2019/2021

Proyecto Final

B.C.F. Application

Javier Pérez Alonso
G.S.DAM

Contenido de la Memoria del Proyecto Final

Introducción

Descripción y finalidad

Proyecto desarrollado por Javier Pérez Alonso en su totalidad. BCF Application es un software dedicado a gestionar el sistema de porras de una peña de socios del Burgos Club de Fútbol.

La aplicación permite las siguientes funcionalidades:

- Gestionar los participantes.
- Gestionar los equipos de fútbol rivales del Burgos C.F. en la temporada regular.
- Gestionar los partidos.
- Realizar apuestas personalizadas de cada usuario sobre un partido en concreto.
- Ver la tabla de clasificación.
- Enviar notificaciones vía correo electrónico a todos los participantes.
- Contactar con soporte de una manera sencilla y rápida ante posibles problemas con la aplicación.
- Agregar documentación actualizada con las reglas a seguir por los participantes y el sistema de puntuación.
- Gestionar el inicio y fin de cada temporada.

Actualmente la aplicación consta de dos versiones, una aplicación pura de escritorio y otra que amplía las posibilidades de la misma gracias a la separación de su backend del resto de la aplicación, lo que en el mundo del desarrollo se denominan clientes.

Este “back” es una Full Rest API que encapsula toda la lógica de negocio y ofrece recursos representados por una URI, accediendo a estos mediante el protocolo HTTP. Las principales ventajas de usar Rest como sistema distribuido son… la gran flexibilidad que proporciona, su interfaz uniforme y el sistema de capas (Clean Arquitecture). También permite ofrecer los recursos en diferentes representaciones (JSON, HTML o XML), en el caso de BCF Application se utiliza JSON como estándar.

La comunicación entre el cliente desktop y la API Rest se resume en la siguiente imagen (Petición/Respuesta).

El propósito del proyecto es automatizar la administración de un sistema de apuestas, facilitando aspectos cómo calcular los puntos de cada jornada, notificar resultados de estos a todos los participantes, mantener los datos guardados para evitar pérdida de información, etc.

Análisis y diseño

Planificación y requisitos

Atendiendo a las necesidades del cliente en cuanto al software se consideran los siguientes requisitos:

- Disponer de un aplicativo funcional lo antes posible. Como mínimo un cliente desktop para poder administrar toda la aplicación.

- Disponer de una versión nueva de la aplicación que permita cambiar de cliente desktop a web o mobile según se requiera en un futuro.

- Desarrollar un cliente con una interfaz amigable, moderna y con un diseño “sencillo” a la vista. Utilizar colores relacionados al BCF o al entorno futbolístico.

En cuanto a funcionalidades que debe ofrecer el producto…

Soporte:

- Enviar mail a soporte con los últimos logs de la aplicación para detectar posibles bugs o funcionamientos no esperados de una forma ágil.

- Permitir visualizar la licencia actual del producto.

Reglas:

- Reglas de puntuación por defecto ya establecidas.

- Almacenar los puntos por acertar los goles del Burgos, los puntos por acertar el signo y los puntos por acertar el resultado del partido.

- Actualizar las reglas de puntuación, tanto con nuevos valores como regresar a los valores por defecto.

- Validar que los puntos/resultado son mayores a los puntos/goles del Burgos C.F. y estos a su vez mayores a puntos/signo. Así mismo deben mantenerse entre un rango de valores, un mínimo de 0 puntos para los puntos por signo.

- Almacenar un documento PDF con las reglas generales de participación y puntuación.

- Permitir la actualización del documento con las reglas.

- Validar que se trata de un documento PDF.

- Obtener los datos de las reglas actuales.

Equipos:

- Almacenar Burgos C.F. como equipo constante en cada temporada.
- Añadir, actualizar o borrar los diferentes equipos que se va a encontrar el B.C.F. durante la temporada.
- Validar que el equipo añadido no se repite y que al menos tiene un mínimo de tres caracteres para identificarlo y un máximo de 60.

Participantes:

- Añadir, actualizar o borrar usuarios/participantes.
- Almacenar el nick y el correo electrónico de cada participante, así como sus puntos totales.
- Obtener los datos de todos los participantes.
- Validar los datos guardados de cada participante. El nick debe tener un mínimo de un carácter y un máximo de veinte. El correo electrónico debe corresponderse con un formato válido siguiendo el patrón #@#.# .
- Visualizar todos los datos de cada participante en cualquier momento.
- Enviar un correo dando la bienvenida a un participante añadido incluyendo el documento con las reglas generales.

Partidos:

- Crear un partido entre el BCF y otro equipo de los que se encuentren guardados.
- Permitir un solo partido abierto para aceptar apuestas.
- Borrar el partido abierto y todas las apuestas sobre el mismo.
- Finalizar el partido y calcular los puntos de todos los participantes.
- Almacenar el nombre del equipo local y del equipo visitante, con las mismas restricciones existentes al guardar cada equipo, el resultado del partido y la lista de apuestas de todos los jugadores para ese partido.
- Validar que el partido es único, que uno de los dos equipos es el BCF y el otro equipo sea diferente.
- Validar que el resultado del partido sigue el patrón # - #, donde el número de goles debe ser un entero positivo y con rango máximo igual a veinte.
- Obtener los datos de todos los partidos guardados.

Apuestas:

- Crear una apuesta sobre un partido en concreto por cada jugador.
- Almacenar apuestas guardando una asociación con el jugador y el partido relacionados a las mismas.
- Almacenar el resultado elegido, el signo ganador, los goles del BCF y los puntos obtenidos al finalizar el partido.
- Validar que el jugador y el partido asociados a la apuesta existen, el resultado del partido sigue el patrón # - #, donde el número de goles debe ser un entero positivo y con rango máximo igual a veinte.
- Obtener todas las apuestas relacionadas a un partido en concreto.

Tablas clasificatorias:

- Visualizar tabla de clasificación actualizada en cualquier momento, permitiendo ordenarla por puntos o alfabéticamente. De inicio mostrar los jugadores según el orden en el que fueron añadidos.
- Visualizar tabla de la jornada abierta con las apuestas actuales, así como los puntos obtenidos al finalizar la misma.

Gestión de la Temporada:

- Permitir reiniciar la temporada eliminando todos los datos no esenciales.

Notificaciones relacionadas a eventos:

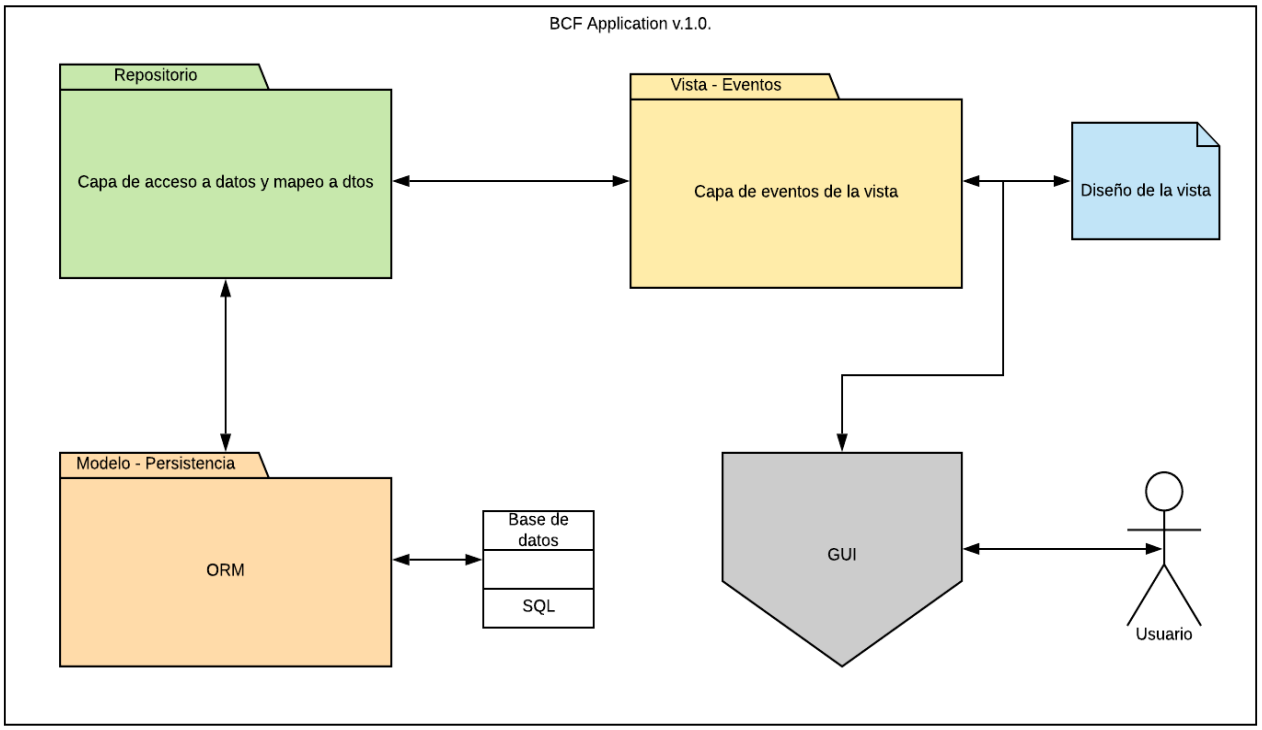
- Enviar notificación vía correo electrónico a un participante cuando es registrado en el sistema adjuntando el pdf con las reglas actuales.
- Enviar notificación vía correo electrónico a todos los participantes cuando se abre una nueva jornada.
- Enviar notificación vía correo electrónico a todos los participantes cuando termina una jornada adjuntando una tabla con la clasificación actual ordenada por puntos totales y mostrando los puntos obtenidos en esa jornada.
- Enviar notificación vía correo electrónico a todos los participantes cuando se actualizan las reglas.
- Enviar notificación vía correo electrónico a todos los participantes cuando se reinicia la temporada incluyendo al ganador de la misma de haberlo.

Desarrollo

Diseño

- Versión 1.0. Aplicación desktop pura.

El software en su primera versión conforma un monolito en el que se encuentra encapsulado tanto el modelo de dominio, la persistencia y acceso de los datos, como la vista, dentro del mismo proyecto siguiendo el siguiente esquema.



El funcionamiento de la aplicación se inicia con una ventana de carga con una imagen de fondo que representa tanto al BCF como todos sus valores. Una vez carga la aplicación se muestra una segunda ventana que permite la interacción total con el usuario, la ventana principal.

Esta ventana dispone de dos botones visibles y un menú-hamburguesa a partir del cual el usuario puede navegar por toda la aplicación. Uno de los botones para apagar la aplicación, no sin antes preguntar primero, el otro muestra un panel con una imagen o texto de ayuda para principiantes.

El menú-hamburguesa presenta diferentes botones para abrir los paneles de gestión de participantes, equipos, tablas, reglas, etc.
Estos paneles ofrecen todas las operaciones necesarias para cumplir con los requisitos.

El usuario inicia eventos, estos se comunican con los repositorios enviando o recibiendo datos encapsulados en DTOs (Data Transfer Objects), ya que es buena práctica no exponer nunca las entidades de dominio, en los repositorios se transforman o se obtienen como entidades a través de un ORM (Mapeo Objeto Relacional) contra la base de datos embebida (SQL).

- Versión 2.0. Aplicación Full REST API + Cliente desktop.

En su segunda versión el software se vuelve más complejo con el fin de facilitar la escalabilidad del proyecto, así como aumentar las posibilidades en cuanto a clientes y separar responsabilidades. La parte visual y la lógica de negocio se separan. La vista se mantiene como en la primera versión sumando nuevas funcionalidades y mejoras en el diseño, nuevos modales, eventos, paneles, …

El backend es una API Rest que expone URIs para acceder a los recursos o funcionalidades. Para enviar u obtener datos se utiliza el formato Json. La API ofrece distintos métodos para comunicarse:

- **GET**: solicita un recurso al servidor.
- **POST**: envía información para crear un nuevo recurso.
- **PUT**: actualiza un recurso de forma completa.
- **DELETE**: borra un recurso.

Después de una petición hay una respuesta, por ello, tanto al obtener un recurso como en una operación de borrado se recibe un mensaje. Este mensaje puede contener datos o no, pero siempre se devuelve un código de respuesta.

Los códigos de respuesta a manejar en el servidor son:

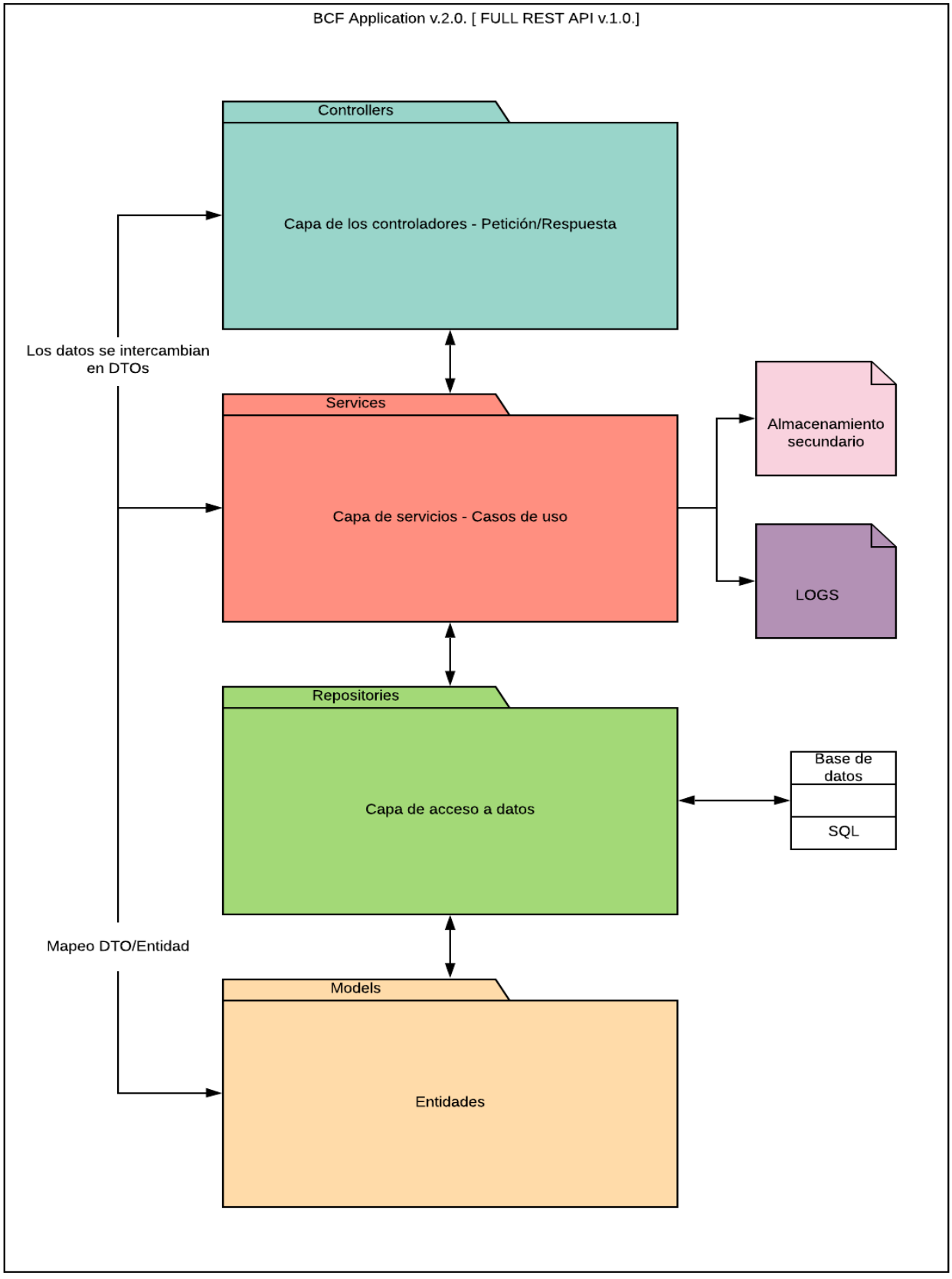
- **200**: comportamiento esperado.
- **400**: request realizada incorrectamente.
- **409**: conflicto con campos únicos.
- **415**: request realizada incorrectamente, media-type no soportado.
- **500**: internal server error.

La Api se forma por capas bien diferenciadas siguiendo una arquitectura limpia, documentada adecuadamente con Swagger para facilitar el desarrollo a posteriori, sobretodo de aumentar plantilla. Nunca se exponen las entidades de dominio. Para el acceso a los datos y mapeo a objetos se utiliza un ORM y en el diseño de software se hace uso de diferentes design patterns como Singleton, Factory method, Dependency Injection, Inversion of Control, Repository, etc.

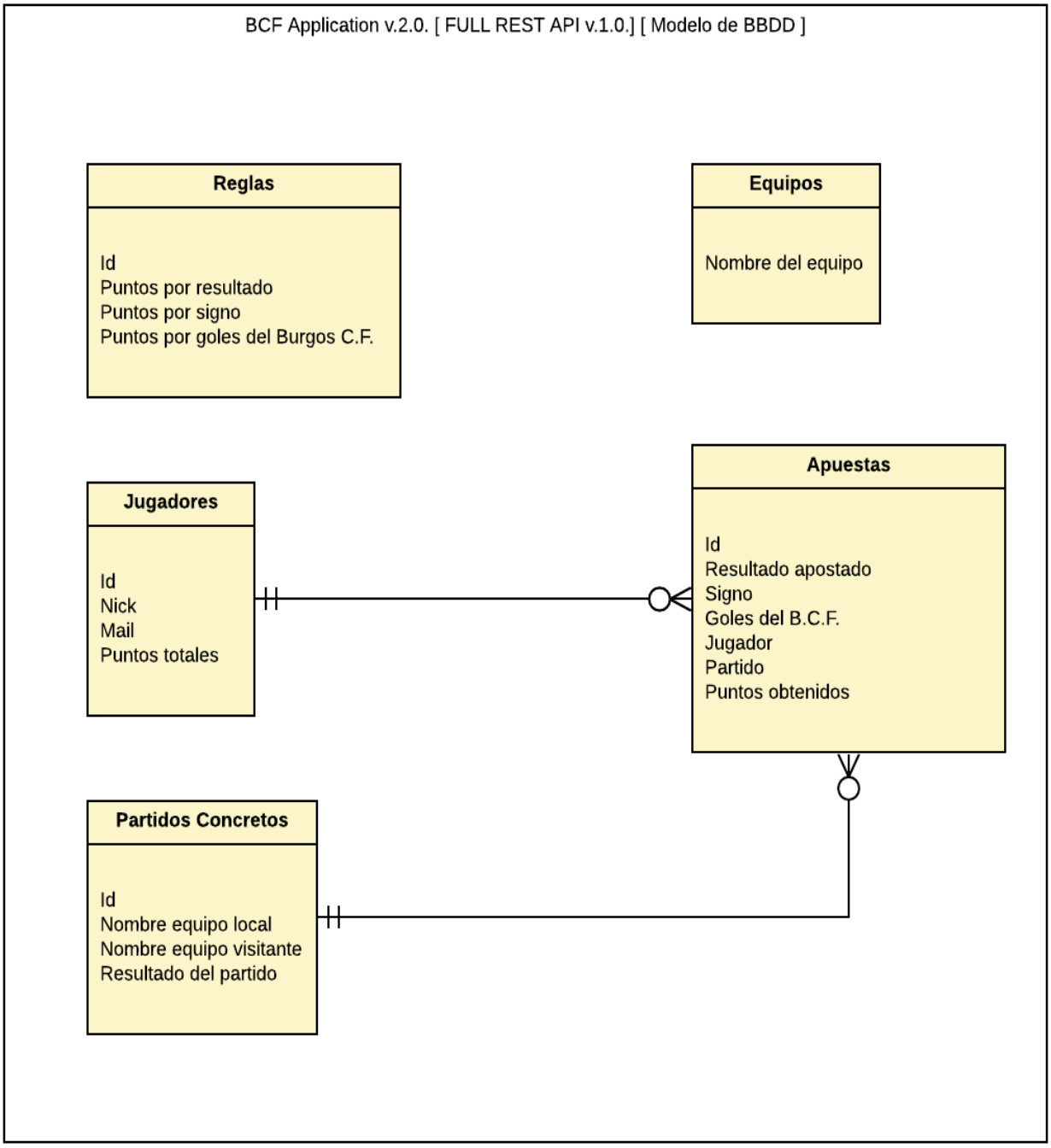
Cuando se hace una petición a la API lo primero que ocurre es validar que es una petición válida, igualmente se filtra los Json u ficheros, si se esperan, los métodos que reciben las request se encuentran en el controller.

Los controllers se mantienen lo más limpios de código posible y desconocen totalmente el modelo, ofrecen métodos, pero derivan la tarea en los servicios.
Los services exponen los métodos adecuados para tratar las peticiones y encapsulan la lógica pesada de la API. Estos conocen el modelo y los repositorios con los que interactúan conformando la respuesta sea devolviendo datos, enviando notificaciones, moviendo ficheros, …

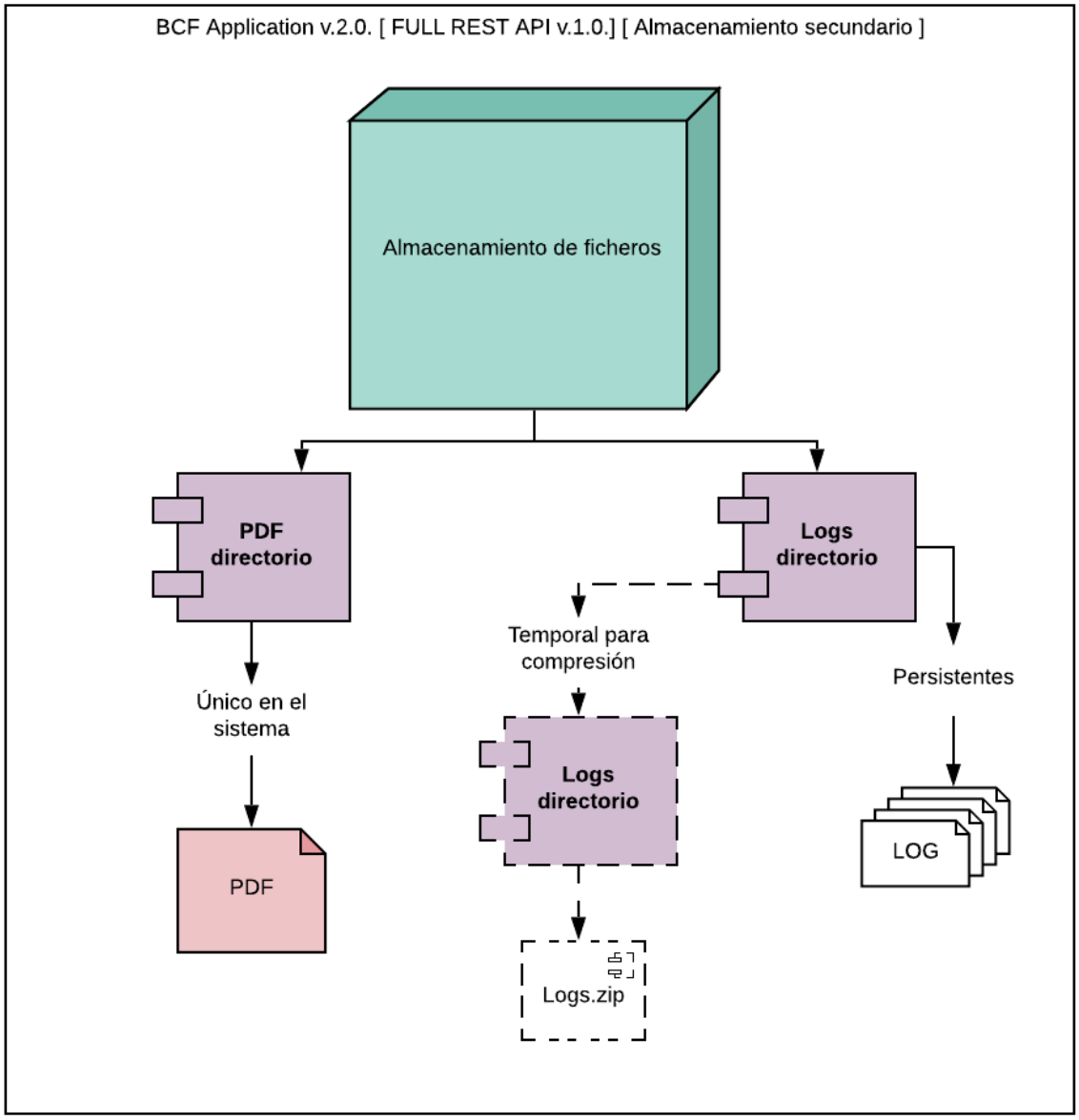
Los siguientes esquemas resumen el diseño y estructura de la API.



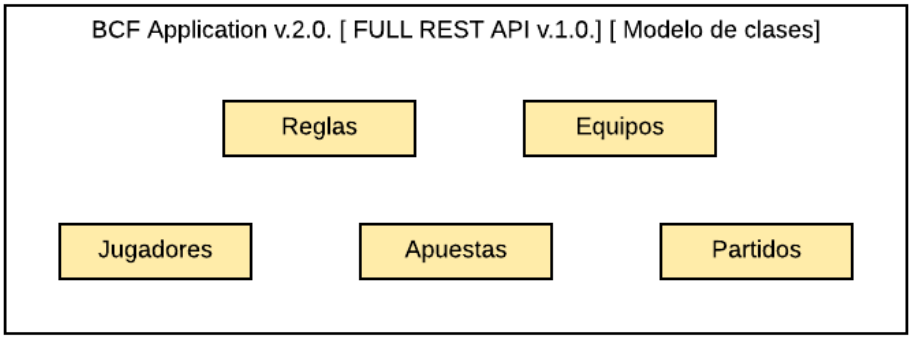
Modelo de Base de datos



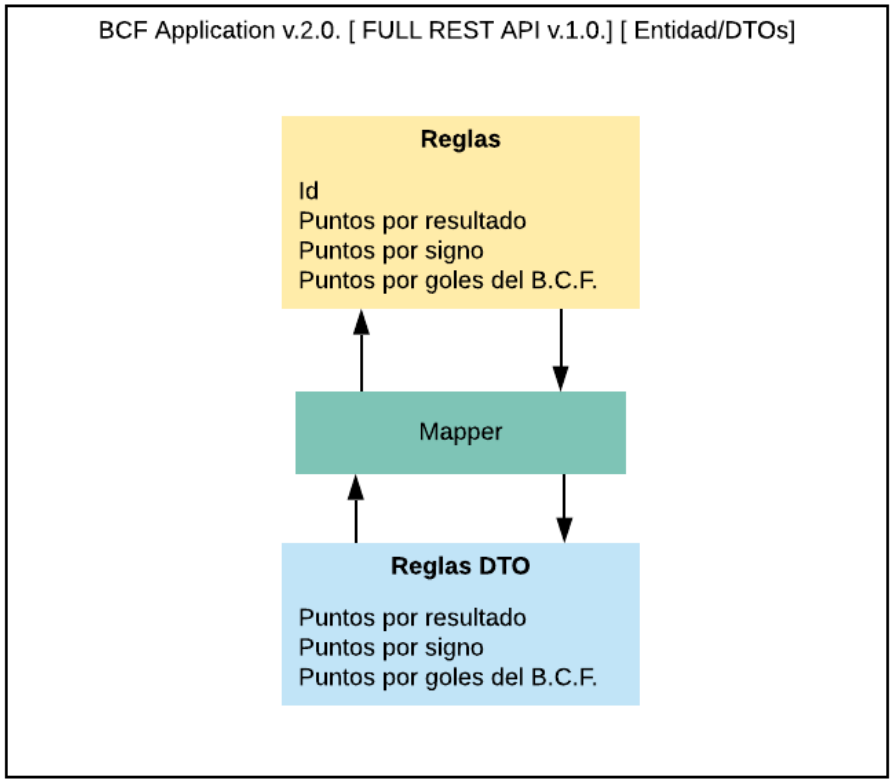
Almacenamiento secundario

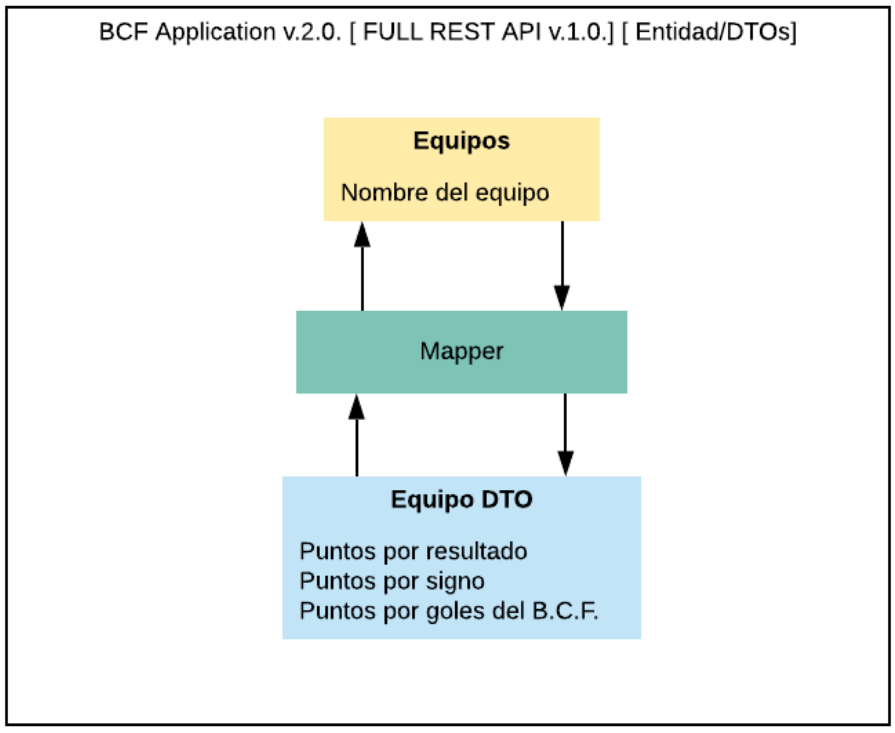


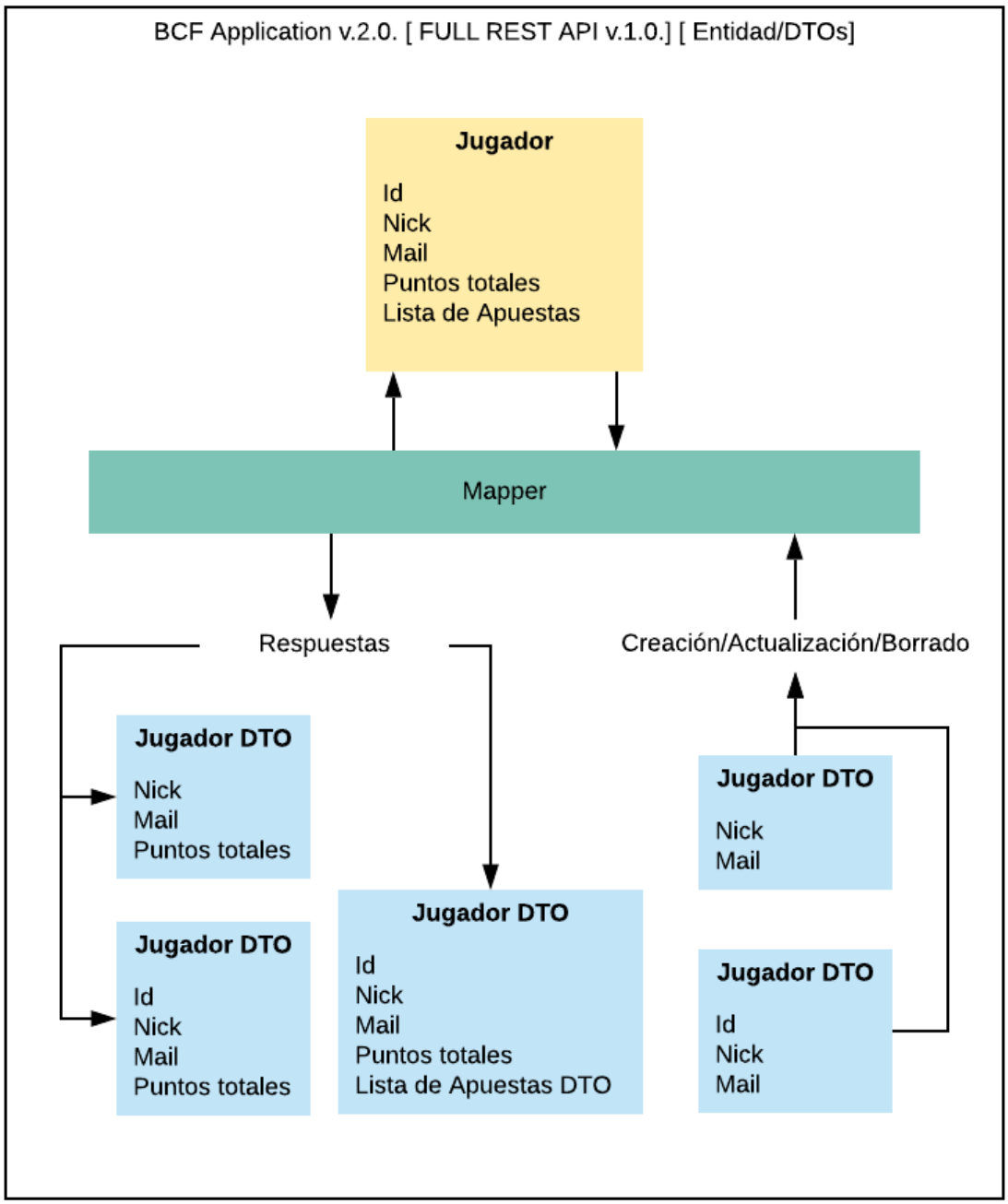
Entidades

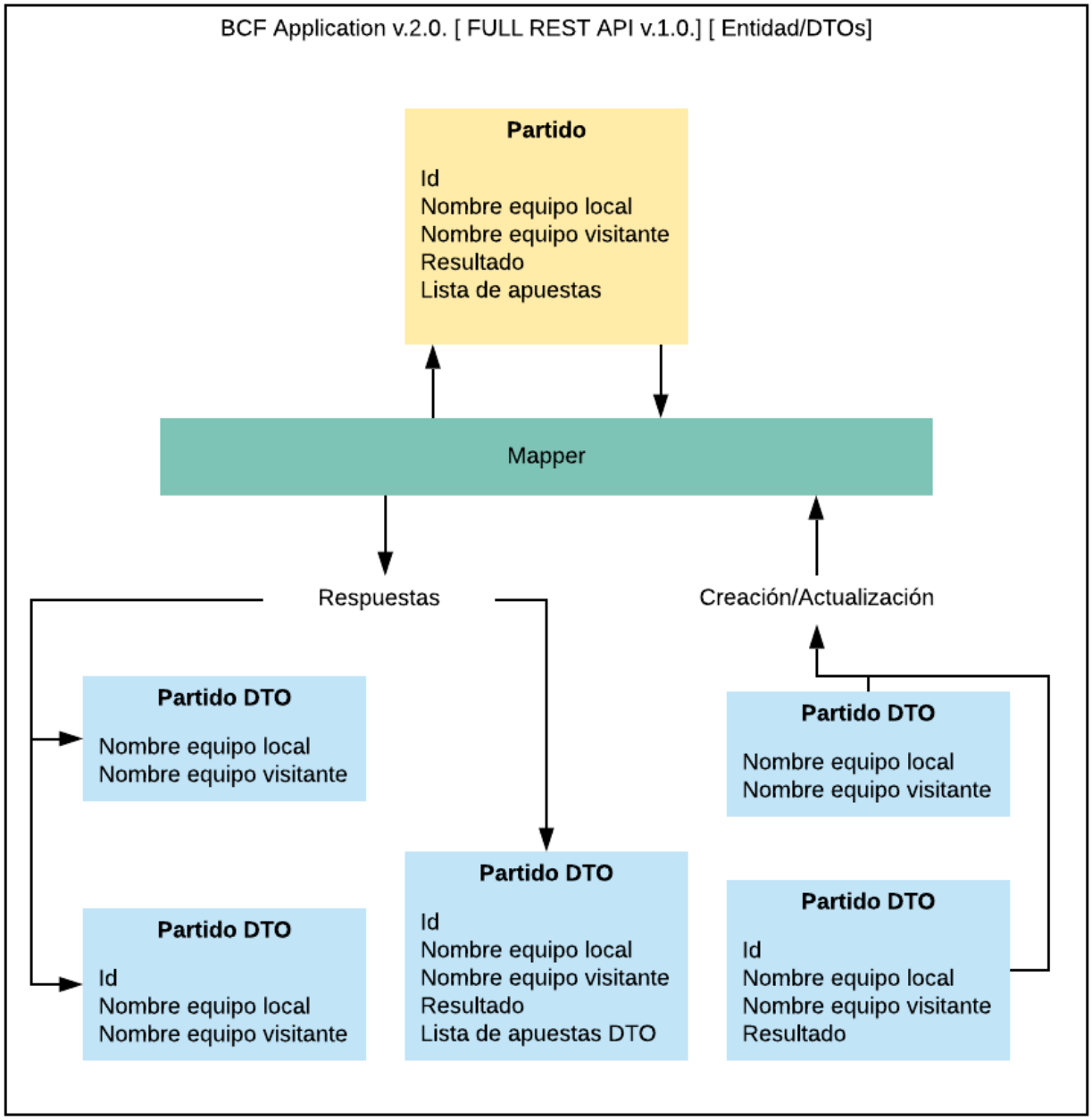


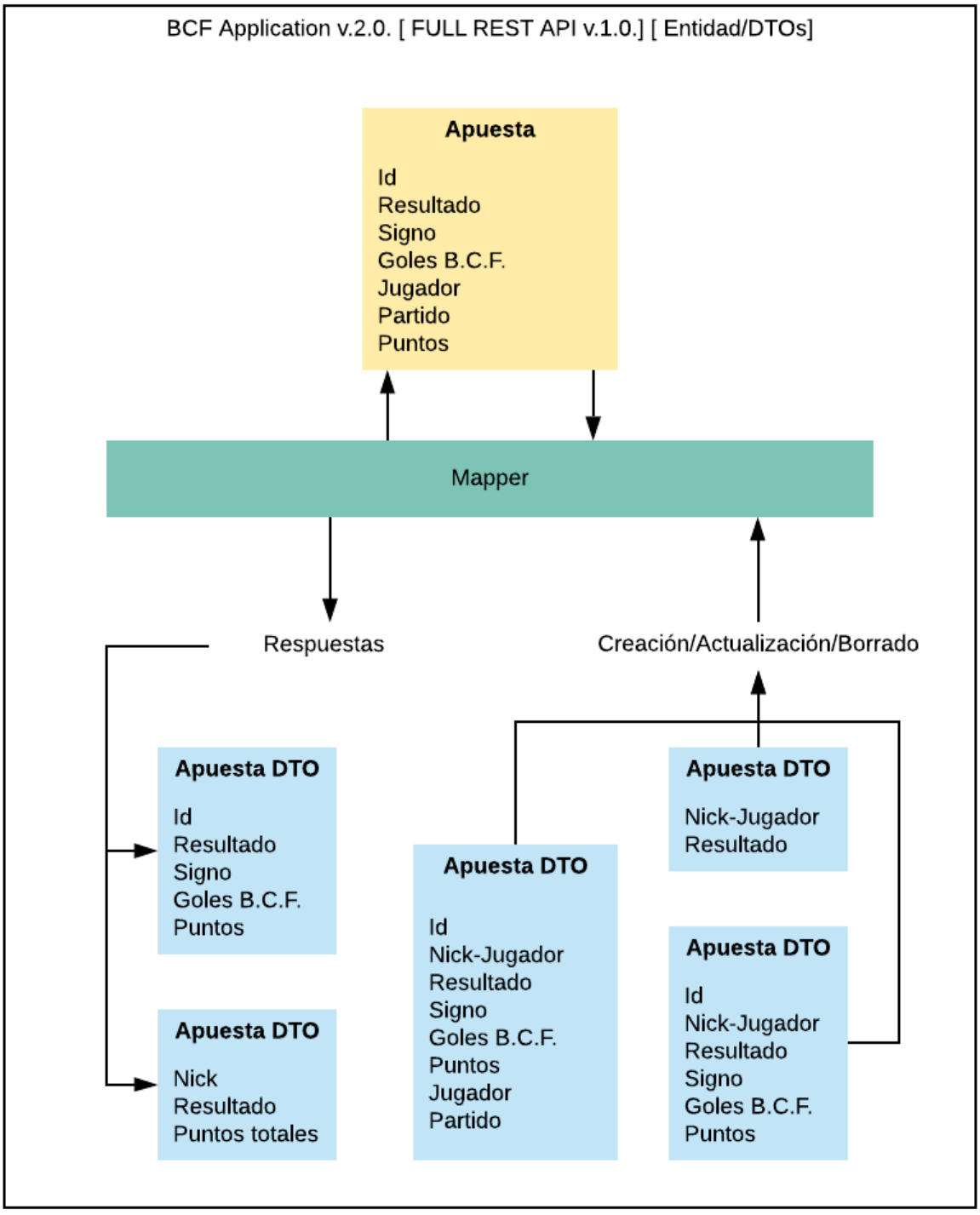
Entidad/DTO


BCF Application v.2.0. [FULL REST API v.1.0.] [Entidad/DTOs]
Reglas
Id
Puntos por resultado
Puntos por signo
Puntos por goles del B.C.F.
Mapper
Reglas DTO
Puntos por resultado
Puntos por signo
Puntos por goles del B.C.F.

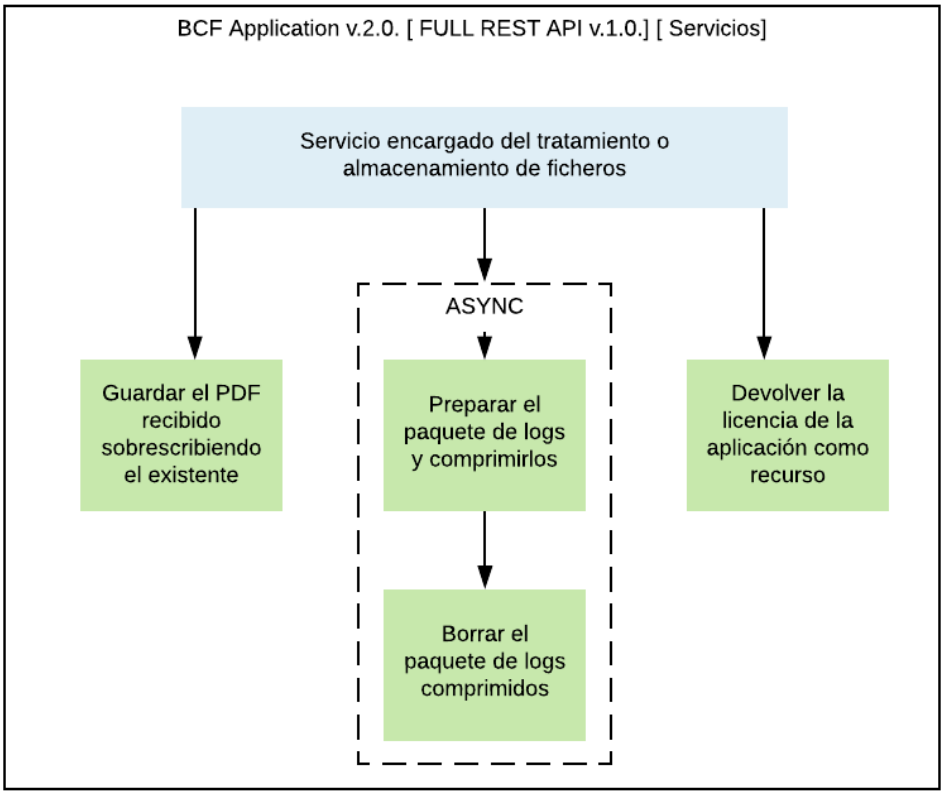

BCF Application v.2.0. [FULL REST API v.1.0.] [Entidad/DTOs]
Equipos
Nombre del equipo
Mapper
Equipo DTO
Puntos por resultado
Puntos por signo
Puntos por goles del B.C.F.

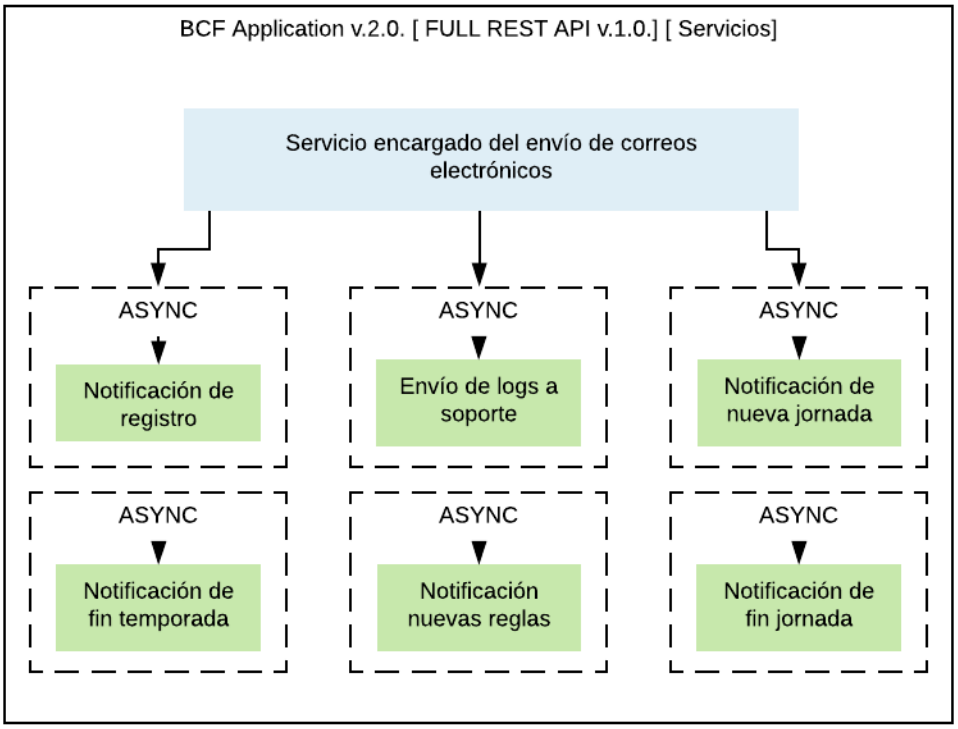

BCF Application v.2.0. [FULL REST API v.1.0.] [Entidad/DTOs]
Jugador
Id
Nick
Mail
Puntos totales
Lista de Apuestas
Mapper
Respuestas
Creación/Actualización/Borrado
Jugador DTO
Nick
Mail
Puntos totales
Jugador DTO
Id
Nick
Mail
Puntos totales
Jugador DTO
Id
Nick
Mail
Puntos totales
Lista de Apuestas DTO
Jugador DTO
Nick
Mail
Jugador DTO
Id
Nick
Mail

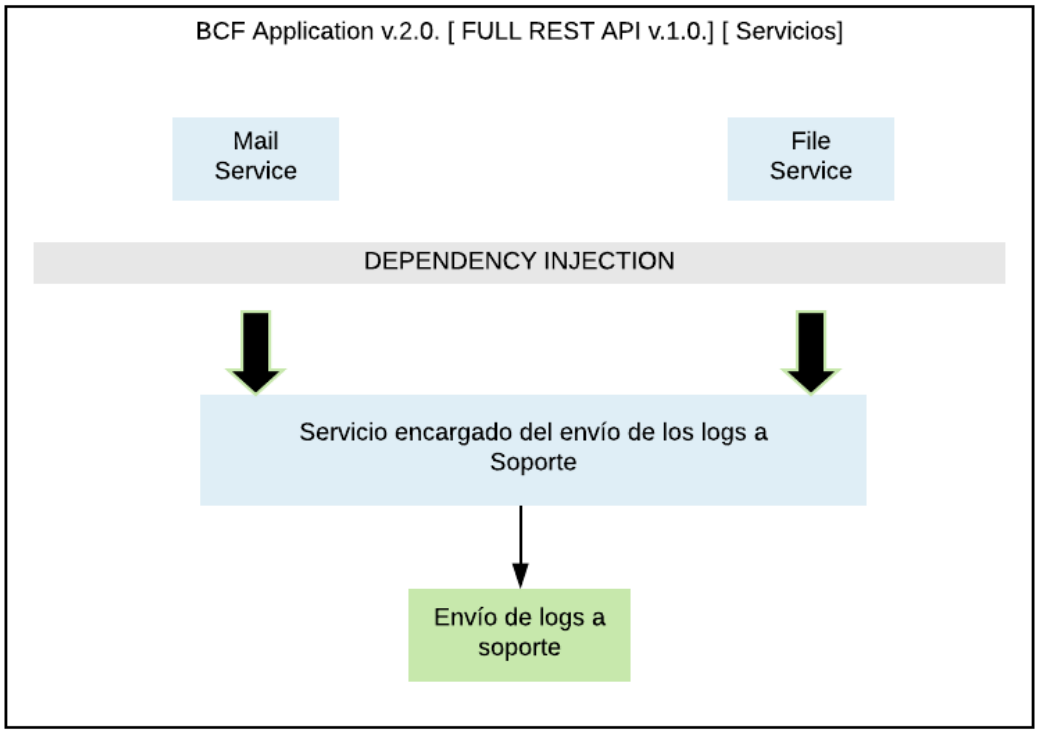

BCF Application v.2.0. [FULL REST API v.1.0.] [Entidad/DTOs]
Partido
Id
Nombre equipo local
Nombre equipo visitante
Resultado
Lista de apuestas
Mapper
Respuestas
Creación/Actualización
Partido DTO
Nombre equipo local
Nombre equipo visitante
Partido DTO
Nombre equipo local
Nombre equipo visitante
Partido DTO
Id
Nombre equipo local
Nombre equipo visitante
Partido DTO
Id
Nombre equipo local
Nombre equipo visitante
Resultado
Lista de apuestas DTO
Partido DTO
Id
Nombre equipo local
Nombre equipo visitante
Resultado

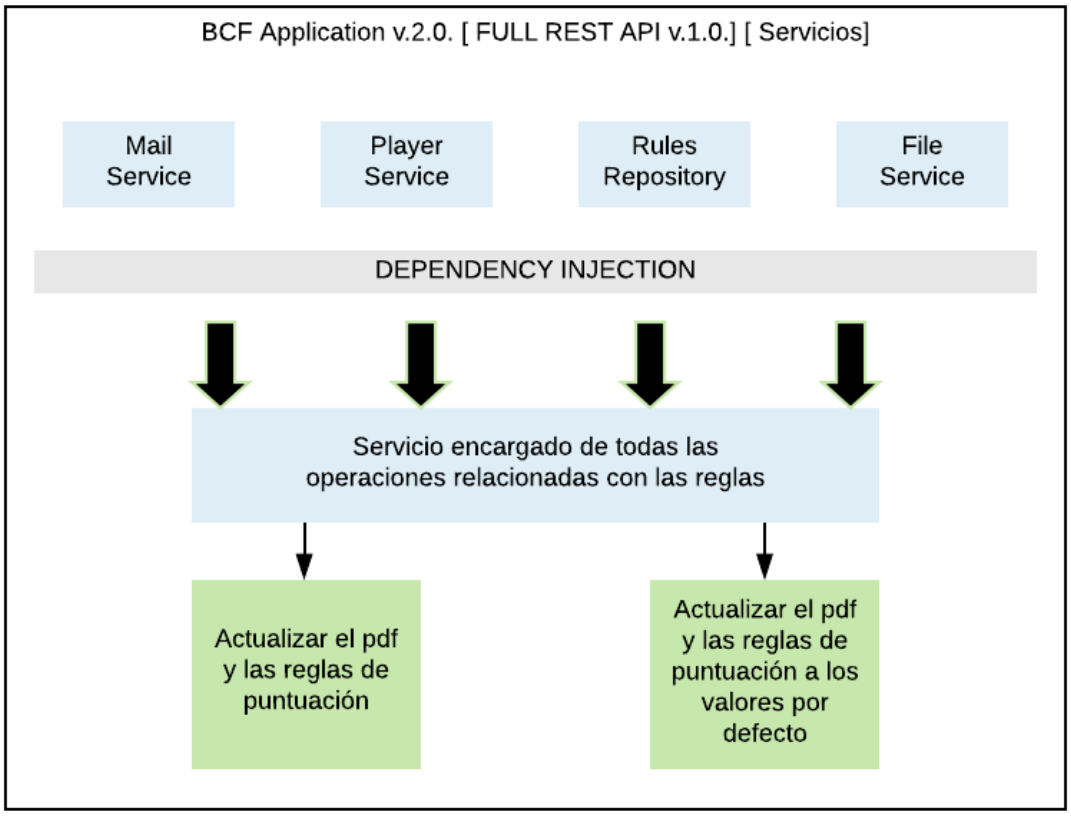

BCF Application v.2.0. [FULL REST API v.1.0.] [Entidad/DTOs]
Apuesta
Id
Resultado
Signo
Goles B.C.F.
Jugador
Partido
Puntos
Mapper
Respuestas
Creación/Actualización/Borrado
Apuesta DTO
Id
Resultado
Signo
Goles B.C.F.
Puntos
Apuesta DTO
Nick
Resultado
Puntos totales
Apuesta DTO
Id
Nick-Jugador
Resultado
Signo
Goles B.C.F.
Puntos
Jugador
Partido
Apuesta DTO
Nick-Jugador
Resultado
Apuesta DTO
Id
Nick-Jugador
Resultado
Signo
Goles B.C.F.
Puntos

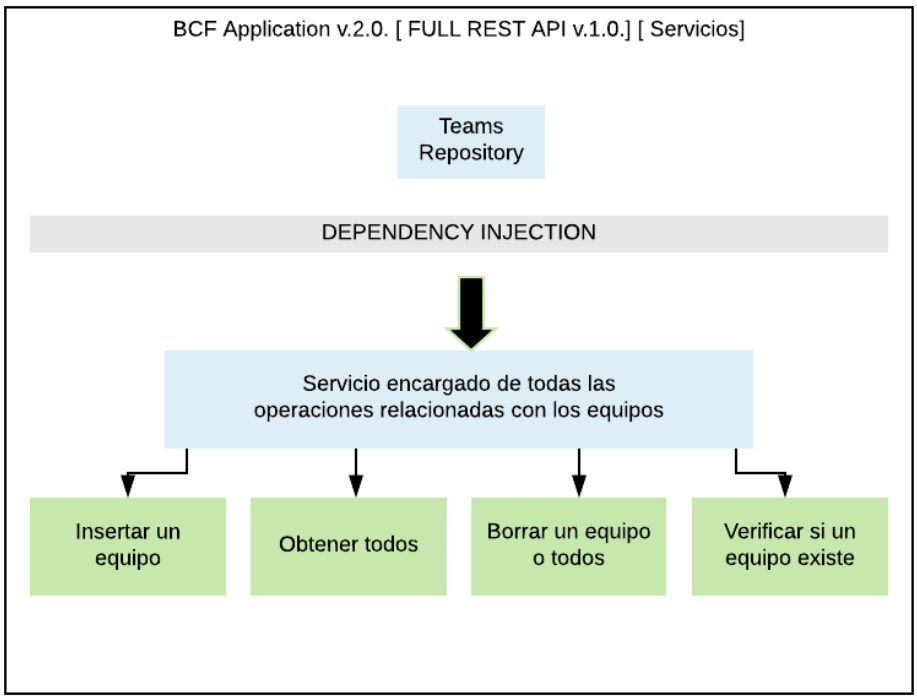
Servicios Implementados/Casos de uso

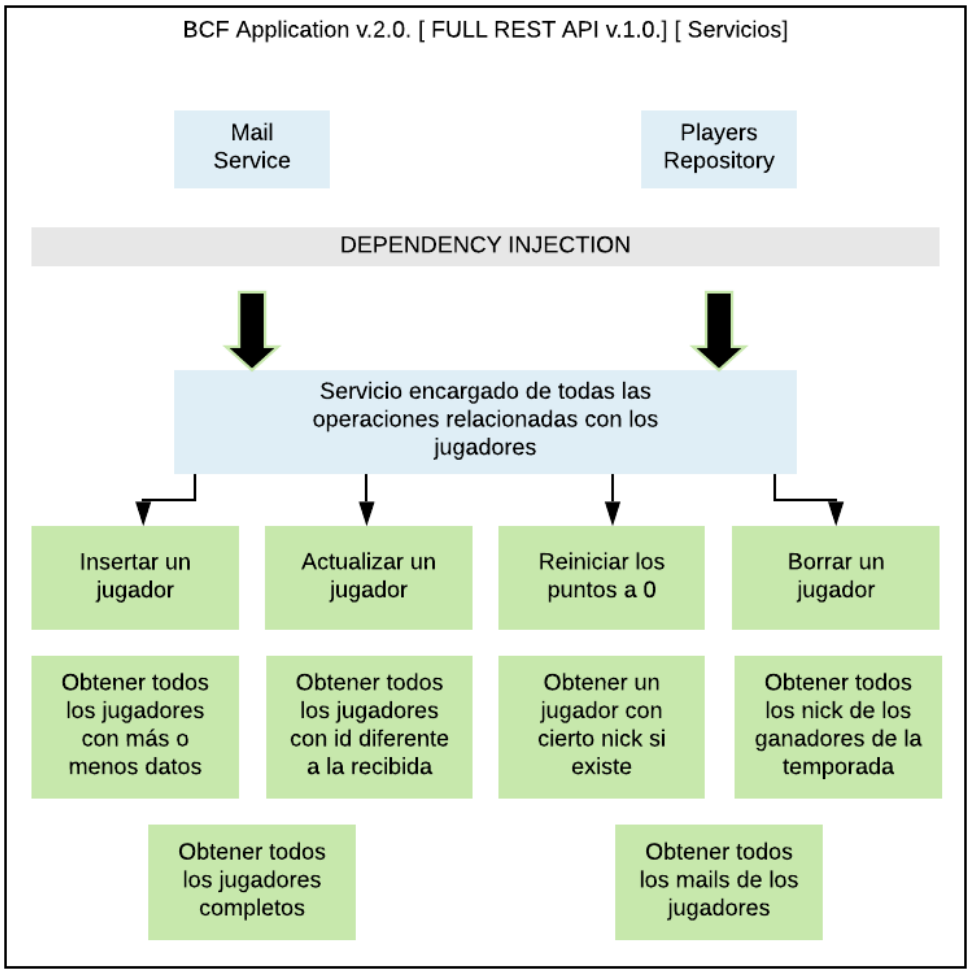


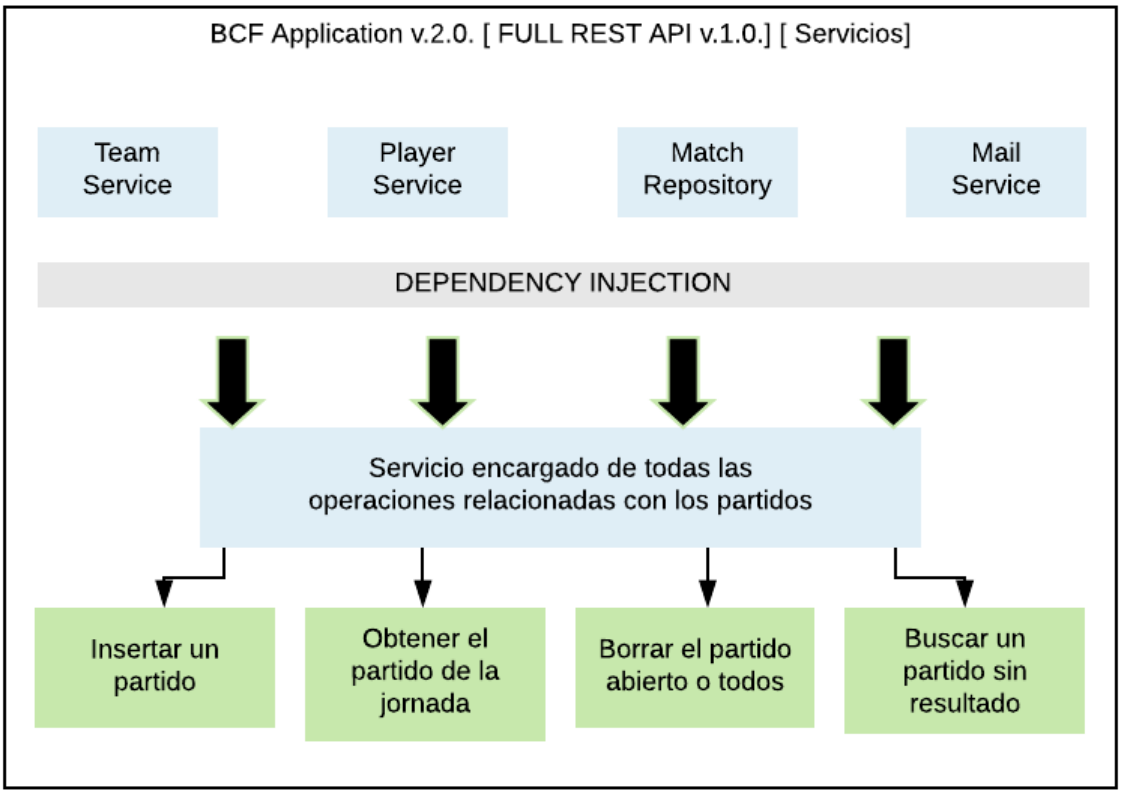


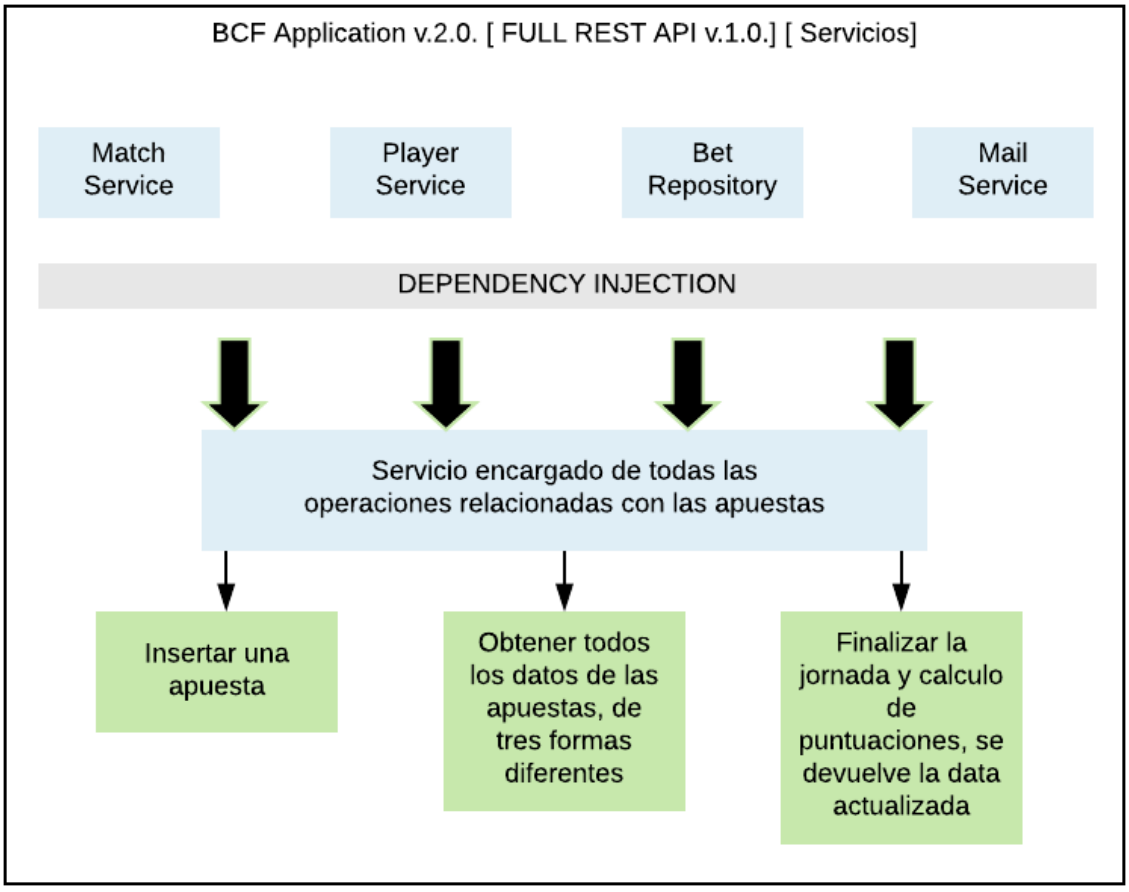

BCF Application v.2.0. [FULL REST API v.1.0.] [Servicios]
Mail Service
File Service
DEPENDENCY INJECTION
Servicio encargado del envío de los logs a Soporte
Envío de logs a soporte

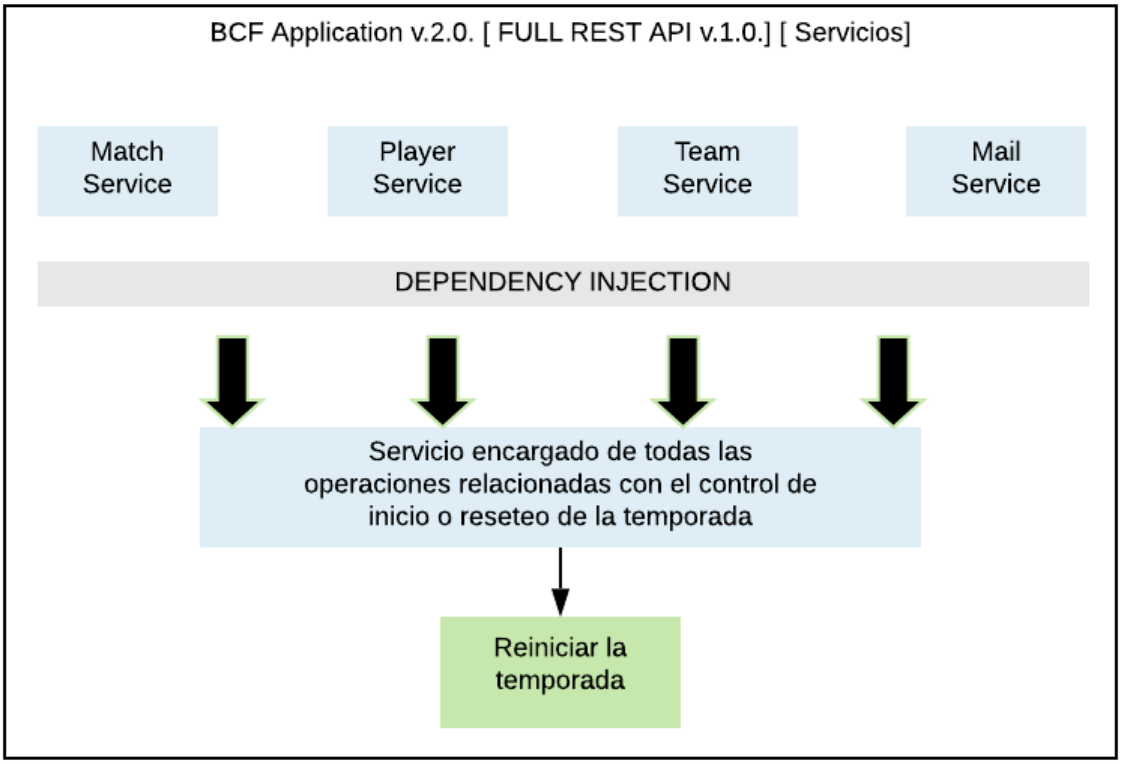

BCF Application v.2.0. [FULL REST API v.1.0.] [Servicios]
Mail Service
Player Service
Rules Repository
File Service
DEPENDENCY INJECTION
Servicio encargado de todas las operaciones relacionadas con las reglas
Actualizar el pdf y las reglas de puntuación
Actualizar el pdf y las reglas de puntuación a los valores por defecto


BCF Application v.2.0. [FULL REST API v.1.0.] [Servicios]
Teams Repository
DEPENDENCY INJECTION
Servicio encargado de todas las operaciones relacionadas con los equipos
Insertar un equipo
Obtener todos
Borrar un equipo o todos
Verificar si un equipo existe

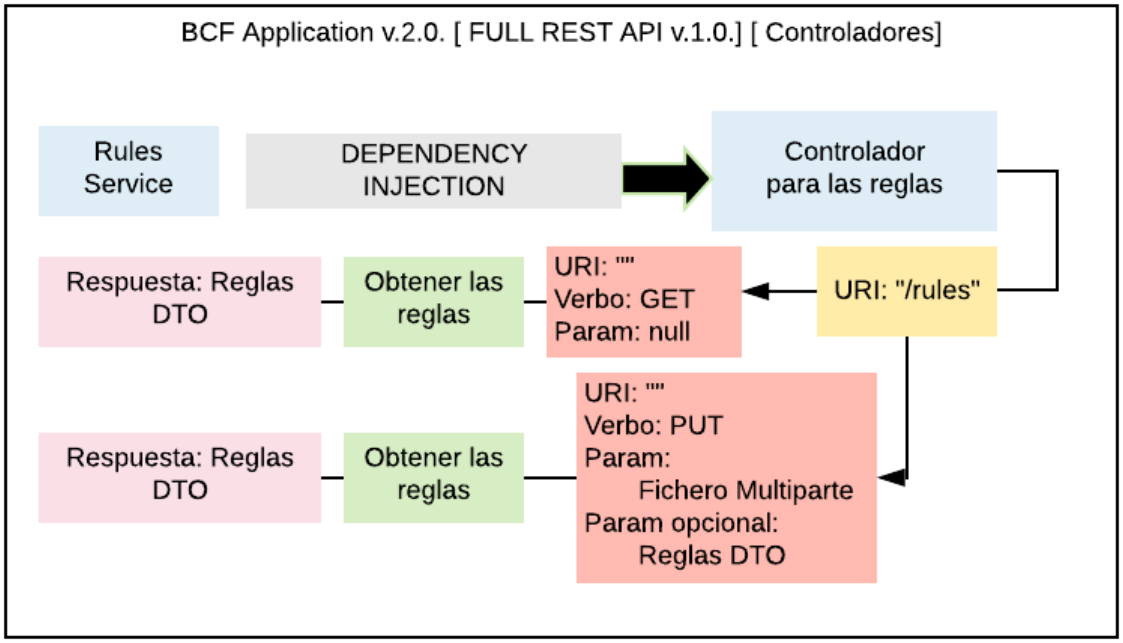

BCF Application v.2.0. [FULL REST API v.1.0.] [Servicios]
Mail Service
Players Repository
DEPENDENCY INJECTION
Servicio encargado de todas las operaciones relacionadas con los jugadores
Insertar un jugador
Actualizar un jugador
Reiniciar los puntos a 0
Borrar un jugador
Obtener todos los jugadores con más o menos datos
Obtener todos los jugadores con id diferente a la recibida
Obtener un jugador con cierto nick si existe
Obtener todos los nick de los ganadores de la temporada
Obtener todos los jugadores completos
Obtener todos los mails de los jugadores

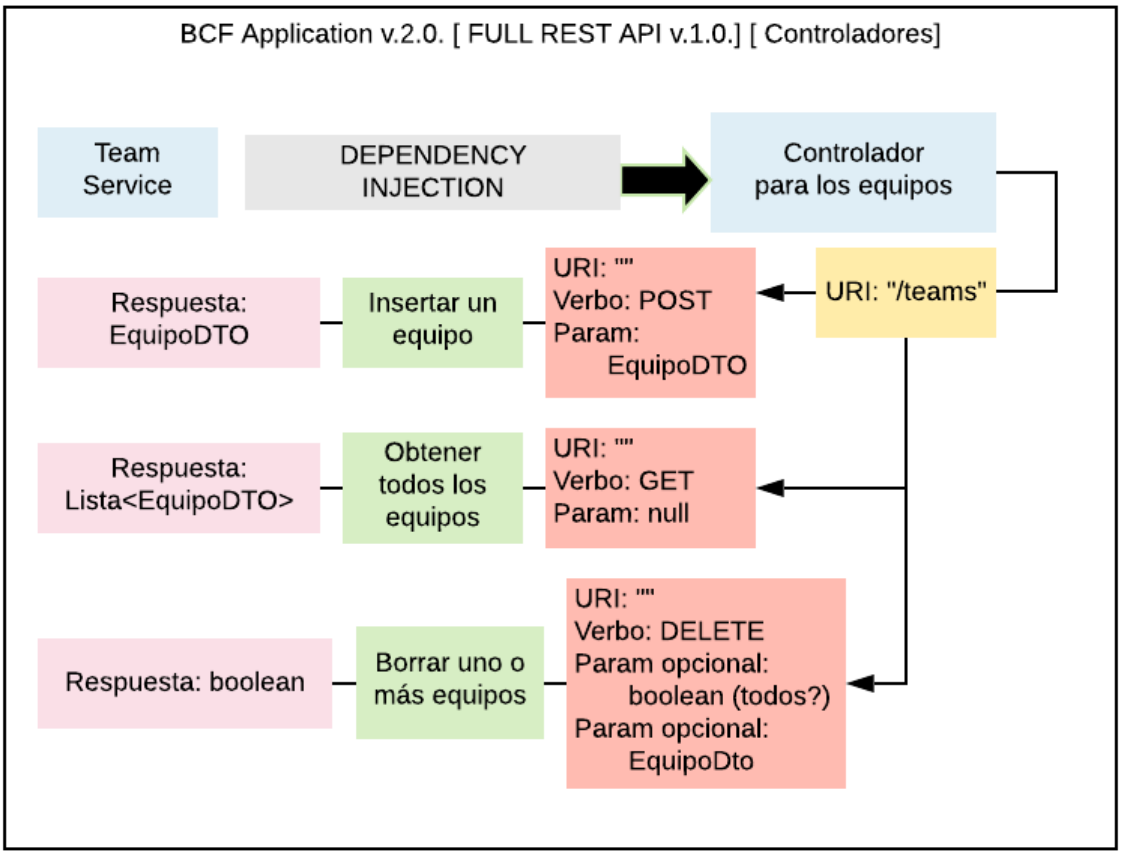

BCF Application v.2.0. [FULL REST API v.1.0.] [Servicios]
Team Service
Player Service
Match Repository
Mail Service
DEPENDENCY INJECTION
Servicio encargado de todas las operaciones relacionadas con los partidos
Insertar un partido
Obtener el partido de la jornada
Borrar el partido abierto o todos
Buscar un partido sin resultado

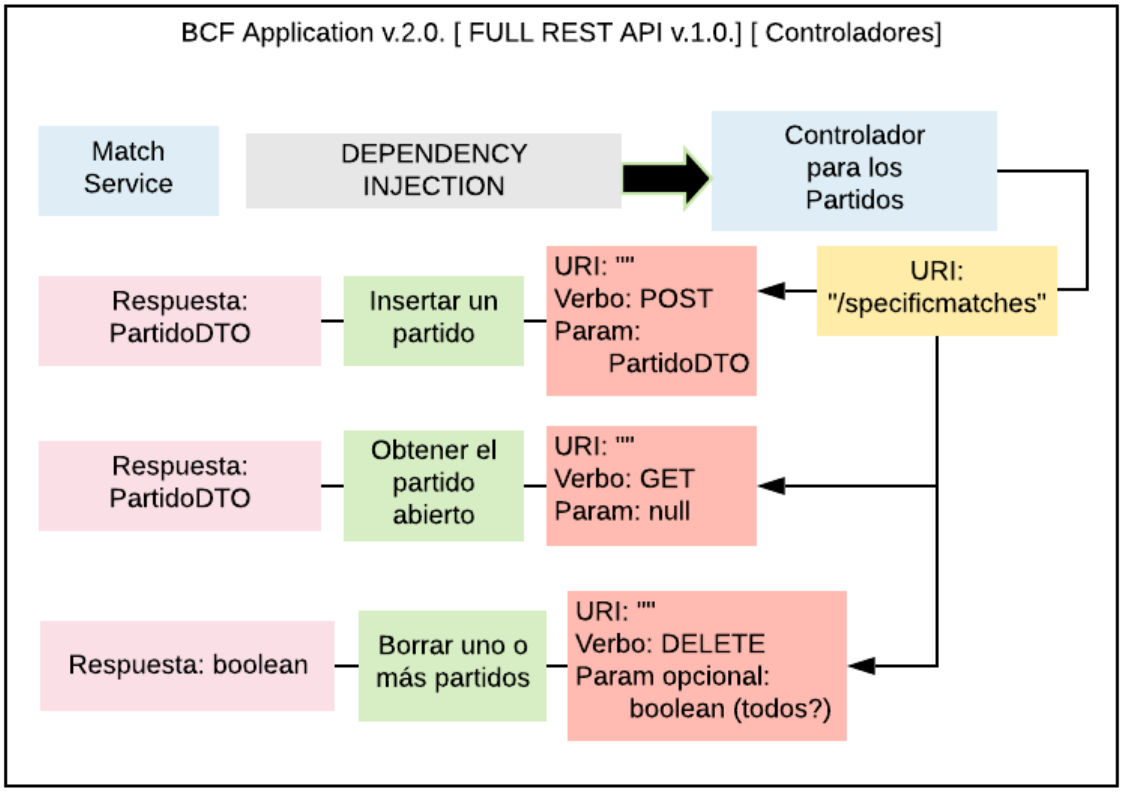

BCF Application v.2.0. [FULL REST API v.1.0.] [Servicios]
Match Service
Player Service
Bet Repository
Mail Service
DEPENDENCY INJECTION
Servicio encargado de todas las operaciones relacionadas con las apuestas
Insertar una apuesta
Obtener todos los datos de las apuestas, de tres formas diferentes
Finalizar la jornada y calculo de puntuaciones, se devuelve la data actualizada

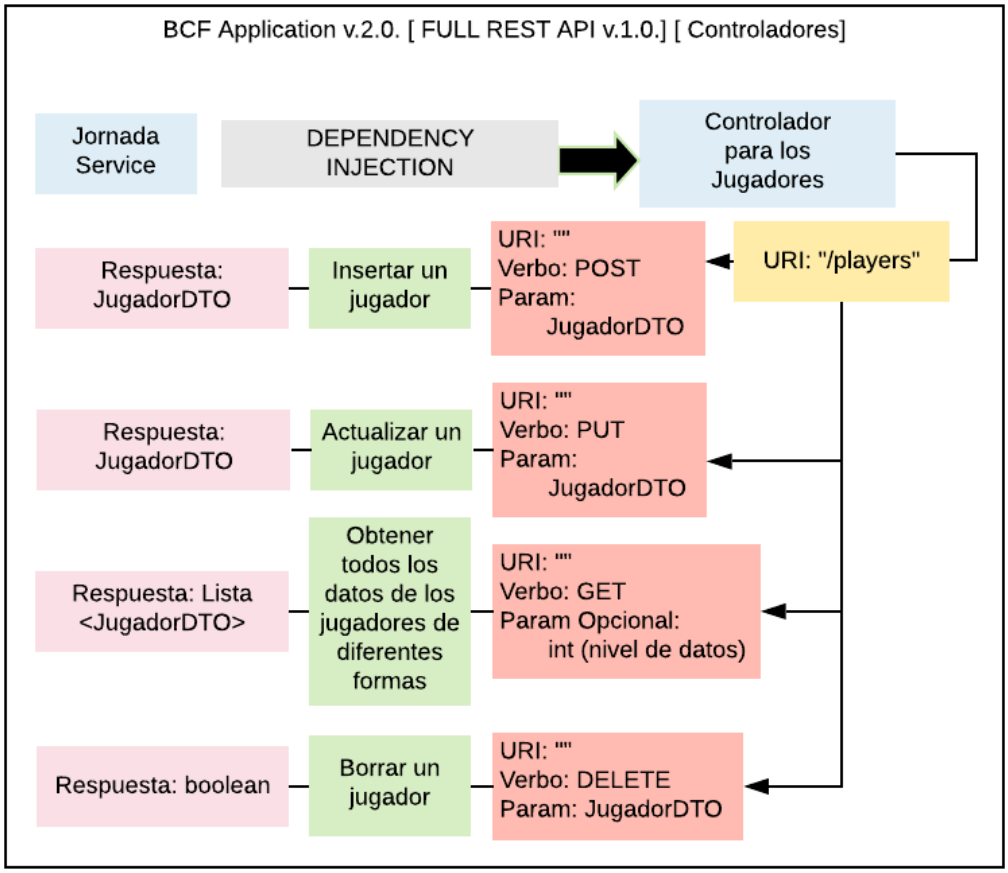


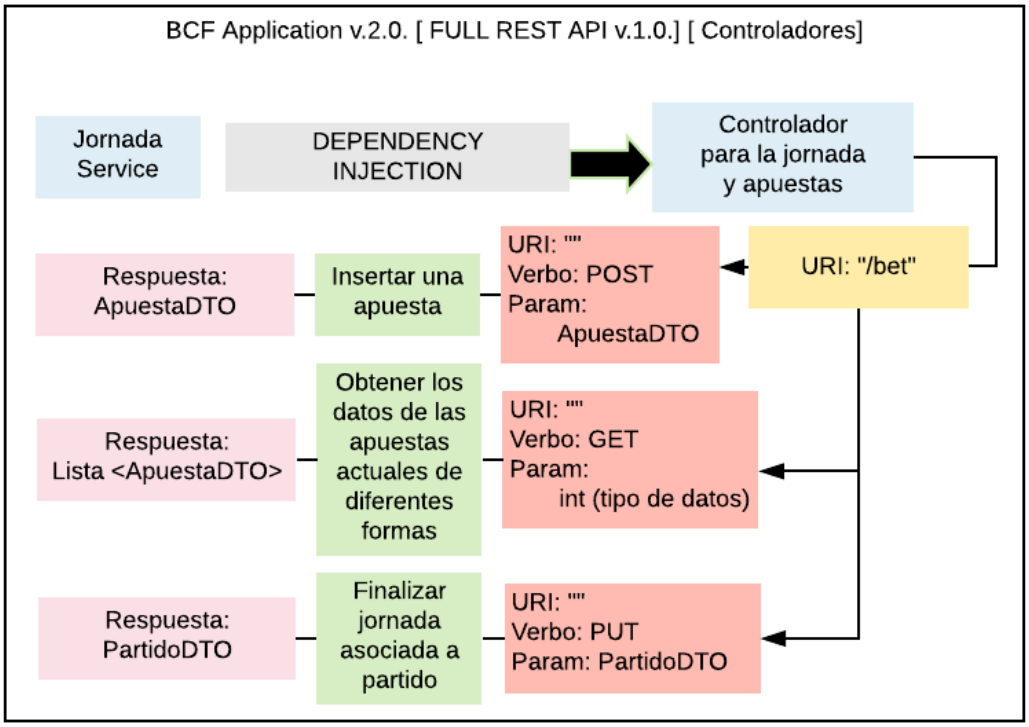
En la capa de controladores a partir de la URL “origen” se exponen los siguientes.

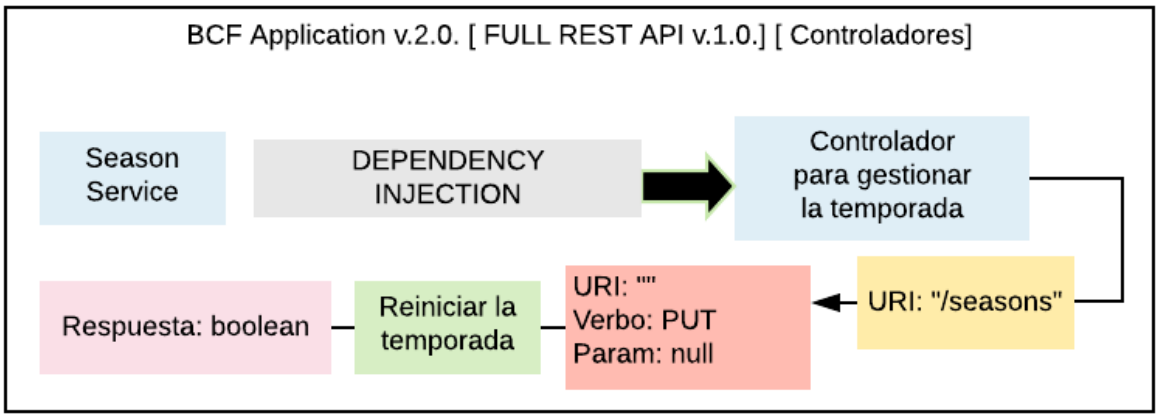


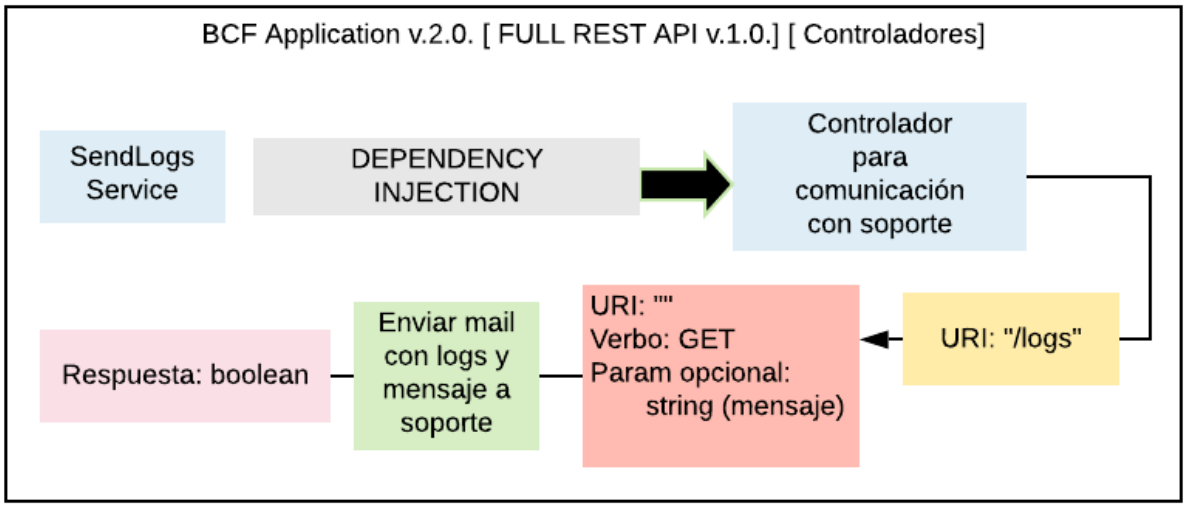

BCF Application v.2.0. [FULL REST API v.1.0.] [Controladores]
Team Service
DEPENDENCY INJECTION
Controlador para los equipos
URI: "/teams"
URI: ""
Verbo: POST
Param: EquipoDTO
Insertar un equipo
Respuesta: EquipoDTO
URI: ""
Verbo: GET
Param: null
Obtener todos los equipos
Respuesta: Lista<EquipoDTO>
URI: ""
Verbo: DELETE
Param opcional: boolean (todos?)
Param opcional: EquipoDto
Borrar uno o más equipos
Respuesta: boolean

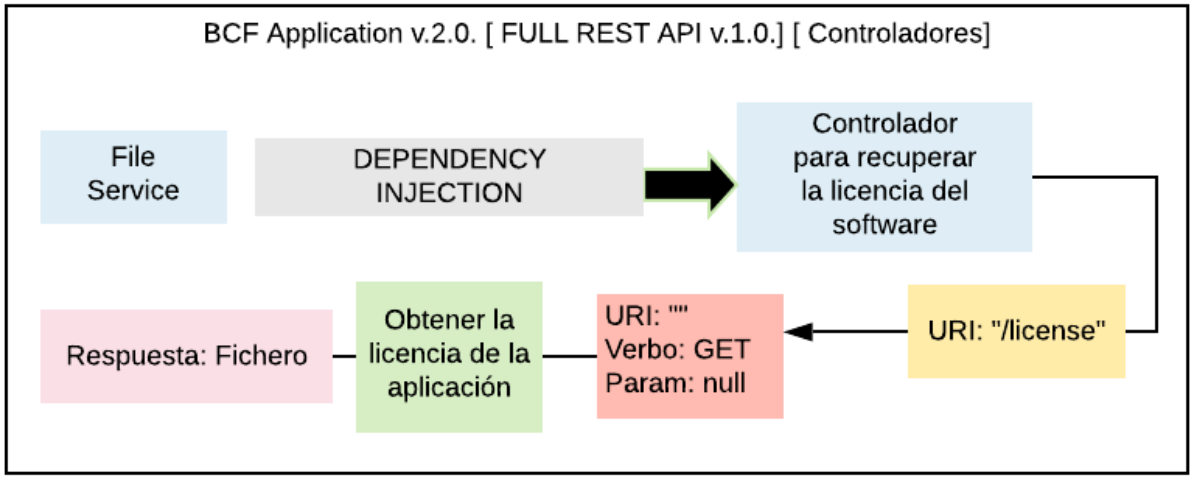

BCF Application v.2.0. [FULL REST API v.1.0.] [Controladores]
Match Service
DEPENDENCY INJECTION
Controlador para los Partidos
URI: "/specificmatches"
URI: ""
Verbo: POST
Param: PartidoDTO
Insertar un partido
Respuesta: PartidoDTO
URI: ""
Verbo: GET
Param: null
Obtener el partido abierto
Respuesta: PartidoDTO
URI: ""
Verbo: DELETE
Param opcional: boolean (todos?)
Borrar uno o más partidos
Respuesta: boolean


BCF Application v.2.0. [FULL REST API v.1.0.] [Controladores]
Jornada Service
DEPENDENCY INJECTION
Controlador para los Jugadores
URI: "/players"
Respuesta: JugadorDTO
Insertar un jugador
URI: ""
Verbo: POST
Param: JugadorDTO
Respuesta: JugadorDTO
Actualizar un jugador
URI: ""
Verbo: PUT
Param: JugadorDTO
Respuesta: Lista <JugadorDTO>
Obtener todos los datos de los jugadores de diferentes formas
URI: ""
Verbo: GET
Param Opcional: int (nivel de datos)
Respuesta: boolean
Borrar un jugador
URI: ""
Verbo: DELETE
Param: JugadorDTO

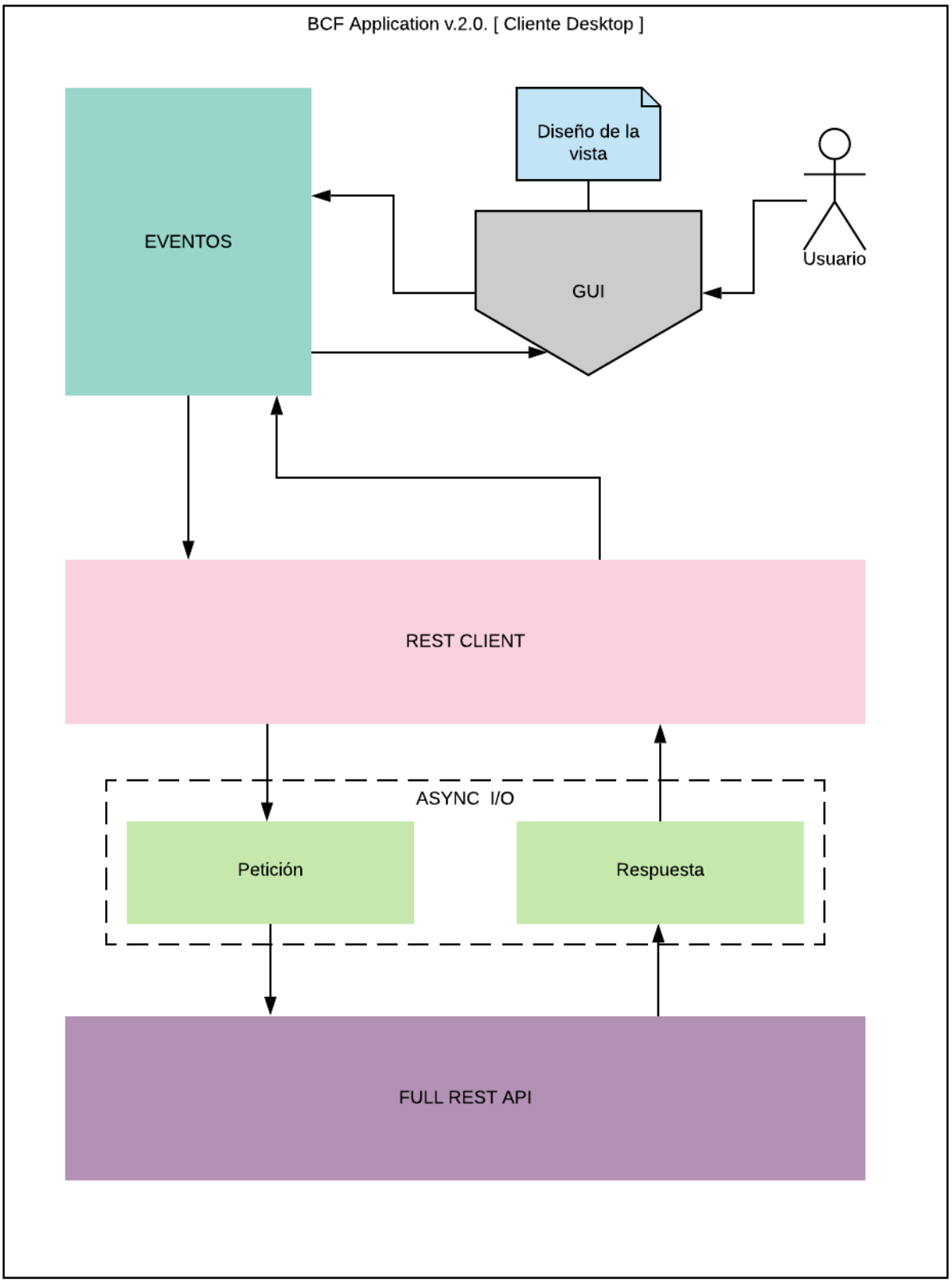

BCF Application v.2.0. [FULL REST API v.1.0.] [Controladores]
Jornada Service
DEPENDENCY INJECTION
Controlador para la jornada y apuestas
URI: "/bet"
Respuesta: ApuestaDTO
Insertar una apuesta
URI: ""
Verbo: POST
Param: ApuestaDTO
Respuesta: Lista <ApuestaDTO>
Obtener los datos de las apuestas actuales de diferentes formas
URI: ""
Verbo: GET
Param: int (tipo de datos)
Respuesta: PartidoDTO
Finalizar jornada asociada a partido
URI: ""
Verbo: PUT
Param: PartidoDTO


BCF Application v.2.0. [FULL REST API v.1.0.] [Controladores]
Season Service
DEPENDENCY INJECTION
Controlador para gestionar la temporada
URI: "/seasons"
URI: ""
Verbo: PUT
Param: null
Reiniciar la temporada
Respuesta: boolean


BCF Application v.2.0. [FULL REST API v.1.0.] [Controladores]
SendLogs Service
DEPENDENCY INJECTION
Controlador para comunicación con soporte
URI: "/logs"
URI: ""
Verbo: GET
Param opcional:
string (mensaje)
Enviar mail con logs y mensaje a soporte
Respuesta: boolean


BCF Application v.2.0. [FULL REST API v.1.0.] [Controladores]
File Service
DEPENDENCY INJECTION
Controlador para recuperar la licencia del software
URI: "/license"
URI: ""
Verbo: GET
Param: null
Obtener la licencia de la aplicación
Respuesta: Fichero

El cliente elegido actualmente es una Desktop Application, partiendo de la versión uno de la aplicación se recupera el diseño visual añadiendo más funcionalidades y agregando el RestClient.



Implementación

Se lleva a cabo la implementación de la API Rest en Java, lenguaje de programación multi-paradigma. El proyecto es inicializado con SpringBoot (inicializador de proyectos Spring (framework)).
Se utilizan diferentes paradigmas, programación orientada a objetos, programación funcional, programación asíncrona y distribuída.

El proyecto “Rest” se estructura en paquetes o módulos que identifican unívocamente las capas o recursos encapsulados.

Estructura simple de paquetes y resumen de lo que contienen:

org.jeycode **{**
- Ejecutable
- ***.configurations*** **{**
  - Configuraciones de los ThreadPool utilizados para las operaciones async
  - Configuración de SWAGGER
- **}**
- ***.controllers*** **{**
  - Controladores
- **}**
- ***.dtos*** **{**
  - ***.concreteMatchDto*** **{**
    - DTOs de los partidos
  - **}**
  - ***.playerDto*** **{**
    - DTOs de los jugadores
  - **}**
  - ***.playerFootballMatchDto*** **{**
    - DTOs de las apuestas
  - **}**
  - ***.rulesDto*** **{**
    - DTOs de las reglas
  - **}**
  - ***teamsDto*** **{**
    - DTOs de los equipos
  - **}**
- ***.exceptionsManaged*** **{**
  - Excepciones y API de gestión de errores o excepciones
- **}**
- ***.mappers*** **{**
  - Interfaces Mappers y sus implementaciones autogeneradas
- **}**
- ***.models*** **{**
  - Entidades de dominio
- **}**
- ***.repositories*** **{**
  - Repositorios para el acceso a datos
- **}**
- ***.services*** **{**
  - ***.components*** **{**
    - Clases que se ocupan de una funcionalidad pequeña usada en los servicios
  - **}**
  - ***.genericService*** **{**
    - Servicios de la aplicación no ligados al modelo
  - **}**
  - ***.repositoryService*** **{**
    - Servicios de la aplicación ligados al modelo
  - **}**
- ***.utilities*** **{**
  - Todas las clases Helper de la aplicación
- **}**

}

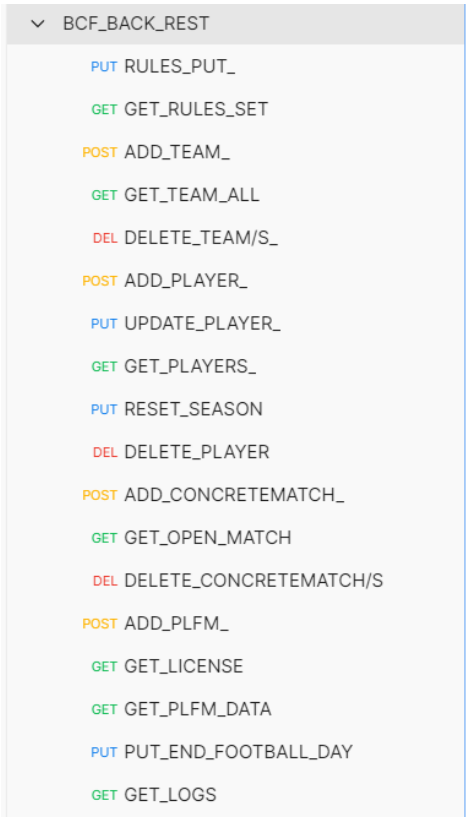
El cliente desarrollado para consumir la API está desarrollado en C# .Net Framework, todas las operaciones I/O se hacen de manera asíncrona.

Sigue la siguiente estructura:

Solución **{**
Ejecutable
Configuraciones
.view {
Ventanas y diálogos – Diseño/Eventos
}
.restClient {
Wrapper – Helper para mapeo de un Json
.request {
Todos los servicios del cliente REST asociados a los controladores de la API contra la que realizan las request
.errors {
Wrapper – Helper para mapeo de los Json de error
}
.model {
Clases que encapsulan los datos a mostrar en la vista, o los datos a enviar junto a las peticiones
}
}
}
}

Pruebas

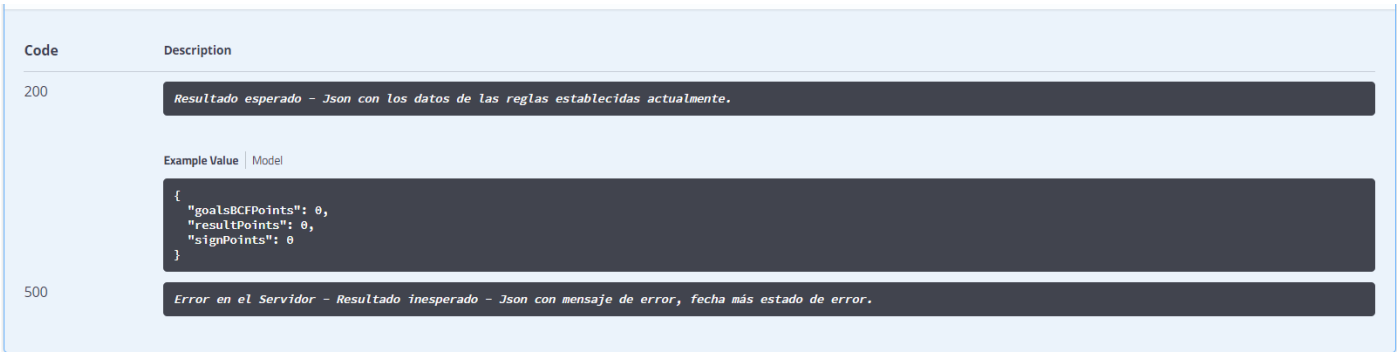
Se realizan pruebas funcionales de toda la aplicación.
La API se prueba con POSTMAN creando una colección de peticiones paralela al proyecto.

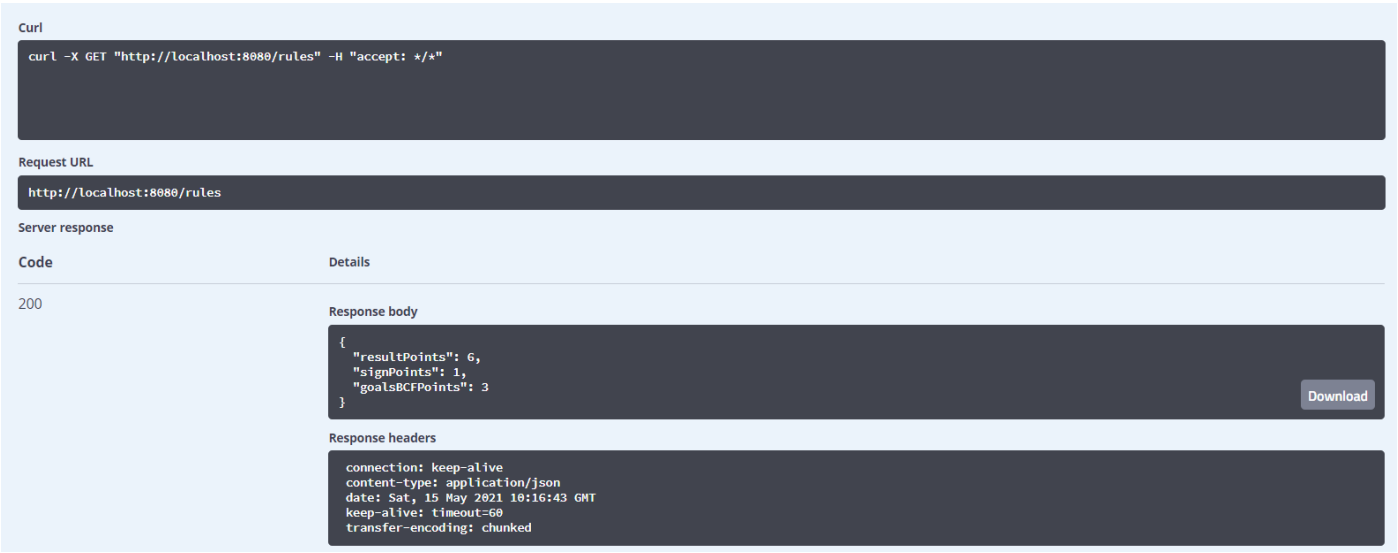


- Con POSTMAN se comprueba que todos los controladores funcionan según lo esperado, se devuelven los datos esperados, incluyendo códigos de respuesta.

- Una vez realizada la documentación con Swagger, se dispone de otra herramienta de testing.

Ejemplo de Swagger realizando una petición a la API.





- La base de datos en desarrollo es H2, ideal para pruebas.

- Se incluyen correos electrónicos de prueba para verificar que se envían perfectamente y con el diseño establecido.

- Se realizan las mismas pruebas contra la API de una tercera forma, se crea una aplicación de consola en el lenguaje de programación del cliente desktop. Realizada todo el módulo de Rest Client en C# se prueba. Una vez comprobado el funcionamiento se procede a acoplarse a la aplicación cliente y se prueba de nuevo asociándose a la GUI.

Recursos

Recursos utilizados durante el desarrollo.

Hardware

Se utiliza un Laptop Lenovo con las siguientes características:

- Procesador: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99 GHz
- RAM: 8GB
- S.O.: Windows 10 Home 64bits

Software

Herramientas de diseño gráfico

- **GIMP** 2.10, manipulación de imágenes.
- **Inkscape**, diseño vectorial.

Entornos de Desarrollo Integrados

- **Eclipse** 2020-09 + plugin Spring Tool Suite v.4, desarrollo de la Full Rest API.
- **Visual Studio** 2019 16.8, desarrollo de la aplicación o cliente desktop.

Herramienta de Gestión y Construcción de Proyectos

- Apache **Maven** 3.6, gestión y construcción de la Full Rest API.

Herramientas de control de versiones

- **GIT**, control de versiones en local utilizando GIT Bash.
- **GitHub**, repositorio online gratuito.

Herramientas de Testeo y Documentación de API Rest

- **PostMan**.
- **Swagger** + Swagger UI.
- **Curl,** command-line tool.

Bases de Datos

- **SQLite**, base de datos embebida en la v.1.0 del proyecto.
- **H2** database, base de datos embebida para ambientes de desarrollo, en la v.2.0 del proyecto.

Frameworks, librerías y extensiones del lenguaje

Distinguir los utilizados en las diferentes versiones de la aplicación.

V.1.0

Frameworks

- [.NET] **Entity Framework** 6.2, ORM para la plataforma de .NET.
- [.NET] **WPF**, framework para crear aplicaciones de escritorio usando C# más XAML.
- [.NET] **MahApps.Metro** 2.4.3, para facilitar el desarrollo de aplicaciones WPF con diseño moderno.

Extensiones del lenguaje

- [.NET] **LINQ**, extensión de C#, para trabajar de forma sencilla con colecciones de datos. Muy parecido a SQL.
- [.NET] **FluentValidations** 9.5.1, librería para construir robustas validaciones de una forma simple.

Librerías

- [.NET] **ControlzEx** 4.4, librería con controles para WPF.

V.2.0

Frameworks

- [.NET] **WPF**, framework para crear aplicaciones de escritorio usando C# más XAML.
- [.NET] **MahApps.Metro** 2.4.3, para facilitar el desarrollo de aplicaciones WPF con diseño moderno.
- [Java] **SpringBoot** 2.4.3, inicializador de proyectos Spring (framework de desarrollo de aplicaciones para la plataforma Java). Incluye por defecto un servidor Tomcat.
- [Java] **Hibernate**, ORM para la plataforma Java.

Extensiones del lenguaje

- [.NET] **LINQ**, extensión de C#, para trabajar de forma sencilla con colecciones de datos.
- [.NET] **Microsoft.Bcl.AsyncInterfaces**, paquetes que incluyen interfaces y helpers para operaciones asíncronas.
- [.NET] **Microsoft.AspNet.WebApi.Client**, paquete necesario para realizar el Rest Client.
- [.NET] **System.Memory, System.Numerics.Vectors, System.Buffers, System.Text.Encoding.Web**. Paquetes necesarios para mejoras de performance u otras utilidades.
- [.NET] **System.Text.Json**, paquete para serialización/deserialización de Objetos/Json.
- [.NET] **System.Threading.Task.Extensions**, paquete que ofrece tipos que simplifican el trabajo de escribir código asíncrono.

Librerías

- [.NET] **ControlzEx** 4.4, librería con controles para WPF.
- [Java] **spring-boot-starter-data-jpa**, dependecia de Springboot que facilita en gran medida el trabajo de desarrollo en el acceso a datos, por ejemplo, los Query-Method-Name.
- [Java] **spring-boot-starter-web**, dependencia de Springboot necesaria para cualquier proyecto web desarrollado con Spring, incluye Tomcat embebido.
- [Java] **spring-boot-starter-validation**, dependencia de Springboot que incluye **javax.validations**, librería que ofrece anotaciones para realizar validaciones de una forma simple. También anotaciones e interfaces para validar dinámicamente.
- [Java] **spring-boot-starter-mail**, dependencia de Springboot para el envío de correos electrónicos.
- [Java] **h2database**, librería para poder usar la base de datos embebida H2.
- [Java] **Jackson ObjectMapper**, librería para la serialización/deserialización de Objetos/Json.
- [Java] **Zip4j** 2.7.0, librería para comprimir archivos.
- [Java] **Slf4J**, ofrece una abstracción para los frameworks de logging.
- [Java] **Lombok**, librería que genera código en tiempo de compilación y mejora la legibilidad del código.
- [Java] **MapStruct** 1.4.2, librería que facilita la creación de clases para mapear objetos, genera código en compilación.

- [Java] **springfox – swagger2 / ui** 2.9.2, librería para implementar swagger con springboot, se configura mediante anotaciones. Ofrece una GUI.

Planificación

Temporal

El proyecto se divide en dos fases principales, además se diseñan unas tarjetas para dividir y resumir las tareas a realizar durante el proyecto.
El formato de las tarjetas es el siguiente:

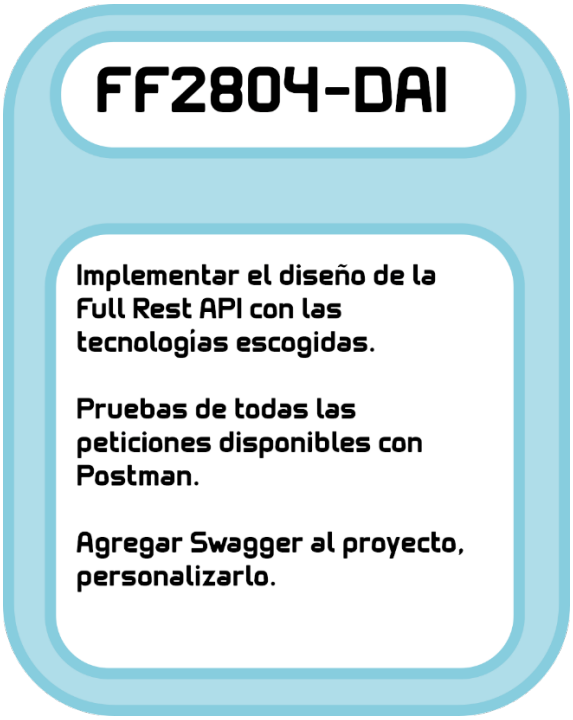
- Header: FF(Fecha Fin) DD(Día del mes) MM (Mes) - ***(Letras ID del desarrollo).

- Body: Resumen de las tareas a realizar.

1. Diseño e Implementación de Full Rest API

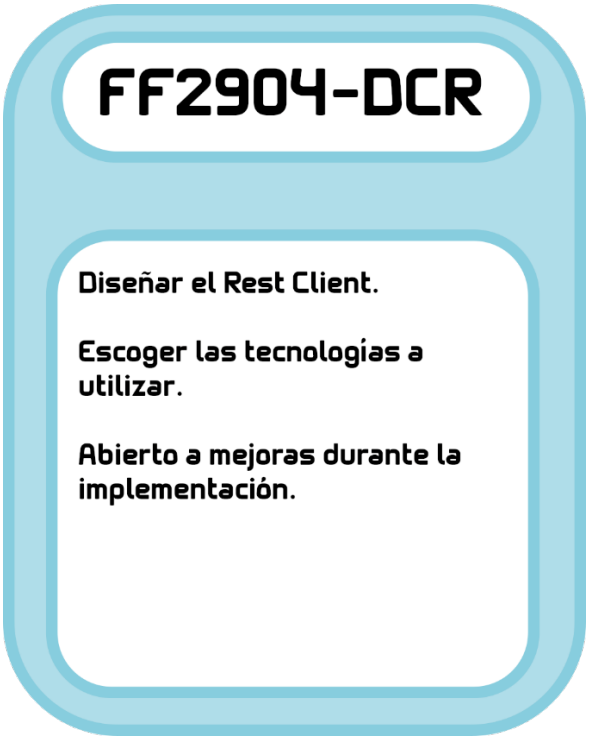
La fase más importante del desarrollo, de no cumplir los plazos el proyecto puede reducir funcionalidad en la entrega final.
Para identificar el trabajo a realizar y las fechas de inicio/fin se dispone de las siguientes tareas.

La tarea inicia y finaliza el mismo día invirtiendo 0,5h y cumpliendo plazo.

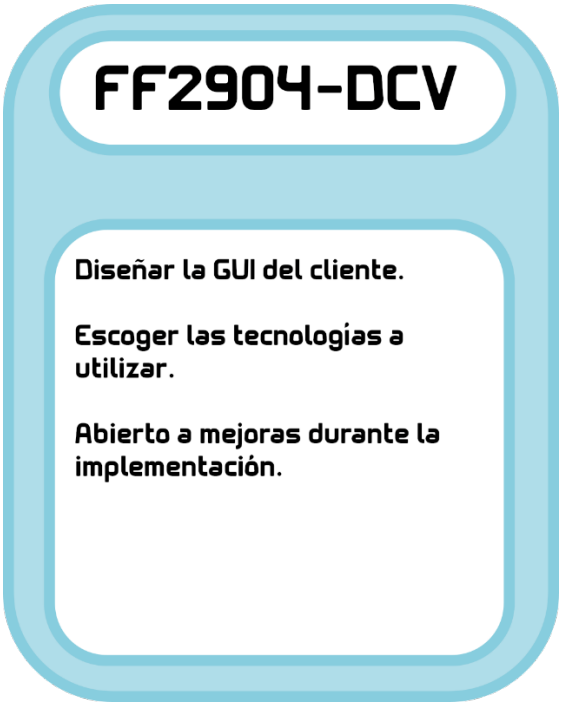


La tarea se inicia el 29/03/2021, se realiza un sprint intensivo hasta el día 03 de Abril, invirtiendo 5h cada día. La tarea se finaliza el 27/04/2021, un día antes del plazo. Horas totales destinadas a la tarea: 58h

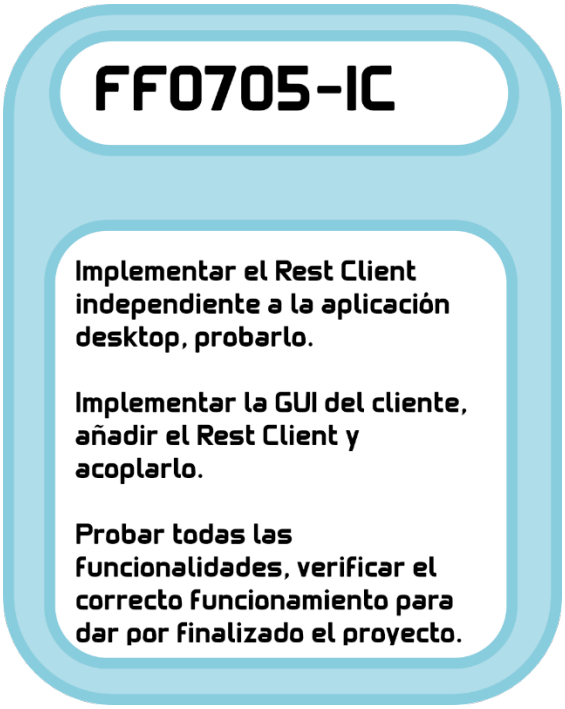
2. Diseño e Implementación del Cliente Rest y la GUI de la aplicación desktop.



La tarea inicia y finaliza el mismo día invirtiendo 0,25h y cumpliendo plazo.



La tarea inicia y finaliza el mismo día invirtiendo 0,25h y cumpliendo plazo.



Tarea iniciada el día 03/05/2021 y finalizada el día 07/05/2021. Se cumplen los plazos.
Horas destinadas: 12h

Horas totales del proyecto → 0,5 + 58 + 0,25 + 0,25 + 12 = **[71h]**

Económica

La Full Rest API desplegada en AWS EC2 en la capa gratuita no supone ningún costo extra para el cliente durante el primer año.

Las horas totales invertidas en el desarrollo del proyecto suman un total de: 71h * 25€/h = **1775 €**
Gastos extraordinarios durante el desarrollo 200€. Contratar soporte supone un coste de 100€/anuales más 25€/h la hora trabajada.

De no contratar la opción de soporte durante el primer año, la venta del producto se incrementará en 300€ como penalización.

Coste del producto:

- **2095 €** contratando soporte durante el primer año.
- **2275 €** adquiriendo el producto sin contratar soporte.

Conclusiones

Cumplimiento de los objetivos fijados

El proyecto finalizado cumple todos los requisitos fijados añadiendo nuevos diseños y funcionalidades y todo ello en los plazos estimados.

La aplicación cumple su objetivo, es funcional. La Full Rest API ofrece muchas posibilidades de futuros clientes a diseñar.
El cliente desktop tiene un diseño bonito y simple con referencias al ámbito futbolístico y, en particular, al Burgos C.F., gran protagonista del proyecto.

Propuestas de mejora o ampliaciones en futuras versiones del producto

Desde el momento en el que se desarrolla toda la lógica de negocio en una Full Rest API, las mejoras a futuro son prácticamente infinitas.

- Al momento de desplegar la aplicación en un hosting, agregar una nueva bbdd externa o embebida, además localizar el almacenamiento secundario en otra ubicación que no sea dentro del jar.

- Añadir Spring Security, asegurar la API con JWT y roles.

- Crear clientes Web y mobile.

- Añadir nuevas funcionalidades a la API.

- Generar un Job que realice WebScrapping sobre la página de la R.F.E.F. (Real Federación Española de Fútbol) para obtener todos los equipos rivales del Burgos en cada re-inicio de la temporada.

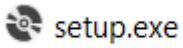
- Agregar a la API un sistema de caché, se puede habilitar y usar el sistema de caché por defecto de Spring.

- Ante un aumento grande de usuarios, crear un dominio de correo electrónico y contratar más cantidad de correos a enviar en un día.

Guías

Instalación

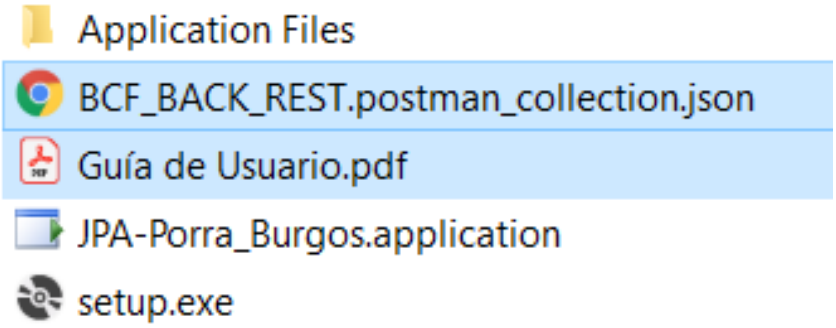
El cliente es un programa portable, únicamente hay que pinchar dos veces en el ejecutable (setup.exe) se instala y arranca la aplicación desktop.



La API se construye con Maven *-- mvn clean package --* generando un .jar con un server embebido. Este jar se puede ejecutar en local o subir a un hosting de aplicaciones. De usarse otro servidor, en vez del Tomcat embebido, debe construirse como war.

Uso

El proyecto incluye una guía de usuario para el cliente Desktop. Se encuentra empaquetada en la carpeta comprimida en donde se encuentra el ejecutable. Junto a estos dos ficheros se incluye un Json con la colección de peticiones realizadas en POSTMAN, es decir, todas las existentes. En caso de que otro desarrollador quiera probar la API en POSTMAN solo necesita importar esta colección.



La Full Rest API incluye otra guía técnica con interfaz gráfica, Swagger ui. Con Swagger se obtiene una gran ventaja a la hora de que otro desarrollador cree un cliente nuevo sin conocer el back, ya que puede revisar continuamente, en desarrollo, lo que devuelve o espera recibir en cada request de los controllers, así también conocer el modelo.

Para visualizar e interactuar con Swagger ui solo hay que poner la ruta a la API en un navegador más /swagger-ui.html#.
Por ejemplo, ejecutando el server en local en el puerto 8080…
http://localhost:8080/swagger-ui.html#

Referencias

Búsqueda: Comprimir un archivo en Zip Java.
Fuente: https://es.stackoverflow.com/questions/75133/comprimir-una-carpeta-completa-en-zip-e-ingresar-el-nombre-de-cual-carpeta-sera
Autor de la respuesta válida: 'Hexadecimal'
Fecha: 24/04/2021

Búsqueda: Rest client en C# multipart.
Fuente: https://stackoverflow.com/questions/59888921/c-sharp-multipart-form-data-in-httpclient-post-rest-api
Autor de la respuesta válida: No se encuentra respuesta válida
Fecha: 05/05/2021

Búsqueda: Alojar API Rest Springboot gratis
Fuente: https://aws.amazon.com/es/free/?all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc&awsf.Free%20Tier%20Types=*all&awsf.Free%20Tier%20Categories=*all
Autor de la respuesta válida: Desconocido
Fecha: 15/05/2021

Búsqueda: Cómo realizar una petición con CURL
Fuente: Documentación propia.
Autor de la respuesta válida: Javier Pérez Alonso
Fecha: 23/04/2021