UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPRATIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

"SVILUPPO DI UN'APPLICAZIONE WEB, CON SERVIZIO DI AUTENTICAZIONE TRAMITE PROTOCOLLO OAUTH2"

Relatore: Prof./ Dott Nanni Loris

Laureando: Alexandru Tiberiu Vilcu

ANNO ACCADEMICO: 2022-2023

Data di laurea: 16/03/2023

Ai miei genitori che mi hanno sostenuto,
a mia sorella a cui voglio un mondo di bene,
a Romeo e Daniel che sono i miei amici più stretti dall'infanzia,
a Delia che mi è stata vicino nei momenti difficili,
a tutti gli amici e persone che mi hanno aiutato a crescere in questo percorso e nella vita di tutti i giorni.

Indice

1 Informazioni Generali **1**

1.1 Struttura ospitante 1

1.2 Obiettivi formazione 1

2 Introduzione **3**

2.1 Applicazione sviluppata 3

3 Strumenti di sviluppo **7**

3.1 Jhipster 7

3.1.1 Architetture possibili 8

3.1.2 Vantaggi e svantaggi tra monolita e microservizi 9

3.1.3 Scelta dell'architettura: 9

3.2 Framework di sviluppo 9

3.3 ide utilizzati 10

4 Spring Boot **11**

4.1 Aspetti sull'ottimizzazione dello sviluppo 11

4.1.1 Injection 11

4.1.2 Occultamento delle servlet 12

4.1.3 Srping MVC 13

4.1.4 ORM e Hibernate 13

4.1.5 CRUD e JPA 14

5 Angular **17**

5.1 Panoramica generale 17

5.2 funzionalità 17

5.3 Reactive programming 18

5.3.1 Promise e observable 18

5.4 Vantaggi e confronto nell'utilizzo di Angular 19

6 API Rest **21**
6.1 Panoramica 21
6.2 Architettura Rest 21
6.3 Impiego in questa tipologia di applicazioni 22

7 Autorizzazione e Autenticazione **23**
7.1 Distinzione tra autorizzazione e autenticazione 23
7.2 Protocollo Oauth2 24
7.2.1 Parti coinvolte 24
7.2.2 Panoramica generale 24
7.2.3 JWT 25
7.2.4 Problematiche risolte 26
7.3 implementazione 28
7.3.1 Creazione di un id client Google 29
7.3.2 Realizzazione del Front End Angular per il log in 30
7.3.3 Realizzazione del Back End Spring Boot 31

8 Docker **33**
8.1 Container e Virtual Machine 34

9 Conclusioni **35**

References **37**

1 Informazioni Generali

Nome e cognome tirocinante: Alexandru Tiberiu Vilcu
Matricola: 1224269
Struttura ospitante: Gruppo Scai
Tutor aziendale: Bledar Gogaj
Periodo: dal 14/11/2022 al 20/1/2023

1.1 Struttura ospitante

Figura 1.1: logo

Gruppo Scai è una società con una rete di aziende in Italia, le quali si occupano di consulenza informatica in vari ambiti, uno di questi è il settore finanziario in cui sono stato introdotto per lo sviluppo, il rilascio e il mantenimento di applicazioni web.

1.2 Obiettivi formazione

Il percorso teorico e pratico mira a formare la figura del Full Stack Developer.

- lato front end, prendere conoscenza di Javascript, React e Angular
- lato back end, attraverso lo studio, e l'applicazione pratica del linguaggio Java, dei framework Spring MVC e Spring Boot per la costruzioni di servizi RESTful

L' Obiettivo è quello di sviluppare le competenze necessarie per:

- creare applicazioni Web utilizzando un database
- creare e distribuire API web basate framework spring
- creare applicazioni FE con javascript di ultima generazione e framework React e/o Angular

Modalità:

- approfondimenti teorici
- pratica guidata
- partecipazione attiva in un progetto interno dell'Azienda

2 Introduzione

Nella tesi si discuterà di Spring Boot e Angular come framework principali per lo sviluppo di applicazioni web. Queste sono integrate in Jhipster, che è un generatore di codice. Entrambi i framework sono molto popolari e molto diffusi per la loro scalabilità, ma anche perchè permettono di sviluppare software utilizzando techinche di sviluppo agile, grazie all'alto livello di programmazione e alla continua ottimizzazione delle librerie a disposizione.

2.1 Applicazione sviluppata

Utilizzando i framework appena citati con i relativi ambienti di sviluppo, tra cui Eclipse EE (enterprise edition) per il lato back-end e visual studio code per il lato front-end, è stata sviluppata un'applicazione con un'utilità interna all'azienda.
Le due parti dell'applicazione comunicano tra di loro attraverso delle API Rest. L'applicazione deve permettere ai dipendenti della sede di Padova di prenotare una postazione di lavoro, scegliendo il giorno, il piano e la postazione (figura 2.2).

Ogni dipendente può accedere a tale applicazione registrandosi con il proprio account aziendale tramite il log in di Google(figura 2.1), oppure nel caso di organizzatori e tecnici, questi potranno accedere con le credenziali personali.

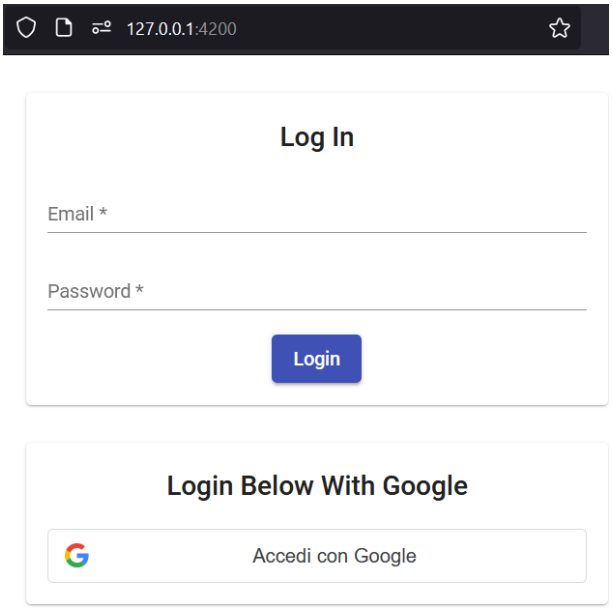

Figura 2.1: Login di Google

L'applicazione permette di verificare il personale presente in base a criteri facoltativi i quali sono:

- una specifica email, con cui identificare una persona
- una data specifica

A seconda dei criteri scelti l'applicazione effettua una ricerca e mostra una lista per tale giorno.
Se non si inserisce alcun criterio per la ricerca, viene restituito un range temporale di prenotazioni dal giorno odierno, nel caso della selezione solamente della data verrà mostrata la lista delle prenotazioni da tale data, nel caso della selezione solamente della persona interessata, verrà mostrata una lista di prenotazioni dalla data odierna, nel caso della compilazione di tuti i campi verranno mostrate le prenotazioni di una specifica persona a partire da una determinata data (figura 2.3).

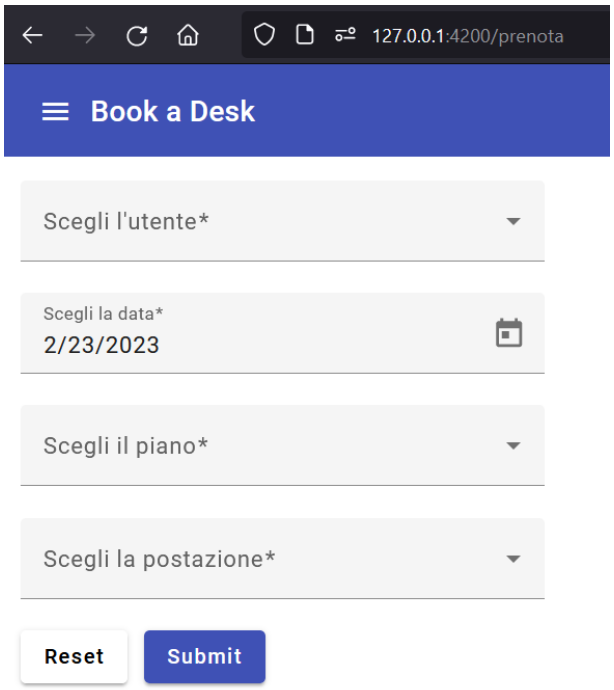

Figura 2.2: Pagina di prenotazione

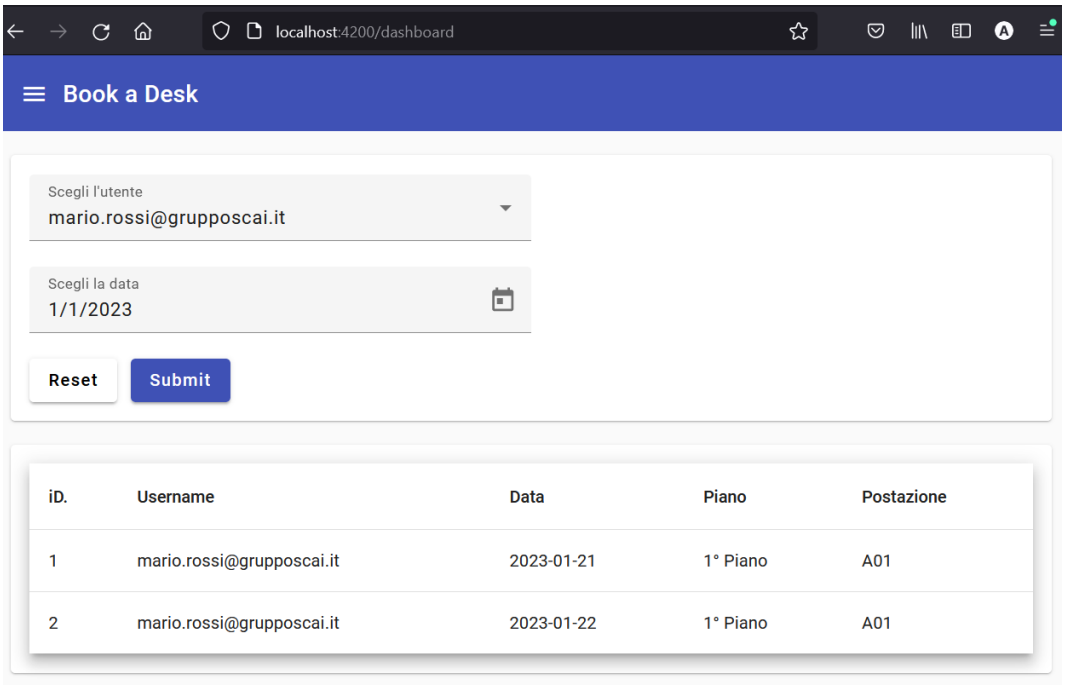

Figura 2.3: Pagina di ricerca / dashboard

3

Strumenti di sviluppo

3.1 Jhipster

Aspetti princiali:

Jhipster[7] è un generatore di codice per lo sviluppo di applicazioni web che utilizzano i linguaggi Java e Javascript, andando così a generare un'intera applicazione scegliendo tra i vari framework disponibili.
Per il lato front-end si possono utilizzare Angular, React o Vue.js, per il lato back-end Spring Boot.
L'applicazione è molto personalizzabile e permette di scegliere gli aspetti principali i quali sono:

- architettura
- tipologia e linguaggio del database
- tipologia di autenticazione
- strumento di building per l'applicazione
- strumenti aggiuntivi per le varie tiplogie di testing

Saranno trattati solo gli aspetti più importanti riguardanti lo sviluppo di questa applicazione i quali sono l'utilizzo di un'architettura per ottenere una buona scalabilità, l'utilizzo del protocollo OAuth2 per l'autenticazione degli utenti.
Per quanto riguarda la tipologia del database e il linguaggio utilizzato, è stato scelto il database relazionale con linguaggio MySql per familiarità.

Lo strumento di building rimane Maven per scelta implementativa, tale strumento permette il building, il testint e l'esecuzione del codice sorgente di un progetto con pochi semplici comandi.

Figura 3.1: schermata di jhipster per la generazione di codice sorgente

3.1.1 Architetture possibili

Tra le architetture che abbiamo a disposizione abbiamo:

- monolitico: il front-end e il back-end giacciono nello stesso spazio di memoria, il monolita viene rilasciato ed eseguito interamente su un server
- microservizi: l'applicazione è divisa in più servizi, questi possono comunicare tra di loro, essere aggiornati e mantenuti in maniera asincrona senza dover compromettere il totale funzionamento dell'applicazione, in quanto un servizio può essere interrotto senza dover interrompere l'intero processo.
- microservizi con gateway: è molto simile all'architettura precedente, ma ha la peculiarità che il traffico proveniente dalle richieste del front-end

verso le Api del back-end viene decentralizzato e gestito da un gateway, il quale reindirizza le richieste verso le Api congrue, inversamente vale anche per le risposte delle Api verso i client, anche queste sono gestite dal gateway.

3.1.2 Vantaggi e svantaggi tra monolita e microservizi

Attualmente si preferisce utilizzare un'architettura a microservizi, in quanto un monolita che ha molte funzionalità ha un mantenimento molto costoso e la difficoltà di sviluppo aumenta con l'aumento delle funzionalità, inoltre non offre una buona scalabilità in quanto bisogna replicare lo stesso monolita su più server.
Un'architettura a microservizi ha il difetto che i servizi a volte possono non avere una netta divisione, in cambio però questi possono essere mantenuti con maggiore facilità, in quanto sono indipendenti e possono essere configurati a piacere. Questo offre una buona scalabilità in quanto si può rilasciare una Api stand alone su più server, e allo stesso tempo distribuire un'applicazione front-end su vari dispositivi i quali possono comunicare direttamente con le Api.

3.1.3 Scelta dell'architettura:

Si è deciso di utilizzare un'architettura a microservizi per una maggiore scalabilità e un migliore mantenimento, con una divisione netta tra il back-end e il front-end. Il back-end comprende le Api sviluppate in Spring Boot che comunicano con un database H2 locale utilizzato per lo sviluppo, mentre il front-end è stato sviluppato in Angular e viene eseguito in differita.

3.2 Framework di sviluppo

Come già anticipato per lo sviluppo di questa applicazione sono stati utilizzati i framework Angular e Spring boot per semplificare la stesura del codice in quanto presentano dei vantaggi nella tecnica di sviluppo agile, assieme a questi sono state implementate delle Api Restful per permettere un migliore scambio di dati.
Tali framework verranno presentati nei prossimi capitoli.

3.3 IDE UTILIZZATI

Per la stesura del codice sono stati utilizzati degli ambienti di sviluppo molto conosciuti, per il front end è stato utilizzato VSCode[1] installando il cli di angular e i moduli di node per la creazione di un front end in grado di comunicare con la rest api. Allo stesso tempo è stato utilizzato Eclipse Java enterprise edition[2], un ambiente di sviluppo gratuito in grado di fornire tutti gli strumenti necessari per sviluppare un'applicazione web, a tal proposito tramite il sito di spring initializr[3] una volta scelta la configurazione desiderata, è possibile importare il progetto maven su eclipse per poter iniziare a sviluppare una qualsiasi applicazione web.

[1] https://code.visualstudio.com/

[2] https://www.eclipse.org/downloads/packages/release/2022-12/r/eclipse-ide-enterprise-java-and-web-developers

[3] https://start.spring.io/

4 Spring Boot

Spring Boot[17] è un framework creato dopo un primo framework chiamato solo Spring[19]. Spring fornisce un framework riutilizzabile per tutte le future applicazioni, mentre il framework Spring Boot rilasciato successivamente permette di rendere un'applicazione avviabile.

4.1 Aspetti sull'ottimizzazione dello sviluppo

Con l'avvento di Spring è stato possibile suddividere i vari servizi potendoli definire in package separati, quelli che per un'applicazione semplice possono essere i modelli, i controller, la gestione delle repository, i percorsi delle API. Vedremo come Spring Boot ha semplificato lo sviluppo di applicazioni nelle sezioni successive.

4.1.1 Injection

Il concetto di Injection è stato introdotto con Spring, molto importante perchè permette di utilizzare oggetti senza doverli definire o dichiarare, per fare ciò dobbiamo dichiarare una classe come Bean o una sottoclasse di Bean.
Rispetto alla classica maniera in cui è necessario definire un oggetto con l'operatore new, con l'injection è possibile dichiarare un oggetto e definirlo autowired, e chiamare le funzioni dell'oggetto, mentre Spring Boot si occuperà di istanziare l'oggetto durante l'esecuzione.

Questo meccanismo viene applicato anche ad un livello di programmazione più alto, per esempio dichiarando una classe come Component (superclasse di Bean), che ha delle funzionalità in esecuzione come la conversione di una stringa in un oggetto data ottenuto tramite un servizio Restful. L'applicazione è in grado di utilizzare questa classe senza necessità di doverla dichiarare come autowired, ma andando direttamente ad utilizzarla quando ne necessita.

```
1  package it.itec.deskbook.config;
2
3  import org.springframework.core.convert.converter.Converter;
4  import org.springframework.stereotype.Component;
5  import java.time.LocalDate;
6  import java.time.format.DateTimeFormatter;
7
8  @Component
9  public class LocalDateConverter implements Converter<String,
       LocalDate> {
10
11     @Override
12     public LocalDate convert(final String s) {
13         return LocalDate.parse(s, DateTimeFormatter.ofPattern("yyyy-
       MM-dd"));
14     }
15 }
```

Listing 4.1: esempio di Component che converte una stringa in oggetto data

I principali Bean utilizzati sono:

- Bean
- Component
- Service
- Controller
- Entity
- SpringBootApplication

  SpringBootApplication è l'endpoint di avvio dell'applicazione.

4.1.2 Occultamento delle servlet

A differenza della classica implementazione a Servlet, per la quale era necessario scrivere molte istruzioni per diverse richieste, grazie a Spring Boot è

possibile trascurare la creazione delle servlet, le quali vengono nascoste dal framework.

4.1.3 Srping MVC

Spring MVC[22] è un framework di Spring utilizzato principalmente per lo sviluppo di applicazioni monolitiche, che utilizzano principalmente file jsp per la generazione di pagine web dinamiche. E' ragionevole utilizzarlo per qualsiasi tipo di applicazione.
MVC è diviso in tre componenti:

- model
- view
- controller

Inizialmente vengono creati dei model che sono un gruppo di oggetti fatti a immagine dei dati che devono essere gestiti dall'applicazione.
Attraverso la view è possibile personalizzare e renderizzare le varie pagine web visibili al client.
Infine il controller a seconda dei model permette la gestione dei dati con la view, come per esempio la renderizzazione nel browser di una pagina nel caso di un'operazione di GET , oppure in altri contesti come in una POST, il controller esatrappolerà i dati dalla view per utilizzarli in una fase successiva.
Allo stesso tempo il controller gestisce anche l'iterazione da una pagina ad un'altra.

4.1.4 ORM e Hibernate

Grazie ad Hibernate[10] è possibile implementare ORM (Object Relation Mapping)[14], che è un processo di persistenza che permette di mappare un oggetto Java in una tabella di un database relazionale.
Nello specifico gli oggetti che verranno mappati sono tutti dei model, in quanto abbiamo la necessità di memorizzare i dati in nostro possesso dentro un database. Il nome della tabella prende il nome dalla classe model presa in causa, mentre gli attributi della tabella vengono mappati con le variabili della classe. E' necessario implementare i metodi getters e setters delle variabili della classe model, per permettere l'accesso e la modifica dei valori degli attributi, così facendo si possono modificare i record della tabella.

```java
package it.itec.deskbook.domain;

import javax.persistence.*;

@Entity(name = "floor")
public class Floor {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  @Column(name= "flr_id")
  private Long id;

  @Column(name = "flr_name")
  private String name;

  public Long getId() {
    return id;
  }

  public void setId(Long id) {
    this.id = id;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }
}
```

Listing 4.2: dichiarazione della classe Floor mappabile da Hibernate

4.1.5 CRUD e JPA

CRUD[20] acronimo di create, read, update e delete che sono le operazioni base per la gestione persistente dei dati.
Scegliendo di utilizzare JPA[18], una interfaccia di Spring Boot per i database relazionali che utilizza Hibernate, è possibile non avere più la necessità di dichiarare query in linguaggio relazionale, ma estendendo l'interfaccia Java JPA si possono chiamare dei metodi standard, che compiono le stesse funzionalità.

Questi metodi standard risiedono nell'interfaccia di JPA, allo stesso tempo se volessimo una query non standard, è possibile implementarla dichiarando solo la firma del metodo necessario all'interno dell'interfaccia, seguendo la sintassi dell'interfaccia.
JPA gestisce autonomamente la connessione,i possibili errori ed eccezzioni che si possono verificare con il database per le varie operazioni, le quali prima dovevano essere gestite dallo sviluppatore con il framework JDBC.

```
1  package it.itec.deskbook.repository;
2
3  import org.springframework.data.jpa.repository.JpaRepository;
4  import org.springframework.data.jpa.repository.
       JpaSpecificationExecutor;
5
6  import it.itec.deskbook.domain.User;
7
8  @Repository
9  public interface UserRepository extends JpaRepository<User, Long> {
10
11   Optional<User> findByEmail(String email);
12
13   Boolean existsByEmail(String email);
14
15 }
```

Listing 4.3: esempio di interfaccia che estende JPA

5 Angular

5.1 Panoramica generale

Angular[8] è un framework sviluppato da Google che utilizza typescript. [9] Questo framework è stato ideato per velocizzare lo sviluppo front-end, anche angular utilizza il paradigma di reactive programming e MVC (model, view, controller) dove per model si intende un gruppo di oggetti fatti a immagine dei dati che devono essere gestiti dall'applicazione, la view sono le pagine html e css ed è ciò che viene renderizzato ad interfaccia utente, mentre il controller viene implementato tramite typescript ed è quello che gestisce il flusso di dati tra il model e la view. Angular è stato ideato per creare siti web single page, in pratica si può navigare in una applicazione web attraverso la stessa pagina che cambia dinamicamente, dando l'impressione all'utente finale di navigare attraverso vari percorsi dell'applicazione web. Questo meccanismo viene chiamato routing, dove l'utente attraverso il browser visualizza classicamente un URL associato per ogni pagina web, mentre effettivamente è il client angular che riproduce questo comportamento, ma di fatto è sempre la stessa pagina ad essere renderizzata.

5.2 funzionalità

Angular presenta molte funzionalità di sviluppo di cui elenchiamo alcune:

- Componenti :
  La struttura può essere creata a componenti per facilitare lo sviluppo, in

quanto per apportare modifiche si può modificare un componente specifico migliorando l'organizzazione del codice, inoltre è utile per facilitare il flusso di dati.

- Direttive :
  Le direttive sono utili per modificare il comportamento dei componenti.

- Two way data binding :
  i dati del model vengono sincronizzati con la view dell'interfaccia utente e viceversa in tempo reale.

5.3 Reactive programming

Il reactive programming[21] è un paradigma di programmazione, per la precisione si intende la programmazione che fa uso di un flusso di dati asincrono. Per implementare questo paradigma in angular si usano gli eventi, e gli observable. L'evento dal nome stesso è correlato ad un'azione dell'utente, ne esistono di molti tipi, un esempio potrebbe essere il click del mouse, la selezione di un parametro in una lista di opzioni, il posizionamento del mouse sopra un'icona, ecc. Mentre un observable permette all'applicazione, appunto di osservare un evento e comportarsi in modo reattivo a ciascuna di questa, generando così un comportamento dinamico.

5.3.1 Promise e observable

Fino al passaggio da Angular-js a quello con typescript, venivano utilizzate le promise e le callback. Le promise ritornano un singolo valore, a differenza degli observable che possono restituirne piu di uno, questo per esempio può permettere di consumare uno stream che prosegue nel tempo e non si esaurisce inizialmente con la conclusione di un evento.
Rispetto alle callback e alle promise, presenti in javascript, gli observable consentono di intercettare il valore dello stream, prima che venga eseguita la funzione indicata in subscribe. Questa funzionalità è resa possibile concatenando funzioni pipe prima della subscribe[13].

5.4 Vantaggi e confronto nell'utilizzo di Angular

Angular con il fatto di essere un framework di sviluppo per applicazioni single page, permette ad un utente di scaricare una singola pagina, la quale può essere inglobata in una applicazione distribuita, permettendo così un alleggerimento del carico e del traffico verso il server, che non dovrà più allocare le pagine web che prima venivano scaricate tramite delle GET. Così facendo il server che gestisce e alloca tutte le informazioni degli utenti, potrà riportare più dati e rispondere ad una quantità maggiore di richieste. Seguendo la nostra architettura le uniche richieste che arriveranno saranno quelle riferite alle API, sulle informazioni delle prenotazioni e per effettuare una prenotazione. Mentre in una configurazione senza angular passando da una pagina all'altra, si sarebbe dovuto scaricare la pagina a cui si fa riferimento tutte le volte, oltre che alle informazioni per popolare le pagine.

6 API Rest

6.1 Panoramica

Una API fornisce una serie di regole per cui una applicazione può accedere a delle risorse all'interno di un'altra applicazione, una API REST (Representational State Transfer) è conforme a dei principi architetturali.[11]
L'applicazione che chiede la risorsa è detta client, mentre l'applicazione che detiene la risorsa è detta server, queste nella nostra configurazione comunicano tramite protocollo HTTP.

6.2 Architettura Rest

L'architettura rest può essere sviluppata in diversi linguaggi di programmazione e supporta vari formati di dati. L'unico requisito è rispettare sei principi di progettazione:

- Interfaccia uniforme :
  definire un'interfaccia uniforme, che definisce in maniera univoca tutte le possibili richieste API per una determinata risorsa.

- Disaccoppiamento client-server :
  Le applicazioni client e server devono essere completamente indipendenti. Le uniche informazioni che l'applicazione client conosce sono gli indirizzi URI per le risorse, questo è l'unico modo per interagire con l'applicazione lato server, allo stesso modo il server è obbligato a rispondere solo alle richieste URI. Così facendo si ottiene una migliore scalabilità, affidabilità, sicurezza, manutenzione ed evoluzione delle parti.

- Stateless :
  Alle API non è consentito mantenere uno stato correlato ad uno storico delle richieste. Per cui ogni richiesta deve contenere le informazioni necessarie ad essere soddisfatta.
- Cacheability :
  Per determinate richieste a risorse se possibile si possono memorizzare nella cache le risposte, per migliorare le prestazioni.
- sistema a livelli :
  Le applicazioni client e server non comunicano direttamente, per cui è necessario che queste non devono capire se stanno o meno comunicando con l'applicazione finale.
- codice on-demand (facoltativo) :
  Solitamente le risorse inviate dal lato server sono statiche, in alcuni casi le risposte possono contenere del codice eseguibile, in quel caso il codice deve essere eseguito solo su richiesta.

6.3 Impiego in questa tipologia di applicazioni

Nel nostro caso l'interfaccia uniforme utilizza gli URI per identificare l'operazione di prenotazione e di visualizzazione delle prenotazioni, dove la prima operazione attraverso una POST acquisisce i dati necessari da inserire nel database delle prenotazioni.
La seconda operazione eseguita da una GET interroga il database in base a dei parametri ricevuti dal client e risponde con una lista di oggetti, dove ogniuno contiene le informazioni di una prenotazione.
Gli oggetti che il client con il server si scambiano sono oggetti di tipo Json. Grazie ai servizi Rest e a Spring Boot è possibile mappare i Json in oggetti Java e viceversa. L'applicazione client attraverso i json è in grado di acquisire dati e renderizzarli nelle pagine web, allo stesso tempo può inviare json per determinate richieste che richiedono informazioni aggiuntive. L'applicazione Server riceve Json che poi mapperà in oggetti Java per eseguire operazioni o per memorizzare dati, infine nel caso ne sia bisogno, risponde creando un json, il quale contiene i dati necessarri all'applicazione client di continuare.

7

Autorizzazione e Autenticazione

7.1 Distinzione tra autorizzazione e autenticazione

Un'applicazione Spring Boot può essere configurata con un package di sicurezza per configurare l'autorizzazione e l'autenticazione di ogni utente.
L'autorizzazione permette ad un utente di essere autorizzato ad usare la rispettiva applicazione o un servizio, di norma esiste un database con gli utenti che sono autorizzati.
L'autorizzazione[15] di un utente è anche correlata al ruolo di un utente che ha in un'applicazione, i più comuni sono: user e admin. In base al ruolo che ricopre un utente questo può accedere a percorsi differenti, nel nostro caso i percorsi sono URI essendo l'applicazione server una API REST. Questo per permettere solo ad utenti autorizzati di modificare dati o impostazioni sensibili dell'applicazione server, per non compromettere il suo funzionamento, oppure per non invalidare i dati e le funzionalità.
L'autenticazione[1] invece è secondaria all'autorizzazione, per cui un utente che è autorizzato viene autentificato, di conseguenza è possibile tenere traccia delle operazioni che richiedono i permessi di autorizzazione, verificando l'utente che è autentificato. Così facendo si può verificare e accertare con sicurezza l'attività degli utenti.

7.2 Protocollo Oauth2

Oauth2 è un framework standard di autorizzazione [4].
Nato per sostituire il protocollo Oauth 1.0 ormai obsoleto e non retrocompatibile. Questo nuovo protocollo consente alle applicazioni di accedere a risorse protette di un servizio o di un'applicazione esterna. Oauth2 definisce un flusso di autorizzazione per applicazioni native, web, mobile.

7.2.1 Parti coinvolte

Le parti in questo modello sono diverse da quelle del classico modello client-server[6]. Le parti principali dette anche ruoli sono:

- Client :
  client non fa riferimento ad una particolare implementazione, ma ad una generica applicazione che fa richieste che necessitano una autorizzazione.
- Server delle risorse :
  il server che ospita le risorse e utilizza i token di accesso.
- Proprietario delle risorse :
  entità che posside le risorse ospitate nel server delle risorse, allo stesso tempo garantisce l'accesso alle risorse protette.
- Server di autorizzazione :
  il server che emette i token di accesso, dopo aver autentificato il proprietario delle risorse e ottenuto l'autorizzazione.

7.2.2 Panoramica generale

Nella figura 7.1 si può notare il flusso di una richiesta alle risorse che utilza il protocollo oauth2[5], l'utente detto anche proprietario delle risorse vuole accedere alle informazioni del suo account, invia una richiesta all'applicazione client, questa lo reindirizza ad una pagina di login del server di autorizzazione. Una volta ottenuta l'autorizzazione il client richiede un token di accesso detto anche jwt, il server di autorizzazione gliene fornisce uno, dopo di chè il client allega il token alle future richieste aggiungendo un header con il rispettivo valore. Nel nostro caso l'applicazione angular è il client, il server di autorizzazione è Google, mentre il server delle risorse è l'applicasione Spring Boot.

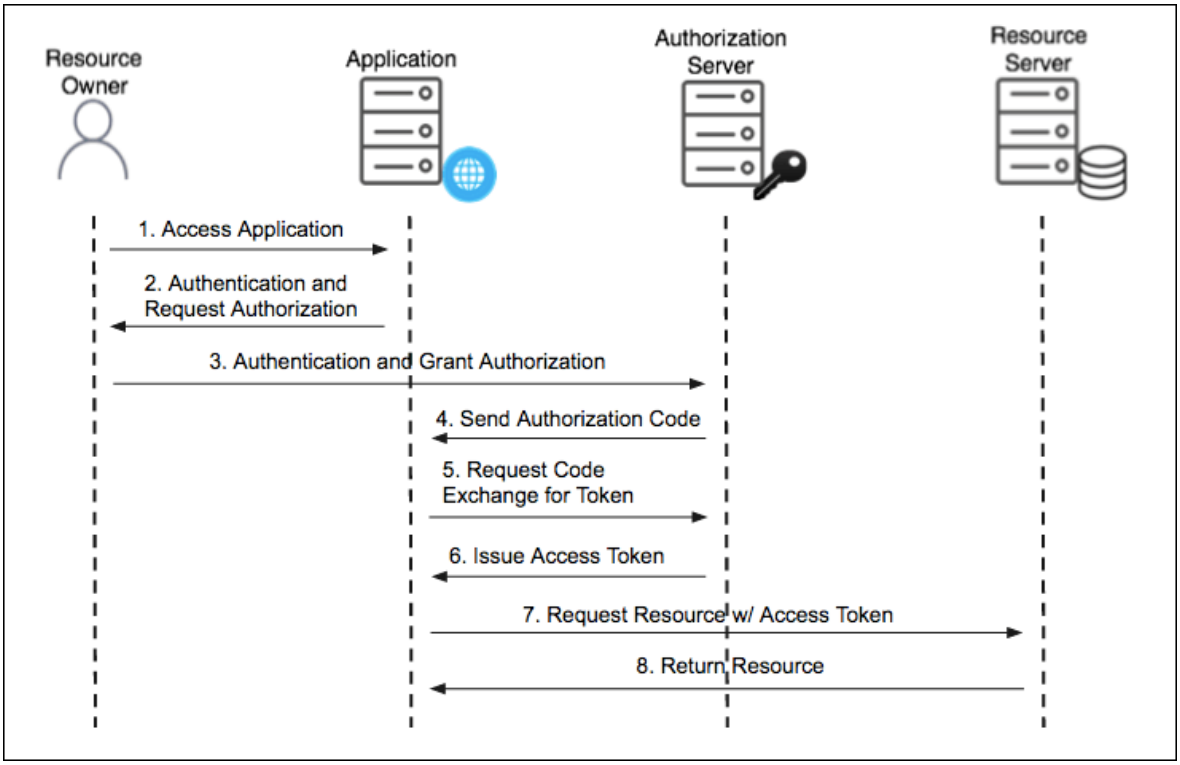

Figura 7.1: flusso di una richiesta che utilizza oauth2

7.2.3 JWT

Citando l'RFC 7519[3] possiamo confermare che un jwt acronimo di json web token, è un token composto da 3 parti, l'header, il payload e la firma. Nell'header abbiamo il tipo del token e l'algoritmo di codifica della firma o del token.
Nel payload abbiamo vari campi che vengono chiamati anche claims, queste sono le informazioni che vengono scambiate come per esempio l'issuer, colui che ha fornito il token, l'audience cioè per chi è destinato il token, il tempo di validità, e altri campi che contengono informazioni riservate. L'header e il payload sono codificati con la codifica base64 e dall'algoritmo prescelto, la struttura è delineata dall'RFC 7515[2]. Affinchè la firma funzioni è necessario utilizzare anche una chiave privata per la generazione della firma e una chiave pubblica per la sua validazione, così facendo si verifica l'autenticità della provenienza del jwt e che questo non sia stato manomesso durante la trasmissione.
Nella figura 7.2 possiamo notare un breve jwt utilizzato durante lo sviluppo e la sua decodifica.[1] Esistono anche altri standard come SWT e SAML, a differenza

[1] `https://jwt.io`

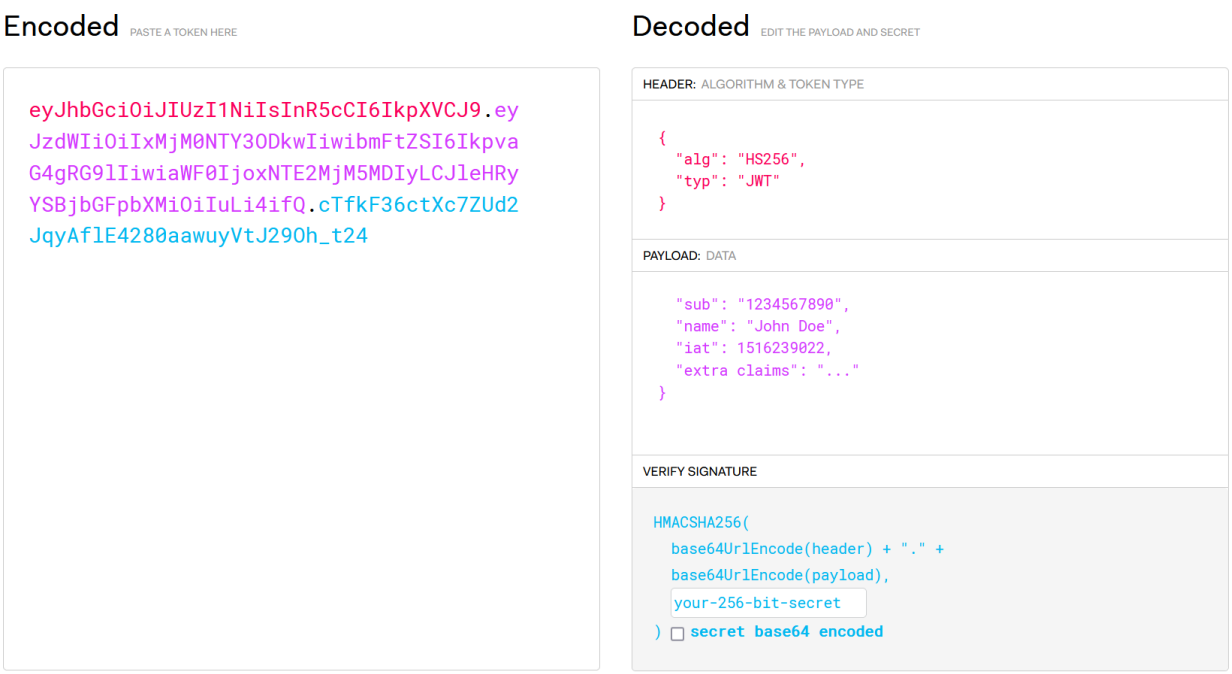

Figura 7.2: decodifica di un jwt

di SWT i token JWT e SAML possono godere della doppia chiave privata e pubblica mentre un SWT utilizza solo una chiave pubblica condivisa.
Il vantaggio di JWT rispetto a SAML sono i seguenti: un JWT utilizza i json i quali sono meno prolissi degli XML, di conseguenza hanno dimensioni ridotte, i token SAML presentano numerose falle di sicurezza, per via anche della complessità del linguaggio. Per concludere sappiamo che un token JWT è un json che ha una mappatura ad oggetto utilizzata in molti linguaggi, mentre un SAML non presenta una mappatura "naturale",tutti questi aspetti rendono preferibile l'utilizzo di un JWT.

7.2.4 Problematiche risolte

Dal libro Spring Security in Action[16] si possono ricavare diverse problematiche risolte grazie all'ausilio del protocollo Oauth2, per esempio nella figura 7.3 notiamo una delle principali problematiche che si possono incontrare utilizzando una richiesta con autorizzazione http basic, ossia il passaggio delle credenziali di accesso ad ogni richiesta, questo a primo avviso risulta una vulnerabilità molto elevata del sistema, in quanto le credenziali dell'utente sono presenti in ogni richiesta.
Un ulteriore svantaggio si presenta nel caso utilizzando l'autorizzazione http

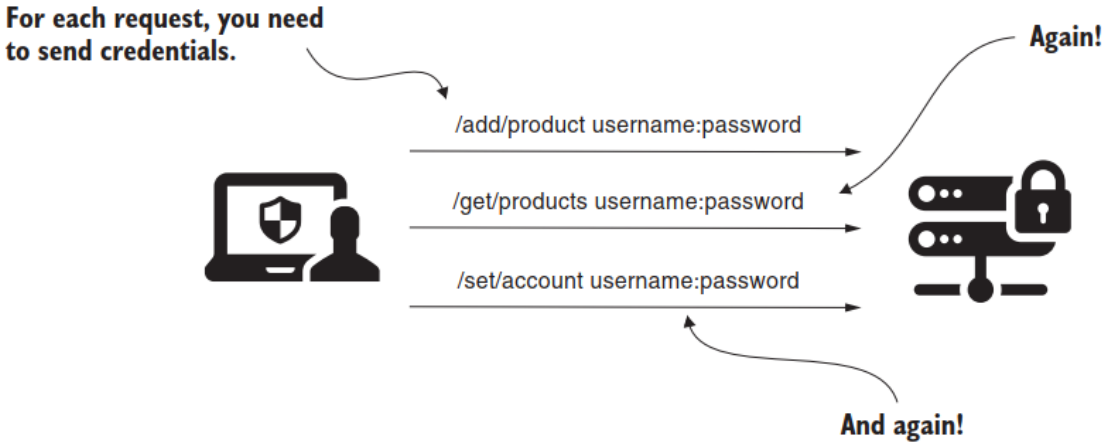

Figura 7.3

Figura 7.4

basic si abbia una organizzazione con più applicazioni distinte, in quanto come si può notare nella figura 7.4 ogni applicazione client dovrà gestire un insieme di utenti e le rispettive credenziali. Detto ciò quando un utente si autentifica su una delle applicazioni, le sue credenziali non sono presenti nelle altre applicazioni dell'organizzazione. Allo stessso tempo la registrazione su più applicazioni porta anche ad una ridondanza di utenti nei database.

Per risolvere queste problematiche partendo dalla prima, si può utilizzare il protocollo oauth2, che permette di non passare più le credenziali ad ogni richiesta, ma richiede attraverso un reindirizzamento ad una pagina web di autentificarsi tramite il server di autorizzazione, dopo di che restituirà un token il quale non conterrà le credenziali dell'utente.

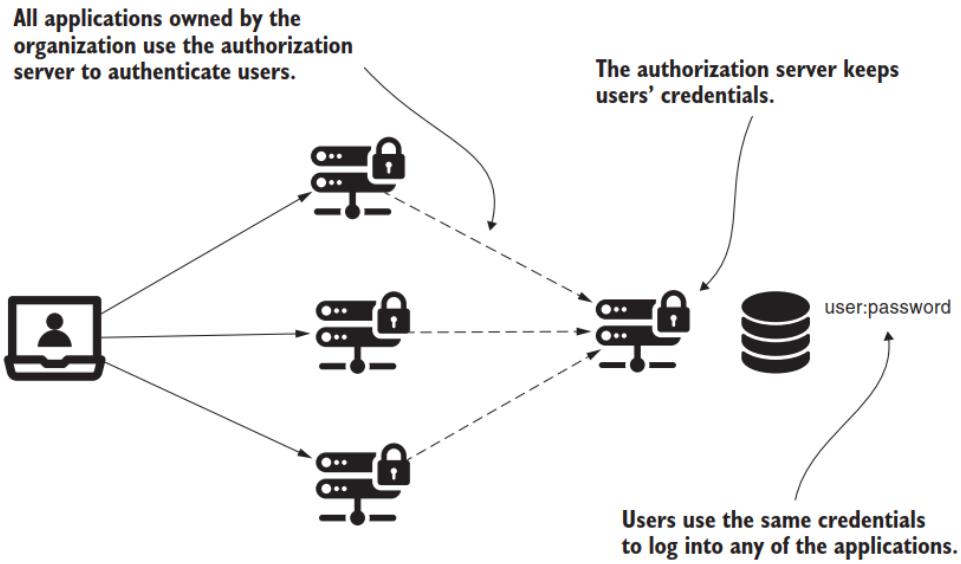

Figura 7.5

Inoltre come si può notare nella figura 7.5 le applicazioni di un'organizzazione possono sfruttare il server di autorizzazione per lo stoccaggio di utenti e usarlo come end point di autorizzazione per tutte le applicazioni dell'organizzazione, in quanto il server di autorizzazione si occuperà dell'autenticazione degli utenti.

Una problematica presente nel modello classico client-server presenta l'impedimento di limitare la durata e l'accesso a un sottoinsieme delle risorse, per esmpio il gestore di un'organizzazione non può revocare l'accesso ad un utente ma bensì deve farlo su un gruppo di utenti a seconda del loro grado di autorizzazione, detto anche ruolo. Invece con il rilascio da parte del server di autorizzazione del token di accesso, in questo si possono racchiudere varie informazioni, tra cui lo scope di informazioni a quale l'utente può accedere e la durata della scadenza del token che delimita il periodo di accesso dell'utente.

7.3 IMPLEMENTAZIONE

L'implementazione di tale applicazione necessita la configurazione e la stesura del codice dei diversi microservizi.
Inizialmente è stato necessario creare un cliend Id Google per poter usufruire del server di autorizzazione di Google.
Si è scelto di implementare un front end Angular per il log in Google e succes-

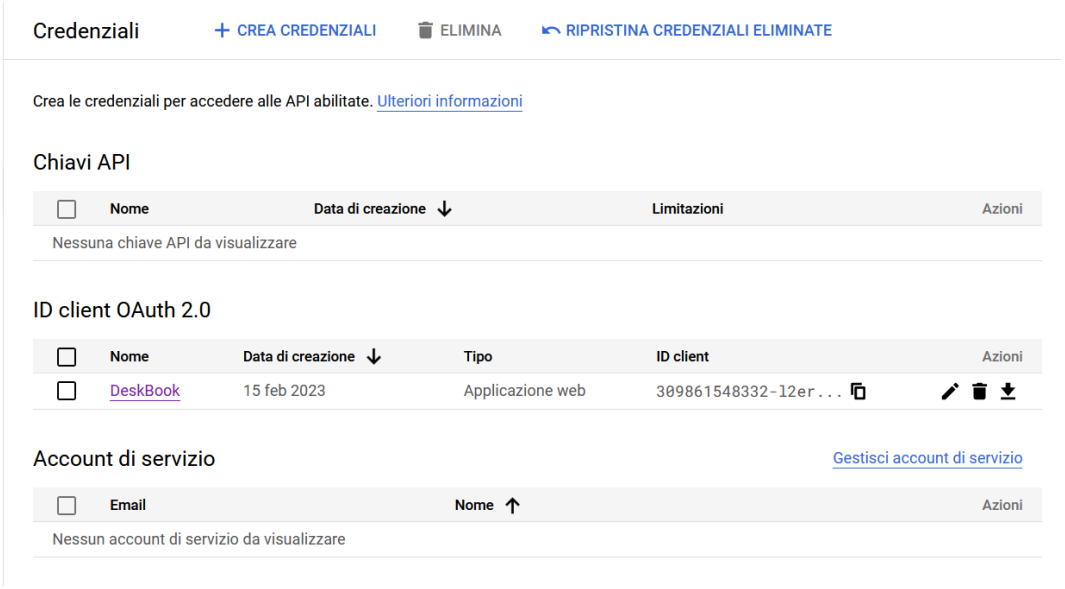

Figura 7.6: google dev console

sivamente della pagina per le prenotazioni, un back end realizzato con Spring Boot che rappresenta un server delle risorse, al quale il front end può ricevere le risorse una volta autentificatosi.
Essendo una applicazione in sviluppo, entrambe le parti verranno eseguite in locale. Si può consultare la pagina Google per le funzioni utilizzate.[2]

7.3.1 Creazione di un id client Google

Attraverso la console di sviluppatori di Google è possibile creare un client id, e inserire gli url delle applicazioni: l'url del front end per l'autorizzazione degli script, mentre gli url di reindirizzamento per la parte di back end. Non viene dato l'accesso a nessun'altro url e percorso relativo, che non appartenga alla lista e utilizzi il client id creato (figura 7.6).
Attraverso la stessa pagina si possono integrare ulteriori servizi come google maps, oppure gmail utilizzando le Api di google.

[2] https://developers.google.com/identity/gsi/web/guides/overview?hl=it

7.3.2 Realizzazione del Front End Angular per il log in

Grazie alla integrazione delle librerie javascript di Google (Listing 7.1) è possibile renderizzare il pulsante "accedi con Google"[3], riferendosi al tag div con uno specifico identificatore della pagina html, nel nostro caso "buttonDiv", inoltre è necessario fornire il client id per poter accedere alla pagina di log in di Google, e definire una funzione che gestirà la risposta contenente il token una volta ricevuto dal server di autorizzazione di Google.

```
ngOnInit(): void {
    window.onload = () => {
      google.accounts.id.initialize({
        client_id: this.clientid,
        callback: this.handleCredentialResponse.bind(this),
        auto_select: false,
        cancel_on_tap_outside: true,
        ux_mode: 'popup'
      });
      google.accounts.id.renderButton(
        document.getElementById("buttonDiv"),
        { theme: "outline", size: "large" }
      );
    }
}
```

Listing 7.1: Google button rendering

Una volta autentificatosi e ottenuto un token di accesso, è possibile inviarlo al back-end assieme alla richiesta, nel caso il token non fosse valido verrà stampato su console l'errore e l'utente sarà reindirizzato alla pagina principale di log in (Listing 7.2).

```
ngOnInit(): void {
    this.logService.getAccess().subscribe(data => {
      this.dto = new MessageDto(data.message, data.picture);
      console.log(this.dto);
    }, err => {
         console.log(err);
         if (err.status === 401) {
```

[3] https://developers.google.com/identity/gsi/web/guides/display-button?hl=it

```
            this.logOut();
        }
    });
}
```

Listing 7.2: invio della request e visualizzazione della response su console

7.3.3 Realizzazione del Back End Spring Boot

Per quanto riguarda il Back End si possono utilizzare i principali Bean di spring boot assieme alle librerie Google per autentificare i token[4] , principalmente la richiesta attraversa la filter chain di Spring Boot dove vengono chiamati i vari componenti per validare il token e determinare l'accesso alle risorse, nel nostro caso la classe che implementa l'interfaccia AuthenticationProvider è stata implementata come nel Listin 7.3.
Dopo di chè una volta validato il token, Spring boot restituisce i dati richiesti dal client all'url della rest api, nella figura 7.7 è rappresentato il token inviato al back end e le informazioni che sono state passate dalla api Rest al client utilizzando un oggetto MessageDTO da entrambe le parti.

```
@Component
public class JwtProvider implements AuthenticationProvider {
    private static final Logger logger = LoggerFactory.getLogger(
    JwtFilter.class);

    @Override
    public Authentication authenticate(Authentication authentication)
    throws AuthenticationException {
        TokenModel tokenModel = (TokenModel) authentication;
        NetHttpTransport transport = new NetHttpTransport();
        GsonFactory gsonFactory = GsonFactory.getDefaultInstance();
        GoogleIdTokenVerifier verifier = new GoogleIdTokenVerifier.
    Builder(transport, gsonFactory)
                .setAudience(Collections.singletonList(clientId))
                .build();
        try {
```

[4] https://developers.google.com/identity/gsi/web/guides/verify-google-id-token?hl=it

```
14              GoogleIdToken idToken = verifier.verify(tokenModel.
    getToken());
15              if(idToken != null) {
16                  GoogleIdToken.Payload payload = idToken.getPayload();
17                  String name = (String) payload.get("name");
18                  String picture = (String) payload.get("picture");
19                  UserModel userModel = new UserModel(name, picture);
20                  System.out.println("name : " + name);
21                  return userModel;
22              }
23          } catch (Exception e) {
24              logger.error("Authenticate fail: {}", e.getMessage());
25          }
26          return null;
27      }
28      @Override
29      public boolean supports(Class<?> authentication) {
30          return authentication.isAssignableFrom(TokenModel.class);
31      }
32  }
```

Listing 7.3: AuthenticationProvider per validazione token

Figura 7.7: Console browser con token di prova inviato

8

Docker

Docker[1] è una tecnologia di virtualizzazione ed è uno strumento fondamentale per lo sviluppo di codice sorgente portabile.
Uno dei principali problemi di sviluppo si manifesta quando un progetto perfettamente funzionante in un ambiente di sviluppo, viene spostato nel reparto produzione dopo un rilascio. L'ambiente utilizzato per lo sviluppo può essere con una probabilità molto alta, diverso dalla macchina su cui l'applicazione verrà eseguita, di conseguenza ci sono molti fattori che potrebbero incidere su un funzionamento non ottimale, oppure non riuscire nemmeno ad eseguire l'applicazione, questi fattori sono in genere le dipendenze e la configurazione del codice per essere eseguito correttamente.
Per esempio si potrebbe avere un sistema operativo diverso, una configurazione della rete diversa, un percorso diverso da quello originale dove il codice viene memorizzato, potrebbe essere presente una versione differente dei linguaggi utilizzati.
Grazie Docker si può creare un container, il quale racchiude tutte le specifiche necessarie per essere eseguito su un ambiente diverso da quello iniziale.

[1] `www.docker.com`

8.1 Container e Virtual Machine

Per creare un ambiente congruo ad una applicazione che si sta sviluppando si potrebbe utilizzare una virtual machine, questa era una delle prima soluzioni per riuscire ad eseguire più applicazioni sullo stesso server, emulando sistemi operativi diversi dove ogniuno ospitava un'applicazione.
Nel caso avessimo microservizi di dimensioni ridotte, creare delle macchine virtuali per ogni microservizio risulterebbe dispendioso e macchinoso sia in termini di costo che di tempo, in quanto per ogni macchina virtuale dobbiamo emulare anche un sistema operativo, questo porta ad una notevole occupazione di memoria.
Inoltre nel caso si voglia aggiornare uno dei microservizi, sarà necessario aggiornare anche la macchina virtuale con le dipendenze necessarie.
Grazie a Docker si può creare l'immagine di un microservizio che contiene le varie dipendenze e configurazioni del codice sorgente, così che una volta caricata l'immagine nel server, venga creato un container, il quale racchiude il microservizio con tutto ciò che è necessario per la sua corretta esecuzione.
A differenza di una macchina virtuale, un container non deve emulare un sistema operativo, perchè questo può condividere con altri container lo stesso sistema operativo, il che rende necessario avere lo stesso sistema operativo[12].
Questo svantaggio si può ovviare utilizzando entrambe le tecnologie insieme, utilizzando più macchine virtuali per emulare sistemi operativi diversi e in questi emulare i rispettivi container congrui.

9

Conclusioni

Abbiamo visto quali sono i vantaggi che porta ad avere un sistema a microservizi che utilizza il protoccolo oauth2 per la comunicazione e l'accesso alle risorse. Tuttavia abbiamo tralasciato il fatto che implementare un'applicazione a microservizi con un protocollo oauth2 è molto dispendioso in termini di tempi e costi, per quanto possa essere diffuso come sistema non tutti ne sono al corrente, questo implica ancora una volta la necessità di studiare l'implementazione in dettaglio, conoscendo e rimanendo aggiornati sui framework per mantenere un certo livello di sicurezza per gli utenti. Detto ciò si può ovviare al dispendio di risorse aziendali utilizzando sistemi di centralizzazione della gestione degli utenti come keycloak oppure okta, i quali permettono di gestire l'utenza, fornire un SSO con un qualsiasi provider come Google. Questo sistema permette di trascurare quello che è l'implementazione del protocollo oauth2 lato server delle risorse e anche client, e concentrare lo sviluppo sugli aspetti principali dando così un ulteriore surplus a quello che è lo sviluppo agile.

References

[1] Internet Engineering Task Force (IETF). «Hypertext Transfer Protocol (HTTP/1.1): Authentication». In: (2014). [online; adressed February-2023]. URL: https://www.rfc-editor.org/rfc/rfc7235.

[2] Internet Engineering Task Force (IETF). «JSON Web Signature (JWS)». In: (2015). [online; adressed February-2023]. URL: https://www.rfc-editor.org/rfc/rfc7515/.

[3] Internet Engineering Task Force (IETF). «JSON Web Token (JWT)». In: (2015). [online; adressed February-2023]. URL: https://www.rfc-editor.org/rfc/rfc7519.

[4] Internet Engineering Task Force (IETF). «The OAuth 2.0 Authorization Framework». In: (2012). [The OAuth 2.0 Authorization Framework; online; adressed February-2023]. URL: https://www.rfc-editor.org/rfc/rfc6749.

[5] Internet Engineering Task Force (IETF). «The OAuth 2.0 Authorization Framework - Protocol flow». In: (2012). [The OAuth 2.0 Authorization Framework; online; adressed February-2023]. URL: https://www.rfc-editor.org/rfc/rfc6749#section-1.2.

[6] Internet Engineering Task Force (IETF). «The OAuth 2.0 Authorization Framework - Roles». In: (2012). [The OAuth 2.0 Authorization Framework; online; adressed February-2023]. URL: https://www.rfc-editor.org/rfc/rfc6749#section-1.1.

[7] *Doing microservices with JHipster*. [online; adressed February-2023]. 2023. URL: https://www.jhipster.tech/microservices-architecture/.

[8] Google. *Introduction to the Angular docs*. [online; adressed February-2023]. 2023. URL: https://angular.io/docs.

[9] Google. *What is angular?* [online; adressed February-2023]. 2023. URL: https://angular.io/guide/what-is-angular.

[10] *Hibernate documentation*. [online; adressed February-2023]. 2023. URL: https://hibernate.org/orm/documentation/6.1/.

[11] IBM. «Api Rest». In: (2021). [online; adressed February-2023]. URL: https://www.ibm.com/it-it/cloud/learn/rest-apis.

[12] Docker Inc. *Use containers to Build, Share and Run your applications*. [online; adressed February-2023]. 2023. URL: https://www.docker.com/resources/what-container/.

[13] Andrea Merlin. «Angular, RxJs e Observable». In: (2020). [online; adressed February-2023]. URL: https://amerlin.keantex.com/angular-e-observable/.

[14] *Object relational mapping*. [online; adressed February-2023]. 2023. URL: https://hibernate.org/orm/.

[15] The Internet Society. «Hypertext Transfer Protocol – HTTP/1.1». In: (1999). [online; adressed February-2023]. URL: https://www.rfc-editor.org/rfc/rfc2616#section-14.8.

[16] Laurentiu Spilca. *Spring security in action*. Manning Publications, 2020.

[17] *Spring Boot Documentation*. [online; adressed February-2023]. 2023. URL: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/.

[18] *Spring Data JPA - Reference Documentation*. [online; adressed February-2023]. 2023. URL: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/.

[19] *Spring Documentation*. [online; adressed February-2023]. 2023. URL: https://docs.spring.io/spring-framework/docs/current/reference/html/.

[20] Maryam Sulemani. «CRUD operations explained». In: (2021). [online; adressed February-2023]. URL: https://www.educative.io/blog/crud-operations.

[21] Doug Tidwell. «An introduction to reactive programming». In: (2019). [online; adressed February-2023]. URL: https://developers.redhat.com/coderland/reactive/reactive-intro#.

[22] *Web MVC framework*. [online; adressed February-2023]. 2023. URL: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html.