# ESCUELA POLITÉCNICA NACIONAL

## FACULTAD DE INGENIERÍA DE SISTEMAS

# DESARROLLO DE UNA APLICACIÓN MÓVIL INTERACTIVA PARA EL REFUERZO ACADÉMICO EN LA ASIGNATURA DE INGLÉS PARA NIÑOS DE 6 A 8 AÑOS APLICANDO LA INGENIERÍA DE LA USABILIDAD

## API RESTFUL PARA LA APLICACIÓN KIDDO ENGLISH CLUB

**TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO DE SOFTWARE**

**KENNETH LEONARDO ULLOA TOBAR**

kenneth.ulloa@epn.edu.ec

**DIRECTOR: PhD. TANIA ELIZABETH CALLE JIMENEZ**

tania.calle@epn.edu.ec

**DMQ, julio 2024**

# CERTIFICACIONES

Yo, Kenneth Leonardo Ulloa Tobar declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

_____

**Kenneth Leonardo Ulloa Tobar**

Certifico que el presente trabajo de integración curricular fue desarrollado por Kenneth Leonardo Ulloa Tobar, bajo mi supervisión.

_____

**PhD. Tania Elizabeth Calle Jiménez**

**DIRECTOR**

# DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

KENNETH LEONARDO ULLOA TOBAR

TANIA ELIZABEHT CALLE JIMÉNEZ

# DEDICATORIA

Dedico este trabajo a mi madre, Anita, quién formó mi carácter para nunca claudicar, siempre apuntar alto y jamás rendirme con dedicación y amor incansables desde mi primer día de vida. Me inspira la manera en que realiza su trabajo con excelencia y siempre busca mejorar.

De la misma forma, dedico este trabajo a mi padre, Pablo, quien siempre estuvo ahí para mí, la nobleza de su corazón me inspira a ser mejor cada día, así mismo trabajó increíblemente duro para que no me faltara nada y me permitió enfocarme en mis estudios. Es mi ejemplo y referente de lo que es una persona de bien.

# AGRADECIMIENTO

Agradezco a Dios por darme todos los talentos y cualidades que me han permitido llegar hasta este punto en mi vida, por otorgarme una familia increíblemente amorosa y protectora y por nunca dejarme a pesar de mis falencias y defectos.

Agradezco a mis padres por cuidarme y apoyarme a lo largo del trayecto de este proyecto.

Agradezco a aquellos profesores que aman lo que hacen, transmiten con alegría aquello que saben e inspiran a sus estudiantes a ser mejores. Extiendo el agradecimiento especialmente a mi tutora Tania Calle por confiar en mí en todo el proceso y a Carlos Iñiguez por su guía en este trabajo y la pasión que pone en las materias que imparte.

# ÍNDICE DE CONTENIDO

# ÍNDICE DE FIGURAS

# RESUMEN

El presente documento presenta los resultados de la investigación, definición de requerimientos, implementación y despliegue de una API Restful como parte del componente de lógica. Esta API tiene como objetivo gestionar los datos y recursos necesarios para crear juegos interactivos para el refuerzo del idioma inglés en niños de 6 a 8 años. Esta aplicación utilizó la metodología de desarrollo Rapid Application Development (RAD) teniendo en cuenta que sus etapas se apegan al ciclo de vida de una API. La aplicación fue desarrollada aplicando la arquitectura limpia y los principios SOLID con Python y Flask. Las pruebas realizadas en el componente constan de pruebas unitarias para verificar y validar los resultados esperados de cada funcionalidad, y de las pruebas de rendimiento con el sistema desplegado para determinar los tiempos de respuesta de la API en un ambiente de producción. Los resultados obtenidos muestran un resumen de la funcionalidad desarrollada y los resultados de las pruebas. Las conclusiones muestran una comparativa entre el trabajo realizado y el trabajo planteado. Las recomendaciones dan consejos y acciones a tomar en base a la experiencia obtenida a lo largo del desarrollo del proyecto.

**PALABRAS CLAVE:** Enseñanza de inglés, API REST, RAD, Arquitectura limpia, Python

# ABSTRACT

This document presents the results of the research, requirements definition, implementation and deployment of a Restful API as part of the logics component. The main goal of the API is to manage the data and resources needed to create interactive games for English knowledge reinforcement in 6 to 8 years old kids. The application used the Rapid Application Development (RAD) methodology taking in count that its stages match the ones in the API lifecycle. The application was developed using clean architecture and SOLID principles with Python and Flask. The test performed in this component consist of unit tests to validate and verify the expected results of each functionality and performance tests to determine the API response times in a production environment. The obtained results show a resume of developed functionality and tests results. The conclusions show a comparative between the job done and the planned one. The recommendations provide of actions and advice based on the experience obtained during the project execution.

**KEYWORDS:** English teaching, API REST, RAD, Clean architecture, Python

# 1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO

## 1.1 Planteamiento del problema

Actualmente el inglés es el idioma más con más aplicaciones en áreas como los negocios, la comunicación entre personas de diferentes países, el desarrollo tecnológico, la publicación de artículos científicos, entre otros. Este hecho plantea la necesidad de aprender este idioma para incorporarse a las diferentes áreas antes mencionadas.

La importancia del aprendizaje del idioma inglés no es algo desconocido en Ecuador. Esto se ve reflejado en que, en el pensum académico del Ministerio de Educación del 2016, se incluye la formación en una lengua extranjera desde las etapas tempranas del desarrollo académico de los niños [1].

Respecto al nivel de dominio de inglés en el país y en el contexto de la edición del 2023 del índice de suficiencia de inglés (EPI) de la empresa Education First (EF) [2], Ecuador tiene un nivel bajo al ubicarse en la posición 80 de los 113 países que formaron parte del estudio. Esto podría ser el resultado de varios factores que afectan la enseñanza del idioma inglés [3]. Factores del entorno como la motivación en el ambiente familiar, o condiciones médicas como la dislexia o el trastorno del aprendizaje no verbal pueden afectar no solo el aprendizaje del idioma inglés, si no el aprendizaje en general [3].

A pesar de que el idioma inglés se enseña desde los primeros años de los niños, el índice EFI muestra que hace falta mejorar los métodos de aprendizaje usados en la enseñanza del idioma inglés.

Según los autores de esta investigación [3], el uso de TICs puede mejorar el aprendizaje respecto a temas de motivación dado que muchos estudiantes tienden a mostrar más interés debido a lo llamativo de los dispositivos digitales. En el estudio presentado en [4], que consistió en encuestas realizadas a docentes especializados en impartir la materia del idioma inglés respecto al uso de herramientas digitales interactivas en el aula de clase, se llegó a la conclusión de que las habilidades lingüísticas en la clase pueden mejorarse con el uso de herramientas digitales.

El uso de juegos para la enseñanza es un enfoque interesante respecto al uso de herramientas digitales. Los juegos son atractivos y, gracias a esto, captan la atención de los niños, por lo tanto, es posible utilizar esta atención para impartir conocimiento mientras juegan y relacionan los temarios con actividades lúdicas de su interés.

Para comprobar esta hipótesis, en años previos se realizaron proyectos que consistieron en el desarrollo de aplicaciones móviles con la temática de enseñar temas básicos del idioma inglés. Algunos de estos proyectos se encuentran listados en la **Tabla 1**.

**Tabla 1.** Aplicaciones móviles para el aprendizaje de inglés en niños

| **Nombre** | **Año** | **Descripción** |
|---|---|---|
| EWORD [5] | 2020 | Juego móvil para el aprendizaje de vocabulario en inglés para niños de 6 a 12 años. |
| VocaGame [6] | 2019 | Juego móvil para el aprendizaje de vocabulario en inglés para niños de 8 a 11 años. |
| LLG [7] | 2022 | Juego móvil para el aprendizaje de vocabulario, lectura y pronunciación en inglés para niños de 11 años. |

Teniendo en cuenta el problema de aprendizaje del idioma inglés en Ecuador, se propone el desarrollo de un juego interactivo en el contexto de la educación inicial y básica para el refuerzo académico en la asignatura de inglés que permita motivar a los niños y a sus padres para incentivar desde edades tempranas la adopción de este idioma. La propuesta encuentra su justificación con el precedente de la realización de este tipo de aplicaciones y sus efectos positivos, así como la apreciación de los beneficios del uso de la tecnología en la enseñanza por parte de docentes especializados en esta materia.

Este componente en específico se encargará de receptar, procesar y almacenar datos requeridos por los juegos para cumplir su objetivo de enseñar el idioma inglés a través del desarrollo de una API Restful que proveerá sus servicios principalmente al aplicativo móvil desarrollado a la par de este componente.

## 1.2 Objetivo general

Desarrollar un servicio web que exponga diferentes funcionalidades para el manejo de los datos de los usuarios, contenido educativo en inglés y los recursos multimedia requeridos y así facilitar la realización de un juego interactivo móvil.

## 1.3 Objetivos específicos

1. Realizar una investigación que permita conocer las necesidades de los niños de 6 a 8 años respecto al aprendizaje del idioma inglés para determinar las funcionalidades que debería exponer el servicio web.

2. Implementar el servicio web teniendo en cuenta los principios SOLID para facilitar el mantenimiento, la escalabilidad y la modificabilidad del código.
3. Realizar pruebas unitarias y de rendimiento para validar que se cumple con la adecuación funcional y la eficiencia.

## 1.4 Alcance

Este componente comprende el desarrollo y despliegue de una API con la arquitectura Restful que permita gestionar los datos de los usuarios de los juegos, así como los datos requeridos por la aplicación cliente para crear los juegos.

El proyecto será desarrollado con la metodología RAD dado que sus etapas se ajustan a la necesidad del proyecto de una definición iterativa de los entregables a través de prototipos y de una etapa específica de desarrollo en la que se minimizan los cambios sin bloquear el regreso a una etapa previa si se requiere.

El proyecto terminará con la entrega del servicio desplegado junto con las pruebas de rendimiento realizadas y la documentación de la API.

## 1.5 Marco teórico

### 1.5.1 Juegos educativos para el aprendizaje de idiomas

Hoy en día, el aprendizaje de idiomas, especialmente el inglés, se ha convertido en una necesidad urgente. Por lo que, la incorporación de tecnologías móviles en la educación ha permitido la creación de nuevas formas de mejorar la adquisición de idiomas entre los estudiantes. Uno de estos enfoques es la implementación de juegos educativos en dispositivos móviles que combinan elementos de gamificación con el contenido educativo, lo que los hace interactivos y atractivos para los usuarios [5].

#### *1.5.1.1 Beneficios y objetivos*

Los juegos móviles educativos tienen como objetivo principal motivar a los estudiantes a aprender inglés. Estos juegos pueden mejorar la retención de vocabulario y estructuras lingüísticas porque crean un entorno de aprendizaje competitivo e interactivo. Los juegos móviles tienen el potencial de superar las limitaciones de los métodos de enseñanza tradicionales, como el tiempo limitado y la falta de interacción activa. Se puede crear una herramienta educativa que no solo hace el proceso de aprendizaje más atractivo, sino que también se adapta a los diferentes niveles y contextos culturales de los estudiantes al combinar la tecnología móvil con el aprendizaje de idiomas [5].

#### *1.5.1.2 Diseño y desarrollo*

- **Contexto cultural:** La adaptación del contenido educativo al contexto cultural de los usuarios es crucial para la efectividad del aprendizaje. Esto implica diseñar juegos que reflejen los valores y referencias culturales de los estudiantes. La localización del contenido puede incluir la adaptación de historias, personajes y escenarios para que sean relevantes y familiares para los usuarios finales [6].
- **Elementos de gamificación:** Incluir elementos de gamificación, como puntos, niveles, y recompensas, puede aumentar la motivación y el compromiso de los estudiantes. Estos elementos hacen que el proceso de aprendizaje sea más dinámico y entretenido. La integración de desafíos y metas incrementales puede mantener a los estudiantes motivados y comprometidos con el proceso de aprendizaje [6].
- **Evaluación de usabilidad:** Es fundamental evaluar la usabilidad de los prototipos de juegos educativos para asegurar que sean intuitivos y accesibles para los estudiantes. Las evaluaciones deben considerar la facilidad de uso, la satisfacción del usuario, y el impacto en el aprendizaje. Pruebas piloto y estudios de usabilidad pueden proporcionar retroalimentación valiosa para mejorar el diseño y la funcionalidad de los juegos [6].

#### *1.5.1.3 Modelos teóricos y marcos de trabajo*

La investigación sobre juegos educativos para el aprendizaje de idiomas ha identificado varios modelos teóricos y marcos de trabajo que pueden guiar el diseño y desarrollo de estos juegos:

- **Modelo de aprendizaje basado en juegos (GBL)**: Este modelo enfatiza el uso de mecánicas de juego para apoyar objetivos educativos. Se enfoca en crear experiencias de aprendizaje que sean tanto atractivas como educativas, integrando principios de diseño de juegos con teorías educativas [7].
- **Teoría de la carga cognitiva:** Este enfoque sugiere que los diseñadores deben equilibrar la carga cognitiva impuesta por el juego para no abrumar a los estudiantes, permitiendo una mejor asimilación del contenido educativo. Es crucial diseñar juegos que proporcionen un desafío adecuado sin ser demasiado difíciles, para mantener el interés sin causar frustración [7].

- **Aprendizaje adaptativo:** Los sistemas de aprendizaje adaptativo utilizan datos de interacción del usuario para ajustar el contenido y la dificultad en tiempo real. Este enfoque puede ser particularmente útil en juegos educativos, permitiendo una personalización del aprendizaje que se ajuste a las necesidades individuales de cada estudiante [7].

El uso de juegos móviles educativos representa una estrategia prometedora para mejorar el aprendizaje de idiomas. La combinación de tecnología móvil y elementos de gamificación puede superar las limitaciones de los métodos tradicionales de enseñanza, proporcionando un ambiente de aprendizaje interactivo y atractivo. Sin embargo, es esencial considerar el contexto cultural y desarrollar modelos holísticos que guíen el diseño y desarrollo de estos juegos para maximizar su efectividad [7].

El sistema propuesto fue desarrollado con base en los siguientes conceptos, herramientas, tecnologías y prácticas:

### 1.5.2 API

Una API o interfaz de programación para aplicaciones es un contrato entre diferentes sistemas o aplicaciones para que estas se comuniquen entre sí sin la interacción directa de un usuario [8]. Este contrato define como el proveedor expone el servicio y como el cliente debe consumir este servicio [8] como los protocolos, los datos de entrada, los datos de salida.

Otra forma de definirlo es como un contrato entre el software y el desarrollador que lo usa [9]. Por ejemplo, se puede hablar de la API de un sistema operativo que utiliza un desarrollador para mostrar componentes interactivos en su aplicación o el portapapeles de ese sistema operativo [8]. Este concepto es independiente de la tecnología que lo implementa. Pueden existir diferentes tipos de API según el tipo de aplicaciones que se intenta crear.

Su objetivo es proveer funcionalidades específicas sin que se conozcan detalles de la implementación específica [8].

Por ejemplo, para mostrar una alerta en un navegador, JavaScript provee de la función ***alert*** cuyo comportamiento es el mismo en todos los navegadores independientemente de su implementación en cada uno de ellos.

```
alert("Está a punto de pasar algo...");
console.log("Alerta emitida");
```

**Figura 1.** API de JavaScript para mostrar una alerta. Fuente: autor.


**Figura 2**. Alerta mostrada con JavaScript en el navegador. Fuente: autor.

Una API debe tener un contrato bien definido y ser inmutable para que sea de utilidad a largo plazo. Los cambios dentro de una misma versión liberada de una API deben limitarse a correcciones de errores. Cualquier cambio al contrato de una API supondrá la liberación de una nueva versión.

### 1.5.3 API Resful

Una API Restful es una API expuesta a través del protocolo HTTP que sigue los principios REST: Representational State Transfer [8], [10]:

- **Cliente-Servidor**
  - Arquitectura donde varios clientes consumen servicios de un servidor.
  - El cliente hace una petición y el servidor responde; cada petición tiene una respuesta.
  - El servidor procesa peticiones entrantes y retorna resultados.
- **Interfaz uniforme:** cada recurso tiene un nombre y un método HTTP [10].
  - **Identificación:** Un recurso tiene una URI, p.ej., /users para “Usuario” [10].
  - **Manipulación:** El cliente debe poder manipular la respuesta; formato definido (JSON, HTML, XML) [10].
  - **Mensajes autodescriptivos:** Cada mensaje contiene toda la información necesaria para su procesamiento [10].
  - **Hipermedia como motor del estado**: Las respuestas pueden incluir enlaces para acciones relacionadas [10].

- **Sin estado:** El servidor no necesita información del estado del cliente; toda información necesaria es provista por el cliente [10].
- **Uso de caché:** Almacenar resultados de peticiones frecuentes para evitar procesamiento innecesario. Puede ser en el servidor, cliente, CDN, u otros [8].
- **Sistema con capas:** Un servidor puede estar detrás de otros servicios; el cliente no debe saber si existen servicios intermedios [8].
- **Código bajo demanda:** Permite la ejecución de programas en el cliente enviados desde el servidor. Debe ser controlado para evitar ataques [8].

### 1.5.4 Ciclo de vida de una API

Una API, al igual que otros productos de software, tiene un ciclo de vida. Este ciclo de vida contiene todos los hitos importantes dentro del tiempo útil de una API.

Etapas del ciclo de vida:

- **Análisis:**
    - **Definir objetivos de negocio y funcionalidades**: Se identifican los objetivos que la API debe cumplir para impulsar las operaciones de la empresa o descubrir nuevas oportunidades [9], [11].
    - **Decidir el modelo de acceso:** Se determina si la API será Pública (accesible a todos), Privada (uso interno) o Asociada (para socios específicos) [9], [11].
    - **Identificar partes interesadas, clientes, casos de uso y experiencia del desarrollador:** Se analizan quiénes serán los usuarios de la API y cómo será su interacción con ella [9], [11].
- **Desarrollo:**
    - **Definir endpoints y métodos de integración:** Se establecen las rutas específicas para acceder a las funcionalidades de la API y se consideran las mejores prácticas para la integración [9], [12].
    - **Ofrecer un SDK si es necesario:** Se proporciona un kit de desarrollo para facilitar a los desarrolladores la integración de la API, especialmente útil para aplicaciones móviles [9], [12].

- **Considerar seguridad, autenticación y autorización:** Se implementan medidas de seguridad para proteger el acceso a la API, asegurando que solo usuarios autorizados puedan utilizarla [9], [12].

- **Operaciones:**
  - **Promover la API para atraer desarrolladores**: Se aplica marketing de APIs para hacerla atractiva y fomentar su uso [8].
  - **Proveer documentación:** Se crea una guía detallada que explica cómo utilizar la API [12].
  - **Registro de aplicaciones:** Se establece un sistema sencillo para registrar aplicaciones y gestionar la seguridad [12].
  - **Consola de pruebas:** Se ofrece una plataforma para que los desarrolladores puedan probar la API antes de integrarla [12].
  - SDK y recursos externos: Se proporcionan herramientas y recursos adicionales que faciliten el uso de la API [12].

- **Retiro:**
  - **Informar a los desarrolladores sobre el retiro de la API**: Se notifica con antelación para que los desarrolladores puedan adaptarse a cambios o buscar alternativas [12].
  - **Razones para el retiro:**
    - **Falta de uso:** Si la API no es utilizada suficientemente para justificar su mantenimiento [12].
    - **Oposición a objetivos de negocio:** Si la API interfiere con las operaciones centrales o la competitividad de la empresa [12].
    - **Cambios tecnológicos:** Si la tecnología en la que se basa la API queda obsoleta [12].
    - **Nuevas versiones:** Si se lanzan versiones nuevas con mejoras significativas que reemplazan a las anteriores [12].

### 1.5.5 Rapid Application Development

El desarrollo rápido de aplicaciones o *Rapid Application Development* es una metodología de desarrollo de software que se basa en el prototipado y en el desarrollo iterativo para suplir los requerimientos de un proyecto. Busca reducir el costo y tiempo de desarrollo al presentar prototipos funcionales a los clientes para obtener retroalimentación [13].

Las fases de RAD son las siguientes:

- **Planificación:** El equipo se reúne con las partes interesadas para identificar los requerimientos, necesidades y expectativas de los usuarios [14] a través de un análisis de los requerimientos funcionales, no funcionales, requerimientos del negocio y las restricciones del proyecto [13].
- **Diseño:** Se crea prototipos en iteraciones de diseño, implementación y pruebas. Se muestran los prototipos funcionales para obtener retroalimentación del usuario y de las partes interesadas [13]. Además, se apunta a diseñar una arquitectura de software adecuada y efectiva [14].
- **Construcción:** Se desarrolla el sistema o producto, se enfatiza la velocidad y la eficiencia. Se utilizan herramientas y frameworks existentes, así como desarrollo iterativo, pruebas frecuentes y retroalimentación del usuario para asegurar de que se cumplen los requerimientos [13]. Durante esta fase, el equipo debe resolver todos los problemas que surgen a medida que avanza el desarrollo [14].
- **Despliegue/manejo:** Se realiza la transición desde el ambiente de desarrollo a un ambiente de prueba o producción según las necesidades del proyecto [14]. Esta etapa involucra el despliegue, las pruebas y el entrenamiento del usuario [13]. A través de las diferentes pruebas se identifican y resuelven los problemas que quedan. Después de las pruebas finales, el sistema es desplegado [13].

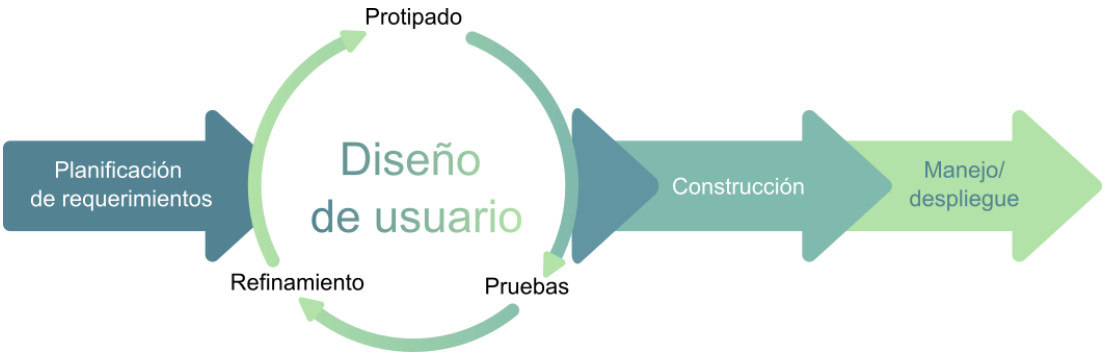



**Figura 3.** Etapas de RAD

### 1.5.6 Arquitectura limpia

La arquitectura limpia es un tipo de arquitectura propuesto por Robert C. Martin que se basa en la división por capas, cada una con diferentes responsabilidades. Busca una arquitectura desacoplada de la tecnología subyacente utilizada para interactuar con el usuario y para comunicarse con otros sistemas. El término “limpio” hace referencia que, en sus capas más internas, no existen dependencias o acoplamiento a detalles específicos de la tecnología utilizada para construir la aplicación o sistema. Entonces se puede tener una representación “limpia” del modelo de negocio en el cual las reglas no cambian debido al cambio de la tecnología usada en su desarrollo [16].

Esta arquitectura propone las siguientes capas:

- **Dominio:** Es la capa base, representa las reglas del dominio, las entidades que interactúan y las interfaces que deben utilizarse para comunicarse con esta capa. Es el centro de la aplicación [16].
- **Casos de uso (o aplicación):** Representa la orquestación de las funcionalidades que se plantea desarrollar. Controla el flujo e interacción entre los objetos de dominio junto con otros casos de uso o sistemas externos a través de interfaces. Contiene la lógica principal de la aplicación y las reglas del negocio [16].
- **Infraestructura:** En esta capa se encuentran las implementaciones específicas de las interfaces definidas en las capas interiores. Se encarga de proveer los detalles específicos de una tecnología en particular [16].
- **Exterior:** Esta capa contiene todo el mundo exterior a la lógica central de la aplicación, se puede entender como la interacción directa con los sistemas exteriores y los usuarios de la aplicación. En este nivel se encuentran las UI, llamadas a bases de datos específicas (SQL, NoSQL, etc.), controladores web, pantallas táctiles entre otros [16].

En esta arquitectura existe una regla respecto la interacción entre las capas propuestas, la regla de dependencia. Esta regla establece que la dependencia entre capas inicia en las capas más externas y apunta a las capas más internas, en otras palabras, las capas más internas no pueden depender o conocer de las capas superiores. Sin embargo, las capas superiores pueden utilizar las capas internas sin ningún problema [16].

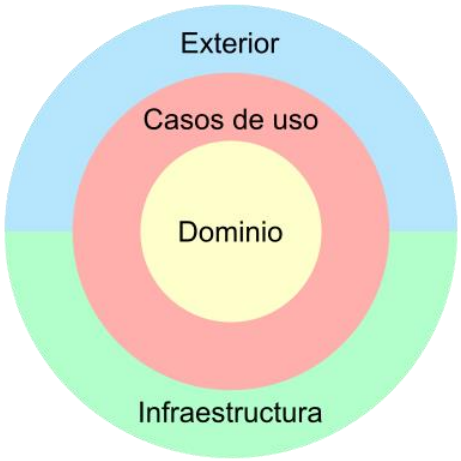



**Figura 4.** Capas de la arquitectura limpia

### 1.5.7 Principios SOLID

Los principios SOLID son guías para organizar las estructuras de datos, funciones, clases, objetos, entre otros elementos del software, con el fin de que el sistema creado sea tolerante al cambio, de fácil lectura y entendimiento y reutilizable por otros sistemas. Fueron creados por Robert C. Martin para guiar la construcción de software [16].

SOLID es un acrónimo para los siguientes principios:

- **Responsabilidad única (Single responsability):** cada módulo solo debe de encargarse de un solo actor, en este contexto, actor hace referencia a un grupo de interesados en una funcionalidad específica. Por lo tanto, esto hace referencia a que un módulo debería encargarse de una parte muy específica del dominio sin interferir directamente en funcionalidades que no le corresponden [16].
- **Abierto-Cerrado (Open-Closed):** un elemento de software debe estar abierto a la extensión, pero cerrado a la modificación. Esto implica que si existe un componente A que provee cierta funcionalidad, esta debe ser extensible por un componente B sin que la extensión signifique modificar la funcionalidad del componente A [16].
- **Sustitución de Liskov (Liskov sustitution):** este principio guía el uso de la herencia entre clases. Establece que, si tenemos un sistema que depende de una clase A y es hija de otra clase B, al intercambiar A por B, el comportamiento del sistema no cambia o falla [16].

- **Segregación de interfaces (Interface Segregation):** cada interfaz definida solo debe definir los métodos que le competen sin forzar a que las clases que la implementan tengan que proveer de lógica a métodos que no utilizan. Apunta a reducir la dependencia entre componentes al reducir las situaciones en las que los cambios en los métodos de una interfaz afectan a clases que no necesitan esos métodos, pero por la relación de herencia tienen que proveer de una implementación que podría causar un choque con la lógica propia de la clase que hereda los métodos desde la interfaz [16].
- **Inversión de dependencias (Dependency inversion):** este principio establece que los sistemas deben depender de abstracciones no de algo concreto. Esta dependencia en conceptos abstractos protege al sistema de los posibles cambios de aquellos módulos concretos que implementan a la abstracción y, por lo tanto, ayuda a construir sistemas más flexibles [16].

### 1.5.8 Python

Python es un lenguaje interpretado creado por Guido van Rossum a principios de 1990 como un sucesor del lenguaje ABC. Python tiene licencia de uso libre y compatible con GPL. Tiene una sintaxis sencilla, es dinámicamente tipado, soporta la orientación a objetos (aunque no se limita a este paradigma de programación), es ideal para el desarrollo de scripts y el desarrollo rápido de aplicaciones. El intérprete de Python puede ser extendido con estructuras de datos provenientes de lenguajes como C o C++ [17].

Su sintaxis está basada primordialmente en la gramática del idioma inglés con el objetivo de proveer una forma sencilla de entender el código creado con este lenguaje de programación. La definición de bloques no está delimitada por llaves si no por identación o espaciado [17].

La filosofía y uso recomendado de la sintaxis están determinadas por la guía de estilo de python llamada PEP en sus diferentes versiones. También es responsable por determinar las funcionalidades incluidas en el lenguaje en futuras versiones [17].

Es ampliamente usado en la ciencia de datos debido al soporte de varias librerías de terceros como Numpy o Pandas. Así mismo, tiene un gran soporte de la comunidad respecto al desarrollo de la Inteligencia Artificial, modelos de lenguaje, modelos de aprendizaje, entre otras aplicaciones similares [17].

### 1.5.9 Flask

Flask es un micro-framework para el desarrollo de aplicaciones web utilizando Python. La idea de Flask es brindar una base confiable y flexible al desarrollador en cuando a las decisiones respecto a cómo gestionar las dependencias, las librerías que decide usar y cómo las piensa usar. Flask no toma decisiones por el desarrollador más allá del motor de plantillas [18]. Toma ventaja del trabajo ya realizado por Werkzeug respecto a la implementación de la especificación WSGI de PEP 333 para la comunicación con servidores web como Apache o Nginx.

Entre las ventajas de Flask se tiene:

- **Flexibilidad:** Flask permite a los desarrolladores personalizar sus aplicaciones de acuerdo con los requisitos específicos del proyecto al no ser un framework con todo incluido [18].
- **Escalabilidad:** La capacidad de Flask para manejar aplicaciones de gran tamaño lo hace adecuado para proyectos que requieren un alto rendimiento y escala. Permite desarrollar aplicaciones modulares utilizando el concepto de Blueprint para facilitar la capacidad de crecer de la aplicación [18].
- **Documentación detallada:** La extensa y detallada documentación de Flask facilita el aprendizaje y la implementación del framework. Esta provee de ejemplos con código respecto a los diferentes aspectos que se pueden configurar en el framework, así como las diferentes formas en que se puede aprovechar su capacidad de extensión para resolver distintos problemas y necesidades que se presentan durante el desarrollo de una aplicación web [18].

### 1.5.10 GitHub

GitHub es una plataforma para alojar y manejar repositorios creados con el sistema de gestión de versiones Git. Se caracteriza por ser usada principalmente para la colaboración entre diferentes equipos y utilizar la funcionalidad de CI/CD provista por la plataforma.

- **Repositorio Centralizado:** GitHub proporciona un lugar centralizado para alojar repositorios Git, lo que facilita la colaboración y la gestión de proyectos [19].
- **Herramientas de Colaboración:** GitHub incluye herramientas como revisiones de código, solicitudes de extracción, y discusión de problemas, lo que facilita el trabajo en equipo y la comunicación [19].

- **Integraciones:** GitHub se integra con otras herramientas de desarrollo, como CI/CD (Integración Continua/Despliegue Continuo), lo que facilita el flujo de trabajo desde la codificación hasta el despliegue [19].
- **GitHub Actions:** Una característica que permite la automatización de tareas como pruebas, despliegues y otras acciones personalizadas dentro del flujo de trabajo [19].
- **Copilot:** Un bot potenciado con inteligencia artificial y con acceso a los repositorios alojados en la plataforma que provee de sugerencias de código según el contexto de cada archivo o proyecto [19].

### 1.5.11 MongoDB

MongoDB es una base de datos documental creada para facilitar el escalamiento y el desarrollo de aplicaciones [20].

**Base de documentos**

Un registro en MongoDB es un documento, es una estructura de datos compuesta por pares clave-valor similares a la estructura de JSON. Un campo puede incluir otros documentos, arreglos y arreglos de documentos [20].

Entre los beneficios de los documentos para almacenar datos se tienen los siguientes:

- Los documentos corresponden a tipos de datos nativos en varios lenguajes de programación.
- Los documentos embebidos pueden reducir el coste y la necesidad de los joins.
- El esquema dinámico permite el polimorfismo.

**Características principales**

- **Alto rendimiento:** El soporte de documentos embebidos reduce las operaciones de lectura/escritura en el sistema. Los índices soportan consultas rápidas y pueden incluir claves dentro de los documentos embebidos [20].
- **API de búsqueda:** La API provee de soporte para operaciones CRUD, agregaciones de datos, búsqueda de texto y consultas geoespaciales [20].

- **Alta disponibilidad:** MongoDB utiliza un conjunto de réplicas, son un conjunto de servidores de base de datos de MongoDB que mantienen los mismos conjuntos de datos para proveer redundancia y disponibilidad de los datos [20].
- **Escalabilidad horizontal:** MongoDB puede organizarse en clusters por regiones de datos basados en claves, las lecturas y escrituras ocurren en una misma zona [20].

### 1.5.12 Apidog

Es un conjunto completo de herramientas que conectan todo el ciclo de vida de una API y ayuda a los equipos a implementar las mejores prácticas para el diseño temprano de la API [21]. Intenta ser un reemplazo para varias herramientas al condensar funcionalidades de diseño, pruebas y documentación en un solo lugar.

Ofrece las siguientes funcionalidades:

- **Diseño colaborativo:** La plataforma provee una forma sencilla para colaborar en el diseño y especificación de la API siguiendo el enfoque API First para diseñar la API antes de implementarla.
- **Publicación de documentación:** La documentación creada puede ser compartida con otros desarrolladores utilizando diferentes formatos, entre ellos el acceso directo al proyecto de la API, la creación de la especificación formal de la API utilizando OpenAPI, entre otros.
- **Simulación/generación inteligente:** La información necesaria para ejecutar una petición puede ser generada en ese mismo instante tanto de manera manual como de manera automática, esto facilita la creación de datos de prueba para campo que se necesite al utilizar autocompletado y análisis de la estructura del objeto que se intente simular.
- **Pruebas automatizadas:** Las pruebas creadas en la aplicación pueden ser ejecutadas de manera automáticas sin que el usuario tenga que probar cada funcionalidad por sí mismo. Se generan reportes con los resultados de las pruebas según el tipo que se esté utilizando (funcionales o de rendimiento).

# 2 METODOLOGÍA

El desarrollo del proyecto se basó en la metodología RAD dado que permitió crear valor al cliente en poco tiempo por su naturaleza iterativa y la refinación de los requerimientos a través del prototipado despejando las dudas del usuario y recolectando información valiosa de su retroalimentación. Para elegir esta metodología se realizó una investigación comparativa respecto a las características de RAD con una metodología ágil como XP, esta comparación se puede ver en el **ANEXO I**.

Como se puede observar en la **Tabla 2**, el ciclo de vida de la API tiene etapas fácilmente relacionables con las etapas de la metodología RAD. Esta similitud permite sacar el máximo provecho de la metodología para apoyar el desarrollo de la API.

**Tabla 2.** Asociación entre las etapas de RAD y el ciclo de vida de una API

| **RAD** | **Ciclo de vida de una API** | **Justificación** |
|---|---|---|
| Requerimientos | Análisis | En ambas etapas se determinan las necesidades que deben ser suplidas con el producto a desarrollar. |
| Prototipado o diseño centrado en el usuario | Análisis | Los prototipos sirven para aclarar las dudas de las partes interesadas antes de empezar con la implementación real de la API. Se puede determinar una especificación de la API para el componente de interacción y refinarlo hasta que se quede de acuerdo que se representan los datos y recursos necesarios para cada punto de enlace. |
| Construcción | Desarrollo | En esta etapa de definen todos los detalles técnicos y se construye el producto final. |
| Cutover | Operaciones | En la etapa de despliegue de la API se realizan las pruebas a todo el producto y se la libera a producción. |

El proyecto inició con el levantamiento de requerimientos de los usuarios: niños de 6 a 8 años y sus padres. Se realizaron reuniones con diferentes niños para conocer cómo se

comportaban al utilizar distintas aplicaciones de aprendizaje de idiomas y, a partir de su experiencia, entender que es lo que les ayuda a aprender de forma sencilla. Con las notas y datos recopilados se realizó un proceso de análisis y extracción de requerimientos.

## 2.1 Requerimientos

Se utilizó el formato de historia de usuario para representar los requerimientos de la aplicación debido a que permite entender el valor que se pretende brindar al usuario.

Teniendo en cuenta que una API es una aplicación que brinda un servicio a otras aplicaciones, los requerimientos se enfocan en el desarrollador como usuario y, por extensión, sus necesidades de información y las funcionalidades que le permitan construir su aplicación o producto de software.

El análisis de los requerimientos del desarrollador se produjo a través de la comunicación con el responsable del desarrollo del aplicativo móvil para los niños, se realizaron reuniones en las que exponían las entradas y salidas que requería para lograr cumplir los objetivos de la aplicación y crear valor a sus clientes.

### HUD1: Temas en el juego

| **Identificador** | HUD1 | **Estimación** | 5 | **Prioridad** | Alta |
|---|---|---|---|---|---|
| **Título** | Temas en el juego | | | | |
| **Descripción** | | | | | |
| Como desarrollador de juegos, quiero acceder a un endpoint de la API que me permita obtener una lista completa de temas relacionados con el aprendizaje de inglés disponibles en el juego, para que los niños puedan reforzar los temas básicos adecuados a su edad. | | | | | |
| **Criterios de aceptación** | | | | | |
| • Cuando el desarrollador hace una petición GET al endpoint, recibe una lista de los temas en formato JSON.<br>• Cuando el desarrollador hace una petición GET al endpoint con el identificador de un tema, recibe los datos de un tema en formato JSON. | | | | | |

### HUD2: Logros en el juego

| **Identificador** | HUD2 | **Estimación** | 5 | **Prioridad** | Alta |
|---|---|---|---|---|---|
| **Título** | Logros en el juego | | | | |
| **Descripción** | | | | | |
| Como desarrollador de juegos, necesito implementar un sistema de logros a través de la API que motive a los niños a aprender inglés, proporcionando recompensas y reconocimientos por sus logros lingüísticos, de manera que los padres puedan seguir el progreso de sus hijos en el aprendizaje del idioma. | | | | | |
| **Criterios de aceptación** | | | | | |

- El desarrollador puede obtener la lista de los logros de un niño al enviar una petición GET con el identificador el niño.

### HUD3: Reporte de avances

| **Identificador** | HUD3 | **Estimación** | 5 | **Prioridad** | Alta |
|---|---|---|---|---|---|
| **Título** | Reporte de avances | | | | |
| **Descripción** | | | | | |
| Como desarrollador de juegos, preciso de endpoints en la API que me permitan generar informes detallados sobre el progreso de los niños en el aprendizaje de los diferentes temas de inglés, para que los padres puedan evaluar el rendimiento de sus hijos y proporcionar apoyo adicional según sea necesario. | | | | | |
| **Criterios de aceptación** | | | | | |
| • El desarrollador puede enviar una petición GET incluyendo el identificador de un niño para obtener un reporte su avance.<br>• El reporte que se genera está en formato JSON y contiene al menos los siguientes datos:<br>  ○ Tiempo promedio de resolución de los juegos.<br>  ○ Top 3 de los temas con más aciertos.<br>  ○ Top 3 de los temas con menos aciertos. | | | | | |

### HUD4: Cuentas para padres e hijos

| **Identificador** | HUD4 | **Estimación** | 5 | **Prioridad** | Alta |
|---|---|---|---|---|---|
| **Título** | Cuentas para padres e hijos | | | | |
| **Descripción** | | | | | |
| Como desarrollador de juegos, necesito implementar un sistema de cuentas internas dentro de la cuenta de los padres de familia, de manera que cada niño pueda tener su propio perfil personalizado y adaptado a su nivel de aprendizaje de inglés, permitiendo un seguimiento individualizado de su progreso lingüístico. | | | | | |
| **Criterios de aceptación** | | | | | |
| • El desarrollador puede obtener una lista de los niños asociados a una cuenta de padre a través de una petición GET.<br>• El desarrollador puede obtener los datos de un niño específico asociado a una cuenta de padre a través de una petición GET enviando el identificador del niño.<br>• El desarrollador puede guardar la información de una cuenta de niño que un padre creó en su aplicación a través de una petición POST al enviar los datos del niño y el identificador del padre.<br>• El desarrollador puede guardar la información de una cuenta creada por un padre a través de una petición POST al enviar los datos asociados a la cuenta creada. | | | | | |

| |
|---|
| • El desarrollador puede obtener la información de la sesión creada cuando un padre ingresa a su aplicación a través de una petición POST con las credenciales de su cuenta. |

### HUD5: Contenido multimedia

| **Identificador** | HUD5 | **Estimación** | 5 | **Prioridad** | Alta |
|---|---|---|---|---|---|
| **Título** | Contenido multimedia | | | | |
| **Descripción** | | | | | |
| Como desarrollador de juegos, quiero endpoints en la API que me permitan acceder a recursos multimedia, como imágenes y sonidos, diseñados específicamente para niños y relacionados con los temas de aprendizaje de inglés del juego, para crear una experiencia de aprendizaje lúdica y atractiva para los niños. | | | | | |
| **Criterios de aceptación** | | | | | |
| • El desarrollador puede obtener una imagen a través de una petición GET al incluir el nombre del archivo solicitado.<br>• El desarrollador puede obtener un archivo de audio a través de una petición GET al incluir el nombre del archivo solicitado. | | | | | |

## 2.2 Prototipos/Diseño de usuario

Durante esta etapa se realizaron distintos ciclos de definición del modelo de dominio que permitiría cumplir con los requerimientos del desarrollador. Con cada modelo la especificación de la API cambió para reflejar las restricciones y condiciones del negocio de tal forma que sean útiles para los desarrolladores que la quieran incorporar en sus aplicaciones.

### Ciclo 1: Modelado del dominio

El primer diagrama se creó con la idea de condensar todo el negocio de tal manera que el servicio se encargara de registrar las interacciones realizadas en las aplicaciones que consuman el servicio para proveer de diferentes métricas que se pudieran mostrar en un resumen general del uso de la aplicación.

Se incluyeron las entidades relacionadas a las cuentas tanto de padre como de niño, así como la entidad juego que representaría un conjunto de actividades a realizar.

Además, se incluyeron entidades que representaban a los recursos audiovisuales que iban a ser utilizados.

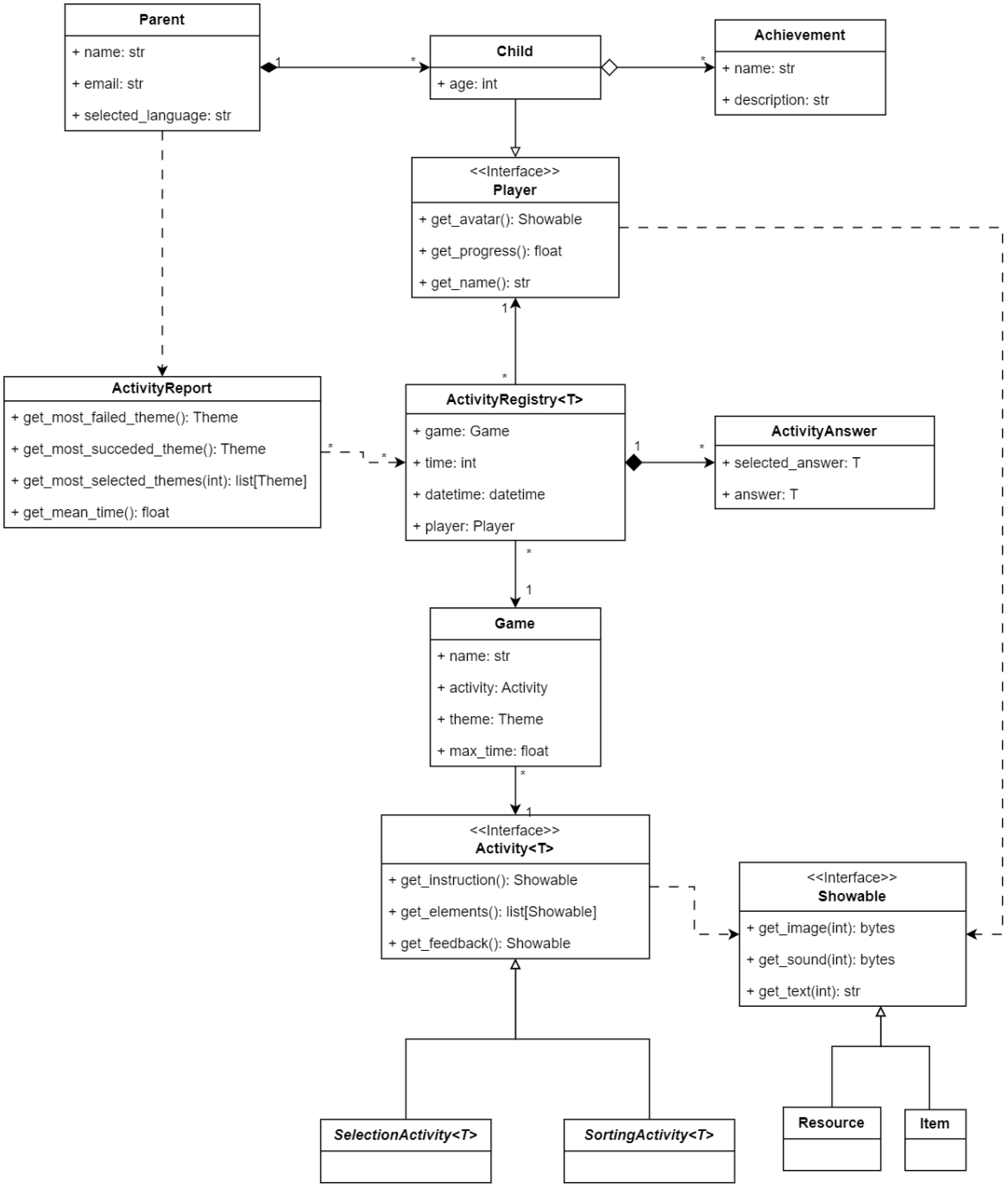



**Figura 5**. Primer diagrama de clases.

## Ciclo 2: Refinamiento del modelo

El diagrama de clases se rediseñó dado que, después de un análisis de la pertinencia de cada una de las clases representadas en el diagrama, aquellas relacionadas a las actividades intentan generalizar la interacción de un usuario con el juego y esto no le corresponde a este componente. Se simplificó el diagrama para tener en cuenta únicamente aquellas clases que representan las reglas de negocio sin tener en cuenta la interacción con el usuario.

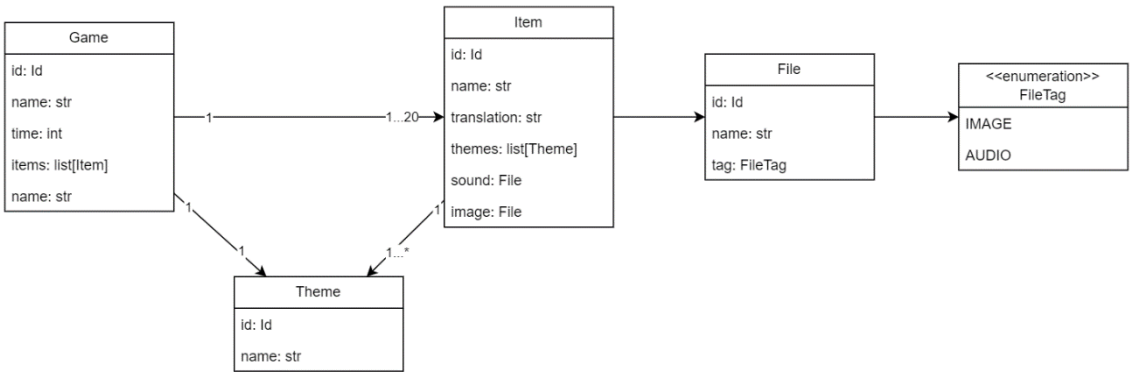



**Figura 6**. Diagrama de clases reducido.

El diagrama contiene a las clases que están involucradas directamente con el concepto de un juego, el vocabulario (llamado tema para adecuarse al concepto del tema de un juego) y los archivos asociados a los temas. El juego no debería almacenar en sí mismo los elementos asociados a un tema específico, en cambio, debería tener una relación únicamente con el tema y a este último solicitarle sus elementos.

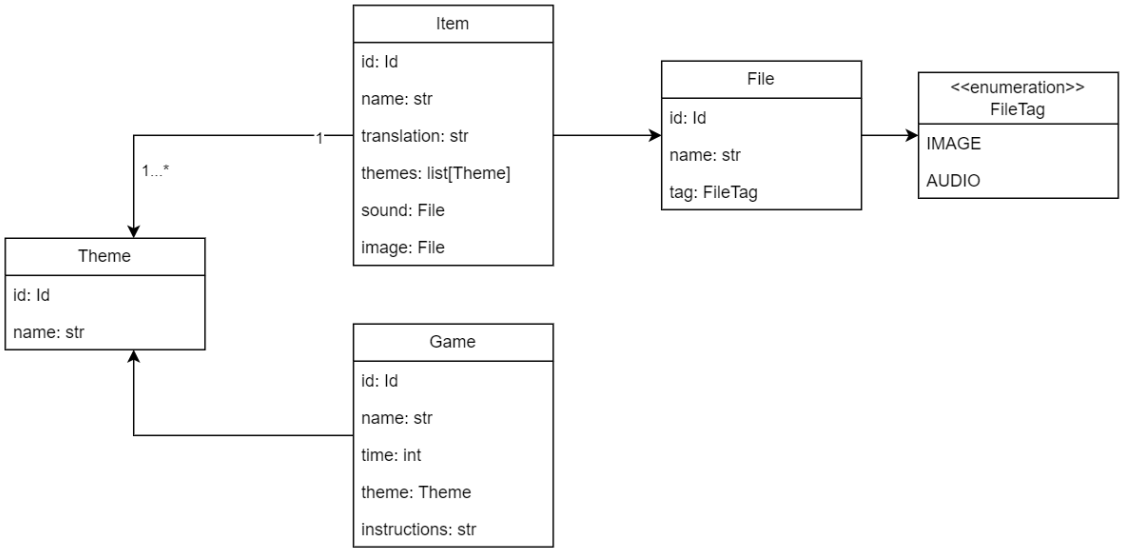



**Figura 7.** Diagrama de clases sin relación entre Item y Game.

Con este diagrama base se creó la primera especificación de la API con OpenAPI y Swagger para empezar la etapa de implementación.

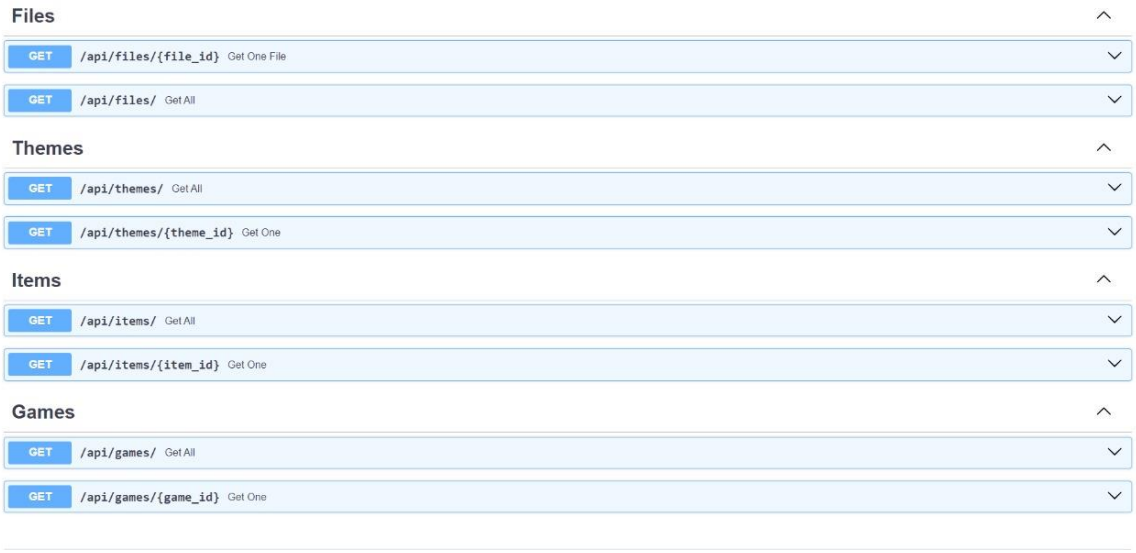



**Figura 8**. Interfaz de la especificación inicial de la API hecha con Swagger.

## Ciclo 4: Inclusión de las cuentas para los usuarios

Después de implementar las funcionalidades que permitían cumplir con la especificación desarrollada, fue necesario modificar el diagrama de clases para incluir aquellas relacionadas con el usuario y con la evaluación del desempeño de los jugadores.

Durante la modificación del diagrama se llegó a la conclusión de que la clase *Game* no aportaba nada más allá de servir como fachada de los temas sin dar información de valor al desarrollador. Por lo tanto, fue eliminada del diagrama.

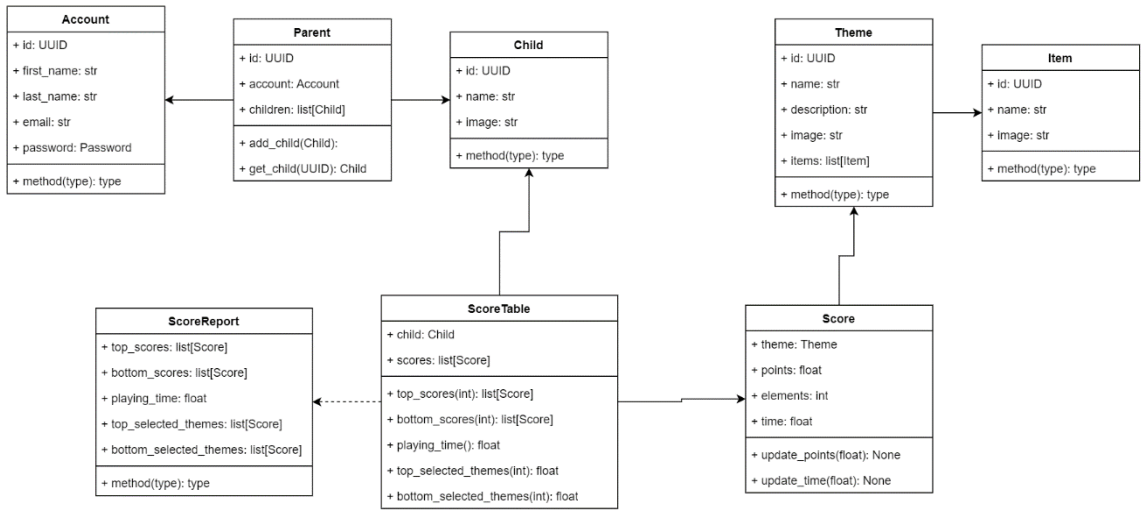



**Figura 9**. Diagrama de clases que incluye las cuentas y puntuaciones de los usuarios.

En esta versión del diagrama, se incluyeron las clases para el manejo del puntaje obtenido por un jugador en un tema específico. El desarrollador que consuma la API podrá

implementar su aplicación como crea conveniente y si desea utilizar el sistema de reportes sobre el desempeño, le basta con enviar la referencia del tema y el puntaje conseguido en un juego específico. Se definió que la manera de evaluación sea un puntaje para utilizar una mecánica que varios juegos ya implementan nativamente y también porque provee una forma cuantificable de medir el desempeño en el dominio de un tema específico.

### Ciclo 5: Simplificación del modelo

El diagrama se simplificó para representar únicamente aquellas clases y entidades que intervienen en el negocio directamente. Se agruparon las clases en paquetes que representan los módulos de la aplicación. Esta modificación permitió tener un modelo de domino que permitió representar de forma más precisa las reglas de negocio.

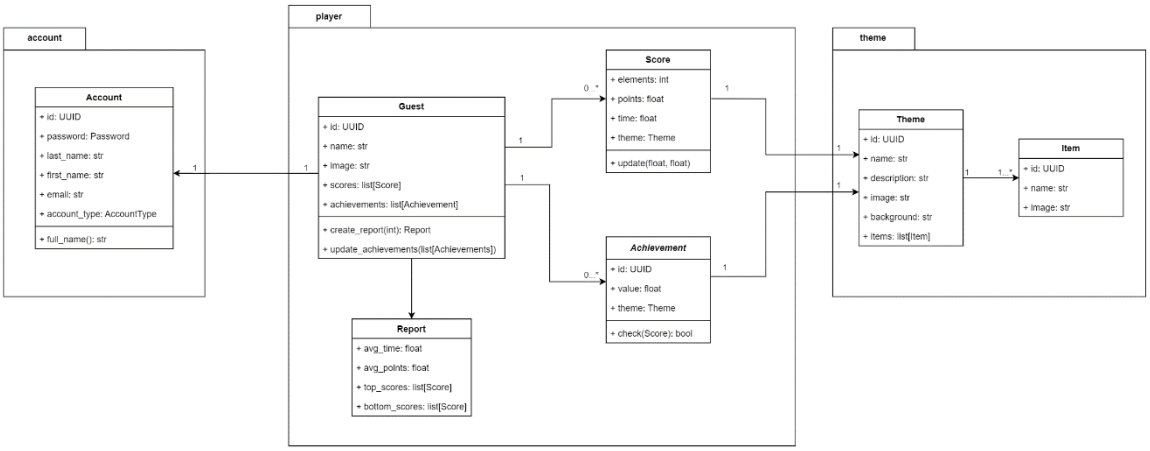



**Figura 10**. Diagrama de clases del negocio

## 2.3 Implementación

### 2.3.1 Estructura interna

Para seguir los principios de **la arquitectura limpia**, el proyecto se organizó en módulos que representan los conceptos más importantes del negocio y su estructura interna como una representación de las capas propuestas por la arquitectura limpia.

En la **Figura 11Figura 1** se muestra como el módulo *account* se organizó según las capas recomendadas por la arquitectura limpia.

En la carpeta *domain* se encuentran las clases referentes al dominio y los repositorios asociados o interfaces para la comunicación con otros módulos.

En la carpeta *application* se encuentran las clases que representan los servicios y casos de usos relacionados a las funcionalidades expuestas por el módulo, esta capa solo contiene referencias a las clases de la carpeta *domain* del mismo módulo y, de ser necesario, referencias a la capa de dominio o aplicación de otro módulo siguiendo la regla de dependencia de la arquitectura limpia.

En la carpeta *infrastructure* se encuentran las clases con las implementaciones específicas usadas en todo el módulo. Estas clases se instancian y se inyectan en el momento de la ejecución utilizando la técnica de la inyección de dependencias.

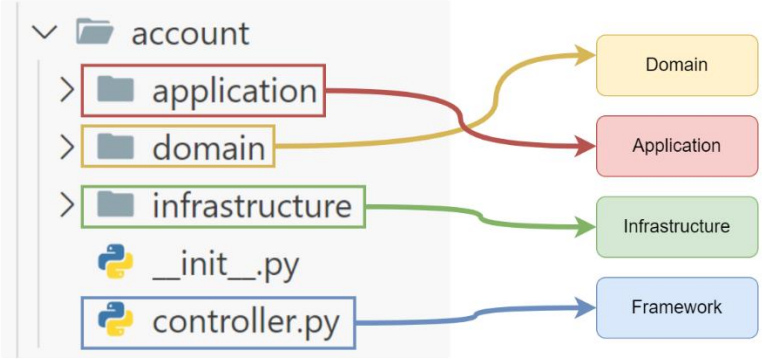



**Figura 11**. Estructura interna del módulo account.

### 2.3.2 Módulo “account”

Este módulo es el responsable de gestionar las cuentas que son registradas en la aplicación, se manejan únicamente las cuentas con credenciales asociadas (generalmente una cuenta que representa quien instaló la aplicación).

En la **Figura 12** se muestran aquellas clases que intervienen en la gestión de las cuentas de usuario. La lógica se encuentra dentro de la clase AccountService, esta misma depende de la interfaz IAccountRepository para que desacoplarse de la base de datos utilizada y que esta pueda ser reemplazada sin afectar la lógica principal de la aplicación. La sección azul representa la capa de infraestructura en la cual se implementa el repositorio de cuentas y el servicio de tokens utilizando el esquema JWT.

Este módulo implementa parte de la funcionalidad especificada en la historia de usuario HUD4 referente a las cuentas de padres e hijos. Se realizó la separación de las cuentas de padres e hijos debido a que estos últimos tienen más necesidades respecto a los datos que manejan y estos están estrechamente ligados a los datos tanto de los juegos como de los logros, por lo tanto, se encontró pertinente asignar un módulo específico para las cuentas de los jugadores (niños).

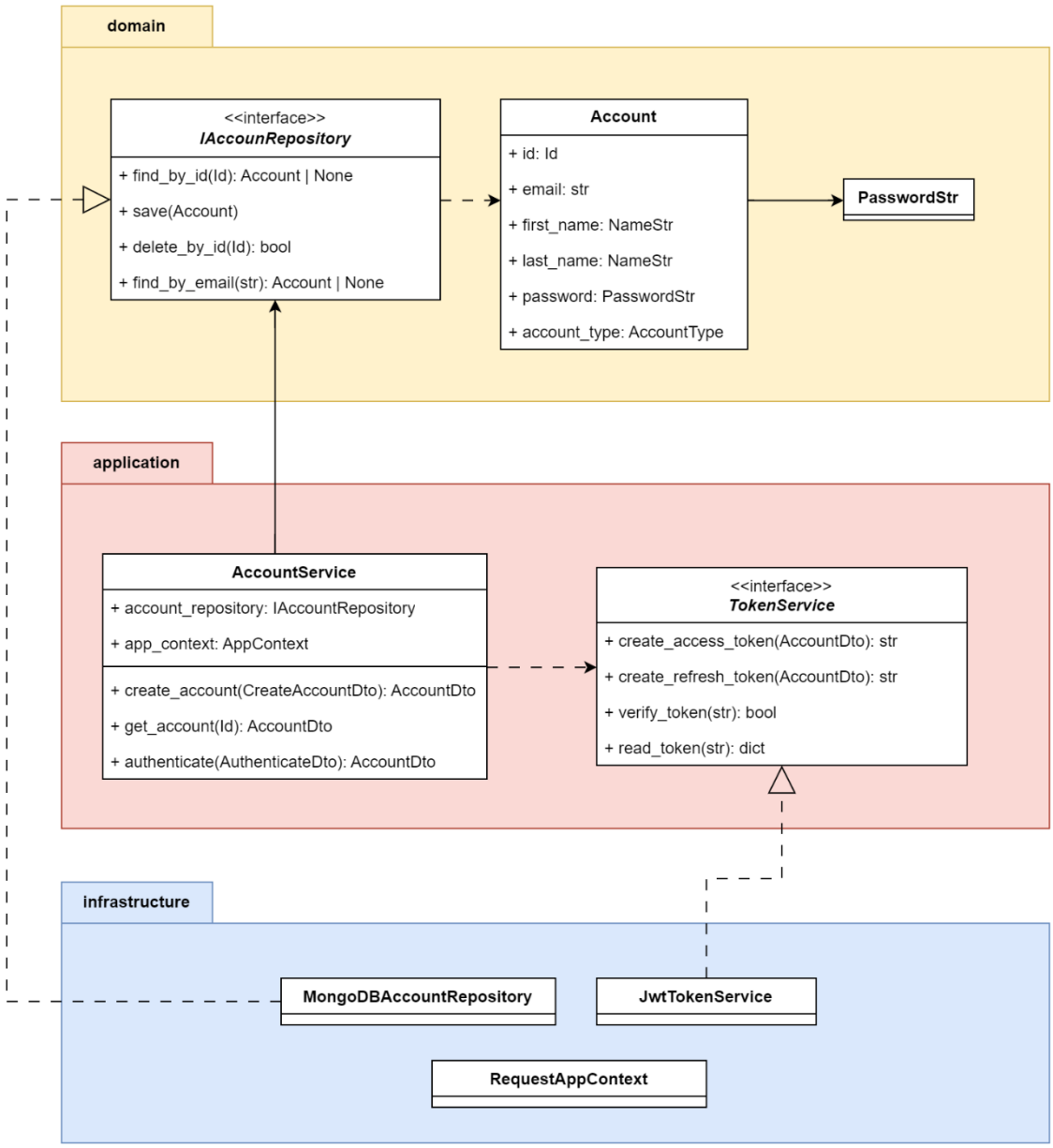



**Figura 12.** Diagrama de clases del módulo account

### 2.3.3 Módulo “player”

Este módulo es el encargado de gestionar las cuentas de los jugadores y sus logros. Aquí se manejan las cuentas, sus puntajes asociados y los logros que se obtienen después de cada juego.

Las cuentas de los jugadores se llamaron “Guest” o invitado debido a que el sistema puede manejar cuentas con credenciales que no necesariamente representan a padres de familia y podrían ser usadas para gestionar los datos del sistema como administradores. Por esta

razón, la relación entre una cuenta de anfitrión y una cuenta de invitado se encuentra en la última. Con esto se consigue que una cuenta anfitriona no tenga conocimiento de las cuentas invitadas asociadas a ella, dando la flexibilidad de que la misma entidad puede ser reutilizada para representar otros roles como un administrador o un gestor de contenido.

En la **Figura 13** se muestran las distintas capas utilizadas para modelar la lógica de negocio y la interacción con la fuente de datos sin tener ninguna dependencia fuerte de ningún tipo de tecnología específica.

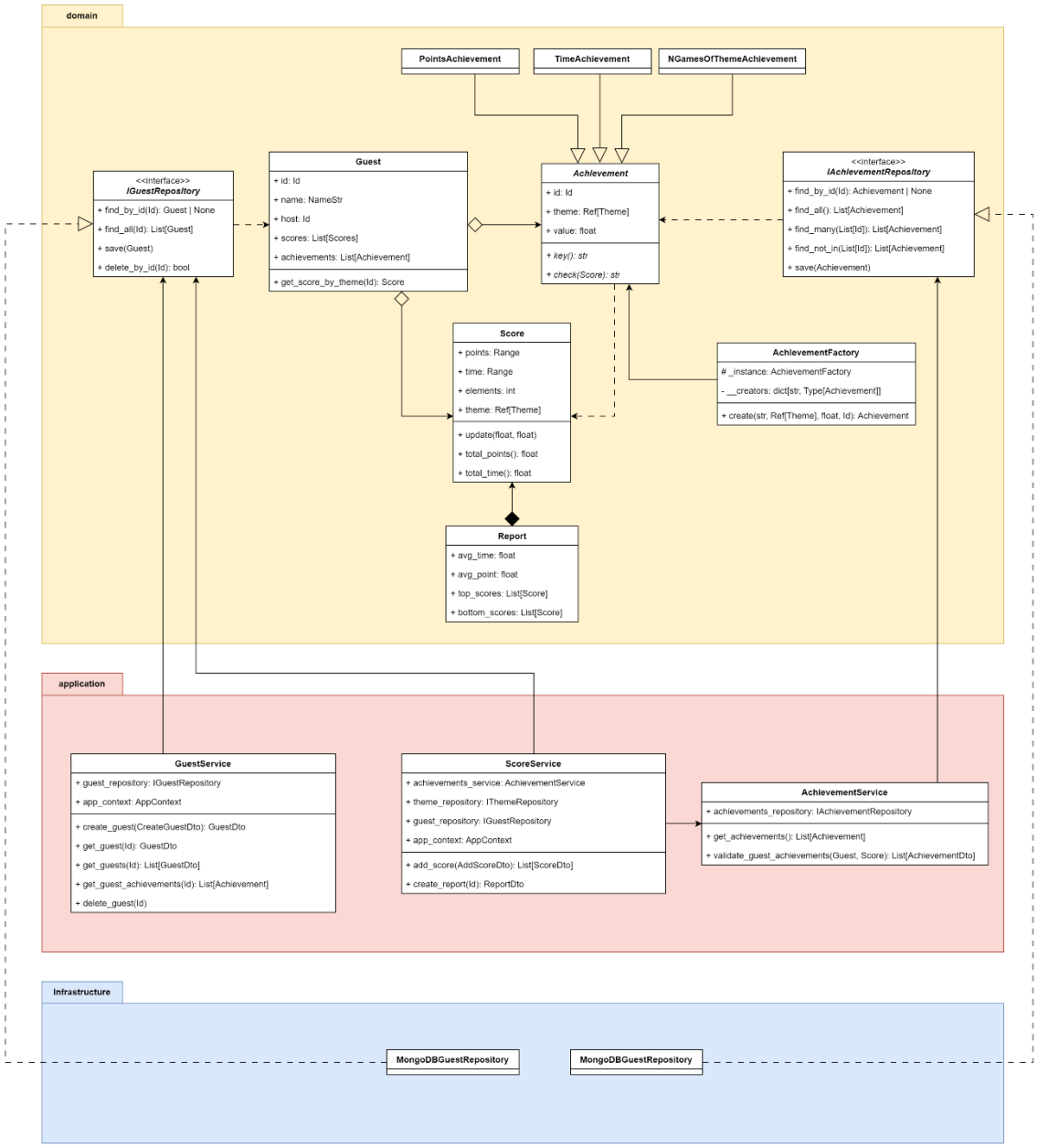



**Figura 13.** Diagrama de clases del módulo player

Este módulo permitió completar las funcionalidades de la historia de usuario HUD4, así como implementar las funcionalidades especificadas en las historias de usuario HUD2 y HUD3 relacionadas a los logros de los jugadores y el reporte de su avance.

### 2.3.4 Módulo “theme”

Este módulo es el encargado de manejar los temas que se van a presentar en la aplicación para reforzar el conocimiento de los jugadores. Se utilizó el término tema para referirse a un temario de palabras y frases. Cada tema está compuesto por ítems o elementos, estos elementos son las palabras o frases que se van a reforzar, por ejemplo: manzana es un elemento del tema comida, verde es un elemento del tema colores, entre otros.

En la **Figura 14** se pueden observar las diferentes capas del módulo y sus relaciones. Dado que se trata de un módulo relativamente simple, no se requirieron de entidades o modelos complejos de datos. Aun así, siguiendo los lineamientos de la arquitectura limpia, se utilizaron abstracciones para el acceso a la base de datos para evitar el acoplamiento.

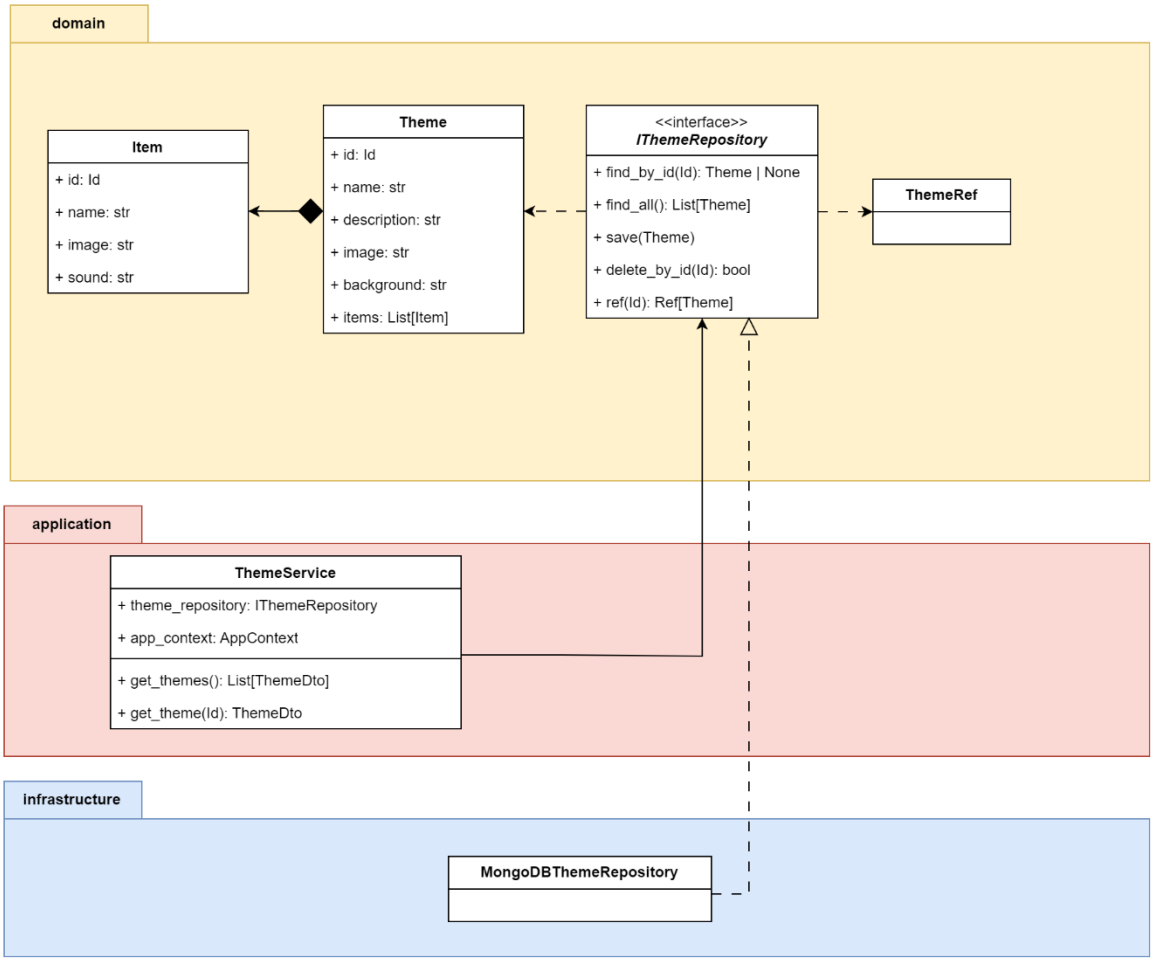



**Figura 14**. Diagrama de clases del módulo theme

Este módulo permitió implementar la funcionalidad especificada en la historia de usuario HUD1 referente a los temas que se van a crear juegos que permitan reforzar el conocimiento de estos en los niños.

### 2.3.5 Módulo “service”

El módulo de servicios se encarga de la interacción con servicios externos a través de abstracciones que permiten utilizar diferentes implementaciones. Para las necesidades del proyecto fue necesario únicamente comunicarse con un sistema de archivos externos para obtener la multimedia asociada a cada uno de los temas del módulo de temas.

En la **Figura 15** se puede observar la división de las diferentes capas de este módulo siguiendo la separación de responsabilidades y la regla de dependencia de la arquitectura limpia.

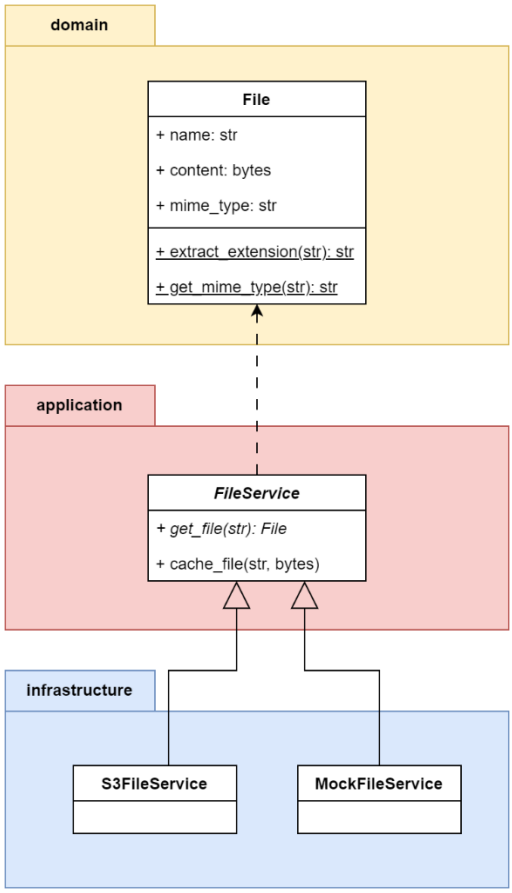



**Figura 15.** Diagrama de clases del módulo services

### 2.3.6 Principios SOLID

Durante la implementación fue importante aplicar correctamente los principios **SOLID** para crear código que sea mantenible, extensible, escalable y probable.

El principio de **responsabilidad única** se aplicó en las clases de dominio, servicio y aplicación para asegurar que cumplen únicamente la función que les compete (pudiendo estar dividida en subfunciones).

En la **Figura 16** se puede observar a la clase *Password* cuya función se limita a manejar las reglas aplicables a las contraseñas utilizadas en las cuentas de los usuarios. Esta clase evita que la lógica de las validaciones aplicadas a una contraseña tenga que recaer directamente en la clase que representa a una cuenta de usuario y, por lo tanto, sobrecargando a esta clase con más funciones de las que debería.

```python
class Password:
    hashed: str

    def __init__(self, password: str, hashed: bool = False):
        # If the password is already hashed, we store it as is
        # Otherwise, we hash it
        # This way, we allow the Password class to be used in two ways:
        # - Password("plain-text-password")
        # - Password("hashed-password", hashed=True)
        if hashed:
            self.hashed = password
        else:
            self.hashed = self._hash(password)

    def _hash(self, password: str) -> str:
        return bcrypt.hashpw(password.encode(), bcrypt.gensalt()).decode()

    def __eq__(self, other: object) -> bool:
        if isinstance(other, Password):
            return self.hashed == other.hashed

        if isinstance(other, str):
            return bcrypt.checkpw(other.encode(), self.hashed.encode())

        return False
```

**Figura 16.** Clase Password.

El principio **abierto-cerrado** se aplicó a través del uso de interfaces y clases abstractas. Se especifica la parte cerrada a la modificación, en el código está representado por los métodos que las interfaces definen para su futura implementación y la parte abierta a la extensión comprende a las clases que implementan las interfaces.

En la **Figura 17** se puede observar que la clase abstracta Achievement define un contrato para la verificación y para la identificación. El contrato no puede ser modificado por las clases hijas, sin embargo, puede ser extendido a través de diferentes implementaciones para los diferentes tipos de logros que puede obtener un usuario. Esto se hace más visible en la **Figura 18** que muestra a la clase *TimeAchievement* cuya lógica permite determinar si un usuario ha completado un tema en una cantidad específica del tiempo sin cambiar el contrato definido por su clase padre.

```python
class Achievement(ABC):
    id: UUID
    theme: Ref[Theme]
    value: float

    def __init__(self, theme: Ref[Theme], value: float, id: UUID = uuid4()):
        self.id = id
        self.theme = theme
        self.value = value

    @abstractmethod
    def key() -> str:
        # Key to identify the type of achievement in the factory
        pass

    @abstractmethod
    def check(self, score: Score) -> bool:
        # Check if the score meets the requirements of the achievement
        pass

    def __eq__(self, value: object) -> bool:
        return isinstance(value, Achievement) and value.id == self.id
```

**Figura 17.** Clase abstracta Achievement

```python
class TimeAchievement(Achievement):
    """
    Achievement class that checks if the player has completed a theme in a certain amount of time.
    """

    def key() -> str:
        return "min_time"

    def check(self, score: Score) -> bool:
        in_range = score.time.min <= self.value and score.time.min > 0
        is_theme = score.theme.id == self.theme.id
        return in_range and is_theme

    def __str__(self) -> str:
        return f"Complete {self.theme.value.name} in less than {self.value} seconds"
```

**Figura 18.** Clase TimeAchievement

El principio de **sustitución de Liskov** se puede ver implementado en la **Figura 18** al tener una clase padre con un comportamiento definido y el mismo comportamiento en el hijo. Esto permite que estas clases sean intercambiables durante la ejecución de la aplicación.

Las clases que requieran usar el comportamiento de los logros estarán trabajando transparentemente entre la clase *Achievement* y *TimeAchievement* al nunca enterarse que clase específica están usando. En la **Figura 19** se puede ver que se utiliza el patrón *Factory* para crear a las instancias de los logros según el tipo definido en la base de datos. Sin embargo, quien llame a este método estará esperando a la clase padre y, como se siguió el principio de sustitución de Liskov, el objeto que recibirá no causará un error en la aplicación.

```python
class DBAchievement(Document):
    id: UUID = Field(alias="_id")
    theme: UUID
    value: float
    key: str

    class Settings:
        name = "achievements"

    @inject
    def to_entity(self, theme_repo: IThemeRepository) -> Achievement:
        return AchievementFactory().create(
            id=self.id,
            theme=theme_repo.ref(self.theme),
            key=self.key,
            value=self.value
        )
```

**Figura 19.** Clase DBAchievement implementando el patrón Factory

El principio de **segregación de interfaces** se aplicó al no sobrecargar una interfaz con métodos que no le competen dado que causaría que la interfaz tenga demasiadas responsabilidades, adicionalmente, se estaría rompiendo el principio de responsabilidad única.

En la **Figura 20** se muestra como la interfaz *IAchievementRepository* tiene métodos específicos para su lógica de negocio. El método *find_not_in* tiene la responsabilidad de encontrar aquellos logros cuyos identificadores que no están en una lista de identificadores pasada como argumento. Esta responsabilidad no debería estar colocada en una interfaz general para todos los repositorios dado que no todas las entidades del dominio pueden ser accedidas de esta forma.

```python
class IAchievementRepository(ABC):

    @abstractmethod
    def find_all(self) -> list[Achievement]:
        pass

    @abstractmethod
    def find_by_id(self, id: UUID) -> list[Achievement]:
        pass

    @abstractmethod
    def find_many(self, ids: list[UUID]) -> list[Achievement]:
        pass

    @abstractmethod
    def find_not_in(self, ids: list[UUID]) -> list[Achievement]:
        pass

    @abstractmethod
    def save(self, entity: Achievement) -> None:
        pass
```

**Figura 20.** Clase IAchievementRepository

El principio de **inversión de dependencias** se aplicó a través de la definición de interfaces entre diferentes clases que requerían interactuar con sistemas externos. En la **Figura 21** se puede observar como la clase *AchievementService* utiliza la interfaz *IAchievementRepository* para acceder a la lógica para extraer los logros de un usuario y su implementación específica para interactuar con una base de datos de MongoDB en la **Figura 22**.

```python
class AchievementService:

    def __init__(self, achievements_repository: IAchievementRepository) -> None:
        self.achievements_repository = achievements_repository

    def get_achievements(self) -> list[Achievement]:
        return self.achievements_repository.find_all()

    def validate_guest_achievements(self, guest: Guest, score: Score) -> list[Achievement]:
        achievements = self.achievements_repository.find_not_in(
            [achievement.id for achievement in guest.achievements])

        def check_achievement(achievement: Achievement, guest: Guest, score: Score) -> bool:
            return achievement.check(score) and achievement not in guest.achievements

        return [achievement for achievement in achievements
                if check_achievement(achievement, guest, score)]
```

**Figura 21.** Clase AchievementService

```python
class MongoDBAchievementRepository(IAchievementRepository):

    def find_all(self) -> list[Achievement]:
        achievements = DBAchievement.find_all().run()
        return [achievement.to_entity() for achievement in achievements]

    def save(self, entity: Achievement) -> None:
        DBAchievement.from_entity(entity).save()

    def find_by_id(self, id: UUID) -> Achievement:
        achievement = DBAchievement.find_one(id=id).run()
        if not achievement:
            return None
        return achievement.to_entity()

    def find_many(self, ids: list[UUID]) -> list[Achievement]:
        achievements = DBAchievement.find({"_id": {"$in": ids}}).run()
        return [achievement.to_entity() for achievement in achievements]

    def find_not_in(self, ids: list[UUID]) -> list[Achievement]:
        achievements = DBAchievement.find({"_id": {"$nin": ids}}).run()
        return [achievement.to_entity() for achievement in achievements]
```

**Figura 22.** Clase MongoDBAchievementRepository

### 2.3.7 Esquema de seguridad en la API

#### *2.3.7.1 JWT y peticiones*

Se utilizó la tecnología JSON Web Token o JWT para identificar a los usuarios y determinar los permisos que tienen en el sistema para realizar una determinada acción o acceder a un recurso específico.

En la **Figura 23** se pueden observar las actividades que se ejecutan para obtener un token para la autenticación en futuras peticiones a la aplicación. El usuario envía su correo y contraseña para obtener un token que le permitirá ejecutar diferentes acciones según las acciones que está autorizado a realizar.

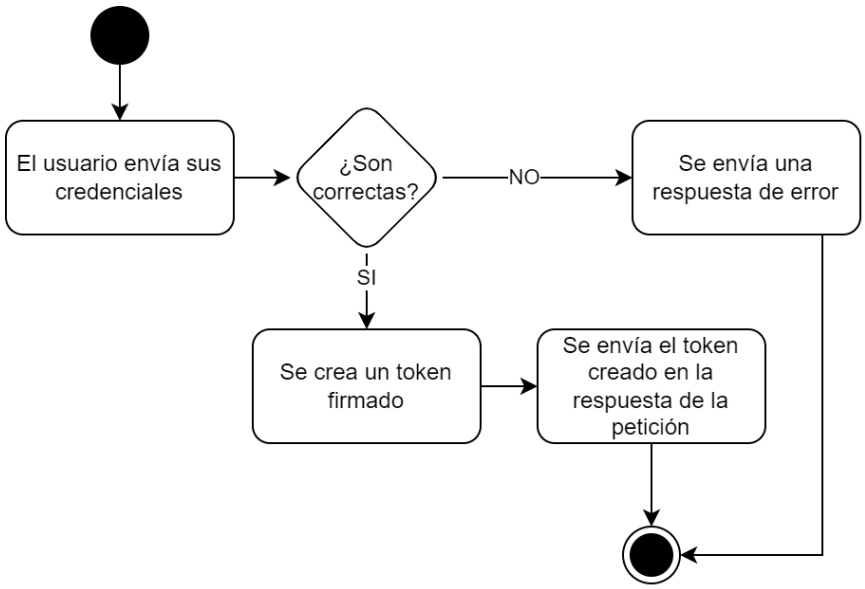



**Figura 23**. Diagrama de actividades de la obtención de tokens

En la **Figura 24** se muestran las actividades que se ejecutan para identificar al usuario, así como autorizarle a realizar la acción que describe la petición que realiza a la aplicación.

La verificación inicia con la revisión de la existencia del token en las cabeceras de la petición del usuario, en caso de no ser encontradas, no se puede identificar al usuario y, por lo tanto, se le negará el acceso a la aplicación.

Una vez comprobada la existencia de un token, se revisa si este es válido. El contenido del token y su firma son validados contra la estructura predefinida en la aplicación y la clave con la que se firman los tokens emitidos. Se negará el acceso a la acción que intenta ejecutar el usuario en el caso de que la autenticidad del token no pueda ser determinada.

Para autorizar la acción que el usuario intenta realizar, se revisa que el token contenga el nivel de acceso requerido para ejecutar dicha acción. Se procede a negar el acceso al usuario si el token provisto no cuenta con el nivel de acceso requerido para dicha acción.

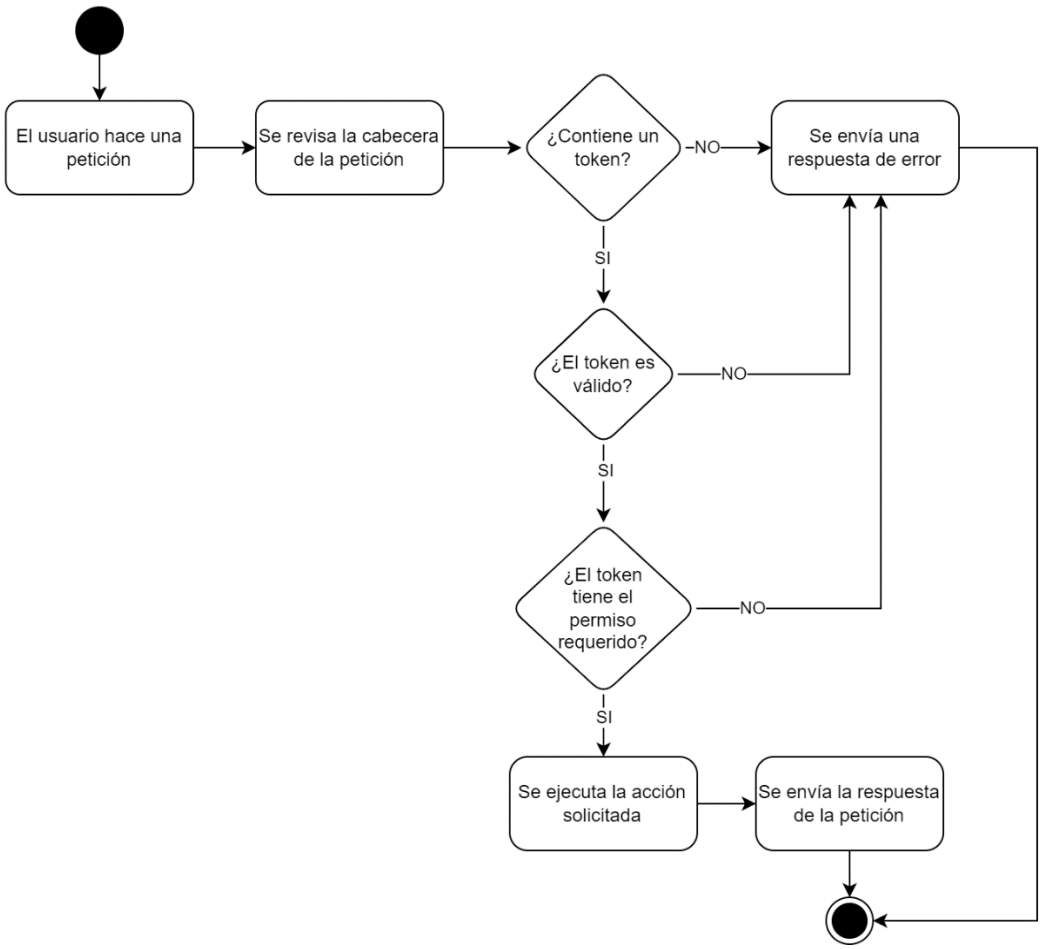



**Figura 24**. Diagrama de actividades para la autenticación y autorización del usuario con un token.

### *2.3.7.2 Validación de datos*

El contenido de las peticiones es validado a través de diferentes mecanismos para evitar problemas con los datos provistos por el usuario.

El primer mecanismo para la validación de los datos es la utilización de esquemas predefinidos para cada punto de acceso de la API. Cuando llega una petición a un punto de acceso que requiere datos provistos por el usuario, se verifica que el contenido tenga la estructura requerida para ese punto de acceso específico. La estructura hace referencia a los nombres de las claves sus tipos de datos esperados.

Se puede observar en la **Figura 25** que se tiene un modelo con los atributos y sus tipos de datos definidos para el DTO de una cuenta de invitado. Esto asegura que la estructura

básica se está respetando y las validaciones siguientes permitirán encontrar inconsistencias y datos incorrectos en la petición del usuario.

```python
class GuestDto(BaseModel):
    id: UUID
    host: UUID
    name: str
    image: str
```

**Figura 25.** Modelo de validación de datos para los DTOs de las cuentas de invitado

Una vez que la estructura de los datos ha sido validada, se procede a validar las reglas de negocio propias de la funcionalidad a la que llama el punto de acceso. Esto se logra a través de ValueObjects que no son más que objetos que contienen la lógica de validación para cada uno de los atributos de un modelo o entidad.

En la **Figura 26** se puede observar la lógica para la validación de las cadenas de texto que representan nombres de personas. Cuando se instancia la clase con un valor, este es validado para determinar que tenga entre 2 y 50 caracteres, que solo contenga caracteres alfabéticos y que inicie con mayúscula.

```python
class NameStr(str):

    def __new__(cls, value, *args, **kwargs):
        n_value = value.strip()
        cls._validate(n_value)
        return super().__new__(cls, n_value)

    @classmethod
    def _validate(cls, value: str) -> None:
        if len(value) < 2:
            raise ValidationError(f"Name must be at least 2 characters long: {len(value)}")
        if len(value) > 50:
            raise ValidationError(f"Name must be at most 50 characters long: {len(value)}")
        if not value.isalpha():
            raise ValidationError(f"Name must contain only alphabetic characters: {value}")
        if not value.istitle():
            raise ValidationError(f"Name must be title cased: {value}")
```

**Figura 26**. ValueObject para la validación de nombres de personas.

En el caso de que la petición conlleve una acción de escritura, la persistencia de datos también contiene un proceso de validación de datos para evitar que se lleguen a persistir datos erróneos que no corresponden a las reglas de una funcionalidad específica o que no tienen sentido bajo ningún criterio. Esta validación se da gracias a los modelos implementados para interactuar con el sistema de persistencia en la capa de infraestructura.

En la **Figura 27** se puede observar el modelo de datos implementado para validar los tipos de datos y la estructura que se guarda en la base de datos. Gracias a las librerías implementadas, se redujo la cantidad de código necesaria para realizar las validaciones del modelo debido a que estas se hacen de manera automática.

```python
class DBGuest(Document):
    id: UUID = Field(alias="_id")
    host: UUID
    name: str
    image: str
    scores: list[DBScore]
    achievements: list[UUID]

    class Settings:
        name = "guests"
```

**Figura 27.** Modelo de datos de MongoDB para la cuenta de invitado

### 2.3.8 Arquitectura del sistema

Se consideró una arquitectura para la ejecución del código, la persistencia de datos y el acceso a archivos de tal manera que los servicios de la API pudieran ser consumidos por diferentes usuarios a través de internet.

En la **Figura 28** se muestran los componentes del sistema y los servicios utilizados para su ejecución. El sistema está compuesto por los siguientes elementos:

- **KiddoAPI:** es la aplicación desarrollada con Flask, está dividida en módulos que representan los servicios que ofrece la API, así como las configuraciones e integraciones con servicios externos.
- **Docker:** es el contenedor utilizado para ejecutar la aplicación y gestionar una sola configuración y que se pueda levantar la aplicación independientemente del equipo en donde esté alojado el contenedor.
- **Railway:** es la cuenta del servicio de alojamiento de aplicaciones, contiene las instrucciones para publicar la aplicación bajo una dirección provista por Railway para que pueda ser utilizada por diferentes usuarios en internet.
- **Cloudflare R2:** instancia del almacenamiento de objetos provisto por Cloudflare, en este servicio se encuentran almacenadas las imágenes referentes al negocio que se ofrecen a los usuarios de la aplicación.

- **MongoDB Atlas:** cuenta del servicio cloud para el alojamiento y manejo de la base de datos MongoDB. Almacena todos los datos que la aplicación utiliza para proveer sus servicios al usuario y determina las direcciones que pueden acceder a la base de datos.

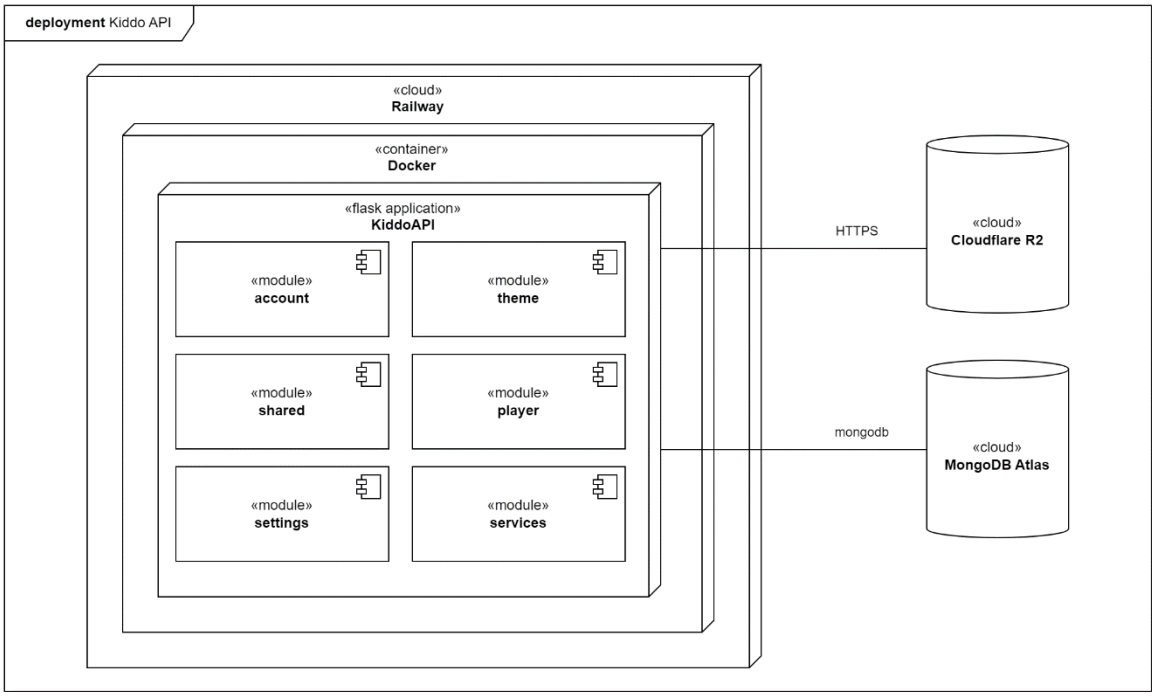



**Figura 28.** Diagrama de despliegue del sistema

### Pruebas unitarias

Para cada funcionalidad base se utilizó una clase de prueba TestCase que describe el caso base y los casos excepcionales. En la **Figura 29** se puede observar la prueba unitaria para la creación exitosa de un reporte para una cuenta interna guest utilizando simulaciones de las dependencias como la base de datos o la petición del usuario que contiene información para identificarlo.

```python
class TestCreateReport(unittest.TestCase):

    def test_create_report(self):
        # Given a guest and an app context
        mock = Mock()
        id = Id()
        mock = mock_app_context(mock, id, AccountType.USER)
        scores = [mock_score(80.0, 300.0, 5) for _ in range(6)]
        average_points = 80.0
        average_time = 300.0
        guest = mock_guest(host=id, scores=scores)
        mock.find_by_id.return_value = guest
        # When the report is created
        service = ScoreService(mock, mock, mock, mock)
        report = service.create_report(guest.id)
        # Then the report contains the average points and time
        self.assertEqual(report.avg_points, average_points)
        self.assertEqual(report.avg_time, average_time)
```

**Figura 29.** Prueba unitaria de la creación exitosa de un reporte

En la **Figura 30** se puede observar que se prueba un caso excepcional en el que se intenta crear un reporte para un invitado inexistente y se verifica que se emita una excepción con un mensaje específico. Para lograr que la clase *ScoreService* pueda ser instanciada, se le proveen de simulaciones de los servicios de los que depende y controlar las salidas de estos para modelar correctamente el escenario de prueba.

```python
def test_create_report_when_guest_not_found(self):
    # Given a not existing guest and an app context
    mock = Mock()
    id = Id()
    mock = mock_app_context(mock, id, AccountType.USER)
    mock.find_by_id.return_value = None
    # When the report is created
    service = ScoreService(mock, mock, mock, mock)
    # Then a NotFound exception is raised
    with self.assertRaisesRegex(exceptions.NotFound, "Guest not found"):
        service.create_report(Id())
```

**Figura 30**. Prueba unitaria para la creación de un reporte para un invitado inexistente

En la **Figura 31** se puede observar que la prueba unitaria está probando que, en el escenario en el que un usuario intenta crear un reporte para un invitado ajeno a su cuenta, se arroje una excepción que rompa el flujo para evitar que se cree el reporte y se envíe a un usuario equivocado.

```python
def test_create_report_with_different_host(self):
    # Given a guest, an app context and a different host
    mock = Mock()
    id = Id()
    mock = mock_app_context(mock, id, AccountType.USER)
    guest = mock_guest(host=Id())
    mock.find_by_id.return_value = guest
    # When the report is created
    service = ScoreService(mock, mock, mock, mock)
    # Then an Unauthorized exception is raised
    with self.assertRaises(exceptions.Unauthorized):
        service.create_report(guest.id)
```

**Figura 31.** Prueba unitaria para la creación de un reporte para un invitado cuyo anfitrión es diferente al usuario actual

De la misma manera se probó el resto de las funcionalidades base con la finalidad de determinar que se comportan como se espera según los diferentes escenarios conocidos (de éxito o de error) y localizar posibles errores introducidos al modificar o agregar funcionalidades a la aplicación.

## 2.4 Despliegue

### Publicación del servicio

Se configuró la integración con la plataforma Railway para el despliegue de la aplicación y que esté disponible para la aplicación móvil que mostrará el contenido de la API a través de juegos interactivos.

Para la construcción y despliegue del código creado se utilizó un contenedor creado a partir de un archivo de configuración de Docker. El despliegue se configuró para que se construya la aplicación cada vez que ocurre un commit en el repositorio de GitHub del proyecto como se muestra en la **Figura 32**.

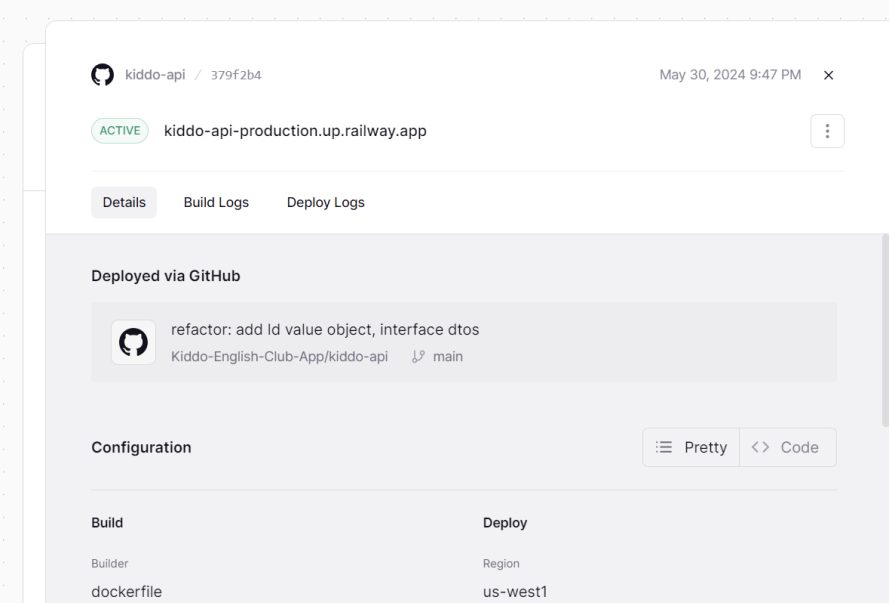



**Figura 32**. Pantalla de despliegue de Railway

En la **Figura 33** se puede observar el proceso de construcción de la aplicación a través de una tarea automatizada una vez que se detectan cambios en el repositorio remoto. Una vez la aplicación está construida, se realiza una verificación su estado para determinar si la aplicación puede ser accedida desde el exterior y si es que la aplicación está siendo ejecutada.

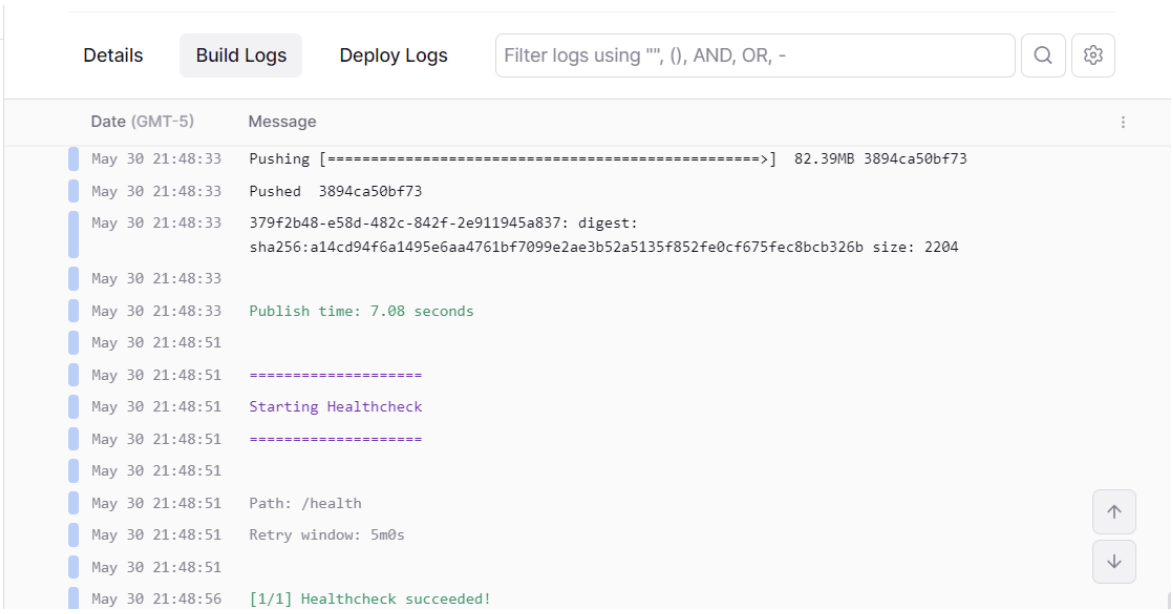



**Figura 33**. Logs de la construcción del contenedor de la aplicación

En la **Figura 34** se pueden observar las salidas de la aplicación en ejecución. Se detallan datos como la cantidad de procesos que se encuentran activos o la dirección interna a la cual se están redirigiendo las peticiones de los usuarios.

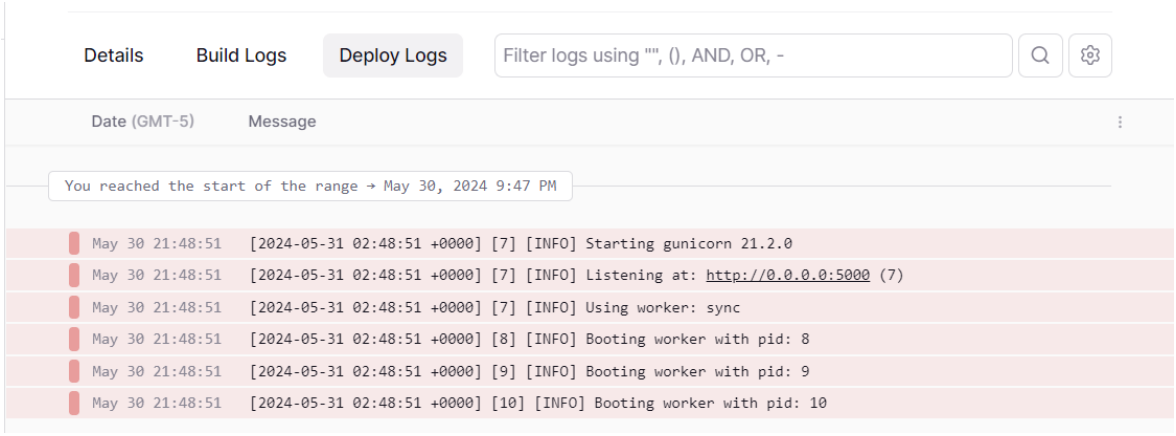



**Figura 34**. Logs del despliegue de la aplicación

### Pruebas de rendimiento

Se aplicaron pruebas de rendimiento para determinar el comportamiento del sistema en diferentes escenarios. Se utilizó el programa Apidog para la ejecución de estas pruebas debido a que provee de un apartado específico para la configuración y ejecución de pruebas automatizadas.

Para cada endpoint se creó una prueba automatizada que simula una carga válida de datos con el fin de determinar los tiempos de respuesta del sistema ante diferentes niveles de carga de parte de los usuarios.

Como se puede observar en la **Figura 35**, se replicó el ambiente de producción en la nube para configurar las integraciones requeridas en el momento de realizar pruebas y que todas las operaciones realizadas sobre los datos puedan ser desechadas sin comprometer los datos reales de los usuarios. Se creó una base de datos específica para los datos de prueba de tal manera que se puede reiniciar la base de datos y volver a cargar la información simulada para que se ajuste a la carga real del usuario.

Adicionalmente, se agregaron dos puntos de acceso que permiten realizar el redespliegue de la información a través de una petición HTTP al punto de acceso correspondiente.

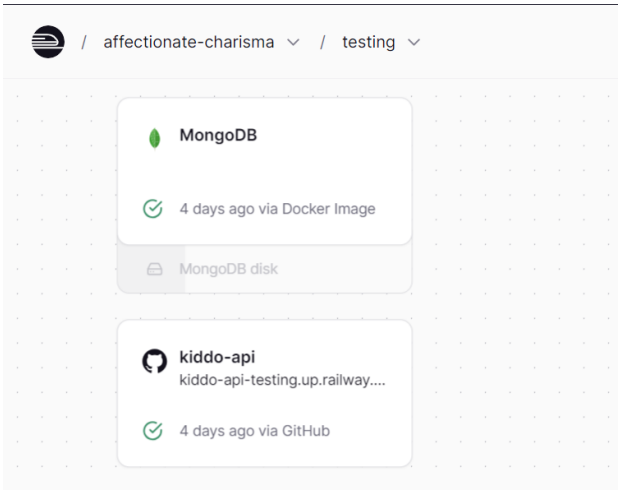



**Figura 35.** Ambiente de pruebas

En la **Figura 36** se muestra cómo se configuraron las pruebas para automatizar la ejecución de peticiones. Cada una de estas peticiones utilizará un conjunto de datos de prueba, para cada registro de este conjunto, se hará una petición que incluye los datos del registro actual. Con eso se logra simular datos muy cercanos a los datos que los usuarios enviarán o utilizarán para interactuar con la aplicación.

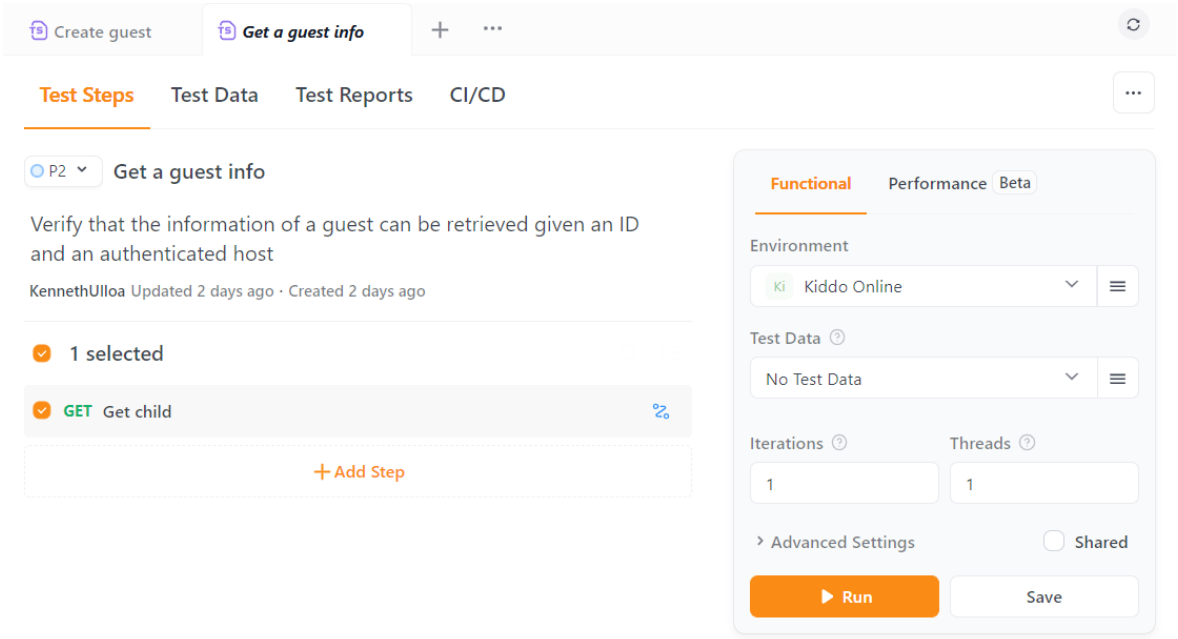



**Figura 36.** Configuración de una prueba automática en Apidog

En la **Figura 37** se muestra cómo se cargan los datos de prueba que van a ser utilizados en la configuración que se mostró previamente. Estos valores se incluirán en el contexto de la petición brindando flexibilidad y reduciendo la carga manual de la ejecución de pruebas. Se utilizó el formato JSON para cargar las pruebas como se muestra en la **Figura 38**, los datos de prueba se encuentran disponibles en el **ANEXO II**.

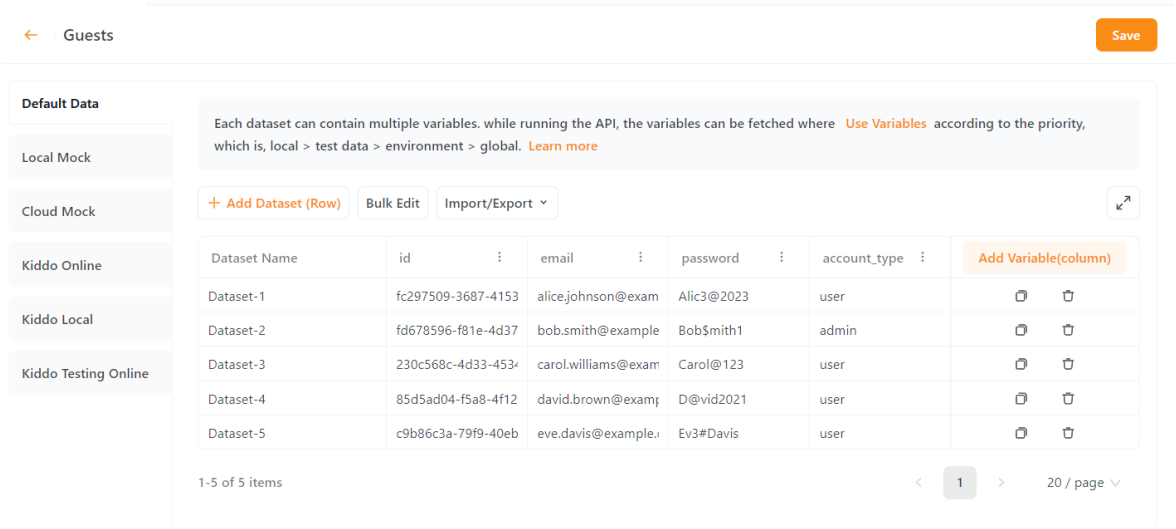



**Figura 37.** Carga de datos para la automatización de pruebas

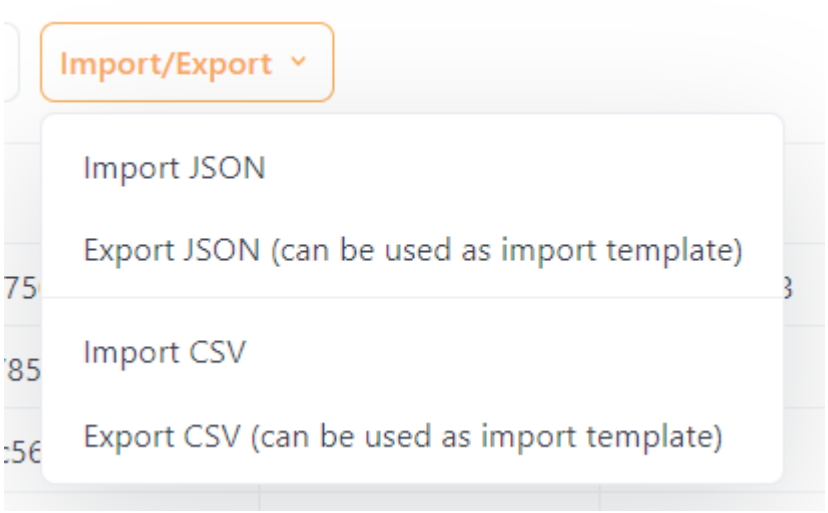



**Figura 38.** Ventana para la carga de datos para las pruebas

# 3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

## 3.1 Resultados

Como resultado del desarrollo de este componente se obtuvo una API funcional y desplegada que brinda las funcionalidades esperadas por el componente de interacción. La API expone los endpoints detallados en la **Tabla 3** y la documentación creada se encuentra en el **ANEXO III**.

**Tabla 3**. Lista de endpoints creados

| Endpoint | Método | Descripción |
|---|---|---|
| **Cuentas** | | |
| /api/accounts/register | GET | Registra una cuenta de usuario con su nombre, apellido, correo electrónico y contraseña. |
| /api/accounts/login | POST | Obtiene un token de autorización utilizando las credenciales asociadas a la cuenta. |
| /api/accounts/profile | GET | Obtiene los datos asociados a la cuenta de quien realiza la petición. |
| Puntuaciones | | |
| /api/scores | POST | Almacena el puntaje obtenido en un juego de un tema específico. |
| /api/scores/report/{child_id} | GET | Obtiene un reporte que contiene los temas que mejor puntuación, peor puntuación, el tiempo promedio de resolución y la puntuación promedio obtenida en todos los juegos. |
| **Temas** | | |
| /api/themes | GET | Obtiene una lista de todos los temas presentes en la aplicación. |
| /api/themes/{theme_id} | GET | Obtiene un tema específico con un ID. |
| **Niños** | | |
| /api/guests/{guest_id} | GET | Obtiene los datos asociados a la cuenta de un niño registrada en una cuenta de padre. |
| /api/guests | GET | Obtiene una lista de todos los niños registrados en una cuenta de padre. |
| /api/guests | POST | Crea una cuenta de niño en la cuenta de un padre. |
| /api/guests/{guest_id}/achievements | GET | Obtiene una lista de los logros de la cuenta de un niño. |
| /api/guests/{child_id} | DELETE | Elimina la cuenta de niño identificada con el ID provisto dentro de una cuenta de padre. |
| **Otros** | | |
| /services/files/{filename} | GET | Obtiene el contenido del archivo identificado por el nombre provisto. |

| | | |
|---|---|---|
| /api/validate/token | POST | Valida el token provisto en el cuerpo de la petición. Determina si la firma es correcta o si ya expiró. |
| /health | GET | Es una ruta de verificación del estado del servicio. |

Todos los endpoints fueron especificados y probados utilizando la herramienta Apidog. En la **Figura 39** se puede observar que para cada endpoint se pueden obtener detalles del funcionamiento, datos de entrada y de salida, así como los esquemas específicos para las respuestas. Esta es la documentación desarrollada para el consumo de la API.

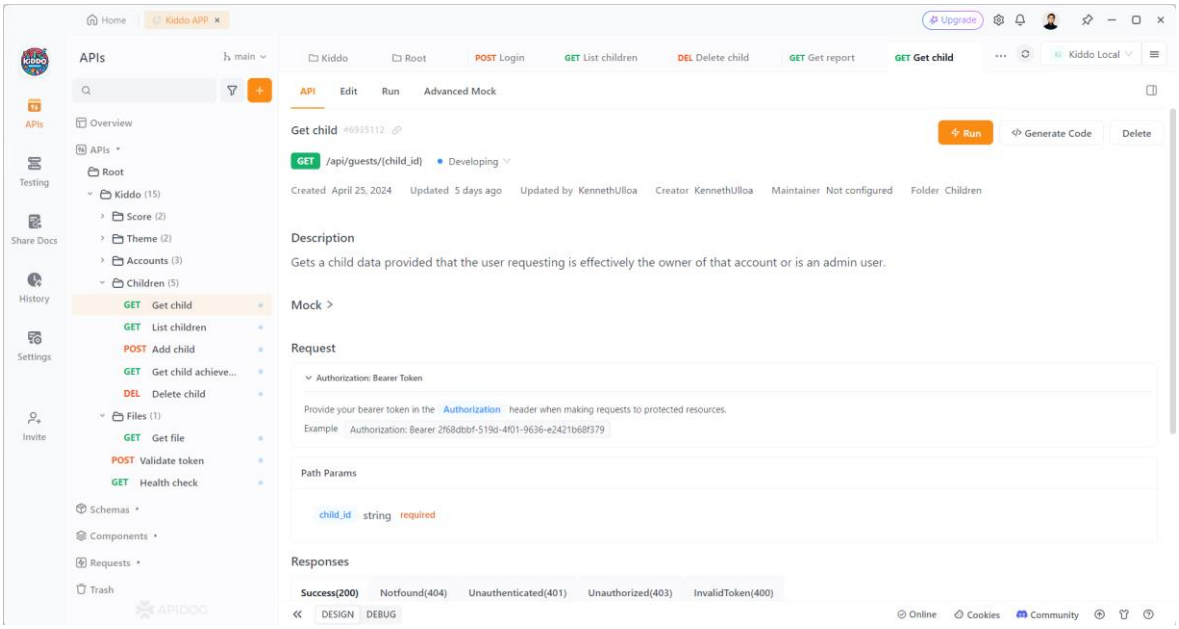



**Figura 39**. Pantalla de Apidog

En la **Figura 40** se puede observar el cómo se realizaron las peticiones de prueba a lo largo del desarrollo de la API. Esto permitió revisar y corregir las salidas para que se ajusten a las necesidades de información del principal cliente que es el componente de interacción del proyecto. Así mismo, permitió determinar el cumplimiento de los criterios de aceptación de las historias de usuario a través de la ejecución de las pruebas funcionales de los endpoints.

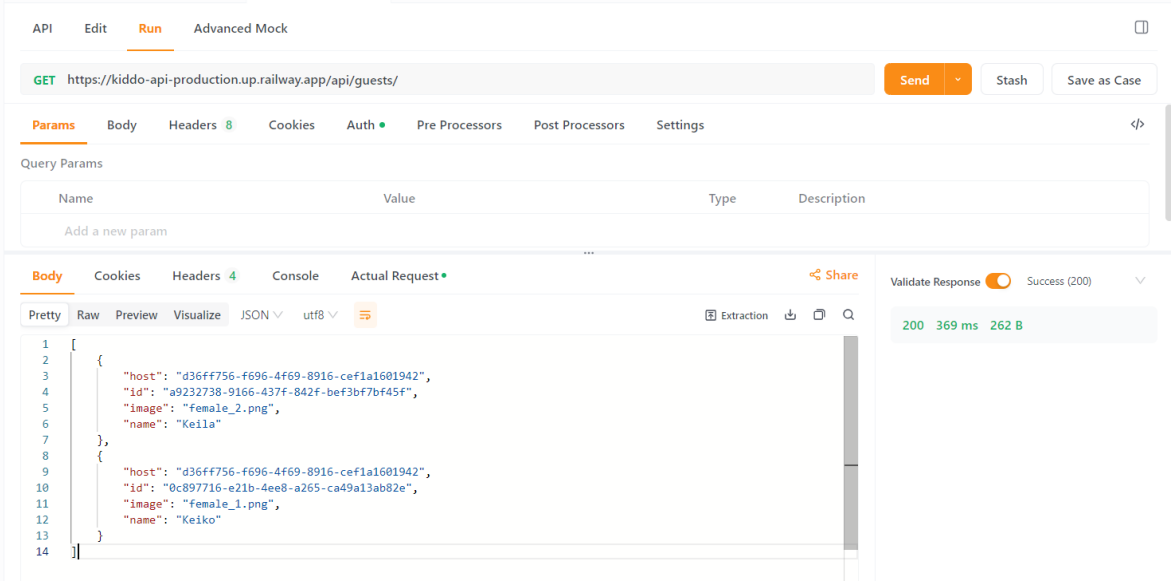
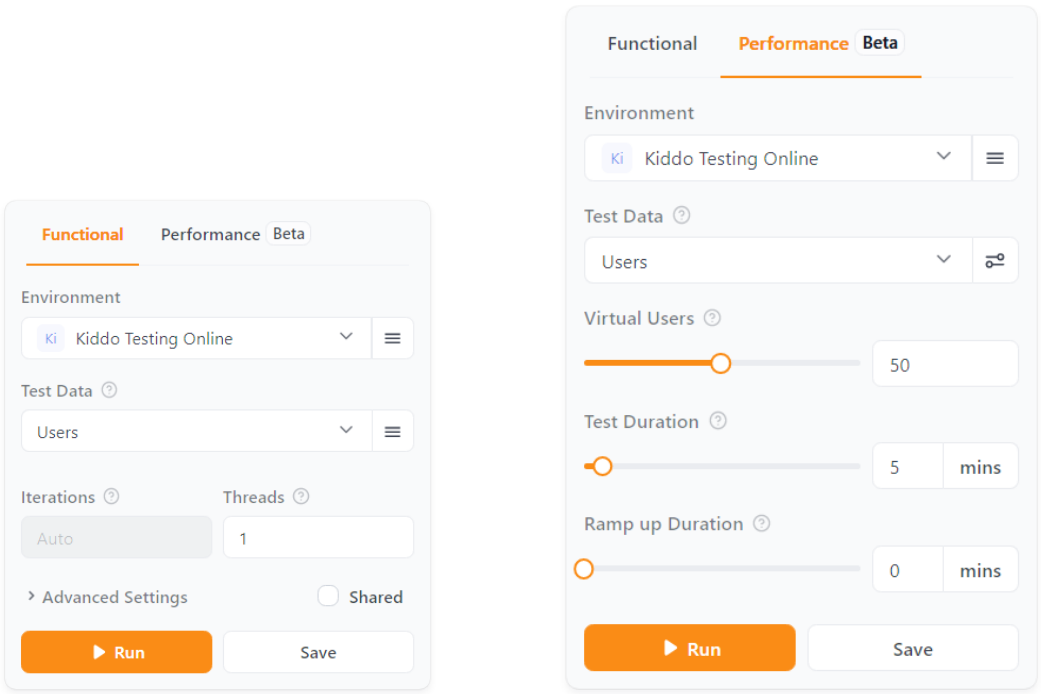



**Figura 40**. Petición en Apidog para obtener la lista de niños asociados a una cuenta

En la **Figura 41** se puede observar un ejemplo de la configuración de (a) las pruebas funcionales y (b) las pruebas de carga para cada endpoint en la herramienta Apidog. Esta característica facilitó la realización de pruebas dado que se puede utilizar directamente la especificación de la API para probar sus endpoints y evaluar los resultados obtenidos en contraste con los resultados esperados.




(a) (b)

**Figura 41**. Configuración de las pruebas en Apidog: a) pruebas funcionales y b) pruebas de rendimiento

Para el desarrollo de las pruebas funcionales se consideraron todos los endpoints. Considerando que existe una restricción de recursos disponibles en la capa gratuita de la base de datos MongoDB y el servicio en el que se encuentra alojada la API, se limitó el alcance de las pruebas a los endpoints relacionados con operaciones de lectura. Esto debido a que, al sobrecargar los servicios con operaciones de escritura, estos alcanzarán rápidamente su cuota máxima y negarán el acceso futuro, limitando así la capacidad de continuar con otro tipo de pruebas.

Para cada endpoint se planteó realizar pruebas de carga con 1, 10, 50 y 100 usuarios simultáneos.

En la **Tabla 4** se muestran resumidos los resultados de las pruebas de rendimiento realizadas a la API desarrollada. Se compara el tiempo de respuesta promedio en milisegundos con la cantidad de usuarios simultáneos que envían peticiones a un endpoint específico. Los detalles de las pruebas y más datos se encuentran presentes en el **ANEXO II**.

**Tabla 4.** Resultados de las pruebas de rendimiento

| # Prueba | Endpoint | Tiempo de respuesta promedio (ms) X número de usuarios simultáneos | | | |
|---|---|---|---|---|---|
| | | **1** | **10** | **50** | **100** |
| 1 | /services/files/{filename} | 458 | 557 | 2 586 | 6 041 |
| 2 | /api/accounts/login | 505 | 516 | 3 147 | 7 976 |
| 3 | /api/accounts/profile | 226 | 226 | 252 | 555 |
| 4 | /api/scores/report/{child_id} | 450 | 449 | 2 257 | 5 650 |
| 5 | /api/themes | 247 | 230 | 242 | 550 |
| 6 | /api/themes/{theme_id} | 249 | 245 | 341 | 543 |
| 7 | /api/guests | 322 | 324 | 940 | 3 044 |
| 8 | /api/guests/{guest_id} | 304 | 292 | 604 | 2 276 |
| 9 | /api/guests/{guest_id}/achievements | 1 198 | 1 533 | 14 227 | 28 084 |

En la **Figura 42** se puede observar que en total se implementaron 56 pruebas unitarias para las funcionalidades principales de la API utilizando abstracciones de los servicios externos necesarios como el acceso a la base de datos, estas pruebas cubren únicamente las capas de dominio y aplicación, la capa de infraestructura se omitió dado que requiere de pruebas de integración y eso no se encuentra dentro del alcance de las pruebas planteadas.

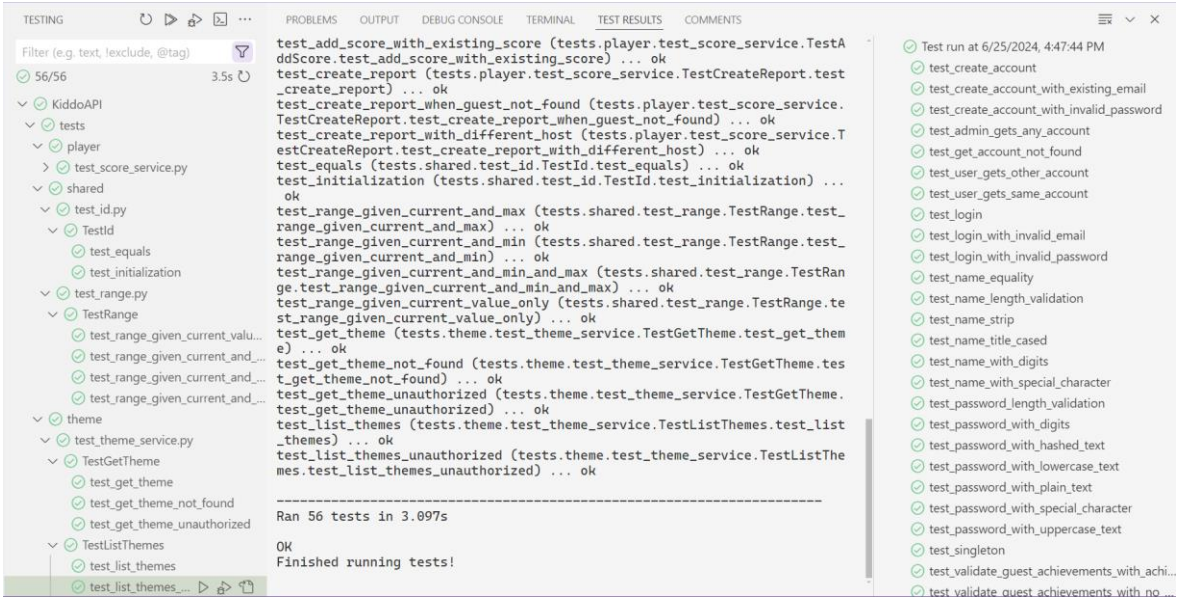



**Figura 42**. Resultados de las pruebas unitarias

## 3.2 Conclusiones

Durante el planteamiento del problema se encontró que previamente ya existían trabajos que abordaban el problema del aprendizaje del idioma inglés a través de juegos educativos. Sin embargo, desde un punto de vista técnico, estos juegos estaban limitados a un vocabulario fijo en cada uno de los trabajos desarrollados en [5], [6] y [7]. Esto implica que, para aumentar y/o cambiar el vocabulario usando para el refuerzo, es necesario modificar la aplicación en su totalidad y volver a distribuirla a los usuarios cada vez. Este componente soluciona el problema descrito al proveer de una fuente de recursos y datos que puede crecer independientemente del cliente que la usa, sea móvil, web o escritorio.

Este componente provee de una forma transparente y desacoplada de la tecnología específica del lado del cliente. Por ejemplo, si alguno de los juegos referenciados quiere mejorar y conectarse a una base de datos o utilizar un servicio de terceros como Google, se requiere que se acoplen a una tecnología en específico provocando que se tenga que volver a distribuir la aplicación en el caso de un cambio.

Este componente no depende de la tecnología utilizada para implementarlo. En un futuro, si se decide continuar con la idea planteada en este documento, se puede cambiar la tecnología utilizada para la implementación (por ejemplo, para mejorar el rendimiento) y el cliente no se vería en la necesidad de hacer cambios o de redistribuir una nueva versión de la aplicación.

El objetivo general de este componente es que las aplicaciones orientadas a los usuarios finales tengan que preocuparse lo menos posible respecto a la materia prima (los temas y recursos) y enfocarse en ofrecer valor a los usuarios a través de interfaces atractivas, juegos interesantes u otras estrategias para lograr el refuerzo educativo.

Respecto al cumplimiento de los objetivos específicos del proyecto:

- Se ha cumplido el objetivo **"Realizar una investigación que permita conocer las necesidades de los niños de 6 a 8 años respecto al aprendizaje del idioma inglés para determinar las funcionalidades que debería exponer el servicio web"** dado que, tanto en el planteamiento del problema y en la etapa de requerimientos del ciclo de desarrollo con RAD, se pudieron investigar las necesidades de los niños respecto al aprendizaje de inglés y estas pudieron ser usadas para determinar las funcionalidades que la API debía exponer para facilitar el desarrollo de juegos enfocados en el aprendizaje.

- Se ha cumplido con el objetivo **"Implementar el servicio web teniendo en cuenta los principios SOLID para facilitar el mantenimiento, la escalabilidad y la modificabilidad del código"** dado que, durante el desarrollo de las funcionalidades de la API, se implementó la arquitectura limpia. Esta arquitectura aplica los principios SOLID en la regla de dependencias, el uso de interfaces para evitar el acoplamiento y la separación de responsabilidades al determinar una estructura de capas bien definidas.

- Se ha cumplido con el objetivo **"Realizar pruebas unitarias y de rendimiento para validar que se cumple con la adecuación funcional y la eficiencia"** dado que, durante la etapa de desarrollo y despliegue, se crearon y ejecutaron pruebas unitarias para cada una de las funcionalidades teniendo en cuenta los casos de éxito y los casos de error, estos resultados se encuentran en la sección de resultados de este documento. Así mismo, se realizaron pruebas de rendimiento a la API a través de simular diferentes cargas de trabajo y medir los tiempos de respuesta, estos resultados se encuentran en el **ANEXO II**.

El proyecto ha cumplido con el alcance en tanto se desarrolló con la metodología RAD, el servicio se encuentra desplegado y se ha desarrollado la documentación de la API.

## 3.3 Recomendaciones

Para el desarrollo de proyectos de la misma naturaleza (desarrollo de APIs) se ponen a disposición las siguientes recomendaciones:

- Utilizar sistemas, programas o aplicaciones que permitan desarrollar documentación de manera sencilla. A lo largo de este proyecto, el uso de Apidog facilitó el proceso de creación de documentación y desarrollo de la API. Sin embargo, existen otras aplicaciones con características similares en el mercado que pueden cubrir diferentes necesidades para todo tipo de proyectos de desarrollo.

- Definir claramente las necesidades de los clientes de la API teniendo en cuenta las convenciones respecto al formato y estructura de los datos que estos necesitan para aprovechar las funcionalidades que se van a exponer. Dado que una API es un contrato entre dos sistemas, es importante que las reglas queden claras para que ambos puedan ser desarrollados con la menor cantidad de interrupciones provocadas por malentendidos respecto a los datos que se van a intercambiar entre ellos.

- Realizar una investigación comparativa antes de elegir una tecnología de implementación. Teniendo en cuenta que el desarrollo de un proyecto se puede ver afectado por las facilidades y carencias de la tecnología subyacente, es importante escoger aquella que se adecúa a sus necesidades y permite otorgar valor de manera sencilla.

- Escoger la arquitectura de software adecuada para el problema que se intenta resolver. Es importante investigar las ventajas y desventajas de la arquitectura que se planea implementar porque se pueden estar causando problemas a futuro en el producto desarrollado evitando que este evolucione de manera correcta o, en el peor de los casos, teniendo que construirlo desde cero. No todos los patrones de arquitectura se pueden aplicar a todos los proyectos, en muchos casos, se cae en un problema de escalabilidad por una arquitectura muy simple o en sobre-ingeniería al aplicar una arquitectura más compleja de lo necesario.

- Entender las características del producto a desarrollar y la metodología que permite desarrollarlo de manera sencilla. Al igual que con la arquitectura, la metodología utilizada para desarrollar un producto de software tiene un gran impacto en su construcción. Es imperativo evitar utilizar una metodología solo por su popularidad y su tendencia de uso; no todos los proyectos requieren ser ágiles, así mismo, no todos los proyectos requieren una planificación muy detallada.

# 4 REFERENCIAS BIBLIOGRÁFICAS


[1] "Lengua Extranjera - Ministerio de Educación." Accessed: Apr. 23, 2024. [Online]. Available: https://educacion.gob.ec/curriculo-lengua-extranjera/

[2] "EF EPI Índice EF de nivel de inglés." [Online]. Available: www.efset.orgwww.ef.com/epi

[3] I. María *et al.*, "Los problemas de aprendizaje en la enseñanza del idioma inglés", [Online]. Available: https://www.eumed.net/rev/atlante/2019/08/problemas-aprendizaje-ingles.html

[4] S. Cacuango, "Herramientas interactivas en el proceso de enseñanza-aprendizaje del idioma inglés" UNIVERSIDAD TECNOLÓGICA INDOAMÉRICA, Quito, 2021.

[5] B. K. Ng, N. M. Suaib, A. J. Sihes, A. Ali, and Z. A. Shah, "Educational mobile game for learning English words," in *IOP Conference Series: Materials Science and Engineering*, IOP Publishing Ltd, Nov. 2020. doi: 10.1088/1757-899X/979/1/012007.

[6] M. M. Elaish, N. A. Ghani, L. Shuib, and A. Al-Haiqi, "Development of a Mobile Game Application to Boost Students' Motivation in Learning English Vocabulary," *IEEE Access*, vol. 7, pp. 13326–13337, 2019, doi: 10.1109/ACCESS.2019.2891504.

[7] K. Ishaq, F. Rosdi, N. A. M. Zin, and A. Abid, "Serious game design model for language learning in the cultural context," *Educ Inf Technol (Dordr)*, vol. 27, no. 7, pp. 9317–9355, Aug. 2022, doi: 10.1007/s10639-022-10999-5.

[8] B. De, *API Management*. Apress, 2017. doi: 10.1007/978-1-4842-1305-6.

[9] J. Higginbotham, "Creating Business Value Through Developer Experience Designing Great Web APIs," 2015.

[10] H. Subramanian and P. Raj, "Hands-On RESTful API Design Patterns and Best Practices Design, develop, and deploy highly adaptable, scalable, and secure RESTful web APIs," 2019.

[11] K. Lane, "The API-First Transformation Abhinav Asthana," 2022.

[12] B. Doerrfeld, B. Pedro, K. Sandoval, and A. Krohn, "The API Lifecycle An Agile Process For Managing the Life of an API Nordic APIs," 2015.

[13] E. Sandler, "Rapid Application Development (RAD)," Code Academy. Accessed: Apr. 28, 2024. [Online]. Available: https://www.codecademy.com/resources/docs/general/software-development-life-cycle/rapid-application-development

[14] A. Stec, "Rapid Application Development," Baeldung. Accessed: Apr. 28, 2024. [Online]. Available: https://www.baeldung.com/cs/rad

[15] "What is Rapid Application Development (RAD)? An Ultimate Guide for 2024," Kissflow.

[16] R. C. Martin, "Clean Architechture," 2016.

[17] "Python 3.12 Documentation." Accessed: May 18, 2024. [Online]. Available: https://docs.python.org/3/

[18] “Design Decisions in Flask,” Palletdocs. Accessed: Apr. 28, 2024. [Online]. Available: https://flask.palletsprojects.com/en/3.0.x/design/

[19] “Features | GitHub,” GitHub. Accessed: Apr. 28, 2024. [Online]. Available: https://github.com/features/

[20] “Introduction to MongoDB,” MongoDB. Accessed: May 18, 2024. [Online]. Available: https://www.mongodb.com/docs/manual/introduction/

[21] “Apidog An integrated platform for API design, debugging, development, mock, and testing,” Apidog.

# 5 ANEXOS

**ANEXO I.** Tabla comparativa de la metodología RAD con XP

| **Características** | **Metodología ágil (XP)** | **RAD** |
|---|---|---|
| Definición | Método de manejo de proyecto, equipos autoorganizados. | Metodología de desarrollo de software, basado en prototipos. |
| Adaptabilidad | Capaz de aceptar el cambio al principio de la reunión del reléase. | El cambio es posible, podría significar caer en retrabajo |
| Enfoque de codificación | TDD. | Generación de prototipos |
| Fase de pruebas | Pruebas unitarias, de integración y de sistema. | Pruebas unitarias, de integración y de sistema. |
| Largo de una iteración | 2 semanas | 2-3 semanas |
| Conocimiento requerido | Producto y dominio | Dominio |
| Estatus del equipo de desarrollo | Programación por pares | Programación individual |
| Comunicación con el cliente | Comunicación directa entre los desarrolladores y el cliente. | Mediación del administrador del proyecto, el cliente evalúa los modelos y solicita cambios. |

**ANEXO II**. Enlace a la carpeta con los anexos de las pruebas y documentación

https://epnecuador-my.sharepoint.com/:f:/g/personal/kenneth_ulloa_epn_edu_ec/EsLhiHYu5qtGqhsJ4mXhk9sB_CeMxzK1QPKDOVpP-G-Irg?e=mx0aO9

**ANEXO III.** Enlace de la documentación de la API en Apidog

https://1bwn5ql42d.apidog.io.

**Contraseña:** awRbBnj8