

# Documentazione progetto

Giulia Sonzogni e Nunzio Marco Bisceglia

Matricole n. 1045979 e 1046319



Università degli Studi di Bergamo

Laurea Magistrale in Ingegneria Informatica

Corso di Informatica III

Modulo di Progettazione e Algoritmi (6 CFU)

Ottobre 2021

# Indice

<b>1</b>	<b>Iterazione 0</b>	<b>5</b>
1.1	Introduzione e panoramica del sistema . . . . .	5
1.2	Requisiti funzionali e analisi dei casi d'uso . . . . .	6
1.2.1	Coda ad alta priorità . . . . .	7
1.2.2	Coda a media priorità . . . . .	7
1.2.3	Coda a bassa priorità . . . . .	7
1.3	Requisiti non funzionali . . . . .	8
1.3.1	Manutenibilità . . . . .	8
1.3.2	Efficienza . . . . .	8
1.3.3	Usabilità . . . . .	8
1.4	Topologia del sistema . . . . .	9
1.5	Design pattern MVP . . . . .	9
1.6	Toolchain e tecnologie . . . . .	12
1.6.1	Elenco completo . . . . .	12
1.6.2	Tecnologie per analisi statica . . . . .	12
1.6.3	Tecnologie per analisi dinamica . . . . .	12
<b>2</b>	<b>Iterazione 1</b>	<b>15</b>
2.1	Introduzione . . . . .	15
2.2	UC1: Gestione barche . . . . .	16
2.2.1	UC1.1 Visualizzazione barche . . . . .	16
2.2.2	UC1.2 Inserimento barca . . . . .	17
2.2.3	UC1.3 Modifica barca . . . . .	18
2.2.4	UC1.4 Eliminazione barca . . . . .	18
2.3	UC2: Gestione escursioni . . . . .	20
2.3.1	UC2.1 Visualizzazione escursioni . . . . .	20
2.3.2	UC2.2 Inserimento escursione . . . . .	21
2.3.3	UC2.3 Modifica escursione . . . . .	21
2.3.4	UC2.4 Eliminazione escursione . . . . .	22
2.4	UC3: Registrazione dell'utente . . . . .	23
2.5	UC4: Login dell'utente . . . . .	24
2.6	UC5: Logout dell'utente . . . . .	24
2.7	UML Component Diagram . . . . .	25
2.8	UML Class Diagram per interfacce . . . . .	26
2.9	UML Class Diagram per tipi di dato . . . . .	27
2.10	UML Deployment Diagram . . . . .	28
2.11	Testing . . . . .	29
2.11.1	Analisi statica . . . . .	29

2.11.2	Analisi dinamica . . . . .	29
2.11.3	Unit Test . . . . .	30
2.12	Documentazione API . . . . .	31
<b>3</b>	<b>Iterazione 2</b>	<b>35</b>
3.1	Introduzione . . . . .	35
3.2	UC6: Gestione prenotazioni . . . . .	35
3.2.1	UC6.1 Nuova prenotazione . . . . .	35
3.2.2	UC6.2 Visualizzazione prenotazioni . . . . .	36
3.3	UC7: Algoritmo barche ed escursioni . . . . .	37
3.4	UML Component Diagram . . . . .	42
3.5	UML Class Diagram per interfacce . . . . .	43
3.6	UML Class Diagram per tipi di dato . . . . .	44
3.7	UML Deployment Diagram . . . . .	45
3.8	Testing . . . . .	46
3.8.1	Analisi statica . . . . .	46
3.8.2	Analisi dinamica . . . . .	46
3.8.3	Unit Test . . . . .	46
3.8.4	Integration Test dell'algoritmo . . . . .	49
3.9	Documentazione API . . . . .	53
<b>4</b>	<b>Guida all'installazione</b>	<b>55</b>
4.1	Installazione Android Studio . . . . .	55
4.2	Installazione Visual Studio Code . . . . .	64

## Elenco delle figure

1	Diagramma UML dei casi d'uso . . . . .	6
2	Model View Presenter . . . . .	10
3	Topologia del sistema . . . . .	11
4	UML Component Diagram . . . . .	25
5	UML Class Diagram per interfacce . . . . .	26
6	UML Class Diagram per tipi di dato . . . . .	27
7	UML Deployment Diagram . . . . .	28
8	Risultati test dinamico . . . . .	29
9	Risultato Unit Test . . . . .	30
10	API per la creazione di una barca . . . . .	31
11	API per la creazione di un'escursione . . . . .	32
12	API per la visualizzazione di escursioni relative ad un giorno specifico	32
13	API per il login di un utente . . . . .	32
14	API per la registrazione di un utente . . . . .	33
15	Pseudocodice algoritmo - Parte 1 . . . . .	39
16	Pseudocodice algoritmo - Parte 2 . . . . .	40
17	Flow chart dell'algoritmo . . . . .	41
18	UML Component Diagram . . . . .	42
19	UML Class Diagram per interfacce . . . . .	43
20	UML Class Diagram per tipi di dato . . . . .	44
21	UML Deployment Diagram . . . . .	45
22	Risultati test dinamico . . . . .	46
23	Risultato Unit Test . . . . .	48
24	Dati per Integration Test . . . . .	50
25	Integration Test dell'algoritmo . . . . .	51
26	API per la creazione di una prenotazione . . . . .	53
27	API per la visualizzazione delle prenotazioni relative ad una escursione specifica . . . . .	53
28	Installazione Android Studio - Fase 1 . . . . .	56
29	Installazione Android Studio - Fase 2 . . . . .	56
30	Installazione Android Studio - Fase 3 . . . . .	57
31	Installazione Android Studio - Fase 4 . . . . .	58
32	Installazione Android Studio - Fase 5 . . . . .	59
33	Installazione Android Studio - Fase 6 . . . . .	60
34	Installazione Android Studio - Fase 7 . . . . .	61
35	Installazione Android Studio - Fase 8 . . . . .	62
36	Installazione Android Studio - Fase 9 . . . . .	63
37	Installazione Visual Studio Code - Fase 1 . . . . .	65

38	Installazione Visual Studio Code - Fase 2 . . . . .	66
39	Installazione Visual Studio Code - Fase 3 . . . . .	67
40	Installazione Visual Studio Code - Fase 4 . . . . .	68
41	Installazione Visual Studio Code - Fase 5 . . . . .	69
42	Installazione Visual Studio Code - Fase 6 . . . . .	70

## Elenco delle tabelle

1	Casi d'uso con priorità alta . . . . .	7
2	Casi d'uso con priorità media . . . . .	7
3	Casi d'uso con priorità bassa . . . . .	7
4	Toolchain e tecnologie . . . . .	13

# 1 Iterazione 0

## 1.1 Introduzione e panoramica del sistema

Il sistema che verrà implementato in questo progetto di studio si occuperà della gestione di un centro di immersioni, noto anche come *diving center* o *dive center*. Un centro di immersione è una struttura che fornisce supporto, attrezzatura e corsi per la pratica delle attività subacquee. Un centro di immersione fornisce principalmente tre tipi di servizi:

- Immersioni guidate.
- Noleggio attrezzatura.
- Scuola di immersione.

Per immersione guidata si intende un'immersione effettuata in gruppo o singolarmente insieme a guide subacquee addestrate, esperte e a conoscenza dei punti di immersione più importanti ed interessanti. Questa soluzione è ottimale quando ci si vuole immergere in luoghi non conosciuti o in parchi marini protetti per cui è necessaria un'autorizzazione e una guida. Inoltre, il centro diving offre i servizi di accompagnamento e assistenza, di trasposto con imbarcazioni o gommoni, che permettono l'entrata e l'uscita dall'acqua con praticità e comfort.

Il focus del nostro progetto sarà la gestione e l'organizzazione efficiente delle immersioni guidate e la gestione del noleggio attrezzatura; non verrà incluso l'insegnamento della pratica subacquea.

Un centro di immersione ha una disponibilità limitata di barche e guide per cui ha la possibilità di effettuare un numero limitato di escursioni durante una giornata. Il sistema che verrà sviluppato si occuperà di organizzare al meglio l'allocazione delle risorse disponibili affinché un numero maggiore di persone possa parteciparvi. Spesso le persone che vogliono partecipare a questo tipo di escursioni si trovano in gruppo, per cui è necessario ottimizzare l'allocazione delle risorse disponibili, che in questo caso costituiscono i posti disponibili in barca, con il vincolo di mantenere il più possibile intatti i gruppi.

Il sistema implementato rappresenterà un centro di immersione: si occuperà della gestione delle risorse (barche, escursioni, attrezzatura disponibile) e della gestione delle prenotazioni delle escursioni da parte degli utenti.

## 1.2 Requisiti funzionali e analisi dei casi d'uso

In questa sezione verranno introdotti i requisiti funzionali del sistema attraverso l'utilizzo dei casi d'uso.

Lo schema UML dei casi d'uso viene riportato in Figura 1.

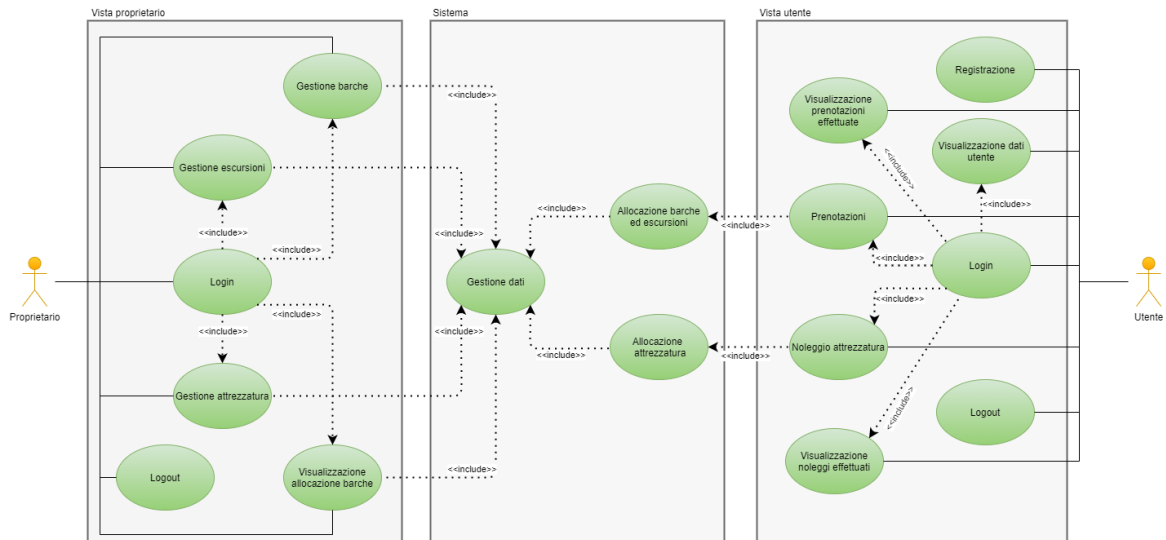


Figura 1: Diagramma UML dei casi d'uso

Al fine di procedere ad uno sviluppo efficiente, è stato deciso di dividere le specifiche funzionali in tre code di priorità: alta, media e bassa. Nella coda ad alta priorità si troveranno i casi d'uso indispensabili al corretto funzionamento dell'applicazione, nella coda a media priorità saranno inseriti i casi d'uso riguardanti le funzionalità aggiuntive e nella coda a bassa priorità le funzionalità non strettamente necessarie e che verranno implementate in versioni future.

### 1.2.1 Coda ad alta priorità

Codice	Titolo
UC1	Gestione barche (visualizzazione, inserimento, modifica, elimina)
UC2	Gestione escursioni (visualizzazione, inserimento, modifica, elimina)
UC3	Registrazione utente
UC4	Login utente
UC5	Logout utente
UC6	Gestione prenotazioni (visualizzazione, inserimento)
UC7	Algoritmo gestione barche ed escursioni

Tabella 1: Casi d'uso con priorità alta

### 1.2.2 Coda a media priorità

Codice	Titolo
UC8	Visualizzazione allocazione barche
UC9	Gestione attrezzatura (visualizzazione, inserimento, modifica, elimina)
UC10	Noleggio attrezzatura
UC11	Algoritmo attrezzatura

Tabella 2: Casi d'uso con priorità media

### 1.2.3 Coda a bassa priorità

Codice	Titolo
UC12	Visualizzazione dati utente
UC13	Modifica dati utente
UC14	Visualizzazione prenotazioni effettuate
UC15	Visualizzazione noleggi effettuati

Tabella 3: Casi d'uso con priorità bassa



## **1.3 Requisiti non funzionali**

Il progetto verrà sviluppato tenendo in considerazione anche alcuni requisiti non funzionali, quali la manutenibilità, l'efficienza e l'usabilità.

### **1.3.1 Manutenibilità**

Il requisito di manutenibilità verrà rispettato mediante l'implementazione di tutte le risorse persistenti, ovvero barche, orari e date delle escursioni, attrezzatura. In questo modo, anche se in futuro dovessero cambiare alcune condizioni, sarebbe facile per il proprietario rifletterle all'interno del software.

### **1.3.2 Efficienza**

Il requisito dell'efficienza è anche l'obiettivo primario del progetto, ovvero l'allocazione ottimale delle risorse disponibili: i posti presenti nelle barche.

### **1.3.3 Usabilità**

L'usabilità è garantita dalla decisione di sviluppare il programma attraverso un'applicazione Android, di facile utilizzo sia per gli utenti che per il proprietario. In base alla topologia (Figura 3) è possibile notare che il database verrà ospitato da un hosting online e quindi sarà sempre possibile, mediante cellulare, accedere a tutte le funzionalità dell'applicazione.

## 1.4 Topologia del sistema

La topologia del sistema mostrata in Figura 3, evidenzia il requisito non funzionale dell'usabilità: sia gli utenti che il proprietario possono accedere al servizio tramite l'applicazione. Il WebServer espone un set di API mediante HTTP/REST con cui è possibile accedere alle funzionalità dell'applicazione. Infine, il WebServer è collegato ad un database relazionale.

Questo progetto è stato sviluppato utilizzando un'architettura *Three-tier*, ovvero un'architettura hardware di tipo multi-tier. Essa prevede la suddivisione dell'applicazione in tre diversi moduli, i quali risiedono su macchine separate, dedicati rispettivamente a:

- Presentation, interfaccia utente
- Application, logica funzionale
- Data, gestione dei dati persistenti.

È possibile visualizzare questa divisione in Figura 3.

## 1.5 Design pattern MVP

Il sistema e i suoi componenti verranno sviluppati utilizzando il design pattern architetturale Model View Presenter (MVP), una derivazione del Model View Controller (MVC). MVP è composto da tre layer:

- Model, che definisce i dati dell'applicazione e può essere visto come un'interfaccia responsabile di accedere alle API connesse con il database.
- View, che visualizza i dati e notifica le azioni dell'utente. Solitamente non presenta nessuna logica applicativa, ma ha una funzione solo visuale.
- Presenter, che contiene la logica applicativa. Funziona come man-in-the-middle tra la View e il Model, prendendo i dati dal Model e formattandoli per la View.

A differenza del MVC, la View non è direttamente collegata al Model; l'interazione può avvenire solo tramite il Presenter. Questo permette uno sviluppo indipendente dei tre livelli. Tale separazione è visibile anche nelle componenti software, come si vede in Figura 4.

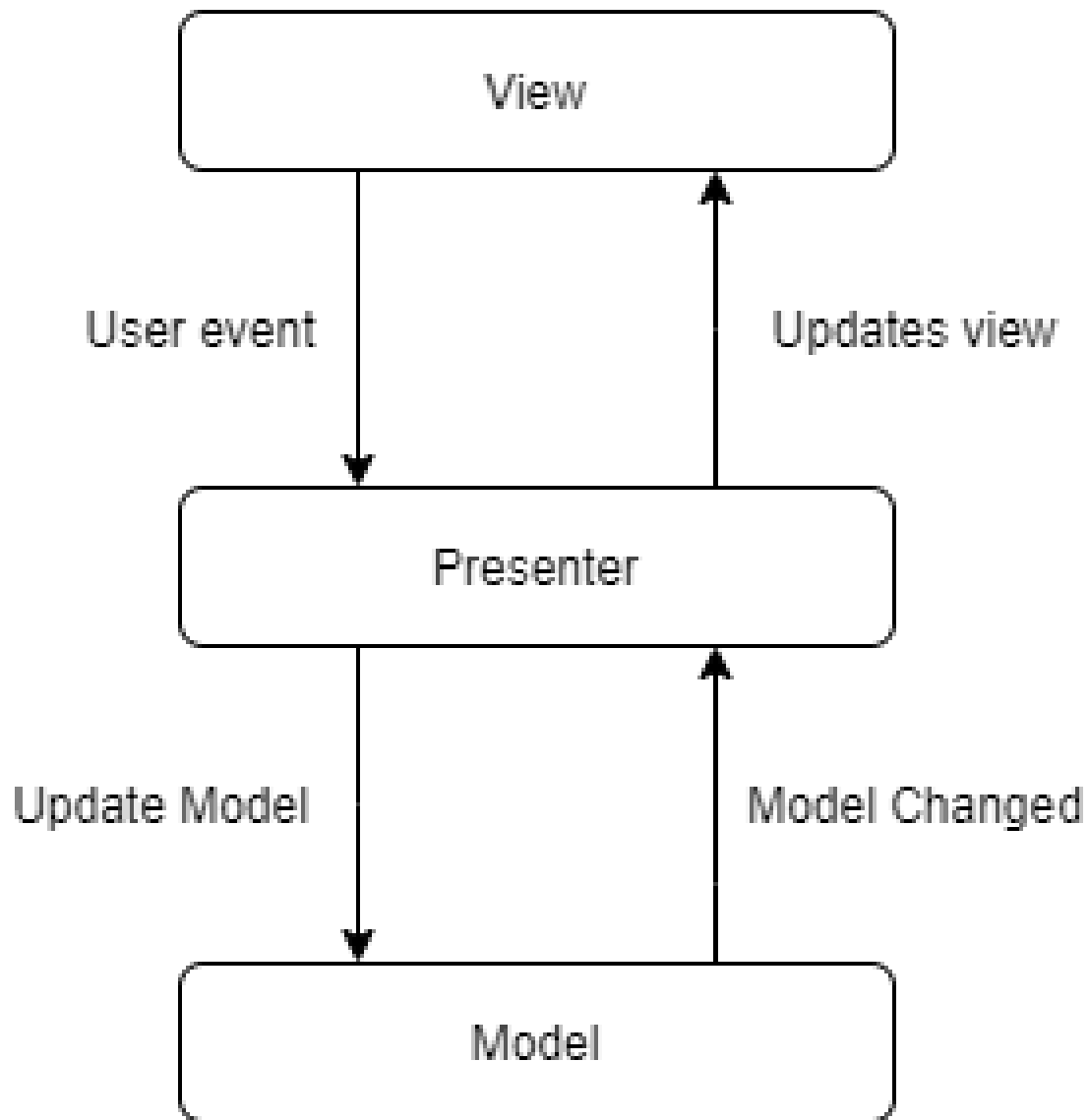


Figura 2: Model View Presenter

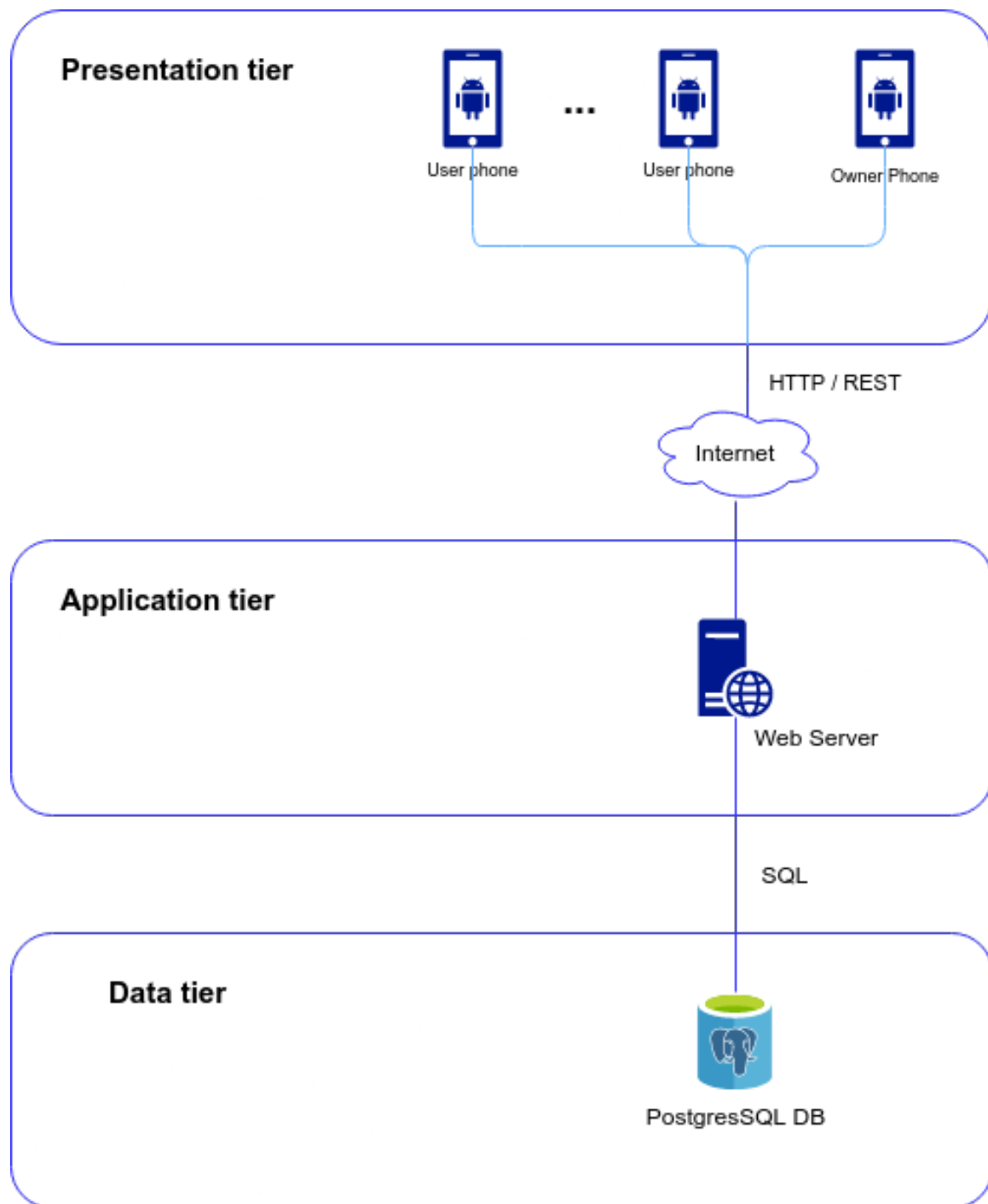


Figura 3: Topologia del sistema

## 1.6 Toolchain e tecnologie

### 1.6.1 Elenco completo

### 1.6.2 Tecnologie per analisi statica

Per l'analisi statica del codice a backend è stata utilizzata l'estensione *Language Support for Java(TM) by Red Hat* fornita dall'IDE *Visual Studio Code*. Questa permette di seguire le regole di formattazione standard di Java e avere una migliore leggibilità del codice, mantenendone alta la qualità.

### 1.6.3 Tecnologie per analisi dinamica

Per l'analisi dinamica del codice è stato utilizzato *Postman*, software che permette sia di richiamare le API realizzate a back-end, sia di testarle realizzando test dinamici.

<b>Tool/Tecnologia</b>	<b>Utilizzo</b>
Visual Studio Code	IDE per sviluppo back-end
Spring Boot	Java Framework per realizzazione back-end
Android Studio	IDE per sviluppo front-end
Kotlin	Linguaggio di programmazione per realizzazione front-end
Git e GitHub	Software e piattaforma per versionamento e condivisione di codice e documentazione
Google Drive	Cloud Storage per condivisione di documenti e immagini
Trello	Software gestionale per schedulazione attività tra i componenti del team per sviluppo agile
Jupyter Notebook con Python	Ambiente per testare l'algoritmo centrale del progetto
Canva	Tool di progettazione grafica per presentazione dell'applicazione
Draw.io	Tool per realizzazione grafici e diagrammi UML
Language Support for Java(TM) by Red Hat	Estensione di Visual Studio Code per analisi statica del codice a back-end
Postman	Piattaforma per test dinamico e documentazione delle API
JUnit 5	Framework per Unit Testing
Overleaf	Software per la stesura e formattazione della documentazione in linguaggio LaTeX

Tabella 4: Toolchain e tecnologie



## 2 Iterazione 1

### 2.1 Introduzione

Nella prima iterazione si è scelto di implementare i seguenti casi d'uso:

- UC1: Gestione barche [*Astratto*]
  - UC1.1 Visualizzazione barche
  - UC1.2 Inserimento barca
  - UC1.3 Modifica barca
  - UC1.4 Eliminazione barca
- UC2: Gestione escursioni [*Astratto*]
  - UC2.1 Visualizzazione escursioni
  - UC2.2 Inserimento escursione
  - UC2.3 Modifica escursione
  - UC2.4 Eliminazione escursione
- UC3: Registrazione utente
- UC4: Login utente
- UC5: Logout utente

In seguito viene riportata una descrizione testuale dei casi d'uso selezionati per la prima iterazione.



## 2.2 UC1: Gestione barche

*Breve descrizione:* il proprietario del diving center deve avere una visione generale e un controllo completo delle sue barche all'interno dell'applicazione. Oltre alla visualizzazione, il proprietario deve poter modificare ed eliminare le imbarcazioni già presenti nel sistema e poter aggiungerne di nuove. La gestione delle barche è quindi suddivisa in 4 casi d'uso concreti:

- UC1.1 Visualizzazione barche
- UC1.2 Inserimento barca
- UC1.3 Modifica barca
- UC1.4 Eliminazione barca

*Attori coinvolti:* Proprietario, Sistema.

*Trigger:* login del proprietario avvenuto con successo.

*Postcondizione:* il proprietario dopo il login, viene indirizzato alla *owner page* e può scegliere se cliccare uno dei button: *Manage Boat*, *Manage Trip* o *Manage Users*. Cliccando il primo viene indirizzato alla pagina della gestione delle barche (*boat page*), dove vengono visualizzate le barche esistenti ed è presente un tasto *Insert Boat*.

*Procedimento:*

1. Proprietario effettua il login.
2. Sistema mostra la pagina dedicata al proprietario (*owner page*) che comprende 3 bottoni: *Manage Boat*, *Manage Trip*, *Manage Users*. Cliccando sul primo bottone il proprietario può accedere alla sezione dedicata alla gestione delle barche (*boat page*).

### 2.2.1 UC1.1 Visualizzazione barche

*Breve descrizione:* Il proprietario visualizza le barche inserite nel sistema e le relative caratteristiche.

*Attori coinvolti:* Proprietario, Sistema.

*Trigger:* Il proprietario clicca il button *Manage Boat*.

*Postcondizione:* Il sistema mostra una vista con l'elenco delle barche.

*Procedimento:*

1. Proprietario effettua il login
2. Proprietario clicca il button *Manage Boat* presente nella owner page.
3. Sistema mostra l'elenco delle barche presenti nel database con le seguenti informazioni:
  - Modello della barca
  - Capienza della barca
4. Ogni item dell'elenco è cliccabile. Cliccando su una barca si viene indirizzati ad un'altra pagina che mostra le informazioni della barca e due tasti: *Delete Boat* per eliminare la barca e *Confirm* mediante il quale confermare le eventuali modifiche eseguite sui parametri della barca.

### **2.2.2 UC1.2 Inserimento barca**

*Breve descrizione:* Il proprietario aggiunge una barca nel sistema. Questo viene fatto principalmente nella fase iniziale in cui il proprietario deve inserire tutte le sue barche all'interno dell'applicazione e poi in seguito all'acquisto di nuove barche.

*Attori coinvolti:* Proprietario, Sistema.

*Trigger:* Il proprietario clicca il button *Insert Boat*.

*Postcondizione:* Il sistema mostra la nuova barca all'interno della pagina di visualizzazione delle barche.

*Procedimento:*

1. Il proprietario si trova sulla boat page e clicca il button *Insert Boat*.
2. Il sistema mostra un form in cui è possibile inserire i dati relativi alla barca.
3. Il proprietario riempie il form.
4. Il proprietario clicca su *Insert Boat* per inserire la barca nel sistema.
5. Il sistema mostra la nuova barca all'interno della pagina di visualizzazione delle barche *boat page*.

### 2.2.3 UC1.3 Modifica barca

*Breve descrizione:* Il proprietario si trova sulla pagina di visualizzazione delle barche. Cliccando su una barca viene visualizzata la pagina relativa alle informazioni della barca selezionata. In questa pagina il proprietario può modificare le informazioni della barca e poi cliccare il tasto *Confirm* per confermare.

*Attori coinvolti:* Proprietario, Sistema.

*Trigger:* Il proprietario clicca sulla barca che vuole modificare.

*Postcondizione:* Il sistema mostra la barca modificata all'interno della pagina di visualizzazione delle barche *boat page*.

*Procedimento:*

1. Il proprietario si trova sulla pagina di visualizzazione delle barche (*boat page*) e clicca su una specifica barca.
2. Il sistema mostra una vista in cui è possibile modificare i dati relativi alla barca.
3. Il proprietario modifica i dati.
4. Il proprietario clicca su *Confirm* per confermare le modifiche dei dati della barca.
5. Il sistema mostra la barca aggiornata all'interno della pagina di visualizzazione delle barche.

### 2.2.4 UC1.4 Eliminazione barca

*Breve descrizione:* Il proprietario si trova sulla pagina di visualizzazione delle barche e clicca su una barca, accede alla pagina relativa alle informazioni della barca selezionata. In questa pagina il proprietario sceglie di eliminare la barca cliccando sul bottone *Delete*. L'eliminazione può essere definitiva o temporanea. L'eliminazione è definitiva se la barca non è più agibile o perché viene sostituita da un'altra; è temporanea nel caso in cui sia necessaria attività di manutenzione.

*Attori coinvolti:* Proprietario, Sistema.

*Trigger:* Il proprietario clicca sulla barca da eliminare.

*Postcondizione:* Il sistema mostra la pagina di visualizzazione delle barche in cui non comparirà la barca eliminata.

*Procedimento:*

1. Il proprietario si trova sulla pagina di visualizzazione delle barche (*boat page*) e clicca su una barca specifica.
2. Il sistema mostra una pagina con le informazioni relative a quella barca e i bottoni *Confirm* e *Delete*.
3. Il proprietario clicca sul tasto *Delete*.
4. Il sistema mostra la pagina di visualizzazione delle barche in cui non comparirà la barca eliminata.

## 2.3 UC2: Gestione escursioni

L'applicazione deve poter semplificare la gestione e l'organizzazione delle escursioni al proprietario del diving center. L'applicazione deve fungere da calendario, mettendo a disposizione le seguenti funzionalità:

- UC2.1 Visualizzazione escursioni
- UC2.2 Inserimento escursione
- UC2.3 Modifica escursione
- UC2.4 Eliminazione escursione

### 2.3.1 UC2.1 Visualizzazione escursioni

*Breve descrizione:* Il proprietario visualizza le date e i turni in cui gli utenti possono prenotarsi.

*Attori coinvolti:* Proprietario, Sistema.

*Trigger:* Il proprietario clicca il button *Manage Trip*.

*Postcondizione:* Il sistema mostra una vista con l'elenco delle escursioni.

*Procedimento:*

1. Il proprietario effettua il login.
2. Il sistema mostra la pagina dedicata al proprietario (*owner page*) che comprende 3 bottoni: *Manage Boat*, *Manage Trip*, *Manage Users*. Cliccando sul bottone *Manage Trip* il proprietario può accedere alla sezione dedicata alla gestione delle escursioni (*trip page*).
3. Il proprietario clicca il button *Manage Trip* presente nella *owner page*.
4. Il sistema mostra la *trip page* contenente l'elenco delle escursioni e un bottone *Insert trip*. Ogni escursione viene visualizzata con le seguenti informazioni:
  - Data.
  - Orario di inizio.
5. Ogni escursione presente nell'elenco è cliccabile. Cliccando su un'escursione si viene indirizzati ad un'altra pagina, in cui vengono mostrate le informazioni dell'escursione. Inoltre sono presenti due tasti: *Delete* per eliminare l'escursione e *Confirm* per confermare eventuali modifiche apportate alle informazioni relative all'escursione.

### 2.3.2 UC2.2 Inserimento escursione

*Breve descrizione:* Il proprietario aggiunge un'escursione al calendario indicandone data e orario in cui verranno rese disponibili le prenotazioni.

*Attori coinvolti:* Proprietario, Sistema.

*Trigger:* Il proprietario clicca il button *Insert trip*.

*Postcondizione:* Il sistema mostra la nuova escursione all'interno della pagina di visualizzazione delle escursioni.

*Procedimento:*

1. Il proprietario si trova sulla *trip page* e clicca il bottone *Insert trip*.
2. Il sistema mostra un form in cui è possibile inserire i dati relativi all'escursione.
3. Il proprietario riempie il form.
4. Il proprietario clicca su *Confirm* per inserire l'escursione nel sistema.
5. Il sistema mostra la nuova escursione all'interno della pagina di visualizzazione delle escursioni *trip page*.

### 2.3.3 UC2.3 Modifica escursione

*Breve descrizione:* Il proprietario si trova sulla pagina di visualizzazione delle escursioni, clicca su una specifica escursione e viene indirizzato ad una nuova pagina che mostra le informazioni dell'escursione selezionata. Nella pagina sono presenti anche due bottoni: *Confirm* e *Delete*. Tale modifica può essere dovuta a cambiamenti climatici o ad altri imprevisti.

*Attori coinvolti:* Proprietario, Sistema.

*Trigger:* Il proprietario clicca su un'escursione.

*Postcondizione:* Il sistema mostra l'escursione modificata all'interno della pagina di visualizzazione delle escursioni.

*Procedimento:*

1. Il proprietario si trova sulla pagina di visualizzazione delle escursioni e clicca su un'escursione.
2. Sistema mostra una vista in cui è possibile modificare i dati relativi all'escursione.

3. Proprietario modifica i dati.
4. Proprietario clicca su *Confirm* per modificare i dati dell'escursione.
5. Il sistema mostra l'escursione aggiornata all'interno della pagina di visualizzazione delle escursioni.

#### **2.3.4 UC2.4 Eliminazione escursione**

*Breve descrizione:* Il proprietario si trova sulla pagina di visualizzazione delle escursioni, clicca su un'escursione e viene indirizzato ad una nuova pagina che mostra le informazioni dell'escursione selezionata. Nella pagina sono presenti anche i bottoni *Confirm* e *Delete*.

*Attori coinvolti:* Proprietario, Sistema.

*Trigger:* Il proprietario clicca il bottone *Delete*.

*Postcondizione:* Il sistema mostra la pagina di visualizzazione delle escursioni in cui non comparirà l'escursione eliminata.

*Procedimento:*

1. Il proprietario si trova sulla pagina di visualizzazione delle escursioni e clicca su un'escursione.
2. Il sistema mostra la pagina relativa all'escursione selezionata, contenente anche i bottoni *Confirm* e *Delete*.
3. Il proprietario clicca il bottone *Delete*.
4. Il sistema mostra la pagina di visualizzazione delle escursioni in cui non comparirà l'escursione eliminata.

## 2.4 UC3: Registrazione dell'utente

*Breve descrizione:* L'utente compila il form per la registrazione al servizio e viene aggiunto al database. La registrazione non può avvenire per utenti già registrati.

*Attori coinvolti:* Utente, Sistema.

*Trigger:* L'utente preme il bottone *Registrazione*.

*Postcondizione:* L'utente è stato inserito nel database e ha ricevuto la conferma dell'operazione.

*Procedimento:*

1. L'utente preme il bottone *Registrazione* nella *Homepage*.
2. L'utente fornisce le seguenti informazioni nel form di registrazione:
  - Nome
  - Cognome
  - Email
  - Password
  - Numero di brevetto
3. Il sistema verifica se l'email inserita è già associata ad un altro utente registrato:
  - (a) se l'email esiste, il sistema impedisce la registrazione e lo comunica all'utente
  - (b) se l'email non esiste, il sistema aggiunge l'utente nel database.



## 2.5 UC4: Login dell'utente

*Breve descrizione:* L'utente o il proprietario compila il form per il login; in caso di credenziali corrette il sistema consente l'accesso al servizio.

*Attori coinvolti:* Utente/Proprietario, Sistema.

*Trigger:* L'utente preme sul bottone *Login*.

*Postcondizione:* Nel caso l'utente sia un amministratore verrà indirizzato alla *owner page*, altrimenti alla *user page*.

*Procedimento:*

1. L'utente preme su *Login* nella pagina iniziale dell'app
2. L'utente fornisce username e password nel form di registrazione
3. Il sistema controlla le credenziali inserite:
  - (a) se sono corrette, avviene il login e il sistema invia una conferma di accesso all'utente
  - (b) se sono errate, viene impedito il login e notificato il mancato accesso all'utente.

## 2.6 UC5: Logout dell'utente

*Breve descrizione:* Il sistema effettua il logout dell'utente dall'applicazione.

*Attori coinvolti:* Utente, Sistema.

*Trigger:* L'utente preme il tasto *back button* della *owner page* o della *user page* e viene reindirizzato alla *Homepage*. In questa pagina è presente la sezione per effettuare il *Login* o la *Registrazione*.

*Postcondizione:* Il sistema mostra la *Homepage*.

*Procedimento:*

1. L'utente preme il *back button* della *owner page* o della *user page*
2. Il sistema effettua il logout dell'utente e mostra la *Homepage*.

## 2.7 UML Component Diagram

Partendo dai casi d'uso selezionati per questa iterazione e procedendo con l'utilizzo delle euristiche di design, è stato possibile progettare l'architettura software del nostro sistema. I casi d'uso sono stati raggruppati in base all'affinità e per ognuno è stato introdotto un componente *boundary*, un componente *control* e un componente *data*.

- «boundary» componenti lato front-end con cui gli attori si interfacciano direttamente.
- «control» componenti lato back-end che gestiscono la logica del programma e espongono delle API al front-end, richiedendone a loro volta al database.
- «data» componenti lato back-end che interagiscono con il database.

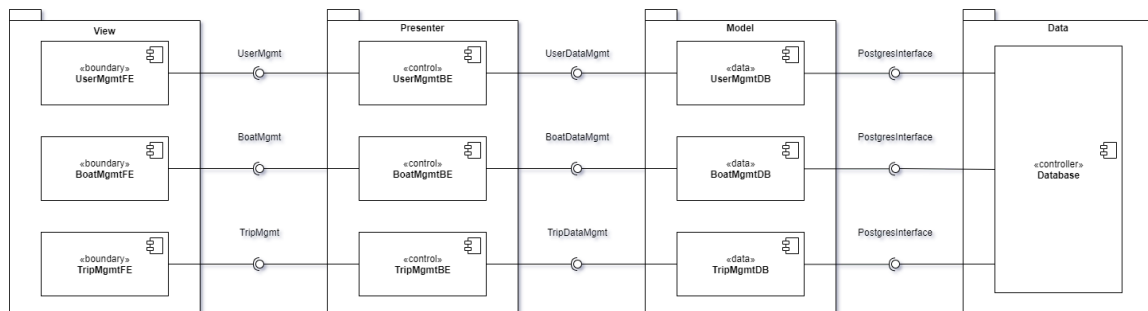


Figura 4: UML Component Diagram

## 2.8 UML Class Diagram per interfacce

Il diagramma in Figura 5 mostra le interfacce del sistema con le relative funzioni e i dati di input e output. Il diagramma evidenzia anche i layers del design pattern Model View Presenter.

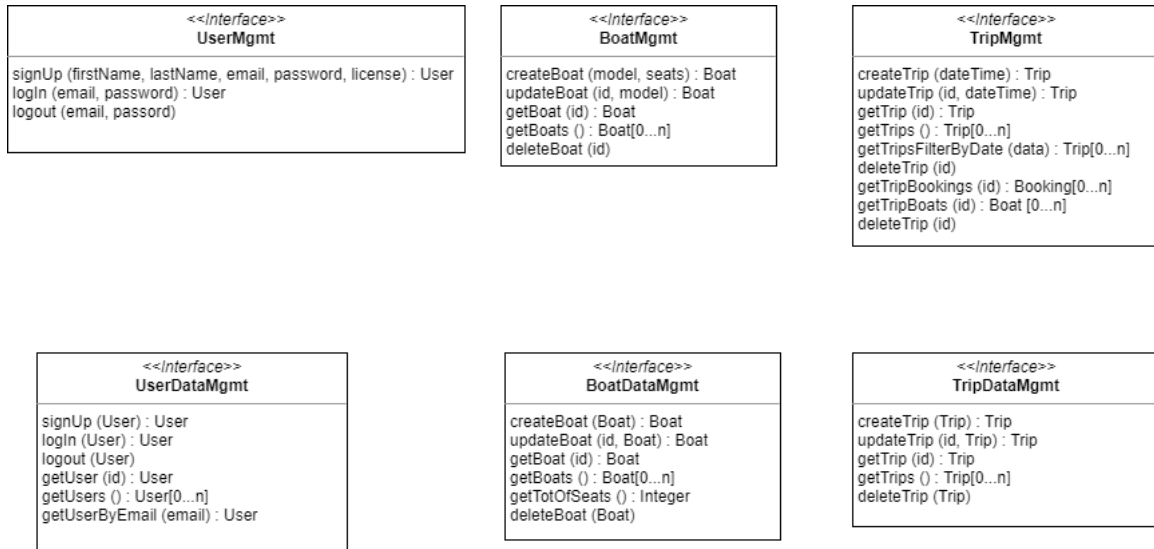


Figura 5: UML Class Diagram per interfacce

## 2.9 UML Class Diagram per tipi di dato

Il diagramma in Figura 6 mostra i tipi di dato necessari allo sviluppo dell'applicazione. Per la progettazione dei tipi di dato e delle interfacce sono state seguite le euristiche di design.

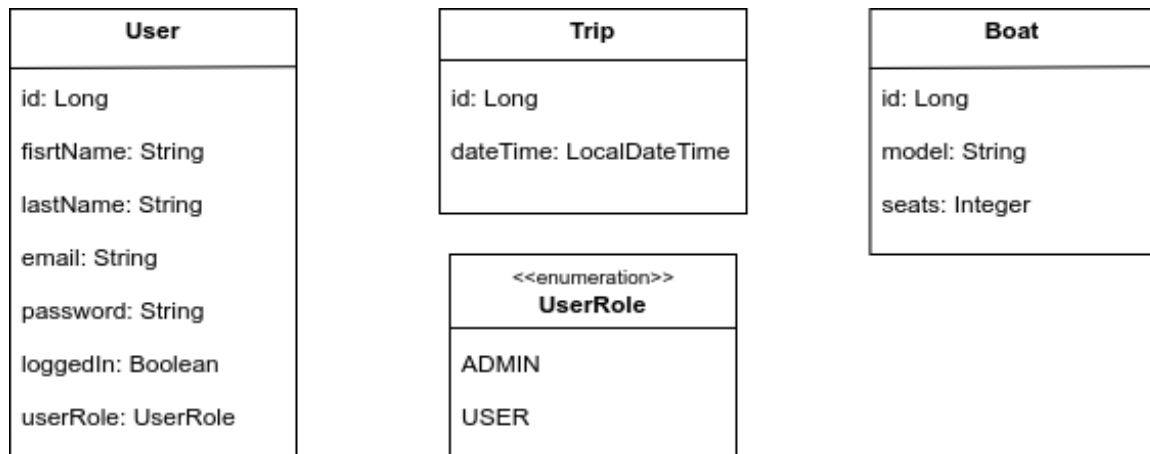


Figura 6: UML Class Diagram per tipi di dato

## 2.10 UML Deployment Diagram

Mediante l'utilizzo di un Deployment Diagram UML è possibile mostrare la rappresentazione hardware e software del sistema. In Figura 7 vengono mostrati i componenti contenuti nei seguenti nodi:

- Cellulare proprietario, nodo su cui l'admin del sistema potrà gestire le barche e il calendario delle escursioni.
- Cellulare utente, nodo su cui un soggetto potrà registrarsi alla piattaforma.
- Web Server, espone le API richieste lato front-end.
- Database, funge da storage dei dati.

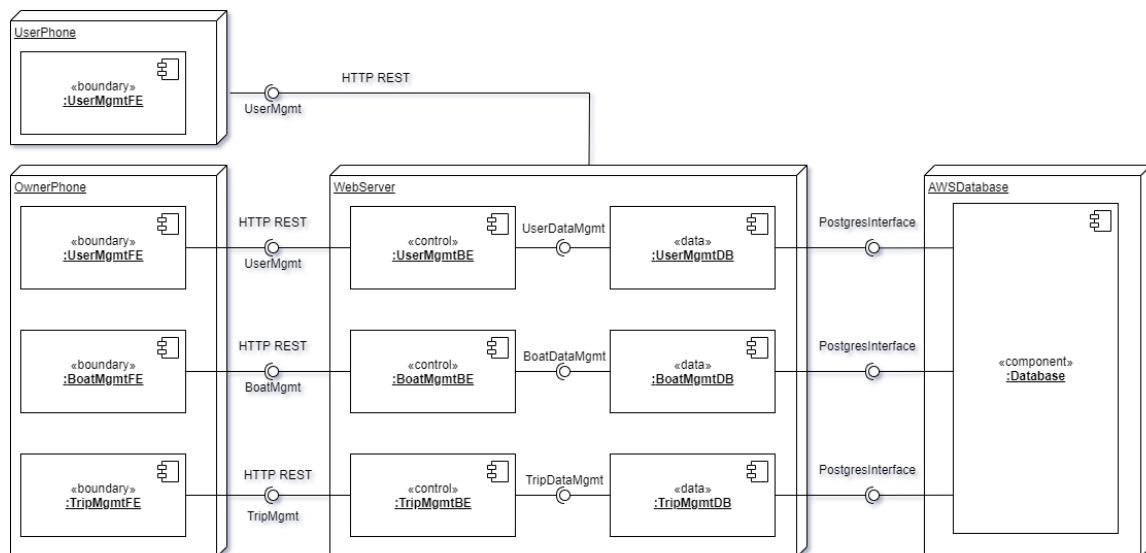


Figura 7: UML Deployment Diagram

## 2.11 Testing

### 2.11.1 Analisi statica

Come visto nel paragrafo 1.6.2 è stata utilizzata l'estensione *Language Support for Java(TM) by Red Hat* fornita dall'IDE *Visual Studio Code*.

### 2.11.2 Analisi dinamica

Per l'iterazione 1 è stato effettuato il test delle seguenti funzioni:

- Login con credenziali corrette
- Login con credenziali errate
- Logout a buon fine
- Creazione barca a buon fine
- Visualizzazione escursioni filtrate per data a buon fine

I risultati sono riportati in Figura 8.

All Tests	Passed (6)	Failed (0)
Iteration 1		
POST Login corretto	localhost:8080/users/login / Tests / Iterazione1 / Login corretto	200 OK 136 ms 299 B
Pass	Status test	
POST Login errato	localhost:8080/users/login / Tests / Iterazione1 / Login errato	404 Not Found 12 ms 304 B
Pass	Status test	
Pass	Status test	
POST Logout corretto	localhost:8080/users/logout / Tests / Iterazione1 / Logout corretto	200 OK 15 ms 165 B
Pass	Status test	
POST CreateBoat corretto	localhost:8080/boats / Tests / Iterazione1 / CreateBoat corretto	200 OK 15 ms 195 B
Pass	Status test	
GET GetTripsFilterByDate	localhost:8080/trips?date=20-05-2022 / Tests / Iterazione1 / GetTripsFilterByDate	200 OK 26 ms 204 B
Pass	Status test	

Figura 8: Risultati test dinamico

### 2.11.3 Unit Test

Per questa iterazione è stata testata la funzione *findByDateTime* presente nel back-end, che serve a filtrare le escursioni e ottenere solo quelle relative ad una certa data.

```
@DataJpaTest
class TripRepositoryTest {

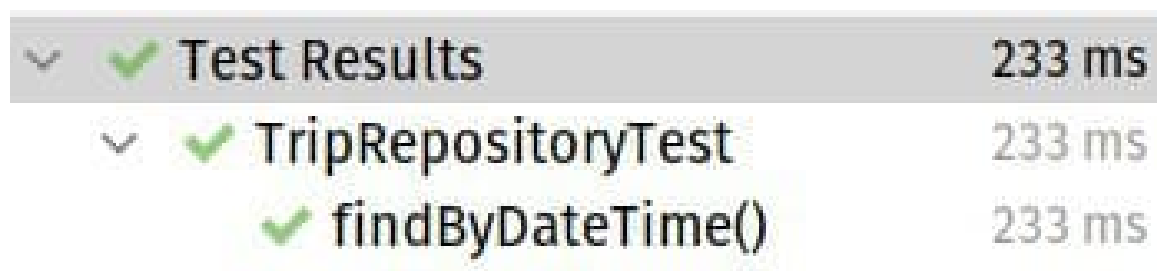
    @Autowired
    private TripRepository underTest;

    @Test
    void findByDateTime() {
        // given
        LocalDateTime date = LocalDateTime.of
            (2022, Month.JANUARY, 01, 10, 00, 00);
        Trip expected = new Trip(date);
        underTest.save(expected);

        // when
        Trip result = underTest.findByDateTime(date).get(0);

        // then
        assertThat(expected).isEqualTo(result);
    }
}
```

Il test di unità con JUnit ha confermato la correttezza della funzione.



▼	✓ Test Results	233 ms
▼	✓ TripRepositoryTest	233 ms
	✓ findByDateTime()	233 ms

Figura 9: Risultato Unit Test

## 2.12 Documentazione API

In questa sezione vengono mostrate alcune delle API sviluppate per l'iterazione 1. È possibile visualizzare tutte le API mediante la collezione di Postman che è possibile scaricare dalla repository su GitHub.

- API per la creazione di una barca. Figura 10
- API per la creazione di un'escursione. Figura 11
- API per la visualizzazione di escursioni relative ad un giorno specifico. Figura 12
- API per il Login di un utente. Figura 13
- API per la registrazione di un utente. Figura 14

### POST CreateBoat

localhost:8080/boats

#### BODY raw

```
{
  "model": "B1",
  "seats": 4
}
```

Figura 10: API per la creazione di una barca



### POST CreateTrip

localhost:8080/trips

#### BODY raw

```
{
  "dateTime": "20-05-2022 17:26"
}
```

Figura 11: API per la creazione di un'escursione

### GET GetTripsFilterByDate

localhost:8080/trips?date=10-07-2022

#### PARAMS

date	10-07-2022
------	------------

Figura 12: API per la visualizzazione di escursioni relative ad un giorno specifico

### POST Login

localhost:8080/users/login

#### BODY raw

```
{
  "email": "email1@email.com",
  "password": "password"
}
```

Figura 13: API per il login di un utente

### POST SignUp

localhost:8080/users/signup

#### BODY raw

```
{
  "firstName": "Nome",
  "lastName": "Cognome",
  "email": "email@email.com",
  "password": "password",
  "license": "ABC1234"
}
```

Figura 14: API per la registrazione di un utente



## 3 Iterazione 2

### 3.1 Introduzione

Nella seconda iterazione si è deciso di implementare i seguenti casi d'uso:

- UC6 Gestioni prenotazioni [*Astratto*]
  - UC6.1 Nuova prenotazione
  - UC6.2 Visualizzazione prenotazioni
- UC7 Algoritmo gestione barche ed escursioni

### 3.2 UC6: Gestione prenotazioni

*Breve descrizione:* L'utente, dopo essersi loggato, può effettuare la prenotazione per una delle escursioni organizzate dal proprietario. È possibile prenotare un posto non solo per sé stesso ma anche per altre persone, indicando il numero di partecipanti totale. Oltre alla creazione di una prenotazione, l'utente deve poter visualizzare le prenotazioni effettuate. Il caso d'uso *Gestione prenotazioni* è suddiviso in 2 casi d'uso concreti:

- UC6.1 Nuova prenotazione
- UC6.2 Visualizzazione prenotazioni

#### 3.2.1 UC6.1 Nuova prenotazione

*Breve descrizione:* L'utente effettua una nuova prenotazione per un'escursione organizzata dal proprietario.

*Attori coinvolti:* Utente, Sistema.

*Trigger:* L'utente clicca il bottone *Book excursion*.

*Postcondizione:* Il sistema mostra la nuova prenotazione all'interno della pagina di visualizzazione delle prenotazioni.

*Procedimento:*

1. L'utente si trova sulla *User page* e clicca il bottone *Book excursion*.
2. Il sistema mostra un calendario.
3. L'utente seleziona la data desiderata.

4. Il sistema mostra una lista degli orari disponibili per quella data, se presenti.
5. L'utente seleziona l'orario desiderato e inserisce il numero di partecipanti all'escursione. Infine, clicca il bottone *Book*.
6. Il sistema utilizza l'algoritmo (si veda il paragrafo 3.3) che permette di allocare in modo efficiente il gruppo alle barche disponibili per l'escursione scelta. Successivamente mostra un messaggio per notificare l'avvenuta o mancata prenotazione dell'escursione.
7. In caso di prenotazione a buon fine, il sistema mostra la prenotazione effettuata nella pagina di visualizzazione delle prenotazioni *My excursions*.

### 3.2.2 UC6.2 Visualizzazione prenotazioni

*Breve descrizione:* L'utente visualizza le prenotazioni effettuate e i relativi dati.

*Attori coinvolti:* Utente, Sistema.

*Trigger:* L'utente clicca il bottone *My excursions*.

*Postcondizione:* Il sistema mostra una vista con l'elenco delle prenotazioni.

*Procedimento:*

1. L'utente effettua il login
2. L'utente clicca il bottone *My excursions* presente nella *User page*.
3. Il sistema mostra l'elenco delle prenotazioni presenti nel database a nome dell'utente con le seguenti informazioni:
  - Data e ora escursione
  - Numero di persone per cui è stata prenotata l'escursione

### 3.3 UC7: Algoritmo barche ed escursioni

*Breve descrizione:* L'utente richiede di effettuare una prenotazione per un'escursione specificando data, orario e numero di persone del gruppo. Il sistema deve allocare il gruppo in modo ottimale, tenendo conto dei posti rimanenti sulle barche. Lo scopo principale dell'algoritmo è quello di minimizzare il numero di barche impiegate per l'escursione, in modo da ridurre i costi del proprietario. Per massimizzare il numero di prenotazioni, se il gruppo è troppo grande per i posti rimanenti sulle barche, si cerca di dividere il gruppo in due e allocarlo su due barche. Nel caso in cui non fosse possibile l'allocazione in questo modo, la prenotazione non può essere effettuata. L'algoritmo è quindi di tipo greedy.

*Attori coinvolti:* Sistema.

*Trigger:* L'utente richiede di effettuare una prenotazione.

*Postcondizione:* Prenotazione effettuata o fallita.

*Passi dell'algoritmo:*

1. Il primo passo dell'algoritmo viene attivato dal tentativo di prenotazione di un'escursione da parte di un utente per un gruppo di **N** persone.
2. Viene calcolato il numero di posti rimanenti (**P**) sulle barche per l'escursione scelta.
  - (a) se  $P > 0$ , l'algoritmo procede con il passo successivo.
  - (b) se  $P \leq 0$ , l'algoritmo termina con il **fallimento** della prenotazione, in quanto per l'escursione scelta non sono presenti posti rimanenti.
3. Si cerca una barca per cui sono stati già allocati altri gruppi, in modo tale da cercare di minimizzare il numero di barche utilizzate per escursione. La barca deve avere abbastanza posti rimanenti (**PR**) per contenere il gruppo per cui l'utente ha prenotato, ovvero deve verificarsi  $PR \geq N$ . La ricerca ritorna una lista di barche.
  - (a) Se la lista è vuota, si procede con il passo successivo.
  - (b) Se la lista non è vuota, essa viene ordinata in ordine crescente di posti rimanenti. Infine viene scelta la prima barca della lista, ovvero la barca che ha il numero minimo di posti rimanenti. La prenotazione termina con **successo**.
4. Si cerca una barca vuota (con capienza **C**), ovvero per cui non è stato già allocato alcun gruppo per l'escursione selezionata dall'utente. La barca deve

poter contenere il gruppo, ovvero deve verificarsi  $C \geq N$ . La ricerca ritorna una lista di barche.

- Se la lista è vuota, si procede con il passo successivo.
  - Se la lista non è vuota, si procede come al punto 3b. La prenotazione quindi termina con **successo**.
5. Arrivati a questo passo dell'algoritmo, significa che il gruppo non può essere allocato per intero né su una barca vuota, né su una barca in cui sono già presenti altri gruppi per l'escursione scelta. La scelta è stata di cercare di allocare il gruppo dividendolo in due gruppi separati solo se composto da un numero di persone superiore ad una soglia minima. In caso contrario c'è il **fallimento** della prenotazione. Si cercano quindi due barche (vuote o già parzialmente occupate) con  $PR \geq N/2$ . In caso di gruppi dispari una barca deve avere  $PR \geq N/2$ , l'altra  $PR \geq (N/2)+1$ . La ricerca ritorna una lista di barche.
- (a) Se esistono due barche in grado di contenere il gruppo diviso, si procede come al punto 3b e la prenotazione termina con **successo**.
  - (b) In caso contrario, neanche dividendo il gruppo è possibile trovare abbastanza posti, quindi si ha il **fallimento** del tentativo di prenotazione da parte dell'utente.

In Figura 15 e 16 viene mostrato lo pseudocodice dell'algoritmo con l'analisi di complessità. Nell'eseguire l'analisi di complessità si considera il numero di barche pari al valore  $n$  e il numero di prenotazioni pari al valore  $m$ . La complessità dell'algoritmo è scritta nell'equazione 1.

$$O(\max(m, \log n)) \quad (1)$$

```

Alg assignBoat(Booking newBooking) : boatAttribute
s[] // ritorna una lista di barche

```

$O(n)$

```

    Int remainingSeats = 0
    For boat in boats
        remainingSeats += boat.rs //campo di boats contenente I posti rimanenti
    end For

    if remainingSeats <=0
        return null
    else
        // ricerca barche in uso per l'escursione
        usedBoats = [] // definizione e inizializzazione di una nuova lista
        For booking in bookings
            if booking.trip == newBooking.trip
                usedBoats.add(booking.boat)
            end if
        end For

        usedBoatsWithEnoughSeats = []
        For usedBoat in usedBoats
            if usedBoat.rs >= newBooking.numPeople
                usedBoatsWithEnoughSeats.add(usedBoat)
            end if
        end For
    end For

```

$O(m)$

$O(n)$

Figura 15: Pseudocodice algoritmo - Parte 1



```

if !usedBoatsWithEnoughSeats.isEmpty()
O(n log n) => MergeSort(usedBoatsWithEnoughSeats) // ordinamento crescente
return List[usedBoatsWithEnoughSeats[0]]
else
// cerca le barche non utilizzate
emptyBoatsWithEnoughSeats = []
For boat in boats
    if boat.capienza >= newBooking.numPeople
        emptyBoatsWithEnoughSeats.add(boat)
    end if
end For

if !emptyBoatsWithEnoughSeats.isEmpty()
    MergeSort(emptyBoatsWithEnoughSeats)
    return List[emptyBoatsWithEnoughSeats[0]]
else
    if newBooking.numPeople < MIN_GROUP
        return null
    else
        Int numGroup1 = Floor(newBooking.numPeople/2)
        Int numGroup2 = newBooking.numPeople - numGroup1
        availableBoats = []
        For boat in boats
            if(boat.rs >= numGroup1)
                availableBoats.add(boat)
            end if
        end For

        MergeSort(availableBoats)
O(n log n) => if (availableBoats[1].rs >= numGroup2)
            // ritorna la lista con i soli primi due elementi
            return List[availableBoats[0], availableBoats[1]]
        else
            return null
        end if
    end if
end if
end if
end Alg

```

Figura 16: Pseudocodice algoritmo - Parte 2

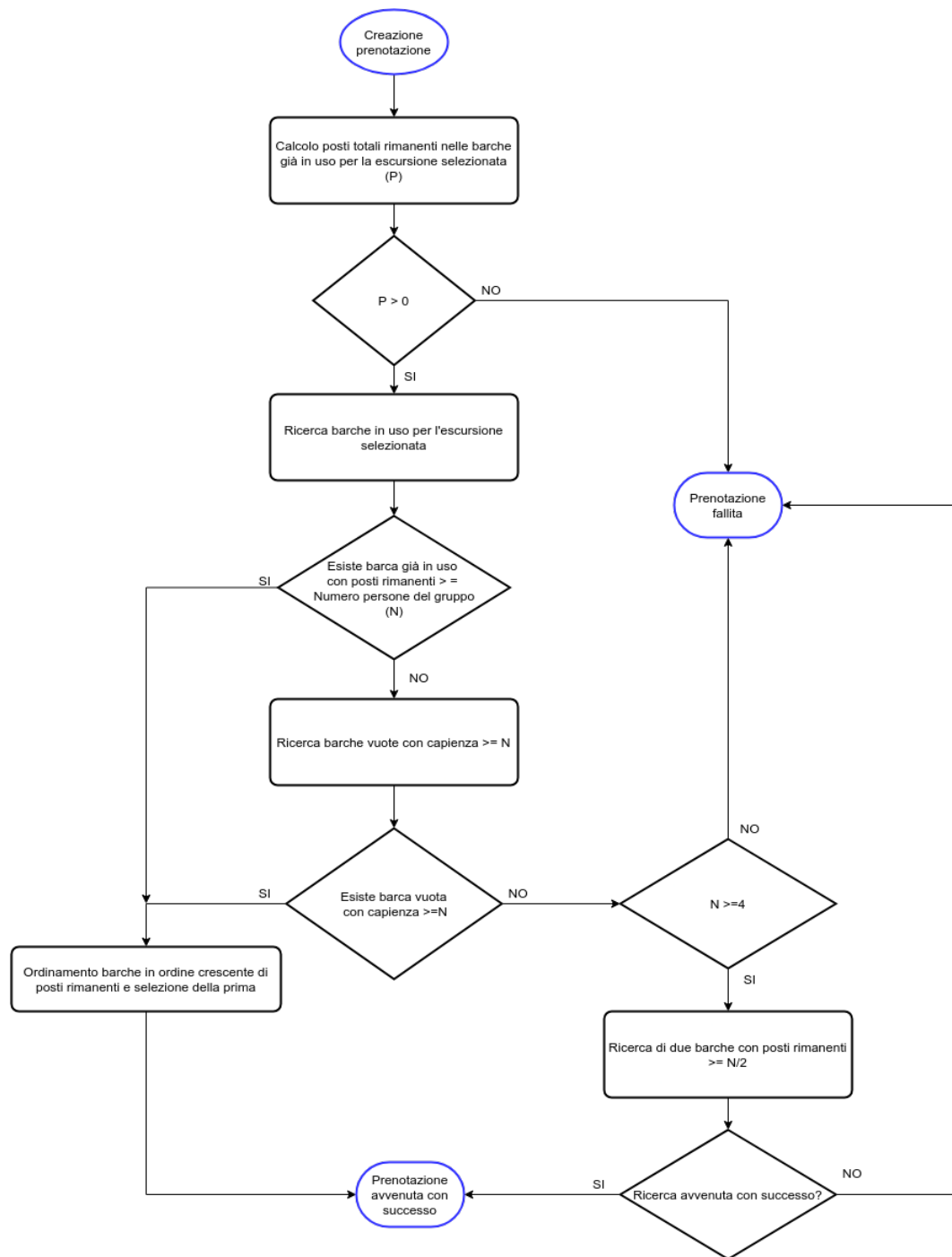


Figura 17: Flow chart dell'algoritmo

### 3.4 UML Component Diagram

I casi d'uso scelti nella seconda iterazione vengono aggiunti al *Component Diagram* dell'iterazione 1 (Figura 4). Il nuovo diagramma viene mostrato in Figura 18.

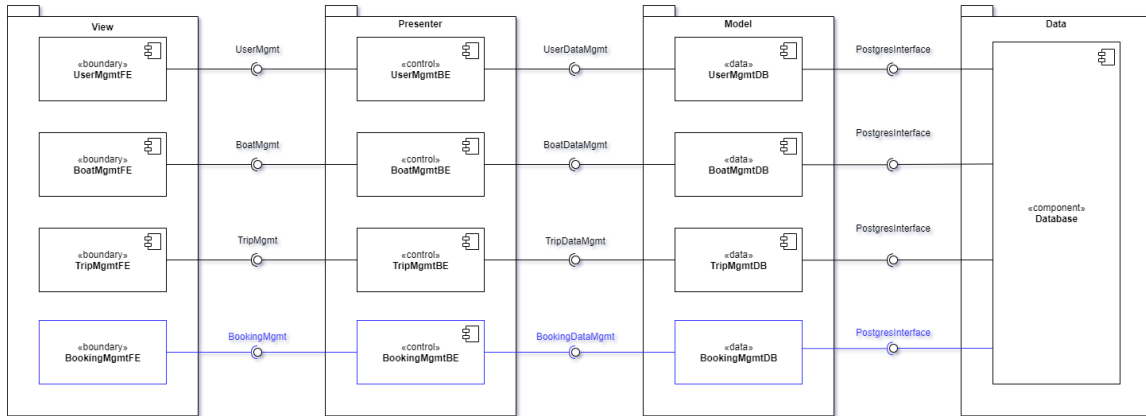


Figura 18: UML Component Diagram

### 3.5 UML Class Diagram per interfacce

Il diagramma in Figura 19 mostra le interfacce relative ai nuovi casi d'uso.

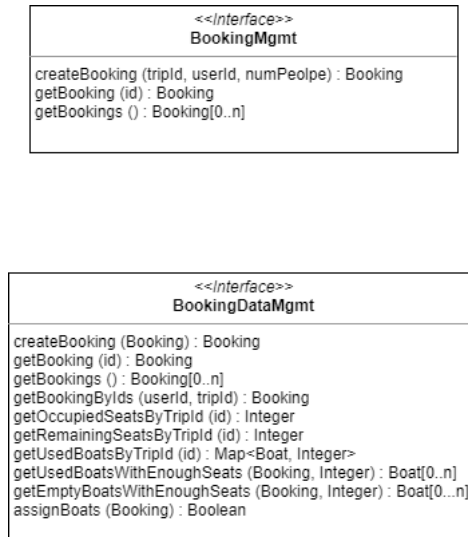


Figura 19: UML Class Diagram per interfacce

### 3.6 UML Class Diagram per tipi di dato

Il tipo di dato *Booking* introdotto in questa iterazione, viene inserito nel *Class Diagram per tipo di dato* (vedi Figura 6) dell'iterazione precedente. Il nuovo diagramma, mostrato in Figura 20, mostra anche le relazioni tra i tipi di dato instauratesi grazie all'aggiunta del nuovo tipo di dato.

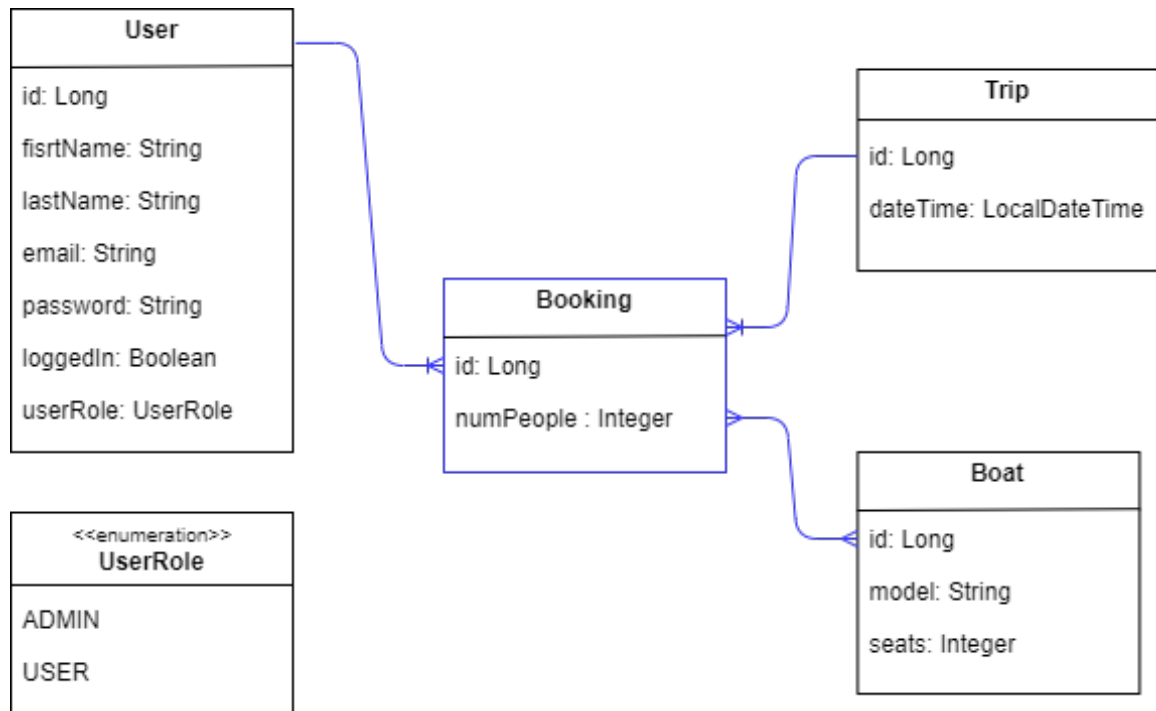


Figura 20: UML Class Diagram per tipi di dato

### 3.7 UML Deployment Diagram

I nuovi componenti relativi all'iterazione 2 vengono inseriti nel *Deployment Diagram* dell'iterazione precedente (vedi Figura 7). Il nuovo diagramma è mostrato in Figura 21.

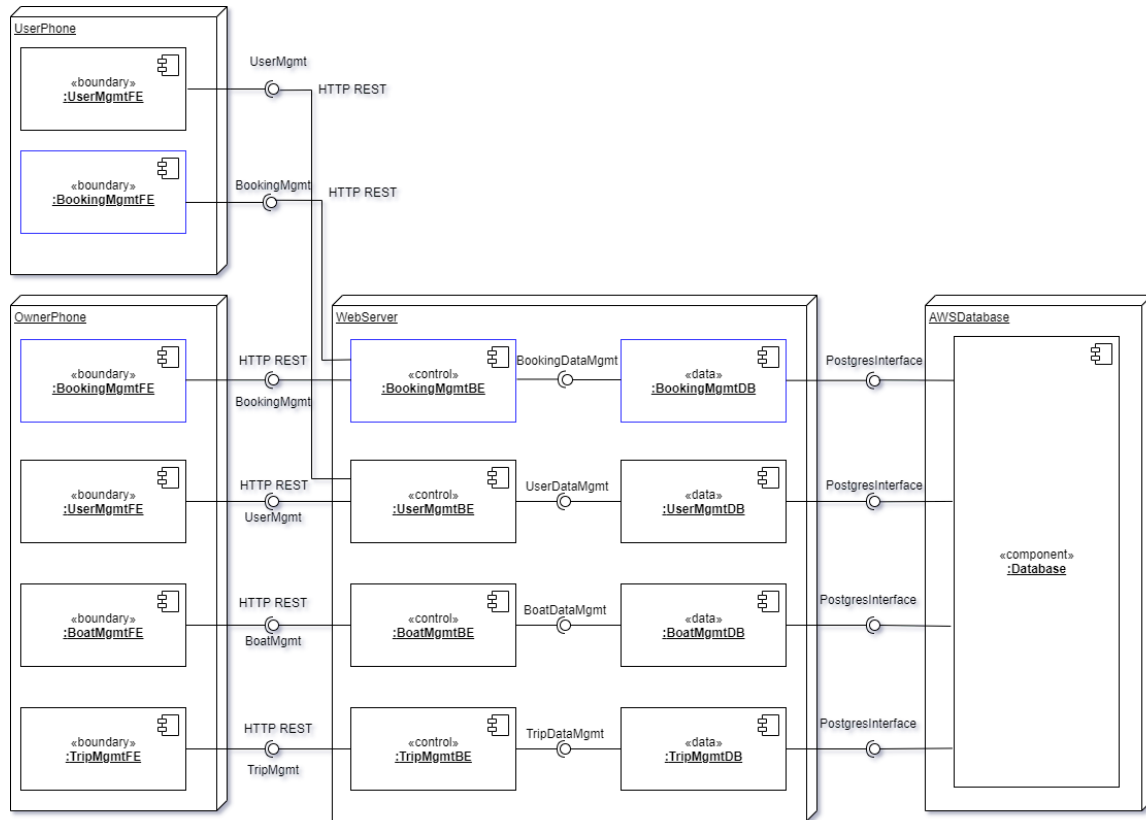


Figura 21: UML Deployment Diagram

## 3.8 Testing

### 3.8.1 Analisi statica

Come visto nel paragrafo 1.6.2 è stata utilizzata l'estensione *Language Support for Java(TM) by Red Hat* fornita dall'IDE *Visual Studio Code*.

### 3.8.2 Analisi dinamica

Per l'iterazione 2 è stato effettuato il test delle seguenti funzioni:

- Creazione prenotazione a buon fine
- Creazione prenotazione errata
- Visualizzazione prenotazione a buon fine

I risultati sono riportati in Figura 22.

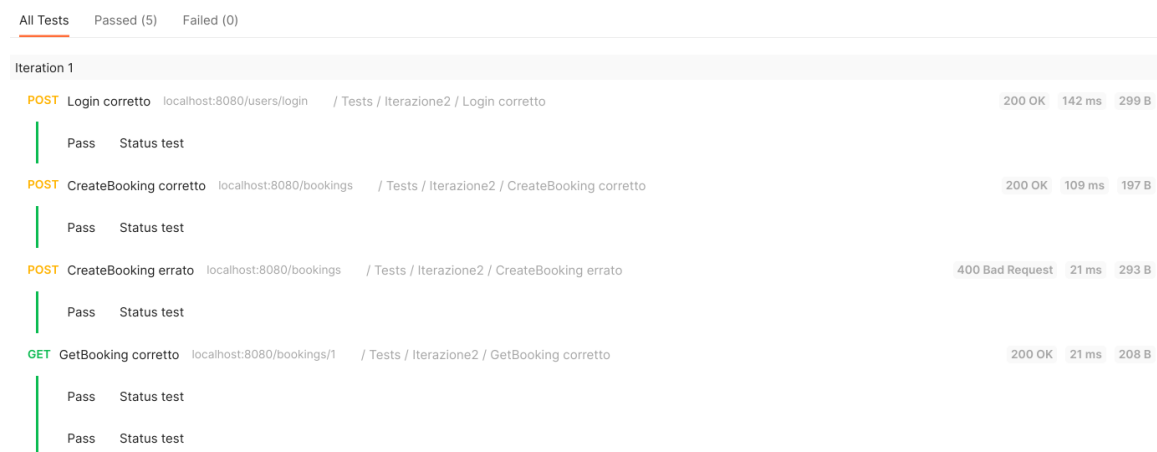


Figura 22: Risultati test dinamico

### 3.8.3 Unit Test

Per questa iterazione è stata testata la funzione *findByTripId* che serve per filtrare le prenotazioni e ottenere solo quelle relative ad una certa data.

```

@DataJpaTest
class BookingRepositoryTest {

    @Autowired
    private BookingRepository underTest;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private BoatRepository boatRepository;

    @Autowired
    private TripRepository tripRepository;

    @Test
    void findByTripIdOk() {
        // given
        User user = new User("Marco", "Bisceglia",
            "marco@gmail.com", "marco", "MMM");
        userRepository.save(user);

        Boat boat = new Boat("B1", 4);
        boatRepository.save(boat);

        LocalDateTime dayTrip = LocalDateTime.of
            (2022, Month.JANUARY, 01, 10, 00, 00);
        Trip trip = new Trip(dayTrip);
        tripRepository.save(trip);

        Booking expected = new Booking(5);
        expected.setUser(user);
        expected.setBoat(boat);
        expected.setTrip(trip);
        underTest.save(expected);

        // when
        List<Booking> result = underTest.findByTripId(1L);

        // then
    }
}

```



```
        assertThat(expected).isEqualTo(result.get(0));
    }
}
```

Il test ha confermato la correttezza della funzione.

✓	✓	Test Results	320 ms
✓	✓	BookingRepositoryTest	320 ms
	✓	findByTripIdOk()	320 ms

Figura 23: Risultato Unit Test

### 3.8.4 Integration Test dell'algoritmo

L'algoritmo che alloca i gruppi di persone sulle barche disponibili, per poter funzionare, necessita di funzioni presenti in moduli differenti: barche, utenti, escursioni e prenotazioni. Non è quindi possibile realizzare un test di unità che testa la singola funzione; si realizza invece un test di integrazione. A tal scopo è stato realizzato uno script in Python durante la fase di programmazione dell'algoritmo, in modo tale da poter testarne il corretto funzionamento utilizzando il modello di sviluppo *Test Driven Development*. Lo script, partendo da un database di test vuoto, lo popola con dei dati di test. Come si può vedere in Figura 24 vengono creati:

- 6 utenti
- 1 escursione
- 3 barche con rispettivamente 4, 8 e 4 posti (ogni posto viene rappresentato da un quadrato verde se disponibile, rosso se occupato)







Viene inoltre effettuato il login di ogni utente, fondamentale per poter effettuare in seguito la prenotazione di dell'escursione.

```
→ python git:(main) X python3 testAPI.py
```


### Registrazione utenti

 Giulia  
 Marco  
 Linda  
 Pippo  
 Valentino  
 Alessandro

### Login utenti

 Giulia logged  
 Marco logged  
 Linda logged  
 Pippo logged  
 Valentino logged  
 Alessandro logged

### Creazione escursione

 20-05-2022 10:00

### Creazione barche

 B1								
 B2								
 B3								

Figura 24: Dati per Integration Test

Una volta popolato il database di test viene testato l'algoritmo.  
Vengono testati i seguenti casi:

1. gruppo allocato correttamente (senza divisione gruppo)
2. gruppo non allocato (posti non terminati, ma non sufficienti per il gruppo, anche dividendolo)
3. gruppo allocato correttamente (con divisione gruppo)
4. gruppo non allocato (posti terminati)

L'esito dei test vengono mostrati in Figura 25.

```
Prenotazioni escursione

? Tentativo di prenotazione per gruppo da 2 persone
✓ Gruppo allocato in 1 barca: B1
B1 ████

? Tentativo di prenotazione per gruppo da 8 persone
✓ Gruppo allocato in 1 barca: B2
B1 ████
B2 ██████████

? Tentativo di prenotazione per gruppo da 9 persone
✗ FAIL. Posti non terminati, ma sparsi. Anche dividendo il gruppo non è stato possibile allocare barche
B1 ████
B2 ██████████

? Tentativo di prenotazione per gruppo da 5 persone
✓ Gruppo allocato in 2 barche: B3 B1
B1 ██████
B2 ██████████
B3 ████

? Tentativo di prenotazione per gruppo da 1 persone
✓ Gruppo allocato in 1 barca: B3
B1 ██████
B2 ██████████
B3 ██████

? Tentativo di prenotazione per gruppo da 2 persone
✗ FAIL. Posti terminati!
B1 ██████
B2 ██████████
B3 ██████
```

Figura 25: Integration Test dell'algoritmo

L'algoritmo si comporta come previsto:

- il primo, secondo e quinto tentativo di prenotazione testano il caso 1
- il terzo tentativo di prenotazione testa il caso 2
- il quarto tentativo di prenotazione testa il caso 3
- il sesto tentativo di prenotazione testa il caso 4.

### 3.9 Documentazione API

In questa sezione vengono mostrate alcune delle API sviluppate per l'iterazione 2.

- API per la realizzazione di una prenotazione. Figura 26
- API per la visualizzazione delle prenotazioni relative ad una escursione specifica. Figura 27

#### GET GetTripBookings

```
localhost:8080/trips/2/bookings
```

Figura 26: API per la creazione di una prenotazione

#### POST CreateBooking

```
localhost:8080/bookings
```

##### BODY raw

```
{
  "tripId": 2,
  "userId": 3,
  "numPeople": 4
}
```

Figura 27: API per la visualizzazione delle prenotazioni relative ad una escursione specifica



## 4 Guida all'installazione

Per lo sviluppo del Back-end del progetto è stato utilizzato l'IDE di sviluppo Visual Studio Code e per la programmazione del Front-end l'IDE Android Studio. Il Database risiede su Amazon AWS.

### 4.1 Installazione Android Studio

1. Andare sul sito <https://developer.android.com> e cliccare sul bottone *Download Android Studio*. Figura 28
2. Accettare i termini e le condizioni e premere il bottone *Download Android Studio*. Figura 29
3. Aprire il file eseguibile. Figura 30
4. Se presenti vecchie versioni di Android Studio verrà chiesta la conferma per cancellarle. Cliccare *Next*. Figura 31
5. Cliccare *Next* per continuare con l'installazione. Figura 32
6. Cliccare *Next* per continuare con l'installazione. Figura 33
7. Scegliere la cartella di installazione e cliccare *Next*. Figura 34
8. Scegliere la cartella dove verranno salvati i programmi e cliccare *Install*, una volta conclusa l'installazione cliccare *Next*. Figura 35
9. Cliccare il bottone *Finish* per terminare l'installazione e far avviare l'IDE. Figura 36



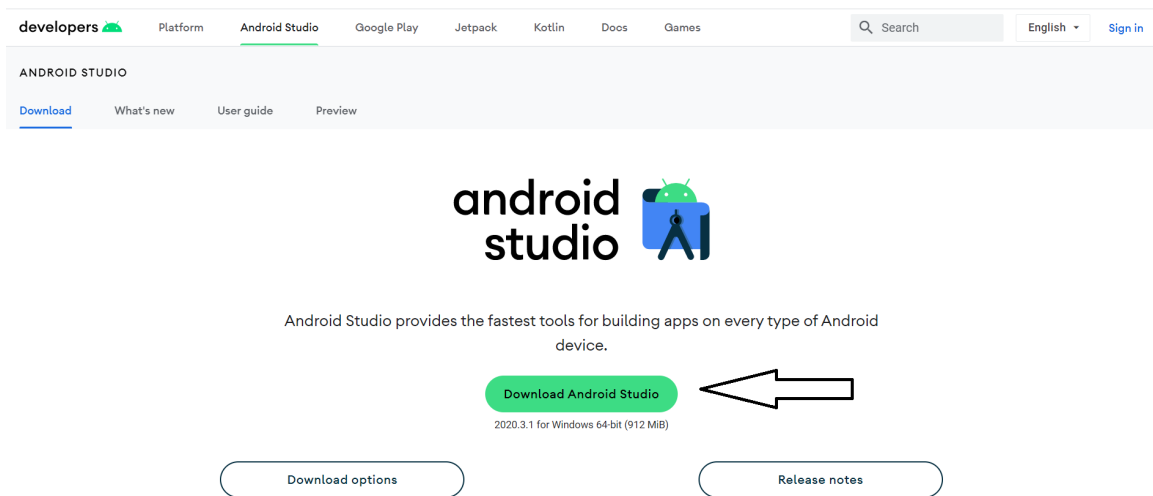


Figura 28: Installazione Android Studio - Fase 1

13.1 Google may make changes to the License Agreement as it distributes new versions of the SDK. When these changes are made, Google will make a new version of the License Agreement available on the website where the SDK is made available.

#### 14. General Legal Terms

14.1 The License Agreement constitutes the whole legal agreement between you and Google and governs your use of the SDK (excluding any services which Google may provide to you under a separate written agreement), and completely replaces any prior agreements between you and Google in relation to the SDK. 14.2 You agree that if Google does not exercise or enforce any legal right or remedy which is contained in the License Agreement (or which Google has the benefit of under any applicable law), this will not be taken to be a formal waiver of Google's rights and that those rights or remedies will still be available to Google. 14.3 If any court of law, having the jurisdiction to decide on this matter, rules that any provision of the License Agreement is invalid, then that provision will be removed from the License Agreement without affecting the rest of the License Agreement. The remaining provisions of the License Agreement will continue to be valid and enforceable. 14.4 You acknowledge and agree that each member of the group of companies of which Google is the parent shall be third party beneficiaries to the License Agreement and that such other companies shall be entitled to directly enforce, and rely upon, any provision of the License Agreement that confers a benefit on (or rights in favor of) them. Other than this, no other person or company shall be third party beneficiaries to the License Agreement. 14.5 EXPORT RESTRICTIONS. THE SDK IS SUBJECT TO UNITED STATES EXPORT LAWS AND REGULATIONS. YOU MUST COMPLY WITH ALL DOMESTIC AND INTERNATIONAL EXPORT LAWS AND REGULATIONS THAT APPLY TO THE SDK. THESE LAWS INCLUDE RESTRICTIONS ON DESTINATIONS, END USERS AND END USE. 14.6 The rights granted in the License Agreement may not be assigned or transferred by either you or Google without the prior written approval of the other party. Neither you nor Google shall be permitted to delegate their responsibilities or obligations under the License Agreement without the prior written approval of the other party. 14.7 The License Agreement, and your relationship with Google under the License Agreement, shall be governed by the laws of the State of California without regard to its conflict of laws provisions. You and Google agree to submit to the exclusive jurisdiction of the courts located within the county of Santa Clara, California to resolve any legal matter arising from the License Agreement. Notwithstanding this, you agree that Google shall still be allowed to apply for injunctive remedies (or an equivalent type of urgent legal relief) in any jurisdiction. July 27, 2021

☒ I have read and agree with the above terms and conditions

[Download Android Studio 2020.3.1 for Windows](#)

*android-studio-2020.3.1.24-windows.exe*

Figura 29: Installazione Android Studio - Fase 2

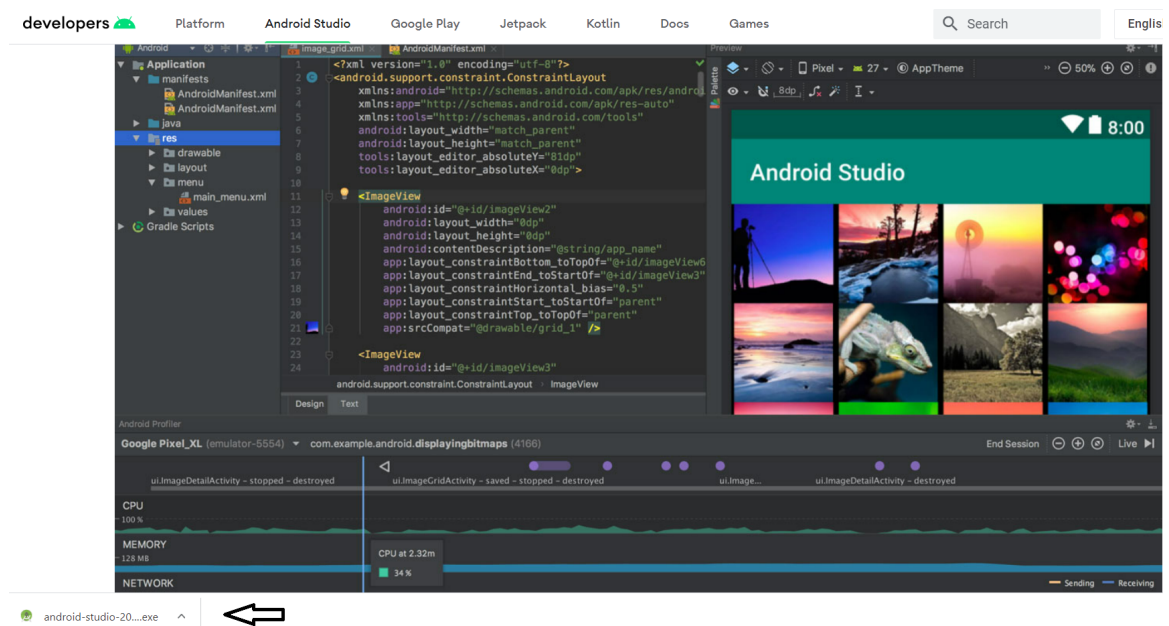


Figura 30: Installazione Android Studio - Fase 3

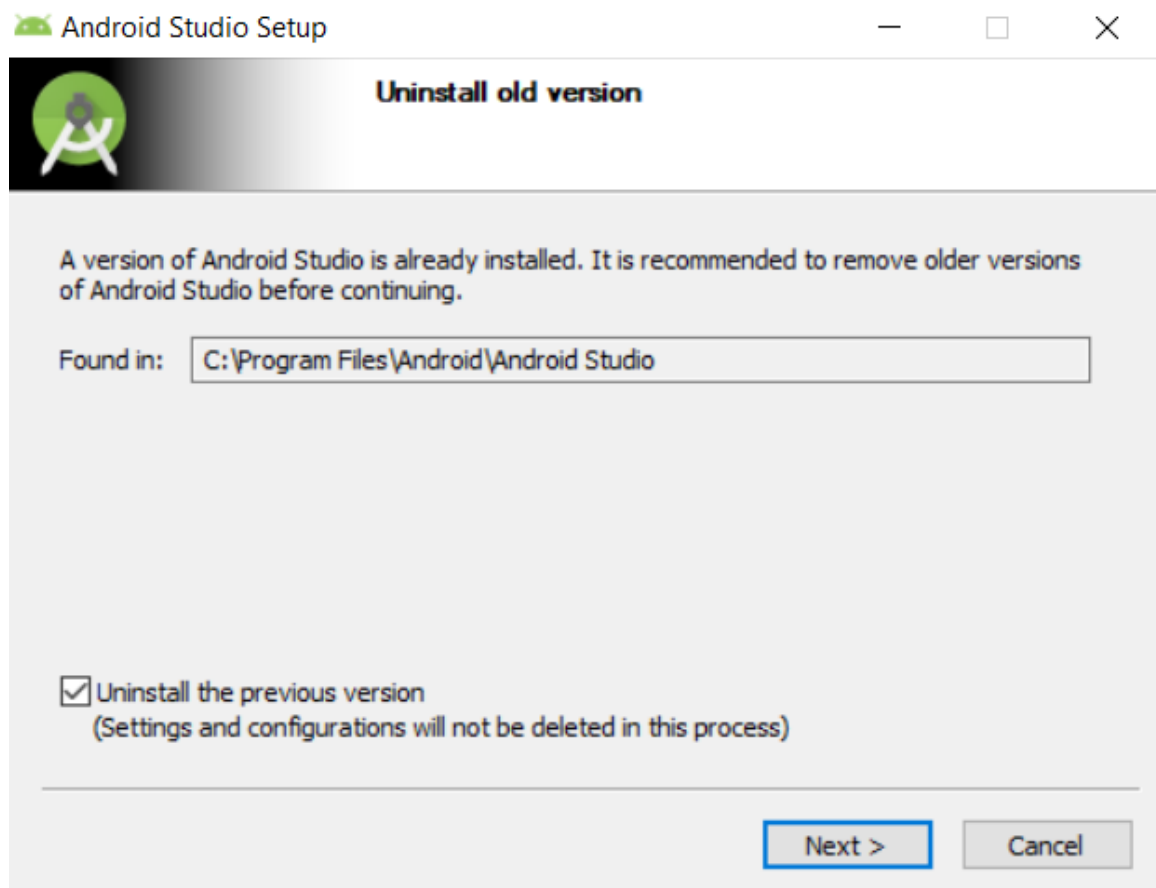


Figura 31: Installazione Android Studio - Fase 4



Figura 32: Installazione Android Studio - Fase 5

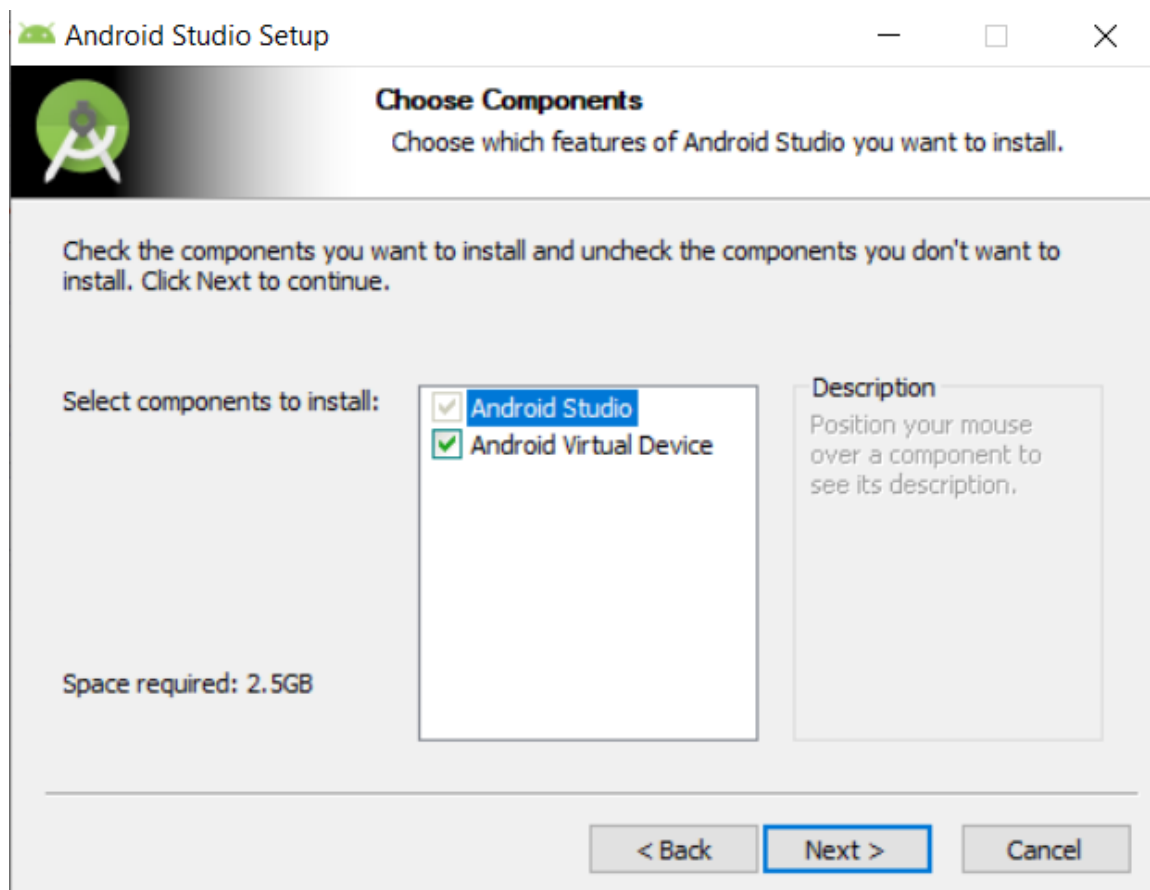


Figura 33: Installazione Android Studio - Fase 6

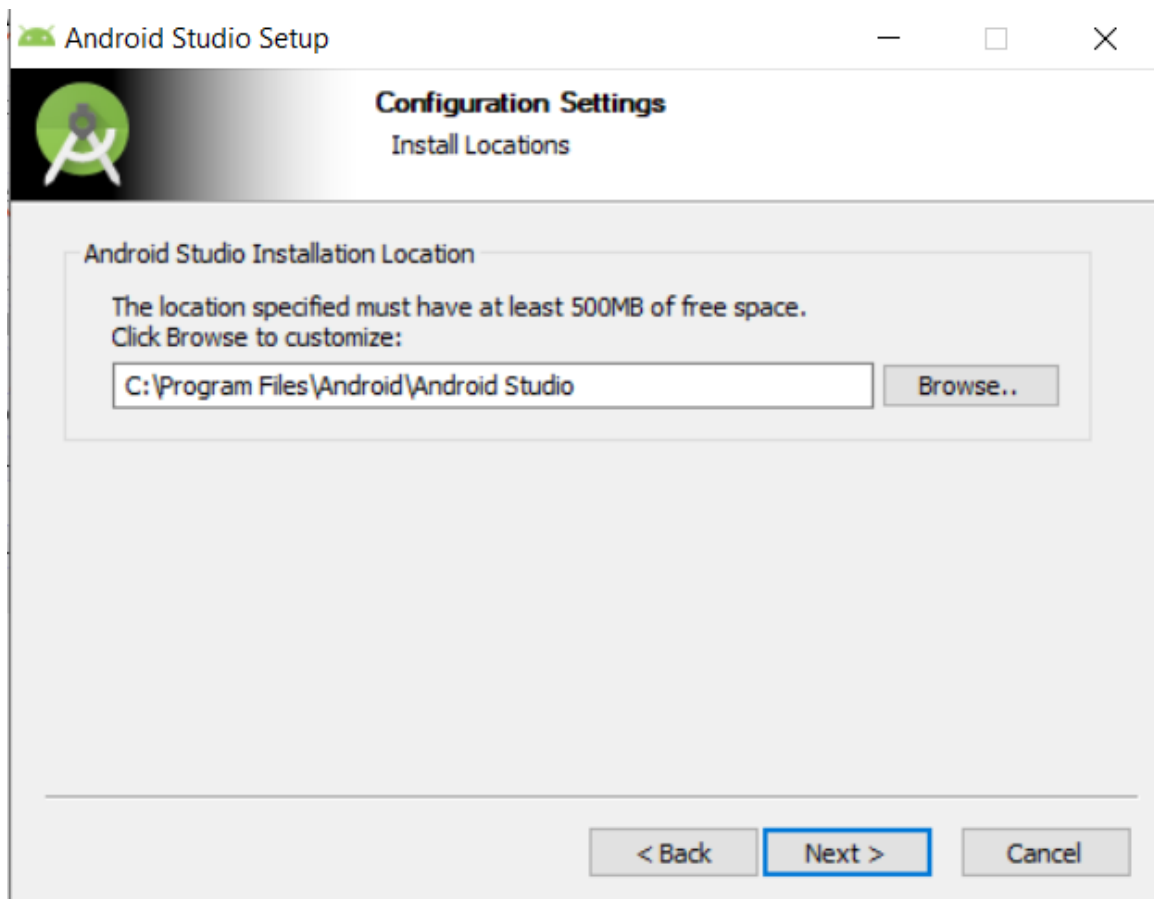


Figura 34: Installazione Android Studio - Fase 7

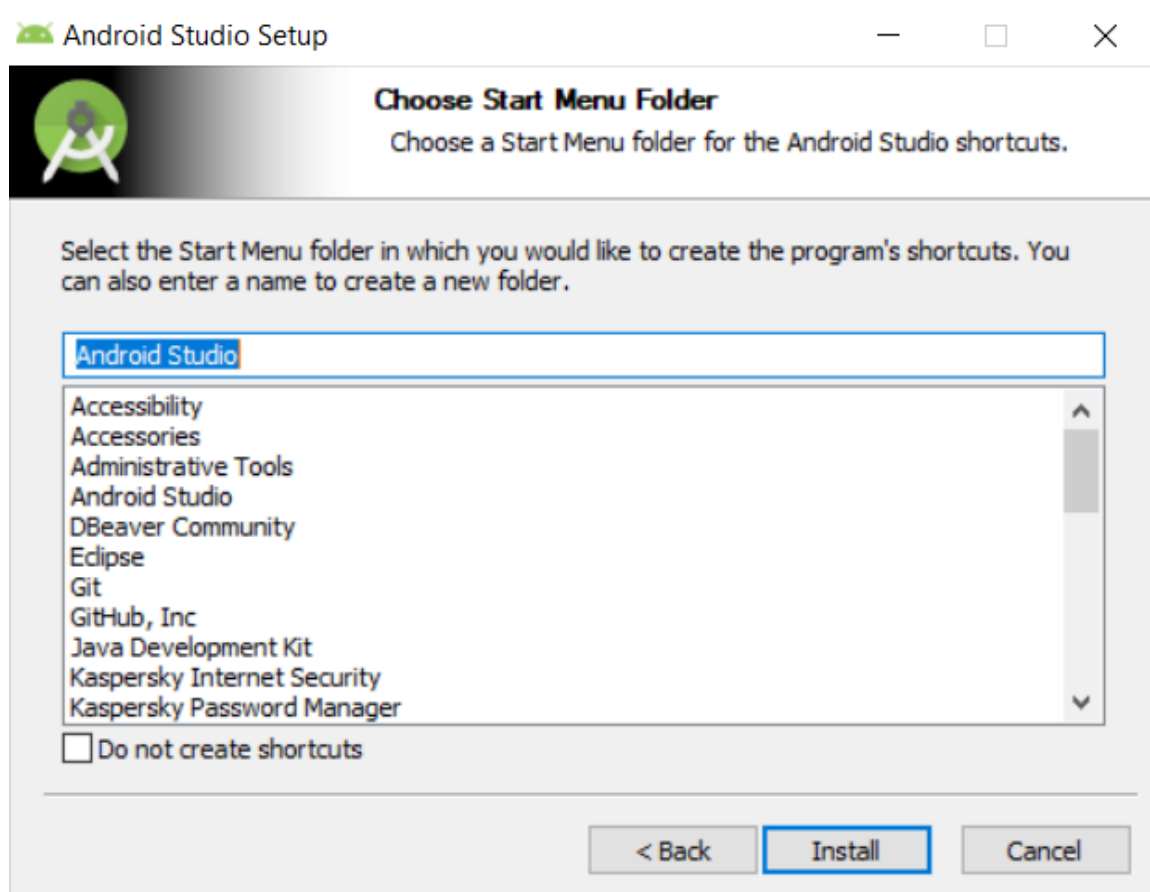


Figura 35: Installazione Android Studio - Fase 8

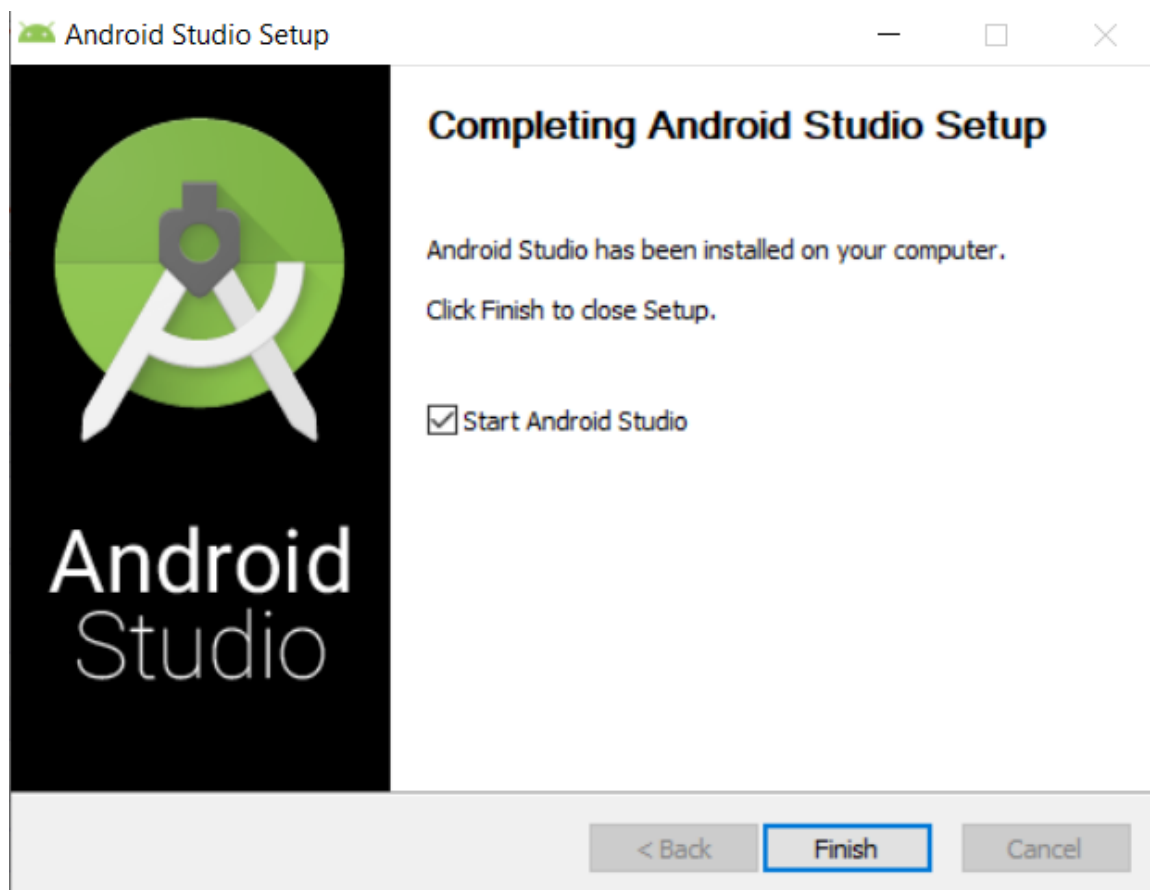


Figura 36: Installazione Android Studio - Fase 9



## 4.2 Installazione Visual Studio Code

1. Andare sul sito <https://code.visualstudio.com/download> e selezionare, in base al proprio sistema operativo il download corretto. In questa guida è stato scelto di cliccare sul bottone relativo a *Windows User installer* versione 64bit. Figura 37
2. Aprire il file eseguibile *VSCodeUserSetup.exe*. Figura 38
3. Accettare i termini di contratto e cliccare il bottone *Avanti*. Figura 39
4. Selezionare i processi aggiuntivi desiderati e premere il bottone *Avanti*. Figura 40
5. Cliccare il bottone *Installa*. Figura 41
6. Cliccare il bottone *Fine* e verrà aperto l'IDE di sviluppo Visual Studio Code. Figura 42

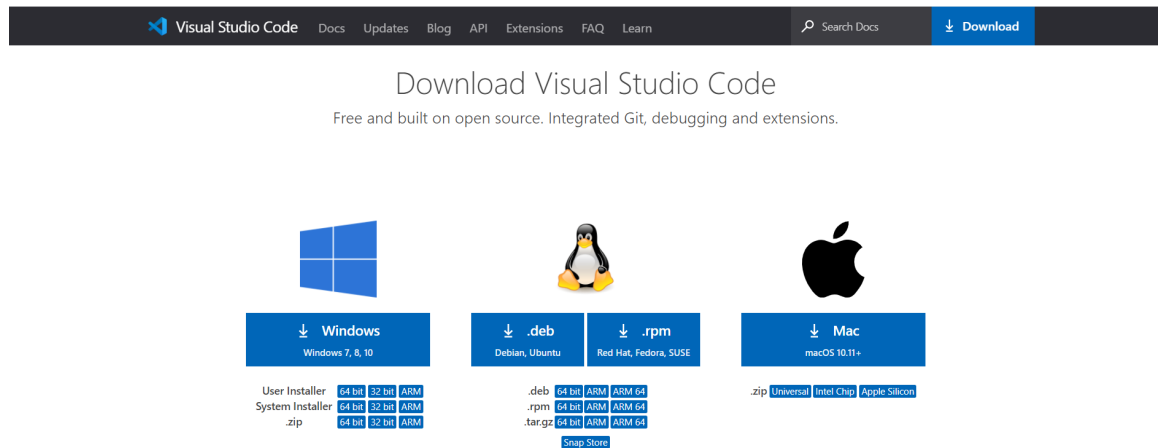


Figura 37: Installazione Visual Studio Code - Fase 1

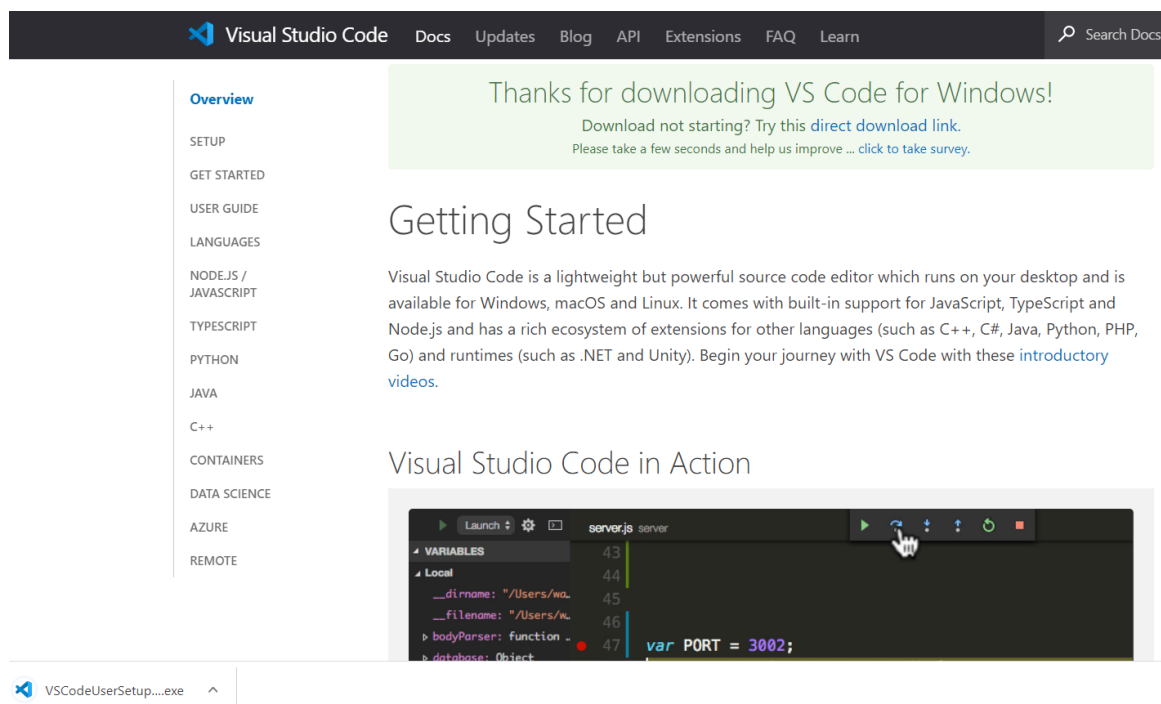


Figura 38: Installazione Visual Studio Code - Fase 2

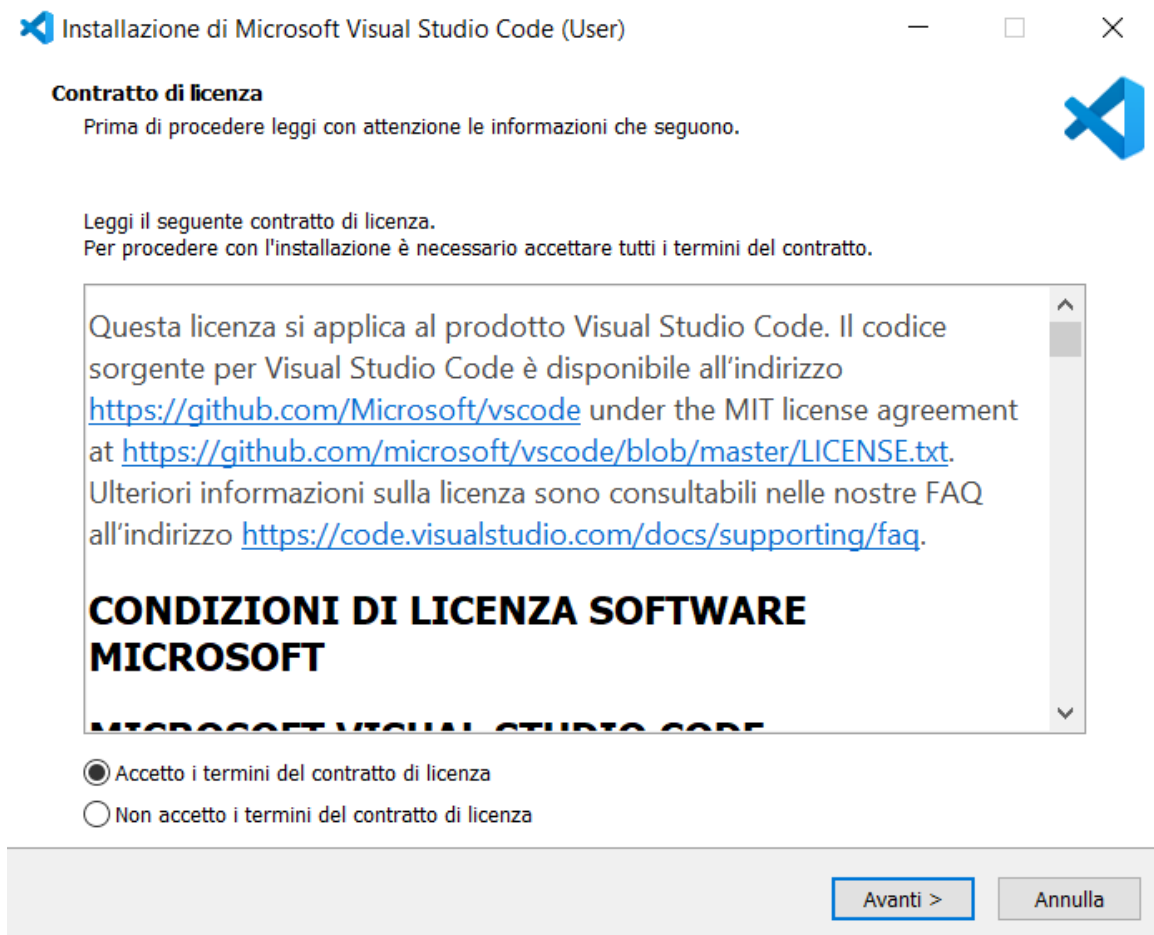


Figura 39: Installazione Visual Studio Code - Fase 3

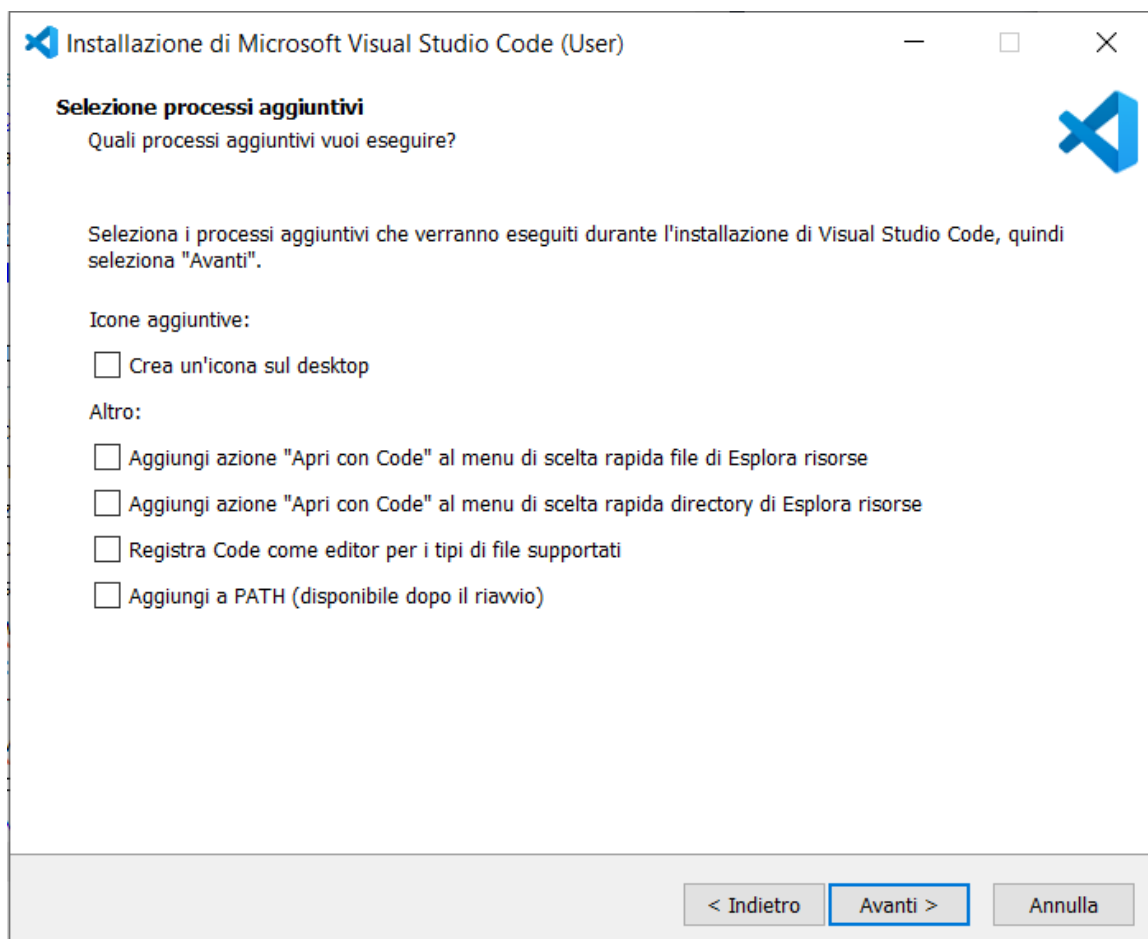


Figura 40: Installazione Visual Studio Code - Fase 4

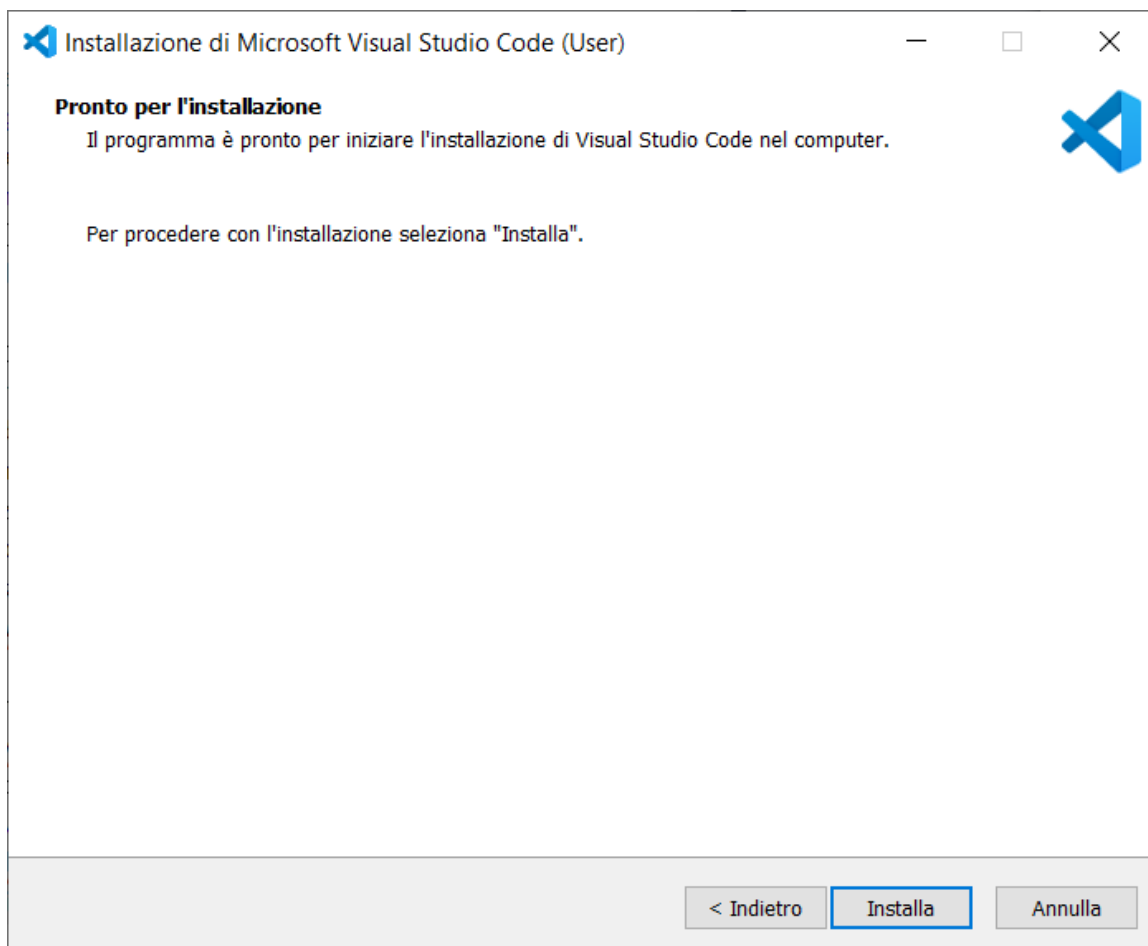


Figura 41: Installazione Visual Studio Code - Fase 5

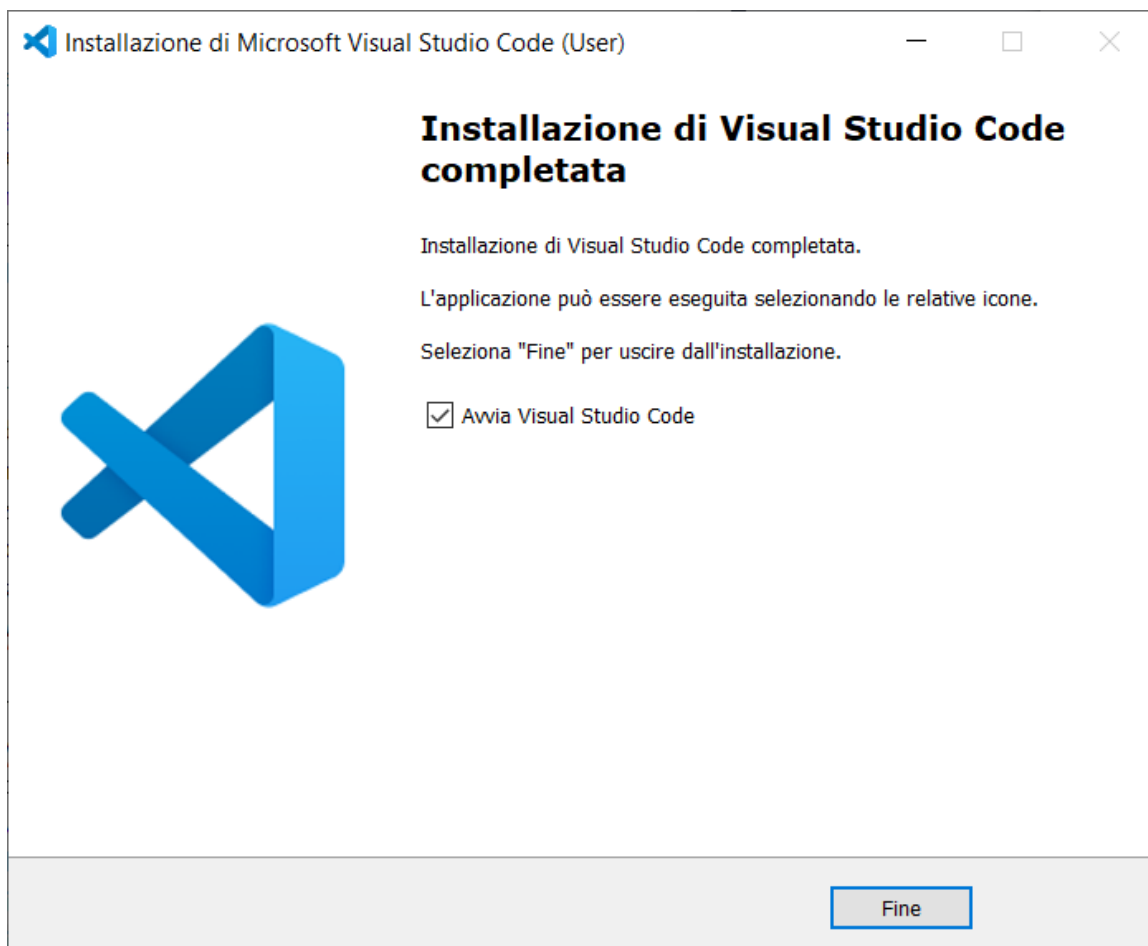


Figura 42: Installazione Visual Studio Code - Fase 6