

# Programmazione Web 2023/24

## Requisiti progetto d'esame

### Introduzione

Vogliamo realizzare una applicazione web che funga da gestore di un bilancio familiare. L'applicazione permetterà ad un utente registrato di caricare delle **spese**, **condividere** queste spese con altri utenti in parti non necessariamente uguali e vedere un **bilancio** di dare/avere. Le spese saranno composte da un **costo totale**, una **descrizione**, una **categoria**, un **elenco di utenti** con cui condividerla e la **quota** di condivisione. Sarà possibile visualizzare tutte le spese, oppure raggruppate per **anno** o **mese**. Le quote devono dare come somma il costo totale. Se un utente vuole fare un *rimborso* verso un altro utente può inserire una spesa con costo totale pari a zero, i soldi che cede con segno positivo e indicare i soldi ricevuti dall'altro utente con segno negativo. Gli utenti si registrano con uno **username** unico, **nome** e **cognome**. Si autenteranno mediante una **password**. Il progetto si comporrà della **parte server**, dove verranno memorizzati i dati e gestita la fase di autenticazione/autorizzazione, e della **parte client** che visualizzerà l'applicazione ed i dati.

### Funzioni principali

- **Visualizzazione bilancio personale:** riassunto della propria situazione finanziaria come differenza tra entrate ed uscite rispetto agli altri utenti.
- **Visualizzazione bilancio rispetto ad un utente:** un utente può vedere il dettaglio delle spese condivise con un altro utente.
- **Creazione di una nuova spesa:** l'utente può inserire una nuova spesa. Una volta inviata la spesa può essere modificata o cancellata. La spesa conterrà:
  - Data (impostata dall'utente)
  - Descrizione
  - Categoria
  - Costo totale
  - Elenco utenti
    - per ogni utente una quota (di default l'utente corrente è sempre presente con una quota pari al totale, se non specificato diversamente)

Per semplificare l'interfaccia dovrà essere possibile inserire l'utente tramite una ricerca (vedi sotto)

- **Visualizzazione delle spese:** ogni utente avrà una pagina da cui visualizzare, le proprie spese. Queste possono essere visibili tutte, oppure solo quelle di un anno o di un certo mese.
- **Ricerca spese:** un utente può effettuare la ricerca tra le sue spese. La ricerca può essere parziale e fatta su più campi (se cerco "Ca" deve trovare la spesa relativa al "Calcetto" o relativa al "Canone RAI")

- **Ricerca utenti:** deve essere possibile fare una ricerca di un utente per visualizzare il proprio bilancio nei suoi confronti. Valgono gli stessi principi di ricerca parziale definiti per le spese.

## Interfaccia REST

Il progetto prevede che sia realizzata una interfaccia REST. L'API da implementare è riportata di seguito.

Metodo	API	Descrizione
POST	/api/auth/signup	Registrazione di un nuovo utente
POST	/api/auth/signin	Login di un utente
GET	/api/budget/	Spese dell'utente loggato
GET	/api/budget/:year	Spese dell'utente loggato relative all'anno <i>year</i>
GET	/api/budget/:year/:month	Spese dell'utente loggato relative al mese <i>month</i> dell'anno <i>year</i>
GET	/api/budget/:year/:month/:id	Dettaglio della spesa <i>id</i> nel mese <i>month</i> dell'anno <i>year</i>
POST	/api/budget/:year/:month	Aggiunta di una spesa nel mese <i>month</i> dell'anno <i>year</i>
PUT	/api/budget/:year/:month/:id	Modifica della spesa <i>id</i> nel mese <i>month</i> dell'anno <i>year</i>
DELETE	/api/budget/:year/:month/:id	Rimozione della spesa <i>id</i> nel mese <i>month</i> dell'anno <i>year</i>
GET	/api/balance	Visualizzazione riassunto dare/avere dell'utente loggato
GET	/api/balance/:id	Visualizzazione del bilancio tra l'utente loggato e l'utente con id <i>userID</i>
GET	/api/budget/search?q=query	Cerca la spesa che matcha la stringa <i>query</i>
GET	/api/budget/whoami	Se autenticato, restituisce le informazioni sull'utente
GET	/api/users/search?q=query	Cerca l'utente che matcha la stringa <i>query</i>

## Consegna del progetto

Per consegnare una demo funzionante del progetto consiglio caldamente di usare [Docker](#). Questo strumento permette di realizzare dei *container*, degli ambienti virtuali che contengono tutto quello che serve per distribuire ed eseguire una applicazione.

**Usate nomi di container che siano univoci, possibilmente con il vostro cognome!**

Un esempio di configurazione di due container che usi node.js e mongodb:

```
version: "3"
services:
  app:
    container_name: app
```

```
build: .
command: nodemon --watch /usr/src/app -e js app.js
ports:
  - "3000:3000"
volumes:
  - ./app:/usr/src/app
links:
  - "mongo:mongosrv"
mongo:
  container_name: mongo
  image: mongo
  volumes:
    - ./data:/data/db
  ports:
    - '27017:27017'
```

Associato ad un Dockerfile per il container "app":

```
FROM node:latest
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
RUN npm install -g nodemon
COPY ./app/package.json /usr/src/app
RUN npm install
COPY ./app /usr/src/app
EXPOSE 3000
```

Potete anche usare altre soluzioni, non necessariamente MongoDB. Se preferite usare MySQL, fate pure, trovate i container con MySQL già pronto (va configurato).

Se dovete darmi una demo funzionante, allegate uno script che carichi i dati nel database.

Non inviatemi giga e giga di dati relativi al database. Soprattutto **non inviatemi i pacchetti di node!**