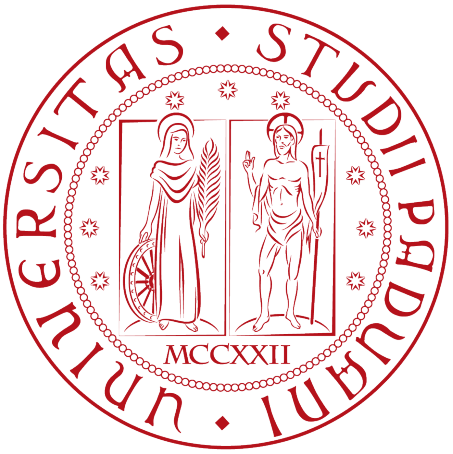
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Documento Tecnico

Documento per sviluppatori e manutentori

Anno Accademico 2020-2021

Indice

1 Introduzione **1**
1.1 Scopo del progetto 1

2 Prima implementazione: Java **3**
2.1 Tecnologie usate 3
2.2 Installazione 3
2.3 Architettura 4
2.3.1 Web Client Builder 4
2.3.2 Porte utilizzate 5
2.4 Documentazione API 5
2.4.1 Gestione dell'utente 5
2.4.2 Gestione candidati 7
2.4.3 Gestione del voto 8
2.5 Diagrammi di sequenza 9
2.5.1 Diagramma di sequenza - ottenimento di tutti gli utenti 9
2.5.2 Diagramma di sequenza per l'inserimento di un nuovo candidato 10
2.5.3 Diagramma di sequenza per l'inserimento di un voto 10

3 Seconda implementazione: Javascript **11**
3.1 Tecnologie usate 11
3.2 Installazione 11
3.3 Architettura 12
3.3.1 Circuit Breaker 13
3.3.2 Porte utilizzate 14
3.4 Documentazione richieste API 15
3.4.1 Gestione degli utenti 15
3.4.2 Gestione dei candidati 16
3.4.3 Gestione del voto 17
3.5 Diagrammi di sequenza 18
3.5.1 Diagramma di sequenza - ottenimento di tutti gli utenti 18
3.5.2 Diagramma di sequenza - inserimento nuovo candidato 18
3.5.3 Diagramma di sequenza - inserimento di un voto 18
3.6 Testing 19

4 Documentazione esterna consultata per il progetto **21**

Elenco delle figure

2.1 Architettura a microservizi 4
2.2 Esempio di uso del WebClientBuilder 5
2.3 esempio di body del post per l'inserimento di un nuovo utente 6
2.4 esempio di body del post per l'inserimento di un candidato 7
2.5 Body del Post per l'inserimento di un voto 8
2.6 Diagramma di sequenza per l'ottenimento di tutti gli user 9
2.7 Diagramma di sequenza per l'inserimento di un nuovo candidato . . . 10
2.8 Diagramma di sequenza per l'inserimento di un voto 10

3.1 Architettura dell'implementazione con javascript 12
3.2 Immagine esemplificativa dell'implementazione del circuit breaker . . . 13
3.3 Immagine esemplificativa del file main.ts 14
3.4 Caption 15
3.5 Caption 16
3.6 Caption 17
3.7 Caption 17
3.8 Diagramma di sequenza - ottenimento di tutti gli utenti 18
3.9 Diagramma di sequenza - inserimento nuovo candidato 18
3.10 Diagramma di sequenza - inserimento di un voto 19

Capitolo 1

Introduzione

Lo scopo del seguente documento è quello di evidenziare entrambe le architetture create per l'implementazione del backend della applicazione di voting online, insieme alle modalità e alle convenzioni da seguire per manutenere ed estendere il progetto didattico.

1.1 Scopo del progetto

Lo scopo del progetto, oltre all'implementazione con architettura a microservizi, è l'analisi comparativa tra le due modalità di implementazione, ovvero Java e JavaScript. In questo documento non verrà esposta tale analisi, nè critiche personali riguardo le implementazioni. Non verrà espresso quindi nessun giudizio, ma verranno mostrate le funzionalità esposte dal progetto.

Il progetto consiste fondamentalmente nell'implementazione di non molte funzionalità, in modo da non ostacolare l'implementazione dell'architettura a microservizi.

tutte le funzionalità e le convenzioni per manutenere il progetto didattico saranno divise secondo le due implementazioni svolte durante il periodo di stage.

Capitolo 2

Prima implementazione: Java

2.1 Tecnologie usate

Il linguaggio di programmazione scelto è Java e come framework è stato usato Spring Boot.I moduli di Spring Boot usati sono:Spring Web, Spring Data JPA (ORM), PostgreSQL Driver, ed il database è di tipo PostgreSQL.

2.2 Installazione

Per eseguire l'applicazione basta scaricare la repository dal seguente link:
https://github.com/falsettimatteo/uni-stage.git.
Dopodiché è necessario entrare nella directory VotingOnline_Java. All'interno di tale directory sono presenti tre folder, ognuno dei quali rappresenta un microservizio implementato.

Per eseguire un microservizio basta entrare nella sua directory ed eseguire il seguento comando da terminale:
./mvnw spring-boot:run

Per far funzionare l'intera applicazione è neccessario avviare tutti i microservizi.

2.3 Architettura

L'architettura a microservizi è stata implementata usando 3 microservizi: l'User manager Service che gestisce l'user, il Nominee Catalog Service che gestisce i candidati e il Voting Manager Service che gestisce il sistema di voto.

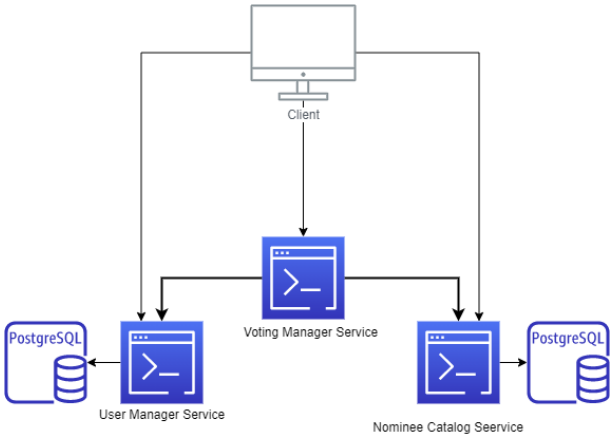


Figura 2.1: Architettura a microservizi

2.3.1 Web Client Builder

In questa implementazione non esiste un punto unico di entrata nel backend. Ovvero non è stato implementato un Api Gateway. I microservizi sono indipendenti e comunicano tra loro mediante richieste http.
Infatti il microservizio Voting Manager Service, che è l'unico microservizio che necessita la collaborazione con altri microservizi, per le chiamate agli altri microservizi usa il WebClientBuilder. Il WebClientBuider permette di fare chiamate asincrone asfruttando il builder pattern. Una possibile alternativa molto usata è il Rest Template, ma la documentazione di Spring Boot la sconsiglia in quanto potrebbe essere deprecata in futuro.
Per usare il WebClient basta costruire una richiesta immettendo, per una richiesta di tipo POST, l'uri ovvero l'indirizzo a cui vogliamo mandare la richiesta ed il body, ovvero l'oggetto che vogliamo mandare insieme alla POST request. Dopodiché basta chiamara la funzione retrieve per ottenere la risposta, specificare il tipo di ritorno con bodyToMono e infine chiamare la funzione block.

```java
@RequestMapping(value = "/register",consumes = "application/json",method = RequestMethod.POST)
public String registerUser(@RequestBody User user) {
    return WebClientBuilder.build().post()
            .uri("http://localhost:8080/user/add")
            .body(Mono.just(user), User.class)
            .retrieve()
            .bodyToMono(String.class)
            .block();

}
```

Figura 2.2: Esempio di uso del WebClientBuilder

2.3.2 Porte utilizzate

I microservizi sono eseguiti in locale sulle seguenti porte:

* User Manager Service: 8000
* Nominee Catalog Service: 8001
* Voting Manager Service: 8002

In caso si vogliano modificare le porte basta aprire il file applicationi.properties che è presente nella directory src/main/recources e modificare il numero della porta.
Nello stesso file è presente anche la configurazione del database PosgreSQL. Il database è eseguito sulla porta 5432.

Le dipendenze di ogni microservizio si trovano nel file pom.xml presente nella directory src

2.4 Documentazione API

2.4.1 Gestione dell'utente

La gestione degli utenti avviene completamente nel microservizio User Manager Service. Il servizio quando avviato sarà distrubuito in locale sulla porta 8000.

Richiesta di ottenimento di tutti gli utenti

* **url**: http://localhost:8000/user/all ,
* **tipo di richiesta**: GET,
* **tipo di ritorno**: List<User>
* La chiamata ritorna una lista contenente tutti gli utenti presenti nell'applicaizone.

Ricerca di un utente

* **url**: http://localhost:8000/user/userid ,
* **tipo di richiesta**: GET,
* **tipo di ritorno**: boolean
* "userid" è il parametro che rappresenta l'id dell'utente
* La chiamata ritorna true se l'utente è presente nell'applicazione

Inserimento di un nuovo utente

* **url**: http://localhost:8000/user/add ,
* **tipo di richiesta**: Post,
* **Content-Type**: application/json,
* **paramteri**: username: nome dell'utente, password: password dell'utente
* **tipo di ritorno**: String
* La chiamata ritorna una stringa che rappresenta la riuscita o non dell'inserimento

```
1  {
2      "username": "nomeUser",
3      "password": "password"
4  }
5
```

Figura 2.3: esempio di body del post per l'inserimento di un nuovo utente

2.4.2 Gestione candidati

La gestione dei candidati avviene completamente nel microservizioNominee Catalog Service. Il servizio quando avviato sarà distrubuito in locale sulla porta 8001.

Ricerca di tutti i candidati

* **url**: http://localhost:8001/all ,
* **tipo di richiesta**: GET,
* **tipo di ritorno**: List<Nominee>
* La chiamata ritorna una lista contenente tutti i candidati

Inserimento di un nuovo candidato

* **url**: http://localhost:8001/add ,
* **tipo di richiesta**: Post,
* **Content-Type**: application/json,
* **paramteri**: id: id del candidato, nominee: nome del candidato,
* **tipo di ritorno**: String
* La chiamata ritorna una stringa che rappresenta la riuscita dell'inserimento

```
1  {
2      "nominee": "Trump"
3  }
4
```

Figura 2.4: esempio di body del post per l'inserimento di un candidato

2.4.3 Gestione del voto

La gestione dello voto avviene completamente nel microservizio Voting Manager Service. Il servizio quando avviato sarà distrubuito in locale sulla porta 8002.
Conteggio dei voti di un candidato

* **url**: http://localhost:8002/counting/nomineeid ,
* **tipo di richiesta**: GET,
* **tipo di ritorno**: Int
* nomineeid rappresenta il parametro passato dell'id del candidato
* La chiamata ritorna un intero che rappresenta il numero di voti dati ad un candidato

Inserimento di un nuovo voto

* **url**: http://localhost:8002/vote ,
* **tipo di richiesta**: Post,
* **Content-Type**: application/json,
* **paramteri**: usernameid: id del votante, nomineeid: id del candidato,
* **tipo di ritorno**: String
* La chiamata ritorna una stringa che rappresenta la riuscita dell'inserimento

```
1  {
2      "usernameid": "2",
3      "nomineeid": "3"
4  }
5
```

Figura 2.5: Body del Post per l'inserimento di un voto

2.5 Diagrammi di sequenza

Verranno mostrati in seguito i diagrammi di sequenza dell'applicazione. Visto che i metodi sono molto semplici e simili tra loro, verrà mostrato in seguito un diagramma per microservizioservizio.

2.5.1 Diagramma di sequenza - ottenimento di tutti gli utenti

il seguente diagramma mostra la sequenza di chiamate per l'ottenimento di tutti gli utenti presenti nell database dell'applicazione. Verrà mostrata soltanto per questa richiesta in quanto il resto delle richieste si comporta nello stesso modo, solamente chiamando metodi diversi. Il nome dei metodi ne facilità il comprendimento.

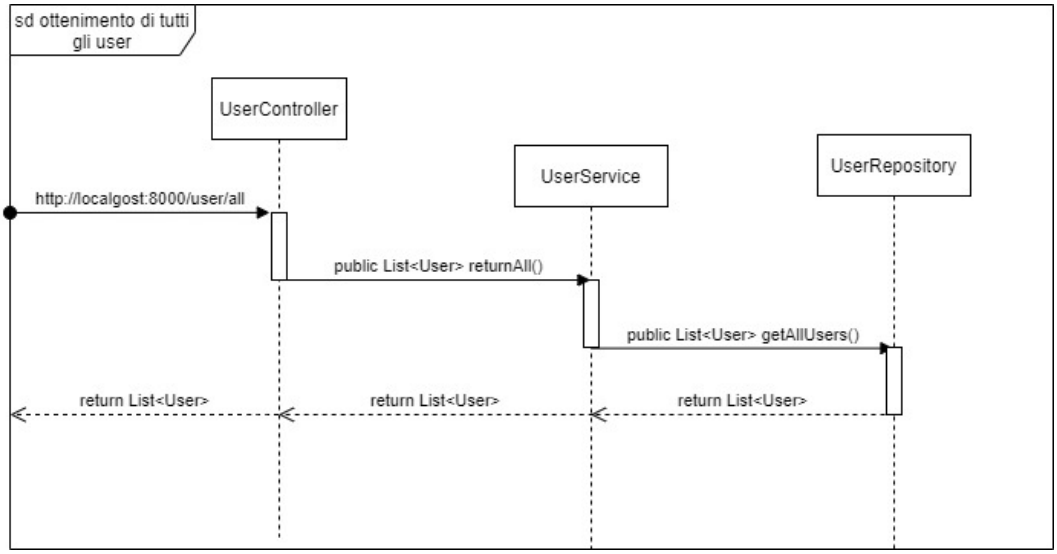


Figura 2.6: Diagramma di sequenza per l'ottenimento di tutti gli user

2.5.2 Diagramma di sequenza per l'inserimento di un nuovo candidato

Il seguente diagramma mostra la sequenza di chiamate per linserimento di un nuovo candidato.

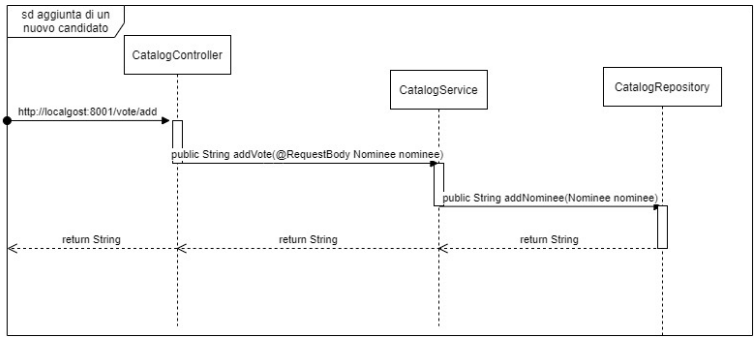


Figura 2.7: Diagramma di sequenza per l'inserimento di un nuovo candidato

2.5.3 Diagramma di sequenza per l'inserimento di un voto

Il seguente diagramma mostra la sequenza per l'inserimento di un nuovo voto.

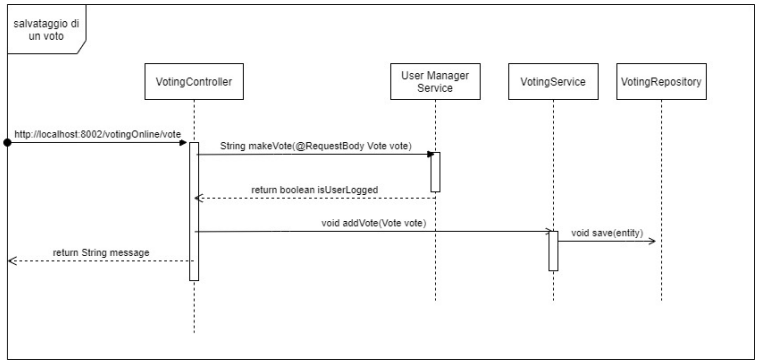


Figura 2.8: Diagramma di sequenza per l'inserimento di un voto

Capitolo 3

Seconda implementazione: Javascript

3.1 Tecnologie usate

Il llinguaggio di programmazione usato per la seconda implementazione è stato javascript con l'ausilio del suntime system NodeJs e framework NestJs per l'implementazione dell'architettura a microservizi.
Inoltre è stato usato Prisma come ORM e postgres come database.
Per l'implementazione del circuit breaker è stata usata la libreira Opossum.

3.2 Installazione

per usare l'applicazione basta, dopo aver scaricato la directory dalla repo:
https://github.com/falsettimatteo/uni-stage.git
bisogna entrare all'interno della directory VotingOnline_Javascript. Dopodiché bisogna entrare in ognuna delle directory presenti in VotingOnline_Javascript, (ovvero api-gateway, user-manager, voting-catalog e voting-manager) che rappresentatno ognuna un microservizio ed usare il comando da terminale:
npm run start.

3.3 Architettura

Per questa implementazione è stato possibile applicare alcune delle best practices delle architetture a microservizi, ovvero ,per quanto riguarda l'architettura è stato implementato un API Gateway [G].
In seguito una rappresentazione figurale dell'architettura

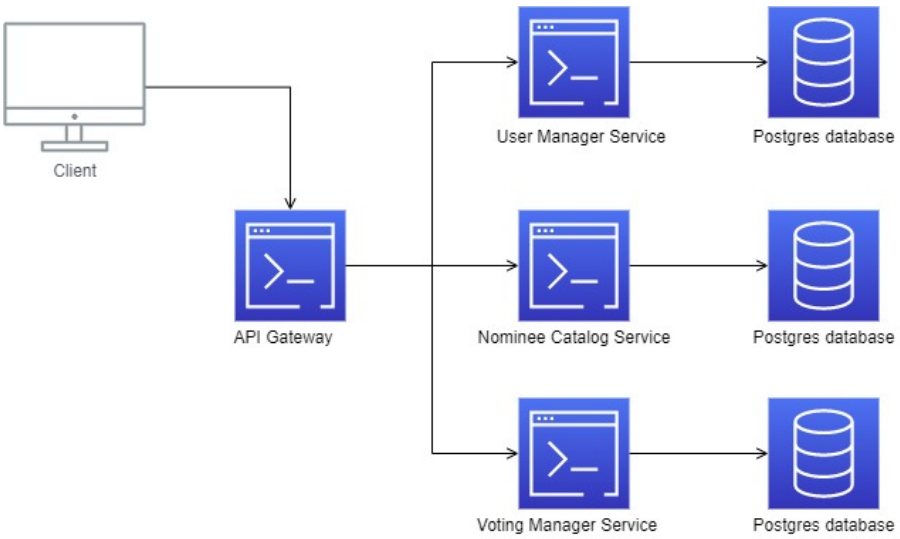


Figura 3.1: Architettura dell'implementazione con javascript

Ogni microservizio è stato implementato sfruttando il framework NestJs, in particolare il metodo createMicroservice presente all'interno dei tre microservizi user Manager Service, Nominee Catalog Service e Voting Manager Service, fa in modo di isolare ogni microservizio da possibili chiamate del cliente. Ovvero il controller dei tre microservizi non risponde a chiamate API ma sfrutta il pattern Observer. Con l'ulteriore implementazione del API Gateway si ottiene così un unico punto di entrata del client verso il back end.

Nei tre microservizi che gestiscono gli utenti i candidati e i voti, è inoltre presente, all'interno della cartella prisma, il modello per la generazione di entità per il popolamento del database postrgres. Infatti all'interno del file user.service.ts troviamo l'implementazione delle funzioni di gestione delle entità del database. possiamo trovare le stesse implementazioni per tutti e tre i microservizi.

Infatti la risposta ad una chiamata verso ogni controller avviene proprio in questo modo. La richiesta ottenuta dal controller chiama una funzione del file con estenstione .service.ts.

Nel file app.module invece troviamo le dipendenze del microservizio, ovvero i controllers, providers e imports.

3.3.1 Circuit Breaker

è stato implementato ache il pattern circuit breaker, con l'ausilio della libreria opossum. Il circuit breaker viene creato con il metodo new CreateBreaker(funz, options) al quale viene passata la funzione alla qualee si vule applicare il circuit breaker (funz) e le opzioni per i valori di timeout e percentuel di funzioni fallite.

Per implementare la funzione di fallback basta chiamare il metodo fallback(fb) e passare come parametro la funzione che si vuole chiamare in caso di fallback (fb), mentre per chiamare la funzione del circuit breaker bisogna chiamare il metodo fire(). . In seguito un esempio di implementazione di un circuit breaker

```
private options = {
  timeout: 3000, // If our function takes longer than 3 seconds, trigger a failure
  errorThresholdPercentage: 50, // When 50% of requests fail, trip the circuit
  resetTimeout: 30000 // After 30 seconds, try again.
};

  //Controller per User Manager service
@Get("/allUsers")
  async getAllUsersBreaker(){
    const breaker = new CircuitBreaker(this.apiService.getAllUser(), this.options);
    breaker.fallback(() => {
      return "FALLBACK : User manager service down";
    })
    return await breaker.fire();
  }
```

Figura 3.2: Immagine esemplificativa dell'implementazione del circuit breaker

3.3.2 Porte utilizzate

una volta inizializzati i microservizi sono attivi nelle seguenti porte:

* **User Manager Service**: 3000,
* **Nominee Catalog Service**: 3001,
* **Voting Manager Service**: 3002,
* **Api Gateway**: 3003,

Per modificare le porte, basta modificare l'opzione "port" all'interno del metodo createMicreservices. Tale metodo si trova nel file main.ts che si trova nella cartella src (src/main.ts)

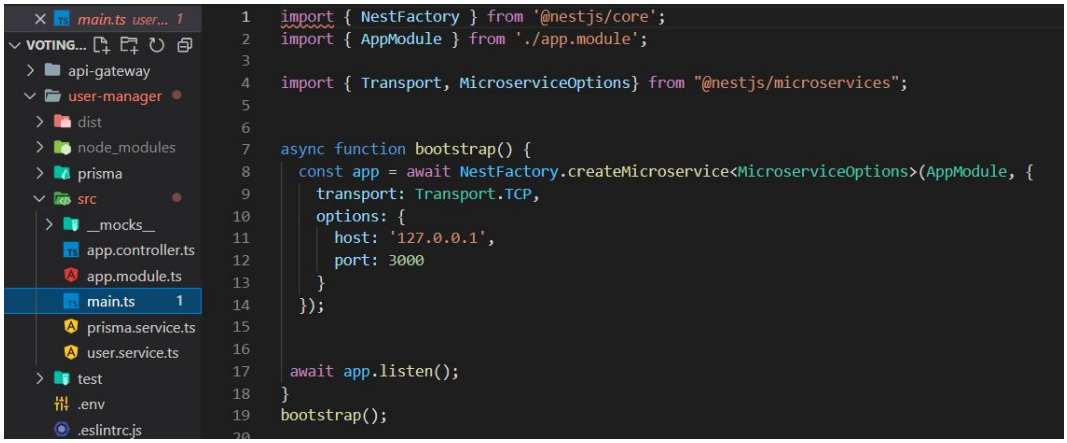


Figura 3.3: Immagine esemplificativa del file main.ts

3.4 Documentazione richieste API

essendo implementato un unico punto di entrata, ovvero l'API Gateway tutte le richieste alla parte server del client iniziano con lo stesso url, ovvero: http://localhost:3003/

3.4.1 Gestione degli utenti

Richiesta di ottenimento di tutti gli utenti

* **url**: http://localhost:3003/allUsers ,
* **tipo di richiesta**: GET,
* **tipo di ritorno**: array JSON
* La chiamata ritorna una lista contenente tutti gli utenti presenti nell'applicaizone.

Verifica della presena di un utente nell'applicazione

* **url**: http://localhost:3003/isLogged/:username/:password ,
* **tipo di richiesta**: GET,
* **tipo di ritorno**: boolean
* username è il nome dell'utente e password è la sua password
* La chiamata ritorna true se l'utente è presente nell'applicazione.

Inserimento di un nuovo utente

* **url**: http://localhost:3003/addUser ,
* **tipo di richiesta**: Post,
* **Content-Type**: application/json,
* **parametri**: username: nome dell'utente, password: password dell'utente
* **tipo di ritorno**: string
* La chiamata ritorna una stringa con l'esito dell'inserimento,

```
1  {
2      "username": "nomeUser",
3      "password": "password"
4  }
5
```

Figura 3.4: Caption

3.4.2 Gestione dei candidati

Ottenimento di tutti i candidati

* **url**: http://localhost:3003/allNominees,
* **tipo di richiesta**: GET,
* **tipo di ritorno**: array JSON
* La chiamata ritorna un array JSON contenente tutti i candidati.

Inserimento di un nuovo candidato

* **url**: http://localhost:3003/addNominee ,
* **tipo di richiesta**: Post,
* **Content-Type**: application/json,
* **parametri**: nominee: nome del candidato,
* **tipo di ritorno**: string
* La chiamata ritorna una stringa con l'esito dell'inserimento,

Figura 3.5: Caption

3.4.3 Gestione del voto

Inserimento di un nuovo voto

* **url**: http://localhost:3003/vote ,
* **tipo di richiesta**: Post,
* **Content-Type**: application/json,
* **parametri**: usernameid: id dell'utente, nomineeid: id del candidato,
* **tipo di ritorno**: string
* La chiamata ritorna una stringa con l'esito dell'inserimento,

Figura 3.6: Caption

Inserimento di un nuovo voto con timeout attenzione, questo metodo che fa andare in timeout il microservizio Voting Manager Service

* **url**: http://localhost:3003/voteTimeout ,
* **tipo di richiesta**: Post,
* **Content-Type**: application/json,
* **parametri**: usernameid: id dell'utente, nomineeid: id del candidato,
* **tipo di ritorno**: string
* La chiamata ritorna una stringa con l'esito dell'inserimento,

Figura 3.7: Caption

3.5 Diagrammi di sequenza

3.5.1 Diagramma di sequenza - ottenimento di tutti gli utenti

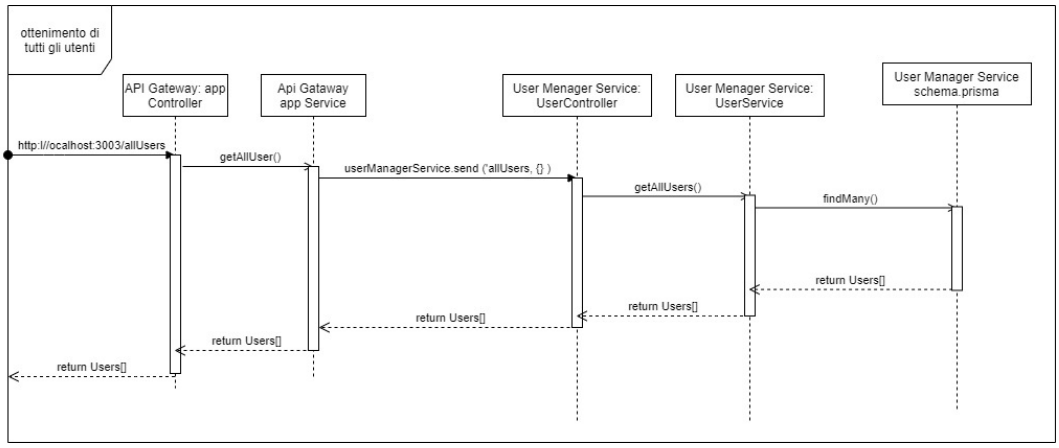


Figura 3.8: Diagramma di sequenza - ottenimento di tutti gli utenti

3.5.2 Diagramma di sequenza - inserimento nuovo candidato

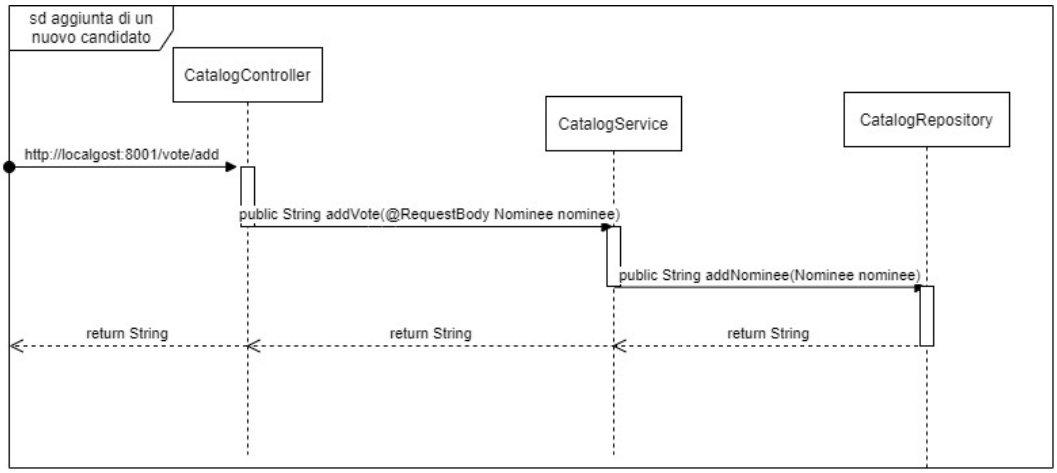


Figura 3.9: Diagramma di sequenza - inserimento nuovo candidato

3.5.3 Diagramma di sequenza - inserimento di un voto

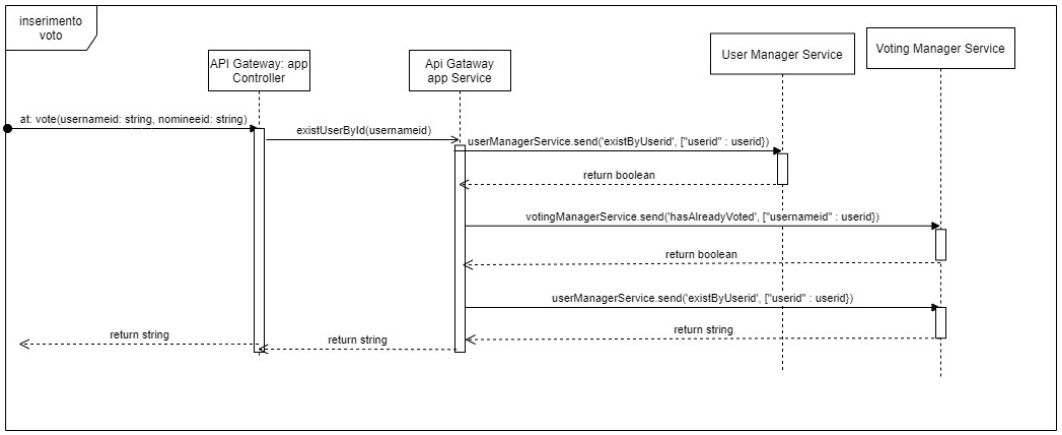


Figura 3.10: Diagramma di sequenza - inserimento di un voto

3.6 Testing

Per la parte javascript sono stati scritti i test di unità. I test sono presenti nella directory test presente in tutti i microservizi sviluppati ai quali è stato possibile scrivere test di unità (quindi tutti tranne l'api Gateway). per eseguire i test basta eseguire il comando jarn test da terminale.

Nella seguente tabella sono descritti i test di unità implementati.
Il codice identificativo dei test è definito dall'abbreviazione TU che sta per test di unità seguitoo dal numero identificativo del test.

ID test	Descrizione	Esito
TU0	si verifica l'ottenimento di tutti gli utenti	Superato
TU1	si verifica la ricerca di un singolo utente per username	Superato
TU2	si verifica la ricerca di un singolo utente per ID	Superato
TU3	si verifica l'inserimento di un nuovo uente	Superato
TU4	si verifica il messaggio di errore in caso di inserimento non riuscito	Superato
TU5	si verifica la ricerca di un candidato per ID	Superato
TU6	si verifica l'ottenimento di tutti i candidati	Superato
TU7	si verifica l'inserimento di un nuovo candidato	Superato
TU8	si verifica il corretto conteggio dei voti	Superato
TU9	si verifica il corretto inserimento di un nuovo voto	Superato
TU10	si verifica il controllo su utenti che hanno già votato	Superato

Capitolo 4

Documentazione esterna consultata per il progetto

Architettura:

* **Microservizi**: https://www.nginx.com/blog/introduction-to-microservices/
* **Api Gateway**: https://microservices.io/patterns/apigateway.html
* **Circuit breaker**: https://martinfowler.com/bliki/CircuitBreaker.html

Implementazione Java:

* **Spring Boot**: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/
* **Web Client**: https://spring.getdocs.org/en-US/spring-framework-docs/docs/spring-web-reactive/webflux-client/webflux-client-builder.html

Implementazione Javascript:

* **NodeJS**: https://nodejs.org/it/docs/
* **NestJS**: https://docs.nestjs.com/microservices/basics
* **Prisma**:https://www.prisma.io/docs/
* **Opossum**: https://github.com/nodeshift/opossum