

Smart Waste Collection

Applicazioni e Servizi Web

Marta Spadoni - 0000978305 {marta.spadoni@studio.unibo.it}

12 Settembre 2022

Indice

1	Introduzione	2
2	Requisiti	3
2.1	Requisiti generali	3
2.2	Requisiti per i manager	3
2.3	Requisiti per i cittadini	4
3	Design	6
3.1	Design Architetturale	6
3.2	Design delle interfacce	7
4	Tecnologie	12
5	Codice	14
5.1	Gestione autenticazione utenti	14
5.2	Gestione socket	14
5.3	Esempio Pinia Store	15
5.4	Modelli MongoDB	16
6	Test	18
6.1	Test del codice	18
6.2	Test di usabilità	18
7	Deployment	20
8	Conclusioni	21

Capitolo 1

Introduzione

Il progetto nasce dall'idea di realizzare un sistema “smart” per la gestione della raccolta differenziata, in particolare il progetto mira a migliorare un sistema preesistente dotando di sensori i cassonetti della spazzatura e i camioncini utilizzati per la raccolta. In questo modo è possibile monitorare in tempo reale lo stato dei vari punti di raccolta e dei camioncini durante le missioni. Inoltre, il nuovo sistema introduce la possibilità di richiedere la raccolta a domicilio dei rifiuti ingombranti, evitando al singolo cittadino di doversi recare al punto di smaltimento, e offre l'opportunità di inviare reclami sullo stato generale del sistema.

L'elaborato è un progetto integrato con i corsi di Pervasive Computing e Laboratorio di Sistemi Software. Per questo motivo, dei vari cassonetti e dei camioncini vengono creati i relativi Digital Twin sulla piattaforma Azure, l'intero sistema si basa su un'architettura a microservizi e l'analisi e la progettazione sono state svolte seguendo i principi del Domain Driven Design.

Capitolo 2

Requisiti

Le caratteristiche e le funzionalità del sistema sono state definite simulando un'intervista con l'ipotetico committente del progetto, di seguito vengono riportati i requisiti relativi all'applicazione web.

2.1 Requisiti generali

- **Visualizzazione dei punti di raccolta:** anche previo login, ciascun utente del sistema deve poter essere in grado di consultare la mappa dell'area geografica coperta dal sistema, al fine di poter visualizzare la posizione dei vari punti di raccolta, cioè le stazioni in cui sono presenti i cassonetti per la raccolta dei rifiuti.
- **Consultazione dello stato di un punto di raccolta:** cliccando su un punto di raccolta presente sulla mappa, un utente deve poter visualizzare quanti e quali cassonetti sono presenti al suo interno e per ciascuno deve poter conoscere il tipo di rifiuto collezionato, il suo stato di operatività e quanto è pieno in percentuale.
- **Registrazione e Login:** il sistema deve permettere agli utenti di registrarsi e successivamente di effettuare il login in modo da poter accedere ad ulteriori funzionalità in base al ruolo ricoperto nel sistema, cittadino o manager dell'azienda.

2.2 Requisiti per i manager

- **Creazione dei punti di raccolta:** i manager devono poter aggiungere nuovi punti di raccolta sulla mappa in modo da allineare lo stato digitale del sistema con quello fisico. In particolare, il sistema deve permettere di creare un nuovo punto di raccolta partendo o da un luogo selezionato dalla mappa o da coordinate inserite dal manager stesso e deve consentire di aggiungere ad esso i vari cassonetti presenti.

- **Gestione dei punti di raccolta:** ciascun punto di raccolta, dopo essere stato creato, deve poter essere modificato, nello specifico il sistema deve permettere al manager di:
 - **Aggiungere un cassonetto:** selezionando il tipo di rifiuto raccolto (plastica, carta, vetro...) e la sua capacità.
 - **Rimuovere un cassonetto**
 - **Rimuovere il punto di raccolta**
- **Visualizzazione dei camioncini in missione:** i manager devono poter visualizzare sulla mappa anche i vari camioncini assegnati ad una missione di raccolta dei rifiuti. Devono infatti, poter monitorare il loro stato di occupazione e in generale l'andamento della missione stessa.
- **Gestione dei camioncini:** il sistema deve permettere ai manager di consultare lo stato della flotta di camioncini posseduti dell'azienda e di aggiungere o rimuovere degli elementi.
- **Visualizzazione delle missioni:** dall'applicazione web deve poter essere possibile visualizzare tutte le missioni di raccolta generate dal sistema, e per ciascuna si deve rendere possibile il monitoraggio dello stato:
 - **Consultazione delle informazioni generali:** tipologia della missione (ordinaria o straordinaria), tipo di rifiuto raccolto, camioncino al quale è assegnata e lo stato (in corso o completata).
 - **Visualizzazione degli step:** ogni missione è costituita da un insieme di step, cioè di punti di raccolta che devono essere raggiunti per svuotare il cassonetto relativo al rifiuto raccolto dalla missione. Il sito deve permettere di consultare quanti degli step sono stati completati e quanti se ne devono ancora terminare.
- **Gestione dei reclami:** i manager devono poter visualizzare tutti i reclami inviati al sistema e contrassegnarli come "chiusi" una volta risolto il problema riportato in ciascuno.
- **Visualizzazione delle prenotazioni:** il sistema deve permettere ai manager di visualizzare tutte le prenotazioni delle raccolte a domicilio richieste dai cittadini al fine di valutare l'utilità, l'efficacia e l'efficienza del nuovo servizio offerto dall'azienda.

2.3 Requisiti per i cittadini

- **Visualizzazione della proprio posizione:** il sistema deve consentire ai cittadini di determinare in modo semplice e veloce in quale punto di raccolta recarsi per conferire i propri rifiuti, per questo devono poter visualizzare sulla mappa anche la proprio posizione in modo da determinare quale sia il punto di raccolta più vicino e adatto a loro.

- **Creazione di una prenotazione:** il sistema deve fornire ai cittadini autenticati la possibilità di prenotare la raccolta a domicilio dei rifiuti considerati straordinari, come ad esempio gli elettrodomestici.
- **Notifiche relative alle prenotazioni:** il sistema deve inviare al cittadino che ha effettuato una prenotazione, una notifica ogni volta che il suo stato viene aggiornato, cioè se è stata assegnata ad una missione o se è stata completata.
- **Creazione di un reclamo:** i cittadini devono poter inviare al sistema dei reclami al fine di migliorare il sistema.
- **Visualizzazione prenotazioni e reclami:** ciascun cittadino deve poter consultare lo stato di tutte le sue prenotazioni e dei suoi reclami.

Capitolo 3

Design

3.1 Design Architettuale

Il design architeturale del sistema è stato fortemente influenzato dai principi chiave del Domain Driven Design e del Web Of Digital Twin. Insieme al team con cui vengono svolti i progetti per i corsi citati in apertura, si è deciso di realizzare un'architettura a microservizi di cui viene riportato uno schema riassuntivo in figura 3.1.

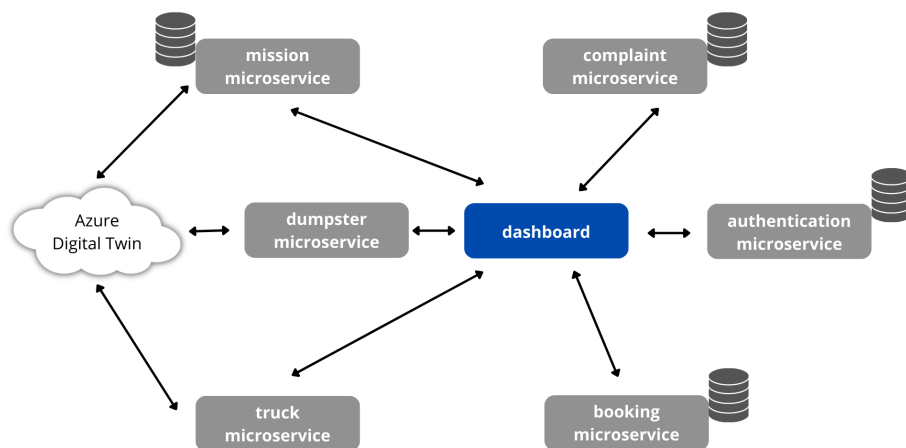


Figura 3.1: Design architettuale dell'intero sistema Smart Waste Collection

Personalmente mi sono occupata in modo esclusivo della progettazione dei microservizi relativi alle prenotazioni (*booking-microservice*) e all'autenticazione (*authentication-microservice*) e dell'applicazione web (*dashboard*).

I due microservizi sono stati progettati seguendo un'architettura REST e sono integrati ciascuno ad un proprio database MongoDB. Dell'API esposta dal *booking-microservice* è stata realizzata una specifica Swagger consultabile qui e di cui si riporta un riassunto in figura 3.2.

GET	/bookings	Get all bookings	▼ ↶
POST	/bookings	Create a booking	▼ ↶
GET	/bookings/{id}	Get a booking	▼ ↶
PUT	/bookings/{id}	Update a booking	▼ ↶
DELETE	/bookings/{id}	Delete a booking	▼ ↶
GET	/bookings/user/{userId}	Get all the bookings of a specific User	▼ ↶

Figura 3.2: Riassunto specifica Swagger booking-micorservice

3.2 Design delle interfacce

Per la progettazione delle interfacce della dashboard sono state utilizzate due tecniche: i Mockup e le Storyboard con disegni. I primi sono stati realizzati utilizzando AdobeXD e sono stati scelti come strumento per definire le interfacce principali in maniera più dettagliata di quanto fosse possibile con dei disegni. Mentre le Storyboard sono state disegnate perchè si è ritenuto il modo più semplice per esplicitare l'interazione con le interfacce.

In generale, tutte le interfacce dell'applicazione sono state progettate seguendo il principio KISS e le Euristiche di Nielsen, dunque si è scelto di realizzare interfacce semplici e lineari che fossero funzionali al loro scopo. In questo modo, i cambiamenti realizzati durante lo sviluppo rispetto a quanto progettato sono stati minimi, il più evidente è la scelta di sostituire la tabella utilizzata per visualizzare la lista delle missioni con una *DataGridView* cioè una visualizzazione a lista, al fine di rendere più agevole l'utilizzo di tale schermata dai dispositivi mobile.

Di seguito vengono riportati i mockup realizzati che sono stati utilizzati come base per lo sviluppo dell'interfaccia definitiva dell'applicazione.

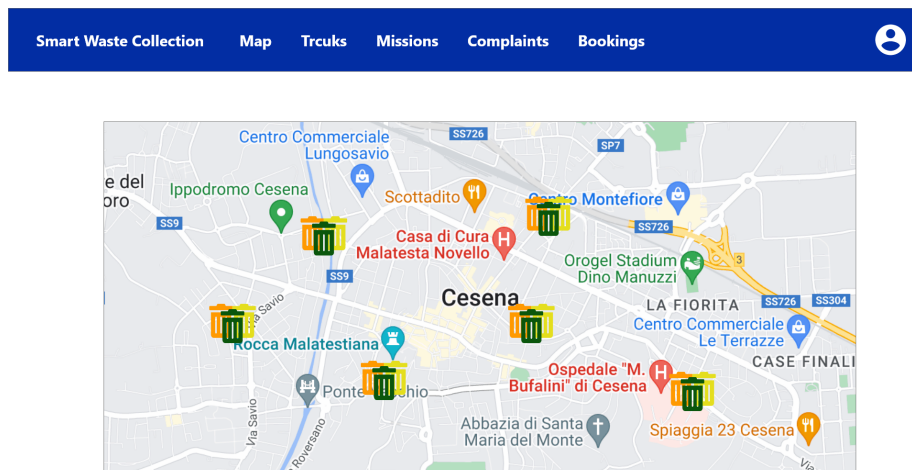


Figura 3.3: Mockup homepage - Mostra la mappa e i punti di raccolta

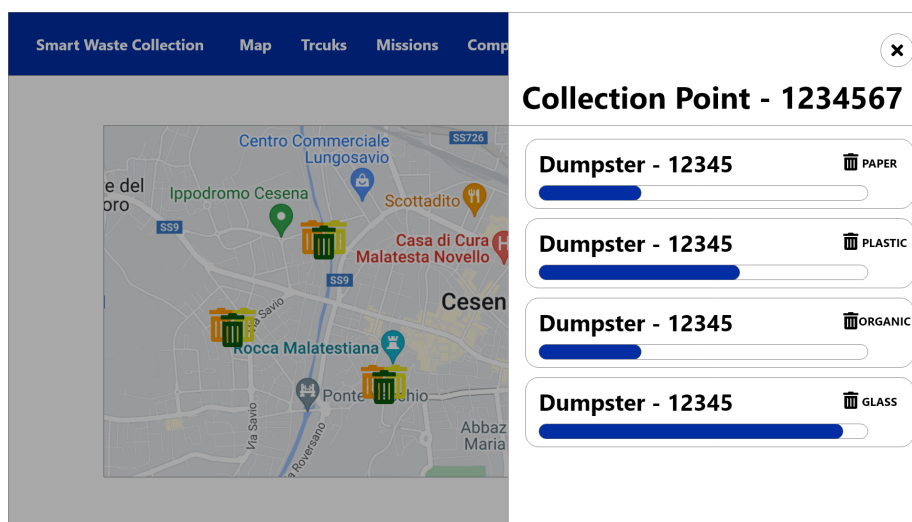
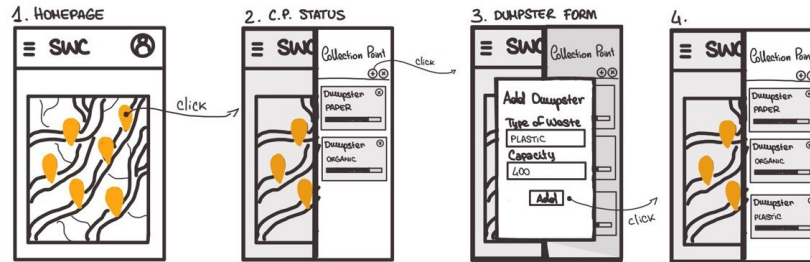


Figura 3.4: Mockup Sidebar - Mostra lo stato dei cassonetti in un punto di raccolta

I primi due Mockup realizzati sono quelli relativi all'homepage del sito, cioè la sezione relativa alla mappa, visualizzabile da qualsiasi utente del sistema. Nei Mockup viene anche presentata la struttura generale delle varie interfacce dell'applicazione, infatti in tutte le schermate sarà presente una Navbar che consentirà all'utente di spostarsi all'interno del sito.

La prima Storyboard (fig.3.5) realizzata mostra appunto le modalità di inte-

razione con la mappa presente nella schermata di homepage al fine di visualizzare le informazioni relative ad un punto di raccolta. Inoltre, come esplicitato dal commento in figura 3.5, dalla stessa Sidebar è possibile, unicamente per i manager, gestire il punto di raccolta, ad esempio, andando ad aggiungere un nuovo cassonetto.



NB: I BOTTONI PER LA MODIFICA DEL PUNTO DI RACCOLTA SONO PRESENTI SOLO PER I MANAGER
I CITTADINI, O IN GENERALE GLI UTENTI NON AUTENTICATI POSSONO SOLO VISUALIZZARE LO STATO (SCHERME 1 e 2)

Figura 3.5: Storyboards per la visualizzazione dello stato dei punti di raccolta e la loro modifica

Successivamente sono stati realizzati i Mockup di due schermate relative unicamente alla sezione dei Manager, cioè la visualizzazione della lista dei camioncini disponibili e del loro stato e la visualizzazione dello storico delle missioni.

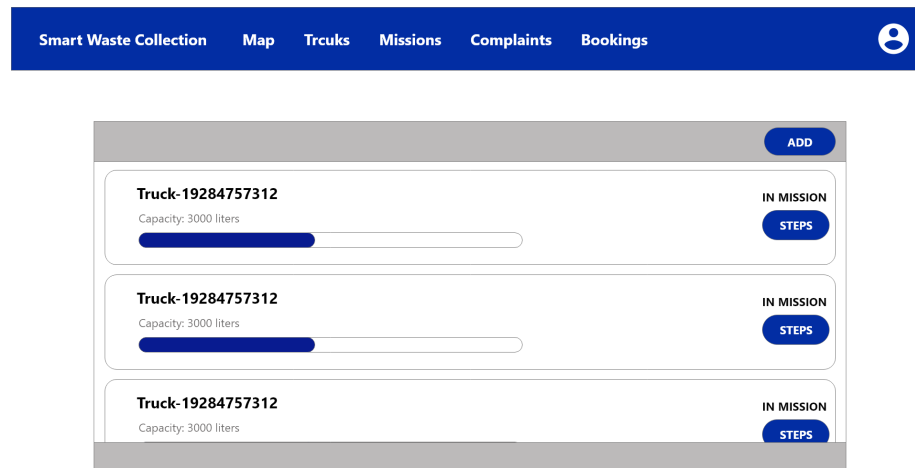


Figura 3.6: Mockup Trucks list - Mostra i camioncini disponibili e il loro stato





ID	Type Of Mission	Type Of Waste	STATUS	STEPS
173547	ORDINARY	PAPER	IN PROGRESS	
173547	ORDINARY	GLASS	IN PROGRESS	
173547	ORDINARY	ORGANIC	COMPLETED	
173547	ORDINARY	PAPER	COMPLETED	

Figura 3.7: Mockup Missions List - mostra lo storico delle missioni

La seconda Storyboard (fig. 3.8) riguarda invece la possibilità di consultare lo stato generale delle varie missioni generate dal sistema e di come si accede alla visualizzazione più dettagliata dei vari step di cui ciascuna è composta. Rispetto a quanto progettato, nella versione implementata del sistema si è preferito rendere ancora più minimale la grafica relativa a ciascuno step e di esprimere lo stato di ciascuno attraverso colori e icone.

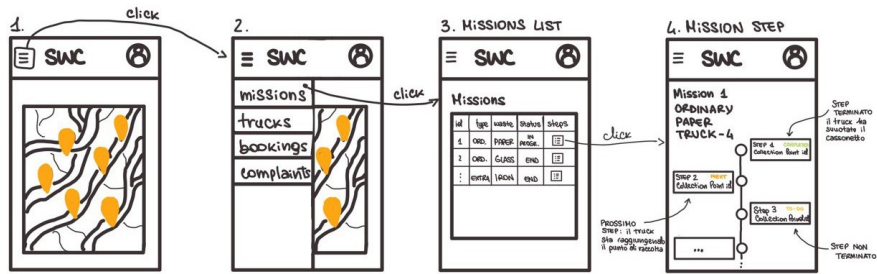


Figura 3.8: Storyboards per la visualizzazione delle missioni e dei loro step

Infine, sono stati prodotti i mockup delle sezioni Complaints e Bookings, rispettivamente dei reclami e delle prenotazioni per la raccolta a domicilio. Le due interfacce differiscono in modo minimale tra la visualizzazione "Cittadino" e quella "Manager", infatti nella prima è presente il pulsante per la creazione di un nuovo elemento mentre nella seconda no. Per questo motivo si riportano i Mockup solo della versione per il cittadino.

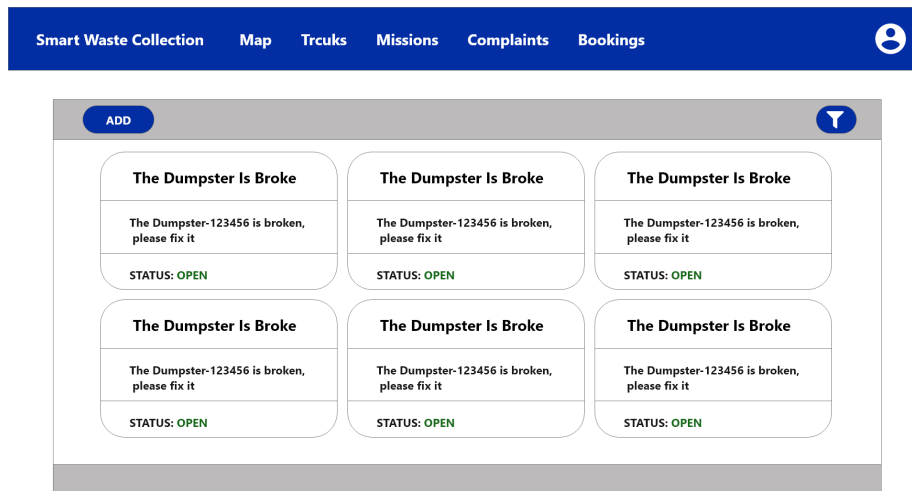


Figura 3.9: Mockup Complaints List - mostra lo storico dei reclami

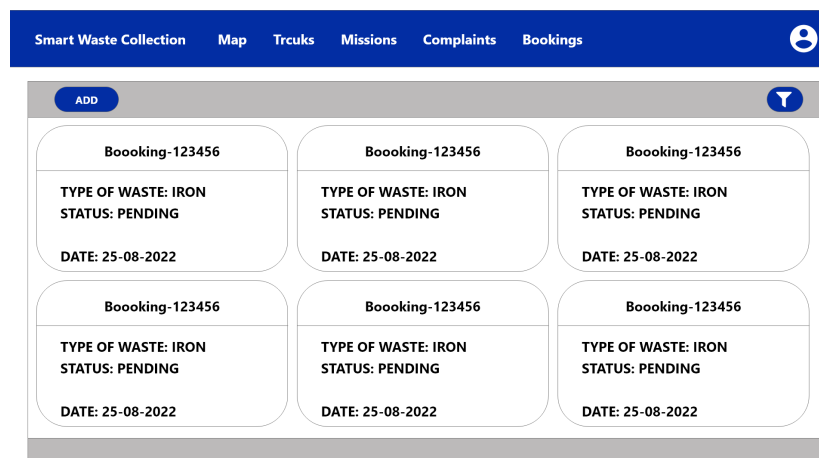


Figura 3.10: Mockup Booking List - mostra lo storico delle prenotazioni

Capitolo 4

Tecnologie

Le tecnologie utilizzate per realizzare il progetto sono molteplici, di seguito vengono presentate quelle ritenute più importanti.

Il *solution stack* adottato è **MEVN** dunque sono stati utilizzati:

- **Mongo DB** per quanto riguarda la persistenza lato booking-microservice e authentication-microservice. Nel primo, in particolare, si hanno collezioni per le prenotazioni e per le notifiche mentre nel secondo vengono memorizzati gli utenti (cittadini o manager) registrati.
- **Express.js** come framework Node.js per implementare i server web presenti nei due microservizi sopracitati.
- **Vue 3** framework Javascript lato client che consente di definire interfacce modulari e scelto soprattutto per la sua facilità d'uso e di apprendimento.
- **Node.js** come motore Javascript.

La scelta di utilizzare come framework lato client Vue3 ha portato poi ad integrare le seguenti tecnologie:

- **Vue-Router**, il router ufficiale di Vue.js che consente di implementare *Single Page Application*
- **Pinia**, la libreria ufficiale Vue.js (ha sostituito Vuex) per la gestione dello stato globale. Consente di semplificare la condivisione di informazione tra i diversi componenti dell'applicazione.
- **Vuelidate**, una libreria *model-based* per la validazione dei dati nei form. Consente di definire dei validators e delle regole in modo semplice, rendendo l'operazione di validazione dei form più semplice ed efficace.
- **PrimeVue**, una libreria di componenti Vue altamente personalizzabili che permette dunque di generare interfacce pulite e funzionali in modo semplice e veloce. Al suo interno si fa anche uso delle librerie *PrimeFlex*

e *PrimeIcons*, la prima è una libreria CSS che permette di modificare i componenti Prime mantenendoli sempre responsive mentre la seconda è una semplice libreria di icone.

Per quanto riguarda la gestione della mappa è stata utilizzata **Leaflet** una libreria javascript per l'integrazione di mappe interattive, nello specifico è stata utilizzata *vue-leaflet* una versione compatibile con Vue3.

Altre tecnologie fondamentali per lo sviluppo del progetto sono state:

- **Mongoose** libreria per la modellazione di oggetti MongoDB in Node.js, semplifica la creazione di query fornendo una sintassi più snella e la possibilità di eseguirle in modo asincrono, inoltre, permette di validare i dati in modo automatico.
- **Axios** client HTTP per Node.js basato su Promise, dunque su programmazione asincrona. Permette di realizzare richieste a servizi REST in modo semplice e veloce e fornisce la possibilità di gestire le risposte e gli errori in modo puntuale.
- **Socket.io** libreria Node.js per la gestione di notifiche push real-time, basata su Websocket e dunque una comunicazione bidirezionale tra client e server invece che su un'architettura a polling.
- **Bcrypt** libreria che implementa l'omonimo algoritmo di hashing considerato lo stato dell'arte per la cifratura delle password. Viene utilizzato all'interno dell'authentication-microservice per la registrazione e autenticazione degli utenti della dashboard.

Capitolo 5

Codice

Di seguito vengono riportare i dettagli implementativi che si ritengono importanti o peculiari.

5.1 Gestione autenticazione utenti

Il listato 5.1 riporta la funzione di login implementata sull'authentication-microservice nel momento in cui un nuovo utente esegue il login sulla dashboard.

```
1 exports.login = (req, res) => {  
2   User.findOne({ username: req.body.username }, (err,user) => {  
3     if (!user) {  
4       res.status(401).json({  
5         message: 'Login not successful',  
6         error: 'Email or password are wrong',  
7       });  
8     } else {  
9       bcrypt.compare(req.body.password, user.password)  
10        .then(result => {  
11         result ? res.status(200).json({  
12           message: 'Login successful',  
13           user,  
14         }) : res.status(401).json({  
15           message: 'Login not successful',  
16           error: 'Email or password are wrong' });  
17       });  
18     }  
19   });  
20 }
```

Listing 5.1: funzione di login per l'autenticazione di utenti registrati

5.2 Gestione socket

Al fine di realizzare un meccanismo di comunicazione real-time per inviare notifiche push ai cittadini, si è deciso di utilizzare **Socket.io**. Dato che le notifiche

devono essere private per ciascun utente, è stato necessario implementare un meccanismo di verifica delle socket che aprono una connessione con il server. Per questo motivo, quando si apre una connessione si verifica che sia stato inviato o il token relativo ad una sessione precedente autenticata o il proprio identificativo per creare una nuova sessione, se mancano entrambe le informazioni la connessione viene rifiutata.

```
1 io.use((socket, next) => {
2   const sessionId = socket.handshake.auth.sessionId;
3   if (sessionId) {
4     const session = sessionStore.findSession(sessionId);
5     if (session) {
6       socket.sessionId = sessionId;
7       socket.userId = session.userId;
8       return next();
9     }
10  }
11  const userId = socket.handshake.auth.userId;
12  if (!userId) {
13    return next(new Error("invalid userId"));
14  }
15  socket.sessionId = randomId();
16  socket.userId = userId;
17  next();
18 });
```

Listing 5.2: socket.io middleware function per verificare le richieste di connessione

5.3 Esempio Pinia Store

Come descritto nel precedente capitolo, si è utilizzato Pinia al fine di poter gestire uno stato globale e quindi di poter accedere più agevolmente da ciascun componente ai dati necessari. Si riporta di seguito un esempio di *Store* Pinia.

```
1 export const useUserStore = defineStore('user', {
2   state: () => ({
3     firstname: '',
4     lastname: '',
5     email: '',
6     userId: '',
7     role: '',
8   }),
9   getters: {
10    getUserName: (state) => {
11      return state.firstname + ' ' + state.lastname;
12    },
13
14    isLoggedIn: (state) => {
15      return state.userId !== '';
16    },
17
18    isCitizen: (state) => {
19      return state.role === 'CITIZEN';
20    }
19  });
```



```

20   },
21
22   isManager: (state) => {
23     return state.role === 'MANAGER';
24   }
25 },
26 actions:{
27   login(user) {
28     this.firstname=user.firstname;
29     this.lastname=user.lastname;
30     this.email=user.username;
31     this.userId=user._id;
32     this.role=user.role;
33   },
34
35   logout() {
36     this.firstname='';
37     this.lastname='';
38     this.email='';
39     this.userId='';
40     this.role='';
41   }
42 },
43 persist:true
44 });

```

Listing 5.3: Esempio di Store definito utilizzando la libreria Pinia

5.4 Modelli MongoDB

La scelta di utilizzare MongoDB come strumento di persistenza ha determinato la necessità di modellare le entità da memorizzare secondo il modello documentale. Nello specifico, la libreria Mongoose richiede di definire dei *Model* per ciascuna delle collezioni trattate nel server Node.js implementato. Di seguito si riportano quindi i principali modelli definiti nel *booking-microservice* e nel *authentication-microservice*.

```

1  const BookingSchema = new Schema({
2    userId: {
3      type: String,
4      required: true
5    },
6    typeOfWaste: {
7      wasteName:{
8        type: String,
9        enum: ["TWIGS", "WASTE OIL", "IRON",
10              "ELECTRONICS", "CLOTHES", "OTHER"]
11      }
12    },
13    datetime: {
14      type: Date,
15      default: Date.now
16    },
17    city: String,

```

```

18 province: String,
19 address: String,
20 status: {
21   type: String,
22   enum: ["PENDING", "ASSIGNED", "COMPLETED"],
23   default: "PENDING"
24 },
25 });

```

Listing 5.4: Schema per le prenotazioni

```

1 const NotificationSchema = new Schema({
2   userId: {
3     type: String,
4     required: true
5   },
6   booking: {
7     type: mongoose.Schema.Types.ObjectId,
8     ref: "Booking",
9     required: true
10  },
11  isRead: {
12    type: Boolean,
13    default: false
14  }
15 });

```

Listing 5.5: Schema per le notifiche

```

1 const UserSchema = new Schema({
2   firstname: {
3     type: String,
4     required: true,
5   },
6   lastname: {
7     type: String,
8     required: true,
9   },
10  username: {
11    type: String,
12    unique: true,
13    required: true,
14  },
15  password: {
16    type: String,
17    required: true,
18  },
19  role: {
20    type: String,
21    default: 'CITIZEN',
22    enum: ['CITIZEN', 'MANAGER'],
23    required: true,
24  }
25 });

```

Listing 5.6: Schema per gli utenti

Capitolo 6

Test

6.1 Test del codice

Il *booking-microservice*, il server Express implementato per gestire tutte le operazioni riguardanti le prenotazioni di raccolta a domicilio, è stato testato utilizzando il framework javascript di testing **Mocha** e la libreria per le asserzioni **Chai**. In questo modo è stato possibile verificare che ciascuna delle rotte definite restituisca le informazioni desiderate in modo corretto e che in caso di richieste formattate in modo errato risponda con errori chiari e funzionali a risolvere il problema.

6.2 Test di usabilità

Durante tutto il processo di progettazione e sviluppo delle interfacce della *dashboard* si è avuta particolare attenzione a mantenere un buon livello di usabilità del sistema. Nello specifico, sono stati utilizzati due metodi per valutarla:

- **Valutazione tramite euristiche:** in questo caso, per valutare l'usabilità del sistema, si è fatto riferimento alle euristiche di Nielsen. Ad esempio, si è voluto mantenere un **design e un'estetica minimalista**, le interfacce sono semplici ed essenziali in modo che l'utente non sia confuso o distratto da elementi puramente decorativi e irrilevanti. Inoltre, si è cercato di semplificare e rendere uniforme l'esperienza di navigazione degli utenti, mantenendo **consistenza e standard** nelle varie interfacce, è infatti sempre ben visibile la *navbar* per muoversi all'interno dell'applicazione e tutte le icone utilizzate mantengono sempre lo stesso significato, in modo tale che sia ben chiara la relativa funzionalità. Grazie all'analisi del dominio, svolta seguendo il *Domain Driven Design*, all'interno del sistema si utilizzano sempre termini chiari e familiari ai *target user*, avendo quindi **corrispondenza tra sistema e mondo reale**. Infine, si è cercato di dare all'utente **controllo e libertà**, ad esempio è possibile consultare la

mappa con i vari punti di raccolta anche senza effettuare il login, oppure per i manager sono presenti diverse modalità (e scorciatoie) per accedere ai dettagli di una missione.

- **Test di usabilità con utenti:** al termine dello sviluppo di ciascuna delle interfacce, è stato chiesto a utenti con diversa conoscenza del dominio applicativo di provare ad eseguire alcuni task, in modo tale da verificare come potenziali cittadini e manager reali avrebbero interagito con l'applicativo e quindi ricevere dei feedback su come perfezionare e semplificare l'interazione uomo-macchina. Ad esempio, da una delle sessioni di *usability test* si è capito come l'iniziale scelta grafica utilizzata per fornire le informazioni riguardo gli step di una missione non fosse chiara come previsto e dunque, accogliendo i consigli ricevuti dagli utenti è stato possibile modificare l'interfaccia al fine di renderla più intuitiva.

Capitolo 7

Deployment

Data la complessa architettura del sistema (3.1) e il fatto che sono presenti 6 microservizi oltre all'applicazione web, è stato necessario utilizzare **Docker** per semplificare e rendere più veloce il deployment dell'intero sistema. Nello specifico, si sono *containerizzati* i singoli componenti dell'architettura e si è generato un *docker-compose* per istanziarli tutti e generare la rete che consente di farli comunicare.

Grazie alle conoscenze acquisite nel corso di Laboratorio di Sistemi Software, l'intero sistema è mantenuto all'interno di un'organizzazione GitHub in cui si possono trovare i repository per ciascun componente, visitabile qui. Inoltre, per ciascun servizio sono stati definiti dei *workflows* per avere *Continuous Integration* e *Continuous Delivery*. In particolare, grazie a quest'ultimi, l'immagine Docker di ciascun elemento, e dunque anche dell'applicazione web e dei due microservizi Node.js da me sviluppati, viene mantenuta aggiornata, in modo automatico, rispetto alle modifiche apportate. Per questo motivo, la consegna di questo progetto è avvenuta fornendo il link di un repository composto da 4 *submodules* che fanno riferimento alle componenti da me interamente sviluppate:

- **booking-microservice:** repository contenente tutto il codice prodotto per implementare il server Node.js che espone le api REST per gestire le prenotazioni.
- **authentication-microservice:** repository contenente tutto il codice prodotto per implementare il server Node.js che espone le api REST per gestire la registrazione e l'autenticazione degli utenti.
- **dashboard:** repository contenente tutto il codice prodotto per implementare il front-end Vue.js dell'intero sistema.
- **smart-waste-collection-system:** repository contenente il file *docker-compose* per istanziare il sistema. Tale file viene mantenuto aggiornato rispetto alle nuove versioni rilasciate dei vari componenti grazie al BOT renovate.

Capitolo 8

Conclusioni

Il presente progetto è stata un'importante occasione per consolidare le abilità e le nozioni apprese durante il corso. Inoltre, mi ha permesso di cimentarmi nell'utilizzo di nuove tecnologie, come ad esempio *Pinia* e di comprendere e sperimentare ancora di più concetti trattati in altri corsi. Ad esempio, ha rafforzato i concetti relativi alla programmazione asincrona trattata in Programmazione Concorrente e Distribuita.

Inoltre, ritengo che l'integrazione del progetto con i corsi di Pervasive Computing e Laboratorio di Sistemi Software abbia generato un progetto di maggior valore didattico.

In conclusione, mi ritengo soddisfatta del lavoro svolto, del risultato ottenuto e delle conoscenze e capacità acquisite, sicuramente spendibili anche in altri futuri progetti e nel mondo del lavoro.