Relazione Progetto Tecnologie Web

Penny Wise

http://tecweb.studenti.math.unipd.it/scaregna/

Referente: simone.caregnato@studenti.unipd.it

Credenziali di acesso:

Username: user@example.com

Password: user

Basso Leonardo	-	2042329
Caregnato Simone	-	2042884
Igbinedion Osamwonyi Eghosa Matteo	-	2042888
Rosso Carlo	-	2034293

Date: 20 giugno 2024

Sommario

Penny Wise è un'idea di Simone Caregnato, nata dalla passione per l'informatica. Questo progetto è stato sviluppato con l'obiettivo di creare due canvas non banali utilizzando esclusivamente JavaScript puro (vanilla JavaScript), come richiesto dai criteri del corso.

L'obiettivo principale del progetto è fornire un'interfaccia semplice e intuitiva per gestire e visualizzare alcune statistiche inerenti alle spese personali. Per rendere il progetto più interessante, è stata aggiunta la funzionalità di condivisione delle spese tra più utenti, permettendo la gestione delle spese comuni, come nel caso di coinquilini.

Le spese possono essere suddivise in categorie, denominate tag, per analizzare la distribuzione delle spese nelle diverse categorie in vari periodi di tempo, offrendo una visione chiara dell'evoluzione delle spese nel tempo.

Indice

1 Analisi dei requisiti

1.1 Target

Penny Wise è un servizio dedicato alla gestione delle finanze personali. Il target principale è costituito da individui di età compresa tra i 20 e i 40 anni, che necessitano di strumenti semplici per monitorare le proprie spese. Il servizio non include funzionalità complesse come il calcolo delle tasse o analisi avanzate sui dati, rendendolo meno adatto a un pubblico più anziano o esperto.
Inoltre, la funzionalità di condivisione delle spese tra più utenti è ideale per progetti informali, come organizzare viaggi o tenere traccia delle spese comuni in un appartamento. Il target è vario, e le scelte comunicative, strutturali e grafiche del servizio sono orientate verso l'intuitività e la facilità d'uso, permettendo sia ai visitatori occasionali di esplorare il sito con facilità, sia agli utenti esperti di trovare rapidamente le informazioni desiderate.

1.2 Attori

Gli attori principali dell'applicazione sono i seguenti:

- **Visitatore**: un utente generico, che non è riconosciuto dal sistema;
- **Utente Registrato**: un utente che ha effettuato il login, per cui è riconosciuto dal sistema.

1.3 Funzionalità

Penny Wise offre le seguenti funzionalità:

1. Visualizzazione delle informazioni generali dall'applicativo; disponibile al Visitatore e all'Utente Registrato;
2. Descrizione della storia che ha condotto allo sviluppo del progetto e introduzione dell persone che lo hanno sviluppato; disponibile al Visitatore e all'Utente Registrato;
3. Registrazione: creazione di un account valido all'interno del sistema e riconoscimento dell'utente da parte del sistema; disponibile al Visitatore;

4. Autenticazione: riconoscimento dell'utente da parte del sistema; disponibile al Visitatore che ha le credenziali di accesso;
5. Modifica delle credenziali dell'account: modifica di nome utente, email e/o password; disponibile all'Utente Registrato;
6. Creazione di un nuovo progetto: creazione di un progetto, ovvero di un contenitore di spese che è caratterizzato da un nome e da una descrizione; disponibile all'Utente Registrato;
7. Modifica di un progetto: modifica del nome e della descrizione del progetto; disponibile all'Utente Registrato che ha il progetto;
8. Eliminazione di un progetto: rimozione del progetto e di tutte le informazioni ad esso correlate dal sistema; disponibile all'Utente Registrato che ha creato il progetto;
9. Condivisione di un progetto: il sistema fornisce un link speciale, che permette a chiunque di visualizzare il progetto in questione; disponibile all'Utente Registrato che ha il progetto;
10. Partecipazione ad un progetto: aggiunta del progetto di un altro Utente Registrato tra i propri progetti; disponibile all'Utente Registrato che ha il link di condivisione del progetto;
11. Dissociazione da un progetto: rimozione del progetto di un altro Utente Registrato dai propri progetti; disponibile all'Utente Registrato che ha un il progetto;
12. Visualizzazione delle transazioni di un progetto; disponibile all'Utente Registrato per tutti i suoi progetti;
13. Visualizzazione dell'andamento delle spese di un progetto; disponibile all'Utente Registrato per tutti i suoi progetti;
14. Visualizzazione di un grafico a torta di spartizione delle spese per tag; disponibile all'Utente Registrato per tutti i suoi progetti;
15. Visualizzazione dei partecipanti ad un progetto: ovvero la lista dei nomi utente degli Utenti Registrati che hanno il progetto; disponibile all'Utente Registrato per tutti i suoi progetti;

16. Visualizzazione dei tag di un progetto: la lista dei tag che sono assegnabili alle transazioni; disponibile all'Utente Registrato per tutti i suoi progetti;
17. Filtraggio delle spese per periodo; disponibile all'Utente Registrato per tutti i suoi progetti;
18. Filtraggio delle spese per tag; disponibile all'Utente Registrato per tutti i suoi progetti;
19. Aggiunta di una transazione ad un progetto; disponibile all'Utente Registrato che ha il progetto;
20. Modifica di una transazione di un progetto; disponibile all'Utente Registrato che ha il progetto;
21. Eliminazione di una transazione da un progetto; disponibile all'Utente Registrato che ha il progetto;
22. Aggiunta di un tag in un progetto: i tag sono abbinabili alle spese, per categorizzarle; disponibile all'Utente Registrato che ha il progetto;
23. Modifica di un tag di un progetto; disponibile all'Utente Registrato che ha il progetto;
24. Eliminazione di un tag da un progetto; disponibile all'Utente Registrato che ha il progetto;

1.4 SEO

Innanzitutto, abbiamo immaginao il *search intent* dei nostri utenti. Abbiamo identificato le seguenti *query* che potrebbero essere utilizzate per cercare il nostro sito:

- *Penny Wise*;
- *Gestione spese personali*;
- *Applicazione per gestire le spese*;
- *Gestire le spese condivise*;

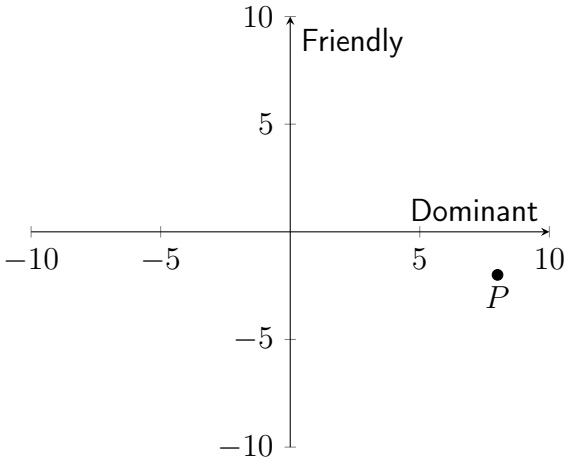
A partire da queste *query*, abbiamo scritto i tag `<title>`, `<meta name="description">` e `<meta name="keywords">` delle pagine del nostro sito. Si noti, che queste informazioni non sono aggiunte alle pagine che richiedono l'autenticazione, in quanto non sono indicizzate dai motori di ricerca e non sono accessibili ai visitatori non autenticati.
Inoltre, sono state adottate soluzioni tecniche, come la divisione tra la struttura, la presentazione ed il comportamento per ridurre il peso delle pagine e migliorare il *ranking* nei motori di ricerca. Le soluzioni tecniche adottate sono spiegate nel dettaglio nelle sezioni dedicate.

2 Progettazione

2.1 Design Persona

Partendo dalla risorsa consigliata nel corso (*Desining for Emotion* di Aaron Walter), abbiamo descritto la personalità di partenza per sviluppare il prodotto:

- **Brand name**: a questo punto il nome del brand è noto, ovvero *Penny Wise*;
- **Overview**: abbiamo scelto di usare come *mascotte* Paperon De Paperoni, infatti, dato che si tratta di un progetto universitario, abbiamo la possibilità di partire da un personaggio fantastico molto noto, senza doverci preoccupare di copyright;
- **Brand traits**:
  - Risparmiatore: la caratteristica che spicca del personaggio è il suo amore per il risparmio;
  - Familiare, ma non scontato (forse anche un po' scontato);
  - Laborioso, ma non *workahoolic*: è partito avendo solo un piccone e deve la sua fortuna al suo duro lavoro;
  - Scherzoso, ma non trasandato: si tratta del personaggio di un fumetto comunque legato all'infanzia di ognuno di noi;
  - Oculato: si tratta del papero pià ricco di tutta Paperopoli!
- **Personality map**:



- **Voice**:
  - Non viene raggiunto un obiettivo di risparmio: "Paperone è infastidito e incoraggia a migliorare";
  - Se raggiungi l'obiettivo di risparmio: "Paperone ti guarda dall'alto del suo gruzzolo di monete, e ti incoraggia a continuare";
  - Se il conto è in rosso: "Paperone è furioso e ti invita a rimediare!";
- **Copy examples**: potrebbe cambiare il logo del sito in base alla situazione del progetto, oppure potrebbe comparire una vignetta;
- **Visual lexicon**:
  - **Colori**: giallo e blu scuro;
  - **Contorni**: arrotondati;
  - **Ombre**: assenti;
  - **Font**: *Sans-Serif Arial*;
- **Engagement methods**:
  - **Feedback**: ad ogni azione corriponde una notifica, in modo tale che l'utente abbia sempre un riscontro;
  - **Design pulito e intuitivo**: l'utente deve essere in grado di capire cosa fare senza dover leggere alcun manuale;

– **Psicologia dei colori**: sono utilizzati dei colori accattivanti che richiamano il personaggio scelto;

2.2 Palette

I colori principali della palette di *Penny Wise* sono il giallo (`#F4DF57`) e il blu molto scuro (#202630): il giallo richiama il nome del prodotto, ovvero *penny*, che si riferisce alle monete, trasmette ottimismo, energia e vitalità; mentre il blu scuro è scelto per esaltare il giallo, senza che quest'ultimo debba essere aggressivo.

2.3 Accessibilità

L'accessibilità è un indice di qualità del sito, pertanto è stata fin da subito un proposito imprescindibile che ha guidato la fase di progettazione e le successive. Per quanto siamo incorsi in alcune difficoltà. Di seguito sono riportate le misure adottate per garantire un'esperienza di utilizzo ottimale per tutti gli utenti.

2.3.1 Orientamento dell'utente

L'utente deve essere in grado di costruire una mappa mentale durante l'esplorazione del sito, in questo modo evita situazioni di disorientamento o sovraccarico cognitivo. Per questo motivo, prima di sviluppare la gerarchia delle pagine, abbiamo creato una mappa concettuale che ci ha permesso di organizzare i contenuti in modo chiaro e coerente. Inoltre, sono adoperati accorgimenti per aiutare gli utenti che navigano le pagine mediante *screen-reader*.

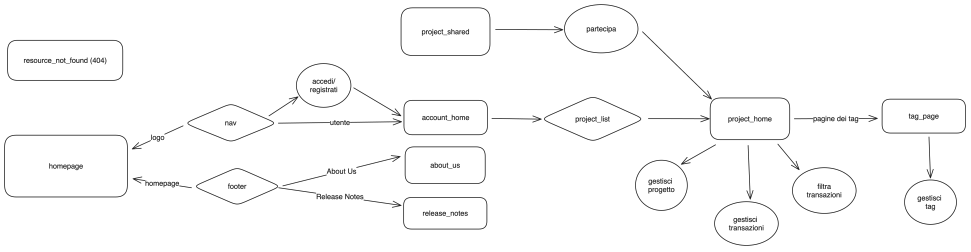


Figura 1: Mappa concettuale del sito

2.3.2 Colori

I colori sono stati scelti appositamente con un contrasto elevato tra loro in modo da garantire una buona leggibilità anche da parte di utenti con deficit parziale della vista, infatti è stato rispettato lo standard WCAG AA.

2.3.3 Responsive layout

Il sito è stato progettato per adattarsi a qualsiasi dispositivo, in modo da garantire un'esperienza di utilizzo ottimale sia da desktop che da mobile. Per questo motivo, sono stati adottati layout flessibili e fluidi, in modo da garantire una buona leggibilità e usabilità indipendentemente dalla dimensione dello schermo o dalle preferenze dell'utente.

2.4 Struttura del sito

Il sito è organizzato in una struttura gerarchica per conciliare facilità di utilizzo e organizzazione delle informazioni in modo ordinato e preciso. I contenuti sono suddivisi secondo uno schema organizzativo per argomento di seguito sono descritte le singole pagine:

- **Home**: si tratta della pagina di accesso al sito. Nell'area sicura sono presenti l'identità (logo e nome del servizio) e il menu. Scorrendo viene fornita un'introduzione al servizio, ovvero alle funzionalità offerte dal sito;
- **About Us**: contiene una breve descrizione delle motivazioni che hanno portato allo sviluppo del prodotto, oltre ad una breve introduzione delle persone che hanno realizzato *Penny Wise*;
- **Release Notes**: *timeline* dei progressi del progetto;
- **Account Home**: contiene le informazioni dell'Utente Registrato, inoltre da questa pagina è possibile accedere al form per modificare tali informazioni. Infine, è presente la lista dei progetti e le funzionalità di gestione di un progetto;
  - **Project Home**: contiene le informazioni di un progetto, ovvero il nome e la descrizione di esso. Mostra anche le transazioni contenute nel progetto, i partecipanti e i tag del progetto. Infine è mostrato un grafico interattivo che mostra l'andamento delle spese del progetto nel periodo di tempo desiderato;

    - **Project Cake**: contiene le medesime informazioni della pagina *Project Home*, però il grafico dell'andamento delle spese viene sostituito con un grafico a torta che mostra la proporzione dei tag rispetto alle spese. Si noti che anche in questo caso è possibile modificare il periodo di tempo considerato;
    - **Tag Page**: questa pagina mostra l'elenco dei tag associati ad un progetto e ne permette la gestione, ovvero la creazione, la modifica o l'eliminazione.

- **Project Shared**: si tratta della pagina per effettuare l'accesso ad un progetto condiviso. Vi si accede tramite il link di condivisione del progetto;

3 Realizzazione

3.1 Backend

Il backend è diviso in due parti principali:

- **Controller**: si occupa di gestire le richieste HTTP, che siano `POST` oppure `GET`. In particolare, si occupa di:
    1. Controllare che l'utente abbia l'autorizzazione per effettuare la richiesta;
    2. Controllare che i parametri della richiesta siano corretti;
    3. Sanitizzare i parametri della richiesta;
    4. Invocare i metodi corretti dal Model;
    5. Tornare la risposta al client, che sia un errore oppure un risultato.
- **Model**: si occupa di gestire la logica dell'applicazione. In particolare, si occupa di:
    1. Interrogare il database;
    2. Definire e applicare le regole di business;
    3. Tornare i risultati al Controller.

3.1.1 Controller

Il Controller è composto da due classi principali:

- **Router**: si occupa di individuare l'`Api` corretta, che si occupa di gestire la richiesta. In particolare, un `Router` è composto da una lista di `Api`. In realtà, come si può vedere in `index.php`, un `Router`, può essere composto anche da una lista di `Router`. Questo permette di creare un sistema di routing modulare, in cui ogni modulo ha il proprio `Router`;

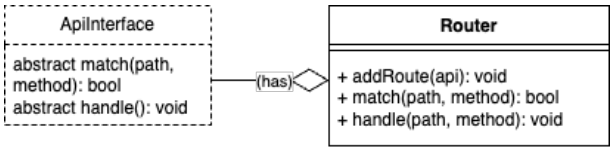


Figura 2: Struttura di un Router

- **Api**: si occupa di gestire una specifica richiesta. In particolare, una `Api` è composta da due metodi principali:
  - `match()`: che ritorna `true` se l'`Api` è in grado di gestire la richiesta, `false` altrimenti;
  - `handle()`: si occupa di invocare il metodo corretto del Model e tornare il risultato al client.

  Si noti che `Api` è una classe astratta, perché richiede l'implementazione del metodo `handle()`. Infatti, `Api` viene implementata da due classi: `JsonApi`, che si occupa di gestire le richieste `POST`; e `HtmlApi`, che si occupa di gestire le richieste `GET`.

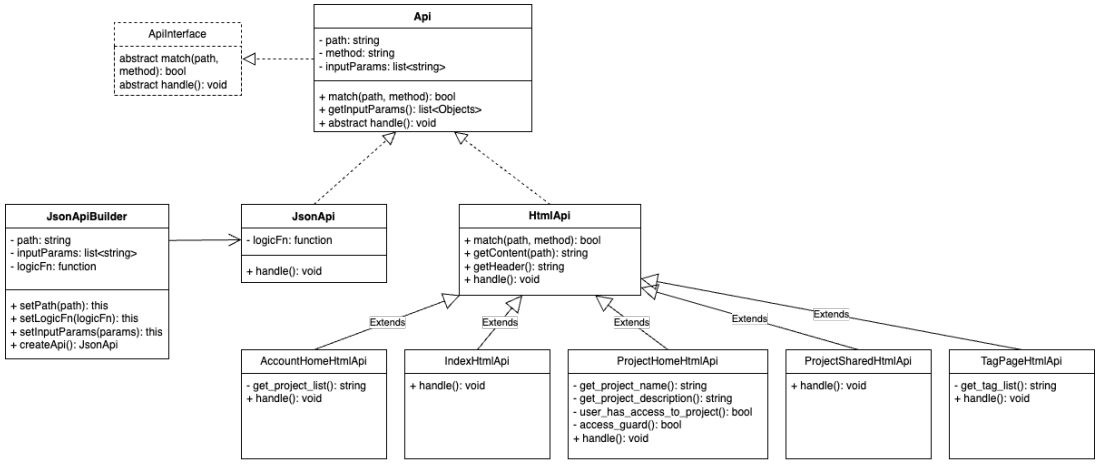


Figura 3: Gerarchia delle classi `Api`

In particolare, `.htaccess` si occupa di reindirizzare tutte le richieste a `index.php`, che utilizza un `Router` per gestire la richiesta. In questo modo, gli URL messi a disposizione dall'applicazione risultano essere più puliti e più facilmente gestibili, contribuendo alla *Search Engine Optimization*.

3.1.2 Model

Il Model è composto da due classi principali:

- **Database**: si occupa di gestire la connessione al database e mettere a disposizione i metodi per interrogarlo. Di seguito, si riportano i metodi principali:
  - `getInstance()`: ritorna l'istanza del database, infatti `Database` è un *Singleton*;
  - `query()`: esegue una query al database;
  - `prepareAndBindParams()`: prepara una query al database e. La preparazione della query è necessaria per evitare *SQL injection*;
- **DatabaseManager**: le estensioni di questa classe si occupano di fornire la *business logic* dell'applicazione. Sono omessi i parametri dei metodi, in quanto non sono rilevanti per la comprensione della struttura del Model. In particolare, le classi offrono i metodi di accesso al database.

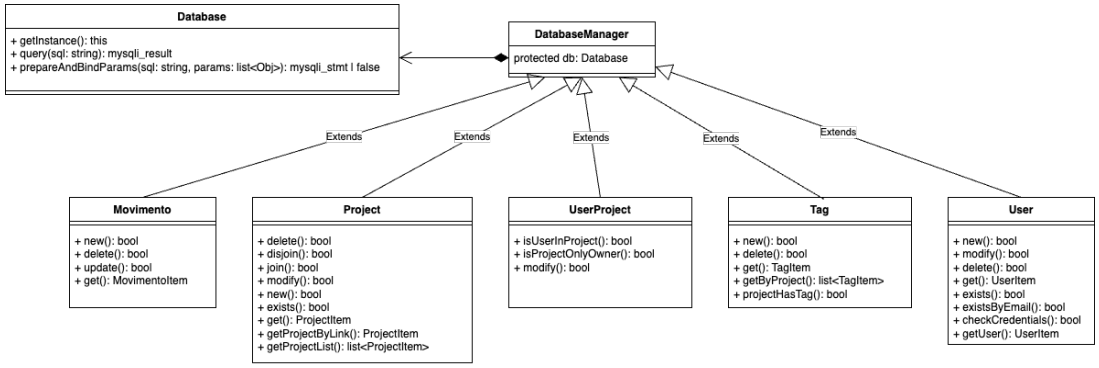


Figura 4: Struttura del `Model`

3.1.3 Database

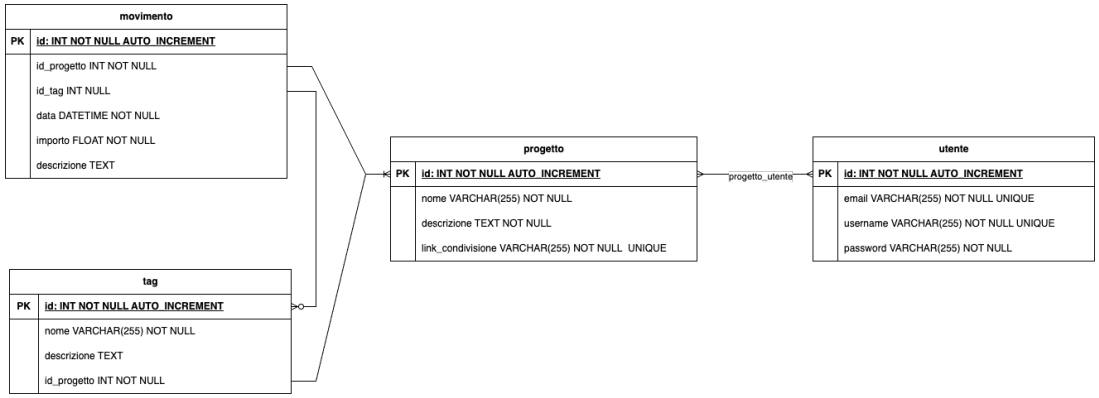


Figura 5: Diagramma del database

Si noti che per quanto riguarda la gestione del database, sarebbe possibile creare due *instance* `tag` per il medesimo `progetto`. In realtà, la *business logic* dell'applicazione non lo permette. Per stabilire la connessione con il database viene utilizzata l'estensione `MySQLi` di PHP.

3.2 Frontend

Il frontend è stato sviluppato utilizzando il pattern architetturale *Model View Controller*. Questa scelta è stata fatta per separare la struttura, la presentazione e il comportamento dell'applicazione, che permette di migliorare il mantenimento del codice, la scalabilità e la riusabilità. Non solo, ma migliora anche il *ranking* nei motori di ricerca, in quanto i motori di ricerca considerano il peso delle pagine web, e una struttura ben organizzata permette di ridurne le dimensioni.

In particolare, la scelta architetturale è evidenziata dalla gestione dei *feedback* e dell'interazione con i canvas, perché il resto dell'applicazione è gestito dal backend.

3.2.1 *Responsiveness*

Per ottenere un design *responsive* sono stati definiti quattro *breakpoint*:

- **Mobile**: per schermi con larghezza inferiore a 483px;
- **Tablet in modalità verticale**: per schermi con larghezza compresa tra 484px e 600px;
- **Tablet in modalità orizzontale o laptop piccoli**: per schermi con larghezza compreso tra 601px e 1280px;
- **Desktop**: per schermi con larghezza superiore a 1281px.

3.2.2 *Feedback*

Per gestire i *feedback* viene definita una funzione che intercetta tutte le chiamate `POST` al server, ovvero tutti gli `input` con `type="submit"`. In particolare, la funzione si occupa di:

1. Invocare la funzione `postRequest()`, che si occupa di inviare la richiesta al server, estraendo i dati dall'`event` passato;
2. Gestire la risposta del server, mostrando un messaggio di errore o di successo, ovvero viene aggiornata la *View* dal *Controller*.

3.2.3 Canvas

Per gestire i canvas è definita una classe `TransazioniSingleton`, che si occupa dei seguenti compiti:

- Mantenere la lista delle transazioni aggiornata;
- Ricordare il periodo di tempo selezionato;
- Ricordare i tag selezionati;

- Notificare le funzioni che si sono registrate per ricevere le transazioni non appena queste sono aggiornate.

Dunque la classe `TransazioniSingleton` utilizza due *desing pattern*: *Singleton* e *Observer*. In particolare, il *Singleton* permette di avere una sola istanza della classe, mentre l'*Observer* permette di notificare le funzioni che si sono registrate per ricevere le transazioni non appena queste sono aggiornate.
Le funzioni che si registrano per ricevere le transazioni sono le seguenti:

- `drawChart()`: si occupa di disegnare il canvas all'interno della pagina "Project Home";
- `drawCakeChart()`: si occupa di disegnare il canvas all'interno della pagina "Project Cake";
- `updateTransazioniTable()`: si occupa di aggiornare la tabella con le transazioni all'interno delle pagine "Project Home" e "Project Cake".

4 Organizzazione del lavoro

4.1 Basso Leonardo

- Definizione del database;
- Sviluppo del comportamento del client, ovvero di tutto il codice che viene scritto in JavaScript e che viene eseguito sul browser dell'utente;
- *Proof of concept* del Model;

4.2 Caregnato Simone

- Sviluppo del Model;
- Popolamento del database;
- Sviluppo di `CakeChart`, ovvero del codice JavaScript che si occupa di disegnare i grafici a torta;
- Testing delle funzionalità nel client;

4.3 Igbinedion Osamwonyi Eghosa

- Sviluppo della presentazione del client, ovvero tutto il codice CSS;
- Correzione dell'HTML;
- Testing dell'accessibilità del client;

4.4 Rosso Carlo

- Sviluppo della struttura del client, ovvero di tutto il codice HTML;
- Sviluppo di `LineChart`, ovvero del codice JavaScript che si occupa di disegnare i grafici a linee;
- Sviluppo del Controller;