

# Ingegneria del Software

## Implementazione di un sistema di prenotazione voli

1066721 Alessandro Belotti

1065352 Andrea Salvi

1064464 Cristian Tironi





# GESTIONALE PER VOLI

Il progetto riguarda l'implementazione di un **sistema per la gestione dei voli** di una compagnia aerea, utilizzato sia da un operatore sia dagli utenti.

In particolare **l'operatore** si interfaccia con il sistema per:

- effettuare il login
- inserire nuovi voli
- visualizzare i voli disponibili

**L'utente** utilizza il sistema per:

- registrarsi
- effettuare il login
- prenotare il volo
- controllare lo storico prenotazioni
- visualizzare i voli disponibili

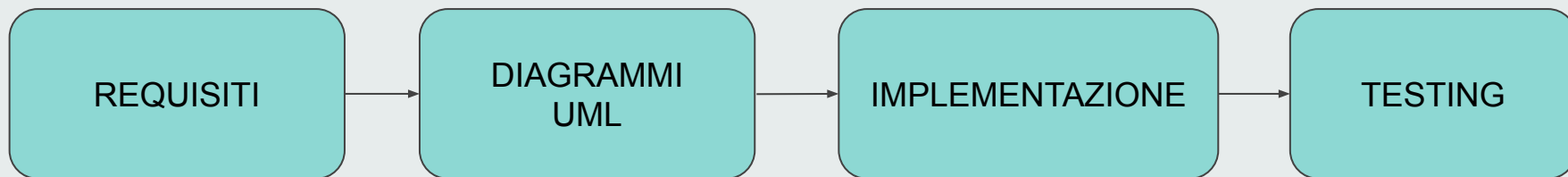


# Software life cycle

Il team ha adottato il modello organizzativo **AGILE** in quanto ogni membro era chiamato a partecipare in modo attivo alla realizzazione del progetto, senza quindi adottare specifiche gerarchie. Si è scelto un modello di sviluppo **Incrementale**.

Per ogni versione si analizzavano e stilavano i requisiti fondamentali da tutti i componenti del gruppo, sulla base di questo sono poi stati prodotti i diagrammi uml. Terminato ciò si è passato al coding e infine al testing del risultato risolvendo in caso di eccezioni.

Le versioni sono state ricavate tramite la tecnica **Clone and Own**.





# Software configuration management

Come **Configuration Control Board** è stato usato GitHub:

- per ogni versione del progetto è stata creata una **Project** con template **Kanban**
- le maggiori funzionalità sono state assegnate ad 1 o più membri del team tramite **Issue**
- i possibili stati di avanzamento delle Issue sono state **to Do, In Progress e Done**
- per ogni versione è stato creato un feature branch (**fb-prima** e **fb-seconda**) di sviluppo, contenente oltre al codice i diagrammi UML e documentazione
- il branch **develop** è stato utilizzato in fase di sviluppo per effettuare il merge del codice scritto dai vari membri del teams
- il branch **main** contiene la versione finale del software, la documentazione e tutti i diagrammi UML

🔒 **Prima versione**

Updated 22 days ago

0 To do

+ ...

0 In progress

+ ...

7 Done

+ ...

✓ **Spring Database** ...

#11 opened by Tironi

bug

invalid



• **Creazione funzione per il cliente :  
ControlloStorico** ...

#15 opened by AleBeloUnibg

enhancement



• **Creazione funzione per l'operatore :  
InserisciVolo** ...

#12 opened by AleBeloUnibg

enhancement



• **Aggiunta assegnamento automatica  
del posto nella EffettuaPrenotazione** ...

#21 opened by AleBeloUnibg

enhancement





# People management and team organization

L'organizzazione delle persone è avvenuta tramite contatto telefonico per organizzare dei meeting nei quali grazie alla piattaforma **Teams** abbiamo condiviso e comunicato durante tutto il processo di sviluppo.

Lo storico del lavoro effettuato e delle ore fatte da ogni persona è stato aggiornato ad ogni commit effettuato.

La stesura dei requisiti e l'implementazione dei diagrammi UML durante la fase di design è stata fatta alla **presenza di tutti** e 3 i membri del gruppo.

Prima di passare all'implementazione sono stati fatti dei meeting per **prendere confidenza** con il framework Spring (guardando video e documentazione ufficiale).

Successivamente le funzionalità sono state **implementate singolarmente** e in autonomia da ogni membro del gruppo, in caso di problemi durante il coding abbiamo sempre risolto lavorando in team.

Durante le riunioni ognuno di noi avvisava gli altri della funzionalità che avrebbe implementato aggiornando la kanban board su Github.

## PRIMA VERSIONE



|         | UML                              | MODEL                 | VIEW      | CONTROLLER | TESTING            |
|---------|----------------------------------|-----------------------|-----------|------------|--------------------|
| Belotti | Sequence-<br>Activity<br>Diagram | Entity<br>-Repository | Cliente   | Cliente    | Testing<br>manuale |
| Salvi   | Use case<br>Diagram              | Repository            |           |            | Testing<br>manuale |
| Tironi  | Class<br>Diagram                 | Entity<br>-Repository | Operatore | Operatore  | Unit<br>Testing    |

## SECONDA VERSIONE



|         | UML  | MODEL                  | VIEW        | CONTROLLER  | TESTING         |
|---------|--|------------------------|-------------|-------------|-----------------|
| Belotti | Class-<br>Sequence-<br>Activity<br>Diagram | Entity -<br>Repository | Operatore   | Operatore   | Unit<br>Testing |
| Salvi   | Use case<br>Diagram                        | Entity                 | Co-presenza | Co-presenza |                 |
| Tironi  |  |                        | Cliente     | Cliente     | Unit<br>Testing |





# Software quality mcCall

Gli attributi di qualità del software che sono stati adottati sono:

- **correttezza** : i requisiti funzionali richiesti del cliente sono stati implementati senza errori, le operazioni che operatore e cliente possono fare sono state largamente testate
- **usabilità** : l'interfaccia è stata realizzata in modo da garantire un utilizzo pratico e semplice al cliente. Inoltre viene sempre mostrato l'esito dell'operazione appena svolta dall'utilizzatore
- **flessibilità** : la struttura utilizzata dal framework Spring permette di effettuare senza problemi modifiche e nuove implementazioni in futuro
- **portabilità** : il software è stato usato su differenti macchine da ogni membro del team di sviluppo senza problemi



# Requirements engineering

|                    |   |
|--------------------|---|
| <u>Must Have</u>   | effettua prenotazione, controllo storico, login operatore e cliente, registrazione cliente e controllo voli |
| <u>Should Have</u> | registrazione operatore   |
| <u>Could Have</u>  | inserimento aerei, prezzo biglietto diverso a seconda del peso  |
| <u>Would Have</u>  | implementazione servizio di car sharing associato all'aeroporto   |

La scelta del **prezzo** totale è stata fatta in fase di sviluppo:

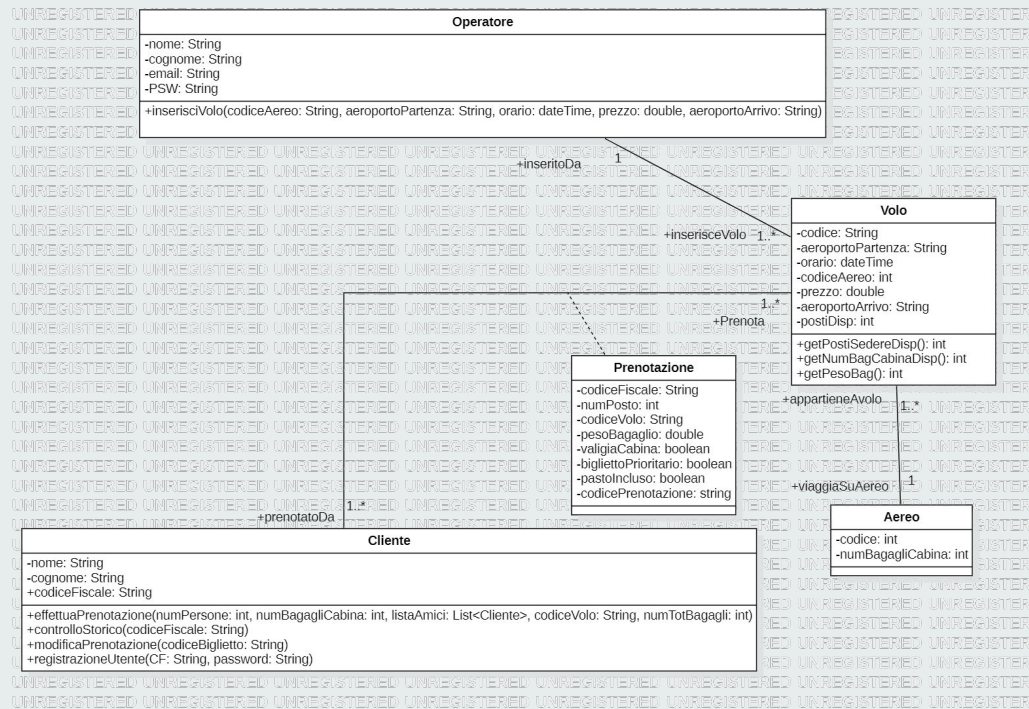
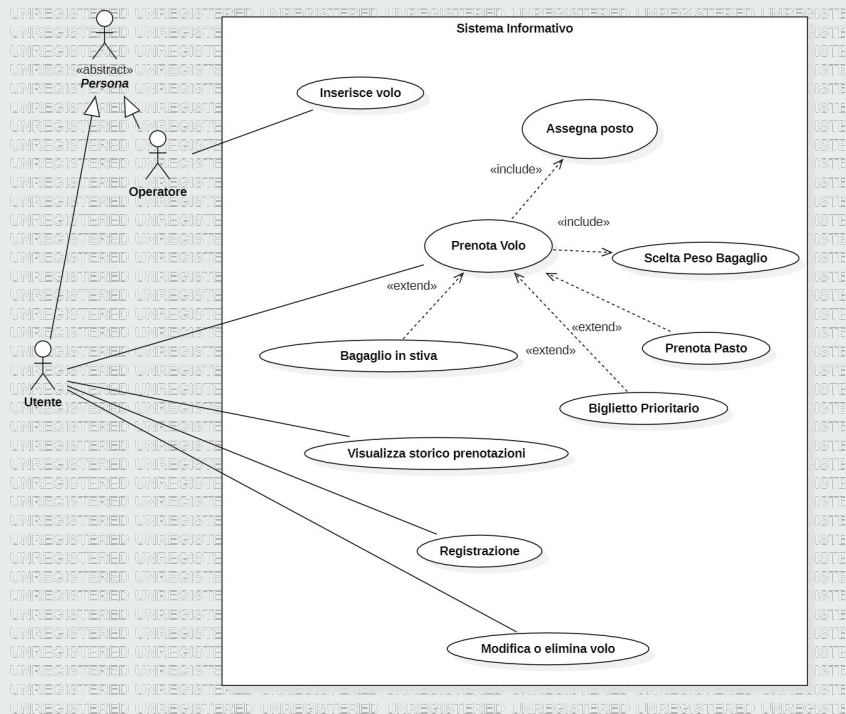
- il prezzo base del biglietto è scelto dall'operatore nel momento dell'inserimento del volo.
- Il prezzo del pasto è fisso a 10 €, mentre il biglietto prioritario costa il 30% in più del prezzo base del volo.



# Modelling

- **Use Case Diagram**  
Lo Use case Diagram modella le possibili interazioni tra gli utenti del sistema ovvero Operatore e Cliente.
- **Class Diagram**  
Scomposizione di tutte le possibili classi del sistema
- **State machine diagram**  
Descrive gli stati del sistema:
  - Prenotazione
  - VisualizzaStorico
- **Sequence Diagram**  
Vengono rappresentati i sequence diagram relativi alla Prenotazione, all'Inserimento Voli e al Controllo storico
- **Activity diagram**  
Riprende le azioni di Prenotazione per l'Utente e Aggiunta Volo per l'Operatore

# Modelling





# Software architecture



Il progetto è stato realizzato utilizzando il framework Spring, che prevede l'organizzazione dei moduli software secondo il **pattern MVC**:

- il package **com.pack.aeroporto.controller** contiene i metodi di controller per utente e operatore
- il package **com.pack.aeroporto.entity** contiene le entità, ossia tutte le classi implementate
- il package **com.pack.aeroporto.repository** rappresenta i repository, permettono lo scambio di dati dal database all'applicazione
- in **resources** sono contenute le pagine HTML che costituiscono le view del sistema
- 

Inoltre il team ha seguito i seguenti standard durante l'implementazione:

- I nomi delle classi sono definiti in Pascal Case (NomeClasseProva).
- I nomi dei metodi sono definiti in Camel Case (nomeMetodoProva).
- I nomi dei package sono definiti in Snake Case (nome\_package\_prova).
- I nomi delle variabili sono definiti in Camel Case (nomeVariabileProva).



# Software Design

Il sistema di gestione voli è stato implementato usando i seguenti linguaggi e risorse:

- per il front end **HTML** e **CSS**
  - il back end è stato sviluppato in linguaggio **Java**
  - il progetto è realizzato a partire dal framework **Spring ( Web, JPA, Thymeleaf )**
  - il database è implementato utilizzando **MySQL** e l'interazione con esso avviene tramite **phpMyAdmin**
  - il testing automatico è stato fatto con **Junit** ed **Eclemma**
- 
- E' stato implementato il pattern: **Singleton** durante il testing



# Software testing and maintenance


I casi di test implementati automaticamente con **Junit** sono stati fatti per la effettuaPrenotazione():







- Richiamo del template HTML
- La prenotazione esaurisce i posti disponibili sul volo
- Non ci sono voli disponibili
- Il cliente che prenota il volo è già presente nel DB
- Il cliente che prenota il volo non è presente nel DB
- Verifica che sia stato selezionato un Volo e che non sia stata svolta una POST sospetta

Per i nomi dei Test sono stati utilizzati dei pattern specifici descritti nel manuale (The Art of Unit Testing: With Examples in C#) : *(before/after)NomeMetodoTest\_risultatoAtteso*

Le restanti funzionalità sono state testate manualmente dal membro del team che le ha sviluppate (vedere documentazione).

# Software testing and maintenance

▼  ClienteControllerTest [Runner: JUnit 5] (0,731 s)

-  effettuaPrenotazioneTest\_nessunVolo() (0,679 s)
-  postEffettuaPrenotazioneTest\_voloNonPresente() (0,004 s)
-  postEffettuaPrenotazioneTest\_esitoAffermativo() (0,014 s)
-  effettuaPrenotazioneTest\_tornaTemplate() (0,008 s)
-  postEffettuaPrenotazioneTest\_clienteNonPresente() (0,012 s)
-  effettuaPrenotazioneTest\_removeVolo() (0,014 s)

| Element   | Coverage   |
|---|--|
| ▼  Aeroporto                     |  59,7 %   |
| ▼  src/main/java                 |  44,4 %   |
| >  com.pack.aeroporto.controller |  40,0 %   |
| >  com.pack.aeroporto.entity     |  55,6 %   |
| >  com.pack.aeroporto.object     | 45,0 %   |
| >  com.pack.aeroporto           | 0,0 %  |
| >  src/test/java               |  91,0 % |

```
int postiValigieCabinaOccupati = numeroPostiDisponibili(prenotazione);

//salvataggio prenotazione

Prenotazione input = new Prenotazione();
input.setCodiceFiscale(clienteResult.getCodiceFiscale());
input.setCodiceVolo(prenotazioneResult.getCodiceVolo());
input.setNumPosto(postiValigieCabinaOccupati + 1);
input.setPasto(prenotazioneResult.isPasto());
input.setBigliettoPrioritario(prenotazioneResult.isBigliettoPrioritario());
input.setValigiaCabina(prenotazioneResult.isValigiaCabina());

double costo = v.getPrezzo();
int pasto = prenotazioneResult.isPasto() ? 1 : 0;
int prioritario = prenotazioneResult.isBigliettoPrioritario() ? 1 : 0;
costo += pasto * 10 + prioritario * 0.3 * v.getPrezzo();
input.setPrezzo(costo);

prenotazioneRepo.save(input);

model.addAttribute("esito", true);
model.addAttribute("status", "Prenotazione andata a buon fine");

return "/cliente/esitoPrenotazione";
```





# Refactoring

Per migliorare la struttura e la leggibilità del codice è stato fatto del refactoring:

- i casi di test automatici che prendono i dati inseriti nella form dall'utente sono stati **rinominati** come `afterEffettuaPrenotazione....`
- i casi di test automatici che non prendono i dati inseriti nella form dall'utente sono stati **rinominati** come `beforeEffettuaPrenotazione....`
- **cancellazione** di classi inutili (`operatoreDTO`, classi test di operatore e aeroporto, template footer e header)