Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica

Sviluppo di un'applicazione mobile per la gestione della partecipazione ad eventi con Flutter

Tesi di laurea triennale

Relatore

Prof.Francesco Ranzato

Laureando

Danilo Stojkovic

Anno Accademico 2021-2022

Sommario

Il seguente documento descrive il lavoro svolto al termine del percorso di studi della laurea triennale in Informatica presso l'azienda Moku s.r.l. con sede a Treviso (TV). Il periodo di stage ha previsto lo sviluppo di un'app mobile per i partecipanti ad eventi di vario tipo (fisici, virtuali o ibridi) che fornisca le funzionalità di visualizzazione, ricerca e iscrizione agli stessi, inoltre è incluso un sistema di check-in/check-out per mezzo di codice QR.

Ringraziamenti

Ringrazio il mio relatore, la mia famiglia, i professori, i compagni e gli amici...

Padova, Settembre 2022 Danilo Stojkovic

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Questo capitolo contiene una breve introduzione all'azienda ed allo scopo del periodo di stage.

1.1 Azienda

Moku s.r.l è una *software house* di Treviso composta da una ventina di dipendenti tra sviluppatori *frontend, backend, mobile* e *designer*.
L'anno di fondazione risale al 2013 quando nasce come startup nell'incubatore *H-farm*, essa è quindi una realtà giovane che si è dedicata sin dal principio alla scoperta e all'inclusione delle nuove tecnologie.
L'approccio di moku allo sviluppo software è *agile* e orientato al cliente, due valori ai quali i dipendenti tengono particolarmente.

Figura 1.1: logo dell'azienda Moku

La collaborazione tra Moku e l'Università di Padova si è consolidata negli anni attraverso l'evento Stage-it che permette all'azienda di accogliere ogni anno un gruppo considerevole di stagisti.
I progetti Moku si concentrano sulla trasformazione digitale, il software su misura ed il mondo delle startup.

L'ambiente di lavoro dell'azienda è confortevole ed il suo personale accogliente, gli strumenti offerti si sono rivelati utili allo svolgimento dello stage che è stato appropriatamente seguito dal tutor aziendale, il quale ha fornito un certo grado di autonomia pur assicurandosi che non ci fossero rallentamenti per mancata assistenza.

1.2 Scopo

Il progetto è *Evvvents*, un *software as a service* multipiattaforma che gestisce la creazione e la fruizione di eventi di vario tipo (fiere, mostre, corsi di formazione).
L'azienda ne gestisce il *backend* (sviluppato con *Ruby on Rails*), il *frontend* e l'applicazione *smartphone* sulla quale si concentra lo stage.
Un prototipo realizzato dal cliente includeva un'implementazione *mobile* che è stata tuttavia scartata in favore di una totale rivisitazione sia dell'aspetto grafico che del *backend* correlato.

I requisiti dell'applicazione sono correlati alle azioni disponibili agli utenti di tipo "partecipante" i quali:

- creano un profilo personale al quale accedere;
- possono visualizzare ed iscriversi ad eventi;
- ricevono notifiche personalizzate;
- possono effettuare il *check-in/check-out* dall'evento.

1.3 Preventivo iniziale

Lo scopo principale di questo bimestre consiste nell'inserire lo studente in un ambiente lavorativo reale dal quale è possibile apprendere tutti gli aspetti dell'impiego non legati allo studio accademico, si tratta di un passaggio fondamentale nella carriera lavorativa data la sua natura ed il suo posizionamento al confine tra la preparazione teorica e l'occupazione.

La durata prevista per il periodo di stage è di approssimativamente 300-320 ore, come esse siano state impiegate e per il raggiungimento di quali obiettivi è stato chiarito nel Piano di lavoro redatto dall'azienda ospitante.
Il preventivo delle ore è esposto nella seguente tabella.

Attività	**Settimana**	**Ore**
Comprensione sistema e obiettivi	1	20
Analisi dei requisiti	1	20
Progettazione	2-3	60
Studio e setup ambiente di sviluppo	3	20
Implementazione	4-7	150
Test e validazione	7-8	30
Documentazione	8	20
Totale		**320**

Capitolo 2

Il progetto

Nel seguente capitolo verranno illustrate in dettaglio le specifiche del progetto di stage; dalla presentazione delle sue tecnologie alla descrizione della sua natura.

2.1 Presentazione del prodotto

Evvvents è un'applicazione mobile il cui obiettivo consiste nell'interfacciarsi con l'utente base (partecipante ad un evento) al fine di fornirgli un'intuitiva presentazione degli eventi organizzati al suo interno e di permettergli l'iscrizione agli stessi, nonché le funzionalità di controllo degli accessi e delle uscite.

Evvvents nasce come piattaforma per la gestione di eventi di svariata natura, siano essi mostre, concerti, riunioni aziendali private o convegni; una gestione unificata di questo tipo di avvenimenti è l'ideale per risparmiare le risorse degli organizzatori e semplificare la partecipazione dei visitatori, si parte da ciò che li accomuna per poi facilitare la navigazione utilizzando ciò che li differenzia; ogni evento ha la sua location (che sia essa fisica o virtuale), il suo organizzatore, i suoi argomenti ed il suo pubblico.

Essa è stata presentata all'azienda ospitante sotto forma di prototipo da arricchire ed evolvere, tuttavia la componente *mobile* di tale prototipo è stata scarsamente sviluppata, l'applicazione prodotta durante lo stage l'ha perciò sostituita.

Figura 2.1: logo della piattaforma Evvvents

2.2 Obiettivi del progetto

Nel piano di lavoro è presente un primo elenco dei requisiti, suddivisi tra obbligatori e desiderabili; questi fanno riferimento sia ai dettagli del prodotto finale che ai metodi dello svolgimento dello stage.

* Requisiti obbligatori:
  - organizzazione del lavoro tramite tecnologie *agile* di tipo SCRUM;
  - comunicazione con il backend tramite REST;
  - sistema di *login/signup* alla piattaforma;
  - gestione delle iscrizioni agli eventi;
  - gestione del processo di *check-in/check-out*.
* Requisiti desiderabili:
  - coordinamento con il *product owner*;
  - integrazione di un sistema di messaggistica;
  - *suite* di *testing* del prodotto.

Lo stagista è stato inserito in un team dedicato alla piattaforma che include:

* un *project manager*;
* uno sviluppatore *backend*;
* un *graphic designer* e *UI/UX expert*;
* uno sviluppatore *frontend* (lato *web*).

2.3 Metodo di lavoro: Scrum

Nell'azienda ospitante il *framework* SCRUM è lo standard dal quale partire per l'impostazione di qualsiasi progetto, di conseguenza anche Evvvents ha rispettato questo sistema di pianificazione.

Il metodo di organizzazione dello sviluppo in maniera iterativa si basa sul concetto di *daily scrum* dal quale prende il nome e che rappresenta il fulcro della comunicazione tra i membri del team racchiuso in un semplice colloquio informale di circa 15 minuti.

L'idea di fondo consiste nel suddividere i periodi di lavoro in *sprint* di durata fissata (es. due settimane) caratterizzati da un insieme di obiettivi (*task*) da realizzare (*sprint backlog*).

Ai limiti di ogni *sprint* si tiene una serie di riunioni particolari:

* *sprint planning*: dove si pianifica il lavoro da svolgere sotto forma di insieme di *task* associati al tempo a loro dedicato, include la partecipazione del *product owner*;
* *sprint review*: revisione dell'incremento effettuato e del *product backlog* (elenco totale delle funzionalità del prodotto), viene eseguito uno studio di quanto ciò che è stato pianificato è stato effettivamente realizzato;
* *sprint retrospective*: riunione conclusiva dello *sprint* durante la quale viene ispezionato il *way of working* del *team* nel periodo trascorso e le possibili migliorie da attuare, ci si concentra su persone, relazioni e strumenti.

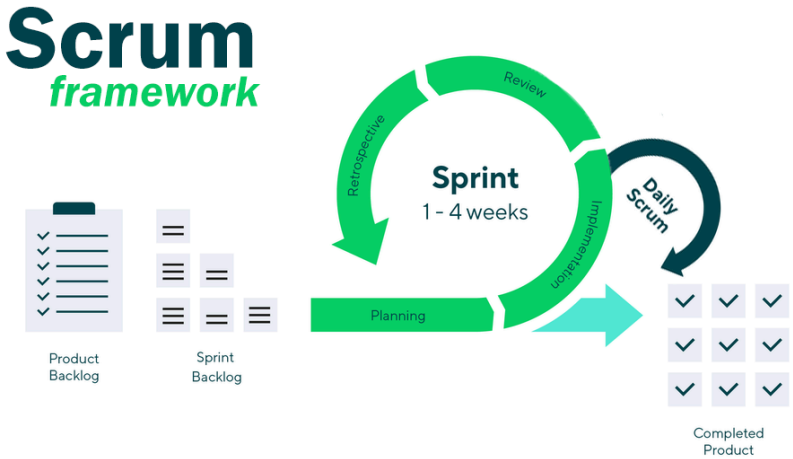

Figura 2.2: grafico delle fasi scrum

Questo metodo ha acquisito negli ultimi anni una straordinaria popolarità nel mondo dell'informatica grazie ai vantaggi offerti, come maggiore collaborazione con l'utente finale, il suo contributo al miglioramento continuo e la superiore gestione dei rischi; pregi rilevati anche durante il progetto di stage.

2.4 Tecnologie di supporto

Gli strumenti elencati in questa sezione sono tecnologie che affiancano lo sviluppo adottate dall'azienda ospitante indipendentemente dal progetto.

BitBucket

Questo servizio di *hosting* di proprietà di *Atlassian* affianca *git* nel versionamento del progetto e ospita la *repository* remota.

L'utilizzo di questo prodotto porta a diversi vantaggi:

* una migliore organizzazione del lavoro attraverso i *feature branch*, un'esempio potrebbe essere il ramo *feature/pagina-preferiti*;
* un sistema di versionamento con una versione stabile sempre disponibile e con possibilità di ripristino in caso si presentino problematiche;
* possibilità di lavoro contemporaneo minimizzando i conflitti operando su *branch* separati e concludendo il tutto con un unico *merge* finale;
* comodità nelle *code review*, componente fondamentale di una *pull request*.

Figura 2.3: loghi di BitBucket e Git

Il *workflow* applicato all'utilizzo di questi strumenti è **gitflow**, questo flusso di lavoro prevede:

* un branch **feature/nomeDellaFeature** dedicato ad una data porzione del lavoro;
* un branch **develop** dal quale partono le *feature* e che ne raccoglie i *merge*;
* un branch **master** nel quale è presente l'ultima *release* stabile.

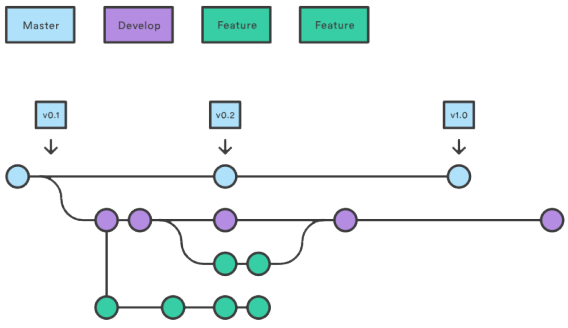

Figura 2.4: grafico di esempio dell'applicazione di gitflow

SourceTree

In associazione con *BitBucket* è necessario citare un altro prodotto di *Atlassian*: *SourceTree*, quest'ultimo è un *client desktop* che permette l'accesso alla *repository* locale agli sviluppatori.

La sua interfaccia intuitiva e le operazioni semplificate per mezzo di bottoni permettono il lavoro autonomo anche degli utenti scoraggiati dalla *git bash*.

Tra i vantaggi di questa tecnologia si citano:

* la rappresentazione grafica dei rami, estremamente utile per intuirne lo stato;
* la schermata di elenco dei *commit* dalla quale è possibile visualizzare le modifiche effettuate;
* la generazione ed il mantenimento di una chiave ssh dedicata alla sicurezza.

Figura 2.5: logo di Sourcetree

Slack

Un altro *software* decisivo per la collaborazione tra i dipendenti Moku, sia in presenza che in *smart working*, è *Slack*: una piattaforma di messagistica istantanea particolarmente adatta all'utilizzo nelle aziende di informatica.

I motivi della sua popolarità nel settore sono molteplici:

* la possibilità di suddividere l'ambiente principale in canali specifici dedicati (ad esempio) al singolo progetto;
* l'autenticazione a due fattori che ne garantisce la sicurezza;
* la possibilità di impostare notifiche da presentare a tutti i membri;
* la facilità di integrazione di tecnologie esterne come *Google Drive*;
* la disponibilità di messaggi speciali contenenti "codice" la cui formattazione è mantenuta intatta.

Figura 2.6: logo di Slack

2.5 Tecnologie di sviluppo

Lo sviluppo del software è accompagnato da un numero sempre maggiore di applicativi di appoggio e varie tecnologie che ne facilitano la realizzazione e la collaborazione all'interno di un team di lavoro. Segue quindi una breve descrizione delle componenti dell'ambiente di produzione.

Flutter

Innanzitutto occorre specificare il *framework* di costruzione dell'applicativo, si tratta di *Flutter*, progetto *open-source* di *Google* il cui vantaggio principale è la generazione di applicazioni multipiattaforma a partire da un unico codice sorgente, permette quindi allo sviluppatore di concentrarsi sul prodotto da realizzare senza dover preferire un sistema operativo *mobile* ad un altro.

Si tratta di un prodotto estremamente recente (la prima versione stabile risale al 2018) che sta rapidamente ottenendo il consenso di molti esperti nel campo per la sua comodità, intuitività e semplicità di utilizzo. Sintomo di questa freschezza è anche il fatto che sono ancora pochi i programmatori che lo conoscono a fondo, nonostante sia una *skill* sempre più richiesta.

Un ultimo pregio di questa tecnologia è la straordinaria quantità di librerie messe a disposizione che semplificano enormemente *task* altrimenti tediosi in altri linguaggi.

Figura 2.7: logo del framework Flutter

Dart

Il linguaggio sul quale si basa Flutter è *Dart*, il quale nacque con l'intento di sostituire *JavaScript* come protagonista delle applicazioni web. Tra i suoi pregi si elencano il compilatore JIT, migliore gestione della sicurezza, la velocità e la maggiore scalabilità. Attualmente il suo utilizzo principale è per l'appunto legato al mondo del *mobile*, grazie al suo compilatore *dart2native*.

Il paradigma principale è l'orientamento agli oggetti, una sua particolarità è data dalla sua attenzione alla *null safety* per la quale nessun valore può essere nullo a meno che questa possibilità non sia esplicitamente dichiarata.
Inoltre la gestione della concorrenza e degli oggetti di tipo *Future*, fondamentali in un'applicazione *mobile*, lo rendono ancora più adatto per il compito assegnatogli.

Figura 2.8: logo del linguaggio Dart

Il tassello principale dei mosaici costruiti con *Flutter* e *Dart* sono i *Widget*, il generico componente grafico che può a sua volta ricorsivamente ospitare ulteriori *Widget*; è proprio questa natura compositiva delle sue parti più semplici a rendere il linguaggio così espressivo e la sua sintassi così particolare.

Insomnia

Come già accennato l'applicazione comunica con un *backend* attraverso un'architettura API/REST, le cui chiamate sono state memorizzate e preventivamente testate attraverso il tool *Insomnia*.
Questo *software* facilita l'organizzazione delle chiamate, ne permette la verifica ed ha inoltre un'interessante meccanismo di esportazione/importazione del corpo delle interrogazioni, che ne permette la condivisione.

Figura 2.9: logo di Insomnia

Capitolo 3

Analisi dei requisiti

Per eseguire uno studio profondo del prodotto che si andrà a costruire occorre partire dai bisogni che esso soddisfa; dedurre le funzionalità da offrire dall'idea del product owner *è un primo passaggio fondamentale dello sviluppo software.*

In informatica ci si affida ad alcuni schemi e formalismi tecnici che guidino lo studio dei requisiti, questi sono dettati dal linguaggio UML. Ciononostante si è preferito inserire nel capitolo un'analisi più di alto livello e rilegare all'appendice A il lavoro più rigoroso.

3.1 Utenti

Studio target

Il compito iniziale di chi ha dovuto definire i requisiti dell'applicazione è stato quello di individuare gli utenti finali che andranno ad utilizzarla.

"A chi serve questo prodotto?" è una domanda fondamentale da fare prima di dedicare risorse alla sua produzione. Evvvents punta certamente ad un pubblico piuttosto variegato in quanto l'obiettivo è stato sin dagli inizi ospitare qualsiasi tipo di evento ricordi almeno vagamente il formato presentato dall'app.
Questo porta all'avere diversi gruppi non omogenei da soddisfare, in generale si è comunque sempre contato su una popolazione di persone giovani che hanno familiarità con il mondo digitale e apprezzano la semplificazione che questo offre.

Il risultato è un software che cerca di non contenere alcuna complicazione inutile e che basa diversi suoi aspetti su concetti già molto popolari in prodotti simili e non.

Classificazione utenti

Per passare dai bisogni, ai requisiti ed infine alla funzionalità è necessario definire una distinzione tra gli stati nei quali si può trovare un utente di Evvvents.

La piattaforma si basa infatti su un sistema di utenti riconosciuti ai quali sono dedicate le funzionalità, questo sottindende di base almeno tre stati dell'utente:

Stato utente	**Azioni concesse**
Non registrato	Registrazione
Registrato	Accesso, Recupero password
Accesso eseguito	Utilizzo standard di Evvvents

Le azioni concesse nella tabella precedente ispirano le prime funzionalità da aggiungere prima ancora di decidere cosa offrirà l'applicazione a chi ha effettuato l'accesso al proprio profilo.

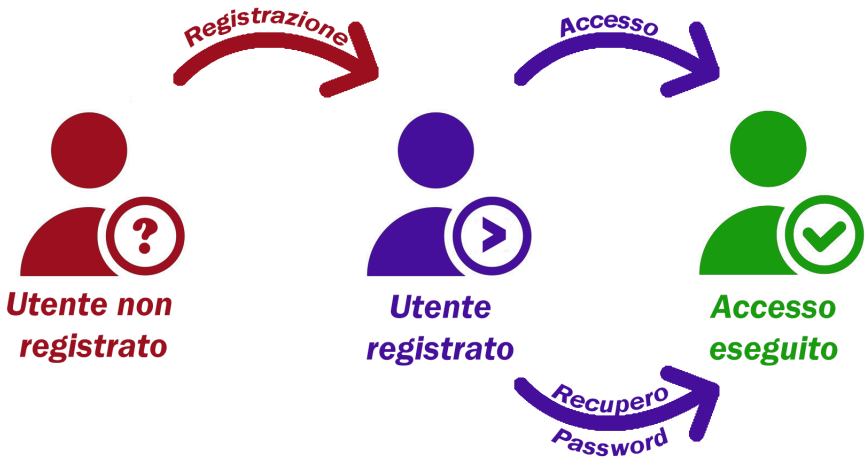

Figura 3.1: Piccolo schema rappresentativo degli stati utente

3.2 Funzionalità

Studiare a quali fini un utente (che ha eseguito l'accesso) utilizzerà l'app porta alla scoperta delle sue funzionalità da progettare e codificare.

Si parte innanzitutto dal cuore di Evvvents, gli eventi; è di sicuro necessario permettere la visualizzazione dei dettagli, l'iscrizione ed il check-in a questi, nonché un modo di esibirne diversi tra cui scegliere all'utente (e filtrarli).
Tutti questi bisogni sono confluiti nella pagina iniziale, detta anche *home*, dell'applicazione in quanto logicamente i più importanti.

Notando poi la necessità dell'utente futuro di avere uno storico o comunque una sorta di elenco degli eventi ai quali ha scelto di partecipare è stata prevista la possibilità di fare proprio questo in un'ipotetica sezione "I miei eventi".

Conoscendo poi la classificazione degli stati di un evento nel *backend*, la quale prevede gli eventi "in anteprima" (a cui non è possibile iscriversi), si è deciso si includere uno stato intermedio tra iscritto e non: "salvato tra i preferiti".
Questi eventi dotati di *"bookmark"* saranno poi consultabili nella loro apposita lista.

Un altro obiettivo del team dietro ad Evvvents è sempre stato mantenere un rapporto stabile di comunicazione con l'utente finale, questo è in parte permesso da un sistema di notifiche visualizzabili direttamente nell'applicazione.

Successivamente, avendo l'utente creato un profilo va previsto l'aggiornamento dei suoi dati, questo lato è stato poi arricchito con statistiche e la possibilità di vedere gli eventi passati.

Infine, similmente alla pagina del profilo, c'è stato bisogno di permettere di personalizzare le impostazioni (ad esempio delle notifiche) e di rivedere documenti importanti legati all'utilizzo di Evvvents.

Tabella 3.1: Tabella delle funzionalità dell'applicazione

Utente	**Macroclasse**	**Funzionalità**
Utente non registrato	Registrazione	Registrazione del profilo
Utente registrato	Accesso	Accesso al profilo
	Recupero password	Reimpostazione password utente
Utente con accesso	Home	Visualizzazione eventi disponibili
		Applicazione di filtri agli eventi
		Visualizzazione dettaglio evento
		Iscrizione all'evento
		Check-in all'evento
	I miei eventi	Visualizzazione eventi ai quali è iscritto
	Eventi preferiti	Visualizzazione eventi salvati
	Notifiche	Visualizzazione delle notifiche *push*
	Profilo	Aggiornamento dei dati del profilo
		Visualizzazione statistiche partecipazione
		Visualizzazione storico eventi passati
	Impostazioni	Impostazione preferenze sulle notifiche
		Visualizzazione privacy policy
		Visualizzazione termini e condizioni

Capitolo 4

Progettazione e codifica

In questo capitolo viene trattata la progettazione che ha poi guidato la realizzazione fino ad ottenere un'applicazione utilizzabile dall'utente.

4.1 Architettura

L'architettura di Evvvents si basa su un *backend* sviluppato in *Ruby on Rails* al quale l'applicazione (ed una webApp in *Angular*) si interfacciano per mezzo di chiamate API/REST.
Queste scelte rappresentano uno *standard* largamente utilizzato nel settore per prodotti di questo tipo, rispecchiano inoltre le decisioni prese dal cliente nel momento della creazione del prototipo ed infine si basano su tecnologie con cui l'azienda ospitante ha molta familiarità.

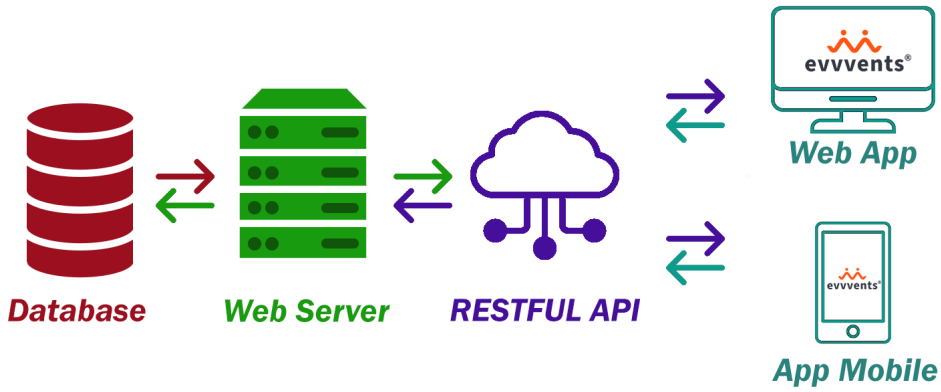

Figura 4.1: rappresentazione grafica dell'architettura scelta

Backend

Il *backend* non è stato argomento di stage, tuttavia è una componente fondamentale del *software*; è stato sviluppato in contemporanea con l'applicazione, motivo per il quale talvolta l'avanzamento è stato rallentato o ha necessitato di *mock*.
Come già accennato è stato realizzato in *Ruby on Rails* da un membro del team di Evvvents.

Figura 4.2: logo di *Rails*

Frontend: App Mobile

L'ultimo componente dell'architettura è l'applicazione mobile, la quale è stata interamente prodotta dallo stagista in *Flutter*.
La sua architettura interna è basata su 4 *layer* che permettono una discreta separazione tra interfaccia utente e metodi di accesso ai dati remoti. Gli strati sono dunque:

* *User Interface*: questo *layer* è composto dalle classi (.*)Screen, caratterizzate dai vari *widget* di *layout*, con le quali l'utente interagisce direttamente.
* *State Management*: gli eventi scatenati dall'utente e lo stato attuale dell'applicazione sono direttamente collegati alle varie classi del *pattern* BLoC, il quale verrà illustrato più a fondo nella sezione successiva.
* *Repository*: un semplice strato di inoltro richieste che "nasconde" il modo attraverso il quale i dati sono ottenuti e permette di cambiare quest'ultimo senza troppe conseguenze sui *layer* superiori.
* *HTTP*: insieme di classi il cui scopo principale è eseguire le chiamate *API* al *backend* e successivamente elaborare la risposte ricevute.

Figura 4.3: funzionamento dei *layer* dell'applicazione

4.2 Progettazione

Flow di utilizzo

Durante i primi *stand up* e *meeting* del team è stato necessario definire quali funzionalità offrire all'utente ed il flusso del susseguirsi di queste.
Aver documentato questi fondamentali dettagli come primo passo ha permesso alle prossime attività di essere svolte in contemporanea e accelerare lo sviluppo del prodotto.

Sign in/Sign up

Si è innanzitutto partiti dalle due schermate base di una qualsiasi applicazione con utenti riconosciuti, è stato inoltre previsto da subito il meccanismo di recupero password attraverso l'email di registrazione.

Home - Eventi principali

La prima sezione da presentare ad un utente che ha eseguito l'accesso consiste in una serie di eventi disponibili classificati e filtrabili con diversi criteri.

Preferiti & I miei eventi

Dalla barra di navigazione prevista dall'applicazione è possibile raggiungere due sezioni di natura molto simile: i "preferiti" e i "miei eventi", le quali raccolgono rispettivamente gli eventi che l'utente ha salvato per rivisitare in futuro e gli eventi ai quali l'utente ha registrato la partecipazione.

Dettaglio evento

Cliccando su un evento in un qualsiasi punto si è riportati alla sua pagina di dettaglio, nella quale sono presenti tutti i dati utili e la possibilità di iscriversi (in caso questo non sia già stato fatto) o effettuare il *check-in* producendo il QR code apposito.

Notifiche

Data la natura dell'applicazione si è da subito posta molta attenzione alla gestione delle notifiche utente, le quali lo possono informare di cambiamenti dell'ultimo minuto, servire da *reminder* o semplicemente consigliare nuovi eventi.

Profilo

L'ultima voce presente nella navbar racchiude in sè tutte le operazioni che un utente può voler svolgere riguardo al proprio profilo, queste possono essere: modificare i propri dati, visualizzare le statistiche o semplicemente effettuare il *logout*.

Impostazioni

Infine non poteva mancare una schermata che assicurasse all'utente la possibilità di esprimere preferenze (ad esempio su come e quando essere notificato) o di cancellare il proprio *account*.

Ovviamente tutte queste considerazioni iniziali si sono evolute e sono state arricchite durante lo sviluppo vero e proprio.

4.3 Struttura dell'applicazione

Un ottimo modo per descrivere la struttura è partire dall'albero delle cartelle, esso infatti rispecchia in buona parte la suddivisione dei compiti fra le varie classi.
Nello sviluppo *Flutter* tutto il codice *Dart* è contenuto nella cartella \lib, la quale ha il seguente contenuto:

* *evvvents.dart*: classe principale nella quale viene definito il *routing*;
* *main.dart*: funzione che lancia l'esecuzione del prodotto.

Il resto di \lib è così organizzato:

\screens

Cartella che ospita tutte le classi di interfaccia utente e le classi BLoC ad esse collegate. Essendo Evvvents un'applicazione con utenti riconosciuti, le successive cartelle sono

* \homepage: nella quale sono presenti le varie "facciate" viste dall'utente come:
  - \main_events;
  - \your_events;
  - \favorites;
  - \profile;
  - \settings;
  - \notifications.
* \login: la quale contiene le "facciate":
  - \forgot_password;
  - \signup;
  - \successful_registration.

Nelle schermate più complesse è possibile vedere un ulteriore suddivisione delle cartelle, ma al fine di comprendere il *flow* dell'applicazione questo studio è sufficiente.

\entities

Dove sono racchiuse una serie di classi *custom* specifiche di Evvvents e che permettono di gestire in modo più semplice i suoi dati, generalmente rispecchiano la tabella corrispondente sul *database*.
Le classi in questione sono:

* *event.dart*: entità con la quale vengono memorizzati i vari eventi che siano essi disponibili, passati, preferiti ecc.;
* *location.dart*: ogni evento si tiene in una *location* specifica ed essa ha una serie di dati racchiusi in questa particolare classe;
* *venue.dart*: una singola *location* è suddivisibile in *venue* nelle quali è anche dichiarata la capacità;
* *notification.dart*: il sistema di notifiche di Evvvents si appoggia su una serie di oggetti di questa classe.

\custom_widgets

La natura compositiva dei *Widget* di *Flutter* permette di costruire elementi di *layout* anche molto complessi e personalizzati, nei casi in cui c'è stato bisogno di questi in più punti è stato creato un *"custom widget"* dedicato, questi sono:

* *event_card.dart*: predispone i dati legati ad un evento in una classica card poi posizionata in un carosello;
* *new_event_card.dart*: versione particolare dedicata agli eventi in anteprima;
* *your_event_card.dart*: card dell'evento presentata nella sezione "i tuoi eventi";
* *fav_event_card.dart*: card dell'evento presentata nella sezione "preferiti";
* *event_description.dart*: descrizione presente nella pagina di dettaglio evento che include un meccanismo di "apertura/chisura";
* *filters_panel.dart* e *filters_class.dart*: utilizzate nella creazione del pannello dei filtri nella *home*;
* *qr_panel.dart*: pannello dedicato alla presentazione del *QR code* per gli eventi ai quali l'utente è iscritto;
* *input_field_evv.dart*: versione personalizzata dell'originale *widget* che prende input testuali;
* *back_button_evv.dart*: un classico bottone per tornare indietro dalla pagina corrente.

\context

Al fine di condividere i dati tra più pagine è stata utilizzata (dove indispensabile) la scelta di una classe di tipo *singleton* che ospitasse il contesto attuale, esse sono:

* *user_context.dart*: dove viene memorizzata la sessione utente e tutti i suoi dati;
* *events_context.dart*: punto di raccolta degli eventi e della loro suddivisione tra disponibili, preferiti, in anteprima e a cui l'utente si è iscritto;
* *notifications_context.dart*: contenente le notifiche attualmente disponibili per l'utente.

\Utils

Questa sottocartella contiene variabili e metodi di tipo statico che possono essere utili alle altre classi, degli esempi sono:

* *validator.dart*: insieme di funzioni di validazione dell'input usate principalmente al momento di registrazione utente;
* *api_utils.dart*: dove è dichiarato il link dell'API per facilitare i test ed il passaggio da *staging* a *release*;
* *privacy_utils.dart*: dove è presente per comodità la lista di articoli sulla privacy da mostrare all'utente.

\Repository & \http

Come già accennato questi due gruppi di classi si occupano della comunicazione con il *backend*, contengono entrambi semplicemente una classe dedicata allo *user* ed una dedicata agli *event*.

```
Elenco del percorso delle cartelle per il volume OS
Numero di serie del volume: 767B-B0C7
lib:.
|
+---context
+---custom_widgets
+---entities
+---http
+---repository
+---screens
|   +---homepage
|   |   +---favorites
|   |   |   \---bloc
|   |   +---main_events
|   |   |   +---bloc
|   |   |   \---event_details
|   |   |       +---bloc
|   |   |       \---ticket_booking
|   |   |           +---bloc
|   |   |           \---confirm_booking
|   |   +---notifications
|   |   |   \---bloc
|   |   +---profile
|   |   |   +---bloc
|   |   |   +---my_details
|   |   |   +---passed_events
|   |   |   \---statistics
|   |   +---settings
|   |   |   +---bloc
|   |   |   +---delete_account
|   |   |   +---notification_management
|   |   |   +---privacy_policy
|   |   |   \---terms_conditions
|   |   \---your_events
|   |       \---bloc
|   \---login
|       +---bloc
|       +---forgot_password
|       |   \---bloc
|       +---signup
|       |   +---bloc
|       |   +---confirm_registration
|       |   \---set_password
|       |       \---bloc
|       \---successful_registration
\---utils
```

Figura 4.4: albero delle sottocartelle del progetto

Wireframe

Il *design* dell'aspetto grafico dell'applicazione è stato affidato ad un membro del team che ha prodotto una serie di *wireframe* utilizzando l'editor online ***Figma***.

Figura 4.5: logo di *Figma*

Il layout così prodotto è stato trasportato il più fedelmente possibile nel codice *Flutter*.

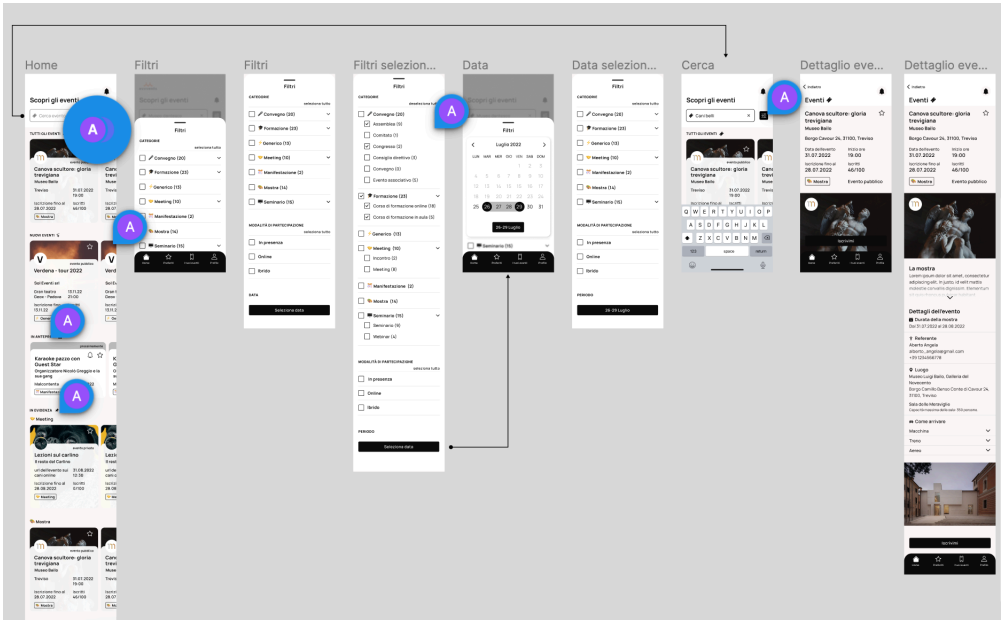

Figura 4.6: schermata dei *mock* up di Evvvents su *Figma*

Figma si è rivelato uno strumento molto interessante grazie alla facilità di condivisione dei *mock up* e alla possibilità di inserire commenti attraverso i quali chiarire i punti che portano a dubbi.

4.4 Design Pattern utilizzati

BLoC

Il *design pattern* del quale è stato fatto più largo uso durante la progettazione dall'app è il ***B**usiness **Lo**gic **C**omponent*, che ha semplificato di molto lo *state management* all'interno delle varie pagine.
Questo *pattern* basato sulla programmazione reattiva è stato ideato da *Google* appositamente per il linguaggio Dart; è quindi nato molto recentemente.
Il suo vantaggio principale consiste nel fatto che elimina la gestione (eccessivamente complessa in Dart) dello stato negli *StatefulWidget* e permette invece di aggiungere uno stato separato ad uno *StatelessWidget*.
Ogni applicazione del *pattern* si compone di tre classi, ognuna con uno scopo ben preciso:

* *State*: classe che effettivamente contiene le variabili di stato e che lo rendono disponibile alle "classi schermata";
* *Event*: classe padre dalla quale ereditano tutti gli eventi (scatenati dall'utente o meno) che possono cambiare lo stato;
* *Bloc*: classe che fa da *middle man* tra chi scatena gli eventi e lo stato da modificare di conseguenza, nella pratica offre un'associazione evento-funzione nella quale assicura una porzione di codice da eseguire nell'evenienza di qualsiasi evento da essa gestito.

Vediamo un esempio del suo utilizzo per poter chiarire meglio la sua efficacia: nel momento in cui l'utente si sta iscrivendo al servizio gli sono richiesti alcuni campi dati *standard*, prendiamo per esempio l'email.

1. Questa è inserita dall'utente all'interno di un *widget InputFieldEvv* il quale aggiungerà allo *stream* di eventi da gestire del *BlocProvider* della schermata un evento di tipo *InputChangedEvent* (contenente un identificatore di quale input è stato modificato e il suo nuovo valore).
2. Una volta ricevuto l'evento la classe BLoC avvia la propria funzione *_triggerOnInputChanged* che si occupa di costruire una nuova versione dello stato (dove adesso sarà salvato il nuovo valore dell'input).
3. Viene infatti chiamata la funzione *copyWith* dello stato che ne costruisce una copia nella quale vengono sostituiti solo i parametri passati (sono tutti opzionali e corrispondono ai parametri di stato).
4. Il nuovo stato viene quindi emesso, cambiamento individuato dal *BlocProvider* il quale aggiorna la pagina di conseguenza.

Decorator

Questo *design pattern* strutturale permette di estendere oggetti utilizzando la composizione invece dell'ereditarietà aggiungendo funzionalità ad una classe preesistente.

All'interno del progetto se ne è fatto uso per comporre *widget* specifici dell'applicazione con diversi punti di utilizzo, un esempio consiste nella classe *InputFieldEvv* che al suo interno si compone di un *TextFormField*.

Singleton

Nei punti in cui c'è stata necessita di condividere dati tra più pagine (come nel caso della sessione utente) si è fatto affidamento sul *design pattern Singleton*.
Questo pattern assicura che esista una ed una sola istanza della propria classe alla quale si accede per mezzo di una chiamata statica, lo stesso stato è quindi condiviso da qualsiasi classe sia a conoscenza della sua esistenza.

Il *singleton* è talvolta sottoposto a critiche da parte degli esperti per la sua natura particolare con la quale aggira la creazione di variabili globali, le complicazioni che crea nei momenti di *testing* e per l'assenza di *thread safety*.

Considerando tutti questi svantaggi (l'ultimo non concerne Flutter in quanto Dart è un linguaggio *single thread*) si è cercato ovviamente di minimizzare l'utilizzo del *pattern* (limitato alle classi di contesto *User*, *Events* e *Notifications*). Inoltre è stata discussa la possibilità di sostituire questi casi con l'applicazione della libreria *Riverpod*.

Builder

Affidarsi ad un *builder* può semplificare di molto la costruzione di oggetti complessi. Questo *pattern* creazionale permette la produzione di diverse rappresentazioni di un oggetto per mezzo dello stesso codice di costruzione.

In Flutter (e nel progetto in questione) è possibile incontrarlo sia sotto forma di *widget* vero e proprio (prevede una funzione di *build* la quale restituisce un *widget*), sia come costruttore particolare di *widget standard* come *ListView*.

Lazy Loading

Per minimizzare i tempi di caricamento di un'applicazione (o di una sua vista) talvolta è importante "preparare" solo ciò che può essere definito come il minimo indispensabile. Il *pattern lazy loading* si occupa proprio di questo, costruisce oggetti e *widget* solamente quando questi sono necessari all'utente.

In Flutter queste tecniche vengono applicate ad esempio nel *ListView.builder* o nel *CarouselSlider*.

Observer

Nella programmazione che fa affidamento ad eventi o messaggi è spesso fondamentale basare alcune implementazioni sul *pattern Observer*, questo permette di rimanere in ascolto, quindi essere notificato, nel caso in cui un evento in particolare si verifichi e di reagire di conseguenza.

Ad esempio in Evvvents sono stati inclusi dei *listener* al fine di individuare subito un'iscrizione eseguita con successo e mostrare la schermata corretta istantaneamente.

4.5 Codifica

Le attività di scrittura del codice hanno rispettato la progettazione e l'architettura preimpostate, tuttavia alcuni aspetti dello sviluppo hanno consumato più risorse del previsto per via della non ancora ideale familiarità con il *framework*. Ciononostante i costi pianificati sono stati più che rispettati.

Generalmente la codifica ha cercato di procedere per *feature* completate scegliendole anche in base a quanto man mano offerto dal *backend* sviluppato in contemporanea. Naturalmente nuove versioni di quest'ultimo o revisioni dei requisiti hanno portato a riaperture di *branch* precedentemente chiusi.

Il linguaggio

Il linguaggio Dart si è rivelato perfetto per il compito in quanto prevalentemente intuitivo, semplice da utilizzare e ricco di librerie eccellentemente documentate.
La progettazione di Flutter accuratamente mirata allo sviluppo *mobile* e la sua freschezza lo hanno confermato come uno strumento estremamente potente per la produzione di applicazioni efficienti velocemente.

Debug USB

Avviare l'applicazione e fare attività di *debugging* direttamente su *smartphone* fisici e non simulati è stato permesso dal *debug USB* offerto da Flutter per mezzo del quale è possibile installare il software su un dispositivo collegato al PC sul quale è aperto il progetto.
Questo ha permesso *testing* più meticoloso e di fare prove con entrambi i sistemi operativi *mobile*.
Infine un altro aspetto che ha facilitato la codifica è la possibilità di effettuare *Hot Reload / Hot Restart*, meccaniche per mezzo delle quali è possibile aggiornare via USB il sorgente dell'app attualmente in esecuzione senza interromperla e perfino mantenendone lo stato.

Contenuto

Il codice prodotto dell'applicazione ha principalmente tre scopi:

* quello di dichiarare il *layout* della pagina e la concatenazione di *widget*, talvolta dettata dallo stato attuale della pagina;
* quello di gestire e conservare lo stato della sessione nelle varie pagine;
* quello di comunicare con il *database* per richiedere dati oppure inviarne di nuovi.

Anche se la quantità di schermate viste dall'utente è considerevole queste sono state prodotte utilizzando un approccio standard per via del fatto che generalmente richiedono le stesse funzionalità adattate al caso specifico della pagina attuale.

Si tratta solitamente della classe screen che produce il *layout* di *widget* nella funzione *build()*, le classi del pattern Bloc che ne gestiscono lo stato e le eventuali classi HTTP che scambiano dati con il *backend*.
Naturalmente sono presenti anche casi di logica scritta esclusivamente per un aspetto del codice come potrebbe essere ad esempio la funzione di applicazione dei filtri oppure il validatore degli input.

Vediamo ora alcune delle pagine sviluppate precedute da una semplice descrizione.

Pagina Home

Una volta eseguito l'accesso l'utente visualizza la *home*.
Il layout si basa su insiemi di *card*-evento disposte in diversi caroselli, ne esistono di appositi per esempio per gli eventi in anteprima o per gli eventi di una certa categoria. La logica è stata implementata entro i limiti di quanto lo stato dell'API offriva, è possibile consultare direttamente sull'applicazione gli eventi presenti nel *backend* ma molti dati sono assenti o "mockati". Per questo motivo, nello stato attuale tutti gli eventi sono interamente *mock* e non hanno un corrispondente remoto.

Un'altra importante funzionalità di questa pagina è la possibilità di applicare filtri di varia natura agli eventi visualizzati, questi sono:

* ricerca per parola/e chiave;
* ricerca per categoria;
* ricerca in base alla modalità (online, fisico o ibrido);
* ricerca limitata da una data di inizio e fine.

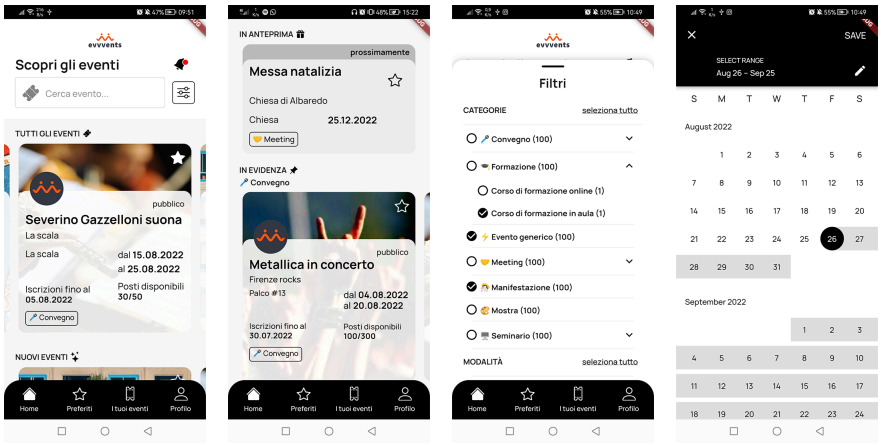

Figura 4.7: schermata degli eventi principali con *slide up panel* per i filtri

Navigazione

Per permettere all'utente di passare da una pagina all'altra tra le principali è stata implementata una *navbar* posizionata nella parte bassa dello schermo, questa ha subito diverse modifiche nel corso di sviluppo sia a livello di grafica che di logica sottostante.

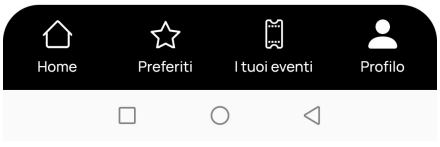

Figura 4.8: Barra di navigazione nella quale è selezionata la pagina del profilo

Pagina Dettaglio Evento

Cliccando su un evento si è ricondotti ad una schermata interamente ad esso dedicata nella quale sono raccolti tutti i suoi dettagli, come il suo relatore, la *location*, come questa possa essere raggiunta ecc.

Da qui è possibile accedere al form d'iscrizione in caso l'utente non sia ancora iscritto oppure accedere al pannello di *check-in* in caso di prenotazione effettuata in precedenza. Allo stato attuale il *check-in* non ha ancora delle regole prestabilite e quindi il QR code presente è un semplice segnaposto.

Infine la *User experience* è stata arricchita da una varietà di messaggi di tipo *alert* che illustrano stati come:

* l'utente ha provato ad iscriversi ad un evento solamente annunciato;
* l'utente ha provato ad iscriversi ad un evento *sold out*;
* l'utente ha attivato le notifiche per l'evento;
* l'utente ha richiesto di visualizzare le istruzioni per il QR code.

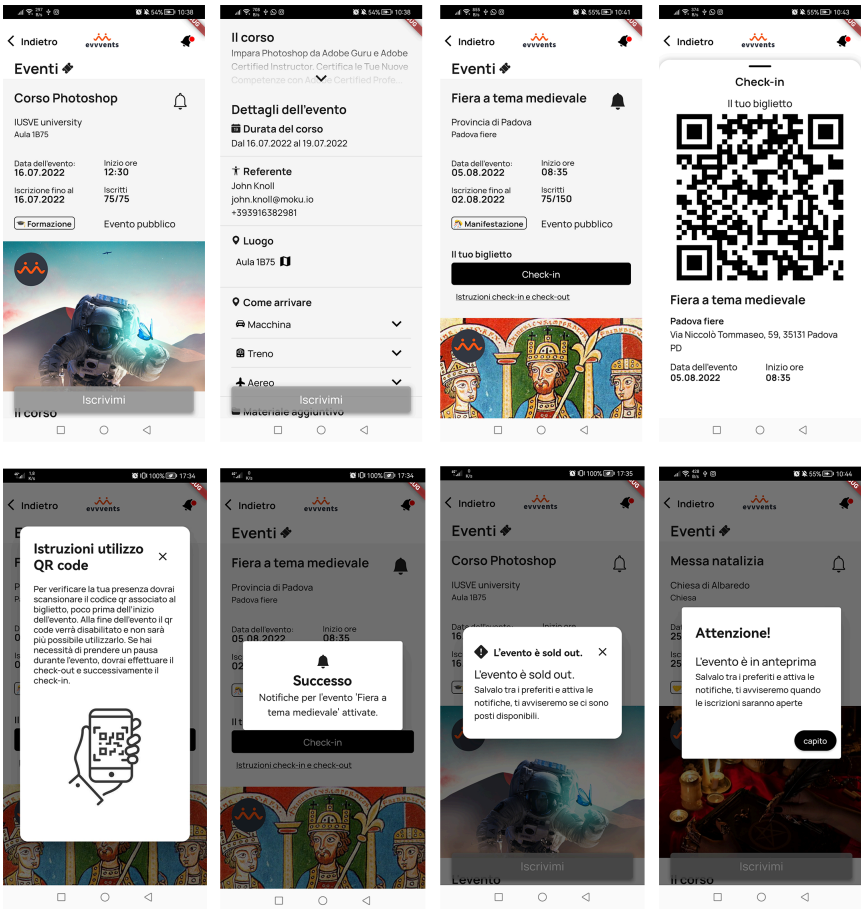

Figura 4.9: Schermate di dettaglio evento con pannello di presentazione QR code e *alert*

Capitolo 5

Conclusioni

Lo stage si è concluso con la soddisfazione generale sia da parte dell'azienda che dello studente, gli obiettivi del progetto sono stati raggiunti ed il prodotto è pronto ad essere evoluto ulteriormente.

5.1 Consuntivo finale

Attività	**Settimana**	**Ore**
Comprensione sistema e obiettivi	1	20
Studio tecnologie	1	20
Sviluppo CyclingHero*	2	40
Progettazione applicazione	3	10
Sviluppo pagine login e signup	3	20
Uteriori contributi a CyclingHero*	3-4	20
Sviluppo Evvvents	4-8	170
Totale		**300**

*Inizialmente il progetto assegnato allo stagista era Cycling Hero, un'applicazione Flutter già in sviluppo nell'azienda; tuttavia il cliente ha deciso di interrompere la collaborazione e di conseguenza è stato cambiato il prodotto da sviluppare.

5.2 Raggiungimento degli obiettivi

* Obbligatori:
  - il lavoro è stato organizzato tramite il *framework* SCRUM, che ha facilitato le comunicazioni tra membri del team e la definizione degli obiettivi intermedi;
  - la comunicazione API/REST è stata implementata per i dati resi disponibili dal *backend*, i rimanenti sono stati mockati in attesa di nuovi sviluppi;
  - l'applicazione possiede un sistema di *login* e *signup* perfettamente funzionanti;
  - l'iscrizione agli eventi è stata correttamente implementata;
  - il processo di *check-in* è presente ma allo stato attuale il QR code non ha uno *standard* deciso dal cliente.
* Desiderabili:
  - il coordinamento con il *product owner* si è rivelato più complesso del previsto, è stata svolta una sola *Sprint Review* al termine dello stage per via di poca disponibilità del cliente;
  - il sistema di messaggistica non ha avuto occasioni nelle quali essere rimosso dal *backlog*;
  - le attività sulle quali investire il *budget* di stage sono state man mano definite dal *project manager* sulla base degli obiettivi del momento, motivo per il quale non c'è stata la possibilità di definire una *suite* di *testing* che seguisse lo sviluppo.

5.3 Conoscenze acquisite

I due mesi trascorsi nell'azienda hanno un chiaro valore istruttivo se si considerano le conoscenze acquisite, sia a livello di tecnologie:

* Flutter;
* API/REST;
* Jira;
* BitBucket;

sia a livello di competenze trasversali:

* lavoro in un team;
* autonomia nello sviluppo;
* comprensione e raggiungimento degli obiettivi richiesti;

5.4 Valutazione personale

Lo stato raggiunto dal progetto è più che ottimale, considerati gli obiettivi iniziali; in generale il periodo di stage ha permesso una graduale prima immersione nel mondo del lavoro nel settore preparando lo studente dal punto di vista pratico e chiarendo eventuali dubbi che l'assenza di esperienza aveva creato.

Capitolo 6

Glossario

Agile

Nella metodologia *agile* si raggruppano un insieme di metodi di sviluppo *software* dichiarati nel Manifesto Agile, propongono un approccio meno strutturato e orientato al prodotto ed al cliente.

API/REST

Acronimo di *Application Programming Interface*, permette la comunicazione tra diversi componenti di un *software* attraverso la definizione di protocolli di richiesta e risposta. RESTful rappresenta una serie di vincoli architetturali per una comunicazione basata sul protocollo *HTTP*.

Backend

Abbreviato anche come *BE*, in informatica denota ciò che l'utente non vede con cui non interagisce direttamente ma che permette il funzionamento dell'applicativo, riceve i dati utente elaborati dal *frontend*.

Code review

Per accertare la qualità del codice si può talvolta fare affidamento su una sua revisione da parte di un altro sviluppatore che ne visualizza i contenuti.

Frontend

Abbreviato anche come *FE*, in informatica denota la parte visibile all'utente con la quale interagisce direttamente, si occupa dell'acquisizione dei dati in ingresso e di renderli fruibili al *backend*.

Product owner

Il *product owner* è la persona responsabile di far emergere il prodotto con il massimo valore aggiunto possibile entro la data desiderata; gestisce i rapporti con gli *stakeholder* e monitora lo sviluppo.

Project manager

Talvolta abbrevviato come *PM*, il *project manager* è il responsabile della gestione del progetto, del quale valuta i rischi, coordina i contributori e pianifica gli sviluppi.

Pull request

Generalmente nell'utilizzo di un *Version Control System* in grado di ospitare vari *branch*, una *pull request* è una richiesta fatta all'autore o comunque contributore con più permessi al fine di effettuare un *merge* del *branch* al quale si sta lavorando.

Software as a Service

Abbreviato anche come *SaaS*, è un servizio reso disponibile all'utente attraverso il web, generalmente basanodosi su tecnologie API/REST.

UML

Abbreviazione di *Unified Modeling Language*, è un linguaggio di modellazione adatto allo sviluppo con il paradigma della programmazione ad oggetti in quanto definisce uno standard di formalismi per l'illustrazione della progettazione di un software.
Ne fanno parte i diagrammi dei casi d'uso, di sequenza e delle classi.

Wireframe

Sono illustrazioni organizzative schematiche dei contenuti presentati dal *software* che comunicano l'idea e le funzionalità del progetto.

Appendice A

Analisi dei requisiti tecnica

In questa appendice viene raccolto lo studio più formale dei requisiti dell'applicativo basato sui diagrammi dei casi d'uso del linguaggio UML.

A.1 Casi d'uso

Per lo studio dei casi d'uso e dei requisiti del prodotto sono stati compilati i seguenti diagrammi UML.

Attori

Gli attori che andranno ad interagire con l'applicazione sono di tre tipi:

* Utente non registrato: che accede alla piattaforma per la prima volta;
* Utente registrato: che possiede un profilo al quale non ha ancora effettuato l'accesso;
* Utente partecipante: che possiede un profilo ed ha eseguito l'accesso ad esso.

Scenario principale: Evvvents

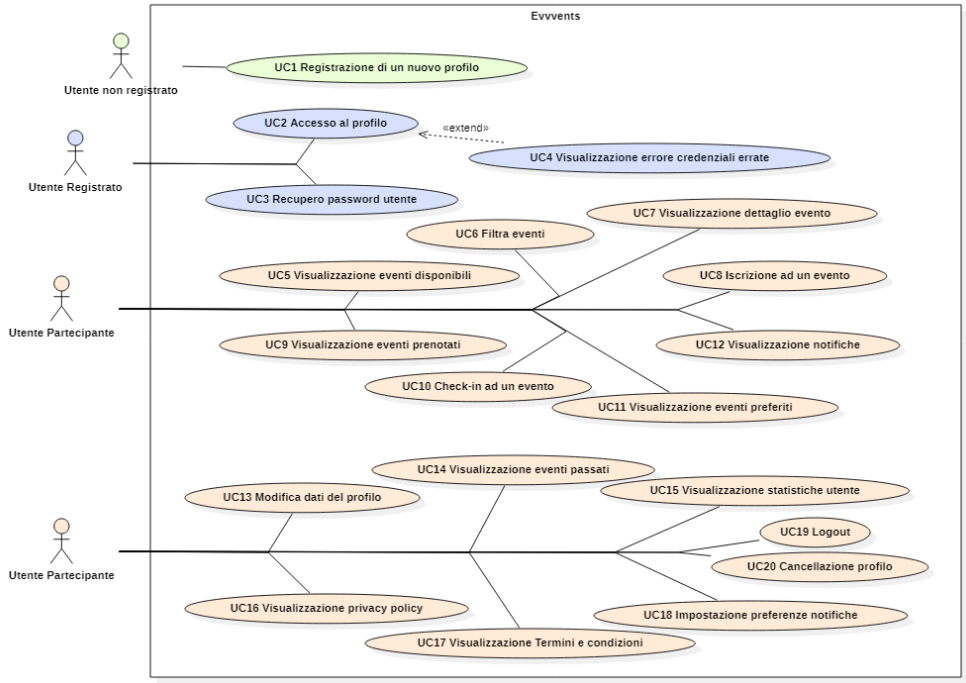

Figura A.1: Use Case - Scenario principale

UC: Utente non registrato

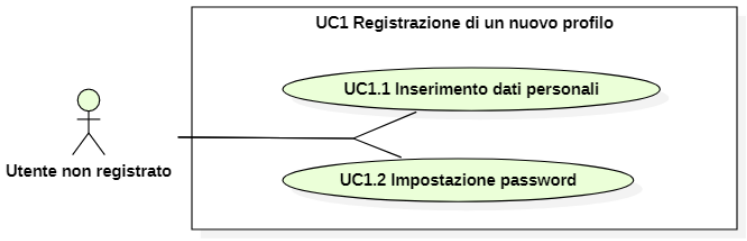

Figura A.2: Use Case 1 - Registrazione di un nuovo profilo

UC1: Registrazione di un nuovo profilo

Attori Principali: Utente non registrato.

Precondizioni: L'utente ha aperto l'applicazione.

Descrizione: L'utente completa i passaggi per la registrazione di un profilo personale su Evvvents.

Postcondizioni: L'utente possiede dei dati con i quali accedere ai rimanenti servizi.

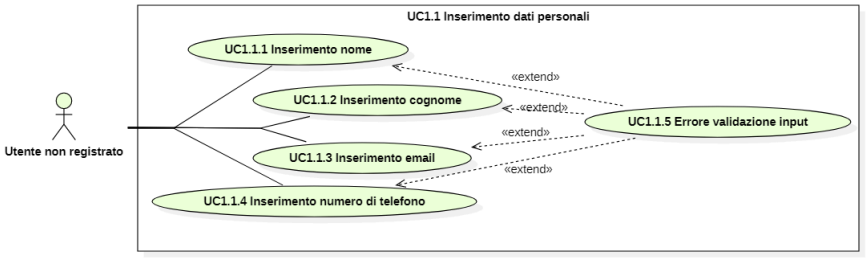

Figura A.3: Use Case 1.1 - Inserimento dati personali

UC1.1: Inserimento dati personali

Attori Principali: Utente non registrato.

Precondizioni: L'utente ha iniziato il processo di registrazione.

Descrizione: L'utente inserisce i dati a lui richiesti (nome, cognome, email e telefono) e corregge in caso di fallita validazione dell'input.

Postcondizioni: L'utente può procede al prossimo passaggio della registrazione profilo.

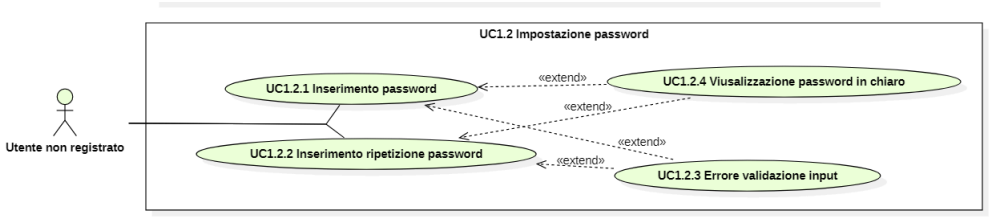

Figura A.4: Use Case 1.2 - Impostazione password

UC1.2: Impostazione password

Attori Principali: Utente non registrato.

Precondizioni: L'utente ha iniziato il processo di registrazione ed ha inserito i propri dati.

Descrizione: L'utente inserisce la propria password (due volte per una questione di sicurezza) e questa viene validata per la complessità.

Postcondizioni: L'utente ha confermato la registrazione e può attivare il proprio profilo.

UC: Utente registrato

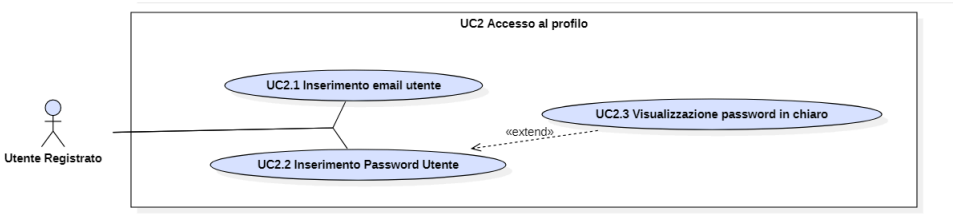

Figura A.5: Use Case 2 - Accesso al profilo

UC2: Accesso al profilo

Attori Principali: Utente registrato.

Precondizioni: L'utente ha un profilo registrato su Evvvents.

Descrizione: L'utente inserisce le proprie email e password la cui correttezza viene successivamente verificata attraverso il bottone di conferma.

Postcondizioni: L'utente ha effettuato l'accesso.

Estensione UC4: L'utente ha inserito credenziali errate e visualizza un messaggio di errore di conseguenza.

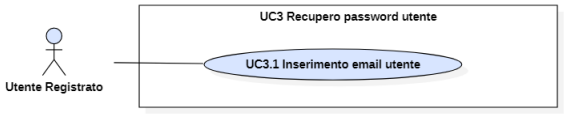

Figura A.6: Use Case 3 - Recupero password utente

UC3: Recupero password utente

Attori Principali: Utente registrato.

Precondizioni: L'utente ha un profilo registrato su Evvvents.

Descrizione: L'utente inserisce la propria email al fine di poter essere contattato e da lì reimpostare la propria password.

Postcondizioni: La nuova password è stata aggiornata nel database di Evvvents e l'utente può ora accedere con facilità.

UC: Utente partecipante

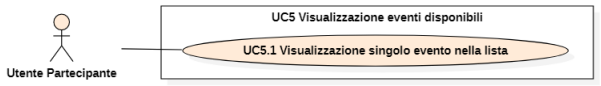

Figura A.7: Use Case 5 - Visualizzazione eventi disponibili

UC5: Visualizzazione eventi disponibili

Attori Principali: Utente partecipante.

Precondizioni: L'utente ha effettuato l'accesso al suo profilo Evvvents.

Descrizione: L'utente visualizza la lista degli eventi a lui disponibili al momento.

Postcondizioni: L'utente ha visualizzato la lista degli eventi disponibili.

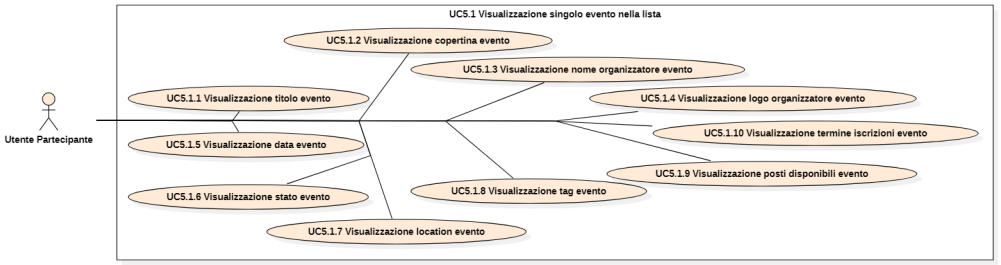

Figura A.8: Use Case 5.1 - Visualizzazione singolo evento nella lista

UC5.1: Visualizzazione singolo evento nella lista

Attori Principali: Utente partecipante.

Precondizioni: L'utente ha effettuato l'accesso al suo profilo Evvvents.

Descrizione: L'utente visualizza la card dell'evento contenente tutti i dati principali (titolo, copertina, organizzatore, logo organizzatore, data, stato, nome location, tag associato, posti disponibili e data termine iscrizioni).

Postcondizioni: L'utente ha visualizzato l'evento e tutte le sue informazioni presentate.

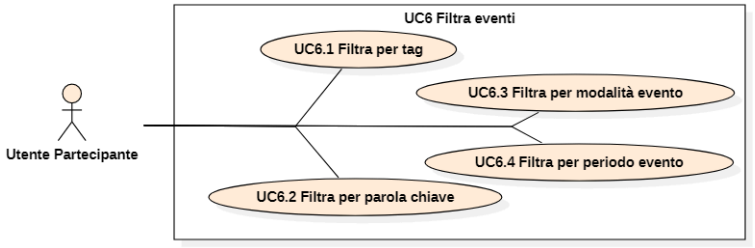

Figura A.9: Use Case 6 - Filtra eventi

UC6: Filtra eventi

Attori Principali: Utente partecipante.

Precondizioni: L'utente ha visualizzato gli eventi disponibili.

Descrizione: L'utente visualizza un sottoinsieme degli eventi disponibili sottoponendo la lista ad una combinazione di quattro filtri differenti (per tag, per modalità, per data, per parola chiave).

Postcondizioni: L'utente ha visualizzato l'insieme di eventi richiesto.

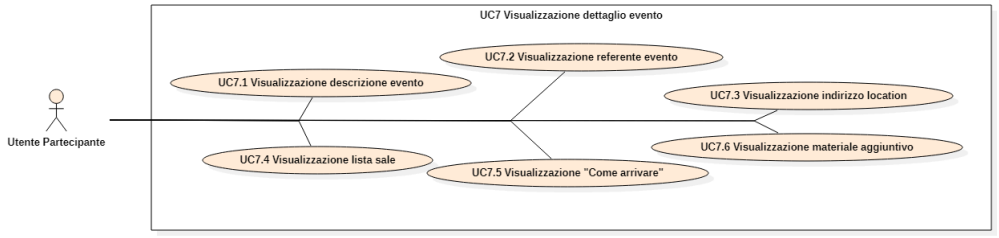

Figura A.10: Use Case 7 - Visualizzazione dettaglio evento

UC7: Visualizzazione dettaglio evento

Attori Principali: Utente partecipante.

Precondizioni: L'utente ha cliccato su un evento nella lista.

Descrizione: L'utente visualizza i dati precedentemente esposti nelle card con l'aggiunta di ulteriori informazioni (descrizione, referente, indirizzo location, lista delle sale, "come arrivare", materiale aggiuntivo).

Postcondizioni: L'utente ha visualizzato le informazioni dell'evento spefico richiesto.

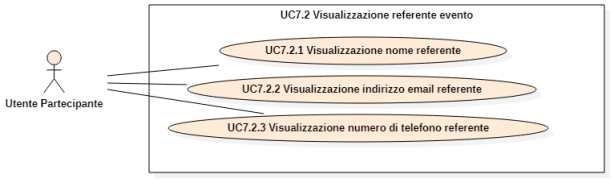

Figura A.11: Use Case 7.2 - Visualizzazione referente evento

UC7.2: Visualizzazione referente evento

Attori Principali: Utente partecipante.

Precondizioni: L'utente sta visualizzando i dettagli di un evento.

Descrizione: L'utente visualizza i dati legati al referente dell'evento (nome, email e telefono).

Postcondizioni: L'utente ha visualizzato le informazioni del referente dell'evento spefico richiesto.

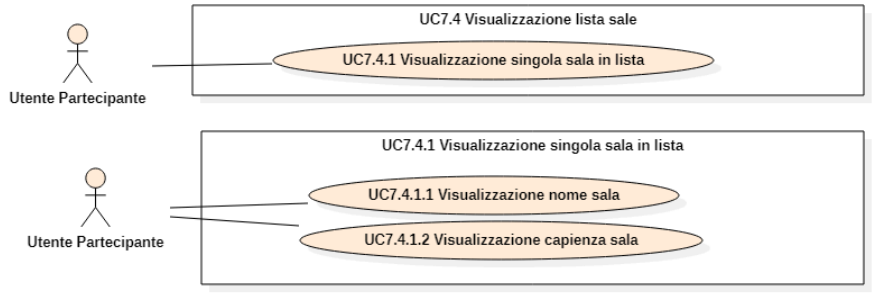

Figura A.12: Use Case 7.4 - Visualizzazione lista sale

UC7.4: Visualizzazione lista sale

Attori Principali: Utente partecipante.

Precondizioni: L'utente sta visualizzando i dettagli di un evento.

Descrizione: L'utente visualizza la lista di sale dell'evento (nome e capienza).

Postcondizioni: L'utente ha visualizzato le informazioni delle sale dell'evento spefico richiesto.

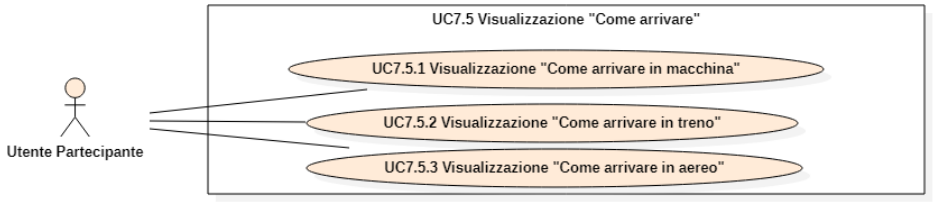

Figura A.13: Use Case 7.5 - Visualizzazione "come arrivare"

UC7.5: Visualizzazione "come arrivare"

Attori Principali: Utente partecipante.

Precondizioni: L'utente sta visualizzando i dettagli di un evento.

Descrizione: L'utente visualizza una serie di istruzioni su come raggiungere la location con tre mezzi di trasporto diversi (macchina, treno, aereo).

Postcondizioni: L'utente ha visualizzato le informazioni su come raggiungere la location.

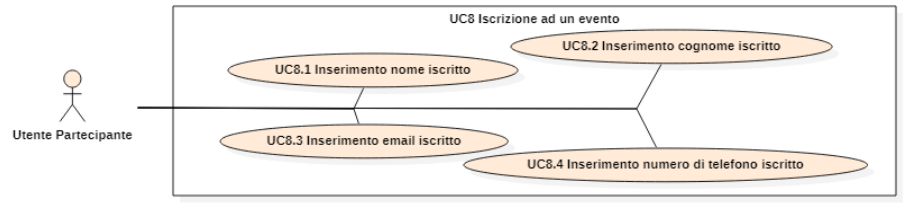

Figura A.14: Use Case 8 - Iscrizione ad un evento

UC8: Iscrizione ad un evento

Attori Principali: Utente partecipante.

Precondizioni: L'utente ha visualizzando i dettagli di un evento.

Descrizione: L'utente inserisce le proprie informazioni al fine di dichiarare la sua partecipazione all'evento.

Postcondizioni: L'utente viene inserito come iscritto all'evento nel database.

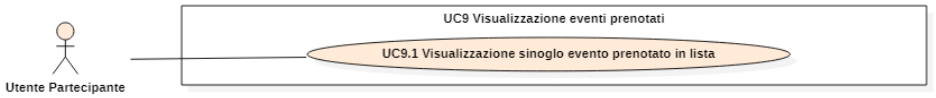

Figura A.15: Use Case 9 - Visualizzazione eventi prenotati

UC9: Visualizzazione eventi prenotati

Attori Principali: Utente partecipante.

Precondizioni: L'utente ha effettuato l'accesso.

Descrizione: L'utente visualizza gli eventi ai quali si è iscritto.

Postcondizioni: L'utente ha visualizzato gli eventi ai quali è iscritto.

UC10: Check-in ad un evento

Attori Principali: Utente partecipante.

Precondizioni: L'utente ha effettuato l'accesso e si è iscritto all'evento in questione.

Descrizione: L'utente visualizza il pannello con QR code e altre informazioni per effettuare il check-in.

Postcondizioni: L'utente è stato segnato come presente all'evento.

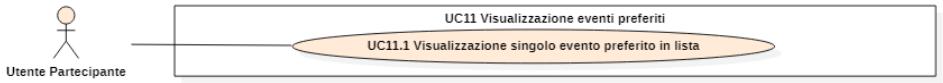

Figura A.16: Use Case 11 - Visualizzazione eventi preferiti

UC11: Visualizzazione eventi preferiti

Attori Principali: Utente partecipante.

Precondizioni: L'utente ha effettuato l'accesso.

Descrizione: L'utente visualizza gli eventi che ha salvato tra i preferiti.

Postcondizioni: L'utente ha visualizzato gli eventi preferiti.

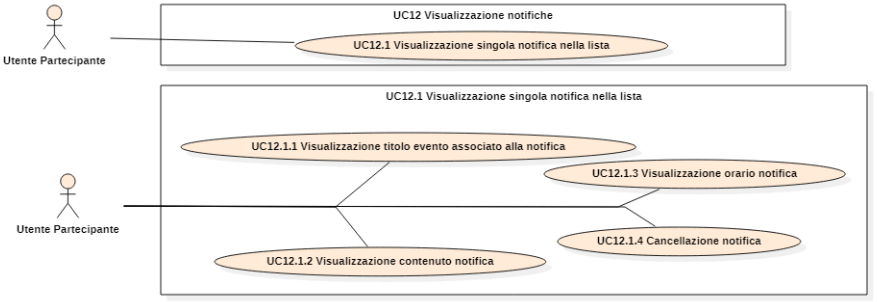

Figura A.17: Use Case 12 - Visualizzazione notifiche

UC12: Visualizzazione notifiche

Attori Principali: Utente partecipante.

Precondizioni: L'utente ha effettuato l'accesso.

Descrizione: L'utente visualizza le notifiche (titolo, contenuto, orario) a lui dedicate o associate ad un evento per il quale ha attivato le notifiche, può inoltre cancellarne alcune.

Postcondizioni: L'utente ha visualizzato le notifiche.

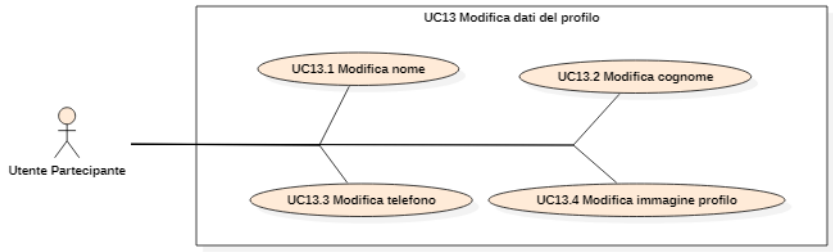

Figura A.18: Use Case 13 - Modifica dati del profilo

UC13: Modifica dati del profilo

Attori Principali: Utente partecipante.

Precondizioni: L'utente ha effettuato l'accesso.

Descrizione: L'utente apporta le modifiche che preferisce ai dati da lui precendentemente inseriti (nome, cognome, telefono, immmagine del profilo).

Postcondizioni: L'utente ha confermato o annullato le modifiche, in caso di conferma esse sono state aggiornate nel backend.

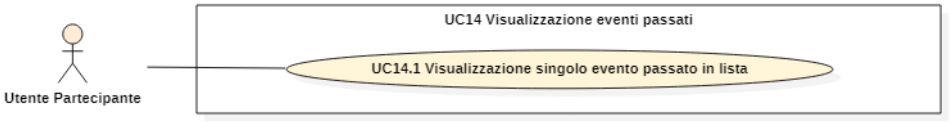

Figura A.19: Use Case 14 - Visualizzazione eventi passati

UC14: Visualizzazione eventi passati

Attori Principali: Utente partecipante.

Precondizioni: L'utente ha effettuato l'accesso.

Descrizione: L'utente visualizza gli eventi ai quali ha partecipato raccolti in una semplice lista.

Postcondizioni: L'utente ha visualizzato gli eventi passati.

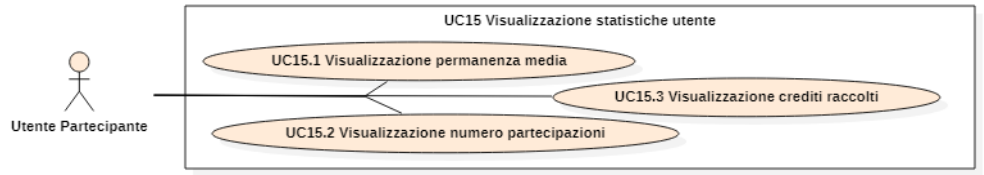

Figura A.20: Use Case 15 - Visualizzazione statistiche utente

UC15: Visualizzazione statistiche utente

Attori Principali: Utente partecipante.

Precondizioni: L'utente ha effettuato l'accesso.

Descrizione: L'utente visualizza dati interessanti raccolti durante il suo utilizzo dell'applicazione quali la permanenza media, i crediti formativi ed il numero di partecipazioni.

Postcondizioni: L'utente ha visualizzato le proprie statistiche.

UC16: Visualizzazione privacy policy

Attori Principali: Utente partecipante.

Precondizioni: L'utente ha effettuato l'accesso.

Descrizione: L'utente visualizza gli articoli della privacy policy di Evvvents.

Postcondizioni: L'utente ha visualizzato la privacy policy.

UC17: Visualizzazione termini e condizioni

Attori Principali: Utente partecipante.

Precondizioni: L'utente ha effettuato l'accesso.

Descrizione: L'utente visualizza i termini e le condizioni di Evvvents.

Postcondizioni: L'utente ha visualizzato gli articoli che lo interessavano.

UC18: Impostazione preferenze notifiche

Attori Principali: Utente partecipante.

Precondizioni: L'utente ha effettuato l'accesso.

Descrizione: L'utente sceglie riguardo a quali aggiornamenti essere notificato.

Postcondizioni: L'utente ha impostato le proprie preferenze sulle notifiche.

UC19: Logout

Attori Principali: Utente partecipante.

Precondizioni: L'utente ha effettuato l'accesso.

Descrizione: L'utente esce dal proprio profilo e chiude la sessione attuale.

Postcondizioni: L'utente non è più loggato.

UC20: Cancellazione profilo

Attori Principali: Utente partecipante.

Precondizioni: L'utente ha effettuato l'accesso.

Descrizione: L'utente cancella il proprio profilo e tutti i dati ad esso legati..

Postcondizioni: L'utente non è più registrato.

A.2 Tracciamento dei requisiti

A partire dai casi d'uso dell'applicazione si è poi giunti ai requisiti i quali sono caratterizzati da un codice composto da:

* la lettera **R**: Requisito;
* una lettera tra
  - **F**: Funzionale;
  - **Q**: Qualitativo;
  - **V**: di Vincolo;
* un numero che identifica il tipo di utente al quale è dedicato il requisito;
* un punto;
* un numero progressivo che indica il requisito univocamente.

I requisiti sono traccati nella tabella seguente.

Tabella A.1: Tabella di tracciamento dei requisiti

Codice	Descrizione	Fonte	Stato
RF1.1	L'utente deve potersi registrare	UC1	Completato
RF2.1	L'utente deve poter accedere al proprio profilo	UC2	Completato
RF2.2	L'utente deve poter avviare il sistema di recupero password	UC3	In lavorazione
RF3.1	L'utente deve poter visualizzare gli eventi disponibili	UC5	Completato
RF3.2	L'utente deve poter filtrare tra gli eventi disponibili	UC6	Completato
RF3.3	L'utente deve poter visualizzare i dettagli di un evento	UC7	Completato

Codice	Descrizione	Fonte	Stato
RF3.4	L'utente deve potersi iscrivere ad un evento	UC8	Completato
RF3.5	L'utente deve poter visualizzare la lista degli eventi a cui è iscritto	UC9	Completato
RF3.6	L'utente deve poter effettuare il check-in ad un evento	UC10	In lavorazione
RF3.7	L'utente deve poter visualizzare la lista degli eventi preferiti	UC11	Completato
RF3.8	L'utente deve poter visualizzare le proprie notifiche	UC12	In lavorazione
RF3.9	L'utente deve poter modificare i propri dati del profilo	UC13	Completato
RF3.10	L'utente deve poter visualizzare gli eventi passati	UC14	Completato
RF3.11	L'utente deve poter visualizzare le proprie statistiche	UC15	In lavorazione
RF3.12	L'utente deve poter visualizzare la privacy policy	UC16	Completato
RF3.13	L'utente deve poter visualizzare i termini e le condizioni	UC17	Completato
RF3.14	L'utente deve poter impostare le proprie preferenze sulle notifiche	UC18	Completato
RF3.15	L'utente deve poter effettuare il logout	UC19	Completato
RF3.16	L'utente deve poter cancellare il proprio profilo	UC20	In lavorazione

Codice	Descrizione	Fonte	Stato
RQ1.1	Le scelte progettuali e di architettura devono essere apertamente discusse e documentate	Committente	Completato
RQ1.2	L'applicazione deve essere accessibile da qualsiasi dispositivo mobile recente	Committente	Completato
RQ1.3	L'applicazione deve chiarire il proprio stato all'utente in qualsiasi momento con messaggi di errore esplicativi	Committente	Completato

Codice	Descrizione	Fonte	Stato
RV1.1	L'applicazione deve essere sviluppata attraverso il framework Flutter	Committente	Completato
RV1.2	L'applicazione deve avere una versione android stabile	Committente	Completato
RV1.3	L'applicazione deve avere una versione iOS stabile	Committente	Completato

Nota: l'applicazione si trova ancora in stato di sviluppo e verrà portata avanti dell'azienda, i requisiti dichiarati come "in lavorazione" si trovano in tale stato in quanto è stato fatto il possibile nei limiti di quanto offerto dal resto dell'architettura.