Progetto di Ingegneria Informatica - Anno 2020 / 2021

ALEXA e la gestione degli impegni per persone con decadimento cognitivo

Alessandro Mazza
Cod. Persona 10670112 Matricola 908651

Davide Marinotto
Cod. Persona 10675966 Matricola 909291

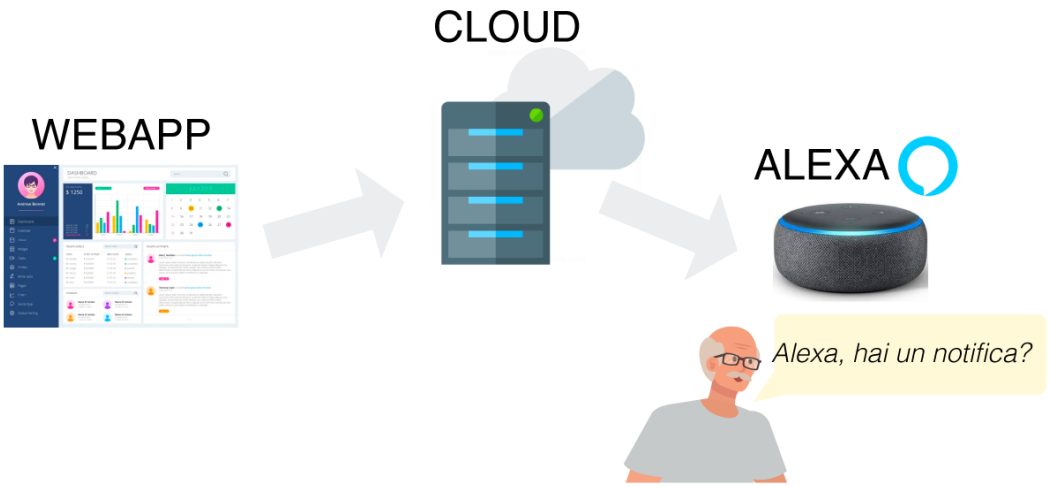
Professore responsabile: Fabio Salice
Mail: fabio.salice@polimi.it

POLITECNICO
MILANO 1863

Sommario

0. Specifica del progetto

Il progetto consiste nella creazione di un'interfaccia studiata per offrire un servizio di supporto a persone con decadimento cognitivo tramite l'utilizzo dell'assistente vocale di Amazon Alexa, consentendo quindi a tali persone di mantenere una propria indipendenza e non richiedere necessariamente i servizi di una RSA
Abbiamo dunque creato un'interfaccia online che permette ai caregiver di impostare facilmente e in maniera immediata notifiche personalizzate che verranno inviate al dispositivo del paziente seguendo i pattern temporali specificati, il paziente poi con un semplice comando vocale potrà chiedere al dispositivo di leggere tutti i promemoria ricevuti.
L'interfaccia è stata strutturata riflettendo la struttura organizzativa di diversi servizi, con un utente Admin in grado di creare nuovi utenti Standard inserendo tutti i dati necessari (nome, cognome, email e password identificativi), mentre gli utenti Standard rappresentano i fruitori principali del servizio, in quanto possono registrare diversi pazienti a cui offrire supporto inserendone nome, cognome e l'id del dispositivo Alexa (nel contesto quindi un utente Standard può essere un cliente privato oppure un'organizzazione che gestisce più pazienti).
Selezionato un paziente sarà quindi possibile impostare un promemoria, salvato con un messaggio, la tipologia di promemoria e la frequenza con cui deve essere recapitato al paziente sfruttando la funzionalità 'NotifyMe' di Alexa, che segnala al paziente tramite un suono e la luce sul dispositivo che è arrivato un promemoria.
Con una semplice frase come "Alexa hai una notifica?" quindi il dispositivo proseguirà a leggere tutti i promemoria ricevuti, ponendoli in una maniera che li renda più user friendly a seconda della tipologia di promemoria (ad esempio, un promemoria segnato come 'Activity' sarà posto come "Che ne dici di ….?", o un 'Reminder' sarà posto come "Non dimenticarti di ….") e garantendo quindi che i promemoria siano effettivamente recapitati al paziente



1. **R**ich **I**nternet **A**pplication

Le RIA sono applicazioni web che possiedono le caratteristiche e le funzionalità delle applicazioni desktop, senza però necessitare di un installazione. Si caratterizzano per la dimensione interattiva, la multimedialità e per la velocità d'esecuzione. Infatti la parte dell'applicazione che elabora i dati è trasferita a livello client e fornisce una pronta risposta all'interfaccia utente, mentre la gran parte dei dati e dell'applicazione rimane sul server remoto, con notevole alleggerimento per il computer utente. Si fondano perciò su un'architettura di tipo distribuito. Anche l'interazione con una RIA avviene in remoto, tramite un comune web browser.

In un certo senso le RIA rappresentano una generazione di applicazioni che permette un'interazione totalmente rinnovata, fondata sugli aspetti migliori delle caratteristiche funzionali e progettuali che finora erano prerogativa alternata del web o delle applicazioni desktop. Inoltre le RIA, per il livello spinto di interattività che esse offrono, rappresentano uno dei canali migliori attraverso il quale si va imponendo il paradigma del cloud computing, che costituisce una nuova modalità di fruizione del software tramite architetture distribuite.

1.1 Database

Il database da noi utilizzato è di tipo relazionale, ovvero tutti i dati sono rappresentati come relazioni e manipolati con gli operatori algebrici, consentendo di creare una rappresentazione consistente e logica tramite appropriate strutture di attributi, valori e vincoli
L'implementazione software da noi scelta è MySQL, un relational database management system (RDBMS) basato su tabelle che rappresentano relazioni tra attributi

1.1.1 Struttura

L'interfaccia online sfrutta un database per il mantenimento dei dati composto da tre tabelle principali: **user**, **patient** e **calendar** strutturate come segue:

```
CREATE TABLE `user` (
  `idUser` int NOT NULL AUTO_INCREMENT,   //identifica univocamente lo user
  `Surname` varchar(45) NOT NULL,
  `Name` varchar(45) NOT NULL,
  `Mail` varchar(320) NOT NULL,
  `Password` varchar(64) NOT NULL,
  `isAdmin` tinyint DEFAULT '0',        //indica se lo user è Admin o Standard
  PRIMARY KEY (`idUser`)
)

CREATE TABLE `patient` (
  `idpatient` int NOT NULL AUTO_INCREMENT, //identifica univocamente il paziente
  `Surname` varchar(45) NOT NULL,
  `Name` varchar(45) NOT NULL,
  `iduser` int NOT NULL,       //indica il caregiver che si occupa del paziente
  `idAlexa` varchar(255) DEFAULT NULL,   //identifica il dispositivo del paziente
  PRIMARY KEY (`idpatient`),
  KEY `idUser_idx` (`iduser`),
  CONSTRAINT `idUser` FOREIGN KEY (`iduser`) REFERENCES `user` (`idUser`) ON
DELETE CASCADE ON UPDATE CASCADE
)
```

```
CREATE TABLE `calendar` (
  `idAlarm` int NOT NULL AUTO_INCREMENT, //identifica univocamente il promemoria
  `message` varchar(255) NOT NULL,   //contiene il messaggio letto dal dispositivo
  `cron` varchar(45) NOT NULL,       //indica in maniera sintetica la frequenza
  `cronString` varchar(255) NOT NULL,  //"traduzione" della voce precedente
  `idPatient` int NOT NULL,       //indica il paziente a cui inviare il promemoria
  `alarmType` varchar(45) DEFAULT NULL,  //indica la tipologia del promemoria
  PRIMARY KEY (`idAlarm`),
  KEY `idPaziente_idx` (`idPatient`),
  CONSTRAINT `idPaziente` FOREIGN KEY (`idPatient`) REFERENCES `patient`
(`idpatient`) ON DELETE CASCADE ON UPDATE CASCADE
)
```

Le rispettive chiavi primarie sono garantite univoche automaticamente all'inserimento di un nuovo elemento grazie alla funzionalità 'AUTO_INCREMENT', inoltre il constraint tra user e patient e quello tra patient e calendar garantiscono che non rimangano dati inutili o incongruenti in seguito a cancellazioni o modifiche dei dati: se viene cancellato uno user, verranno cancellati di conseguenza i relativi pazienti, e allo stesso modo cancellando un paziente tutti i suoi promemoria verranno rimossi dal database.

1.1.2 **D**ata **A**ccess **O**bject

Il DAO è un pattern architetturale per la gestione della persistenza.
Si tratta fondamentalmente di una classe con relativi metodi che rappresenta un'entità tabellare di un RDBMS. Questi hanno lo scopo di stratificare e isolare l'accesso ad una tabella tramite query (poste all'interno dei metodi della classe) ovvero al data layer da parte della business logic creando un maggiore livello di astrazione ed una più facile manutenibilità. I metodi del DAO con le rispettive query dentro verranno così richiamati dalle classi della business logic.
Il vantaggio relativo all'uso del DAO è dunque il mantenimento di una rigida separazione tra le componenti di un'applicazione, le quali potrebbero essere il "Modello" e il "Controllo" in un'applicazione basata sul paradigma Model-View-Controller (MVC).

1.2 BackEnd

Il backend è il lato dell' applicazione completamente collocato nel server, con lo scopo di gestire i dati acquisiti dal front-end e compiere azioni che nella nostra applicazione possono interagire con il database per mezzo dei DAO, gestire i cookie di un utente oppure interagire con le API Alexa.

1.2.1 Node.js & NPM

Node.js è un runtime system open source multipiattaforma orientato agli eventi per l'esecuzione di codice JavaScript. In origine JavaScript veniva utilizzato principalmente lato client con script generalmente incorporati all'interno dell'HTML di una pagina web, poi interpretati dal motore di esecuzione interno del Browser. Node.js consente invece di utilizzare JavaScript anche per scrivere codice da eseguire lato server, ad esempio per la produzione del contenuto delle pagine web dinamiche prima che la pagina venga inviata al Browser dell'utente. Avendo un'architettura orientata agli eventi, Node.js rende possibile l'I/O asincrono, ottimizzando il Throughput e la scalabilità nelle applicazioni web con molte operazioni di input/output, inoltre è ottimo per applicazioni web Sistema real-time.

Una dei principali punti di forza di Node è la presenza di molti pacchetti open source sviluppati e mantenuti dalla community o da aziende per integrare i propri prodotti: per rendere facile e veloce l'installazione, l'aggiornamento e eventualmente la rimozione di questi package Node si affida di default a un gestore esterno chiamato Node Package Manager (NPM) che permette la gestione di pacchetti pubblici e privati contenuti in un database online tramite linee di comando

1.2.2 Express.Js

Express è un framework open source per applicazioni web per Node.js. Progettato per creare web application, è ormai definito il server framework standard de facto per Node.js. I suoi punti di forza sono la leggerezza e la flessibilità grazie alle quali si possono creare prodotti orientati alle performance con basso consumo di banda, quindi adatti al web moderno in particolare al mondo mobile. Questo framework offre molte funzioni aggiuntive per la gestione del middleware, facilita le operazioni di route, lo scambio di dati client server, la gestione dei cookie e la verifica dei dati.

1.2.3 Cron

Per legare i record nel database (nel quale è presente un messaggio, un destinatario e un trigger temporale) con un vero e proprio call to action al momento designato usiamo un modulo di Node.js che implementa un sistema cron-like ispirato al cron dei sistemi Unix che consente la pianificazione di comandi, ovvero la registrazione di questi presso il sistema per poi mandarli in esecuzione periodicamente in maniera automatica.
Per fare ciò cron usa un "demone" crond, il quale sarà quindi costantemente in esecuzione in background e, una volta al minuto, legge i contenuti del registro dei comandi pianificati per eseguire quelli per cui si è esaurito il periodo di attesa.

I file crontab (utilizzati come «registri» dei comandi pianificati) contengono la lista dei *job* e altre istruzioni per il demone di cron e seguono un formato particolare, composto da una serie di campi divisi da spazi o tabulazioni.
Uno schema riassuntivo della sintassi:

```
Min  Hour Day  Mon  Weekday

*    *    *    *    *  command to be executed

┬    ┬    ┬    ┬    ┬
│    │    │    │    └─── Weekday  (0=Sun .. 6=Sat)
│    │    │    └──────── Month    (1..12)
│    │    └───────────── Day      (1..31)
│    └────────────────── Hour     (0..23)
└─────────────────────── Minute   (0..59)
```

In questo progetto sfruttiamo quindi i primi cinque campi di tale formato (rispettivamente, minuti, ore, giorno del mese, mese e giorno della settimana), che specificano con che frequenza e quando eseguire un comando.
Per esempio, “Ogni 3 ore” è salvato con il cron ‘0 */3 * * *’, “Ogni giorno alle 12:00” con il cron ‘0 12 * * *’ e “Ogni Lunedì e Giovedì alle 12:30” con il cron ‘30 12 * * 1,4’

1.2.4 DataExport - JsonToCSV

Per capire nel dettaglio come rendere esportabili i dati della nostra RIA abbiamo analizzato i tipi di file che possono entrare in gioco.
Nel backend i dati sono oggetti serializzabili tramite JSON (JavaScript Object Notation), un semplice formato per lo scambio di dati. Per le persone è facile da leggere e scrivere, mentre per le macchine risulta facile da generare e analizzare la sintassi.
JSON è un formato di testo completamente indipendente dal linguaggio di programmazione, ma utilizza convenzioni conosciute dai programmatori di linguaggi della famiglia del C.
Questa caratteristica fa di JSON un linguaggio ideale per lo scambio di dati.
JSON è basato su due strutture: un insieme di coppie nome/valore, realizzato in diversi linguaggi come un oggetto, un record, uno struct, un dizionario, una tabella hash, un elenco di chiavi o un array associativo, e un elenco ordinato di valori, realizzato spesso con un array, un vettore, un elenco o una sequenza.
In JSON un oggetto è una serie non ordinata di nomi/valori. Un oggetto inizia con una parentesi graffa sinistra “{“ e finisce con una parentesi graffa destra “}”. Ogni nome è seguito dai due punti e la coppia di nome/valore sono separata da una virgola.
Invece per una serializzazione più vicina ai software di calcolo tabellare abbiamo deciso di appoggiarci ai file CSV comma-separated values, ovvero un formato di file basato su file di testo tipicamente utilizzato per l’esportazione di una tabella di dati.
In questo formato, ogni riga della tabella (record) è normalmente rappresentata da una linea di testo, che a sua volta è divisa in campi (le singole colonne) separati da un apposito carattere separatore, ciascuno dei quali rappresenta un valore.
La conversione tra questi due formali avviene nel backend quando richiesta dall'utente tramite un apposito pulsante nel front end e viene scaricato un file in maniera automatica.

1.3 Front-End

Il Front-end è il lato dell'applicazione collocato sul client e gestisce le interazioni tra questo e il server, acquisendo dati e gestendo le richieste del client, così da fornire i dati al backend.
I principali punti di forza del nostro front-end sono la sua leggerezza e la sua responsività che lo rendono fruibile praticamente da ogni dispositivo.

1.3.1 Template Client Side - Handlebars.js

Il template client-side consiste nel creare template pensati per essere eseguiti dal client (solitamente un interprete incluso nel browser) ed è usato per superare i limiti di architetture gestite completamente da lato server, come la necessità di ricaricare completamente la pagina con ciascuna interazione.
Il Client Side Templating permette quindi di creare applicazioni "fat client" che donano parte di logica al client senza richiedere necessariamente l'intervento del server.

Handlebars è uno dei tanti linguaggi usati per il templating e sfrutta un template assieme ad un oggetto di input per generare una pagina HTML.
Un template generato con Handlebar è quindi scritto come un testo in cui sono comprese delle espressioni comprese tra parentesi graffe che verranno poi sostituite con valori dati dall'oggetto di input.

1.3.2 jQuery

jQuery è una libreria JavaScript per applicazioni web che consente di semplificare la selezione, la manipolazione e la gestione degli eventi permettendo agli sviluppatori JavaScript di astrarre le interazioni a basso livello con i contenuti delle pagine HTML. Il framework fornisce metodi e funzioni per gestire al meglio aspetti grafici e strutturali come posizione di elementi o effetto di click su immagini, mantenendo la compatibilità tra browser diversi e standardizzando gli oggetti messi a disposizione dall'interprete JavaScript del browser.

1.3.3 Framework - Bootstrap

La scelta dell'utilizzo di un framework è dovuta alla nostra necessità di avere un'interfaccia chiara, pulita e di facile lettura da parte dell'utente senza nessuna grafica particolarmente creativa.

Il framework Bootstrap è una raccolta di strumenti liberi per la creazione di applicazioni per il Web contenente modelli di progettazione basati su HTML e CSS, sia per la tipografia che per le varie componenti dell'interfaccia, come moduli, pulsanti e navigazione, così come alcune estensioni opzionali di JavaScript. Bootstrap è compatibile con tutti i principali browser supportando il responsive web design, che regola in maniera dinamica il layout delle pagine web tenendo conto le caratteristiche della pagina e del dispositivo. Grazie a questo framework il nostro flusso di lavoro di styling produttivo è stato veloce e facilmente gestibile.

1.3.4 Icon - Font Awesome

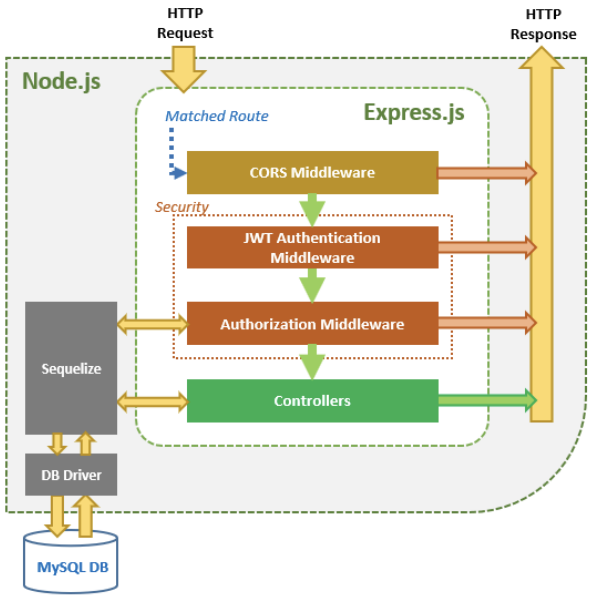
Font Awesome è un Icon Fonts, ovvero un font che contiene delle icone al posto dei classici caratteri tipografici. Viene inserito nella nostra RIA attraverso un Content Delivery Network (CDN) che lo distribuisce ai nostri client in modo da minimizzare il carico di lavoro sul server e ottimizzare i tempi di risposta.

Abbiamo scelto di utilizzare un Icon Fonts perchè le icone aiutano gli utenti ad orientarsi, esplicitano la funzione degli elementi e fanno in modo che siano comprensibili senza bisogno di parole.

Font Awesome

1.4 Sicurezza

La sicurezza è una componente molto importante di qualsiasi sistema informativo su diversi punti di vista: non solo è importante proteggere i dati personali degli utenti, ma bisogna anche implementare protezioni da possibili attacchi ai dati che potrebbero compromettere il funzionamento del sistema.
Architettura del sistema con focus sulla sicurezza:



1.4.1 Secure Header - Helmet.js

Helmet.js è un utile modulo di Node.js che fornisce protezione di default agli header HTTP restituiti dalle applicazioni create con Express.

Gli header HTTP sono una parte importante del protocollo HTTP, contenendo importanti dati della HTTP request/response usati per lo scambio di informazioni tra client e server. Solitamente gli header sono invisibili agli user, tuttavia possono lasciare scoperte informazioni sensibili sull'applicazione, come informazioni sul server provider, è quindi giusto applicare protezioni da attacchi che potrebbero compromettere la sicurezza.

1.4.2 Request Verify - Express Validator

Express Validator è un insieme di middleware express.js a supporto di funzioni di validazione e sanificazione a lato server che effettua la validazione dei dati prima che venga eseguita la funzione di callback della route, evitando l'inserimento di dati erronei e malevoli nel caso riescano ad aggirare il controllo a lato client.
Una volta incluso nel progetto, il modulo permette quindi di applicare diversi tipi di controlli alle diverse parti del body di una request, come controllare che una certa informazione sia di un dato tipo (numerico, booleano, JSON o una mail) o che abbia certe caratteristiche (come la lunghezza)

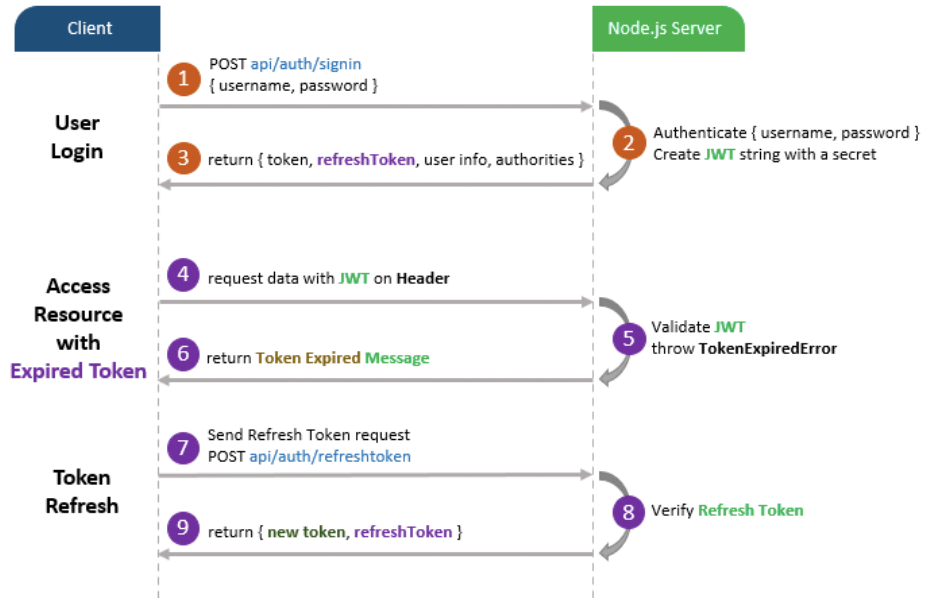
1.4.3 Password Hashing - Bcrypt

Usando una password per l'accesso al sistema è ormai uno standard far si che queste non siano salvate in maniera diretta sul database, ma che invece queste vengano usate per generare un hash, ovvero una stringa di dimensione costante (indipendentemente dalla lunghezza della password originale) che corrisponde in maniera univoca alla password inserita per inserirla nel database al posto della password stessa.
In questo modo se non si è in possesso della funzione che ha generato l'hash è molto difficile ottenere la password originale. Per implementare questa funzionalità abbiamo usato un modulo di Node.js, bcryptjs, che implementa la funzione di hash bcrypt.

Bcrypt si basa sulla cifratura Blowfish, incorpora nel sua hash un salt per proteggere la password contro attacchi brute-force con hash tipici.
Inoltre bcrypt è una funzione adattiva: con il tempo il conteggio dell'iterazione può essere aumentato per renderla più lenta, in modo da essere resistente ad attacchi di forza bruta anche con capacità computazionale crescente.

1.4.4 **J**son **W**eb **T**oken

Il Json Web Token è un formato standard per la creazione di dati con firma e crittografia opzionali il cui payload contiene un file JSON con una serie di asserzioni. Questo genera quindi dei token protetti tramite una chiave privata o pubblica a seconda dei casi, spesso generati dopo la fase di login e salvati localmente, e sono solitamente utilizzati per controllare l'autorizzazione degli utenti all'accesso di pagine o contenuti.
Quando il client cerca di accedere a route o risorse protette, il JWT viene inviato per il controllo tramite l'header "Authorization", sfruttando un meccanismo di autenticazione stateless che controlla direttamente le autorizzazioni del client. Il JWT contiene direttamente tutte le informazioni necessarie, riducendo quindi la necessità di ripetere più volte query simili nel database.



2. Amazon Alexa

Amazon Alexa è un assistente personale intelligente sviluppato dall'azienda statunitense Amazon, in grado di interpretare il linguaggio naturale e dialogare con gli umani fornendo informazioni di diverso tipo ed eseguendo differenti comandi vocali. Alexa possiede diverse funzioni native e consente l'estensione delle proprie funzioni tramite l'attivazione di applicazioni di terze parti chiamate Skill.

2.1 Proactive Events API

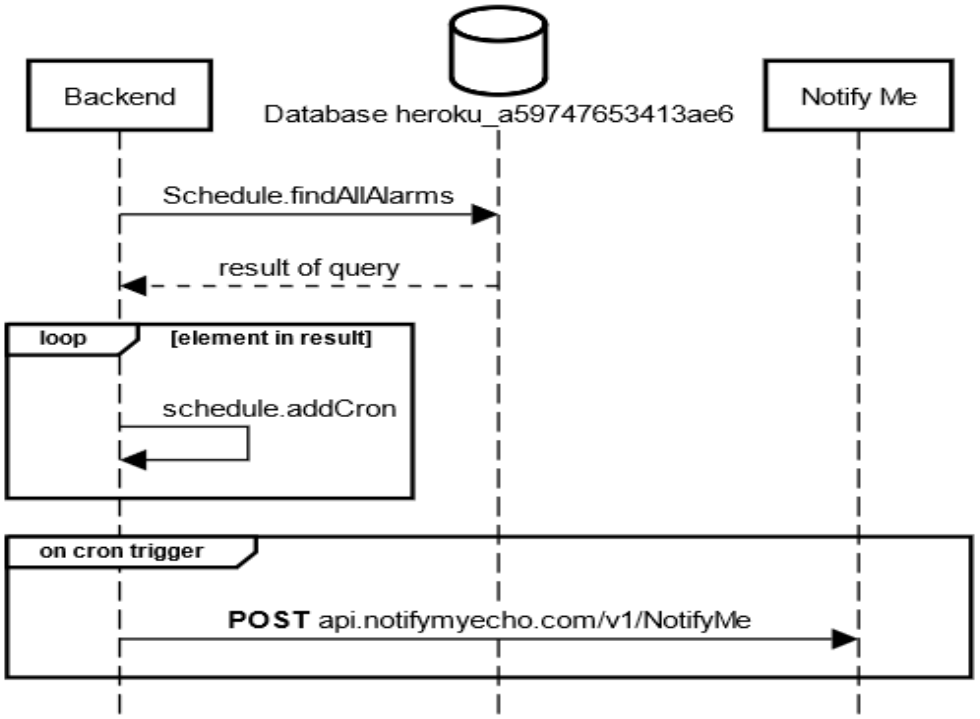
I Proactive Events API possono essere utilizzati per inviare eventi specifici al dispositivo Alexa senza che questi siano esplicitamente richiesti dall'utente. Non appena questi sono ricevuti, Alexa procede a riportare l'informazione agli utenti selezionati per ricevere gli eventi stessi.
La notifica all'utente avviene mediante l'emissione di un suono a volume regolabile solo dall'utente e l'accensione del Ring con una luce gialla a effetto pulsazione, cosicché anche dispositivi non dotati di schermo possono mostrare la presenza di una notifica.
Quando l'utente pronuncia la frase "Alexa, hai una notifica?", il dispositivo pronuncerà il messaggio detto passato tramite API, secondo uno schema preimpostato non personalizzabile ma in aggiornamento periodico.

2.2 NotifyMe API

La Skill "Notify Me" sviluppata da Thomptronics è una delle skill attivabili sui dispositivi Alexa che permette di inviare notifiche personalizzate al dispositivo, tramite le Proactive Events API.
Una volta attivata infatti la skill fornisce un access code univoco per il dispositivo che sarà poi utilizzato come destinatario delle notifiche che si vogliono creare

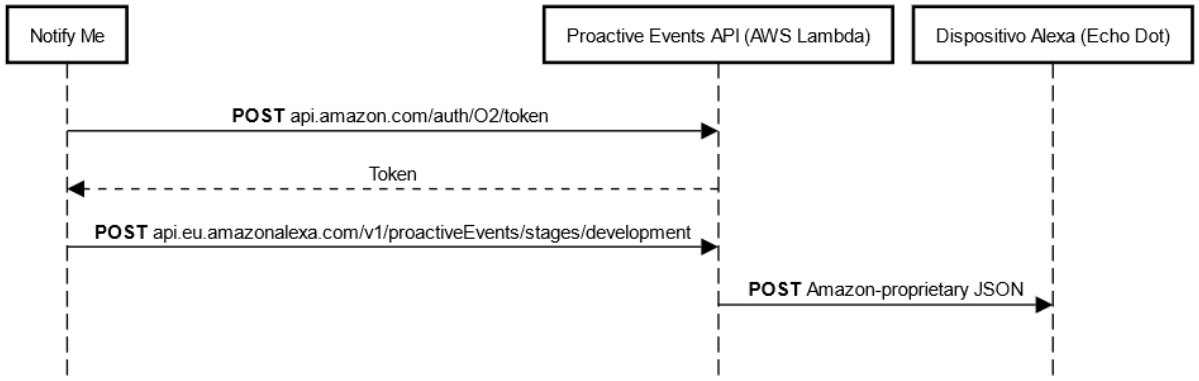
2.3 Interazioni tra backend e API

2.3.1 Comunicazione backend-Notify Me



All'avvio del server, il backend avvia la funzione `Schedule.findAllAlarms`, che esegue una query sul database, grazie alla quale ottiene un oggetto contenente una lista con tutte le informazioni sui diversi promemoria salvati nel database, ovvero id, messaggio, cron (trigger temporale) , categoria, id del dispositivo Alexa e nome e cognome del paziente.
Il backend quindi itera sugli elementi di tale lista eseguendo la funzione `Schedule.addCron` per ciascuno di essi: a ciascun promemoria ora è legata una funzione `scheduleJob`, funzione compresa nel `package node-schedule` che esegue il proprio contenuto basandosi su un sistema cron-like.
Ora all'attivarsi di un cron il backend genera un JSON contenente la notifica (solitamente il messaggio legato al promemoria con alcune informazioni aggiuntive) e l'access Code al dispositivo Alexa per poi fare una `http.request` di tipo POST che invia la notifica alla Skill Notify Me.

2.3.2 Comunicazione Proactive Events API-Notify Me



La prima parte gestisce l'autenticazione, con la POST iniziale Notify Me ottiene il token.
Questa POST usata ha un contenuto del body di tipo x-www-form-urlencoded con i seguenti parametri :
`grant_type: client_credentials` // richiede prova dell' autorizzazione di tipo client_credentials
`client_id: …` // salvato nel database come prima parte di idAlexa
`client_secret: ...` // salvato nel database come seconda parte di idAlexa
`scope: alexa::proactive_events` // il motivo della richiesta token

La seconda post avviene se l'autenticazione è andata a buon fine, ha proprio lo scopo di notificare il messaggio. L' Header della richiesta contiene i seguenti parametri;
`Authorization:...` //Riporta il tuo token
`Content-Type:application/json`

Mentre il Body è un JSON con una struttura analoga alla seguente

```
{
    "timestamp": "...", // indica quando è stato creato il file JSON
    "referenceId": ..., // id unico dell' istanza
    "expiryTime": "...", // indica la durata della validità dei dati
    "event": {
      "name": "AMAZON.MessageAlert.Activated",
      "payload": {
                  "state": { // Oggetto che descrive lo stato del messaggio
                    "status": "UNREAD", // Stato iniziale non letto
                    "freshness": "NEW" // Messagio Nuovo
                  },
```

```
                    "messageGroup": {
                      "creator": {
                        "name": "Notify Me"
                      },
                      "count": 1 // Numero messaggi nel gruppo, raggruppati per autore
                    }
                  }
                },
        "relevantAudience": { // Destinatario di questo evento
          "type": "Unicast", // Unicast è usato per ottenere informazioni contenti
                                l'userId relativo per eventi individuali
          "payload": {
               "user": "..." // Il valore di userId a cui è indirizzato l'evento
          }
    }
}
```

3 Conclusioni

Il progetto realizzato corrisponde appieno al nostro obiettivo, ovvero la realizzazione di un'interfaccia semplice che permettesse di impostare e pianificare facilmente una serie di promemoria recapitati al paziente in maniera efficace e regolare tramite Alexa, così da permettere un supporto ai pazienti lasciando che essi mantengano una propria indipendenza.

Il progetto quindi si presenta come un'ottima piattaforma di partenza a future estensioni, sia per quanto riguarda le funzionalità dell'interfaccia, come maggiori opzioni durante la creazione di promemoria, una sicurezza e controllo dei dati migliorati, sia per quanto riguarda la comunicazione con il paziente, ad esempio con un supporto ampliato tramite comandi vocali più complessi, la creazione di una Skill Alexa dalle maggiori funzionalità o anche potersi configurare con dispositivi più avanzati per ottenere dati sul supporto direttamente dal paziente

4 References

4.1 Documentazioni Ufficiali per Alexa

- Proactive Events Api: https://developer.amazon.com/en-US/docs/alexa/smapi/proactive-events-api.html
- Proactive Events Schemas: https://developer.amazon.com/en-US/docs/alexa/smapi/schemas-for-proactive-events.html
- Notify Me Skill by Thomptronics: https://www.thomptronics.com/about/notify-me

4.2 Tools utilizzati

- MySQL Workbench: https://www.mysql.com/it/products/workbench/
- IntelliJ IDEA https://www.jetbrains.com/idea/
- PostMan https://www.postman.com/company/about-postman/