EvenToNight

Applicazioni e Servizi Web

Federico Bravetti {federico.bravetti2@studio.unibo.it}
Tommaso Brini {tommaso.brini@studio.unibo.it}
Alice Alfonsi {alice.alfonsi2@studio.unibo.it}

16 febbraio 2026

Indice

Capitolo 1

Introduzione

Questo progetto ha portato alla realizzazione di una piattaforma digitale chiamata **EvenToNight**, pensata per mettere in contatto organizzazioni che promuovono eventi sociali e utenti interessati a scoprirli e parteciparvi.

La piattaforma è caratterizzata da un'interfaccia in stile social network, rendendo così l'esperienza di utilizzo semplice, intuitiva e coinvolgente.

L'applicazione consente agli utenti di:

- **Esplorare** eventi culturali, musicali e sociali
- **Acquistare biglietti** direttamente dalla piattaforma
- **Interagire** con le organizzazioni attraverso recensioni e chat
- **Salvare** e tenere traccia degli eventi di interesse

Per le organizzazioni, la piattaforma offre:

- **Visibilità** per i propri eventi
- **Gestione** completa di eventi e biglietteria
- **Analytics** sull'engagement degli utenti (like, follower)
- **Comunicazione** diretta con i partecipanti

Il progetto è stato sviluppato adottando un'architettura a microservizi, garantendo scalabilità, manutenibilità e separazione delle responsabilità.

Capitolo 2

Requisiti

2.1 2 — Requisiti

Di seguito sono riportati i principali requisiti che l'applicazione deve soddisfare.

2.1.1 2.1 — Requisiti di Business

- La piattaforma consente alle organizzazioni di creare e pubblicare post relativi agli eventi da loro promossi.
- Gli utenti possono utilizzare la piattaforma come punto di riferimento per scoprire eventi nelle vicinanze, in base alla località e ai propri interessi.
- Il sistema abilita la vendita online dei biglietti degli eventi fornendo alle organizzazioni uno strumento per monetizzare le proprie attività.

2.1.2 2.2 — Requisiti Funzionali

Tipologie di utenti supportate dal sistema:

- Utenti non registrati.
- Utenti registrati, che possono fruire dei contenuti della piattaforma.
- Utenti registrati come organizzazioni, che possono creare eventi, vendere biglietti e fruire dei contenuti della piattaforma.

Per tutti gli utenti:

- Visualizzare la schermata Home con le modalità di interazione: ricerca eventi, visualizzazione eventi popolari, prossimi eventi e nuove aggiunte.
- Visualizzare dalla schermata Esplora tutti gli eventi pubblicati sulla piattaforma, tutti gli utenti registrati e applicare filtri di ricerca.

Per utenti registrati:

- Ricevere un feed di eventi personalizzato, basato sugli interessi specificati.

- Mettere e togliere like a un evento.
- Mettere e togliere follow a un membro e a un'organizzazione.
- Acquistare biglietti per gli eventi.
- Lasciare una recensione dopo la partecipazione a un evento.
- Contattare direttamente le organizzazioni all'interno della piattaforma per richiedere supporto.
- Ricevere notifiche su:
  - nuovo follower.
  - pubblicazione di nuovo evento da parte di organizzazione seguita.
  - nuovo messaggio.

Per utenti registrati come organizzazioni:

- Creare eventi, scegliendo se renderli pubblici o salvarli come bozza.
- Specificare collaboratori durante la creazione degli eventi.
- Ricevere notifiche su like e recensioni ai propri eventi.

2.1.3 2.3 — Requisiti Non Funzionali

- Accessibilità: l'interfaccia grafica deve essere accessibile.
- Portabilità: l'applicazione risulta responsive per adattarsi a schermi di diverse dimensioni pc/tablet/mobile.
- Deployability: il sistema in automatico deve aggiornarsi alla versione dell'ultima release.
- Availability: il sistema deve essere tollerante ai guasti per garantire la disponibilità, deve poter effettuare un recupero automatico in caso di errore e prevedere la ridondanza dei componenti critici per assicurare la continuità del servizio.
- Sicurezza: gli utenti del sistema devono autenticarsi per verificare la loro identità e saranno poi autorizzati ad accedere alle risorse in base alle regole definite. Inoltre per assicurare la confidenzialità delle password queste saranno salvate in modo cifrato.
- Robustezza: l'applicazione deve gestire input errati e generare errori coerenti.
- Affidabilità: l'applicazione deve essere stabile, evitando crash.
- Manutenibilità: il codice deve essere ben strutturato e ben documentato.
- Estendibilità: il progetto deve favorire la personalizzazione e l'aggiunta di funzionalità.

Inoltre si è scelto, anche per esigenze didattiche, di aggiungere come requisito architetturale lo sviluppo del sistema con un'architettura a microservizi.

Capitolo 3

Design

3.1 3 — Design

3.1.1 3.1 — Metodologia progettuale

Lo sviluppo della piattaforma è stato condotto seguendo l'approccio *User Centered Design* (UCD), con l'obiettivo di progettare un sistema conforme ai principi HCI ottimizzando l'esperienza utente.

Per garantire la centralità dell'utente durante il design e lo sviluppo, non potendo coinvolgere utenti reali per l'intero ciclo progettuale, sono state adottate tecniche di virtualizzazione degli utenti promosse da UCD, quali la metodologia ***Personas*** combinata con gli **scenari d'uso**.

Analisi dei target user

Dopo aver delineato i requisiti di business della piattaforma, il team di sviluppo ha individuato il target di riferimento. La piattaforma è progettata per tre tipologie di utenti: utenti non registrati, utenti registrati e utenti registrati come organizzazione.

Per rappresentare le caratteristiche e i bisogni di ciascun gruppo di utenti del sistema, sono state create delle *Personas*. Per ogni *Personas* è stato inoltre simulato uno scenario d'uso della piattaforma, in modo da evidenziare le diverse modalità di interazione con il sistema e come questo possa rispondere efficacemente alle esigenze degli utenti.

Personas: Carlo

Carlo è un adulto appassionato di eventi culturali e spettacoli, ma fatica a scoprire nuove attività tramite i canali di informazione tradizionali. Non ama condividere i propri dati sul web e vuole trovare rapidamente eventi interessanti a cui partecipare, da solo o con amici e familiari.

Carlo ha bisogno di uno strumento semplice e immediato che gli permetta di esplorare gli eventi in programma che lo interessano, capire rapidamente

orari, luoghi e dettagli principali, senza dover registrarsi o inserire informazioni personali.

Scenario d'uso:

Carlo visita il sito della piattaforma senza registrarsi, scorre gli eventi in programma e consulta foto, descrizioni e informazioni pratiche come orario e luogo.

Grazie all'interfaccia chiara e semplice, Carlo può farsi un'idea immediata di quali eventi potrebbero interessargli e pianificare eventuali uscite. Pur senza account, ottiene tutte le informazioni necessarie e apprezza poter scoprire eventi diversi rispetto a quelli tradizionali, senza condividere dati personali.

Personas: Francesca

Francesca ha 22 anni e si è appena trasferita in una città universitaria vivace. Frequenta il primo anno della magistrale in Comunicazione Digitale e Marketing, divide il suo tempo principalmente tra lezioni e studio, e sente il desiderio di vivere esperienze che la aiutino a integrarsi nella nuova città.

Usa costantemente lo smartphone per restare aggiornata su eventi e tendenze locali. È attratta da contenuti visivi e immediati, come foto con una breve descrizione, che le permettono di percepire rapidamente l'atmosfera di un evento. Apre diverse app e social network alla ricerca di eventi, scorrendo post e pagine locali. Questo processo la stanca: le informazioni sono spesso sparse e incomplete e non sempre riesce a trovare esperienze che le interessano.

Francesca cerca uno strumento che le permetta di scoprire eventi nella nuova città in cui vive, in modo intuitivo e veloce.

Scenario d'uso:

Alla sera, terminato di studiare, Francesca accede alla piattaforma dal suo smartphone ed esplora i prossimi eventi in programma filtrando quelli vicino a lei.

Quando il post di un evento cattura la sua attenzione, controlla la pagina dell'organizzazione per vedere gli eventi passati e leggere le recensioni, così da capire il tipo di esperienze che l'organizzazione propone e valutare se l'evento possa essere affidabile e di suo gradimento. Essendo in una città nuova, queste informazioni la aiutano a scegliere con maggiore sicurezza.

Grazie alla piattaforma, Francesca riesce a trovare rapidamente eventi vicino a lei compatibili con i suoi interessi, ottenendo tutte le informazioni necessarie in un unico luogo e acquistando i biglietti in modo semplice e immediato come è abituata a fare in altre piattaforme.

Personas: Emma Lopez

Emma ha 27 anni e vive a Madrid. Sta organizzando un weekend in Italia con le sue amiche e vuole scoprire locali, eventi musicali autentici e buoni ristoranti italiani. Pianifica con attenzione ogni dettaglio ed è importante per lei trovare attività che soddisfino gli interessi di tutte le sue amiche.

Per cercare eventi e locali, Emma usa principalmente il computer. Naviga tra diversi siti, ma non ha un modo comodo e veloce per salvare e confrontare gli eventi che le interessano. Spesso deve tradurre siti italiani per capirne i dettagli. Si affida alle foto per farsi un'idea dell'evento, ma non sa sempre come scegliere quali eventi siano davvero interessanti. Vorrebbe prenotare qualcosa

per assicurarsi un posto e godersi la vacanza senza rischiare di perdere gli eventi più popolari.

Scenario d'uso:

Emma si registra sulla piattaforma che trova direttamente in lingua spagnola, essendo in Spagna. Dal suo computer esplora gli eventi filtrando in base alle sue preferenze e a quelle delle sue amiche, per selezionare quelli più adatti al gruppo. Può salvare direttamente i pos degli eventi che le interessano e vedere quanti like hanno ricevuto, aiutandola a capire quali sono più popolari. Una volta convinta, acquista i biglietti per lei e per le sue amiche, assicurandosi i posti.

Emma è contenta di poter consultare tutti gli eventi in programma in un unico sito già tradotto nella propria lingua e sfruttare le funzionalità della piattaforma per salvare, gestire, filtrare e prenotare i suoi eventi.

Personas: Simone

Simone ha 34 anni e gestisce un locale poco conosciuto in periferia. Crede nelle potenzialità del suo locale: l'ambiente è bello e accogliente, ma fatica a farlo conoscere. Il sito web del locale riceve poche visite e non permette di capire quanti utenti siano interessati agli eventi. Nel locale Simone organizza principalmente cene, ma gli piacerebbe collaborare con organizzatori di spettacoli dal vivo o show da ospitare nel suo locale per aumentare la visibilità e attrarre più clienti.

Simone ha bisogno di uno strumento che gli permetta di promuovere il suo locale, monitorare facilmente l'interesse degli utenti e collaborare con altri organizzatori in modo semplice ed efficace.

Scenario d'uso:

Simone si registra come organizzazione sulla piattaforma e crea il profilo del suo locale, rendendolo visibile a tutti gli utenti. Ogni volta che riceve un nuovo follower o che un suo evento ottiene un like, Simone riceve una notifica che gli fornisce un feedback immediato sull'interesse degli utenti. La piattaforma gli consente inoltre di collaborare con altre organizzazioni, definendo un calendario ricco di eventi per i prossimi mesi presso il suo locale.

Grazie alla piattaforma, Simone vede finalmente il suo locale apprezzato e frequentato.

Dall'analisi dei target user, attraverso le *Personas* e i relativi scenari d'uso, sono emersi i principali task che gli utenti desiderano svolgere sulla piattaforma. Queste informazioni hanno guidato la progettazione dell'interfaccia grafica.

3.1.2 3.2 — Progettazione dell'Interfaccia Grafica

In questa sezione viene descritta la progettazione dell'interfaccia grafica dell'applicazione.

L'obiettivo principale è stato quello di definire un design chiaro, intuitivo e coerente, in grado di guidare l'utente attraverso i principali flussi funzionali della piattaforma.

La sezione comprende:

- la presentazione dei **mockup** delle schermate principali;
- una panoramica degli **storyboard**, illustrando i principali flussi di interazione dell'utente.

Mockup

Per la fase iniziale di progettazione dell'interfaccia grafica sono stati creati dei **mockup**, con l'obiettivo di definire una prima proposta del possibile **look & feel** dell'applicazione, prima di passare all'implementazione vera e propria.

Seguendo un approccio **agile**, i mockup sono stati successivamente sostituiti da **demo funzionanti incrementali**, permettendo di testare e validare progressivamente le funzionalità dell'applicazione.

I mockup sono stati realizzati utilizzando Figma. In particolare, sono state sviluppate le schermate delle **tre aree principali** dell'applicazione:

- **Home**
- **Esplora**
- **Profilo**

Approccio Mobile-First La progettazione è stata effettuata seguendo l'approccio **mobile-first**, prendendo come modello di riferimento l'iPhone 16.

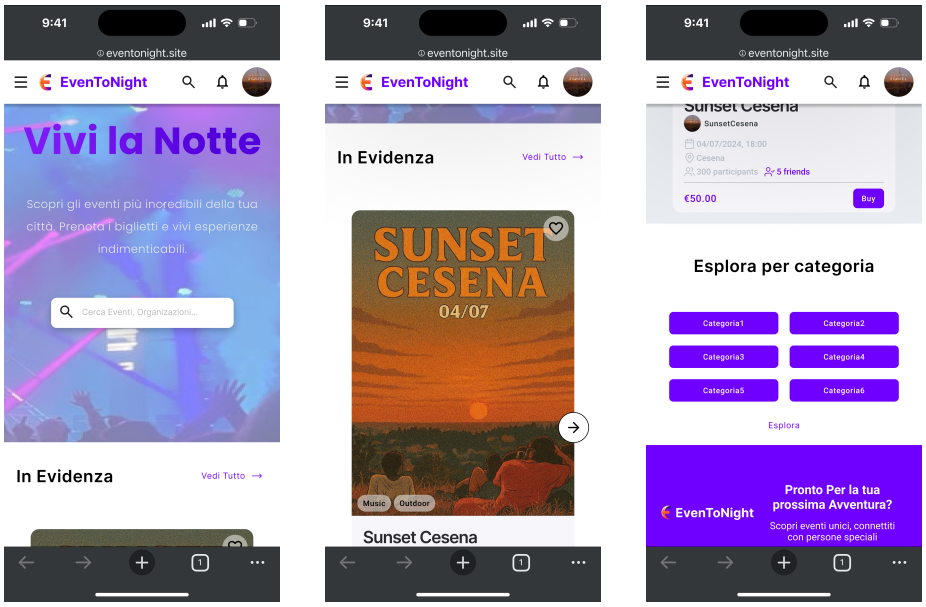
Questo approccio è stato scelto principalmente per due motivi:

1. **Vincoli più restrittivi**: il formato mobile costringe a dare priorità ai contenuti più importanti e a semplificare la navigazione, promuovendo un'interfaccia chiara e intuitiva che rispetta il principio **KISS**.
2. **Target principale da smartphone**: la maggior parte degli utenti accederà all'app tramite telefono, quindi è stata data priorità all'ottimizzazione del design per questo genere di dispositivi.

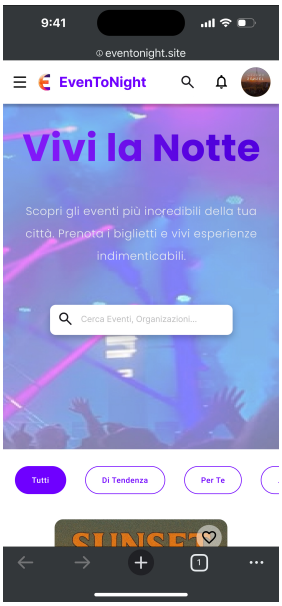
Nel design è stato inoltre seguito il principio **DRY,** riutilizzando ove possibile gli stessi componenti per garantire un aspetto coerente e familiare dell'applicazione.

Ad ogni modo il design è stato pensato e realizzato anche per essere responsive ed avere successivamente una buona resa anche su schermi desktop.

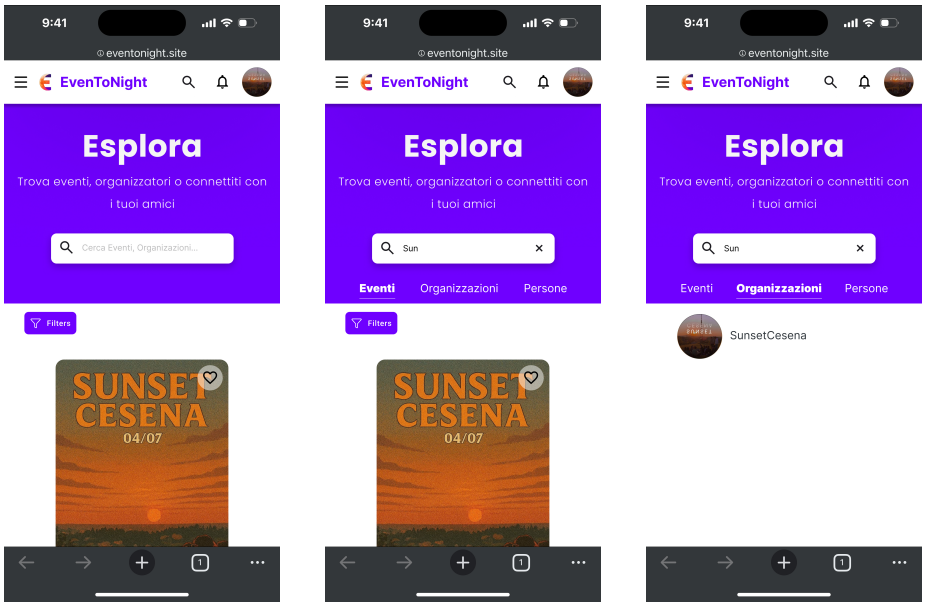
Home Di seguito è riportato il mockup per la schermata home. Questa è la schermata iniziale proposta all'utente, da cui potrà da subito cercare degli eventi o semplicemente vedere gli eventi proposti in vetrina. Da questa schermata l'utente potrà anche accedere o registrarsi alla piattaforma.



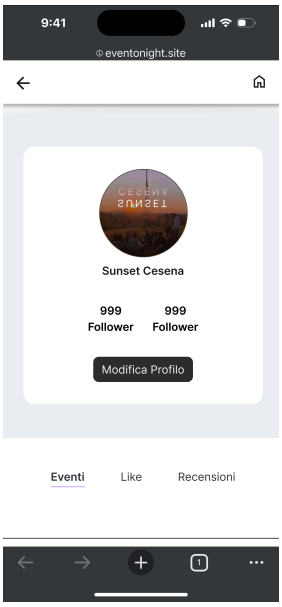
Inizialmente era stata anche proposta una versione alternativa con un diverso sistema di navigazione, che però è stata successivamente scartata vista la scarsa integrazione con il design dell'applicazione, in particolare in combinazione con la schermata **Esplora**.



Esplora Di seguito è riportato il design della sezione esplora. In questa sezione è possibile andare a visualizzare tutti gli eventi presenti sulla piattaforma e cercare anche tutti gli utenti per visualizzarne il profilo, seguirli e contattare le organizzazioni.



Profilo Da questa schermata l'utente avrà accesso alle sue informazioni, potrà modificare il suo profilo e le sue preferenze.



Storyboard

Di seguito sono riportati alcuni esempi di interazione con l'applicazione, che illustrano i principali flussi di utilizzo.

In particolare, vengono mostrati i seguenti casi d'uso:

- **Esplorare la piattaforma**
- **Login e Registrazione**
- **Creare un evento**
- **Partecipare ad un evento**
- **Recensire un evento**
- **Contattare un'organizzazione**

Le storyboard sono presentate direttamente utilizzando la piattaforma sviluppata, mostrando le principali interazioni dell'utente.

Nel progettare i flussi di navigazione si è sempre tenuto conto della **regola dei tre click**, cercando di rendere le principali funzionalità accessibili in pochi passaggi. Negli esempi riportati di seguito, alcune dinamiche di navigazione secondarie o ripetitive sono state omesse.

Esplorare la piattaforma

L'utente che apre la piattaforma può iniziare ad esplorarla, in particolare può scoprire gli eventi proposti o cercarli direttamente tramite la barra di ricerca. Nel caso in cui voglia visualizzare maggiori risultati può andare in una pagina dedicata all'esplorazione dei contenuti della piattaforma dove è possibile filtrare gli eventi e cercare in maniera più comoda utenti e organizzazioni.

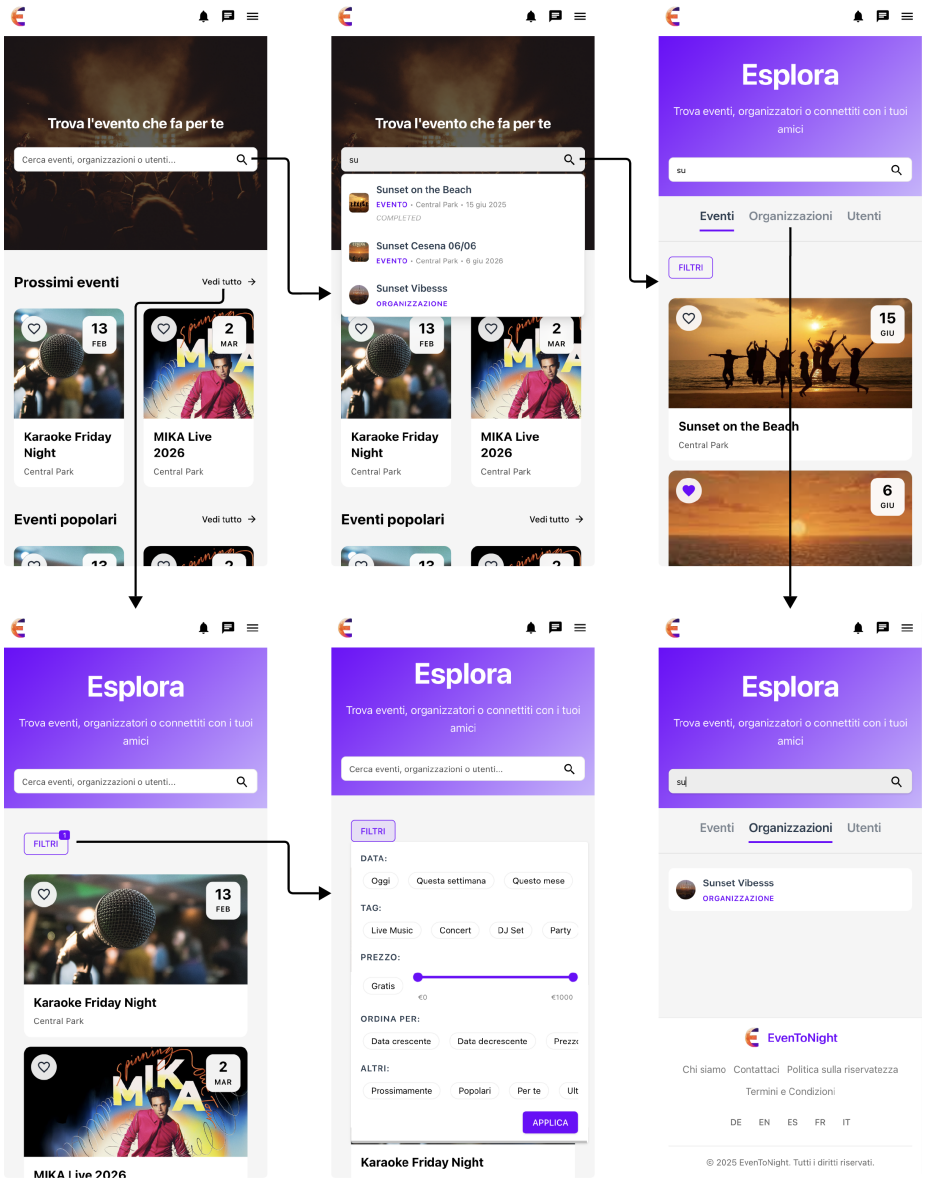


Figura 3.1: Storyboard Esplora

Login e Registrazione

Dopo aver aperto l'applicazione, l'utente per accedere alle funzionalità aggiuntive che la piattaforma offre può accedere o registrarsi.

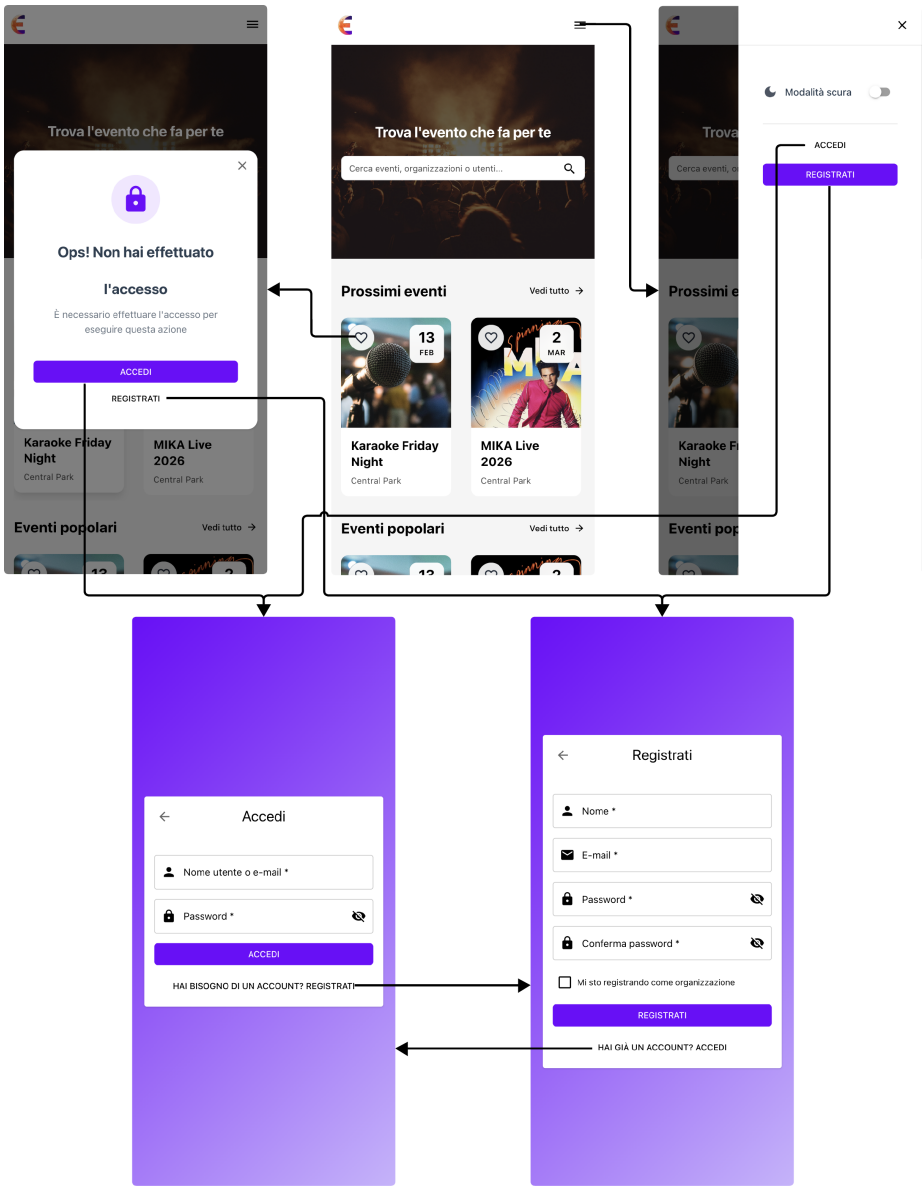


Figura 3.2: Storyboard Login

Creare un evento

Un'organizzazione che si è registrata sulla piattaforma ha la possibilità di creare degli eventi, la creazione avviene attraverso un form in cui inserire tutti i vari

dati. Inoltre è possibile anche temporaneamente creare una bozza dell'evento e continuare a modificarla successivamente.

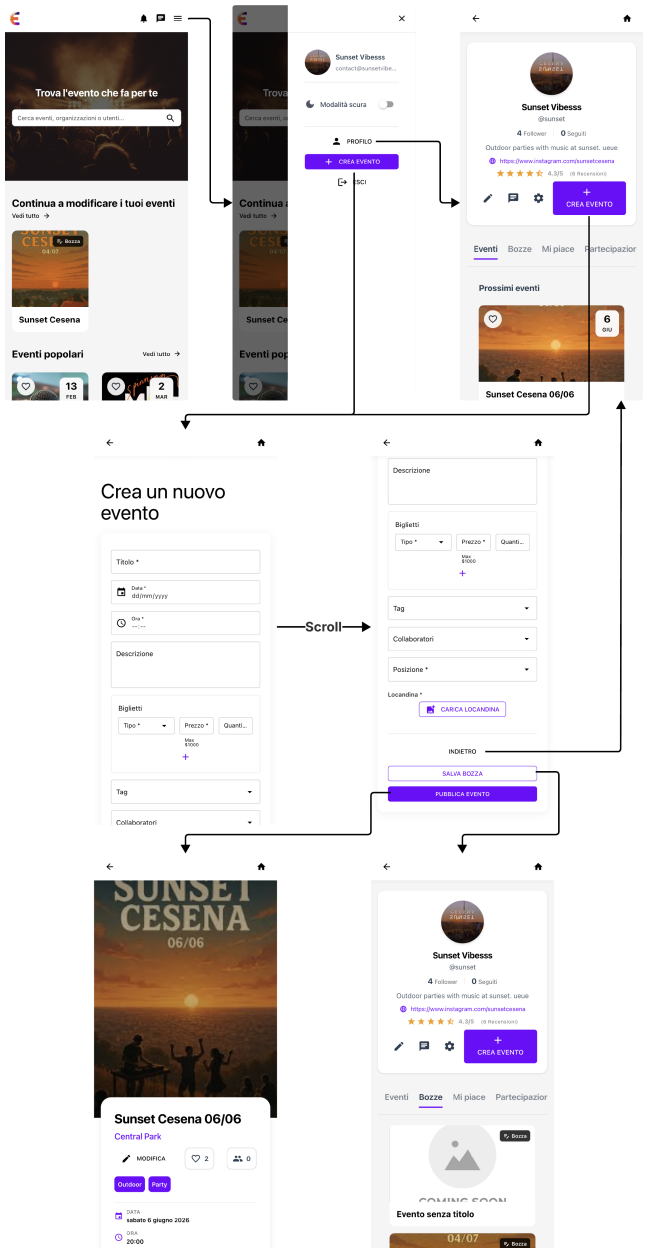


Figura 3.3: Storyboard Crea Evento

Partecipare ad un evento

Un utente registrato sulla piattaforma ha la possibilità di acquistare i biglietti per i diversi eventi e visualizzarli in seguito, i biglietti conterranno un QR code che può essere usato dalle organizzazioni per verificarli.

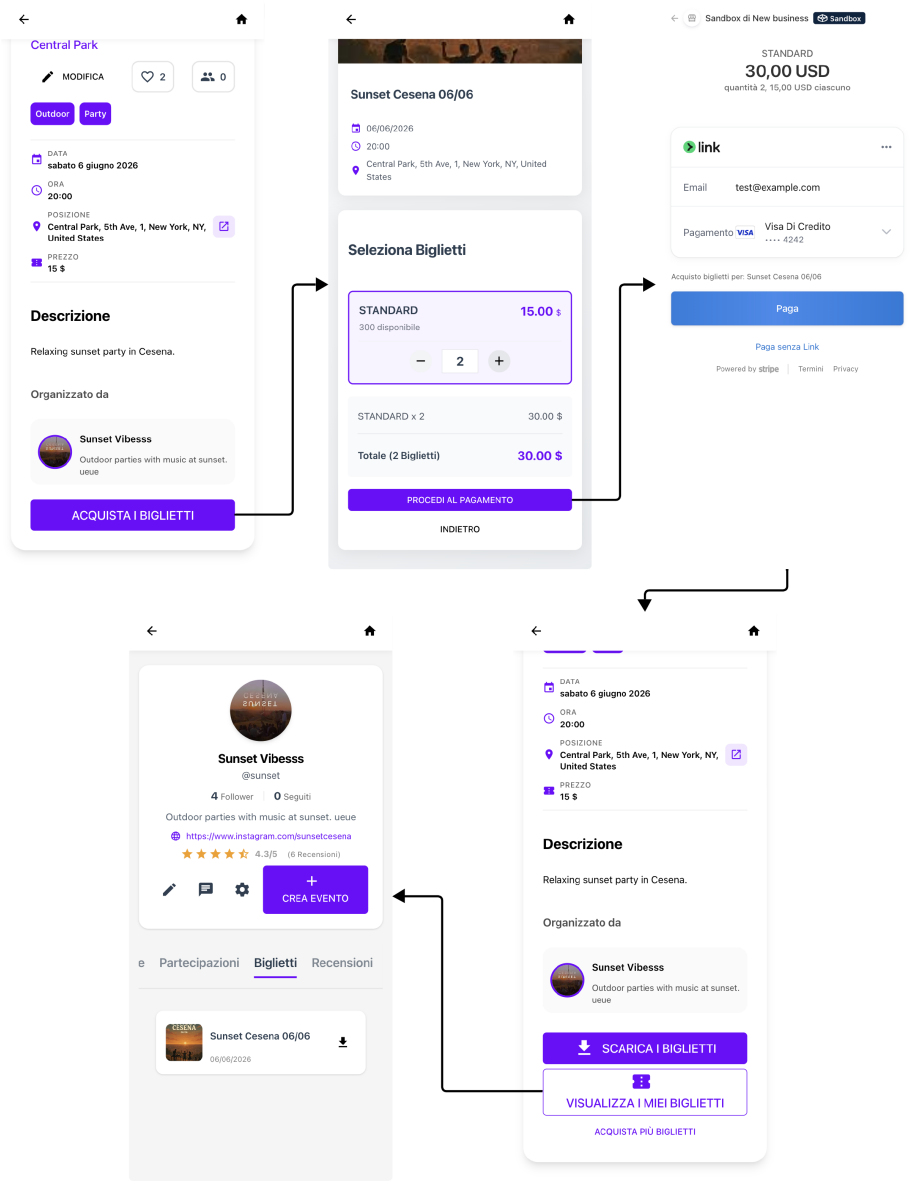


Figura 3.4: Storyboard Compra Biglietto

Recensire un evento

In seguito alla partecipazione ad un evento, un utente può decidere di lasciare una sua recensione. Una volta lasciata non ne può lasciare altre per lo stesso evento ma può modificarla o eliminarla.

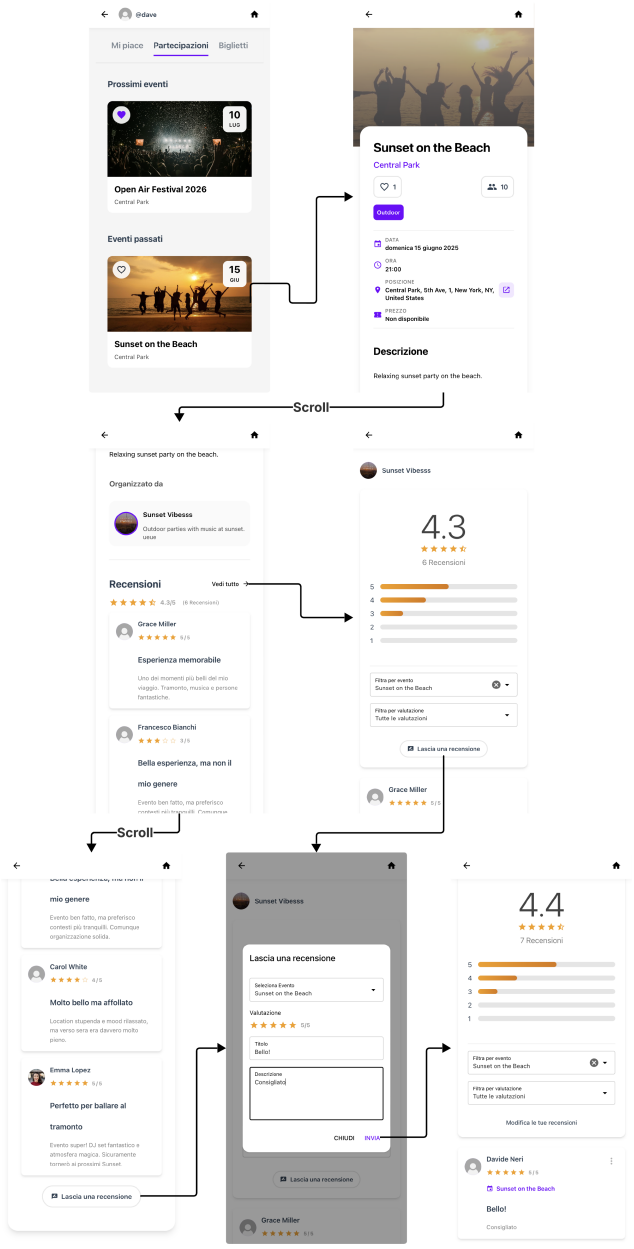


Figura 3.5: Storyboard Recensione

Contattare un'organizzazione

Un utente registrato può avere la necessità di contattare un'organizzazione per chiedere maggiori informazioni o per eventuali problemi.

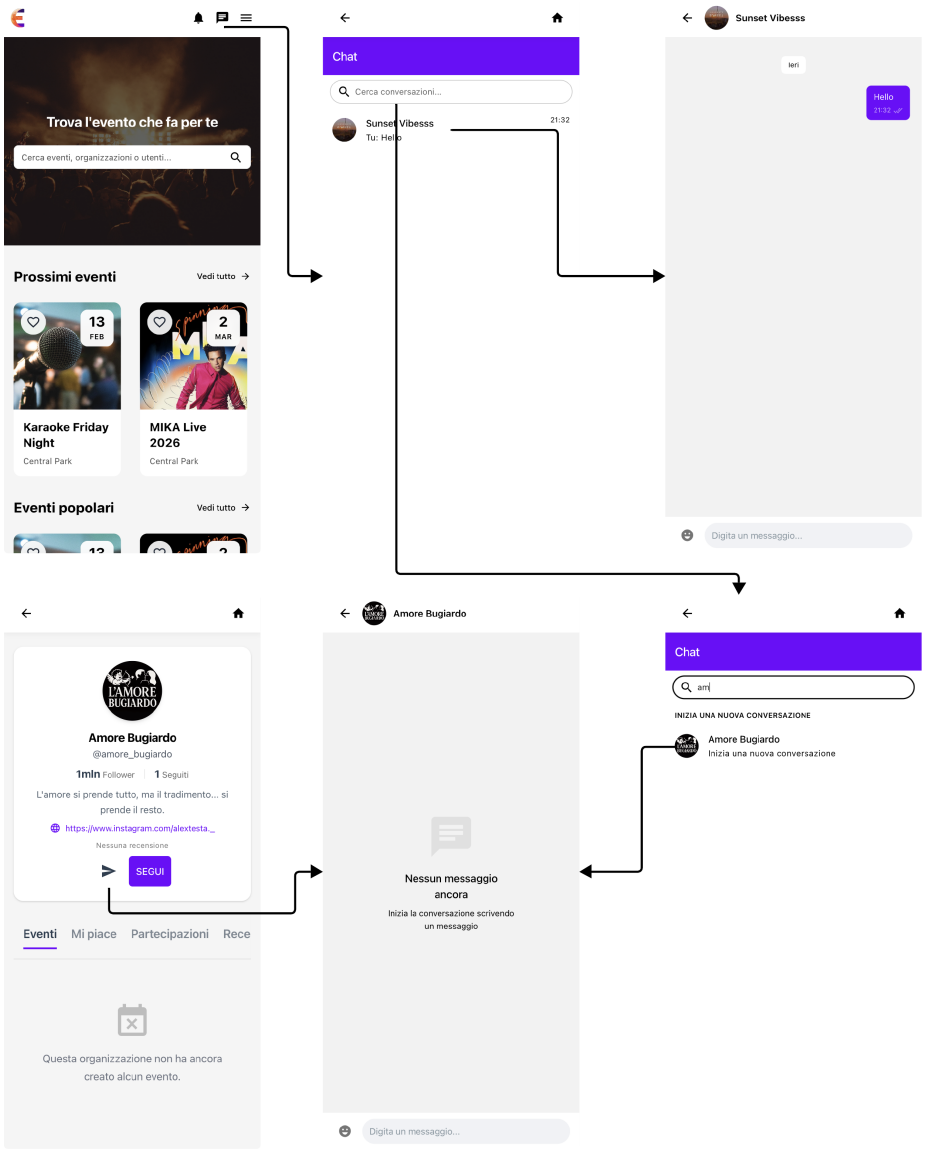


Figura 3.6: Storyboard Chat

3.1.3 3.3 — Dominio

Durante la fase iniziale di analisi sono state individuate le entità significative del dominio.

Il sistema si basa su due tipologie di utenti, membri e organizzazioni, che condividono una base comune ma possiedono funzionalità differenti.

I membri rappresentano gli utenti finali della piattaforma e interagiscono con gli eventi pubblicati dalle organizzazioni. Possono esplorare gli eventi applicando filtri quali data, città, prezzo, esprimere interesse tramite like e partecipare agli eventi acquistandone i relativi biglietti. Dopo aver partecipato all'evento, gli utenti possono lasciare recensioni, contribuendo alla valutazione e alla popolarità dell'organizzazione.

Inoltre, possono seguire altri utenti e comunicare direttamente con le organizzazioni tramite messaggi.

Le organizzazioni, oltre alle funzionalità comuni ai membri, hanno la possibilità di creare e gestire eventi, definendone le informazioni e rendendoli disponibili sulla piattaforma. Possono inoltre ricevere recensioni da utenti che hanno partecipato ai loro eventi.

Tutti gli utenti ricevono delle notifiche a seguito di eventi di dominio che li riguardano.

Da questa analisi emergono le principali entità del dominio, tra cui utenti, eventi, biglietti, notifiche e interazioni, che costituiscono la base per il design delle API e dell'architettura del sistema.

3.1.4 3.4 — Design delle risorse

A partire dall'analisi del dominio, sono state definite come principali risorse gli utenti e gli eventi.

Oltre alle risorse principali, il sistema modella ulteriori risorse rilevanti, come:

- **tickets**, associati agli eventi
- **conversations**, per la comunicazione tra utenti
- **interactions**, che aggregano like, review e follow
- **notifications,** associate agli utenti

Questo approccio permette di mantenere un modello REST coerente e facilmente estendibile.

/users

La risorsa /users rappresenta gli utenti del sistema ed è modellata secondo l'archetipo **collection**, mentre /users/{userId} rappresenta una risorsa **document**.

Ad ogni utente sono associate anche ulteriori **sotto-risorse** (altre **collection**), tra cui:

- /users/{userId}/likes
- /users/{userId}/reviews
- /users/{userId}/followers
- /users/{userId}/following
- /users/{userId}/events

- /users/{userId}/conversations
- /users/{userId}/notifications

Alcuni endpoint seguono l'archetipo del **controller** in quanto escono dal classico paradigma RESTfull indicando delle azioni, ad esempio: — /users/login — /users/register

/events

La risorsa /events rappresenta l'insieme degli eventi disponibili nel sistema (archetipo **collection**), mentre /events/{eventId} rappresenta un singolo evento (**document**).

Ad ogni evento sono associate anche ulteriori **sotto-risorse** (altre **collection**), tra cui:

- /events/{eventId}/participants
- /events/{eventId}/likes
- /events/{eventId}/reviews
- /events/{eventId}/tickets

Su alcuni endpoint vengono utilizzati dei query params per filtrare i dati delle collections ed effettuare una richiesta paginata tramite limit e offset, ad esempio:

- /events/search?query=sunset&limit=10&offset=0&orderBy=date

3.1.5 3.5 — Design delle API

Le API sono state progettate seguendo i principi del REST API Design, con particolare attenzione all'uso corretto dei metodi HTTP, all'utilizzo di sostantivi negli url e al rispetto degli archetipi REST.

Alcuni esempi di API implementate seguendo i principi citati:

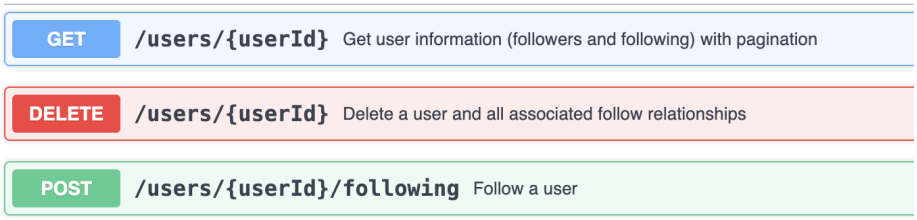


Figura 3.7: Alcune API per la collection Users

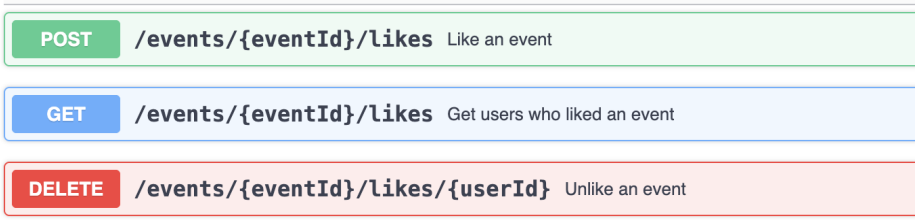


Figura 3.8: Alcune API per la collection Events

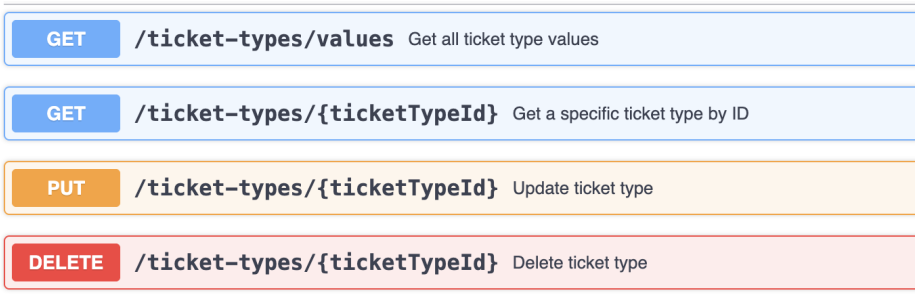


Figura 3.9: Alcune API per la collection Ticket-Types

Ogni microservizio espone una documentazione **Swagger/OpenAPI**, che descrive endpoint, input e possibili risposte HTTP.

Swagger è stato utilizzato sia come strumento di documentazione che come supporto al testing manuale delle API. Di seguito il link `https://eventonight.github.io/EvenToNight/openAPI/`

3.1.6 3.6 — Architettura

L'architettura del sistema è basata su un insieme di microservizi indipendenti, ciascuno responsabile di un sotto-dominio applicativo specifico.

Il frontend rappresenta il punto di accesso per gli utenti e comunica esclusivamente con i servizi backend tramite API REST e socket, senza accesso diretto ai database.

Ogni servizio infatti possiede la propria logica di business, entità comuni a più servizi (es. gli eventi) vengono rappresentati in maniera diversa in ognuno di essi.

Ogni servizio è containerizzato tramite Docker, comunica con gli altri servizi principalmente tramite messaggi asincroni ed espone un insieme coerente di API REST.

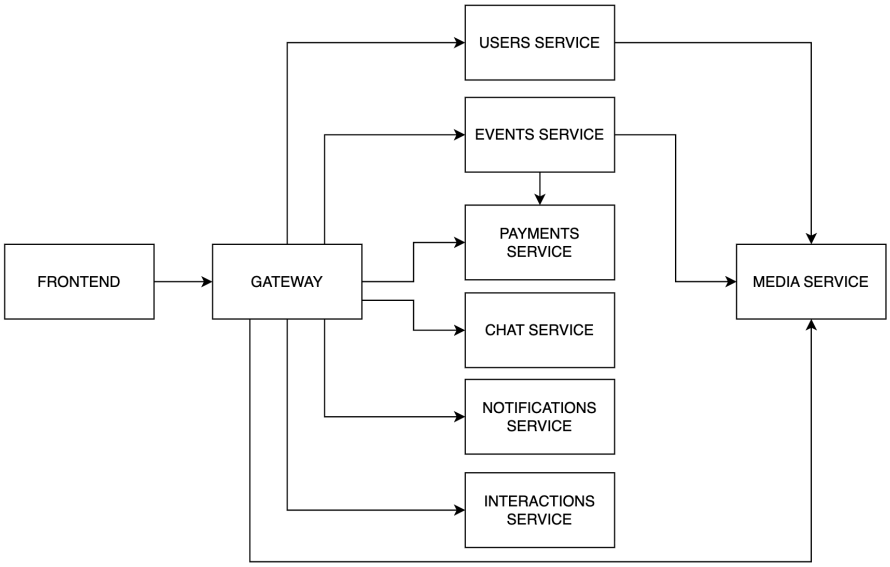


Figura 3.10: Panoramica dell' architectura

Le risorse individuate nella fase precedente sono state organizzate e distribuite nei vari servizi, di seguito una breve descizione.

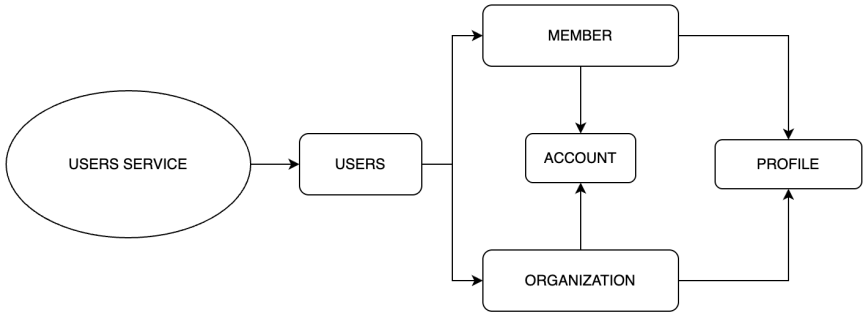


Figura 3.11: Archiettura del servizio Users

Il servizio *users* è responsabile della gestione delle risorse utente e di una parte delle relative sotto-risorse. Per ogni utente, il sistema gestisce l'auenticazione e le informazioni riguardanti l'accoun (e.g username, email, interessi) e il profilo (e.g. nome, bio).

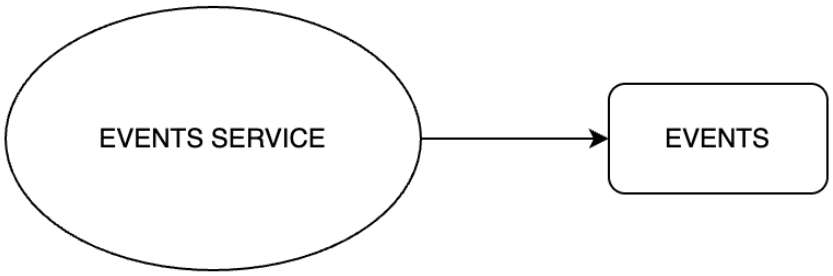


Figura 3.12: Archiettura del servizio Events

Il servizio *events* è responsabile della gestione delle risorse eventi e delle loro informazioni. Gestisce sia gli aspetti di creazione degli eventi sia il recupero degli eventi filtrati in base a caratteristiche specificate (e.g.popolari, interessi).

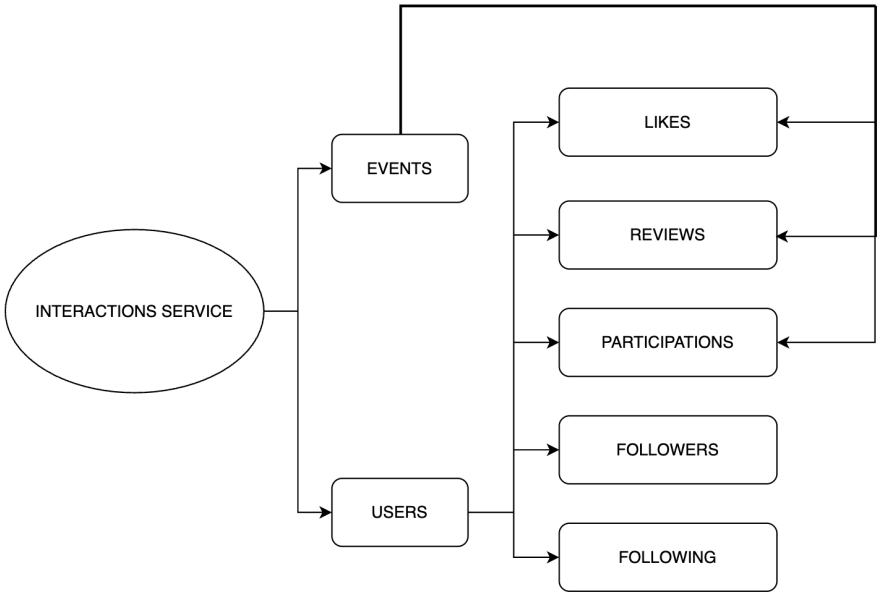


Figura 3.13: Archiettura del servizio Interactions

Il servizio *interactions* è responsabile di alcune sotto-risorse degli utenti e degli eventi. In particolare, gestisce tutte le informazioni riguardanti le interazioni

che un utente può avere con un evento o con un altro utente. Tra le interazioni con gli eventi sono stati implementati i likes, le reviews e le partecipazioni, mentre con gli altri utenti è stato implementato un sistema di following.

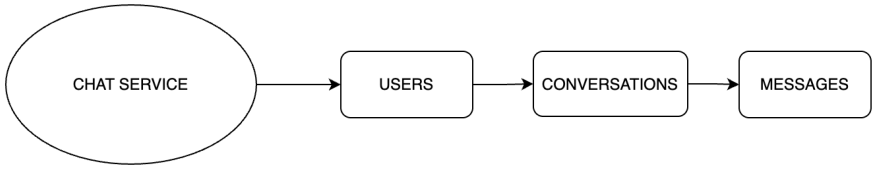


Figura 3.14: Archiettura del servizio Chat

Il servizio *chat* è responsabile di alcune sotto-risorse degli utenti. In particolare, gestisce tutte le informazioni riguardanti le conversazioni che l'utente può avere con altri utenti.

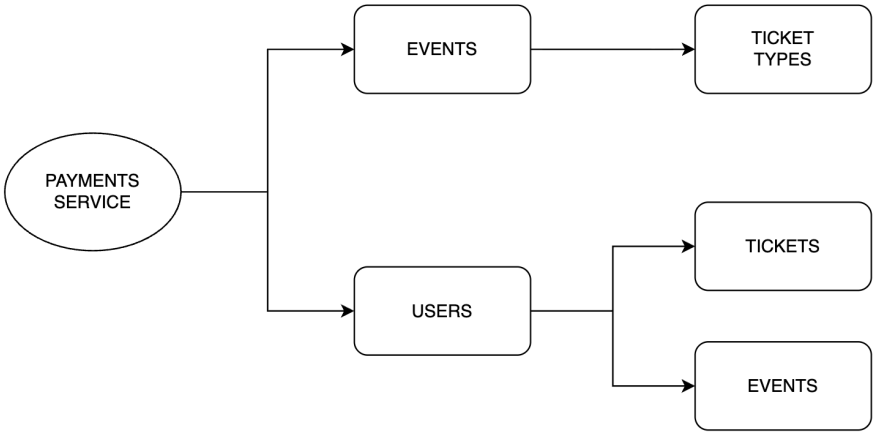


Figura 3.15: Archiettura del servizio Payments

Il servizio payments è responsabile di alcune risorse degli utenti e degli eventi, in particolare della gestione dei biglietti che gli utenti possono acquistare per partecipare agli eventi.

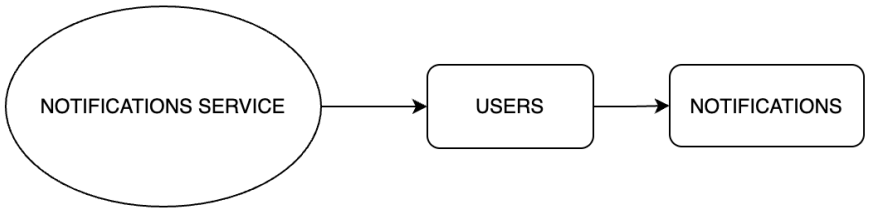


Figura 3.16: Archiettura del servizio Notifications

Il servizio *notifications* è responsabile di alcune sotto-risorse degli utenti, in particolare gestisce tutte le informazioni riguardanti le notifiche che un utente riceve. Le notifiche possono essere di tipi diversi, quali like ricevuto, review ricevuta, nuovo follower o nuovo evento creato da un'organizzazione seguita. Si occupa anche di notificare l'arrivo di un nuovo messaggio, gestito poi dal servizio chat.

3.1.7 3.7 — Comunicazione: RabbitMQ e Socket.IO

Per la comunicazione tra servizi abbiamo utilizzato un broker di messaggi, in particolare RabbitMQ. Ogni servizio definisce alcuni *domain event (e.g. nuovo evento pubblicato, like aggiunto, utente creato)* e li comunica sull'exchange *eventonight*, al quale sono connesse e in ascolto le code dei vari servizi.

Per funzionalità che richiedono aggiornamento in tempo reale (come le notifiche e le chat) abbiamo utilizzato una comunicazione basata su socket centralizzandola nel servizio notifications in modo da aprire una sola connessione con ciascun client. Il servizio notifications tramite RabbitMQ ascolta tutti i *domain event* che vogliamo comunicare real-time, li elabora e quando necessario li inoltra nei canali degli utenti connessi interessati.

Capitolo 4

Tecnologie

4.1 4 — Tecnologie

Durante lo sviluppo del progetto sono state utilizzate molteplici tecnologie, sia per esigenze implementative sia per scopi di apprendimento.

Per realizzare il frontend è stato utilizzato il framework Vue.js.

Per la realizzazione del backend invece, sono stati utilizzati diversi framework. Oltre allo stack MEVN infatti abbiamo utilizzato NestJS per la gestione di alcuni servizi, Cask per implementare le API in Scala, Socket.IO per la comunicazione server-client, RabbitMQ per la comunicazione tra servizi.

Per la gestione dei pagamenti è stato utilizzato Stripe (in modalità SandBox), per il salvataggio delle immagini è stato utilizzato MinIO.

Per il deploy è stato utilizzato Docker, e tramite i tunnel di CloudFlare è stato esposto in rete.

Tecnologie:

- Vue.js
- MongoDB
- Express.js
- Node.js
- Socket.IO
- Gradle
- Docker
- RabbitMQ
- Traefik
- NestJS
- MinIO
- Cask
- Swagger — OpenAPI

Servizi esterni

- Keycloak

- Stripe
- CloudFlare

Linguaggi

- Typescrip
- Scala

Capitolo 5

Codice

5.1 5 — Codice

L'architettura del progetto **EvenToNight** si basa su una **Single Page Application (SPA)** per il frontend e un layer di **microservizi backend** esposti attraverso **Traefik** come reverse-proxy/API gateway.

5.1.1 5.1 — Frontend

Il frontend è stato sviluppato con **Vue 3** utilizzando la **Composition API** e **TypeScript**, sfruttando il framework **Quasar** per i componenti UI.

L'applicazione fa uso delle principali funzionalità di Vue come `provide/ inject`, `props/ emit`, `defineModel`, `defineExpose`, `watchers` e in alcune situazioni la direttiva `:key` per innescare il refresh dei componenti.

Struttura del Progetto

```
/src/
|--- api/                # Layer di astrazione API con supporto mock
|   |--- adapters/       # Adapter per allinare i dati ricevuti dalle
↪  API
|   |--- mock-services/  # Implementazioni mock per sviluppo
|   |--- services/       # Implementazioni API dei servizi
|   `--- client.ts       # Client HTTP con gestione JWT
|--- components/         # Componenti Vue riutilizzabili
|--- composables/        # Funzionalità riutilizzabili dai vari
↪  componenti
|--- i18n/               # File di lingua
|--- layouts/            # Layout condivisi
|--- router/             # Configurazione routing e guards
|--- stores/             # Pinia store
`--- views/              # Pagine dell'applicazione
```

Gestione dello Stato con Pinia

Lo store Pinia gestisce l'autenticazione dell'utente e i token JWT. Il sistema implementa il refresh automatico dei token prima della scadenza:

```
interface Tokens {
  accesToken: AccessToken,
  refreshToken: RefreshToken,
  refreshExpiresAt: number,
}

export const useAuthStore = defineStore('auth', () => {
  const user = ref<User | null>(null)
  const tokens = ref<Tokens | null> (null)

  const isAuthenticated = computed(() => {
    if (!tokens.value || !user.value) return false
    return tokens.refreshExpiresAt.value > Date.now()
  })

  // Set data to local storage to avoid data loss on page refresh
  const setAuthData = async (authData: LoginResponse) => {
    setTokens(authData)
    setUser(authData.user)
    setupAutoRefresh()
    await api.notifications.connect(authData.user.id,
    ↪  authData.accessToken) //Connect to Socket
  }

  // Restores auth data from session storage (if any)
  const refreshCurrentSessionUserData = () => {}

  // Auto-refresh 5 minutes before expiration
  const setupAutoRefresh = () => {
    const refreshTime = tokens.value.refreshExpiresAt - Date.now() - 5
    ↪  * 60 * 1000
    if (refreshTime > 0) {
      setTimeout(() => refreshAccessToken(), refreshTime)
    }
  }
})
```

Layer API e Mocking Strategy

Un aspetto fondamentale dello sviluppo è stato il **layer di astrazione API**, che ha permesso di prototipare il frontend indipendentemente dalla disponibilità dei microservizi backend.

Il sistema utilizza una variabile d'ambiente per utilizzare API reali o mock:

```
const useRealApi: boolean = import.meta.env.VITE_USE_MOCK_API ===
↪  'false'

export const api = {
  events: useRealApi ? createEventsApi(createEventsClient()) :
↪  mockEventsApi,
  chat: useRealApi ? createChatApi(createChatClient()) : mockChatApi,
  notifications: useRealApi
    ? createNotificationsApi(createNotificationsClient())
    : mockNotificationsApi,
  // other services
}
```

Il client HTTP gestisce centralmente l'injection del token JWT e il refresh automatico in caso di 401:

```
const token = tokenProvider?.()
if (token) {
  headers['Authorization'] = `Bearer ${token}`
}

// Refresh after receiving 401
if (response.status === 401 && !isRetry && onTokenExpired) {
  const refreshed = await onTokenExpired()
  if (refreshed) {
    return this.request(endpoint, options, true)
  }
}
```

Questo approccio, seguendo il principio **Dependency Inversion (DIP)**, ha permesso di sviluppare componenti UI indipendenti dall'implementazione concreta delle API e ha facilitato il testing.

Router e Navigation Guards

Il sistema di routing utilizza due livelli: un livello root per gestire redirect e rotte speciali, e un livello nested sotto `/:locale` per tutte le rotte localizzate. Questa separazione è stata necessaria perché alcune rotte (come quella codificata nel QR code nei biglietti) non richiedono il prefisso della lingua.

```
const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/',
      redirect: () => `/${getInitialLocale()}`,
```

```
    },
    {
      path: '/verify/:ticketId',
      redirect: (to) =>
↪  `/${getInitialLocale()}/verify/${to.params.ticketId}`,
    },
    {
      path: '/:locale',
      component: LocaleWrapper,
      children: [
        { path: '', name: 'home', component: Home },
        { path: 'login', name: 'login', component: () =>
↪  import('../views/AuthView.vue'), beforeEnter: requireGuest },
        { path: 'events/:id', name: 'event-details', component: () =>
↪  import('../views/EventDetailsView.vue'), beforeEnter:
↪  requireNotDraft },
        { path: 'create-event', name: 'create-event', component: () =>
↪  import('../views/CreateEventView.vue'), beforeEnter:
↪  requireRole('organization') },
        // ... other routes
      ],
    },
  ],
})

// Global guard for i18n synchronization
router.beforeEach((to, _from, next) => {
  const locale = to.params.locale as string

  // Redirect if locale is not supported
  if (locale && !SUPPORTED_LOCALES.includes(locale)) {
    return next(`/${DEFAULT_LOCALE}${to.path.substring(locale.length +
    ↪  1)}`)
  }

  // Restore saved locale preference on navigation
  const savedLocale = localStorage.getItem('user-locale')
  if (savedLocale && locale && locale !== savedLocale &&
  ↪  SUPPORTED_LOCALES.includes(savedLocale)) {
    return next({
      name: to.name as string,
      params: { ...to.params, locale: savedLocale },
      query: to.query,
      replace: true,
    })
  }

  // Sync i18n with URL locale
  if (locale && i18n.global.locale.value !== locale) {
```

```
    i18n.global.locale.value = locale as Locale
    localStorage.setItem('user-locale', locale)
  }

  next()
})
```

Le **navigation guards** proteggono le rotte in base all'autenticazione e ai ruoli:

```
export const requireAuth = (to, _from, next) => {
  const authStore = useAuthStore()
  if (!authStore.isAuthenticated) {
    next({ name: LOGIN_ROUTE_NAME, query: { redirect: to.fullPath } })
  } else {
    next()
  }
}

export const requireRole = (role: string) => {
  return (to, from, next) => {
    const authStore = useAuthStore()
    if (authStore.user?.role !== role) {
      next({ name: FORBIDDEN_ROUTE_NAME })
    } else {
      next()
    }
  }
}
```

Le guards guidano l'utente nell'utilizzo dell'applicazione, impedendo l'accesso a pagine non autorizzate e reindirizzandolo automaticamente (ad esempio, se un organizzazione vuole creare un evento ma non ha effettuato l'accesso, prima si ha un redirect alla pagina di login).

Comunicazione Real-time con WebSocket

La comunicazione in tempo reale è gestita tramite **Socket.IO** per le notifiche e i messaggi chat:

```
socket = io(url, {
  auth: { token, userId },
  reconnection: true,
  reconnectionAttempts: 5,
  transports: ['websocket', 'polling'],
})

socket.on('connect', () => {
```

```
  handlers.forEach(({ handler, eventType }) => {
    socket?.on(eventType, handler)
  })
})
```

Gli eventi gestiti includono:

- `user-online` / `user-offline` — Stato online degli utenti
- `new-message` — Nuovi messaggi in chat
- `like-received` / `follow-received` — Notifiche di interazione
- `new-event-published` — Nuovi eventi pubblicati da utenti seguiti

Composables Riutilizzabili

I composables incapsulano logiche riutilizzabili tra i componenti. Un esempio significativo è `useInfiniteScroll` per la paginazione:

```
export function useInfiniteScroll<R>(config:
↪  InfiniteScrollConfiguration<R>) {
  const items: Ref<R[]> = ref([])
  const hasMore = ref(true)
  const loading = ref(true)

  const onLoad = async (_index: number, done: (stop?: boolean) =>
  ↪  void) => {
    if (!hasMore.value) {
      done(true)
      return
    }
    await loadItems(true)
    done(!hasMore.value)
  }

  return { items, hasMore, loading, onLoad, reload }
}
```

Altri composables includono:

- `useUserProfile` — Logica profilo utente (isOwnProfile, isOrganization)
- `useDarkMode` — Gestione tema chiaro/scuro con persistenza
- `useTranslation` — Traduzioni con prefisso automatico

Layout Condivisi

Sono stati definiti anche layout riutilizzabili per garantire consistenza nell'interfaccia:

- **NavigationWithSearch**: Utilizzato in home ed explore, serve a gestire la comparsa della barra di ricerca nella barra di navigazione dal momento che esce dalla viewport e viceversa.

- **TwoColumnLayout**: Utilizzato in chat e impostazioni, con supporto mobile che mostra una colonna alla volta

Internazionalizzazione (i18n)

L'applicazione supporta 5 lingue (en, es, fr, it, de). Le traduzioni sono generate automaticamente in CI a partire dal sorgente inglese:

```
const localeModules = import.meta.glob('./locales/*.ts', { eager:
↪  true })

const i18n = createI18n({
  legacy: false,
  locale: DEFAULT_LOCALE,
  fallbackLocale: DEFAULT_LOCALE,
  messages,
})
```

L'utilizzo nei componenti avviene tramite il composable useTranslation:

```
const { t } = useTranslation('components.cards.EventCard')
```

Inoltre il selettore delle lingue nelle impostazioni del profilo utilizza l'API nativa `Intl.DisplayNames` per mostrare ogni lingua nel proprio nome nativo (es. "Italiano", "Français", "Deutsch"), migliorando l'accessibilità per gli utenti:

```
const getLanguageInfo = (code: string): LanguageOption => {
  const nativeNames = new Intl.DisplayNames([code], { type:
  ↪  'language' })
  return {
    code,
    nativeName: nativeNames.of(code) || code.toUpperCase(),
    flag: getFlagEmoji(code),
  }
}
```

L'internazionalizzazione si estende anche al backend: il servizio Payments genera i **biglietti PDF nella lingua dell'utente**, con traduzioni dedicate.

Il backend predispone inoltre la gestione dei prezzi con le valute e un sistema di conversione con caching, attualmente non utilizzato dal frontend ma pronto per future estensioni:

```
export class CurrencyConverter {
  private static readonly CACHE_DURATION = 24 * 60 * 60 * 1000 // 24
  ↪  hours
```

```
  static async convertAmount(amount: number, fromCurrency: string,
  ↪  toCurrency: string): Promise<number> {
    const fromRates = await this.fetchRatesForCurrency(from)
    return this.converter.convert(amount, from, to, fromRates)
  }
}
```

Limiti attuali: I contenuti user-generated come le descrizioni degli eventi non sono attualmente tradotti e vengono memorizzati nella lingua in cui sono stati inseriti dall'organizzazione. Un'estensione futura potrebbe prevedere la possibilità di inserire descrizioni in più lingue, con fallback alla lingua originale quando la traduzione non è disponibile.

Geolocalizzazione degli eventi

Per l'inserimento della posizione durante la creazione di un evento è stata utilizzata l'API pubblica di **Nominatim** (OpenStreetMap) per la ricerca e il geocoding degli indirizzi. La scelta di OpenStreetMap è stata dettata dalla sua natura open-source e dall'assenza di costi di utilizzo.

Poiché la maggior parte degli utenti utilizza Google Maps per la navigazione, i dati ricevuti da Nominatim vengono elaborati per generare link compatibili con Google Maps, permettendo all'utente di cliccare sulla posizione dell'evento e aprirla direttamente nell'applicazione di navigazione.

```
export const extractLocationMapsLink = (location: LocationData): string
↪  => {
  const query = `${location.name},${location.road},${location.city},
  ↪  ${location.country}`
  return `https://www.google.com/maps/search/?api=1&query=${
  ↪  encodeURIComponent(query)}`
}
```

5.1.2 5.2 Backend

Il backend è composto da **7 microservizi**: 2 sviluppati in **Scala 3** con il framework **Cask** (Users ed Events), 3 in **NestJS** (Interactions, Chat e Payments) e 2 in **Express.js** (Notifications e Media).

Ogni servizio ha la propria istanza **MongoDB** dedicata e comunica con gli altri tramite **RabbitMQ** con topic exchange. Il servizio Notifications gestisce le connessioni WebSocket tramite **Socket.IO** per le notifiche real-time e il tracciamento dello stato online degli utenti.

Struttura del Progetto

Essendo i microservizi eterogenei segue una descrizione di massima rappresentativa della struttura delle varie implementazioni, viene usata la terminologia

dello stack MEVN ma alcuni componenti potrebbero avere un nome/implementazione diversa (e.i. in Nest il concetto di router e controller viene unito in un unico componente rispetto ad express).

```
/src/
|--- presentation/        # Layer di ingresso al servizio
|--- application/         # Layer contenenti le logiche applicative
|--- domain/              # Modello dei dati
`--- infrastructure/      # Dipendenze del dominio
```

Più nel dettaglio, nel primo layer di presentazione troviamo i router che definiscono le rotte (path + metodo http) supportate da ciascun servizio. In application sono specificati i diversi DTO usati per validare i dati in ingresso agli endpoint e strutturare i dati di risposta degli stessi e i controller delle varie rotte. Nel dominio sono definiti i modelli delle entità di interesse per il particolare servizio e in infrastructure troviamo le varie dipendenze esterne, tipicamente le implementazioni dei connectors a database e message-broker

Autenticazione JWT

L'autenticazione è basata su **JWT**. Il servizio Users integra **Keycloak** come identity provider per la gestione delle credenziali utente (registrazione, login, gestione delle sessioni). Keycloak genera i token JWT firmati e il servizio Users espone un endpoint `/public-keys` che restituisce le chiavi pubbliche necessarie per la validazione.

Gli altri microservizi recuperano le chiavi pubbliche da questo endpoint e le salvano in cache localmente. Quando ricevono una richiesta autenticata, estraggono il token dall'header `Authorization`, ne verificano la firma e decodificano i claim.

Oltre alle REST API, l'autenticazione è stata implementata anche per le connessioni WebSocket: il token viene passato durante l'handshake della connessione Socket.IO e validato prima di stabilire il canale. Questo permette di inviare notifiche con dati (oltre che solo segnali) in modo sicuro.

Express Middlewares

Per gestire le richieste HTTP in **Express** sono stati utilizzati i middleware, in particolare è stato definito un middleware per l'autenticazione:

```
export function createAuthMiddleware(options: { optional?: boolean }
↪  = {}) {
  return (req: Request, res: Response, next: NextFunction): void => {
    const token = req.headers.authorization?.replace("Bearer ", "")

    if (!token) {
      if (options.optional) return next()
```

```
      return res.status(401).json({ error: "No token provided" })
    }

    try {
      const payload = await JwtService.verifyToken(token)
      req.userId = payload.user_id
      next()
    } catch (error) {
      res.status(401).json({ error: "Authentication failed" })
    }
  }
}

export const authMiddleware = createAuthMiddleware()
export const optionalAuthMiddleware = createAuthMiddleware({ optional:
↪  true })
```

Le routes utilizzano poi questo middleware per proteggere gli endpoint:

```
export function createNotificationRoutes(controller:
↪  NotificationController): Router {
  const router = Router()
  router.use(authMiddleware)

  router.get("/", (req, res, next) =>
    controller.getNotificationsByUserId(req, res, next))
  router.get("/unread-count", (req, res, next) =>
    controller.getUnreadCount(req, res, next))
    //other routes...

  return router
}
```

A livello applicativo vengono inoltre utilizzati i middleware standard di Express: `cors()` per gestire le richieste cross-origin (necessario per le chiamate dal frontend) e `express.json()` per il parsing automatico del body JSON.

Authenticated WebSocket Gateway con Socket.IO

Il gateway **Socket.IO** gestisce l'autenticazione delle connessioni WebSocket e la distribuzione delle notifiche:

```
export class SocketIOGateway implements NotificationGateway {
  private userSockets: Map<string, Set<string>> = new Map()

  private setupAuthMiddleware(): void {
    this.io.use(async (socket: Socket, next) => {
      const token = socket.handshake.auth.token ||
```

```
↪  socket.handshake.headers.authorization?.replace("Bearer ", "")

      if (!token) {
        return next(new Error("Authentication error: No token
        ↪  provided"))
      }

      try {
        const payload = await JwtService.verifyToken(token)
        socket.data.userId = payload.user_id
        next()
      } catch (err) {
        next(new Error("Authentication error: Invalid token"))
      }
    })
  }

  sendNotificationToUser(userId: string, notification: any):
  ↪  Promise<void> {
    const topic = this.getTopicFromNotificationType(notification.type)
    this.io.to(`user:${userId}`).emit(topic, notification)
    return Promise.resolve()
  }

  broadcastUserOnline(userId: string): void {
    this.io.except(`user:${userId}`).emit("user-online", {
      userId,
      timestamp: new Date()
    })
  }

  broadcastUserOffline(userId: string): void {
    this.io.emit("user-offline", {
      userId,
      timestamp: new Date()
    })
  }
}
```

Endpoint con autenticazione opzionale

Alcuni endpoint utilizzano l'**autenticazione opzionale**: sono accessibili sia da utenti autenticati che non, in questo caso la risposta può variare pur mantenendo il formato consistente. Ad esempio, l'endpoint per ottenere il profilo di un utente restituisce solo i dati pubblici (username e informazioni del profilo) se la richiesta non è autenticata, mentre include anche i dati privati dell'account (e.g. email, interessi) se l'utente autenticato sta richiedendo le proprie informazioni:

```
@cask.get("/:userId")
  def getUser(userId: String, req: Request): Response[String] =
    userService.getUserById(userId) match
      case Left(err) => Response(err, 404)
      case Right(role, user) =>
        val isOwner: Boolean = authenticateAndAuthorize(req,
        ↪  userId).isRight
        val json = if isOwner then
            user.toOwnedUserDTO(userId, role).asJson
        else
          user.toUserDTO(userId, role).asJson
        Response(json.spaces2, 200, Seq("Content-Type" ->
        ↪  "application/json"))
```

Capitolo 6

Test

6.1 6 — Test

Il frontend prodotto è stato testato principalmente su Chrome sfruttando i Chrome DevTools per testare la responsività del layout su diversi dispositivi ma anche direttamente da mobile una volta messo online il sito. Per quanto riguarda il backend sono stati effettuati sia unit test per i singoli componenti sia test e2e principalmente da Swagger e in integrazione con il frontend.

6.1.1 6.1 — Accessibilità (a11y)

Per garantire l'accessibilità dell'interfaccia sono stati adottati due approcci complementari: **analisi statica** durante lo sviluppo e **test automatizzati** sulle pagine web a runtime.

Analisi Statica con ESLint

Il progetto integra **eslint-plugin-vuejs-accessibility** nella configurazione ESLint, identificando potenziali violazioni delle linee guida WCAG direttamente durante lo sviluppo:

```
import vuejsAccessibility from 'eslint-plugin-vuejs-accessibility'

export default defineConfig([
  // ... altre configurazioni
  ...vuejsAccessibility.configs['flat/recommended'],
])
```

Il plugin verifica automaticamente la presenza di attributi `alt` nelle immagini, la corretta associazione di label ai form e l'uso appropriato di ruoli `ARIA` segnalando le violazioni come warning o errori durante il linting.

Test Automatizzati con Puppeteer e Lighthouse

Oltre all'analisi statica, il sistema implementa test di accessibilità dinamici tramite **Puppeteer** e **Lighthouse**. Lo script `a11y-check-multi.js` esegue un audit automatizzato sulle pagine specificate in fase di configurazione:

```
const config = {
  baseUrl: process.env.BASE_URL || 'http://localhost:5173/it',
  pages: [
    { name: 'Home', path: '/' },
    { name: 'Explore', path: '/explore' },
    { name: 'Login', path: '/login' },
    { name: 'Register', path: '/register' },
    { name: 'Create Event', path: '/create-event' },
    { name: 'Event Details', path:
↪   '/events/547a3b27-344a-4318-b17e-edf7cd14aee3' },
    {
      name: 'Organization Profile [Published Events Tab]',
      path:
↪   '/users/7dee946f-3ab9-41b5-92e9-ea6264d9dd35#publishedEvents',
    },
  ],
  minScore: parseInt(process.env.MIN_A11Y_SCORE || '80'),
  themeMode: process.env.THEME_MODE || 'both', // 'light', 'dark',
↪   'both'
}
```

Lo script:

1. **Lancia Chrome in modalità headless** tramite `chrome-launcher`
2. **Connette Puppeteer** per controllare il browser e impostare il tema
3. **Esegue Lighthouse** sulla categoria accessibilità per ogni pagina
4. **Testa entrambi i temi** (light e dark mode) per garantire l'accessibilità in tutte le condizioni
5. **Genera report dettagliati** in formato HTML e un summary testuale

```
async function runLighthouseOnPage(url, port, themeName) {
  const options = {
    logLevel: 'error',
    output: ['json', 'html'],
    onlyCategories: ['accessibility'],
    port: port,
  }
  const runnerResult = await lighthouse(url, options)
  return {
    lhr: runnerResult.lhr,
    reportHtml: runnerResult.report[1],
    theme: themeName,
```

```
  }
}
```

Il report generato include:

- **Score di accessibilità** per ogni pagina (scala 0–100)
- **Violazioni critiche** con conteggio degli elementi coinvolti
- **Top 5 problemi più frequenti** nell'intera applicazione
- **Status pass/fail** basato sulla soglia minima configurata (default 80/100)

Per lanciare i test è stato necessario predisporre una modalità ad hoc, eseguibile con `npm run prod:a11y` che prevede login automatico come organizzazione e disabilita le guardie lato frontend così da permettere la corretta visualizzazione di tutte le pagine.

Lo script può poi essere eseguito tramite `npm run a11y:multi` e restituisce un exit code non-zero in caso di pagine sotto la soglia minima, permettendo l'integrazione in pipeline CI/CD.

Al momento, dato che il processo di seed iniziale genera ogni volta id casuali, è necessario ricontrollare i link nella configuazione ad ogni deploy con seeding.

6.1.2 6.2 — Euristiche di Nielsen

Per offrire una migliore usabilità e user experience, il sistema è stato sottoposto alla valutazione delle euristiche di Nielsen. Di seguito le considerazioni emerse:

1. **Visibilità dello stato del sistema**: per segnalare attività in corso viene mostrata un'icona animata di caricamento. Lo stato online degli utenti è visibile in tempo reale nella chat tramite un apposito label, e le notifiche push informano l'utente di nuovi messaggi, like ricevuti o eventi pubblicati dalle organizzazioni seguite.
2. **Corrispondenza tra sistema e mondo reale**: la terminologia e le icone utilizzate sono in linea con le convenzioni dei social network (cuore per i like, fumetto per la chat, campanella per le notifiche). I concetti di "evento", "biglietto" e "organizzazione" rispecchiano il dominio reale.
3. **Controllo e libertà per l'utente**: l'applicazione non presenta percorsi obbligati. L'utente può navigare liberamente tra le sezioni, tornare indietro in qualsiasi momento e le operazioni critiche (come l'eliminazione di un evento) richiedono conferma esplicita.
4. **Consistenza e standard**: l'applicazione mantiene un linguaggio visivo uniforme grazie al framework Quasar. I pulsanti primari, secondari e di pericolo hanno stili distintivi e coerenti in tutto il sistema. La palette colori è stata definita in fase di design e applicata uniformemente, inclusa la modalità dark.
5. **Prevenzione dall'errore**: i form implementano validazione in tempo reale (formato di email, password e campi obbligatori in generale) con feedback immediato. Le navigation guards impediscono l'accesso a pagine non autorizzate, reindirizzando automaticamente l'utente (es. alla pagina di login se necessaria autenticazione).

6. **Riconoscimento anziché ricordo**: i layout sono consistenti tra le pagine. La barra di navigazione e i tab mantengono la stessa posizione, le card degli eventi hanno struttura uniforme e le icone sono autoesplicative senza necessità di tooltip.
7. **Flessibilità ed efficienza d'uso**: per gli utenti esperti sono disponibili scorciatoie da tastiera globali: `Ctrl/Cmd + D` per il toggle della dark mode, `Ctrl/Cmd + H` per tornare alla home, `Ctrl/Cmd + P` per aprire il profilo e `Ctrl/Cmd + E` per accedere alla creazione eventi.
8. **Design e estetica minimalista**: il design segue il principio KISS con approccio mobile-first. Ogni pagina presenta poche azioni ben distinte: la home mostra gli eventi in evidenza, la pagina explore permette la ricerca, il profilo gestisce le informazioni personali.
9. **Aiuto all'utente**: i messaggi di errore sono contestuali e descrittivi. Le notifiche toast informano l'utente dell'esito delle operazioni e i campi dei form mostrano label di errore specifiche (e.g. "Email non valida", "Password troppo corta").
10. **Documentazione**: vista la semplicità d'uso derivante dalle scelte di design, non è stata necessaria documentazione esterna. Le scorciatoie da tastiera disponibili (`Ctrl/Cmd + D/H/P/E`) sono documentate internamente nel composable `useKeyboardShortcuts`, pronte per essere esposte in una futura sezione "Keyboard Shortcuts" nelle impostazioni.

6.1.3 6.3 — Test di Usabilità

Durante lo sviluppo dell'applicazione, questa è stata fatta provare a diversi utenti, per ricevere di volta in volta dei feedback utili a migliorare lo sviluppo e la resa finale del prodotto.

Ai soggetti del test non è stata fornita nessuna linea guida, ci si è limitati a richiedere una particolare azione da svolgere al fine di osservare come gli utenti avrebbero cercato di eseguire i compiti richiesti vedendo per la prima volta il sistema.

Test per Organizzazioni: fra i compiti assegnati hanno figurato la creazione di eventi mediante l'apposito editor, che si è rivelato semplice ed intuitivo.

Test per Membri: fra i compiti figurava la ricerca degli eventi, l'acquisto dei biglietti, la modifica del profilo e di effettuare una recensione per un evento.

In generale gli utenti hanno trovato la disposizione dei vari elementi sostanzialmente adeguata. Le osservazioni sollevate erano perlopiù su aspetti stilistici e sono state incorporate nelle successive iterazioni di sviluppo.

Capitolo 7

Deployment

7.1 7 — Deployment

Il deploy dell'applicazione è stato automatizzato tramite Docker e Docker Compose: tutti i servizi vengono eseguiti come container indipendenti e lanciati tramite uno script centralizzato.

7.1.1 7.1 — Installazione

1. Clonare il repository

```
git clone https://github.com/EvenToNight/EvenToNight.git
cd EvenToNight
```

2. Configurare le variabili d'ambiente

```
cp .env.template .env
# Modificare .env e compilare tutti i campi richiesti
# Nota: Se si utilizza il flag --no-deps, le chiavi Stripe possono
↪  contenere valori arbitrari. Tutti i valori devono essere compilati.
```

3. Avviare l'applicazione

Opzione A: Utilizzo di immagini pre-compilate da ghcr.io (Consigliato) Download delle immagini:

```
./scripts/composeApplication.sh pull
```

Download delle immagini con seeding del database:

```
./scripts/composeApplication.sh --init-db pull
```

Deploy dell'applicazione:

```
./scripts/composeApplication.sh up -d --wait
```

Deploy con seeding del database:

```
./scripts/composeApplication.sh --init-db up -d --wait
```

Deploy in modalità sviluppo (con porte mappate sull'host e dashboard per database/RabbitMQ/Traefik):

```
./scripts/composeApplication.sh --init-db --dev up -d --wait
```

Opzione B: Build locale Aggiungere il flag `--build` per compilare i servizi localmente invece di utilizzare le immagini pre-compilate:

```
# Build e deploy
./scripts/composeApplication.sh up --build -d --wait

# Build e deploy con seeding
./scripts/composeApplication.sh --init-db up --build -d --wait
```

Flag aggiuntivi `--no-deps`: Esclude le dipendenze esterne (Stripe)

È possibile aggiungere `--no-deps` a qualsiasi comando di deploy per escludere i servizi esterni:

```
# Deploy senza dipendenze esterne
./scripts/composeApplication.sh --no-deps up -d --wait

# Deploy con seeding ma senza dipendenze esterne
./scripts/composeApplication.sh --init-db --no-deps up -d --wait
```

Nota: Quando si utilizza `--no-deps`, le chiavi Stripe **(STRIPE_SECRET_KEY, STRIPE_PUBLISHABLE_KEY, STRIPE_WEBHOOK_SECRET)** in `.env` possono contenere valori arbitrari.

Configurazione Stripe **Per i pagamenti Stripe in ambiente locale** (richiesto solo se NON si utilizza `--no-deps`):

```
./services/payments/scripts/local-webhooks.sh
```

Questo script deve essere eseguito per inoltrare i webhook di Stripe all'ambiente locale.

Per maggiori informazioni sull'utilizzo della modalità sandbox, consultare la documentazione Stripe.

Setup alternativo

Utilizzare Gradle per configurare l'intero ambiente con seeding e listener Stripe:

```
./gradlew setupApplicationEnvironment
```

Teardown

Arresto dell'applicazione:

```
./scripts/composeApplication.sh down
```

Arresto e rimozione dei volumi:

```
./scripts/composeApplication.sh down -v
```

Come ulteriore alternativa, è possibile visualizzare il sito già in produzione al link https://eventonight.site/it

Capitolo 8

Conclusioni

8.1 8 — Conclusioni

Siamo soddisfatti della piattaforma realizzata, nata da una nostra idea di business. La presentazione della piattaforma a potenziali utenti ha dimostrato che l'interfaccia è intuitiva e di facile utilizzo, confermando l'efficacia dell'approccio user-centered e le best practice di HCI utilizzate.

Lo sviluppo del progetto ha richiesto l'impiego di diverse tecnologie orientate alla realizzazione di applicazioni e servizi web moderni e distribuiti. In particolare, l'utilizzo dello stack MEVN ma soprattutto il framework NestJS, sperimentato in alternativa a express, hanno permesso di sviluppare in modo efficace API modulari e manutenibli. L'architettura modulare ha favorito il lavoro in parallelo su diverse funzionalità, mentre la specifica Swagger/OpenAPI usata per documentare le API ha favorito il testing e l'integrazione con il frontend.

Il sistema implementa le principali funzionalità previste e risulta facilmente estendibile. Tra le possibili evoluzioni future si evidenziano: la possibilità per le organizzazioni di rispondere alle recensioni, l'introduzione di notifiche aggiuntive (biglietti in esaurimento, suggerimenti di amicizia, aggiornamenti sugli eventi seguiti), insight e statistiche sul profilo, gestione dei rimborsi e supporto di più valute per l'internazionalizzazione.

Complessivamente, il progetto ha permesso di realizzare una piattaforma web moderna, modulare, scalabile e accessibile, con un'interfaccia intuitiva e pronta per l'uso da parte di utenti reali. Ha rappresentato al contempo un'importante occasione di apprendimento, consolidando competenze nello sviluppo full-stack, dalla progettazione e realizzazione di un'interfaccia utente efficace alla progettazione e deploy di un sistema backend a microservizi.