
Progetto Ingegneria del Software

CashFlow



Matteo Soldini, Paolo Zanotti, Samuel Gherardi

Obiettivo

Abbiamo deciso di realizzare questo progetto in quanto sentiamo l'esigenza di tenere traccia delle spese fatte giornalmente e di pianificare la gestione delle entrate per il futuro (budget e obiettivi) in modo facile e veloce.

Le applicazioni attualmente sul mercato non soddisfano completamente le nostre necessità, abbiamo quindi pensato di sviluppare una **applicazione web** che permetta di **gestire le entrate ed uscite personali**.

Criteri da rispettare:

1. Ben documentata
2. Tecnologie più moderne
3. Best practices sviluppo web

Difficoltà incontrate

⚠ Nessuna difficoltà, se non durante il fetch delle API lato frontend (con JWT) causa tecnologia nuova per il team.

✅ Siamo riusciti a coordinarci senza alcun tipo di problema



Paradigma di programmazione e tools

- **Paradigmi**
 - Programmazione: a oggetti
 - Modellazione: UML
- **Linguaggi di programmazione**
 - FRONTEND: TypeScript (NextJS)
 - BACKEND: Python (Django)
- **Tools**
 - Github
 - Postman API, Swagger
 - Google Meet
 - VSCode

Software configuration management

Tutto il lavoro viene salvato sulla piattaforma Github, in condivisione con i membri del team.

- **Struttura**
 - 3 cartelle: docs, src/frontend e src/backend
- **Branch**
 - Seguito l'approccio *Git flow* (main, develop, feature/abc)
- **Issue**
 - Tramite la board Kanban per gestire i task e gli issue (*TO DO*, *IN PROGRESS* e *DONE*)



Software life cycle

Per il processo di sviluppo del software il team ha deciso di utilizzare un approccio di tipo **Agile**.

1. *I lavori vengono assegnati in base alle abilità*
2. *Suddivisione equa dei ruoli decisionali*
3. *Pair programming*
4. *Il team è propenso al cambiamento*
5. *Model Driven Architecture (MDA)*

Requisiti

Abbiamo deciso di suddividere i diversi requisiti secondo il modello **MoSCoW**.

- **Funzionali**

- Accesso degli utenti tramite e-mail e password.
- Ogni utente può visualizzare l'andamento delle sue spese/guadagni sulla base di grafici chiari ed esplicativi.

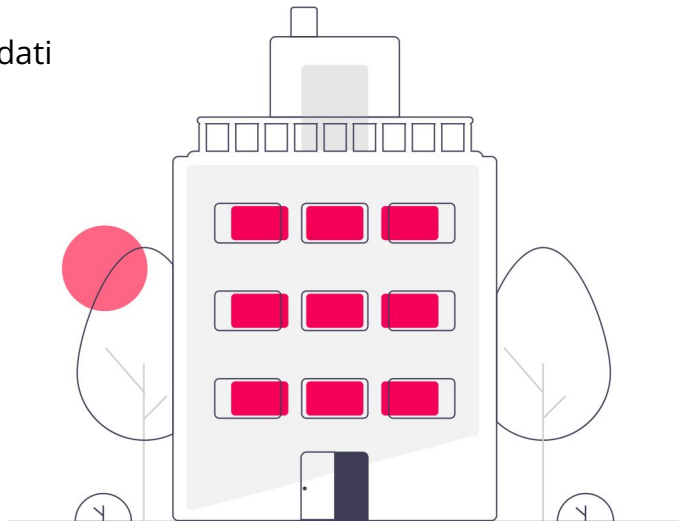
- **Non funzionali**

- La password degli utenti deve essere memorizzata tramite cifratura hash a 256 bit (SHA-256).
- Il sistema viene progettato per poter utilizzato sia da personal computer che da smartphone.

Architettura

L'architettura del nostro software è basata sul pattern **MVC** (Model View Controller).

- **Model**
 - Front-end | Gestione di tutte le attività relative ai dati
- **View**
 - Back-end | Visualizzare i dati dal Model
- **Controller**
 - Back-end | Manipolare il modello della vista



Design pattern

Durante lo sviluppo del codice il team ha deciso di utilizzare i design pattern **Observer** e **Adapter**.

- Observer
 - Applicato attraverso lo strumento React degli **Hooks**.
- Adapter
 - Realizzato grazie allo strumento **Serializer** di Django.

Software quality

Sono stati seguiti i principali parametri di **McCall**:

- **Operatività del software**

- *Correttezza*
- *Affidabilità*
- *Efficienza*
- *Integrità*
- *Usabilità*

- **Transizione verso nuovo ambiente:**

- *Portabilità*
- *Riusabilità*
- *Interoperabilità*

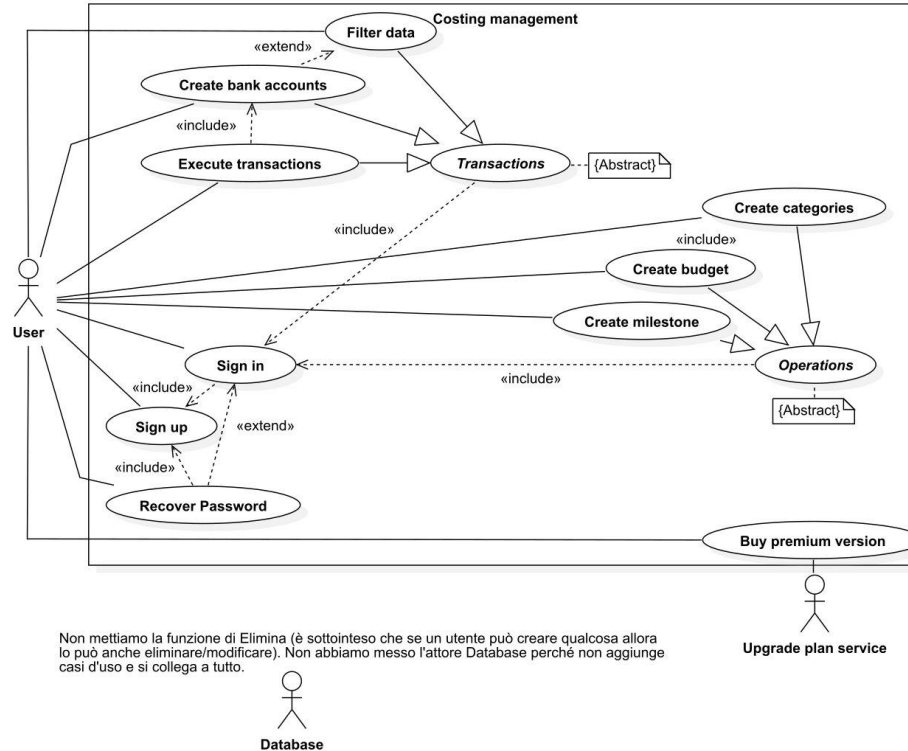
Modellazione

Abbiamo utilizzato i seguenti diagrammi UML

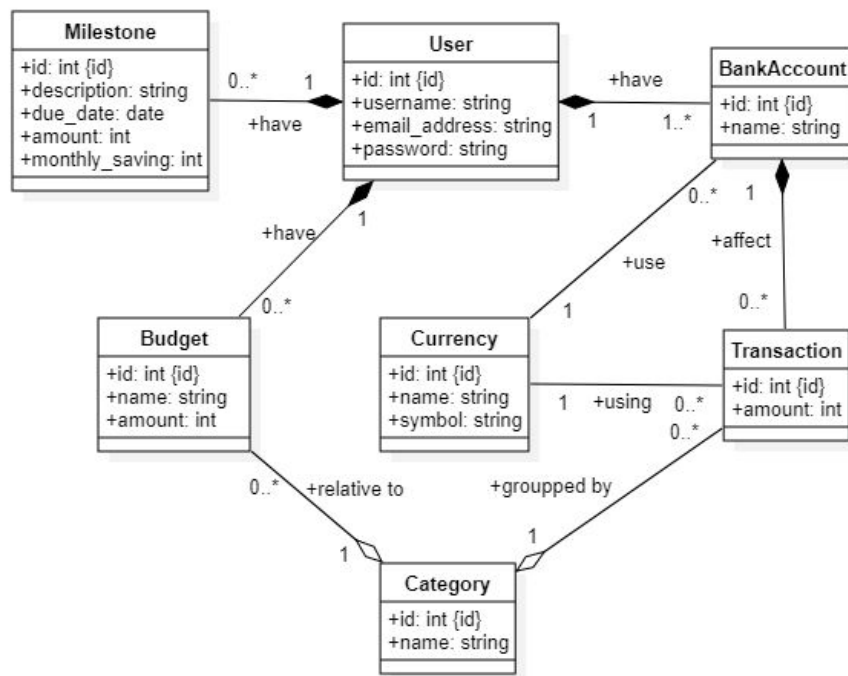
- Use case diagram
- State chart diagram
- Sequence diagram
- Class diagram
- Activity diagram

Inoltre per la progettazione del Database abbiamo utilizzato il modello ER.

Use Case Diagram

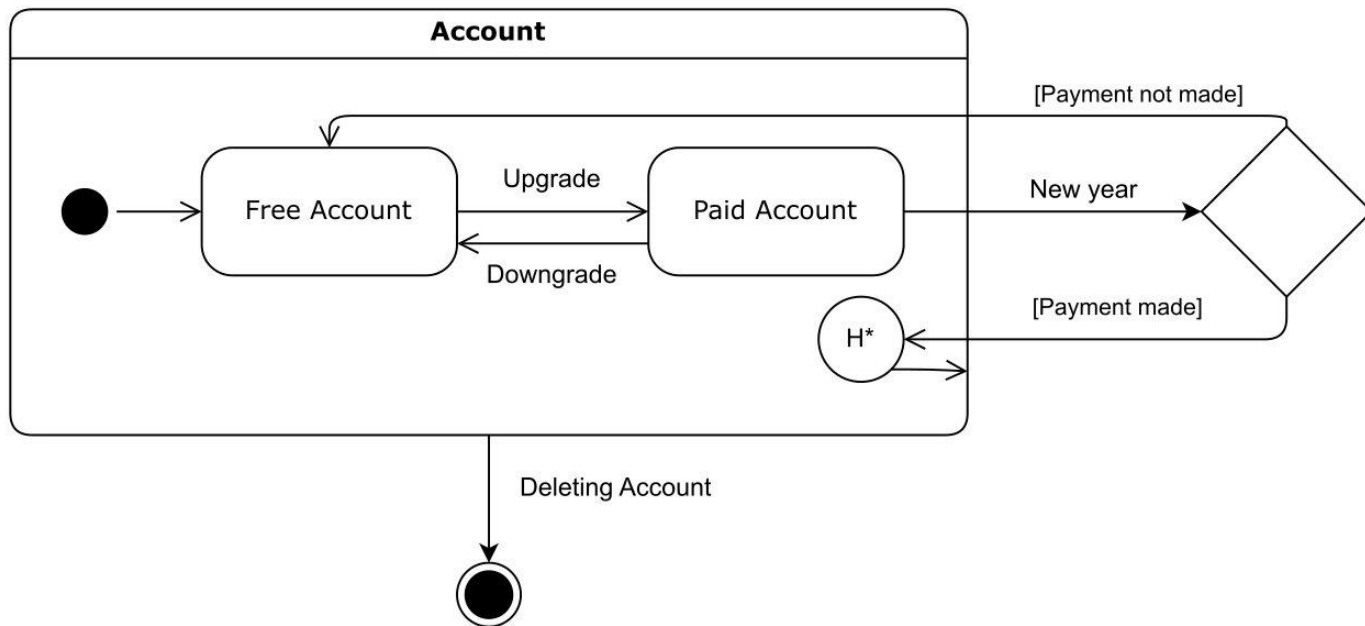


Class Diagram



State Chart Diagram

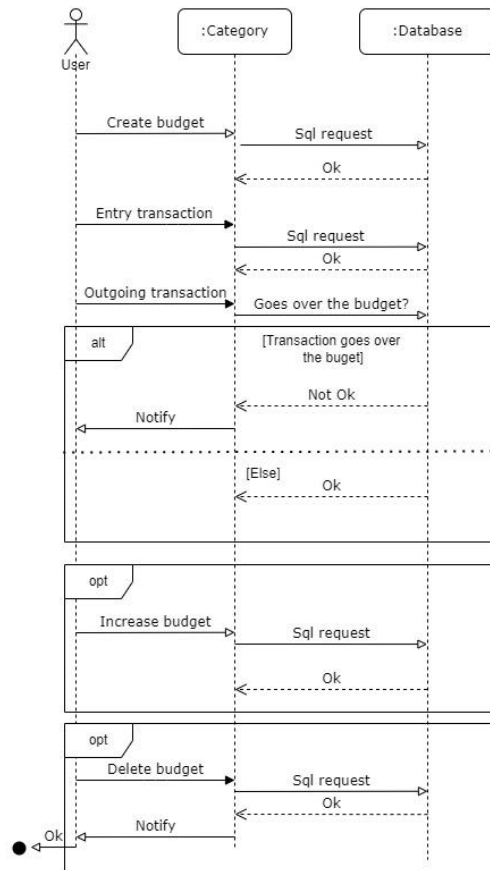
Meccanismo di creazione account e passaggio ad account premium



Sequence Diagram

Meccanismo di creazione del budget

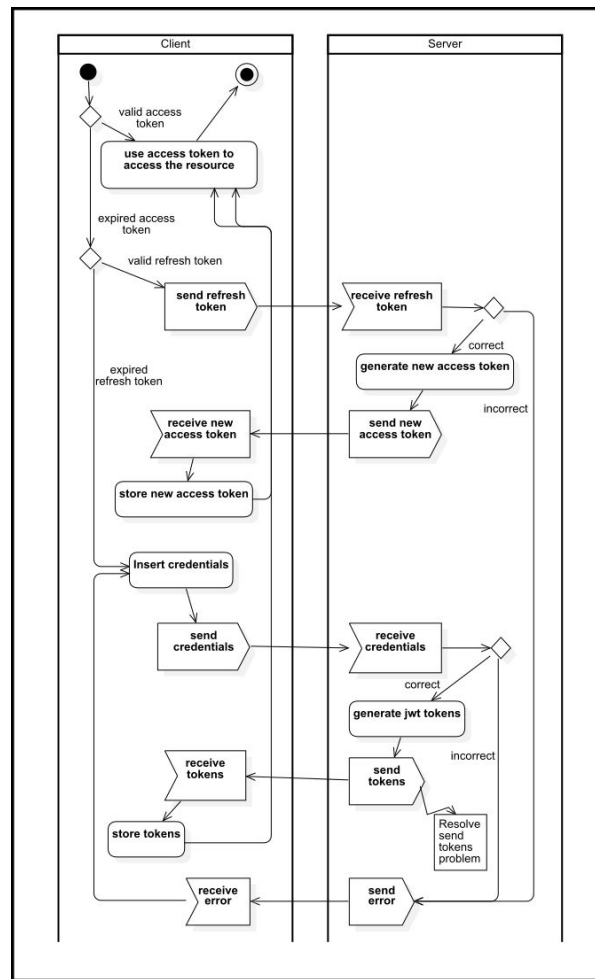
L'utente può creare un budget e successivamente effettuare transazioni. Il sistema richiede al database se tale spesa sfiora il budget e se così fosse allora viene inviata una notifica all'utente.



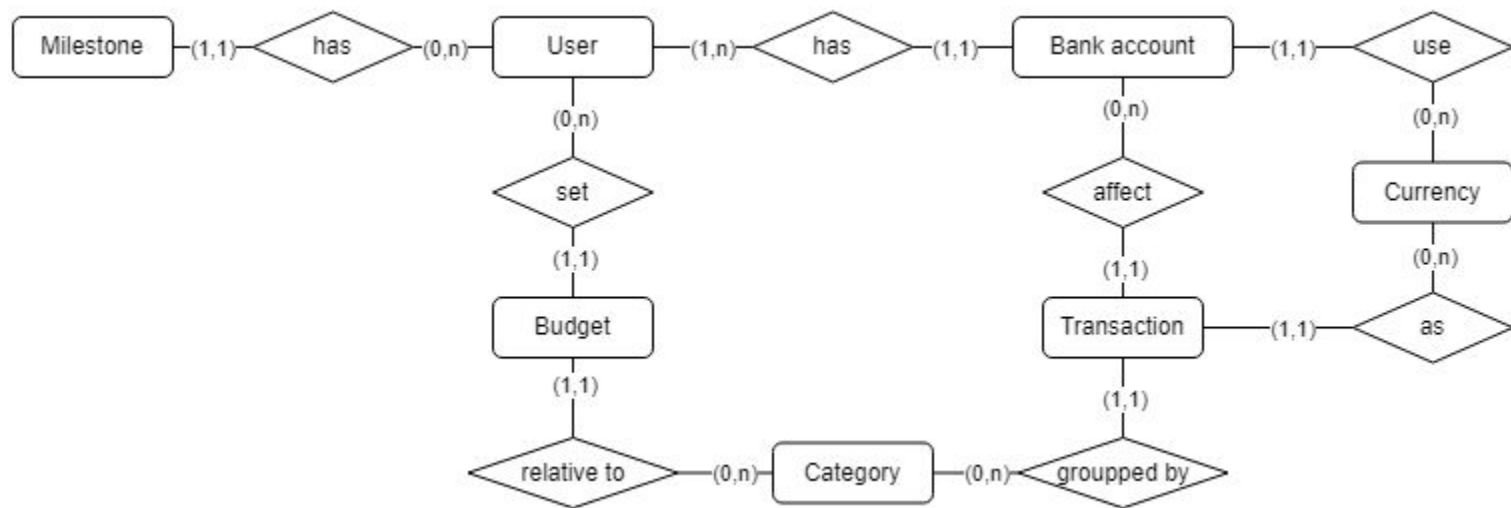
Activity Diagram

Meccanismo di accesso tramite token JWT

Se l'utente è presente nel database e la password è corretta allora vengono ritornati 2 token: *access* per autenticare le richieste alle api e *refresh* per rinnovare il token access.



ER Model



People Management

Il team ha scelto di seguire una struttura organizzativa di tipo **adhocratica**, ed è inoltre organizzato secondo il modello di sviluppo **SWAT**.

- **Vantaggi**

- Tranquillità e flessibilità nello sviluppo da parte dei partecipanti.
- Suddivisione dei task principalmente effettuata sulla base del piacere.

	Progetto DB	Back-end	Front-end	Testing
Matteo	v	v		
Paolo	v		v	
Samuel	v			v

Software Maintenance

Perfettiva e refactoring

- Migliorata la UI dopo uno studio condotto sui beta tester (informazioni non facilmente raggiungibili)

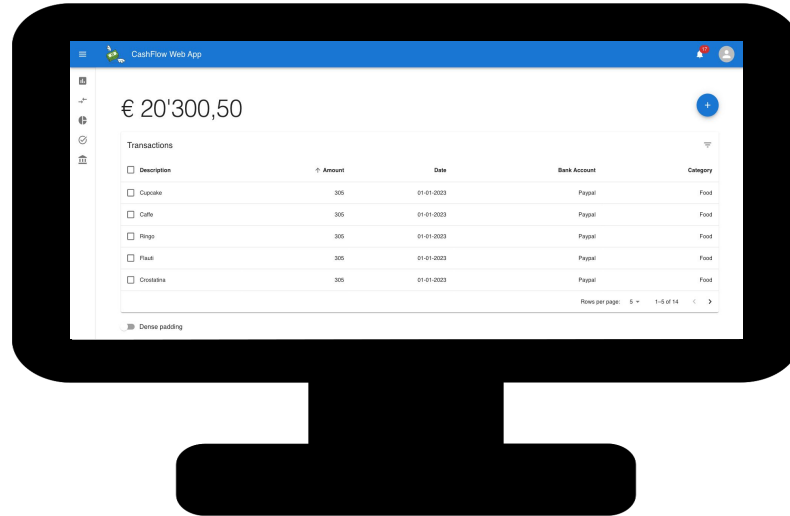
Preventiva

- Aggiornata la documentazione ad ogni nuova funzionalità
- Procedure importanti commentate

 **Correttiva:** progetto ancora in fase di sviluppo, quindi no guasti

 **Adattiva:** software si occupa di default delle interazioni con l'hardware e con il sistema operativo (dipendenze)

Demo



Testing

Test di unità backend:

→ modulo **TestCase** incluso in Django.

In modo particolare abbiamo analizzato la classe **Transactions** e abbiamo individuato i seguenti test da effettuare dal punto di vista delle api:



Descrizione del test	Risultato atteso
<i>Visualizzare tutte le transazioni</i>	✓ Successo
<i>Visualizzare una transazione che non appartiene all'utente</i>	✗ Fallito
<i>Creare una transazione</i>	✓ Successo
<i>Creare una transazione con un campo mancante</i>	✗ Fallito
<i>Creare una transazione con una valuta inesistente</i>	✗ Fallito
<i>Creare una transazione con account bancario che non appartiene all'utente</i>	✗ Fallito
<i>Creare una transazione con una categoria che non appartiene all'utente</i>	✗ Fallito
<i>Aggiornare un campo di una transazione</i>	✓ Successo
<i>Aggiornare un campo inesistente di una transazione</i>	✗ Fallito
<i>Aggiornare un campo di una transazione inesistente</i>	✗ Fallito
<i>Eliminare una transazione</i>	✓ Successo
<i>Eliminare una transazione di un altro utente</i>	✗ Fallito