Relazione progetto Ingegneria del Software 2019-2020

Compagnia Aerea, Gruppo 1

Introduzione . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

Analisi dei Requisiti e Fattibilità . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .2

Requisiti Cliente . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

Analisi fattibilità. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3

Scelte strutturali e tecnologie . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

MVC Pattern . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .4

Pianificazione tempi e distribuzione risorse . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .5

Sviluppo progetto e difficoltà riscontrate . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .6

Sviluppo e Pattern usati. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .6

Funzionalità Mancate . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .7

Conclusioni . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

**Introduzione**

Relazione progetto Ingegneria del Software 2019-2020

Compagnia Aerea Gruppo 1

“Triple A Flights”

Documentazione relativa al progetto di una compagnia aerea fittizia, contestualmente ad un applicativo di gestione acquisto biglietti da parte dei clienti di tale compagnia.

Diagrammi di riferimento allegati nella directory Documentazione del file di progetto.

Membri del gruppo:

*Restelli Andrea*

*Feltrin Andrea*

*Calaminici Andrea*

**Analisi dei Requisiti e Fattibilità**

**Requisiti Cliente**:

Il sistema realizzato dovrà gestire diverse funzionalità secondo i casi d’uso del diagramma sotto riportato, le cui funzionalità rispetto ai vincoli cliente sono, in ordine di priorità:

* Consulazione da parte dell’utente della lista voli, secondo vari parametri
* Comprare biglietti relativi ad un volo
* Possibilità di Login di un account esistente
* Creazione Account registrato (Fedeltà) e relativi benefici
* Pagamento dei servizi tramite Carta di Credito
* Prenotare biglietti relativi ad un volo per utenti Fedeltà
* Feature punti Fedeltà cumulabili e spendibili da utenti Fedeltà per acquisto biglietti
* Consultazione informazioni personali di account e punti Fedeltà accumulati
* Possibilità di cancellare un biglietto o prenotazione
* Routine di controllo scadenza prenotazioni con relativi avvisi
* Routine di controllo promozioni nuove con relativi avvisi
* Routine di controllo Infedeltà e relativi avvisi
* Consultazione promozioni relative a voli
* Supporto per funzionalità di Cambio date o orario biglietto, e relativa differenza monetaria

Vincoli e specifiche cliente:

* Possibilità di comprare più biglietti da uno stesso utente
* Vincolo di singola prenotazione
* Promozioni differenziate relative a voli specifici, finestre di tempo, e su tipologia account cliente
* Vincoli di tempo su scadenze prenotazioni (3 giorni dal volo), infedeltà (3 anni senza acquisti)
* Vincoli di tempo su avvisi routine prenotazioni (24h da scadenza) e infedeltà (2 anni senza acquisti)

**Analisi Fattibilità**

L’insieme delle features richieste di alta priorità e di importanza critica sono svolgibili agevolmente con una buona ripartizione delle risorse (in dettaglio nella prossima sezione).

Alcune fra le richieste ausiliarie (di bassa priorità) si dimostrano più complesse e verranno accodate nello sviluppo del progetto relativamente alla possibilità di un’implementazione in dettaglio.

Ad esempio, gli avvisi via mail per le routine di controllo denotano una certa complessità e una base di competenze richieste di cui lacuniamo, e richiedono l’implementazione di componenti complessi o prefabbricati per una funzione secondaria (in merito alle tecnologie scelte, in dettaglio nella prossima sezione).

Per queste ragioni, abbiamo dato priorità in ordine come notato sopra e sviluppano il progetto partendo dalle funzionalità più critiche, per poi passare a quelle più leggere e secondarie, ed accodando quelle più laboriose ma ausiliarie, rispetto ai vincoli di tempo.

Alcune richieste erano lasciate ad interpretazione:

* La questione punti fedeltà è stata gestita in modo che, dati X€ di spesa biglietto, un cliente accumuli X/10 punti. Un biglietto di costo Y è acquistabile con Y punti.
* Siccome l’iscrizione Fedeltà coincideva con le normali informazioni account, abbiamo assunto che un cliente registrato sia Fedeltà, ed uno non registrato non lo sia. Da cui, la possibilità di comprar biglietti da parte di chiunque, mentre prenotazione, edita e cancella biglietto solo dagli utenti registrati (fedeltà)

**Scelte strutturali e tecnologie**

Un incontro preliminare all’avvio effettivo del progetto, con conseguente brainstorm e valutazione delle alternative tecnologiche, che ci ha portato a decidere quali scelte strutturali e tecnologiche seguire.

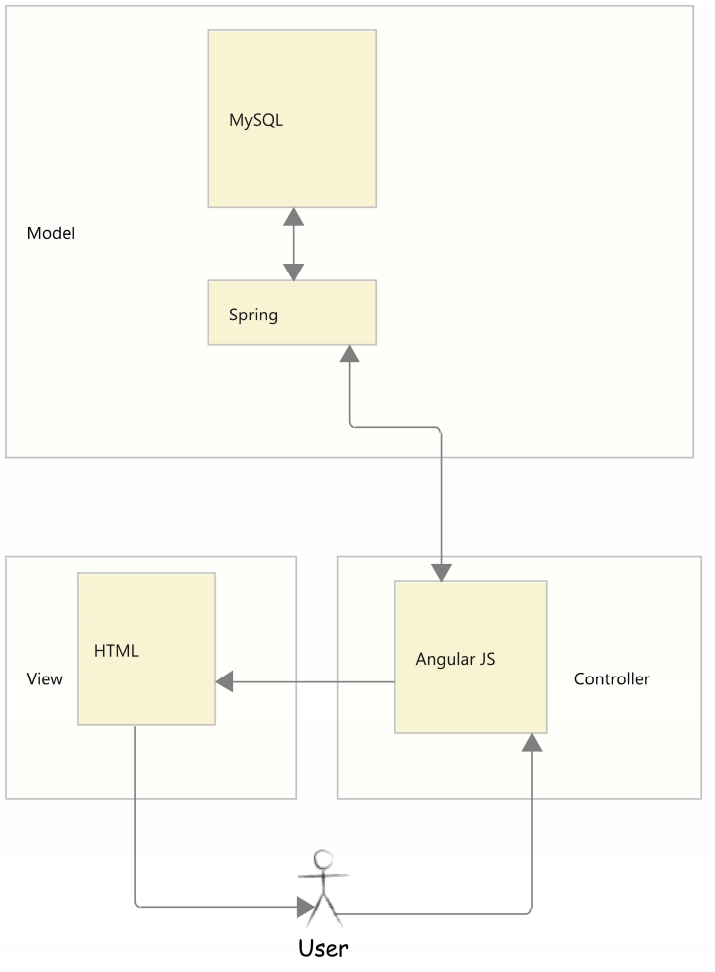
Come proposto da Andrea Restelli, che già lavorando nell’ambio informatico ha le conoscenze acquisite maggiori del gruppo, abbiamo optato per la realizzazione di un sito web per implementare il sistema di gestione, che ci ha permesso la maggior flessibilità e miglior implementazione delle richieste.

Dopo aver valutato varie alternative, abbiamo concordato per un front end di un sito HTML , gestito da Angular, un framework open-source in Typescript (JS) tramite l’IDE IntelliJ, accompagnato dal framework Java SpringBoot per una gestione backend dei servizi necessari, ed infinite un database SQL da cui attingere per le query necessarie per le funzionalità sopracitate.

La combinazione di linguaggi HTML, Typescript, Java e SQL si è rivelata quella con maggior flessibilità per le nostre necessità, e che facendo leva sulle conoscenze acquisite nel nostro percorso di studi, ci ha permesso di coordinarci in modo migliore grazie ad una base di conoscenza comune fra i membri del gruppo.

Altri tool di accompagnamento ai linguaggi sono stati usati, come Bulma per la gestione modulare di CSS, e Hybernate, un middleware open-source per lo sviluppo di applicazioni Java, attraverso il quale ci fornisce un servizio di mapping per la persistenza dei dati sul database, e Maven, che come strumenti di appoggio per buildare il progetto, consolida le dipendenze del progetto e vidima il codice rendendolo più facilmente usabile e coerente, MySQLWorkBench per la gestione di alcune funzionalità back-end come la creazione di istanze DB per poter operare le query utente, e Git per gestire la programmazione concorrenziale ed ovviare alla difficoltà di doversi trovare fisicamente spesso per aggiornarci sul progresso del progetto.

**Design Pattern**

Nel diagramma dell’architettura allegato (qui riportato) abbiamo indicato solo i framework e i macrolinguaggi usati, che costituiscono il cuore della struttura del progetto.

Come si intuisce dal diagramma, abbiamo seguito un **Design Pattern MVC** (Model View Controller).

Questo pattern architetturale ci ha permesso di strutturare logicamente il nostro progetto in maniera più chiara ed efficace, e l’abbiamo scelto perché ritenuto il più affine nel gestire un Utente che, in base al caso d’uso, possa richiedere o inviare informazioni che devono essere gestite dal sistema “dietro le quinte”, a prescindere dall’interfaccia in evidenza. Avendo la necessità di gestione dell’utente e dei dati in input/output anche in relazione ad un back-end dinamico, il modello MVC ci permette di implementare funzionalmente le features richieste lasciando all’utente un’interfaccia minimale, intuitiva e funzionale, mentre il controllore Java è in grado di comunicare con il back-end mascherando le informazioni non utili all’utente, abbinata poi ad una maggior facilità di utilizzo da parte del cliene, e di gestione modulare del codice da parte nostra.

A livello di codice, abbiamo suddiviso in due macro-cartelle il progetto: server-side, e client-side. Nel primo si trova il backend ed i controller, dove chiamate a database ed input da parte dell’utente vengono gestite a livello Java, e comprende la gestione a livello software come descritta nei casi d’uso.

Nella directory client-side si trovano le pagine html, css e typescript per gestire il front-end e l’aggancio (tramite API) agli altri componenti. Grazie al framework di angular abbiamo potuto rendere modulari diversi componenti ed utilizzarli senza duplicare codice, ma riusando ed estendendo parti di codice che altrimenti sarebbero stati comuni a più componenti. Tramite Angular le query utente da front-end vengono poi passate al controllore java e gestite per poi, a cascata, eventualmente chiamare il database.

**Pianificazione tempi e distribuzione risorse**

La metodologia di sviluppo da noi adottata è stata lo Scrum, metodologia agile che ci ha permesso più flessibilità evolvendo una base funzionante di programma, che è stata settimanalmente arricchita gradualmente da nuove funzionalità.

A differenza dello Scrum classico, i nostri meeting non sono stati quotidiani ma settimanali, intervallati da sprint di implementazione delle funzionalità prefissate, ed un procedimento iterativo che ci ha permesso di progredire accodando eventuali problemi in base a criticità e complessità di implementazione.

Come Scrum Master, Andrea Restelli ed Andrea Feltrin han collaborato per supervisionare, aggiornare e prefissare i prossimi obiettivi di meeting in meeting.

Come backlog, l’insieme di prefissi settimanali seguendo casi d’uso e lista prioritaria di richieste e vincoli cliente, è bastata per permetterci di suddividere i nostri sforzi nell’arco della durata del progetto.

Dopo una riunione preliminare di strutturazione progetto, abbiamo suddiviso i vari compiti in base alle conoscenze pregresse di ciascun membro e alle nostre competenze.

A livello di distribuzione risorse, avevamo pianificato di suddividere così i nostri sforzi:

- Andrea Restelli: Setup angular e build iniziale progetto, parte del back-end Java, e supervisione generale del progetto.

- Andrea Feltrin: Parte del front-end HTML, CSS, parte del back-end Java, e popolamento Database, parte dei diagrammi.

- Andrea Calaminici: Parte del front-end, API, documentazione, parte dei diagrammi.

Suddivisione tempistiche come da diagramma Gantt allegato al progetto.

Causa di alcuni inconvenienti sul percorso di sviluppo, e poco riscontro da parte di un membro del gruppo, abbiamo infine così lavorato:

- Andrea Restelli: Setup angular e build iniziale progetto, TUTTO il back-end Java, API e supervisione generale del progetto.

- Andrea Feltrin: TUTTO il front-end HTML, CSS, parte del back-end Java, popolamento Database, parte dei diagrammi, documentazione

- Andrea Calaminici: parte dei diagrammi.

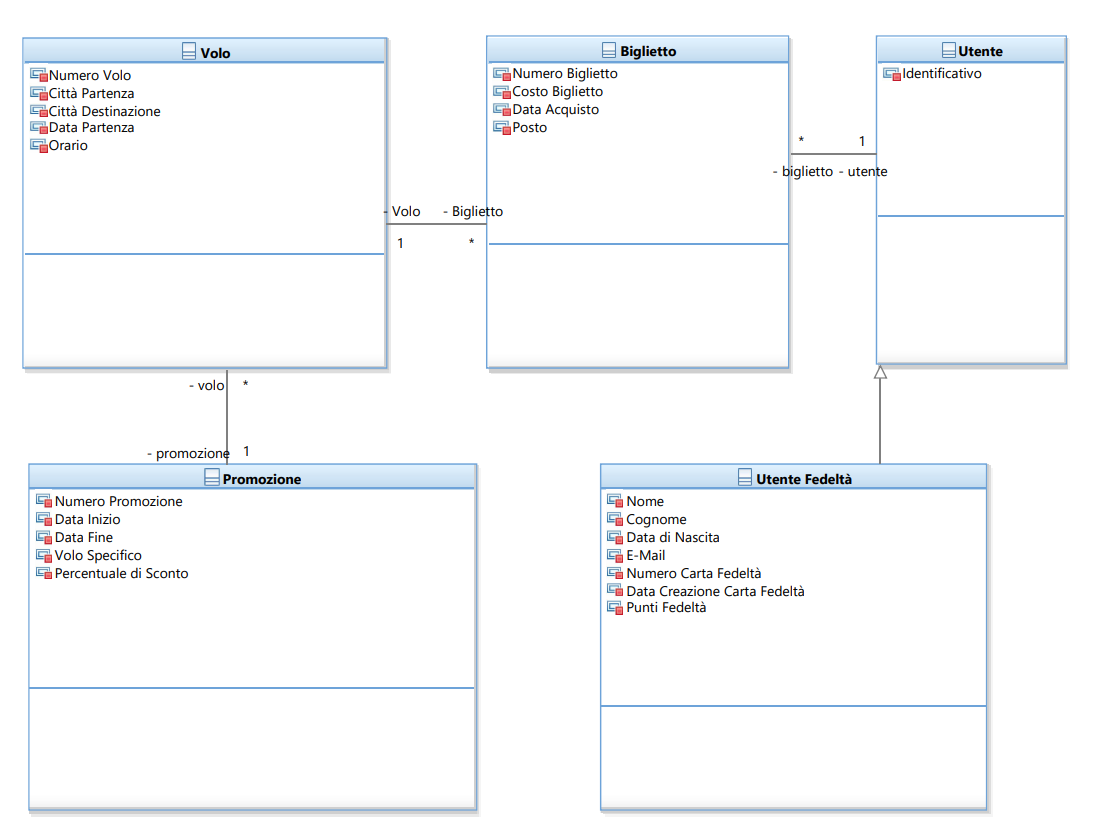
**Sviluppo progetto e difficoltà riscontrate:**

**Pattern**

Lo sviluppo del progetto è continuato seguendo alcuni design pattern (anche grazie a Springboot che è predisposto alla creazione codice tramite questi pattern):

- Inversion of control (IOC): siccome Spring ha un container IOC, tramite il quale crea e gestisce gli oggetti creati, è stato abbastanza semplice impostare gli oggetti codificati in modo che le proprie dependencies vengano usate anche da altri oggetti. Questo ci ha permesso di alleggerire il carico complessivo di codice e ridurre l’accoppiamento degli oggetti singoli.

- Factory Design: l’utilizzo di questo pattern è derivato principalmente dalla necessità: per gestire in modo fluente i passaggi API fra diversi blocchi di codice differente, passare oggetti non tipizzati (eccetto in alcune particolari interazioni), per poi gestire l’oggetto generico e ri-classificarlo tramite un metodo a sé stante per ri-definite l’oggetto passato e renderlo utilizzabile dal codice in questione. In modo specifico, la generazione di oggetti .json da DB passati tramite API a livello superiore, per poi essere ridefinito in base all’utilizzo richiesto, rispecchia appieno le nostre esigenze e segue in modo fedele questo tipo di Pattern.

- Singleton: l’unicità di un tipo di oggetto per ciascuna istanza dell’esecuzione del progetto è tanto utile quanto importante: essendoci solo un utente loggando alla volta, una sola query ricerca voli attivo per volta, ed una sola query di creazione/modifica biglietto, l’utilizzo di multipli oggetti dello stesso tipo avrebbe appesantito troppo il codice, ed avrebbe portato informazioni ridondanti. Per ciascun strato, un solo tipo di oggetto viene passato e gestito.

Tramite i diagrammi allegati di casi d’uso, stati e classi di dominio (pagina precedente), è possibile comprendere in maniera sintattica come funziona il codice scritto. A causa di una difficoltà programmativa abbiamo dovuto deragliare dall’iniziale struttura di Classi ed abbiamo dovuto aggiungere una classe di appoggio FideltyCard per gestire alcuni parametri intermedi, e funzioni di appoggio per gestire alcune convalide, che altrimenti avrebbero dato conflitto con il voler seguire il Factory Design pattern. Per questo motivo potremmo considerarlo un Anti-pattern, anche se generalmente ha portato più benefici che problemi.

**Funzionalità Mancate e problemi riscontrati**

Il più grande problema che ci ha impedito di consegnare un progetto completo è dovuto ad un membro del gruppo che non ha rispettato scadenze e compiti assegnati, obbligando così gli altri ad appesantire il proprio carico di lavoro. Essendo studenti lavoratori il vincolo di tempo ha gravato enormemente, ed il lavoro aggiunto ci ha introdotto difficoltà inaspettate che non siamo riusciti a risolvere in tempo.

Un altro problema che ci ha rallentato è stato un problema di congruenza nel riconoscere le tipologie di Date, passate da SQL a Java ed infine a TS, che ci ha spinto ad optare per diversi metodi risolutivi del problema e riscrivendo parte del codice.

A causa del poco tempo, generale inesperienza e del grande carico di lavoro gravante solo su due persone, nonostante le richieste principali cliente sono funzionanti, le seguenti features non sono complete:

* Modifica biglietto/prenotazione mancante causa vincoli di tempo
* Alcuni problemi API riguardo al fetch utente, e quindi alcune funzionalità annesse.
* Invio Mail non implementato, causa inesperienza nostra nell’ambito e scarso tempo per cercare altre soluzioni.
* E’ possibile comprare solo un biglietto alla volta per utente.
* Ricerca voli data Partenza vuota ritorna query nulla.
* Non c’è stato tempo per utilizzare Sonar.
* Il sistema avvisa per scadenza prenotazione solo 3 giorni prima dal volo, non 24h prima di scadenza
* Assenza avviso Infedeltà: dopo i 2 anni, l’utente viene declassato immediatamente senza avviso.
* Minuzie grafiche HTML

**Conclusioni**:

Personalmente ho molto apprezzato il doverci cimentare in questo tipo di progetto come “esame alternativo”, ho avuto modo di imparare molto e forzarmi a trovare soluzioni fronte a problemi programmativi inaspettati. Mi dispiace non siamo riusciti a completare il progetto, ma l’insieme di poco tempo e di assenza di contributo da parte di uno dei membri è stata fatale nello sviluppo del progetto nelle ultime sue fasi.

Un grande freno al progetto di marcia è stata l’inesperienza nostra di fronte ad un progetto cui non avevamo abbastanza conoscenze acquisite, che ci han forzato a spendere tempo nell’imparare di fretta l’utilizzo di tool e linguaggi con cui non avevamo troppa dimestichezza, ma in conclusione siamo soddisfatti degli sforzi che abbiamo impiegato nonostante le nostre limitazioni, e siamo sicuri che avendo avuto piu’ tempo, o una persona in più saremmo riusciti a completare il progetto per intero.