UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE
MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA TEHNOLOGIA INFORMAȚIEI

LUCRARE DE LICENȚĂ

Absolvent

Andriță Lucian-Gabriel

Coordonator științific

Prof. dr. Cristian Kevorchian

București, iunie 2023

UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE
MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA TEHNOLOGIA INFORMAȚIEI

Lucrare de licență

PROIECTAREA ȘI IMPLEMENTAREA ARHITECTURILOR CHATBOT CU APLICAȚII ÎN INDUSTRIA BANCARĂ

Absolvent

Andriță Lucian-Gabriel

Coordonator științific

Prof. dr. Cristian Kevorchian

București, iunie 2023

Rezumat

Acesta este un șablon C++ care utilizează Dialogflow ES de la Google Cloud Platform pentru interacțiuni între oameni și chatbot în diferite contexte, în funcție de cazurile de utilizare ale întreținătorului. Această aplicație în particular folosește procesarea în cloud într-un context bancar și a fost creată ca o dovadă a unui concept, obiectivul principal fiind de a-mi extinde cunoștințele cu privire la interacțiunile C++ într-un mediu modern, utilizând diferite servicii cloud pentru a izola funcționalitățile principale ale aplicației, mutând implementarea locală într-una găzduită în cloud.

Serverul care gestionează conexiunea între clienți și agent este găzduit local. Atunci când se primește o solicitare de la un client, serverul decide ce tip de fișiere să trimită. Pe măsură ce clientul își face interogari către agent, un script va acționa ca parser pentru textul dat și va asculta răspunsul de la server. Când serverul primește intrarea, o va trimite la agent. În cloud, agentul va potrivi textul primit cu o intenție, va extrage unii parametrii în funcție de intenție și va apela un webhook. Webhook-ul constă într-o funcție Google Cloud Function, care este găzduită pe Google Source Repository, unde va căuta în dosarul *cloud* și va extrage funcționalitatea de acolo.

Funcția va avea privilegii CRUD+L asupra unui bucket Google Cloud Storage care stochează informațiile necesare. Acestă lucrare reprezintă o contribuție semnificativă la înțelegerea modului în care serviciile de cloud pot fi integrate cu succes în aplicațiile bazate pe C++, cu un accent specific pe dezvoltarea chatbot-urilor în industria bancară.

Abstract

This is a C++ template that uses Google Cloud Platform's Dialogflow ES for human-chatbot interactions in different contexts, depending on the use cases of the maintainer. This particular application uses cloud processing in a banking context and was created as a proof of concept, its main goal being to further expand my knowledge regarding C++ interactions in a modern enviroment, while using different cloud services to isolate main functionalities of the application moving the local implementation to a cloud hosted one.

The server that is serving the connection between the clients and the agent is locally hosted. When a request from a client is received, the server decides what type of files to send. As the client queries his intent to the agent, a script will act as a parser for the given text, and will listen for the response from the server. When the server receives the input, it will send it to the agent. In the cloud, the agent will match the received text with an intent, it will extract some parameters based on the intent, and it will call a webhook. The webhook consists of a Google Cloud Function, that's being hosted on Google Source Repository, where it will look into the *cloud* folder and extract the functionality from there.

The function will have CRUD+L privileges on a Google Cloud Storage bucket that stores the needed information. This paper represents a significant contribution to the understanding of how cloud services can be successfully integrated into C++ based applications, with a specific focus on the development of chatbots in the banking industry.

Cuprins

Capitolul I

Introducere

I.1 Nevoia migrării în cloud

În era digitală modernă, tendința de procesare mutată în cloud a devenit o parte integrală a multor domenii și industrii. Această tendință se referă la transferul și executarea operațiunilor de procesare a datelor și sarcinilor complexe în cloud, în loc să fie realizate local.

Procesarea mutată în cloud oferă o abordare inovatoare și scalabilă pentru gestionarea volumelor mari de date și sarcini computaționale intensive. Prin intermediul infrastructurii cloud, această tendință permite accesul la resurse computaționale puternice și flexibile, oferind astfel utilizatorilor o experiență îmbunătățită și performanțe superioare în timp real [14].

Prin transferarea sarcinilor în mediul cloud, utilizatorii beneficiază de o serie de avantaje. Unul dintre acestea este accesul la capacități de procesare și resurse de stocare extinse, care pot depăși de multe ori puterea de calcul a dispozitivelor individuale. Această scalabilitate permite utilizatorilor să gestioneze sarcini complexe, cum ar fi analiza unor seturi de date foarte mari, procesarea grafică avansată sau algoritmi de învățare automată, fără a se confrunta cu constrângerile tehnice ale dispozitivelor personale.

Pe lângă avantajele de performanță, procesarea mutată în cloud oferă și o mai mare flexibilitate și mobilitate. Utilizatorii pot accesa și gestiona datele și aplicațiile lor de pe diferite dispozitive, indiferent de locație sau de sistemul de operare utilizat. Acest aspect aduce un nivel crescut de colaborare și sincronizare între utilizatori, sporind eficiența și productivitatea în mediul de lucru modern.

Totuși, pe lângă beneficiile pe care le aduce, procesarea mutată în cloud aduce anumite provocări și preocupări. Una dintre ele este legată de securitatea datelor și confidențialitatea informațiilor. Deoarece datele sunt transferate și procesate în mediul cloud, există riscul ca acestea să fie expuse la amenințări cibernetice sau să fie accesate de persoane neautorizate. Drept urmare, furnizorii de servicii cloud trebuie să

implementeze măsuri solide de securitate și criptare pentru a proteja datele utilizatorilor.

I.2 Scopul lucrării

Aplicația este un template care îmbină procese moderne de prelucrare și stocare a datelor folosind servicii cloud. Scopul acesteia este de a fi o implementare practică a unui concept și de a-mi îmbunătăți cunoștiințele cu privire la servicii cloud, arhitectură de aplicații, limbajul C++, clean coding, precum și best practices în acest context. Codul sursă se poate găsi la adresa https://github.com/AndritaLucianGabriel/BachelorsDegree.

Alegerea unui limbaj precum C++ pentru dezvoltarea unei aplicații web a prezentat o provocare în ceea ce privește alegerea utilitarului potrivit pentru crearea și gestionarea serverului. Multe aplicații întâlnite folosesc JavaScript și utilitare pentru conectarea la diferiți furnizori de servicii cloud, astfel curiozitatea mea m-a făcut să mă întreb de aplicabilitatea unor alte suite de tehnologii pentru o astfel de aplicație, păstrând totuși ideea centrală de utilizare a cât mai multe servicii cloud pentru diferite functionalități.

I.3 Obiective

În partea de implementare propriu-zisă, am avut în vedere câteva obiective:

1. Investigarea caracteristicilor unice ale Dialogflow și motivul pentru care acesta este o alegere potrivită pentru dezvoltarea chatbotilor în sisteme bancare.

2. Proiectarea și implementarea unui model de chatbot bazat pe Dialogflow, specializat pentru utilizare în sistemul bancar. Acesta ar trebui să fie capabil să răspundă la întrebări frecvente, să ajute clienții să-și gestioneze conturile și tranzacțiile.

3. Evaluarea performanței modelului de chatbot, în ceea ce privește acuratețea și promptitudinea răspunsurilor, precum și satisfacția generală a utilizatorilor.

4. Explorarea viitoarelor posibilități de îmbunătățire și adaptare a chatbotilor pentru a răspunde mai bine nevoilor utilizatorilor de servicii bancare.

I.4 Motivația personală

Într-o eră în care tehnologia se dezvoltă cu o rapiditate neegalată, am ales să îmi concentrez lucrarea de licență asupra unui subiect care mă fascinează: arhitectura de tip chatbot în sisteme bancare. Această

alegere nu a fost aleatorie, ci a fost alimentată de interesul meu pentru dezvoltarea software, în special în limbajul C++, și de dorința de a explora potențialul aplicațiilor cloud în acest domeniu.

Încă de la începutul studiilor mele, am fost atras de complexitatea și puterea limbajului de programare C++. Acest limbaj mi-a oferit posibilitatea de a construi soluții software robuste, eficiente și flexibile. Dezvoltarea mea personală în acest sens a reprezentat o provocare continuă, dar și o sursă constantă de satisfacție. Alegerea acestei teme pentru lucrarea mea de licență este o oportunitate excelentă de a îmbina abilitățile mele în programarea C++ cu cele în AI și cloud computing.

În ultimii ani, am urmărit cu interes cum organizațiile bancare au început să adopte soluții bazate pe AI pentru a îmbunătăți serviciile pentru clienți și pentru a eficientiza operațiunile interne. Deși există multe implementări de succes (de exemplu ADA [11], asistentul virtual BCR), acest domeniu încă prezintă un potențial incomplet exploatat. Existența acestui potențial mi-a stârnit curiozitatea și m-a determinat să încerc să răspund la următoarea întrebare: este cu adevărat realizabil un sistem de chatbot eficient și complet bazat pe text, fără alt tip de input?

În plus, sunt convins că utilizarea tehnologiei cloud în dezvoltarea acestui chatbot nu va îmbunătăți doar scalabilitatea și disponibilitatea sistemului, dar va permite și implementarea mai ușoară a unor funcționalități avansate de AI. Prin utilizarea serviciului Dialogflow, mă aștept să pot dezvolta un chatbot capabil să îmbunătățească în mod semnificativ interacțiunea dintre bănci și clienții lor.

Așadar, am ales această temă de licență deoarece îmi oferă oportunitatea de a explora aceste idei în detaliu și de a contribui la dezvoltarea soluțiilor tehnologice în domeniul bancar, dar și pentru a-mi deschide și alte porți cu privire la înțelegerea legăturilor dintre servicii cloud independente.

I.5 Scurt istoric al integrării asistenților virtuali

Introducerea asistenților virtuali în sistemele bancare din România a reprezentat un pas semnificativ în modernizarea și digitalizarea serviciilor financiare. Cu o populație tot mai conectată la tehnologie și cu un sector financiar dinamic și inovator, România se aliniază la tendințele globale, îmbrățișând avantajele oferite de inteligența artificială și de tehnologia cloud.

Primii pași în această direcție s-au făcut în ultimii ani, mai exact în 2017, când băncile românești au început să recunoască nevoia de a oferi servicii mai eficiente și mai personalizate clienților lor. Prima bancă românească care a introdus un asistent virtual a fost Banca Transilvania, ea lansând chatbot-ul *Livia* [18]. Confruntate cu o concurență acerbă și cu așteptările tot mai mari ale clienților, instituțiile bancare au început să exploreze diferite soluții tehnologice.

În acest context, asistenții virtuali au apărut ca un instrument promițător pentru îmbunătățirea expe-

rienței clientului. Următorul pas pentru Banca Transilvania a fost să migreze în 2019 către platforma Druid, platformă ce se ocupă de dezvoltarea chatbotilor conversaționali. Aceasta tendință a fost urmată și de către BCR, ADA fiind construită tot pe Druid [11].

Asistenții virtuali în serviciile bancare au rolul de a fluidiza interacțiunile cu clienții și de a îmbunătăți calitatea serviciilor. Ei pot răspunde în timp real la o gamă largă de întrebări, pot efectua tranzacții simple în numele clienților și pot oferi asistență personalizată, reducând astfel timpul de așteptare și îmbunătățind satisfacția clientului.

Implementarea asistenților virtuali nu a fost fără provocări. Pe lângă dificultățile tehnice, a fost necesară depășirea reticenței unor clienți de a schimba modul în care interacționau cu orice serviciu bancar, fiind în natura umană de a încerca să nu ieși din zona ta de comfort. În plus, au fost necesare eforturi considerabile pentru a asigura securitatea datelor și pentru a respecta reglementările privind confidențialitatea.

Cu toate acestea, beneficiile pe care le aduc chatbotii în sistemul bancar sunt semnificative. În primul rând, ei pot funcționa non-stop, asigurând asistență clienților în orice moment al zilei sau al nopții. În al doilea rând, pot gestiona un volum mare de solicitări simultan, ceea ce este dificil de realizat pentru angajații umani. În al treilea rând, folosirea chatbotilor poate reduce costurile operaționale, întrucât necesită mai puține resurse umane.

Deși este încă la început, implementarea asistenților virtuali în sistemele bancare din România arată promițător. Cu un nivel din ce în ce mai mare de acceptare și cu progresele continue în tehnologia AI, chatbotii vor ajunge să joace un rol tot mai important în transformarea digitală a sectorului bancar românesc și nu numai.

I.6 Structura lucrării

Lucrarea este structurată în următoarele capitole:

1. **Introducere**: prezintă o viziune generală asupra conținutului lucrării, punând accent pe tendința modernă de mutare a procesării în cloud cât și automatizarea proceselor prin chatboti conversaționali.
2. **Preliminarii**: explică elementele cheie ale lucrării și subdomeniului de care aceasta aparține.
3. **Decizii arhitecturale**: descrie modul de funcționare al platformei alese (Google Cloud Platform) cât și beneficiile utilitarului POCO față de alte librării.
4. **Setup și compilare**: explică procesul de configurare și compilare al aplicației.
5. **Detalierea implementării**: explică design-ul cât și funcționalitatea aplicației.

6. **Concluzii**: această secțiune sintetizează informațiile discutate și propune posibile direcții de dezvoltare pentru viitor.

Capitolul II

Preliminarii

II.1 Noțiuni de bază

Înainte de a discuta detaliile specifice acestei lucrări, trebuie descrise câteva noțiuni fundamentale care stau la baza dezvoltării și implementării unui asistent virtual în domeniul bancar.

- **Chatbot**

  Chatbotul este un program software ce simulează o conversație umană. În mod tradițional, utilizatorii interacționează cu chatbotii prin intermediul unei interfețe de tip text, deși unii chatboti moderni pot interacționa și prin comenzi vocale.

  Există două tipuri principale de chatboti: chatbotii bazati pe reguli și chatbotii bazati pe învățarea automată. Chatbotii bazați pe reguli sunt proiectați pentru a răspunde la întrebări specifice și sunt programați cu un set de reguli predefinite. Pe de altă parte, chatbotii bazați pe învățare automată se bazează pe algoritmi de inteligență artificială și sunt capabili să învețe din interacțiunile cu utilizatorii.

  În industria bancară, chatbotii sunt utilizați în special în serviciul de asistență pentru clienți, unde pot răspunde la întrebări frecvente, pot ajuta la rezolvarea problemelor și pot ghida utilizatorii prin diferite procese.

- **Tehnologia Cloud**

  Tehnologia cloud se referă la livrarea de servicii IT prin internet, în loc de a folosi infrastructura fizică locală. Aceasta poate include servicii de calcul, stocare, baze de date, rețelistică, software, analize și inteligență.

  Există trei tipuri principale de servicii cloud: Infrastructură ca Serviciu (IaaS), Platformă ca Serviciu

(PaaS) și Software ca Serviciu (SaaS). În industria bancară, cloud computing-ul poate ajuta la reducerea costurilor, la îmbunătățirea eficienței operaționale și la scalabilitate.

- **Inteligența artificială**

  Inteligența Artificială (AI) se referă la simularea inteligenței umane de către mașini, în special sistemele informatice. Sarcinile AI pot include învățarea (abilitatea de a dobândi și aplica cunoștințe și abilități), raționamentul (utilizarea de reguli pentru a ajunge la concluzii aproximative sau definite), auto-corectarea și procesarea limbajului natural.

  Chatbotii bazați pe AI, cum ar fi cei dezvoltați cu Dialogflow, pot învăța din interacțiuni și pot îmbunătăți continuu calitatea conversațiilor pe care le au cu utilizatorii.

- **Dialogflow**

  Dialogflow, deținut de Google, este o platformă care permite dezvoltatorilor să creeze interfețe de conversație pentru site-uri web, aplicații mobile și platforme populare de mesagerie [16]. Dialogflow utilizează tehnologia de înțelegere a limbajului natural (NLU - *Natural Language Understanding*) pentru a înțelege și a procesa limbajul uman.

  În contextul dezvoltării chatbotilor pentru instituțiile bancare, Dialogflow poate ajuta la crearea de boti care să înțeleagă și să răspundă la cererile utilizatorilor într-un mod mai natural și mai intuitiv.

II.2 Stadiul actual al subdomeniului

Implementarea chatbotilor în sistemul bancar reprezintă o inovație semnificativă în ceea ce privește interacționarea băncilor cu clienții lor. Începând cu Banca Transilvania în 2017, multe alte instituții bancare din România au urmat exemplul și au început să folosească chatboti pentru a îmbunătăți serviciile oferite clienților.

Chatbotii oferă un nivel de disponibilitate non-stop, care este deosebit de valoros într-o industrie precum cea bancară, care necesită suport pentru clienți la orice oră. Prin eliminarea nevoii de interacțiune umană pentru a rezolva probleme comune, chatbotii permit băncilor să economisească resurse și să se concentreze pe probleme mai complexe. Acești asistenți virtuali pot răspunde rapid la întrebări, pot rezolva probleme și pot oferi asistență în tranzacții financiare, îmbunătățind astfel satisfacția generală a clienților.

Cu toate acestea, în ciuda progreselor semnificative în adoptarea tehnologiei chatbot în industria bancară, există încă multe oportunități de explorare și îmbunătățire. Un domeniu cheie de cercetare este îmbunătățirea înțelegerii naturale a limbajului de către chatboti. Deși chatbotii actuali sunt capabili să gestioneze o serie de interacțiuni de bază, abilitatea lor de a înțelege nuanțele și complexitatea limbajului uman este limitată.

Un alt domeniu de potențială îmbunătățire este personalizarea și adaptabilitatea. În prezent, majoritatea chatbotilor bancari folosesc algoritmi pre-programați pentru a răspunde la interacțiunile utilizatorilor. Cu toate acestea, există oportunități semnificative pentru utilizarea tehnologiilor avansate de AI, cum ar fi învățarea profundă (deep learning), pentru a dezvolta chatboti care pot învăța și se pot adapta la comportamentul și preferințele individuale ale utilizatorilor.

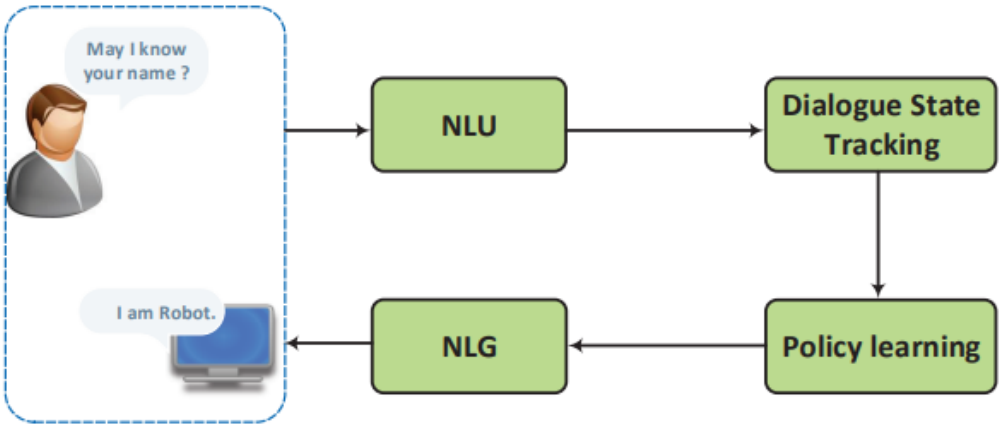

Figura II.1: Pipeline general pentru sisteme orientate pe task-uri [9].

În plus, există încă multe provocări legate de securitate și confidențialitate care trebuie abordate înainte ca chatbotii să fie adoptați pe scară largă în industria bancară. Acestea includ protejarea datelor personale și financiare ale utilizatorilor și asigurarea că chatbotii nu pot fi utilizați pentru activități frauduloase.

Așadar, în timp ce adoptarea chatbotilor în sistemul bancar a presupus progrese remarcabile în ultimii ani, există încă multe oportunități de cercetare și dezvoltare în acest domeniu, precum și probleme care trebuie adresate. Pe măsură ce tehnologia continuă să avanseze, este de așteptat ca chatbotii să devină un element tot mai integral al serviciilor bancare.

II.3 Obiectivele lucrării în contextul subdomeniului

Obiectivul principal al acestei lucrări este de a investiga posibilitățile pe care tehnologia chatbot și serviciile cloud le oferă în contextul sistemului bancar. În particular, accentul este pus pe implementarea unui model de chatbot bazat pe platforma Dialogflow, cu suport pentru limbajul de programare C++. C++ este un limbaj de programare popular, cunoscut pentru eficiența și puterea sa, fiind astfel un candidat potrivit pentru dezvoltarea unei platforme ce va ține un agent.

În actualul stadiu de dezvoltare a tehnologiei chatbot, această lucrare vizează să prezinte și să contribuie la progresul subdomeniului, prin oferirea unei viziuni detaliate asupra procesului de implementare a unui chatbot în serviciile bancare.

În plus, lucrarea explorează și potențialele căi de îmbunătățire și adaptare a chatbotilor. O astfel de direcție ar putea fi folosirea de tehnologii avansate de AI, cum ar fi învățarea profundă, pentru a dezvolta chatboti care pot învăța și se pot adapta la comportamentul și preferințele individuale ale utilizatorilor. Prin urmare, această lucrare își propune să deschidă calea către o utilizare mai extinsă și mai eficientă a tehnologiei chatbot în industria bancară.

De asemenea, un alt aspect pe care se concentrează prezenta lucrare este cloud computing-ului, o tehnologie care a revoluționat modul în care datele și serviciile sunt gestionate. Cu ajutorul cloud computing-ului, chatbotii pot fi ușor scalabili, ceea ce înseamnă că pot servi un număr mare de utilizatori simultan, fără a compromite performanța. În plus, cloud computing-ul facilitează actualizările și îmbunătățirile continuu, fără întreruperi majore ale serviciului [12].

În concluzie, această lucrare are ca scop principal să demonstreze cum tehnologiile moderne, precum chatbotii și cloud computing-ul, pot transforma modul în care băncile interacționează cu clienții lor. Se urmărește evidențierea beneficiilor potențiale ale utilizării agenților conversaționali pentru optimizarea proceselor și a serviciilor bancare, în speranța de a stimula o adoptare mai largă a acestor tehnologii în industria bancară.

Capitolul III

Decizii arhitecturale

III.1 Ecosistemul Google Cloud

III.1.1 Alegerea platformei agentului

Comparând Dialogflow cu alte servicii de agenți conversaționali, există mai multe caracteristici care îl diferențiază și îl recomandă pentru utilizarea în sectorul bancar.

Dialogflow, deținut de Google, este o platformă avansată pentru dezvoltarea de aplicații de conversație, care folosește tehnologia AI pentru a interpreta intențiile și contextul utilizatorului [7]. Aceasta oferă o gamă largă de funcționalități, inclusiv integrarea cu diverse platforme de mesagerie, asistenți virtuali și alte servicii Google, cum ar fi Google Cloud Functions.

La rândul lor, serviciile alternative, cum ar fi IBM Watson, Amazon Lex și Microsoft Luis, prezintă și ele avantaje. IBM Watson se remarcă prin puterea sa de a învăța în mod continuu și de a se adapta la diverse contexte de utilizare [10]. Amazon Lex beneficiază de integrarea nativă cu ecosistemul Amazon Web Services (AWS), oferind posibilități extinse de dezvoltare și scalare [1]. În cele din urmă, Microsoft Luis are avantajul integrării strânse cu suita de produse Microsoft, care includ Office 365 și Azure [13].

Cu toate acestea, Dialogflow se distinge prin mai multe aspecte-cheie. În primul rând, Dialogflow este foarte flexibil, permițând dezvoltatorilor să creeze experiențe de conversație personalizate pentru diferite platforme și canale de comunicare. Acesta poate fi integrat cu o multitudine de servicii, de la Google Assistant și Amazon Alexa, până la Facebook Messenger și Slack.

În al doilea rând, Dialogflow folosește servicii integrate în ecosistemul Google Cloud. Astfel, permite dezvoltatorilor să creeze, să testeze și să implementeze chatboti direct în cloud, profitând de avantajele acestuia.

Aici intervine Google Cloud Functions [3], un serviciu de calcul care permite dezvoltatorilor să execute cod ca răspuns la evenimente specifice, fără a fi nevoie să administreze o infrastructură de server. Acest serviciu poate fi utilizat în tandem cu Dialogflow pentru a crea funcții de backend pentru chatbot, cum ar fi procesarea cererilor utilizatorului, integrarea cu alte sisteme sau baze de date, sau gestionarea autentificării și a securității.

Google Cloud Functions se integrează perfect cu Google Source Repositories [17], un serviciu de găzduire de cod sursă care oferă un loc sigur și scalabil pentru a stoca și a gestiona codul. Acest lucru permite dezvoltatorilor să colaboreze eficient la proiecte, să gestioneze versiunile de cod și să implementeze automat codul în Cloud Functions.

În final, Google Cloud Storage oferă un serviciu de stocare de obiecte scalabil și durabil, care poate fi utilizat pentru a stoca și a servi datele utilizate de chatbot, cum ar fi înregistrări de conversații, profile de utilizator, sau alte date de context [4].

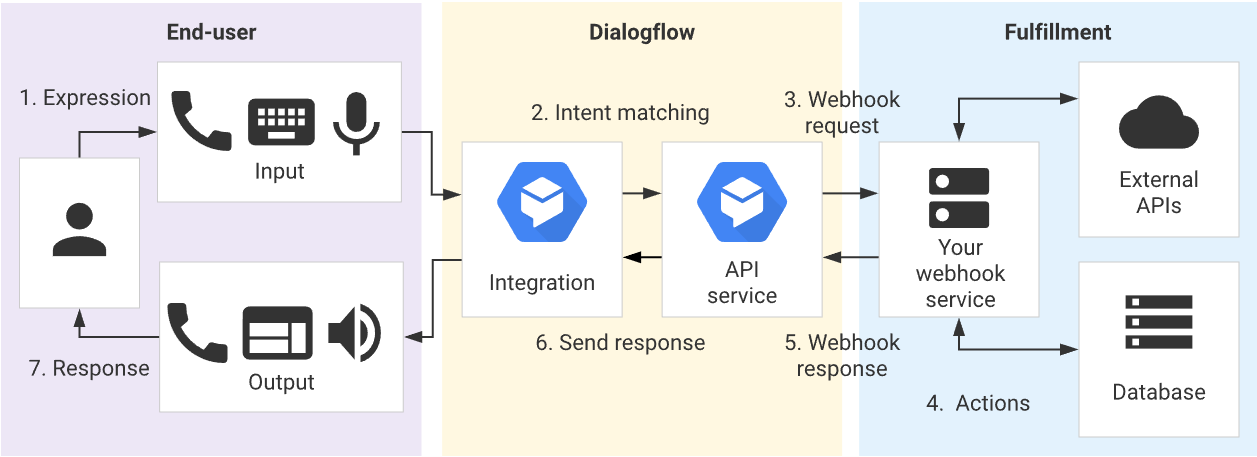

Figura III.1: Flow-ul intern al Dialogflow [7].

În ansamblu, alegerea Dialogflow, împreună cu Google Cloud Functions, Google Source Repositories și Google Cloud Storage, oferă o soluție robustă și flexibilă pentru dezvoltarea de chatboti în sectorul bancar. Prin folosirea acestor tehnologii, băncile pot crea experiențe de conversație personalizate, eficiente și securizate pentru clienții lor.

III.1.2 Cum funcționează ecosistemul

Dialogflow folosește input-ul trimis de către Dialogflow API C++ Client [6], acesta fiind parsat și trecut prin verificarea lor internă cu intențiile create în prealabil[1]. În funcție de cum este creat intent-ul și scopul

[1]O intenție reprezintă un anumit rezultat pe care doriți să îl obțineți de la interacțiunea utilizatorului. De exemplu, o intenție poate fi „programare întâlnire” sau „informații despre cont”.

său, pot exista parametrii scoși sub formă de entități[2] din textul primit (sau este un intent default, cu rol de legătură între altele cu functionalități).

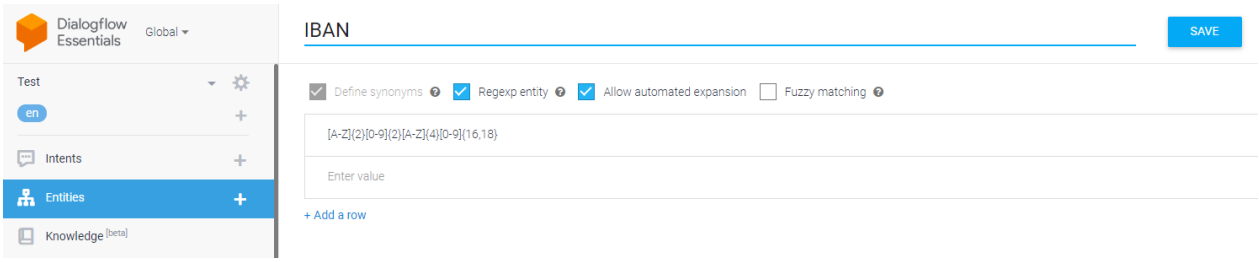

Figura III.2: Reprezentarea unei entități pentru IBAN folosind un regex.

În Dialogflow, o ”intenție” reprezintă un anumit rezultat pe care îl doriți de la o interacțiune cu utilizatorul. Atunci când un utilizator trimite un input (cum ar fi o întrebare sau o comandă), Dialogflow potrivește inputul cu cea mai bună intenție pe baza a ceea ce ați setat în agentul dvs. În cadrul unei intenții, există mai multe câmpuri și concepte cheie care sunt utilizate pentru a defini și a rafina comportamentul intenției.

Contextele permit agentului Dialogflow să înțeleagă starea conversației și să răspundă în mod corespunzător. Există două tipuri de contexte: contexte de intrare și contexte de ieșire. Contextele de intrare sunt cele pe care agentul le caută înainte de a potrivi o intenție, în timp ce contextele de ieșire sunt stabilite după ce o intenție este potrivită.

Exemple de declarații ale utilizatorului sunt exemple de ceea ce utilizatorul ar putea spune pentru a declanșa această intenție (vezi Figura III.3). Sistemul utilizează aceste exemple pentru a învăța modelul de limbaj și pentru a recunoaște aceeași intenție din declarații diferite.

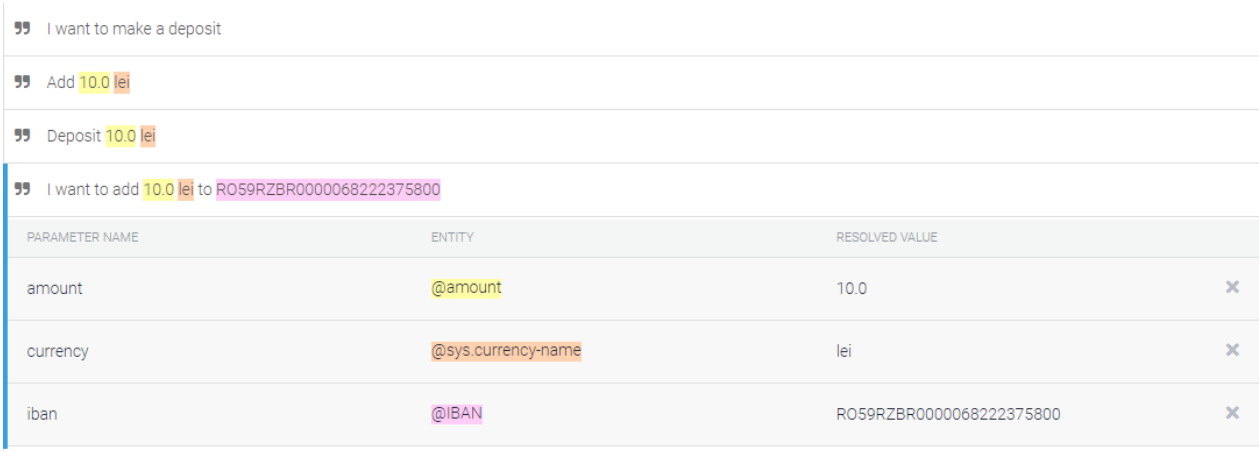

Figura III.3: Frazele de antrenament ale botului.

Răspunsurile reprezintă modul de interacționare hard-codată a agentului cu inputul clientului. Astfel,

[2]Entitățile sunt concepte valoroase care pot fi extrase din declarațiile utilizatorilor. De exemplu, în declarația „Doresc să programez o întâlnire pentru marți”, „marți” este o entitate de tip „dată”.

un răspuns toate fi considerat fie final de conversație, fie poate avea un răspuns default, folosind funcționalitatea webhook-ului ulterior.

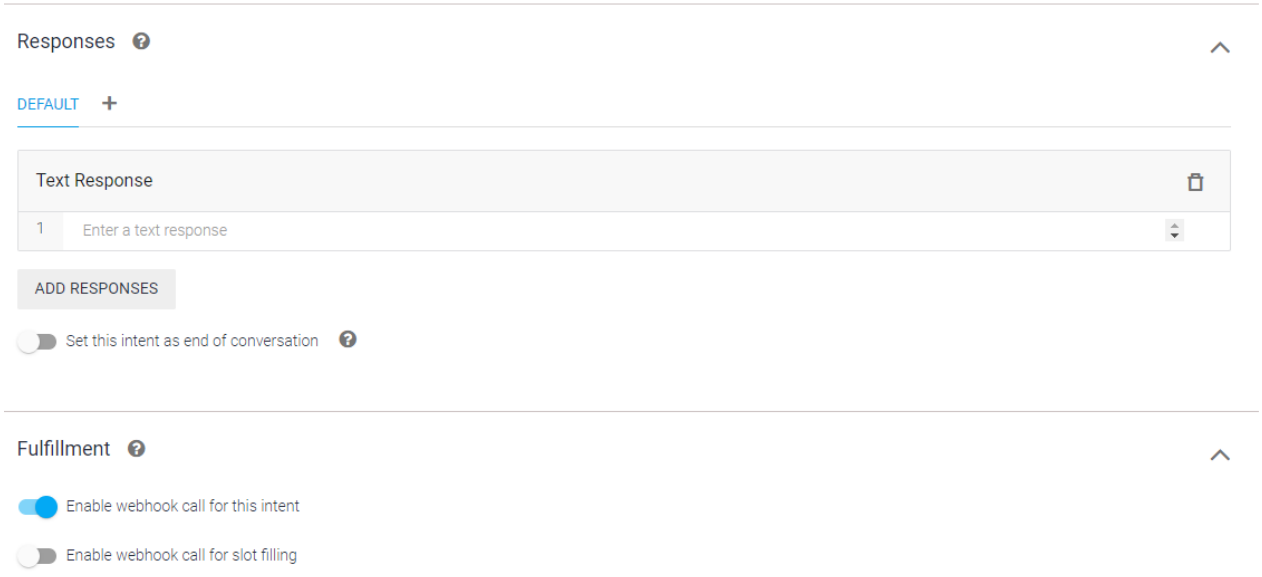

Figura III.4: Selectarea fulfillment-ului.

În cadrul platformei există niște entități predefinite [8] pentru a ușura munca utilizatorului atunci când vine vorba de extragere de parametrii, dar funcționalitatea puternică al acestei opțiuni este extragerea parametrizată a informațiilor. Dupa cum putea vedea in Figura III.5, acțiunea propriu-zisă oferă flexibilitate atât la prezența parametrului (daca acesta se poate scoate din text), cât și la un text default dacă acesta nu a fost găsit. Coloana *value* reprezintă numele variabilei care este trimisă către metoda din Google Cloud Function, prin webhook-ul definit în tabul de *fulfillment*.

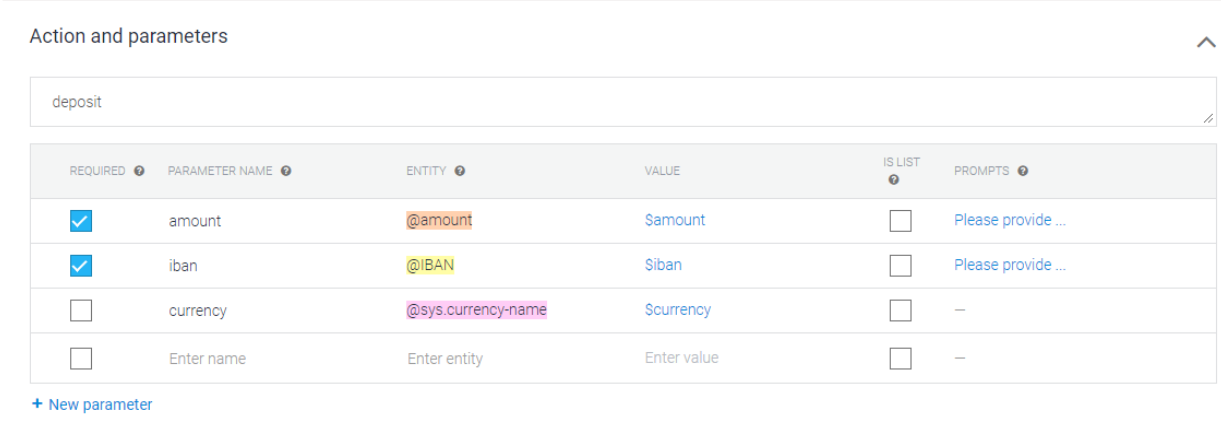

Figura III.5: Extragerea de parametrii din inputul clientului.

În tab-ul de fulfillment putem găsi modul de procesare al informației extrase de către agent. În cazul de

față, am folosit un webhook extern, care folosește integrarea cu Google Cloud Function. Astfel, parametrii scoși vor fi trimiși sub forma unui request către acel URL. Detaliile acelui request se află în Figura III.6.


```
{
  "responseId": "9f9b9222-2da7-4fb2-a68e-73b76a01d3e7-12c5f707",
  "queryResult": {
    "queryText": "check balance for check balance for RO59RZBR0000068222375800",
    "parameters": {
      "iban": "RO59RZBR0000068222375800"
    },
    "allRequiredParamsPresent": true,
    "fulfillmentText": "The account with IBAN 'RO59RZBR0000068222375800' has a balance of: 74.79 RON.",
    "fulfillmentMessages": [
      {
        "text": {
          "text": [
            "The account with IBAN 'RO59RZBR0000068222375800' has a balance of: 74.79 RON."
          ]
        }
      }
    ],
    "intent": {
      "name": "projects/licenta-383311/agent/intents/5dc70a8c-b7a7-4839-9646-5c852453e390",
      "displayName": "Check Balance"
    },
    "intentDetectionConfidence": 0.90945506,
    "diagnosticInfo": {
      "webhook_latency_ms": 2033
    },
    "languageCode": "en",
    "sentimentAnalysisResult": {
      "queryTextSentiment": {
        "score": -0.2,
        "magnitude": 0.2
      }
    }
  },
  "webhookStatus": {
    "message": "Webhook execution successful"
  }
}
```


Figura III.6: Exemplu de request trimis către Google Cloud Function.

Explicațiile parametrului pentru o mai bună înțelegere a modului de transmitere a informației între seriviciile Google Cloud Platform:

1. responseId: este un identificator unic pentru fiecare interacțiune cu Dialogflow.
2. queryResult: este secțiunea care conține majoritatea informațiilor legate de interacțiunea cu utilizatorul.

   queryText: este întrebarea sau afirmația pe care utilizatorul a transmis-o.

   parameters: sunt parametrii extrasi din întrebarea utilizatorului. În acest caz, ”iban” este un parametru și valoarea sa este ”RO59RZBR0000068222375800”.

   allRequiredParamsPresent: acest câmp indică dacă toți parametrii necesari pentru intenție sunt prezenți. În acest caz, este adevărat, ceea ce înseamnă că toți parametrii necesari sunt prezenți.

fulfillmentText: este textul care va fi returnat utilizatorului ca răspuns la întrebarea sa.

fulfillmentMessages: este o listă de mesaje care vor fi trimise înapoi utilizatorului. În acest caz, este doar un mesaj, care este același cu fulfillmentText.

intent: este intenția care a fost potrivită pentru întrebarea utilizatorului.

intentDetectionConfidence: este gradul de încredere cu care Dialogflow a potrivit intenția. În acest caz, este de aproximativ 90.

diagnosticInfo: sunt informații suplimentare privind interacțiunea. În acest caz, indică timpul de latență pentru webhook.

languageCode: este codul de limbă al interogării utilizatorului.

sentimentAnalysisResult: este analiza sentimentului textului interogării. Scorul indică sentimentul general (pozitiv sau negativ), iar magnitudinea indică intensitatea sentimentului.

3. webhookStatus: este statusul execuției webhook-ului, care în acest caz a fost reușită.

Tabul de "Validation" din Dialogflow ES are de-a face cu revizuirea și aprobarea sau respingerea propunerilor pe care Dialogflow le face pentru îmbunătățirea modelului asistentului virtual. Acesta este un element al învățării automate interactive, unde sistemul învață din feedbackul dat de utilizator.

După ce asistentul virtual a fost folosit o vreme, va începe să învețe din interacțiunile cu utilizatorii și va încerca să îmbunătățească precizia detecției intențiilor și a entităților. Acest lucru este realizat prin generarea de "sugestii" pe baza interacțiunilor anterioare. Aceste sugestii apar în tabul "Validation".

Fiecare sugestie conține următoarele elemente:

Training phrase - Este textul interacțiunii dintre utilizator și asistentul virtual.

Intent - Este numele intenției pe care Dialogflow o propune pentru fraza de instruire. Acesta poate fi o intenție existentă sau o nouă intenție sugerată de Dialogflow.

Action - Este un câmp opțional, care permite asocierea unei acțiuni cu o intenție. Dialogflow poate sugera o acțiune pe baza contextului frazei de instruire.

Entities - Sunt entitățile pe care Dialogflow le propune pentru a fi asociate cu fraza de instruire.

Fiecare sugestie poate fi aprobată sau respinsă, în funcție de dorință (vezi Figura III.7). Dacă sugestia este aprobată, Dialogflow va actualiza modelul asistentului virtual pentru a include informațiile din sugestie, îmbunătățind astfel precizia detecției de intenții și entități în interacțiunile viitoare. Dacă sugestia este respinsă, Dialogflow nu va face nicio modificare.

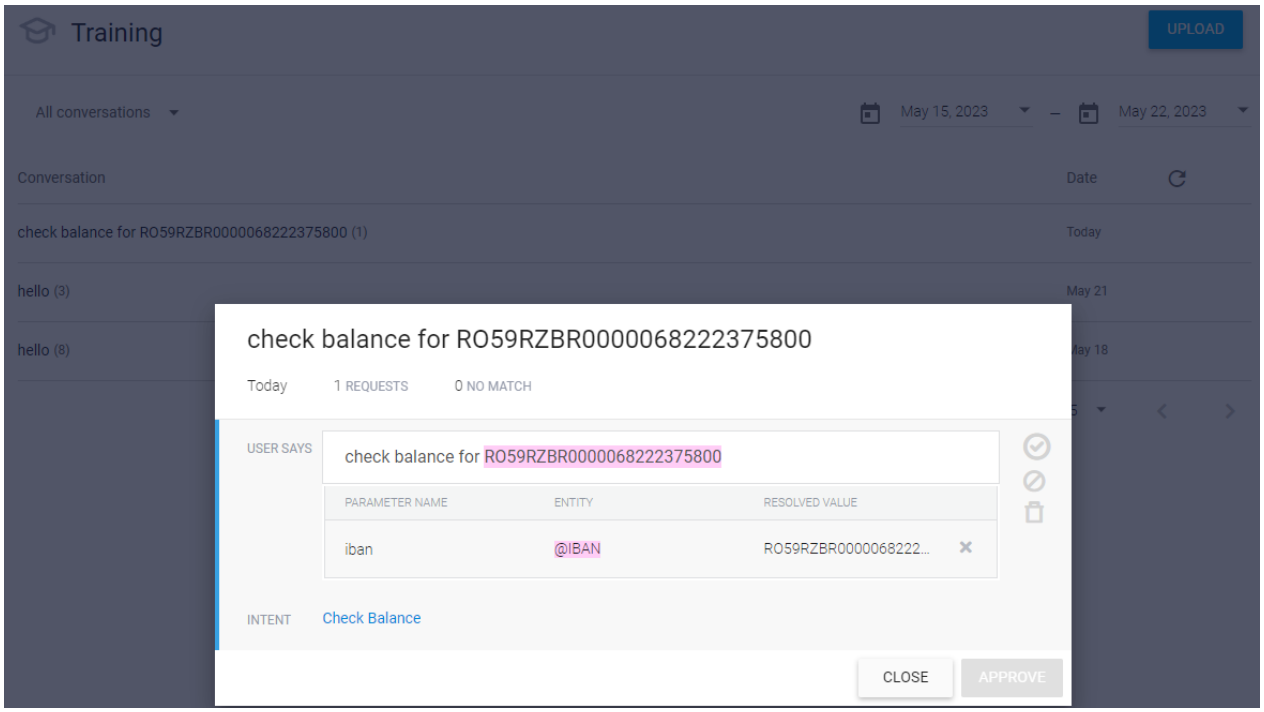

Figura III.7: Validarea rezultatului agentului.

III.2 Alegerea utilitarului

Biblioteca Portable Components (POCO) este un set popular de biblioteci open-source pentru dezvoltarea de software în C++. Se evidențiază printr-o gamă largă de facilități care îi permit să fie o soluție potrivită pentru dezvoltarea unei aplicații web. Înainte de a trece la detaliile bibliotecii POCO, este esențial să se înțeleagă contextul general al instrumentelor și bibliotecilor disponibile pentru dezvoltarea aplicațiilor web în C++.

Din punct de vedere istoric, dezvoltarea aplicațiilor web în C++ a fost deseori privită ca un proces complex, în mare parte din cauza faptului că limbajul C++ este orientat spre dezvoltarea software de nivel înalt, cu un control detaliat asupra resurselor hardware, precum și a complexității implicite a acestui limbaj. Cu toate acestea, cu ajutorul unor instrumente precum bibliotecile POCO [15], Boost [2], Wt [19] și CPPCMS [5], dezvoltarea aplicațiilor web în C++ a devenit mult mai ușoară și mai eficientă.

III.2.1 POCO: avantaje și caracteristici

POCO se distinge prin designul său robust și portabil, care permite dezvoltatorilor să creeze software eficient și de înaltă performanță, care poate fi portat cu ușurință pe diferite platforme și sisteme de operare. Bibliotecile POCO au fost concepute pentru a fi ușor de înțeles și de utilizat, fără a impune niciun model de programare specific. Acest lucru le face flexibile și adaptabile, încurajând dezvoltarea rapidă și agile.

În contextul dezvoltării aplicațiilor web, biblioteca POCO oferă numeroase facilități, inclusiv suport pentru HTTP, FTP, sendmail, URI, cookie-uri, HTML formular și altele. Acesta include, de asemenea, un framework pentru servere web, care permite dezvoltarea serverelor web multithreaded și sigure. De asemenea, biblioteca NetSSL furnizează suport pentru comunicațiile securizate SSL/TLS, care sunt esențiale pentru orice aplicație web modernă.

III.2.2 POCO în comparație cu alte biblioteci C++

Să comparăm acum POCO cu alte biblioteci C++ populare pentru dezvoltarea aplicațiilor web: Boost, Wt și CPPCMS.

Boost este o colecție vastă de biblioteci C++, care oferă o gamă largă de facilități. Cu toate acestea, Boost nu este specializat în dezvoltarea aplicațiilor web și nu include suport direct pentru HTTP sau alte protocoale web specifice. Prin urmare, dezvoltarea unei aplicații web cu Boost ar necesita o cantitate semnificativă de cod adițional sau utilizarea altor biblioteci pentru a acoperi aceste funcționalități.

Wt este o bibliotecă C++ pentru dezvoltarea aplicațiilor web care utilizează un model de programare similar cu cel al Qt. Deși Wt este puternic și flexibil, are un model de programare mai complex decât POCO și poate fi mai dificil de învățat și de utilizat, în special pentru cei care nu sunt familiarizați cu Qt.

CPPCMS este un framework specializat pentru dezvoltarea aplicațiilor web în C++. Deși CPPCMS include o serie de facilități puternice pentru dezvoltarea aplicațiilor web, el impune un anumit model de programare, care poate fi mai restrictiv decât abordarea mai flexibilă a POCO.

Prin urmare, POCO oferă un echilibru excelent între ușurința de utilizare, flexibilitate, portabilitate și performanță, ceea ce o face o alegere bună pentru dezvoltarea aplicațiilor web în C++. Cu toate acestea, alegerea finală ar trebui să depindă de cerințele specifice ale proiectului, de preferințele dezvoltatorului și de alte considerații.

Capitolul IV

Setup și compilare

IV.1 Instalarea bibliotecilor

Utilizatorul trebuie să aibă o cheie de acces pentru un cont de serviciu pe platforma Google Cloud pentru a accesa API-ul Dialogflow ES. Această cheie trebuie plasată sub directorul *res*. Mai multe informații găsiți aici.

Această cheie va fi setată ca variabilă de mediu `GOOGLE_APPLICATION_CREDENTIALS`, care va funcționa ca un sistem de autentificare pentru platformă.

În directorul rădăcină al proiectului, se află un script Bash numit `getting_started.sh` (Notă: dacă scriptul nu poate fi rulat pe o platformă Unix, se recomandă verificarea formatului greșit al sfârșitului de linie prin executarea comenzii `dos2unix` pentru script și verificarea din nou). Când este rulat pe o platformă Unix, acesta va instala toate dependențele necesare (vezi Configurarea inițială). După finalizarea scriptului, utilizatorul trebuie să ruleze manual configurarea inițială pentru CLI-ul GC:

```
./google-cloud-sdk/install.sh
source ~/.bashrc
./google-cloud-sdk/bin/gcloud init
```

Notă: Configurarea inițială va dura un timp **îndelungat** (în special pe mașini mai vechi).

IV.2 Configurarea inițială

Fluxul scriptului `getting_started.sh` este următorul:

Notă importantă: Trebuie să fie rulat cu opțiunea `-i` sau `--install`, iar după finalizarea acestuia, utilizatorul trebuie să ruleze manual configurarea inițială pentru gcloud (vezi pașii de mai sus).

1. Configurarea mediului:
   (a) Submodulele sunt inițializate.
   (b) Pe baza diferenței în terminarea greșită a liniei, atât scriptul `build.sh`, cât și executabilul `sass` vor fi convertite la formatul Unix-like.
   (c) Se instalează `doxygen` și dependența sa `graphviz`.
2. Revenirea la versiunile dorite ale bibliotecilor:
   (a) Se utilizează versiunea `POCO v1.12.4`.
   (b) Se utilizează versiunea `VCPKG v2023.04.15 Release`.
   (c) Se utilizează versiunea `GCP v2.9.1`.
3. Configurarea POCO. Mai multe informații găsiți aici:
   (a) Se instalează dependențele (`openssl, libssl-dev, git, g++, make, cmake`).
   (b) Se configurează fișierul `CMakeLists.txt`.
   (c) Se construiește fișierul de configurare.
   (d) Se instalează biblioteca.
4. Configurarea npm:
   (a) Se instalează `npm` prin intermediul gestionarului de pachete implicit.
5. Configurarea Google Cloud CLI. Mai multe informații găsiți aici:
   (a) Se descarcă arhiva care conține codul sursă.
   (b) Se extrage arhiva.
6. Configurarea Sass:
   (a) Se creează un link simbolic (`symlink`) și se forțează crearea acestuia, chiar dacă există, către executabilul furnizat (vezi directorul `libs/sass`).
7. Configurarea VCPKG. Mai multe informații găsiți aici:
   (a) Se instalează dependențele (`curl, zip, unzip, tar`).

(b) Se configurează managerul vcpkg pentru prima dată prin intermediul scriptului de pornire (`libs/vcpkg/bootstrap-vcpkg.sh`).

8. Configurarea Google Cloud Platform. Mai multe informații găsiți aici:

   (a) Se instalează modulele reale pe care le vom utiliza în proiect (`core` și `dialogflow-es`) prin intermediul vcpkg.

IV.3 Cum se construiește

Din directorul rădăcină al proiectului, putem utiliza scriptul `build.sh` pentru a construi și rula aplicația. În mod implicit, serverul va utiliza portul **9090**, dar acesta poate fi modificat din fișierul `constants.h` (vezi `globals::serverPort`).

Există câteva opțiuni care pot fi utilizate cu scriptul:

- `-b` sau `--build` => compilează și rulează proiectul
- `-bscss` sau `--build-scss` => compilează doar fișierele SCSS pentru a actualiza aspectul HTML-ului
- `-docs` sau `--generate-docs` => generează documentația tehnică utilizând Doxygen
- `-c` sau `--clean` => curăță proiectul
- `-v` sau `--verbose` => opțional, afișează mai multe informații în consolă
- `-h` sau `--help` => afișează această informație de ajutor

Notă: Nu se recomandă rularea opțiunii `-c` împreună cu alte opțiuni, deoarece curățarea trebuie să fie un proces singular.

Capitolul V

Detalierea implementării

V.1 Script-uri auxiliare

Ambele scripturi folosesc același cod pentru prelucrarea inputului utilizatorului și tratarea erorilor.

Acestea sunt procesate de funcția `func_read_cli_options` și sunt verificate cu lista implicită (care diferă în mare parte pentru ambele scripturi, deoarece au roluri distincte, dar opțiunile `-v` și `-h` sunt comune, deoarece au o utilitate comună).

- Flaguri comune:
  - Flagul `verbose` permite scripturilor să afișeze mai multe informații (atât de la ele, cât și din jurnalele aplicației) în consolă. Această opțiune va seta variabila globală `VERBOSE` la true.
  - Flagul `help` va afișa doar informațiile de ajutor.
- Flagurile specifice pentru `getting_started.sh`:
  - Flagul `install` instalează și configurează toate dependențele pentru proiect.
- Flagurile specifice pentru `build.sh`:
  - Flagul `build` va compila și executa proiectul.
  - Flagul `build-scss` va compila doar fișierele SCSS pentru a actualiza aspectul HTML-ului.
  - Flagul `generate-docs` va genera documentația tehnică utilizând Doxygen.
  - Flagul `clean` va curăța proiectul de fișiere temporare, jurnale, executabile, fișiere binare, etc.

Dacă Flagul există în lista dorită, variabila `COMMAND` stochează acțiunea dorită și apoi este pasată funcției `execute`.

Funcția `execute` va apela apoi metoda corespunzătoare funcționalității dorite. Acest nivel de abstractizare permite concatenarea mai multor comenzi sub același nume, dacă este necesar.

Notă: Din acest moment înainte, se va analiza fiecare script în mod individual, deoarece au metode diferite pentru cazurile lor de utilizare diferite.

Iată codul LaTeX actualizat, utilizând listele pentru enumerare:

V.1.1 Getting Started

Pentru `getting_started.sh`:

- Metoda `execute` poate apela doar funcția `install`. Această metodă va parcurge o listă de comenzi și va instala dependențele necesare (vezi Configurarea inițială).

V.1.2 Build

Pentru `build.sh`:

- Metoda `execute` poate apela una dintre următoarele funcții:
  - Funcția `compile_sass` va itera peste toate fișierele SCSS din subfolderul `res/scss` și le va compila, generând fișierele CSS dorite în subfolderul `res/css`. Aceste fișiere nu vor conține hărți CSS generate automat, deoarece nu există mult CSS pentru început și nu este necesară crearea unei logici suplimentare pentru ca serverul să gestioneze aceste solicitări.
  - Funcția `generate_docs` va genera documentația tehnică. În fișierul Doxyfile, următoarele taguri au fost modificate:
    * `PROJECT_NAME = "Licență"`
    * `PROJECT_BRIEF = "O aplicație bancară C++ care utilizează Dialogflow ES de pe Google Cloud Platform pentru interacțiuni uman-bot într-un context bancar."`
    * `OUTPUT_DIRECTORY = "./res/docs"`
    * `FULL_PATH_NAMES = NO`
    * `BUILTIN_STL_SUPPORT = YES`
    * `EXTRACT_ALL = YES`
    * `EXTRACT_PRIVATE = YES`
    * `EXTRACT_STATIC = YES`
    * `INPUT = ./res`

* RECURSIVE = YES
* HTML_OUTPUT = .
* DISABLE_INDEX = YES
* GENERATE_TREEVIEW = YES
* GENERATE_LATEX = NO
* CALL_GRAPH = YES
* CALLER_GRAPH = YES
* ALPHABETICAL_INDEX = NO

– Funcția `build` este o metodă de nivel superior care are ca scop construirea și rularea executabilului final. Executabilul se numește `licenta_EXECUTABLE` și poate fi găsit în directorul rădăcină al proiectului. Acesta poate fi rulat cu două indicatoare: `DISABLE_DEBUG` sau `ENABLE_DEBUG`. Aceste indicatoare depind de valoarea variabilei globale `VERBOSE`. Dacă este `true`, sistemul de jurnalizare al proiectului (`MyLogger` - un wrapper peste clasa `Poco's Logger`) va crea un canal `Poco::ConsoleChannel`, care va fi furnizat unui canal `Poco::SplitterChannel`. Astfel, orice jurnal se va genera, va fi afișat și în consolă, pe lângă folderul `logs`.

– 1. Funcția `build` este o metodă de nivel superior care are ca scop construirea și rularea executabilului final. Executabilul se numește `licenta_EXECUTABLE` și poate fi găsit în directorul rădăcină al proiectului. Acesta poate fi rulat cu două indicatoare: `DISABLE_DEBUG` sau `ENABLE_DEBUG`. Aceste indicatoare depind de valoarea variabilei globale `VERBOSE`. Dacă este `true`, sistemul de jurnalizare al proiectului (`MyLogger` - un wrapper peste clasa `Poco's Logger`) va crea un canal `Poco::ConsoleChannel`, care va fi furnizat unui canal `Poco::SplitterChannel`. Astfel, orice jurnal se va genera, va fi afișat și în consolă, pe lângă folderul `logs`.
  2. Se apelează funcția `compile_sass`. Consultați mai sus pentru mai multe explicații.
  3. Funcția `compile_cpp` configurează mai întâi fișierul `CMakeLists.txt` și trece variabila `CMAKE_TOOLCHAIN_FILE` la CMake-ul de pe Google Cloud Platform. Deoarece aceasta nu este utilizată direct în acest CMake, vom adăuga și indicativul `--no-warn-unused-cli` pentru a suprima avertismentul referitor la această variabilă. Dacă configurarea reușește, se va efectua construirea efectivă a proiectului.
  4. Funcția `configure_gcp` va exporta variabila `GOOGLE_APPLICATION_CREDENTIALS`, astfel încât să fie vizibilă pentru fiecare proces și subproces care pornește din script.

Proiectul CMake utilizează standardul `C++17`, iar executabilul este generat în directorul rădăcină al proiectului. Bibliotecile utilizate pentru legare sunt `Poco::Foundation Poco::Net Poco::Util google-cloud-cpp::dialogflow_es`.

V.2 Logica agentului

Codul poate fi găsit în **./cloud** și constă într-un scrip JavaScript utilizat în contextul unui serviciu de îndeplinire a cerințelor (fulfillment) pentru Dialogflow, un serviciu de înțelegere a limbajului natural și de gestionare a conversațiilor dezvoltat de Google. Acest script gestionează cererile primite de la Dialogflow și furnizează răspunsurile corespunzătoare. De asemenea acesta este rulat prin Google Cloud Function, codul fiind luat din serviciul Google Source Repository.

La nivel top-level, se regăsesc următoarele funcționalități:

1. Se importă modulele necesare: 'firebase-functions', 'dialogflow-fulfillment', 'google-cloud/storage' și 'axios'. Aceste module facilitează comunicarea cu Dialogflow, gestionarea stocării și efectuarea de cereri HTTP.

2. Se definește funcția 'dialogflowFirebaseFulfillment' ca o funcție de tip 'functions.https.onRequest', care va fi apelată în momentul în care se primește o cerere HTTP către URL-ul specificat în configurația Firebase.

3. În interiorul funcției 'dialogflowFirebaseFulfillment', se inițializează un obiect 'WebhookClient' cu request-ul și response-ul primite. Acest obiect facilitează procesarea cererii și furnizarea răspunsului către Dialogflow.

4. Se definesc funcțiile care vor fi apelate în funcție de intent-ul identificat de Dialogflow. Aceste funcții gestionează diferite scenarii de conversație și furnizează răspunsuri specifice. De exemplu, funcția 'welcome' furnizează un mesaj de bun venit, iar funcția 'checkBalance' verifică soldul unei anumite conturi bancare. Fiecare funcție primește ca parametru obiectul 'agent', care reprezintă un context de conversație și permite adăugarea de mesaje și interacțiune cu utilizatorul.

5. În cadrul funcțiilor, se folosește obiectul 'agent' pentru a adăuga mesaje de răspuns, a efectua verificări și a interacționa cu sistemul de stocare în cloud folosind modulul 'google-cloud/storage'.

6. Funcția 'getConversionRate' este utilizată pentru a obține rata de conversie între două valute folosind un serviciu extern. Această funcție face o cerere HTTP către un API și returnează rata de conversie.

7. Este creată o instanță a obiectului ‘WebhookClient‘ și se apelează funcția ‘handleRequest‘ pentru a procesa cererea și a furniza răspunsul corespunzător.

8. Este definită o hartă de intent-uri (‘intentMap‘) care asociază funcțiile definite mai sus cu intent-urile specifice. Astfel, în funcția ‘handleRequest‘, se va apela funcția corespunzătoare în funcție de intent-ul identificat de Dialogflow.

Acest script oferă suport pentru diferite funcționalități de gestionare a conversațiilor cu un chatbot, cum ar fi verificarea soldului, transferul de bani între conturi, crearea și ștergerea de conturi bancare. De asemenea, folosește un serviciu extern pentru a obține ratele de conversie valutară. Prin intermediul acestui script, se poate crea un chatbot interactiv și inteligent care să furnizeze răspunsuri personalizate și să interacționeze cu utilizatorii într-un mod natural.

V.3 Logica locală

Arhitectural vorbind, partea din aplicație care se află local reprezinta doar scheletul funcționalității principale, partea importantă fiind mutată în serviciile oferite de Google Cloud Platform.

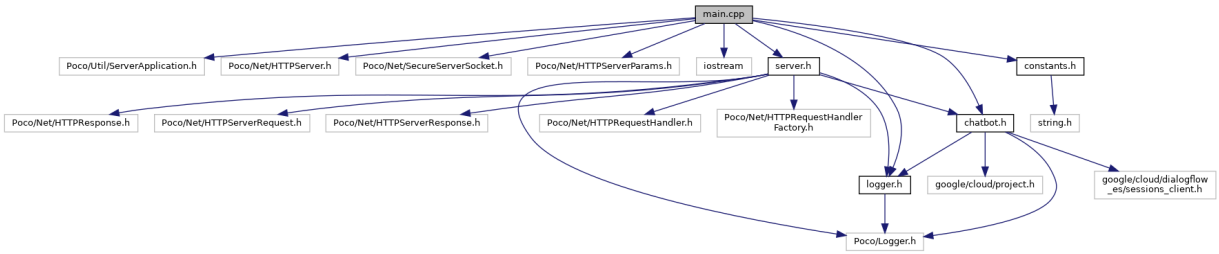

Figura V.1: Include graph-ul principal.

V.3.1 Main

Codul începe prin includerea mai multor fișiere de bibliotecă necesare pentru funcționalitatea serverului. Aceste fișiere de bibliotecă sunt:

- Poco/Util/ServerApplication.h: Această bibliotecă conține clasa ServerApplication, care oferă funcționalități pentru implementarea unei aplicații server.

- Poco/Net/HTTPServer.h: Această bibliotecă conține clasa HTTPServer, care gestionează cererile HTTP primite de la clienți.

- Poco/Net/SecureServerSocket.h: Această bibliotecă conține clasa SecureServerSocket, care permite comunicarea securizată între server și clienți utilizând SSL/TLS.
- Poco/Net/HTTPServerParams.h: Această bibliotecă conține clasa HTTPServerParams, care permite configurarea parametrilor serverului HTTP.

În plus față de bibliotecile POCO, codul include și alte fișiere specifice aplicației, cum ar fi ”server.h”, ”logger.h”, ”chatbot.h” și ”constants.h”. Aceste fișiere conțin declarații și implementări specifice aplicației și sunt utilizate în codul principal pentru a oferi funcționalitatea dorită.

După includerea fișierelor de bibliotecă și a fișierelor specifice aplicației, se utilizează namespace-ul Poco::Util pentru a evita utilizarea repetată a acestuia în cod.

Următoarea parte a codului definește clasa MyServerApp, care este o clasă derivată din ServerApplication. Această clasă acționează ca un wrapper peste clasa ServerApplication și permite rularea aplicației ca un daemon Unix. Aceasta înseamnă că aplicația poate fi rulată în fundal, fără o interfață grafică și fără a bloca consola (vezi Figura V.2).

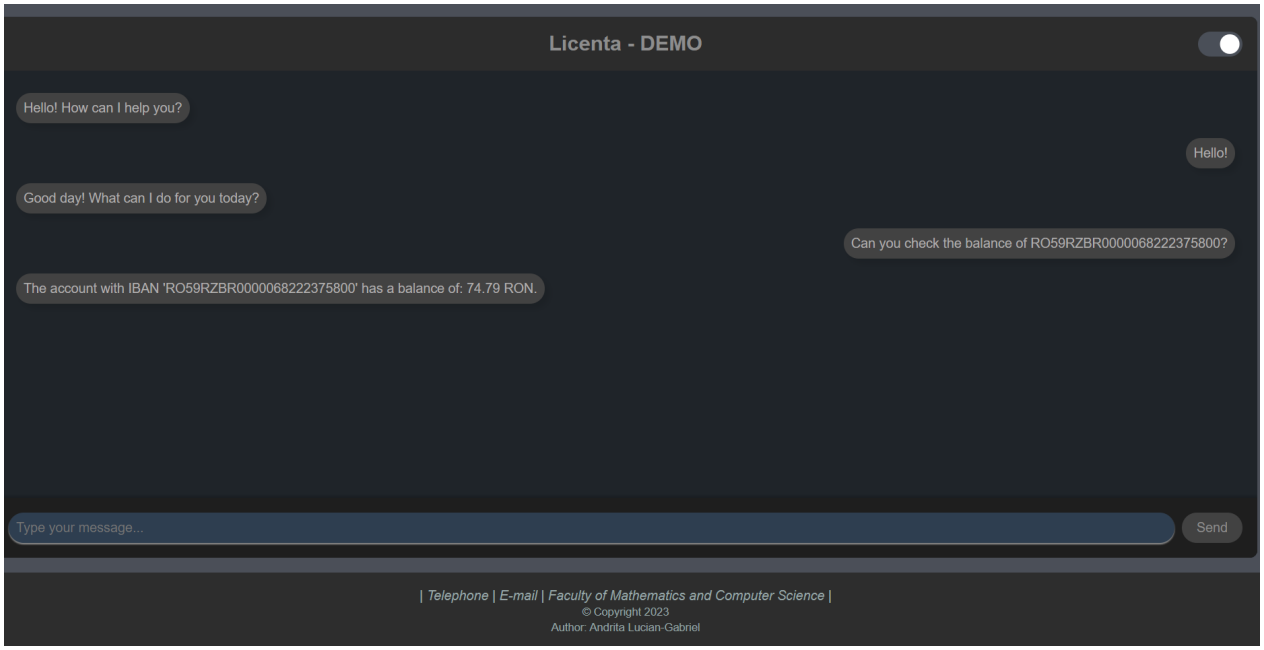

Figura V.2: Exemplu de conversație.

Clasa MyServerApp conține o metodă main suprascrisă, care implementează funcționalitatea serverului. În această metodă, se configurează parametrii serverului HTTP utilizând clasa HTTPServerParams. Parametrii, cum ar fi numărul maxim de conexiuni acceptate și numărul maxim de thread-uri, sunt setați în acest pas.

Apoi, se creează un obiect de tip HTTPServer utilizând clasa HTTPServer și se specifică un MyRequestHandlerFactory care se ocupă de generarea handler-elor pentru cererile primite de la clienți. De asemenea, se specifică portul serverului utilizând obiectul ServerSocket și se folosesc parametrii configurați anterior pentru a inițializa serverul.

După ce serverul este creat și configurat, acesta este pornit utilizând metoda start a obiectului HTTPServer. În plus, se afișează un mesaj de informare folosind obiectul **logger** sau prin afișarea directă la consolă.

V.3.2 Chatbot

Clasa Chatbot este responsabilă pentru gestionarea comunicării cu Google Cloud Platform pentru trimiterea și primirea informațiilor. Această clasă este implementată ca un singleton, ceea ce înseamnă că există o singură instanță a acestei clase în întregul program.

Pentru a utiliza funcționalitățile necesare pentru comunicarea cu Google Cloud Platform, avem incluse mai multe fișiere de bibliotecă, cum ar fi "google/cloud/dialogflow_es/sessions_client.h", "google/cloud/-project.h" și "Poco/Logger.h". Aceste fișiere de bibliotecă conțin clase și funcționalități esențiale pentru interacțiunea cu platforma de cloud.

În cadrul clasei Chatbot, avem o serie de metode și membri de date importante. Pentru a asigura că există o singură instanță a clasei Chatbot, avem metoda statică 'getInstance()' care verifică dacă obiectul 'agent' a fost deja creat. Dacă da, returnează referința către această instanță existentă, iar dacă nu, creează o nouă instanță și returnează referința către aceasta. Acest mecanism de singleton asigură că avem o singură instanță a clasei Chatbot în întregul program.

O altă metodă importantă este 'sendMessage(const std::string& message)' care primește un mesaj de la client și îl setează ca text de intrare pentru obiectul 'request'. Apoi, obiectul 'client' trimite informația la Google Cloud Platform și așteaptă un răspuns. Dacă textul de completare din răspuns este gol, un mesaj implicit este stocat în membrul 'outputText' al obiectului 'agent'. Această metodă asigură trimiterea mesajelor și primirea răspunsurilor în cadrul conversației cu platforma de cloud.

Există și metode pentru setarea și obținerea textului de ieșire al agentului, respectiv 'setOutputText(const std::string& outputParam)' și 'getOutputText()'. Aceste metode permit manipularea și accesarea textului de ieșire generat de agentul de chat.

În plus, clasa Chatbot conține și membri de date relevanți pentru funcționarea acesteia. Avem un obiect 'logger' de tip Poco::Logger, care este utilizat pentru înregistrarea mesajelor de sistem și este accesibil prin intermediul clasei Logger definită în fișierul "logger.h". De asemenea, avem un obiect 'client' de tip dialo-

gflow_es::SessionsClient care gestionează interacțiunea cu sesiunile de comunicare cu platforma de cloud. Un alt membru important este obiectul ‘request‘ de tip v2::DetectIntentRequest care reprezintă cererea de comunicare trimisă la platforma de cloud.

În general, clasa Chatbot oferă o interfață simplificată pentru trimiterea și primirea mesajelor către/de la Google Cloud Platform, facilitând astfel implementarea unui agent de chat funcțional. Această clasă utilizează bibliotecile și funcționalitățile oferite de Google Cloud Platform și POCO pentru a asigura o comunicare eficientă și robustă.

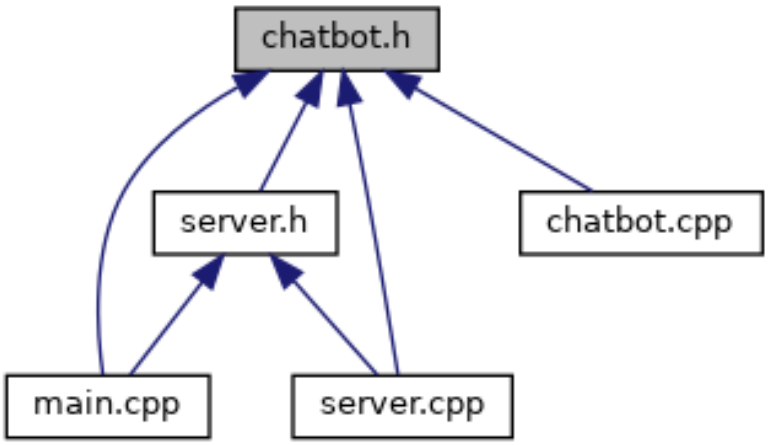

Figura V.3: Include graph-ul pentru clasa Chatbot.

V.3.3 Constants

În codul de mai sus, avem fișierul header ”constants.h” care conține câteva constante și un namespace pentru variabilele de configurare utilizate în program.

În interiorul namespace-ului ”globals”, avem mai multe constante definite:

1. ”serverPort” reprezintă portul folosit pentru server și are valoarea 9090.
2. ”projectID” reprezintă numele proiectului utilizat pe platforma Google Cloud și are valoarea ”licenta-383311”.
3. ”sessionID” reprezintă ID-ul sesiunii utilizate pentru client și are valoarea ”123456789”.
4. ”agentLanguage” reprezintă limba sesiunii și are valoarea ”en” (engleză).

Aceste constante sunt folosite în cadrul programului pentru a configura și personaliza diferite aspecte legate de comunicarea cu platforma Google Cloud.

Fișierul header ”constants.h” este protejat împotriva includerii multiple în codul sursă utilizând macro-ul de preprocesare ”#ifndef CONSTANTS_H_” și ”#endif”, ceea ce asigură că fișierul este inclus o singură dată în cadrul unui fișier sursă.

Utilizarea acestor constante și a namespace-ului ”globals” asigură o gestionare eficientă a valorilor de configurare în cadrul programului și facilitează modificarea acestora într-un singur loc. Aceasta oferă flexibilitate și ușurință în adaptarea programului la diferite scenarii și cerințe specifice.

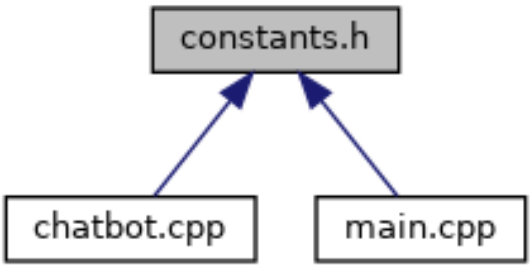

Figura V.4: Include graph-ul pentru clasa Constants.

V.3.4 Logger

Clasa ”MyLogger” care este un wrapper singleton peste clasa Poco::Logger. Acesta este un sistem de înregistrare și logging oferit de biblioteca POCO.

La nivel top-level, clasa ”MyLogger” oferă următoarele funcționalități:

- Metoda statică ”getLogger()” verifică dacă obiectul logger a fost deja creat și returnează o referință către acel obiect. Dacă obiectul nu a fost creat, se va crea un canal pentru fluxul de consolă și un canal pentru fluxul sistemului de fișiere. Se setează anumiți parametri pentru serviciu, iar în funcție de modul în care a fost rulată aplicația (cu ENABLE_DEBUG sau DISABLE_DEBUG), sistemul de înregistrare va folosi canalul de consolă ca utilitate de depanare.

- Metoda statică ”init()” setează valoarea membrului de date ”debug” în funcție de modul în care a fost rulată aplicația. Parametrul ”debugParam” este valoarea transmisă din funcția ”main”.

- Metoda statică ”getDebug()” returnează valoarea membrului de date ”debug”. Dacă aplicația a fost rulată cu ENABLE_DEBUG, se va returna true (utilizează canalul de consolă pentru înregistrări de depanare), altfel se va returna false (utilizează doar canalul de fișiere).

- Metoda statică ”cleanUp()” setează obiectul logger la nullptr. Poco::AutoPtr este un *wrapper* pentru pointeri inteligenți similar cu shared_ptr și verifică numărul de utilizări ale referinței. Deoarece clasa MyLogger este un singleton și are întotdeauna o singură referință, dacă această referință este setată la nullptr, atunci se gestionează automat memoria.

Clasa ”MyLogger” oferă un mod simplu și eficient de gestionare a sistemului de înregistrare în aplicație. Prin intermediul metodelor sale statice, se poate accesa și utiliza obiectul logger într-un mod centralizat și ușor de întreținut. Utilizarea acestui wrapper singleton asigură că obiectul logger este creat o singură dată și poate fi accesat în mod global în cadrul aplicației.

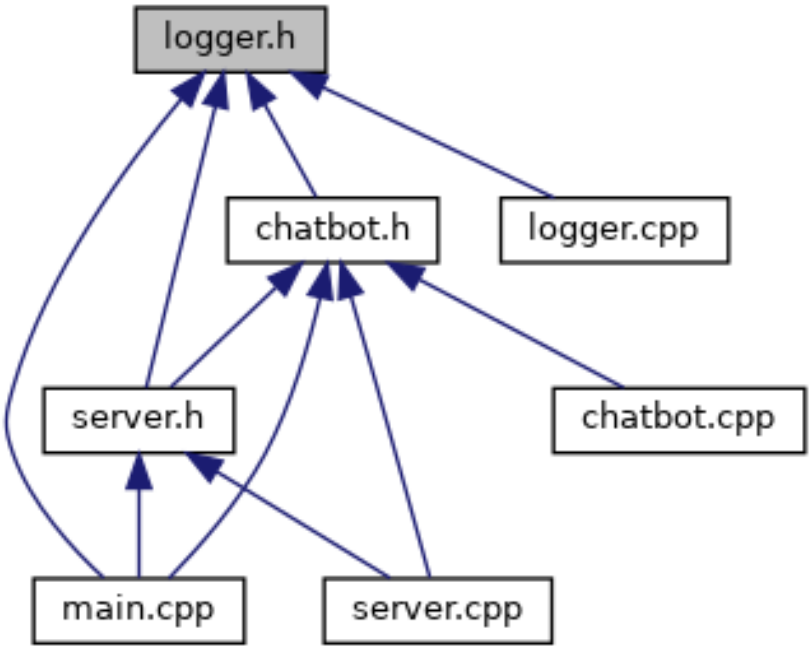

Figura V.5: Include graph-ul pentru clasa Logger.

V.3.5 Server

În fișierul ”Server.h” avem două clase, ”MyRequestHandler” și ”MyRequestHandlerFactory”, care sunt wrappere peste clasele Poco::Net::HTTPRequestHandler și Poco::Net::HTTPRequestHandlerFactory. Aceste clase conțin logica necesară pentru a gestiona cererile și a furniza răspunsurile serverului.

La nivel top-level, avem următoarele funcționalități:

1. Clasa ”MyRequestHandler” este o clasă derivată din clasa Poco::Net::HTTPRequestHandler și reprezintă handler-ul care procesează cererile primite de la utilizatori. Metoda ”handleRequest” primește cererea HTTP de la utilizator și verifică tipul acesteia. Dacă cererea este un payload JSON, atunci este mesajul trimis de utilizator prin JavaScript. Dacă conținutul este o cerere simplă pentru un fișier (sau legături către fișiere), atunci cererea este pasată metodei ”serveResponse” pentru a trimite răspunsul (adică fișierul) corespunzător.

2. Metoda ”serveResponse” trimite prin răspuns fișierul solicitat de client sau o pagină HTML. Aceasta primește răspunsul HTTP și numele fișierului solicitat împreună cu extensia acestuia. Este necesară specificarea extensiei deoarece aceasta indică folderul în care se află fișierul.

3. Clasa ”MyRequestHandlerFactory” este o clasă derivată din clasa Poco::Net::HTTPRequestHandlerFactory și reprezintă o clasă de tip *factory* de handler-e pentru fiecare cerere individuală. Metoda ”createRequestHandler” creează un nou handler pentru fiecare cerere primită.

Ambele clase utilizează obiectul ”logger” din clasa ”MyLogger” pentru a înregistra informații relevante și a le afișa în consolă sau a le scrie în fișiere de log.

Aceste wrapper-e peste clasele din biblioteca POCO oferă un nivel de abstractizare și encapsulare pentru a gestiona cererile primite de la utilizatori și a furniza răspunsurile corespunzătoare. Ele permit serverului să ofere un serviciu robust și scalabil, gestionând diferite tipuri de cereri și direcționându-le către funcționalitățile adecvate ale aplicației.

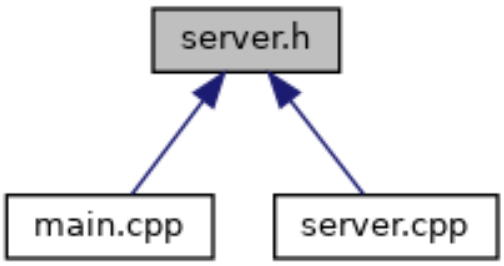

Figura V.6: Include graph-ul pentru clasa Server.

Capitolul VI

Concluzii

În această lucrare de licență, am abordat implementarea și funcționarea unui chatbot utilizând Dialogflow, un serviciu dezvoltat de Google pentru înțelegerea limbajului natural și gestionarea conversațiilor. Chatbot-ul implementat are capacitatea de a interacționa cu utilizatorii într-un mod inteligent și de a oferi răspunsuri personalizate și relevante.

Am început prin a explora principiile de bază ale tehnologiei de chatbot și am identificat avantajele pe care le oferă, cum ar fi automatizarea proceselor, îmbunătățirea experienței utilizatorilor și creșterea eficienței operaționale. Am realizat o analiză a platformei Dialogflow și a funcționalităților sale, precum și a modului în care aceasta se integrează cu alte servicii și API-uri.

În continuare, am prezentat implementarea unui chatbot utilizând Dialogflow și Google Cloud Function. Am examinat modul în care se configurează un agent Dialogflow și cum se stabilesc intent-uri și întrebări de căutare pentru a ghida conversația cu utilizatorii. Am detaliat logica de procesare a cererilor și generare a răspunsurilor, inclusiv interacțiunea cu servicii externe pentru a obține informații sau a efectua operații specifice.

De asemenea, am explorat integrarea chatbot-ului cu un serviciu de stocare în cloud și modul în care acesta poate fi utilizat pentru a gestiona și accesa date specifice. Am prezentat exemple concrete de funcționalități implementate în chatbot, cum ar fi verificarea soldului conturilor bancare, transferul de bani, crearea și ștergerea de conturi.

În concluzie, această lucrare de licență a evidențiat potențialul și beneficiile implementării unui chatbot utilizând Dialogflow. Chatbot-ul oferă o modalitate eficientă și interactivă de a interacționa cu utilizatorii, facilitând gestionarea conversațiilor și furnizarea de răspunsuri personalizate. Implementarea chatbot-ului a implicat utilizarea unor tehnologii și servicii avansate, precum Dialogflow, Google Cloud Storage și Google Cloud Function, care au oferit un mediu puternic și scalabil pentru dezvoltarea și rularea chatbot-ului.

În viitor, există potențial pentru extinderea funcționalităților chatbot-ului și integrarea sa cu alte sisteme și platforme. De asemenea, se pot explora metode avansate de înțelegere a limbajului natural și de procesare a conversațiilor pentru a oferi o experiență și mai fluidă și personalizată utilizatorilor.

Implementarea și dezvoltarea unui chatbot reprezintă o provocare interesantă și promițătoare în domeniul inteligenței artificiale și al interacțiunii om-calculator. Prezenta lucrare a oferit oportunitatea de a explora această tehnologie și de a înțelege beneficiile și provocările implicate.

Bibliografie

[1] *Amazon Lex*, 2023, URL: https://docs.aws.amazon.com/lexv2/latest/dg/how-it-works.html, accesat: 12.04.2023.

[2] *Boost Documentation*, 2023, URL: https://www.boost.org/doc/, accesat: 19.04.2023.

[3] *Cloud Functions Documentation*, 2023, URL: https://cloud.google.com/functions/docs, accesat: 9.04.2023.

[4] *Cloud Storage Documentation*, 2023, URL: https://cloud.google.com/storage/docs, accesat: 6.04.2023.

[5] *CppCMS Documentation*, 2023, URL: http://cppcms.com/cppcms_ref/1.1.0/, accesat: 19.04.2023.

[6] *Dialogflow API C++ Client*, 2023, URL: https://googleapis.dev/cpp/google-cloud-dialogflow_es/latest/index.html, accesat: 12.02.2023.

[7] *Dialogflow Documentation*, 2023, URL: https://cloud.google.com/dialogflow/docs, accesat: 5.04.2023.

[8] *Dialogflow System Entities*, 2023, URL: https://cloud.google.com/dialogflow/es/docs/reference/system-entities, accesat: 20.05.2023.

[9] Chen Hongshen, Liu Xiaorui, Yin Dawei și Tang Jiliang, „A Survey on Dialogue Systems:Recent Advances and New Frontiers", în *ACM SIGKDD Explorations Newsletter* 19.2 (2017), pp. 25–35, DOI: 10.1145/3166054.3166058.

[10] *IBM Watson Documentation*, 2023, URL: https://cloud.ibm.com/developer/watson/documentation, accesat: 12.04.2023.

[11] Cătălin Ioniță, *BCR lansează chatbot-ul ADA*, URL: https://www.bcr.ro/ro/presa/informatii-de-presa/2022/04/14/BCR-lanseaza-chatbot-ul-ADA-primul-asistent-virtual-care-poate-oferi-suport-rapid-si-informatii-personalizate-atat-pentru-persoane-fizice-cat-si-solutii-de-finantare-pentru-companii, accesat: 20.02.2023.

[12] Peter Mell și Timothy Grance, *The NIST Definition of Cloud Computing*, rap. teh. 800-145, National Institute of Standards și Technology, 2011, URL: https://doi.org/10.6028/NIST.SP.800-145.

[13] *Microsoft LUIS*, 2023, URL: https://learn.microsoft.com/en-us/azure/cognitive-services/luis/, accesat: 12.04.2023.

[14] Ajay Ohri, *Importance of Cloud Computing in 2020*, URL: https://u-next.com/blogs/cloud-computing/importance-of-cloud-computing/, accesat: 27.02.2023.

[15] *Poco Documentation*, 2023, URL: https://pocoproject.org/documentation.html, accesat: 19.04.2023.

[16] Rakesh Sharma, „Google's Chatbot Analytics Platform Chatbase Launches to Public", în *Business Standard* (2018), URL: https://www.business-standard.com/article/technology/google-s-chatbot-analytics-platform-chatbase-launches-to-public-118031300494_1.html, accesat: 5.03.2023.

[17] *Source Repositories Documentation*, 2023, URL: https://cloud.google.com/source-repositories/docs, accesat: 14.04.2023.

[18] Ion Surdu, *Cum ajută asistenții virtuali activitatea băncilor: experiența BT, care a fost prima bancă din România ce a lansat un chatbot pentru clienți*, URL: https://www.transilvaniabusiness.ro/2020/10/13/cum-ajuta-asistentii-virtuali-activitatea-bancilor-experienta-bt-care-a-fost-prima-banca-din-romania-ce-a-lansat-un-chatbot-pentru-clienti/, accesat: 28.02.2023.

[19] *WT Documentation*, 2023, URL: https://www.webtoolkit.eu/wt/documentation, accesat: 19.04.2023.