

Recycle

Applicazioni e Servizi Web

Andrea Rettaroli - 0000977930 {andrea.rettaroli@studio.unibo.it}

27 Marzo 2023

Indice

1	Introduzione	5
2	Requisiti	6
2.1	Personas	6
2.1.1	Letizia	6
2.1.2	Marco	6
2.1.3	Luca	7
2.1.4	Mario	7
2.1.5	Francesco	7
2.2	Definizione dei requisiti	7
2.2.1	Requisiti funzionali	8
2.2.2	Requisiti non funzionali	9
3	Design	10
3.1	Architettura del sistema	10
3.2	Architettura del server	10
3.2.1	Funzionalità del server	10
3.2.2	Utilizzo di promises	11
3.3	Struttura del database	11
3.4	Architettura del client	12
3.4.1	Pagine	12
3.4.2	Componenti	12
3.4.3	stato dell'applicazione	13
3.4.4	Comunicazioni via Socket	14
3.5	Schema dell'architettura	15
3.6	Metodologia di Sviluppo	15
3.7	Experience Prototyping	15
3.8	Interfaccia utente	16
3.8.1	Mockup interfaccia utente	16
4	Technologie	18
4.1	Tecnologie comuni	18
4.1.1	Node Package Manager	18
4.1.2	Socket.io	19

4.2	Tecnologie Frontend	19
4.2.1	React	20
4.2.2	LocalStorage e SessionStorage	20
4.2.3	Hook and Custom Hook	21
4.2.4	React router v6	21
4.2.5	Vite	22
4.2.6	Axios	22
4.2.7	Tailwindcss	22
4.2.8	Redux	23
4.2.9	Recharts	23
4.2.10	React-hook-form	23
4.2.11	il8n-next	24
4.3	Tecnologie Backend	24
4.3.1	Node.js	24
4.3.2	Express	25
4.3.3	Mongoose	25
4.3.4	MongoDB	25
4.3.5	Mongo-Express	25
4.3.6	Bcryptjs	25
4.3.7	Librerie di test	26
4.3.8	Swagger	26
4.3.9	jsonwebtoken	26
4.4	Linguaggi	26
4.4.1	TypeScript	26
5	Codice	28
5.1	Codice Server	28
5.1.1	Middleware	28
5.2	Socket	29
5.3	Codice Client	30
5.3.1	useUserSession custom hook	30
5.3.2	Axios	32
5.3.3	Redux Store	32
5.3.4	Router	33
6	Test	35
6.0.1	Test-Driven Development	35
6.0.2	Acceptance Test-Driven Development	35
6.0.3	Behavior-Driven Development	36
6.1	Test Backend	36
6.1.1	Swagger e Postman	36
6.2	Test Frontend	37
6.2.1	Redux Developer Tools	37
6.2.2	Axe Accessibility	37
6.2.3	Usability Test	37

7	Deployment	39
7.1	Deploy manuale	39
7.2	Deploy con Docker	40
7.3	Vercel	41
8	Conclusioni	43
8.0.1	Sviluppi futuri	43
8.0.2	Conclusioni	43

Elenco delle figure

3.1	Schema dell'architettura dell'applicazione	15
6.1	Risultato dei test	36

Capitolo 1

Introduzione

L'idea di Recycle nasce per aiutare le persone nella raccolta differenziata premiandole per la loro attenzione al riciclo dei rifiuti.

Recycle è uno strumento di monitoraggio e gestione dei rifiuti domestici che grazie all'aiuto di alcuni sensori posizionati sui cestini dei rifiuti rendiconta tutte le operazioni che vengono fatte in tempo reale. La piattaforma permette la creazione di una propria area cestini caratterizzati da dimensione e tipo di rifiuto. Allo stesso tempo l'applicazione fornisce una ricca dashboard che riassume quanto stai contribuendo positivamente all'ambiente e al tuo portafogli grazie al riciclo dei tuoi rifiuti.

Recycle è quindi uno strumento alla portata di tutti, facile e intuitivo.

Capitolo 2

Requisiti

2.1 Personas

Al fine di delineare nello specifico i requisiti, si è deciso di creare delle Personas che siano rappresentative del target che si vuole raggiungere con l'applicativo.

2.1.1 Letizia

- Nome: Letizia;
- Età: 35 anni;
- Lavoro: segretaria;
- Contesto familiare: sposata con un figlio;
- Abitudini: per motivi lavorativi, Letizia si trova spesso a dover gestire i rifiuti della sua casa.

2.1.2 Marco

- Nome: Marco;
- Età: 17 anni;
- Lavoro: studente a tempo pieno;
- Contesto familiare: single;
- Abitudini: Essendo giovane Marco si trova a fare le faccende di casa e buttare i rifiuti.

2.1.3 Luca

- Nome: Luca;
- Età: 12 anni;
- Lavoro: studente a tempo pieno;
- Contesto familiare: single;
- Abitudini: frequentemente in questa età si inizia ad essere più autonomi e Luca deve imparare a differenziare i rifiuti.

2.1.4 Mario

- Nome: Mario;
- Età: 64 anni;
- Lavoro: insegnante;
- Contesto familiare: Sposato con tre figli;
- Abitudini: Vuole educare i figli al rispetto dell'ambiente, inoltre si occupa personalmente di portar fuori i cestini della spazzatura .

2.1.5 Francesco

- Nome: Francesco;
- Età: 55 anni;
- Lavoro: Proprietario di un Hotel;
- Contesto lavorativo: deve gestire molti rifiuti di vario genere e vuole farlo nel rispetto dell'ambiente, inoltre vuole limitare gli sprechi e monitorare ciò che accade nella sua struttura;

2.2 Definizione dei requisiti

Le Personas hanno permesso di delineare in modo chiaro il tipo di utenti che potrebbero utilizzare il sistema sviluppato, permettendo un'individuazione più chiara dei requisiti.

I requisiti sono stati suddivisi in funzionali e non funzionali.

2.2.1 Requisiti funzionali

I requisiti funzionali descrivono le funzionalità e i servizi che il sistema deve fornire all'utente. Tipicamente vanno a delineare il comportamento del sistema a fronte di determinati input.

I requisiti funzionali che il sistema deve mettere a disposizione sono i seguenti:

- Permettere di registrare un account;
- Permettere di accedere nel proprio account;
- Permettere di disconnettersi dal proprio account;
- Permettere di aggiornare le informazioni di base dell'utente;
- Permettere di creare un nuovo cestino;
- Permettere di creare massimo un cestino per tipo di rifiuto;
- Permettere di decidere la dimensione e il tipo di rifiuto in fase di creazione del cestino;
- Permettere di aggiornare la dimensione del cestino;
- Permettere di eliminare un cestino;
- Permettere di registrare gli eventi provenienti dai sensori dei cestini;
- Permettere l'aggiornamento in tempo reale dei dati relativi ai cestini con gli eventi provenienti dai sensori;
- Permettere gli eventi di inserimento di un rifiuto in un cestino;
- Permettere gli eventi di rimozione di un rifiuto in un cestino;
- Permettere gli eventi di ritiro dei rifiuti in un cestino;
- Notificare in qualche maniera gli eventi sopra citati;
- Permettere la visualizzazione delle informazioni relative ad un cestino, compreso un piccolo storico degli eventi raccolti dai suoi sensori;
- Permettere a un utente di aver accesso alle informazioni relative alle quantità di materiali riciclati in base alla tipologia di rifiuto.
- Permettere a un utente di aver accesso alle informazioni relative alle acquisizioni dei sensori in base alla tipologia di rifiuto.
- Permettere ad un utente di comprendere quanto ha guadagnato grazie al corretto riciclo in base alla tipologia di rifiuto.
- Permettere a un utente con privilegio *Admin* di inserire i prezzi dei rifiuti distinti per tipologia;

- Permettere a un utente con privilegio *Admin* di modificare i prezzi dei rifiuti distinti per tipologia;
- Permettere a un utente con privilegio *Admin* di eliminare i prezzi dei rifiuti distinti per tipologia;

Come si può notare dai requisiti funzionali elencati, ci sono due diversi ruoli di utenti: **user**, **admin**. Dove l'utente è la parte attiva e predominante nelle funzionalità, l'amministratore per ora si occupa solo di definire i prezzi dei rifiuti in base alla loro categoria. Si precisa che al ruolo di amministratore potrebbero essere aggiunte molti altri requisiti.

2.2.2 Requisiti non funzionali

I requisiti non funzionali riguardano le caratteristiche e le proprietà che il sistema deve avere e in questo caso la web application deve:

- Essere facile da utilizzare anche ad utenti non esperti;
- Fornire un'interfaccia utente responsive con focus principali su Desktop e mobile;
- Essere accessibile;
- Essere intuitiva;
- Essere sicura mediante meccanismi di autenticazione;
- Essere estendibile a nuove funzionalità;

Capitolo 3

Design

Nel seguente capitolo si andrà a descrivere l'architettura del sistema sviluppato.

3.1 Architettura del sistema

Applicazione è stata costruita come stack MERN, acronimo di:

- **Mongo DB**
- **Express**
- **React**
- **Node.js**

MERN è una variante dello stack MEAN (dove lato frontend il framework Angular viene sostituito con React) che ha permesso l'uso di Javascript e JSON in tutta l'architettura, divisa in backend e frontend.

3.2 Architettura del server

Il server, seguendo la logica dello stack MERN, è basato su Node.js e si interfaccia a un database MongoDB per memorizzare i dati persistenti dell'applicazione.

3.2.1 Funzionalità del server

Il server si suddivide nelle seguenti componenti:

- **Controllers:** dove vengono definite le varie funzionalità che il server svolge;
- **Models:** dove vengono definiti gli schemi degli elementi delle collezioni di MongoDB;
- **Routes:** dove vengono definiti punti di accesso alle funzionalità.

- **Middleware:** dove vengono definiti comportamenti intermediari come l'autenticazioni e la socket per interazione con i sensori.

Nello specifico le funzionalità che il server mette a disposizione sono le seguenti:

- **Autenticazione:** registrazione, accesso e verifica dell'identità dell'utente attraverso un sistema di token;
- **Gestione degli eventi:** il server mette a disposizione anche il servizio di WebSocket attraverso il quale vengono catturati e gestiti gli eventi provenienti dai sensori;
- **CRUD:** creare, leggere, modificare e eliminare tutti gli elementi delle varie collezioni definite nel database.

Ognuna di queste funzionalità costituisce un "modulo" composto da più file al cui interno vengono definite e implementate le routes, i controllers e il model necessarie al server per poter eseguire le operazioni una volta contattato dal client.

3.2.2 Utilizzo di promises

Le promises, caratteristiche dell'implementazione event-driven asincrona che si è scelta per l'applicazione, sono gestite attraverso i costrutti di ES6 **async-await** (e non attraverso il costrutto *then()* visto a lezione) per evitare il Continuation-Passing Style, con numerose chiamate sequenziali il codice avrebbe molti livelli di nesting e il codice risulta meno leggibile e di conseguenza mantenibile. Inoltre il costrutto *await* non è bloccante nell'esecuzione dell'event-loop di Node.js.

3.3 Struttura del database

Utilizzando lo stack MERN, il database è di tipo document-based implementato con MongoDB, ed è costituito dalle seguenti collezioni di documenti:

- **Users:** ogni documento di questa collezione contiene i dati appartenenti esclusivamente ad un utente specifico ovvero l'anagrafica di base, la lingua di preferenza, il ruolo;
- **Baskets:** contiene i documenti relativi ai cestini dei rifiuti, ogni cestino è caratterizzato dall'id dell'utente a cui appartiene, il tipo che ne caratterizza il rifiuto che accoglie, la dimensione e il riempimento.
- **Acquisitions:** contiene i documenti relativi alle acquisizioni dei sensori sui cestini. Le acquisizioni sono caratterizzate dall'id dell'utente, dall'id del cestino, dal nome del rifiuto, dal tipo del rifiuto e dal peso del rifiuto.
- **Withdrawals:** contiene i documenti relativi al ritiro dei rifiuti. I ritiri sono caratterizzati dall'id dell'utente, dall'id del cestino, dal tipo del cestino, dal peso del contenuto del cestino e dal rispettivo valore.

- **Prices:** contengono i documenti relativi al valore dei tipi di rifiuto, si caratterizza quindi da un tipo e da un valore.

3.4 Architettura del client

Il client è stato sviluppato utilizzando il framework React; è possibile discutere della sua architettura concentrandosi sugli aspetti principali: Pagine, componenti, stato dell'applicazione.

3.4.1 Pagine

Le pagine sono dei macroelementi che compongono l'applicazione, vengono gestite dalla libreria di routing esterna react router v6. Le pagine delineate per questa applicazione sono le seguenti:

- **Login:** permette l'autenticazione dell'utente tramite credenziali che gli permette l'accesso all'applicazione;
- **Signup:** ermette di registrarsi all'applicazione;
- **Home:** è la pagina principale dell'applicazione dove viene mostrato lo status dei cestini dell'utente.
- **Statistics:** è la pagina che contiene i grafici e le metriche relative all'utente.
- **Profile:** è la pagina che contiene i dati dell'utente e ne permette l'aggiornamento, consente il logout.
- **BasketDetails:** consente di accedere ai dettagli specifici di un cestino e di modificarne alcune caratteristiche o di eliminarlo.
- **AddBasket:** consente di aggiungere un cestino.
- **NotFound:** E' una pagina che cattura tutte le rotte sbagliate e permette il reindirizzamento alla home o al login.

Le pagine sono un insieme di funzionalità e componenti.

3.4.2 Componenti

I componenti vanno a costituire l'interfaccia grafica dell'applicazione. Possono essere visti come l'unità base dell'applicazione. Si tende a rendere componente tutto ciò che si voglia sia riusabile nell'applicazione o che rappresenti un elemento o un concetto. I componenti come le pagine sono file di tipo JSX o TSX.

JavaScript XML o TypeScript XML sono un'estensione delle sintassi e vengono utilizzate per descrivere l'aspetto che dovrebbe avere la UI. React riconosce il fatto che la logica di renderizzazione è per sua stessa natura accoppiata con le

altre logiche che governano la UI: la gestione degli eventi, il cambiamento dello stato nel tempo, la preparazione dei dati per la visualizzazione.

Invece di separare artificialmente le tecnologie inserendo il codice di markup e la logica in file separati, React separa le responsabilità utilizzando unità debolmente accoppiate chiamate “componenti” che contengono entrambi.

3.4.3 stato dell'applicazione

Alcuni dati fondamentali dell'applicazione che devono essere raggiungibili da tutti i componenti sono stati incapsulati all'interno di Redux.

Nello specifico sono presenti i seguenti dati:

- **User store:** contiene le informazioni dell'utente che ha effettuato l'accesso:
 - **id:** l'id dell'utente;
 - **name:** il nome dell'utente;
 - **surname** il cognome dell'utente;
 - **email:** l'indirizzo email dell'utente;
 - **address:** l'indirizzo di residenza dell'utente;
 - **province:** la provincia di residenza dell'utente;
 - **language:** la lingua di preferenza dell'utente;
 - **role:** il ruolo, utente o admin;
 - **token:** il token fornito dal server durante l'autenticazione;
 - **createdAt:** la data in cui l'utente ha creato il suo account;
 - **updatedAt:** la data in cui l'utente è stato aggiornato;
- **Baskets store:** contiene le informazioni relative ai cestini di un utente:
 - **Baskets:** rappresenta una lista di cestini, ogni cestino è composto dai seguenti campi: id, userId, type, dimension, filling, createdAt, updatedAt;
 - **fetchData:** è un flag booleano che indica se i dati sono stati ricevuti.
- **Error store:** contiene le informazioni a un possibile errore:
 - **errorMessage:** rappresenta la stringa da far visualizzare all'utente in caso di errore.
 - **isOnErrorState:** è un flag booleano che indica se siamo in uno stato di errore.

Altri stati vengono gestiti internamente ai componenti utilizzando l'hook di React useState.

3.4.4 Comunicazioni via Socket

La webSocket è stata utilizzata per simulare il comportamento dei sensori. La socket funge da connettore tra i tre elementi architetturali: sensori, server, client.

Il flusso dei messaggi parte dai sensori che inviano messaggi di tre tipi:

- **put_trash**: è il messaggio che identifica l'inserimento nel cestino di un nuovo rifiuto.

```
{
  "userId": "6408b8f57c1434784ced7a62",
  "basketId": "64137e6d023af5573dcec92a",
  "wasteName": "bar",
  "wasteType": "METALS",
  "wasteWeight": 0.2
}
```

- **remove_trash**: è il messaggio che identifica la rimozione dal cestino di un rifiuto.

```
{
  "userId": "6408b8f57c1434784ced7a62",
  "acquisitionId": "640764eb7be6b9805386231d"
}
```

- **clear_trash**: è il messaggio che indica che il cestino è stato svuotato dall'operatore di raccolta.

```
{
  "userId": "6408b8f57c1434784ced7a62",
  "basketId": "6415c6da56f984cbca91d912"
}
```

Il server riceve questi messaggi e si occupa di gestirli e di trasmettere al client gli stessi messaggi. I client sono in ascolto solo dei loro messaggi questo grazie all'identificativo dell'utente. Il server ritrasmette il messaggio con le seguenti strutture:

- put_trash:userId
- remove_trash:userId
- clear_trash:userId

Il server assume così la funzionalità di broker. Architettturalmente si ispira al comportamento in ambito IoT secondo il protocollo MQTT(Message Queuing Telemetry Transport).

3.5 Schema dell'architettura

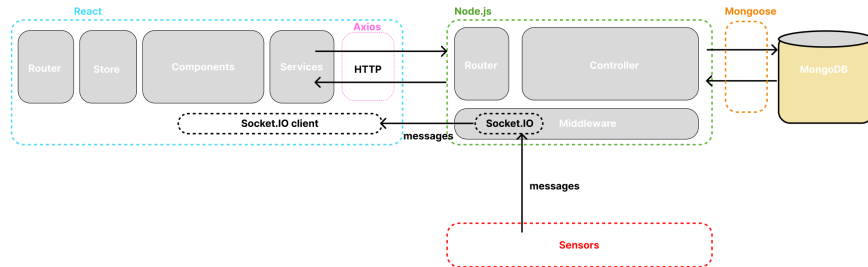


Figura 3.1: Schema dell'architettura dell'applicazione

3.6 Metodologia di Sviluppo

Il modello utilizzato è di tipo iterativo e basato su User Centered Design con utenti inizialmente virtualizzati. La scelta è ricaduta su questa metodologia in quanto sin dalle prime fasi di progettazione, si è messo in primo piano la HCI (Human Computer Interaction), cercando di focalizzarsi sulle esigenze degli utenti e sull'ottenimento di una buona usabilità. Sono stati individuati gli utenti target dell'applicazione e su di essi si sono sviluppate le Personas che hanno contribuito alla comprensione delle funzionalità del sistema e come possano esse rispondere ai requisiti utente. Si è gestito tutto il progetto tramite una dashboard Jira che ha permesso di organizzare lo sviluppo in vari sprint; in ognuno dei quali sono state individuate le funzionalità da implementare. Il team ha seguito quindi un framework di sviluppo AGILE. Nello svolgimento del progetto e del design delle interfacce si è sempre cercato di rispettare i seguenti principi:

- Responsive Design: sono state realizzate pagine Web in grado di adattarsi graficamente e in modo automatico ai dispositivi coi quali vengono;
- KISS: le interfacce sono state realizzate per contenere solamente gli elementi fondamentali ed in modo facilmente accessibile all'utente.

3.7 Experience Prototyping

Una volta creati i draft dei mockup in Figma si è costruita una storyboard con essi al fine di costruire un Experience Prototyping dell'applicazione e di passare da un approccio User Centered Design con Personas virtuali ad un approccio più partecipativo e inclusivo. Questo passaggio ha permesso di rendere reali le Personas ipotizzate e di ricevere feedback concreti per migliorare la User

Interface e la User Experience. In questa fase è stato chiesto di svolgere task all'interno della storyboard, come:

- Registrarsi;
- Effettuare il Login;
- Aggiungere un Basket;
- Visualizzare le statistiche;
- Modificare il profilo;
- Fare il logout;

3.8 Interfaccia utente

3.8.1 Mockup interfaccia utente

Durante la fase di progettazione sono stati creati dei mockup dell'interfaccia utente, che sono stati revisionati dopo i feedback degli utenti, ed in seguito sono stati utilizzati come linee guida durante l'effettivo sviluppo del client.

Login

Le schermate di Login sono molto classiche ed entrambe presentano il logo dell'applicazione e i gli input in una card molto semplice e pulita.

Signup

Le schermate di Signup seguono lo stile di quelle di Login e presentano il logo dell'applicazione e i gli input in una card molto semplice e pulita.

Home

Le schermate di Home cercano di seguire al massimo il principio di Kiss e vogliono abbracciare a pieno la prima e la ottava Euristiche di Nielsen ovvero: Visibilità dello stato del sistema e Design e estetica minimalista. Infatti risulta molto semplice comprendere lo status dei cestini e capire come aggiungerli o compiere operazioni su di essi.

Statistics

Le schermate di Statistics cercano di fornire una dashboard semplice e intuitiva.

Profile

Le schermate di Profile cercano riprendono lo stile delle form realizzate per il Login e per il Signup.

In generale, sia nel mobile che nel Desktop si è cercato di dare risalto al menù di navigazione delle schede principali (home, statistics, profile), lasciandolo sempre in evidenza e in una posizione comoda per entrambi i formati. Nel mobile si è preferito piazzarlo in basso in modo da ricordare lo stile del **Bottom Tabs Navigator** spesso utilizzato nelle applicazioni mobile.

Capitolo 4

Technologie

All'interno del progetto, essendo sviluppato con stack MERN, sono state utilizzate diverse tecnologie, che differiscono da frontend a backend. MERN ha offerto tutto il necessario per lo sviluppo di un'applicazione cross-platform, indipendente al sistema operativo utilizzato.

4.1 Tecnologie comuni

Alcune delle tecnologie sono state utilizzate sia nello sviluppo del client che nello sviluppo del server. Verranno pertanto presentate in questa sezione dedicata, al fine di evitare ridondanza.

4.1.1 Node Package Manager

Node.js è una runtime di JavaScript multiplatforma per l'esecuzione di codice JavaScript, costruita sul motore JavaScript V8 di Google Chrome.

Node.js dispone di una grande quantità di moduli scritti completamente in JavaScript. Essendo il progetto open source è inoltre possibile per gli sviluppatori aggiungere i propri moduli in modo da renderli disponibili pubblicamente.

Il gestore di pacchetti predefinito per l'ambiente di runtime JavaScript Node.js si chiama Node Package Manager (npm).

Il comando base per ottenere un pacchetto è:

```
npm install [PACKET_NAME]
```

Tutte le dipendenze e i conflitti vengono gestiti automaticamente.

Per avviare il sistema, utilizzando le dipendenze, è sufficiente utilizzare il seguente comando:

```
npm start
```

4.1.2 Socket.io

Socket.io è una libreria JavaScript che offre delle API JavaScript cross-browser che permettono la creazione di un canale di comunicazione full-duplex tra domini a bassa latenza tra il browser e il server web.

Socket.io tenta di utilizzare prima di tutto le WebSocket native. In caso fallisca (in caso di incompatibilità coi sistemi o a causa di altri problemi) ricorrerà all'utilizzo di polling HTTP.

Oltre alle funzionalità offerte da una tradizionale WebSocket, Socket.io offre inoltre le seguenti feature:

- reliability (switch a polling HTTP nel caso la connessione WebSocket non possa essere stabilita);
- riconnessione automatica;
- buffering dei pacchetti;
- acknowledgments;
- broadcast a tutti i client o ad un sottoinsieme di essi;
- multiplexing.

Grazie alle proprietà offerte da questa libreria, è stato possibile implementare il sistema che simula gli eventi dei sensori sui cestini.

Inoltre Socket.io è stata utilizzata lato server per registrare gli eventi relativi a tutti i cestini e ritrasmetterli agli specifici utenti che dopo il login sono in ascolto grazie a Socket.IO-Client degli eventi al fine di mostrare nella UI lo status dei cestini in tempo reale.

4.2 Tecnologie Frontend

Per la parte frontend sono state utilizzate:

- **React**: framework che rispetta il pattern MVVM;
- **Socket.IO-Client**: permette l'uso di WebSocket lato client;
- **Hooks, LocalStorage e SessionStorage**: per mantenere lo stato a livello locale;
- **Custom Hook**: per l'accesso automatico;
- **React router v6**: per il routing lato frontend;
- **Vite**: come framework agnostic tool per la gestione dei moduli;
- **Axios**: libreria per le richieste HTTP;

- **Tailwindcss**: framework CSS per la gestione degli stili;
- **Redux**: libreria per la gestione dello stato a livello di applicazione;
- **Recharts**: libreria per i grafici;
- **React-hook-form**: libreria per le form;
- **i18n-next**: libreria per il multi-lingua.

4.2.1 React

React è un framework open-source che permette di implementare applicazioni web seguendo i principi della programmazione ad oggetti.

Comprende 3 concetti principali:

- **JSX**: è un'estensione di JavaScript che permette di richiamare un componente React tramite un meccanismo analogo ai tag i html;
- **Componenti**: il componente è una classe contenente uno stato che verrà mantenuto ed eventualmente aggiornato durante il ciclo di vita del componente. Un componente padre può passare dati ai componenti figli mediante l'uso di props. Nella creazione del client sono stati utilizzati principalmente componenti funzione, i quali forniscono le stesse funzionalità dei componenti classe, ma con una sintassi più concisa;
- **Stato**: è un insieme di proprietà di un componente; alla loro modifica, React si occupa di aggiornare l'interfaccia grafica, rendendola quindi reattiva.

4.2.2 LocalStorage e SessionStorage

LocalStorage e SessionStorage sono due opzioni di archiviazione web utilizzate per memorizzare i dati sul browser del client. Sia SessionStorage che LocalStorage consentono di archiviare i dati lato client, il che significa che i dati vengono memorizzati sul browser dell'utente anziché su un server. La principale differenza tra SessionStorage e LocalStorage è quanto a lungo i dati persistono. I dati di SessionStorage sono disponibili solo per la durata della sessione dell'utente, il che significa che i dati vengono persi quando l'utente chiude il browser o passa a un altro sito web. D'altra parte, i dati di LocalStorage persistono anche dopo che l'utente chiude il browser e sono disponibili per l'utilizzo la prossima volta che l'utente visita il sito web.

Anche se potrebbe sembrare che LocalStorage sia la scelta migliore perché consente ai dati di persistere nel tempo, ci sono alcune ragioni per cui potresti scegliere di utilizzare SessionStorage tra cui la sicurezza. Ciò perché i dati vengono eliminati non appena l'utente chiude il browser, quindi c'è meno rischio di esporre dati sensibili se il computer dell'utente viene compromesso.

La scelta del SessionStorage è stata fatta anche per un discorso sperimentale avendo io spesso usato il LocalStorage.

4.2.3 Hook and Custom Hook

Gli Hooks di React sono una caratteristica introdotta nella versione 16.8 di React che consente di utilizzare lo stato e altre funzionalità di React all'interno dei componenti funzionali. Ciò significa che i componenti funzionali possono ora avere accesso allo stato interno e ad altre funzionalità precedentemente disponibili solo nei componenti di classe.

Gli Hooks predefiniti di React includono `useState`, `useEffect`, `useContext` e molti altri. Questi Hooks predefiniti consentono di gestire lo stato, di effettuare chiamate API e di accedere ai contesti di React in modo più semplice e pulito rispetto alla gestione dello stato all'interno dei componenti classe.

Inoltre, esiste anche la possibilità di creare dei Custom Hooks in React. Un Custom Hook è semplicemente una funzione JavaScript che utilizza una o più degli Hooks predefiniti di React. Questo consente di creare funzionalità personalizzate utilizzando le stesse API che sono disponibili all'interno degli Hooks predefiniti.

I Custom Hooks possono essere utilizzati per organizzare il codice in modo più pulito e leggibile, nonché per riutilizzare la logica di stato e di effetto tra diversi componenti. Inoltre, possono essere utilizzati per incapsulare la logica di un componente e renderla più modulare e facilmente testabile.

In generale, le Hooks di React e le Custom Hooks rappresentano un modo più semplice e pulito per gestire lo stato e altre funzionalità, rendendo più facile lo sviluppo di applicazioni web complesse.

In questa applicazione è stato utilizzato l'hook `useUserSession` per determinare dalla sessione se l'utente avesse effettuato il login o no.

4.2.4 React router v6

React Router v6 è l'ultima versione della popolare libreria di React per la gestione delle rotte all'interno delle applicazioni web non inclusa all'interno del framework stesso. La principale novità di questa versione è la semplificazione dell'utilizzo della libreria grazie all'eliminazione di alcuni concetti precedenti, come il concetto di `Switch` e di `Route component`.

Invece, React Router v6 utilizza un sistema di routing basato su funzioni. Questo significa che le rotte sono definite come funzioni che restituiscono elementi JSX, rendendo più facile la definizione delle rotte in modo programmatico.

Inoltre, React Router v6 supporta la navigazione gerarchica, consentendo di definire rotte nidificate in modo più semplice rispetto alle versioni precedenti. Ciò consente di gestire facilmente rotte complesse, come quelle con parametri dinamici.

Infine, React Router v6 supporta la gestione degli errori, consentendo di definire rotte per gestire eventuali errori nella navigazione, come ad esempio la navigazione a una pagina non esistente.

4.2.5 Vite

Vite.js è un nuovo strumento di sviluppo web veloce e leggero che mira a migliorare l'esperienza di sviluppo per i progetti basati su Vue.js e React. Vite.js utilizza una tecnica di elaborazione dei moduli a tempo di compilazione, consentendo di eseguire la compilazione in modo veloce e leggero.

Inoltre, Vite.js utilizza il concetto di server di sviluppo a moduli singoli, che consente di avviare rapidamente un server di sviluppo e di testare le modifiche senza dover ricompilare l'intera applicazione.

Vite.js supporta anche l'hot-reloading, consentendo di visualizzare immediatamente le modifiche apportate al codice senza dover ricaricare l'intera pagina.

Inoltre, Vite.js offre un'esperienza di sviluppo efficiente grazie alla gestione automatica delle dipendenze, consentendo di utilizzare solo le dipendenze richieste per il progetto.

Infine, Vite.js supporta la generazione di un pacchetto di produzione ottimizzato, garantendo un tempo di caricamento veloce e un'esperienza utente ottimale.

4.2.6 Axios

Axios è una popolare libreria JavaScript per la gestione delle richieste HTTP, che può essere utilizzata sia lato client che lato server. È basata sulle Promise e supporta diverse piattaforme, tra cui browser e Node.js.

Axios fornisce una sintassi semplice ed elegante per l'invio di richieste HTTP, consentendo di definire facilmente gli header, i parametri e il corpo della richiesta.

Inoltre, Axios gestisce automaticamente la trasformazione dei dati in formato JSON, consentendo di gestire facilmente la risposta della richiesta.

Axios supporta anche l'intercettazione delle richieste, consentendo di aggiungere intercettatori globali o locali per l'elaborazione di eventuali errori o modifiche alle richieste.

Inoltre, Axios supporta la cancellazione delle richieste, consentendo di annullare una richiesta in corso in caso di necessità.

Rappresenta una libreria essenziale per la gestione delle richieste HTTP in JavaScript, offrendo un'interfaccia semplice e intuitiva per la comunicazione tra client e server.

4.2.7 Tailwindcss

Tailwind CSS è una libreria CSS utility-first che permette di creare rapidamente interfacce utente personalizzate. È basata sul concetto di utility classi, che permette di definire rapidamente e facilmente gli stili CSS di elementi specifici. Inoltre, Tailwind CSS offre una vasta gamma di classi predefinite per la gestione delle dimensioni, dei margini, dei padding, dei colori e di altri aspetti dell'interfaccia utente.

Tailwind CSS supporta anche la personalizzazione delle classi predefinite, consentendo di definire facilmente le proprie classi utilizzando variabili CSS.

Inoltre, Tailwind CSS integra facilmente con altri framework e librerie, come ad esempio React, Vue.js e Angular.

Tailwind CSS è altamente performante grazie al fatto che utilizza una tecnica di purging, che consente di eliminare le classi CSS non utilizzate nel pacchetto finale.

Per queste ragioni è diventato molto popolare superando Bootstrap a livello di download come è possibile vedere da **npm trends**.

4.2.8 Redux

Redux è un contenitore di stato per applicazioni JavaScript e possiede alcune caratteristiche fondamentali: è deterministico, centralizzato, debug oriented e flessibile.

Redux è costituito da:

- **store**: lo store contiene la struttura ad albero dello stato e fornisce metodi per la lettura dello stato corrente;
- **reducer**: i reducer definiscono la struttura dello store;
- **actions**: le actions permettono di modificare la struttura dello store.

Il motivo per cui Redux è fondamentale nello sviluppo del client di questo progetto è dato dal modo in cui React gestisce lo stato interno. React permette di passare informazioni tra componenti solo dall'alto verso il basso, quindi da un componente padre a un componente figlio; non è raro che le informazioni ottenute da un componente figlio debbano andare a modificare in qualche modo il padre e per fare questo Redux è indispensabile.

L'uso di slice consente di definire facilmente i reducers, le actions e i selectors in un solo file, semplificando l'organizzazione e la gestione del codice.

4.2.9 Recharts

Recharts è una libreria JavaScript per la visualizzazione di dati in grafici interattivi. Si basa sulla libreria di visualizzazione dei dati D3.js, semplificando l'utilizzo di grafici complessi. Offre una vasta gamma di grafici predefiniti, tra cui grafici a linea, a barre, a torta, a dispersione e a radar, con una vasta scelta di opzioni di visualizzazione per personalizzare facilmente i grafici tramite l'utilizzo di opzioni di configurazione e di stili CSS.

4.2.10 React-hook-form

React Hook Form è una libreria di validazione dei form leggera e performante per React. Si basa sul concetto di hook di React, semplificando la gestione dei form in modo intuitivo. Offre un'ampia gamma di funzionalità, tra cui la validazione delle form, la gestione degli errori e la gestione dello stato delle

form. React Hook Form è altamente performante grazie alla sua architettura leggera e alla sua semplicità di utilizzo; supporta la validazione dei campi di input tramite regole di validazione predefinite o personalizzate. Inoltre, React Hook Form offre anche funzionalità avanzate come la validazione asincrona, la gestione della formattazione dei dati e la gestione della riproduzione automatica degli errori. E' dotata di una vasta gamma di opzioni di personalizzazione e di configurazione, offrendo la flessibilità necessaria per la creazione di form complessi e personalizzati.

4.2.11 i18n-next

i18n-next è una libreria di internazionalizzazione leggera e altamente configurabile per React. Si basa sul concetto di hook di React, semplificando la gestione delle traduzioni in modo intuitivo. E' dotata di un'ampia gamma di funzionalità, tra cui la traduzione di testo, la gestione delle lingue e la gestione della localizzazione.

4.3 Tecnologie Backend

- **Node.js**: come runtime JavaScript;
- **Express**: come framework;
- **MongoDB**: come database;
- **Mongoose**: libreria per l'interazione con il database;
- **Mongo-Express**: come interfaccia sul web per avere un supporto nel controllare visivamente la struttura del database;
- **Bcryptjs**: libreria per l'hashing delle password;
- **Socket.IO**: libreria per la WebSocket;
- **Mocha - Chai - Chai HTTP - SuperTest**: librerie per i test lato server.
- **Swagger**: libreria di utilità per esporre API documentate.
- **jsonwebtoken**: libreria per la gestione dei token.

4.3.1 Node.js

Node.js è un ambiente di esecuzione di JavaScript lato server basato sul motore JavaScript V8 di Google. È diventato molto popolare per lo sviluppo di applicazioni web scalabili e performanti. Node.js utilizza un modello di I/O asincrono non bloccante, che consente di gestire facilmente un grande numero di richieste contemporaneamente senza influire sulle performance; è altamente

personalizzabile e modulare grazie alla vasta gamma di moduli npm (Node Package Manager) disponibili, semplificando lo sviluppo di applicazioni. Inoltre, è compatibile con il protocollo HTTP, permettendo di creare facilmente server web e servizi web.

4.3.2 Express

Express è un framework Javascript che fornisce una serie di funzionalità avanzate compatibili con Node.js.

In questo progetto è stato usato principalmente per conseguire una gestione più agile del routing e per definire e concatenare middleware.

4.3.3 Mongoose

Questa libreria fornisce un interfacciamento diretto e schema-based al database MongoDB, gestendo il type-casting, la validazione e la costruzione di query. La manipolazione del database avviene tramite oggetti Javascript denominati "Schema", che espongono i metodi di interrogazione al database sottostante. È stata usata lato server per creare, interrogare e modificare le collezioni del database, permettendo query compatte e leggibili tramite le funzioni `findById` e `findByIdAndUpdate/Delete`.

4.3.4 MongoDB

MongoDB è un database non-relazionale basato sui documenti che permette di modellare oggetti JSON complessi e innestati. La flessibilità garantita da questo approccio ha permesso, oltre che di avere strutture dati che rispecchiavano fedelmente le entità del dominio applicativo prese in esame, anche di implementare facilmente operazioni che con database relazionali sarebbero state più difficoltose e prolisse.

4.3.5 Mongo-Express

Mongo-Express è un'interfaccia web open-source per la gestione dei database MongoDB. Permette di visualizzare, modificare e creare documenti all'interno del database in modo facile e intuitivo.

4.3.6 Bcryptjs

Bcryptjs è una libreria JavaScript open-source per la crittografia delle password. Utilizza l'algoritmo di hash Bcrypt per creare hash sicuri delle password. Bcryptjs è facilmente integrabile in applicazioni JavaScript e Node.js, consentendo di proteggere le password degli utenti in modo efficace e sicuro.

4.3.7 Librerie di test

Mocha è un framework di testing JavaScript per Node.js e browser, che offre funzionalità avanzate come il supporto per test asincroni.

Chai è una libreria di asserzioni per Node.js e browser, che fornisce un'interfaccia elegante e intuitiva per testare le aspettative del codice.

Chai HTTP è un plugin per Chai che fornisce un'interfaccia semplificata per effettuare richieste HTTP durante i test.

SuperTest è una libreria Node.js che utilizza Chai HTTP per effettuare richieste HTTP simulando l'interazione con un server reale.

Insieme, questi strumenti forniscono un'esperienza di testing completa e intuitiva per le applicazioni web, semplificando la gestione di test unitari e di integrazione, prenderemo meglio questi aspetti nella sezione relativa ai Test.

4.3.8 Swagger

Swagger è una suite di strumenti open-source per la progettazione, la creazione e la documentazione di API RESTful. Offre un'interfaccia intuitiva per la creazione di API, semplificando il processo di sviluppo. Permette di generare automaticamente la documentazione delle API in diversi formati, come JSON e HTML, facilitando la comprensione e l'utilizzo delle API da parte degli sviluppatori. Offre numerosi strumenti per testare le API e monitorarne le prestazioni in tempo reale, semplificando la risoluzione di problemi e la manutenzione delle API.

4.3.9 jsonwebtoken

jsonwebtoken è una libreria JavaScript per la creazione e la verifica di token di autenticazione JSON Web Token (JWT). La libreria offre una crittografia sicura per proteggere i dati sensibili dell'utente e fornisce funzionalità avanzate come gestione del token.

4.4 Linguaggi

I linguaggi utilizzati all'interno del progetto sono:

- TypeScript: lato client;
- JavaScript: lato server;

4.4.1 TypeScript

TypeScript è un superset del linguaggio JavaScript, open-source sviluppato da Microsoft che aggiungendo il supporto per il typing statico. Il typing statico fornisce un maggiore livello di controllo sui tipi di dati utilizzati all'interno del codice, aiutando a prevenire errori comuni durante la fase di sviluppo.

TypeScript è compatibile con la maggior parte delle librerie e dei framework JavaScript esistenti, tra cui React, Node.js e Angular, rendendolo una scelta popolare per lo sviluppo di applicazioni web. Fornisce anche funzionalità avanzate come l'interfaccia e la classe, la possibilità di definire tipi personalizzati e la gestione delle eccezioni.

Capitolo 5

Codice

5.1 Codice Server

In generale il codice presente lato server è stato suddiviso in Models, Controllers e Routes. Nel file index.js avviene la connessione con database MongoDB grazie a Mongoose, vengono definite le routes e avviata la WebSocket. La parte server consiste principalmente in semplici operazioni create-read-update-delete CRUD, di rilievo sono i middleware definiti in seguito per la gestione dell'autenticazione e della socket.

5.1.1 Middleware

Grazie a Express è stato possibile utilizzare i middleware, automatizzando la stragrande maggioranza dei controlli dell'input lato server, mantenendo al minimo la ridondanza del codice e incrementando di conseguenza la sua robustezza e la sua manutenibilità.

Auth

Questo middleware viene eseguito prima di ogni operazione effettuabile da un utente per verificarne l'effettiva identità tramite un token di identificazione criptato creato e convalidato dalla libreria jsonwebtoken.

```
const jwt = require("jsonwebtoken");

const verifyToken = (req, res, next) => {
  const authHeader = req.headers["authorization"];
  var token = authHeader && authHeader.split(" ")[1]; //split Bearer from token

  if (!token) {
    return res.status(401).send("A token is required for authentication");
  }
}
```

```

    try {
      const decoded = jwt.verify(token, process.env.TOKEN_KEY, { maxAge: "8h" });
      req.user = decoded;
    } catch (err) {
      console.error(" ~ file: auth.js:21 ~ verifyToken ~ err:", err);
      return res.status(401).send("Invalid Token");
    }
    return next();
  };

module.exports = verifyToken;

```

5.2 Socket

Questo middleware serve alla gestione dei messaggi in arrivo dai sensori e utilizza socket.io che viene inizializzato all'interno di index.js .

```

const Utils = require("../utils/functions");

const PUT_TRASH = "put_trash";
const REMOVE_TRASH = "remove_trash";
const CLEAR_TRASH = "clear_trash";

/**
 * This function catch the events for put trash into a user trash basket
 * example of data = {
 *   "userId": "64023aac471e5c26eccd26bd",
 *   "basketId": "6407403eb6505fe776812ffb",
 *   "wasteName": "Bottle",
 *   "wasteType": "PLASTIC",
 *   "wasteWeight": 0.2
 * }
 * also will produce createdAt and an id
 */
const onPut = (socket, client) => async (data) => {
  console.log(" ~ file: socket.js:36 ~ onPut ~ data:", data);
  await Utils.putAcquisition(data);
  socket.emit(PUT_TRASH + ":" + data.userId, data);
};

/**
 * This function catch the events for remove trash in a user trash basket
 * example of data = {
 *   acquisitionId: "33023aac471e5c26eccd26bd",
 *   userId: "64023aac471e5c26eccd26bd",
 * }
 */

```

```

const onRemove = (socket, client) => async (data) => {
  await Utils.removeAcquisition(data);
  socket.emit(REMOVE_TRASH + ":" + data.userId, data);
};
/**
 * This function catch the events for claer a user trash basket
 * example of data = {
 *   userId: "64023aac471e5c26eccd26bd",
 *   basketId: "14023aac471e5c26eccd26sd",
 * }
 */
const onClear = (socket, client) => async (data) => {
  await Utils.garbageCollection(data);
  socket.emit(CLEAR_TRASH + ":" + data.userId, data);
};
// This function is responsible for the websocket event registration on all lock commands
exports.socketHandler = (socket) => {
  socket.on("connection", (client) => {
    client.on(PUT_TRASH, onPut(socket, client));
    client.on(REMOVE_TRASH, onRemove(socket, client));
    client.on(CLEAR_TRASH, onClear(socket, client));
  });
  return socket;
};

```

5.3 Codice Client

In generale il codice presente lato client non presenta grandi peculiarità. Si è prestata molta attenzione alla riusabilità dei componenti proprio per questo si sono definiti dei componenti di UI. Nelle sezione seguenti vengono riportati alcuni frammenti di codice degni di nota.

5.3.1 useUserSession custom hook

Questo è un esempio di un hook custom creato per la gestione della sessione utente.

```

import { useState, useEffect } from "react";
import { useNavigate } from "react-router-dom";

interface userData {
  userId: string;
  token: string
}

```

```

interface UserSession {
  isLoggedIn: boolean;
  loggedUser: userData
  login: (token: string, userId: string) => void
  logout: () => void
}

const useUserSession = (): UserSession => {
  const [isLoggedIn, setIsLoggedIn] = useState(false);
  const [loggedUser, setLoggedUser] = useState({ token: "", userId: "" });
  const navigate = useNavigate()

  useEffect(() => {
    // Check if user session exists in sessionStorage
    const userSession = sessionStorage.getItem("userSession");
    console.log(" ~ file: useUserSession.ts:20 ~ useEffect ~ userSession:", userSession)

    if (userSession) {
      const { token, userId } = JSON.parse(userSession);
      setLoggedUser({ token, userId });
      setIsLoggedIn(true);
    }
  }, []);

  const login = (token: string, userId: string) => {
    // Store user session in sessionStorage
    //localStorage.setItem("auth", response.data.token);
    sessionStorage.setItem("userSession", JSON.stringify({ token, userId }));
    setLoggedUser({ token, userId });
    setIsLoggedIn(true);
  };

  const logout = () => {
    // Remove user session from sessionStorage
    sessionStorage.removeItem("userSession");
    setLoggedUser({ token: "", userId: "" });
    setIsLoggedIn(false);
    navigate("/login")
  };

  return {
    isLoggedIn,
    loggedUser,
    login,
    logout,
  }
}

```



```

    };
  };

  export default useUserSession;

```

5.3.2 Axios

Il seguente codice mostra come Axios riesca a configurare il token dinamicamente durante le richieste.

```

import axios from "axios";

const DEV_HOST_BACKEND = "http://localhost:3000";
const PROD_HOST_BACKEND = "https://recycle-project-api.vercel.app";

const Api = axios.create({
  baseURL: DEV_HOST_BACKEND
});

Api.interceptors.request.use(async (config) => {
  const userSession = sessionStorage.getItem("userSession");
  if (userSession) {
    const { token } = JSON.parse(userSession);
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
  }
  return config;
});

export default Api;

```

5.3.3 Redux Store

In seguito vengono riportati dei frammenti di codice relativi a uno degli store Redux utilizzati nell'applicazione dove viene mostrata la configurazione dello store relativo all'utente con le azioni per modificare l'utente nello stato Redux e per rimuovere l'utente dallo stato Redux riportandolo allo stato iniziale.

```

interface UserState {
  user: User | null;
}

const initialState: UserState = {
  user: null,

```

```

};

export const userSlice = createSlice({
  name: 'user',
  initialState,
  reducers: {
    setUser: (state, action: PayloadAction<User>) => {
      console.log(" ~ file: user.slice.ts:35 ~ action.payload:", action.payload)
      state.user = action.payload;
    },
    clearUser: (state) => {
      state.user = null;
    },
  },
});

export const { setUser, clearUser } = userSlice.actions;

export default userSlice.reducer;

```

5.3.4 Router

Di seguito viene riportato come è stato fatto per creare delle route accessibili solo agli utenti che hanno correttamente effettuato l'accesso. Il primo frammento di codice mostra come sia stato definito il componente PrivateRoutes; mentre il secondo frammento mostra come sia stato utilizzato il componente PrivateRoutes nella definizione delle Routes del Router.

```

import { Navigate, Outlet } from "react-router-dom";

const PrivateRoutes = () => {
  const userSession = sessionStorage.getItem("userSession");
  if (userSession) {
    const { token } = JSON.parse(userSession);
    if (token) {
      return <Outlet />;
    }
  }

  return <Navigate to="/login" />;
};

export default PrivateRoutes;

```

```

const Router: React.FC = (): ReactElement => {
  return (
    <Routes>
      <Route element={<PrivateRoutes />}>
        <Route path="/" element={<Home />} />
        <Route path="/statistics" element={<Statistics />} />
        <Route path="/profile" element={<Profile />} />
        <Route path="/add-basket" element={<AddBasket />} />
        <Route path="/basket-details/:basketId" element={<BasketDetails />} />
      </Route>
      <Route path="/login" element={<Login />} />
      <Route path="/signup" element={<Signup />} />
      <Route path="*" element={<NotFound />} />
    </Routes>
  );
};

```

Capitolo 6

Test

I test sono una parte molto importante durante lo sviluppo di un'applicazione web. Infatti esistono delle vere e proprie metodologie di sviluppo che partono proprio da un approccio ai test, queste sono: TDD, ATDD e BDD. Questi approcci possono essere applicati sia allo sviluppo frontend che allo sviluppo backend.

6.0.1 Test-Driven Development

Il Test-Driven Development (TDD) è una pratica di sviluppo del software che si concentra sulla scrittura di test prima di scrivere il codice. In questo modo, si garantisce che il codice sia sempre testato e che ogni modifica non rompa le funzionalità esistenti. Il processo TDD prevede tre fasi: scrivere un test, farlo fallire e infine scrivere il codice per farlo passare. Questo ciclo viene ripetuto per ogni nuova funzionalità o modifica del codice. Il TDD aiuta a migliorare la qualità del codice e a ridurre il tempo necessario per individuare e risolvere i bug.

6.0.2 Acceptance Test-Driven Development

L'Acceptance Test-Driven Development (ATDD) è una pratica di sviluppo del software che si concentra sulla definizione di test di accettazione prima di scrivere il codice. Questi test vengono sviluppati in collaborazione tra i vari stakeholder (sviluppatori, utenti, analisti) e servono a verificare che il software soddisfi i requisiti funzionali e non funzionali. Il processo ATDD prevede tre fasi: definire i requisiti, sviluppare i test di accettazione e infine scrivere il codice per soddisfare i requisiti e far passare i test. L'ATDD aiuta a garantire che il software sia sviluppato con la massima qualità possibile e che soddisfi le esigenze degli utenti.

6.0.3 Behavior-Driven Development

Il Behavior-Driven Development (BDD) è una pratica di sviluppo del software che si concentra sulla definizione di comportamenti del software in modo chiaro e comprensibile a tutti gli stakeholder. Il BDD prevede l'utilizzo di una notazione di storie utente (user stories) e scenari per descrivere i comportamenti attesi. Questi scenari vengono poi trasformati in test automatizzati e utilizzati come strumenti di verifica continua durante lo sviluppo. Il BDD aiuta a migliorare la comunicazione tra i vari membri del team e a garantire che il software sviluppato soddisfi i requisiti funzionali e non funzionali.

6.1 Test Backend

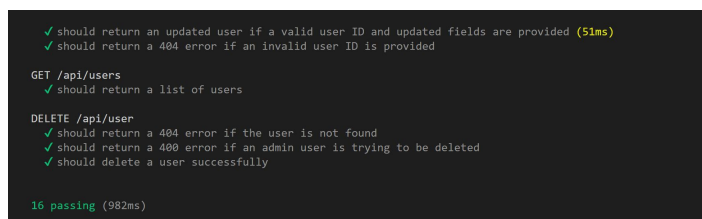
Per ragioni di tempo si è deciso di non sviluppare i test per tutte le parti in gioco nel backend ma di adottare l'approccio Test-Driven Development nello sviluppo della parte di gestione dell'utente. Nel file `user.test.js` vengono infatti definiti i test per le funzionalità di Login, Signup, Get User, Update User, Delete User, User List. I test sono stati sviluppati utilizzando le seguenti librerie di test:

- Mocha
- Chai
- Chai HTTP
- SuperTest

Per lanciare i test è sufficiente utilizzare il seguente comando nella cartella server.

```
npm run test
```

L'immagine seguente mostra il risultato.



```
✓ should return an updated user if a valid user ID and updated fields are provided (51ms)
✓ should return a 404 error if an invalid user ID is provided

GET /api/users
✓ should return a list of users

DELETE /api/user
✓ should return a 404 error if the user is not found
✓ should return a 400 error if an admin user is trying to be deleted
✓ should delete a user successfully

16 passing (982ms)
```

Figura 6.1: Risultato dei test

6.1.1 Swagger e Postman

Lo Swagger Postman sono strumenti molto comodi quando si vogliono testare API REST.

Nello specifico è stato utilizzato principalmente Postman che ha permesso di

testare anche la WebSocket. Anche le API esposte dal server sono state testate tramite Postman prima di essere utilizzate dal client, di modo da verificare il comportamento, sia in caso di successo che in caso di errore. Questo ha permesso di lavorare con API testate, evitando di dover andare ad individuare a posteriori se la risposta di errore fosse causata dal server o dal client.

6.2 Test Frontend

Per ragioni di tempo lato frontend non sono stati fatti test funzionali. Per lo sviluppo si è utilizzato Redux Developer Tools e i console log come strumenti di supporto, debug e verifica di correttezza. Si è preferito dare rilievo ai testi di accessibilità e usabilità.

6.2.1 Redux Developer Tools

Molti browser, tra cui Chrome e Firefox (quelli principalmente utilizzati durante lo sviluppo), mettono a disposizione un'estensione che permette di visualizzare lo stato di Redux in tempo reale. Inoltre Redux Developer Tools mostra anche tutto lo storico dei cambiamenti dello stato, di modo da facilitarne il debug.

6.2.2 Axe Accessibility

Axe Accessibility è un tool che permette tramite un'estensione installabile direttamente su browser di ispezionare una pagina web, al fine di individuare problemi di accessibilità. Il tool categorizza i problemi riscontrati come critical, serious, moderate e minor, a seconda della gravità. Suggerisce inoltre delle linee guida per risolvere i problemi.

L'uso del tool ha permesso di identificare problemi relativi ad esempio al poco contrasto dei colori in alcuni casi ma soprattutto che la libreria Recharts utilizzata per i test non inserisce i campi alt nelle immagini svg che genera per rappresentare i grafici, purtroppo la libreria non fornisce la possibilità di inserire la proprietà alt alle immagini svg dei grafici manualmente per risolvere il problema.

In questo modo si ha la certezza che l'accessibilità del client sia ottimale.

6.2.3 Usability Test

Gli Usability test vengono effettuati con utenti reali, per verificare che l'usabilità del sistema sviluppato sia soddisfacente.

Per fare questo, viene proposto all'utente l'applicazione, con una minima introduzione su che tipo di funzionalità aspettarsi. A quel punto si osserva l'utente interagire con il sistema, verificando in quali punti trova difficoltà.

Ciò è stato svolto con diversi utenti e dal sondaggio posto loro a seguito del test è emerso che tutti gli utenti hanno trovato particolarmente semplice svolgere le seguenti operazioni:

- Quanto è stato facile o difficile registrarsi al sito?
- Quanto è stato facile o difficile accedere al sito?
- Quanto è stato facile o difficile aggiungere un cestino?
- Quanto è stato facile o difficile comprendere lo status del cestino?
- Quanto è stato facile o difficile modificare un cestino?
- Quanto è stato facile o difficile eliminare un cestino?
- Quanto è stato facile o difficile comprendere le statistiche?
- Quanto è stato facile o difficile aggiornare il tuo profilo?
- Quanto è stato facile o difficile uscire dal tuo account?

Grazie alla metodologia di sviluppo scelta per il design delle interfacce il questionario ha avuto esiti molto positivi; infatti, tutte le domande la risposta media è stata tra il facile e il molto facile (che era il massimo della scala).

Capitolo 7

Deployment

Il deploy del sistema può avvenire nei seguenti modi:

- deploy manualmente;
- deploy con Docker;
- utilizzando tools di deploy;

Nelle seguenti sezioni si andranno a discutere queste le metodologie.

7.1 Deploy manuale

Al fine di avere la possibilità di effettuare con successo il deploy manuale, è necessario assicurarsi di installare i seguenti software:

- **npm**: indispensabile per installare le dipendenze;
- **node**: necessario per il funzionamento del client;
- **mongodb**: permette di avere a disposizione il database.

Successivamente è necessario creare un file `.env` nella root del server. In questo file devono essere dichiarate le variabili d'ambiente che il sistema utilizza:

```
TOKEN_KEY=[token_for_encryption]
DB_CONNECTION_STRING=[database_connection_string]
PORT=[server_port_number]
```

Per il client è sufficiente modificare gli endpoint di Axios e della WebSocket. Una volta che il set-up dell'ambiente è stato completato, si può passare ad eseguire il client e il server.

Per eseguire il **server**, per prima cosa devono essere installate le dipendenze con il comando


```
npm install
```

e poi si può avviare il server con

```
node index.js
```

Anche per eseguire il client è necessario installare le dipendenze con il comando

```
npm install
```

e l'effettiva esecuzione si ottiene con il comando

```
npm start
```

7.2 Deploy con Docker

L'utilizzo di Docker ha permesso di gestire container che eseguono processi in ambienti isolati. La creazione di un container Docker si ottiene andando a specificare un **Dockerfile** per ogni servizio.

Nel sistema sono stati infatti creati due Dockerfile, uno per il client e uno per il server. Inoltre entrambi i servizi dispongono di un **.dockerignore**, che permette di specificare che cosa non includere nel container creato.

I Dockerfile create non fanno altro che andare a scaricare un'immagine di Docker già esistente che al suo interno abbia installato node; da qui, sia nel caso del client che nel caso del server, vengono scaricate le dipendenze necessarie e vengono lanciati i comandi per eseguire i servizi.

Il deployment di più sottosistemi può essere effettuato utilizzando lo strumento **Compose** di Docker, il quale permette di eseguire un'applicazione Docker multi-container.

Nel caso del sistema sviluppato il file **docker-compose.yaml** specifica quali servizi sono presenti e i relativi Dockerfile. Sono presenti 4 servizi: il server, il client e il database mongodb un interfaccia al database creata con mongo-express.

In questa casistica, per eseguire il sistema non è necessario installare nessun altro software a parte Docker.

Tutte le variabili d'ambiente necessarie sono state inserite nel docker-compose, di modo che non sia nemmeno necessaria una fase di set-up ulteriore.

Per avviare l'esecuzione è necessario creare le immagini di client e server con i seguenti comandi:

```
cd client
docker build -t recycle .
cd server
docker build -t recycle-server .
```

Successivamente per lanciare la nostra applicazione è possibile utilizzare il comando:

```
docker-compose up
```

Per terminare l'esecuzione invece si deve utilizzare il comando:

```
docker-compose down
```

inserire dei dati di default si può comodamente utilizzare la dashboard di mongo-express esposta alla porta 8082 dove è possibile importare le collezioni.

7.3 Vercel

Vercel è una piattaforma di sviluppo web che consente ai team di creare, distribuire e gestire siti web e applicazioni con facilità. Creata nel 2015, Vercel è diventata una delle piattaforme di hosting web più popolari al mondo grazie alla sua facilità d'uso, alle prestazioni elevate e alla scalabilità. Con Vercel, si possono creare applicazioni e siti web utilizzando qualsiasi linguaggio di programmazione, framework o strumento. La piattaforma supporta inoltre l'integrazione con i principali strumenti di sviluppo, come Git e GitHub, per semplificare il flusso di lavoro. Una delle caratteristiche più interessanti di Vercel è la sua capacità di distribuire le applicazioni in modo globale attraverso una rete di server distribuiti in tutto il mondo. Ciò significa che i siti web e le applicazioni sono sempre veloci e reattivi, indipendentemente dalla posizione dell'utente. Inoltre, offre una vasta gamma di funzionalità avanzate, come il supporto per il rendering del lato server, la gestione della cache e la scalabilità automatica. Queste funzionalità consentono ai team di gestire applicazioni web di qualsiasi dimensione e complessità.

Per fare il deploy di un progetto è necessario seguire i seguenti step:

- Registrarsi a Vercel;
- Creare un nuovo Progetto;
- Fornire i permessi al repository GitHub per l'iterazione con Vercel;
- Compilare la voce Build & Development Settings con il tipo di framework che si sta utilizzando, nel nostro caso per il frontend selezionare ViteJs per il backend selezionare Others;
- Configurare il file di env come descritto prima per il deploy manuale;
- Avviare il deploy;

Per il backend va inoltre aggiunto un file di configurazione chiamato Vercel.json . Purtroppo con il tipo di account che detengo (quello gratuito) non sono riuscito a far funzionare la WebSocket.

Per completare il deploy è necessario il database MongoDB. MongoDB Atlas permette di creare un database MongoDB in uno spazio cloud; Per farlo è necessario:

- Registrarsi a MongoDB Atlas;
- Creare un Progetto;
- Configurare il cluster;
- Creare una coppia di credenziali di cui dovrete salvarvi la password;
- Una volta configurato il cluster vi viene fornita la connection string da utilizzare per connettersi al database, potete testare la connessione con MongoDB Compass;
- Potete aggiungere i dati sia dopo esservi connessi con MongoDB Compass che dal portale web di MongoDB Atlas;
- configurare correttamente la connessione sul server;

Capitolo 8

Conclusioni

8.0.1 Sviluppi futuri

Al fine di migliorare e ampliare il progetto si delineano i seguenti sviluppi futuri:

- lo sviluppo di un interfaccia per gli utenti di tipo amministratore al fine di poter visionare altre metriche e maneggiare agilmente i prezzi;
- creazione della funzionalità per il recupero della password di un utente;
- creazione di processi di CI/CD per automatizzare test e deploy.

8.0.2 Conclusioni

Grazie allo sforzo ingegneristico posto in essere dai requisiti minimi dell'applicazione mi ritengo molto soddisfatto del lavoro svolto. E' stata una challenge che mi ha permesso di sperimentare nuove librerie e funzionalità e a porre una maggiore attenzione agli aspetti legati all'Human Computer Interaction, all'accessibilità e all'usabilità. Il risultato finale rispecchia in pieno la visione originale dell'applicazione proposta come elaborato, e non si esclude né una sua possibile espansione a livello di funzionalità.