UNIVERSITAT DE BARCELONA

Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

SEGUIMENT DE PARTITS D'E-SPORTS EN VIU UTILITZANT COMPUTACIÓ SERVERLESS

Salvador Mira Morillas

Director: Alberto Guerra, Eloi Puertas
Realitzat a: Departament de
Matemàtiques i Informàtica

Barcelona, 24 de Gener de 2023

Índex

1. Introducció

League of Legends és un videojoc d'estil MOBA (Multiplayer Online Battle Arena), desenvolupat per l'empresa Riot Games, on s'enfronten dos equips de 5 jugadors cadascun amb l'objectiu de destruir la base del rival. Al tenir cada jugador un rol molt concret dins la partida, és similar estructuralment als esports tradicionals més populars del món, com el bàsquet o el futbol, així doncs, al llarg dels últims 10 anys s'ha convertit en l'esport electrònic més jugat i visualitzat de la indústria del videojoc. Així doncs, és molt important que un fenomen actual d'aquest calibre es pugui seguir correctament i de les maneres més diverses possibles.

Per desgràcia, l'única manera de poder seguir un partit en directe d'aquest esport electrònic és a través de les plataformes de streaming de Twitch o Youtube. O en el seu defecte a la web oficial lolesports.com, on es retransmeten les senyals de Youtube i Twitch a més a més d'ensenyar les dades dels equips i els jugadors del partit que s'està jugant en directe. Com que només es pot seguir en vídeo, m'he trobat en moltes situacions en les que no podia mirar-lo, ja sigui perquè estava fent altres coses o les dades mòbils no donaven per més. Em va semblar una genial idea cobrir aquesta necessitat amb el TFG.

Primer de tot s'ha hagut de mirar si era factible dur a terme aquest projecte, així que he hagut de comprovar si cadascuna de les parts que el formen presentava alguna dificultat. Com que ja he treballat anteriorment amb Android i Firebase, sabia que no tindria problema amb la part de l'aplicació (o frontend) i la base de dades, llavors havia de comprovar si hi hauria algun inconvenient amb l'obtenció de dades i el processament d'aquestes.

Per a poder retransmetre un partit en directe hi ha dues opcions, o es busca una manera en la que un programa pugui anar consultant les dades o hi ha una persona mirant cada partit i indicant què passa a cada moment. La intenció en aquest projecte era implementar la primera. Això es podia fer o bé fent webscraping de la web de lolesports.com, que pot ser perillós ja que a la mínima que canviï el codi html es trenca l'aplicació; o bé buscant una API on pugui consultar les dades que necessiti el projecte.

Amb una mica de recerca s'ha vist que existeix una API pública que permet consultar l'estat dels partits en un moment concret, però es necessita especificar l'id del mapa del què vull obtenir les dades. Per desgràcia, no és una cosa pública, així doncs he hagut de buscar i parlar amb empleats de Riot Games a veure si em poden ajudar. Al final m'he trobat amb l'Alberto Guerra, al qual li estic molt agraït, que no només s'ha compromès a ajudar-me i proporcionar-me accés a les APIs privades de Riot, sinó que ha accedit a la meva petició de tutelar-me el TFG ja que

al ser un projecte estrictament relacionat amb la seva empresa, crec que és molt important tenir l'opinió i l'orientació professional d'algú com ell.

Addicionalment, per tal de poder fer servir les dades obtingudes a través de les APIs de forma eficient i en temps real, s'ha investigat la computació servless i com utilitzar-la i adaptar-la per a aquest projecte.

1.1. Definició d'objectius i motivació del problema

Com he comentat a la introducció, només es pot seguir un partit oficial a través d'un canal d'streaming de vídeo. Però i si no podem estar pendents de la imatge? O estem fora de casa o en algun lloc sense connexió WiFi i depenemt de les dades mòbils? I si la senyal de les dades mòbils és molt limitada o el límit que tenim per gastar és molt curt?

L'objectiu principal d'aquest treball és posar una solució real a aquest problema. Així doncs, mirant totes les maneres diferents que hi ha de transmetre i/o seguir un partit de bàsquet o de futbol, la que millor es pot adaptar als problemes exposats és la de fer una aplicació mòbil, ja que:

1. Permet el seguiment en directe dels partits mitjançant notificacions.
2. Estalvia dades mòbils avisant dels moments clau de cada partit sense haver d'estar pendent d'un streaming.
3. Permet mirar en qualsevol moment els esdeveniments que han anat passant al llarg de cada partida sense haver de revisar el VOD (Video On Demand).
4. Guarda les dades de les partides jugades per a tenir un historial que es pot consultar en qualsevol moment.
5. Permet veure quins partits s'han jugat, s'estan jugant en directe i es jugaran en un futur.

Per tant, farem una aplicació en Android amb els següents objectius i característiques:

1. Generar notificacions amb els esdeveniments que estan passant en un partit a partir de les dades obtingudes de les APIs de Riot Games.
2. Tenir 3 llistes fàcilment accessibles de partits:
    a. En directe
    b. Pendents de jugar
    c. Jugats
3. Poder seguir els partits que nosaltres desitgem per tal de rebre notificacions dels esdeveniments que van passant mentres s'estigui jugant en directe.

4. Per a cada partit, tenir una llista cronològica dels esdeveniments que han passat (o que estan passant). Així podrem veure com s'ha desenvolupat un partit ja jugat o com s'està desenvolupant un que està en directe.

2. Antecedents

Ja que hem posat com a exemple esports com el futbol o el bàsquet, anem a observar quines són les diferents maneres en les que es pot seguir un partit de qualsevol lliga oficial:

Televisió	Pública. Pagament. Inclou diferents opcions d'audio a escollir (actualment) ja sigui la pròpia retransmissió del canal corresponent o la sincronització amb la senyal de diversos canals de ràdio.
Ràdio	Rac1 Catalunya Ràdio Radio Marca
Web	MundoDeportivo Marca
Aplicacions mòbils	Sofascore (aplicació multiesport) Onefootball (especialitzat en futbol) Permeten seguir el partit a través de notificacions i permeten tenir un historial de cada partit amb el resum corresponent.

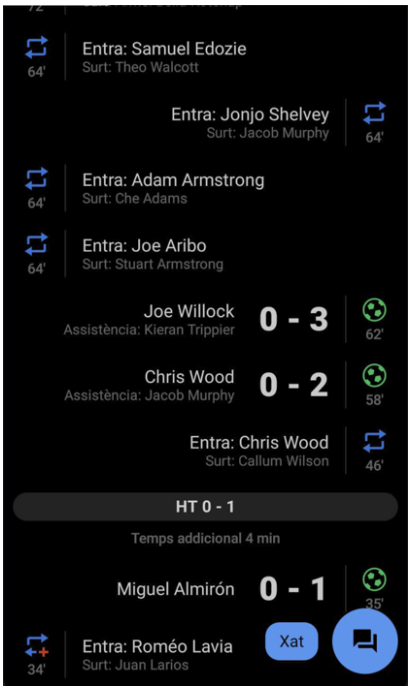
Ara per a poder comparar, anem a veure quines són les diferents maneres de seguir un partit de League of Legends:

Streaming	Twitch: Hi ha un canal dedicat a cada comunitat (espanyola, francesa, alemanya, brasilenya, etc) Youtube
Web oficial	lolesports.com Estat de la partida en directe Integració amb les plataformes d'streaming Youtube i Twitch
Aplicació mòbil	Riot Mobile Permet veure els partits que s'han jugat i els que es jugaran. Només es pot veure el VOD (Video On Demand) dels partits que s'han jugat.

Com es pot veure, no hi ha gaires medis professionals a través dels quals es pugui seguir en directe una partida d'una competició oficial del League of Legends, i els que hi ha disponibles, tots depenen d'una senyal d'streaming.

3. Anàlisis

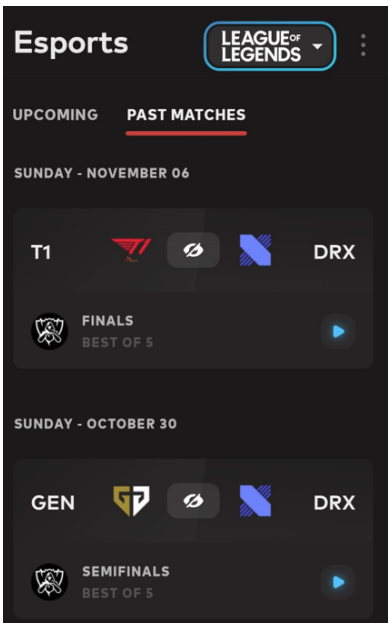
Ja que hem posat com a exemple les aplicacions de futbol per a seguir els esdeveniments dels partits en directe, he agafat l'aplicació Sofascore per a fixar-me en com ho han fet:



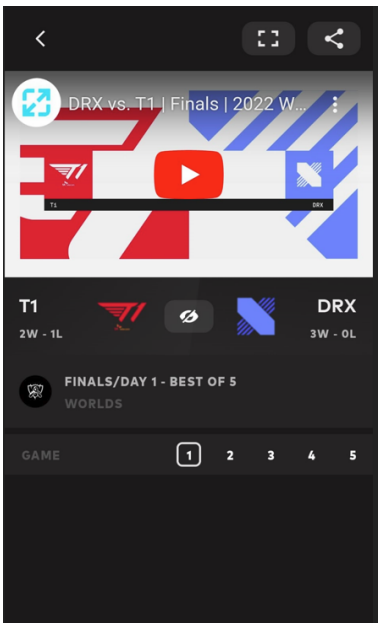
Resum d'un partit de futbol en l'aplicació Sofascore

Podem veure que és una llista d'esdeveniments ordenats de més recent a més antic on es pot veure clarament quina és la puntuació de cada equip i el què ha passat amb el minut corresponent.

També hem pogut veure que Riot Games té una aplicació on es poden seguir en certa mesura els partits de les competicions més prestigioses, així que la meva intenció és fer l'aplicació que sigui fàcilment integrable amb el que ja existeix de l'oficial. Anem a veure què ens ofereix:



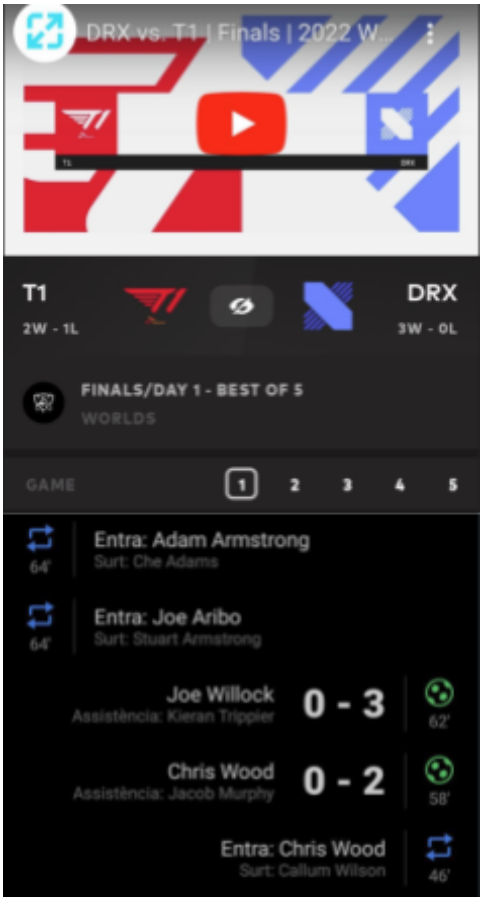
Llista dels partits en l'aplicació Riot Mobile



Pantalla que mostra els mapes d'un partit en l'aplicació Riot Mobile

En la primera imatge tenim un exemple de com organitzar les llistes dels partits, on podem veure que estan organitzades en targetes i quines dades podem ensenyar en cadascuna.

La segona imatge és la pantalla que obtenim un cop seleccionem un dels partits (en aquest cas he agafat el primer) on es pot escollir el mapa que es vol veure. En aquest cas és un Bo5 (Best of 5, el primer que guanyi 3 mapes guanya la partida), així que tenim 5 mapes diferents per escollir. A sota de la selecció del número de mapa hi ha un espai buit, que aprofitarem intencionadament per a posar els esdeveniments que han passat en el mapa seleccionat. Quedaria una cosa similar a la següent:



Exemple de com quedaria la pantalla que mostra els mapes d'un partit amb el resum de cadascun

Ja tenim clars quins són els objectius, què és el que volem i com ho volem gràcies a una investigació prèvia i a les fonts que hem pogut obtenir.

4. Arquitectura i Disseny

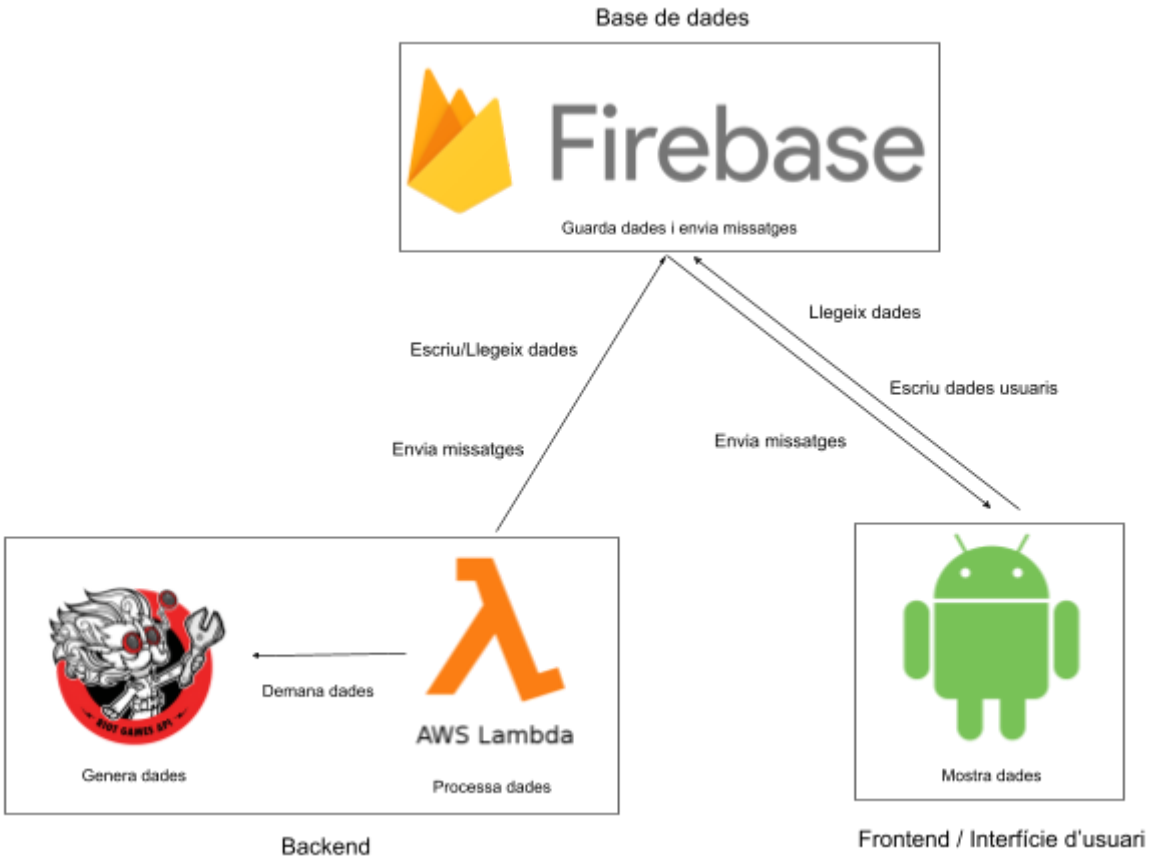
Per a poder seguir els partits en directe, hi ha una API pública de feed.lolesports.com, però per a poder fer les requests corresponents, es necessita l'id del mapa que s'està jugant. Per desgràcia, no hi ha cap tipus d'API pública que pugui proporcionar aquests ids així que el tutor Alberto Guerra, treballador de Riot Games, m'ha pogut aconseguir permís per a poder fer servir la API privada que tenen (amb la documentació corresponent.

Ja tenim com obtenir les dades, així que necessitem alguna manera de processar-les. Ens interessa seguir els partits només quan s'estan jugant, si no hi ha cap en directe, no hem de tenir funcionant cap tipus de servidor. Així doncs, després de mirar escrupulosament quina seria la implementació més adient per a les nostres necessitats, vam decidir fer-la serverless fent ús del sistema de funcions Lambda que ofereix Amazon Web Services. Aquestes funcions processaran les dades obtingudes de les APIs de Riot i en guardaran les resultants a la base de dades.

Per a guardar les dades necessitem una base de dades Tenint en compte que l'objectiu és fer l'aplicació en Android, el servei més còmode d'implementar i de fer servir és el de Firebase, ja que és una base de dades NoSQL i ofereix una fàcil i senzilla integració amb Android, ja que ambdues són de Google. Ja he tingut una prèvia experiència a treballar amb aquesta combinació gràcies a fer l'assignatura de Projecte Integrat de Software, fet que ha facilitat també que m'acabés de decidir. A més a més, ofereix un servei de missatgeria per tòpics, que facilita molt la part de les notificacions.

Ja per acabar aquesta part, només falta la part visual, el frontend. I per això, s'aprofitaran els coneixements que vaig obtenir a l'assignatura de PIS i faré l'aplicació en Android fent servir l'Android Studio. Però en aquest cas, aprofitaré aquest projecte com a excusa per a aprendre a treballar amb Kotlin, un llenguatge de programació que s'executa sobre la màquina virtual de Java i que es fa servir majoritàriament per al desenvolupament d'aplicacions Android. Aquest frontend el que farà serà llegir les dades de Firebase i ensenyar-les de la manera que necessitem, i guardar exclusivament els partits als que estan subscrits cada usuari.

Tenim els 4 elements principals que donen forma al projecte:



4.1. APIs de Riot Games

Per a poder dur a terme aquest projecte, he necessitat moltes dades, metadades dels equips, metadades dels partits, metadades dels jugadors (també anomenats participants bastant sovint); així com les dades dels partits en directe. S'han hagut de filtrar la major part de les dades ja que n'he necessitat només una part. Aquestes han estat proporcionades per dues APIs diferents:

4.1.1. esports-api.lolesports.com

Aquesta és la API interna del departament de e-sports de Riot Games. Les seves request retornen les metadades, en format JSON, de tot l'entorn de e-sports emmagatzemades en la pròpia base de dades de Riot, que consisteixen bàsicament en:

- Partits jugats, per jugar i en directe
- Equips

- Jugadors
- Tornejos
- Lligues

Al ser una API interna, no és d'accés públic, així que l'Alberto m'ha pogut aconseguir una clau (amb permís de la pròpia empresa) per a poder-la fer servir exclusivament per a aquest projecte.

4.1.2. feed.lolesports.com

Aquesta és una API que està disponible per al seu ús públic. Serveix per a obtenir les dades d'un mapa en forma d'snapshots. D'aquí podrem detectar els esdeveniments que passaran al llarg d'un mapa. Aquestes snapshots són instants concrets de la partida i van en intervals de 10 segons, és a dir, en cada interval hi haurà entre 15-20 snapshots (no és una snapshot cada segon). És l'API més important del projecte, però s'ha d'especificar de quin mapa volem les snapshot, i per això és necessària l'API de esports-api.lolesports.com.

4.2. Serverless computing

Ara que ja tenim com obtenir les dades, hem de buscar alguna manera de processar-les. La primera idea era fer el deployment del backend a un servidor de Heroku, ja que és el que hem fet servir en múltiples assignatures del grau, però vam veure que aquest any va passar a ser de pagament i no era una opció. Després de revisar múltiples cops quines eren les nostres necessitats, vam observar que només necessitàvem que el seguiment de partits funcionés precisament quan hi hagués un en directe, ja que no té gaire sentit que estigui intentant seguir un partit quan no hi ha cap jugant-se. Aquest concepte encaixa perfectament amb l'anomenat Serverless computing.

4.2.1. Definició

La informàtica sense servidor (Serverless computing) és un mètode per a proporcionar serveis de backend a diversos usuaris. Un proveïdor d'arquitectura sense servidor permet als usuaris escriure i implementar codi sense que hagin de preocupar-se per a la infraestructura. No es manté cap tipus de recurs en memòria, sinó que quan acaba l'execució del codi, totes les dades generades s'eliminaran automàticament. Quan no hi ha cap aplicació en ús, no s'està fent servir cap tipus de recurs.

Una empresa o particular que vulgui contractar un servei serverless haurà de pagar només en funció de la quantitat de serveis que faci servir, en comptes de pagar per una quantitat fixa de recursos, ja que la funcionalitat anirà escalant automàticament segons les necessitats immediates. Així doncs, aquest model és molt útil per quan no es saben o és molt complicat fer una estimació de la quantitat d'usuaris que faran servir l'aplicació. En comparació amb l'alternativa tradicional, el server de tota la vida, aquesta et permet tenir un cost econòmic molt baix des de l'inici, ja que es paguen els recursos estrictament necessaris, no els que s'esperen per a poder allotjar el número d'usuaris previst.

Una cosa molt important i que val la pena comentar és el manteniment. El fet de que sigui serverless fa que sigui possible fer un deployment del codi o una actualització sense haver d'interrompre el servei, ja que mentres s'actualitza es pot fer servir amb el codi antic, i un cop acabi l'actualització, el proper cop que s'hagi d'utilitzar, ja serà amb el codi actualitzat. A més a més, el desenvolupador només s'ha de preocupar del codi, les dependències d'aquest i els permisos que necessiten les funcions per a poder-se invocar (que han de tenir exclusivament els que necessiten per tema de seguretat), mentres que de tota la resta de la infraestructura se n'encarrega el proveïdor del servei (protegir l'accés als servidors, assegurar-se de fer una gestió correcta dels mateixos, etc.).

4.2.2 Opcions

Ara ja sabem què volem i necessitem, hem estat buscant múltiples opcions i proveïdors i hem trobat els 3 que són més importants:

- **Amazon Web Services Lambda**
- **Azure Functions**
- **Google Cloud Functions**

A efectes pràctics les 3 ofereixen el mateix tipus de servei, però cadascuna té les seves peculiaritats, anem a veure-les:

1. **Llenguatges disponibles i desplegament**
    a. **AWS Lambda**: Permet l'ús natiu de llenguatges com Java, PowerShell, Golang, Node.js, C#, Python i fins i tot Ruby. Mitjançant l'ús d'imatges de contenidors, anomenats capes (layers), es pot fer servir qualsevol tipus de llenguatge de programació. Permet el desplegament de qualsevol funció Lambda simplement creant un arxiu .zip amb el codi i les dependències necessàries.

b. **Azure Functions**: Permet l'ús de C#, JavaScript, Python, PowerShell, F#, Java (v8 i v11) i TypeScript. També permet el desplegament d'aplicacions a partir d'arxius .zip. A més a més permet el desplegament via FTP, Cloud Sync, Local Git, plantilles de JSON i Continuous deployment.
c. **Google Cloud Functions**: Facilita l'escriptura de codi en Node.js, Golang, Java, Python i frameworks de .NET (com C#, F# i Visual Basic). Els paquets de Cloud Functions poden ser desplegats des de Local Machine, Cloud Source Repos, Cloud Console, Source Control i APIs.

2. **Gestió de dependències**
    a. **AWS Lambda**: Les dependències estan controlades molt fàcilment a través del paquet de desplegament de la funció Lambda corresponent. L'entorn de desenvolupament de les funcions Lambda conté llibreries com la de AWS SDK que permet introduir funcions i paquets nous.
    b. **Azure Functions**: Fa servir una Application Performance Management (APM) per a mesurar, controlar i les dependències.
    c. **Google Cloud Functions**: La gestió de dependències depèn del llenguatge de programació escollit.

3. **Emmagatzematge**
    a. **AWS Lambda**: Té un límit per als recursos de computació i emmagatzematge: 1000 execucions concurrents i 75 GB d'emmagatzematge per a funcions i capes. Si es vol ampliar l'espai fins a diversos TB, s'ha de demanar permís.
    b. **Azure Functions**: Es pot obtenir des de 40 TB a 500 TB d'emmagatzematge offline mitjançant una solució al núvol anomenada Data Box.
    c. **Google Cloud Functions**: Permet la gestió de l'emmagatzematge directament des del seu Cloud Storage amb la possibilitat de fer servir Firebase.

4. **Gestió d'identitats i accés (IAM)**
    a. **AWS Lambda**: Permet crear les polítiques IAM necessàries amb rols específics per a un ús personalitzat. En té 3 de principals: Full-Access, Read-Only Access i Role.
    b. **Azure Functions**: Es poden controlar les polítiques de cada funció a través de Resource Based Access Control.
    c. **Google Cloud Functions**: Proveeix un control d'accés molt elevat. Fa servir els recursos de Google Cloud per a crear una política de seguretat unificada a mida que els requeriments progressen.

5. **Activadors (Triggers)**
    a. **AWS Lambda**: Permet la invocació de funcions a través d'HTTPS, a més d'oferir una gran quantitat d'alternatives i serveis d'AWS que poden activar lambdes de múltiples maneres, com la programació horària.
    b. **Azure Functions**: Permet l'activació de funcions fent servir cues, temporitzadors, graelles d'esdeveniments o HTTP.
    c. **Google Cloud Functions**: Gestiona les activacions en dos nivells funcionals diferents: HTTP i Background.

6. **Gestió d'estats**
    a. **AWS Lambda**: Té el que s'anomenen les Step Functions. Aquest mòdul permet registrar la condició de cada funció un cop executada. També permet la creació de màquines d'estat, facilitant així la creació de fluxos més complexes.
    b. **Azure Functions**: Es pot gestionar i automatitzar tasques fent servir Azure Logic i Durable Functions, d'aquesta manera, es poden integrar els dos serveis al núvol.
    c. **Google Cloud Functions**: Integració amb Cloud Composer. Configura les funcions en forma de Directed Acyclic Graphs (DAGs).

7. **Concurrència i temps d'execució**
    a. **AWS Lambda**: Limita el nombre d'execucions concurrents a 1000 i es pot controlar a nivell individual (funció) o a nivell de compte d'usuari. El timeout funcional de cada execució és de 900 segons o 15 minuts.
    b. **Azure Functions**: El número d'activitats i execucions concurrents està limitada a 10X depenent de la quantitat de nuclis de la VM. El temps límit d'execució és de 600 segons o 10 minuts.
    c. **Google Cloud Functions**: Per defecte, només es poden tenir 80 execucions concurrents i el temps límit d'execució varia entre 60 i 540 segons (1 i 9 minuts).

8. **Escalabilitat i disponibilitat**
    a. **AWS Lambda**: Per a poder gestionar ràfegues de trànsit, predetermina el número de funcions a executar depenent de la regió. Aquest número varia entre 500 i 3000 segons la regió.
    b. **Azure Function**: Té dues opcions: una que escala la funció quan aquesta fa timeout després d'un període de temps establert; i una altra que executa les funcions en VM dedicades, que fa que el host on estan les funcions sempre estigui encès i funcionant correctament.
    c. **Google Cloud Functions**: Té escalabilitat automàtica, però que està limitada pels límits de trànsit de dades.

9. **Registres i supervisió (Logging)**
    a. **AWS Lambda**: Supervisa les funcions mostrant les mètriques d'aquestes a través d'Amazon CloudWatch on inclou el número de requests, la latència per request i la quantitat de requests fallides. Automàticament crea un grup de registres per a cadascuna de les funcions Lambda.
    b. **Azure Functions**: Supervisa les funcions amb una aplicació integrada anomenada Azure Application Insights. És opcional i normalment es fa servir un sistema de registre natiu.
    c. **Google Cloud Functions**: Fa servir Google Stackdriver, que té eines per a registrar i reportar errors, a part que es poden veure els temps d'execució, la quantitat d'execucions i l'ús de memòria.

10. **Preus**
    a. **AWS Lambda**: Té una part gratuïta que inclou 1 milió de requests a funcions i 400.000 GB-seg per mes. Després de la part gratuïta, es paga 0,20$ per milió de requests i 0,00001667 per cada GB-seg. GB-seg és la quantitat de RAM feta servir multiplicada pel temps que ha estat en ús.
    b. **Azure Functions**: La part gratuïta cobreix 5000 transaccions i 100.000 requests al mes amb 64 GB de SSD i 5 GB d'emmagatzematge per a arxius. Després pots tenir fins a un any de servei gratuït o escollir un pla que varia entre 0,02$/h i 1,41$/h. També dóna 200$ de crèdit pels primers 30 dies, a part de tenir un pla per a estudiants.
    c. **Google Cloud Functions**: Funciona una mica diferent a les altres dues opcions, ja que té en compte el temps d'execució, la quantitat d'invocacions i quants recursos es dediquen a cada funció. Proveeix de 300$ els primers 90 dies d'ús. Té una part gratuïta que inclou 2 milions de requests. Calcula el temps de computació en els ja anomenats GB-seg i GHz-seg.

Un cop revisats tots i cadascun dels punts, podem arribar a la conclusió que tots els 3 serveis ofereixen un producte molt similar amb unes diferències negligibles per l'escala del projecte que volem implementar. Així doncs, després d'estar uns dies mirant com funciona cadascuna a nivell pràctic, em vaig decidir pels següents motius:

- Em va semblar molt més còmode de treballar i navegar tant visualment com pràcticament.
- Em vaig entendre molt millor amb la documentació.
- Permet la programació de funcions Lambda amb precisió de minuts molt fàcilment, punt que és de vital importància per al projecte, ja que ens interessa que les funcions segueixin el partit a partir de l'hora indicada a les metadades d'aquest.

4.2.3. Funcions Lambda

Ja tenim clar que farem servir les funcions Lambda d'Amazon Web Services per a implementar el nostre backend. Així doncs, he fet servir dos elements claus:

- **Normes d'Amazon EventBridge**: Permeten la programació de funcions Lambda ja sigui per a executar-les un cop a un dia i hora concrets, com per a executar-les de manera recurrent en un interval de temps determinat. Una mateixa norma pot tenir múltiples objectius, és a dir, es poden executar múltiples funcions Lambda diferents a través d'una mateixa norma, o la mateixa funció Lambda amb diferents inputs.
- **Funcions Lambda**: Seran scripts de Python que gestionaran la detecció de partits programats i de partits en viu, així com el seguiment d'aquests últims. Són capaces d'invocar altres funcions Lambda, així com interactuar amb Amazon EventBridge per a treballar dinàmicament amb les seves normes.

4.2.3.1. Com funcionen?

En el nostre cas, les funcions Lambda seran scripts de Python. L'arxiu de codi principal es dirà "lambda_function.py" i el mètode main s'haurà de dir "lambda_handler". Val a dir que són els valors per defecte i que es poden definir amb noms diferents a la configuració de cada funció Lambda, que per qüestions de claredat he decidit no modificar. El mètode main tindrà dos paràmetres d'entrada:

- **Event**: És un document en format JSON que conté dades per a que la funció Lambda pugui processar. Poden ser tant dades que es volen fer servir a la funció (com els paràmetres d'un mètode qualsevol), com dades del servei que ha activat la funció.
- **Context**: És un objecte que permet l'accés a les dades, propietats i entorn d'execució de la funció Lambda. També dóna accés a un mètode que permet consultar la quantitat de milisegons que queden fins que l'execució de la funció faci time-out. Alguns dels exemples més útils:
    - **function_name**: El nom de la funció.
    - **function_version**: La versió de la funció.
    - **invoked_function_arn**: El nom del recurs d'Amazon (Amazon Resource Name) que s'ha fet servir per a invocar la funció.

Es poden invocar manualment (fent un test) o a través de triggers (també coneguts com disparadors o activadors), que poden ser altres funcions Lambda o

les normes ja comentades que permeten programar funcions, entre un ventall molt gran d'opcions i serveis que ofereix el mateix Amazon.

4.2.3.2. Configuració

Les funcions Lambda tenen una gran configurabilitat que permeten adaptar-la al màxim a les necessitats de cada projecte. Les opcions de configuració que farem servir més seran les següents:

- **General configuration**: Inclou la descripció de la funció, la quantitat de memòria RAM que es pot fer servir (min: 128 MB, max: 10240 MB), la quantitat d'emmagatzematge efímer (min: 512 MB, max: 10240 MB) i el temps límit d'execució (timeout) que té un màxim de 15 minuts.
- **Triggers**: Permet veure i modificar els serveis o els recursos que poden activar la funció Lambda. Aquests serveis i recursos es guarden en un document intern amb les dades corresponents, que té una capacitat d'emmagatzematge limitada d'uns 20 MB.
- **Permissions**: Permet veure i modificar els permisos i les polítiques que té la funció Lambda per a poder interactuar amb la SDK d'Amazon Web Services, és a dir, gestiona si pot invocar altres funcions, crear o modificar normes, etc.
- **Layers**: Permet veure i modificar les capes (layers) que té la funció Lambda. Una capa o layer permet a la funció fer servir codi addicional, contingut de llibreries externes i dependències que normalment no estarien en la VM que fa servir AWS per a executar-la. En aquest projecte s'han necessitat per a poder fer servir les llibreries de Firebase i el mòdul de requests.

4.2.3.3. Monitorització

Cada funció Lambda té associada per defecte un grup de registres a CloudWatch, on es podran veure els resultats de cada execució feta i tots els prints i log que s'hagin inclòs al codi. A més a més, té una secció de monitoreig on es poden veure les estadístiques més destacades:

- **Invocations**: Mostra la quantitat de vegades que s'ha invocat la funció.
- **Duration**: Mostra la duració màxima, mínima i mitjana de les invocacions que s'han fet de la funció.
- **Error count and success rate (%)**: Mostra la quantitat d'invocacions que han acabat en error i quin % són del total d'invocacions fetes.
- **Concurrent executions**: Mostra quantes execucions concurrents s'han estat fent.

4.3. Firebase

Per a guardar totes les dades que generi l'aplicació o necessiti consultar, haurem de tenir una base de dades. En aquest cas he escollit fer servir Firebase per diversos motius:

1. És molt fàcil d'integrar amb Android, ja que els dos estan portats per Google.
2. És una base de dades NoSQL, vol dir que els seus elements es guarden en format clau-valor, així que és molt fàcil traduir dels JSON retornats per les APIs de Riot Games.
3. Té una llibreria que és molt fàcil de fer servir en Python. Tot i que després m'he trobat amb algun problema a l'hora d'executar-la a les funcions Lambda d'AWS (que s'ha arreglat fent servir les capes).
4. L'he fet servir ja altres cops a altres assignatures, així que estic molt familiaritzat amb el seu funcionament i les seves capacitats.

4.3.1. Seguretat

Per a evitar que qualsevol persona pugui accedir a les dades guardades a la base de dades, Firebase proporciona sistemes de certificació tant per al backend en Python com per al frontend d'Android. Consisteix en un document JSON amb els tokens, les claus privades i les direccions necessàries per a que l'aplicació pugui accedir als continguts de la base de dades de Firebase (són diferents el JSON necessari pel codi Python i el JSON necessari per Android).

4.4. Aplicació Android

La part visual d'aquest projecte és una aplicació d'Android que s'ha programat fent servir el llenguatge de programació Kotlin i Android Studio. Tot i haver fet una assignatura d'Android en Java, he cregut que seria interessant fer-lo en Kotlin, ja que sempre he volgut aprendre a treballar amb ell i és molt més còmode de fer servir que Java.

4.4.1. Kotlin

Kotlin neix de la idea de superar a Java sense deixar de banda el seu codi, seguint així el mateix concepte de llenguatge orientat a objectes. Així doncs, l'execució de codi escrit en Kotlin està preparada per a funcionar amb la JVM (Java

Virtual Machine) i Android. Inclou l'opció de que el seu codi sigui compilat a JavaScript, amb la finalitat de mantenir un nivell de migració senzill de Java a Kotlin. Per tant, els seus programes poden ser executats en totes i cadascuna de les plataformes que siguin compatibles amb Java, sense la necessitat d'aplicar una recompilació.

4.4.1.1. Característiques principals

- Té una corba d'aprenentatge molt curta, ja que la seva sintaxi és simple i molt fàcil de llegir.
- És interoperable amb Java. Permet combinar el seu codi amb d'altre escrit en Java.
- És molt fàcil migrar de Java a Kotlin i a més a més té un risc molt baix, ja que permet testejar només una part del codi. Així doncs, no fa falta modificar el codi de tot el projecte abans de fer efectiva la migració i anar provant que tot està bé.
- No permet guardar variables amb valor NULL independentment del seu tipus, així doncs elimina per complet els NPE (Null Pointer Exception).
- Permet tenir un constructor principal i alhora gestionar constructors secundaris pels objectes.
- Permet emmagatzemar dades d'un JSON dins d'un mapa (similar als diccionaris de Python).
- És possible treballar amb frameworks o llibreries JQuery, React, Angular, entre d'altres.
- Compta amb un excel·lent suport per a Android Studio.
- Fa servir dues paraules claus a l'hora de declarar variables:
    - "val": S'aplica quan una variable sempre tindrà el mateix valor, o ens interessi que sempre el mantingui.
    - "var": S'aplica quan una variable tindrà un valor que es podrà modificar al llarg de l'execució.

5. Desenvolupament

5.1. Anàlisi de les APIs

Per a poder gestionar tota la informació que ofereixen les APIs de Riot Games, abans he hagut d'analitzar quines dades puc obtenir i en quin format estan i així treure el màxim profit possible. Com ja hem vist abans, estan separades en dues parts: esports-api s'encarrega de les metadades dels equips, partits, lligues, jugadors, etc; mentres que feed s'encarrega de les dades dels mapes i ens permetrà detectar els esdeveniments importants que estan passant per així poder enviar les notificacions corresponents.

5.1.1. esports-api.lolesports.com

5.1.1.1. Scheduled events

Retorna una llista ordenada cronològicament de 500 partits. Té un sistema de paginació que permet navegar per la base de dades i obtenir partits més antics i partits més nous. Si no s'especifica la pàgina al fer la request, retornarà la pàgina de 500 partits completa més recent (la que tingui els partits més nous). Com el seu nom indica, retorna les dades dels partits programats, independentment de que s'hagin jugat com si no.

Utilitat

Com ja hem dit, d'aquesta request podem obtenir les metadades de tots els partits jugats i per jugar, així que ho podrem fer servir en l'aplicació per a saber:

- Quins partits es jugaran i quan (minut, hora, dia, mes i any). El quan és orientatiu ja que molts cops l'hora d'inici depèn del partit que es jugui abans. Com que no tenen una durada de temps fixa, poden acabar molt d'hora o es poden allargar molt, fet que pot adelantar o endarrerir l'hora d'inici del partit que vingui després.
- Quins són els equips que s'enfronten.
- De quina lliga o torneig és el partit i de quina etapa.
- Quants mapes es jugaran (si és al millor de 1, 3 o 5 mapes per exemple, tot i que en algunes lligues es juga al millor de 2, d'aquí que els equips puguin tenir un rècord de victòries, derrotes i empats).
- La informació de cada mapa individual, que farem servir per a poder fer el seu seguiment en directe.
- En quin costat del mapa ha jugat o jugarà cada equip.

Ens proporciona moltes més dades, però pel que volem fer amb el projecte no ens són pas útils, com per exemple:

- Informació sobre els streams de cada partit (no hi ha res en cap partit, així que tampoc es podria aprofitar encara que volgués).
- Informació dels VODs, que són els vídeos dels mapes ja jugats.
- El rècord de cada equip que juga el partit dins de la competició en la que estan jugant (victòries, derrotes i empats).

5.1.1.2. Live events

Retorna la llista dels partits que s'estan jugant en directe. Això no vol dir que el primer mapa hagi començat, sinó que l'esdeveniment com a tal es dóna oficialment per començat. Poden haver múltiples partits tant d'una mateixa lliga o torneig com de diferents jugant-se a la vegada.

Utilitat

L'estructura del JSON obtingut és exactament la mateixa que en la request de Scheduled events, però l'ús que donarem de la informació obtinguda serà completament diferent. Com que les dades de cada partit que hi ha en directe ja les tenim guardades a la base de dades gràcies a la request ja mencionada, aquesta la farem servir exclusivament per a saber quan un partit està en directe per a poder començar el seu seguiment en directe.

5.1.1.3. All teams

Retorna les metadades de tots els equips professionals registrats (actualment i que ho han estat en algun moment de la seva història) a la base de dades de Riot Games, ja siguin equips que estan inactius en l'actualitat, equips fets per a partits d'exhibició i altres equips mal formats o directament buits.

Utilitat

No es pot fer una request passant el id d'un equip com a paràmetre per a obtenir les dades només d'aquest equip, així que si vull obtenir les dades d'un equip en concret, he d'anar per tota la llista comparant l'id corresponent. Això genera molta feina computacional, així que he trobat que era més fàcil i ràpid guardar les dades dels equips a la base de dades i identificar cada equip amb el seu id corresponent. Per a poder fer això, he hagut de filtrar el màxim possible les dades proporcionades per la API, per a eliminar equips que:

- Estan inactius.
- No són de League of Legends.
- Buits o amb dades mal formades.
- Equips d'exhibició

Un cop tenim els equips filtrats i les seves dades guardades a la base de dades, s'utilitzarà la seva informació per a mostrar-la per l'aplicació. La imatge per a veure el seu logo, el nom per a poder veure el nom de l'equip i el codi (no el id) per a l'abreviació que es fa servir a l'hora de veure estadístiques i altres coses en les quals no faci falta mostrar el nom sencer del club (per exemple, el codi de Fnatic seria FNC i el del Barça Esports seria BAR.

5.1.1.4. All players

Retorna les metadades de tots els jugadors professionals registrats (actualment i que ho han estat en algun moment de la seva història) a la base de dades de Riot Games, és igual que estiguin en actiu com si no.

Utilitat

Igual que amb la request que retorna les dades de tots els equips, no es pot passar l'id d'un jugador com a paràmetre per tal d'obtenir les dades concretes d'aquest. Així que he hagut de guardar les dades dels jugadors a la base de dades. Però el fet d'haver guardat ja les dades dels equips m'ha facilitat molt el filtratge de les dades (ja que òbviament no vull guardar jugadors que no estan en actiu) perquè cada equip guarda una llista amb els ids dels seus jugadors. Així doncs només he hagut de guardar els jugadors que estaven en un equip de la base de dades.

Per als objectius que tenim en aquest projecte per a l'aplicació, no ens és gaire útil la informació personal de cada jugador a part del nom d'invocador (en anglès "summoner name") que és el nom clau que escullen els jugadors per als partits, perquè només ens interessa veure les dades dels equips i què fa cadascun en un mapa. Però en el cas que en un futur volgués ampliar la funcionalitat de l'aplicació i fer que es puguin veure les dades dels jugadors, llavors sí que podria treure més utilitat d'aquí, com per exemple agafar els noms reals i la imatge.

5.1.1.5. All leagues

Retorna les metadades de cada lliga registrada a la base de dades de Riot Games, inclou tant les lligues de primer nivell, com la europea (LEC), la coreana (LCK) o la xinesa (LPL), com les lligues regionals o acadèmiques. Puntualitzar que es considera lliga fins i tot les competicions internacionals, com el Mid-Season Invitational (MSI) o Worlds (els mundials).

Utilitat

Un dels extres que tenia pensat afegir a l'aplicació en el cas que fos factible (de temps) era poder-se subscriure a una lliga sencera per a poder així rebre les notificacions de tots els partits que es juguessin. Una altra havia de ser poder filtrar per lligues els partits que s'ensenyen a l'aplicació, ja que es poden acumular molts molt fàcilment.

5.1.1.6. Tournament by id

Retorna les metadades del torneig amb el id passat com a paràmetre registrat a la base de dades de Riot Games. Només es pot fer servir especificant l'id del torneig del qual es vol obtenir informació, no es pot obtenir tots els tornejos de cop. Es defineix com a torneig qualsevol edició de qualsevol de les lligues obtingudes a la request anterior.

Utilitat

No he planejat cap tipus d'utilitat funcional ni en els objectius del projecte ni en els possibles extres per a la informació, però es podrien aprofitar les dades obtingudes per a una millor personalització de l'aplicació, ja que es podria filtrar per tornejos, dins de cada torneig es podria filtrar per fase, etc. Tota aquesta informació es podria mostrar d'alguna manera també com a detall de cada partit per a que l'experiència d'usuari fos la més completa i detallada possible.

5.1.2. feed.lolesports.com

5.1.2.1 Livestats window

Retorna les dades dels equips en múltiples instants (finestres, windows, snapshots o frames) d'un mapa que s'està jugant en directe (o ja s'ha jugat). Per a poder obtenir una resposta exitosa, s'hauran de passar un parell de paràmetres obligatòriament, que són:

1. L'id del mapa (o esportsGameId): Determinarà de quin mapa volem obtenir les dades.
2. Data i hora (format aaaa-mm-ddThh:mm:ssZ) de l'instant que volem obtenir les dades. Aquesta part té múltiples peculiaritats:
    a. Si passem una data i hora anteriors a l'inici de la partida, la request fallarà.
    b. Si passem una data i hora posteriors a la finalització de la partida, la request retornarà les últimes snapshot del mapa.

c. Si no passem cap paràmetre de data i hora, la request retornarà les primeres snapshot del mapa.
d. La part dels segons ha de ser múltiple de 10, és a dir, només s'accepta 00, 10, 20, 30, 40, 50.
e. Cada resposta conté totes les snapshot guardades en la base de dades en intervals de 10 segons. D'aquí que s'hagin de fer servir múltiples de 10 per als segons.

Utilitat

Aquesta és la request més important de tot el projecte, ja que permet el seguiment d'una partida en directe i generar les notificacions i esdeveniments corresponents. Això funcionarà de la següent manera:

1. Cada resultat obtingut conté entre 10 i 20 snapshots, on cada snapshot conté les següents dades destacables per al projecte:
    a. Data i hora de la snapshot, per a calcular el temps del mapa.
    b. L'estat del mapa, que poden ser un total de 3:
        i. in_game: El mapa s'està jugant correctament.
        ii. paused: El mapa està pausat.
        iii. finished: El mapa s'ha acabat de jugar. Sempre l'última snapshot d'una partida tindrà aquest.
    c. L'estat de cadascun dels equips en aquell instant del mapa:
        i. Assassinats.
        ii. Or.
        iii. Torres.
        iv. Inhibidors.
        v. Barons.
        vi. Dracs. Aquests no venen en quantitat, sinó en una llista, ja que hi ha fins a 7 de diferents i són molt importants per al desenvolupament del mapa.
    d. L'estat de cadascun dels jugadors en aquell instant del mapa:
        i. Assassinats
        ii. Morts
        iii. Assistències
        iv. Or
2. Com que les dades que necessitarem de tant els equips com dels jugadors són sempre incrementals (menys la vida actual, currentHealth, i casos molt específics que comentarem més tard), només farà falta mirar els canvis que hi ha hagut entre una snapshot i la anterior per a saber si ha passat quelcom interessant: un assassinat, una torre caiguda, un drac caçat, etc.

Les metadades de tant l'equip blau com de l'equip vermell es podran aprofitar per a definir quins jugadors participen en cada equip i en quina posició. Recordar que les dades de cada equip i jugador es podien obtenir a través d'una request, però un equip pot tenir múltiples jugadors en una posició o un jugador que jugui en diferents posicions, així podrem saber amb exactitud quina és l'alineació concreta per al mapa que s'està jugant (que pot variar entre mapes d'un mateix partit).

Un altre ús interessant que he pogut trobar i que ha estat molt important per a la part de seguiment de partits, és la peculiaritat (ja comentada a la descripció) que si no es passa una data i hora com a paràmetre retorna les primeres snapshots del mapa. Això es pot aprofitar per a obtenir la data i hora d'inici del mapa amb una precisió de milisegons, tot i que no fa falta ser tant escrupulós, ja que la primera snapshot retornada serà la primera de la partida i, com he dit abans, tindrà la data i hora a la que s'ha fet.

5.1.2.2. Livestats details

Retorna les dades detallades de les estadístiques i l'estat de cada jugador del mapa en cada instant concret de la partida (snapshots). Funciona exactament igual que la request anterior, s'ha de passar com a paràmetre el id del mapa i la data i hora amb els segons en múltiples de 10. Òbviament, si posem una data i hora anterior a l'inici del mapa, la request fallarà; si posem una posterior a la finalització del mapa retornarà les últimes snapshot; i si no en posem, es retornaran les primeres snapshot del mapa.

Utilitat

Per desgràcia, les dades que podem fer servir d'aquí per a completar els objectius d'aquest projecte ja les obtenim a la request anterior, així que no podrem aprofitar res directament. Però en el cas que en volgués aprofitar quelcom, es podria fer servir per a obtenir una experiència d'usuari molt més detallada quan estigui seguint un partit activament, ja que ens permet veure:

- El percentatge de mal fet a l'enemic en comparació a la resta del seu equip.
- Les estadístiques del personatge que està jugant cada jugador.
- Els objectes comprats.
- Les habilitats millorades.
- Les runes i maestries que està fent servir.

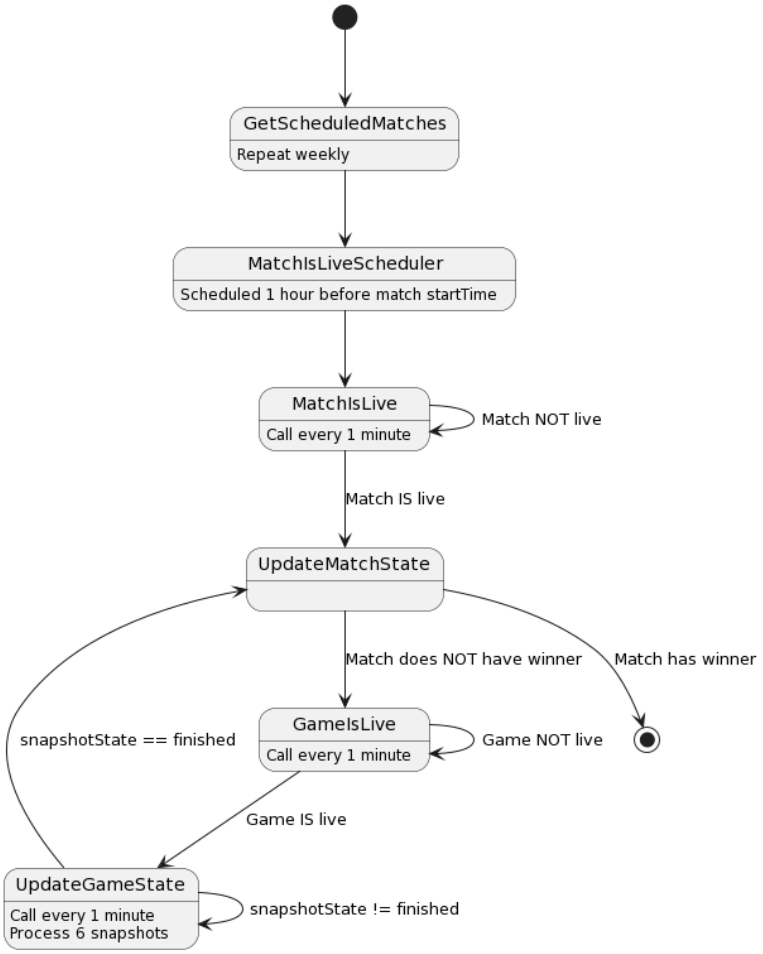
5.2. Funcions Lambda

5.2.1. Organització de les funcions i algoritme

Primer de tot, mirarem periòdicament a la API de Riot games si hi ha partits programats nous. Llavors, una hora abans de l'inici del partit començarem a mirar si el partit està en directe. Això es deu a que la majoria de cops un partit mai comença a l'hora que indica a les seves metadades, ja que a no ser que sigui el primer de la jornada, depèn del temps de joc que hagi tingut el partit anterior, així doncs, és molt probable que comenci abans d'hora.

Un cop el partit estigui en directe, començarem a mirar si el primer mapa està en directe. Quan detectem que el mapa està en directe, començarem a seguir les snapshots d'aquest a mida que vagi avançant, per així anar notificant a l'usuari dels esdeveniments més destacats.

Quan el mapa hagi acabat, actualitzarem l'estat del partit i mirarem si hi ha un guanyador. En el cas que hi hagi un, el seguiment del partit s'acabarà; però si no hi ha cap encara, es repetirà el cicle de seguiment de mapa que he descrit anteriorment amb el següent mapa. Alhora, tot això es repetirà fins que el partit tingui un guanyador.



5.2.1.1. GetScheduledMatches

Aquesta funció Lambda s'encarregarà de mirar quins partits hi ha programats a la base de dades de Riot Games i en programarà una funció Lambda per a cadascun per a seguir-los.

Primer de tot, farà una request a la direcció de Scheduled Events per a veure quins partits no s'han jugat encara. Aquests partits es filtraran segons la lliga (per a no sobrecarregar l'aplicació, he agafat les 4 lligues més importants a nivell mundial), segons l'estat en el que estiguin (unstarted) i el temps que quedi per a que comencin (que he establert en 2 setmanes com a màxim). Un cop filtrats, mirarà quins no estan a la base de dades i crearà una norma que invoqui a la funció Lambda MatchIsLiveScheduler (que s'encarregarà de començar el seguiment de cada partit). Aquesta norma estarà programada per invocar la funció una hora abans de començar, ja que l'hora d'inici real pot variar depenent del partit que s'hagi jugat anteriorment (si ha durat poc, el següent partit començarà abans de l'hora programada). Val la pena comentar que tant l'hora donada per l'API de Riot com el rellotge intern que fan servir les funcions Lambda estan en format UTC, així que no s'ha de tocar res de cap zona horària. Si la norma s'ha pogut crear correctament, s'afegirà el partit corresponent a la base de dades. A aquesta norma se li donarà com a input el id del partit (matchId o esportsMatchId).

Seguidament, mirarà si a la request feta al principi hi ha indicada una pàgina més nova (ja que recordem, Scheduled Events retorna la pàgina més recent completa de partits, això vol dir que pot haver una pàgina nova amb més partits però sense estar completada). En el cas que existeixi, tornarà a fer una request a Scheduled Events indicant la nova pàgina a consultar, i repetirà el mateix procés anterior, afegint els partits que volem a la base de dades i creant les normes corresponents.

Polítiques

Les polítiques que necessita aquesta funció per a poder dur a terme la seva feina són:

EventBridge	PutRule: Permet a la funció afegir normes a EventBridge. PutTargets: Permet a la funció afegir objectius a les normes d'EventBridge.
Lambda	AddPermissions: Permet a la funció donar permisos d'invocació des d'EventBridge a altres funcions Lambda.

5.2.1.2. MatchIsLiveScheduler

Aquesta funció s'encarrega de programar la invocació cada 1 minut de la funció MatchIsLive (comprova si un partit està en directe). La intenció principal era fer que GetScheduledMatches s'encarregués d'aquesta feina, però no es pot crear una norma que invoqui una funció a una hora i data concretes i que després la vagi repetint cada minut, així que he hagut de crear aquesta funció que permet programar des de GetScheduledMatches la invocació de MatchIsLive cada minut a partir d'una hora i data concretes.

Primer de tot eliminarà la norma creada per GetScheduledMatches, ja que estava programada per a una hora i data en concret, així que un cop passada aquesta data, ja no serveix de res. Juntament amb la norma, s'eliminaran tots els permisos associats a ella.

Després comprova si hi ha la norma que invoca MatchIsLive cada 1 minut creada, si no ho està, la crearà sense cap objectiu. Un cop està la norma creada, s'afegirà com a objectiu la funció MatchIsLive amb l'id del partit (matchId) com a input. Llavors es comprovarà si la norma està activada, i en el cas que no ho estigui s'activarà.

Polítiques

Les polítiques que necessita aquesta funció per a poder dur a terme la seva feina són:

EventBridge	Put/DeleteRule: Permet a la funció afegir i eliminar normes a EventBridge. Put/RemoveTargets: Permet a la funció afegir i eliminar objectius a les normes d'EventBridge. Enable/DisableRule: Permet a la funció activar i desactivar normes d'EventBridge. DescribeRule: Permet a la funció veure els detalls d'una norma. ListRules: Permet a la funció veure quines normes hi ha a EventBridge.
Lambda	Add/RemovePermissions: Permet a la funció donar i treure permisos d'invocació des d'EventBridge a altres funcions Lambda.

5.2.1.3. MatchIsLive

Aquesta funció s'encarrega de mirar si el partit que coincideix amb el matchId passat com a paràmetre ha començat o no. En el primer cas, invocarà la funció UpdateMatchState que ja s'encarregarà de fer el seguiment del partit.

Primer de tot, farà una request a LiveEvents per tal d'obtenir un JSON amb els partits que s'estan jugant en aquell moment. Mirarà partit per partit a veure si hi ha algun que coincideix amb el matchId rebut com a input i que el seu estat sigui "inProgress".

Un cop trobat, s'afegiran les metadades del partit a la base de dades, es crearà l'objecte Match corresponent a partir d'aquestes i es guardarà també a la base de dades. Seguidament, s'eliminarà l'objectiu de la norma que invocava aquesta funció en intervals d'1 minut amb el matchId corresponent com a input ja que s'ha trobat que el partit ja està en directe. Un cop eliminat aquest objectiu, si la norma MatchIsLive no en té cap, es desactivarà.

Per acabar, s'invocarà la funció UpdateMatchState passant l'id del partit (matchId) corresponent com a input. Aquesta s'encarregarà de començar el seguiment del partit.

Polítiques

Les polítiques que necessita aquesta funció per a poder dur a terme la seva feina són:

EventBridge	RemoveTargets: Permet a la funció eliminar objectius a les normes d'EventBridge. DisableRule: Permet a la funció desactivar normes d'EventBridge. DescribeRule: Permet a la funció veure els detalls d'una norma. ListTargetsByRule: Permet a la funció veure la llista d'objectius d'una norma en concret.
Lambda	InvokeFunction: Permet a la funció invocar una altra funció Lambda.

5.2.1.4. UpdateMatchState

Aquesta funció s'encarregarà d'actualitzar l'estat d'un partit en directe. Si no està començat el començarà i si està començat mirarà després d'acabar un mapa si algun equip ha guanyat. En el cas que hagi guanyat algun equip finalitzarà el

seguiment del partit en qüestió. Això vol dir que tindrà dos fluxos alternatius segons l'input que tingui.

Si l'input que rep és només l'id del partit, vol dir que la funció MatchIsLive ha detectat que està en directe i és torn de la funció actual començar el seguiment. El primer que farà serà obtenir de la base de dades l'objecte Match corresponent al id del partit rebut. Enviarà una notificació als usuaris que estan subscrits a aquest partit indicant que acaba de començar. Després, per seguretat, mirarà si les normes per a les funcions GameIsLive i UpdateGameState existeixen, i en cas que no sigui així, les crearà. Un cop assegurat que la norma per a invocar GameIsLive en períodes d'1 minut existeix, l'activarà en cas que estigui desactivada i afegirà com a objectiu la funció GameIsLive amb l'input que contingui l'id del partit (matchId) i l'id del primer mapa (gameId).

Si l'input que rep és l'id del partit (matchId) i un id d'un mapa (gameId), vol dir que la funció ha estat invocada des de la funció UpdateGameState, indicant que ha acabat un mapa. Llavors recuperarà l'objecte Match de la base de dades amb el matchId corresponent i afegirà el guanyador del mapa que just ha acabat. Llavors mirarà si el partit té ja un guanyador. Si el té mourà el partit en la base de dades de la col·leció de partits en directe a la col·lecció de partits jugats. En canvi, si no té guanyador encara, enviarà una notificació als usuaris subscrits al partit en qüestió indicant quina és la puntuació actual i actualitzarà l'objecte Match de la base de dades. Llavors, igual que abans, afegirà com a objectiu a la norma de GameIsLive la pròpia funció GameIsLive amb l'input que contingui l'id del partit (matchId) i l'id del següent mapa a jugar (gameId) i l'activarà en cas que estigui desactivada.

Polítiques

Les polítiques que necessita aquesta funció per a poder dur a terme la seva feina són:

EventBridge	DescribeRule: Permet a la funció veure els detalls d'una norma. Enable/DisableRule: Permet a la funció activar o desactivar normes. ListRules: Permet a la funció veure quines normes hi ha a EventBridge. PutRule: Permet a la funció afegir normes a EventBridge. PutTargets: Permet a la funció afegir objectius a les normes d'EventBridge.
Lambda	AddPermission: Permet a la funció donar permisos d'invocació des d'EventBridge a altres funcions Lambda.

	RemovePermissions: Permet a la funció treure els permisos d'invocació des d'EventBridge a altres funcions Lambda.

5.2.1.5. GameIsLive

Aquesta funció s'encarregarà de comprovar si un mapa en concret s'ha començat a jugar o no. En el cas que ja hagi començat, passarà a la següent funció per a començar el seguiment.

Com que aquesta funció s'invoca a través d'una norma, rebrà com a paràmetres l'id del partit (matchId) i l'id del mapa (gameId) que es vol mirar si ha començat o no. Per a comprovar-ho, farem servir una de les característiques de la request de l'API feed.lolesports.com, Livestats window. Com ja s'ha comentat abans, si a aquesta request no se li passa cap data i hora per paràmetre, retorna les primeres snapshot del mapa, així doncs, si fem la request i aquesta falla, voldrà dir que aquestes primeres snapshot encara no existeixen i, per tant, el mapa encara no ha començat. Si és així, no farà res més i la funció es tornarà a invocar en 1 minut, gràcies a la norma d'EventBridge, per tornar a intentar veure si el mapa està en directe.

Si la request no dóna error i conté un paràmetre anomenat "esportsGameId", voldrà dir que el mapa ja ha començat, ja que la request haurà retornat les primeres snapshot d'aquest. Llavors recuperarem l'objecte Match corresponent al matchId rebut i crearem un objecte Game corresponent al gameId rebut, que posteriorment guardarem a la base de dades.

Un cop actualitzades les dades dels objectes Match i Game, afegirem com a objectiu a la norma UpdateGameState la funció del mateix nom amb el següent input: matchId (el mateix rebut), gameId (el mateix rebut) i la data i hora de la primera snapshot obtinguda arrodonida sempre a la desena superior (si els segons són 43, al arrodonir es convertiran en 50), ús que explicaré en el funcionament de la funció UpdateGameState.

Per acabar, eliminarem l'objectiu de la norma GameIsLive que invocava aquesta funció amb l'input corresponent, així es pararà d'executar cada 1 minut. Comprovarà si hi ha algun objectiu més a la norma, ja que si no hi ha cap no té sentit que estigui activa, així que la desctivarà si és el cas.

Polítiques

Les polítiques que necessita aquesta funció per a poder dur a terme la seva feina són:

EventBridge	PutRule: Permet a la funció afegir normes a EventBridge.

	Remove/PutTargets: Permet a la funció afegir i eliminar objectius de les normes d'EventBridge. DescribeRule: Permet a la funció veure els detalls d'una norma. Disable/Enable rule: Permet a la funció activar o desactivar normes. ListTargetsByRule: Permet a la funció veure la llista d'objectius d'una norma en concret.
Lambda	AddPermission: Permet a la funció donar permisos d'invocació des d'EventBridge a altres funcions Lambda.

5.2.1.6. UpdateGameState

Aquesta funció s'encarregarà de processar les snapshots del mapa que s'estigui jugant en directe per a poder detectar els esdeveniments més importants. Les snapshots es reben en intervals de 10 segons, però les invocacions a funcions Lambda es poden programar com a mínim en intervals d'un minut, així que processarà totes les snapshot generades en 1 minut, que seran un total de 6 requests. Per a que sigui el més fidel possible a un seguiment en viu, hauria d'haver una espera de 10 segons entre request i request, però això faria que augmentés en gran mesura el temps d'execució, i per tant, seria molt més costós a nivell econòmic.

Com que aquesta funció s'invoca a través d'una norma, rebrà com a paràmetres el id del partit (matchId), el id del mapa (gameId) i una data i hora (date). Recuperarà l'objecte Game corresponent al gameId rebut de la base de dades per a poder veure quin és l'últim estat del mapa.

El nucli d'aquesta funció consistirà en un bucle que es recorrerà un total de 6 vegades, una per cada interval de 10 segons. En la primera iteració d'aquest bucle s'agafarà el paràmetre date rebut a través de l'input i es farà una request a Livestats window, que retornarà totes les snapshot fetes en els 10 segons anteriors. Llavors s'anirà iterant snapshot per snapshot actualitzant l'estat del mapa fent servir l'objecte Game recuperat de la base de dades anteriorment. Al fer l'actualització, el propi objecte Game compararà el nou estat (snapshot) amb el que tenia abans per a veure si ha passat alguna cosa. La comparació es farà restant les dades de l'estat antic a les dades de l'estat nou, així si si ha alguna variable diferent a 0 es podrà veure què ha canviat. Aquestes variables seran:

- Dels equips:
    - Torres. Es podrà detectar la primera torre destruïda del mapa.
    - Dracs. Es podrà detectar quan un equip aconsegueix l'ànima d'un drac (que s'aconsegueix al matar a un total de 4 dracs).
    - Inhibidors
    - Baron Nashor
- Dels jugadors:

- Assassinats. Amb una mica de lògica interna, es podrà veure també els assassinats múltiples (matar més d'un jugador enemic dins d'un període específic de temps) i les matances (la quantitat d'assassinats fets per un jugador sense que el matin). També es podrà detectar el primer assassinat del mapa (primera sang).

Aquest procés es pararà si es detecta que el mapa ja té un guanyador. Si no és el cas, al paràmetre date se li sumaran 10 segons i es repetirà el mateix procés descrit anteriorment per al següent lot de snapshots. Al final de tot aquest procés s'actualitzarà l'objecte Game guardat a la base de dades amb les noves dades processades.

Un cop acabat el bucle que processa totes les snapshot de l'últim minut, es mirarà si el mapa té un guanyador, és a dir, s'ha acabat. Si no és així, s'actualitzarà l'objectiu de la norma que invoca aquesta funció amb un nou input que contindrà el mateix matchId i gameId però una data i hora (date) per a que la següent invocació pugui processar les snapshot del següent interval d'1 minut.

Si el mapa ha acabat, s'eliminarà l'objectiu de la norma que invoca aquesta funció per a parar el seguiment del mapa. Es mirarà si la mateixa norma té algun objectiu, i si no en té cap, es desactivarà, ja que si no té cap objectiu no té sentit tenir la norma activada (és una despesa innecessària). Per acabar, s'invocarà la funció UpdateMatchState passant com a paràmetres el matchId i el gameId, per a que actualitzi l'estat del partit.

Polítiques

Les polítiques que necessita aquesta funció per a poder dur a terme la seva feina són:

EventBridge	DisableRule: Permet a la funció desactivar normes. ListTargetsByRule: Permet a la funció veure la llista d'objectius d'una norma en concret. Remove/PutTargets: Permet a la funció afegir i eliminar objectius de les normes d'EventBridge.
Lambda	InvokeFunction: Permet a la funció invocar una altra funció Lambda.

5.2.2. Layers

Per a poder executar el codi de totes aquestes funcions, són necessaris els següents mòduls de Python:

- requests: Per a poder executar les consultes a les APIs de Riot Games.
- firebase-admin: Per a poder escriure i llegir dades de la base de dades de Firebase.

He hagut de crear un contenidor en Docker que tingui instal·lat Python 3.8, ja que és la versió de Python sobre la que està escrit el codi de les funcions Lambda, i els mòduls request i firebase-admin per a poder fer servir el seu codi. Llavors a partir d'aquest contenidor he pogut extreure la carpeta on estan instal·lats tant Python 3.8 com els mòduls que necessito i així crear la layer per a que les funcions Lambda puguin funcionar correctament.

5.3. Firebase

5.3.1. Organització

Com que és una base de dades NoSQL, treballarem amb col·leccions. Tindrem 5 tipus d'elements principals:

- **Partits (Matches)**: Tenen com a id identificatiu en la base de dades el seu corresponent esportsMatchId, que és l'id donat per Riot Games. De totes les dades que es poden rebre en la request de Scheduled events, es guarden a Firebase el esportsMatchId, la quantitat de mapes a jugar, l'hora prevista d'inici i les dades dels dos equips que s'enfronten (que obtindrem de la col·lecció equips). Les dades de cada mapa que s'hagi de jugar en cada partit es guarda en una subcol·lecció. Per qüestions de comoditat amb l'aplicació Android, estan repartits en 3 col·leccions diferents, cadascuna amb les seves peculiaritats:
    - **ScheduledMatches**: Partits programats que encara no s'han jugat. Tindran les dades tal qual les obtenim de la base de dades de Riot Games.
    - **LiveMatches**: Partits que s'estan jugant en directe. Com que hi ha una funció Lambda d'AWS que s'encarrega de moure les dades de la col·lecció d'ScheduledMatches a LiveMatches, tindran les mateixes dades que tenien a la primera. També tindran un JSON anomenat matchObjectJson que contindrà les dades de l'objecte Match que es fa

servir a les funcions Lambda per a processar el partit en directe. Aquest JSON s'anirà actualitzant a mida que avança el partit.
    - **CompletedMatches**: Igual que amb LiveMatches, hi ha una funció Lambda d'AWS que s'encarrega de moure les dades de LiveMatches a aquesta, així doncs, els partits tindran les mateixes dades però el matchObjectJson no es modificarà en cap moment.

- **Mapes (Games)**: Tenen com a id identificatiu en la base de dades el seu esportsGameId, que és l'id donat per la base de dades de Riot Games. Estan organitzats en subcol·leccions dels partits, on cada partit té una subcol·lecció dels mapes corresponents. Depenent de la col·lecció en la que estigui el partit tindran unes dades o altres:
    - **ScheduledMatches**: Es guarden només l'esportsGameId, el número de mapa que correspon dins l'ordre del partit, l'estat (que serà sempre "unstarted") i les dades dels equips en el context del mapa, id de cadascun, costat en el que jugaran i resultat.
    - **LiveMatches**: Les mateixes dades que a ScheduledMatches però amb l'afegit del JSON que defineix l'objecte Game que fan servir les funcions Lambda per a seguir el mapa (gameObjectJson). Les notificacions generades per les mateixes funcions es guarden en una subcol·lecció anomenada Notifications. Tant el gameObjectJson com la subcol·lecció de Notificacions es van modificant a mida que es va jugant el mapa.
    - **CompletedMatches**: La mateixa estructura que a LiveMatches però en aquest cas no es modificarà en cap moment el gameObjectJson ni la subcol·lecció de les notificacions.

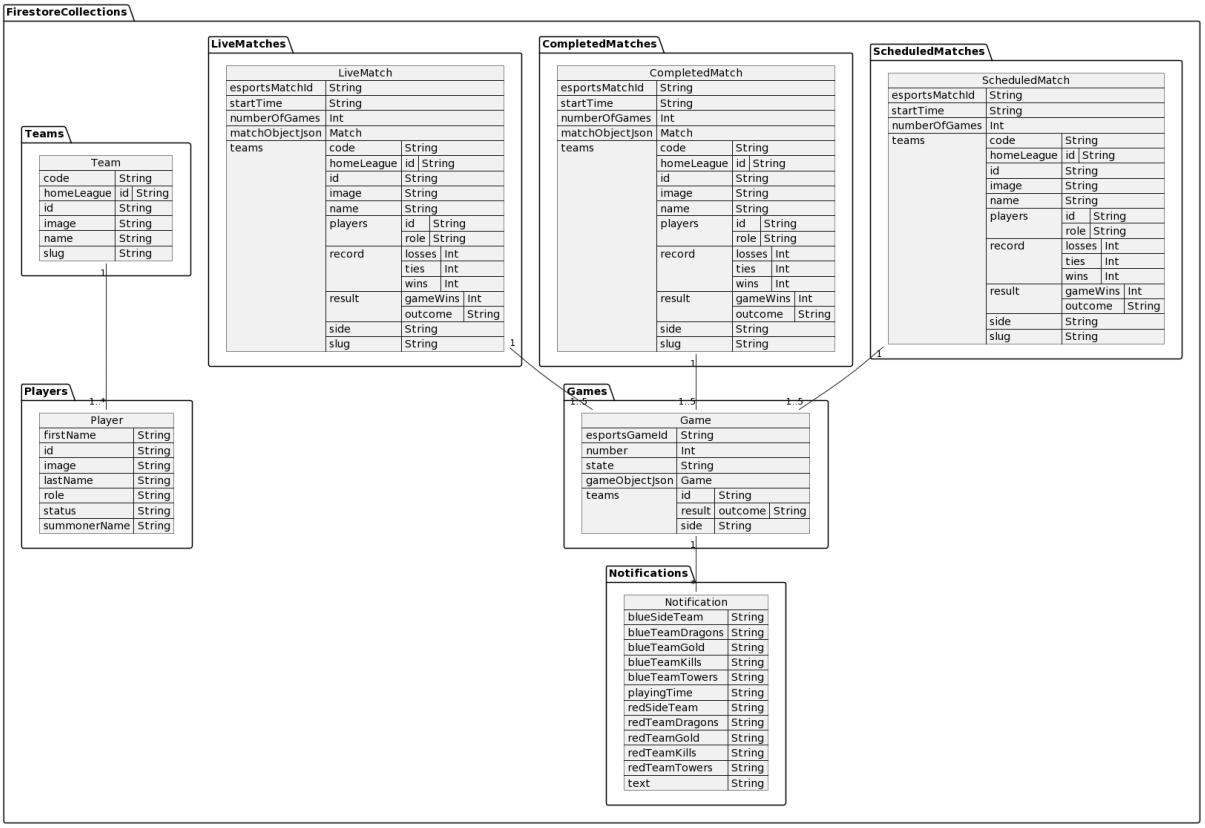
- **Notificacions (Notifications)**: Tenen com a id l'instant de temps de joc en el que s'han creat. Es guarden les estadístiques de cada equip segons el costat en el que estiguin jugant (costat blau o vermell), el text descriptiu de l'esdeveniment i el mateix instant de temps en el que ha passat.

- **Usuaris (Users)**: Tenen com a id un codi donat al fer el login amb el seu compte de Google. Només es guarda aquest codi i una llista amb els partits als quals estan subscrits.

- **Equips (Teams)**: Tenen com a id el mateix id donat per Riot Games en la seva base de dades. Es guarda el codi (una abreviació de 3 lletres en majúscula), l'id de la lliga a la que pertanyen, el nom, el mateix id i l'enllaç de la imatge del seu logo. Les dades dels seus jugadors estaran en una subcol·lecció anomenada Players.

- **Jugadors (Players)**: Tenen com a id el mateix id donat per Riot Games en la seva base de dades. Es guarda el seu nom complet, el rol que juguen, l'estat (si està actiu o no, etc), el nom d'invocador (que és l'àlies amb el que juguen els partits) i l'enllaç de la seva imatge de perfil. Cada jugador està en la subcol·lecció corresponent al seu equip.

Així doncs, l'estructura de la base de dades és la següent:



5.3.2. Servei de missatgeria

Firebase ofereix un sistema de missatgeria amb múltiples característiques, però hi ha una en específic que s'adapta a la perfecció a les necessitats de l'aplicació. Aquesta és la missatgeria per tòpics.

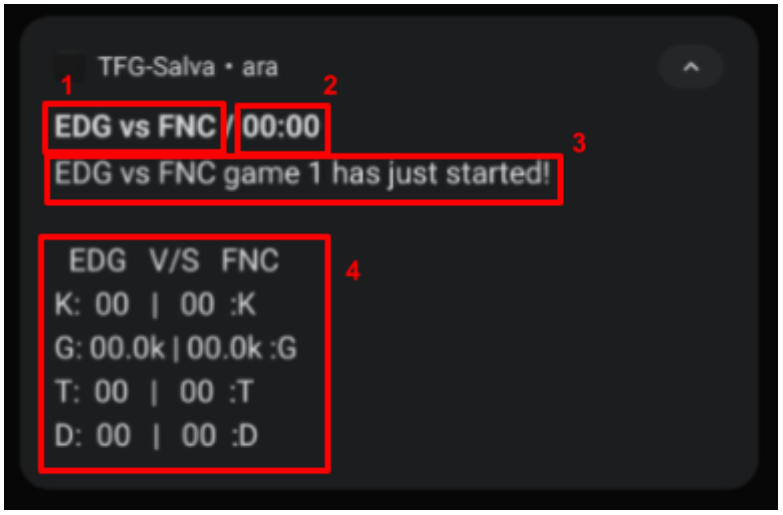
Basat en un sistema de publicació/subscripció, Firebase Cloud Messaging (FCM) permet enviar missatges a múltiples dispositius que hagin escollit estar subscrits a un tòpic o tema particular. Llavors des de l'aplicació només s'hauran de crear els missatges amb el tòpic corresponent segons sigui necessari, i FCM s'encarregarà del routing i de fer arribar els missatges als dispositius que hi estiguin subscrits.

És molt senzill de fer servir i en aquest cas l'hem utilitzat per a enviar les notificacions amb els esdeveniments que van passant als partits que s'estan jugant en directe. Llavors com a tòpic farem servir l'id del partit, així doncs, des de l'aplicació, quan l'usuari es vulgui subscriure a un partit s'estarà subscrivint al tòpic corresponent a l'id d'aquest.

5.4. Aplicació Android

5.4.1. Notificacions

La primera dels elements principals són les notificacions. Permeten a l'usuari saber com va un mapa d'un partit que està seguint sense haver d'obrir ni el telèfon ni l'aplicació. Aquesta notificació proporcionarà informació clau de l'esdeveniment que ha passat i de l'estat dels dos equips en aquell mateix moment. Només es rebran les notificacions dels mapes dels partits als quals l'usuari està subscrit. Cadascuna tindrà la següent estructura i elements:



1. Indica quins són els dos equips que s'estan enfrontant en el mapa que s'està jugant. L'ordre a més a més indica a quin costat del mapa juga cadascun, sent el de l'esquerra el que juga al costat blau i el de la dreta el que juga al costat vermell.

2. Indica el temps de joc en el moment que ha passat l'esdeveniment. Està en minuts i segons. En el cas que un mapa superés l'hora de duració, llavors s'afegirien les hores també, però com que és molt estrany que passi, he deixat per defecte que no es mostrin.

3. El text amb el missatge de l'esdveniment que ha succeït. Els esdeveniments que poder ser notificats són:
    a. Assassinats. Indica també si ha estat un assassinat múltiple i de quin calibre i la ratxa d'assassinats que porta el jugador que l'ha dut a terme.
    b. Estructures destruïdes. Inclou torres i inhibidors.
    c. Objectius neutrals aconseguits. Inclou dracs elementals, dracs ancians i Baron Nashor. La informació donada per l'API de feed.lolesports.com noté en compte els heralds, així que no sortiran a les notificacions.
    d. Estats generals del mapa i el partit:
        i. Inici del partit i de cada mapa.
        ii. Final del partit i de cada mapa.
        iii. Pauses que poden tenir els mapes. S'avisa quan es pausa i quan es reprèn. També s'indica la quantitat de temps que ha estat el joc pausat.
        iv. Puntuació del partit quan s'acaba un mapa.

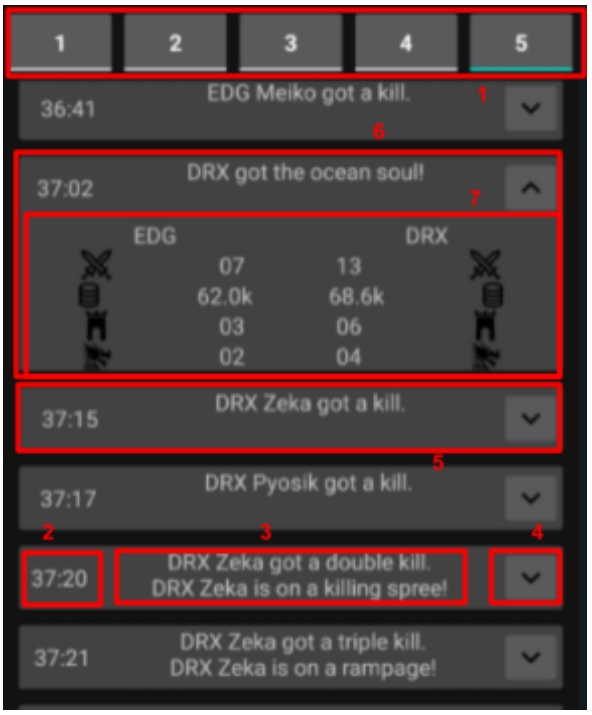
4. Mostra informació extra amb les estadístiques de cada equip en el moment en el que ha succeït l'esdeveniment. Igual que en el punt 1, l'ordre en el que es mostren les dades fa referència al costat del mapa en el que està jugant cada equip: a l'esquerra el costat blau i a la dreta el costat vermell. Quan un mapa s'està jugant, és complicat saber amb exactitud quin equip té un avantatge sobre un altre, sobretot si és un partit lleugerament igualat, així que he considerat adient mostrar les següents estadístiques de cada equip:
    a. Assassinats (Kills). Indica la quantitat total d'assassinats que han aconseguit els jugadors de l'equip. És el primer element en el que la gent es fixa (i es mostra en els streamings) ja que és l'esdeveniment més celebrat pel públic.
    b. Or (Gold). És la moneda interna del joc que s'aconsegueix ja sigui fent assassinats, aconseguint objectius, etc. És una estadística més important que els assassinats ja que indica quin equip té un avantatge directe sobre l'altre, perquè permet als jugadors poder comprar recursos per a millorar el seu personatge.
    c. Torres. Són les línies de defensa de cada base de cada equip. Indica quantes torres ha destruït cada equip al seu adversari. Com més alta sigui la puntuació més control tindrà un equip sobre el territori de l'altre.
    d. Dracs. Són objectius neutrals del mapa, és a dir, els poden caçar tant un equip com l'altre. Cada drac aporta estadístiques de combat a cada jugador de l'equip que l'ha caçat.

    El Baron Nashor és també un objectiu neutral molt important dins del desenvolupament del mapa, sobretot cap a les darreres etapes d'aquest.

Però l'avantatge que aporta a l'equip que l'ha aconseguit és temporal, 180 segons, així doncs no he cregut que sigui necessari mostrar-lo per aquí.

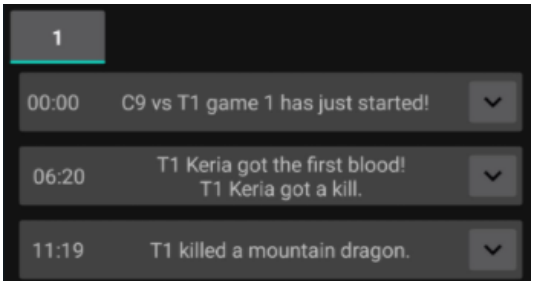
5.4.2. Timeline dels partits

És important saber també què ha passat en un partit, ja sigui quan s'estigui jugant com quan ja s'ha jugat. Així doncs, he fet que l'aplicació vagi recopilant les dades de les notificacions dels mapes i mostrar-les en una llista. Per als mapes que s'estiguin jugant en directe, s'anirà actualitzant la llista amb els nous esdeveniments que s'hagin notificat. L'actualització de la llista es fa manualment lliscant cap a baix, que és la manera més intuïtiva ja que la fan servir una gran quantitat d'aplicacions. L'estructura de la llista i cada element és la següent:



1. Són els botons que deixen escollir de quin mapa es volen veure els esdeveniments. Això permet tenir en una mateixa pantalla tots els mapes d'un sol partit, així que només s'ha de seleccionar el botó amb el número corresponent al mapa que es vol consultar per a veure els esdeveniments. Si es selecciona un mapa que encara no s'ha jugat, la llista estarà buida ja que encara no han passat esdeveniments. Exemple:

Hi haurà tants botons com el número màxim de mapes que pugui tenir el partit, és a dir, si és al millor de 3 hi haurà 5, si és al millor de 2 hi haurà 3 i si és al millor d'1 hi haurà 1. Exemple:



2. És el temps de joc en el que ha passat l'esdeveniment. Igual que amb les notificacions, està en format minuts i segons. Si es donés el cas que el mapa superés l'hora de durada, llavors s'afegirien les hores, però com que és molt estrany que passi, per defecte no es mostren.

3. És el text descriptiu de l'esdeveniment. És el mateix text que es mostra per les notificacions, així que informa dels assassinats, les torres destruïdes, els objectius neutrals aconseguits i de l'estat general del mapa (quan comença, quan acaba i qui és el guanyador, quan s'ha pausat el mapa i quan s'ha begut amb la quantitat de temps que ha estat el mapa pausat). No es mostra l'estat general del partit, ja que la llista conté esdeveniments específics de cada mapa.

4. És un botó que permet ampliar l'element de la llista en qüestió per a poder veure les estadístiques generals de cada equip en l'instant en el que ha passat l'esdeveniment. Si l'element no està ampliat el botó tindrà una fletxa cap avall, ja que "s'allarga" per sota. Si l'element està ampliat tindrà una fletxa cap amunt, ja que "pujarà" o s'amagarà la part extra que conté les estadístiques generals dels equips.

5. És un element de la llista sense ampliar els detalls. Conté només el temps de joc, el text descriptiu de l'esdeveniment i el botó que permet ampliar-lo per a veure les estadístiques generals dels equips.

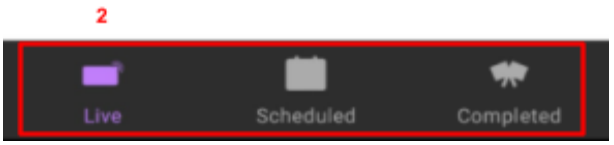
6. És un element de la llista amb els detalls ampliats. Conté el temps de joc, el text descriptiu de l'esdeveniment, les estadístiques generals dels equips i el botó que permet amagar-les.

7. Són les estadístiques generals dels equips en l'instant en el que ha passat l'esdeveniment. Igual que en les notificacions, es mostren de dalt a baix:
    a. Els assassinats totals de cada equip (la suma dels assassinats de cada membre).
    b. L'or total aconseguit de cada equip (la suma de l'or aconseguit per cada membre).
    c. Les torres destruïdes per cada equip.
    d. La quantitat de dracs aconseguits per cada equip.

    Estan posats en aquest ordre per exactament la mateixa raó en la que ho estan a les notificacions.

5.4.3. Llista de partits

És l'activitat principal de l'aplicació. Té 3 vistes diferents, una per cada estat en el que pot estar un partit: programat (scheduled), són els partits pendents per jugar dels que ja se sap la data i hora; en directe (live), són els partits que s'estan jugant actualment; i completats (completed), són els partits que ja s'han jugat i per tant es pot veure què ha passat en cadascun d'ells. L'estructura de la llista i de cada element és la següent:



1. És l'element representatiu d'un partit. Múltiples d'aquests formen les llistes de partits en directe, programats i completats. Fent click a qualsevol d'aquests obrirà la timeline dels partit corresponent amb tots els esdeveniments que han passat al llarg dels mapes que s'han jugat. Els partits que estan programats no seran clickables ja que no s'han jugat encara i per tant no haurà passat cap esdeveniment.

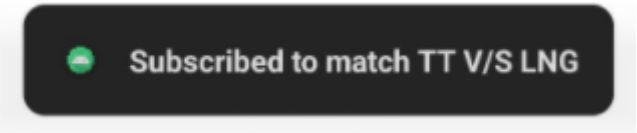
2. És el menú principal que permet navegar per les tres llistes diferents. Al obrir l'aplicació, per defecte es mostrarà la llista de partits en directe.

3. És el logo i el nom complet del primer equip que formarà part del partit. Al estar a l'esquerra, indica que el primer mapa d'aquell partit el començarà jugant al costat blau.

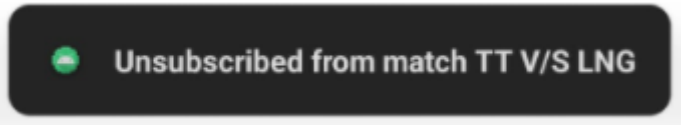
4. És el logo i el nom complet del segon equip que formarà part del partit. Al estar a la dreta, indica que el primer mapa d'aquell partit el començarà jugant al costat vermell.

5. És un botó en forma de cor que serveix per a subscriure's a un partit. Això farà que l'usuari rebi les notificacions d'aquest partit quan s'estigui jugant en directe. En l'exemple es mostra com és quan l'usuari no està subscrit. Si l'usuari es subscriu, serà de la següent manera:

Al subscriure's l'usuari, es mostrarà el següent missatge breument per pantalla:



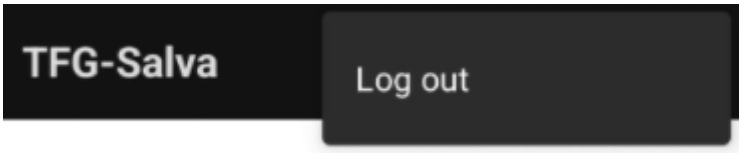
En canvi, al deixar d'estar subscrit, es mostrarà el següent:



6. Indica l'hora i dia d'inici programats per al començament del partit.

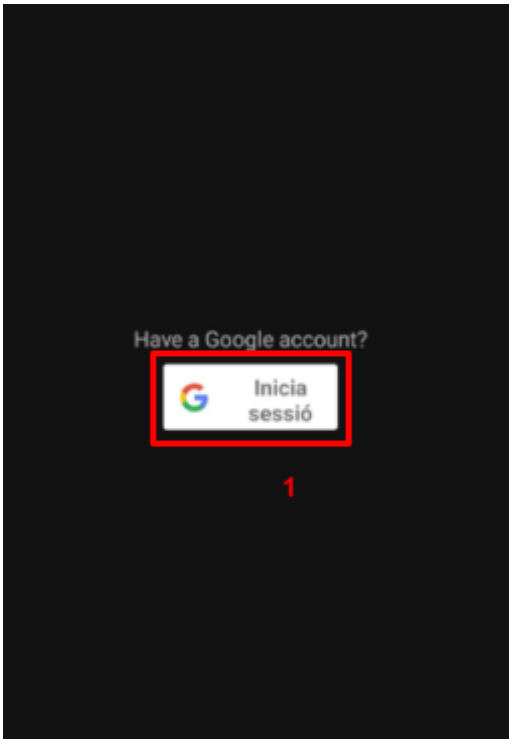
7. Indica quin tipus de partit serà, és a dir, el número màxim de mapes que es jugaran, i per tant, la quantitat de mapes que necessitarà guanyar un equip per a guanyar el partit. Normalment seran de 3 tipus diferents:
    a. Best of 1: Es jugarà només 1 mapa, i per tant qui el guanyi serà el guanyador del partit.
    b. Best of 3: Es jugaran com a molt 3 mapes, i per tant, el primer en guanyar 2 guanyarà el partit.
    c. Best of 5: Es jugaran com a molt 5 mapes, i per tant, el primer en guanyar 3 guanyarà el partit.

8. És un botó que si es fa click desplegarà un menú per a poder fer log out de l'aplicació:



5.4.4. Pantalla Login Google

És el primer que ens surt només obrir l'aplicació. Si ja hi ha un usuari loguejat es mostrarà directament la llista de partits sense haver de passar per aquesta pantalla. En cas contrari es mostrarà el botó de login de Google per a que l'usuari pugui escollir amb quin correu es vol loguejar. És una pantalla molt simple, l'estructura és la següent:



1. És el botó d'inici de sessió que ofereix Google per defecte. Permet triar entre els correus electrònics de Gmail que hi ha guardats al dispositiu per a poder fer el login.

6. Metodologia

Per a poder dur a terme aquest projecte, he escollit seguir la metodologia Agile, així doncs tant els meus tutors com jo hem format un equip cadascun amb uns rols marcats. Al ser un projecte curt, vam establir en fer sprints d'una setmana per a poder fer un seguiment bastant detallat del procés, ja que no podíem fer les reunions diàries que proposa aquesta metodologia. El fet de fer sprints tant curts, també ens ha anat molt bé per a poder reaccionar a temps a les dificultats que han anat sortint al llarg del desenvolupament, així que gràcies a això, s'han pogut enllestir diverses tasques en un temps menor.

Els membres de l'equip de treball amb els seus rols corresponents han estat els següents:

Salvador Mira:
- Product Owner
- Tech Lead
- Developer

Alberto Guerra:
- SME & Agile coach
- Punt de contacte amb Riot Games
- Technical advisor

Eloi Puertas:
- Technical Advisor
- Academic Advisor

Els sprints que s'han fet al llarg del projecte i els objectius de cadascun han estat els següents:

Sprint 0: Treball previ a l'inici oficial del projecte.
- Investigació tècnica de les APIs de Riot Games.
- Fer una demo per comprovar que realment era possible detectar els esdeveniments més importants d'un mapa a partir de les APIs de Riot Games.
- Demo de notificacions amb els esdeveniments detectats.

Sprint 1
- Mockups aplicació
- Mirar funcionament Render.com

Sprint 2

- Backend: Detectar partits en directe i que es van a jugar i guardar-los a la base de dades. Eliminar-los quan ja no ho estiguin.
- Firebase: Crear base de dades per a guardar els partits.
- Aplicació: Fer la part gràfica per a poder veure els partits que hi ha en directe.

Sprint 3

- Fer un diagrama de classes simulant un JSON de la base de dades.
- Poder veure els esdeveniments de cada mapa jugat o que s'està jugant.

Sprint 4

- Investigar com funciona Localstack i les funcions Lambda d'AWS. Preparació pel següent sprint.

Sprint 5: Aquest va ser un sprint de dues setmanes degut a que no vam poder coincidir el dia de la reunió. Així que es va tenir en compte i es va programar més feina que a la resta d'sprints.

- Diagrama de flux de les funcions Lambda d'AWS.
- Crear mètodes per a poder programar normes d'EventBridge dinàmicament.
- Passar el backend de l'aplicació de local a les funcions Lambda.

Sprint 6

- Provar d'optimitzar el màxim possible la quantitat de funcions Lambda i les vegades que s'invoquen.
- Mirar diferències entre StepFunctions Express i Standard.
- Buscar i provar edgecases de partides.
- Generar backup a la base de dades de partides de testing per si l'API de Riot Games falla.

Sprint 7

- Investigar i provar Azure Logic Apps.
- Provar i solucionar l'edgecase de partida "pausa".
- Testejar i polir programació normes d'EventBridge dinàmicament.
- Modificar diagrama de flux de les funcions Lambda.
- Arreglar l'actualització de la llista d'esdveniments de cada mapa de l'aplicació.

Sprint 8

- Arreglar els problemes amb els permisos de les normes d'EventBridge.
- Començar la memòria del TFG.
- Fer la llista de coses que falten per fer a l'aplicació.

Sprint 9: Últim sprint en el que s'ha anat acabant tant la memòria com l'aplicació.

- Acabar memòria.
- Incloure els detalls que falten a l'aplicació de la llista feta a l'anterior sprint i anar polint els errors que puguin anar sortint.
- Acabar i polir el funcionament de les funcions Lambda.
- Testejar el conjunt del projecte quan comencin les competicions oficials.

7. Planificació i costos

7.1. Escalabilitat

7.1.1 Problemes

Per a aquest projecte he decidit fer el seguiment dels partits de les quatre lligues principals: la xinesa, la coreana, l'europea i la nord-americana. A més, a l'aplicació només es mostren els partits que hi ha programats dins de dues setmanes. Tot això té raó de ser.

La primera és que a l'aplicació no es poden filtrar per lligues els partits programats, així que si es posessin tots els que es poden obtenir de la base de dades (que a data 17 de Gener del 2023 són uns 600) es sobrecarregaria la llista de partits programats de l'aplicació (tant en el sentit pràctic com en el sentit tècnic). Així que d'alguna manera s'havien de reduir la quantitat de partits que es mostren.

La segona és que al detectar un partit programat que no està a la base de dades de l'aplicació, la funció GetScheduledMatches crea una norma que invocarà la funció MatchIsLiveScheduler. Això provoca que la norma creada s'afegeixi a la llista de Triggers de la segona funció, i com ja s'ha comentat a l'apartat de configuració de les funcions Lambda, aquesta llista està guardada en un document que té una mida d'uns 20 MB, que vénen a ser uns 55 triggers (o partits). Un cop s'arriba al límit ja no es poden afegir més triggers, així que tampoc es poden afegir els partits a la llista de partits programats (no tindria sentit tenir-los i que les funcions Lambda corresponents no es poguessin invocar).

7.1.2 Solucions

La part de l'aplicació és molt fàcil de solucionar, ja que és simplement implementar un filtre de lligues per a que l'usuari pugui veure els partits només de les lligues que ell vulgui. En el cas que vulgui veure moltes lligues i per tant hi hagués risc de sobrecarregar l'aplicació, es podrien fer pàgines per tal de limitar la quantitat d'elements dins la llista que s'està mostrant per pantalla.

La part de la limitació de Triggers de les funcions Lambda és una mica més complicada de gestionar que l'anterior, ja que cada norma per a poder invocar una funció ha d'estar com a Trigger de la segona. Així doncs no hi ha una manera directa d'optimitzar-ho. Una solució seria fer un grup de funcions Lambda per a cada lliga, és a dir, cadascuna tindria un conjunt de funcions Lambda dedicades a

processar els partits d'aquesta. Tot i així s'hauria d'anar amb compte, ja que per exemple, la fase regular de la lliga europea normalment són uns 90 partits (10 equips anada i tornada), que ja superaria el límit de 55 Triggers de les funcions Lambda. Llavors ja no seria només fer un conjunt de funcions Lambda per lliga, sinó per setmanes, fases i/o jornades.

7.2. Manteniment de l'aplicació

Per la part d'AWS només s'ha creat manualment el codi de cada funció Lambda (i tot el que comporta la funció com a tal) i la norma que executa la funció GetScheduledMatches setmanalment. Tota la resta de normes i recursos necessaris per a l'execució de l'algoritme es va creant dinàmicament per a cada partit. En el cas d'aquest projecte, el codi de cada funció s'ha anat modificant en el mateix editor que ofereix la web d'AWS, per comoditat al anar investigant i desenvolupant les funcions. És molt millor, i és possible de fer, un procés per a crear una release i així fer el deploy a totes les funcions a la vegada.

La part de la base de dades de Firebase i de l'aplicació d'Android depèn en la seva totalitat de l'ús que en vulgui fer jo, així doncs només s'hauran de modificar en cas que vulgui implementar alguna funcionalitat nova.

Llavors l'única part que sí hauria de tenir un manteniment real seria la gestió de les APIs de Riot Games, ja que no depèn de mi i s'hauria d'anar vigilant per si en algun moment es modifiquen i el codi de les funcions Lambda deixa de funcionar. Al no dependre de mi, si pel motiu que sigui les APIs deixen de funcionar, l'aplicació també ho farà, ja que totes les dades necessàries provenen d'aquí (que no hauria de passar almenys en un futur proper).

7.3. Cost econòmic

Anem a analitzar el cost que tindria mantenir tot aquest servei. Calcularem el cost mensualment (4 setmanes i mitja) de seguir les 4 lligues principals: la coreana (LCK), la xinesa (LPL), l'europea (LEC) i la nord-americana (LCS). Per a simplificar-ho, anem a mirar quants mapes per setmana es juguen en cadascuna de les lligues:

- LCK: Es juguen 10 partits al millor de 3 mapes, llavors posem que de mitjana es juguen 5 partits de 2 mapes (un equip guanya 2-0) i 5 partits de 3 mapes (un equip guanya 2-1). Sortirien un total de 5*2 + 5*3 = 25 mapes per setmana repartits en 10 partits.
- LPL: Es juguen 6 partits al millor de 3 mapes, llavors posem que de mitjana es juguen 3 partits de 2 mapes (un equip guanya 2-0) i 3 partits de 3 mapes (un equip guanya 2-1). Sortirien un total de 3*2 + 3*3 = 15 mapes per setmana repartits en 6 partits.

- LCS: Es juguen 10 partits al millor d'1 mapa. Sortirien un total de 10 partits i 10 mapes per setmana.
- LEC: Es juguen 15 partits al millor d'1 mapa. Sortirien un total de 15 partits i 15 mapes per setmana.

Surten un total de 25+15+10+15=65 mapes setmanals repartits entre 10+6+10+15=41 partits. Calcularem el cost per a una setmana i després l'extrapolarem a un mes, ja que Amazon calcula els costos mensualment. Cal tenir en compte que Amazon dóna gratuïtament una quantitat fixa de recursos.

Hi ha moltes variables a tenir en compte, però és un projecte tan "senzill" que no tindrem cap tipus de problema amb l'espai d'emmagatzematge. Així doncs, ens podrem centrar amb el cost que generen les invocacions de les funcions Lambda, així com el seu temps d'execució, i amb el cost que genera crear, modificar, obtenir i/o llistar elements d'AWS mitjançant la seva SDK (permisos de funcions Lambda, normes, etc). Suposarem un escenari el més genèric possible:

Els partits començaran a la seva hora prevista, així doncs, la funció MatchIsLive es cridarà un total de 60 cops. Cada execució tindrà un temps aproximat de 530ms i farà servir 128 MB de memòria. A més a més, es necessita invocar un cop la funció MatchIsLiveScheduler, que té un temps d'execució aproximat de 1,75 segons. Així doncs el cost per una setmana serà:

- 60 invocacions * 0,530 segons * 0,125GB * 41 partits = 162,975 GB-Seconds
- 1 invocació * 1,75 segons * 0,125GB * 41 partits = 8,969 GB-Seconds
- 60 invocacions * 41 partits = 2460 request
- 1 invocació * 41 partits = 41 request

Es començarà a mirar si un mapa s'està jugant aproximadament 15 minuts abans de que comenci (que és el temps que hi ha entre mapa i mapa d'un mateix partit). Llavors la funció GameIsLive s'invocarà un total de 15 cops. De tots aquests cops, el que detecta que el mapa ja ha començat, tindrà un temps d'execució aproximat de 5 segons, mentres que els altres 14 seran d'uns 250ms aproximadament. La funció GameIsLive té assignada 128MB de memòria, així doncs el cost per setmana serà:

- 14 invocacions * 0,25 segons * 0,125 GB * 65 mapes = 28,438 GB-Seconds
- 1 invocació * 5 segons * 0,125 GB * 65 mapes = 40,625 GB-Seconds
- 15 invocacions * 65 mapes = 975 request

Cada mapa durarà aproximadament 35 minuts, així doncs, la funció UpdateGameState s'invocarà un total de 35 cops. Cada execució tindrà un temps aproximat de 9 segons i la funció tindrà assignada 128 MB de memòria. Així doncs, el seu cost serà:

- 35 invocacions * 9 segons * 0,125 GB * 65 mapes = 2559,375 GB-Seconds
- 35 invocacions * 65 mapes = 2275 request

La funció UpdateMatchState també s'invocarà múltiples cops, que seran un cop abans de cada mapa i al final de cada partit per tancar-lo. El primer cas tindrà una duració aproximada de 4,2 segons i la del segon serà de 10 segons. La funció té assignada 128 MB de memòria, així doncs, el seu cost serà:

- 1 invocació * 4,2 segons * 0,125 GB * 65 mapes = 34,125 GB-Seconds
- 1 invocació * 10 segons * 0,125 GB * 41 partits = 51,125 GB-Seconds
- 1 invocació * 65 mapes = 65 request
- 1 invocació * 41 partits = 41 request

La funció GetScheduledMatches s'invoca un cop per setmana només, així que el seu cost és negligible.

És molt més complicat saber el cost que genera crear, modificar, obtenir i/o llistar elements d'AWS mitjançant la seva SDK, així doncs després de fer una petita simulació he aproximat els següents costos per mapa:

- 700 request PUT, COPY, POST, LIST * 65 maps = 45500 request
- 650 request GET * 65 = 42250 request

Els costos finals que surten mensualment (posant que un mes té 4,5 setmanes) seran els següents:

- 2885,332 GB-Seconds * 4,5 setmanes = 12983,994 GB-Seconds
- 5857 request Lambda * 4,5 setmanes = 26357 request Lambda
- 45500 request PUT, COPY, POST, LIST * 4,5 setmanes = 204750 request
- 42250 request GET * 4,5 setmanes = 190125 request

AWS ofereix una quantitat de recursos gratuïts mensualment, així que els haurem de tenir en compte per al càlcul final del cost mensual:

	Cost total	**Cost gratuït**	**Cost final**	**Cost econòmic / mes**
GB-Seconds	12983,994	400000	0	0$
Request Lambda	26357	1000000	0	0$

Request PUT, COPY, POST, LIST	204750	2000	202750	0,0053$ / 1000 request = 1,07$
Request GET	190125	20000	170125	0,0004$ / 1000 request = 0,07$

Tenim que el cost econòmic de l'aplicació serà de **1,14$ / mes** si seguim només les 4 lligues ja nombrades.

En el cas que volgués seguir les quasi 40 lligues que hi ha, es quedaria el següent cost:

40 lligues * 1,14$ / 4 lligues = **11,4$ / mes**

Així doncs fent el seguiment de les 40 lligues diferents que hi ha, augmentaria el cost a 11,4$ / mes.

8. Concordança de resultats i objectius

Després de tot el treball i desenvolupament fet al llarg d'aquests dies, puc afirmar que els resultats compleixen a la perfecció amb els 4 objectius principals per a l'aplicació en Android establerts a la introducció de la memòria. Anem a comprovar que realment sigui així:

Objectiu 1: Fent servir la tecnologia serverless que proporcionen les funcions Lambda d'AWS, he aconseguit desenvolupar un sistema que utilitza les funcions com a servei per a gestionar el seguiment de partits en directe, fent servir les dades que obtenim de les APIs de Riot Games. Això m'ha permès detectar quan ha passat un esdeveniment significatiu en qualsevol mapa del partit i poder-lo enviar com a notificació.

Objectiu 2: He pogut implementar exitosament les 3 llistes per a cadascun dels tipus de partit: en directe, pendents de jugar o programats i jugats. A més a més he pogut fer que siguin les 3 fàcilment accessibles gràcies a fer un menú en l'activitat principal de l'aplicació que permet alternar quina llista de partits s'està mostrant. Les dades d'aquestes llistes són obtingudes gràcies a una funció Lambda d'AWS que gestiona les dades rebudes de les APIs de Riot Games i les guarda a la base de dades de l'aplicació.

Objectiu 3: He pogut utilitzar el servei de missatgeria per tòpics de Firebase per a enviar notificacions als usuaris. Juntament amb un simple sistema d'usuaris fent servir el login de Google, m'ha permès fer que els usuaris es puguin subscriure als partits com si fossin tòpics, per així poder rebre les notificacions dels partits que volen seguir en directe.

Objectiu 4: Gràcies a que en l'objectiu 1 he pogut detectar els esdeveniments que han anat passant en un partit, he pogut anar-los guardant a la base de dades de Firebase. Així doncs, per a poder mostrar una llista cronològica dels esdeveniments de cada mapa de cada partit, només fa falta recuperar les dades ja guardades per anar-les mostrant a l'aplicació.

8.1. Coses a millorar

8.1.1. Aplicació

Hi ha moltes coses a millorar des de la part de l'aplicació. Començant pel primer gran problema: no es guarda cap tipus d'informació en caché. Això vol dir

que cada cop que s'obri haurà de descarregar totes les dades de 0 des de la base de dades, des de les imatges dels logos dels equips com els partits o les subscripcions de l'usuari que està loguejat. Així doncs que mentres no hi hagi gaire informació per a mostrar no hi haurà cap problema, però a la que la quantitat de partits sigui més nombrosa, tardarà més en carregar i en algun punt pot arribar a generar algun conflicte que fa que es tanqui l'aplicació.

Una millora que hauria de ser important també seria polir gràficament els elements visuals de l'aplicació, que al no ser un objectiu principal, he fet el que era més senzill però alhora més fàcilment llegible.

La resta serien petites millores en l'experiència de l'usuari, així com alternatives de login (no tenir només el de Google), poder filtrar els partits per lligues o equips i poder-se subscriure a tots els partits d'una lliga o d'un equip.

8.1.2. Funcions Lambda d'AWS

Crec que es podria optimitzar l'ús de les funcions Lambda una mica més, però el més important és la limitació en la quantitat de triggers que pot tenir una funció (com he explicat en un dels problemes d'escalabilitat), que provoca que la quantitat de lligues i partits programats que es puguin veure a l'aplicació es vegi altament reduïda. Ja he aportat algunes solucions en l'apartat corresponent, i si realment hagués pogut fer alguna d'aquestes, igualment hauria d'haver alguna millora de rendiment i el filtre per lligues per a que l'aplicació es pogués utilitzar en condicions amb tants partits.

8.1.3. Detecció d'esdeveniments de mapes

Sempre hi haurà alguna coseta més a mostrar aquí, com per exemple saber a quin oponent s'ha assassinat, quan hi ha un ace (vol dir que un equip ha assassinat a tots els jugadors de l'oponent, o que no queda cap viu, tenint en compte que tornen a aparèixer al joc passats uns segons) o detectar casos molt extrems de partides, com els remakes (vol dir que hi ha hagut un error en algun lloc del mapa, ja sigui tècnic o humà, molt gros i s'ha de tornar a començar de 0) o el chronobreak (és un recurs que s'utilitza per rebobinar un mapa quan hi ha hagut un error del joc o una interacció entre elements d'aquests no previst).

9. Conclusions

Aquest ha estat un projecte molt interessant de fer ja que m'ha permès aprendre i treballar en tots els aspectes que comporten la creació d'una aplicació Android des de 0, amb una utilitat real i funcional. A més a més he pogut investigar i aprendre sobre el funcionament de la computació serverless, més concretament, les funcions com a servei com són les funcions Lambda d'AWS.

Les funcions Lambda d'Amazon Web Services han resultat ser la tecnologia serverless que necessitava, ja que aquest servei es fa servir única i exclusivament quan hi ha un partit jugant-se en viu. Així doncs, m'han permès optimitzar els temps d'execució, la quantitat de recursos necessaris per dur a terme cada tasca i reduir al màxim la quantitat de temps que es fan servir aquests. Minimitzant en gran mesura el cost econòmic en comparació amb l'ús de tecnologia de servidors.

Una base de dades NoSQL com és Firebase m'ha permès organitzar les meves dades d'una manera més còmoda i ha estat molt fàcil d'interactuar, ja que només guardo i agafo les dades directament, no he de fer cap query complicada per filtrar-les (que llavors potser hagués estat millor una base de dades SQL). A més a més, Firebase ha estat molt senzill de treballar i integrar tant amb el backend com amb el frontend, fent així que no m'hagi hagut de preocupar gaire per guardar i recuperar les dades. Per acabar, el servei de missatgeria del núvol de Firebase (Firebase Cloud Messaging) m'ha solucionat de manera molt ràpida i efectiva l'enviament de notificacions als usuaris subscrits als partits.

Kotlin ha resultat ser un llenguatge molt intuïtiu d'aprendre, de treballar i de llegir, a part de tenir una documentació molt ben organitzada i neta, fet que ha facilitat en gran mesura la transició de Java a Kotlin. Gràcies a això, l'aplicació en Android s'ha pogut desenvolupar bastant fàcilment, havent-me de preocupar quasi només pels elements individuals de l'aplicació i recuperar els conceptes apresos a Projecte Integrat de Software.

Treballar amb les APIs de Riot Games ha costat molt de temps de prova i error ja que les dades que es poden obtenir d'aquestes no estan fetes específicament per l'ús que els hi he donat en aquest projecte. Així que he hagut de preguntar múltiples cops dubtes a l'Alberto o provar coses pel meu compte per veure com interactuaven amb les funcionalitats que volia implementar. També cal tenir en compte que les dades estan molt ben organitzades i descrites, fet que ha facilitat en gran mesura tot aquest procés de descobriment i investigació.

Per acabar, donar les gràcies a l'Alberto Guerra que m'ha permès tenir accés a les APIs privades de Riot Games i m'ha ofert la seva tutelació juntament amb l'Eloi Puertas al llarg de tot el desenvolupament d'aquest projecte.

10. Bibliografia

1. Akiwatkar R. AWS Lambda vs Azure Functions vs Google Cloud Functions: Comparing Serverless Providers [Internet]. Simform - Product Engineering Company. 2020 [citat 2 gener 2023]. Disponible a: https://www.simform.com/blog/aws-lambda-vs-azure-functions-vs-google-functions/

2. Baron Nashor (League of Legends) [Internet]. League of Legends Wiki. [citat 14 gener 2023]. Disponible a: https://leagueoflegends.fandom.com/wiki/Baron_Nashor_(League_of_Legends)

3. Kotlin (lenguaje de programación). En: Wikipedia, la enciclopedia libre [Internet]. 2022 [citat 2 gener 2023]. Disponible a: https://es.wikipedia.org/w/index.php?title=Kotlin_(lenguaje_de_programaci%C3%B3n)&oldid=145135188

4. Kotlin vs Java, diferencias y ventajas [Internet]. OpenWebinars.net. 2021 [citat 12 gener 2023]. Disponible a: https://openwebinars.net/blog/kotlin-vs-java/

5. LCK/2023 Season/Spring Season [Internet]. Leaguepedia | League of Legends Esports Wiki. [citat 8 gener 2023]. Disponible a: https://lol.fandom.com/wiki/LCK/2023_Season/Spring_Season

6. LCS/2023 Season/Spring Season [Internet]. Leaguepedia | League of Legends Esports Wiki. [citat 8 gener 2023]. Disponible a: https://lol.fandom.com/wiki/LCS/2023_Season/Spring_Season

7. League of Legends. En: Viquipèdia, l'enciclopèdia lliure [Internet]. 2022 [citat 2 gener 2023]. Disponible a: https://ca.wikipedia.org/w/index.php?title=League_of_Legends&oldid=30805569

8. LEC/2023 Season/Winter Season [Internet]. Leaguepedia | League of Legends Esports Wiki. [citat 8 gener 2023]. Disponible a: https://lol.fandom.com/wiki/LEC/2023_Season/Winter_Season

9. LPL/2023 Season/Spring Season [Internet]. Leaguepedia | League of Legends Esports Wiki. [citat 8 gener 2023]. Disponible a: https://lol.fandom.com/wiki/LPL/2023_Season/Spring_Season

10. Precios Amazon Web Service S3 | Amazon Simple Storage Service [Internet]. Amazon Web Services, Inc. [citat 17 gener 2023]. Disponible a: https://aws.amazon.com/es/s3/pricing/

11. Precios de Amazon SQS | Servicio de cola de mensajes | AWS [Internet]. Amazon Web Services, Inc. [citat 17 gener 2023]. Disponible a: https://aws.amazon.com/es/sqs/pricing/

12. ¿Qué es la informática sin servidor? | Definición de Sin servidor [Internet]. Cloudflare. [citat 2 gener 2023]. Disponible a: https://www.cloudflare.com/es-es/learning/serverless/what-is-serverless/

13. Serverless computing. En: Wikipedia, la enciclopedia libre [Internet]. 2022 [citat 2 gener 2023]. Disponible a: https://es.wikipedia.org/w/index.php?title=Serverless_computing&oldid=147975668

14. Topic messaging on Android | Firebase Cloud Messaging [Internet]. [citat 19 gener 2023]. Disponible a: https://firebase.google.com/docs/cloud-messaging/android/topic-messaging

15. Viewing your monthly charges - AWS Billing [Internet]. [citat 17 gener 2023]. Disponible a: https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/invoice.html#understanding-billing-details

16. What is Amazon S3? - Amazon Simple Storage Service [Internet]. [citat 17 gener 2023]. Disponible a: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

11. Annexos

11.1. Imatges

El següent enllaç conté una carpeta anomenada Annexos Memòria on estan les imatges corresponents als diagrames UML de l'estructura de cada JSON retornat per cada API de Riot Games, a més del diagrama de flux de les funcions Lambda d'AWS i el diagrama de les col·leccions de la base de dades de Firebase.

https://drive.google.com/drive/folders/1qrJMudMeOLeMyBvzpq5zYPq6stQGLTrE?usp=sharing

11.2. Codi font

En la carpeta Codi Font hi ha 3 subcarpetes:

- **APK**: Conté el fitxer app-debug.apk que es pot instal·lar a un dispositiu Android per a poder provar l'aplicació (es pot fer en un simulador d''Android mateix, com per exemple Bluestacks).

- **Funcions Lambda**: Conté una carpeta comprimida en zip per a cada funció Lambda amb el codi corresponent. També conté una subcarpeta que es diu Layer amb el Dockerfile per a crear el contenidor necessari per a muntar la layer i així poder executar les funcions. Dins la mateixa subcarpeta Layer hi ha una carpeta comprimida amb la Layer ja muntada. Així doncs si es vol provar el codi mitjançant AWS només s'han de crear les funcions tal qual venen donades amb els corresponents zip i crear i afegir la layer ja muntada.

- **Projecte Android Studio**: Conté una carpeta zip amb tot el projecte d'Android Studio, així doncs si es vol provar l'aplicació també es pot fer a través del mateix simulador que ofereix Android Studio.