



ALMA MATER STUDIORUM  
UNIVERSITÀ DI BOLOGNA

## SELFIE

**Progetto di Tecnologie Web 2024/25**  
*Estensione 18-27*

Ivan De Simone - 0001069314

Payam Salarieh - 0001077673

Nicolò Tambini - 0001088816

# Introduzione

Il progetto SELFIE è stato ideato per semplificare la vita degli studenti universitari, offrendo strumenti per organizzare impegni, pianificare attività e supportare lo studio con funzionalità come un editor di testo e un timer pomodoro.

Accedendo alla webapp, l'utente viene accolto da una dashboard che fornisce una panoramica delle principali funzionalità, ciascuna accompagnata da una breve anteprima. Le sezioni principali includono:

- **Profilo utente**, per modificare le informazioni personali
- **Calendario**, per gestire eventi e attività
- **Note**, un editor di testo integrato
- **Lista attività**, per visualizzare tutte le attività pianificate
- **Pomodoro**, per organizzare il tempo di studio

SELFIE offre un sistema di autenticazione e gestione degli account semplice ma efficace, garantendo affidabilità ed efficienza.

Il progetto comprende un client sviluppato in React.js, che comunica con un backend basato su Node.js e un database MongoDB. L'applicazione è ospitata sui server del DISI ed è raggiungibile al seguente link: [SELFIE](#).

## Funzionalità

### Eventi

Il calendario consente di programmare eventi, contraddistinti da un titolo ed una descrizione. L'evento, che si può estendere per uno o più giorni, può avere orari di inizio e fine precisi oppure occupare l'intera giornata. È possibile impostare una regola di ricorrenza per ripetere l'evento indefinitamente oppure fino ad una scadenza, indicata tramite numero di ripetizioni o data limite. L'evento può essere geolocalizzato, nel qual caso sarà presente un collegamento verso Google Maps. È inoltre attivabile un servizio di notifiche, recapitate con un certo anticipo attraverso notifiche push o email.

Effettuando la sincronizzazione con Google, è possibile visualizzare nel calendario gli eventi importati da Google Calendar, i quali non possono essere modificati o eliminati.

## **Attività**

Il calendario permette di aggiungere i task, ovvero attività da completare entro una certa data. Anche queste sono caratterizzate da un titolo ed una descrizione, nonché dal giorno di scadenza. Le attività non completate entro la data pianificata sono segnalate come "in ritardo". Come per gli eventi, è possibile impostare delle notifiche, con la differenza che in caso di ritardo esse continueranno ad arrivare giornalmente fino al completamento dell'attività. Dalla dashboard è possibile accedere ad una sezione che mostra le attività in formato lista, suddividendole in completate e non.

## **Note**

L'applicazione include un editor di testo per creare, modificare ed eliminare note. Ciascuna nota ha un titolo e una serie di categorie. Il contenuto supporta la sintassi markdown e può essere di lunghezza arbitraria. Le note salvate sono visualizzabili in un elenco ordinabile per data di creazione, titolo, lunghezza o ultima modifica. Ogni nota offre una breve anteprima e opzioni per copiare, modificare, duplicare, scaricare o eliminare il contenuto. È possibile effettuare ricerche testuali all'interno delle note.

## **Pomodoro**

Il timer pomodoro aiuta a gestire lo studio alternando sessioni di lavoro e pause, ripetute per un numero definito di cicli. La sezione dedicata consente di avviare, resettare, riavviare o saltare sessioni, con un'animazione di sfondo che si adatta al timer. Dalle impostazioni si può sfruttare un meccanismo per calcolare cicli di durata ottimale a partire dai minuti disponibili. È possibile pianificare sessioni pomodoro come attività nel calendario, con notifiche e gestione dei ritardi.

## **Time Machine**

La Time Machine è un servizio aggiuntivo che permette lo spostamento avanti e indietro nel tempo per testare le funzionalità temporali. Una modifica temporale provoca l'aggiornamento immediato di tutte le visualizzazioni. Il tempo si può facilmente allineare al valore di sistema.

# Dettagli implementativi

## UI

L'interfaccia grafica è realizzata sfruttando i componenti della libreria [HeroUI](#), i quali implementano tutti gli standard di accessibilità attuali, validano l'input dei form e attuano un comportamento responsivo efficace. Tali componenti sono stati personalizzati tramite il framework CSS [Tailwind](#), il quale fornisce stili comodamente applicabili inline.

Un problema noto della libreria HeroUI è l'obbligo del formato data MM/dd/yyyy nei componenti di date-picking. Sul repository GitHub è aperto un issue al riguardo, come specificato anche nella documentazione ufficiale.

## Profilo

Il database memorizza gli account degli utenti, coloro i quali forniscono nome e cognome, username, password, email ed eventualmente data di nascita. Nel momento in cui si aggiorna la data di nascita viene creato automaticamente un evento di compleanno con ricorrenza annuale e scadenza indefinita.

L'autenticazione dell'utente è gestita tramite il middleware [Passport](#) utilizzando la strategia JWT con scadenza di 24 ore. Il salvataggio della password e la sua trasmissione da client a server è protetta da crittografia. Lato client l'autenticazione viene messa in atto da un context, il quale fornisce verifica dell'identità, login e logout.

La foto profilo dell'utente viene memorizzata tramite il middleware [Multer](#) sul server, dove è presente anche un'immagine default per gli utenti che non l'hanno caricata.

Dalla pagina del profilo personale è possibile effettuare il collegamento con il proprio account Google, il quale consente l'accesso agli eventi di Google Calendar tramite un token OAuth2 (che verrà poi salvato nell'account SELFIE).

Se un profilo utente viene eliminato, vengono eliminate anche tutte le risorse ad esso associate.

## Calendario

La visualizzazione del calendario utilizza la libreria [FullCalendar](#), che permette una gestione comoda degli eventi. Tramite il plugin di FullCalendar sul client ed il pacchetto [rrule](#) sul server, vengono gestite le regole di ricorrenza secondo lo standard iCalendar. Per quel che concerne la geolocalizzazione degli eventi, viene utilizzata l'API [LocationIQ](#) per il geocoding. Se la posizione calcolata viene approvata dall'utente, comparirà un bottone per aprirla su Google Maps.

Per quanto riguarda le attività, sono per default di colore ambrato. In caso di completamento il colore diventerà verde. Contrariamente, in caso di ritardo il colore diventerà rosso. La stessa paletta si applica alla lista di attività visualizzabile separatamente. Le attività si possono segnare come completate sia dal calendario che dall'elenco, in quanto il componente utilizzato alla base è lo stesso.

## Notifiche

Il sistema di notifica utilizza le WebPush API per le notifiche in-app ed il pacchetto [nodemailer](#) per le notifiche via mail. Entrambi per essere attivati necessitano del permesso dell'utente, la cui preferenza resta salvata all'interno del profilo.

Questo servizio è implementato tramite due demoni che vengono eseguiti ogni 60 secondi: ciascuna esecuzione recupera dal db tutti gli eventi/attività aenti delle notifiche impostate, verifica se è il momento di mandare la notifica (orario esatto al minuto) e nel caso richiama le funzioni per recapitarla.

È presente un demone aggiuntivo per notificare le attività in ritardo, eseguito ogni 180 secondi. Tale demone prende dal db tutte le attività aenti notifiche impostate, dopodiché per le sole attività in ritardo da almeno un giorno verifica il timestamp dell'ultima notifica. Se è passato almeno un giorno dall'ultima notifica, ne recapita una nuova calibrata in base al ritardo. Se l'utente viaggia indietro nel tempo, tutte le attività in ritardo vengono notificate (se passato almeno un giorno).

## Note

Le note sono memorizzate come semplice testo all'interno del database. È posta particolare attenzione al salvataggio delle categorie, il cui input viene ripulito dal client prima dell'invio al server. Per quel che concerne le date di creazione e modifica, sono interamente gestite dal server al momento delle operazioni. La visualizzazione in markdown è stata implementata usando il pacchetto [marked](#), seguendo la sintassi classica di GitHub.

## Pomodoro

Il timer pomodoro è realizzato interamente in vanilla JavaScript. Essendo indipendente, è integrato nell'applicazione tramite un iFrame. Lo sfondo della pagina segue gradualmente l'avanzamento del timer, mostrando un'animazione CSS che si adatta al momento di studio oppure pausa.

È possibile pianificare la sessione pomodoro attraverso un collegamento con il calendario, che crea un'attività legata al pomodoro definito. Il pomodoro comunica con il resto dell'applicazione React tramite il local storage.

## Time Machine

La Time Machine è implementata lato server tramite un offset in millisecondi rispetto al tempo di sistema. L'offset è salvato nell'account dell'utente e tramite due route è possibile recuperarlo ed impostarlo.

Lato client il servizio è implementato attraverso un context che mantiene il riferimento temporale, aggiornandolo ogni minuto. Tale contesto permette di recuperare, impostare o resettare il tempo in vigore per l'utente.

Tutte le annotazioni temporali all'interno dell'applicazione fanno riferimento al tempo specifico di ogni utente.

## Team

Il gruppo di lavoro è formato da tre componenti.

Nome e cognome	Matricola	Email
Ivan De Simone	0001069314	<a href="mailto:ivan.desimone@studio.unibo.it">ivan.desimone@studio.unibo.it</a>
Payam Salarieh	0001077673	<a href="mailto:payam.salarieh@studio.unibo.it">payam.salarieh@studio.unibo.it</a>
Nicolò Tambini	0001088816	<a href="mailto:nicolo.tambini@studio.unibo.it">nicolo.tambini@studio.unibo.it</a>

Ogni componente del gruppo ha apportato un contributo personale nelle seguenti aree:

- Ivan De Simone: profilo personale, dashboard, notifiche, UI, Time Machine
- Payam Salarieh: eventi, attività, note
- Nicolò Tambini: timer pomodoro

# Intelligenza Artificiale

Durante lo sviluppo del progetto è stato fatto uso di generative AI come copilota, principalmente al fine di generare lo scheletro di codice complesso, rifinito e modificato manualmente, oppure di troubleshooting, in caso di errori nascosti particolarmente difficoltosi da risolvere. Di seguito un esempio di prompt.

## Prompt a Claude:

why after the authentication flow with google, that goes alright, i get a cors error at the moment of the response to the frontend?

this is the code:

```
router.put('/google', auth, async (req, res) => {
  try {
    const user = await User.findOne({ username: req.user.username })
    const client = await authorize()
    console.log(client)
    if (!client) return res.status(400).send('Authentication failed')
    if (!user) return res.status(404).send(`No user found with username ${req.user.username}`)
    user.google = true
    await user.save()
    return res.json(user)
  } catch (err) {
    return res.status(500).send('Problems during Google authentication')
  }
})
```

this is the error:

Object { stack:

```
"AxiosError@http://localhost:3000/static/js/bundle.js:79775:18\nhandleError@http://localhost:3000/static/js/bundle.js:79133:14\n", message: "Network Error", name: "AxiosError", code: "ERR_NETWORK", config: {...}, request: XMLHttpRequest }
```

code: "ERR\_NETWORK"

## Risposta

The CORS error occurs because your backend server needs to be configured to accept requests from your frontend domain. Since your frontend is running on localhost:3000, you need to enable CORS for this origin.

Add CORS configuration to your Express server. Here's how:

1. First, install the cors middleware if you haven't already:

```
npm install cors
```

2. Then add it to your server setup. If you're using Express, add this near the top of your server file (usually app.js or index.js):

```
const express = require('express');
const cors = require('cors');
const app = express();

// Enable CORS for all routes
app.use(cors({
  origin: 'http://localhost:3000', // Your frontend URL
  credentials: true // If you're using cookies/sessions
}));
```

Or if you want to enable CORS just for specific routes:

```
const cors = require('cors');

router.put('/google', cors({
  origin: 'http://localhost:3000',
  credentials: true
}), auth, async (req, res) => {
  // Your existing code
});
```

Also, make sure your Axios request from the frontend includes credentials:

```
axios.put('/google', data, {
  withCredentials: true
});
```

This should resolve the CORS error. Let me know if you're still seeing issues!