Joy-bot personal healthcare

Progetto Sistemi Distribuiti e Cloud Computing

Elisa Verza
0311317

Daniele La Prova
0320429

***Abstract*—In questo documento è presentata la web app JoyBot, della quale sono elencate le caratteristiche progettate e al momento implementate. Ne viene descritta l'architettura, elencando i microservizi di cui è composta e citando come essi comunicano tra loro e come sono organizzati al loro interno. È illustrato il processo deployment nel dettaglio delle fasi in cui si articola. Sono elencate le tecnologie utilizzate per lo sviluppo del sistema. Chiude il documento una breve trattazione dei problemi riscontrati durante lo sviluppo.**

I. Introduzione

Joy-Bot è un sistema con lo scopo di semplificare la gestione delle faccende mediche degli utenti. Tra le features progettate si possono trovare:

- Archiviazione dei documenti medici dell'utente (ricette mediche, risultati di analisi, ...);
- Studio dei dati estratti dai documenti medici per fornire predizione sull'andamento di parametri;
- Facilitare il contatto tra un utente e i suoi dottori.

Al momento, sono state implementate le seguenti features:

- Registrazione al servizio come utente Paziente;
- Caricamento e archiviazione di ricette mediche;
- Estrazione del nome dei farmaci prescritti nelle ricette, insieme alla loro frequenza;
- Presentazione dei risultati estratti al punto precedente all'utente.

Il codice è ospitato al seguente indirizzo: https://github.com/caballo-domestico/joy-bot

II. Architettura

La web app Joy-bot è stata sviluppata avvalendosi di un'architettura a microservizi basata su container (1). La comunicazione sincrona è stata sviluppata tramite chiamate RPC, mentre si è adottato un sistema pub/sub per la comunicazione asincrona. Per garantire disaccoppiamento dei dati tra microservizi è stato scelto il pattern database per service, mentre si è scelto di utilizzare il pattern circuit breaker per la rilevazione di fault dei microservizi.

A. Struttura sistema: container, decomposizione microservizi

Il sistema è decomposto nei seguenti microservizi:

- `manageusers`, responsabile della registrazione e autenticazione degli utenti;
- `webserver`, che espone i servizi offerti dall'applicazione mappandoli a delle rotte HTTP;
- `prescription-analyzer`, che estrae e memorizza dati rilevanti dalle ricette mediche caricate dagli utenti utilizzando il servizio Amazon Textract;
- `Amazon Dynamo DB`, per permettere agli altri microservizi di disporre di tabelle private dove memorizzare i propri dati;
- `Amazon S3`, utilizzato per poter memorizzare il contenuto in bytes delle ricette;
- `Amazon SNS`, utilizzato dal microservizio manageusers per inviare OTPs come parte del processo di registrazione.

I microservizi comunicano tra loro con protocolli diversi a seconda dei partecipanti alla comunicazione:

- Le chiamate sincrone tra i microservizi in container (manageuser, prescription-analyzer, webserver) avvengono mediante gRPC e Protocol Buffer;
- Le chiamate agli AWSs sono effettuate mediante l'Amazon SDK corrispondente alla tecnologia in cui il microservizio richiedente è sviluppato (ad esempio, Boto3 per Python)
- Le chiamate asincrone tra i microservizi in container sono implementate mediante l'uso di Apache Kafka come sistema Pub-Sub, previa serializzazione delle notifiche in Protocol Buffer.

L'orchestrazione del sistema è delegata alla tecnologia Docker Compose, che si preoccupa di:

- Istanziare i container ospitanti i microservizi e il middleware di comunicazione pub-sub;
- Avviare i containers secondo un ordine che rispetti il loro grafo delle dipendenze;
- Montare un volume contenente le credenziali e le configurazioni necessarie per interagire con AWS;
- esporre la porta 8080 permettendo al webserver di accogliere le richieste HTTP.

Il sistema delega ai servizi di aws la fornitura dell'infrastuttura necessaria a implementare una Base di dati (Amazon DynamoDB), un archivio di files (Amazon S3), ed una macchina host (Amazon EC2) se richiesta.

Il sistema è installabile su una macchina locale o su un'istanza di Amazon EC2. Per i dettagli dell'installazione e configurazione si rimanda al README nella repository.

B. Organizzazione database e datastorage

L'applicazione utilizza DynamoDB per memorizzare i dati. Sfruttando il pattern *database per service* ogni microservizio

che deve effettuare operazioni sui dati avrà accesso solo alle tabelle a lui dedicate. Nel caso in cui un microservizio necessiti di dati provenienti dalle proprie tabelle potrà effettuare lui stesso una query, se invece i dati si trovano in una tabella dedicata ad un altro microservizio sarà quest'ultimo a dover effettuare la query ricevendo i parametri necessari tramite una chiamata gRPC e resituendo i riultati ottenuti.

In (2) è possibile vedere il diagramma ER del database. Le tabelle necessarie all'applicazione sono le seguenti:

- `Prescriptions`: contiene i riferimenti ai file con le prescrizioni dei medici per ogni paziente. Ha come partition key il nome univoco del file contenente le prescrizioni del medico, il riferimento al file e il nome dell'utente a cui appartiene;
- `Prescription-analysis`: Annota il risultato dell'analisi del contenuto delle ricette mediche, in particolare i farmaci prescritti e le rispettive frequenze di assunzione;
- `Users`: serve per memorizzare i dati di tutti gli utenti registrati all'applicazione. Ha come partition key il numero di telefono dell'utente, un campo per verificare che l'utente abbia confermato la sua identità inserendo il codice OTP ricevuto tramite sms, un nome utente, un'email e la password. Per rendere efficienti tutte le query si è reso necessario creare un *Global secondary index*:
  - `Users-username`: contiene il numero di telefono (in quanto partition key della tabella primaria) e come attribute projection la colonna username che diventa partition key della tabella.
- `Pin`: Questa tabella serve per memorizzare i codici OTP inviati agli utenti in fase di registrazione, ma ancora non inseriti per la verifica dell'account. La partition key è il numero di telefono dell'utente, la tabella ha solo un altro campo dedicato al pin. Appena l'utente verifica il proprio account l'entry con il suo pin viene eliminata.

DynamoDB, per permettere query più veloci impone un limite di 400KB per item, quindi per mantenere i file con le prescrizioni mediche degli utenti nella tabella `Prescriptions` è stato utilizzato un bucket s3, in modo tale da poter referenziare i file tramite link. La creazione delle tabelle e del bucket viene effettuata da Terraform.

C. Struttura webserver

Il webserver contiene l'interfaccia che permette all'utente di interagire con l'applicazione. E' responsabile della gestione del frontend per questo prende in carico tutte le richieste http effettuate dagli utenti. Si trova su un container dedicato e per quanto riguarda il database può accedere solo alla tabella `Prescriptions`. Il codice è scritto in Python mentre per il server è stato utilizzato Flask. Nel webserver è possibile effettuare le seguenti operazioni:

- Login: per poter utilizzare l'app è necessario autenticarsi, le credenziali sono numero di telefono e la password. Ci sono due stati in cui può trovarsi un utente:
  - Confirmed: utenti con un account attivo perchè hanno portato a termine la procedura di registrazione inserendo anche il codice OTP ricevuto via sms.
  - Unconfirmed: utenti che non hanno un account attivo e quindi non possono avere accesso all'app perchè non hanno verificato la propria identità tramite il codice OTP.

  Una volta inseriti i dati richiesti nell'apposito form verrà invocato il metodo `login()` che prenderà i dati dalla richiesta http e dovrà procedere all'autenticazione dell'utente. A questo scopo viene effettuata una chiamata gRPC al container `manage users`, il quale è l'unico ad avere accesso alla tabella `users`. La struttura del messaggio di richiesta contiene solo la password dell'utente che richiede l'autenticazione, mentre il messaggio di risposta conterrà l'username ed un campo booleano posto a `true` se l'utente si trova nello stato confirmed, `false` altrimenti. Quindi il `webserver` procede al confronto tra la password restituita da manageuser con quella fornita dall'utente, dal quale si può verificare uno dei seguenti scenari:
  - Password corretta, l'utente è autenticato e può usufruire dei servizi dell'app
  - Il campo password della risposta gRPC è impostato a `"not found"`, in questo caso la query non ha trovato risultati quindi o l'utente ha inserito credenziali sbagliate o non è registrato, in ogni caso viene segnalato l'errore dopo aver reindirizzato l'utente nuovamente alla pagina di login.
  - Il campo confirmed della risposta gRPC è `false`, in questo caso l'utente viene reindirizzato alla pagina in cui poter inserire il codice OTP e finalizzare la registrazione

  Andata a buon fine l'operazione lo username viene salvato come cookie insieme ad un campo `logged` posto a `true`, in modo tale da poter distinguere se l'utente è autenticato o no. In 3 è possibile vedere il diagramma di flusso per il login.
- Signin: direttamente dalla pagina del login è possibile accedere alla pagina di registrazione per nuovi utenti. Vengono richiesti all'utente email, username, numero di telefono e password, sono tutti campi necessari con username e numero di telefono unici all'interno dell'applicazione. Il signin effettua una sola chiamata gRPC verso il container `manage users`, il messaggio di richiesta contiene tutti i campi compilati nel form di registrazione ed in aggiunta un campo booleano `confirmed` posto a `false`. Il messaggio di risposta contiene due campi booleani `available` e `unregistered`: se posto a `true` il primo indica che il nome utente scelto è disponibile, il secondo assicura che il numero di telefono non sia già presente nella tabella degli utenti e viceversa. Se uno dei due è `false` viene visualizzato a schermo il relativo errore (*username/numero di telefono esistenti* a seconda del caso), se tutti e due risultano validi si procede con la

registrazione:

  - Vengono impostati cookie con nome utente, numero di telefono e un contatore inizialmente posto ad 1 che servirà nella fase di conferma dell'account;
  - Viene invocato il metodo che tramite la libreria boto3 ed il servizio SNS invia un SMS, contenente un codice OTP, all'utente e con una chiamata gRPC al container `manage users` memorizza nella tabella `Pin` il codice OTP associato al numero di telefono dell'utente.
  - L'utente viene reindirizzato alla pagine dove inserire il codice OTP

- Conferma registrazione: Per verificare il codice OTP viene preso dai cookies il numero di telefono dell'utente e viene effettuata una chiamata gRPC verso il container `manage users`, il messaggio di richiesta contiene: numero di telefono, tipo di operazione richiesta che può essere get o delete sulla tabella dei Pin e un campo booleano, `real user`, per specificare se la conferma del codice OTP è andata a buon fine o no. La prima volta viene invocata con il campo operazione posto a `get` e `real user` a false. Il messaggio di risposta conterrà il codice OTP da verificare. L'utente ha solo tre tentativi per farlo e il conteggio viene tenuto dal contatore presente nei cookies. Ci si può trovare in uno dei seguenti scenari:
  - Il codice OTP inserito è corretto, viene invocata nuovamente la chiamata gRPC precedentemente usata con oprazione remove e `real user` posto a true. Il container `manage users` si occuperà di eliminare dalla tabella Pin l'entry relativa all'utente ed impostare il campo `confirmed` della tabella Users a true, il contatore nei cookies viene eliminato.
  - Il codice OTP inserito non è corretto, ma sono stati effettuati meno di tre tentativi. In questo caso, viene incrementato il contatore presente nei cookies e segnalato l'errore all'utente che si troverà reindirizzato nella stessa pagina di autenticazione
  - Il codice OTP inserito non è corretto e sono stati effettuati già tre tentativi. In questo caso il campo counter dei cookies viene resettato, si invoca nuovamente la chiamata gRPC con il campo operation posto a `remove` e `real user` false. Il container `manage users` si occuperà di eliminare l'entry relativa all'utente sia nella tabella Pin che nella tabella Users. Verrà segnalato il relativo errore all'utente che verrà reindirizzato alla pagina di signin per eseguire nuovamente la procedura di registrazione.

  Andata a buon fine questa operazione si viene reindirizzati alla pagina per effettuare il login. In 4 è possibile vedere il diagramma di flusso per il signin.

Una volta che l'utente si è autenticato ha accesso a tutte le funzionalità dell'app. In questa sezione anche il webserver deve aver accesso al database a tale scopo è stata implementato un `Dao` che mette a disposizione operazioni per fare upload e download di file dal bucket S3 ed aggiungere elementi ad una tabella. Le funzionalità dell'app disponibili all'utente sono le seguenti:

- Logout: i dati dell'utente, username e numero di telefono, vengono eliminati dai cookies e il campo logged viene posto a false.
- Prescription upload: la relativa interfaccia utente contiene un campo tramite cui selezionare il file contenente la prescrizione che si desidera caricare. Come prima cosa si verifica che il file sia stato effettivamente selezionato, quindi presente nella richiesta e che abbia l'estensione corretta, i formati supportati sono `pdf, png, jpg, jpeg`. Se il file rispetta i criteri precedenti si procede con la memorizzazione. Insieme all'url relativo al file salvato nel bucket S3, vengono salvati lo username e il nome del file. Per le prescrizioni è stato implementato il `Prescription Dao`, che provvederà a pubblicare il messaggio dell'upload sul sistema pub-sub Kafka, in modo tale da comunicare con il container `Prescription Analyzer` che analizzerà le prescrizioni caricate. A questo punto viene caricato sul bucket s3 il file e viene inserita la nuova entry nel database tramite i metodi messi a disposizione dal `Dao`. Completato il caricamento l'utente verrà reindirizzato in una pagina dove saranno mostrate tutte le prescrizioni da lui caricate, il nome dell'utente viene inserito nella richiesta http per poter successivamente eseguire l'operazione list illustrata di seguito.
- List prescription: restituisce una lista di tutte le prescrizioni caricate dall'utente tramite l'applicazione. Una volta ottenuto lo username dalla richiesta http, viene effettuata una query sulla tabella `Prescriptions` tramite il `Prescription Dao`. Il risultato della query contenente la lista dei file viene messa a disposizione per l'utente.
- Get prescription: dalla pagina list prescription, cliccando sulla singola prescrizione è possibile scaricare il documento. Il nome utente viene letto dalla richiesta http e anche in questo caso il `Prescription Dao` tramite le operazioni offerte dal `Dao`, carica il file richiesto da s3 e lo invia all'utente destinatario.
- Dashboard: questa operazione permette all'utente di avere una lista completa delle medicine con la frequenza di assunzione di quest'ultime. Tramite il client gRPC viene creato un canale per lo stream di dati con il container `prescription analyzer`. Il messaggio da parte del webserver contiene lo username dell'utente. Il prescription analyzer chiama il suo `PrescribedDrugsDao` per ottenere dalla tabella `Prescription analysis` una lista di oggetti `PrescribedDrug` conenenti ciascuno i farmaci dell'utente con la frequenza di assunzione e li invia come stream al server.

D. Struttura manage users

In questo container avvengono tutte le interazioni con la tabella `users` del database e le operazioni vengono invocate

solo tramite chiamate gRPC da parte del webserver. E' stato implementato un `Dao` che mette a disposizione le operazioni di add, get, delete, update di una entry della tabella ed un metodo che esegue le operazioni di get sul GSI della tabella `Users`. Tramite `manage users` è possibile:

- Registrare un utente: per salvare i dati dell'utente ricevuti nel messaggio di richiesta della chiamata gRPC è stato implementato il `Registration dao`. Bisogna verificare che il nome utente e il numero di telefono inserito dall'utente non siano già stati registrati in precedenza, quindi tramite il `Registration Dao` si effettuano le queries sulla tabella `Users` e sul suo indice `Users username`. Se entrambe le queries non producono risutati, i dati dell'utente vengono salvati nel database. La risposta alla chiamata gRPC imposterà i campi `available` e `unregistered` a true o false a seconda dell'esito del controllo fatto in precedenze
- Memorizzare codice OTP: questa operazione serve a gestire la tabella `Pin` del database, a questo scopo è stato implementato il `Pin dao` che permette di eseguire tutte le interazioni necessarie con il database. A seconda dell'operazione scelta, tramite parametro `op` passato in input, si possono eseguire diversi comandi:
  - `get`: utilizzato per effettuare la query, dato il numero di telefono dell'utente che deve finalizzare la registrazione, sulla tabella `Pin`. In questo caso la risposta alla chiamata gRPC avrà il codice OTP nel campo pin.
  - `add`: in fase di signin viene chiamato per registrare un nuovo pin nella relativa tabella. In questo caso il campo pin della risposta alla chiamata gRPC vine posto a 0
  - `delete`: una volta terminata la procedura di registrazione bisogna eliminare l'entry del pin utilizzato dalla tabella. In questo caso viene effettuato il controllo sul campo della chiamata gRPC `real user`: se posto a true viene eliminata l'entry nella tabella Pin e viene impostato l'attributo `confirmed` della tabella Users a true, se è false invece viene eliminata l'entry relativa all'utente sia dalla tabella Pin che da Users, in questo caso la registrazione non è andata a buon fine per l'inserimento di troppi codici OTP errati.
- Effettuare il login: per questa operazione vengono richiesti dal webserver i dati dell'utente che sta cercando di autenticarsi. Tramite il numero di telefono viene effettuata una query sulla tabella Users, se viene trovato l'elemento si inserisce nella risposta alla chiamata gRPC altrimenti viene ritornata la stringa `not found` in modo che il webserver possa segnalare l'errore all'utente.

E. Struttura prescription-analyzer

Il microservizio prescription-analyzer ha la responsabilità di sottoporre le ricette mediche caricate dagli utenti al servizio di analisi offerto da Amazon Textract, in grado tra le altre cose di riconoscere le coppie chiave-valore all'interno di esse. Tale lavoro è rappresentato dal sequence diagram in figura 5.

Ogni qual volta un utente carica una ricetta medica nel sistema una notifica sul topic `PRESCRIPTION_UPLOADED` viene pubblicata mediante Apache Kafka, con la struttura osservabile in V. Questa notifica è consegnata dal broker a un Notification Listener del prescription-analyzer, ovvero una goroutine avviata in precedenza dal Main e inscritta al suddetto topic. Il Notification Listener deserializza la notifica e la consegna al Main in attesa su un canale Go dedicato (Notification Channel). A questo punto il Main contatta Amazon Textract specificando la chiave e il bucket che identificano la ricetta, e ottiene i dati grezzi prodotti dall'analisi. Una volta raffinati, i dati dell'analisi sono salvati in una tabella di DynamoDB associandoli all'utente che ha caricato la ricetta.

Amazon Textract analizza i dati dei documenti sottoposti individuando dei blocchi di testo e categorizzandoli a seconda del loro ruolo all'interno del testo. È inoltre in grado di riconoscere le relazioni tra i blocchi registrando come CHILD l'id di un blocco all'interno di un altro. Il blocco CHILD è quello che subisce la relazione. Nell'esempio mostrato in figura 7 i rettagoli rappresentano i blocchi della categoria scritta nel rispettivo centro, e le frecce sono le relazioni tra essi, la cui punta cade sul blocco registrato come CHILD nel blocco da cui parte. A seguito dell'analisi è stata individuata una coppia chiave-valore con chiave `Name:` e valore `Ana Carolina` in una delle pagine del testo. La chiave è registrata in un blocco KEY che ha come CHILD un blocco WORD che ne contiene il testo e un blocco VALUE che ne rappresenta il valore. A sua volta il blocco VALUE ha diversi CHILDs di tipo WORD, uno per ogni parola che compone il testo del valore associato a quella chiave.

Prescription-analyzer elabora ulteriormente i dati grezzi ricevuti da Amazon Textract per tradurre le relazioni presenti tra i blocchi KEY e VALUE in una mappa Go per facilitarne l'uso nel sistema. Per ogni blocco KEY si ricerca il corrispettivo blocco VALUE e si registra l'associazione nella mappa usando come chiave e valore il rispettivo testo dei blocchi. Per ricostruire il testo completo di un blocco, si ricercano tutti i CHILD dello stesso e si concatenano le parole così ottenute separandole con degli spazi.

Se i farmaci prescritti nella ricetta sono annotati come in figura 8 Prescription-analyzer è in grado di estrarli e di annotarli in un campo `DRUGS` della tabella `Prescription_analysis` mediante la funzione `dao.StoreAnalysis()`, il quale rappresenta una lista di nomi di farmaci prescritti con la rispettiva frequenza di assunzione. Inoltre, è annotato un timestamp che rappresenta la data di scadenza della ricetta da cui è stata prodotta l'analisi.

Prescription-analyzer espone agli altri miscroservizi una chiamata gRPC per poter ritrovare i farmaci prescritti tra tutte le ricette caricate da un utente (6). Un server gRPC è lanciato in esecuzione su una goroutine dedicata e si impegna ad accogliere e servire le richieste in arrivo dal resto del sistema, chiedendo i dati richiesti alla DAO e restituendo il risultato al

middleware di comunicazione. La DAO esegue una query su AmazonDB occupandosi di:

- Specificare un filtro che esclude tutte le ricette scadute al momento della query;
- Iterare sui risultati paginati della query e aggregarli prima di restituirli al chiamante.

III. Deployment Journey

A. Infrastruttura - Terraformazione

La creazione e il mantenimento dell'infrastuttura necessaria al sistema è delegata agli script Terraform presenti nella cartella `demiurges`. Gli script si dividono in:

- `create-ec2-frontend`: responsabile della creazione di un'istanza Amazon EC2 small con Amazon Linux accessibile via internet con protocolli SSH e HTTP. È necessario specificare il nome di una coppia di chiavi creata precedentemente su AWS, che potrà poi essere usata per accedere all'istanza mediante SSH. Al termine della creazione, scrive sul file `infrastructure.json` l'indirizzo IP dell'istanza;
- `create-microservices-resources`: crea le risorse AWS necessarie al sistema per funzionare, come ad esempio tabelle dynamoDB. È possibile specificare durante l'esecuzione il nome del bucket che il sistema dovrà usare per archiviare le ricette mediche. Se omesso, deve essere creato con un altro script terraform o con altri mezzi.

Gli script terraform possono essere eseguiti usando il wrapper `demiurges/launch.sh`, il quale permette di specificare:

- Una serie di script terraform da eseguire, ciascuno in una sua cartella dedicata;
- Quali file `config` e `credentials` da usare per interagire con AWS

Tale wrapper esegue le seguenti operazioni:

- Preparare la cartella di lavoro dello script terraform desiderato per l'utilizzo di Terraform;
- Formattare lo script per migliorarne la leggibilità;
- Validare la correttezza sintattica;
- Eseguire lo script terraform;
- Mostrare lo stato dell'infrastruttura a schermo.

Il fallimento di una qualsiasi delle sopracitate operazioni interrompe l'esecuzione.

B. Compilazione files proto - Preparazione al viaggio

Prima di caricare il codice sulla macchina ospite è necessario compilare i file `.proto` in codice sorgente. Nella cartella `frontend` è presente un Makefile top-level il quale espone un target `stubs`. All'invocazione di `make stubs` verranno eseguiti i Makefiles interni alle cartelle corrispondenti ai microservizi, dove ognuno di essi ha specificato quali file proto descrivono gli oggetti necessari al loro funzionamento. Usare questo meccanismo ricorsivo implica che per ogni microservizio vengono compilati solo i file proto a lui necessari, riducendo il tempo di compilazione complessivo. Inoltre, l'uso del comando `make` comporta che vengono effettivamente compilati solo i file proto che hanno subito modifiche più recenti della data di ultima modifica dei corrispettivi sorgenti creati. Infine, avviare la compilazione prima di effetuare il deploy del sistema presenta il vantaggio non trascurabile di non dover appesantire la macchina ospite con l'ambiente necessario alla compilazione. Si prenda il Makefile in V come esempio rappresentativo dei Makefiles di ogni microservizio.

C. Deployment - Colonizzazione

Uno script python `getIp.py` legge il file `infrastructure.json` per scoprire l'indirizzo ip della macchina su cui effettuare il deploy dell'applicazione, e usa questa informazione insieme al percorso di una chiave ssh per accederci passato come variabile ambientale per preparare un file testuale `hosts`.

Il deploy vero e proprio è delegato ad Ansible, il quale verrà lanciato usando il file `hosts` per fornire le informazioni necessarie per accedere alla macchina che ospiterà il sistema. `deploy.sh` lancerà Ansible con i seguenti playbooks:

- `docker.yml`: Installa e configura Docker e le sue dipendente, preparandolo ad ospitare containers;
- `deployApp.yml` (V): Allinea i files di configurazione AWS e i sorgenti presenti sulla macchina remota con quelli presenti su quella locale usando l'utility `rsync`, trasferendo files sulla rete e apportando modifiche al file system solo se necessario. Inoltre, si assicura che i containers siano up and running attraverso docker-compose, invocando una nuova build dei containers solo se sono state effettivamente apportate modifiche durante l'allineamento dei sorgenti. Un file `.rsync-filter` elenca quali cartelle e files escludere da questo meccanismo di sincronizzazione. Tale meccanismo è stato preferito rispetto al checkout della repository in quanto avrebbe comportato il caricamento sulla macchina ospite di files non necessari all'esecuzione del sistema, come gli stessi strumenti di deployment o la documentazione.

IV. Tecnologie

- Terraform: creazione e monitoring dell'infrastruttura richiesta da AWS;
- Ansible: deploy del sistema sull'istanza EC2, preparazione del suo ambiente, installazione delle dipendenze e controllo della salute del sistema;
- Python, Go: per lo sviluppo dei microservizi;
- Docker, docker-compose: per la messa in container e l'orchestrazione dei microservizi
- Flask: gestione richieste HTTP da user agents;
- Boto3, AWS SDK for go: comunicazione del sistema con i servizi AWS;
- pybreaker, gobreaker: librerie che implementano il pattern Circuit Breaker;
- bootstrap-flask: libreria Python che integra Bootstrap con Flask;
- Protocol Buffer: serializzazione dei messaggi scambiati tra i microservizi nel sistema;

- gRPC: comunicazione sincrona tra i microservizi nel sistema;
- Apache Kafka: comunicazione asincrona tra i microservizi nel sistema;
- kafka-python, kafka-go: librerie connettrici per Apache Kafka;

V. Problemi e limitazioni

- I bucket s3 usati dall'applicazione non sono pubblici. Questo implica che è possibile accederne ai contenuti solamente se il sistema utilizza delle credenziali generate con lo stesso account con il quale sono stati allocati i bucket (#43);
- A causa di limitazioni non meglio chiarite dal supporto AWS, Amazon SNS al momento non invia l'OTP al numero di telefono degli utenti che si vogliono registrare al sistema. La verifica del numero in sandbox tuttavia funziona; (#42);
- Lo script Terraform responsabile della creazione dell'istanza EC2 su cui eseguire il sistema non riavvia l'istanza se essa si trova nello stato `STOPPED` (#3)

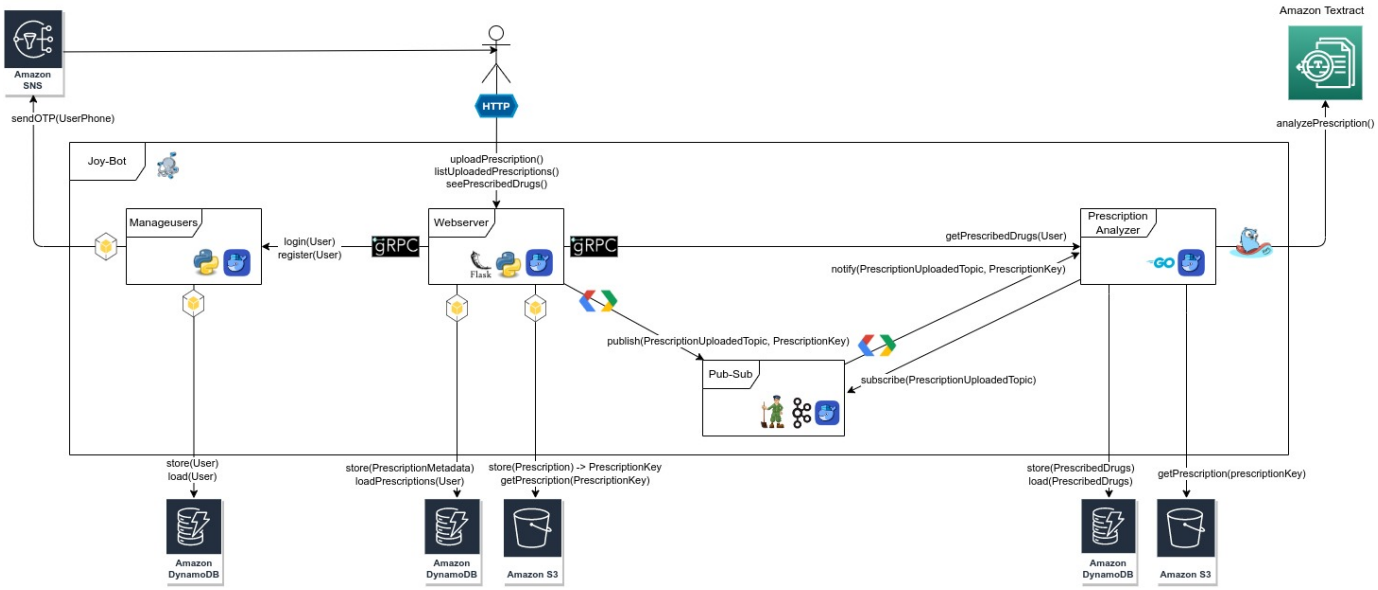

Fig. 1. Joy-Bot Architectural map

```
message PrescriptionUploaded{
    string key = 1;          // key of prescription in bucket
    string username = 2;     // username of user who uploaded prescription
    string s3link = 3;       // link to prescription in s3 bucket
    string bucket = 4;       // name of s3 bucket storing the prescription
}
```

Listing 1: Protocol Buffer definition of a Prescription Uploaded notification.

```
tasks:
  - name: Install rsync
    yum:
      name: rsync
      state: present

  - name: Copy aws files
    ansible.posix.synchronize:
      src: ~/.aws/
      dest: /home/ec2-user/.aws

  - name: Copy app files
    ansible.posix.synchronize:
      src: "{{ local_project_path }}/frontend/"
      dest: "{{ remote_project_path }}"
    register: app_files

  - name: Build and start app
    community.docker.docker_compose:
      project_src: "{{ remote_project_path }}"
      files:
        - docker-compose.yml
      build: "{{ app_files.changed }}"

```

Listing 2: Tasks of `deployApp.yml` Ansible playbook.

```
# what proto files are of the microservice interest
src_protos=prescription_analyzer.proto users.proto notifications.proto
# suffices of the stubs generated, depending on the language
pb_suffix=_pb2.py
grpc_suffix=_pb2_grpc.py
pyi_suffix=_pb2.pyi

src_protos_dir=../proto
generatedDir=.
basenames=$(shell basename -a $(src_protos) 2>/dev/null | awk -F . '{print $$1}')
generated_pb=$(addprefix $(generatedDir)/, $(addsuffix $(pb_suffix), $(basenames)))
generated_grpc=$(addprefix $(generatedDir)/, $(addsuffix $(grpc_suffix), $(basenames)))
generated_pyi=$(addprefix $(generatedDir)/, $(addsuffix $(pyi_suffix), $(basenames)))

$(generated_pb): $(generatedDir)/%$(pb_suffix): $(src_protos_dir)/%.proto
        protoc -I=$(src_protos_dir) --python_out=$(generatedDir) $<

$(generated_grpc): $(generatedDir)/%$(grpc_suffix): $(src_protos_dir)/%.proto
        python -m grpc_tools.protoc -I=$(src_protos_dir) --grpc_python_out=$(generatedDir) $<

$(generated_pyi): $(generatedDir)/%$(pyi_suffix): $(src_protos_dir)/%.proto
        python -m grpc_tools.protoc -I=$(src_protos_dir) --pyi_out=$(generatedDir)  $<

stubs: $(generated_pb) $(generated_grpc) $(generated_pyi)
```

Listing 3: Makefile del microservizio webserver. Il Makefile degli altri microservizi presenta una struttura analoga.

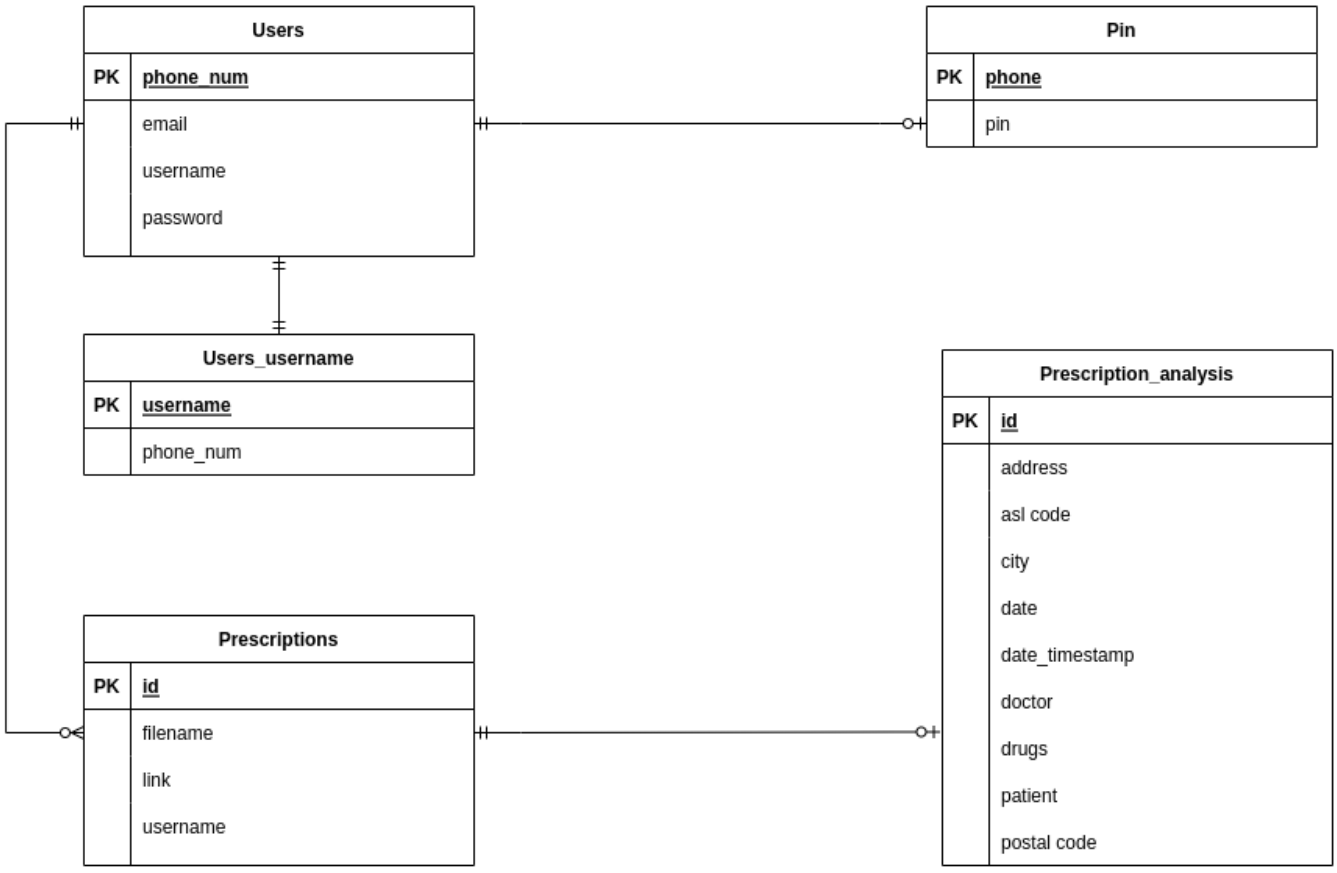

Fig. 2. Struttura del database

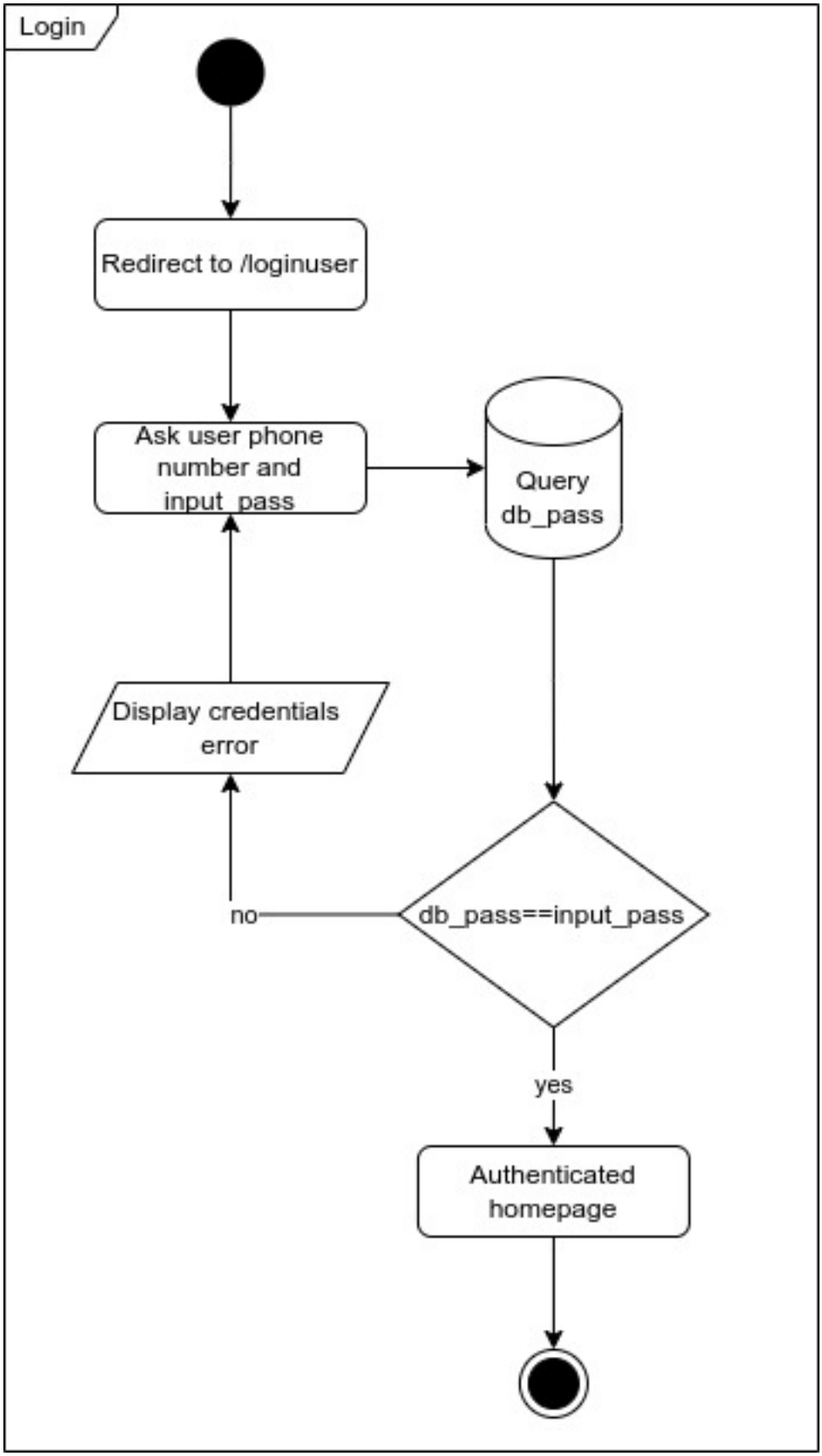

Fig. 3. Diagramma di flusso del login

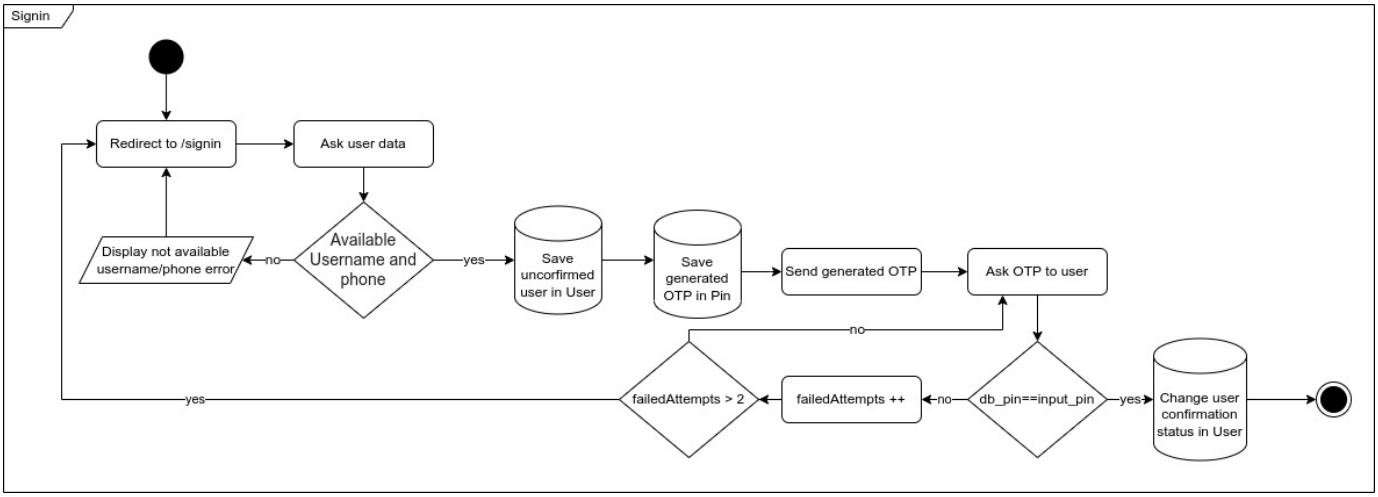

Fig. 4. Diagramma di flusso del signin

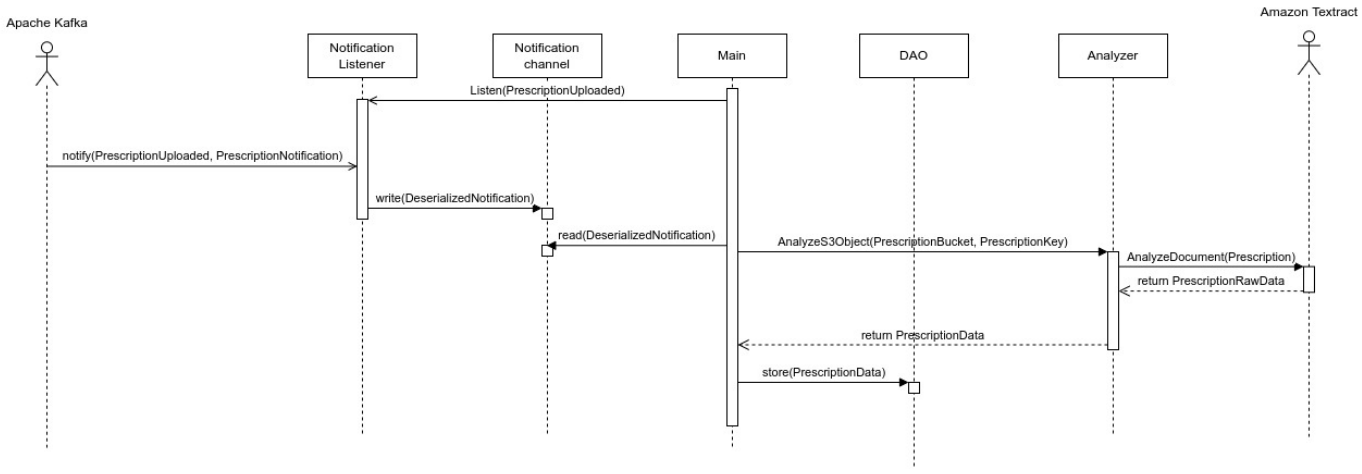

Fig. 5. Sequence diagram del microservizio prescription-analyzer con il focus sul caso d'uso dell'analisi delle ricette.

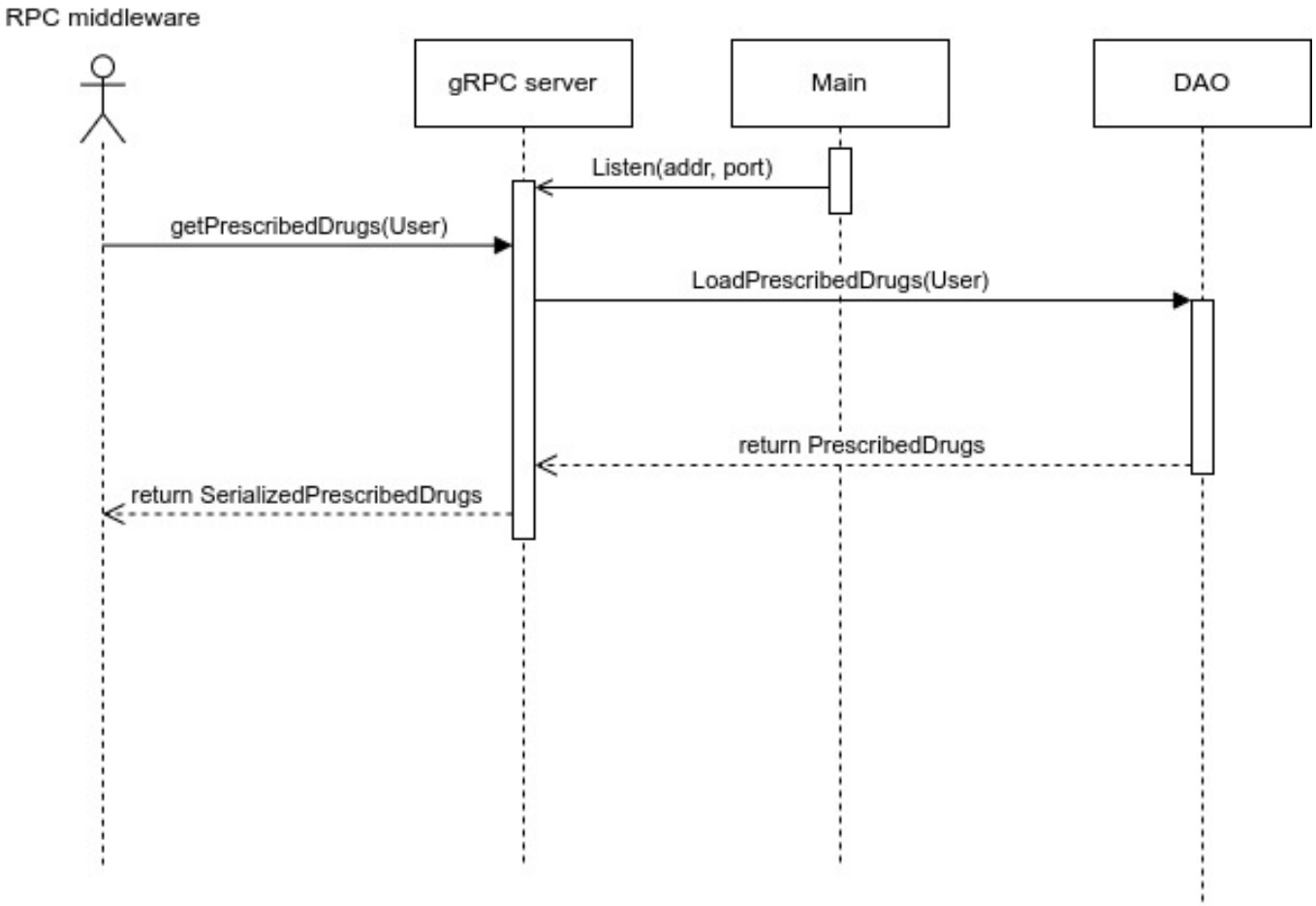

Fig. 6. Sequence diagram del microservizio prescription-analyzer con il focus sul caso d'uso del ritrovo dei farmaci prescritti.

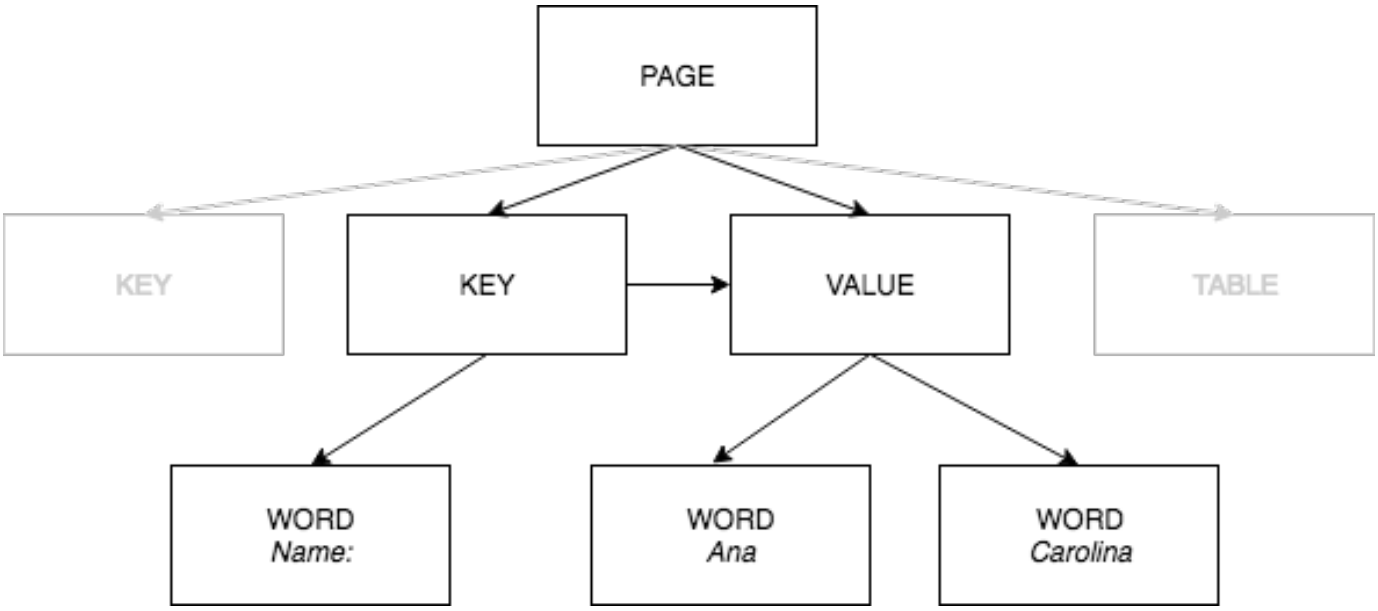

Fig. 7. Diagramma che illustra un esempio dei dati grezzi restituiti da Amazon Textract a seguito dell'individuazione in un documento della coppia chiave-valore `Name: Ana Carolina` (fonte: https://docs.aws.amazon.com/textract/latest/dg/how-it-works-kvp.html)

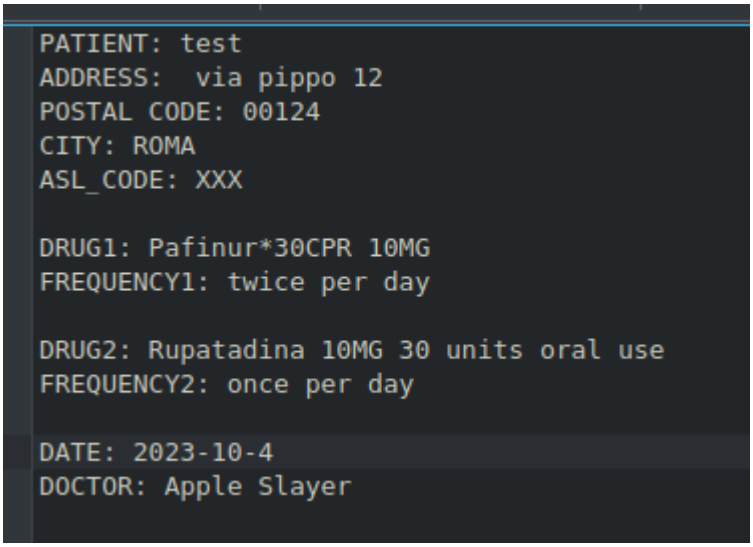

Fig. 8. Esempio di ricetta usato per effettuare i test del microservizio Prescription-analyzer. I farmaci prescritti devono essere scritti nel formato mostrato per renderli riconoscibili ad Amazon Textract.