



*Centres universitaris adscrits a la*



**Grado en Ingeniería Informática de Gestión y Sistemas de Información**

**Desarrollo de un backend para la creación de videojuegos**

**Memoria**

**Miquel Navarro Fernández**

**Tutor: Dr. David Ródenas Picó**

**Curso 2020-2021**



## **Agradecimientos**

En primer lugar, agradezco a mi tutor, David Ródenas, por sus comentarios y sugerencias para incrementar el valor del proyecto.

También me gustaría dar las gracias a Montse, mi pareja, por su apoyo y su paciencia ante mis cambios de humor durante todo el proceso de desarrollo.

## **Abstract**

The objective of this project is to develop a scalable, secure and flexible backend, aimed at video game companies. After an analysis of options, it has been decided to create the infrastructure on Amazon Web Services. The backend has been developed simultaneously with an application in Unity, responsible for consuming it and demonstrating its capabilities. Finally, the product has been created with user management and cloud storage, using access control services, REST API or serverless code execution.

## **Resum**

L'objectiu d'aquest projecte és desenvolupar un backend escalable, segur i flexible, orientat a empreses de videojocs. Després d'una anàlisi d'opcions, s'ha decidit crear la infraestructura a Amazon Web Services. El backend s'ha desenvolupat simultàniament amb una aplicació a Unity, encarregada de consumir-lo i demostrar les seves capacitats. Finalment s'ha creat el producte amb gestió d'usuaris i emmagatzematge en el núvol, utilitzant serveis de control d'accés, API REST o execució de codi sense servidor.

## **Resumen**

El objetivo de este proyecto es desarrollar un backend escalable, seguro y flexible, orientado a empresas de videojuegos. Tras un análisis de opciones, se ha decidido crear la infraestructura en Amazon Web Services. El backend se ha desarrollado simultáneamente con una aplicación en Unity, encargada de consumirlo y demostrar sus capacidades. Finalmente se ha creado el producto con gestión de usuarios y almacenamiento en la nube, utilizando servicios de control de acceso, API REST o ejecución de código sin servidor.



## Índice

Glosario de términos.....	XI
1. Introducción .....	1
2. Marco teórico y análisis de referentes.....	3
2.1. Contexto.....	3
2.1.1. Estudio de tecnologías y servicios.....	4
2.2. Antecedentes .....	8
2.2.1. Media Molecule.....	8
2.2.2. Ubisoft .....	9
2.2.3. Halfbrick.....	10
2.2.4. Síntesis de los antecedentes .....	11
2.3. Necesidades de información .....	11
3. Objetivos y alcance .....	13
4. Metodología .....	15
4.1. Iteraciones.....	17
4.1.1. Primera iteración.....	17
4.1.2. Segunda iteración .....	17
4.1.3. Tercera iteración .....	17
4.1.4. Cuarta iteración.....	18
4.1.5. Quinta iteración .....	18
4.1.6. Sexta iteración .....	18

4.1.7.	Séptima iteración .....	18
5.	Desarrollo .....	19
5.1.	Definición de requerimientos funcionales y tecnológicos .....	19
5.2.	Especificación .....	20
5.2.1.	Casos de uso .....	20
5.2.2.	API.....	25
5.3.	Diseño .....	27
5.3.1.	Arquitectura backend.....	28
5.3.2.	Arquitectura frontend .....	29
5.3.3.	Diagrama de navegación .....	30
5.4.	Implementación .....	33
5.4.1.	Elección de plataforma .....	33
5.4.2.	Creación de cuenta.....	34
5.4.3.	Servicios .....	35
5.4.4.	Preparación de Unity .....	38
5.4.5.	Registro.....	39
5.4.6.	Inicio de sesión .....	40
5.4.7.	Guardado en la nube .....	42
5.4.8.	API.....	44
5.4.9.	Cambio de contraseña.....	45
5.4.10.	Recuperación de contraseña.....	46

---

5.4.11.	Conexión entre capas .....	47
5.4.12.	Escalabilidad .....	52
5.5.	Análisis de las iteraciones .....	54
5.5.1.	Primera iteración.....	54
5.5.2.	Segunda iteración .....	54
5.5.3.	Tercera iteración .....	55
5.5.4.	Cuarta iteración.....	55
5.5.5.	Quinta iteración .....	56
5.5.6.	Sexta iteración .....	57
5.5.7.	Séptima iteración .....	57
6.	Conclusiones .....	59
7.	Posibles ampliaciones .....	61
8.	Bibliografía.....	63





## Índice de figuras

Figura 1. Logo de Media Molecule. Fuente: [14].....	9
Figura 2. Logo de Ubisoft. Fuente: [16].....	10
Figura 3. Logo de Halfbrick. Fuente: [18] .....	11
Figura 4. Fases del modelo espiral. Fuente: [19].....	16
Figura 5. Iconos utilizados para representar a los actores del sistema. Fuente: Elaboración propia.....	20
Figura 6. Diagrama de casos de uso de autenticación. Fuente: Elaboración propia.....	21
Figura 7. Diagrama de casos de uso de guardado en la nube. Fuente: Elaboración propia.....	21
Figura 8. Diseño general del sistema. Fuente: Elaboración propia .....	27
Figura 9. Infraestructura del backend diseñado en capas. Fuente: Elaboración propia.....	28
Figura 10. Ejemplo de la arquitectura utilizada en el frontend. Fuente: Elaboración propia .....	29
Figura 11. Diagrama de navegación de la aplicación. Fuente: Elaboración propia .....	30
Figura 12. Esqueleto de la pantalla Main Menu. Fuente: Elaboración propia .....	30
Figura 13. Esqueleto de la pantalla Load World Menu. Fuente: Elaboración propia .....	31
Figura 14. Esqueleto de la pantalla Sign In. Fuente: Elaboración propia .....	31
Figura 15. Esqueleto de la pantalla Sign Up. Fuente: Elaboración propia.....	32
Figura 16. Esqueleto de las pantallas My Account y Change Password. Fuente: Elaboración propia.....	32
Figura 17. Esqueleto de las pantallas Forgot Password y Confirm Forgot Password. Fuente: Elaboración propia.....	32

Figura 18. Configuración de API Gateway de game-pcg-api y company-auth-api. Fuente: Elaboración propia.....	37
Figura 19. Formulario de registro desde Unity. Fuente: Elaboración propia .....	39
Figura 20. Formulario de inicio de sesión desde Unity. Fuente: Elaboración propia .....	41
Figura 21. Menú para gestionar los mundos en Unity. Fuente: Elaboración propia .....	43
Figura 22. Menú “My Account” y formulario de cambio de contraseña. Fuente: Elaboración propia.....	46
Figura 23. Formularios (paso 1 y 2) para la recuperación de contraseña. Fuente: Elaboración propia.....	47
Figura 24. Petición de inicio de sesión. Fuente: Elaboración propia .....	47
Figura 25. Inicialización de la Lambda <i>company-auth-lambda</i> . Fuente: Elaboración propia .....	48
Figura 26. Función de Sign In en la Lambda <i>company-auth-lambda</i> . Fuente: Elaboración propia.....	48
Figura 27. Función para obtener el nombre de usuario en la Lambda <i>company-auth-lambda</i> . Fuente: Elaboración propia.....	49
Figura 28. Gestión de la respuesta de inicio de sesión en la aplicación. Fuente: Elaboración propia.....	49
Figura 29. Petición para obtener un mundo. Fuente: Elaboración propia .....	50
Figura 30. Inicialización de la Lambda <i>game-pcg-worlds-lambda</i> . Fuente: Elaboración propia.....	50
Figura 31. Función para obtener el prefijo de un archivo de S3 en la Lambda <i>game-pcg-worlds-lambda</i> . Fuente: Elaboración propia .....	50
Figura 32. Función para obtener un mundo en la Lambda <i>game-pcg-worlds-lambda</i> . Fuente: Elaboración propia.....	51

Figura 33. Gestión de la respuesta de obtención del mundo en la aplicación. Fuente: Elaboración propia.....	51
--	----



## Índice de tablas

Tabla 1. Comparativa de las diferentes plataformas respecto a las necesidades principales del proyecto. Fuente: Elaboración propia.....	4
Tabla 2. Resultados de la prueba con Firebase Predictions. Fuente: [17] .....	10
Tabla 3. Síntesis de los antecedentes. Fuente: Elaboración propia .....	11
Tabla 4. Ventajas y desventajas del modelo espiral. Fuente: [20] .....	16
Tabla 5. Especificación de la API de gestión de usuarios y autenticación. Fuente: Elaboración propia.....	26
Tabla 6. Especificación de la API de almacenamiento en la nube. Fuente: Elaboración propia .....	27
Tabla 7. Evaluación de Azure y AWS. Fuente: Elaboración propia .....	33
Tabla 8. Decisión tomada respecto a cada plataforma. Fuente: Elaboración propia.....	34



## Glosario de términos

API	Application Programming Interface. Conjunto de funciones diseñadas para ser utilizadas por otro software y añadirle una capa de abstracción.
AWS	Amazon Web Services. Plataforma en la nube de Amazon que ofrece gran cantidad y variedad de servicios.
Azure	Abreviatura de Microsoft Azure. Plataforma en la nube de Microsoft que ofrece gran cantidad y variedad de servicios.
Endpoint	Punto de enlace. En el caso de API Gateway, hace referencia a la URL utilizada para invocar una función.
Espiral	Iteración del modelo espiral. Se usa fase o etapa como su sinónimo.
REST	Representational State Transfer. Arquitectura de software para sistemas hipermedia distribuidos.
SDK	Software Development Kit. Conjunto de herramientas que permite desarrollar aplicaciones para una plataforma específica.
TI	Tecnología de la Información.
URL	Uniform Resource Locator. Identificador uniforme que hace referencia a recursos que pueden variar con el tiempo como, por ejemplo, webs.





# 1. Introducción

Durante los últimos años se han producido casos de videojuegos con problemas en las primeras semanas desde su lanzamiento o después de alguna actualización. Algunos de estos problemas son derivados de su backend, como servidores que no pueden soportar tantos jugadores [1] o errores que provocan que los jugadores pierdan sus partidas guardadas [2].

El objetivo de este proyecto es plantear y desarrollar un backend genérico orientado a empresas de videojuegos, centrándose en los servicios que se suelen usar en esta industria. Los productos que crean estas empresas pueden tener gran cantidad de usuarios concurrentes en momentos concretos como el lanzamiento, para reducirse en gran medida un tiempo después. Por esta razón, todo este sistema a desarrollar debe ser fácilmente escalable.

La plataforma donde desarrollar esta solución debe adaptarse a las necesidades presentes y futuras de este tipo de empresas, tanto a nivel de posibilidades que ofrece como los costes que supone. Por eso es apropiado iniciar el proceso planteando posibles plataformas sobre las que desarrollar el producto y analizar sus ventajas y desventajas. A partir de este análisis se puede elegir la plataforma que mejor se adapte a las necesidades definidas y, finalmente, desarrollar el producto sobre la opción más conveniente.

Finalmente, la plataforma escogida ha sido Amazon Web Services. De esta plataforma se han utilizado seis servicios: CodeCommit, Cognito, S3, API Gateway, Lambda y CloudWatch. Con la combinación de estos se han podido desarrollar diversas funcionalidades de autenticación y almacenamiento en la nube, que son utilizadas desde una aplicación creada en Unity.

El consumo de los servicios de AWS ha sido implementado en la aplicación y posteriormente migrado a funciones Lambda pasando por una API REST. Este cambio ha incrementado la seguridad del producto y ha reducido el acoplamiento entre el frontend y el backend. Aun así, este proceso se considera innecesario, ya que se podría haber realizado la implementación en funciones Lambda desde el inicio, reduciendo el tiempo total utilizado en el proceso.

El backend desarrollado puede ser consumido desde cualquier motor de juego o aplicación. Tras la migración a las funcionalidades a la API, sólo es necesario realizar peticiones HTTP

para comunicarse con el producto desarrollado. En el proyecto únicamente se ha creado una aplicación en Unity, a modo de ejemplo y validación. Aun así, el backend no dispone de ninguna limitación respecto a la plataforma que desea consumirlo, permitiendo su uso desde múltiples de ellas simultáneamente incluso si utilizan diferentes lenguajes.

En conclusión, Amazon Web Services ha demostrado ser eficaz para desarrollar este proyecto. Los servicios utilizados permiten satisfacer los requerimientos definidos con relativa facilidad, incluso los de baja prioridad. La plataforma ha permitido un desarrollo fluido en el que se han completado todos los objetivos establecidos.

## 2. Marco teórico y análisis de referentes

### 2.1. Contexto

Para empezar el contexto del proyecto es importante definir algunos conceptos. El backend de una aplicación hace referencia a la parte de esta que el usuario no ve [3]. Esto incluye la capa de acceso a datos y la capa lógica o de negocio. Por el contrario, el frontend refiere a la parte visible, la interfaz con la que interactúa el usuario directamente, conocida como capa de presentación. El backend se encarga de la lógica de la aplicación y procesamiento de datos, y provee al frontend de la información requerida para que este la muestre [4].

Una práctica común en las empresas es desplegar su backend en una plataforma en la nube (*cloud platform*). Estas plataformas son entornos que ofrecen servicios de informática en la nube (*cloud computing*): recursos TI distribuidos a través de internet bajo demanda. Estos servicios incluyen, entre otros, cómputo, almacenamiento o bases de datos [5]. El beneficio de estas plataformas y servicios es que eliminan la necesidad de comprar y mantener servidores físicos propios.

Una gran cantidad de empresas de videojuegos crean sus productos utilizando Unity, una plataforma de desarrollo en tiempo real. Es popular como motor para la creación de videojuegos debido a que proporciona ventajas como: la capacidad de desarrollar cualquier tipo de juego, la compilación multiplataforma y una baja barrera de entrada. Al ser un motor de uso tan extendido, se ha decidido utilizar Unity en el proyecto para desarrollar la aplicación que consume el backend. De este modo actuaría como la plataforma en el que un cliente está desarrollando un supuesto videojuego y requiere ciertos servicios en su backend.

Uno de los servicios que se implementan en este backend es la autenticación para acceder a recursos privados de cada usuario. El proceso a usar para autorizar a un usuario se basa en el especificado en la propuesta de estándar RFC 6749 [6]. Como se explica, un usuario que inicie sesión obtiene un token de acceso y otro de actualización. El *access token* sirve para acceder a los recursos, mientras que el *refresh token* permite actualizar el token de acceso cuando este caduque. De este modo el usuario puede mantener su sesión iniciada sin tener que introducir sus credenciales continuamente. Sin embargo, el token de actualización no debe ser usado para acceder al servidor de recursos directamente.

## 2.1.1. Estudio de tecnologías y servicios

### 2.1.1.1. Tecnologías

En función de los requerimientos definidos, se analizan cuatro alternativas respecto a la tecnología a utilizar: un servidor propio, Microsoft Azure, Amazon Web Services y Firebase. En este apartado simplemente se comprueba si ofrecen una solución para las necesidades del proyecto, sin comparar las plataformas entre ellas.

El primer factor analizado es si proporcionan opciones para implementar las diferentes funcionalidades del proyecto. A esto se le suma la necesidad poder consumir los servicios desde Unity, por lo que es requerido un SDK concreto o una manera de acceder a ellos desde el motor de juego. También se considera imprescindible la existencia de documentación para realizar la implementación. Además, se incluye la necesidad de desplegar el proyecto completo en la misma plataforma. Por esta razón se requiere un alojamiento propio para repositorios Git para el control de versiones de los diferentes juegos.

La Tabla 1 muestra las características requeridas y qué plataformas las ofrecen. Las plataformas en la nube indican qué servicio cubre esa característica o si cumple esa característica. La columna de servidor indica si es factible implementar una solución propia en el marco de este proyecto.

<i>Característica</i>	<i>Azure</i>	<i>AWS</i>	<i>Firebase</i>	<i>Servidor</i>
<i>Gestión de usuarios</i>	Active Directory	Cognito	Authentication	✓
<i>Almacenamiento</i>	Blob Storage	S3	Cloud Storage	✓
<i>API</i>	API Management	API Gateway	Management API	✓
<i>Escalabilidad</i>	✓	✓	✓	✗
<i>Seguridad</i>	✓	✓	✓	✗
<i>Soporte para Unity</i>	Usando REST	AWS SDK para .NET	Unity SDK	Usando REST
<i>Documentación</i>	✓	✓	✓	✗
<i>Repositorio Git</i>	Repos	Codecommit	✗	✗

Tabla 1. Comparativa de las diferentes plataformas respecto a las necesidades principales del proyecto.

Fuente: Elaboración propia

### 2.1.1.2. Servicios AWS

Una vez tomada la decisión de utilizar Amazon Web Services, explicada en el apartado 5.4.1, se han estudiado los servicios a utilizar en el proyecto.

#### Cognito

Amazon Cognito es el servicio encargado del control de acceso, registro e inicio de sesión de los usuarios [7]. Ofrece escalado para millones de usuarios y los protege del uso de credenciales vulnerables. El servicio cumple varios requisitos de seguridad y conformidad, permitiendo despreocuparse de las regulaciones en la mayoría de los casos.

Cognito ofrece grupos de usuarios y grupos de identidades. Las identidades proporcionan credenciales temporales para el acceso a servicios de AWS. Los grupos de usuarios, en cambio, permiten el registro e inicio de sesión para acceso a las aplicaciones. Comúnmente solo se utilizan grupos de usuarios, ya que las identidades tienen un uso muy específico que no es necesario en la mayoría de casos.

#### S3

Amazon S3 (Simple Storage Service) es un servicio de almacenamiento de objetos que ofrece gran escalabilidad, disponibilidad de datos, seguridad y rendimiento [8]. Proporciona una durabilidad de 99,999999999 % (11 nueves). El servicio se paga por uso, de manera que no se le cobra al usuario por espacio de almacenamiento sin utilizar.

Los archivos se guardan en S3 junto con sus metadatos en forma de unas entidades llamadas objetos. Estos objetos siempre se encuentran almacenados en contenedores, denominados buckets. S3 utiliza una estructura no jerárquica, pero permite organizar los datos con prefijos. Dos archivos nombrados *folder/file1* y *folder/file2* se encuentran dentro de la “carpeta” *folder/*. Por esta razón un archivo subido a S3 no puede terminar con el carácter /. Combinando el nombre del objeto, el nombre del bucket y la región donde se encuentra, se forma la URL de acceso al objeto.

S3 ofrece dos tipos de replicación: SSR (Same-Region Replication) y CRR (Cross-Region Replication). Ambos tipos permiten la replicación a una cuenta de AWS diferente para, por ejemplo, conseguir seguridad extra ante pérdidas por eliminación accidental.

- **SSR:** replica los objetos en la misma región de AWS. El coste viene dado por el almacenamiento de espacio y las solicitudes PUT utilizadas.
- **CRR:** replica los objetos en diferentes regiones de AWS. Reduce la latencia al permitir acceder al archivo situado en la región más cercana al usuario. También ofrece una seguridad superior al permitir recuperación de desastres. Supone un coste superior a SSR al tener que pagar por la transferencia de datos entre regiones.

## CodeCommit

AWS CodeCommit es un servicio administrado de control de código fuente que aloja repositorios basados en Git [9]. El servicio también ofrece seguridad y escalabilidad. Además, al funcionar con IAM, permite asignar permisos específicos a los usuarios. Del mismo modo que otros alojamientos, posibilita la creación de reglas de aprobación para controlar las *pull requests*.

El servicio ofrece la opción de crear triggers: acciones generadas como respuesta a otras acciones. Esto permite, por ejemplo, detectar cuando se hace un push a una rama concreta y ejecutar una build o enviar una notificación por correo.

## API Gateway

Amazon API Gateway es un servicio que facilita la creación, la publicación, el mantenimiento, el monitoreo y la protección de API a cualquier escala [10]. Las API actúan como la "puerta de entrada" para que las aplicaciones accedan a los datos, la lógica empresarial o la funcionalidad de sus servicios de backend. El servicio se encarga de gestionar la aceptación y procesamiento de cientos de miles de llamadas API simultáneas.

Se pueden crear cuatro tipos de API:

- **HTTP API:** APIs RESTful de baja latencia y coste. Dispone de características integradas como OIDC y OAuth 2. Es la mejor opción para APIs que solo requieren funcionalidad de proxy.
- **WebSocket API:** APIs para comunicación bidireccionales en tiempo real, con conexiones persistentes.

- **REST API:** APIs RESTful con un control total sobre las solicitudes y respuestas. Es la alternativa preferible si se requieren características de administración además de funcionalidades de proxy.
- **REST API Private:** igual a REST API, pero únicamente accesible desde una VPC (Virtual Private Cloud).

## Lambda

AWS Lambda es un servicio de informática sin servidor (*serverless*), que permite ejecutar el código en la nube sin necesidad de proporcionar ni gestionar servidores, administrar tiempos de ejecución o crear la lógica de escalado [11]. El código puede ser activado llamándolo directamente o a través de una gran cantidad de servicios de AWS (como API Gateway, por ejemplo). Cada petición se procesa por separado, ejecutando el código en paralelo. Esto permite escalar de forma dinámica y precisa adaptándose a la carga de trabajo.

Cada instancia de una función Lambda puede ser una copia nueva o reutilizar instancias anteriores para mejorar el desempeño. Por este motivo, las funciones no deben tener estado, ya que no se comparte entre instancias y no se puede asumir que una petición será procesada con los datos un estado concreto. En caso de necesitar información para ejecutar la función, esta puede ser almacenada en otros servicios como S3 o DynamoDB y ser consultada desde Lambda.

AWS Lambda admite de forma nativa siete lenguajes y frameworks: .NET Core (C# y Powershell), Node.js, Java, Python, Go y Ruby. De cada uno de ellos, se encuentran disponibles diferentes versiones de tiempo de ejecución.

## CloudWatch

Amazon CloudWatch es un servicio de monitorización y observación que ofrece datos e información procesable para monitorizar las aplicaciones [12]. El objetivo del servicio es facilitar el análisis del sistema para responder a cambios de rendimiento, optimizar el uso de recursos y mostrar una vista unificada del estado de las operaciones. Para ello, CloudWatch permite detectar comportamientos anómalos, definir alarmas, realizar acciones automatizadas...

CloudWatch Logs permite recopilar y almacenar registros de los recursos, aplicaciones y servicios casi en tiempo real. Esto permite, por ejemplo, visualizar los registros de cada ejecución de una función de Lambda, algo que no sería posible por otros medios. De este modo, se simplifica la detección y corrección de errores en el código ejecutado en ellas.

## **2.2. Antecedentes**

Actualmente la gran mayoría de compañías de videojuegos utilizan algún servicio online para sus productos, desde simples mensajes dentro del juego hasta servidores para partidas multijugador. En este apartado se analizan diferentes casos de empresas que externalizan estos servicios en plataformas en la nube. No se dispone información de las entidades que optan por otro tipo de entornos. De todas maneras, debido a la naturaleza del proyecto, los casos explicados se consideran los más relevantes para este.

Existe una gran cantidad de empresas que optan por la opción de delegar su backend en una plataforma en la nube. Aun así, sólo una pequeña cantidad de ellas se enfocan en el entretenimiento. Por esta razón, se considera como antecedentes principales a aquellas empresas de videojuegos que utilizan alguna de las plataformas mencionadas en el punto 2.3 para desarrollar y/o mantener sus productos.

Las empresas mencionadas a continuación, a pesar de utilizar plataformas en la nube, utilizan sus propias implementaciones y arquitecturas. No se han encontrado soluciones genéricas que utilicen múltiples compañías, y si alguna de ellas dispone de una, no ofrece públicamente esa información.

### **2.2.1. Media Molecule**

Media Molecule es una desarrolladora de videojuegos situada en Reino Unido. Utiliza Amazon Web Services desde 2009 y su CTO, Alex Evans, afirma:

AWS parecía la mejor solución disponible para permitir que una empresa pequeña e independiente a desarrolle y pruebe rápidamente una nueva infraestructura y la aloje. También nos encantó la flexibilidad que nos proporcionaba AWS, al crear entornos de prueba más pequeños, para pruebas beta, control de calidad, localización y durante el desarrollo. El bajo costo inicial también fue crucial. [13]



Evans también explica que los videojuegos suelen tener un gran pico de usuarios en los primeros días desde el lanzamiento. Por ello no pueden construir poco a poco una infraestructura a medida que aumenta la audiencia, sino que deben poder soportar millones de usuarios de un día para otro.

Little Big Planet (LBP), su primer juego, estaba funcionando sobre hardware tradicional. Para Little Big Planet 2 rehicieron los servidores para funcionar en AWS, migraron el primer LBP antes de la salida del segundo. De esta manera, en el momento del lanzamiento, ambos juegos utilizaban el mismo conjunto de servidores en AWS. Estos nuevos servidores soportaron, en su máximo pico, 80000 usuarios concurrentes y 30000 peticiones HTTP por segundo.



Figura 1. Logo de Media Molecule. Fuente: [14]

### 2.2.2. Ubisoft

Ubisoft Entertainment es una gran desarrolladora y distribuidora de videojuegos que utiliza Azure para su *shooter* multijugador Rainbow Six Siege. Benjamin Azoulay, Live Operations Manager de Ubisoft, afirma que el juego fue lanzado para tres plataformas simultáneamente sin tener que gestionar tres bases de códigos gracias a Azure [15].

La empresa decidió confiar a Microsoft tanto el centro de datos como los servidores, capaces de gestionar sus 3 millones de usuarios diarios y los cientos de miles de partidas simultáneas. Al dejar que Azure se encargue de la infraestructura, Ubisoft ha podido centrarse en el desarrollo del juego y sus mejoras continuas.

También destacan la flexibilidad y escalabilidad de los servidores. Microsoft puede proporcionar los servicios con alcance mundial, y los servidores autoescalan en función de la demanda. Por esta razón los jugadores pueden conectarse sin encontrar servidores ocupados o no disponibles y experimentan bajas latencias.



Figura 2. Logo de Ubisoft. Fuente: [16]

### 2.2.3. Halfbrick

Halfbrick Studios es la empresa desarrolladora de juegos famosos como Fruit Ninja o Jetpack Joyride. En este caso, utilizan Firebase para configuración remota y retención de usuarios, difiriendo en uso de los dos antecedentes explicados. Sus juegos han sido creados para dispositivos móviles y no disponen de funciones multijugador, por lo que no requieren de servidores para partidas.

En su juego Dan The Man utilizaban la configuración remota de la plataforma de Google, y en este probaron las funciones de predicción. Se realizó una prueba con tres grupos de usuarios: sin cambios, modelo de predicción propio y Firebase Predictions. El resultado, visible en la Tabla 2, fue que el tercer grupo incrementó su retención de siete días en un 20% respecto a los otros conjuntos de usuarios [17].

Grupo de usuarios	Usuarios activos 1 día	Usuarios activos 7 días
Grupo 0: Sin promoción	59.52%	25.35%
Grupo 1: Premio después del nivel 3 (modelo propio)	59.07%	25.34%
Grupo 2: Predicted Churners (Firebase Predictions)	62.12%	30.24%

Tabla 2. Resultados de la prueba con Firebase Predictions. Fuente: [17]



Figura 3. Logo de Halfbrick. Fuente: [18]

#### 2.2.4. Síntesis de los antecedentes

Empresa	Plataforma	Características
Media Molecule	Amazon Web Services	<ul style="list-style-type: none"><li>• Usado para servidores y centros de datos.</li><li>• Pico de 80000 usuarios y 30000 peticiones HTTP por segundo.</li></ul>
Ubisoft	Microsoft Azure	<ul style="list-style-type: none"><li>• Usado para servidores y centros de datos.</li><li>• 3 millones de usuarios diarios.</li></ul>
Halfbrick	Firebase	<ul style="list-style-type: none"><li>• Usado para configuración remota y retención de usuarios.</li><li>• Incremento de retención de siete días en un 20%.</li></ul>

Tabla 3. Síntesis de los antecedentes. Fuente: Elaboración propia

### 2.3. Necesidades de información

La principal información que requiere este proyecto es qué opciones existen que permitan desarrollar el producto definido y qué características tienen. También se debe analizar los costes que supone cada uno de estos entornos. Respecto a la alternativa de utilizar plataformas en la nube se toman en consideración Amazon Web Services, Microsoft Azure y Google Firebase.

También es necesario consultar la documentación de cada una de las plataformas. De esta manera se puede comprobar si disponen de compatibilidad con Unity, una característica requerida para el desarrollo de este proyecto. Además, la calidad de la documentación también es un factor a tener en cuenta, ya que es la principal fuente de información durante la implementación del backend.

### 3. Objetivos y alcance

El cliente de la infraestructura son las empresas de videojuegos, que utilizarían este backend para desarrollar sus productos. El usuario final, en cambio, son los jugadores que comprenden el juego, ya que son quienes acaban utilizando los servicios que ofrece y beneficiándose de ellos. De esta manera, podemos definir los siguientes objetivos principales:

- Desarrollar un backend con servicios básicos comunes utilizados en videojuegos:
  - Gestión de usuarios: los usuarios pueden registrarse e iniciar sesión.
  - Guardado de datos en la nube: los usuarios pueden almacenar archivos en la nube (en este caso, partidas guardadas o mapas). El objetivo incluye la subida de los archivos y su descarga.

El usuario final no accede a estos servicios directamente, sino que lo hace a través del videojuego comercializado por el cliente. Este objetivo se valida si todas las funcionalidades funcionan correctamente. Para ejecutar estas funciones se utiliza el frontend (un objetivo explicado a continuación), ya que es el encargado de consumir el backend.

- Crear un pequeño frontend en Unity para utilizar los servicios mencionados. Este permite al usuario utilizar cómodamente las funcionalidades ofrecidas por el backend explicado en el primer objetivo. Esta aplicación simula un videojuego real y se comporta como si fuera uno más de la empresa ficticia. Este objetivo tiene dos funciones: actuar como ejemplo en caso de ofrecer el producto a una empresa, y validar el backend desarrollado, ya que es el encargado de consumir sus servicios. Este objetivo se valida al consumir satisfactoriamente todas las funcionalidades del backend, provocando una validación recíproca.
- Ofrecer escalabilidad en el backend desarrollado para permitir decenas de miles de usuarios concurrentes. Además, también debe poder ser consumido por múltiples aplicaciones simultáneamente sin limitar su capacidad de respuesta. El objetivo se considera completado si la infraestructura creada dispone de métodos para incrementar su límite de carga de trabajo en caso de ser necesario.

También podemos definir algunos objetivos secundarios:

- Crear APIs para acceder a los servicios para incrementar la seguridad y el nivel de abstracción. A pesar de que el usuario común no aprecia ninguna diferencia, beneficia al cliente al facilitar el desarrollo del videojuego y mejorar la protección de sus datos e información sensible. Este objetivo es validado si todas las funcionalidades pueden ser utilizadas realizando peticiones a la API en lugar de consumir los servicios directamente desde la aplicación.
- Añadir las opciones de cambiar y recuperar contraseña. El usuario puede cambiar la contraseña utilizando su contraseña anterior y recuperar una contraseña olvidada usando su correo electrónico. El objetivo queda validado al poder realizar satisfactoriamente ambas funcionalidades.
- Generar archivos desde Unity para ser almacenados en la nube. Estos ficheros pueden tener cualquier extensión y contener cualquier tipo de información. El objetivo se cumple satisfactoriamente al disponer de un archivo en el dispositivo que se haya generado desde el motor.

Las siguientes características no forman parte de los objetivos del proyecto:

- Desarrollar un videojuego que consuma el backend. El frontend desarrollado simula un videojuego y actúa como prueba de concepto, pero no implica crear un videojuego completo y funcional.
- Permitir a los usuarios acceso directo a sus datos. Todas las gestiones que un usuario puede realizar están disponibles en la aplicación, en ningún momento se le va a ofrecer acceso directo a los servicios. En caso de querer eliminar completamente sus datos del sistema debe contactar con un administrador que realizará las acciones necesarias.
- Generar múltiples tipos de archivos. El archivo generado puede ser de cualquier tipo, pero no se considera necesario crear ficheros con extensiones diferentes.

## 4. Metodología

En este proyecto se utiliza el proceso de software conocido como “modelo espiral” [19]. Es categorizado como un modelo *risk-driven*, ya que todo desarrollo se guía por los riesgos implicados. Cada iteración de este proceso se llama espiral, que se divide en cuatro fases y termina con un incremento del producto. El modelo espiral es una metodología adecuada para el proyecto, debido a que:

- Un error puede suponer un gran riesgo como no llegar a cumplir todos los objetivos o, en el peor de los casos, no terminar el proyecto a tiempo. Por esta razón, planificar y analizar los riesgos de cada espiral minimiza las probabilidades de un error sin solución.
- La duración de cada espiral es indeterminada. Al utilizar una tecnología desconocida hasta el momento, las estimaciones de tiempo de este proyecto son especulaciones. En este caso, es preferible planificar en base a los riesgos de cada funcionalidad en vez de basarse en la duración de su desarrollo como hacen otras metodologías.

Cada espiral conta de las siguientes etapas (visualizadas en la Figura 4):

1. **Planificación:** se inicia con la identificación de los objetivos y las alternativas para conseguirlos, ya sean diferentes diseños o la opción de reutilizar elementos. También se debe identificar las restricciones de su implementación, como el tiempo o el coste de desarrollo de esa espiral.
2. **Análisis de riesgo:** evaluar las alternativas identificadas y analizar los riesgos que suponen para formular un plan que resuelva estos factores de riesgo. Esta fase incluye crear prototipos, realizar simulaciones, modelos analíticos...
3. **Desarrollo:** en esta etapa se desarrolla y valida el software en base a los objetivos definidos. La validación incluye revisar el cumplimiento de todos los requerimientos de la espiral y el *testing* del software desarrollado.
4. **Evaluación:** evaluar la espiral que finaliza y planificar la iteración siguiente. Se analiza si se han solucionado los riesgos identificados y aporta información para ayudar a las siguientes planificaciones.

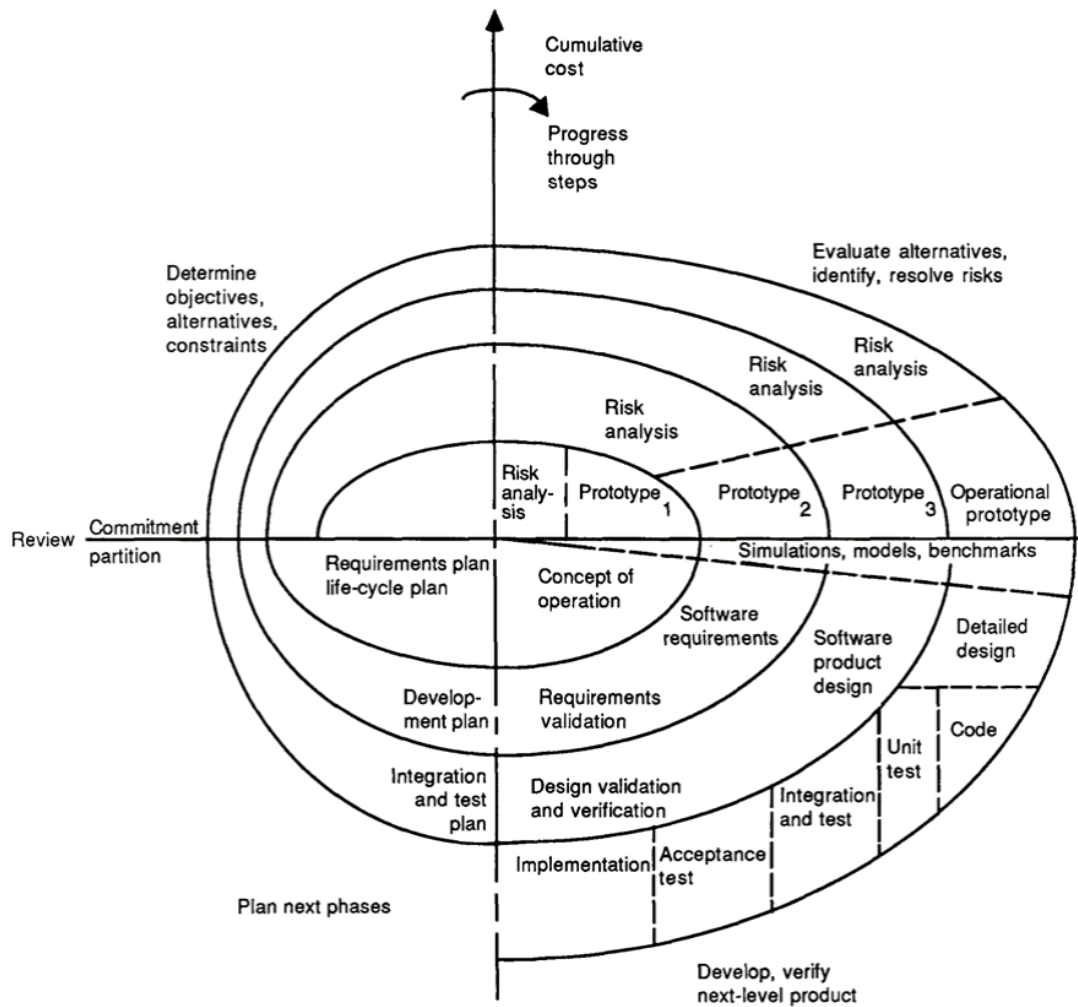


Figura 4. Fases del modelo espiral. Fuente: [19]

Finalmente se pueden concluir que este proceso de software tiene las siguientes ventajas y desventajas:

Ventajas	Desventajas
Los factores de riesgo son reducidos.	La duración de la ejecución no es concreta.
El desarrollo es iterativo y se pueden incorporar funcionalidades progresivamente.	Fallos en el análisis de riesgos podría influir negativamente a todo el proyecto.

Tabla 4. Ventajas y desventajas del modelo espiral. Fuente: [20]



## **4.1. Iteraciones**

Cada iteración del proyecto corresponde a una espiral, y todo el desarrollo está dividido en estas espirales. Las tareas de cada iteración incluyen la investigación y documentación, y no solo la creación del producto final. Se asume que la redacción de la documentación se realiza en todas las fases, donde se añade o modifica la información correspondiente. Este apartado explica las diferentes etapas a partir del momento de entregar el anteproyecto hasta la entrega de la memoria final.

### **4.1.1. Primera iteración**

La espiral inicial tras completar el primer hito (entrega del anteproyecto). En esta fase se investiga la información mencionada en el apartado 2.3 y se realizan los análisis y comparativas pertinentes. Al finalizar, se decide qué opción tomar para desarrollar el backend, por lo que es la base del resto de etapas. Es la espiral con el mayor factor de riesgo, ya que una mala decisión puede condicionar y dificultar el resto del proyecto.

### **4.1.2. Segunda iteración**

Tras elegir la plataforma en la que desarrollar el backend, se empieza a preparar el entorno con los servicios necesarios. En caso de elegir una plataforma en la nube (y no utilizar servidor propio), esto implica crear la cuenta y configurar los servicios necesarios, además de consultar su documentación. Al no tener una aplicación que consuma estos servicios todavía, se probarán manualmente desde la propia plataforma. Una vez terminado, el backend debe estar preparado para, al menos, poder registrar usuarios, iniciar sesión y almacenar archivos.

### **4.1.3. Tercera iteración**

En la tercera fase se empieza a desarrollar el frontend en Unity. Se crea un formulario simple para registrarse y otro para iniciar sesión. Para poder implementar las funcionalidades, se deben buscar e importar los paquetes necesarios en Unity para consumir los servicios. Al finalizar, ambas funcionalidades deben funcionar correctamente. Es una espiral de riesgo moderado ya que, si no se prepara bien el entorno o no se eligen los paquetes correctos, puede dificultar el resto del proyecto.

#### **4.1.4. Cuarta iteración**

Implementar la subida y descarga de archivos para su almacenamiento en el backend. Previamente debemos completar el objetivo de generar un archivo en la aplicación para poder utilizar esta funcionalidad. El acceso al servicio realiza directamente desde la aplicación, sin pasar por un intermediario. Al terminar la etapa, se obtiene una primera versión de los objetivos principales y se consigue el hito de la memoria intermedia.

#### **4.1.5. Quinta iteración**

Migrar la subida de ficheros a una API REST. Esto implica:

- Establecer la puerta de entrada y obtener una URL para acceder a las funciones.
- Crear las funciones que permitan la subida y acceso a los archivos.
- Modificar el frontend para consumir la API creada en vez de acceder directamente al servicio.

Esta espiral implica cierto riesgo por los diferentes pasos que supone, que pueden complicarse o requerir mucho tiempo. Al completarse, se ha logrado otro de los objetivos secundarios.

#### **4.1.6. Sexta iteración**

Implementar la opción de cambio y recuperación de contraseña. Se deben crear pequeños formularios para ambas funcionalidades. Es el objetivo menos prioritario, pero debe poder cumplirse sin inconvenientes si las espirales anteriores se completan de la forma esperada.

#### **4.1.7. Séptima iteración**

Espiral final, termina al entregar la memoria final. En esta fase se termina de pulir y revisar la documentación. También se añaden a la memoria los apartados de conclusiones y posibles mejoras.

## 5. Desarrollo

### 5.1. Definición de requerimientos funcionales y tecnológicos

Se han dividido los requerimientos en alta y baja prioridad. El primer caso son requerimientos que el producto debe cumplir obligatoriamente, ya sea porque son necesarios para cumplir los objetivos o por una imposición externa, como el uso de ciertos lenguajes o los aspectos legales. El segundo caso son aquellos requerimientos de menor relevancia, que aumentan la calidad del producto o podrían implementarse posteriormente.

#### Funcionales de alta prioridad:

- Poder registrarse e iniciar sesión desde la interfaz de Unity, de manera que los archivos queden vinculados a la cuenta correspondiente.
- Poder subir su archivo de guardado y acceder a él desde Unity. Este almacenamiento tiene una disponibilidad inmediata a nivel mundial a través de internet si se ha iniciado sesión con la cuenta propietaria.
- Permitir al cliente eliminar los datos de un usuario en caso de que lo solicite.

#### Funcionales de baja prioridad:

- Permitir al cliente el acceso directo a los archivos de guardado de todos los usuarios, de manera que pueda dar soporte a los jugadores o gestionar los datos.

#### Tecnológicos de alta prioridad:

- Utilizar C# como lenguaje para el frontend en Unity. En el caso de las APIs el lenguaje depende de las opciones proporcionadas por la plataforma.
- Ofrecer una autenticación segura y un almacenamiento durable de alta disponibilidad.
- Permitir almacenar cualquier formato de archivo.
- Velar por la privacidad y seguridad de los ficheros, de manera que no puedan ser robados ni se pierdan.

### Tecnológicos de baja prioridad:

- Soportar múltiples juegos, por lo que futuros lanzamientos de la empresa podrían utilizarlo sin tener que modificar el primero.
- Poder utilizar el backend desde otros motores aparte de Unity, como Unreal Engine.

## 5.2. Especificación

### 5.2.1. Casos de uso

#### 5.2.1.1. Definición de los actores

Un actor, en un caso de uso, hace referencia a toda entidad externa al sistema que se relaciona con este y le demanda una funcionalidad. En este proyecto se distinguen dos actores diferenciados, explicados a continuación. Estos actores aparecen representados con los iconos mostrados en la Figura 5.

- **User:** actor que utiliza la aplicación sin haber iniciado sesión. Este actor únicamente puede crear una cuenta, autenticarse o recuperar su contraseña si la ha olvidado.
- **Signed User:** actor utiliza la aplicación y ha iniciado sesión. Un Signed User puede cerrar su sesión, modificar sus credenciales o realizar todas las acciones correspondientes al guardado en la nube (listar mundos, descargarlos, eliminarlos...). Iniciar sesión es la precondition para todos los casos de uso cuyo actor sea un Signed User.

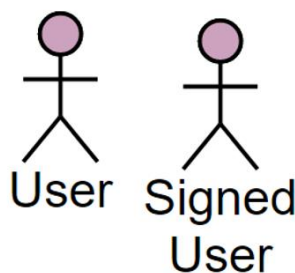


Figura 5. Iconos utilizados para representar a los actores del sistema. Fuente: Elaboración propia

#### 5.2.1.2. Diagramas de casos de uso

En este apartado se muestran los diagramas con las acciones que puede realizar cada actor respecto a cada servicio ofrecido en la aplicación.

## Autenticación

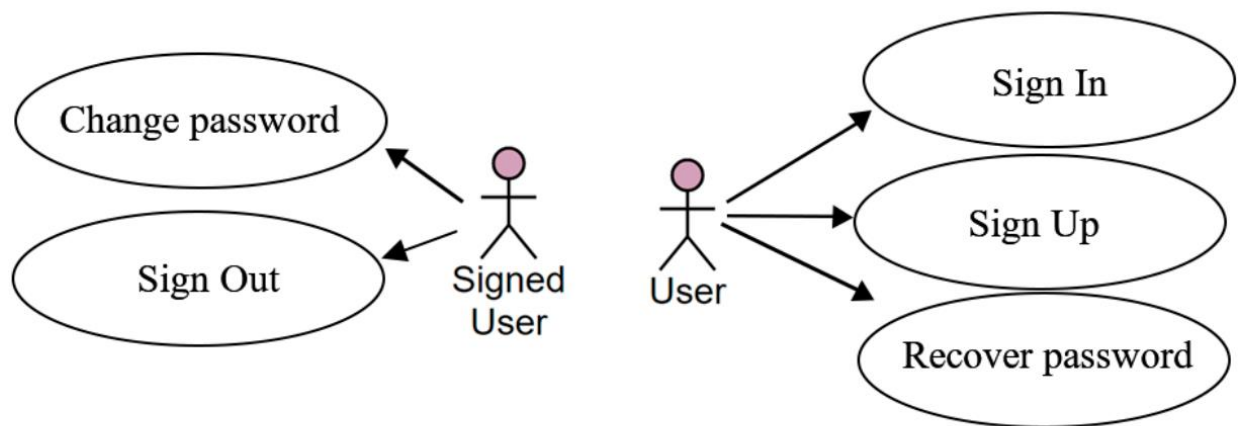


Figura 6. Diagrama de casos de uso de autenticación. Fuente: Elaboración propia

## Guardado en la nube

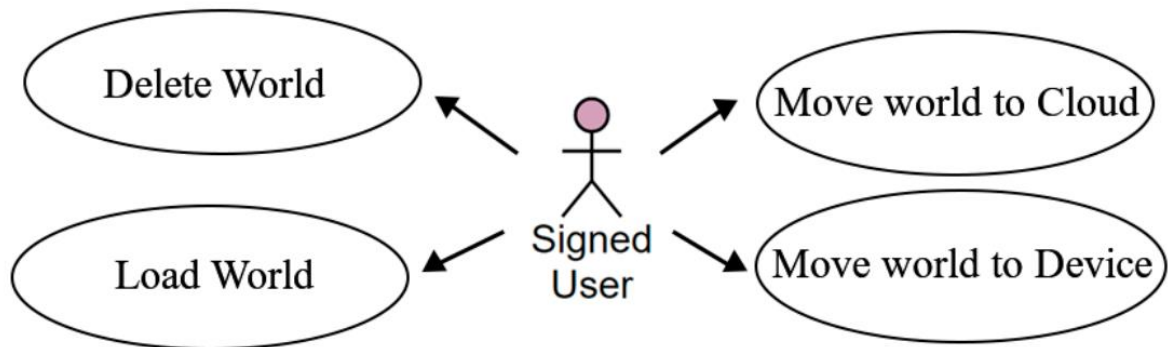


Figura 7. Diagrama de casos de uso de guardado en la nube. Fuente: Elaboración propia

### 5.2.1.3. Descripción de casos de uso

A continuación se describen los casos de uso mostrados en el apartado anterior. En ellos se muestra el actor implicado, la garantía de éxito, el flujo típico de éxito y, si existen, los flujos alternativos.

**Autenticación**

<b>Sign Up</b>	
Actor	User
Garantía de éxito	Se ha creado una cuenta para el usuario y se ha activado.
Flujo típico de éxito	<ol style="list-style-type: none"> <li>1. El actor indica que quiere crear una cuenta e introduce los datos.</li> <li>2. Los datos son validados y se crea la cuenta del usuario. El usuario es informado.</li> <li>3. Se envía un correo de activación de cuenta al email especificado.</li> <li>4. El usuario comprueba su correo electrónico y abre el enlace de activación.</li> </ol>
Flujos alternativos	<ol style="list-style-type: none"> <li>2.a. Los datos no son válidos y no se crea la cuenta. Se informa al usuario y vuelve al paso 1.</li> <li>4.a. El usuario no activa su cuenta mediante el link recibido en su correo. La cuenta está creada pero no activada. Termina el caso de uso.</li> </ol>

<b>Sign In</b>	
Actor	User
Garantía de éxito	Se ha iniciado la sesión del usuario.
Flujo típico de éxito	<ol style="list-style-type: none"> <li>1. El actor indica que quiere iniciar sesión e introduce sus credenciales.</li> <li>2. Los datos son validados y se inicia la sesión. El usuario es informado.</li> </ol>
Flujos alternativos	<ol style="list-style-type: none"> <li>2.a. Los datos no son válidos y no se inicia la sesión. Se informa al usuario y vuelve al paso 1.</li> </ol>

<b>Recover password</b>	
Actor	User
Garantía de éxito	Se ha establecido una nueva contraseña en la cuenta de un usuario que la había olvidado.
Flujo típico de éxito	<ol style="list-style-type: none"> <li>1. El actor indica que ha olvidado su contraseña e introduce su nombre de usuario.</li> <li>2. Se envía un correo con un código de confirmación al email asociado a la cuenta especificada.</li> <li>3. El usuario introduce la nueva contraseña junto con el código de confirmación recibido.</li> <li>4. Los datos son validados y se establece la nueva contraseña. El usuario es informado</li> </ol>
Flujos alternativos	<p>2.a. La cuenta de usuario especificada no existe. El usuario no recibe el código de confirmación. Termina el caso de uso.</p> <p>4.a. Los datos no son válidos y no se establece la nueva contraseña. Se informa al usuario y vuelve al paso 3.</p>

<b>Sign Out</b>	
Actor	Signed User
Garantía de éxito	Se ha cerrado la sesión del usuario.
Flujo típico de éxito	<ol style="list-style-type: none"> <li>1. El actor indica que quiere cerrar su sesión.</li> <li>2. Se cierra la sesión y se informa al usuario.</li> </ol>

Change Password	
Actor	Signed User
Garantía de éxito	Se ha cambiado la contraseña del usuario.
Flujo típico de éxito	<ol style="list-style-type: none"> <li>1. El actor indica que quiere cambiar la contraseña e introduce la contraseña anterior y la nueva.</li> <li>2. Los datos son validados y se cambia la contraseña. El usuario es informado.</li> </ol>
Flujos alternativos	2.a. Los datos no son válidos y no se cambia la contraseña. Se informa al usuario y termina el caso de uso.

### Guardado en la nube

Move world to Cloud	
Actor	Signed User
Garantía de éxito	El mundo se ha movido del dispositivo del usuario a su carpeta en la nube.
Flujo típico de éxito	<ol style="list-style-type: none"> <li>1. El actor indica que quiere mover un mundo a la nube.</li> <li>2. El archivo se mueve a la nube y se elimina del dispositivo del usuario. El actor es informado.</li> </ol>
Flujos alternativos	2.a. El actor dispone del número máximo de mundos almacenados en la nube. Se informa al usuario y termina el caso de uso.



Move world to Device	
Actor	Signed User
Garantía de éxito	El mundo se ha movido de la carpeta en la nube al dispositivo del usuario.
Flujo típico de éxito	<ol style="list-style-type: none"> <li>1. El actor indica que quiere mover un mundo a su dispositivo.</li> <li>2. El archivo se mueve al dispositivo del usuario y se elimina de la carpeta en la nube. El actor es informado.</li> </ol>

Load World	
Actor	Signed User
Garantía de éxito	El mundo almacenado en la nube se ha cargado.
Flujo típico de éxito	<ol style="list-style-type: none"> <li>1. El actor indica que quiere cargar un mundo a la nube.</li> <li>2. El mundo en la nube carga y el jugador puede jugar en él.</li> </ol>

Delete World	
Actor	Signed User
Garantía de éxito	El mundo se ha eliminado de la carpeta en la nube del usuario.
Flujo típico de éxito	<ol style="list-style-type: none"> <li>1. El actor indica que quiere eliminar un mundo de la nube.</li> <li>2. El archivo se elimina de la nube. El actor es informado.</li> </ol>

### 5.2.2. API

Debido a que el backend puede ser consumido por múltiples sistemas, incluso varios simultáneamente, es necesario especificar a qué endpoints se deben enviar las solicitudes. También se debe definir qué tipo de peticiones puede gestionar y responder. Por cada tipo de solicitud se especifican los parámetros e información que necesita cada una de ellas y qué

información contiene la respuesta en caso de completar la acción satisfactoriamente. No todas las solicitudes retornan datos, algunas se ejecutan y devuelven una respuesta vacía.

La API encargada de gestión de usuarios y autenticación dispone de los endpoints visibles en la Tabla 5. El endpoint “/edituser/password” está diseñado para que, en un futuro, nuevos endpoints para modificar datos de usuario se encuentren bajo la misma raíz “/edituser”.

API de gestión y autenticación de usuarios			
Endpoint	Tipo de petición	Requiere	Respuesta
/signin	POST	<ul style="list-style-type: none"> <li>Nombre de usuario</li> <li>Contraseña</li> </ul>	<ul style="list-style-type: none"> <li>Access token</li> <li>Refresh token</li> <li>Nombre de usuario</li> </ul>
/signup	POST	<ul style="list-style-type: none"> <li>Nombre de usuario</li> <li>Email</li> <li>Contraseña</li> </ul>	<ul style="list-style-type: none"> <li>Nombre de usuario</li> </ul>
/refreshaccesstoken	POST	<ul style="list-style-type: none"> <li>Refresh token</li> </ul>	<ul style="list-style-type: none"> <li>Access token</li> <li>Nombre de usuario</li> </ul>
/edituser/password	POST	<ul style="list-style-type: none"> <li>Access token</li> <li>Contraseña anterior</li> <li>Contraseña nueva</li> </ul>	
/forgotpassword	POST	<ul style="list-style-type: none"> <li>Nombre de usuario</li> <li>Confirmar</li> </ul> <p>Si Confirmar es <i>true</i>:</p> <ul style="list-style-type: none"> <li>➤ Código de confirmación</li> <li>➤ Contraseña nueva</li> </ul>	
/username	GET	<ul style="list-style-type: none"> <li>Access token</li> </ul>	<ul style="list-style-type: none"> <li>Nombre de usuario</li> </ul>

Tabla 5. Especificación de la API de gestión de usuarios y autenticación. Fuente: Elaboración propia

La API encargada del almacenamiento en la nube, por su lado, dispone de los endpoints visibles en la Tabla 6. Esta API está diseñada para dar soporte a toda una aplicación y guardar cualquier archivo, pero actualmente solo dispone de funciones de guardado en la nube para los mundos. Cuando se requiere un Mundo o se devuelve uno como respuesta, hace referencia al archivo. Este fichero se envía, en ambas direcciones, codificado como string en Base64.

API de almacenamiento en la nube			
Endpoint	Tipo de petición	Requiere	Respuesta
/worlds	GET	<ul style="list-style-type: none"><li>• Access token</li></ul>	<ul style="list-style-type: none"><li>• Lista de mundos del usuario</li></ul>
/world	GET	<ul style="list-style-type: none"><li>• Access token</li><li>• Nombre del mundo</li></ul>	<ul style="list-style-type: none"><li>• Mundo</li></ul>
	PUT	<ul style="list-style-type: none"><li>• Access token</li><li>• Nombre del mundo</li><li>• Mundo</li></ul>	<ul style="list-style-type: none"><li>• Nombre del mundo</li></ul>
	DELETE	<ul style="list-style-type: none"><li>• Access token</li><li>• Nombre del mundo</li></ul>	<ul style="list-style-type: none"><li>• Nombre del mundo</li></ul>

Tabla 6. Especificación de la API de almacenamiento en la nube. Fuente: Elaboración propia

5.3. Diseño

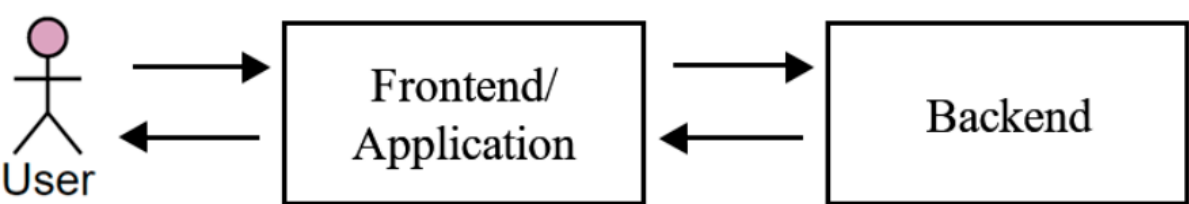


Figura 8. Diseño general del sistema. Fuente: Elaboración propia

El diseño general del sistema se puede visualizar en la Figura 8. Como se puede observar, el usuario únicamente interactúa con la aplicación, nunca con el backend directamente. Por

otro lado, el frontend y el backend son independientes, y se comunican exclusivamente a través de mensajes.

Esta independencia implica un bajo acoplamiento entre la aplicación y el backend, que pueden desconocer totalmente sobre qué sistema se ejecuta el otro componente o qué arquitectura utiliza. Los dos elementos pueden incluso utilizar lenguajes diferentes: únicamente es necesario que ambos cumplan la especificación de mensajes definida en el apartado 5.2.2.

### 5.3.1. Arquitectura backend

El diseño de la infraestructura se puede visualizar en la Figura 9. Como se puede observar, en la infraestructura final todo el backend se concentra en AWS, y Unity accede a los servicios a través de los endpoints de la API.

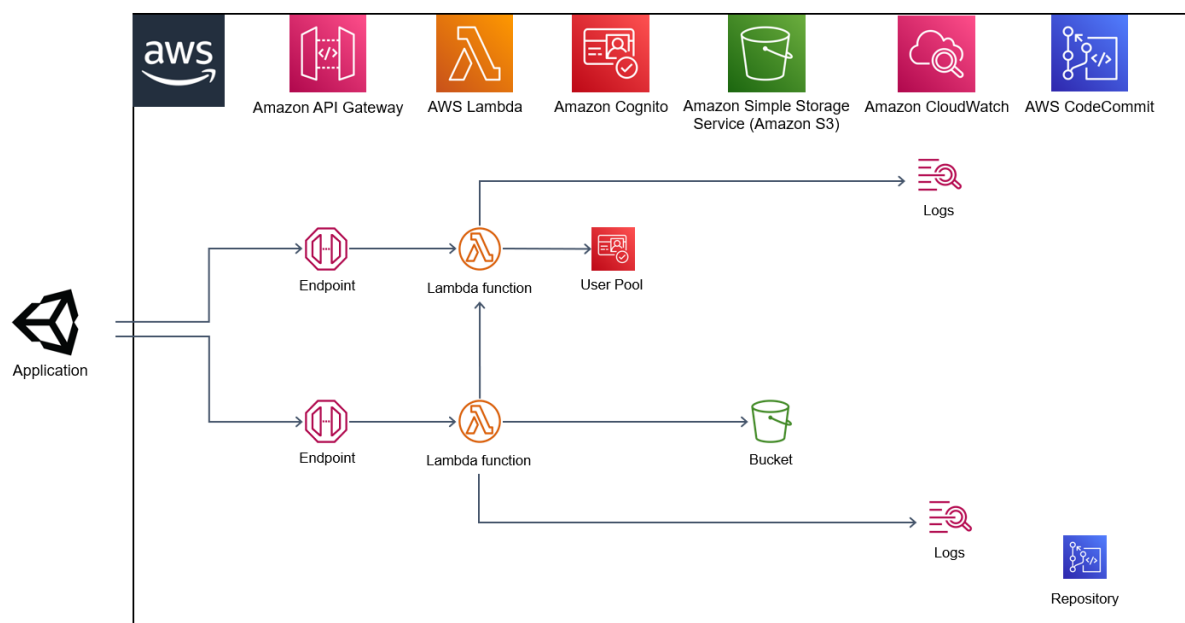


Figura 9. Infraestructura del backend diseñado en capas. Fuente: Elaboración propia

### 5.3.2. Arquitectura frontend

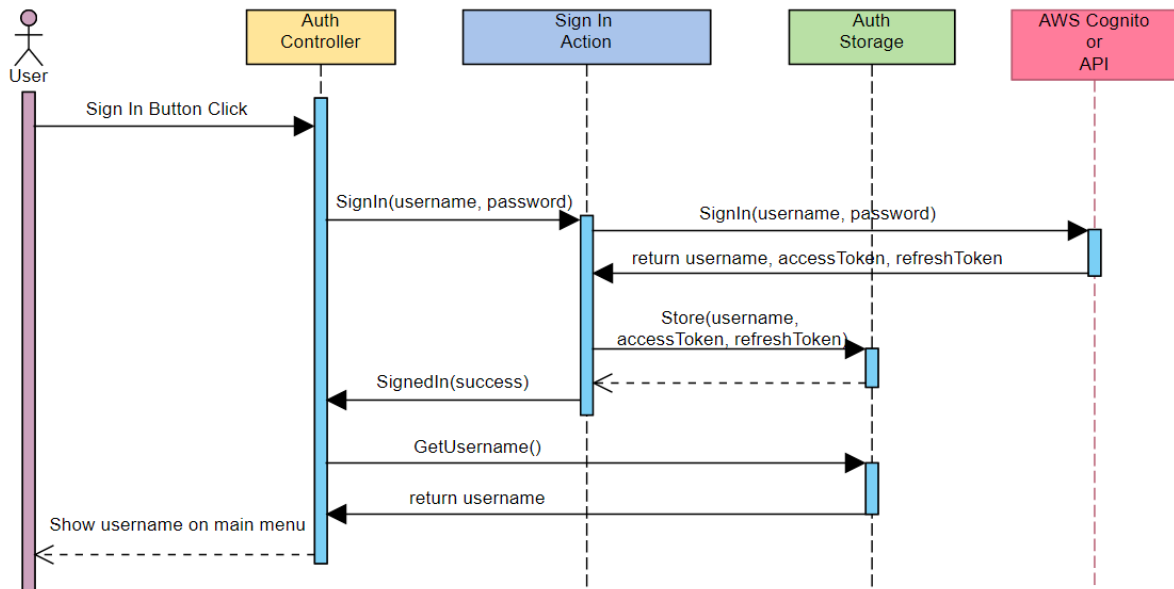


Figura 10. Ejemplo de la arquitectura utilizada en el frontend. Fuente: Elaboración propia

Se ha utilizado una arquitectura basada en tres capas: *Controllers*, *Actions* y *Storages*. En la arquitectura implementada, el usuario interactúa con la vista (escena de Unity). Al interactuar con, por ejemplo, un botón, se recoge el evento por un *Controller*. Este controlador llama a la *Action* correspondiente que realiza la acción, con las llamadas necesarias al backend. Esta guarda la nueva información en un *Storage* y, al terminar, informa al *Controller* a través de un callback. Si la *Action* ha terminado satisfactoriamente, el *Controller* actúa en consecuencia con los nuevos datos del *Storage* y cambia la escena, mostrando el resultado al usuario.

Esta arquitectura, además de permitir una separación de funcionalidades, facilita el mantenimiento y migración. Únicamente la capa de *Actions* conoce el backend utilizado. De este modo, se puede cambiar la plataforma de backend o la forma de consumirlo sin necesidad de modificar el resto de clases. Durante el desarrollo del proyecto las *Actions* se han alterado para consumir los servicios a través de una API y el resto de la aplicación ha continuado funcionando con normalidad.

### 5.3.3. Diagrama de navegación

La Figura 11 muestra el diagrama de navegación diseñado para la aplicación. Algunas de las pantallas únicamente son accesibles si se ha iniciado sesión, y otras si el usuario no se ha autenticado. La pantalla de “My Account” está diseñada para contener diferentes apartados de gestión de la cuenta, pero en este proyecto solo se incluye la funcionalidad de cambiar contraseña. Todas las funciones del almacenamiento en la nube se realizan desde la pantalla de “Load World Menu”. La pantalla de “Main Menu” es el punto de entrada en la aplicación.

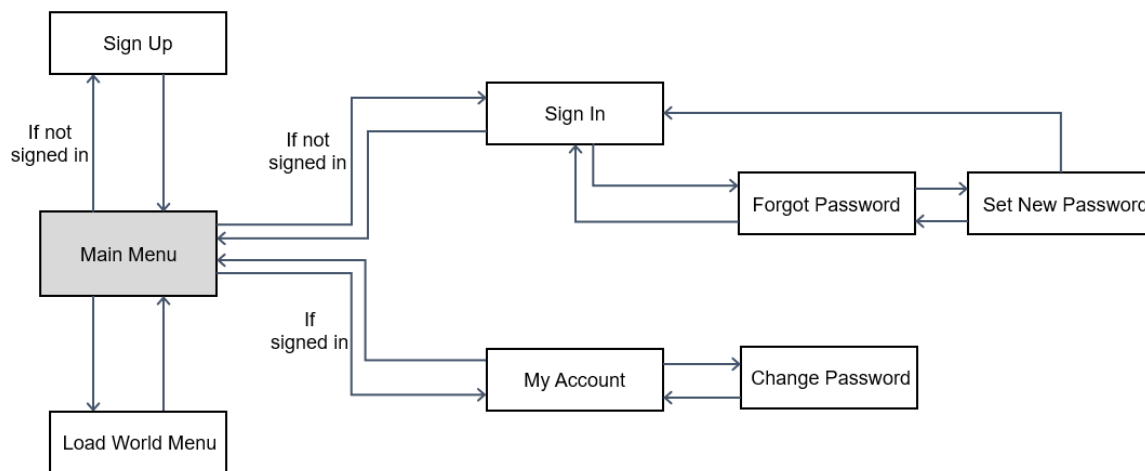


Figura 11. Diagrama de navegación de la aplicación. Fuente: Elaboración propia

A continuación se muestran los esqueletos de cada una de las pantallas, con los campos básicos requeridos en cada una. Los óvalos representan botones, mientras que los rectángulos equivalen a un *input*, independientemente de su tipo. Las figuras de bordes intermitentes delimitan áreas que contienen otros elementos.

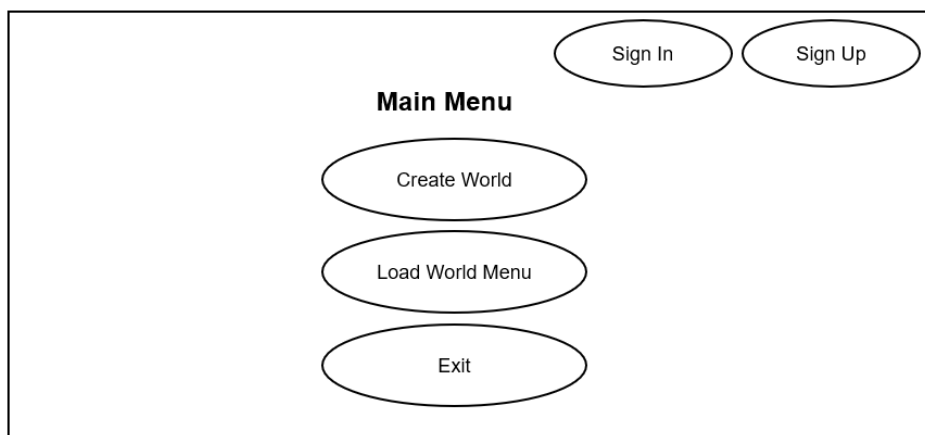


Figura 12. Esqueleto de la pantalla Main Menu. Fuente: Elaboración propia

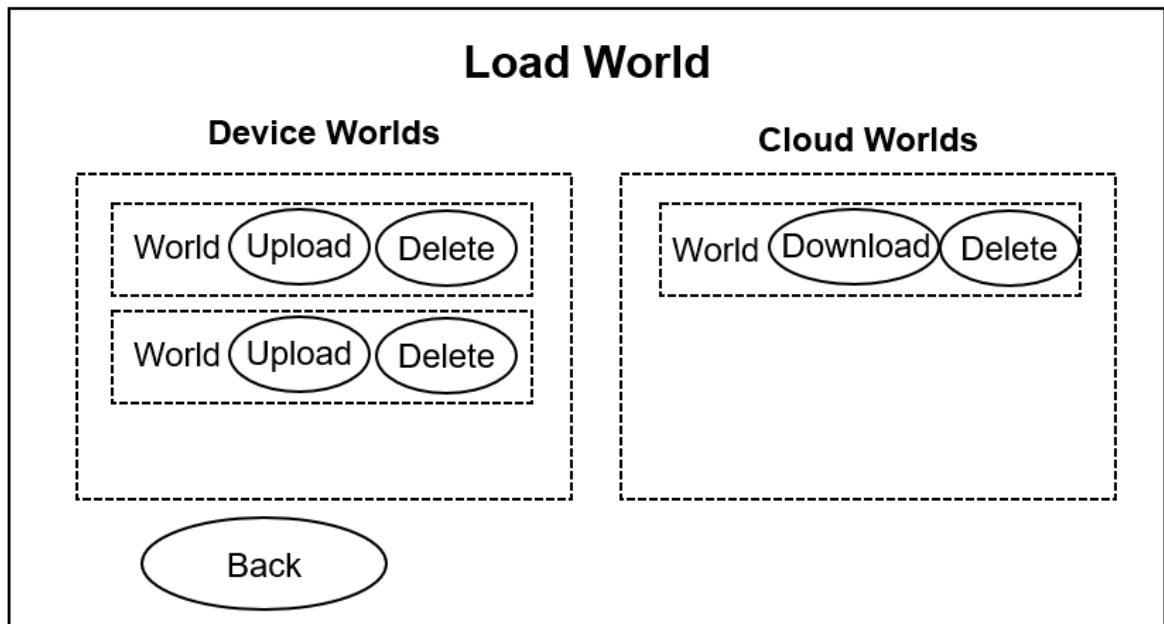


Figura 13. Esqueleto de la pantalla Load World Menu. Fuente: Elaboración propia

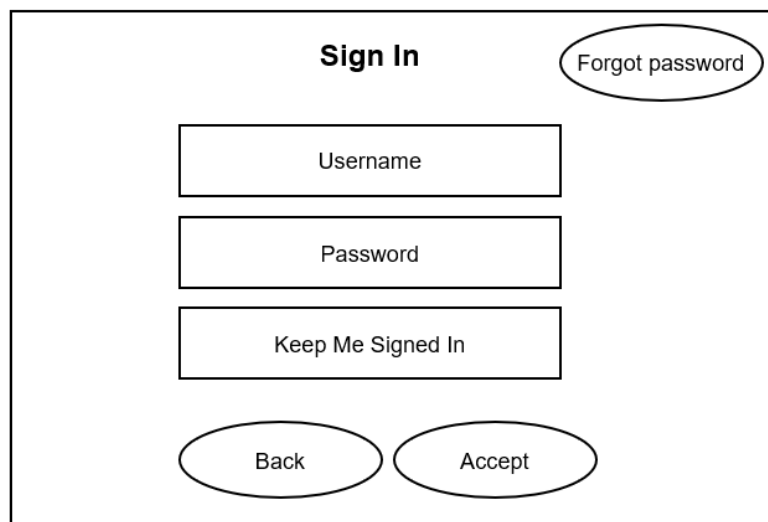
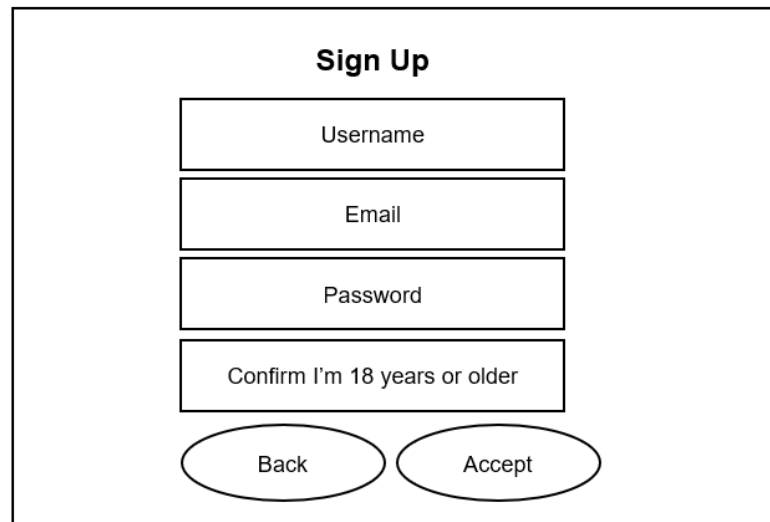
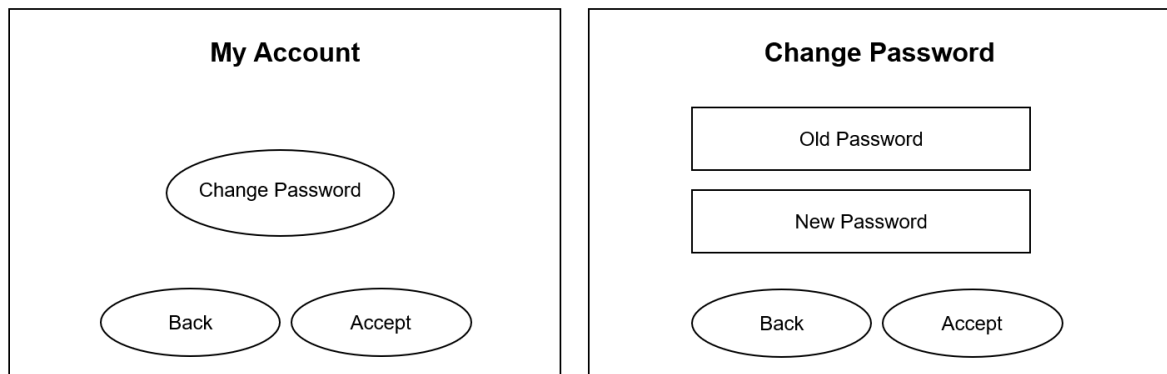


Figura 14. Esqueleto de la pantalla Sign In. Fuente: Elaboración propia



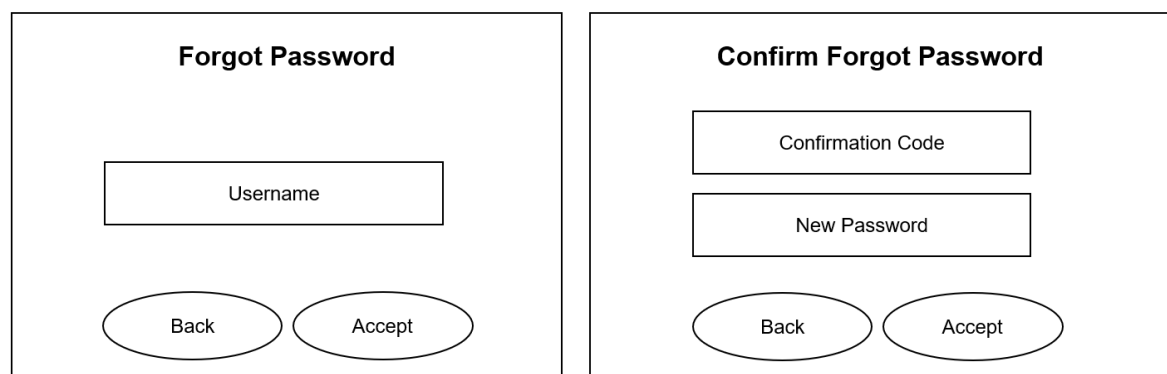
The wireframe for the 'Sign Up' screen is enclosed in a rectangular border. At the top center is the title 'Sign Up' in bold. Below the title are four stacked rectangular input fields. The first field is labeled 'Username', the second 'Email', the third 'Password', and the fourth 'Confirm I'm 18 years or older'. At the bottom of the screen are two oval buttons: 'Back' on the left and 'Accept' on the right.

Figura 15. Esqueleto de la pantalla Sign Up. Fuente: Elaboración propia



This block contains two side-by-side wireframes. The left wireframe, titled 'My Account' in bold, features a central oval button labeled 'Change Password'. At the bottom are two oval buttons: 'Back' on the left and 'Accept' on the right. The right wireframe, titled 'Change Password' in bold, has two stacked rectangular input fields labeled 'Old Password' and 'New Password'. At the bottom are two oval buttons: 'Back' on the left and 'Accept' on the right.

Figura 16. Esqueleto de las pantallas My Account y Change Password. Fuente: Elaboración propia



This block contains two side-by-side wireframes. The left wireframe, titled 'Forgot Password' in bold, has a single rectangular input field labeled 'Username'. At the bottom are two oval buttons: 'Back' on the left and 'Accept' on the right. The right wireframe, titled 'Confirm Forgot Password' in bold, has two stacked rectangular input fields labeled 'Confirmation Code' and 'New Password'. At the bottom are two oval buttons: 'Back' on the left and 'Accept' on the right.

Figura 17. Esqueleto de las pantallas Forgot Password y Confirm Forgot Password. Fuente: Elaboración propia



## 5.4. Implementación

### 5.4.1. Elección de plataforma

El apartado 2.1.1.1 actúa como una fase de eliminación respecto la plataforma a escoger, ya que permite eliminar las alternativas carentes de alguna característica requerida. Al terminar esta etapa preliminar, se han descartado las opciones de Firebase y un servidor propio.

El servidor propio se elimina fácilmente, ya que no ofrece opciones de escalabilidad, uno de los objetivos del proyecto. Tampoco seguridad ni otras de las funcionalidades requeridas. Algunas de las características podrían ser implementadas desde cero, pero esto incrementaría enormemente el tiempo y complejidad del proyecto. Se podría plantear externalizar algunas de estas funciones, pero eso se opondría al ideal de englobar todo el proyecto en una única plataforma.

En cuanto a Firebase, a pesar de cumplir casi todas las necesidades, no dispone de alojamiento propio para repositorios de Git propio. Esto obligaría a utilizar una plataforma adicional para el control de versiones, fraccionando el proyecto y repartiéndolo entre diferentes entornos. Además, es una plataforma orientada en gran medida al desarrollo de apps móviles, un factor que podría haber provocado problemas durante el desarrollo. Aun así, esta particularidad no se ha tenido en cuenta en la fase inicial de eliminación.

Las opciones restantes son Azure y AWS. La Tabla 7 muestra la comparación entre las dos plataformas, puntuando del cero al cinco las diferentes características deseables para el backend. Un cero implica que no ofrece esa característica, mientras que un cinco significa que la ofrece de una calidad óptima. En características negativas como el coste se sigue la misma regla, por lo que un valor superior indica un coste inferior.

<i><b>Característica</b></i>	<i><b>Azure</b></i>	<i><b>AWS</b></i>
<i>Coste</i>	4	4
<i>Durabilidad</i>	5	5
<i>Servicios</i>	4	5
<i>Regiones</i>	2	3
<i>Total</i>	15	17

Tabla 7. Evaluación de Azure y AWS. Fuente: Elaboración propia

Ambas opciones están muy igualadas y no hay grandes diferencias. Ambos tienen unos costes muy similares y asequibles sin tener en cuenta sus capas gratuitas. También coinciden en su disponibilidad: 99,999999999% (once nueves). A pesar de que Azure tiene más regiones planificadas donde expandirse, Amazon dispone actualmente una mayor cantidad de zonas de disponibilidad.

AWS supera a Azure en el apartado de servicios a pesar de que las dos plataformas tienen una cantidad similar. La puntuación superior se debe a que Amazon dispone de su propio motor de juegos: Lumberyard. Este factor es especialmente relevante para este proyecto, ya que el backend está enfocado a empresas de videojuegos. Por esta razón, si una empresa usa el motor de Amazon, su juego estará mejor integrado con el backend en AWS que un juego en cualquier otro motor.

Por estas razones, finalmente se ha tomado la decisión de utilizar Amazon Web Services como plataforma para el desarrollo del proyecto.

<i><b>Plataforma</b></i>	<i><b>Decisión</b></i>	<i><b>Causa</b></i>
<i>Azure</i>	Eliminado	Menos características deseables
<i>Amazon Web Services</i>	Elegido	Mejor opción
<i>Firebase</i>	Eliminado	No cumple requerimientos
<i>Servidor propio</i>	Eliminado	No cumple requerimientos

Tabla 8. Decisión tomada respecto a cada plataforma. Fuente: Elaboración propia

### 5.4.2. Creación de cuenta

La cuenta de AWS se crea fácilmente, con los datos comunes y añadiendo la tarjeta de crédito para los cobros. El usuario creado es el “usuario raíz”: una cuenta con privilegios superiores a los de cualquier otro usuario. La plataforma no recomienda trabajar y utilizar los servicios usando esta cuenta, ni siquiera para tareas administrativas. Para el uso cotidiano es preferible crear un usuario IAM para ello, con los permisos correspondientes.

IAM (AWS Identity and Access Management) es el servicio encargado de controlar la autenticación y autorización de la cuenta de AWS. Cada usuario de IAM es una identidad con unos permisos concretos. Esto permite, por ejemplo, que una empresa tenga un usuario de IAM para cada trabajador con diferentes privilegios cada uno. También se puede

establecer si un usuario tiene acceso de consola, acceso programático o ambos. El primer caso proporciona acceso a la consola con una contraseña, mientras que el segundo permite acceder a los servicios usando una *access key ID* y *secret access key*.

Para el desarrollo del proyecto, se ha creado un usuario con acceso de consola y programático con permisos de administrador.

### 5.4.3. Servicios

#### 5.4.3.1. Cognito

Se ha creado un grupo de usuarios nombrado *company-users*. Se ha configurado para que los usuarios puedan iniciar sesión con su nombre de usuario o correo. Se ha escogido que la contraseña deba incluir letras minúsculas, mayúsculas y números, no son necesarios caracteres especiales.

Los correos de verificación y cambio de contraseña se enviarán directamente a través de Cognito. De este modo, no es necesario utilizar otro servicio de AWS para enviar estos simples correos. Al registrarse, el usuario recibirá un link para verificar su cuenta.

También se ha añadido un cliente de aplicación para obtener las credenciales (ClientId) para poder utilizar el grupo de usuarios desde Unity. Esto es necesario para realizar las solicitudes a Cognito desde aplicaciones externas.

#### 5.4.3.2. S3

Se ha creado un bucket nombrado *game-pcg-bucket* en la región *eu-west-3* (París). El bucket bloquea todo el acceso público y tiene desactivado el versionado (no guarda diferentes versiones de un mismo archivo). Esta decisión se debe a que almacenar las versiones consumiría un espacio múltiples veces superior, ya que se generaría una versión nueva cada vez que el usuario guarde su mundo. Además, no se considera que sea necesario para el guardado de mundos ya que, generalmente, los jugadores quieren mantener su mapa en el estado más reciente.

#### 5.4.3.3. CodeCommit

Se ha creado un repositorio privado llamado *game-pcg*, que incluye, inicialmente, el proyecto de Unity y la documentación elaborada. No se necesita ningún trigger para el desarrollo del proyecto.

#### 5.4.3.4. API Gateway

Se han creado dos APIs. Ambas se han creado como API REST, ya que incluye más personalización y funcionalidades que una HTTP API. Estas funcionalidades extra incluyen: caché, validación de petición/respuesta, tests, logs de ejecución... No todas las características extra han sido utilizadas, pero las dos últimas se han considerado un requisito.

- *company-auth-api*: punto de entrada para las funcionalidades de gestión de cuenta de usuario.
- *game-pcg-api*: encargada de gestionar exclusivamente las llamadas relacionadas con el “videojuego” creado en Unity. Actualmente proporciona los endpoints para la funcionalidad de guardado en la nube.

Esta separación permite una mayor flexibilidad para usar los servicios y evitar funcionalidades repetidas. Al disponer de un endpoint exclusivo para funciones de autenticación y gestión de cuenta, no recibirá tráfico para funcionalidades de un juego concreto. Cada juego, en cambio, puede tener su propia API para sus necesidades específicas.

Cada API está formada por diferentes recursos y métodos. Los recursos son diferentes endpoints dentro de misma URL base, mientras que los métodos son las solicitudes disponibles en cada recurso. La configuración realizada para cada API se puede visualizar en la Figura 18.

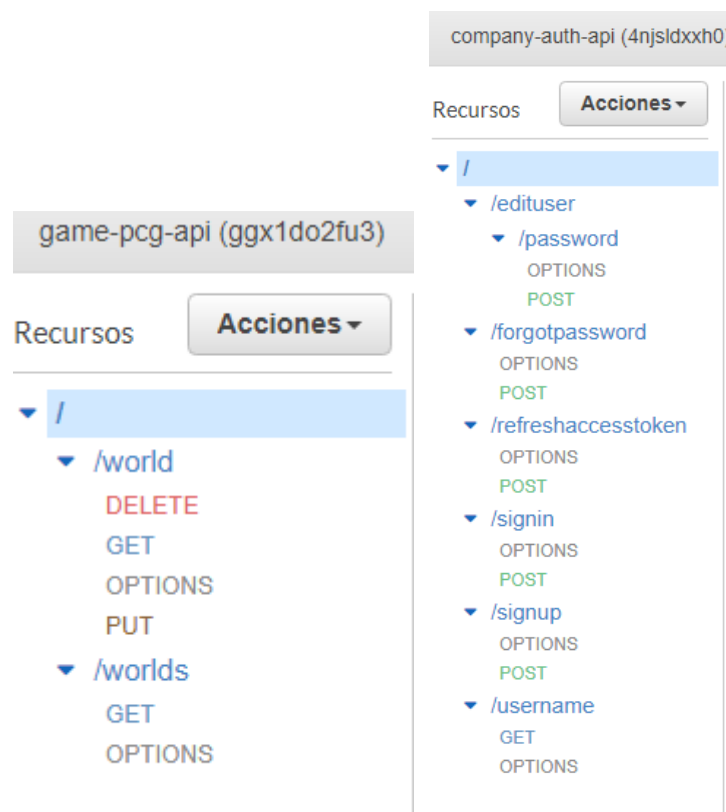


Figura 18. Configuración de API Gateway de game-pcg-api y company-auth-api. Fuente: Elaboración propia

Antes de publicar una API, cada método debe de disponer de una “solicitud de integración”, que corresponde al destino al que llamará el método. En esta implementación, ambas APIs gestionarán todos sus métodos a través de funciones Lambda, cuya implementación se explica en el apartado siguiente. Además, se usa la integración de proxy Lambda, lo que implica que no es necesaria configuración extra y la información de cada solicitud llega a la función Lambda en forma de parámetro.

Los métodos de *company-auth-api* utilizarán la Lambda *company-auth-lambda* mientras que los de *game-pcg-api* llamarán a *game-pcg-worlds-lambda*. Actualmente cada API utiliza una única Lambda, pero no es necesario que sea así. Esta implementación está diseñada de modo que una nueva funcionalidad de backend en la aplicación utilice la misma API, pero los nuevos métodos apunten a una nueva Lambda.

#### 5.4.3.5. Lambda

Al elegir el lenguaje a utilizar en las funciones Lambda se han tenido en cuenta Node.js, Python y Ruby. Este filtro inicial se debe a que son los tres únicos lenguajes que admite el editor de código de la consola de AWS. El uso de la consola facilita la publicación de las

funciones, y elegir otro lenguaje ralentiza el proceso de publicación cada vez que se realicen cambios. Además, estos tres lenguajes disponen del SDK de AWS integrado en todas sus versiones, por lo que no necesita de su instalación manual.

Finalmente se ha decidido utilizar Node.js como lenguaje para las funciones Lambda, concretamente la versión 14.x. De los tres lenguajes mencionados, se ha elegido Node.js por su facilidad para trabajar con JSONs, ya que es el formato utilizado para transmitir información en las solicitudes.

Otra opción sopesada es .NET, ya que es el lenguaje utilizado en la aplicación creada en Unity. Aun así, esta no es una razón válida para escogerlo, ya que en todo momento se ha mantenido independencia entre la aplicación y el backend. Además, el backend está diseñado para poder ser consumido por un juego creado en cualquier motor y cualquier lenguaje. Este factor, añadido no disponer de los beneficios aportados por los otros tres lenguajes, ha provocado que Node.js se afiance como la mejor opción como lenguaje para las funciones Lambda creadas.

Del mismo modo que las APIs, se han creado dos Lambdas:

- *company-auth-lambda*: encargada de las funcionalidades de autenticación y gestión de las cuentas de usuario.
- *game-pcg-worlds-lambda*: encargada de gestionar todas las solicitudes referentes al guardado de mundos de *game-pcg*.

#### 5.4.3.6. CloudWatch

Se han creado dos grupos de registros, uno para cada Lambda. De este modo se pueden visualizar los registros de ejecución de cada una de ellas, facilitando su desarrollo y la detección de errores.

#### 5.4.4. Preparación de Unity

Para desarrollar la aplicación cliente, se ha instalado la versión de Unity 2019.4.20f1 LTS. Se ha escogido una versión *Long Term Service*, ya que estas reciben soporte de larga duración. Este factor es relevante para proyectos largos que se encuentran en desarrollo, y este proyecto simula esa situación. Se ha añadido al proyecto de Unity el paquete AWS SDK para .NET, obtenido de [21]. Este paquete incluye las librerías para usar todos los servicios

de AWS, por lo que se han eliminado las que no son necesarias para este proyecto. Únicamente se han mantenido las que se utilizan en el desarrollo y sus dependencias directas.

### 5.4.5. Registro

En la Figura 19 se visualiza el sencillo formulario de registro creado en Unity, muy similar al esqueleto diseñado. Al enviar el formulario se realizan comprobaciones en el cliente, como que las contraseñas coincidan, el patrón de email o los caracteres mínimos de los campos. Aun así, en caso de error, Cognito responde con una excepción, y se muestra al usuario el mensaje correspondiente. Los errores que puede devolver el servicio en este caso son, principalmente, no cumplir con las políticas de contraseña o si el usuario ya existe.



The image shows a 'Sign Up' form with a blue background. At the top, the text 'Sign Up' is displayed in white. Below this, there are four input fields with white backgrounds and blue borders, each preceded by a label: 'Username:', 'Email:', 'Password:', and 'Confirm Password:'. Under the 'Password:' field, there is a checkbox with the text 'I acknowledge that I'm 18 years or older.' At the bottom of the form, there are two buttons: 'Back' and 'Sign Up', both in a bold, sans-serif font.

Figura 19. Formulario de registro desde Unity. Fuente: Elaboración propia

Para realizar las peticiones se utiliza la clase *AmazonCognitoIdentityProviderClient*, al que le enviamos una petición de registro (*SignUpRequest*). La petición requiere el *ClientId*, el nombre de usuario y contraseña. También se le añaden los campos que hemos marcado como obligatorios, en este caso email y apodo. Si el registro se ha completado satisfactoriamente, Cognito responde con una *SignUpResponse*, pero la aplicación no necesita información de esta respuesta.

Se ha mencionado que el apodo (*nickname*) es un campo obligatorio. Sin embargo, esta información no aparece en el formulario, sino que se inicializa con el mismo valor que el nombre de usuario. A pesar de que esta variable no se utiliza actualmente, se ha añadido para dejar abierta la opción de permitir a los usuarios cambiar su apodo, y que este sea visible

en los juegos. Se ha incorporado el campo con vistas de futuro debido a que Cognito no permite cambiar los atributos obligatorios. Al incluirlo desde el principio, evitamos tener que crear otro grupo de usuarios y migrar todos los datos.

Al crear la cuenta, se informa al usuario que la operación ha sido exitosa y que debe verificar la cuenta con el link recibido en su correo. Este link redirige al usuario a un dominio de este grupo de usuarios y confirma automáticamente al usuario sin que tenga que introducir el código de verificación manualmente.

#### **5.4.6. Inicio de sesión**

En el caso de inicio de sesión, solo se comprueba la longitud de los campos. Cognito responde con error en caso de usuario o contraseña incorrectos o si el usuario no ha activado su cuenta con el link del correo. Si no hay error, la respuesta incluye diferentes campos, como el tiempo de expiración del token o el metadata del nuevo dispositivo. De esta información únicamente se recoge el token de acceso (access token) y el token de actualización (refresh token).

Para realizar el inicio de sesión se vuelve a utilizar *AmazonCognitoIdentityProviderClient* para enviar la petición, pero en este caso es de tipo *InitiateAuthRequest* (iniciar solicitud de autenticación). En la solicitud se envía el *ClientId* y que tipo de autenticación se va a realizar, en este caso se va a autenticar un usuario usando su contraseña. En caso de que las credenciales sean correctas, se recibe un objeto *InitiateAuthResponse* con los tokens mencionados.

La respuesta puede incluir cual es el siguiente “challenge” y los parámetros que necesita. Estos desafíos son pasos extra necesarios antes de obtener los tokens e iniciar sesión satisfactoriamente. Estos pasos pueden ser, por ejemplo, cambio de contraseña obligatorio en el primer inicio de sesión o introducir el código de activación recibido por SMS. No se ha configurado ningún desafío, por lo que el inicio de sesión se completa en un único paso.



The image shows a 'Sign In' form on a blue background. At the top, the text 'Sign In' is displayed in a light blue font. Below this, there are two white input fields. The first is labeled 'Username:' and the second is labeled 'Password:'. Under the password field, there is a checkbox followed by the text 'Keep me signed in.'. At the bottom of the form, there are two buttons: 'Back' in orange text and 'Sign In' in white text.

Figura 20. Formulario de inicio de sesión desde Unity. Fuente: Elaboración propia

Como se aprecia en la Figura 20, se ha añadido la opción de mantener la sesión iniciada. En caso de iniciar sesión correctamente con la opción marcada, se almacena el token de actualización. El dato se guarda en `PlayerPrefs`, una clase de Unity utilizada para guardar preferencias del jugador entre sesiones de juego [22]. De esta manera, la información se almacena en el registro de la plataforma del usuario, en caso de Windows se almacena en la clave “`HKCU\Software\ExampleCompanyName\ExampleProductName`”.

Con la sesión iniciada, el usuario puede desconectarse. Esta funcionalidad simplemente elimina los valores de nombre de usuario y tokens guardados de la memoria y el token de actualización del registro. De esta manera ya no se dispondrá del token para las solicitudes, obligando a iniciar sesión de nuevo.

Finalmente se ha implementado la funcionalidad de actualizar el token de acceso. Este tiene una caducidad predeterminada de una hora, mientras que el token de actualización se puede utilizar durante un mes. Cada vez que el usuario inicia la aplicación o cuando una solicitud responde con un error de token caducado, se utiliza el token de actualización para obtener un nuevo token de acceso. Si la obtención del nuevo token falla, significa que han caducado ambos tokens y se cierra la sesión automáticamente.

Un usuario activo no necesitará iniciar sesión regularmente, ya que siempre tendrá un token de actualización válido. Esto se debe a que, al refrescar el token de acceso, se recibe también un token de actualización nuevo si este está cerca de su fecha de caducidad. La solicitud de refresco del token es realmente un inicio de sesión, pero en lugar de utilizar el nombre de usuario y contraseña, se utiliza el token de actualización.

### 5.4.7. Guardado en la nube

Los ficheros que se almacenan en S3, son archivos binarios de extensión *.world*. Estos se generan desde la aplicación y representan un mundo que el juego puede cargar para generar un mapa jugable. El tamaño del archivo depende de las dimensiones del mapa, pero el caso común, un mapa de 1000x1000 bloques, ocupa alrededor de 977KB. Las rutas donde se almacena el archivo son:

- S3: dentro del bucket *game-pcg-bucket*, “<username>\worlds”.
- Dispositivo: Unity proporciona una ruta al directorio donde almacenar datos persistentes, que depende del sistema donde se ejecuta el juego. Dentro de esta carpeta se ha creado un directorio “\worlds” donde guardar los mundos. De este modo, la ruta donde se encuentran los mundos en el caso de Windows es “%userprofile%\AppData\LocalLow\<companyname>\<productname>\worlds”.

Como se puede observar, tanto en el dispositivo del usuario como en S3, se ha creado una subcarpeta “\worlds”. Esta decisión se ha tomado en vista de que el juego pudiera crecer y se generen archivos nuevos. De esta manera, si se tuvieran que almacenar personajes (o cualquier tipo de fichero), se crearía una nueva carpeta donde guardarlos. No sería necesario modificar nada respecto a los mundos y no se encontrarían todos los archivos mezclados en la raíz.

Para gestionar el guardado en la nube se han implementado cinco funcionalidades:

1. **ListWorlds:** obtiene la lista de mundos que usuario almacena en su carpeta de S3. Únicamente recoge los nombres de los archivos para mostrarlos en la lista.
2. **GetWorld:** descarga el mundo y lo mantiene en memoria en forma de *Stream*, sin almacenarlo en el dispositivo. Este *Stream* representa un mundo en la nube, y puede ser convertido a un objeto *World* que el juego puede interpretar y cargar.
3. **DownloadWorld:** utiliza *GetWorld* para obtener un mundo y almacena el *Stream* en el dispositivo en forma de archivo *.world*. Una vez guardado, borra el mundo de la carpeta en S3.
4. **DeleteWorld:** elimina un mundo de la carpeta de S3 del usuario.
5. **UploadWorld:** sube un mundo a la carpeta de S3 del usuario. Una vez subido, elimina el mundo del dispositivo.

La lista de mundos se carga únicamente la primera vez que se abre el menú para cargar mundos. Una vez se obtiene la lista, la adición y eliminación de elementos en esta se realiza en la propia aplicación, evitando peticiones innecesarias a S3. Si se cierra la sesión la lista se reinicia, impidiendo que un nuevo usuario visualice la lista de mundos del anterior jugador. Además, si un nuevo cliente se autentica, el sistema detecta que es la primera vez que accede al menú y carga la nueva lista.

En la Figura 21 se puede visualizar el menú para gestionar los mundos, indicando que acción realiza cada botón. La columna izquierda incluye los archivos que se encuentran en el dispositivo del cliente, mientras que la columna derecha contiene la lista de mundos almacenados en S3. En caso de no haber iniciado sesión, la lista de mundos en la nube aparece desactivada mostrando un mensaje informativo. Todas las acciones muestran un mensaje si se completan satisfactoriamente, excepto GetWorld que carga el mundo directamente y cambia a la pantalla del juego.

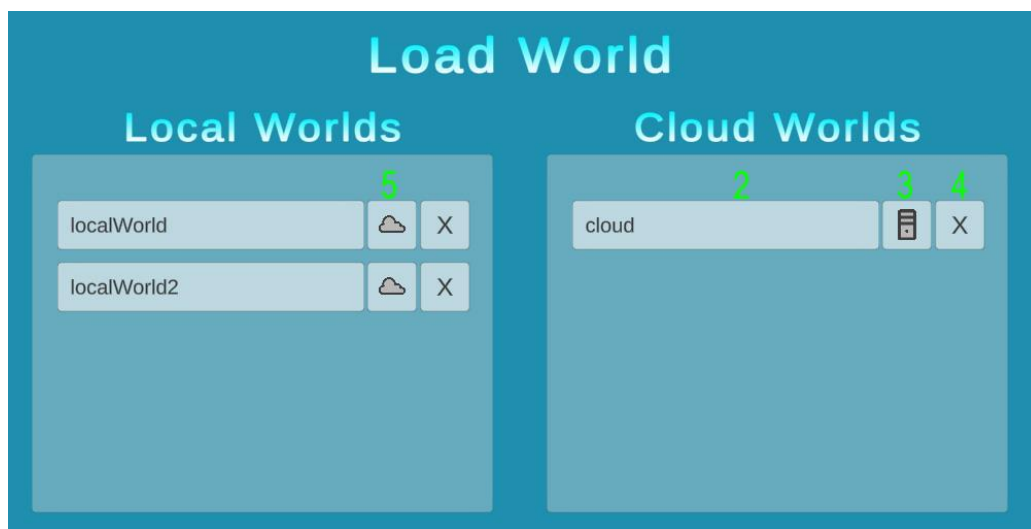


Figura 21. Menú para gestionar los mundos en Unity. Fuente: Elaboración propia

Después de sopesar diferentes opciones para autorizar a los usuarios, se ha optado por lo siguiente: el acceso a los archivos se realiza con permisos de administración, pero a través de Cognito se valida el token y se obtiene el identificador de la carpeta. Para ello, cada vez que se realiza una solicitud que requiere autorización, se envía una petición GetUserRequest a Cognito. De este modo, proporcionando un token de acceso válido, se obtiene el nombre de usuario del cliente i se forma la ruta de la carpeta de S3. Esto proporciona un cierto nivel de seguridad, ya que no se utiliza el nombre de usuario almacenado en memoria.

Sin embargo, la solución escogida no es óptima y presenta dos problemas principales:

- Las credenciales de administrador (accessKey y secretKey) para el acceso a S3 se encuentran en el código de la aplicación.
- Si un usuario malintencionado consigue modificar la información de la respuesta a la petición GetUserRequest, este podría hacerse pasar por otro usuario. Para poder realizar esta práctica, la modificación debe hacerse antes de que el paquete de red llegue a la aplicación. Del mismo modo, podría modificar la información del paquete con la ruta de la carpeta de S3, antes de que este sea enviado.

No obstante, ambos inconvenientes se solucionan en el siguiente punto del desarrollo. Al mover la funcionalidad a una API REST, las claves ya no se encuentran en la aplicación del cliente. Un usuario con malas intenciones tampoco podría causar problemas modificando la los paquetes de red, ya la información sensible nunca llega al cliente.

#### 5.4.8. API

El proceso de migración del acceso a datos de la aplicación a la API, empieza con la creación de las APIs y funciones Lambda, configurados como se explica en los apartados 5.4.3.4 y 5.4.3.5 respectivamente. Una vez creados, se ha creado una Action en la aplicación para comprobar que la conexión funcionase bien. La prueba ha sido satisfactoria y se pueden empezar a migrar las funcionalidades.

Al migrar la primera función ha surgido un problema, un comportamiento inesperado en las Lambdas: los callbacks no funcionan. Todos los ejemplos utilizan callbacks para gestionar las llamadas asíncronas, e incluso el código funciona correctamente al ejecutarlo en local. Aun así, ese mismo código ejecutado en la Lambda provocaba que esta nunca respondiera, ya que las respuestas se encontraban en las funciones de callback. Finalmente se encontró una manera de solucionar el problema utilizando *Promises*.

Gracias a los conocimientos adquiridos previamente, únicamente ha sido necesario consultar la documentación en Node.js para migrar las funcionalidades existentes sin grandes problemas. Las funciones referentes al guardado en la nube, requieren acceder a funcionalidades de autenticación. Para hacer esta comunicación entre Lambdas, se ha instalado el paquete *axios*: un cliente para solicitudes HTTP basado en promesas. De este modo una Lambda accede a la otra a través de una petición a su endpoint.

La aplicación en Unity, por su parte, realiza las peticiones HTTP utilizando *UnityWebRequest*, del paquete *UnityEngine.Networking*. Además, se ha instalado el paquete *Newtonsoft.Json* para facilitar el uso de JSONs en .NET, ya que es el formato utilizado para transmitir la información en las solicitudes.

Gracias al diseño utilizado, solo las Actions han sido modificadas para realizar la migración. Todas las funcionalidades han sido migradas y, a partir de este cambio, el acceso a datos y tareas de seguridad se ejecutan en la nube. La información sensible como *accessKeys* y *secretKeys* se encuentra también en la nube, por lo que nunca llegan a estar en el dispositivo del cliente, mejorando la seguridad.

Una vez terminado el proceso, se ha eliminado el SDK de AWS de Unity. En consecuencia, la aplicación del cliente no conoce sobre qué plataforma está construida el backend y, en caso de necesitar otro cambio, únicamente sería necesario cambiar las URL.

#### **5.4.9. Cambio de contraseña**

La funcionalidad para que un usuario pueda cambiar la contraseña se ha implementado directamente en la función Lambda *company-auth-lambda*. Para ello se han creado dos recursos nuevos en API Gateway: */edituser* y */password* dentro de este (*/edituser/password*). Esta separación se ha diseñado en vista de implementar funcionalidades para modificar diferentes atributos del usuario, creando recursos nuevos dentro del mismo recurso compartido.

Mientras que para modificar otros atributos Cognito solo requiere el *accessToken*, editar la contraseña requiere de un parámetro extra: la contraseña actual. Un usuario debe conocer su contraseña para poder cambiarla, pero esta característica simplemente afecta al formulario y los parámetros enviados a la Lambda. La Figura 22 muestra el menú para la gestión de la cuenta, que únicamente permite cambiar la contraseña, y el formulario creado para realizar la modificación.

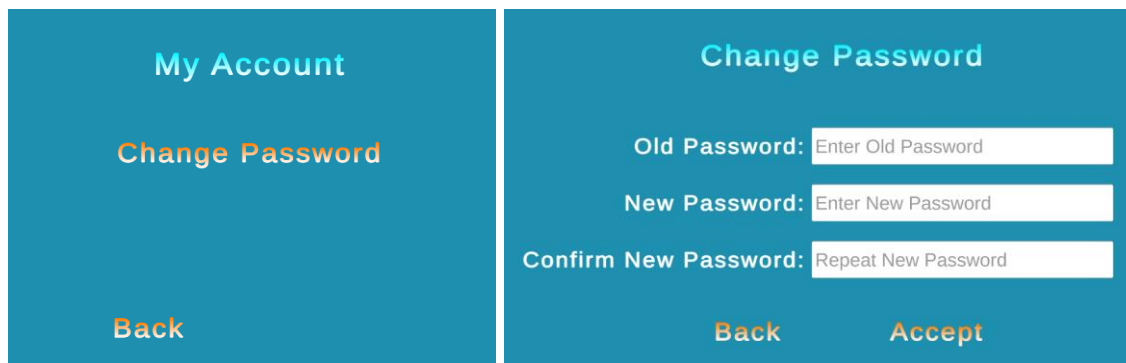


Figura 22. Menú “My Account” y formulario de cambio de contraseña. Fuente: Elaboración propia

Las comprobaciones para realizar el cambio de contraseña son las mismas que las aplicadas en el registro de los usuarios. Cognito también aplica las mismas políticas que se han configurado, por lo que las contraseñas siempre son consistentes respecto a las reglas establecidas. El campo de confirmar la nueva contraseña no es necesario, pero se ha añadido para asegurar que el usuario la escriba correctamente.

#### 5.4.10. Recuperación de contraseña

Del mismo modo que el cambio de contraseña, la recuperación de esta se ha implementado en la Lambda correspondiente, accediendo a ella a través de un recurso nuevo en la API. No obstante, esta funcionalidad requiere de dos pasos: solicitar la recuperación de contraseña y establecer la nueva clave. No se ha creado un recurso o método para cada uno, sino que a través de la misma solicitud se pueden realizar las dos acciones. Para decidir qué paso ejecutar, se utiliza un parámetro de URL (*query string*).

El primer paso consiste simplemente en introducir el nombre de usuario o email de la cuenta de la que se quiere recuperar la contraseña. En ese momento, el usuario recibe un correo al email asociado a su cuenta con un código de verificación. En el segundo paso, el usuario debe introducir el código que ha recibido y establecer una nueva contraseña. La Figura 23 muestra los formularios creados para los dos pasos mencionados.

The image displays two sequential steps of a 'Forgot Password' process. Both steps are contained within a blue rectangular area with a white title 'Forgot Password' at the top center.   
Step 1 (left): Features a single text input field labeled 'Username:' with the placeholder text 'Enter Username'. At the bottom, there are two orange buttons: 'Back' on the left and 'Next' on the right.   
Step 2 (right): Features three text input fields. The first is labeled 'Confirmation Code:' with placeholder 'Enter Confirmation Code'. The second is labeled 'Password:' with placeholder 'Enter Password'. The third is labeled 'Confirm Password:' with placeholder 'Repeat Password'. At the bottom, there are two orange buttons: 'Back' on the left and 'Accept' on the right.

Figura 23. Formularios (paso 1 y 2) para la recuperación de contraseña. Fuente: Elaboración propia

Las comprobaciones aplicadas a la nueva contraseña son, de nuevo, las mismas que las empleadas en el registro de los usuarios y el cambio de contraseña. No es necesario que la nueva contraseña sea diferente a la anterior, es decir, un usuario puede utilizar este proceso para terminar estableciendo la misma contraseña. Una vez recuperada, la aplicación vuelve al formulario de inicio de sesión y el usuario puede autenticarse con su nueva contraseña.

#### 5.4.11. Conexión entre capas

En este apartado se muestran dos ejemplos del flujo seguido al realizar una acción de autenticación y otra de almacenamiento en la nube. Se puede observar la conexión entre frontend y backend, y los principales atributos necesarios en cada operación. El código visualizado no es exactamente el utilizado, ya que se ha simplificado algunas partes para facilitar su comprensión y visualización.

##### 5.4.11.1. Sign in

Tras rellenar los campos de nombre de usuario y contraseña se realiza una simple validación en la extensión de estos para evitar llamadas innecesarias. Tras esta comprobación se inicia la acción de iniciar sesión.

```
WWWForm signinBody = new WWWForm();  
signinBody.AddField("username", username);  
signinBody.AddField("password", password);  
string url = BaseUrlAuth+UrlSignin;  
UnityWebRequest request = UnityWebRequest.Post(url, signinBody);  
yield return request.SendWebRequest();
```

Figura 24. Petición de inicio de sesión. Fuente: Elaboración propia

La Figura 24 muestra el código utilizado para realizar la petición POST a la API. Se forma el *body* a partir del nombre de usuario y la contraseña, y se envía la petición a la URL de Sign In.

```
const AWS = require('aws-sdk')
AWS.config.region = 'eu-west-3';
var cognitoIdentityProvider = new AWS.CognitoIdentityServiceProvider();
```

Figura 25. Inicialización de la Lambda *company-auth-lambda*. Fuente: Elaboración propia

Una vez en la función Lambda, se importa el paquete *aws-sdk*, que existe integrado en todas las versiones de Node.js que ofrece AWS. En ese momento se configura la región y se crea un nuevo proveedor de identidad de Cognito. Esta inicialización de la Lambda se realiza siempre al iniciar, ya que todas las funciones en la lambda *company-auth-lambda* requieren del proveedor de identidad.

```
const signin = async (data) => {
  let params = {
    AuthFlow: 'USER_PASSWORD_AUTH',
    ClientId: ClientId,
    AuthParameters: {
      'USERNAME': data.username,
      'PASSWORD': data.password
    }
  };
  try {
    let authResponse = await cognitoIdentityProvider.initiateAuth
      (params).promise()

    let response = authResponse.AuthenticationResult
    response.Username = await getUsername
      (authResponse.AuthenticationResult.AccessToken)

    return success(response)
  } catch (error) {
    console.error(error)
    return failure(error)
  }
}
```

Figura 26. Función de Sign In en la Lambda *company-auth-lambda*. Fuente: Elaboración propia



En la Figura 26 se muestra el proceso para realizar el inicio de sesión utilizando Cognito. Tras establecer los parámetros indicando el Flow para el inicio de sesión, el ClientId (que indica el grupo de usuarios al que acceder) y los datos de autenticación, se realiza la autenticación. La respuesta de la llamada no incluye el nombre del usuario, por lo que posteriormente se realiza una nueva llamada (mostrada en la Figura 27) para obtener los datos del usuario que ha iniciado sesión. Este nuevo dato se añade a la respuesta y se responde satisfactoriamente a la solicitud.

```
const getUsername = async (accessToken) => {  
    let userResponse = await cognitoIdentityProvider.getUser  
        ({ AccessToken: accessToken }).promise()  
    return userResponse.Username  
}
```

Figura 27. Función para obtener el nombre de usuario en la Lambda *company-auth-lambda*. Fuente: Elaboración propia

Finalmente, al recibir la respuesta en la aplicación se deserializa el JSON y se comprueba que no se ha producido un error. Si el proceso se ha completado satisfactoriamente, se almacenan los datos en el *Storage* correspondiente. Este paso final se muestra en la Figura 28.

```
Dictionary<string, string> response =  
JsonConvert.DeserializeObject<Dictionary<string, string>>(request.text);  
if (request.responseCode == 200)  
{  
    AuthStorage.SetValues(response["Username"],  
        response["AccessToken"], response["RefreshToken"]);  
    callback?.Invoke(true, null);  
}  
else  
{  
    callback?.Invoke(false, response["message"]);  
}
```

Figura 28. Gestión de la respuesta de inicio de sesión en la aplicación. Fuente: Elaboración propia

#### 5.4.11.2. Get World

La función de Get World sirve para cargar un mundo y jugarlo, sin necesidad de almacenarlo en el dispositivo. La función Download World utiliza Get World pero, en lugar de mantener

el mundo en memoria, lo almacena en el dispositivo. La Figura 29 muestra la petición GET, con la que se envía el access token del usuario y el nombre del mundo a obtener.

```
string url = BaseUrlCloudStorage + UrlWorld + "?accessToken=" + token +
"&worldName=" + worldName;
UnityWebRequest request = UnityWebRequest.Get(url);
yield return request.SendWebRequest();
```

Figura 29. Petición para obtener un mundo. Fuente: Elaboración propia

En la Lambda *game-pcg-worlds-lambda* se importa *axios* además de *aws-sdk*. El nuevo paquete, al no estar integrado como el SDK, se ha añadido manualmente a la Lambda. A partir de aquí se configura la región y se crea un objeto de S3 para realizar las llamadas, configurando las credenciales de acceso como administrador.

```
const axios = require('axios');
const AWS = require('aws-sdk')
AWS.config.region = 'eu-west-3';

var s3 = new AWS.S3({ accessKeyId: AWSAccessKey, secretAccessKey: AWSSe
cretKey });
```

Figura 30. Inicialización de la Lambda *game-pcg-worlds-lambda*. Fuente: Elaboración propia

La función para obtener el mundo, visible en la Figura 32, empieza comprobando que exista el nombre del mundo, evitando que se suba un archivo sin nombre. Posteriormente se obtiene el prefijo para formar la clave del archivo en S3. En esta función (Figura 31) se realiza una llamada con *axios* a la Lambda *company-auth-lambda*, en la que se valida el access token y se obtiene el nombre del usuario, necesario en el prefijo.

```
const getPrefix = async (token) => {
  try {
    let response = await axios.get(BaseUrlAuth + 'username?accessToken='
+ token)
    return response.data.username + '/worlds/'
  } catch (err) {
    console.error(err)
  }
}
```

Figura 31. Función para obtener el prefijo de un archivo de S3 en la Lambda *game-pcg-worlds-lambda*.

Fuente: Elaboración propia

```
const getWorld = async (token, worldName) => {
  if (!worldName) return failure('Missing World Name')
  try {
    let prefix = await getPrefix(token)
    let params = {
      Bucket: bucketName,
      Key: prefix + worldName + fileExtension
    };
    let response = await s3.getObject(params).promise()
    let world = Buffer.from(response.Body).toString()
    return success({ world })
  } catch (error) {
    console.error(error)
    return failure(error)
  }
}
```

Figura 32. Función para obtener un mundo en la Lambda *game-pcg-worlds-lambda*. Fuente: Elaboración propia

Tras terminar las comprobaciones, se establecen los parámetros y se obtiene el archivo. Posteriormente, se devuelve el mundo codificado como string en Base64. En la aplicación (Figura 33), se recoge el mundo y se decodifica para obtener un array de bytes. Finalmente se guarda el mundo como *MemoryStream* y se llama al callback, que cargará el mundo para poderlo jugar.

```
Dictionary<string, string> response = JsonConvert.DeserializeObject<Dic
tionary<string, string>>(request.downloadHandler.text);

if (request.responseCode == 200)
{
    byte[] world = Convert.FromBase64String(response["world"]);
    CloudWorldStorage.SetWorld(new MemoryStream(world));
    callback?.Invoke(true, null);
}
else
{
    callback?.Invoke(false, response["message"]);
}
```

Figura 33. Gestión de la respuesta de obtención del mundo en la aplicación. Fuente: Elaboración propia

### **5.4.12. Escalabilidad**

Una de las características clave del backend desarrollado es su escalabilidad. Esta propiedad viene proporcionada por AWS, junto con la seguridad de los datos del usuario y la durabilidad de los archivos almacenados. Aun así, en este apartado se realiza un pequeño análisis de las características del sistema que permiten que escale adecuadamente.

Todos los servicios pueden incrementar sus límites de uso, asumiendo los costes correspondientes. Aun así, algunos de estos disponen de capacidades máximas, limitando la potencial escalabilidad del sistema. Sin embargo, en el análisis se considera que una solución es escalable si el límite es suficientemente alto. Además, en la mayoría de ellos, se podría rediseñar la forma en la que se usa para conseguir un mejor rendimiento y no sobrepasar los máximos existentes.

La adaptabilidad para satisfacer necesidades mayores afecta a todos los servicios utilizados. Esto implica que no influye únicamente a las funcionalidades ofrecidas para ser consumidas por los frontends, sino que también deben escalar los elementos que son utilizados por los desarrolladores. Por lo tanto, el análisis incluye si el backend puede soportar tanto mayores cargas de trabajo como equipos de desarrollo más grandes.

Los servicios utilizados para dar soporte a los desarrolladores han sido CodeCommit y CloudWatch. El primero permite crear hasta 1000 repositorios de forma predeterminada, y una cantidad ilimitada bajo pedido. También ofrece almacenamiento, peticiones Git y usuarios activos sin límites. Del mismo modo, CloudWatch ofrece registros, peticiones y métricas sin limitaciones.

Respecto a los servicios para apoyar a las aplicaciones o juegos, restan los siguientes:

- Cognito: permite usuarios mensuales activos ilimitados.
- S3: ofrece 100 buckets de forma predeterminada y hasta 1000 bajo demanda. El tamaño máximo de un único objeto es 5TB. No hay límite para el almacenamiento total en un bucket.
- API Gateway: puede gestionar un número ilimitado de solicitudes, para cualquiera de los tipos de API.
- Lambda: cada función puede disponer de hasta 10 GB de memoria para la ejecución del código. Aun así, no existe un máximo de solicitudes que puede gestionar.

A pesar de que S3 dispone de un número máximo de buckets, es una cantidad muy elevada. Teniendo en cuenta el diseño de utilizar un bucket por aplicación, incluso el límite predeterminado de buckets (100) debería ser suficiente. Aun así, en el caso extremo de crear más aplicaciones, se podría rediseñar fácilmente el sistema para que múltiples frontends compartieran un bucket o incluso utilizar todos el mismo.

#### **5.4.12.1. Rendimiento**

Además de los límites de uso mencionados, es adecuado analizar las diferentes cargas de trabajo que pueden soportar algunos servicios. A pesar de los altos límites en capacidades, un rendimiento insuficiente puede provocar tiempos de espera o incluso errores. Las soluciones estudiadas son S3, API Gateway y Lambda, ya que son los componentes del sistema que pueden recibir solicitudes constantes, en cantidades muy variables.

S3 admite más de 3500 solicitudes por segundo para agregar datos y 5500 solicitudes por segundo para recuperarlos. Las solicitudes se gestionan en paralelo, y estas tasas se aplican por prefijo. Esto significa que, utilizando prefijos, se puede escalar un bucket para poder gestionar casi cualquier cantidad de solicitudes por segundo. Además, el número de buckets en la cuenta no afecta a su rendimiento.

API Gateway puede gestionar cualquier nivel de tráfico, con un máximo (incrementable bajo demanda) de 10000 peticiones por segundo. Estas peticiones únicamente incluyen el establecimiento de la conexión, por lo que se pueden tener una cantidad muy superior de conexiones establecidas. Además, el servicio permite configurar una caché para responder directamente a solicitudes sin necesidad de llamar de forma repetida a otros servicios. Responder solicitudes utilizando esta caché reduce el tiempo de respuesta, así como la carga de trabajo en otros servicios.

Las funciones Lambda permanecen inactivas mientras no son necesarias, y se crea una instancia cuando se recibe un evento de invocación. No existe un límite de solicitudes que pueda gestionar, debido a que puede crear nuevas instancias de la misma Lambda en caso de no disponer de capacidad suficiente para realizar todas las ejecuciones. El único “inconveniente” de esta característica es que las Lambdas no pueden asegurar que se gestione una solicitud con un estado de Lambda concreto. En cambio, utilizar múltiples instancias ofrece grandes beneficios: un alto rendimiento constante y escalabilidad fluida, creando nuevas instancias cuando es necesario y destruyéndolas cuando no se les da uso.

El análisis concluye, finalmente, con un claro resultado: el backend puede escalar y soportar decenas, incluso centenares, de miles de usuarios sin problemas. La mayoría de servicios se adaptan de forma automática a la carga necesaria en cada momento. No obstante, algunos casos requieren un pequeño ajuste en la configuración para su correcto funcionamiento. Esto únicamente sucede en situaciones poco comunes como, por ejemplo, necesitar una gran cantidad de memoria para ejecutar una Lambda.

## 5.5. Análisis de las iteraciones

### 5.5.1. Primera iteración

**Planificación:** el objetivo de esta iteración es analizar las diferentes alternativas para el desarrollo del proyecto y elegir cuál de ellas utilizar.

**Análisis de riesgos:** es la espiral más arriesgada. Esto no se debe a su dificultad, sino a las consecuencias que puede tener una mala decisión causado por un análisis deficiente. Este caso afectaría al resto del proyecto severamente, por lo que se debe asegurar elección.

**Desarrollo:** después de analizar las diferentes opciones, se ha escogido Amazon Web Services como plataforma para desarrollar el backend.

**Evaluación:** se considera que AWS es la mejor decisión para este caso y debería permitir el desarrollo del resto del proyecto sin problemas. Aun así, algunas de las otras opciones eran muy similares a la escogida, por lo que tomar otra decisión no significaría un cambio drástico en la implementación.

### 5.5.2. Segunda iteración

**Planificación:** la segunda espiral se focaliza en crear la cuenta en AWS y configurar los servicios que necesita el backend diseñado.

**Análisis de riesgos:** no se identifica ningún riesgo notable en esta iteración. El mayor problema que puede surgir es configurar incorrectamente algún servicio y tener que volver a hacerlo más adelante.

**Desarrollo:** se han configurado sin problemas los servicios planificados para este momento: Cognito, S3 y CodeCommit.

**Evaluación:** se han establecido las bases para el desarrollo del resto del proyecto. Con los servicios configurados, el siguiente paso es crear la aplicación que los consuma.

### 5.5.3. Tercera iteración

**Planificación:** la meta de esta espiral es instalar y preparar Unity para poder consumir todos los servicios. También se incluye la implementación del primero de ellos: Cognito.

**Análisis de riesgos:** el único riesgo destacable es que aparezca algún problema inesperado durante la implementación. Esto implicaría una inversión de tiempo superior a la esperada.

**Desarrollo:** se han cumplido los objetivos satisfactoriamente. También se ha añadido esta documentación y el proyecto de Unity a CodeCommit para su control de versiones.

**Evaluación:** la documentación y ejemplos han sido suficientes para implementar las funcionalidades sin dificultades. Finalmente ha tomado un tiempo ligeramente inferior al planificado. La aplicación ya permite crear cuenta e iniciar sesión, por lo que se puede continuar con la subida de archivos asociados a un usuario concreto.

### 5.5.4. Cuarta iteración

**Planificación:** el propósito de esta iteración es permitir la subida y descarga de un archivo creado desde Unity, almacenándolo en S3.

**Análisis de riesgos:** de igual manera que en la espiral anterior, el mayor riesgo es encontrar algún contratiempo durante la implementación. En este caso es un riesgo ligeramente superior, ya que no es un servicio independiente al tener que utilizar los tokens obtenidos de Cognito.

**Desarrollo:** la autorización de solicitudes de S3 ha presentado más problemas de los esperados, pero finalmente se ha llegado a una solución considerada como buena (y que se mejorará en la siguiente espiral). También se ha encontrado la necesidad de implementar funcionalidades que no se habían planificado inicialmente, como listar los archivos de un usuario o eliminar un archivo. Además, se ha tenido que implementar una funcionalidad que realmente pertenece a la espiral anterior: la obtención de nuevos tokens de acceso cuando caducan. Hasta llegar a esta espiral y darles un uso a los tokens, no se había presentado la necesidad de actualizarlos, por lo que no se había planificado con anterioridad.

Estos factores se han sumado a la necesidad de reestructurar parte de la documentación y elaborar apartados no previstos inicialmente. Todo esto ha supuesto una inversión de tiempo considerablemente superior a la esperada. Aun así, se han cumplido los objetivos establecidos y se preparado la documentación para la segunda entrega.

**Evaluación:** el núcleo de funcionalidad de la aplicación ya está implementada. Las tareas restantes son, esencialmente, mejoras con diferentes objetivos, como incrementar la seguridad o proporcionar funcionalidades adicionales a los usuarios.

### 5.5.5. Quinta iteración

**Planificación:** el objetivo de esta espiral es migrar el acceso a los servicios al propio backend, aumentando así la seguridad y separando aún más las funcionalidades.

**Análisis de riesgos:** el principal riesgo de esta iteración radica en la necesidad de utilizar servicios nuevos y tener que programar código que se ejecutará en la nube, dificultando testear las funcionalidades y solucionar potenciales errores. Se ha reservado una cantidad de tiempo considerada suficiente para realizar la espiral, pero podría tomar más de la cuenta en caso de encontrar muchos problemas inesperados. Esto supondría disponer de menos tiempo para las iteraciones restantes o incluso tener que suprimir alguna de las tareas. Aun así, se espera poder completarla en el tiempo planificado incluso si surge algún pequeño contratiempo.

**Desarrollo:** la migración de cada funcionalidad ha sido un proceso relativamente mecánico una vez realizados los primeros cambios, por lo que ha tomado cierto tiempo, pero no ha sido un proceso complejo. El cambio de lenguaje ha supuesto una pequeña barrera inicial, pero ha acabado acelerando el proceso una vez superada. A pesar del contratiempo encontrado respecto a los callbacks de las funciones Lambda, se han podido migrar todas las funcionalidades en un tiempo similar al planificado. Probar las funcionalidades en la nube es más complejo que en local, pero CloudWatch ha facilitado esta tarea y se han podido solucionar los problemas en tiempos razonables.

**Evaluación:** al terminar esta espiral, el proyecto ya cumple con todos los requerimientos establecidos. Se han cumplido los objetivos satisfactoriamente y el objetivo de las tareas restantes es mejorar el producto y la documentación del proyecto.



### 5.5.6. Sexta iteración

**Planificación:** la finalidad de la sexta iteración es incrementar las funcionalidades disponibles respecto a las cuentas de usuario, permitiendo cambiar la contraseña y recuperarla en caso de olvidarla.

**Análisis de riesgos:** el tiempo disponible para terminar el trabajo es cada vez más reducido, y esto es el mayor riesgo que se espera de esta espiral. Se deben implementar las dos funcionalidades manteniendo la calidad del resto del proyecto, pero debe terminar con suficiente margen como para poder revisar y pulir todo el proyecto. Por otro lado, la funcionalidad de recuperar la contraseña puede suponer algún problema, ya que es una acción que requiere dos pasos y es un proceso que no ha sido necesario hasta ahora.

**Desarrollo:** se han completado las dos funcionalidades satisfactoriamente, y en un tiempo inferior al esperado. La implementación es similar a otras funcionalidades, por lo que se ha podido realizar de forma simple, incluso la recuperación de contraseña que supone dos pasos.

**Evaluación:** con esta iteración termina la parte práctica del proyecto. Se dispone de un tiempo algo superior al esperado para la última espiral, que se invertirá en terminar y mejorar la documentación.

### 5.5.7. Séptima iteración

**Planificación:** en la última espiral se terminará la documentación del proyecto y se preparará para su entrega.

**Análisis de riesgos:** la iteración no supone ningún riesgo destacable, ya que es un tiempo reservado para la finalización de la documentación. En función de cómo se invierta este tiempo restante, se obtendrá un resultado de mayor o menor calidad.

**Desarrollo:** se ha completado la documentación final y preparado para la entrega. Se han pulido algunos aspectos de esta, reestructurado algunas partes y añadido apartados nuevos.

**Evaluación:** con esta fase, termina el proyecto y está listo para su entrega final.



## 6. Conclusiones

La primera conclusión destacable es que la decisión tomada respecto a la plataforma sobre la que desarrollar el backend ha sido acertada. AWS ha permitido un desarrollo fluido en el que se han cumplido todos los objetivos establecidos. Además, se ha completado el proyecto sin ningún coste extra a los mencionados en el presupuesto.

Se ha demostrado que una aplicación, concretamente un juego en este caso, puede comunicarse eficazmente con el backend desarrollado. Esto implica que el producto está listo para utilizarse para juegos comerciales, cubriendo diversas necesidades tanto para los desarrolladores como los usuarios finales. Como se menciona en el apartado siguiente, el producto puede expandirse de diversas maneras, incrementando así su valor.

Durante el desarrollo se ha invertido mucho tiempo en implementar las funcionalidades en Unity para después migrarlas a funciones Lambda. A pesar de que el conocimiento adquirido en el primer paso ha ayudado a la migración posterior, con el cambio de lenguaje se han tenido que rehacer de cero. Desarrollar las funcionalidades en las Lambdas desde el principio hubiera tomado más tiempo de aprendizaje, pero se hubiera terminado en un tiempo inferior al que se ha necesitado con el proceso seguido. Por esta razón se considera que este proceso ha sido innecesario, y el tiempo extra podría haberse dedicado a crear más funcionalidades o utilizar nuevos servicios.

El proyecto concluye con un backend escalable, seguro y funcional. Además, el bajo coste y el cobro por uso de AWS permite que el backend crezca con el negocio al que respalda. El proyecto ha sacado provecho de los servicios de Amazon para cumplir con los objetivos establecidos y obtener un producto flexible que puede adaptarse a casi cualquier tipo de juego o aplicación.

Aunque en el proyecto el backend se consume desde Unity, este puede ser utilizado por cualquier juego desarrollado en cualquier motor. La migración de las funcionalidades a Lambda a través de API Gateway permite que únicamente sean necesarias peticiones HTTP para poder acceder a cualquiera de los servicios ofrecidos. De este modo, la aplicación no conoce realmente sobre qué plataforma se ha creado la infraestructura, reduciendo así el acoplamiento entre el juego y el backend.

Finalmente, se ha logrado completar satisfactoriamente todos los requerimientos establecidos, incluso los de baja prioridad.



## 7. Posibles ampliaciones

La ampliación más simple y evidente reside en incrementar las funcionalidades del backend. Esto incluye permitir la modificación de datos del usuario (además de la contraseña), permitir a los usuarios desactivar su cuenta, almacenaje de otros archivos para los juegos... Estas ampliaciones mejorarían la experiencia del usuario y les ofrecería más control y características nuevas.

Dado que las cuentas son independientes de los juegos, se podría crear una plataforma (web o aplicación de escritorio) desde donde los usuarios gestionarían y modificarían la información. De este modo en los juegos sólo se mantendría el inicio de sesión, o incluso se eliminaría si se permite el inicio de sesión en un cliente que agrupe todos los juegos. Esto supondría un incremento en la comodidad del usuario, que dispondría de una plataforma única e independiente donde realizar sus gestiones de cuenta. Además, supondría un mayor grado de especialización, separando diferentes funcionalidades en diferentes plataformas.

Del mismo modo que se pueden incrementar las funcionalidades de la autenticación y el guardado en la nube, se pueden añadir nuevas funciones que utilicen otros servicios. Se pueden crear bases de datos con DynamoDB para guardar, por ejemplo, estadísticas de cada usuario. AWS también puede alojar incluso servidores dedicados para los juegos, permitiendo añadir opciones multijugador con el mismo backend.

Actualmente el backend mantiene las credenciales (accessKey y secretKey) en la Lambda y no llegan nunca al dispositivo del usuario, por lo que se mantienen de forma segura. Aun así, esta seguridad puede incrementarse incluso más eliminando las credenciales totalmente del código. Para ello pueden guardarse en el almacén de parámetros dentro del servicio de AWS Systems Manager, desde donde estos se enviarían a la Lambda cuando fuera necesario. De este modo el código, además de inaccesible, no contendría información sensible.

Los emails enviados actualmente (verificación de cuenta y código de confirmación) son los predeterminados de AWS. Se ha usado el propio Cognito para enviarlos, por lo que la personalización se limita a modificar el mensaje. Una mejora sería utilizar el servicio de Amazon Simple Email Service y enviar correos personalizados. Estos emails tendrían un origen con dominio personalizable y utilizarían plantillas en HTML para dar forma a su contenido.

Durante el desarrollo, las publicaciones de API y Lambda se han realizado manualmente. Para mejorar este proceso y facilitar las publicaciones, se puede crear un canalizador usando CodePipeline. Este servicio se encargaría de coger el código de CodeCommit, realizar las acciones necesarias (como una build) y finalmente publicar los cambios. De este modo, se vincularía una rama del repositorio de modo que, al subir cambios en ella, se ejecute el pipeline creado y se publiquen los recursos.

Un cambio que ayudaría a la rentabilidad del proyecto es utilizar CloudFormation o CDK (Cloud Development Kit). Ambas son maneras de definir la infraestructura del backend para poder crear (o eliminar) y configurar todos los servicios necesarios para la aplicación. CloudFormation los define utilizando plantillas, mientras que CDK lo hace mediante código con seis lenguajes disponibles para ello. Con cualquiera de estos métodos, todo el proceso de establecer la infraestructura se reduce a ejecutar un simple comando de consola. Esto permitiría reducir enormemente el tiempo necesario para crearlo e incluso facilitar su remuneración.

## 8. Bibliografía

- [1] P. Pérez Cesari, «Reportan problemas en los servidores de Fall Guys,» *LevelUp*, 19 Agosto 2020.
- [2] tModLoader, «v0.11.7.2 Update,» Steam, 22 Mayo 2020. [En línea]. Available: <https://store.steampowered.com/news/app/1281930?emclan=103582791467148542&emgid=2188132457780082266>. [Último acceso: 8 Octubre 2020].
- [3] TechTerms, «Backend,» [En línea]. Available: <https://techterms.com/definition/backend>. [Último acceso: 10 Enero 2021].
- [4] S. C. Yadav y S. K. Singh, An introduction to client/server computing, New Delhi: New Age International, 2019.
- [5] Amazon Web Services, «¿Qué es la informática en la nube?,» [En línea]. Available: <https://aws.amazon.com/es/what-is-cloud-computing/>. [Último acceso: 2021 Enero 11].
- [6] E. D. Hardt, «The OAuth 2.0 Authorization Framework», RFC 6749, DOI 10.17487/RFC6749, Octubre 2021. Available: <https://www.rfc-editor.org/info/rfc6749>.
- [7] Amazon Web Services, «Amazon Cognito,» [En línea]. Available: <https://aws.amazon.com/es/cognito/>. [Último acceso: 4 Marzo 2021].
- [8] Amazon Web Services, «Amazon S3,» [En línea]. Available: <https://aws.amazon.com/es/s3/>. [Último acceso: 4 Marzo 2021].
- [9] Amazon Web Services, «CodeCommit,» [En línea]. Available: <https://aws.amazon.com/es/codecommit/>. [Último acceso: 23 Marzo 2021].

- [10] Amazon Web Services, «Amazon API Gateway,» [En línea]. Available: <https://aws.amazon.com/es/api-gateway/>. [Último acceso: 4 Mayo 2021].
- [11] Amazon Web Services, «AWS Lambda,» [En línea]. Available: <https://aws.amazon.com/es/lambda/>. [Último acceso: 6 Mayo 2021].
- [12] Amazon Web Services, «Amazon CloudWatch,» [En línea]. Available: <https://aws.amazon.com/es/cloudwatch/>. [Último acceso: 7 Mayo 2021].
- [13] A. Evans, «MediaMolecule Case Study,» Amazon Web Services, 2014. [En línea]. Available: <https://aws.amazon.com/es/solutions/case-studies/mediamolecule/>. [Último acceso: 15 Enero 2021].
- [14] Media Molecule, 2006. [En línea]. Available: <https://www.mediamolecule.com/about/press>. [Último acceso: 22 Enero 2021].
- [15] B. Azoulay, «Rainbow Six Siege uses Azure to deliver immersive multiplayer games globally,» Microsoft, 16 Marzo 2018. [En línea]. Available: <https://customers.microsoft.com/en-ca/story/ubisoft-media-telco-azure>. [Último acceso: 16 Enero 2021].
- [16] Ubisoft, «1000marcas,» 2017. [En línea]. Available: <https://1000marcas.net/ubisoft-logo/>. [Último acceso: 22 Enero 2021].
- [17] M. Pastor, «Halfbrick uses Firebase Predictions to boost retention by 20%,» featured customers, [En línea]. Available: [https://cdn.featuredcustomers.com/CustomerCaseStudy.document/firebase\\_halfbrick\\_None.pdf](https://cdn.featuredcustomers.com/CustomerCaseStudy.document/firebase_halfbrick_None.pdf). [Último acceso: 20 Enero 2021].
- [18] Halfbrick, 2001. [En línea]. Available: <https://www.halfbrick.com/>. [Último acceso: 22 Enero 2021].
- [19] B. Boehm, «A Spiral Model of Software Development,» Computer, 1988, pp. 61-71.



- [20] ASPgems, «Metodología de desarrollo de software (III) – Modelo en Espiral,» 5 Abril 2019. [En línea]. Available: <https://aspgems.com/metodologia-de-desarrollo-de-software-iii-modelo-en-espiral/>. [Último acceso: 26 Enero 2021].
- [21] Amazon Web Services, Obtaining assemblies for the AWS SDK for .NET.
- [22] Unity, «PlayerPrefs,» 16 Marzo 2021. [En línea]. Available: <https://docs.unity3d.com/ScriptReference/PlayerPrefs.html>. [Último acceso: 21 Marzo 2021].