A.S.D. Tennis Pro

Applicazioni e Servizi Web

01 Settembre 2023

Indice

1 Introduzione 2
2 Requisiti 2
2.1 Requisiti funzionali 2
2.1.1 Analisi del sistema preesistente 2
2.1.2 Intervista 3
2.2 Requisiti non funzionali 4
3 Design e architettura 4
3.1 Design 4
3.2 Architettura 5
3.2.1 Frontend 5
3.2.2 Backend 7
3.3 Interfaccia e mockup 8
3.3.1 Pagina Login/Registrazione 9
3.3.2 Pagina prenotazioni 10
3.3.3 Pagina statistiche (solo admin) 10
3.3.4 Pagina gestione crediti 11
3.3.5 Pagina profilo utente 12
3.3.6 Note 12
3.4 Target User Analysis 12
4 Tecnologie 14
4.1 Elementi generali 14
4.2 Elementi frontend 14
4.3 Elementi backend 15
5 Codice 15
6 Test 17
6.1 Test sul codice 17
6.2 Test di usabilità 17
7 Deployment 17
8 Conclusioni 18

1 Introduzione

Il progetto nasce dalla necessità di migliorare il servizio web based a cui si appoggia il circolo tennis Forum tennis Forlì per organizzare le prenotazioni settimanali. Il sito in questione non nasce specificatamente per gestire le esigenze degli iscritti al circolo, bensì è una soluzione su larga scala adattabile a diversi contesti sportivi.
L'obiettivo è quindi quello di realizzare una nuova versione del software che tenga conto delle esigenze degli utenti. Nello specifico il nuovo sistema permette agli utenti di prenotare i campi invitando i partecipanti, dividersi le spese e gestire il proprio credito, proprio come nella vecchia versione, ma in aggiunta, gli utenti possono accettare o rifiutare gli inviti e ottenere informazioni riguardo allo stato dei campi e al meteo, oltre che visualizzare grafici sull'andamento del circolo nel caso di utenti amministratori.

2 Requisiti

2.1 Requisiti funzionali

Per la definizione dei requisiti funzionali, il team ha analizzato in prima battuta il sistema preesistente, estrapolando il core funzionale del progetto, per poi concentrarsi sulla raccolta dei feedback da parte degli utenti al fine di individuare le feature più richieste e utili.

2.1.1 Analisi del sistema preesistente

I requisiti sono stati individuati a partire dall'analisi del sistema attualmente in uso presso ASD Forum Tennis Forlì. Nello specifico, il circolo utilizza un'applicazione web che prevede due tipi di utenti: users e administrators; queste due tipologie di utenti ricoprono rispettavamente il ruolo di soci del circolo e consiglieri, i quali formano il consiglio direttivo del circolo, incaricato di amministrare ogni attività. Gli user utilizzano il sito per prenotare i campi di ora in ora nella settimana corrente, pagando le prenotazioni tramite il conto crediti associato al loro utente.

L'area crediti dell'applicazione permette ai soci di gestire il loro denaro (depositi, ritiri e invii), nonchè di visualizzare ogni transazione passata. Durante la prenotazione gli utenti possono specificare se desiderano le luci accese o il riscaldamento (se il campo è coperto) e con chi vogliono giocare.
Gli invitati si dividono solitamente il prezzo del campo, ma esiste l'opzione di offrire la partita agli altri partecipanti.

Attualmente non esiste un sistema di inviti vero e prorio, quindi chiunque può essere aggiunto ad una partita senza esprimere consenso. Per gestire eventuali errori o imprevisti (meteo sfavorevole, disdetta con anticipo), gli administrator hanno la possibilità di cancellare le prenotazioni e rimborsare i partecipanti.

Oltre a questi privilegi gli administrator utilizzano il proprio account come un user.

Il sito dispone di altre funzionalità secondarie, che però non vengono prese in considerazione dagli utenti che sono stati intervistati, perciò abbiamo deciso di non includerle nella nostra analisi. Di seguito riportiamo i requisiti evinti dall'analisi del sistema in uso:

- L'applicazione deve consentire login e registrazioni degli utenti
- L'applicazione deve gestire le prenotazioni settimanali degli utenti, rendendo facile e chiaro specificare l'esigenza di illuminazione e ricascaldamento
- L'applicazione deve gestire i crediti degli utenti, permettendo di ricarire il proprio conto ed utilizzarlo per pagare le prenotazioni, anche dividendo il costo fra i partecipanti. Inoltre, ci deve essere la possibilità di invio di somme a beneficio di altri utenti
- L'applicazione deve prevedere un super utente in grado di cancellare le prenotazioni e rimborsare i partecipanti

2.1.2 Intervista

In un secondo momento sono state prese in considerazione le opinioni degli utenti che fanno parte del circolo tennis. La raccolta delle testimoninze ha coinvolto anche Giacomo Romagnoli (coautore), che risulta essere un membro attivo del circolo tennis forlivese, oltre ad un campione di utenti iscritti al sito di sua conoscenza.
Le interviste hanno rivelato che il sistema è utilizzato solo in parte dagli utenti, che ne necessitano unicamente per effettuare le prenotazioni. Infatti molte funzionalità secondarie sono risultate sconosciute o poco apprezzate, mentre il core dell'applicazione è stato criticato in quanto non si tiene conto di problematiche quali: il cattivo tempo non è segnalato, gli inviti sono spesso affetti da errori da parte degli utenti che non possono intervenire autonomamente, ecc... Di seguito elenchiamo i requisiti evinti dalle critiche degli utenti all'attuale sistema:

- Il nuovo sistema dovrebbe dare la possibilità agli utenti di rifiutare un invito, disponendo automaticamente il rimborso a tutti i partecipanti.
- Il nuovo sistema dovrebbe mettere a disposizione degli utenti il meteo settimanale per permettere di scegliere i giorni migliori in cui giocare.
- Il nuovo sistema dovrebbe avvertire gli utenti riguardo ad eventuali problemi del campo prima di effettuare la prenotazione, per permettere di scegliere di conseguenza.
- Il nuovo sistema potrebbe rendere disponibili delle statistiche sull'andamento dei vari campi e delle prenotaizoni al fine di tenere aggiornati gli amministratori.

2.2 Requisiti non funzionali

Infine il team si è confrontato per definire i requisiti non funzionali sulla base delle interviste e dell'analisi del sistema esistente, di seguito quanto emerso dal confronto:

- L'applicazione deve tenere un design delle interfacce minimale che ricordi quello attualmente in uso per evitare di confondere gli utenti più anziani e meno pratici.
- L'applicazione deve essere aggiornata in tempo reale sulle prenotazioni degli utenti, che tendono a prenotare i campi la domenica della settimana precedente o il lunedi della settimana corrente.

3 Design e architettura

L'applicazione web è stata costruita sulla base dello stack MEVN:

- MongoDB
- Express.js
- Vue.js (ultima versione)
- Node.js

L'utilizzo di tale stack ci ha permesso di organizzare la struttura dell'applicativo in maniera efficace, dividendo il codice backend da quello frontend.
Per entrambi i livelli è stato fatto largo uso di package esterni attraverso il gestore pacchetti `npm`. L'utilizzo di MongoDB come DBMS ha reso facile e accessibile la gestione della memorizzazione dei dati in document `JSON`.
Come libreria di connessione tra l'ambiente di Node.js e il database si è optato per `Mongoose`.

3.1 Design

Come filosofia di progettazione è stata adottata la *progettazione centrata sull'utente (User-centered design [1])* in modo da venire incontro ai bisogni dell'utente massimizzando l'usabilità dell'applicativo.
Progettazione e sviluppo sono state organizzate attraverso la definizione degli obiettivi, seguendo una metodologia *agile*, facendo utilizzo della piattaforma *Notion* come strumento per la suddivisione e la puntualizzazione degli step da completare.
Lo sviluppo è stato accompagnato da frequenti chiamate sulla piattaforma *Teams* per risolvere eventuali problemi e per fare il punto della situazione riguardo gli step e i task precedentemente assegnati.
L'applicativo è stato fortemente realizzato attraverso una logica incrementale, partendo dalla definizione della struttura generale e passando, di volta in volta,

all'implementazione delle funzionalità fondamentali. Per tutta la durata dello svolgimento si è cercato di rispettare il principio *KISS* con l'obiettivo di mantenere il prodotto minimale, evitando quindi di sovraccaricare l'applicativo di elementi non utili al fine per il quale è destinato.
A livello visivo, si è adottata la strategia *mobile first*, per rendere il prodotto fruibile anche da dispositivi mobile, mantenendo un design minimale con pagine organizzate in *cards*.

3.2 Architettura

L'architettura dell'applicativo è stata suddivisa in:

- *frontend*: composto dall'insieme dei componenti, dagli aspetti grafici puramente estetici e dagli elementi javascript puramente strutturali
- *backend*: composto dall'insieme degli elementi volti alla memorizzazione dei dati persistentemente e alla loro gestione

In generale, per la comunicazione e l'accesso ai dati si è utilizzato un approccio `REST`. Entrambe le divisioni sono dotate dei propri file `package.json` nel quale sono esplicitati tutti i pacchetti esterni utilizzati.

3.2.1 Frontend

La parte frontend può essere così riassunta:

1. *Componenti*: le sezioni del sito che rappresentano le funzionalità, ciò che l'utilizzatore finale può fare interagendo con l'applicativo
2. *Layout*: la parte statica, che rimane uguale al cambio di sezione ed entro la quale vengono caricati i componenti
3. *Elementi strutturali*: gli elementi che forniscono strumenti per gestire comportamenti di base dell'applicativo nel modo più accessibile possibile

L'intero frontend è stato sviluppato facendo uso di `Vuetify` come *component framework*. Esso ha permesso di trovare un compromesso tra adottamento di uno stile grafico unico e semplicità di utilizzo.

Vediamo ora nel dettaglio quali sono i componenti di cui è composto l'applicativo:

- *Login/Signup*: rispettivamente per il login e la registrazione di un utente. Questi componenti sono dotati anche di recupero della password tramite link inviato via email, nel caso del login, e con conferma tramite OTP nel caso di nuova registrazione
- *Prenotazioni*: la sezione in cui l'utente controlla lo stato dei campi e gli orari disponibili ed effettua la prenotazione in base alla propria discrezione, invitando uno o più altri utenti iscritti all'applicativo. Il pannello è interamente reattivo ed offre anche la possibilità di visionare lo stato meteo dei giorni della settimana

- *Crediti*: la sezione in cui l'utente gestisce il proprio denaro attraverso l'inserimento di un metodo di pagamento che permette la ricarica del proprio bilancio. In questa sezione è quindi possibile depositare o ritirare denaro dal proprio bilancio interno, ma anche inviare denaro ad un utente iscritto. All'interno della sezione è possibile vedere la lista delle transazioni con relativi grafici
- *Notifiche*: la sezione in cui l'utente riceve le notifiche relative alle prenotazioni e accetta/rifiuta gli inviti
- *Profilo*: la sezione in cui l'utente vede i propri dati e il grafico delle partite giocate nell'ultimo anno

Il layout è così composto:

- *Appbar*: la barra in alto che comprende il logo (in mezzo), e l'icone della notifica affianco a quella del profilo utente (a destra)
- *Sidebar*: il pannello laterale che mostra il nome dell'utente collegato e sotto le sezioni nelle quali si può navigare

Gli elementi strutturali principali utilizzati sono:

- *Vuex*: utilizzato per venire incontro alla necessità di avere le informazioni di uso comune all'interno dell'applicativo in un unico luogo facilmente raggiungibile. Vuex fornisce uno store centralizzato per tutti componenti dell'applicazione ed è quindi stato utilizzato per la memorizzazione di informazioni ricorrenti come il bilancio, l'email utente ecc...
- *Vue router*: il componente ufficiale di Vue per la gestione del routing. L'utilizzo di vue router ha funto da elemento cardine nella costruzione di un meccanismo semplice e scalabile per gestire la navigazione tra i componenti
- *js-cookie*: il pacchetto esterno che ha fornito gli strumenti per la gestione lato client dei cookies
- *Navigation guards*: l'elemento di Vue router utilizato per controllare la navigazione impedendo così accessi non autorizzati. Ad esempio, nel caso di un utente non loggato, le guardie reinderizzano l'utente alla pagina di login. Ancora, un utente con privilegi più bassi non può visionare le funzionalità predisposte per utenti con permessi più alti
- *Axios*: la libreria utilizzata per inviare richieste HTTP agli endpoint definiti nella directory `/src/api`
- *Web socket*: la scelta di utilizzare le web socket migliora l'aggiornamento real time di tutta l'applicazione, evitando polling o altre bad practice. La scelta è stata guidata dai requisiti, che prevedono la realizzazione di un sistema real time (esso è un elemento sia di backend che di frontend, ma non verrà reintrodotto di seguito)

3.2.2 Backend

La parte backend è stata suddivisa in:

- *Rotte*: le rotte sono dichiarate in questa sezione e ognuna è aggregata alle proprie sottorotte, nella loro dichiarazione è incluso il controller o il middlewear che dovrà gestire la richiesta
- *Middelwears*: nei middlewear avvengono i controlli su quelle specifiche rotte che necessitano di una verifica intermedia, se l'esito dei controlli è positivo la richiesta viene passata al giusto controller, altrimenti viene immediatamente notificato il client dell'errore
- *Controllers*: ogni controller aggrega la gestione delle richieste http ed eventi web socket relativi a una particolare risora, preoccupandosi di interagire con il DB per restituire ai client i dati necessari, aggiornare lo stesso DB o comunicare con gli utenti connessi tramite web socket
- *Modelli*: ogni modello rappresenta lo schema dei documenti appartenenti ad una collezione sul DB e sono utili per eseguire query, applicare constraint ai documenti e definire le chiavi di ricerca, così come prevede la libreria Mongoose

Oltre agli elementi spiegati sopra, la directory `docker` costituisce parte fondamentale del backend. Essa contiene i file di configurazione docker per far sì che il tutto esegua nel modo più *platform-independent* possibile. Sono inoltre presenti gli script, sia `.bat` che `.sh`, per l'avvio del sistema. Nel capitolo sul deployment questa parte sarà spiegata con maggiore dettaglio.

Di seguito mostriamo il diagramma delle relazioni fra le entità che compongono il database:

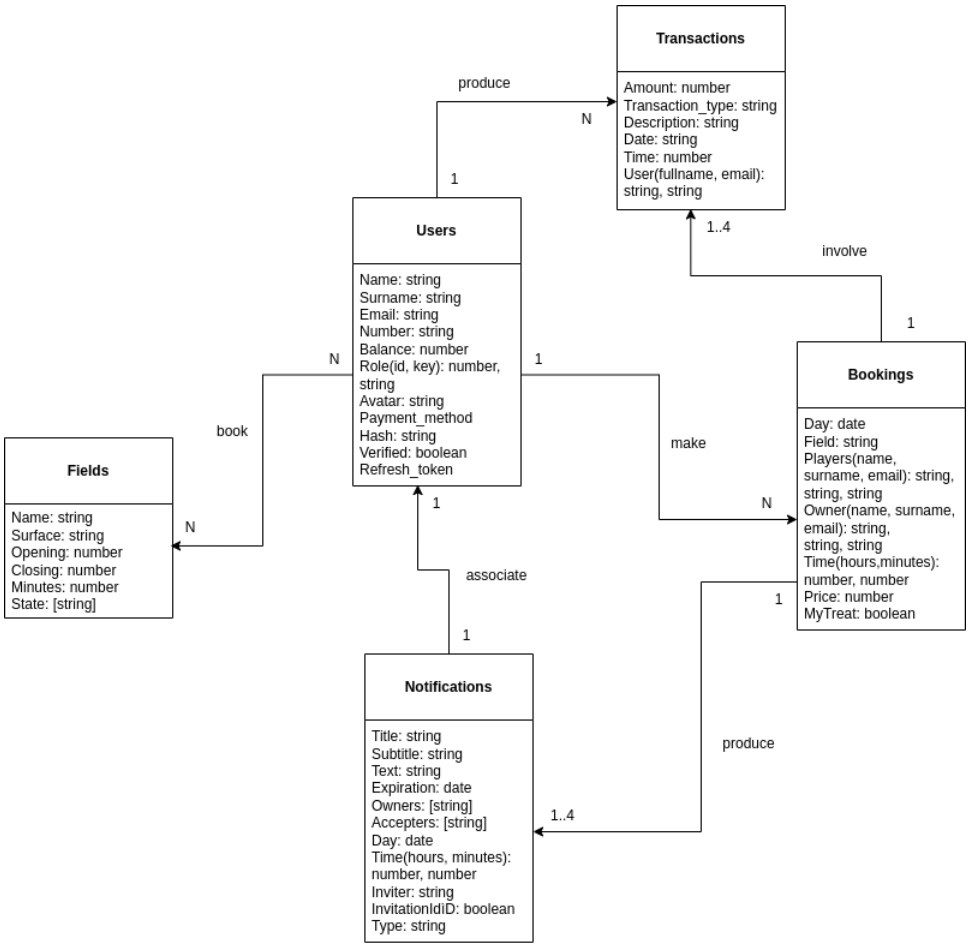

Figure 1: diagramma delle relazioni

3.3 Interfaccia e mockup

In questa sezione verranno mostrati i mockup realizzati in fase di progettazione che hanno contribuito a fornire una prima visione di ciò che ci si aspettava di ottenere.

3.3.1 Pagina Login/Registrazione

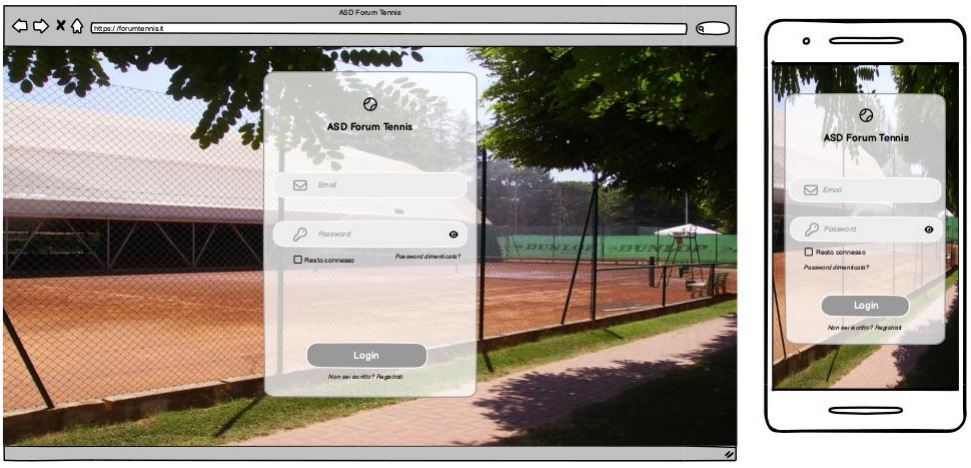

Figure 2: mockup desktop e mobile della pagina di login

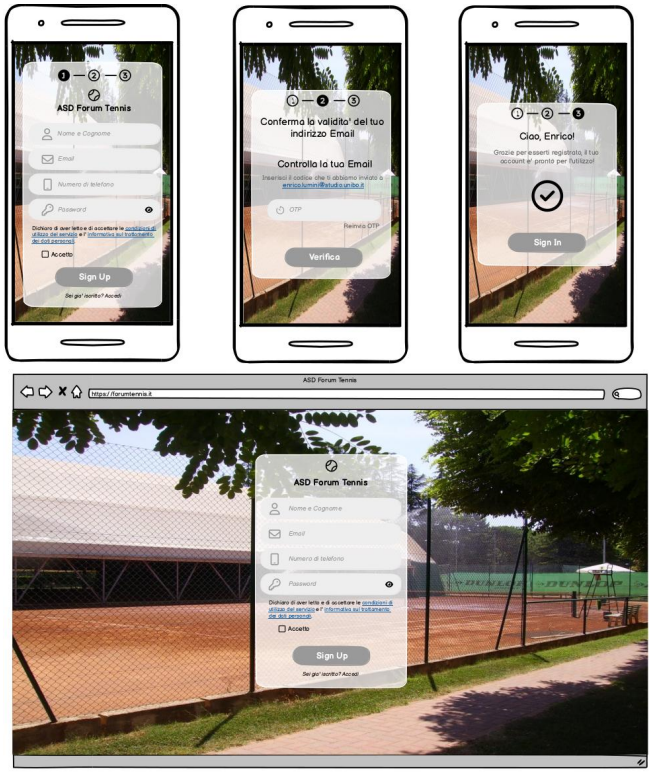

Figure 3: mockup desktop e mobile della pagina di registrazione

3.3.2 Pagina prenotazioni

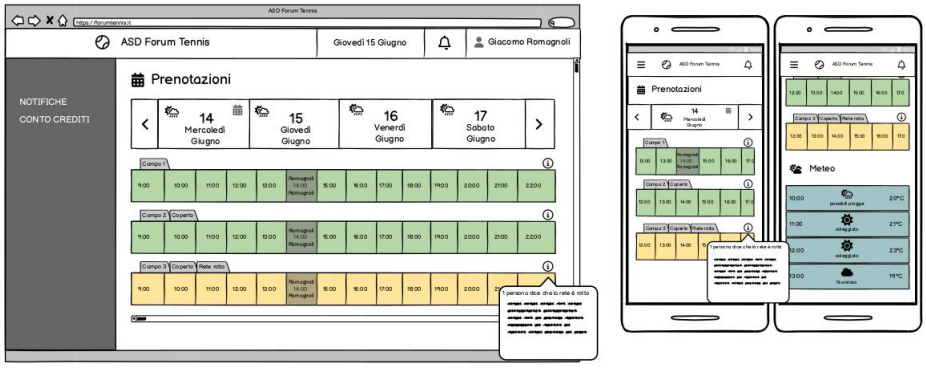

Figure 4: mockup desktop e mobile della pagina di prenotazione

3.3.3 Pagina statistiche (solo admin)

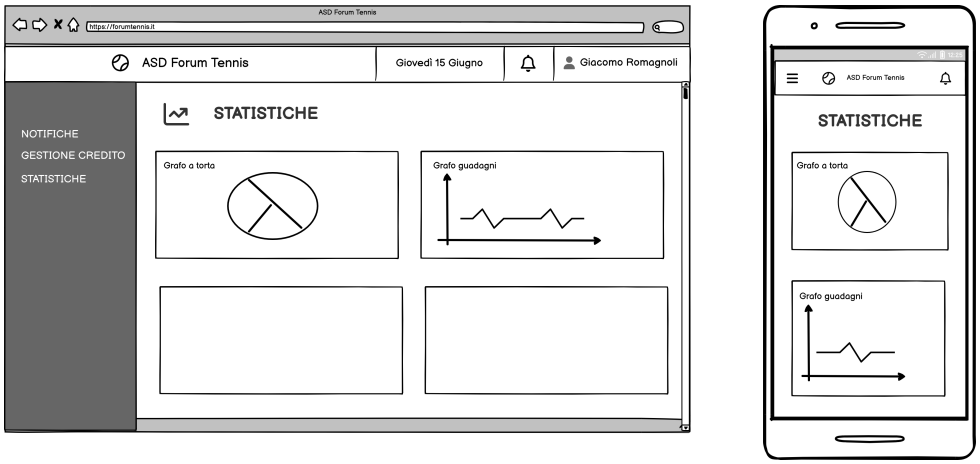

Figure 5: mockup desktop e mobile della pagina statistiche

3.3.4 Pagina gestione crediti

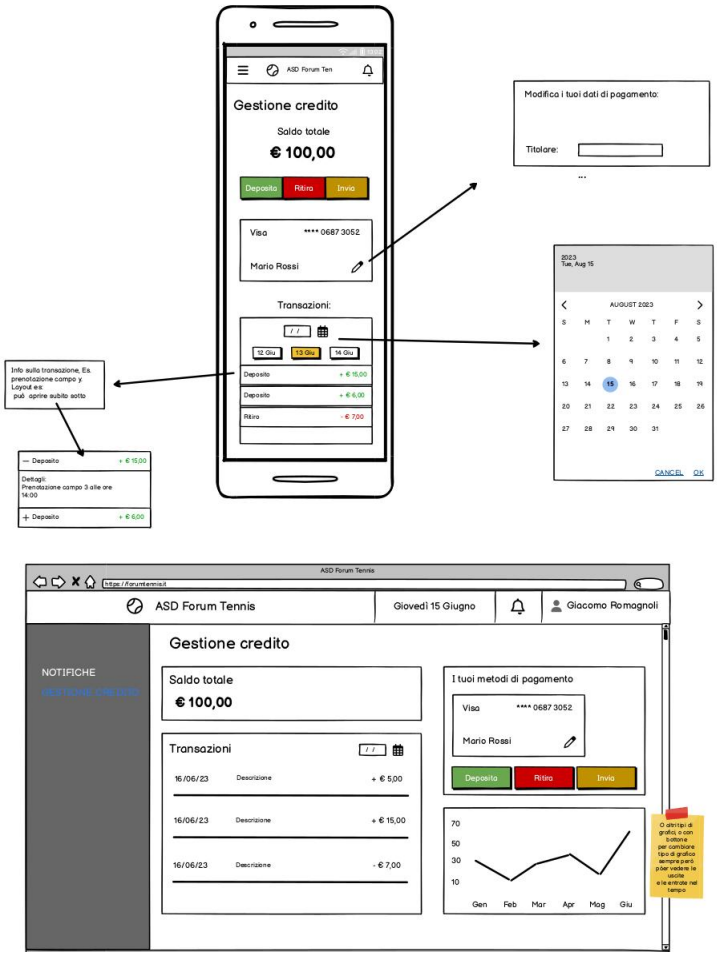

Figure 6: mockup desktop e mobile della pagina di gestione crediti

3.3.5 Pagina profilo utente

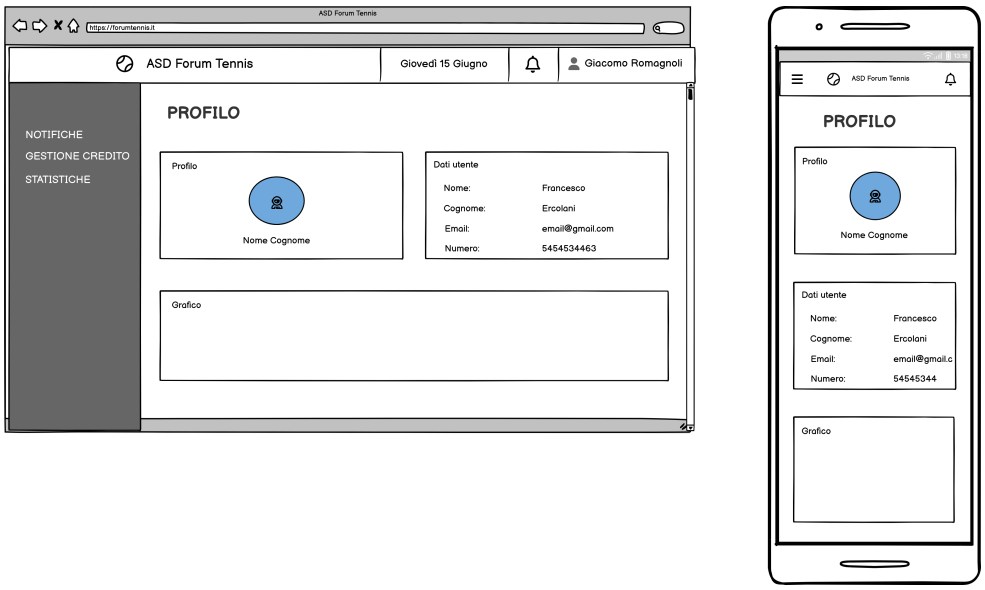

Figure 7: mockup desktop e mobile della pagina del profilo utente

3.3.6 Note

Il nome dell'applicativo all'interno dei mockup è stato scelto puramente come placeholder. In fase di sviluppo è stato cambiato.

3.4 Target User Analysis

In fase di intervista sono stati inquadrati due tipologie di utenti: utenti iscritti al circolo tennis e amministratori facenti parte del consiglio direttivo. Un indagine statistica sugli iscritti evidenzia come la maggior parte dell'utenza faccia parte di una fascia di età over 40, mentre solo una piccola percentuale ha un età inferiore ai 30; anche il consiglio direttivo conta diversi membri anziani che compongono la totalità degli amministratori. Si deduce che l'utente target non è particolarmente a proprio agio con la tecnologia, per questo sono presentate personas che rispecchino questo aspetto delicato.

Personas: Fabrizio

Fabrizio è un consigliere del circolo di 60 anni ed è iscritto da quando ne aveva 30. Prima del pensionamento era un rappresentante commerciale esperto, per questo è stato nominato tesoriere. Il suo ruolo impone di redigere budget e previsioni economiche annuali, grazie alle quali il circolo può prendere decisioni

sulla gestione in modo più consapevole. Nel tempo libero Fabrizio gioca a tennis al circolo con gli amici di vecchia data.

- Fabrizio visita regolarmente il sito alla sezione admin per aggiornarsi sul numero di prenotazioni, utenti iscritti e incassi.
- Fabrizio prenota le proprie ore sul sito, ma di tanto in tanto capita che inviti per errore l'utente sbagliato.
- Fabrizio non è molto esperto di tecnologia e necessita di essere notificato quando effettua una prenotazione o in generale interagisce con il sistema, così da rassicurarsi di aver eseguito la giusta procedura.

Personas: Graziano

Graziano è un consigliere del circolo in pensione. Avendo esperienza come tuttofare, si occupa della manutenzione dei campi che hanno sempre bisogno di manodopera. Graziano non gioca più a tennis dopo un infortunio al ginocchio, ma ama passare il suo tempo libero al circolo guardando gli altri giocare e chiacchierando con gli amici. Nel circolo è un punto di riferimento per chiunque necessiti di cancellare una prenotazione, segnalare qualche problema che necessita delle attenzioni del tuttofare o semplicemente per chi non è molto avvezzo alla tecnologia e ha bisogno di aiuto per interagire con il sito.

- Graziano visita giornalmente il sito per aggiornare lo stato dei campi.
- Graziano è incaricato di cancellare le prenotazioni su richiesta dei soci, qualora la politica del circolo lo consenta.

Personas: Michele

Michele è uno studente universitario appassionato di tennis. Michele gioca regolarmente al circolo e aspira a raggiungere il suo best ranking, per questo sfida chiunque sia disponibile per una partita.

- Michele usa regolarmente il sito per prenotare i campi, riservando il prima possibile le ore per la settimana, di modo da essere sicuro di allenarsi.
- Michele, essendo ancora uno studente, non ha grandi disponibilità finanziare, per questo controlla spesso i suoi movimenti sul conto crediti, così da tenere sotto controllo le proprie spese.

Personas: Alberto

Alberto è un'impiegato delle poste che passa i pomeriggi liberi sui campi del circolo. Alberto è un tennista vecchio stile, attento sulla scelta del campo e degli orari in cui giocare; infatti, organizza in anticipo la sua settimana così da assicurarsi di giocare nei giorni soleggiati e sui campi migliori.

- Alberto prenota le ore consultando sempre il meteo e lo stato dei campi.

4 Tecnologie

In questa sezione verranno elencate tutte le tecnologie utilizzate nello sviluppo dell'applicativo. Parte di essere sono già state spiegate nel capitolo precedente, ma verranno comunque citate.

4.1 Elementi generali

Le tecnologie utilizzate in ambito generale sono:

- *Vue 3 (versione 3.3.4)* [2]
- *MongoDB*
- *NodeJS*
- *ExpressJS (versione 4.18.2)* [3]
- *NPM*: come gestore dei pacchetti
- *HTML*
- *CSS*
- *Javascript e Typescript*
- *Docker*
- *WebSocket* [4]

4.2 Elementi frontend

- *Axios* [5]
- *Echarts* [6]: framework utilizzato nella creazione di grafici statistici
- *js-cookie* [7]
- *Mitt* [8]: libreria esterna utilizzata come rimpiazzo dell'event bus (dalla versine 3 di Vue il pattern event bus è stato deprecato)
- *Vee-validate* [9]: form library utilizzata nello sviluppo delle pagine di login e registrazione
- *Vuetify* [10]: UI library utilizzata nello sviluppo di tutti i componenti dell'applicativo
- *Vuex* [11]
- *Vue-router* [12]
- *Vue-profile-avatar* [13]

- *Vue-paycard* [14]: componente esterno importato manualmente (per esigenze di modifica interne)
- *Vite* [15]: come build tool

4.3 Elementi backend

- *bcrypt [16] e cookie-parser [17]*: rispettivamente per l'hashing delle password e la gestione dei cookies
- *otp-generator [18] e mailgen [19] + Nodemailer [20]*: rispettivamente per la creazione di un codice otp e la conferma via mail al momento della registrazione di un nuovo utente
- *Mongoose* [21]: libreria di programmazione JavaScript per la creazione di una connessione tra MongoDB e l'ambiente di runtime JavaScript Node.js
- *jose* [22]: libreria per la gestione dei JSON Web Tokens
- *cors* [23]: pacchetto che fornisce un middleware utilizzato per abilitare CORS con varie opzioni
- *dotenv* [24]: per il caricamento di variabili d'ambiente da file `.env`

5 Codice

In questa sezione verrano riportate le parti di codice più rilevanti con il fine di chiarire eventuali aspetti particolari.
Per facilitare la comprensione dell'interazione tra frontend e backend prendiamo un esempio di ciò che avviene nella pagina di gestione crediti, all'interno del pannello in cui vengono mostrate le transazioni. Tutte le transazioni relative all'utente sono memorizzate all'interno dello store `Vuex` e per ottenerle si effettua:

```
this.$store.dispatch('transactions/getTransactions',
data).then(() => {
  this.transactions = this.$store.getters.transactions;
})
```

Ci si mette inoltre in ascolto sulla socket in modo che la lista delle transazioni sia aggiornata dinamicamente in real time senza il bisogno di aggiornare la pagina:

```
this.socket.on("new-transaction", (transactionData) => {
  const { user } = transactionData;
  if (user.email === this.$store.getters.userEmail) {
    this.$store.commit("transactions/ADD_TRANSACTION",
        transactionData)
    this.transactions = this.$store.getters.transactions;
  }
});
```

All'interno dello store, alla chiamata della `dispatch`, viene chiamato il rispettivo metodo `getTransactions`:

```
getTransactions({commit}, email) {
return new Promise((resolve, reject) => {
  getTransactions(email).then(response => {
    const { data } = response
    commit('SET_TRANSACTIONS', data)
    resolve()
  }).catch(error => {
    reject(error)
  })
})
```

All'interno viene chiamato un metodo con lo stesso nome, importato dalla directory `/api/`, ovvero la directory nella quale sono definiti tutti gli endpoint con axios. Vediamolo nel dettaglio:

```
export function getTransactions(data) {
  return request({
    url: "/credits/getTransactions",
    method: "post",
    data
  })
}
```

Lato backend, le rotte sono registrate all'interno della directory `/routes/`, nel rispettivo file.

```
router.post('/getTransactions',
    creditsController.getTransactions)
```

Ciò che viene eseguito è quindi il rispettivo metodo del controller che gestisce tale rotta, in questo caso `creditsController.js`:

```
async function getTransactions(req, res) {
  const fullname = req.body.fullname
  const email = req.body.email

  const data = await Transactions.find(
  { user: {fullname: fullname, email: email }}).exec()

  return res.status(200).json(data)
}
```

Attraverso `Mongoose` viene effettuata la query e viene ritornata per essere infine mostrata in tempo reale nell'apposito pannello della pagina di gestione crediti.

6 Test

6.1 Test sul codice

Il client è stato testato su desktop utilizzando i dev tools di Chrome per simulare diversi dispositivi mobile di nuova generazione, come Tablet e smartphone. Ulteriori verifiche sul funzionamento sono state condotte sui browser più utilizzati, quali Chrome, Safari e Mozzilla Firefox. Dai test sono emerse criticità inerenti alla gestione dei cookie di alcuni dei browser, per questo il server tiene conto del browser utilizzato dal client, in modo da adattarsi alle diverse esigenze. Anche la web-api è stata testata come singola unità tramite Postman, per simulare le diverse richieste e controllarne il funzionamento.

6.2 Test di usabilità

I test con gli utenti sono stati condotti iterativamente con l'avanzamento del progetto, questo ha permesso al team di individuare quegli aspetti che necessitavano di modifiche e adattarsi prontamente alle esigenze degli utenti. Durante i test, agli utenti è stato chiesto di riprodurre il comportamento che aveva il vecchio applicativo, così da verificare se le conoscenze già apprese fossero sufficienti per l'utilizzo della nuova versione. Per quanto riguarda le nuove funzionalità, è stato necessario guidare gli utenti meno esperti, mostrando loro come interagire con le nuove interfacce. Il giudizio è stato tutto sommato positivo, al netto della barriera tecnologica che può aver influito sul pensiero di alcuni. Di seguito un riassunto di quanto emerso dai test:

- L'applicazione risulta minimale, efficace e simile alla precedente nell'interazione.
- Le nuove funzionalità necessitano di un numero limitato di prove per apprenderne l'utilizzo.
- Il design grafico è adeguato alle aspettative.

7 Deployment

Come già accennato nei capitoli precedenti, per il deployment si è fatto uso di `Docker` per il backend per massimizzare la portabilità dell'applicativo. In particolare si è definito un container avviato attraverso l'esecuzione dello script all'interno della directory `/backend/docker`.
Lato frontend, viene utilizzato `Vite` in quanto permette l'utilizzo di tutto ciò che è in grado di fornire un normale web bundler ma in maniera più efficiente in quanto, come spiegato nella doc ufficiale, è composto da:

1. Un dev server che fornisce ricchi miglioramenti per i moduli ES nativi

2. Una command buid che effettua il bundle del codice con Rollup, preconfigurato per produrre static assets altamente ottimizzati per la produzione

Oltre a quelli per il backend e frontend, esiste un file `package.json` esterno (nella root) che definisce il comando per l'avvio dell'applicativo:

```
"scripts": {
    "backend-npm-install": "npm install --prefix backend",
    "backend-npm-run": " npm run dev --prefix backend",
    "frontend-npm-install": "npm install --prefix frontend",
    "frontend-npm-run": " npm run dev --prefix frontend",
    "mongodb-nix": "cd backend/docker && ./start_db.sh",
    "mongodb-win": "cd backend\\docker && start_db.bat",
    "mongodb-macos": "cd backend/docker && ./start_db_macos.sh",
    "server": "npm-run-all backend-npm-install backend-npm-run ",
    "backend": "npm-run-all --parallel server",
    "frontend": "npm-run-all frontend-npm-install frontend-npm-run ",
    "start": "run-script-os",
    "start:windows": "concurrently \"npm run mongodb-win\"
        \"npm run backend\" \"npm run frontend\"",
    "start:nix": "concurrently \"npm run mongodb-nix\"
        \"npm run backend\" \"npm run frontend\" ",
    "start:macos": "concurrently \"npm run mongodb-macos\"
        \"npm run backend\" \"npm run frontend\" ",
    "stop": "run-script-os",
    "stop:windows": "cd backend\\docker && stop_db.bat",
    "stop:nix": "cd backend/docker && ./stop_db.sh",
    "stop:macos": "cd backend/docker && ./stop_db_macos.sh"
},
```

All'esecuzione del comando `npm run start` vengono eseguiti tutti gli script necessari a cascata, in base al sistema operativo dal quale viene avviato il comando. In particolare, vengono installati i `node modules` sia il backend che per il frontend e viene tirato su il container per il db.

Per l'avvio dell'applicativo è quindi sufficiente assicurarsi di avere docker installato, posizionarsi nella root del codice ed eseguire il comando `npm run start` da terminale. Per fermare il database bisogna eseguire il comando `npm run stop`.

8 Conclusioni

Nel complesso il progetto rispetta le aspettative del team. L'inesperienza con le nuove tecnologie ha richiesto un periodo di formazione individuale consistente, che però si è rivelato decisivo durante la fase di sviluppo, consentendo ad ogni sviluppatore di implementare correttamente le funzionalità previste nel progetto. Siccome l'utente target non risulta particolarmente interessato alle

novità, non sono state considerate nuove funzionalità o aggiornamenti, questo però non rende impossibile ulteriori integrazioni data l'attenzione con cui si sono sviluppati componenti semplici e indipendenti.

L'utilizzo di un nuovo stack è fondamentale nell'approccio professionale al web development e riteniamo quindi che lo sviluppo di questo progetto non ci sia stato di aiuto soltanto educativamente ma anche, e soprattutto, in prospettiva di ricoprire un ruolo lavorativo in questo settore.

Inoltre, il lavoro di gruppo ha contribuito ad apportare ulteriore esperienza e consapevolezza di quello che è il *modus operandi* standard nel mondo del software development.

Bibliography

[1] *User-centered-design*. URL: https://en.wikipedia.org/wiki/User-centered_design.

[2] *Vue 3*. URL: https://vuejs.org/guide/introduction.html#what-is-vue.

[3] *ExpressJS*. URL: https://expressjs.com/en/api.html.

[4] *WebSocket*. URL: https://socket.io/.

[5] *Axios*. URL: https://axios-http.com/docs/intro.

[6] *Echarts*. URL: https://echarts.apache.org/examples/en/index.html#chart-type-line.

[7] *js-cookie*. URL: https://github.com/js-cookie/js-cookie.

[8] *Mitt*. URL: https://github.com/developit/mitt.

[9] *VeeValidate*. URL: https://vee-validate.logaretm.com/v4/.

[10] *Vuetify*. URL: https://vuetifyjs.com/.

[11] *Vuex*. URL: https://vuex.vuejs.org/.

[12] *Vue Router*. URL: https://router.vuejs.org/.

[13] *vue-profile-avatar*. URL: https://github.com/pau7rr/vue-profile-avatar.

[14] *vue-paycard*. URL: https://github.com/guastallaigor/vue-paycard.

[15] *Vitejs*. URL: https://vitejs.dev/.

[16] *bcrypt*. URL: https://github.com/kelektiv/node.bcrypt.js.

[17] *cookie-parser*. URL: https://github.com/expressjs/cookie-parser.

[18] *opt-generator*. URL: https://github.com/Maheshkumar-Kakade/otp-generator.

[19] *mailgen*. URL: https://github.com/eladnava/mailgen.

[20] *nodemailer*. URL: https://nodemailer.com/.

[21] *Mongoose*. URL: https://mongoosejs.com/docs/documents.html.

[22] *jose*. URL: https://github.com/panva/jose.

[23] *cors*. URL: https://github.com/expressjs/cors.

[24] *dotenv*. URL: https://www.dotenv.org/.