MoMa

Applicazioni e Servizi Web

Michele Torroni - 0001050680 {michele.torroni@studio.unibo.it}

23 Giugno 2022

Indice

0.1 Introduzione 3

0.2 Requisiti 4

0.2.1 Requisiti Funzionali 4

0.2.2 Requisiti Non Funzionali 5

0.2.3 Requisiti Di Implementazione 5

0.3 Design e Architettura 6

0.3.1 Design 6

0.3.2 Architettura Sistema 6

0.3.3 Architettura Interfacce Utente – Mockup e StoryBoard . 8

0.3.3.1 Login 8

0.3.3.2 Sign Up 9

0.3.3.3 Application Layout 11

0.3.3.3.1 Categorie 12

0.3.3.3.1.1 Categories 12

0.3.3.3.1.2 Add Category 13

0.3.3.3.1.3 Edit Category 14

0.3.3.3.2 Portafogli 15

0.3.3.3.2.1 Wallets 15

0.3.3.3.2.2 Add Wallet 16

0.3.3.3.3 Transazioni 17
0.3.3.3.3.1 Transactions 17
0.3.3.3.3.2 Add Transaction 18
0.3.3.3.4 Account 19
0.3.4 Design di dettaglio 21
0.3.4.1 Backend 21
0.3.4.2 Frontend 21
0.3.5 Tartget User Analysis 22
0.3.5.1 Personas: Giacomo 23
0.3.5.2 Personas: Francesca 23
0.4 Tecnologie 25
0.4.1 MongoDB e Mongoose 25
0.4.2 Express 26
0.4.3 Vue.js, Vue Router e Vuetify 26
0.4.4 Node.js 27
0.4.5 Socket.IO 28
0.4.6 Socket.IO 28
0.5 Codice - Aspetti Salienti 29
0.5.1 Backend 29
0.5.2 Frontend 30
0.6 Test 32
0.7 Deployment 34
0.8 Conclusioni 35

0.1 Introduzione

MoMa (abbreviazione di “Money Manager”) é un’applicazione web che consente agli utenti di tenere traccia delle proprie transazioni monetarie.

Il sistema fornisce la possibilità di creare categorie di spesa (“categories”, ad esempio: “food”, “shopping”, “presents”) e dei portafogli (“wallets”, ad esempio: “wallet”, “credit card”) al fine di categorizzare le transazioni; infatti ogni spesa é discriminata da una category e un wallet di riferimento, e, alla creazione, MoMa aggiorna in automatico i dati di tutte le entità in gioco, in modo da fornire all’utente uno storico quanto piú preciso possibile.

Per fornire un’adeguata sicurezza dei contenuti, il sistema richiede un’iscrizione (con password) e l’utente potrà decidere in qualsiasi momento di eliminare tutti i contenuti (compreso il proprio account) dall’applicazione.

0.2 Requisiti

0.2.1 Requisiti Funzionali

L'utente all'interno dell'applicazione potrà:

- Creare il proprio profilo, consistente in uno username, una password e un'immagine;
- Modificare tutte le informazioni del proprio account dopo aver effettuato l'accesso, compresa la possibilità di cancellare il profilo e tutti i dati ad esso collegati;
- Creare le proprie categorie di spesa, ognuna con un nome e un colore di riferimento;
- Modificare e cancellare le categorie create e le transazioni ad esse collegate;
- Creare i propri portafogli, con un nome e un colore ciascuno, analogamente a quanto accade per le categorie, oltre ad un saldo di partenza;
- Aggiornare il saldo dei portafogli;
- Eliminare i portafogli e tutte le transazioni ad essi collegate;
- Creare transazioni, discriminate da una categoria e un portafoglio di riferimento, oltre ad un nome, un ammontare in denaro e una data;
- Eliminare le transazioni;
- Filtrare le transazioni per intervallo di date, categoria e portafoglio di riferimento;

0.2.2 Requisiti Non Funzionali

L'applicazione dovrà:

- Avere un'interfaccia responsiva;
- Essere Accessibile, semplice e facilmente interpretabile;
- Fornire meccanismi di autenticazione sicuri;
- Essere estensibile con nuove funzionalità;

0.2.3 Requisiti Di Implementazione

L'applicazione dovrà:

- Essere sviluppata utilizzando lo stack M.E.V.N.;
- Utilizzare MongoDB come DBMS (non relazionale);
- Fare utilizzo della libreria Socket.IO per fornire comunicazioni real-time;

0.3 Design e Architettura

0.3.1 Design

L'applicazione é stata sviluppata facendo uso dello stack MEVN, una variante dello stack MEAN (in cui viene utilizzato il framework javascript Vue.Js in sostituzione ad Angular.Js) e il cui acronimo rappresenta le tecnologia utilizzate:

- MongoDB
- Express
- React
- Node.js

Queste tecnologie verranno descritte nello specifico all'interno del capitolo 4.

Durante lo svolgimento del progetto si é fatto fede ai principi KISS ("Keep It Stupid, Simple") e DRY ("Don't Repeat Yourself"), notevolmente aggevolati dal meccanismo dei "components" fornito da Vue.Js.

Il Design dell'applicativo si basa sul modello iterativo UCD (acronimo di "User Centered Design"), con utenti virtualizzati mediante il meccanismo delle "Personas": si tratta di utenti target "immaginari", ma con caratteristiche peculiari riscontrabili in individui reali.

0.3.2 Architettura Sistema

L'architettura del sistema é ascrivibile al modello REST ("Representational State Transfer"), nella quale le comunicazioni avvengono su HTTP e il sistema é di tipo "stateless".

L'applicazione puó essere descritta mediante quattro Entità:

1. User: utente finale che utilizza il sistema e ha la completa proprietà dei propri dati;

2. Categories: categorie di spesa, personalizzabile nel colore e nel nome, che deve essere univoco all'interno delle categorie dell'utente;

3. Wallets: portafogli, che modellano tutti i possibili loci in cui l'utente conserva i propri risparmi, come "salvadanaio", "conto corrente", "carta prepagata", ecc. . . ;

4. Transactions: transazioni, che rappresentano le spese dell'utente, ognuna delle quali si riferisce ad una specifica categoria e uno specifico portafoglio.

Di seguito uno schema riassuntivo delle relazioni tra le entità.

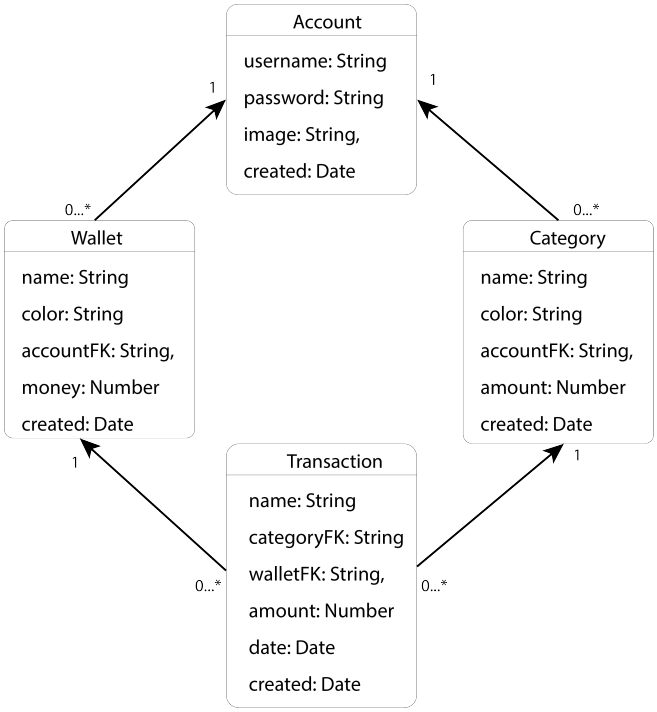

Figure 1: Schema Entità e relazioni

N.B. tutte le entità in gioco sono collegate al solo utente proprietario di

quelle informazioni, per garantire il "Diritto all'oblio" dell'utente nel caso in cui esso sia intenzionato ad eliminare il proprio account e tutte le informazioni ad esso collegate.

0.3.3 Architettura Interfacce Utente – Mockup e Story-Board

Di seguito si presentano i mockups disegnati in fase di analisi del progetto a confronto con le corrispondenti schermate sviluppate, in quanto queste ultime seguono quasi fedelmente le idee dei bozzetti.

0.3.3.1 Login

Appena l'utente apre l'applicazione web si trova davanti alla schermata di login, in cui deve inserire username e password con cui ha eseguito l'iscrizione al sistema.

Figure 2: Mockup Login

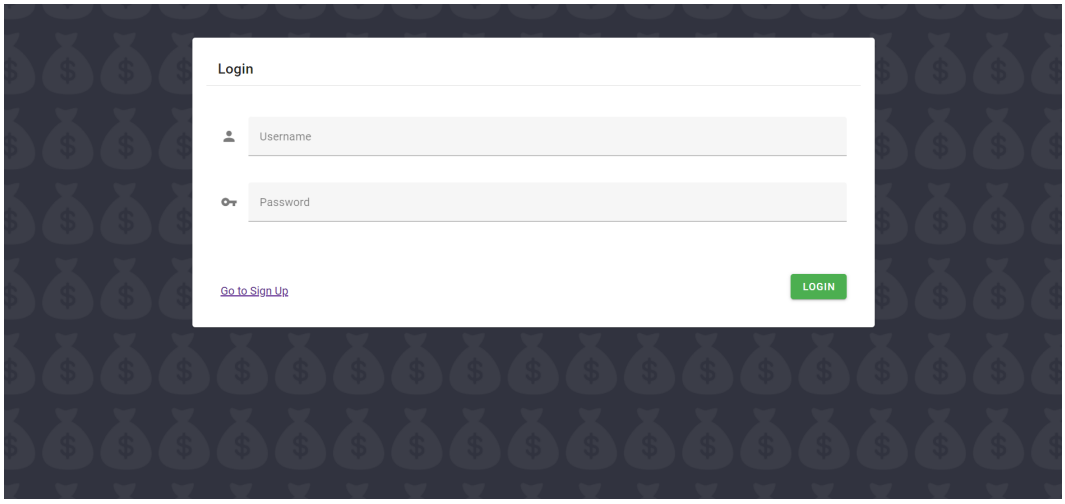

Figure 3: Screen Login

0.3.3.2 Sign Up

Se l'utente non ha ancora effettuato il Sign Up all'applicazione, potrà registrarsi mediante l'apposita schermata, accessibile da un link posizionato in basso a sinistra della schermata di Log In.

La schermata presenta la possibilità di inserire un'immagine di profilo, uno username e una password (che deve essere ripetuta correttamente al fine di procedere all'iscrizione al sistema).
L'interfaccia presenta eventuali errori all'utente (come "username already used") mediante un banner posizionato sotto ai campi compilabili

Figure 4: Mockup Sign Up

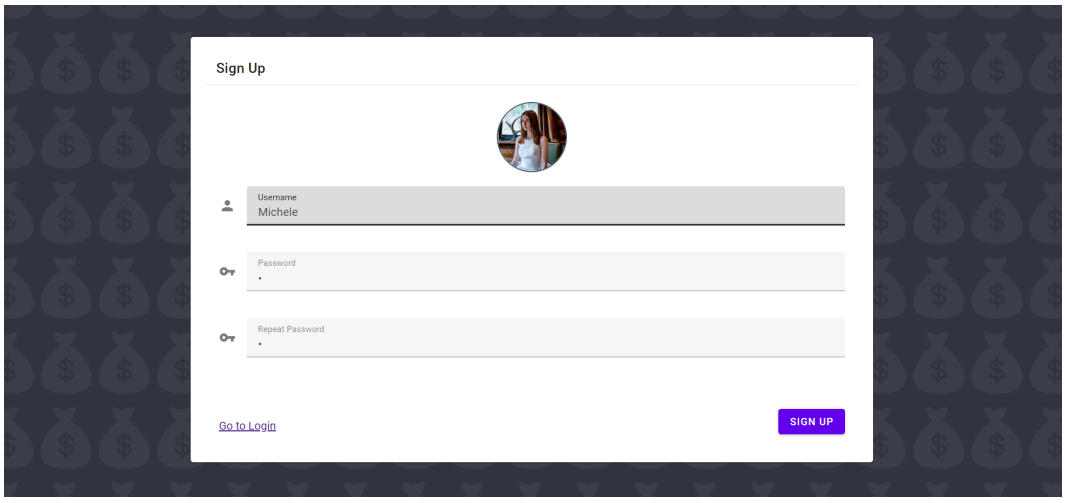

Figure 5: Screen Sign Up

0.3.3.3 Application Layout

Succesivamente ad aver effettuato l'accesso, l'utente potrà interagire con l'applicazione, che presenta una struttura composta da tre componenti:

1. Una Toolbar, contente un icona "menu a panino", il nome dell'applicazione, lo username dell'utente connesso e la sua immagine di profilo
2. Una Sidebar apribile/nascondibile con il sopracitato menu a panino, e che contiene i link di navigazione alle varie schermate dell'applicazione
3. Tutta la porzione rimanente dell'interfaccia contiene le informazioni contestuali alla schermata navigata (verranno presentati ai punti 3.3.6.x di questa tesi di progetto)

In aggiunta, tutte le schermate accessibili mediante la sidebar presentano un "floating button" in basso a destra che permette di creare una nuova entità, contestualmente alla schermata in cui ci si trova (es: Categories -¿ Add Category)

Figure 6: Mockup Application Layout

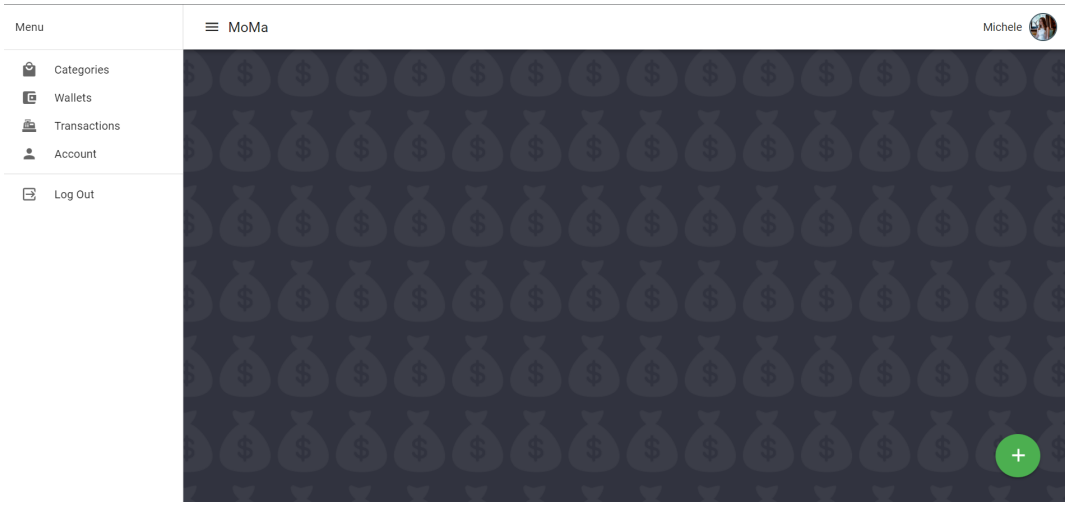

Figure 7: Screen Application Layout

0.3.3.3.1 Categorie

0.3.3.3.1.1 Categories Schermata che mostra le categorie dell'utente, mediante card colorate (colore selezionabile dall'utente) disposte con un layout

a “griglia adattiva”.

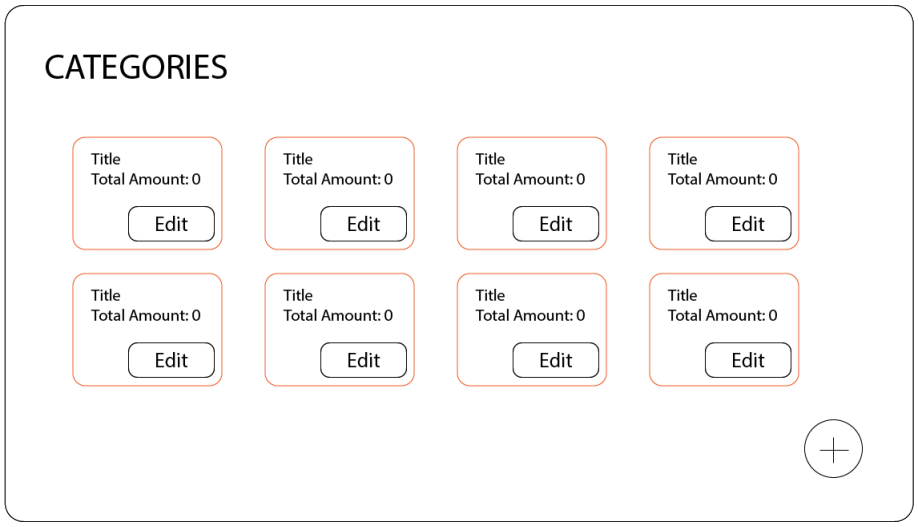

Figure 8: Mockup Categories

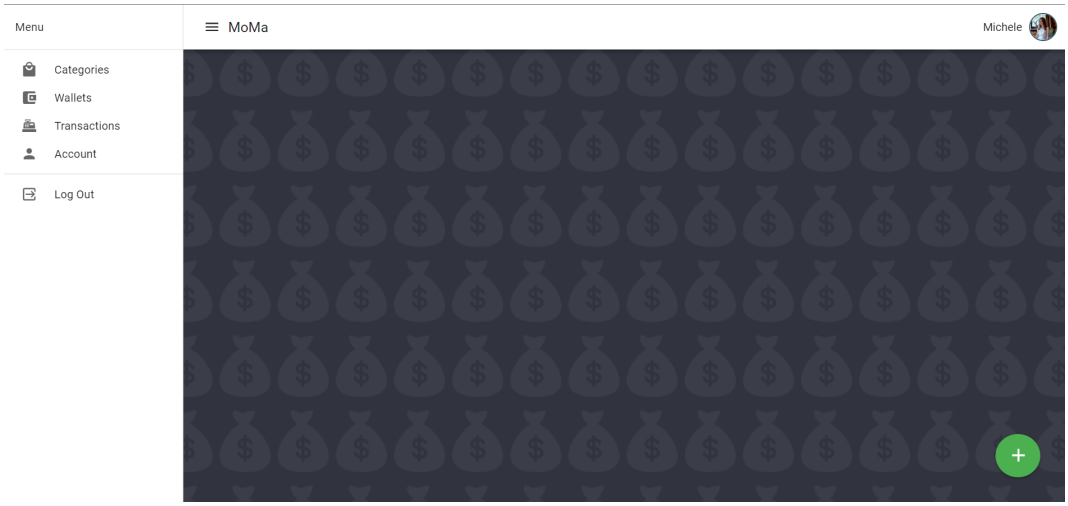

Figure 9: Screen Categories

0.3.3.3.1.2 Add Category Schermata, accessibile mediante il floating button citato a 3.3.6, che permette di aggiungere una nuova categoria

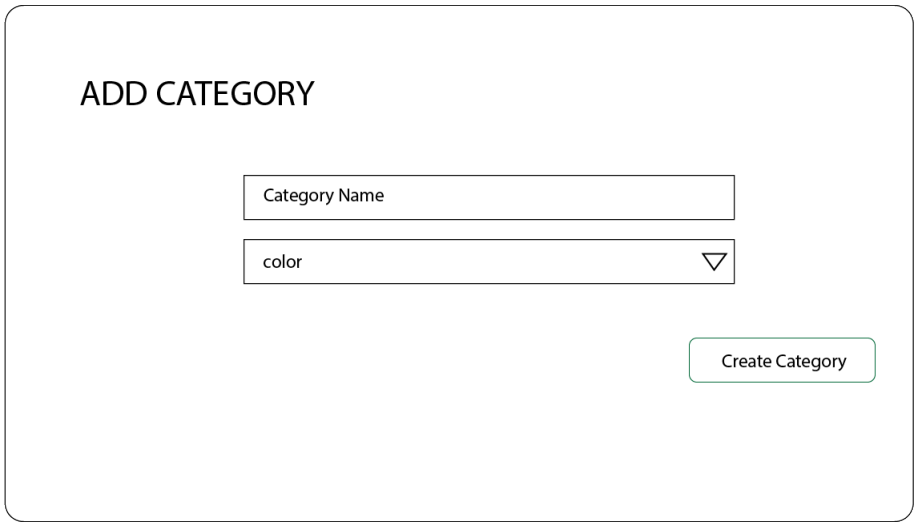

Figure 10: Mockup Add Category

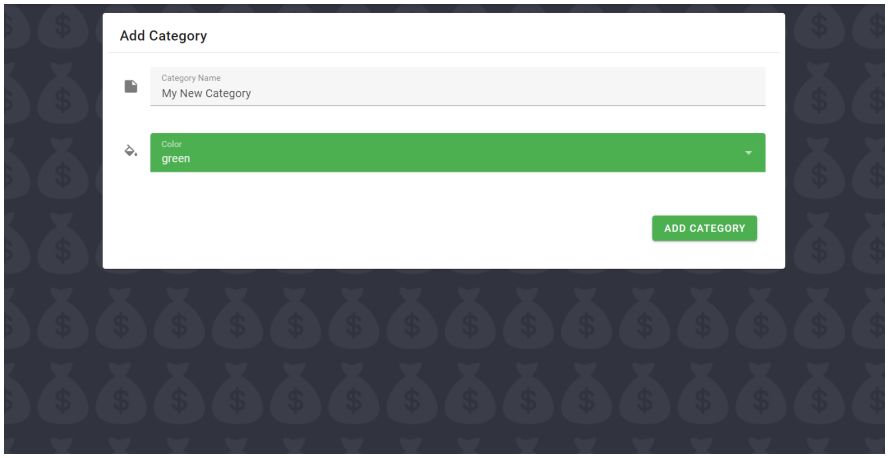

Figure 11: Screen Add Category

0.3.3.3.1.3 Edit Category Al click sul pulsante “Edit” (presente sulla card delle Categorie), verrà aperta la schermata di editing della categoria selezionata, dove sarà inoltre possibile eliminare l’entità.

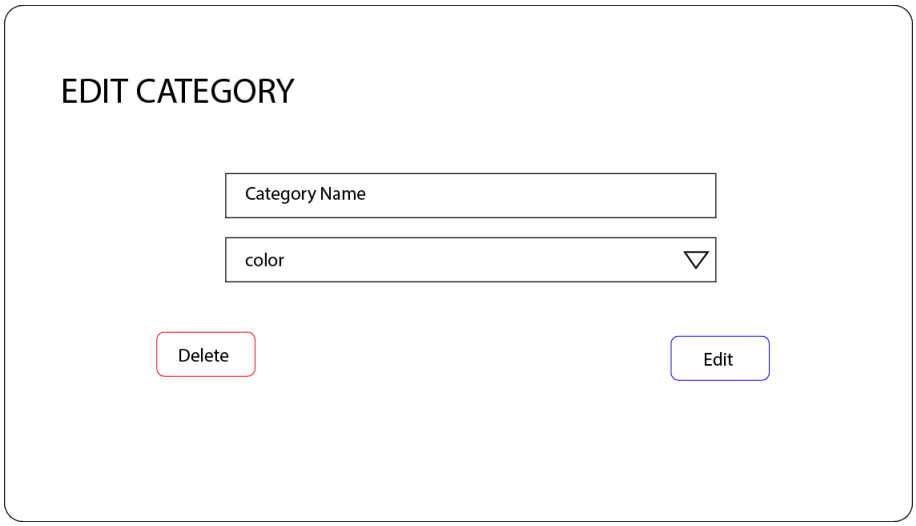

Figure 12: Mockup Edit Category

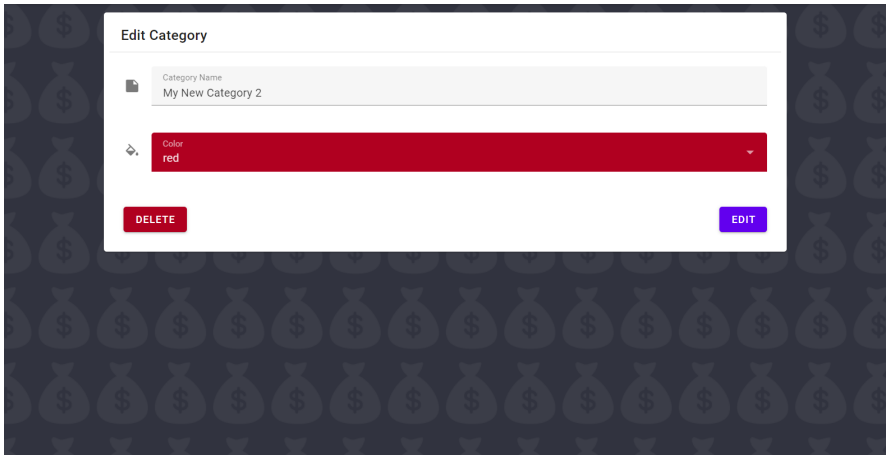

Figure 13: Screen Edit Category

0.3.3.3.2 Portafogli

0.3.3.3.2.1 Wallets Schermata di visualizzazione dei portafogli, accessibile mediante apposito link nella sidebar. I Wallets sono presentati mediante card “orizzontali”, attraverso le quali é possibile eliminare il portafoglio (e le

relative transazioni), oltre a poter aggiungere e togliere denaro.

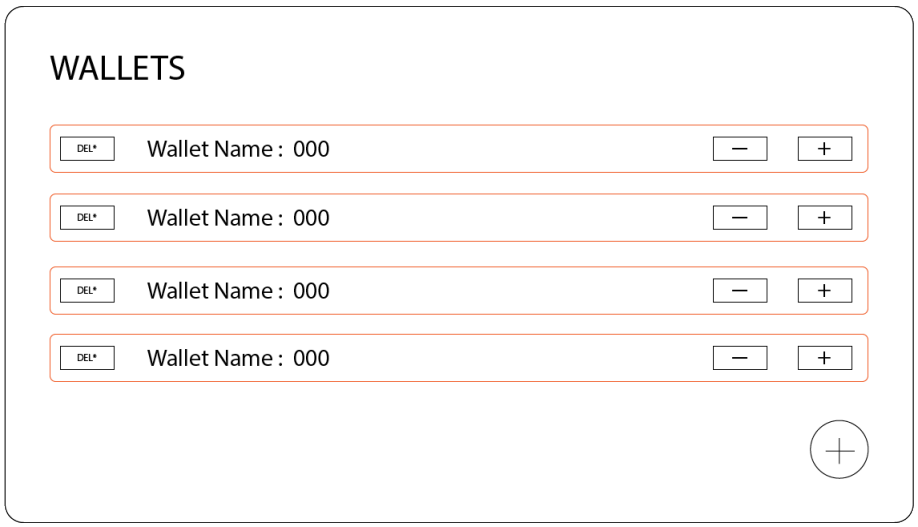

Figure 14: Mockup Wallets

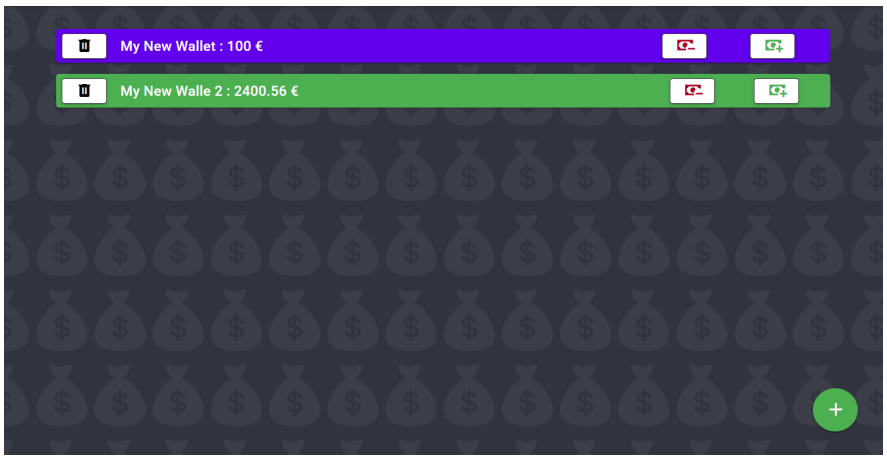

Figure 15: Screen Wallets

0.3.3.3.2.2 Add Wallet Schermata, accessibile mediante il floating button citato a 3.3.6, che permette di aggiungere un nuovo portafoglio.

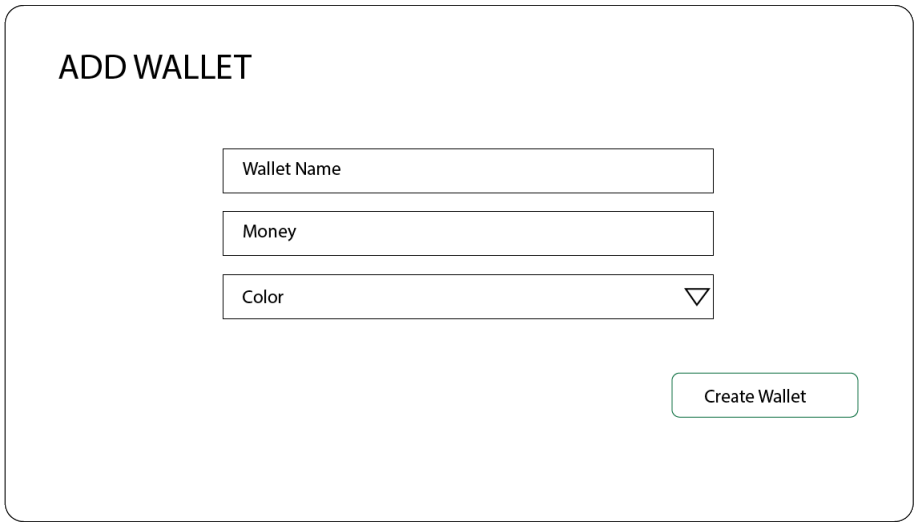

Figure 16: Mockup Add Wallet

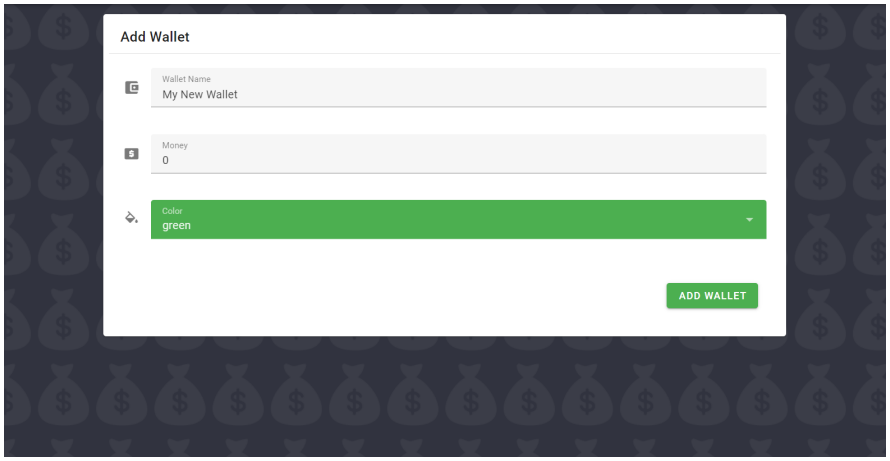

Figure 17: Screen Add Wallet

0.3.3.3.3 Transazioni

0.3.3.3.3.1 Transactions Schermata di visualizzazione delle transazioni, accessibile mediante apposito link nella sidebar oppure cliccando su una card dei Wallets o delle Categories e, in questi casi, la pagina si apre con i filtri

prevalorizzati con l'elemento scelto. Le Transactions si presentano come card "orizzontali", attraverso le quali é possibile eliminare la transazione.

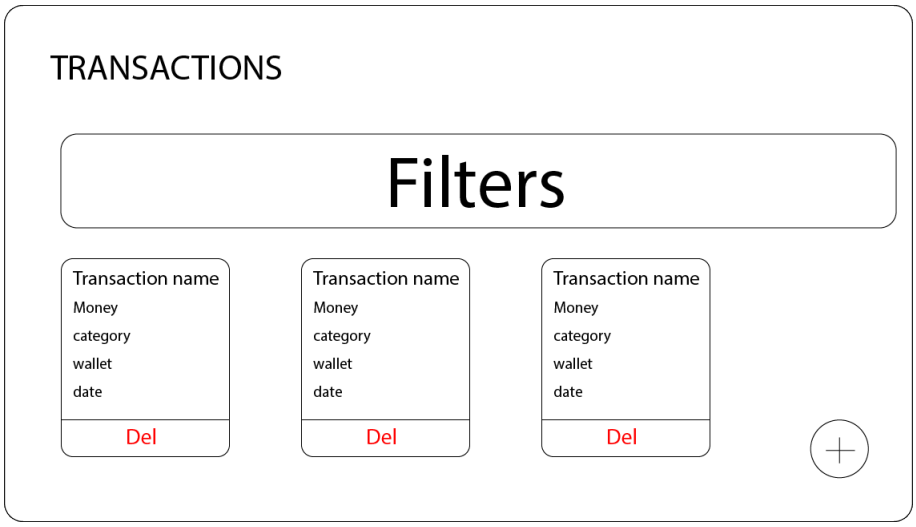

Figure 18: Mockup Transactions

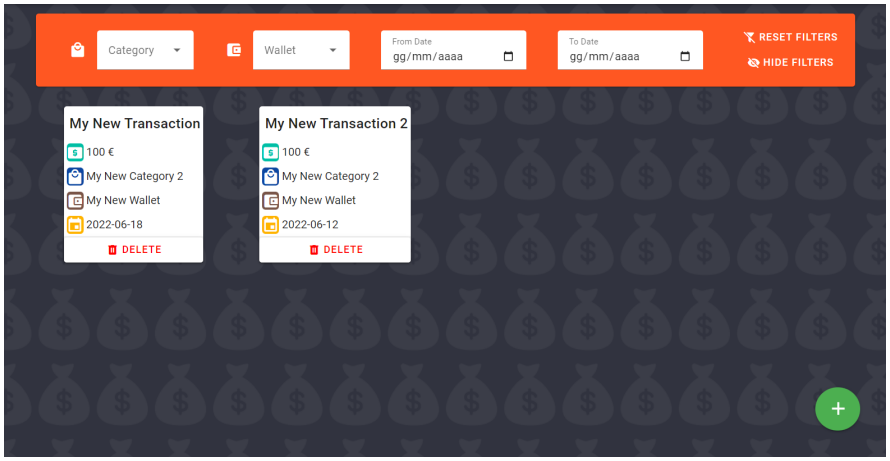

Figure 19: Screen Transactions

0.3.3.3.3.2 Add Transaction Schermata, accessibile mediante il floating button citato a 3.3.6, che permette di aggiungere una nuova transazione. Sono presenti due dropdown, relative rispettivamente ai Wallets e alle Cate-

gories, grazie alle quali é possibile scegliere gli estremi della transazione.

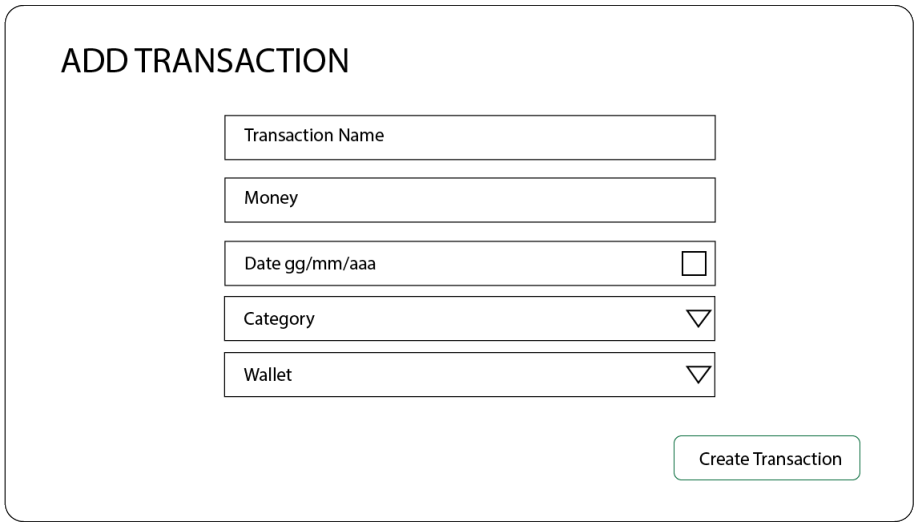

Figure 20: Mockup Add Transaction

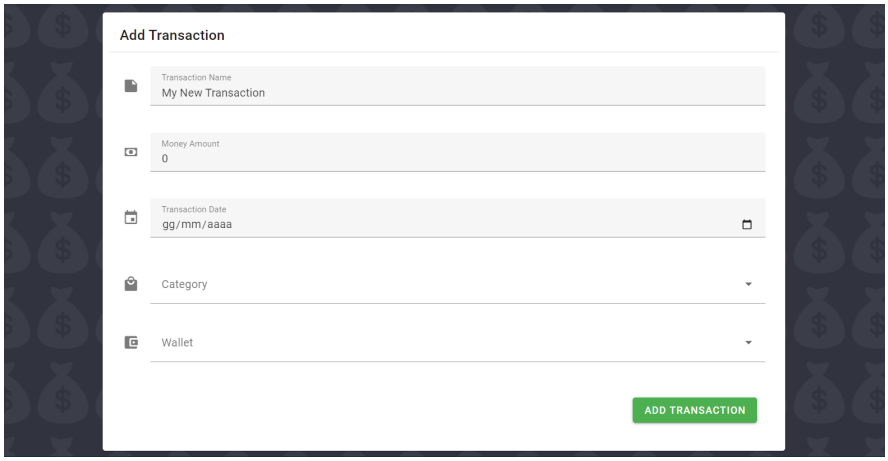

Figure 21: Screen Add Transaction

0.3.3.3.4 Account Schermata relativa alle informazioni dell'utente, raggiungibile dal relativo link all'interno della sidebar oppure cliccando l'immagine dell'utente nella toolbar in cima alla pagina. É possibile modificare tutte le informazioni dell'utente (Username, Immagine e Password) oppure eliminare

il profilo, ma é necessario inserire la password precedentemente inserita (come prevenzione).

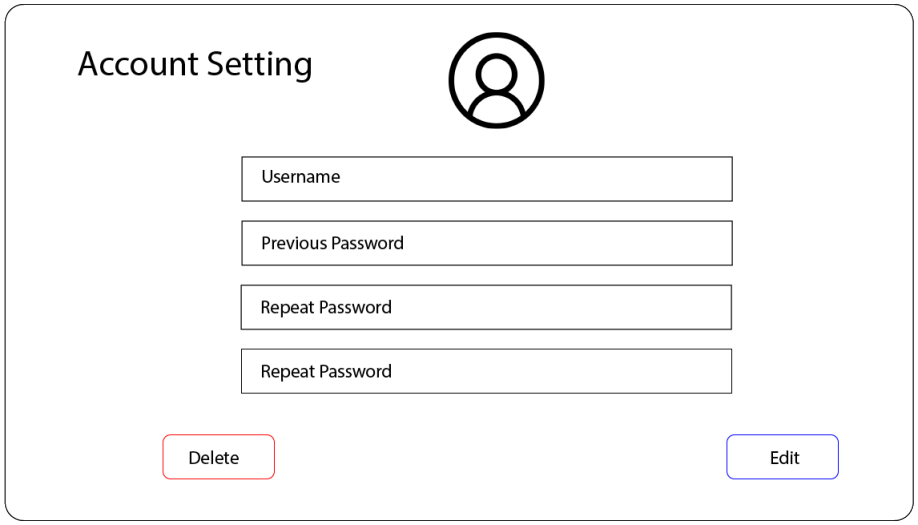

Figure 22: Mockup Account Settings

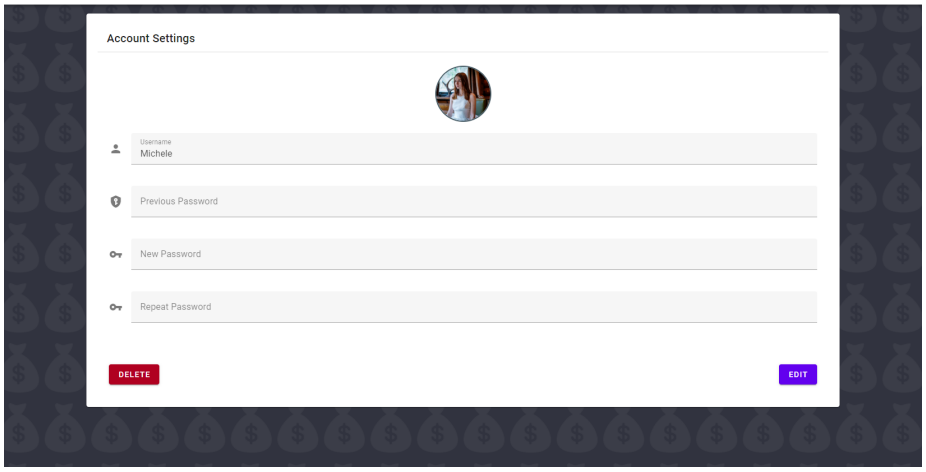

Figure 23: Screen Account Settings

0.3.4 Design di dettaglio

0.3.4.1 Backend

Il backend é stato sviluppato attraverso l'uso delle tecnologie Express, Mongoose e Socket.Io.

I controllers sono rappresentati da classi javascript che espongono API per interagire con il backend, nello specifico sono presenti operazioni di tipo CRUD (Create, Read, Update, Delete) e sfruttano I Models, ovvero classi javascript che esportano Mongoose Schemas (rappresentazioni della struttura delle entità). Nel progetto si é mantenuta ove possibile una struttura "1 a 1" per ogni entità, quindi sono presenti quattro file di controller, quattro di model e quattro di routes.

I file di routes utilizzano Express per fornire path e metodi specifici per richiamare le web Api esposte dai controller.

L'entry point del backend ("./server" nei file di progetto) é il file "app.js", che fa uso di Express, Mongoose, Socket.Io e dei file di route sopracitati. Socket.Io é utilizzato all'interno del progetto per rendere possibile la sincronizzazione in tempo reale delle operazioni effettuate dal medesimo utente su piú dispositivi/istanze dell'applicazione.

Le immagini utente vengono salvate all'interno del backend (nella directory './uploads' statica) grazie alla libreria node "Multer".

0.3.4.2 Frontend

Il frontend é stato sviluppato utilizzando il framework javascript Vue.Js 3.0, Vue-router e la libreria node Vuetify, che mette a disposizione componenti (discriminati dall'apposizione di "v-" nel tag html) nativamente reattivi e in stile material design.

La struttura dell progetto Frontend (“./client/src” nell’applicativo) si compone di sei parti principali:

1. Il file javascript “main.js”, entry point dell’applicazione e resposabile dell’instanziamento di Vue, Vuetify e router;
2. Il file Vue “App.vue”, vero e propria root dell’instanza di Vue;
3. Directory “views”, al cui interno risiedono tutte le schermate esposte ai punti 3.3.* e sviluppate come file Vue, in maniera quindi modulare;
4. Directory “router”, contenente il file “index.js” responsabile del routing di Vue-router;
5. Directory “components”, al cui interno sono presenti componenti riutilizzabili all’interno dell’applicazione, come floating button e message confirm, oltre ad un componente che gestisce le comunicazioni con socket.Io;
6. Directory “Api”, al cui interno sono presenti, in maniera speculare al backend, i quattro file javascript responsabili delle chiamate alle api delle entità.

0.3.5 Tartget User Analysis

Il target principale del sistema é un individuo che vuole tener traccia delle proprie spese e non gli basta o non puó o non vuole utilizzare le applicazioni bancarie.

Di seguito vengono caratterizzate diverse personas, differenti per esigenze e modalità di utilizzo del sistema

0.3.5.1 Personas: Giacomo

Giacomo é un ragazzo minorenne che non ha ancora un conto bancario ma gli piacerebbe tener traccia delle sue spese (date le limitate finanze) ed essere sempre a conoscenza dell'ammontare di soldi che possiede tra portafoglio e salvadanaio e carta prepagata.

Scenario d'uso: Giacomo riceve una paghetta di 50€ dai genitori e si compra un panino come merenda a scuola, per un totale di 2.50€.

- Giacomo accede alla piattaforma;
- Si iscrive registrando i propri dati e inserendo la foto scattata poco prima con il suo cane;
- Crea subito una category "Merenda a scuola" e un wallet "Portafoglio".
- Aggiunge 50€ al portafoglio.
- Crea una transazione "Panino con la cotoletta", con data odierna, valore di 2.50€, della categoria "Merenda a scuola" e wallet "Portafoglio".
- Dalle relative schermate nota che per la categoria "Merenda a scuola" ha speso per ora un totale di 2.50€ e nel portafoglio gli rimangono 47.50 €.

0.3.5.2 Personas: Francesca

Francesca é un'imprenditrice di successo e possiede molti conti in banche separate (che non comunicano tra loro). Francesca vorrebbe una soluzione per segnare tutte le spese che esegue e tenere traccia del conto con piú spese.

Scenario d'uso: Francesca vuole comprarsi una nuova auto sportiva, ma non si ricorda in quale conto ha abbastanza fondi al momento.

- Francesca accede alla piattaforma;
- Disponendo già delle credenziali di accesso, inserisce i propri username e password;
- Crea una nuova categoria “Automobili”;
- Controlla tra I wallet quale ha abbastanza denaro per l’auto dei suoi sogni e nota che é il conto corrente “Banca Verde”;
- Inserisce una transazione in data odierna, del valore *****€, categoria “Automobili” e wallet “Banca Verde”

0.4 Tecnologie

Il progetto, come sopra evidenziat, é stato sviluppato utilizzando il solution stack MEVN e le principali tecnologie utilizzate sono state:

- MongoDB e Mongoose
- Express
- Vue.js, Vue Router e Vuetify
- Node.js
- Socket.IO
- Docker

0.4.1 MongoDB e Mongoose

Lato backend, per gestire la persistenza dei dati, si é fatto uso di MongoDB come DBMS non relazionale e Mongoose.

MongoDB, anche grazie alla sua natura non relazionale, puó essere classificato come database di tipo NoSQL, molto piú permissivo in termine di struttura dei dati grazie all'utilizzo di documenti in stile JSON per la persistenza dei dati.

Mongoose é una libreria (scaricabile dal package manager di Node.js, ovvero "npm") orientata agli oggetti e in grado di creare una connessione tra il framework Express e il database MongoDB. Questa tecnologia mette a disposizione una funzione "Schema()" che produce oggetti su cui é possibile chiamare le classiche procedure di interffacciamento al db.

0.4.2 Express

Express é un server framework open source per applicazioni web sviluppate con Node.js, progettato per creare Web API e generare percorsi per le chiamate alle stesse.

0.4.3 Vue.js, Vue Router e Vuetify

Vue.js è un framework progressivo e monolitico JavaScript open-source per la creazione di interfacce utente e single-page applications in ambito web.
Con il termine single-page applications s'intende un'applicazione web che mette a disposizione utente una singola pagina, al fine di fornire un'esperienza simile ad un'applicazione desktop anche se si trova sul browser web. Questa soluzione è particolarmente performante dal punto di vista dell'utilizzatore poiché tutto il codice necessario (HTML, CSS, JavaScript, ecc...) è caricato all'apertura della pagina e, in risposta di azione dell'utente, vengono caricate e sostituite dinamicamente le risorse appropriate senza ricaricare la pagina web [Vue].
Vue.js è un framework in configurazione MVVM, ovvero Model-View-ViewModel, un patter software variante del "Presentation Model design" e che consiste nel delineamento di quattro componenti:

1. Model (o modello): implementazione del modello del dominio dell'applicazione;
2. View (o vista): aspetto e layout di ciò che l'utente vede a schermo;
3. View Model (o modello della vista): collegamento tra Model e View, responsabile della logica dell'interfaccia utente. Solitamente l'interazione con il Model avviene richiamando metodi dello stesso, che vengono poi inviati alla View per essere rappresentati a schermo correttamente;
4. Binder: meccanismo con il quale viewModel e view vengono sincronizzati, in modo da automatizzare l'aggiornamento dei dati nel momento del loro

cambiamento.

Dal punto di vista della programmazione, Vue.js fa largo utilizzo dei "classici" linguaggi HTML, CSS e JavaScript, introducendo alcuni meccanismi e metodi atti all'ottimizzazione sia dal punto di vista prestazionale che della riusabilità/brevità del codice scritto.

Vue Router è la componente fondamentale per permettere la creazione di single-page applications poiché si occupa, appunto, del routing e della navigazione delle pagine.

Vuetify é una libreria (scaricabile sempre da npm) che mette a disposizione componenti in stile "Material Design" per la creazione di interfacce utente con Vue.js (é simile a quanto accade con boostrap in relazione ai classici html e css).

0.4.4 Node.js

Node.js è un runtime system, ovvero un software atto a fornire i servizi necessari all'esecuzione di un programma e non facente parte del sistema operativo, distribuito dalla Node.js Foundation per l'esecuzione di codice JavaScript lato server, costruito sul motore JavaScript V8 del web browser Google Chrome. Si tratta di un software open source, quindi ogni detentore è abilitato alla modifica, studio, utilizzo e redistribuzione del suo codice sorgente ed è multipiattaforma.
Node possiede un'architettura orientata agli eventi, che rende possibile l'ottimizzazione del throughput e della scalabilità in tutte le applicazioni web con molte operazioni di input/output (come programmi di comunicazione real time o browser game, ma non solo) grazie ad un design che ne permette l'asincronicitá.
Come già detto, JavaScript è tipicamente interpretato dalle componenti del browser ed utilizzato prettamente lato client, ma grazie a Node.js è possi-

bile creare codice eseguibile lato server, al fine di produrre pagine web dinamiche prima che queste vengano inviate al browser dell'utente. Questo permette un'unificazione dello sviluppo di applicazioni web intorno a JavaScript (paradigma "JavaScript everywhere").

0.4.5 Socket.IO

Socket.IO é una libreria Javascript (scaricabile sempre da npm) orientata agli eventi, che permette la comunicazione in tempo reale di applicazioni web, mettendo a disposizione un "canale" bidirezionale tra client e server, dando quindi la possibilità a quest'ultimo di inviare dati anche in assenza di una chiamata lato frontend.

Meccanismo interessante (e utilizzato all'interno del progetto) sono le "rooms", ovvero gruppi di client all'interno dei quali é possibile inviare comunicazioni di tipo broadcast solo ai partecipanti.

0.4.6 Socket.IO

Docker é una piattaforma software che fornisce la possibilitá si creare, scaricare ed eseguire unitá di codice sotto forma di "immagini". Le immagini possono essere eseguite molteplici volte (anche in parallelo) e ogni istanza é chiamata "container".

All'interno del progetto é stata utilizzata la funzionalitá "docker compose" che permette di creare "bundle" di container che vengono eseguiti all'unisono pur proveniendo da immagini distinte (si tratta infatti della modalitá di deploy scelta per questa WebApp)

0.5 Codice - Aspetti Salienti

0.5.1 Backend

Gestione Immagini Account:

- ROUTES

```
    let storage = multer.diskStorage({
    destination: function (req, file, callback) {
        callback(null, './uploads')
    },
    filename: function (req, file, callback) {
        callback(null, file.fieldname + "_" + Date.now() + "_" + file.originalnar
    }
})

let upload = multer({
    storage: storage
}).single("image")
```

- CONTROLLER (API)

```
    static async updateAccount(req, res) {
        const id = req.params.id
        let new_image = ''
        if (req.file != null) {
            new_image = req.file.filename
            try {
                fs.unlinkSync('./uploads/' + req.body.old_image)
            } catch (error) {
                console.log(error)
```

```
        }
    } else {
        new_image = req.body.old_image
    }

    const newAccount = req.body
    newAccount.image = new_image

    try {
        await Account.findByIdAndUpdate(id, newAccount)
        res.status(200).json({ message: 'Account updated successfully' })
    } catch (error) {
        res.status(404).json({ message: error.message })
    }
}
```

0.5.2 Frontend

Parte significativa del codice frontend risiede nella gestione delle comunicazioni di Socket.IO, che permettono di sincronizzare in tempo reale dei dati visualizzati dall'utente su piú istanze dell'applicazione: Alla login, l'utente viene inserito in una room caratterizzata dal proprio username. Al conseguimento di un'operazione che comporta modifiche al db, la View emette un evento al parentNode (ApplicationPage.Vue), il quale, grazie al component custom "SocketIOMessage" invia un messaggio al backend; il backend invia quindi un messaggio broadcast a tutte le istanze dell'applicazione aperte dall'utente (tranne quella in cui é avvenuta l'operazione). Questo messaggio viene intercettato dall'ApplicationPage, che modifica la prop "update:Boolean" del router view,

scatenando l'aggiornamento della view aperta al momento. In questo modo, ad esempio, se vengono aggiornati i wallets, se l'utente ha l'Add Transaction aperta vedrà la tendina dei portafogli aggiornata.

In questo caso si omette la trascrizione del codice poiché le operazioni sopracitate si scatenano in piú punti dell'applicazione, e il codice sarebbe di difficile comprensione.

0.6 Test

Il corretto funzionamento dell'applicazione é stata testata sui principali Browser (Chrome, Firefox ed Edge), sopratutto per quanto riguarda la reattività del sistema.

Per testare le API e le funzionalità di Socket.IO é stata utilizzata l'applicazione desktop "Postman", che permette di simulare chiamate http, fornendo un'interfaccia grafica semplice e intuitiva.

Come controllo di tipo qualitativo sull'usabilità dell'interfaccia si é cercato di applicare le euristiche di Nielsen:

1. Visibilità dello stato del sistema: Il sistema presenta un'interfaccia per lo piú statica, nella quale si inseriscono eventuali messaggi (rappresentati da banner con sfondo colorato) ben riconoscibili;

2. Corrispondenza tra sistema e mondo reale: le terminologie utilizzate all'interno dell'applicazione sono poche (non si fa mai uso di sinonimi) e rimandano anche l'utente meno tecnico all'idea dell'entità corrispondente nella sua vita di tutti i giorni;

3. Dare all'utenza controllo e libertà: l'applicazione non presenta rindirizzamenti automatici e l'utente ha sempre la possibilità di spostarsi da una schermata all'altra, aiutato anche dalla toolbar, sempre presente all'interno della pagina;

4. Consistenza e standard: grazie all'utilizzo della libreria grafica Vuetify 3.0, il sistema presenta uno stile omogeneo in tutte le sue schermate;

5. Prevenzione dall'errore: il progetto é stato pensato per guidare l'utente in

ogni operazione esso debba intraprendere, intercettando eventuali errori/-mancanze e fornendo feedback istantanei.

6. Riconoscimento piú che ricordo: ogni elemento dell'interfaccia presenta icone (provenienti dalla libreria mdi di Google) standard e "familiari" anche agli utenti meno avvezzi alla tecnologia, in modo da far intuire anche ad un "New User" le operazioni che puó eseguire prima di procedere;

7. Flessibilità ed Efficienza: il sistema spesso fornisce molteplici strade per raggiungere la schermata desiderata; si noti come, per accedere alla schermata delle transazioni, sia possibile cliccare una categoria o un wallet oppure ancora interagire con la sidebar;

8. Estetica e progettazione minimalista: il minimalismo é garantito, sia lato interfaccia utente che di progettazione, grazie all'uso della libreria Vuetify e della struttura dei file Vue.

9. Aiuto Utente: come già detto, all'utente viene fornito sempre un feedback, sotto forma di messaggio scritto in linguaggio umano, in caso di errori/mancanze/comunicazione da parte dell'amministratore;

10. Documentazione: data la semplicità e la limitatezza delle azioni intraprendibili dall'utente, si é ritenuto che una documentazione di tipo testuale non fosse necessaria;

I test di usabilità sono stati effettuati su dieci individui con caratteristiche molto diverse per quanto riguarda la conoscenza e le modalità di fruizione della tecnologia, dai primi prototipi/mockup fino alla termine dello sviluppo, evidenziando componenti o azioni non intuitive al fine di migliorare l'esperienza utente.

0.7 Deployment

La messa in funzione dell'applicativo è possibile grazie alle funzionaità messe a disposizione da Docker, nello specifico si farà uso di Docker compose:

- Clonare il repository al link:Github - Moma
- Aprire un terminale nella root del progetto ed eseguire il comando: ' docker-compose up –build '
- Accedere ad un browser web all'indirizzo: Localhost:8080

0.8 Conclusioni

Il progetto, dato anche il limitato tempo di progettazione e sviluppo, si presenta sí completo rispetto all'idea iniziale, ma aperto dal punto di vista di implementazioni e aggiunte future, possibili anche grazie ad un'ingegnerizzazione modulare delle sue componenti.

Rimanendo sul tema delle possibili evoluzioni del sistema, é possibile, per esempio, dare la possibilità agli utenti di interagire tra loro con chat real time e transazioni "cross-user"; oppure ancora, il sistema potrà mettere a disposizione grafici e interfacce per fornire report istantanei dell'andamento per categoria o portafoglio.

Complessivamente mi ritengo soddisfatto, non tanto per l'applicativo in sé, ma per la possibilità di studiare e applicare nuove tecnologie, con tutte le sfide tipiche di un nuovo approccio ad una realtà. É stato inoltre di grande arricchimento l'aver sviluppato tutte le parti di un'applicativo web "completo".

Bibliography

[Node] sito ufficiale Node : https://nodejs.org/en/ [Vue] sito ufficiale VueJs : https://vuejs.org/ [Vuetify] sito ufficiale Vuetify : https://vuetifyjs.com/en/ [Mongoose] sito ufficiale Mongoose : https://mongoosejs.com/. [Socket.IO] sito ufficiale Socket.IO : https://socket.io/. [Docker] sito ufficiale Docker : [Nielsen] Jakob Nielsen. Usability Engineering. (English). San Diego: Academic Press, 1994.