

EvenTour

18 gennaio 2022

Corso di Informatica IIIB AA. 2019/2020

Progetto Progettazione e Algoritmi

Colombi Simone

Scarpellini Stefano

Contents

Manuale Utente.....	7
1. Avvio del server API.....	7
2. Avvio del client web.....	7
Iterazione 0.....	9
1. Requisiti utente	9
2. Tool Chain	11
3. Use Cases	12
Descrizione testuale.....	12
Use Case diagram.....	21
4. Architettura	22
Architecture model.....	22
Deployment diagram	23
Database schema.....	24
Component diagram	28
Pattern architetturali	31
Iterazione 1 (implementazione Iterazione 0).....	35
1. Class diagram dell'API.....	35
2. Definizione dei dati di test e di utilizzo.....	40
3. Algoritmo evenTour.....	42
4. Testing interfaccia grafica.....	44
Per l'interfaccia grafica sono state fatte due tipi di analisi.	44
Analisi statica: Eslint.....	44
Analisi dinamica: NightWatch	46
5. Testing API	48
Analisi statica	48
Analisi dinamica	49
Iterazione 2.....	51
1. Use Cases	51
Descrizione testuale.....	51
Use Case diagram.....	62
2. Architettura	63

Deployment diagram	63
Component diagram	64
3. Class diagram.....	65
4. Novità implementative introdotte	70
5. Algoritmo evenTour.....	70
6. Testing interfaccia grafica.....	73
Analisi statica	73
Analisi dinamica	74
7. Testing API.....	78
Analisi statica	78
Analisi dinamica	79
Iterazione 3	81
1. Use Cases	81
Descrizione testuale.....	81
Use Case diagram.....	91
2. Architettura	92
Deployment diagram	92
Component diagram	92
3. Class diagram.....	94
4. Algoritmo evenTour.....	100
5. Variazioni al database nella iterazione corrente	107
6. Testing interfaccia grafica.....	108
Analisi statica	108
Analisi dinamica	108
7. Testing API.....	114
Analisi statica	114
Analisi dinamica	115
Sviluppi futuri	117

Manuale Utente

Per vedere in opera il progetto EvenTour devono essere scaricati tutti i file contenuti nel repository di GitHub al seguente indirizzo:

<https://github.com/ScarpelliniStefano/EvenTOUR>

Una volta scaricato in locale, per ogni iterazione, saranno disponibili una cartella contenente il progetto Eclipse del server che fornisce le API e una cartella zippata (consigliamo 7Zip O winRAR come programma di decompressione della cartella compressa^[1]) contenente il progetto Vue.

1. Avvio del server API

- a. Aprire Eclipse (la versione consigliata è *Eclipse 2021.06 IDE for Enterprise Java and Web Developers*^[2], che è portable e, dopo estrazione con winRAR o 7zip, può essere aperto)
- b. Posizionarsi come workspace la cartella <cartella di download del progetto>/iterazioneN/SERVER_API/.
- c. Il progetto può essere importato in questo modo: nella sezione Package Explorer, effettuare *Import* o *Import Project>Maven>Existing Maven Projects*. Cliccando su Next, inserire poi il path sopracitato. In questo modo si potrà selezionare il file POM e verranno importate tutte le dipendenze necessarie.
- d. Posizionandosi sul progetto “eventour”, espandendolo, posizionandosi sulla source folder *src/main/java* e cliccando con il tasto destro, si potrà avviare l'esecuzione del server API tramite *Run As>Java application*.

2. Avvio del client web

- a. Installare Node.js seguendo la guida che fa riferimento al proprio sistema operativo:
 - Windows: <https://www.nodeacademy.it/installare-node-js-windows/>
 - Linux: <https://www.nodeacademy.it/installare-node-js-ubuntu-linux/>
 - Mac Os: <https://www.nodeacademy.it/installare-node-js-mac-os-x/>
- b. Dopo aver installato Node.js, è necessario recarsi tramite console dei comandi all'interno del progetto Vue.
- c. Eseguire il comando **npm start**
- d. Per fermare l'esecuzione del server web, premere CTRL+C, digitare s e premere invio.

Si trovano anche online ai seguenti indirizzi:

Iterazione 1: <https://eventouriterazione1.netlify.app/>

¹ <https://www.7-zip.org/download.html>

² <https://www.eclipse.org/downloads/packages/release/2021-06/r>

Iterazione 2: <https://eventouriterazione2.netlify.app/>

Iterazione 3: <https://eventouriterazione3.netlify.app/>

IMPORTANTE!

È importante che per l'uso della iterazione 1 venga usato il server api presente in esso, e così via per la 2 e la 3.

Nel documento “Credenziali di accesso” sono esplicitate le credenziali di accesso per la prova delle funzionalità.

Iterazione 0

1. Requisiti utente

L'azienda milanese evenTOUR Italia s.p.a. ha deciso di collaborare con l'Università degli Studi di Bergamo per fornire a tutti gli utenti sul territorio italiano una piattaforma per la prenotazione di eventi sparsi lungo tutto il territorio resi disponibili da associazioni ed enti/organizzazioni.

Gli eventi disponibili sulla piattaforma possono essere sia gratuiti che a pagamento, a discrezione dell'ente che lo ha inserito, con, in quest'ultimo caso, modalità di pagamento online che si basano su circuiti nazionali come Bancomat o carte di credito dei maggiori circuiti internazionali quali Visa, Mastercard, American Express.

All'interno della piattaforma esistono differenti ruoli che interagiscono con l'evento e con altri ruoli, quali utenti, manager (enti organizzatori), ticket inspector e administrator.

Gli utenti, per poter prenotare qualsiasi tipo di evento, valutare gli eventi a cui hanno partecipato, visualizzare gli eventi sulla base di differenti filtri o sfruttare la potenza dell'algoritmo evenTour, su cui si basa la collaborazione e l'intero progetto, devono essere registrati alla piattaforma fornendo i propri dati quali:

- Nome e cognome
- Sesso
- Data di nascita (l'utente deve essere maggiorenne)
- Luogo di residenza
- Tipi di evento preferiti
- Username, mail e password di accesso

Per quanto riguarda i manager, essi hanno la possibilità di gestire diversi eventi illimitatamente tramite il pagamento di una quota d'iscrizione per il mantenimento del proprio status e tramite la registrazione sulla piattaforma, fornendo i propri dati quali:

- Nome e cognome
- Data di nascita (il manager deve essere maggiorenne)
- Luogo di residenza
- Partita IVA

- Ragione sociale
- Mail e password

Oltre alle funzionalità precedentemente descritte, il manager ha anche la possibilità di visualizzare un report relativo a tutti gli eventi passati con dati relativi alla valutazione media del singolo evento e a tutto ciò che è relativo alla affluenza e alla parte economica legata alla manifestazione.

I ticket inspectors sono dei dipendenti, associati a vari eventi, che permettono il controllo delle prenotazioni all'ingresso dell'evento. Anche loro possono accedere alla piattaforma digitando un codice e una password che vengono inviati a loro tramite comunicazione da parte del manager e della piattaforma. Il controllo delle prenotazioni avviene tramite uno strumento smart di lettura di codici QR. L'utilizzo della pagina associata al controllo dei biglietti deve essere molto semplice e intuitiva anche per l'utilizzo su dispositivi mobili.

Gli administrator sono i veri e propri amministratori della piattaforma: essi gestiscono le richieste di iscrizione da parte dei manager e verificano che i dati inseriti al momento dell'iscrizione siano relativi a enti reali esistenti sul territorio, andando ad escludere le iscrizioni potenzialmente dannose o false, che non permettono alla piattaforma di proseguire nel suo compito nel migliore dei modi. Controllano altresì il pagamento delle quote di iscrizione o di rinnovo. Possono, a loro discrezione, attribuire dei malus ai manager che non rispettano i termini di servizio imposti dall'azienda.

Gli utenti, dopo una fase di registrazione e un successivo login, possono abilitare la funzionalità legata alla newsletter grazie alla quale sono sempre a conoscenza delle migliori proposte e delle novità riguardo a eventi già osservati in precedenza, nonché alle questioni organizzative di eventi già prenotati.

2. Tool Chain

Per la realizzazione del software presentato in questo report sono stati utilizzati i seguenti tool:

- **Modellazione**
 - Draw.io^[3]: utilizzato per la modellizzazione di Use Case Diagram, Architecture Diagram, Deployment Diagram e Component Diagram.
 - Enterprise Architect (trial edition)^[4]: test di modellizzazione Class Diagram
 - UML Lab (trial edition)^[5] e CDA^[6]: Class Diagram
- **Implementazione software**
 - Eclipse: IDE per l'implementazione del codice Java lato server (realizzazione dell'API) tramite SpringBoot
 - Visual Studio Code: Editor per lo sviluppo dell'interfaccia web tramite il framework Vue.js
 - MongoDB Compass e Studio3T (academic license): Strumenti per la gestione del database MongoDB presente su un cluster Atlas.
- **Analisi del software**
 - NightWatch.js: per l'analisi dinamica del codice javascript
 - Eslint: per l'analisi statica delle componenti Vue
 - JUnit: per l'analisi dinamica del codice Java
 - Spotbugs e PMD: per l'analisi statica del codice Java
 - JArchitect: report per la qualità del codice
- **Altri tools**
 - Discord: canale di discussione del progetto
 - GitHub: per il versioning con interfaccia grafica
 - Microsoft Word: per la scrittura della documentazione
 - JSON-server e faker.js: per la generazione randomica di dati di test
 - Postman: piattaforma API per la gestione delle API

³ <https://drawio-app.com/>

⁴ <https://sparxsystems.com/>

⁵ <https://www.uml-lab.com/en/uml-lab/>

⁶ <http://www.dependency-analyzer.org/>

3. Use Cases

Descrizione testuale

Nome	UC1: Visualizzazione Eventi
Descrizione	Mostra tutti gli eventi disponibili
Attori	/
Obiettivi	Mostrare gli eventi disponibili
Precondizioni	/
Passi	<ol style="list-style-type: none"> 1. L'utente entra nella pagina di visualizzazione degli eventi 2. Il sistema stampa a video tutti gli eventi disponibili 3. L'utente può selezionare un evento di interesse 4. Il sistema mostra altri dettagli relativi a esso
Casi d'uso associati / flussi alternativi	Generalizzazione: UC1.1: Visualizzazione per data UC1.2: Visualizzazione per luogo
Postcondizioni	/

Nome	UC1.1: Visualizzazione per data
Descrizione	Mostra tutti gli eventi disponibili ordinati per data ascendente o discendente
Attori	User
Obiettivi	Mostrare gli eventi disponibili in un certo ordine
Precondizioni	/
Passi	<ol style="list-style-type: none"> 1. L'utente entra nella pagina di visualizzazione degli eventi 2. Il sistema stampa a video tutti gli eventi disponibili in ordine crescente di data (dal più vicino al più lontano)

	<ul style="list-style-type: none"> 3. L'utente sceglie l'ordine con il quale mostrare gli eventi (dal più vicino al più lontano come date o viceversa) 4. Il sistema mostra a video gli eventi ordinati secondo l'ordine richiesto 5. L'utente può selezionare un evento di interesse 6. Il sistema mostra altri dettagli relativi a esso
Casi d'uso associati / flussi alternativi	Specializzazione UC1: Visualizzazione Eventi
Postcondizioni	/

Nome	UC1.2: Visualizzazione per luogo
Descrizione	Mostra tutti gli eventi disponibili ordinati per luogo in ordine alfabetico o inverso
Attori	User
Obiettivi	Mostrare gli eventi disponibili svolti in luoghi ordinati alfabeticamente
Precondizioni	/
Passi	<ul style="list-style-type: none"> 1. L'utente entra nella pagina di visualizzazione degli eventi 2. Il sistema stampa a video tutti gli eventi disponibili in ordine crescente di data (dal più vicino al più lontano) 3. L'utente richiede l'ordinamento crescente (alfabetico) o decrescente (alfabetico inverso) per regione, provincia, comune 4. Il sistema mostra a video gli eventi relativi all'ordine richiesto 5. L'utente può selezionare un evento di interesse 6. Il sistema mostra altri dettagli relativi a esso
Casi d'uso associati / flussi alternativi	Specializzazione UC1: Visualizzazione Eventi
Postcondizioni	/

Nome	UC2: Login
Descrizione	L'utente può accedere al suo account per effettuare prenotazioni, se utente, o altre funzioni, se manager o ticket inspector
Attori	User, Manager, Ticket Inspector
Obiettivi	Accedere alla propria area privata nel sito per svolgere funzioni
Precondizioni	Essere registrato
Passi	<ol style="list-style-type: none"> 1. L'attore entra nella pagina di login 2. L'attore compila il campo "username" 3. L'attore compila il campo "password" 4. L'attore richiede di essere autenticato con quelle credenziali 5. Il sistema autorizza l'accesso all'utente e lo porta nella sua area riservata <ol style="list-style-type: none"> 5.a. Nel caso di autenticazione di un User, il sistema riporta alla pagina di visualizzazione evento 5.b. Nel caso di autenticazione di un Manager, il sistema riporta alla pagina di visualizzazione eventi del manager 5.c. Nel caso di autenticazione di un Ticket Inspector, il sistema riporta alla pagina di scansione del QR
Casi d'uso associati / flussi alternativi	/
Postcondizioni	L'utente è autenticato

Nome	UC2.2: Login Errato
Descrizione	L'utente, nell'accesso alla piattaforma, non viene autenticato
Attori	User, Manager, Ticket Inspector
Obiettivi	Non permettere l'accesso alla propria area privata nel sito a causa di errori di autenticazione
Precondizioni	Essere registrato

Passi	<ol style="list-style-type: none"> 1. L'attore entra nella pagina di login 2. L'attore compila il campo "username" 3. L'attore compila il campo "password" 4. L'attore richiede di essere autenticato con quelle credenziali 5. Il sistema non autorizza l'accesso all'utente e segnala l'errore <ol style="list-style-type: none"> 5.a. Nel caso di autenticazione di un User, il sistema riporta alla pagina di login segnalando l'errore e l'utente ha la possibilità di ritentare l'accesso. 5.b. Nel caso di autenticazione di un Manager, il sistema riporta alla pagina di login segnalando le credenziali come errate e il manager ha la possibilità di ritentare l'accesso. 5.c. Nel caso di autenticazione di un Ticket Inspector, il sistema riporta alla pagina di accesso.
Casi d'uso associati / flussi alternativi	<>extend>> UC2: Login
Postcondizioni	Non essere autenticato

Nome	UC3: Sign Up
Descrizione	Permette la registrazione di un visitatore alla piattaforma
Attori	User
Obiettivi	Permettere di registrarsi alla piattaforma
Precondizioni	L'utente non deve essere già registrato
Passi	<ol style="list-style-type: none"> 1. L'utente entra nella pagina di registrazione alla piattaforma 2. L'utente compila i dati anagrafici e di autenticazione (mail e password) 3. Il sistema verifica che non esistano account associati alla stessa mail 4. Il sistema salva i dati dell'utente nel database

	5. Il sistema segnala all'utente l'avvenuta registrazione alla piattaforma e riporta alla pagina di visualizzazione eventi
Casi d'uso associati / flussi alternativi	/
Postcondizioni	L'utente è registrato, se la registrazione rispetta i controlli fatti

Nome	UC4: Prenotazione Evento
Descrizione	L'utente può prenotare il posto per un evento o rimuovere una prenotazione da un evento già prenotato.
Attori	User
Obiettivi	Prenotare un posto per un evento da parte di un utente registrato
Precondizioni	<ul style="list-style-type: none"> • L'utente è loggato al sito • L'evento ha posti disponibili • L'evento avviene in una data futura • L'utente non è già prenotato all'evento in caso di prenotazione. • L'utente è già prenotato all'evento in caso di disdetta della prenotazione. • L'utente si trova nella pagina di visualizzazione degli eventi
Passi	<p>1.a. Prenotazione a un evento:</p> <p> 1.a.1. L'utente richiede la prenotazione all'evento nella pagina di visualizzazione dell'evento</p> <p> 1.a.2. Il sistema controlla le condizioni richieste</p> <p> 1.a.3. Il sistema, dopo aver verificato la disponibilità del posto richiesto, prenota il posto per l'evento e aggiorna il database</p> <p> 1.a.4. Il sistema conferma la prenotazione del posto richiesto</p> <p>1.b. Disdetta di una prenotazione</p> <p> 1.b.1. L'utente richiede la disdetta della prenotazione all'evento nella pagina di visualizzazione dell'evento</p>

	<p>1.b.2. Il sistema disdice la prenotazione il posto per l'evento e aggiorna il database</p> <p>1.b.3. Il sistema conferma la disdetta della prenotazione del posto richiesto</p>
Casi d'uso associati / flussi alternativi	<>Include>> UC2: Login
Postcondizioni	L'utente è prenotato all'evento e ha un posto riservato nel caso di prenotazione

Nome	UC5: Scan QR Code
Descrizione	Il sistema permette al Ticket Inspector di scansionare il QR presente sulla prenotazione di un utente e autorizzarne l'accesso all'evento
Attori	Ticket Inspector
Obiettivi	Validare una prenotazione di un utente a un evento
Precondizioni	Il Ticket Inspector deve essere autenticato
Passi	<ol style="list-style-type: none"> 1. Il Ticket Inspector scansiona con lo scanner il QR presente sulla prenotazione 2. Il sistema verifica che la prenotazione sia valida <ol style="list-style-type: none"> 3.a. Il sistema segnala, in caso di esito positivo, l'accesso consentito all'utente che ha prenotato 3.b. Il sistema segnala, in caso di esito negativo, l'accesso rifiutato all'utente per prenotazione non valida
Casi d'uso associati / flussi alternativi	<>Include>> UC2: Login
Postcondizioni	/

Nome	UC6: Inserimento Evento
Descrizione	Il sistema permette a un manager la creazione di un evento per l'inserimento di esso sulla piattaforma

Attori	Manager
Obiettivi	Creazione di un evento da parte del manager
Precondizioni	Il Manager deve essere autenticato
Passi	<ol style="list-style-type: none"> 1. Il Manager richiede l'inserimento di un evento al sistema 2. Il sistema mostra una pagina da compilare con i dati relativi all'evento da inserire 3. Il Manager compila i dati relativi all'evento 4. Il sistema verifica la correttezza dei dati inseriti (data futura, ad esempio) <ul style="list-style-type: none"> 5.a. Il sistema, in caso di esito positivo, segnala l'avvenuto inserimento dell'evento sulla piattaforma 5.b. Il sistema, in caso di esito negativo, segnala gli errori commessi nell'inserimento dell'evento
Casi d'uso associati / flussi alternativi	<<Include>> UC2: Login
Postcondizioni	L'evento è presente sulla piattaforma

Nome	UC7: Visualizzazione Eventi Manager
Descrizione	Il sistema permette la visualizzazione degli eventi relativi al manager
Attori	Manager
Obiettivi	Mostrare gli eventi inseriti da un manager
Precondizioni	Il Manager deve essere autenticato
Passi	<ol style="list-style-type: none"> 1. Il sistema mostra i dati di base degli eventi relativi al manager <ul style="list-style-type: none"> 2.1. Il manager può richiedere la rappresentazione dettagliata di un evento 2.2. Il sistema mostra i dettagli dell'evento richiesto
Casi d'uso associati / flussi alternativi	<<Include>> UC2: Login

Postcondizioni	/
-----------------------	---

Nome	UC8: Gestione Ticket Inspector
Descrizione	Il sistema permette al Manager l'organizzazione dei propri Ticket Inspector.
Attori	Manager
Obiettivi	Organizzare i Ticket Inspector di un Manager
Precondizioni	Il Manager deve essere autenticato
Passi	<ol style="list-style-type: none"> 1. Il Manager richiede l'accesso alla pagina di visualizzazione di un evento dettagliato 2. Il sistema mostra la pagina dove poter visualizzare i Ticket Inspector di un evento.
Casi d'uso associati / flussi alternativi	<<Include>> UC2: Login
Postcondizioni	/

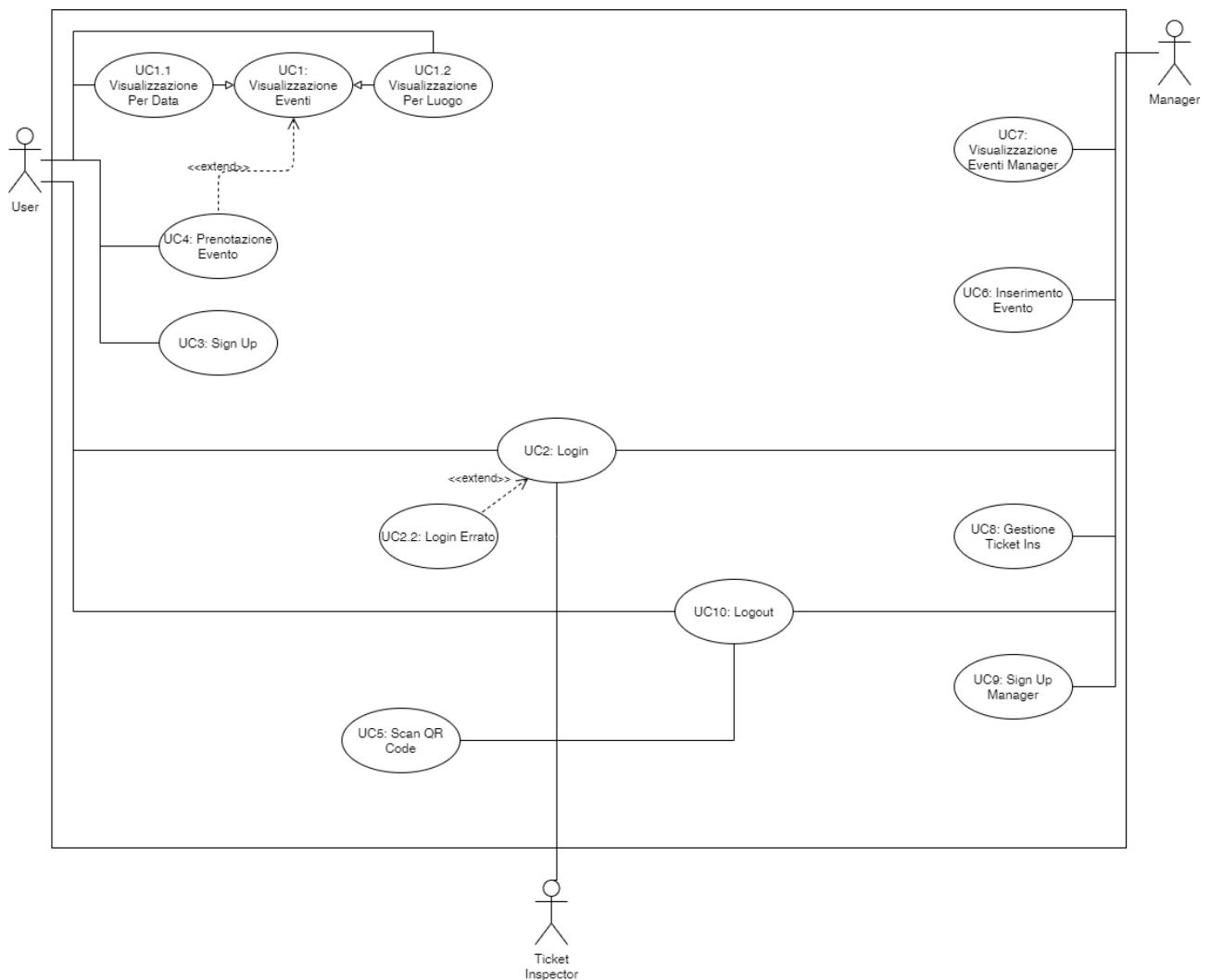
Nome	UC9: Sign Up Manager
Descrizione	Il sistema permette all'utente di registrarsi come Manager (organizzatore di eventi)
Attori	Manager
Obiettivi	Registrare un Manager come tale
Precondizioni	Il Manager non è ancora registrato
Passi	<ol style="list-style-type: none"> 1. L'utente richiede la registrazione alla piattaforma come Manager 2. Il sistema mostra la pagina di registrazione alla piattaforma come Manager 3. L'utente compila tutti i dati anagrafici e dell'organizzazione 4. L'utente richiede la registrazione con i dati inseriti 5. Il sistema controlla i dati inseriti

	<p>6.a. In caso di validazione corretta, il sistema conferma la richiesta di registrazione via mail al manager</p> <p>6.b. In caso di validazione errata, il sistema richiede la modifica dei dati errati.</p>
Casi d'uso associati / flussi alternativi	<>include>> UC2: Login
Postcondizioni	Il Manager è registrato se la registrazione è andata a buon fine

Nome	UC10: Logout
Descrizione	L'utente può disconnettersi dal suo account
Attori	User, Manager, Ticket Inspector
Obiettivi	Uscire dalla propria area privata nel sito per svolgere funzioni
Precondizioni	Essere autenticato
Passi	<ol style="list-style-type: none"> 1. L'attore preme il pulsante di logout 2. Il sistema autorizza la disconnessione all'utente e lo porta nella pagina iniziale di visualizzazione eventi
Casi d'uso associati / flussi alternativi	/
Postcondizioni	L'utente non è più autenticato

Use Case diagram

Il diagramma seguente mostra i casi d'uso ad alta priorità, le fondamenta su cui basare l'intero progetto. Le funzionalità principali sono l'autenticazione di un utente, di un organizzatore e di un ticket inspector, una prenotazione molto basilare di un evento, la visualizzazione degli eventi per data o per luogo (ordine alfabetico della regione, a seguire ordine alfabetico della provincia e infine ordine alfabetico del comune), la registrazione alla piattaforma da parte di utente o manager, l'inserimento da parte del manager di un evento con la possibile gestione dei propri ticket inspector associati all'evento e, per il ticket inspector, la possibilità di scansionare un codice QR della prenotazione e quindi permettere o negare l'accesso all'evento.



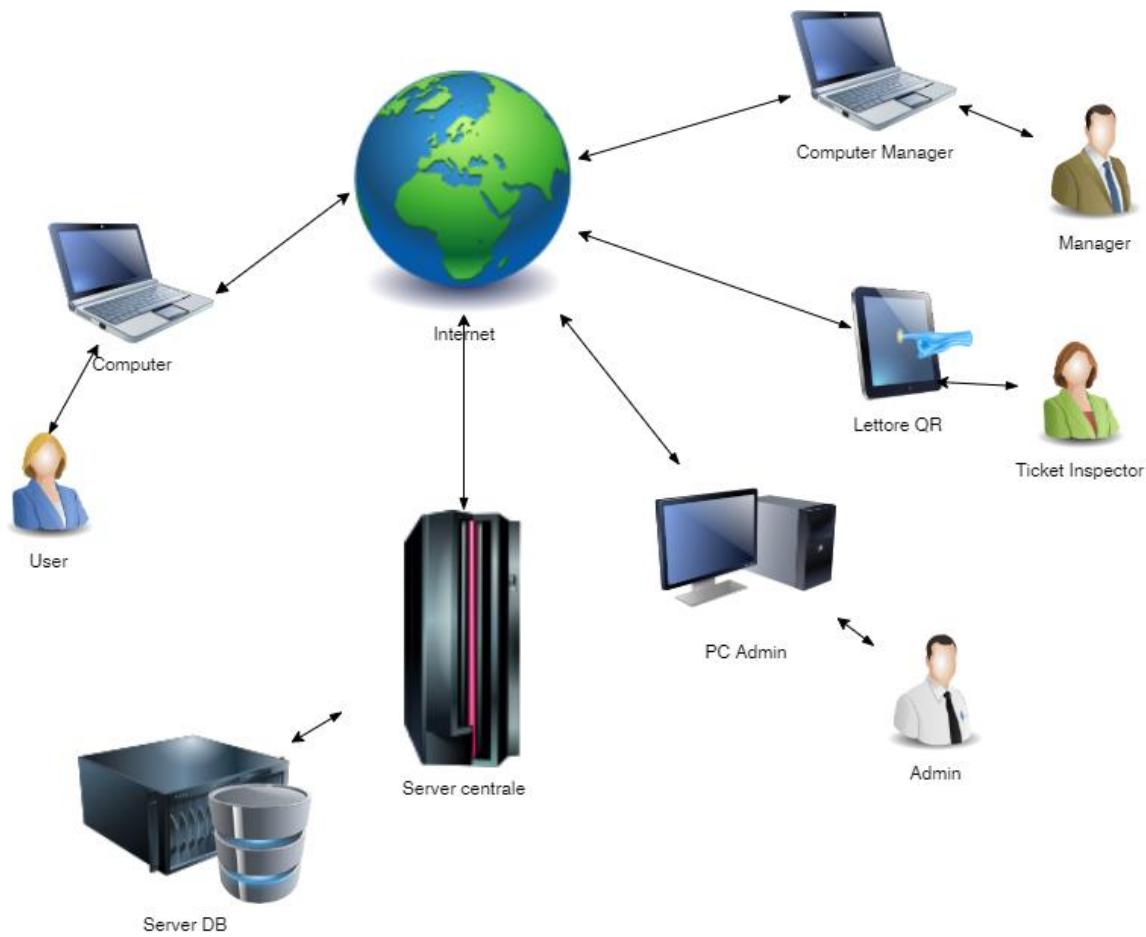
Modello 2.1. Use Case Diagram

4. Architettura

Data la natura del progetto, si decide di realizzare una piattaforma che sfrutti tecnologie web che comunichino attraverso internet.

Architecture model

Il modello architettonico di massima mostra la possibilità per un utente o per un manager di accedere attraverso un computer a internet, tipicamente tramite l'uso di un browser a suo piacimento. Ciò non vieta tuttavia di usare un dispositivo mobile per la navigazione nel portale. Si è pensato di collocare su un server centrale il server web che fornisce l'interfaccia utente e il server che fornisce le API necessarie al corretto funzionamento della piattaforma. Il ticket inspector, essendo in un luogo fisico in mobilità e avendo necessità di scansionare dei codici QR contenuti sulle prenotazioni degli utenti, potrà usare tablet o smartphone connessi a Internet per l'espletazione della sua funzione. Il server API comunicherà poi con un DB server che fornirà e permetterà una gestione separata della banca dati su cui fa da fondamenta alla piattaforma.



Modello 2.2. Hardware Architecture diagram

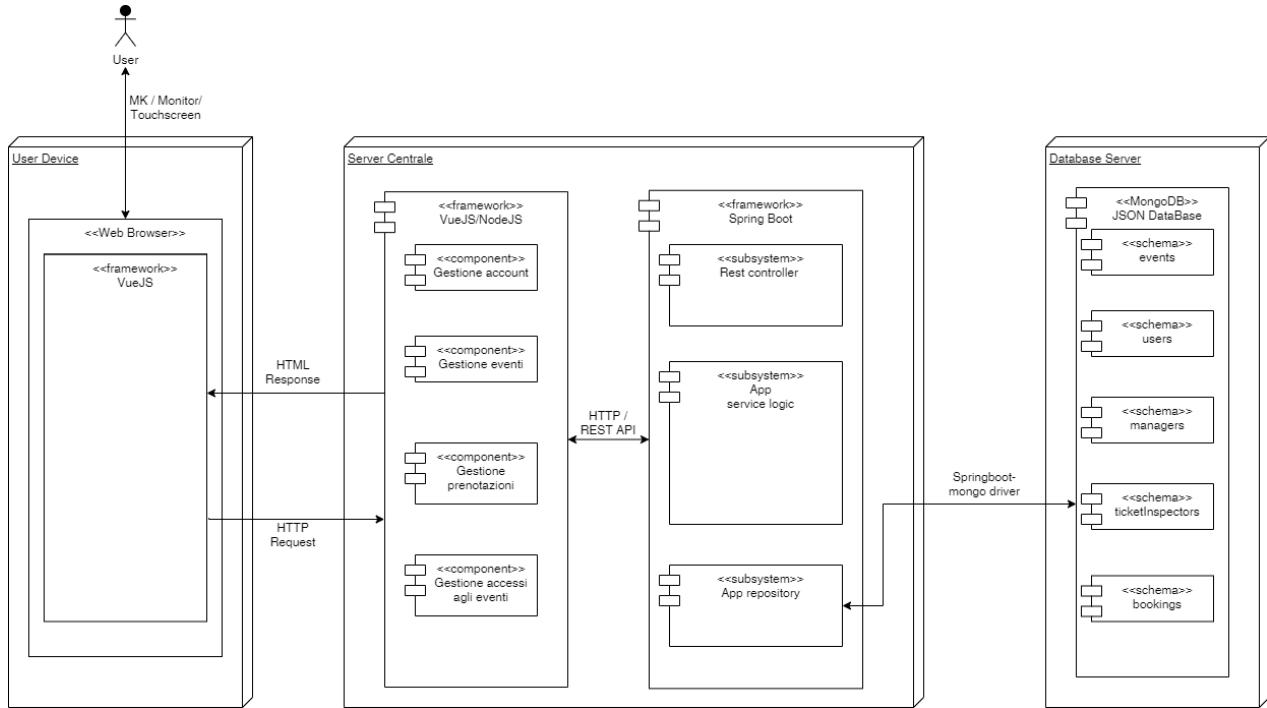
Deployment diagram

Nel modello 2.2 e 2.3 sono mostrati l'Hardware Architecture Diagram e il Deployment Diagram del sistema progettato per lo sviluppo della piattaforma EvenTour.

Si può osservare che si tratta di un'architettura *Three Tiers*:

1. A sinistra si individua lo *User Device*, ovvero il dispositivo su cui è presente un browser web che permetta la fruizione dei contenuti della piattaforma. Si nota che esso sfrutterà il framework di Vue.js per la comunicazione con il server web messo a disposizione sempre dal framework e gestito da esso, basato su Node.js;
2. Nella parte centrale individuiamo il secondo layer, che contiene sia il server web sopraccitato sia il server delle API messe a disposizione per l'uso della piattaforma. Questa scelta è stata fatta per mantenere una comunicazione sicura tra web server e api di accesso ai dati, avere sulla stessa macchina il controllo di tutte le operazioni e immaginando che l'impresa EvenTOUR abbia a disposizione un solo server.
3. Nella parte destra del modello 2.3 si individua il DB server (MongoDB Atlas) che contiene il cluster con presenti tutti i dati di eventi, utenti, manager, prenotazioni e risorse ausiliarie.

Per la comunicazione tra il dispositivo da cui è connesso l'utente e il server centrale si utilizzano meccanismi di richieste HTTP e risposte HTML, le cui pagine vengono generate dinamicamente dal server web e restituite al richiedente. La comunicazione tra web server e server API viene effettuata attraverso HTTP e REST API. La comunicazione con il database server sfrutta i driver disponibili per Spring Boot per la comunicazione con database non relazionali come mongoDB.



Modello 2.3. Deployment Diagram

Database schema

Si è scelto di utilizzare un database non relazionale perché più indicato per applicazioni che fanno un uso massiccio di moli di dati grandi, per una latenza d'utilizzo bassa e per avere modelli di dati flessibili. Nella iterazione iniziale si è scelto di inserire le collezioni di dati aventi struttura specificata dalla Tabella 2.1. Per quanto concerne le località di svolgimento degli eventi o di residenza dei manager o degli utenti, saranno approfonditi i campi nella Tabella 2.2.

Collezione	Campo	Tipo di dato	Indice	Descrizione
events	_id	ObjectId	✓	ID univoco generato all'inserimento nel database
	title	String		Titolo dell'evento
	description	String		Descrizione dell'evento
	dataOra	Date		Data e ora dell'evento
	location	Object		Struttura spiegata successivamente, contiene il luogo nel quale viene svolto l'evento

	types	Array		Vettore di stringhe con i codici dei tipi dell'evento
	managerId	ObjectId		ID univoco del manager che ha creato l'evento
	urlImage	String		URL dell'immagine di "copertina" dell'evento
	totSeat	Int32		Posti totali disponibili per l'evento
	freeSeat	Int32		Posti ancora liberi per l'evento
	price	Double		Prezzo dell'evento (0 se gratuito)

Collezione	Campo	Tipo di dato	Indice	Descrizione
managers	_id	ObjectId	✓	ID univoco generato all'inserimento nel database
	mail	String	✓	Mail (univoca) di registrazione del manager
	password	String		Password di accesso
	name	String		Nome del manager
	surname	String		Cognome del manager
	dateOfBirth	Date		Data di nascita del manager
	residence	Object		Struttura spiegata successivamente, contiene il luogo nel quale risiede il manager
	codicePIVA	String		Partita IVA del manager
	ragioneSociale	String		Ragione sociale

Collezione	Campo	Tipo di dato	Indice	Descrizione
users	_id	ObjectId	✓	ID univoco generato all'inserimento nel database
	username	String		Username per l'accesso alla piattaforma previa registrazione
	mail	String	✓	Mail (univoca) di registrazione dell'utente
	password	String		Password di accesso
	name	String		Nome dell'utente
	surname	String		Cognome dell'utente
	dateOfBirth	Date		Data di nascita dell'utente
	sex	String		Sesso dell'utente
	residence	Object		Struttura spiegata successivamente, contiene il luogo nel quale risiede l'utente
	types	Array		Vettore di stringhe con i codici dei tipi di evento preferiti

Collezione	Campo	Tipo di dato	Indice	Descrizione
bookings	_id	ObjectId	✓	ID univoco generato all'inserimento nel database
	userID	ObjectId		Username per l'accesso alla piattaforma previa registrazione
	eventID	ObjectId		Mail (univoca) di registrazione dell'utente
	prenotedSeat	Int32		Password di accesso
	come	Boolean		Nome dell'utente

Collezione	Campo	Tipo di dato	Indice	Descrizione
ticketInspectors	_id	ObjectId	✓	ID univoco generato all'inserimento nel database
	code	String	✓	Codice generato alla creazione per identificare il ticket inspector
	password			Password di accesso
	fullName	String		Nome e cognome del ticket inspector
	eventId	ObjectId	✓	ID univoco dell'evento a cui è associato il ticket inspector

Tabella 2.1 Collezioni presenti su mongodb con descrizione dei singoli campi

Per quanto concerne gli indici, le spunte nere sono indici, con ordine ascendente, relativi all'identificativo univoco di ogni oggetto. Le spunte colorate di blu, verde e rosso identificano invece indici ausiliari per verificare che non esistano più manager con la stessa mail di iscrizione, più utenti con la stessa mail di iscrizione o più ticket inspector con coppia <code, eventId> uguale.

Oggetto	Campo	Tipo di dato	Descrizione
location/residence	locality	String	Località, via e civico
	city	String	Città o paese
	cap	String	Codice di avviamento postale del comune
	provincia	String	Provincia del comune specificato
	sigla	String	Sigla della provincia
	regione	String	Regione di appartenenza
	lat	Double	Latitudine in gradi
	lng	Double	Longitudine in gradi

Tabella 2.2 Descrizione dei campi presenti negli oggetti location e residence

Si riportano inoltre di seguito i codici e i relativi nomi associati ai tipi di evento. Essi saranno utilizzati sia nella API sia nell'applicativo web (un evento identificabile come un concerto punk rock sarà identificato dal tipo 1.2.3):

1. concerto
 1. musica commerciale

1. pop
 2. disco/dance
 3. altro
 2. rock
 1. hard rock
 2. metal
 3. punk
 4. altro
 3. rap
 1. trap
 2. freestyle
 3. battle
 4. classica
 1. lirica
 2. orchestrale
 3. strumentale
 4. gospel
2. teatro
 1. musical
 2. commedia
 3. opera lirica
 4. prosa
 5. magia/cabaret
 6. tragedia
 7. spettacolo di danza
 3. altri eventi
 1. sfilata
 2. proiezione speciale (non solo al cinema)
 3. evento sportivo

4. firmacopie/presentazione

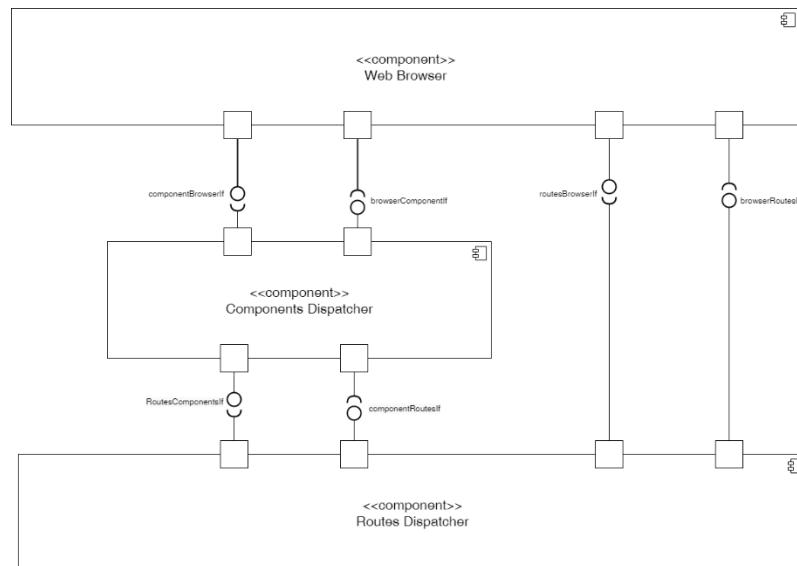
5. circo

Component diagram

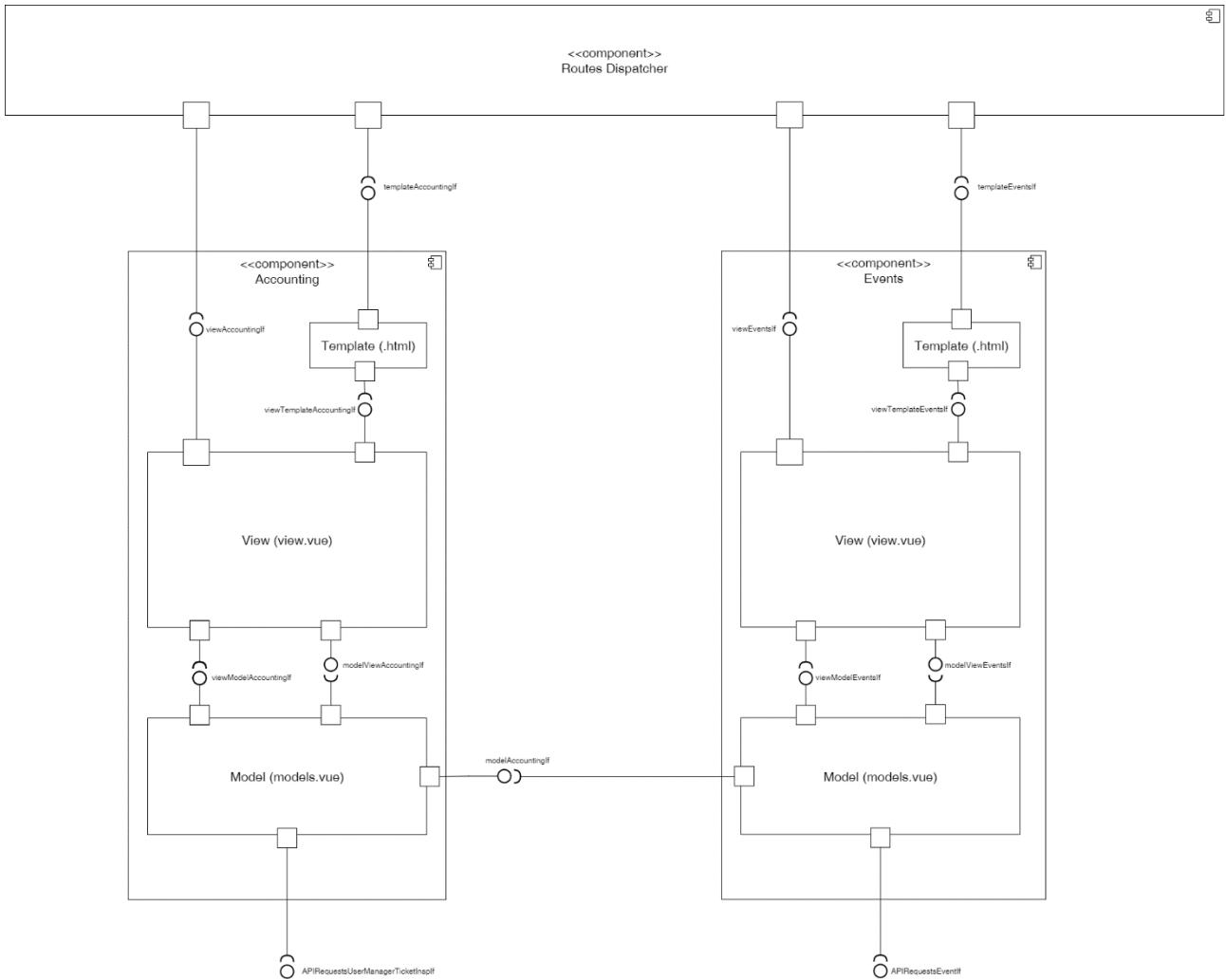
Il diagramma a componenti riportato successivamente (visibile nella sua interezza nella cartella DOCUMENTAZIONE/model_it0) permette di visualizzare i componenti che compongono funzionalmente le interconnessioni e le interfacce di comunicazione tra i diversi componenti.

L'utente accede attraverso un web browser alla piattaforma. Il web browser comunica direttamente con il routes dispatcher, che fornisce i percorsi di accesso alle varie pagine web che compongono EvenTour, e con il component dispatcher, che permette il salvataggio di componenti che hanno un impatto maggiore sul sistema e che vengono caricati frequentemente. Il component dispatcher ha bisogno anch'esso di comunicare con il routes dispatcher per l'accesso alle risorse. L'interazione è bidirezionale.

Per accedere alla pagina web richiesta dall'utente, il routes dispatcher richiede al macro-componente, Accounting, Events, Bookings e Access, la vista o dei singoli componenti tramite la view del componente vue oppure tramite un template in formato html che definisce la struttura e che comunica a sua volta con la vista in formato vue. Ogni vista, grazie al pattern architettonale MVVM, comunica direttamente e bidirezionalmente con il modello del macrocomponente. Essi comunicano tra loro tramite dei costrutti del View-Model, i quali permettono una comunicazione efficiente tra le due logiche.

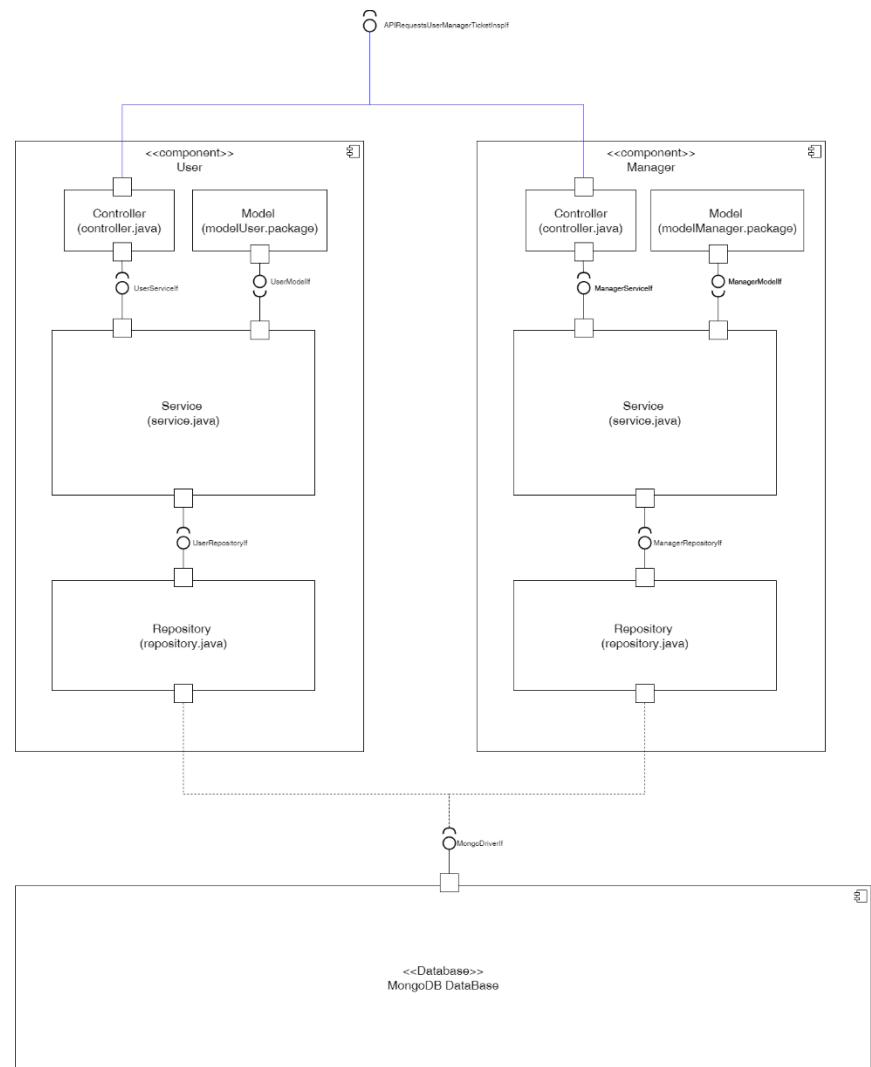


Modello 2.4 Component Diagram – dettaglio user client



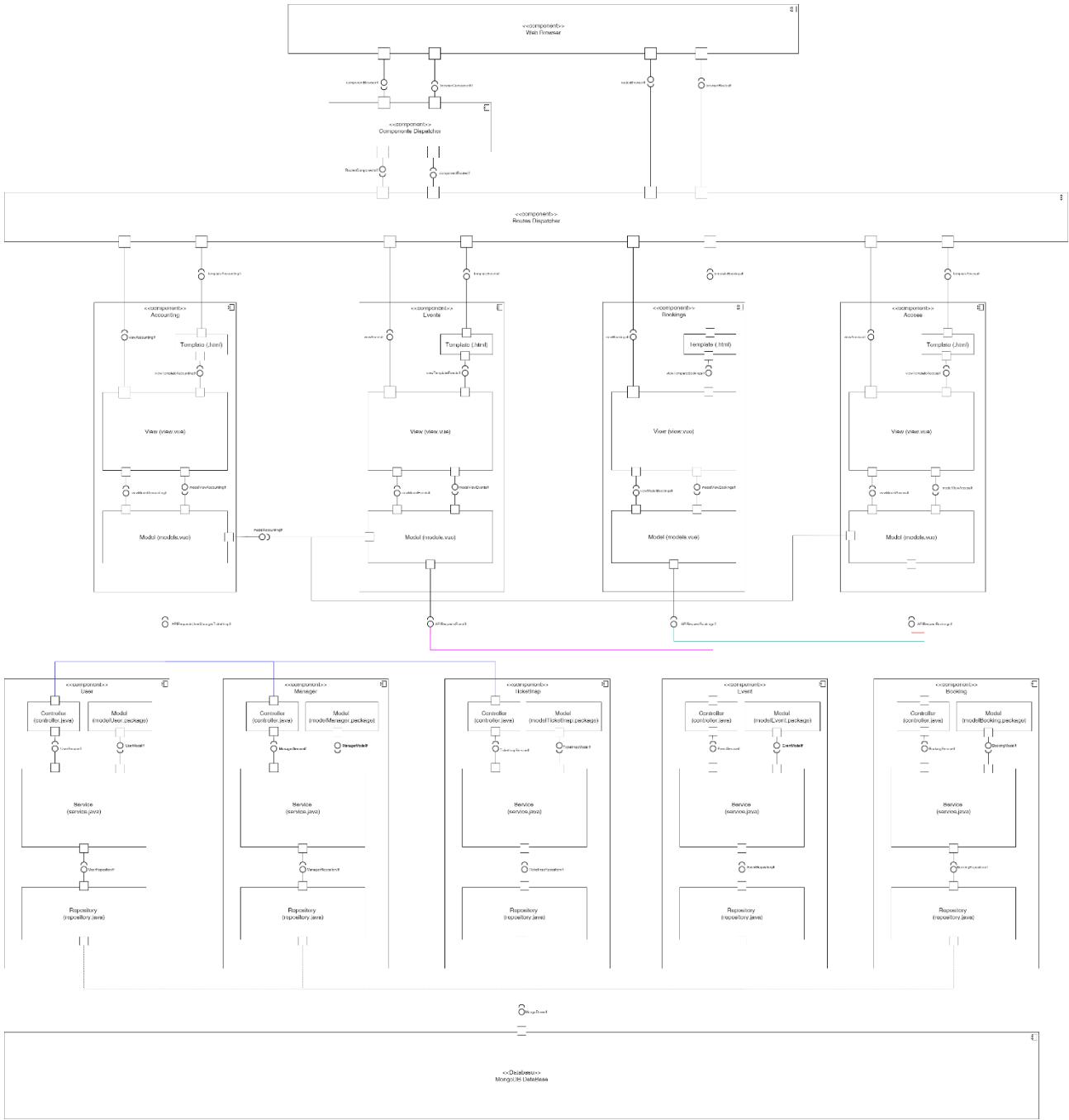
Modello 2.5 Component Diagram – dettaglio Vue framework

Ogni model, oltre che a comunicare con i modelli degli altri macrocomponenti, richiede i dati o elaborazioni da altri componenti, presenti come API, quali Utenti, Manager, Eventi, Prenotazioni e Ticket inspectors. La struttura base di ogni componente citato è formata secondo il pattern CSR (Controller-Service-Repository), in cui il Controller comunica direttamente con funzionalità del Service, che elabora dati sia grazie alle strutture dei dati presenti nel database come esplicitate nei modelli sia grazie ai repository tramite l'uso di interfacce Java e driver. Ogni repository richiederà dati grezzi al MongoDB database. Seppur non espresso in termini grafici nello schema sottostante, ogni service può avere accesso a più di un repository grazie al pattern singleton di Spring che mantiene l'univocità delle istanze.



Modello 2.6 Component Diagram – Dettaglio API

Si mostra di seguito il modello completo.



Modello 2.7 Component Diagram completo

Pattern architetturali

All'interno del progetto è identificabile il design pattern Singleton che in Spring non possiede tutte le caratteristiche. Se in linea di massima, il Singleton puro garantisce l'esistenza di una sola istanza di un oggetto per applicazione, anche il Singleton legato al framework Spring possiede tale caratteristica, ma questo vincolo può essere rilassato dato che Spring limita invece Singleton a un oggetto per contenitore, andando a creare solo un

bean di ogni tipo per contesto applicativo. Quindi differisce dalla definizione standard di Singleton poiché un'applicazione può avere più di un contenitore Spring. Pertanto, più oggetti della stessa classe possono esistere in una singola applicazione se abbiamo più contenitori.

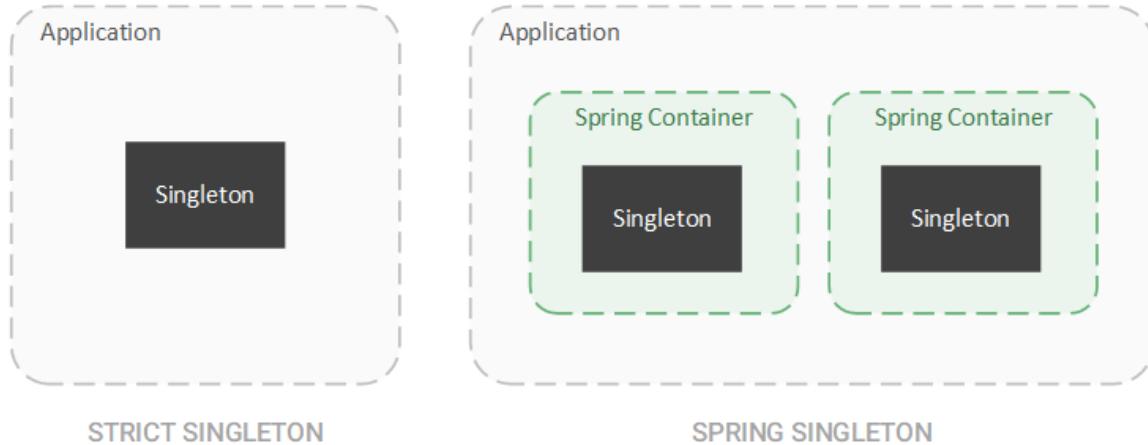


Figura 2.1 Springoboot Singleton pattern

Lo schema rappresenta il singleton visto dal framework Spring. Come standard, Spring crea tutti i Bean come singleton. Nel nostro progetto troviamo vari esempi di questo utilizzo; ecco qualche esempio:

```
@Service
public class UserService {

    /** The user repository. */
    @Autowired
    private UserRepository userRepository;

    /** The manager repository. */
    @Autowired
    private ManagerRepository managerRepository;

    /** The ticket insp repository. */
    @Autowired
    private TicketInspRepository ticketInspRepository;
```

Codice 2.1 Codice classe UserService

```

@Service
public class BookingService {

    /** The booking repository. */
    @Autowired
    private BookingRepository bookingRepository;

    /** The event repository. */
    @Autowired
    private EventRepository eventRepository;

    /** The event repository. */
    @Autowired
    private UserRepository userRepository;
}

```

Codice 2.2 Codice classe BookingService

Come si può osservare dal codice delle classi UserService e BookingService, entrambe fanno riferimento tramite l'annotazione `@Autowired` allo stesso repository (`UserRepository`), inserendo lo stesso bean in entrambi i Service. I controller citati in precedenza contengono solamente il service legato alla classe base di riferimento; quindi, applica ancora una volta Singleton anche su questi componenti.

Un altro pattern architettonale sfruttato è legato al framework Vue.js, ossia MVVM, Model-View-ViewModel, variante del Presentation Model Design. Il pattern astrae lo stato di view e il suo comportamento. Esistono 4 componenti che identificano il pattern MVVM:

- Model
- View
- View-Model: intermediario tra la vista e il modello, responsabile della logica della vista, interagendo principalmente con il modello invocando i metodi delle classi del modello, fornendo i dati in una forma che la vista può usare facilmente.
- Binder: meccanismo che permette la costante sincronizzazione tra la View e il View-Model, garantendo che i dati modificati dall'utente tramite la View vengano automaticamente riportati nel View-Model e che le modifiche avvenute su quest'ultimo vengano mostrato all'utente tramite la vista.

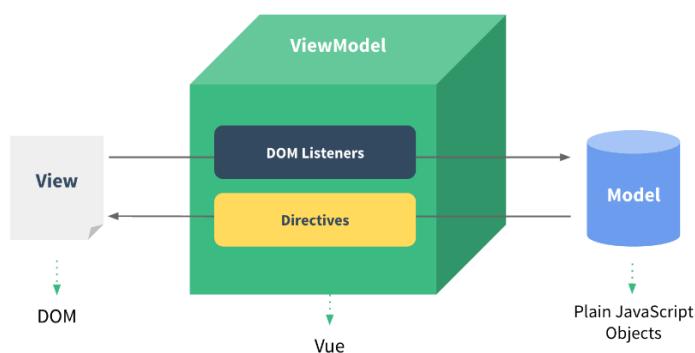


Figura 2.2 Schema modello MVVM

Un ulteriore pattern architettonale che si può trovare all'interno del progetto è legato all'architettura dei microservizi, in netta contrapposizione rispetto a quella monolitica, visto che è realizzata con componenti indipendenti che eseguono ciascun processo applicativo come un servizio che comunicano attraverso un'interfaccia ben definita che utilizza API, mantenendo tutti i vantaggi associati all'architettura quali agilità, scalabilità, flessibilità e semplicità di distribuzione. Di seguito uno schema che mostra l'utilizzo di un'architettura a microservizi, in particolare quella legata al concetto di API Gateway:

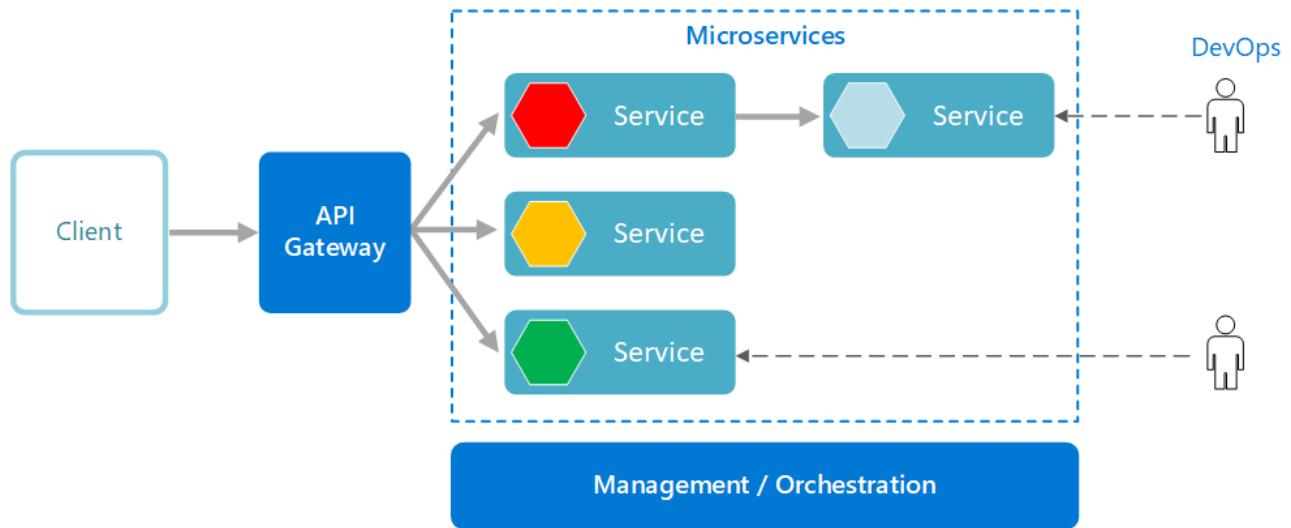


Figura 2.3 Microservice structure

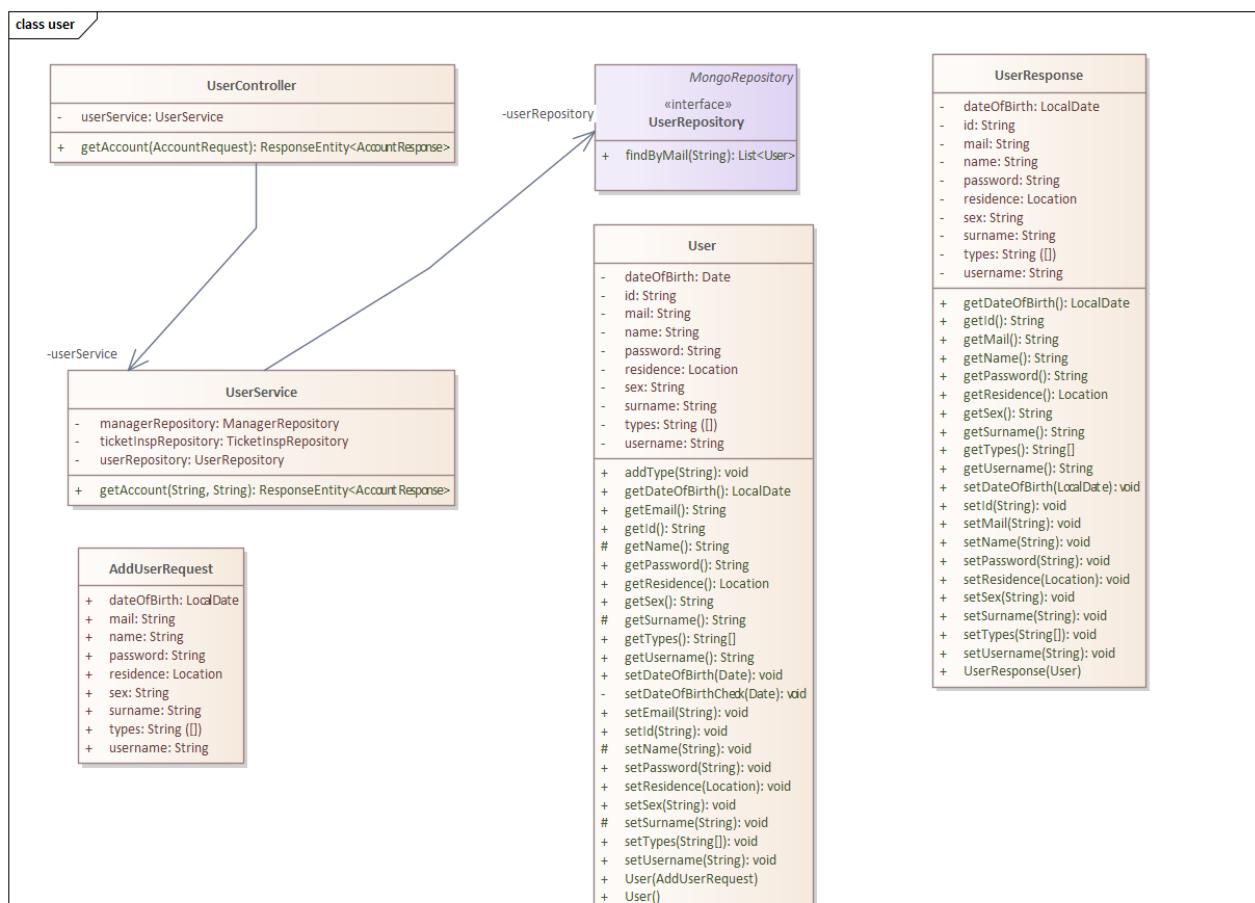
Il client si interfaccia con l'API Gateway per effettuare delle richieste e quest'ultimo richiamerà i servizi creati dagli sviluppatori per rispondere alle esigenze del client. Nel nostro caso, i servizi sono quelli legati agli utenti, alle prenotazioni, agli eventi, ai manager e ai ticket inspector.

Iterazione 1 (implementazione Iterazione 0)

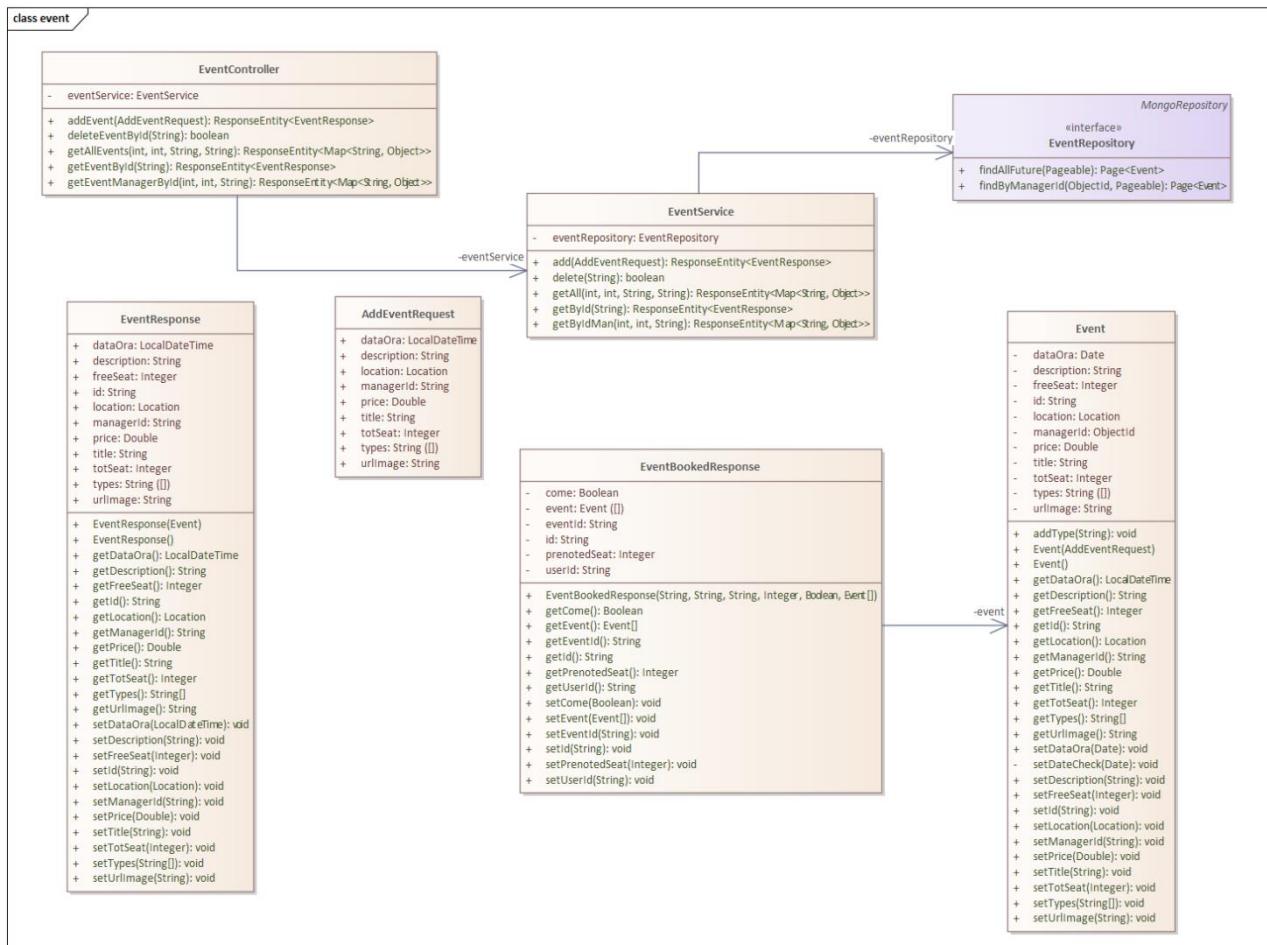
In questa iterazione si è implementata una prima parte del codice con riferimento alla modellizzazione espressa nella iterazione 0.

1. Class diagram dell'API

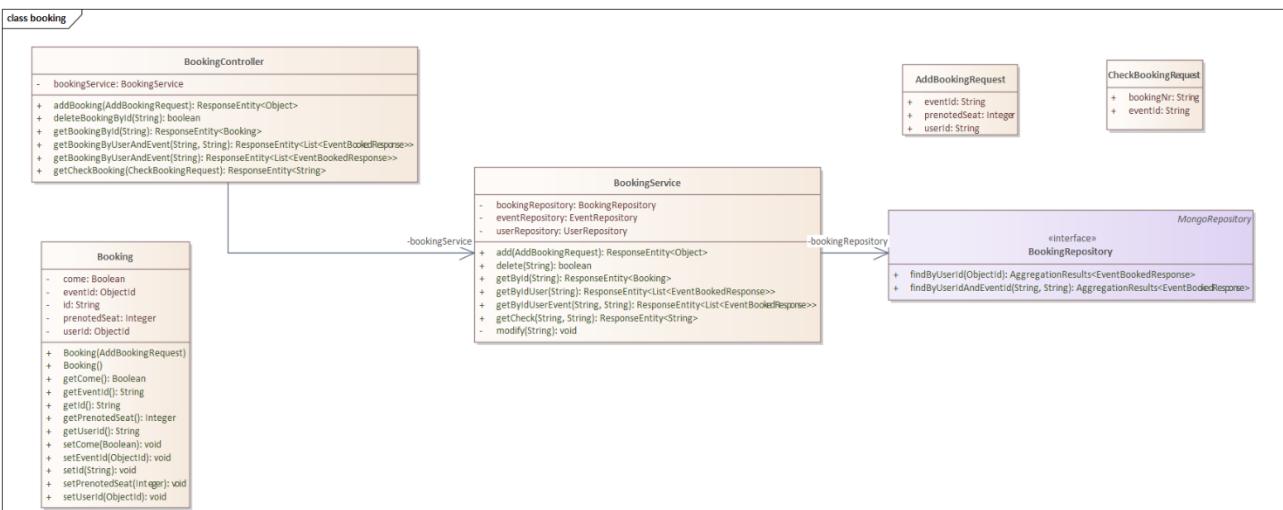
Si mostrano ora tutti i diagrammi delle classi suddivisi per ruolo.



Modello 3.1 User Class Diagram



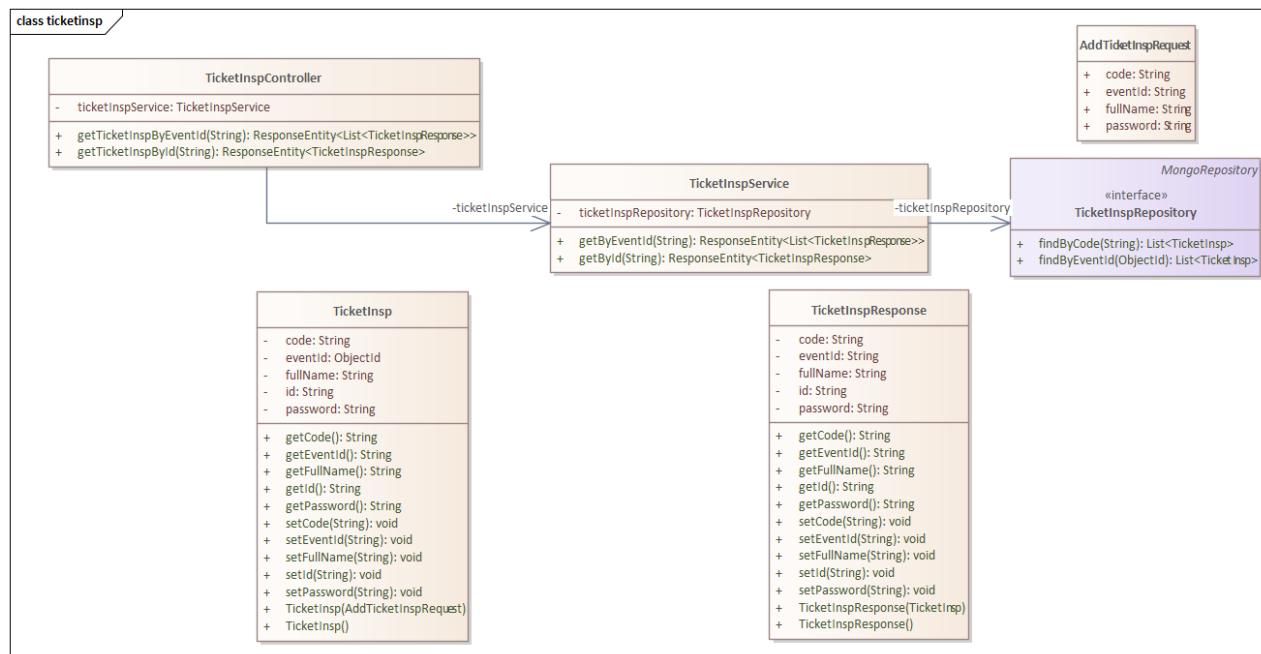
Modello 3.2 Event Class Diagram



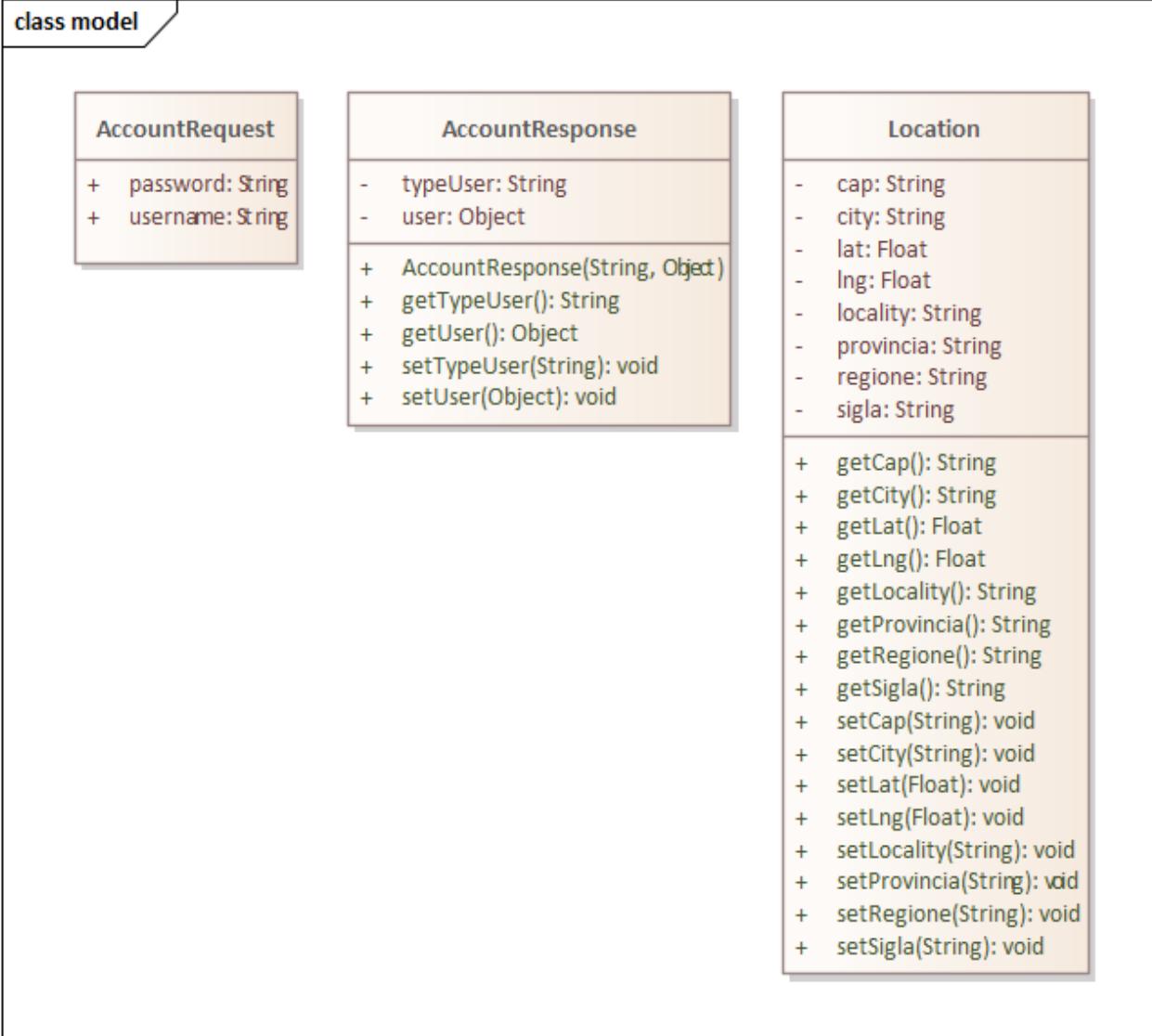
Modello 3.3. Booking Class Diagram



Modello 3.4 Manager Class Diagram



Modello 3.5 Ticket Inspector Class Diagram



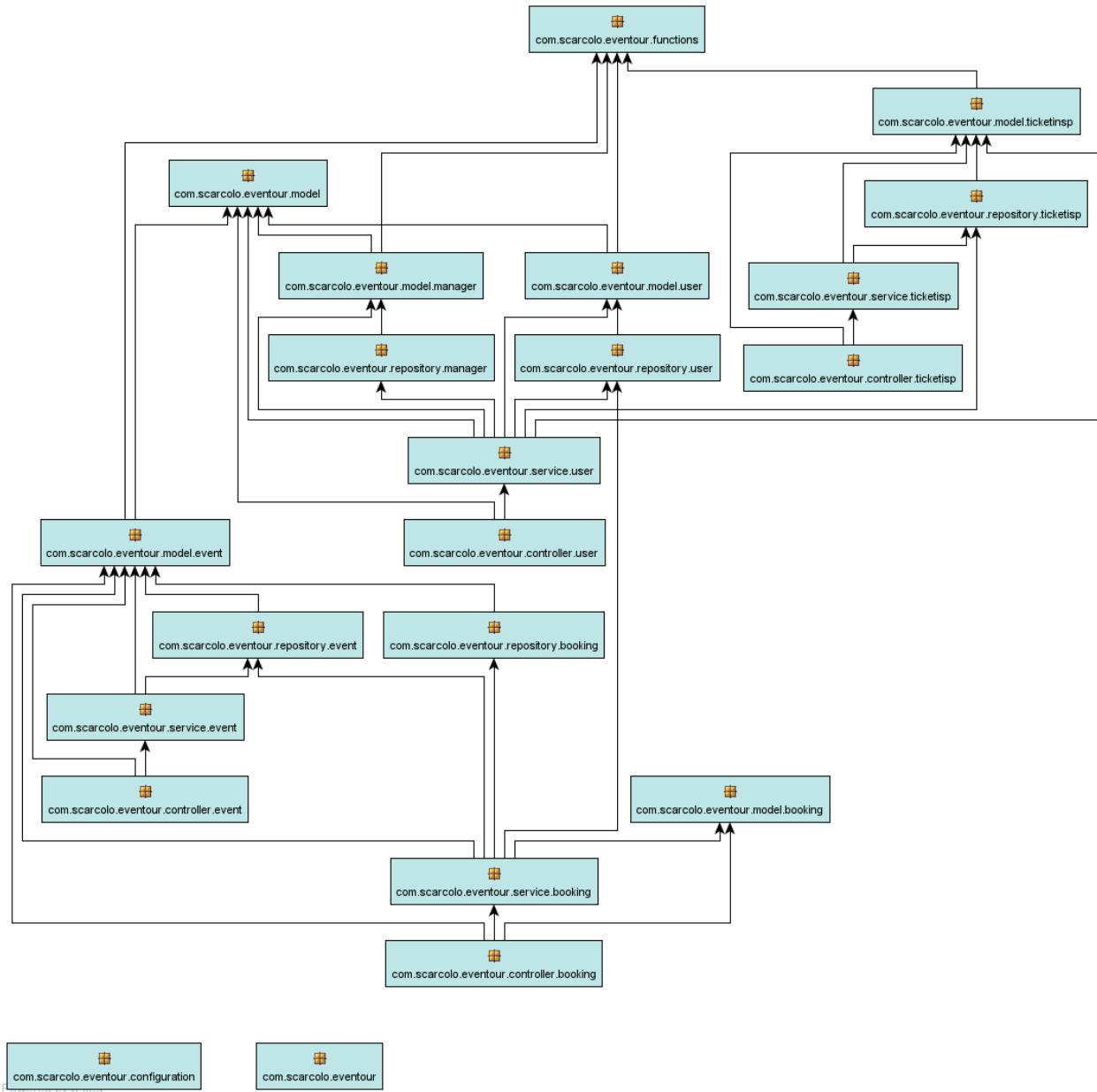
Modello 3.6 Other Models Diagram

Il *Modello 3.6* contiene le classi che sono usate da più di una classe appartenente a differenti package. In particolare, *AccountRequest* contiene i dati che vengono passati alla API durante la richiesta di login, *AccountResponse* è la risposta al login e *Location* è la struttura dati comune ai dati di residenza e location degli eventi.

Si vede chiaramente attraverso i diversi class diagram e il diagramma dei package, riportato di seguito, che la struttura uscente fa riferimento a una architettura a microservizi, descritta al meglio nella sezione “Pattern architetturali”.

I campi delle classi dei package “model” sono stati configurati per essere di tipo private, con metodi get e set associati. I class diagram precedenti fanno riferimento a dei package costruiti ad hoc per la migliore visualizzazione della struttura e per mostrare le relazioni di ogni ruolo o struttura dati.

Questo meccanismo si ripercuote in tutte le iterazioni successive, con ampliamenti ai singoli metodi di ogni classe.



Modello 3.7 Package Diagram

2. Definizione dei dati di test e di utilizzo

Per la generazione di dati di test e di utilizzo durante lo sviluppo del progetto, sono state effettuate varie generazioni di collezioni di test, per arrivare a circa 16.000 eventi, 76,862 prenotazioni, 5.000 utenti e 50 manager.

In prima istanza, l'idea del progetto era il mantenimento di un json server mock che generasse a ogni avvio nuovi dati. Ciò si ripercuoteva tuttavia sullo sviluppo, andando a complicare il testing e le "prove sul campo". Si è quindi deciso di sfruttare un database non relazionale come MongoDB con server online (Mongo Atlas) e generare in caso di necessità i dati necessari tramite lo script sviluppato precedentemente. Il progetto dell'intero json-server può essere trovato nella cartella lib del progetto. In esso si fa uso, per la generazione di dati, di una API che fornisce tutti i dati sui comuni e locazioni (durante lo sviluppo essa è stata modificata dall'autore, con conseguente aggiornamento del codice sviluppato da noi) e di una libreria javascript per la generazione randomica di dati di vario tipo.

Si riporta nella pagina seguente la parte di codice che genera i 16000 eventi.

```

var faker = require('faker/locale/it');
var jsonData = require('./province.json');
var json=require('./comuni.json');

function generateEvents () {
    for (var id = 0; id <= 16000; id++) {
        var title ='EVENTO '+id;
        var description = faker.lorem.paragraph();
        var paeseIndex = faker.datatype.number({
            'min': 0,
            'max': resultC.length-1
        });
        var via=faker.address.streetName() + ", " +faker.datatype.number({
            'min': 0,
            'max': 100
        });
        var location=json[paeseIndex];
        var provincia=jsonData.find(item => item.codice==location.provincia);
        var price=faker.datatype.float({'min':0,'max':100,'precision':0.10});

        var type1=faker.random.arrayElement(["1.1.1","1.1.2","1.1.3","1.2.1",
            "1.2.2","1.2.3","1.2.4","1.3.1","1.3.3","1.4.1",
            "1.4.2","1.4.3","1.4.4","2.1","2.2","2.3","2.4","2.5",
            "2.6","2.7","3.1","3.2","3.3","3.4","3.5"]);
        var type2=type1;
        while(type2==type1)
            type2=faker.random.arrayElement(["1.1.1","1.1.2","1.1.3","1.2.1",
                "1.2.2","1.2.3","1.2.4","1.3.1","1.3.3","1.4.1",
                "1.4.2","1.4.3","1.4.4","2.1","2.2","2.3","2.4","2.5",
                "2.6","2.7","3.1","3.2","3.3","3.4","3.5"]);
        var datetime=faker.date.past(1).toISOString();

        var manager;

        /*do*/
        manager=faker.random.arrayElement(managerA);
        /*}while(manager.residence.regione!=(provincia.regione.charAt(0)
            .toUpperCase() + provincia.regione.slice(1)))*/
        var urlImage='https://picsum.photos/seed/'+id+'/640/480';
        var totSeat=faker.datatype.number({min:50, max:100});
        var freeSeat=faker.datatype.number({min:0, max:totSeat});

        events.push({
            "_id": mongoObjectID() require('mongoose').Types.ObjectId(),
            "title": title,
            "description": description,
            "location": {
                "locality" : via,
                "city" : location.nome,
                "cap" : location.cap,
                "provincia" : provincia.nome.charAt(0)
                    .toUpperCase() + provincia.nome.slice(1),
                "sigla" : provincia.sigla.toUpperCase(),
                "regione" : provincia.regione.charAt(0)
                    .toUpperCase() + provincia.regione.slice(1),
                "lat":location.lat,
                "lng":location.lng
            },
            "types": [
                type1,
                type2
            ],
            "dataOra": datetime,
            "managerId": manager._id,
            "urlImage": urlImage,
            "price" : price,
            "freeSeat": freeSeat,
            "totSeat": totSeat
        })
    }
    return events
};

```

Codice 3.1 Codice generatore degli eventi

3. Algoritmo evenTour

Lo scopo della versione iniziale dell'algoritmo progettato è quella di trovare un numero di eventi, definito dalla invocazione della API, che siano disponibili e che siano nella regione di residenza dell'utente. Essa, in questa versione, è stata solo progettata in quanto la vera e propria implementazione è stata fatta nella iterazione successiva.

La funzione, dopo aver analizzato i dettagli dell'utente che ne richiede l'eventour, ricerca tutti gli eventi della regione di appartenenza dell'utente che siano disponibili ordinandoli per data crescente ed esclude quelli già prenotati dall'utente e quelli che sono nella stessa data di quelli prenotati. Per ogni evento rimanente:

1. Viene effettuato un controllo di raggiungimento della soluzione (sono stati trovati N eventi richiesti):
 1. se veritiero, viene restituita la soluzione,
 2. altrimenti viene selezionato l'evento se rispetta due condizioni:
 - Almeno uno dei tipi di quell'evento è uguale a uno dei tipi preferiti dall'utente.
 - La soluzione da restituire non contiene ancora eventi oppure l'ultimo evento inserito nella soluzione da restituire non ha la stessa data dell'evento che si vuole aggiungere alla soluzione.
2. Se la soluzione non contiene N eventi e tutti gli eventi sono stati analizzati, allora viene restituito un errore.

Si riporta lo pseudocodice dell'algoritmo:

```

1. //id: id dell'utente, N: numero eventi richiesti da un utente
2. function EvenTour(id, N)
3. {
4.     datiUtente <- ricercaUtenteDaID(id);
5.     listaEventi <- ricercaEventiPerRegioneConPostiDisponibiliOrdinatiPerData
                           (datiUtente.regione);
6.     bookingData <- ricercaPrenotazioniDaIdUtente(id);

7.     s <- Lista vuota //Soluzione da restituire
8.     c <- 0 //Numero eventi inseriti nella soluzione
9.
10.    SE (bookingData ha eventi) ALLORA
11.    {
12.        PER OGNI eventoPrenotato IN bookingData
13.        {
14.            rimuoviEventoGiàPrenotato(eventoPrenotato);
15.            rimuoviEventoConDateUguali(eventoPrenotato);
16.        }
17.    }
18.
19.    PER OGNI evento IN listaEventi
20.    {
21.        sel <- false //indica se l'evento è già stato inserito visto il controllo dei tipi
22.        SE (c = N) ALLORA
23.        {
24.            RETURN s;
25.        }
26.        PER OGNI tipo IN evento.tipi
27.        {
28.            SE (NOT sel AND datiUtenti.tipi CONTIENE tipo)
29.            {
30.                SE (s = VUOTO OR NOT
                           controlloDateUguali(ultimo elemento inserito di s, evento)
31.                {
32.                    s = s U evento;
33.                    sel <- true;
34.                    c <- c + 1
35.                }
36.            }
37.        }
38.    }
39.
40.    return ERROR;
41.
42. }

```

Codice 3.2 Codice evenTour iterazione 1

4. Testing interfaccia grafica

Per l'interfaccia grafica sono state fatte due tipi di analisi.

Analisi statica: EsLint

```
module.exports = {
  "env": {
    "browser": true,
    "es6": true
  },
  "extends": ["standard", "plugin:vue/recommended"],
  "globals": {
    "Atomics": "readonly",
    "SharedArrayBuffer": "readonly"
  },
  "parserOptions": {
    "ecmaVersion": 2018,
    "sourceType": "module"
  },
  "plugins": ["vue"],
  "rules": {}
};
```

Codice 3.3 File di configurazione EsLint per l'analisi statica nella parte client

Per quanto concerne l'analisi statica della parte in Vue.js, lo strumento di EsLint ha permesso di individuare i potenziali problemi legati al codice scritto, soprattutto per quanto riguarda la parte strettamente grafica come, ad esempio, i tag di chiusura di elementi HTML, l'ordine degli attributi legati a determinati tag e l'indentazione, grazie ai quali si sono formattati meglio i documenti garantendo uno stile più corretto di scrittura dal punto di vista qualitativo rispetto al formato e al testo digitato. Per quanto riguarda il codice scritto in linguaggio JavaScript, ha permesso di evidenziare problematiche legate all'uso di Props e all'uso di costrutti ormai di uso comune nella maggior parte dei linguaggi, ma che, nel contesto analizzato, non corrispondono al metodo più corretto da implementare. Nel caso di props, ossia le variabili ottenute dal processo padre, il problema è relativo all'istanziazione delle stesse, ma senza alcuna soluzione trovata, almeno per il momento. Sotto un esempio di warning legato a questo tipo di problema.

128:5 warning Prop 'backgroundImage' requires default value to be set

Per quanto riguarda i costrutti a cui si faceva riferimento precedentemente, la questione principale è legata all'uso del simbolo di uguaglianza tra due variabili; se, nei linguaggi classici come java, il simbolo “`==`” rappresenta un confronto tra due valori per verificarne l'uguaglianza in termini di valore, in JavaScript, per la mancanza di tipizzazione, è fortemente consigliato l'uso del simbolo “`==`” poiché non solo effettua un confronto tra valori, ma anche un confronto tra tipi, visto che JavaScript non è tipizzato ma possiede lo stesso una rappresentazione minima dei vari tipi di oggetto. Così facendo non si creano potenziali confusioni dettate dal simbolo “`==`”. Nel progetto si è comunque deciso di mantenere il simbolo “`==`” poiché, nell'implementazione, non ha fornito particolari interferenze, visto i tipi semplici che abbiamo sfruttato e ben definiti in termini di tipo. Sotto un esempio di questo tipo di errore:

```
218:43 error Expected '===' and instead saw '=='
```

Analisi dinamica: NightWatch

```
'Home Page Loading' : function(browser) {
    browser
        .url('http://localhost:8080')
        .waitForElementVisible('body')
        .assert.titleContains('frontend-springboot-event')
        .waitForElementVisible('div[id=container]')
        .end();
},
'From Home to Login' : function(browser) {
    browser
        .url('http://localhost:8080')
        .waitForElementVisible('button[id=login]')
        .click('button[id=login]')
        .assert.urlEquals('http://localhost:8080/login')
        .end();
},
'Login and Logout User' : function(browser) {
    browser
        .url('http://localhost:8080')
        .click('button[id=login]')
        .assert.urlEquals('http://localhost:8080/login')
        .setValue('input[id=username]', 'Colombano72@hotmail.com')
        .assert.value('input[id=username]', 'Colombano72@hotmail.com')
        .setValue('input[id=password]', 'yelukare')
        .click('button[id=btnlogin]')
        .waitForElementVisible('button[id=logout]')
        .assert.urlEquals('http://localhost:8080/')
        .waitForElementVisible('button[id=logout]')
        .click('button[id=logout]')
        .waitForElementVisible('button[id=login]')
        .end();
},
'Login and Try Access Denied' : function(browser) {
    browser
        .url('http://localhost:8080')
        .click('button[id=login]')
        .assert.urlEquals('http://localhost:8080/login')
        .setValue('input[id=username]', 'Colombano72@hotmail.com')
        .setValue('input[id=password]', 'yelukare')
        .click('button[id=btnlogin]')
        .waitForElementVisible('button[id=logout]')
        .assert.urlEquals('http://localhost:8080/')
        .waitForElementVisible('button[id=logout]')
        .url('http://localhost:8080/homemanager')
        .assert.urlEquals('http://localhost:8080/')
        .waitForElementVisible('button[id=logout]')
        .end();
}
```

Codice 3.4 Test dinamico effettuato sulla classe User

Sono stati effettuati ulteriori test riguardo le classi dei Manager e dei Ticket Inspector, tali test non sono rappresentati poiché brevi e relativi solo all'accesso da parte del tipo di

account che si è registrato. In tutti i casi i risultati sono positivi e tutti gli assert sono stati confermati e validati. Ogni test ha un output per ogni classe testata che identifica quanti controlli sono stati effettuati e in quanto tempo. Gli output sono presenti all'interno della cartella di test della parte di progetto legata al client. Di seguito uno screenshot dell'output legato alle funzioni elencate nel codice 3.2:

```
<?xml version="1.0" encoding="UTF-8" ?>
<testsuites errors="0"
             failures="0"
             tests="4">

    <testsuite name="UserTest"
               errors="0" failures="0" hostname="" id="" package="UserTest" skipped="0"
               tests="4" time="96.82" timestamp="">

        <testcase name="Home Page Loading" classname="UserTest" time="20.73"
assertions="3">

            </testcase>

        <testcase name="From Home to Login" classname="UserTest" time="22.86"
assertions="2">

            </testcase>

        <testcase name="Login and Logout User" classname="UserTest" time="24.17"
assertions="6">

            </testcase>

        <testcase name="Login and Try Access Denied" classname="UserTest" time="29.07"
assertions="6">

            </testcase>

    </testsuite>
</testsuites>
```

Codice 3.5 Risultato test del codice 3.4

5. Testing API

Analisi statica

Per quanto riguarda l'analisi statica effettuata sul server per l'esposizione delle API, dopo aver generato una copia di test in MongoDB Atlas e aver modificato la stringa di connessione nell'API, si sono riscontrati circa 25 bugs segnalati da SpotBugs e varie violazioni segnalate grazie a PMD.

Per quanto riguarda Spotbugs, i bug segnalati sono stati risolti partendo da quelli più critici per finire a quelli meno critici.

Per quanto riguarda PMD, le violazioni di livello Blocker e Critical (bandiera rossa e azzurra) sono stati risolte, mentre quelli di livello inferiore sono stati controllate e analizzate senza correggerle.

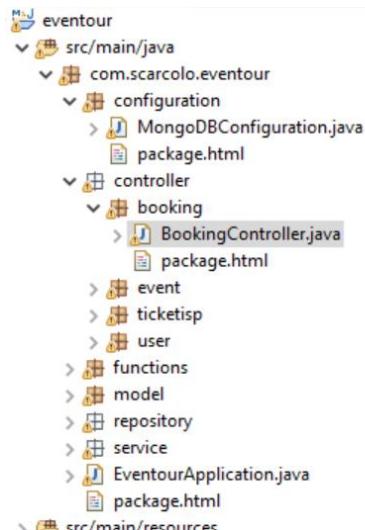


Figura 3.1 Package progetto in Eclipse con visualizzazione dei bug segnalati da spotbugs e delle violazioni di livello alto mostrate

Il report di JArchitect evidenzia inoltre il grafo delle dipendenze mostrato di seguito:

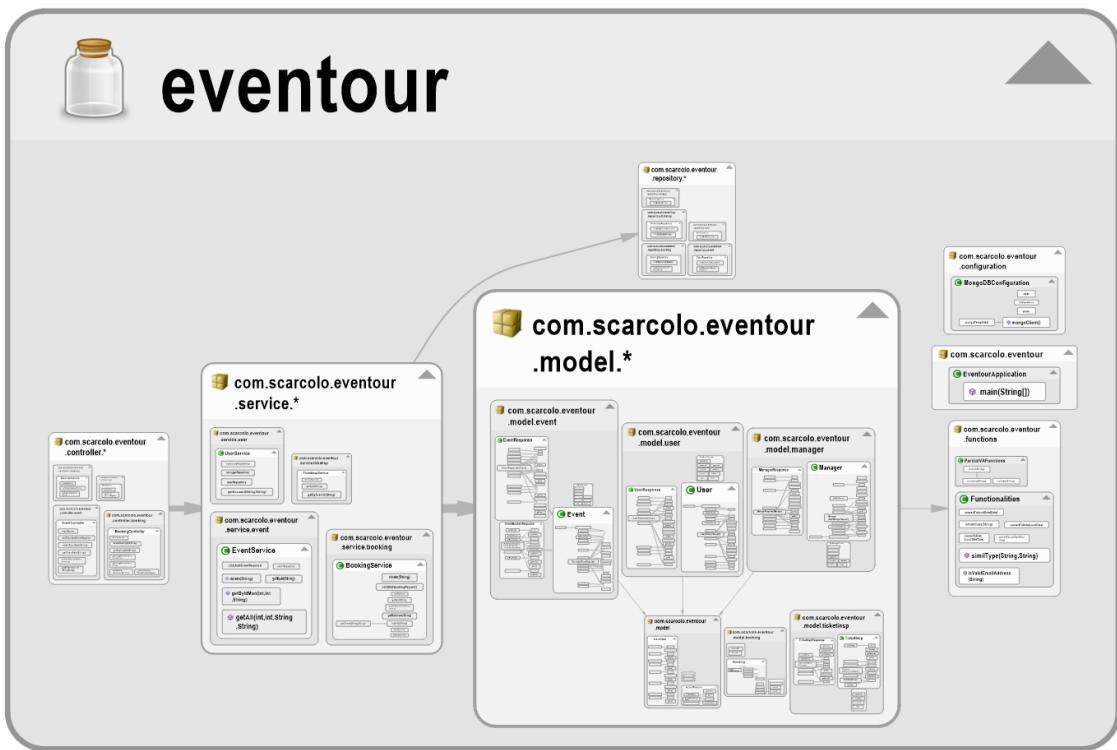


Figura 3.2 Modello report JArchitect

Analisi dinamica

Per l'esecuzione dei test, dopo previa modifica della stringa di connessione contenuta nel file `MongoDBConfiguration.java`, si deve recarsi nel package `com.scarcolo.eventour.configuration` e si deve avviare i test posizionandosi nella source folder dei test e con tasto destro, effettuare *Run As>JUnit Test*.

Questo vale per tutte le iterazioni.

Tramite JUnit si sono svolti numerosi test, come visibile nella figura di seguito. Purtroppo i test non possono essere esaustivi di tutte le funzionalità presenti nella API in quanto tutti i metodi get e set non vengono usati spesso a causa del solo test dei costruttori richiamati dalle chiamate API effettuate. Si nota tuttavia che i test hanno coperto tutte le chiamate realizzate nella iterazione corrente.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
eventour	80,0 %	3.218	804	4.022
> src/test/java	100,0 %	1.556	0	1.556
> src/main/java	67,4 %	1.662	804	2.466
> com.scarcolo.eventour.configuration	100,0 %	52	0	52
> com.scarcolo.eventour.controller.booking	100,0 %	37	0	37
> com.scarcolo.eventour.controller.event	100,0 %	33	0	33
> com.scarcolo.eventour.controller.ticketisp	100,0 %	13	0	13
> com.scarcolo.eventour.controller.user	100,0 %	11	0	11
> com.scarcolo.eventour.model.booking	98,7 %	78	1	79
> com.scarcolo.eventour.service.user	96,7 %	175	6	181
> com.scarcolo.eventour.service.ticketisp	92,2 %	71	6	77
> com.scarcolo.eventour.service.booking	91,5 %	258	24	282
> com.scarcolo.eventour.model	90,6 %	77	8	85
> com.scarcolo.eventour.model.event	72,1 %	272	105	377
> com.scarcolo.eventour.service.event	70,3 %	282	119	401
> com.scarcolo.eventour.model.ticketisp	54,8 %	80	66	146
> com.scarcolo.eventour	37,5 %	3	5	8
> com.scarcolo.eventour.model.user	35,1 %	107	198	305
> com.scarcolo.eventour.model.manager	33,6 %	97	192	289
> com.scarcolo.eventour.functions	17,8 %	16	74	90

Figura 3.3 Copertura test JUnit

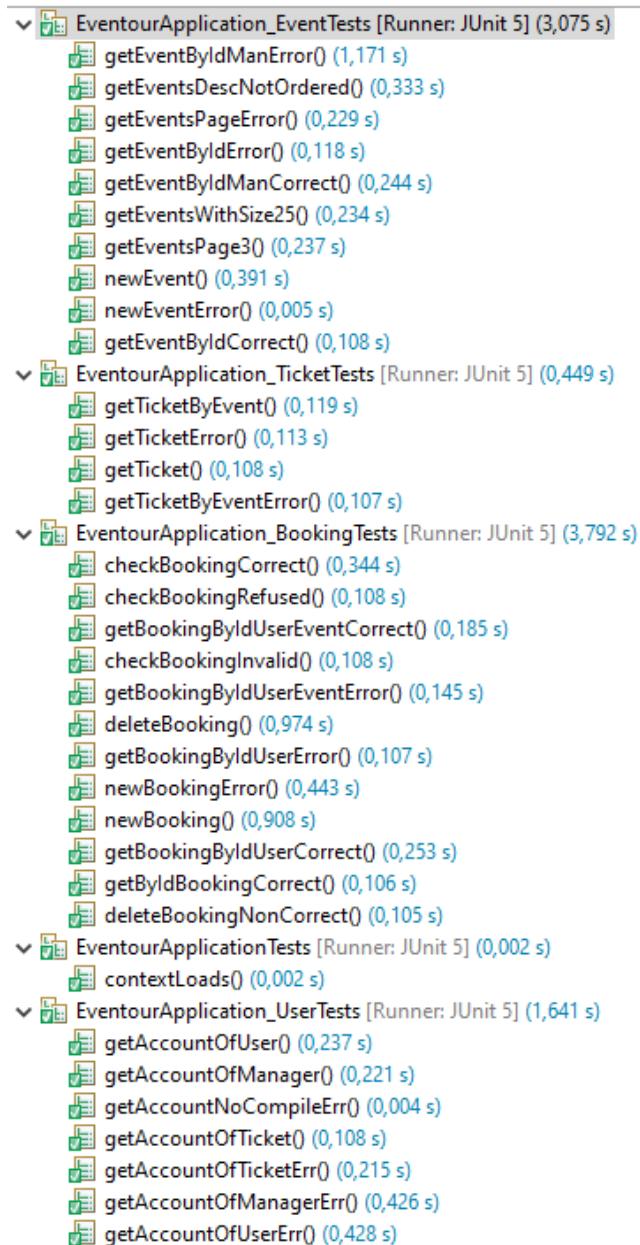


Figura 3.4 Test effettuati e passati

Iterazione 2

Durante questa iterazione sono state introdotte numerose funzionalità quali ricerche di eventi più complete, possibilità di visualizzare un evenTour, pagamento di prenotazioni, visualizzazione degli eventi prenotati e gestione completa degli eventi da parte del manager.

1. Use Cases

Descrizione testuale

Sono stati definiti nuovi casi d'uso:

Nome	UC1.3: Visualizzazione per tipo
Descrizione	Mostra tutti gli eventi disponibili di un tipo o insieme di tipi indicato/i
Attori	User
Obiettivi	Mostrare gli eventi disponibili del tipo/tipi scelto/i
Precondizioni	L'utente deve essere autenticato
Passi	<ol style="list-style-type: none">1. L'utente entra nella pagina di visualizzazione degli eventi2. Il sistema stampa a video tutti gli eventi disponibili in ordine crescente di data (dal più vicino al più lontano)3. L'utente richiede la visualizzazione di eventi di uno o più tipi specifici4. Il sistema mostra a video tutti gli eventi ordinati per data crescente appartenenti a uno dei tipi specificati5. L'utente può selezionare un evento di interesse6. Il sistema mostra altri dettagli relativi a esso e la possibilità di gestirne la prenotazione
Casi d'uso associati / flussi alternativi	Specializzazione UC1: Visualizzazione Eventi
Postcondizioni	/

Nome	UC1.4: Visualizzazione per preferenze
Descrizione	Mostra tutti gli eventi disponibili in base a uno o più tipi preferiti dall'utente
Attori	User
Obiettivi	Mostrare gli eventi disponibili di tipi preferiti
Precondizioni	L'utente deve essere autenticato
Passi	<ol style="list-style-type: none"> 1. L'utente entra nella pagina di visualizzazione degli eventi 2. Il sistema stampa a video tutti gli eventi disponibili in ordine crescente di data (dal più vicino al più lontano) 3. L'utente richiede la visualizzazione di eventi secondo le sue preferenze 4. Il sistema mostra a video tutti gli eventi ordinati per data crescente con tipi preferiti dall'utente inseriti nella registrazione 5. L'utente può selezionare un evento di interesse 6. Il sistema mostra altri dettagli relativi a esso e la possibilità di gestirne la prenotazione
Casi d'uso associati / flussi alternativi	Specializzazione UC1: Visualizzazione Eventi
Postcondizioni	/

Nome	UC11: Impostazioni Utente
Descrizione	Permette di cambiare le impostazioni legate all'utente
Attori	User
Obiettivi	Cambiare impostazioni dell'utente
Precondizioni	L'utente deve essere loggato
Passi	<ol style="list-style-type: none"> 1. L'utente entra nella pagina delle impostazioni 2. Il sistema mostra tutte le informazioni legate all'utente 3. Se l'utente vuole modificare i dati

	<p>3.1.L'utente inserisce nei campi disponibili per la modifica i valori che intende modificare</p> <p>3.2.L'utente salva le informazioni</p> <p>3.3.Il sistema aggiorna le informazioni modificate nel database</p>
Casi d'uso associati / flussi alternativi	Flusso alternativo: L'utente può uscire dalla modifica delle impostazioni senza effettuare il salvataggio, perdendo così tutte le modifiche effettuate.
Postcondizioni	Le informazioni sono modificate, se l'utente ha effettuato cambiamenti.

Nome	UC12: Eventi Utente
Descrizione	Visualizzazione degli eventi prenotati da un utente e gestione di essi
Attori	User
Obiettivi	Visualizzazione degli eventi prenotati
Precondizioni	L'utente deve essere loggato
Passi	<ol style="list-style-type: none"> 1. L'utente accede alla pagina di visualizzazione degli eventi prenotati 2. Il sistema mostra tutti gli eventi prenotati dall'utente 3. L'utente può selezionare un evento tra quelli prenotati 4. Il sistema mostra altri dettagli relativi a esso e la possibilità di gestirne la prenotazione 5. In caso di cambio di alcune prenotazione, ritornando sulla pagina saranno riaggiorionate le prenotazioni
Casi d'uso associati / flussi alternativi	/
Postcondizioni	Se sono state apportate modifiche alla prenotazione, i dati delle prenotazioni sono aggiornati.

Nome	UC13: Eventour
-------------	----------------

Descrizione	Visualizzazione di un eventour
Attori	User
Obiettivi	Visualizzazione di un tour di eventi
Precondizioni	L'utente deve essere loggato
Passi	<ol style="list-style-type: none"> 1. L'utente accede alla pagina di visualizzazione dell'eventour 2. Il sistema mostra un tour di eventi ottimizzato sulla base dell'utente e delle sue informazioni 3. L'utente può visualizzare più dettagli su un evento 4. Il sistema mostra altri dettagli relativi a esso e la possibilità di gestirne la prenotazione
Casi d'uso associati / flussi alternativi	/
Postcondizioni	L'utente, in caso di prenotazione di uno, alcuni o tutti gli eventi proposti dall'eventour, ritroverà questi eventi prenotati nella pagina dedicata

Nome	UC14: Pagamento
Descrizione	Viene effettuato il pagamento dell'evento da prenotare.
Attori	User
Obiettivi	Effettua il pagamento di un evento
Precondizioni	<p>L'utente deve essere loggato</p> <p>L'utente deve essere nella pagina della visualizzazione dei dettagli di un evento o nella pagina della visualizzazione di un eventour</p>
Passi	<ol style="list-style-type: none"> 1. Il sistema, durante la prenotazione, controlla che l'evento non sia gratuito e mostra in tal caso la richiesta di inserimento dei dati di pagamento 2.a. L'attore vuole confermare il pagamento e la prenotazione <ol style="list-style-type: none"> 2.a.1. Il sistema carica una pagina di inserimento dati per il metodo di pagamento

	<p>2.a.2. L'attore inserisce i dati della carta</p> <p>2.a.3. L'attore conferma i dati inseriti</p> <p>2.a.4. Il sistema controlla i dati inseriti e richiede il pagamento</p> <p>2.a.5.a. Se la transazione viene confermata, il sistema avvisa della conferma del pagamento e conferma la prenotazione</p> <p>2.a.5.b. Se la transazione non viene confermata, il sistema avvisa dell'errore riscontrato durante il pagamento e non conferma la prenotazione</p> <p>2.b. L'attore non vuole confermare il pagamento</p> <p>2.b.1. L'attore annulla la richiesta di pagamento o riaggiorna la pagina</p> <p>2.b.2. Il sistema non effettua la prenotazione</p>
Casi d'uso associati / flussi alternativi	<<extend>> UC4: Prenotazione evento Flusso alternativo 1: L'attore non conferma i dati inseriti con conseguente cancellazione delle informazioni. Flusso alternativo 2: L'attore inserisce le informazioni del metodo di pagamento, ma non richiede il pagamento, uscendo prima della conferma di transazione.
Postcondizioni	L'attore ottiene un messaggio di notifica di avvenuto pagamento se tutto è andato a buon fine

Nome	UC15: Report Manager
Descrizione	Visualizzazione dei report relativi agli eventi e ai dati del manager.
Attori	Manager
Obiettivi	Visualizzazione report attività manager
Precondizioni	Il manager deve essere loggato
Passi	1. Il manager entra nella pagina dei report

	2. Il sistema mostra a video l'elenco di tutti i report degli eventi passati
Casi d'uso associati / flussi alternativi	/
Postcondizioni	/

Nome	UC16: Cancellazione Evento
Descrizione	Un evento viene cancellato dall'elenco degli eventi creati.
Attori	Manager
Obiettivi	Cancellare un evento
Precondizioni	<ul style="list-style-type: none"> • Il manager deve essere loggato • Il manager deve essere nella pagina di visualizzazione di un evento specifico
Passi	<ol style="list-style-type: none"> 1. Il manager seleziona un evento creato 2. Il manager clicca su "Cancella evento" 3. Il sistema cancella l'evento dal database. 4. Il sistema aggiorna la pagina di visualizzazione degli eventi
Casi d'uso associati / flussi alternativi	/
Postcondizioni	L'evento è cancellato

Nome	UC17: Modifica Evento
Descrizione	Vengono modificate le informazioni relative ad un evento
Attori	Manager
Obiettivi	Modifica informazioni relative ad un evento
Precondizioni	<ul style="list-style-type: none"> • Il manager deve essere loggato • Il manager deve essere nella pagina di visualizzazione di un evento specifico

Passi	<ol style="list-style-type: none"> 1. Il manager clicca su "Modifica Evento" 2. Il sistema mostra le informazioni modificabili dell'evento 3. Il manager può modificare le informazioni compilando i campi modificabili 4. Il manager clicca sul pulsante di salvataggio delle informazioni 5. Il sistema salva le informazioni dell'evento 6. Il sistema mostra a video l'evento e i suoi dettagli aggiornati.
Casi d'uso associati / flussi alternativi	Flusso alternativo: L'uscita dalla pagina prima del salvataggio, comporta la mancata modifica delle informazioni.
Postcondizioni	Se sono state apportate modifiche, i dati dell'evento sono aggiornati.

Alcuni casi d'uso sono stati modificati o perfezionati per l'implementazione successiva di funzionalità aggiuntive:

Nome	UC1.1: Visualizzazione per data
Descrizione	Mostra tutti gli eventi disponibili ordinati per data o in intervallo di date
Attori	User
Obiettivi	Mostrare gli eventi disponibili in un certo ordine o data
Precondizioni	Se richiesto per data specifica, l'utente deve essere autenticato
Passi	<ol style="list-style-type: none"> 1. L'utente entra nella pagina di visualizzazione degli eventi 2. Il sistema stampa a video tutti gli eventi disponibili in ordine crescente di data (dal più vicino al più lontano) 3.a. Vista ordinata di un certo ordine <ul style="list-style-type: none"> 3.a.1. L'utente sceglie l'ordine con il quale mostrare gli eventi (dal più vicino al più lontano come date o viceversa)

	<p>3.a.2. Il sistema mostra a video gli eventi ordinati secondo l'ordine richiesto</p> <p>3.b. Vista di eventi in una certa data o date</p> <p>3.b.1. L'utente richiede di visualizzare gli eventi in una data o intervallo di date</p> <p>3.b.2 Il sistema mostra a video gli eventi ordinati per data crescente inclusi nell'intervallo selezionato</p> <p>4. L'utente può selezionare un evento di interesse</p> <p>5. Il sistema mostra altri dettagli relativi a esso e la possibilità di gestirne la prenotazione</p>
Casi d'uso associati / flussi alternativi	Specializzazione UC1: Visualizzazione Eventi
Postcondizioni	/

Nome	UC1.2: Visualizzazione per luogo
Descrizione	Mostra tutti gli eventi disponibili di un luogo indicato
Attori	User
Obiettivi	Mostrare gli eventi disponibili svolti in luoghi scelti
Precondizioni	L'utente deve essere autenticato
Passi	<p>1. L'utente entra nella pagina di visualizzazione degli eventi</p> <p>2. Il sistema stampa a video tutti gli eventi disponibili in ordine crescente di data (dal più vicino al più lontano)</p> <p>3.a. Vista di eventi in una regione</p> <p>3.a.1. L'utente richiede la visualizzazione di eventi in una regione specifica</p> <p>3.a.2. Il sistema mostra a video tutti gli eventi ordinati per data crescente svolti nella regione richiesta</p> <p>3.b. Vista di eventi in una provincia</p> <p>3.b.1. L'utente richiede la visualizzazione di eventi in una provincia specifica</p>

	<p>3.b.2. Il sistema mostra a video tutti gli eventi ordinati per data crescente svolti nella provincia richiesta</p> <p>3.c. Vista di eventi in un comune</p> <p>3.c.1. L'utente richiede la visualizzazione di eventi in un comune specifico</p> <p>3.c.2. Il sistema mostra a video tutti gli eventi ordinati per data crescente svolti nel comune richiesto</p> <p>4. L'utente può selezionare un evento di interesse</p> <p>5. Il sistema mostra altri dettagli relativi a esso e la possibilità di gestirne la prenotazione</p>
Casi d'uso associati / flussi alternativi	Specializzazione UC1: Visualizzazione Eventi
Postcondizioni	/

Nome	UC4: Prenotazione Evento
Descrizione	L'utente può prenotare il posto per un evento o rimuovere una prenotazione da un evento già prenotato.
Attori	User
Obiettivi	Prenotare un posto per un evento da parte di un utente registrato
Precondizioni	<ul style="list-style-type: none"> • L'utente è loggato al sito • L'evento ha posti disponibili • L'evento avviene in una data futura • L'utente non è già prenotato all'evento in caso di prenotazione. • L'utente è già prenotato all'evento in caso di disdetta della prenotazione. • L'utente si trova nella pagina di visualizzazione degli eventi
Passi	1.a. Prenotazione a un evento:

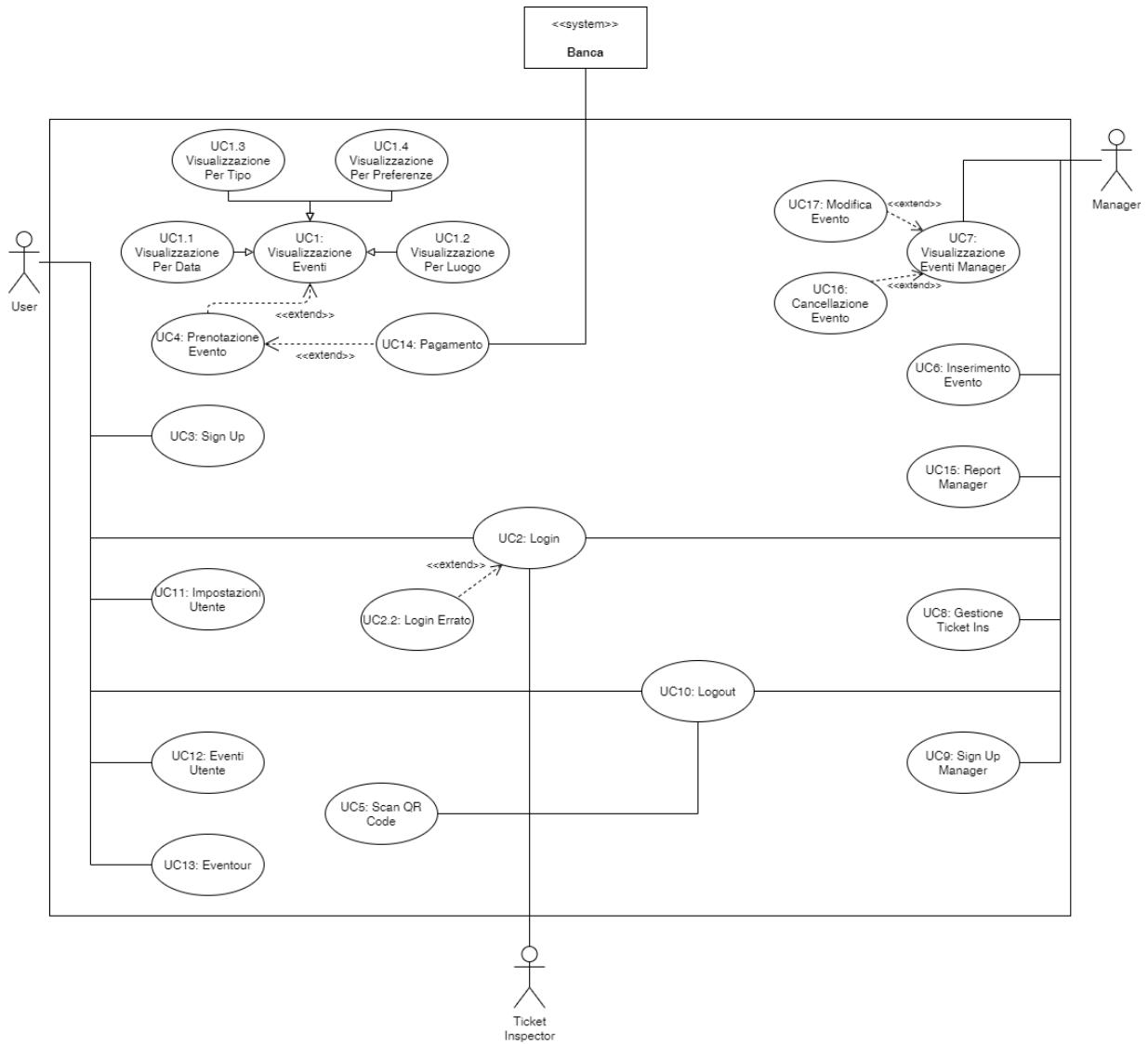
	<p>1.a.1. L'utente richiede la prenotazione all'evento nella pagina di visualizzazione dell'evento</p> <p>1.a.2. Il sistema controlla le condizioni richieste</p> <p>1.a.3. Il sistema, dopo aver verificato la disponibilità del posto o posti richiesto/i, prenota il posto per l'evento e aggiorna il database</p> <p>1.a.4. Il sistema conferma la prenotazione del posto richiesto</p> <p>1.b. Disdetta di una prenotazione</p> <p>1.b.1. L'utente richiede la disdetta della prenotazione all'evento nella pagina di visualizzazione dell'evento</p> <p>1.b.2. Il sistema disdice la prenotazione del posto o posti per l'evento e aggiorna il database</p> <p>1.b.3. Il sistema conferma la disdetta della prenotazione del posto richiesto</p>
Casi d'uso associati / flussi alternativi	/
Postcondizioni	L'utente è prenotato all'evento e ha un posto riservato nel caso di prenotazione

Nome	UC8: Gestione Ticket Inspector
Descrizione	Il sistema permette al Manager la gestione dei propri Ticket Inspector.
Attori	Manager
Obiettivi	Organizzare i Ticket Inspector di un Manager
Precondizioni	Il Manager deve essere autenticato
Passi	<ol style="list-style-type: none"> 1. Il Manager richiede l'accesso alla pagina di visualizzazione di un evento dettagliato 2. Il sistema mostra la pagina dove poter visualizzare i Ticket Inspector di un evento. 3.a. Inserimento di un ticket inspector per l'evento

	<p>3.a.1. L'utente richiede l'inserimento di un nuovo ticket inspector</p> <p>3.a.2. Il sistema richiede i dati del nuovo ticket inspector</p> <p>3.a.3. L'utente compila i dati richiesti e chiede l'inserimento</p> <p>3.a.4. Il sistema registra il nuovo ticket inspector salvando le informazioni nel database e conferma all'utente il nuovo ticket inspector</p> <p>3.b. Cancellazione di un ticket inspector per l'evento</p> <p>3.b.1. L'utente richiede la cancellazione di un ticket inspector</p> <p>3.b.2. Il sistema rimuove il ticket inspector dalla lista dei ticket inspector associati all'evento</p>
Casi d'uso associati / flussi alternativi	/
Postcondizioni	/

Use Case diagram

Nel diagramma seguente si vedono i casi d'uso aggiunti ai casi ad alta priorità effettuati nella iterazione precedente, quali le nuove tipologie di visualizzazione, la comparsa di possibilità di visualizzare e modificare alcune informazioni dell'utente, la possibilità di vedere gli eventi prenotati e la proposta di un tour di eventi, il pagamento all'atto della prenotazione e la generazione di report per il manager.

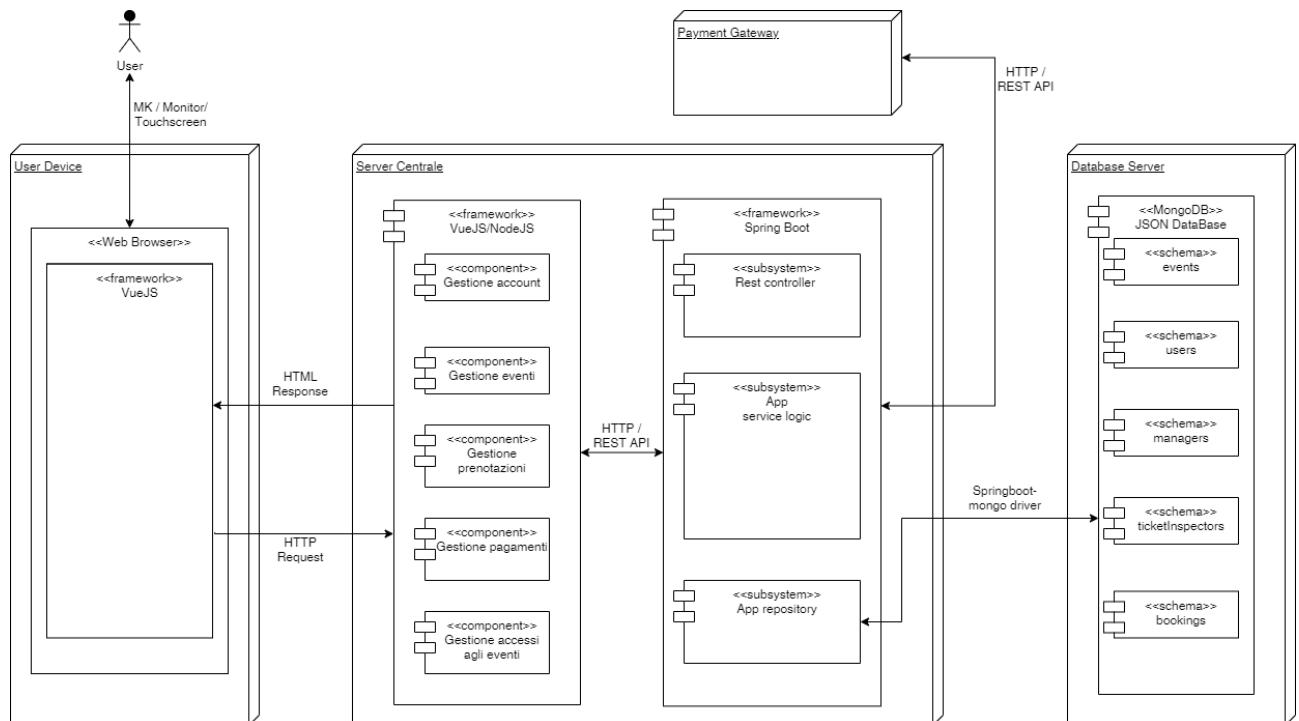


Modello 4.1 Use Case diagram

2. Architettura

Deployment diagram

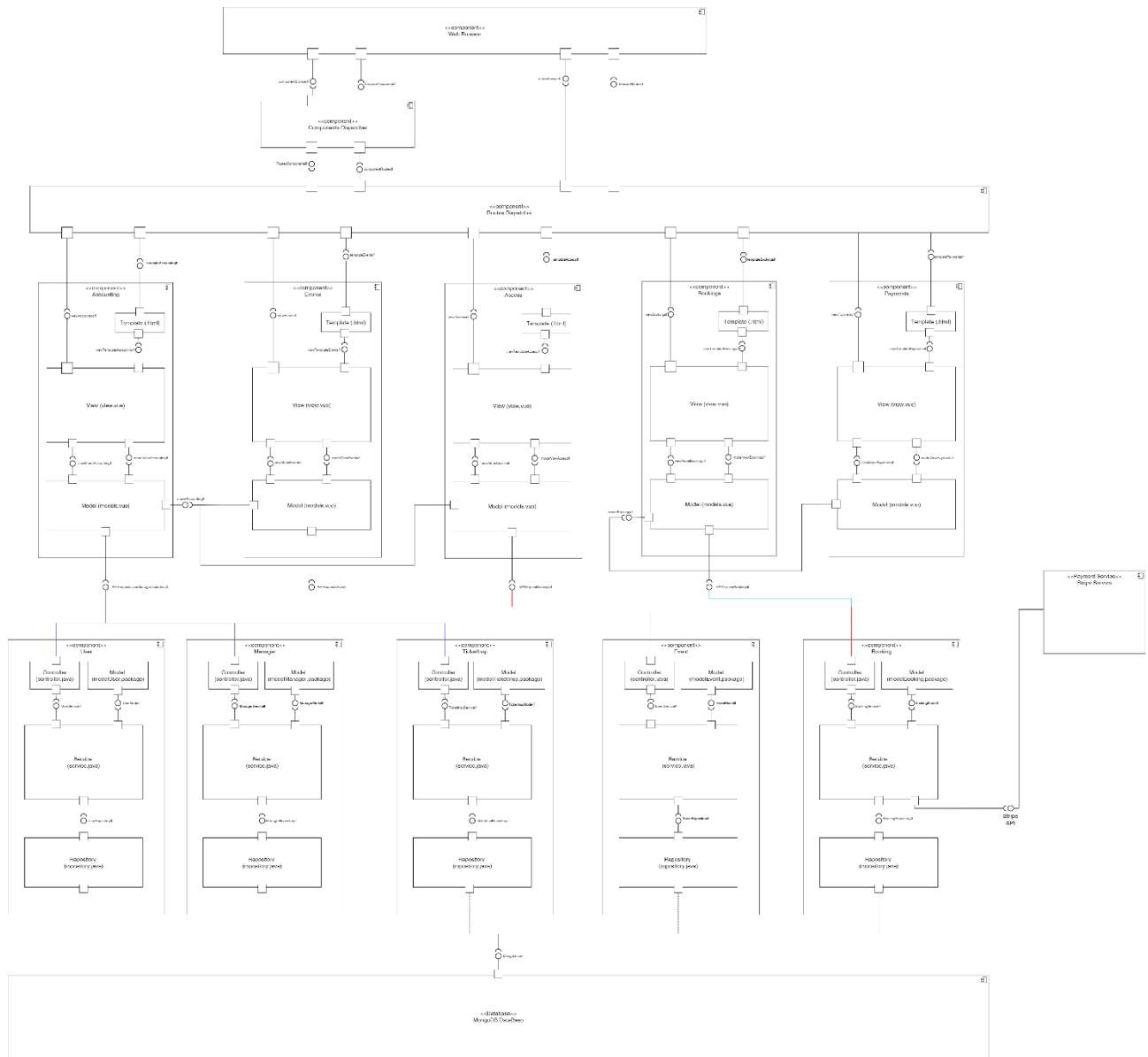
Per quanto concerne il deployment diagram, l'introduzione di nuove funzionalità ha richiesto l'aggiunta di un componente esterno alla piattaforma e alla API che permetta la comunicazione con i principali sistemi di pagamento. È stato quindi selezionato Stripe API che fornisce numerose API per la gestione e il pagamento online. Inoltre, si sono aggiunti nuovi componenti interni alla parte web visuale del sistema per la gestione di pagamenti.



Modello 4.2 Deployment Diagram

Component diagram

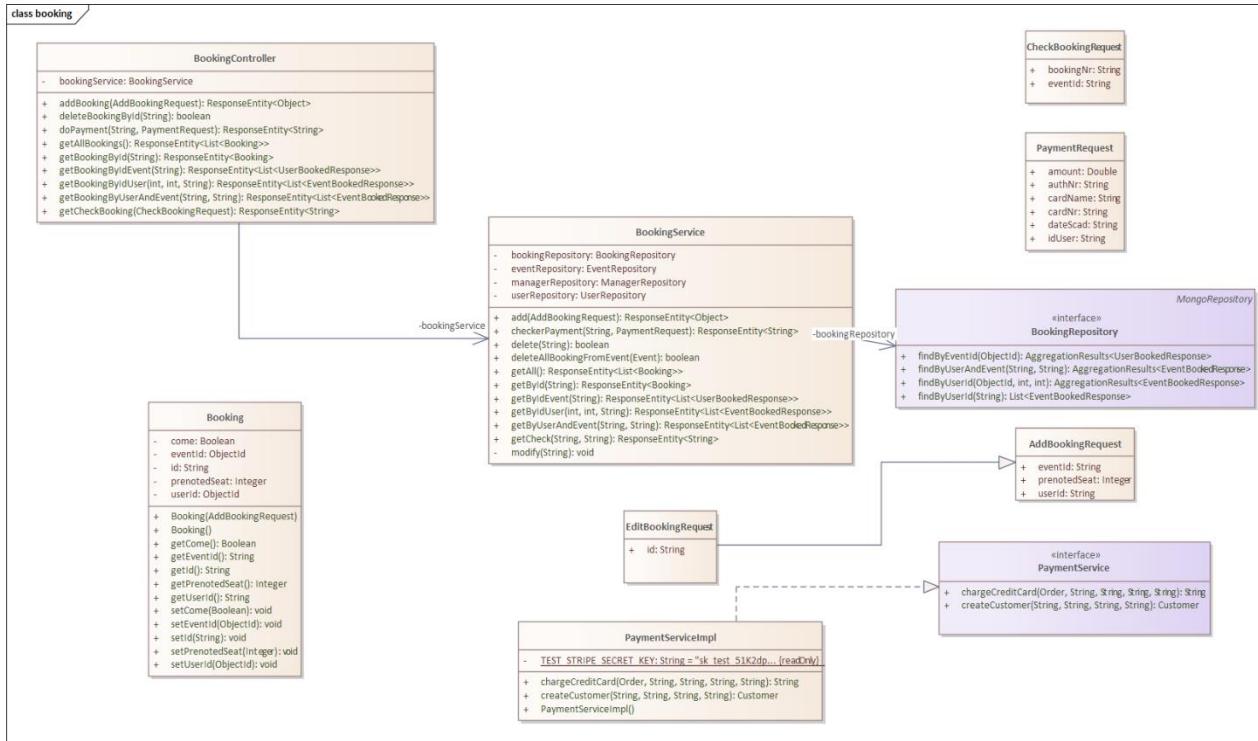
La modifica del deployment diagram si ripercuote strettamente nel component diagram, in cui le iterazioni tra i vari model in Vue si incrementano e in cui viene aggiunto un nuovo macrocomponente per la gestione dei pagamenti. La parte di gestione dei pagamenti a livello di API viene elaborata nel componente di gestione delle prenotazioni, che comunica tramite Stripe API con uno Stripe Service.



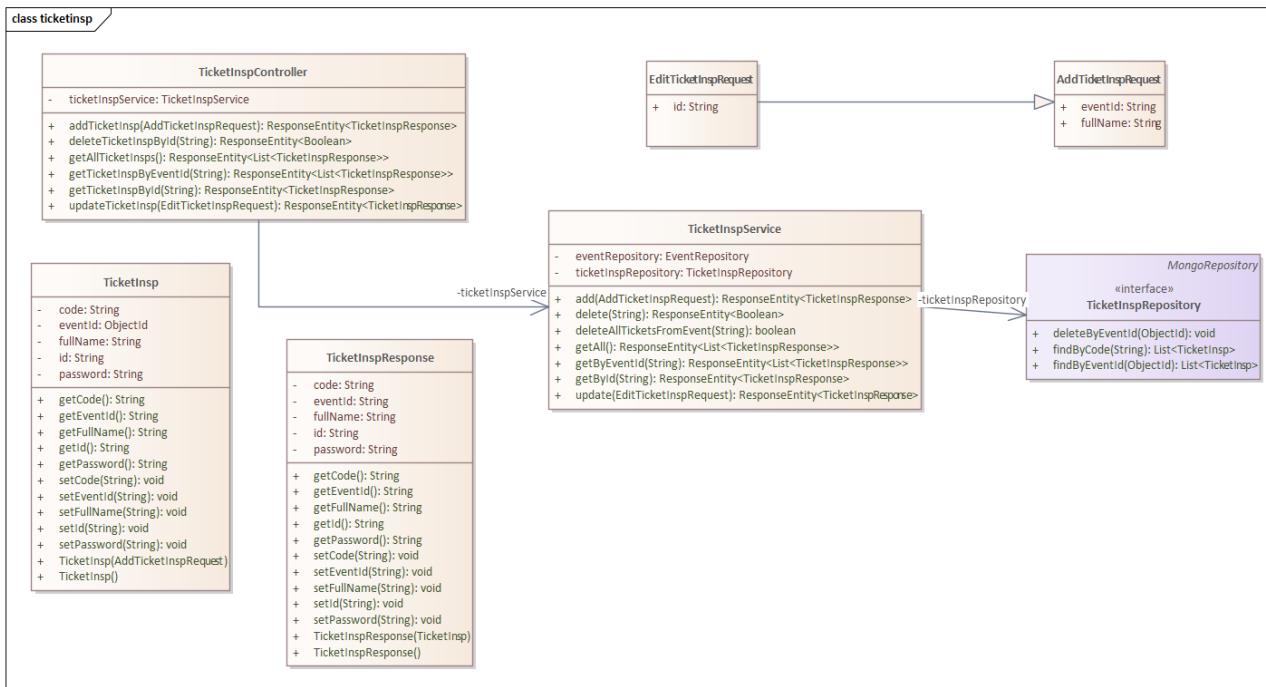
Modello 4.3 Component Diagram

3. Class diagram

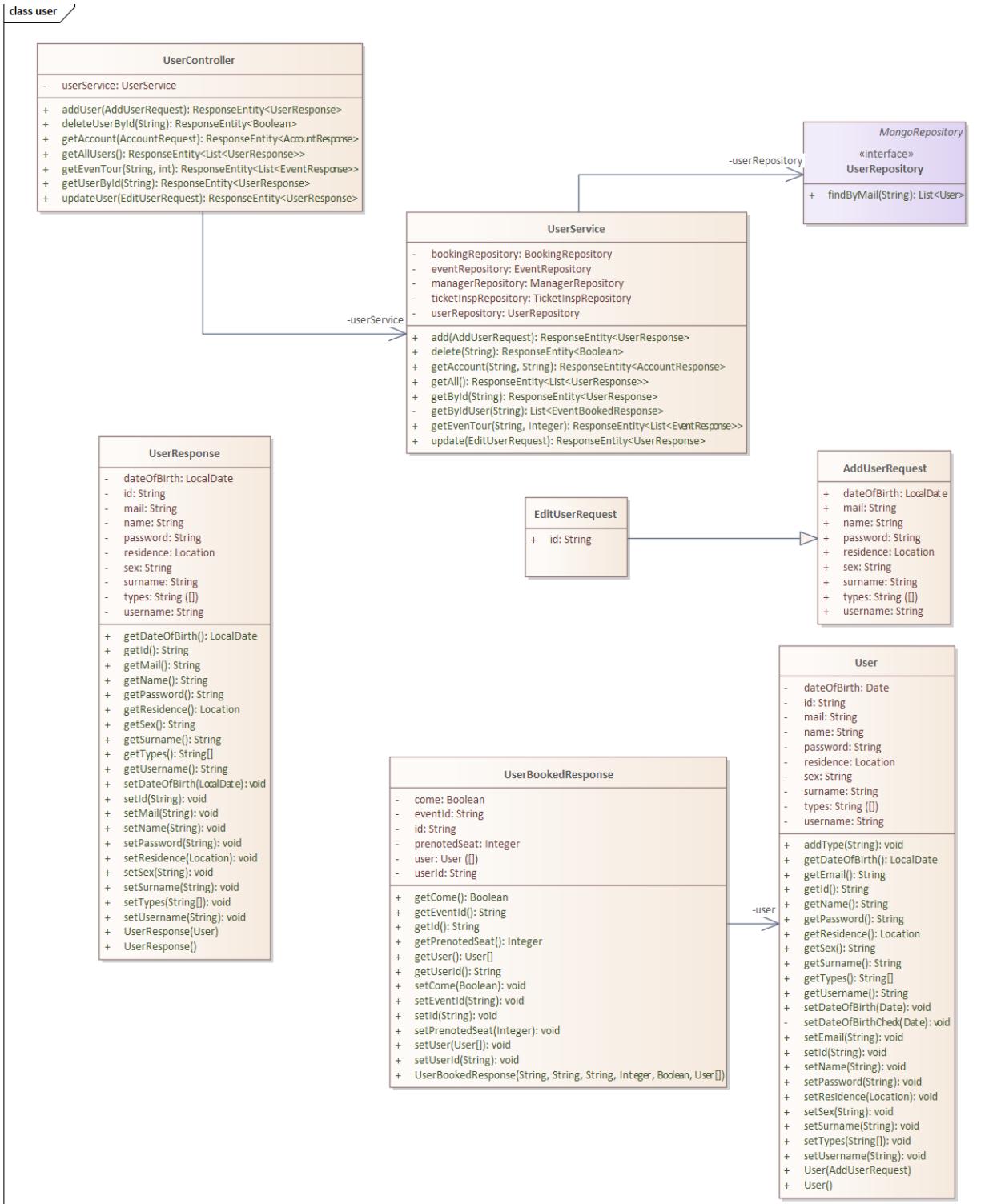
Di seguito sono mostrati i class diagram relativi alle classi utilizzate per l'implementazione dell'iterazione corrente.



Modello 4.4 Booking Class Diagram



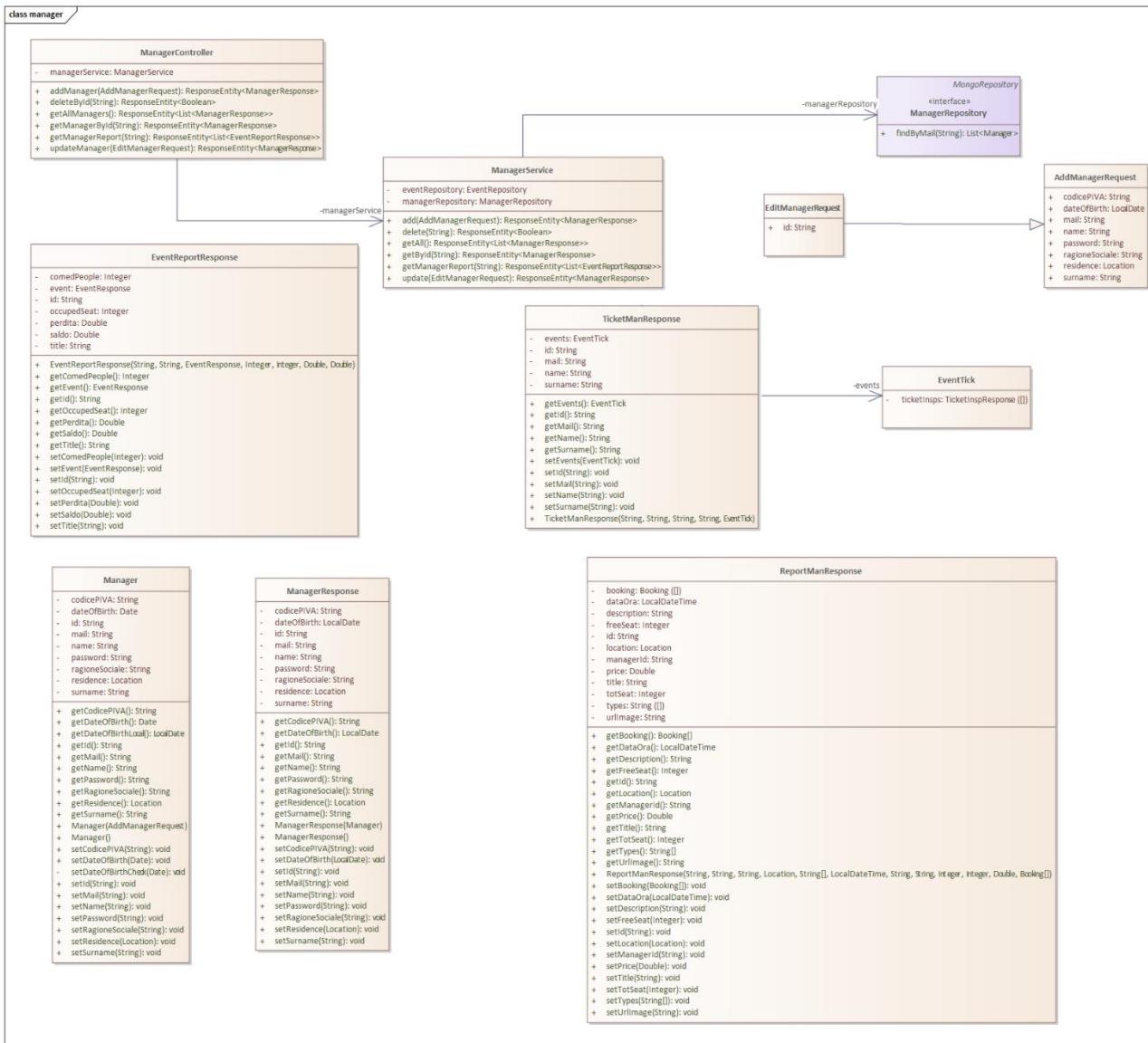
Modello 4.5 Ticket Inspector Class Diagram



Modello 4.6 User Class Diagram



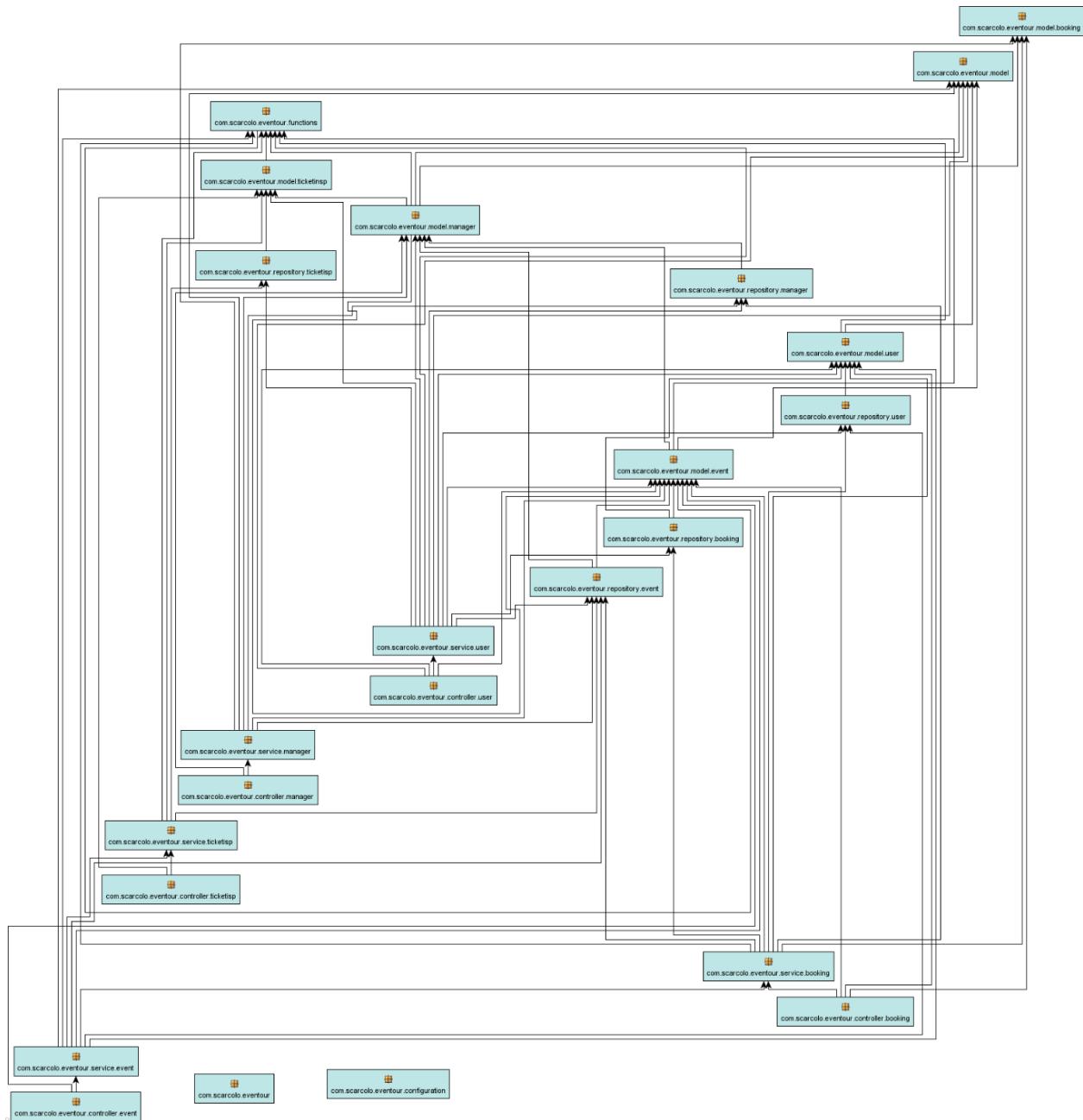
Modello 4.7 Event Class Diagram



Modello 4.8 Manager Class Diagram

A causa dell'aumento di complessità dei casi d'uso introdotti, anche le classi hanno avuto un incremento di metodi e funzioni introdotte.

Si riporta di seguito il package diagram, dove si nota l'aumento di complessità.



Modello 4.9 Package Diagram

4. Novità implementative introdotte

Per quanto riguarda le novità implementative introdotte, sono state principalmente aggiunte tre novità implementative:

- MD5: dalla iterazione corrente, seppur nel database le password sono ancora salvate in chiaro per una questione di testing delle funzionalità, lo scambio di password tra API gateway e Vue sono effettuate facendo uso dell'hash MD5, che è comunque una prima scelta per una sicurezza maggiore sui dati degli utenti.
- Invio di mail: dalla iterazione corrente, è stato introdotto l'uso della libreria javax.mail lato API che permette l'invio di email a un utente in maniera semplice. Essa è stata usata importandone la dipendenza nel POM.
- Pagamenti: la novità maggiore introdotta nella iterazione corrente è l'utilizzo di una API per il pagamento di eventi e, dalla iterazione successiva, quote di iscrizione. La API utilizzata è Stripe^[7], che permette lo sfruttamento dei maggiori metodi di pagamento internazionali e si occupa di mantenere la sicurezza dei dati in automatico. L'importazione di essa è stata fatta a livello implementativo ma le funzionalità sono state commentate per evitare pagamenti indesiderati, visto l'uso di dati mock. Anche nell'applicativo web si è fatto uso di un componente pre-realizzato da altri sviluppatori^[8].

5. Algoritmo evenTour

Lo scopo della versione corrente dell'algoritmo progettato è quella di trovare ancora un numero di eventi definito dalla invocazione della API, che siano disponibili e che siano nella regione di residenza dell'utente, ma stavolta andando a includere anche gli eventi il cui tipo di evento sia simile a quello preferito dall'utente.

La versione ottimizzata è riportata nella pagina successiva, mentre quella non ottimizzata era realizzata sfruttando un doppio ciclo degli eventi:

Partendo dallo pseudocodice della iterazione precedente, si è deciso in prima istanza di aggiungere successivamente al primo ciclo di tutti gli eventi, un secondo ciclo identico ma in cui venivano aggiunti alla soluzione anche gli eventi con tipo simile (un tipo di evento è simile a un altro se la sua tipologia ha, nella tipizzazione padre, lo stesso valore. Ad esempio, i tipi 1.2.4 e 1.2.2 sono simili, così come sono simili i tipi 3.2 e 3.5, mentre i tipi 1.1.2 e 1.2.1 non lo sono).

⁷ <https://stripe.com/docs/development>

⁸ <https://madewithvuejs.com/interactive-paycard>

Dopo una analisi si è ottimizzato l'algoritmo, per evitare il doppio ciclo, controllando quindi i tipi nel medesimo ciclo, aggiungendoli a due liste separate che vengono poi unite per una parte solo nel caso in cui gli eventi trovati come soluzione siano minori del numero richiesto. In questo modo è stato possibile anche ordinarli per data e ora prima di restituirli.

Nel caso in cui neanche l'unione degli eventi del medesimo tipo di quello preferito dall'utente con quelli con tipologia simile siano abbastanza, l'algoritmo restituisce un errore di soluzione non esistente.

Si riporta lo pseudocodice dell'algoritmo nella pagina successiva.

```

1. //id: id dell'utente, N: numero eventi richiesti da un utente
2. function EvenTour(id, N)
3. {
4.     datiUtente <- ricercaUtenteDaID(id);
5.     listaEventi <- ricercaEventiPerRegioneConPostiDisponibiliOrdinatiPerData
6.                                         (datiUtente.regione);
7.
8.     s <- Lista vuota //Soluzione da restituire
9.     c <- 0 //Numero eventi inseriti nella soluzione
10.    cSub <- 0 //Numero eventi con tipo simile inseriti nella possibile soluzione
11.
12.    SE (bookingData ha eventi) ALLORA
13.    {
14.        PER OGNI eventoPrenotato IN bookingData
15.        {
16.            rimuoviEventoGiàPrenotato(eventoPrenotato);
17.            rimuoviEventoConDateUguali(eventoPrenotato);
18.        }
19.    }
20.
21.    PER OGNI evento IN listaEventi
22.    {
23.        sel <- false //indica se l'evento è già stato inserito visto il controllo dei tipi
24.        selSub <- false //indica se l'evento con tipo simile è stato inserito
25.        SE (c = N) ALLORA
26.        {
27.            RETURN s
28.        }
29.        PER OGNI tipo IN evento.tipi
30.        {
31.            PER OGNI tipoUtente IN datiUtente.tipi
32.            {
33.                SE (NOT sel AND (s VUOTO OR NOT dateUguali(evento, s[s.size - 1])))
34.                {
35.                    SE (tipo = tipoUtente)
36.                    {
37.                        SE (sSub NOT vuoto && sSub[sSub.size - 1].id = evento.id)
38.                        {
39.                            sSub <- sSub - {sSub[sSub.size - 1]}
40.                        }
41.                        s <- s U evento
42.                        sel <- true
43.                        c <- c + 1
44.                    }
45.                    ALTRIMENTI SE (!selSub && cSub < N && tipiSimili(tipo, tipoUtente))
46.                    {
47.                        sSub <- sSub U evento
48.                        selSub <- true
49.                        cSub <- cSub + 1
50.                    }
51.                }
52.            }
53.        }
54.    }
55.    SE s.size+sSub.size >= N
56.    {
57.        PER OGNI c DA 0 A (N-s.size)
58.        s <- s U sSub[h];
59.        s <- ordinamentoPerDataOra(s)
60.        RETURN s
61.    }
62.    return ERROR;
63. }

```

6. Testing interfaccia grafica

Analisi statica

Per quanto riguarda la parte di analisi statica, è stato utilizzato nuovamente il componente di EsLint che permette l'analisi del codice. Come espresso precedentemente, sono state riportati tutti i problemi legati alla struttura del codice sia per la parte grafica sia per la parte legata ai dati. Gli errori e i warning espressi nell'iterazione precedente permangono anche in questa iterazione.

Analisi dinamica

```
'Login User EvenTour' : function(browser) {
    browser
        .url('http://localhost:8080')
        .click('button[id=login]')
        .assert.urlEquals('http://localhost:8080/login')
        .setValue('input[id=username]', 'Colombano72@hotmail.com')
        .setValue('input[id=password]', 'yelukare')
        .click('button[id=btnlogin]')
        .waitForElementVisible('button[id=logout]')
        .assert.urlEquals('http://localhost:8080/')
        .waitForElementVisible('button[id=logout]')
        .click('button[id=btnEventour]')
        .assert.urlEquals('http://localhost:8080/eventour/61a0a933bce0e98fbb2d999d/
numevents/5')
        .waitForElementVisible('button[id=logout]')
        .waitForElementVisible('div[id=timeline]')
        .waitForElementVisible('div[id=cardEvenTour]')
        .end();
},
'Login User EvenTour Payment' : function(browser) {
    browser
        .url('http://localhost:8080')
        .click('button[id=login]')
        .assert.urlEquals('http://localhost:8080/login')
        .setValue('input[id=username]', 'Colombano72@hotmail.com')
        .setValue('input[id=password]', 'yelukare')
        .click('button[id=btnlogin]')
        .waitForElementVisible('button[id=logout]')
        .assert.urlEquals('http://localhost:8080/')
        .waitForElementVisible('button[id=logout]')
        .click('button[id=btnEventour]')
        .assert.urlEquals('http://localhost:8080/eventour/61a0a933bce0e98fbb2d999d/
numevents/5')
        .waitForElementVisible('button[id=logout]')
        .waitForElementVisible('div[id=timeline]')
        .waitForElementVisible('div[id=cardEvenTour]')
        .click('button[id=btnPayTour]')
        .waitForElementVisible('input[id=v-card-number]')
        .setValue('input[id=v-card-number]', '4000003800000008')
        .setValue('input[id=v-card-name]', 'Colombi Simone')
        .setValue('select[id=v-card-month]', '02')
        .setValue('select[id=v-card-year]', '22')
        .setValue('input[id=v-card-cvv]', '222')
        .click('button[id=btnPay]')
        .assert.urlEquals('http://localhost:8080/eventour/61a0a933bce0e98fbb2d999d/
numevents/5')
        .end();
},
'Login User My Event' : function(browser) {
    browser
```

```

.url('http://localhost:8080')
.click('button[id=login]')
.assert.urlEquals('http://localhost:8080/login')
.setValue('input[id=username]', 'Colombano72@hotmail.com')
.setValue('input[id=password]', 'yelukare')
.click('button[id=btnlogin]')
.waitForElementVisible('button[id=logout]')
.assert.urlEquals('http://localhost:8080/')
.waitForElementVisible('button[id=logout]')
.click('button[id=btnMyevents]')
.assert.urlEquals('http://localhost:8080/events/user/61a0a933bce0e98fbb2d99
9d')
.waitForElementVisible('button[id=logout]')
.waitForElementVisible('div[id=qrcode]')
.end();
},
'Login User Settings' : function(browser) {
  browser
.url('http://localhost:8080')
.click('button[id=login]')
.assert.urlEquals('http://localhost:8080/login')
.setValue('input[id=username]', 'Colombano72@hotmail.com')
.setValue('input[id=password]', 'yelukare')
.click('button[id=btnlogin]')
.waitForElementVisible('button[id=logout]')
.assert.urlEquals('http://localhost:8080/')
.waitForElementVisible('button[id=btnSettings]')
.click('button[id=btnSettings]')
.assert.urlEquals('http://localhost:8080/settings/61a0a933bce0e98fbb2d999d'
)
.waitForElementVisible('button[id=logout]')
.waitForElementVisible('button[id=SaveSettings]')
.click('button[id=SaveSettings]')
.assert.urlEquals('http://localhost:8080/')
.end();
}

```

Codice 4.2 Test classe User relativi alla iterazione 2

Anche per questa iterazione si è usato Nightwatch per l'esecuzione dei test di analisi dinamica del codice. Si riportano in seguito alcuni dei test effettuati.

```
'Login Manager Report' : function(browser) {
    browser
        .url('http://localhost:8080')
        .click('button[id=login]')
        .assert.urlEquals('http://localhost:8080/login')
        .setValue('input[id=username]', 'Domenica.Acquadro@hotmail.com')
        .assert.value('input[id=username]', 'Domenica.Acquadro@hotmail.com')
        .setValue('input[id=password]', 'jesobuje')
        .click('button[id=btnlogin]')
        .waitForElementVisible('button[id=logout]')
        .assert.urlEquals('http://localhost:8080/homemanager')
        .waitForElementVisible('button[id=logout]')
        .click('button[id=btnReportManager]')
        .assert.urlEquals('http://localhost:8080/manager/61a0a0eeb5f9b12d06e95237/r
eport')
        .waitForElementVisible('div[id=divReportManager]')
        .end();
}
```

Codice 4.3 Nuovo test relativo alla classe Manager

I test appena presentati sono aggiunte e/o modifiche rispetto ai test attuati nell’iterazione precedente. Di seguito gli output dell’esecuzione dei test:

```
<?xml version="1.0" encoding="UTF-8" ?>
<testsuites errors="0"
             failures="0"
             tests="2">

    <testsuite name="ManagerTest"
               errors="0" failures="0" hostname="" id="" package="ManagerTest" skipped="0"
               tests="2" time="48.79" timestamp="">

        <testcase name="Login and Logout Manager" classname="ManagerTest" time="25.76"
                  assertions="7">

            </testcase>

            <testcase name="Login Manager Report" classname="ManagerTest" time="23.03"
                      assertions="7">

                </testcase>
        </testsuite>
    </testsuites>
```

Codice 4.4 Output Manager Test

```

<?xml version="1.0" encoding="UTF-8" ?>
<testsuites errors="0"
             failures="0"
             tests="8">

    <testsuite name="UserTest"
               errors="0" failures="0" hostname="" id="" package="UserTest" skipped="0"
               tests="8" time="180.6" timestamp="">

        <testcase name="Home Page Loading" classname="UserTest" time="13.76"
assertions="3">

            </testcase>

        <testcase name="From Home to Login" classname="UserTest" time="21.47"
assertions="2">

            </testcase>

        <testcase name="Login and Logout User" classname="UserTest" time="26.22"
assertions="6">

            </testcase>

        <testcase name="Login and Try Access Manager Denied" classname="UserTest"
time="26.40" assertions="6">

            </testcase>

        <testcase name="Login User EvenTour" classname="UserTest" time="23.54"
assertions="8">

            </testcase>

        <testcase name="Login User EvenTour Payment" classname="UserTest" time="20.51"
assertions="10">

            </testcase>

        <testcase name="Login User My Event" classname="UserTest" time="21.59"
assertions="7">

            </testcase>

        <testcase name="Login User Settings" classname="UserTest" time="27.11"
assertions="8">

```

7. Testing API

Analisi statica

Per quanto riguarda l'analisi statica effettuata sul server per l'esposizione delle API, dopo aver generato una copia di test in MongoDB Atlas e aver modificato la stringa di connessione nell'API, si sono riscontrati circa 30 bugs segnalati da SpotBugs e varie violazioni segnalate grazie a PMD.

Per quanto riguarda Spotbugs, i bug segnalati sono stati risolti partendo da quelli più critici per finire a quelli meno critici. Sono rimasti due bugs non critici di cui uno segnala un possibile null-pointer che viene controllato a priori e una scrittura su metodo statico.

Per quanto riguarda PMD, le violazioni di livello Blocker e Critical (bandiera rossa e azzurra) sono stati risolte, mentre quelli di livello inferiore sono stati controllate e analizzate senza correggerle.

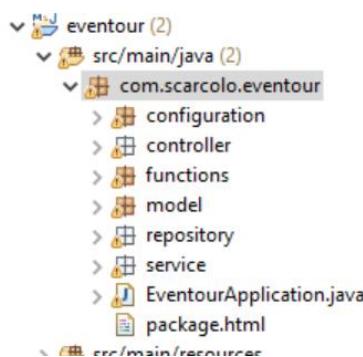


Figura 4.1 Package progetto in Eclipse con visualizzazione dei bug segnalati da spotbugs e delle violazioni di livello alto mostrate

Il report di JArchitect evidenzia inoltre il grafo delle dipendenze mostrato di seguito:

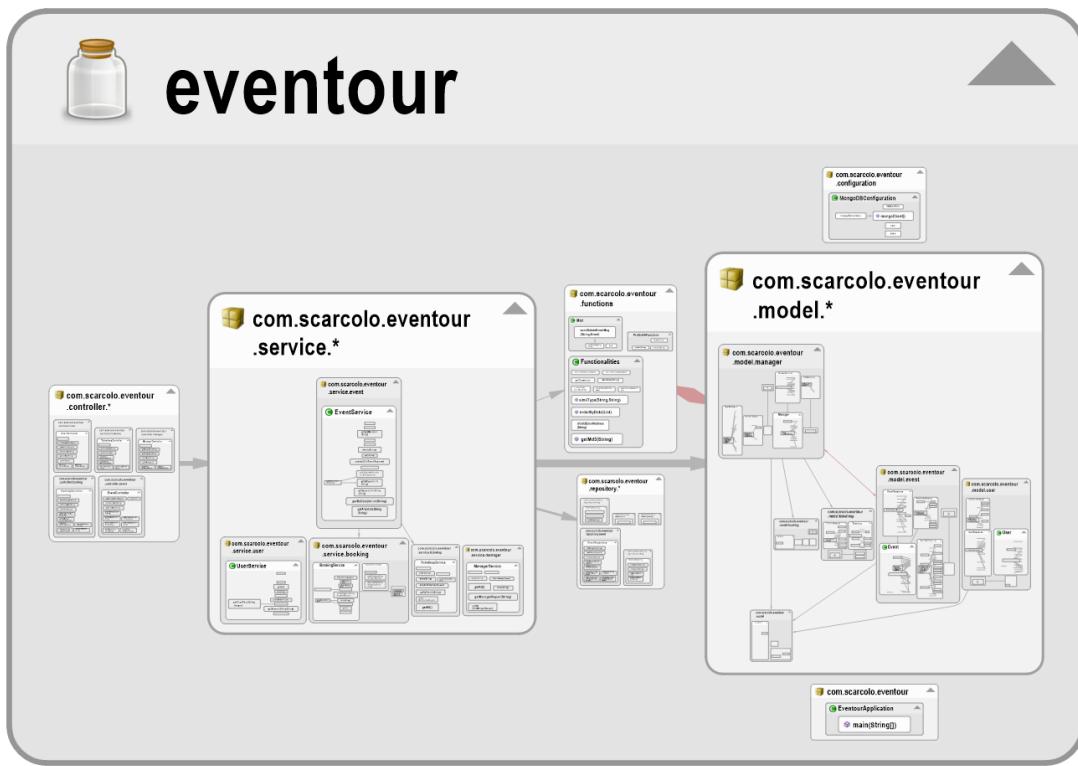


Figura 4.2 Modello report JArchitect

Analisi dinamica

Tramite JUnit si sono svolti numerosi test, come visibile nella figura di seguito. Purtroppo, i test non possono essere esaustivi di tutte le funzionalità presenti nella API in quanto tutti i metodi get e set non vengono usati spesso a causa del solo test dei costruttori richiamati dalle chiamate API effettuate. Si nota tuttavia che i test hanno coperto tutte le chiamate realizzate nella iterazione corrente.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
eventour	87,2 %	8.562	1.257	9.819
src/test/java	100,0 %	4.225	2	4.227
src/main/java	77,6 %	4.337	1.255	5.592
com.scarcolo.eventour.configuration	100,0 %	52	0	52
com.scarcolo.eventour.controller.booking	100,0 %	54	0	54
com.scarcolo.eventour.controller.event	100,0 %	68	0	68
com.scarcolo.eventour.controller.manager	100,0 %	32	0	32
com.scarcolo.eventour.controller.ticketisp	100,0 %	32	0	32
com.scarcolo.eventour.controller.user	100,0 %	42	0	42
com.scarcolo.eventour.service.ticketisp	92,8 %	256	20	276
com.scarcolo.eventour.service.manager	91,5 %	333	31	364
com.scarcolo.eventour.model	90,6 %	77	8	85
com.scarcolo.eventour.model.ticketisp	88,7 %	118	15	133
com.scarcolo.eventour.service.user	87,0 %	614	92	706
com.scarcolo.eventour.model.booking	82,4 %	70	15	85
com.scarcolo.eventour.service.booking	82,3 %	527	113	640
com.scarcolo.eventour.functions	80,8 %	362	86	448
com.scarcolo.eventour.service.event	69,6 %	741	324	1.065
com.scarcolo.eventour.model.user	67,9 %	266	126	392
com.scarcolo.eventour.model.manager	63,1 %	356	208	564
com.scarcolo.eventour.model.event	61,2 %	334	212	546
com.scarcolo.eventour	37,5 %	3	5	8

Figura 4.3 Copertura test JUnit

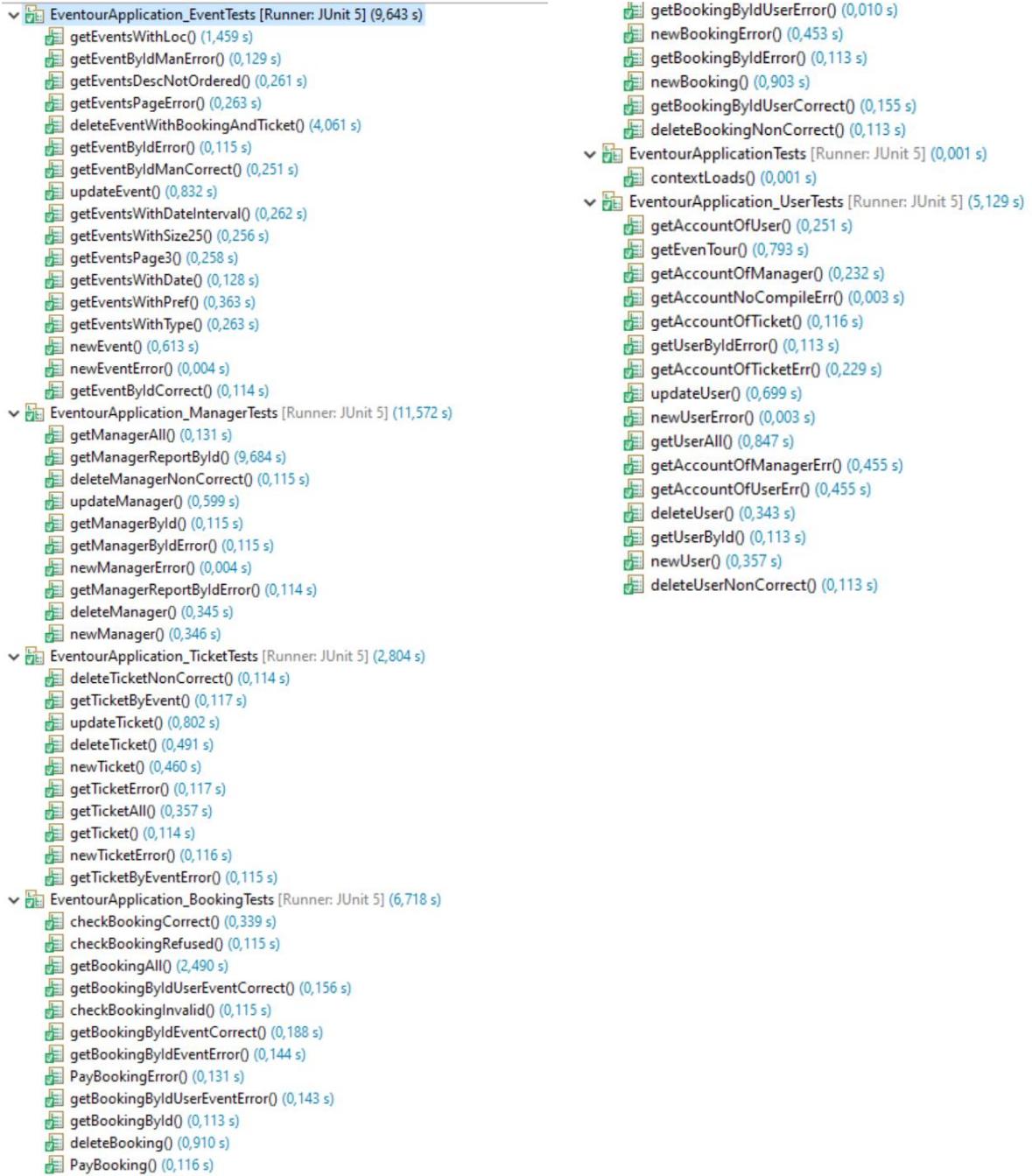


Figura 4.4 Test effettuati e passati

Iterazione 3

L'iterazione 3 ha visto la comparsa di un nuovo ruolo all'interno della piattaforma. Ciò ha comportato l'inserimento di tutta la gestione delle richieste di iscrizione dei manager da parte degli amministratori, con una gestione semplificata dei manager. Si è inoltre introdotta la possibilità di rilasciare un riscontro agli eventi a cui sono stati da parte degli utenti, il rinnovo dell'iscrizione e il pagamento della quota d'iscrizione da parte del manager e la newsletter per gli utenti.

1. Use Cases

Descrizione testuale

Sono stati definiti nuovi casi d'uso:

Nome	UC1521: Report
Descrizione	Visualizzazione di report
Attori	/
Obiettivi	Visualizzazione report
Precondizioni	L'utente che lo richiede deve essere loggato
Passi	<ol style="list-style-type: none">1. L'utente loggato entra nella pagina dei report2. Il sistema mostra a video l'elenco di tutti i report
Casi d'uso associati / flussi alternativi	<<include>> UC2: Login Specializzazione UC15: Report Manager, UC21: Report Admin
Postcondizioni	/

Nome	UC18: Feedback
Descrizione	L'utente può fornire una valutazione relativa all'evento a cui ha partecipato

Attori	User
Obiettivi	Fornire dei feedback che possano essere usati per la valutazione del Manager
Precondizioni	l'utente deve essere loggato
Passi	<ol style="list-style-type: none"> 1. L'utente richiede di accedere alla pagina dei feedback 2. Il sistema riporta l'utente nella pagina dei feedback 3. Il sistema mostra tutti gli eventi passati per cui l'utente si è prenotato e presentato e nei quali è possibile fornire un feedback 4. L'utente sceglie uno degli eventi per cui vuole fornire il feedback e modifica o inserisce una valutazione da 1 a 5 del gradimento dell'evento 5. Il sistema salva il feedback dato dall'utente 6. Il sistema aggiorna i dati relativi al feedback dato nel database e aggiorna la pagina dei feedback
Casi d'uso associati / flussi alternativi	Flusso alternativo: L'uscita dalla pagina senza aver modificato o inserito feedback comporta la mancata modifica di recensioni.
Postcondizioni	Avere un riscontro da parte dell'utente relativo all'evento svolto

Nome	UC19: Accept Request
Descrizione	Il sistema permette all'Admin di autorizzare i nuovi manager
Attori	Admin
Obiettivi	Autorizzare la registrazione di un manager da parte dell'Admin
Precondizioni	L'Admin deve essere autenticato
Passi	<ol style="list-style-type: none"> 1. L'admin richiede di accedere alla pagina di gestione delle richieste 2. Il sistema mostra la lista dei Manager che richiedono di poter svolgere il loro ruolo nella piattaforma 3.a. Accettazione di una richiesta

	<p>3.a.1. L'Admin autorizza un manager che rispetta i requisiti</p> <p>3.a.2. Il sistema inserisce il manager come autorizzato a operare nella piattaforma</p> <p>3.a.3. Il sistema notifica ai manager via mail che la loro registrazione è stata autorizzata</p> <p>3.b. Rifiuto di una richiesta</p> <p>3.b.1. L'Admin elimina la richiesta di un manager che non rispetta i requisiti</p> <p>3.b.2. Il sistema notifica via mail che la richiesta del manager di registrarsi non è stata autorizzata</p> <p>3.b.3. Il sistema elimina la registrazione del manager dalla piattaforma</p>
Casi d'uso associati / flussi alternativi	<>extend>> UC20: Gestione manager
Postcondizioni	I manager autorizzati sono operativi sulla piattaforma, se autorizzati. Se sono state rifiutate delle richieste, esse e i relativi manager saranno cancellati

Nome	UC20: Gestione manager
Descrizione	L'Admin può gestire i manager attribuendo un malus o visualizzando i dati di essi
Attori	Admin
Obiettivi	Gestione dei Manager da parte dell'Admin
Precondizioni	L'Admin deve essere autenticato
Passi	<ol style="list-style-type: none"> 1. L'Admin richiede di entrare nella gestione dei manager 2. Il sistema mostra la pagina di gestione dei manager 3.a. L'admin visualizza i manager e i dati collegati 3.b. Attribuzione di un malus 3.b.1. L'admin attribuisce un malus a un manager

	<p>3.b.2. Il sistema attribuisce il malus</p> <p>3.b.3. il sistema riaggiorna la pagina di gestione dei manager</p>
Casi d'uso associati / flussi alternativi	/
Postcondizioni	Nel caso di malus attribuito a un manager, il manager sarà costretto al primo accesso a dover ripagare nuovamente il rinnovo dell'iscrizione come penalità.

Nome	UC21: Report Admin
Descrizione	Stampa di report relativi all'attività dei manager.
Attori	Admin
Obiettivi	Stampa report dei manager presenti
Precondizioni	L'admin deve essere autenticato
Passi	<ol style="list-style-type: none"> 1. L'admin entra nella pagina dei report 2. Il sistema mostra a video l'elenco di tutti i report relativi ai manager presenti sulla piattaforma
Casi d'uso associati / flussi alternativi	Generalizzazione UC1521: Report
Postcondizioni	/

Nome	UC22: Impostazioni manager
Descrizione	Il Manager visualizza e modifica eventualmente le proprie impostazioni
Attori	Manager
Obiettivi	Visualizzazione e modifica delle impostazioni di un Manager
Precondizioni	Il Manager deve essere loggato
Passi	<ol style="list-style-type: none"> 1. Il Manager richiede l'accesso alla pagina delle proprie impostazioni 2. Il sistema visualizza le impostazioni del Manager

	<p>3.a. Il sistema permette l'uscita dalle impostazioni</p> <p>3.b. modifica di impostazioni</p> <p>3.b.1. Il sistema mostra i campi disponibili per la modifica, precompilati col valore precedente</p> <p>3.b.2. Il Manager modifica i dati di interesse</p> <p>3.b.3. Il sistema controlla le modifiche fatte</p> <p>3.b.4.a. Se le modifiche sono corrette, il sistema notifica l'avvenuta modifica</p> <p>3.b.4.b. Se le modifiche sono incorrette, il sistema mostra l'errore e richiede la correzione della modifica</p> <p>3.b.5. Il sistema riporta il manager nella pagina di visualizzazione delle impostazioni</p>
Casi d'uso associati / flussi alternativi	/
Postcondizioni	Le informazioni sono state modificate (se richiesto dal Manager)

Nome	UC23: Rinnovo iscrizione
Descrizione	Il Manager effettua se necessario il rinnovo dell'iscrizione
Attori	Manager
Obiettivi	Rinnovo iscrizione alla piattaforma
Precondizioni	Il manager deve essere nella pagina di login
Passi	<p>1. Il sistema, all'autenticazione alla piattaforma, verifica che il manager sia attivo e debba effettuare il rinnovo dell'iscrizione.</p> <p>2.a. Rinnovo dell'iscrizione</p> <p>2.a.1. In caso di necessario rinnovo, il sistema mostra al manager i dati da compilare per effettuare il pagamento del rinnovo dell'iscrizione.</p> <p>2.a.2. Il sistema riporta alla pagina di pagamento (UC14) e, in caso di esito positivo di esso, permette l'accesso al manager.</p>

	<p>2.b. Il Manager esce dal rinnovo dell'iscrizione nel UC14 e non viene autenticato.</p> <p>2.c. In caso di non necessità di rinnovo, il manager viene regolarmente autenticato</p>
Casi d'uso associati / flussi alternativi	<<include>> UC14: Pagamento <<extend>> UC2: Login
Postcondizioni	L'iscrizione alla piattaforma è attiva regolarmente nel caso di pagamento completato

Nome	UC24: Newsletter
Descrizione	Il sistema invia una newsletter in base alle impostazioni scelte dagli utenti
Attori	Admin
Obiettivi	Inviare una newsletter sugli eventi futuri
Precondizioni	L'Admin deve essere autenticato
Passi	<ol style="list-style-type: none"> 1. L'amministratore richiede l'invio della newsletter agli iscritti. 2. Il sistema elabora una mail con gli eventi preferiti dall'utente che si è registrato alla newsletter e la invia a esso
Casi d'uso associati / flussi alternativi	/
Postcondizioni	Invio delle newsletter agli utenti registrati completato

Alcuni casi d'uso sono stati modificati come di seguito:

Nome	UC2: Login
Descrizione	L'utente può accedere al suo account per effettuare prenotazioni, se utente, o altre funzioni, se manager o ticket inspector
Attori	User, Manager, Ticket Inspector, Admin
Obiettivi	Accedere alla propria area privata nel sito per svolgere funzioni

Precondizioni	Essere registrato
Passi	<ol style="list-style-type: none"> 1. L'attore entra nella pagina di login 2. L'attore compila il campo "username" 3. L'attore compila il campo "password" 4. L'attore richiede di essere autenticato con quelle credenziali 5. Il sistema autorizza l'accesso all'utente e lo porta nella sua area riservata <ol style="list-style-type: none"> 5.a. Nel caso di autenticazione di un User, il sistema riporta alla pagina di visualizzazione evento 5.b. Nel caso di autenticazione di un Manager, il sistema riporta alla pagina di visualizzazione eventi del manager 5.c. Nel caso di autenticazione di un Ticket Inspector, il sistema riporta alla pagina di scansione del QR 5.d. Nel caso di autenticazione di un Admin, il sistema riporta alla pagina di gestione manager
Casi d'uso associati / flussi alternativi	/
Postcondizioni	L'utente è autenticato

Nome	UC9: Sign Up Manager
Descrizione	Il sistema permette all'utente di registrarsi come Manager (organizzatore di eventi)
Attori	Manager
Obiettivi	Registrare un Manager come tale
Precondizioni	Il Manager non è ancora registrato
Passi	<ol style="list-style-type: none"> 1. L'utente richiede la registrazione alla piattaforma come Manager 2. Il sistema mostra la pagina di registrazione alla piattaforma come Manager 3. L'utente compila tutti i dati anagrafici e dell'organizzazione 4. L'utente richiede la registrazione con i dati inseriti

	<p>5. Il sistema controlla i dati inseriti</p> <p>6.a. Validazione corretta</p> <p>6.a.1. In caso di validazione corretta, il sistema riporta alla pagina di pagamento della quota di iscrizione (UC14).</p> <p>6.a.2.a. Se il pagamento va a buon fine, il sistema inserirà il manager nella lista dei manager richiedenti l'attivazione da parte di un amministratore.</p> <p>6.a.2.b. Se il pagamento non va a buon fine, viene rimostrata la pagina di pagamento.</p> <p>6.b. In caso di validazione errata, il sistema richiede la modifica dei dati errati.</p>
Casi d'uso associati / flussi alternativi	<<include>> UC14: Pagamento
Postcondizioni	Il Manager è registrato ma non attivato se la registrazione è andata a buon fine

Nome	UC14: Pagamento
Descrizione	Viene effettuato il pagamento dell'evento da prenotare o della quota di iscrizione.
Attori	User, Manager
Obiettivi	Effettua il pagamento di un evento o della quota di iscrizione
Precondizioni	<ul style="list-style-type: none"> • L'utente deve essere loggato • L'utente deve essere nella pagina della visualizzazione dei dettagli di un evento o nella pagina della visualizzazione di un eventour o nella pagina di registrazione di un manager o di rinnovo iscrizione
Passi	<p>Pagamento di un evento o di un eventour</p> <p>1. Il sistema, durante la prenotazione, controlla che l'evento non sia gratuito e mostra in tal caso la richiesta di inserimento dei dati di pagamento</p> <p>2.a. L'utente vuole confermare il pagamento e la prenotazione</p> <p>2.a.1. Il sistema carica una pagina di inserimento dati per il metodo di pagamento</p>

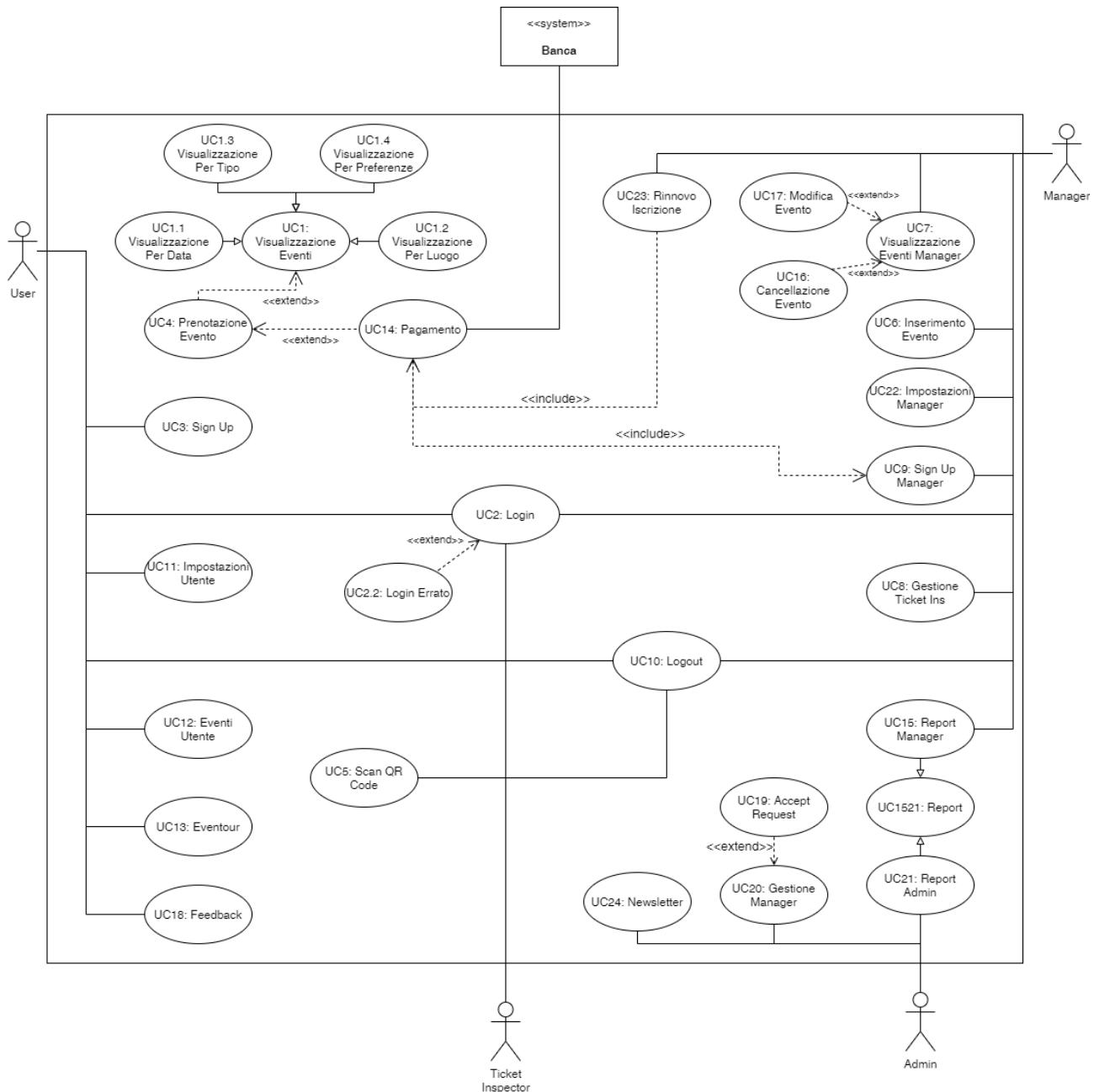
	<p>2.a.2. L'utente inserisce i dati della carta</p> <p>2.a.3. L'utente conferma i dati inseriti</p> <p>2.a.4. Il sistema controlla i dati inseriti e richiede il pagamento</p> <p>2.a.5.a. Se la transazione viene confermata, il sistema avvisa della conferma del pagamento e conferma la prenotazione</p> <p>2.a.5.b. Se la transazione non viene confermata, il sistema avvisa dell'errore riscontrato durante il pagamento e non conferma la prenotazione</p> <p>2.b. L'utente non vuole confermare il pagamento</p> <p>2.b.1. L'utente annulla la richiesta di pagamento o riaggiorna la pagina</p> <p>2.b.2. Il sistema non effettua la prenotazione</p> <p>Pagamento della quota di iscrizione</p> <p>1. Il sistema, durante la registrazione o durante il rinnovo dell'iscrizione, in cui verifica se l'utente deve rinnovare l'iscrizione, mostra la richiesta di inserimento dei dati di pagamento</p> <p>2.a. Il manager vuole confermare il pagamento della quota</p> <p>2.a.1. Il sistema carica una pagina di inserimento dati per il metodo di pagamento</p> <p>2.a.2. Il manager inserisce i dati della carta</p> <p>2.a.3. Il manager conferma i dati inseriti</p> <p>2.a.4. Il sistema controlla i dati inseriti e richiede il pagamento</p> <p>2.a.5.a. Se la transazione viene confermata, il sistema avvisa della conferma del pagamento e registra l'utente come da attivare, nel caso della registrazione, o con data di rinnovo odierna, nel caso di rinnovo.</p>
--	---

	<p>2.a.5.b. Se la transazione non viene confermata, il sistema avvisa dell'errore riscontrato durante il pagamento e attende nuovi dati.</p> <p>2.b. Il manager non vuole confermare il pagamento</p> <p> 2.b.1. Il manager annulla la richiesta di pagamento o riaggiorna la pagina</p> <p> 2.b.2. Il sistema non effettua il pagamento</p>
Casi d'uso associati / flussi alternativi	<p><<extend>> UC4: Prenotazione evento</p> <p>Flusso alternativo 1: L'attore non conferma i dati inseriti con conseguente cancellazione delle informazioni.</p> <p>Flusso alternativo 2: L'attore inserisce le informazioni del metodo di pagamento, ma non richiede il pagamento, uscendo prima della conferma di transazione.</p>
Postcondizioni	L'attore ottiene un messaggio di notifica di avvenuto pagamento se tutto è andato a buon fine

Nome	UC16: Cancellazione Evento
Descrizione	Un evento viene cancellato dall'elenco degli eventi creati.
Attori	Manager
Obiettivi	Cancellare un evento
Precondizioni	<ul style="list-style-type: none"> • Il manager deve essere loggato • Il manager deve essere nella pagina di visualizzazione di un evento specifico
Passi	<ol style="list-style-type: none"> 1. Il manager seleziona un evento creato 2. Il manager clicca su "Cancella evento" 3. Il sistema cancella l'evento dal database, elimina tutti i ticket inspectors associati a quell'evento e cancella tutte le prenotazioni relative a esso, avvisando via mail gli utenti che avevano prenotato 4. Il sistema aggiorna la pagina di visualizzazione degli eventi

Casi d'uso associati / flussi alternativi	/
Postcondizioni	L'evento è cancellato

Use Case diagram

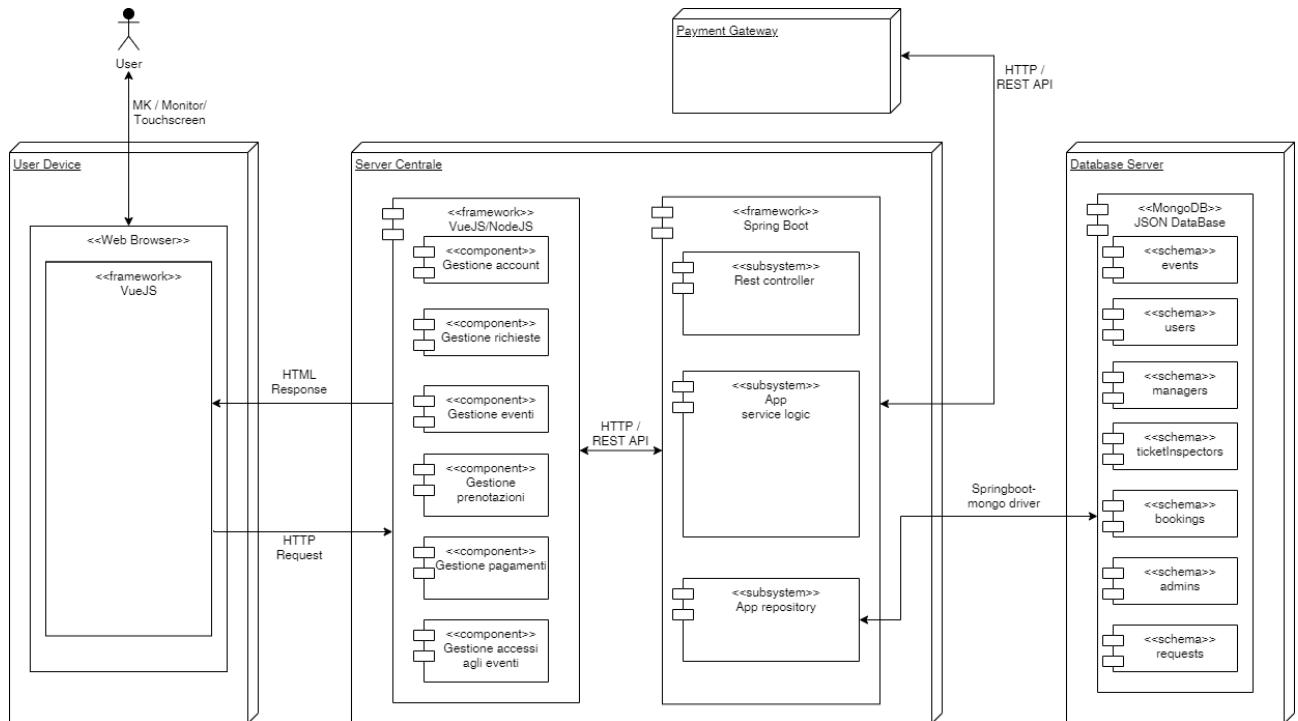


Modello 5.1 Use Case Diagram

2. Architettura

Deployment diagram

Per quanto concerne il deployment diagram, l'introduzione di nuove funzionalità ha richiesto l'aggiunta di nuovi componenti interni alla parte web visuale del sistema e l'introduzione di nuove collezioni di dati.

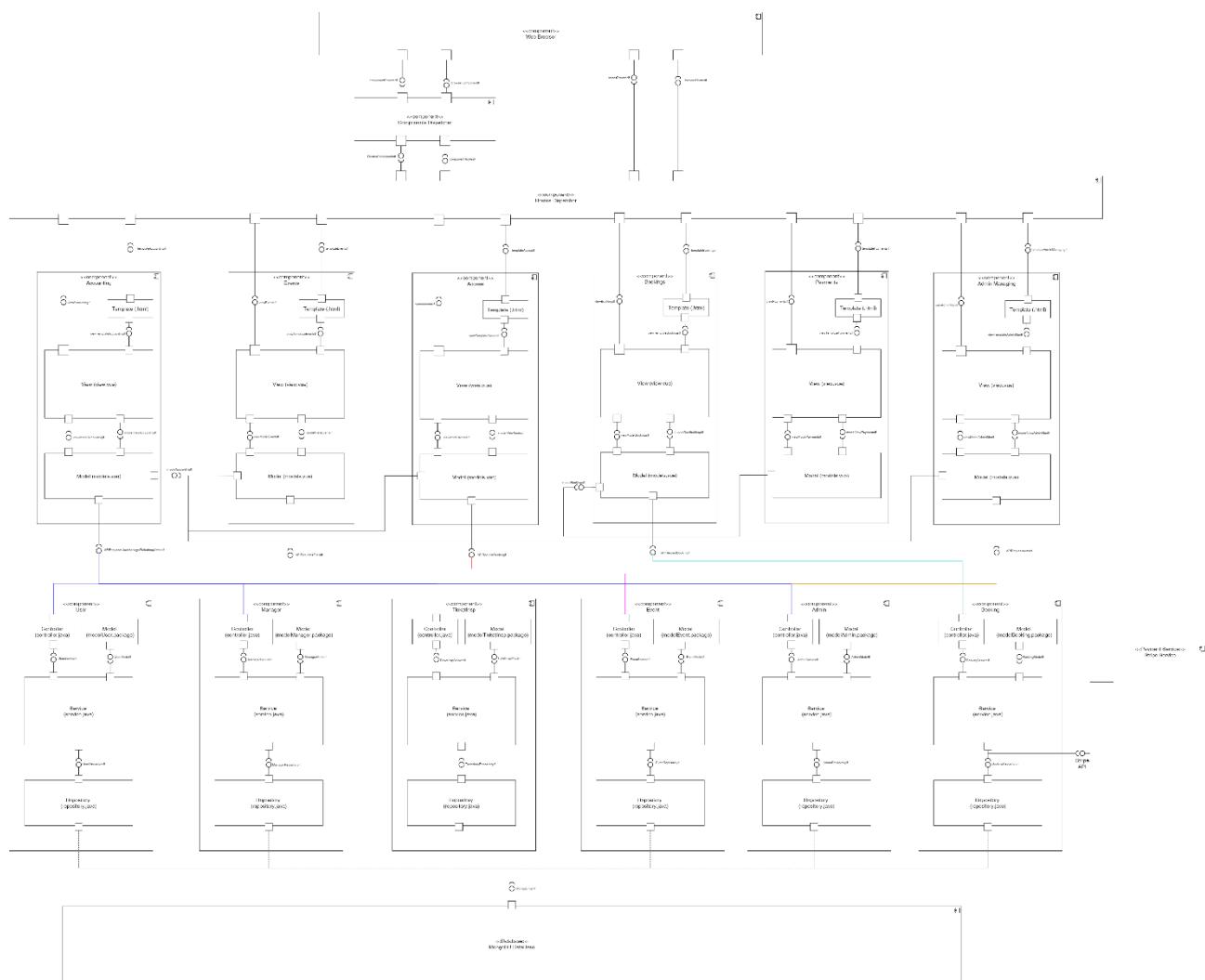


Modello 5.2 Deployment Diagram

Component diagram

La modifica del deployment diagram si ripercuote strettamente nel component diagram, in cui le iterazioni tra i vari model in Vue si incrementano e in cui viene aggiunto un nuovo macrocomponente per la gestione degli amministratori. La parte di gestione degli amministratori a livello di API viene elaborata in un suo componente che comunica con altri componenti.

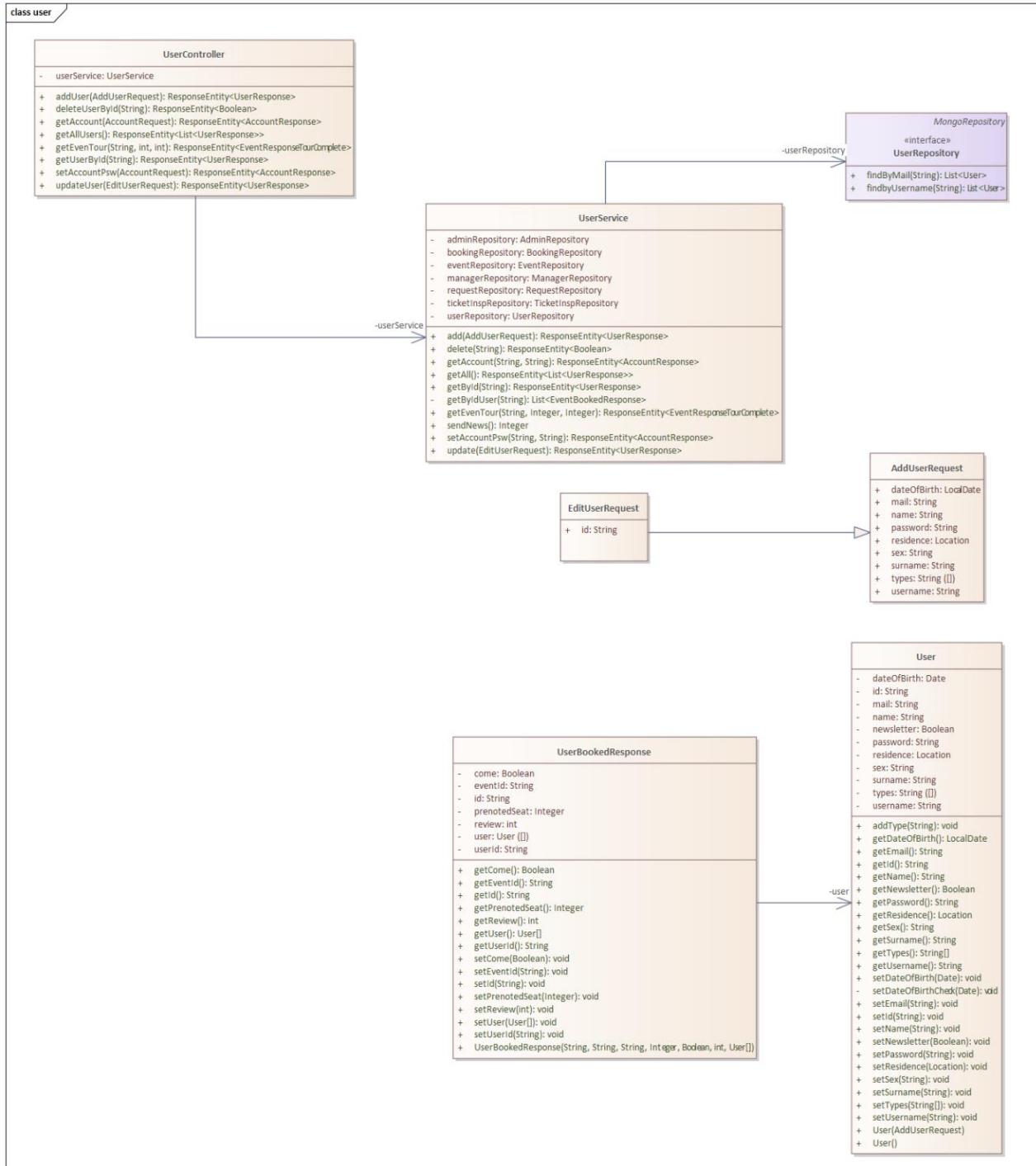
Si riportano le aggiunte di seguito.



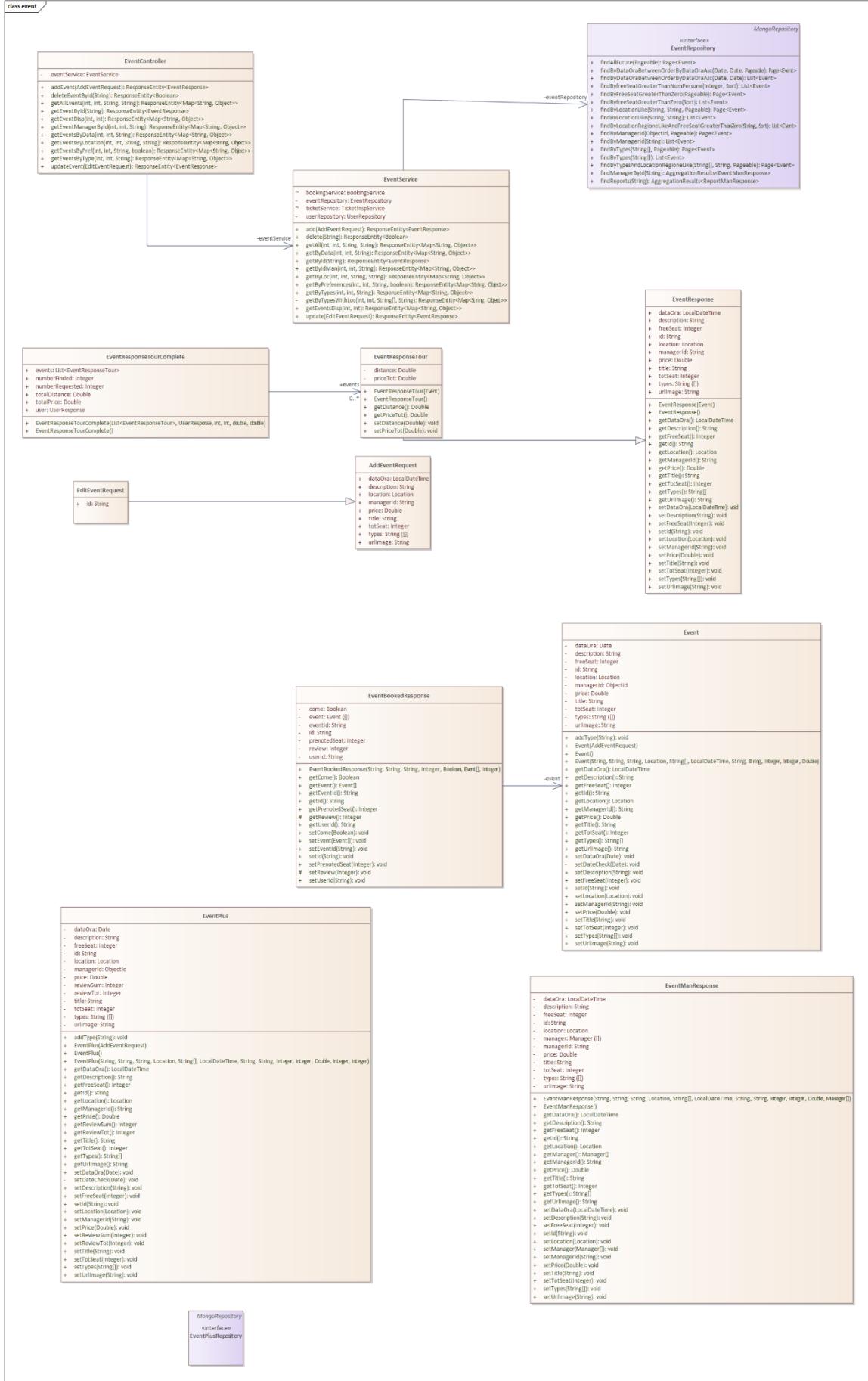
Modello 5.3 Component Diagram

3. Class diagram

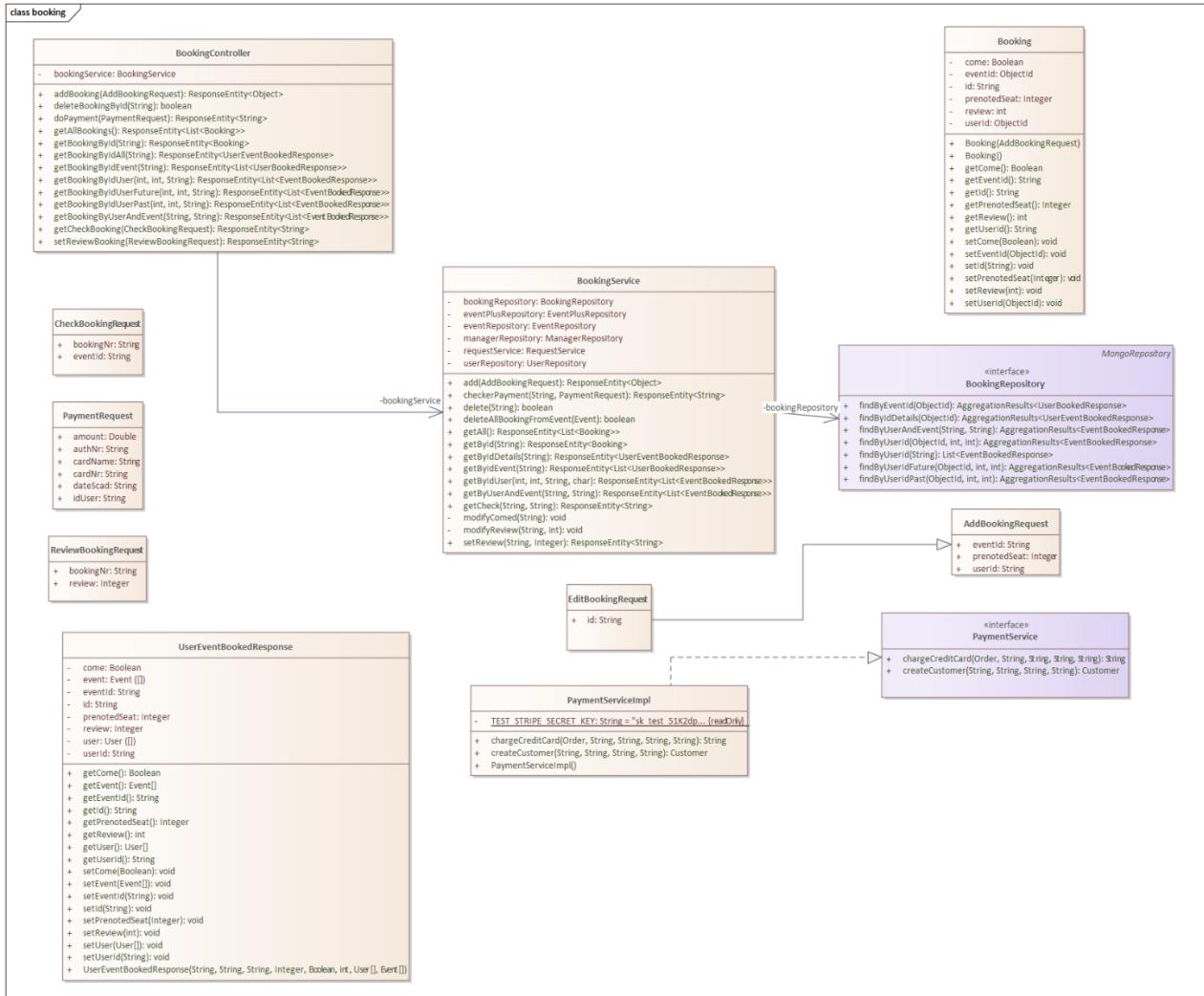
Di seguito sono mostrati i class diagram relativi alle classi utilizzate per l'implementazione dell'iterazione corrente.



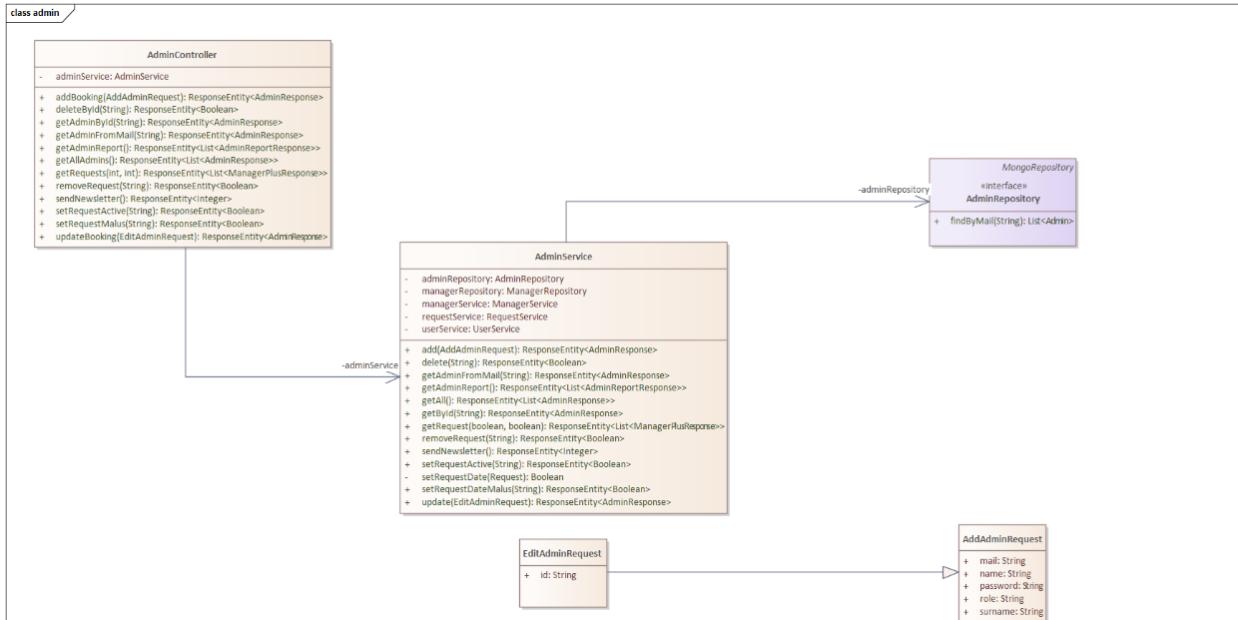
Modello 5.4 User Class Diagram



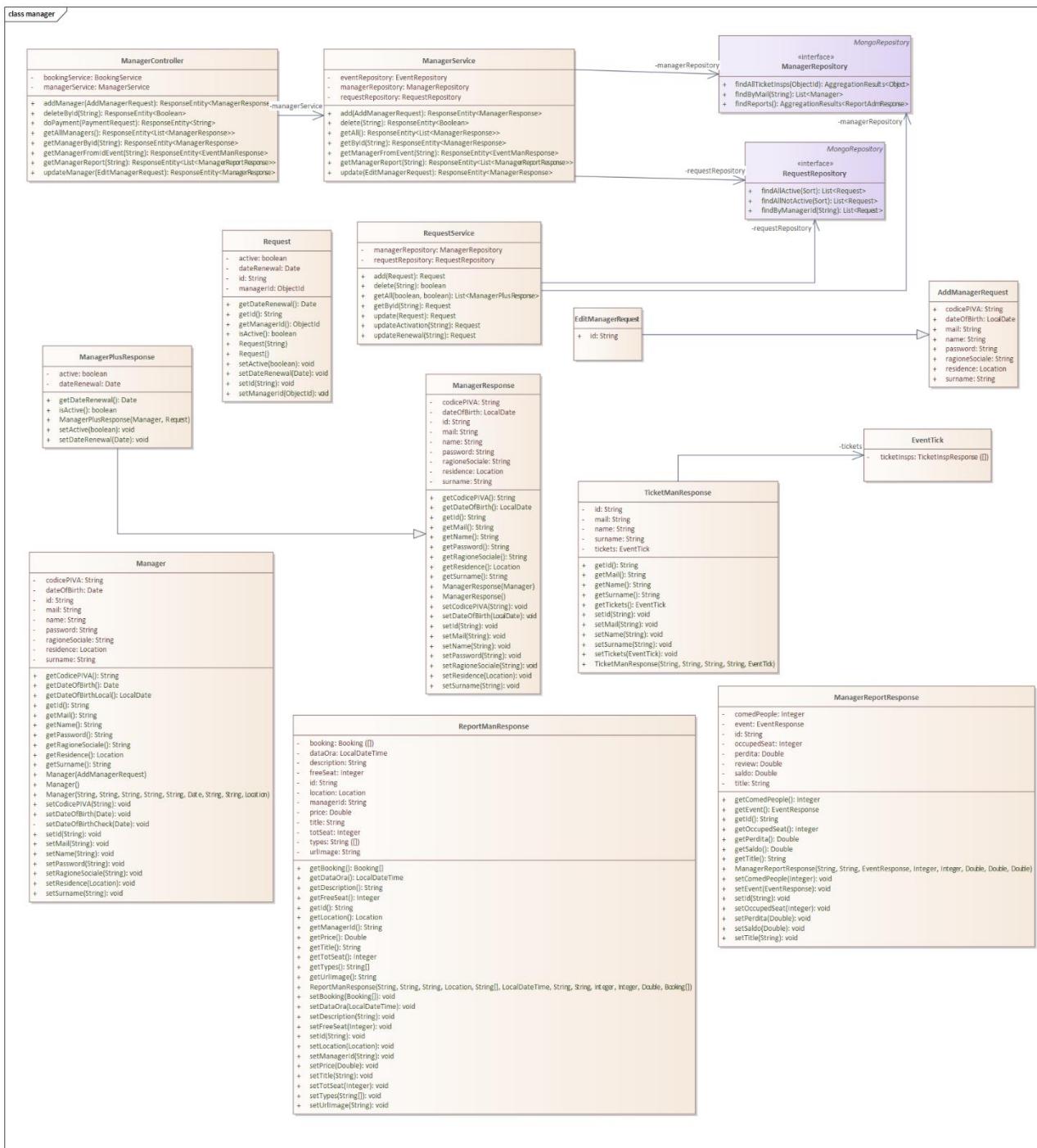
Modello 5.5 Event Class Diagram



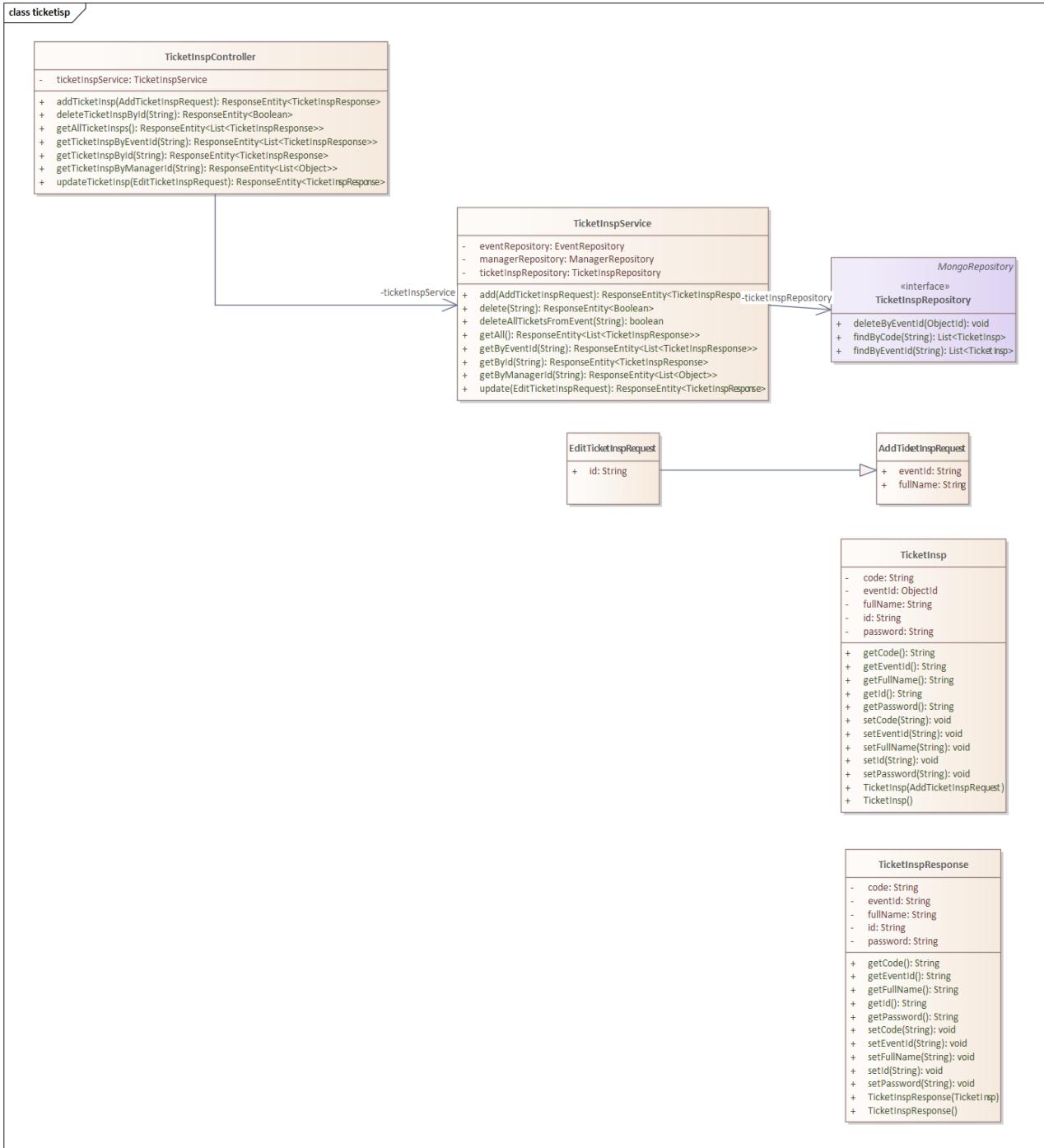
Modello 5.6 Booking Class Diagram



Modello 5.7 Admin Class Diagram



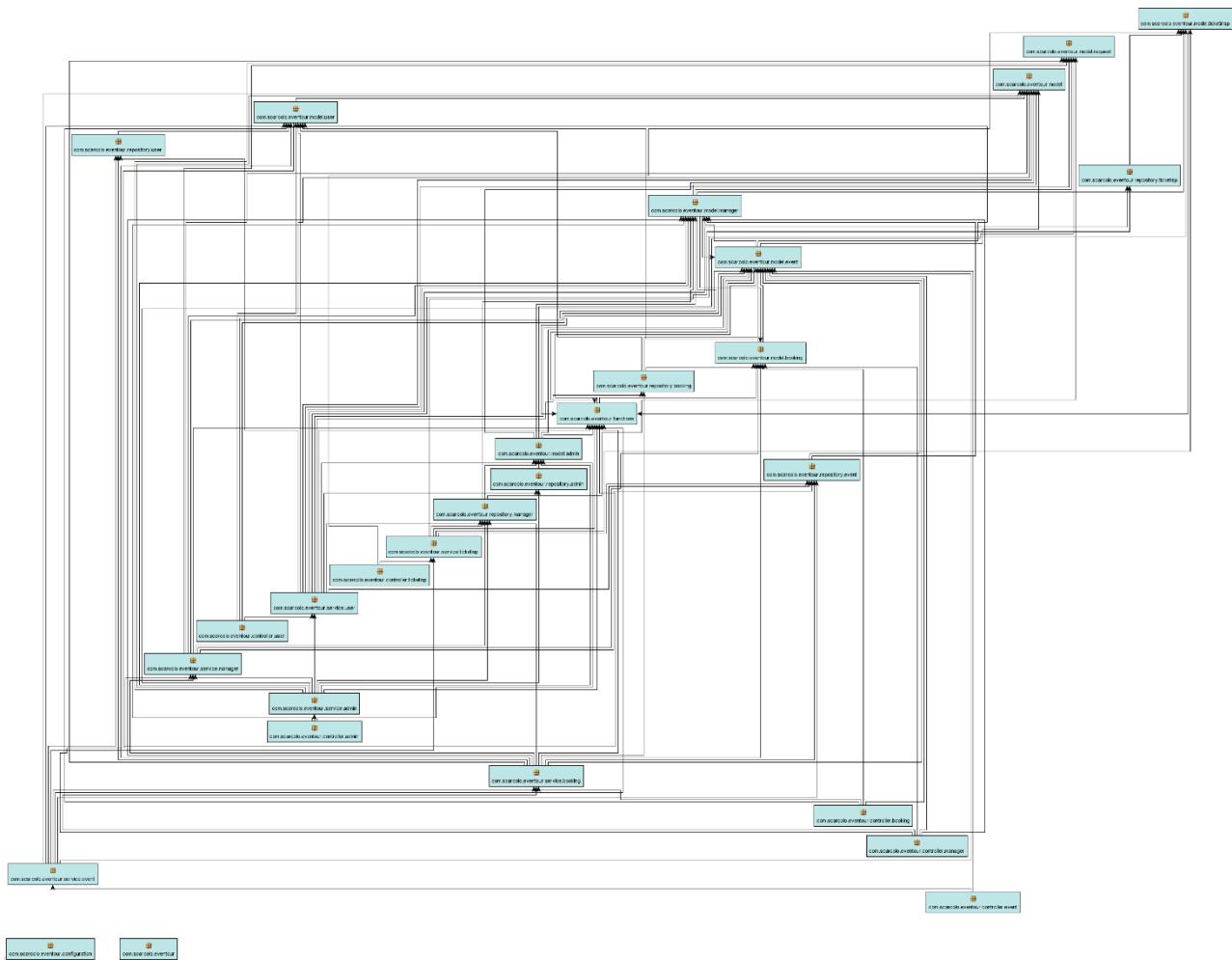
Modello 5.8 Manager Class Diagram



Modello 5.9 Ticket Inspector Class Diagram

A causa dell'aumento di complessità dei casi d'uso introdotti, anche le classi hanno avuto un incremento di metodi e funzioni introdotte, o addirittura l'introduzione di nuovi modelli e strutture applicanti il pattern architetturale CSR.

Si riporta di seguito il package diagram, dove si nota l'aumento di complessità.



Modello 5.10 Package Diagram

4. Algoritmo evenTour

Lo scopo della versione corrente dell'algoritmo progettato è quella di trovare ancora un numero di eventi definito dalla invocazione della API, che siano disponibili per un numero definito di persone e che creino un tour di eventi, che l'utente può fare.

Per prima cosa, l'algoritmo ricerca l'utente in base al suo id (ipotizzando d utenti presenti e ipotizzando che MongoDB sfrutti l'indice come chiave, portando la ricerca a $O(1)$). Nel caso in cui l'utente non esista viene segnalato l'errore direttamente.

Vengono poi ricercati tutti gli eventi che hanno almeno nPersone posti disponibili, successivi alla data odierna e ordinati per data. Esso ha complessità tempo pari a $O(N_{tot})$ per la selezione degli eventi compatibili e $O(n * \log n)$ nell'ordinamento di tutti i dati selezionati. Ipotizziamo che tutti gli eventi siano corrispondenti alla richiesta ($n=N_{tot}$)

Vengono poi selezionate tutte le prenotazioni fatte dall'utente (con complessità tempo $O(M)$).

L'algoritmo esclude poi, come nella iterazione precedente, tutti gli eventi già prenotati o gli eventi che hanno date uguali a essi. Ciò richiederà $O(m * n)$.

```

1. //id: id dell'utente, N: numero eventi richiesti da un utente, nPersone: numero di persone ch
e vogliono prenotare l'eventour
2. function EvenTour(id, N, nPersone)
3. {
4.     datiUtente <- ricercaUtenteDaID(id); //O(1)
5.     SE (datiUtente NON SONO PRESENTI) { //O(1)
6.         return ERROR;
7.     }
8.
9.     //n: numero di eventi in listaEventi
10.    listaEventi <- ricercaEventiConNPersonePostiDisponibiliOrdinatiPerData
                                         (datiUtente.regione);
                                         //O(n*log n)
11.
12.    //M: elementi totali delle prenotazioni
13.    bookingData <- ricercaPrenotazioniDaIdUtente(id); //O(M)
14.    //m: elementi totali di bookingData (m<=M)
15.
16.    s <- Lista vuota //Soluzione da restituire
17.
18.    SE (bookingData ha eventi) ALLORA //O(m*n)
19.    {
20.        PER OGNI eventoPrenotato IN bookingData //O(m*n)
21.        {
22.            rimuoviEventoGiàPrenotato(listaEventi, eventoPrenotato); //O(n)
23.            rimuoviEventiConDateUguali(listaEventi, eventoPrenotato); //O(n)
24.        }
25.    }
26.

27.    latU <- datiUtente.Residenza.Latitudine //O(1)
28.    lngU <- datiUtente.Residenza.Longitudine //O(1)

```

Codice 5.1: Algoritmo parte 1

L'algoritmo si divide poi in due macroblocchi:

- Selezione dell'evento con distanza dall'utente minore di una distanza fissata e che sia ottimale per lui
- Selezione, successiva, degli eventi ottimali che siano a una distanza minore dall'evento precedentemente selezionato nella soluzione, con limitazioni in un range di date.

Per il primo blocco, l'algoritmo scorre tutti gli eventi filtrati, per ognuno calcola la distanza tra utente ed evento sfruttando i valori di latitudine e longitudine secondo la formula:

$$\text{distanza } (A, B) = R * \arccos(\sin(latA) * \sin(latB) + \cos(latA) * \cos(latB) * \cos(lonA - lonB))$$

La funzione sottostante è l'implementazione di essa in codice dopo previa conversione delle latitudini e longitudini da gradi a radianti.

```

public static Double distance(Double lat1, Double lng1, Double lat2, Double lng2)
{
    //punto A: conversione da gradi a radianti delle latitudini e longitudini del primo punto
    Double[] pointA= {lat1*Math.PI/180,lng1*Math.PI/180};
    //punto B: conversione da gradi a radianti delle latitudini e longitudini del secondo punto
    Double[] pointB= {lat2*Math.PI/180,lng2*Math.PI/180};
    //calcolo della distanza in linea d'aria tra i due punti in KM
    int R=6372.795477598;
    return R *
        Math.acos(
            Math.sin(pointA[0]) * Math.sin(pointB[0])
            +
            Math.cos(pointA[0]) * Math.cos(pointB[0])
            * Math.cos(pointA[1]-pointB[1])
        );
}

```

Codice 5.2: Funzione calcolo distanza in linea d'aria

Dopo ciò, l'algoritmo, nel caso l'evento rispetti le condizioni precedentemente imposte di tipologia di evento, attribuisce un peso all'evento sulla base di uguaglianza o similitudine di tipi (in questo caso il peso è moltiplicato per 1.25 volte), distanza dall'utente e prezzo rispetto all'evento selezionato come ottimale fino a quel momento.

In questo modo l'algoritmo andrà a minimizzare questo peso per trovare l'evento che fornisce l'evento iniziale ottimale per l'utente. Nel caso in cui nessun evento venga selezionato, l'algoritmo restituisce un errore.

In maniera grafica si può vedere la scelta fatta dall'algoritmo di seguito:

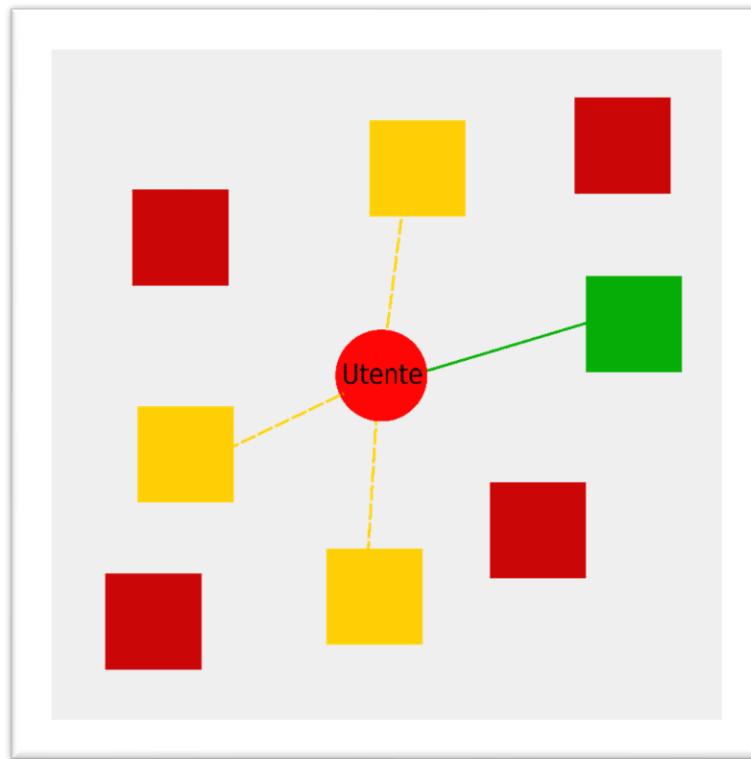


Figura 5.1 Scelta del miglior evento iniziale

```

29.
30.    rimuoviEventiMaggioriDiUnaCertaDistanza(listaEventi, latU, lngU); //O(n)
31.    eventoScelto <- listaEventi(0) //O(1)
32.    distSel <- 0.0 //O(1)
33.    peso <- MAX_DOUBLE_VALUE //O(1)
34.
35.    PER OGNI evento IN listaEventi //O(n*t*x)
36.    {
37.        dist <- calcolaDistanza(latU, lngU, evento.lat, evento.lng) //O(1)
38.        //t: numero tipi per ogni evento
39.        PER OGNI tipo IN evento.tipi //O(t*x)
40.        {
41.            //x: numero tipi preferiti dall'utente
42.            PER OGNI tipoUtente IN datiUtente.tipi //O(x)
43.            {
44.                SE (tipo = tipoUtente) //O(1)
45.                {
46.                    SE (peso > 0.5 * (evento.Prezzo - eventoScelto.Prezzo) + 0.5 * dist)
47.                        //O(1)
48.                    {
49.                        eventoScelto <- evento //O(1)
50.                        distSel <- dist //O(1)
51.                        peso <- 0.5 * (evento.Prezzo - eventoScelto.Prezzo) + 0.5 * dist
52.                        //O(1)
53.                    }
54.                }
55.            }
56.        }
57.    }
58.
```

```

52.          }
53.          ALTRIMENTI SE (tipiSimili(tipo, tipoUtente))
54.          {
55.              SE (peso > 1.25 * (0.5 * (evento.Prezzo - eventoScelto.Prezzo)
56.                                + 0.5 * dist)) //0(1)
57.              {
58.                  eventoScelto <- evento //0(1)
59.                  distSel <- dist //0(1)
60.                  peso <- 0.5 * (evento.Prezzo - eventoScelto.Prezzo) + 0.5 * dist
61.                                //0(1)
62.              }
63.
64.          }
65.      }
66.
67.      SE (eventoScelto NON ESISTE) { //0(1)
68.          return ERROR; //0(1)
69.      }
70.
71.      s <- s U eventoScelto //0(1)

```

Codice 5.3: Algoritmo parte 2

Per il secondo blocco, l'algoritmo scorre tutti gli eventi selezionati prima del filtro sulla distanza e inizia a riscorrerli tutti (tranne quello selezionato). Per ogni evento da selezionare successivamente per raggiungere il numero di eventi richiesti, l'algoritmo esclude gli eventi superiori a una distanza fissata dall'ultimo evento selezionato e che siano compresi tra la data dell'evento selezionato e le due settimane successive a esso. Per ognuno, scorre gli eventi filtrati e calcola la distanza tra evento selezionato precedentemente ed evento attuale sfruttando i valori di latitudine e longitudine secondo la formula espressa prima.

Dopo ciò, l'algoritmo, nel caso l'evento rispetti le condizioni precedentemente imposte di tipologia di evento, attribuisce un peso all'evento sulla base di uguaglianza o similitudine di tipi (in questo caso il peso è moltiplicato per 1.25 volte), distanza dall'utente e prezzo rispetto all'evento selezionato come ottimale fino a quel momento.

Si mostra quindi la selezione del secondo evento e del tour di eventi completo di seguito:

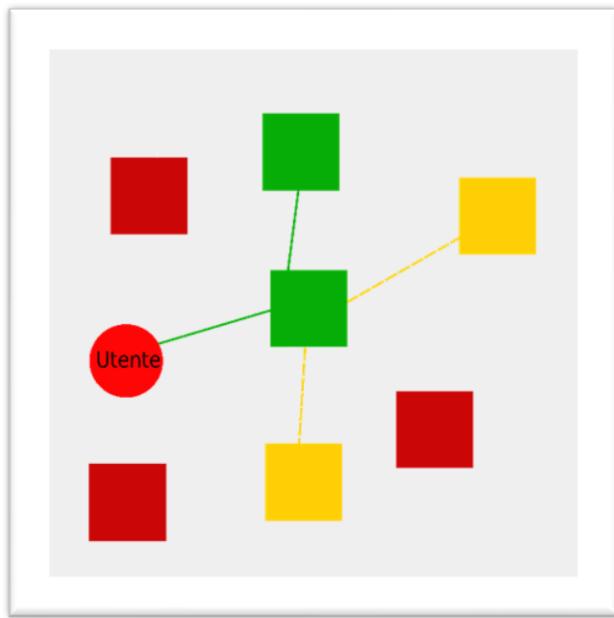


Figura 5.2 Selezione secondo evento

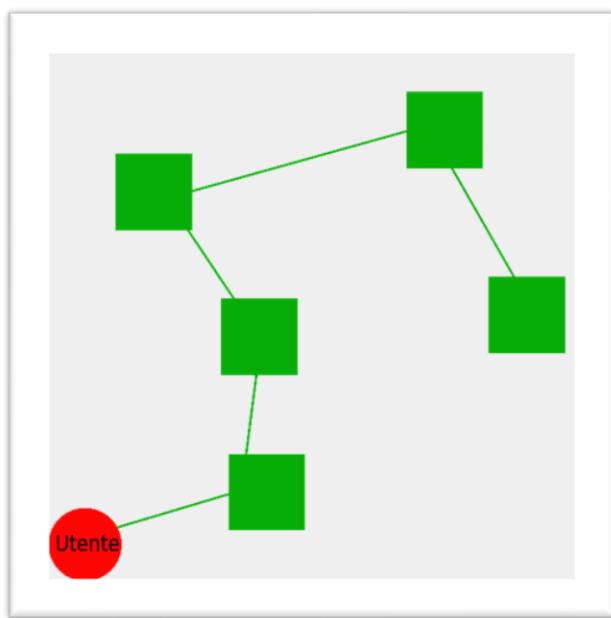


Figura 5.3 Selezione di tutti gli eventi

Dopo aver selezionato un numero di eventi richiesto dall'utente, l'algoritmo calcola la distanza totale e la spesa totale raggiunte con il tour, comprendendo anche il ritorno al comune di residenza come distanza, visibile in questa immagine:

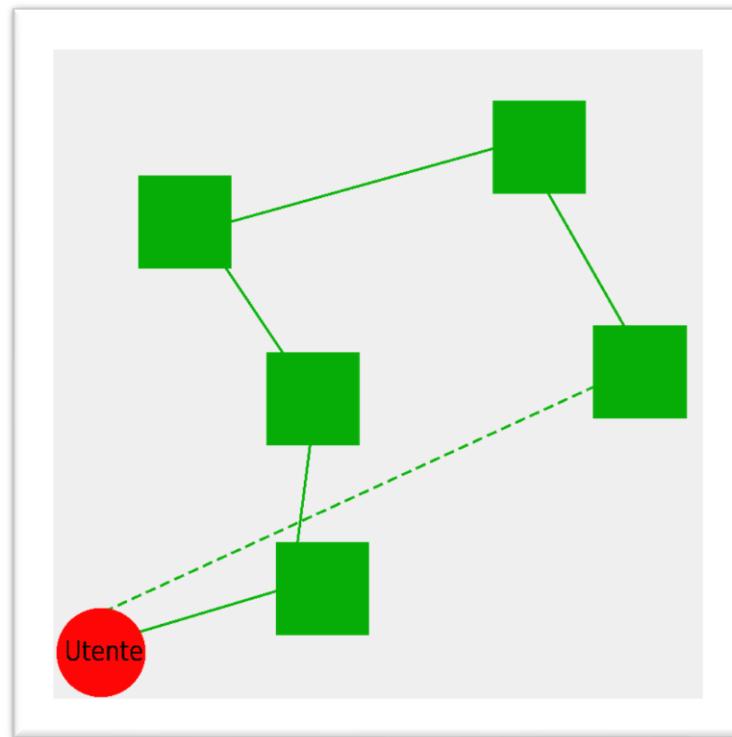


Figura 5.4 Vista della distanza percorsa

Si riporta lo pseudocodice dell'algoritmo rimanente comprendente di complessità tempo:

```

72.    choiceNull <- false
73.
74.    //s: numero di eventi del tour
75.    WHILE (s.size < N AND choiceNull IS FALSE) { //O(s*n*t*x)
76.        listaEventi <- listaEventi - eventoScelto //O(1)
77.        latRef <- s[ultimo].lat //O(1)
78.        lngRef <- s[ultimo].lng //O(1)
79.        rimuoviEventiAdUnaCertaDistanzaEEntroDueSettimaneDallUltimoSelezionato
80.                                         (latRef, lngRef, listaEventi ); //O(n)
81.
82.        SE (listaEventi.size = 0) ALLORA { O(n*t*x)
83.            choiceNull <- true O(1)
84.        } ALTRIMENTI {
85.            eventoScelto <- listaEventi(0) //O(1)
86.            peso <- MAX_DOUBLE_VALUE //O(1)
87.
88.            PER OGNI evento IN listaEventi //O(n*t*x)
89.            {
90.                dist <- calcolaDistanza(latRef, lngRef, evento.lat, evento.lng) //O(1)
91.                PER OGNI tipo IN evento.tipi //O(t*x)
92.                {
93.                    PER OGNI tipoUtente IN datiUtente.tipi O(x)
94.                    {
95.                        SE (s VUOTO OR NOT dateuguali(evento, s[s.size - 1])) //O(1)
96.                        {
97.                            SE (tipo = tipoUtente) //O(1)
98.                            {
99.                                SE (peso > 0.5 * (evento.Prezzo - eventoScelto.Prezzo)
100.                                     + 0.5 * dist)
101.                                    {
102.                                        eventoScelto <- evento //O(1)
103.                                        distSel <- dist //O(1)
104.                                        peso <- 0.5 * (evento.Prezzo - eventoScelto.Prezzo)
105.                                         + 0.5 * dist //O(1)
106.                                    }
107.                                }
108.                                ALTRIMENTI SE (tipiSimili(tipo, tipoUtente))
109.                                {
110.                                    peso <- 1.25 * (0.5 * (evento.Prezzo -
111.                                         eventoScelto.Prezzo) + 0.5 * dist)
112.                                         //O(n)
113.                                }
114.                            }

```

```

115.                     }
116.                 }
117.             }
118.         }
119.     }
120.     SE (eventoScelto NON ESISTE) ALLORA {
121.         s <- s U eventoScelto
122.     } ALTRIMENTI {
123.         choiceNull <- True
124.     }
125. }
126. }
127.
128. eventResponse <- []
129. distTot <- 0.0
130. priceTot <- 0.0
131.
132. PER OGNI (evento IN s) { //0(s)
133.     distTot <- distTot + sommaDistanza(evento, eventoPrecedente, utente) //0(1)
134.     priceTot <- priceTot + sommaPrezzo(evento) //0(1)
135. }
136. distTot <- sommaDistanzaFinale(distTot, distFinale) //0(1)
137. return s, distTot, priceTot;
138.
139. }

```

Codice 5.4: Algoritmo parte 3

5. Variazioni al database nella iterazione corrente

Nella iterazione 3 si è scelto di inserire le collezioni di dati aventi struttura specificata dalla Tabella 5.1 per quanto riguarda l'admin e le richieste di iscrizione da parte del manager.

Si è anche aggiunto un campo per ogni utente che specifica se richiede o meno l'iscrizione alla newsletter.

Inoltre, sono stati introdotti nuovi campi per le prenotazioni e per gli eventi. Per quanto riguarda le prenotazioni, si è inserito un campo numerico Int32 che salva la valutazione data dall'utente per quell'evento (da 1 a 5). Se il valore è -1, allora non è stata effettuata nessuna valutazione. Gli eventi comprendono due campi "reviewSum" e "reviewTot", aggiornati ad ogni cambio di valutazione delle prenotazioni, che permettono, seppur con ridondanza, di avere sempre i dati di quanti utenti hanno valutato quell'evento e effettuare la media delle valutazioni.

Collezione	Campo	Tipo di dato	Indice	Descrizione
admins	_id	ObjectId	✓	ID univoco generato all'inserimento nel database
	mail	String		Mail di accesso
	password	String		Password di accesso
	name	String		Nome amministratore
	surname	String		Cognome amministratore
	role	String		Ruolo dell'amministratore, tra ADMIN, MODERATOR e HELPER

Collezione	Campo	Tipo di dato	Indice	Descrizione
requests	_id	ObjectId	✓	ID univoco generato all'inserimento nel database
	managerId	ObjectId	✓	ID univoco del manager
	active	Boolean		Identifica se il manager è attivo o meno sulla piattaforma, diventa attivo quando l'amministratore lo abilita
	dateRenewal	Date		Data di rinnovo dell'iscrizione o della attivazione

Tabella 5.1 Struttura dati delle nuove collezioni presenti nel database alla iterazione corrente

6. Testing interfaccia grafica

Analisi statica

Per quanto riguarda la parte di analisi statica, è stato utilizzato nuovamente il componente di EsLint che permette l'analisi del codice. Come espresso precedentemente, sono state riportati tutti i problemi legati alla struttura del codice sia per la parte grafica sia per la parte legata ai dati. Gli errori e i warning espressi nell'iterazione precedente permangono anche in questa iterazione.

Analisi dinamica

Anche per questa iterazione si è usato Nightwatch per l'esecuzione dei test di analisi dinamica del codice. Si riportano in seguito alcuni dei test effettuati.

```

'Login User Rating' : function(browser) {
    browser
    .url('http://localhost:8080')
    .click('button[id=login]')
    .assert.urlEquals('http://localhost:8080/login')
    .setValue('input[id=username]', 'Colombano72@hotmail.com')
    .setValue('input[id=password]', 'yelukare')
    .click('button[id=btnlogin]')
    .waitForElementVisible('button[id=btnRatings]')
    .assert.urlEquals('http://localhost:8080/')
    .waitForElementVisible('button[id=btnRatings]')
    .click('button[id=btnRatings]')
    .assert.urlEquals('http://localhost:8080/ratings/61a0a933bce0e98fbb2d999d')
    .waitForElementVisible('button[id=logout]')
    .waitForElementVisible('div[id=divRatings]')
    .end();
},
'Login User Page And Num of Elem' : function(browser) {
    browser
    .url('http://localhost:8080')
    .click('button[id=login]')
    .assert.urlEquals('http://localhost:8080/login')
    .setValue('input[id=username]', 'Colombano72@hotmail.com')
    .setValue('input[id=password]', 'yelukare')
    .click('button[id=btnlogin]')
    .assert.urlEquals('http://localhost:8080/')
    .setValue('input[id=inputPage]', '2')
    .setValue('input[id=inputNelem]', '30')
    .click('button[id=btnCerca]')
    .assert.urlEquals('http://localhost:8080/p/2?nelem=30&order=asc')
    .waitForElementVisible('button[id=logout]')
    .end();
},
'Login User Pref' : function(browser) {
    browser
    .url('http://localhost:8080')
    .click('button[id=login]')
    .assert.urlEquals('http://localhost:8080/login')
    .setValue('input[id=username]', 'Colombano72@hotmail.com')
    .setValue('input[id=password]', 'yelukare')
    .click('button[id=btnlogin]')
    .assert.urlEquals('http://localhost:8080/')
    .setValue('input[id=inputPage]', '2')
    .setValue('input[id=inputNelem]', '30')
    .click('button[id=btnPreferenze]')
    .assert.urlEquals('http://localhost:8080/p/2?nelem=30&pref=true')
    .waitForElementVisible('button[id=logout]')
    .end();
},

```

Codice 5.5 Casi di test User: iterazione 3

```

'Login Manager Edit Event' : function(browser) {
    browser
        .url('http://localhost:8080')
        .click('button[id=login]')
        .assert.urlEquals('http://localhost:8080/login')
        .setValue('input[id=username]', 'Domenica.Acquadro@hotmail.com')
        .assert.value('input[id=username]', 'Domenica.Acquadro@hotmail.com')
        .setValue('input[id=password]', 'jesobuje')
        .click('button[id=btnlogin]')
        .waitForElementVisible('button[id=EVENT04103]')
        .click('button[id=EVENT04103]')
        .assert.urlEquals('http://localhost:8080/events/manager/61a0a85ebce0e98fbb2d9615?idAccount=61a0a0eeb5f9b12d06e95237')
            .waitForElementVisible('button[id=btnEditEvent]')
            .click('button[id=btnEditEvent]')
            .assert.urlEquals('http://localhost:8080/events/manager/edit/61a0a85ebce0e98fbb2d9615')
                .setValue('textarea[id=inputDescription]', 'Nuova Descrizione derivata dai casi di test')
                    .waitForElementVisible('input[id=inputRegione]', 10000)
                    .setValue('input[id=inputRegione]', 'lombardia')
                    .waitForElementVisible('input[id=inputProvincia]', 10000)
                    .setValue('input[id=inputProvincia]', 'milano')
                    .setValue('input[id=inputComune]', 'Legnano')
                    .waitForElementVisible('button[id=btnEdit]')
                    .click('button[id=btnEdit]')
                    .assert.urlEquals('http://localhost:8080/homemanager')
                    .end();
},

```

Codice 5.6 Casi di test Manager: iterazione 3

```

'Login and Logout Admin' : function(browser) {
    browser
    .url('http://localhost:8080')
    .click('button[id=login]')
    .assert.urlEquals('http://localhost:8080/login')
    .setValue('input[id=username]', 'admin@gmail.com')
    .assert.value('input[id=username]', 'admin@gmail.com')
    .setValue('input[id=password]', 'administrator')
    .click('button[id=btnlogin]')
    .waitForElementVisible('button[id=logout]')
    .assert.urlEquals('http://localhost:8080/admin')
    .waitForElementVisible('button[id=logout]')
    .click('button[id=logout]')
    .assert.urlEquals('http://localhost:8080/')
    .waitForElementVisible('button[id=login]')
    .end();
},
'Login Admin Report' : function(browser) {
    browser
    .url('http://localhost:8080')
    .click('button[id=login]')
    .assert.urlEquals('http://localhost:8080/login')
    .setValue('input[id=username]', 'admin@gmail.com')
    .assert.value('input[id=username]', 'admin@gmail.com')
    .setValue('input[id=password]', 'administrator')
    .click('button[id=btnlogin]')
    .waitForElementVisible('button[id=logout]')
    .assert.urlEquals('http://localhost:8080/admin')
    .click('button[id=btnReportAdmin]')
    .waitForElementVisible('div[id=tableReportAdmin]')
    .waitForElementVisible('button[id=logout]')
    .click('button[id=logout]')
    .assert.urlEquals('http://localhost:8080/')
    .waitForElementVisible('button[id=login]')
    .end();
},
'Login Admin Requests' : function(browser) {
    browser
    .url('http://localhost:8080')
    .click('button[id=login]')
    .assert.urlEquals('http://localhost:8080/login')
    .setValue('input[id=username]', 'admin@gmail.com')
    .assert.value('input[id=username]', 'admin@gmail.com')
    .setValue('input[id=password]', 'administrator')
    .click('button[id=btnlogin]')
    .waitForElementVisible('button[id=logout]')
    .assert.urlEquals('http://localhost:8080/admin')
    .waitForElementVisible('div[id=tableReportAdmin]')
    .waitForElementVisible('button[id=btnRequestAdmin]')
    .click('button[id=btnRequestAdmin]')
}

```

Codice 5.7 Casi di test Admin: iterazione 3

I test appena presentati sono aggiunte e/o modifiche rispetto ai test attuati nell'iterazione precedente. In questa iterazione, sono stati aggiunti i test relativi al nuovo ruolo, gli amministratori. I test di questa classe riprendono, in larga parte, quelli legati al ruolo del manager viste le somiglianze in termini di funzionalità. Di seguito gli output dell'esecuzione dei test:

```
<?xml version="1.0" encoding="UTF-8" ?>
<testsuites errors="0"
             failures="0"
             tests="3">

    <testsuite name="AdminTest"
               errors="0" failures="0" hostname="" id="" package="AdminTest" skipped="0"
               tests="3" time="103.9" timestamp="">

        <testcase name="Login and Logout Admin" classname="AdminTest" time="36.94"
                  assertions="7">

            </testcase>

            <testcase name="Login Admin Report" classname="AdminTest" time="42.13"
                      assertions="8">

                </testcase>

                <testcase name="Login Admin Requests" classname="AdminTest" time="24.79"
                          assertions="7">

                    </testcase>

        </testsuite>
    </testsuites>
```

Codice 5.8 Output Test Admin: iterazione 3

```

<?xml version="1.0" encoding="UTF-8" ?>
<testsuites errors="1"
    failures="1"
    tests="4">

    <testsuite name="ManagerTest"
        errors="1" failures="1" hostname="" id="" package="ManagerTest" skipped="0"
        tests="4" time="59.76" timestamp="">

        <testcase name="Login and Logout Manager" classname="ManagerTest" time="9.037"
assertions="7">

            </testcase>

            <testcase name="Login Manager Report" classname="ManagerTest" time="14.73"
assertions="7">

            </testcase>

            <testcase name="Login Manager Edit Event" classname="ManagerTest" time="17.70"
assertions="10">

            </testcase>

            <testcase name="Login Manager Add AND Delete Ticket Inspector"
classname="ManagerTest" time="18.30" assertions="8">

                <failure message="Timed out while waiting for element
<button[id=NicetoRossi]> to be present for 5000 milliseconds. - expected
visible; but got: not found" (5134ms)">      at Object.Login
Manager Add AND Delete Ticket Inspector (C:\Users\Simone\Desktop\eventour
Project\frontend-springboot-event\src\tests\ManagerTest.js:82:10)
                at processTicksAndRejections (internal/process/task_queues.js:95:5)</failure>

                <error message="Timed out while waiting for element
<button[id=NicetoRossi]> to be present for 5000 milliseconds. - expected
[0;32mvisible[0m but got: [0;31mnot found[0m [0;90m(5134ms)[0m"
type="error"><![CDATA[
                    at Object.Login Manager Add AND Delete Ticket Inspector
(C:\Users\Simone\Desktop\eventour Project\frontend-springboot-
event\src\tests\ManagerTest.js:82:10)
                    at processTicksAndRejections (internal/process/task_queues.js:95:5)
                ]]></error>

            </testcase>
        </testsuite>
    </testsuites>

```

Codice 5.9 Output Test Manager: iterazione 2

Sui casi di test relativi al manager è presente un errore dettato da questioni di carattere grafico, dato che non riesce a visualizzare e cliccare un componente a causa di un altro componente che sembra essere sopra ad esso. Questa causa un errore che non consente di proseguire con i test. In fase di esecuzione, il caso di test effettuato manualmente non presenta alcun tipo di problema e i restanti casi vengono svolti correttamente.

7. Testing API

Analisi statica

Per quanto riguarda l'analisi statica effettuata sul server per l'esposizione delle API, dopo aver generato una copia di test in MongoDB Atlas e aver modificato la stringa di connessione nell'API, si sono riscontrati circa 35 bugs segnalati da SpotBugs e varie violazioni segnalate grazie a PMD.

Per quanto riguarda Spotbugs, i bug segnalati sono stati risolti partendo da quelli più critici per finire a quelli meno critici. Sono rimasti due bugs non critici di cui uno segnala un possibile null-pointer che viene controllato a priori e una scrittura su metodo statico.

Per quanto riguarda PMD, le violazioni di livello Blocker e Critical (bandiera rossa e azzurra) sono stati risolte, mentre quelli di livello inferiore sono stati controllate e analizzate senza correggerle.

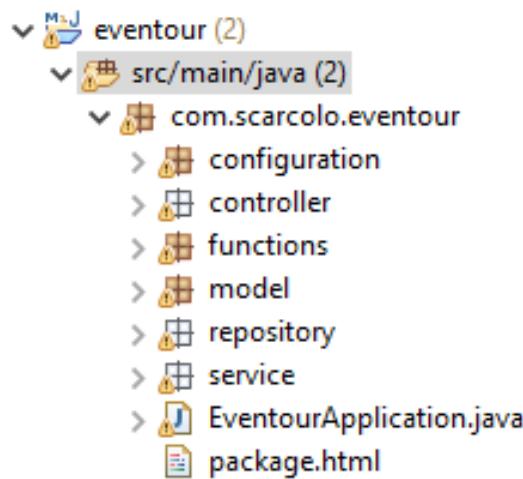


Figura 5.5 Package progetto in Eclipse con visualizzazione dei bug segnalati da spotbugs e delle violazioni di livello alto mostrate

Il report di JArchitect evidenzia inoltre il grafo delle dipendenze mostrato di seguito:

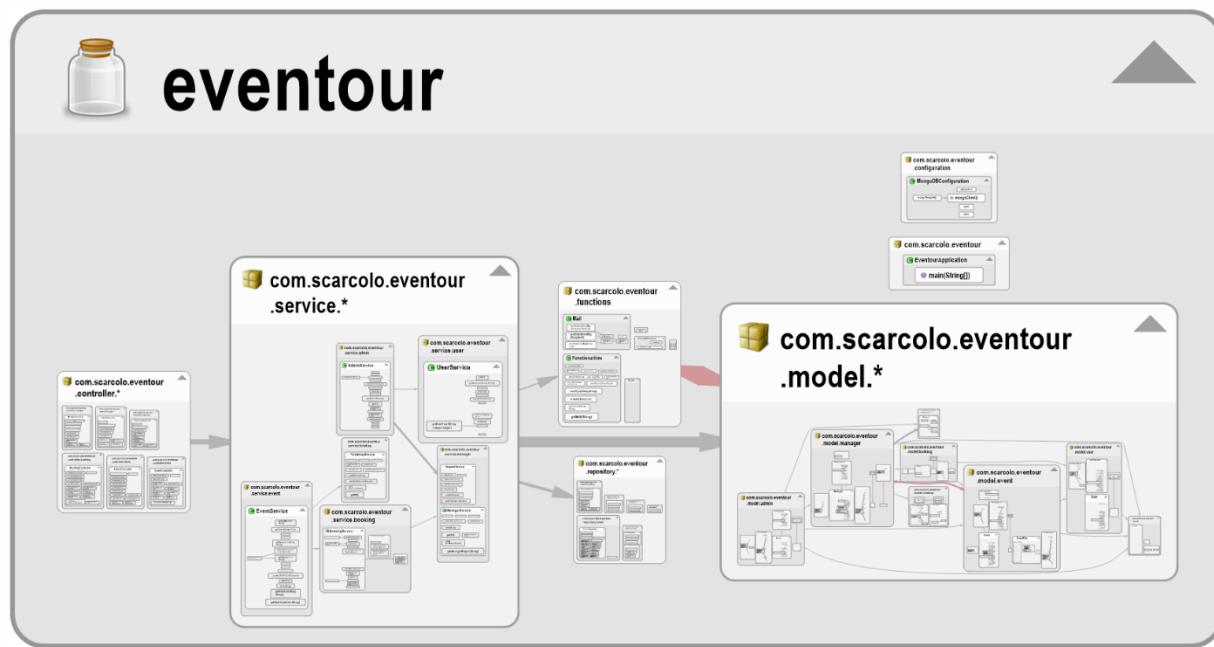


Figura 5.6 Modello report JArchitect

Analisi dinamica

Tramite JUnit si sono svolti numerosi test, come visibile nella figura di seguito. Purtroppo, i test non possono essere esaustivi di tutte le funzionalità presenti nella API in quanto tutti i metodi get e set non vengono usati spesso a causa del solo test dei costruttori richiamati dalle chiamate API effettuate. Si nota tuttavia che i test hanno coperto tutte le chiamate realizzate nella iterazione corrente.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
eventour	87,5 %	14.523	2.078	16.601
src/test/java	100,0 %	6.687	3	6.690
src/main/java	79,1 %	7.836	2.075	9.911
com.scarcolo.eventour.configuration	100,0 %	52	0	52
com.scarcolo.eventour.controller.manager	100,0 %	43	0	43
com.scarcolo.eventour.controller.ticketisp	100,0 %	37	0	37
com.scarcolo.eventour.controller.user	100,0 %	52	0	52
com.scarcolo.eventour.controller.admin	97,1 %	67	2	69
com.scarcolo.eventour.controller.booking	93,3 %	84	6	90
com.scarcolo.eventour.controller.event	92,5 %	74	6	80
com.scarcolo.eventour.model	90,6 %	77	8	85
com.scarcolo.eventour.service.user	89,2 %	1.393	169	1.562
com.scarcolo.eventour.model.ticketisp	88,7 %	118	15	133
com.scarcolo.eventour.service.ticketisp	87,0 %	282	42	324
com.scarcolo.eventour.service.booking	86,7 %	767	118	885
com.scarcolo.eventour.service.admin	85,8 %	546	90	636
com.scarcolo.eventour.service.manager	82,9 %	573	118	691
com.scarcolo.eventour.model.request	82,6 %	38	8	46
com.scarcolo.eventour.service.event	81,8 %	964	215	1.179
com.scarcolo.eventour.model.user	81,6 %	351	79	430
com.scarcolo.eventour.model.manager	71,9 %	496	194	690
com.scarcolo.eventour.model.booking	71,7 %	142	56	198
com.scarcolo.eventour.model.admin	66,8 %	280	139	419
com.scarcolo.eventour.model.event	63,5 %	575	331	906
com.scarcolo.eventour.functions	63,4 %	822	474	1.296
com.scarcolo.eventour	37,5 %	3	5	8

Figura 5.7 Copertura test JUnit

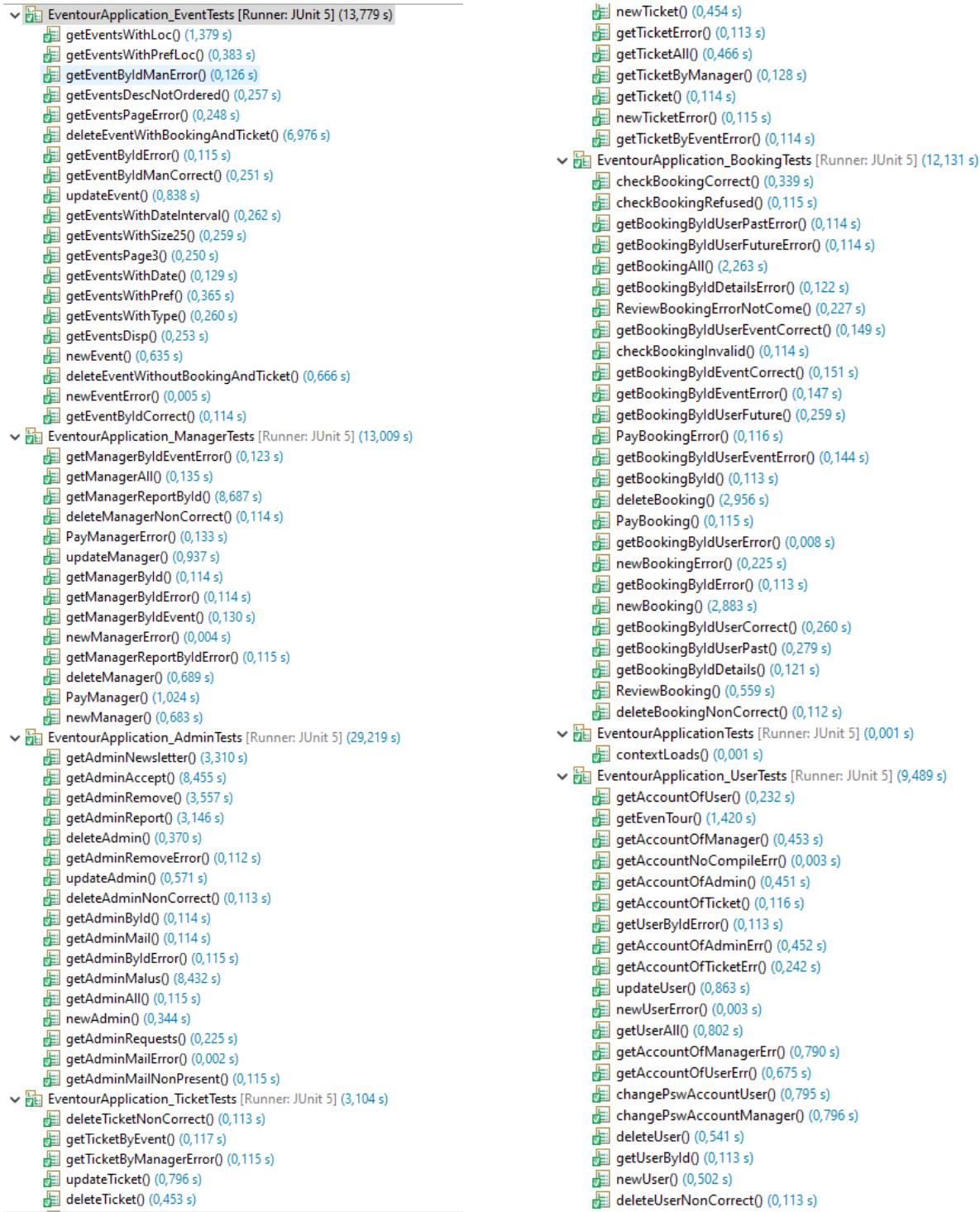


Figura 5.8 Test effettuati e passati

Sviluppi futuri

Il progetto, per come è stato sviluppato, si trova in una fase avanzata se si guarda alle caratteristiche su cui si basa. Possibili sviluppi futuri riguardano senz'altro l'algoritmo, il quale può essere ulteriormente raffinato e ottimizzato, introducendo differenti parametri che permetteranno la generazione di molteplici soluzioni con la conseguente scelta della migliore secondo i gusti personali dell'utente.

Un ulteriore miglioramento può essere svolto sulla sicurezza dei dati salvati nel database, introducendo un sistema di crittografia che permetta il salvataggio delle password non in chiaro.

Un ulteriore sviluppo è il completamento delle funzionalità “mock” quali registrazioni reali di utenti e manager e pagamenti realmente attivi e l'invio di newsletter programmate, senza necessità da parte dell'amministratore di inviarla manualmente.