Documentazione

Organizzazione Del Personale **2**

Specifica dei requisiti (Standard IEEE 830) **2**

Introduzione 3

Descrizione Generale 3

Requisiti Specifici 4

Architettura del Software **4**

Stile Architetturale 4

Viste architetturali 5

Design Pattern **5**

Dependency Analysis 6

Diagrammi UML 7

Testing **8**

Organizzazione Tests 8

Refactoring 10

Conclusioni **10**

1. Organizzazione Del Personale

L’organizzazione del progetto viene gestita da tre persone: Davide Revrena, Andrea Trionfetti e Fabio Assolari, tutti responsabili del progetto e che si occuperanno di tutte le fasi del processo di sviluppo.
Essendo un numero limitato di persone e tutte aventi le stesse competenze e ruoli l’organizzazione non potrà essere rigorosamente gerarchica, quindi il team è organizzato secondo la metodologia ‘Agile’.
I membri lavoreranno a stretto contatto (con videochiamate) con delle numerose sessioni di ‘brainstorming’ per eludere tutte le criticità che si riscontrano durante lo sviluppo.

Un'organizzazione con caratteristiche molto simili a quella di SWAT (Skilled With Advanced Tools), vengono utilizzati diversi tools viene meno però la presenza di personale esperto e preparato necessario per gestire il gruppo di lavoro.

Il Team lavorerà congiuntamente sullo stesso computer per la parte della progettazione più critica, come la stesura dei requisiti, il design o la modellazione dell’architettura software, mentre per l’implementazione del codice e dei test dello stesso o la stesura della documentazione si agisce separatamente con degli obiettivi fissati dagli sprint con un periodo di 2-6 settimane.

2. Specifica dei requisiti (Standard IEEE 830)

2.1. Introduzione

2.1.1. Scopo

Questa sezione del documento ha lo scopo di indicare chiaramente i requisiti dell’applicazione **MyWallet** per la gestione del conto corrente bancario. Questa sezione sarà la base che guiderà le successive fasi dello sviluppo del prodotto, a partire da architettura e design.

2.1.2. Ambito

L’applicazione renderà più comodo e semplice gestire le quotidiane operazioni bancarie e di pagamento dei nostri clienti. Più dettagli nel punto 2.3 di questa sezione.

2.1.3. Definizioni

Utente: può essere un Cliente privato oppure un Azienda ad avere accesso al proprio conto bancario.
Transazioni: operazione di pagamento inviato o ricevuto da un altro utente.
Storico: lista aggiornata dal sistema di gestione, contiene le ultime transazioni effettuate in ordine cronologico.
Cash-Back: una quota percentuale che il cliente riavrà a seguito di un pagamento verso determinati esercenti.
Obiettivo Risparmio: un parte del saldo che viene ‘virtualmente’ spostato dal proprio conto per essere messo da parte come risparmio, questo non verrà contato del proprio

bilancio ma sarà in qualunque momento riportato sul conto dell'utente quando lo richiederà.

2.1.4. Panoramica

Il punto 2 di questa sezione fornisce una visione generale del sistema, il punto 3 ne dettaglia nello specifico i requisiti, divisi in varie categorie.

2.2. Descrizione Generale

2.2.1. Prospettiva del Prodotto

L'applicazione entra in un mercato già rodato e ricco di competitor, ma l'obiettivo è quello di fornire al cliente una sola applicazione funzionante e completa con cui gestire a 360 gradi il proprio conto in modo semplice e intuitivo.

2.2.2. Funzioni del Prodotto

Ha molte funzioni per gestire il proprio account bancario, oltre ai pagamenti si possono chiedere prestiti, ricevere Cash-Back, utilizzare il conto risparmio, consultare lo storico delle entrate e uscite e visualizzare il proprio saldo. Tutte le funzionalità sono state descritte in precedenza.

2.2.3. Caratteristiche degli utenti

Il software è pratico e utilizzabile da chiunque in modo intuitivo e semplice, gli utenti non necessitano, dunque, di alcun tipo di preparazione.

2.2.4. Vincoli

L'unico vincolo imposto dal sistema all'utente è quello di avere un saldo sufficiente per eseguire le operazioni di pagamento.

2.3. Requisiti Specifici

2.3.1. Interfaccia Esterna

L'interfaccia utente fornisce tutte le funzioni disponibili, in un unica schermata di 'home' oltre a quella di 'login/registrazione'.
Il software è disponibile per ora solo su Pc con un comune sistema operativo, preferibilmente Windows. È indispensabile una connessione internet.
Il programma verrà successivamente implementato anche per device mobile.

2.3.2. Requisiti Funzionali

2.3.2.1. Autenticazione

- Registrazione: vengono richiesti i dati del privato o dell'azienda, a seconda della scelta, per eseguire la registrazione (nome, cognome, CF, e altri dati anagrafici)
- Login: inserire 'nome utente' e 'password' scelte in fase di registrazione per accedere alla propria area personale

2.3.2.2. Area Personale

- Bilancio: riporta la disponibilità economica.
- Transazione: tasto per eseguire una transazione, riportare la cifra in € e gli estremi dell'account a cui effettuare il pagamento.

- Transazioni recenti: è una lista in ordine cronologico delle ultime transazioni eseguite e ricevute (le entrate saranno positive, le uscite negative).
- Risparmi: con il comando 'Invia denaro al conto risparmio' si spostano 'virtualmente' dal bilancio del proprio conto al fondo risparmio.

2.3.3. Requisiti sulle Performance

Non sono richieste risorse computazionali elevate per questo applicativo. È solo necessaria la connessione internet in quanto le operazioni vengono eseguite e registrate sui server e database.

3. Architettura del Software

3.1. Stile Architetturale

Con lo scopo di aumentare la qualità del software si è deciso di adottare uno stile architetturale di tipo '**Model-View-Controller**'.

3.1.1. Problema

Si vuole separare la UI (interfaccia utente) dal Database e dal software che gestisce tutte le operazioni. Ci permetterà di non ripeterci e in secondo luogo, aiuta a creare una solida struttura per la nostra applicazione.

3.1.2. Soluzioni

Svilupperemo la nostra applicazione seguendo questi 3 componenti architetturali.

- Il **Model** gestisce direttamente i dati, comunica con il database ed è fondamentale per fare in modo di registrare tutte le modifiche e di prelevare i dati da rappresentare.
- Il **View** che gestisce l'interfaccia utente che verrà visualizzata dall'utente. Questo componente comunicherà con **controller** per aggiornare i dati da visualizzare.
- Il **Controller** che comprende tutte le classi 'manager' che gestiscono le regole dell'applicazione prendendo i dati dal **model** e spiega come rappresentarli al **view**, serve anche per tutte le operazioni di upgrade/refresh

3.2. Viste architetturali

L'architettura del sistema è stata descritta in 6 diagrammi UML, che descrivono le viste architetturali dell'applicazione.

3.2.1. Logical view (class diagram, state machine diagram e sequence diagram)

Rappresenta le interazioni e le relazioni interne tra i vari componenti del sistema. I diagrammi di seguito descrivono una vista statica e dinamica dell'intero sistema.

3.2.1.1. Class Diagram

Viene usato un class diagram che comprende le classi implementate con relativi campi e metodi. Questo diagramma verrà successivamente utilizzato per generare il codice mediante StarUML.

nota: si faccia riferimento a "MyWallet\uml\images\Class-Diagram.png".

3.2.1.2. State Machine Diagram

Descrive il percorso di stati che fa l'utente dalla richiesta di registrazione alla scrittura su database dei dati inseriti, comprende la verifica degli stessi e l'interruzione in caso di errori.
nota: si faccia riferimento a "MyWallet\uml\images\State-Machine-Diagram.png".

3.2.1.3. Sequence Diagram

Illustra il processo di comunicazione tra utente, sistema di gestione e database per una transazione. L'utente invia una richiesta di pagamento, il sistema verifica la disponibilità di saldo, in caso positivo la transazione viene eseguita e visualizzata a schermo.
nota: si faccia riferimento a "MyWallet\uml\images\Sequence-Diagram.png".

3.2.2. Process view (activity diagram)

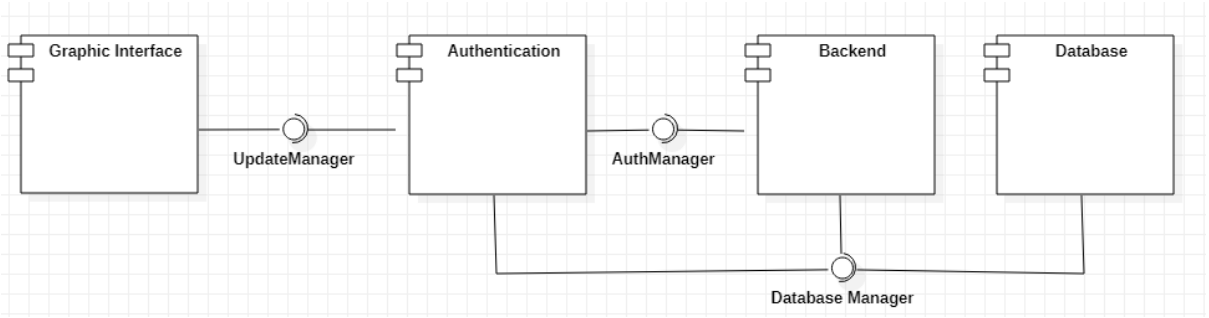
Mostra il comportamento delle parti del sistema, in determinate casistiche mostrando il flusso di lavoro del sistema. L'Activity Diagram rappresenta il processo di registrazione, interpretato da tre interpreti coinvolti, l'utente, il sistema di gestione ed il database manager, separati graficamente dalle partizioni.
nota: si faccia riferimento a "MyWallet\uml\images\Activity-Diagram.png".

3.2.3. Use Case View

Questa vista architetturale serve per mostrare le funzionalità che il sistema fornisce ai vari attori interpretati nel nostro caso da Utente (attore astratto interpretato da Cliente o Azienda) e dal Database.
nota: si faccia riferimento a "MyWallet\uml\images\Usa-Case-Diagram.png".

4. Design Pattern

Per come è stata progettata l'architettura del software, si è deciso di adottare principalmente l'utilizzo del design pattern Singleton per la creazione delle classi riguardanti la gestione del database (classe DatabaseManager), e quella relativa al sistema di autenticazione (classe AuthManager).



Questa scelta di design comporta quindi che per le classi precedentemente citate venga prevista la creazione di un'unica istanza.

Per quanto riguarda il database manager, la sua istanza si occupa di gestire l'accesso al database per operazioni di lettura e scrittura inerenti a tutti gli account connessi in una determinata sessione. Più nel dettaglio offre una serie di funzioni con le quali il componente principale è in grado di inviare le richieste da parte degli utenti. Queste

richieste le gestisce con delle query che interrogano il database locale, ritornando poi l'esito dell'operazione insieme ai dati precedentemente richiesti.

Per quanto riguarda l'auth manager, la sua istanza viene inizializzata nel componente principale che si occupa di gestire l'interfaccia grafica al momento della sua creazione. Questa istanza gestisce la verifica delle credenziali dell'utente nei vari tentativi di login. La scelta di adottare questo design pattern per questa classe è dovuta al fatto che per la verifica delle credenziali è necessario l'utilizzo della crittografia, che è appunto gestita dall'istanza della auth manager. Il metodo fornito da questa classe ritorna infine l'esito della verifica, indicando la tipologia dell'utente che ha effettuato l'accesso.

4.1. Dependency Analysis

Col fine di controllare la fase di sviluppo dell'applicazione e assicurare che vengano rispettate le scelte di design prese nelle fasi precedenti, ci siamo avvalsi di un tool per l'analisi delle dipendenze (Stan4J). Questo, attraverso la generazione di grafici permette di visualizzare in maniera più chiara le dipendenze tra i vari moduli/componenti che costituiscono l'applicazione.

La scelta di effettuare questo tipo di analisi è dovuta anche al fatto che apporta benefici e migliorie per quanto riguarda l'attività refactoring. Infatti, con l'aumentare della dimensione del progetto diventa di estrema importanza riuscire a definire con chiarezza come interagiscono i vari moduli e quindi i componenti al loro interno, per essere poi in grado di effettuare.

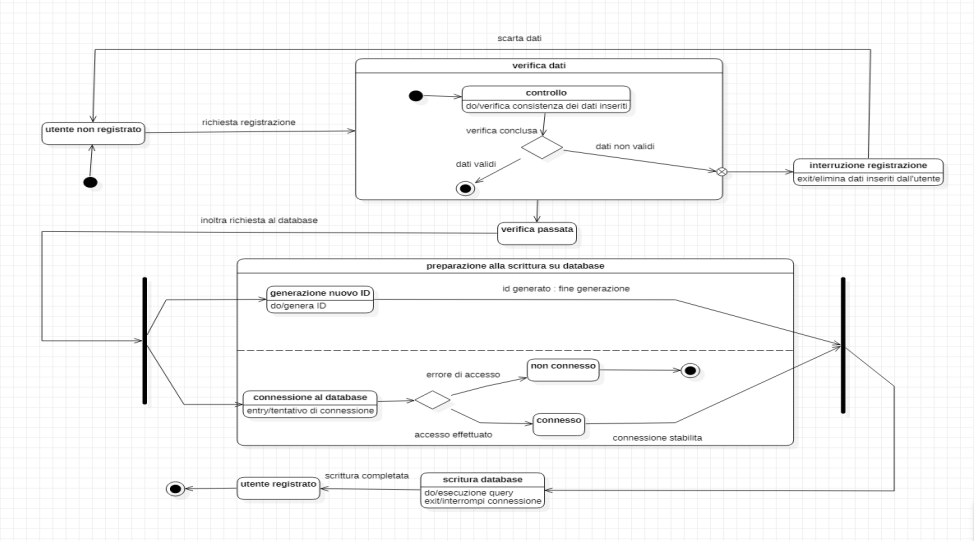
Inoltre, l'utilizzo di tool per l'analisi delle dipendenze facilita le situazioni nelle quali si ha la necessità di apportare determinate modifiche ad un componente, riuscendo ad analizzare in precedenza l'impatto che queste modifiche possono comportare all'intero sistema.

4.2. Diagrammi UML

Le caratteristiche del design dell'applicazione sono inoltre rappresentate da dei diagrammi UML.

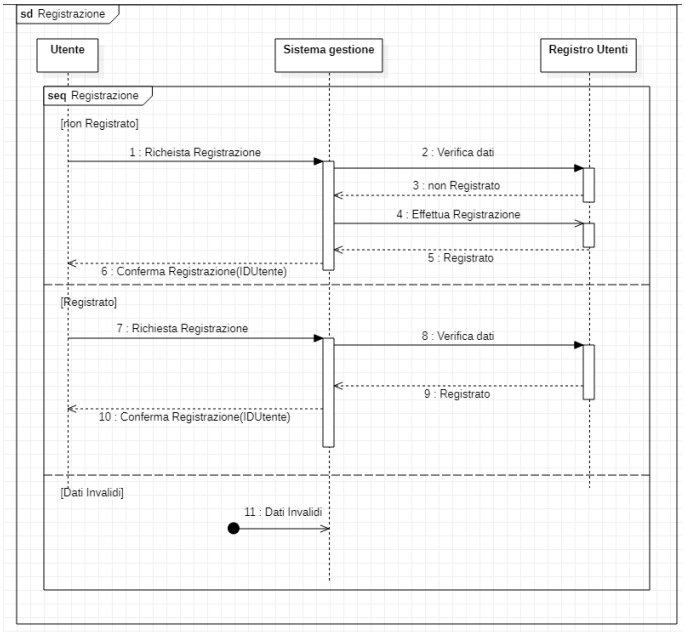
4.2.1. State chart diagram

Con questo diagramma si rappresenta il processo di sign-in, dalla richiesta di registrazione dell'utente, fino alla creazione della sua anagrafica nel database. Un aspetto importante che vuole evidenziare il diagramma è che al fine di avere un esito positivo dal processo di registrazione di un nuovo utente è necessario che la connessione con il database avvenga con successo. In caso contrario l'intero procedimento viene interrotto, come descritto con il final state nell'ortogonal state.



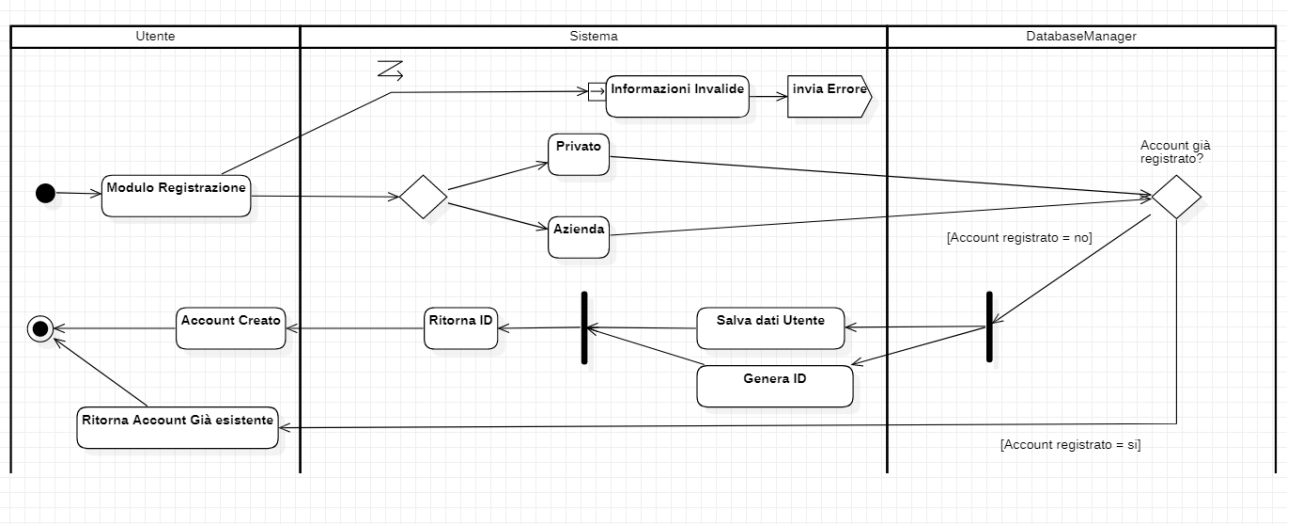
4.2.2. Sequence diagram

Con questo diagramma si sono volute modellare le due attività principali gestite dall'applicazione. In particolare, sono descritti gli scambi di messaggi tra i vari componenti inerentemente al processo di registrazione di un nuovo utente e all'effettuazione di una transazione. Entrambi i diagrammi mostrano dettagliatamente come interagiscono i vari componenti e come sono gestiti i processi.



4.2.3. Activity diagram

Anche con questo diagramma si modella il processo di registrazione dell'utente. Con questo diagramma si vuole introdurre un maggior livello di dettaglio per quanto riguarda il flusso delle attività tra i componenti. Infatti, sono messi in evidenza i vari componenti e le rispettive operazioni/attività di loro competenza.



5. Testing

Come specificato nel project plan, la modalità di sviluppo è orientata verso una metodologia agile, quindi verrà adottato un approccio TDD (Test-Driven-Development).
Junit4 è il framework impiegato per l'esecuzione dei test tramite IDE eclipse.

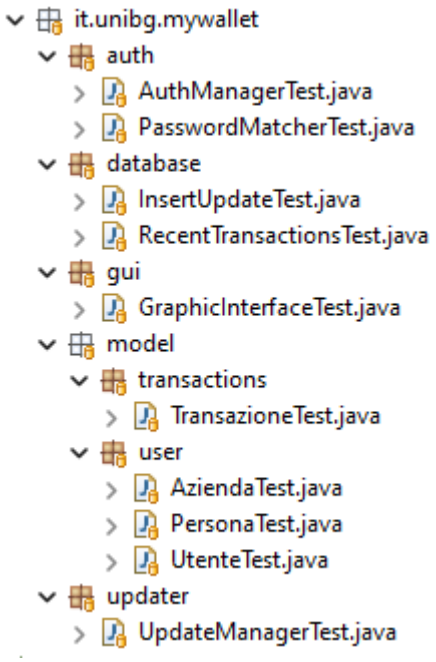
Il processo di sviluppo segue dei passi ben definiti:

- Scrittura caso di test
- Esecuzione e verifica corretto funzionamento
- In caso di fallimento, Il test viene riscritto e ri-eseguito assieme a tutti gli altri case.

Tutti i componenti del gruppo avente pari responsabilità e diritti possono procedere alla scrittura dei casi di test.

5.1. Organizzazione Tests

I casi di test sono organizzati e suddivisi per package e classi:



Nei vari casi di test vengono testati prevalentemente i metodi getter e costruttori per quanto riguarda le classi presenti nel package models.
Per la scrittura dei casi di test riguardanti il Database Manager abbiamo utilizzato un database di test in modo da poter testare il corretto funzionamento delle query sia per estrarre che inserire i dati.
I dati inseriti nel Database di test vengono poi ripristinati in modo da avere una base dati consistente per tutti i test.
Parte importante dei test riguarda il meccanismo di autenticazione che abbiamo testato con il 100% di code coverage essendo una parte critica per un sistema di questo tipo, ogni possibile situazione riguardo inserimento di dati sbagliati/corretti/mancanti è stata testata.
Buona parte dell'interfaccia grafica viene testata nel package gui, per quanto sia possibile testarla.
Nel package update abbiamo un caso di test riguardante l'Update Manager, è stato scritto in seguito ad aver rilevato alcune difficoltà nell'interrompere il thread che si occupa di aggiornare l'interfaccia grafica. In definitiva il progetto contiene 10 classi di test con una copertura totale del codice del 72,1%.

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
MyWallet	72,1 %	3.202	1.239	4.441
src/main/java	68,6 %	2.676	1.226	3.902
it.unibg.mywallet	76,1 %	1.839	578	2.417
it.unibg.mywallet.database	48,5 %	516	549	1.065
it.unibg.mywallet.updater	39,5 %	60	92	152
it.unibg.mywallet.model.user	93,8 %	61	4	65
it.unibg.mywallet.model.user.impl	95,9 %	70	3	73
it.unibg.mywallet.auth	100,0 %	77	0	77
it.unibg.mywallet.model.transactions	100,0 %	33	0	33
it.unibg.mywallet.model.transactions.impl	100,0 %	20	0	20
src/test/java	97,6 %	526	13	539

Tutti i casi di test vengono eseguiti attraverso maven quando il software viene buildato, se uno di questi dovesse fallire il processo viene interrotto.

5.2. Refactoring

Durante lo sviluppo del software sono stati fatti vari refactoring principalmente riguardo convenzioni (naming variabili, finalizzazione campi assegnati una sola volta...), un tool che abbiamo utilizzato per rilevare eventuali code smells è PMD utilizzando il classico ruleset per il linguaggio Java, escludendo alcune regole che abbiamo ritenuto superflue.

```
<?xml version="1.0"?>

<ruleset name="MyWallet rules"
    xmlns="http://pmd.sourceforge.net/ruleset/2.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://pmd.sourceforge.net/ruleset/2.0.0 https://pmd.sourceforge.io/ruleset_2_0_0.xsd">

    <description>
        MyWallet rules
    </description>


    <!-- Your rules will come here -->
    <rule ref="category/java/codestyle.xml">
        <exclude name="ShortVariable"/>
        <exclude name="LocalVariableCouldBeFinal"/>
        <exclude name="MethodArgumentCouldBeFinal"/>
    </rule>

</ruleset>
```

6. Conclusioni

Durante lo sviluppo del progetto abbiamo dedicato troppo tempo all'implementazione del database e all'interfaccia grafica trascurando inizialmente la documentazione.
L'interfaccia grafica ha richiesto più tempo di quanto ne avessimo stimato.
I casi di test hanno occupato una buona parte del tempo ma non abbiamo incontrato particolari difficoltà durante la loro scrittura.
Github è stato una parte fondamentale per la gestione del progetto che ci ha permesso di lavorare in parallelo su diverse funzionalità.