



INSTITUTO TECNOLÓGICO DE CD. GUZMÁN

TITULACIÓN INTEGRAL

INFORME TÉCNICO DE RESIDENCIA PROFESIONAL

TEMA:

**MINI JUEGOS QUE APOYEN EN LA EDUCACIÓN
NUTRIMENTAL DE NIÑOS (PARTE 1)**

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN SISTEMAS COMPUTACIONALES

PRESENTA:

CARLOS ALBERTO GARCÍA DE ALBA CHÁVEZ

ASESOR(A):

DRA. ROSA MARÍA MICHEL NAVA

CD. GUZMÁN JALISCO, MÉXICO, AGOSTO DE 2020

I. Índice General

II. Índice de Figuras	iii
III. Índice de Tablas	v
1. Preliminares.....	6
2. Generalidades del proyecto	8
3. Marco teórico	11
4. Desarrollo	28
5. Resultados	52
6. Conclusiones	57
7. Competencias desarrolladas	58
8. Fuentes de información	61

II. Índice de Figuras

Figura 1 Proceso de Scrum	12
Figura 2 Comparación entre arquitectura monolítica y microservicios	25
Figura 3 Logotipo de Spring Boot.....	26
Figura 4 Logotipo de Unity3D	26
Figura 5 Logotipo de IntelliJ IDEA.....	27
Figura 6 Logotipo de DigitalOcean	27
Figura 7 Diagrama de Despliegue del proyecto	30
Figura 8 Diagrama ER del proyecto	31
Figura 9 Clase Entidad de Food	38
Figura 10 Interfaz repositorio de Food	39
Figura 11 Clase de servicio de Food	40
Figura 12 Clase controlador de Food	41
Figura 13 Generación del ejecutable con Gradle.....	42
Figura 14 Localización del ejecutable generado	43
Figura 15 Pantalla del Droplet de producción en DigitalOcean	43
Figura 16 Conexión al Droplet por SSH	44
Figura 17 Deployment de la aplicación por SFTP	44
Figura 18 Directorio de la configuración de los servicios en GNU/Linux.....	45
Figura 19 Archivo de configuración del servicio	45
Figura 20 Consulta de estatus del servicio	45
Figura 21 Modelo Food de la interfaz de Unity	46
Figura 22 Wrapper del modelo Food.....	47
Figura 23 Estructura de la clase EndpointBuilder	48
Figura 24 Segunda parte de la estructura de la clase EndpointBuilder	49
Figura 25 Estructura de la clase APIConnection y el método ExecuteQueryAtEndpoint	50
Figura 26 Método FormatJsonForDeserializingList.....	50
Figura 27 Método GetAllFood	51

Figura 28 Consulta desde navegador a la API.....	52
Figura 29 Diagrama de cascada con los archivos recibidos por HTTP desde el servidor.....	52
Figura 30 Visor del JSON recibido desde el servidor	53
Figura 31 JSON recibido desde el servidor en Postman	53
Figura 32 JSON recibido desde el servidor en Postman, utilizando otro endpoint	54
Figura 33 Visor de jerarquía del proyecto de prueba	54
Figura 34 Inspector del objeto API Client de prueba	55
Figura 35 Estructura del archivo TestScript	55
Figura 36 Salida de la ejecución de TestScript.....	56

III. Índice de Tablas

Tabla 1 Diccionario de Datos del proyecto	34
Tabla 2 Tabla de Endpoints de la API.....	37

1. Preliminares

Agradecimientos

Me gustaría agradecerles a todas aquellas personas que me apoyaron durante el desarrollo de este proyecto. Entre ellos, a mi asesora, que siempre estuvo atenta a nuestros avances y nos otorgó las facilidades para poder desarrollar el proyecto bajo nuestros lineamientos y haciendo uso de nuestro conocimiento del tema.

A mis profesores, que me acompañaron con sus enseñanzas y sus consejos para poder lograr todos los objetivos planteados en este documento. A mis padres que siempre fueron un pilar fundamental en mi educación. A mis compañeros de proyecto, íntimos amigos míos y con los que trabajé de forma muy amena y tranquila. Finalmente, agradezco a la casualidad, que me dio la oportunidad de haber participado en este gran proyecto. Gracias y un abrazo fraternal a todos.

Resumen

Nutritional Games es un proyecto para apoyar la tesis de una alumna de la Maestría en Ciencias de la Computación. Consta de una plataforma que permitirá que los niños aprendan conceptos básicos de nutrición y mejoren su alimentación mientras se divierten con minijuegos en su computadora personal o dispositivo móvil.

El proyecto es de gran importancia, dado que los índices de sobrepeso y obesidad infantil en México están muy elevados, con una prevalencia del 34.4%, una cifra muy alarmante que

tiene consecuencias directas a la salud de los niños de entre 5 y 11 años, los cuales forman parte de nuestro público objetivo.

Para desarrollar este proyecto, se utilizó tecnología de vanguardia, como lo es el motor de videojuegos llamado Unity, así como el framework para desarrollo de aplicaciones Web conocido como Spring Boot. Esta pila de tecnología permite desarrollar una plataforma robusta, escalable y segura, gracias a la tecnología de deployment soportada por la empresa DigitalOcean.

El principal resultado del desarrollo de este proyecto fue una base de datos, la cual sirve como punto de entrada y salida de información hacia una aplicación cliente desarrollada en Unity, la cual se encarga de proporcionar 2 minijuegos al niño y analizar su progreso, para posteriormente enviar la información de vuelta a la base de datos a través de la API (Application Programming Interface/Interfaz para Programación de Aplicaciones) desarrollada en Spring.

2. Generalidades del proyecto

Introducción

Como se dijo anteriormente, el proyecto consta de diseñar una plataforma educativa que permita que los niños aprendan conceptos básicos de nutrición de forma completamente lúdica, a través de una aplicación que contiene una serie de videojuegos.

La estructura de este documento divide la información en varios capítulos, que se describirán a continuación.

En la sección de Marco Teórico, se abordará toda la documentación que otorga sustento a nuestro proyecto, además de sentar las bases de desarrollo. El capítulo de Desarrollo contiene toda la información acerca de qué fue lo que se hizo dentro del proyecto y cuáles fueron las contribuciones del autor. En el capítulo de resultados, se describen cuáles fueron los resultados del proyecto, los cuales se relacionan a la fase de pruebas dentro de la metodología. Finalmente, la sección de Conclusiones presenta un texto breve con las conclusiones personales del autor acerca del proyecto, las dificultades presentadas y la forma de resolverlas.

Adicionalmente, se presentan dos secciones más, que conforman las competencias desarrolladas a lo largo del proyecto y una lista de las referencias bibliográficas que dan sustento a la información contenida dentro del presente.

Descripción de la empresa u organización y del puesto o área del trabajo del estudiante

El Instituto Tecnológico de Ciudad Guzmán provee educación superior a la región Sur de Jalisco desde 1972, formando profesionales en las áreas de Ingenierías y Ciencias Económico-Administrativas. Nutritional Games es un proyecto en la Academia de Sistemas Computacionales del Instituto. El puesto dentro del proyecto se puede definir como Software Architect, Game Developer, Game Designer y Project Manager. Las principales responsabilidades del puesto son: diseñar la arquitectura que va a sustentar la aplicación, así como desarrollar la plataforma de videojuegos y la administración propia del proyecto. Esto, incluye el desarrollo de una API que funcione como interfaz entre la base de datos y las aplicaciones cliente.

Objetivos

El objetivo general del proyecto es:

Desarrollar dos minijuegos (memorama y el juego de las sillitas) que apoyen en la educación nutrimental de niños con sobrepeso u obesidad, identificando qué y cuánto seleccionan para alimentarse.

Con base en lo anterior, se plantean los siguientes objetivos:

- Analizar y diseñar cada minijuego para que contemple la revisión del tipo y la cantidad de alimentos que seleccionan como parte de su dieta, los niños que presentan sobrepeso u obesidad.
- Desarrollar los minijuegos, de tal manera que permitan almacenar el seguimiento del niño en cada sesión.
- Poner a prueba cada minijuego, revisando que cumpla con los requerimientos establecidos y con las características de diseño centrado en el usuario.

- Verificar que se almacenan correctamente los datos del tipo y cantidad de alimentos que niños con sobrepeso u obesidad seleccionan una vez que utilizan los minijuegos.
- Documentar los resultados obtenidos de los minijuegos, redactando también el informe final de la residencia profesional.

Justificación

Según la Encuesta Nacional de Salud y Nutrición (ENSANUT, 2012) en México los niños en edad escolar (ambos sexos), de 5 a 11 años, presentaron una prevalencia nacional combinada de sobrepeso y obesidad de 34.4%, 19.8% para sobrepeso y 14.6% para obesidad. En Jalisco, las prevalencias de sobrepeso y obesidad fueron 23.9 y 15.7%, respectivamente (suma de sobrepeso y obesidad, 39.6%). La prevalencia de sobrepeso en localidades urbanas aumentó de 2006 a 2012 de 20.0 a 24.1% y para las rurales pasó de 31.0 a 23.1%, respectivamente.

Ante el problema planteado, que implica cambios de comportamientos y establecer hábitos alimenticios saludables, es necesario mejorar la educación nutrimental desde la infancia para evitar la prevalencia de sobrepeso u obesidad en la edad adulta.

Por consiguiente, se propone mediante este proyecto, que, a través de minijuegos, los niños aprendan a seleccionar qué y cuántos alimentos deben consumir, para que puedan tomar decisiones saludables dentro del entorno donde se desenvuelven.

3. Marco teórico

En esta sección del documento se incluirán todos aquellos procedimientos y metodologías necesarias que brinden soporte al análisis de los requerimientos del proyecto y que sirven como guía para desarrollar cada uno de los módulos que conforman al sistema. Se presentarán respetando el orden de la metodología utilizada dentro del proyecto, comenzando con ésta.

Metodología Scrum

Scrum es un framework en el que los miembros de un equipo de desarrollo pueden enfrentarse a desarrollos muy complejos de manera rápida y productiva, produciendo resultados de muy alta calidad. Como tal, Scrum puede ser aplicado a cualquier equipo, sin embargo, es especialmente utilizado en desarrollo de software.

Scrum es un método heurístico en donde equipos pequeños trabajan en conjunto durante divisiones de tiempo conocidas como Sprints. En cada sprint, se definen roles y metas muy específicas, que al término serán evaluadas por el equipo y se realizarán los ajustes correspondientes. Scrum forma parte de las metodologías de software conocidas como Ágiles o Agile Development.

En Scrum un proyecto se ejecuta en ciclos temporales cortos y de duración fija (iteraciones que normalmente son de 2 semanas, aunque en algunos equipos son de 3 y hasta 4 semanas, límite máximo de feedback de producto real y reflexión). Cada iteración tiene que proporcionar un resultado completo, un incremento de producto final que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando lo solicite.

En la Figura 1, se puede observar el ciclo de vida de los proyectos que utilizan Scrum.



Figura 1 Proceso de Scrum

El proceso parte de la lista de objetivos/requisitos priorizada del producto, que actúa como plan del proyecto. En esta lista el cliente (Product Owner) prioriza los objetivos balanceando el valor que le aportan respecto a su coste (que el equipo estima considerando la Definición de Hecho) y quedan repartidos en iteraciones y entregas.

Las actividades que se llevan a cabo en Scrum son las siguientes (los tiempos indicados son para iteraciones de 2 semanas):

Planificación de la iteración

El primer día de la iteración se realiza la reunión de planificación de la iteración. Tiene dos partes:

Selección de requisitos (2 horas). El cliente presenta al equipo la lista de requisitos priorizada del producto o proyecto. El equipo pregunta al cliente las dudas que surgen y selecciona los requisitos más prioritarios que prevé que podrá completar en la iteración, de manera que puedan ser entregados si el cliente lo solicita.

Planificación de la iteración (2 horas). El equipo elabora la lista de tareas de la iteración necesarias para desarrollar los requisitos seleccionados. La estimación de esfuerzo se hace de manera conjunta y los miembros del equipo se autoasignan las tareas, se autoorganizan para trabajar incluso en parejas (o grupos mayores) con el fin de compartir conocimiento (creando un equipo más resiliente) o para resolver juntos objetivos especialmente complejos.

Ejecución de la iteración

Cada día el equipo realiza una reunión de sincronización (15 minutos), normalmente delante de un tablero físico o pizarra (Scrum Taskboard). El equipo inspecciona el trabajo que el resto está realizando (dependencias entre tareas, progreso hacia el objetivo de la iteración, obstáculos que pueden impedir este objetivo) para poder hacer las adaptaciones necesarias que permitan cumplir con la previsión de objetivos a mostrar al final de la iteración. En la reunión cada miembro del equipo responde a tres preguntas:

- ¿Qué he hecho desde la última reunión de sincronización para ayudar al equipo a cumplir su objetivo?
- ¿Qué voy a hacer a partir de este momento para ayudar al equipo a cumplir su objetivo?
- ¿Qué impedimentos tengo o voy a tener que nos impidan conseguir nuestro objetivo?

Durante la iteración el Facilitador (Scrum Master) se encarga de que el equipo pueda mantener el foco para cumplir con sus objetivos.

- Elimina los obstáculos que el equipo no puede resolver por sí mismo.
- Protege al equipo de interrupciones externas que puedan afectar el objetivo de la iteración o su productividad.

Durante la iteración, el cliente junto con el equipo refina la lista de requisitos (para prepararlos para las siguientes iteraciones) y, si es necesario, cambian o replanifican los objetivos del proyecto (10%-15% del tiempo de la iteración) con el objetivo de maximizar la utilidad de lo que se desarrolla y el retorno de inversión.

Inspección y adaptación

El último día de la iteración se realiza la reunión de revisión de la iteración. Tiene dos partes:

- Revisión (demostración) (1,5 horas). El equipo presenta al cliente los requisitos completados en la iteración, en forma de incremento de producto preparado para ser entregado con el mínimo esfuerzo. En función de los resultados mostrados y de los cambios que haya habido en el contexto del proyecto, el cliente realiza las adaptaciones necesarias de manera objetiva, ya desde la primera iteración, replanificando el proyecto.
- Retrospectiva (1,5 horas). El equipo analiza cómo ha sido su manera de trabajar y cuáles son los problemas que podrían impedirle progresar adecuadamente, mejorando de manera continua su productividad. El Facilitador se encargará de eliminar o escalar los obstáculos identificados que estén más allá del ámbito de acción del equipo.

En Scrum, el equipo se focaliza en construir software de calidad. La gestión de un proyecto Scrum se centra en definir cuáles son las características que debe tener el producto a construir

(qué construir, qué no y en qué orden) y en vencer cualquier obstáculo que pudiera entorpecer la tarea del equipo de desarrollo.

El equipo Scrum está formado por los siguientes roles:

- **Scrum Master:** Persona que lidera al equipo guiándolo para que cumpla las reglas y procesos de la metodología. Gestiona la reducción de impedimentos del proyecto y trabaja con el Product Owner para maximizar el ROI (Return of Investment/Retorno de la Inversión).
- **Product Owner (PO):** Representante de los accionistas y clientes que usan el software. Se focaliza en la parte de negocio y él es responsable del ROI del proyecto (entregar un valor superior al dinero invertido). Traslada la visión del proyecto al equipo, formaliza las prestaciones en historias a incorporar en el Product Backlog y las prioriza de forma regular.
- **Team:** Grupo de profesionales con los conocimientos técnicos necesarios y que desarrollan el proyecto de manera conjunta llevando a cabo las historias a las que se comprometen al inicio de cada sprint.

Fases del ciclo de vida de desarrollo de software

A continuación, se expone la teoría que sustenta cada una de las fases del ciclo de vida del proyecto.

Fase de análisis

En la fase de análisis se definieron las premisas básicas, por lo que la teoría de diseño de videojuegos sustenta el proceso por el cual se diseñó la plataforma.

Definición y premisas de diseño de videojuegos

Las fundaciones de diseño del videojuego son las siguientes:

Fundación

Los títulos siempre deben de comenzar con el jugador en su estado más vulnerable. Una situación clásica es que el jugador comience con mejoras parciales o completas de sus habilidades, y después de un evento, se le arrebaten.

Uno de los principales errores al diseñar con enfoque al progreso, es hacer el inicio del juego muy tedioso y frustrante para mostrar la evolución del personaje en escenarios posteriores, esto puede llegar a incluir el arrebato de habilidades como correr, saltar e incluso atacar.

Es errado querer que los jugadores tengan inmediatamente una experiencia frustrante al empezar el juego. Si se va a comenzar sin habilidades esenciales, se requieren incluir de manera muy rápida, para que el jugador pueda comenzar a experimentar las mecánicas básicas. El núcleo de tu título debe de ser sólido desde el primer minuto.

Los títulos deben comenzar controlando responsivamente al personaje y mejorar a partir de ahí.

Diseño orientado al progreso

Una de las mecánicas más importantes en los títulos de este subgénero, son las que permiten la mejora o el progreso. La mejora de habilidades es un estándar dentro de los juegos 15 de acción, sin embargo, en este caso se pretende ir un paso más allá, pues las habilidades van a modificar de manera fundamental cómo se juega.

La mejora de habilidades es la parte más importante en el diseño, puesto que de aquí depende el éxito del videojuego. Las mejoras en las habilidades del personaje van desde aumento de salud, daño y el resto de las variables básicas, hasta permitir al jugador teletransportarse, saltar de manera infinita, evadir y contrarrestar ataques enemigos.

Habilidades como llaves

Un concepto básico que hay que concretar es el de habilidades como llaves. Una llave es un objeto que permite al jugador acceder a un objeto, sección o mecánica. Es común dentro del diseño de videojuegos que se diseñe una habilidad como llave, alterando el entorno en el que se mueve el jugador y permitiéndole acceder a secciones que anteriormente no podía. Un buen diseño permitirá al jugador cambiar la ruta por la que se mueve a través del juego.

Diseño ambiental

El diseño ambiental o de escenarios está diseñado para incentivar y recompensar la exploración. Generalmente, el ambiente es un mundo interconectado, dividido en diversas secciones o escenarios. Esto permite al jugador tomar diversas rutas para llegar a zonas importantes.

A medida que el jugador avanza dentro de la historia del juego, los obstáculos deben cambiar para reflejar las habilidades que el personaje principal ha adquirido hasta el momento. El ejemplo más prominente tiene que ver con la movilidad. A medida que el jugador progresa, se vuelve más móvil, por lo que el diseño de escenarios comenzará a colocar plataformas más altas o separadas una de otra, para sacar provecho y forzar al jugador a implementar sus habilidades acrobáticas.

Un buen diseño de habilidades conlleva un buen diseño de escenarios, dado que puede funcionar como fuente de inspiración a la hora de construir los niveles de la mitad o final del juego.

Coleccionables y exploración.

Como se dijo anteriormente, el género posee un enfoque muy prominente hacia la exploración y recompensa al jugador por hacerlo. Es muy importante que el videojuego posea secretos, los cuales generalmente vienen en dos presentaciones.

La primera presentación son los secretos que el jugador puede encontrar con las habilidades y mecánicas básicas. El otro caso, son los que requieren una habilidad en específico. De aquí, parte siguiente premisa de diseño.

Backtracking

Backtracking es el término que se le otorga a regresar a áreas previamente completadas en busca de secretos, coleccionables y caminos a los que previamente no se tenía acceso, para tentar al jugador y otorgarle un sentimiento de progreso, al ver la diferencia en jugabilidad desde la última vez que pasó por la zona.

Existe un peligro inminente al diseñar las zonas con backtracking, el cual consta en volverlo tedioso al diseñar la zona muy larga, que tome mucho tiempo regresar, o sobreexplotar la mecánica.

Una de las técnicas más utilizadas para minimizar este riesgo, es el crear estaciones de fast travel. Una estación de fast travel es una mecánica en la cual interactúas con un objeto del juego que te permite viajar a zonas previamente finalizadas, esto vuelve el backtracking más corto y simple de implementar.

Planeación de mecánicas de sistema

Para construir las mecánicas básicas, hay que seguir los siguientes pasos:

1. Definir la premisa del videojuego y qué es lo que lo hace único

- ¿El videojuego será 3D o 2D?
- ¿Cómo funciona la cámara? (Side scroller, overhead, controlada por el jugador, tercera persona, basada en spline, programada, estática, etcétera).
- ¿Qué es lo que controla el jugador? (Un personaje, vehículo, objeto, etcétera).
- ¿De qué estilo es el juego? (Sci-fi, fantasía, realista, cartoony, exageración, o una combinación de estilos).
- ¿Qué es lo que define al juego y lo hace único? (Mecánicas, premisa, trabajo visual, trabajo de audio, historia, personajes).

2. Definir las mecánicas básicas del jugador

- ¿Cómo se mueve el personaje? (Vuela, camina, corre, se desliza, etcétera).
- ¿Qué hace el jugador para interactuar con el mundo? (Salta, dispara, corta, besa, agarra, toca).

3. Definir cuáles son las habilidades, armas, herramientas, equipamiento que el personaje va a poder adquirir

- ¿De cuántas maneras se puede mejorar la movilidad del personaje? (Salta más alto, corre, se agacha, se desliza, nada, gira, etc).
- ¿De cuántas maneras se puede transformar el personaje? (Cambia su tamaño, se convierte en otro personaje, altera su estado, sus habilidades). Todas las transformaciones modifican el movimiento, propiedades, colisiones y vulnerabilidades, por lo que requieren limitaciones y mecánicas únicas.
- ¿De cuántas maneras puede crecer? (Armas, proyectiles, magia, tipos de daño elemental, velocidad).
- ¿De cuántas maneras puede ver el mundo y cómo el mundo lo ve a él? (Opciones de visibilidad, invisibilidad, luces, dimensiones, secretos).

- ¿Cuántos tipos de obstáculos y hazards hay en el juego y de qué maneras pueden prevenirse? (Armadura, sombreros, trajes, amuletos para prevenir daño de hazards como el fuego, lava, agua, enemigos, electricidad, vacío, etc).
- ¿Cuáles habilidades puede adquirir que le permitan alterar el ambiente? (Detener el tiempo, cambiar la gravedad, fast travel, teletransportación).
- ¿El personaje tiene salud, requiere recargar munición, combustible o alguna otra variable? ¿Se recargan por sí mismas? (Con cuánto comienza y cuál es la capacidad máxima).

Implementación de sistemas

Una vez que se planearon y conceptualizaron las mecánicas de sistema, es momento de implementarlas. Un buen diseño debe implementar las mecánicas de manera versátil, ofreciendo múltiples usos para la misma habilidad. Las mecánicas crean nuevas oportunidades en el campo de la jugabilidad e incluyen limitaciones que creen retos para el jugador. En esta área es en la que fallan la gran mayoría de títulos.

Es recomendable colocar una lista de las habilidades a ser adquiridas por el jugador. Podría lucir de la siguiente manera:

- Habilidades por defecto: caminar, cortar, saltar.
- Upgrades a habilidades: salto alto, proyectil de fuego, transformación en gato, traje de veneno, visión nocturna, gravedad invertida, correr.

Estándares de habilidades

Es recomendable tener estándares de construcción que muestren los rangos en los que se desempeñarán las habilidades. Esto ayuda mucho al momento de probar y balancear el juego. Por ejemplo, que la altura máxima por defecto de salto sea de 2 metros y la altura máxima

del salto alto sea de 5 metros. Siempre es buena práctica tener ilustraciones y diagramas en los que se represente visualmente el estándar.

Diseño de pruebas de secuencia y progresión

Una vez que se implementan las mecánicas, se necesita diseñar un entorno de pruebas que te permita probar los estándares de habilidades previamente definidos y codificados.

El entorno de pruebas es una colección de cuartos que permitirá diseñar la secuencia en la que las habilidades van a desbloquearse y presentarse al jugador. Los objetivos del entorno de pruebas son los siguientes:

- Permitir al jugador entrar al cuarto y ver una representación gráfica de la habilidad que van a desbloquear
- Forzar al jugador a una situación en la que no opción y debe de tomar el ítem
- Permitir al jugador el interactuar con el objeto para adquirir la nueva habilidad
- Permitir al jugador utilizar la habilidad adquirida para poder entrar al siguiente cuarto

Cada uno de los cuartos deben encapsular la experiencia, no necesitan ser muy grandes y no necesitan verse bien. Toda la información debe ser provista de inmediato en la cámara y debe de representar el uso más general de la habilidad. La situación debe de ser muy simple de comprender. Una vez que los cuartos se encuentran diseñados, hay que colocarlos en un orden en el que sea interesante adquirirlos.

Una vez respondidas las preguntas, hay que mejorar el diseño modificando el orden y los estándares de las habilidades, repitiendo la cuantificación y pruebas hasta que exista una completa satisfacción.

Pruebas ambientales, de objetos y enemigos

Hay que generar otro entorno de pruebas para ajustar el ambiente, los objetos y los enemigos a implementar dentro del juego. Algunos de los elementos más importantes a colocar dentro de este entorno son los siguientes:

- Puertas: las puertas separan los diversos escenarios y áreas del juego, además de ser una excelente manera de bloquear el progreso hasta que se adquiera la llave correspondiente.
- Destructibles: son objetos con los que se puede interactuar al destruirlos, romperlos o dañarlos. Cada uno debe servir un propósito y ser ajustado para ser superados con éxito tras adquirir la herramienta.
- Obstáculos: son todos los objetos que puedan dañar al jugador. Todos deben de servir un propósito y ser superados con facilidad con la herramienta correcta.
- Mecanismos y objetos que se mueven: son todo lo que requiera el uso de habilidades para ser activadas. Ejemplos de este tipo son las ziplines, los elevadores, plataformas móviles, engranes y pistones.

Enemigos

Todos los enemigos deben de ser divertidos y suponer un reto, sin importar con qué habilidad se les enfrente. Se deben de planear de tal manera que exista por lo menos un enemigo por habilidad, o que sea vulnerable a la misma. Es importante tener enemigos que supongan un reto, inclusive cuando el jugador ha llegado al punto máximo de crecimiento. Son una parte muy importante en la ecuación que asegura el éxito del juego.

Diseño de niveles

Los niveles deben de ser diseñados de tal manera que supongan un reto e incentiven el uso de las habilidades adquiridas por el jugador, para otorgarle un sentimiento de progreso y recompensa. Al diseñar un nivel, hay que crear un mapa y una simbología que represente las

diversas partes que lo conforman, como lo son las puertas, rutas, nuevas habilidades, rutas, jefes y secretos. Se deben de etiquetar los bloques o secciones de un nivel en base a su propósito. Algunos de los propósitos generales que puede llegar a tener una sección del nivel, son los siguientes:

Exploración

Asegurar el uso de una habilidad

- Ruta alternativa
- Almacenar un secreto
- Cuarto de combate
- Cuarto de introducción
- Cuarto de transición
- Cuarto de puzzle
- Cuarto de jefe
- Cuarto de transición
- Salida
- Salida alternativa

Diseño

La fase de diseño está soportada por la teoría de la arquitectura de microservicios, la cual se presenta a continuación. Además, se escogieron las herramientas necesarias para llevar a cabo el proyecto, incluyendo los lenguajes de programación, el entorno de desarrollo y la plataforma de deployment.

Arquitectura de Microservicios

Los microservicios son un enfoque arquitectónico de desarrollo de software en donde la aplicación consta de una colección de servicios autónomos, independientes y pequeños. Los servicios se comunican entre sí mediante API bien definidas y los detalles de la implementación interna de cada servicio se ocultan frente a otros servicios. A nivel de desarrollo, cada uno de estos microservicios es un código base independiente, que puede administrarse por un equipo de desarrollo pequeño.

A diferencia de una arquitectura monolítica, en donde todos los procesos están estrechamente asociados, en una arquitectura de microservicios. Los servicios son independientes entre sí y cada uno debe implementar una funcionalidad de negocio individual.

Las principales ventajas de utilizar una arquitectura de Microservicios frente a las demás, es la agilidad con la que se administran las correcciones de errores y el versionado, las bases de código pequeñas, la gran variedad de tecnologías, la escalabilidad y el aislamiento de los datos. Además, permite desacoplar completamente una funcionalidad de negocio del resto.

Los principales retos frente a una arquitectura orientada a microservicios son la complejidad y la falta de gobernanza, pues con tantos lenguajes y marcos de trabajo, la aplicación puede ser difícil de mantener. Otro reto es el de la integridad de los datos, pues cada microservicio es responsable de la conservación de sus propios datos.

En la Figura 2, se puede apreciar la comparación entre una arquitectura monolítica y una enfocada a microservicios.

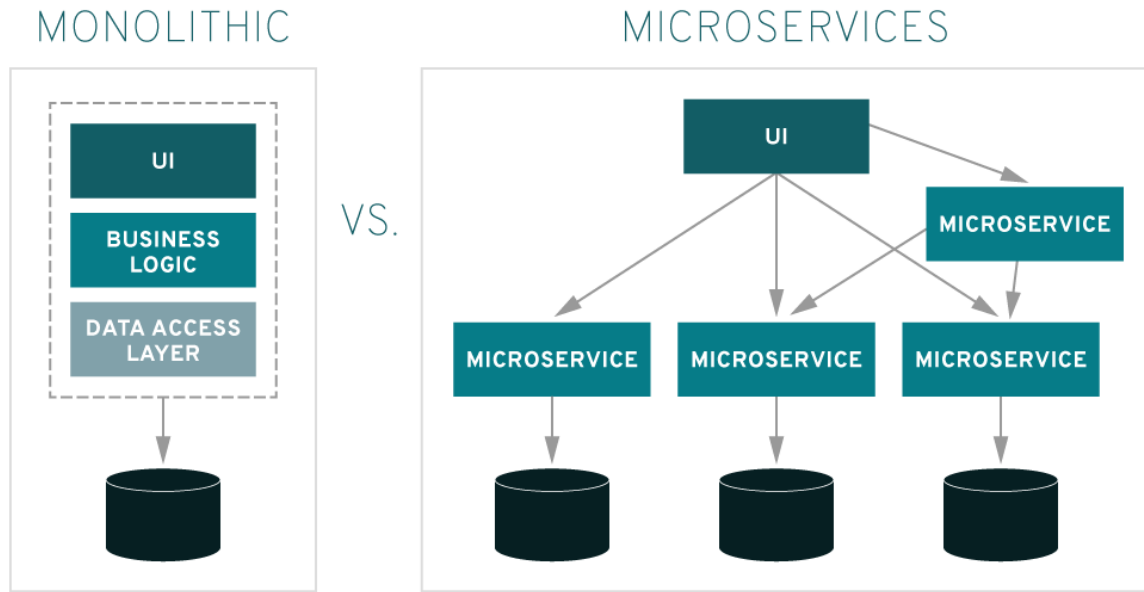


Figura 2 Comparación entre arquitectura monolítica y microservicios

Spring Boot

Spring es un framework basado en Java que permite desarrollar aplicaciones basadas en microservicios, para la nube, Web apps, serverless y API. Es el framework Web más popular de Java. Spring Boot es una suite que permite facilitar el desarrollo en Spring, otorgando al framework una serie de configuraciones automatizadas que aceleran el desarrollo de la aplicación, pasando más tiempo en el desarrollo de la lógica de negocio que en configurar librerías y servicios. Todo esto lo hace gracias a un patrón de diseño de software conocido como inyección de dependencias.

En la Figura 3, se puede observar el logotipo de Spring Boot



Figura 3 Logotipo de Spring Boot

Unity3D

Unity3D o Unity es una plataforma de desarrollo 3D que permite la creación de diversos tipos de contenido multimedia, como lo son animaciones y videojuegos. Coloquialmente se le considera un Game Engine o motor de videojuegos, pero su uso se extiende hacia la industria cinematográfica y la animación digital. Posee una curva de aprendizaje muy simple y permite la creación de videojuegos multiplataforma bajo la misma base de código (se escribe la aplicación una vez, se exporta a diversos sistemas operativos).

En la Figura 4 se puede observar el logotipo de Unity.



Figura 4 Logotipo de Unity3D

IntelliJ IDEA

IntelliJ IDEA es un entorno de desarrollo integrado creado por JetBrains. Soporta una gran variedad de lenguajes, como lo son Python, Java, Clojure, Rust, Dart, Erlang, Go, Perl, Scala y Kotlin.

En la Figura 5, se puede observar el logotipo de IntelliJ IDEA.



Figura 5 Logotipo de IntelliJ IDEA

DigitalOcean

DigitalOcean es una empresa americana que provee infraestructura de nube. Otorga servicio a desarrolladores de software y compañías de IT para alojar sus aplicaciones y escalarlas.

En la Figura 6, se puede observar el logotipo de DigitalOcean.



Figura 6 Logotipo de DigitalOcean

4. Desarrollo

El proyecto consta de apoyar a una de las estudiantes de Maestría en Ciencias de la Computación con su tesis, que involucra una investigación acerca de qué tanto puede influir el desarrollo de videojuegos educativos en los hábitos alimenticios de los niños. Al igual que en el marco teórico, se presentará la información en el mismo orden que soporta la metodología de Scrum.

Análisis

El análisis se realizó en una serie de juntas a lo largo de las primeras semanas del proyecto. Se reunió con la asesora externa, quien otorgó al equipo la información necesaria respecto a los alimentos que se pueden incluir en la alimentación de los niños y junto a ella, se definieron los videojuegos que formarían parte de la aplicación. Además, se definieron roles y se estipularon fechas para hacer las juntas de fin de sprint, las daily meetings y la retroalimentación.

Diseño

La fase de diseño constó en la creación de la arquitectura general del proyecto y de cada uno de los módulos que la integrarían. El trabajo realizado incluye el diseño de la API, la integración con la base de datos previamente diseñada y la creación de una interfaz que le permitiera al desarrollador de los minijuegos control total de los puntos de entrada en la API y se puedan escalar de forma simple.

La plataforma consta de dos partes:

- 1- Una API que simplificará la conexión entre la base de datos y los videojuegos, con fines de poder escalar el proyecto de forma mucho más simple.
- 2- Una aplicación que contendrá una serie de videojuegos para que el niño refuerce conceptos del área nutricional mientras se divierte.

Para el desarrollo del proyecto, en el rol de arquitecto de software, se definió que la mejor manera de organizar la aplicación sería bajo microservicios, dado el alcance y la escalabilidad que implicaría a futuro la aplicación, otorgando extensibilidad al proyecto.

En la Figura 7, se puede observar el diagrama de despliegue de la arquitectura general de la plataforma.

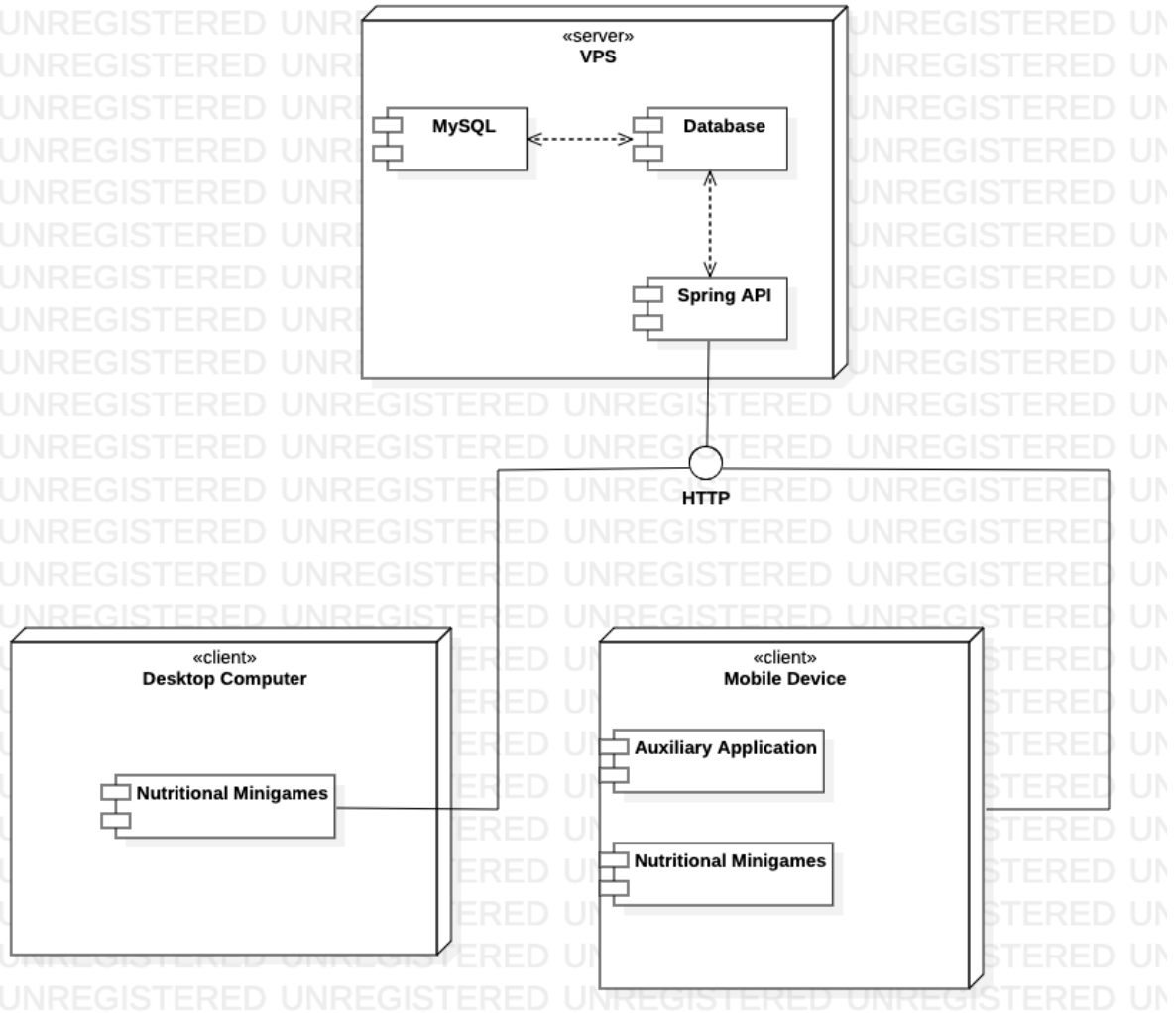


Figura 7 Diagrama de Despliegue del proyecto

La Base de Datos fue previamente desarrollada por otro miembro del equipo y se conecta directamente con la API, la cual se encarga de servir como interfaz entre los datos y los clientes, que en este caso vienen a ser la aplicación auxiliar desarrollada en Android y los minijuegos, implementados en Unity. Dado que Unity es multiplataforma, se consideran clientes de escritorio y móviles. Finalmente, todos los nodos de la plataforma se conectan mediante el protocolo HTTP.

El motor de base de datos previamente seleccionado fue MySQL. No existió ningún conflicto al momento de integrarla con Spring Boot, dado que Spring posee soporte nativo a este motor de base de datos. En la figura 8, se puede observar el diagrama Entidad-Relación de la base de datos, bajo el estándar de UML 2.5

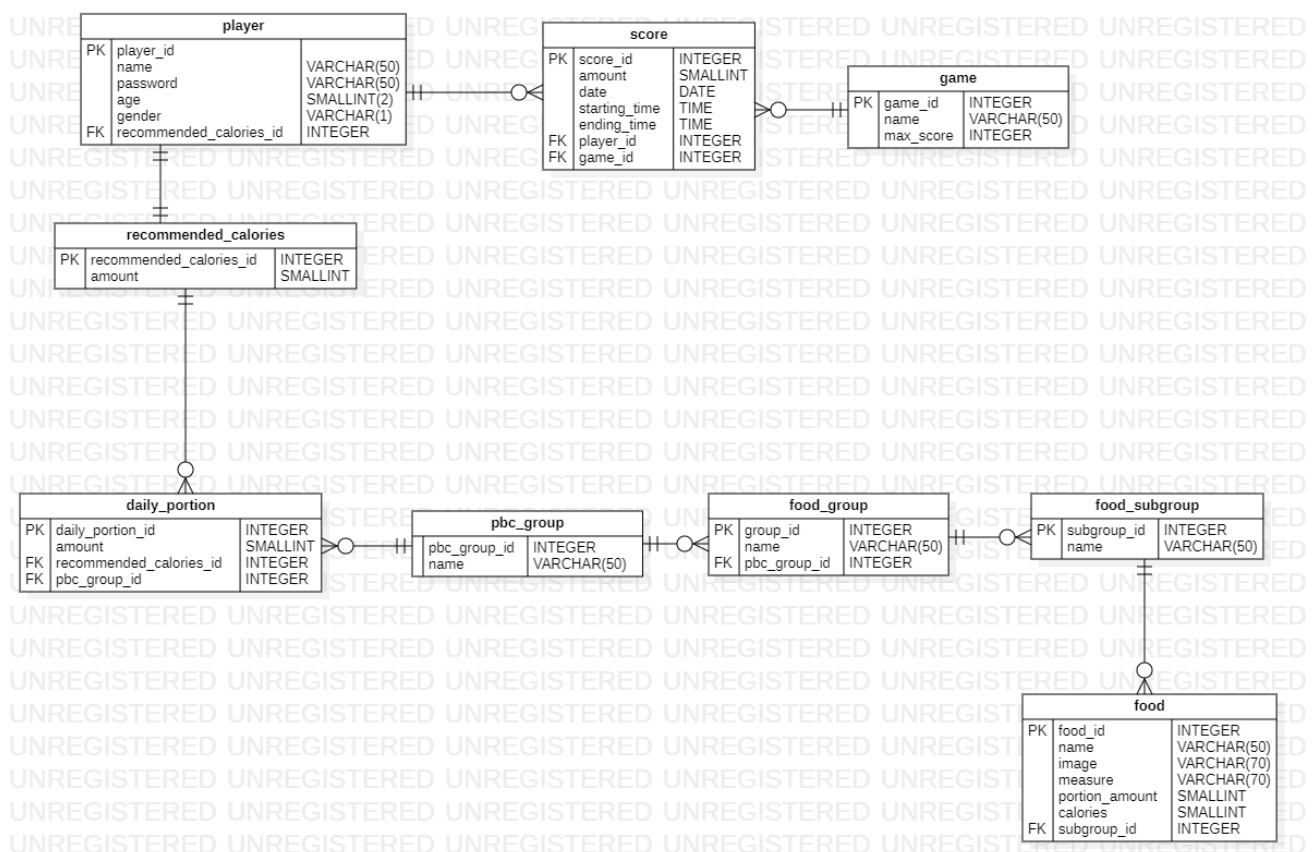


Figura 8 Diagrama ER del proyecto

En la tabla 1, se puede observar el diccionario de datos correspondiente a la base de datos del proyecto.

Tabla	Campo	Tamaño	Tipo de Dato	Descripción
-------	-------	--------	--------------	-------------

Food	<i>food_id</i>	255	Entero int	Id y llave primaria
	<i>name</i>	50	Cadena varchar	Nombre del alimento
	<i>image</i>	70	Cadena varchar	Enlace hacia la imagen que representa al alimento
	<i>measure</i>	70	Cadena varchar	Medida que se usa en el alimento
	<i>portion_amount</i>	3	Entero smallint	Cantidad por porción
	<i>calories</i>	5	Entero smallint	Cantidad de calorías por porción
	<i>subgroup_id</i>	255	Entero int	Llave Foránea a la tabla Subgroups
	<i>group_id</i>	255	Entero int	Llave Foránea a la tabla Groups
Subgroups	<i>subgroup_id</i>	255	Entero int	Id y llave primaria
	<i>name</i>	50	Cadena varchar	Nombre del subgrupo de alimentos
Groups	<i>group_id</i>	255	Entero int	Id y llave primaria
	<i>name</i>	50	Cadena varchar	Nombre del grupo de alimentos
	<i>pb_group_id</i>	255	Entero int	Llave foránea a la tabla PbcGroups
PbcGroups	<i>pb_group_id</i>	255	Entero int	Id y llave primaria
	<i>name</i>	50	Cadena varchar	Nombre del grupo de alimentos en el plato del buen comer
DailyPortions	<i>daily_portion_id</i>	255	Entero int	Id y llave primaria
	<i>amount</i>	5	Entero smallint	Cantidad definida en la porción

	<i>recommended_calories_id</i>	255	Entero int	Llave foránea a la tabla RecommendedCalories
	<i>pbcs_group_id</i>	255	Entero int	Llave foránea a la tabla PbcGroups
Recommend edCalories	<i>recommended_calories_id</i>	255	Entero int	Id y llave primaria
	<i>amount</i>	5	Entero smallint	Cantidad recomendada de calorías
Players	<i>player_id</i>	255	Entero int	Id y llave primaria
	<i>name</i>	50	Cadena varchar	Nombre del jugador
	<i>password</i>	50	Cadena varchar	Contraseña del jugador
	<i>age</i>	2	Entero smallint	Edad del jugador
	<i>gender</i>	1	Cadena Enum	Género del jugador
	<i>recommended_calories_id</i>	255	Entero int	Llave foránea a la tabla RecommendedCalories
Scores	<i>score_id</i>	255	Entero int	Id y llave primaria
	<i>amount</i>	255	Entero int	Cantidad lograda en el puntaje
	<i>date</i>	NA	Fecha date	Fecha de realización
	<i>starting_time</i>	NA	Hora time	Hora de inicio
	<i>ending_time</i>	NA	Hora time	Hora final
	<i>player_id</i>	255	Entero int	Llave Foránea a la tabla Players
	<i>game_id</i>	255	Entero int	Llave Foránea a la llave Games
Games	<i>game_id</i>	255	Entero int	Id y llave primaria

	<i>name</i>	50	Cadena varchar	Nombre del juego
	<i>max_score</i>	255	Entero int	Puntuación Máxima del juego

Tabla 1 Diccionario de Datos del proyecto

Para desarrollar la API, se optó por diseñar bajo el framework conocido como Spring Boot. La API funciona bajo el protocolo HTTP y funciona similar a una RESTful API, con puntos de entrada que realizan operaciones muy específicas dentro de la base de datos. El criterio HATEOAS no se tomó en cuenta, por lo que la API no es considerada completamente RESTful. Por simplicidad, se tomó la decisión de no implementar metadatos ni volverla navegable mediante hipertexto, sin embargo, puede ser fácilmente implementada.

El diseño de los endpoints de la API es un paso muy importante, puesto que en este se definen cuáles son las operaciones que los clientes pueden realizar dentro de la base de datos. Esto tiene implicaciones tanto de seguridad, como de arquitectura, puesto que es fundamental para otorgarle escalabilidad al proyecto. En la tabla 2, se puede observar el diseño final de los endpoints que soportará la primera versión de la API.

Endpoint	Verbo HTTP	Descripción
<i>v1/food</i>	GET	Retorna todos los alimentos
<i>v1/food?name=</i>	GET	Busca y retorna un alimento por nombre
<i>v1/food?measure=</i>	GET	Busca y retorna una lista de alimentos por medida
<i>v1/food?portion_amount=</i>	GET	Busca y retorna una lista de alimentos por porcion

<i>v1/food/{food_id}</i>	GET	Busca y retorna un alimento por id
<i>v1/foodGroups</i>	GET	Retorna todos los grupos de alimentos
<i>v1/foodGroups?name=</i>	GET	Busca y retorna un grupo de alimentos por nombre
<i>v1/foodGroups/{food_group_id}</i>	GET	Busca y retorna un grupo de alimentos por id
<i>v1/foodGroups/{food_group_id}/food</i>	GET	Busca y retorna los alimentos que pertenecen a un grupo
<i>v1/foodSubgroups/</i>	GET	Retorna todos los subgrupos de alimentos
<i>v1/foodSubgroups?name=</i>	GET	Busca y retorna un subgrupo de alimentos por nombre
<i>v1/foodSubgroups/{food_subgroup_id}</i>	GET	Busca y retorna un subgrupo de alimentos por id
<i>v1/foodSubgroups/{food_subgroups_id}/food</i>	GET	Busca y retorna los alimentos que pertenecen a un subgrupo
<i>v1/games</i>	GET	Retorna todos los minijuegos
<i>v1/games?name=</i>	GET	Busca y retorna un minijuego por nombre
<i>v1/games/{game_id}</i>	GET	Busca y retorna un minijuego por id. Adicionalmente, posee un parámetro opcional que le permite buscar por player_id

<i>v1/pbcGroups</i>	GET	Retorna todos los grupos del plato del buen comer
<i>v1/pbcGroups?name=</i>	GET	Busca y retorna un grupo del plato del buen comer por nombre
<i>v1/pbcGroups/{pbc_group_id}</i>	GET	Busca y retorna un grupo del plato del buen comer por id
<i>v1/pbcGroups/{pbc_group_id}/foodGroups</i>	GET	Busca y retorna los grupos de alimentos asociados con un grupo del plato del buen comer
<i>v1/pbcGroups/{pbc_group_id}/dailyPortions</i>	GET	Busca y retorna las porciones diarias asociadas con un grupo del plato del buen comer
<i>v1/players</i>	GET	Retorna todos los jugadores
<i>v1/players</i>	POST	Sube la información de un jugador
<i>v1/players/{player_id}</i>	GET	Busca y retorna la información de un jugador. Posee parámetros opcionales de género y nombre
<i>v1/players/{player_id}/scores</i>	GET	Busca y retorna los puntajes relacionados a un jugador.
<i>v1/recommendedCalories</i>	GET	Retorna todas las calorías recomendadas
<i>v1/recommendedCalories/{recommended_calories_id}</i>	GET	Busca y retorna un conjunto de calorías recomendadas por id
<i>v1/scores</i>	GET	Retorna todos los puntajes. Posee parámetros opcionales para buscar por fecha, por id de juego o por id de jugador

<code>v1/scores/</code>	POST	Sube la información de un puntaje
<code>v1/scores/{score_id}</code>	GET	Busca y retorna un puntaje por id

Tabla 2 Tabla de Endpoints de la API

Codificación

En la sección de codificación, se va a abordar el desarrollo propio de la API y todos sus módulos, además de la interfaz para Unity que funcionará como conector entre ambas tecnologías.

Nutritional Games API

La arquitectura de la API se compone de 3 capas:

- **Capa de Repositorio:** La capa de repositorio se encarga de ejecutar sentencias SQL en la base de datos. Utiliza la librería Hibernate y los repositorios JPA para generar código SQL de manera dinámica.
- **Capa de Servicio:** La capa de servicio sirve como interconexión entre la capa de repositorio y la capa de controlador. Se encarga de tareas misceláneas como la validación de información, el manejo de errores y la conversión de solicitudes HTTP en lógica de negocio.
- **Capa de Controlador:** La capa de controlador se encarga de ofrecer a los clientes una interfaz común, en donde cada uno de los puntos de acceso o endpoints, permiten generar una consulta, crear un registro o editarlo.

Antes de describir la arquitectura a fondo, hay que describir clases que representan los objetos que se van a almacenar dentro de la base de datos, en Spring Boot, esto se conoce como las Entidades.

En la Figura 9, se puede apreciar un ejemplo de la Entidad Food, la cual almacena la información de cada uno de los alimentos.

```
@Entity
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "food")
public class Food {

    @Id
    @Column(name = "food_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @JsonProperty("food_id")
    @Setter
    @Getter
    private Integer foodId;

    @JsonProperty("name")
    @Column(name = "name")
    @Setter
    @Getter
    private String name;

    @JsonProperty("image")
    @Column(name = "image")
    @Setter
    @Getter
    private String image;

    @JsonProperty("measure")
    @Column(name = "measure")
    @Setter
    @Getter
    private String measure;
```

Figura 9 Clase Entidad de Food

Se utilizan varias anotaciones de Java para agregarle información a la Entidad. Por ejemplo, se utilizan anotaciones de la librería Lombok, para generar Setters y Getters mediante programación dinámica, los cuales van a inyectarse automáticamente dentro del código en tiempo de ejecución. Además de esto, se definen los nombres de las columnas y las propiedades del JSON que devuelve la API.

La capa de repositorio funciona como una interfaz, la cual inyectará el código necesario para crear las sentencias SQL, todo esto utilizando la librería de Hibernate conocida como JPA. Esta librería te permite generar código que se comunica con la base de datos solamente describiendo su funcionamiento en el nombre del método.

En la Figura 10, se puede observar la capa de repositorio de la Entidad Food.

```
package com.itcg.nutritionalgames.repositories;

import ...

@Repository
public interface FoodRepository extends JpaRepository<Food, Integer> {
    Optional<Food> findByName(String name);
    List<Food> findByMeasure(String measure);
    List<Food> findByPortionAmount(short portionAmount);
    List<Food> findByCalories(short calories);
    List<Food> findByGroupId(int foodGroupId);
    List<Food> findBySubgroupId(int foodSubgroupId);
}
```

Figura 10 Interfaz repositorio de Food

La capa de servicio funciona como middleware entre la capa de repositorio y la capa de controlador, por lo que se encarga de transformar la información y de manejar los errores que se puedan presentar en la base de datos. En la Figura 11, se puede observar la capa de servicio de la Entidad Food.

```

public Food findFoodByName(String name) {
    return foodRepository.findByName(name)
        .orElseThrow(() -> new EntityNotFoundException("No food could be found with name = " + name));
}

public Food findFoodById(Integer foodId) {
    return foodRepository.findById(foodId)
        .orElseThrow(() -> new EntityNotFoundException("No food could be found with food_id : " + foodId));
}

public List<Food> findFoodByMeasure(String measure) {
    List<Food> foodList = foodRepository.findByMeasure(measure);

    if(foodList.isEmpty()) {
        throw new EntityNotFoundException("No food could be found with measure = " + measure);
    }

    return foodList;
}

```

Figura 11 Clase de servicio de Food

Finalmente, la capa de controlador es la que se encarga de generar los puntos de entrada mediante los cuales la aplicación se va a comunicar. Estos puntos de entrada te permiten definir operaciones únicas, como lo es solicitar todos los alimentos en la tabla Food, o subir el puntaje de un jugador. En la Figura 12, se puede observar la capa de controlador de la Entidad Food.


```

private final FoodService foodService;

@Autowired
public FoodController(FoodService foodService) { this.foodService = foodService; }

@GetMapping(value = "v1/food")
public List<Food> findAllFood() { return foodService.findAllFood(); }

@GetMapping(value = "v1/food", params = "name")
public Food findFoodByName(@RequestParam(value = "name") String name) { return foodService.findFoodByName(name); }

@GetMapping(value = "v1/food", params = "measure")
public List<Food> findFoodByMeasure(@RequestParam(value = "measure") String measure) {
    return foodService.findFoodByMeasure(measure);
}

@GetMapping(value = "v1/food", params = "portion_amount")
public List<Food> findFoodByPortionAmount(@RequestParam(value = "portion_amount") Short portionAmount) {
    return foodService.findFoodByPortionAmount(portionAmount);
}

@GetMapping(value = "v1/food", params = "calories")
public List<Food> findFoodByCalories(@RequestParam(value = "calories") Short calories) {
    return foodService.findFoodByCalories(calories);
}

@GetMapping(value = "v1/food/{food_id}")
public Food findFoodById(@PathVariable(value = "food_id") Integer foodId) {
    return foodService.findFoodById(foodId);
}

```

Figura 12 Clase controlador de Food

Configuración del entorno de producción y deployment de la API

Para la configuración del servidor de producción, se decidió alojar tanto la base de datos como la API dentro de un VPS (Servidor Virtual Privado), que forma parte de la nube de DigitalOcean. A continuación, se presenta una breve explicación del proceso de configuración para el entorno de producción:

En principio, se genera un archivo ejecutable en formato JAR, el cual contendrá el código de la API. La herramienta IntelliJ IDEA contiene una opción que te permite generarlo de forma muy simple. Se selecciona la opción bootJar en el apartado de Gradle.

En la Figura 13, se puede observar la pantalla de generación del ejecutable, utilizando Gradle.

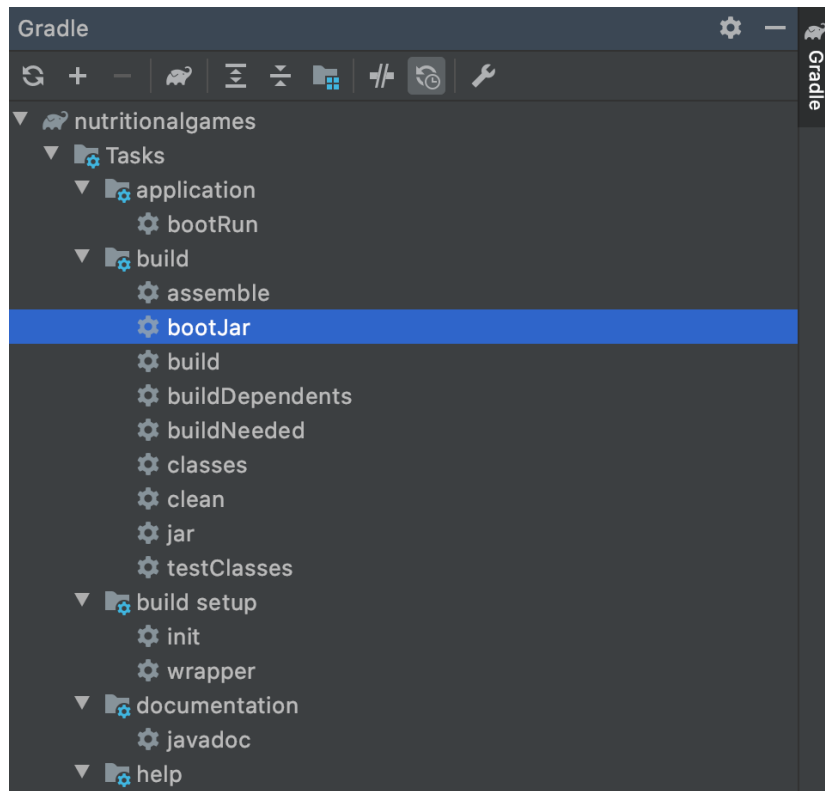


Figura 13 Generación del ejecutable con Gradle

Posteriormente, el ejecutable se encontrará en la siguiente ruta.

[carpeta del Proyecto]/build/libs/

En la Figura 14, se puede observar la carpeta en la que se localiza el ejecutable ya generado.

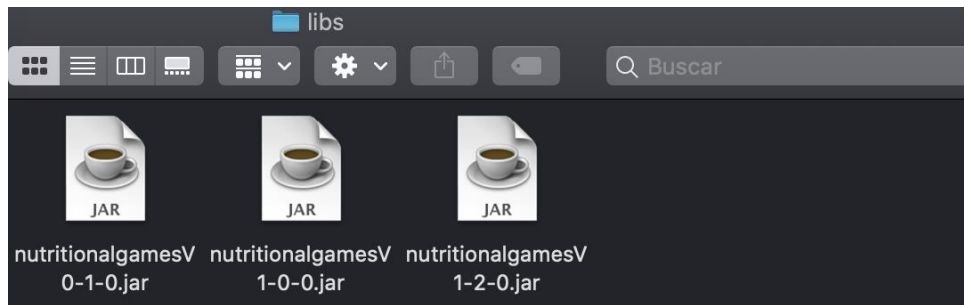


Figura 14 Localización del ejecutable generado

El siguiente paso, es conectarse a la infraestructura de Digital Ocean. Se crea un VPS o como se le llama en esta infraestructura, un droplet, con las características que requiera la aplicación. Después, se accede a el mediante la interfaz Web de consola, o a través de SSH. Se requiere instalar el servidor de MySQL en el droplet y la máquina virtual de Java, ya sea la oficial de Oracle, o la que contiene el OpenJDK.

En la Figura 15, se puede observar la pantalla principal del Droplet de producción, desde la interfaz de DigitalOcean. En la Figura 16, se puede observar la conexión exitosa por SSH hacia el Droplet de producción.

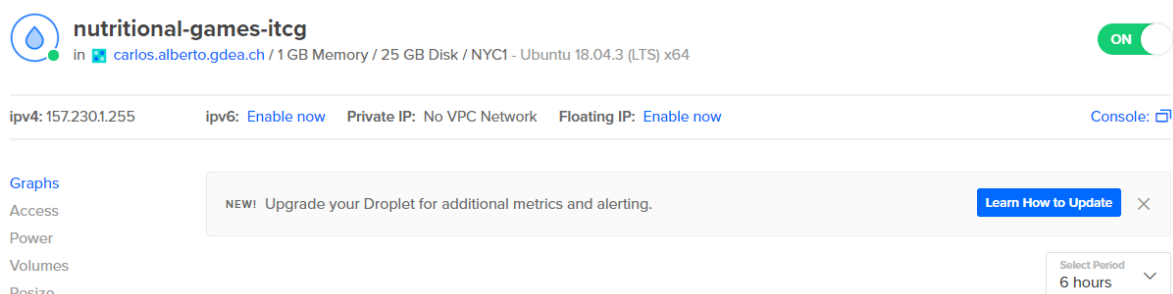


Figura 15 Pantalla del Droplet de producción en DigitalOcean

```

~ via v2.7.16 on us-east-2
> ssh root@157.230.1.255
Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 4.15.0-66-generic x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:        https://ubuntu.com/advantage

System information as of Mon May 25 16:05:35 UTC 2020

System load:  0.0               Processes:    91
Usage of /:   10.2% of 24.06GB  Users logged in:  0
Memory usage: 77%              IP address for eth0: 157.230.1.255
Swap usage:  0%

* Canonical Livepatch is available for installation.
- Reduce system reboots and improve kernel security. Activate at:
  https://ubuntu.com/livepatch

43 packages can be updated.
0 updates are security updates.

*** System restart required ***
Last login: Thu Apr 23 20:45:45 2020 from 187.194.35.20
root@nutritional-games-itcg:~#

```

Figura 16 Conexión al Droplet por SSH

Una vez que se tiene acceso, hay que subir el archivo hacia el servidor. En este caso, se utilizó el protocolo SFTP. En la Figura 17 se puede observar el deployment exitoso de la aplicación.

```

nutritional-games-api/build/libs on master [!] via v9.0.1 on us-east-2
> sftp root@157.230.1.255
Connected to 157.230.1.255.
sftp> put nutritionalgamesV1-2-0.jar
Uploading nutritionalgamesV1-2-0.jar to /root/nutritionalgamesV1-2-0.jar
nutritionalgamesV1-2-0.jar
sftp>

```

Figura 17 Deployment de la aplicación por SFTP

Posteriormente, se tiene que crear un archivo de configuración en el servidor, que permitirá que el archivo ejecutable JAR se pueda ejecutar como un servicio del sistema operativo.

Para comenzar este paso, se accede a la ruta /etc/systemd/system, como se puede observar en la Figura 18.

```

root@nutritional-games-itcg:/etc/systemd/system# ls
cloud-final.service.wants      final.target.wants      multi-user.target.wants      paths.target.wants      syslog.service
cloud-init.target.wants        getty.target.wants      network-online.target.wants  sockets.target.wants    timers.target.wants
dbus-org.freedesktop.resolve1.service  graphical.target.wants  nutritional-games.service    sshd.service
default.target.wants          iscsi.service          open-vm-tools.service.requires  sysinit.target.wants
root@nutritional-games-itcg:/etc/systemd/system#

```

Figura 18 Directorio de la configuración de los servicios en GNU/Linux

Después, se crea el archivo de servicio, que en este caso lleva por nombre nutritional-games.service. Este archivo se rellena con la información necesaria para que el servicio se pueda ejecutar correctamente. En la Figura 19, se puede observar el archivo de configuración del servicio.

```

GNU nano 2.9.3 nutritional-games.service

[Unit]
Description=NutritionalGames
After=syslog.target

[Service]
User=root
ExecStart=/nutritionalgames.jar SuccessExitStatus=143

[Install]
WantedBy=multi-user.target

```

Figura 19 Archivo de configuración del servicio

Una vez que se tiene el archivo guardado, se lanza el servicio y debe de dar estatus activo en color verde, como se puede observar en la Figura 20.

```

root@nutritional-games-itcg:/etc/systemd/system# sudo service nutritional-games status
● nutritional-games.service - NutritionalGames
   Loaded: loaded (/etc/systemd/system/nutritional-games.service; disabled; vendor preset: enabled)
   Active: active (running) since Tue 2020-03-10 21:47:13 UTC; 2 months 14 days ago
     Main PID: 15366 (nutritionalgame)
        Tasks: 29 (limit: 1152)
      CGroup: /system.slice/nutritional-games.service
              └─15366 /bin/bash /nutritionalgames.jar SuccessExitStatus=143
                └─15397 /usr/bin/java -Dsun.misc.URLClassPath.disableJarChecking=true -jar //nutritionalgames.jar SuccessExitStatus=143

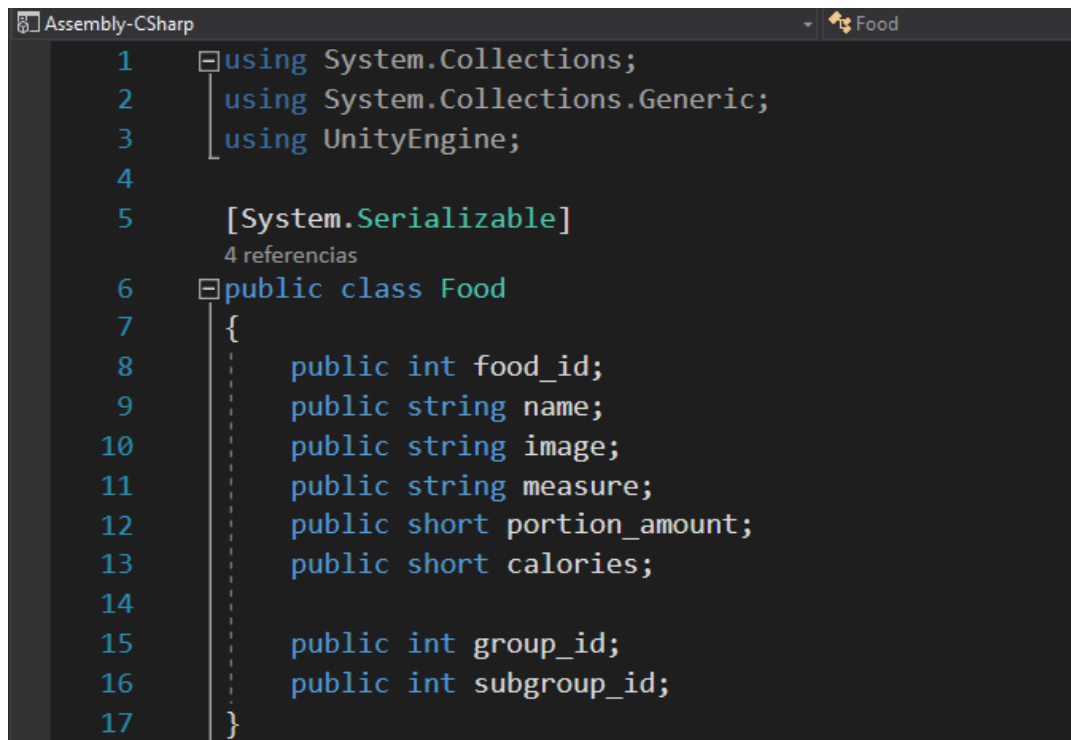
May 25 04:56:49 nutritional-games-itcg nutritionalgames.jar[15366]: at org.apache.coyote.http11.Http11Processor.service(Http11Processor.jav
May 25 04:56:49 nutritional-games-itcg nutritionalgames.jar[15366]: at org.apache.coyote.AbstractProcessorLight.process(AbstractProcessorLi
May 25 04:56:49 nutritional-games-itcg nutritionalgames.jar[15366]: at org.apache.coyote.AbstractProtocol$ConnectionHandler.process(Abstract
May 25 04:56:49 nutritional-games-itcg nutritionalgames.jar[15366]: at org.apache.tomcat.util.net.NioEndpoint$SocketProcessor.doRun(NioEndp
May 25 04:56:49 nutritional-games-itcg nutritionalgames.jar[15366]: at org.apache.tomcat.util.net.SocketProcessorBase.run(SocketProcessorBa
May 25 04:56:49 nutritional-games-itcg nutritionalgames.jar[15366]: at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecuto
May 25 04:56:49 nutritional-games-itcg nutritionalgames.jar[15366]: at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecuto
May 25 04:56:49 nutritional-games-itcg nutritionalgames.jar[15366]: at org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskT
May 25 04:56:49 nutritional-games-itcg nutritionalgames.jar[15366]: at java.lang.Thread.run(Thread.java:748) [na:1.8.0_242]
May 25 04:56:49 nutritional-games-itcg nutritionalgames.jar[15366]: 2020-05-25 04:56:49.765 WARN 15397 --- [nio-8080-exec-7] .w.s.m.s.DefaultHandl
Lines 1-19/19 (END)

```

Figura 20 Consulta de estatus del servicio

Una vez que la API se encontraba en funcionamiento, otra de las actividades exitosas, fue la de desarrollar una interfaz en el motor de Unity, que permitiera conectarse y recibir información desde el servidor. Esto se realizó en el lenguaje C# y a continuación se presenta el diseño final.

La primera sección de la interfaz de cliente en Unity contiene una lista de modelos, los cuales reflejan la información contenida en el archivo JSON. Citando un ejemplo, en la Figura 21 se presenta la información contenida en cada uno de los objetos de la clase Food.

The image shows a screenshot of a C# code editor window titled 'Assembly-CSharp'. The code defines a public class named 'Food' which is marked with the '[System.Serializable]' attribute. The class contains several public fields: 'food_id' (int), 'name' (string), 'image' (string), 'measure' (string), 'portion_amount' (short), 'calories' (short), 'group_id' (int), and 'subgroup_id' (int). The code is as follows:

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  [System.Serializable]
6  public class Food
7  {
8      public int food_id;
9      public string name;
10     public string image;
11     public string measure;
12     public short portion_amount;
13     public short calories;
14
15     public int group_id;
16     public int subgroup_id;
17 }
```

Figura 21 Modelo Food de la interfaz de Unity

En Unity, se marca con la sentencia `System.Serializable`, para permitirle a Unity que pueda serializar esta clase utilizando la interfaz de JSON. Dado que la información de la clase Food

no tiene ninguna restricción de acceso, se optó por manejarla de forma pública, sin generar Setter/Getter.

Una vez que se tiene el modelo, se crea una clase contenedora, la cual va a encapsular a la clase Food, como se puede observar en la Figura 22.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  [System.Serializable]
6  public class FoodListWrapper
7  {
8      public List<Food> data;
9  }
```

Figura 22 Wrapper del modelo Food

La clase FoodListWrapper, se encarga de encapsular la clase Food para que se asemeje al formato de objeto que devuelve la API.

Posteriormente, se creó una clase llamada EndpointBuilder, la cual contiene toda la información para que el desarrollador pueda extender fácilmente el comportamiento de la API, sin necesidad de reestructurar el código. Todo esto se diseñó bajo el patrón de diseño Builder. En la Figura 23, se puede apreciar la estructura de la clase.

```

5 public class EndpointBuilder
6 {
7
8     private string url = "http://157.230.1.255:8080";
9
10    1 referencia
11    private EndpointBuilder()
12    {
13    }
14
15    9 referencias
16    public static EndpointBuilder AnEndpoint()
17    {
18        return new EndpointBuilder();
19    }
20
21    9 referencias
22    public EndpointBuilder UsingVersion1()
23    {
24        url += "/v1/";
25        return this;
26    }
27 }

```

Figura 23 Estructura de la clase EndpointBuilder

Este patrón de diseño permite al desarrollador concatenar palabras en inglés plano, describiendo de forma muy simple el endpoint que requiere. Al final, se llama el método Build(), el cual va a retornar la liga hacia el recurso, como podemos observar en la Figura 24.


```

67 1 referencia
68 public EndpointBuilder WithRecommendedCaloriesUrl()
69 {
70     url += "recommendedCalories";
71     return this;
72 }
73 1 referencia
74 public EndpointBuilder WithScoresUrl()
75 {
76     url += "scores";
77     return this;
78 }
79 9 referencias
80 public string Build()
81 {
82     return url;
83 }

```

Figura 24 Segunda parte de la estructura de la clase EndpointBuilder

Se creó una clase llamada APIConnection, la cual se conecta directamente al servidor utilizando la clase WebClient, propia de Unity. Otra de sus actividades, es el ejecutar las Query a los Endpoints y serializar la información desde el archivo JSON. Para esto, posee el método ExecuteQueryAtEndpoint, el cual descarga una cadena con el contenido que devuelve la operación HTTP y lo manda hacia el método FormatJsonForDeserializingList.

En la Figura 25 se puede apreciar la estructura y el método ExecuteQueryAtEndpoint.

```

9 referencias
public class APIConnection
{
    private static WebClient webClient = new WebClient();

    9 referencias
    public static string ExecuteQueryAtEndpoint(string endpoint)
    {
        string retrievedJson = webClient.DownloadString(endpoint);
        retrievedJson = FormatJsonForDeserializingList(retrievedJson);
        return retrievedJson;
    }
}

```

Figura 25 Estructura de la clase APIConnection y el método ExecuteQueryAtEndpoint

El método FormatJsonForDeserializingList, agrega un conjunto de caracteres a la cadena recibida desde el Endpoint, para hacerla coincidir con el formato que recibe el deserializador de Unity, como se puede observar en la Figura 26.

```

1 referencia
private static string FormatJsonForDeserializingList(string json)
{
    return "{\\"data\\": " + json + "}";
}

```

Figura 26 Método FormatJsonForDeserializingList

Finalmente, se generaron métodos para recibir la información desde cada uno de los endpoints. En el siguiente ejemplo se puede ver la implementación del patrón Builder y cómo simplifica el desarrollo. En vez de generar un endpoint literal, almacenado en una cadena como “<http://157.230.1.255:8080/v1/food>”, se genera describiendo qué es lo que el desarrollador requiere.

“Build an endpoint using version 1 with Food Url”, que traducido al español indica “Construye un endpoint utilizando la versión 1 con la url de Alimentos”.

La liga se genera de forma dinámica y permite extender la interfaz de forma muy rápida y simple. Finalmente, se ejecuta la sentencia en el Endpoint y la información se deserializa utilizando la librería JsonUtility, propia de Unity, pasándole como parámetro cuál es el Wrapper que contiene el formato a ser deserializado. En la Figura 27, se puede observar el método GetAllFood con toda la deserialización integrada.

```
1 referencia
public static List<Food> GetAllFood()
{
    string endpoint = EndpointBuilder.AnEndpoint()
        .UsingVersion1()
        .WithFoodsUrl()
        .Build();

    Debug.Log("Query: " + endpoint);

    string retrievedJson = ExecuteQueryAtEndpoint(endpoint);
    return JsonUtility.FromJson<FoodListWrapper>(retrievedJson).data;
}
```

Figura 27 Método GetAllFood

Pruebas

Lo relacionado a las pruebas, se presentará en el siguiente capítulo, en la sección de Resultados.

5. Resultados

El principal resultado, fue la codificación de la API, que funciona como interfaz entre una base de datos y los clientes que requieran de su servicio. Como se puede observar en la Figura 28, si se intenta conectar al droplet que sirve la API y se le genera una consulta mediante el endpoint “v1/food”, la interfaz devuelve en texto plano la información contenida dentro de la base de datos, específicamente en la tabla Food.

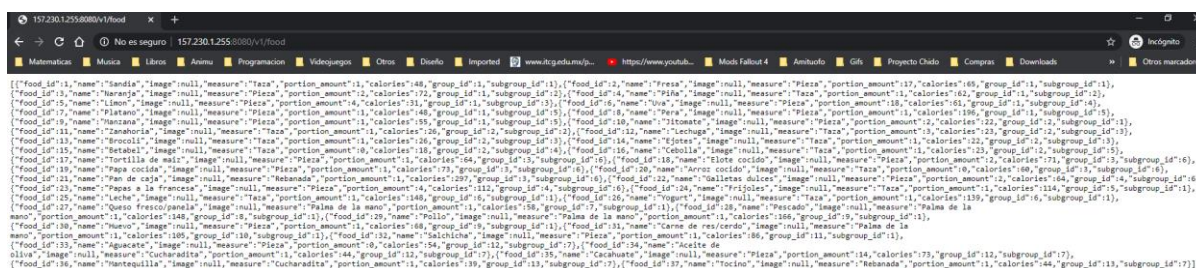


Figura 28 Consulta desde navegador a la API

En la Figura 29, se puede notar en el modo de desarrollador de Google Chrome, como la información es devuelta utilizando el formato JSON. En la Figura 30, se observa el resultado desde el visor de JSON de Google Chrome.

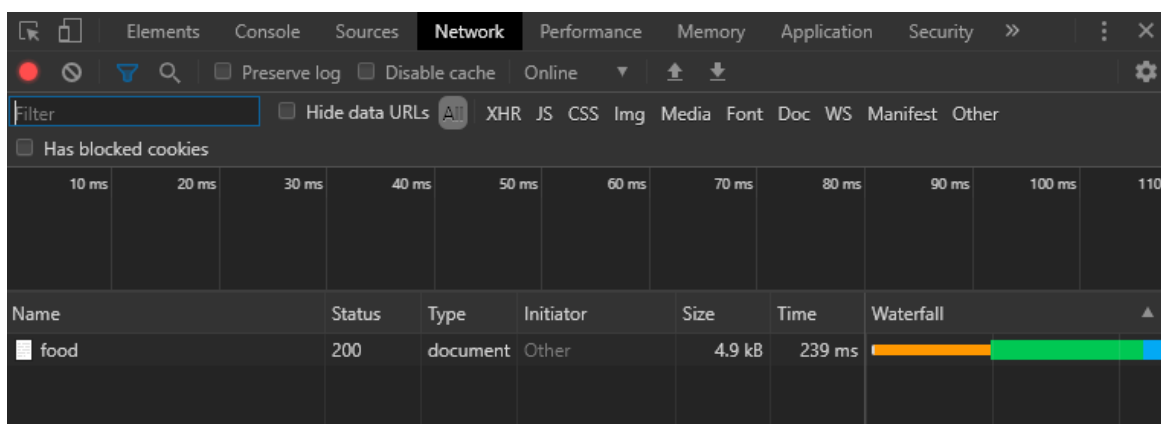


Figura 29 Diagrama de cascada con los archivos recibidos por HTTP desde el servidor

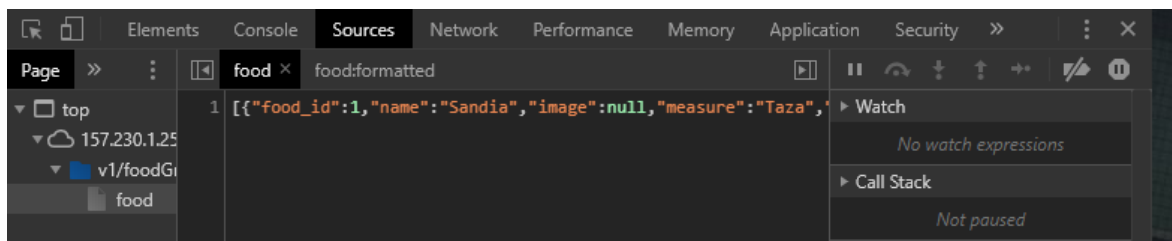


Figura 30 Visor del JSON recibido desde el servidor

Como se puede apreciar en las Figuras 31 y 32, se hicieron varias pruebas en el servidor de producción utilizando una herramienta de desarrollo de APIs conocida como Postman. Las pruebas a diferentes endpoints devolvieron la información pertinente.

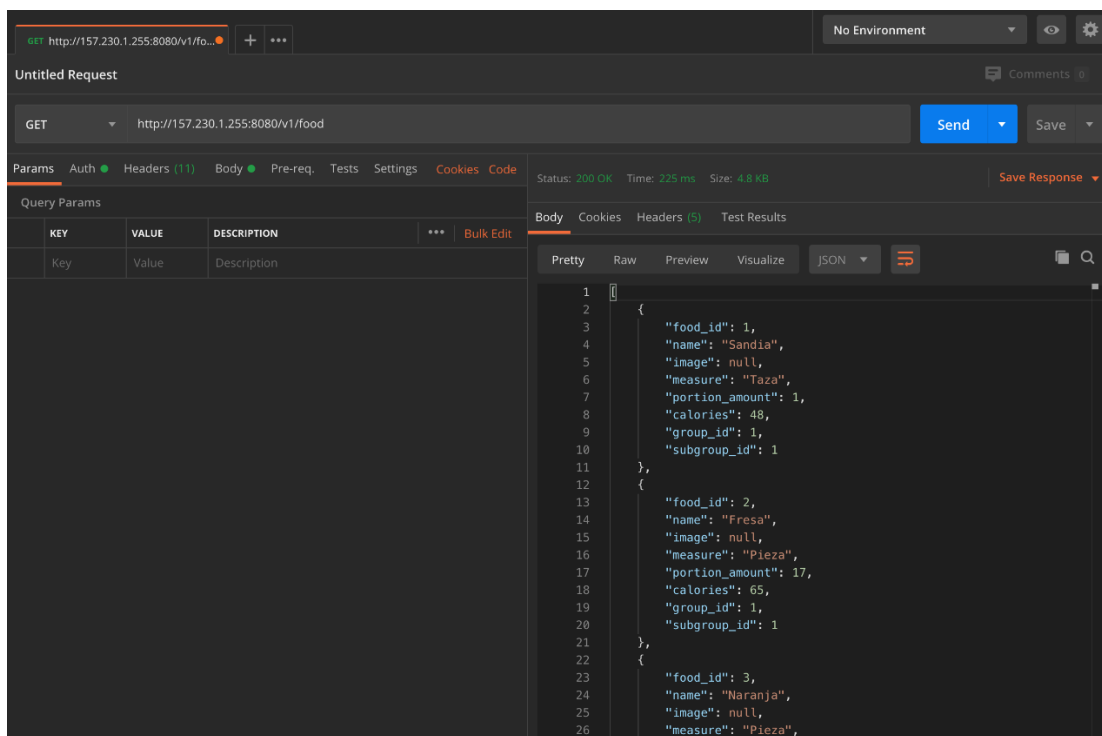


Figura 31 JSON recibido desde el servidor en Postman

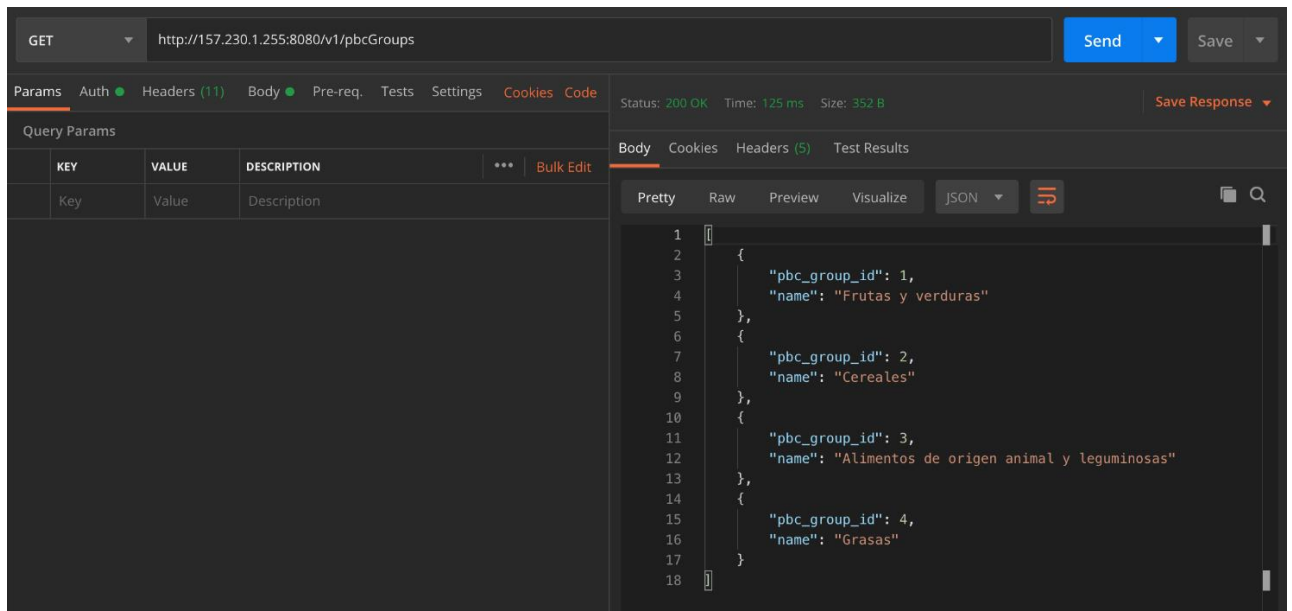


Figura 32 JSON recibido desde el servidor en Postman, utilizando otro endpoint

Para probar la interfaz de la API con Unity, se utilizó un proyecto de prueba, el cual contiene un objeto en la escena llamado API Client. En la Figura 33 se aprecia el visor de la jerarquía del proyecto, con el objeto API Client dentro de la escena por defecto.

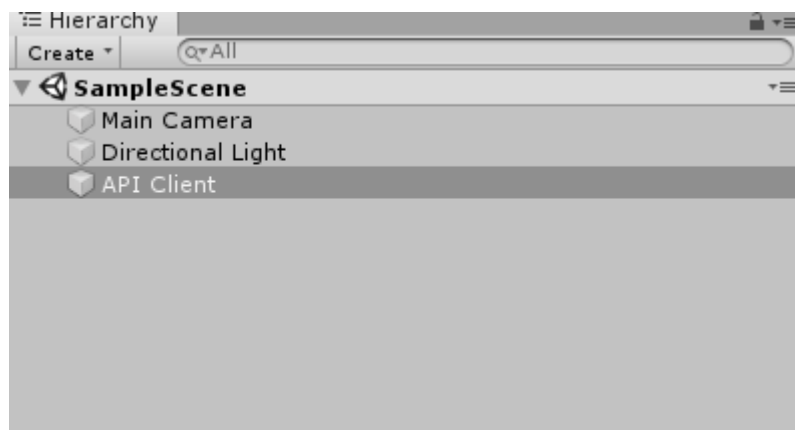


Figura 33 Visor de jerarquía del proyecto de prueba

Este objeto, posee conectado un Script de prueba, el cuál utiliza la librería previamente creada, como se muestra en el inspector del motor de la Figura 34.

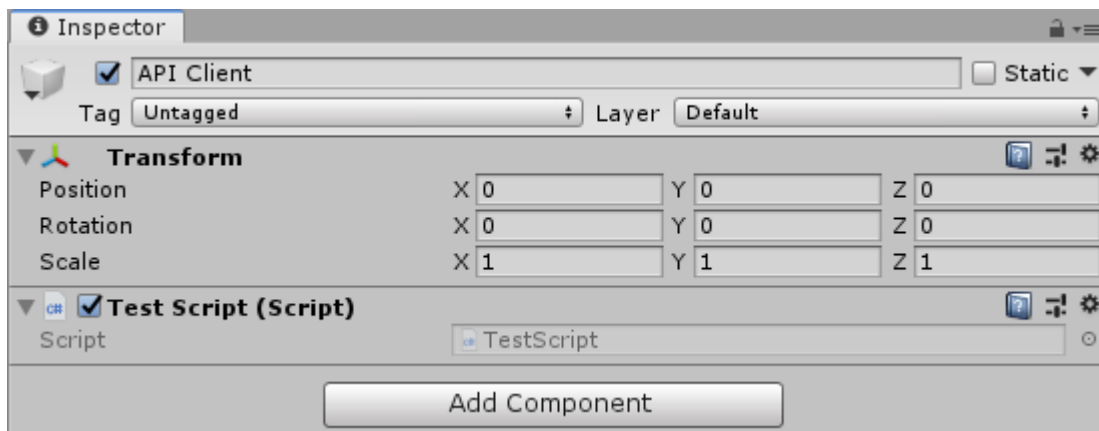


Figura 34 Inspector del objeto API Client de prueba

El archivo TestScript, contiene un método que se ejecuta antes de la primera actualización de frame, por lo que se manda llamar una sola vez. Dentro de este método llamado Start(), se crea una lista de alimentos utilizando la clase APIConnection y se itera dentro de ella en un ciclo for each, el cual imprime el id del alimento y su nombre, como podemos observar en la Figura 35.

```

0 referencias
public class TestScript : MonoBehaviour
{
    // Start is called before the first frame update
    0 referencias
    void Start()
    {
        List<Food> list = APIConnection.GetAllFood();
        foreach(Food food in list)
        {
            Debug.Log(food.food_id);
            Debug.Log(food.name);
        }
    }
}

```

Figura 35 Estructura del archivo TestScript

Como se puede apreciar en la Figura 36, en el momento de ejecutar el motor gráfico, se imprime la información contenida en la API.



Figura 36 Salida de la ejecución de TestScript

Después de finalizar las pruebas y tener seguro que la interfaz funciona sin problemas en Unity, se pasó hacia otro compañero para utilizarla en el desarrollo de los videojuegos.

6. Conclusiones

En este apartado, se presentarán las conclusiones del informe en base al aprendizaje adquirido y al trabajo realizado.

Personalmente, puedo concluir que el proyecto de residencias me otorgó mucho conocimiento y adquirí competencias importantes para en un futuro, poder integrarme más fácilmente al ambiente laboral.

Una de las principales habilidades que adquirí, fue la de diseñar la aplicación en base a múltiples consideraciones y tomando en cuenta una cantidad bastante variada de requerimientos, lo cual me dio mucha experiencia en arquitectura de software.

Otra habilidad muy importante, fue la de dirigir un equipo. A pesar de haber participado como líder de proyecto en actividades durante la universidad, en un entorno serio es muy diferente y requiere practicar mucho las conocidas como “soft skills”, me parece que fue un muy buen ejercicio para entender cómo tratar con el resto de los desarrolladores y con la asesora.

Agradezco mucho al Instituto Tecnológico de Ciudad Guzmán por otorgarme la oportunidad de desarrollar el proyecto, puesto que el desarrollo de videojuegos es una de mis grandes pasiones y un camino que me interesa mucho para mi carrera profesional.

A pesar de la situación que sufrimos con la pandemia de COVID-19 y la dificultad para organizar reuniones con el resto del equipo, me parece que tuvimos buenos resultados y sentamos las bases para extender el proyecto de manera muy fácil y rápida, por lo que concluyo que las residencias fueron un éxito.

7. Competencias desarrolladas

En este apartado se muestran las competencias y habilidades obtenidas y aplicadas durante el desarrollo del proyecto.

Competencias Instrumentales

Son las competencias que funcionan como instrumentos para lograr algún objetivo. Las siguientes competencias instrumentales fueron las necesarias para que el residente realizara el proyecto.

- Habilidades de gestión de información, utilizada con el objetivo de controlar, almacenar y recuperar información de forma eficaz y productiva, proporcionando una mayor seguridad para la toma de decisiones.
- Solución de problemas, ya que en cualquier proyecto los problemas surgirán, aunque se haya hecho el análisis, la planeación y la modelación de manera correcta. Dicha habilidad fue necesaria para tener la capacidad de analizar y solucionar problemas de manera eficaz.
- Toma de decisiones, siendo esta habilidad la que permite elegir la mejor entre varias opciones, haciendo uso de diferentes técnicas para ello.

Competencias Interpersonales

Son las competencias que incluyen las competencias para el trabajo en equipo y las relativas al compromiso con el trabajo.

- Trabajo en equipo, ya que, por la naturaleza del proyecto, el software se dividió en módulos, siendo necesario el trabajo en equipo, desarrollando la habilidad de tratar con personas para lograr una serie de metas específicas, haciendo que la buena comunicación entre los integrantes del equipo y la toma de decisiones grupales algo de vital importancia.
- Habilidad de trabajar en un ambiente laboral: la realización de este proyecto proporcionó una experiencia muy cercana a la laboral por lo cual fue necesario desarrollar esta habilidad, permitiendo una adaptación más natural en futuros proyectos.
- Compromiso ético: esta habilidad fue utilizada para respetar los compromisos con la asesora interna y los trabajos realizados por otras personas.

Competencias Sistemáticas

Estas hacen referencia a la integración de capacidades cognitivas, destrezas prácticas y disposiciones.

- Habilidades de investigación: esta habilidad permitió una forma más eficaz de obtener, verificar y utilizar información.
- Capacidad de generar nuevas ideas: esta habilidad permitió generar las ideas necesarias para desarrollar el proyecto, diseñar su funcionalidad, interfaz gráfica y la solución de problemas.

Competencias por asignaturas

- Estructura de datos: manejo de estructuras de datos en la programación de la página Web.
- Programación Orientada a Objetos: Base teórica para el diseño de la arquitectura de software

- Fundamentos de bases de datos: base teórica para normalización de la base de datos.
- Taller de base de datos: programación de las sentencias SQL para los movimientos en la base de datos, así como configuración del gestor de base de datos.
- Administración de base de datos: configuración de base de datos, simplificación y uso correcto de sentencias SQL.
- Fundamentos de Ingeniería de Software: base teórica para la definición de requerimientos del sistema.
- Administración de Redes: base teórica para los deployments y la administración del servidor principal.
- Gestión de proyectos de software: base teórica del ciclo de desarrollo de un software.
- Ingeniería de Software: correcta ejecución del desarrollo del software, así como la implementación de pruebas y trato con el cliente.
- Programación Web: fundamento teórico del funcionamiento de la API

8. Fuentes de información

Amazon Web Services, Inc. (s.f.). *Microservicios*. Recuperado el 03 de Marzo de 2020, de Amazon Web Services: <https://aws.amazon.com/es/microservices/>

Bycer, J. (2017). *The 3 Essential Elements of Metroidvania Design*. Recuperado el 03 de Marzo de 2020, de Game Wisdom: <http://game-wisdom.com/critical/metroidvania-design>

Fenlon, W. (2017). *How to design a great Metroidvania map*. Recuperado el 03 de Marzo de 2020, de PC Gamer: <https://www.pcgamer.com/how-to-design-a-great-metroidvania-map/>

IntelliJ IDEA. (s.f.). Recuperado el 03 de Marzo de 2020, de JetBrains: <https://www.jetbrains.com/es-es/idea/>

Microsoft. (30 de Octubre de 2020). *Estilo de arquitectura de microservicios*. Recuperado el 03 de Marzo de 2020, de Microsoft Azure Docs: <https://docs.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices>

Pearson, K. (2013). *Guide to Making Metroidvania Style Games: Part 1*. Recuperado el 03 de Marzo de 2020, de Subtractive design Blogspot: <http://subtractivedesign.blogspot.com/2013/01/guide-to-making-metroidvania-style.html>

Pearson, K. (2013). *Guide to Making Metroidvania Style Games: Part 2ish!* Recuperado el 03 de Marzo de 2020, de Subtractive design Blogspot: http://subtractivedesign.blogspot.com/2013/01/guide-to-making-metroidvania-style_16.html

Pivotal Software. (s.f.). *Spring Boot*. Recuperado el 03 de Marzo de 2020, de Spring: <https://spring.io/projects/spring-boot>

Pivotal Software. (s.f.). *Why Spring?* Recuperado el 03 de Marzo de 2020, de Spring: <https://spring.io/why-spring>

Proceso y Roles de Scrum. (s.f.). Recuperado el 03 de Marzo de 2020, de Softeng:

<https://www.softeng.es/es-es/empresa/metodologias-de-trabajo/metodologia-scrum/proceso-roles-de-scrum.html>

Qué es Scrum. (s.f.). Recuperado el 2020 de Marzo de 03, de Proyectos Ágiles:

<https://proyectosagiles.org/que-es-scrum/>

Unity Technologies. (s.f.). *Unity*. Recuperado el 03 de Marzo de 2020, de Unity:

<https://unity.com/products/core-platform>

Wikipedia Community. (s.f.). *DigitalOcean*. Recuperado el 03 de Marzo de 2020, de

Wikipedia: <https://en.wikipedia.org/wiki/DigitalOcean>

Wikipedia Community. (s.f.). *IntelliJ IDEA*. Recuperado el 03 de Marzo de 2020, de

Wikipedia: https://es.wikipedia.org/wiki/IntelliJ_IDEA