuario.

3.3.6 Atributos del sistema software

- Seguridad: todas las transacciones tienen que ser seguras ya que los usuarios nos confían sus bitcoins, no puede perderse ninguna transacción.
- Disponibilidad: el sistema debe estar operativo 24/7/365.
- Usabilidad: la plataforma tiene que tener una interfaz sencilla y que sea intuitiva y fácil de usar.
- Mantenibilidad: el sistema debe ser fácil de mantener por otros equipos de programadores que tengan nociones del patrón MVC.
- Ampliable (Software): el sistema debe permitir que en el futuro se puedan añadir nuevas funcionalidades, idiomas y criptomonedas entre otras cosas.
- Cumplir estándares W3C: el sistema debe cumplir los estándares HTML 5.0 y CSS 3.0 (o superiores).

4 ANÁLISIS

4.1 DIAGRAMA DE CLASES/ENTIDADES

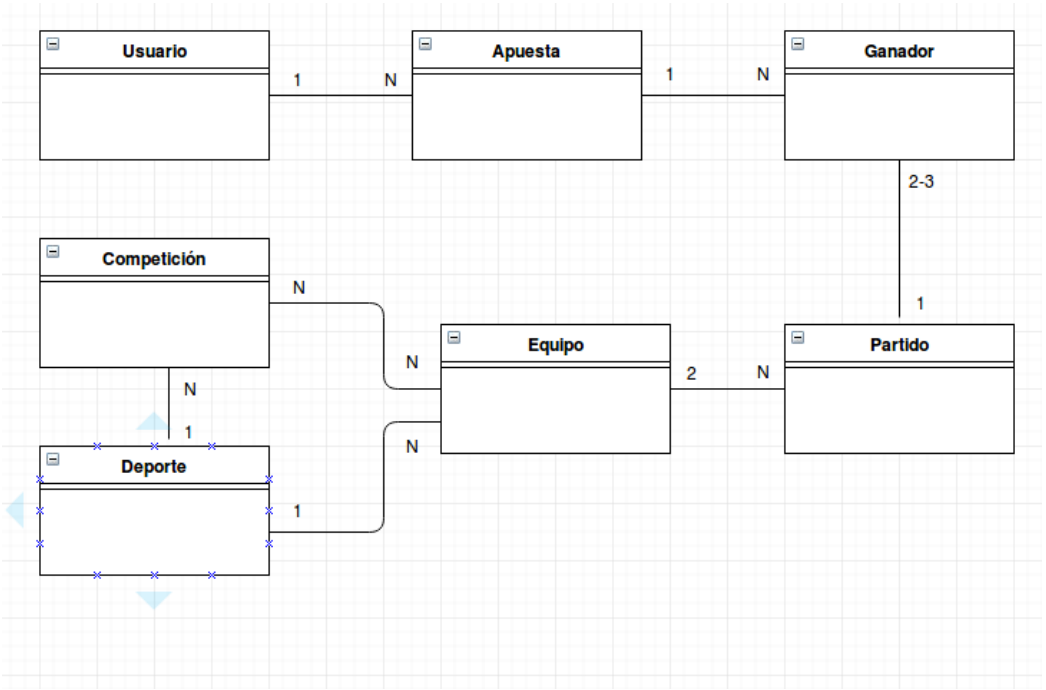


Figura 9: diagrama de clases

La clase principal es el usuario, vemos que este usuario puede realizar varias apuestas.

Las apuestas pueden tener varios ganadores, esto representa las apuestas de múltiple selección.

El que los partidos tengan dos o tres ganadores representa los dos tipos de partidos que hay, de dos opciones o de tres como en el caso del fútbol que se puede empatar.

Los partidos solo pueden tener dos equipos que en este caso serán los dos rivales.

En cuanto a los deportes competiciones y equipos las relaciones quedan más claras un deporte puede tener varias competiciones y un equipo puede estar en más de una competición.

4.2 CASOS DE USO

Vamos a hacer una representación de casos de uso por actores principales, en nuestro caso al no distinguirse del usuario registrado al no registrado solo tendremos un actor principal.

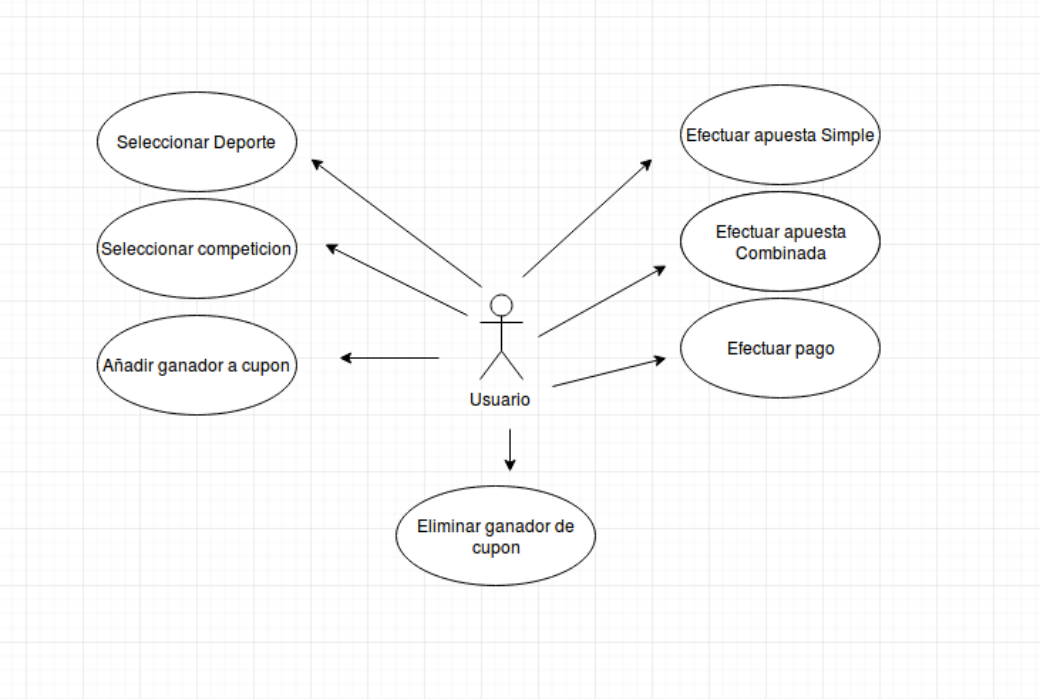


Figura 10: casos de uso

Caso de uso	Seleccionar deporte
Descripción	Este caso de uso describe el proceso en el que el usuario puede elegir entre varios deportes
Actores	Usuario
Precondición	Estar dentro de Enjoybets
Escenario Principal	• El caso de uso se inicia cuando el usuario entra en la página principales • El sistema lista todos los deportes disponibles • El usuario elige uno entre todos los disponibles

Caso de uso	Seleccionar competición
Descripción	Este caso de uso describe el proceso en el que el usuario puede elegir entre varias competiciones dentro de un mismo deporte
Actores	Usuario
Precondición	Estar dentro de Enjoybets y haber seleccionado un deporte
Escenario Principal	• El caso de uso se inicia cuando el usuario selecciona un deporte • El sistema muestra un listado de competiciones de ese deporte • El usuario elige una de las competiciones en la cual quiere realizar su apuesta

Caso de uso	Añadir ganador al cupón
Descripción	Este caso de uso describe el proceso en el cual el usuario selecciona un ganador entre la oferta de mercados desplegados en cada competición o deporte
Actores	Usuario
Precondición	El usuario tiene que tener a la vista algún mercado donde poder seleccionar ganador
Escenario Principal	• El caso de uso se inicia cuando el usuario tiene el listado de mercados disponibles • El usuario selecciona un ganador del mercado • El sistema lo añade al cupón

Caso de uso	Eliminar ganador del cupón
Descripción	Este caso de uso describe el proceso en el cual el usuario borra una selección que tiene en el cupón
Actores	Usuario
Precondición	El usuario tiene que tener algún ganador seleccionado en el cupón
Escenario Principal	• El caso de uso se inicia cuando el usuario tiene ganadores seleccionados en el cupón • El usuario hace clic en eliminar ganadores • el sistema lo elimina del cupón

Caso de uso	Efectuar apuesta simple
Descripción	Este caso de uso describe el proceso en el cual el usuario convierte el cupón con una selección en apuesta
Actores	Usuario
Precondición	El usuario solo tiene que tener una selección dentro del cupón
Escenario Principal	• El caso de uso se inicia cuando el usuario tiene una única selección en el cupón • El usuario hace clic en el botón de apostar • El sistema muestra una ventana donde el usuario introduce la cantidad que quiere apostará • El usuario pone la cantidad • El usuario acepta la apuesta

Caso de uso	Efectuar apuesta combinada
Descripción	Este caso de uso describe el proceso en el cual el usuario convierte el cupón con más de una selección en apuesta
Actores	Usuario
Precondición	El usuario solo tiene que tener más de una selección dentro del cupón
Escenario Principal	• El caso de uso se inicia cuando el usuario tiene varias selecciones en el cupón • El usuario hace clic en el botón de apostar • El sistema muestra una ventana donde el usuario introduce la cantidad que quiere apostará • El usuario pone la cantidad • El usuario acepta la apuesta

Caso de uso	Efectuar pago
Descripción	Este caso de uso describe el proceso en el cual el usuario selecciona un ganador entre la oferta de mercados desplegados en cada competición o deporte
Actores	Usuario
Precondición	Haber efectuado una apuesta
Escenario Principal	• Este caso de uso se inicia cuando el usuario indica el importe que quiere apostar. • El usuario hace clic sobre el botón de aceptar • El sistema procesa la apuesta y efectúa el cobro de la cartera del usuario

5 DISEÑO

5.1 INTRODUCCIÓN

Aquí se verá en más detalle la plataforma web que se va a desarrollar.

Dividiremos el diseño en tres capas, la capa de presentación donde veremos las primeras capturas del producto final, la capa de persistencia donde veremos los diagramas de las tablas empleadas en la base de datos y la capa de lógica que la veremos con un pequeño diagrama de flujo.

5.2 CAPA DE PRESENTACIÓN

Figura 11: página principal Enjoybets

En la imagen anterior podemos ver la página de bienvenida de nuestra plataforma de apuestas, como se puede ver es una página principal bastante sencilla y atractiva de cara al público.

Figura 12: página de apuestas

En esta imagen vemos la pantalla donde el usuario puede realizar todas las acciones, vemos que está muy bien estructurada y definida.

En la parte izquierda de la pantalla tenemos los deportes, como podremos ver más adelante se despliega y dentro de cada deporte estas sus competiciones. Al seleccionar uno se despliega en la parte central los partidos disponibles.

En la parte central como ya hemos dicho están situados todos los partidos con los ganadores que podemos seleccionar y sus respectivas cuotas, en esta parte central es donde el usuario realizara más acciones dentro de nuestro portal y siguiendo la línea es un apartado muy bien estructurado limpio y de fácil utilización ya que con un simple clics sobre el ganador que quiera seleccionar el usuario este se añadirá en la parte derecha donde se encuentra el apartado de cupón.

En la parte derecha de la pantalla tenemos dos apartados, una las apuestas realizadas por el usuario y la otra el cupón, en el apartado de las apuestas se encuentran las apuestas ya realizadas por el usuario y en el apartado del cupón lo que veremos son las selecciones de ganadores elegidas por el usuario antes de realizar la apuesta.

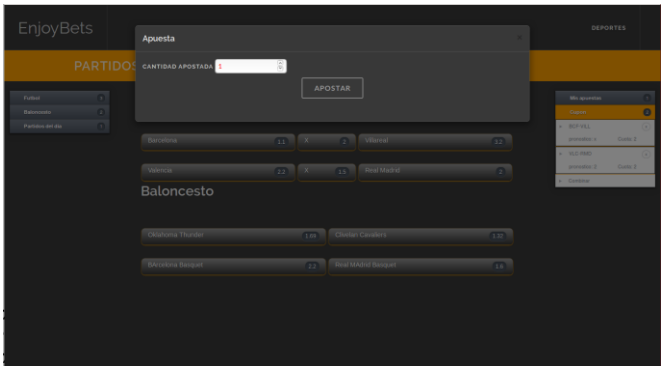


Figura 13: ventana de apuesta

Esta ventana emergente sale cuando el usuario pulsa el botón de realizar apuesta sobre el cupón. Como podemos ver solo con poner la cantidad que quiere apostar y pulsando sobre el botón apostar tendríamos la apuesta realizada de manera muy fácil y directa.

5.3 CAPA DE PERSISTENCIA

La base de datos será de tipo relacional, usaremos MySQL para la capa de persistencia. En este apartado explicaremos un poco las tablas sus campos y relaciones entre tablas que existe en la aplicación.

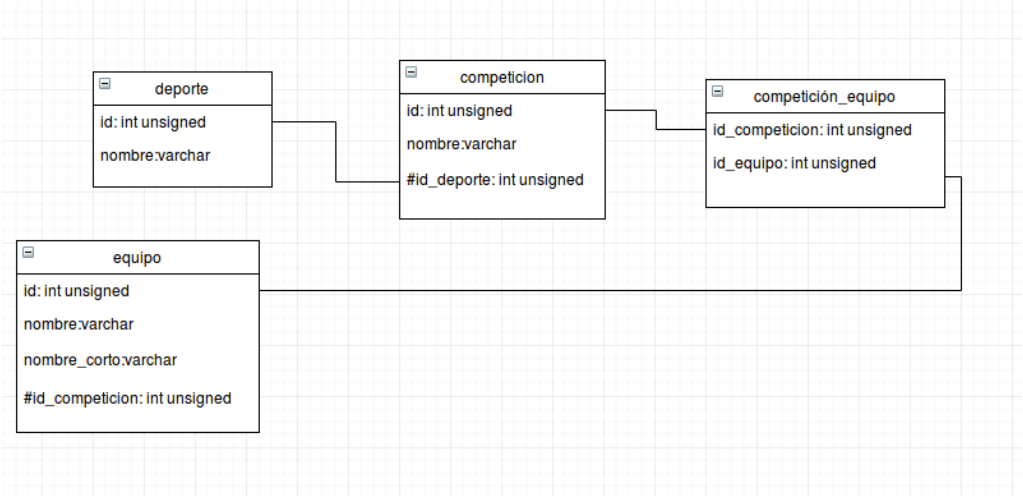


Figura 14: Organización de los mercados

En la ilustración 11 observamos las tablas con las cuales organizamos jerárquicamente los equipos deportes y competiciones.

Estas tablas son muy simples los deportes tienen varias competiciones y cada competición puede tener muchos equipos, al mismo tiempo una competición solo puede pertenecer a un deporte y un equipo puede estar en más de una competición.

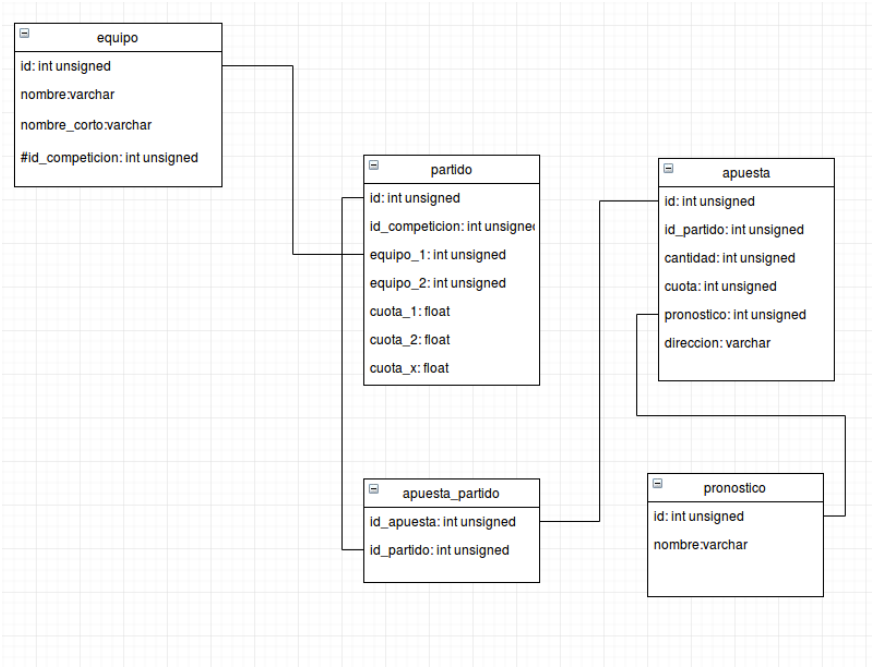


Figura 15: tablas de la parte de las apuestas

Esta parte de la base de datos es algo menos intuitiva ya que es la estructura que nos permite gestionar todo el sistema de las apuestas, por ello haremos una breve explicación de algunos campos y de cómo se relacionan entre si las tablas.

Un partido siempre va a tener dos equipos y una apuesta puede estar formada por uno o más partidos al mismo tiempo que un partido lo podemos tener en más de una apuesta.

En la tabla partidos tenemos los siguientes campos:

- id: identificador.
- id_competicion: sirve para identificar el partido por si se repitiera un mismo partido en varias competiciones.
- $equipo_1$:id del equipo local.
- $equipo_2$: id del equipo visitante.
- cuota_1: cuota o probabilidad del equipo 1.
- cuota_2: cuota o probabilidad del equipo 2.
- cuota_x: cuota o probabilidad del empate en caso de haberlo.

En la tabla apuesta tenemos los siguientes campos:

- id: identificador.
- id_partido: sirve para identificar el partido.
- cantidad: cantidad de bitcoins apostada.
- cuota: el multiplicador por el cual tenemos que pagar si sale ganadora esta apuesta.
- dirección: dirección bitcoin a la cual tenemos que ingresar en el caso de ser ganador.

5.4 CAPA DE LÓGICA

En esta parte pretendemos exponer la parte de la lógica de negocio de la aplicación de apuestas.

Esta capa es donde se sitúa toda la lógica del negocio, es la que se encarga de la transformación de datos.

- recibe entrada de datos por parte de los usuarios.
- realiza todas las operaciones oportunas sobre estos.
- almacena en la capa de persistencia los resultados de las operaciones efectuadas sobre los datos de entrada.

En nuestra aplicación será la encargada de recibir las selecciones de los usuarios tales como:

- selección de ganadores.
- cantidad apostada.
- dirección de cartera bitcoin.

Realizar las operaciones pertinentes como:

- combinar cuotas.
- efectuar el cobro de la cantidad apostada por el usuario de su cartera bitcoin.
- almacenar los datos de interés en la capa de persistencia.

6 DETALLES DE IMPLEMENTACIÓN

6.1 TECNOLOGÍAS UTILIZADAS

6.1.1 Parte del servidor

Figura 16: logo Laravel

Laravel es un framework PHP enfocado principalmente a desarrollo y servicios web, y que ofrece mucha flexibilidad en la arquitectura de la aplicación a desarrollar. Una de las características de Laravel es la gran cantidad de funcionalidad *"out of the box"* que aporta al desarrollador, lo que redunda en una velocidad y calidad de desarrollo bastante altas.

Figura 17: logo MySQL

MySQL es sistema de gestión de bases de datos relacional, multihilo y multiusuario. En el caso de las webs uno de los gestores más utilizados, este sistema que nos permite a través de una serie de sentencias, tener una información almacenada en una base de datos recuperarla en el momento en el que la necesitemos de una forma eficiente y rápida.

Figura 18: logo apache

Apache es un poderoso servidor web, cuyo nombre proviene de la frase inglesa “a patchy server” y es completamente libre, ya que es un software Open Source y con licencia GPL. Una de las ventajas más grandes de Apache, es que es un servidor web multiplataforma, es decir, puede trabajar con diferentes sistemas operativos y mantener su excelente rendimiento.

Figura 19: logo GIT

Git es un sistema de control de versiones distribuido cuyo objetivo es el de permitir mantener una gran cantidad de código. Hay dos características que lo diferencian de otros controles de versiones.

La primera es la forma en la que trata las modificaciones en los ficheros, en GIT a diferencia de los otros controladores de versiones si un archivo no es modificado no crea ninguna copia nueva.

La otra gran diferencia es su eficiencia, al guardar una copia local en cada programador no se hacen muchas operaciones en la red.

6.1.2 Parte del cliente

En la parte del cliente hemos utilizado HTML 5 CSS 3 JS y su librería jQuery.

Figura 20: logos de tecnologías de parte del cliente

A partir de un código HTML + CSS que representa la estructura y estilos básicos de una página web, JavaScript nos sirve para dotar de procesos y efectos dinámicos a la aplicación web.

6.2 DISEÑO ARQUITECTÓNICO

La arquitectura de Laravel es un flujo de comunicación entre el ***Foundation*** del framework, los ***Services Providers***, una estructura de Controllers con ***Middlewares*** y una capa de servicios que se comunica con el acceso a datos del ***ORM*** y al final, con la base de datos.

6.2.1 Elementos de la arquitectura

- **El Foundation**
  Es el núcleo del framework, que en teoría nunca deberías tocar. Es código de Laravel que no es parte de tu app. Está en la carpeta vendor.

- **Los Services Providers**
  Son la base del inicio de tu aplicación. Están encargados de arrancar tu app y ahí encontrarás la conexión entre el Foundation de Laravel y las rutas de la app, interfaces, objetos de servicios, containers, eventos, errores, etc. La configuración de toda tu app pasa por los Service Providers.

- **Los Middlewares**
  Son objetos que cubren las cosas que siempre necesitas en una web app. Cosas como cifrado (encriptación) de cookies, autenticación y usuarios,
- protección contra CSRF y XSS, etc.

- **El ORM**
  Es el acceso a la base de datos. En lugar de escribir SQL directo (que también puedes, si quieres), usa el método de PHP, similar al Active Record de Rails. Las consultas a través del ORM de Laravel ya están optimizadas.

6.2.2 Estructura de los ficheros y directorios

Al empezar una nueva aplicación con Laravel, se creará una estructura muy ordenada de directorios que nos facilitará trabajar con él.

El directorio de una aplicación de Laravel se ve de la siguiente forma:

Figura 21: directorio aplicación Laravel

- **app**
  Esta carpeta es en donde estará la mayor parte del código de tu proyecto. Aquí encontraremos directorios como Console, HTTP, que funcionan como una API al núcleo de tu aplicación.
  Otras carpetas corresponden a patrones de diseño orientado a objetos, que puedes usar dentro de Laravel.

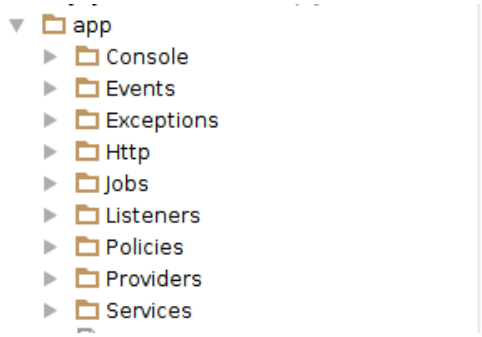


Figura 22: carpeta app

- **bootstrap**
  Son archivos autogenerados por Laravel para arrancar tu app.
- **config**
  Laravel configura con arrays de PHP. Los archivos donde se declaran esos arrays están en esta carpeta. La configuración común a todos los entornos está aquí. En el resto de tu app, puedes leer esto con el objeto Config.
- **database**
  En Laravel, las bases de datos y su estructura interna requiere ser entendida antes de iniciar a programar. Esto se logra a través de migraciones. Además, existen semillas para generar datos de prueba, entre otras herramientas. Para eso es esta carpeta.
- **public**
  El contenido de esta carpeta será accesible por tus usuarios. Aquí se almacenan los archivos estáticos de tu aplicación, archivos html, javascript, css, vídeos e imágenes.
- **resources**
  Aquí se guardan los archivos de vistas y raw assets (LESS, SASS, CoffeeScript). Es decir, aquí está el Frontend de la aplicación. Además, están archivos de traducción, si tienes una app multilenguaje.
- **storage**
  Es la carpeta temporal de Laravel. Se autogeneran logs, cache de templates, etc.
- **tests**
  Aquí se guardan las clases responsables de realizar el unit testing a los diversos componentes de tu aplicación. Laravel, por defecto, usa PHPUnit para hacer testing.

- **vendor**

  Esta carpeta es administrada por Composer. Aquí verás las dependencias y librerías del proyecto. Aquí está el core de Laravel.

6.3 PATRONES UTILIZADOS

6.3.1 MVC

El patrón MVC se encarga de separar la lógica de negocio de la interfaz de usuario y es el más utilizado en aplicaciones web, ya que facilita la funcionalidad, mantenibilidad, y escalabilidad del sistema, de forma cómoda y sencilla, a la vez que ayuda no mezclar lenguajes de programación en el mismo código.

Hemos decidido aplicar el patrón MVC porque después de haber hecho varias aplicaciones web me he dado cuenta que aporta muchos más beneficios que desventajas a pesar de la curva de aprendizaje para poder aplicarlo de manera correcta.

Algunos de los motivos de su elección has sido:

Laravel te da facilidades para la aplicación del patrón.
Laravel es un framework basado en el patrón MVC y al estar desarrollando la aplicación con Laravel hemos decidido explotar todos los recursos que nos ofrece.

Separar responsabilidades
Nos permite separar tipos de código, por una parte, tenemos el HTML, por otra parte, el código PHP y por otra parte los datos.

Reutilizar Código
Nos permite reutilizar código, regresar vistas totales o parciales, evitando duplicar estilos o contenido en las vistas. Todo el manejo de datos se realiza en los modelos, por lo que si modificamos la base de datos solo es necesario modificar el modelo correspondiente para que permita manejar los datos actualizados, sin necesidad de actualizar cada lugar donde es utilizado.

6.3.1.1 Implementación del patrón MVC en nuestro proyecto

El Modelo Vista Controlador es un patrón de diseño de aplicaciones que se ha puesto muy de moda en los últimos años. Tiene varias razones, pero la más importante es separar el código en tres partes bien definidas: el modelo, la vista y el controlador.
Laravel propone en el desarrollo usar 'Routes with Closures', en lugar de un MVC tradicional con el objetivo de hacer el código más claro. Aun así, permite el uso de MVC tradicional.

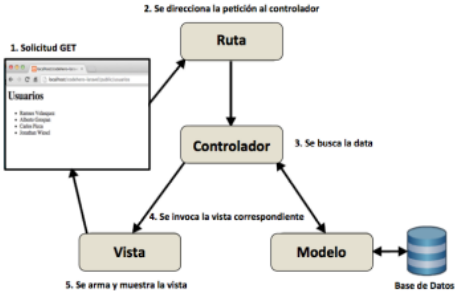
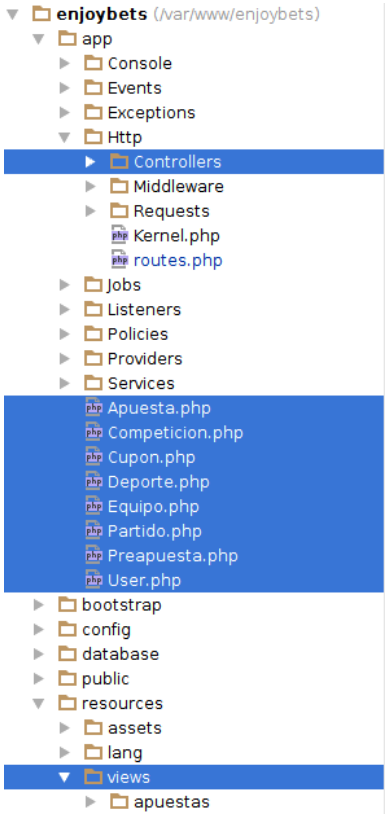


Figura 23: patrón MVC adaptado al proyecto

Esta es la arquitectura que sigue Laravel para atender las solicitudes del usuario.

6.3.1.2 Estructura del MV en el proyecto



En esta carpeta están los controladores

Este es el archivo de las rutas

Aquí podemos ver los modelos

Y en esta carpeta tenemos las vistas

Figura 24: estructura carpetas

Rutas

Las rutas son las encargadas de recibir la petición del usuario y redirigirlo a los controladores para que estos se comuniquen con los modelos y devuelvan los datos a las vistas.

```php
<?php

/*...*/

/*Route::get('/', function () {
    return view('welcome');
});

Route::auth();

Route::get('/home', 'HomeController@index');*/

Route::group(['Midelware'=>['web']],function(){
    Route::get('/', function () {
        return view('welcome');
    });

    Route::auth();
    Route::get('/home', 'HomeController@index');

    Route::get('partidos', 'PartidosController@index');

    Route::get('futbol', 'PartidosController@futbol');


    Route::get('baloncesto', 'PartidosController@baloncesto');


    Route::get('tenis', 'PartidosController@tenis');


    Route::post('apostar','ApuestasController@apostar');
    Route::post('combinar','ApuestasController@combinar');

    Route::get('eliminarcombinada/{id_partido}','ApuestasController@descombinar');
    Route::any('apostar-combi','ApuestasController@apostarcombi');

});
```

Figura 25: archivo de rutas

Código de rutas que nos lleva a los métodos del controlador correspondiente.

Modelos

Los modelos son clases encargadas de trabajar con las consultas de la base de datos, es decir que por cada tabla tendremos una clase, cada registro será un objeto y las consultas se llamarán a través de métodos de esas clases. A su vez Laravel trabaja con Eloquent que es un ORM que nos facilitará el trabajo de las consultas a través de métodos ya establecidos, estos nos permitirán realizar las tareas más comunes y que más se repiten en una base de datos como insertar, recuperar registros por su id, modificar esos registros, listarlos, eliminarlos, etc.

```php
use Illuminate\Database\Eloquent\Model;
class Partido extends Model
{
    /**
     * The database table used by the model.
     *
     * @var string
     */
    protected $table = 'partido';
    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */

    protected $fillable = [
        'id_competicion', 'equipo_1', 'equipo_2','cuota_empate'
    ];
    public function equipo1()
    {
        return $this->belongsTo('App\Equipo','equipo_1');
    }
    public function equipo2()
    {
        return $this->belongsTo('App\Equipo','equipo_2');
    }
    public function competicion(){
        return $this->belongsTo('App\Competicion','id');
    }

    public function apuestas(){
        return $this->belongsToMany('App\Partido','apuesta_partido','id_partido','id_apuesta');
    }
    public function cupones(){
        return $this->belongsToMany('App\Partido','cupon_partido','id_partido','id_cupon');
    }

    public $timestamps = false;
}
```

Figura 26: código de un modelo

En el Modelo está el nombre de la tabla en la base de datos y los campos accesibles desde Laravel con Eloquent como los métodos para usar el patrón repositorio que veremos más adelante

Vistas

Es el producto final de una petición, el código HTML que se le devuelve al cliente, aquí no debería haber ninguna lógica, sin embargo, puede contener impresiones de variables, condicionales o bucles; pero no más que eso. La vista tiene un fin y es ése, entregar el código HTML de respuesta.

```
@extends('layouts.app')

@section('content')

    <section id="slider" class="slider-parallax swiper_wrapper full-screen force-full-screen clearfix">

        <div class="slider-parallax-inner">


            <div class="swiper-container swiper-parent">

                <div class="swiper-wrapper">
                    <a href="/partidos"><div class="swiper-slide dark" style="...">

                            <div class="container clearfix">
                                @if (count($errors) > 0 )
                                    <div class="slider-caption slider-caption-center alert alert-danger center nobottommargin">
                                        <strong style="..."> {!! $errors !!}</strong>
                                        <div><a href="login" style="...">Login</a></div>
                                    </div>
                                @else
                                <div class="slider-caption slider-caption-center">
                                    <h2 data-caption-animate="fadeInUp">APUESTA YA</h2>
                                    <p data-caption-animate="fadeInUp" data-caption-delay="200" >
                                        Todas las apuestas de tus deportes favoritos a un solo click. A que esperas!.</p>
                                </div>
                            </div>
                        @endif
                    </a>
                </div>



            </div>

    </section>
@endsection
```

Figura 27: código de una vista

Como vemos en Laravel se usa layouts para hacer más reutilizable el código que es uno de los propósitos de aplicar este tipo de patrones.

Controladores

Los controladores son clases con métodos, también llamados acciones, estas acciones se comunicarán con los modelos para hacer consultas a la base de datos, y con las vistas para devolver una respuesta al cliente.

```php
<?php

namespace App\Http\Controllers;

use ...

class ApuestasController extends Controller
{
    public function index()
    {

        return view('apuestas.index');
    }


    public function combinar(Request $request){//add
        $user = $request->user();
        //dd($user->id);
        if (!isset($user)) {
            return view('errors/503');
        }
        $data =$request->all();

        //compruebo si la apuesta ya existe en la combinada por que no puede estar duplicada
        $combinada=Cupon::where('id_usuario',$user->id)->where('id_partido',$data['partido_id'])->get();
        //dd($combinada[0]);
        if(!empty($combinada[0])){
            $error='este partido no se puede combinar';
            return back()->with(['combinada'=>$combinada,'errors'=>$error]);
        }else{
            $Grupo_apuestas= new BetsGroup();
            $Grupo_apuestas->addBet(new Bet($data['partido_id'],$user->id,$data['cuota_partido'],$data['pronostico']));

            $combinadas=Cupon::where('id_usuario',$user->id)->get();

            return back();
        }


    }
```

Figura 28: controlador de las apuestas

6.3.2 Simple factory

Similar a la Abstract Factory, este patrón se utiliza para crear la serie de objetos relacionados o dependientes. La diferencia entre este y el patrón de fábrica abstracta es que el patrón de la fábrica simple utiliza un solo método para crear todos los tipos de objetos que puede crear.

Aquí tenemos un ejemplo de la jerarquía de este patrón de diseño.

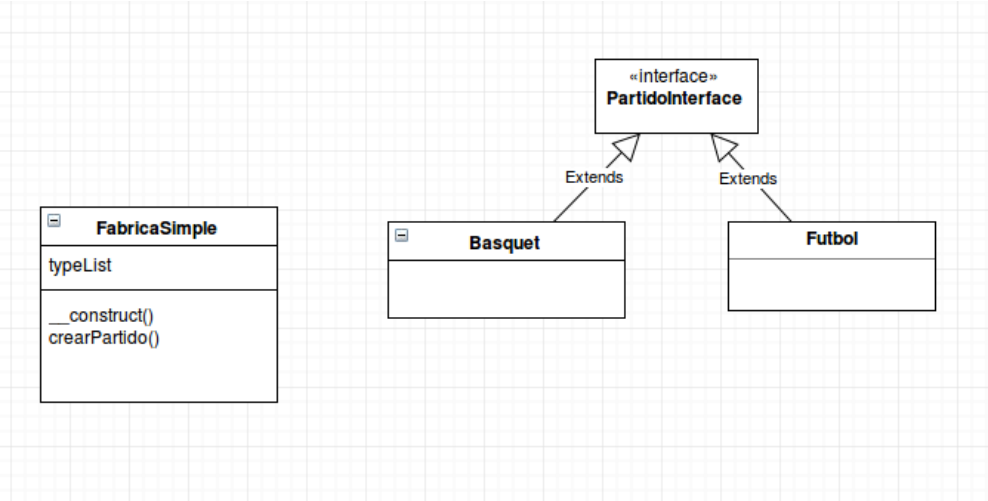


Figura 29: patrón simple factory

En nuestro proyecto las clases utilizadas son las siguientes:

Interface

```php
<?php
/** Created by PhpStorm. ...*/

namespace App\Services\SimpleFactory;


interface PartidoInterface
{
    public function Cuota();
}
```

Figura 30: Interface de la clase partidos

Esta es nuestra interface de la que van a heredar los partidos de fútbol de básquet y en un futuro de todos los deportes que vayamos poniendo en la web.

Futbol:

```php
<?php
/** Created by PhpStorm. ...*/

namespace App\Services\SimpleFactory;


use ...

class Futbol implements PartidoInterface
{
    protected $equipo_1;
    protected $equipo_2;
    protected $cuota_1;
    protected $cuota_X;
    protected $cuota_2;
    protected $id_competicion;
    protected $nom_competicion;
    protected $nom_equipo_1;
    protected $nom_equipo_2;
    protected $deporte;
    protected $id;

    public function __construct($partido){...}

    public function cuota()
    {

        // TODO: Implement partidoDe() method.
    }

    public function getId()
    {
        return $this->id;
    }

    public function getCuota1()
    {
        return $this->cuota_1;
```

Figura 31: clase de un deporte con tres opciones

Aquí vemos que hereda de la clase PartidoInterface.

Básquet:

```php
<?php
/** Created by PhpStorm. ...*/

namespace App\Services\SimpleFactory;


class Basquet implements PartidoInterface
{
    protected $equipo_1;
    protected $equipo_2;
    protected $cuota_1;
    protected $cuota_X;
    protected $cuota_2;
    protected $id_competicion;
    protected $nom_competicion;
    protected $nom_equipo_1;
    protected $nom_equipo_2;
    protected $deporte;
    protected $id;

    public function __construct($partido){...}

    public function cuota()
    {

        // TODO: Implement cuota() method.
    }

    public function getId()
    {
        return $this->id;
    }

    public function getCuota1()
    {
        return $this->cuota_1;
    }
```

Figura 32: clase de un deporte con dos opciones

Aquí vemos que hereda de la clase PartidoInterface.

FabricaSimple:

```php
<?php
/** Created by PhpStorm. ...*/

namespace App\Services\SimpleFactory;


class FabricaSimple
{

    /**
     * @var array
     */
    protected $typeList;
    /**
     * You can imagine to inject your own type list or merge with
     * the default ones...
     */
    public function __construct()
    {
        $this->typeList = array(
            'futbol' => __NAMESPACE__.'\Futbol',
            'baloncesto' => __NAMESPACE__.'\Basquet'
        );
    }
    /** Creates a vehicle. ...*/
    public function crearPartido($tipo,$partido)
    {
        if (!array_key_exists($tipo, $this->typeList)) {
            throw new \InvalidArgumentException("$tipo no es un deporte valido");
        }

        $nombre_clase = $this->typeList[$tipo];

        return new $nombre_clase($partido);
    }
}
```

Typo: In word 'Futbol' more... (Ctrl+F1)

Figura 33: clase fábrica de partidos

Esta clase es la encargada de llamar dependiendo el deporte que queramos a una u otra.

Llamada a la Factory:

```php
$this->factory= new SimpleFactory\FabricaSimple();

$deporte=Deporte::find(1);

foreach($deporte->competiciones as $competicion){

    foreach($competicion->partidos as $partido) {

        $partidos[] = $this->factory->crearPartido("futbol",$partido);


    }}
```

Figura 34: llamada a la factoría

Aquí podemos ver como creamos una fábrica simple para después llamarla con el deporte que queremos crear.

6.3.3 Composite

El patrón Composite nos permite construir objetos complejos partiendo de otros más sencillos utilizando una estrategia de composición recursiva. Así podemos tratar a las apuestas combinadas como apuestas simples. Cada apuesta simple puede combinarse con otras apuestas simples para formar una apuesta combinada.

Combina objetos en estructuras de árbol para representar jerarquías de parte-todo. Permite que los clientes traten de manera uniforme a los objetos individuales y a los compuestos.

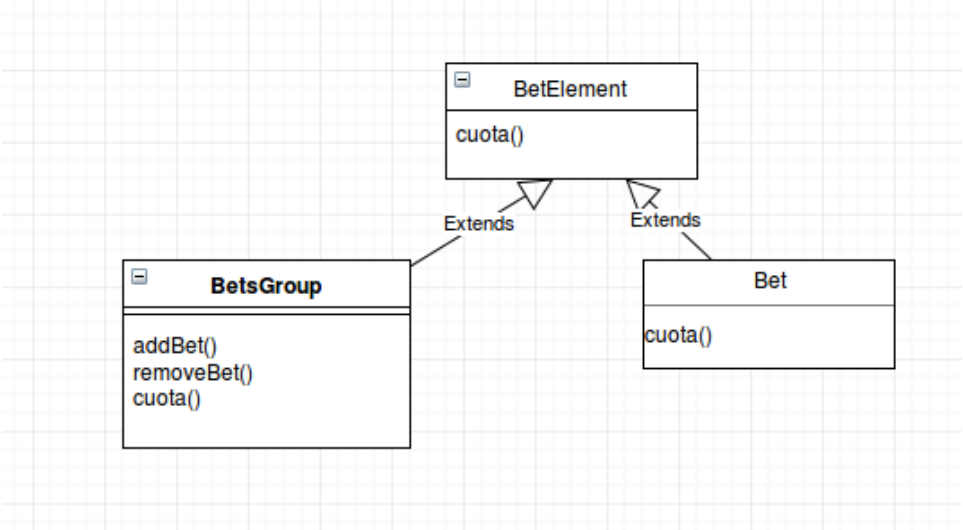


Figura 35: patrón composite para las apuestas

BetElement:

```php
<?php
/** Created by PhpStorm. ...*/

namespace App\Services;


abstract class BetElement
{
    abstract public function cuota(BetElement $bet);
}
```

Figura 36: código de la clase abstracta BetElement

Esta es la case abstracta de la que extienden tanto el objeto simple como el compuesto.

Bet:

```php
<?php
/** Created by PhpStorm. ...*/

namespace App\Services;


use App\Cupon;

class Bet extends BetElement
{
    public $id_partido;
    public $id_usuario;
    public $cuota;
    public $pronostico;

    public function __construct($id_partido,$id_usuario,$cuota,$pronostico){
        $this->id_partido=$id_partido;
        $this->id_usuario=$id_usuario;
        $this->cuota=$cuota;
        $this->pronostico=$pronostico;
        return $this;
    }
    public function cuota(BetElement $bet)
    {
        $cuota=$this->cuota;
        return $cuota;
    }

}
```

Typo: In word 'pronostico' more... (Ctrl+F1)

Figura 37: clase BetElement

Esta es la clase del objeto simple que es la apuesta.

BetsGroup:

```php
namespace App\Services;

use ...

class BetsGroup extends BetElement
{

    public function cuota(BetElement $bet){...}

    public function addBet(BetElement $bet)
    {


        $combinada=new Cupon();
        $combinada->id_partido=$bet->id_partido;
        $combinada->id_usuario=$bet->id_usuario;
        $combinada->cuota=$bet->cuota;
        $combinada->pronostico=$bet->pronostico;

        $combinada->save();
        $combinada->partidos()->attach($combinada->id_partido);

    }
    public function removeBet( $id_apuesta)
    {
        $combinadas=Cupon::find($id_apuesta);
        $combinadas->delete();
        return $combinadas;

    }

    public function apostar($id, $cantidad){...}
}
```

Figura 38: clase BetGroup

En esta tendríamos el objeto compuesto que es la apuesta combinada.

6.3.4 Singelton

Como hemos visto en clase, el patrón Singleton es muy importante y más aún en PHP ya que gracias a él con una única instancia del objeto podemos acceder a él las veces que necesitemos sin la necesidad de crear copias en cada punto del código donde se vaya a utilizar y malgastando recursos.

En nuestro caso he decidido utilizar este patrón para realizar la conexión con la base de datos.

Una conexión estándar a la base de datos sin la utilización de patrones sería algo como:

```
$this->dbh = new PDO("mysql:host=$host; dbname=$nombeBD", $usuario, $clave);
```

Esto se repetiría en todas las clases donde se quiera acceder a la base de datos.

A partir de esto decidimos crear una clase SingletonBD, donde recogeremos toda la información necesaria para establecer la conexión con la base de datos y haremos que sea una especie de "burbuja" que solo se instanciará una vez y a partir de esta se solicitará la información necesaria.

```
<?php
class SingletonBD {
        //creamos las variables globales necesarias para almacenar
        //toda la información de la base de datos
        private $_linkid;
        private $_servidor;
        private $_nombreBD;
        private $_usuario;
        private $_clave;

        // Esta variable de tipo objeto nos servira mas adelante para comprobar
        // si la base de datos ya ha sido instanciada/cargada.
        private static $_self = null;
        private function __construct(){
                $this->_servidor = "192.168.1.222";
                $this->_nombreBD = "BDPoker";
                $this->_usuario = "ruslankyrch";
                $this->_clave = "proyectopoker";
        }
        /**
        * Este metodo instanciar es la esencia del singleton ya que nos evitara crear
        * nuevas instancias de la base de datos, comprueba si existe alguna y si es asi
        * nos la devuelve, en caso contrario crea una nueva
        */

        public static function instranciar(){
                //si la variable $_self, antes declarada, esta vacia...
                if(!self::$_self instanceof self){
                        //se crea un objeto tipo singletonBD en la variable
                        self::$_self = new self;
                }
                return self::$_self;
        }
        public function conectar(){
                //si la variable linkid esta a 0 significa que no tenemos ninguan
                //base de datos conectada actualmente y procedemos a crear la conexion
                if($this->_linkid == 0){

                        $conexionSql = mysql_connect(
                                $this->_servidor,
                                $this->_usuario,
                                $this->_clave);
                        //si no se ha conseguido realizar la conexion...
                        if(!$conexionSql){

                                //lanzamos una excepcion y salimos
                                die('No pudo conectarse: ' . mysql_error());
                        }
                        else //si la conexion se ha establecido correctamente
                        {
                                mysql_set_charset('utf8',$this->linkid);
                        }
                }
                return true;
        }
        private function get_servidor(){
                return $this->_servidor;
                }
        private function get_usuario(){
                return $this->_usuario;
                }
        private function get_clave(){
                return $this->_clave;
                }
        private function get_link_id(){
                return $this->_linkid;
                }
        //Devuelve un entero mayor que 0 si hay conexion,
        //en caso contrario 0
        private function get_nombre_bd(){
                return $this->_nombreBD;
                }
}
```

Figura 39: código singelton

A partir de aquí, nuestras conexiones a la base de datos se resumirán en algo como esto:

```
<?PHP
REQUIRE_ONCE 'SINGLETONBD.PHP';
.
.
.
//ACCEDEMOS AL METODO INSTANCIAR DE NUESTRO SINGLETON Y CONECTAMOS CON LA BASE DE DATOS
$THIS->_BASEDATOS = SINGLETONBD::INSTANCIAR();
$THIS->_BASEDATOS->CONECTAR();
?>
```

Ahora ya podemos utilizar la variable global *_BaseDatos* para realizar las consultas necesarias a nuestra base de datos.

Builder

La intención de este patrón no es más que la de abstraer el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto, de tal forma que el mismo proceso de construcción pueda crear representaciones diferentes.

En Laravel la clase AuthManater necesita crear algunos elementos de seguridad para reusarlos, como las cookies y las sesiones. Para conseguir esto la clase AuthManager necesita usar funciones de guardado como callCustomCreator() y getDrivers() desde la claseManager.

Vamos a ver como se usa este patrón dentro de la estructura de Laravel, para verlo vamos a las siguientes rutas:

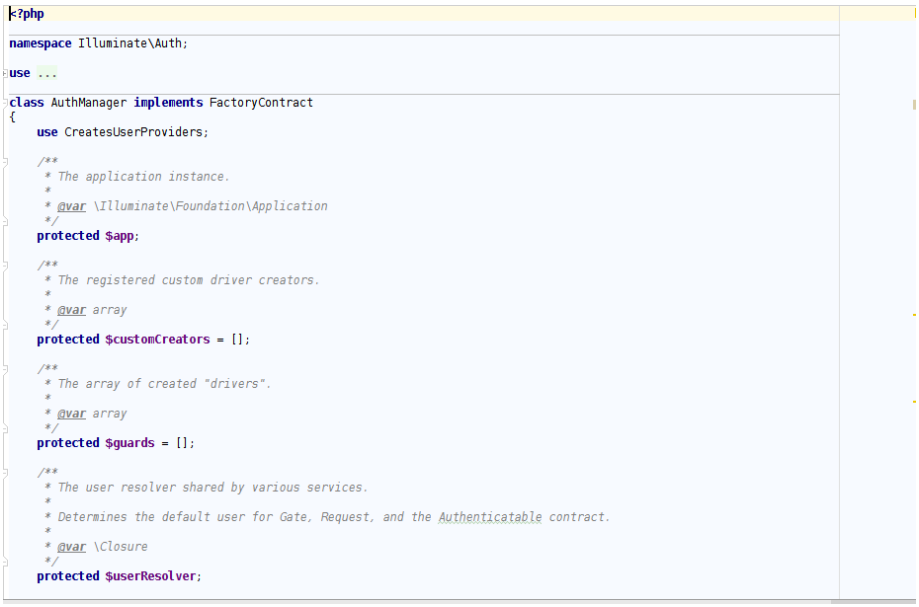
vendor/laravel/framework/src/Illuminate/Support/Manager.php

```php
<?php

namespace Illuminate\Support;

use ...

abstract class Manager
{
    /**
     * The application instance.
     *
     * @var \Illuminate\Foundation\Application
     */
    protected $app;

    /**
     * The registered custom driver creators.
     *
     * @var array
     */
    protected $customCreators = [];

    /**
     * The array of created "drivers".
     *
     * @var array
     */
    protected $drivers = [];

    /**
     * Create a new manager instance.
     *
     * @param
     * @return void
     */
    public function __construct($app)
    {
        $this->app = $app;
    }
```

Return type does not match more... (Ctrl+F1)

Figura 40: clase manager

vendor/laravel/framework/src/Illuminate/Auth/AuthManager.php



```php
<?php

namespace Illuminate\Auth;

use ...

class AuthManager implements FactoryContract
{
    use CreatesUserProviders;

    /**
     * The application instance.
     *
     * @var \Illuminate\Foundation\Application
     */
    protected $app;

    /**
     * The registered custom driver creators.
     *
     * @var array
     */
    protected $customCreators = [];

    /**
     * The array of created "drivers".
     *
     * @var array
     */
    protected $guards = [];

    /**
     * The user resolver shared by various services.
     *
     * Determines the default user for Gate, Request, and the Authenticatable contract.
     *
     * @var \Closure
     */
    protected $userResolver;
```

Figura 41: clase authManager

Laravel usa un mecanismo básico de autenticación, por lo tanto, necesitamos guardar las credenciales en la base de datos. En primer lugar, la clase comprueba la configuración por defecto de nuestra base de datos con el AuthManager: setDefaultDriver función (). Esta función utiliza realmente la clase Manager para operaciones eloquent. Todas las opciones de la base de datos y de autenticación (como el nombre de la cookie) se obtienen a partir del fichero de configuración de la aplicación, excepto el nombre de la tabla modelo de autenticación.

6.3.5 The repository Pattern

Este patrón permite separar la lógica que entrega los datos y los mapea a la entidad desde la lógica de negocios que actúa sobre el modelo. La lógica de negocios debe ser agnóstica con respecto al tipo de datos que se obtiene desde la capa de fuente de datos. Por ejemplo, la fuente de datos puede ser una base de datos, un web service, o un array de datos.

ORM en realidad no es un patrón de repositorio. Hemos añadido una capa de abstracción repositorio delante de sus modelos para llevar a cabo funciones tales como la creación, la búsqueda, borrado, etc. Por ejemplo, pensar en hacer una etiqueta CRM. Al principio es posible que tenga las etiquetas conjunto en una lista CSV en un modelo de usuario. Por lo tanto, el uso de técnicas tradicionales elocuentes que siempre tendría que tener:

```
$USERS = USER::WHERE('TAGS', 'LIKE', $TAG)->GET();
```

Pero para una relación muchos a muchos significa que todo el mundo que usara la consulta anterior se tendría que cambiar a:

```
$TAGS = TAGS::WITH('USERS')->WHERE('TAG', $TAG)->GET();
$USERS = $TAGS->USERS;
```

En su lugar, la creación de un repositorio da la abstracción necesaria para que su controlador tiene el siguiente aspecto:

```
$USERS = $THIS->USERREPOSITORY->GETWHERETAG($TAG);
```

6.4 REFACTORIZACIÓN

6.4.1 Código duplicado en las vistas

Problema:

Esto produce que cada vez que tengamos que hacer un cambio en la parte de los menús laterales que son los deportes disponibles y las apuestas tengamos que aplicar los cambios en todas las vistas donde se cargan estos componentes.

Solución:

Extraer este código a otro archivo y cargarlo desde las diferentes vistas para que si hacemos cambios en el archivo todas las vistas se actualicen.

Ejemplos de código antes del refactoring:

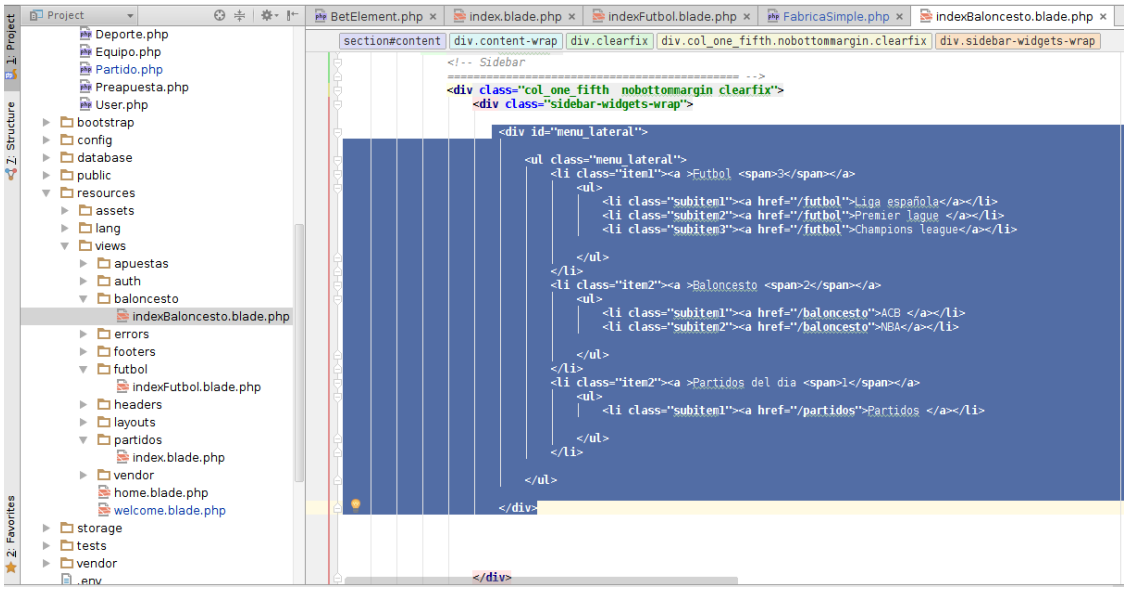


Figura 42: código duplicado, antes del refactoring

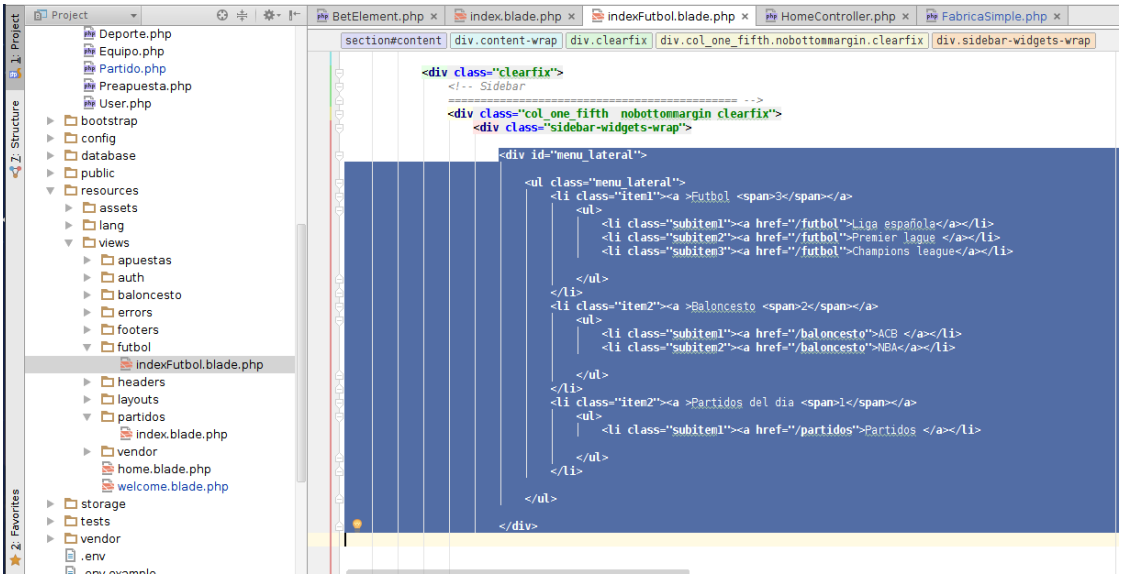


Figura 43: código duplicado, antes del refactoring

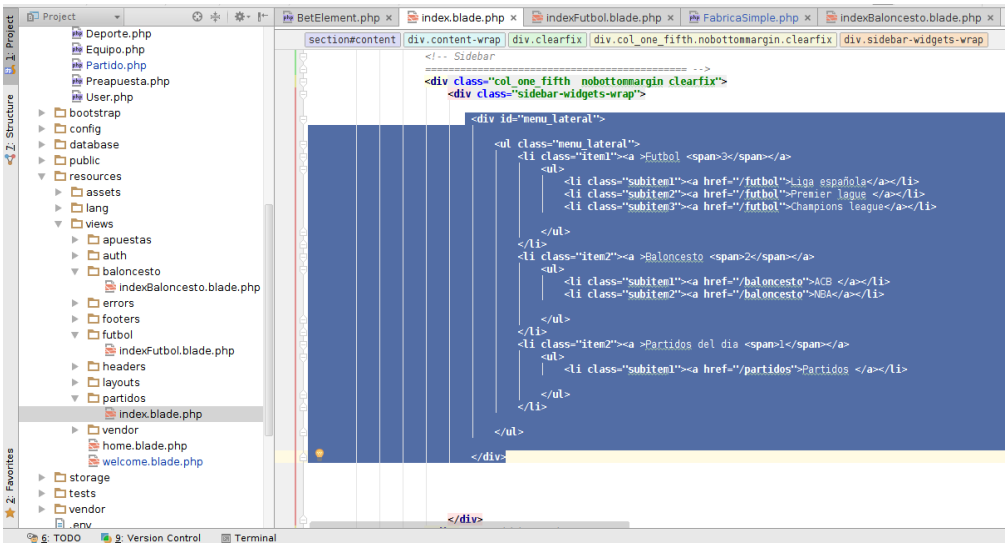


Figura 44: código duplicado, antes del refactoring

Podemos ver como en las tres vistas de los partidos, tanto de baloncesto, fútbol y partidos del día se repite siempre el mismo código.

Ejemplos de código después del refactoring:

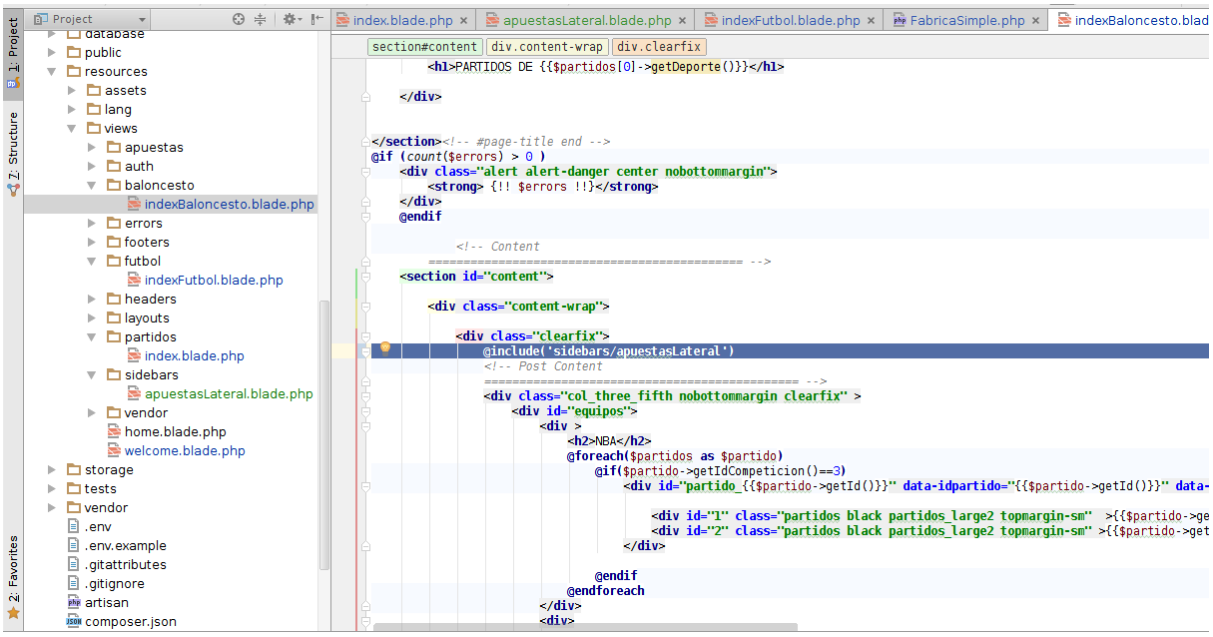


Figura 45: código duplicado, después del refactoring

Aquí podemos ver como quedan las tres vistas antes nombradas después del refactoring. Hemos creado el fichero apuestasLateral.blade.php y lo hemos importado en cada una de las vistas con la línea que se ve en la imagen de arriba, con ello a parte de ahorrarnos líneas de código ahora es más fácil mantener nuestro sitio web, en la refactorización de este desplegable también se hizo el de los partidosLateral.blade.php de la misma manera.

6.4.2 Nombres de variables

Problema:

Elegir mal el nombre de variables puede llevarte a confusiones al interpretar y querer cambiar el código, esto también nos obliga a tener que comentar más de la cuenta el código para que sea entendible.

Solución:

Buscar nombres descriptivos y significativos para las variables y métodos.

Ejemplos de código antes del refactoring:

```
use ...

class ApuestasController extends Controller
{
    public function index()
    {

        return view('apuestas.index');
    }


    public function combinar(Request $request){//add
        $user = $request->user();
        //dd($user->id);
        if (!isset($user)) {
            return view('errors/503');
        }
        $data =$request->all();

        //compruebo si la apuesta ya existe en la combinada por que no puede estar duplicada
        $cupon=Cupon::where('id_usuario',$user->id)->where('id_partido',$data['partido_id'])->get();
        //dd($combinada[0]);
        if(!empty($cupon[0])){
            $error='este partido no se puede combinar';
            return back()->with(['combinada'=>$cupon,'errors'=>$error]);
        }else{
            $Grupo_apuestas= new BetsGroup();
            $Grupo_apuestas->addBet(new Bet($data['partido_id'],$user->id,$data['cuota_partido'],$data['pronostico']));

            $cupon=Cupon::where('id_usuario',$user->id)->get();

            return back();
        }


    }

    public function descombinar(Request $request,$id apuesta){//remove
```

Figura 46: nombres de variables, antes del refactoring

Ejemplos de código después del refactoring:

```
public function combinar(Request $request){//add
    $user = $request->user();

    if (!isset($user)) {
        return view('errors/503');
    }
    $data =$request->all();

    //compruebo si la apuesta ya existe en la combinada por que no puede estar duplicada
    $combinada=Cupon::where('id_usuario',$user->id)->where('id_partido',$data['partido_id'])->get();

    if(!empty($combinada[0])){
        $error='este partido no se puede combinar';
        return back()->with(['combinada'=>$combinada,'errors'=>$error]);
    }else{
        $Grupo_apuestas= new BetsGroup();
        $Grupo_apuestas->addBet(new Bet($data['partido_id'],$user->id,$data['cuota_partido'],$data['pronostico']));

        return back();
    }
```

Figura 47: nombres de variables, después del refactoring

Podemos ver el cambio de nombre en la variable $cupon al mismo tiempo se han eliminado comentarios y alguna línea de código que había quedado obsoleta después de alguna modificación.

6.4.3 Clases con el mismo propósito

Problema:

En este caso nos vemos una función que es apostar y otra que es combinar, la de apostar sirve para apuestas simples y la de combinar para combinadas, pero hacen lo mismo.

Solución:

Hemos eliminado la función de apostar y ahora pasa todo por combinar en el caso de que solo haya una apuesta apostará simple y si hay más apostará combinando las cuotas.

Ejemplos de código antes del refactoring:

```
        return view('apuestas.index');
    }

    public function apostar(Request $request){//bet
        $user = $request->user();
        //dd($user->id);
        if (!isset($user)) {
            return view('errors/503');
        }

        $data =$request->all();

        $cuota=$data['cuota_partido'];
        $combinadas=Cupon::where('id_usuario',$user->id)->get();
        foreach($combinadas as $apuesta){
            $cuota=$cuota*$apuesta->cuota;
            $apuesta->delete();
        }

        $apuesta= new Apuesta();
        $apuesta->id_user=$user->id;
        $apuesta->id_partido=$data['partido_id'];
        $apuesta->cantidad_apostada=$data['cantidad_apostada'];
        $apuesta->cuota=$cuota;
        $apuesta->pronostico=$data['pronostico'];
        $apuesta->save();
        //dd($apuesta);
        $apuesta->partidos()->attach($data['partido_id']);

        return back();
    }

    public function combinar(Request $request){//add
        $user = $request->user();
        //dd($user->id);
        if (!isset($user)) {
            return view('errors/503');
        }

        $data =$request->all();
```

Figura 48: clases con el mismo propósito, antes del refactoring

Ejemplos de código después del refactoring:

```
use ...

class ApuestasController extends Controller
{
    public function index()
    {

        return view('apuestas.index');
    }


    public function combinar(Request $request){//add
        $user = $request->user();

        if (!isset($user)) {
            return view('errors/503');
        }
        $data =$request->all();

        //compruebo si la apuesta ya existe en la combinada por que no puede estar duplicada
        $combinada=Cupon::where('id_usuario',$user->id)->where('id_partido',$data['partido_id'])->get();

        if(!empty($combinada[0])){
            $error='este partido no se puede combinar';
            return back()->with(['combinada'=>$combinada,'errors'=>$error]);
        }else{
            $Grupo_apuestas= new BetsGroup();
            $Grupo_apuestas->addBet(new Bet($data['partido_id'],$user->id,$data['cuota_partido'],$data['pronostico']));

            return back();
        }


    }

    public function descombinar(Request $request,$id_apuesta){//remove
        $user = $request->user();
```

Figura 49: clases con el mismo propósito, después del refactoring

Podemos ver como se ha eliminado la función de apostar y se ha pasado todo a la función de combinar, esta función unifica el hecho de apostar tanto una combinada como una apuesta simple porque al final lo único que hace falta para apostar es una cantidad y una cuota.

7 PRUEBAS

Laravel 5.2 es un framework desarrollado pensando en el uso profesional, por ello esta funcionalidad no podía quedar de lado y de hecho Laravel ahora no sólo incluye soporte para PHPUnit en cada instalación, sino que también tiene su propio componente con el que podemos crear nuestras pruebas de integración.

Aquí dejamos algunos ejemplos de test que tenemos sobre la web.

```php
<?php

use ...

class TestWeb extends TestCase
{
    /**
     * A basic functional test example.
     *
     * @return void
     */
    public function testIndex()
    {
        $this->visit('/')
             ->see('Apuesta ya');

    }
    //prueba para la vista de registro
    public function testRegistro()
    {
        $this->visit('/')
            ->click('Registrarme')
            ->seePageIs('/register');

    }
    //preuba vista de login
    public function testLogin()
    {
        $this->visit('/')
            ->click('Login')
            ->seePageIs('/login');
    }
```

Figura 50: pruebas 1

Estos test nos sirven para confirmar que el archivo de rutas está apuntando bien en las direcciones que nos tiene que redirigir .

```php
//prueba de registro
public function testRegistroNuevoUsuario()
{
    $this->visit('/register')
        ->type('David', 'name')
        ->type('prueba@gmail.com', 'email')
        ->type('741852963', 'password')
        ->type('741852963', 'password_confirmation')
        ->press('Register')
        ->seePageIs('/');
}
//preuba de logearse
public function testLoginUsuario()
{
    $this->visit('/login')
        ->type('prueba@gmail.com', 'email')
        ->type('741852963', 'password')
        ->press('Login')
        ->seePageIs('/');
}

//preuba de apostar
public function testApostar()
{
    $this->testLoginUsuario();
    $this->visit('/partidos')
        ->click('Barcelona')
        ->click('Cupon')
        ->click('Apostar')
        ->type('10','cantidad_apostada')
        ->click('Apostarbtn')
        ->seePageIs('/partidos');
```

Figura 51:Pruebas 2

En estas segundas pruebas estamos probando más el funcionamiento de la web como se puede observar. Cabe destacar que gracias a la prueba de apostar encontramos y solucionamos un error en el campo de cantidad que se podían introducir letras.

7.1 VALIDACIÓN

En las pruebas de validación hemos comprobado el código HTML con la herramienta de W3c.

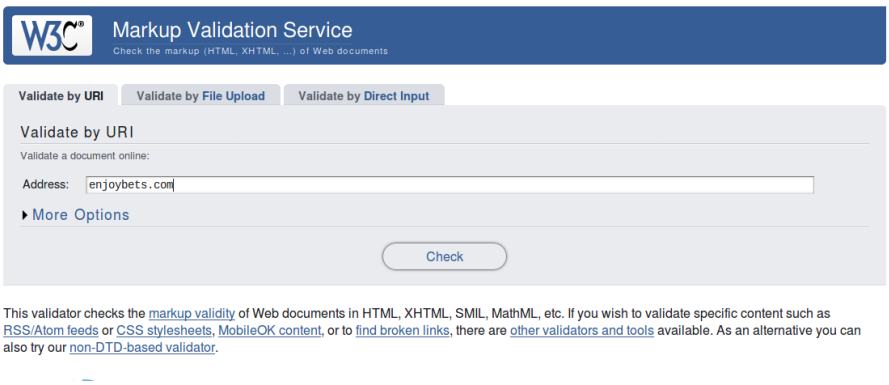


Figura 52: herramienta de validación HTML

Para validar el código CSS hemos utilizado su homonimia, también de W3c.

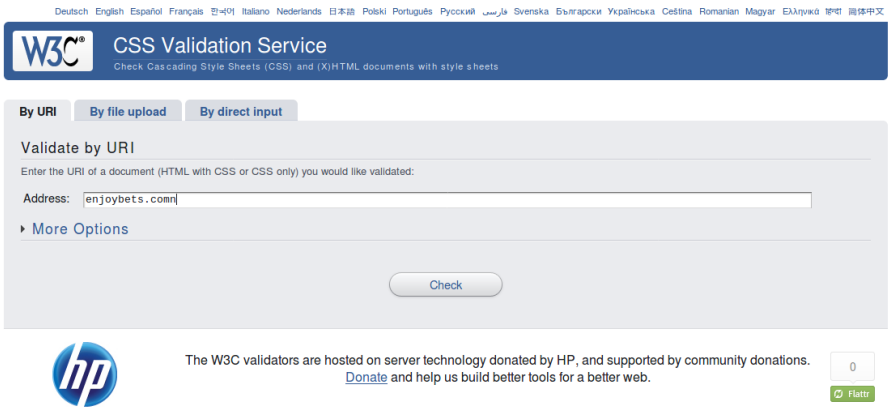


Figura 53: herramienta de validación CSS

7.2 PRUEBAS DE USO

En este apartado vamos a mostrar las diferentes pantallas de la aplicación y daremos una breve explicación de las interacciones que se pueden hacer sobre ellas.

Figura 54: Inicio

La página de inicio es simplemente para dar la bienvenida, el usuario solo tiene la opción de pulsar sobre apuesta ya para entrar a la página de las apuestas. En momentos puntuales del año también se podrán ver las ofertas especiales o promociones existentes en Enjoybets.

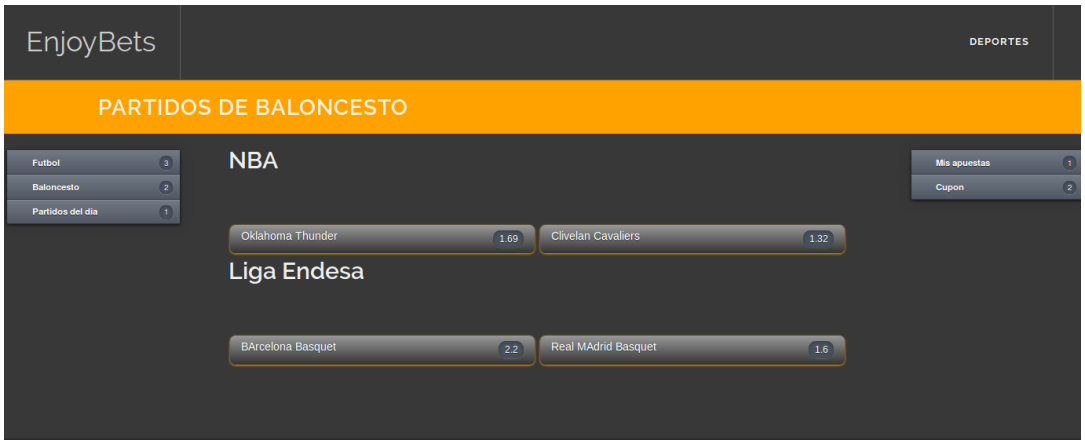


Figura 55: página de apuestas

Desde esta ventana de apuestas es desde la cual el usuario pude interactuar. Desde aquí se puede hacer la selección de deportes y competiciones en la parte izquierda de la pantalla, esto hace que en la parte central aparezcan los partidos disponibles de dicha competición o deporte.

En la parte central, en los partidos el usuario podrá elegir de entre todos sus ganadores para incluirlos en sus cupones y posteriormente serán sus apuestas.

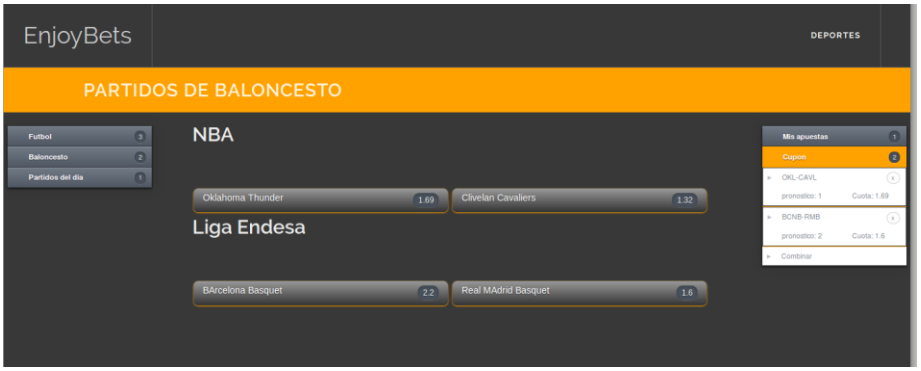


Figura 56: cupón

En el cupón el usuario puede añadir apuestas de la forma que hemos explicado anteriormente, pero también podrá eliminar sus elecciones y una vez el cupón este como le gusta al usuario pulsar el botón de combinar o apostar dependiendo el número de ganadores seleccionados.

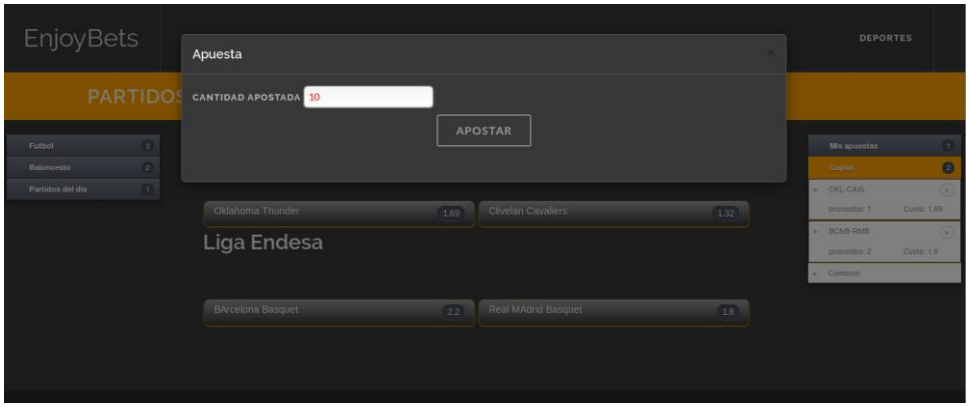


Figura 57:introdución de cantidad a apostar

La ventana emergente de apuesta permite al usuario seleccionar el importe que quiere introducir para realizar su apuesta ahora solo faltara meter su dirección de la cartera de bitcoins.

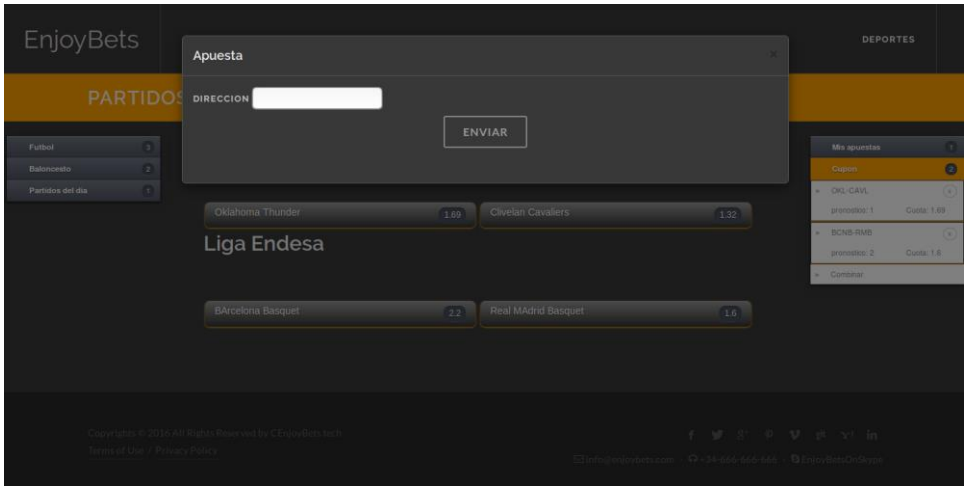


Figura 58: introducción de dirección cartera bitcoin

Esta ventana es el último paso antes de realizar una apuesta, el usuario introducirá la dirección de su cartera bitcoin para que la plataforma proceda al cobro de la cantidad apostada y si todo es correcto se realizara la apuesta.

8 CONCLUSIONES

En este apartado vamos a comentar tanto las experiencias personales como las posibles ampliaciones que podría incluir la aplicación en un futuro.

8.1 CONCLUSIONES PERSONALES

El desarrollo de esta aplicación web me ha permitido explorar tecnologías que no hemos abordado durante el transcurso del grado.

Al enfrentarme a todo el ciclo de vida de un desarrollo software me he dado cuenta que pueden surgir muchos problemas durante cada una de las etapas, pero cada uno de los problemas que han ido apareciendo a lo largo de todo el desarrollo no han hecho más que asentar los conocimientos previos que ya tenía y ampliarlos.

La fase que más reto me ha supuesto ha sido la parte de implementación ya que me han surgido varios problemas y he tenido que realizar varios cambios hasta dar con las soluciones óptimas para solucionar dichos problemas.

Cambien ha sido muy interesante trabajar con las criptomonedas, ya que hasta el momento era un mundo poco conocido para mí y este proyecto me ha dado la oportunidad de conocerlo desde la perspectiva del programador y ha sido un reto importante.

En cuanto a las tecnologías utilizadas para el desarrollo del proyecto puedo decir que no había utilizado mucho ninguna de ellas y en este momento me veo capacitado para el desarrollo de cualquier tipo de aplicación web con cualquiera de ellas.

A nivel personal el framework Laravel me ha sorprendido mucho porque creo que hoy en día sería inviable o con costes muy altos el desarrollo de cualquier aplicación web sin la utilización de algún tipo de framework y para mi Laravel se ha adaptado en todo momento perfectamente y es un framework muy potente y flexible y esta flexibilidad hace que la curva de aprendizaje no sea tan dura.

En líneas generales se puede decir que se han cumplido todos los objetivos propuestos al principio del documento, aunque no sin encontrarme con piedras en el camino.

8.2 POSIBLES AMPLIACIONES

Después del desarrollo de este proyecto cabe recordar que este no es más que el primer paso del camino y que la plataforma queda abierta a muchas ampliaciones de futuro.

Aquí vamos a listar alguna de ellas.

- Añadir más deportes.
- Explorar la posibilidad de hacer usuarios registrados si fuera necesario.
- Añadir más idiomas a la plataforma.
- Aceptar otras criptomonedas.
- Hacer sistemas de fidelización.
- Ofrecer apuestas en vivo.
- Ofrecer más mercados dentro de cada evento deportivo.

9 BIBLIOGRAFÍA

- Manual de PHP: http://php.net/manual/es/index.php
- Manual Laravel: https://laravel.com/docs/5.4
- Apache: https://httpd.apache.org/docs/
- Composer: https://getcomposer.org/doc/
- Manual de GIT: https://git-scm.com/docs/user-manual.html
- Wikipedia: https://es.wikipedia.org/wiki/Wikipedia:Portada
- Blockchain: https://blockchain.info/es
- W3c: http://www.w3c.es/
- Validación CSS: https://jigsaw.w3.org/css-validator/
- Validación HTML: https://validator.w3.org/
- Documentación asignatura Diseño de software
- Documentación asignatura análisis y especificación de requisitos