**Documentazione progetto**

Immagine che contiene Elementi grafici, rosso, simbolo, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.**SmarTrip**

Componenti:

Daniele Gotti - 1079011

Bolis Filippo - 1079493

Gabriele Mazzoleni - 1079514

Università degli Studi di Bergamo

Laurea Magistrale in Ingegneria Informatica

Corso di Progettazione, Algoritmi e Computabilità

Prof. Patrizia Scandurra

Anno Accademico 2024/2025

**Indice**

[**Iterazione 0** 4](#_Toc197385754)

[**Introduzione** 4](#_Toc197385755)

[**Toolchain** 4](#_Toc197385756)

[**Analisi dei requisiti** 5](#_Toc197385757)

[**Casi d’uso** 6](#_Toc197385758)

[**Analisi delle specifiche** 8](#_Toc197385759)

[**Analisi dell’architettura** 8](#_Toc197385760)

[**Iterazione 1** 10](#_Toc197385761)

[**Aggiornamento dei casi d’uso** 10](#_Toc197385762)

[**Architettura del server** 10](#_Toc197385763)

[**Database** 11](#_Toc197385764)

# **Iterazione 0**

## **Introduzione**

SmarTrip è una soluzione software progettata per ottimizzare l'organizzazione dei viaggi, offrendo agli utenti un'esperienza semplice ed efficace nella pianificazione degli itinerari. A partire dalle città e dai luoghi selezionati, l'app è in grado di generare automaticamente il percorso migliore, tenendo conto di vincoli temporali e preferenze personali.

L'obiettivo principale di SmarTrip è fornire uno strumento innovativo e intuitivo che consenta di risparmiare tempo e migliorare l’esperienza di viaggio, grazie all’impiego di algoritmi di ottimizzazione e tecnologie moderne.

Lo sviluppo del software sarà guidato da una metodologia agile, scelta per garantire flessibilità, efficienza del prodotto finale. Questo approccio prevede la suddivisione del lavoro in iterazioni brevi e continue, permettendo un miglior controllo del processo, una rapida risposta ai cambiamenti e una comunicazione costante tra i membri del team. L'utilizzo della metodologia agile contribuisce a ridurre significativamente i costi e i tempi di sviluppo, assicurando al contempo un software più solido, affidabile e in linea con le reali esigenze degli utenti.

Il prodotto finale sarà composto da due elementi principali: un server e un'applicazione mobile. Il server gestirà la logica applicativa, i dati e l'esecuzione degli algoritmi di ottimizzazione, mentre l'app mobile SmarTrip costituirà l'interfaccia utente, progettata per offrire un'esperienza semplice.

## **Toolchain**

|  |  |
| --- | --- |
| **Tool** | **Utilizzo** |
| Eclipse | Utilizzato come ambiente di sviluppo integrato (IDE) per scrivere, eseguire e debuggare il codice Java del server. |
| Visual Studio Code | Usato come editor principale per lo sviluppo dell'applicazione mobile Flutter. |
| GitHub | Impiegato per il versionamento del codice e la collaborazione tra i membri del team tramite repository remoti. |
| Java | Linguaggio utilizzato per implementare la logica del server, inclusi i controller e i servizi. |
| Maven | Strumento di build e gestione delle dipendenze per i progetti Java del backend. |
| Spring Boot | Framework scelto per semplificare lo sviluppo del server e la gestione delle API REST. |
| jOOQ | Utilizzato per interagire con il database in modo typesafe e generare codice SQL a partire dallo schema. |
| JGraphT | Libreria per rappresentare e manipolare strutture grafiche (nodi, archi, percorsi), fondamentali per il calcolo degli itinerari ottimizzati. |
| DbBrowser for SQLite | Usato per visualizzare, modificare e gestire manualmente il contenuto del database SQLite. |
| CodeMR | Strumento utilizzato per l’analisi della qualità del codice e delle metriche di complessità del progetto Java. |
| JUnit 5 | Framework impiegato per scrivere ed eseguire i test automatici delle funzionalità del server. |
| Postman | Usato per testare le API REST esposte dal server durante le fasi di sviluppo e debugging. |
| Flutter | Framework scelto per realizzare l’interfaccia grafica dell’app mobile in modo cross-platform. |
| Dart | Linguaggio utilizzato per implementare la logica dell’app mobile sviluppata con Flutter. |
| Draw.io | Usato per creare diagrammi UML e schemi architetturali da inserire nella documentazione del progetto. |
| Microsoft Word | Utilizzato per redigere la documentazione tecnica e descrittiva del progetto. |
| Microsoft PowerPoint | Usato per realizzare la presentazione finale da esporre in sede di discussione del progetto. |

## **Analisi dei requisiti**

L’identificazione dei requisiti ha richiesto un’attenta riflessione sulle reali necessità degli utenti finali, con l’obiettivo di progettare un sistema che rispondesse in modo efficace e mirato ai loro bisogni. Per farlo, è stato necessario considerare diversi contesti d’uso e ipotizzare situazioni concrete in cui il software sarebbe stato impiegato. Questo processo ha permesso di delineare in modo chiaro sia i requisiti funzionali che quelli non funzionali, costituendo una base solida per le fasi successive di progettazione e sviluppo.

**Requisiti dell’utente**

* L’utente può scegliere una città tra quelle disponibili per organizzare un viaggio.
* L’utente può visualizzare la lista dei luoghi della città selezionata in modo da poter scegliere quali visitare.
* L’utente deve poter decidere i “ritmi” (giorni, orari partenza, ritorno, pausa, pranzo, ecc.) della vacanza.
* L’utente deve ricevere in maniera automatica l’itinerario di viaggio. L’itinerario calcolato massimizza il numero di luoghi che l’utente vuole visitare tra quelli selezionati, ottimizzando il percorso tra una tappa e l’altra.
* L’utente deve poter scegliere tra una lista di ristoranti disponibili più vicini al punto in cui effettuerà pause pranzo.

**Requisiti dell’addetto dell’amministrazione cittadina**

* L’addetto deve essere in grado di inserire con facilità la sua città e i luoghi visitabili.
* L’addetto deve essere in grado di inserire per ogni luogo tutte le caratteristiche utili alla costruzione di un itinerario di viaggio.

NOTA: per questo progetto abbiamo deciso di concentrarci sulla soddisfazione dei requisiti dell’utente, cioè si realizzerà database, server e app mobile per utente funzionanti. L’app mobile per l’amministratore non verrà implementata, ma sarà possibile usufruire di luoghi default inseriti nel database nella fase di sviluppo software. Questa scelta è anche dovuta al fatto che esistono già API a pagamento che consentono di prendere caratteristiche dei luoghi, rendendo ridondante la creazione di un’applicazione apposita.

## **Casi d’uso**

**Use case diagram**

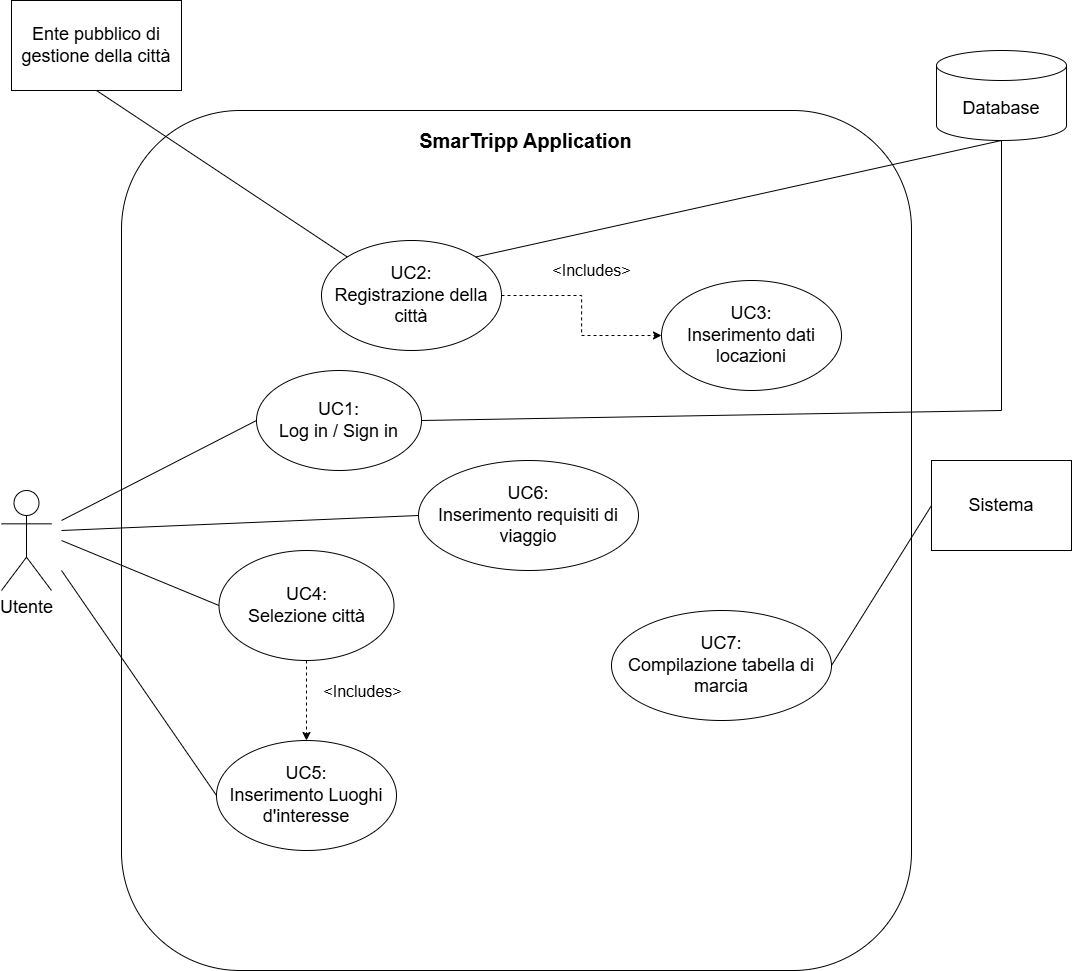
Il diagramma seguente illustra in modo dettagliato tutti i requisiti funzionali che il sistema dovrà implementare, ad eccezione dei casi d’uso del gestore della città che, come già anticipato, non saranno implementati. Ogni funzionalità rappresentata riflette un comportamento atteso dell’applicazione, definendo ciò che l’utente potrà fare e come il sistema dovrà reagire alle diverse interazioni.

Figura 1: use case diagram.

**Descrizione dei casi d’uso**

* UC1: LOG IN / SIGN IN.L’utente accede all’applicazione, inserendo una mail, un username e una password. Qualora la mail sia già in uso, viene restituito all’utente un messaggio d’errore.

POSTCONDIZIONI: l’utente può eseguire nuovamente l’accesso all’app inserendo solamente username e password.

* UC2: REGISTRAZIONE DELLA CITTA’. L’apposito ente incaricato registra la città all’interno del database dell’applicazione.

POSTCONDIZIONI: la città diventa disponibile alla selezione da parte degli utenti.

* UC3: INSERIMENTO DATI LOCAZIONI. L’ente inserisce i dati relativi ai punti d’interesse che intende rendere visitabili attraverso l’applicazione, inserendo informazioni riguardo a orari di apertura, prezzi etc.

POSTCONDIZIONI: le locazioni saranno disponibili per essere selezionate dagli utenti, e potranno essere utilizzate dal sistema per calcolare i percorsi migliori.

* UC4: SELEZIONE CITTA’. L’utente seleziona la città che desidera visitare, tra quelle che hanno precedentemente aderito all’applicazione.

POSTCONDIZIONI: viene proposta all’utente la scelta sui luoghi d’interesse da visitare.

* UC5: INSERIMENTO LUOGHI D’INTERESSE. L’utente compila una lista di luoghi che gli interessa visitare; questa lista può essere modificata in seguito e deve essere dunque conservata.
* UC6: INSERIMENTO REQUISITI DI VIAGGIO. L’utente inserisce una serie di dati relativi alle specifiche temporali del suo viaggio, quali data di arrivo, data di ritorno e a preferenze su ritmi di visita.
* UC7: COMPILAZIONE TABELLA DI MARCIA. Il sistema, basandosi sulle informazioni inserite dall’utente, calcola il percorso ottimale per permettere la visita del numero maggiore di luoghi d’interesse nei tempi e nei ritmi forniti.

PRECONDIZIONE: per poter avviare questo caso d’uso, l’utente deve aver inserito almeno un luogo d’interesse e aver compilato i requisiti di viaggio.

## **Analisi delle specifiche**

Le specifiche funzionali sono state suddivise in due livelli di priorità: alta e bassa. Le funzionalità assegnate alla priorità alta comprendono le operazioni essenziali per il corretto funzionamento del sistema. Quelle a priorità media includono caratteristiche utili ma non indispensabili, mentre la priorità bassa raccoglie le funzionalità opzionali, che potranno essere introdotte in versioni successive dell'applicazione.

|  |  |  |
| --- | --- | --- |
| **Specifica** | **Priorità** | **Implementato** |
| Sign in | Alta | Sì |
| Log in | Alta | Sì |
| Visualizzazione città disponibili | Alta | Sì |
| Visualizzazione luoghi disponibili della città selezionata | Alta | Sì |
| Inserimento dati del viaggio | Alta | Sì |
| Algoritmo ottimizzazione viaggio | Alta | Sì |
| Visualizzazione mappe precedenti | Bassa | Sì |
| Inserimento città (app gestore) | Bassa | No |
| Inserimento luoghi (app gestore) | Bassa | No |

## **Analisi dell’architettura**

Per garantire modularità, manutenibilità e scalabilità, il progetto è stato progettato seguendo un'architettura ben definita, in grado di separare logicamente le diverse componenti del sistema. In questa sezione vengono illustrati i principali aspetti architetturali attraverso due diagrammi.

**Deployment diagram**

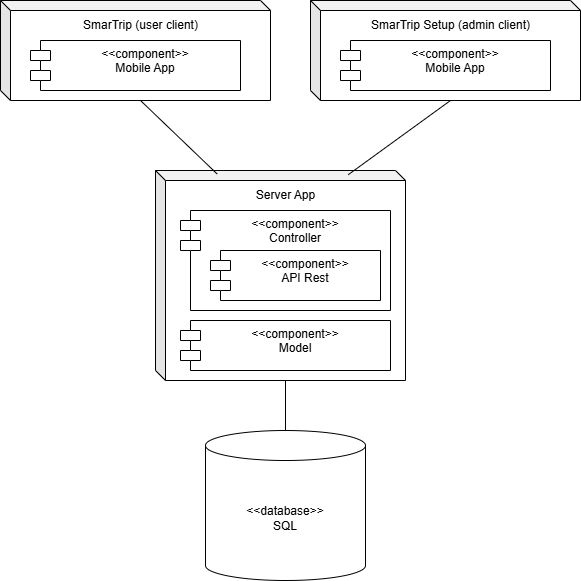
Il diagramma rappresenta la distribuzione fisica delle componenti software e le modalità con cui queste interagiscono all’interno dell’infrastruttura. In particolare, si evidenzia la separazione tra l’applicazione mobile, che funge da interfaccia utente, e il server, responsabile della logica applicativa. Quest’ultimo espone le API necessarie alla comunicazione con il client e gestisce i modelli dei dati, che vengono recuperati e manipolati attraverso un database locale. Tale organizzazione permette una chiara divisione delle responsabilità e facilita l’evoluzione indipendente delle diverse componenti del sistema.

Figura 2: deployment diagram.

**MVC diagram**

Il diagramma MVC (Model-View-Controller) illustra la suddivisione logica delle responsabilità all’interno del sistema. Nel nostro progetto è stata adottata una variante semplificata del modello tradizionale, strutturata come segue:

Alla base si trova il Database, che rappresenta la fonte persistente dei dati. Il livello Model, situato sul server, contiene le logiche di accesso ai dati e interagisce con il database tramite funzioni che chiamano le query presenti nel progetto Database. Il Controller, anch'esso parte del server, gestisce la logica applicativa e si occupa di elaborare le richieste ricevute, richiamando le funzioni del Model. Infine, la View è rappresentata dall’app mobile, che comunica con il Controller attraverso chiamate API. Questa separazione consente una gestione ordinata del flusso dati e favorisce la scalabilità e la manutenibilità del sistema.

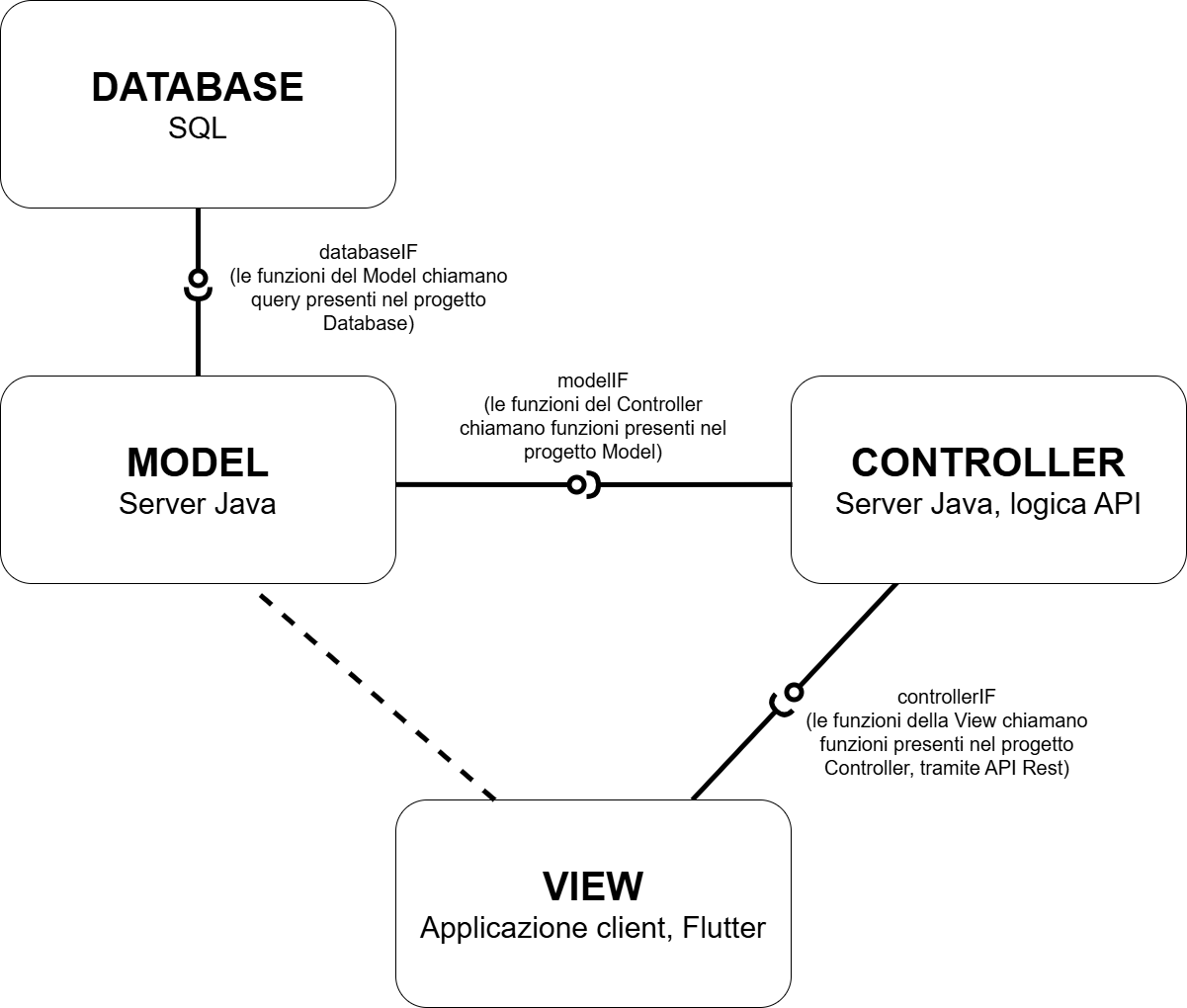


Figura 3: MVC diagram.

# **Iterazione 1**

## **Aggiornamento dei casi d’uso**

* UC1: LOG IN / SIGN IN. Eliminazione variabile email. Non è necessario l’utilizzo di una mail per il login, in quanto non viene utilizzata in alcun modo.

POSTCONDIZIONI: l’utente può eseguire nuovamente l’accesso all’app inserendo username e password.

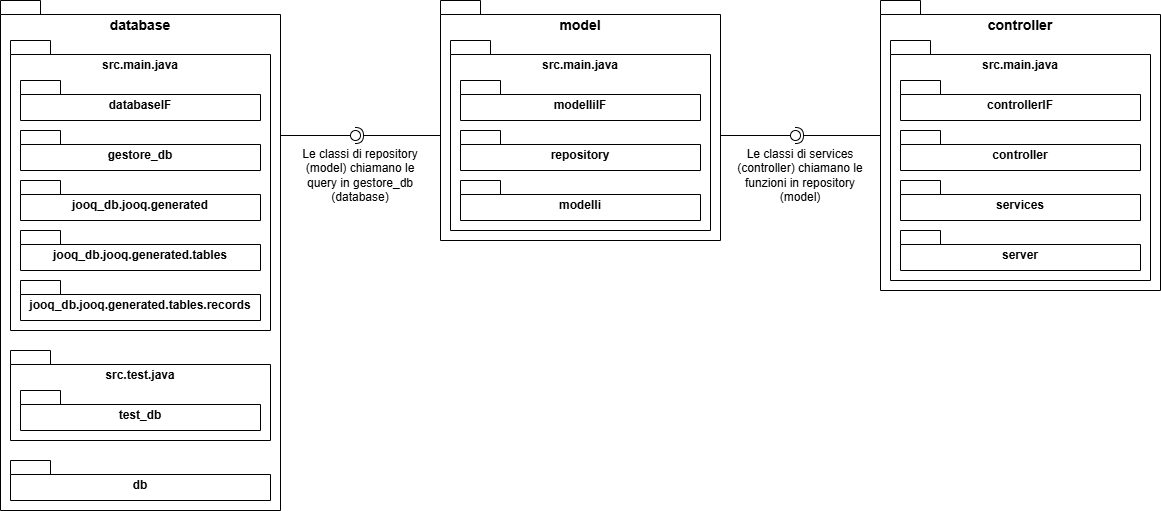
* UC2 e UC3. I presenti casi d’uso restano invariati, è stata presa la decisione di non implementarli nella presente versione dell’app; il ruolo di “Ente” verrà ricoperto dal team di sviluppo, che si occuperà di popolare il database dei luoghi.
* UC4 – UC7. I presenti casi d’uso restano invariati.
* UC8 (NUOVO): SELEZIONE MAPPA. L’utente visualizza i viaggi che ha pianificato in precedenza, scegliendo se visualizzarne nuovamente uno o se crearne uno nuovo. (Per comodità i piani di viaggio sono identificati come “mappe” o “itinerari”).

PRECONDIZIONE: l’utente deve avere eseguito il log-in all’applicazione.

POSTCONDIZIONE: se l’utente decide di creare una nuova mappa, vengono avviati i casi d’uso da 4 a 7.

## **Architettura del server**

In questa iterazione ci si concentra sulla progettazione e sulla scrittura del codice, partendo dal progetto server Java in Eclipse IDE.

**Package diagram**

Come mostra il diagramma dei pacchetti, l’applicazione è suddivisa in tre progetti Maven separati: database, model e controller, collegati tra loro tramite dipendenze dichiarate nel pom.xml.

* database: contiene il pacchetto gestore\_db, che include le classi responsabili della creazione del database e dell'esecuzione delle query (utilizzate dalle API); il pacchetto databaseIF, con le interfacce delle classi di gestore\_db; i pacchetti generati automaticamente da jOOQ; test\_db, per i test JUnit delle query; e infine db, che conterrà i file del database principale e di quello di test, al fine di evitare la modifica involontaria dei dati originali.
* model: comprende i pacchetti modelli, che contengono le classi che rappresentano gli oggetti del dominio (ad esempio Utente, Luogo, ecc.); repository, con le classi annotate con @Repository di Spring Boot, le quali si interfacciano con il database invocando le query; e modelliIF, che contiene le interfacce dei repository.
* controller: include services, con le classi annotate @Service di Spring Boot, responsabili della logica delle API e dell’interazione con il progetto model; controller, con le classi annotate @RestController, che gestiscono le richieste HTTP e l’invio delle risposte API; controllerIF, con le interfacce dei controller; e infine server, che conterrà la classe per l’avvio del server.

## **Database**

Bla bla