FootballBoard

Manual del desarrollador

Versión 1.0

Historia de revisiones

|  |  |  |  |
| --- | --- | --- | --- |
| Fecha | Versión | Descripción | Autor |
| 11/10/2016 | 1.0 | Descripción de las funcionalidades y estructura del código | Mabel Geronimo, Carlos Valerio, Brandon Mendoza y Nathalia Garcia |

Contenido

[Estructura de código: Manual para el desarrollador 2](#_Toc463968247)

[Objetivo del documento 2](#_Toc463968248)

[Ambiente de desarrollo 2](#_Toc463968249)

[Patrón de arquitectura 2](#_Toc463968250)

[Vistas 3](#_Toc463968251)

[Modelos 4](#_Toc463968252)

[Controladores 5](#_Toc463968253)

# **Estructura de código:** Manual para el desarrollador

## **Objetivo del documento**

El programa FootballBoard es un marcador de jugadas de un juego de Fútbol desarrollado por cuatro jóvenes estudiantes de las carreras sistemas y software de la universidad INTEC. La finalidad de este manual es dar a conocer a personas de experiencia en el desarrollo y la programación, así como otros interesados, aspectos claves referentes al código del proyecto.

La jerarquía de carpetas/paquetes se define en la imagen que se encuentra como apéndice a este documento denominada “jerarquiaCarpetas.png”. Es esta jerarquía, los archivos que incluye y las funcionalidades que permite las que iremos a describir a continuación.

## **Ambiente de desarrollo**

Se ha usado Netbeans para el desarrollo de la aplicación y Java como lenguaje de programación. El programa ha de funcionar con la versión 1.7 de Java o versiones posteriores.

Además, se hace uso de dos librerías extras. Una llamada jcalendar y la otra beansbinding, cuyos archivos jar de no poseerse tendrán que ser instalados y agregados a su carpeta java. En la aplicación, existe una carpeta llamada librerías en la cual se pueden encontrar los archivos jar mencionados.

## **Patrón de arquitectura**

Entiéndase por este término que los patrones arquitectónicos, o patrones de arquitectura, también llamados arquetipos son los encargados de ofrecer soluciones a problemas de arquitectura de software en ingeniería de software. Dan una descripción de los elementos y el tipo de relación que tienen junto con un conjunto de restricciones sobre cómo pueden ser usados. Un patrón arquitectónico expresa un esquema de organización estructural esencial para un sistema de software, que consta de subsistemas, sus responsabilidades e interrelaciones. En comparación con los patrones de diseño, los patrones arquitectónicos tienen un nivel de abstracción mayor.

Basados en esta definición establecemos la importancia y razón de comenzar por esta parte, ya que la arquitectura es parte fundamental para comprender y aprovechar el código ya creado para crear más sobre el mismo.

En nuestro proyecto hemos hecho uso del patrón MVC. El modelo–vista–controlador (MVC) es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. 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.

Las ventajas de utilizarlo es que nos permite segmentar de forma entendible la lógica del código, además de que la mayoría de miembros del mismo habían tenido experiencia con el mismo. La descripción de los archivos en cada uno de los pilares se encuentra en el documento “jerarquiaCarpetas.png”

## **Vistas**

Las vistas principales que se manejan en el presente proyecto son las siguientes:

**Menú principal.** Provee un menú inicial con las opciones que permite el programa realizar en el entorno.

Es importante conocer que las funcionalidades generales del proyecto son:

* + - * + Registrar equipos. Para esto se debe proveer el nombre del equipo y una lista de sus jugadores.
        + Guardar información de un juego que ha sucedido antes con el fin de tenerlo en la base de datos y consultarlo posteriormente.
        + Guardar información de un juego que sucede en el momento con el fin de tenerlo en la base de datos y consultarlo posteriormente. Así como obtener la puntuación de cada equipo de forma automática.
        + Consultar información de jugadas registradas anteriormente.

**Equipo.** Es la vista que describe y muestra los equipos que se encuentran en el proyecto. Así como las funcionalidades que se pueden ejecutar sobre esta. Entre estas están:

* + - * + Suministrar una lista de jugadores de forma manual a través de la digitación de los datos pertinentes en la misma aplicación.
        + Suministrar una lista de jugadores de forma automática por medio del envío de un archivo .csv que contenga un encabezado y 3 columnas (nombre jugador, apellido de jugador y posición). El nombre de las columnas de encabezado es indiferente para el programa.
        + Editar equipos registrados. El usuario contará con la capacidad de cambiar los valores de los jugadores asignados a un equipo, así como la capacidad de añadir o eliminar algún miembro.

**Iniciar.** Es la vista que permite registrar las jugadas de un partido de fútbol sucediendo en vivo. Guarda semejanza con la vista de subir archivo y se diferencia en el hecho de que las jugadas se registran en base a un cronómetro que aparece en la aplicación.

**Subir archivo.** Es la vista que permite registrar las jugadas de un partido de fútbol sucedido anteriormente. El flujo que describe es la elección de equipos, el llenar las características solicitadas del juego y la creación manual o automática de las jugadas. Al final cuenta con una vista de resumen de lo creado en la vista. De las funcionalidades descritas más complejas son las siguientes y por eso las definimos más a fondo:

* + - * + Suministrar una lista de jugadas de forma manual a través de la digitación de los datos pertinentes en la misma aplicación.
        + Suministrar una lista de jugadas de forma automática por medio del envío de un archivo .csv que contenga un encabezado y 3 columnas (nombre de jugada, nombre de jugador y equipo). El nombre de las columnas de encabezado es indiferente para el programa.

**Histórico.** Ver y consultar información de torneos anteriores. Permite como funcionalidad la edición de campos. Sin embargo, los mismos no se pueden quedar vacíos.

## **Modelos**

Los modelos describen las entidades con las que interactúa el programa. Más explicación de los mismos se pueden encontrar en el diagrama de clases que también se encuentra en la misma carpeta de este documento. Las entidades son:

Juego. Define cada juego o partido y se compone de las siguientes características

* Código. Es una cadena de caracteres que se forma con la siguiente estructura, nos sirve de identificador único.

[Fecha]->[Hora de inicio]->[Equipo A] VS. [Equipo B]

* Título. Es una cadena de caracteres que se forma con la siguiente estructura, nos sirve como identificador para el usuario.

[Equipo A] VS. [Equipo B]

* Equipo A. Es una entidad de tipo equipo que refiere a uno de los equipos que tomó parte en el juego.
* Equipo B. Es una entidad de tipo equipo que refiere a uno de los equipos que tomó parte en el juego.
* Estadio. Es una cadena de caracteres.
* Ciudad. Es una cadena de caracteres.
* Fecha. Es una cadena de caracteres que se forma con la fecha y tiempo en el que se hizo el registro o la seleccionada por el usuario.
* Árbitro. Es una cadena de caracteres.
* Hora de inicio. Es una cadena de caracteres que se forma con las horas, minutos y segundos especificadas por el usuario.
* Hora de fin. Es una cadena de caracteres que se forma con las horas, minutos y segundos especificadas por el usuario.
* Puntuación A. Es un dato entero.
* Puntuación B. Es un dato entero.
* Jugadas. Es un arreglo de entidades tipo jugada.

Jugador

* Nombre. Es una cadena de caracteres.
* Apellido. Es una cadena de caracteres.
* Posición. Es una cadena de caracteres.
* Equipo. Es una entidad tipo Equipo.

Jugada

* Nombre. Es una cadena de caracteres.
* Juego. Es una entidad tipo Juego.
* Jugador. Es una entidad tipo Jugador.
* Equipo. Es una entidad tipo Equipo.

Equipo

* Nombre. Es una cadena de caracteres.
* Jugadores. Es un arreglo de entidades tipo jugador

## **Controladores**

Cada vista descrita posee un controlador, en el cual se encuentran separados en funciones todas las acciones descritas para realizar por cada vista.

Directamente en el paquete controlador se encuentran las funciones aprovechadas de forma común por todas las vistas. Entre estos tenemos los siguientes:

* AbrirVistas. Nos permite abrir las diferentes vistas, desde los diferentes tipos de vistas, en los formatos de Exit para salir o Dispose para salir.
* Cronometro. Es la clase que controla el cronómetro.
* FiltroDeArchivos. Es la clase que delimita que los archivos a usar son tipo csv
* WindowCloseListener. Un listener que nos permite activar el modo Dispose para salir y realizar las tareas que deseamos una vez se sale de esta forma, tareas tales como recargar campos para que se vea el cambio efectuado.
* dataArreglos. Es la clase que interactúa con los arreglos en los que se guarda la información de los archivos csv, creados por la aplicación, y los devuelve en forma de listas de string, para su fácil adecuación a los elementos de las vistas.
* onlyDigitsListener. Es la clase que limita ciertos campos a recibir sólo dígitos.
* onlyLettersListener. Es la clase que limita ciertos campos a recibir sólo caracteres alfabéticos.

En este paquete o carpeta se definen otros directorios. Estos son:

**ArchivoController**

Posee cuatro clases para interactuar con los archivos csv que el programa crea en su ejecución y que le permiten guardar de forma permanente los datos registrados. Estas son:

* Archivos. Hace uso del patrón de diseño SingleTon, que permite una única instancia y esta instancia se realiza desde el archivo main principal de la aplicación. Su función es llamar los métodos de la clase CrearArchivos, en el orden y forma correctos.
* CrearArchivos. Es una clase que posee el conjunto de métodos que permite la creación de los 4 archivos csv que se usan (juego, jugador, equipo y jugada), así como la carpeta que los contiene, cuyo nombre es “archivos”.
* CargarArchivos. Es la clase que alimenta los arreglos con la información guardada en ejecuciones anteriores de la aplicación. Existe con el fin de tener de forma local la información, para evitar la consulta y escritura constante hacia los archivos y con esto reducir el desempeño de la aplicación.
* EscribirArchivos. Es la clase que redacta los cambios definitivos en los archivos csv creados.

**EquipoRegistroController**

* ComprobarCampos. Es la clase designada para comprobar los campos solicitados en el registro de equipos. Realiza dos comprobaciones y estas son, verificar que se haya creado al menos 1 jugador y que los 3 campos de dicho jugador han sido completados.
* EquipoRegistroController. Es la clase que le da vida a todos los botones que aparecen en la vista. En esta vista los únicos dos botones que aparecen son el botón guardar y cancelar.

**EquiposController**

* ComprobarCampos. Es la clase designada para comprobar los campos solicitados en la vista de equipos. Las comprobaciones que realiza son:

Exista un nombre de equipo cuando se registre de forma automática, es decir con la subida de un archivo.

Que al editar un jugador no se quede ningún campo en blanco.

* EquiposController. Es la clase que le da vida a todos los botones que aparecen en la vista.

Confirma la aprobación del usuario al borrar algún jugador.

Usa un menú desplegable para que el usuario seleccione un jugador y se cargue en la vista su información de forma automática.

Habilita una vista adicional para capturar los datos de los jugadores al editarlos o agregarlos.

Guardar de forma definitiva en el archivo equipos los cambios ingresados manualmente o con la subida de un csv.

**HistoricoController**

* HistoricoController. Es la clase que le da vida a todos los botones que aparecen en la vista.

Habilita una vista adicional para capturar los datos a editar

Usa un menú desplegable para que el usuario seleccione un juego y se cargue en la vista de forma automática.

**SubirController**

* SubirController. Es la clase que se encarga de preparar y refrescar la información para que los diferentes tabs que componen esta vista estén habilitados para trabajar.
* TabEquipos. Da vida a los menús desplegables desde los cuales se seleccionan los dos equipos a jugar. Comprueba que no siempre se escoja 2 equipos diferentes. Conecta al botón añadir equipo con la vista de equipos.
* TabJuegos. Muestra en una tabla los juegos registrados entre los dos equipos seleccionados y captura los datos solicitados al usuario propios del juego y los escribe de forma local y temporal al hacer click en el botón siguiente. Además, valida que ningún campo quede vacío.
* TabJugadas. Captura los datos solicitados al usuario propios de la jugada y los escribe de forma local y temporal al hacer click en el botón siguiente. Además, valida que ningún campo quede vacío y muestra en una lista la secuencia de jugadas para el usuario.
* TabGuardar. Resume el juego, da a conocer la puntuación y estado final del juego y al dar click al botón terminar guarda de forma definitiva en el archivo Juegos y Jugadas, los cambios almacenados temporalmente

**IniciarController**

* IniciarController. Es la clase que se encarga de preparar y refrescar la información para que los diferentes tabs que componen esta vista estén habilitados para trabajar.
* TabEquipos. Da vida a los menús desplegables desde los cuales se seleccionan los dos equipos a jugar. Comprueba que no siempre se escoja 2 equipos diferentes. Conecta al botón añadir equipo con la vista de equipos.
* TabJuegos. Muestra en una tabla los juegos registrados entre los dos equipos seleccionados y captura los datos solicitados al usuario propios del juego y los escribe de forma local y temporal al hacer click en el botón siguiente. Además, valida que ningún campo quede vacío.
* TabJugadas. Interactúa con la clase cronómetro para capturar las jugadas, junto con los datos solicitados al usuario propios de la jugada. Esto lo escribe de forma local y temporal al hacer click en el botón siguiente. Además, valida que ningún campo quede vacío y muestra en una lista la secuencia de jugadas para el usuario. Está diseñado de manera secuencial y poka-joke para evitar que se avancen a etapas sin rellenar campos anteriores.
* TabGuardar. Resume el juego, da a conocer la puntuación y estado final del juego y al dar click al botón terminar guarda de forma definitiva en el archivo Juegos y Jugadas, los cambios almacenados temporalmente

## **Patrón de diseño**

En el proyecto hemos hecho uso de algunos patrones de diseños. Si no se está familiarizado con el concepto, entiéndase por patrón de diseño la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Un patrón de diseño resulta ser una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

El patrón usado en el proyecto es el SingleTon. Definido para la clase archivos y el mismo nos permite garantizar la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. Restringe la instanciación de una clase o valor de un tipo a un solo objeto.

## **Repositorio en GitHub**

Siéntete libre de clonar o copiar nuestro repositorio para que juntos hagamos un mejor FootballBoard:

https://github.com/m4b3l/FootballBoard