UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA DE SISTEMAS INFORMÁTICOS

PROYECTO FIN DE GRADO
GRADO EN INGENIERÍA DEL SOFTWARE

Desarrollo de una plataforma de juegos de programación vía API REST

Autores:

Elena Flecha Alfonso

Miguel Ángel Hernández Castro

Tutor: Adolfo Yela Ruiz

Curso 2019/2020

PREFACIO

El presente texto constituye la documentación final del Proyecto de Fin de Grado titulado "*Desarrollo de una plataforma de juegos de programación vía API REST*", realizado para la obtención del título en la ETSISI de la Universidad Politécnica de Madrid por los alumnos de Ingeniería de Software: *Elena Flecha Alfonso* y *Miguel Ángel Hernández Castro*.

El proyecto en cuestión se llevó a cabo a partir de diciembre de 2019 y hasta mayo de 2020 bajo la tutoría de don *Adolfo Yela Ruiz.*

"Hay dos maneras de diseñar software: una es hacerlo tan simple que sea obvia su falta de deficiencias, y la otra es hacerlo tan complejo que no haya deficiencias obvias"

-- C.A.R. Hoare

1. RESUMEN

El proyecto aborda el flujo completo de estudio, diseño e implementación de una plataforma online de juegos de programación. Esta pretende enseñar lo que es y cómo se utiliza una API REST de una manera "gamificada", animando a los usuarios a implementar algoritmos para la resolución automática de juegos haciendo llamadas al API. Los juegos soportados por la plataforma son "Bulls and Cows" y "Laberinto".

La plataforma está desarrollada con tecnologías punteras y patrones de diseño y arquitectura de software avanzados. Cuenta con un portal web implementado con Angular que explica las reglas de los juegos y permite visualizar los resultados de las partidas. Además, tiene una API REST implementada con el framework Micronaut que expone los endpoints para ejecutar las partidas y dar servicio al portal web.

2. ABSTRACT

The project addresses the complete flow of study, design and implementation of an online programming games platform. It aims to teach what a REST API is and how it is used in a "gamified" way, encouraging users to implement algorithms for automatic game resolution by making API calls. Supported games for the platform are "Bulls and Cows" and "Labyrinth".

The platform was developed with cutting-edge technologies and advanced software architecture and design patterns. It has a web portal implemented with Angular that explains the rules of the games and shows the results of the matches. Also it has a REST API implemented with the Micronaut framework that exposes the endpoints to execute the matches and provide service to the web portal.

3. ÍNDICE DE CONTENIDOS

4. INTRODUCCIÓN

4.1. Objetivos

El objetivo del proyecto es la creación de una plataforma con la que enseñar lo que es y cómo se utiliza una API REST de una manera "gamificada" animando a practicar la implementación de algoritmos para la resolución automática de juegos. Así también se plantea y propone la utilización de la plataforma como mecanismo de demostración de conocimientos, para por ejemplo, utilizarlo en el proceso de selección de personal para programadores de software o en la evaluación de una asignatura de programación.

El usuario de la plataforma deberá implementar una solución capaz de:

- Manejar el flujo de las partidas de un juego, usando una API REST para iniciar las partidas, ejecutar sus movimientos/turnos y consultar el estado del tablero/partida.
- Implementar **algoritmos** que resuelvan los juegos de la manera más óptima posible.

Para la ejecución del proyecto se pondrán en práctica todas las nociones aprendidas en cuanto a gestión de un proyecto, ejecutando paso a paso su **planteamiento teórico, definición, análisis, diseño e implementación**.

Para ello la plataforma contará con dos partes principales:

- **API** - Servidor que implementará el manejo de los juegos y expondrá un API para su explotación y uso.
- **Web** - Página que permitirá el registro de los usuarios (bots) y la consulta de todas las partidas ejecutadas de una manera visual.

Para la implementación de la solución se utilizarán **metodologías, arquitecturas y tecnologías** modernas que permitan un código estructurado, testeado, eficiente y óptimo.

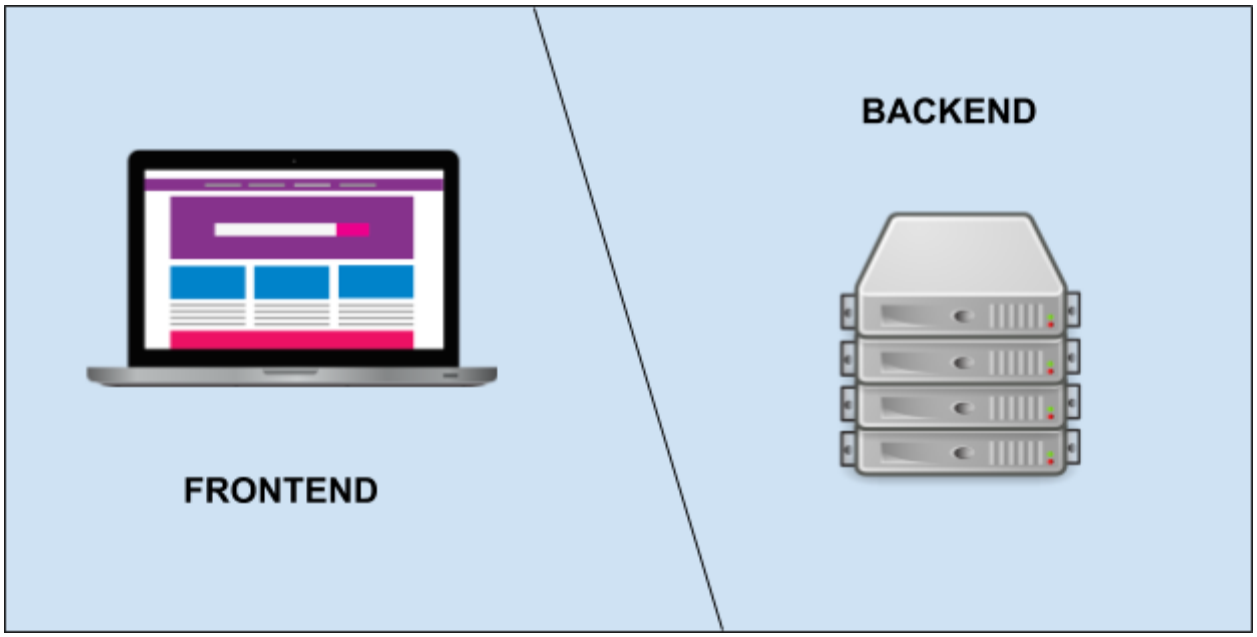
La plataforma contará con **dos juegos** (para esta primera versión) y será desarrollada con capacidad para ser ampliable de manera sencilla y rápida:

- **Bulls and cows**: descubre un código secreto de números con las pistas que te ofrece el sistema con cada prueba.
- **Laberinto**: consigue salir del laberinto con la información que tienes de tus posiciones más cercanas.

5. ESTUDIO TEÓRICO

En este apartado estudiaremos el "**estado del arte**" del panorama actual tecnológico. Realizaremos un estudio de las tecnologías disponibles seleccionando las más adecuadas para la elaboración del proyecto.

El desarrollo de aplicaciones se suele dividir en dos partes principales. El **Backend** y el **Frontend**.



El **Backend** está enfocado en hacer que todo lo que está detrás de un sitio web funcione correctamente. Recoge los datos, los procesa y los envía al usuario, además de encargarse de las consultas o peticiones a la **Base de Datos**, la exposición de los servicios y lógica de negocio desde el servidor.

En el caso del proyecto a desarrollar el Backend contendrá toda la lógica de ejecución de los juegos y expondrá sus servicios de consulta y ejecución de partidas como un **API REST**.

El **Frontend** se enfoca en el usuario, en todo con lo que podemos interactuar y lo que vemos mientras navegamos y usamos la aplicación/web.

En el caso del proyecto a desarrollar el frontend será una página web donde se podrán ver y analizar las partidas ejecutadas y se podrá leer toda la documentación y tutoriales sobre el uso de la solución.

5.1. Frontend

5.1.1. HyperText Markup Language (HTML)

HTML es un "lenguaje de marcas de hipertexto". Este es un estándar para la construcción de páginas web.

El código de una página web es simple texto que debe ser interpretado por un navegador web compatible para unir los elementos formando la visualización de la página web.

Cuando un navegador carga una página web crea un **"Document Object Model" (DOM)** de la página. Este representa, mediante un árbol, todo el contenido de la página estructurando sus elementos para poder ser renderizados para su visualización. La modificación de este objeto conlleva un cambio en la vista mostrada de la página web.

La inclusión de ficheros **CSS** define el aspecto del contenido de la página.

Un fichero HTML puede incluir **scripts de código** en lenguajes como JavaScript para modificar el comportamiento y contenido de las páginas web dinámicamente.

5.1.2. Cascading Style Sheets (CSS)

Una **hoja de estilos CSS** es usada para describir la presentación de un documento web HTML. Esto permite separar la presentación definida con CSS del contenido definido en el HTML.

Entre las cosas que permite definir CSS se encuentran las colocaciones, tamaños, colores, tipos de letra, fondos, etc.

El código CSS se divide en dos partes principales: los selectores y los bloques de declaración.

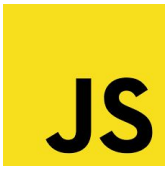
Un **selector** permite definir a qué conjunto de elementos afecta una declaración.

Un **bloque de declaración** contiene una serie de propiedades y valores que definen los estilos del conjunto de elementos indicado por el selector.

5.1.3. Lenguaje de scripts

5.1.3.1. JavaScript

El lenguaje de programación más tradicional que utilizan los websites y ejecutan los navegadores web es **JavaScript (JS)**.



JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Los navegadores interpretan el código JavaScript que integran las páginas web para permitir interactuar con esta modificando el DOM (Document Object Model) permitiendo la implementación de aplicaciones realmente complejas.

Mediante el uso de tecnologías como **AJAX** se permite la conectividad asíncrona con el servidor para interactuar y modificar la página con nueva información remota sin

actualizar la página. Es habitual que esta información se obtenga invocando endpoints de una API REST via HTTP.

El estándar **ECMAScript** fue lanzado en junio de 1997, desde entonces se han lanzado muchas versiones. En los últimos años ECMAScript ha evolucionado mucho y la manera de escribir código con este lenguaje ha cambiado increíblemente con las nuevas posibilidades de este lenguaje. En la siguiente tabla se pueden ver las fechas de publicación de las diferentes versiones:

Edición	Fecha de publicación
1	Junio de 1997
2	Junio de 1998
3	Diciembre de 1999
4	Abandonado
5	Diciembre de 2009
5.1	Junio de 2011
6	Junio de 2015[12]
7	Junio de 2016
8	Junio de 2017
9	Junio de 2018

Como se puede observar entre 1997 y 2011 se publicaron 6 versiones. Y solo en los últimos 4 años se han publicado 4 nuevas versiones. Estas han agregado cambios significativos en la sintaxis del lenguaje que ayudan a la implementación de aplicaciones complejas como pueden ser: clases, módulos, colecciones, promesas, async/await, operador rest, iteración asíncrona y mucho más.

5.1.3.2. TypeScript

Para suplir las faltas de JavaScript por ser un lenguaje no tipado surgió TypeScript. Este lenguaje ha sido desarrollado y es mantenido por Microsoft.

Autodefinido como el "JavaScript" que escala, TypeScript es un **superset tipado de JavaScript** que se "compila" a JavaScript plano. Esta compilación transforma el código programado en código estándar JavaScript soportado y ejecutable en cualquier navegador web.

Su principal aporte es que añade tipos estáticos sin perder la esencia de JavaScript. Permite así definir variables y funciones tipadas pero cualquier código JavaScript es soportado como código TypeScript, pues sus cambios son solo "añadidos" al lenguaje.

Esto permite escalar el código con ayudas en tiempo de programación y errores detectables de manera temprana en tiempo de compilación.

En este proyecto se utilizará TypeScript como lenguaje de programación de la parte Frontend para contar con todas las ventajas de un lenguaje tipado y se hará uso de las más modernas capacidades del estándar ECMAScript para tener un código limpio y elegante.

5.1.4. Framework Web

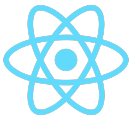
Los tiempos de implementar aplicaciones web sin la ayuda de un framework se acabaron hace tiempo. Con el surgimiento de **AngularJS** se revolucionó el panorama de desarrollo web terminando con el uso de **jQuery** como principal herramienta para modificar el DOM del website.

Los nuevos frameworks web ofrecen una manera óptima de diseñar una aplicación web sin los problemas de rendimiento que provoca la constante modificación de la estructura de la página. Así marcan una manera ordenada, clara y eficiente de conexión entre el código JavaScript con la lógica y la vista en HTML.

Actualmente existen tres principales alternativas con un enfoque diferente aunque con similares objetivos: React, Vue y Angular.

5.1.4.1. React

React es una librería JavaScript que permite la **construcción sencilla y eficiente de interfaces de usuario** como aplicaciones web de una sola página.



Entre sus principales ventajas están que es declarativo, basado en componentes y sencillo. Para ello Rect ofrece una solución centrada principalmente en la Vista en un contexto MVC (Modelo-Vista-Controlador).

Entre las principales características de este framework se encuentran el Virtual DOM, que permite determinar que partes del DOM han cambiado y actualizarlo de la manera más eficiente, los ciclos de vida de los componentes, el estado y propiedades de estos, y la sintaxis JSX que permite un código más legible con experiencia similar a HTML.

5.1.4.2. Vue

Vue es un framework JavaScript open-source basado en el modelo **MVVM** (Model-view-viewmodel) para construir interfaces de usuario web de una sola página.

Ofrece un rendimiento verdaderamente bueno utilizando y ofreciendo para ello componentes, plantillas, reactividad, transiciones y funcionalidades de enrutado.

Tiene una sintaxis sencilla y es fácil de aprender y de utilizar para nuevos usuarios. Soporta TypeScript como lenguaje. En contra de este lenguaje está su "juventud" frente al resto de frameworks que están más asentados, y que su flexibilidad en la estructura puede llevar a problemas para programadores menos expertos.

5.1.4.3. Angular

Angular es el framework actualmente más extendido y permite la implementación sencilla de **webs con alto rendimiento, muy modulares** y con capacidad de crecer sin degradación.

Este framework open-source de aplicaciones web ha sido desarrollado por Google y ha surgido como evolución al antiguo framework ahora conocido como "AngularJS". Si bien mantiene su nombre, la sintaxis y el core son completamente diferentes, por lo que se considera un framework distinto.

Angular ofrece un framework web muy completo que cuenta con muchas funcionalidades para la construcción de páginas web como pueden ser formularios, enrutado web, seguridad, internacionalización, doble databinding, el uso de Web Workers, animaciones, cliente http, renderizado en servidor, etc. Cuenta con un sistema de inyección de dependencias y basa su estructura en módulos con lo que es fácilmente extensible y muy reutilizable.

El lenguaje de programación recomendado para utilizar Angular es TypeScript.

Si bien cualquiera de las tres opciones expuestas es válida, en este proyecto se utilizará **Angular** como framework para la implementación de la web.

5.1.5. Librería de componentes: Angular Material

Angular Material es una librería de componentes para mejorar el aspecto visual de la aplicación.

Mediante el uso de esta librería podremos diseñar páginas web rápidas y consistentes compatibles con los navegadores web actuales. Es versátil y configurable permitiendo personalizar por ejemplo las paletas de colores al gusto.

Está optimizado para su uso con el framework Angular y cumple con las directrices de "**Material Design**" para la construcción de interfaces intuitivas y elegantes.

5.2. Backend

5.2.1. Lenguaje de programación: Java

Se utilizará **Java** como lenguaje de programación del Backend.

Este lenguaje fue comercializado por primera vez en 1995 por **Sun Microsystem** y actualmente es propiedad de **Oracle**. Es un lenguaje muy extendido y utilizado en gran variedad de dispositivos y plataformas como centros de datos, consolas y teléfonos móviles. Es así uno de los lenguajes de programación más populares para aplicaciones cliente-servidor.

Entre las bases del lenguajes está la orientación a objetos, la multiplataforma, el trabajo en red, la ejecución remota y la facilidad de uso.

Al igual que JavaScript este lenguaje ha evolucionado mucho en los últimos años. Con Java 8 se dió un gran salto y se renovó el lenguaje añadiendo funcionalidades clave orientadas a facilitar la **programación funcional** como son las expresiones Lambda y los Streams.

Java basa su ejecución en la **Java virtual machine (JVM)**. Esta permite a los distintos tipos de computadores y dispositivos ejecutar programas compilados a **Java bytecode**. Esto hace que los programas Java sean multiplataforma y las implementaciones sean independientes de las particularidades del hardware de cada plataforma.

5.2.2. Framework API

En el panorama actual **Spring** es el framework Java más utilizado para la construcción de aplicaciones web, pero se ha querido analizar y utilizar para este proyecto una alternativa llamada **Micronaut**. A continuación se analizan y exponen ambas alternativas.

5.2.2.1. Spring

Es el framework más completo para la implementación de aplicaciones sobre JVM.

Cuenta con distintos módulos para dar solución a los principales problemas de la construcción de un aplicación completa:

- Spring Boot: diseñado para la construcción y ejecución de aplicaciones lo más rápido posible ofreciendo un sistema de configuraciones automáticas, para la rápida implementación de aplicaciones listas para producción.
- Spring Cloud: basado en la suite de Netflix ofrece una solución para el despliegue de una arquitectura de microservicios implementando patrones de resiliencia, coordinación, autodescubrimiento, etc.
- Spring Web MVC: Solución tradicional para la construcción de web usando la API de Servlets de JEE.
- Spring WebFlux: Solución con un enfoque reactivo en la construcción web sin bloqueo de hilos en su ejecución.
- Spring Security: facilita la protección en aplicaciones de una manera sencilla, cumpliendo los estándares de seguridad para autenticar y autorizar a los usuarios.
- Spring Data: ofrece una solución para el acceso y manejo de datos persistentes en base de datos relacionales y no relacionales.

- Otros: Spring Integration, Spring Cloud Data Flow, Spring Batch, Spring AMQP.

5.2.2.2. Micronaut

Micronaut es un moderno framework basado en JVM para construir aplicaciones modulares y testeables que permite el uso del paradigma serverless. Este framework ofrece tiempos de arranque muy buenos permitiendo desarrollos rápidos y eficientes de clientes reactivos http.

Es un proyecto open-source desarrollado por los creadores del framework Grails, basando su diseño en la construcción de aplicaciones con arquitectura de microservicios.

Soporta Java, Groovy y Kotlin como lenguajes de programación. Es muy **liviano**, lo que permite la creación de aplicaciones que **consumen muy poca memoria** RAM y arrancan increíblemente rápido. Es bastante similar a Spring Boot en su sintaxis y estructura, utilizando un sistema de inyección de dependencias inspirado en el de Spring.

Soporta su ejecución como funciones "lambda" o "**serverless**" lo cual es un servicio de cómputo que permite la ejecución del código sin necesidad de aprovisionamiento ni administración de servidores, ejecutando las instancias necesarias y escalando automáticamente. El servicio más famoso de serverless actualmente es ofrecido por Amazon Web Services como AWS Lambda.

Micronaut ofrece de serie solución a la mayoría de problemas en la construcción de una aplicación web: configuración, seguridad, métricas, trazabilidad, "Circuit Breaker", controladores web, cliente http con balanceo en cliente, acceso a base de datos, GORM, manejo de errores, validación de datos, CORS, etc.

Entre las principales diferencias con Spring están que maneja toda la información en tiempo de compilación arrancando así de manera mucho más rápida al no hacer uso apenas de reflexión ni proxies en tiempo de ejecución.

Otra gran ventaja de este framework es que el bajo uso de reflexión permite utilizar **GraalVM** como máquina virtual. Esta permite reducir considerablemente el consumo de RAM y mejora bastante los tiempos de ejecución compilando a código nativo de la máquina.

5.2.3. Arquitectura

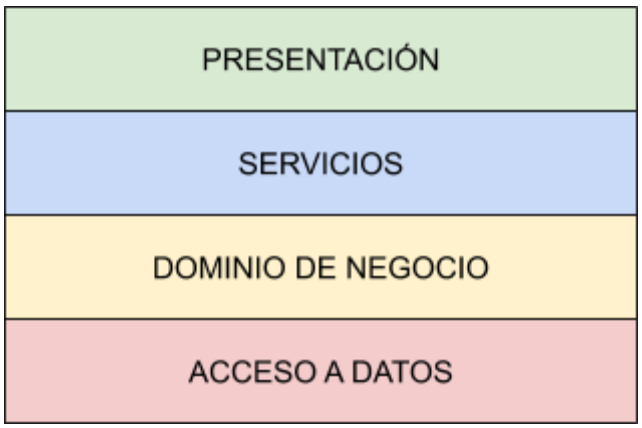
A continuación se va a analizar algunas de las arquitecturas más utilizadas y aprobadas por la comunidad para la construcción de aplicaciones complejas. Se verán las ventajas y diferencias entre cada una de ellas para seleccionar la solución que se considere más correcta.

5.2.3.1. Arquitectura multicapa

La arquitectura más básica y comúnmente utilizada en el desarrollo de aplicaciones web es la arquitectura multicapa. Este concepto puede referirse "multi-tier" o "multi-layer".

La arquitectura "multi-tier" propone una separación del sistema en **varias capas físicas**, separando en distintas máquinas las distintas partes del sistema. Esto ocurre por ejemplo en una arquitectura cliente-servidor. En este apartado no es este el caso que se quiere analizar, pues estamos buscando una solución para la organización y relación lógica de los componentes y no para su distribución física.

La arquitectura "multi-layer" busca la **separación lógica en capas** de un sistema software. Así propone la organización jerárquica de las clases una encima de otras con dependencias siempre hacia abajo. De esta manera cada capa solo dependerá de las capas inferiores y nunca de las superiores.

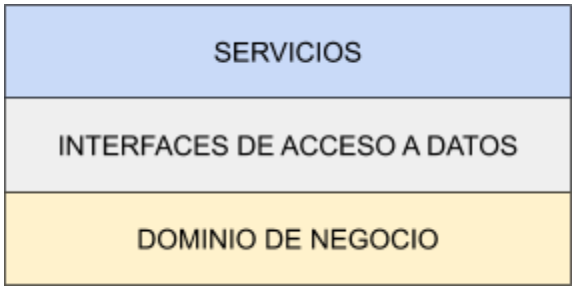


La principal desventaja de una arquitectura como esta es que todos los niveles dependen para el "acceso a datos" (el sistema de persistencia) del centro o último nivel del sistema.

5.2.3.2. Onion architecture

Esta arquitectura intenta solucionar el problema anteriormente expuesto usando el principio de **inversión de dependencias** y dejando en el centro y última capa de dependencias el propio modelo de **dominio**.

Para ello se utilizan interfaces de repositorio que permiten desligar el comportamiento del almacenamiento de la implementación. Así la lógica de negocio usa estas interfaces y solo en tiempo de ejecución tendrá las implementaciones habiendo roto esta dependencia y permitiendo a la aplicación ser más independiente de las tecnologías de acceso a los datos.

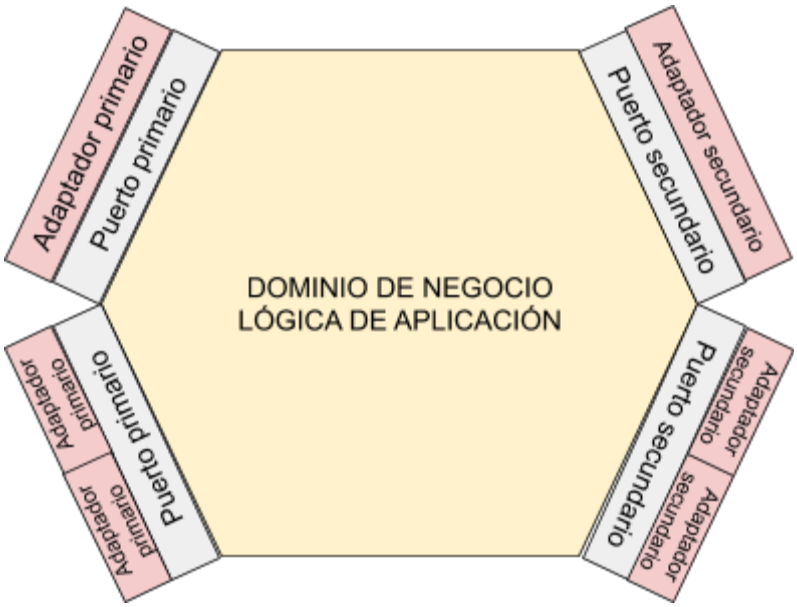


5.2.3.3. Arquitectura hexagonal

Con el enfoque de "Onion architecture" la arquitectura hexagonal define un modelo más estricto de organización que de nuevo pone en el centro de la arquitectura el **dominio** de negocio y toda su lógica propia.

De esta manera define las fronteras entre la lógica propia del negocio o aplicación y cualquier tipo de agente externo ya sea de entrada o de salida.

Diferencia tres partes en su arquitectura: la lógica central de la aplicación, los puertos y los adaptadores.



Los **puertos** definen sólo la interacción con el exterior exponiendo objetos del dominio. Por tanto el core no conoce los objetos externos ni toda la lógica de transformación a estos objetos eliminando toda dependencia con tecnología, APIs o componentes de terceros.

Los **puertos primarios** definen como se expone el sistema al exterior para que se comunique con él un usuario, cliente o aplicación, Mientras que los **puertos secundarios** definen que requiere el sistema de terceros.

Los **adaptadores** son las implementaciones de estas comunicaciones con el exterior he incluyen cualquier tipo de transformación necesaria de los datos.

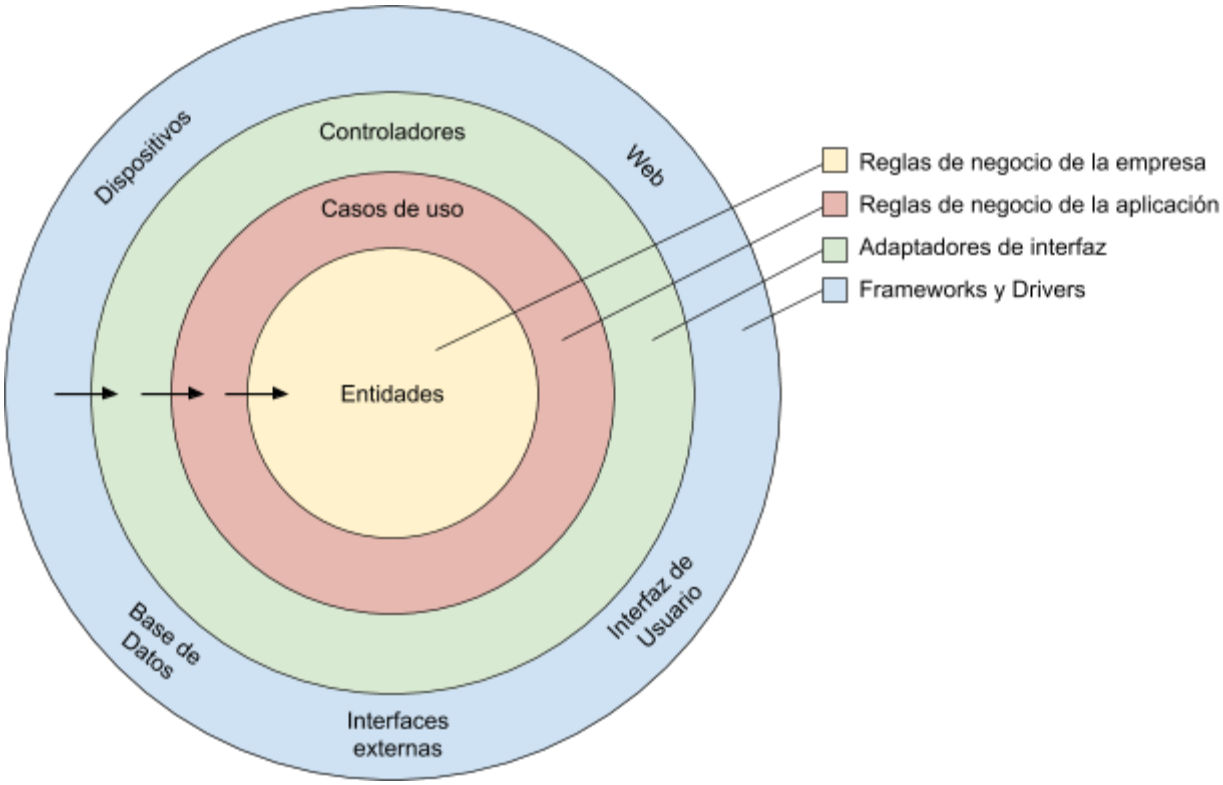
Un puerto puede tener más de un adaptador. Así un caso de uso o funcionalidad de de una aplicación puede ser expuesto de varias maneras al exterior por ejemplo como un endpoint REST y con un websocket. Con los puertos secundario del mismo modo se podrían implementar dos adaptadores para resolver el acceso a un tipo de datos permitiendo con la misma interfaz acceder a este datos por ejemplo desde una caché o desde una base de datos.

5.2.3.4. Clean Architecture

Creada por Uncle Bob, uno de los referentes en ingeniería de software del panorama actual, esta arquitectura es una evolución de la arquitectura hexagonal con nombres un poco más descriptivos y algo más especificada internamente.

Cuenta con **cuatro capas**:

- En el centro y como último nivel de dependencia están las propias **entidades de dominio**.
- En la siguiente capa estarían los **casos de uso** que es la lógica propia del negocio o aplicación. En esta misma capa estarían los "puertos" de la arquitectura hexagonal llamándose en este caso "Use Case Input Port" y "Use Case Output Port". La implementación de los de entrada si es en este caso parte de esta capa llamándose "Use Case Interactor".
- En la siguiente capa estarían los **adaptadores** que podrían ser controladores, presentadores, accesos a terceros, etc
- Define una última capa en la que coloca los dispositivos de terceros o **externos** con los que nos conectamos y esta capa no es realmente parte del código desarrollado.



Como se ha visto "Clean Architecture" se podría ver como el último paso de una evolución natural en la organización de aplicaciones.

5.2.4. Programación reactiva

Para este proyecto se ha querido aprender un **nuevo enfoque de programación** al que no se estaba acostumbrado en la parte backend y que es actualmente una tendencia, la programación reactiva.

Este es un **paradigma** en el que se trabaja con los datos de manera **asíncrona** utilizando **promesas** de flujos de datos finitos o infinitos.

Este paradigma surge de los problemas que tienen los paradigmas bloqueantes:

- Desaprovechamiento de CPU debido al alto uso de I/O
- Alto uso de memoria debido al uso abusivo de hilos (threads).
- Ineficiencia por interacciones bloqueantes que causan llamadas constantes al sistema bloqueando hilos.

Java ha evolucionado en los últimos años para facilitar la programación funcional y cuenta ahora con más facilidades para escribir código reactivo de manera más limpia, pero aun así se necesita alguna librería extra que mejore la legibilidad y ofrezca operadores comunes para el trabajo con el flujo de datos asíncrono. La librerías más utilizadas actualmente y adoptadas por los grandes frameworks son "RxJava" y "Project Reactor".

5.2.5. API Rest

API (Application Programming Interface, o Interfaz de Programación de Aplicaciones) es un conjunto de reglas que dos o más sistemas pueden utilizar para comunicarse entre ellos.

REST (Representational State Transfer, o Transferencia de Estado Representacional) se utiliza para describir una interfaz que se utilice para comunicar dos sistemas entre sí vía HTTP. Surgió en el año 2000 y es una alternativa en auge a otros protocolos estándar de intercambio de datos como SOAP (Simple Object Access Protocol), que disponen de una gran capacidad pero también mucha complejidad. A veces es preferible una solución más sencilla de manipulación de datos como REST.

REST es utiliza para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON; siendo este último el más recomendable.

Las principales características de un API REST son:

- **Protocolo cliente/servidor sin estado**: cada petición HTTP contiene toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor

necesiten recordar ningún estado previo para satisfacerla. Aunque esto es así, algunas aplicaciones HTTP incorporan memoria caché. Se configura lo que se conoce como protocolo cliente-caché-servidor sin estado: existe la posibilidad de definir algunas respuestas a peticiones HTTP concretas como cacheables, con el objetivo de que el cliente pueda ejecutar en un futuro la misma respuesta para peticiones idénticas. De todas formas, que exista la posibilidad no significa que sea lo más recomendable.

- Las **operaciones** más importantes relacionadas con los datos en cualquier sistema REST y la especificación HTTP son cuatro: **POST (crear), GET (leer y consultar), PUT (editar) y DELETE (eliminar)**.

- Los objetos en REST siempre se manipulan a partir de la URI. Es la **URI** y ningún otro elemento el identificador único de cada recurso de ese sistema REST. La URI nos facilita acceder a la información para su modificación o borrado, o, por ejemplo, para compartir su ubicación exacta con terceros.

- **Interfaz uniforme**: para la transferencia de datos en un sistema REST, este aplica acciones concretas (POST, GET, PUT y DELETE) sobre los recursos, siempre y cuando estén identificados con una URI. Esto facilita la existencia de una interfaz uniforme que sistematiza el proceso con la información.

- **Sistema de capas**: arquitectura jerárquica entre los componentes. Cada una de estas capas lleva a cabo una funcionalidad dentro del sistema REST.

- **Uso de hipermedios**: hipermedia es un término acuñado por Ted Nelson en 1965 y que es una extensión del concepto de hipertexto. Ese concepto llevado al desarrollo de páginas web es lo que permite que el usuario puede navegar por el conjunto de objetos a través de enlaces HTML. En el caso de una API REST, el concepto de hipermedia explica la capacidad de una interfaz de desarrollo de aplicaciones de proporcionar al cliente y al usuario los enlaces adecuados para ejecutar acciones concretas sobre los datos.

Principales ventajas de usar un API REST:

- **Separación entre el cliente y el servidor**: el protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos. Eso tiene algunas ventajas cuando se hacen desarrollos. Por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.

- **Visibilidad, fiabilidad y escalabilidad**: la separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de desarrollo puede escalar el producto sin excesivos problemas. Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta. Esta separación facilita tener en servidores distintos el front y el back y eso convierte a las aplicaciones en productos más flexibles a la hora de trabajar.

- La API REST siempre es **independiente del tipo de plataformas o lenguajes**: la API REST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo. Con una API REST se pueden tener servidores PHP, Java, Python o Node.js. Lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

API First

Hay varias formas de crear una API, la más "tradicional" sería hacer el desarrollo y después, con el desarrollo terminado, crear un API para que otros consuman los servicios.

API first propone justo lo contrario, crear primero la definición del API para después empezar con el desarrollo. Sirve para que los equipos **planifiquen, organicen y compartan una primera visión** de lo que van a desarrollar. Hay varias formas de documentar una API, por ejemplo RAML, API Blueprint o Swagger.

Algunas de las ventajas de trabajar con API First son:

- Los equipos de desarrollo pueden **trabajar en paralelo**. Al crear primero el API pueden empezar los desarrollos tanto el equipo que la implementa como los equipos que la consumen.
- **Detectar** nuevas **necesidades antes** del desarrollo. De esta forma equipos que vayan a consumir los servicios expuestos en este proyecto, pueden detectar necesidades antes de empezar con el desarrollo. Además de esta forma se puede conseguir reducir los costes del desarrollo. Incluso, en algunos casos, puede ser interesante involucrar a los consumidores en el desarrollo del API.
- **Buena experiencia** para el desarrollador. Los consumidores de una API suelen ser desarrolladores. Las APIS bien diseñadas y documentadas hacen que el desarrollo sea más positivo porque hace que el código sea más reutilizable y hace que sea más probable no tener que hacer modificaciones en el desarrollo.

5.2.6. OpenApi/Swagger

"OpenAPI", originalmente era conocida como "Especificación Swagger", es una especificación estándar ampliamente adoptada por la industria para definir APIs modernas.

Esto permite definir una interfaz públicamente accesible para proveer a los desarrolladores acceso a aplicaciones software o web services.

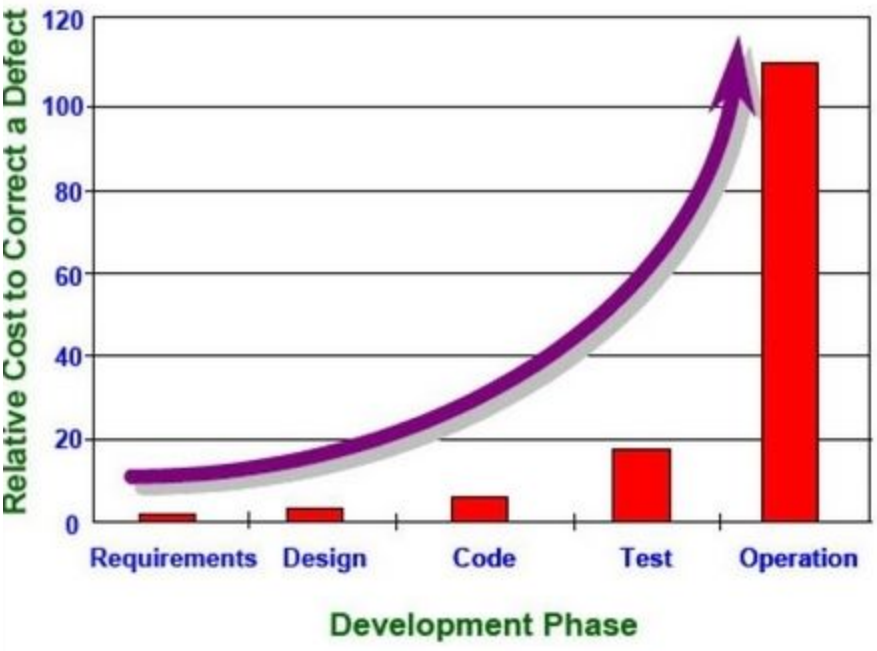
Este tipo de especificación es una interfaz legible para máquinas con el objetivo de **describir, producir, consumir y visualizar RESTful web services**. Esta es agnóstica al lenguaje utilizado en la implementación de la API.

Micronaut ofrece soporte para producir automáticamente la especificación Swagger en formato YAML en tiempo de compilación. Se utilizan anotaciones y comentarios javadoc que se incluyen en el propio código para definir los detalles e información extra a incluir en la definición formal del API generada.

5.2.7. Testing

Los tests son parte fundamental del desarrollo de software. Es importante hacer pruebas para poder **encontrar fallos** y comportamientos extraños antes de la promoción del software a entornos superiores (como el productivo).

El **coste de los errores** (bugs) suele ser mayor cuanto más tarde sea detectado. Si se piensa en un caso durante la especificación de requerimientos, o el diseño será lo óptimo. Si se aborda durante el desarrollo será mejor que si ya se ha desplegado en entornos posteriores como QA/Staging. Y en el peor de los casos se detectará el problema en producción requiriendo mucho más esfuerzo y urgencia para resolverlo, y con las consecuentes pérdidas que pueda causar.



Un código con baja cobertura de tests implica un código poco robusto y disminuirá su confianza en él. Por otro lado un código con **buena cobertura** permitirá refactorizar con suficiente nivel de **confianza** y por tanto el código estará más limpio y será más **mantenible** en el tiempo.

Existen distintos tipos de pruebas en función de su utilidad:

- **Pruebas unitarias**: para validar la lógica, comportamiento y funcionalidad de manera independiente.
- **Pruebas de integración**: verifican el comportamiento integrado de todo el flujo de la aplicación.
- **Pruebas de validación**: comprueban que se cumplan los requisitos del sistema.
- **Prueba de sistema**: comprueban la integración con el entorno donde se desplegará.

- **Pruebas de aceptación**: comprueban que se cumplan los requisitos de los usuarios.
- **Pruebas de regresión**: se utilizan para comprobar todas las pruebas existentes después de hacer un cambio volviendo a ejecutarlas.

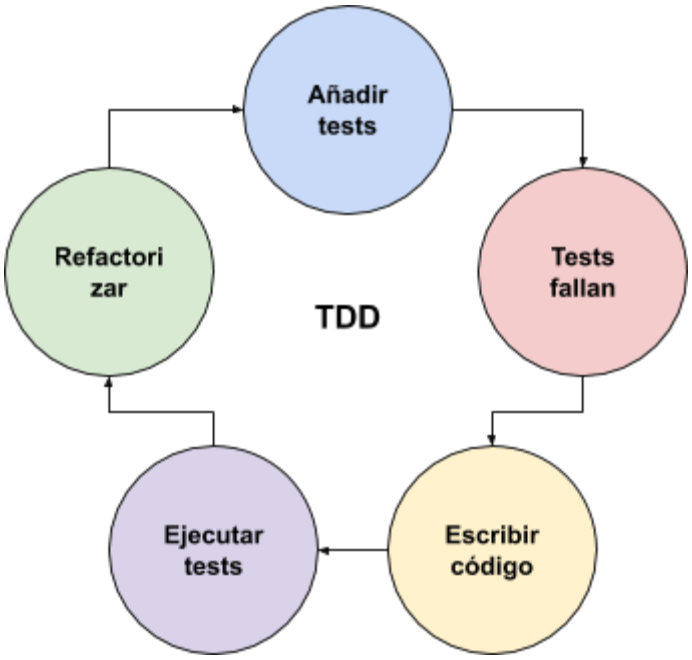
Un buen test unitario sigue los principios **FIRST**:

- **Fast**: pequeño para que se ejecute en un corto periodo de tiempo.
- **Isolated**: no deben estar acoplados. Se pueden ejecutar en cualquier orden con los mismos resultados.
- **Repeatable**: deben soportar su ejecución más de una vez sin cambiar el resultado ni el estado del sistema.
- **Self-validating**: deben indicar claramente cuál fue su resultado (Pass/fail/warning).
- **Timely**: se escriben inmediatamente antes que el código que quiere que evalúe.

Este último punto es uno de los dos principales pilares de TDD. El **desarrollo guiado por pruebas de software** o Test-driven development (**TDD**) es una práctica de ingeniería de software que involucra otras dos prácticas:

- **Escribir las pruebas primero** (Test First Development)
- **Refactorización** (Refactoring)

Así define un flujo de trabajo iterativo en el que primero se escriben los tests que definen un problema encontrado, se comprueba que los tests fallan, luego se programa un test que resuelve el problema, se comprueba que los tests pasen, luego se refactoriza el código para dejarlo tan limpio como sea posible y se vuelve a comprobar que los test siguen pasando.



De esta manera se logra que la **cobertura de código** sea muy alta como consecuencia y no como meta. El código estará tan **limpio y ordenado** como sea posible y los desarrolladores tendrán **confianza para refactorizar** sin miedo a romper nada. Indirectamente se conseguirá un **mejor diseño** pues un diseño testable es **más desacoplado y reutilizable**. Los tests además serán una **documentación** de cada funcionalidad.

5.2.8. Persistencia

Dentro de las bases de datos existen dos tipos, las relacionales y no relacionales. Dependiendo de la necesidad de cada proyecto y de cómo se quieran utilizar los datos tendrá más sentido usar una u otra.

5.2.8.1. Bases de datos relacionales

Una base de datos relacional se basa en un conjunto de **tablas interrelacionadas**. Es un modelo **sencillo** y a la vez muy **potente**. Los datos se representan en tablas, cada tabla tiene un ID único llamado clave. Cada fila es una entrada nueva de datos y las

columnas contienen los atributos de los datos. Las tablas se relacionan entre sí por las claves.

Respecto a las no relacionales, las relacionales son más restrictivas en cuanto al tipo de datos y la integridad de los datos. Favorece la normalización por ser más comprensible y aplicable. Es decir, es una base de datos relacional es una buena candidata cuando los datos deben ser consistentes sin dar posibilidad al error.

Algunas de las bases de datos más utilizadas son:

- Oracle
- MySQL
- Microsoft SQL Server
- PostgreSQL
- DB2

5.2.8.2. Bases de datos no relacionales

La principal característica de las bases de datos no relacionales es que no tienen una definición de atributos preestablecida, es decir cada documento o registro de datos puede tener una **estructura de datos diferente**, por lo que suelen ser bases de datos mucho más **abiertas y flexibles**. Permiten adaptarse a necesidades de proyectos mucho más fácilmente que los modelos de Entidad Relación. En cuanto a sus ventajas, son sumamente **escalables**, se pueden editar al mismo tiempo que el código desarrollado, necesitan menos recursos, son eficaces en análisis de grandes cantidades de datos en modo lectura, no se necesita invertir tanto tiempo en diseñar una estructura ya que podemos almacenar datos no estructurados y manipularlos libremente.

Algunas de las bases de datos más utilizadas son:

- MongoDB

- Apache Cassandra
- CouchDB
- Redis

5.2.8.3. MongoDB

MongoDB es una base de datos distribuida (NoSQL) y de código abierto. Es una **base de datos documental**, lo que significa que almacena datos en forma de documentos tipo JSON. Los documentos son almacenados en formato BSON, que no es más que una extensión de JSON (JavaScript Object Notation) que se representa de manera binaria. Es muy potente, porque incluso si los datos están anidados dentro de los documentos JSON, seguirá siendo consultable e indexable.

Al ser una base de datos distribuida permite la fragmentación horizontal de los datos, también llamado Sharding. Esto consiste en partir la cantidad total de documentos de una colección entre todos los fragmentos disponibles.

5.3. Computación en la nube

El paradigma de computación en la nube, en inglés cloud computing, ofrece **servicios de computación a través de Internet**. Esto permite disfruta de servicios sin gestión activa directa por parte del usuario, principalmente almacenamiento de datos y capacidad de cómputo.

Actualmente los mayores proveedores de computación en la nube son propiedad de Amazon, Microsoft y Google:

- Amazon Web Services: https://aws.amazon.com/
- Azure: https://azure.microsoft.com/
- Google Cloud Platform: https://cloud.google.com/

Por su parte Salesforce tiene un plataforma de computación en la nube llamada **Heroku**. Esta es muy sencilla de utilizar y especialmente apropiada para proyectos pequeños que requieren un rápido "time-to-market".

Heroku será la alternativa utilizada para el despliegue del proyecto en producción.

5.4. Integración y despliegue continuo

La **integración continua** es una práctica de desarrollo de software que potencia la integración del código en el repositorio con la mayor frecuencia posible. Además automatiza la ejecución de pruebas y construcción de artefactos con cada integración.

Con este tipo de prácticas se mejora la **productividad** del desarrollo, se promueve la **detección** y resolución **de errores** en etapas tempranas y se **reducen los tiempos entre entregas**.

Durante la integración continua se pueden incluir fases extra de revisión de calidad y seguridad como pueden ser análisis de código estático o análisis de vulnerabilidades sobre dependencias.

Dando un paso más sobre este paradigma se encuentra el **despliegue continuo**. Este implica que tras la ejecución de los test y la construcción del artefacto la aplicación sea desplegada automáticamente.

Heroku ofrece pipelines de integración continua fáciles de configurar totalmente integrados con sus repositorios git. De este modo la ejecución de un push de código a repositorio conllevará la ejecución de la compilación, tests y despliegue de la aplicación de manera totalmente automática.

5.5. Análisis de código estático

En desarrollo de software es útil la utilización de herramientas de análisis dinámico como los analizadores estáticos de código fuente. Esto permitirá **mejorar la calidad del código detectando problemas** típicos de complejidad, nombramiento, orden y seguridad.

También aportan **métricas** útiles para ver el estado actual de un proyecto pudiendo indicar el número de errores/warnings detectados y la deuda técnica que estos tienen.

5.5.1. TSLint

TSLint es una herramienta de **análisis estático** que comprueba el **código TypeScript** en busca de errores de legibilidad, mantenibilidad y funcionalidad. Es configurable de tal modo que puedes personalizar tus propias reglas, configuraciones y formatos de código.

5.5.2. SonarQube

SonarQube es una herramienta para gestionar la **deuda técnica**, para ello analiza: pruebas unitarias, complejidad, código duplicado, diseño, comentarios, estándares, reglas propias.. Es compatible con más de 25 lenguajes de programación. SonarQube proporciona a los desarrolladores una guía para ayudar a los equipos a conseguir un código de mayor calidad y más seguro. Está preparado para integrarse en proyecto con integración continua.

6. APLICACIÓN PRÁCTICA

6.1. Especificaciones / Requisitos

6.1.1. Definición básica del producto

Se implementará un sistema de juegos online con dos juegos con el que el usuario podrá **aprender a utilizar una API REST** y tendrá que **diseñar un bot** que automatice la ejecución de simulaciones (conjunto de partidas) con un **algoritmo** para superar las pruebas de la mejor manera posible.

Para cada juego se le explicará el **funcionamiento del juego** y se mostrará lo que tiene que hacer con **ejemplos de llamadas a la API** para ejecutar las partidas.

Se ofrecerá al usuario la documentación de la API que debe utilizar con formato **OpenAPI**.

Cada **simulación** contará con la ejecución de N **partidas** de las cuales se extraerá una puntuación final para poder valorar el resultado y medirse contra otros usuarios de la plataforma.

Se podrán visualizar las "simulaciones" ejecutadas viendo cada una de las partidas paso a paso con todas las decisiones ejecutadas y con un visualizador paso a paso basado en tecnología canvas.

6.1.2. Juegos

6.1.2.1. Juego 1 - Bull And Cows

Este juego es el más sencillo y con él se pretende familiarizar al usuario con el uso de la plataforma y la API Rest. El usuario tendrá que diseñar un algoritmo con el que resolver el juego en el mínimo número de pasos posible.

Este juego es mundialmente conocido y se puede encontrar mucha información al respecto en cualquier sitio online como en la propia wikipedia:

https://en.wikipedia.org/wiki/Bulls_and_Cows

Su funcionamiento es parecido al mítico juego "Mastermind". El oponente (en este caso el servidor) establece un código secreto que debe ser descubierto. Este está formado por una cantidad definida de cifras. El jugador podrá probar claves una por una y para cada intento recibirá una respuesta donde se indica el número de cifras acertadas y el número de cifras que existen en la clave pero no están en el lugar correcto, el resto no están en el código secreto a descubrir. La respuesta no informa de que posiciones son las que están bien o las que están mal, solo del número total de ocurrencias.

A continuación se expone una partida simple. Todas las llamadas deben incluir las credenciales del usuario en el header de la petición.

1. El usuario debe crear primero una simulación (batería de partidas para probar tu algoritmo). El servidor te responde con el número de partidas que se deben jugar en el campo "totalNumMatches" y el id de la simulación creada en el campo "simulationId". Se deberá pasar como parámetro en la url (como path param) el identificador del juego. En este caso "bullsandcows".

   `POST: /games/bullsandcows/simulations`

```
{
    "botName": "botname",
    "gameId": "bullsandcows",
    "simulationId": "9006e5af",
    "creationDateTime": "2020-04-25T10:14:22.542",
    "points": 0,
    "totalNumMatches": 51,
    "finishedMatches": 0
}
```

2. El usuario crea una partida.

`POST: /games/bullsandcows/simulations/9006e5af/matches`


```
{
    "botName": "botname",
    "gameId": "bullsandcows",
    "simulationId": "9006e5af",
    "matchNum": 1,
    "creationDateTime": "2020-04-25T11:57:30.024",
    "points": 0,
    "status": "PLAYING",
    "rules": {
        "maxTurns": 10,
        "numDigits": 2
    },
    "turns": []
}
```


Entre la información devuelta por esta petición podemos ver en el campo "matchNum" el número de partido asignado que se utilizará en llamadas posteriores, y las reglas de la partida en el json "rules". Entre las reglas podemos ver que la clave secreta consta de 2 cifras y el número máximo de turnos es 10. Internamente genera el código secreto el cual no es informado al cliente, ya que es el código que debe descubrir.

3. El usuario lanza su primer intento enviando la clave secreta "01". El servidor le informa de que ninguna de las cifras está en su sitio (bulls) y que ninguna está en el lugar equivocado (cows). Por tanto se pueden descartar los números 0 y 1.

POST /games/bullsandcows/simulations/9006e5af/matches/1/turns

Request body

```
{
  "combination": "01"
}
```

Response body

```
{
  "turnNum": 1,
  "creationDateTime": "2019-12-13T07:51:02.948",
  "action": {
    "combination": "01"
  },
  "turnResult": {
    "status": "PLAYING",
    "bulls": 0,
    "cows": 0
  }
}
```

4. El usuario lanza una segunda petición con la clave secreta "23". El servidor informa de que ninguna cifra está en su sitio (bulls) pero una cifra está en el lugar equivocado (cows). Con esto tenemos que suponer que hay dos opciones:

    a. El 2 no está en la clave y el 3 va en la primera posición.

    b. o el 3 no está en la clave y el 2 va en la segunda posición.

POST /games/bullsandcows/simulations/9006e5af/matches/1/turns

Request body

```
{
  "combination": "23"
}
```

Response body

```
{
  "turnNum": 2,
  "creationDateTime": "2019-12-13T07:51:02.948",
  "action": {
    "combination": "23"
  },
  "turnResult": {
    "status": "PLAYING",
    "bulls": 1,
    "cows": 1
  }
}
```

5. El usuario lanza una tercera petición con la clave secreta "34". El servidor informa de que dos cifras están en su sitio (bulls) y por tanto ya se ha acertado la clave completa que era de 2 cifras según las reglas iniciales. Te informa también de que has ganado la partida.

   Si el usuario no descubriera el código secreto a tiempo en su décimo turno se le informaría en la respuesta de que ha perdido la partida.

POST /games/bullsandcows/simulations/9006e5af/matches/1/turns

Request body

```
{
  "combination": "34"
}
```

Response body

```
{
  "turnNum": 3,
  "creationDateTime": "2019-12-13T07:51:02.948",
  "action": {
    "combination": "34"
  },
  "turnResult": {
    "status": "PLAYING",
    "bulls": 2,
    "cows": 0
  }
}
```

6. Volver al punto 2 hasta alcanzar el número de partidas indicadas en la simulación.

6.1.2.2. Juego 2 - Laberinto

Este juego es un laberinto en el que al usuario en cada turno se le dice lo que tiene alrededor (pared, camino libre o salida) y como en todo laberinto el objetivo es encontrar la salida ejecutando movimientos turno a turno en las cuatro direcciones (Arriba, abajo, derecha, izquierda).

Cabe destacar que las peticiones son bastante parecidas entre todos los juegos, con esto conseguimos que sea más fácil para el usuario jugar a un segundo juego y que la API sea más amigable.

A continuación se expone el inicio de una partida. Todas las llamadas deben incluir las credenciales del usuario en el header de la petición.

1. El usuario debe crear primero una simulación (batería de partidas para probar tu algoritmo). El servidor te responde con el número de partidas que se deben jugar en el campo "totalNumMatches" y el id de la simulación creada en el campo "simulationId". Se deberá pasar como parámetro en la url (como path param) el identificador del juego. En este caso "labyrinth".

   `POST: /games/labyrinth/simulations`


```
{
    "botName": "botname",
    "gameId": "labyrinth",
    "simulationId": "fd76f791",
    "creationDateTime": "2020-04-25T10:14:22.542",
    "points": 0,
    "totalNumMatches": 50,
    "finishedMatches": 0
}
```


2. El usuario crea una partida.

   `POST: /games/labyrinth/simulations/fd76f791/matches`


```
{
    "botName": "botname",
    "gameId": "labyrinth",
    "simulationId": "fd76f791",
    "matchNum": 1,
    "creationDateTime": "2020-04-25T10:21:03.019",
    "points": 0,
    "status": "PLAYING",
    "rules": {
        "maxTurns": 200,
        "mapWidth": 10,
        "mapHeight": 10,
        "viewRange": 1
    },
    "turns": [
        {
            "turnNum": 1,
            "creationDateTime": "2020-04-25T10:21:03.031",
```

```
    "action": {},
    "turnResult": {
        "status": "PLAYING",
        "surroundingCells": [
            {
                "coordinate": {
                    "x": 0,
                    "y": 0
                },
                "exit": false,
                "topIsWall": true,
                "rightIsWall": false,
                "bottomIsWall": true,
                "leftIsWall": true
            },
            {
                "coordinate": {
                    "x": 0,
                    "y": 1
                },
                "exit": false,
                "topIsWall": true,
                "rightIsWall": false,
                "bottomIsWall": false,
                "leftIsWall": true
            },
            {
                "coordinate": {
                    "x": 1,
                    "y": 0
                },
                "exit": false,
                "topIsWall": true,
                "rightIsWall": true,
                "bottomIsWall": false,
                "leftIsWall": false
            },
            {
                "coordinate": {
                    "x": 1,
                    "y": 1
                },
                "exit": false,
                "topIsWall": false,
```

```
                    "rightIsWall": true,
                    "bottomIsWall": true,
                    "leftIsWall": false
                }
            ]
        }
    }
  ]
}
```

Entre la información devuelta por esta petición podemos ver en el campo "matchNum" el número de partido asignado que se utilizará en llamadas posteriores, y las reglas de la partida en el json "rules". Entre las reglas podemos ver el tamaño del tablero, el rango de visión del usuario y el número máximo de turnos para resolver el laberinto.

Al crear la partida, se crea automáticamente el primer turno. Este primer turno siempre empieza en la posición x:0, y:0. Además, como en cada turno, puedes conocer el estado del turno (PLAYING/jugando) y la información de la posición actual y las adyacentes. En el primer turno siempre tendrás información de las celdas más cercanas (El número de estas depende del parámetro "rules.viewRange").

Visualmente tendrías esta información:

3. El usuario juega un turno. En la petición de los turnos hay que pasar el id de la simulación y el número de partido como parámetros en la url y en el body se pasa únicamente la dirección en la que te quieres mover. En este caso solo puede ir en una dirección (hacia la derecha).

```
POST /games/labyrinth/simulations/fd76f791/matches/1/turns
```

```
{
        "direction": "RIGHT"
}
```

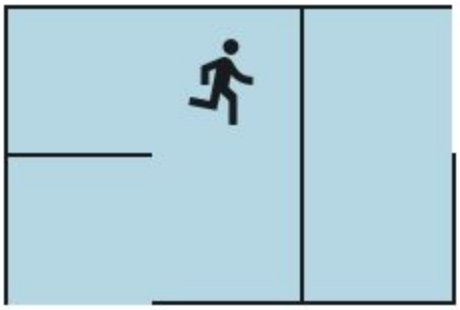
Respuesta del servidor:


```
{
    "turnNum": 2,
    "creationDateTime": "2020-04-25T10:54:07.694",
    "action": {
        "direction": "RIGHT"
    },
    "turnResult": {
        "status": "PLAYING",
        "surroundingCells": [
            {
                "coordinate": {
                    "x": 0,
                    "y": 0
                },
                "exit": false,
                "topIsWall": true,
                "rightIsWall": false,
                "bottomIsWall": true,
                "leftIsWall": true
            },
            {
                "coordinate": {
                    "x": 0,
                    "y": 1
                },
                "exit": false,
                "topIsWall": true,
```

```
            "rightIsWall": false,
            "bottomIsWall": false,
            "leftIsWall": true
        },
        {
            "coordinate": {
                "x": 1,
                "y": 0
            },
            "exit": false,
            "topIsWall": true,
            "rightIsWall": true,
            "bottomIsWall": false,
            "leftIsWall": false
        },
        {
            "coordinate": {
                "x": 1,
                "y": 1
            },
            "exit": false,
            "topIsWall": false,
            "rightIsWall": true,
            "bottomIsWall": true,
            "leftIsWall": false
        },
        {
            "coordinate": {
                "x": 2,
                "y": 0
            },
            "exit": false,
            "topIsWall": true,
            "rightIsWall": false,
            "bottomIsWall": false,
            "leftIsWall": true
        },
        {
            "coordinate": {
                "x": 2,
                "y": 1
            },
            "exit": false,
            "topIsWall": false,
            "rightIsWall": true,
```

```
                "bottomIsWall": true,
                "leftIsWall": true
            }
        ]
    }
}
```

Visualmente tendrías esta información:



Si intentamos ir en una dirección no válida el servidor nos devolverá una respuesta donde indica que se ha perdido la partida. Ponemos un ejemplo:

`POST /games/labyrinth/simulations/fd76f791/matches/1/turns`

```
{
        "direction": "TOP"
}
```

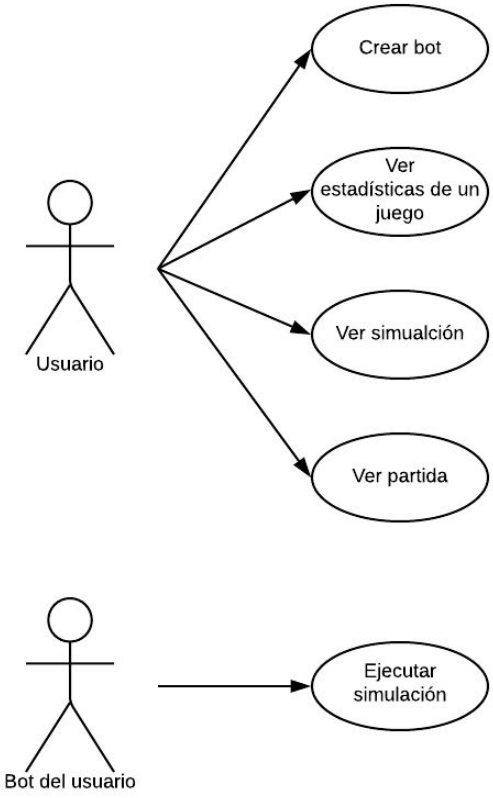
Respuesta del servidor:

```
{
    "turnNum": 2,
    "creationDateTime": "2020-04-25T10:39:37.202",
    "action": {
        "direction": "TOP"
    },
```

```
    "turnResult": {
        "status": "LOST",
        "details": "Invalid action direction"
    }
}
```

4. Se repite el apartado 3 de jugar turno hasta que el resultado de la partida sea WIN (has ganado) o LOST (has perdido). En ambos casos se da por finalizada la partida.
5. Se repite el apartado 2 de crear partida. Para finalizar una simulación hay que completar todas sus partidas, en este ejemplo serían 50 las partidas.

6.1.3. Diagramas de casos de uso



6.1.4. Casos de uso extendidos

Nombre	**Ejecutar simulación**
Descripción	Ejecución de una simulación
Actores	Bot del usuario
Precondiciones	El bot tiene que estar identificado en todas las llamadas
Flujo normal	1. Crear simulación

	2. Crear nueva partida 3. Jugar turno (Se repite hasta que termine la partida) 4. Volver al punto 2 si no ha terminado la simulación (Hasta el número de partidas que corresponda)
Excepciones	El movimiento del bot no es válido

Nombre	**Ver partida**
Descripción	Visualizar el detalle de una partida
Actores	Usuario
Precondiciones	El usuario tiene que estar logado en el portal web
Flujo normal	1. Entrar al portal web 2. Ir a la sección del juego en cuestión 3. Ir al listado de simulaciones 4. Seleccionar la simulación que queremos visualizar 5. Elegir una partida concreta de la simulación
Excepciones	El usuario no tiene simulaciones

6.2. Análisis

6.2.1. Diagrama entidad/relación

En blanco se pueden ver las clases globales independientes de los juegos:

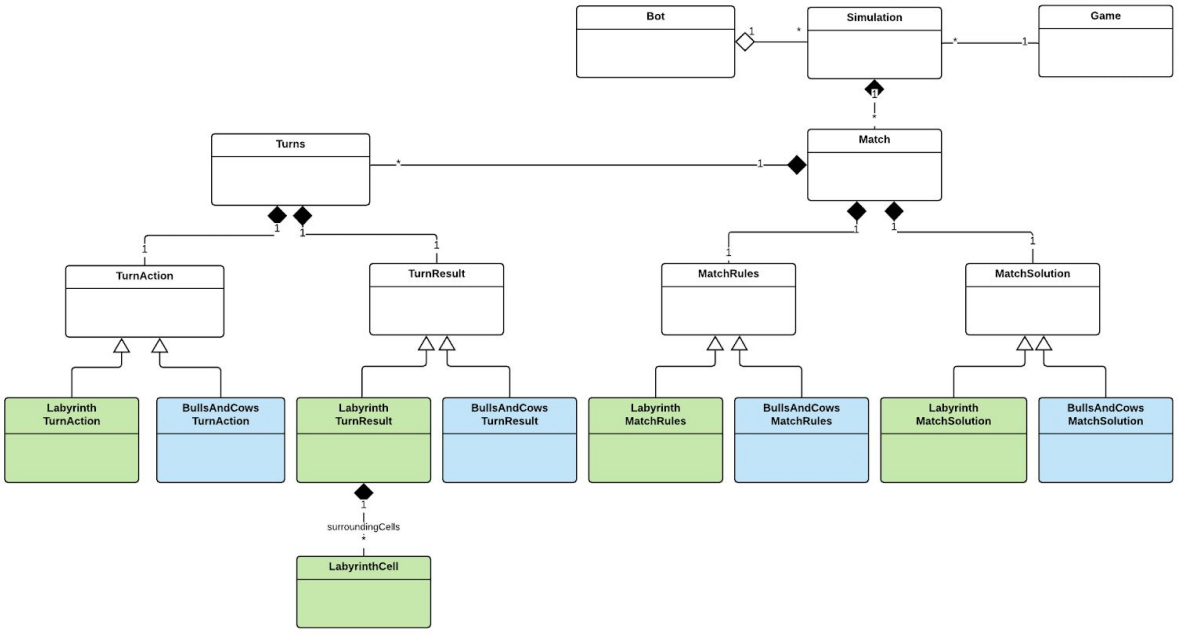
- **Bot**: usuario de la aplicación.

- **Game**: juego disponible en la plataforma.
- **Simulation**: simulación ejecutada por un bot sobre un juego. Esta incluirá varias partidas.
- **Match**: partida sobre un juego en una simulación concreta. Esta tendrá unas reglas, un resultado y un conjunto de turnos jugados.
- **Turn**: cada turno de una partida. Contiene una acción ejecutada por el bot y un resultado de la acción.

Por cada juego que implemente la plataforma se deben incluir las correspondientes clases heredadas con los atributos necesarios para ese juego de los objetos:

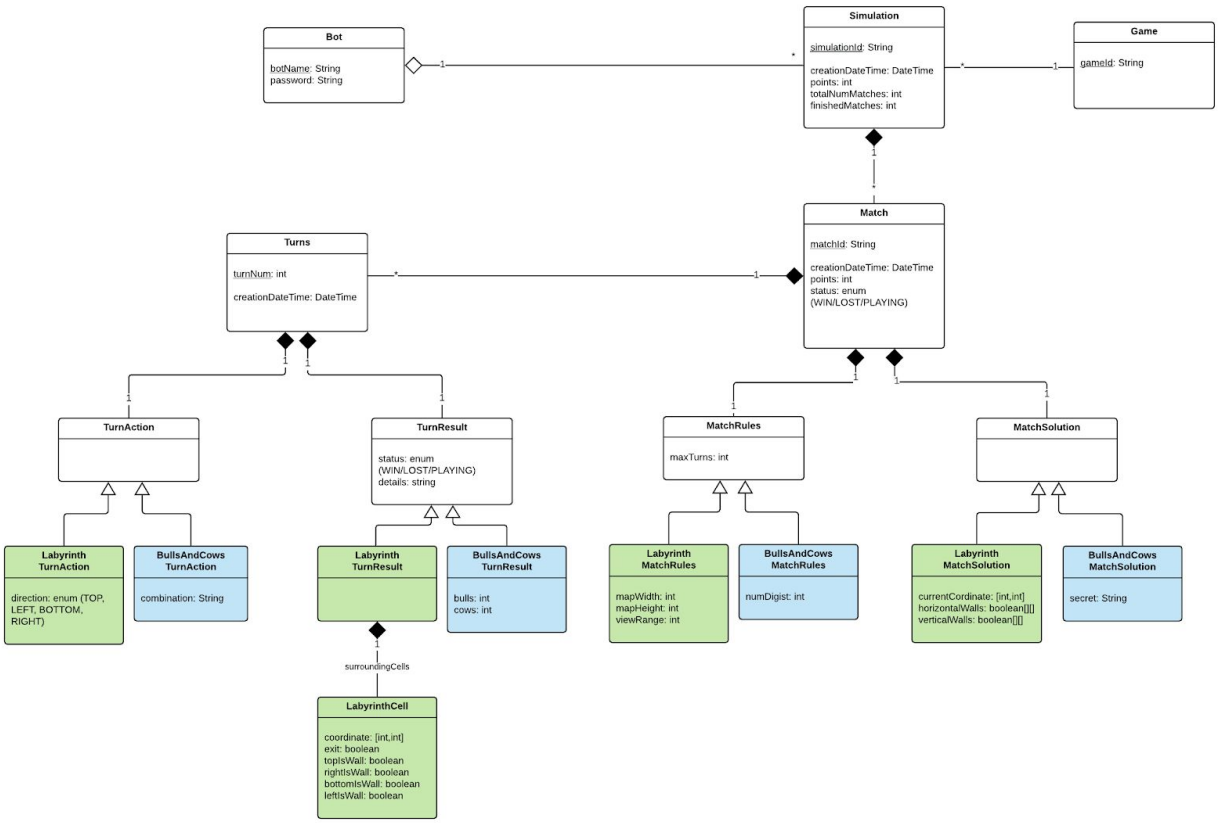
- **TurnAction**: acción ejecutada en un turno.
- **TurnResult**: resultado del turno.
- **MatchRules**: reglas del juego.
- **MatchSolution**: solución al juego.

En el diagrama se muestran en verde las clases correspondientes al juego “Laberinto” y en azul las del juego “Bulls And Cows”.



6.2.1. Diagramas de clases del dominio

A continuación se expone el diagrama de clases del dominio. Este incluye todos los atributos de las clases.



6.2.2. Autenticación

El sistema de autenticación estará basado en "Basic access authentication". Este sistema requiere que el usuario (cliente) envíe una cabecera "Authorization" con sus **credenciales** construida con el siguiente procedimiento:

- El **usuario y el password** son combinados con el carácter dos puntos (":").
- El resultado es **codificado** en **Base64**.

- Se concatena el método de autorización, en este caso "**Basic**" con un espacio y la cadena codificada generada.

Ejemplo: para un usuario con nombre "pedro" y password "miclave1234" se codificará en base 64 el string "pedro:miclave1234" dando como resultado la cadena "cGVkcm86bWljbGF2ZTEyMzQ=". Por tanto en las peticiones se debe incluir una cabecera "Authorization" con valor "Basic cGVkcm86bWljbGF2ZTEyMzQ=".

6.2.3. Definición del API

La definición del API ha sido especificada con **Open API**. En el "Anexo D" se puede encontrar el fichero con la **definición formal detallad**a.

A continuación se exponen los **endpoints a alto nivel**.

GET	/games/{gameId}/simulations	
Descripción	Devuelve un listado de simulaciones para un juego concreto. Las simulaciones corresponden a las jugadas por el bot autenticado.	
Parámetros	authorization (header)	Token que valida la autenticación de la petición.
	gameId (path)	Identificador del juego. Puede tomar los valores: • bullsandcows • labyrinth

Respuesta	[
	{
	"botName": "string",
	"gameId": "string",
	"simulationId": "string",
	"creationDateTime": "2020-04-26T15:07:26.812Z",
	"points": 0,
	"totalNumMatches": 0,
	"finishedMatches": 0
	}	

	]

POST	/games/{gameId}/simulations	
Descripción	Crea una nueva simulación para el juego indicado asociada al bot autenticado.	
Parámetros	authorization (header)	Token que valida la autenticación de la petición.
	gameId (path)	Identificador del juego del que se quiere crear la simulación. Puede tomar los valores: • bullsandcows • labyrinth
Body	{}	

Respuesta	[
	{
	"botName": "string",
	"gameId": "string",
	"simulationId": "string",
	"creationDateTime": "2020-04-26T15:07:26.812Z",
	"points": 0,
	"totalNumMatches": 0,
	"finishedMatches": 0
	}
	]	

GET	/games/{gameId}/simulations/{simulationId}	
Descripción	Devuelve la información de una simulación.	
Parámetros	authorization (header)	Token que valida la autenticación de la petición.
	gameId	Identificador del juego. Puede tomar los valores:

	(path)	• bullsandcows • labyrinth
	simulationId (path)	Identificador de la simulación que se quiere obtener.

Respuesta	{
	"botName": "string",
	"gameId": "string",
	"simulationId": "string",
	"creationDateTime": "2020-04-26T15:19:21.626Z",
	"points": 0,
	"totalNumMatches": 0,
	"finishedMatches": 0
	}	

GET	/games/{gameId}/stats/ranking	
Descripción	Devuelve un listado con las posiciones y puntos de los bots (ranking) para un juego concreto.	
Parámetros	gameId (path)	Identificador del juego. Puede tomar los valores: • bullsandcows • labyrinth

Respuesta	[
	{
	"position": 0,
	"botName": "string",
	"maxPoints": 0
	}
	]	

POST	/bots
Descripción	Creación de nuevo bot.
Body	{

	"botName": "string", "password": "string", "picture": "string" }

Respuesta	{
	"botName": "string",
	"picture": "string"
	}

GET	/bots/{botName}	
Descripción	Devuelve el detalle de un bot.	
Parámetros	authorization (header)	Token que valida la autenticación de la petición.
	botName (path)	Nombre del bot.

Respuesta	{
	"botName": "string",
	"picture": "string"
	}	

PUT	/bots/{botName}	
Descripción	Modifica el detalle de un bot.	
Parámetros	authorization (header)	Token que valida la autenticación de la petición.
	botName (path)	Nombre del bot.
Body	{ "botName": "string", "picture": "string"	

	}

Respuesta	{
	"botName": "string",
	"picture": "string"
	}

DELETE	/bots/{botName}	
Descripción	Elimina un bot.	
Parámetros	authorization (header)	Token que valida la autenticación de la petición.
	botName (path)	Nombre del bot.

PATCH	/bots/{botName}/password	
Descripción	Modifica el la contraseña de un bot.	
Parámetros	authorization (header)	Token que valida la autenticación de la petición.
	botName (path)	Nombre del bot.
Body	Cadena de texto con el valor de la nueva contraseña.	

GET	/games/bullsandcows/simulations/{simulationId}/matches	
Descripción	Listado de partidas del juego bullsandcows para una simulación concreta.	
Parámetros	authorization (header)	Token que valida la autenticación de la petición.

	simulationId (path)	Identificador de la simulación que se quiere obtener.

Respuesta	[
	{
	"botName": "string",
	"gameId": "string",
	"simulationId": "string",
	"matchNum": 0,
	"creationDateTime": "2020-04-26T15:36:39.221Z",
	"points": 0,
	"status": "WIN",
	"rules": {
	"maxTurns": 0,
	"numDigits": 0
	},
	"solution": {
	"secret": "string"
	},
	"turns": [
	{
	"turnNum": 0,
	"creationDateTime": "2020-04-26T15:36:39.221Z",
	"action": {},
	"turnResult": {}
	}
	]
	}
	]	

POST	/games/bullsandcows/simulations/{simulationId}/matches	
Descripción	Creación de una partida del juego bullsandcows para una simulación concreta.	
Parámetros	authorization (header)	Token que valida la autenticación de la petición.

	simulationId (path)	Identificador de la simulación que se quiere obtener.
Body	{}	

Respuesta	{
	"botName": "string",
	"gameId": "string",
	"simulationId": "string",
	"matchNum": 0,
	"creationDateTime": "2020-04-26T15:39:40.171Z",
	"points": 0,
	"status": "WIN",
	"rules": {
	"maxTurns": 0,
	"numDigits": 0
	},
	"solution": {
	"secret": "string"
	},
	"turns": [
	{
	"turnNum": 0,
	"creationDateTime": "2020-04-26T15:39:40.171Z",
	"action": {},
	"turnResult": {}
	}
	]
	}	

GET	/games/bullsandcows/simulations/{simulationId}/matches/{matchNum}	
Descripción	Detalle de una partida del juego bullsandcows para una simulación concreta.	
Parámetros	authorization (header)	Token que valida la autenticación de la petición.

	simulationId (path)	Identificador de la simulación de la que se quiere obtener la partida.
	matchNum (path)	Número de partida que se quiere obtener.

Respuesta	{
	"botName": "string",
	"gameId": "string",
	"simulationId": "string",
	"matchNum": 0,
	"creationDateTime": "2020-04-26T15:39:40.171Z",
	"points": 0,
	"status": "WIN",
	"rules": {
	"maxTurns": 0,
	"numDigits": 0
	},
	"solution": {
	"secret": "string"
	},
	"turns": [
	{
	"turnNum": 0,
	"creationDateTime": "2020-04-26T15:39:40.171Z",
	"action": {},
	"turnResult": {}
	}
	]
	}	

POST	/games/bullsandcows/simulations/{simulationId}/matches/{matchNum}/turns
Descripción	Creación de un nuevo turno de una partida del juego bullsandcows para una simulación concreta. Equivale a un nuevo movimiento del bot.

Parámetros	authorization (header)	Token que valida la autenticación de la petición.
	simulationId (path)	Identificador de la simulación.
	matchNum (path)	Número de partida.
Body	{ "combination": "string" }	
Respuesta	{ "turnNum": 0, "creationDateTime": "2020-04-26T15:44:26.815Z", "action": { "combination": "string" }, "turnResult": { "status": "WIN", "details": "string", "bulls": 0, "cows": 0 } }	

GET	/games/labyrinth/simulations/{simulationId}/matches	
Descripción	Listado de partidas del juego labyrinth para una simulación concreta.	
Parámetros	authorization (header)	Token que valida la autenticación de la petición.
	simulationId	Identificador de la simulación que se quiere

	(path)	obtener.

Respuesta	[
	{
	"botName": "string",
	"gameId": "string",
	"simulationId": "string",
	"matchNum": 0,
	"creationDateTime": "2020-04-26T15:47:45.910Z",
	"points": 0,
	"status": "WIN",
	"rules": {
	"maxTurns": 0,
	"mapWidth": 0,
	"mapHeight": 0,
	"viewRange": 0
	},
	"solution": {
	"map": {
	"width": 0,
	"height": 0,
	"horizontalWalls": [
	{}
	],
	"verticalWalls": [
	{}
	],
	"exitCoordinate": {
	"x": 0,
	"y": 0
	}
	},
	"currentCoordinate": {
	"x": 0,
	"y": 0
	}
	},
	"turns": [
	{
	"turnNum": 0,	

	<pre>"creationDateTime": "2020-04-26T15:47:45.910Z",
	"action": {},
	"turnResult": {}
	}
	]
	}
	]</pre>

POST	/games/labyrinth/simulations/{simulationId}/matches	
Descripción	Creación de una partida del juego labyrinth para una simulación concreta.	
Parámetros	authorization (header)	Token que valida la autenticación de la petición.
	simulationId (path)	Identificador de la simulación que se quiere obtener.
Body	{}	

Respuesta	<pre>{
	"botName": "string",
	"gameId": "string",
	"simulationId": "string",
	"matchNum": 0,
	"creationDateTime": "2020-04-26T15:49:04.940Z",
	"points": 0,
	"status": "WIN",
	"rules": {
	"maxTurns": 0,
	"mapWidth": 0,
	"mapHeight": 0,
	"viewRange": 0
	},
	"solution": {
	"map": {
	"width": 0,
	"height": 0,</pre>	

```
      "horizontalWalls": [
        {}
      ],
      "verticalWalls": [
        {}
      ],
      "exitCoordinate": {
        "x": 0,
        "y": 0
      }
    },
    "currentCoordinate": {
      "x": 0,
      "y": 0
    }
  },
  "turns": [
    {
      "turnNum": 0,
      "creationDateTime": "2020-04-26T15:49:04.940Z",
      "action": {},
      "turnResult": {}
    }
  ]
}
```


GET	/games/labyrinth/simulations/{simulationId}/matches/{matchNum}	
Descripción	Detalle de una partida del juego labyrinth para una simulación concreta.	
Parámetros	authorization (header)	Token que valida la autenticación de la petición.

	simulationId (path)	Identificador de la simulación de la que se quiere obtener la partida.
	matchNum (path)	Número de partida que se quiere obtener.

Respuesta	{
	"botName": "string",
	"gameId": "string",
	"simulationId": "string",
	"matchNum": 0,
	"creationDateTime": "2020-04-26T15:49:04.940Z",
	"points": 0,
	"status": "WIN",
	"rules": {
	"maxTurns": 0,
	"mapWidth": 0,
	"mapHeight": 0,
	"viewRange": 0
	},
	"solution": {
	"map": {
	"width": 0,
	"height": 0,
	"horizontalWalls": [
	{}
	],
	"verticalWalls": [
	{}
	],
	"exitCoordinate": {
	"x": 0,
	"y": 0
	}
	},
	"currentCoordinate": {
	"x": 0,
	"y": 0
	}
	},	

	"turns": [
	{
	"turnNum": 0,
	"creationDateTime": "2020-04-26T15:49:04.940Z",
	"action": {},
	"turnResult": {}
	}
	]
	}

POST	/games/labyrinth/simulations/{simulationId}/matches/{matchNum}/turns	
Descripción	Creación de un nuevo turno de una partida del juego labyrinth para una simulación concreta. Equivale a un nuevo movimiento del bot.	
Parámetros	authorization (header)	Token que valida la autenticación de la petición.
	simulationId (path)	Identificador de la simulación.
	matchNum (path)	Número de partida.

Body	{
	"direction": "TOP"
	}	

Respuesta	{
	"turnNum": 0,
	"creationDateTime": "2020-04-26T15:51:03.279Z",
	"action": {
	"direction": "TOP"
	},
	"turnResult": {
	"status": "WIN",
	"details": "string",
	"surroundingCells": [	

```
    {
      "coordinate": {
        "x": 0,
        "y": 0
      },
      "exit": true,
      "topIsWall": true,
      "rightIsWall": true,
      "bottomIsWall": true,
      "leftIsWall": true
    }
  ]
 }
}
```


6.2.4. Flujo del portal web

A continuación se define el comportamiento general del portal web haciendo uso de **diagramas y mockups**.

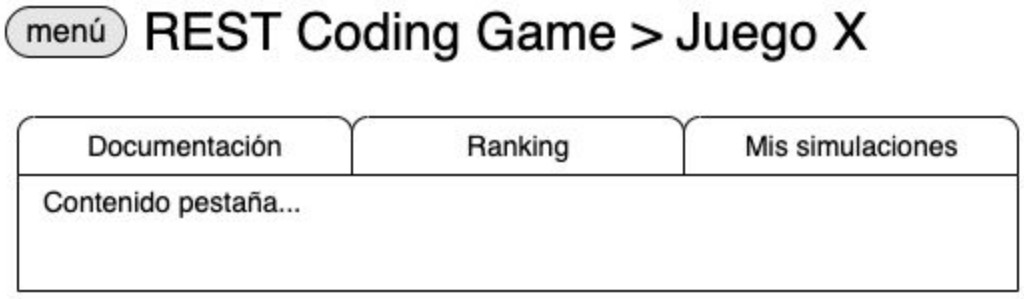
El **flujo de pantallas** e información general será:

Las pantallas con el icono naranja **requieren inicio de sesión** para ser accedidas.

La pantalla de "**inicio"** contiene la información general de la plataforma con documentación genérica independiente del juego como es la autenticación. También contiene accesos directos a los distintos juegos que provee la plataforma ordenados por dificultad.

Todas las pantallas contienen la cabecera y el botón que muestra un menú desplegable con enlaces a las principales secciones.

Cada "**juego"** tiene una pantalla con 3 pestañas:



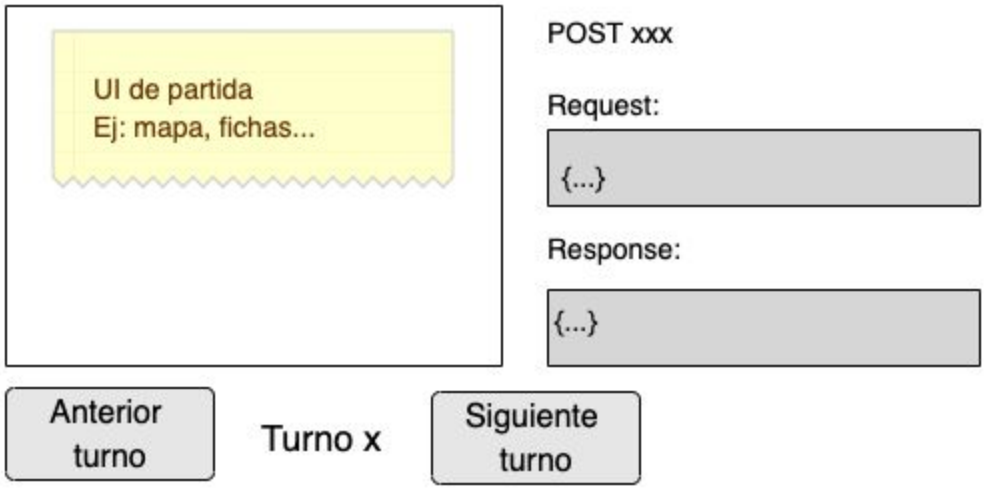
La "**documentación"** explica cómo se juega al juego en cuestión y mediante un ejemplo mostrará cómo deben ejecutarse los distintos pasos para llevar a cabo las partidas de una simulación mediante la API REST que se proporciona.

El "**ranking"** muestra un listado con todos los bots que han participado en este juego junto a su mejor puntuación entre todas las simulaciones.

En "**mis simulaciones**" podrás ver un listado con todas las simulaciones ejecutadas por el usuario actual. Se mostrará la fecha de inicio de la simulación, los puntos obtenidos y el número de partidos terminados y total que tiene la simulación. Pulsando sobre una de las simulaciones del listado podemos ir a la pantalla de detalle de una simulación.

Para cada "**simulación**" se nos muestra un listado de partidas con los detalles más relevantes de cada una: número, reglas, número de turnos jugados, puntos obtenidos. Estas partidas se podrán extender en modo acordeón para ver los detalles de una partida concreta.

Para cada "**partida**" se mostrará un reproductor para ver turno a turno los detalles de la partida. En la parte izquierda se mostrará la visualización gráfica de la partida y a la derecha las llamadas REST ejecutadas. Mediante unos botones se podrá ir pasando entre los distintos turnos.

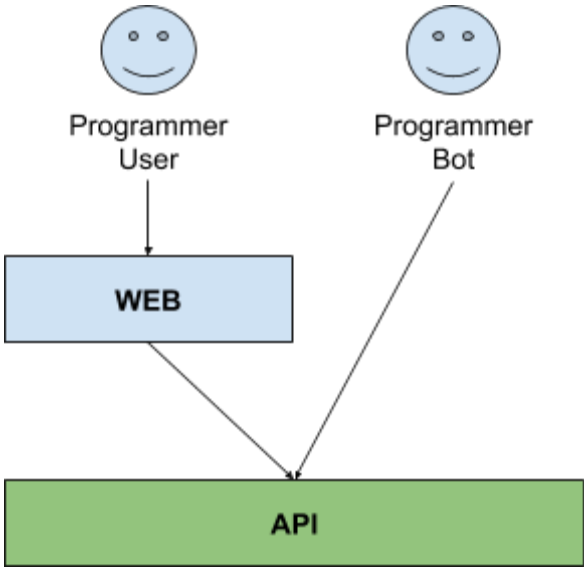


6.3. Diseño

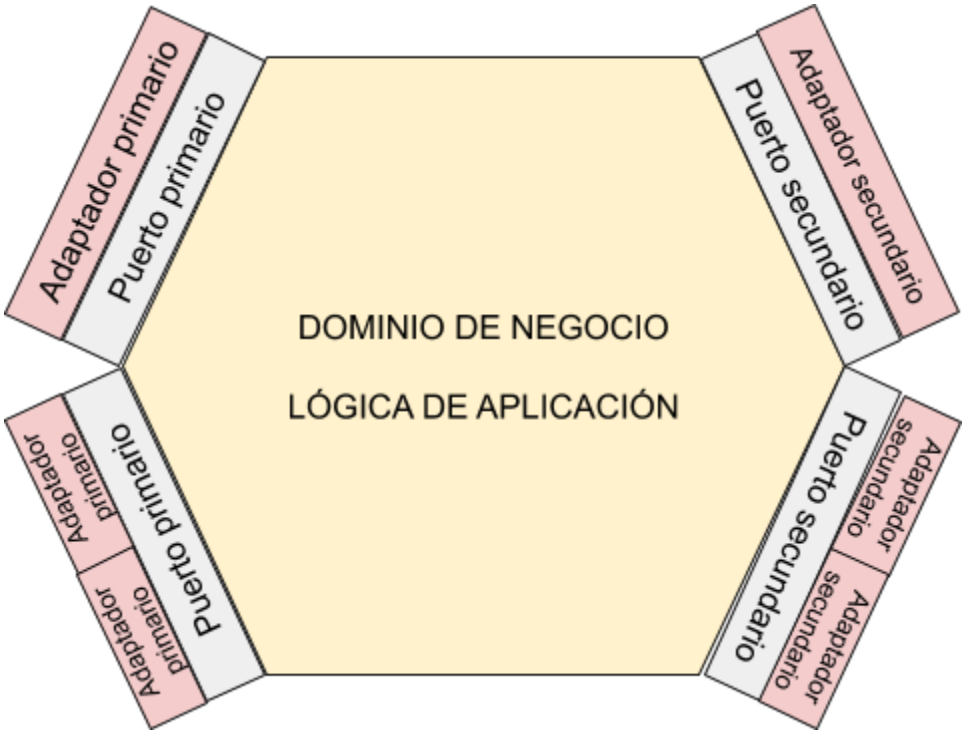
6.3.1. Diagrama de componentes y capas

A nivel global la plataforma se divide en dos sistemas totalmente independientes:

- **API**: Contiene toda la parte servidora. Esta es consumida directamente por los bots programados por los clientes y también por el portal web. Su código está en el proyecto "rest-coding-game-api" (Anexo A).
- **Web**: Contiene el portal web. Esta es utilizada por los programadores para visualizar a través de una web los resultados de sus simulaciones y la documentación. Extrae los datos a mostrar sobre las simulaciones ejecutadas haciendo uso de la API. Su código está en el proyecto "rest-coding-game-web" (Anexo B).

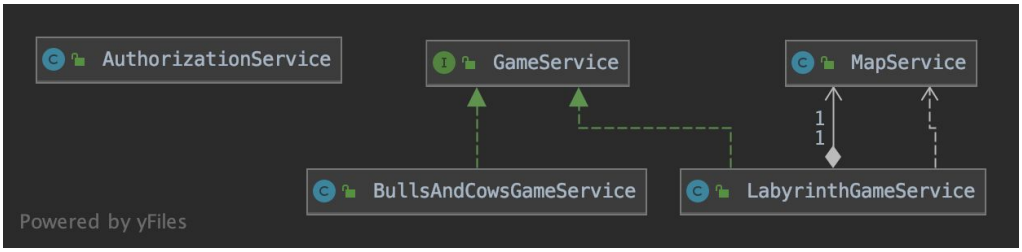


El proyecto backend sigue una arquitectura hexagonal teniendo las siguientes capas:



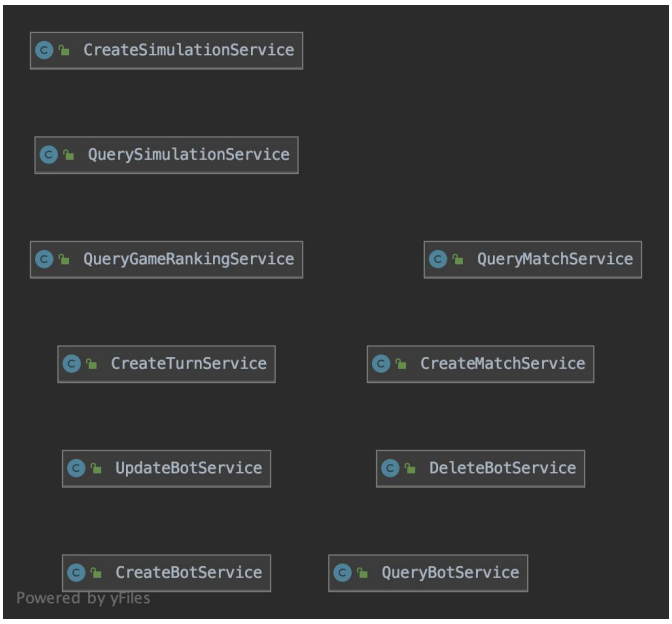
El "**core"** de la aplicación está formado por:

- **Dominio de negocio**: contenido en el paquete "domain". Contiene las clases de dominio. Son objetos que definen la estructura del dominio sin contenido funcional. El contenido de este paquete se puede ver en el diagrama de clases del dominio.
- **Lógica de la aplicación**: contenido en el paquete "usecases.services". Contiene los servicios principales de la aplicación. Hacen uso de los puertos de salida.

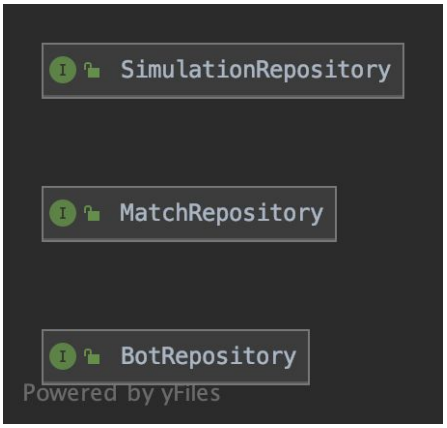


Las definiciones de interacciones con el core se dividen en dos tipos:

- **Primary/input ports**: contenido en el paquete "usecases.input". Son los procesos a ejecutar bajo una petición de entrada a la aplicación. Definen el comportamiento para que sean usados por los adaptadores primarios.

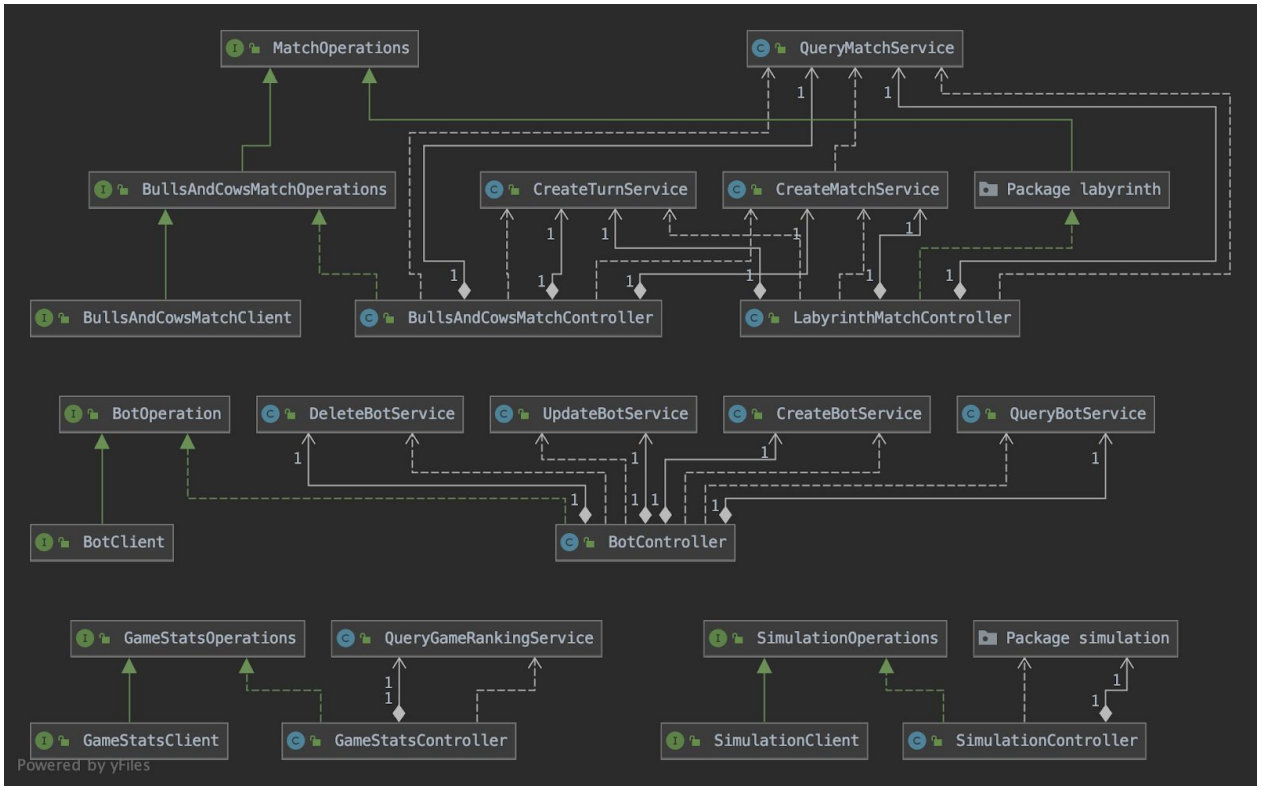


- **Secondary/output ports**: contenido en el paquete "usecases.output". Son iteraciones de salida a la aplicación. Definen el interfaz para interactuar con un tercero para que sean implementados por los adaptadores secundarios.

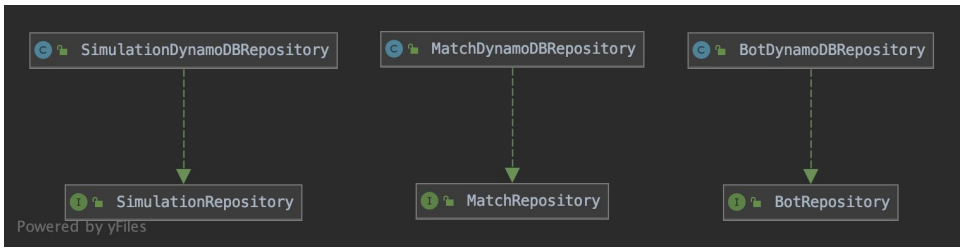


Las implementaciones de la iteraciones son los adaptadores de un puerto concreto. Existen por tanto también dos tipos:

- **Primary/input adapters**: contenidos en el paquete "adapters.input" definen el método de entrada a la aplicación. Por ejemplo a través de una llamada REST definiendo un "controller".



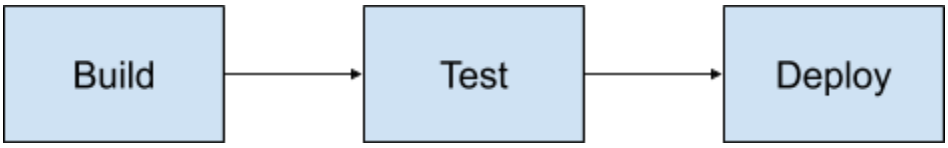
- **Secondary/output adapters**: contenidos en el paquete "adapters.output" definen el método de salida a la aplicación. Por ejemplo el acceso a una base de datos mongoDB para la obtención de las simulaciones.



6.3.2. Integración y despliegue continuo

Se ha configurado **dos pipelines de integración y despliegue continuo** con Heroku: uno para frontend y otro para backend. Estos se ejecutan automáticamente cuando el código es subido al repositorio de código fuente de Heroku.

Los pipelines tienen los pasos básicos más habituales:



- **Build**: Primeramente se construye la aplicación. Si hay cualquier problema de compilación esta fase fallará y se interrumpirá el pipeline.
- **Test**: Se ejecutan todos los test automatizados de la aplicación que están contenidos en el propio repositorio de código.
- **Deploy**: Se despliega la aplicación sobre el entorno productivo y está accesible para todo el público.

Este pipeline se podría mejorar para desplegar antes en un entorno pre-productivo en el que hacer pruebas manuales antes de hacerlo accesible al público general en el entorno productivo.

Para la ejecución de la construcción y los tests en frontend que utiliza npm y para backend (API) se utiliza gradle. Como parte de la configuración de construcción se deben definir todas las dependencias a utilizar. En el caso del frontend se tratan de dependencias de npm como pueden ser "angular" o "express". En el caso del backend estas dependencias son descargadas de repositorios maven como pueden ser "micronaut" y "mapstruct".

- Frontend: configurado en el fichero package.json

```
{
"name": "rest-coding-game-web",
```

```
"version": "1.0.0",
"main": "server.js",
"scripts": {
  "ng": "ng",
  "start": "ng serve",
  "build": "ng build",
  "test": "ng test",
  "lint": "ng lint",
  "e2e": "ng e2e",
  "heroku-postbuild": "ng build --configuration=heroku --aot"
},
"private": true,
"dependencies": {
  "@angular/animations": "^8.2.6",
  "@angular/cdk": "^8.2.0",
  "@angular/common": "~8.2.5",
  "@angular/compiler": "~8.2.5",
  "@angular/core": "~8.2.5",
  "@angular/flex-layout": "^8.0.0-beta.27",
  "@angular/forms": "~8.2.5",
  "@angular/material": "^8.2.0",
  "@angular/platform-browser": "~8.2.5",
  "@angular/platform-browser-dynamic": "~8.2.5",
  "@angular/router": "~8.2.5",
  "component": "^1.1.0",
  "express": "^4.17.1",
  "lodash": "^4.17.15",
  "ng2-ace-editor": "^0.3.9",
  "rxjs": "~6.4.0",
  "rxjs-compat": "^6.5.3",
  "swagger-ui-dist": "^3.24.3",
  "tslib": "^1.10.0",
  "zone.js": "~0.9.1"
},
"devDependencies": {
  "@angular-devkit/build-angular": "^0.803.20",
  "@angular/cli": "~8.3.4",
  "@angular/compiler-cli": "~8.2.5",
  "@angular/language-service": "~8.2.5",
  "@types/jasmine": "~3.3.8",
  "@types/jasminewd2": "~2.0.3",
  "@types/node": "^8.10.54",
  "codelyzer": "^5.0.0",
  "jasmine-core": "~3.4.0",
  "jasmine-spec-reporter": "~4.2.1",
```

```
    "karma": "~4.1.0",
    "karma-chrome-launcher": "~2.2.0",
    "karma-coverage-istanbul-reporter": "~2.0.1",
    "karma-jasmine": "~2.0.1",
    "karma-jasmine-html-reporter": "^1.4.0",
    "protractor": "~5.4.0",
    "ts-node": "~7.0.0",
    "tslint": "~5.15.0",
    "typescript": "~3.5.3"
  },
  "engines": {
    "node": "10.11.0",
    "npm": "6.11.3"
  }
}
```

- Backend: configurado en el fichero build.gradle

```
plugins {
   id 'net.ltgt.apt' version '0.8'
   id "net.ltgt.apt-eclipse" version "0.21"
   id "net.ltgt.apt-idea" version "0.21"
   id "com.github.johnrengelman.shadow" version "5.0.0"
   id "application"
}

version "0.1"
group "fm.rcg.api"

repositories {
   mavenCentral()
   maven { url "https://jcenter.bintray.com" }
}

configurations {
   // for dependencies that are needed for development only
   developmentOnly
}

dependencies {
   annotationProcessor 'org.projectlombok:lombok:1.18.10'
   annotationProcessor 'org.mapstruct:mapstruct-processor:1.3.0.Final'
```

```
    annotationProcessor platform("io.micronaut:micronaut-bom:$micronautVersion")
    annotationProcessor "io.micronaut:micronaut-inject-java"
    annotationProcessor "io.micronaut:micronaut-validation"
    annotationProcessor "io.micronaut.configuration:micronaut-openapi"

    implementation platform("io.micronaut:micronaut-bom:$micronautVersion")
    implementation "io.micronaut:micronaut-inject"
    implementation "io.micronaut:micronaut-validation"
    implementation "io.micronaut:micronaut-runtime"
    implementation "javax.annotation:javax.annotation-api"
    implementation "io.micronaut:micronaut-http-server-netty"
    implementation "io.micronaut:micronaut-http-client"
    compile 'io.micronaut:micronaut-management'

    compileOnly 'org.projectlombok:lombok:1.18.10'
    compile 'org.mapstruct:mapstruct:1.3.1.Final'
    compile 'org.mapstruct:mapstruct-jdk8:1.2.0.Final'
    compile 'org.mapstruct:mapstruct-processor:1.2.0.Final'
    compile 'io.micronaut.configuration:micronaut-mongo-reactive'
    compile "io.swagger.core.v3:swagger-annotations"
    compile group: 'org.apache.commons', name: 'commons-lang3', version: '3.9'
    compile group: 'com.google.code.gson', name: 'gson', version: '2.8.6'

    runtimeOnly "ch.qos.logback:logback-classic:1.2.3"
    testAnnotationProcessor platform("io.micronaut:micronaut-bom:$micronautVersion")
    testAnnotationProcessor "io.micronaut:micronaut-inject-java"
    testImplementation platform("io.micronaut:micronaut-bom:$micronautVersion")
    testImplementation "org.junit.jupiter:junit-jupiter-api"
    testImplementation "io.micronaut.test:micronaut-test-junit5"
    testCompile "org.mockito:mockito-junit-jupiter:2.22.0"
    testCompile 'de.flapdoodle.embed:de.flapdoodle.embed.mongo:2.2.0'
    testRuntimeOnly "org.junit.jupiter:junit-jupiter-engine"
}

test.classpath += configurations.developmentOnly

mainClassName = "fm.rcg.api.configuration.Application"

// use JUnit 5 platform
test {
    environment "MICRONAUT_ENV_DEDUCTION", "false"
    useJUnitPlatform()
}
tasks.withType(JavaCompile){
    options.encoding = "UTF-8"
```

```
    options.compilerArgs.add('-parameters')
    options.fork = true
    options.forkOptions.jvmArgs <<
'-Dmicronaut.openapi.views.spec=rapidoc.enabled=true,swagger-ui.enabled=true,swagger-ui.theme=flattop'
}

task stage(dependsOn: ['build', 'clean'])
build.mustRunAfter clean

shadowJar {
    mergeServiceFiles()
}

run.classpath += configurations.developmentOnly
run.jvmArgs('-noverify', '-XX:TieredStopAtLevel=1', '-Dcom.sun.management.jmxremote')
```

Para la **configuración del despliegue** se ha creado un fichero "Procfile" en cada proyecto. Este fichero define el comando a ejecutar para el despliegue que utiliza **"node" para el frontend y "java" para el backend**:

- Frontend:

```
web: node server.js
```

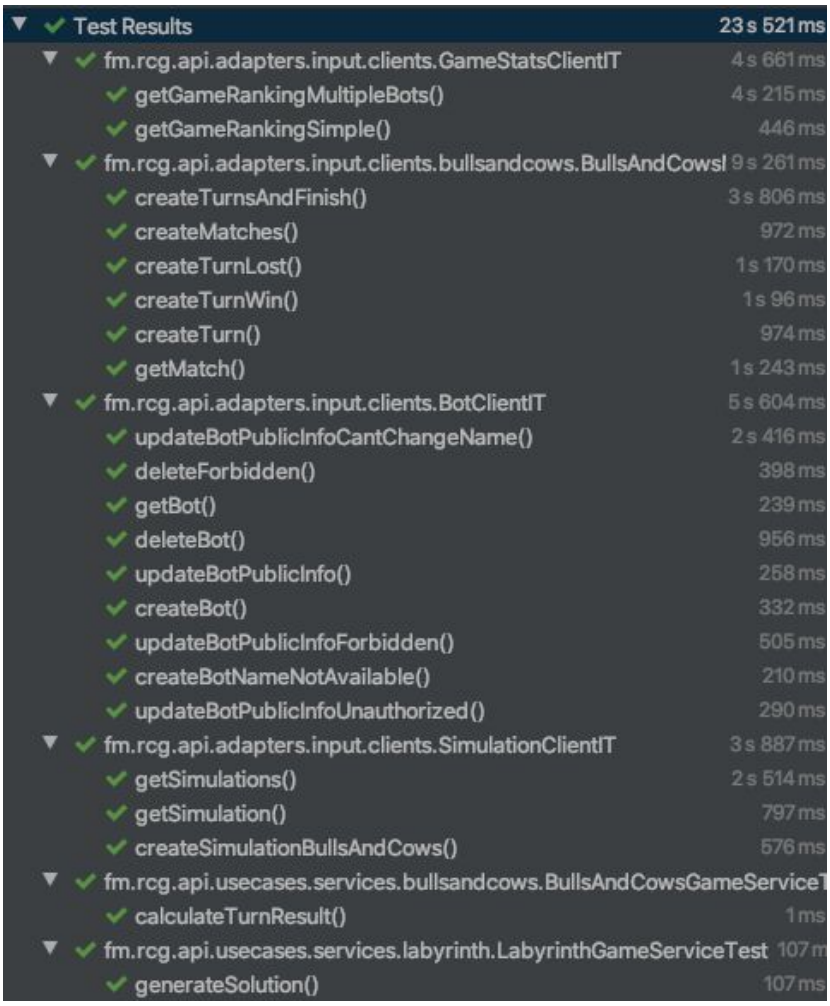
- Backend:

```
web: java -jar build/libs/rest-coding-game-api-0.1-all.jar
```

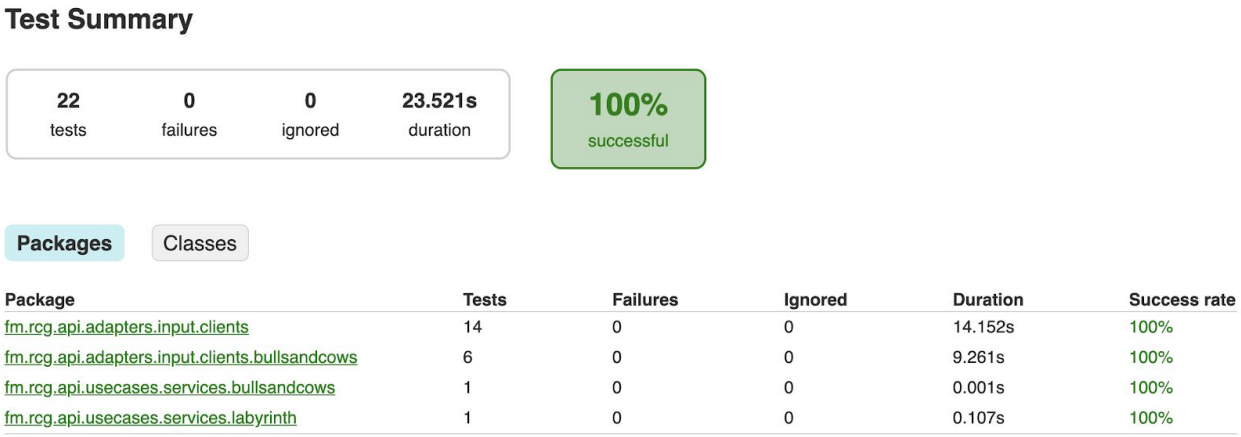
6.3.3. Tests y cobertura

El proyecto *backend* tiene los tests en el directorio "src/test/java".

Se han implementado **tests unitarios y de integración** sumando en total 22 tests automatizados.



Se puede visualizar un informe web generado por gradle sobre la ejecución de los tests:



Test Summary

22 tests	0 failures	0 ignored	23.521s duration

100% successful

Packages | Classes

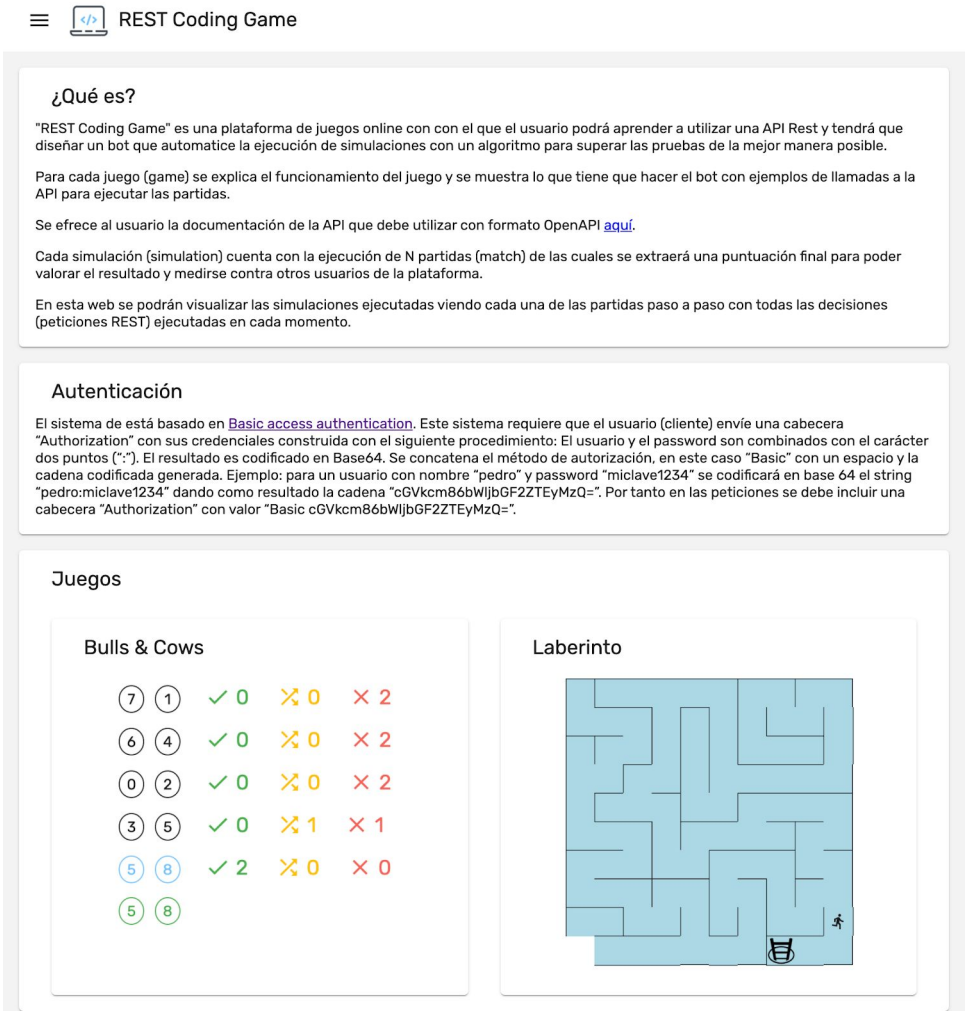
Package	Tests	Failures	Ignored	Duration	Success rate
fm.rcg.api.adapters.input.clients	14	0	0	14.152s	100%
fm.rcg.api.adapters.input.clients.bullsandcows	6	0	0	9.261s	100%
fm.rcg.api.usecases.services.bullsandcows	1	0	0	0.001s	100%
fm.rcg.api.usecases.services.labyrinth	1	0	0	0.107s	100%

6.3.4. Interfaz gráfica del portal web

Se ha implementado la siguiente interfaz gráfica en el portal web de la plataforma.

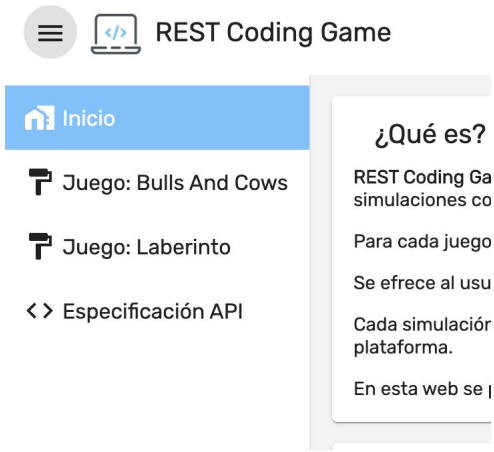
Pantalla de inicio:

Muestra toda la información general de la plataforma, la documentación genérica y los enlaces a los distintos juegos disponibles.



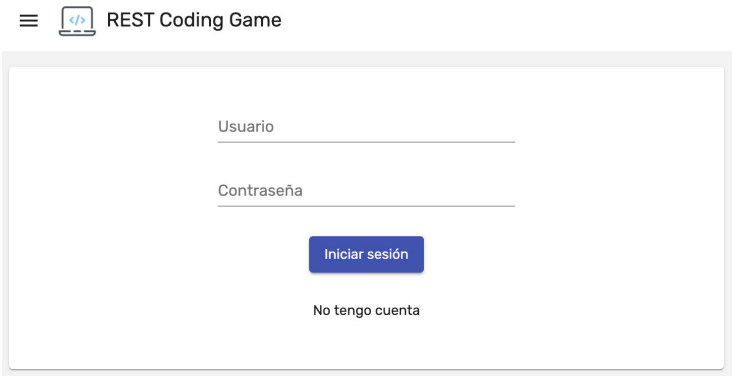
Menú lateral:

Se despliega desde la parte lateral izquierda de la pantalla automáticamente al pulsar el botón de menú de la cabecera. Se oculta al pulsar fuera de este o seleccionar alguna de las secciones que ofrece.



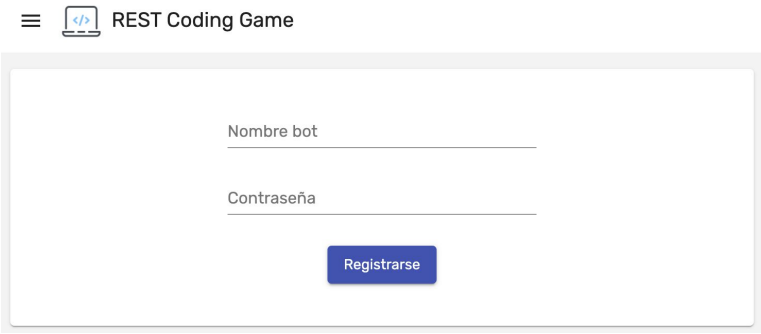
Inicio de sesión:

Algunas pantallas requieren iniciar sesión. Si se pulsa sobre la sección de login o se intenta entrar en una de las secciones que requieren inicio de sesión se muestra esta pantalla.



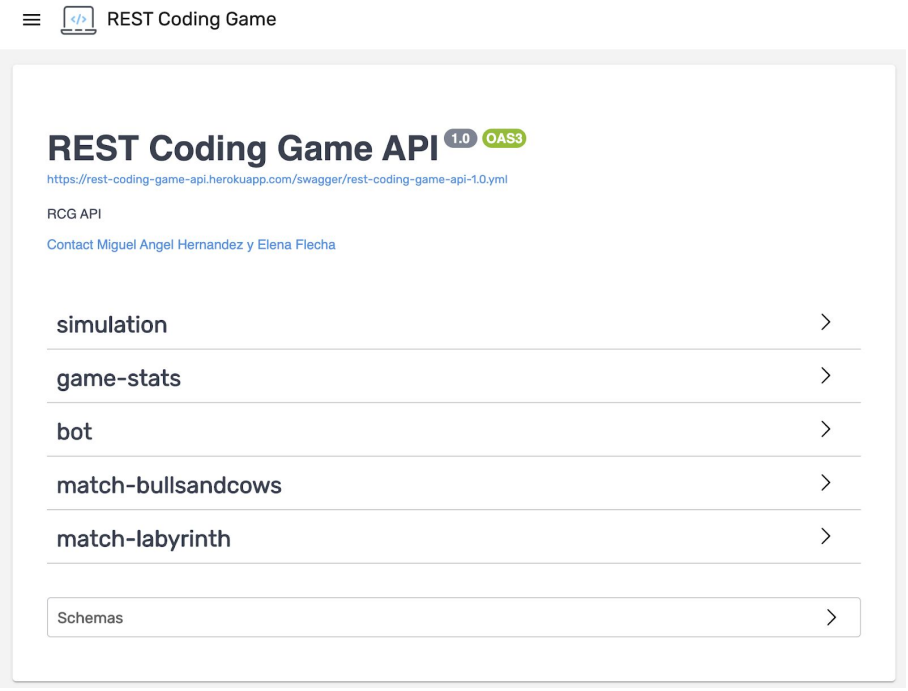
Registro:

Si el usuario no tiene cuenta se le permite crear una nueva cuenta de bot de manera rápida con un sencillo formulario de registro.



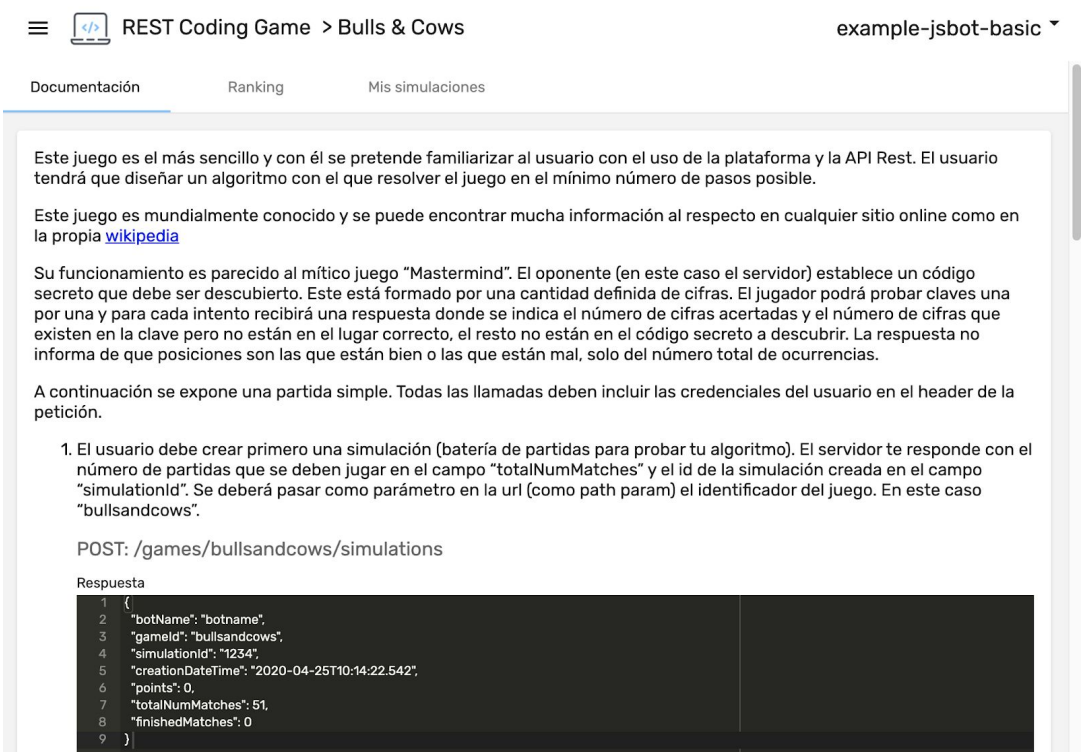
Open API spec:

Integra la documentación del API definida con Swagger en el portal web.



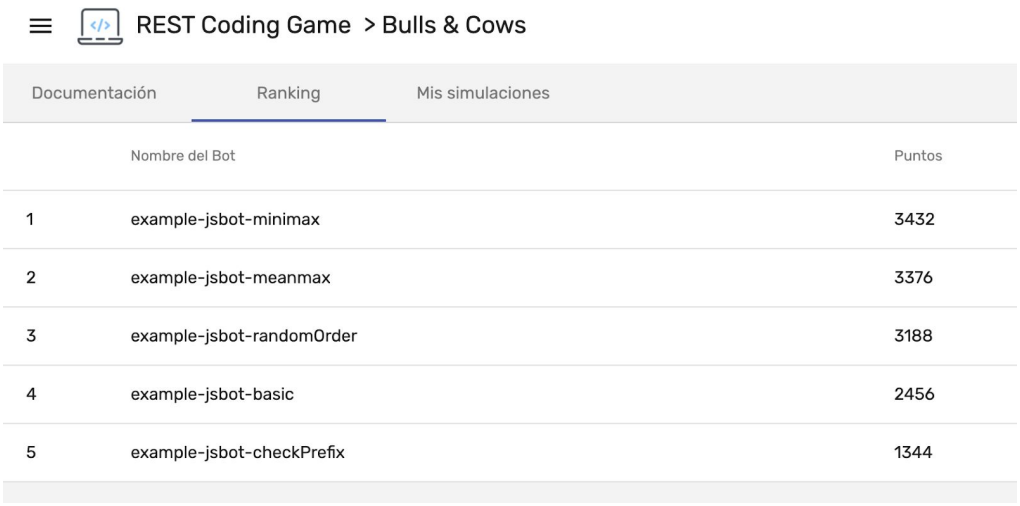
Documentación del juego:

Muestra la documentación del juego con un ejemplo de partida y sus correspondientes llamadas y respuestas al API. Esta documentación depende del juego en cuestión.



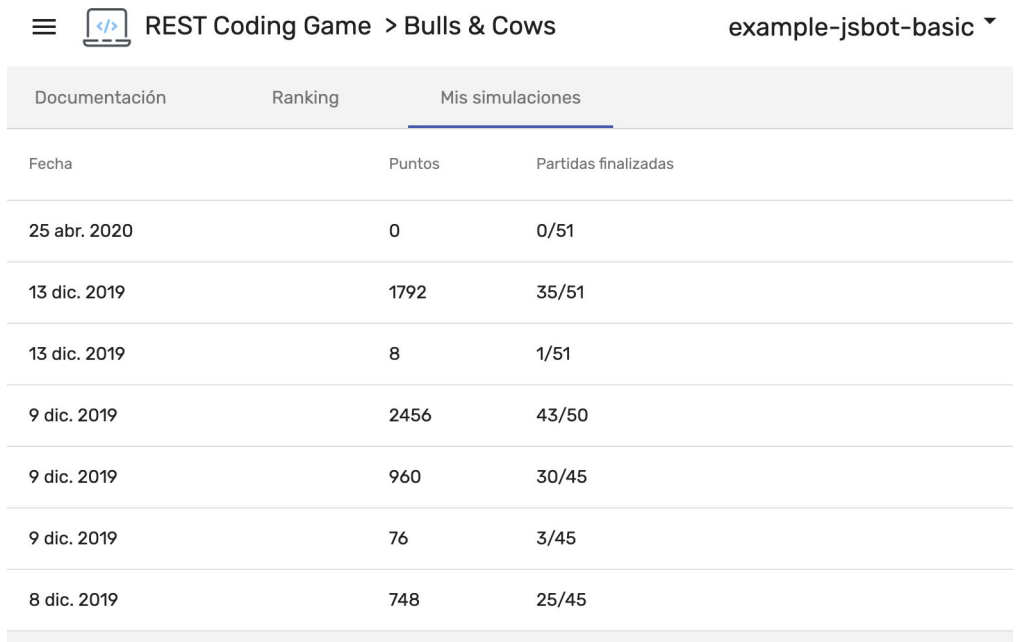
Ranking:

Muestra el ranking de bots que han lanzado simulaciones contra el juego en cuestión con la mejor puntuación obtenida por cada uno de ellos. Está ordenado de mayor a menor puntuación récord.



Mis simulaciones:

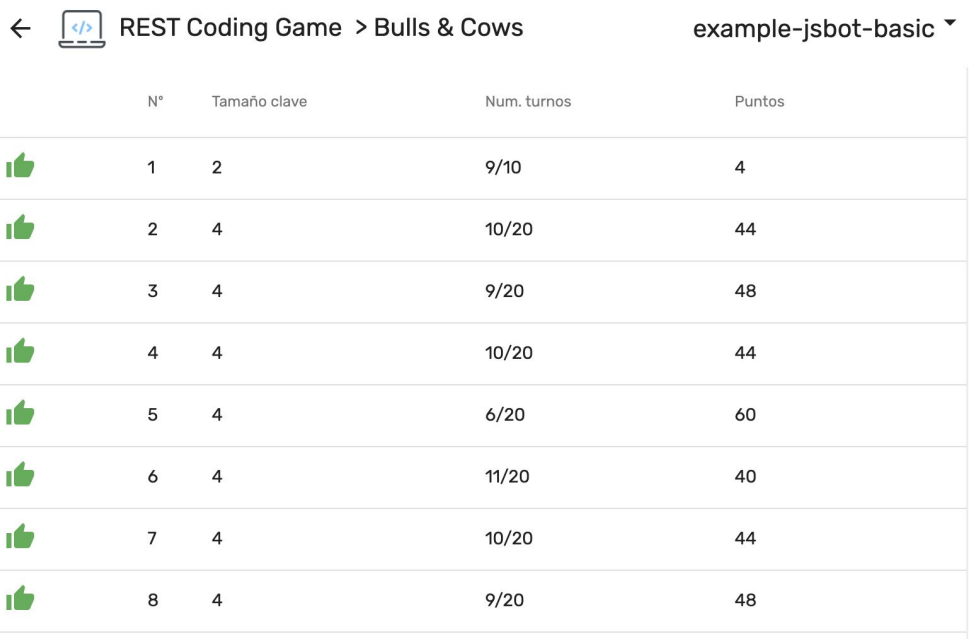
Muestra el listado de todas las simulaciones ejecutadas por el usuario actual. Para cada partida simulación se muestra la fecha, los puntos y el progreso en número de partidas finalizadas.



Simulación:

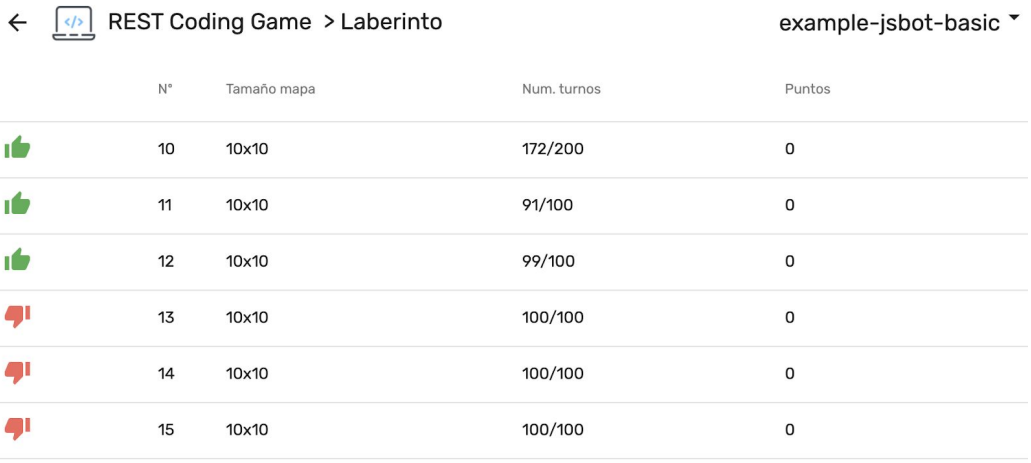
Se muestra el listado de todas las partidas que componen una simulación. Se puede ver fácilmente las partidas que han finalizado con victoria y las que han finalizado con derrota y las reglas de cada partida.

Ejemplo para el juego "Bulls & Cows":



	N°	Tamaño clave	Num. turnos	Puntos
	1	2	9/10	4
	2	4	10/20	44
	3	4	9/20	48
	4	4	10/20	44
	5	4	6/20	60
	6	4	11/20	40
	7	4	10/20	44
	8	4	9/20	48

Ejemplo para el juego "Laberinto":



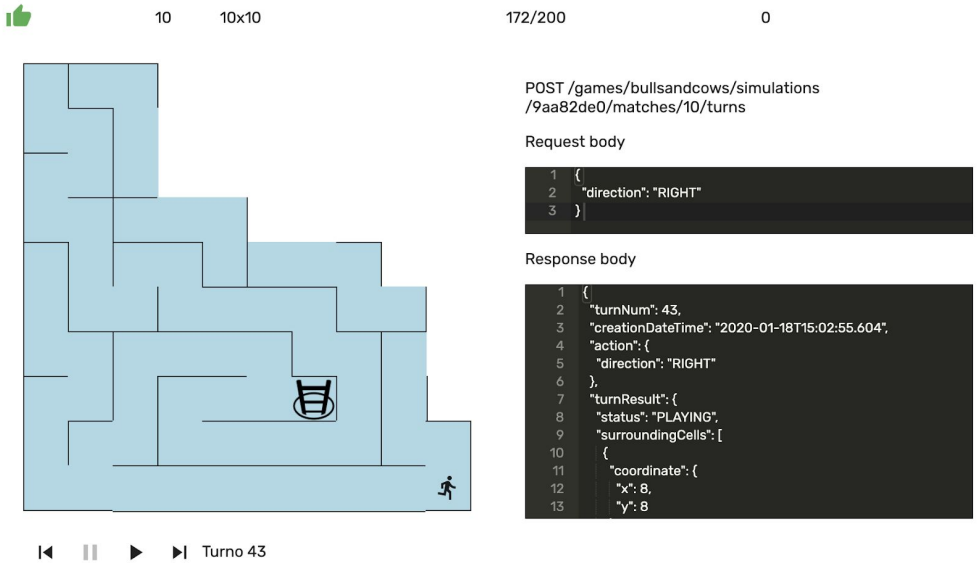
Partida:

Pulsando sobre una de las partidas se muestra el detalle de esta permitiendo visualizar turno por turno todos los detalles de esa partida. Se puede visualizar en la parte izquierda la interfaz gráfica del juego si se jugara de manera no automatizada con un bot y en la parte derecha la peticiones y respuestas ejecutadas para cada uno de los turnos.

Ejemplo para "Bulls and Cows":

Ejemplo para “Laberinto”:

Este permite además reproducir la partida en modo rápido para ver el camino seguido de manera sencilla.



6.3.5. Selección de personal

La solución podría ser utilizada para la **selección de personal de perfiles desarrolladores de software**. De esta manera se podrá retar al candidato a realizar un bot para demostrar sus capacidades de entendimiento de documentación, implementación de algoritmos e implementación de cliente de un API Rest.

El proceso podría ser **desatendido** hasta la entrega enviando el enunciado con los objetivos de juegos a resolver (en función del puesto) y las credenciales de la cuenta a utilizar. Se le solicitará la entrega del código fuente utilizado para poder evaluar la calidad del código escrito además de los resultados en puntuación de las simulaciones ejecutadas por sus bots.

Entre las ventajas de este método de evaluación técnica se encuentran:

- Es un proceso de selección **entretenido**.
- Aporta **mucha información** sobre las capacidades de programación, **entendimiento** y explicación del candidato.
- La implementación es muy variable en cuanto a alcance y este dependerá de las **capacidades y ganas** del candidato.

6.3.6. Bots de ejemplo

Dentro del proyecto "rest-coding-game-example-jsbot" (Anexo C) se han implementado **varios bots** de ejemplo que resuelven el problema planteado para los dos juegos que ofrece actualmente la plataforma.

No vamos a entrar en detalle de los algoritmos empleados dado que no es el objetivo del proyecto la implementación de una solución a de los juegos. Solo mencionar que para comprobar que la plataforma puede ser utilizada y que la complejidad de los

problemas es asequible, se han implementado varias alternativas con algoritmos de distintas características. Desde uno con simple **fuerza bruta** que suele perder al superar el número de turnos máximo permitidos, hasta otros con lógica más compleja como **algoritmos minimax**.

7. CONCLUSIONES

Como resultado de este trabajo se ha conseguido obtener un **producto** con el que se pretende que otras personas, por ejemplo estudiantes, puedan **aprender** acerca de lo que es y cómo se utiliza una **API** de una forma más dinámica y entretenida.

El cumplimiento de los objetivos del proyecto ha sido un éxito. Tras un proceso ordenado de planteamiento teórico, definición, análisis, diseño e implementación, se ha obtenido una **plataforma funcional y totalmente operativa disponible** al público a través de Internet.

La ejecución del proyecto nos ha servido **personalmente** para **aprender** y conocer en profundidad **nuevas tecnologías** que no conocíamos como Micronaut y Gradle, y **mejorar** en otras como el paradigma de programación reactiva.

El enfoque tecnológico utilizado ha sido todo un acierto. La **curva de aprendizaje** del framework Micronaut ha sido realmente **sencilla** por su parecido con Spring, el cual conocíamos. La arquitectura de capas utilizada ha permitido la implementación de un **código ordenado y totalmente desacoplado**. La solución cuenta con una bóveda de seguridad basada en **tests funcionales unitarios y de integración** que ofrecen una alta confianza y facilidad en futuros mantenimiento y evolutivos.

8. BIBLIOGRAFÍA

Mozilla. (s.f.). JavaScript | MDN. Recuperado el 20 de diciembre de 2019 de https://developer.mozilla.org/en-US/docs/Web/JavaScript

ECMAScript. (s.f.). En Wikipedia. Recuperado el 20 de diciembre de 2019 de https://es.wikipedia.org/wiki/ECMAScript

ECMA International (Junio 2019). Standard ECMA-262. Recuperado el 20 de diciembre de 2019 de http://www.ecma-international.org/publications/standards/Ecma-262.htm

Microsoft. (s.f.). TypeScript - JavaScript that scales. Recuperado el 21 de diciembre de 2019 de https://www.typescriptlang.org/

Facebook. (s.f.). React - A JavaScript library for building user interfaces. Recuperado el 22 de diciembre de 2019.

Evan you. (s.f.). Vue.js. Recuperado el 23 de diciembre de 2019 de https://vuejs.org/

Google. (s.f.). Angular. Recuperado el 23 de diciembre de 2019 de https://angular.io/

Google. (s.f.) Design- Material Design. Recuperado el 23 de diciembre de 2019 de https://material.io/design

Google. (s.f.). Angular Material UI component library. Recuperado el 26 de diciembre de 2019 de https://material.angular.io/

Java (lenguaje de programación). (s.f.). En Wikipedia. Recuperado el 27 de diciembre de 2019 de https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)

VMware. (s.f.). Spring. Recuperado el 28 de diciembre de 2019 de https://spring.io/

Object Computing. (s.f.). Micronaut Framework. Recuperado el 29 de diciembre de 2019 de https://micronaut.io/

Amazon Web Services (s.f.). AWS | Lambda - Gestión de recursos informáticos. Recuperado el 10 de enero de 2020 de https://aws.amazon.com/es/lambda/

Robert C. Martin. (13 de agosto de 2012). En Clean Coder Blog. The Clean Architecture. Recuperado el 13 de enero de 2020 de https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html.

Hexagonal architecture (software). (s.f.). En Wikipedia. Recuperado el 13 de enero de 2020 de https://en.wikipedia.org/wiki/Hexagonal_architecture_(software)

Grzegorz Ziemoński. (13 de febrero de 2017). DZone. Layered Architecture Is Good. Recuperado el 13 de enero de 2020 de https://dzone.com/articles/layered-architecture-is-good

ReactiveX. (s.f.). GitHub. RxJava. Recuperado el 15 de enero de 2020 de https://github.com/ReactiveX/RxJava

Jonathan Ordóñez. (s.f.). En Idento. ¿Qué es una API REST? Recuperado el 16 de enero de 2020 de https://www.idento.es/blog/desarrollo-web/que-es-una-api-rest/

Palantir Technologies. (s.f.). TSLint. Recuperado el 20 de enero de 2020 de https://palantir.github.io/tslint/

SonarSource (s.f.). SonarQube. Recuperado el 24 de enero de 2020 de https://www.sonarsource.com/products/sonarqube/

Basic access authentication. (s.f.). En Wikipedia. Recuperado el 3 de febrero de 2020 de https://en.wikipedia.org/wiki/Basic_access_authentication

Base 64. (s.f.). En Wikipedia. Recuperado el 3 de febrero de 2020 de https://en.wikipedia.org/wiki/Base64

Desarrollo guiado por pruebas. (s.f.). En Wikipedia. Recuperado el 10 de febrero de 2020 de https://es.wikipedia.org/wiki/Desarrollo_guiado_por_pruebas

Computación en la nube. (s.f.). En Wikipedia. Recuperado el 17 de febrero de 2020 de https://es.wikipedia.org/wiki/Computaci%C3%B3n_en_la_nube

Integración continua. (s.f.). En Wikipedia. Recuperado el 20 de febrero de 2020 de https://es.wikipedia.org/wiki/Integraci%C3%B3n_continua

Salesforce. (s.f.). Heroku CI | Heroku Dev Center. Recuperado el 21 de febrero de 2020 de https://devcenter.heroku.com/articles/heroku-ci

Emiliano Zublena. (26 de junio de 2017). En Medium. API-First development. Recuperado el 23 de febrero de 2020 de https://medium.com/@emilianozublena/api-first-development-c202a61cf3b2

Janet Wagner. (s.f.). Understanding the API-First Approach to Building Products. Recuperado el 25 de febrero de 2020 de https://swagger.io/resources/articles/adopting-an-api-first-approach/

Base de datos relacional. (s.f.). En Wikipedia. Recuperado el 2 de marzo de 2020 de https://es.wikipedia.org/wiki/Base_de_datos_relacional

MongoDB. (26 de junio de 2017). The most popular database for modern apps | MongoDB. Recuperado el 3 de marzo de 2020 de https://www.mongodb.com/

Minimax. (s.f.). En Wikipedia. Recuperado el 10 de abril de 2020 de https://es.wikipedia.org/wiki/Minimax

9. APÉNDICE

9.1. ANEXO A: Código fuente API

Contiene el código fuente del proyecto backend "rest-coding-game-api" que implementa la API. Utiliza principalmente las tecnologías *Java, Gradle y Micronout*.

9.2. ANEXO B: Código fuente web

Contiene el código fuente del proyecto frontend "rest-coding-game-web", que implementa la el portal web. Utiliza principalmente las tecnologías *HTML/CSS, Typescript y Angular*.

9.3. ANEXO C: Código fuente bots de ejemplo

Contiene el código fuente del proyecto "rest-coding-game-example-jsbot" con varios bots de ejemplo implementados con javascript. Incluye varias implementaciones para la resolución de los juegos "Bulls And Cows" y "Laberinto".

9.4. ANEXO D: Especificación del API

Contiene la definición formal detallada de API en formato Open API.