UNIVERSITÀ DEGLI STUDI DI PADOVA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Implementazione sicura di microservizi in una applicazione web per il voto elettronico

---

Relatore: Carlo Fantozzi

Laureando: Samir Masato

ANNO ACCADEMICO 2020-2021

7 Marzo 2022

Ringraziamenti

Mi sento in dovere di dedicare questa pagina del presente elaborato alle persone che mi hanno supportato nella redazione dello stesso e durante il percorso di studi.

Innanzitutto, ringrazio il mio relatore Carlo Fantozzi, sempre pronto a darmi le giuste indicazioni in ogni fase della realizzazione dell'elaborato. Grazie ad esso ho avuto la possibilità di portare a termine questo percorso.

Un ringraziamento speciale va al mio Tutor di tirocinio Fabio Pallaro che mi ha aiutato nell'individuare il percorso adatto essendo sempre presente e disponibile in ogni mia necessità per il corretto svolgimento dello stage, presso l'azienda Sync Lab s.r.l.

Ringrazio tutto lo staff dell'azienda Sync Lab s.r.l, in cui ho svolto un tirocinio formativo della durata di 3 mesi e complementare alla redazione della tesi, per l'ospitalità e per le competenze acquisite sul campo.

Il ringraziamento più sentito e importante va ai miei genitori Agostino e Magidah, per il supporto in ogni mia scelta, accademica e di vita: senza di loro non avrei mai potuto intraprendere, proseguire e concludere questo percorso di studi

Indice

Premessa

Il seguente elaborato descrive il percorso di tirocinio, della durata di 250 ore, effettuato durante l'anno accademico 2020/2021, in particolare dal primo Aprile duemilaventuno al primo Settembre duemilaventuno. Durante il sopracitato arco temporale la suddivisione del carico di lavoro non è stata uniforme in quanto oltre al tirocinio ho dovuto seguire i rimanenti corsi del semestre e sostenere due sessioni d'esame, pertanto il focus è stato principalmente sui mesi di Aprile, Maggio, Agosto. Sommariamente il percorso di tirocinio è consistito nello sviluppo del Backend a microservizi di una applicazione per il voto elettronico: a tal scopo è stato necessario dedicare gran parte del monte ore allo studio degli aspetti teorici dal momento che la preparazione sugli stessi, al tempo di inzio del tirocinio, era quasi nullo. Di seguito una breve panoramica dei capitoli presenti nell'elaborato.

- Capitolo 1 Introduzione: presentazione dell'azienda, degli obiettivi e dello svolgimento del tirocinio
- Capitolo 2 Aspetti teorici: una panoramica sugli argomenti oggetto di studio necessari per lo sviluppo del progetto
- Capitolo 3 Strumenti Software complementari: i vari software necessari per la gestione del progetto
- Capitolo 4 Descizione dell'applicazione: Struttura, funzionalità e utilizzo del Backend dell'applicazione
- Capitolo 5 Conclusioni: Considerazioni finali
- Capitolo 6 Bibliografia

Capitolo 1: Introduzione

1.1 L'azienda

Sync Lab nasce come Software house tramutatasi rapidamente in System Integrator attraverso un processo di maturazione delle competenze tecnologiche, metodologiche ed applicative nel dominio del software. L'azienda, propone sul mercato interessanti quanto innovativi prodotti Software, nati nei laboratori di ricerca e sviluppo. Attraverso questi prodotti Sync Lab ha gradualmente conquistato significative fette di mercato nei seguenti settori: mobile, videosorveglianza e sicurezza delle infrastrutture informatiche aziendali. L'obiettivo finale è supportare il cliente nella realizzazione, messa in opera e governance di soluzioni IT, sia dal punto di vista tecnologico, sia nel governo del cambiamento organizzativo.

1.2 Progetto intrapreso

Scelta dell'azienda

La scelta dell'azienda è avvenuta tramite una lista di possibili enti ospitanti già convenzionati per il tirocinio fornita dall'Università di Padova. Tra le altre proposte mi ha colpito l'attinenza con il mio percorso di studi offerta da Sync Lab; di fatto è stato il fattore determinante per la decisione. In seguito agli aspetti burocratici e alla gentile disponibilità del professor Carlo Fantozzi come responsabile interno all'università e in seguito come relatore, ho contattato l'azienda rappresentata dall' ingegner Fabio Pallaro, l'attuale tutor all'interno dell'azienda che mi ha seguito dall'inizio dell'esperienza fino al completarsi della stessa.

Proposta di progetto

Durante il primo incontro effettuatosi da remoto a causa delle restrizioni in seguito alla pandemia di SARS-CoV-2 l'ingegner Pallaro mi ha proposto diversi percorsi formativi da poter intraprendere. Tra gli altri abbiamo deciso lo studio del Backend di una applicazione Web

tramite Framework Spring. La propensione per questo argomento deriva dal mio interesse di conoscere la tecnologia alla base della gestione delle risorse in ambito web e l'implementazione della stessa con focus sulla securizzazione dell'intero sistema. Tale ambito tecnologico rappresenta inoltre un'importante competenza nel mondo lavorativo.
Il progetto consiste nello sviluppo di un'applicazione web per il voto elettronico: Ciò ha richiesto di studiare tutto il sistema di Backend implementato dai vari moduli del Framework Spring, i quali verranno poi descritti nel dettaglio nella sezione dedicata alla teoria. Inoltre data la securizzazione dell'applicazione verranno presentati i protocolli di sicurezza utilizzati a tal scopo.

1.3 Organizzazione del lavoro

Inizialmente mi è stata fornita una lista di argomenti oggetto di studio da svolgere in autonomia, complice il fatto che inizialmente il tirocinio si è svolto da remoto a causa delle restrizioni governative in merito al contenimento della pandemia in corso. Con cadenza settimanale si sono svolti degli allineamenti per eventuali dubbi o difficoltà e per mantenere aggiornate le parti sul proseguimento dello studio con eventuali modifiche. Come descritto nella premessa la regolarità si è mantenuta solo nei mesi in cui non vi era la preparazione agli esami del semestre in corso. Inoltre per mantenere traccia degli argomenti trattati si è fatto uso della piattaforma Moodle Formazione Aziendale. Quest'ultima permette di spezzettare i passi fondamentali dello stage e quindi indicare quelli svolti. Successivamente, al completamento della teoria, si è passati all'implementazione del sistema; per la condivisione del codice ed eventuale revisione da parte del tutor si è utilizzato GitHub.

Obiettivi

Fin dall'inizio si sono identificati i seguenti obiettivi:

- Obbligatori
  - Acquisizione delle compentenze necessarie all'utilizzo esaustivo del Framework Spring
  - Capacità di organizzazione per lo studio in autonomia dei vari argomenti di studio
  - Implementare l'applicazione soddisfando tutti i requisiti di quest'ultima (descritti nel capitolo 3 sez. 2)

- Facoltativi
  - Implementare qualche maschera per il Frontend dell'applicazione

Capitolo 2: Aspetti teorici

2.1 Introduzione

I primi due mesi e mezzo di tirocinio si sono incentrati sullo studio dei vari argomenti teorici. Rappresentando un carico di lavoro non indifferente ha impegnato la maggior parte del periodo. Questo è stato dovuto anche all'attenzione dedicata agli aspetti di sicurezza. Di seguito una panoramica descrittiva delle tecnologie e dei protocolli studiati.

2.2 Spring Framework

Spring è un Framework opensource per lo sviluppo di applicazioni Java enterprise basato su JVM. Spring è composto da molti moduli, alcuni dei quali studiati e descritti nel seguito. I moduli sono stati progettati per fornire maggiori funzionalità alla piattaforma, la modularità garantisce la possibilità di mantenere il progetto il più snello possibile evitando l'importazione di moduli e quindi di codice inutile. Inoltre la politica di Spring prevede per natura la necessità di rendere il codice meno strettamente correlato possibile utilizzando pattern come Inversion of control attuato tramite il pattern **dependency injection**.

Dependency Injection

Dependency injection è un design pattern dell'ingegneria del Software che trasferisce il controllo di oggetti o parti di un programma a un contenitore o framework. Genericamente è utilizzato nel contesto della programmazione orientata agli oggetti. I principali vantaggi di questo approccio sono:

- disaccoppiare l'esecuzione di un task dalla sua implementazione,
- rendere più semplice il passaggio tra diverse implementazioni,
- maggiore modularità del software,

- maggior semplicità nell'effettuare test sul software

Questo design pattern viene largamente utilizzato da Spring ed è implementato all'interno del **Core Container**. Di seguito viene presentato lo schema ad alto livello della struttura in moduli del Framework Spring

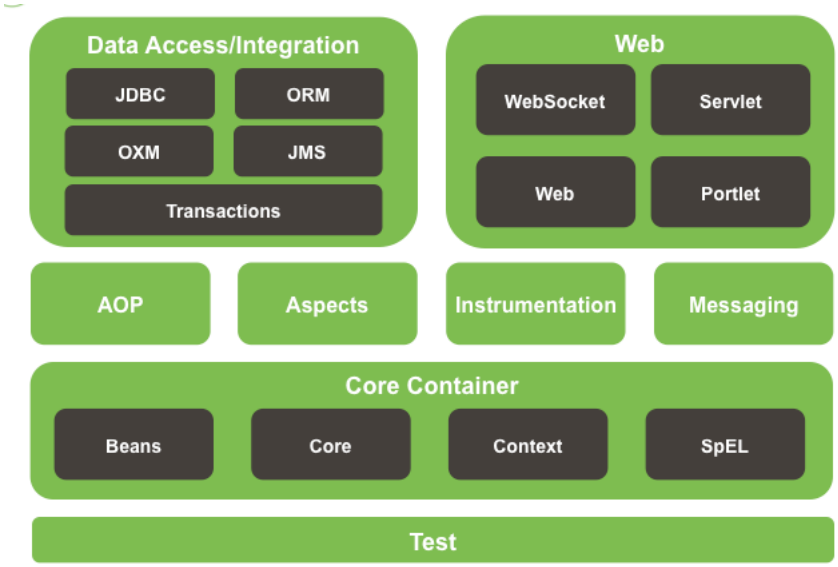

Figura 2.1: Schema ad alto livello della struttura di Spring, [2]

2.3 Spring Core Container

Lo Spring Core Container è il cuore del Framework Spring(figura 2.1). Il container crea gli oggetti, li collega, li configura e gestisce il loro ciclo di vita completo dalla creazione fino alla distruzione. Utilizzando Dependency Injection gestisce i componenti che costituiscono un'applicazione. Questi oggetti sono chiamati **Beans**. Il container è in grado di effettuare queste operazioni sugli oggetti interpretando le informazioni fornite tramite metadati di configurazione. Questi possono essere rappresentati tramite XML, Java annotations o codice Java. Nello specifico del progetto si sono utilizzate annotazioni e codice Java.

Bean

Gli oggetti che formano lo scheletro dell'applicazione, gestiti dallo Spring Core Container, sono detti beans. La gestione di questi oggetti è diversa da quella classica di una tradizionale applicazione Java. Quest'ultima infatti consiste nell'istanziazione di un oggetto con la keyword *new*, quando un oggetto non è più utilizzato viene gestito dal *garbage collector*. I Bean all'interno

di Spring invece hanno un ciclo di vita più elaborato. In base a come viene definita la configurazione di un Bean è possibile definire determinate informazioni durante l'istanziazione. Di seguito viene presentato uno diagramma delle possibili operazioni per tale processo.

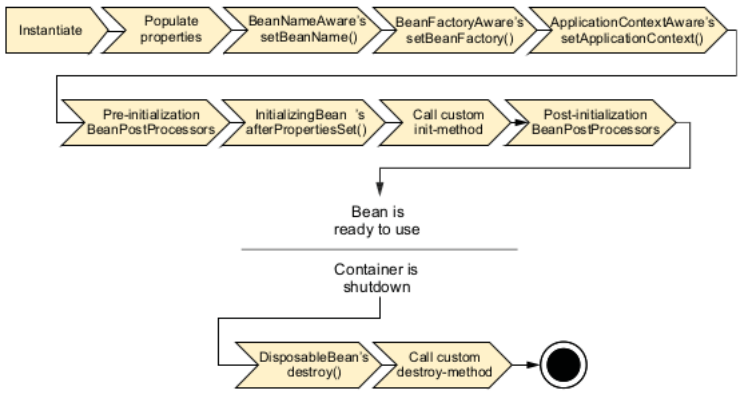

Figura 2.2: Serie di operazioni possibili durante la creazione di un Bean [5]

2.4 Spring Boot

Spring Boot è un Framework open source basato su Spring. Come precedentemente indicato nel paragrafo di introduzione a Spring, il framework è composto da vari moduli che implementano diverse funzionalità, ad esempio per la gestione dei dati, per la sicurezza e, in questo caso, per la configurazione dell'applicazione, fornendo un modo più user friendly per la creazione di applicazioni Spring. Si occupa di eliminare gran parte della configurazione necessaria nella versione classica: infatti anche se Spring si presenta leggero in termini di codice è pesante da un punto di vista dei file di configurazione. Spring Boot consente di concentrare la maggior parte del tempo nello sviluppo della logica dell'applicazione più che sulla parte di configurazione. Inoltre fornisce un modo semplice per la gestione delle dipendenze le quali vengono dichiarate all'interno di un file *pom.xml* che tramite *Maven* vengono importate come file JAR.

2.4.1 Caratteristiche di Spring Boot

- **Configurazione automatica**: in base alle dipendenze elencate all'interno del file *pom.xml* Spring Boot configura automaticamente l'applicazione.
- **Starter di dipendenze**: attraverso il file *pom.xml* si dichiarano le dipendenze di cui si ha bisogno, Spring Boot ha la responsabilità di gestirle e renderle disponibili nell'applicazione.

Configurazione Automatica

In una qualsiasi applicazione Spring si trova sia codice Java che codice di configurazione XML che forniscono un certo tipo di funzionalità e dettagli all'applicazione. Per esempio per accedere ad un database in un'applicazione Spring si deve definire il Bean *JdbcTemplate* nel file di *ApplicationContext*. Questo Bean crea un'istanza JdbcTemplate che necessita di un altro Bean, *DataSource*. Quindi vi è la necessità di creare anche questo Bean. Per esempio si supponga di voler utilizzare un DBMS H2: una volta creato quest'ultimo Bean l'applicazione è in grado di comunicare con il database. Spring Boot elimina la necessità di definire questi due Bean in quanto provvede automaticamente a configurare il connector per il database scelto: è sufficiente indicarlo all'interno del file XML.

2.4.2 Vantaggi e Svantaggi

Vantaggi

- Riduce i tempi di sviluppo e incrementa la produttività del team di sviluppo
- Aiuta nella configurazione dei vari componenti
- Rende più semplice la creazione e il testing di applicazioni Java-based fornendo un setup di default per unit e integration testing
- Pemette di evitare la scrittura di codice ripetitivo e di configurazioni XML
- Fornisce un Server HTTP come *Tomcat* per testare le applicazioni

Svantaggi

- Uno degli svantaggi maggiori è la mancanza di controllo sul sistema
- Le versioni delle dipendenze devono essere costantemente monitorate, possibili incompatibilità portano a difficoltà nell'individuare il problema.

2.5 Spring Data

Spring Data è un modulo dell'ecosistema Spring volto alla gestione dei dati, si occupa della comunicazione con il database di riferimento (o i database nel caso di microservizi) fornendo interfacce di semplice comprensione e di alto livello per la generazione di tabelle(DBMS relazionale), di transazioni e di interrogazioni al database. Lo scopo è di unificare e facilitare

l'accesso a diversi tipi di archivi di persistenza, siano essi sistemi di database relazionali o non relazionali (NoSql).

Per ognuna delle tipologie di archiviazione, vengono utilizzati dei componenti chiamati Repository (Data Access Object) i quali hanno il compito di rendere l'applicazione il più indipendente possibile tra le parti di codice definito. Queste interfacce in genere forniscono operazioni CRUD (Create-Read-Update-Delete) sugli oggetti gestiti dall'applicazione. Spring Data fornisce interfacce generiche per questo tipo di operazioni così come permette delle implementazioni definite dallo sviluppatore per determinate operazioni non previste di default da Spring Data.

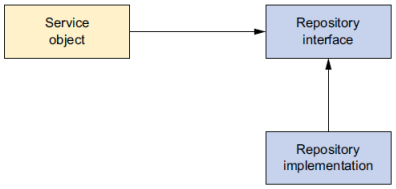

Figura 2.3: I serivice Objects non gestiscono direttamente l'accesso ai dati. Quest'operazione viene delegata alle repository. Essendo interfacce mantengono l'applicazione disaccoppiata tra le parti. [5]

2.5.1 ORM: Object Relational Mapping

Come accennato precedentemente, per la gestione dei dati nei database relazionali si utilizza il sotto-modulo Spring Data JPA il quale ha introdotto uno standard per mappare classi Java in tabelle del database di riferimento. Per lo sviluppatore è sufficiente usare determinate annotazioni (JPA) all'interno di una POJO per ottenere classi entity definendo tutte le caratteristiche di una tabella come gli attributi, le chiavi oppure mappare associazioni tra gli oggetti (entities). Nel modello a tabella, questi attributi rappresentano le chiavi esterne. Queste associazioni sono mappate tramite attributi del tipo dell'entity associata o un insieme di entities associate. Di seguito vengono presentate alcune delle più utilizzate annotazioni per la mappatura di classi in tabelle.

- **@Entity**: definisce una classe Java persistente, ovvero indica che sarà una tabella. Ogni volta che l'applicazione viene avviata la classe con quest'annotazione viene sincronizzata con la rispettiva tabella nel database.
- **@GeneratedValue**: indica che la variabile associata (in genere chiave primaria) è stata inizializzata nel database. Viene ad esempio utilizzata per generare id progressivi per identificare univocamente una entry nella rispettiva tabella.

- **@Id**: assegna alla variabile a cui è assegnata il ruolo di chiave primaria
- **@Table**: viene utilizzata per specificare il nome della tabella nel database, di solito utilizzata quando si vuol specificare il nome della tabella diverso dal nome della classe
- **@Column**: specifica il nome di una colonna, oppure ne identifica vincoli sui valori assunti. Ad esempio può essere utilizzata per impedire l'inserimento di valori null.

Per le associazioni tra classi si utilizzano le seguenti annotazioni

- @ManyToMany
- @ManyToOne
- @OneToMany
- @OneToOne

Ad esempio nel caso in cui sia necessaria una relazione tra tabelle del tipo molti a molti, si utilizza l'annotation @ManyToMany ovvero entrambe le classi possono riferirsi a multiple istanze dell'altra classe. Un caso d'uso pratico è dato dalle tabelle Studente e Corso, correlate dall'associazione Iscrizione. Infatti uno studente può iscriversi a più di un corso e un corso ha più studenti iscritti. In JPA tramite l'annotation @ManyToMany si possono mantenere i riferimenti agli studenti e ai corsi nelle rispettive classi (entities) Studente e Corso.

```
@Entity
class Studente {

    @Id
    Long id;

    @ManyToMany
    Set<Corso> corsiIscritti;

    // altre proprietà
    // costruttori, getters e setters
}

@Entity
class Corso {

    @Id
    Long id;

    @ManyToMany
    Set<Studente> studentiIscritti;

    // altre proprietà
    // costruttori, getters e setters
}
```

Figura 2.4: Esempio di utilizzo dell'annotazione @ManyToMany

2.6 Spring Cloud

Spring Cloud è un Framework per l'implementazione di applicazioni in cloud. Questo modulo semplifica lo sviluppo fornendo soluzioni a molti dei problemi comuni che si affrontano

durante la migrazione ad un sistema distribuito, ad esempio durante la progettazione di un sistema a microservizi. Un'architettura di questo tipo permette agli sviluppatori di concentrarsi su un problema alla volta in quanto la natura frammentata dei microservizi fonda le basi sulla separazione dei ruoli che una determinata parte di codice svolge all'interno dell'applicazione. Inoltre risulta possibile effettuare modifiche su un microservizio senza introdurre regressioni nella totalità dell'applicazione, e nel momento in cui sorgono errori o bug questi restano circoscritti all'ambito di microservizio associato. Tuttavia sono necessari degli accorgimenti che ad esempio in un'applicazione monolitica non sorgono, tra cui:

- Esternalizzazione dei file di configurazione
- Si deve utilizzare un service discovery per il tracciamento dei vari microservizi
- Implementazione di un'API Gateway per la gestione della sicurezza dei microservizi

2.6.1 Gestione dei file di configurazione in Spring Cloud

Gestire le configurazioni di un'applicazione è un processo critico per i microservizi in cloud in quanto le istanze dei vari microservizi necessitano di essere lanciate velocemente con il minimo intervento dello sviluppatore. La gestione delle configurazioni si basa sui seguenti principi.

- Segregate ("segregazione"): si vogliono separare completamente le informazioni di configurazione dall'effettivo *deployment* del servizio. A tal fine i dati di configurazione sono quindi forniti al microservizio al momento di avvio dello stesso come variabili d'ambiente prese da una Repository centralizzata.
- Abstract ("astrazione"): l'accesso ai dati di configurazione dovrebbe essere fatto attraverso un'interfaccia. Quindi invece di scrivere codice che accede direttamente alla repository si effettua tale operazione tramite un servizio REST-based in formato JSON che recupera i dati.
- Centralize ("centralizzazione"): dal momento che un'architettura a microservizi può arrivare ad avere centinaia di istanze è importante che i file di configurazione siano il più centralizzati possibile, ovvero che vengano utilizzate il minor numero di repository possibile.
- Harden ("solidità"): dal momento che i file di configurazione sono separati dal deploy del microservizio è importante che qualsiasi soluzione si utilizzi possa essere implementta per essere altamente accessibile e ridondante.

E' fondamentale che questo processo sia effettuato correttamente in quanto errori sulla definizione della gestione dei dati di configurazione spesso provocano errori di difficile identificazione in quanto non sollevano specifici errori: semplicemente l'applicazione potrebbe non funzionare come previsto costringendo gli sviluppatori ad impiegare risorse ed energie inutilmente per la risoluzione dei bug.

Architettura per la gestione dei dati di configurazione

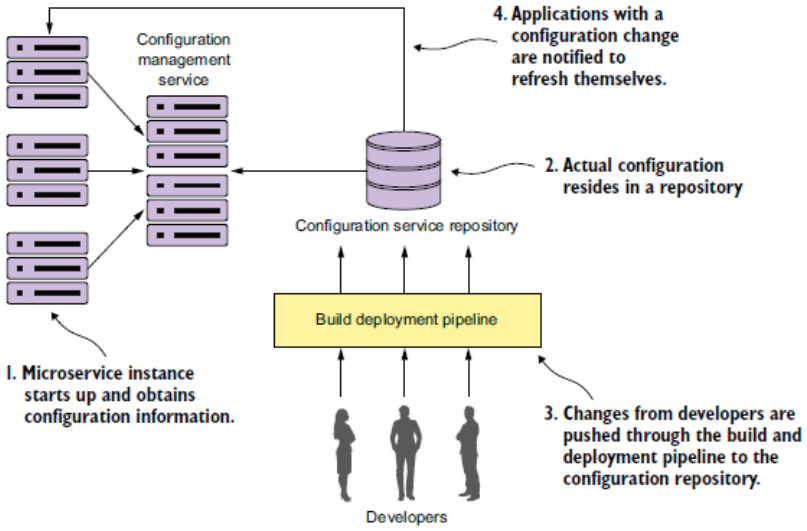

Figura 2.5: Architettura per la gestione delle configurazioni di microservizi [1]

Nella figura 2.5 si possono osservare i vari step per il recupero dei dati di configurazione.

- 1. Quando si avvia l'istanza di un microservizio viene effettuata una chiamata all'endpoint (quindi si effettua una REST-based request) che recupera i dati di configurazione.
- 2. Le informazioni, come enunciato precedentemente, sono centralizzate in una Repository esterna
- 3. Eventuali modifiche ai file di configurazione sono effettuate tramite la pipeline di build e deployment
- 4. Quando vengono effettuate modifiche queste devono essere notificate ai microservizi oggetto della variazione, i quali dovranno essere aggiornati con una copia dei nuovi dati di configurazione.

Una delle implementazioni più facilmente attuabili per la gestione delle configurazioni è tramite una repository *Git*. Usando questa tecnologia si possono sfruttare tutti i benefici derivanti da essa, tra cui la ben collaudata gestione dei file, il versioning e le ramificazioni.

2.6.2 Service Discovery: localizzazione dei microservizi

In ogni architettura distribuita si ha la necessità di identificare univocamente l'indirizzo fisico al quale si trovano le varie macchine. Questo concetto di localizzazione prende il nome di *Service Discovery*. Per una piattaforma a microservizi tale concetto risulta fondamentale dal momento

che offre la possibilità di scalare orizzontalmente l'applicazione in maniera rapida oppure aumentare il numero di istanze di un microservizio. Tale necessità deriva dal livello di affidabilità che si vuol fornire all'applicativo. Per una piattaforma con un elevato flusso di request è preferibile che vi siano numerose istanze. Inoltre scalare orizzontalemente permette di dislocare determinate funzioni dal codice presente, cosa impossibile in un'architettura monolitica.

L'architettura di un service discovery

A prescindere dal tipo di implementazione che viene effettuata per costituire un service discovery resta di base la struttura che definisce i seguenti concetti

- Registrazione dei servizi
- Condivisione delle informazioni
- Monitoraggio della salute dei servizi

2.6.3 Api Gateway: uniformare l'accesso all'applicazione

In un sistema distribuito come quello a microserizi, si identifica la necessità di fornire funzionalità quali: sicurezza, possibilità di accesso da parte di utenti con ruoli diversi, tracciamento degli utenti e una serie di altre peculiarità che permettono la corretta e sicura navigazione all'interno dell'applicativo. Per implementare tali comportamenti, senza la necessità di descriverne di specifici per ogni microservizio, si utilizza un *API Gateway*. Quest'ultimo realizza di fatto un filtro per le varie request che arrivano dall'esterno indirizzandole ai rispettivi microservizi. Un API Gateway è pertanto un intermediario tra il servizio che viene invocato e il Client che lo invoca: il client utilizza un URL con riferimento al gateway, il quale poi reinderizza correttamente la chiamata al servizio invocato. Nel farlo, se previsto, possono esser effettuati una serie di controlli sull'utenza che effettua la richiesta di servizio. Per esempio, se implementato, è possibile che il gateway blocchi richieste per particolari livelli di autorizzazione assegnati ad un'utenza, oppure permetta solo determinate azioni. Di seguito viene proposto un possibile caso d'uso in presenza di API Gateway.

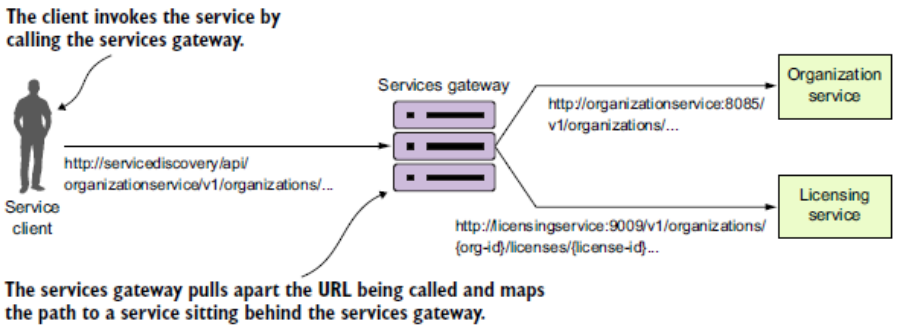

Figura 2.6: Architettura per la gestione delle configurazioni di microservizi [1]

2.7 Spring Security

Spring Security è il modulo del framework Spring che fornisce a suo volta un Framework potente, efficace e altamente configurabile per l'autenticazione, l'autorizzazione, il controllo degli accessi all'applicazione oltre a fornire configurazioni volte alla protezione dai più comuni attacchi informatici. In sostanza questo modulo si pone come obiettivo principale la securizzazione delle applicazioni sviluppate tramite Spring.
Come visto per Spring Data, anche in Spring Security si utilizzano le annotations per definire i vari oggetti per la gestione degli endpoint, i ruoli, la definizione di utenti e tutte le operazioni correlate a questo modulo. Come per ogni framework, uno degli obiettivi principali è permettere allo sviluppatore di scrivere meno codice possibile ottenendo il massimo possibile, permettendo quindi di concentrare tempo e risorse nella progettazione dell'applicazione più che nel dover rischivere parti di codice ridondante. Tuttavia Spring Security garantisce la possibilità di scrivere tutte le logiche per una determinata funzionalità in maniera del tutto libera senza vincoli dati dal codice predefinito.
Le principali feature di Spring Security sono:

- Codificazione delle Password (password encoding)
- In Memory Autentication
- Session Management
- API Security con JWT e OAuth2
- Ruoli Utente ed autorizzazione basata su quest'ultimi

2.7.1 Password Encoding

I benefici riguardanti l'utilizzo di Spring Security non sono limitati alla semplice autenticazione e autorizzazione degli utenti: questo modulo di Spring fornisce anche aiuto nell'applicare le "best practice" per memorizzare gli utenti, ovvero garantire un sistema di registrazione. Quando si fornisce tale caratteristica all'applicazione spesso si incorre nell'errore di salvare password di utenti in chiaro senza alcun tipo di codifica. Questo non solo permette ad eventuali malintenzionati con accesso al Server di raccogliere informazioni sensibili ma espone anche gli utenti a rischi non indifferenti dal momento che molti utenti usano la stessa password per molti account su differenti piattaforme. Per aggirare la criticità Spring Security permette agli sviluppatori di assegnare un *password encoder* all'oggetto *UserDetails*. Di default usa **BCrypt** per criptare le password; quest'ultimo è un algoritmo ben conosciuto. È inoltre possibile definire il numero di hashing rounds per una determinata codifica. Un possibile utilizzo per la codifica tramite BCryptPasswordEncoder è il seguente:

- BCryptPasswordEncoder(BCryptPasswordEncoder.BCryptVersion version, int strength, java.security.SecureRandom random)

Una delle implementazioni di default che si definiscono nel file di configurazione per Spring Security è la seguente:

```
@Bean
public BCryptPasswordEncoder bCryptPasswordEncoder() {
    return new BCryptPasswordEncoder();
}

@Override
public void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(userDetailsService).passwordEncoder(bCryptPasswordEncoder);
}
```

Figura 2.7: Possibile implementazione BCryptPasswordEncoder in file di configurazione Spring Security

2.7.2 In Memory Authentication

L'autenticazione in memoria consiste nell'utilizzare un database che sostanzialmente esiste solo in RAM ovvero quando l'applicativo è attivo. Un esempio di tale database è H2: si possono salvare utenti ed effettuare autenticazioni senza effettivamente salvarli su un database persistente come MySql, Postgres o Sql Developer. Un database come H2 può essere utile quando si sta

sviluppando e si necessita di effettuare dei test in modo da non andare a modificare il database che viene utilizzato per salvare dati reali. Spring Security fornisce la possibilità di autenticare utenti che risiedono in database non persistenti. Un esempio di tale configurazione è il seguente:

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {


    // Some other configuration code here...


    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
      auth.inMemoryAuthentication().withUser("temporary").password("temp123").roles("ADMIN");
    }
}
```

Figura 2.8: Possibile implementazione di un utente in memoria centrale

2.7.3 Session Management

Nelle applicazioni in cui il Frontend e il Backend non sono nettamente separati, ad esempio nelle applicazioni Spring MVC, ogni qualvolta un utente effettua l'accesso viene creata e memorizzata una sessione con le informazioni riguardanti quest'ultimo. Le varie operazioni che si possono fare su questa sessione prendono il nome di Session Management ovvero gestione della sessione. Spring Security fornisce un meccanismo di controllo di questo particolare oggetto, crea la sessione quando l'utente accede e lo ricicla quando effettua la disconnessione, di fatto gestendo il ciclo di vita della sessione. Inoltre Spring Security fornisce misure aggiuntive per l'utilizzo sicuro di una sessione:

- La sessione è configurabile per permettere la disattivazione della riscrittura degli URL, che serve per arginare la possibilità di attacco di tipo "Session tracking"
- Spring Security permette di utilizzare solo sessioni HTTP e flag di sicurezza sui Cookie di sessione per proteggere quest'ultimi

2.8 Protocolli di sicurezza

I protolli di sicurezza, nel Web di oggi, incontrano la necessità di proteggere i dati degli utenti che con il tempo sono diventati sempre più sensibili. Si pensi a tutte le informazioni che si

forniscono per usufruire dei vari servizi offerti telematicamente, a partire da conti bancari a informazioni riguardanti la salute. Il mutamento del nostro stile di vita a favore di una sempre più forte digitalizzazione necessita quindi di una sempre più attenta securizzazione delle nostre identità e dei nostri dati di carattere sensibile. A tal fine sono stati introdotti protocolli di sicurezza in grado di soddisfare questa necessità di privacy. Nell'applicazione sviluppata si sono usati:

- JSON Web Token (JWT)
- Open Authentication 2.0 (OAuth 2.0)

2.8.1 JSON Web Token (JWT)

Un JWT è un contenitore che porta differenti tipi di asserzioni o *claims* dal Client all'applicazione in maniera sicura. Un'asserzione è un dato che viene definito/dichiarato da parte di una entity all'interno della comunicazione. Le varie asserzioni rappresentano le informazioni utili al Login, ad esempio lo username, mail, nome, cognome e role (necessario per fornire determinate funzionalità riservate a particolari tipi di utenze). Tali informazioni vengono firmate da parte del Server secondo la specifica JSON Web Signature (JWS): in questo modo il server è in grado di riconoscere se sono state generate da se stesso o meno.

Struttura del token

Dal nome "JSON Web Token" si possono evincere le principali informazioni che definiscono il modo in cui è strutturato.

- JSON: usa il formato JSON per formattare i dati trasmessi
- Web: è definito per essere utilizzato all'interno di web requests
- Token: è l'implementazione di un token ovvero una particolare stringa che successivamente il server potrà identificare univocamente

Un JWT è composto di 3 parti separate da un punto e codificate in *Base64*. Un esempio è il seguente:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4

Figura 2.9: Esempio di codifica in base64 di un JWT [13]

Nella figura 2.9 si identificano 3 parti distinte:

- **Header**: nell'header si definiscono i metadati rigurdanti il tipo di algoritmo utilizzato per la signature e il tipo di Token utilizzato
- **Body**: nel body si definiscono i dati effettivamente utili per l'autenticazione dell'utente (Payload)
- **Signature**: la firma del token, rappresenta il risultato della criptazione fornita dall'algoritmo indicato nell'header.

Importanza della signature

Il punto fondamentale dell'efficienza del Token consiste nella presenza di una signature generata tramite algoritmo crittografico da parte del server. Infatti solo quest'ultimo conoscendo la chiave di cifratura è in grado di determinare se un deterinato JWT è congruo con quanto generato o meno. Infatti il JWT non essendo totalmente criptato può essere intercettato e modificato da parte di un malintenzionato il quale se effettua modifiche al Payload rende il token invalido.

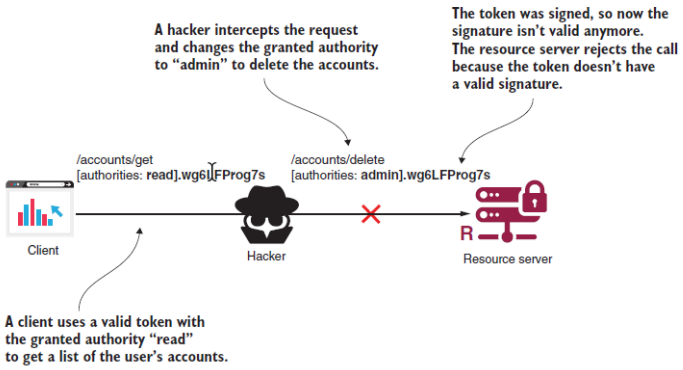

Figura 2.10: Nel caso di intercettazione e modifica del token questo risulterà invalido [3]

Scadenza del token

Durante l'autenticazione, quando l'utente inserisce le proprie credenziali ed effettua il Login, viene ritornato un JWT. Dal momento che i Token sono credenziali, devono essere gestiti in modo da ridurre i rischi correlati ad eventuali furti digitali. A tal fine si è definita la necessità di rendere i token validi per un determinato arco temporale; tale informazione viene inserita all'interno del token il quale avrà un id correlato alla data di generazione e un id correlato alla data di fine validità. La differenza tra i due determina la durata del token.

2.8.2 Open Authentication 2.0

OAuth 2.0 è un protocollo di sicurezza volto all'autenticazione sicura da parte degli utenti tramite piattaforme esterne all'applicazione in questione. Da notare che OAuth non è una particolare implementazione o una libreria preconfezionata: si tratta di un modus operandi ed è quindi possibile applicarlo a diversi Framework e linguaggi di programmazione. Per comprendere la necessità di un protocollo che permetta l'invio sicuro di credenziali sensibili sul Web si consideri il seguente esempio. Si supponga di utilizzare un'autenticazione tramite HTTP basic authentication: in tal caso le credenziali devono essere inviate tramite un Header ogni qualvolta si effettua una request, inoltre le credenziali sarebbero memorizzate da un sistema terzo (il Browser). Questi rappresentano dei punti di debolezza da un punto di vista della sicurezza. Al fine quindi di evitare tali criticità è opportuno isolare la responsabilità della gestione delle credenziali in un solo componente, ovvero *l'authorization server*.

Le componenti dell'architettura OAuth 2

Le componenti del protocollo OAuth 2.0 sono le seguenti.

- **Resource server**: l'applicazione che contiene le informazioni/risorse possedute dall'utente.
- **L'utente**: il proprietario o l'utilizzatore (con relativo permesso) delle risorse rese disponibili dal server. In genere un utente viene identificato univocamente da username e password.
- **Il client**: l'applicazione che effettua l'accesso alle risorse. Il Client usa un ID e un client secret per identificarsi.
- **L'authorization server**: l'applicazione che autorizza il client ad accedere alle risorse rese disponibili dal server delle risorse. Quando l'authorization server riconosce come

valida la richiesta di accesso da parte del client alle risorse genera un token. Il client poi utilizza questo token per provare al resource server di essere in possesso del permesso di accesso.

Possibili implementazioni del protocollo OAuth 2

Per implementare il protocollo esistono diversi tipi di percorsi di autenticazione al fine di ottenere il Token per effettuare l'accesso alle risorse. Queste implementazioni prendono il nome di *grants*. Alcune delle più comuni sono:

- authorization code,
- password,
- refresh token,
- client credentials.

Nell'implementazione del progetto si è previsto l'utilizzo del *grant type password* descritto nel successivo paragrafo

Grant type password

Nelle applicazioni che utilizzano tale grant type si assume che le credenziali di accesso vengano inserite da parte del Client per comunicazione dell'utente. Ovvero nel momento in cui si effettua l'accesso si passano nella request degli Header con le credenziali: se si utilizza una connessione protetta oppure si cifrano le credenziali tramite JWE non ci sono problemi da un punto di vista della sicurezza. Il processo autorizzativo è cosi definito:

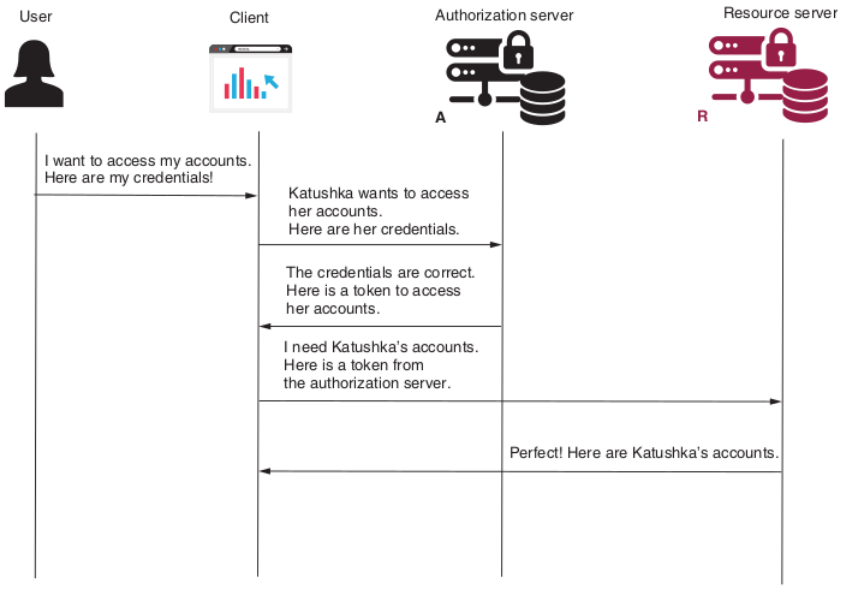

Figura 2.11: Flow di funzionamento per il grat di tipo Password [3]

Nello schema 2.11 si identificano le entità definite in 2.8.2 le loro azioni sono le seguenti.

- **User** comunica le credenziali per effettuare l'accesso
- **Client** passa le credenziali all' **authorization server** il quale, nel caso in cui le credenziali siano valide, ritorna un Token (ad esempio un JWT) per accedere all'account, ovvero alle risorse
- **Resource server** riceve la richiesta da parte del Client attraverso il token generato al punto precedente, nel caso in cui il token sia valido soddisfa la richiesta effettuata.

Capitolo 3: Strumenti software utilizzati

Per il corretto svolgimento dell'esperienza di tirocinio sono stati utilizzati diversi strumenti Software e tecnologie: in questo capitolo vengono riportati i più significativi.

Spring Tool Suite: ambiente di sviluppo integrato, derivato da Eclipse, che mette a disposizione strumenti improntati allo sviluppo di applicazioni basate su Spring. Fornisce supporto al linguaggio Java, al Framework Spring ed all'eventuale ambiente di sviluppo. Per maggiori informazioni si rimanda al sito ufficiale [7]

GitHub: servizio cloud basato su GIT, che aiuta gli sviluppatori ad archiviare e gestire il codice, tracciandolo e versionandolo tramite branches (rami). Utilizzato per permettere la visione del codice e dei progressi da parte del responsabile di tirocinio.
Direttamente consultabile al sito ufficiale [8]

Git: è un sistema di controllo versioni distribuito, il che significa che l'intero codice del progetto e la cronologia sono disponibili sul computer di ogni sviluppatore, il che permette di creare facilmente ramificazioni e fusioni. Per maggiori informazioni si rimanda al sito ufficiale [9]

Postman: come descritto alla sezione 4.7.1 è un Software utilizzato per effettuare delle chiamate HTTP ai vari endpoint del backend di un'applicazione per testarne le funzionalità. La peculiarità che rende postman un ottimo software di test per API e Backend di applicazioni è che fornisce numerosi dati in merito alle request e alle response, inoltre fornisce un alto grado di libertà nella composizione delle request, potendo inserire headers custom, definire variabili per determinati ambienti e passare qualsiasi tipo di body nel formato corretto. Per maggiori dettagli si visiti il seguente sito ufficiale [10]

Capitolo 4: Descrizione dell'applicazione

In questo capitolo vengono descritti i requisiti che l'applicazione deve soddifare, la struttura dell'applicazione OnlineVoting, le implementazioni degli aspetti di sicurezza descritti nel capitolo "Aspetti teorici" e le funzionalità fornite dal Backend del sistema di voto elettronico.

4.1 Requisiti

L'applicazione OnlineVoting si pone come obbiettivo principale la possibilità di usufruire in maniera sicura di un sistema per la votazione di partiti politici tramite una piattaforma digitale. Di seguito verranno elencati i requisiti divisi in requisiti funzionali e progettuali.

Requisiti funzionali

I requisiti funzionali descrivono cosa l'applicazione dev'essere in grado di offrire all'utente che ne fa utilizzo. Durante il colloquio per la definizione degli stessi si è scelto di identificare inoltre due diversi sottogruppi di requisiti data l'incertezza dei tempi necessari per l'implementazione. Quindi si sono identificati requisiti obbligatori e desiderabili secondo la seguente nomenclatura.

Requisito	Descrizione
REQ001	Requisito obbligatorio
REQ002	Requisito desiderabile

Di seguito l'elenco dei requisiti con relativo sottogruppo di appartenenza

- (REQ001) Autenticazione di un utente tramite le proprie credenziali (Username e password).
- (REQ001) Determinazione del livello di privilegi di autorizzazione di un utente definendo ruoli associati ad essi.
- (REQ002) Modifica dello username di un utente controllandone l'univocità
- (REQ002) Modifica dell'email di un utente controllandone l'univocità
- (REQ002) Eliminazione di un'utenza tramite id dell'utente
- (REQ001) Ottenimento dela lista completa degli utenti
- (REQ001) Inserimento di un voto garantendo l'anonimato del votante
- (REQ001) Inserimento di una votazione per un determinato partito con il relativo candidato.
- (REQ002) Inserimento di un voto come in una votazione reale, quindi sono previste le seguenti possibilità di scelta:
  - Votazione di un partito e un candidato, dove quest'ultimo appartiene al partito votato
  - Votazione di un partito e un candidato, dove quest'ultimo non appartiene al partito votato
  - Votazione di un partito ma non di un candidato
  - Votazione di un candidato ma non di un partito
  - Votazione nulla
- (REQ001) In base all'id del voto ottenimento informazioni su di esso, ovvero l'id del voto, il nome dell'eventuale candidato votato e l'id associato, il nome dell'eventuale partito votato e l'id associato, la data e l'orario di inserimento della votazione.
- (REQ001) Ottenimento della lista delle votazioni effettuate
- (REQ001) Nel caso in cui una votazione venga ritenuta invalida si riserva la possibilità di eliminazione di una votazione
- Ottenimento del vincitore dell'elezione

- (REQ001) Inserimento di un partito con il relativo candidato
- (REQ001) Ottenimento della lista dei partiti e i relativi candidati
- (REQ001) Eliminazione di determinati partiti e i relativi candidati

Requisiti progettuali

L'architettura dell'applicazione deve permettere il salvataggio sicuro delle password a db, ovvero lo sviluppatore in grado di recuperare la lista delle utenze non deve entrare in possesso delle password degli utenti. Inoltre all'interno del codice dev'essere ridotta al massimo la presenza in memoria secondaria di password in chiaro per, ad esempio, la comunicazione con il database o per la definzione della signature nella generazione dei token JWT.
Devono inoltre esser definiti dei ruoli per le utenze in modo che determinate funzionalità siano disponibili solo ad un determinato tipo di utenza. Ad esempio tutte le operazioni che non concernono direttamente l'utente autenticato non devono essere accessibili a quest'ultimo. Un caso d'uso potrebbe esser rappresentato dall'eliminazione di una o più utenze oppure dal recuperare tutte le utenze registrate. La stessa logica di limitazione delle funzionalità va esteso all'accesso delle votazioni registrate o alla registrazione di nuovi candidati o partiti.

4.2 Rappresentazione del progetto ad alto livello

4.2.1 Progettazione dell'applicazione

La progettazione dell'applicazione è iniziata all'inizio dell'esperienza di tirocinio, prima dello studio dei vari aspetti teorici, i quali sono stati poi approfonditi al fine di esser in grado di implementare una tale applicazione. La decisione dei vari aspetti progettuali è stata presa in collaborazione con il responsabile di tirocinio interno all'azienda. L'applicazione quindi è stata progettata da zero e tutte le implementazioni sono state definite dal sottoscritto.

4.2.2 Struttura dell'applicazione

La struttura dell'applicazione è a microservizi, ognuno contente una business logic atta a soddisfare un certo tipo di richieste. Dal momento che il Frontend non è separato come l'architettura a Backend non vi sono parti di quest'ultimo che comunicano solo ed esclusivamente con un determinato microservizio, quindi si suppone che in una tal sistuazione il frontend possa effettuare request direttamente al servizio che necessita. Un tal sistema però porterebbe a dover implementare la sicurezza in tutti i microservizi di fatto scrivendo codice ridondante e logicamente

poco significativo. Inoltre ci sarebbero problemi di latenza, ovvero il tempo che intercorre tra l'invio di una request e la response tenderà ad essere elevato. La soluzione da adottare quindi è di prevedere un API Gateway che funge da filtro per le chiamate che arrivano dal frontend, verificandone l'autenticità e in seguito effettuando internamente all'applicazione la request al microservizio identificato tramite l'endpoint (l'URL). La responsabilità è stata quindi ripartita tra quattro microservizi.

- Un microservizio è responsabile della gestione degli utenti con implementazione dell'API Gateway.
- Un microservizio è responsabile della gestione delle votazioni.
- Un microservizio è responsabile della gestione dei partiti votati.
- Un microservizio è responsabile della gestione dell'applicativo, service discovery.

Di seguito viene presentato uno schema grafico rappresentante l'architettura dell'applicazione. Da notare come il microservizio che implementa l'API Gateway si interpone tra l'esterno e l'interno dell'applicazione. Di fatto è all'interno di quest'ultimo che avviene l'autenticazione e l'autorizzazione degli utenti che richiedono di accedere all'applicativo. Inoltre si noti la presenza del microservizio per la gestione dell'applicativo identificato con "Eureka Discovery Service".

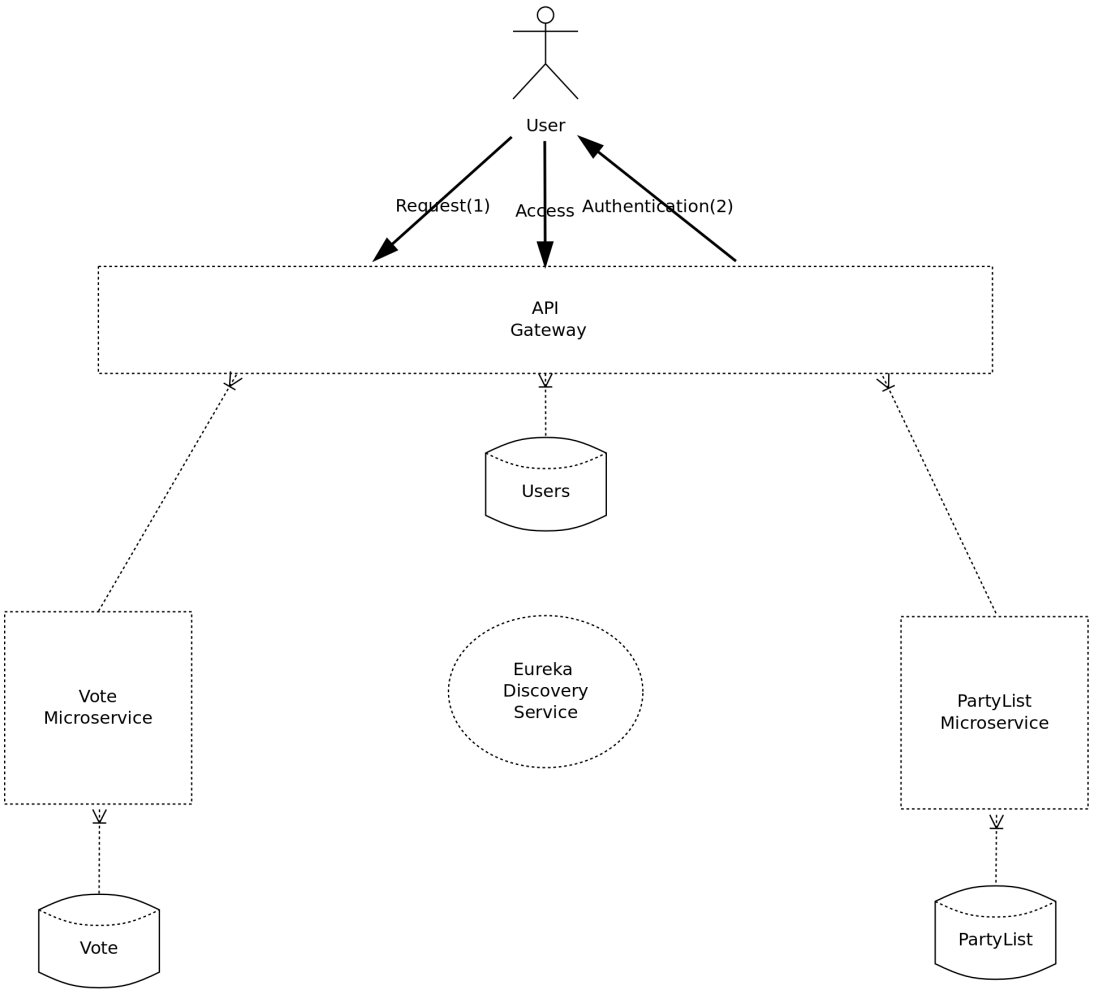

Figura 4.1: Overview ad alto livello dell'applicazione

4.3 Eureka Discovery Service

Nella seguente applicazione l'implementazione scelta per il service discovery (definizione a 2.6.2) è Netflix Eureka appartenente al modulo Spring Cloud. La configurazione consiste nell'importare nel pom le dipendenze necessarie per poter utilizzare le annotation proprie di questo modulo. Si definisce un microservizio quindi per il monitoraggio delle risorse, specificando in un file di configurazione la porta alla quale poter accedere al microservizio (Figura 4.2).

```
1 server:
2   port: 8761
3
4 eureka:
5   client:
6     register-with-eureka: false
7     fetch-registry: false
8
```

Figura 4.2: Configurazione del port per il discovery service

Eureka rende disponibile una pagina a Frontend alla quale è possibile consultare la salute dei vari microservizi e informazioni riguardanti lo stato dell'applicazione

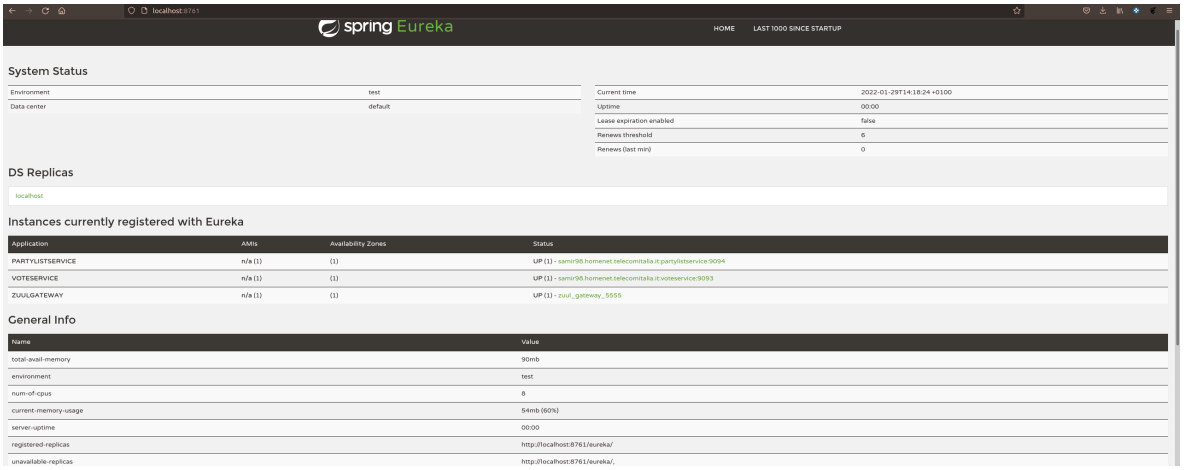

Figura 4.3: Esempio di come appare la pagina di monitoring per Eureka

Registrazione dei microservizi su Eureka

Affinchè tutto il processo di controllo dell'applicazione da parte di Eureka funzioni correttamente i microservizi devono essere registrati. Per tal fine il processo risulta piuttosto semplice, infatti è sufficiente importare la dependency "spring-cloud-starter-eureka" e definire il seguente file di configurazione. Nota: tale processo risulta identico per ogni microservizio cambiandone

```yaml
server:
  port: 9094

spring:
  application:
    name: partylistservice

eureka:
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://localhost:8761/eureka/
  instance:
    hostname: localhost

```

Figura 4.4: Registrazione di un microservizio sul discovery service

i parametri collegati, ovvero nome, port ed endpoint.

4.4 API Gateway

Nell'applicativo si è utilizzato Zuul API Gateway compatibile con le annotazioni Spring Cloud per l'implementazione del gateway. Zuul offre numerose funzionalità tra cui:

- Reindirizzare tutti i percorsi ai relativi microservizi tramite un singolo punto d'accesso, quindi la prima parte dell URL consisterà in un unica coppia dominio-porta. Zuul non

è limitato ad un signolo percorso nell'URL, infatti se ne possono definire multipli, tuttavia per l'implementazione dell'App OnlineVoting non è stato ritenuto necessario date le dimensioni ridotte.

- Si possono definire dei filtri in grado di implementare logiche personalizzate applicabili ad ogni richiesta entrante verso i microservizi. Esempi di logiche definite in tal modo consistono in un insieme di politiche di sicurezza, accesso o tracciamento verso i microservizi. Di fatto quindi un filtro o una catena di filtri (in base a quante logiche sono definite) permetto di applicare regole e politiche caratteristiche di ogni applicazione. In Zuul esistono 3 tipi di filtri:
  - Pre-filtro: è invocato prima dell'effettiva richiesta. Di solito implementa logiche di controllo sul formato del messaggio, oppure sul livello autorizzativo dell'utente che effettua una richiesta.
  - Post-filtro: è invocato dopo la risposta del microservizio. Di solito implementa logiche di registrazione delle risposte inviate.
  - filtro di instradamento della richiesta: è invocato durante la richiesta. Di solito implementa logiche volte al corretto instradamento delle richieste nel caso in cui sia implementato un instradamento dinamico.

I vari percorsi vengono definiti grazie all'utilizzo congiunto del discovery service Eureka. Infatti nel file di configurazione si vanno ad indicare i microservizi registrati. Zuul è un reverse proxy ovvero è un Server intermediario tra il client e i microservizi. Il Client non è a conoscenza del fatto che le sue richieste vengono eseguite per mano del proxy. Di seguito una rappresentazione di come viene gestita una richiesta da parte di un utente.

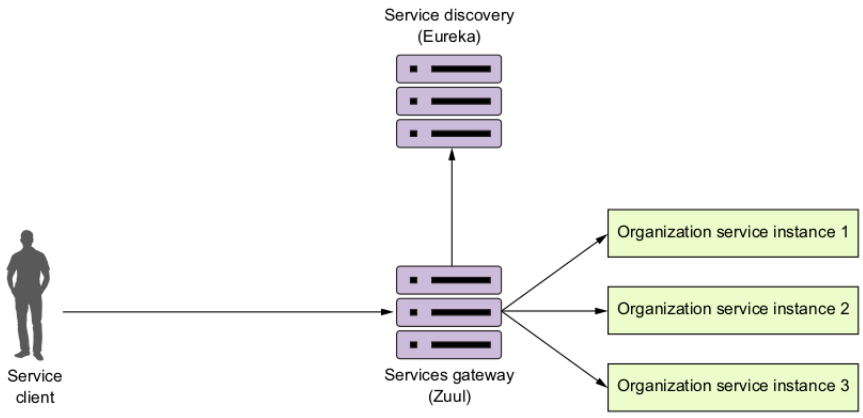

Figura 4.5: Utilizzo di Zuul con Eureka [1]

4.5 Implementazione

4.5.1 Implementazione microservizio per le votazioni

In questa sezione viene presentata l'implementazione del microservizio dedicato per la gestione delle votazioni. Gli altri microservizi hanno struttura e implementazione simili.

Definizione tabella Vote

Per la gestione delle votazioni serve una tabella in grado di memorizzare i dati relativi ad una votazione, a tal fine si definisce una classe decorata con l'annotazione @Entity (2.5.1) le quali variabili saranno gli attributi della tabella nel database. Di seguito l'implementazione della stessa

```java
@Entity
public class Vote {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long voteId;
    private Timestamp timeStamp;
    private String party;
    private String candidate;
    private Long partyId;
    private Long candidateId;

    @Column(name = "userId", unique = true)
    private Long userId;
```

Figura 4.6: Implementazione della classe Vote

Definizione classi di appoggio

Per effettuare operazioni sulle votazioni è necessario gestire anche dati relativi ai partiti/candidati. Ad esempio per il salvataggio di una votazione è necessario recuperare dalla richiesta l'eventuale nome del partito/candidato; pertanto è necessatio salvare questi dati attraverso classi POJO con le stesse caratteristiche delle classi che definiscono le tabelle per questi oggetti. Quindi si è prevista l'implementazione delle seguenti classi

- **Party**: classe definita per memorizzare i dati relativi alla richiesta di inserimento di una votazione
- **ElectionWinner**: classe definita per memorizzare i dati relativi al vincitore dell'elezione, fornisce nome del vincitore e numero di votazioni ricevute

Interazione con il database: classe di servizio e repository

Per recuperare dati dal database nei microservizi vengono definite delle classi di repository e servizio (in inglese *service*) che permettono la comunicazione con il database di riferimento. Ad esempio per recuperare una votazione tramite ID dell'utente oppure per recuperare una votazione tramite l'ID della stessa.

```java
@Repository
public interface VoteRepository extends JpaRepository<Vote, Long> {

    Vote getByVoteId (Long voteId);
    Vote findByUserId(Long userId);
}
```

Figura 4.7: Implementazione della classe VoteRepository

Nella classe VoteService vengono implementati i metodi definiti della classe di Repository, alcuni utilizzano direttamente i metodi forniti da Spring JPA altri invece devono essere definiti dallo sviluppatore come ad esempio "getElectionWinner()".

```java
@Service
public class VoteService {

    @Autowired
    private VoteRepository voteRepository;

    @Autowired
    private PartyListServiceClient partyListServiceClient;

    @Autowired
    private EntityManager em;

    public Vote saveVote(Vote vote) {

        return voteRepository.save(vote);
    }

    public Vote getByVoteId(Long voteId) {
        return voteRepository.getByVoteId(voteId);
    }

    public List<Vote> getListOfAllVotes() {
        return voteRepository.findAll();
    }

    public void deleteVoteById(Long voteId) {
        voteRepository.deleteById(voteId);

    }

    public Vote findByUserId(Long userId) {
        return voteRepository.findByUserId(userId);
    }

    public ElectionWinner getElectionWinner() {
        Query q = em.createNativeQuery("SELECT candidate, COUNT(*) as c FROM vote GROUP BY candidate ORDER BY c DESC LIMIT 1");
        //@SuppressWarnings("unchecked")
        List<Object[]> obj = q.getResultList();
        ElectionWinner ew = new ElectionWinner();
        System.out.println(obj.get(0)[0]);
         ew.setWinner((String)obj.get(0)[0]);
         ew.setNumberOfVotes((BigInteger)obj.get(0)[1]);
        return ew;
    }
}
```

Figura 4.8: Implementazione della classe VoteService

Gestione degli endpoint: classe controller

Per la definizione degli endpoint e delle logiche associate si definisce una classe detta "controller" all'interno della quale si definiscono i criteri per una determinata operatività. Ad esempio l'endpoint per l'inserimento di una votazione oppure per recuperare informazioni su di una votazione Di seguito viene riportato un esempio, l'implementazione dell'endpoint per l'inserimento di una votazione

```java
RestController
@RequestMapping("/vote")
public class VoteController {

    public static final String SUCCESS = "success";
    public static final String UNAUTHORIZED = "Unauthorized";
    public static final String FORBIDDEN = "Forbidden";
    public static final String CONFLICT = "Resource conflict";
    public static final String NOT_FOUND = "Not found";

    @Autowired
    private VoteService voteService;

    @Autowired
    private PartyListServiceClient partyListServiceClient;

    @Autowired
    private UserServiceClient UserServiceClient;

    @PostMapping("/{user_id}")
    public ApiResponse saveVote(Authentication authentication, @RequestBody Vote vote, @PathVariable("user_id") Long userId) {

        List<Party> checkList = partyListServiceClient.getListOfPartiesComplete();

        ApiResponse ap = UserServiceClient.getUserById(authentication, userId);
        Object checkUser = ap.getResult();

        //utilizzo di regex per separare l'oggetto ottenuto in elementi separati tra cui l'id dell'utente
        String[] tokens = checkUser.toString().split("=|,");
        Long checkuserId = Long.parseLong(tokens[1]);
        if (checkuserId.equals(userId)) {
            Timestamp timestamp = new Timestamp(System.currentTimeMillis());

            //Caso base: è possibile inserire una votazione nulla
            if(vote.getCandidate().equals("") && vote.getParty().equals("")) {
                vote.setTimeStamp(timestamp);
                vote.setUserId(userId);
                return new ApiResponse(HttpStatus.OK, SUCCESS, voteService.saveVote(vote));
```

```java
        }
        for(int i = 0; i < checkList.size(); i++) {
            //candidate != null AND party = null
            if(checkList.get(i).getCandidateName().equals(vote.getCandidate()) && vote.getParty().equals("")){
                vote.setCandidateId(checkList.get(i).getCandidateId());
                vote.setTimeStamp(timestamp);
                vote.setUserId(userId);
                return new ApiResponse(HttpStatus.OK, SUCCESS, voteService.saveVote(vote));
            }
            //candidate == null AND party != null
            else if(checkList.get(i).getPartyName().equals(vote.getParty()) && vote.getCandidate().equals("")) {
                vote.setTimeStamp(timestamp);
                vote.setUserId(userId);
                vote.setPartyId(checkList.get(i).getPartyId());
                return new ApiResponse(HttpStatus.OK, SUCCESS, voteService.saveVote(vote));
            }
            //Necessario per inserire una candidato diverso dal partito di appartenenza
            for(int j = 0; j < checkList.size(); j++) {
                if(checkList.get(i).getCandidateName().equals(vote.getCandidate()) && checkList.get(j).getPartyName().equals(vote.getParty())) {
                    vote.setCandidateId(checkList.get(i).getCandidateId());
                    vote.setPartyId(checkList.get(i).getPartyId());
                    vote.setTimeStamp(timestamp);
                    vote.setUserId(userId);
                    return new ApiResponse(HttpStatus.OK, SUCCESS, voteService.saveVote(vote));
                }
            }
        }
    }
    throw new IllegalArgumentException("The inserted party or candidate are not valid, please check the partylist available at http://localhost:5555/api/partylist-service/partylist/list");
}
```

Figura 4.9: Implementazione dell'endpoint per l'inserimento di una votazione

4.5.2 Implementazione del protocollo OAuth 2.0 con JWT

Per quest'applicazione si è prevista l'implementazione di un Client custom per effettuare il Login all'interno dell'applicazione; in alternativa si sarebbe potuto utilizzare un account Google

o Facebook, ma l'implementazione andava oltre i requisiti del progetto. Di seguito vengono presentati snippet di codice in cui si sono definite le logiche per l'implementazione del flow OAuth2. Nell'esempio si sceglie di presentare l'autenticazione tramite *grant type password* (vedi 2.8.2).

La classe che definisce l'authorization Server deve essere all'interno dell'API Gateway in quanto è questo che si interfaccia con l'esterno. Sempre in quest'ultimo viene definito il client per l'implementazione dell'OAuth 2. Si noti che la password non viene salvata in chiaro ma co-

```
@Configuration
@EnableAuthorizationServer
public class AuthorizationServerConfig extends AuthorizationServerConfigurerAdapter {

    private static final String CLIEN_ID = "onlinevoting";
    private static final String CLIENT_SECRET = "$2a$10$m90Gjb9GONhCr9hGe6t2VOUHexgruzraMtb7pUjeff6p/k2gEeMFy";
    private static final String GRANT_TYPE_PASSWORD = "password";
    private static final String REFRESH_TOKEN = "refresh_token";
    private static final String SCOPE_READ = "read";
    private static final String SCOPE_WRITE = "write";
    private static final String TRUST = "trust";
```

Figura 4.10: Definizione del client

dificata tramite algoritmo BCrypt (vedi 2.7.1). In seguito il Client cosi definito deve essere registrato in modo che quando si effettuano delle request venga riconosciuto come valido, la registrazione è stata implementata nel seguente modo

```
    @Override
    public void configure(ClientDetailsServiceConfigurer configurer) throws Exception {

        configurer
                .inMemory()
                .withClient(CLIEN_ID)
                .secret(CLIENT_SECRET)
                .authorizedGrantTypes(GRANT_TYPE_PASSWORD, REFRESH_TOKEN)
                .scopes(SCOPE_READ, SCOPE_WRITE, TRUST);
    }
```

Figura 4.11: Registrazione del client

4.6 Funzionalità

In questa sezione vengono riportati i vari endpoint resi disponibili per l'utilizzo delle logiche implementate dall'applicazione. L'insieme di questi endpoint incontra tutti i requisiti richiesti (sia obbligatori che desiderabili) enunciati alla sezione 4.1.

Endpoints List			
HTTP Method	Endpoint	Microservizio	Breve descrizione
POST	/user/	zuul-gateway	Inserimento di un utente
GET	/user/id	zuul-gateway	Recuperare dati di un utente tramite ID
PUT	/user/modify/ username/id	zuul-gateway	Modificare username
PUT	/user/modify/ email/id	zuul-gateway	Modificare email
DELETE	/user/delete/id	zuul-gateway	Eliminare un utente tramite id
POST	/ partylist/	partylist-service	Inserire un partito+candidato
GET	/partylist/id	partylist-service	Recuperare partito+candidato
GET	/partylist/list	partylist-service	Recuperare lista di tutti i partiti+candidati(senza chiavi)
GET	/partylist/ listComplete	partylist-service	Recuperare lista di tutti i partiti+candidati(con chiavi)
DELETE	/partylist/delete/all	partylist-service	Elimina tutti

DELETE	/partylist/ delete/party_id	partylist-service	Elimina la riga con il party_id indicato
POST	/vote/	vote-service	Inserimento di una votazione
GET	/vote/voteID/id	vote-service	Recuperare voto dato l'utente
DELETE	/vote/id	vote-service	Elimina una votazione in base all'id
GET	/vote/votationList	vote-service	Recuperare tutte le votazioni
GET	/vote/risultati /vincitore	vote-service	Annuncia il vincitore delle elezioni

4.7 Test

Tutti gli endpoint presentati nella precedente sezione sono stati testati con esito positivo, per le casistiche in cui sono necessari dati preesistenti questi sono stati inseriti ad hoc nelle rispettive tabelle del database. Tuttavia, in questa sezione, ho ritenuto significativo riportare il processo di autenticazione all'interno della piattaforma data l'enfasi sulla securizzazione della stessa. Tramite questo test si possono identificare i principali aspetti di sicurezza implementati.

4.7.1 Autenticazione nella piattaforma

Al fine di testare l'applicazione viene usato *Postman*, un Software in grado di effettuare delle richieste basate su protocollo HTTP in modo da simulare il comportamento del Frontend, ovvero l'interfaccia che interagisce con l'utente finale.

- Per prima cosa si devono inserire le credenziali del Client in modo da poter esser riconosciuti da parte del Server di autorizzazione
- Si devono poi passare le credenziali e il tipo di grant type per ottenere il Token da utilizzare nella request ad un servizio.

Ad esempio si supponga il caso in cui un utente voglia ottenere informazioni riguardo a se stesso. Nota: si utilizza un utenza con ruolo ADMIN, quindi non ci saranno limitazioni sui possibili endpoint raggiungibili.

- Credenziali client

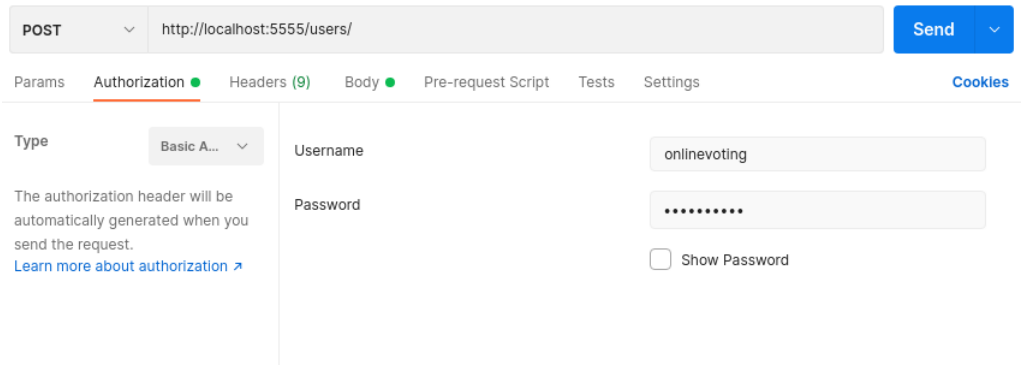

Figura 4.12: Inserimento credenziali client per Basic Authentication

- Credenziali utente e tipo di grant
- Request HTTP POST con endpoint *http://localhost:55555/oauth/token*

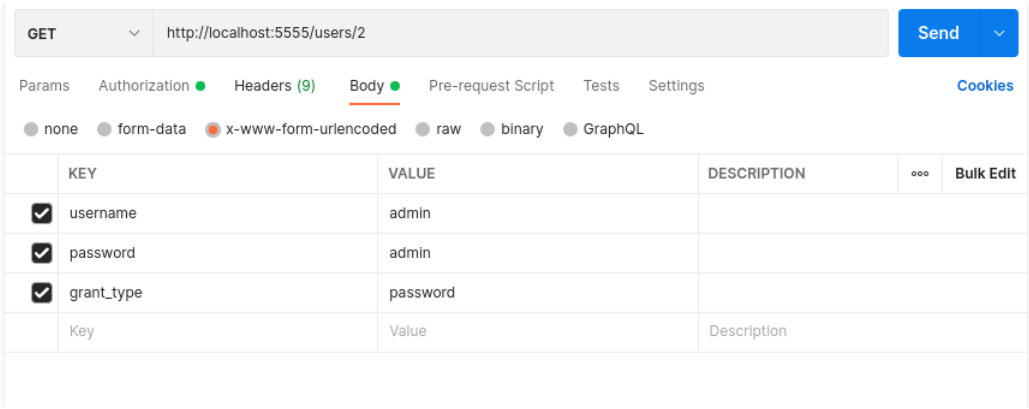

Figura 4.13: Inserimento credenziali utente per ottenere il token

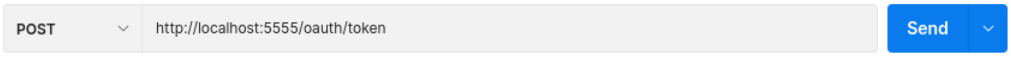

Figura 4.14: Url per ottenimento del token

- Ottenimento del Bearer Token, del token di refresh e dell'expiration time

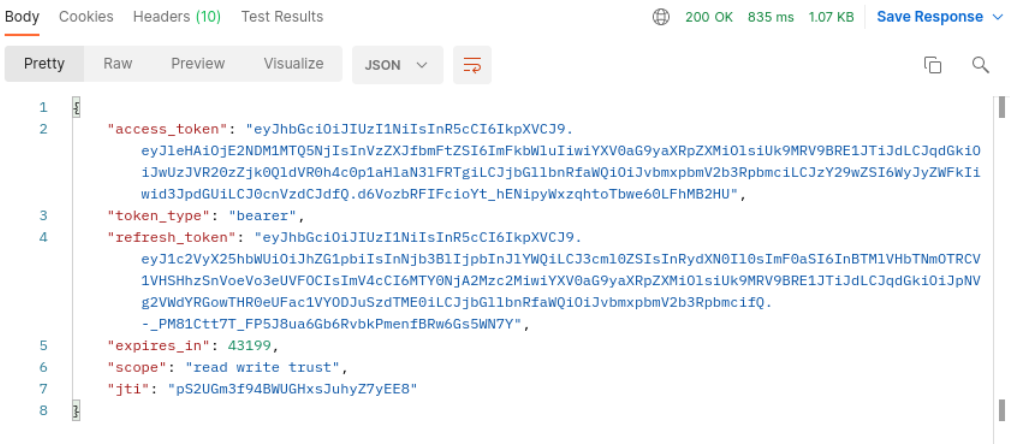

Figura 4.15: Risultato della request all'authentication server

Si noti che nella figura 4.15 sono presenti informazioni riguardo al refresh Token, quest'ultimo tema è stato spiegato nella sezione 2.8.1.

Una volta che si ottiene il token è possibile effettuare una richiesta ad un determinato endpoint, ad esempio:

- Si inserisce il token Bearer come Header *Authorization*

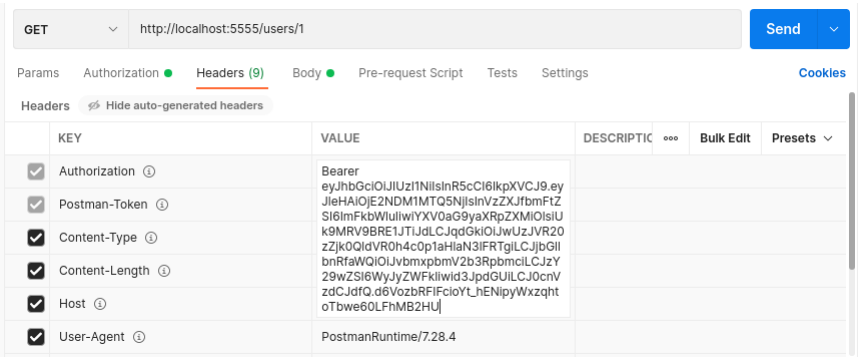

Figura 4.16: Inserimento del Bearer Token nell'Header relativo all'autorizzazione della request

- Si invia la request HTTP GET

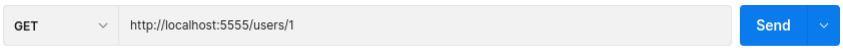

Figura 4.17: Inserimento dell'URL per ottenere le informazioni dell'utente con id = 1

- Si ottiene la risposta dal resource server tramite API Gateway dell'informazione richiesta, per vedere tutti gli endpoint disponibili si faccia riferimento alla sezione 4.6

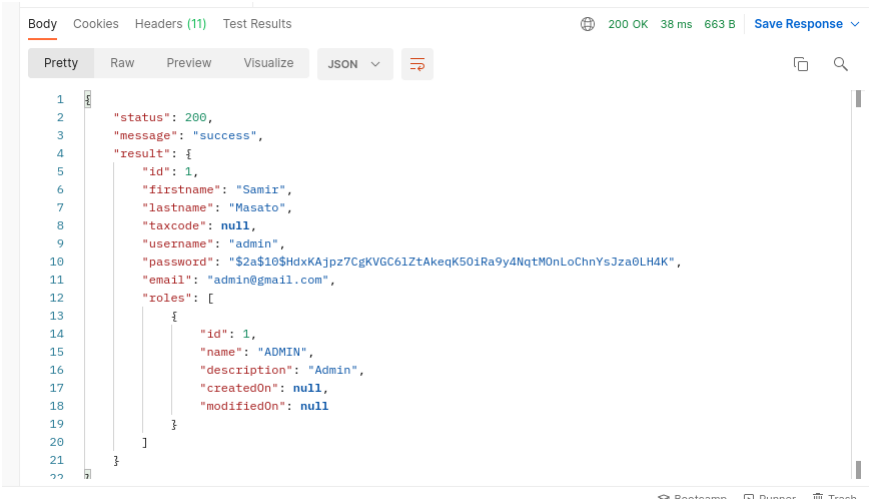

Figura 4.18: Ottenimento della response

Il processo di autenticazione è andato a buon fine, l'utente si è correttamente autenticato e ha ottenuto le informazioni richieste tramite il Bearer token codificato in formato JWT.

Capitolo 5: Conclusioni

Nel seguente capitolo vengono riportate considerazioni conclusive sull'esperienza di tirocinio.

5.1 Conoscenze e compentenze acquisite

Durante questo percorso formativo sono stati introdotti diversi argomenti, dall'ecosistema Spring agli aspetti riguardanti la sicurezza informatica, che purtroppo non vengono trattati durante il corso di studi. Quasi tutte le tecnologie studiate e applicate durante l'esperienza di tirocinio mi sono risultate nuove o in concomitanza con argomenti teorici studiati durante il semestre. Le skill tecniche imparate saranno utili nel mondo lavorativo in quanto la maggior parte degli impieghi nell'ambito informatico richiedono conoscenze di sviluppo web, in questo caso particolare, riguardanti il Backend.

5.2 Considerazioni finali

Il progetto mi ha permesso di conoscere un nuovo framework e di applicare le conoscenze acquisite durante gli insegnamenti della laurea triennale, fornendo di fatto un quadro complessivo, all'interno del quale i vari argomenti si intrecciano a formare un unico progetto vicino ad un'applicazione reale. L'aspetto che maggiormente mi ha soddisfatto è appunto questo, ovvero la progettazione e la successiva implementazione di un'applicazione che effettivamente è in grado di fornire delle funzionalità tangibili e pratiche. Ritengo che il percorso di tirocinio sia fondamentale in un percorso di studi per permettere a chi lo sostiene di percepire ciò che il mondo lavorativo rappresenta, preparando lo studente ad affrontare consapevolmente le sfide che lo attendono. Inoltre, data la situazione pandemica, ho imparato la gestione del lavoro in smartworking e la capacità auto organizzativa per la gestione del carico di studio ulteriore a quello previsto dai corsi del semestre. Complessivamente ritengo molto positiva la mia esperienza in SyncLab, ed a posteriori mi ritengo fortunato per la scelta effettuata.

Glossario

API API è l'abbreviazione di interfaccia di programmazione delle applicazioni (application programming interface), ovvero un insieme di definizioni e protocolli per la creazione e l'integrazione di software applicativi. 14, 17, 26, 29, 32, 36

App Abbreviazione di applicazione. 32

Backend La parte delle applicazioni che non risulta visibile all'utente finale, genericamente svolge il ruolo di elaborazione dei dati attuando le logiche del software. 4–6, 20, 26, 27, 29, 39

Base64 Tipo di codifica dei dati, utilizzato nella codifica dei JSON web token. 21

Bearer Token "Token portatore", un token di sicurezza con la proprietà che qualsiasi parte in possesso di esso può effettuare richieste verso il backend.. 35, 36

Body Rappresenta la parte di dati che vengono trasmessi in una comunicazione. 22

Browser Software per la navigazione sul web, permette di accedere a risorse tramite indirizzi URL. 23

Client Nei sistemi di reti informatiche ad architettura client-server, il client svolge il ruolo di utilizzatore dei servizi offerti dal server. Indica generalmente, anche il software in grado di accedere ai servizi server: il browser ne è un esempio. 17, 21, 23–25, 32–34

Cookie Strumento del protocollo HTTP. I cookie vengono utilizzati dalle applicazioni lato server per archiviare informazioni da utilizzare in sessioni future sul lato client. 20

DBMS Data Base Management System. 11

Framework In informatica e specificamente nello sviluppo software, è un'architettura logica sul quale un software può essere progettato e realizzato; spesso fornisce strumenti preconfezionati per agevolare gli sviluppatori. 6, 8–10, 13, 18, 23, 26

Frontend La parte delle applicazioni visibile all'utente che ne fa utilizzo. Genericamente funge da interfaccia tra utente e software che implementa le varie logiche elaborando i dati trasmessi. 7, 20, 29, 31, 34

Gateway È uno strumento che collega due reti informatiche di tipo diverso operando sia al livello di rete che ai livelli superiori del modello ISO/OSI. Lo scopo principale è di veicolare i pacchetti di rete all'esterno di una rete locale (LAN). 14, 17, 29, 32, 36

GET Metodo del protocollo HTTP. 36

Google Motore di ricerca web introdotto sul mercato il 15 Settembre 1997. 33

Header Header, in informatica e nella commutazione di pacchetto, indica quella parte di un pacchetto, o più in generale della PDU, che contiene informazioni di controllo necessarie al funzionamento della rete. 22–24, 35, 36

HTTP Acronimo di hypertext transfer protocol, usata in informatica, e in particolare nella tecnica della rete Internet, che indica il protocollo di accesso alle informazioni. 11, 20, 23, 34, 36

Java In informatica è un linguaggio di programmazione ad alto livello, orientato agli oggetti e a tipizzazione statica, che si appoggia sull'omonima piattaforma software di esecuzione, specificamente progettato per essere il più possibile indipendente dalla piattaforma hardware di esecuzione. 8, 9, 11, 12, 26

JSON Acronimo di JavaScript Object Notation, è un formato adatto all'interscambio di dati in applicazioni client/server. 21

JWE JSON Web Encryption: contenitore che porta differenti tipi di asserzioni o claims dal Client all'applica- zione in maniera sicura, in questa particolare implementazione l'intero contenuto viene criptato. 24

JWT JSON Web Token: contenitore che porta differenti tipi di asserzioni o claims dal Client all'applica- zione in maniera sicura. 21, 23, 25, 29, 36

Login Processo di inserimento delle credenziali per accedere ad un determinato account. 21, 23, 33

Maven Apache Maven è uno strumento di gestione di progetti software basati su Java e compilazione automatica. 10

OAuth Open Authentication, è un protocollo di sicurezza open-source per il trasferimento sicuro di dati tra diversi applicativi. 21, 23, 33

OnlineVoting Nome dell'applicazione sviluppata, oggetto del capitolo 4. 27, 32

Payload Sinonimo di Body. 22

POJO Plain Old Java Object, ovvero una classe Java classica. 12

Repository È un archivio strutturato (spesso ad accesso aperto e libero) che raccoglie, conserva e mette a disposizione dati e materiali in formato digitale. 12, 14, 16

Server In informatica e telecomunicazioni è un componente o sottosistema informatico di elaborazione e gestione del traffico di informazioni che fornisce, a livello logico e fisico, un qualunque tipo di servizio ad altre componenti (tipicamente chiamate clients, cioè clienti) che ne fanno richiesta attraverso una rete di computer o anche direttamente in locale su un computer. 11, 19, 21, 32–34

Signature Traduzione inglese di firma. Viene utilizzata per la generazione della parte di comunicazione criptata tramite una stringa segreta, appunto la signature. 21, 22

Software Un programma codificato in linguaggio macchina o in linguaggio di programmazione (codice sorgente), memorizzato su uno o più supporti fisici. 4, 5, 8, 26, 34

Spring È un framework open-source per lo sviluppo di applicazioni su piattaforma Java. 6, 8–11, 13, 18, 32

Token Stringa di caratteri codificata o criptata, utilizzata dagli utenti come "lascia passare" per accedere a risorse di tipologia sesibile. 21–25, 34, 35

URL Uniform Resource Locator. 17, 20, 29

Web World Wide Web, abbreviato con Web, è uno dei principali servizi di Internet, che permette di navigare e usufruire di un insieme molto vasto di contenuti amatoriali e professionali (multimediali e non) collegati tra loro attraverso legami (link), e di ulteriori servizi accessibili a tutti o ad una parte selezionata degli utenti di Internet. 5, 20, 21, 23

XML È un metalinguaggio per la definizione di linguaggi di markup, ovvero un linguaggio marcatore basato su un meccanismo sintattico che consente didefinire e controllare il significato degli elementi contenuti in un documento oin un testo. 9, 11

Bibliografia

[1] J. Carnell, *Spring Microservices in Action*, 31 Luglio 2017, Manning Publications Co.

[2] Dinesh Rajput, *Spring 5 Design Patterns*, 6 Ottobre 2017, Packt Publishing

[3] L. Spilca, *Spring Security in Action*, 3 Novembre 2020, Manning Publications Co.

[4] P. Siriwardena, *Microservices Security in Action*, 5 Ottobre 2020, Manning Publications Co.

[5] C. Walls, *Spring in Action*, 27 Novembre 2014, Manning Publications Co.

[6] S.Chacon, B. Straub, *Pro git*, 9 Novembre 2014, Apress

[7] Sito ufficiale Spring Tool Suite https://spring.io/tools

[8] Sito ufficiale GitHub https://github.com/

[9] Sito ufficiale Git https://git-scm.com/doc

[10] Sito ufficiale Postman https://www.postman.com/

[11] ”Key annotations you need to know when working with JPA and Hibernate” https://thorben-janssen.com/key-jpa-hibernate-annotations

[12] ”Spring Data – One API To Rule Them All?” https://www.infoq.com/articles/spring-data-intro/

[13] ”Spring Security Overview” https://auth0.com/blog/spring-security-overview/

[14] ”Get Started with JSON Web Tokens” https://auth0.com/learn/json-web-tokens/

[15] ”JWT.IO allows you to decode, verify and generate JWT.” https://jwt.io/

[16] BCrypt encoder e decoder https://bcrypt-generator.com/

[17] Esempio di sviluppo di applicazione a microservizi ”Microservices using SpringBoot”