



Scuola Politecnica e delle Scienze di Base  
Corso di Laurea Magistrale in Ingegneria Informatica

*Elaborato di Software Architecture  
Design - WishlistApp*

Anno Accademico 2021/2022

Simone Staiano M63001037  
Luca Vivenzo M63001072

# Contents

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Unified Process . . . . .	1
1.2	Prima iterazione . . . . .	3
1.3	Seconda iterazione . . . . .	5
1.4	Tecnologie di supporto . . . . .	6
1.4.1	Bootstrap . . . . .	6
1.4.2	IntelliJ IDEA . . . . .	6
1.4.3	Google Chrome . . . . .	7
1.4.4	Microsoft Teams . . . . .	7
1.4.5	PgAdmin . . . . .	8
1.4.6	Spring Boot . . . . .	8
1.4.7	Visual Paradigm . . . . .	9
1.4.8	GitKraken . . . . .	9
1.4.9	Postman . . . . .	10
<b>2</b>	<b>Visione</b>	<b>11</b>
2.1	Descrizione testuale dei requisiti . . . . .	11
2.2	Obiettivi . . . . .	12
2.3	Portatori di interesse . . . . .	12

2.4	Glossario . . . . .	13
2.5	Stima dei tempi . . . . .	13
<b>3</b>	<b>Requisiti funzionali e analisi di dominio</b>	<b>17</b>
3.1	Casi d'uso . . . . .	18
3.1.1	Cerca utente . . . . .	18
3.1.2	Crea evento . . . . .	19
3.1.3	Crea Wishlist . . . . .	20
3.1.4	Elimina amico . . . . .	21
3.1.5	Elimina evento . . . . .	22
3.1.6	Elimina wishlist . . . . .	22
3.1.7	Gestisci le richieste di amicizia . . . . .	23
3.1.8	Gestisci regali della wishlist . . . . .	24
3.1.9	Invia richiesta di amicizia . . . . .	25
3.1.10	Invita amici ad un evento . . . . .	26
3.1.11	Login . . . . .	27
3.1.12	Registrati . . . . .	28
3.1.13	Segna regalo come acquistato . . . . .	29
3.1.14	Visualizza eventi degli amici . . . . .	30
3.1.15	Visualizza evento personale . . . . .	31
3.1.16	Visualizza lista amici . . . . .	32
3.1.17	Visualizza wishlist degli amici . . . . .	32
3.1.18	Visualizza wishlist personale . . . . .	33
3.2	Context Diagram . . . . .	34
3.3	System Domain Model . . . . .	34
3.4	Entity-Relationship Diagram . . . . .	35

<b>4 Requisiti non funzionali</b>	<b>36</b>
4.1 Scenari . . . . .	37
4.1.1 Modifiability . . . . .	37
4.1.2 Robustness . . . . .	38
4.1.3 Security . . . . .	39
4.1.4 Usability . . . . .	40
<b>5 Architettura del sistema</b>	<b>42</b>
5.1 Front-end . . . . .	42
5.1.1 Thin client . . . . .	43
5.2 Back-end . . . . .	44
5.2.1 Spring . . . . .	45
5.2.2 Database . . . . .	51
5.3 Package Diagram . . . . .	51
5.3.1 Entity Class Diagram . . . . .	51
5.3.2 Repository Class Diagram . . . . .	52
5.3.3 Service Class Diagram . . . . .	53
5.3.4 Controller Class Diagram . . . . .	54
5.4 Component Diagram . . . . .	54
5.5 API . . . . .	55
5.5.1 Evento . . . . .	55
5.5.2 Wishlist . . . . .	58
5.5.3 AppUser . . . . .	62
5.5.4 Friendship . . . . .	63
5.5.5 Authentication . . . . .	65

<b>6 Diagrammi comportamentali</b>	<b>67</b>
6.1 Sequence Diagram . . . . .	67
6.1.1 addPresent . . . . .	68
6.1.2 getAllWishlists . . . . .	68
6.1.3 login . . . . .	69
6.1.4 register . . . . .	70
6.1.5 createWishlist . . . . .	71
6.1.6 removeWishlist . . . . .	72
6.1.7 createEvent . . . . .	73
6.1.8 removePresent . . . . .	74
6.1.9 deleteEvent . . . . .	75
6.1.10 getFriendsWishlists . . . . .	76
6.1.11 search . . . . .	76
6.1.12 getFriendsWishlist . . . . .	77
6.1.13 getWishlist . . . . .	78
6.1.14 getWishlistOfAFriend . . . . .	79
6.1.15 buy . . . . .	80
6.1.16 getEvent . . . . .	81
6.1.17 listEvents . . . . .	82
6.1.18 getInvitations . . . . .	82
6.1.19 getInvitationsByAFriend . . . . .	83
6.1.20 getInvitation . . . . .	84
6.1.21 invite . . . . .	84
6.1.22 addFriend . . . . .	85
6.1.23 deleteFriend . . . . .	86
6.1.24 getFriends . . . . .	86

6.1.25	getPendingRequests . . . . .	87
6.1.26	setFriendship . . . . .	88
6.2	Flowchart . . . . .	90
6.3	Activity Diagram . . . . .	92
<b>7</b>	<b>Rilascio e Testing</b>	<b>95</b>
7.1	Deployment diagram . . . . .	95
7.1.1	Install view . . . . .	96
7.2	Testing d'unità . . . . .	97
7.3	Testing d'integrazione . . . . .	110
7.3.1	Test backend . . . . .	110
7.3.2	Test intero sistema tramite GUI . . . . .	116
<b>8</b>	<b>Manuale Utente</b>	<b>117</b>
8.1	Pagina di login (login.html) . . . . .	117
8.2	Navbar . . . . .	118
8.3	Pagina Home (index.html) . . . . .	119
8.4	Le mie Wishlist (myWishlists.html) . . . . .	119
8.5	Wishlist (wishlist.html?id=) . . . . .	120
8.6	I miei eventi (myEvents.html) . . . . .	121
8.7	Il tuo Evento (event.html?id=) . . . . .	122
8.8	Eventi degli amici (events.html) . . . . .	124
8.9	Amici (friends.html) . . . . .	125
8.10	Richieste di amicizia (friendsrequests.html) . . . . .	125
8.11	Eventi degli amici (events.html) . . . . .	126
8.12	Wishlist di un amico (friendWishlist.html) . . . . .	126
8.13	Evento di un amico (friendEvent.html) . . . . .	127

## CONTENTS

---

8.14 Il mio profilo (myProfile.html) . . . . .	128
8.15 Profilo amico (profile.html?id=) . . . . .	129

# Chapter 1

## Introduzione

WishlistApp è un'applicazione web il cui scopo è offrire agli utenti la possibilità di poter organizzare e gestire, in maniera semplice e veloce, delle liste desideri con i propri amici. L'applicazione è dotata di conseguenza anche di funzionalità social, in quanto gli utenti possono aggiungere amici alla propria rete, visualizzare le loro liste regalo, organizzare eventi insieme e invitare amici agli stessi.

### 1.1 Unified Process

WishlistApp è stata progettata e sviluppata seguendo le linee guida definite dal processo software Unified Process Model (UP). Si è fatta questa scelta per diverse motivazioni. In primis, si è ritenuto necessario adottare un processo di tipo iterativo che permetesse di realizzare un'applicazione robusta a eventuali modifiche dei requisiti funzionali e che fornisse dei feed-

back per il team di sviluppo sullo stato del progetto con frequenza regolare. Tali riscontri sono stati fondamentali a causa della necessità di utilizzare tecnologie nuove al gruppo. Fondamentale è stata anche la scelta di un modello di sviluppo che fosse guidato dalla valutazione del rischio, così da prediligere innanzitutto le funzionalità di base e, solo in una fase successiva, considerare quelle accessorie. Considerando inoltre che una web application è per sua natura fortemente interattiva, si è preferito l'utilizzo di un modello guidato dai casi d'uso e che ponesse fortemente l'attenzione sulle sequenze delle azioni e sulle modalità di interazione tra gli utenti finali e il sistema. Infine, si è deciso di seguire un processo che privilegiasse e supportasse il linguaggio UML per la definizione e realizzazione dei diagrammi, così da rendere la documentazione il più possibile uniforme e vicina agli standard. Il processo UP è risultato essere adatto a questi scopi. Lo sviluppo incrementale è stato favorito dalla suddivisione del lavoro di progetto in due iterazioni dalla durata di 19 giorni ciascuna. Per ogni iterazione, lo scopo è quello di produrre un incremento valido, ovvero un sottosistema (o sistema nell'iterazione finale) eseguibile, testato, correttamente funzionante e aderente ai requisiti specificati. Con questa tecnica si ha la possibilità di procedere per raffinamenti sia per quanto riguarda le specifiche dei requisiti sia per la realizzazione dell'architettura del sistema. Poiché si è quindi adottato un modello evolutivo e non prototipale, prima di tutto è stato realizzato nella prima iterazione, un nucleo di base dell'architettura del sistema e successivamente, nell'iterazione finale, si è modificato e raffinato lo stesso con

lo sviluppo delle funzionalità meno critiche. Si sono quindi tenuti due workshop di iteration planning per stabilire gli obiettivi dell'iterazione successiva insieme a dei workshop riepilogativi durante le iterazioni, con il fine di analizzare passo passo gli output prodotti. A prescindere da questi workshop, chiaramente il team si è confrontato quotidianamente per scopi di sincronizzazione delle attività e di collaborazione su particolari compiti (ad esempio stesura della documentazione, test di alcune funzionalità, etc.).

## 1.2 Prima iterazione

Durante la prima iterazione si è deciso chiaramente di dedicarsi alle attività più critiche e importanti dell'applicazione. Per prima cosa si è analizzato il dominio applicativo, estraendo le entità concettuali del progetto e individuando i requisiti funzionali e non funzionali. Si è quindi proceduto a progettare lo scheletro architetturale del sistema e implementare le funzionalità relative al login, all'autenticazione, all'autorizzazione, alla creazione delle wishlist e degli eventi. Inoltre, per lo sviluppo della web app, si è seguito il principio del "security by design", ovvero le questioni e le scelte relative alla security sono state affrontate sin dalle prime fasi di progettazione. I documenti prodotti a valle di questa fase sono:

- Il documento di visione
- Use Case Diagram
- Gli scenari di alto livello legati ai casi d'uso implementati durante la

prima iterazione

- Entity Relationship Diagram
- Le viste architetturali (Class Diagram, Package Diagram, Component Diagram)
- Sequence Diagram per le funzionalità di autenticazione e autorizzazione

I casi d'uso sono i seguenti:

1. Registrazione
2. Login
3. Crea wishlist
4. Aggiungi regalo alla wishlist
5. Elimina wishlist
6. Crea evento
7. Elimina evento

Per la descrizione degli stessi si rimanda al capitolo relativo ai requisiti funzionali dell'applicazione. Durante la prima iterazione sono stati inoltre svolti i primi test di unità e integrazione.

## 1.3 Seconda iterazione

Durante la seconda iterazione l'obiettivo è stato quello di estendere l'architettura del software con l'aggiunta delle funzionalità accessorie, come ad esempio la segnalazione di acquisto di un regalo, oltre poi a tutte le funzionalità "social", legate alla ricerca degli utenti e alle amicizie. I documenti prodotti a valle di questa fase sono:

- Sequence Diagram di dettaglio relativi ai casi d'uso implementati nella prima iterazione
- Gli scenari di alto livello relativi ai casi d'uso della seconda iterazione
- Sequence Diagram di dettaglio relativi ai casi d'uso implementati nella seconda iterazione.
- Il Diagramma di Deployment e la Install View.

I casi d'uso sono della seconda iterazione sono i seguenti:

- Gestione eventi
- Gestione regali della wishlist
- Segna regalo come acquistato
- Ricerca utente (per username)
- Gestione richieste di amicizia
- Visualizza lista amici

- Invita amici ad un evento

Per la descrizione degli stessi si rimanda al capitolo relativo ai requisiti funzionali dell'applicazione. Sono stati infine svolti i test di unità e integrazione finali ed è stata completata la documentazione del progetto.

## 1.4 Tecnologie di supporto

### 1.4.1 Bootstrap

Bootstrap è un framework HTML, CSS e Javascript per lo sviluppo di applicazioni web, in particolar modo per il front-end. Mette a disposizione componenti per l'interfaccia grafica come pulsanti, form di compilazione, navbar e cards.



Figure 1.1: Bootstrap

### 1.4.2 IntelliJ IDEA

IntelliJ IDEA è un ambiente di sviluppo integrato (IDE) per il linguaggio di programmazione Java, sviluppato da JetBrains e disponibile sia in licenza free sia in licenza commerciale. In questo contesto è stata utilizzata la licenza

gratuita. L'IDE è stato fondamentalmente utilizzato per lo sviluppo del back-end. La scelta di tale ambiente è dovuta alla familiarità del team con con lo stesso, altre alle numerose funzionalità e utilities offerte.



Figure 1.2: IntelliJ IDEA

### 1.4.3 Google Chrome

Google Chrome è un browser web sviluppato da Google, basato sul motore di rendering Blink e sul browser Chromium. È stato utilizzato dal team per testare la User Interface e le funzionalità del client.



Figure 1.3: Google Chrome

### 1.4.4 Microsoft Teams

Microsoft Teams è una piattaforma di comunicazione e collaborazione facente parte del pacchetto Microsoft Office 365. Supporta le videoconferenze e la condivisione simultanea di file multimediali e dello schermo. È stata utilizzata dal team per pianificare le riunioni e condividere materiale.



Figure 1.4: Microsoft Teams

#### 1.4.5 PgAdmin

pgAdmin è un software open-source che consente di amministrare in modo semplice un database PostgreSQL, tramite interfaccia grafica.



Figure 1.5: pgAdmin

#### 1.4.6 Spring Boot

Spring è un framework open-source per lo sviluppo di applicazioni web che supporta il linguaggio Java. In particolare si è utilizzato Spring Boot, un micro-framework derivato da Spring. Per i dettagli sull'architettura del framework si rimanda alle sezioni successive.



Figure 1.6: Spring Boot

### 1.4.7 Visual Paradigm

Visual Paradigm è un tool di supporto alla progettazione dei diagrammi UML. È stato utilizzato dal team per la realizzazione dei diagrammi. Disponibile sia in licenza free (Community Edition) sia a pagamento (Premium Edition), in questo caso è stata utilizzata la Community Edition.



Figure 1.7: Visual Paradigm

### 1.4.8 GitKraken

GitKraken è una suite di developer tools. Tra i vari strumenti, fornisce un client Git con un’interfaccia grafica molto intuitiva e dettagliata. È stato utilizzato dal team per lavorare contemporaneamente, da remoto, sullo stesso codice, condividendo i file e le reciproche modifiche agli stessi.



Figure 1.8: GitKraken

### 1.4.9 Postman

Postman è una piattaforma API per sviluppatori, usata per progettare, costruire e testare le proprie API. È stata usata per testare le API prima dello sviluppo del client.



**POSTMAN**

Figure 1.9: Postman

# Chapter 2

## Visione

### 2.1 Descrizione testuale dei requisiti

Una wishlist è letteralmente una "lista di desideri", un insieme di regali che una persona avrebbe piacere di ricevere. Ogni lista è quindi costituita da un insieme di regali diversi e per ognuno è previsto un link che porta direttamente all'acquisto dello stesso. Ogni utente registrato all'applicazione può condividere le proprie wishlist con tutti i suoi amici. Inoltre, è prevista la possibilità di creare degli eventi a cui invitare gli amici (tutti o una parte).

Ad ogni evento, oltre alle informazioni relative alla data e il luogo, è associata una wishlist, così che per tutti gli invitati sia semplice sapere cosa l'organizzatore desideri che gli venga regalato. Nel caso in cui un invitato decida di comprare uno dei regali, può segnare lo stesso come "acquistato" direttamente sulla wishlist, così che tutti lo possano vedere. Chiaramente le wishlist associate agli eventi saranno visibili esclusivamente agli invitati.

Tutte le funzionalità dell'applicazione sono destinate agli utenti registrati.

## 2.2 Obiettivi

WishlistApp ha l'obiettivo di fornire agli utenti la possibilità di:

- condividere le proprie "liste di desideri" con gli amici, così che ognuno sappia quali sono i regali desiderati dai propri amici.
- semplificare tutte le classiche dinamiche di scelta di un regalo che spesso si verificano tra gruppi di amici.
- invitare in modo facile e veloce un insieme di amici ad un proprio evento, fornendo anche indicazioni sullo stesso come data e luogo.

## 2.3 Portatori di interesse

Sulla base degli obiettivi sopracitati, l'utente tipo indicato per WishlistApp può essere una qualsiasi persona che voglia condividere la propria lista di desideri con i propri amici. Inoltre, l'applicazione risulta particolarmente adatta se utilizzata da un gruppi di amici che condividono spesso eventi insieme, così da potersi scambiare in maniera facile e veloce inviti e informazioni sugli stessi. L'applicazione è destinata esclusivamente agli utenti registrati, che possono:

1. accedere al sistema;

2. creare, modificare, eliminare le proprie wishlist ed eventi;
3. aggiungere amici alla loro rete per visualizzarne i contenuti.

## 2.4 Glossario

Analizzando la descrizione testuale, sono stati estrapolati i seguenti termini, che corrispondono alle entità concettuali del dominio:

Termino	Significato
Present	un regalo, caratterizzato da un link per l'acquisto.
Wishlist	una collezione di regali.
Event	un avvenimento futuro, a cui è associata una wishlist.
User	una persona che può organizzare eventi e creare wishlist, o partecipare ad eventi e visualizzare wishlist di altri utenti. Può inviare e ricevere richieste di amicizia da altri utenti.

## 2.5 Stima dei tempi

Per stimare i tempi di realizzazione del progetto, si è scelto di utilizzare il metodo degli Use Case Points. Per prima cosa si vanno quindi a calcolare gli Unadjusted Use Case Weight (UUCW). Si va quindi ad associare un diverso peso ad ogni Use Case in base al numero delle transazioni che presenta. In particolare, per fare ciò, si fa riferimento alla seguente tabella:

Complessità del caso d'uso	Transazioni	Peso
Bassa	3 o meno	5
Media	da 4 a 7	10
Alta	più di 7	15

Si sono raccolti dunque in una tabella i risultati delle stime effettuate per il conteggio dei pesi dei casi d'uso. Vengono omessi i casi d'uso di estensione in quanto conteggiati con i casi d'uso base.

Caso d'uso	Transazioni	Peso
Cerca utente	3	5
Crea evento	6	10
Elimina amico	3	5
Elimina evento	3	5
Elimina wishlist	3	5
Gestisci le richieste di amicizia	3	5
Gestisci regali della wishlist	3	5
Invia richiesta di amicizia	3	5
Invita amici ad un evento	3	5
Login	3	5
Registrati	3	5
Segna regalo come acquistato	3	5
Visualizza eventi degli amici	6	10
Visualizza evento personale	6	10
Visualizza lista amici	3	5
<b>Totale</b>		<b>90</b>

Si ha quindi un totale di UUCW pari a 90. Si può quindi proseguire con il conteggio degli Unadjusted Actor Weight (UAW). In questo caso si fa riferimento alla seguente tabella.

Tipo di attore	Descrizione	Peso
Semplice	Sistema esterno che interagisce tramite API	1
Medio	Sistema esterno che interagisce attraverso un protocollo o una persona che interagisce attraverso un'interfaccia testuale	2
Complesso	Una persona che interagisce attraverso una GUI	3

Nel sistema in esame sono presenti due tipi di attori: Utente ed Utente

Registrato. Essi rappresentano degli attori di tipo complesso in quanto sono persone che interagiscono col nostro sistema attraverso un’interfaccia grafica. In realtà però, questi due utenti interagiscono con il sistema allo stesso modo, quindi è possibile considerarli come un unico attore. Si ha quindi un UAW=3.

Si può quindi ora calcolare l’Unadjusted Use Case Point (UUCP) come somma degli UUCW e UAW. Abbiamo quindi UUCP=93.

Si prosegue ora con il calcolo dei fattori di aggiustamento TCF ed EF con cui calcolare il valore degli UCP. Nella seguente tabella è riportato il calcolo del TFactor per il sistema in esame.

Fattore	Descrizione	Peso	Valutazione	Prodotto
T1	Sistema distribuito	2	0	0
T2	Obiettivi di performance	2	1	2
T3	Efficienza per l’utente finale	1	2	2
T4	Complessità di elaborazione	1	1	1
T5	Riusabilità del codice	1	1	1
T6	Facilità di installazione	0.5	2	1
T7	Facilità d’uso	0.5	2	1
T8	Portabilità	2	1	2
T9	Modificabilità	1	2	2
T10	Calcolo parallelo	1	0	0
T11	Sicurezza	1	2	2
T12	Accesso per terze parti	1	0	0
T13	Necessità di training per l’utente finale	1	0	0
TFactor				14

Quindi:

$$TCF = 0.6 + (0.01 * TFactor) = 0.74$$

Nella seguente tabella, invece, è riportato il calcolo dell'EFactor.

Fattore	Descrizione	Peso	Valutazione	Prodotto
E1	Familiarità col processo di sviluppo	1.5	4	6
E2	Esperienza applicativa	0.5	4	2
E3	Esperienza object oriented	1	4	4
E4	Capacità di analisi	0.5	4	2
E5	Motivazione	1	5	5
E6	Stabilità dei requisiti	2	5	10
E7	Personale part-time	-1	0	0
E8	Complessità del linguaggio di programmazione	-1	2	-2
EFactor				27

Quindi:

$$EF = 1.4 + (-0.03 * EFactor) = 0.59$$

Si possono quindi calcolare gli UCP complessivi come:

$$UCP = UUCP * TCF * EF = 93 * 0.74 * 0.59 = 41$$

Andando a considerare ogni UCP pari a 15 ore di lavoro, si ha un totale di 615 ore di lavoro. Distribuendo queste ore tra i due componenti del gruppo e considerando le giornate di lavoro di 8 ore, è stata prevista una durata del progetto di circa 38 giorni.

# Chapter 3

## Requisiti funzionali e analisi di dominio

In questo capitolo del documento si presentano i risultati dell'analisi del dominio: il diagramma UML dei casi d'uso e gli scenari associati ad essi; il modello di dominio del sistema, che mostra le entità concettuali del dominio e le associazioni da cui sono legate; il modello dei dati, esplicitato attraverso il diagramma Entity-Relationship.

## 3.1 Casi d'uso

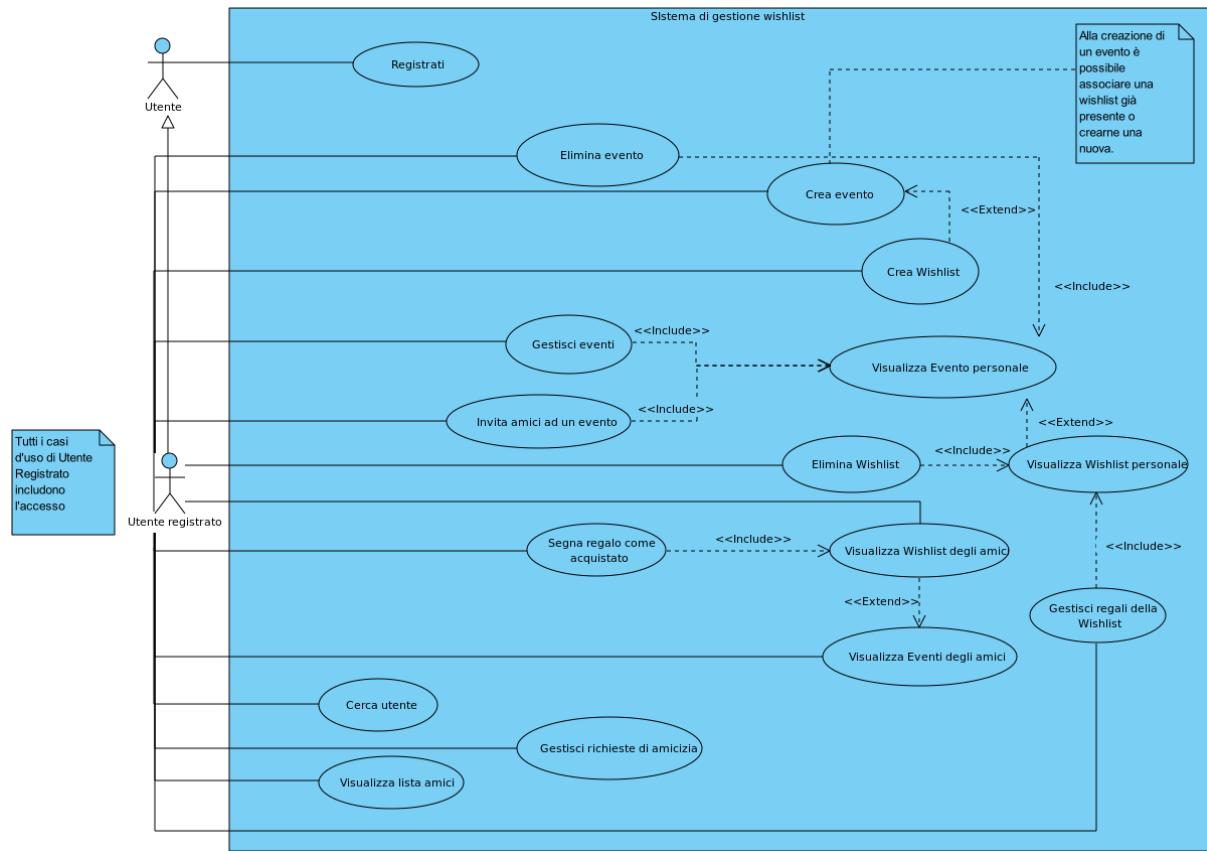


Figure 3.1: Use Case Diagram

Sono presentati di seguito gli scenari associati ai singoli casi d'uso individuati.

### 3.1.1 Cerca utente

- Nome: Cerca utente
- Scopo: permettere all'utente di cercare un altro utente all'interno dell'applicazione, conoscendone (in toto o in parte) l'username.
- Precondizioni: l'utente è registrato, ha effettuato l'accesso ed ha a

disposizione un token valido. L'utente conosce il nome (o parte del nome) dell'utente che vuole cercare.

- Flusso principale:
  1. L'utente inserisce il nome dell'altro utente nella Navbar.
  2. Il sistema trova l'utente che contiene la stringa inserita.
  3. Il sistema restituisce la lista degli utenti trovati.
- Flusso secondario: Il sistema non trova alcun utente che soddisfi la stringa inserita e non restituisce alcuna lista.
- Attori: utente registrato
- Casi d'uso inclusi: Login.

### 3.1.2 Crea evento

- Nome: Crea evento
- Scopo: permette ad un utente registrato di creare un proprio evento.
- Precondizioni: l'utente è registrato, ha effettuato l'accesso ed ha a disposizione un token valido.
- Flusso principale:
  1. L'utente inserisce il nome dell'evento, la descrizione, la data, il luogo. Seleziona quindi (se vuole) la wishlist da associare.

2. Il sistema verifica che il token fornito dall'utente sia valido.
  3. Il sistema crea il nuovo evento associato all'utente.
- Estensioni: "Crea Wishlist"
    - 3a. L'utente inserisce il nome della wishlist e una descrizione.
    - 3b. Il sistema verifica che il token fornito dall'utente sia valido.
    - 3c. Il sistema crea la nuova wishlist associata all'utente.
- Post-condizioni: l'evento è stato creato ed è stato associato all'utente.
  - Attori: utente registrato.
  - Casi d'uso inclusi: Login.

### 3.1.3 Crea Wishlist

- Nome: Crea Wishlist
- Scopo: permette ad un utente registrato di creare una propria wishlist.
- Precondizioni: l'utente è registrato, ha effettuato l'accesso ed ha a disposizione un token valido.
- Flusso principale:
  1. L'utente inserisce il nome della wishlist e una descrizione.
  2. Il sistema verifica che il token fornito dall'utente sia valido.
  3. Il sistema crea la nuova wishlist associata all'utente.

- Post-condizioni: la wishlist è stata creata ed è stata associata all'utente.
- Attori: utente registrato
- Casi d'uso inclusi: Login.
- Casi d'uso estesi: Crea evento

### 3.1.4 Elimina amico

- Nome: Elimina amico.
- Scopo: permette ad un utente registrato di eliminare un proprio amico.
- Precondizioni: l'utente è registrato, ha effettuato l'accesso ed ha a disposizione un token valido. L'utente è amico di almeno un altro utente.
- Flusso principale:
  1. L'utente richiede di eliminare un amico tramite GUI.
  2. il sistema elabora la richiesta, verificando che i due utenti siano effettivamente amici.
  3. il sistema elimina l'amicizia tra i due utenti e li aggiorna.
- Post-condizioni: l'amicizia è stata eliminata.
- Attori: utente registrato.
- Casi d'uso inclusi: Login.

### 3.1.5 Elimina evento

- Nome: Elimina evento.
- Scopo: permettere ad un utente registrato di eliminare un proprio evento.
- Precondizioni: l'utente è registrato, ha effettuato l'accesso ed ha a disposizione un token valido. L'utente è l'organizzatore dell'evento.
- Flusso principale:
  1. L'utente richiede di eliminare l'evento tramite GUI.
  2. il sistema verifica che l'utente sia l'organizzatore dell'evento.
  3. il sistema elimina l'evento e aggiorna l'utente.
- Post-condizioni: l'evento è stato eliminato.
- Attori: utente registrato.
- Casi d'uso inclusi: Login, Visualizza evento personale.

### 3.1.6 Elimina wishlist

- Nome: Elimina wishlist.
- Scopo: permettere ad un utente registrato di eliminare una propria wishlist.

- Precondizioni: l'utente è registrato, ha effettuato l'accesso ed ha a disposizione un token valido. L'utente è proprietario della wishlist.
- Flusso principale:
  1. L'utente richiede di eliminare la wishlist tramite GUI.
  2. Il sistema verifica che l'utente sia proprietario della wishlist.
  3. Il sistema elimina la wishlist e aggiorna l'utente.
- Post-condizioni: la wishlist è stata eliminata.
- Attori: utente registrato.
- Casi d'uso inclusi: Login, Visualizza wishlist personale.

### 3.1.7 Gestisci le richieste di amicizia

- Nome: Gestisci le richieste di amicizia
- Scopo: permettere ad un utente di visualizzare le richieste di amicizia ricevute, dandogli la possibilità di accettare o rifiutare le stesse.
- Precondizioni: l'utente è registrato, ha effettuato l'accesso ed ha a disposizione un token valido. L'utente ha ricevuto almeno una richiesta di amicizia.
- Flusso principale: Accetta richiesta

1. L'utente, cliccando sul bottone "Richieste di amicizia" presente nella GUI, richiede di visualizzare la lista delle richieste di amicizia ricevute.
  2. L'utente clicca sul bottone "Accetta".
  3. Il sistema elabora la richiesta d'amicizia rendendola attiva. Aggiorna i due utenti.
- Flusso secondario: Rifiuta richiesta
    1. L'utente, cliccando sul bottone "Richieste di amicizia" presente nella GUI, richiede di visualizzare la lista delle richieste di amicizia ricevute.
    2. L'utente clicca sul bottone "Rifiuta".
    3. Il sistema elimina la richiesta d'amicizia. Aggiorna i due utenti.
  - Post-condizioni (1): i due utenti sono amici.
  - Post-condizioni (2): La richiesta di amicizia viene rifiutata ed eliminata.
  - Attori: utente registrato.
  - Casi d'uso inclusi: Login.

### 3.1.8 Gestisci regali della wishlist

- Nome: Gestisci regali della wishlist

- Scopo: permette all'utente di effettuare operazioni relative ai regali presenti nella wishlist, come ad esempio l'aggiunta o la rimozione.
- Precondizioni: l'utente è registrato, ha effettuato l'accesso ed ha a disposizione un token valido. Ha creato almeno una wishlist.
- Flusso principale:
  1. L'utente seleziona la wishlist di cui è proprietario, compila il form relativo ai dati del regalo, inserendo il nome del regalo, la descrizione e il link per l'acquisto e clicca sul bottone "Aggiungi Regalo".
  2. Il sistema elabora la richiesta.
  3. Il sistema crea il regalo. Aggiorna la wishlist.
- Post-condizioni: il regalo è stato creato e salvato nel database. La wishlist presenta un regalo in più.
- Attori: utente registrato.
- Casi d'uso inclusi: Login, Visualizza wishlist personale.

### 3.1.9 Invia richiesta di amicizia

- Nome: Invia richiesta di amicizia
- Scopo: permettere ad un utente di inviare una richiesta di amicizia.

- Precondizioni: l'utente è registrato, ha effettuato l'accesso ed ha a disposizione un token valido.
- Flusso principale:
  1. L'utente cerca nel sistema l'utente che vuole aggiungere.
  2. Clicca sul relativo bottone "Aggiungi amico".
  3. Il sistema elabora la richiesta d'amicizia inviandola all'altro utente.
- Post-condizioni: la richiesta d'amicizia è stata inviata all'altro utente.
- Attori: utente registrato.
- Casi d'uso inclusi: Login, Cerca utente.

### 3.1.10 Invita amici ad un evento

- Nome: Invita amici ad un evento
- Scopo: permette ad un utente di invitare degli amici ad un proprio evento.
- Precondizioni: l'utente è registrato, ha effettuato l'accesso ed ha a disposizione un token valido. L'utente è organizzatore di un evento. L'utente è amico almeno di un altro utente.
- Flusso principale:

1. L'utente seleziona l'evento a cui vuole aggiungere degli invitati, clicca sul bottone "Aggiungi invitati" e dal menù, seleziona gli amici che desidera invitare per poi cliccare su "Aggiungi invitati".
  2. Il sistema elabora la richiesta.
  3. Invita gli amici selezionati all'evento. Aggiorna l'evento e tutti gli utenti invitati.
- Post-condizioni: gli amici selezionati sono stati invitati all'evento.
  - Attori: utente registrato.
  - Casi d'uso inclusi: Login, Visualizza evento personale.

### 3.1.11 Login

- Nome: Login
- Scopo: permettere ad un utente registrato di accedere all'applicazione.
- Precondizioni: l'utente è già registrato.
- Flusso principale:
  1. L'utente fornisce Username e Password al sistema attraverso la pagina di accesso.
  2. Il sistema verifica la validità e corrispondenza delle credenziali.
  3. Il sistema autentica l'utente, gli restituisce un JWT Token e un messaggio di login effettuato.

- Flusso secondario: se le credenziali non sono corrette, il sistema restituisce un messaggio di errore.
- Post-condizioni: l'utente ha effettuato l'accesso e ha ricevuto un token con il quale è autorizzato a esplorare le funzionalità dell'applicazione.
- Attori: utente registrato.

### 3.1.12 Registrati

- Nome: Registrati
- Scopo: permettere ad un utente di registrarsi.
- Precondizioni: l'utente non risulta registrato.
- Flusso principale:
  1. L'utente inserisce Username, Email e Password nella pagina di Registrazione.
  2. Il sistema controlla che non ci sia un altro utente associato alla stessa e-mail.
  3. Il sistema crea l'account e restituisce un messaggio di avvenuta registrazione.
- Flusso secondario: se al punto 2 il controllo fallisce, la registrazione non va a buon fine e viene restituito un messaggio di errore.

- Post-condizioni: l'account è stato memorizzato e l'utente risulta registrato.
- Attori: utente non registrato.

### 3.1.13 Segna regalo come acquistato

- Nome: Segna regalo come acquistato
- Scopo: permette all'utente di segnare un regalo come "acquistato", così che tutti gli altri utenti (incluso il creatore della wishlist) sappiano che quello specifico regalo è stato acquistato.
- Precondizioni: l'utente è registrato, ha effettuato l'accesso ed ha a disposizione un token valido. L'utente è amico/invitato all'evento del creatore della wishlist. La wishlist contiene almeno un regalo.
- Flusso principale:
  1. L'utente seleziona la wishlist, visualizzandone i regali.
  2. Il sistema elabora la richiesta.
  3. Il sistema segna il regalo come acquistato aggiornando lo stato del regalo e della wishlist.
- Attori: utente registrato.
- Casi d'uso inclusi: Login, Visualizza wishlist degli amici.

### 3.1.14 Visualizza eventi degli amici

- Nome: Visualizza eventi degli amici
- Scopo: Permettere ad un utente di visualizzare l'evento di un suo amico.
- Precondizioni: l'utente è registrato, ha effettuato l'accesso ed ha a disposizione un token valido. L'utente deve essere stato invitato ad almeno un evento.
- Flusso principale:
  1. L'utente accede alla pagina in cui sono presenti gli eventi degli amici, cliccando sul bottone "Eventi" presente nella GUI.
  2. L'utente seleziona lo specifico evento che vuole visualizzare, cliccandoci.
  3. Il sistema elabora la richiesta e restituisce l'evento.
- Estensioni: "Visualizza wishlist di un amico"
  - 3a. L'utente accede alla pagina in cui sono presenti le wishlist degli amici.
  - 3b. L'utente seleziona la specifica wishlist che vuole visualizzare, cliccandoci.
  - 3c. Il sistema elabora la richiesta e restituisce la wishlist.
- Attori: utente registrato.

- Casi d'uso inclusi: Login.

### 3.1.15 Visualizza evento personale

- Nome: Visualizza evento personale
- Scopo: permette ad un utente di visualizzare un evento di cui è organizzatore.
- Precondizioni: l'utente è registrato, ha effettuato l'accesso ed ha a disposizione un token valido. L'utente è organizzatore dell'evento che vuole visualizzare.
- Flusso principale:
  1. L'utente accede alla pagina in cui sono presenti tutti gli eventi di cui è organizzatore e seleziona l'evento che vuole visualizzare.
  2. Il sistema elabora la richiesta.
  3. Il sistema restituisce l'evento all'utente.
- Estensioni:
  - 3a. Una volta visualizzato l'evento, l'utente può decidere di visualizzare anche la wishlist associata, selezionandola.
  - 3b. Il sistema elabora la richiesta.
  - 3c. Il sistema restituisce la wishlist all'utente.
- Attori: utente registrato.

- Casi d'uso inclusi: Login.

### 3.1.16 Visualizza lista amici

- Nome: Visualizza lista amici
- Scopo: permettere ad un utente di visualizzare la lista dei propri amici.
- Precondizioni: l'utente è registrato, ha effettuato l'accesso ed ha a disposizione un token valido.
- Flusso principale:
  1. L'utente, cliccando sul bottone "Amici" presente nella GUI, richiede di visualizzare la lista degli amici.
  2. Il sistema elabora la richiesta.
  3. Il sistema restituisce la lista degli amici.
- Attori: utente registrato.
- Casi d'uso inclusi: Login.

### 3.1.17 Visualizza wishlist degli amici

- Nome: Visualizza wishlist degli amici
- Scopo: permette all'utente di visualizzare le wishlist degli amici.
- Precondizioni: l'utente è registrato, ha effettuato l'accesso ed ha a disposizione un token valido. L'utente ha almeno un amico.

- Flusso principale:
  1. L'utente accede alla pagina in cui sono presenti le wishlist degli amici.
  2. L'utente seleziona la specifica wishlist che vuole visualizzare, cliccandoci.
  3. Il sistema elabora la richiesta e restituisce la wishlist.
- Attori: utente registrato.
- Casi d'uso inclusi: Login.
- Casi d'uso estesi: Visualizza eventi degli amici.

### 3.1.18 Visualizza wishlist personale

- Nome: Visualizza wishlist personale
- Scopo: permette all'utente di visualizzare una delle proprie wishlist, con i relativi regali.
- Precondizioni: l'utente è registrato, ha effettuato l'accesso ed ha a disposizione un token valido. L'utente ha creato almeno una wishlist.
- Flusso principale:
  1. L'utente accede alla pagina in cui sono presenti le proprie wishlist, tramite GUI e seleziona la wishlist che vuole visualizzare.

2. Il sistema elabora la richiesta.
  3. Il sistema restituisce la wishlist.
- Attori: utente registrato.
  - Casi d'uso inclusi: Login.

## 3.2 Context Diagram

Di seguito è riportato il System Context Diagram, che mostra esplicitamente i confini tra il sistema, trattato come black box e l'ambiente esterno.

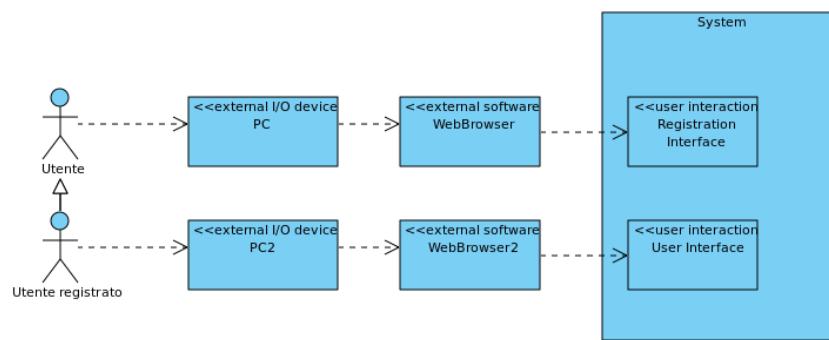


Figure 3.2: Context Diagram

## 3.3 System Domain Model

Si riporta di seguito il System Domain Model ricavato dall'analisi testuale. Il diagramma rappresenta le entità concettuali del dominio e le associazioni/relazioni tra le stesse.

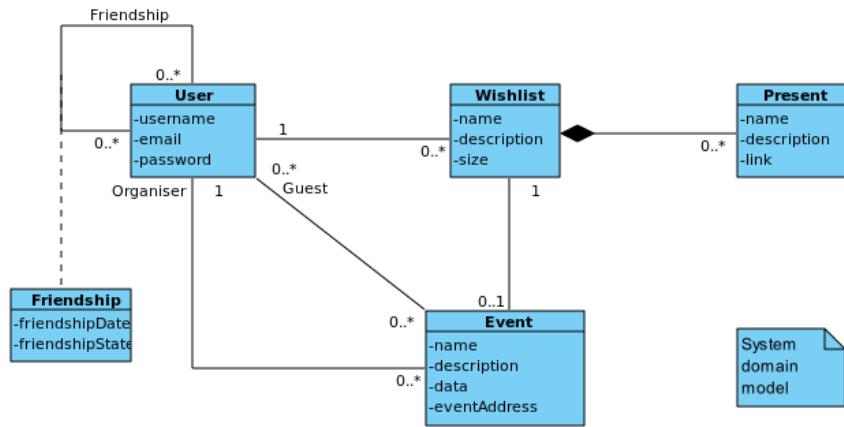


Figure 3.3: System Domain Model

### 3.4 Entity-Relationship Diagram

Di seguito è riportato l'Entity-Relationship Diagram, che mette in risalto le associazioni tra le tabelle del database relazionale. Si noti che ogni tabella corrisponde ad una ed una sola entità concettuale del Domain Model.

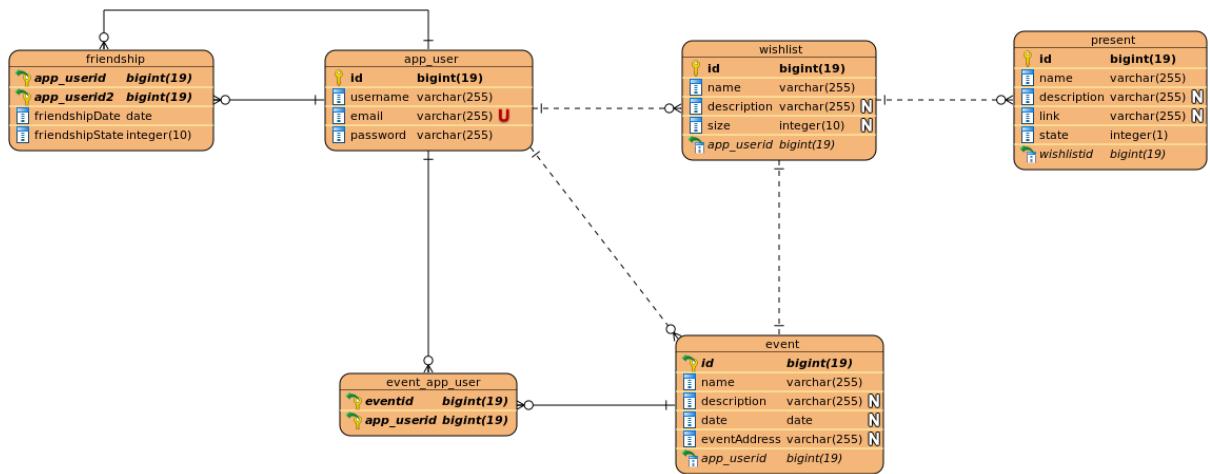


Figure 3.4: ER Diagram

# Chapter 4

## Requisiti non funzionali

In questa sezione sono riportati i principali attributi di qualità dell'applicazione e ne sono descritti gli scenari. Uno scenario è tipicamente costituito da:

- Source: l'entità che dà inizio allo scenario.
- Stimulus: l'evento che dà inizio allo scenario.
- Artifact: la porzione del sistema coinvolta.
- Environment: l'ambiente di funzionamento del sistema (ad esempio, normale utilizzo).
- Response: la risposta all'evento da parte del sistema.
- Response measure: metrica quantitativa per la valutazione della risposta.

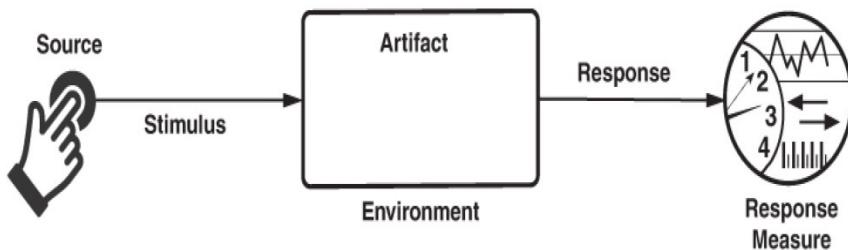


Figure 4.1: Esempio di scenario

## 4.1 Scenari

### 4.1.1 Modificability

La modificabilità è supportata dall'architettura client/server e dalla struttura a livelli del sistema. L'architettura client/server permette di avere un basso accoppiamento dei componenti del sistema grazie al fatto che il server non ha a priori informazioni sul client, a cui risulta quindi essere scarsamente accoppiato. Inoltre, non c'è accoppiamento tra i client, in quanto sono tutti indipendenti e non comunicano tra di loro. Il client e il server possono essere modificati indipendentemente, a patto che le interfacce concordate in fase di progettazione restino le stesse. La struttura a livelli, in più, favorisce sia l'alta coesione all'interno dei moduli, in quanto ogni livello ha delle responsabilità precise, sia il basso accoppiamento tra i moduli, in quanto ogni livello comunica soltanto con il livello sottostante. Le due proprietà di alta coesione e basso accoppiamento incidono positivamente sulla modificabilità del sistema software realizzato.

- Source: team di sviluppo.
- Stimulus: aggiunta della funzionalità di segnare un regalo come "acquistato".
- Artifact: classi associate a Wishlist, Present, Utente.
- Environment: fase di progettazione.
- Response: funzionalità di segnare un regalo come "acquistato" aggiunta.
- Response Measure: occorrono al massimo 6 ore per aggiungere la funzionalità.

#### 4.1.2 Robustness

La robustezza del sistema è garantita dalla validazione degli input.

- Source: utente.
- Stimulus: riempimento del campo "Email" all'atto della registrazione.
- Artifact: classi associate a Utente.
- Environment: normale funzionamento del sistema.
- Response: il sistema esegue la registrazione solo se l'email inserita dall'utente non è già stata utilizzata.
- Response Measure: -

### 4.1.3 Security

Per garantire il requisito della sicurezza si è seguito il principio della "Security by Design". La security è stata implementata sfruttando le funzionalità offerte dal framework "Spring Security". In particolare, si è scelto di adottare il meccanismo dei *JWT* Token. Ad ogni utente che effettua l'accesso viene assegnato un token che lo stesso deve fornire nel momento in cui effettua richieste verso endpoint protetti. Il sistema verifica la validità del token e, solo successivamente ad esito positivo della stessa, autorizza l'utente ad espletare la funzionalità richiesta. I token hanno una expiration date di 1 ora (eventualmente modificabile) e sono firmati. Inoltre si è deciso di cifrare le password all'interno del database.

- Source: utente che fornisce un *JWT* Token non valido.
- Stimulus: richiesta ad un endpoint protetto del sistema.
- Artifact: server, *JwtAuthorizationFilter*, *SecurityConfiguration*.
- Environment: normale funzionamento del sistema.
- Response: all'atto della richiesta, il sistema non autorizza l'utente ad effettuare la richiesta.
- Response Measure: -

#### 4.1.4 Usability

L'usabilità è garantita dall'uso di un browser come client. Ciò implica l'assenza di dipendenze o di programmi esterni da installare da parte dell'utente. All'utente basta il browser e la registrazione all'applicazione per usufruire di tutte le funzionalità.

- Source: utente.
- Stimulus: creare un evento.
- Artifact: GUI.
- Environment: normale funzionamento del sistema.
- Response: è possibile accedere alla funzionalità semplicemente compilando un form e cliccando su un bottone.
- Response Measure: per il calcolo della response measure si individuano tre categorie di utente:
  - a) utente esperto: un utente con esperienza, capace di navigare in maniera agevole su un sito web.
  - b) utente medio: un utente capace di arrivare alla soluzione dopo un numero esiguo di tentativi.
  - c) un utente poco capace e poco esperto in termini di navigazione su un sito web.

La metrica individuata per il calcolo si basa su una media pesata dei tempi di risoluzione del problema. La media dipende chiaramente dalla frequenza delle classi di utente:  $T = \frac{0.25*T_a + 0.5*T_b + 0.25*T_c}{T_a + T_b + T_c}$

# Chapter 5

## Architettura del sistema

In questa sezione si presenta in maniera dettagliata l'architettura del sistema software. Come precedentemente anticipato, il software segue il paradigma client/server.

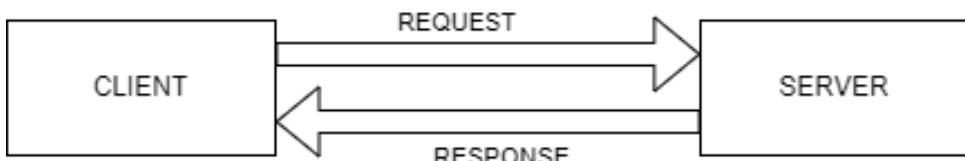


Figure 5.1: Modello Client/Server

Il server inoltre è basato su una struttura a livelli.

### 5.1 Front-end

Il front-end dell'applicazione è costituito da un sito web dinamico che espone pagine HTML, all'interno delle quali risiede il codice Javascript che ne definisce il comportamento e l'aspetto. Le pagine sono state progettate mediante il

framework grafico Bootstrap. Ogni file Javascript richiama la libreria jQuery per generare in background le richieste HTTP verso il back-end. Il metodo di tale libreria è ajax, in ingresso naturalmente prende il percorso da presentare all'API.

### 5.1.1 Thin client

Il front-end funge da presentation layer per la web app, in quanto si limita a richiamarne l'API e a presentare in maniera opportuna all'utente i dati ottenuti in risposta. Il paradigma così delineato è quello del "thin client", secondo il quale nel frontend non risiede alcuna parte della business logic. A causa di ciò, non c'è bisogno che quest'ultimo venga testato allo stesso modo del back-end, poiché è possibile stabilirne il corretto funzionamento solamente navigando il sito web; dunque, il testing del frontend coincide a livello di logica grosso modo con il test d'integrazione del back-end. Per comodità si è fatto uso della notazione UML, però nel seguente diagramma quest'ultima non è in accordo con il suo preciso significato. Infatti, si vuole semplicemente mostrare una mappa di navigabilità tra le pagine web, con elencati i wrapper alle chiamate server.

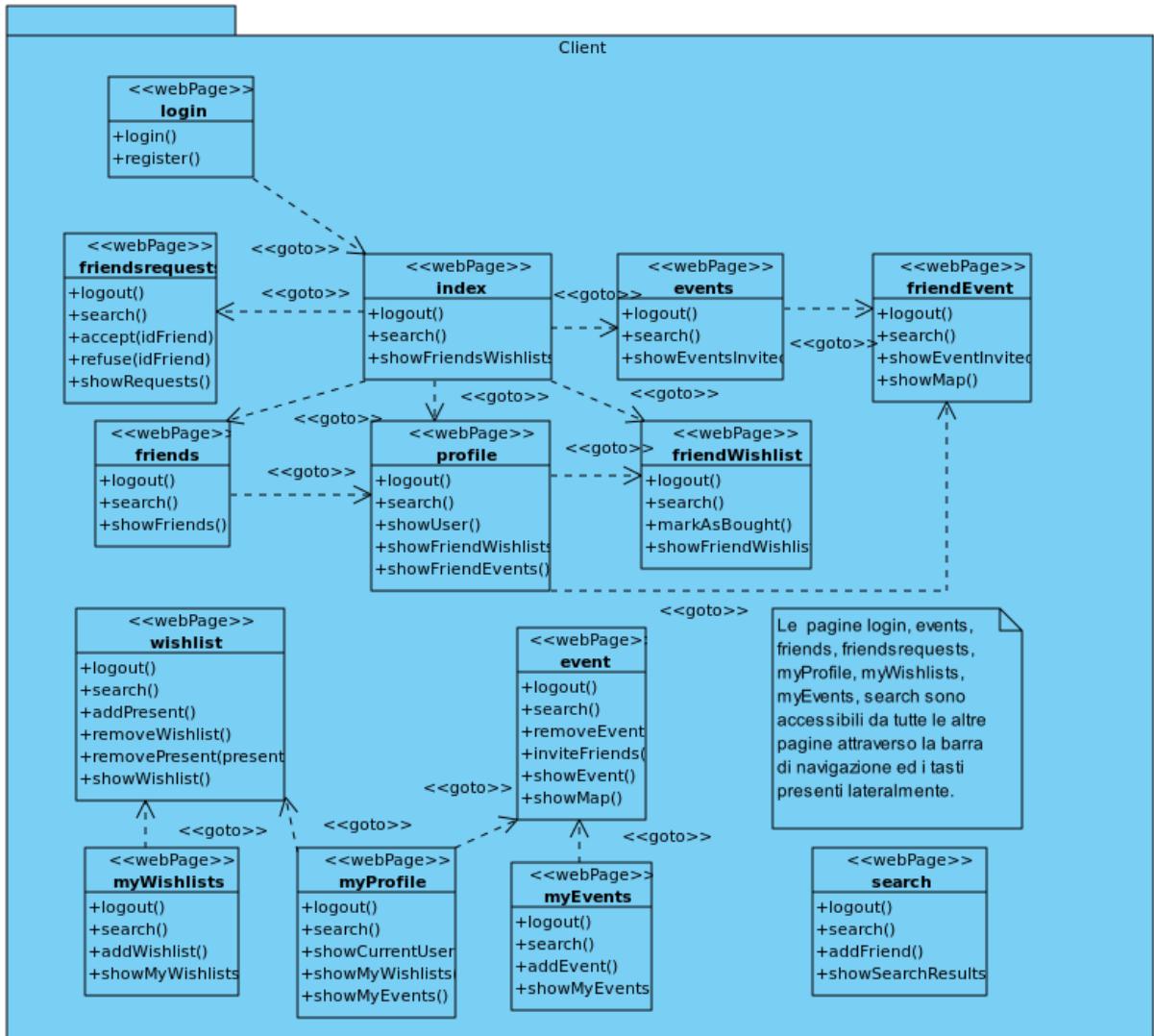


Figure 5.2: Diagramma di navigabilità delle pagine web

## 5.2 Back-end

Il back-end è costituito da un server e da un database. Il server è responsabile della business-logic dell'applicazione, il database invece dello storage delle tabelle, quali eventi, wishlist, etc. e della persistenza dei dati. Il server è stato implementato con l'ausilio di Spring.

### 5.2.1 Spring

Spring è uno dei framework Java più popolari e utilizzato dagli sviluppatori a causa dei suoi pregi. Esso permette infatti di generare codice altamente testabile, riusabile e performante. Inoltre ha un basso impatto sulla web app in termini di memoria. Spring si basa inoltre su un'architettura modulare, permettendo allo sviluppatore di concentrarsi esclusivamente sulla business logic, disinteressandosi in gran parte del codice necessario per eseguire il web server. Infine, utilizzare Spring è vantaggioso in quanto supporta la maggior parte delle tecnologie esistenti, ad esempio Hibernate in qualità di ORM (Object Relational Mapping) oppure Tomcat come JSP (Java Serve Pages).

### Spring Boot

Spring Boot è un micro-framework derivato da Spring. Offre allo sviluppatore Java la possibilità di creare un'applicazione Spring tralasciando molti dei dettagli, in quanto configura automaticamente gli strumenti sottostanti. In questo modo, lo sviluppatore può concentrarsi fin da subito nella stesura della business-logic. Spring Boot include, oltre al nucleo Spring, Hibernate in qualità di ORM e Tomcat in qualità di JSP. Garantisce, inoltre, il supporto a qualunque RDBMS.

Come già anticipato, Spring segue il paradigma a livelli, secondo cui ogni livello può comunicare soltanto con il livello sottostante. I livelli dell'architettura di Spring Boot sono quattro:

- Il controller layer, che intercetta le richieste HTTP inviate dal client.

- Il service layer, contiene la business-logic dell'applicazione.
- Il repository layer, che implementa la logica di memorizzazione e inserisce gli oggetti del dominio di business nelle tabelle (e viceversa). Inoltre, per ogni entità, il repository contiene un'interfaccia dotata di metodi che interagiscono con il database.
- Il database layer è il livello che corrisponde al database vero e proprio. In questo layer si realizzano le vere e proprie operazioni sulle tabelle, come le query.

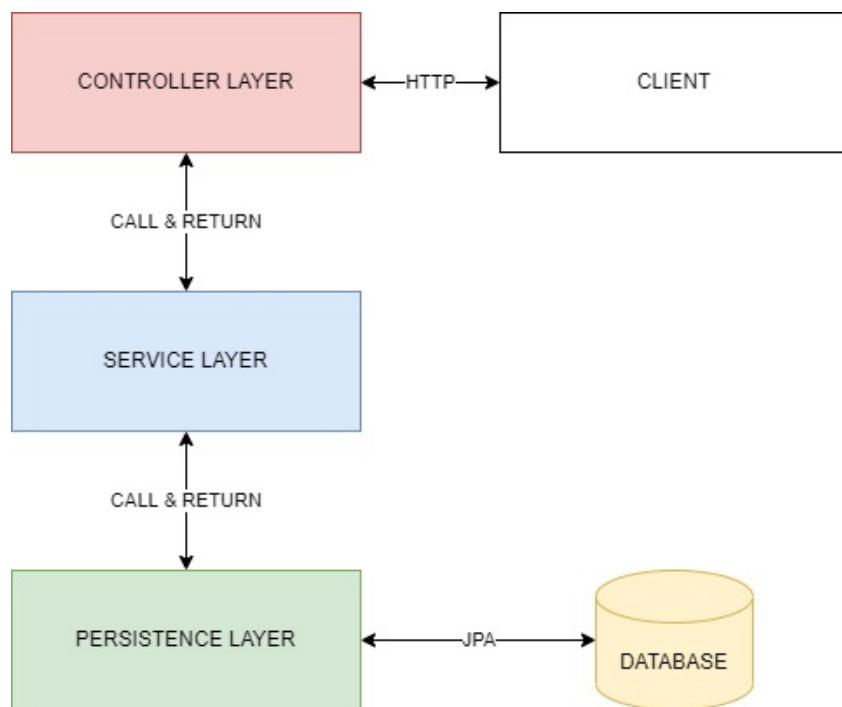


Figure 5.3: Architettura di Spring Boot

## Spring Security

Spring Security è un framework che fornisce autenticazione, autorizzazione e protezione contro gli attacchi più comuni e costituisce lo standard de facto

per la protezione delle applicazioni basate su Spring. Alcune delle funzionalità fornite da Spring Security riguardano:

- Il supporto per autenticazione ed autorizzazione;
- cifratura delle password degli utenti dell'applicazione;
- protezione dalle classiche vulnerabilità a cui è esposta una web app, come ad esempio *CSRF*, *XSS*, etc.
- integrazione con la Servlet API e con il Servlet Container, attraverso un Servlet Filter che si trova a monte del Dispatcher Servlet, che ha il compito di intercettare le richieste dirette all'applicazione e filtrarle.

Di seguito sono riportate le funzionalità di Spring Security adoperate per mettere in sicurezza WishlistApp.

### **PasswordEncoder**

L'interfaccia PasswordEncoder di Spring Security viene utilizzata per eseguire una trasformazione unidirezionale di una password, in modo che la stessa venga memorizzata in maniera sicura sul database. In particolare l'implementazione scelta è stata il "BCryptPasswordEncoder", che utilizza l'algoritmo di cifratura bcrypt per eseguire l'hash delle password.

### **Authentication**

Per quanto riguarda l'autenticazione degli utenti dell'applicazione, si è deciso di sfruttare le funzionalità offerte da Spring Security e di modificarle con delle

funzioni "custom", così da adattarle alle esigenze del progetto. In particolare, Spring Security implementa i meccanismi di autenticazione e autorizzazione utilizzando le "Security Filter Chain". Tutte le richieste HTTP dirette verso gli endpoint dell'applicazione vengono intercettate dal filtro che, a seconda della specifica richiesta e dai parametri in ingresso, decide se filtrarla oppure autorizzarla.

Nel momento in cui un utente effettua una richiesta di login, la stessa viene intercettata dal filtro. Nel caso in esame, si è deciso di non proteggere l'endpoint relativo al login, facendo in modo che il filtro "bypassi" la richiesta. A questo punto, le credenziali fornite in ingresso vengono passate all'Authentication Manager, l'API che definisce come i filtri di Spring Security debbano eseguire l'autenticazione. L'Authentication Manager (che è un'interfaccia) viene implementato dal Provider Manager, che delega la funzione di autenticazione all'Authentication Provider. Quest'ultimo è l'entità finale che sa come eseguire l'autenticazione. Per decidere se autenticare o meno l'utente, il Provider acquisisce le informazioni relative a quest'ultimo interfacciandosi con l'UserDetailsService. Presa la decisione, l'AuthenticationProvider la restituisce all'AuthorizationManager mediante un oggetto di tipo *authentication*. Se l'esito della decisione è positivo, allora l'AuthenticationController genera un JwtToken che restituisce all'utente, che lo stesso deve utilizzare per potersi autenticare presso gli endpoint protetti per le richieste successive. Infatti, la Session Policy che si è deciso di implementare è *STATELESS*.

Di seguito sono riportati i diagrammi relativi alla fase di autenticazione.

## CHAPTER 5. ARCHITETTURA DEL SISTEMA

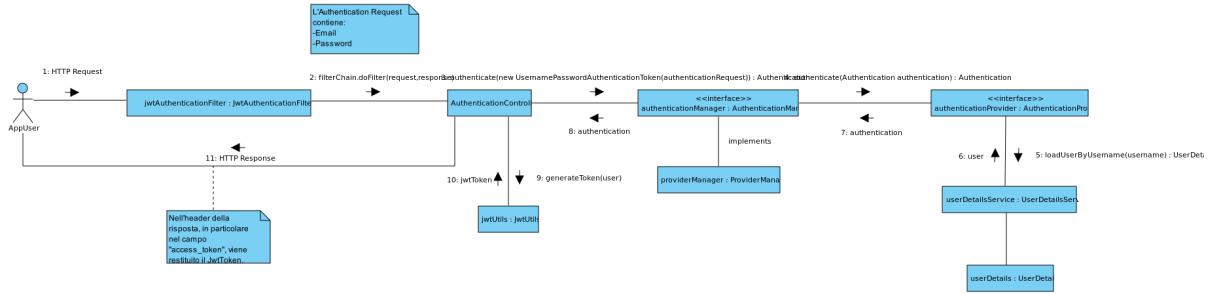


Figure 5.4: Communication Diagram per l'autenticazione con username e password

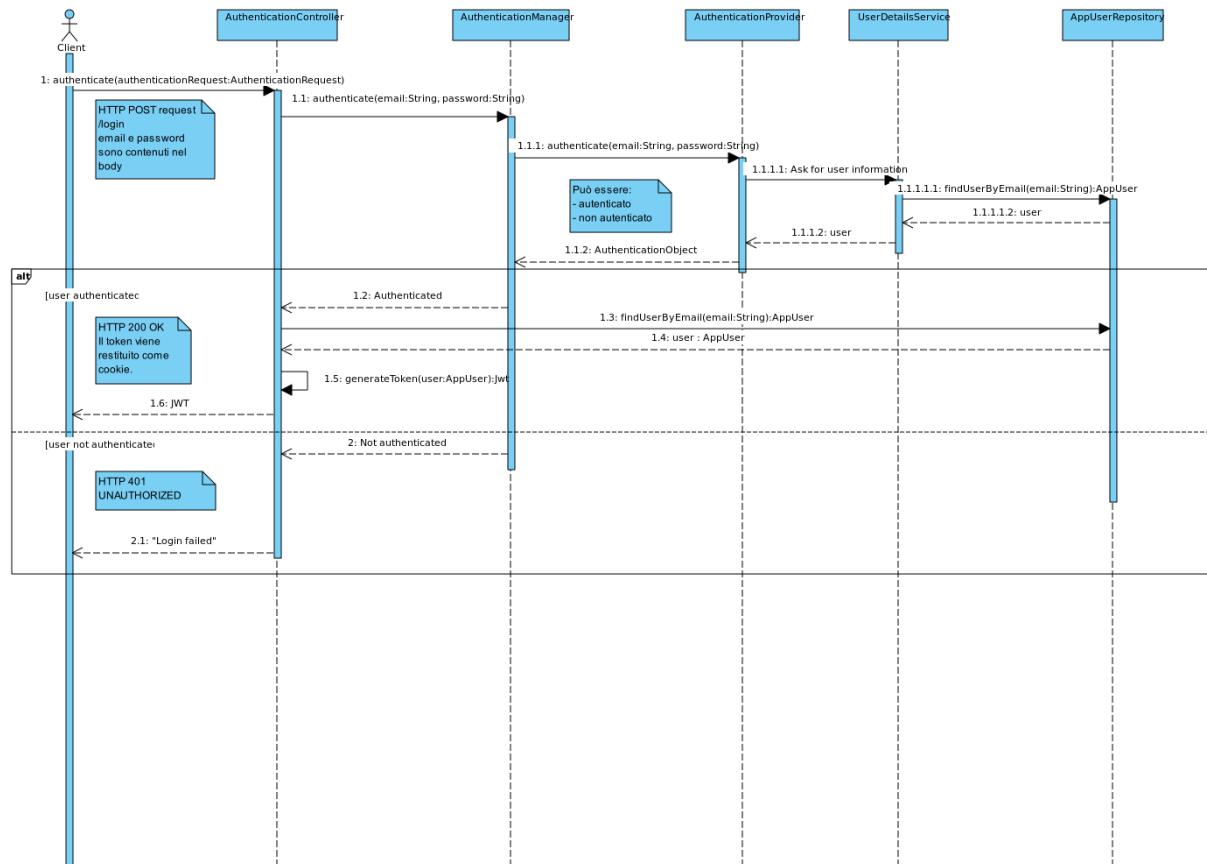


Figure 5.5: Sequence diagram dell'autenticazione con username e password

Per quanto riguarda invece tutte le HTTP Request verso gli endpoint protetti, queste saranno sempre intercettate da un FilterChain. Nel caso in

esame si è deciso di implementare un filtro custom, il JwtAuthenticationFilter. Il filtro ha lo scopo di verificare che le informazioni contenute nel token siano corrette e che il token stesso sia valido. In caso di esito positivo, il filtro aggiorna lo stato del Security Context Holder, che è il luogo in cui Spring Security memorizza le informazioni di chi è autenticato, autenticando di fatto il richiedente. Infine la richiesta viene lasciata passare dal filtro e indirizzata verso l'endpoint a cui era destinata.

Di seguito è riportato il sequence diagram relativo alle HTTP Request dirette verso endpoint protetti.

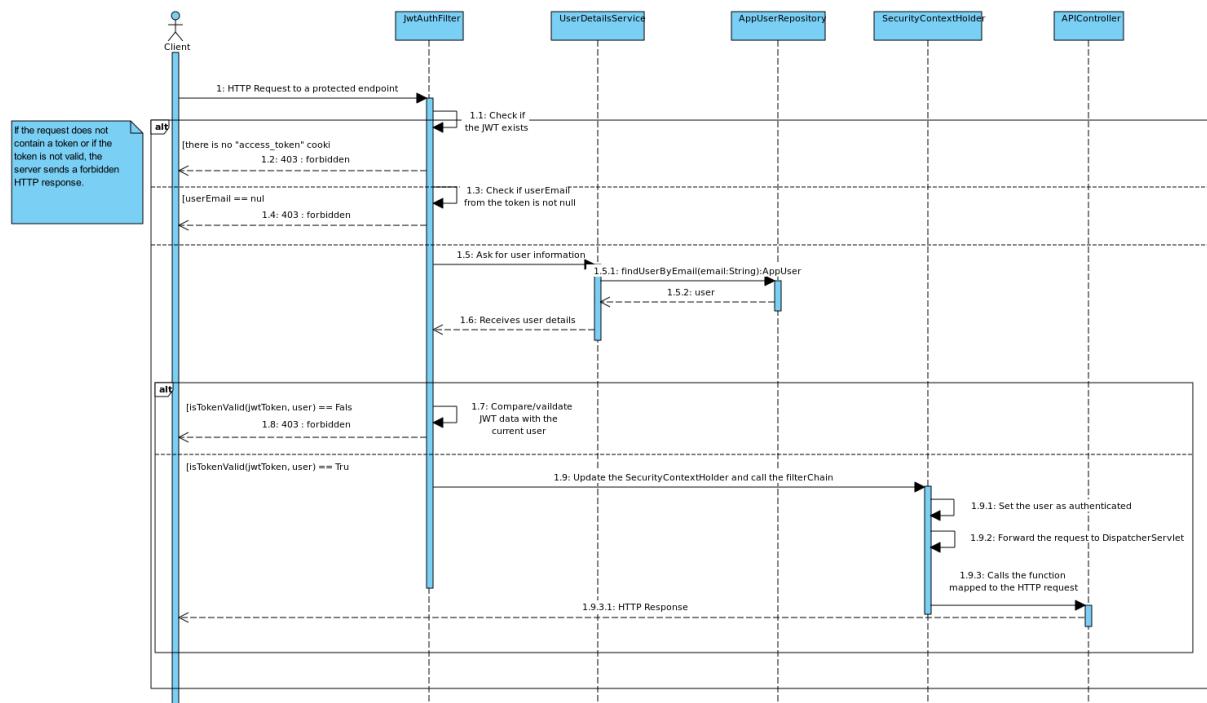


Figure 5.6: Sequence diagram della fase di autenticazione con JWT token verso endpoint protetti

### 5.2.2 Database

Per il salvataggio e la persistenza dei dati si è utilizzato il database open-source PostgreSQL. PostgreSQL è un RDBMS altamente estensibile che dispone di numerose interfacce per la connessione con diversi linguaggi, tra cui Java.

## 5.3 Package Diagram

Di seguito è riportato il Package Diagram, che mostra la struttura a livelli della web application. Ogni livello "usa" soltanto il livello sottostante.

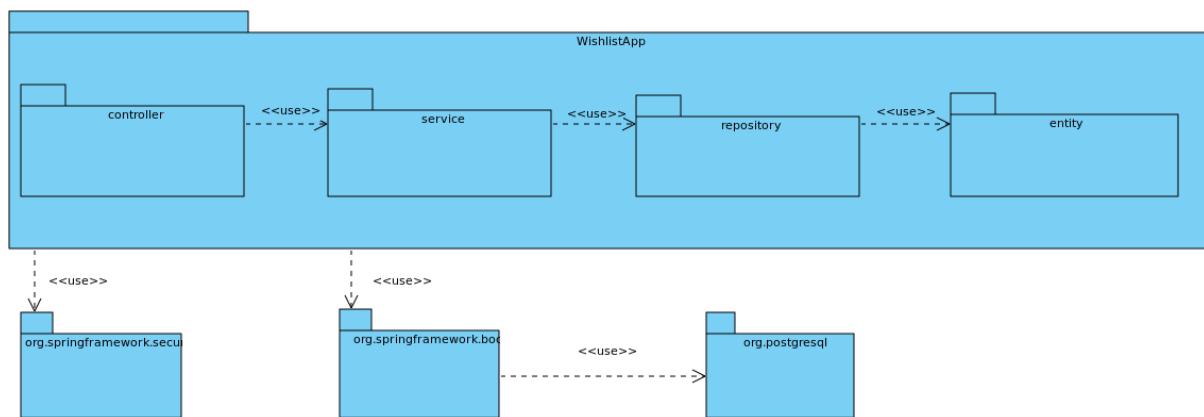


Figure 5.7: Package diagram

### 5.3.1 Entity Class Diagram

Di seguito è riportato il class diagram del package Entity ottenuto in seguito al raffinamento del System Domain Model.

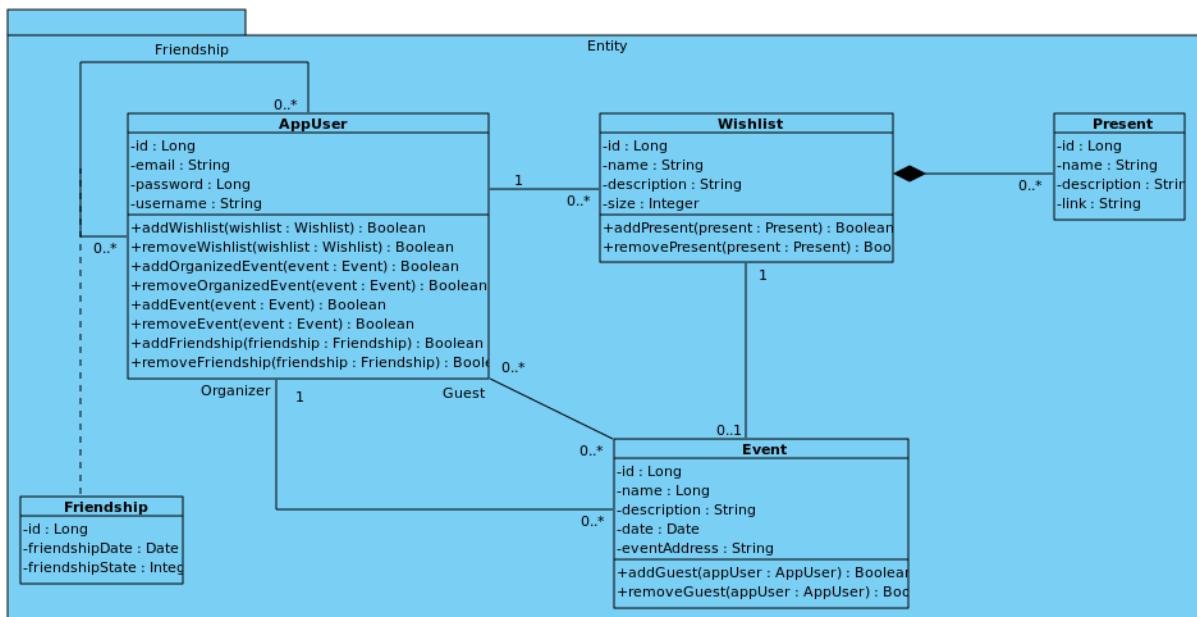


Figure 5.8: Entity Class Diagram

### 5.3.2 Repository Class Diagram

Di seguito si presenta il class diagram del package Repository, che contiene le interfacce JPA per la comunicazione con il database.

## CHAPTER 5. ARCHITETTURA DEL SISTEMA

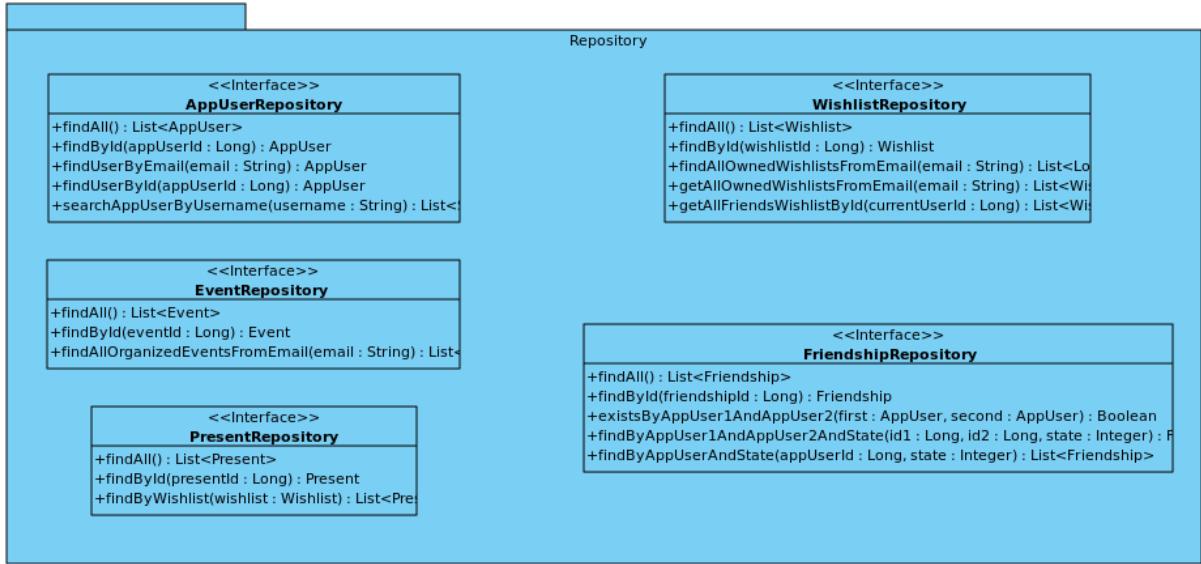


Figure 5.9: Repository Class Diagram

### 5.3.3 Service Class Diagram

Di seguito si presenta il class diagram del package Service, responsabile della business logic della web application.

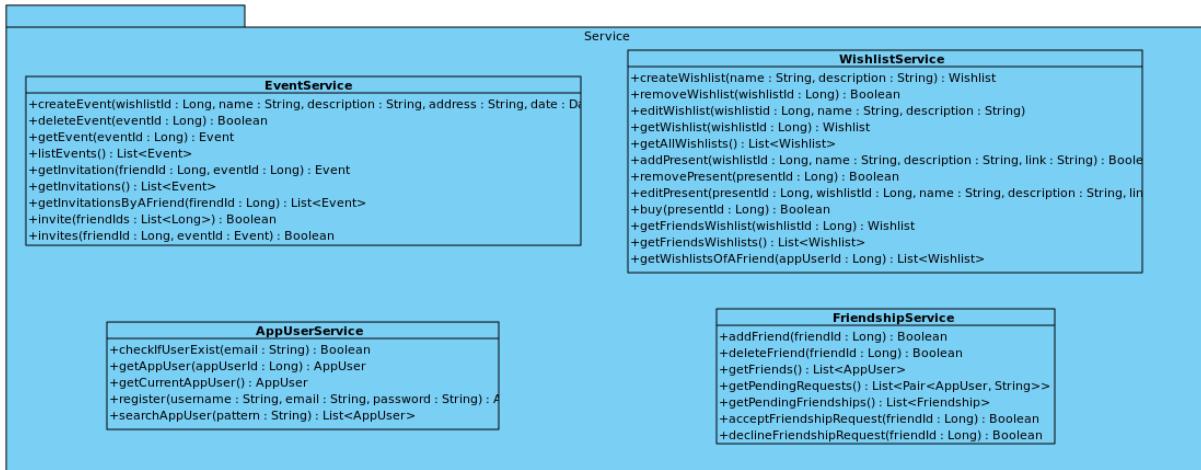


Figure 5.10: Service Class Diagram

### 5.3.4 Controller Class Diagram

Di seguito si presenta il class diagram del package Controller, che espone i metodi costituenti l'API fruibili tramite richieste HTTP.

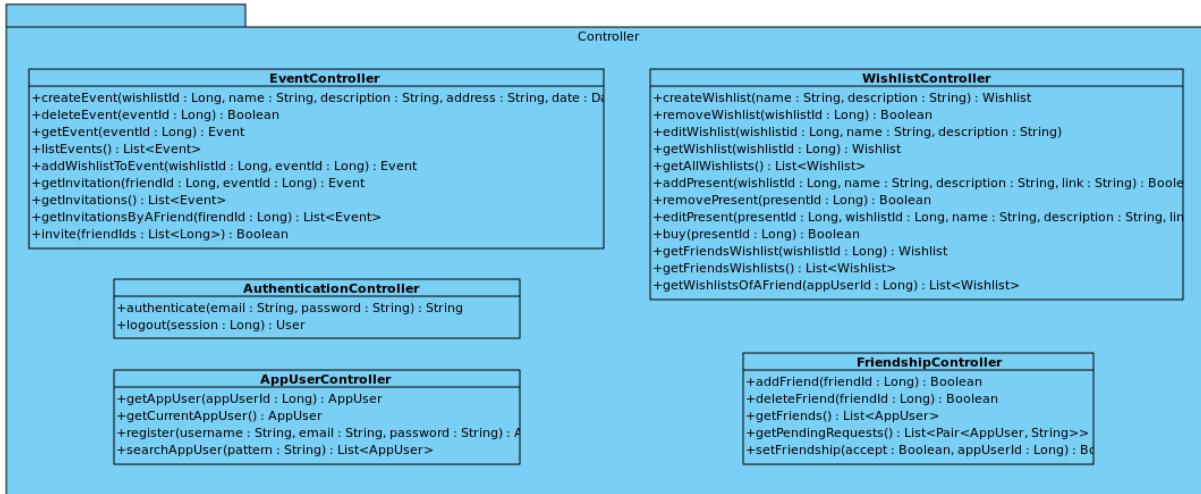


Figure 5.11: Controller Class Diagram

## 5.4 Component Diagram

Di seguito si presenta il component diagram, nel quale il client e il server comunicano tramite protocollo HTTP, mentre il server comunica con il database tramite protocollo JDBC. Si noti che esiste un rapporto N a 1 tra i moduli del sistema e i componenti. Infatti, i file Java associati alle classi che compongono il server si manifestano al run-time in un unico componente, il WebApplicationServer. Analogamente, i file HTML e Javascript che compongono la logica di presentazione del client si manifestano al run time in un unico componente, il WebApplicationClient.



Figure 5.12: Component Diagram

## 5.5 API

Di seguito si è deciso di documentare tutti i metodi facenti parte delle API.

Per ognuno di essi, viene riportato il nome, il percorso sul file system remoto, i parametri di input, i parametri di output e il comportamento atteso.

### 5.5.1 Evento

#### **createEvent**

- Percorso: event/create
- Input: Long idWishlist, String name, String description, String dateString, String eventAddress.
- Output: Event
- Comportamento: Il metodo crea l'evento con le specifiche inserite in input.

#### **deleteEvent**

- Percorso: event/delete/idEvent

- Input: Long idEvent
- Output: Boolean
- Comportamento: il metodo restituisce true se l'evento è stato eliminato, false altrimenti.

### **getEvent**

- Percorso: event/idEvent
- Input: idEvent
- Output: Event
- Comportamento: restituisce il singolo evento indicato in input, a patto che l'utente richiedente ne sia l'organizzatore.

### **listEvents**

- Percorso: event/myevents
- Input: -
- Output: List<Event>
- Comportamento: restituisce tutti gli eventi organizzati dall'utente che ha sottoposto la richiesta.

### **getInvitations**

- Percorso: event/myinvitations
- Input: -
- Output: List<Event>
- Comportamento: Restituisce tutti gli eventi a cui l'utente è invitato

### **getInvitationsByAFriend**

- Percorso: event/myinvitationsby/idFriend
- Input: Long idFriend
- Output: List<Event>
- Comportamento: Restituisce tutti gli eventi di un amico a cui l'utente è stato invitato.

### **getInvitation**

- Percorso: event/invitation
- Input: Long idFriend, Long idEvent
- Output: Event
- Comportamento: restituisce all'utente, se invitato, lo specifico evento dell'amico indicato in input.

### **invites**

- Percorso: event/invite
- Input: List<Long> idUsers, Long idEvent
- Output: Boolean
- Comportamento: restituisce true se gli amici vengono invitati all'evento con successo, false altrimenti.

### **5.5.2 Wishlist**

#### **createWishlist**

- Percorso: wishlist/create
- Input: String name, String description
- Output: Wishlist
- Comportamento: restituisce la wishlist creata.

#### **removeWishlist**

- Percorso: wishlist/delete
- Input: Long idWishlist
- Output: -
- Comportamento: elimina la wishlist indicata in input.

### **getFriendsWishlist**

- Percorso: wishlist/idWishlist/friend
- Input: Long idWishlist
- Output: Wishlist
- Comportamento: restituisce la wishlist di un amico specificata in input, a patto che:
  - i due utenti siano amici;
  - se la wishlist è associata ad un evento, allora l'utente per visualizzarla deve essere invitato allo stesso.

### **getWishlist**

- Percorso: wishlist/idWishlist
- Input: Long idWishlist
- Output: Wishlist
- Comportamento: restituisce la wishlist specificata in ingresso, a patto che l'utente ne sia proprietario.

### **getAllWishlists**

- Percorso: wishlist/all
- Input: -

- Output: List<Wishlist>
- Comportamento: Restituisce tutte le wishlist di cui l'utente (in sessione) è proprietario.

### **addPresent**

- Percorso: wishlist/wishlistId/add
- Input: Long idWishlist, String name, String description, String link
- Output: Wishlist
- Comportamento: aggiunge il regalo specificato in input alla wishlist, a patto che l'utente ne sia proprietario.

### **removePresent**

- Percorso: wishlist/idWishlist/delete/idPresent
- Input: Long idWishlist, Long idPresent
- Output: Boolean
- Comportamento: restituisce true se il regalo viene eliminato dalla wishlist specificata in input, false altrimenti. La richiesta va a buon fine a patto che chi la sottopone sia il proprietario della wishlist.

### **getFriendsWishlists**

- Percorso: wishlist/friendswishlist

- Input: -
- Output: List<Wishlist>
- Comportamento: restituisce tutte le wishlist di tutti gli amici. Verranno restituite tutte le wishlist:
  - appartenenti ad un amico e non associate ad alcun evento;
  - appartenenti ad un amico e associate ad un evento a cui l'utente è invitato.

### **getWishlistsOfAFriend**

- Percorso: wishlist/wishlistsofafriend
- Input: Long idFriend
- Output: List<Wishlist>
- Comportamento: restituisce tutte le wishlist appartenenti ad uno specifico amico indicato in input:
  - associate ad un amico e non associate ad alcun evento;
  - associate ad un amico e associate ad un evento a cui l'utente è invitato.

### **buy**

- Percorso: wishlist/buy

- Input: Long idPresent
- Output: Boolean
- Comportamento: il metodo restituisce true se il regalo viene segnato come "acquistato" correttamente, false altrimenti.

### 5.5.3 AppUser

#### register

- Percorso: /register
- Input: String username, String email, String password
- Output: Boolean
- Comportamento: restituisce true se l'utente viene registrato correttamente, false altrimenti. L'email specificata in ingresso non deve essere già stata utilizzata da altri utenti del sistema.

#### searchAppUser

- Percorso: /search
- Input: String pattern
- Output: List<AppUser>
- Comportamento: restituisce una lista di utenti che sono conformi alla stringa pattern immessa in input.

### **getAppUser**

- Percorso: /user
- Input: Long idUser
- Output: AppUser
- Comportamento: restituisce l'AppUser associato all'identificativo inserito in input.

#### **5.5.4 Friendship**

##### **addFriend**

- Percorso: /addFriend
- Input: Long idFriend
- Output: Boolean
- Comportamento: restituisce true se la richiesta di amicizia viene inviata correttamente, false altrimenti. Affinchè la richiesta vada a buon fine:
  - i due utenti non devono essere già amici;
  - non deve esserci, tra i due utenti, una richiesta di amicizia già in pending.

### **deleteFriend**

- Percorso: /deletefriend
- Input: Long idFriend
- Output: Boolean
- Comportamento: il metodo restituisce true se l'amico viene eliminato correttamente, false altrimenti.

### **getFriends**

- Percorso: /listFriends
- Input: -
- Output: List<AppUser>
- Comportamento: restituisce la lista di tutti gli amici dell'utente che ha sottoposto la richiesta.

### **getPendingRequests**

- Percorso: /listPendingRequests
- Input: -
- Output: List<Pair<AppUser, String>>

- Comportamento: il metodo restituisce, per ogni richiesta di amicizia pendente, l'utente che l'ha effettuata e la data in cui è stata inviata la richiesta di amicizia.

### **setFriendship**

- Percorso: /setFriendship
- Input: Boolean set, Long idFriend
- Output: String
- Comportamento: a seconda del valore "set" in input, il metodo svolge funzioni diverse:
  1. Set = 1: la richiesta di amicizia inviata dall'utente specificato in input viene accettata.
  2. Set = 0: la richiesta di amicizia inviata dall'utente specificato in input viene rifiutata.

### **5.5.5 Authentication**

#### **authenticate**

- Percorso: /login
- Input: String email, String password
- Output: String

- Comportamento: se le credenziali fornite sono corrette, l’utente viene autenticato. Gli viene inoltre fornito un JWT Token con il quale potrà essere autorizzato a contattare gli endpoint protetti.

# Chapter 6

# Diagrammi comportamentali

In questo capitolo saranno presentati i diagrammi comportamentali dettagliati del software, che riportano una panoramica delle operazioni eseguite in Java in caso di utilizzo delle varie API.

## 6.1 Sequence Diagram

Riportiamo in questa sezione i sequence diagram realizzati per le varie API.

### 6.1.1 addPresent

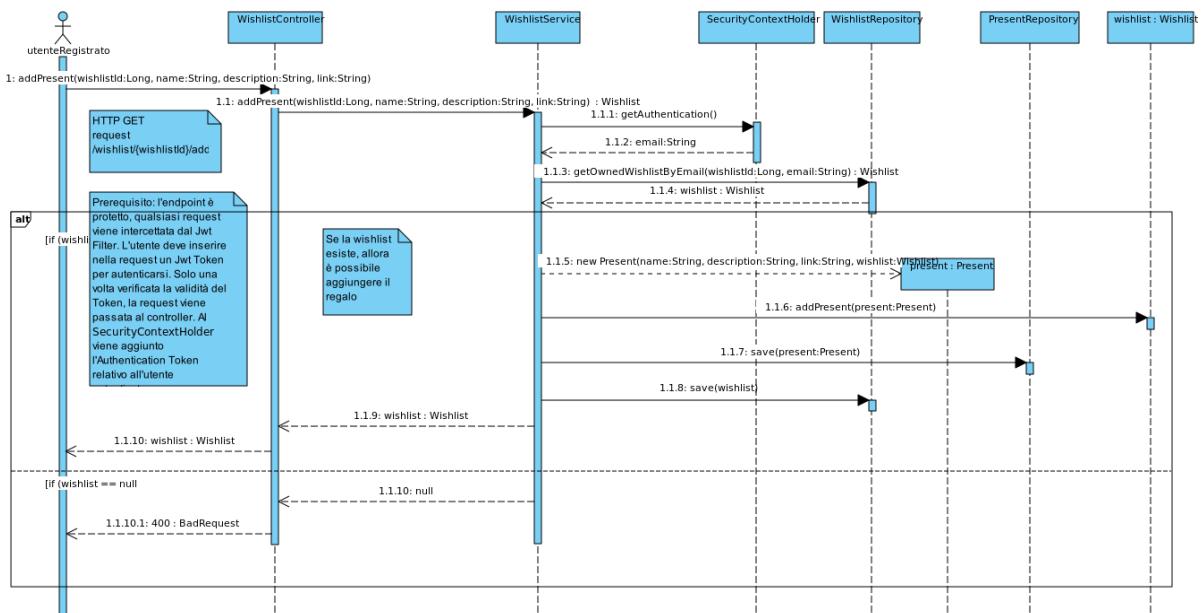


Figure 6.1: Sequence diagram della funzione *addPresent*

### 6.1.2 getAllWishlists

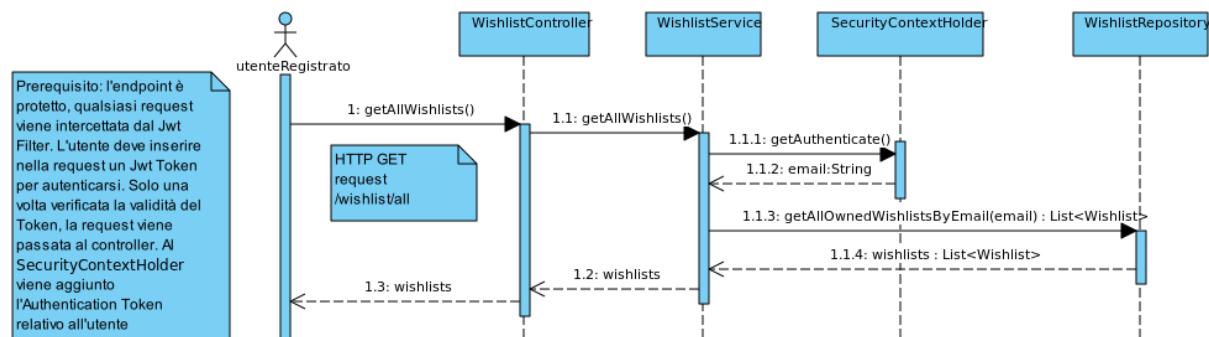


Figure 6.2: Sequence diagram della funzione *getAllWishlists*

### 6.1.3 login

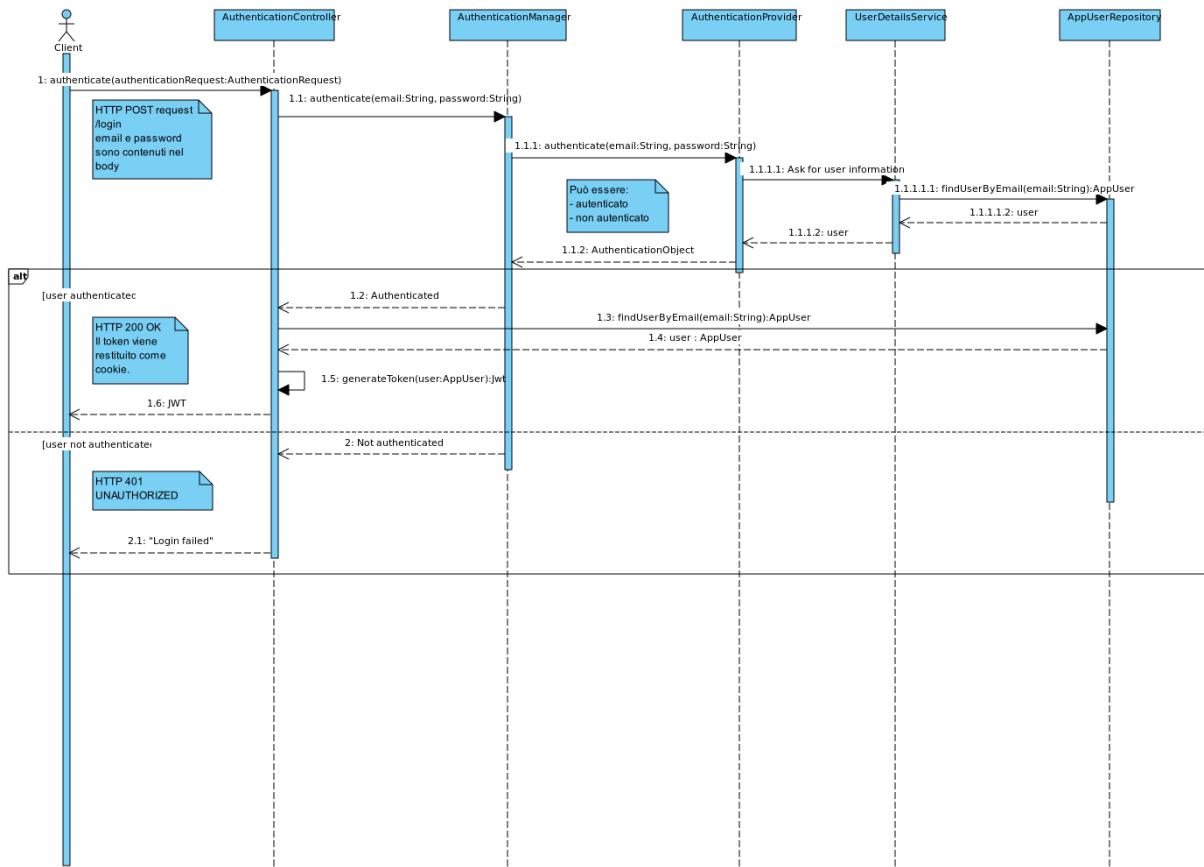


Figure 6.3: Sequence diagram della funzione *login*

### 6.1.4 register

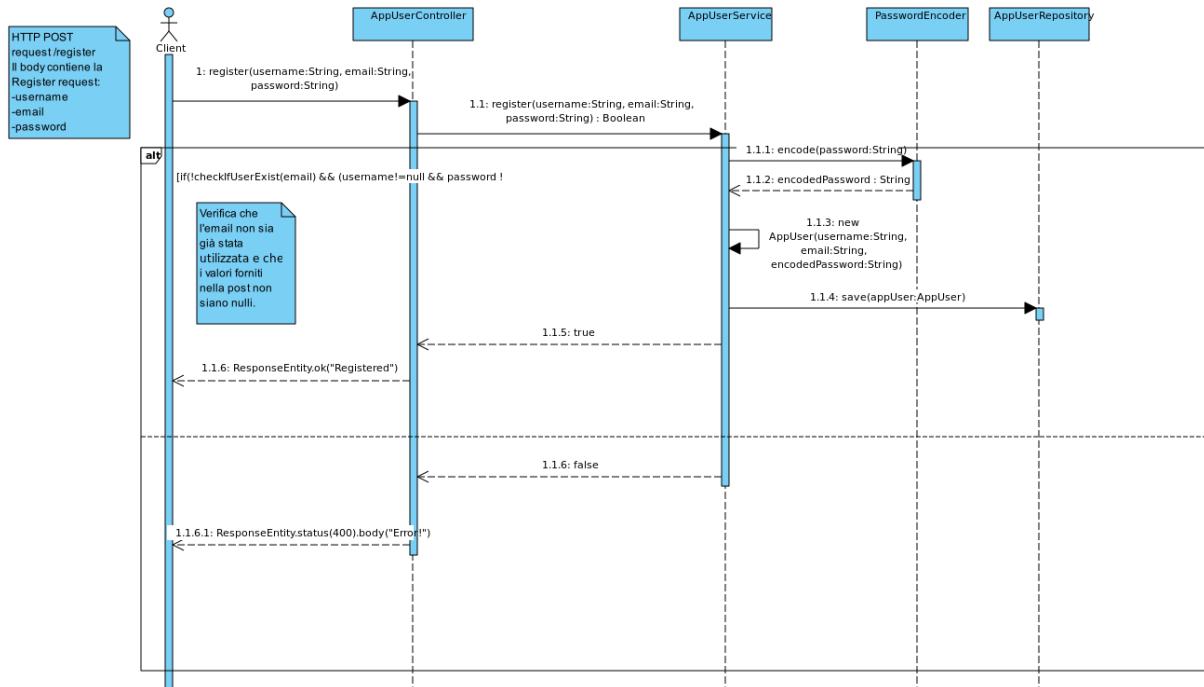


Figure 6.4: Sequence diagram della funzione *register*

### 6.1.5 createWishlist

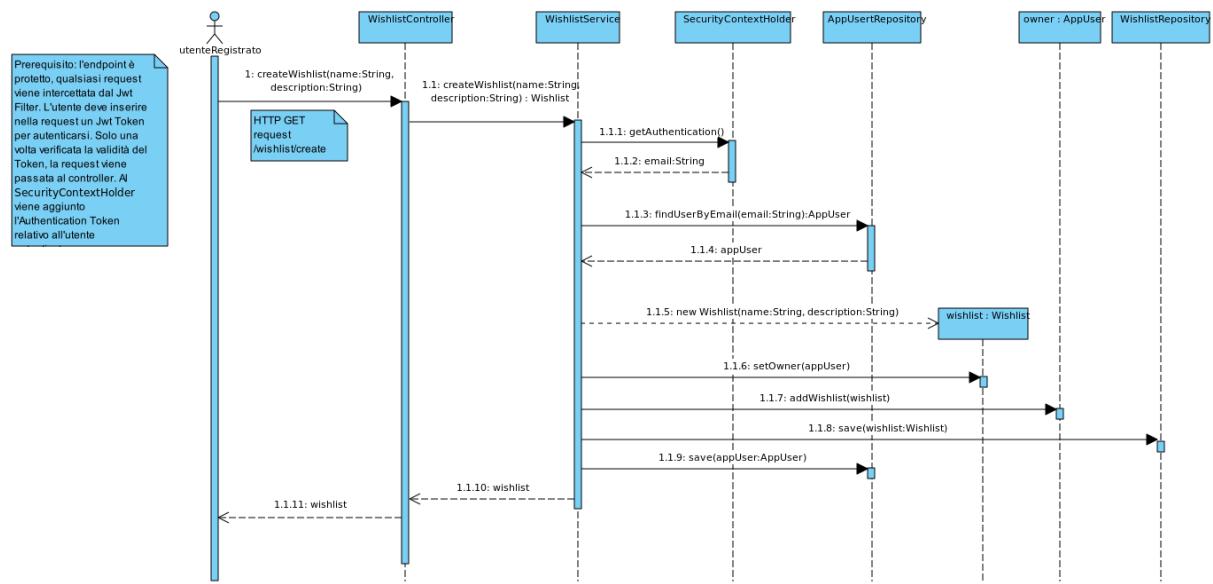


Figure 6.5: Sequence diagram della funzione `createWishlist`

### 6.1.6 removeWishlist

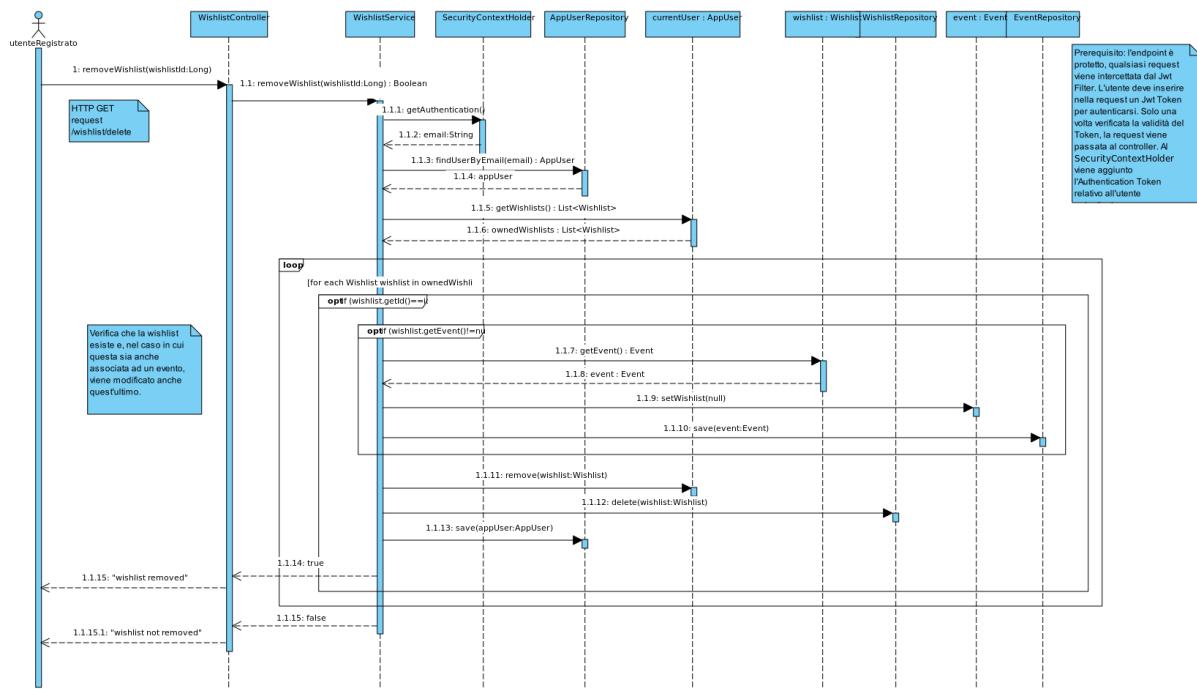


Figure 6.6: Sequence diagram della funzione `createWishlist`

## 6.1.7 createEvent

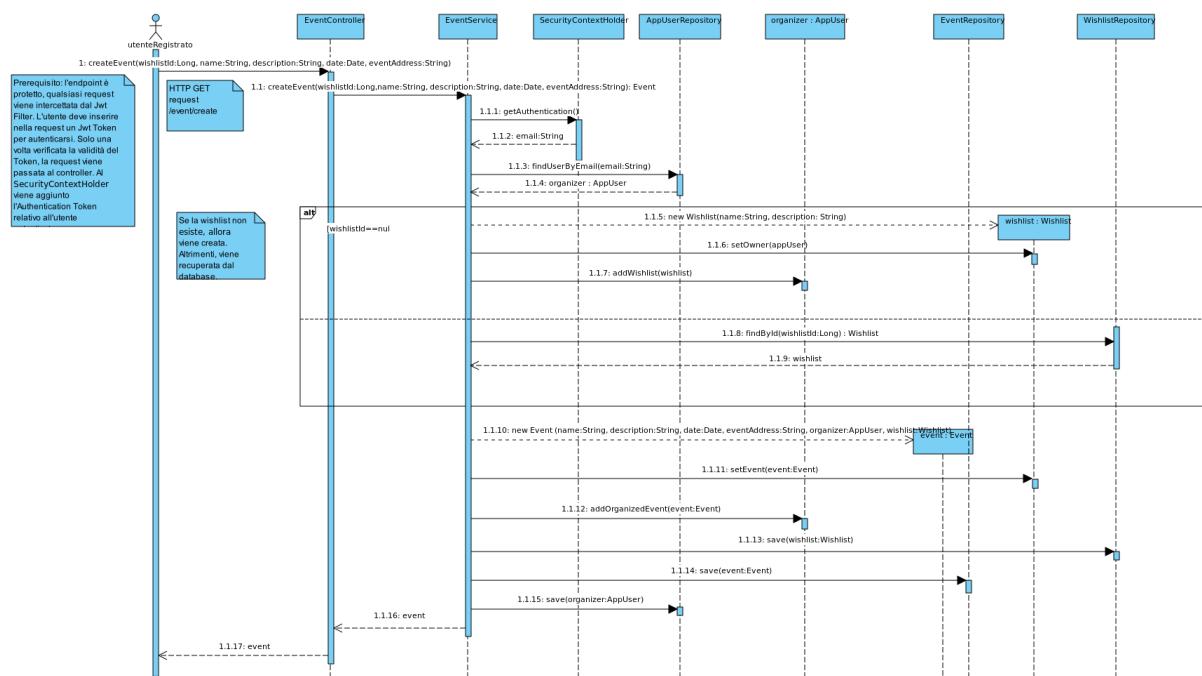


Figure 6.7: Sequence diagram della funzione `createEvent`

### 6.1.8 removePresent

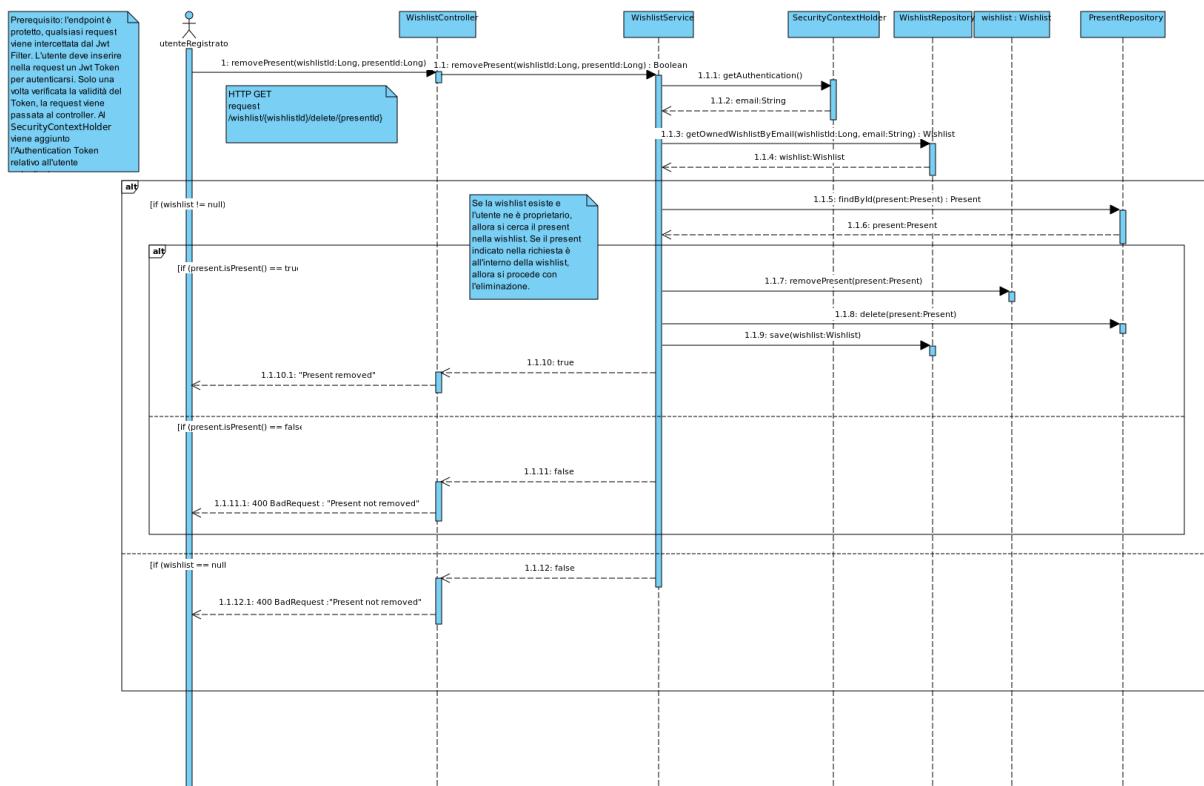


Figure 6.8: Sequence diagram della funzione *removePresent*

### 6.1.9 deleteEvent

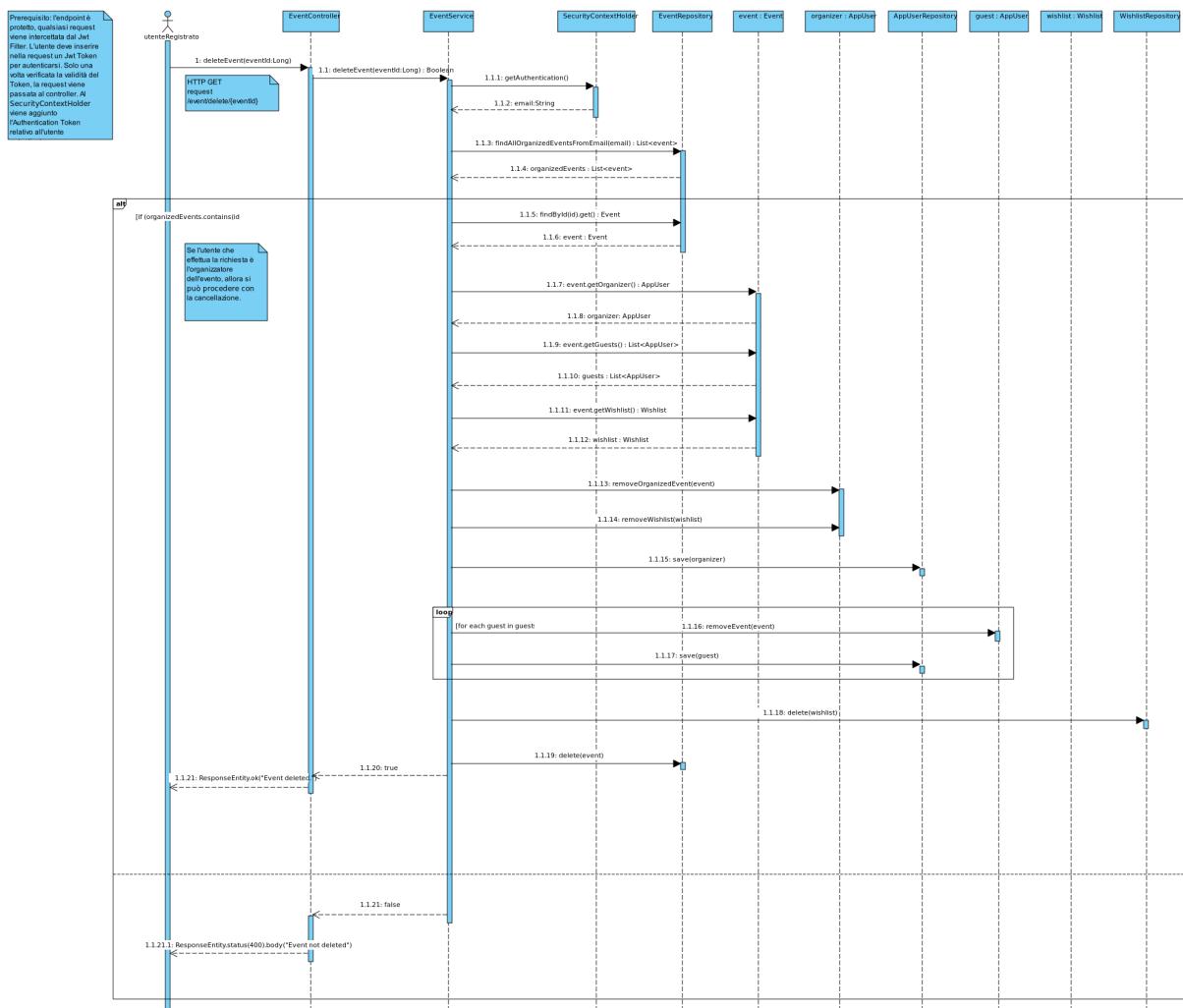


Figure 6.9: Sequence diagram della funzione *deleteEvent*

### 6.1.10 getFriendsWishlists

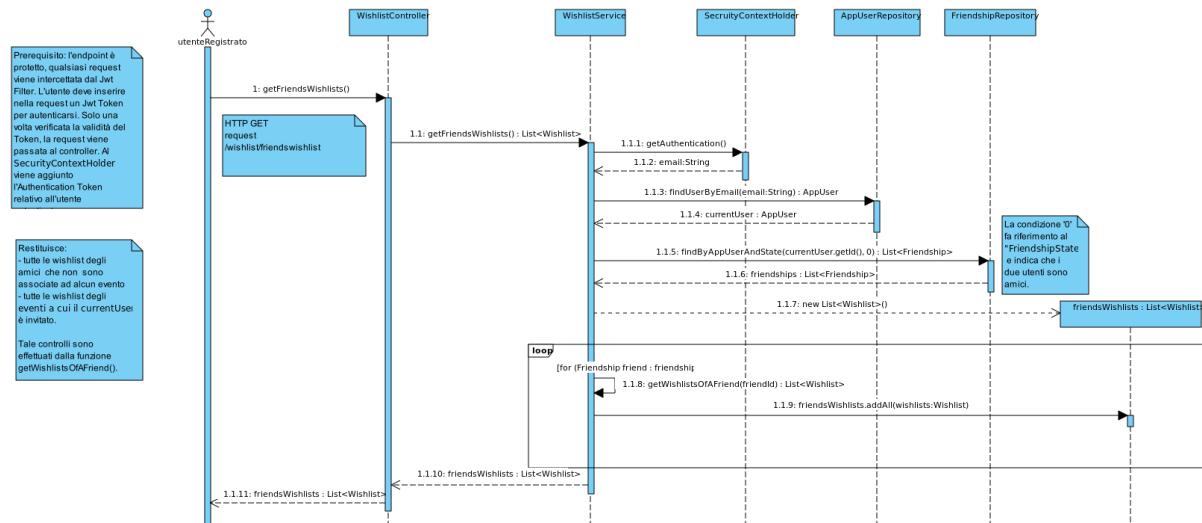


Figure 6.10: Sequence diagram della funzione `getFriendsWishlists`

### 6.1.11 search

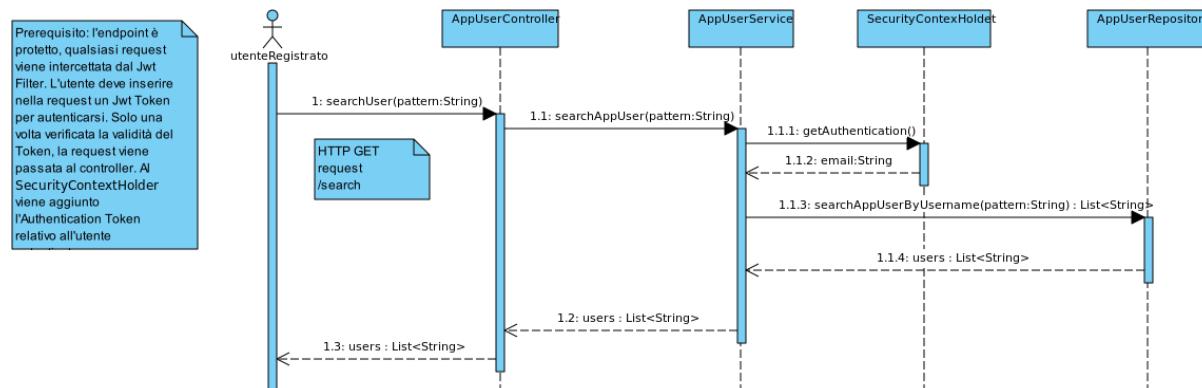


Figure 6.11: Sequence diagram della funzione `search`

### 6.1.12 getFriendsWishlist

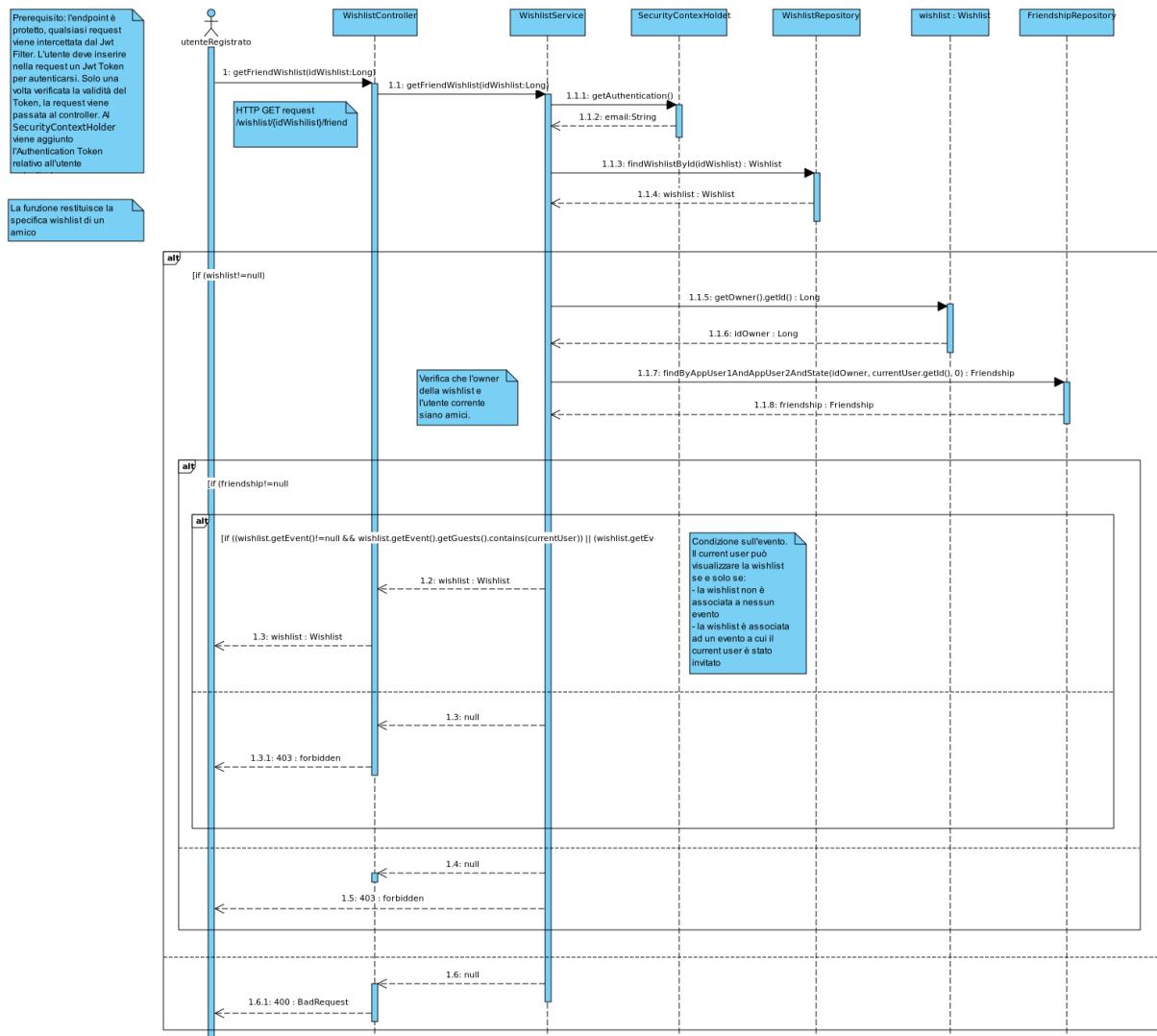


Figure 6.12: Sequence diagram della funzione *getFriendsWishlist*

### 6.1.13 getWishlist

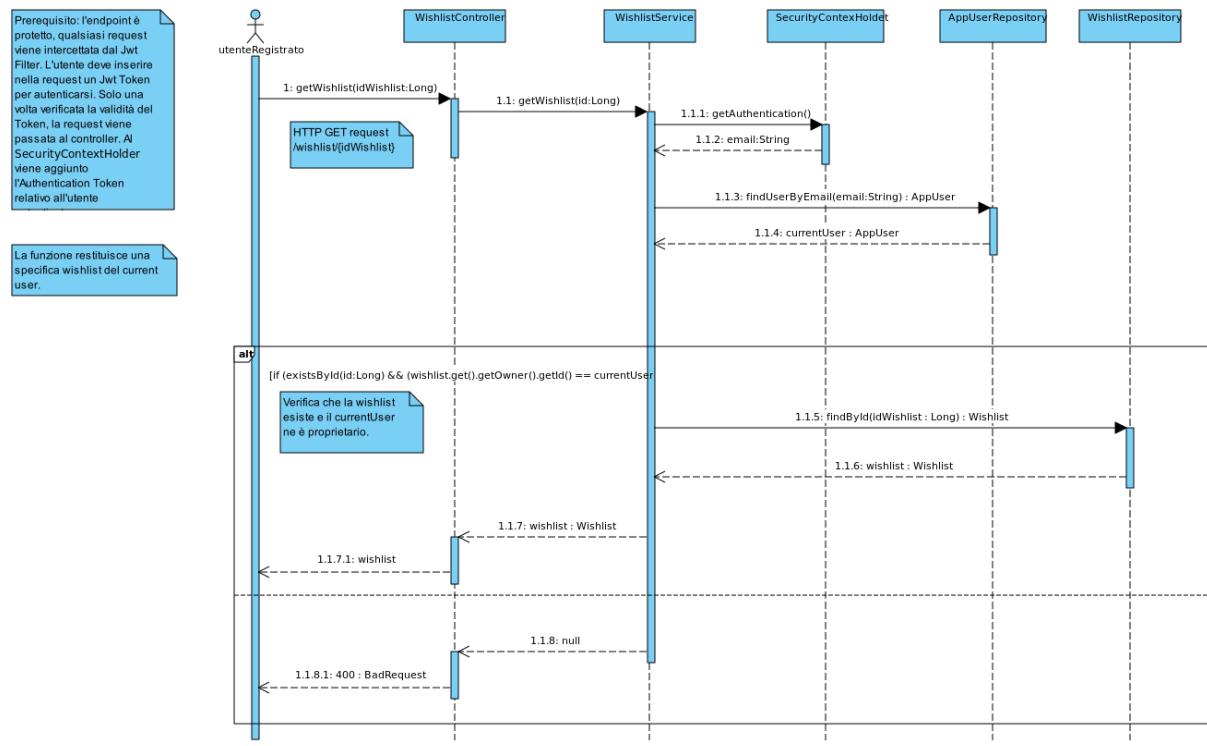


Figure 6.13: Sequence diagram della funzione `getWishlist`

### 6.1.14 getWishlistOfAFriend

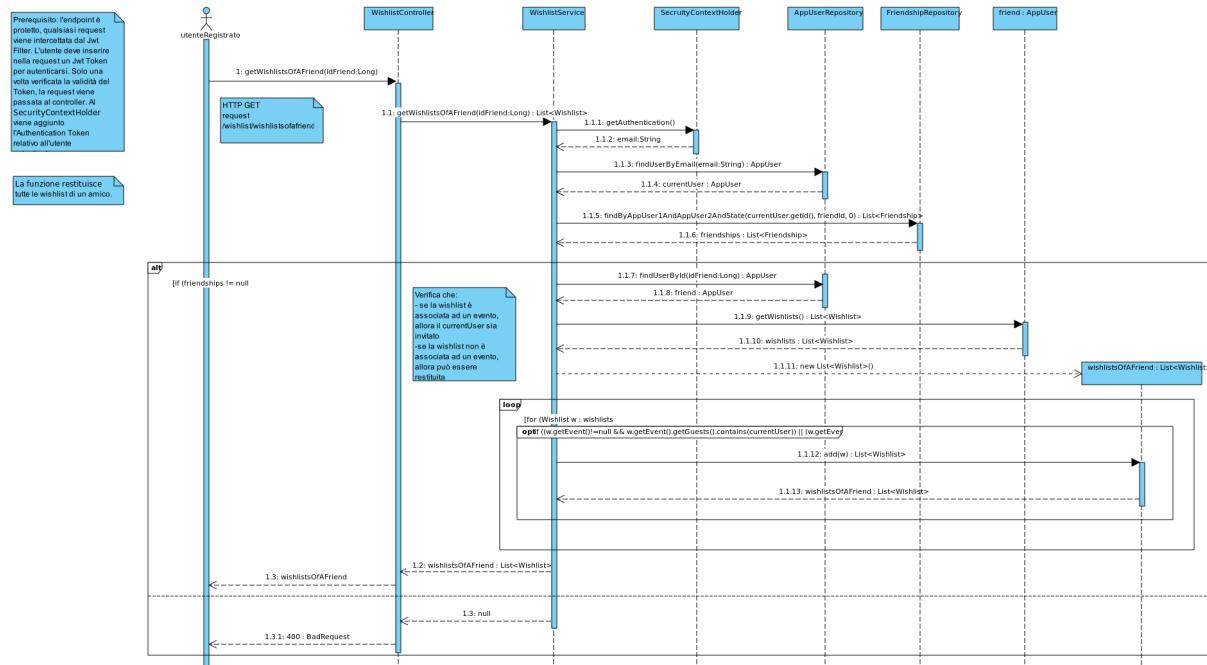


Figure 6.14: Sequence diagram della funzione `getWishlistOfAFriend`

### 6.1.15 buy

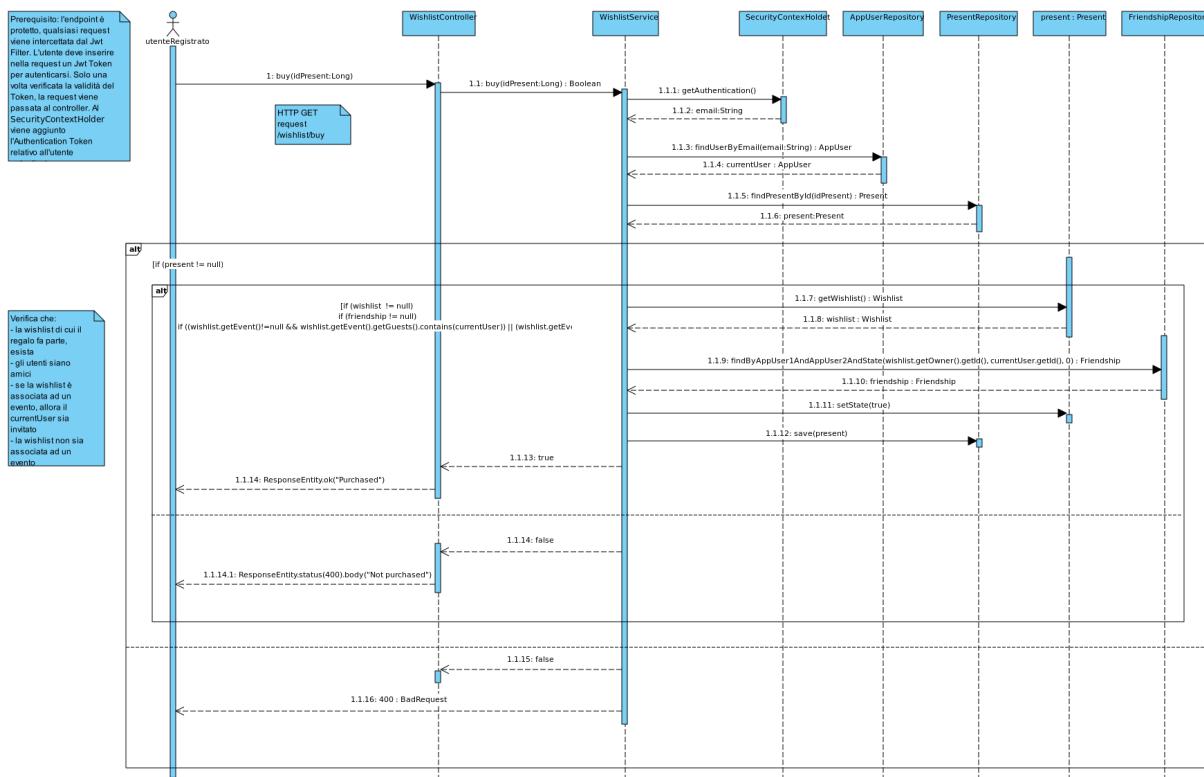


Figure 6.15: Sequence diagram della funzione `buy`

### 6.1.16 getEvent

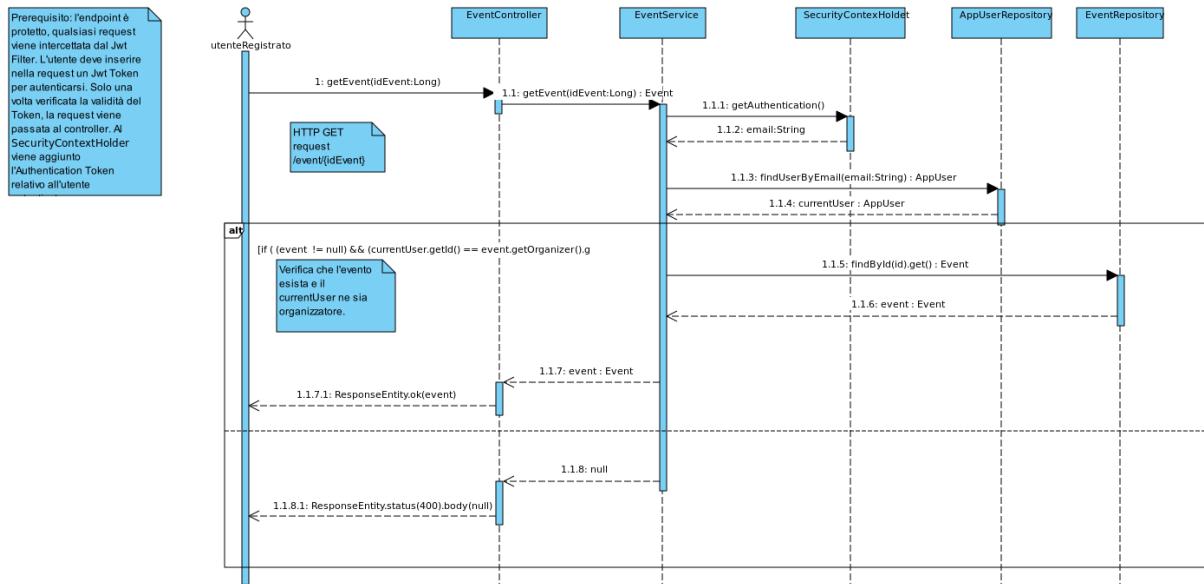


Figure 6.16: Sequence diagram della funzione *getEvent*

### 6.1.17 listEvents

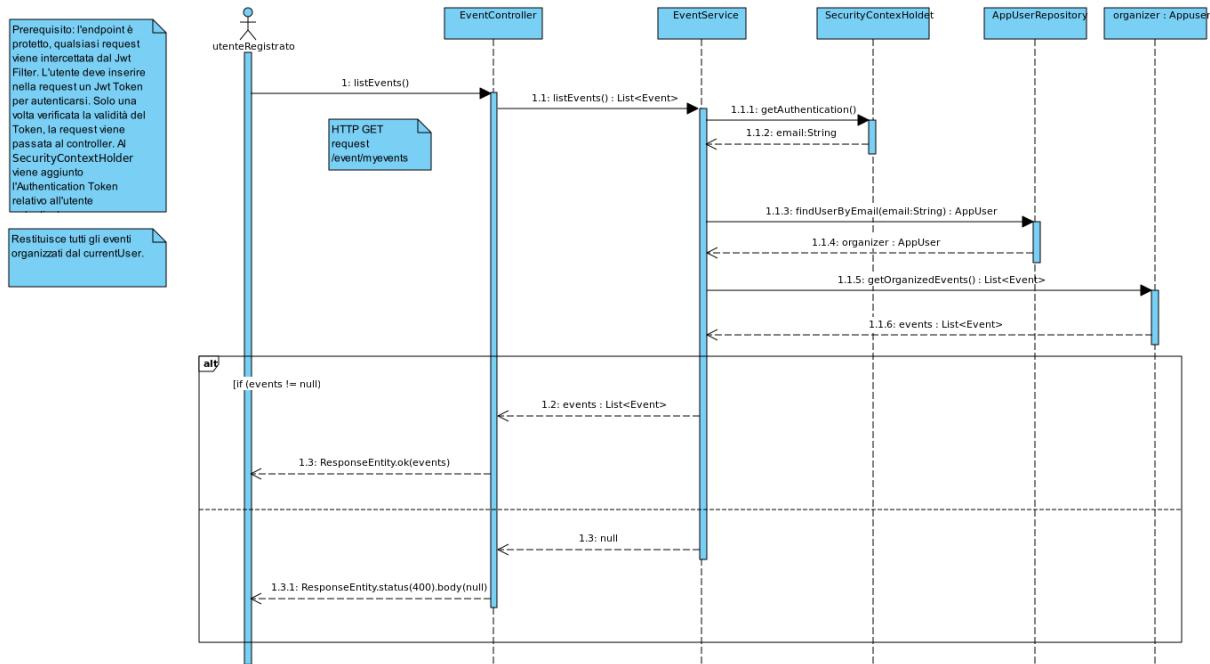


Figure 6.17: Sequence diagram della funzione *listEvents*

### 6.1.18 getInvitations

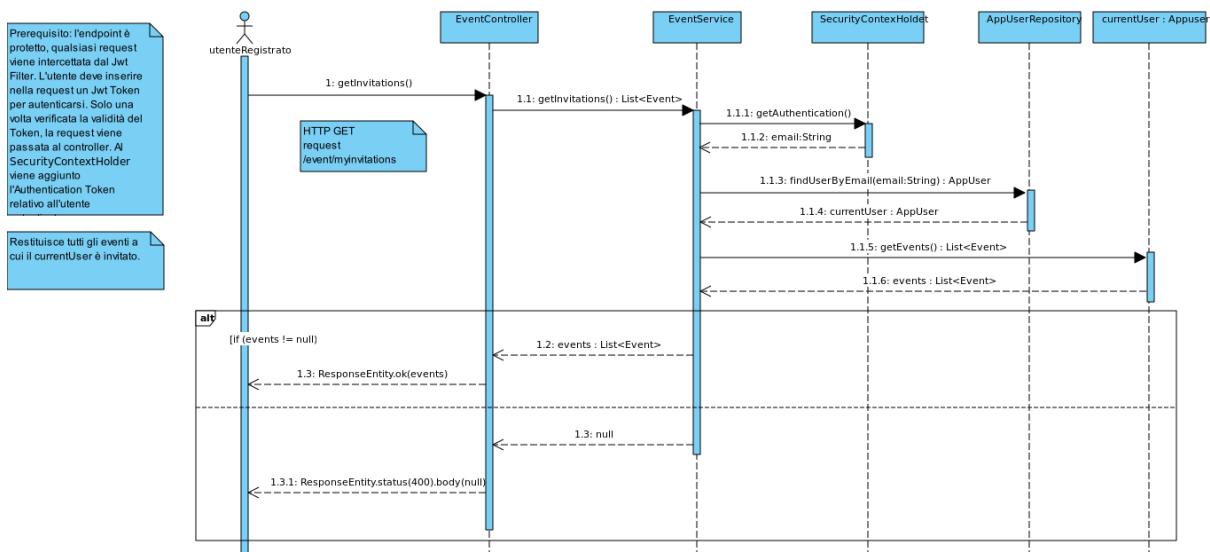


Figure 6.18: Sequence diagram della funzione *getInvitations*

### 6.1.19 getInvitationsByAFriend

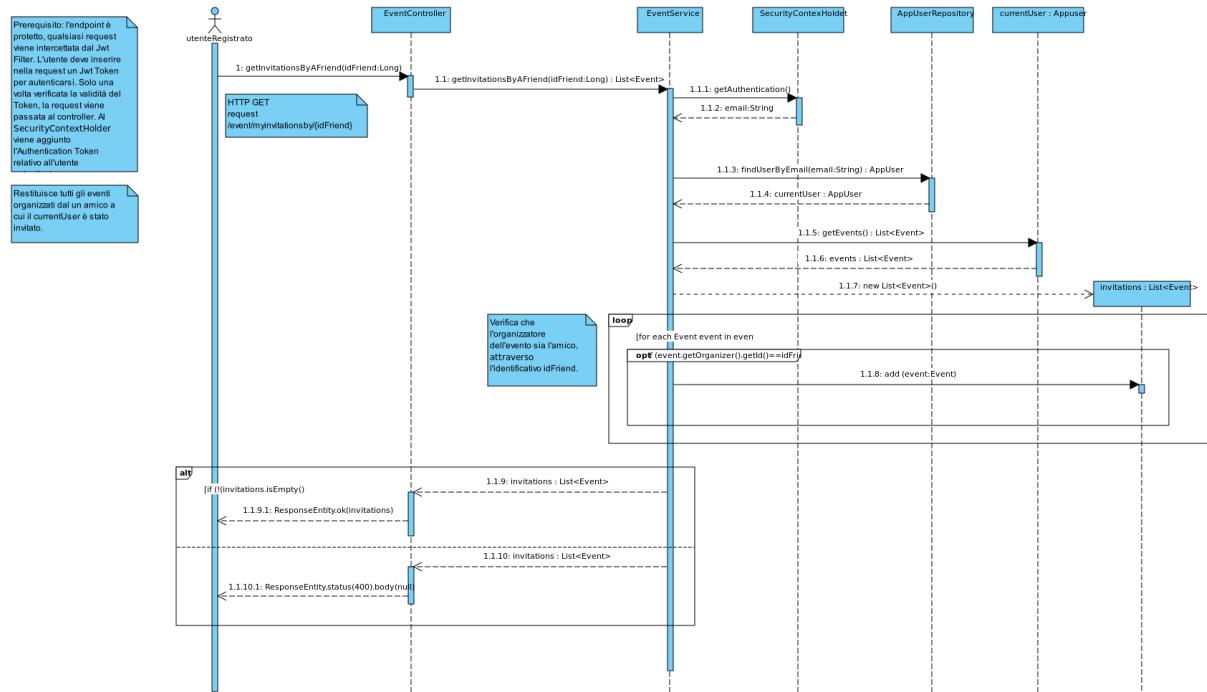


Figure 6.19: Sequence diagram della funzione `getInvitationsByAFriend`

### 6.1.20 getInvitation

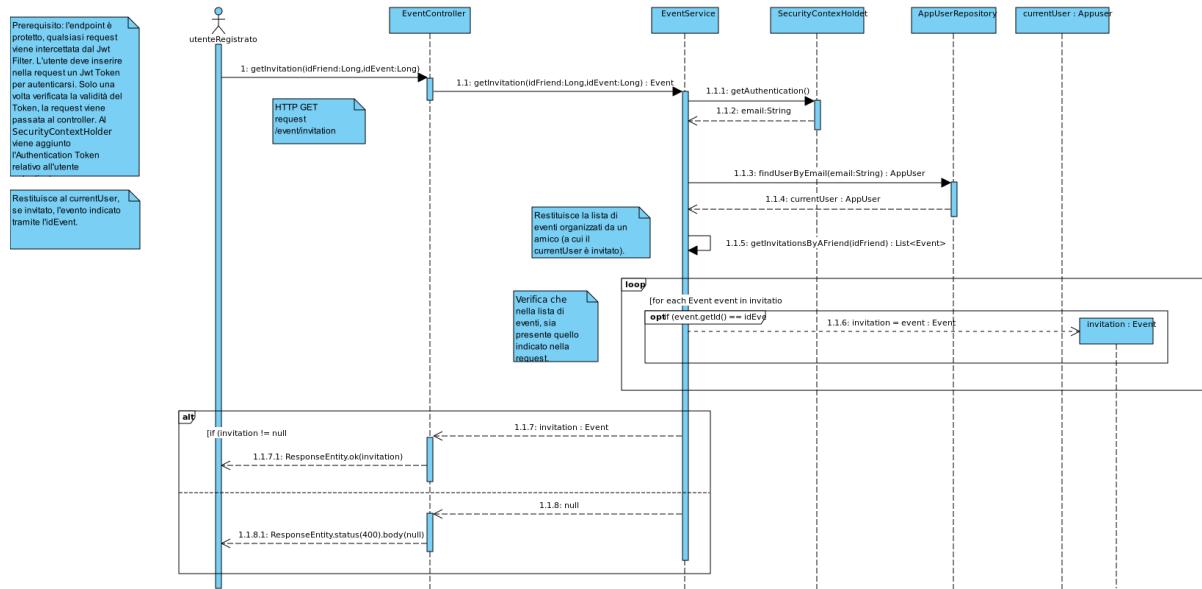


Figure 6.20: Sequence diagram della funzione `getInvitation`

### 6.1.21 invite

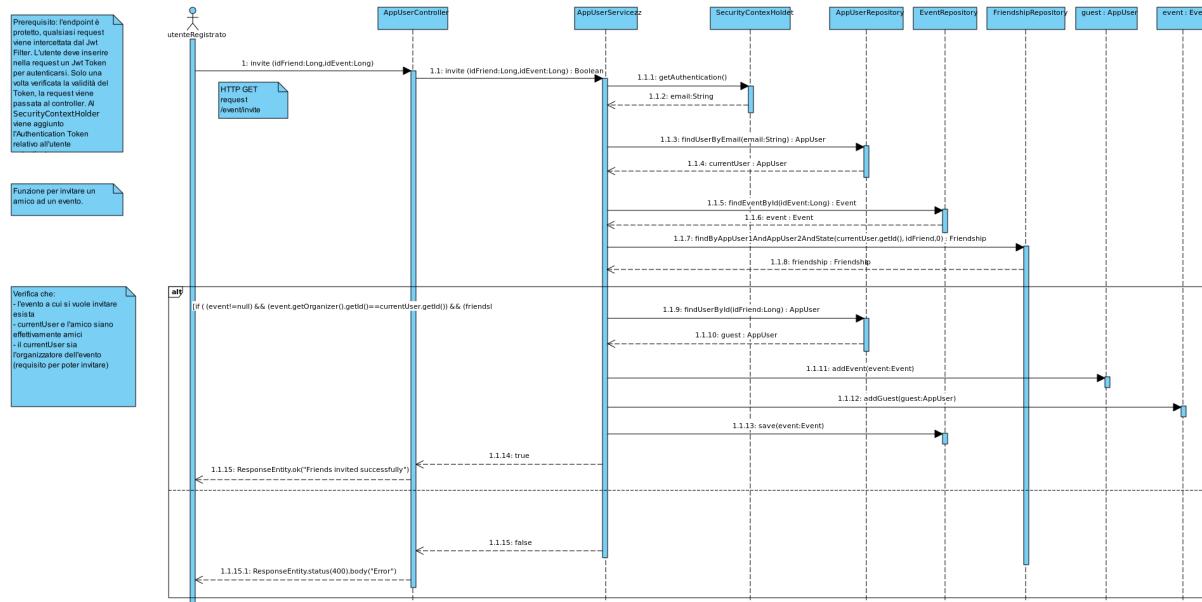


Figure 6.21: Sequence diagram della funzione `invite`

### 6.1.22 addFriend

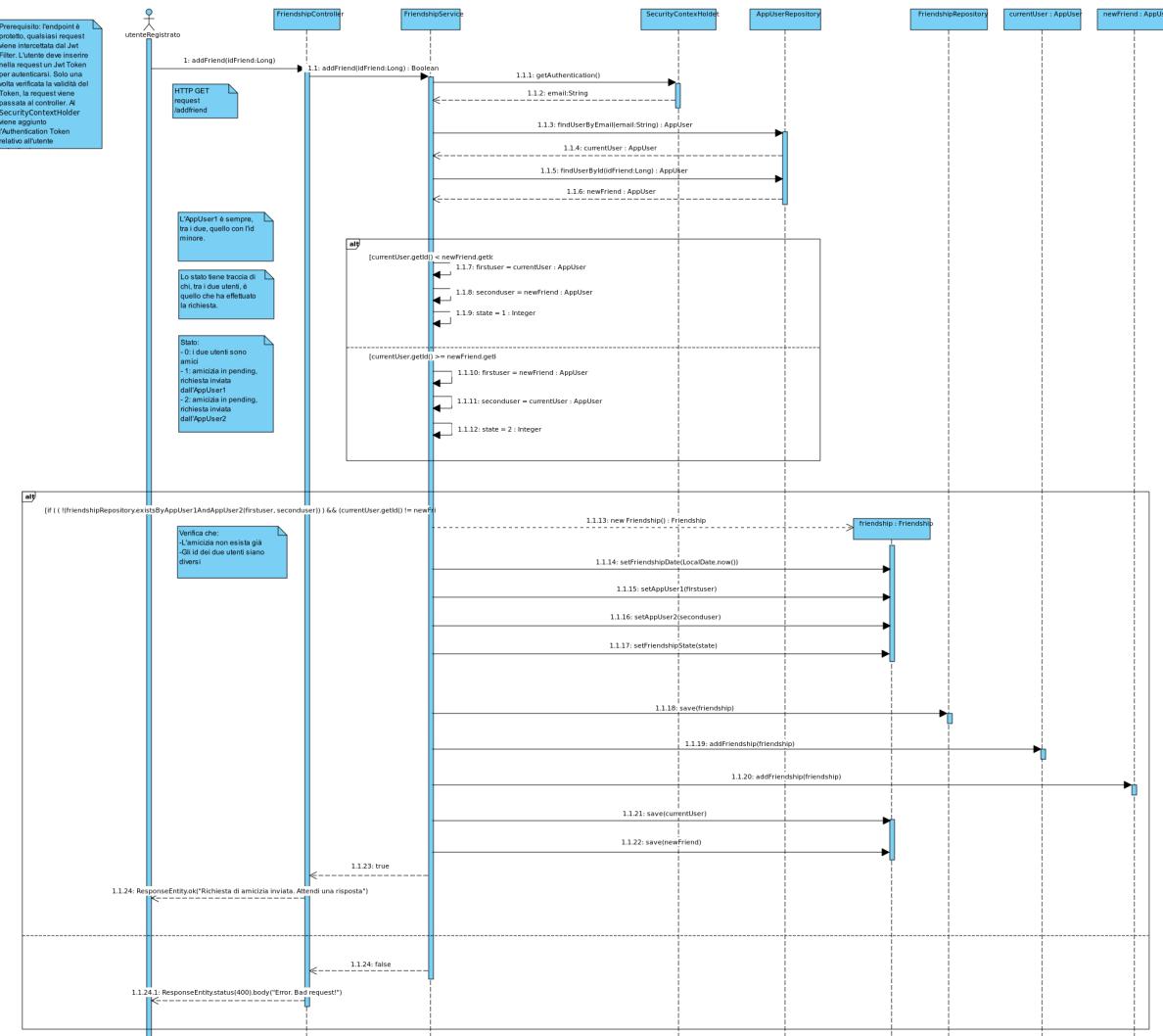


Figure 6.22: Sequence diagram della funzione `addFriend`

### 6.1.23 deleteFriend

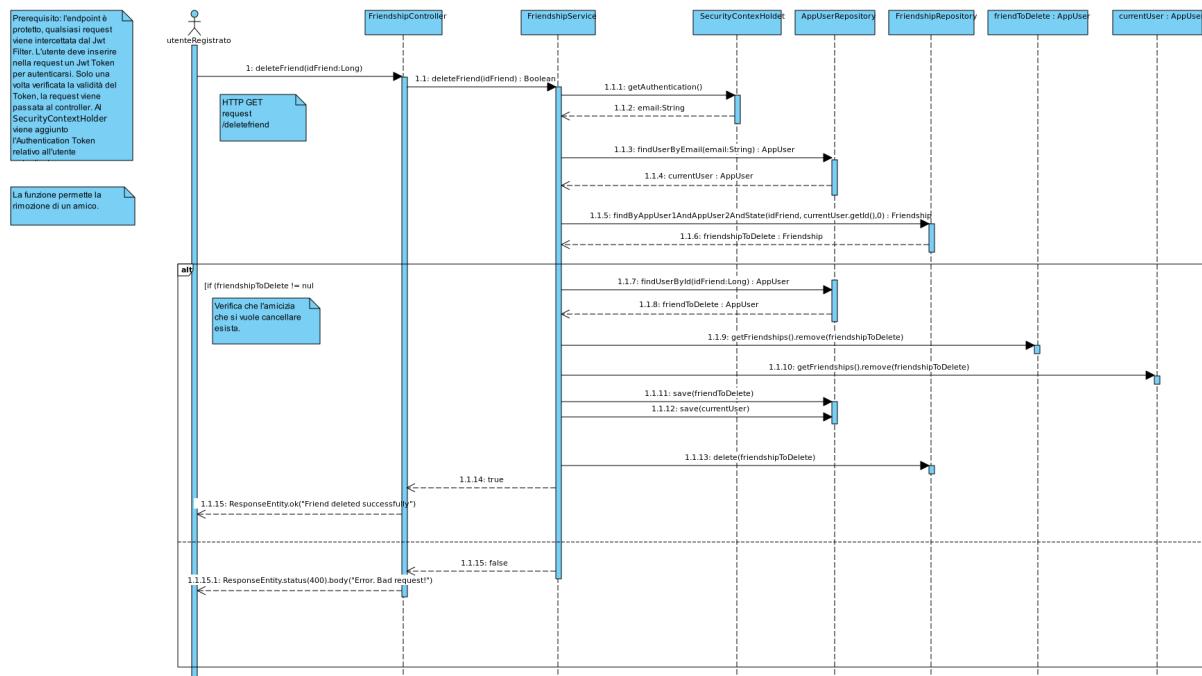


Figure 6.23: Sequence diagram della funzione *deleteFriend*

### 6.1.24 getFriends

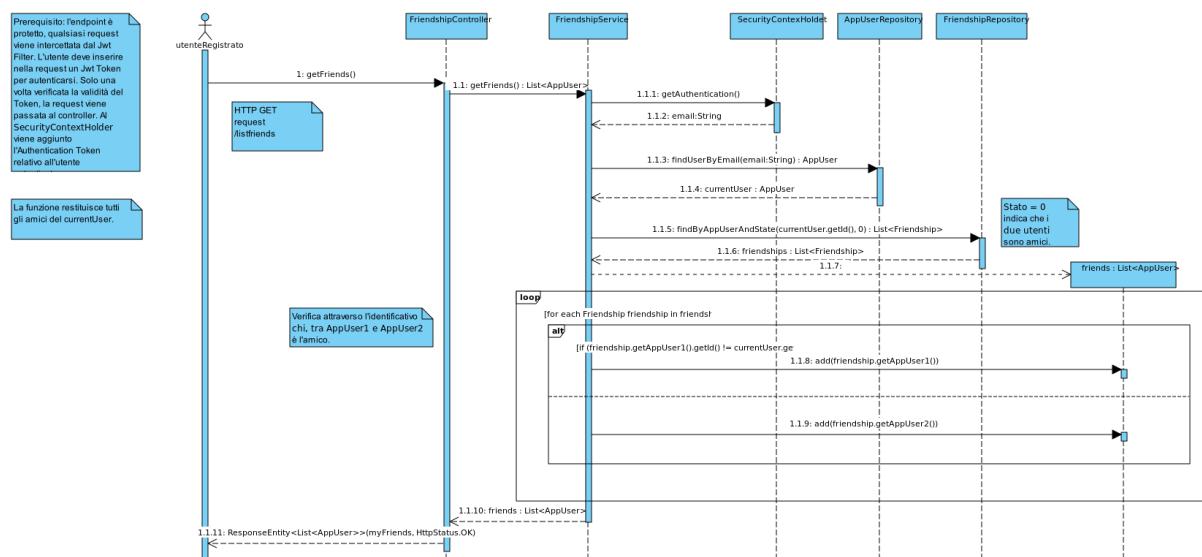


Figure 6.24: Sequence diagram della funzione *getFriends*

### 6.1.25 getPendingRequests

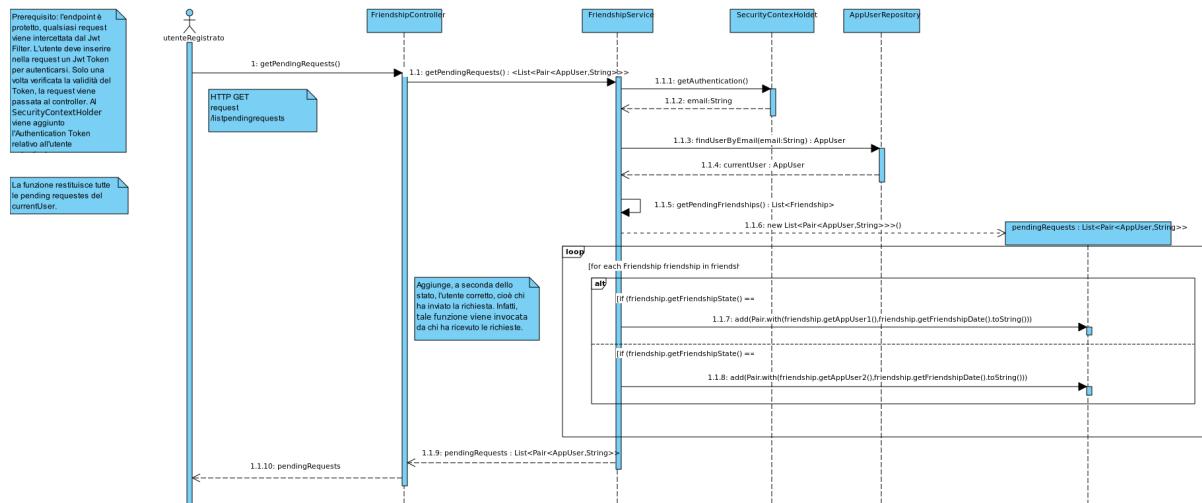


Figure 6.25: Sequence diagram della funzione `getPendingRequests`

### 6.1.26 setFriendship

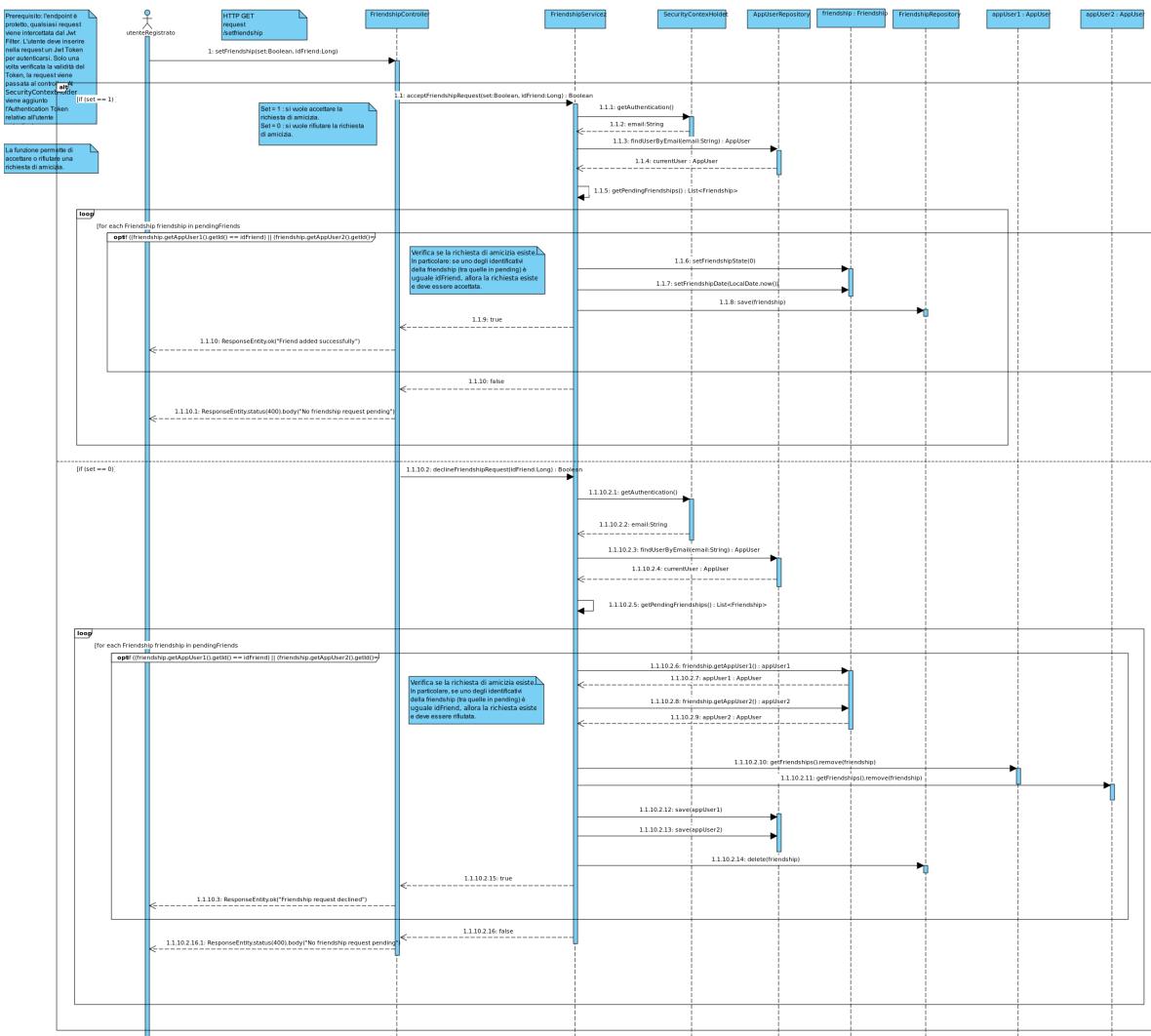


Figure 6.26: Sequence diagram della funzione `setFriendship`

Per chiarezza si è scelto di riportare anche i diagrammi `acceptFriendshipRequest` e `declineFriendshipRequest`.

## CHAPTER 6. DIAGRAMMI COMPORTAMENTALI

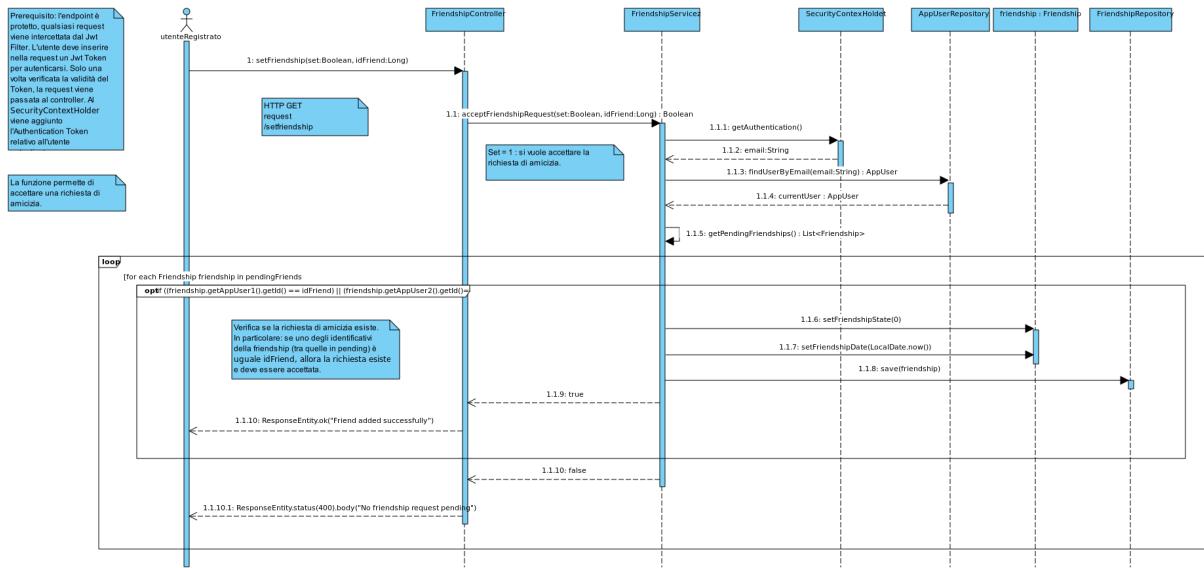


Figure 6.27: Sequence diagram della funzione *acceptFriendshipRequest*

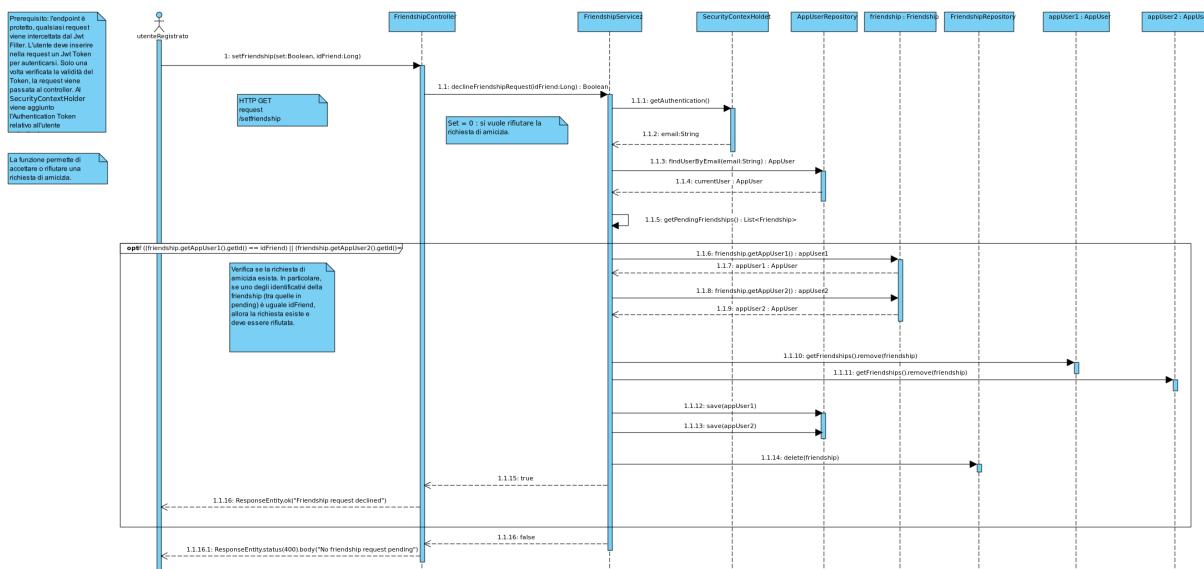


Figure 6.28: Sequence diagram della funzione *declineFriendshipRequest*

## 6.2 Flowchart

In questa sezione si andrà invece a riportare i diagrammi di flusso di alcune tipiche operazioni svolte da un utente che racchiudono in esse più casi d'uso.

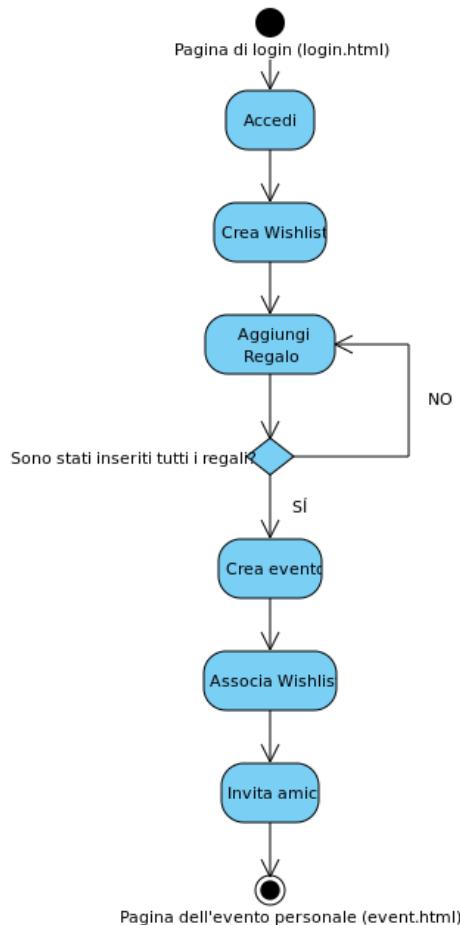


Figure 6.29: Creazione di un evento con una wishlist

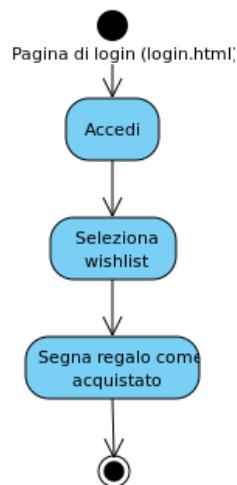


Figure 6.30: Indicare un oggetto di una wishlist di un amico come acquistato

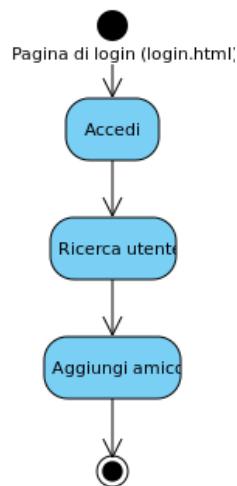


Figure 6.31: Invio di una richiesta di amicizia

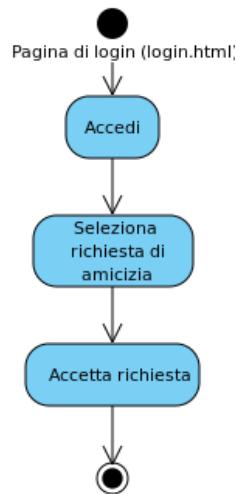


Figure 6.32: Accettare una richiesta di amicizia

## 6.3 Activity Diagram

Andiamo in questa sezione a descrivere tali operazioni più nel dettaglio con degli Activity Diagram, che riportano anche i parametri inseriti in input.

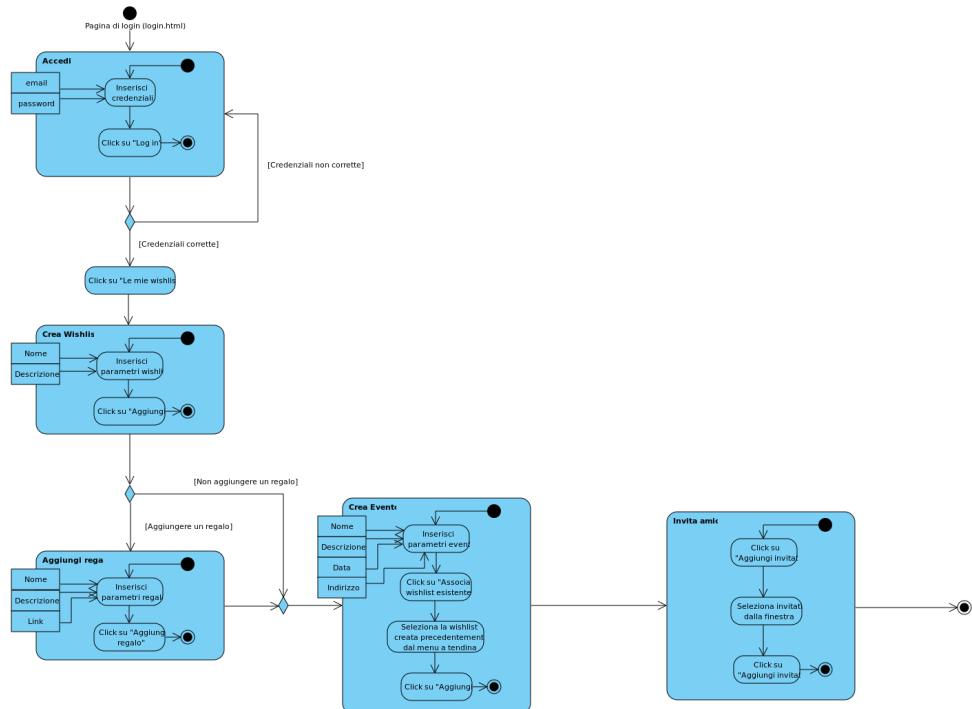


Figure 6.33: Creazione di un evento con una wishlist

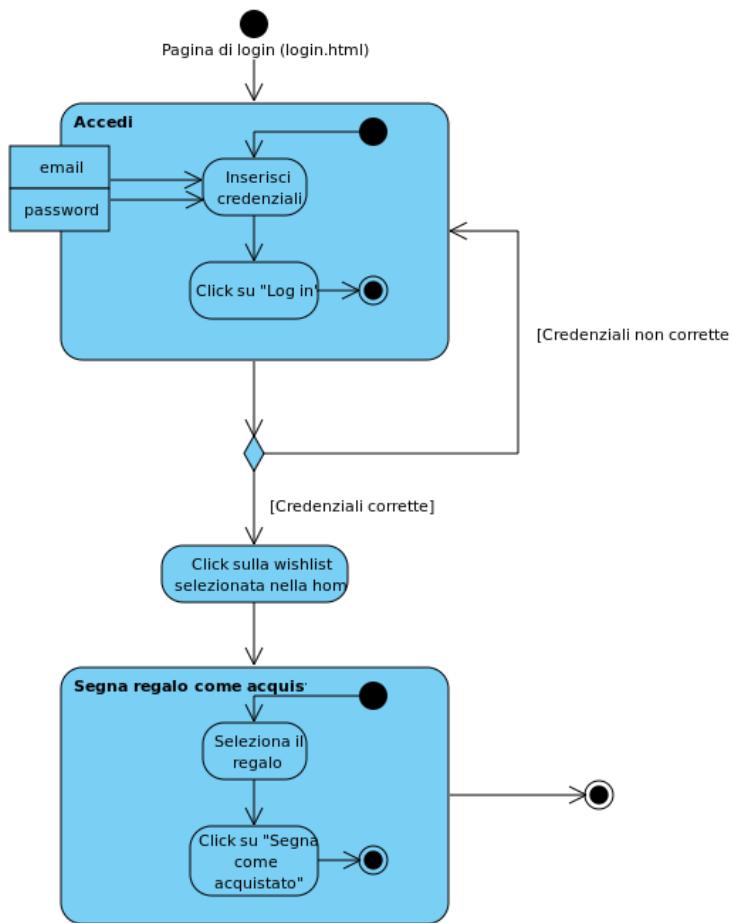


Figure 6.34: Indicare un oggetto di una wishlist di un amico come acquistato

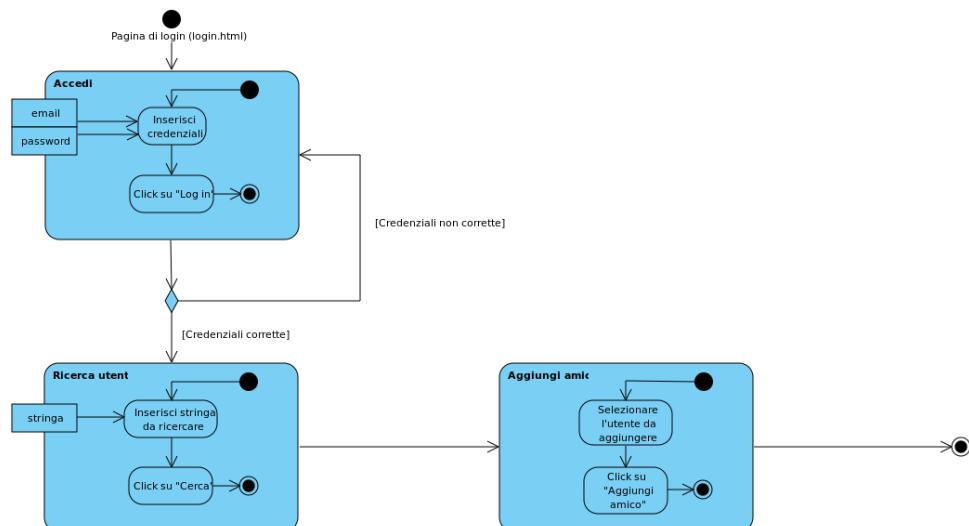


Figure 6.35: Invio di una richiesta di amicizia

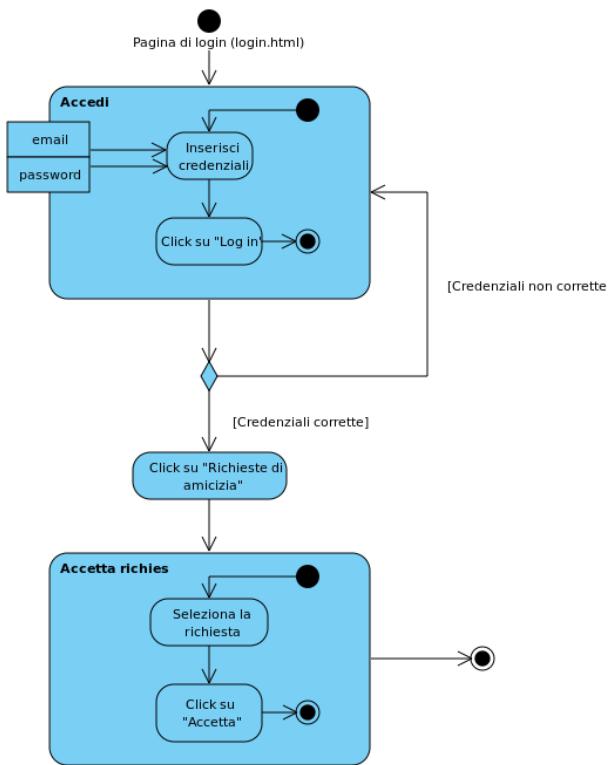


Figure 6.36: Accettare una richiesta di amicizia

# Chapter 7

## Rilascio e Testing

La fase finale del progetto include il rilascio dell'applicazione e il completamento dei test. Il rilascio consiste nella creazione del file *.war* utilizzando l'ambiente di sviluppo IntelliJ IDEA. Questo file è un archivio che racchiude il codice compilato dell'applicazione, file HTML, CSS e JS per il client, librerie esterne in formato jar e file di configurazione. Tra questi file vi sono:

- il file "pom.xml", che elenca le dipendenze del progetto Maven;
- il file "application.properties", che contiene le impostazioni di configurazione per Spring.

### 7.1 Deployment diagram

Di seguito si riporta il deployment diagram dell'applicazione. Client e server sono eseguiti sullo stesso PC su sistema operativo Windows. A fini esemplificativi, si è immaginato di eseguire database e server su due macchine

diverse. In realtà, a causa di mancanza di risorse, essi sono entrambi presenti sullo stesso PC. Il client è rappresentato da un web browser che comunica tramite protocollo HTTP con il server. Il server è dispiegato sul JSP Tomcat e comunica con il database tramite protocollo JDBC. L'artefatto ROOT.war in fase di esecuzione svolge il ruolo di server. Esso contiene anche le pagine HTML e i file Javascript utilizzati dal client e ottenuti tramite richieste HTTP. L'artefatto wishlistapp si manifesta come RDBMS. Si noti che l'applicazione è stata dispiegata sull'URL radice "localhost://"; per il deployment su un dominio diverso, è necessario andare a modificare il path di base dei file html.

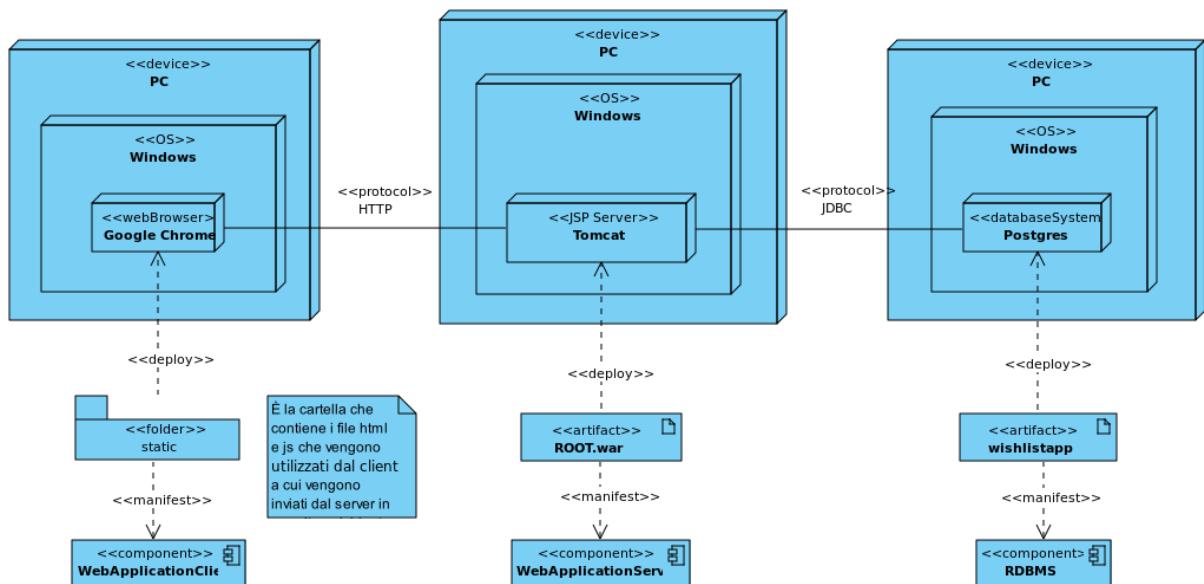


Figure 7.1: Deployment Diagram dell'applicazione

### 7.1.1 Install view

Si riporta qui il contenuto del file ROOT.war.

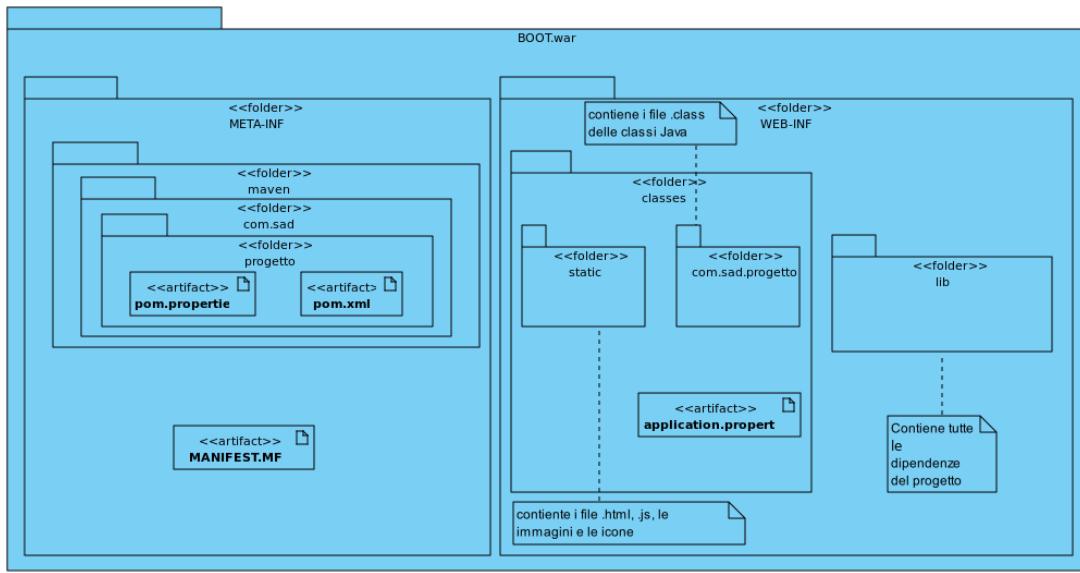


Figure 7.2: Install View

## 7.2 Testing d'unità

Per il testing d'unità si è deciso di usare JUnit, modulo con cui è possibile testare singolarmente i metodi di tutti le classi dell'applicazione. Nel caso in esame, i metodi di interesse da testare, sono i seguenti:

- Event `Event::addGuest(AppUser guest)`
- Event `Event::removeGuest(AppUser guest)`
- Wishlist `Wishlist::addPresent(Present present)`
- Wishlist `Wishlist::removePresent(Present present)`
- AppUser `AppUser::addEvent(Event event)`
- AppUser `AppUser::removeEvent(Event event)`

- AppUser AppUser::addWishlist(Wishlist wishlist)
- AppUser AppUser::removeWishlist(Wishlist wishlist)
- AppUser AppUser::addFriendship(Friendship friendship)
- AppUser AppUser::removeFriendship(Friendship friendship)

### **Event::addGuest**

Il metodo accetta in ingresso un utente *guest*, cioè un nuovo invitato all’evento. Restituisce *true* se l’utente è stato aggiunto all’evento con successo, *false* altrimenti. Si consideri un evento senza invitati e un utente. Se sull’evento si richiama il metodo add, che ha per ingresso l’utente, allora l’evento corrisponderà all’oggetto costruito con un invitato.

### **Event::removeGuest**

Il metodo accetta in ingresso un utente *guest*, cioè un invitato da rimuovere dall’evento. Restituisce *true* se l’utente è stato rimosso dall’evento con successo, *false* altrimenti. Si consideri un evento con un invitato e un utente da rimuovere. Se sull’evento si richiama il metodo remove, che ha per ingresso l’utente, allora l’evento corrisponderà all’oggetto costruito senza invitati.

---

```
package com.sad.progetto.event;

import com.sad.progetto.appUser.AppUser;
import org.junit.jupiter.api.Test;
```

```
import java.time.LocalDate;
import java.util.HashSet;
import java.util.Set;

import static org.junit.jupiter.api.Assertions.*;

class EventTest {

    @Test
    void addGuest() {
        // Un evento senza partecipanti, con name = ... ,
        // description = ... , date = ... , eventAddress = ...
        Event testEvent = new Event(new String("Name"), new
            String("Description"), LocalDate.now(), new
            String("Address"), null, new HashSet<>(), null );
        // Nuovo partecipante
        AppUser guest = new AppUser();
        testEvent.addGuest(guest);

        Event expectedEvent = new Event(new String("Name"), new
            String("Description"), LocalDate.now(), new
            String("Address"), null, Set.of(guest), null );
        assertEquals(expectedEvent, testEvent);
    }

    @Test
    void removeGuest() {
        // Un evento senza partecipanti, con name = ... ,

```

```
    description = ... , date = ... , eventAddress = ...  
  
Event testEvent = new Event(new String("Name"), new  
    String("Description"), LocalDate.now(), new  
    String("Address"), null, new HashSet<>(), null );  
  
// Nuovo partecipante  
  
AppUser guest = new AppUser();  
testEvent.addGuest(guest);  
  
testEvent.removeGuest(guest);  
  
Event expectedEvent = new Event(new String("Name"), new  
    String("Description"), LocalDate.now(), new  
    String("Address"), null, new HashSet<>(), null);  
assertEquals(expectedEvent, testEvent);  
  
}  
  
}
```

---

## Wishlist::addPresent

Il metodo accetta in ingresso un regalo *present*, cioè un nuovo regalo da aggiungere alla wishlist. Restituisce *true* se il regalo è stato aggiunto alla wishlist con successo, *false* altrimenti. Si consideri una wishlist senza regali e un regalo. Se sulla wishlist si richiama il metodo add, che ha per ingresso il regalo, allora la wishlist corrisponderà all'oggetto costruito con il regalo.

## Wishlist::removePresent

Il metodo accetta in ingresso un regalo *present*, cioè un regalo da rimuovere dalla wishlist. Restituisce *true* se il regalo è stato rimosso dalla wishlist con successo, *false* altrimenti. Si consideri una wishlist con un regalo e un regalo da rimuovere. Se sulla wishlist si richiama il metodo remove, che ha per ingresso il regalo, allora la wishlist corrisponderà all'oggetto costruito senza regali (vuoto).

---

```
package com.sad.progetto.wishlist;

import com.sad.progetto.present.Present;
import org.junit.jupiter.api.Test;

import java.util.HashSet;
import java.util.Set;

import static org.junit.jupiter.api.Assertions.*;

class WishlistTest {

    @Test
    void addPresent() {
        // Una wishlist senza regali, con nome = ...
        description = ... size = ... appuser = ...
        Wishlist testWishlist = new Wishlist(new
            String("Wishlist1"), new String("Description"),
            Integer.valueOf(0), null, null, new HashSet<>());
    }
}
```

```
// Un regalo con nome = ... description = ... link =
... state = ... wishlist = testWishlist
Present testPresent = new Present(new
String("Present1"), new String("Description"), new
String("Link"), Boolean.valueOf(false), testWishlist);
testWishlist.addPresent(testPresent);

Wishlist expectedWishlist = new Wishlist(new
String("Wishlist1"), new String("Description"),
Integer.valueOf(0), null, null, Set.of(testPresent));

assertEquals(expectedWishlist,testWishlist);
}

@Test
void removePresent() {
    // Una wishlist con nome = ... description = ... size =
    ... appuser = ...
    Wishlist testWishlist = new Wishlist(new
String("Wishlist1"), new String("Description"),
Integer.valueOf(0), null, null, new HashSet<>());
    // Un regalo con nome = ... description = ... link =
    ... state = ... wishlist = testWishlist
    Present present = new Present(new String("Present"),
new String("Description"), new String("Link"),
Boolean.valueOf(false), testWishlist);
    //Aggiunto regalo alla wishlist
    testWishlist.addPresent(present);
```

```
    testWishlist.removePresent(present);

    // Una wishlist senza regali con nome = ... description
    // = ... size = ... appuser = ...

    Wishlist expectedWishlist = new Wishlist(new
        String("Wishlist1"), new String("Description"),
        Integer.valueOf(0), null, null, new HashSet<>());

    assertEquals(expectedWishlist, testWishlist);
}

}
```

---

## AppUser::addEvent

Il metodo accetta in ingresso un evento *event*, cioè un nuovo evento da aggiungere all’utente, così da risultare invitato. Restituisce *true* se l’evento è stato aggiunto alla lista di eventi a cui è invitato l’utente con successo, *false* altrimenti. Si consideri un utente non invitato a nessun evento e un evento. Se sull’utente si richiama il metodo add, che ha per ingresso l’evento, allora l’utente corrisponderà all’oggetto costruito con già un evento a cui è invitato.

## AppUser::removeEvent

Il metodo accetta in ingresso un evento *event*, cioè un evento da rimuovere dalla lista di eventi a cui è invitato l’utente, così da risultare non invitato. Restituisce *true* se l’evento è stato rimosso dalla lista di eventi a cui è invitato l’utente con successo, *false* altrimenti. Si consideri un utente invitato ad un

evento e l'evento da cui si desidera rimuoverlo. Se sull'utente si richiama il metodo remove, che ha per ingresso l'evento, allora l'utente corrisponderà all'oggetto costruito con la lista di eventi a cui è invitato vuota.

---

```
@Test
```

```
void addEvent() {  
  
    //Un utente con name = ... , email = ... , password =  
  
    ...  
  
    AppUser testUser = new AppUser(new String("Name"), new  
        String("Email"), new String("Password"), new  
        HashSet<>(), new HashSet<>(), new HashSet<>());  
  
    //Un evento con name = ... , description = ... , date =  
    ... , eventAddress = ... ,  
  
    Event event = new Event(new String("Name"), new  
        String("Description"), LocalDate.now(), new  
        String("Address"), null, new HashSet<>(), null);  
  
    testUser.addEvent(event);
```

```
AppUser expectedUser = new AppUser(new String("Name"),  
    new String("Email"), new String("Password"), new  
    HashSet<>(), new HashSet<>(), Set.of(event));  
  
assertEquals(expectedUser, testUser);  
}
```

```
@Test
```

```
void removeEvent() {  
  
    //Un evento con name = ... , description = ... , date =  
    ... , eventAddress = ...
```

```
Event event = new Event(new String("Name"), new
String("Description"), LocalDate.now(), new
String("Address"), null, new HashSet<>(), null);
//Un utente con name = ... , email = ... , password =
...
AppUser testUser = new AppUser(new String("Name"), new
String("Email"), new String("Password"), new
HashSet<>(), new HashSet<>(), new HashSet<>());
testUser.addEvent(event);

testUser.removeEvent(event);

AppUser expectedUser = new AppUser(new String("Name"),
new String("Email"), new String("Password"), new
HashSet<>(), new HashSet<>(), new HashSet<>());
assertEquals(expectedUser, testUser);
}
```

---

## AppUser::addWishlist

Il metodo accetta in ingresso una wishlist *wishlist*, cioè una nuova wishlist di cui l'utente è proprietario. Restituisce *true* se la wishlist è stata aggiunta alla lista di wishlist dell'utente con successo, *false* altrimenti. Si consideri un utente senza wishlist associate. Se sull'utente si richiama il metodo add, che ha per ingresso la wishlist, allora l'utente corrisponderà all'oggetto costruito con già una wishlist associata.

## AppUser::removeWishlist

Il metodo accetta in ingresso una wishlist *wishlist*, cioè una wishlist che si vuole eliminare dalla lista di wishlist di cui l'utente è proprietario. Restituisce *true* se la wishlist è stata eliminata con successo, *false* altrimenti. Si consideri un utente con una wishlist associata. Se sull'utente si richiama il metodo remove, che ha per ingresso la wishlist, allora l'utente corrisponderà all'oggetto costruito senza wishlist associate.

---

```
@Test
```

```
void addWishlist() {  
    //Un utente con name = ... , email = ... , password =  
    ...  
    AppUser testUser = new AppUser(new String("Name"), new  
        String("Email"), new String("Password"), new  
        HashSet<>(), new HashSet<>(), new HashSet<>(), new  
        HashSet<>());  
    // Una wishlist con nome = ... description = ... size =  
    ... appuser = ..., size = ...  
    Wishlist wishlist = new Wishlist(new  
        String("Wishlist1"), new String("Description"),  
        Integer.valueOf(0), null, null, new HashSet<>());  
  
    testUser.addWishlist(wishlist);  
  
    AppUser expectedUser = new AppUser(new String("Name"),  
        new String("Email"), new String("Password"), new  
        HashSet<>(), new HashSet<>(), new HashSet<>(),  
        new HashSet<>());
```

```
        Set.of(wishlist));  
  
        assertEquals(expectedUser, testUser);  
    }  
  
  
    @Test  
    void removeWishlist() {  
  
        //Un utente con name = ... , email = ... , password =  
  
        ...  
  
        AppUser testUser = new AppUser(new String("Name"), new  
            String("Email"), new String("Password"), new  
            HashSet<>(), new HashSet<>(), new HashSet<>(), new  
            HashSet<>());  
  
        // Una wishlist con nome = ... description = ... size =  
  
        ... appuser = ...  
  
        Wishlist wishlist = new Wishlist(new  
            String("Wishlist1"), new String("Description"),  
            Integer.valueOf(0), null, null, new HashSet<>());  
  
        testUser.addWishlist(wishlist);  
  
  
        testUser.removeWishlist(wishlist);  
  
  
        AppUser expectedUser = new AppUser(new String("Name"),  
            new String("Email"), new String("Password"), new  
            HashSet<>(), new HashSet<>(), new HashSet<>(), new  
            HashSet<>());  
  
        assertEquals(expectedUser, testUser);  
    }

---


```

## AppUser::addFriendship

Il metodo accetta in ingresso una amicizia *friendship*, cioè una nuova amicizia dell’utente. Restituisce *true* se la friendship è stata aggiunta all’utente con successo, *false* altrimenti. Si consideri un utente senza amicizie associate. Se sull’utente si richiama il metodo add, che ha per ingresso la friendship, allora l’utente corrisponderà all’oggetto costruito con già una friendship associata.

## AppUser::removeFriendship

Il metodo accetta in ingresso una amicizia *friendship*, cioè una amicizia dell’utente che si deve rimuovere. Restituisce *true* se la friendship è stata rimossa dall’utente con successo, *false* altrimenti. Si consideri un utente con una friendship associata. Se sull’utente si richiama il metodo remove, che ha per ingresso la friendship, allora l’utente corrisponderà all’oggetto costruito senza friendship associate.

---

```
@Test
```

```
void addFriendship() {  
    //Un utente con name = ... , email = ... , password =  
    ...  
    AppUser testUser = new AppUser(new String("Name"), new  
        String("Email"), new String("Password"), new  
        HashSet<>(), new HashSet<>(), new HashSet<>(), new  
        HashSet<>());  
    // Un’amicizia, con id = ... , appuser1 = ... ,  
    appuser2 = ... , friendshipDate = ... ,  
    friendshipState = ...
```

```
Friendship friendship = new Friendship(Long.valueOf(0),  
    new AppUser(), new AppUser(), LocalDate.now(),  
    Integer.valueOf(0));  
  
testUser.addFriendship(friendship);  
  
AppUser expectedUser = new AppUser(new String("Name"),  
    new String("Email"), new String("Password"),  
    Set.of(friendship), new HashSet<>(), new HashSet<>(),  
    new HashSet<>());  
  
assertEquals(expectedUser, testUser);  
}  
  
@Test  
void removeFriendship() {  
    // Un utente con name = ... , email = ... , password =  
    ...  
    AppUser testUser = new AppUser(new String("Name"), new  
        String("Email"), new String("Password"), new  
        HashSet<>(), new HashSet<>(), new HashSet<>(), new  
        HashSet<>());  
    // Un' amicizia, con id = ... , appuser1 = ... ,  
    // appuser2 = ... , friendshipDate = ... ,  
    // friendshipState = ...  
    Friendship friendship = new Friendship(Long.valueOf(0),  
        new AppUser(), new AppUser(), LocalDate.now(),  
        Integer.valueOf(0));  
    testUser.addFriendship(friendship);
```

```
testUser.removeFriendship(friendship);

AppUser expectedUser = new AppUser(new String("Name"),
    new String("Email"), new String("Password"), new
    HashSet<>(), new HashSet<>(), new HashSet<>(), new
    HashSet<>());
assertEquals(expectedUser,testUser);

}
```

---

## 7.3 Testing d'integrazione

Nel test di integrazione si va a verificare il comportamento dei moduli combinati tra loro. Nel nostro caso andremo per prima cosa a testare il backend della nostra applicazione per poi andare a testare l'intero sistema attraverso l'interfaccia grafica offerta dal client.

### 7.3.1 Test backend

Per testare il server si sono andate a generare ed inviare delle richieste *HTTP* tramite Postman. In particolare, è stato verificato il corretto funzionamento di tutte le funzionalità offerte dal server. Sono inoltre stati effettuati anche dei test per verificare il corretto funzionamento delle richieste anche in caso di input diversi dal caso base.

Tali test sono riportati nelle seguenti tabelle.

## CHAPTER 7. RILASCIO E TESTING

SAD / Add Friend

GET <http://127.0.1:8080/addFriend?friendId=2>

Params	Authorization	Headers (8)	Body	Pre-request Script	Tests	Settings
<input checked="" type="checkbox"/> Postman-Token <a href="#">ⓘ</a>		<calculated when request is sent>				
<input checked="" type="checkbox"/> Host <a href="#">ⓘ</a>		<calculated when request is sent>				
<input checked="" type="checkbox"/> User-Agent <a href="#">ⓘ</a>		PostmanRuntime/7.30.1				
<input checked="" type="checkbox"/> Accept <a href="#">ⓘ</a>		*/*				
<input checked="" type="checkbox"/> Accept-Encoding <a href="#">ⓘ</a>		gzip, deflate, br				
<input checked="" type="checkbox"/> Connection <a href="#">ⓘ</a>		keep-alive				
<input checked="" type="checkbox"/> Set-Cookie		access_token=eyJhbGciOiJIUzI1NiJ9eyJzdWliOiJoYW1z...				
Key		Value			Description	

Body Cookies (1) Headers (11) Test Results Status: 200 OK

Pretty Raw Preview Visualize Text [JSON](#)

```
1 Richiesta di amicizia inviata. Attendi una risposta
```

Figure 7.3: Test richiesta di amicizia

SAD / Create Wishlist

GET <http://127.0.1:8080/wishlist/create?name=Compleanno%202023&description=La%20wishlist%20del%20mio%20Compleanno>

Params	Authorization	Headers (8)	Body	Pre-request Script	Tests	Settings
<input checked="" type="checkbox"/> Postman-Token <a href="#">ⓘ</a>		<calculated when request is sent>				
<input checked="" type="checkbox"/> Host <a href="#">ⓘ</a>		<calculated when request is sent>				
<input checked="" type="checkbox"/> User-Agent <a href="#">ⓘ</a>		PostmanRuntime/7.30.1				
<input checked="" type="checkbox"/> Accept <a href="#">ⓘ</a>		*/*				
<input checked="" type="checkbox"/> Accept-Encoding <a href="#">ⓘ</a>		gzip, deflate, br				
<input checked="" type="checkbox"/> Connection <a href="#">ⓘ</a>		keep-alive				
<input checked="" type="checkbox"/> Set-Cookie		access_token=eyJhbGciOiJIUzI1NiJ9eyJzdWliOiJtYXJpb...				
Key		Value			Description	

Body Cookies (1) Headers (11) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON

```
2
3   "id": 1,
4   "name": "Compleanno 2023",
5   "description": "La wishlist del mio compleanno",
6   "size": null,
7   "event": null,
8   "owner": {
9     "id": 1,
10    "username": "Mario Rossi",
11    "email": "mario@email.com"
12  },
13  "presents": null
```

Figure 7.4: Test creazione di una wishlist

## CHAPTER 7. RILASCIO E TESTING

The screenshot shows a Postman test for inviting guests to an event. The request is a POST to `http://127.0.1:8080/event/invite`. The body is JSON with fields `idFriend` and `idEvent`. The response status is 200 OK, and the body contains the message "Friends invited successfully".

```
POST http://127.0.1:8080/event/invite
Body
1
2 ... "idFriend": [2, 3],
3 ... "idEvent": 1
4
```

Status: 200 OK

Pretty Raw Preview Visualize Text

1 Friends invited successfully

Figure 7.5: Test invito ad un evento

The screenshot shows a Postman test for logging in. The request is a POST to `http://127.0.0.1:8080/login`. The body is JSON with fields `email` and `password`. The response status is 200 OK, and it includes a cookie table with one entry: `access_token` with value `eyJhbGciOiJI...` .

```
POST http://127.0.0.1:8080/login
Body
1
2 ... "email":"mario@email.com",
3 ... "password":"password"
4
```

Status: 200 OK

Name	Value	Domain	Path	Expires
access_token	eyJhbGciOiJI...	127.0.0.1	/	Session

Figure 7.6: Test login

## CHAPTER 7. RILASCIO E TESTING

The screenshot shows a Postman test environment for a 'Register' endpoint. The request method is POST, and the URL is `http://127.0.0.1:8080/register`. The 'Body' tab is selected, showing a JSON payload:

```
1 {"username": "Mario Rossi",  
2 "email": "mario@email.com",  
3 "password": "password"}  
4  
5
```

The response status is 200 OK, and the body of the response is 'Registered'.

Figure 7.7: Test registrazione

<b>Test</b>	<b>Funzionalità</b>	<b>Input</b>	<b>Output atteso</b>	<b>Esito</b>
1	Login	Credenziali errate	Login fallito	OK
2	Registrazione	Email già presente	Registrazione fallita	OK
3	Cerca utente	Combinazione di caratteri	Lista di utenti contenenti tale combinazione	OK
4	Visualizza Eventi		Lista degli eventi a cui si è stati invitati	OK
5	Visualizza Wishlists		Lista delle wishlist degli amici senza le wishlist associate ad eventi a cui si è stati invitati	OK
6	Aggiunta amico	Utente già amico	Richiesta fallita	OK
7	Aggiunta regalo	Wishlist non propria	Aggiunta fallita	OK
8	Visualizza evento	Evento a cui non si è stati invitati	Richiesta fallita	OK
9	Visualizza Wishlist	Wishlist non di un amico	Richiesta fallita	OK
10	Visualizza Wishlist	Wishlist di un amico associata ad un evento a cui non si è stati invitati	Richiesta fallita	OK
11	Richiesta di una risorsa protetta	GET/POST senza JWT	Accesso negato	OK

## CHAPTER 7. RILASCIO E TESTING

---

Test	Funzionalità	API	Stato	Tipo di ritorno	Esito
1	Registrazione	/register	200	String	OK
2	Accesso	/login	200	String+Cookie	OK
3	Ricerca utente	/search	200	List<AppUser>(JSON)	OK
4	Creazione wishlist	/wishlist/create	200	Wishlist (JSON)	OK
5	Rimozione wishlist	/wishlist/delete	200	NULL	OK
6	Visualizza una tua wishlist	/wishlist/{id}	200	Wishlist (JSON)	OK
7	Visualizza tutte le tue wishlist	/wishlist/all	200	List<Wishlist>(JSON)	OK
8	Aggiungi un regalo alla tua wishlist	/wishlist/{id}/add	200	Present (JSON)	OK
9	Rimuovi un regalo dalla tua wishlist	/wishlist/{wishlistId}/delete/{presentId}	200	String	OK
10	Visualizza la wishlist di un amico	/wishlist/{id}/friend	200	Wishlist (JSON)	OK
11	Visualizza tutte le wishlist di un amico	/wishlist/wishlistsofafriend	200	List<Wishlist>(JSON)	OK
12	Visualizza le wishlist di tutti i tuoi amici	/wishlist/friendswishlist	200	List<Wishlist>(JSON)	OK
13	Segna un regalo come acquistato	/wishlist/buy	200	String	OK
14	Crea un evento	/event/create	200	Event (JSON)	OK
15	Elimina un evento	/event/delete/{id}	200	String	OK
16	Visualizza un tuo evento	/event/{id}	200	Event (JSON)	OK
17	Visualizza tutti gli eventi organizzati da te	/event/myevents	200	List<Event>(JSON)	OK
18	Visualizza tutti gli eventi a cui si è stati invitati	/event/myinvitations	200	List<Event>(JSON)	OK
19	Visualizza tutti gli eventi di un amico a cui si è invitati	/event/myinvitationsby/{idFriend}	200	List<Event>(JSON)	OK
20	Visualizza un evento a cui si è invitati	/event/invitation	200	Event (JSON)	OK
21	Invita amici ad un evento	/event/invite	200	String	OK
22	Invia richiesta di amicizia	/addFriend	200	String	OK
23	Rimuovi amico	/deletefriend	200	String	OK
24	Visualizza tutti i tuoi amici	/listFriends	200	List<AppUser>(JSON)	OK
25	Visualizza le tue richieste di amicizia	/listPendingRequests	200	List<Pair<AppUser, String>(JSON)	OK
26	Accetta/Rifiuta richiesta di amicizia	/setFriendship	200	String	OK
27	Visualizza utente	/user	200	AppUser (JSON)	OK
28	Visualizza utente corrente	/currentuser	200	AppUser (JSON)	OK

### 7.3.2 Test intero sistema tramite GUI

Sono riportati nella seguente tabella i test effettuati attraverso l'interfaccia grafica fornita dal client.

Test	Funzionalità	Azione	Input	Output	Esito
1	Cerca utente	Click su "cerca"	"qwer"	Nessun utente	OK
2	Cerca utente	Click su "cerca"	"marco"	appUser "Marco Salvi"	OK
3	Aggiungi un utente	Clck su "Aggiungi"	Utente già amico	Errore	OK
4	Aggiungi un utente	Aggiunta di un utente	Utente non amico	Conferma di invio richiesta	OK
5	Visualizza eventi degli amici	Clck su "Eventi"		Lista di eventi a cui si è invitati	OK
6	Crea wishlist	Clck su "Aggiungi"	"", ""	Errore	OK
7	Crea wishlist	Clck su "Aggiungi"	"Wishlist", "descrizione"	Nuova wishlist	OK
8	Aggiungi regalo	Clck su "Aggiungi regalo"	"", "", "", "	Errore	OK
9	Aggiungi regalo	Clck su "Aggiungi regalo"	"Regalo", "descrizione", "link"	Wishlist con nuovo regalo	OK
10	Accetta richiesta	Clck su "Accetta"		Conferma nuova amicizia	OK
11	Rifiuta richiesta	Clck su "Rifiuta"		Conferma eliminazione richiesta	OK
12	Rimuovi amico	Clck su "Rimuovi amico"		Conferma rimozione amicizia	OK
13	Crea evento	Clck su "Aggiungi"	"", "", "", "", "	Errore	OK
14	Crea evento	Clck su "Aggiungi"	"evento", "desc", "2024-01-01", "indirizzo"	Nuovo evento con wishlist autogenerata	OK
15	Crea evento	Clck su "Aggiungi" e selezione di una wishlist	"evento", "desc", "2024-01-01", "indirizzo"	Nuovo evento con wishlist scelta	OK
16	Aggiungi invitati	Clck su "Aggiungi invitati" e selezione degli utenti		Evento con nuovi invitati	OK

# **Chapter 8**

## **Manuale Utente**

Il presente manuale ha lo scopo di introdurre i concetti fondamentali dell'utilizzo dell'applicazione agli utenti.

### **8.1 Pagina di login (login.html)**

In questa pagina l'utente può registrarsi per la prima volta con username, email e password oppure effettuare l'accesso.

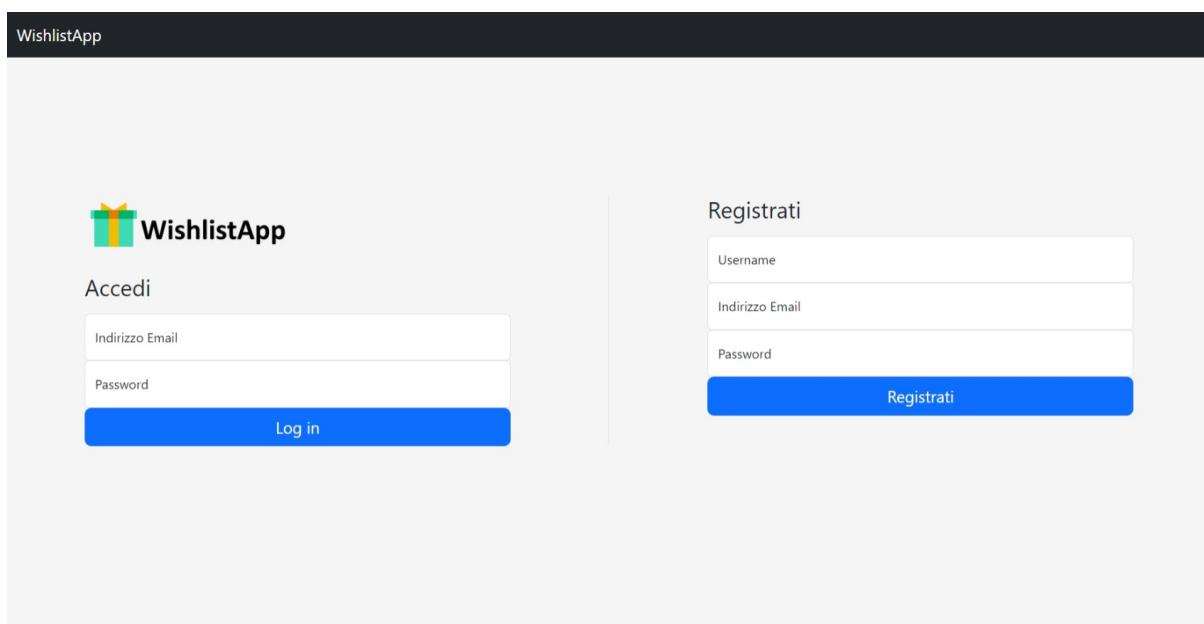


Figure 8.1: login.html

## 8.2 Navbar

Un elemento fondamentale per la navigazione attraverso la web application è la Navbar superiore che permette in qualsiasi momento di visitare la pagina del profilo utente, la pagina "Le mie Wishlist" e la pagina "I miei eventi". In più, permette agli utenti connessi di poter effettuare il logout e al centro è presente un box per cercare altri utenti all'interno dell'applicazione, così da potergli inviare una richiesta di amicizia.



Figure 8.2: Barra di navigazione quando estesa

## 8.3 Pagina Home (index.html)

Nella Homepage l'utente può visualizzare le wishlist dei propri amici. Inoltre è presente un menù laterale che permette di:

- visitare la pagina Eventi (events.html), contenente tutti gli eventi a cui si è stati invitati dai propri amici.
- visitare la pagina Amici (friends.html), contenente la lista degli amici.
- visitare la pagina Richieste di amicizia (friendsrequests.html), contenente tutte le richieste di amicizia ricevute.

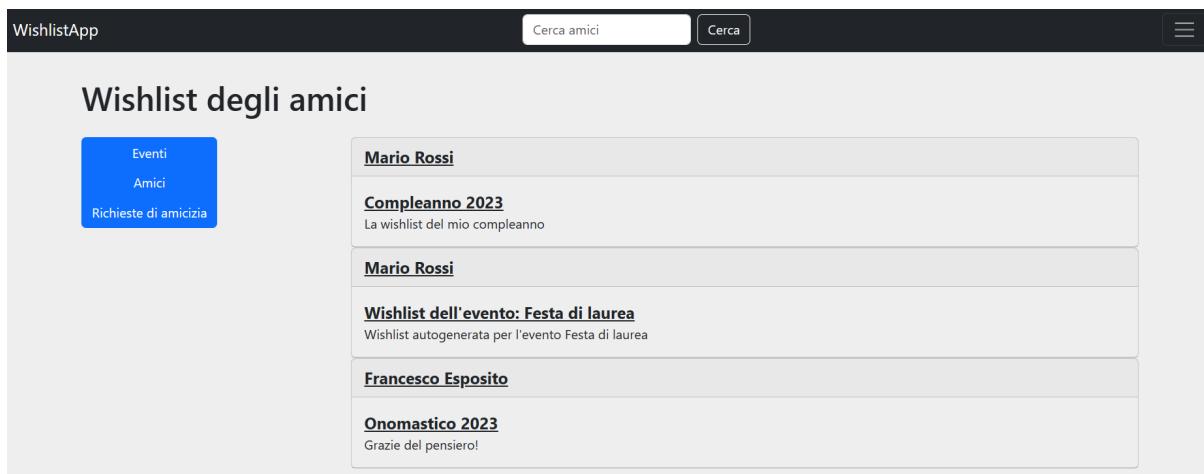


Figure 8.3: index.html

## 8.4 Le mie Wishlist (myWishlists.html)

In questa pagina ogni utente può visualizzare tutte le proprie wishlist. Sulla destra è presente un form per la creazione di una wishlist, costituito da

due campi in cui inserire il nome della nuova wishlist e una descrizione. È possibile inoltre cliccare su una delle wishlist per essere rimandati alla pagina relativa alla stessa.

The screenshot shows the 'WishlistApp' interface. At the top, there is a navigation bar with the app name, a search bar labeled 'Cerca amici', a 'Cerca' button, and a menu icon. Below the navigation bar, the main content area is titled 'Le mie wishlist'. On the left, there is a sidebar with three buttons: 'Eventi', 'Amici', and 'Richieste di amicizia'. The main content area displays two wishlist entries:

- Compleanno 2023**: La wishlist del mio compleanno
  - Regalo 1: Chitarra elettrica
  - Descrizione: Fender Telecaster
  - Link: <http://strumentimusicani.it>
- Wishlist dell'evento: Festa di laurea**: Wishlist autogenerata per l'evento Festa di laurea
  - Regalo 1: Penna
  - Descrizione: Parker nera
  - Link: <http://amazon.com>
  - Regalo 2: Polo
  - Descrizione: Lacoste
  - Link: <http://zanando.it>

To the right of the wishlist entries, there is a form for adding a new wishlist:

**Aggiungi nuova wishlist**

Nome wishlist  
Descrizione wishlist

**Aggiungi**

Figure 8.4: myWishlists.html

## 8.5 Wishlist (wishlist.html?id=)

Ad ogni Wishlist è associata una specifica pagina in cui è possibile gestire la wishlist. In particolare, è possibile aggiungere dei regali tramite un form e rimuoverli, così come è inoltre possibile eliminare la stessa wishlist. In questa schermata è anche possibile visualizzare se un regalo sia stato acquistato da un altro utente o meno.

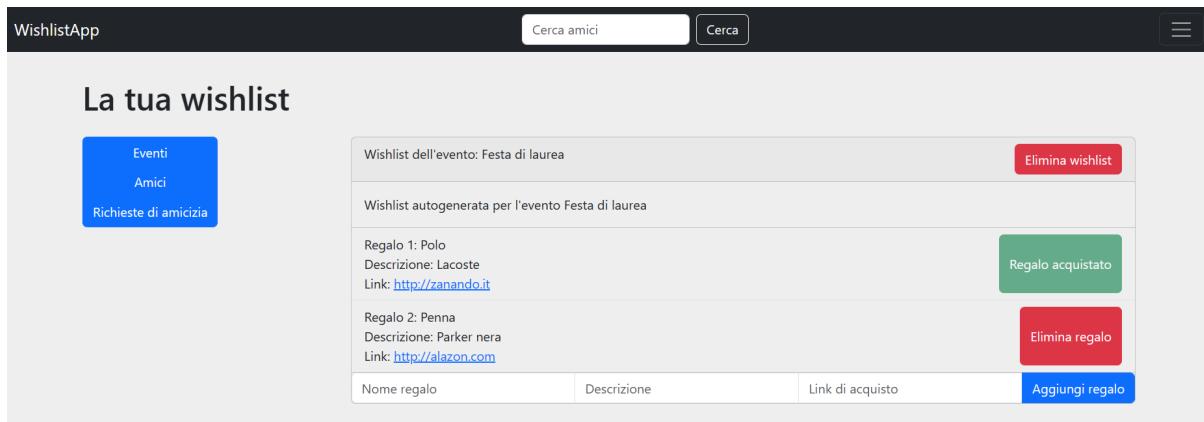


Figure 8.5: wishlist.html

## 8.6 I miei eventi (myEvents.html)

In questa pagina è possibile visualizzare tutti gli eventi a cui l'utente partecipa. Sempre sulla destra, è presente un form per la creazione di un nuovo utente, costituito dai campi:

- Nome evento
- Descrizione
- Data Evento
- Indirizzo Evento
- Associa a wishlist esistente
- Seleziona wishlist

In particolare, l'utente può decidere all'atto della creazione dell'evento, cliccando sulla specifica spunta, se associare ad esso una wishlist già esistente,

selezionandola eventualmente dal menù a tendina che restituisce tutte le wishlist associate all'utente, oppure crearne una nuova.

È possibile inoltre cliccare su uno degli eventi per essere rimandati alla pagina relativa allo stesso.

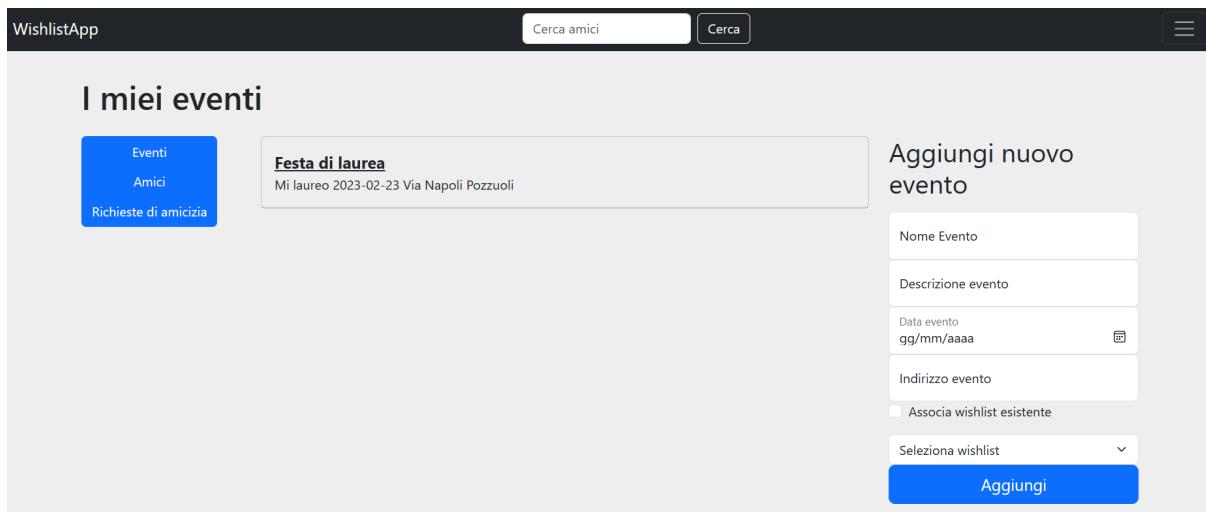


Figure 8.6: myEvents.html

## 8.7 Il tuo Evento (event.html?id=)

In questa pagina è possibile visualizzare uno specifico evento, con tutte le relative informazioni.

Da questa pagina è possibile invitare gli amici al proprio evento: cliccando sul bottone "Aggiungi invitati", vengono mostrati tutti gli amici non ancora invitati all'evento. Per invitarli, è sufficiente selezionare gli amici e cliccare su *Aggiungi invitati*.

Nella parte inferiore della pagina è presente un riferimento *Map* al luogo dell'evento.

## CHAPTER 8. MANUALE UTENTE

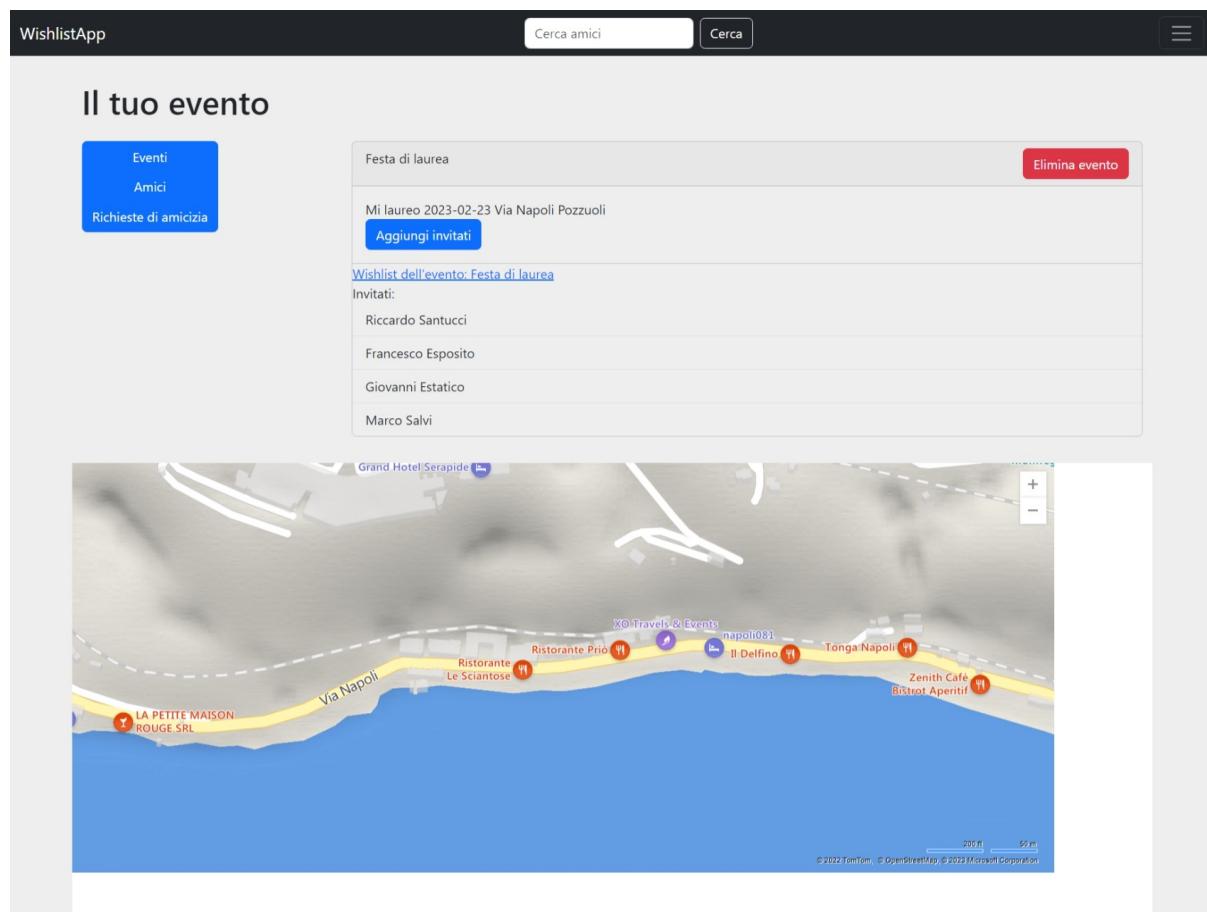


Figure 8.7: event.html

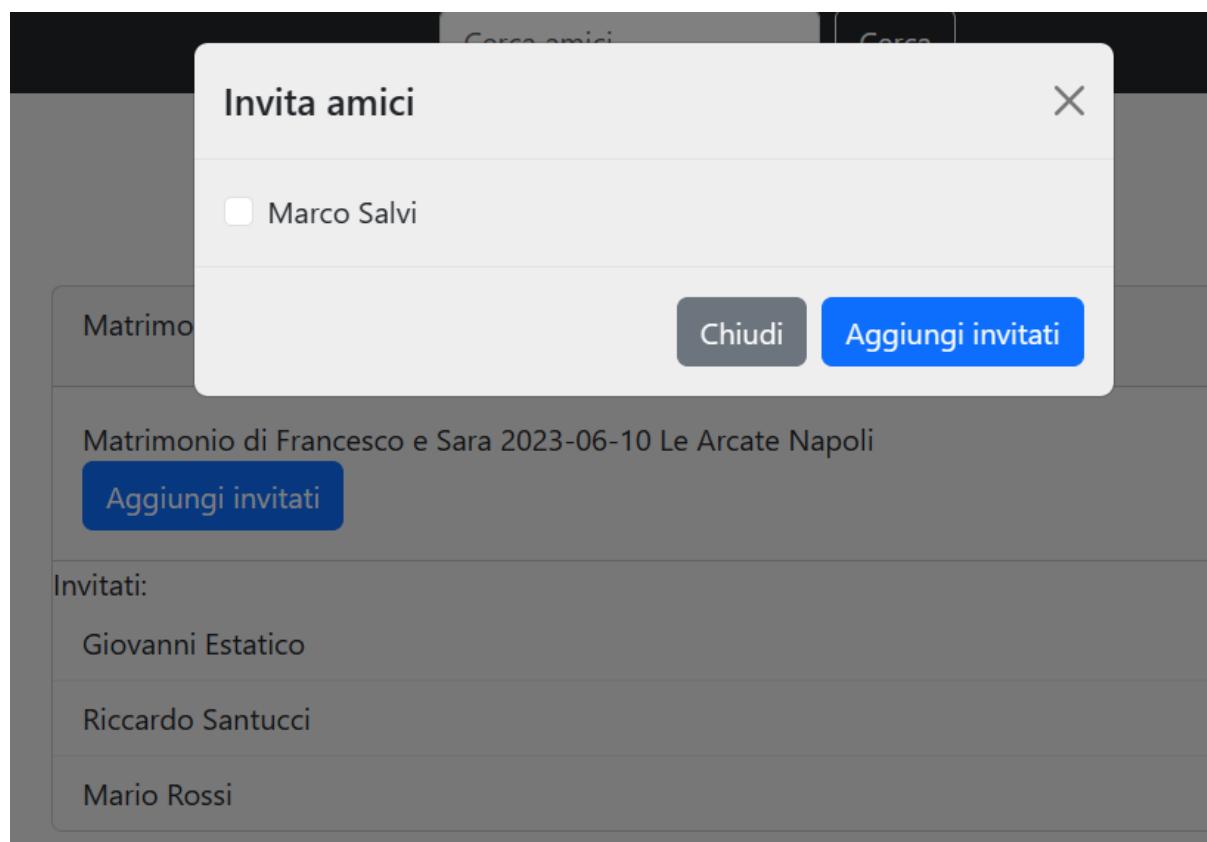


Figure 8.8: Menu di invito

## 8.8 Eventi degli amici (events.html)

In questa pagina è possibile visualizzare gli eventi dei propri amici. Per visualizzarli però non basta essere amici, ma bisogna anche essere invitati agli stessi.



Figure 8.9: events.html

## 8.9 Amici (friends.html)

In questa pagina è possibile visualizzare tutti i propri amici. Per rimuovere gli amici è sufficiente cliccare sul button "Rimuovi amico". Cliccando sugli amici si viene rimandati alle pagine dei relativi profili.

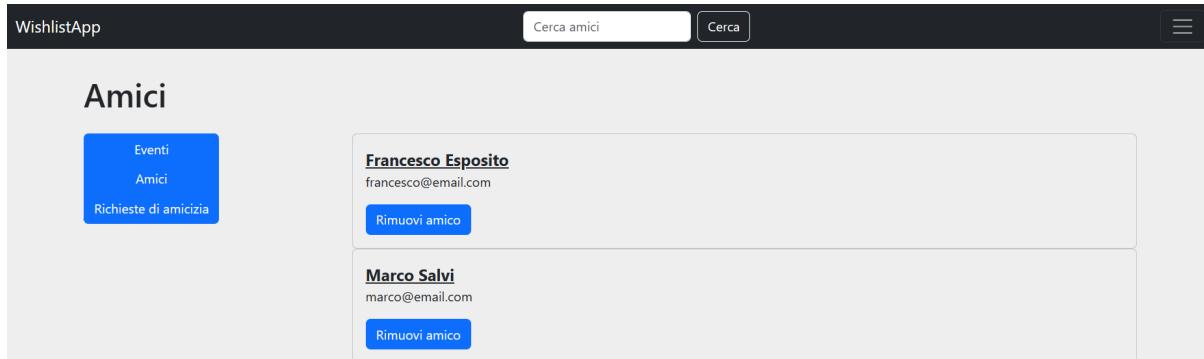


Figure 8.10: friends.html

## 8.10 Richieste di amicizia (friendsrequests.html)

In questa pagina l'utente può visualizzare le richieste di amicizia ricevute. Per ognuna, può decidere se accettare o rifiutare.



Figure 8.11: friendsrequests.html

## 8.11 Eventi degli amici (events.html)

In questa pagina l’utente può visualizzare tutti gli eventi a cui è stato invitato, con l’indicazione del nome, descrizione, data e luogo dell’evento. Cliccando sull’evento si viene rimandati alla pagina relativa allo stesso.



Figure 8.12: events.html

## 8.12 Wishlist di un amico (friendWishlist.html)

In questa pagina l’utente può visualizzare la wishlist di un amico, con tutti i relativi regali. Nel caso l’utente decida di acquistare uno dei presenti, può segnalarlo cliccando sul bottone associato "Segna come acquistato".



Figure 8.13: friendWishlist.html

## 8.13 Evento di un amico (friendEvent.html)

In questa pagina viene mostrato l'evento selezionato insieme alle relative informazioni, quali:

- Nome evento
- Descrizione
- Data dell'evento
- Luogo dell'evento
- Lista degli invitati all'evento

Nella parte inferiore della pagina è presente un riferimento *Map* al luogo dell'evento.

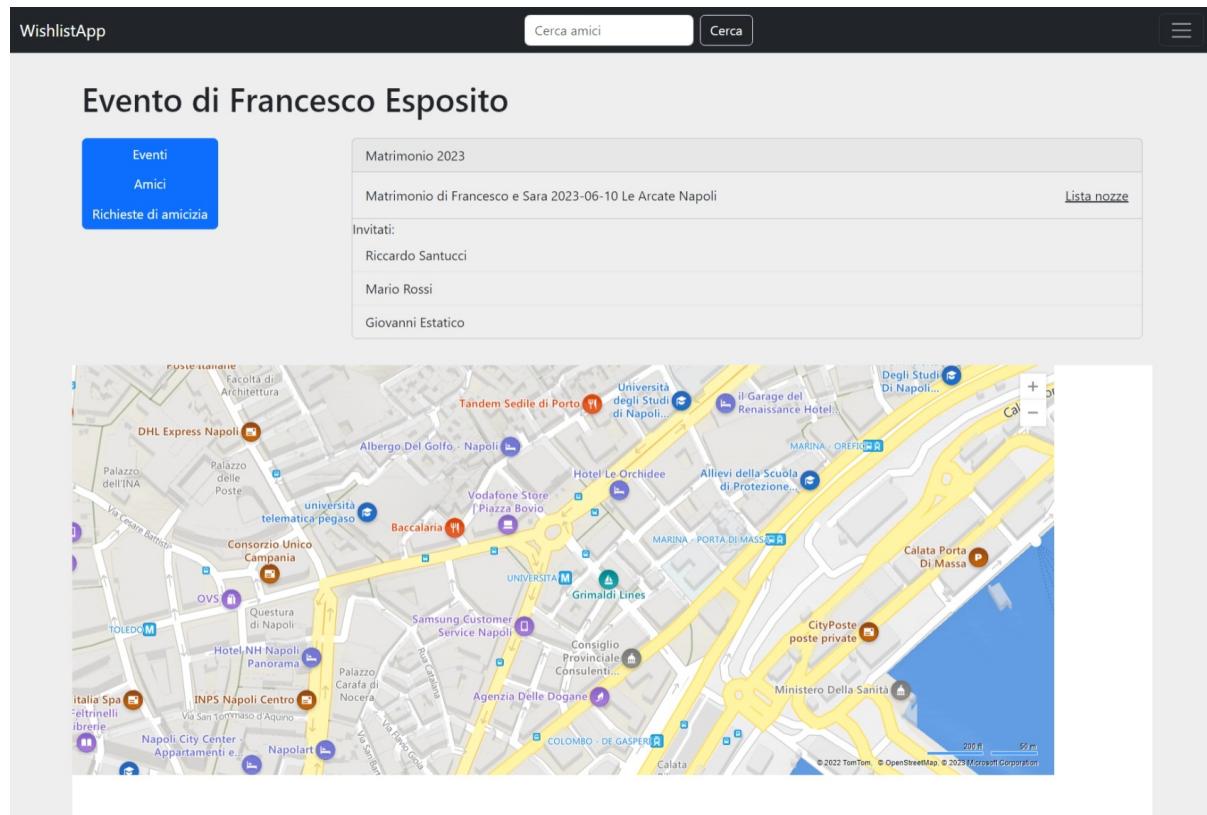


Figure 8.14: friendEvent.html

## 8.14 Il mio profilo (myProfile.html)

Questa è la pagina relativa al profilo dell’utente. Contiene informazioni relative all’utente e a tutte le wishlist ed eventi ad esso associato. Da questa pagina l’utente può sia navigare nell’applicazione tramite il menù sulla sinistra, sia accedere alle pagine delle specifiche wishlist o eventi, cliccando sui relativi riferimenti.

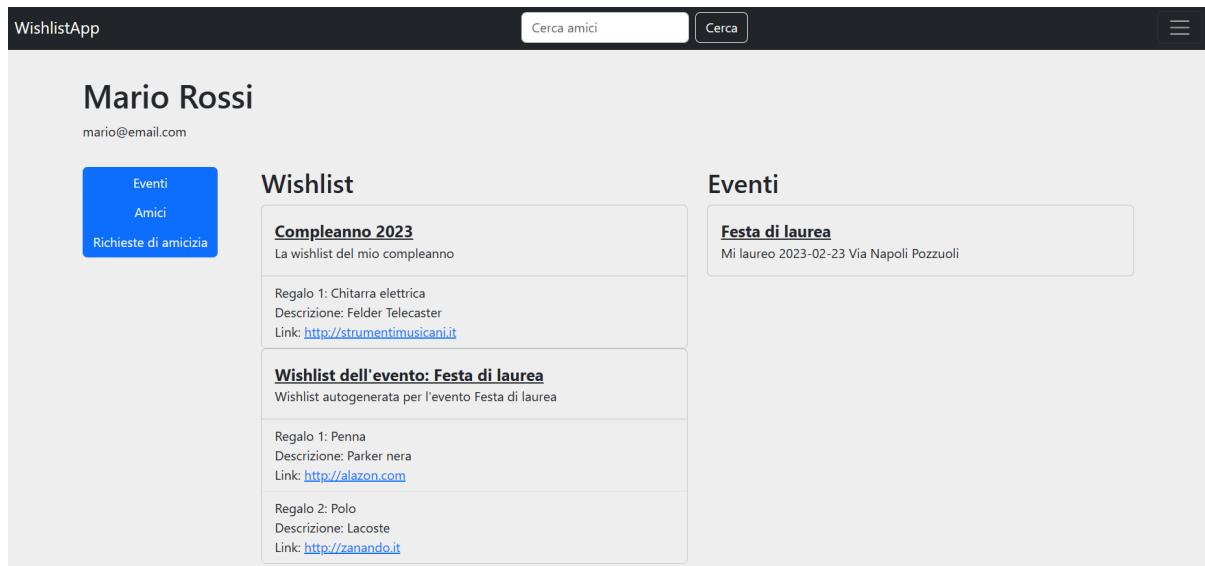


Figure 8.15: myProfile.html

## 8.15 Profilo amico (profile.html?id=)

Questa è la pagina relativa al profilo utente di un amico. Contiene informazioni relative all'utente e a tutte le wishlist ed eventi ad esso associato. Da questa pagina l'utente può sia navigare nell'applicazione tramite il menù sulla sinistra, sia accedere alle pagine delle specifiche wishlist o eventi, cliccando sui relativi riferimenti.

## CHAPTER 8. MANUALE UTENTE

The screenshot shows the user interface of the WishlistApp. At the top, there is a dark header bar with the text "WishlistApp" on the left, a search bar with the placeholder "Cerca amici" and a "Cerca" button in the middle, and a menu icon on the right.

The main content area displays the user's profile information:

**Francesco Esposito**  
francesco@email.com

On the left, there is a sidebar with three blue buttons: "Eventi", "Amici", and "Richieste di amicizia".

The main content is divided into two sections:

**Wishlist**

- Lista nozze**  
Mi sposo con Sara!
- Regalo 1: Servizio piatti  
Descrizione: Per 12  
Link: <http://piatti.com>
- Regalo 2: Friggitrice  
Descrizione: Marca qualsiasi  
Link: <http://cucinashop.it>

**Eventi**

- Matrimonio 2023**  
Matrimonio di Francesco e Sara 2023-06-10 Le Arcate Napoli

Figure 8.16: profile.html