Recycle

Applicazioni e Servizi Web

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

27 Marzo 2023

Indice

Elenco delle figure

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 requisisti.
I requisisti 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: deve 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 fornisce anche il servizio di web-Socket attraverso il quale vengono catturati e gestisit 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 alla 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**: continene 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 accede 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;
  - **fetchedData**: 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, clinet.

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. Architetturalmente si ispira al comportamento in ambito IoT secondo il protocollo MQTT(Message Queuing Telemetry Transport).

3.5 Schema dell'architettura

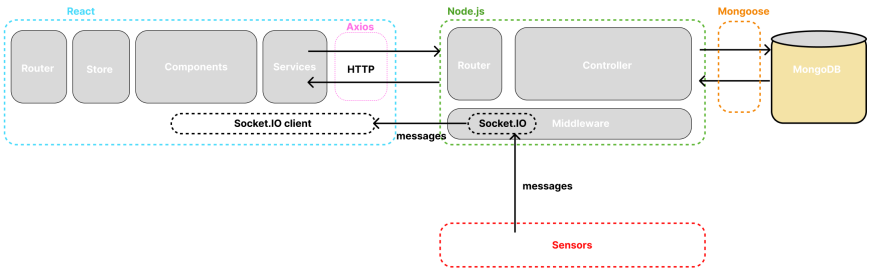

Figura 3.1: Schema dell'architettura dell'applicazione

3.6 Interfaccia utente

3.6.1 Mockup interfaccia utente

Durante la fase di progettazione sono stati creati dei mockup dell'interfaccia utente, che in seguito sono stati utilizzati come linee guidata durante l'effettivo sviluppo del client.