Università Politecnica delle Marche

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica e dell'Automazione

Tesi di Laurea

Progettazione e implementazione di un sistema per la gestione di un'azienda tessile: la componente relativa alle risorse interne

Design and implementation of a system for the management of a textile company: the internal resource component

Relatore

Prof. Domenico Ursino

Candidato

Simone Scalella

Anno Accademico 2020-2021

Indice

Elenco delle figure

Elenco dei listati

Introduzione

La storia ci insegna che, nel corso del tempo, le aziende, se vogliono essere competitive sul mercato e vogliono aumentare i loro guadagni, devono rimanere aggiornate con le moderne tecnologie. Si cominciò nel passato, durante la prima rivoluzione industriale, nel 1760, con l'introduzione della macchina a vapore, fino ad arrivare all'Industria 4.0 dei giorni nostri. La parola chiave è "digitalizzazione"; sostanzialmente tutti i processi produttivi e informativi devono essere affiancati, o completamente sostituiti, da sistemi informatici.

I computer e l'informatica sono degli strumenti molto potenti che, nel giro di poche decine di anni, hanno rivoluzionato il lavoro nelle industrie e nelle aziende; non esiste processo che non sia stato reso più efficiente grazie all'utilizzo di un computer. Attualmente un'azienda che punta ad aumentare i propri guadagni, migliorando il proprio modo di lavorare, non può più utilizzare, ad esempio, i fogli di carta per raccogliere informazioni; questi strumenti sono obsoleti e non si possono paragonare con l'efficienza di un computer. I sistemi informatici sono affidabili; svolgono delle mansioni senza commettere errori; se gli utenti sbagliano durante l'esecuzione di un'operazione, li avvisano; sono efficienti, infatti le operazioni, ad esempio, di ricerca delle informazioni vengono svolte in tempi brevissimi.

Un altro strumento molto importante è Internet; negli ultimi anni si è osservato come molte aziende hanno spostato i loro investimenti su progetti che la coinvolgono, in quanto offre una vastissima scelta di servizi.

La realtà aziendale che viene studiata in questa tesi è, come vedremo successivamente, un'azienda manifatturiera che lavora nel settore tessile, e che vuole investire nella digitalizzazione dei processi, in quanto molte operazioni, dalle più semplici alle più importanti, vengono ancora svolte con fogli di carta o, al massimo, documenti Excel. Ovviamente questo modo di lavorare non solo è lento, ma può anche portare a situazioni difficili da gestire, dovute, ad esempio, allo smarrimento o danneggiamento di uno di questi documenti.

Il sistema che implementeremo permetterà all'azienda di risolvere le sue problematiche e di migliorare i processi produttivi. Infatti, avremo un database, che rappresenta una soluzione al problema dei documenti cartacei o dei file Excel; in esso, memorizzeremo tutte le informazioni di interesse dell'azienda, da quelle economiche (come, ad esempio, le entrate, le uscite, gli stipendi, etc.) a quelle riguardanti il personale, i clienti e la lavorazione. Avremo un server che si interfaccerà con il

database, e dei client che effettueranno le richieste al server. Implementeremo un'architettura client-server, in cui i client richiedono informazioni al server; quest'ultimo, una volta accettata la richiesta, interroga il database e restituisce al client un file JSON contenente tutte le informazioni richieste; sarà, poi, il client ad effettuare il parsing della risposta e a visualizzare le informazioni richieste dall'utente.

All'interno del server implementeremo tutte le rotte che serviranno ai client per effettuare le varie richieste; implementeremo, inoltre, un sistema di autenticazione per evitare che personale non autorizzato possa accedere alle funzionalità del sistema. Le funzionalità implementate saranno accessibili dai client, i quali, tramite le richieste, le faranno eseguire al server sui dati contenuti nel database.

Lato client sarà possibile effettuare l'inserimento di un elemento, compilando i campi visualizzati all'interno di una finestra; tale richiesta, una volta inoltrata al server, sarà validata per controllare che l'utente non abbia inserito informazioni errate.

Se la richiesta viene accettata, l'elemento viene inserito nel database. Sarà possibile effettuare la modifica di un elemento già esistente all'interno del database, con un procedimento analogo al precedente. Si potranno visualizzare le informazioni presenti nel database sia in forma generica, all'interno di liste, sia in forma dettagliata, all'interno di una finestra specifica; in questo secondo caso verranno visualizzate le informazioni dettagliate di un singolo elemento. Infine, vi sarà l'operazione di cancellazione, per eliminare gli elementi.

Il sistema descritto sopra è stato sviluppato nell'ambito di due tesi. In questa tesi viene presentata la componente relativa alle risorse interne, mentre nella tesi del candidato `Yihang Zhang` viene illustrata la componente finanziaria. La tesi è strutturata come di seguito specificato:

- Nel Capitolo 1 verrà riportata l'intervista con il responsabile dell'azienda, evidenziando le domande che gli sono state fatte e le relative risposte. Successivamente, viene presentata una descrizione del sistema da realizzare, dove viene spiegato quali sono i compiti che devono essere svolti dalle varie componenti del sistema. Infine, viene riportato un glossario dove vengono raccolti i termini tecnici e ne viene spiegato il significato.
- Nel Capitolo 2 viene riportata la descrizione generale del sistema complessivo. In particolare, viene proposta l'analisi dei requisiti, sia funzionali che non funzionali, vengono descritti i diagrammi dei casi d'uso e vengono approfonditi alcuni casi d'uso. Infine, riportiamo la progettazione del database, mostrando lo schema E/R e lo schema logico.
- Nel Capitolo 3 verrà riportata la progettazione della componente relativa alle risorse interne del sistema. Riporteremo i diagrammi delle classi, delle sequenze e delle attività; ogni diagramma sarà accompagnato da una descrizione, che ne spiega il significato. Infine, illustreremo i mockup del sistema.
- Nel Capitolo 4 verrà riportata l'implementazione della componente relativa alle risorse interne del sistema. Riportiamo dei listati che contengono pezzi di codice e classi, i quali implementano le varie funzionalità del sistema. Ad ogni listato è associata una descrizione, che ne spiega il funzionamento.
- Nel Capitolo 5 verrà riportato il manuale utente. Questo è composto da immagini, le quali contengono dei simboli composti da lettere e numeri; a tali simboli

sono associate delle istruzioni. Il manuale serve per facilitare l'uso del sistema da parte dell'utente.

- Nel Capitolo 6 verranno tratte delle conclusioni in merito al lavoro svolto e si analizzeranno alcuni possibili sviluppi futuri.

1

Contesto di riferimento

In questo capitolo descriviamo il bisogno che hanno le aziende nel 2021 di compiere una transizione verso la digitalizzazione dei propri processi produttivi. Le aziende manifatturiere, che devono aumentare il proprio guadagno e devono migliorare i propri processi produttivi, devono abbandonare i metodi tradizionali di gestione dell'informazione e devono digitalizzarli.

1.1 Realtà d'interesse

Il soggetto del lavoro è un'azienda manifatturiera che lavora nel campo tessile. Quest'azienda si occupa di prendere del materiale semilavorato da altre aziende nel quale poi effettua altre operazioni. L' azienda è un'azienda di medie dimensioni e conta un personale di circa 30 dipendenti. Tutti i semilavorati nell'azienda possiedono dei documenti grazie ai quali possono essere trasportati; le mansioni tra i vari dipendenti sono differenti e la gestione della contabilità è abbastanza complessa.

1.2 Intervista al responsabile dell'azienda

Viene riportata di seguito l'intervista avuta con il responsabile dell'azienda,necessaria conoscere la realtà di interesse e i requisiti di funzionamento del sistema.

Leggenda

`Sc:` Scalella Simone
`Zh:` Zhang Yihang
`Res:` Responsabile

Intervista

`Sc,Zh:` Buongiorno, vorrei iniziare chiedendole informazioni generali sulla sua azienda.

`Res:` L'azienda di cui sono il responsabile è un'azienda di medie dimensioni, che si chiama "Scang", è un'azienda che si trova in Abruzzo, all'interno della quale lavorano all'incirca 30 persone. Essa opera nel campo della manifattura tessile e svolge il compito di eseguire determinate operazioni su prodotti tessili semilavorati.

`Sc,Zh:` Quali sono i processi da ottimizzare?

`Res:` L'azienda vuole investire nella digitalizzazione. Come prima cosa serve un sistema per rendere più efficiente le procedure di memorizzazione dei dati all'interno dell'azienda, in quanto, attualmente, utilizziamo dei documenti cartacei e dei file excel, che rendono le operazioni di ricerca e verifica estremamente lente.

`SC,Zh:` Quali sono i dati che devono essere memorizzati?

`Res:` I dati da gestire sono molti, riguardano clienti, dipendenti, flussi monetari e le commesse che vengono assegnate all'azienda.

`Sc,Zh:` Quali sono i dati dei clienti che devono essere memorizzati?

`Res:` I dati dei clienti che devono essere gestiti sono tutte le informazioni di contatto, come numero di telefono, indirizzo e nome dell'azienda.

`Sc,Zh:` Quali sono, invece, i dati che devono essere memorizzati sulle commesse?

`Res:` I dati che devono essere memorizzati sulle commesse sono tutte le informazioni che riguardano le operazioni da eseguire; quindi il modello, il tessuto e altre informazioni generali. Inoltre, il sistema deve memorizzare il corrispondente listino prezzi e i relativi documenti DDT.

`Sc,Zh:` Che tipo di informazione rappresenta il listino prezzi?

`Res:` Il listino prezzi contiene informazioni riguardanti il compenso extra da assegnare ad ogni dipendente professionale, e il compenso che l'azienda riceve per ogni tipo di lavoro commissionato.

`Sc,Zh:` Invece, quali sono i documenti DDT che deve gestire?

`Res:` I documenti DDT sono i documenti di trasporto; sono dei documenti che vengono generati ogni volta che la commessa (cioè l'insieme dei semilavorati tessili) viene spostata da un'azienda ad un'altra. La prima volta che la commessa arriva in azienda è già fornita di un DDT; poi, ogni volta che essa viene spostata, ad esempio per andare in lavanderia, viene generato un DDT che, però, è non contabile, e quindi deve essere separato dagli altri. I DDT contabili vanno considerati nella fattura, mentre quelli non contabili no.

`Sc,Zh:` Quali sono i dipendenti che il sistema deve gestire?

`Res:` L'azienda presenta diversi tipi di dipendenti: il responsabile, un contabile, i dipendenti professionali, i dipendenti generici e un cuoco.

`Sc,Zh:` Quali sono le differenze tra queste figure che il sistema deve gestire?

`Res:` I primi sono le figure alle quali viene assegnato inizialmente il semilavorato, e sono le persone autorizzate a lavorare sui macchinari dove è richiesta più esperienza. Il lavoro ad ogni dipendente viene assegnato in maniera proporzionale alla sua esperienza e al tipo di lavoro da compiere. Lo stipendio di tali dipendenti è composto di una parte fissa e di una parte variabile, che dipende dal numero di pezzi prodotti. I dipendenti professionali possono scambiarsi tra di loro quantità di lavoro, a causa, ad esempio, di motivi personali, oppure perchè non riescono a rispettare la scadenza

di consegna, ad un prezzo che loro stessi decidono, ovviamente con l'autorizzazione del responsabile. Tale modifica deve essere registrata per il calcolo degli stipendi fatto dal contabile. Invece, i dipendenti generici, lavorano con macchinari molto semplici, svolgono lavori manuali, sono le figure che suddividono, una volta assegnato, il semilavorato per i dipendenti professionali, e il loro stipendio è composto solo da una parte fissa. Il responsabile deve gestire il lavoro nell'azienda; il contabile si occupa di generare le fatture da presentare ai clienti e calcola gli stipendi che devono essere assegnati ai vari dipendenti; il cuoco si occupa del servizio di catering. Tutti i dipendenti hanno accesso al magazzino per poter utilizzare gli strumenti necessari per il lavoro, come, ad esempio, il filo, le forbici, e altre cose di questo tipo. Le informazioni di interesse sui dipendenti sono tutti i dati anagrafici, come codice fiscale, nome, cognome, etc. Di ogni dipendente dobbiamo salvare i vari importi lavorativi, le ore di lavoro, lo straordinario e le ferie. Il sistema dovrà gestire tutte le informazioni sulla suddivisione delle commesse tra i vari dipendenti professionali.

`Sc,Zh:` Che tipo di informazioni deve gestire sulla suddivisione delle commesse?

`Res:` Su questo aspetto è neccessario gestire tutte le informazioni riguardanti le quantità di commessa assegnata per ogni dipendente, il modello e i valori economici; inoltre bisogna registrare ogni richiesta di scambio di lavoro tra un dipendente e l'altro, che, poi, dovrà essere approvata dal responsabile. Bisogna memorizzare anche la data relativa al termine del lavoro di un dipendente professionale, momento in cui la commessa passa nelle mani dei dipendenti generici per essere lavorata.

`Sc,Zh:` Come funziona il lavoro dei dipendenti professionali che il sistema deve gestire?

`Res:` All'interno dell'azienda ad ogni dipendente professionale viene assegnata una postazione "privata" che comprende un macchinario "lineare" usato molto frequentemente durante tutti i lavori sui semilavorati. All'interno della postazione ci sono, anche, un paio di forbici, filo, una pinzetta e due cacciavite e aghi; tali attrezzature vengono fornite dall'azienda periodicamente o quando necessario. All'interno dell'azienda troviamo all'incirca dieci macchine "pubbliche" che possono essere utilizzate da tutti i dipendenti professionali, ma soltanto uno alla volta. La coda dei dipendenti che vogliono usare il macchinario, viene gestita da loro stessi. Risulta importante gestire al meglio i turni di lavoro sulle macchine pubbliche; attualmente l'organizzazione dei dipendenti è verbale o, al massimo, cartacea, e questo genera malcontento e perdite di tempo nel caso di dimenticanze o smarrimento del foglio. Gestire in maniera digitale le code di accesso ai macchinari dovrebbe permettere di eliminare i tempi morti, dovuti magari al fatto che un dipendente cancella una prenotazione senza avvisare gli altri. Inoltre tale processo dovrebbe permettere ai dipendenti di controllare con più semplicità le prenotazioni degli altri.

`Sc,Zh:` Quali sono le informazioni delle macchine pubbliche che il sistema deve gestire?

`Res:` Le informazioni sulle macchine pubbliche sono informazioni generali. Il sistema dovrà anche gestire il magazzino.

`Sc,Zh:` Che tipo di funzioni del magazzino il sistema deve controllare?
`Res:` Le funzioni da controllare riguardano le operazioni di prelievo e inserimento, monitorando le quantità di scorte presenti nel magazzino, per evitare un uso poco responsabile del materiale messo a disposizione dall'azienda. I rifornimenti vengono erogati puntualmente dall'azienda, ma un dipendente può andare anche in maniera autonoma a prelevare tale materiale dal magazzino.
`Sc,Zh:` Sulla parte contabile quali sono le informazioni che il sistema deve gestire?
`Res:` Il sistema deve tenere traccia di tutte le operazioni monetarie dell'azienda; quindi, bisogna memorizzare gli stipendi, le retribuzioni variabili che ogni dipendente professionale deve percepire, le informazioni riguardanti il pagamento delle fatture da parte dei clienti, e tutte le altre spese, come, ad esempio, bollette o acquisto rifornimenti.

1.3 Descrizione del sistema da realizzare

Il progetto proposto consiste nella realizzazione di un'architettura costituita da quattro componenti necessari per soddisfare le richieste dell'azienda implementando i servizi necessari.

1.3.1 Gestione dei dati

Abbiamo un database per memorizzare tutte le informazioni e i dati di interesse; il database è la soluzione migliore in sostituzione dei file cartacei e dei file excel. Nel database verranno salvate, in generale, tutte le informazioni riguardanti i clienti, commesse, cioè i lavori assegnati, i dipendenti e i flussi monetari.

Per ogni dipendente saranno memorizzati informazioni personali e lavorative, come, ad esempio, il codice fiscale, il nome e cognome, codice IBAN per il versamento dello stipendio; poi ci sono informazioni riguardanti il tipo di dipendente e, di conseguenza, gli importi orari per il suo lavoro; dovranno essere registrati tutti i prelievi fatti dal magazzino da parte del dipendente.

Per quanto riguarda i clienti dovranno essere salvate le informazioni generali, come l'email, la località dell'azienda e altre informazioni di contatto.

Dovranno essere memorizzate le commesse; le informazioni di interesse sono il codice della merce, le quantità, il tessuto e altre informazioni tecniche che la riguardano; inoltre ad ogni commissione è associato un listino prezzi con il valore del modello commissionato, sia per il cliente che per il dipendente.

Si dovrà tenere traccia di tutte le suddivisioni del lavoro fatte ai vari dipendenti professionali, anche quelle che riguardano i trasferimenti interni; quindi saranno salvate le informazioni sulla quantità e sul valore economico.

Infine si dovranno memorizzare i flussi monetari; le informazioni di interesse sono quelle che riguardano le spese e le entrate. Per le spese dovranno essere memorizzate informazioni legate allo stipendio di ogni dipendente, la retribuzione variabile che riguarda solo i dipendenti professionali e che si somma al loro stipendio, e anche altre spese generiche che possono essere, ad esempio, le bollette; per tutti questi

flussi monetari si dovranno salvare la data, l'importo e altre informazioni. Per le entrate si registrano, invece, i pagamenti effettuati dai clienti.

Si dovranno memorizzare tutte le informazioni relative alla documentazione della merce e ad ogni suo spostamento; i DDT verranno suddivisi in contabili e non contabili e saranno salvate le informazioni sulle fatture emesse dall'azienda.

1.3.2 Gestione amministrativa

Successivamente si dovrà implementare un'applicazione desktop per elaborare e visualizzare i dati del database, dotata di interfacce grafiche per velocizzare e migliore le operazioni effettuate dal responsabile. Il contabile potrà accedere a un sottoinsieme delle funzionalità messe a disposizione del responsabile; questo dipendente può solo effettuare operazioni finanziarie.

1.3.3 Gestione dei dipendenti

Siccome i dipendenti non hanno un computer nella loro postazione, verrà sviluppata un'applicazione mobile necessaria per il coordinamento e il miglioramento del lavoro in azienda. Come descritto in precedenza, l'applicazione deve gestire le prenotazioni sulle macchine pubbliche; ci dovranno essere delle interfacce diverse per i diversi tipi di dipendenti. L'applicazione dovrà migliorare la gestione del lavoro permettendo agli utenti di essere avvisati sulle disponibilità delle macchine pubbliche; essa velocizzerà tutte le operazioni svolte dai vari dipendenti. I dipendenti che utilizzeranno l'applicazione mobile sono il dipendente generico, il cuoco e il dipendente professionale.

1.3.4 Scelta architetturale

Per poter collegare le tre componenti sopra citate si è scelto di utilizzare una quarta componente. Un server è necessario per gestire tutte le richieste che vengono fatte al database. Come prima cosa, per evitare che personale non autorizzato acceda a informazioni dell'azienda, verranno fornite delle credenziali che verranno controllate ad ogni richiesta. Inoltre, ci si occuperà di controllare che i dati inseriti dagli utenti, per fare delle richieste specifiche o degli inserimenti, siano corretti. Nel caso dell'inserimento di informazioni non corrette il server avviserà gli utenti; nel caso in cui le richieste siano corrette restituirà i dati contenuti nel database.

1.4 Glossario dei termini

Di seguito viene riportato un glossario con le parole più significative utilizzate nel testo precedente,all'interno viene riportato il nome, la descrizione il tipo (se riferito all'azienda o al linguaggio tecnico) ed eventuali sinonimi.

Termini	Descrizione	Tipo	Sinonimi
Fornitori	Azienda che ci fornisce vari prodotti, come, ad esempio, filo, oppure aghi.	Aziendale	Grossista
Semilavorato	Materiale tessile che ha subito una prima lavorazione, come, ad esempio, il taglio del modello.	Aziendale	Nessun sinonimo
Commissione	Lavoro che viene assegnato e retribuito all'azienda.	Aziendale	Incarico
Dipendente	Persona fisica che svolge una mansione all'interno dell'azienda.	Aziendale	Lavoratore
Documento di trasporto (DDT)	Documento che contiene informazioni importanti sulla merce.	Aziendale	Bolla
Documento di lavorazione	Documento che contiene informazioni sulla lavorazione che deve essere effettuata su quella merce.	Aziendale	Scheda di lavorazione
Fattura	Documento che viene generato dall'azienda e che contiene informazioni sul prodotto finale, nonchè l'importo totale che deve essere versato all'azienda.	Aziendale	Ricevuta fiscale
Situazione tributaria	Tasse pagate dall'azienda.	Aziendale	Tasse
Modello	Tipo di lavoro che deve essere fatto in una commessa.	Aziendale	Nessun sinonimo
Listino prezzi cliente	Lista complessiva dei prezzi da far visionare ai clienti in base al tipo di lavoro richiesto.	Aziendale	Catalogo prezzi
Listino compensi dipendenti	Lista complessiva dei compensi che vengono retribuiti ai dipendenti professionali nello stipendio.	Aziendale	Catalogo retribuzioni
Database	Collezione di dati.	Tecnico	DB
App	Applicazione mobile.	Tecnico	Nessun sinonimo
Interfacce grafiche	Schermate video che servono per facilitare il lavoro dell'utente e per renderlo visivamente più gradevole.	Tecnico	GUI
Statistiche	Applicazione di formule matematiche sui dati raccolti per estrarre ulteriori informazioni.	Tecnico	Nessun sinonimo
Sistema di autenticazione	Meccanismo fondamentale per riconoscere le persone autorizzate a compiere determinate azioni.	Tecnico	Convalida delle credenziali
Memorizzare	Salvare in maniera permanente su un dispositivo hardware.	Tecnico	Salvare
Macchine pubbliche	Macchinari che possono essere utilizzati da tutti i dipendenti, ma solo uno alla volta.	Aziendale	Nessun sinonimo
Scorte	Quantità di materiale conservato all'interno del magazzino.	Aziendale	Nessun sinonimo
Ciclo di lavoro	Insieme di operazioni da compiere su un semilavorato.	Aziendale	Tipo di lavorazione

Figura 1.1. Dizionario dei termini

2

Descrizione generale del sistema complessivo

In questo capitolo descriviamo il sistema complessivo in generale, analizzeremo i requisiti specifici, i casi d'uso e il schema E/R per il database.

2.1 Analisi dei requisiti

In questa sezione vengono mostrati i requisiti funzionali e non funzionali; successivamente, vengono presentati i casi d'uso con una breve descrizione degli stessi.

Nella Figura 2.1 viene visualizzata tutta l'analisi dei requisiti, sia funzionali che non funzionali. I requisiti funzionali sono dei requisiti che definiscono una funzione del sistema appartenente ad una o più delle sue componenti; servono per soddisfare una richiesta fatta, ad esempio, dal cliente. I requisiti non funzionali non sono pensati per a soddisfare delle richieste, ma influenzano in maniera decisiva il sistema, in quanto ne descrivono i vincoli e le proprietà. Ci possono essere vincoli di natura temporale, vincoli sul processo di sviluppo e sugli standard da adottare. Tali vincoli ci permettono di definire come vanno implementate certe funzionalità.

L'insieme dei requisiti è stato suddiviso in quattro sottoinsiemi, ciascuno dei quali corrisponde a una componente dell'architettura, ovvero da database, server, client in Python e mobile.

Per una maggiore comprensione delle funzionalità che il sistema deve avere, essi vengono riportati come una lista numerata contenente il nome del requisito funzionale e una sua breve descrizione.

Nella Figura 2.2 troviamo tutti i requisiti uguali per tutte le componenti; infatti quando abbiamo, ad esempio, che viene effettuata una richiesta di inserimento da parte dal client, che può essere mobile o Python, essa viene ricevuta dal server, che la effettua sul database; quindi, questi requisiti sono riportati una sola volta.

Nella Figura 2.3, invece, abbiamo i requisiti funzionali che hanno solo i client Python e mobile; essi vengono riportati nello stesso modo. Alcuni di questi requisiti sono diversi in quanto le viste sono presenti solo lato client, cioè la visualizzazione dettagliata dei dati e quella nelle liste avviene solo tramite queste applicazioni.

2.2 Diagrammi dei casi d'uso

I casi d'uso sono una tecnica utilizzata nei processi di ingegneria del software per effettuare in maniera esaustiva e non ambigua, la raccolta dei requisiti al fine di produrre software di qualità. Servono per valutare ogni requisito focalizzandosi sugli attori che interagiscono col sistema: valutandone le varie interazioni; essi descrivono e individuano gli scenari elementari di utilizzo del sistema da parte degli attori che si interfacciano con esso. Per una migliore spiegazione del diagramma dei casi d'uso si è scelto di suddividere il digramma in varie parti.

Nella Figura 2.4 viene riportata la parte del sistema che gestisce i dati dei dipendenti, dei clienti, del listino prezzi e delle macchine pubbliche; questa è la parte che viene modificata con meno frequenza, ed è usata soprattutto per consultare i dati. L'unico utente che può accedere in questa sezione è il responsabile utilizzando l'applicazione client Python.

Nella Figura 2.5 viene riportata la parte strategica, che descrive nel medio e lungo periodo l'andamento dell'azienda; infatti questi casi d'uso rappresentano la possibilità per gli attori di visualizzare, inserire e modificare i dati finanziari, cioè le entrate, le uscite e tutte le altre spese. Gli utenti che possono accedere a queste funzionalità sono il responsabile e il contabile utilizzando l'applicazione client Python. Questi casi d'uso comprendono anche la gestione del magazzino; l'unico utente che può accedere in questa sezione è il responsabile, sempre con la stessa applicazione.

Nella Figura 2.6 viene riportata la parte relativa alla lavorazione, una parte del sistema che viene modificata molto frequentemente; essa gestisce le commesse, le suddivisioni lavoro e i documenti DDT. Infatti queste sono tutte informazioni con cui il responsabile lavora ogni giorno; questi casi d'uso rappresentano la possibilità per l'utente di inserire e modificare i dati lavorativi. Il responsabile è l'unico utente che può accedere a queste funzionalità; il contabile può solo accedere alla parte riguardante i documenti DDT.

Nella Figura 2.7 viene riportata la parte del sistema mobile che coinvolge principalmente i dipendenti dell'azienda con i relativi casi d'uso. I dipendenti che possono accedere all' applicazione sono i dipendenti generici, quelli professionali e il cuoco. Il cuoco accede alle funzionalità, riguardanti la gestione del magazzino, a cui possono accedere anche il dipendente professionale e il generico; il generico accede a funzionalità esclusive della suddivisione, come l' inserimento e la cancellazione. Infine abbiamo le funzionalità accessibili solo dal dipendente professionale, il quale può gestire le prenotazioni per le macchine pubbliche, i trasferimenti di lavoro, può visualizzare i messaggi e le suddivisioni del lavoro.

2.3 Casi d'uso

I casi d'uso di questa architettura sono numerosi, però sono simili tra di loro; quindi ne riportiamo solo alcuni; gli altri sono analoghi.

2.3.1 Visualizzare una lista dei dipendenti

Questo caso d'uso si verifica qualora il responsabile voglia visualizzare tutti i dipendenti dell'azienda; verranno mostrati tutti i dipendenti presenti nel database. Il

caso d'uso inizia quando il responsabile interagisce con l'interfaccia Python; subito dopo viene inoltrata una richiesta al server che restituisce tutte le informazioni necessarie sui dipendenti. Infine, l'applicazione visualizza una lista dove ogni elemento contiene le informazioni riguardanti un singolo dipendente.

2.3.2 Dettagli di un dipendente

Questo caso d'uso si verifica qualora il responsabile voglia visualizzare nel dettaglio i dati di un dipendente dell'azienda; verranno mostrati i dati di quel dipendente contenuti nella tabella "dipendente" del database. Il caso d'uso inizia quando il responsabile interagisce con la lista dei dipendenti; verrà inoltrata una richiesta al server che restituisce tutte le informazioni di quel dipendente. Infine, l'applicazione visualizza, all'interno di una lista, tutte le informazioni di quel dipendente.

2.3.3 Inserimento di un dipendente

Questo caso d'uso si verifica qualora il responsabile voglia inserire un nuovo dipendente. Il caso d'uso inizia quando l'utente interagisce con l'interfaccia Python; viene mostrata una form da compilare con i dati richiesti; tramite una richiesta al server il nuovo dipendente viene salvato nel database; nel caso di fallimento, il responsabile viene avvisato dell'errore che si è verificato.

2.3.4 Modifica di una suddivisione del lavoro

Questo caso d'uso si verifica qualora il responsabile voglia modificare una suddivisione del lavoro. Il caso d'uso inizia quando l'utente interagisce con la lista delle suddivisioni del lavoro; verrà inoltrata una richiesta al server che restituisce tutte le informazioni di quella suddivisione del lavoro che possono essere modificate. L'utente modifica i campi di interesse e, con la successiva richiesta, il server effettua la modifica dell'elemento. Nel caso di fallimento il responsabile viene avvisato dell'errore che si è verificato.

2.3.5 Cancellazione di una suddivisione lavoro

Questo caso d'uso si verifica qualora il responsabile voglia eliminare una suddivisione del lavoro. Il caso d'uso inizia quando l'utente interagisce con la lista delle suddivisioni del lavoro; verrà inoltrata una richiesta al server, che cancellerà l'elemento selezionato e restituirà un messaggio di conferma.

2.4 Progettazione del database

In questa sezione viene mostrato lo schema E/R e gli schemi relazionali del database. Lo scherma Entità - Relazione è uno schema concettuale di dati e, come tale, fornisce una serie di strutture, atte a descrivere la realtà in una maniera facile da comprendere e che prescinde dai criteri di organizzazione dei dati nei calcolatori.

Lo schema E/R usa simboli grafici per favorire l'immediatezza della comprensione; sono infatti schemi essenzialmente grafici. L'entità rappresenta una classe di oggetti del mondo reale (oggetti sia materiali che immateriali), è caratterizzata da un nome e viene rappresentata con un rettangolo. La relazione rappresenta un legame logico tra entità. Ogni entità possiede degli attributi e un identificatore.

2.4.1 Schema E/R

Nella Figura 2.8 viene mostrato lo schema E/R del database dell'architettura. Questo è lo schema E/R finale; per arrivare a questo livello di dettaglio la prima versione dello schema subisce una serie di operazioni. La prima è il calcolo delle ridondanze, vengono valutati gli attributi, per vedere se conviene toglierne qualcuno oppure aggiungerlo; gli attributi coinvolti in questo processo sono quelli che possono essere derivati come combinazione degli altri. Ovviamente basandosi su un calcolo delle ridondanze si prende una decisione; questo calcolo coinvolge il tipo di operazione da eseguire, il numero di entità coinvolte e la frequenza con la quale viene eseguita questa operazione.

Successivamente si procede con la normalizzazione dello schema; si tratta di un procedimento volto all'eliminazione della ridondanza informatica e del rischio di incoerenza del database. Questo processo si fonda su un semplice criterio: se una relazione presenta più concetti tra di loro indipendenti la si decompone in relazioni più piccole, una per ogni concetto. Poi abbiamo l'eliminazione delle generalizzazioni; basandosi sempre sulle operazioni che devono essere effettuate sul database si decide quali entità bisogna tenere e quali, invece, raggruppare nelle rimanenti. Le generalizzazioni descrivono un concetto a cui ne sono collegati altri e che ne rappresentano dei casi particolari. Si decide se far rimanere il concetto specifico, o quello generale, oppure, se necessario, anche tutti e due. Infine, eliminiamo gli attributi multi-valore che diventano delle entità collegate a quella di partenza.

2.4.2 Schema logico del database

Nelle Figure 2.9 e 2.10 viene mostrato lo schema relazionale del database dell'architettura. Dopo aver realizzato lo schema E/R, dopo aver effettuato tutte le operazioni per migliorarlo e raffinarlo, si arriva al livello logico. Questo livello è necessario in quanto il solo schema E/R non ci permette di avere una visione chiara e definita delle tabelle e dei vincoli che comporranno il database. Abbiamo, infatti, bisogno di rappresentarli nello stesso modo in cui saranno rappresentati nel database. Quindi si effettuerà questa traduzione; alla fine si otterrà l'insieme delle tabelle con le relative proprietà che saranno implementate in SQL; infatti troviamo tutti i loro attributi, le chiavi primarie e i vincoli d'integrità referenziale con le altre tabelle. Un vincolo d'integrità referenziale è un vincolo di tipo interrelazionale, ovvero una proprietà, la quale, per essere soddisfatta, richiede che ogni valore di un attributo di una relazione esista come valore di un altro attributo in un'altra relazione.

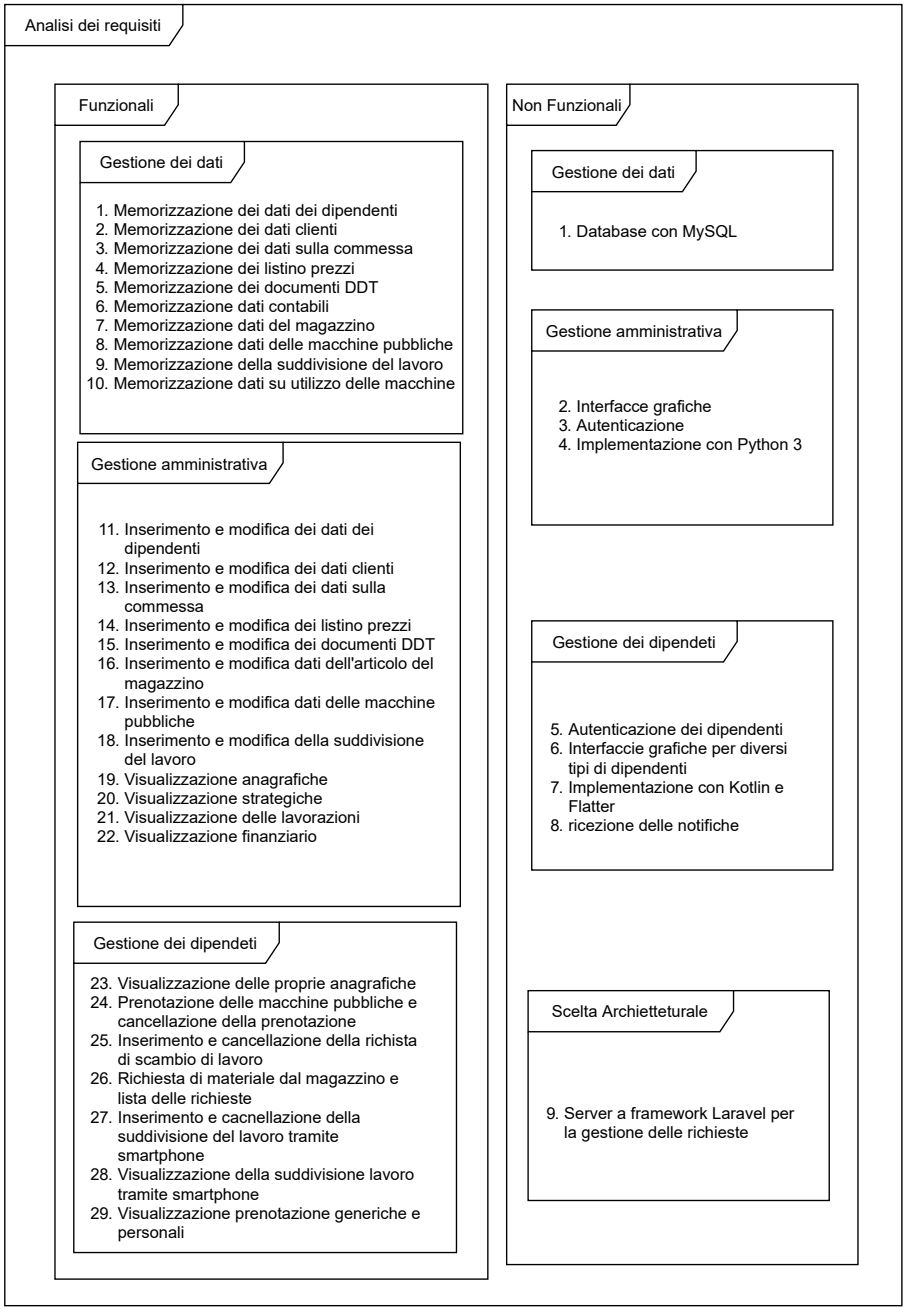

Figura 2.1. Analisi dei requisiti

Numero del requisito	*Requisito*	*Descrizione*
N.1	Memorizzazione dei dati dei dipendenti	Nel database verranno memorizzati i dati anagrafici di ogni dipendente
N.2	Memorizzazione dei dati dei clienti	Nel database verranno memorizzati i dati relativi ad ogni cliente
N.3	Memorizzazione dei dati sulla commessa	Nel database verranno memorizzati i dati appartenenti ad ogni commessa
N.4	Memorizzazione dei listini prezzi	Nel database verranno memorizzati i dati relativi al listino prezzi
N.5	Memorizzazione dei documenti DDT	Nel database verranno memorizzati i dati dei DDT di ogni commessa
N.6	Memorizzazione dei dati contabili	Nel database verranno memorizzati i dati contabili relativi agli ingressi e alle uscite
N.7	Memorizzazione dei dati del magazzino	Nel database verranno memorizzati i dati relativi alle scorte presenti nel magazzino
N.8	Memorizzazione dei dati delle macchine pubbliche	Nel database verranno memorizzati i dati delle macchine pubbliche presenti in azienda
N.9	Memorizzazione della suddivisione del lavoro	Nel database verranno memorizzati i dati delle suddivisioni del lavoro tra i vari dipendenti
N.10	Memorizzazione dei dati sull'utilizzo delle macchine pubbliche	Nel database verranno memorizzati i dati sulle prenotazioni delle macchine pubbliche
N.11	Inserimento e modifica dei dati dei dipendenti	Inserimento e modifica tramite software python dei dati anagrafici dei dipendenti memorizzati nel database
N.12	Inserimento e modifica dei dati dei clienti	Inserimento e modifica tramite software Python dei dati dei clienti memorizzati nel database
N.13	Inserimento e modifica dei dati di una commessa	Inserimento e modifica tramite software Python dei dati di ogni commessa memorizzati nel database
N.14	Inserimento e modifica dei listini prezzi	Inserimento e modifica tramite software Python dei dati del listino prezzi memorizzati nel database
N.15	Inserimento e modifica dei documenti DDT	Inserimento e modifica tramite software Python dei dati relativi ai DDT memorizzati nel database
N.16	Inserimento e modifica dei dati degli articoli del magazzino	Inserimento e modifica tramite software Python dei dati delle scorte presenti in magazzino memorizzati nel database
N.17	Inserimento e modifica dei dati delle macchine pubbliche	Inserimento e modifica tramite software Python dei dati delle macchine pubbliche memorizzati nel database
N.18	Inserimento e modifica della suddivisione del lavoro	Inserimento e modifica tramite software Python dei dati sulla suddivisione del lavoro memorizzati nel database
N.19	Consultazione dei propri dati anagrafici	L'applicazione Python e quella mobile devono permettere ai dipendenti di visualizzare i loro dati anagrafici

Figura 2.2. Descrizione dei requisiti (Parte 1)

N.20	Visualizzazione dei dati strategici	L'applicazione Python deve permettere di visualizzare alcuni dati strategici per aggiornarsi sull'economia dell'azienda
N.21	Visualizzazione delle lavorazioni	L'applicazione Python deve permettere di visualizzare i dati della lavorazione che variano nel breve periodo
N.22	Visualizzazione dei dati finanziari	L'app Python deve permettere di visualizzare i dati finanziari dell'azienda, ovvero le entrate e le uscite
N.23	Prenotazione delle macchine pubbliche e cancellazione di una prenotazione tramite smartphone	L'app mobile deve permettere ad ogni dipendente professionale di prenotarsi per poter utilizzare le macchine pubbliche
N.24	Inserimento e cancellazione di una richiesta di scambio lavoro della lista delle richieste tramite smartphone	L'app mobile deve permettere ai dipendenti professionali di effettuare una richiesta di scambio di quantità di lavoro verso un altro dipendente professionale; essa deve anche permettere di cancellare una richiesta. L'app dovrà visualizzare una lista delle richieste effettuate
N.25	Richiesta materiale dal magazzino e lista delle richieste tramite smartphone	L'app mobile deve permettere ai dipendenti di effettuare una richiesta per accedere alle risorse del magazzino e dovrà visualizzare una lista delle richieste effettuate
N.26	Inserimento e cancellazione della suddivisione del lavoro tramite smartphone	L'app mobile deve permettere ai dipendenti generici di inserire una suddivisione del lavoro ai dipendenti professionali; essa, inoltre, deve dare la possibilità di visualizzare le suddivisioni di un determinato dipendente in una lista.
N.27	Visualizzazione della suddivisione del lavoro tramite smartphone	L'app mobile deve permettere al dipendente professionale di visualizzare tutte le suddivisioni del lavoro che gli sono state assegnate all'interno di una lista
N.28	Visualizzazione delle prenotazioni generiche e personali tramite smartphone	L'app mobile deve permettere all'utente professionale di visualizzare le sue prenotazioni all'interno di una lista e, sempre in una lista, deve poter visualizzare le prenotazioni generiche di una determinata macchina pubblica

Figura 2.3. Descrizione dei requisiti (Parte 2)

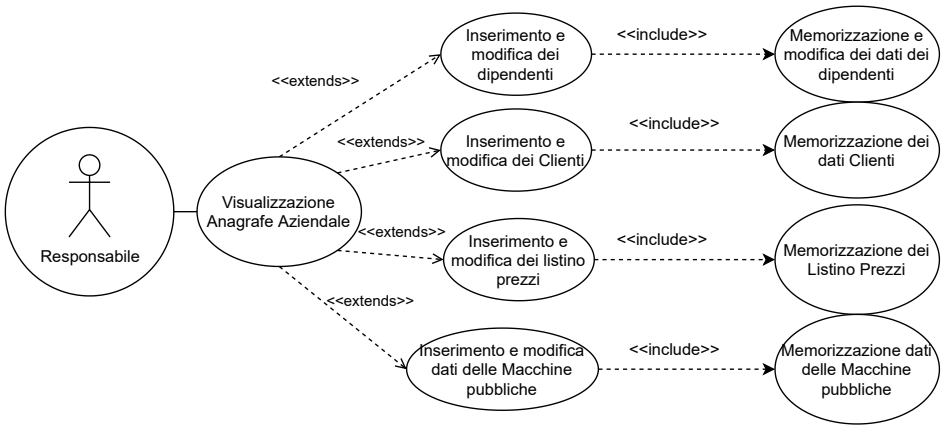

Figura 2.4. Parte relativa alle Anagrafiche

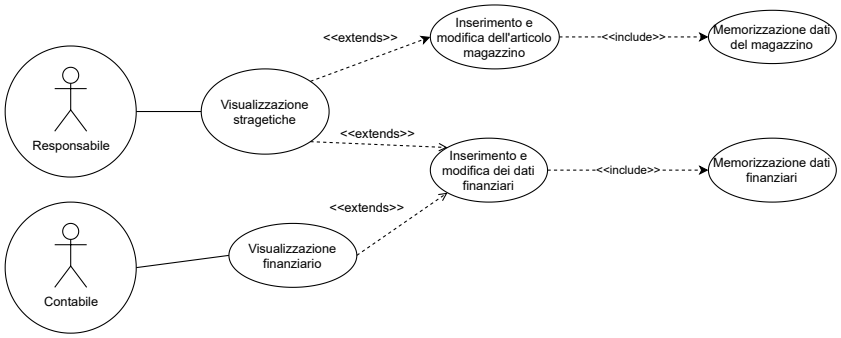

Figura 2.5. Parte relativa alle Strategiche

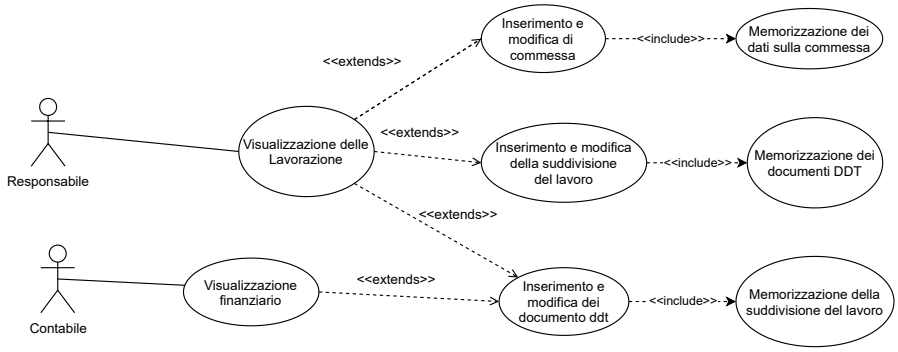

Figura 2.6. Parte relativa alla Lavorazione

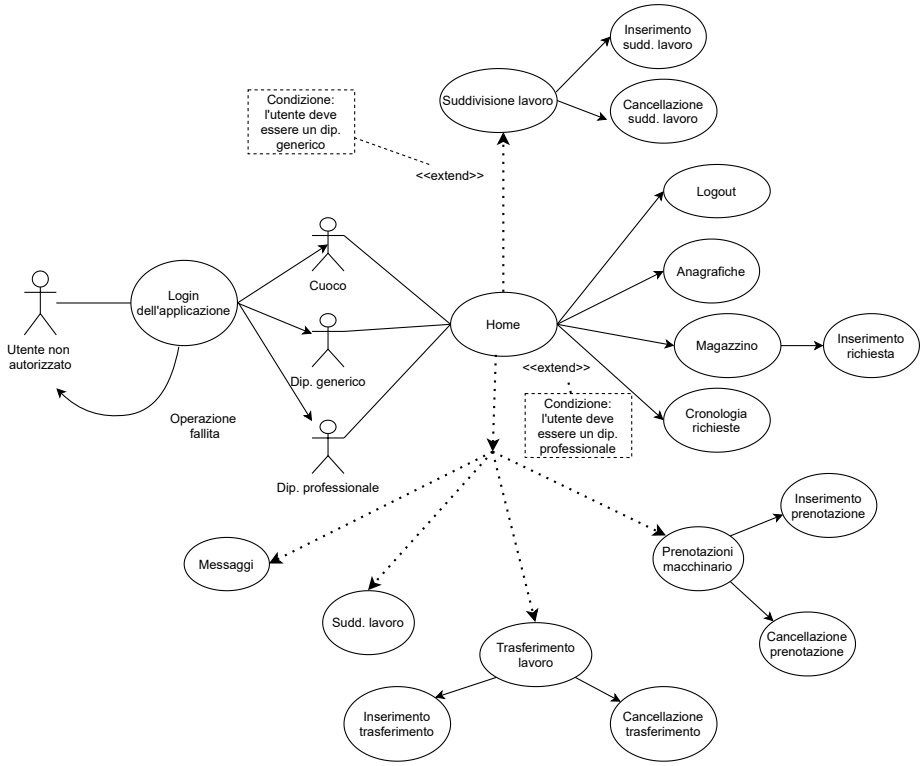

Figura 2.7. Parte relativa all'applicazione Mobile

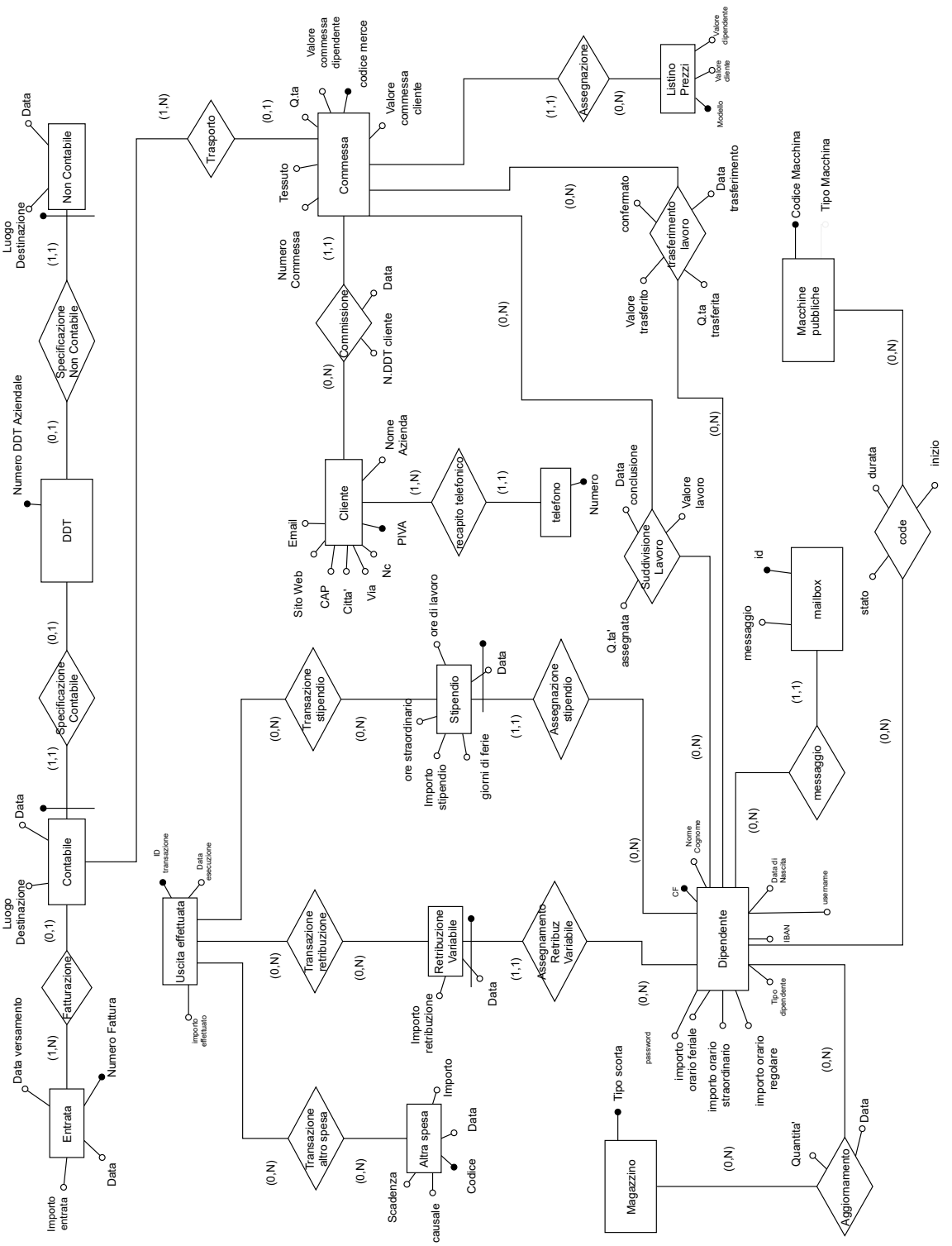

Figura 2.8. Schema E/R del database

Entità	Traduzione
Cliente	cliente(Piva, NomeAzienda, Email, Web, Cap, Citta', Via, NumCiv)
Telefono	telefono(Numero, cliente)
Commessa	commessa(CodiceMerce,Client, NumCommessa, Tessuto, Qtà, ValDip, ValCliente, NDDTCliente, DataIngresso, DDTAziendale, ModComm)
Listino prezzi	listino_prezzi(Modello, ValClient, ValDipen)
DDT	DDT(NumeroDDT)
Contabile	contabile(NumeroDDT,LuogoDestinazione,DataTrasferimento)
Non contabile	non_contabile(NumeroDDT,LuogoDestinazione,DataTrasferimento)
Entrata	entrata(NumeroFattura, Datafatt, ImportoEntr, DataVersamento)
Suddivisione lavoro	suddivisione_lavoro(Merce, Dipendente, QtàAssegnata, QtàTrasferita, ValoreTrasferito, DataTrasf, ValoreLavoro)
Dipendente	dipendente(CF, NomeCogn, DatNascita, IBAN, TipoDip, ImpOrFeriale, ImpOrStraordinario, ImpOrRegolare)
Stipendio	stipendio(DataStip, Dip, OrLavoro, OrStraordinario, GiorniFerie, ImportoStip)
Retribuzione variabile	retribuzione_variabile(DataRetrVar, DipProf, ImportoRetribuzione)
Altra spesa	altra_spesa(Codice, Importo, Scadenza, Causale, DataSpesa)
Uscita effettuata	uscita_effettuata(ID, DataEsecuzione, ImportoEffettuato)
Transazione altra spesa	transazione_altra_spesa(CodSpe, IDTran)
Transazione retribuzione	transazione_retribuzione(IDTran, DataRetrib, DipCf)
Transazione stipendio	transazione_stipendio(IDTran, DataStip, DipCF)
Trasferimento lavoro	trasferimento_lavoro(id,codice_trasf,valore_trasferito,commessa,dipendente,data_trasferimento,valore_trasferito,quantita_trasferita,confermato)
Macchine pubbliche	Macchine_pubbliche(codice_macchina,tipo_macchina)
Coda	coda(id,dipendente,codice_macchina,inizio,durata)
Mailbox	mailbox(id,dipendente,messaggio)
Magazzino	magazzino(tipo_scorta)
Aggiornamento	aggiornamento(id,dipendente,data,quantita,magazzino)

Figura 2.9. Traduzione delle entità nel modello logico

TRADUZIONE	VINCOLO DI RIFERIMENTO
cliente(Piva, NomeAzienda, Email, Web, Cap, Citta', Via, NumCiv)	*
telefono(Numero, cliente)	cliente->Cliente.P-iva
commessa(CodiceMerce,Client, NumCommessa, Tessuto, Qtà, ValDip, ValCliente, NDDTCliente, DataIngresso, DDTAziendale, ModComm)	Client->Cliente.P-iva ModComm->ListinoPrezzi.Modello DDTAziendale->DDT.NumeroDDT
listino_prezzi(Modello, ValClient, ValDipen)	*
DDT(NumeroDDT)	NumFattura->Entrata.NumeroFattura
contabile(NumeroDDT,LuogoDestinazione,DataTrasferimento)	NumeroDDT->DDT.NumeroDDT
non_contabile(NumeroDDT,LuogoDestinazione,DataTrasferimento)	NumeroDDT->DDT.NumeroDDT
entrata(NumeroFattura, Datafatt, ImportoEntr, DataVersamento)	*
suddivisione_lavoro(Merce, Dipendente, QtàAssegnata, QtàTrasferita, ValoreTrasferito, DataTrasf, ValoreLavoro)	Merce->Commessa.CodiceMerce Dipendente->Dipendente.CF
dipendente(CF, NomeCogn, DatNascita, IBAN, TipoDip, ImpOrFeriale, ImpOrStraordinario, ImpOrRegolare)	*
stipendio(DataStip, Dip, OrLavoro, OrStraordinario, GiorniFerie, ImportoStip)	Dip->Dipendente.CF
retribuzione_variabile(DataRetrVar, DipProf, ImportoRetribuzione)	DipProf->Dipendente.CF
altra_spesa(Codice, Importo, Scadenza, Causale, DataSpesa)	*
uscita_effettuata(ID, DataEsecuzione, ImportoEffettuato)	*
transazione_altra_spesa(CodSpe, IDTran)	CodSpe->AltraSpesa.Codice IDTran->UscitaEffettuata.ID
transazione_retribuzione(IDTran, DataRetrib, DipCf)	DipCF->RetribuzioneVariabile.DipProf DataRetrib-> RetribuzioneVariabile. DataRetrVar IDTran->UscitaEffettuata.ID
transazione_stipendio(IDTran, DataStip, DipCF)	DipCF->Stipendio.Dip DataStip-> Stipendio. DataStip IDTran->UscitaEffettuata.ID
trasferimento_lavoro(id,codice_trasf,valore_trasferito,commessa,dipendente,data_trasferimento,valore_trasferito,quantita_trasferita,confermato)	Commessa->commessa.id Dipendente->dipendente.CF
Macchine_pubbliche(codice_macchina,tipo_macchina)	*
coda(id,dipendente,codice_macchina,inizio,durata)	dipendente->dipendente.CF codice_macchina->macchine_pubbliche.codice_macchina
mailbox(id,dipendente,messaggio)	dipendente->dipendente.CF
magazzino(tipo_scorta)	*
aggiornamento(id,dipendente,data,quantita,magazzino)	dipendente->dipendente.CF magazzino->magazzino->tipo_scorta

Figura 2.10. Lo schema logico del database

3

Progettazione della componente relativa alle risorse interne

In questo capitolo verranno presentati i diagrammi delle classi, i diagrammi delle sequenze e i diagrammi delle attività. Questi saranno analizzati e spiegati per facilitarne la comprensione, in quanto sono dei diagrammi molto importanti per capire il funzionamento del sistema. Infine saranno inseriti i mockup; che danno l'idea di come il committente desiderava fosse l'interfaccia grafica.

3.1 Diagrammi delle classi

I diagrammi delle classi consentono di descrivere tipi di entità, con le loro caratteristiche e le loro eventuali relazioni. Gli strumenti concettuali utilizzati sono il concetto di classe, più altri che derivano dal paradigma della programmazione ad oggetti. Questi diagrammi sono utilizzati per descrivere, sostanzialmente, qualsiasi contesto a qualsiasi livello di astrazione.

3.1.1 Diagramma delle classi dei dipendenti lato server

Nella Figura 3.1 viene mostrato il diagramma delle classi lato server.

Per lo sviluppo del sistema si aveva bisogno di un pattern da seguire, in quanto l'intera architettura è abbastanza grande e complessa. Un pattern consiste in una soluzione progettuale a un problema ricorrente; quello selezionato è il pattern MVC, cioè Model, View, Controll; quindi abbiamo implementato, lato server, per ogni tabella del database, una classe model e una classe controller.

La classe model è la classe che si interfaccia direttamente con il database. Sono stati definiti, all'interno di questa classe, alcuni attributi che corrispondono a quelli che possono essere inseriti all'interno della tabella; altri, invece, contengono il nome dell'attributo che corrisponde alla chiave primaria della tabella. Poi vi è un attributo che contiene il nome della tabella corrispondente. Nella classe model abbiamo dei metodi che servono per rappresentare i vincoli d'integrità referenziale tra questa tabella e le altre. Infine, abbiamo dei metodi che ci restituiscono gli elementi della tabella; uno ci restituisce tutti gli elementi interni alla tabella; un altro, invece, ci restituisce solo gli elementi ottenuti dalla ricerca basata sull'uso di una chiave;

l'ultimo, infine, ci restituisce gli elementi ottenuti dalla ricerca basata sull'uso della chiave associata alle nostre anagrafiche, cioè il codice fiscale.

La classe controller, invece, estende una classe controller più generica; essa contiene tutti i metodi per eseguire le operazioni di inserimento, modifica e cancellazione. La classe controller più specifica contiene un'istanza del model corrispondente; infatti solo i controller possono interfacciarsi con i model; sull'istanza del model vengono eseguiti i vari metodi sopra descritti. In alcune classi ci possono essere, anche, altri metodi che servono per effettuare delle operazioni prima o dopo l'esecuzione di altre, ad esempio, prima o dopo una cancellazione, un inserimento, etc. Un esempio di questi metodi è quello che, prima di cancellare una prenotazione ad un macchinario, inserisce, nella tabella dei messaggi, un messaggio per ogni dipendente, per avvisarlo che la macchina è libera.

I metodi dei vari controller sono associati a delle rotte. Laravel ci permette di implementare questo meccanismo dove, ad ogni rotta, è associato un controller, e un metodo di quel controller. Abbiamo, inoltre, creato le rotte in maniera tale che siano suddivise per tipologia di utente, inserendo nelle stringhe le parole "pro", "gen", etc; inoltre, nella rotta si possono specificare dei parametri che possono essere utilizzati nel metodo del controller.

L'ultima cosa da dire sul server è che abbiamo implementato delle classi Request, una per ogni operazione di inserimento dati da parte dell'utente, cioè per le operazioni di modifica, cancellazione e inserimento. Quando si lavora con dati inseriti dall'utente si ha la necessita di implementare dei meccanismi di sicurezza, in quanto l'utente può, sostanzialmente, inserire qualsiasi valore. Quindi le classi Request servono per implementare dei vincoli, che devono essere rispettati per poter completare l'operazione richiesta. Ad esempio, se vogliamo assegnare uno suddivisione del lavoro, dobbiamo assicurarci che il codice fiscale dell'utente selezionato esista all'interno della tabella relativa ai dipendenti. In caso positivo si prosegue e la procedura si conclude con un messaggio di conferma, altrimenti l'operazione si interrompe e viene generato un messaggio che segnala l'errore.

3.1.2 Diagramma delle classi dei dipendenti lato client

Come si può osservare, nella descrizione precedente manca la terza parte del pattern, cioè la vista; questa parte è stata implementata lato client; inoltre, alle viste di una tabella è associato un controller. Nella Figura 3.2 viene mostrato il diagramma delle classi lato client implementato con Python.

Il controller, nel caso del client, non si interfaccia con una classe model, i dati vengono recuperati dal controller lato server tramite le rotte; infatti, sfruttando la libreria `requests` di Python, abbiamo implementato tutti i metodi necessari per effettuare le varie richieste GET e POST. Con tali richieste eseguiremo i metodi lato server, che gestiscono i dati dei model e li restituiscono sotto forma di file JSON. Il formato JSON (acronimo di JavaScript Object Notation) è un formato adatto all'interscambio di dati fra applicazioni Client/Server. Questo formato è basato su due tipi di strutture di dati: serie di coppie nome/valore e liste ordinate di valori, chiamate array. Esso supporta molti tipi di dato, tra cui gli array associativi e gli array, che sono utilizzati dal sistema in quanto rappresentano i risultati delle query (interrogazioni del database).

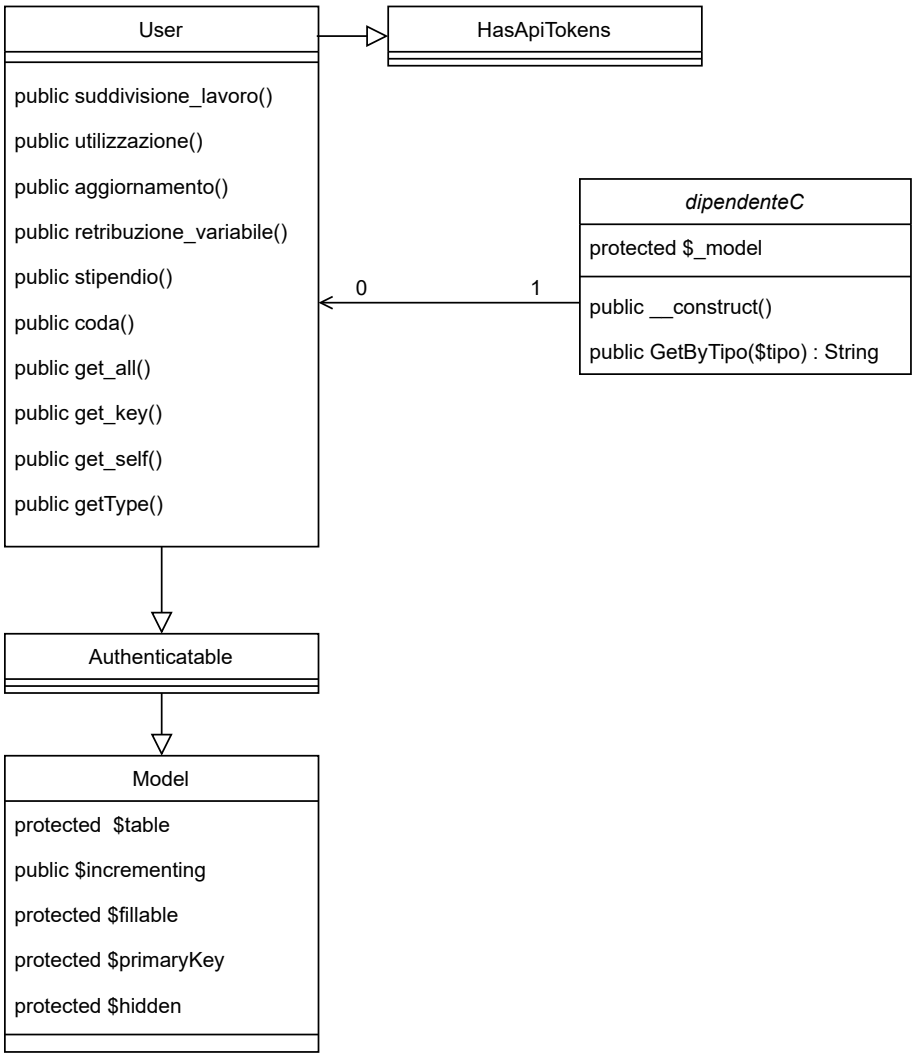

Figura 3.1. Diagramma delle classi dei dipendenti lato server

Il controller del client viene istanziato nella vista corrispondente, la quale invoca un metodo per eseguire una richiesta al server; una volta che arriva la risposta, le viste mostrano i dati all'interno di liste, contenute dentro delle finestre implementate usando PyQT5; questo è un framework utilizzato per creare dei widget.

3.1.3 Diagramma delle classi lato client Android

Nella Figura 3.3 viene mostrato il diagramma delle classi di un altro client, ovvero quello relativo ad Android. Abbiamo una situazione analoga alla precedente: abbiamo un client, solo che questa applicazione è mobile, cioè viene mandata in esecuzione

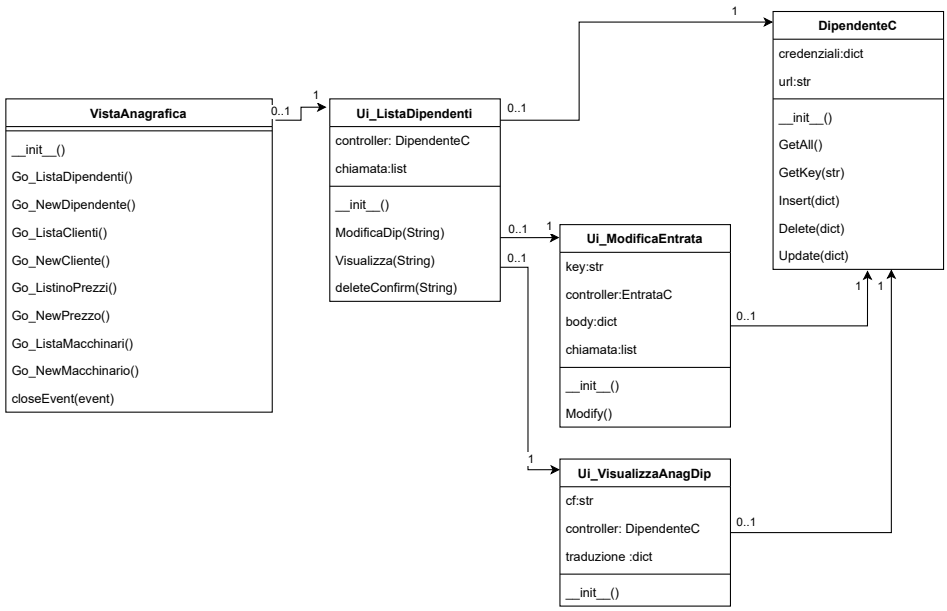

Figura 3.2. Diagramma delle classi dei dipendenti lato client

su smartphone. Questa applicazione è stata sviluppata usando come linguaggio di programmazione Kotlin; esso ci permette di sviluppare applicazioni native, cioè che possono essere mandate in esecuzione solo su dispositivi con sistema operativo Android.

L'applicazione ha all'interno delle classi che contengono le Activity; questa componente rappresenta una schermata di un'applicazione. Questa classe contiene dei metodi per effettuare l'inflating del layout; cioè, ad una istanza di questa classe è collegato un file XML che rappresenta il layout visualizzato sullo schermo dell'utente. Un'altra componente utilizzata sono i Fragment; tale componente rappresenta un behavior o una porzione di UI in un'Activity e deve essere ospitata da un'Activity a cui è legato il suo ciclo di vita. I Fragment possono essere aggiunti o rimossi a runtime; vengono utilizzati insieme alle Activity per avere diverse schermate attraverso le quali l'utente può navigare all'interno dell'app.

All'interno dell'applicazione è stata usata la componente Retrofit per poter effettuare richieste HTTP al server. La componente Moshi, invece, serve per effettuare il parsing della risposta che otteniamo dal server. Per poter gestire il contenuto delle risposte, e per definire il Body delle richieste di tipo POST, sono state utilizzate delle Data Class (classi utili per memorizzare dati).

Tulle le informazioni implementate nelle liste vengono visualizzate tramite delle RecyclerView; esse hanno la proprietà di riutilizzare delle view per rendere lo scrolling più performante; inoltre, supportano animazioni e transizioni. La RecyclerView utilizza un Adapter; esso trasforma dati e layout in modo che il RecyclerView sia in grado di mostrarli. Tutte le funzioni che fanno delle richieste al server vengono chiamate all'interno di coroutine, che ci permettono di implementare programmazione asincrona; infatti le funzioni vengono eseguite su thread separati, per non rallentare

il thread principale.

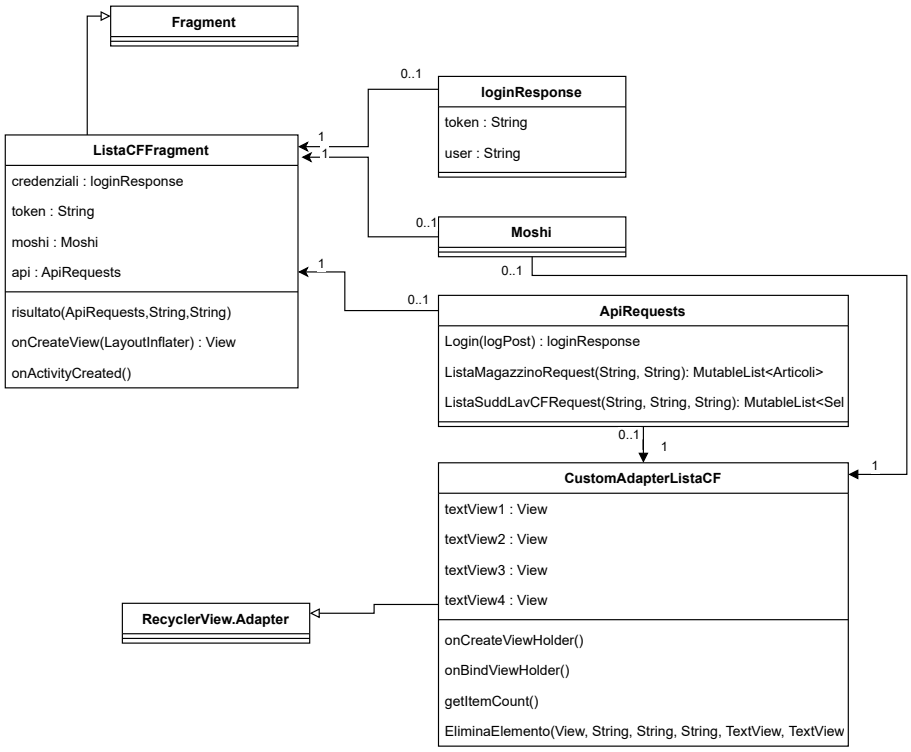

Figura 3.3. Diagramma delle classi lato client Kotlin

3.1.4 Diagramma delle classi lato client Flutter

Nella Figura 3.4 viene mostrato il diagramma delle classi di un altro client, ovvero il client Flutter. Abbiamo una situazione analoga alla precedente: abbiamo un client per i dispositivi mobile, solo che questa applicazione è cross platform, cioè può essere mandata in esecuzione sia su dispositivi con sistema operativo Android, sia su dispositivi con sistema operativo IOS. L'applicazione è sviluppata usando il framework Flutter che sfrutta il linguaggio Dart.

Rispetto al caso precedente, Flutter ci permette di creare applicazioni basate interamente sul concetto di widget; infatti le nostre schermate sono dei widget che ne contengono altri; questo framework gestisce direttamente la visualizzazione di ciascun pixel sullo schermo (Widget Rendering), garantendo la stessa visualizzazione sia su Android che su iOS. I widget si dividono in due gruppi fondamentali: widget con stato e widget senza stato. Quelli senza stato sono immutabili, cioè non cambiano durante l'esecuzione dell'app; quelli con lo stato, invece, possono cambiare. Alla nostra applicazione corrisponde un albero di widget (widget tree); quando

un widget riceve una modifica del suo stato, Flutter effettua la ricostruzione grafica di quel widget e di tutto il suo sotto albero, cioè di tutti i widget in esso contenuti.

Le classi della nostra applicazione restituiscono dei widget; le richieste HTTP vengono effettuate usando il package `http`, che fornisce le principali funzionalità per il networking. Le funzioni che fanno le richieste al server usano i Future, una sorta di promessa di una funzione di ritornare un valore in un momento futuro. Esse vengono usate per mettere in pausa l'esecuzione della funzione, in attesa che il risultato sia disponibile. Inoltre, per chiamare la funzione di parsing nell'applicazione, sono stati usati degli Isolate, che ci permettono di eseguire i task su dei background thread; tutto questo viene fatto per non rallentare l'applicazione.

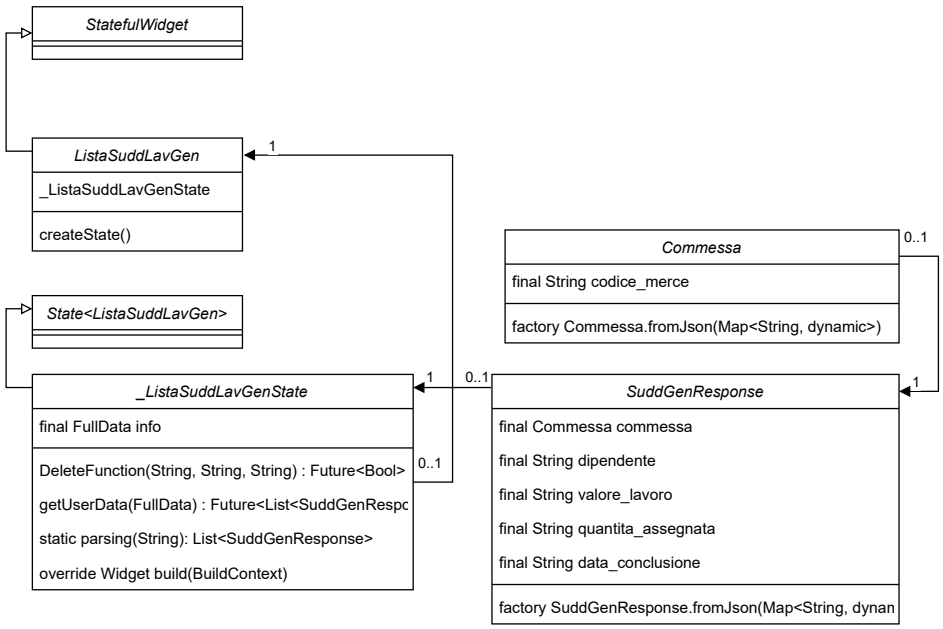

Figura 3.4. Diagramma delle classi lato client Flutter

3.2 Diagrammi delle sequenze

Un diagramma delle sequenze è un diagramma utilizzato per descrivere uno scenario. Uno scenario è una determinata sequenza di azioni in cui tutte le scelte sono state già effettuate; in pratica nel diagramma non compaiono scelte, nè flussi alternativi.

3.2.1 Diagramma delle sequenze per la visualizzazione in una lista degli oggetti del magazzino

Nella Figura 3.5 abbiamo il diagramma delle sequenze che mostra la situazione in cui l'utente voglia visualizzare gli articoli del magazzino nell'applicazione mobile in Android. Come primo requisito l'utente deve effettuare il login, cioè deve inserire le proprie credenziali (username e password). Successivamente l'applicazione Android effettua la richiesta al server e rimane in attesa; se la riposta è negativa l'utente viene avvisato; altrimenti, in base al tipo di utente, l'app visualizza la schermata di Home corrispondente.

A questo punto l'utente seleziona, da una lista di pulsanti, quello indicato per visualizzare gli articoli del magazzino; una volta effettuata l'azione, l'app passa alla schermata successiva, viene inoltrata la richiesta al server e, insieme ad essa, vengono passati due parametri nell'header. Il primo parametro è "Accept" a cui assegniamo il valore "application/json"; esso serve per comunicare al server il tipo di formato da usare per la risposta. Il secondo è "Authorization", a cui assegnamo il token che il server ci ha inviato nella risposta di login. Il token è necessario per essere riconosciuti dal server come utenti autorizzati ad effettuare certe richieste; questi due parametri vengono utilizzati, analogamente, in tutte le richieste che vedremo negli esempi successivi. Infine, il server ci restituisce i dati richiesti, viene effettuato il parsing nell'app e il risultato viene visualizzato all'interno di una RecyclerView.

3.2.2 Diagramma delle sequenze per la visualizzazione nel dettaglio di un dipendente

Nella Figura 3.6 abbiamo il diagramma delle sequenze che mostra la situazione in cui l'utente voglia visualizzare nel dettaglio i dati personali di un dipendente (codice fiscale, nome, cognome, etc) nell'applicazione Python (desktop). Come primo requisito l'utente, come nel caso dell'applicazione mobile, deve effettuare l'operazione di login; quindi verrà effettuata la richiesta al server. Se l'esito dell'operazione è negativo, come nella situazione precedente, l'utente viene avvisato; altrimenti, se l'esito è positivo, il server risponde con un messaggio di conferma che permette all'applicazione di andare avanti.

Successivamente l'utente seleziona il pulsante "Visualizza Anagrafiche"; nella schermata successiva seleziona la lista dei dipendenti e qui abbiamo una prima richiesta al server che ci restituirà la lista dei dipendenti. Per ogni dipendente nella lista abbiamo a disposizione tre pulsanti, uno per visualizzare nel dettaglio, uno per la modifica e uno per la cancellazione. Viene selezionato dall'utente quello per la visualizzazione e, da qui, viene effettuata una seconda richiesta al server, dove, oltre a passare i due parametri citati precedentemente (application/json e il token), viene passato un altro parametro, che è il codice fiscale del dipendente. In questo modo il server ci restituirà tutti i dati di quel dipendente, che verranno visualizzati all'interno di una lista.

3.2.3 Diagramma delle sequenze per l'inserimento di un dipendente

Nella Figura 3.7 abbiamo il diagramma delle sequenze che mostra la situazione in cui l'utente voglia inserire un nuovo dipendente usando l'applicazione Python. Come primo requisito l'utente deve effettuare il login, come nei casi precedenti; il secondo requisito richiesto è che l'utente non sia già presente all'interno della base di dati, altrimenti avremo, lato server, un problema con le chiavi primarie che, per definizione, sono uniche.

Quindi, dopo aver effettuato il login, navighiamo tra le schermate come nel caso precedente; solo che questa volta si sceglie il pulsante per l'inserimento di un dipendente, e non quello per la visualizzazione della lista dei dipendenti.

Una volta premuto il pulsante viene aperta una schermata dove l'utente andrà a completare i campi mostrati; alcuni di questi campi sono obbligatori, mentre altri possono essere nulli. Dopo aver compilato i campi viene effettuata una richiesta di tipo POST al server; all'interno del Body ci sono i dati inseriti dall'utente. Questi vengono validati lato server; se i controlli hanno esito negativo, si avvisa l'utente, altrimenti si procede con l'inserimento. Una volta completato l'inserimento, il server restituisce un messaggio per avvisare l'utente sull'esito positivo dell'operazione.

3.2.4 Diagramma delle sequenze per la modifica di una suddivisione lavoro

Nella Figura 3.8 abbiamo il diagramma delle sequenze che mostra la situazione in cui l'utente voglia modificare una suddivisione del lavoro usando l'applicazione Python. Come primo requisito, analogo al caso precedente, abbiamo che l'utente deve effettuare il login. Successivamente egli accede alla schermata "Vista Lavorazione" e poi clicca sul pulsante lista lavoro. Qui abbiamo la prima richiesta al server, che ci restituirà tutte le suddivisioni del lavoro, che verranno visualizzate in una lista. Per ogni elemento della lista abbiamo il pulsante della modifica. Selezionando tale pulsante viene fatta una richiesta al server. Tale richiesta è analoga al caso per la visualizzazione dettagliata; quindi, oltre ai precedenti due parametri, viene passato il valore della chiave di quella suddivisione; così il server ci restituisce tutti i dati di quell'elemento. Questi dati vengono filtrati lato client sulla base di quello che l'utente può o non può modificare; successivamente, questi valori vengono visualizzati all'interno di una lista e i vari campi obbligatori vengono contrassegnati. Infine, una volta compilati i campi, viene inoltrata la richiesta al server, i dati vengono validati e si avvisa l'utente, sia in caso di esito positivo sia in caso di esito negativo, come nei casi precedenti.

3.2.5 Diagramma delle sequenze per la cancellazione di una prenotazione ad un macchinario

Nella Figura 3.9 abbiamo il diagramma delle sequenze che mostra la situazione in cui l'utente voglia cancellare la prenotazione ad un macchinario nell'applicazione mobile cross platform. Come primo requisito dell'applicazione sviluppata in Flutter, analogamente al caso precedente, abbiamo che l'utente deve effettuare il login.

Successivamente in base al tipo di utente, viene visualizzata la Home corrispondente. All'interno della Home l'utente deve selezionare il pulsante che gli permette di visualizzare la lista delle proprie prenotazioni. A questo punto viene fatta una richiesta al server che ci restituisce la lista delle prenotazioni di quell'utente. Una volta ricevuti i dati, questi vengono visualizzati all'interno di una lista. Tale lista, al verificarsi dell'evento "onlongclick" (pressione con il dito, mantenuta per un paio di secondi, sull'elemento selezionato) visualizza un messaggio Dialog che allerta l'utente sulla possibilità di eliminare l'elemento. L'utente conferma di voler compiere tale azione e, successivamente, viene inviata una richiesta di cancellazione al server a cui passiamo la chiave dell'elemento. Il server elimina l'elemento selezionato e restituisce un messaggio per confermare l'esito positivo. L'app avvisa l'utente sull'avvenuta cancellazione dell'elemento.

3.3 Diagrammi delle attività

Il diagramma delle attività è un tipo di diagramma che permette di descrivere un processo attraverso dei grafi, in cui i nodi rappresentano le attività e gli archi l'ordine con cui esse vengono eseguite. L'attività di login viene effettuata sempre prima di accedere ad una Home; per motivi di spazio viene riportata solo in due diagrammi.

3.3.1 Diagramma delle attività per l'inserimento di un dipendente

Le azioni che l'utente compie sono quella di accedere alla Home dopo il login, selezionare la schermata "Visualizza Anagrafiche", accedere alla schermata per l'inserimento di un dipendente e inserire i dati. A seguito di ciò viene effettuata la richiesta al server che valida i dati; l'operazione può avere esito positivo, se non ci sono problemi di validazione, oppure esito negativo, se qualche parametro inserito non è corretto. Se l'esito è positivo si conclude il processo. Nella Figura 3.10 abbiamo il diagramma di attività corrispondente.

3.3.2 Diagramma delle attività per la modifica di una suddivisione lavoro

Le azioni che l'utente compie sono quella di accedere alla Home dopo il login. A seguito di ciò, l'utente accede alla schermata "Visualizzazione lavorazione", seleziona la lista delle suddivisioni del lavoro, clicca sul pulsante per modificare l'elemento. A questo punto, viene effettuata la richiesta al server che restituisce tutti i dati dell'elemento, vengono visualizzati gli attributi che possono essere modificati, con gli attuali valori. L'utente esegue le modifiche e procede con il completamento dell'operazione. A seguito di ciò, viene effettuata la richiesta al server che valida i dati; l'operazione può avere esito positivo, se non ci sono problemi di validazione, oppure esito negativo, se qualche parametro inserito non è corretto. Se l'esito è positivo si conclude il processo. Nella Figura 3.11 abbiamo il diagramma di attività corrispondente.

3.3.3 Diagramma delle attività per la cancellazione di una prenotazione ad un macchinario

Le azioni che l'utente compie sono quella di accedere alla Home professionale. Dopo il login, seleziona la schermata "Visualizzazione prenotazioni", accede alla lista delle proprie prenotazioni e seleziona l'elemento da eliminare. Viene visualizzato un Dialog per richiedere all'utente di confermare l'operazione di eliminazione. L'utente può confermare l'operazione, oppure rifiutarla. Se rifiuta il processo termina senza cancellazione; altrimenti, viene effettuata la richiesta al server per eliminare l'elemento e si conclude il processo. Nella Figura 3.12 abbiamo il diagramma di attività corrispondente.

3.3.4 Diagramma delle attività per la visualizzazione in una lista degli articoli del magazzino

Le azioni che l'utente compie sono quella di effettuare il login inserendo le proprie credenziali; questa operazione può avere esito negativo, se l'utente non esiste nel database, oppure non è autorizzato ad usare l'app; altrimenti ha esito positivo, se soddisfa entrambi i requisiti precedenti. Una volta autenticato, l'utente accede alla Home, in questo caso quella del dipendente generico; successivamente egli accede alla schermata degli articoli del magazzino, cliccando sul pulsante corrispondente. A questo punto viene effettuata la richiesta al server, e il risultato viene visualizzato in una lista. Nella Figura 3.13 abbiamo il diagramma di attività corrispondente.

3.3.5 Diagramma delle attività per la visualizzazione dettagliata di un dipendente

Le azioni che l'utente compie sono quella di effettuare il login inserendo le proprie credenziali, come nel caso precedente. Successivamente egli accede alla Home, seleziona la schermata "Visualizzazione Anagrafiche" e da quella schermata, clicca sul pulsante per la lista dei dipendenti. A questo punto, l'utente sceglie l'elemento da visualizzare nel dettaglio cliccando sul pulsante "Visualizza". Viene fatta la richiesta al server, e le informazioni del dipendente vengono visualizzate in una lista. Nella Figura 3.14 abbiamo il diagramma di attività corrispondente.

3.4 Mockup del sistema

3.4.1 Mockup delle app mobile

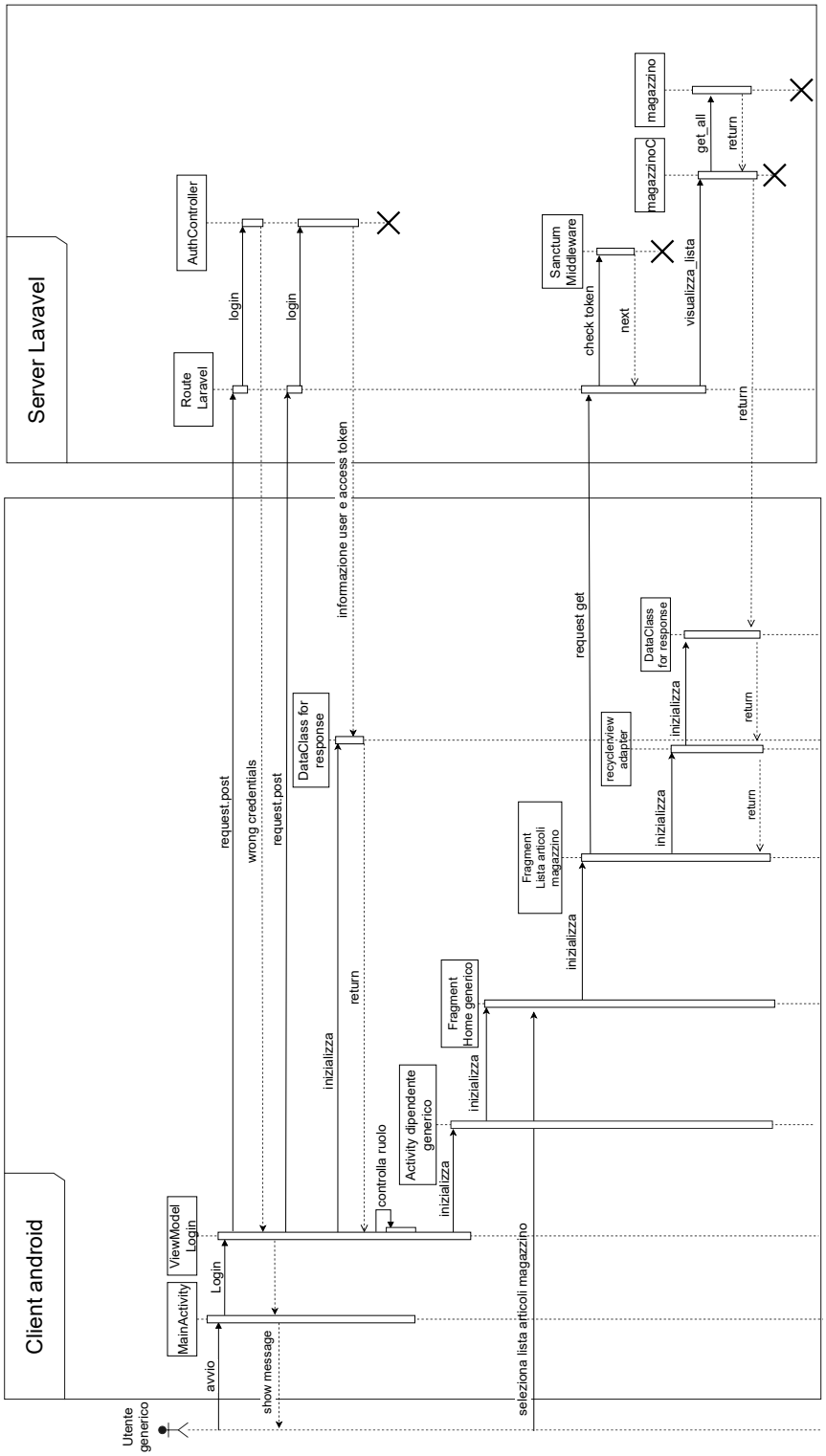

Figura 3.5. Diagramma delle sequenze per la visualizzazione, in una lista, degli oggetti del magazzino

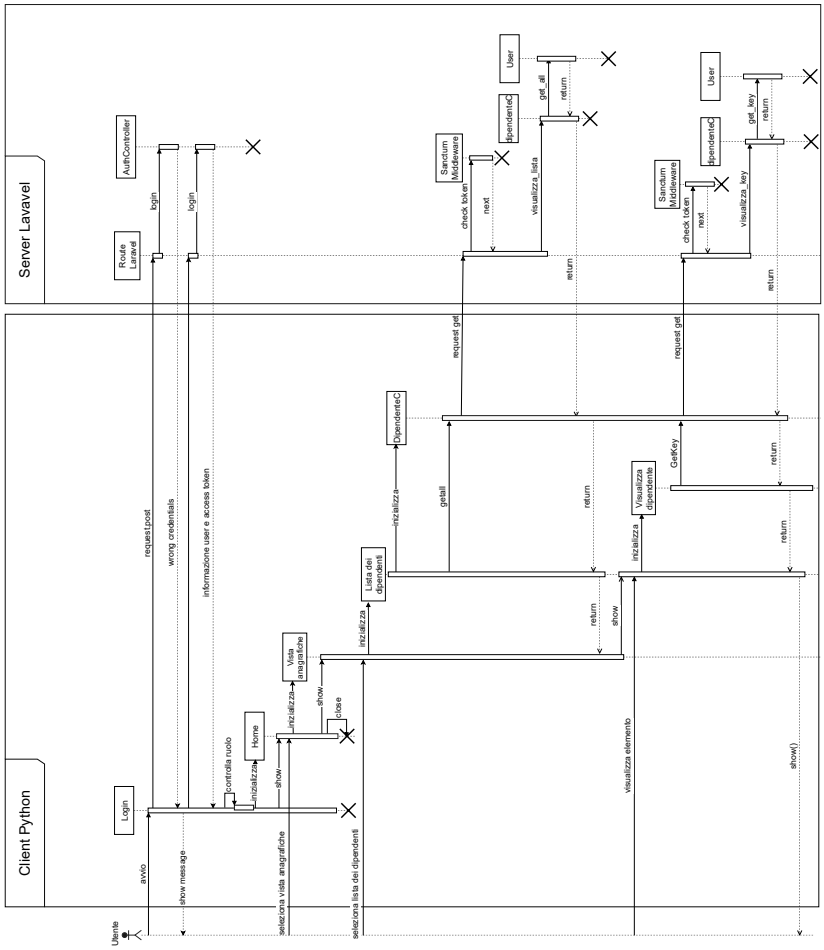

Figura 3.6. Diagramma delle sequenze per la visualizzazione nel dettaglio di un dipendente

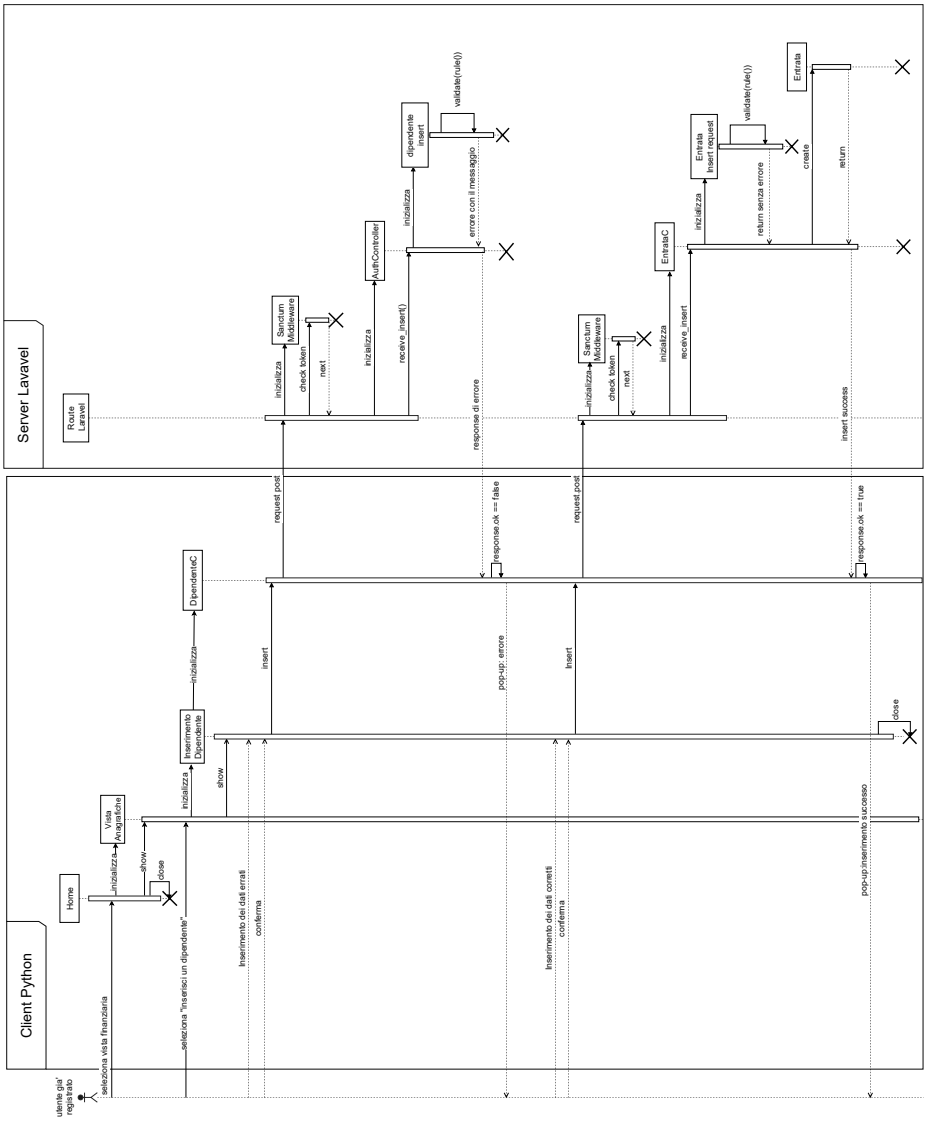

Figura 3.7. Diagramma delle sequenze per l'inserimento di un dipendente

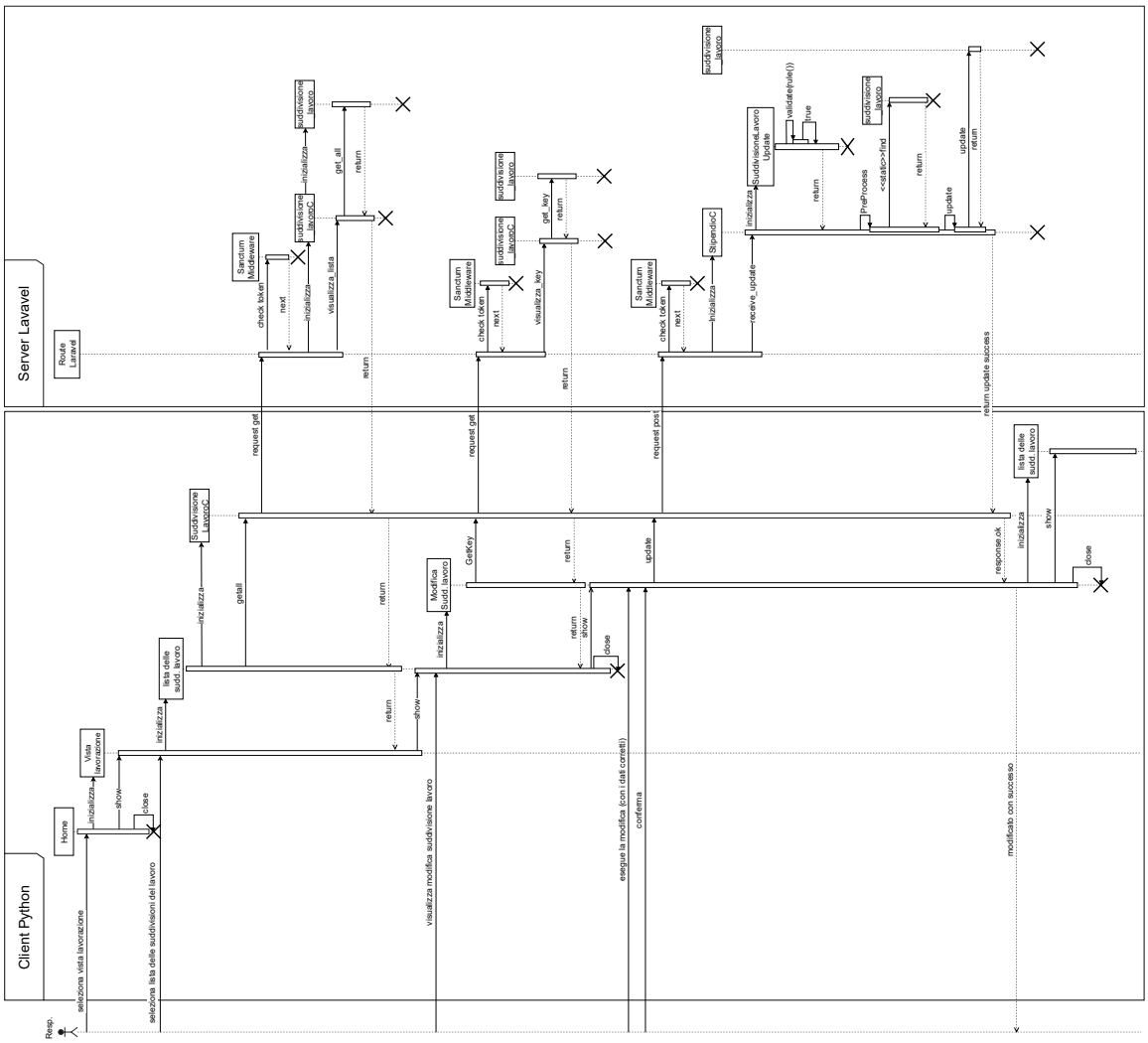

Figura 3.8. Diagramma delle sequenze per la modifica di una suddivisione lavoro

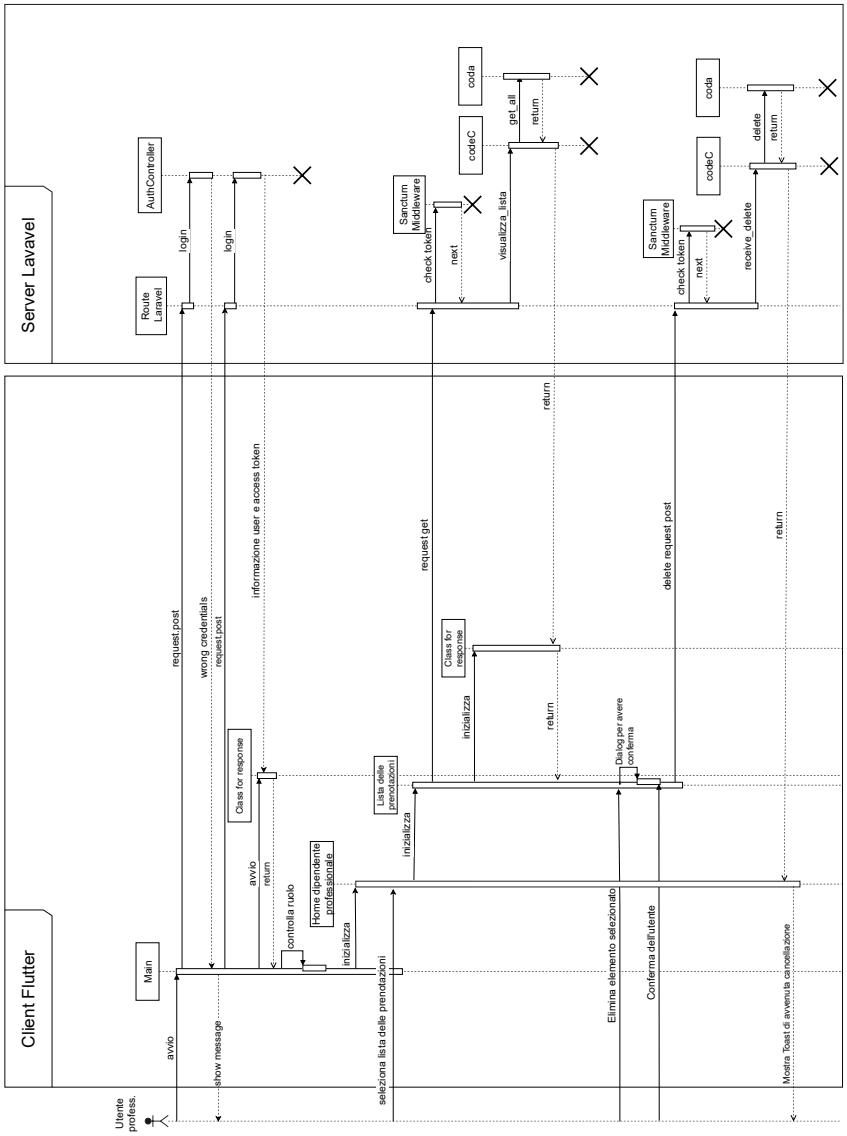

Figura 3.9. Diagramma delle sequenze per la cancellazione di una prenotazione ad un macchinario

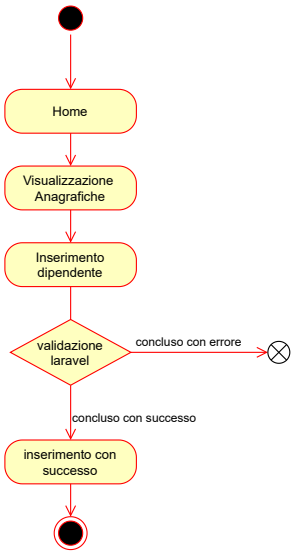

Figura 3.10. Diagramma delle attività per l'inserimento di un dipendente

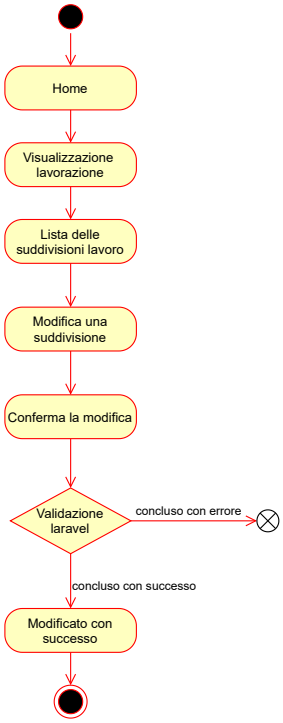

Figura 3.11. Diagramma delle attività per la modifica di una suddivisione lavoro

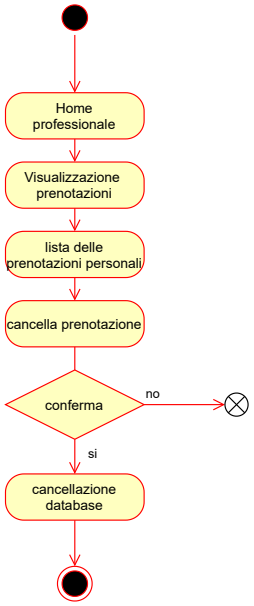

Figura 3.12. Diagramma delle attività per la cancellazione di una prenotazione ad un macchinario

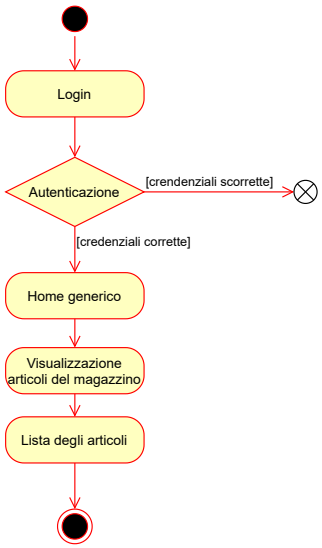

Figura 3.13. Diagramma delle attività per la visualizzazione in una lista degli articoli del magazzino

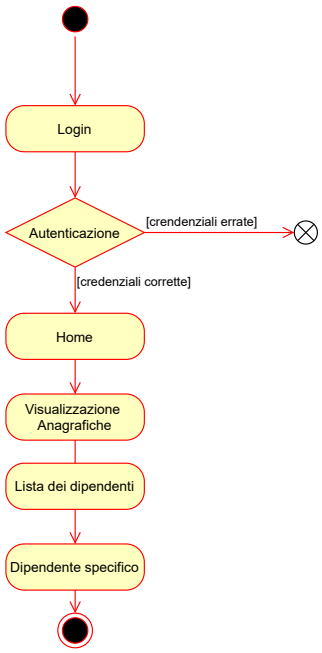

Figura 3.14. Diagramma delle attività per la visualizzazione dettagliata di un dipendente

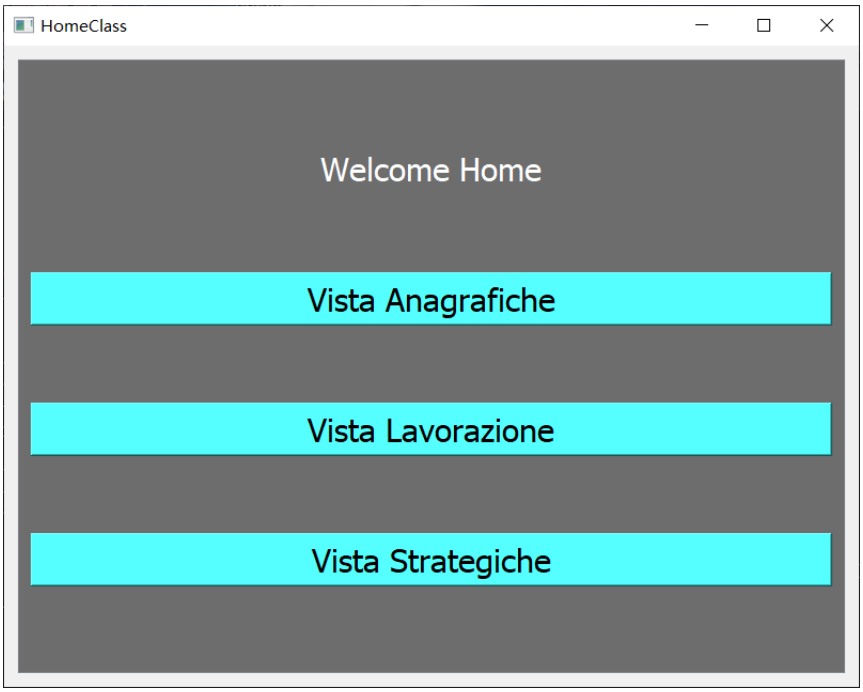

Figura 3.15. Mockup del sistema: Home

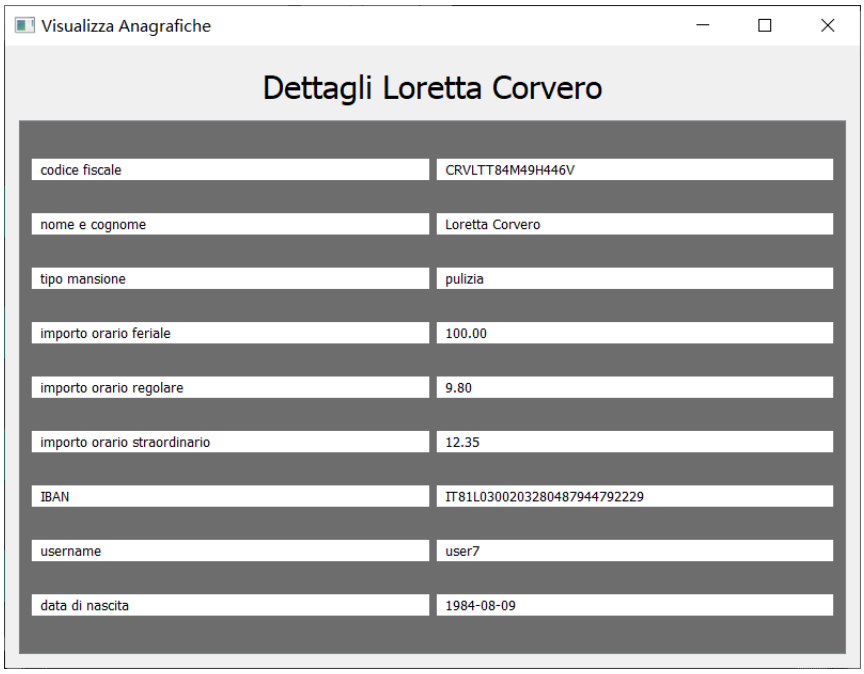

Figura 3.16. Mockup del sistema: dettaglio dipendente

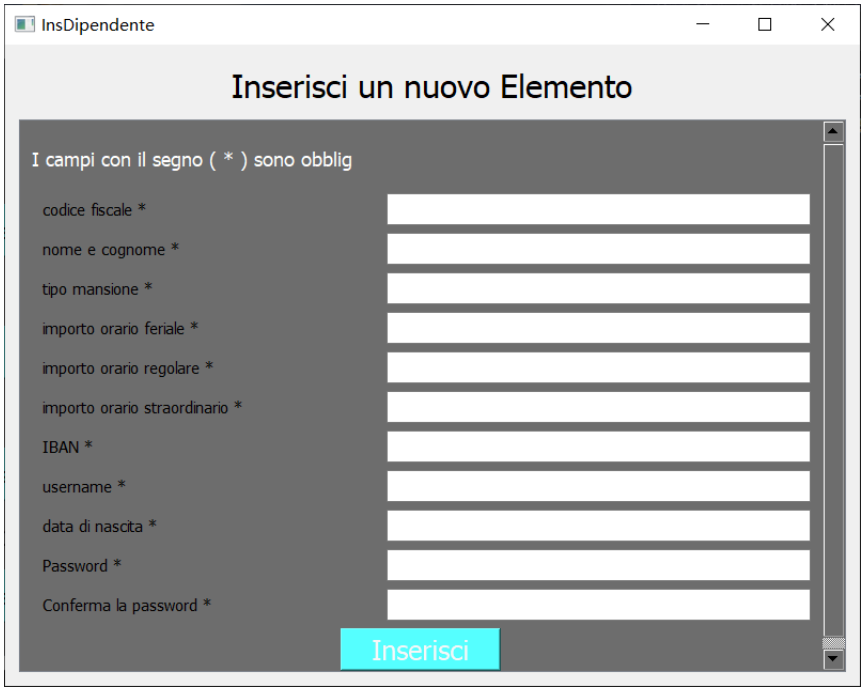

Figura 3.17. Mockup del sistema: inserimento dipendente

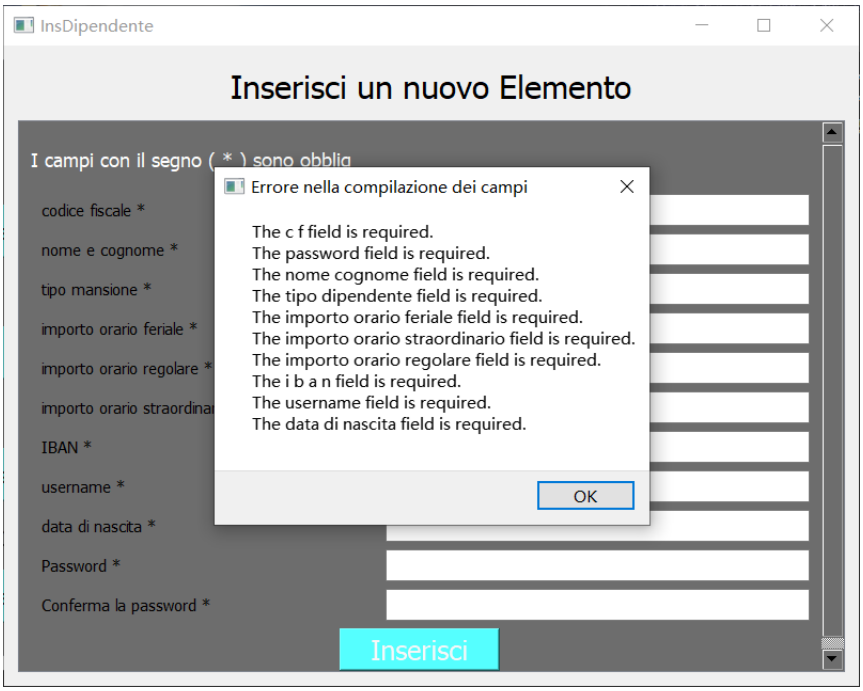

Figura 3.18. Mockup del sistema: inserimento dipendente con messaggio di errore

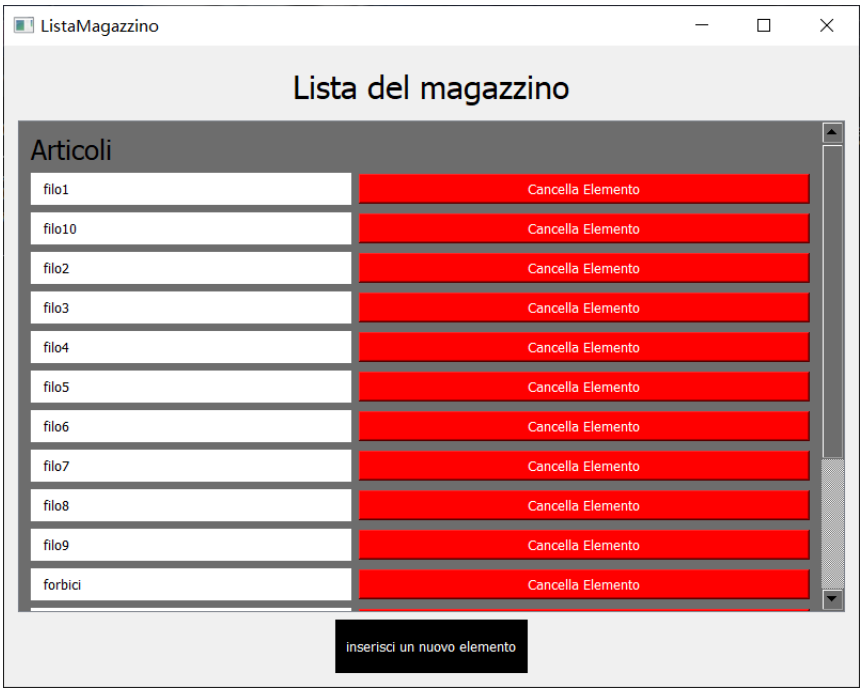

Figura 3.19. Mockup del sistema: magazzino

Figura 3.20. Mockup del sistema: Lista dei dipendenti

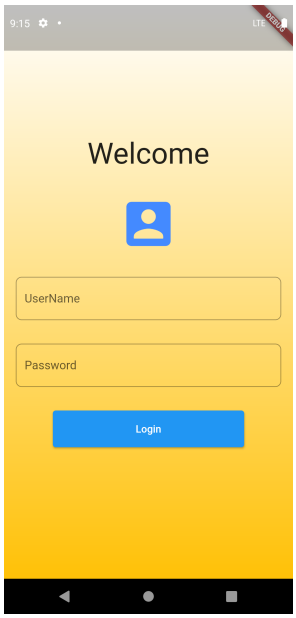

Figura 3.21. Mockup del sistema: Login su Flutter

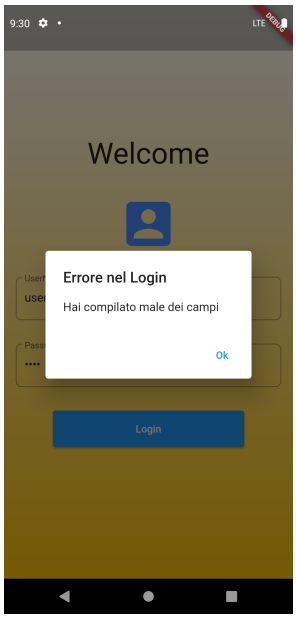

Figura 3.22. Mockup del sistema: Login su Flutter con errore

Figura 3.23. Mockup del sistema: Home con notifica su Flutter per un dipendente professionale

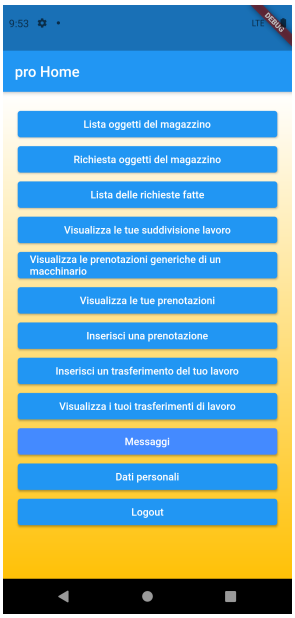

Figura 3.24. Mockup del sistema: Home senza notifica su Flutter per un dipendente professionale

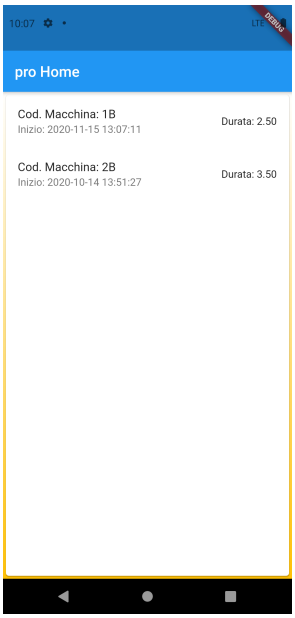

Figura 3.25. Mockup del sistema: Lista delle prenotazioni con Flutter

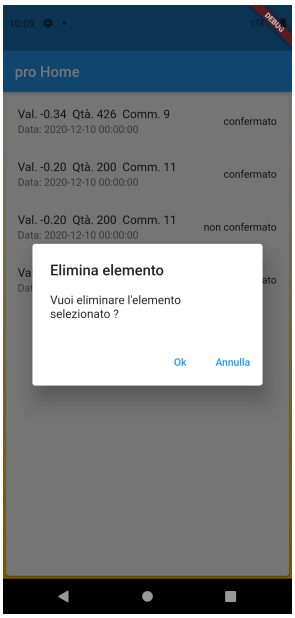

Figura 3.26. Mockup del sistema: Conferma eliminazione con Flutter

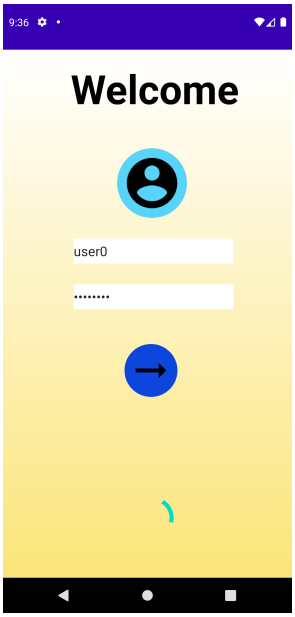

Figura 3.27. Mockup del sistema: Login su Kotlin

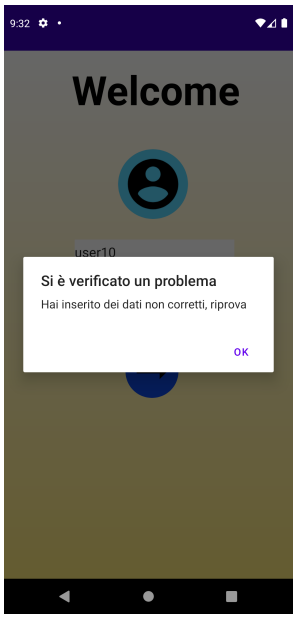

Figura 3.28. Mockup del sistema: Login su Kotlin con messaggio di errore

Figura 3.29. Mockup del sistema: Home del dipendente generico su Kotlin

Figura 3.30. Mockup del sistema: Home del dipendente professionale su Kotlin

Figura 3.31. Mockup del sistema: lista degli articoli del magazzino su Kotlin

4

Implementazione della componente relativa alle risorse interne

In questo capitolo mostreremo come le varie funzioni sono state implementate, riportando il codice e spiegandone il funzionamento.

4.1 Implementazione lato server

Abbiamo utilizzato Laravel, un framework open-source per lo sviluppo di applicazioni e servizi web basato su PHP, che utilizza una libreria di componenti codificate come classi PHP.

4.1.1 Model

Metodi generici

Nei Listati 4.1 e 4.2 vengono riportati tre casi diversi di uno stesso metodo. Nel primo caso viene restituito il risultato semplice di una ricerca all'interno di una determinata tabella, usando una chiave che viene passata come parametro. Nel secondo e nel terzo caso, invece, viene restituito il risultato di una ricerca più complessa, estesa anche ad altre tabelle, usando sempre la chiave che viene passata come parametro. Nel secondo caso, passando il codice fiscale di un dipendente, viene restituito un array contenente tutte le suddivisioni del lavoro che gli sono state assegnate, più il codice della merce della commessa a cui esse appartengono. Nel terzo caso, invece, viene restituita una suddivisione del lavoro più il codice della merce della commessa a cui appartiene, passando come parametri il codice fiscale del dipendente a cui è stata assegnata, più il numero della commessa di quella suddivisione.

```
public function get_key($key)
{
    return $this->with(['coda'])->find($key);
}
```

Listato 4.1. Questo è il metodo generico di un model, riportiamo `get_key()` nel caso semplice (macchine_pubbliche)

```
    public function get_key($key){
    return $this->with(['commessa' =>function($query){ $query->select('codice_merce');}])
    ->where('dipendente',$key)->get();
}

public function get_ckey($key){
    return $this->with(['commessa' =>function($query){ $query
    ->select('codice_merce','numero_commessa');}])
           ->where('dipendente',$key['dipendente'])
           ->where('commessa',$key['commessa'])->first();
}
```

Listato 4.2. Questo è il metodo generico di un model, riportiamo `get_key()` nel caso complesso (suddivisione_lavoro)

Nei Listati 4.3 e 4.4 vengono riportati due casi diversi di uno stesso metodo. Nel primo caso viene, semplicemente, restituito un array che contiene tutti gli elementi della tabella "macchine_pubbliche". Nel secondo caso, invece, abbiamo una situazione più complicata, analoga al caso precedente; quindi, viene restituito un array che contiene tutte le suddivisioni del lavoro, più tutte le informazioni del dipendente a cui esse sono state assegnate.

```
    public function get_all(){
    return $this->all();
}
```

Listato 4.3. Questo è il metodo generico di un model, riportiamo `get_all()` nel caso semplice (macchine_pubbliche)

```
    public function get_all(){
    return $this->with(['dipendente'])->get();
}
```

Listato 4.4. Questo è il metodo generico di un model, riportiamo `get_all()` nel caso complesso (suddivisione_lavoro)

coda(model semplice)

La classe riportata nel Listato 4.5 rappresenta una implementazione di un semplice model, che non contiene metodi particolari. Ci sono inizialmente le dipendenze per importare delle classi; successivamente abbiamo degli attributi che rappresentano la chiave primaria della tabella corrispondente, il nome della tabella a cui essa è associata, un array che contiene i nomi degli attributi della tabella che possono essere manipolati, una variabile `timestamps` per non salvare dati temporali. Successivamente abbiamo dei metodi che servono per esprimere dei vincoli con altre tabelle; sono quelli con `belongsTo` e significa che questo model ha una sua occorrenza in un altro model.

```
    <?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class coda extends Model
```

```
{
    use HasFactory;
    protected $table = 'coda';
    protected $fillable = [
        'codice_macchina',
        'inizio',
        'durata',
        'dipendente'
    ];
    protected $primaryKey = 'id';
    public $timestamps = false;

    public function dipendente(){
        return $this->belongsTo(User::class,'dipendente');
    }
    public function macchine_pubbliche(){
        return $this->belongsTo(macchine::class,'codice_macchina');
    }
    public function get_key($key){
        return $this->where('dipendente',$key)->get();
    }
    public function get_macchina($key){
        return $this->where('codice_macchina',$key)->get();
    }
    public function get_self($self){
        return $this->get_key($self);
    }

    public function get_all(){
        return $this->get();
    }

}
```

Listato 4.5. Questo è il caso di un model semplice (coda)

trasferimento_lavoro(model complesso)

La classe riportata nel Listato 4.6 rappresenta una implementazione di un model più complessa, cioè che contiene metodi particolari. Questo esempio ha molti metodi e attributi analoghi al caso precedente, quindi non saranno commentati. Abbiamo un attributo `incrementing` che ci serve per specificare se la chiave va incrementata di uno dopo aver inserito un elemento. Inoltre, abbiamo un elemento `get_key` che ci restituisce l'elemento trasferimento lavoro, più il codice merce appartenente alla commessa corrispondente, utilizzando, per la ricerca, la chiave passata come argomento. Infine, abbiamo `get_self` che ci restituisce l'elemento trasferimento lavoro, più il codice fiscale, il nome e il cognome appartenenti al dipendente corrispondente, utilizzando, per la ricerca, la chiave passata come argomento.

```
    <?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class trasferimento_lavoro extends Model
{
    use HasFactory;
    protected $table = 'trasferimento_lavoro';
    protected $fillable =[
        'codice_trasf',
        'commessa',
        'dipendente',
        'data_trasferimento',
        'valore_trasferito',
        'quantita_trasferita',
    ];

    protected $primaryKey = 'id';
    public $incrementing = false;
    public $timestamps = false;

    public function dipendente()
    {
        return $this->belongsTo(User::class,'dipendente','CF');
```

```
    }

    public function commessa()
    {
        return $this->belongsTo(commessa::class,'commessa','codice_merce');
    }
    public function get_all(){
        return $this->with(['dipendente'])->get();
    }

    public function get_key($key){
        return $this->with(['commessa' =>function($query){ $query
        ->select('codice_merce');}])->find($key);
    }

    public function get_self($key){
        return $this->with(['dipendente' =>function($query){ $query
        ->select('CF','nome_cognome');}])->where('dipendente',$key)->get();
    }
}
```

Listato 4.6. Questo è il caso di un model complesso (trasferimento_lavoro)

4.1.2 Controller

Metodi generici

Nell'esempio riportato nel Listato 4.7 vengono mostrati i metodi del controller per visualizzare i dati presenti nelle tabelle del database; a tal fine viene utilizzata un'istanza del model e i suoi metodi, descritti in precedenza.

```
    public function visualizza_lista(){
        $response = $this->_model->get_all();
        return response($response,200)->header('Content-Type', 'application/json');
    }

    public function visualizza_key($chiave){
        $response = $this->_model->get_Key($chiave);
        return response($response,200)->header('Content-Type', 'application/json');
    }

    public function visualizza_self(Request $prova){
       $response = $this->_model->get_self($prova->user()->CF);
       return response($response,200)->header('Content-Type', 'application/json');
    }
```

Listato 4.7. Questi sono i metodi generici del controller per le `visualizzazione`

Nell'esempio riportato nel Listato 4.8 viene mostrato il metodo utilizzato dal controller generico per effettuare un inserimento nel database. Si usano i metodi messi a disposizione da eloquent.

```
    public function insert(array $data,array $message = null,bool $preProcess = true,
     bool $postProcess = true)
    {
        $data_array = $data;
        if ($preProcess) {$this->preProcess($data_array);}
        $save = isset($save)? $save:true;
        $model = $this->_model;
        $message = isset($message)?$message:['messagge'=>'insert success'];
        $model->fill($data_array)->save();
        if ($postProcess) {$this->postProcess($data);}
        return response($message,201)->header('Content-Type', 'application/json');
    }
```

Listato 4.8. Questo è il metodo generico del controller per l'inserimento `insert`

Nel Listato 4.9 viene mostrato il metodo utilizzato dal controller generico per effettuare la modifica di un elemento nel database. Questo metodo inizialmente

identifica la chiave primaria e il suo tipo, per controllare se essa è composta da uno o più attributi. Infine, si usano i metodi messi a disposizione da eloquent.

```
public function update(array $data,array $message = null,bool $preProcess = true,
 bool $postProcess = true){
    $data_array = $data;
    if ($preProcess) {$this->preProcess($data_array);}
    $model = $this->_model;
    $message = isset($message)?$message:['messagge'=>'update success'];
    $primary_keys = $model->getkeyname();
    $primary_arrays = array();
    if (is_array($primary_keys)){
        foreach($primary_keys as $key){
            $primary_arrays[$key] = $data_array[$key];
            unset($data_array[$key]);
        }
    }
    else{
        $primary_arrays[$primary_keys] = $data_array[$primary_keys];
        unset($data_array[$primary_keys]);
    }
    $model = $model->where($primary_arrays)->update($data_array);
    if ($postProcess) {$this->postProcess($data);}
    return response($message,201)->header('Content-Type', 'application/json');
}
```

Listato 4.9. Questo è il metodo generico del controller per la modifica `update`

Nel Listato 4.10 viene mostrato il metodo utilizzato dal controller generico per effettuare la cancellazione di un elemento nel database. Questo metodo inizialmente identifica la chiave primaria, come nel caso precedente; infine si usano i metodi messi a disposizione da eloquent.

```
public function delete(array $data,array $message = null,bool $preProcess = true,
 bool $postProcess = true){
    $data_array = $data;
    if ($preProcess){$this->preProcess($data_array);}
    $model = $this->_model;
    $message = isset($message)?$message:['messagge'=>'delete success'];
    $primary_keys = $model->getkeyname();
    $primary_arrays = array();
    if (is_array($primary_keys)){
        foreach($primary_keys as $key){
            $primary_arrays[$key] = $data_array[$key];
            unset($data_array[$key]);
        }
    }
    else{
        $primary_arrays[$primary_keys] = $data_array[$primary_keys];
        unset($data_array[$primary_keys]);
    }
    $model = $model->where($primary_arrays)->delete($data_array);
    if ($postProcess) {$this->postProcess($data);}
    return response($message,201)->header('Content-Type', 'application/json');
}
```

Listato 4.10. Questo è il metodo generico del controller per la cancellazione `delete`

Request di validazione

Nel Listato 4.11 vengono rappresentati i vincoli che deve soddisfare una richiesta, ad esempio di modifica, per poter essere accettata dal server; i vincoli devono essere soddisfatti dai dati inseriti dall'utente. Con il vincolo `required` intendiamo che quell'informazione deve essere necessariamente inserita o passata lato client; successivamente, ci sono vincoli sul tipo di dato, come ad esempio `Int` e poi abbiamo `exists`; questo vincolo indica che il codice fiscale inserito nella richiesta deve esistere nella tabella dipendente, nella colonna dei codici fiscali.

```php
<?php

namespace App\Http\Requests\update;

use Illuminate\Foundation\Http\FormRequest;

class aggiornamento_update extends FormRequest
{
    public function rules()
    {
        return [
            'dipendente' => ['required','String','exists:dipendente,CF'],
            'data' => ['required','date'],
            'quantita' => ['required','Int','min: 0'],
            'magazzino' => ['required','String','exists:magazzino,tipo_scorta']
        ];
    }
}
```

Listato 4.11. Questo è un esempio della classe `Request` utilizzata per la validazione della richiesta di modifica di un aggiornamento

Aggiornamento (controller semplice)

Nel Listato 4.12 viene mostrata l'implementazione di un controller semplice; per ogni richiesta abbiamo una classe request che contiene i vincoli necessari; sono utilizzati i metodi del controller generale.

```php
    <?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\aggiornamento;

use App\Http\Requests\insert\aggiornamento_insert;
use App\Http\Requests\update\aggiornamento_update;
use App\Http\Requests\delete\aggiornamento_delete;

class aggiornamentoC extends Controller
{
    protected $_model;

    public function __construct()
    {
        $this->_model = new aggiornamento;
    }

    public function receive_insert(aggiornamento_insert $request){
        return $this->insert($request->all());
    }
    public function receive_update(aggiornamento_update $request)
    {
        return $this->update($request->all());
    }

    public function receive_delete(aggiornamento_delete $request)
    {
        return $this->delete($request->all());
    }
}
```

Listato 4.12. Questo è il controller specifico dell'entità aggiornamento, chiamato `aggiornamentoC`

Code (controller complesso)

Il Listato 4.13 rappresenta un esempio di controller complesso, dove viene usato il metodo `preProcess_custom`. Come si evince dal listato, prima della cancellazione di una prenotazione ad un macchinario, vengono prese tutte le informazioni di quella prenotazione, e vengono passate al metodo `preProcess_custom`, il quale le usa

per creare un messaggio; successivamente vengono presi tutti i codici fiscali dei dipendenti professionali e, per ciascuno di essi, viene inserito questo messaggio; alla fine viene completata l'operazione di cancellazione. I messaggi vengono usati nelle applicazioni mobile e servono per tenere aggiornati i dipendenti. Gli altri metodi sono analoghi ai casi precedenti.

```
    <?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Requests\insert\coda_insert;
use App\Http\Requests\delete\coda_delete;

use App\Models\coda;
use App\Models\User;
use App\Models\messaggi;
use App\Http\Requests\coda_macchina_request;

class codeC extends Controller
{
    protected $_model;
    public function __construct()
    {
        $this->_model = new coda;
    }

    public function preProcess_custom($data){
        $messaggio = new messaggi;

        $testo = "E' stata eliminata la prenotazione della macchina ".$data->codice_macchina."
         prevista per le ore ".$data->inizio." della durata ".$data->durata;
        $prof = User::where('tipo_dipendente',"pro")->get();
        foreach($prof as $i){
            $messaggio::create(['messaggio' => $testo,'dipendente' => $i->CF]);
        }

    }
    public function get_all(){
        return $this->all();
    }

    public function receive_insert(coda_insert $request){
        return $this->insert($request->all());
    }

    public function receive_delete(coda_delete $request)
    {
        $codaDelete = coda::where('id',$request->get('id'))->first();
        $this->preProcess_custom($codaDelete);
        return $this->delete($request->all());
    }
    public function receive_macchina(Request $request){
       return $this->visualizza_simo($request);
    }
}
```

Listato 4.13. Questo è il controller specifico dell'entità code, chiamato `codeC`

4.1.3 Le rotte del Server

Nel Listato 4.14 vengono mostrate alcune delle rotte implementate nel server utilizzando Laravel.

```
Route::post('/login','AuthController@login');

   Route::middleware('auth:sanctum')->prefix('/authed')->group(function(){

      Route::prefix('')->group(function(){ //cuoco

          Route::prefix('/dipendente')->group(function(){
              Route::get('/anagrafica','dipendenteC@visualizza_self');
          });

          Route::prefix('/aggiornamento')->group(function(){...});
          Route::prefix('/magazzino')->group(function(){...});
```

```
            Route::prefix('/gen')->group(function(){ //utente generico

                Route::prefix('/suddivisione_lavoro')->group(function(){...});

                Route::prefix('/pro')->group(function(){ //utente professionale

                    Route::prefix('/messaggi')->group(function(){...});
                    Route::prefix('/macchine_pubbliche')->group(function(){...});
                    Route::prefix('/code')->group(function(){...});
                    Route::prefix('/suddivisione_lavoro')->group(function(){...});
                    Route::prefix('/trasferimento_lavoro')->group(function(){...});

                    Route::prefix('/resp')->group(function(){ //responsabile

                                            Route::prefix('/trasferimento_lavoro')->group(function(){

                                    Route::get('/all','trasferimento_lavoroC@visualizza_lista');
                                    Route::get('/key/{chiave}','trasferimento_lavoroC@visualizza_key');
                                    Route::match(['post','put'],'/confirm',
                                    'trasferimento_lavoroC@confirm_transaction');
                                    Route::match(['post','put'],'/update',
                                    'trasferimento_lavoroC@receive_update');
                                    Route::match(['post','delete'],'/delete',
                                    'trasferimento_lavoroC@receive_delete');
            });

                    });
                });
            });
        });
    });
```

Listato 4.14. Questa è la struttura delle rotte del server, usate dai client; per motivi di spazio ne saranno riportate solo alcune

4.2 Implementazione lato client con Python

Uno dei client è stato sviluppato utilizzando Python e PyQT5; la seconda componente è stata utilizzata per costruire tutte le finestre e gli elementi grafici.

4.2.1 Controller

Nel Listato 4.15 mostriamo un controller lato client; in questo caso, abbiamo implementato tutti i metodi necessari per fare le richieste al server; le rotte sono costruite concatenando delle stringhe. Il server restituisce le informazioni usando il formato JSON che viene convertito in Python come un dizionario, o una lista di dizionari, oppure come altre strutture dati più complesse.

```
    import requests
import env

class DipendenteC:

    def __init__(self):
        self.credenziali = {"Authorization": "Bearer "+env.token,"Accept": env.impostazione}
        self.url = "gen/pro/resp/dipendente/"

    def GetAll(self):
        response = requests.get(env.host + env.Url+self.url,headers=self.credenziali)
        if response.ok:
            print("va bene")
            return response.json()
        else:
            return response.json()

    def GetSelf(self):
        response = requests.get(env.host + env.Url+self.url+'self/',headers=self.credenziali)
        if response.ok:
            print("va bene")
            return response.json()
        else:
            raise Exception("Errore. ",response.status_code)
```

```
    def GetKey(self,key):
        response = requests.get(env.host + env.Url+self.url+'key/'+key,headers=self.credenziali)
        if response.ok:
            print("va bene")
            return response.json()
        else:
            raise Exception("Errore. ",response.status_code)

    def Insert(self, body):
        response = requests.post(
            env.host + env.Url+self.url+'insert', data=body, headers=self.credenziali)
        if response.ok:
            print("va bene")
            return response.json()
        else:
            return response.json()

    def Delete(self,body):
        response = requests.post(env.host + env.Url+self.url+'delete',
        data=body,headers=self.credenziali)
        if response.ok:
            print("va bene")
            return response.json()
        else:
            return response.json()

    def Update(self,body):
        response = requests.post(env.host + env.Url+self.url+'update',
        data=body,headers=self.credenziali)
        if response.ok:
            print("va bene")
            return response.json()
        else:
            return response.json()
```

Listato 4.15. Controller lato client (`dipendenteC`)

4.2.2 View

View di una lista (Lista degli articoli del magazzino)

Nel Listato 4.16 viene mostrata una lista lato client; per motivi di spazio non riportiamo le varie componenti di PyQt5. Ogni lista ha, in generale, per ogni elemento, tre pulsanti, uno per la visualizzazione dettagliata, uno per la modifica e uno per la cancellazione. In alcuni casi, però, ci sono delle liste che hanno solo uno o due pulsanti su tre. Inoltre, nella classe, riportiamo il messaggio di conferma per la cancellazione. Dopo quest'ultima attività, nella lista viene effettuato un aggiornamento del contenuto.

```
    from Magazzino.View.InserimentoMagazzino import Ui_InserisciMagazzino
from Magazzino.Controller.MagazzinoC import MagazzinoC
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QMessageBox, QSizePolicy, QWidget


class Ui_ListaMagazzino(QWidget):
    def __init__(self, parent=None,fakeparent=None):
        super(Ui_ListaMagazzino, self).__init__(parent)

        ...

        self.controller = MagazzinoC()
        self.chiamata = self.controller.GetAll()

        ...

        self.traduzione = {
        'tipo_scorta':"Articoli",
        }

        self.listAttr = ['tipo_scorta']

        ...

        for attr in range(len(self.listAttr)):
```

```
            #Aggiunge l'intestazione della tabella
            self.AddTableHeader(self.traduzione[self.listAttr[attr]],attr,sizes[attr])

            #aggiunge l'elemento nella tabella
            for a in range(0,len(self.chiamata)):
                self.AddTableContent(attr,a+1,self.chiamata[a][self.listAttr[attr]],sizes[attr])

        #Aggiunge i pulsanti delle operazioni
        self.OperationButtons=[
            {
                "name":"Cancella Elemento",
                "StyleSheet":"background-color: rgb(255, 0, 0);\n color: rgb(255, 255, 255);",
                "function":self.deleteConfirm
            },
        ]

        X_offset = len(self.listAttr)
        for i in range(0,len(self.OperationButtons)):
            for a in range(0,len(self.chiamata)):
                #print(i)
                self.OperationButtons[i]
                self.AddOperationButton(X_offset+i,a,self.OperationButtons[i],
                lambda state,b=a,c=i: self.OperationButtons[c]["function"](self.chiamata[b]))

        ...

    #chiamata alla funzione per la cancellazione di un elemento
    def deleteConfirm(self,articolo):

        msg = QMessageBox()
        msg.setWindowTitle('Conferma')
        msg.setText('sei sicuro di voler cancellare l\' articolo '+articolo['tipo_scorta'] + '?')
        msg.setStandardButtons(QMessageBox.Yes | QMessageBox.No)
        okButton = msg.button(QMessageBox.Yes)
        noButton = msg.button(QMessageBox.No)
        okButton.setText('si')
        retval = msg.exec_()
        if(msg.clickedButton() == okButton):
            print('cancellazione confermata')
            postbody = {'tipo_scorta':articolo['tipo_scorta']}
            res = self.controller.Delete(postbody)
            print(res)
            self.RefreshLista = Ui_ListaMagazzino(fakeparent=self.fakeparent)
            self.RefreshLista.show()
            self.handClose = 0
            self.close()
        else:
            print('cancellazione annullata')

       ...

    #Qui viene aggiunta una funzione ai bottoni, da attivare quando uno specifico
    # bottone viene premuto
    def AddOperationButton(self,x,y,buttonDict,function):
        OperationButton = QtWidgets.QPushButton(self.scrollAreaWidgetContents)
        OperationButton.setStyleSheet(buttonDict["StyleSheet"])
        OperationButton.setObjectName("OperationButton")
        OperationButton.setMinimumSize(QtCore.QSize(100, 25))
        OperationButton.setText(self._translate("ListaMagazzino", buttonDict["name"]))
        OperationButton.clicked.connect(function)
        self.gridLayout.addWidget(OperationButton, y+1, x, 1, 1)
        return OperationButton

      ...

    def GoInserimentoMagazzino(self):
        self.InserimentoMagazzino = Ui_InserisciMagazzino(fakeparent = self)
        self.InserimentoMagazzino.show()
        self.handClose = 0
        self.close()

    def closeEvent(self, event):
        if(self.handClose ==0):
            self.handClose = 1
        else:
            self.fakeparent.show()
        event.accept()
```

Listato 4.16. Vista lato client (MagazzinoV)

View di un inserimento e modifica (inserimento di un dipendente)

Nel Listato 4.17 viene mostrato la vista per l'inserimento lato client; per motivi di spazio non riportiamo le varie componenti di PyQt5. In questa vista abbiamo sempre una variabile `body` che rappresenta l'insieme dei valori inseriti dall'utente

e che definiscono il nuovo elemento; abbiamo, inoltre, una variabile `traduzione`, che contiene dei nomi più precisi rispetto ai nomi degli attributi, in quanto, in alcuni casi, abbiamo usato abbreviazioni e parole attaccate tra di loro. Il metodo di inserimento utilizza i dati inseriti dall'utente per creare il `body`; successivamente viene fatta la richiesta di tipo POST al server, usando il metodo del controller. Infine, si attende la risposta del server per aggiornare l'utente sull'esito dell'operazione. Nel caso della modifica, abbiamo una situazione analoga.

```
from Dipendente.Controller.DipendenteC import DipendenteC
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QMessageBox, QWidget
import copy

class Ui_InserisciDip(QWidget):
    def __init__(self, parent=None):
        super(Ui_InserisciDip, self).__init__(parent)
        self._translate = QtCore.QCoreApplication.translate
        self.controller = DipendenteC()

        ...

        self.traduzione = {'CF':'codice fiscale *',
        'nome_cognome':'nome e cognome *',
        'tipo_dipendente':'tipo mansione *',
        'importo_orario_feriale':'importo orario feriale *',
        'importo_orario_regolare':'importo orario regolare *',
        'importo_orario_straordinario':'importo orario straordinario *',
        'IBAN':'IBAN *',
        'username':'username *',
        'data_di_nascita':'data di nascita *',
        'password': 'Password *',
        'password_confirmation': 'Conferma la password *'
        }

        self.body = {
        'CF': '',
        'nome_cognome': '',
        'tipo_dipendente':'',
        'importo_orario_feriale':'',
        'importo_orario_regolare':'',
        'importo_orario_straordinario':'',
        'IBAN':'',
        'username':'',
        'data_di_nascita': '',
        'password': '',
        'password_confirmation': ''
        }

        ...

        for a in self.traduzione:
            self.AddLabelCampo(self.traduzione[a])
            self.listaInput[a] = self.AddEdit()

        ...

    #Aggiunge un pulsante con dato testo e funzione
    def AddButton(self,text,function):
        self.pushButton = QtWidgets.QPushButton(self.scrollAreaWidgetContents)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum,
        QtWidgets.QSizePolicy.Fixed)

        ...

    #Funzione dell'inserimento
    def Insert(self):
        for a in self.listaInput:
            self.body[a] = self.listaInput[a].toPlainText().replace(' ', '')
        self.risultato = self.controller.Insert(self.body)
        self.messaggio = ""
        if('message' in self.risultato):
            if('errors' in self.risultato):
                for a in self.risultato['errors']:
                    self.messaggio += self.risultato['errors'][a][0]+'\n'
            else:
                QMessageBox.about(self, "Errore del server ",str(self.risultato['message']))
            QMessageBox.about(self, "Errore nella compilazione dei campi",self.messaggio)
        else:
            QMessageBox.about(self, "Esito operazione","Operazione completata con successo")
            self.close()

    ...
```

Listato 4.17. View di inserimento

View di un dettaglio (dettaglio di un dipendente)

Nel Listato 4.18 mostriamo una vista che rappresenta la visualizzazione dettagliata di un elemento; in questo caso l'utente non può effettuare operazioni sull'elemento. Questa classe contiene solo la variabile traduzione; ci sono delle visualizzazioni dettagliate più complesse, dove vengono riportate tutte le informazioni dell'elemento selezionato, ma anche altre informazioni, di altri model, ad esso collegati. Quindi, in questo caso, abbiamo più variabili traduzione, il cui scopo è analogo al caso precedente (serve per avere una traduzione, più precisa, degli attributi di una tabella del database).

```
from typing import KeysView
from Dipendente.Controller.DipendenteC import DipendenteC
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QMessageBox, QWidget
import copy

class Ui_VisualizzaAnagDip(QWidget):
    def __init__(self,CF, parent=None):
        super(Ui_VisualizzaAnagDip, self).__init__(parent)
        self.cf = CF
        VisualizzaAnagDip = self
        self.controller = DipendenteC()

        ...

        self.traduzione = {'CF':'codice fiscale',
        'nome_cognome':'nome e cognome',
        'tipo_dipendente':'tipo mansione',
        'importo_orario_feriale':'importo orario feriale',
        'importo_orario_regolare':'importo orario regolare',
        'importo_orario_straordinario':'importo orario straordinario',
        'IBAN':'IBAN',
        'username':'username',
        'data_di_nascita':'data di nascita'}

        #esclude gli elementi non desiderati dalla visualizzazione
        self.viewList = copy.deepcopy(self.chiamata)
        self.exclude = ['remember_token']
        for elem in self.exclude:
            self.viewList.pop(elem)


        self.count = 0

        ...

        for a in self.viewList:
            #Aggiunge un label descrittivo
            self.AddElement(self.gridLayout,self.traduzione[a],0,self.count)
            #Aggiunge un label contenuto
            self.AddElement(self.gridLayout,self.chiamata[a],1,self.count)

            self.count += 1

        ...
```

Listato 4.18. Una view per visualizzare le informazioni dettagliate di un `Dipendente`

4.3 Implementazione lato client con Flutter

Un altro client è stato sviluppato con Flutter, un framework che utilizza il linguaggio Dart.

4.3.1 Lista delle proprie prenotazioni

Nel Listato 4.19 possiamo vedere come è stata realizzata una schermata in Flutter; questa classe restituisce un widget con lo stato; quindi abbiamo due classi e la secon-

da rappresenta lo stato. Nella seconda classe abbiamo, come prima cosa, il recupero dei dati (credenziali) passate dalla schermata precedente; nel return della funzione `build` descriviamo il nostro widget; quindi partiamo con l'appbar e proseguiamo con il body; in esso abbiamo un widget "Container" all'interno del quale definiamo delle decorazioni. Successivamente, abbiamo la dichiarazione della `listView`; all'interno della `listView` viene definito il testo da visualizzare fino a quando l'app non riceve una risposta dal server. Infine, abbiamo l'esecuzione del metodo `getUserData` a cui passiamo il TOKEN per l'autenticazione lato server. In questo metodo abbiamo l'effettiva chiamata al server, dei log per il debug, e, infine, la chiamata al metodo `parsing` sfruttando gli Isolate (per chiamare il metodo usiamo `compute`); così il metodo viene eseguito su un background thread.

Al metodo `parsing` passiamo la risposta del server; questo metodo ci server per fare il parsing della risposta, così ci restituisce una lista di oggetti (sono istanze della classe `ListaPrenotazioniSelfResponse`) che contengono i dati del database. Inoltre, all'interno della lista, è definita un'azione per l'evento `onLongPress`; se si verifica questo evento su un elemento della lista viene viene visualizzato un Dialog per avvisare l'utente sulla possibilità di cancellare l'elemento selezionato; se l'utente non conferma l'operazione continuerà a visualizzare la lista; altrimenti, in caso di conferma, viene eseguita la funzione `DeleteFunction`.

Questa funzione serve per fare la richiesta di cancellazione dell'elemento al server; vengono passati come parametri il TOKEN e l'id dell'elemento selezionato; se la richiesta ha esito positivo, viene ritornato il valore booleano `true`, altrimenti, in caso di errore, `false`. In base al valore ritornato dal metodo, l'app avvisa l'utente sull'esito della cancellazione.

Entrambi i metodi `getUserData` e `DeleteFunction` sono di tipo `Future`, e utilizzano la keyword `async`, che, usata subito dopo la dichiarazione della funzione serve per specificare che essa ritornerà un `Future`. Abbiamo, inoltre, la keyword `await`; usata subito prima della chiamata di una funzione, serve a metterne l'esecuzione in pausa, in attesa che il risultato sia disponibile. L'esecuzione riprenderà da quel punto.

```
import 'dart:convert';

...

import 'ListaPrenotazioniSelfResponse.dart';
/*
Questa classe implementa la schermata che contiene la lista delle proprie prenotazioni
 */
class ListaPrenotazioniSelf extends StatefulWidget {
  const ListaPrenotazioniSelf({Key? key}) : super(key: key);
  @override
  _ListaPrenotazioniSelfState createState() => _ListaPrenotazioniSelfState();
}

class _ListaPrenotazioniSelfState extends State<ListaPrenotazioniSelf> {
//Questa e' la funzione che serve per eliminare un elemento
//selezionato precedentemente dall'utente
  Future<bool> DeleteFunction(String id,String token) async {
    log("va bene 4");
    var response = await http.post(
        Uri.http("10.0.2.2", "progettolaurea/public/api/authed/gen/pro/code/delete"),
        headers: <String, String>{
          'Accept': "application/json",
          'Authorization' : "Bearer "+token,
        },
        body: <String, String>{
          'id': id,
        }
    );
    if(response.statusCode == 201){
      log('va bene 5');
```

```
      return true;
    }
    else{
      log(response.statusCode.toString());
      return false;
    }
  }
//Questa e' la funzione che effettua la chiamata al server, restituisce una lista di oggetti definiti precedentemente
  Future<List<ListaPrenotazioniSelfResponse>> getUserData(String chiave) async {
    var response = await http.get(Uri.http("10.0.2.2",
    "progettolaurea/public/api/authed/gen/pro/code/self"),
      headers: <String, String>{
        'Accept': "application/json",
        'Authorization' : "Bearer "+chiave,
      },
    );
    if(response.statusCode == 200)
      log("va bene 1");
    else
      log(response.body);
    return compute(parsing,response.body);
  }
//Questa e' la funzione che effettua il parsing della risposta, viene richiamata
//dalla funzione precedente e lanciata su un background thread
  static List<ListaPrenotazioniSelfResponse> parsing(String responseBody){
    log("va bene 2");
    var jsonData = jsonDecode(responseBody);
    List<ListaPrenotazioniSelfResponse> dati = [];

    for(var u in jsonData){
      log("va bene 3");
      ListaPrenotazioniSelfResponse dato = ListaPrenotazioniSelfResponse.fromJson(u);
      dati.add(dato);
    }
    return dati;
  }

  @override
  Widget build(BuildContext context) {
//Recupero le credenziali che vengono passate dalla schermata precedente
    final token = ModalRoute.of(context)!.settings.arguments as ResponseLogin;

// Qui vengono definiti tutti i widget che compongono
//la schermata e che vengono visualizzati dall'utente
    return Scaffold(
        appBar: AppBar(
          title: Text(token.utente.tipo_dipendente+' Home'),
          automaticallyImplyLeading: false,
        ),
        body: Container(
          decoration: BoxDecoration(
              gradient: LinearGradient(
                  begin: Alignment.topRight,
                  end: Alignment.bottomRight,
                  colors: [Colors.white,Colors.amber]
              )
          ),
          child: Card(child: FutureBuilder<List<ListaPrenotazioniSelfResponse>>(
              future: getUserData(token.token),
              builder: (context, snapshot){
                if(snapshot.data == null){
                  return Container(
                    decoration: BoxDecoration(
                        gradient: LinearGradient(
                            begin: Alignment.topRight,
                            end: Alignment.bottomRight,
                            colors: [Colors.white,Colors.amber]
                        )
                    ),
                    child: Center(
                      child: Text('Loading...'),
                    ),
                  );
                }
                else return ListView.builder(
                    physics: ScrollPhysics(),
                    itemCount: snapshot.data!.length,
                    itemBuilder: (BuildContext context, int index){
                      return ListTile(
                        title: Text("Cod. Macchina: "+snapshot.data![index].codice_macchina),
                        subtitle: Text("Inizio: "+snapshot.data![index].inizio),
                        trailing: Text("Durata: "+snapshot.data![index].durata),
                          onLongPress: () {
                            //Qui definisco l'Alert da visualizzare nel caso in cui l'utente
                            //vuole cancellare un elemento della lista
                            showDialog(context: context,
                              builder: (BuildContext DialogContext) => AlertDialog(
                                title: const Text('Elimina elemento'),
                                content: const Text("Vuoi eliminare l'elemento selezionato ?"),
                                actions: [
                                  TextButton(
                                    onPressed: () {
                                      Navigator.pop(DialogContext, 'Ok');
                                      var can = DeleteFunction(snapshot.data![index].id,token.token);
```

```
                                can.then((value) => (value) ? {
                                  log("va bene 1"),
                                  //Se la funzione esegue tutto correttamente l'utente viene
                                  //avvisato con un messaggio e l'applicazione torna alla
                                  // schermata precedente
                                  Fluttertoast.showToast(msg: "Operazione completata
                                  con successo", fontSize: 18,
                                      timeInSecForIosWeb: 18),
                                  Navigator.pop(context),
                                }
                                    : Fluttertoast.showToast(msg: "Operazione non riuscita,
                                    riprova piu' tardi ", fontSize: 18,
                                    timeInSecForIosWeb: 85),
                                );
                              },
                              child: const Text('Ok'),
                            ),
                            TextButton(
                              onPressed: () {
                                Navigator.pop(context, 'Annulla');
                              },
                              child: const Text('Annulla'),
                            )
                          ],

        ...

    }
```

Listato 4.19. Questo è un esempio di schermata implementata per la visualizzazione delle proprie prenotazioni all'interno di una lista

4.4 Implementazione lato client con Kotlin

L'ultimo client è stato sviluppato in Kotlin, un linguaggio che ci permette di sviluppare app native per sistemi operativi Android.

4.4.1 Lista degli articoli del magazzino

Nel Listato 4.20 possiamo vedere come è stato realizzato un Fragment in Kotlin, per l'applicazione Android. Il metodo `onActivityCreated` è quello lanciato dopo che il fragment è stato creato. Al suo interno, come prima cosa, recuperiamo le credenziali che sono state inviate dalla precedente schermata; esse contengono il TOKEN necessario per essere autenticati lato server; dopo, per poter fare la richiesta al server, creiamo una istanza di Moshi per il parsing della risposta, e una di Retrofit, per la richiesta. A tal fine viene chiamata la funzione `risultato`; essa lancia una coroutine usando il "GlobalScope" avvalendosi del thread di I/O.

Dentro la coroutine abbiamo un blocco `try/catch` per gestire le eccezioni e alcuni Log che servono per fare debug; viene fatta concretamente la richiesta al server e la risposta viene assegnata alla variabile "response". Se la risposta è affermativa viene assegnata ad una variabile il corpo della risposta, ci spostiamo dal thread di I/O al Main thread, perchè solo dentro questo thread possiamo modificare la UI.

Dopo di ciò viene creata la variabile `recyclerView` collegata alla recyclerview dentro il file XML; infine, vengono valorizzate alcune proprietà di questa variabile, tra cui l'adapter, a cui passiamo una istanza dell'adapter creato da noi. Per creare questa istanza passiamo al costruttore il corpo della risposta del server. Sullo schermo del dispositivo mobile verrà visualizzata la lista degli articoli del magazzino.

```
package homeProf.magazzino

import android.os.Bundle
import android.util.Log
import androidx.fragment.app.Fragment

...

class listaMagazzino : Fragment() {

    ...

    override fun onActivityCreated(savedInstanceState: Bundle?) {
        super.onActivityCreated(savedInstanceState)
    //vengono recuperate le credenziali
        var credenziali = arguments?.get("credenziali") as loginResponse

        var token = credenziali.token

        val moshi = Moshi.Builder()
            .add(KotlinJsonAdapterFactory())
            .build()

        val api = Retrofit.Builder()
            .baseUrl("http://10.0.2.2/progettolaurea/")
            .addConverterFactory(MoshiConverterFactory.create(moshi))
            .build()
            .create(ApiRequests::class.java)

        risultato(api,token)

    }
//viene fatta la richiesta al server
    fun risultato(api: ApiRequests,token: String){
        val impostazioni = "applicatio/json"
        Log.d("mainActivityPro","va bene1")
        GlobalScope.launch(Dispatchers.IO) {
            Log.d("mainActivityPro","va bene2")
            try {
                Log.d("mainActivityPro","va bene3")
                var response = api.ListaMagazzinoRequest(impostazioni,"Bearer "+token)
            // se la risposta e' positiva viene inizializzata la RecycleView con il risultato
                if (response.isSuccessful) {
                    Log.d("mainActivityPro","va bene3")
                    var data = response.body()!!
                    withContext(Dispatchers.Main){
                        //viene inizializzata la recycleView con la risposta del server
                        val recyclerView = view?.findViewById<RecyclerView>(R.id.recycler_view_simo)
                        recyclerView?.layoutManager = LinearLayoutManager(activity)
                        recyclerView?.adapter= CustomAdapterListaMagazzino(data)
                        recyclerView?.addItemDecoration(
                                DividerItemDecoration(
                                        activity,
                                        DividerItemDecoration.VERTICAL
                                )
                        )
                    }
                }
                else {
                    Log.d("mainActivityPro",response.code().toString())
                }

              } catch (e: Exception) {
                  withContext(Dispatchers.Main) {
                      Log.e("MainActivity", e.toString())
                  }
              }
        }
    }
}
```

Listato 4.20. Questo è un esempio di schermata implementata per la visualizzazione delle proprie prenotazioni all'interno di una lista

4.4.2 Adapter della lista sugli articoli del magazzino

Nel Listato 4.21 possiamo vedere come è stato realizzato un adapter per una specifica recyclerview; questa classe che estende `RecyclerView.Adapter` deve fare l'override di tre metodi. Il primo è `onCreateViewHolder`, che serve per fare l'inflating degli elementi della lista. Il secondo metodo è `onBindViewHolder`, che serve per aggiorna-

re gli elementi della lista; infine, abbiamo il metodo `getItemCount`, che ci restituisce il numero totale di elementi della lista.

La classe `ViewHolder` rappresenta il singolo elemento della lista; infatti, in questo esempio, abbiamo che ogni elemento della lista contiene una singola textView all'interno della quale viene inserito il nome dell'articolo del magazzino.

```
package homeProf.magazzino.lista

import android.view.LayoutInflater
import android.view.View

...

/*
In questa classe e' definita la RecycleView che visualizza i dati restituiti dal server
 */
class CustomAdapterListaMagazzino(private val data: List<Articoli>) :
 RecyclerView.Adapter<CustomAdapterListaMagazzino.ViewHolder>() {
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        val rowItem: View = LayoutInflater.from(parent.context).inflate(R.layout.list_item_view, parent,false)
        return ViewHolder(rowItem)
    }
//qui vengono valorizzate le textView del fragment
    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        var lunghezza = data[position].toString().length
        holder.textView.text =" "+ data[position].toString().subSequence(21,lunghezza-1)
    }

    override fun getItemCount(): Int {
        return data.size
    }

    class ViewHolder(view: View) : RecyclerView.ViewHolder(view){
        val textView: TextView

        init {
            textView = view.findViewById(R.id.textview_simo)
        }
    }

}
```

Listato 4.21. Questo è un esempio di adapter utlizzato per le recyclerview

5

Manuale utente della componente relativa alle risorse interne

In questo capitolo riportiamo il manuale d'uso per l'utente finale. In particolare, spiegheremo in maniera sintetica e precisa le azioni che si possono compiere per utilizzare al meglio tutte le funzionalità del sistema. Le istruzioni riportate sono analoghe per gli altri casi.

5.1 Istruzioni per l'autenticazione

5.1.1 Primo utilizzo

Una volta avviata l'applicazione viene visualizzata la schermata di Figura 5.1 tramite la quale bisogna autenticarsi. Come prima cosa bisogna inserire le proprie credenziali. In particolare, nel campo (L1) è necessario inserire lo username; nel campo (L2), invece, bisogna inserire la password. Infine, è necessario premere il pulsante (L3) per completare l'operazione di autenticazione.

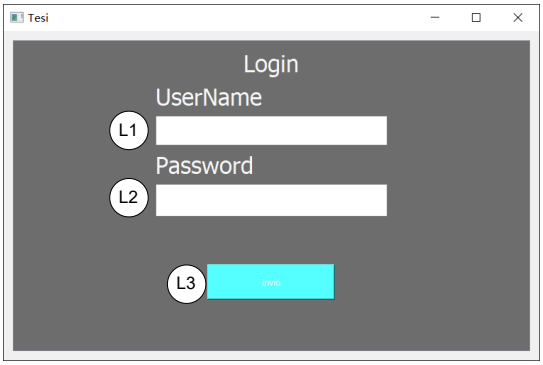

Figura 5.1. Schermata di login

5.1.2 Utilizzo nel caso di errore

Nella schermata di Figura 5.2 l'utente ha inserito delle credenziali sbagliate, o non è autorizzato ad usare l'applicazione. Quindi, siamo in una situazione di errore. In questo caso, è necessario leggere il contenuto del messaggio di errore, e premere il pulsante (Le1) per continuare.

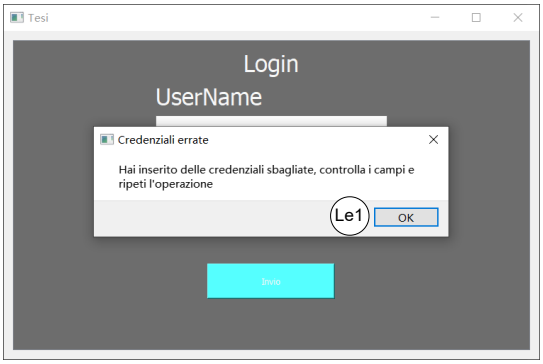

Figura 5.2. Schermata di errore del login

5.2 Istruzioni per la navigazione

5.2.1 Primo utilizzo per il responsabile

Dopo essersi autenticati, si giunge alla schermata di Figura 5.3. In essa è possibile cliccare sul pulsante (H1) per accedere alla `Vista Anagrafiche`, oppure sul pulsante (H2) per accedere alla `Vista Lavorazione`, o anche sul pulsante (H3) per accedere alla `Vista Strategiche`.

5.2.2 Primo utilizzo per il contabile

Nella schermata di Figura 5.4 è possibile cliccare solo sul pulsante (Hc1) per accedere alla `Vista Dati Finanziari`.

5.3 Istruzioni per la visualizzazione dei dati anagrafici

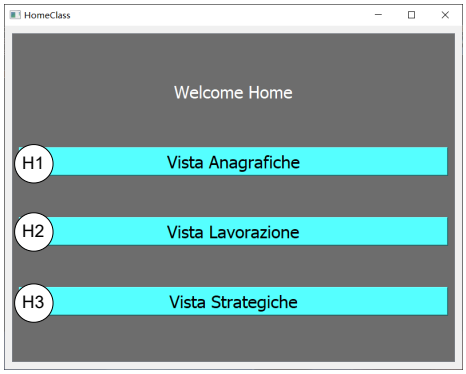

Figura 5.3. Schermata Home per il responsabile

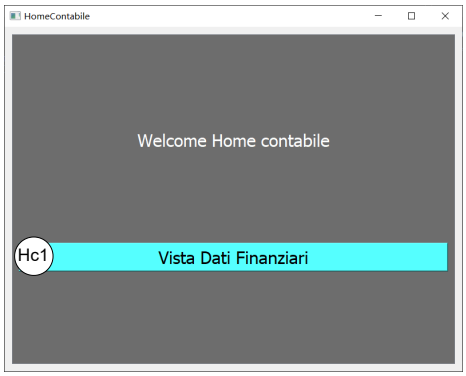

Figura 5.4. Schermata Home per il contabile

5.3.1 Primo utilizzo

Nella schermata di Figura 5.5 è possibile accedere ad alcune liste, ovvero: la `Lista dipendenti`, cliccando sul pulsante (A1), la `Lista clienti`, cliccando sul pulsante (A2), la `Lista modelli`, cliccando sul pulsante (A3), e, infine, la `Lista macchinari` cliccando sul pulsante (A4). È, inoltre, possibile accedere, da questa schermata, ad alcune funzionalità di inserimento. Si può inserire un dipendente cliccando sul pulsante (A5), un cliente cliccando sul pulsante (A6), un modello cliccando sul pulsante (A7); infine, si può inserire un macchinario cliccando sul pulsante (A8).

5.4 Istruzioni per la visualizzazione dei dati sulle lavorazioni

Figura 5.5. Schermata Vista Anagrafiche

5.4.1 Primo utilizzo

Nella schermata di Figura 5.6 è possibile accedere ad alcune liste, ovvero: la `Lista commesse`, cliccando sul pulsante (`La1`), la `Lista DDT`, cliccando sul pulsante (`La2`), la `Lista lavoro`, cliccando sul pulsante (`La3`) e, infine, la `Lista Trasferimento`, cliccando sul pulsante (`La7`). È, inoltre, possibile accedere, da questa schermata, ad alcune funzionalità di inserimento. Si può inserire una commessa cliccando sul pulsante (`La4`), un DDT cliccando sul pulsante (`La5`); infine, si può assegnare un lavoro cliccando sul pulsante (`La6`).

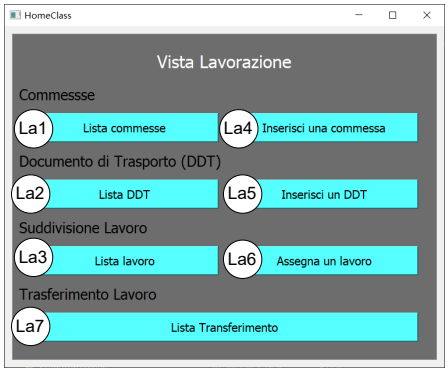

Figura 5.6. Schermata Vista Lavorazione

5.5 Istruzioni per la visualizzazione dei dati strategici

5.5.1 Primo utilizzo

Nella schermata di Figura 5.7 è possibile accedere ad alcune liste, ovvero: la `Lista aggiornamenti`, cliccando sul pulsante (`S1`), la lista che contiene gli `Articoli del magazzino`, cliccando sul pulsante (`S2`); infine, abbiamo la `Lista dati finanziari` cliccando sul pulsante (`S3`).

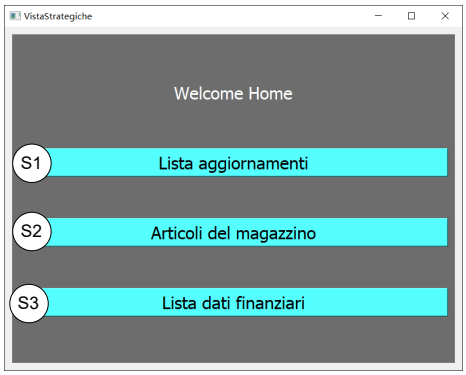

Figura 5.7. Schermata Vista Strategica

5.6 Istruzioni per la visualizzazione di una lista

5.6.1 Primo utilizzo

Nella schermata di Figura 5.8, è possibile visualizzare l'elenco di tutti i dipendenti. Per ogni suo elemento è possibile effettuare la visualizzazione dettagliata dell'elemento selezionato, cliccando sul pulsante (`Li1`). In alternativa, è possibile effettuare la modifica dell'elemento selezionato, cliccando sul pulsante (`Li2`). Infine, è possibile effettuare la cancellazione dell'elemento selezionato, cliccando sul pulsante (`Li3`). In generale, un elemento della lista può avere meno di tre funzioni.

5.7 Istruzioni per la visualizzazione dettagliata

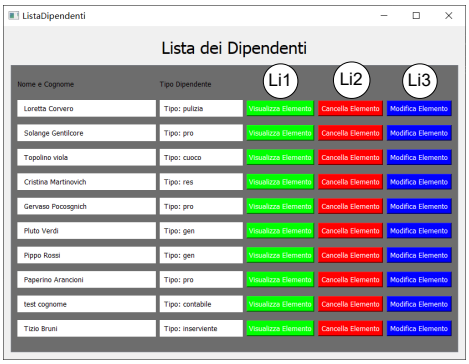

Figura 5.8. Lista dei dipendenti

5.7.1 Primo utilizzo

Nella schermata di Figura 5.9 è possibile visualizzare dettagliatamente tutti i dati di un dipendente (Dv).

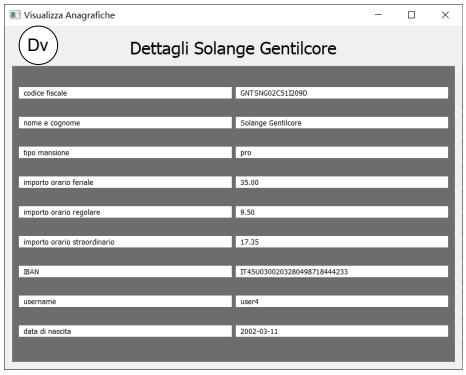

Figura 5.9. Dettagli dipendente

5.8 Istruzioni per un inserimento

5.8.1 Primo utilizzo

Nella schermata di Figura 5.10 è possibile effettuare l'operazione di inserimento. A tal fine, è necessario anzitutto compilare tutti i campi obbligatori, cioè contrassegnati con " * ". I campi da compilare con i dati del nuovo dipendente sono: il campo (D1), in cui si inserisce il codice fiscale; il campo (D2), in cui si inserisce il nome ed

il cognome; il campo (D3), in cui si inserisce il tipo di mansione; il campo (D4), in cui si inserisce l'importo per l'orario feriale; il campo (D5), in cui si inserisce l'importo per l'orario regolare; il campo (D6), in cui si inserisce l'importo per l'orario straordinario; il campo (D7), in cui si inserisce il codice IBAN; il campo (D8), in cui si inserisce lo username; il campo (D9), in cui si inserisce la data di nascita; il campo (D10), in cui si inserisce la password; il campo (D11), in cui si inserisce di nuovo la password, per confermarla. Infine, cliccando sul pulsante (D12), si completa l'operazione.

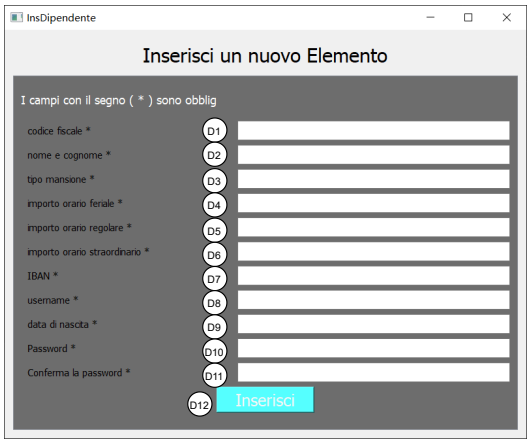

Figura 5.10. Schermata per l'inserimento di un dipendente

5.8.2 Utilizzo nel caso di errore

Nella schermata di Figura 5.11 è possibile visualizzare il messaggio di errore che l'applicazione restituisce quando inseriamo dei dati non corretti. In questo caso, è necessario premere il pulsante (De1) per continuare ad usare l'applicazione.

5.9 Istruzioni per una modifica

5.9.1 Primo utilizzo

Nella schermata di Figura 5.12 è possibile effettuare l'operazione di modifica. A tal fine è necessario, anzitutto mostrare tutti i campi con gli attuali valori; i campi

Figura 5.11. Messaggio di errore per l'inserimento

obbligatori sono contrassegnati con " * ". I campi che si possono modificare sono: il campo (Md1), che contiene il tipo di mansione; il campo (Md2), che contiene l'importo orario feriale; il campo (Md3), che contiene l'importo orario regolare; il campo (Md4), che contiene l'importo orario straordinario; il campo (Md5), che contiene l'IBAN; il campo (Md6), che contiene lo username. Infine, cliccando sul pulsante (Md7), si conclude l'operazione.

Figura 5.12. Modifica di un dipendente

5.10 Istruzioni per una cancellazione

5.10.1 Primo utilizzo

Nella schermata di Figura 5.13 è possibile effettuare l'operazione di cancellazione. A tal fine, viene visualizzato il messaggio che richiede la conferma per la cancellazione. Cliccando sul pulsante (Del1) si conferma l'operazione; invece, cliccando sul pulsante (Del2), si annulla l'operazione.

Figura 5.13. Schermata per la cancellazione di un dipendente

5.11 Istruzioni per l'autenticazione nell'app Android

5.11.1 Primo utilizzo

Una volta avviata l'applicazione, viene visualizzata la schermata di Figura 5.14 tramite la quale è necessario autenticarsi. A tal fine, è necessario, innanzitutto, inserire le proprie credenziali. In particolare, nel campo (KL1) è necessario inserire lo username, nel campo (KL2), invece, è necessario inserire la password. Infine, è necessario premere il pulsante (KL3) per completare l'operazione di autenticazione.

5.11.2 Utilizzo nel caso di errore

Nella schermata di Figura 5.15 l'utente ha inserito delle credenziali sbagliate, o non è autorizzato ad usare l'applicazione. Quindi, siamo in una situazione di errore. In questo caso, è necessario leggere il contenuto del messaggio di errore e, successivamente, premere il pulsante (Ke1) per continuare.

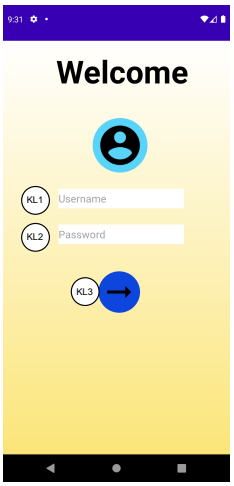

Figura 5.14. Schermata di login Android

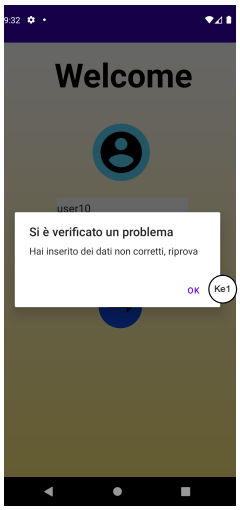

Figura 5.15. Schermata di errore del login Android

5.12 Istruzioni per la navigazione nell'app Android

5.12.1 Home page del dipendente professionale

Dopo essersi autenticati, siamo nella schermata delle Figure 5.16 e 5.17. In essa è possibile cliccare sul pulsante (KH1) per accedere alla `LISTA MAGAZZINO`, sul pulsante (KH2) per effettuare una `RICHIESTA MAGAZZINO`, sul pulsante (KH3) per accedere alla

LISTA RICHIESTE MAGAZZINO, sul pulsante (KH4) per effettuare una PRENOTAZIONE MACCHINARIO, sul pulsante (KH5) per accedere alla LISTA PRENOTAZIONI personali, sul pulsante (KH6) per accedere alla LISTA PRENOTAZIONI GENERICHE, sul pulsante (KH7) per accedere alla LISTA DELLE SUDDIVISIONI LAVORO, sul pulsante (KH8) per accedere alla LISTA DEI MIEI TRASFERIMENTI DI LAVORO, sul pulsante (KH9) per effettuare un NUOVO TRASFERIMENTO DI LAVORO, sul pulsante (KH10) per accedere alle ANAGRAFICHE, sul pulsante (KH11) per accedere alla lista dei MESSAGGI e, infine, sul pulsante (KH12) per effettuare il LOGOUT.

Figura 5.16. Home page del dipendente professionale (prima parte)

5.13 Istruzioni per la visualizzazione di una lista nell'app Android

5.13.1 Primo utilizzo

Siamo nella schermata di Figura 5.18. In essa è possibile visualizzare l'elenco di tutte le prenotazioni che sono state effettuate dall'utente. Per ogni suo elemento è

Figura 5.17. Home page del dipendente professionale (seconda parte)

possibile effettuare l'operazione di cancellazione (KL1*) tenendo premuto su di esso.

Figura 5.18. Lista delle prenotazioni personali

5.14 Istruzioni per una cancellazione nell'app Android

5.14.1 Primo utilizzo

Nella schermata di Figura 5.19 è possibile effettuare l'operazione di cancellazione. A tal fine, viene visualizzato il messaggio che richiede la conferma di tale operazione. Cliccando sul pulsante (KD2) si conferma l'operazione; invece, cliccando sul pulsante (KD1), si annulla l'operazione.

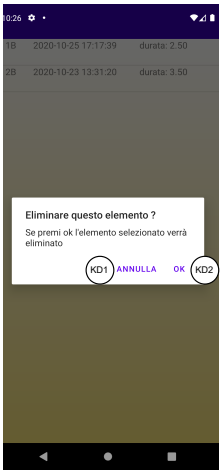

Figura 5.19. Schermata per la cancellazione di una prenotazione in Android

5.15 Istruzioni per un inserimento nell'app Android

5.15.1 Primo utilizzo

Nella schermata di Figura 5.20 è possibile effettuare l'operazione di inserimento. Come prima cosa è necessario compilare tutti i campi. Tali campi sono: il campo (KI1), in cui si inserisce il codice della macchina; il campo (KI2), in cui si inserisce la durata dell'utilizzo del macchinario, espressa in ore; i campi (KI3), (KI4), (KI5), (KI6) in cui si inserisce la data (espressa in giorno, mese, anno) e l'ora in cui si utilizzerà il macchinario. Infine, cliccando sul pulsante (KI7), si completa l'operazione.

Figura 5.20. Schermata per l'inserimento di una prenotazione in Android

5.15.2 Utilizzo nel caso di errore

Nella schermata di Figura 5.21 è possibile visualizzare il messaggio di errore che l'applicazione restituisce quando inseriamo dei dati non corretti. In questo caso, è necessario premere il pulsante (Ke2) per continuare ad usare l'applicazione.

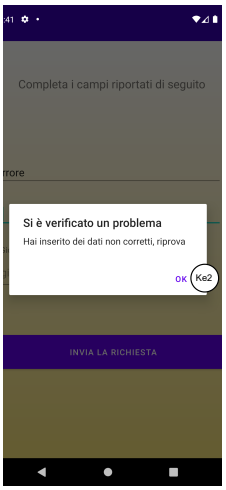

Figura 5.21. Messaggio di errore per l'inserimento in Android

5.16 Istruzioni per l'autenticazione nell'app Flutter

5.16.1 Primo utilizzo

Una volta avviata l'applicazione, viene visualizzata la schermata di Figura 5.22 tramite la quale è necessario autenticarsi. A tal fine è necessario, innanzitutto, inserire le proprie credenziali. Nel campo (FL1) è necessario inserire lo username, nel campo (FL2), invece, bisogna inserire la password. Infine, è necessario premere il pulsante (FL3) per completare l'operazione di autenticazione.

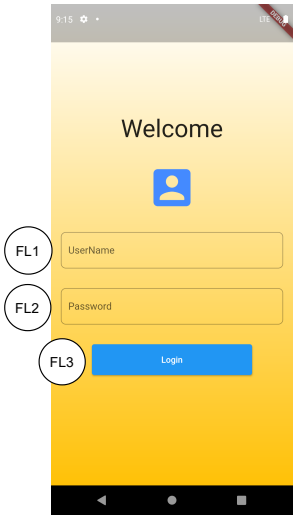

Figura 5.22. Schermata di login Flutter

5.16.2 Utilizzo nel caso di errore

Nella schermata di Figura 5.23 l'utente ha inserito delle credenziali sbagliate, o non è autorizzato ad usare l'applicazione. Quindi, siamo in una situazione di errore. In questo caso, è necessario leggere il contenuto del messaggio di errore e, successivamente, premere il pulsante (LFe1) per continuare.

5.17 Istruzioni per la navigazione nell'app Flutter

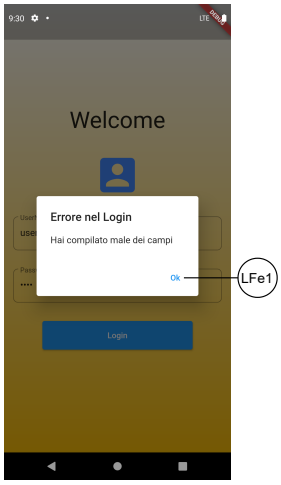

Figura 5.23. Schermata di errore del login Flutter

5.17.1 Home page del dipendente professionale

Dopo essersi autenticati, si arriva alla schermata di Figura 5.24. In essa è possibile cliccare sul pulsante (FH1), per accedere alla `Lista oggetti del magazzino`, sul pulsante (FH2), per effettuare una `Richiesta oggetti del magazzino`, sul pulsante (FH3), per accedere alla `Lista delle richieste fatte`, sul pulsante (FH4), per accedere alla `Lista delle tue suddivisioni lavoro`, sul pulsante (FH5), per accedere alla `Lista delle prenotazioni generiche`, cliccare sul pulsante (FH6) per accedere alla `Lista delle prenotazioni personali`, sul pulsante (FH7), per effettuare l'`Inserimento di una prenotazione`, sul pulsante (FH8), per effettuare l'`Inserimento di un trasferimento del tuo lavoro`, sul pulsante (FH9), per accedere alla `Lista dei tuoi trasferimenti di lavoro`, sul pulsante (FH10), per accedere alla lista dei `Messaggi`, sul pulsante (FH11), per accedere ai `Dati personali` e, infine, sul pulsante (FH12) per effettuare il `Logout`.

Figura 5.24. Home page del dipendente professionale in Flutter

5.18 Istruzioni per la visualizzazione di una lista nell'app Flutter

5.18.1 Primo utilizzo

Nella schermata di Figura 5.25 è possibile visualizzare l'elenco di tutti i trasferimenti di lavoro che sono stati effettuati dall'utente. Per ogni elemento della schermata è possibile, tenendolo premuto, effettuare l'operazione di cancellazione (FL1*).

5.19 Istruzioni per una cancellazione nell'app Flutter

5.19.1 Primo utilizzo

Nella schermata di Figura 5.26 è possibile effettuare l'operazione di cancellazione. Innanzitutto, viene visualizzato il messaggio che richiede la conferma per la cancellazione. Cliccando sul pulsante (FD1) si conferma l'operazione; invece, cliccando sul pulsante (FD2), si annulla l'operazione.

5.20 Istruzioni per un inserimento nell'app Flutter

Figura 5.25. Lista dei trasferimenti di lavoro

Figura 5.26. Schermata per la cancellazione di un trasferimento di lavoro in Flutter

5.20.1 Primo utilizzo

Nella schermata di Figura 5.27 è possibile effettuare l'operazione di inserimento. Innanzitutto, è necessario compilare tutti i campi. Tali campi sono: il campo (FI1), in cui si inserisce il codice della commessa, il campo (FI2), in cui si inserisce la data del trasferimento di lavoro, il campo (FI4), in cui si inserisce il valore unitario del

trasferimento di lavoro, il campo (FI5), in cui si inserisce la quantità di lavoro che si vuole trasferire, il campo (FI6), in cui si inserisce il codice fiscale del dipendente che deve ricevere tale trasferimento. Infine, cliccando sul pulsante (FI7), si completa l'operazione.

Figura 5.27. Schermata per l'inserimento di un trasferimento di lavoro

5.20.2 Utilizzo nel caso di errore

Nella schermata di Figura 5.28 è possibile visualizzare il messaggio di errore che l'applicazione restituisce quando inseriamo dei dati non corretti. In questo caso, è necessario premere il pulsante (Fe2) per continuare ad usare l'applicazione.

Figura 5.28. Messaggio di errore per l'inserimento in Flutter

6

Conclusioni

In questa tesi è stato studiato e implementato un sistema per digitalizzare i processi produttivi all'interno di un'azienda manifatturiera di medie dimensioni. In particolare, la presente tesi si è concentrata sulla componente relativa alle risorse interne, mentre la tesi del candidato `Yihang Zhang` si è occupata della componente finanziaria.

Abbiamo cominciato studiando la realtà d'interesse; le informazioni necessarie sono state reperite mediante un'intervista con il responsabile dell'azienda. Questa figura ci ha spiegato come venivano gestiti i vari processi produttivi e le informazioni che tali processi producevano. Durante l'intervista sono state effettuate delle domande generiche, il cui scopo era quello di acquisire una visione globale dell'azienda; successivamente sono state fatte domande più specifiche per capire al meglio il funzionamento dell'azienda, per raccogliere ulteriori informazioni su di essa, per poter iniziare a sviluppare un'idea su quali componenti erano necessarie al nostro sistema per risolvere al meglio le problematiche dell'azienda.

Una volta completata l'intervista, abbiamo scelto il tipo di componenti e l'architettura che il nostro sistema doveva avere. Il sistema doveva gestire, in maniera efficiente, una notevole quantità di dati; quindi, una delle componenti scelte è consistita in un database. Bisognava gestire i vari tipi di dipendenti che lavorano nell'azienda, ciascuno con i propri compiti, e con la sua postazione di lavoro. Quindi, le successive componenti scelte per il sistema sono state un'applicazione desktop e una mobile.

Infine, avevamo bisogno di collegare queste tre componenti, e la scelta migliore era quella di implementare un server; di conseguenza, l'architettura scelta per il sistema è quella client-server.

Il passo successivo è stato quello di analisi dei requisiti. Analizzando l'intervista abbiamo ottenuto i requisiti che dovevano essere soddisfatti dal nostro sistema, e i casi d'uso generati dall'interazione degli utenti con esso. Successivamente abbiamo progettato tutte le varie componenti; in questa fase abbiamo, anche, scelto quale pattern utilizzare per la nostra architettura. In particolare, la scelta è ricaduta sul pattern MVC (Model, View, Controller). Dopo la fase di progettazione siamo passati all'implementazione; durante questa fase abbiamo utilizzato vari framework e linguaggi, in quanto ogni componente richiedeva un linguaggio diverso.

Il risultato finale ottenuto è un sistema costituito dalle componenti sopra citate, in cui tutti i client possono interagire con il database solo tramite il server. Per poter effettuare tali interazioni sono stati implementati dei meccanismi di sicurezza; infatti qualsiasi utente deve effettuare l'autenticazione per poter accedere alle varie funzionalità. Inoltre, il server riconosce il tipo di utente e lo comunica al client, il quale permette all'utente che si è autenticato di accedere solo a specifiche funzionalità.

In generale le funzionalità messe a disposizione dal sistema sono quelle di inserimento, modifica, cancellazione, visualizzazione generica e dettagliata dei dati presenti nel database; il server, ad ogni richiesta, effettua dei controlli sui dati inseriti dall'utente. Abbiamo anche scritto un manuale per gli utenti che serve per semplificare l'utilizzo del sistema.

Il nostro sistema è adatto per un'azienda manifatturiera di medie dimensioni; però questo è solo un punto di partenza. In futuro quest'architettura può essere ampliata, per adattarla a realtà aziendali più grandi e più complesse; infatti, si possono aggiungere altre funzionalità, possono essere inseriti altri tipi di utenti, si possono utilizzare altri protocolli di sicurezza. Le modifiche possono essere effettuate anche lato client, inserendo altri controlli di sicurezza; infine, è possibile migliorare l'interfaccia grafica.

Il nostro sistema può anche essere un'ottima soluzione per aziende di piccole dimensioni; infatti è possibile ridurre le componenti o le funzionalità per adattarlo a realtà aziendali più piccole; si può eliminare uno dei client, ad esempio quello mobile o quello desktop e il sistema funzionerebbe sempre senza problemi; si possono ridurre le dimensioni del database e, di conseguenza, si riducono le dimensioni di tutte le altre componenti.

Riferimenti bibliografici

1. Laravel 8 Documentation. `https://laravel.com/docs/8.x`, 2020.
2. PHP manual. `https://www.php.net/manual/en/`, 2020.
3. PyQt5 Reference Guide. `https://www.riverbankcomputing.com/static/Docs/PyQt5/`, 2020.
4. Python 3.8.12 Documentation. `https://docs.python.org/3.8/`, 2020.
5. Google guides for developers. `https://developers.google.com/`, 2021.
6. P. Atzeni, S. Ceri, P. Fraternali, S. Paraboschi, and R. Torlone. *Basi di dati: modelli e linguaggi di interrogazione*. McGraw-Hill, 2013.
7. P. Atzeni, S. Ceri, P. Fraternali, S. Paraboschi, and R. Torlone. *Basi di dati*. McGraw-Hill, 2018.
8. N. Bellini. *Laravel*. Independently published, 2020.
9. M. Boscaini. *Imparare a programmare in Python*. Apogeo, 2020.
10. Dr M. Fitzpatrick. *Create GUI Applications with Python & Qt5 (PyQt5 Edition): The hands-on guide to making apps with Python*. Independently published, 2020.
11. A. Miola. *Flutter Complete Reference: Create beautiful, fast and native apps for any device*. Independently published, 2020.
12. A. Shvets. *Dive into Design Patterns*. Ebook, 2017.
13. N. Smyth. *Android Studio 4.1 Development Essentials - Kotlin Edition: Developing Android 11 Apps Using Android Studio 4.1, Kotlin and Android Jetpack*. Payload Media, 2020.
14. I. Sommerville. *Ingegneria del Software Decima Edizione*. Pearson Education, 2017.
15. S. Venneri. *PHP 8: Tecniche & Esempi per padroneggiare il linguaggio*. Independently published, 2021.

Ringraziamenti

Finalmente ho raggiunto il termine di questa prima parte del mio percorso universitario. Durante questi tre anni, nonostante le tante difficoltà, ho avuto la possibilità di conoscere ed imparare cose di cui sono veramente appassionato. È stato un percorso difficile, specialmente all'inizio: mi ricordo ancora le prime lezioni, i primi esami ed i tanti momenti di tensione, di ansia e preoccupazione. Le mie motivazioni, però, sono sempre state forti, fin dall'inizio, e questo mi ha permesso di andare sempre avanti, a testa alta, senza abbassarmi di fronte alle difficoltà, che in un percorso universitario sono sempre presenti. Durante questi anni, l'università e gli studi hanno occupato la maggior parte del mio tempo e, devo dire che non è affatto tempo sprecato, anzi ne è valsa veramente la pena!

Vorrei ringraziare il Professore Domenico Ursino, per essere stato sempre presente, fin dall'inizio di questa tesi, con grandissima disponibilità, pazienza, e anche simpatia. Vorrei ringraziare anche il suo collaboratore Enrico Corradini, che è stato molto disponibile durante la fase iniziale della tesi.

Inoltre, voglio fare un grande ringraziamento ai miei amici e compagni di corso. Ma un ringraziamento particolare va al mio collega e compagno di studio, di progetti, di esami, di ricevimenti, di tesi, Yihang Zhang (per gli amici Leo). Una persona simpatica e intelligente con cui ho avuto il piacere di condividere questi tre anni di università. Ci siamo conosciuti quando entrambi preparavamo l'esame di algebra lineare e geometria, e da allora abbiamo sempre studiato insieme e preparato, più o meno, gli stessi esami; ci siamo supportati a vicenda ottenendo ottimi risultati. Ringrazio ancora Alessio Giaccaglia, la sua famiglia e tutte le altre persone che ho conosciuto in palestra: anche se ci siamo visti poche volte, avete contribuito a rendere più spensierati e divertenti questi anni di studio. Un altro ringraziamento particolare va alla mia ragazza Elisa, che in questi tre anni mi è sempre stata vicina, mi ha sempre supportato e, qualche volta, anche sopportato.

Infine, desidero ringraziare la mia Famiglia, i miei genitori, mia nonna, i miei fratelli e mia sorella, che mi hanno sempre appoggiato in questi tre anni di università. Sono veramente contento di aver reso la mia famiglia ancor più fiera di me!