BringMeFood

Applicazioni e Servizi Web

Luca Bracchi - 0000974076 {luca.bracchi3@studio.unibo.it}

Fabio Muratori - 0000984550 {fabio.muratori2@studio.unibo.it}

18 Gennaio 2021

Introduzione

L'obiettivo del progetto è quello di creare un'applicazione per il supporto alle attività del Banco Alimentare di Cesena: tramite questa app gli utenti saranno in grado di donare cibo, mettendosi prima in contatto con i volontari. I volontari potranno organizzare le attività di raccolta, preparazione, consegna e gestione del magazzino, oltre a poter sfruttare il sistema per organizzare e promuovere eventi di sensibilizzazione e raccolta. Tramite l'applicazione sarà inoltre possibile segnalare le famiglie bisognose alle quali sarà destinato il cibo donato.

Contents

Chapter 1

Requisiti

In questo capitolo verranno descritte le caratteristiche e le funzionalità che il sistema prevede

1.1 Requisiti utente

L'utente standard può:

- Donare alimenti al banco alimentare
- Segnalare le famiglie bisognose

L'utente volontario può:

- Creare giri di raccolta delle donazioni
- Aggiungere al magazzino gli alimenti donati
- Creare pacchi alimentare
- Creare giri di consegna dei pacchi alimentari
- Segnalare le famiglie bisognose
- Creare eventi di sensibilizzazione e raccolta cibo
- Indicare l'avvenuta verifica delle segnalazioni di famiglie bisognose

L'ente pubblico (utente trusted), oltre a poter fare ciò che può fare l'utente volontario; può:

- Eleggere a volontari gli utenti standard
- Accettare o rifiutare le richieste di famiglie bisognose sottomesse da utenti o volontari

1.2 Requisiti funzionali

Il sistema prodotto deve permettere di:

- Aggiungere e visualizzare le inserzioni per le donazioni di cibo
- Modificare e cancellare le donazioni non ancora prese in carico
- Creare giri di raccolta delle donazioni
- Inserire e visualizzare gli alimenti nel magazzino
- Rimuovere gli alimenti dal magazzino
- Aggiungere unità agli alimenti del magazzino
- Segnalare e visualizzare le famiglie bisognose
- Modificare e cancellare le segnalazioni delle famiglie non ancora verificate
- Indicare l'avvenuta verifica delle segnalazioni delle famiglie bisognose
- Creare e visualizzare pacchi alimentari associati alle famiglie bisognose
- Eliminare i pacchi alimentari non ancora inseriti in un giro di consegna
- Creare giri di consegna dei pacchi alimentari alle famiglie bisognose
- Creare eventi di sensibilizzazione e raccolta cibo
- Visualizzare la lista dei propri eventi
- Modificare ed eliminare gli eventi
- Visualizzare una serie di dati aggregati che diano un'idea sull'utilizzo del sistema, in particolare:
  - Informazioni sulle tipologie di alimenti donati
  - Numero di famiglie bisognose segnalate e verificate
  - Numero di pacchi consegnati
  - Numero di alimenti donati
  - Prossimi eventi programmati, ordinati dal più al meno prossimo
- Registrarsi inserendo le proprie informazioni personali
- Accedere tramite la coppia ¡email,password¿
- Effettuare il logout
- Navigare l'area personale solo se l'utente ha effettuato il login
- Modificare le informazioni personali dell'utente

1.3 Requisiti non funzionali

Il sistema prodotto deve:

- l'interfaccia utente dovrà essere il più possibile user-friendly ma anche permettere un utilizzo agile delle funzionalità del sistema;
- l'applicazione dovrà avere un buon grado di accessibilità;
- ogni errore generato nell'utilizzo dell'applicazione dovrà essere opportunamente mostrato (o con apposite schermate o messaggi a scomparsa);
- l'applicazione dovrà poter essere utilizzata sia da dispositivi mobile che desktop;
- le funzionalità offerte agli utenti dovranno essere accessibili solo se in presenza degli opportuni privilegi.

1.4 Requisiti di implementazione

Per quel che riguarda i requisiti di implementazione, l'applicazione:

- userà lo stack MEVN:
  - sarà sviluppata in Vue.js lato frontend
  - sarà implementata in Node.js lato backend
  - userà Express.js per la parte di API
  - userà un DBMS NoSQL MongoDB
- farà uso di TypeScript sia lato frontend che lato backend
- farà uso di Socket.IO per implementare la chat tra utenti

Chapter 2

Design

2.1 Backend

Il server è strutturato come una serie di router, ognuna relativa ad un'API associata ad una delle collezioni mantenute nel database. Ogni router è associato a un controller che contiene le funzioni che effettuano le operazioni richieste tramite la request.

Ad ogni documento relativo alle varie collezioni è associata un'interfaccia, che ne definisce gli attributi in maniera compatta e di più semplice lettura, e un interfaccia che definisce informazioni aggiuntive su obbligatorietà e validazione degli attributi.

2.1.1 Database

La figura 2.1 illustra il modello del database MongoDB.

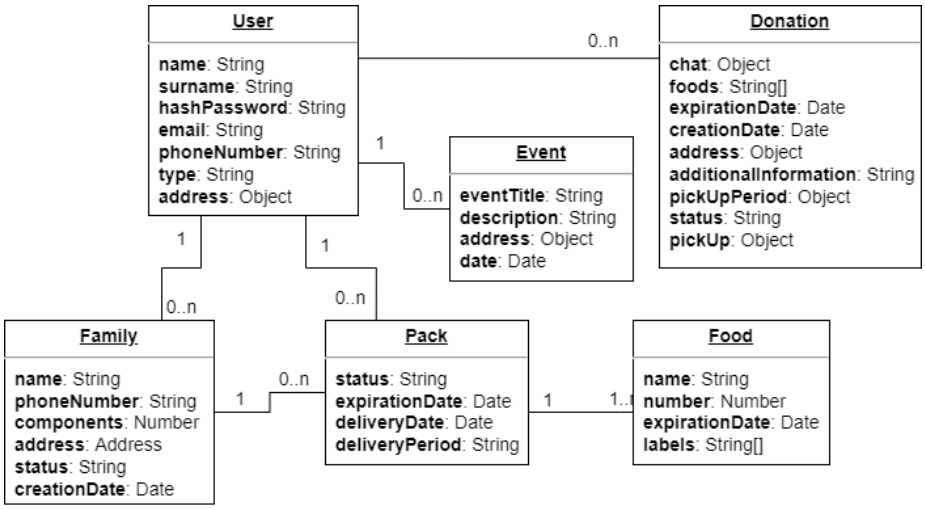

Figure 2.1: Diagramma che illustra il modello del database

Alcuni attributi sono stati implementati come oggetti composti:

- l'attributo Address contiene le stringhe relative a città, via e numero civico e una doppia di coordinate usata per localizzare l'indirizzo nella mappa
- l'attributo Chat comprende un array di nodi, identificati da un id sequenziale utile anche ai fini dell'ordinamento dei messaggi. Oltre all'id del mittente, al messaggio e alla data di invio si è deciso di salvare anche il nome del mittente, per semplificare l'interrogazione. Infine una coppia di booleani permettono di tenere traccia dell'avvenuta lettura del messaggio da parte del destinatario e del fatto che l'evento sia o meno un evento di sistema (es. presa in carico della donazione)
- il PickUpPeriod è stato modellato come una coppia di stringhe: una per il giorno della settimana e una per il periodo del giorno. La giornata è stata suddivisa in 3 periodi possibili: mattina, pomeriggio e sera.
- PickUp contiene informazioni sul prelievo della donazione, in particolare identifica il volontario che l'ha presa in carico e data e periodo di prelievo.

2.1.2 Autenticazione

Per l'autenticazione si è deciso di utilizzare il pacchetto npm jsonwebtoken per generare un token jwt all'azione di login dell'utente.

Una login con esito positivo restituisce al mittente un oggetto contenente le informazioni relative al suo account e un token jwt; quest'ultimo dovrà essere inserito nell'header di ogni richiesta http alle API del backend insieme all'id dell'utente.

Al momento della creazione il server codifica con una chiave segreta l'id dell'utente, producendo un hash che viene restituito all'utente come token jwt. Quando arriva una chiamata ad uno degli endpoint serviti dal backend, il token viene decriptato e confrontato con l'id dell'utente (entrambe le informazioni sono contenute nell'header della chiamata): se il match va a buon fine, la richiesta viene passata all'api interessata, altrimenti il server restituirà errore

Il token jwt viene invalidato in due casi: sono passate 12 ore dalla sua creazione o l'utente ha effettuato il log-out.

2.2 User centered design

L'applicazione verrà utilizzata principalmente da 2 categorie di utenti: donatori, i quali offrono in dono beni alimentari e volontari, i quali invece raccolgono le offerte ed organizzano pacchi alimentari da consegnare a famiglie bisognose.

Inoltre, si assume che le richieste di inserimento di famiglie bisognose possano essere effettuate indipendentemente da utenti o da volontari e che possano essere vagliate da utenti volontari *trusted*.

Le famiglie bisognose sono state volutamente ignorate nell'utilizzo della SPA considerando il fatto che potrebbero potenzialmente non essere in grado di usufruirne indipendentemente vista la loro situazione.

2.2.1 Personas

Carlo

Carlo è un uomo di 45 anni ed un cittadino che ha a cuore la situazione in cui versano le persone meno fortunate nella sua città. Egli vorrebbe poter contribuire in maniera diretta a migliorare la loro situazione. Tuttavia, non avendo molto tempo a disposizione, difficilmente è in grado di apportare un reale contributo e gradirebbe poter donare un pasto quando possibile.

Raramente gli capita di vedere situazioni di disagio e venire a conoscenza di persone o gruppi famigliari disagiati. Carlo vorrebbe poter aiutarli quando possibile ma spesso non è in grado di fornire un supporto duraturo e costante.

Chiara

Chiara è una donna di 27 anni e, come Carlo, vuole apportare un contributo nella sua città ed ha deciso di posizionarsi in prima linea per aiutare i più bisognosi. Chiara vorrebbe inoltre sensibilizzare la popolazione organizzando eventi pubblici di varia natura. Per questi motivi si è iscritta ad un associazione di volontariato nel suo comune.

2.2.2 Casi d'uso

Di seguito sono elencati i principali scenari d'uso dell'applicativo, relativamente alle *personas* presentate.

Effettuare donazioni

L'utente vuole:

- creare una donazione accedendo ad un'appropriata pagina dell'applicazione (navigando dalla pagina principale) ed inserendo i dati necessari per il ritiro come alimenti donati, luogo del ritiro, momenti della giornata in cui è possibile effettuare il ritiro e scadenza della donazione;
- monitorare le proprie donazioni, prevalentemente quelle attive, e visualizzarne informazioni di dettaglio o permetterne la cancellazione;
- comunicare direttamente con un volontario del ritiro delle donazioni: la *chat* è quindi relativa ad una sola donazione e mette in comunicazione l'utente con il singolo volontario attualmente incaricato.

Il volontario vuole:

- prendere una donazione in incarico per il ritiro, tramite la visualizzazione di tutte le donazioni attive nella città; la selezione delle donazioni avviene consultando una mappa interattiva dove è possibile mostrare le donazioni e le relative informazioni;
- visualizzare le donazioni per cui si è auto-incaricato del ritiro per visualizzarne le informazioni in qualsiasi momento, oppure per tenere traccia delle donazioni da ritirare in giornata, ma anche eventualmente di cancellare la prenotazione;
- comunicare direttamente con l'utente per il ritiro della donazione.

Segnalare famiglie bisognose

L'utente ed il volontario vogliono:

- inserire nuove segnalazioni, specificando tutte le informazioni necessarie per identificare, localizzare e contattare la famiglia bisognosa segnalata;
- monitorare le segnalazioni effettuate per verificarne lo stato o modificare i dati inseriti.

Il volontario *trusted* vuole:

- visualizzare le segnalazioni effettuate da utenti e volontari, esplorarne i dati per poter verificare la legittimità della richiesta;
- validare una segnalazione oppure invalidare una famiglia inserita nel sistema

Pacchi alimentari e alimenti

Il volontario vuole:

- aggiungere nuovi alimenti reperiti dalle donazioni effettuate;
- modificare o cancellare alimenti presenti in magazzino nel caso in cui essi siano scaduti;
- creare pacchi alimentari selezionando inizialmente una famiglia bisognosa ed in seguito gli alimenti da inserire nel pacco;
- consegnare pacchi alimentari precedentemente creati anche da altri volontari;
- visualizzare la lista di pacchi alimentari da consegnare per cui ha preso incarico.

Eventi

Il volontario vuole:

- creare un evento che dovrà essere mostrato nella pagina principale della applicazione;
- visualizzare e modificare la lista eventi creati, con particolare focus agli eventi futuri.

2.3 Frontend

Viste le personas e gli scenari d'uso, i goal e i task previsti per realizzare le funzionalità necessarie vengono ora utilizzati per creare le interfacce utente. La struttura di massima dell'applicazione così come alcune delle funzionalità previste verranno di seguito analizzate.

2.3.1 Struttura generale

In figura 2.2 è mostrato il layout generale della pagina. Si è scelto di disporre all'interno di una sidebar gli elementi di navigazione per facilitare il raggiungimento delle funzionalità agli utenti.

2.3.2 Donazioni

In figure 2.3 e 2.4 sono mostrate diverse schermate rispettivamente per utenti e volontari.

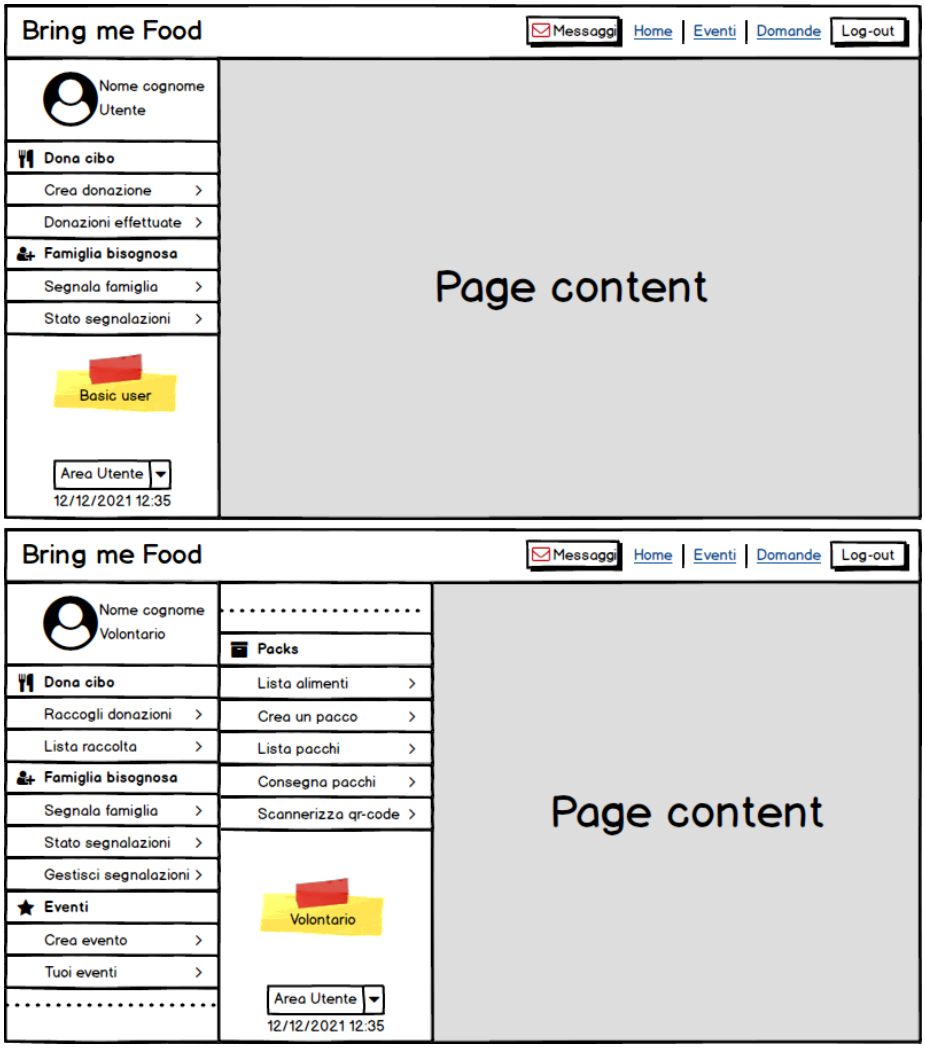

Figure 2.2: Layout di massima della pagina, sia per utenti donatori (sopra) che per utenti volontari (sotto).

Utente donatore

Visto che gli utenti devono essere in grado di creare, gestire e modificare le proprie donazioni, è stato deciso di implementare queste funzionalità come mostrato nella figura 2.3.

Nella prima immagine viene mostrato come creare una donazione ed i dati necessari come alimenti, una data di scadenza stimata dall'utente entro la quale si suppone la donazione sarà raccolta da un volontario, i periodi della giornata in cui sarà possibile reperire la donazione ed il luogo di ritiro.

I periodi di ritiro disponibili sono stati intesi come fasce orarie (mattino, pomeriggio e sera); in questo modo un volontario può sapere quando è il momento migliore per raccogliere la donazione senza imporre degli orari troppo stringenti. Eventualmente un utente può specificare informazioni aggiuntive o comunicare direttamente tramite la chat.

Inoltre, per migliorare il reperimento, l'utente dovrà indicare il più precisamente possibile il luogo del ritiro anche tramite una mappa interattiva. Potenzialmente, questo passaggio potrà far utilizzo del luogo inserito dall'utente stesso al momento della registrazione.

Nella terza immagine in figura 2.3 è possibile vedere la *chat* tramite la quale l'utente comunica con il volontario incaricato del ritiro della donazione. Questa stessa schermata sarà molto simile sia per volontario che utente con l'eccezione delle possibili azioni attuabili:

- l'utente sarà in grado di modificare o cancellare la donazione;
- il volontario sarà in grado di annullare la prenotazione per il ritiro oppure segnalare la donazione come ritirata.

Volontario

In figura 2.4 sono mostrate tre schermate inerenti alla prenotazione di donazioni da ritirare. Tramite una mappa interattiva (seconda immagine in figura), un volontario sarà in grado di visualizzare le donazioni attive e le informazioni più interessanti come alimenti offerti e data scadenza. Inoltre, i *marker* mostrati dipendono dalle impostazioni scelte per il ritiro quali data e periodo. In questo modo, date le impostazioni scelte, il volontario può agevolmente programmare la raccolta di donazioni.

2.3.3 Pacchi alimentari

Un'altra funzionalità cardine dell'applicazione è la gestione dei pacchi alimentari assegnati a famiglie bisognose registrate. La funzionalità si articola in 2 fasi:

- creazione di un pacco per una famiglia;
- consegna del pacco presso la relativa famiglia.

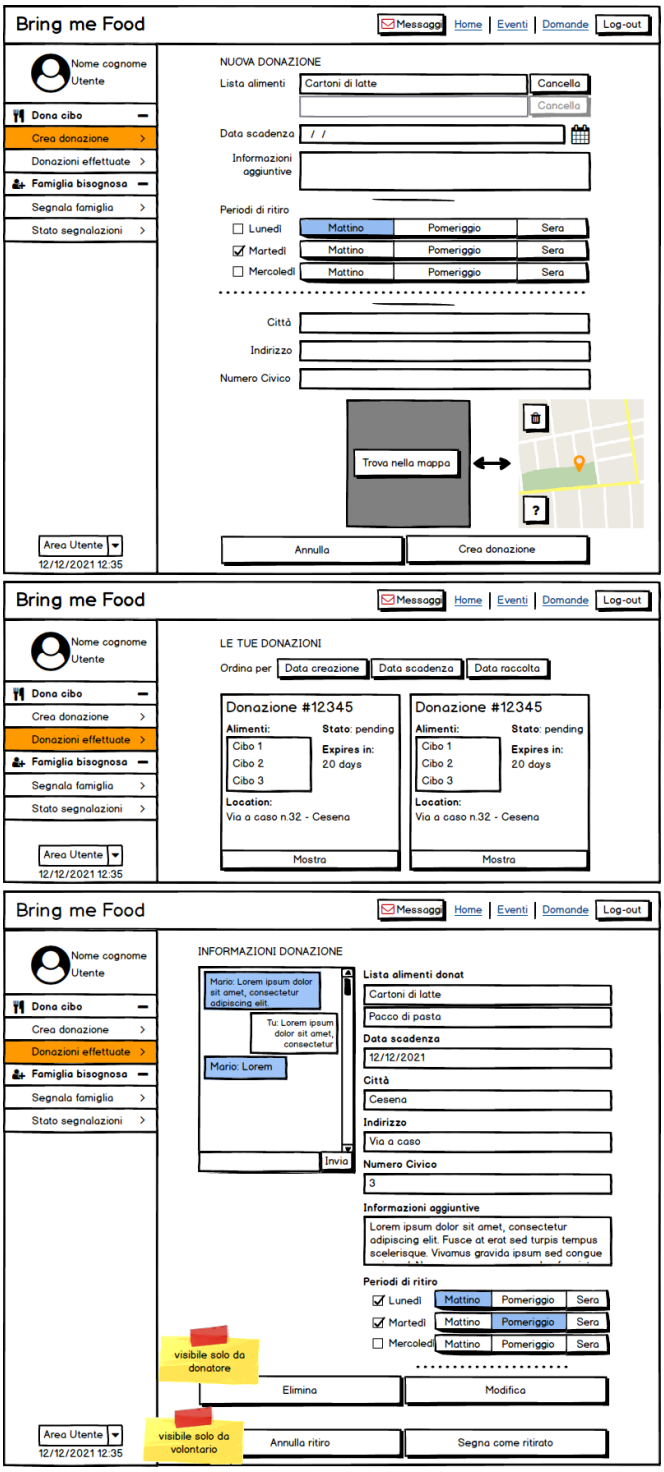

Figure 2.3: Layout e contenuto delle pagine inerenti alla creazione e gestione di donazioni da parte di utenti donatori.

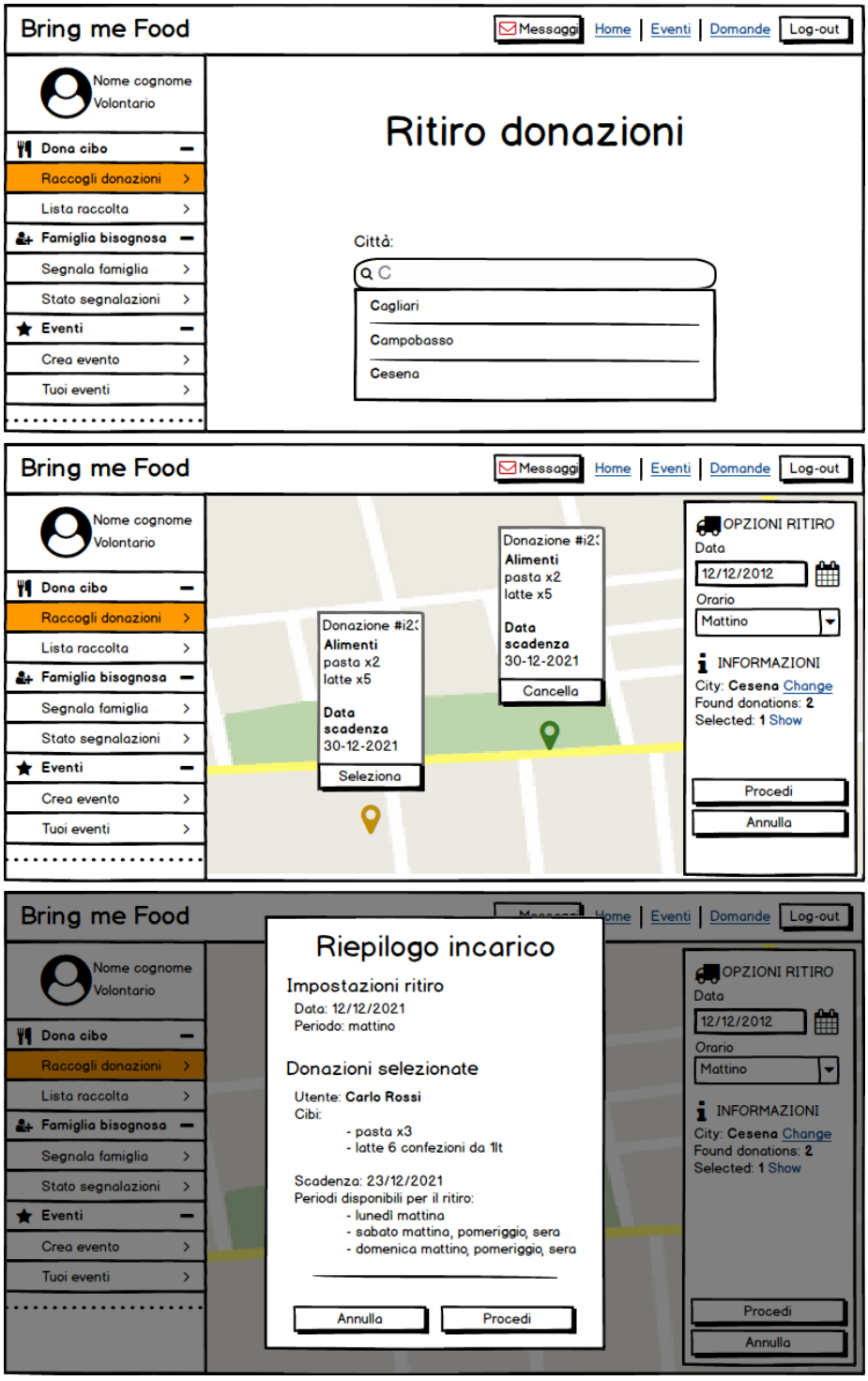

Figure 2.4: Layout e contenuto delle schermate inerenti alla visualizzazione e gestione di donazioni raccolte ad un utente volontario.

Creazione pacco alimentare

La creazione di un pacco alimentare mostrata in figura 2.5 prevede tre passaggi:

1. selezione della famiglia bisognosa;
2. selezione degli alimenti;
3. visualizzazione delle informazioni principali per la stampa del **codice QR** utile al momento del ritiro.

Ogni pacco è caratterizzato da una data di scadenza definita dalla data di scadenza più vicina tra i prodotti alimentari scelti.

In questo processo non sono definite data e ora per la consegna del pacco. Questo aspetto è definito in un secondo momento tramite la visualizzazione dei pacchi pronti alla consegna all'interno di una mappa interattiva, similmente a quanto fatto per la raccolta di donazioni in figura 2.4.

In questo modo è possibile separare i due aspetti ed anche da un punto di vista pratico di gestione del magazzino e delle attività fisiche, si rende possibile la creazione di molteplici pacchi (magari a carico di un particolare volontario) e successivamente la consegna tramite un'apposita schermata (attività svolta da un altro volontario specializzato nella consegna).

Gestione e consegna di un pacco alimentare

In figura 2.6 sono mostrate 3 schermate:

1. lista dei pacchi alimentari, sia prenotati dall'utente che tutti i pacchi;
2. scanner per l'acquisizione di un codice QR;
3. visualizzare informazioni del pacco acquisito e impostarlo come consegnato.

Ogni pacco è caratterizzato da un codice QR per agevolare la modifica dello stato al momento della consegna.

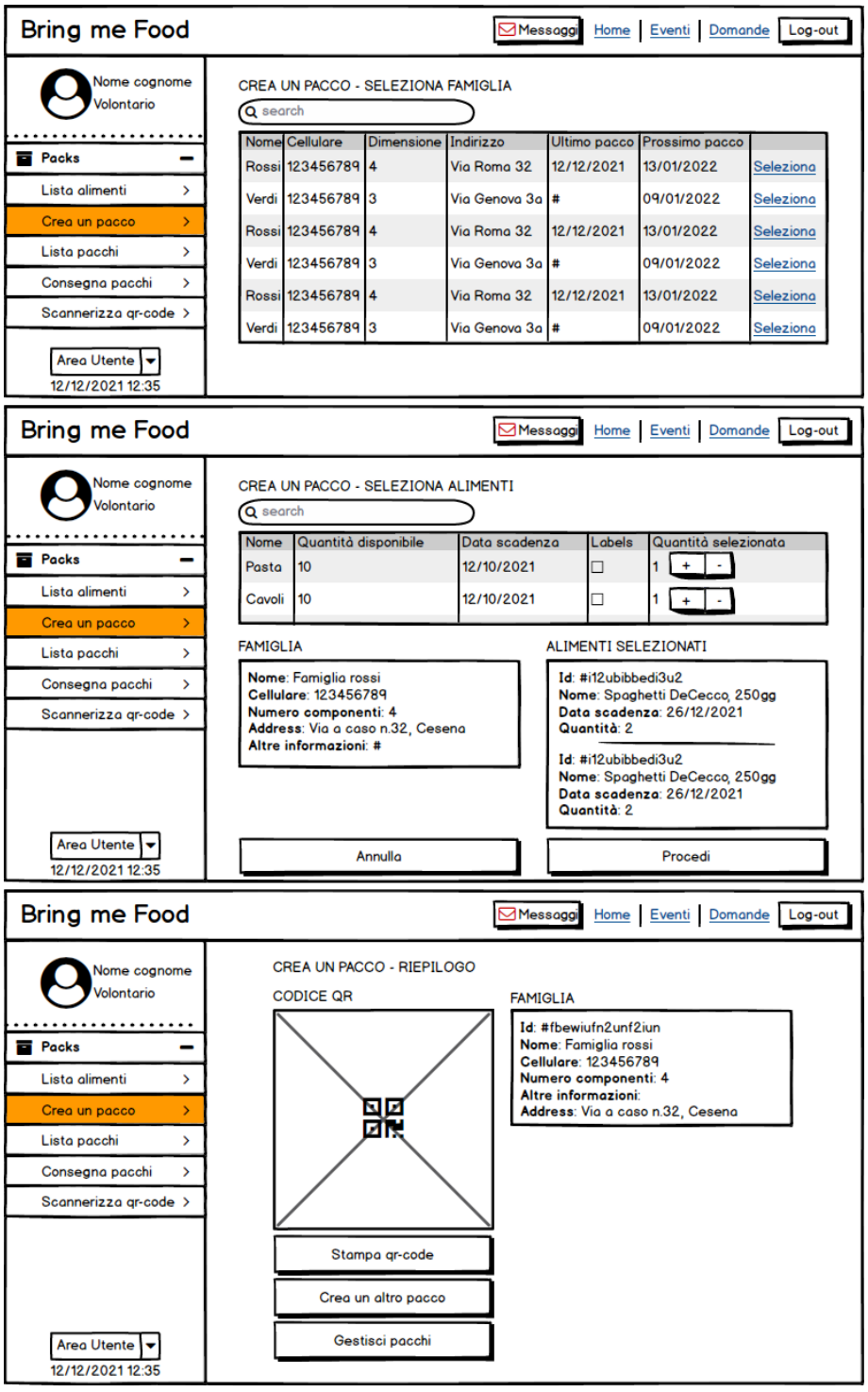

Figure 2.5: Layout e contenuto delle schermate inerenti alla creazione di un pacco alimentare da parte di un utente volontario.

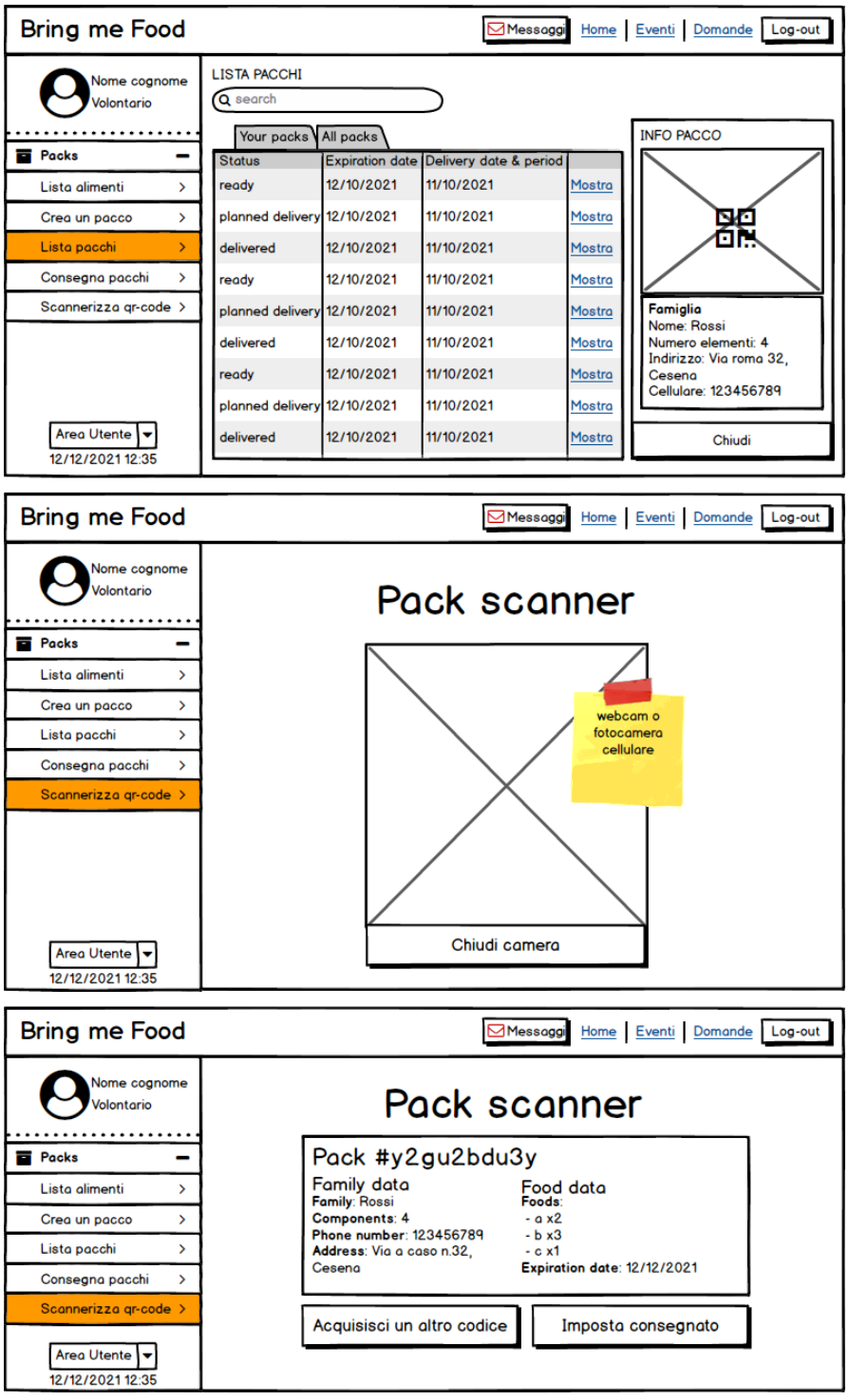

Figure 2.6: Layout e contenuto delle schermate inerenti alla gestione dei pacchi alimentari (sopra), scannerizzazione e visualizzazione dello stato del pacco (centro e sotto).

Chapter 3

Tecnologie

3.1 Stack MEVN

3.1.1 MongoDB

Per il database è stato usato MongoDB, DBMS non relazionale orientato a documenti in stile JSON anzichè alle tabelle come i RDBMS.

La libreria Mongoose ha permesso l'interazione con il database dal backend attraverso JavaScript grazie alle API di interrogazione fornite. In particolare sono stati definiti sul backend i **modelli** relativi ai documenti delle varie collezioni, che descrivono la struttura dei suddetti documenti, mentre nei **controller** sono state racchiuse le operazioni che permettono invece di eseguire le varie operazioni CRUD sulle collezioni.

Si è deciso di spostare le funzioni comuni dei vari controller in una classe factory che prende in input il modello del documento e restituisce un oggetto in grado effettuare le operazioni di find, add, delete e update sulla collezione contenente i documenti con quel modello. Questo ha permesso di riciclare le funzioni comuni senza andare a ripeterle su ogni controlle, mentre le funzioni più specifiche per un determinato documento sono state implementate nello specifico controller.

3.1.2 Express

Il framework Express è stato usato sul backend per implementare il il sistema di routing e i relativi handler associati a ogni route.

Un handler comune a quasi tutti gli endpoint è rappresentato dalla funzione di validazione del token jwt, implementata usando il pacchetto npm *javawebtoken*. In caso di avvenuta validazione la richiesta viene passata all'handler vero e proprio relativo a quella route.

3.1.3 Vue

Vue è stato utilizzato per lo sviluppo dell'intera porzione frontend dell'applicazione. E' stato scelto poichè ritenuto più semplice da apprendere. E' stato successivamente necessario ampliare le funzionalità offerte dal framework con l'utilizzo di VueRouter per la gestione della navigazione all'interno della Single Page Application ed anche VueX nella gestione dello stato come ad esempio una sessione utente. Inoltre quest'ultimo ha trovato interessanti integrazioni con SocketIO.

Altri moduli aggiuntivi utilizzati sono:

- vue-persistedstate per permettere il refresh della pagina web mantenendo la sessione utente;
- vue-socket.io per l'integrazione di SocketIO con Vue ed in particolare vuex;
- qrode-vue per la creazione di codici QR e vue-qrcode-reader, un componente che permette l'utilizzo di webcam da computer e fotocamere da cellulare per l'acquisizione di codici QR;
- gmap-vue, vue2-gmap-custom-marker e vue-google-autocomplete vari moduli per l'integrazione di api google-maps.

3.1.4 Node

Per l'implementazione del server è stato usato NodeJs, ambiente javascript event-driven che ha permesso di rimanere fedeli al principio *javascript-everywhere*: JS usato sia lato frontend che lato backend, ottenendo così un'ottima interoperabilità.

3.2 TypeScript

Si è deciso di usare TypeScript per i vantaggi e le funzionalità offerte dal superset di JavaScript: in particolare la possibilità di definire classi e interfacce e la possibilità di specificare tipi di parametri e tipi di ritorno delle funzioni. Il *type checking* statico ha permesso di sviluppare il sistema riducendo il numero di errori riscontrati a runtime.

3.3 SocketIO

SocketIO, un framework per la comunicazione bidirezionale e real-time tra client e server, è stato utilizzato per consentire l'implementazione di una chat in tempo reale tra utenti all'interno dell'applicazione. SocketIO ha trovato lato front-end interessanti integrazioni con il framework Vue ed un suo modulo vuex.

Chapter 4

Codice

Il listing seguenti riportano le funzione di import e export dei documenti della collezione User, che contiene i dati degli utenti registrati al sistema.

Listing 4.1: Export dei dati dell'utente dal database

```
exportUserCollection = () => {
  UserModel.find().select("+hashPassword")
    .then((data) => this.save("users", data))
}
```

Listing 4.2: Salvataggio dei dati estratti in un file JSON

```
private save = (fileName: string, data: any) => {
  const rawdata = JSON.stringify(data)
  const filePath = __dirname + '/../../data/' + fileName + '.json'
  fs.writeFileSync(filePath, rawdata)
}
```

Listing 4.3: Import dei dati da file JSON al database

```
private load = (collectionName: string): any => {
  mongoose.connection.db.dropCollection(collectionName)

  const fileName = __dirname + '/../../data/' + collectionName + '.json'
  const rawdata = fs.readFileSync(fileName).toString()

  return JSON.parse(rawdata)
}
```

4.1 Chat con SocketIO

4.1.1 Server side

Lato server è necessario essere a conoscenza di quali siano tutti gli utenti attualmente connessi ed autenticati visto che essi potrebbero in qualsiasi momento ricevere messaggi.

Per questo motivo sono stati implementati i comani *login* e *logout* attraverso i quali viene mantenuta una mappa, chiamata activeSockets, di coppie (id utente, id socket).

Listing 4.4: gestione di multiple socket tramite messaggi *login* e *logut*.

```
let activeSockets = new Map();

socket.on("login", (userId: string) =>
  { activeSockets.set(userId, socket.id); })

socket.on("logout", (userId: string) =>
  { activeSockets.delete(userId); })
```

Un altra funzionalità essenziale è la ricezione di messaggi inviati dalla chat degli utenti, rappresentata dal messaggio *message to server*. Proprio in questo punto vengono inizialmente inseriti i messaggi all'interno del database ed eventualmente vengono anche inviati agli utenti interessati tramite la loro socket qualora essi siano connessi contemporaneamente al sistema.

Listing 4.5: Messaggio *message to server* per permettere il salvataggio e invio di messaggi tra utenti connessi.

```
socket.on("message to server", (obj: any) => {

addMessageToChat(obj.donationId, obj.userId, obj.fullname,
obj.message, obj.isEventMessage)
.then(newMessage => {

// send the new message to all the users involved in the chat
// (owner and optionally a volunteer)
getDonationUsers(obj.donationId)
.then((ids:any) => {

  if (ids) {
    const userId = ids['userId'] ?
      ids['userId'].toString() : null;
    const volunteerId = ids['volunteerId'] ?
      ids['volunteerId'].toString() : null;

    // send the message to the user
```

```
    if (userId && activeSockets.has(userId)) {
      const destSocket =
          io.sockets.sockets.get(activeSockets.get(userId))
      if (destSocket)
        destSocket.emit("chat message",
        JSON.stringify(newMessage));
    }

    // send the message to the volunteer
    if (volunteerId && activeSockets.has(volunteerId)) {
      const destSocket =
          io.sockets.sockets.get(activeSockets.get(volunteerId))
      if (destSocket)
        destSocket.emit("chat message",
        JSON.stringify(newMessage));
    }
  }
});  })  });
```

Ultimo aspetto importante del processo è l'impostazione di un messaggio come visualizzato. Questa attività avviene non appena un utente con la chat aperta riceve un messaggio inviato dal server.

Listing 4.6: Ricezione del messaggio *visualize message* per la visualizzazione di un singolo messaggio da parte dell'utente.

```
socket.on("visualize_message", (jsonMessage: any) => {
  const message = JSON.parse(jsonMessage)
  setMessageAsVisualized(message.donationId, message.message.index);
});
```

4.1.2 Client side

Lato Client, la gestione della comunicazione per mezzo di SocketIO è stata implementata sfruttando l'integrazione *vue-socket.io* tramite *vuex*, un modulo che estende l'utilizzo di VueJS con una più agevole gestione dello stato dell'applicazione.

Il Client riceve il messaggio *chat message* inviato dal server qualora un nuovo messaggio debba essere ricevuto. Questo aspetto comprende anche i messaggi stessi inviati dall'utente, i quali trasparentemente all'utente stesso sono salvati e inviati ad altri utenti interessati, ed anche messaggi appartenenti a diverse donazioni, permettendo un opportuna notificazione all'utente connesso della presenza di messaggi non letti. Alla ricezione di un nuovo messaggio, è necessario capire se esso deve essere immediatamente visualizzato (di fatto quando la chat è aperta) oppure no. Un messaggio visualizzato provoca un messaggio *visualize message* inviato tramite la socket al server il quale provvederà ad impostare un flag all'interno del messaggio nel database.

```
SOCKET_connect({ commit }: { commit: Commit }): void {
  commit("setConnected", true);
},

SOCKET_disconnect({ commit }: { commit: Commit }): void {
  commit("setConnected", false);
},

SOCKET_chat_message(
  { commit, state, rootState }, stringMessage: string): void {
  const message = JSON.parse(stringMessage);

  if (state.donationId == message._id) {
    commit("addMessage", message.message);

    // set message as visualized
    if (message.message.userId != rootState.session.userData._id)
      new Vue().$socket.emit("visualize_message", stringMessage);
  } else if (rootState.session.userData._id !=
             message.message.userId) {
    // update messages count
    commit("addUnreadMessage", message);
  }
},
```

Chapter 5

Test

Per testare il sistema sviluppato si è deciso di adottare la metodologia di Cognitive Walkthrough.

Sono stati studiati i principali task completabili attraverso il sistema e sono stati definiti due insiemi associati alle tipologie principali di utenti, cioè gli utenti standard e gli utenti volontari.

In seguito sono stati individuati 6 potenziali utenti, suddivisi nei due gruppi di utenti sopracitati, ed è stato chiesto loro di portare a termine la lista di task associata a quella tipologia di utente.

I task per gli utenti standard erano i seguenti:

- donazione di un insieme di alimenti
  - accedere alla vista di donazione degli alimenti
  - compilare e sottoscrivere il relativo form
  - verificare le informazione relative alla donazione appena effettuata
  - utilizzare la chat relativa alla donazione per chiedere delucidazioni a un utente volontario
- segnalazione di una famiglia bisognosa
  - accedere alla vista di segnalazione delle famiglie
  - compilare e sottoscrivere il form di segnalazione
  - verificare lo stato della propria segnalazione

I task per gli utenti volontari erano i seguenti:

- creazione di un evento di sensibilizzazione
  - accedere alla schermata di creazione di eventi
  - creazione di un evento tramite compilazione del form

- verifica dell'evento appena creato sia dalla lista dei propri eventi che dalla homepage comune

creazione di un giro di raccolta delle donazioni

- accedere alla schermata di creazione dei giri di consegna
- selezione della città di consegna
- compilazione del form tramite selezione di una consegna da raccogliere e di data e periodo della giornata in cui farlo
- verifica del prossimo giro di raccolta previsto per l'utente

creazione di un pacco alimentare

- accedere alla schermata di creazione dei pacchi alimentari
- selezione della famiglia per la quale creare il pacco
- selezione degli alimenti da inserire nel pacco
- sottoscrizione del form
- verifica delle informazioni del pacco appena creato: famiglia associata e cibo contenuto

Dai task sono stati omessi alcuni passaggi comuni, quali la registrazione dell'utente e l'accesso al sistema tramite login. Agli utenti volontari sono stati forniti account pre-registrati che avevano già subito l'upgrade.

Sulla base dei feedback rilasciati dai potenziali utenti è stato possibile apportare miglioramenti al sistema per potenziarne l'usabilità.

I test relativi a comportamenti anomali o input non validi nei vari form proposti sono stati studiati e sistemati dal team di sviluppo.

Chapter 6

Deployment

I moduli frontend e backend sono compilabili ed eseguibili singolarmente tramite i comandi definiti nei rispettivi package.json:

- `npm run serve` per il frontend
- `npm run build-start` per il backend

Questa procedura è stata usata in fase di sviluppo, mentre per il deployment è stato usato il container manager Docker. Ad ogni modulo è stata associata un'immagine definita nel relativo Dockerfile. Per l'esecuzione delle 3 immagini come unica applicazione multi-container è stato usato Docker Compose. Di seguito i comandi necessari per eseguire le immagini frontend, backend e mongodb:

```
docker-compose build
docker-compose up
```

Prima di eseguire l'applicazione è necessario cambiare la configurazione del backend perchè punti al database corretto e non a quello locale in localhost(la configurazione è contenuta nel file properties.env).

Per avere il database pre-popolato si è deciso di usare il seguente workaround basato su due servizi localizzati nel backend: con un servizio è possibile esportare le collezioni necessarie all'applicazione in una serie di file json contenuti in un'apposita cartella (questo ha permesso di versionare anche i dati del database) con il secondo servizio è possibile importare i file json sopracitati e sovrascrivere le corrispondenti collezioni attuali In questo modo è possibile creare il container mongo e popolarlo con i file json versionati precedentemente.

Chapter 7

Conclusioni

Il progetto ha permesso al team di sviluppo di accrescere il proprio bagaglio in termini di tecnologie per il web development. Premettendo che tutti i membri avevano già in partenza un background professionale sul tema, è stato interessante approfondire certi temi e in particolare interfacciarsi con un database NoSQL, prima esperienza condivisa da tutto il team di sviluppo.

Per quanto riguarda gli sviluppi futuri sarebbe utile implementare alcune feature non incluse nei requisiti per contenere i tempi di progetto e concentrarsi su i requisiti principali: in particolare la possibilità di impostare immagini utente o relative agli eventi da questi creati. Sarebbe interessante testare alcune feature in maniera automatica tramite framework come Mocha, in maniera da poter implementare anche una pipeline di CI su GitHub.