

On Track

Applicazioni e Servizi Web

Riccardo Mazzi - 0001189448
riccardo.mazzi@studio.unibo.it

Nicolas Amadori - 0001182992
nicolas.amadori@studio.unibo.it

21 Gennaio 2026

Contents

1	Introduzione	3
2	Requisiti	4
2.1	Requisiti funzionali	4
2.2	Requisiti non funzionali	5
2.3	Requisiti di implementazione	6
3	Design	7
3.1	Design delle interfacce	7
3.2	Design architetturale	10
3.2.1	Infrastruttura	10
3.2.2	Entità del Sistema	10
3.3	Design API	10
3.4	Analisi degli Utenti Target	12
3.5	Definizione delle Personas e Scenari d’Uso	12
3.5.1	Persona 1: Marco, il padre di famiglia	12
3.5.2	Persona 2: Giulia, la viaggiatrice frequente	12
3.5.3	Persona 3: Alessandro, l’operatore di centrale	13
3.5.4	Persona 4: Luca, il viaggiatore occasionale	13
3.5.5	Persona 5: Elena, l’utente senza dispositivi	13
4	Tecnologie	14
4.1	Backend e API	14

4.2	Frontend	14
4.3	Persistenza Dati	14
4.4	Infrastruttura	15
5	Codice	16
5.1	Definizione delle rotte e middleware a cascata	16
5.2	Logica di autorizzazione granulare	16
5.3	Gestione Real-time con Socket.IO	17
5.4	Identificazione dei Viaggio in Corso	17
6	Test	19
6.1	User experience: Euristiche di Nielsen	19
6.2	Verifica Tecnica e Strumenti di Sviluppo	20
6.3	Test di usabilità	20
6.3.1	Metriche	20
6.3.2	Elenco dei Task	21
6.3.3	Raccolta Dati	21
6.3.4	Risultati	22
7	Deployment	24
8	Conclusioni	25

1 Introduzione

Il progetto OnTrack consiste in una Web Application progettata per modernizzare l'esperienza ferroviaria sincronizzando le operazioni di prenotazione remota con le dinamiche di bordo in tempo reale. L'obiettivo principale è fornire una piattaforma integrata con le informazioni ufficiali di Trenitalia che permetta non solo la ricerca e la prenotazione dei titoli di viaggio con selezione concorrente del posto, ma anche il monitoraggio live dello stato del treno, dei ritardi e dell'occupazione dei servizi igienici.

Gli utenti finali che accederanno alla piattaforma si possono dividere in 3 ruoli:

- Utente: ricerca tratte, prenotazione posto (con mappa posti real-time), gestione delle prenotazioni, informazioni sul proprio viaggio in corso e ricezione notifiche
- Admin: gestione treni, gestione prenotazioni, gestione utenti
- Totem on-board: visualizzazione info di viaggio e stato servizi igienici in tempo reale

2 Requisiti

2.1 Requisiti funzionali

1. Login e registrazione
2. Sidebar per la navigazione tra le pagine
3. Profilo personale
 - Modifica dei propri dati come nome e cognome
 - Modifica della password previo inserimento della password attuale
4. Ricerca di soluzioni di viaggio disponibili
 - Auto completamento nell'inserimento di stazione di partenza/arrivo
 - Selezione data e ora di partenza
 - Selezione numero di viaggiatori
5. Esposizione dei risultati trovati
 - Ordinamento in base all'orario di partenza
 - Evidenziazione dei più convenienti
6. Prenotazione della soluzione scelta. Per ogni passeggero:
 - Inserimento nome e cognome
 - Scelta del posto per ogni treno
7. Pagina delle prenotazioni personali
 - Ordinamento in base all'orario partenza
 - Eliminazione della prenotazione
 - Possibilità di mostrare il dettaglio (biglietto) della prenotazione
8. Pagina del proprio viaggio in corso, che mostra:
 - Ritardo corrente
 - Tempo mancante a destinazione
 - Disponibilità dei servizi igienici
 - Possibilità di inserirsi in una coda virtuale per essere notificati quando un servizio igienico viene liberato
 - Biglietti della prenotazione
9. Pagina per i chioschi "OnBoard" presenti sui treni attivi che mostra:
 - Codice del treno
 - Ritardo del treno

- Disponibilità dei servizi igienici del treno
10. Ricezione notifiche
- Quando si libera un servizio igienico, se l'utente era nella coda virtuale
 - Quando viene cancellato/ripristinato un treno, se l'utente ha una prenotazione con quel treno
11. (Admin) Gestione degli utenti della piattaforma
- Ricerca utenti tramite mail
 - Dare/togliere privilegi admin ad un utente
 - Eliminazione utente
12. (Admin) Gestione delle prenotazioni associate ad un utente
- Ordinamento in base alla partenza
 - Eliminazione della prenotazione
 - Possibilità di mostrare il dettaglio (biglietto) della prenotazione
13. (Admin) Gestione dei treni salvati in piattaforma
- Ricerca treni tramite codice e data
 - Cancellare/ripristinare un treno
14. (Admin) Pagina per un totem/tablet "OnBoard" presente sui treni attivi utilizzabile dal macchinista o capotreno per una simulazione degli eventi
- Modifica ritardo del treno
 - Modifica disponibilità dei servizi igienici del treno
15. Logout

2.2 Requisiti non funzionali

1. Accesso concorrente ai posti di un treno durante la prenotazione
2. Usabilità per utenti inesperti
3. Raccolta di dati reali tramite uso di API Trenitalia ufficiali
4. Disponibilità dell'applicazione 24h su 24
5. Controllo dell'autenticazione e dei permessi durante l'accesso ai dati
6. Salvataggio sicuro delle password con protezione tramite hashing

2.3 Requisiti di implementazione

1. Supporto RESTful APIs per maggiore integrazione
2. Architettura MEVN (MongoDB, Express.js, Vue.js, Node.js)
3. Utilizzo di Socket.io per connessioni e notifiche push
4. Applicazione Web responsive per garantire l'accesso multipiattaforma
5. Struttura basata su Docker

3 Design

3.1 Design delle interfacce

In questa sezione viene presentato il processo di progettazione visuale dell'applicazione. Il design è stato guidato da un approccio **mobile-first**, dando priorità all'usabilità sui dispositivi mobili per poi adattare l'esperienza agli schermi desktop tramite un layout responsive.

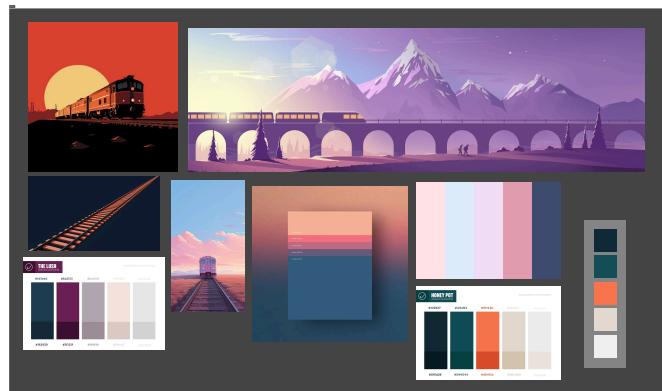
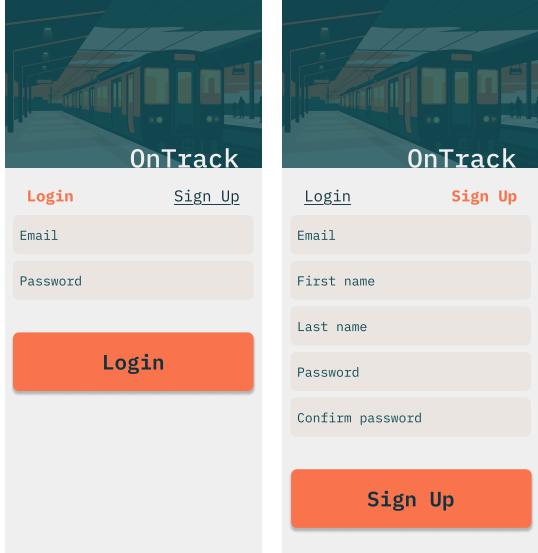


Figure 1: Moodboard

I mockup riportati di seguito rappresentano la guida estetica e funzionale seguita durante la fase di programmazione. Sebbene in fase di sviluppo siano state apportate lievi ottimizzazioni per migliorare l'esperienza utente finale, il sistema implementato mantiene un'alta fedeltà rispetto ai prototipi iniziali.



(a) Login

(b) Registrazione

Figure 2: Schermate di Autenticazione

(a) Home

(b) Risultati Ricerca

Figure 3: Ricerca Soluzioni

(a) Acquisto

(b) Posti

(a) Profilo

(b) Prenotazioni

(c) Viaggio

Figure 4: Pagina di Acquisto

Figure 5: Pagine Personali

User Management

Search for a user

mario.rossi@email.com	<input type="button" value="🛡️"/>	<input type="button" value="trash"/>
luigineri@email.com	<input type="button" value="🛡️"/>	<input type="button" value="trash"/>
riccardo.bianchi@outlook.com	<input type="button" value="🛡️"/>	<input type="button" value="trash"/>
mattia.amadori@icloud.com	<input type="button" value="🛡️"/>	<input type="button" value="trash"/>

Reservations Management

Reservations Management

mario.rossi@email.com

17/12/2025 <input type="button" value="1 change"/>	From Cesena <input type="button" value="€21.50"/>
To Bologna Centrale	1h 1min
07:09 → 8:10	
<input type="button" value="RV"/> <small>Frecciarossa</small>	<input type="button" value="trash"/>
17/12/2025	
From Cesena <input type="button" value="€8.00"/>	
To Bologna Centrale	1h 17min
17:22 → 18:39	
<input type="button" value="RV"/>	<input type="button" value="trash"/>
19/12/2025 <input type="button" value="1 change"/>	From Cesena <input type="button" value="€21.50"/>
To Bologna Centrale	1h 1min
07:09 → 8:10	
<input type="button" value="RV"/> <small>Frecciarossa</small>	<input type="button" value="trash"/>
19/12/2025	
From Cesena <input type="button" value="€8.00"/>	
To Bologna Centrale	1h 17min
17:22 → 18:39	
<input type="button" value="RV"/>	<input type="button" value="trash"/>

Train Management

17/12/2025

Search for a train

RV3904 <input type="button" value="RV"/>	<input type="button" value="trash"/>
FR9802 <small>Frecciarossa</small>	<input type="button" value="trash"/>
RV3910 <input type="button" value="RV"/>	<input type="button" value="trash"/>
RE11753 <input type="button" value="R"/>	<input type="button" value="trash"/>
RV3912 <input type="button" value="RV"/>	<input type="button" value="trash"/>
FR9810 <small>Frecciarossa</small>	<input type="button" value="trash"/>
RE11724 <input type="button" value="R"/>	<input type="button" value="trash"/>

(a) Gestione Utenti

(b) Gestione Prenotazioni

(c) Gestione Treni

Figure 6: Admin



(a) On-Board Totem

(b) On-Board Admin

Figure 7: On-Board

3.2 Design architetturale

L'architettura client-server di OnTrack si basa su MVC (Model View Controller) e in particolare utilizza lo stack MEVN (MongoDB, Express.js, Vue.js, Node.js), prestandosi per connessioni client-server bidirezionali.

3.2.1 Infrastruttura

L'infrastruttura di OnTrack è distribuita su diversi componenti. I client accedono all'applicazione dai loro browser. Un frontend web server stabilisce le connessioni con i client gestendo l'invio della one-page web application e inoltra le loro richieste ad un server backend separato. Il server backend gestisce la logica dell'applicazione e accede ai dati tramite una connessione ad un database server separato, gestito da MongoDB. Tutti i server sono costruiti sulla stessa macchina fisica ma in container separati, utilizzando Docker. Inoltre, sono presenti due network distinte: outer per connettere frontend e backend, inner per connettere backend con database. In questo modo si garantisce un maggior incapsulamento dei componenti e un maggior controllo sull'accesso dei dati, che rimane confinato all'interno delle API esplicitamente esposte.

3.2.2 Entità del Sistema

- **User:** contiene le informazioni anagrafiche (nome, cognome, email), gestisce i livelli di autorizzazione tramite il flag `is_admin` e mantiene il riferimento storico a tutte le proprie prenotazioni.
- **Train:** contiene dati operativi come il codice identificativo, lo stato di servizio (`cancelled`), eventuali ritardi (`delay`) e la gestione dinamica dello stato dei servizi igienici a bordo (disponibilità e relative code virtuali).
- **Solution:** rappresenta la soluzione di viaggio, aggregando diversi *nodi* (cambi) e definendo per ciascuno il treno assegnato, gli orari e le stazioni di partenza/arrivo, il prezzo totale basato sul percorso complessivo.
- **Reservation:** formalizza la prenotazione di una soluzione di viaggio da parte di un utente. Memorizza i dati dei passeggeri e l'assegnazione dei posti specifici per ogni singolo nodo della tratta acquistata.

3.3 Design API

Il sistema espone un'interfaccia basata sul paradigma **REST (Representational State Transfer)**. Tutte le API sono accessibili tramite l'endpoint base `/api`, il quale funge da punto di ingresso unico per le diverse componenti del backend. L'architettura è suddivisa in moduli logici indipendenti, ognuno responsabile di un dominio specifico dell'applicazione, come mostrato alla Figura 8. L'autenticazione, l'autorizzazione e la presenza dei parametri richiesti sono controllati tramite una serie di middleware a cascata che garantiscono un accesso sicuro alle risorse del sistema.

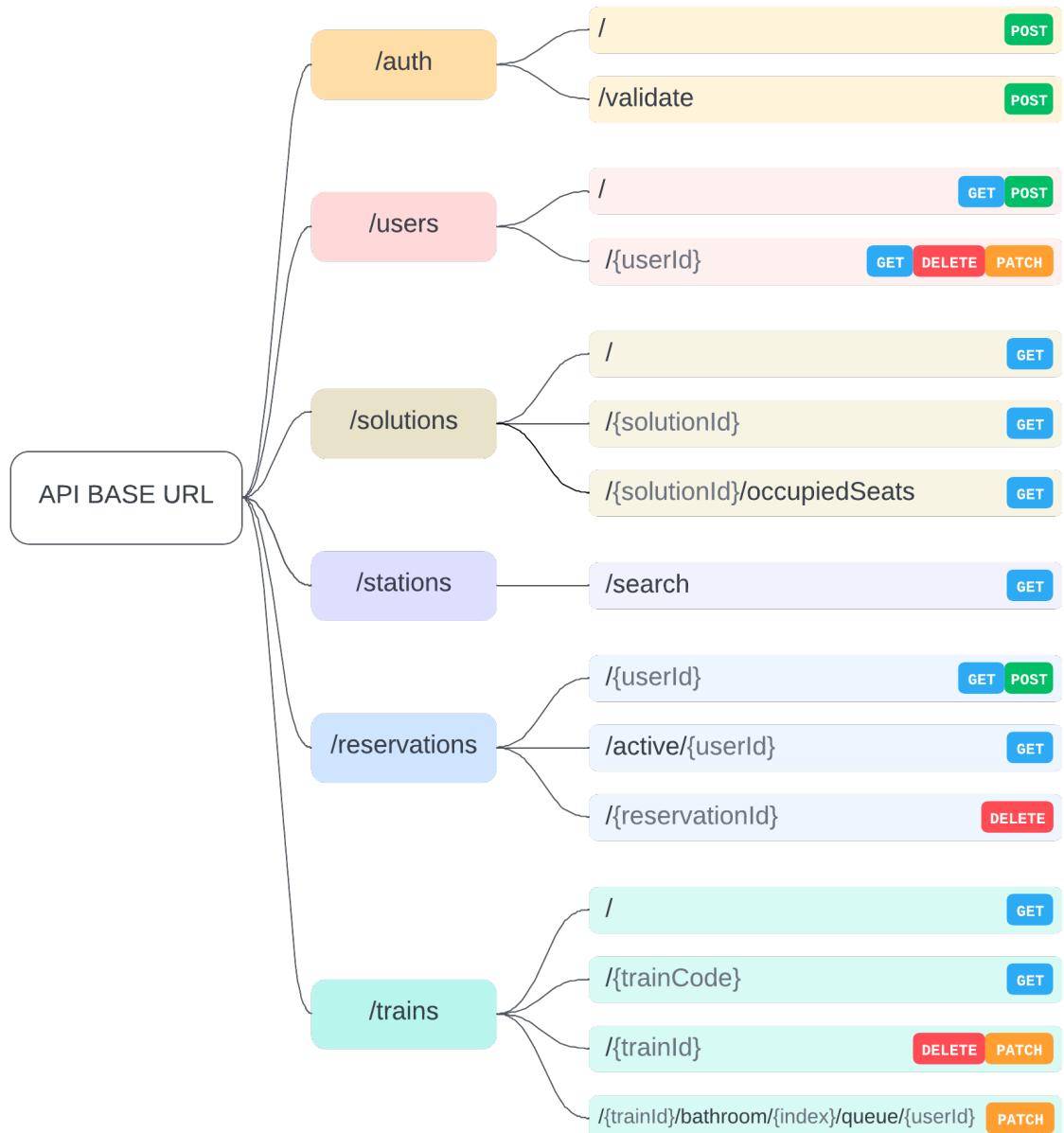


Figure 8: Struttura delle API

3.4 Analisi degli Utenti Target

L'efficacia di un sistema di acquisto titoli di viaggio dipende strettamente dalla comprensione delle necessità e dei comportamenti dei suoi utilizzatori finali. Per questa ragione, la fase di progettazione è stata preceduta da un'analisi dettagliata del target di riferimento, volta a identificare le diverse tipologie di attori che interagiscono con l'ecosistema dell'applicazione.

Per modellare tali requisiti, il lavoro è stato strutturato attraverso la definizione di specifiche **Personas**. Ogni profilo rappresenta un segmento d'utenza caratterizzato da obiettivi e competenze tecnologiche peculiari.

A integrazione di tali profili, sono stati delineati degli **Scenari d'uso** che descrivono come le Personas interagiscono con il sistema per risolvere problemi specifici. Questo approccio ha permesso di validare le scelte architettoniche e l'interfaccia utente rispetto a casi d'uso reali e verificabili.

3.5 Definizione delle Personas e Scenari d'Uso

Di seguito vengono presentati i profili utente identificati e le situazioni tipiche in cui si trovano a interagire con il sistema.

3.5.1 Persona 1: Marco, il padre di famiglia

Profilo: 45 anni, impiegato, poca dimestichezza con interfacce complesse.

Obiettivo: Acquistare biglietti per l'intero nucleo familiare (4 persone) cercando di ottimizzare tempi e costi.

Scenario 1: Acquisto di gruppo Marco deve organizzare un weekend a Firenze con la moglie e i due figli minorenni. Utilizza la funzione di acquisto multiplo, inserisce i dati dei passeggeri e sceglie 4 posti vicini.

3.5.2 Persona 2: Giulia, la viaggiatrice frequente

Profilo: 28 anni, consulente, esperta di tecnologia. Viaggia spesso per lavoro e necessita di informazioni in tempo reale.

Obiettivo: Monitorare la puntualità del treno e conoscere i servizi di bordo.

Scenario 2: Monitoraggio live e servizi Mentre si reca in stazione, Giulia apre l'applicazione per verificare lo stato del suo treno. Una volta a bordo, necessita di utilizzare i servizi igienici: attraverso la mappa interattiva del treno fornita dal sistema, visualizza in tempo reale quali bagni sono disponibili o fuori servizio, evitando di spostarsi inutilmente tra le carrozze.

3.5.3 Persona 3: Alessandro, l'operatore di centrale

Profilo: 50 anni, tecnico esperto del settore ferroviario. Ha accesso al backend del sistema e deve gestire le criticità operative.

Obiettivo: Mantenere aggiornata l'utenza sulla regolarità del servizio.

Scenario 3: Gestione guasto improvviso A causa di un guasto tecnico, un convoglio deve essere fermato. Alessandro accede alla dashboard amministrativa, seleziona il treno interessato e imposta lo stato di cancellato. Il sistema invia automaticamente una notifica push a tutti gli utenti che hanno acquistato un biglietto per quel treno.

3.5.4 Persona 4: Luca, il viaggiatore occasionale

Profilo: 35 anni, libero professionista.

Obiettivo: Gestire la cancellazione dei titoli di viaggio in modo rapido.

Scenario 4: Cancellazione per imprevisto A causa di un imprevisto, Luca non può più partire per il viaggio prenotato per il pomeriggio stesso. Accede alla sua area personale, seleziona la prenotazione attiva e richiede la cancellazione.

3.5.5 Persona 5: Elena, l'utente senza dispositivi

Profilo: 70 anni, non possiede uno smartphone o ha il dispositivo scarico durante il viaggio.

Obiettivo: Consultare le informazioni sul viaggio direttamente a bordo del treno.

Scenario 5: Utilizzo dei totem di bordo Elena si trova sul treno e desidera sapere quanto manca alla sua fermata. Non avendo con sé un telefono, si avvicina al totem interattivo installato nel corridoio della carrozza e visualizza le informazioni sul treno.

4 Tecnologie

Il sistema è stato sviluppato utilizzando lo stack **MEVN** (MongoDB, Express.js, Vue.js, Node.js), integrato con tecnologie per la comunicazione in tempo reale e la gestione della cache. Di seguito vengono dettagliate le tecnologie e le librerie principali suddivise per area di competenza.

4.1 Backend e API

Il lato server è costruito su un'architettura containerizzata, utilizzando **Node.js** come runtime environment.

- **Express.js:** Framework web utilizzato per la creazione delle API RESTful e la gestione delle rotte del server backend.
- **Socket.IO:** Libreria che abilita la comunicazione bidirezionale in tempo reale basata su eventi tra client e server (WebSocket), essenziale per le funzionalità live del progetto come notifiche e selezione posti in real-time.
- **Gestione Sicurezza e Autenticazione:**
 - **@node-rs/argon2:** Utilizzato per l'hashing sicuro delle password, offrendo performance elevate e resistenza agli attacchi brute-force.
 - **jose:** Libreria per la gestione e firma dei JSON Web Tokens (JWT) per l'autenticazione stateless.
 - **express-validator:** Middleware per la validazione e sanificazione dei dati in ingresso nelle API.

4.2 Frontend

- **Vue 3:** Framework per la gestione della logica lato client e il rendering dei componenti.
- **Tailwind CSS:** Framework CSS adottato per la progettazione dell'interfaccia. ?

4.3 Persistenza Dati

Per la gestione dei dati abbiamo optato per una soluzione NoSQL per garantire scalabilità e velocità.

- **MongoDB:** Database NoSQL orientato ai documenti, utilizzato come archivio persistente principale per l'applicazione.
- **Mongoose:** Object Data Modeling (ODM) per Node.js, utilizzato per modellare lo schema dei dati applicativi e interagire con MongoDB in modo strutturato.
- **Redis:** utilizzato nel progetto per permettere la gestione centralizzata e concorrente della prenotazione dei posti.

- **Mongo Express:** Interfaccia di amministrazione web-based per MongoDB, utilizzata in ambiente di sviluppo per la gestione visuale dei dati.

4.4 Infrastruttura

L'intero ciclo di vita dell'applicazione è gestito tramite containerizzazione per garantire la portabilità e la consistenza tra gli ambienti.

- **Docker:** Piattaforma utilizzata per containerizzare i servizi (Frontend, Backend, Database).
- **Docker Compose:** Strumento di orchestrazione utilizzato per definire ed eseguire l'applicazione multi-container.
- **Gestione Reti:** Sono state configurate due reti distinte nel `docker-compose`:
 - `inner`: Rete interna isolata (driver bridge, internal true) per la comunicazione sicura tra Backend, Database e Redis, non accessibile dall'esterno.
 - `outer`: Rete pubblica per esporre i servizi accessibili all'utente finale (Frontend e Backend API).

5 Codice

5.1 Definizione delle rotte e middleware a cascata

Questo estratto dalla definizione delle rotte delle prenotazioni, illustra l'architettura di sicurezza a "pipeline" adottata dal backend per verificare la possibilità di eseguire i controller. Mostra come ogni richiesta venga filtrata sequenzialmente: prima viene validata la sintassi dell'input (authHeaderValidator), poi viene verificata l'identità dell'utente (verifyToken), e infine vengono controllati i permessi specifici (requireAdminOrSelf).

```
1 // Catena di middleware:  
2 // Validazione Header-> Verifica Token-> Controllo Permessi-> Controller  
3 router.post('/:userId',  
4   authHeaderValidator,  
5   verifyToken,  
6   requireAdminOrSelf, // Autorizza solo se l'ID nel token corrisponde  
   al parametro o admin  
   create_reservation  
 );  
9  
10 router.delete('/:reservationId',  
11   authHeaderValidator,  
12   verifyToken,  
13   requireAdminOrReservationOwner, // Controllo granulare sulla  
   proprietà della risorsa  
   delete_reservation  
14 );  
15 
```

5.2 Logica di autorizzazione granulare

Questo middleware rappresenta un componente fondamentale per la sicurezza dei dati. Questo codice implementa una logica Role-Based Access Control: interroga il database per verificare che l'utente richiedente sia l'effettivo proprietario della risorsa che sta tentando di modificare.

```
1 export const requireAdminOrReservationOwner = async (req, res, next) => {  
2   try {  
3     const userIdFromToken = req.user.id;  
4     const reservationId = req.params.reservationId;  
5  
6     const user = await User.findById(userIdFromToken).exec();  
7  
8     const isAdmin = req.user.is_admin === true;  
9     const hasReservation = user.reservations.some(reservation =>  
10       reservation.toString() === reservationId);  
11  
12     if (!isAdmin && !hasReservation) {  
13       return res.status(403).json({ message: 'Access forbidden: You can  
14         not access this reservation' });  
15     }  
16     next();  
17   } catch (err) {  
18     console.error(err);  
19     res.status(500).json({ message: 'Internal server error' });  
20   }  
21 };
```

```

16     } catch (err) {
17       return res.status(500).json({ message: 'Error during authorization: '
18         + err.message });
19   }

```

5.3 Gestione Real-time con Socket.IO

Il codice evidenzia l'uso del concetto di "rooms" per isolare il traffico dati relativo a specifici treni e la gestione degli eventi di join e lock. È rilevante mostrare come il server sincronizzi istantaneamente lo stato dei posti (occupati o selezionati) tra tutti i client connessi allo stesso treno, prevenendo conflitti di prenotazione.

```

1 io.on('connection', async (socket) => {
2   // ...
3   socket.on('join_room', async (roomName) => {
4     socket.join(roomName);
5     const seats = await getSelectedSeats(roomName.replace("train_", ""));
6     seats.forEach(s => io.to(socket.id).emit('seat_selected', s));
7   );
8
9   socket.on('lock_seats', async (data) => {
10     if (!id) return; // Richiede autenticazione
11     try {
12       await handleSeatLock(id, data.bookingGroups);
13     } catch (error) {
14       console.error("Socket lock error:", error);
15     }
16   );
17 });

```

5.4 Identificazione dei Viaggio in Corso

Questo codice mostra come il backend filtri le soluzioni di viaggio attive in base all'orario corrente, e di queste mantenga solo i cambi (nodes) attivi al momento. Vengono quindi isolati e restituiti solo i dati dei cambi (nodes) attualmente in transito permettendo al frontend di mostrare contestualmente solo le informazioni rilevanti (come il posto attuale) e ignorare le tratte già concluse o future.

```

1 export const get_active_reservations_nodes = async function(req, res) {
2   // ...
3   const allReservations = await joinReservationsWithSolutions(
4     reservations, userId); // Join tra le prenotazioni utente e le
5     soluzioni di viaggio contenute nel db
6
7   const cetNow = new Date().getTime();
8   // Mantiene solo i nodi attivi
9   const activeNodes = allReservations
10     .flatMap(r => r.nodes)

```

```

9      .filter(n => !n.train.cancelled && new Date(n.departure_time).
10     getTime() < cetNow && new Date(n.arrival_time).getTime() > cetNow);
11
12     // Prende i passeggeri di questi nodi per poterli mostrare
13     const activeNodeIds = activeNodes.map(n => n._id.toString());
14     const passengers = allReservations
15       .flatMap(r => r.passengers)
16       .filter(p => p.seats.some(s => activeNodeIds.includes(s.node._id
17         .toString())));
18
19     // Mantiene solo i posti per questi nodi
20     passengers.forEach(p => p.seats = p.seats.filter(s => activeNodeIds.
21       includes(s.node._id.toString())));
22     // ...
23
24     return res.status(200).json({
25       success: true,
26       count: activeNodes.length,
27       nodes: activeNodes,
28       passengers: passengers
29     });
30   }

```

6 Test

6.1 User experience: Euristiche di Nielsen

Lo sviluppo dell'applicazione per l'acquisto non si è limitato alla sola implementazione funzionale. Durante l'intero ciclo di progettazione, l'interfaccia utente (UI) è stata costruita seguendo delle linee guida di usabilità, al fine di garantire un'esperienza fluida sia per l'utente finale (viaggiatore) che per gli amministratori.

In particolare, il riferimento principale sono state le Euristiche di Usabilità di Nielsen. Queste regole generali hanno guidato le scelte di layout, navigazione e interazione, assicurando che il sistema rispetti gli standard cognitivi e visivi attesi dagli utenti.

- 1. Visibilità dello stato del sistema** Per garantire che l'utente sia sempre consapevole di ciò che accade, abbiamo inserito feedback visivi immediati: ad esempio, breadcrumbs di navigazione per indicare la posizione nel sito e indicatori di caricamento (spinner) durante la ricerca delle soluzioni.
- 2. Corrispondenza tra sistema e mondo reale** Abbiamo adottato un linguaggio familiare all'utente, evitando tecnicismi del database. Le etichette utilizzano termini comuni del settore ferroviario e le icone scelte (come la locomotiva o il biglietto stilizzato) richiamano concetti fisici immediatamente riconoscibili.
- 3. Controllo e libertà dell'utente** Considerando la possibilità di errori accidentali, abbiamo integrato chiare "uscite di emergenza". L'utente può, ad esempio, utilizzare il pulsante "Back" per fare modifiche al proprio flusso di attività o utilizzare la sidebar per cambiare pagina in qualsiasi momento.
- 4. Coerenza e standard** Abbiamo mantenuto una rigorosa coerenza visiva e funzionale in tutte le schermate. Tutti gli elementi, sia primari che secondari, condividono lo stesso codice colore e posizionamento, così come le icone di modifica ed eliminazione, riducendo il carico cognitivo necessario per orientarsi.
- 5. Prevenzione dell'errore** Abbiamo progettato l'interfaccia per prevenire errori a monte. Ad esempio, il pulsante "Search Trains" rimane disabilitato finché l'utente non seleziona stazioni di partenza e arrivo valide, e i campi data impediscono la selezione di giorni passati o di un numero di passeggeri negativo.
- 6. Riconoscimento anziché ricordo** Per evitare che l'utente debba affidarsi alla memoria, abbiamo reso visibili le informazioni chiave in ogni fase. Nella schermata di acquisto, vengono riproposti chiaramente i dettagli del viaggio selezionato (orari, stazioni e cambi) scelti nei passaggi precedenti.
- 7. Flessibilità ed efficienza d'uso** Data la natura essenziale del sistema, abbiamo puntato a progettare un flusso ottimizzato in modo tale che il numero di passaggi per completare ogni operazione sia già ridotto al minimo indispensabile senza quindi inserire scorciatoie.

- 8. Estetica e design minimalista** Abbiamo adottato uno stile grafico minimale per mantenere l'interfaccia pulita. Per comunicare informazioni come le tipologie di treno o i servizi offerti, abbiamo scelto di utilizzare loghi e icone intuitive al posto di etichette testuali ingombranti; questa scelta ha permesso di ridurre il carico di informazioni e di convogliare l'attenzione dell'utente esclusivamente sui dati essenziali, come orari e prezzi.
- 9. Aiuto agli utenti nel diagnosticare e recuperare dagli errori** Abbiamo utilizzato messaggi di errore in linguaggio naturale. In caso di input errato (es. login fallito), il sistema suggerisce come correggere il problema (es. "Il formato dell'email non è valido").
- 10. Aiuto e documentazione** Considerando intuitività del sistema e l'adozione rigorosa di pattern di interazione e componenti standard (comuni alla maggior parte dei siti web e delle app moderne), abbiamo ritenuto non necessaria la creazione di una documentazione di supporto. L'interfaccia è stata progettata per essere autoesplicativa, guidando l'utente in modo naturale.

6.2 Verifica Tecnica e Strumenti di Sviluppo

Parallelamente all'applicazione delle euristiche, abbiamo condotto una fase continua di test tecnici durante l'intero ciclo di sviluppo per garantire la robustezza del sistema.

Per assicurare la compatibilità cross-browser, l'applicazione è stata testata regolarmente su **Google Chrome** e **Mozilla Firefox**. Inoltre, coerentemente con l'approccio progettuale **mobile-first**, è stata fondamentale la verifica su schermi di dimensioni ridotte. Abbiamo utilizzato gli strumenti di sviluppo dei browser (Device Mode) per simulare diversi viewport di smartphone e tablet, assicurandoci che il layout responsivo si adattasse correttamente e che l'esperienza d'uso rimanesse ottimale anche su dispositivi touch.

Relativamente al backend, abbiamo validato il corretto funzionamento delle API REST utilizzando **Postman** per interrogare gli endpoint e verificare le risposte HTTP. L'utilizzo di **Mongo-express** è stato infine essenziale per visualizzare in tempo reale lo stato del database, permettendoci di confermare che le operazioni di inserimento e modifica si riflettessero correttamente sui dati persistenti.

6.3 Test di usabilità

Per ciascuno dei task elencati di seguito, ai partecipanti verrà richiesto di valutare l'esperienza d'uso basandosi su tre metriche fondamentali (ognuna con un voto da 1 a 5):

6.3.1 Metriche

- **Intuitività:** Il flusso è semplice da capire e da usare?
- **Velocità:** Il sistema è reattivo e l'interazione è priva di frizioni?

- **Estetica:** L'interfaccia risulta leggibile e con un look moderno?

6.3.2 Elenco dei Task

- Onboarding:** Creazione di un nuovo account ed effettuazione del login.
- Acquisto:** Ricerca di una soluzione di viaggio e acquisto del relativo biglietto.
- Gestione Prenotazioni:** Visualizzazione delle proprie prenotazioni ed eliminazione di una prenotazione esistente.
- Amministrazione Utenti:** Accesso alla lista utenti per visualizzare le loro prenotazioni, eliminare un account o promuovere un utente al ruolo di amministratore.
- Cancellazione Treni:** Cancellazione di un treno dal sistema.
- Gestione Treni:** Impostazione di un ritardo per un treno specifico e modifica dello stato di disponibilità dei servizi igienici (WC).

6.3.3 Raccolta Dati

Intuitività Semplicità di apprendimento e uso dell'interfaccia

Task	Mauro	Elena	Francesco	Serena	Eleonora	Media
Onboarding	4	4	5	5	5	4.6
Acquisto	5	3	3	3	2.5	3.3
Gestione Prenotazioni	4	4	4	5	5	4.4
Amministrazione Utenti	4	3	4	5	5	4.2
Cancellazione Treni	4	3	3	4	4	3.6
Gestione Treni	3	5	4	4	4	4

Table 1: Valutazione degli utenti sull'intuitività (Media totale: 4.02)

Velocità Reattività e mancanza di frizione dell'interfaccia

Task	Mauro	Elena	Francesco	Serena	Eleonora	Media
Onboarding	5	5	5	5	5	5
Acquisto	5	5	5	5	5	5
Gestione Prenotazioni	5	5	5	5	5	5
Amministrazione Utenti	5	5	5	5	5	5
Cancellazione Treni	5	5	5	5	5	5
Gestione Treni	5	5	5	5	5	5

Table 2: Valutazione degli utenti sulla velocità (Media totale: 5.00)

Estetica Leggibilità e look moderno dell'interfaccia

Task	Mauro	Elena	Francesco	Serena	Eleonora	Media
Onboarding	4	4	4	4	4	4
Acquisto	4	4	5	5	4.5	4.5
Gestione Prenotazioni	5	5	5	5	5	5
Amministrazione Utenti	4	4	5	5	5	4.6
Cancellazione Treni	5	4	5	5	4	4.6
Gestione Treni	5	4	4	5	5	4.6

Table 3: Valutazione degli utenti sull'estetica (Media totale: 4.55)

6.3.4 Risultati

L'interfaccia utente è risultata molto veloce, con poca frizione, con una buona estetica e nella maggior parte dei task abbastanza intuitiva. I test di usabilità sono serviti al loro scopo, sollevando piccole problematiche che sono state poi risolte, migliorando l'usabilità di OnTrack.

1. Visibilità della password:

gli utenti durante la registrazione non hanno la possibilità di visualizzare la password inserita (in quanto censurata per questioni di sicurezza)

2. Selezione posti di una soluzione con più cambi:

gli utenti selezionano solo il posto del primo treno, senza notare la freccia per selezionare i posti dei treni degli altri cambi. Inoltre questa operazione aumenta la frizione per utenti che non vorrebbero scegliere il posto ma preferirebbero che gli venisse assegnato un posto qualsiasi.

3. Icona info che apre il biglietto della prenotazione:

alcuni utenti trovano poco intuitivo che un pulsante "info" apra un popup con il biglietto dentro

4. Treni cliccabili da admin per entrare nella pagina OnBoard:

gli utenti (nel ruolo di admin) si trovano spaesati nel momento in cui, dalla pagina di gestione treni, cliccano su un treno e si ritrovano nella pagina OnBoard admin di quel treno. Questa pagina è fatta per un totem a bordo e quindi radicalmente diversa da tutte le altre pagine, di fatto interrompendo il flusso di navigazione dell'applicazione.

5. Icona cancellazione treno poco visibile, non intuitiva per riabilitare un treno:

gli utenti (nel ruolo di admin) trovano l'icona di cancellazione treno poco visibile. Quando il treno è cancellato l'icona si riempie ed è poco intuitivo ricliccarla per poter riabilitare il treno.

Di seguito una lista delle migliorie apportate per risolvere tali problematiche:

1. Inserimento del pulsante nella casella di input per mostrare/nascondere la password.
2. Selezione dei posti casuale durante l'apertura della pagina di acquisto, pur lasciando la possibilità di selezionarne uno specifico. In questo modo l'utente vede fin da subito che nel pulsante di selezione posti è già presente un posto per ogni cambio, e nel caso non voglia sceglierli può tenere quelli assegnati dal sistema.
3. Sostituita l'icona info con l'icona di un biglietto, che risulta essere più adeguata ad indicare l'azione di apertura del biglietto di prenotazione.
4. Rimosso il link alla pagina admin OnBoard del treno in modo da evitare di accedere involontariamente ad essa, lasciando invariata la possibilità di accedervi tramite link diretto. Come per la pagina OnBoard classica, è separata dall'applicazione principale e deve essere acceduta dal totem solo una volta, rimuovendo la necessità di un collegamento con l'applicazione principale.
5. L'icona di cancellazione è sempre piena, migliorando la leggibilità. Quando un treno è cancellato, viene sostituita con un'icona che sia più adatta ad indicare la riabilitazione del treno.

7 Deployment

Avendo utilizzato **Docker**, l'installazione e la messa in funzione dell'applicazione risulta estremamente semplice: basterà seguire i passi elencati di seguito.

Installazione

- Clonare la repository di OnTrack:
`git clone https://github.com/NicolasAmadori/OnTrack.git`
- Avviare il daemon di Docker
 - **Avviando l'applicazione “Docker Desktop”**
 - Oppure usando i comandi shell:
`sudo service docker start # for SysVinit`
`sudo systemctl start docker # for Systemd`
- Inizializzare i volumi di MongoDB lanciando lo script
`./FORCE_CREATE_NOVOLUME.sh`

Avvio

- **Usando lo script `start.sh`**
- Oppure usando i comandi shell:
`docker compose build; docker compose up -d`

Arresto

- **Usando lo script `./stop.sh`**
- oppure usando il comando shell:
`docker compose down --rmi all`

Riavvio

- **Usando lo script `./restart.sh`**
- Oppure usando i due script:
`./stop.sh; ./start.sh`

Connessione

- Il dispositivo su cui è stato fatto il deployment può accedere ad OnTrack connettendosi a `http://localhost:5173`
- Tutti gli utenti nella rete locale possono accedere ad OnTrack connettendosi al dispositivo in cui è stato fatto il deployment dell'applicazione, sulla porta `5173`

8 Conclusioni

Il team ritiene di avere implementato con successo tutti gli obiettivi che si era prefissato. OnTrack è un applicativo completo che permette di risolvere tutti i casi d'uso individuati tramite un'interfaccia che, tramite i test, ha dimostrato di essere semplice e intuitiva sin dal primo utilizzo. Sono stati seguiti i paradigmi delle interfacce web moderne, facendo sempre attenzione all'accessibilità della piattaforma.

Durante lo sviluppo una delle sfide principali è stata l'adattamento del nostro sistema alla struttura delle informazioni offerte dalle API ufficiali Trenitalia.

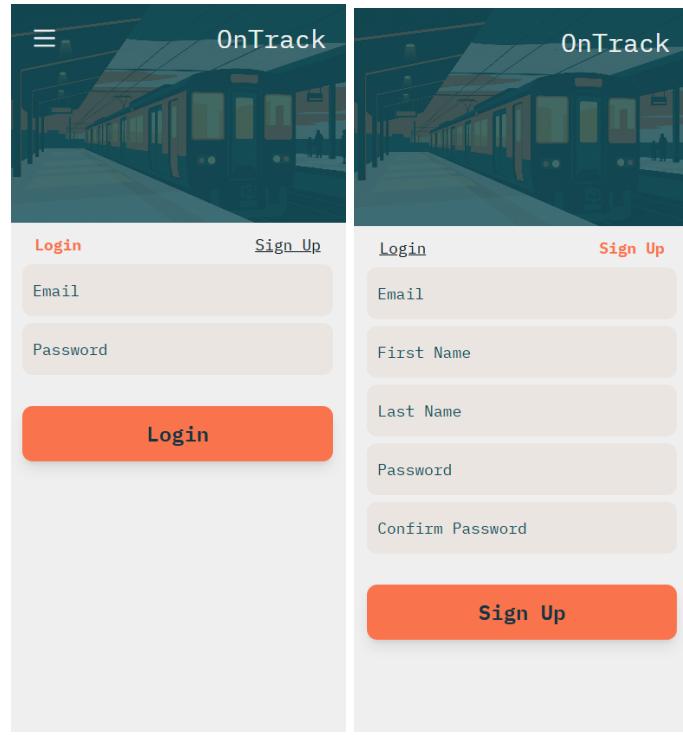
Abbiamo avuto modo di prendere confidenza con lo sviluppo di un'applicazione MEVN utilizzando vari strumenti presenti negli applicativi moderni.

Il team non ha riscontrato problemi nella coordinazione interna, garantendo una divisione dei compiti e un'ottima sincronizzazione grazie alle metodologie di versioning.

Nonostante gli obiettivi prefissati siano stati raggiunti, il team ha individuato possibili sviluppi futuri quali un'integrazione completa alle API ufficiali che permetta l'acquisto reale dei biglietti e l'implementazione di ridondanza all'interno dei componenti dell'architettura di rete, in modo da garantire un servizio più stabile e affidabile.

Schermate

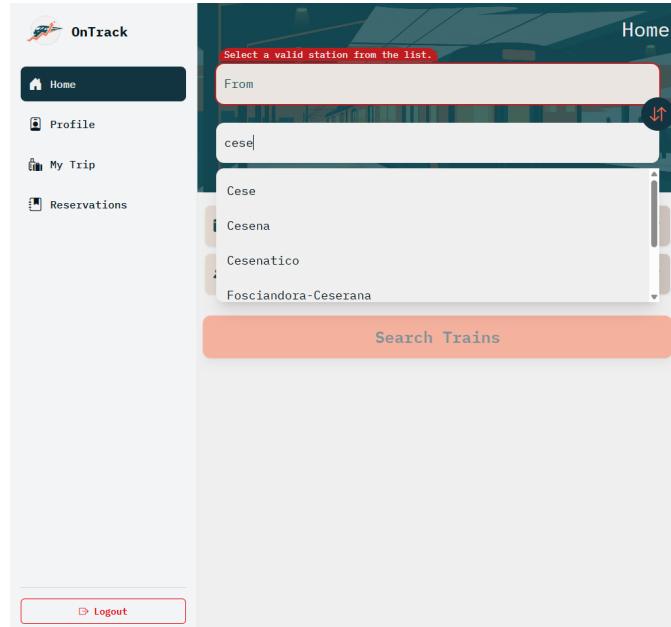
Le seguenti schermate sono volutamente di dimensioni di schermo differenti per mostrare l'aspetto responsivo di OnTrack dai punti di vista di dispositivi diversi.



(a) Login

(b) Sign Up

Figure 9: Schermate di autenticazione

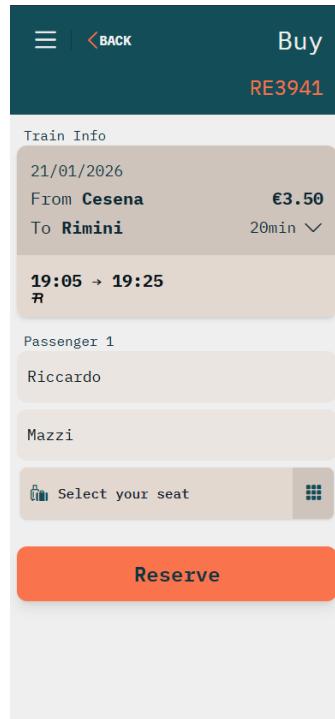


(a) Home

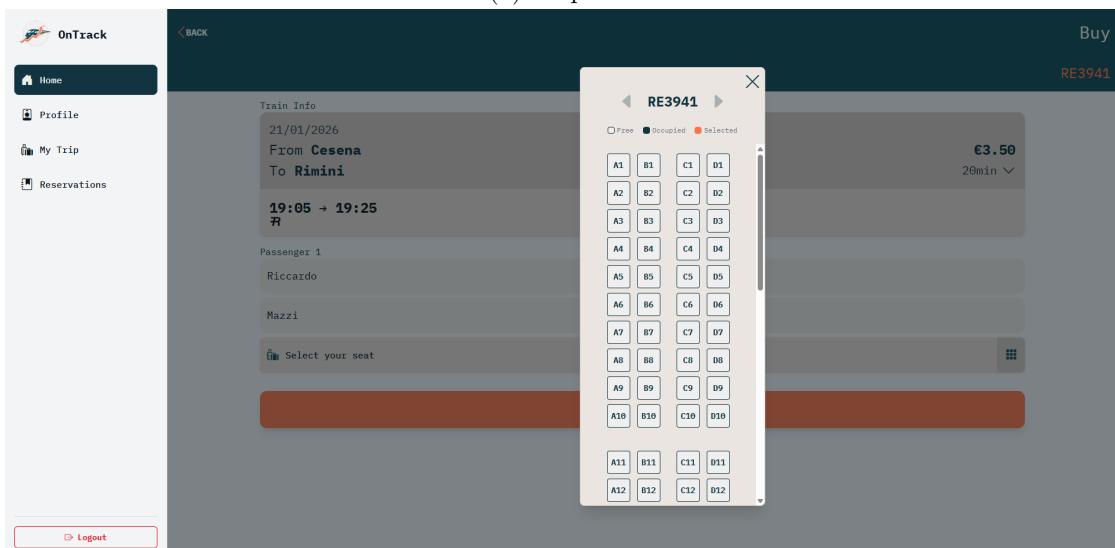
Results		
Cesena → Rimini		
Cesena → Rimini 19:05 → 19:25 ⚡		3.50€
Cesena → Rimini 19:21 → 19:55 ⚡		3.50€
Cesena → Rimini 19:42 → 20:01 RV		3.50€
Cesena → Rimini 19:55 → 20:18 ⚡		3.50€
Cesena → Rimini 20:29 → 20:49 FRECCIAROSSA		16.00€
Cesena → Rimini 20:46 → 21:12 RV		3.50€
Cesena → Rimini 21:19 → 21:55 ⚡		3.50€
Cesena → Rimini 21:29 → 21:49 FRECCIAROSSA		16.00€
Cesena → Rimini 21:42 → 22:01 RV		3.50€

(b) Risultati ricerca

Figure 10: Ricerca Soluzioni

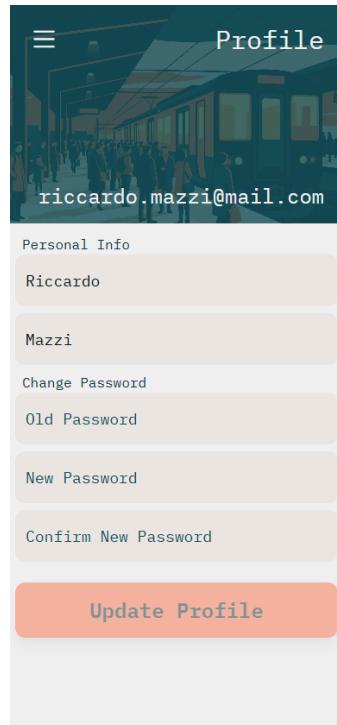


(a) Acquisto



(b) Popup di selezione posti

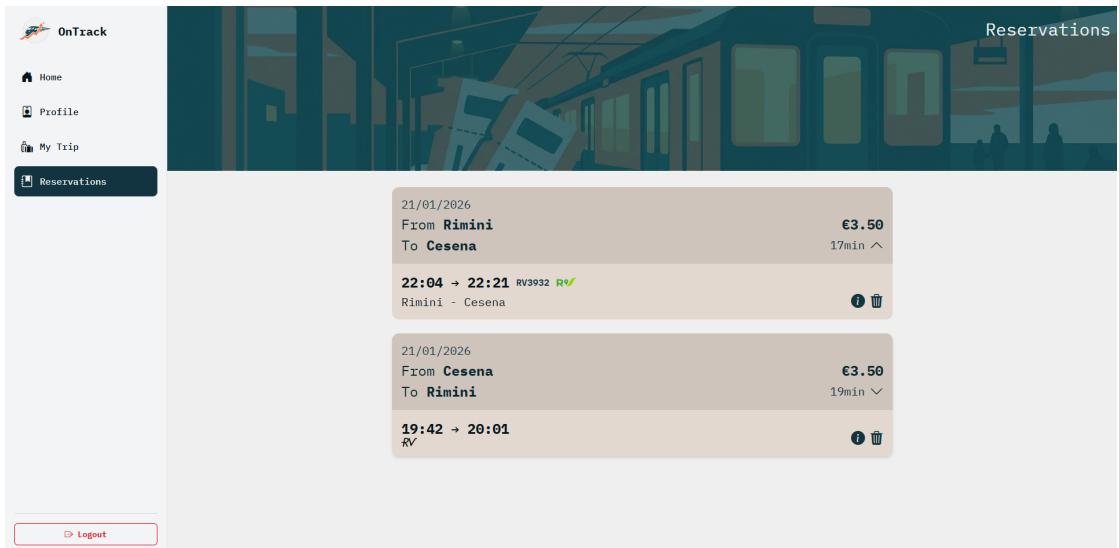
Figure 11: Pagina di Acquisto



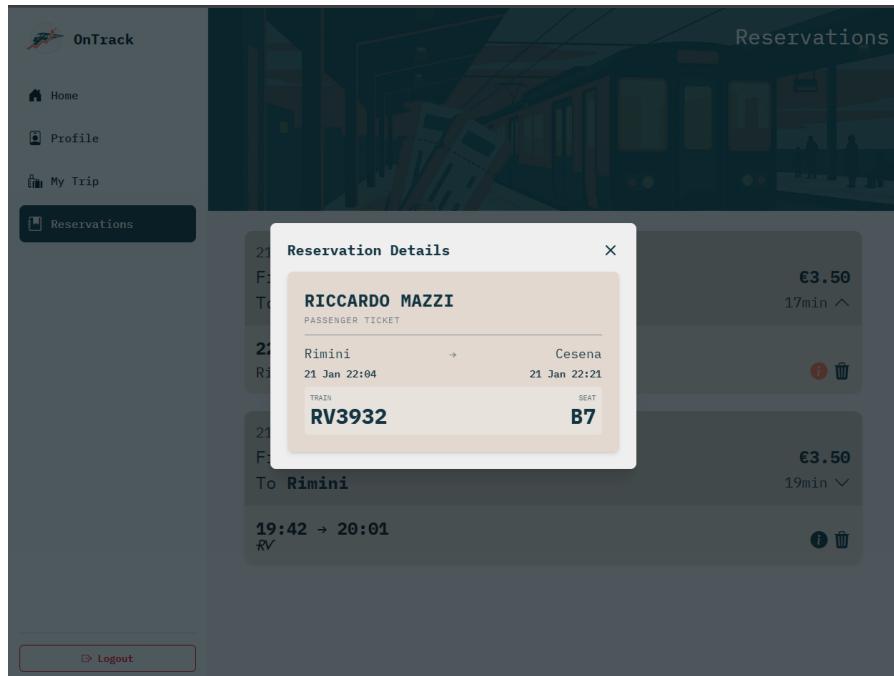
(a) Profilo

(b) Il mio viaggio

Figure 12: Pagine personali

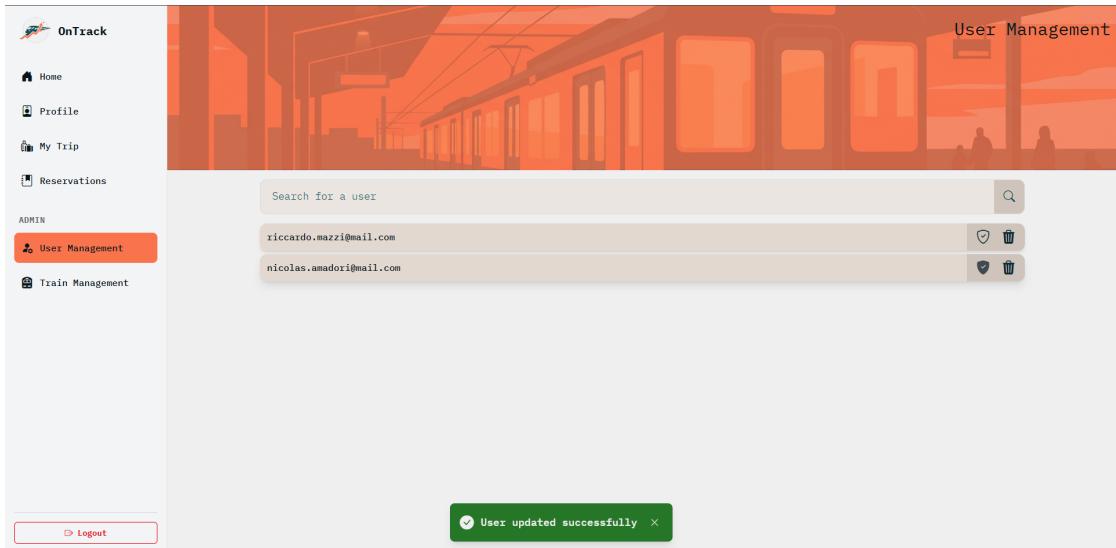


(a) Le mie prenotazioni



(b) Dettaglio prenotazione (biglietto)

Figure 13: Prenotazioni



(a) Gestione Utenti

Reservation Management

riccardo.mazzi@mail.com

21/01/2026	Wednesday, 21 January 2026
From Rimini	€3.50
To Cesena	17min ▲
22:04 → 22:21 RV3932 R✓	
Rimini - Cesena	

21/01/2026	Wednesday, 21 January 2026
From Cesena	€3.50
To Rimini	19min ▼
19:42 → 20:01 RV	

Train Management

Wednesday, 21 January 2026

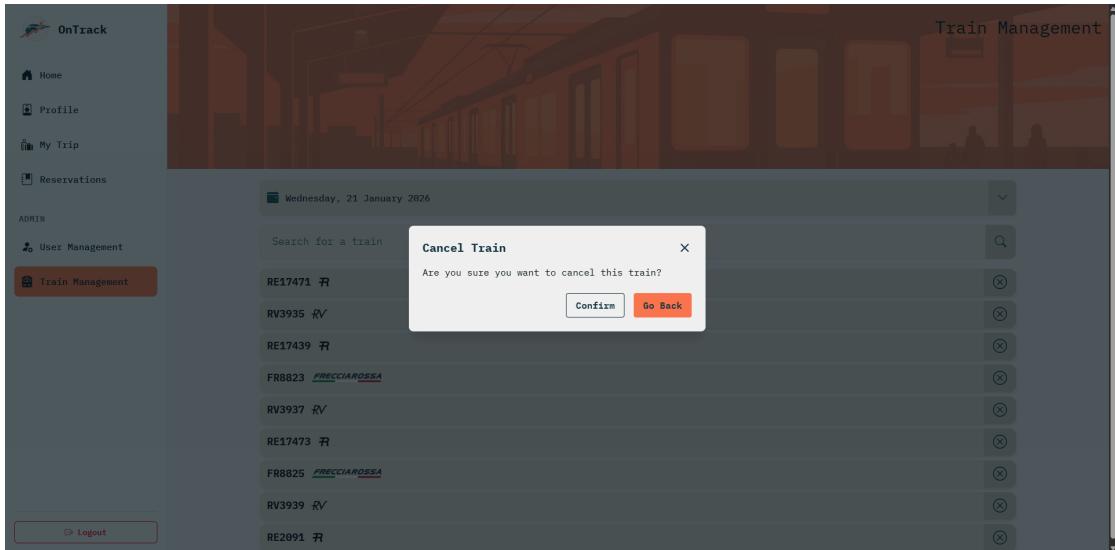
17

- RE17471 R (X)
- RE17439 R (X)
- RE17473 R (X)

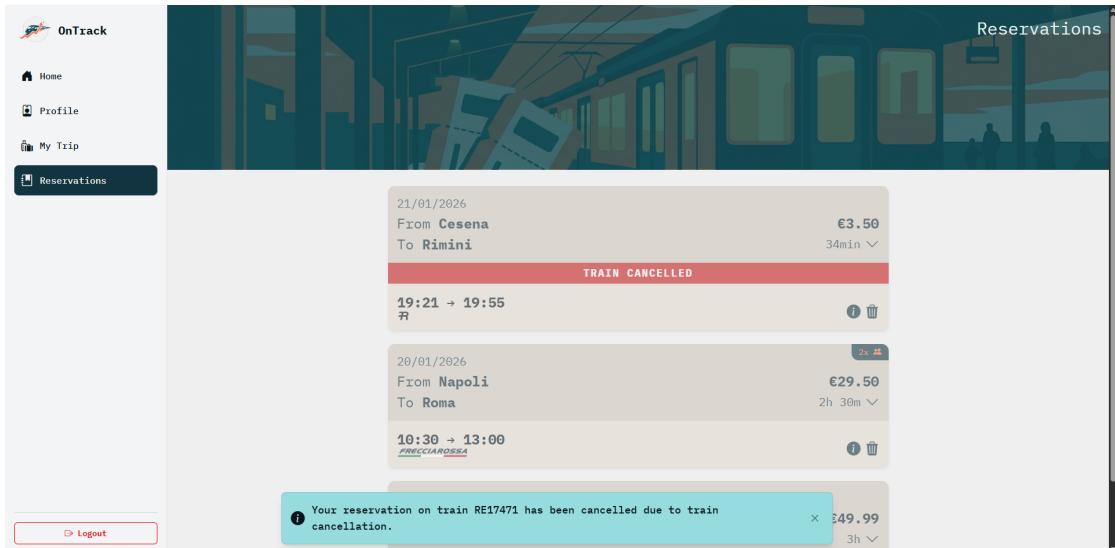
(b) Gestione Prenotazioni

(c) Gestione Treni

Figure 14: Admin



(a) Admin cancella treno



(b) Utente riceve notifica e aggiornamento stato prenotazione

Figure 15: Cancellazione treno



(a) On-Board Totem



(b) On-Board Admin

Figure 16: On-Board