PROGETTO SAW 2025

HAIR TIME

Natalia Ceccarini

598537

Introduzione

HairTime è una progressive web app sviluppata per un salone di parrucchieri, con l'obiettivo di digitalizzare e semplificare il processo di prenotazione degli appuntamenti. La piattaforma consente agli utenti di registrarsi, autenticarsi, gestire le proprie prenotazioni e lasciare recensioni sui servizi ricevuti, offrendo così un'esperienza completa e interattiva.

Dal punto di vista funzionale, l'app offre una serie di caratteristiche mirate a migliorare l'esperienza dell'utente e ottimizzare la gestione degli appuntamenti:

- **Registrazione e autenticazione**: gli utenti possono creare un account personale per accedere alle funzionalità riservate.
- **Logout**: possibilità di uscire in qualsiasi momento dal proprio account.
- **Visualizzazione dei servizi**: l'app mostra l'elenco completo dei servizi offerti dal salone, con la possibilità di prenotarli.
- **Visualizzazione delle recensioni**: l'utente può consultare le recensioni lasciate da altri clienti per orientarsi meglio nella scelta.
- **Prenotazione appuntamenti**: l'utente può selezionare un servizio, scegliere la data e la fascia oraria preferita per fissare un appuntamento.
- **Gestione notifiche**: è possibile iscriversi a un servizio di notifica che invia un promemoria un giorno prima dell'appuntamento; è prevista anche la possibilità di disiscriversi.
- **Area personale**: ogni utente ha accesso alla propria area riservata dove può visualizzare lo storico degli appuntamenti dell'anno e, se necessario, annullarli.
- **Inserimento recensioni**: al termine del servizio, l'utente può lasciare una valutazione espressa in stelle e un commento testuale opzionale.

Esecuzione

Avviare il Database con Docker

Assicurati di avere **Docker** installato. Una volta installato, esegui i seguenti comandi nel terminale:

Scarica l'immagine di Postgres

docker pull postgres:latest

Crea un volume per la persistenza dei dati

docker volume create postgresqldata

Avvia il container PostgreSQL

docker run -d --name postgres -e POSTGRES_PASSWORD=natalia -e POSTGRES_USER=natalia -e POSTGRES_DB=hairtimedb -p 5432:5432 -v postgresqldata:/var/lib/postgresql/data postgres:latest

Avviare il Server

Assicurati di avere **Node.js** installato.

Vai nella cartella server del progetto e poi esegui:

Installa Nodemon (per il reload automatico in sviluppo)

npm install --save-dev nodemon

Installa le dipendenze del progetto

npm install

Avvia il server in modalità sviluppo

npm run dev

Avviare il Client

Vai nella cartella del client ed esegui:

Installa le dipendenze

npm install

Costruisci il progetto per la produzione

npm run build

Avvia l'anteprima del progetto buildato

npx vite preview --host

Architettura del Sistema

L'architettura di *HairTime* si basa su un modello **client-server**, in cui il client (l'interfaccia utente della web app) comunica con un server backend attraverso richieste HTTP seguendo il paradigma **RESTful**.

Server

Il backend è sviluppato in **Node.js** con il framework **Express.js**. Si occupa della gestione della logica applicativa, dell'interazione con il database e dell'esposizione di API REST per tutte le operazioni principali dell'applicazione.

Il server gestisce le seguenti responsabilità:

- **Autenticazione e registrazione** (/auth) degli utenti. L'autenticazione utilizza un **Bearer Token JWT**, generato al login e inviato al client.
- **Gestione servizi** (/services): visualizzazione dei **servizi offerti dal salone**
- **Gestione delle prenotazioni** (/appointments): creazione, visualizzazione, cancellazione
- **Gestione delle recensioni** (/reviews): inserimento e consultazione
- **Servizio di notifiche push** (/push): iscrizione e disiscrizione dagli avvisi automatici pre-appuntamento
- **Job scheduler**: tramite uno script cron viene inviato un promemoria automatico agli utenti il giorno prima dell'appuntamento

Il backend utilizza **PostgreSQL** come sistema di gestione del database relazionale, dove sono memorizzate tutte le informazioni relative a utenti, appuntamenti, servizi e recensioni.

Database

Il database PostgreSQL viene inizializzato all'avvio del server, con la creazione di tabelle e popolamento dei servizi. La connessione a PostgreSQL avviene tramite parametri salvati in **variabili d'ambiente**, letti tramite il modulo dotenv.

La struttura del database è composta dalle seguenti tabelle principali:

users: memorizza le informazioni degli utenti registrati.

reviews: contiene le recensioni lasciate dagli utenti.

services: elenca tutti i servizi offerti dal salone, con nome, costo e durata.

appointments: registra gli appuntamenti fissati dagli utenti, con data, orario e numero di telefono.

appointment_services: tabella di relazione molti-a-molti tra appuntamenti e servizi prenotati.

user_push_subscriptions: memorizza le sottoscrizioni alle notifiche push per ciascun utente.

Client

Il client dell'applicazione *HairTime* è una **Progressive Web App** sviluppata in **React**, Il progetto è configurato con **Vite** per migliorare le performance in fase di sviluppo e build.

Librerie principali

- **React**: per la gestione dell'interfaccia e del flusso applicativo.
- **React Router**: per la navigazione tra le pagine.
- **React Bootstrap**: per creare l'interfaccia.
- **AOS (Animate On Scroll)**: per aggiungere transizioni agli elementi durante lo scroll.
- **Axios**: per effettuare le chiamate HTTP verso il backend.

Struttura dell'app

Il componente principale App definisce le rotte e l'organizzazione generale dell'interfaccia. Include:

- Un **NavigationBar** sempre visibile in alto

- Un **footer** fisso in fondo
- Un sistema di **modal per login e registrazione**
- Rotte principali:
    - /: homepage con presentazione del salone, servizi offerti e recensioni generali
    - /appuntamenti: sezione di prenotazione, dove l'utente può scegliere servizi, data, ora e numero di telefono
    - /mie-prenotazioni: area personale dell'utente con lo storico degli appuntamenti, possibilità di cancellarli e inviare una recensione

All'avvio dell'applicazione viene visualizzata la **homepage**, che carica due tipologie di risorse:

- **Servizi offerti**
- **Recensioni degli utenti**

La gestione di queste risorse avviene tramite **custom hook** che implementano strategie di caching differenti in base alla natura dei dati.

Servizi – Strategia Cache First

I servizi rappresentano dati **statici**.
Per questo motivo utilizzo una **strategia cache-first**:

- Alla prima richiesta, i servizi vengono recuperati dal **server** e salvati nella **cache**.
- Alle richieste successive, i dati vengono letti direttamente dalla **cache**, evitando ulteriori chiamate API non necessarie.

Recensioni – Strategia Network First + Polling

Le recensioni, al contrario, sono date dinamiche e soggette a frequenti aggiornamenti da parte degli utenti.
Per questo applico una **strategia Network First** con meccanismo di **polling automatico**:

- L'app effettua una chiamata al **server ogni minuto** per ottenere recensioni sempre aggiornate.
- Se la connessione è assente, viene utilizzata la **cache locale** come fallback temporaneo, garantendo comunque una visualizzazione dei dati, seppur non aggiornati in tempo reale.

Questo sistema assicura che l'utente abbia sempre accesso a contenuti aggiornati quando online, ma non venga penalizzato se l'app è offline.

Gestione dell'autenticazione

L'app utilizza un **context di autenticazione** per gestire lo stato dell'utente. Questo context fornisce:

- Il **token JWT** dell'utente, necessario per effettuare chiamate API protette (lettura e modifica di risorse utente).
- Lo **username**, utilizzato ad esempio per salutarlo nella navigation bar.
- Uno **stato booleano** per sapere se l'utente è loggato (true/false).
- Uno **stato di caricamento** per gestire le operazioni asincrone iniziali.

Gestione persistente del login

Poiché il token ha una validità di **2 ore lato server**, viene salvato nel localStorage insieme a username e timestamp al momento del login.
In fase di avvio, l'app controlla se il token è ancora valido (cioè non sono passate 2 ore):

- Se sì, lo stato utente viene ripristinato.
- Se no, i dati vengono automaticamente rimossi e l'utente sloggato.
  Questo meccanismo evita di dover ripetere il login a ogni ricaricamento della pagina.

Prenotazione degli appuntamenti

L'utente può accedere alla sezione dedicata alla **prenotazione degli appuntamenti**, dove dovrà compilare **quattro campi obbligatori** per completare la procedura.

1. Selezione dei servizi

La prima azione richiesta è la selezione dei **servizi desiderati** (es. taglio, piega, colore, ecc.).
L'elenco dei servizi viene recuperato utilizzando una strategia **cache-first**:

- Se i servizi sono già disponibili in cache, vengono utilizzati direttamente.
- In caso contrario, viene effettuata una **chiamata API** per recuperarli dal server.

2. Disponibilità

Una volta che l'utente ha selezionato uno o più servizi, viene effettuata una **chiamata API** per ottenere tutte le **date e fasce orarie disponibili** per quei servizi, nei **tre mesi successivi**.
Queste disponibilità tengono conto della durata combinata dei servizi scelti e degli slot liberi nel calendario del salone.

3. Scelta giorno e orario

L'utente può quindi:

- Selezionare il **giorno preferito**
- Scegliere una **fascia oraria** tra quelle disponibili

4. Inserimento del numero di telefono

Infine, l'utente inserisce il proprio **numero di telefono**, necessario per confermare e gestire l'appuntamento.

5. Conferma prenotazione

Una volta compilati tutti i campi, l'utente può **confermare la prenotazione**, che verrà inviata al server tramite una richiesta API.

Promemoria Appuntamenti – Gestione Sottoscrizione Notifiche

All'interno della sezione **prenotazione appuntamenti**, l'utente ha la possibilità di **attivare o disattivare un promemoria automatico**, che gli invierà una **notifica il giorno prima dell'appuntamento**.

Quando la pagina viene caricata:

Viene effettuata una **chiamata API** per recuperare **tutte le sottoscrizioni attive dell'utente**. Questo perché l'utente può essere iscritto al servizio di promemoria da **più dispositivi** o browser.

Il sistema verifica se **l'endpoint (device/browser corrente)** è già presente tra le sottoscrizioni attive:

- **Se l'endpoint è presente**:

        - La **checkbox è selezionata**.
        - L'utente può **solo disiscriversi**.
    - **Se l'endpoint non è presente**:
        - La **checkbox è deselezionata**.
        - L'utente può selezionarla per **iscriversi** al promemoria.

Azione di iscrizione

Se l'utente **seleziona la checkbox**, viene:

- Chiesta l'abilitazione del permesso di ricezione delle notifiche sul browser.
- Effettuata una chiamata API per registrare l'**endpoint corrente** tra le sottoscrizioni nel server.

In caso di successo:

- L'utente riceverà ogni 5 minuti una **notifica push automatica il giorno prima** dell'appuntamento.

Azione di disiscrizione

Se l'utente **deseleziona la checkbox** (con endpoint registrato):

- Viene effettuata una **chiamata API per disiscrivere l'endpoint corrente**.
- **Revoca la sottoscrizione push**: il browser non riceverà più notifiche push per questo ServiceWorker.

Gestione delle prenotazioni dell'utente

Quando l'utente è autenticato, può accedere alla propria area personale per visualizzare e gestire le prenotazioni.
L'accesso avviene tramite il **menu a tendina** che compare cliccando su **"Benvenuto, [username]"** nella **NavigationBar**.

Selezione del mese

L'utente può selezionare il **mese desiderato** da un carosello. Una volta scelto, vengono caricati gli **appuntamenti relativi a quel mese**:

- Appuntamenti **già effettuati**
- Appuntamenti **futuri**

Ogni appuntamento mostra: d**ata e ora e servizi prenotati**

Disdetta appuntamenti futuri

- Per gli appuntamenti **ancora da effettuare**, l'utente ha la possibilità di **annullare la prenotazione**.
- Cliccando sul pulsante **"Disdici"**, viene inviata una **chiamata API al server** per cancellare la prenotazione.
    - Il server **rimuove la voce corrispondente** nella tabella degli appuntamenti.
    - In seguito, l'interfaccia viene aggiornata per riflettere l'annullamento.

Gestione messaggi

Per gestire i messaggi di errore e successo, l'app utilizza un **custom hook** dedicato. Questo hook mantiene:

- Il **titolo** del messaggio
- Il **contenuto**
- Un flag per indicare se si tratta di un **errore o successo**

Questi dati vengono poi passati ai componenti FailureAlert e SuccessAlert, che mostrano i messaggi.

Rilascio di una recensione

All'interno della sezione **"Le mie prenotazioni"**, l'utente ha anche la possibilità di **lasciare una recensione**. L'utente può assegnare una valutazione da **0 a 5 stelle e un commento testuale. Quando clicca sul bottone "Invia recensione" viene effettuata la richiesta al server dell'inserimento della nuova recensione.**

Service Worker e Caching

HairTime utilizza un **Service Worker** generato automaticamente tramite il plugin vite-plugin-pwa, per abilitare funzionalità offline.

Il Service Worker è configurato per:

- **Gestire le notifiche push** in background

- **Memorizzare in cache alcune risorse**, seguendo diverse strategie:
    - /services: **Cache First** – i dati dei servizi vengono serviti prima dalla cache per migliorare la velocità di caricamento e ridurre le chiamate al server
    - /reviews: **Network First** – viene privilegiata la rete per mostrare recensioni aggiornate, ma in caso di assenza di connessione si usa la cache

Inoltre:

- Il Service Worker viene aggiornato automaticamente grazie all'opzione registerType: "autoUpdate"
- Le risorse statiche (HTML, JS, CSS, JPG, PNG ecc...) sono incluse nella cache precostruita
- Le cache obsolete vengono rimosse automaticamente con cleanupOutdatedCaches: true