LAB University of Applied Sciences

Sijaintitietoa hyödyntävän monialustaisen mobiilisovelluksen kehittäminen

LAB-ammattikorkeakoulu
Insinööri (AMK), Tieto- ja viestintätekniikka
Syksy 2020
Iiro Lahdenmaa

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Lahdenmaa, Iiro	Opinnäytetyö, AMK	Syksy 2020
	Sivumäärä	
	42	

Työn nimi

Sijaintitietoa hyödyntävän monialustaisen mobiilisovelluksen kehittäminen

Tutkinto

Insinööri (AMK)

Ohjaavan opettajan nimi, titteli ja organisaatio

Matti Welin, yliopettaja, Tieto- ja viestintätekniikka

Toimeksiantajan nimi, titteli ja organisaatio

Young Success Oy

Tiivistelmä

Opinnäytetyössä toteutettiin monialustainen, skaalautuva ja sijaintitietoa hyödyntävä mobiilisovellus sekä sovelluksen vaatima infrastruktuuri Amazon Web Services -pilvipalveluun.

Mobiilisovelluksen kehityksessä käytettiin Flutter-ohjelmistokehityspakettia ja infrastruktuuri toteutettiin IaC-menetelmällä, käyttäen Terraform-ohjelmistoa ja Serverless-sovelluskehystä.

Opinnäytetyössä perehdytään sovelluksessa käytettyihin teknologioihin kuten Flutter, Serverless, Terraform, GraphQL sekä Amazon Web Services -pilvipalvelun infrastruktuuriin. Toteutusvaiheessa sovellukselle suunnitellaan ja kehitetään käyttöliittymä, käyttäjänhallinta, tietokannat, GraphQL-rajapinta sekä laitteen sijaintitietoon perustuvat hakutoiminnot.

Opinnäytetyön lopputuloksena oli tilaajan toiminnalliset -ja tekniset vaatimukset täyttävä mobiilisovellus, sekä sen käyttämä pilvipalveluinfrastruktuuri.

Asiasanat

Flutter, Amazon Web Services, mobiilisovellus

Abstract

Author(s)	Type of Publication	Published
Lahdenmaa, Iiro	Bachelor´s thesis	Autumn 2020
	Number of Pages	
	42	

Title of Publication

Development of cross-platform mobile application that utilizes location data

Name of Degree

Bachelor of Engineering

Name, title and organization of the supervising teacher

Matti Welin, Principal Lecturer, Information and Communications Technology

Name, title and organization of the client

Young Success Oy

Abstract

The objective of this thesis was to design and develop a scalable cross-platform mobile application that utilizes location data, and the required cloud service infrastructure for the application.

The mobile application was developed using the Flutter software development kit, and the infrastructure was implemented using infrastructure as a code technique and tools such as Terraform and Serverless Framework.

This thesis introduces the technologies and services, which were used in the application. These technologies and services include Flutter, Serverless, Terraform, GraphQL, and Amazon Web Services cloud infrastructure. The implementation phase of the study goes through the design and development of user interface, user management, databases, GraphQL API, and location-based search functionalities for the application.

The result was a mobile application and cloud service infrastructure that fulfills the technical and functional requirements that were set by the client.

Keywords

Flutter, Amazon Web Services, mobile application

Sisällys

1 Johdanto

Tässä opinnäytetyössä toteutetaan laitteen sijaintitietoa hyödyntävä monialustainen mobiilisovellus sekä kehitettävän mobiilisovelluksen vaatima infrastruktuuri Amazon Web Services -pilvipalveluun. Opinnäytetyössä esitellään sovelluksen kehityksessä käytetyt työkalut ja menetelmät sekä perehdytään niiden hyödyntämiin teknologioihin.

Opinnäytetyössä kehitettävän sovelluksen lähtökohtana on Young Success Oy:n tarve konseptitasoiselle mobiilisovellukselle, joka hyödyntää laitteen sijaintitietoa yhdistääkseen samoja urheilulajeja harrastavia käyttäjiä. Young Success Oy on suomalainen digitaaliseen markkinointiin keskittyvä yritys, joka on toiminut vuodesta 2011. Tässä projektissa kehitettävän sovelluksen on tarkoitus toimia pohjana jatkokehitettävälle sovellukselle, joka julkaistaan Apple App Store -ja Google Play sovelluskaupoissa.

Projektin tarkoituksena on suunnitella ja kehittää monialustainen mobiilisovellus, käyttäen Flutter-ohjelmistokehityspakettia sekä mobiilisovelluksen tarvitsema infrastruktuuri IaC-menetelmää (Infrastructure as Code) käyttäen. Sovelluksen tekniset -ja toiminnalliset vaatimukset, ovat tilaajayrityksen määrittelemiä ja ne tähtäävät sovelluksen skaalautuvuuteen suurelle määrälle käyttäjiä. Projektin sisältöön ei kuulu sovelluksen julkaisu sovelluskaupoissa.

2 Käytetyt työkalut ja menetelmät

2.1 Flutter

Flutter on Googlen kehittämä avoimen lähdekoodin ohjelmistokehityspaketti, jolla voidaan kehittää sovelluksia useille eri alustoille, kuten Android, iOS, Windows, Linux sekä Mac OSX (Google a). Tässä opinnäytetyössä keskitytään Flutter-ohjelmistokehityspaketin mobiilikehitystoimintoihin Android- ja iOS-alustoille. Ensimmäinen vakaa Flutter versio julkaistiin joulukuussa 2018. Tämän opinnäytetyön kirjoittamishetkellä uusin julkaistu Flutter versio oli 1.20.4. (Google b.)

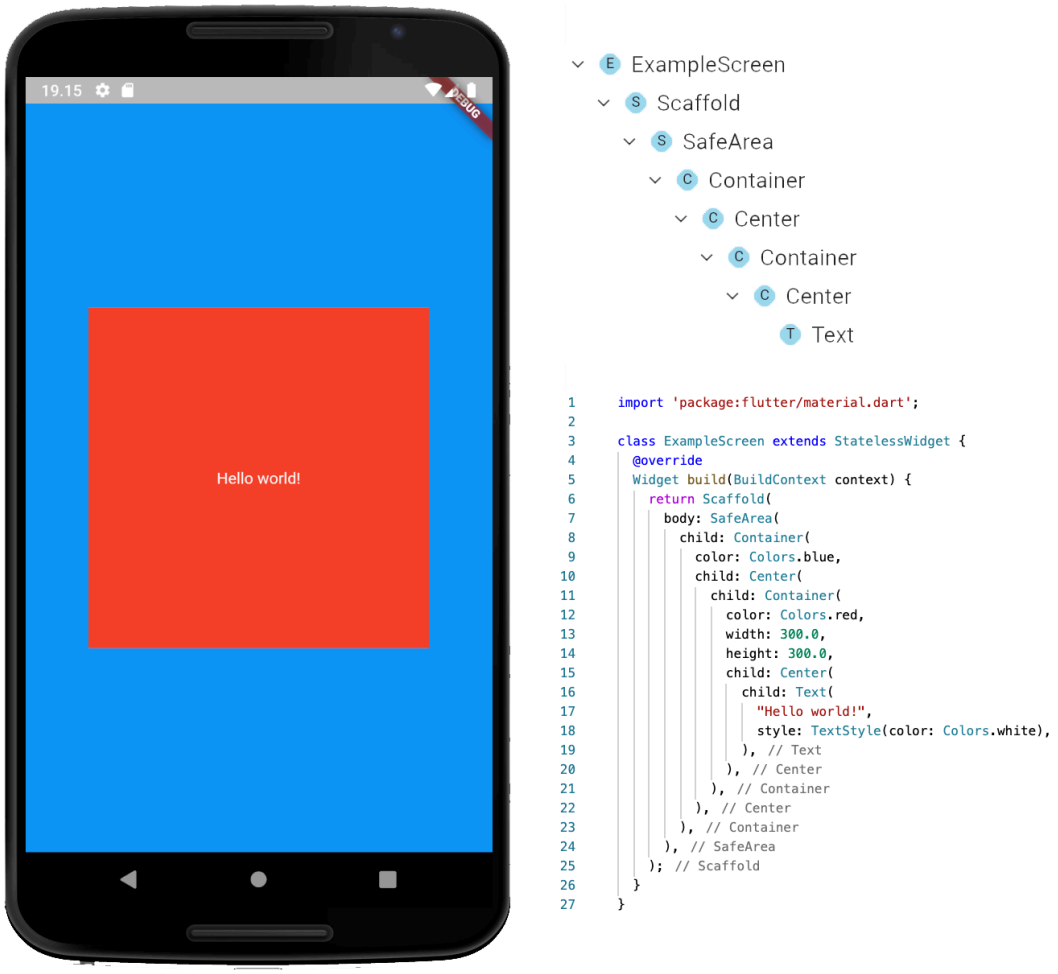
Flutter-ohjelmistokehityspaketti koostuu kahdesta osasta, Flutter SDK:sta (Software Development Kit) sekä Flutter UI Frameworkista. Flutter SDK sisältää tarvittavat työkalut, joiden avulla kehitetty ohjelmisto käännetään valitun alustan natiiville kielelle. Flutter UI Framework -sovelluskehys on kokoelma uudelleenkäytettäviä käyttöliittymäelementtejä, jotka mahdollistavat nopean alustariippumattoman käyttöliittymän kehittämisen. (Gaël 2020.)

Flutter-ohjelmistokehityspaketin keskeinen ajatus on mahdollistaa useilla eri alustoilla toimivan sovelluksen kehittäminen yhtä ohjelmointikieltä käyttäen. Flutter sovelluksia kehitetään käyttäen Dart-ohjelmointikieltä. Flutterin etu natiiviin mobiilikehitykseen verrattaessa on sen keskitetty lähdekoodi, joka käännetään kunkin alustan natiiville kielelle. Keskitetty lähdekoodi säästää ohjelmistokehitykseen kuluvaa aikaa, koska samaa toiminnallisuutta ei ole tarvetta kirjoittaa erikseen eri alustoille. Myös sovellusten lähdekoodin ylläpito ja uusien toiminnallisuuksien lisääminen sovellukseen yksinkertaistuu, kun käytössä on yksi sovelluksille yhteinen lähdekoodi. (Google a.)

Flutter-sovellukset koostuvat Widget-elementeistä, jotka muodostavat sovelluksen hierarkian. Useista muista alustariippumattomista ohjelmointikehyksistä poiketen, Flutter-sovellukset koostuvat ainoastaan edellä mainituista Widget-elementeistä, jotka ovat vastuussa käyttöliittymän toiminnasta ja ulkoasusta. Myös käyttöliittymän elementtien asemointi tapahtuu Widgettien avulla. Flutterin Widget-kirjastosta löytyy esimerkiksi Align Widget, jonka avulla voidaan asemoida kyseisen Widgetin lapseksi määritelty Widget sovelluksen näkymässä. (Google c.)

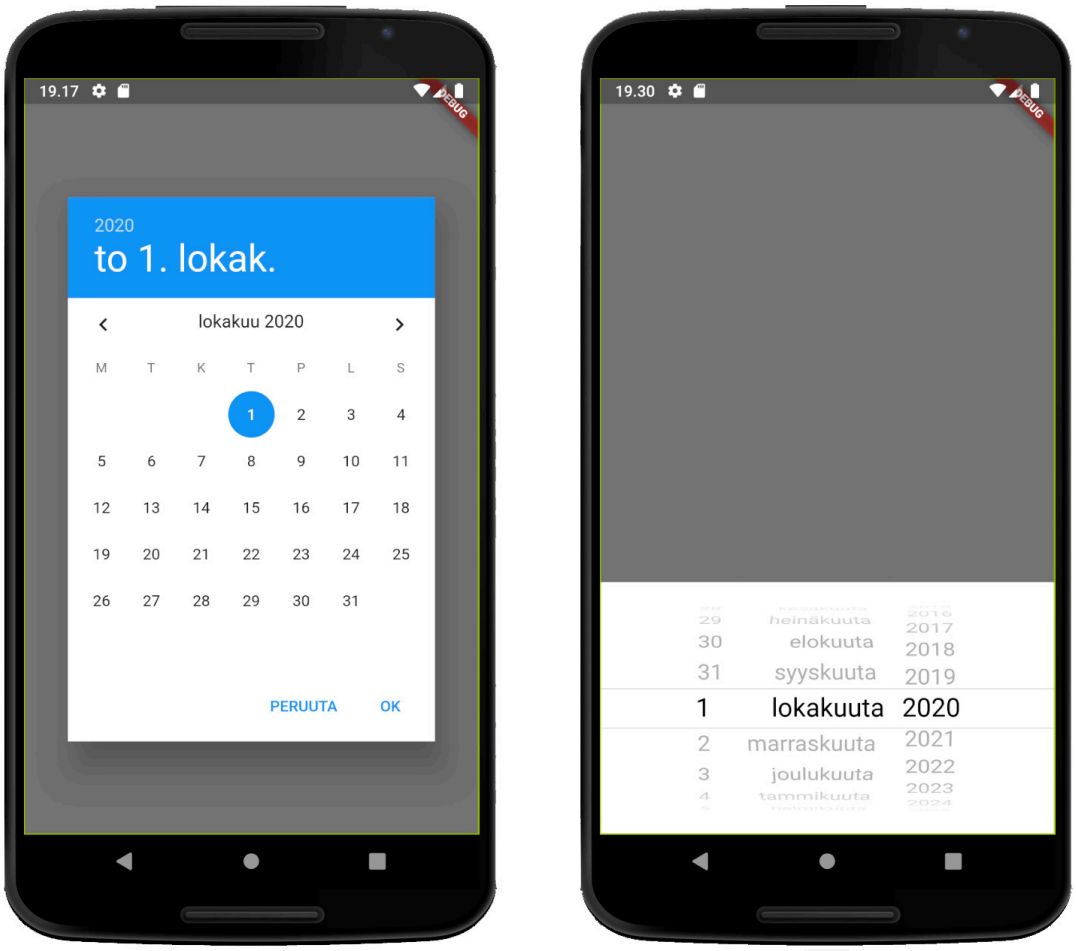
Flutter-sovelluksen Widget-elementit ovat suhteessa toisiinsa niin kutsutulla parent-child suhteella. Tieto Widget-elementtien välillä kulkee hierarkiassa ylemmältä tasolta alaspäin. Sovelluksen Widgetit voivat sisältää myös toiminnallisuutta, kuten reagoida käyttäjän syötteisiin tai eleisiin. Yhdistämällä Widgettejä toisiinsa, voidaan luoda uusia Widgettejä, joilla saavutetaan sovellukselle haluttuja toiminnallisuuksia. Flutter tarjoaa suuren määrän valmiita Widgettejä, joiden avulla voidaan toteuttaa yleisimpiä mobiilisovellusten tarpeita. Flutter-sovelluksen näkymää muutetaan korvaamalla Widgettejä uusilla Widgeteillä, jolloin käyttöliittymä päivittyy uuden Widget-hierarkian mukaiseksi. (Google c.)

Kuvassa (kuva 1) on esitetty yksinkertainen Flutter-mobiilisovelluksen käyttöliittymänäkymä, sekä sen Widget-hierarkia ja Dart-koodi. Ylimmän tason Widget näkymässä on Scaffold Widget, joka toteuttaa Material Design -asettelun mukaisen kehyksen näkymälle. Scaffold Widgetin alapuolella hierarkiassa on SafeArea Widget, joka lisää elementin reunoille täytteen, joka varmistaa, että alempana hierarkiassa olevan Widgetin sisältö ei jää käyttöjärjestelmän vaatiman näyttötilan alle. SafeArea Widgetin alapuolella hierarkiassa on Container Widget, jonka väriksi on määritelty sininen. Container Widgetin alapuolella hierarkiassa on Center Widget, joka asemoi itsensä hierarkiassa yläpuolella olevan Widgetin keskelle. Center Widgetin alapuolella hierarkiassa on toinen Container Widget jonka väriksi on määritelty punainen. Container Widgetin alapuolella hierarkiassa on toinen Center Widget, jonka alapuolella hierarkiassa on Text Widget, joka piirtää käyttöliittymään tekstin ”Hello world!”.



Kuva 1. Mobiilisovelluksen käyttöliittymänäkymä, Flutter Widget -hierarkia sekä Dart-koodi

Flutter pyrkii tarjoamaan mobiilisovelluskehityksen tueksi työkaluja, joiden avulla saavutetaan lähes natiivitason suorituskyky alustariippumattomassa sovelluskehitysympäristössä (Feoktistov). Flutter tarjoaa myös natiivia käyttöliittymäkokemusta jäljitteleviä UI-elementtejä sekä Android-, että iOS-alustoille. Flutter-sovelluskehyksen käyttöliittymäkomponenttikirjastosta löytyy Android-käyttöjärjestelmän natiiveja UI-komponentteja sisältävä Material Design -elementtikirjasto sekä iOS UI-komponentteja sisältävä Cupertino-elementtikirjasto. (Google d.) Kuvassa (kuva 2) on esitetty Android-käyttöjärjestelmälle suunnatun Material Design -kirjaston, sekä iOS käyttöjärjestelmälle suunnatun Cupertino-kirjaston päivämäärän valitsemiseen tarkoitetut Widgetit.



Kuva 2. Material Design- ja Cupertino-käyttöliittymäkomponenttikirjastojen päivämäärän valitsemiseen tarkoitetut Widgetit

2.2 Dart

Flutter-sovelluksia kehitetään käyttämällä Dart-ohjelmointikieltä. Dart on Googlen kehittämä luokkapohjainen olio-ohjelmointikieli, joka voidaan kääntää natiiviksi konekieleksi Android- ja iOS-laitteille, sekä JavaScript-ohjelmakoodiksi (Google e). Dart-ohjelmointikieli esiteltiin ensimmäisen kerran vuonna 2011 ja versio 1.0 julkaistiin vuonna 2013 (Balbaert & Ridjanovic 2013, 9). Tämän opinnäytetyön kirjoittamishetkellä uusin julkaistu vakaa Dart versio oli 2.10.0 (Google f).

Dart-ohjelmointikielellä kirjoitettua lähdekoodia voidaan suorittaa Dart-ohjelmistokehityspakettiin sisältyvässä Dart Virtual Machine -ympäristössä. Dart Virtual Machine voi kääntää koodia kahdessa eri tilassa JIT (Just-In-Time) sekä AOT (Ahead-Of-

Time). (Google e.) Flutter jaottelee eri Dart-ohjelmistokoodin kääntämistilat debug sekä release tilaan.

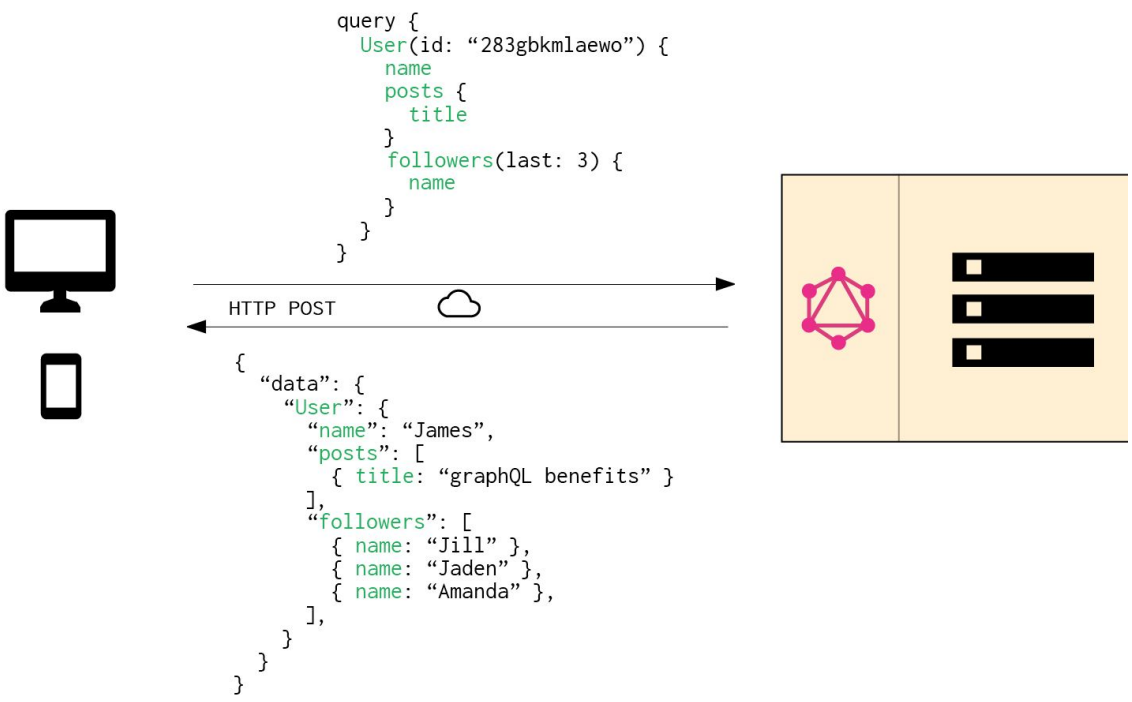
Debug-tila on tarkoitettu sovelluksen kehittämisen aikaiseen ohjelmistokoodin suorittamiseen. Debug-tilassa koodi käännetään binääriseen muotoon ohjelmakoodin suorituksen aikana (JIT). JIT-kääntäminen mahdollistaa useita ohjelmistokehitystä nopeuttavia ja helpottavia ominaisuuksia, kuten Flutter-ohjelmistokehityspaketin Hot Reload -toiminnon, jossa kehittäjän tekemät muutokset lähdekoodissa ovat lähes reaaliaikaisesti nähtävillä kehitettävässä käyttöliittymässä, ilman että sovelluksen tila menetetään. Suorituksen aikana tehdyn kääntämisen haittapuolena on sovelluksen suoritustason heikentyminen, sillä osa saatavilla olevista resursseista käytetään koodin kääntämiseen. (Google g.)

Release-tila on tarkoitettu valmiin sovelluksen kääntämiseen ja suorittamiseen. Release-tilassa sovelluksen lähdekoodi käännetään binäärimuotoon ennen ohjelmakoodin suorittamista (AOT). Release-tilassa sovellus voi käyttää kaikki saatavilla olevat resurssit sovelluksen suorittamiseen, joka johtaa debug-tilaa parempaan sovelluksen suoritustasoon. (Google g.)

2.3 AWS AppSync

AWS AppSync on vuonna 2017 julkaistu, Amazon Web Services -yrityksen kehittämä palvelimetonta arkkitehtuuria käyttävä palvelu, joka mahdollistaa GraphQL-viestintäprotokollaa käyttävien rajapintojen luomisen Amazon AWS -pilvipalveluun (Serverless Inc).

Jokainen GraphQL-kysely sisältää kuvauksen sen odottaman vastauksen sisällöstä, jolloin suoritettu kysely palauttaa ainoastaan kyselyn yhteydessä määritetyt tiedot (The GraphQL Foundation a). Tämä mahdollistaa yhden keskitetyn rajapinnan käyttämisen. GraphQL-API kerros on vastuussa kyselyn käsittelemisestä sekä tarvittavien tietolähteiden yhdistämisestä toisiinsa (Serverless Inc). Kuvassa (kuva 3) on esitetty GraphQL-kysely, joka hakee ja yhdistää tiedot kolmesta eri tietolähteestä. Kyselyssä on kuvattuna vastauksen rakenne, sekä tieto siitä, mitä tietoja kyselyn halutaan palauttavan.



Kuva 3. QraphQL-kysely (Devathon 2020)

GraphQL-kyselyt tukevat kolmea eri operaatiotyyppiä, query, mutation ja subscription. Query-operaatio hakee tietoa, mutation-operaatio lisää tai muokkaa olevassa olevaa tietoa ja subscription muodostaa pitkäkestoisen yhteyden, joka vastaanottaa tietoa aina kun subscription-kyselyssä määritetty tapahtuma tapahtuu. (Serverless Inc.)

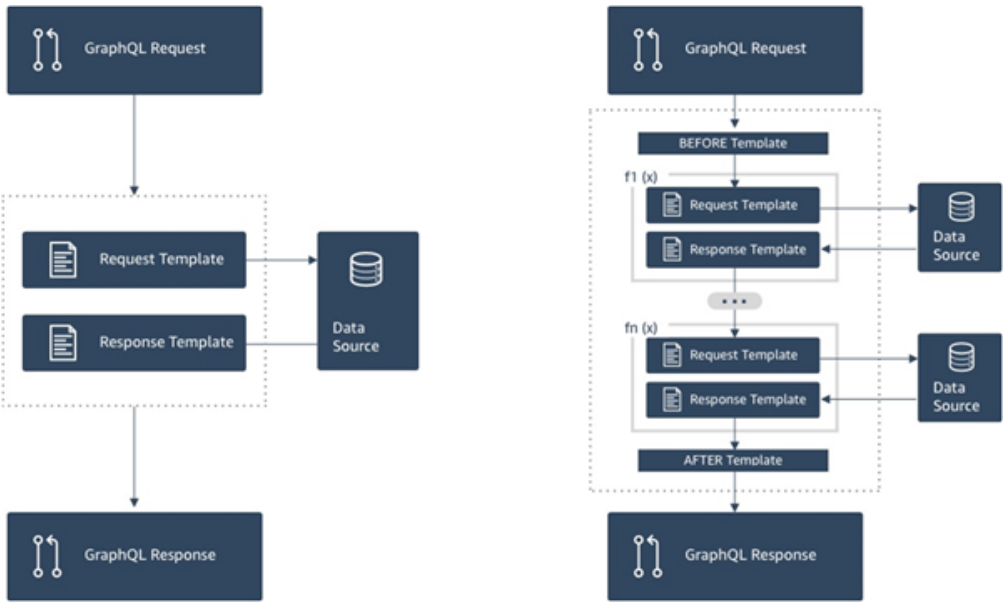
GraphQL-API:n toimintaa kuvataan GraphQL-schemalla, joka kirjoitetaan GraphQL:n omalla SDL-kielellä (Schema Definition Language). GraphQL-schemassa määritetään rajapinnan query, mutation ja subscription-operaatiot, sekä muut rajapinnan toiminnan kannalta oleelliset tiedot kuten tietokenttien tietotyypit. (The GraphQL Foundation b.) Kuvassa (kuva 4) on esitettynä esimerkki SDL-kielellä kirjoitetusta GraphQL-API:n schemasta.

```
schema {
  query: Query
  mutation: Mutation
}

type Query {
  getUsers: [User]
}

type Mutation {
  addUser(id: ID!, name: String, description: String, status: UserStatus): User
}

type User {
  id: ID!
  name: String
  description: String
  status: UserStatus
}

enum UserStatus {
  confirmed
  pending
}

```

Kuva 4. Schema Definition Language kielellä kirjoitettu koodi

AWS AppSync on suunniteltu yhdistämään eri Amazon Web Services -palvelut saumattomasti toisiinsa. AppSync mahdollistaa muun muassa haut Amazon DynamoDB -tietokannasta, sekä AWS Lambda -funktioiden kutsumisen osana GraphQL-kyselyä. AppSync-rajapintaan tehtävät kyselyt voidaan autentikoida käyttäen API-avainta, AWS IAM -käyttäjäroolia tai Amazon Cognito -palvelun käyttäjänhallintaa. (Serverless Inc.)

AWS AppSync -kyselyille määritellään resolver-funktio, joka käsittelee kyselyn ja vastaa kysytyn tiedon palauttamisesta. Kyselyt määritellään GraphQL-schemassa tietotyypeillä Query, Mutation ja Subscription. Resolver funktio voidaan määritellä Unit Resolveriksi, joka koostuu yhdestä funktiosta ja tietolähteestä, tai Pipeline Resolveriksi, joka muodostuu useista funktioista, joilla kaikilla voi olla oma erillinen tietolähteensä. Pipeline Resolverin sisältämät funktiot suoritetaan järjestyksessä ja niiden käsittelemää tietoa voidaan siirtää funktiosta toiseen suorituksen aikana. (Amazon Web Services Inc g.) Kuvassa (kuva 5) on esitetty Unit Resolverin- ja Pipeline Resolverin -suoritusprosessi kaaviona. Vasemmalla kuvassa on kuvattuna Unit Resolverin toiminta ja oikealla Pipeline Resolverin toiminta.



Kuva 5. AppSync Unit Resolver- sekä Pipeline Resolver -suoritusprosessikaavio (Amazon Web Services Inc g)

Resolver-funktioissa voidaan tietolähteenä käyttää esimerkiksi Amazon DynamoDB -tietokantaa, mutta tietolähde voi olla myös erillisen AWS Lambda -funktion suoritus. AWS Lambda -funktiot mahdollistavat esimerkiksi tiedon hakemisen kolmansien osapuolien REST API -rajapinnoista ja niiden palauttaman tiedon käyttämisen osana Pipeline Resolverin -suoritusta. (Serverless Inc a.) AWS AppSync -funktioita ohjelmoidaan käyttäen JAVA-ohjelmointikieleen perustuvaa Apache Velocity Template Language (VTL) -kieltä (Amazon Web Services Inc g). Kuvassa (kuva 6) on esitetty esimerkki Apache Velocity Template Language -kielellä kirjoitetusta AppSync-funktiosta, joka lisää uuden tietueen Amazon DynamoDB tietokantaan.

```
{
    "version": "2017-02-28",
    "operation": "PutItem",
    "key": {
      "id": $util.dynamodb.toDynamoDBJson($ctx.args.id)
    },
    "attributeValues": $util.dynamodb.toMapValuesJson($ctx.args)
  }

```

Kuva 6. Apache Velocity Language -kielellä kirjoitettu koodi

2.4 Amazon DynamoDB

Amazon DynamoDB on Amazonin AWS-pilvialustalla toimiva ylläpidetty NoSQL-tietokantapalvelu. Amazon DynamoDB -tietokantaan voidaan tallentaa tietoa dokumenttimuodossa tai avain-arvopareina. Amazon DynamoDB:n vahvuuksiin kuuluu sen skaalautuvuus, joustava tietorakenne sekä suorituskyky. Amazon DynamoDB -tietokanta pystyy käsittelemään 10 biljoonaa kyselyä vuorokaudessa ja se selviytyy hetkellisesti 20 miljardista kyselystä sekunnissa. (Amazon Web Services Inc a.)

Amazon DynamoDB -tietokantaan tallennettuja tietueita kutsutaan termillä item, joka vastaa perinteisen relaatiotietokannan riviä. Item koostuu attribuuteista, jotka tallennetaan tietokantaan avain-arvopareina. Tämän opinnäytetyön kirjoitushetkellä Amazon DynamoDB:n tukemia attribuuttien primitiivisiä JAVA-tietotyyppejä olivat, String, Boolean, Byte, Date, Calendar, Long, Integer, Double, Float, BigDecimal ja BigInteger. Tämän lisäksi attribuutit tukevat JAVA-kokoelmatyyppejä Set, List ja Map. (Amazon Web Services Inc b.)

Amazon DynamoDB -tietokantaan lisättävät itemit tallennetaan tietokantatauluun. Taulua luodessa, taululle on määriteltävä primääriavain (primary key). Primääriavain voi muodostua jaotteluavaimesta (partition key) tai jaotteluavaimen ja lajitteluavaimen (sort key) yhdistelmästä, jolloin primääriavain on komposiittiavain. Primääriavaimessa määritetyt avaimet vastaavat tietokantaan lisätyn tiedon attribuutteja. Jokaisen tauluun lisättävän itemin primääriavaimen on oltava uniikki, sillä sitä käytetään itemin tunnistamiseen taulussa. (Balasubramanian 2017.)

Koska Amazon DynamoDB on NoSQL-tietokanta, voidaan tietokantaan tallentaa tietoa perinteistä relaatiotietokantaa joustavammin. Tietokantaan lisättävien itemien ei tarvitse sisältää kaikkia samoja attribuutteja, ainoastaan primääriavaimessa määritellyt attribuutit ovat pakollisia. NoSQL-tietokanta mahdollistaa monien eri tyyppisten tietojen tallentamisen yhteen tietokantatauluun. (DeBrie a.)

Kuvassa (kuva 7) on esitetty esimerkki kuvitteellisen verkkokaupan Amazon DynamoDB -tietokantataulun rakenteesta. Kuvan tietokantataulu sisältää kahta erilaista tietoa, jotka ovat käyttäjien profiilitiedot, sekä verkkokaupan tilausten tiedot. Tietokantataulun primääriavaimeksi on määritelty komposiittiavain, jonka jaotteluavain on tietoon liittyvän käyttäjän käyttäjänimi (Username) ja komposiittiavaimen lajitteluavain muodostetaan liittämällä yhteen tiedon tyyppi (PROFILE# tai ORDER#) ja

mahdollinen tietoon liittyvä yksilöllinen tunniste, kuten tilauksen tilausnumero (OrderId). Profiilitieto ei tarvitse yksilöllistä tunnistetta, sillä yhdellä käyttäjänimellä voi olla ainoastaan yksi profiili, joten komposiittiavaimesta muodostuu aina uniikki.

Primary Key		Attributes				
PK (Username)	**SK (Data)**					
johndoe	PROFILE#	**Name**	**Email**	**Addresses**	**CreatedBy**	**CreatedAt**
		John Doe	johndoe@example.com	{"Work" : {"StreetAddress"	johndoe	02/03/2020
	ORDER#35478	**OrderId**	**Status**	**Address**	**CreatedBy**	**CreatedAt**
		35478	PENDING	{"StreetAddress" :	johndoe	11/04/2020
	ORDER#87546	**OrderId**	**Status**	**Address**	**CreatedBy**	**CreatedAt**
		87546	PENDING	{"StreetAddress" :	johndoe	12/04/2020
	ORDER#56464	**OrderId**	**Status**	**Address**	**CreatedBy**	**CreatedAt**
		564654	FINISHED	{"StreetAddress" :	johndoe	07/06/2020
	ORDER#56494	**OrderId**	**Status**	**Address**	**CreatedBy**	**CreatedAt**
		564894	RECEIVED	{"StreetAddress" :	system	15/08/2020
janedoe	PROFILE#	**Name**	**Email**	**Adresses**	**CreatedBy**	**CreatedAt**
		Jane Doe	janedoe@example.com	{"Work" : {"StreetAddress"	janedoe	01/01/2020
	ORDER#84654	**OrderId**	**Status**	**Address**	**CreatedBy**	**CreatedAt**
		84654	RECEIVED	{"StreetAddress" :	janedoe	11/01/2020
	ORDER#65465	**OrderId**	**Status**	**Address**	**CreatedBy**	**CreatedAt**
		65465	FINISHED	{"StreetAddress" :	janedoe	15/06/2020

Kuva 7. Amazon DynamoDB tietokantataulun rakenne

Tietokantataulun komposiittiavaimen jaotteluavainta ja lajitteluavainta voidaan käyttää hyödyksi haettaessa haluttua tietoa taulusta. Lajitteluavaimeen kohdistuvassa haussa voidaan käyttää ehdollista begins_with-ilmaisua, jolloin lajitteluavaimen sisältöä ei tarvitse tietää kokonaisuudessaan. (Amazon Web Services Inc c.)

Kuvassa (kuva 8) on esitetty tietokantatauluun tehtävä haku, jossa primääriavaimen lajitteluavaimeen kohdistuvassa haussa käytetään ehdollista begins_with-ilmaisua. Tietokantahaussa haetaan kaikki itemit, joissa käyttäjänimenä on johndoe ja Data attribuutin sisältö alkaa merkkijonolla ”ORDER#”. Kuva havainnollistaa tietokannan palauttamat tiedot, jotka vastaavat annettua hakulauseketta.

```
import * as AWS from 'aws-sdk';

var docClient = new AWS.DynamoDB.DocumentClient();

var params = {
  TableName: "ExampleTable",
  KeyConditionExpression: "Username = :username AND begins_with(#data, :data)",
  ExpressionAttributeNames: {
    "#data": "Data",
  },
  ExpressionAttributeValues: {
    ":username": "johndoe",
    ":data" : "ORDER#"
  },
};

docClient.query(params, function (err, data) {
  if (err) {
    console.error("Unable to query. Error:", JSON.stringify(err, null, 2));
  } else {
    console.log("Query succeeded.");
    return data;
  }
});
```

Primary Key		Attributes				
PK (Username)	**SK (Data)**					
johndoe	ORDER#35478	**OrderId**	**Status**	**Address**	**CreatedBy**	**CreatedAt**
		35478	PENDING	{"StreetAddress" :	johndoe	11/04/2020
	ORDER#87546	**OrderId**	**Status**	**Address**	**CreatedBy**	**CreatedAt**
		87546	PENDING	{"StreetAddress" :	johndoe	12/04/2020
	ORDER#56464	**OrderId**	**Status**	**Address**	**CreatedBy**	**CreatedAt**
		564654	FINISHED	{"StreetAddress" :	johndoe	07/06/2020
	ORDER#56494	**OrderId**	**Status**	**Address**	**CreatedBy**	**CreatedAt**
		564894	RECEIVED	{"StreetAddress" :	system	15/08/2020

Kuva 8. JavaScript-koodi ja Amazon DynamoDB -tietokantataulun tietoja taulukossa

Amazon DynamoDB -tietokantataululle voidaan määrittää vaihtoehtoisia indeksejä. Vaihtoehtoiset indeksit mahdollistavat tehokkaamman pääsymallin tietokannan sisältämiin tietoihin. Vaihtoehtoiselle indeksille voidaan määrittää oma primääriavain, jonka avulla tietokannasta voidaan hakea tehokkaammin haluttua tietoa. Vaihtoehtoiset indeksit jaotellaan kahteen eri luokkaan, Local Secondary Index (LSI) ja Global Secondary Index (GSI). (Amazon Web Services Inc d.)

Local Secondary Index voidaan luoda tauluun, jonka primääriavain on komposiittiavain. Tällaisen vaihtoehtoisen indeksin primääriavaimeksi voidaan määrittää komposiittiavain, joka koostuu taulun alkuperäisen komposiittiavaimen jaotteluavaimesta ja vapaasti valitusta lajitteluavaimesta. (Amazon Web Services Inc d.)

Global Secondary Index mahdollistaa vapaavalintaisen primääriavaimen luomisen indeksille. Global Secondary Indexin primääriavain voi olla yksinkertainen primääriavain, tai komposiittiavain, riippumatta taulun alkuperäisen primääriavaimen rakenteesta. (Amazon Web Services Inc d.)

Vaihtoehtoisten indeksien primääriavaimissa määriteltyjen attribuuttien ei tarvitse olla itemeillä uniikkeja. Vaihtoehtoiset indeksit saattavat sisältää itemejä, joiden attribuuttien rakenne on identtinen. Tietokantatauluun voidaan myös lisätä itemejä, joille ei ole määritelty vaihtoehtoisien indeksien primääriavaimissa määriteltyjä attribuutteja, tällöin kyseisiä itemejä ei sisällytetä vaihtoehtoiseen indeksiin. (DeBrie b.)

Vaihtoehtoisia indeksejä luodessa voidaan määrittää indeksiin heijastettavat attribuutit. Itemistä voidaan valita heijastettavaksi kaikki attribuutit, indeksin primääriavaimen sisältämät attribuutit tai yksittäiset halutut attribuutit (Vyas 2014, 69-73).

Vaihtoehtoiset indeksit mahdollistavat monipuolisemmat pääsymallit tietokantataulun tietoihin, mutta niihin tehtävien erilaisten kyselytoimintojen määrä on suppeampi kuin taulun primääriavaimella tehtävissä kyselyissä. Amazon DynamoDB -tietokantataulun tietorakennetta suunniteltaessa tulee ottaa huomioon, mitä tietoja taulusta haetaan ja millaisia pääsymalleja tietoihin vaaditaan. Tehokas tietokantarakenteen suunnittelu mahdollistaa kustannustehokkaan tietokannan käytön. (Vyas 2014, 67-81.)

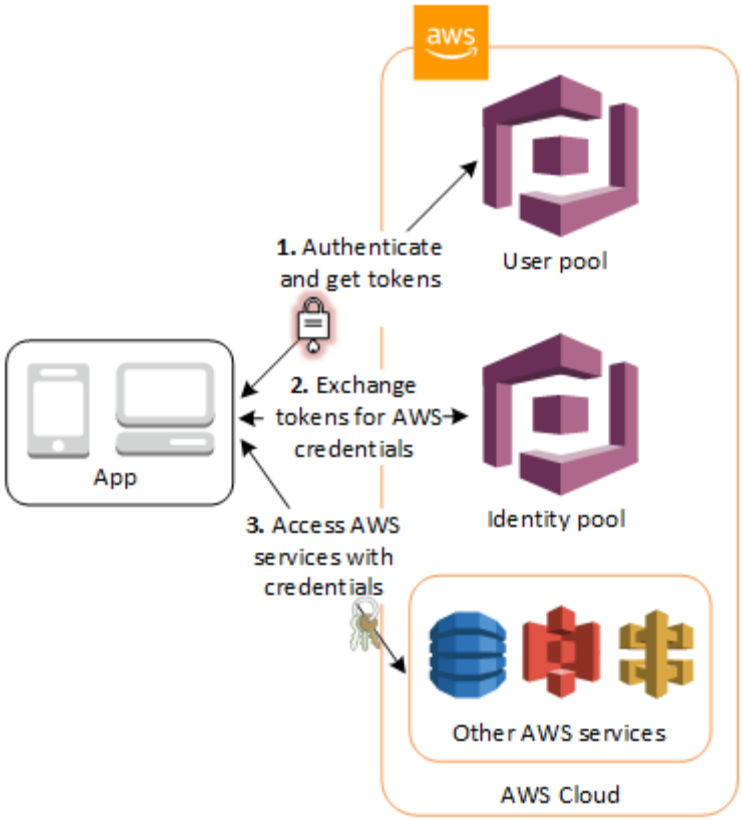
2.5 Amazon Cognito

Amazon Cognito on Amazon Web Services -yrityksen tarjoama käyttäjienhallintapalvelu, joka mahdollistaa käyttäjien rekisteröitymisen, kirjautumisen sekä käyttäjien pääsynhallinnan AWS-pilvipalvelussa toimiviin resursseihin (Amazon Web Services Inc f).

Amazon Cognito on suunniteltu skaalautumaan miljoonille käyttäjille ja se tukee käyttäjien attribuuttien vahvistusta, monivaiheista tunnistautumista (Multi Factor Authentication) sekä tunnistautumista käyttäen kolmannen osapuolen identiteetin tarjoajaa kuten Facebook, Google tai Microsoft Active Directory (Mistry 2018, 191-196).

Amazon Cognito koostuu kahdesta palvelusta, jotka ovat User Pool ja Identity Pool. User Pool on käyttäjienhallinta palvelu, joka mahdollistaa käyttäjien rekisteröitymisen ja autentikoinnin sekä käyttäjätietojen hallinnan. Identity Pool on käyttäjien pääsynhallintapalvelu, jossa voidaan hallita eri käyttäjäryhmien ja yksittäisten käyttäjien pääsyä eri AWS-pilvipalvelussa toimiviin resursseihin. (Amazon Web Services Inc e.)

Kuvassa (kuva 9) on esitetty tyypillinen Amazon Cognito -palvelun käyttötapaus vaiheittain. Vaiheessa yksi, käyttäjä autentikoidaan käyttäen User Pool -käyttäjienhallintaa, josta käyttäjä saa valtuutustunnuksen (token). Vaiheessa kaksi, käyttäjä valtuutetaan Identity Pool -palvelun avulla käyttämään määriteltyjä AWS-pilvipalvelussa toimivia resursseja. Vaiheessa kolme, käyttäjällä on pääsy AWS-pilvipalvelun resursseihin valtuutuksesta saamillaan pääsyavaimilla.



Kuva 9. Amazon Cognito -palvelun toimintaperiaate (Amazon Web Services Inc f)

Amazon Cognito on suunniteltu mobiili- ja websovellusten käyttäjien- ja pääsynhallintaan. Amazon Cognito tarjoaa kehittäjille valmiin rajapinnan tietoturvalliseen- ja vaivattomaan käyttäjienhallintaan ja se täyttää useimmat henkilökohtaisten tietojen käsittelemistä koskevat asetukset. Amazon Cogniton vahvuuksia ovat muun muassa sen skaalautuvuus kysynnän kasvaessa, sekä yksinkertainen integrointi kehitettäviin sovelluksiin, joka mahdollistaa nopeamman kehitysprosessin. (Amazon Web Services Inc f.)

2.6 Terraform

Terraform on HashiCorp-yrityksen kehittämä avoimen lähdekoodin IaC-ohjelmisto (Infrastrcture as Code), jonka ensimmäinen versio julkaistiin vuonna 2014 (Sheiks 2018). Tämän opinnäytetyön kirjoitushetkellä uusin vakaa Terraform versio oli 0.13.3 (HashiCorp a).

Terraform mahdollistaa ulkoisten resurssien, kuten pilvipalvelun infrastruktuurin kuvailemisen koodina ja sen provisioimisen pilvipalveluun (HashiCorp b). Terraform tukee useiden tunnettujen pilvipalvelutoimittajien ympäristöjä, kuten Amazon Web Services, Microsoft Azure, IBM Cloud, Google Cloud Platform, Oracle Cloud Infrastructure sekä DigitalOcean (HashiCorp c).

Infrastruktuuria kuvaillaan Terraformissa käyttäen HashiCorpin kehittämää HCL-kieltä (HashiCorp Configuration Language) (HashiCorp d). Kuvassa (kuva 10) on esitetty HCL-kielellä kuvattu Amazon Cognito -palvelun konfiguraatio, joka luo Amazon Web Services -pilvialustalle Amazon Cognito User Pool -palvelun määritellyillä asetuksilla. Kuvan koodissa on kuvattuna ”aws_cognito_user_pool”-resurssi, jossa on kuvattuna nimi palvelulle, palvelun uusille käyttäjille lähettämä viesti sekä palvelun salasanan vaatimusmääritykset. Itse palvelun lisäksi koodissa on kuvattuna ”aws_cognito_user_group”-resurssi, joka luo palveluun käyttäjäryhmän nimeltä ”examplegroup”.

terraform > cognito.tf > ...

```
resource "aws_cognito_user_pool" "example_user_pool" {
  name = "example_user_pool"

  admin_create_user_config {
    invite_message_template {
      sms_message   = "Welcome! You are now part of Example User Pool."
      email_message = "Welcome! You are now part of Example User Pool."
      email_subject = "Welcome to Example pool"
    }
  }

  password_policy {
    minimum_length                   = 8
    require_lowercase                = true
    require_numbers                  = true
    require_symbols                  = false
    require_uppercase                = true
    temporary_password_validity_days = 60
  }
}

resource "aws_cognito_user_group" "example_user_group" {
  name         = "examplegroup"
  user_pool_id = aws_cognito_user_pool.driver_user_pool.id
}
```

Kuva 10. HashiCorp Configuration Language -koodi

Infrastruktuurin kuvaileminen koodina mahdollistaa kokonaisen infrastruktuurin, tai infrastruktuurimuutosten nopean provisioimisen valitulle pilvialustalle. HCL-kielellä kuvailtu infrastruktuuri provisioidaan pilvipalveluun CLI-komennolla (Command Line Interface), jolloin Terraform suorittaa tarvittavat CRUD-operaatiot (create, read, update and delete) käyttäjän puolesta. (HashiCorp e.) Automatisoitu infrastruktuurin provisioiminen säästää kehittäjien aikaa ja pienentää konfiguraatiovirheiden riskiä (IBM Cloud Education 2020).

2.7 Serverless Framework

Serverless Framework on avoimen lähdekoodin ohjelmistokehys, joka mahdollistaa palvelitonta arkkitehtuuria käyttävien sovellusten yksinkertaisen käyttöönoton eri palveluntarjoajien pilvipaluissa. Serverless Framework tukee useimpien tunnettujen

pilvipalveluntarjoajien ympäristöjä kuten, Amazon Web Services, Google Cloud Platform ja Microsoft Azure. (Serverless Inc b.)

Tässä opinnäytetyössä Serverless Framework -ohjelmistokehystä käytettiin AWS AppSync -palveluun kehitetyn GraphQL-rajapinnan tarvitsemien AWS Lambda -funktioiden kehitykseen ja käyttöönottoon Amazon Web Services -pilvipalvelussa.

Kuvassa (kuva 11) on esitetty TypeScript-ohjelmointikielellä kirjoitettu AWS Lambda -funktio, jonka suoritus voidaan laukaista sille määritellyn tapahtuman (event) avulla. Funktio suoritetaan käyttäjän luodessa uutta käyttäjätiliä Amazon Cognito -käyttäjänhallintaan. Funktio suorittaa tarkistusprosessin, jossa tarkistetaan, löytyykö palvelusta vahvistettu käyttäjä samalla puhelinnumerolla. Mikäli käyttäjä löytyy, rekisteröinti evätään.

user-service > lambdas > TS pre-sign-up.ts > ...

```ts
import * as AWS from "aws-sdk";

const identity = new AWS.CognitoIdentityServiceProvider();

exports.handler = async (event, _context, callback) => {
  if (event.request.validationData.force_sign_up === "true") {
    console.log(`SignUp request was triggered with force flag. Proceeding with signUp...`);
    callback(null, event);
  } else if (event.request.userAttributes.phone_number) {
    const { phone_number: phone_number } = event.request.userAttributes;
    const userParams = {
      UserPoolId: event.userPoolId,
      AttributesToGet: ["phone_number", "phone_number_verified"],
      Filter: `phone_number = \"${phone_number}\"`,
    };
    try {
      const { Users } = await identity.listUsers(userParams).promise();
      if (Users && Users.length > 0) {
        console.log(`Found '${Users.length}' with matching phone number..`);
        let verifiedUserWithPhoneNumberExists = false;
        Users.find((user) =>
          user.Attributes.find((attribute) => {
            if (attribute.Name === "phone_number_verified" && attribute.Value === "true") {
              verifiedUserWithPhoneNumberExists = true;
            }
          })
        );

        if (verifiedUserWithPhoneNumberExists) {
          console.log(`Found verified user with matching phone number.`);
          console.log(`Declining signUpProcess`);
          callback("PhoneExistsException", null);
        } else {
          console.log(`No verified users with matching phone number were found. Proceeding with signUp...`);
          callback(null, event);
        }
      } else {
        callback(null, event);
      }
    } catch (error) {
      console.log("SignUp failed!", error);
      callback({ error }, null);
    }
  } else {
    callback("MissingParameters", null);
  }
};
```

Kuva 11. AWS Lambda -funktio

Serverless Framework -projekti koostuu palveluista (service), joiden kuvaus määritellään serverless.yml tiedostossa. Palvelun kuvaus sisältää tiedon käytettävästä palveluntarjoajasta (provider), palvelun käyttämistä funktioista (functions), tapahtumista, jotka laukaisevat funtioiden suorituksen (events) sekä palvelun käyttämistä resursseista (resources). (Serverless Inc c.)

Kuvassa (kuva 12) on esitetty pelkistetty esimerkki Serverless Framework -projektin palvelun kuvauksesta. Kyseisen esimerkin palvelu luo määritetyt funktiot Amazon Web Services -pilvipalveluun, mutta ei määrittele funktioiden suoritusta laukaisevia tapahtumia. Tässä opinnäytetyössä funktioiden suorituksen laukaisu tapahtuu AWS AppSync -palvelussa luodun GraphQL-rajapinnan kautta, joten tapahtumia ei tarvitse määritellä Serverless Framework -projektin palvelun kuvauksessa.

```
1   service:
2     name: example-service
3
4   provider:
5     name: aws
6     runtime: nodejs12.x
7     region: eu-west-1
8
9   functions:
10    exampleFunction1:
11      handler: lambdas/example1.handler
12    exampleFunction2:
13      handler: lambdas/example2.handler
14
```

Kuva 12. Serverless Framework -projektin palvelun kuvaus

3 Suunnittelu

3.1 Sovelluksen toiminnalliset- ja tekniset vaatimukset

Kehitettävälle sovellukselle oli määritetty sekä toiminnallisuuteen että toteutukseen liittyviä teknisiä vaatimuksia. Kaikki toteutukseen liittyvät vaatimukset tähtäsivät sovelluksen jatkokehitysmahdollisuuksien- ja tuotantoversion skaalautuvuuden mahdollistamiseen.

Toiminnalliset vaatimukset

Käyttäjän oli pystyttävä luomaan sovelluksessa käyttäjätili, kirjautumaan sisään sovellukseen luomallaan käyttäjätilillä sekä kirjautumaan ulos sovelluksesta. Sovelluksen käynnistymisen yhteydessä tuli tarkistaa onko käyttäjä kirjautuneena sovellukseen ja ohjata käyttäjä joko suoraan sisään sovellukseen tai sisäänkirjautumisnäkymään.

Sisäänkirjautumisen jälkeen käyttäjän oli pystyttävä määrittelemään itselleen omat henkilökohtaiset asetukset. Käyttäjän henkilökohtaisten asetusten tuli sisältää hakusäde, jonka sisältä käyttäjä haluaa löytää samaa urheilulajia harrastavia käyttäjiä, käyttäjän omat harrastukset, sekä oma taitotaso kussakin harrastuksessa. Käyttäjän tuli pystyä muuttamaan asetuksia niiden asettamisen jälkeen. Asetuksissa määriteltävät harrastukset tuli ladata sovellukseen tietokannasta, jolloin uusien harrastusmahdollisuuksien lisääminen sovellukseen voi jatkossa tapahtua erillisen hallintatyökalun avulla.

Omien asetusten määrittelemisen jälkeen käyttäjän oli pystyttävä näkemään listaus samaa lajia harrastavista käyttäjistä, jotka sijaitsevat käyttäjän määrittelemän hakusäteen sisällä. Samaa urheilulajia harrastavien käyttäjien listaus tuli pystyä lajittelemaan harrastuksen perusteella.

Tekniset vaatimukset

Sovelluksen kehityksessä tuli käyttää Flutter-ohjelmistokehityspakettia. Sovelluksen tuli toimia pystyasennossa, Android- sekä iOS-älypuhelimilla ja sen tuli tukea mahdollisimman laajaa skaalaa erikokoisia laitteita. Sovelluksen tuli sisältää tuki kielikäännöksille, toteutettavan sovelluksen tuli sisältää käännökset kielille suomi ja

englanti. Kielikäännös tuli valita automaattisesti laitteen asetuksissa määritellyn kielen perusteella.

Sovelluksen käyttäjänhallinta ja autentikointi tuli toteuttaa käyttäen Amazon Cognito -palvelua ja käyttäjätilit tuli varmentaa käyttäen puhelinnumeroa. Sovelluksen tietokantana tuli käyttää Amazon DynamoDB -tietokantaa. Tiedonhaun ja lisäämisen rajapinta tuli toteuttaa käyttäen AWS AppSync -palvelua. Koko sovelluksen käyttämä Amazon Web Services -pilvipalvelun infrastruktuuri tuli toteuttaa käyttäen Terraform-työkalua. Mahdolliset AWS Lambda -funktiot tuli toteuttaa Serverless-ohjelmointikehystä käyttäen.

3.2 Käyttöliittymä

Sovelluksen käyttöliittymälle ei ollut asetettu erillisiä vaatimuksia tai rajoituksia, joten sen suunnittelu toteutettiin vastaamaan sovelluksen toiminnallisia vaatimuksia. Käyttöliittymän suunnitteluun ei käytetty suurta määrää resursseja, sillä tiedossa oli, että sovelluksen jatkokehitysvaiheessa käyttöliittymä suunniteltaisiin uudelleen vastaamaan sen hetkisiä sovelluksen toiminnallisuuksia.

Käyttöliittymän suunnittelussa käytettiin apuna Adobe XD-ohjelmistoa, jotta käyttöliittymäsuunnittelua voitaisiin jatkaa vaivattomasti jatkokehitysvaiheessa. Mikäli jokin osa tai komponentti käyttöliittymästä haluttaisiin säästää, olisi se helposti saatavilla ja muokattavissa. Kuvassa (kuva 13) on esitettynä toiminnallisten vaatimusten pohjalta sovellukselle suunnitellut käyttöliittymänäkymät.

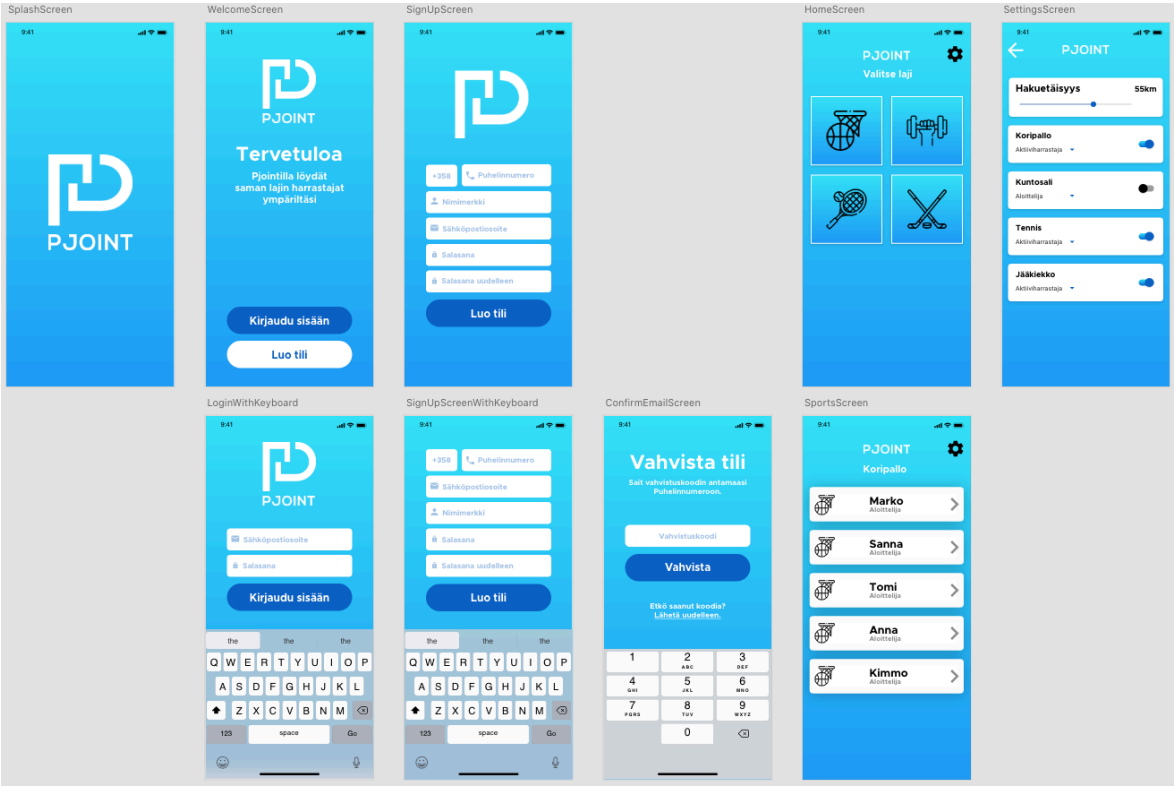
Sovelluksen avausnäkymässä (SplashScreen) haetaan puhelimen muistista sovelluksen käynnistymisen yhteydessä tarvittavia tietoja, sekä suoritetaan sovelluksen toiminnan kannalta tarvittavaa ohjelmakoodia, ennen käyttäjän siirtämistä varsinaiseen sovellukseen.

Sisäänkirjautumisnäkymässä (WelcomeScreen) käyttäjä voi kirjautua sisään sovellukseen tai valita luovansa uuden tilin sovellukseen. Sisäänkirjautumisnäkymästä suunniteltiin näkymä, jossa laitteen näppäimistö on näkyvissä (LoginWithKeyboard).

Käyttäjätilin luontinäkymässä (SignUpScreen) käyttäjä voi luoda uuden käyttäjätilin sovellukseen. Käyttäjätilin luontinäkymästä suunniteltiin myös variaatio käyttäjän puhelinnumeron vahvistusta varten, sekä variaatiot, joissa laitteen näppäimistö on näkyvillä.

Sovelluksen päänäkymässä (HomeScreen) käyttäjä voi valita urheilulajin, jonka harrastajia haluaa selata näkymässä. Käyttäjän valitessa päänäkymässä urheilulajin käyttäjä navigoidaan urheilulajilistaukseen (SportsScreen).

Asetusnäkymässä (SettingsScreen) käyttäjä voi asettaa itselleen haluamansa hakusäteen, sekä määritellä itselleen urheilulajit, joita harrastaa, sekä taitotason kullekin urheilulajille.



Kuva 13. Sovelluksen käyttöliittymänäkymät

Suunniteltuja käyttöliittymänäkymiä käytettiin suuntaviivana sovelluksen käyttöliittymän toteuttamiselle. Koska sovelluksen käyttöliittymälle ei ollut asetettu vaatimuksia tai rajoituksia, ei täydellistä vastaavuutta ollut tarvetta saavuttaa.

3.3 Tietokanta

Sovelluksen tietokantana käytettiin Amazon DynamoDB -tietokantaa. Koska DynamoDB on NoSQL-tyyppinen tietokanta, aloitettiin tietokannan suunnittelu määrittelemällä tietokantataulut ja sovelluksen toiminnallisuuksien vaatimat pääsymallit tietokannan taulujen sisältämiin tietoihin.

Tietokantasuunnittelussa päädyttiin kolmeen eri tietokantatauluun, jotka muodostavat omat erilliset kokonaisuutensa. Käyttäjien sijaintitiedot säilöttäisiin geo_data tietokantatauluun. Käyttäjän asetukset, eli harrastettavat urheilulajit ja hakusäde säilöttäisiin user_data tietokantatauluun. Käyttäjälle saatavilla olevat harrastettavat urheilulajit säilöttäisiin admin_data tietokantatauluun.

Käyttäjän sijaintitiedon hakemiseen ja tallentamiseen päätettiin käyttää Geo Library for Amazon -kirjastoa. Geo Library for Amazon -kirjasto sisältää valmiin rajapinnan sijaintipisteiden hakemiseen määritetyn sijaintipisteen ympäriltä. Kirjastossa oli myös suora tuki hakusäteen antamiselle, joten se todettiin sovelluksen käyttötarkoitusta vastaavaksi. (GitHub Inc.) Kirjasto edellytti luotavalta tietokannalta ennalta määriteltyä rakennetta, joten sen suunnittelussa ei ollut valinnanvaraa (Amazon Web Services Inc h).

Sovelluksen toiminnan kannalta oleellisia pääsymalleja olivat käyttäjän valitseman hakusäteen sisällä olevat muut käyttäjät, yksittäisen käyttäjän urheiluharrastukset, yksittäisen käyttäjän itselleen määrittelemä hakusäde, sekä sovelluksessa käytössä olevat harrastettavat urheilulajit. Edellä mainittujen pääsymallien mukaisesti admin_data sekä user_data tietokantatauluista laadittiin kuvassa (kuva 14) esitetyt tietokantamallit.

Table: admin_data GSI: typeIndex (PK: type)

Primary Key		Attributes
PK (id)	SK (type)	name
068-ef80-0534-4f99-8d72-dfb46480c59b	SPORT	ICEHOCKEY
069-ef80-0844-4f49-8d42-dfr46480c59b	SPORT	TENNIS

Table: user_data

Primary Key		Attributes				
PK (username)	SK (data)	type	skillLevel	searchRadius	name	active
068-ef80-0534-4f99-8d72-dfb46480c59b	SPORT#ICEHOCKEY	SPORT	beginner		ICEHOCKEY	true
068-ef80-0534-4f99-8d72-dfb46480c59b	SEARCHRADIUS	SEARCHRADIUS		20		

Kuva 14. DynamoDB-tietokantataulujen tietokantamallit

Laaditut tietokantamallit mahdollistivat sovelluksen vaatimat pääsymallit taulujen sisältämiin tietoihin. Sovelluksessa saatavilla olevat harrastukset sisältävälle admin_data tietokantataululle päätettiin luoda Global Secondary Index, jonka jaotteluavaimena toimii tiedon tyyppi (type). Tämä mahdollisti kaikkien saatavilla olevien harrastuksien hakemisen DynamoDB:n Query -operaatiolla.

Käyttäjän asetukset sisältävässä user_data tietokantataulussa jaotteluavaimena käytettiin käyttäjän käyttäjänimeä ja lajitteluavaimena tietoa kuvaavaa attribuuttia (data). Tietoa kuvaava attribuutti muodostettiin urheilulajien kohdalla siten, että tiedon tyyppi (type) yhdistettiin tiedon nimeen (name). Koska käyttäjällä oli ainoastaan yksi hakusäde, voitiin hakusäteen kohdalla data attribuutissa käyttää ainoastaan tiedon tyyppiä (type).

Kaikki käyttäjän tiedot voitiin hakea DynamoDB:n Query -operaatiolla käyttäen käyttäjän käyttäjänimeä. Kaikki käyttäjän harrastamat urheilulajit voitiin hakea DynamoDB:n Query -operaatiolla, jossa käytettiin käyttäjän käyttäjänimeä sekä data attribuuttiin kohdistettua ehdollista begins_with-ilmaisua. Kuvassa (kuva 15) on esitetty esimerkki kaikkien käyttäjän harrastamien urheilulajien hakemisesta tietokantataulusta.

```
{
  TableName: "user_data",
  KeyConditionExpression: "username = :username and begins_with(#data, :data)",
  ExpressionAttributeNames: {
    "#data" = "data"
  },
  ExpressionAttributeValues: {
    ":username" : "068-ef80-0534-4f99-8d72-dfb46480c59b",
    ":data" : "SPORT"
  }
}
```

Kuva 15. DynamoDB Query -koodi

Koska Geo Library for Amazon -kirjasto käyttää jaotteluavaimena tallennettavan sijainnin geohash-tietoa, ei tallennettua sijaintipistettä voida päivittää. Mikäli käyttäjän sijaintia haluttaisiin päivittää, pitäisi sijaintipiste poistaa ja luoda uusi sijaintipiste. Käyttäjän sijaintipisteen poistamiseksi tulisi tietää edellinen tallennettu sijainti. Tästä syystä tauluun luodaan sijaintipistettä tallennettaessa niin sanottu metatieto-item, joka sisältää luodun pisteen sijaintitiedon. Koska jaotteluavain on tyyppiä number, ei tauluun voida lisätä itemiä, jonka jaotteluavain olisi merkkijono, kuten käyttäjän käyttäjänimi.

Metatieto-item muodostetaan niin, että käyttäjän käyttäjänimestä, joka on uuid (universally unique identifier), poistetaan muut merkit kuin numerot. Metatieto-itemin lajitteluavaimena käytetään käyttäjän käyttäjänimeä. Metatieto-itemille lisätään lisäksi attribuutti, joka sisältää sijaintipisteen sijaintitiedon.

Kun käyttäjän sijaintitieto päivitetään, haetaan tietokannasta käyttäjän käyttäjänimen avulla metatieto-item, joka sisältää käyttäjän vanhan sijaintipisteen tiedot. Vanhan sijaintipisteen tiedon avulla voidaan poistaa käyttäjän edellinen sijaintipiste ja luoda käyttäjälle uusi sijaintipiste.

Kuvassa (kuva 16) on esitetty sijaintitiedon sisältävän geo_data taulun tietokantamalli, joka vastaa käytettävän Geo Library for Amazon -kirjaston edellyttämää tietokantamallia. Kuvassa ylemmällä rivillä on esitetty Geo Library for Amazon -kirjaston luoma sijaintipiste-item ja alemmalla rivillä tietokantaan tallennettava metatieto-item.

Primary Key		Attributes		
PK (hashKey)	SK (username)	geoJson	geohash	metaHash
-86885	068-ef80-0534-4f99-8d72-dfb46480c59b	{"type" : "POINT", "coordinat	-8688591364790179341	
688005344998724000000	068-ef80-0534-4f99-8d72-dfb46480c59b			-86885

Kuva 16. DynamoDB-tietokantataulun tietokantamalli

3.4 AppSync API

Sovelluksen pääsy tietokannan tietoihin tuli toteuttaa käyttäen AWS AppSync -palvelua, joka luo GraphQL-viestintäprotokollaa käyttävän rajapinnan Amazon Web Services -pilvipalveluun. Kaikki sovelluksen eri tietokantatauluista hakema ja niihin lisäämä tieto suunniteltiin kulkemaan yhden keskitetyn rajapinnan kautta.

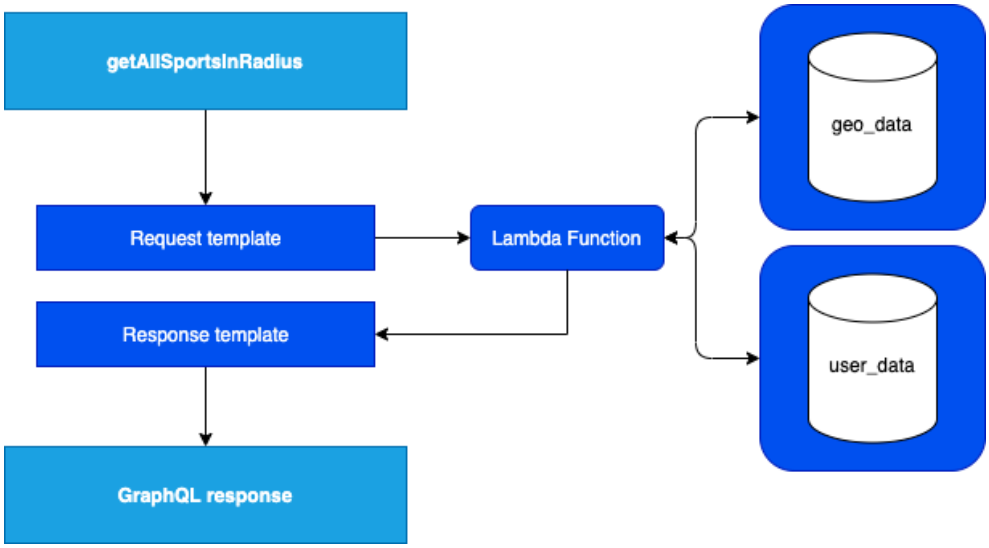
Rajapinnasta luotiin suunnitteluvaiheessa ylätason kuvaus, joka sisältää tarvittavat tiedon haku (Query) ja tiedon lisäys (Mutation) tyypit. Tyypit pyrittiin nimeämään mahdollisimman kuvaavasti niin, että suoritettavan GraphQL-kyselyn nimestä voitaisiin päätellä sen suorittama operaatio. Kuvassa (kuva 17) on esitetty rajapinnalle suunnitteluvaiheessa luotu ylätason kuvaus, joka sisältää sovelluksen tarvitsemat Query- ja Mutation tyypit.

OPERATION TYPE	OPERATION NAME	USE CASE
Query	getAllSportsInRadius	Get all active sport items that are inside the users search radius.
Query	getSettingsForCurrentUser	Get settings for current user.
Mutation	updateUserSettings	Update items settings value based on type.
Mutation	updateUserLocation	Update location data for current user.

Kuva 17. GraphQL-rajapinnan ylätason kuvaus

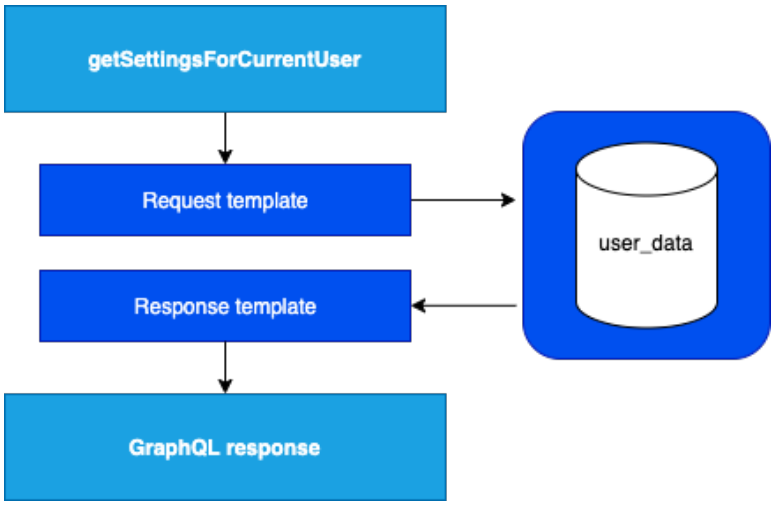
Rajapinnan ylätason suunnittelun jälkeen rajapinnalle suunniteltiin ylätason kuvauksessa kuvattujen Query- ja Mutation tyyppien Resolvereita kuvaavat prosessikaaviot. Kuvassa (kuva 18) on esitettynä getAllSportsInRadius Query:n Unit Resolverin suoritusta kuvaava prosessikaavio. Resolverin AWS Lambda-funktio hakee

käyttäjän hakusäteen sisällä olevat käyttäjät geo_data tietokantataulusta. Käyttäjien haun jälkeen haetaan user_data tietokantataulusta hakusäteen sisällä olevien käyttäjien itemit, jotka ovat tyyppiä SPORT ja joiden active attribuutin arvo on true. Koska hakusäteen sisällä olevien käyttäjien hakeminen vaatii Amazon Geo Library for DynamoDB -kirjaston käyttöä, jouduttiin Unit Resolverissa käyttämään tietolähteenä AWS Lambda -funktiota, joka hakee tietoa geo_data sekä user_data tietokantatauluista.



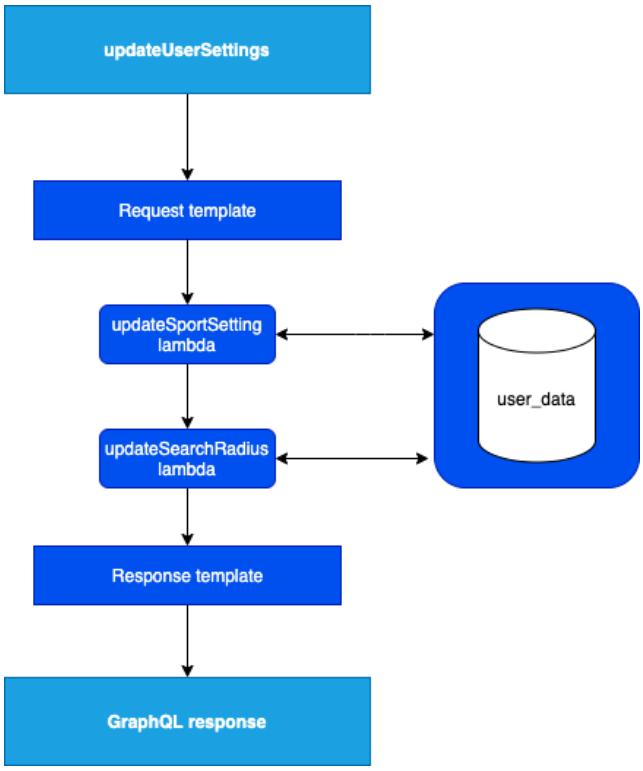
Kuva 18. AppSync Unit Resolver suoritusprosessikaavio

Kuvassa (kuva 19) on esitettynä getSettingsForCurrentUser Query:n Unit Resolverin suoritusta kuvaava prosessikaavio. Käyttäjän asetukset voitiin hakea käyttäen AppSyncin tukemaa Apache Velocity Template Language -kielellä kirjoitettua funktiota, joka noutaa tiedon user_data taulusta.



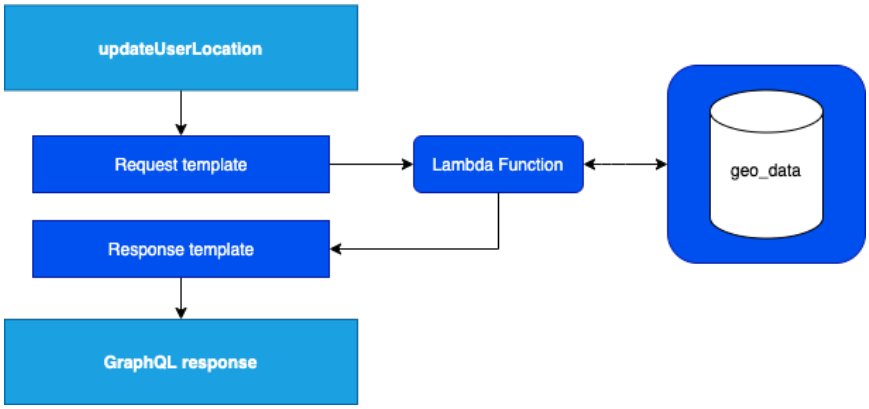
Kuva 19. AppSync Unit Resolver suoritusprosessikaavio

Kuvassa (kuva 20) on esitettynä updateUserSettings Mutationin Pipeline Resolverin suoritusta kuvaava prosessikaavio. Käyttäjän asetuksia voitiin päivittää käyttäen AppSyncin tukemaa Apache Velocity Template Language -kielellä kirjoitettua funktiota, joka päivittää tiedon user_data tauluun. Resolver määriteltiin Pipeline tyyppiseksi resolveriksi, joka sisältää erillisen päivitysfunktion urheilulajin asetuksien päivittämiseksi, sekä käyttäjän hakusäteen päivittämiselle. Pipeline Resolver suorittaa joko urheilulajien päivitysfunktion, tai hakusäteen päivitysfunktion perustuen input parametrina annetun asetuksen type ominaisuuden arvon perusteella.



Kuva 20. AppSync Pipeline Resolver suoritusprosessikaavio

Kuvassa (kuva 21) on esitettynä updateUserLocation Mutationin Unit Resolverin suoritusta kuvaava prosessikaavio. Koska käyttäjän sijaintitiedon lisääminen vaatii Amazon Geo Library for DynamoDB -kirjastoa käyttöä, jouduttiin Unit Resolverissa käyttämään tietolähteenä AWS Lambda -funktiota, joka poistaa käyttäjän edellisen sijaintitiedon geo_data taulusta ja lisää käyttäjän uuden sijainnin geo_data tauluun.



Kuva 21. AppSync Unit Resolver suoritusprosessikaavio

4 Toteutus

4.1 Responsiivisuus

Sovelluksen toteutuksessa tuli ottaa huomioon sen tarve toimia useilla eri kokoisilla laitteilla. Toteutuksessa päädyttiin mittaamaan laitteen näytön mitat sovelluksen käynnistyksen yhteydessä näytettävässä SplashScreen näkymässä. Ratkaisuun päädyttiin, sillä laitteen mitat eivät muutu sovelluksen käytön aikana, joten oli perusteltua mitata laitteen näytön mitat ainoastaan kerran ja käyttää näitä mittoja sovelluksen muilla näytöillä.

Sovelluksen mittojen käytössä päädyttiin luomaan SizeConfig-luokka, jonka olio sisältää laitteen näytön mitat. Mittojen käyttöä helpottamaan laitteen näytöltä laskettiin näkyvä osa eli alue, joka ei peity laitteen käyttöjärjestelmän omien komponenttien, kuten akun varaustason ja operaattorin nimen osoittavan yläpalkin alle. Näkyvä osa jaettiin leveys- ja pituussunnassa 100 osaan, jotta mittoja voitaisiin käyttää prosenttimuodossa.

Luokalle määriteltiin init-funktio, joka suorittaa laitteen näytön mittaamisen käyttäen Flutter-rajapinnan MediaQuery.of(context)-funktiota, joka palauttaa tiedot laitteen asennosta ja mitoista (Google h). SizeConfig-luokan init-funktiota kutsuttiin sovelluksen aloitusnäkymän (SplashScreen) build-funktiossa. Näin toimimalla sovelluksen muissa näkyvissä, voitiin käyttää laitteen näytön mittoja muodossa SizeConfig.safeBlockHorizontal * x käyttöliittymän Widget-elementtien kokoja määritettäessä.

4.2 Tuki kielikäännöksille

Sovellukselle tuli toteuttaa tuki kielikäännöksille ja sovelluksen kielen tuli vastata laitteen käyttöjärjestelmän asetuksista valittua kieltä. Sovellukselle päädyttiin luomaan AppLocalizations-luokka, joka lataa laitteen asetuksista valitun kielen mukaisen JSON-tiedoston (JavaScript Object Notation), joka sisältää sovelluksen merkkijonot valitulla kielellä. AppLocalization-luokalle määriteltiin trans-funktio, jolle annetaan input parametrina käännettävän merkkijonon JSON-tiedoston avainta vastaava merkkijono ja funktio palauttaa käyttöliittymässä näytettävän merkkijonon. Tiedostomuodoksi valittiin JSON sen selkeän rakenteen vuoksi.

AppLocalizations-luokan lisäksi sovellukselle luotiin AppLocalizationsDelegate-luokka, joka perii Flutterin LocalizationsDelegate-luokan. AppLocalizationsDelegate-luokka vastaa laitteen asetuksista valitun kielen hakemisesta, sekä käännettyjen resurssien toimittamisesta sovellukselle (Google i).

Sovelluksen käyttämät kieliresurssit määritetään sovelluksen päätasolla MaterialApp Widgetin localizationsDelegates-ominaisuudessa. Kuvassa (kuva 22) on esitetty esimerkki Flutter-sovelluksen päätason MaterialApp Widgetistä, jolle on määritelty MaterialDesign Widgettien kielikäännökset, Cupertino Widgettien kielikäännökset sekä yleinen Widgettien paikallisuustietodelegaatti, joka määrittelee laitteen tekstin suunnan joko vasemmalta-oikealle tai oikealta-vasemmalle.

```
import 'package:flutter_localizations/flutter_localizations.dart';

MaterialApp(
 localizationsDelegates: [
   // ... app-specific localization delegate[s] here
   GlobalMaterialLocalizations.delegate,
   GlobalWidgetsLocalizations.delegate,
   GlobalCupertinoLocalizations.delegate,
 ],
 supportedLocales: [
    const Locale('en', ''), // English, no country code
    const Locale('he', ''), // Hebrew, no country code
    const Locale.fromSubtags(languageCode: 'zh'), // Chinese *See Advanced Locales below*
    // ... other locales the app supports
  ],
  // ...
)
```

Kuva 22. Flutter sovelluksen päätason MaterialApp Widget (Google i)

4.3 Käyttöliittymä

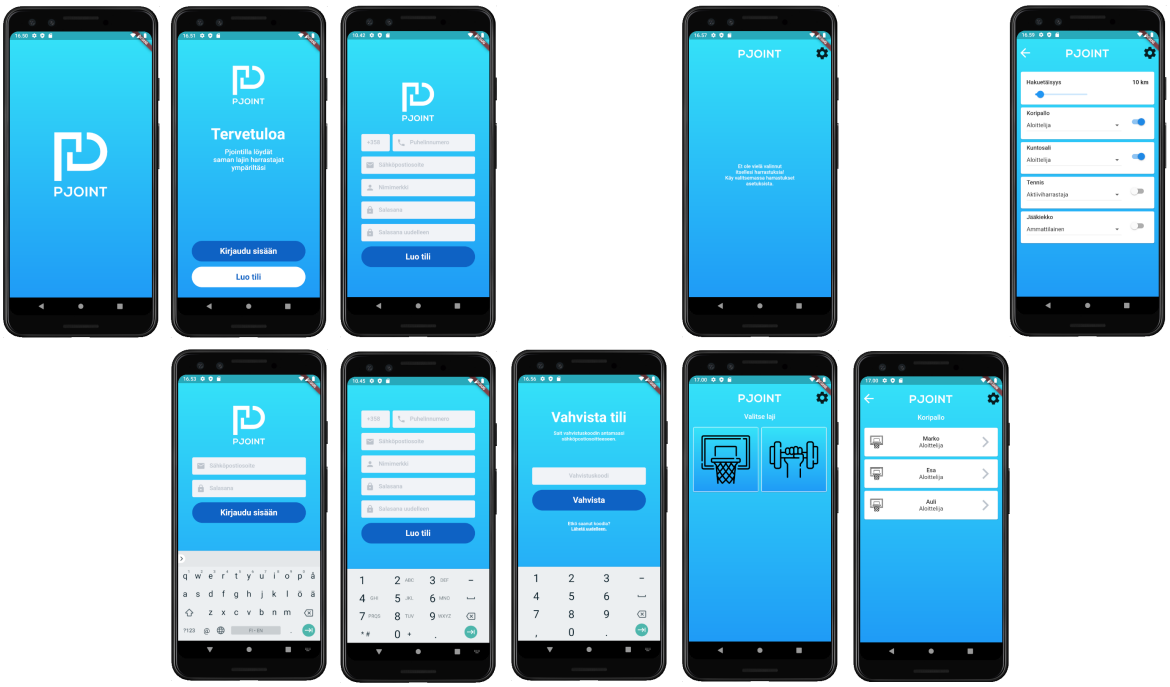
Sovelluksen käyttöliittymän kehitys toteutettiin mahdollisimman pitkälle ennen varsinaisten toiminnallisuuksien kehitystä. Näin toimimalla varmistuttiin siitä, että sovelluksen tarvitsemat tiedot vastaavat aiemmin suunnitteluvaiheessa määriteltyjä tarpeita. Mikäli käyttöliittymää kehitettäessä olisi huomattu tarpeiden olevan ristiriidassa suunnitelman kanssa, olisi taustajärjestelmän suunnitteluun voitu palata ja toteuttaa tarvittavat muutokset, ennen kuin kehitykseen olisi käytetty aikaa.

Käyttöliittymää toteutettaessa, käytettiin apuna kovakoodattuja listoja, jotka simuloivat käyttäjän tietokannasta hakemaa listausta samojen urheilulajien harrastajista ja listausta käyttäjän omista asetuksista. Koska käytössä oli kuvaus tietokannan sisältämistä tiedoista, voitiin kovakoodatut listat toteuttaa vastaamaan tietokannasta myöhemmin saatavissa olevien listausten rakennetta. Tämä mahdollisti käyttöliittymän toteuttamisen lähes lopulliseen muotoonsa, ennen taustajärjestelmän kehittämistä.

Käyttöliittymän kehityksen aikana, todettiin sen sisältävän yhden toistuvan elementin, joka oli yleiskäyttöinen nappi, jota käytettiin sisäänkirjautumisnäkymässä ja käyttäjätilin luontinäkymässä. Kyseisen käyttöliittymäelementin kohdalla oli perusteltua luoda elementistä erillinen Widget-elementti, jolle voitiin määritellä väri, koko, napin teksti sekä funktio, joka suoritetaan nappia painettaessa.

Myös käyttöliittymässä käytettävistä väreistä ja fonttivariaatioista luotiin erillinen tiedosto, joka sisälsi luokan, jolle määriteltiin ominaisuuksiksi käyttöliittymässä käytettävät värit ja fonttivariaatiot. Luokka mahdollisti värien ja fonttivariaatioiden käyttämisen selkokielisinä ilmaisuina, kuten PjointTheme.lightBlue. Koska värejä ja fonttivariaatioita käytettiin luokan ominaisuuksien kautta, olisi niiden muuttaminen koko sovelluksessa mahdollista yhden keskitetyn tiedoston avulla.

Kuvassa (kuva 23) on esitetty sovellukselle kehitetyt käyttöliittymänäkymät. Kehitetyn käyttöliittymän ja käyttöliittymäsuunnitelman välille, ei syntynyt mainittavia poikkeavuuksia.



Kuva 23. Sovelluksen käyttöliittymänäkymät

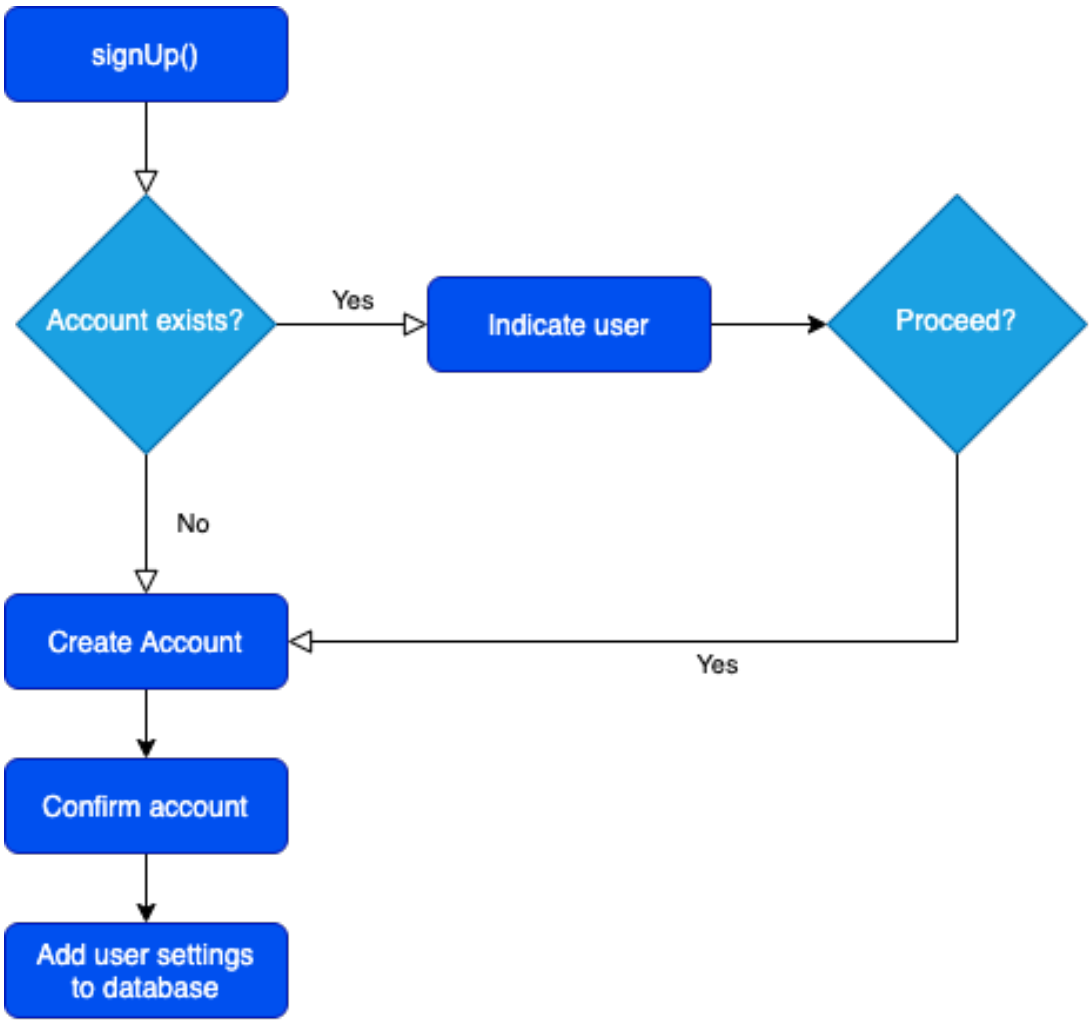
4.4 Käyttäjätilin luominen ja kirjautuminen sovellukseen

Sovelluksen käyttäjänhallinta toteutettiin Amazon Cognito -palvelua käyttäen. Ennen sovelluksen käyttäjätilin luontitoiminnon ja sisäänkirjautumistoiminnon kehittämistä, oli Amazon Web Services -pilvipalveluun luotava Amazon Cognito User Pool -resurssi, johon käyttäjät lisättiin.

Amazon Cognito User Pool -resurssin lisäksi sisäänkirjautumista varten täytyi luoda DynamoDB-tietokantataulut user_data sekä admin_data ja kaksi AWS Lambda -funktiota preSignUp ja postConfirmation.

Kyseiset resurssit tarvittiin käyttäjätilinluontiprosessin kehittämiseksi niin, että käyttäjän luodessa uuden käyttäjätilin, Amazon Cognito User Pool suorittaa automaattisesti preSignUp-lambdafunktion, joka tarkistaa onko käyttäjän antamalla puhelinnumerolla jo olemassa vahvistettu käyttäjätili. Mikäli puhelinnumerolla löytyy vahvistettu käyttäjätili, indikoidaan tieto käyttäjälle ja kysytään yrittääkö käyttäjä kirjautua sisään. Mikäli käyttäjä klikkaa uudelleen ”Luo tili” -painiketta, ohitetaan preSignUp-lambdafunktion suorittama tarkistus ja palveluun luodaan uusi käyttäjätili samalla puhelinnumerolla. Tämän jälkeen käyttäjä ohjataan puhelinnumeron

vahvistussivulle, jossa käyttäjä antaa tekstiviestinä saamansa vahvistuskoodin ja vahvistaa käyttäjätilin puhelinnumeron. Puhelinnumeron vahvistuksen yhteydessä mahdollinen aiempi samalla puhelinnumerolla luotu aktiivinen tili muuttuu automaattisesti vahvistamattomaksi. Tilin vahvistuksen jälkeen Amazon Cognito User Pool suorittaa automaattisesti postConfirmation-lambdafunktion, joka hakee admin_data tietokantataulusta käytössä olevat urheilulajit ja lisää tämän jälkeen user_data tietokantatauluun saatavissa olevat urheilulajit sekä hakusäteen. Tietojen lisäyksessä käytetään username attribuuttina käyttäjän käyttäjänimeä, urheilulajien taitotasoksi merkitään oletusarvoisesti taso ”beginner”, urheilulajien active attribuutille annetaan arvoksi false, urheilulajien owner attribuutin arvoksi annetaan käyttäjän nimimerkki ja hakusäteen searchRadius attribuutille annetaan oletusarvo 50. Kuvassa (kuva 24) on esitetty käyttäjätilin luontiprosessi vuokaaviona.



Kuva 24. Käyttäjätilin luonti

Amazon Cognito User Pool -resurssi sekä DynamoDB-tietokantataulut luotiin määrittelemällä resurssit Terraformissa. Koska käyttäjätili vahvistettiin käyttäen tekstiviestiä, luotiin Terraformilla myös tekstiviestin lähettämisen vaatimat resurssit. Näitä olivat Amazon IAM -rooli, joka sallii Amazon Cognito User Poolin lähettää tekstiviestejä, sekä Amazon SNS -konfiguraatio (Simple Notification Service), joka määrittelee tekstiviestin oletuslähettäjän, sekä kuukausikohtaisen budjetin tekstiviestipalvelulle. Käyttäjätilin luonnin vaatimat AWS Lambda -funktiot luotiin käyttämällä Serverless-ohjelmointikehystä ja TypeScript-ohjelmointikieltä.

Sovelluksen sisäänkirjautumis -ja käyttäjätilin luonti -toiminnot toteutettiin käyttämällä Amazon Cognito Identity SDK for Dart -kirjastoa, joka sisältää valmiit rajapinnat Amazon Cognito -palvelun käyttämiseen Dart-ohjelmointikielellä (Google 2019).

4.5 Istunnonhallinta

Sovelluksen käyttäjäistunnon tuli pysyä aktiivisena, vaikka sovellus suljettaisiin käyttökertojen välillä. Käyttäjän kirjautuessa sisään sovellukseen palauttaa Amazon Cognito User Pool -palvelu käyttäjälle CognitoUserSession-luokan olion, joka sisältää käyttäjän pääsytunnuksen (CognitoAccessToken), jonka avulla käyttäjä voi tunnistautua Amazon Web Services -pilvipalvelun resursseihin. Käyttäjän pääsytunnus vanhenee oletusarvoisesti 60 minuutin kuluessa sen luomisesta. CognitoUserSession-luokan olio sisältää myös pidempikestoisen päivitystunnuksen (CognitoRefreshToken), jonka avulla käyttäjän pääsytunnus voidaan uusia. Päivitystunnus vanhenee oletusarvoisesti 30 päivän kuluessa sen luomisesta.

Käyttäjän pääsytunnuksen uusiminen toteutettiin funktiolla, joka suoritetaan aina, kun sovellus käyttää käyttäjän pääsytunnusta. Pääsytunnuksen uusimisessa käytettiin hyödyksi Amazon Cognito Identity for Dart -kirjastoa, joka sisältää valmiin rajapinnan pääsytunnuksen uusimiseen päivitystunnuksen avulla.

Käyttäjän käyttäjänimi ja päivitystunnus tallennettiin laitteen muistiin uuden käyttäjätilin luonnin, sekä sisäänkirjautumisen yhteydessä. Tietojen tallentamiseen käytettiin Shared preferences plugin -kirjastoa, joka mahdollistaa yksinkertaisten avain-arvoparien tallentamisen laitteen muistiin ja tiedon hakemisen laitteen muistista (Google 2020a).

Laitteen muistiin tallennettuja tietoja haettiin sovelluksen käynnistysnäkymässä, ja mikäli laitteen muistista löytyi tarvittavat tiedot, yritettiin käyttäjäistunnon päivitystä. Mikäli käyttäjäistunnon päivitys onnistui, ohjattiin käyttäjä suoraan sovelluksen päänäkymään (HomeScreen). Mikäli tarvittavia tietoja ei löytynyt laitteen muistista, tai käyttäjäistunnon päivitys epäonnistui, ohjattiin käyttäjä sovelluksen sisäänkirjautumisnäkymään.

4.6 Käyttäjän asetusten hakeminen tietokannasta ja tietojen muuttaminen

Sovelluksen tarvitsemien tietojen hakemiseen tietokannasta ja tietojen muuttamiseen ja lisäämiseen, tuli käyttää Amazon AppSync -palveluun toteutettua GraphQL-rajapintaa. AppSync-rajapinta toteutettiin määrittelemällä Terraformissa rajapinnan kuvaus (schema), rajapinnan resolverit, resolvereiden käyttämät funktiot, sekä niiden tarvitsemat AWS IAM -roolit. Rajapinta toteutettiin suunnitteluvaiheessa tehdyn suunnitelman mukaisesti, mutta sijaintitietoon liittyvät resolverit, sekä niiden käyttämät funktiot toteutettiin kehityksen myöhemmässä vaiheessa.

Mobiilisovelluksen GraphQL-kyselyt toteutettiin käyttäen GraphQL Client -kirjastoa, joka tarjoaa suoraviivaisen rajapinnan GraphQL-clientin luomiseen ja GraphQL-kyselyjen tekemiseen luodun clientin avulla (Google 2020b).

Käyttäjän asetusten hakeminen tietokannasta suoritettiin aina kun sovelluksen päänäkymä (HomeScreen) avataan. Käyttäjän asetusten tallentaminen tietokantaan suoritettiin, kun käyttäjä muuttaa asetuksiaan asetusnäkymässä (SettingsScreen). Hakusäteen muuttamisen yhteydessä tallennus tietokantaan suoritettiin ainoastaan, kun käyttäjä päästää irti hakusäteen liukusäätimestä. Näin toimimalla vähennettiin tietokantaan tehtävien tallennusten määrää, sillä arvoa ei tallenneta aina kun liukusäätimen arvo muuttuu.

4.7 Käyttäjän sijaintitiedon tallentaminen tietokantaan

Käyttäjän sijaintitiedon tallentamisessa tietokantaan käytettiin AWS Lambda -funktiota, joka ensin hakee tietokannasta metatieto itemin, joka sisältää käyttäjän edellisen sijaintipisteen. Mikäli tietokannasta ei löydy käyttäjän käyttäjänimeä vastaavaa metatieto itemiä, käyttäjälle luodaan uusi sijaintipiste sekä sitä vastaava metatieto item. Mikäli tietokannasta löytyy käyttäjän käyttäjänimeä vastaava metatieto item, poistetaan sen sisältämän sijaintitiedon avulla käyttäjän aiempi sijaintitieto. Vanhan

sijaintipisteen poistamisen jälkeen tietokantaan tallennetaan käyttäjän nykyinen sijaintipiste ja sitä vastaava metatieto item päivitetään vastaamaan uutta sijaintipistettä.

Sijaintia päivittävä AWS Lambda -funktio määriteltiin tietolähteeksi Amazon AppSync GraphQL -rajapinnan updateUserLocation resolverille. Lambda funktion ja GraphQL Resolverin lisäksi DynamoDB-tietokantaan määriteltiin Terraformissa geo_data tietokantataulu, joka vastasi Amazon Geo Library for DynamoDB -kirjaston edellyttämää tietokantamallia ja AppSyncin AWS IAM -roolille annettiin tarvittavat oikeudet AWS Lambda -funktion kutsumiseen.

GraphQL Resolverin input parametreiksi määritettiin sijainti koordinaatteina, jotka välitettiin suoritettavalle AWS Lambda -funktiolle. Koordinaattien lisäksi funktiolle välitettiin GraphQL-kyselyn suorittavan käyttäjän käyttäjänimi.

Laitteen sijainnin hakemisessa käytettiin Flutter Location plugin -kirjastoa, joka sisältää valmiin rajapinnan laitteen sijainnin hakemiseen ja tarvittavien käyttöoikeuksien kysymiseen käyttäjältä (Google 2020c). Laitteen sijaintitieto haettiin ja lähetettiin tietokantaan aina käyttäjän kirjautuessa sisään, kun käyttäjä käynnisti sovelluksen sisäänkirjautuneena tai kun käyttäjä oli luonut uuden käyttäjätilin. Käyttäjän sijainnin muuttumista myös jäätiin kuuntelemaan sovelluksen ollessa käynnissä. Mikäli käyttäjän sijainti muuttui, käyttäjän uusi sijainti tallennettiin tietokantaan. Sijainnin muuttumisen tarkastelussa käytettiin 10 minuutin intervallia, sekä 10 metrin tarkkuutta.

4.8 Hakusäteen sisällä olevien urheilulajien hakeminen tietokannasta

Käyttäjän hakusäteen sisällä olevien urheilulajien hakemisessa käytettiin AWS Lambda -funktiota, joka hakee Amazon Geo Library for DynamoDB -kirjaston avulla geo_data tietokantataulusta kaikki käyttäjät, jotka ovat annetun hakusäteen sisällä käyttäjän sijainnista. AWS Lambda -funktion input parametreinä annetaan käyttäjän käyttäjänimi, käyttäjän sijainti sekä käyttäjän määrittelemä hakusäde.

Hakusäteen sisällä olevien käyttäjien haun jälkeen hakutuloksesta suodatetaan pois hakeva käyttäjä. Mikäli hakusäteen sisältä ei löydy käyttäjiä funktio palauttaa arvon null ja funktion suoritus lopetetaan. Mikäli hakusäteen sisältä löytyy käyttäjiä, funktio

hakee user_data tietokantataulusta käyttäjien urheilulajit, joiden attribuutin active arvo on true. Funktio palauttaa listan aktiivista urheilulajeista.

Käyttäjien urheilulajit tietokannasta hakeva AWS Lamba -funktio määriteltiin tietolähteeksi Amazon AppSync GraphQL -rajapinnan getAllSportsInRadius resolverille. Resolverin input parametreiksi määritettiin AWS Lambda -funktiolle välitettävä hakusäde sekä käyttäjän sijainti koordinaatteina. Input parametrien lisäksi funktiolle välitettiin GraphQL-kyselyn suorittavan käyttäjän käyttäjänimi.

Mobiilisovelluksessa GraphQL-kysely suoritettiin aina kun käyttäjän sijainti haettiin tai sijainti muuttui sekä käyttäjän palatessa asetusnäkymästä päänäkymään. Näin toimimalla listaus voitiin päivittää vastaamaan käyttäjän mahdollisesti muuttamaa hakusädettä. Tällä ratkaisulla estettiin tilanne, missä käyttäjän hakusäteen toistuva muuttaminen asetussivulta poistumatta, aiheuttaisi toistuvien tietokantakyselyjen tekemisen. Urheilulajien lajittelu suoritettiin suodattamalla haettu aktiivisten urheilulajien lista käyttäjän aloitusnäkymässä valitseman urheilulajin perusteella.

5 Yhteenveto

Opinnäytetyössä suunniteltiin ja kehitettiin monialustainen sijaintitietoa hyödyntävä mobiilisovellus sekä sen käyttämä infrastruktuuri Amazon Web Services pilvipalveluun. Sovelluksen tuli toimia IOS -ja Android käyttöjärjestelmää käyttävillä älypuhelimilla. Sovelluksen toiminnalliset -ja tekniset vaatimukset oli asetettu tilaajayrityksen taholta.

Sovelluksen kehitys aloitettiin suunnittelemalla sovelluksessa olevat käyttöliittymänäkymät, tietokantamallit sekä Amazon AppSync GraphQL -rajapinnan toiminnot asetettujen toiminnallisten vaatimusten pohjalta. Suunnitteluvaiheessa päätettiin sijaintitietoon perustuvan käyttäjähaun tekninen toteutus, joka ohjasi osittain tietokantarakenteen suunnittelua.

Sovelluksen toteutus aloitettiin käyttöliittymän kehityksestä, jossa pyrittiin toteuttamaan suunniteltua käyttöliittymää vastaava toteutus. Käyttöliittymä kehitettiin lähes lopulliseen muotoonsa ennen taustajärjestelmän infrastruktuurin kehitystä. Käyttöliittymän kehityksen jälkeen toteutettiin sovelluksen vaatima taustajärjestelmän infrastruktuuri käyttäen IaC-menetelmää, jossa infrastruktuuri kuvattiin koodina, joka oli provisioitavissa Amazon Web Services -pilvipalveluun.

Opinnäytetyön lopputuloksena syntyi jatkokehityskelpoinen, projektille asetetut vaatimukset täyttävä monialustainen mobiilisovellus. Kehitetyn sovelluksen tekninen toteutus mahdollistaa sen skaalautuvuuden ja vaivattoman jatkokehityksen, joka oli projektille asetettu tavoite.

Lähteet

Amazon Web Services Inc. a. Amazon DynamoDB [viitattu: 24.9.2020]. Saatavissa: https://aws.amazon.com/dynamodb/

Amazon Web Services Inc. b. Supported Data Types – Amazon DynamoDB [viitattu: 24.9.2020]. Saatavissa: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DynamoDBMapper.DataTypes.html

Amazon Web Services Inc. c. Working with Queries in DynamoDB [viitattu: 25.9.2020]. Saatavissa: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Query.html

Amazon Web Services Inc. d. Improving Data Access with Secondary Indexes [viitattu: 24.9.2020]. Saatavissa: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SecondaryIndexes.html

Amazon Web Services Inc. e. What´s the difference between Amazon Cognito user pools and identity pools? [viitattu: 25.9.2020]. Saatavissa: https://aws.amazon.com/premiumsupport/knowledge-center/cognito-user-pools-identity-pools/

Amazon Web Services Inc. f. What Is Amazon Cognito [viitattu: 25.9.2020]. Saatavissa: https://docs.aws.amazon.com/cognito/latest/developerguide/what-is-amazon-cognito.html

Amazon Web Services Inc. g. Resolver Mapping Template Overview [viitattu: 1.10.2020]. Saatavissa: https://docs.aws.amazon.com/appsync/latest/devguide/resolver-mapping-template-reference-overview.html

Amazon Web Services Inc, h. Geo Library for Amazon DynamoDB – Part1: Table Structure [viitattu: 8.10.2020]. Saatavissa:

https://aws.amazon.com/blogs/mobile/geo-library-for-amazon-dynamodb-part-1-table-structure/

Blogi: Balasubramanian, G. 2017. Choosing the Right DynamoDB Partition Key [viitattu: 27.9.2020]. Amazon Web Services Inc. Saatavissa: https://aws.amazon.com/blogs/database/choosing-the-right-dynamodb-partition-key/

Balbaert, I. & Ridjanovic, D. 2013. Learning Dart. Birmingham, United Kingdom: Packt Publishing Ltd.

DeBrie, A. a. Anatomy of an Item [viitattu: 24.9.2020]. Saatavissa: https://www.dynamodbguide.com/anatomy-of-an-item/

DeBrie, A. b. Secondary Indexes [viitattu: 24.9.2020]. Saatavissa: https://www.dynamodbguide.com/secondary-indexes

Blogi: Devathon. 2019. GraphQL vs REST in 2020: A Detailed Comparison [viitattu: 23.9.2020]. Saatavissa: https://devathon.com/blog/graphql-vs-or-rest/

Blogi: Feoktistov, I. Top 7 Flutter Advantages and Why You Should Try Flutter on Your Next Project [viitattu: 21.9.2020]. Relevant Software LP. Saatavissa: https://relevant.software/blog/top-8-flutter-advantages-and-why-you-should-try-flutter-on-your-next-project/

Artikkeli: Gaël, T. 2020. What is Flutter and Why You Should Learn it in 2020 [viitattu: 21.9.2020]. Free Code camp, Inc. Saatavissa: https://www.freecodecamp.org/news/what-is-flutter-and-why-you-should-learn-it-in-2020/

GitHub Inc. Geo Library for Amazon DynamoDB [viitattu: 8.10.2020]. Saatavissa: https://github.com/rh389/dynamodb-geo.js

Google. a. FAQ [viitattu: 21.9.2020]. Saatavissa: https://flutter.dev/docs/resources/faq

Google. b. Flutter SDK Releases [viitattu 21.9.2020]. Saatavissa: https://flutter.dev/docs/development/tools/sdk/releases?tab=windows

Google. c. Introduction to widgets [viitattu: 21.9.2020]. Saatavissa: https://flutter.dev/docs/development/ui/widgets-intro

Google. d. Widget Catalog [viitattu: 21.9.2020]. Saatavissa: https://flutter.dev/docs/development/ui/widgets

Google. e. Platforms [viitattu: 27.9.2020]: Saatavissa: https://dart.dev/platforms

Google. f. Get the Dart SDK [viitattu: 1.10.2020]. Saatavissa: https://dart.dev/get-dart

Google. g. Flutter performance profiling [viitattu 27.9.2020]. Saatavissa: https://flutter.dev/docs/perf/rendering/ui-performance

Google. h. MediaQuery Class [viitattu: 9.10.2020]. Saatavissa: https://api.flutter.dev/flutter/widgets/MediaQuery-class.html

Google. i. Internationalizing Flutter apps [viitattu: 10.10.2020]. Saatavissa: https://flutter.dev/docs/development/accessibility-and-localization/internationalization

Google. 2019. Amazon Cognito Identity SDK for Dart [viitattu: 16.10.2020]. Saatavissa: https://pub.dev/packages/amazon_cognito_identity_dart

Google. 2020a. Shared preferences plugin [viitattu: 16.10.2020]. Saatavissa: https://pub.dev/packages/shared_preferences

Google. 2020b. GraphQL Client [viitattu: 17.10.2020]. Saatavissa: https://pub.dev/packages/graphql

Google. 2020c. Flutter Location Plugin [viitattu: 20.10.2020]. Saatavissa: https://pub.dev/packages/location

HashiCorp. a. Download Terraform [viitattu: 28.9.2020]. Saatavissa: https://www.terraform.io/downloads.html

HashiCorp. b. Introduction to Terraform [viitattu: 28.9.2020]. Saatavissa: https://www.terraform.io/intro/index.html

HashiCorp. c. Providers [viitattu: 28.9.2020]. Saatavissa: https://www.terraform.io/docs/providers/index.html

HashiCorp. d. Configuration Syntax [viitattu: 28.9.2020]. Saatavissa: https://www.terraform.io/docs/configuration/syntax.html

HashiCorp. e. Perform CRUD operations with Providers [viitattu: 28.9.2020]. Saatavissa: https://learn.hashicorp.com/tutorials/terraform/provider-use

IBM Cloud Education. 2020. Terraform [viitattu: 28.9.2020]. IBM. Saatavissa: https://www.ibm.com/cloud/learn/terraform

Mistry, A. 2018. Expert AWS Development, United Kingdom: Packt Publishing Ltd.

Pikalainavertailu. Pikavippi [viitattu: 26.9.2020]. Saatavissa: https://www.pikalainavertailu.info/pikavippi/

Serverless Inc. a. AWS AppSync – The Ultimate Guide [viitattu: 26.9.2020]. Saatavissa: https://www.serverless.com/aws-appsync

Serverless Inc. b. Serverless Framework Open Source [viitattu: 9.10.2020]. Saatavissa: https://www.serverless.com/open-source/

Serverless Inc. c. AWS Introduction [viitattu: 9.10.2020]. Saatavissa: https://www.serverless.com/framework/docs/providers/aws/guide/intro/

Sheiks, R. 2018. Infrastructure-as-Code with Terraform [28.9.2020]. Groupware Technology. Saatavissa: https://www.groupwaretech.com/infrastructure-as-code-terraform/

The GraphQL Foundation. a. Queries and Mutations [viitattu: 23.9.2020]. Saatavissa: https://graphql.org/learn/queries/

The GraphQL Foundation. b. Schemas and Types [viitattu 23.9.2020]. Saatavissa: https://graphql.org/learn/schema/

Vyas, U. 2014. DynamoDB Applied Design Patterns. Birmingham, United Kingdom: Packt Publishing Ltd