



Documentazione progetto

EasyBooking

Componenti del gruppo:

Bikhtancer Mohammed Amine, matr. 1057874

Lorenzi Luca, matr. 1057482

Maffioletti Gianluca, matr. 1059062

Matera Simone, matr. 1059936

Laurea Magistrale in Ingegneria Informatica

Corso di Progettazione, Algoritmi e Computabilità

Cod. 38090-MOD1

Università degli Studi di Bergamo

A.A. 2021/2022

Indice

1	Introduzione al progetto	11
1.1	Introduzione	11
1.2	Toolchain	12
1.2.1	Spring framework	13
2	Iterazione 0	15
2.1	Casi d'uso	15
2.1.1	Attori.....	15
2.1.2	Use case stories: Cliente	15
2.1.3	Use case stories: Proprietario	17
2.1.4	Use case stories: Admin	18
2.1.5	Use case stories: Sistema di geolocalizzazione	19
2.1.6	Use case stories: Sistema di pagamento	19
2.1.7	Use case stories: Servizio mail	19
2.2	Requisiti	20
2.2.1	Requisiti funzionali	20
2.2.2	Requisiti non funzionali	24
2.3	Fully dressed descriptions	24
2.3.1	UC1: Registrazione utente.....	25
2.3.2	UC2: Log-in nell'applicazione	26
2.3.3	UC3: Log-out dall'applicazione	27
2.3.4	UC4: Modifica profilo utente	27

2.3.5	UC5: Cancellazione utente	28
2.3.6	UC6: Inserimento alloggio	29
2.3.7	UC7: Modifica alloggio.....	30
2.3.8	UC8: Rimozione alloggio.....	31
2.3.9	UC9: Sponsorizzazione alloggio	32
2.3.10	UC10: Visualizzazione profilo utente	33
2.3.11	UC11: Visualizzazione scheda alloggio	34
2.3.12	UC12: Ricerca alloggi	35
2.3.13	UC13: Inserimento prenotazione.....	36
2.3.14	UC14: Modifica prenotazione	37
2.3.15	UC15: Accettazione/rifiuto prenotazione	38
2.3.16	UC16: Disdetta prenotazione.....	39
2.3.17	UC17: Conferma pernottamento	40
2.3.18	UC18: Inserimento recensione alloggio	40
2.3.19	UC19: Inserimento recensione utente.....	41
2.3.20	UC20: Inserimento reclami	42
2.3.21	UC21: Gestione reclami	43
2.3.22	UC22: Sospensione utente o alloggio.....	43
2.3.23	UC23: Gestione chat.....	44
2.4	Use case diagram	45
2.5	Architettura	47
2.5.1	Architettura hardware	47
2.5.2	Architettura software	48
2.5.3	Pattern architetturale.....	49
3	Iterazione 1	53
3.1	Component diagram.....	53

3.2	Interface diagram	56
4	Iterazione 2	58
4.1	Selezione delle funzioni da implementare	58
4.1.1	Use Case descriptions.....	58
4.2	Registrazione	61
4.2.1	Funzionamento	61
4.2.2	Component diagram	62
4.2.3	Sequence diagram.....	64
4.3	Login - Logout.....	65
4.3.1	Spring security	65
4.4	Front-End	66
4.4.1	Interazione con il back-end.....	66
4.4.2	Retrofit.....	66
4.4.3	Gson parser	69
4.4.4	Android View e ViewGroup.....	69
4.5	Test e analisi del componente	75
4.5.1	Analisi statica	75
4.5.2	Analisi dinamica	78
5	Iterazione 3	84
5.1	Selezione delle funzioni da implementare	84
5.1.1	Use Case descriptions.....	84
5.2	Ricerca singola.....	87
5.2.1	Pseudocodice algoritmo.....	88
5.2.2	Codifica Java e complessità.....	89
5.2.3	Query	93
5.3	Ricerca multipla	94

5.3.1	Pseudocodice algoritmo.....	94
5.3.2	Codifica Java e complessità.....	97
5.3.3	Query	106
5.4	Component diagram.....	107
5.4.1	DistanceMatrixApi	109
5.5	Sequence diagram	110
5.6	Front-End	113
5.7	Test e analisi del componente	118
5.7.1	Analisi statica	118
5.7.2	Analisi dinamica.....	124
6	Conclusione.....	131
7	Guida all'installazione e all'uso	136

Indice delle figure

Figura 1: Use Case diagram.....	46
Figura 2: Architettura del sistema	48
Figura 3: Deployment diagram.....	49
Figura 4: MVC vs MVP	50
Figura 5: Component diagram MVP	54
Figura 6: Component diagram black-box	55
Figura 7: Interface diagram	56
Figura 8: Component diagram del servizio di registrazione.....	63
Figura 9: Sequence diagram del servizio di registrazione - Richiesta.....	64
Figura 10: Sequence diagram del servizio di registrazione - Conferma.....	64
Figura 11: Struttura form di login.....	65
Figura 12: Interfaccia UserAPI realizzata con Retrofit	67
Figura 13: Codice per gestione login.....	68
Figura 14: Register Activity	71
Figura 15: Register Activity - Compilazione campi.....	72
Figura 16: Login Activity	73
Figura 17: Login Activity - Test output	74
Figura 18: Analisi statica del codice – Iterazione 2.....	75
Figura 19: Distance – Model e Presenter GestioneUtenze	76
Figura 20: Legenda grafici CodeMR.....	77
Figura 21: Quality attributes - Model.GestioneUtenze	77
Figura 22: Quality attributes - Presenter.GestioneUtenze	77
Figura 23: Test registrazione con PostMan	78
Figura 24: HTTP response code per registrazione avvenuta.....	79
Figura 25: HTTP response code per errore durante la registrazione - Utente già registrato ma non confermato	79
Figura 26: HTTP response code per errore durante la registrazione - Utente già registrato	79
Figura 27: Mail di conferma registrazione	79

Figura 28: Pagina HTML per conferma registrazione di un account	80
Figura 29: Pagina HTML per errore durante la registrazione di un account – Link già utilizzato	80
Figura 30: Pagina HTML per errore durante la registrazione di un account – Link scaduto	81
Figura 31: Test positivo Login	81
Figura 32: Test negativo Login	81
Figura 33: Output delle chiamate di test sulla funzione di Login	82
Figura 34: Pseudocodice dell'algoritmo di ricerca singola	88
Figura 35: Metodo singleResearch()	90
Figura 36: Metodo checkDistance()	92
Figura 37: Query findAvailableApartmentsPerNumGuests()	93
Figura 38: Query findAvailableApartmentsPerNumGuestsAndMaxPricePerNight() ...	93
Figura 39: Pseudocodice dell'algoritmo di ricerca multipla	95
Figura 40: Pseudocodice dell'algoritmo di ricerca delle combinazioni	96
Figura 41: Metodo multipleResearch() – prima parte	98
Figura 42: Metodo multipleResearch() – seconda parte.....	100
Figura 43: Algoritmo noto per la generazione di combinazioni.....	101
Figura 44: Esempio albero delle combinazioni	102
Figura 45: Metodo ricorsivo apartmentCombination()	103
Figura 46: Query findAvailableApartmentsPerMaxNumGuests()	106
Figura 47: Query findAvailableApartmentsPerMaxNumGuestsAndMaxPricePerNight()	
.....	106
Figura 48: Component diagram della funzione ricerca	108
Figura 49: Esempio di risposta JSON della DistanceMatrixAPI	110
Figura 50: Sequence diagram ricerca singola.....	111
Figura 51: Sequence diagram ricerca multipla.....	112
Figura 52: Search Activity.....	114
Figura 53: Body della richiesta HTTP in Json	114
Figura 54: ResultSearchActivity e ApartmentRecViewAdapter.....	115
Figura 55: ResultSearchActivity e ApartmentRecViewAdapter versione estesa.....	116
Figura 56: ApartmentActivity	117

Figura 57: Analisi statica del codice – Iterazione 3.....	118
Figura 58: Distance - Model GestionePrenotazioni e GestioneAlloggi	119
Figura 59: Dettaglio dell'Efferent coupling e Afferent Coupling.....	119
Figura 60: Distance: Presenter GestionePrenotazioni e GestioneAlloggi	120
Figura 61: Quality attributes - Model.GestioneAlloggi	121
Figura 62: Quality attributes - Model.GestionePrenotazione	122
Figura 63: Quality attributes - Presenter.GestioneAlloggi	122
Figura 64: Quality attributes - Presenter.GestionePrenotazione	123
Figura 65: Richiesta Json per la ricerca di alloggi.....	124
Figura 66: Test ricerca multipla tramite PostMan.....	125
Figura 67: Test ricerca singola tramite PostMan.....	126
Figura 68: Test ricerca singola – prezzo inferiore	126
Figura 69: Test ricerca singola - date occupate	127
Figura 70: Test ricerca singola - numero eccessivo di persone	127
Figura 71: Esempio di test del metodo multipleResearch()	129
Figura 72: Risultato del test sul metodo multipleResearch()	130
Figura 73: Copertura JUnit tests.....	132
Figura 74: Quality attributes – Overall project.....	133
Figura 75: Metric Values by Package – C3	134
Figura 76: Map chart by C3 value	135
Figura 77: Guida alla creazione di un dispositivo virtuale – Parte 1.....	137
Figura 78: Guida alla creazione di un dispositivo virtuale - Parte 2	137
Figura 79: Guida alla creazione di un dispositivo virtuale - Parte 3	138
Figura 80: Guida alla creazione di un dispositivo virtuale - Parte 4	139
Figura 81: Guida alla creazione di un dispositivo virtuale - Parte 5	140
Figura 82: Modifica parametro baseUrl - Parte 1	141
Figura 83: Modifica parametro baseUrl - Parte 2.....	141
Figura 84: Esempio ricerca standard	142
Figura 85: Esempio ricerca avanzata.....	143
Figura 86: Esempio ricerca per gruppi di alloggi	144

Indice delle tabelle

Tabella 1: Requisiti Funzionali - Gestione utenze	21
Tabella 2: Requisiti Funzionali - Gestione alloggi.....	21
Tabella 3: Requisiti Funzionali - Ricerca alloggi.....	22
Tabella 4: Requisiti Funzionali - Gestione prenotazioni	22
Tabella 5: Requisiti Funzionali - Gestione pagamenti	23
Tabella 6: Requisiti Funzionali - Gestione recensioni e reclami.....	23
Tabella 7: Requisiti Funzionali - Gestione comunicazioni	24
Tabella 8: Complessità algoritmo di ricerca delle combinazioni	105

1 Introduzione al progetto

1.1 Introduzione

Lo scopo del progetto consiste nello sviluppo di un sistema di booking, ossia un'applicazione che permette di cercare e di prenotare delle case/appartamenti/stanze per i propri viaggi.

L'idea è quella di realizzare una web-app che consenta agli utenti, tramite un'interfaccia user-friendly, di registrarsi, collegarsi al proprio account, visualizzare, inserire e modificare le proprie informazioni personali (quali credenziali, indirizzo mail, password, metodi di pagamento, ...), navigare tra gli alloggi disponibili, filtrandoli attraverso l'utilizzo di opportuni filtri (date di pernottamento, numero ospiti, prezzo massimo, distanza massima, tags) e scegliere quello che più si avvicina alle proprie esigenze, portando a termine la prenotazione e il relativo pagamento.

L'applicazione offre la possibilità di diventare proprietario a tutti gli utenti che desiderano mettere in affitto un alloggio per la community. Questo avviene in maniera automatica quando l'utente registrato inserisce per la prima volta un alloggio nel sistema. L'utente proprietario, quindi, ha a disposizione nella propria applicazione funzionalità aggiuntive per la visualizzazione e gestione dei propri alloggi (inserimento, modifica, rimozione). Inoltre, può visualizzare le prenotazioni ricevute per i propri alloggi, e decidere se accettarle, rifiutarle, o eventualmente disdirle in un momento successivo.

Il sistema offre anche un servizio di recensione, sia sugli alloggi da parte dei clienti, sia sui clienti da parte dei proprietari. Grazie a questo gli utenti hanno a disposizione ulteriori informazioni per poter effettuare le proprie scelte.

Oltre al servizio di recensione, è prevista la possibilità di presentare dei reclami al gestore del sistema, nel caso un utente abbia avuto un'esperienza particolarmente negativa con un cliente, un proprietario o un alloggio particolare. In questo caso, interviene uno degli

admin, che è la figura incaricata di verificare le controversie, richiedendo informazioni e prove di quanto reclamato, ed eventualmente di procedere con la sospensione di un’utenza o di un alloggio.

Gli utenti del sistema possono comunicare tra loro tramite un apposito servizio di chat, presente all’interno dell’applicazione.

1.2 Toolchain

Per lo sviluppo del sistema si è scelto di seguire la metodologia agile. Di conseguenza, nella valutazione delle tecnologie, sono state scelte quelle più adatte e coerenti con questa filosofia operativa e che permettessero di abbattere i costi e i tempi di sviluppo, nonché una programmazione leggera e flessibile.

- **Modellazione**
 1. Use case diagram, deployment diagram, component diagram, sequence diagram: draw.io, UMLet
- **Implementazione software**
 1. Linguaggio di programmazione: Java
 2. IDE: IntelliJ IDEA, Android Studio
 3. API development: Spring, Maven, Postman
 4. DBMS: PostgreSQL (hosting su piattaforma cloud Heroku), interazione con JPA
- **Analisi**
 1. Analisi statica: CodeMR
 2. Analisi dinamica: JUnit
- **Documentazione, versioning e organizzazione**
 1. Documentazione: Microsoft Word
 2. Versioning: Git e GitHub
 3. Repository: OneDrive
 4. Organizzazione: gruppi di lavoro e riunioni su Microsoft Teams, sessioni di Pair-Programming in presenza o in remoto tramite utilizzo del plugin “Code With Me”, utilizzo di modalità “Timeboxing”

1.2.1 Spring framework

Spring è il framework più consono con questo tipo di approccio.

Nei framework tradizionali, di norma, si ha che quando una classe *A* ha come parametro un’istanza della classe *B*, si dice che *A* dipende da *B*. Le dipendenze di questo tipo possono causare vari problemi, in quanto una modifica di *B* rischia di propagarsi e influenzare anche la logica interna della classe *A*. Uno sviluppatore, dunque, dopo aver fatto le modifiche in *B*, dovrà occuparsi di fare in modo che la modifica non causi problemi nelle classi dipendenti da questa. Ad esempio: se viene modificata la firma del costruttore di *B*, lo sviluppatore è costretto a propagare la modifica dovunque il costruttore venga richiamato.

La metodologia agile vuole invece che il programmatore si concentri sullo sviluppo di logica e non sulla manutenzione del codice. Con Spring si evitano problemi precipitati in quanto, grazie alle *dependency injection*, le classi possono essere in dipendenza tra di loro pur essendo indipendenti dalle firme dei metodi. Ritornando all’esempio precedente, con Spring si potrà utilizzare un’istanza di *B* nella classe *A* senza però richiamare il suo costruttore, sarà il framework stesso a costruire un’istanza (il cosiddetto *Bean*) da “iniettare” in *A*. In questo modo, le modifiche non si propagano e non serve spendere tempo per fare refactoring.

Un altro esempio dello sviluppo agile permesso da Spring consiste nella possibilità di utilizzare dentro una classe non il nome di una classe specifica, ma un’interfaccia. Ad esempio, si considerino tre classi: *A*, *B* e *C*. Se *B* e *C* implementano la stessa interfaccia, al posto di richiamare una delle due classi, in *A* si potrà semplicemente utilizzare il nome dell’interfaccia. Nei file di configurazione, poi, si scriverà quale delle due classi utilizzare quando Java incontra il nome dell’interfaccia. L’utilità di una programmazione di questo tipo consiste nel poter imporre l’utilizzo della classe *B* o *C* cambiando semplicemente il riferimento nella configurazione. Questo è molto utile ai fini della portabilità. Ad esempio, *B* e *C* potrebbero essere due classi che definiscono la logica di connessione a un DB; quindi, con una semplice modifica nel file di configurazione, il programmatore potrà dire all’intero applicativo Java di utilizzare o il DB a cui si connette la class *B* o quello a cui si connette la classe *C*. Tutta la logica interna del progetto che usa/recupera dati

rimane invariata e non necessita di modifiche. Questa implementazione fornita da Spring viene chiamata *dependency injection*.

La *dependency injection*, inoltre, risulterà molto utile anche in fase di testing. Si ipotizzi che a dipendere dalla classe *B* non sia solo *A*, ma altre quattro classi; quindi, si avrebbero cinque classi che dipendono da *B*. Testare tutte le cinque classi in una programmazione tradizionale significherebbe creare un’istanza di *B* per ogni test. Dal momento che bisogna garantire una copertura totale del codice, non basterà un singolo test per classe. Se si dovessero fare N test, per ciascuna classe, Java creerebbe $N \cdot 5$ istanze di *B*. Utilizzando Spring, ed in particolare i *Bean*, invece si può definire lo scope come *singleton* e quindi riutilizzare lo stesso *Bean* per i vari riferimenti, evitando di riempire lo heap in memoria (non si ha certezza i quando il garbage collector farà pulizia).

È stato quindi deciso di procedere con l’utilizzo di Spring Boot, un’estensione di Spring che permette di evitare l’onerosa scrittura di file di configurazione in XML. Spring Boot fa molto più utilizzo delle annotazioni, che rende più immediata la comprensione di quale porzione di codice è influenzata dal framework e in che modo.

Di Spring Boot si è deciso di utilizzare i seguenti moduli:

- Spring Web
- Spring Data JPA
- Spring Security

Per permettere la conversione di dati da SQL a Java, e viceversa, si è scelto di usare il connettore Spring Data JPA, e non Spring Data JDBC.

Il vantaggio principale è che con JDBC è necessario memorizzare la query SQL che si vuole applicare sul DB in una stringa, mentre con JPA è sufficiente utilizzare dei metodi Java con nomi specifici che verranno tradotti automaticamente in query (ad esempio, *findItemById(String ID)* viene tradotto come *select * from items where items.id=ID*).

Un altro aspetto favorevole dell’utilizzo di JPA è che non tutti i DB relazionali usano la stessa versione SQL. Quindi, se si volesse cambiare DB vi sarebbe la necessità di rendere tutte le query coerenti con la sintassi del nuovo DB. Questo sarebbe un problema per la specifica di portabilità. Con JPA, invece, il problema della sintassi della query non si pone.

2 Iterazione 0

2.1 Casi d'uso

Partendo dai requisiti iniziali, abbiamo innanzitutto determinato gli attori che interagiscono all'interno del sistema e, per ciascuno di essi, abbiamo definito un elenco esaustivo di use case stories, suddivise per macroaree.

2.1.1 Attori

Gli attori individuati del nostro sistema sono di seguito elencati:

- **Cliente:** utenza principale, naviga all'interno dell'app alla ricerca di un alloggio da prenotare;
- **Proprietario:** utenza che mette a disposizione uno o più alloggi;
- **Admin:** gestore del sistema;
- **Sistema di geolocalizzazione:** API esterna utilizzata per la geolocalizzazione degli alloggi;
- **Sistema di pagamento:** API esterna utilizzata per la gestione dei pagamenti all'interno dell'app;
- **Servizio mail:** API esterna utilizzata per l'invio di mail agli utenti.

2.1.2 Use case stories: Cliente

Log-in/Log-out:

- Voglio poter effettuare log-in nell'applicazione
- Voglio poter effettuare log-out dall'applicazione

Gestione dati:

- o Voglio potermi registrare
- o Voglio poter aggiungere un metodo di pagamento
- o Voglio poter modificare un metodo di pagamento
- o Voglio poter rimuovere un metodo di pagamento
- o Voglio poter aggiornare il mio profilo
- o Voglio poter eliminare il mio profilo
- o Voglio poter diventare proprietario

Ricerca alloggi:

- o Voglio poter ricercare degli alloggi
- o Voglio poter visualizzare le informazioni di un alloggio
- o Voglio poter filtrare la ricerca di un alloggio per luogo
- o Voglio poter filtrare la ricerca di un alloggio per date di check-in e check-out
- o Voglio poter filtrare la ricerca di un alloggio per numero di posti letto
- o Voglio poter filtrare la ricerca di un alloggio per costo massimo per notte
- o Voglio poter filtrare la ricerca di un alloggio per distanza massima da un luogo
- o Voglio poter filtrare la ricerca di un alloggio per tags
- o Voglio poter filtrare la ricerca di un alloggio per valore medio delle recensioni
- o Voglio poter visualizzare le date di prenotazione disponibili per un alloggio
- o Voglio poter confrontare degli alloggi (al massimo 3)
- o Voglio poter definire un itinerario, impostando i luoghi da vedere (1 al giorno), le date e il costo massimo
- o Voglio poter definire un viaggio a sorpresa, impostando dei tag, la data ed escludendo delle mete già visitate (massimo 3)

Prenotazioni:

- o Voglio poter effettuare una prenotazione per un alloggio
- o Voglio poter effettuare un pagamento completo o a rate per una prenotazione
- o Voglio poter modificare una prenotazione
- o Voglio poter cancellare una prenotazione

Recensioni:

- o Voglio poter recensire un alloggio
- o Voglio poter vedere le mie recensioni effettuate
- o Voglio poter vedere le mie recensioni ricevute
- o Voglio poter segnalare una recensione
- o Voglio poter effettuare un reclamo

Comunicazione:

- o Voglio poter chattare con un proprietario
- o Voglio poter chattare con un admin

2.1.3 Use case stories: Proprietario

Log-in/Log-out:

- o Voglio poter effettuare log-in nell'applicazione
- o Voglio poter effettuare log-out dall'applicazione

Gestione dati:

- o Voglio poter aggiungere un alloggio
- o Voglio poter modificare un alloggio
- o Voglio poter eliminare un alloggio
- o Voglio poter aggiornare il mio profilo
- o Voglio poter eliminare il mio profilo

Prenotazioni:

- o Voglio poter accettare una prenotazione
- o Voglio poter rifiutare una prenotazione
- o Voglio poter disdire una prenotazione
- o Voglio poter confermare che un utente ha pernottato nel mio alloggio

Sponsorizzazione:

- o Voglio poter sponsorizzare i miei alloggi
- o Voglio poter effettuare un pagamento per la sponsorizzazione

Recensioni:

- o Voglio poter recensire un cliente
- o Voglio poter vedere le mie recensioni effettuate
- o Voglio poter vedere le recensioni ricevute dai miei alloggi
- o Voglio poter vedere le recensioni ricevute dagli utenti che hanno effettuato una prenotazione per un mio alloggio
- o Voglio poter segnalare una recensione
- o Voglio poter effettuare un reclamo

Comunicazione:

- o Voglio poter chattare con un cliente
- o Voglio poter chattare con un admin

2.1.4 Use case stories: Admin

Log-in/Log-out:

- o Voglio poter effettuare log-in nel sistema di gestione dell'applicazione
- o Voglio poter effettuare log-out dal sistema di gestione dell'applicazione

Gestione utenze:

- o Voglio poter sospendere un cliente per troppe recensioni negative o reclami
- o Voglio poter sospendere un alloggio per troppe recensioni negative o reclami
- o Voglio poter sospendere un proprietario per troppe recensioni negative o reclami

Recensioni:

- o Voglio poter visualizzare le recensioni di clienti e proprietari
- o Voglio poter rimuovere una recensione segnalata

Comunicazione:

- o Voglio poter chattare con un cliente
- o Voglio poter chattare con un proprietario

2.1.5 Use case stories: Sistema di geolocalizzazione

Ricerca alloggi:

- o Voglio poter fornire la distanza tra un luogo di origine e uno o più luoghi di destinazione

2.1.6 Use case stories: Sistema di pagamento

Prenotazioni:

- o Voglio poter autorizzare la transazione per il pagamento di una prenotazione
- o Voglio poter autorizzare la transazione per il pagamento di una sponsorizzazione

2.1.7 Use case stories: Servizio mail

Gestione dati:

- o Voglio poter inviare una mail di conferma registrazione
- o Voglio poter inviare una mail per il reset della password
- o Voglio poter inviare una mail di conferma cancellazione utenza
- o Voglio poter inviare una mail di conferma rimozione alloggio

Prenotazioni:

- o Voglio poter inviare una mail di conferma prenotazione
- o Voglio poter inviare una mail di conferma pagamento
- o Voglio poter inviare una mail di aggiornamento stato prenotazione

2.2 Requisiti

Successivamente, basandoci sulle use case stories generate, abbiamo ricavato i requisiti funzionali e non che il sistema dovrà soddisfare per garantire un corretto funzionamento.

2.2.1 Requisiti funzionali

Le funzionalità individuate sono state suddivise in tre fasce di priorità:

- Funzioni con priorità alta: sono le funzioni primarie, che consentono di realizzare gli scopi principali del sistema;
- Funzioni con priorità media: sono le funzioni con elevata importanza ma che riguardano operazioni secondarie;
- Funzioni con priorità bassa: sono le funzioni di contorno, non sono fondamentali per il funzionamento dell'applicazione ma rendono completo il suo utilizzo.

Di seguito vengono elencate, suddivise per macroaree e con i relativi codici e priorità assegnati.

Gestione utenze:

Codice	Nome funzione	Priorità
GU1	Registrazione utente	Alta
GU2	Invio mail di conferma registrazione	Alta
GU3	Modifica profilo utente	Alta
GU4	Cancellazione utenza	Media
GU5	Invio mail di conferma cancellazione	Bassa
GU6	Reset password utente	Bassa
GU7	Invio mail di conferma reset password	Bassa
GU8	Upgrade da cliente a proprietario	Alta

GU9	Log-in all'applicazione	Alta
GU10	Log-out dall'applicazione	Alta
GU11	Visualizzazione profilo utente	Bassa
GU12	Sospensione utente	Bassa
GU13	Inserimento dati nel database	Alta

Tabella 1: Requisiti Funzionali - Gestione utenze

Gestione alloggi:

Codice	Nome funzione	Priorità
GA1	Inserimento alloggio	Alta
GA2	Modifica alloggio	Media
GA3	Rimozione alloggio	Bassa
GA4	Visualizzazione scheda di un alloggio	Media
GA5	Visualizzazione calendario disponibilità di un alloggio	Alta
GA6	Sponsorizzazione di un alloggio	Bassa
GA7	Sospensione di un alloggio	Bassa
GA8	Inserimento dati nel database	Alta

Tabella 2: Requisiti Funzionali - Gestione alloggi

Ricerca alloggi:

Codice	Nome funzione	Priorità
RA1	Ricerca alloggi	Alta
RA2	Filtra per luogo	Alta

RA3	Filtra per disponibilità (date)	Alta
RA4	Filtra per numero di posti letto	Alta
RA5	Filtra per costo	Alta
RA6	Filtra per tag	Media
RA7	Filtra per distanza	Bassa
RA8	Filtra per recensione	Bassa

Tabella 3: Requisiti Funzionali - Ricerca alloggi

Gestione prenotazioni:

Codice	Nome funzione	Priorità
GP1	Inserimento prenotazione	Alta
GP2	Invio mail di avvenuta prenotazione	Media
GP3	Visualizzazione prenotazioni effettuate	Bassa
GP4	Modifica prenotazione	Media
GP5	Cancellazione prenotazione	Media
GP6	Invio mail di aggiornamento prenotazione	Media
GP7	Visualizzazione prenotazioni ricevute	Alta
GP8	Accettazione prenotazione	Alta

Tabella 4: Requisiti Funzionali - Gestione prenotazioni

Gestione pagamenti:

Codice	Nome funzione	Priorità
PG1	Aggiunta metodo di pagamento	Alta
PG2	Modifica metodo di pagamento	Media

PG3	Rimozione metodo di pagamento	Bassa
PG4	Esecuzione pagamento	Alta
PG5	Invio mail di avvenuto pagamento	Media
PG6	Visualizzazione pagamenti	Bassa
PG7	Richiesta rimborso	Bassa

Tabella 5: Requisiti Funzionali - Gestione pagamenti

Gestione recensioni e reclami:

Codice	Nome funzione	Priorità
GR1	Inserimento recensione di un utente	Bassa
GR2	Visualizzazione recensioni di un utente	Bassa
GR3	Inserimento recensione di un alloggio	Bassa
GR4	Visualizzazione recensioni di un alloggio	Bassa
GR5	Segnalazione di una recensione	Bassa
GR6	Inserimento di un reclamo per un utente	Bassa
GR7	Inserimento di un reclamo per un alloggio	Bassa
GR8	Visualizzazione reclami/segnalazioni	Bassa
GR9	Inserimento dati nel database	Bassa

Tabella 6: Requisiti Funzionali - Gestione recensioni e reclami

Gestione comunicazioni:

Codice	Nome funzione	Priorità
GC1	Invio messaggio ad un utente	Bassa
GC2	Visualizzazione chat con un utente	Bassa

GC3	Invio messaggio ad un admin	Bassa
GC4	Visualizzazione chat con un admin	Bassa

Tabella 7: Requisiti Funzionali - Gestione comunicazioni

2.2.2 Requisiti non funzionali

Data la natura e il contesto dell'applicazione da realizzare in questo progetto, sono stati tenuti in considerazione alcuni requisiti non funzionali particolarmente rilevanti:

- o Portabilità

L'applicazione deve poter essere utilizzata su una grande varietà di dispositivi, sia mobile che fissi, e con diversi sistemi operativi. Per questo motivo, è stato scelto di realizzare una web-app che permettesse di scalare gran parte del sistema, andando a modificare solo il livello di visualizzazione

- o Usabilità

Trattandosi di un sistema di booking, il pubblico a cui è rivolto il progetto è vasto ed eterogeneo, in particolar modo per quanto riguarda le fasce di età. Per questo motivo, l'applicazione dovrà risultare semplice da utilizzare, non dovrà richiedere l'utilizzo di nessun manuale e dovrà presentare solamente le informazioni necessarie.

2.3 Fully dressed descriptions

Per ogni use case identificato è stata redatta una descrizione completa (fully dressed description), in cui vengono specificati la descrizione, i requisiti coperti, gli attori coinvolti, le precondizioni e postcondizioni, il processo, gli scenari alternativi, e le possibili estensioni.

2.3.1 UC1: Registrazione utente

<i>Descrizione</i>	Un utente vuole registrarsi nel sistema di booking
<i>Requisiti coperti</i>	GU1, GU2, GU13, PG1, PG8
<i>Attori coinvolti</i>	Cliente
<i>Precondizioni</i>	L'utente non è presente nel database
<i>Postcondizioni</i>	L'utente è stato aggiunto al database
<i>Processo</i>	<ol style="list-style-type: none"> 1. L'utente clicca sul tasto “Sign up!”. 2. L'utente compila il form in cui sono richiesti i propri dati personali (nome, cognome, codice fiscale, e-mail, data di nascita, password). 3. L'utente preme “Register” per confermare la volontà di registrarsi con i dati inseriti nel form. <ol style="list-style-type: none"> a) L'utente riceve una e-mail con all'interno un link di attivazione dell'account. b) I dati dell'utente vengono inseriti all'interno del database. c) L'applicazione reindirizza sulla schermata di login.
<i>Alternative</i>	<ol style="list-style-type: none"> 1. <u>Inserimento dati errati</u> Al passo (2) sono inseriti dei dati errati (codice fiscale troppo corto, password non conforme alle policy di sicurezza dell'applicazione), in questo caso la registrazione dell'utenza è bloccata. Viene mostrato un pop-up di errore e viene inibito il tasto “Register”. 2. <u>Scadenza link di attivazione</u> L'utenza deve cliccare sul link di attivazione all'interno della e-mail a cui si fa riferimento nel passo (3.a) prima che esso

	<p>scada altrimenti il link non sarà più valido e dovrà ripetere la procedura da capo.</p> <p>3. <u>Utente già registrato</u></p> <p>Se l'utenza è già registrata appare un pop-up a schermo che riporta la scritta “User already registered”.</p>
--	--

2.3.2 UC2: Log-in nell'applicazione

<i>Descrizione</i>	Un utente vuole effettuare il log-in nell'applicazione
<i>Requisiti coperti</i>	GU9
<i>Attori coinvolti</i>	Cliente, Proprietario
<i>Precondizioni</i>	L'utente non è loggato nell'applicazione
<i>Postcondizioni</i>	L'utente è loggato nell'applicazione
<i>Processo</i>	<ol style="list-style-type: none"> 1. L'utente clicca sul tasto “Login” e viene reindirizzato alla schermata per inserire le credenziali. 2. L'utente compila i campi con le proprie credenziali (indirizzo mail e password). 3. L'utente clicca sul tasto “Login” per confermare la volontà di voler accedere con il proprio account 4. Viene visualizzato un pop-up con scritto “Login successfully completed” e si viene reindirizzati alla schermata precedente
<i>Alternative</i>	<ul style="list-style-type: none"> • <u>Inserimento dati errati</u> <p>Al passo (2) sono inseriti dei dati errati (mail non valida, credenziali errate). Viene mostrato un pop-up di errore con scritto “Wrong credentials”.</p>

2.3.3 UC3: Log-out dall'applicazione

<i>Descrizione</i>	Un utente vuole effettuare il log-out dall'applicazione
<i>Requisiti coperti</i>	GU10
<i>Attori coinvolti</i>	Cliente, Proprietario
<i>Precondizioni</i>	L'utente è loggato nell'applicazione
<i>Postcondizioni</i>	L'utente non è loggato nell'applicazione
<i>Processo</i>	<ol style="list-style-type: none">1. L'utente, loggato all'interno del proprio account sull'applicazione, entra nel menu a scomparsa in alto a sinistra e clicca su "Logout"2. È richiesta la conferma dell'azione: se l'azione è confermata, l'utente effettua il logout.3. Viene visualizzato un pop-up con scritto "Logout successfully completed" e si viene reindirizzati alla schermata Home

2.3.4 UC4: Modifica profilo utente

<i>Descrizione</i>	Un utente vuole effettuare una modifica al proprio profilo
<i>Requisiti coperti</i>	GU3, GU6, GU7, GU11, GU13, PG1, PG2, PG3, PG8
<i>Attori coinvolti</i>	Cliente, Proprietario
<i>Precondizioni</i>	Un utente è loggato nell'applicazione
<i>Postcondizioni</i>	Alcuni dati del profilo dell'utente sono stati modificati

<i>Processo</i>	<ol style="list-style-type: none"> 1. L'utente, loggato all'interno del proprio account sull'applicazione, entra nel menu a scomparsa in alto a sinistra e clicca su "My profile" 2. Entrato nella sezione "My profile", l'utente clicca sul tasto "Change profile" <ol style="list-style-type: none"> a) Viene mostrata la form per la modifica dei dati. b) L'utente modifica i dati. 3. L'utente clicca su "Confirm". <ol style="list-style-type: none"> a) È richiesta la conferma dell'azione: se l'azione è confermata, l'utente è modificato. b) I nuovi dati dell'utente vengono salvati all'interno del database.
<i>Alternative</i>	<ul style="list-style-type: none"> • <u>Inserimento dati errati</u> Al passo (2.b) sono inseriti dati errati, in questo caso la modifica è annullata e viene lanciato un messaggio di errore. Con "dati errati" si intende mail, password, data di nascita, ecc.
<i>Estensioni</i>	<ul style="list-style-type: none"> • <u>Conferma tramite e-mail</u> L'utente riceve una e-mail contenente la conferma di inserimento ed il riepilogo delle informazioni inserite.

2.3.5 UC5: Cancellazione utente

<i>Descrizione</i>	Un utente vuole cancellare il suo profilo dal sistema
<i>Requisiti coperti</i>	GU4, GU5, GA3, GP5, PG3
<i>Attori coinvolti</i>	Cliente, Proprietario

<i>Precondizioni</i>	Un utente è loggato nell'applicazione
<i>Postcondizioni</i>	Il profilo dell'utente viene rimosso dal database
<i>Processo</i>	<ol style="list-style-type: none"> 1. L'utente, loggato all'interno del proprio account sull'applicazione, entra nel menu a scomparsa in alto a sinistra e clicca su "My profile" 2. Entrato nella sezione "My profile", l'utente clicca sul tasto "Delete profile" 3. È richiesta la conferma dell'azione: se l'azione è confermata, l'utente viene eliminato dal database. <ol style="list-style-type: none"> a) Viene visualizzato un pop-up con scritto "Operation complete" e si viene reindirizzati alla schermata Home b) L'utente riceve una mail di conferma della cancellazione del profilo
<i>Alternative</i>	<ul style="list-style-type: none"> • <u>Utente con prenotazioni attive</u> Al passo (2) se l'utenza che richiede di essere cancellata ha delle prenotazioni attive viene mostrato un messaggio di errore in cui si invita a disdire le prenotazioni ancora presenti. • <u>Utente con alloggi attivi</u> Al passo (2) se l'utenza che richiede di essere cancellata ha degli alloggi per cui sono registrate delle prenotazioni attive viene mostrato un messaggio di errore in cui si invita a disdire le prenotazioni ancora presenti.

2.3.6 UC6: Inserimento alloggio

<i>Descrizione</i>	Un proprietario vuole inserire un alloggio
--------------------	--

<i>Requisiti coperti</i>	GA1, GA8
<i>Attori coinvolti</i>	Proprietario
<i>Precondizioni</i>	Un proprietario è loggato nell'applicazione
<i>Postcondizioni</i>	Un alloggio del proprietario viene inserito nel database
<i>Processo</i>	<ol style="list-style-type: none"> 1. L'utente, loggato all'interno del proprio account sull'applicazione, entra nel menu a scomparsa in alto a sinistra e clicca su "My apartments". 2. Entrato nella sezione "My apartments", l'utente clicca sul tasto "Insert new apartments". 3. L'utente compila il form in cui sono richiesti i dati dell'alloggio (nome, posizione, numero di posti letto, prezzo per notte, ecc). 4. L'utente preme "Register apartment" per confermare la volontà di registrare il nuovo alloggio con i dati inseriti nel form. <ul style="list-style-type: none"> a) L'utente viene reindirizzato alla pagina di visualizzazione del nuovo alloggio. b) I dati dell'alloggio vengono inseriti all'interno del database.
<i>Alternative</i>	<ul style="list-style-type: none"> • <u>Inserimento dati errati</u> Al passo (3) sono inseriti dati errati, in questo caso la modifica è annullata e viene lanciato un messaggio di errore.

2.3.7 UC7: Modifica alloggio

<i>Descrizione</i>	Un proprietario vuole modificare i dati di un suo alloggio
--------------------	--

<i>Requisiti coperti</i>	GA2, GA4, GA5, GA8
<i>Attori coinvolti</i>	Proprietario
<i>Precondizioni</i>	Un proprietario è loggato nell'applicazione
<i>Postcondizioni</i>	Alcuni dati di un alloggio del proprietario sono stati modificati
<i>Processo</i>	<ol style="list-style-type: none"> 1. L'utente, loggato all'interno del proprio account sull'applicazione, entra nel menu a scomparsa in alto a sinistra e clicca su "My apartments". 2. Entrato nella sezione "My apartments", l'utente clicca sulla scheda dell'appartamento che vuole modificare. 3. Aperta la scheda dell'appartamento desiderato, l'utente clicca sul tasto "Edit apartment". <ol style="list-style-type: none"> a) Viene mostrata la form per la modifica dei dati. b) L'utente modifica i dati. 4. L'utente clicca su "Confirm". <ol style="list-style-type: none"> a) È richiesta la conferma dell'azione: se l'azione è confermata, l'utente è modificato. b) I nuovi dati dell'utente vengono salvati all'interno del database.
<i>Alternative</i>	<ul style="list-style-type: none"> • <u>Inserimento dati errati</u> Al passo (3.b) sono inseriti dati errati, in questo caso la modifica è annullata e viene lanciato un messaggio di errore.

2.3.8 UC8: Rimozione alloggio

<i>Descrizione</i>	Un proprietario vuole rimuovere un suo alloggio
<i>Requisiti coperti</i>	GA3, GA4

<i>Attori coinvolti</i>	Proprietario
<i>Precondizioni</i>	Un proprietario è loggato nell'applicazione
<i>Postcondizioni</i>	Un alloggio del proprietario è stato rimosso dal database
<i>Processo</i>	<ol style="list-style-type: none"> 1. L'utente, loggato all'interno del proprio account sull'applicazione, entra nel menu a scomparsa in alto a sinistra e clicca su "My apartments". 2. Entrato nella sezione "My apartments", l'utente clicca sulla scheda dell'appartamento che vuole eliminare. 3. Aperta la scheda dell'appartamento desiderato, l'utente clicca sul tasto "Delete apartment". 4. È richiesta la conferma dell'azione: se l'azione è confermata, l'alloggio viene eliminato dal database. 5. Viene visualizzato un pop-up con scritto "Operation complete" e si viene reindirizzati alla lista degli alloggi dell'utente.
<i>Alternative</i>	<ul style="list-style-type: none"> • <u>Alloggio con prenotazioni attive</u> Al passo (3), se per l'alloggio sono registrate delle prenotazioni attive, viene mostrato un messaggio di errore in cui si invita a disdire le prenotazioni ancora presenti.
<i>Estensioni</i>	<ul style="list-style-type: none"> • <u>Conferma tramite e-mail</u> L'utente riceve una e-mail contenente la conferma di rimozione dell'alloggio.

2.3.9 UC9: Sponsorizzazione alloggio

<i>Descrizione</i>	Un proprietario vuole sponsorizzare un proprio alloggio
<i>Requisiti coperti</i>	GA4, GA6, PG4, PG5

<i>Attori coinvolti</i>	Proprietario, Sistema di pagamento
<i>Precondizioni</i>	Un proprietario è loggato nell'applicazione
<i>Postcondizioni</i>	Un alloggio del proprietario viene sponsorizzato
<i>Processo</i>	<ol style="list-style-type: none"> 1. L'utente, loggato all'interno del proprio account sull'applicazione, entra nel menu a scomparsa in alto a sinistra e clicca su "My apartments". 2. Entrato nella sezione "My apartments", l'utente clicca sulla scheda dell'appartamento che vuole modificare. 3. Aperta la scheda dell'appartamento desiderato, l'utente clicca sul tasto "Sponsor your apartment!". <ol style="list-style-type: none"> a) Vengono mostrate i vari livelli di sponsorizzazione e i relativi costi. b) L'utente seleziona il livello. c) L'utente seleziona il metodo di pagamento per la pubblicità. 4. L'utente clicca su "Confirm". <ol style="list-style-type: none"> a) È richiesta la conferma dell'azione: se l'azione è confermata, viene applicata la sponsorizzazione.
<i>Alternative</i>	<ul style="list-style-type: none"> • <u>Metodo di pagamento non valido</u> Al passo (3.c) il metodo di pagamento non è valido o la transazione non è andata a buon fine.

2.3.10 UC10: Visualizzazione profilo utente

<i>Descrizione</i>	Si vuole visualizzare il profilo di un utente
<i>Requisiti coperti</i>	GU11, GP3, GP7, PG6, GR2
<i>Attori coinvolti</i>	Cliente, Proprietario

<i>Precondizioni</i>	Sono presenti vari utenti all'interno del database
<i>Postcondizioni</i>	Viene visualizzato il profilo di un utente
<i>Processo</i>	<p>1. L'utente entra nel menu a scomparsa in alto a sinistra e clicca su "My profile" oppure clicca sull'immagine del profilo di un altro utente (per esempio in una recensione o scheda di un alloggio).</p> <p>2. Viene visualizzato il profilo dell'utente selezionato, con i dettagli principali, le recensioni registrate per quell'utente ed eventuali alloggi pubblicati sull'applicazione.</p> <ul style="list-style-type: none"> a) Se il profilo visualizzato appartiene all'utente stesso, allora vengono visualizzati anche i metodi di pagamento, la lista delle prenotazioni effettuate e delle eventuali prenotazioni ricevute. b) Se l'utente clicca sulla scheda di un alloggio appartenente all'utente visualizzato, si procede come da UC11.

2.3.11 UC11: Visualizzazione scheda alloggio

<i>Descrizione</i>	Si vuole visualizzare la scheda di un alloggio
<i>Requisiti coperti</i>	GA4, GA5, GR4
<i>Attori coinvolti</i>	Cliente, Proprietario
<i>Precondizioni</i>	Vari alloggi sono presenti nel database
<i>Postcondizioni</i>	Viene visualizzata la scheda di un alloggio
<i>Processo</i>	<p>1. L'utente clicca sulla scheda di un alloggio (per esempio in una ricerca o nella scheda "My apartments").</p>

	<p>2. Viene visualizzato il profilo dell'alloggio selezionato, con i dettagli principali, le recensioni registrate per quell'alloggio ed il calendario di disponibilità per eventuali prenotazioni.</p>
--	---

2.3.12 UC12: Ricerca alloggi

<i>Descrizione</i>	Un cliente vuole ricercare un alloggio
<i>Requisiti coperti</i>	RA1-RA9
<i>Attori coinvolti</i>	Cliente, Sistema di geolocalizzazione
<i>Precondizioni</i>	Un cliente è loggato nell'applicazione
<i>Postcondizioni</i>	Viene visualizzato un elenco di alloggi
<i>Processo</i>	<ol style="list-style-type: none"> 1. L'utente clicca sulla barra di ricerca per gli alloggi. 2. L'utente inserisce la località dove vuole ricercare l'alloggio. <ol style="list-style-type: none"> a) L'utente può impostare dei filtri come: distanza, prezzo, tag, ecc. b) L'utente inserisce le date in cui vuole prenotare l'alloggio. c) L'utente può scegliere se visualizzare solo alloggi singoli o anche gruppi di alloggi. 3. L'utente clicca il simbolo della lente di ingrandimento per far partire la ricerca degli alloggi che rispetta i filtri selezionati al punto (2). 4. Il sistema restituisce la lista degli alloggi.
<i>Alternative</i>	<ul style="list-style-type: none"> • <u>Non ci sono risultati per la ricerca corrente</u> Al passo (3) la ricerca non restituisce risultati per i filtri impostati.

2.3.13 UC13: Inserimento prenotazione

<i>Descrizione</i>	Un cliente vuole effettuare una prenotazione
<i>Requisiti coperti</i>	GA4, GA5, GP1, GP2, GP6, GP12, PG4, PG5
<i>Attori coinvolti</i>	Cliente, Sistema di pagamento
<i>Precondizioni</i>	Un cliente è loggato nell'applicazione
<i>Postcondizioni</i>	La prenotazione del cliente è stata effettuata
<i>Processo</i>	<ol style="list-style-type: none"> 1. L'utente entra all'interno della scheda di un alloggio. 2. L'utente clicca sul tasto “Book this apartment”. 3. L'utente seleziona il metodo di pagamento per la prenotazione. 4. L'utente clicca su “Confirm”. <ol style="list-style-type: none"> a) È richiesta la conferma dell'azione: se l'azione è confermata, viene inserita la prenotazione. 5. La richiesta di prenotazione viene inoltrata al proprietario dell'alloggio.
<i>Alternative</i>	<ul style="list-style-type: none"> • <u>Metodo di pagamento non valido</u> Al passo (3.c) il metodo di pagamento non è valido o la transazione non è andata a buon fine.
<i>Estensioni</i>	<ul style="list-style-type: none"> • <u>Conferma tramite e-mail</u> L'utente riceve una e-mail contenente la conferma di dell'avvenuta prenotazione e il riepilogo dei dati della prenotazione.

2.3.14 UC14: Modifica prenotazione

<i>Descrizione</i>	Un cliente vuole modificare una sua prenotazione
<i>Requisiti coperti</i>	GP3, GP4, GP5, GP6, GP12, PG4, PG5, PG7
<i>Attori coinvolti</i>	Cliente
<i>Precondizioni</i>	Un cliente è loggato nell'applicazione
<i>Postcondizioni</i>	Una prenotazione del cliente è stata modificata
<i>Processo</i>	<ol style="list-style-type: none"> 1. L'utente entra nel menu a scomparsa in alto a sinistra e clicca su "My reservations". 2. Entrato nella sezione "My reservations", l'utente clicca sulla scheda della prenotazione che vuole modificare. 3. Aperta la scheda dell'appartamento desiderato, l'utente clicca sul tasto "Edit reservation". <ol style="list-style-type: none"> a) Viene mostrata la form per la modifica dei dati. b) L'utente modifica i dati oppure clicca su "Delete reservation". 4. L'utente clicca su "Confirm". <ol style="list-style-type: none"> a) È richiesta la conferma dell'azione: se l'azione è confermata, la prenotazione è modificata. b) I nuovi dati della prenotazione vengono salvati all'interno del database. c) Se vi è un supplemento di costo, si effettua il pagamento scegliendo il metodo di pagamento. d) Se vi è una diminuzione di costo, viene inviata una richiesta di rimborso parziale al proprietario dell'alloggio.
<i>Alternative</i>	<ul style="list-style-type: none"> • <u>Inserimento dati errati</u>

	<p>Al passo (3.b) sono inseriti dati errati, per dati errati si intende una data non disponibile per la prenotazione. In questo caso la modifica è annullata e viene lanciato un messaggio di errore.</p>
<i>Estensioni</i>	<ul style="list-style-type: none"> • <u>Conferma tramite e-mail</u> L’utente riceve una e-mail contenente la conferma di dell’avvenuta modifica della prenotazione e il riepilogo dei dati della prenotazione.

2.3.15 UC15: Accettazione/rifiuto prenotazione

<i>Descrizione</i>	Un proprietario deve accettare o rifiutare la prenotazione di un cliente
<i>Requisiti coperti</i>	GP7, GP8, GP9, GP12
<i>Attori coinvolti</i>	Proprietario
<i>Precondizioni</i>	Un proprietario è loggato nell’applicazione, il proprietario riceve una notifica per la ricezione di una richiesta di prenotazione per un suo appartamento
<i>Postcondizioni</i>	Il proprietario ha accettato o rifiutato una prenotazione
<i>Processo</i>	<ol style="list-style-type: none"> 1. Il proprietario entra nel menu a scomparsa in alto a sinistra e clicca su “Booking requests”. 2. Il proprietario clicca sulla richiesta di prenotazione ricevuta e decide se approvarla o meno. 3. Viene chiesto al proprietario di confermare la decisione. <ol style="list-style-type: none"> a) Se il proprietario conferma la prenotazione, i dati vengono registrati all’interno del database e l’utente

	<p>che ha inoltrato la richiesta viene avvisato che la sua prenotazione è stata accettata.</p> <p>b) Se il proprietario rifiuta la prenotazione, l'utente che ha inoltrato la richiesta viene avvisato che la sua prenotazione è stata declinata.</p>
--	---

2.3.16 UC16: Disdetta prenotazione

<i>Descrizione</i>	Un proprietario deve disdire la prenotazione di un cliente
<i>Requisiti coperti</i>	GP7, GP10, GP12
<i>Attori coinvolti</i>	Proprietario
<i>Precondizioni</i>	Un proprietario è loggato nell'applicazione e ha almeno una prenotazione effettuata per uno dei suoi alloggi
<i>Postcondizioni</i>	Il proprietario ha disdetto una prenotazione per un suo alloggio
<i>Processo</i>	<ol style="list-style-type: none"> 1. Il proprietario entra nel menu a scomparsa in alto a sinistra e clicca su "Booking requests". 2. Il proprietario clicca sulla prenotazione che intende disdire e poi clicca su "Cancel reservation". 3. Viene chiesto al proprietario di confermare la decisione. <ol style="list-style-type: none"> a) Se il proprietario conferma la disdetta, il cliente che aveva effettuato la prenotazione viene avvisato che la sua prenotazione è stata cancellata e viene emesso il rimborso.

2.3.17 UC17: Conferma pernottamento

<i>Descrizione</i>	Un proprietario deve confermare l'avvenuto pernottamento di un cliente in un suo alloggio
<i>Requisiti coperti</i>	GP7, GP11, GP12
<i>Attori coinvolti</i>	Proprietario
<i>Precondizioni</i>	Un proprietario è loggato nell'applicazione e ha almeno una prenotazione effettuata per uno dei suoi alloggi
<i>Postcondizioni</i>	Il proprietario ha confermato il pernottamento di un cliente in un suo alloggio
<i>Processo</i>	<ol style="list-style-type: none"> 1) Il proprietario entra nel menu a scomparsa in alto a sinistra e clicca su “Booking requests”. 2) Il proprietario clicca sulla prenotazione per cui deve confermare il pernottamento e clicca su “confirm accomodation”. 3) Viene chiesto al proprietario di confermare la decisione. <ol style="list-style-type: none"> a) Se il proprietario conferma il pernottamento, sia il proprietario che il cliente sono autorizzati ad inserire una recensione.

2.3.18 UC18: Inserimento recensione alloggio

<i>Descrizione</i>	Un cliente vuole effettuare una recensione su un alloggio
<i>Requisiti coperti</i>	GA4, GR3, GR9
<i>Attori coinvolti</i>	Cliente

<i>Precondizioni</i>	Un utente è loggato nell'applicazione, l'utente deve aver alloggiato nell'appartamento che vuole recensire
<i>Postcondizioni</i>	Una recensione viene inserita nel database
<i>Processo</i>	<ol style="list-style-type: none"> 1. L'utente entra nel menu a scomparsa in alto a sinistra e clicca su "My reservations". 2. Entrato nella sezione "My reservations", l'utente clicca sulla scheda della prenotazione che vuole recensire. 3. Aperta la scheda dell'alloggio, l'utente clicca sul tasto "Review the apartment". 4. L'utente compila la form della recensione con i dati richiesti (descrizione, punteggio generale, punteggio per pulizia, comodità, ecc). 5. L'utente preme il tasto "Insert review" per confermare la volontà di voler inserire la recensione.

2.3.19 UC19: Inserimento recensione utente

<i>Descrizione</i>	Un proprietario vuole effettuare una recensione su un cliente
<i>Requisiti coperti</i>	GU11, GR1, GR9
<i>Attori coinvolti</i>	Proprietario
<i>Precondizioni</i>	Un proprietario è loggato nell'applicazione, il proprietario deve aver fornito in passato uno dei suoi alloggi all'utente che vuole recensire.
<i>Postcondizioni</i>	Una recensione viene inserita nel database
<i>Processo</i>	<ol style="list-style-type: none"> 1. Il proprietario entra nel menu a scomparsa in alto a sinistra e clicca su "Reservations".

	<ol style="list-style-type: none"> 2. Entrato nella sezione “Reservations”, l’utente clicca sulla scheda della prenotazione che vuole recensire. 3. Entrato nella pagina dell’utente, l’utente clicca sul tasto “Review the customer”. 4. L’utente compila la form della recensione con i dati richiesti (descrizione, punteggio generale). 5. L’utente preme il tasto “Insert review” per confermare la volontà di voler inserire la recensione.
--	---

2.3.20 UC20: Inserimento reclami

<i>Descrizione</i>	Un utente vuole effettuare un reclamo
<i>Requisiti coperti</i>	GU11, GA4, GR5, GR6, GR7, GR9
<i>Attori coinvolti</i>	Cliente, Proprietario
<i>Precondizioni</i>	Un utente è loggato nell’applicazione
<i>Postcondizioni</i>	Un reclamo viene inviato
<i>Processo</i>	<ol style="list-style-type: none"> 1. L’utente entra nel menu a scomparsa in alto a sinistra e clicca su “Insert claim”. 2. L’utente compila la form del reclamo inserendo i dati richiesti (titolo, descrizione, utente/alloggio per il quale si inoltra il reclamo). 3. L’utente preme il tasto “Insert claim” per confermare la volontà di voler inserire la recensione. 4. Il reclamo viene inoltrato agli admin dell’applicazione per poter essere revisionato. 5. I dati del reclamo vengono inseriti all’interno del database.

2.3.21 UC21: Gestione reclami

<i>Descrizione</i>	Un admin riceve e gestisce un reclamo
<i>Requisiti coperti</i>	GU11, GA4, GR8, GR9
<i>Attori coinvolti</i>	Admin
<i>Precondizioni</i>	Un admin è loggato nell'applicazione, l'admin ha ricevuto un reclamo
<i>Postcondizioni</i>	Un reclamo viene accettato o eliminato
<i>Processo</i>	<ol style="list-style-type: none">1. L'admin entra nel menu a scomparsa in alto a sinistra e clicca su "View claims".2. L'admin clicca sulla scheda del reclamo che deve gestire.3. L'admin analizza il reclamo.<ol style="list-style-type: none">a) Se il reclamo è accettato, l'admin clicca sul tasto "Accept claim" e viene inoltrata una notifica di accettazione all'utente che ha inserito il reclamo e all'utente/proprietario dell'appartamento a cui è stato fatto il reclamo.b) Se il reclamo è rifiutato, l'admin clicca sul tasto "Deny claim" viene inoltrata una notifica di negazione all'utente che ha inserito il reclamo.4. I dati del reclamo vengono aggiornati all'interno del database.

2.3.22 UC22: Sospensione utente o alloggio

<i>Descrizione</i>	Un admin sospende un utente o un alloggio dopo aver ricevuto un reclamo
--------------------	---

<i>Requisiti coperti</i>	GU11, GU12, GA4, GA7, GR2, GR4, GR8
<i>Attori coinvolti</i>	Admin
<i>Precondizioni</i>	Un admin è loggato nell'applicazione, un utente/alloggio ha superato il numero massimo di reclami che può ricevere.
<i>Postcondizioni</i>	Viene sospeso un utente o un alloggio
<i>Processo</i>	<ol style="list-style-type: none"> 1. L'admin ricerca l'utente/alloggio da sospendere. 2. L'admin accede alla scheda dell'utente/alloggio. 3. L'admin clicca sul tasto “Suspend”. 4. L'utente o il proprietario dell'alloggio riceve una notifica via mail all'interno della quale è comunicato che la sua utenza/il suo alloggio sono stati temporaneamente sospesi.

2.3.23 UC23: Gestione chat

<i>Descrizione</i>	Un utente vuole comunicare con un altro utente o con un admin
<i>Requisiti coperti</i>	GU11, GC1-GC4
<i>Attori coinvolti</i>	Cliente, Proprietario, Admin
<i>Precondizioni</i>	Un utente è loggato nell'applicazione
<i>Postcondizioni</i>	Viene inviato un messaggio, viene visualizzata una chat
<i>Processo</i>	<ol style="list-style-type: none"> 1. L'utente entra nella sezione della chat all'interno dell'applicazione, cliccando sul simbolo a forma di “baloon”. 2. All'interno della sezione è presente la lista degli utenti con cui ha avuto una conversazione. 3. Cliccando su uno di essi l'utente può visualizzare la cronologia della chat ed inviare un messaggio.

- | | |
|--|---|
| | <ol style="list-style-type: none">4. Cliccando sull'editor di testo visualizzato nella parte bassa dello schermo, l'utente può scrivere un messaggio.5. Cliccando sul tasto di invio, l'utente invia il messaggio alla persona desiderata. |
|--|---|

2.4 Use case diagram

In *Figura 1* viene riportato lo use case diagram.

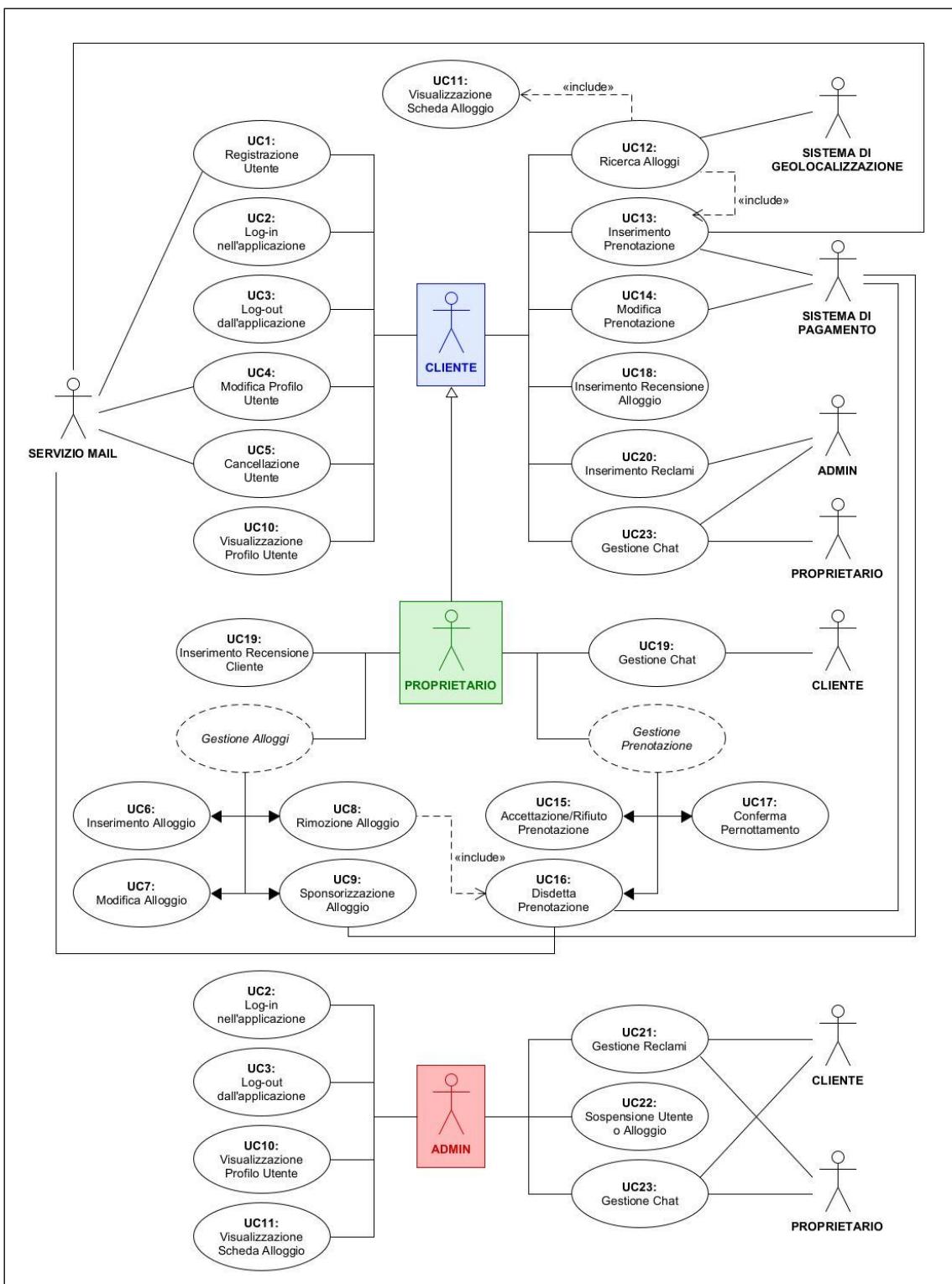


Figura 1: Use Case diagram

2.5 Architettura

2.5.1 Architettura hardware

In *Figura 2* viene rappresentato l'architettura finale prevista per la realizzazione del sistema.

Essa è composta da:

- Un server centrale, che interagisce con tutti gli altri componenti e svolge il ruolo di intermediario;
- Un server esterno su cui risiede il database;
- Tre dispositivi client:
 - Lo smartphone dei clienti;
 - Lo smartphone dei proprietari;
 - Il computer dell'admin.
- Tre server esterni a cui vengono richiesti dei servizi:
 - Il server del sistema di pagamento;
 - Il server del sistema di geolocalizzazione;
 - Il server del servizio di mail.

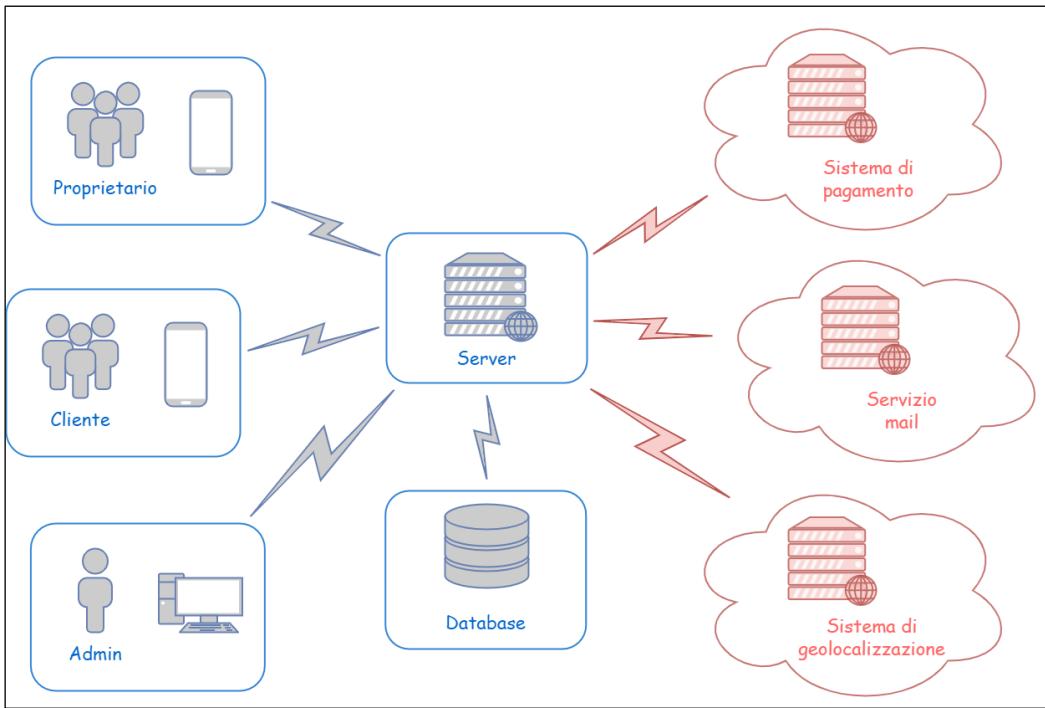


Figura 2: Architettura del sistema

2.5.2 Architettura software

Per quanto riguarda l'implementazione software, in *Figura 3* si riporta il deployment diagram.

I client relativi a clienti e proprietari vengono realizzati come applicazione Android mentre il client relativo all'admin come software Java. Entrambi i componenti interagiscono con il server centrale tramite REST API.

Il DBMS remoto utilizzato è PostgreSQL ed è localizzato sulla piattaforma cloud Heroku.

Infine, i tre server esterni utilizzati per i servizi di pagamento, di geolocalizzazione e di mail sono rispettivamente PayPal, Google Maps e Gmail.

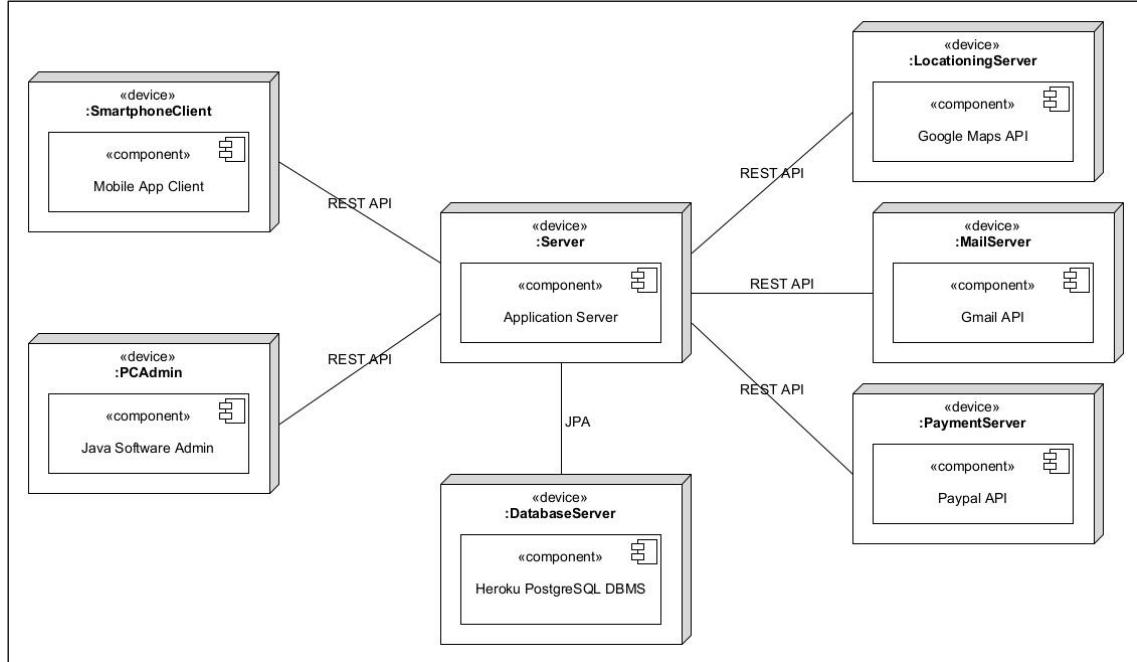


Figura 3: Deployment diagram

2.5.3 Pattern architetturale

Per rispettare le specifiche non funzionali richieste dal sistema è stato inizialmente valutato di realizzare un’architettura *Three-tier* basandoci sul pattern architetturale MVC (Model View Controller). Questo avrebbe consentito di suddividere il sistema in tre moduli da distribuire su tre macchine distinte:

- View: interfaccia utente, lato client
- Controller: logica funzionale, lato server
- Model: gestione dei dati persistenti, lato server (DataBase)

In questo modo sarebbe stato possibile “allocare” la parte di visualizzazione al client (smartphone, portatile, ecc) e procedere alla realizzazione della web-app.

Successivamente, però, si è scelto di applicare un differente pattern architetturale, il MVP (Model View Presenter). Il MVP è una derivazione del pattern Model-View-Controller che si distingue per l’eliminazione del collegamento diretto tra la componente di View e quella di Model (*Figura 4*). In questo modo viene forzata l’interazione tra Model e View sempre e solo tramite la componente Presenter che svolge il ruolo di “man-in-the-middle”.

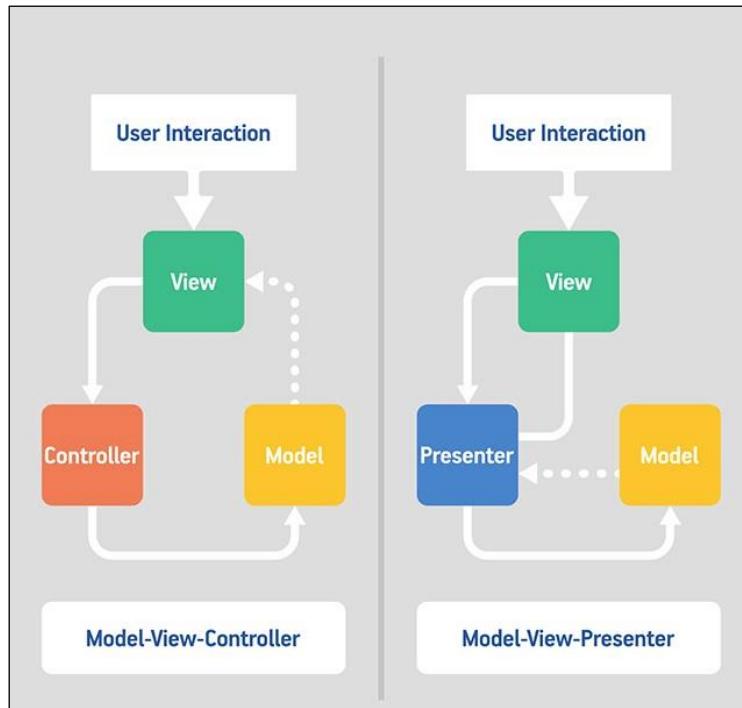


Figura 4: MVC vs MVP

Di seguito si elencano in dettaglio i ruoli svolti dai tre layer:

1. **Model:** è lo strato di Data Access per la gestione dei dati. Può essere visto come una interfaccia che è responsabile di accedere alle API connesse con un database locale oppure una cache;
2. **View:** lavora con il Presenter per mostrare i dati e notificare le azioni dell'utente, solitamente non ha nessuna logica applicativa, ma solo visuale. Nel caso di applicazioni multiplattaforma (Web, Mobile, Desktop), possiamo avere una singola interfaccia per la view e multiple implementazioni;
3. **Presenter:** risponde alle richieste della View, recuperando i dati dal Model, elaborandoli e restituendoli alla View.

I principali vantaggi dell'azione di un pattern architettonale MVP sono:

- o È possibile implementare lo strato di presentazione in modo completamente autonomo: ad esempio, potrebbe essere necessario avere un'implementazione completamente diversa per i dispositivi mobile rispetto alla versione desktop classica.

- o Permette di isolare le logiche di interazione, facilitandone la manutenzione e la leggibilità.
- o Le logiche di presentazione sono facilmente testabili ed è più semplice scrivere Unit-tests.
- o L'aggiornamento dei layer Model e View avviene contemporaneamente.

3 Iterazione 1

Dopo aver identificato gli attori del nostro sistema e le loro interazioni, le componenti hardware e la loro distribuzione nel contesto, nonché le tecnologie e i pattern da utilizzare, si procede con la specifica dei singoli componenti e delle interfacce di comunicazione tra di essi.

3.1 Component diagram

Per prima cosa, sono state individuate le macro componenti astratte presenti all'interno del sistema, e sono state modellate secondo il pattern Model View Presenter scelto.

Queste macro componenti sono:

- **Gestione utenze:** comprende la registrazione, il log-in e il log-out, la visualizzazione, la modifica e la cancellazione del profilo;
- **Gestione alloggi:** comprende l'inserimento, la modifica e la rimozione di un alloggio, la visualizzazione delle schede, la sponsorizzazione di un alloggio e la ricerca con l'applicazione di filtri;
- **Gestione prenotazioni:** comprende l'inserimento, la modifica, l'accettazione, il rifiuto e la disdetta di una prenotazione, i relativi pagamenti e la conferma del pernottamento;
- **Gestione recensioni, reclami e comunicazione:** comprende l'inserimento e la visualizzazione di recensioni e reclami, l'invio e lettura di messaggi nella chat tra utenti e la sospensione di utenti o alloggi (solo per l'admin).

In *Figura 5* si può osservare la suddivisione delle quattro componenti individuate nei tre sottosistemi “*EasyBookingView*”, “*EasyBookingPresenter*” e “*EasyBookingModel*”.

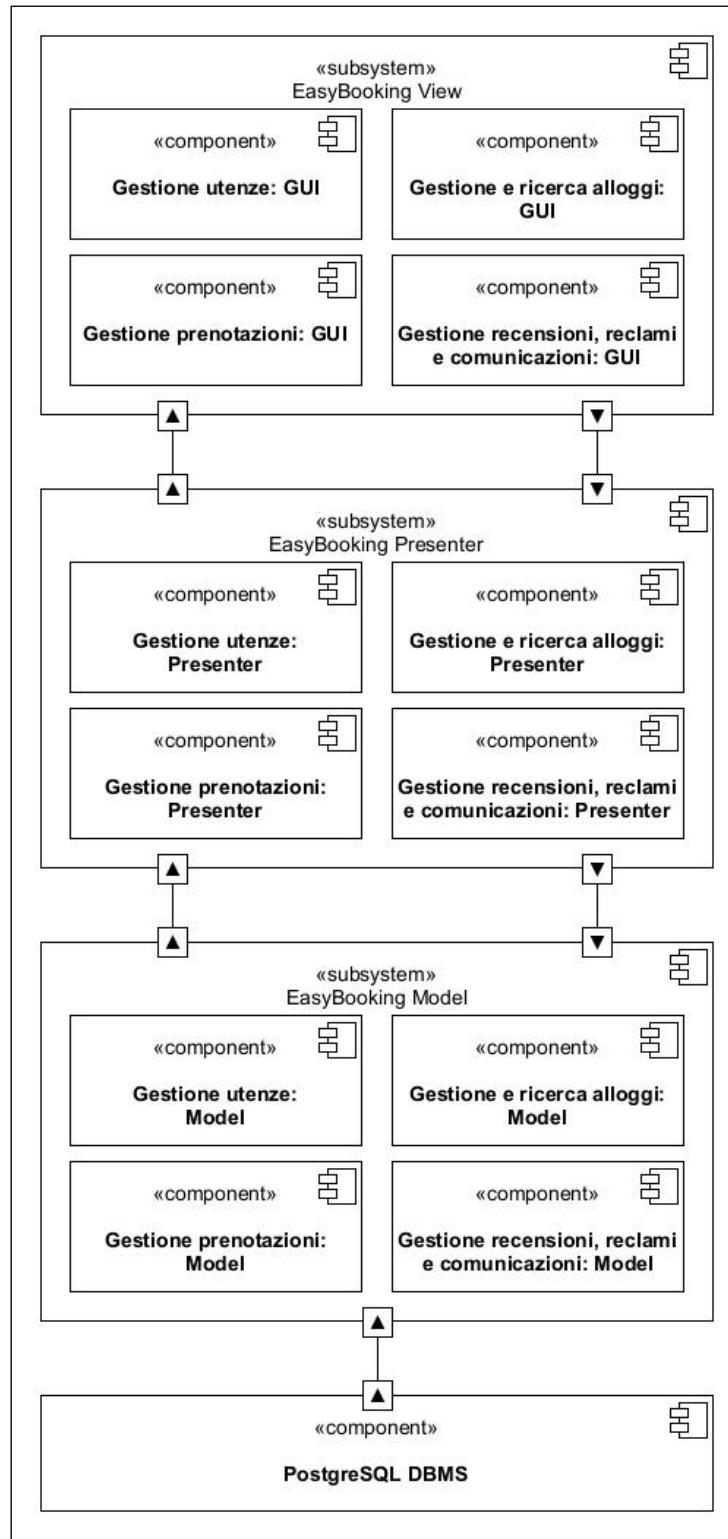


Figura 5: Component diagram MVP

Successivamente sono state posizionate le varie sottocomponenti nei diversi dispositivi che vengono utilizzati nel sistema (*Figura 6*).

Sul server centrale sono presenti tutte le componenti Presenter e Model, mentre sui client sono allocate le componenti View.

Inoltre, sono state definite ad alto livello le interfacce esposte dal server centrale per l'interazione con i client, nonché le interfacce fornite dai provider di servizi esterni.

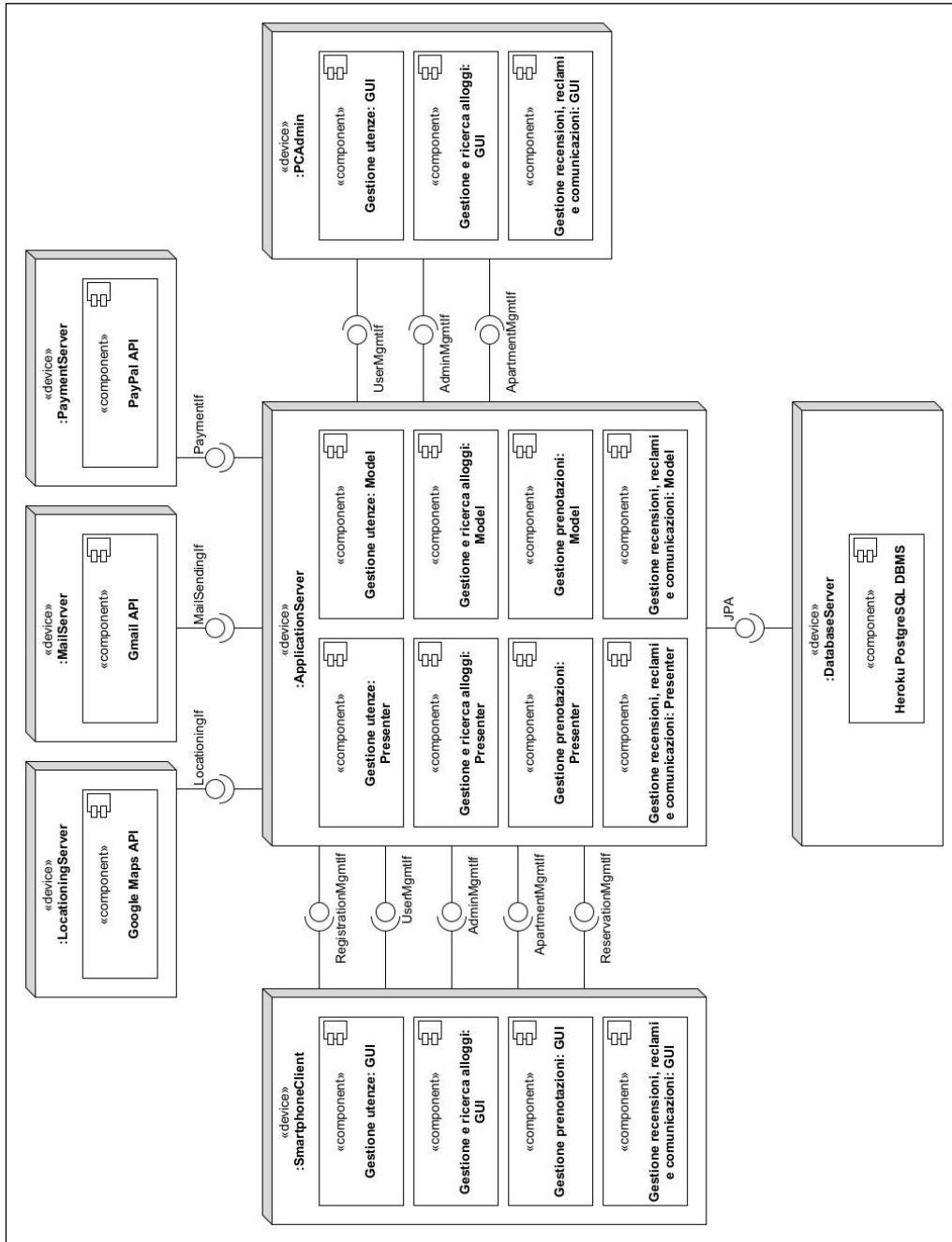


Figura 6: Component diagram black-box

3.2 Interface diagram

In Figura 7 sono state raggruppate le interfacce esposte all'esterno dai vari server.

Le interfacce fornite dal server interno al front-end sono suddivise per scopo:

- **RegistrationMgmtIf**: contiene i metodi per la registrazione di un utente;
- **UserMgmtIf**: contiene i metodi per la gestione dei profili utenti;
- **AdminMgmtIf**: contiene i metodi per la gestione dei profili admin;
- **ApartmentMgmtIf**: contiene i metodi per la gestione e ricerca degli appartamenti;
- **ReservationMgmtIf**: contiene i metodi per l'inserimento e la gestione delle prenotazioni.

In questa fase, le interfacce e i relativi metodi sono stati definiti in maniera astratta, omettendo segnature e tipi restituiti. Queste verranno poi raffinate nelle iterazioni successive, quando verranno effettivamente implementate.

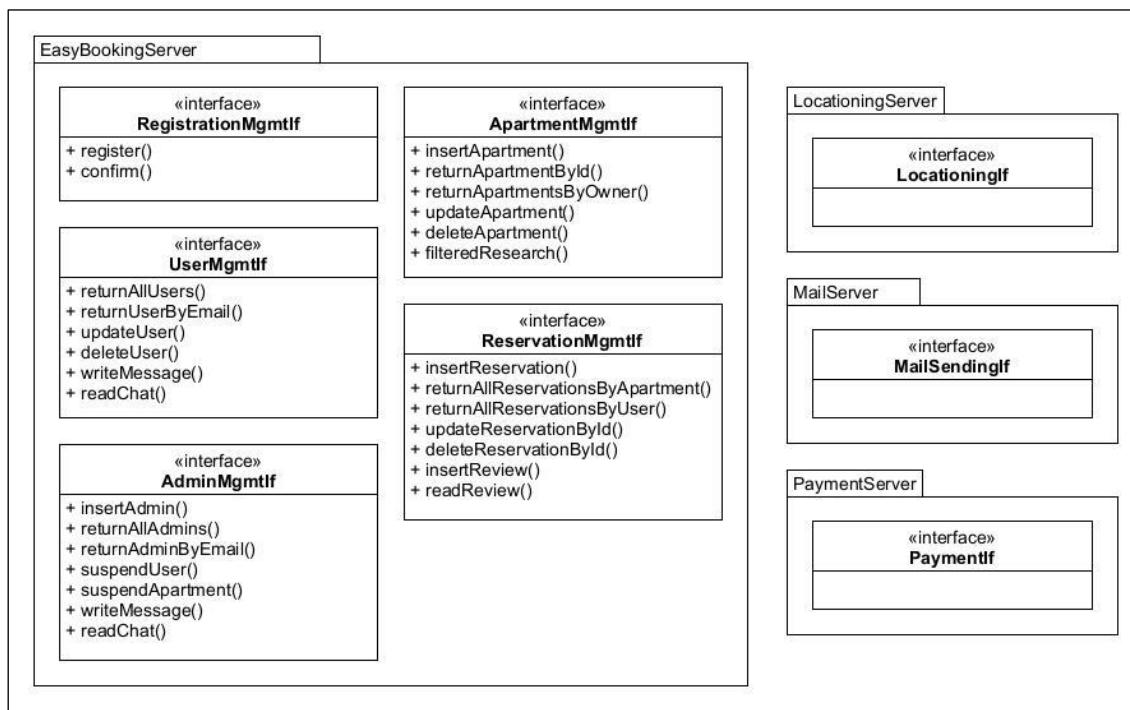


Figura 7: Interface diagram

4 Iterazione 2

4.1 Selezione delle funzioni da implementare

Per l'iterazione 2 del processo di sviluppo dell'applicazione si è scelto di implementare le funzionalità chiave per l'accesso dell'utente all'applicazione, ovvero le funzionalità descritte all'interno dei seguenti use case:

- UC1: Registrazione utente
- UC2: Log-in nell'applicazione
- UC3: Log-out dall'applicazione

4.1.1 Use Case descriptions

UC1: Registrazione utente

<i>Descrizione</i>	Un utente vuole registrarsi nel sistema di booking.
<i>Requisiti coperti</i>	GU1, GU2, GU13, PG1, PG8
<i>Attori coinvolti</i>	Cliente
<i>Precondizioni</i>	L'utente non è presente nel database
<i>Postcondizioni</i>	L'utente è stato aggiunto al database
<i>Processo</i>	<ol style="list-style-type: none">1. L'utente clicca sul tasto “Sign up!”.2. L'utente compila il form in cui sono richiesti i propri dati personali (nome, cognome, codice fiscale, e-mail, data di nascita, password).

	<p>3. L’utente preme “Register” per confermare la volontà di registrarsi con i dati inseriti nel form.</p> <ul style="list-style-type: none"> a) L’utente riceve una e-mail con all’interno un link di attivazione dell’account. b) I dati dell’utente vengono inseriti all’interno del database. c) L’applicazione reindirizza sulla schermata di login.
<i>Alternative</i>	<ul style="list-style-type: none"> • <u>Inserimento dati errati</u> Al passo (2) sono inseriti dei dati errati (codice fiscale troppo corto, password non conforme alle policy di sicurezza dell’applicazione), in questo caso la registrazione dell’utenza è bloccata. Viene mostrato un pop-up di errore e viene inibito il tasto “Register”. • <u>Scadenza link di attivazione</u> L’utenza deve cliccare sul link di attivazione all’interno della e-mail a cui si fa riferimento nel passo (3.a) prima che esso scada altrimenti il link non sarà più valido e dovrà ripetere la procedura da capo. • <u>Utente già registrato</u> Se l’utenza è già registrata appare un pop-up a schermo che riporta la scritta “User already registered”.

UC2: Log-in nell’applicazione

<i>Descrizione</i>	Un utente vuole effettuare il log-in nell’applicazione
<i>Requisiti coperti</i>	GU9

<i>Attori coinvolti</i>	Cliente, Proprietario
<i>Precondizioni</i>	L'utente non è loggato nell'applicazione
<i>Postcondizioni</i>	L'utente è loggato nell'applicazione
<i>Processo</i>	<ol style="list-style-type: none"> 1. L'utente clicca sul tasto "Login" e viene reindirizzato alla schermata per inserire le credenziali. 2. L'utente compila i campi con le proprie credenziali (indirizzo mail e password). 3. L'utente clicca sul tasto "Login" per confermare la volontà di voler accedere con il proprio account 4. Viene visualizzato un pop-up con scritto "Login successfully completed" e si viene reindirizzati alla schermata precedente
<i>Alternative</i>	<ul style="list-style-type: none"> • <u>Inserimento dati errati</u> Al passo (2) sono inseriti dei dati errati (mail non valida, credenziali errate). Viene mostrato un pop-up di errore con scritto "Wrong credentials".

UC3: Log-out dall'applicazione

<i>Descrizione</i>	Un utente vuole effettuare il log-out dall'applicazione
<i>Requisiti coperti</i>	GU10
<i>Attori coinvolti</i>	Cliente, Proprietario
<i>Precondizioni</i>	L'utente è loggato nell'applicazione
<i>Postcondizioni</i>	L'utente non è loggato nell'applicazione

<i>Processo</i>	<ol style="list-style-type: none"> 1. L'utente, loggato all'interno del proprio account sull'applicazione, entra nel menu a scomparsa in alto a sinistra e clicca su "Logout" 2. È richiesta la conferma dell'azione: se l'azione è confermata, l'utente effettua il logout. 3. Viene visualizzato un pop-up con scritto "Logout successfully completed" e si viene reindirizzati alla schermata Home
-----------------	--

4.2 Registrazione

La prima componente implementata in questa iterazione riguarda la fase di registrazione iniziale da parte di un utente.

Nonostante sia possibile accedere all'applicazione e ricercare appartamenti come utente ospite, risulta necessario avere un profilo registrato per poter poi effettuare prenotazioni e mettere a disposizione appartamenti.

4.2.1 Funzionamento

La procedura di registrazione viene suddivisa in due fasi: richiesta di registrazione e conferma registrazione.

Inizialmente l'utente fornisce al sistema i dati necessari tramite un modello predisposto (sezione 4.4.4). Una volta compilata la form e inoltrata la richiesta, il sistema si occupa di verificare l'eventuale presenza di un utente già registrato con lo stesso indirizzo email oppure lo stesso codice fiscale. In tal caso, la richiesta viene respinta e viene chiesto di verificare se l'utente è già registrato o la correttezza dei dati inseriti.

Altrimenti, il sistema invia una mail all'indirizzo e-mail fornito, all'interno della quale vi è un link da cliccare per confermare la registrazione. Il link di conferma ha una scadenza di 24 ore; perciò, se il link viene cliccato successivamente a questo termine, la registrazione non andrà a buon fine.

Se invece l’utente conferma la registrazione correttamente, allora potrà effettuare il login e accedere alle funzionalità dell’app.

4.2.2 Component diagram

In *Figura 8* viene mostrato il component diagram contenente le componenti e le interfacce predisposte per la fase di registrazione.

Il componente *RegistrationController* ricopre il ruolo di intermediario, fornendo alla View l’interfaccia di accesso e inoltrando le richieste di quest’ultima ai sottocomponenti specifici.

Il componente *RegistrationService* interagisce con lo *UserService* per inserire nel database il nuovo utente, con l’*EmailSender* per inviare la mail di conferma registrazione e con il *ConfirmationTokenService* per verificare la scadenza della conferma di registrazione.

Infine, i componenti *UserDao* e *ConfirmationTokenDao* sono quelli adibiti a gestire il collegamento con il database, i cui metodi di fatto vengono trasformati in query di SELECT, INSERT, UPDATE e DELETE.

Per la gestione della conferma della registrazione il sistema genera con l’inserimento dell’utente nel database un token, al cui interno vi è memorizzato l’utente, la richiesta a cui fa riferimento e la scadenza temporale. Nella mail inviata all’utente, il link inserito fa riferimento alla pagina di conferma registrazione e nell’URL viene inserito il codice del token relativo.

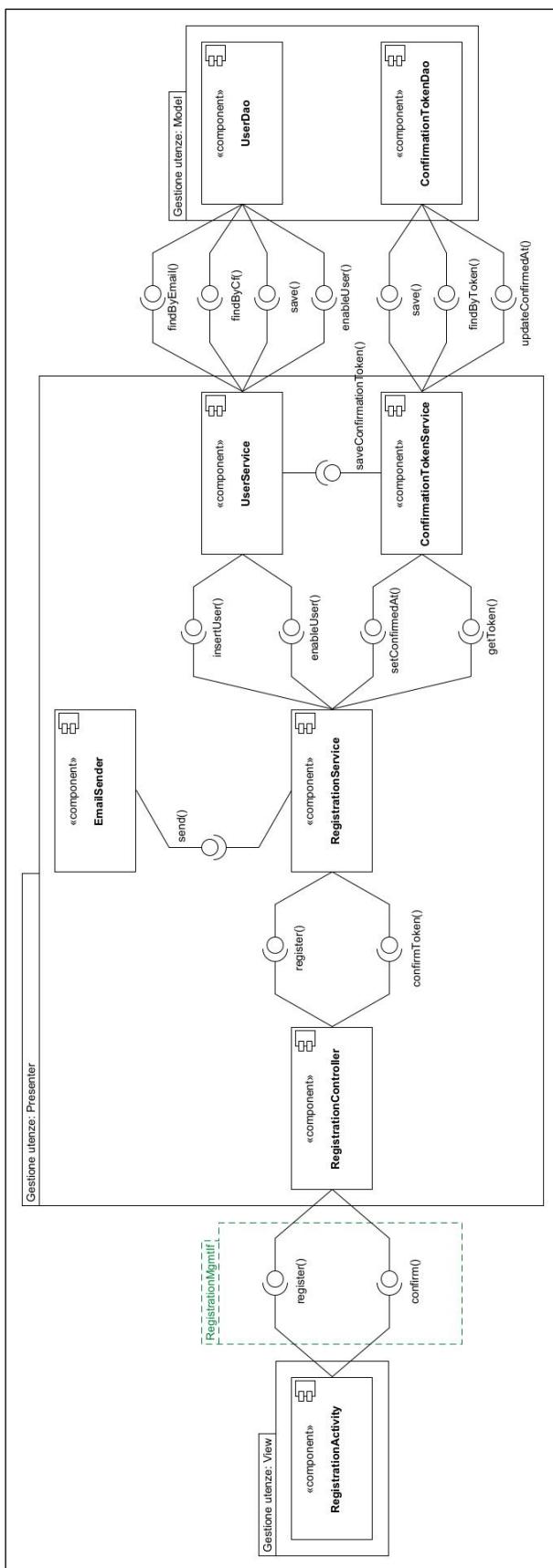


Figura 8: Component diagram del servizio di registrazione

4.2.3 Sequence diagram

In questa sezione viene mostrato il processo di registrazione tramite due sequence diagram, uno per la richiesta e uno per la conferma.

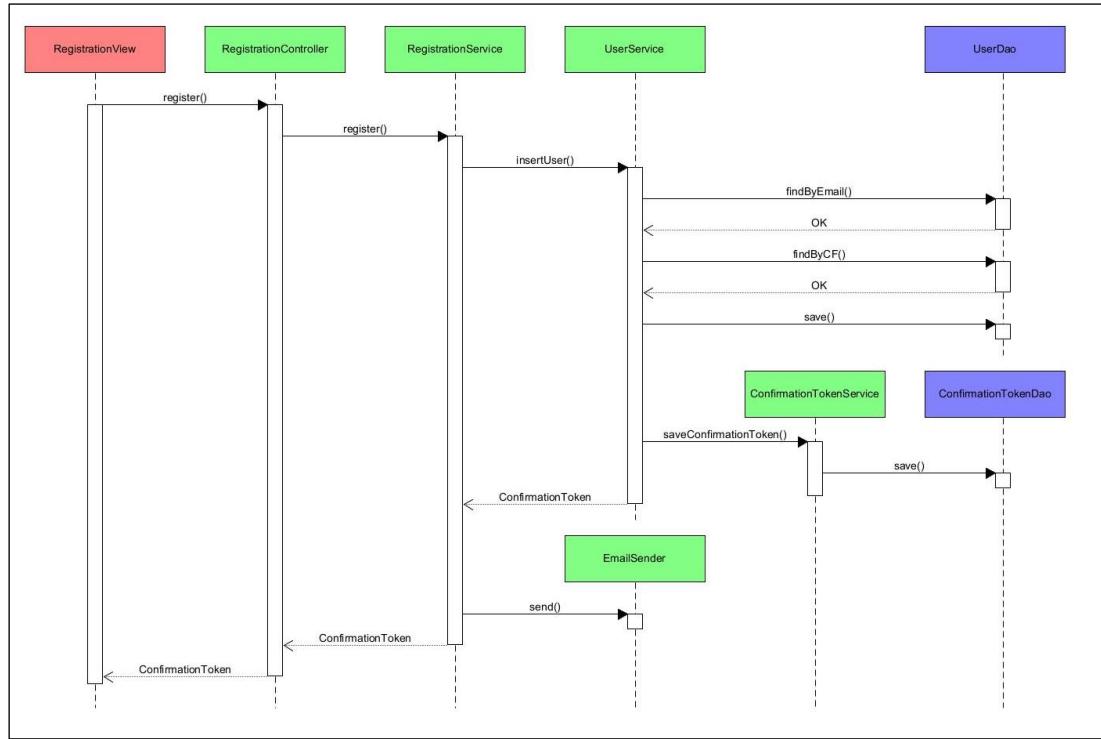


Figura 9: Sequence diagram del servizio di registrazione - Richiesta

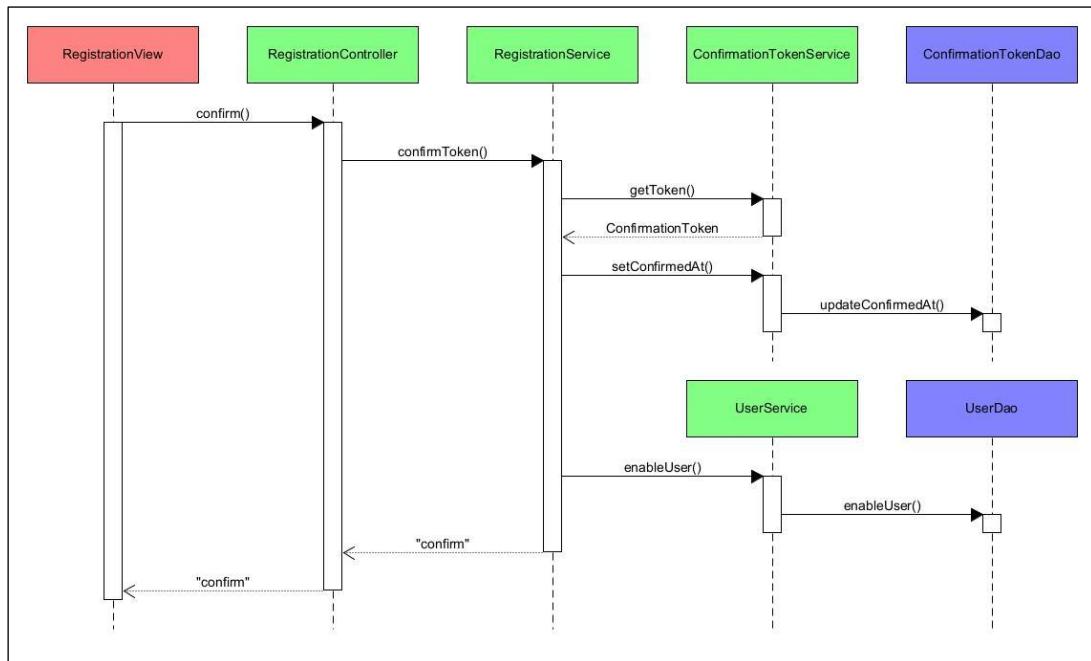


Figura 10: Sequence diagram del servizio di registrazione - Conferma

4.3 Login - Logout

La seconda componente implementata nell’iterazione corrente riguarda i processi di login e logout dall’applicazione. Anche in questo caso, le procedure sono effettuate tramite interfacce user-friendly definite nella sezione 4.4.4.

4.3.1 Spring security

Per gestire il login degli utenti è stato fatto affidamento alla componente Spring Security.

Spring Security ha permesso di implementare in modo *seamless* la logica del login senza il bisogno di dover esporre un endpoint adibito a questa funzione.

Con questo modulo di Spring, infatti, è possibile implementare le configurazioni di autenticazione in una classe che estende *WebSecurityConfigurerAdapter.java*, la classe stessa espone in modo automatico l’endpoint “/login” a cui il front-end dell’applicazione invierà una form-data costituita da username e password.

KEY	VALUE	DESCRIPTION	...	Bulk Edit
username	I.Rossi7@gmail.com			
password	abcd123			
Key	Value	Description		

Figura 11: Struttura form di login

In questa classe è stato possibile, tramite la classe di Spring Security *AuthenticationManagerBuilder*, implementare la logica di controllo delle password salvate sul database. Il front-end invia l’e-mail e la password inserite dall’utente e l’*AuthenticationManagerBuilder* recupera dal database la password relativa alla mail. Dal momento che sul database le password sono cryptate, tramite la classe *BCryptPasswordEncoder* viene cifrata la password ricevuta dal front-end e poi si controlla che corrisponda a quanto ottenuto dal database.

In Spring Security il fallimento in fase di autenticazione si gestisce istanziando una classe che implementa l’interfaccia *AuthenticationFailureHandler*. L’interfaccia, e in

particolare il metodo *onAuthenticationFailure*, sono state implementate in modo che in caso di password sbagliata venga restituito a front-end l'HTTP status code 401 UNAUTHORIZED.

4.4 Front-End

4.4.1 Interazione con il back-end

L'implementazione dei casi d'uso per la registrazione e il login/logout dell'utente hanno richiesto che l'app Android potesse comunicare con il back-end attraverso l'utilizzo di connettori che fornissero lo scambio di dati in un formato compatibile.

4.4.2 Retrofit

Il connettore scelto è Retrofit. Si tratta di una libreria ampliamente utilizzata in applicazioni realizzate tramite Android Studio che consente la realizzazione di un HTTP client che rispetta il “type-safe”.

Retrofit trasforma le API HTTP fornite dal back-end in una interfaccia Java che è possibile utilizzare per effettuare chiamate HTTP/HTTPS verso il server.

Funzionamento: Creazione Interfaccia

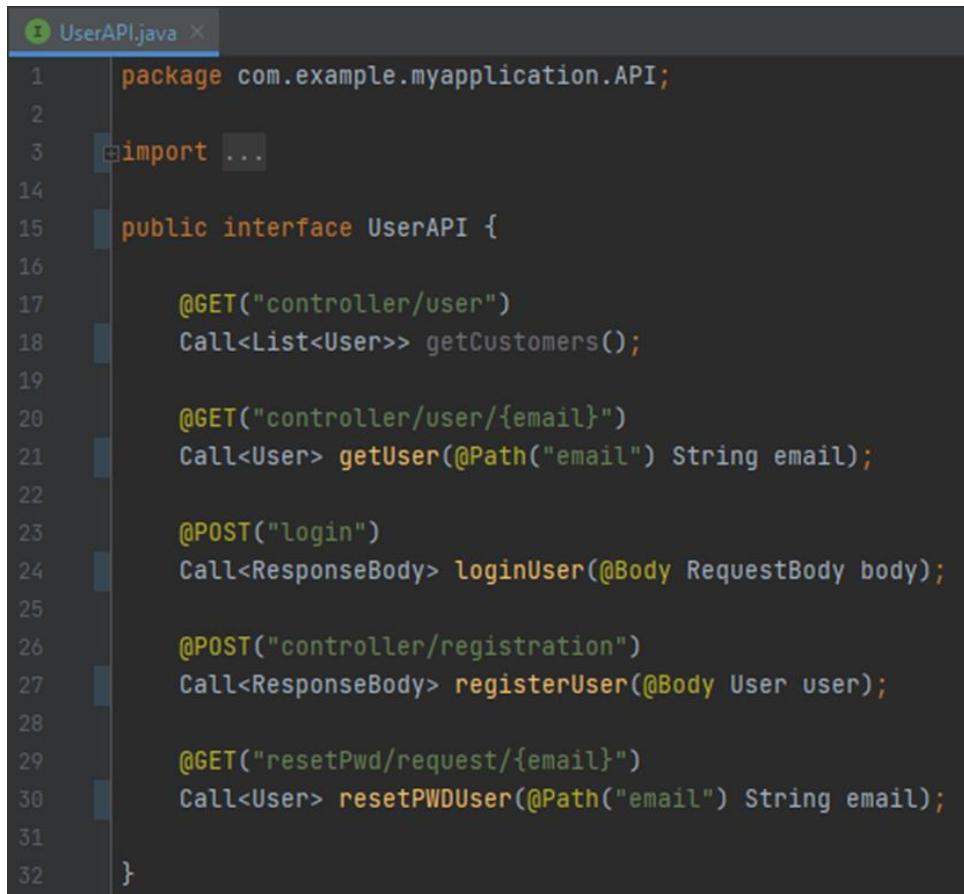
Il primo elemento da realizzare è l'interfaccia all'interno della quale è necessario definire i metodi, i metodi HTTP associati (get, post, update, ecc) e i rispettivi URL.

La struttura è semplice e lineare:

```
@MetodoHTTP ("URL")
```

```
MetodoInterfaccia (Parametri)
```

In Figura 12 si riporta l'interfaccia *UserAPI* utilizzata per la realizzazione gli Use case dell'iterazione 2:



```
1 package com.example.myapplication.API;
2
3 import ...
4
5 public interface UserAPI {
6
7     @GET("controller/user")
8     Call<List<User>> getCustomers();
9
10    @GET("controller/user/{email}")
11    Call<User> getUser(@Path("email") String email);
12
13    @POST("login")
14    Call<ResponseBody> LoginUser(@Body RequestBody body);
15
16    @POST("controller/registration")
17    Call<ResponseBody> registerUser(@Body User user);
18
19    @GET("resetPwd/request/{email}")
20    Call<User> resetPWDUser(@Path("email") String email);
21
22}
23
```

Figura 12: Interfaccia UserAPI realizzata con Retrofit

Retrofit consente la manipolazione degli URL in maniera dinamica tramite l'utilizzo di blocchi e parametri sostituibili all'interno del metodo. Per esempio, il metodo `Call<User> getUser(@Path("email") String email)` permette di modificare il campo "email" all'interno della path dell'URL in modo tale da ottenere le info del cliente specifico associato all'indirizzo di posta passato per parametro.

Funzionamento: Definizione chiamata HTTP

Il primo passo consiste nel creare un oggetto di tipo retrofit passando come parametri il "baseURL", da contattare tramite le chiamate HTTP, e il "converterFactory" ovvero l'oggetto che deserializerà la chiamata http (rif. 4.3.3 *Gson parser*).

In seguito, si costruisce un oggetto dell'interfaccia `UserAPI`, che verrà utilizzato per realizzare le chiamate sincrone/asincrone verso il server, e il body della chiamata HTTP contenente i dati da inviare al back-end.

Il metodo "enqueue" consente di effettuare una chiamata HTTP asincrona.

Tramite i metodi “onResponde” / ”onFailure” e ai codici di risposta HTTP è possibile personalizzare il comportamento dell’applicazione. Per esempio, effettuando la chiamata POST a <http://10.0.0.10:8080/login> (Host locale) si può ottenere:

- a) Una risposta di tipo “Successfull” (codici di stato HTTP 2xx): in questo caso il login dell’utente è andato a buon fine.
- b) Una risposta di tipo “Client error” (codici di stato HTTP 4xx): in questo caso il client non ha inserito credenziali valide e quindi il login dell’utente è fallito.
- c) Il fallimento della chiamata HTTP: ciò può essere dovuto ad un errore di comunicazione con il server per mancanza della linea internet o per indisponibilità momentanea del server.

In Figura 13 viene riportato il codice per la gestione del login dell’utente:

```
Retrofit retrofit = new Retrofit.Builder().baseUrl("http://10.0.0.10:8080/").addConverterFactory(GsonConverterFactory.create()).build();

UserAPI userAPI = retrofit.create(UserAPI.class);

RequestBody requestBody = new MultipartBody.Builder()
    .setType(MultipartBody.FORM)
    .addFormDataPart("username", inputEmail.getText().toString())
    .addFormDataPart("password", inputPassword.getText().toString())
    .build();

userAPI.loginUser(requestBody).enqueue(new Callback<ResponseBody>() {
    @Override
    public void onResponse(Call<ResponseBody> call, Response<ResponseBody> response) {
        if(!response.isSuccessful()) {
            Toast.makeText(context, LoginActivity.this, text: "WRONG CREDENTIALS.", Toast.LENGTH_LONG).show();
        } else {
            Toast.makeText(context, LoginActivity.this, text: "LOGIN COMPLETED SUCCESSFULLY.", Toast.LENGTH_LONG).show();
            userAPI.getUser(inputEmail.getText().toString()).enqueue(new Callback<User>() {
                @Override
                public void onResponse(Call<User> call, Response<User> response) {
                    if(response.isSuccessful()) {
                        Toast.makeText(context, LoginActivity.this, text: "Something gone wrong, please make another attempt.", Toast.LENGTH_LONG).show();
                    } else {
                        User loggedUser = response.body();

                        currentSession = new SessionManager(_context: LoginActivity.this);
                        currentSession.createLoginSession(loggedUser.id, loggedUser.name, loggedUser.surname, loggedUser.email, loggedUser.cf, loggedUser.dob);

                        flag = false;
                        startActivity(new Intent(packageContext: LoginActivity.this, HomePageActivity.class));
                    }
                }

                @Override
                public void onFailure(Call<User> call, Throwable t) {
                    Toast.makeText(context, LoginActivity.this, text: "Error during retrieving data.", Toast.LENGTH_LONG).show();
                }
            });
        }
    }

    @Override
    public void onFailure(Call<ResponseBody> call, Throwable t) {
        Toast.makeText(context, LoginActivity.this, text: "Error during login, please make another attempt.", Toast.LENGTH_LONG).show();
    }
});
```

Figura 13: Codice per gestione login

4.4.3 Gson parser

Di default, Retrofit può deserializzare solo i body HTTP del tipo *ResponseBody* di *OkHttp*. Di conseguenza, per poter gestire dei body in altri formati è necessario implementare nel codice una libreria in grado di effettuare il parsing di body in formati differenti.

Il formato dei dati che si è scelto di utilizzare è Json, di conseguenza è stata implementata all'interno del codice del front-end la libreria “Gson”.

Questa libreria consente di effettuare con semplicità e immediatezza la serializzazione/deserializzazione di oggetti Json in oggetti Java.

4.4.4 Android View e ViewGroup

La componente grafica di Android Studio si basa sul concetto di “View”. Una “View” è un blocco base della User Interface di Android realizzata in XML che risponde agli input dell'utente. Esistono vari tipi di View: EditText, Buttons, CheckBox, ecc.

Questi elementi possono essere raggruppati all'interno di *ViewGroup* che ne consentono la gestione e il posizionamento all'interno dell'applicazione Android.

Essendo definite con il linguaggio XML questa gestione viene implementata sfruttando la struttura ad albero dei documenti XML.

Le tre principali tipologie di *ViewGroup* sono:

1. Linear Layout: consente di allineare tutti gli elementi figli (le varie View) in una singola direzione: orizzontale o verticale.
2. Relative Layout: visualizza le Child View in posizioni relative. La posizione di ciascuna vista può essere specificata come relativa agli elementi di pari livello (come a sinistra o al di sotto di un'altra vista) o in posizioni relative al genitore.
3. Constraint Layout: vincola la posizione di una View ad una determinata distanza (espressa in pixel o in percentuale della dimensione del display) con gli altri oggetti presenti nel layout.

Per la realizzazione delle interfacce di login e di registrazione si è scelto di usare il Constraint Layout in quanto consentiva di rendere la UI compatibile con device con schermi di diversa dimensione e formato in maniera semplice e scalabile.

Register Activity

L’interfaccia di registrazione è mostrata in *Figura 14*.

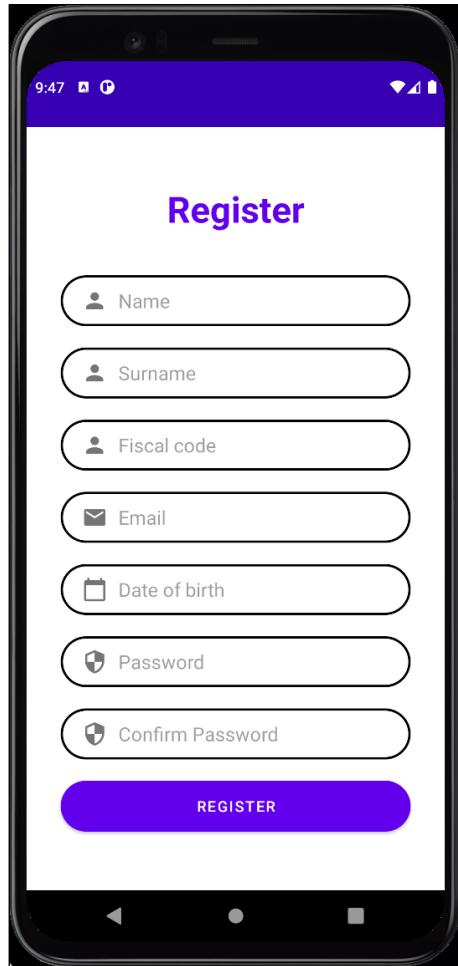


Figura 14: Register Activity

Nell’implementazione sono stati introdotti diversi controlli per verificare che l’input fosse valido e che un utente già registrato non potesse registrarsi nuovamente.

I controlli implementati sono i seguenti:

- Controllo validità codice fiscale (tramite regular expression – RegEx).
- Controllo validità mail (tramite regular expression – RegEx).
- Controllo validità password (tramite regular expression – RegEx).
- Controllo di campi non compilati.
- Verifica per utente già registrato (tramite controllo dell’esistenza del codice fiscale o della mail all’interno del database).

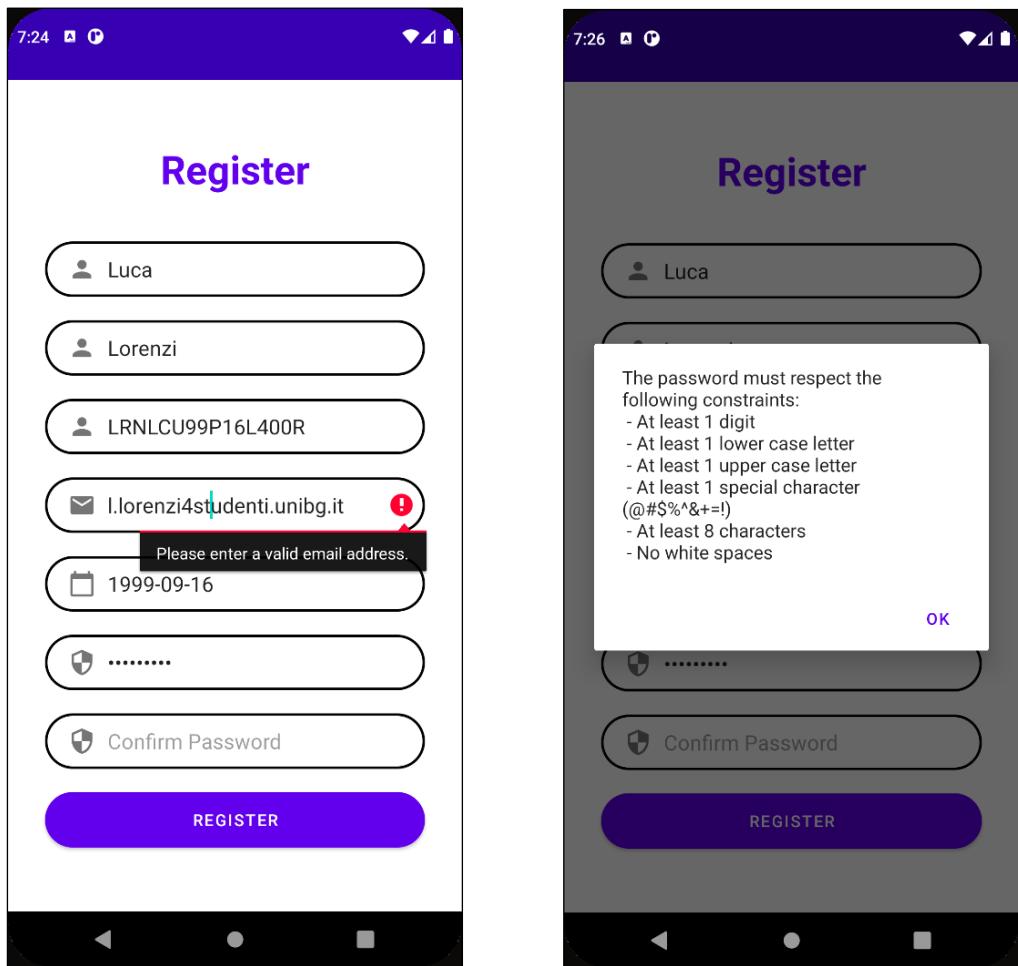


Figura 15: Register Activity - Compilazione campi

Cliccando sul bottone “register” i dati dell’utente sono inoltrati al back-end che verifica se l’utente esiste già oppure se è possibile procedere con la procedura di registrazione inviando la mail di attivazione. Nel primo caso l’applicazione mostrerà un errore a schermo indicando che l’utente è già registrato; nel secondo verrà recapitata la mail e l’utente sarà reindirizzato alla schermata di login.

Login Activity

L’interfaccia di login è riportata in *Figura 16*.

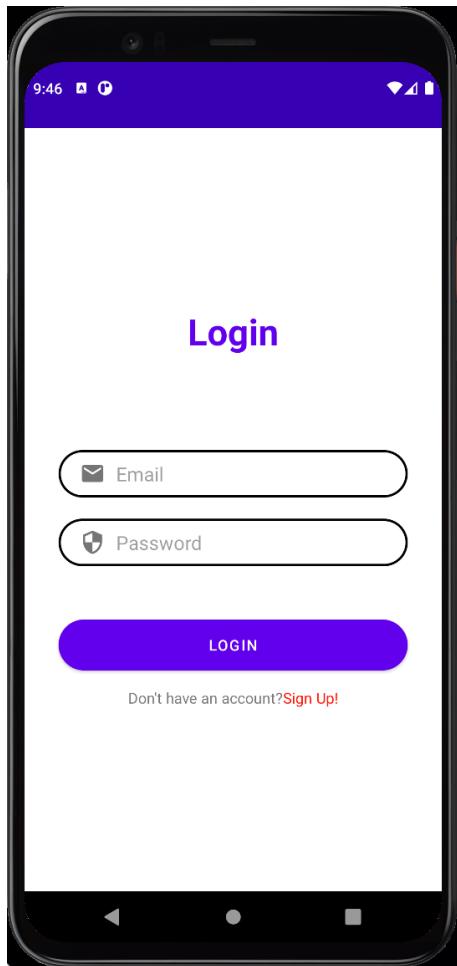


Figura 16: Login Activity

Analogamente alla Register activity sono stati implementati dei controlli sui campi compilabili con i rispettivi messaggi a schermo in caso di errore o mancata compilazione. Cliccando sul bottone “Login” i dati dell’utente sono inviati al back-end che verifica la correttezza delle credenziali. Se l’autenticazione va a buon fine l’utente viene avvisato che l’operazione è andata a buon fine e viene reindirizzato sulla home; nel caso in cui l’autenticazione fallisse verrà visualizzato un messaggio di errore a schermo.

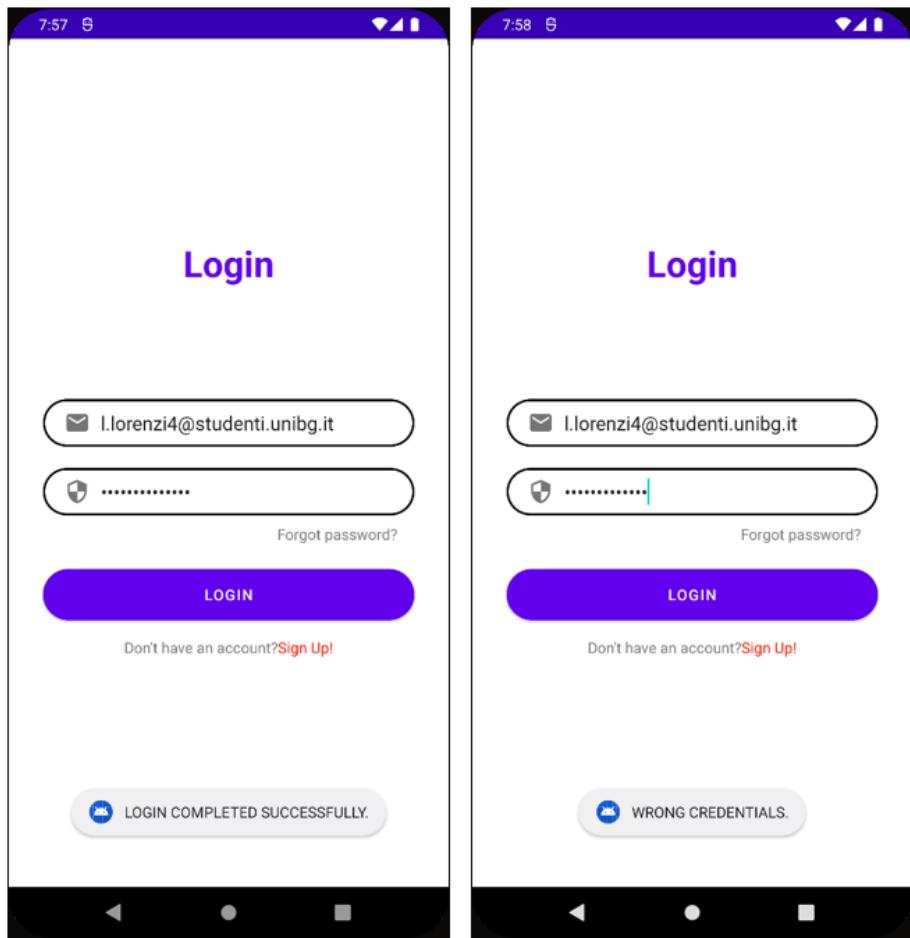


Figura 17: Login Activity - Test output

4.5 Test e analisi del componente

4.5.1 Analisi statica

Come riportato nel paragrafo 1.2 per l'analisi statica è stato utilizzato CodeMR, un plug-in fornito dall'IDE IntelliJ IDEA.

Analizzando quanto riportato in *Figura 18*, risulta rispettata la struttura MVP che si era proposto di realizzare.

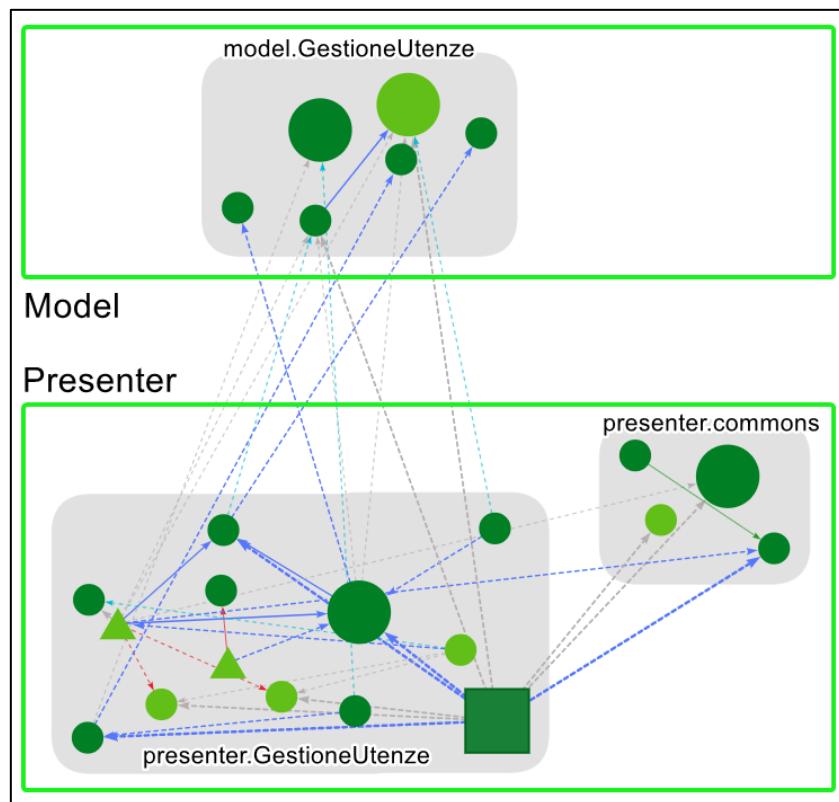


Figura 18: Analisi statica del codice – Iterazione 2

Abstractness e Instability

Dai risultati ottenuti dall'analisi effettuata da CodeMR sono emersi i seguenti valori di abstractness e instability:

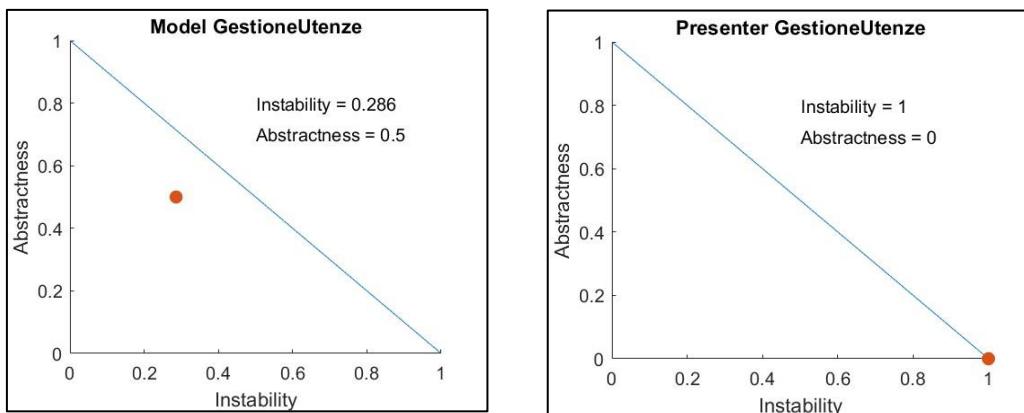


Figura 19: Distance – Model e Presenter GestioneUtenze

Osservando le figure sopra si può affermare che:

1. La componente Model per la Gestione utenze ha un valore vicino alla retta ideale, indicando un buon equilibrio tra Abstractness e Instability.
2. La componente Presenter per la Gestione utenze invece ha una forte concretezza e instabilità che non risultano essere un problema in quanto nessuna classe dipende direttamente da quelle presenti nel package.

Quality Attributes

Di seguito vengono riportati gli attributi di Complexity, Coupling, Lack of Cohesion e Size inerenti ai package *Model.GestioneUtenze* e *Presenter.GestioneUtenze*.



Figura 20: Legenda grafici
CodeMR

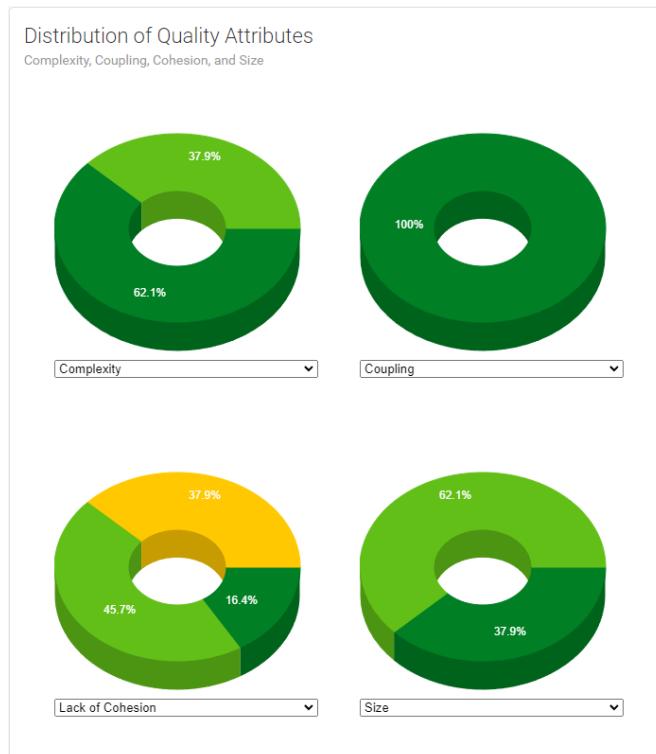


Figura 21: Quality attributes - Model.GestioneUtenze

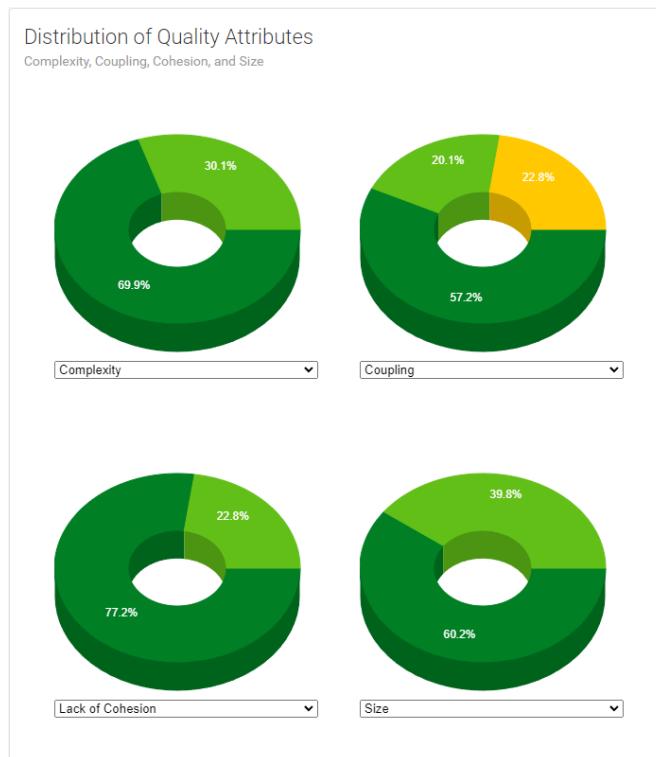


Figura 22: Quality attributes - Presenter.GestioneUtenze

Dai risultati ottenuti dall'esecuzione di CodeMR si evidenziano valori medio-bassi degli attributi di qualità. Questi valori indicano una buona qualità del codice in quanto risulta essere semplice e riutilizzabile.

4.5.2 Analisi dinamica

In questa sezione verranno analizzati alcuni casi di test relativi al codice implementato in questa iterazione.

In particolare, è stato effettuato il test delle seguenti funzioni:

- UC1: Registrazione utente
- UC2: Log-in nell'applicazione

Nello specifico vengono mostrate le request e le response ottenute richiamando gli endpoint delle api fino ad ora implementate. La registrazione di un utente avviene trasmettendo un form Json contenente i dati dell'utente, se la registrazione va a buon fine (il server risponde con HTTP Status 200) viene inviata al cliente una mail con un link da cliccare per attivare l'account. Così facendo si è sicuri che il cliente che ha inviato la richiesta di registrazione abbia effettivamente accesso alla mail indicata.

The screenshot shows the PostMan interface with the following details:

- Collection:** New Collection / iterazione 2 / registrazione utente
- Method:** POST
- URL:** http://localhost:8080/controller/registration
- Body:** Body tab selected, showing JSON content:

```
1 {"cfn": "MTRSMN99P27423L",
2 "name": "Simone",
3 "surname": "Matera",
4 "dob": "1999-06-08",
5 "email": "trapani.matera@gmail.com",
6 "pwd": "abcd123"
7 }
```
- Headers:** Headers (8) tab selected
- Tests:** Tests tab selected
- Settings:** Settings tab selected
- Buttons:** Save, Send, and other icons

Figura 23: Test registrazione con PostMan

Nella registrazione sono presenti tre casistiche diverse, testate ed elencate di seguito:

1. La prima riguarda la registrazione dell'account con conseguente trasmissione della mail per la conferma dell'account, ci si aspetta una risposta http 200.

POST	registrazione utente	http://localhost:8080/controller/registration	/ iterazione 2 / registrazione utente	200 OK	1587 ms	305 B
Pass	Status test					

Figura 24: HTTP response code per registrazione avvenuta

- La seconda riguarda la registrazione di un account già presente ma che non è stato confermato via mail, ci si aspetta una risposta http 406.

POST	registrazione utente	http://localhost:8080/controller/registration	/ iterazione 2 / registrazione utente	406 Not Acceptable	1429 ms	317 B
Pass	Status test					

Figura 25: HTTP response code per errore durante la registrazione - Utente già registrato ma non confermato

- La terza riguarda la registrazione di un account già presente e confermato via mail, ci si aspetta una risposta http 409.

POST	registrazione utente	http://localhost:8080/controller/registration	/ iterazione 2 / registrazione utente	409 Conflict	27 ms	311 B
Pass	Status test					

Figura 26: HTTP response code per errore durante la registrazione - Utente già registrato

Dopo aver richiamato l'endpoint di registrazione, che correttamente ha risposto con un HTTP Status 200, si evidenzia che il server ha effettivamente invocato il servizio di invio mail, che è stata inviata all'indirizzo mail inserito nel Json body della registrazione.

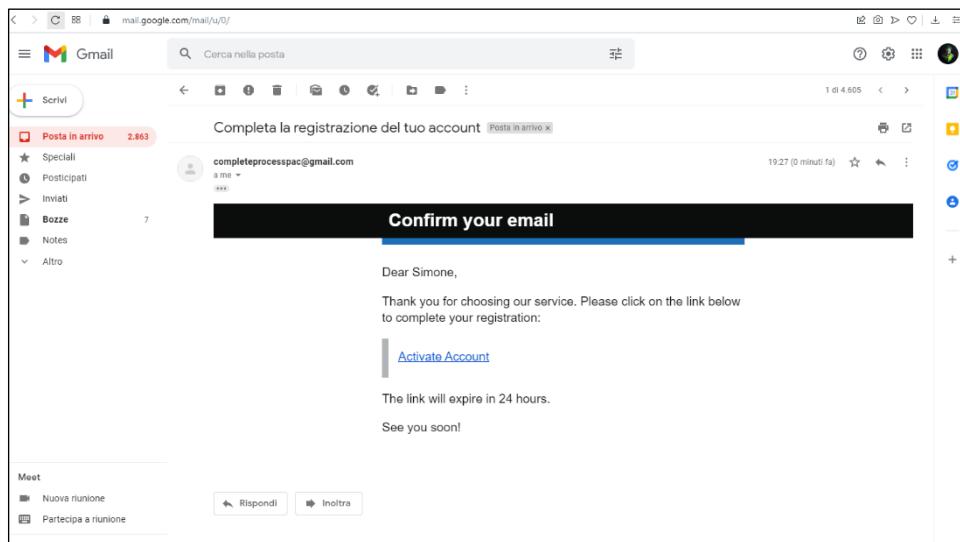


Figura 27: Mail di conferma registrazione

Cliccando il link ricevuto per mail viene mostrata la pagina di conferma.

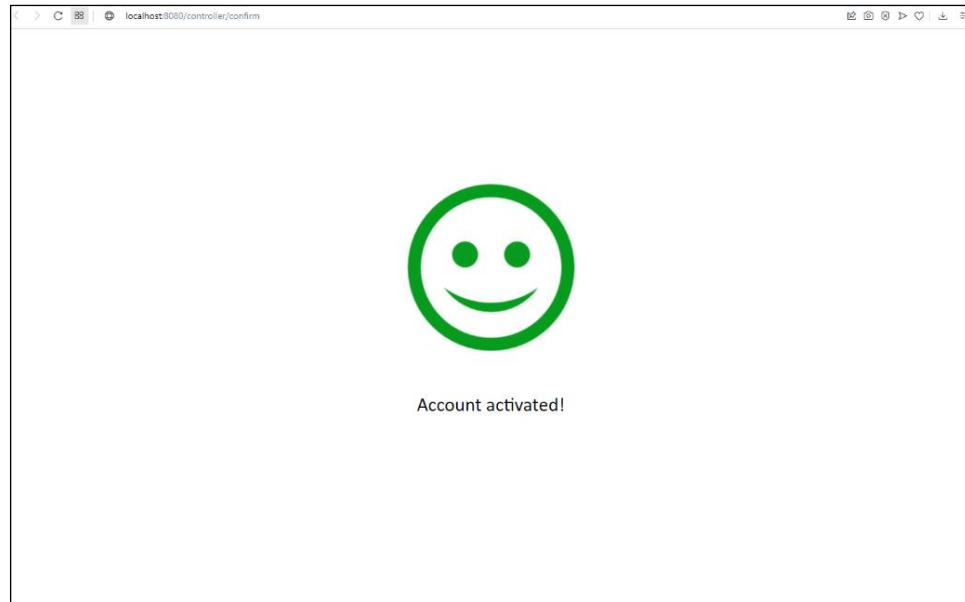


Figura 28: Pagina HTML per conferma registrazione di un account

La conferma dell'account via e-mail non è possibile in due casi: se l'account è già stato attivato (quindi se il relativo link inviato per mail è già stato cliccato); oppure se è scaduto l'intervallo di validità del link (15 minuti) quando esso viene cliccato. Entrambi i casi sono gestiti, se il cliente prova a utilizzare il link in queste casistiche viene mostrata una pagina di errore con un esplicito messaggio che spiega il motivo dell'errore (vedasi *Figura 29* e *Figura 30*).

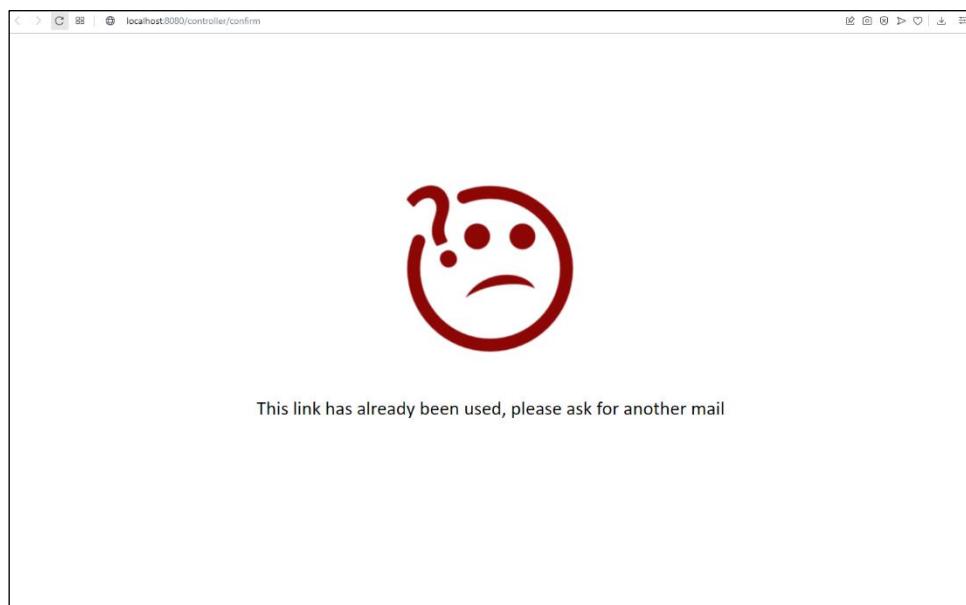


Figura 29: Pagina HTML per errore durante la registrazione di un account – Link già utilizzato

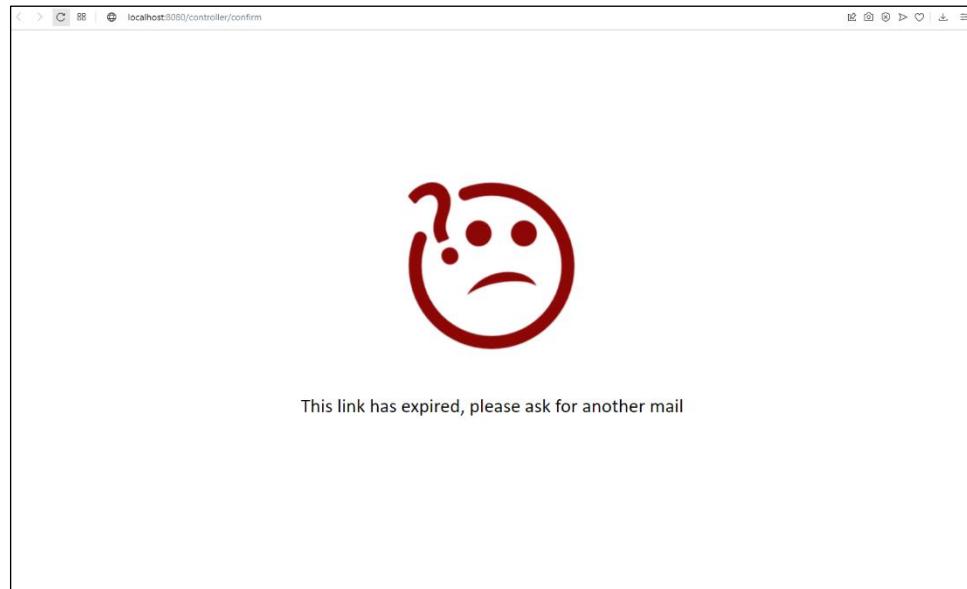


Figura 30: Pagina HTML per errore durante la registrazione di un account – Link scaduto

Per quanto riguarda il log-in sono state prese in considerazione le casistiche di login effettuato correttamente e non correttamente, mostrati in seguito;

- Log-in corretto: il form è compilato con le credenziali corrette

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> username	trapani.matera@gmail.com			
<input checked="" type="checkbox"/> password	abcd123			
Key	Value	Description		

Figura 31: Test positivo Login

- Log-in scorretto: il form è compilato con una password errata

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> username	trapani.matera@gmail.com			
<input checked="" type="checkbox"/> password	abcd123gd			
Key	Value	Description		

Figura 32: Test negativo Login

Nel caso del login corretto il server ha risposto correttamente con un HTTP Status 200, mentre nel caso di credenziali scorrette il server ha risposto con un HTTP Status 401 che indica che il tentativo di chiamata eseguito è rifiutato in quanto non autorizzato.

I risultati dei test sono riportati in *Figura 33*.

POST	log-in corretto	http://127.0.0.1:8080/login	/ Iterazione 2 / log-in corretto	200 OK	95 ms	305 B
	Pass	Status test				
POST	log-in scorretto	http://127.0.0.1:8080/login	/ iterazione 2 / log-in scorretto	401 Unauthorized	240 ms	315 B
	Pass	Status test				

Figura 33: Output delle chiamate di test sulla funzione di Login

5 Iterazione 3

5.1 Selezione delle funzioni da implementare

Per l'iterazione 3 si è scelto di implementare una delle funzioni primarie di questa applicazione, ossia la ricerca degli alloggi. Ad essa sono collegate anche la visualizzazione dei risultati, la visualizzazione della scheda completa degli alloggi e l'inserimento di una prenotazione.

Perciò, verranno sviluppati i seguenti casi d'uso:

- UC11: Visualizzazione scheda alloggio
- UC12: Ricerca alloggi
- UC13: Inserimento prenotazione

Per la ricerca e la prenotazione di un alloggio viene fornita un'interfaccia user-friendly, descritta nella sezione 5.5, nella quale l'utente deve inserire un indirizzo, le date di inizio e fine della prenotazione, e può specificare una serie di filtri, tra i quali il numero di ospiti, il massimo prezzo per notte, la massima distanza dall'indirizzo indicato e una serie di tag che descrivono le caratteristiche del luogo che vogliono visitare. Inoltre, l'utente può scegliere se ricercare solo singoli alloggi o anche gruppi di alloggi (gruppi da 2, 3, 4 o massimo 5 appartamenti).

Una volta cliccato il tasto per la ricerca, vengono visualizzati, se presenti, i risultati corrispondenti ai vincoli imposti, e cliccando su uno di questi verrà visualizzata la scheda completa dell'alloggio, corredata di fotografia, dati descrittivi e mappa interattiva.

Cliccando successivamente il tasto “BOOK NOW!”, è possibile inserire la prenotazione.

5.1.1 Use Case descriptions

UC11: Visualizzazione scheda alloggio

<i>Descrizione</i>	Si vuole visualizzare la scheda di un alloggio
<i>Requisiti coperti</i>	GA4, GA5, GR4
<i>Attori coinvolti</i>	Cliente, Proprietario
<i>Precondizioni</i>	Vari alloggi sono presenti nel database
<i>Postcondizioni</i>	Viene visualizzata la scheda di un alloggio
<i>Processo</i>	<ol style="list-style-type: none"> 1. L'utente clicca sulla scheda di un alloggio (per esempio in una ricerca o nella scheda "My apartments"). 2. Viene visualizzato il profilo dell'alloggio selezionato, con i dettagli principali, le recensioni registrate per quell'alloggio ed il calendario di disponibilità per eventuali prenotazioni.

UC12: Ricerca alloggi

<i>Descrizione</i>	Un cliente vuole ricercare un alloggio
<i>Requisiti coperti</i>	RA1-RA9
<i>Attori coinvolti</i>	Cliente, Sistema di geolocalizzazione
<i>Precondizioni</i>	Un cliente è loggato nell'applicazione
<i>Postcondizioni</i>	Viene visualizzato un elenco di alloggi
<i>Processo</i>	<ol style="list-style-type: none"> 1. L'utente clicca sulla barra di ricerca per gli alloggi. 2. L'utente inserisce la località dove vuole ricercare l'alloggio. <ol style="list-style-type: none"> a) L'utente può impostare dei filtri come: distanza massima, prezzo massimo, tag, ecc. b) L'utente inserisce le date in cui vuole prenotare l'alloggio.

	<p>c) L'utente può scegliere se visualizzare solo alloggi singoli o anche gruppi di alloggi.</p> <p>3. L'utente clicca il simbolo della lente di ingrandimento per far partire la ricerca degli alloggi che rispetta i filtri selezionati al punto (2).</p> <p>4. Il sistema restituisce la lista degli alloggi.</p>
<i>Alternative</i>	<ul style="list-style-type: none"> • <u>Non ci sono risultati per la ricerca corrente</u> Al passo (3) la ricerca non restituisce risultati per i filtri impostati.

UC13: Inserimento prenotazione

<i>Descrizione</i>	Un cliente vuole effettuare una prenotazione
<i>Requisiti coperti</i>	GA4, GA5, GP1, GP2, GP6, GP12, PG4, PG5
<i>Attori coinvolti</i>	Cliente, Sistema di pagamento
<i>Precondizioni</i>	Un cliente è loggato nell'applicazione
<i>Postcondizioni</i>	La prenotazione del cliente è stata effettuata
<i>Processo</i>	<ol style="list-style-type: none"> 1. L'utente entra all'interno della scheda di un alloggio. 2. L'utente clicca sul tasto “Book this apartment”. 3. L'utente seleziona il metodo di pagamento per la prenotazione. 4. L'utente clicca su “Confirm”. <ul style="list-style-type: none"> a) È richiesta la conferma dell'azione: se l'azione è confermata, viene inserita la prenotazione. 5. La richiesta di prenotazione viene inoltrata al proprietario dell'alloggio.
<i>Alternative</i>	<ul style="list-style-type: none"> • <u>Metodo di pagamento non valido</u>

	Al passo (3.c) il metodo di pagamento non è valido o la transazione non è andata a buon fine.
<i>Estensioni</i>	<ul style="list-style-type: none"> • <u>Conferma tramite e-mail</u> L'utente riceve una e-mail contenente la conferma dell'avvenuta prenotazione e il riepilogo dei dati della prenotazione.

5.2 Ricerca singola

L'algoritmo analizzato riguarda appunto la ricerca degli alloggi in base ai filtri specificati in ingresso dall'utente. La ricerca per singoli alloggi e la ricerca per gruppi di alloggi vengono eseguite tramite due metodi differenti, perciò si è deciso di distinguere i due casi e analizzare gli algoritmi separatamente.

In questa sezione si tratta la prima versione, quella della ricerca singola.

5.2.1 Pseudocodice algoritmo

Innanzitutto, è stato scritto lo pseudocodice dell'algoritmo in un linguaggio misto astratto.

```
algoritmo singleResearch(int numGuests, String location, int maxDistance, String tags, double maxPricePerNight, Date prenotationStart, Date prenotationEnd) -> Lista di appartamenti

    L <- lista vuota di appartamenti

    if (maxPricePerNight > 0)
        then L = appartamenti disponibili filtrati per numGuests e maxPricePerNight
        else L = appartamenti disponibili filtrati per numGuests

    if (tags not empty && L not empty)
        then L.remove(appartamenti che non contengono tutti i tag in tags)

    if (L not empty) then
        if (maxDistance == 0) then maxDistance = 50000
        L.remove(appartamenti che distano più di maxDistance)

return L;
```

Figura 34: Pseudocodice dell'algoritmo di ricerca singola

L'algoritmo si divide in tre fasi, ciascuna delle quali va a raffinare la lista dei risultati.

In primis, vengono recuperati dal database gli appartamenti disponibili, ossia che non hanno già prenotazioni per le date specificate dall'utente (*prenotationStart* e *prenotationEnd*), filtrati per numero di ospiti (*numGuests*) e massimo prezzo per notte (*maxPricePerNight*), se quest'ultimo non è nullo.

Successivamente, vengono rimossi dalla lista gli appartamenti che non contengono tutti i tag selezionati dall'utente (*tags*), se questi non sono nulli.

Infine, vengono rimossi gli appartamenti che distano più della massima distanza indicata dall'utente (*maxDistance*). Se questa è nulla viene impostata ad un valore di default (50 km).

Quindi viene restituita la lista dei risultati.

5.2.2 Codifica Java e complessità

In seguito, l'algoritmo è stato implementato in codice Java, e ne è stata calcolata la complessità, valutando le possibili ottimizzazioni.

Il metodo *filteredResearch()* è mostrato in *Figura 35*.

Come classe per gestire l'elenco di appartamenti si è scelto di utilizzare la classe *ArrayList()*, che presenta diversi vantaggi a livello prestazionale, come si vedrà successivamente, in quanto è strutturata internamente come un array puro.

La prima fase prevede una chiamata al database, perciò la complessità dipende dalla complessità della query e dai meccanismi di efficienza gestiti dal DBMS PostgreSQL. Le query implementate sono descritte nel paragrafo 5.2.3.

Successivamente, per l'applicazione del vincolo sui tag, la complessità è $O(N \cdot M)$, dove N è il numero di appartamenti nella lista e M è il numero di tag degli appartamenti. Questo perché il metodo *removeIf()* della classe *ArrayList()* scandisce gli elementi dell'array, marca gli elementi che soddisfano il predicato e che devono essere rimossi, poi scandisce nuovamente l'array e ne crea uno nuovo con i soli elementi non rimossi, e perciò ha complessità $O(N)$. In questo caso, però, il predicato prevede la creazione di un array di tags tramite la funzione *split()*, che ha complessità $O(M)$, e l'esecuzione del metodo *containsAll()*, per verificare se tutti i tag inseriti dall'utente sono contenuti. Per rendere efficiente questo metodo si è scelto di effettuare un cast della lista creata in un oggetto *HashSet*, il cui metodo *containsAll()* ha complessità $O(M)$, in quanto sfrutta l'accesso diretto della tabella hash, mentre per un *ArrayList* avrebbe avuto complessità $O(M \cdot M)$ per il fatto che avrebbe dovuto scandire entrambe le liste interamente. Quindi per ogni elemento della lista viene richiamato il metodo *split()*, e di conseguenza la complessità di questa porzione di codice è $O(N \cdot M)$.

```

public List<Apartment> singleResearch(int numGuests, String location, int maxDistance, String tags,
                                         double maxPricePerNight, Date startReservation, Date endReservation) throws Exception {

    List<Apartment> apartmentList;

    if (maxPricePerNight > 0)
        apartmentList = apartmentDao.findAvailableApartmentsPerNumGuestsAndMaxPricePerNight(startReservation, endReservation, numGuests, maxPricePerNight);
    else
        apartmentList = apartmentDao.findAvailableApartmentsPerNumGuests(startReservation, endReservation, numGuests);

    if (!tags.equals("")) && !apartmentList.isEmpty() {
        List<String> tagList = Arrays.stream(tags.split(",")).toList();
        apartmentList.removeIf(a -> !(new HashSet<>(Arrays.stream(a.getTags().split(",")).toList()).containsAll(tagList)));
    }

    if (!apartmentList.isEmpty()) {
        if (maxDistance == 0)
            maxDistance = 50000;
        apartmentList = checkDistance(apartmentList, location, maxDistance);
    }
}

return apartmentList;
}

```

Figura 35: Metodo *singleResearch()*

Infine, viene richiamato il metodo *checkDistance()*, mostrato in *Figura 36*, che ha complessità $O(N)$. Questo metodo costruisce prima un array di stringhe che rappresentano gli indirizzi degli appartamenti in lista. Il primo ciclo *foreach* ha complessità $O(N)$ in quanto scandisce una volta sola la lista e inserisce la stringa in una posizione specifica di un array puro. Successivamente viene invocata l'API esterna e vengono valutati i risultati. Il secondo ciclo *foreach* ha complessità $O(N)$ poiché vengono scanditi uno alla volta i risultati dell'API, viene recuperato l'appartamento all'indice i tramite il metodo *get()* della classe *ArrayList()*, che prevede un accesso puntuale con complessità $O(1)$, e viene eseguito il metodo *add()* della classe *ArrayList()*, che aggiunge in coda all'array ($O(1)$). Per questo il metodo *checkDistance()* ha complessità $O(N)$.

La complessità tempo totale del metodo *filteredReasearch()* risulta

$$O(N) + O(N \cdot M) + O(N) = O(N \cdot M)$$

Considerando che il numero di tag è molto contenuto, è possibile approssimare M ad una costante, e quindi la complessità del metodo diventa $O(N)$.

```

private List<Apartment> checkDistance(List<Apartment> apartmentList, String location, int maxDistance) throws Exception {

    List<Apartment> result = new ArrayList<>();

    // creo un array di destinazioni
    String[] destinations = new String[apartmentList.size()];
    int i = 0;
    for (Apartment a: apartmentList) {
        destinations[i++] = a.getLocation();
    }

    // creo la connessione ed eseguo una richiesta per l'API DistanceMatrixApi
    GeoApiClientContext context = new GeoApiClientContext.Builder().apiKey(GEO_API_KEY).build();

    DistanceMatrix distanceMatrix = DistanceMatrixApi.newRequest(context)
        .origins(Location)
        .destinations(destinations)
        .mode(TravelMode.WALKING)
        .language("en")
        .units(Unit.METRIC)
        .await();

    // aggiungo alla lista dei risultati gli appartamenti che rispettano il vincolo
    i = 0;
    for (DistanceMatrixElement e: distanceMatrix.rows[0].elements) {
        if (e.distance != null) {
            if (e.distance.inMeters <= maxDistance)
                result.add(apartmentList.get(i));
        }
        i++;
    }

    context.shutdown();
    return result;
}

```

Figura 36: Metodo *checkDistance()*

5.2.3 Query

Le query utilizzate in questa fase sono due. Come già detto precedentemente, esse sono utilizzate per recuperare dal database gli appartamenti disponibili (che non hanno prenotazioni attive nel periodo specificato dalla ricerca), con numero di ospiti compreso tra quello imposto dall'utente e lo stesso maggiorato di 2, e si distinguono per l'applicazione o meno del filtro sul massimo prezzo per notte.

```
findAvailableApartmentsPerNumGuests(Date startR, Date endR, int numG)

SELECT      *
FROM        Apartment a
WHERE       a.numGuests BETWEEN numG AND numG+2
              AND a.id NOT IN (
SELECT      distinct a2.id
FROM        Apartment a2 INNER JOIN Reservation r ON a2.id = r.apartmentId
WHERE       r.startReservation BETWEEN startR AND endR
              OR r.endReservation BETWEEN startR AND endR
)
```

Figura 37: Query *findAvailableApartmentsPerNumGuests()*

```
findAvailableApartmentsPerNumGuestsAndMaxPricePerNight(Date startR, Date endR, int numG,
double maxP)

SELECT      *
FROM        Apartment a
WHERE       a.numGuests BETWEEN numG AND numG+2
              AND a.pricePerNight <= maxP
              AND a.id NOT IN (
SELECT      distinct a2.id
FROM        Apartment a2 INNER JOIN Reservation r ON a2.id = r.apartmentId
WHERE       r.startReservation BETWEEN startR AND endR
              OR r.endReservation BETWEEN startR AND endR
)
```

Figura 38: Query *findAvailableApartmentsPerNumGuestsAndMaxPricePerNight()*

5.3 Ricerca multipla

In questa sezione si tratta il secondo caso, quello della ricerca multipla.

A riguardo sono state effettuate le seguenti assunzioni:

- Possono essere restituiti gruppi di al massimo 5 alloggi;
- Per restituire coppie di appartamenti il numero di ospiti complessivo deve essere superiore o uguale a 3;
- Per restituire gruppi di 3 appartamenti il numero di ospiti complessivo deve essere superiore o uguale a 6;
- Per restituire gruppi di 4 appartamenti il numero di ospiti complessivo deve essere superiore o uguale a 12;
- Per restituire gruppi di 5 appartamenti il numero di ospiti complessivo deve essere superiore o uguale a 15;
- Il numero minimo di ospiti di ogni appartamento in un gruppo è pari al rapporto tra numero di ospiti richiesti e il massimo numero di appartamenti per gruppo, arrotondato per difetto.

$$\minNumGuests = \left\lceil \frac{\text{numGuests}}{\maxNumApartments} \right\rceil$$

Le assunzioni sul numero minimo di ospiti sono state fatte per evitare di restituire gruppi di alloggi con numeri di ospiti troppo sbilanciati (ad esempio, se un utente effettua una ricerca con numero di ospiti pari a 10, il numero massimo di appartamenti per gruppi è di 3, e il numero minimo di ospiti per appartamento è $\left\lceil \frac{10}{3} \right\rceil = 3$; in questo modo una coppia di appartamenti rispettivamente da 2 e da 8 ospiti non viene restituita).

5.3.1 Pseudocodice algoritmo

Di seguito si riporta lo pseudocodice dell'algoritmo, che mostra ad alto livello le fasi in cui si suddivide la procedura. I dettagli dell'implementazione e la complessità vengono trattati nel paragrafo 5.3.2 successivo.

```

algoritmo multipleResearch(int numGuests, String location, int maxDistance, String tags,
double maxPricePerNight, Date prenotationStart, Date prenotationEnd) -> Lista di appartamenti

    L <- lista vuota di appartamenti

    if (maxPricePerNight > 0)
        then L = appartamenti disponibili filtrati per max numGuests e maxPricePerNight
    else L = appartamenti disponibili filtrati per max numGuests

    if (tags not empty && L not empty)
        then L.remove(appartamenti che non contengono tutti i tag in tags)

    if (L not empty) then
        if (maxDistance == 0) then maxDistance = 50000
        L.remove(appartamenti che distano più di maxDistance)

    R <- list dei risultati

    foreach Apartment a in L do
        if (a.numGuests = numGuests)
            then R.add(a, 1, 1)

    int maxNumApartments <- intero che rappresenta la massima dimensione dei gruppi
    if (numGuests < 3) then return R
    else if (numGuests < 6) then maxNumApartments = 2
    else if (numGuests < 12) then maxNumApartments = 3
    else if (numGuests < 15) then maxNumApartments = 4
    else maxNumApartments = 5

    int minNumGuests <- intero che rappresenta il minimo numero di ospiti
    minNumGuests = floor(numGuests/maxNumApartments)

    L.remove(appartamenti singoli con numero ospiti = numGuests)
    mergeSort(L) (ordine decrescente per numero di ospiti)

    data[] <- array di supporto utilizzato per la creazione delle combinazioni
    apartmentCombination(L, data, 0, L.size()-1, 0, maxNumApartments, minNumGuests,
    maxPricePerNight, R)

return R

```

Figura 39: Pseudocodice dell'algoritmo di ricerca multipla

```

algoritmo APARTMENTCOMBINATION(List<Apartment> L, Apartment[] data, int start, int end,
int index, int maxNumAp, int minNumGuests, int numGuests, double maxPricePerNight,
List<WrappedApartment> resultList)

    if (index == numMaxAp) then return

    int totGuests = 0
    int totPrice = 0.0

    for i = 0 to index-1 do
        totGuests += data[i].numGuests
        totPrice += data[i].pricePerNight

    for i = start to end do

        Apartment a = L.get(i)
        if (a.numGuests < minNumGuests) then continue
        if (totGuests + a.numGuests > numGuests) then continue
        if (maxPricePerNight == 0 & (totPrice + a.pricePerNight) > maxPricePerNight)
            then continue

        if (totGuests + a.numGuests == numGuests) then
            for i = 0 to index-1 do
                R.add(data[i], i+1, index+1)
                R.add(a, index+1, index+1)
            continue

            data[index] = a
            APARTMENTCOMBINATION(L, data, i+1, end, index+1, maxNumAp,
minNumGuests, numGuests, maxPricePerNight, R)

    end for

```

Figura 40: Pseudocodice dell'algoritmo di ricerca delle combinazioni

5.3.2 Codifica Java e complessità

In questo paragrafo si analizza l'implementazione in codice Java dell'algoritmo di ricerca multipla.

Le prime operazioni (*Figura 41*) sono simili o uguali alla ricerca singola, ossia la ricerca degli appartamenti tramite query filtrandoli per numero di ospiti e massimo prezzo per notte e la rimozione degli appartamenti che non rispettano i vincoli sui filtri e sulla massima distanza. La differenza in questo caso è che vengono recuperati gli appartamenti con numero di ospiti uguale o anche inferiore a quello specificato dall'utente, in quanto successivamente si andranno a creare gruppi di appartamenti, il cui totale numero di ospiti dovrà uguagliare quello richiesto. Come visto prima la complessità di questo blocco di operazioni è $O(N)$, con N numero di appartamenti.

Successivamente (*Figura 42*), si inizializza la lista dei risultati che verranno restituiti al front-end.

Per gestire i gruppi di appartamenti si è deciso di creare una classe *WrappedApartment*, che ha i seguenti campi:

- *apartment*, riferimento della classe *Apartment*, che rappresenta l'oggetto appartamento;
- *order*, intero che indica l'ordine in cui compare nel gruppo di appartamenti;
- *total*, intero che indica il numero totale di appartamenti nel gruppo.

Così una volta ricevuta la lista, l'applicazione sarà in grado di riconoscere quanti e quali appartamenti fanno parte del gruppo.

Innanzitutto, vengono inseriti nella lista dei risultati gli appartamenti singoli il cui numero di ospiti è uguale a quello richiesto dall'utente. La complessità è $O(N)$, in quanto si tratta di un semplice ciclo *foreach*.

Poi si determinano i valori di *maxNumApartments* e *minNumGuests*, effettuando una serie di costrutti *if* in base al valore di *numGuests*. Se questo è inferiore a 3, viene restituita la lista dei risultati finora costituita, in quanto non sarà necessario creare combinazioni, per le assunzioni fatte all'inizio della sezione 5.3.

```

public List<WrappedApartment> multipleResearch(int numGuests, String location, int maxDistance, String tags,
                                                double maxPricePerNight, Date startReservation, Date endReservation) throws Exception {

    List<Apartment> apartmentList;

    if (maxPricePerNight > 0)
        apartmentList = apartmentDao.findAvailableApartmentsPerMaxNumGuestsAndMaxPricePerNight(startReservation, endReservation, numGuests, maxPricePerNight);
    else
        apartmentList = apartmentDao.findAvailableApartmentsPerMaxNumGuests(startReservation, endReservation, numGuests);

    if (!tags.equals("") && !apartmentList.isEmpty()) {
        List<String> tagList = Arrays.stream(tags.split(" ")).toList();
        apartmentList.removeIf(a -> !new HashSet<>(Arrays.stream(a.getTags().split(" ")).toList()).containsAll(tagList));
    }

    if (!apartmentList.isEmpty()) {
        if (maxDistance == 0)
            maxDistance = 50000;
        apartmentList = checkDistance(apartmentList, location, maxDistance);
    }
}

```

Figura 41: Metodo *multipleResearch()* – prima parte

In seguito, vengono rimossi dalla lista tutti gli appartamenti il cui numero di ospiti è uguale a quello imposto dall'utente, in quanto sono già stati inseriti nella lista dei risultati e non vanno utilizzati per la creazione di combinazioni, utilizzando nuovamente il metodo `removeIf()`, con complessità $O(N)$.

Inoltre, la lista residua viene ordinata in ordine decrescente per numero di ospiti degli appartamenti, invocando il metodo `sort()` della classe `ArrayList`, il quale utilizza una versione dell'algoritmo Merge Sort con complessità $O(N \cdot \log N)$, e passandogli come parametro una lambda function che confronta il campo `numGuests` dei due oggetti `Apartment`. L'ordinamento risulta funzionale per la restituzione e la visualizzazione finale della lista di appartamenti nel front-end.

Per ultimo, si inizializza un array puro di oggetti `Apartment` di dimensione `maxNumApartments`, che sarà di supporto al metodo ricorsivo che crea le combinazioni di appartamenti, e si richiama il metodo `apartmentCombination()`, che si occupa di popolare la lista dei risultati, che infine vengono restituiti.

```

List<WrappedApartment> resultList = new ArrayList<>();

for (Apartment a: apartmentList) {
    if (a.numGuests == numGuests)
        resultList.add(new WrappedApartment(a, 1, 1));
}

int maxNumApartments;

if      (numGuests < 3)    return resultList;
else if (numGuests < 6)    maxNumApartments = 2;
else if (numGuests < 12)   maxNumApartments = 3;
else if (numGuests < 15)   maxNumApartments = 4;
else                      maxNumApartments = 5;

int minNumGuests = Math.floorDiv(numGuests, maxNumApartments);

apartmentList.removeIf(a -> a.numGuests == numGuests);

apartmentList.sort((a1, a2) -> {
    return Integer.compare(a2.numGuests, a1.numGuests);
});

Apartment[] data = new Apartment[maxNumApartments];
apartmentCombination(apartmentList, data, 0, apartmentList.size()-1, 0, maxNumApartments,
                      minNumGuests, numGuests, maxPricePerNight, resultList);

return resultList;
}

```

Figura 42: Metodo *multipleResearch()* – seconda parte

Per la scrittura del metodo per la generazione dei gruppi di appartamenti si è preso spunto da un algoritmo noto per la generazione di combinazioni di r elementi a partire da una lista di n elementi, riportato in *Figura 43*.

Questo algoritmo crea ricorsivamente un albero di combinazioni fissando il primo elemento libero e accostandogli i restanti elementi fino a che si verifica la condizione

$$end - i + 1 \geq r - index$$

In questo modo, viene resa efficiente la creazione di tutte le combinazioni di r elementi, contenute nei nodi foglia, in quanto verrà generato esattamente il numero di combinazioni

distinte, evitandone ripetizioni, e i nodi intermedi saranno solo quelli che portano ad un nodo foglia.

```
class Combination {

    static void combinationUtil(int arr[], int data[], int start,
                                int end, int index, int r)
    {

        if (index == r) {
            for (int j=0; j<r; j++)
                System.out.print(data[j]+" ");
            System.out.println("");
            return;
        }

        for (int i=start; i<=end && end-i+1 >= r-index; i++) {
            data[index] = arr[i];
            combinationUtil(arr, data, i+1, end, index+1, r);
        }
    }

    static void printCombination(int arr[], int n, int r)
    {
        int data[] = new int[r];
        combinationUtil(arr, data, 0, n-1, 0, r);
    }

    public static void main (String[] args) {
        int arr[] = {1, 2, 3, 4, 5};
        int r = 3;
        int n = arr.length;
        printCombination(arr, n, r);
    }
}
```

Figura 43: Algoritmo noto per la generazione di combinazioni

In *Figura 44* si mostra un esempio di albero generato da questo algoritmo, in cui viene richiesto di determinare le combinazioni di 3 elementi a partire da una lista di 5 elementi. I nodi generati da questo algoritmo sono quelli raffigurati in colore blu. Come si vede, le foglie contengono tutte e sole le combinazioni distinte, e i nodi intermedi che non portano ad una foglia non vengono considerati nell'algoritmo.

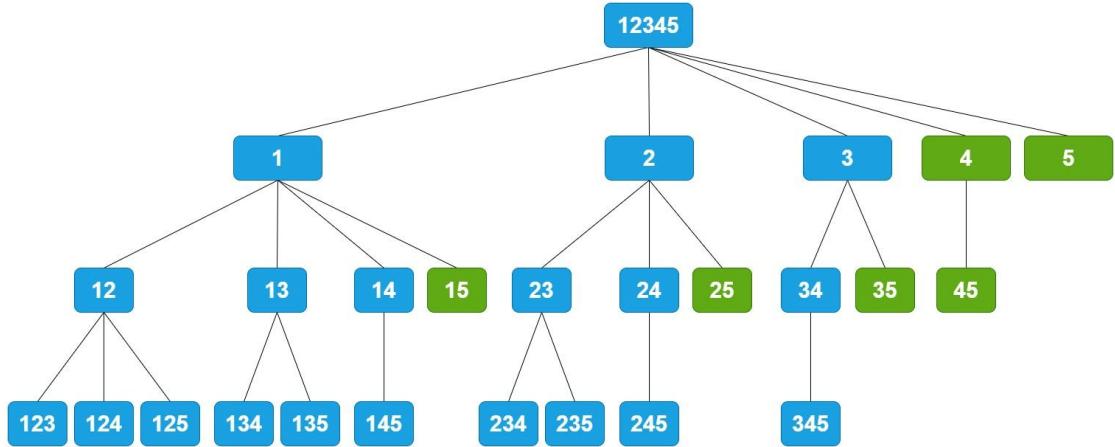


Figura 44: Esempio albero delle combinazioni

Nel nostro caso, però, si vogliono generare tutte le possibili combinazioni distinte con numero di elementi da 1 a r , ossia anche quelle raffigurate in colore verde.

Per questo, l'algoritmo è stato modificato per adattarsi al nostro scopo (*Figura 45*).

In particolare, è stata rimossa la condizione di uscita dal ciclo *for* vista precedentemente, in modo tale che venissero eseguite tutte le iterazioni da *start* a *end*. Così facendo, vengono generati anche i nodi intermedi che non portano a dei nodi foglia, evitando comunque la creazione di ripetizioni.

Il caso base di uscita dalla ricorsione rimane la condizione *index == maxNumApartments*, ossia quando l'ultima combinazione generata, contenuta nell'array *data*, è composta dal numero massimo di appartamenti, il metodo viene interrotto.

Prima di entrare nel ciclo *for* di scansione degli appartamenti successivi, viene calcolato il totale del numero di ospiti e del prezzo per notte con gli appartamenti finora presenti nella combinazione.

```

private void apartmentCombination(List<Apartment> apartmentList, Apartment[] data, int start, int end, int index, int maxNumAp,
                                 int minNumGuests, int numGuests, double maxPricePerNight, List<WrappedApartment> resultList) {

    if (index == maxNumAp) return;

    int totGuests = 0;
    double totPrice = 0.0;

    for (int i=0; i<index; i++) {
        totGuests += data[i].numGuests;
        totPrice += data[i].pricePerNight;
    }

    for (int i=start; i<end; i++) {

        Apartment a = apartmentList.get(i);

        if (a.numGuests < minNumGuests) continue;
        if (totGuests + a.numGuests > numGuests) continue;
        if (maxPricePerNight > 0 && (totPrice + a.pricePerNight) > maxPricePerNight) continue;

        if (totGuests + a.numGuests == numGuests) {
            for (int j = 0; j < index; j++)
                resultList.add(new WrappedApartment(data[j], j+1, index+1));
            resultList.add(new WrappedApartment(a, index+1, index+1));
            continue;
        }
    }

    data[index] = apartmentList.get(i);
    apartmentCombination(apartmentList, data, i+1, end, index+1, maxNumAp, minNumAp, numGuests, numGuests, maxPricePerNight, resultList);
}
}

```

Figura 45: Metodo ricorsivo *apartmentCombination()*

Nel ciclo *for* sono state aggiunte le seguenti condizioni che, se verificate, portano a saltare l’iterazione ed evitare una nuova ricorsione:

- L’appartamento della corrente iterazione non ha il minimo numero di ospiti;
- Aggiungendo l’appartamento della corrente iterazione alla combinazione il numero di ospiti totale eccede quello richiesto dall’utente;
- Aggiungendo l’appartamento della corrente iterazione alla combinazione il prezzo totale eccede quello massimo imposto dall’utente (se presente);
- Aggiungendo l’appartamento della corrente iterazione alla combinazione il numero di ospiti totale uguaglia quello richiesto dall’utente. In questo caso, la combinazione corrente generata rispetta tutti i vincoli imposti, perciò viene aggiunta alla lista dei risultati.

Se nessuna di queste è verificata, si aggiunge l’appartamento alla combinazione, inserendolo nell’array *data* in posizione *index*, e si procede con la ricorsione, incrementando gli indici *start* e *index* passati come parametri.

Il metodo così scritto porta alla generazione di tutte e sole le combinazioni distinte di appartamenti contenuti nella lista di partenza, di dimensione da 1 a *maxNumApartments*.

Per questo la complessità del metodo è pari al numero di combinazioni generate. Dalla teoria di calcolo combinatorio, il numero di combinazioni distinte di *r* elementi a partire da una lista di *n* elementi è

$$nC_r = \frac{n!}{(n - r)! \cdot r!}$$

Nel nostro caso, vengono generate le combinazioni con *r* da 1 a *maxNumApartments*, quindi il numero totale di combinazioni è

$$\sum_{r=1}^{maxNumApartments} nC_r = \sum_{r=1}^{maxNumApartments} \frac{n!}{(n - r)! \cdot r!}$$

Si ricorda che il valore di *maxNumApartments* è variabile e dipende dal valore di *numGuests* richiesto dall’utente.

Nel caso peggiore, ossia quando *maxNumApartments*=5, si ha:

$$\begin{aligned}
nC_5 &= \frac{n!}{(n-5)! \cdot 5!} = \frac{n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot (n-4) \cdot (n-5)!}{(n-5)! \cdot 5!} \\
&= \frac{n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot (n-4)}{120}
\end{aligned}$$

Di conseguenza l'ordine di complessità risulta $O(n^5)$.

Generalizzando, la complessità del metodo è $O(N^{maxNumApartments})$. In tabella 8 si mostrano i casi che si possono presentare.

<i>maxNumApartments</i>	Complessità <i>apartmentCombination()</i>
2	$O(N^2)$
3	$O(N^3)$
4	$O(N^4)$
5	$O(N^5)$

Tabella 8: Complessità algoritmo di ricerca delle combinazioni

La complessità del metodo *multipleResearch()* prima della chiamata al metodo ricorsivo *apartmentCombination()* era determinata dall'ordinamento della lista di appartamenti, ossia $O(N \cdot \log N)$, perciò in ogni caso la complessità dell'intero metodo è determinata da quella del metodo ricorsivo $O(N^{maxNumApartments})$.

5.3.3 Query

Le query utilizzate in questo caso sono due. A differenza della ricerca singola il campo numGuests è obbligatorio e diventa il limite massimo, mentre nella singola era concesso un valore addizionale di due persone.

```
findAvailableApartmentsPerMaxNumGuests(Date startR, Date endR, int numG)

SELECT      *
FROM        Apartment a
WHERE       a.numGuests <= numG
            a.id NOT IN (
                SELECT      distinct a2.id
                FROM        Apartment a2 INNER JOIN Reservation r ON a2.id = r.apartmentId
                WHERE      r.startReservation BETWEEN startR AND endR
                           OR r.endReservation BETWEEN startR AND endR
            )
```

Figura 46: Query *findAvailableApartmentsPerMaxNumGuests()*

```
findAvailableApartmentsPerMaxNumGuestsAndMaxPricePerNight(Date startR, Date endR, int numG, double maxP)

SELECT      *
FROM        Apartment a
WHERE       a.numGuests <= numG
            AND a.pricePerNight <= maxP
            AND a.id NOT IN (
                SELECT      distinct a2.id
                FROM        Apartment a2 INNER JOIN Reservation r ON a2.id = r.apartmentId
                WHERE      r.startReservation BETWEEN startR AND endR
                           OR r.endReservation BETWEEN startR AND endR
            )
```

Figura 47: Query *findAvailableApartmentsPerMaxNumGuestsAndMaxPricePerNight()*

5.4 Component diagram

In *Figura 48* viene raffigurato il component diagram relativo alla funzione ricerca.

La struttura è piuttosto semplice, con il controller *ApartmentController* che fornisce l’interfaccia *filteredResearch()* al componente *View* e che invoca il metodo *filteredResearch()* del componente *ApartmentService*. Quest’ultimo interagisce con il database tramite il componente *ApartmentDao*, il quale traduce i metodi esposti in query strutturate, e con il server di GoogleMaps tramite l’API *DistanceMatrixApi* (descritta nel paragrafo 5.3.2).

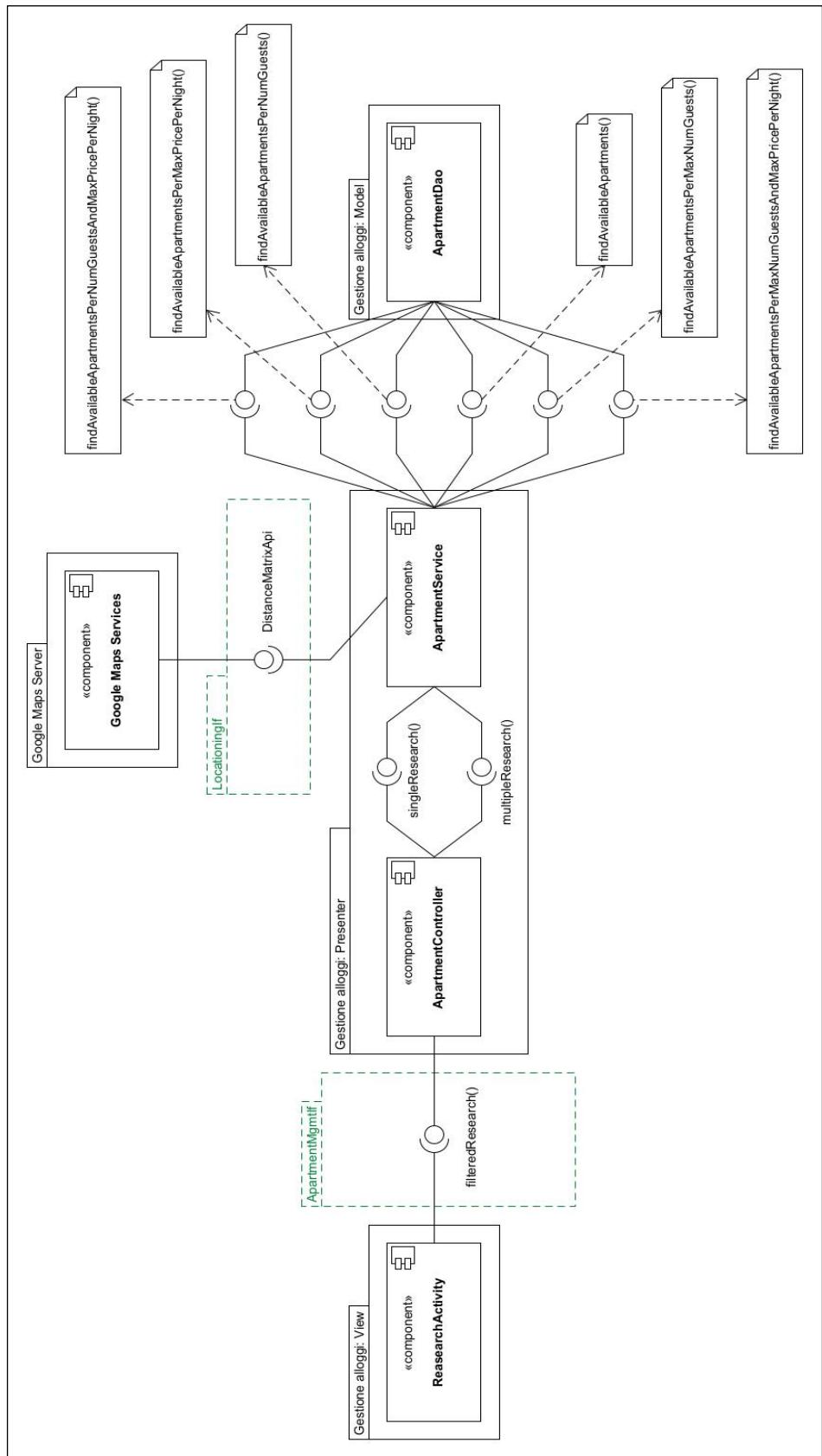


Figura 48: Component diagram della funzione ricerca

5.4.1 DistanceMatrixApi

Per il calcolo della distanza fra il luogo indicato dall'utente nella form di ricerca e l'indirizzo degli appartamenti presenti nel database è stata utilizzata l'API *DistanceMatrixApi*, fornita da Google Maps Services, la quale richiede in ingresso:

- *origins*: una stringa o un array di stringhe che rappresentano gli indirizzi sorgente da cui calcolare la distanza;
- *destinations*: una stringa o un array di stringhe che rappresentano gli indirizzi destinazione da cui calcolare la distanza.

E altri parametri opzionali, tra cui:

- *mode*: la modalità di spostamento (DRIVING, WALKING, BYCICLING, TRANSIT, UNKNOWN);
- *units*: l'unità di misura in cui viene espressa la distanza (METRIC, IMPERIAL);
- *language*: la lingua in cui vengono ritornati i risultati.

I risultati vengono memorizzati in un oggetto della classe *DistanceMatrix*.

Di seguito un esempio di risposta ad una chiamata in formato JSON (che viene automaticamente trasposto nei campi dell'oggetto *DistanceMatrix*):

A partire dall'oggetto della classe *DistanceMatrix*, si estrae una lista di oggetti *DistanceMatrixElement*, contenuti nella riga 0 poiché nel campo *origins* viene specificato un solo indirizzo, corrispondente a quello inserito dall'utente.

Da ciascuno di questi poi si ottiene il valore in metri della distanza calcolata tramite il metodo *distance.inMeters()*, e lo si confronta con quello massimo richiesto dall'utente.

```
{
  "destination_addresses": ["Lexington, MA, USA", "Concord, MA, USA"],
  "origin_addresses": ["Boston, MA, USA", "Charlestown, Boston, MA, USA"],
  "rows": [
    [
      {
        "elements": [
          [
            {
              "distance": { "text": "33.3 km", "value": 33253 },
              "duration": { "text": "27 mins", "value": 1620 },
              "duration_in_traffic": { "text": "34 mins", "value": 2019 },
              "status": "OK",
            },
            {
              "distance": { "text": "41.5 km", "value": 41491 },
              "duration": { "text": "33 mins", "value": 1981 },
              "duration_in_traffic": { "text": "39 mins", "value": 2342 },
              "status": "OK",
            }
          ],
        ],
      },
      {
        "elements": [
          [
            {
              "distance": { "text": "31.1 km", "value": 31100 },
              "duration": { "text": "26 mins", "value": 1543 },
              "duration_in_traffic": { "text": "29 mins", "value": 1754 },
              "status": "OK",
            },
            {
              "distance": { "text": "39.3 km", "value": 39338 },
              "duration": { "text": "32 mins", "value": 1904 },
              "duration_in_traffic": { "text": "35 mins", "value": 2077 },
              "status": "OK",
            }
          ],
        ],
      },
      "status": "OK",
    }
  ]
}
```

Figura 49: Esempio di risposta JSON della DistanceMatrixAPI

5.5 Sequence diagram

In *Figura 50* e *Figura 51* vengono mostrati i sequence diagram relativi alle due funzioni ricerca, singola e multipla, che riportano la cronologia delle interazioni tra le componenti già evidenziate nei codici e nel component diagram delle sezioni precedenti.

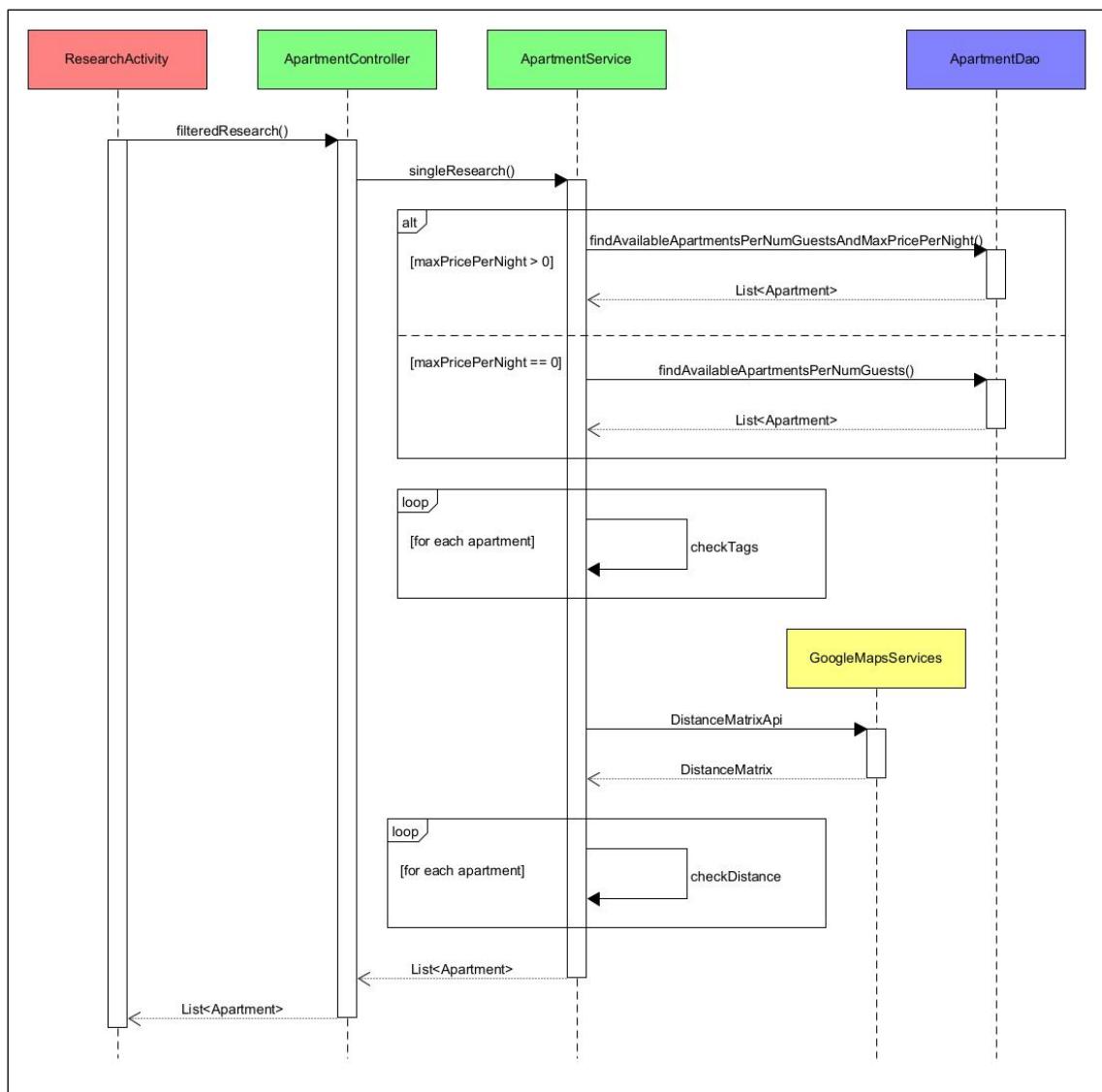


Figura 50: Sequence diagram ricerca singola

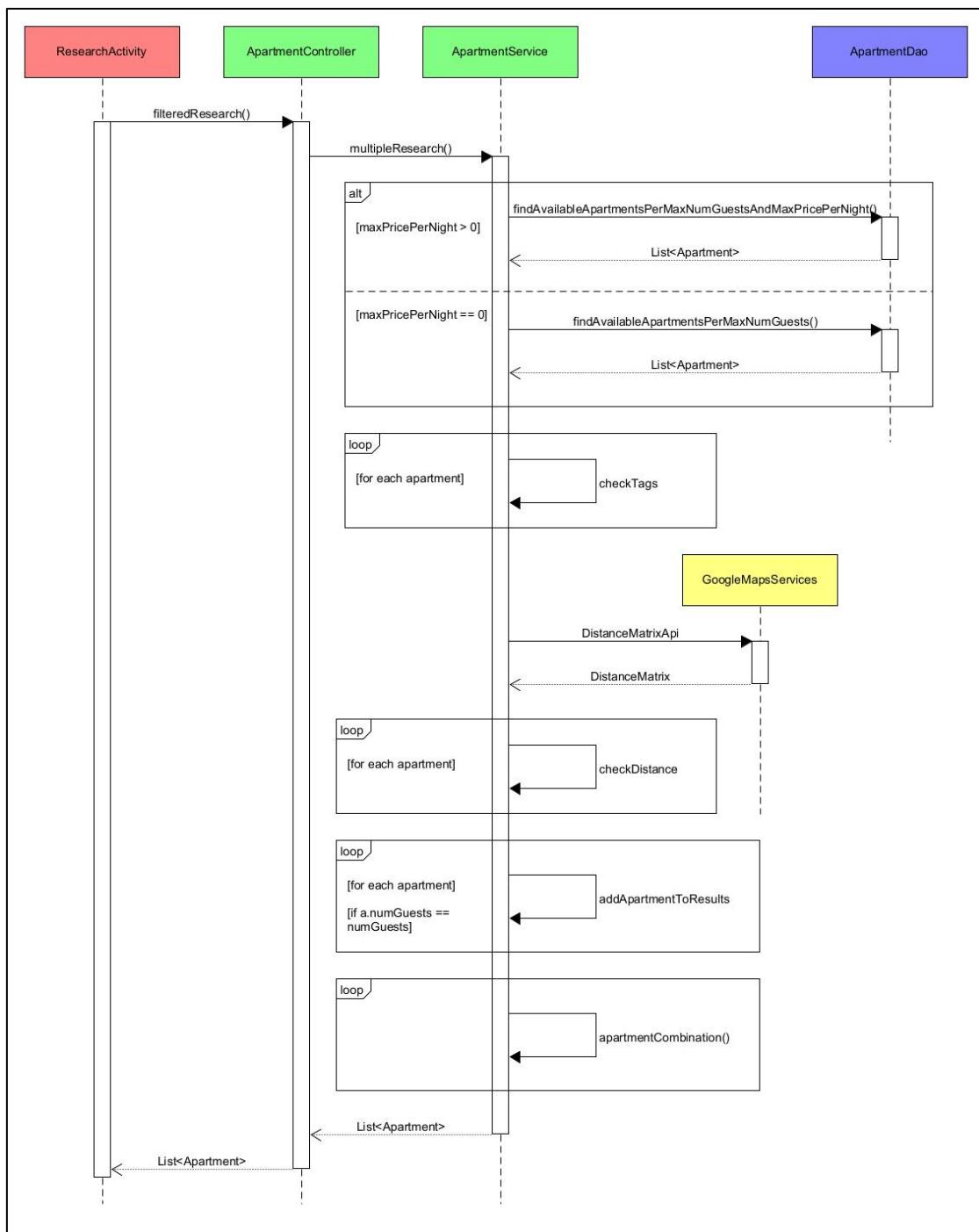


Figura 51: Sequence diagram ricerca multipla

5.6 Front-End

Per la realizzazione degli UC11, UC12 e UC13 sono state realizzate quattro activity e due classi di supporto:

- **SearchActivity**: contenente la barra di ricerca e i filtri.
- **ApartmentActivity**: contenente le informazioni dell'alloggio.
- **ResultSearchActivity** e **ResultSplitSearchActivity**: activities di supporto, inizializzano la lista dinamica e vi passano i parametri.
- **ApartmentRecViewAdapter** e **MultipleApartmentRecViewAdapter**: classi di supporto necessarie per la realizzazione della lista dinamica degli alloggi restituiti dalla ricerca.

ResultSplitSearchActivity e *MultipleApartmentRecViewAdapter* vengono specificamente usate per la ricerca di gruppi di alloggi.

Recycler view

La realizzazione della lista dei risultati non si è appoggiata alle strutture dati già presenti in android studio in quanto poco dinamiche e di difficile utilizzo. Al loro posto abbiamo deciso di implementare una *RecyclerView*.

La *RecyclerView* rende la visualizzazione di grandi insiemi di dati facile ed efficiente. È sufficiente fornire i dati e definire l'aspetto di ogni elemento e la libreria *RecyclerView* crea dinamicamente gli elementi quando sono necessari. I vantaggi di questa libreria, oltre alla semplicità di utilizzo e alla sua malleabilità, sono: il miglioramento delle prestazioni, della reattività e del consumo energetico.

SearchActivity

La activity per la ricerca degli alloggi è riportata in *Figura 52*:

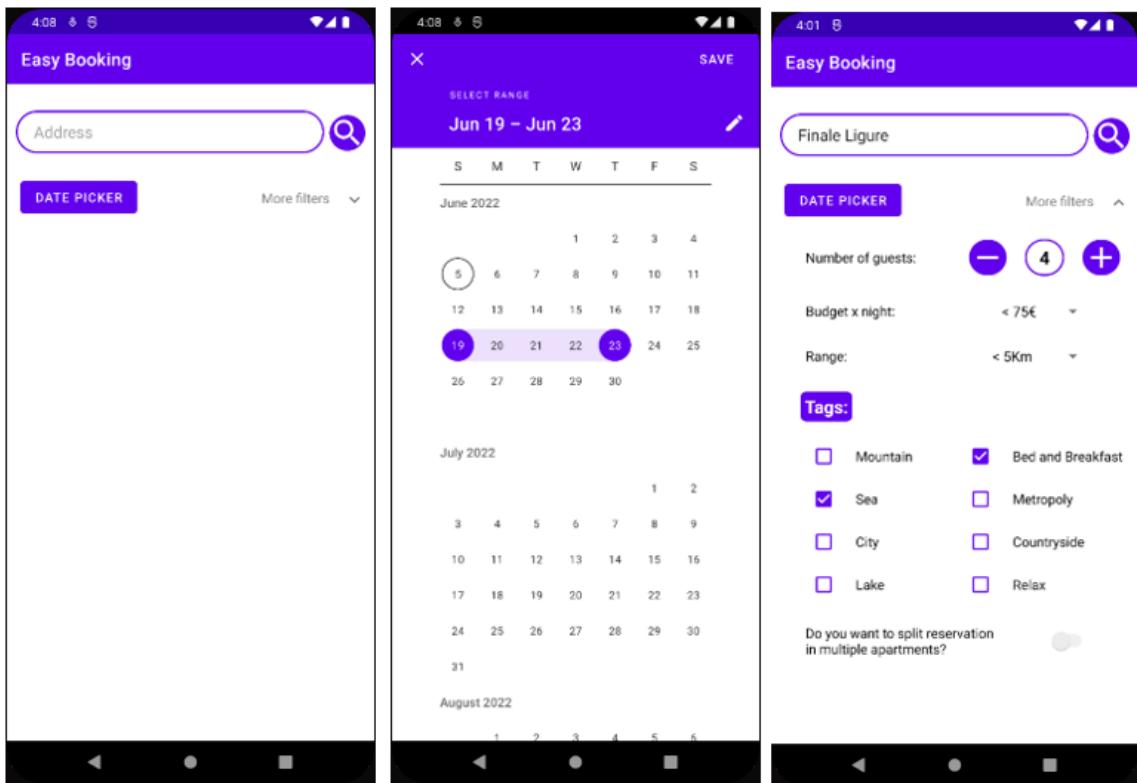


Figura 52: Search Activity

La schermata si compone di una barra di ricerca, affiancata da un *RangeDatePicker* ovvero un calendario dal quale è possibile selezionare un periodo di tempo con data di inizio e data di fine del soggiorno. Sulla destra troviamo il tasto della lente per la ricerca e tasto “*More filters*” per espandere i filtri di ricerca che di default sono nulli.

Selezionati i filtri desiderati è possibile inoltrare la richiesta al back-end tramite una chiamata HTTP Post con i parametri della ricerca serializzati in Json nel body della chiamata.

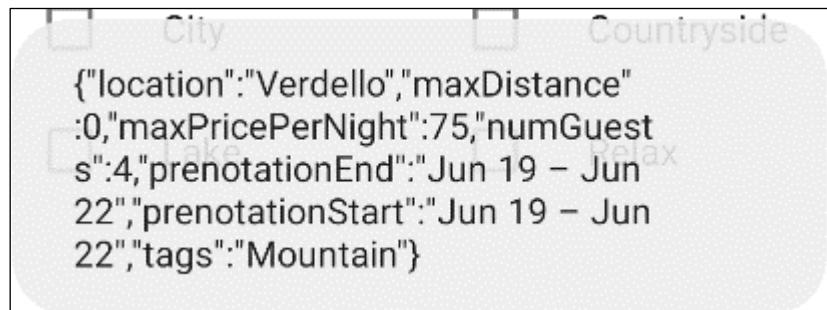


Figura 53: Body della richiesta HTTP in Json

ResultSearchActivity e ApartmentRecViewAdapter

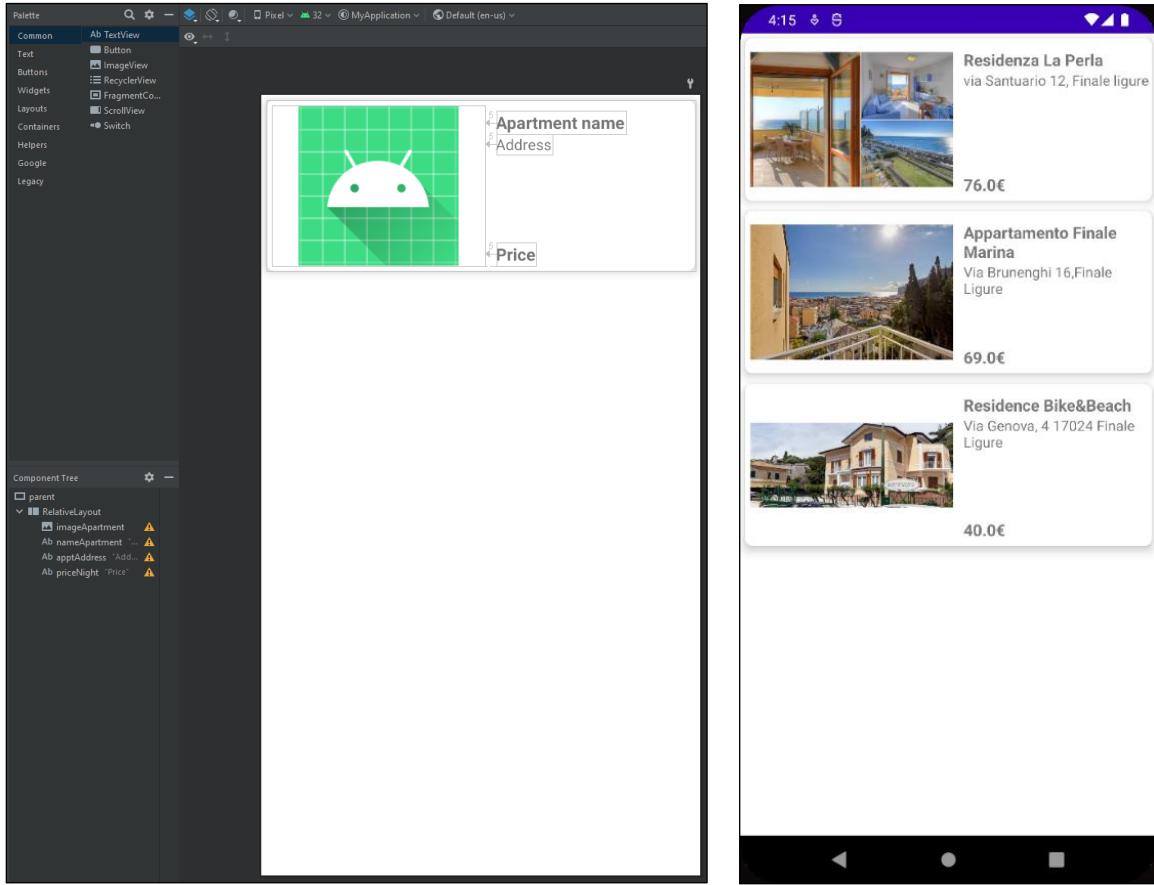


Figura 54: ResultSearchActivity e ApartmentRecViewAdapter

Per la realizzazione della *RecyclerView* è stato necessario definire per prima cosa la struttura del singolo elemento della lista, la sua visualizzazione e i campi da compilare.

In seguito, viene costruita la logica della lista dinamica tramite l'utilizzo del *RecyclerView.ViewHolder* e del *RecyclerView.Adapter*. Questi due elementi ci consentono di legare i dati ottenuti dal *back-end* agli elementi della struttura dati.

Cliccando su uno degli elementi si viene poi reindirizzati alla schermata specifica dell'appartamento dove è possibile effettuare la prenotazione.

Per la gestione dei gruppi di alloggi è stata realizzata una versione estesa della *RecyclerView* standard:

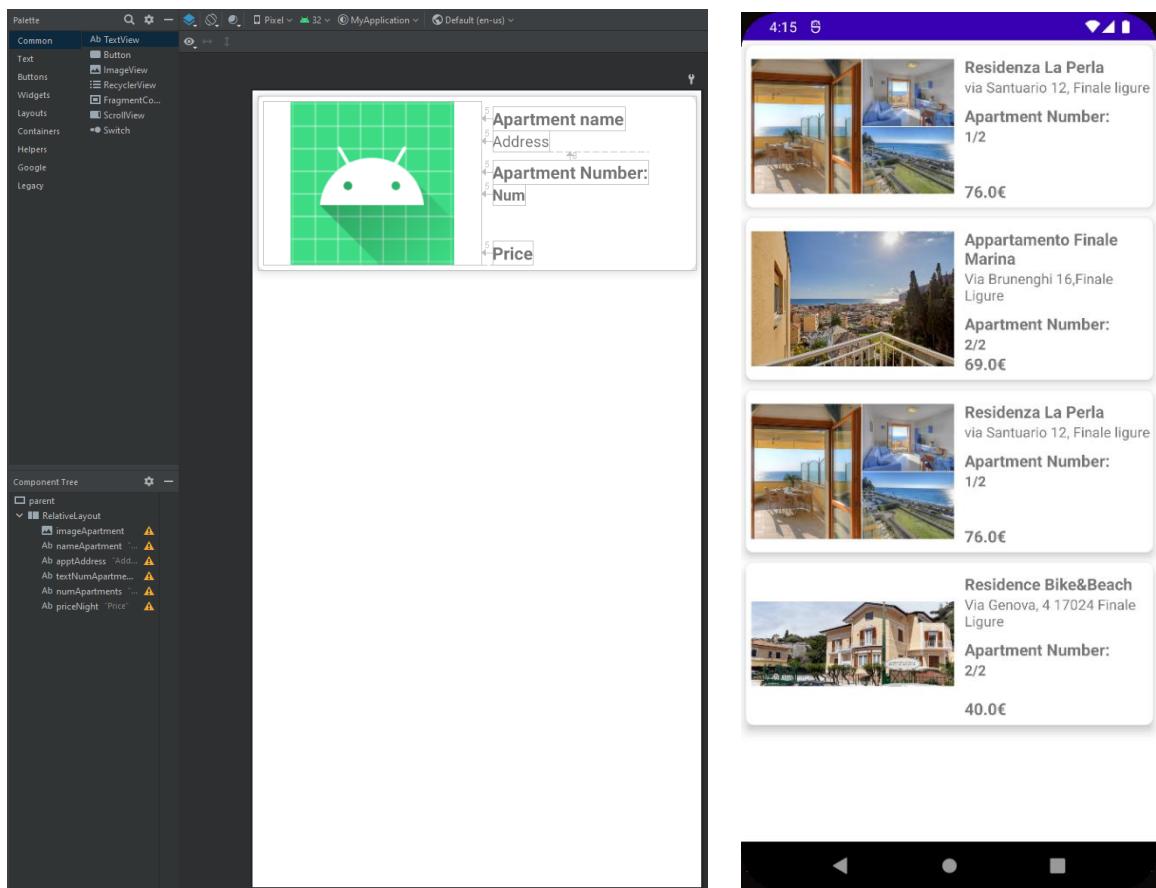


Figura 55: ResultSearchActivity e ApartmentRecViewAdapter versione estesa

ApartmentActivity

Di seguito si riporta un esempio della pagina di un appartamento:

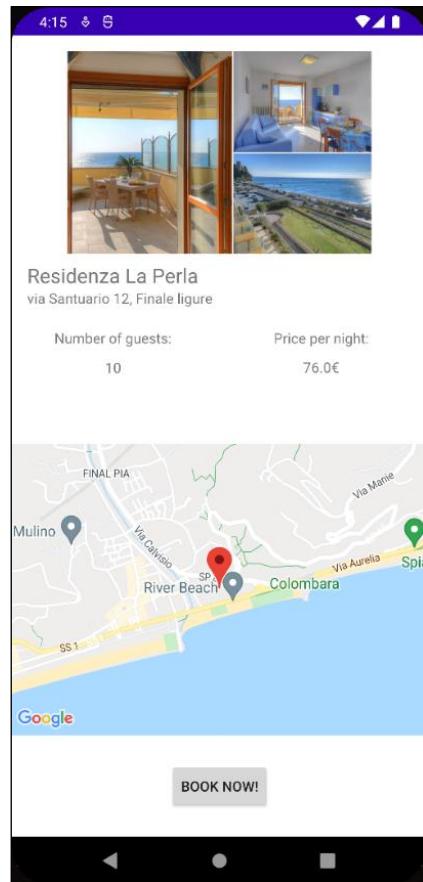


Figura 56: ApartmentActivity

All'interno si trova, oltre ai dettagli dell'appartamento, la mappa navigabile di Google Maps con il marker sulla posizione dell'alloggio.

La mappa è stata integrata grazie all'utilizzo della *MapView* di Android Studio e all'API di Google Maps. Per utilizzarla è stato necessario registrarsi gratuitamente al portale di Google Developers e generare una API Key. Inoltre, è stata implementata la classe *Geocoder* presente in Android Studio per convertire gli indirizzi fisici degli appartamenti in coordinate spaziali necessarie per interagire con Google Maps.

Sul fondo della pagina infine è presente il tasto per registrare la prenotazione. Anche in questo caso viene utilizzata una chiamata Post con i parametri incorporati nel body della chiamata.

5.7 Test e analisi del componente

5.7.1 Analisi statica

Analizzando quanto riportato in *Figura 57* sottostante risulta rispettata la struttura MVP che si era proposto di realizzare.

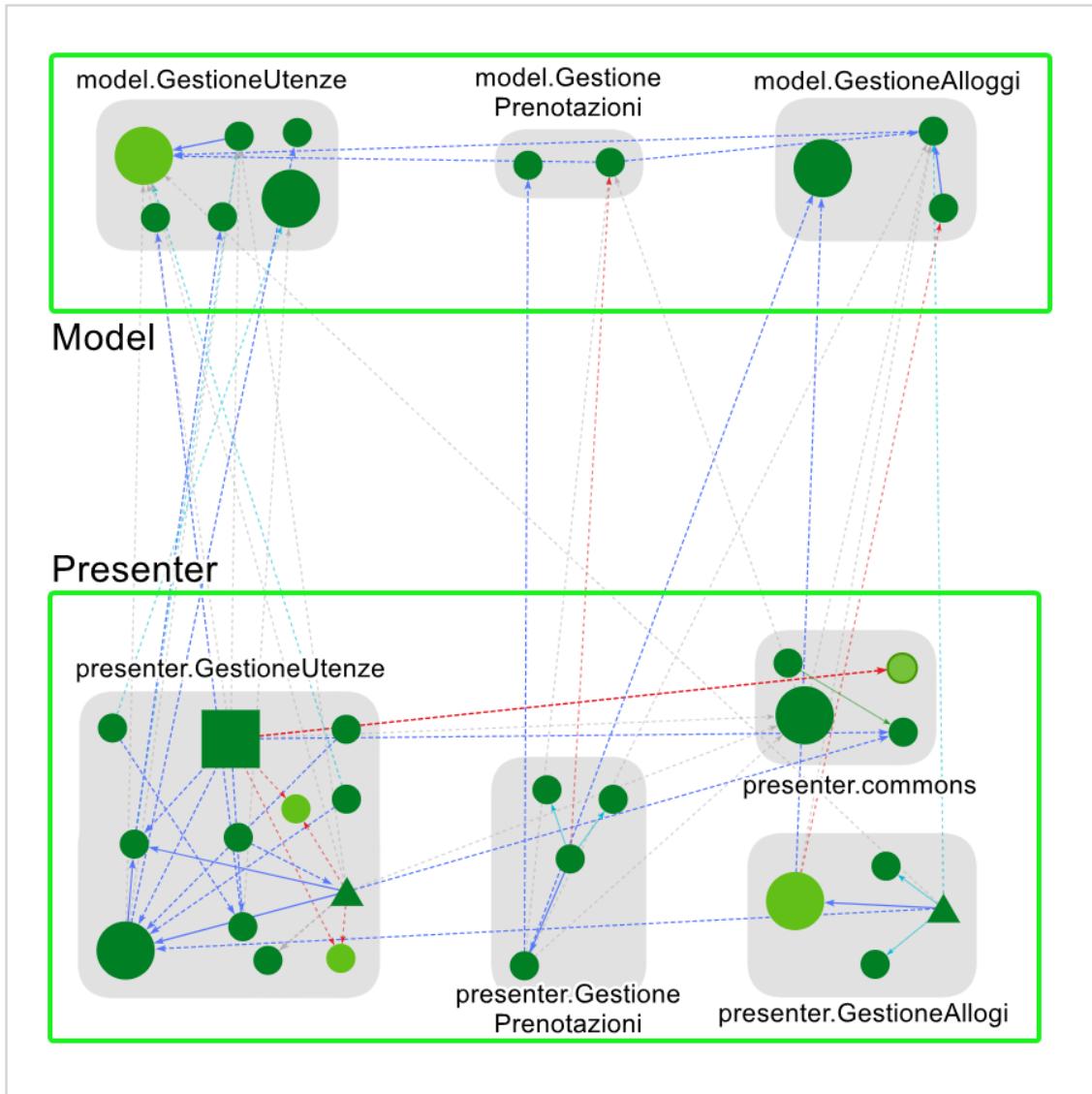


Figura 57: Analisi statica del codice – Iterazione 3

Nello specifico viene mostrato come sono collegati tra loro i vari package e non tutte le classi, per facilitarne la lettura.

Abstractness e Instability

Dai risultati ottenuti dall'analisi effettuata da CodeMR sono emersi i seguenti valori di abstractness e instability:

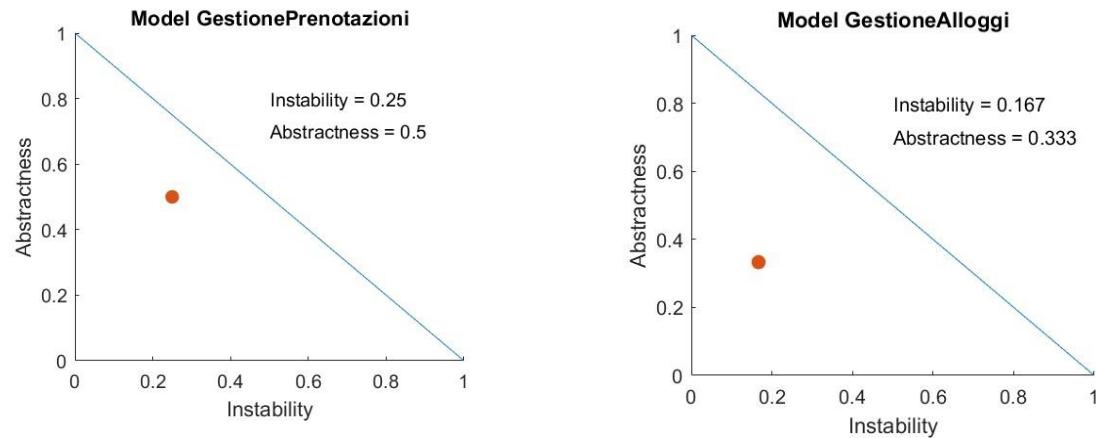


Figura 58: Distance - Model GestionePrenotazioni e GestioneAlloggi

Il package *model.GestionePrenotazioni* assume valori vicini alla retta ideale.

Il package *model.GestioneAlloggi* invece dista maggiormente dalla retta ideale. Questo è dovuto principalmente al fatto che la classe *Apartment* viene referenziata più volte dalle classi presenti in altri package:

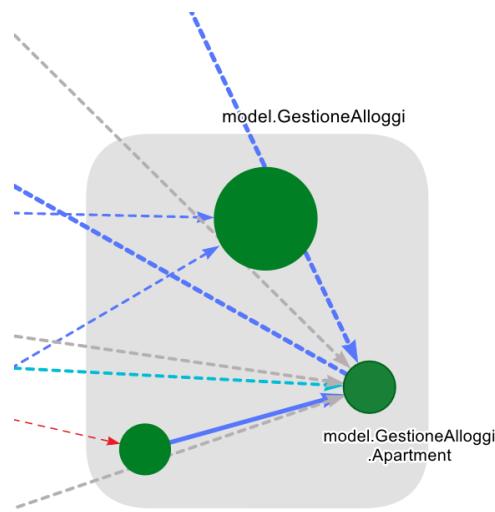


Figura 59: Dettaglio dell'Efferent coupling e Afferent Coupling

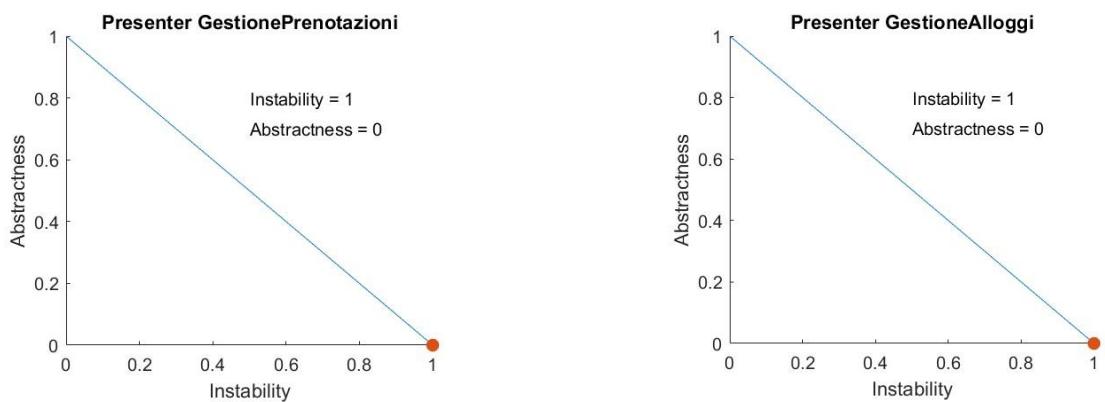


Figura 60: Distance: Presenter GestionePrenotazioni e GestioneAlloggi

Osservando la figura si può affermare che, come nel caso del package *Presenter.GestioneUtenze*, la componente Presenter per entrambi i package ha una forte concretezza e instabilità. Di conseguenza, l'intera componente Presenter dell'applicazione creata ha gli stessi valori di Abstractness e Instability.

Quality Attributes

Di seguito riportiamo gli attributi di Complexity, Coupling, Lack of Cohesion e Size inerenti ai package analizzati nel paragrafo precedente:

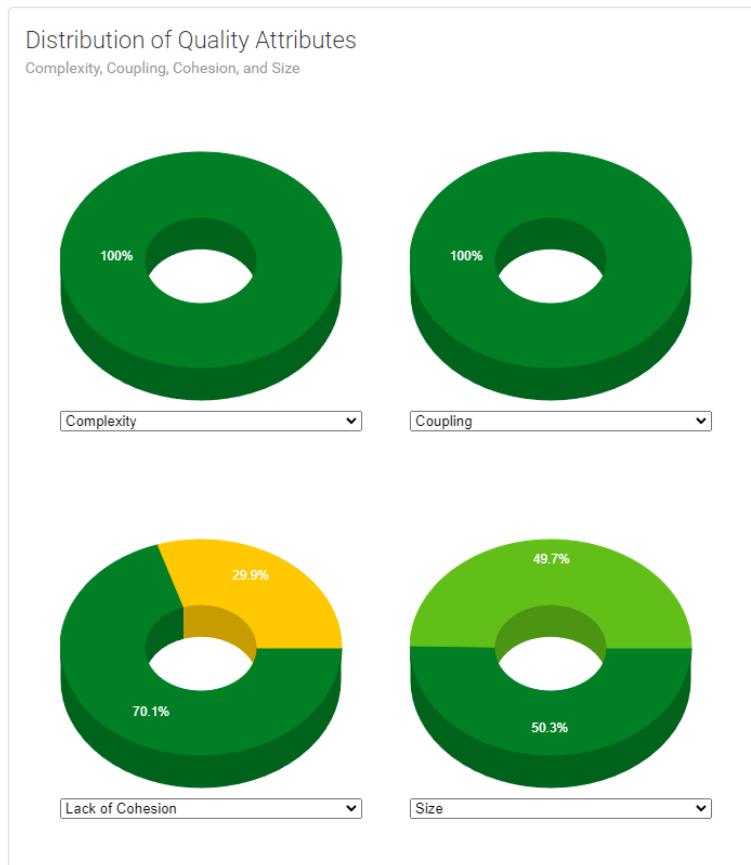


Figura 61: Quality attributes - Model.GestioneAlloggi

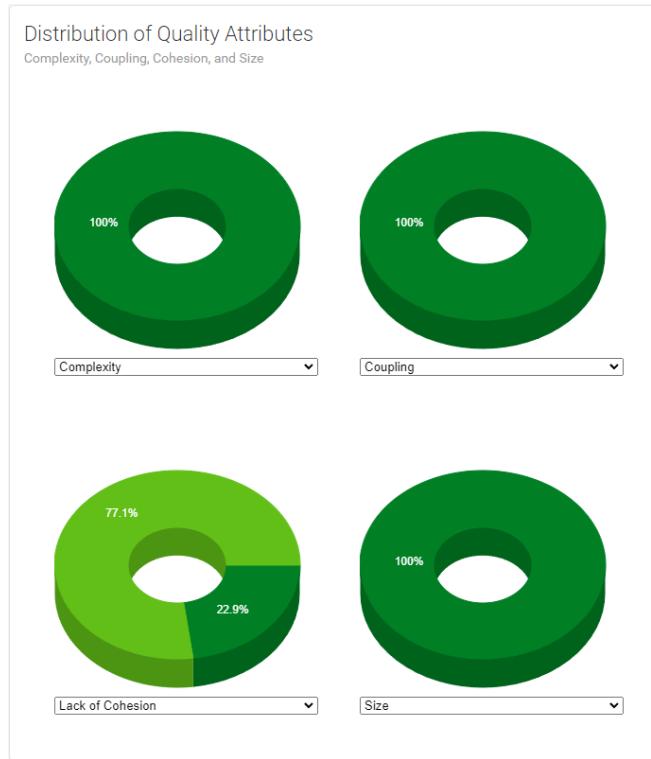


Figura 62: Quality attributes - Model.GestionePrenotazione

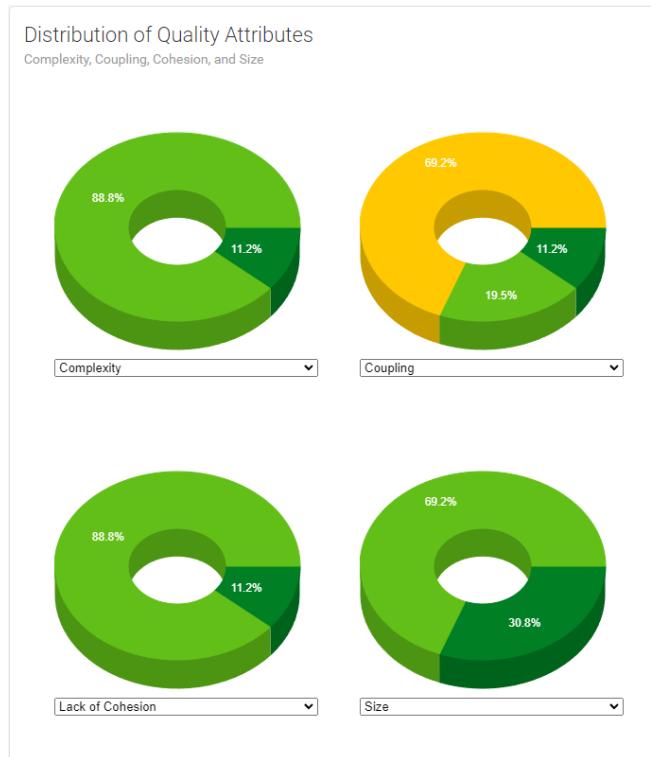


Figura 63: Quality attributes - Presenter.GestioneAlloggi

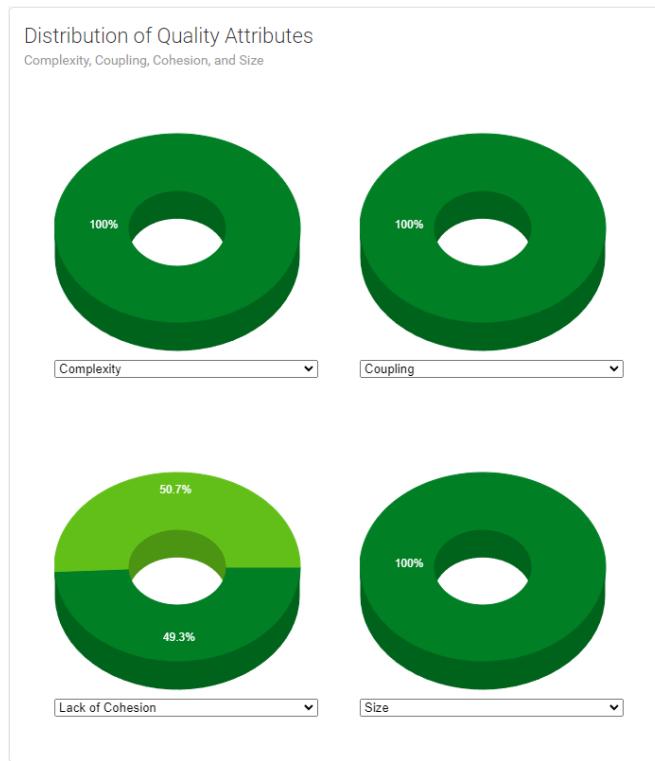


Figura 64: Quality attributes - Presenter.GestionePrenotazione

Come nel paragrafo 4.5.1 sono stati ottenuti dei valori di qualità medio-bassi e, di conseguenza, una buona qualità del codice.

5.7.2 Analisi dinamica

Per garantire la massima copertura possibile del codice implementato in questa fase è stato realizzato un numero esaustivo di casi di test.

In particolare, è stato effettuato il test della seguente funzione:

- UC12: Ricerca alloggi

Per questa funzione verranno mostrati i test delle chiamate API effettuati tramite PostMan e i test del codice di back-end effettuati tramite dei test JUnit.

PostMan

Per testare tramite PostMan sono stati creati due appartamenti locati nella stessa città, in questo caso Trapani, le cui dimensioni disponibili sono rispettivamente di 10 e 8 persone mentre i prezzi per la notte sono rispettivamente di 20 e 25 euro.

La ricerca avviene tramite l'invio di un form Json contenente il numero di ospiti, il luogo dove si desidera soggiornare, la massima distanza dove è possibile trovare il proprio appartamento, una lista di tags, il prezzo massimo che si desidera spendere per ogni notte, le date di inizio e fine del soggiorno ed infine la possibilità di restituire solo singoli appartamenti o anche gruppi di appartamenti (valore 1 per la ricerca multipla, 0 per la ricerca singola).

The screenshot shows the PostMan application interface. At the top, it says "New Collection / Iterazione 3 / filtered request". Below that is a header bar with "POST" selected, "http://localhost:8080/controller/research", and a "Send" button. Underneath the header are tabs for "Params", "Authorization", "Headers (8)", "Body" (which is green and selected), "Pre-request Script", "Tests", and "Settings". The "Body" tab has several options: "none", "form-data", "x-www-form-urlencoded", "raw" (which is red and selected), "binary", and "GraphQL". Below these tabs is a "JSON" dropdown set to "JSON". The main body area contains a code editor with the following JSON content:

```
1
2   ...
3   ...
4   ...
5   ...
6   ...
7   ...
8   ...
9   ...
10  ...
```

The code editor has line numbers from 1 to 10. Lines 1 through 9 are empty. Line 10 ends with a closing brace "}".

Figura 65: Richiesta Json per la ricerca di alloggi

Inviata la request, il risultato atteso è che vengano trovati entrambi gli appartamenti descritti precedentemente poiché rispettano i vincoli precisati dalla richiesta effettuata.

The screenshot shows the Postman application interface. At the top, it displays a POST request to the URL `http://localhost:8080/controller/research`. The status bar indicates a `200 OK` response with a time of `2.10 s` and a size of `736 B`. Below the URL, there are tabs for Params, Authorization, Headers (8), Body (green dot), Pre-request Script, Tests (green dot), and Settings. The **Body** tab is selected, showing a JSON response with two apartment objects. The JSON content is as follows:

```

1  [
2   {
3     "a": {
4       "description": null,
5       "numGuests": 10,
6       "location": "TRAPANI",
7       "tags": "HOT-SEA",
8       "pricePerNight": 25.0,
9       "imageUrl": null,
10      "calendar": {},
11      "id": "5b29a881-9e72-44e8-b9fe-4b3544a800ff"
12    },
13    "order": 1,
14    "total": 2
15  },
16  {
17    "a": {
18      "description": null,
19      "numGuests": 8,
20      "location": "TRAPANI",
21      "tags": "HOT-SEA",
22      "pricePerNight": 20.0,
23      "imageUrl": null,
24      "calendar": {},
25      "id": "4dea881c-03f1-428b-bcd0-659b265c6242"
26    },
27    "order": 2,
28    "total": 2
29  }
30 ]

```

Figura 66: Test ricerca multipla tramite PostMan

Per la ricerca singola (*multipleApartments = 0*) si mostrano anche le casistiche in cui la prenotazione per quel giorno è già presente oppure quando il numero di persone che cercano un appartamento è superiore alla capienza disponibile dall'appartamento oppure quando i limiti di spesa per la notte sono inferiori alla richiesta dell'appartamento:

- Caso base dove vengono rispettati tutti i criteri:

The screenshot shows the PostMan interface for a search request. The URL is `http://localhost:8080/controller/research`. The request method is POST. The Body tab contains the following JSON payload:

```

1  {
2     "numGuests": 8,
3     "location": "TRAPANI",
4     "maxDistance": 10,
5     "tags": "HOT-SEA",
6     "maxPricePerNight": 80.00,
7     "prenotationStart": "2022-09-01",
8     "prenotationEnd": "2022-09-09",
9     "multipleApartments": 0
10 }

```

The response status is 200 OK, with a time of 994 ms and a size of 684 B. The response body is displayed in JSON format:

```

1  [
2     {
3         "description": null,
4         "numGuests": 10,
5         "location": "TRAPANI",
6         "tags": "HOT-SEA",
7         "pricePerNight": 25.0,
8         "imageUrl": null,
9         "calendar": {},
10        "id": "6b29a881-9e72-44e8-b9fe-4b3544a500ff"
11    },
12    {
13    }
14 ]

```

Figura 67: Test ricerca singola tramite PostMan

- Caso in cui il prezzo è inferiore alla richiesta:

The screenshot shows the PostMan interface for a search request with a lower price limit. The URL is `http://localhost:8080/controller/research`. The request method is POST. The Body tab contains the following JSON payload:

```

1  {
2     "numGuests": 8,
3     "location": "TRAPANI",
4     "maxDistance": 10,
5     "tags": "HOT-SEA",
6     "maxPricePerNight": 10.00,
7     "prenotationStart": "2022-09-01",
8     "prenotationEnd": "2022-09-09",
9     "multipleApartments": 0
10 }

```

The response status is 200 OK, with a time of 28 ms and a size of 348 B. The response body is displayed in JSON format:

```

1  [
2     {}
1 ]

```

Figura 68: Test ricerca singola – prezzo inferiore

- Caso in cui è già presente una prenotazione per quella data

The screenshot shows a Postman collection named "New Collection / Iterazione 3 / filtered request". A POST request is made to "http://localhost:8080/controller/research". The "Body" tab is selected, showing the following JSON payload:

```

1  [
2   ...
3     "numGuests":8,
4     "location":"TRAPANI",
5     "maxDistance":0,
6     "tags":"HOT-SEA",
7     "maxPricePerNight":80.00,
8     "prenotationStart":"2022-09-01",
9     "prenotationEnd":"2022-09-09",
10    "multipleApartments":0
11  ]

```

The response status is 200 OK, with a response body containing a single object:

```

1  [
2   {
3     "description": null,
4     "numGuests": 10,
5     "location": "TRAPANI",
6     "tags": "HOT-SEA",
7     "pricePerNight": 25.0,
8     "imageUrl": null,
9     "calendax": {},
10    "id": "5b29a881-9e72-44e8-b9fe-4b3544a500ff"
11  }
12 ]

```

Figura 69: Test ricerca singola - date occupate

(corretto perché il risultato è l'altro appartamento presente a Trapani)

- Caso in cui le persone superano la capienza dell'appartamento

The screenshot shows a Postman collection named "New Collection / Iterazione 3 / filtered request". A POST request is made to "http://localhost:8080/controller/research". The "Body" tab is selected, showing the following JSON payload:

```

1  [
2   ...
3     "numGuests":20,
4     "location":"TRAPANI",
5     "maxDistance":0,
6     "tags":"HOT-SEA",
7     "maxPricePerNight":80.00,
8     "prenotationStart":"2022-08-01",
9     "prenotationEnd":"2022-08-09",
10    "multipleApartments":0
11  ]

```

The response status is 200 OK, with an empty response body:

```

1  []

```

Figura 70: Test ricerca singola - numero eccessivo di persone

JUnit test

Per quanto riguarda i test di JUnit è stata utilizzata la tecnica del mock per chiamare effettivamente i metodi senza che essi però si connettano al DB, infatti, siamo noi in base al test che si vuole effettuare a decidere cosa restituire ai metodi chiamati (sono sempre coerenti con il tipo restituito dai metodi).

In generale il mocking è un processo utilizzato negli unit test quando l'unità sottoposta a test ha dipendenze esterne. Lo scopo del mocking è isolare e concentrarsi sul codice in fase di test e non sul comportamento o sullo stato delle dipendenze esterne.

In particolare, con la funzione `when(oggetto mock).thenReturn()` si simula la chiamata all'oggetto mockato e tramite il `.thenReturn()` si impone l'oggetto da ritornare quando viene invocato il metodo sull'oggetto mocked.

Il controllo del test avviene verificando tramite la funzione `verify()` che ci sia stata almeno un'interazione con il metodo dell'oggetto specificato e questo viene indicato tramite la funzione `times(1)`.

Di seguito viene mostrato un unit test sulla ricerca multipla dove avviene la restituzione corretta dei due appartamenti inseriti:

```

    @Test
    void multipleResearchTest1() throws Exception {
        int numGuests = 10;
        String location = "Dalmine";
        Date startReservation=Date.valueOf(LocalDate.of(2022, 10, 10));
        Date endReservation=Date.valueOf(LocalDate.of(2022, 10, 20));
        int maxDistance = 100;
        String tags = "CITY";
        double maxPricePerNight = 25.5;
        List<Apartment> result = new ArrayList<>();
        Apartment a1 = new Apartment(
                "",
                10,
                "Dalmine",
                "CITY",
                25.5
        );
        Apartment a2 = apartmentBuilder2();
        result.add(a1);
        result.add(a2);
        when(apartmentRepository.findAvailableApartmentsPerMaxNumGuestsAndMaxPricePerNight(
                startReservation,
                endReservation,
                numGuests,
                maxPricePerNight)).thenReturn(result);
        serviceToTest.multipleResearch(numGuests, location, maxDistance, tags, maxPricePerNight, startReservation, endReservation);
        verify(apartmentRepository, times(1)).findAvailableApartmentsPerMaxNumGuestsAndMaxPricePerNight(any(), any(), eq(numGuests), eq(25.5));
    }
}

```

Figura 71: Esempio di test del metodo multipleResearch()

Ed il risultato del test è:

▼	✓	Test Results	2 sec 112 ms
▼	✓	ApartmentServiceTest	2 sec 112 ms
	✓	multipleResearchTest10	2 sec 112 ms

Figura 72: Risultato del test sul metodo multipleResearch()

6 Conclusiones

Al termine dell'iterazione tre abbiamo ottenuto come risultato un'applicazione in grado di implementare i seguenti casi d'uso:

- UC1: Registrazione utente
- UC2: Log-in nell'applicazione
- UC3: Log-out dall'applicazione
- UC11: Visualizzazione scheda alloggio
- UC12: Ricerca alloggi
- UC13: Inserimento prenotazione

La stesura del codice necessario per implementare questi casi d'uso ha posto l'intero team davanti ad una stimolante sfida che ci ha permesso di conoscere applicativi, librerie e metodi di lavoro che hanno migliorato le skills e le competenze di ciascuno di noi.

In particolare, il metodo di lavoro agile è stato l'elemento chiave per accrescere le proprie abilità nel lavoro di gruppo e nell'affrontare gli imprevisti che si sono presentati durante le varie iterazioni.

In conclusione, siamo riusciti a:

- Realizzare il log-in e log-out ad un'applicazione grazie all'implementazione delle librerie di *Spring-Security* e *Spring-Mail* e alla creazione di un *SessionManager* in Android Studio.
- Implementare un form di registrazione semplice ed intuitivo da inoltrare al nostro Back-End e Database SQL usufruendo anche dei link di attivazione messi a disposizione da *Spring-Mail*.
- Effettuare una ricerca che prendesse in considerazione più filtri, implementando un algoritmo di programmazione avanzato e integrando i servizi di geolocalizzazione di *GoogleMapsServices*.
- Realizzare una scheda di presentazione di un alloggio che comprendesse le informazioni principali, la possibilità di effettuare una prenotazione e una vista della mappa di Google Maps con un marker per ottenere le indicazioni stradali verso l'alloggio.

Il tutto è stato meticolosamente testato con metodologie di analisi statica e dinamica. I risultati sono stati riportati nelle apposite sezioni presenti nelle iterazioni di sviluppo.

La copertura dinamica dei servizi ottenuta tramite JUnit test viene riportata in *Figura 73*:

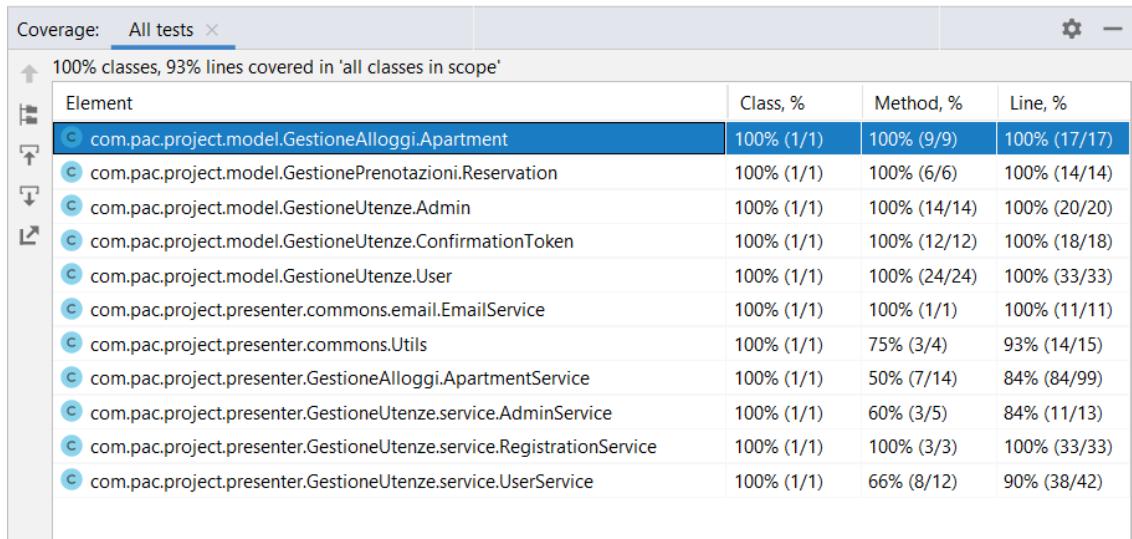


Figura 73: Copertura JUnit tests

Di seguito allegiamo i grafici contenenti la valutazione globale dell'analisi statica ottenuta da CodeMR:

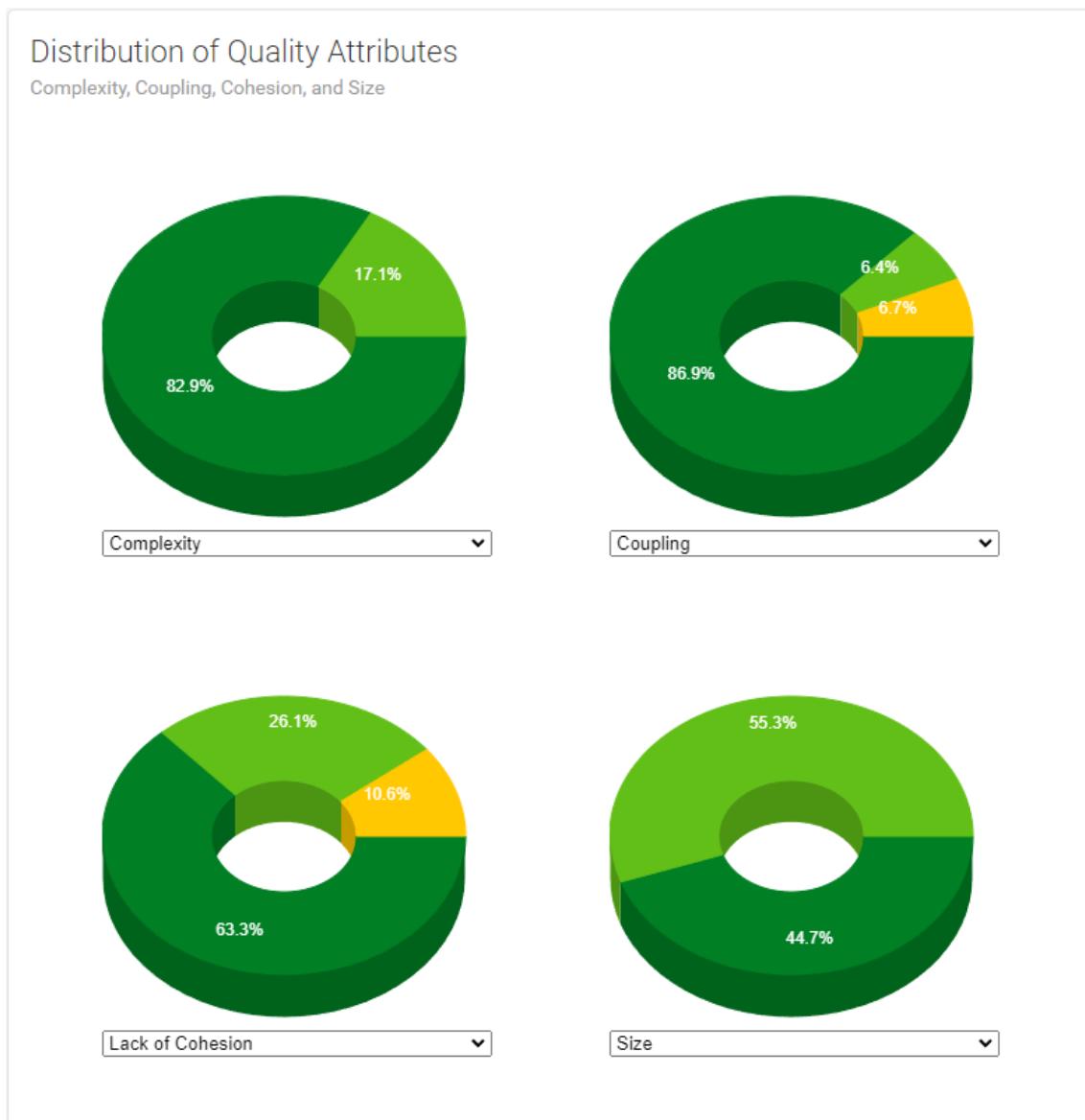


Figura 74: Quality attributes – Overall project

C3: Parametro associato ai seguenti valori: Coupling, Cohesion, Complexity. Rappresenta il massimo valore tra le misure di Coupling, Cohesion, Complexity.

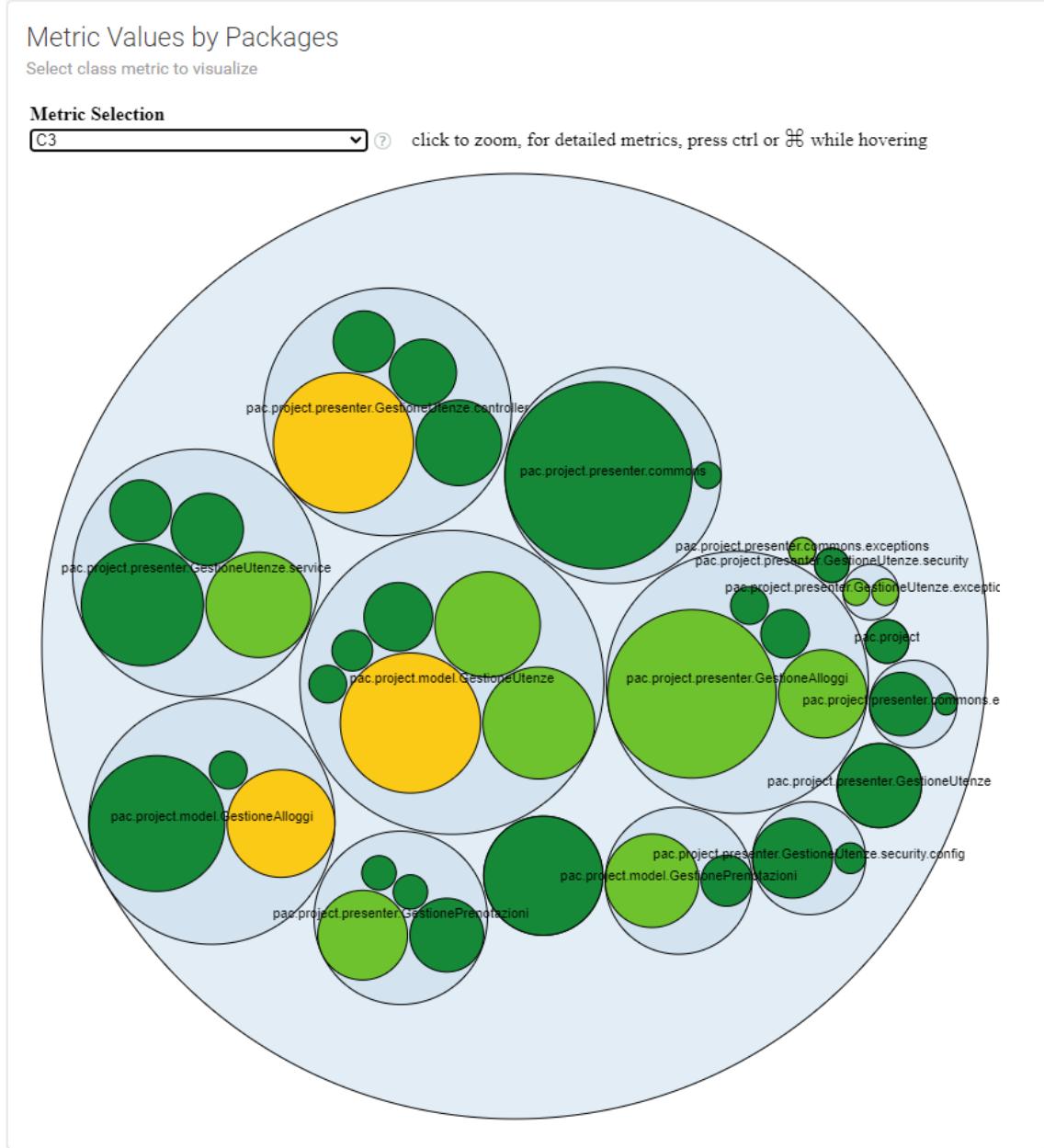


Figura 75: Metric Values by Package – C3

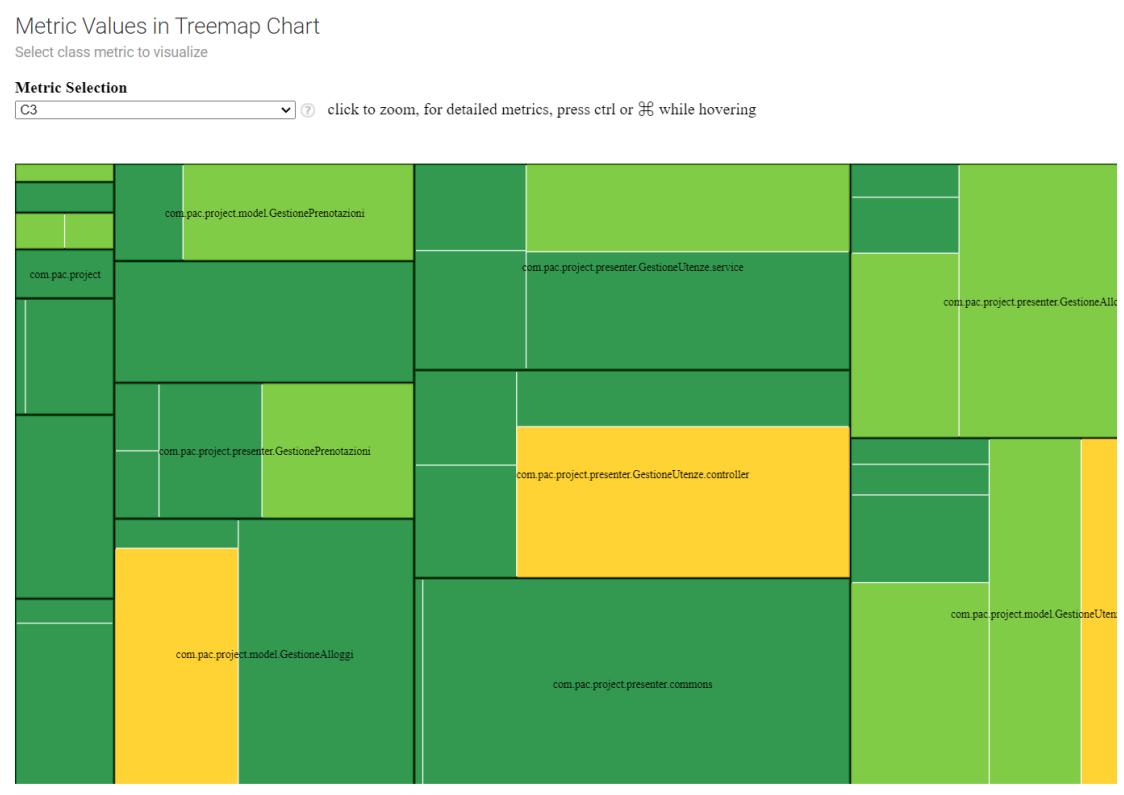


Figura 76: Map chart by C3 value

7 Guida all'installazione e all'uso

Per poter utilizzare e testare l'applicazione è necessario installare gli IDE dedicati a back-end e front-end. Consigliamo di utilizzare:

- IntelliJ IDEA per il back-end, per avere piena compatibilità con l'ultima versione del codice e i plugin utilizzati.
- Android Studio per il front-end, in quanto direttamente lasciato da Google e integrando nativamente uno ambiente di virtualizzazione per dispositivi android.

Una volta installati i due IDE è possibile scaricare il codice presente nelle repositories dedicate al progetto dai seguenti link:

- Back-end:

https://github.com/GMaffio99/PAC_Project_EasyBooking/tree/main/Codice_BackEnd

- Front-end:

https://github.com/GMaffio99/PAC_Project_EasyBooking/tree/main/Codice_FrontEnd

Per poter utilizzare l'applicazione Android sarà necessario:

1. Eseguire il codice del back-end.
2. Avere una connessione internet che permetta la comunicazione tra il back-end e il database online.
3. Creare un dispositivo android virtuale con installati i play services (altrimenti non sarà possibile usufruire dell'integrazione di Google Maps) oppure collegare un dispositivo android in modalità sviluppatore.
4. Modificare il parametro “baseUrl” presente nel codice inserendo l'indirizzo IP della propria scheda di rete.
5. Accertarsi che i link di attivazione/reset password vengano usati sull'host in cui è in esecuzione il server back-end.

Creazione di un dispositivo android virtuale

Per prima cosa è necessario andare sotto la sezione “Tools” per aprire l’AVD Manager:

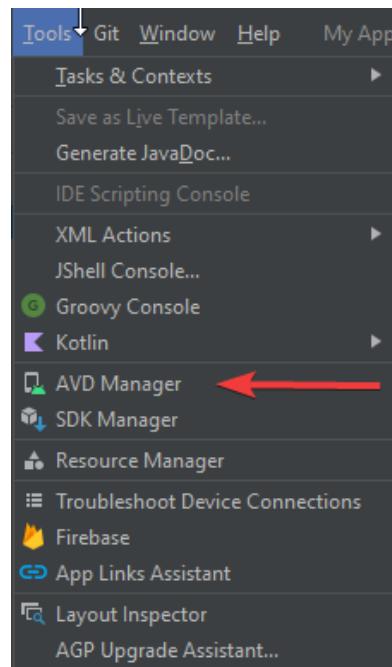


Figura 77: Guida alla creazione di un dispositivo virtuale – Parte 1

In seguito, cliccate su “Create Virtual Device” in basso a sinistra:

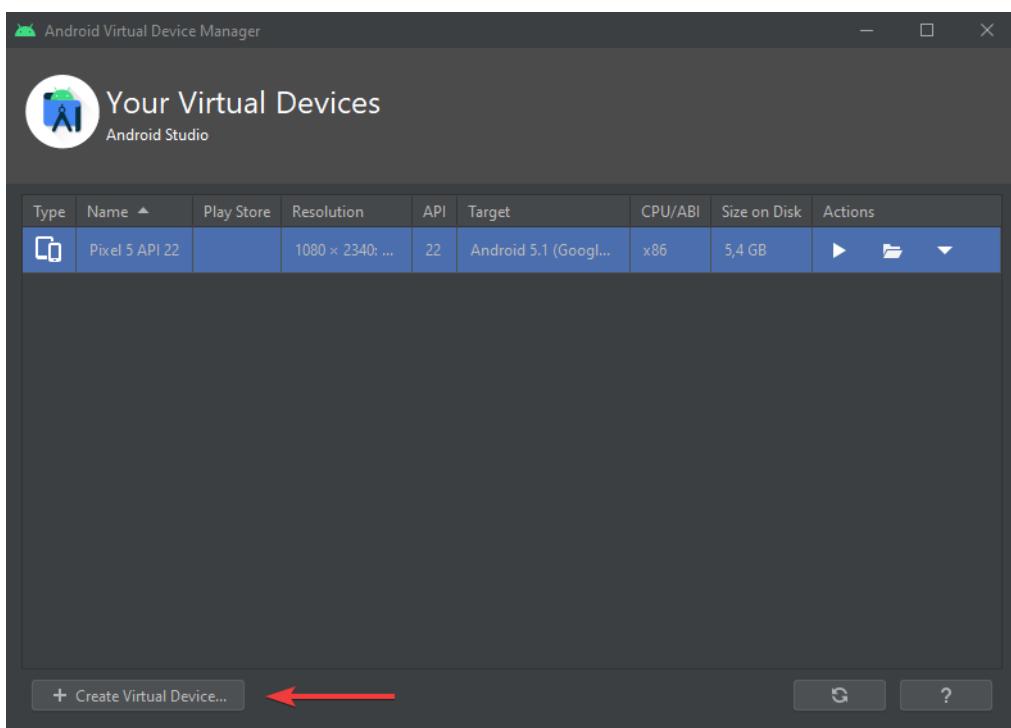


Figura 78: Guida alla creazione di un dispositivo virtuale - Parte 2

Selezzionate il device da virtualizzare:

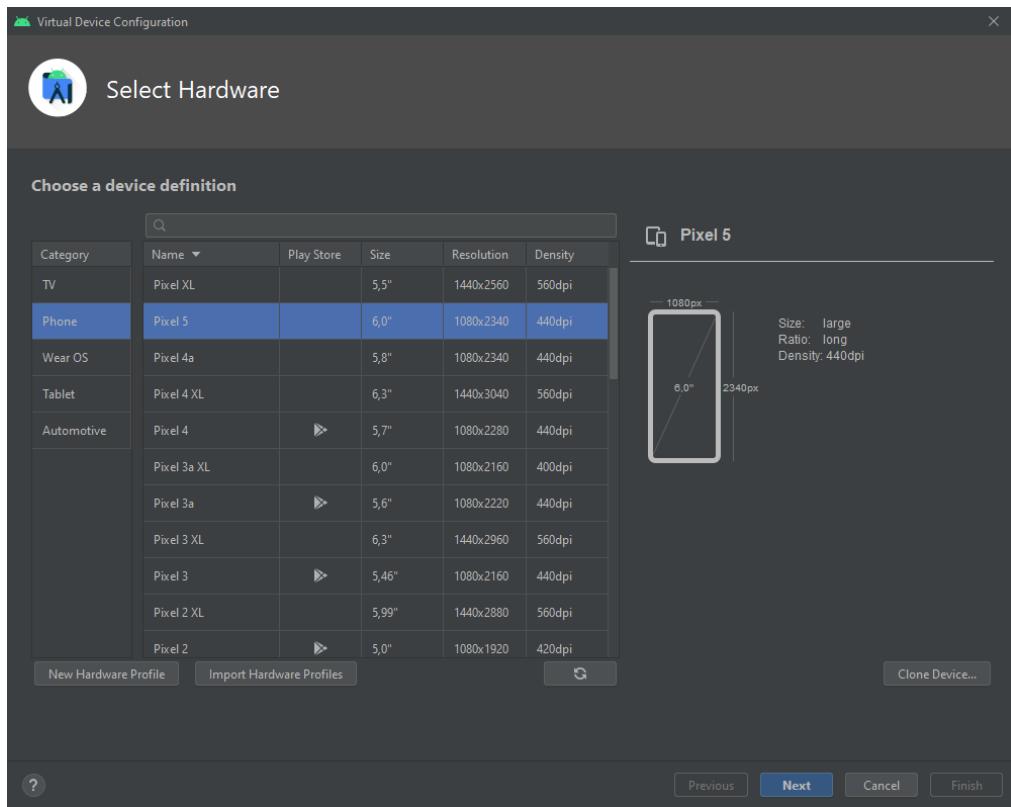


Figura 79: Guida alla creazione di un dispositivo virtuale - Parte 3

Di default sono disponibili i dispositivi prodotti direttamente da Google, tuttavia, è possibile recuperare su internet anche altri hardware profiles di altri produttori.

A questo punto è necessario scegliere e scaricare la versione di android che volete installare sul virtual device.

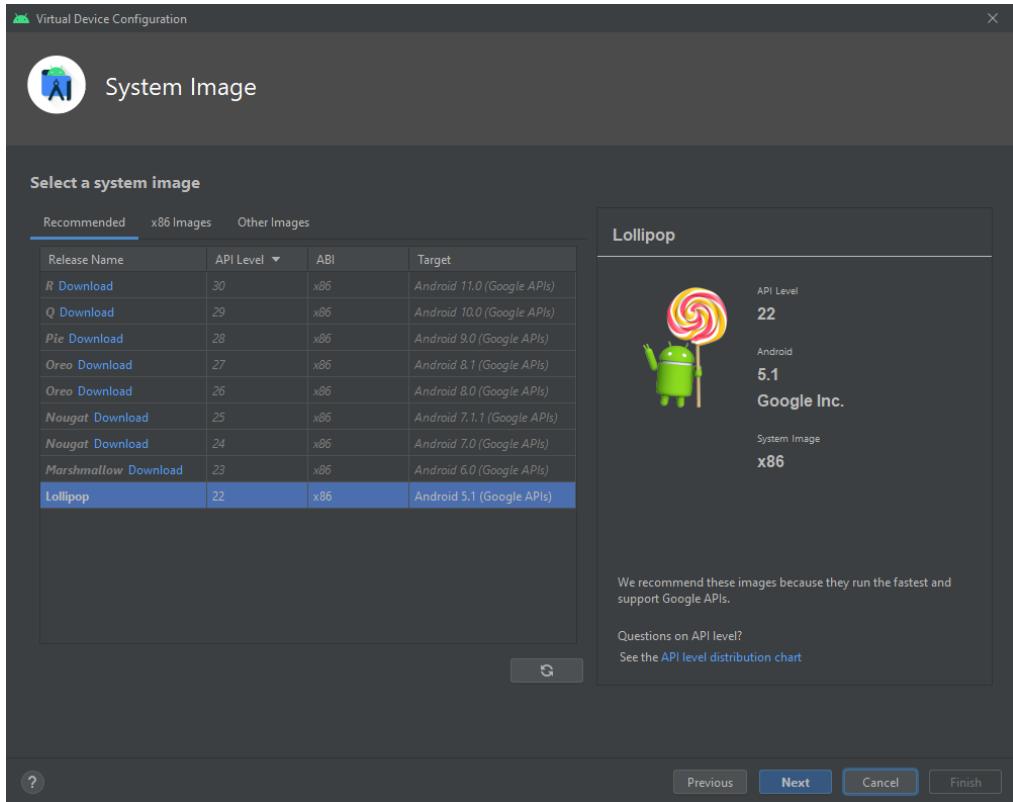


Figura 80: Guida alla creazione di un dispositivo virtuale - Parte 4

Di norma è consigliato utilizzare la versione meno recente tra quelle consigliate per garantire il funzionamento su più dispositivi possibili. Noi abbiamo optato per la versione 10 di Android.

Scegliete gli ultimi dettagli e fate partire il deploy del device.

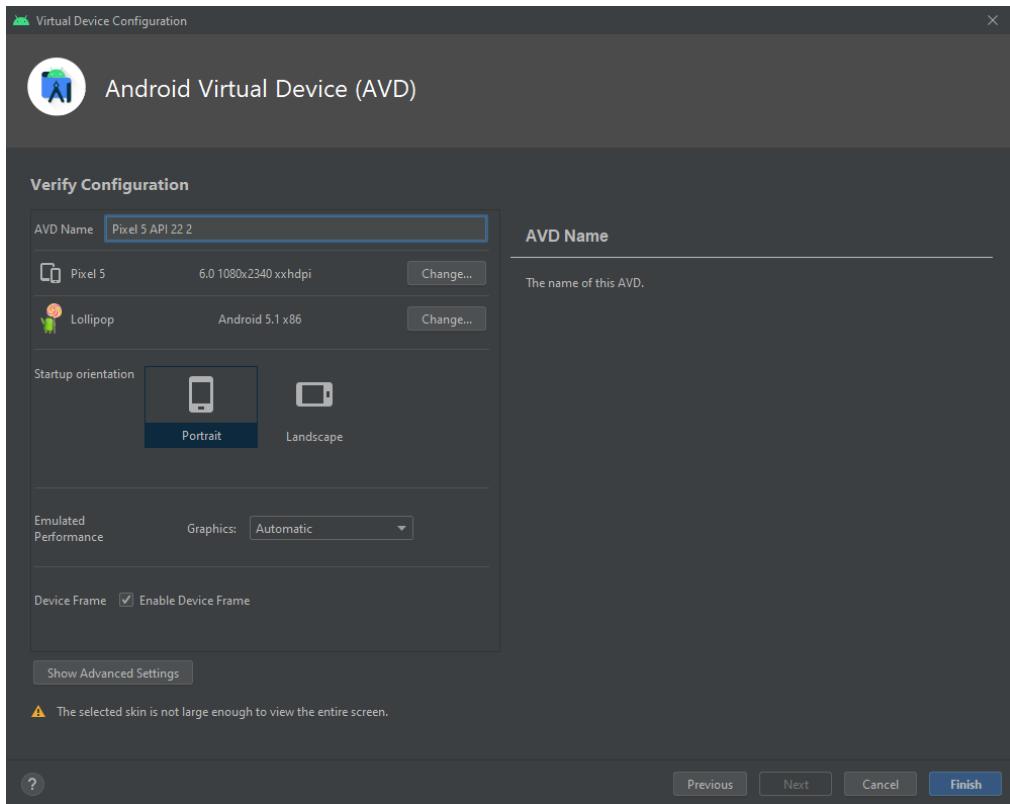


Figura 81: Guida alla creazione di un dispositivo virtuale - Parte 5

Per poter utilizzare questa funzione di android è importante abilitare la virtualizzazione del processore nelle impostazioni del sistema o nel bios della scheda madre.

Di seguito una guida dettagliata che copre le maggior parte dei produttori di hardware:

<https://www.aranzulla.it/come-attivare-la-virtualizzazione-nel-bios-1231556.html>

Modificare il parametro “baseUrl”

L’applicazione android effettua delle chiamate al server back-end che è in esecuzione in locale sulla stessa macchina in cui viene eseguito il codice del frontend. Per questo motivo le chiamate HTTP vengono effettuate in localhost.

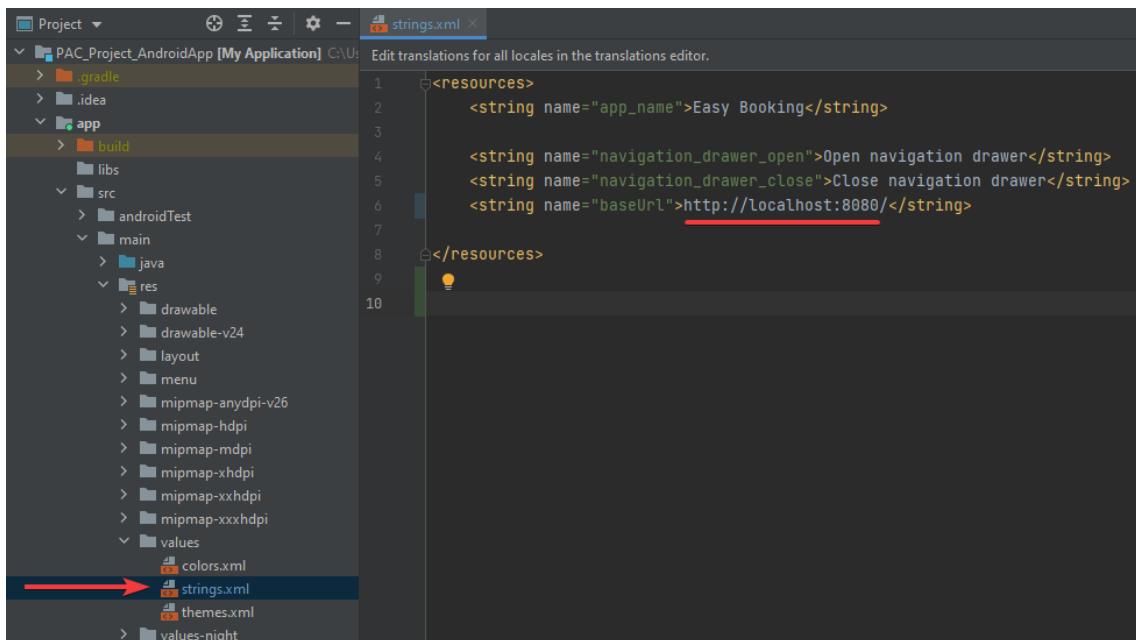


Figura 82: Modifica parametro baseUrl - Parte 1

In alcuni casi il sistema non è in grado di interpretare correttamente la chiamata a localhost ma richiede che venga inserito l’IP privato del pc.

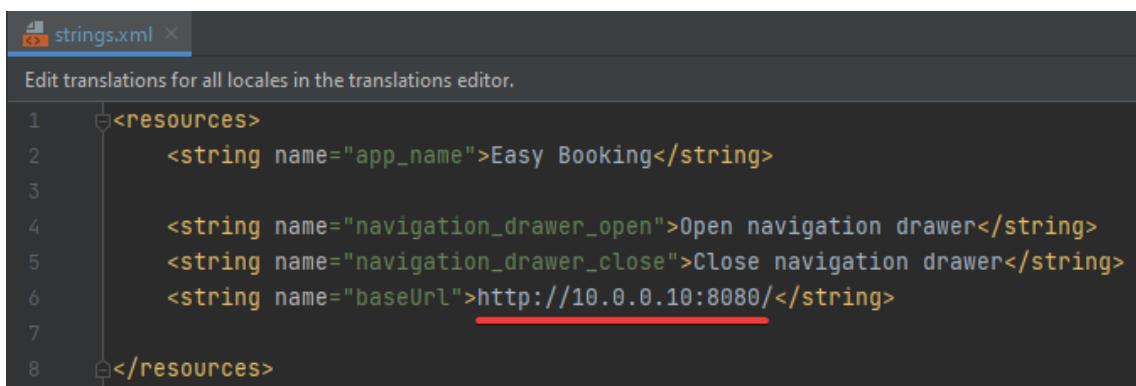


Figura 83: Modifica parametro baseUrl - Parte 2

Per recuperare questo valore è sufficiente guardare nella sezione proprietà della propria scheda di rete.

Esempio d'uso

Di seguito riportiamo alcuni parametri con i quali è possibile testare l'applicazione con i relativi risultati.

Login

È possibile effettuare il login dell'utenza con le seguenti credenziali:

E-mail: progetto.pac2022@gmail.com

Password: Universita2022!

Ricerca del singolo alloggio

Ricerca standard:

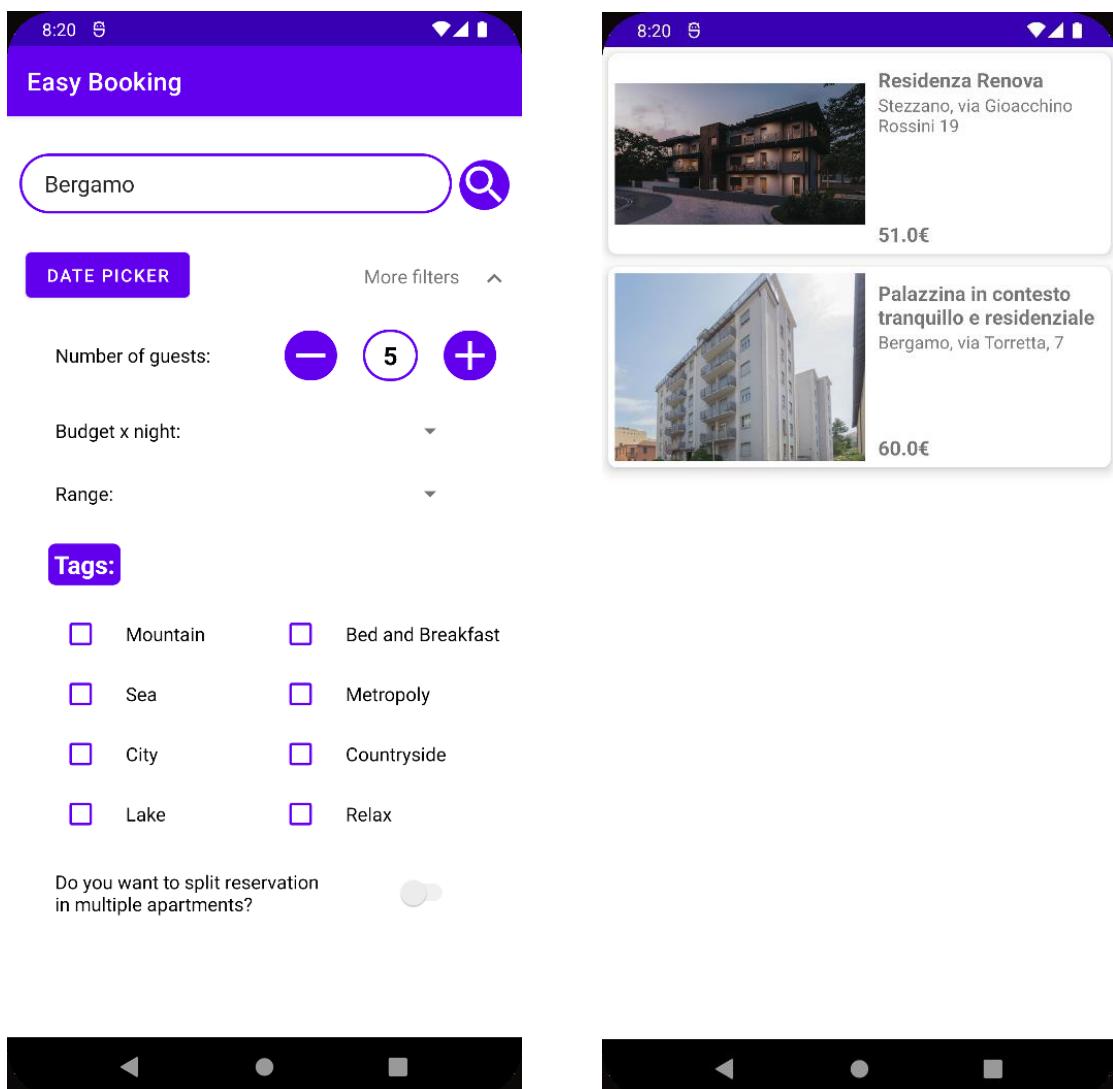


Figura 84: Esempio ricerca standard

Ricerca avanzata:

The screenshot shows the 'Easy Booking' mobile application interface. At the top left is the time '8:21'. On the right, there are signal and battery icons. The title 'Easy Booking' is at the top center. Below it is a search bar with the text 'Bergamo' and a magnifying glass icon. To the right of the search bar is a date picker button labeled 'DATE PICKER' and a 'More filters' button with a dropdown arrow.

Below the search bar, the text 'Number of guests:' is followed by a minus button, a circular button with '12' (which is highlighted with a purple border), and a plus button.

Further down, the text 'Budget x night:' is followed by a dropdown menu set to '< 75€'. Below it, the text 'Range:' is followed by a dropdown menu set to '< 5Km'.

A section titled 'Tags:' contains several checkbox options. Some are checked (indicated by a purple checkmark) and others are not. The checked tags are: 'Bed and Breakfast', 'City', and 'Relax'. The unchecked tags are: 'Mountain', 'Sea', 'Metropoly', 'Countryside', and 'Lake'.

At the bottom, the text 'Do you want to split reservation in multiple apartments?' is followed by a toggle switch that is currently off (grey).

At the very bottom, there are two black navigation bars with white icons for back, forward, and other functions.

Figura 85: Esempio ricerca avanzata

Ricerca di un gruppo di alloggi

The screenshot displays the 'Easy Booking' mobile application interface. At the top left is the app's logo and time (8:23). On the right are standard Android navigation icons: back, home, and recent apps.

The main screen features a search bar with the placeholder 'Finale Ligure' and a magnifying glass icon. Below the search bar are several filter options:

- DATE PICKER**: A button to open a date selection calendar.
- More filters**: A button to access additional filtering options.
- Number of guests:** Set to 15, indicated by a blue circular button with the number '15'.
- Budget x night:** Set to '100€+'.
- Range:** Set to '10Km+'.
- Tags:** A section with checkboxes for various travel preferences:
 - Mountain
 - Bed and Breakfast
 - Sea
 - Metropoly
 - City
 - Countryside
 - Lake
 - Relax
- Do you want to split reservation in multiple apartments?**: A toggle switch set to 'On'.

On the right side of the screen, there is a vertical list of four accommodation options, each displayed in a card format with a small image, title, address, apartment number, and price:

- Residenza La Perla**, via Santuario 12, Finale Ligure. Apartment Number: 1/2, Price: 76.0€. Includes three small images: an interior room, an exterior balcony, and a scenic view of the sea.
- Appartamento Finale Marina**, Via Brunenghi 16, Finale Ligure. Apartment Number: 2/2, Price: 69.0€. Includes a large image showing a balcony with a view of the town and sea.
- Residenza La Perla**, via Santuario 12, Finale Ligure. Apartment Number: 1/2, Price: 76.0€. Includes three small images: an interior room, an exterior balcony, and a scenic view of the sea.
- Residence Bike&Beach**, Via Genova, 4 17024 Finale Ligure. Apartment Number: 2/2, Price: 40.0€. Includes a large image showing a modern building complex.

At the bottom of the screen are standard Android navigation icons: back, home, and recent apps.

Figura 86: Esempio ricerca per gruppi di alloggi