Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica

Metodologie agili applicate allo sviluppo di una componente d'interfaccia grafica web in Kotlin per l'analisi di Big Data

Tesi di laurea

Relatore
Prof. Claudio Enrico Palazzi

Laureando
Marco Rampazzo

Anno Accademico 2019-2020

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di trecentoventi ore, dal laureando Marco Rampazzo presso l'azienda GRUPPO4 S.r.l. L'obiettivo principale da raggiungere era quello di realizzare una componente d'interfaccia grafica il cui scopo è quello di permettere ad un utente di esplorare dati mediante una tabella pivot. Questo componente verrà utilizzato dall'azienda ospitante per sostituire un loro software correntemente in uso da oltre dieci anni.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Questa tesi descrive l'esperienza e il percorso lavorativo svolto presso l'azienda GRUPPO4 sotto la supervisione di Tobia Conforto.

1.1 L'azienda

GRUPPO4 è una web agency Padovana da oltre vent'anni. In questo periodo GRUPPO4 ha accumulato competenze e l'esperienza necessaria per fornire soluzioni efficaci e innovative nel settore web. Mediante un modello organizzativo consolidato e certificato sviluppano WebApp che si distinguono per la chiarezza dell'interfaccia utente (UX/UI) e per la loro usabilità.

1.2 L'idea

GRUPPO4 da oltre dieci anni fornisce una struttura web, creavista, per permettere ai loro clienti di esplorare una grande mole di dati velocemente. Per fare questo GRUPPO4 utilizza una tabella pivot che permette di collegare, mediante relazioni, diverse informazioni. Le tabelle pivot hanno proprio il vantaggio di unire tutte le informazioni all'interno di un database e fornire un'interfaccia facile da utilizzare per esplorare tali dati.
Questo tirocinio aveva come scopo quello di sostituire la tabella pivot di creavista, ormai datata, con una nuova componente realizzata con tecnologie innovative e sicure quali.

1.3 Organizzazione del testo

Il secondo capitolo descrive ...

Il terzo capitolo approfondisce ...

Il quarto capitolo approfondisce ...

Il quinto capitolo approfondisce ...

Il sesto capitolo approfondisce ...

Nel ottavo capitolo descrive ...

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*[g];
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Processi e metodologie

In questo capitolo verrà fornito una descrizione dei metodi e dei processi messi in atto durante il tirocinio, in particolare riguardo i seguenti argomenti: metodologia agile, programmazione funzionale e il concetto di Dataflow dell'applicazione.

2.1 Metodologia Agile

Per lo sviluppo del prodotto è stato deciso di utilizzare una metodologia agile in modo da reagire velocemente a problemi e a cambiamenti così da migliorare ed l'efficienza nella realizzazione della componente. L'azienda ha deciso di utilizzare una metodologia agile simile a SCRUM. Infatti applicare nella sua interezza il metodo SCRUM sarebbe stato impossibile dato il ristretto numero di sviluppatori nel team di sviluppo.
Le caratteristiche principali della metodologia agile applicata per la realizzazione di questo progetto sono le seguenti:

- **Modello incrementale**: vengono realizzati rilasci multipli e successivi che aiutano a definire più chiaramente i requisiti più importanti dato che essi verranno implementati per primi. Ogni rilascio corrisponde ad una parte funzionante di applicazione;
- **Modello iterativo**: un modello iterativo ha la caratteristica di avere una maggior capacità di adattamento in seguito a problemi di implementazione e cambiamenti nei requisiti da parte del cliente;
- **Organizzazione in sprint di sviluppo**: il processo[controlla se è il termine giusto] di codifica viene suddiviso in sprint di sviluppo, data la breve durata del tirocinio curriculare essi avranno una durata di circa 4-5 giorni;
- **Backlog**:
  - **Product Backlog**: rappresenta i requisiti e le funzionalità del prodotto definiti mediante le User Stories;
  - **Sprint Backlog**: rappresenta l'insieme delle User Stories da realizzare nello sprint indicato;
- **User Stories**: l'idea alla base di uno sviluppo agile è la realizzazione delle User Stories. Ogni user story è definita da una **descrizione del problema** e da una **priorità**.

- **Riunioni**:
  - **Sprint planning**: per pianificare il lavoro da svolgere durante lo sprint;
  - **Sprint review**: riunione retrospettiva per verificare il lavoro svolto durante lo sprint;
  - **Backlog refinement**: per aggiungere nuove User Stories o migliorare e/o modificare User Stories già create;
  - **Riunioni giornaliere**: per verificare lo svolgimento del lavoro, in questo tirocinio sono state sostituite con comunicazioni telematiche giornaliere.

2.1.1 Definizione delle User Stories

Per prima cosa il team di sviluppo si è occupato di realizzare le User Stories. Per definire un singolo elemento è stata utilizzata la seguente struttura:

Id	**Descrizione**	**Priorità**	**Implementato**
US1.1	Descrizione dell'user story	A	SI

Tabella 2.1: Esempio tabella User Story

Per ogni descrizione di un user story si possono identificare le seguenti informazioni:

- **Ruolo**: definisce il tipo di utente;
- **Obiettivo**: definisce di che cosa ha bisogno l'utente;
- **Beneficio**: definisce i vantaggi che porta all'utente.

La sinteticità e la facilità nel definire le user story porta a vantaggi nella comunicazione tra il team di sviluppo e il cliente, rende più semplice l'aggiornamento dei requisiti e i costi di scrittura e manutenzione delle user stories sono molto bassi.

2.1.2 Definizione del Product Backlog

Uno dei primi obiettivi del progetto posti dal team di sviluppo è stato quello di definire il Product Backlog, cioè i requisiti del prodotto. Per ognuna delle user story precedentemente scritta è stata assegnata una priorità seguendo la seguente legenda:

A	*Priorità alta*	funzionalità necessarie per il corretto funzionamento dell'applicazione
M	*Priorità media*	funzionalità che migliorano il prodotto
B	*Priorità bassa*	funzionalità non necessarie per il corretto funzionamento dell'applicazione

Tabella 2.2: Tabella priorità User Story

Questo ci ha permesso di categorizzare le funzionalità principali del componente d'interfaccia grafico da quelle opzionali. Abbiamo quindi popolato il Product Backlog

per ordine di *priorità*, in questo modo la suddivisione delle user story per sprint, mediante le riunioni di *Sprint Planning*, è stata chiara e veloce.
Infatti abbiamo concentrato le funzionalità principali da implementare nei primi due sprint così da avere già a partire dal terzo sprint una prodotto con le funzionalità principali già implementate.

2.2 Programmazione funzionale

La programmazione funzionale è un paradigma di programmazione dichiarativa dove un programma è costituito dall'applicazione e dalla composizione di funzioni. In questo progetto si è utilizzata, dove possibile, la programmazione funzionale in particolare mediante le *Funzioni di ordine superiore* fornite da Kotlin. Queste funzioni sono molto utili perchè permettono di scrivere codice più leggibile, conciso e soprattutto hanno la caratteristica di evitare *side-effects*. Come definito dalla documentazione di Kotlin (bib: https://kotlinlang.org/docs/reference/lambdas.html), le funzioni scritte nel linguaggio Kotlin sono considerate come *first-class* quindi esse possono essere contenute in variabili e strutture dati, possono essere passate come argomento di altre funzioni e possono essere ritornate da altre *Funzioni di ordine superiore*. Queste funzioni sono state usate estensivamente nella realizzazione del "parser" descritto nel capitolo 4:descrizione-stage.

2.3 Dataflow dell'applicazione

Un concetto che è stato discusso molto dal team di sviluppo riguardava il *dataflow* dell'applicazione. Durante la prima settimana del tirocinio è stato discusso come verrà gestito lo stato dell'applicazione e in particolare delle soluzioni per garantire la scalabilità del componente. Per gestire lo stato dell'applicazione sono state individuate due possibili soluzioni: React e Redux.

2.3.1 React

React è una libreria per realizzare interfacce utente che utilizza un dataflow unidirezionale. Questo perchè ogni componente può avere uno stato locale accessibile solo dal componente stesso e può passare informazioni ai suoi componenti figli mediante le *props*.
Questo dataflow è molto semplice però presenta alcune limitazioni per quanto riguarda la scalabilità. Per avere uno stato unico di tutta l'applicazione bisognerebbe dare la responsabilità ad un componente grafico di gestirlo e passarlo mediante le sue *props*. L'architettura che deriva da questo dataflow, nel caso di applicazioni complesse con molti componenti, è limitata, difficile da manutenere e poco scalabile.
Tuttavia questo semplice dataflow ha il vantaggio che per famiglie di componenti piccole i cambiamenti di stato locale e i passaggi di informazioni mediante le *props* sono molto veloci e semplici.

2.3.2 Redux

Per garantire garantire scalabilità e facilità di gestione dello stato dell'applicazione abbiamo discusso riguardo l'utilizzo di Redux. Come React essa offre un dataflow unidirezionale tuttavia lo stato non è più contenuto all'interno di una gerarchia di componenti grafici, bensì esso viene contenuto in una struttura chiamata *store*. Questa

struttura esterna ai componenti grafici rende la gestione dello stato dell'applicazione più prevedibile e più manutenibile. Redux si basa su tre principi:

- lo stato è l'unica fonte di verità;
- lo stato non è modificabile direttamente;
- le modifiche avvengono medianti funzioni pure (glossario) che creano un nuovo stato per evitare side effects.

Gli elementi dell'architettura di Redux sono i seguenti:

- **Actions**: oggetti che rappresentano un azione che innesca un update dello stato;
- **Reducers**: *funzioni pure* che hanno come parametro un Actions e si occupano di modificare lo stato;
- **Store**: lo Store è un'interfaccia che contiene lo stato dell'applicazione e fornisce funzioni per leggere, modificare e registrare listeners allo stato.

2.3.3 Soluzione

La soluzione adottata è stata quella di usare sia React che Redux. React è stato usato per gestire solo gli aggiornamenti visivi dell'applicazione, mentre Redux per gestire lo stato dell'applicazione. Il dataflow del prodotto è quindi il seguente:

1. se lo stato deve cambiare verrà mandata allo Store una richiesta di cambiamento con un Action;
2. lo Store si occuperà di chiamare un Reducer in modo da modificare lo stato;
3. lo stato verrà aggiornato e i cambiamenti saranno visibili a tutti i componenti React che sono registrati allo Store.

Capitolo 3

Progettazione

3.1 Introduzione

In questo capitolo verrà discusso il modo in cui sono state progettate le singole parti dell'applicazione e il ragionamento alla base di tale progettazione. In particolare verranno descritte le seguenti strutture:

- stato dell'applicazione;
- componenti grafici;
- parser json;
- adapter dello stato.

3.2 Stato dell'applicazione

3.2.1 Struttura dati

Per rappresentare la struttura dati dello stato della tabella pivot ho utilizzato le `data class` di kotlin. Degli oggetti che hanno come unico scopo quello di contenere informazioni. Per prima cosa ho realizzato un nuovo package di nome `Entities` all'interno dell'applicazione, al suo interno ho definito le `data class`, che chiamerò con il nome di entità.
L'idea alla base della struttura dello stato dell'applicazione era quella di avere una struttura dati semplice da iterare in modo da avere funzioni di renderizzazione concise e facili da manutenere. Le entità che ho realizzato sono le seguenti:

- `TableState`;
- `DimensionsNode`;
- `NodeActionType`;
- `BodyCells`;
- `HeaderAction`.

3.2.1.1 TableState

L'entità `TableState` è la `data class` che contiene l'intero stato dell'applicazione. Al suo interno sono presenti tutte le informazioni necessarie per la corretta renderizzazione della tabella pivot.

Listing 3.1: TableState

```
data class TableState(
  val cols: ArrayList<ArrayList<DimensionsNode>>,
  val rows: ArrayList<ArrayList<DimensionsNode>>,
  val cells: ArrayList<ArrayList<BodyCells>>,
  val rowAction: ArrayList<ArrayList<HeaderAction>>,
  val colAction: ArrayList<ArrayList<HeaderAction>>
)
```

Descrizione campi dati
I primi due campi dati da descrivere sono `cols` e `rows`. Sono due matrici di `NodeDimensions` che rappresentano rispettivamente le celle d'intestazione della tabella pivot delle colonne e delle righe.
Il campo dato `cells` è una matrice di `BodyCells` che rappresenta le celle che contengono gli effettivi dati della tabella pivot.
Infine si hanno `rowAction` e `colAction`. Sono due matrici di `HeaderAction` e rappresentano i pulsanti all'interno del componente TableActionUI.

3.2.1.2 DimensionsNode

Listing 3.2: DimensionsNode

```
data class DimensionsNode(
  var id: String,
  var label: String,
  var level: Int? = null,
  var childDepth: Int? = null,
  var path: List<String>? = null,
  var actionType: NodeActionType = NodeActionType.NULL,
  var isChild: Boolean = false
)
```

Descrizione campi dati

3.2.1.3 HeaderAction

Listing 3.3: HeaderAction

```
data class HeaderAction(
  val actionType: NodeActionType,
  val dim: Int,
  val depth: Int,
)
```

Descrizione campi dati

3.2.1.4 NodeActionType

Listing 3.4: NodeActionType

```
enum class NodeActionType(val type: String) {
  EXPAND("E"),
  COLLAPSE("C"),
  NULL("")
}
```

Descrizione campi dati

3.2.1.5 BodyCells

Listing 3.5: BodyCells

```
data class BodyCells(
  val value: Int,
  val cPath: List<String>,
  val rPath: List<String>
)
```

Descrizione campi dati

3.2.2 Stato Redux

Lo stato gestito da Redux è equivalente alla struttura di `TableState` con l'aggiunta del campo dato `isLoading` così da avere un variabile per gestire la tabella durante il caricamento dall'API esterna di dati. Per rendere modulare lo stato è stata realizzata una "Slice". Uno "Slice" è definito come una parte di stato dell'applicazione che può essere unito con altri slice per realizzare lo stato completo dell'applicazione. Questa architettura è molto utile per garantire scalabilità allo stato. In questo componente lo stato dell'applicazione è stato definito in uno "Slice" dello stato chiamato `TableStateSlice` che è costituito da quattro elementi:

- *state*: `data class` che contiene tutte le informazioni necessarie per la renderizzazione della tabella pivot;
- *actions*: oggetti di tipo `class : RAction` che vengono usati per innescare l'update dello stato;
- *thunk*: oggetti di tipo `object` che hanno come scopo quello di realizzare operazioni complesse asincrone, è stato usato un thunk per gestire il caricamento asincrono dei dati dall'API;
- *reducers*: funzione pura che riceve come argomento un `RAction` e ritorna una copia dello stato modificato.

3.3 Componenti grafici

3.3.1 View

3.3.2 Container

3.4 Parser JSON

3.4.1 Struttura dati @Serializable

3.5 Adapter

3.5.1 Utilizzo del paradigma della programmazione funzionale

3.5.1.1 Funzioni utilizzate

3.6 Suddivisione dei componenti grafici

3.6.1 View

3.6.1.1 TableView

Questo componente è stato realizzato per definire la struttura generale del componente e per suddividere in modo responsivo lo spazio in quattro quadranti. Come si può vedere dalla seguente immagine ho realizzato il componente utilizzando il layout CSS Grid. Questo mi ha permesso di gestire, in modo responsivo, la struttura del componente.

3.6.1.2 TableHeaderView

Questo componente rappresenta le dimensioni superiori della tabella pivot. Dire che la struttura della tabella è stata realizzata con flex, table, etc.

3.6.1.3 TableSidebarView

Questo componente rappresenta le dimensioni laterali della tabella pivot. Dire che la struttura della tabella è stata realizzata con flex, table, etc.

3.6.1.4 TableBodyView

Questo componente rappresenta i dati della tabella pivot Dire che la struttura della tabella è stata realizzata con flex, table, etc.

3.6.1.5 TableLabel

Questo componente rappresenta i dati della tabella pivot Dire che la struttura della tabella è stata realizzata con flex, table, etc.

3.6.1.6 TableActionUI

Questo componente rappresenta i dati della tabella pivot Dire che la struttura della tabella è stata realizzata con flex, table, etc.

3.6.2 Container Redux

3.6.2.1 TableController

3.6.2.2 TableHeaderController

3.6.2.3 TableSidebarController

3.6.2.4 TableBodyController

3.6.2.5 TableActionUIController

3.6.2.6 TableLabelController

3.7 Struttura dati dell'applicazione

3.7.1 Descrizione

Per definire lo stato dell'applicazione sono state create delle 'data class' per gestire in modo ordinato lo stato del prodotto.

3.7.1.1 TableState

Lo stato del prodotto è definito da un singolo oggetto: 'data class TableState'.

3.7.1.2 Cells

3.7.1.3 CellAction

3.7.1.4 BodyCells

3.8 Realizzazione del JSON parser

Lo stato dell'applicazione viene popolato in seguito ad una richiesta ad API. Queste API ritornano risultati in formato JSON. Lavorando con Kotlin non è possibile avere la stessa libertà di accesso ad un JSON come in Javascript, quindi sono stati creati delle classi per eseguire la lettura del JSON e la sua traduzione in oggetti 'data class'. GRUPPO4 ha fornito una API che deve essere compatibili con questo componente quindi per ho realizzato un parser per trasformare un json in oggetti 'data class' e un adapter per trasformare gli oggetti data class nella data structure utilizzata nello stato dell'applicazione.

3.8.1 Adapter: creavista

Descrivi l'API
Descrivi il parser
La realizzazione del parse e del successivo adapter è stato uno delle difficoltà più grandi che ho superato. La struttura del JSON faceva cagare. Per realizzare un datastructure decente ho preso i dati dal json e li ho trasformati in una rappresentazione ad albero. Nel seguente modo: ESEMPIO FOTO OR WHATEVER. Dopodichè ho trasformato la rappresentazione ad albero degli oggetti in List che costituite nel seguente modo: ESEMPIO. IN particolare per quanto riguarda la dimensione delle righe ho suddiviso questa list in tante liste quante sono le dimensioni delle righe. Mentre per le coonne ho suddiviso la List in tante liste quante sono le dimensioni delle colonne.

Descrivi le funzioni

Capitolo 4

Descrizione dello stage

4.1 Studio delle tecnologie

4.2 Sprint di sviluppo

4.2.1 Primo sprint di sviluppo: Realizzazione dei componenti grafici

Descrizione

Sprint Backlog

Soluzioni implementate

4.2.2 Secondo sprint di sviluppo: Realizzazione della struttura dei dati e del parser JSON

Descrizione

Sprint Backlog

Soluzioni implementate

4.2.3 Terzo sprint di sviluppo:

Descrizione

Sprint Backlog

Soluzioni implementate

4.2.4 Quarto sprint di sviluppo:

Descrizione

Sprint Backlog

Soluzioni implementate

4.3 Codifica dei test

4.3.1 Unit test

Capitolo 5

Analisi dei requisiti

5.1 Product Backlog

Id	Descrizione	Priorità	Implementato
US1.1	Come Cliente voglio poter visualizzare i miei dati e le dimensioni relative ai dati	A	SI
US1.2	Come Cliente voglio poter utilizzare questa applicazione web dal mio PC, dal mio telefono e dal mio Tablet	A	SI
US1.3	Come Cliente voglio poter visualizzare solo i dati senza le dimensioni relative ad essi	M	SI
US2.1	Come Cliente voglio poter utilizzare l'API precedente di Creavista	A	SI
US2.2	Come Cliente voglio poter utilizzare la nuova API di Creavista	A	SI
US2.3	Come Cliente voglio avere un caricamento veloce	M	SI
US3.1	Come cliente voglio poter salvare la mia configurazione	M	SI
US3.2	Come cliente voglio poter esportare i miei dati	B	SI
US3.3	Come cliente voglio poter decidere quali filtri voglio utilizzare per ogni dimensione	A	SI
US3.4	Come cliente voglio poter modificare i filtri che sto utilizzando	A	SI

5.2 Requisiti individuati dal Product Backlog

Id	**Descrizione**	**Tipo**	**Impl.**	**User Story**
R1.0	Definizione e pianificazione dei componenti	O	SI	US1.1
R1.1	Sviluppo dei componenti React visivi	O	SI	US1.1
R1.1.1	Sviluppo di TableView	O	SI	-
R1.1.2	Sviluppo di TableHeaderView	O	SI	-
R1.1.3	Sviluppo di TableSidebarView	O	SI	-
R1.1.4	Sviluppo di TableBodyView	O	SI	-
R1.2	Sviluppo dei container Redux	O	SI	US1.1
R1.1.1	Sviluppo di TableController	O	SI	-
R1.1.2	Sviluppo di TableHeaderController	O	SI	-
R1.1.3	Sviluppo di TableSidebarController	O	SI	-
R1.1.4	Sviluppo di TableBodyController	O	SI	-
R2.0	Sviluppo dei JSON parser	O	NO	US2.1
R2.1	Sviluppo parser per Creavista	O	SI	-
R2.1.1	Sviluppo data structure in Kotlin	O	SI	-
R2.1.2	Sviluppo adapter	O	SI	-
R2.2	Sviluppo parser per API nuova	O	NO	US2.2
R2.2.1	Sviluppo data structure in Kotlin	O	NO	-
R2.2.2	Sviluppo adapter	O	NO	-
R3.0	Sviluppo di un sistema di caricamento automatico	O	NO	US2.3
R3.1	Sviluppo di InfiniteScroller per gestire il caricamento automatico	O	SI	-

Capitolo 6

Tecnologie e strumenti

6.1 Tecnologie

INSERIRE L'IDEA DI COME LE TECNOLOGIE SONO STATE UTILIZZATE NEL PROGETTO

Kotlin

Kotlin è un linguaggio tipizzato, realizzato da JetBrains, utilizzato da molti sviluppatori per il fatto che il codice è conciso, sicuro e permette di lavorare utilizzando libreria per la JVM, Android e il browser.

React

React è una libreria che presenta principalmente due caratteristiche:

- l'uso del Virtual DOM;
- realizzazione di componenti migliorare la reutilizzazione del codice.

Kotlin Wrappers

kotlin-react

kotlin-redux

kotlin-react-redux

6.2 Strumenti

Di seguito viene data una descrizione delle tecnologie che sono stati utilizzati durante il tirocinio.

6.2.1 Versionamento della soluzione

Git

Git è un VCS (Version Control System) distribuito che permette di tenere traccia delle modifiche in un prodotto software e di organizzare la codifica del prodotto.

GitLab

Strumento web che permette di implementare un DevOps lifecycle che fornisce una gestione di repository git, un ITS (Issue Tracking System) e altri strumenti quali la "Continuous integration" e "Continuous deployement".

6.2.2 Ambiente di sviluppo locale

IntelliJ IDEA Community Edition

IntelliJ IDEA Community Edition è una IDE realizzata da JetBrains che fornisce funzionalità di supporto per lo sviluppo di molti linguaggi, specialmente Kotlin.

Gradle

Gradle è uno strumento di "Build automation" per molti linguaggi tra cui Kotlin e Java. E' stato usato per la gestione e l'installazione delle dipendenze.

6.2.3 Organizzazione del lavoro

Trello

Per l'organizzazione del lavoro, in particolare per la gestione dei macro obiettivi di ogni sprint, ho utilizzato Trello che fornisce un'interfaccia Kanban.

Capitolo 7

Conclusioni

7.1 Problemi riscontrati

7.2 Raggiungimento degli obiettivi

7.3 Conoscenze acquisite

7.3.1 Metodologia agile

7.3.2 Kotlin

7.3.3 React e Redux

7.3.4 Programmazione funzionale

7.3.5 Lavorare in un team di sviluppo

7.4 Valutazione personale

7.4.1 Effettività delle metodologie agili

7.4.2 Effettività di Kotlin per la realizzazione di UI per il web

7.4.3 Effettività di React e Redux

7.4.4 Effettività della programmazione funzionale

Appendice A

Appendice A

Citazione

Autore della citazione

Bibliografia

"Life is really simple, but we insist on making it complicated"

— Confucius

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Claudio Enrico Palazzi, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

Desidero ringraziare con la mia famiglia per essermi stati vicini in ogni momento durante gli anni di studio.

Ho desiderio di ringraziare poi i miei amici per tutti i bellissimi anni passati insieme e le mille avventure vissute.

Padova, Dicembre 2020 Marco Rampazzo