




OCTOBER 31, 2021

AIFO MINIRPROJEKT-BERICHT

JUNO PORTFOLIO MANAGER

MIRIO EGGMANN
AI FOUNDATIONS
OST - RAPPERSWIL



Inhaltsverzeichnis

EINLEITUNG.....	2
IDEE	2
ARCHITEKTUR.....	2
UMSETZUNG	3
PROJEKTSTRUKTUR UND ABLAGE	3
JAVA CLIENT.....	3
<i>Interaktionsbeispiel</i>	4
TELEGRAM BOT	5
<i>Interaktionsbeispiel</i>	5
DIALOGFLOW AGENT	5
<i>Entities</i>	5
<i>Intents</i>	6
<i>Weitere Einstellungen</i>	6
GOOGLE CLOUD FUNCTIONS FULFILLMENTS	7
COINMARKETCAP API	8
FIRESTORE DATENBANK.....	9
DISKUSSION	10
LITERATURVERZEICHNIS	11

Abbildungsverzeichnis

ABBILDUNG 1 ARCHITEKTUR	2
ABBILDUNG 2 PROJEKTSTRUKTUR	3
ABBILDUNG 3 KLASSENDIAGRAMM JAVA CLIENT	3
ABBILDUNG 4 CODE-AUSSCHNITT DIALOGFLOW PARAMETER IN JAVA CLIENT AUSLESEN	3
ABBILDUNG 5 JAVA CLIENT INTERAKTION 1	4
ABBILDUNG 5 JAVA CLIENT INTERAKTION 2	4
ABBILDUNG 6 TELEGRAM BOT INTERAKTION.....	5
ABBILDUNG 7 GOOGLE CLOUD FUNCTIONS FULFILLMENT INLINE EDITOR.....	7
ABBILDUNG 9 CODE-AUSSCHNITT CURRENCY.DEPOSIT FULFILLMENT	7
ABBILDUNG 10 CODE-AUSSCHNITT COINMARKETCAP API REQUEST OPTIONEN.....	8
ABBILDUNG 11 COINMARKETCAP API ACCOUNT OVERVIEW.....	8
ABBILDUNG 12 CODE-AUSSCHNITT FIRESTORE DATENBANK CREDENTIALS CONFIG.....	9
ABBILDUNG 13 FIRESTORE DATENBANK ÜBERSICHT.....	9

Einleitung

Im Rahmen des Moduls AI Foundations war es die Aufgabe ein Miniprojekt mit Google-Dialogflow zu realisieren und anschliessend einen Bericht darüber zu verfassen. Das Ziel vom Projekt ist praktische Erfahrung mit der Google-Cloud und Dialogflow-Suite zu gewinnen und ein allgemeines Verständnis für AI unterstützte Software zu bekommen.

Folgend mein Bericht zu diesem Miniprojekt. Das Dokument ist so aufgebaut, dass ich zuerst die Projektidee vermittele, anschliessend auf die Architektur und Umsetzung eingehe und zum Schluss eine Diskussion aufführe, wo ich mein Fazit und einen Ausblick gebe.

Idee

Meine Grundidee war es einen Währungsumrechner mit Fokus auf Kryptowährungen zu entwickeln. Seit vier Jahren setze ich mich vertieft mit Kryptowährungen auseinander und war darum interessiert, ein Projekt in diesem Bereich zu realisieren.

Weil es mich dann zu wenig gefordert hat und ich noch etwas mehr zum Thema lernen wollte, kam ich auf die Idee das Projekt von einem Umrechner zu einem Portfolio Manager zu ändern. Somit konnte ich die Umrechnungs-Funktionalität beibehalten und dann weitere Technologien, wie Firestore, zum Speichern von Daten, ausprobieren. Somit ist dann das Projekt Juno Portfolio Manager entstanden.

Architektur

In folgender Abbildung ist die Architektur dargestellt. Auf der linken Seite sieht man den User, welcher den Portfolio-Manager entweder über den Java Client oder den Telegram Bot benutzen kann. Die gelbe Komponente stellt den Dialogflow Agent dar, mit dem die beiden violett dargestellten Benutzerschnittstellen interagieren. Auf der rechten Seite in blau sieht man wie der Dialogflow Agent sogenannte Fulfillments einsetzt. Dies damit die Coinmarketcap API die aktuellen Kryptowährungskurse liefern kann und eine Firestore Datenbank zur Speicherung der Portfolios genutzt werden kann.

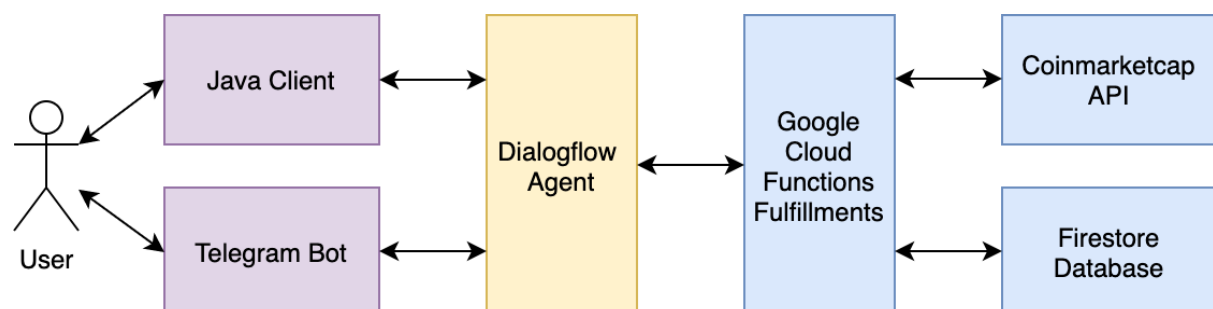
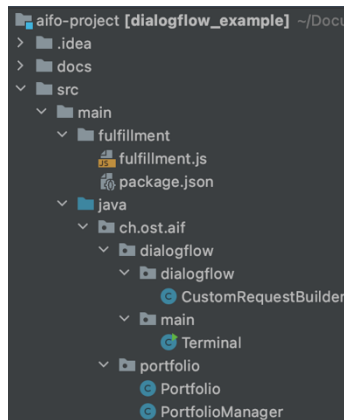


Abbildung 1 Architektur

Umsetzung

Projektstruktur und Ablage

Das Projekt ist in einem Git-Repository abgespeichert und folgendermassen strukturiert:



docs: Hier wird das Word-Dokument für diesen Bericht abgespeichert.

src/main/fulfillment: Hier ist der Code abgelegt, welcher für die Google Cloud Functions Fulfillments verwendet wird.

src/main/java: Hier ist der Java Client Code abgelegt.

Das Git-Repository ist abgelegt und erreichbar unter:

<https://github.com/mirioeggmann/aifo-project>

Abbildung 2 Projektstruktur

Java Client

Als Grundlage für den Client wurde die Vorlage vom Modul genommen, welche in Woche 2 ausgeteilt wurde. Diese wurde so verändert, damit der Client mit dem Juno Dialogflow Agent verwendet werden kann. Zusätzlich wurde implementiert, dass Sachen wie die E-Mail vom Portfolio clientseitig zwischengespeichert werden kann. Das Zwischenspeichern der E-Mail vereinfacht die Interaktion für den User, weil er z.B. nicht mehr bei jedem Deposit oder Withdraw Currency Aufruf seine E-Mail-Adresse mitgeben muss.

Insgesamt wurden zwei neue Klassen angelegt und die CustomRequestBuilder- und Terminal-Klasse wurden dem Projekt entsprechend angepasst. Zudem wurden die unnötigen Teile vom Template von Woche 2 entfernt.

Die Klasse PortfolioManager verwaltet eine Instanz der Klasse Portfolio. In der Klasse Portfolio wird die E-Mail-Adresse des aktiven Portfolios abgespeichert.

Im nebenstehenden Klassendiagramm sind die Instanzvariablen, Konstruktoren und Methoden der neuen Klassen ersichtlich.

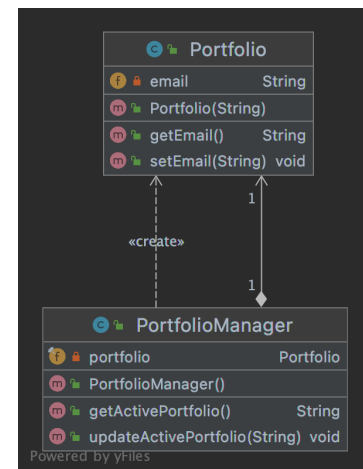


Abbildung 3 Klassendiagramm Java Client

Folgend ein Code-Ausschnitt wie im Java Client umgesetzt wurde, dass ein bestimmtes Portfolio selektiert werden kann.

Zum Auslesen der Parameter muss umständlich eine Map erstellt werden über die iteriert werden kann. Falls es einen Key «email» findet, wird dieser entsprechend mit «updateActivePortfolio()» im PortfolioManager abgespeichert.

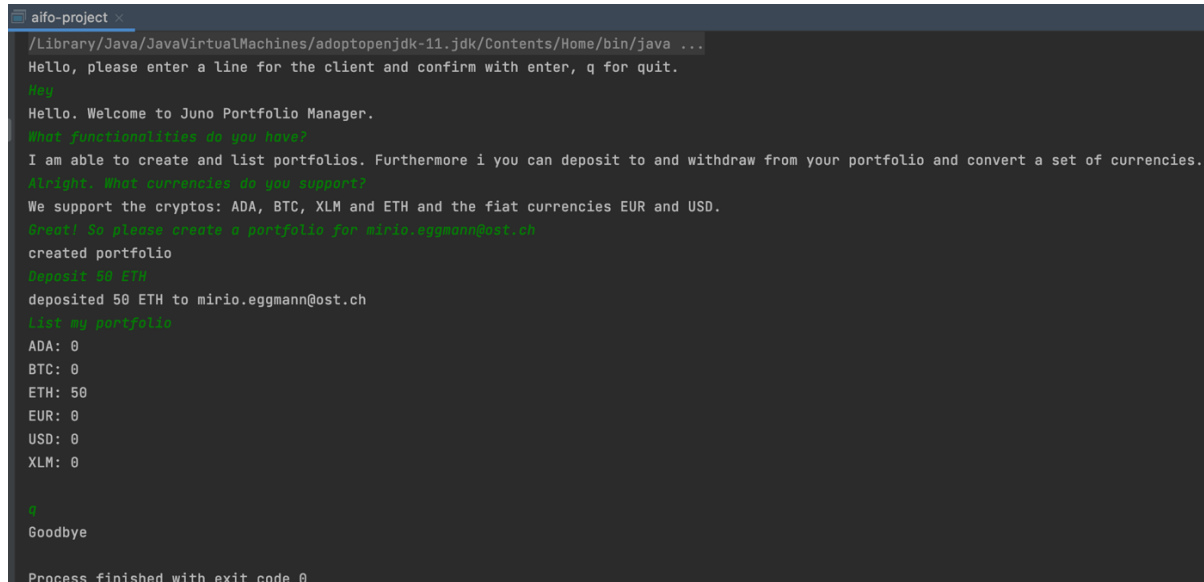
```

case "portfolio.select":
    for (Map.Entry<String, Value> entry :
        queryResult.getParameters().getFieldsMap().entrySet()) {
        if (entry.getKey().equals("email")) {
            portfolioManager.updateActivePortfolio(entry.getValue().getStringValue());
        }
    }
    break;
  
```

Abbildung 4 Code-Ausschnitt Dialogflow Parameter in Java Client auslesen

Interaktionsbeispiel

In folgender Abbildung ist ersichtlich, wie eine Konversation über den Client aussehen kann. Zuerst wird gefragt, welche Funktionalitäten angeboten und welche Währungen unterstützt werden. Dann wird ein Portfolio angelegt und 50 Ethereum darauf gutgeschrieben. Anschliessend wird das Portfolio ausgegeben worauf ersichtlich ist, dass das Portfolio nun 50 ETH hält.

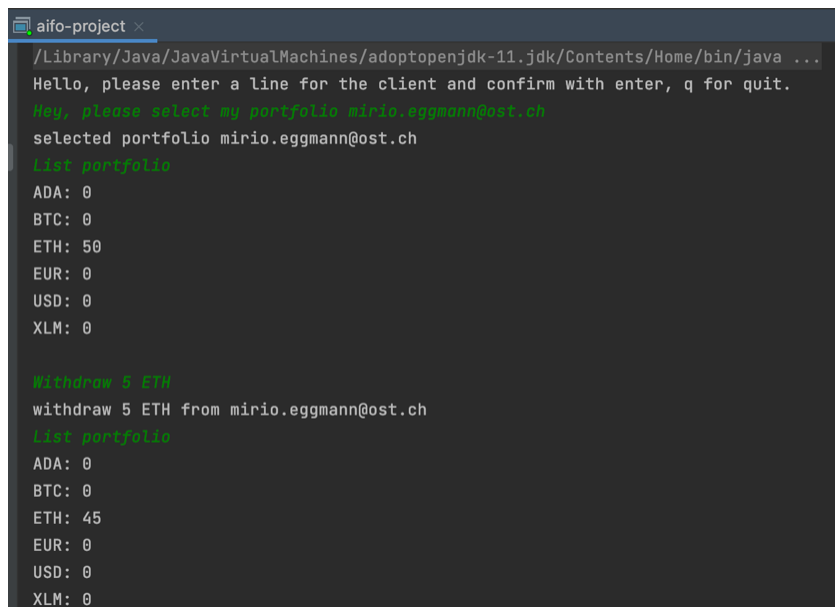


```
aifo-project x
/Library/Java/JavaVirtualMachines/adoptopenjdk-11.jdk/Contents/Home/bin/java ...
Hello, please enter a line for the client and confirm with enter, q for quit.
Hey
Hello. Welcome to Juno Portfolio Manager.
What functionalities do you have?
I am able to create and list portfolios. Furthermore i you can deposit to and withdraw from your portfolio and convert a set of currencies.
Alright. What currencies do you support?
We support the cryptos: ADA, BTC, XLM and ETH and the fiat currencies EUR and USD.
Great! So please create a portfolio for mirio.eggmann@ost.ch
created portfolio
Deposit 50 ETH
deposited 50 ETH to mirio.eggmann@ost.ch
List my portfolio
ADA: 0
BTC: 0
ETH: 50
EUR: 0
USD: 0
XLM: 0

q
Goodbye
Process finished with exit code 0
```

Abbildung 5 Java Client Interaktion 1

In der nächsten Abbildung ist ersichtlich, wie ein Portfolio auch bei einem erneuten Start des Clients wiederverwendet werden kann. Das Portfolio wird erneut aufgelistet und es ist ersichtlich, dass die 50 ETH immer noch enthalten sind. Dies, weil es in einer Firestore Datenbank auf Seite des Backends gespeichert wird. Des Weiteren ist auch noch ersichtlich, dass eine Auszahlung getätigt werden kann und vom Portfolio entsprechend abgebucht wird.



```
aifo-project x
/Library/Java/JavaVirtualMachines/adoptopenjdk-11.jdk/Contents/Home/bin/java ...
Hello, please enter a line for the client and confirm with enter, q for quit.
Hey, please select my portfolio mirio.eggmann@ost.ch
selected portfolio mirio.eggmann@ost.ch
List portfolio
ADA: 0
BTC: 0
ETH: 50
EUR: 0
USD: 0
XLM: 0

Withdraw 5 ETH
withdraw 5 ETH from mirio.eggmann@ost.ch
List portfolio
ADA: 0
BTC: 0
ETH: 45
EUR: 0
USD: 0
XLM: 0
```

Abbildung 6 Java Client Interaktion 2

Telegram Bot

Aus Neugier wurde zusätzlich noch ein Telegram Bot umgesetzt. Die Google Cloud hat dazu bereits Integrationen, welche es einfach machen einen solchen Bot anzubinden.

Dazu musste ein Bot über den sogenannten «botfather» von Telegram erstellt werden (<https://telegram.me/botfather>) und der API Key anschliessend in Dialogflow hinterlegt werden.

Interaktionsbeispiel

Eine Konversation kann wie in nebenstehender Abbildung mit /start gestartet werden.

Der Bot hat im Gegensatz zum Java-Client jedoch den Nachteil, dass bei Aktionen die mit dem Portfolio zu tun haben immer die E-Mail mitgegeben werden muss.

Man sieht ebenfalls wie der Agent dank dem Small-Talk-Plugin auf Fragen wie «How are you?» antworten kann.

Auch ist ersichtlich, wie der Umrechnungskurs laut Coinmarketcap von 100'000 Euro zu Bitcoin aktuell aussieht.

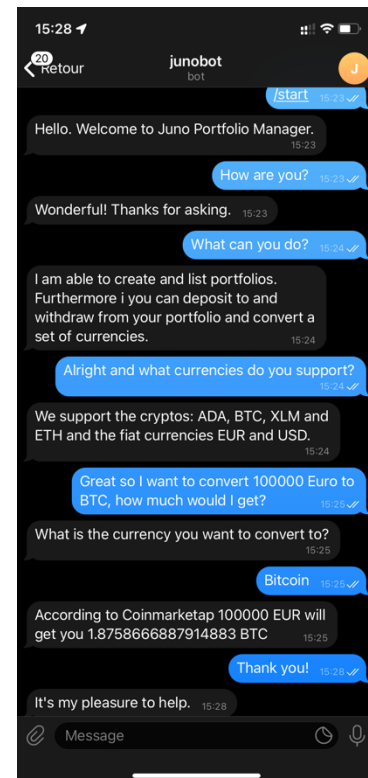


Abbildung 7 Telegram Bot Interaktion

Dialogflow Agent

Als Grundlage für den Agent wurde der Currency-Converter-Prebuilt-Agent verwendet. Dieser bietet eine minimale Grundlage zur Erkennung von Währungen im currency.convert-Intent. Allerdings wird keine Währungsumwandlungs-Logik mitgeliefert. Auch konnte er noch keine Antworten auf Fragen vom User liefern. Dies musste ich alles selbst implementieren und daher habe ich den Agent für dieses Miniprojekt praktisch von Grund auf neu gebaut.

Entities

Als Entities konnten grösstenteils bestehende System-Entities verwendet werden. Die verwendeten Entities können in der Tabelle der Intents in der Spalte Parameter angeschaut werden. Eine zusätzliche Entity wurde ergänzt, damit die unterstützten Währungen sauber erkannt werden können.

Entity	Konfiguration	Beschreibung												
@currency	<div><input checked="" type="checkbox"/> Define synonyms ⓘ <input type="checkbox"/> Regexp entity ⓘ <input type="checkbox"/> Allow automated expansion <input checked="" type="checkbox"/> Fuzzy matching ⓘ</div> <table><tr><td>BTC</td><td>BTC, Bitcoin, btc</td></tr><tr><td>ETH</td><td>ETH, Ethereum</td></tr><tr><td>XLM</td><td>XLM, Stellar, Stellar Lumens</td></tr><tr><td>ADA</td><td>ADA, Cardano</td></tr><tr><td>EUR</td><td>EUR, Euro</td></tr><tr><td>USD</td><td>USD, Dollar</td></tr></table>	BTC	BTC, Bitcoin, btc	ETH	ETH, Ethereum	XLM	XLM, Stellar, Stellar Lumens	ADA	ADA, Cardano	EUR	EUR, Euro	USD	USD, Dollar	Hat definierte Synonyme und setzt «Fuzzy matching» ein, also kann somit auch Wörter erkennen, obwohl der Benutzer Tippfehler macht.
BTC	BTC, Bitcoin, btc													
ETH	ETH, Ethereum													
XLM	XLM, Stellar, Stellar Lumens													
ADA	ADA, Cardano													
EUR	EUR, Euro													
USD	USD, Dollar													

Intents

In folgender Tabelle sind die Intents vom Miniprojekt ersichtlich. Die neu hinzugefügten Intents sind grün, die gelben sind auf der Basis vom Prebuilt-Agent erweitert. Der «Default Welcome Intent» und «Default Fallback Intent» werden nicht aufgeführt, weil die nicht angepasst wurden. Zu allen Intents wurden auch noch Trainingssätze erfasst. Diese werden hier nicht noch explizit aufgeführt.

Intent	Parameter	Beschreibung												
currency.available	-	Auflistung aller unterstützten Währungen des Portfolios Manager.												
currency.convert <i>mit fulfillment</i>	<table> <tr> <th>PARAMETER NAME</th><th>ENTITY</th><th>VALUE</th></tr> <tr> <td>currencyFrom</td><td>@currency</td><td>\$currencyFrom</td></tr> <tr> <td>currencyTo</td><td>@currency</td><td>\$currencyTo</td></tr> <tr> <td>amount</td><td>@sys.number</td><td>\$amount</td></tr> </table>	PARAMETER NAME	ENTITY	VALUE	currencyFrom	@currency	\$currencyFrom	currencyTo	@currency	\$currencyTo	amount	@sys.number	\$amount	Gibt den laut Coinmarketcap aktuellen Wechselkurs für ein Währungspaar zurück.
PARAMETER NAME	ENTITY	VALUE												
currencyFrom	@currency	\$currencyFrom												
currencyTo	@currency	\$currencyTo												
amount	@sys.number	\$amount												
currency.deposit <i>mit fulfillment</i>	<table> <tr> <th>PARAMETER NAME</th><th>ENTITY</th><th>VALUE</th></tr> <tr> <td>currency</td><td>@currency</td><td>\$currency</td></tr> <tr> <td>amount</td><td>@sys.number</td><td>\$amount</td></tr> <tr> <td>email</td><td>@sys.email</td><td>\$email</td></tr> </table>	PARAMETER NAME	ENTITY	VALUE	currency	@currency	\$currency	amount	@sys.number	\$amount	email	@sys.email	\$email	Einzahlen einer unterstützten Währung auf ein Portfolio.
PARAMETER NAME	ENTITY	VALUE												
currency	@currency	\$currency												
amount	@sys.number	\$amount												
email	@sys.email	\$email												
currency.withdraw <i>mit fulfillment</i>	<table> <tr> <th>PARAMETER NAME</th><th>ENTITY</th><th>VALUE</th></tr> <tr> <td>currency</td><td>@currency</td><td>\$currency</td></tr> <tr> <td>email</td><td>@sys.email</td><td>\$email</td></tr> <tr> <td>amount</td><td>@sys.number</td><td>\$amount</td></tr> </table>	PARAMETER NAME	ENTITY	VALUE	currency	@currency	\$currency	email	@sys.email	\$email	amount	@sys.number	\$amount	Auszahlen einer Währung eines Portfolios.
PARAMETER NAME	ENTITY	VALUE												
currency	@currency	\$currency												
email	@sys.email	\$email												
amount	@sys.number	\$amount												
functionalities	-	Gibt eine Beschreibung zurück, welche Funktionalitäten der Bot alles bietet.												
portfolio.create <i>mit fulfillment</i>	<table> <tr> <th>PARAMETER NAME</th><th>ENTITY</th><th>VALUE</th></tr> <tr> <td>email</td><td>@sys.email</td><td>\$email</td></tr> </table>	PARAMETER NAME	ENTITY	VALUE	email	@sys.email	\$email	Erstellen eines Portfolios.						
PARAMETER NAME	ENTITY	VALUE												
email	@sys.email	\$email												
portfolio.list <i>mit fulfillment</i>	<table> <tr> <th>PARAMETER NAME</th><th>ENTITY</th><th>VALUE</th></tr> <tr> <td>email</td><td>@sys.email</td><td>\$email</td></tr> </table>	PARAMETER NAME	ENTITY	VALUE	email	@sys.email	\$email	Auflistung aller Währungen in einem Portfolio.						
PARAMETER NAME	ENTITY	VALUE												
email	@sys.email	\$email												
portfolio.select	<table> <tr> <th>PARAMETER NAME</th><th>ENTITY</th><th>VALUE</th></tr> <tr> <td>email</td><td>@sys.email</td><td>\$email</td></tr> </table>	PARAMETER NAME	ENTITY	VALUE	email	@sys.email	\$email	Intent, welcher vom Java Client verwendet wird um das gewünschte Portfolio zu selektieren, damit nicht immer die E-Mail mitgegeben werden muss bei Abfragen wo sie als Parameter benötigt wird.						
PARAMETER NAME	ENTITY	VALUE												
email	@sys.email	\$email												

Weitere Einstellungen

Es wurde noch das «Small Talk» Plugin aktiviert, damit der Bot etwas schlagfertiger ist und auf einige unerwartete Fragen vom Benutzer antworten kann.

Google Cloud Functions Fulfillments

In nebenstehender Grafik ist der aktivierte Inline-Editor ersichtlich, mit welchem ich die Fulfillments realisiert habe. Dazu musste bei Firebase noch der Billing-Account hinzugefügt werden.

Anschliessend kann grundsätzlich direkt in diesem Inline-Editor gearbeitet werden. Relativ schnell habe ich dann aber zum Modell gewechselt, dass ich den Code zuerst in meinem Git-Repository geschrieben habe und anschliessend in diesen Inline-Editor reinkopiert habe. Dies, weil ich an einem Punkt eine Änderung gemacht habe, welches ein Fulfillment unbrauchbar gemacht hat und ich dann aber nicht auf einen alten Stand zurückkehren konnte, weil dieser Online-Editor keine Versionierung bietet.

⚡ Fulfillment

Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the [webhook requirements](#) specific to the API version enabled in this agent.

Inline Editor (Powered by Google Cloud Functions)

ENABLED

Build and manage fulfillment directly in Dialogflow via Cloud Functions. [Docs](#)

Newly created cloud functions now use Node.js 10 as runtime engine. Check [migration guide](#) for more details.

```
index.js  package.json
154
155     function fallback(agent) {
156         agent.add('I didn't understand');
157         agent.add('I'm sorry, can you try again?');
158     }
159
160     let intentMap = new Map();
161     intentMap.set('currency.deposit', depositCurrencyHandler);
162     intentMap.set('currency.withdraw', withdrawCurrencyHandler);
163     intentMap.set('currency.convert', convertCurrencyHandler);
164     intentMap.set('portfolio.create', createPortfolioHandler);
165     intentMap.set('portfolio.list', listPortfolioHandler);
166     intentMap.set('Default Fallback Intent', fallback);
167     agent.handleRequest(intentMap);
168 };
```

Abbildung 8 Google Cloud Functions Fulfillment Inline Editor

Folgend noch ein Code-Ausschnitt, welcher zeigt, wie ich eines dieser aufgeführten Fulfillments genau implementiert habe. Es handelt sich um das Fulfillment für den currency.deposit-Intent. Im ersten Abschnitt werden die Parameter email, currency und amount ausgelesen, welche im Intent vom Agent gesammelt wurden. Anschliessend wird der Pfad für die Währung in Firestore in eine Variable gespeichert. Wie die Hierarchie von Firestore in diesem Projekt genau aufgebaut ist, wird weiter unten im Kapitel «Firestore Datenbank» erklärt. Im letzten Abschnitt sieht man wie der aktuelle Kontostand aus der DB geladen wird und anschliessend der neue Stand persistiert wird.

```
function depositCurrencyHandler(agent) {
    const email = agent.parameters.email;
    const currency = agent.parameters.currency;
    const amount = agent.parameters.amount;

    const currencyRef =
db.collection('portfolio').doc(email).collection('currency').doc(currency);

    return currencyRef.get()
        .then(doc => {
            let balance = JSON.stringify(doc.data().balance);
            let newBalance = parseFloat(balance) + parseFloat(amount);
            return currencyRef
                .set({balance: newBalance})
                .then(() => agent.add(`deposited ${amount} ${currency}`))
                .catch(() => agent.add(`deposit failed.`));
        }).catch((err) => {
            console.log(err);
            agent.add(`Error occurred fetching value of ${currency}`);
        });
}
```

Abbildung 9 Code-Ausschnitt currency.deposit Fulfillment

Ich habe auch noch versucht, ob ich anstelle vom vorherigen Auslesen ein Firebase-Increment machen könnte. Dies würde einen Call auf die Firestore DB sparen. Allerdings habe ich auch nach mehreren Stunden nicht herausgefunden, wie man dieses Increment zum Laufen bringt und habe mich daher auf den Rest des Projekts fokussiert. In der Diskussion werde ich diesen Punkt nochmal aufbringen.

Coinmarketcap API

Damit die Währungskurse nicht statisch in Dialogflow hinterlegt werden mussten, habe ich diese mithilfe der Fulfillments über eine API bezogen. Ich habe mich für die Coinmarketcap API entschieden, weil Coinmarketcap ein grosser Player in der Krypto-Szene ist und eine solide API anbietet, die zu experimentellen Zwecken (bis 333 Requests pro Tag) gratis nutzbar ist. Die API bietet neben allen Kryptowährungskursen auch die der bekannten staatlichen Währungen, wie Dollar und Euro an.

Zur Nutzung der API musste man sich lediglich auf der Webseite (<https://pro.coinmarketcap.com/account>) registrieren und anschliessend einen API Key generieren.

Hier noch ein Code-Ausschnitt, wie die API in den Fulfillments genutzt wird und wie der API Key dann mitgegeben werden muss.

```
const requestOptions = {
  method: 'GET',
  uri: 'https://pro-api.coinmarketcap.com/v1/tools/price-conversion',
  qs: {'symbol': currencyFrom, 'convert': currencyTo, 'amount': amount},
  headers: {'X-CMC_PRO_API_KEY': 'HIER_KOMMT_DER_KEY_HIN'},
  json: true,
  gzip: true
};
```

Abbildung 10 Code-Ausschnitt Coinmarketcap API Request Optionen

Als Library zum Aufruf der API wurde zuerst mit Axiom (<https://www.npmjs.com/package/axiom-api>) gearbeitet und dann habe ich es aber noch auf request-promise (<https://www.npmjs.com/package/request-promise>) geändert. Diese ist zwar deprecated, funktioniert aber deutlich intuitiver und reicht für einen Proof of Concept wie in diesem Projekt grundsätzlich völlig. Die Quellen, welche zum Verständnis der gesamten Implementierung genutzt wurden, sind im Literaturverzeichnis ersichtlich.

Folgend ein Screenshot der Account Übersicht von der Coinmarketcap API. Wie man sieht, kann man dort den Key generieren und schauen wie viel man von seinem Kontingent bereits genutzt hat.

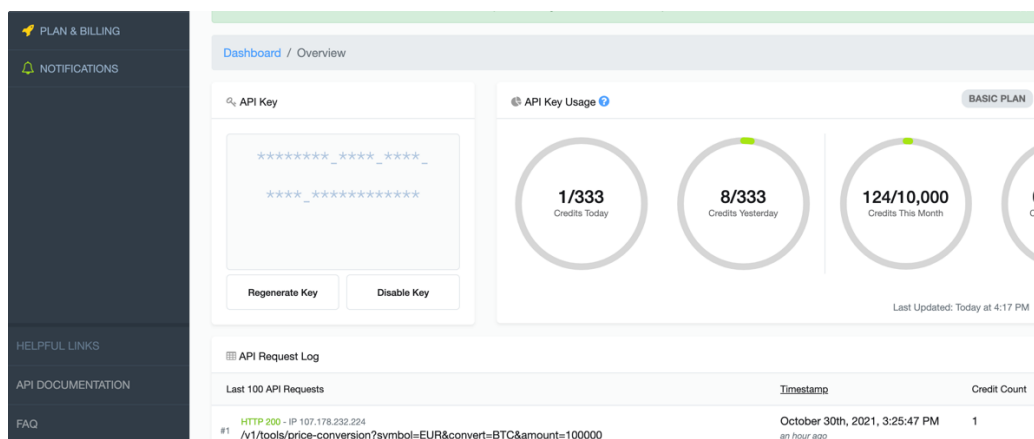


Abbildung 11 Coinmarketcap API Account Overview

Firestore Datenbank

Zur Speicherung vom Portfolio wurde eine Firestore DB eingesetzt. Da ich noch nie mit Firebase gearbeitet habe, musste ich mich zuerst darüber schlau machen. Ich habe diverse Youtube-Videos zu diesem Thema gefunden und konnte den Code entsprechend implementieren. Jedoch hat es nicht auf Anhieb geklappt und ich habe einige Stunden damit verbracht herauszufinden, wieso ich mich nicht auf die Datenbank verbinden kann. Die Implementation vom Youtube-Video, welches ich im Literaturverzeichnis verlinkt habe, funktioniert nicht mehr ganz so in der aktuellen Google Cloud Functions Version und daher musste ich noch folgende Anpassung machen. Dies damit die richtigen Credentials verwendet werden.

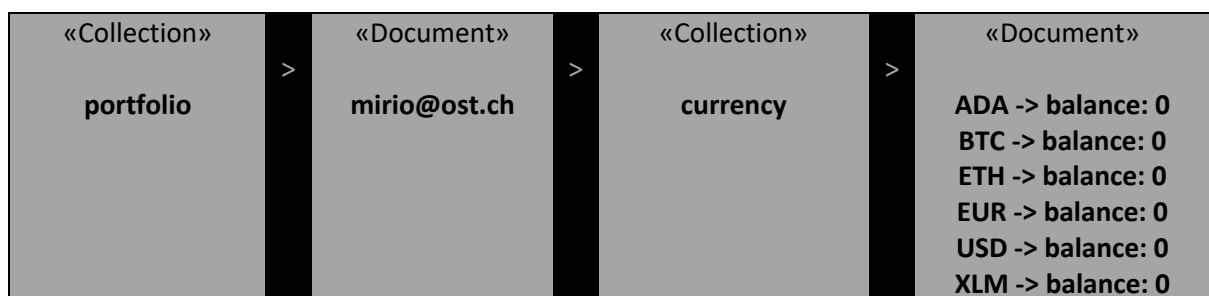
```
admin.initializeApp(functions.config().firebase);
const db = admin.firestore();
```

Abbildung 12 Code-Ausschnitt Firestore Datenbank Credentials Config

Die Firestore Datenbank wurde folgendermassen strukturiert.



Folgend ein Beispiel anhand von «mirio@ost.ch» bei einem neu erstellten Portfolio.



In der unterstehenden Abbildung ist die Firebase Benutzeroberfläche mit dem Firestore Navigator ersichtlich. In diesem Fall mit dem Portfolio von «mirio.eggmann@ost.ch» als Beispiel.

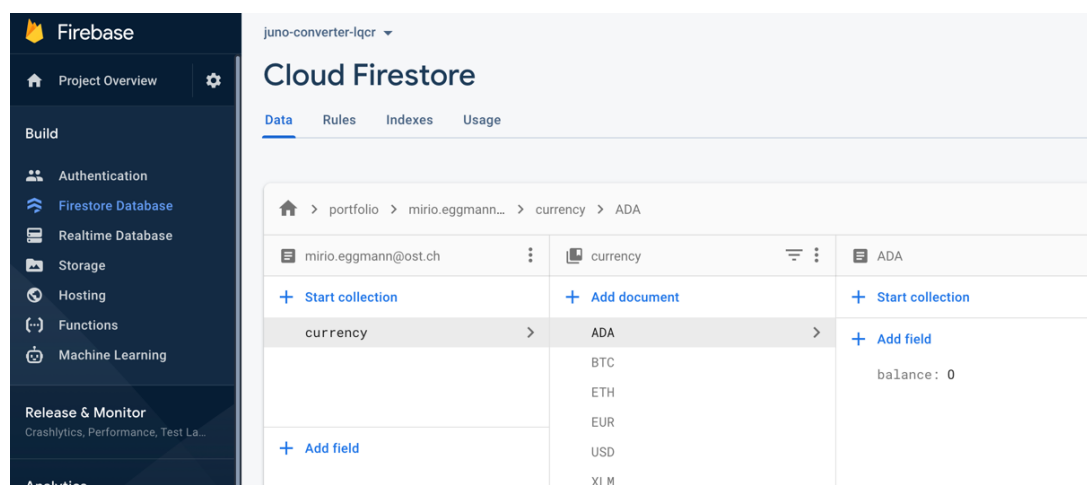


Abbildung 13 Firestore Datenbank Übersicht

Diskussion

Durch dieses Miniprojekt konnte ich Erfahrung mit Dialogflow und diversen weiteren Technologien, Konzepten und Plattformen wie Firebase und Google Cloud Functions sammeln. Bei Dialogflow hätte ich erwartet, dass es noch etwas mehr kann als intelligente Texterkennung. Richtig eingesetzt finde ich es aber durchaus praktisch, wenn man z.B. in kurzer Zeit einen Chatbot bauen möchte ohne grossen Aufwand. Bei einem zukünftigen Hackathon könnte ich es mir durchaus vorstellen Dialogflow für ein Proof of Concept einzusetzen.

Bei einem nächsten Projekt in diesem Stil werde ich darauf schauen, dass ich es wirklich als Teamarbeit machen kann wie empfohlen. In diesem Fall ging es leider gerade nicht gut auf in meiner Übungsgruppe und dann habe ich mich entschieden das Projekt allein zu realisieren. Trotzdem hat es gut funktioniert, weil ich mich regelmässig mit meiner Lerngruppe austauschen konnte und so trotzdem vom Teamarbeits-Lerneffekt profitieren konnte.

Das Umsetzen der Google Cloud Functions in JavaScript für die Fulfillments war ziemlich mühsam. Damit ich neue Funktionalitäten testen konnte, musste ich jeweils eine neue Version deployen und das hat immer mehrere Minuten gedauert. Setze ich in Zukunft nochmals solche Fulfillments ein, würde ich zuerst eine anständige Entwicklungsumgebung einrichten, damit ich diese auch lokal ausführen und testen könnte.

Was ich für neue Erkenntnisse gewonnen habe bei der Arbeit mit den Google Cloud Functions ist wie diese serverless Services funktionieren und generell, wie man mit JavaScript Promises umgehen muss.

Beim Anbinden der Firestore Datenbank in den Fulfillments ist mir aufgefallen, dass viele Beispiele und Unterlagen dazu ziemlich veraltet sind und auch das mit den Increments hat nicht recht funktioniert. Trotzdem werde ich mich wohl in naher Zukunft mal noch genauer mit Firebase auseinandersetzen, weil ich es trotzdem sehr spannend fand, was man damit anstellen kann. In kurzer Zeit konnte ich damit eine Datenbank erstellen, welche dann über das Web abrufbar ist. In einem nächsten Schritt müsste ich dafür aber noch ein Sicherheitskonzept erstellen, damit auch wirklich nur gewünschte Teilnehmer auf der Datenbank lesen und schreiben können.

Den Portfolioteil würde ich in einem nächsten Release noch verbessern, dass es eine Art Login gibt, damit nur die Besitzer der Portfolios auf Ihre eigenen Daten zugreifen können. Da es sich hier aber nur um einen Proof of Concept handelte wurde der Fokus auf die wesentlichen Funktionalitäten, wie Ein- und Auszahlen gelegt.

In einem weiteren Schritt müsste der Bot auch noch mit deutlich mehr Trainingssätzen ausgestattet werden, damit er auch komplexere Sätze oder Tippfehler besser erkennen könnte. Gerade beim Beispiel vom Telegram Bot hat man gesehen, dass er bei gewissen Satzstrukturen noch etwas Mühe hat, das wesentliche rauszufiltern.

Im Rahmen von diesem Projekt habe ich mich bewusst dazu entschieden die ganze Persistenz serverseitig zu lösen. Somit hatte ich die Möglichkeit viele neue Technologien kennenzulernen. Auch konnte ich sehr einfach einen Telegram Bot hinzufügen, weil man zum Gebrauch von meinem Agent keinen speziell modifizierten Client benötigt, weil grundsätzlich die Logik und Persistenz alles im Backend implementiert ist. Trotzdem könnte man sich überlegen, ob es je nachdem Sinn machen würde eine persönliche Portfolioverwaltung eher lokal persistieren zu wollen.

Alles in allem war es eine tolle Arbeit und ich kann viel neue Erkenntnisse daraus mitnehmen.

Literaturverzeichnis

Lehmann Marco, Purandare Mitra (2021). AIFo 2021 Graded Miniproject. OST.

Lehmann Marco, Purandare Mitra (2021). AI as a Service: Learn how to use the Dialogflow-API. OST.

Einsatz von Fulfillments. Zugriff am 19.10.2021. Verfügbar unter https://www.youtube.com/watch?v=9THX-z7C_ZA.

Beispiele von Google Cloud Functions Fulfillment. Zugriff am 19.10.2021. Verfügbar unter <https://github.com/dialogflow/dialogflow-fulfillment-nodejs>

Externe API in Google Cloud Functions Fulfillments verwenden. Zugriff am 21.10.2021. Verfügbar unter <https://www.youtube.com/watch?v=n4IPOeFCDxI>

Coinmarketcap API-Doc zur Währungsumrechnung. Zugriff am 21.10.2021. Verfügbar unter <https://coinmarketcap.com/api/documentation/v1/#operation/getV1ToolsPriceconversion>

Beispiel Firestore in Google Cloud Functions Fulfillments. Zugriff am 26.10.2021. Verfügbar unter <https://github.com/dialogflow/fulfillment-firestore-nodejs>

Anleitung zur Integration von Firestore in Fulfillments. Zugriff am 26.10.2021. Verfügbar unter <https://www.youtube.com/watch?v=sdbgWXwPW0c>

Lösung zum Credentials-Problem vom Anleitungsvideo für die Firestore Integration. Zugriff am 26.10.2021. Verfügbar unter <https://stackoverflow.com/questions/58974879/using-firebase-admin-what-are-service-account-credentials-used-for>

Firestore Dokumentation zur Nutzung mit NodeJS Fulfillments. Zugriff am 26.10.2021. Verfügbar unter <https://firebase.google.com/docs/firestore/>

Die Antwort zu dieser Frage hat Inspiration gegeben, wie Firestore Collections und Dokumente strukturiert werden können. Zugriff am 27.10.2021. Verfügbar unter <https://stackoverflow.com/questions/56323638/how-to-set-data-to-firebase-map-object>

Dialogflow Telegram-Integrations-Dokumentation. Zugriff am 30.10.2021. Verfügbar unter <https://cloud.google.com/dialogflow/es/docs/integrations/telegram>

Telegram Bot Dokumentation. Zugriff am 30.10.2021. Verfügbar unter <https://core.telegram.org/bots>

Beispiel von Firestore Increment. Zugriff am 30.10.2021. Verfügbar unter <https://github.com/firebase/snippets-web/blob/1e8f41c904d557f486cdab2a1401ec5f6033dc39/firestore/test.firestore.js#L505>

Anleitung zu sauberem Schreiben. Zugriff am 31.10.2021. Verfügbar unter <https://www.acrolinx.com/blog/how-to-tidy-up-your-writing/>