Università degli Studi di Milano - Bicocca
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea in Informatica

Tempo

Progettazione e sviluppo di un'applicazione web gestionale basata su REST API.

Relatore: Prof. Claudio Zandron

Co-relatore: Dott.ssa Annalisa Marra (Sync Lab srl)

Relazione della prova finale di:
Davide Raccuglia
Matricola 807160

Anno Accademico 2019-2020

Alla mia famiglia, che mi ha supportato e permesso di arrivare sino alla fine di questo lungo ed estenuante percorso.
Ad Arianna, per il continuo sostegno, la fiducia in me riposta, per ogni gesto che mi ha reso felice e dato la forza di rialzarmi nei periodi difficili.
A Pietro, per la costante presenza, per i momenti di creatività che infondono ispirazione e motivazione.

Abstract

L'informatizzazione dei processi aziendali gioca un ruolo assai rilevante e ha certamente impatto sull'efficienza generale di un'azienda, nonché sui costi che questa deve sostenere. Questo progetto, frutto di un periodo di stage curricolare, ha come obiettivo quello di informatizzare la rendicontazione dell'attività lavorativa di un dipendente. Con tale elaborato si vogliono, quindi, presentare quelle che sono state le varie fasi coinvolte durante la progettazione e lo sviluppo di tale software, motivando le varie decisioni intraprese. Vengono inoltre introdotti concetti teorici, riguardanti le tecnologie utilizzate, indispensabili per la comprensione del funzionamento generale dell'applicazione e nello specifico per capire l'interazione tra i vari elementi che la compongono dal punto di vista tecnico. Tali concetti sono altresì utili per capire le diverse scelte di implementazione concepite.

Indice

Elenco delle figure

Capitolo 1

Introduzione

L'obiettivo di tale relazione è quello di esporre quanto è stato realizzato durante il periodo di stage, svoltosi presso l'azienda Sync Lab srl, operante nel settore dell'IT Consultant & Service. Il tirocinio, della durata di 3 mesi, ha coinvolto attività di formazione sui framework Spring[A.2], Hibernate[A.3] e Angular[A.1] e l'attuazione di quanto appreso mediante progettazione e sviluppo di un'applicazione web, oggetto di questa tesi.

1.1 TEMPO

Il software progettato è un prodotto gestionale, sviluppato per l'azienda presso cui si è svolta l'attività, il cui compito primario è quello di permettere la gestione della rendicontazione dell'orario di lavoro giornaliero del singolo dipendente, in relazione ai clienti ad esso associati. Lo scopo principale è stato, quindi, quello di sviluppare un sistema in grado di gestire un discreto numero di utenze, una per ogni dipendente, e che, a seconda del livello gerarchico in azienda, fornisca una serie di funzionalità, tra cui permettere l'inserimento dell'attività svolta giornalmente con il relativo numero di ore lavorate. Inoltre, il sistema deve essere in grado di memorizzare l'anagrafica dei dipendenti e dei clienti per tenere traccia degli eventuali progetti attivi e conclusi, e il numero di risorse che vi sono allocate. In particolare ogni volta che una risorsa viene allocata presso un cliente, a questa viene assegnato un codice univoco, il quale mette in relazione, oltre a cliente e risorsa, anche l'attività svolta e il progetto su cui il dipendente viene assegnato.

L'applicazione in questione è stata denominata TEMPO, una sorta di acronimo costituito da una combinazione di alcune tra le prime lettere delle parole *Employees' Timesheet Portal*.

1.2 Processo software utilizzato

Il processo software adottato per la creazione di tale prototipo è UP (*Unified Process*), ossia un framework agile per la gestione del ciclo di sviluppo del software, iterativo ed incrementale, e concepito per gestire progetti e prodotti software. A partire quindi dal-

la specifica fornita, sono state effettuate in alternanza, delle fasi di analisi dei requisiti, progettazione, implementazione e test. In questo modo è stato possibile garantire la realizzazione di un prodotto in grado di soddisfare le esigenze richieste e la gestione di eventuali casi eccezionali o di errore. Tuttavia, date le dimensioni discrete del software da progettare, sarebbe stato possibile adottare un processo a cascata, ormai obsoleto in ogni contesto di sviluppo di medie o grandi dimensioni. Si è comunque deciso di adottare un processo software *agile*, in quanto propedeutico per il mondo lavorativo, ma soprattutto perché più efficiente e riduce al minimo la probabilità di fallimento.

1.3 Unified Modeling Language

Per quanto riguarda la progettazione, si è fatto uso del linguaggio UML (*Unified Modeling Language*), ossia un linguaggio di modellazione e di specifica molto utile quando si utilizza il paradigma di programmazione orientata agli oggetti. Esso permette, tramite l'utilizzo di modelli visuali, di analizzare, descrivere, specificare e documentare un sistema software, anche complesso. In particolare sono stati utilizzati i seguenti diagrammi:

- Diagramma dei casi d'uso, utile per la rappresentazione visuale e sintetica dei casi d'uso sviluppati.

- Diagramma di sequenza, sostanzialmente legato al diagramma dei casi d'uso ed utilizzato per definire la serie di azioni compiute dall'utente e dal sistema quando viene richiesto un determinato servizio, andando a mostrare le interazioni tra i due e gli altri eventuali attori coinvolti.

- Diagramma delle classi, forse quello più importante tra tutti e che rappresenta la struttura dell'intero progetto. Da tale diagramma si andranno poi a definire le classi Java, e le relazioni fra di esse, che andranno a comporre l'applicazione.

- Diagramma di attività, utilizzato per mostrare i vari passi compiuti nelle varie attività che vengono eseguite per espletare una determinata funzione o servizio previsti dal sistema.

1.4 Pattern MVC e REST API

Un ulteriore obiettivo è stato quello di assicurare le principali caratteristiche che un buon software deve possedere, tra cui scalabilità, sicurezza e manutenibilità. Per tale ragione si è pensato ad un'architettura mirata a garantire l'indipendenza fra i vari componenti, utilizzando il pattern MVC (*Model View Controller*) unito ad un web service di tipo REST.

1.4.1 Model View Controller

Model View Controller (comunemente chiamato MVC) è un pattern di progettazione che definisce una struttura del codice dell'applicazione suddivisa in tre componenti differenti, ognuna indipendente dall'altra e con un compito preciso:

- *Model*: è responsabile della gestione dei dati dell'applicazione e dunque fornisce i metodi di accesso ad essi;

- *View*: si occupa di visualizzare i dati all'utente e di gestire l'interazione fra quest'ultimo e l'infrastruttura sottostante;

- *Controller*: riceve i comandi dell'utente attraverso la View e reagisce eseguendo delle operazioni che possono interessare il Model e che portano, generalmente, ad un cambiamento di stato della View.

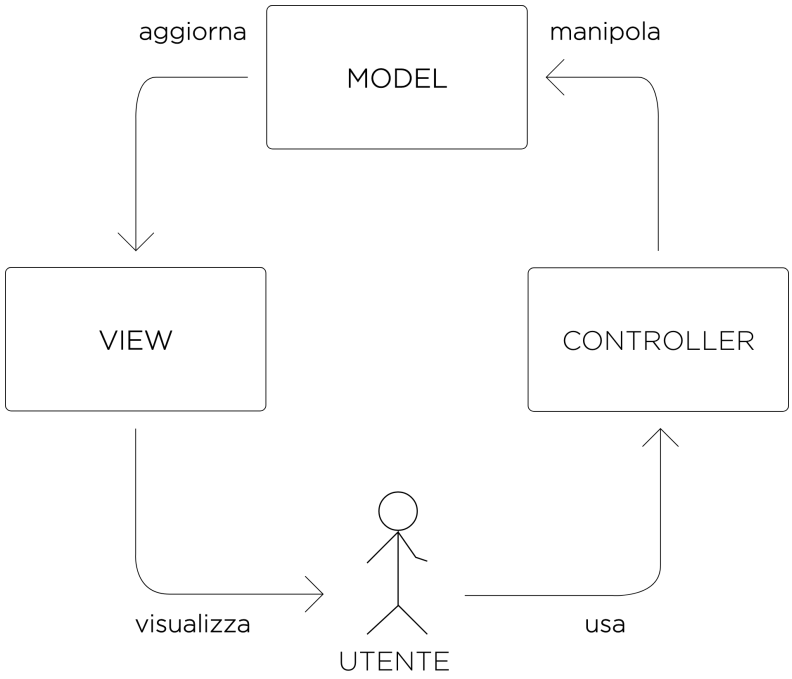

Figura 1.1: Schema di alto livello del pattern MVC.

L'utilizzo di tale pattern ha il vantaggio principale di avere un codice comprensibile, riusabile e soprattutto mantenibile, in quanto vi è una separazione completa dei compiti da svolgere. Inoltre, grazie all'indipendenza tra i vari componenti si ha la possibilità di creare viste e controller diversi, utilizzando lo stesso modello di accesso ai dati e dunque, come già accennato, riutilizzando parte del codice esistente. Nel caso in cui si dovesse verificare un cambio al tipo di database, basterà apportare modifiche al modello, adattandolo alle esigenze e dunque senza dover cambiare il codice delle viste e dei controller.

1.4.2 REST

Quando si progetta un'applicazione web, un aspetto importante da considerare è sicuramente la fruibilità dei servizi che questa fornisce attraverso la rete. REST (*REpresentational State Transfer*) è uno dei principi architetturali per la progettazione di un sistema *network-based* e sostanzialmente offre un insieme di linee guida che si possono riassumere in 4 principi:

- **Identificazione delle risorse**: ogni risorsa deve essere identificata univocamente ed essendo in ambiente web è naturale farlo mediante URI (*Uniform Resource Identifier*).
- **Utilizzo esplicito dei metodi HTTP**: GET, POST, PUT e DELETE, rispettivamente per l'accesso, la creazione, la modifica e l'eliminazione di una risorsa e i quali si andranno a mappare alle operazioni CRUD (Create, Read, Update e Delete).
- **Risorse autodescrittive e collegate fra loro**: le risorse sono separate dalla rappresentazione restituita al client, il quale si preoccuperà lui stesso di definire nella richiesta HTTP il tipo di formato da restituire; ogni risorsa deve inoltre garantire i collegamenti a risorse correlate.
- **Comunicazione senza stato**: tale caratteristica fa parte del protocollo HTTP, difatti ciascuna richiesta non ha alcuna relazione con quelle precedenti e successive.

Dunque per la creazione di un sistema RESTful è sufficiente che lato server venga implementato un servizio con le caratteristiche appena descritte. Mentre, lato client occorre un sistema in grado di effettuare richieste mediante protocollo HTTP, a delle URI che identificano le risorse, e in tali richieste si andrà a specificare un tipo di formato da ricevere che il client stesso sarà in grado di interpretare.

Capitolo 2

Analisi dei requisiti

Allo stato attuale, la gestione della rendicontazione delle ore di lavoro di ogni dipendente viene effettuata mediante un foglio di calcolo Excel, compilato in ogni sua parte, inserendo nome, cognome e orario di lavoro svolto giornalmente. Successivamente, alla fine di ogni mese, il dipendente spedisce tale file via email alla segreteria e al proprio responsabile di sede per l'approvazione. A seguito di un esito positivo viene effettuato il calcolo del salario mensile e quindi l'emissione dello stesso.

Le specifiche fornite inizialmente riguardano perlopiù funzionalità che tale prodotto deve fornire e il tipo di informazioni che deve essere in grado di gestire. Il sistema che s'intende realizzare vuole rendere fruibile il processo di scambio ed elaborazione delle informazioni appena descritto, mediante un prodotto software che ne semplifichi e ne automatizzi alcuni aspetti. Tali specifiche sono state poi raffinate, con un'analisi dei requisiti approfondita, anche grazie alla comunicazione con l'utente finale. Inoltre, in seguito all'analisi dei requisiti, viene presentata l'analisi tecnica atta a spiegare le scelte di progettazione ed implementazione.

2.1 Specifiche generali

Formalmente le funzionalità principali richieste sono le seguenti:

1. assegnamento di un'utenza, ossia nome utente e password, per effettuare l'accesso.
2. inserimento, da parte del dipendente, dei dati utili alla rendicontazione mensile dell'attività lavorativa;
3. invio del rendiconto lavorativo mensile, per approvazione da parte del proprio responsabile;
4. verifica e validazione di tale rendiconto;

Ognuna di queste è stata poi analizzata singolarmente per ricavare nel dettaglio i requisiti necessari ed, eventualmente, delle relative funzionalità aggiuntive. Vengono presentati di seguito i requisiti funzionali e non funzionali rilevati durante l'analisi.

2.2 Requisiti funzionali

Il software in analisi deve garantire un accesso protetto all'area riservata dell'utente. Per accedere esso inserisce username e password e, qualora i dati inseriti fossero errati, il sistema mostra una errore e non consente l'accesso.
Nel momento in cui si accede all'area personale è prevista una sezione in cui è possibile inserire le ore di lavoro, per ogni singolo giorno lavorativo del mese corrente. Per quanto riguarda l'assenza totale o parziale per una o più giornate, si potrà indicare il numero di ore di permesso usufruite e nel caso specifico di assenza totale, quindi per l'intera giornata lavorativa, sarà inibito l'inserimento del numero di ore da parte dell'utente. È importante che in corrispondenza di ogni inserimento sia possibile specificare il codice relativo all'attività svolta in quella giornata. Tale codice viene generato dal responsabile di sede e reso disponibile nell'area personale dell'impiegato. Inoltre l'utente può compilare parzialmente il mese in corso e salvarne lo stato. A quel punto il sistema memorizzerà fisicamente i dati inseriti e quando si verifica un nuovo accesso, ripristinerà la compilazione precedente, permettendone la modifica, la cancellazione oppure l'invio definitivo. Una volta inviato il timesheet, viene impedita l'ulteriore modifica, o cancellazione, dei dati inviati e non è possibile effettuare un nuovo invio.
L'applicazione fornisce altresì una funzionalità per la consultazione dei timesheet relativi ai mesi precedenti e permette di scaricarli localmente in un formato Excel (`.xls`).
Accedendo con l'utenza manager si possono generare dei codici, cosiddetti *commesse*, e assegnarli ad uno specifico impiegato. Non può verificarsi un assegnamento della stessa commessa a più impiegati, poiché questa identifica la mansione svolta in relazione con il cliente e il relativo progetto.
In aggiunta all'utenza manager, il sistema permette all'utente in questione la visualizzazione di tutti i timesheet inviati dai relativi dipendenti appartenenti alla sede cui esso afferisce. Una volta selezionato un timesheet e controllatone la correttezza dei dati, il manager può eseguirne l'approvazione o il rigetto, rispettivamente se i dati inseriti sono coerenti con l'attività svolta in quel determinato mese oppure sono errati. Qualora si verificasse un rigetto, il sistema provvederà a riabilitare la modifica del timesheet nell'area personale del dipendente. Infine, ogni volta che si verifica un assegnamento di una commessa oppure l'avvenuta approvazione o rigetto del timesheet, occorre notificarlo all'impiegato nell'apposita sezione. Viene inoltre mostrata una notifica anche nel caso in cui il dipendente non abbia ancora effettuato l'invio entro il terzultimo giorno del mese corrente.

2.3 Casi d'uso

I casi d'uso sviluppati riguardano i processi di login, di compilazione del timesheet (ts) da parte dell'impiegato e di validazione dello stesso da parte dell'utente manager, i quali vengono riportati nel diagramma in figura. 2.1.

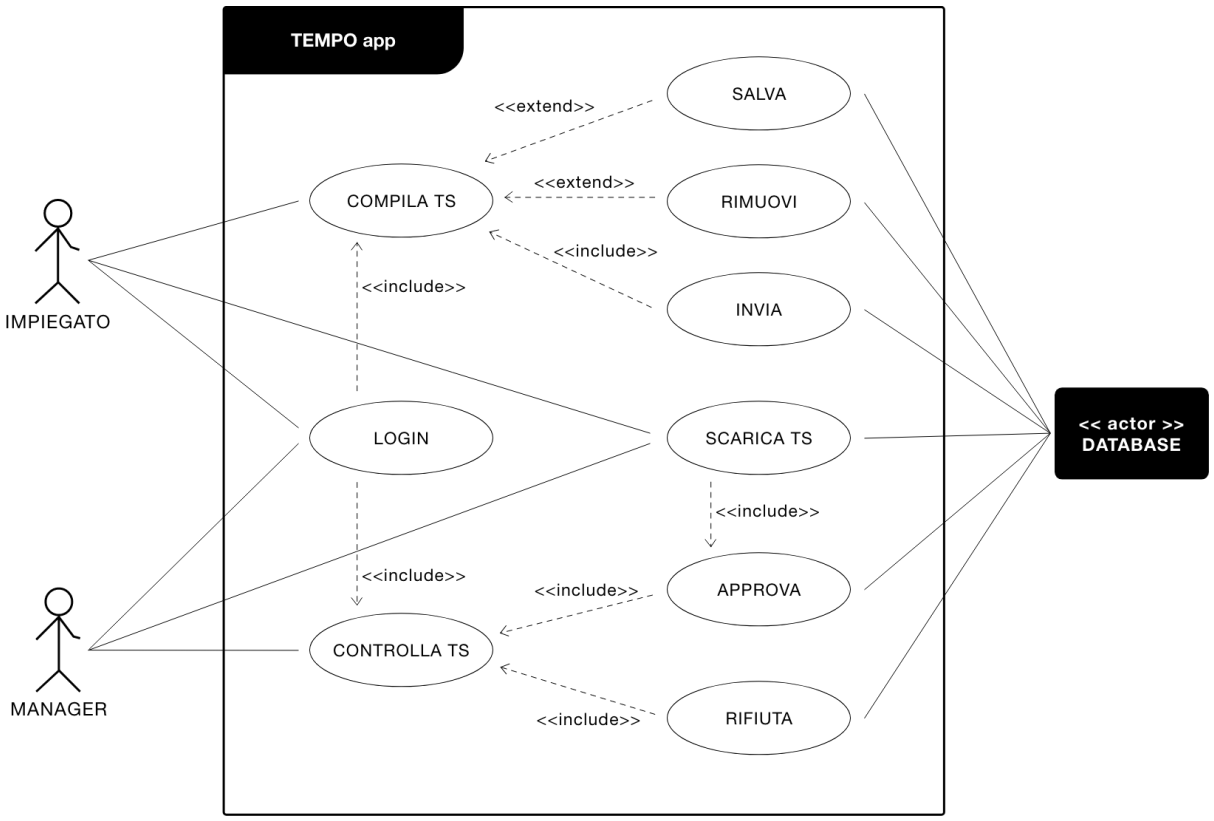

Figura 2.1: Diagramma dei casi d'uso.

All'interno delle ultime due macro-funzionalità citate, ne sono comprese altre. La compilazione del timesheet viene estesa con l'operazione di salvataggio, ossia la memorizzazione fisica dei vari inserimenti effettuati dall'utente, e con l'operazione di rimozione, ovvero la cancellazione dei dati memorizzati mediante l'operazione precedente di salvataggio; e infine include l'invio, che avviene nel momento in cui l'utente decide che la compilazione è completata. Per quanto riguarda il controllo da parte del manager, esso comprende l'approvazione oppure il rigetto del timesheet inviato. Infine, nel momento in cui si verificano invio e conseguente approvazione, sarà disponibile la funzionalità che permetterà lo scaricamento in locale del file Excel.

2.3.1 Login utente

Nome: Login.
Portata: Tempo.
Livello: Obiettivo utente.
Attore primario: Impiegato, Manager, Azienda.
Parti interessate e interessi: Utente (Impiegato, Manager, Azienda), vuole effettuare l'accesso all'applicazione.
Pre-condizioni: L'utente è già stato registrato.

Garanzia di successo: L'utente si autentica e accede alla sua area personale.
Scenario principale di successo:

1. L'utente avvia l'applicazione.
2. Il sistema visualizza i campi in cui inserire le credenziali di accesso.
3. L'utente inserisce username e password.
4. Il sistema verifica la correttezza dei dati inseriti.
5. Il sistema autentica l'utente e visualizza la sua area personale.

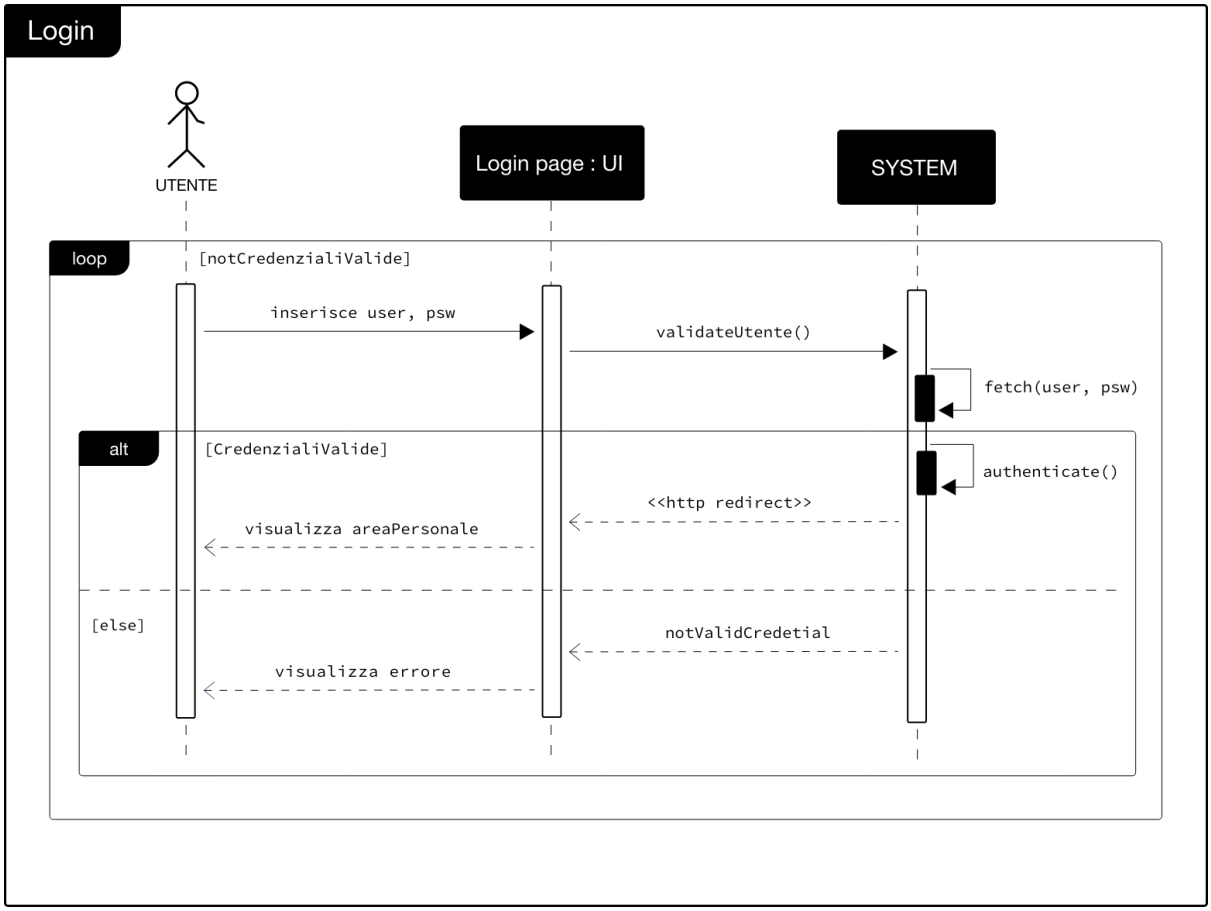

Figura 2.2: Diagramma di sequenza relativo al login utente.

2.3.2 Compilazione timesheet

Nome: Compila ts.
Portata: Tempo.
Livello: Obiettivo utente.
Attore primario: Impiegato.
Parti interessate e interessi: Impiegato, vuole compilare il timesheet indicando le ore

di lavoro giornaliere, eventuali giorni di permesso/ferie e/o malattia.
Pre-condizioni: L'impiegato deve essere registrato e aver effettuato il login.
Garanzia di successo: L'impiegato annota correttamente le ore di lavoro giornaliere e il manager di sede può consultarle per l'eventuale approvazione.
Scenario principale di successo:

1. L'impiegato sceglie il mese d'interesse nell'apposito calendario.
2. L'impiegato seleziona uno fra i codici commessa disponibili.
3. L'impiegato compila la tabella per la prima volta, inserendo le ore di lavoro effettuate per ogni giornata e/o inserendo eventuali giorni di permesso/malattia.
4. L'impiegato termina la compilazione del mese selezionato e invia il timesheet.
5. Il sistema registra i dati inseriti dall'utente.
6. Il sistema mostra un messaggio di completamento dell'operazione.
7. Il sistema disabilita la modifica al timesheet.

Estensioni:
3a. L'impiegato aveva già salvato una compilazione parziale:

1. L'impiegato procede con la compilazione del mese selezionato in precedenza.
2. L'impiegato termina la compilazione del mese selezionato e invia il timesheet.
3. Il sistema registra i dati inseriti dall'utente.
4. Il sistema mostra un messaggio di conferma.
5. Il sistema disabilita la modifica al timesheet.

3b. L'impiegato aveva già salvato una compilazione parziale e decide di eliminarla:

1. L'impiegato rimuove il timesheet precedentemente salvato.
2. Il sistema cancella tutti i dati relativi a quella compilazione.
3. Il sistema mostra un messaggio di conferma.
4. L'impiegato inizia una nuova compilazione.

4a. L'impiegato salva il modulo senza inviarlo:

1. Il sistema salva la compilazione e mantiene i dati per la prossima sessione.
2. Il sistema mostra un messaggio di conferma.

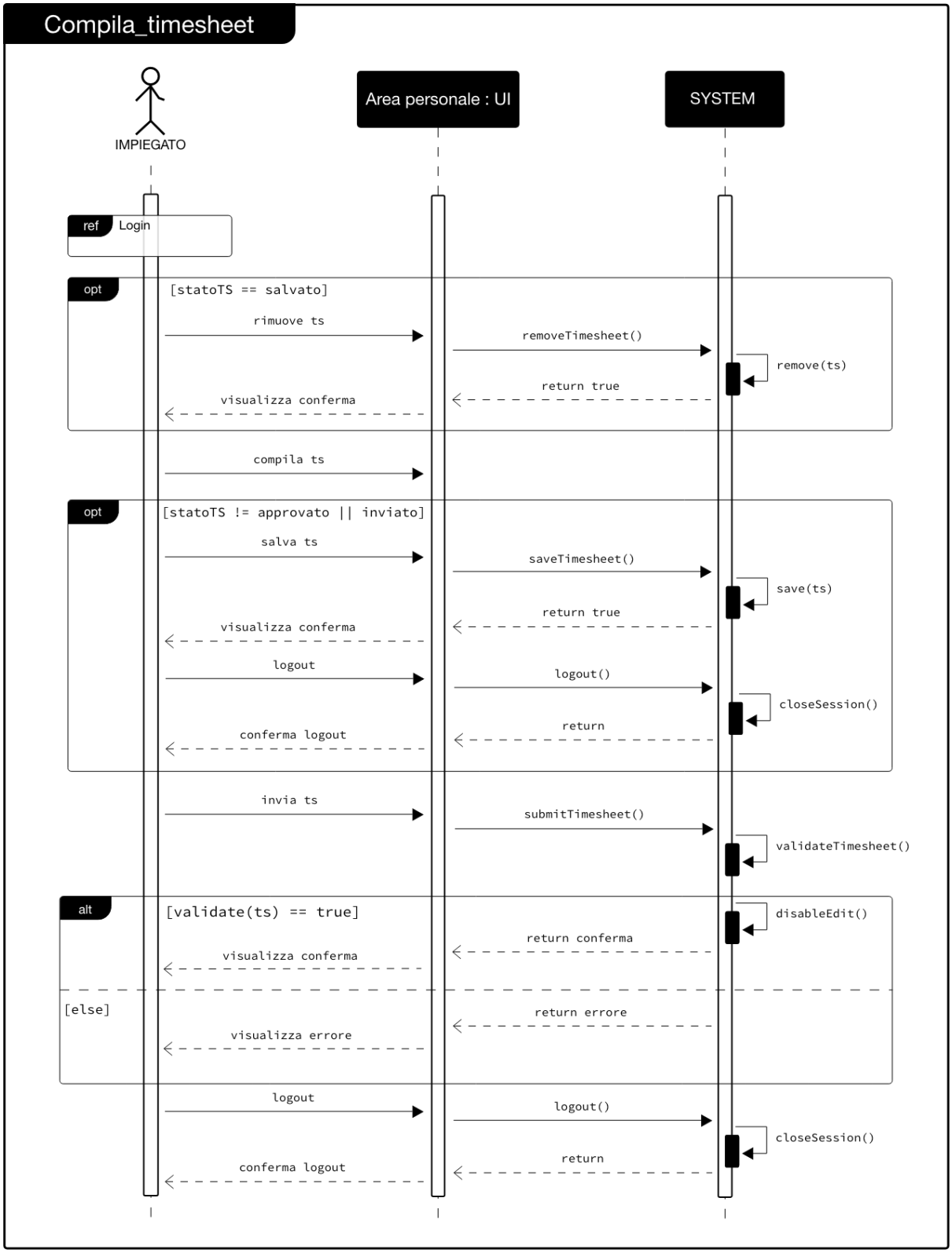

Figura 2.3: Diagramma di sequenza relativo alla compilazione del timesheet.

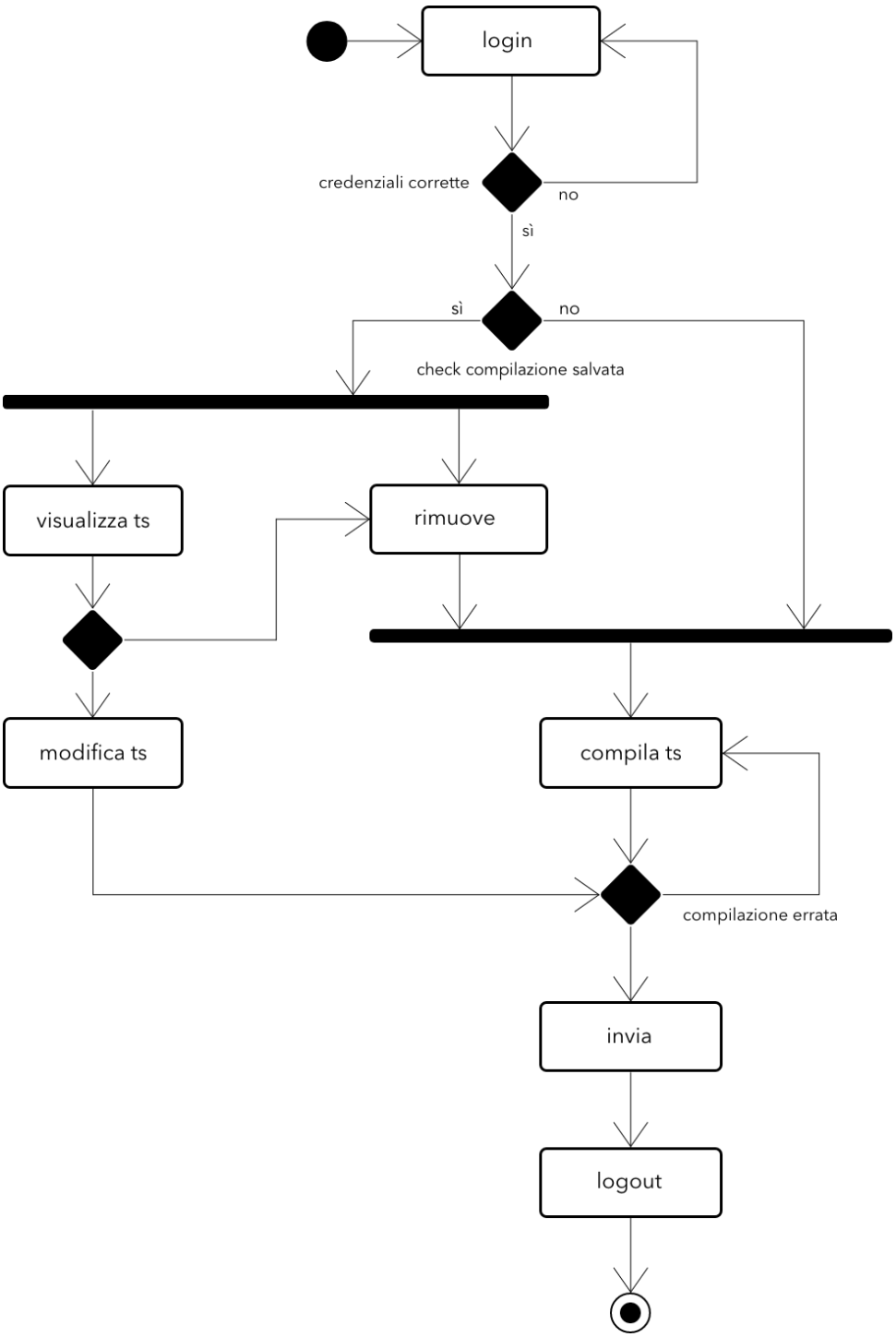

Figura 2.4: Diagramma di attività relativo alla compilazione del timesheet.

2.3.3 Verifica e approvazione del timesheet

Nome: Controlla ts.
Portata: Tempo.
Livello: Obiettivo utente.

Attore primario: Manager.
Parti interessate e interessi:

- Manager, vuole approvare i timesheet inviati dai dipendenti, che siano conformi all'attività di lavoro svolta e compilati correttamente, al fine di completare le rendicontazione mensile.

- Impiegato, vuole la conferma sul timesheet al fine del calcolo del salario mensile.

Pre-condizioni: L'utente manager deve aver effettuato il login.
Garanzia di successo: Il manager segnala all'azienda di procedere con il calcolo del salario e delle imposte dovute. L'azienda emette il pagamento nei confronti del dipendente.
Scenario principale di successo

1. Il manager entra nella sezione relativa ai timesheet inviati dai dipendenti.

2. Il manager seleziona un timesheet e lo esamina.

3. Il manager controlla che il codice commessa inserito sia corretto.

4. Il manager verifica che le ore giornaliere inserite siano coerenti con l'attività svolta dal dipendente.

5. Il manager approva il timesheet.

6. Il sistema registra i dati e mostra un messaggio di conferma.

7. Il sistema rende disponibile il download del timesheet in formato Excel.

Estensioni
5a. Il manager non approva il timesheet:

1. Il manager rigetta il timesheet del dipendente.

2. Il sistema registra i dati e mostra un messaggio di conferma.

3. Il sistema riabilita la modifica del timesheet nell'area personale del relativo dipendente.

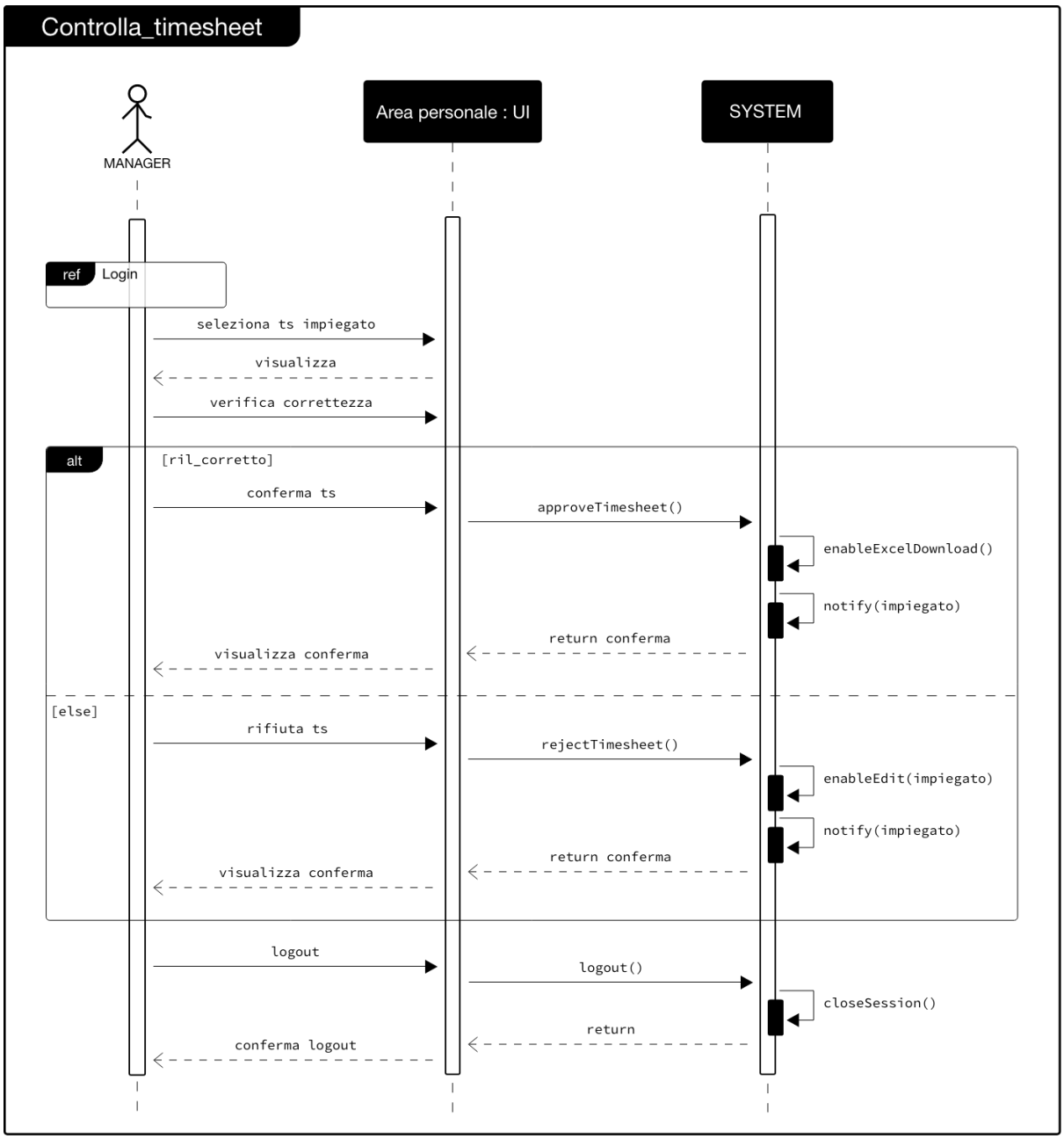

Figura 2.5: Diagramma di sequenza relativo al controllo del timesheet.

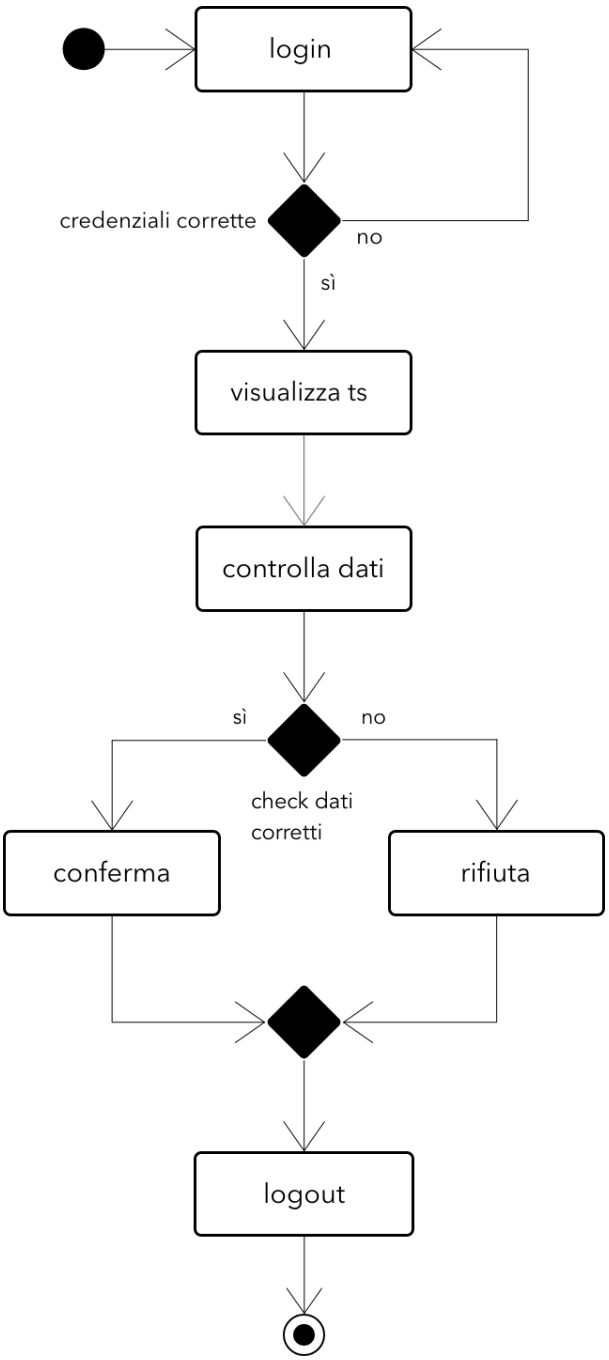

Figura 2.6: Diagramma di attività relativo al controllo del timesheet.

2.3.4 Download del timesheet

Nome: Scarica ts.
Portata: Tempo.
Livello: Obiettivo utente.
Attore primario: Impiegato, Manager, Azienda.
Parti interessate e interessi: Utente (Impiegato, Manager, Azienda), vuole scaricare

localmente il timesheet.
Pre-condizioni: L'utente deve aver effettuato il login. Il timesheet deve esser stato approvato dal manager.
Garanzia di successo: L'utente possiede una copia in locale del timesheet.
Scenario principale di successo:

1. L'utente seleziona il mese di interesse.
2. L'utente verifica lo stato del timesheet.
3. L'utente effettua il download del timesheet in formato Excel.

2.4 Requisiti non funzionali

L'accesso all'area personale viene eseguito attraverso una pagina di login apposita.
Per quanto riguarda i ruoli, il sistema deve prevedere tre livelli di accesso differenti:

- *master*, pensato come utenza aziendale globale, a cui è consentito effettuare qualsiasi tipo di operazione messa a disposizione dall'applicazione e che ha visibilità totale dei dati;
- *manager*, pensato per il responsabile di sede e attraverso cui si ha visibilità sulla rendicontazione dei singoli dipendenti, relativi a quella specifica sede. Tale utenza viene creata dall'utente master;
- *employee*, utenza di base pensata per il dipendente e la quale dà accesso alle sole informazioni personali. Questo tipo di utenza deve essere creata dall'utente master oppure dal manager di sede relativo.

Una volta effettuata l'autenticazione il sistema visualizza la pagina principale, contente il timesheet di lavoro del mese corrente, un calendario e una sezione dedicata alle notifiche. La rappresentazione del timesheet viene effettuata tramite una tabella, la quale comprende le seguenti colonne: codice commessa, data, numero di ore lavorate, numero di ore di permesso/ferie e infine un campo per indicare l'assenza per malattia.
Al momento del salvataggio dei dati inseriti o dell'invio del timesheet, il sistema deve mostrare una conferma di avvenuta operazione oppure un errore se non è stato possibile eseguire l'azione richiesta.

Capitolo 3

Progettazione

In questo capitolo si andranno ad esporre quelle che sono state le scelte di progettazione per l'implementazione del software oggetto di questa relazione. In particolare verranno discusse le scelte tecniche per ciò che concerne la progettazione del client e del server, nonché le scelte sulle tecnologie utilizzate Quest'ultime vengono descritte dettagliatamente nell'appendice A.

Il software in questione è stato pensato per essere scalabile, mantenibile ed integrabile, eventualmente, con altri sistemi già esistenti. Per tale ragione si è deciso di adottare un'architettura software suddivisa in vari livelli, utilizzando in particolare il pattern *Model View Controller* (MVC) introdotto nel capitolo 1, sia lato client che lato server. Mentre per ciò che riguarda i servizi web, si è scelto di utilizzare un'architettura REST (anche questo concetto è stato introdotto nel capitolo 1), implementando una API (*Application Programming Interface*) lato server e accedendo quindi alle risorse mediante protocollo HTTP (*Hypertext Transfer Protocol*). In particolare si tratta di una API che restituisce i dati richiesti in formato JSON (*JavaScript Object Notation*), un'alternativa al classico XML (*Extensible Markup Language*). Questa scelta deriva dal fatto che tale formato è di semplice lettura, in quanto ha uno stile più compatto e intuitivo rispetto all'XML. Inoltre, il processo di analisi, ossia la velocità di interpretazione da parte di un computer, risulta essere più veloce rispetto all'altro linguaggio citato, e questo sempre dato dalla compattezza, dunque dall'utilizzo di meno dati per descrivere una risorsa. Infine, un ulteriore vantaggio è dato dal fatto che un oggetto JSON nella maggior parte dei casi corrisponde interamente agli oggetti dell'applicazione, ciò rende semplice e performante il *binding* dei dati in entrambi i sensi (dal server al client e viceversa).

3.0.1 Client

Lato client l'applicazione viene eseguita mediante web browser. Per l'implementazione si è scelto di utilizzare il framework Angular 8[A.1], il quale è basato sul linguaggio TypeScript e adoperato per la creazione di applicazioni web *single-page*. La scelta di tale framework deriva, appunto, dalla necessità di avere un'applicazione sviluppata in una

singola pagina, costituita da vari componenti (o moduli), i quali vengono caricati dinamicamente a seconda delle azioni dell'utente e senza il bisogno di ricaricare una nuova pagina web, facendo un'ulteriore richiesta al server. Inoltre, l'uso di tale tecnologia rende del tutto indipendente il livello client dal livello server, basando la loro comunicazione esclusivamente sull'impiego di un'architettura REST. In particolare, questa comunicazione avviene usando il protocollo HTTP, sfruttando i quattro metodi `GET`, `POST`, `PUT` e `DELETE`, e servendosi di determinati *endpoints*, ovvero delle URLs (*Uniform Resource Locators*) messe a disposizione dall'API e mediante le quali è possibile accedere ai servizi forniti lato server e quindi alle risorse.
Una possibile alternativa sarebbe potuta essere React.js, una libreria JavaScript sviluppata e mantenuta da Facebook, la quale permette, in modo concettualmente simile, di sviluppare un'applicazione in singola pagina. Un punto a favore è sicuramente l'efficienza, che risulta maggiore rispetto ad Angular, tuttavia possiede diverse lacune in termini di funzionalità e completezza e talvolta è necessario l'utilizzo di ulteriori strumenti di terze parti, ad esempio per il servizio di routing fra componenti dell'applicazione.

Riguardo la struttura del codice lato *front-end*, ossia lato client, come già precedentemente anticipato, viene adottata una struttura multi-livello, in particolare seguendo il paradigma MVC. Nello specifico si ha:

- un livello controller per ogni classe principale: tale ruolo viene svolto dal componente stesso, il quale rappresenta un oggetto dell'applicazione;
- un livello service: contiene la logica di business e contiene le istruzioni per le chiamate HTTP;
- un livello per il modello: viene utilizzato per il *binding* dei dati e quindi definisce la struttura di una specifica entità dell'applicazione.

Ad esempio, considerando di dover rappresentare la lista degli impiegati si avrà quanto segue:

- il componente che rappresenta gli impiegati è costituito dai seguenti files:
  - impiegati.component.css, dove è possibile definire lo stile estetico del componente, una volta visualizzato dal browser;
  - impiegati.component.html, definisce la struttura del componente che il browser dovrà visualizzare e insieme al precedente file va a costituire la vista dell'applicazione;
  - impiegati.component.ts, il quale rappresenta il controller.
- impiegato.service.ts, ossia il livello di servizio, dove vengono definite le chiamate HTTP e la logica di business;

- impiegato.ts, dove viene definita la struttura dell'entità impiegato, dunque gli attributi che lo caratterizzano.

Nota: il tipo di estensione ".ts" è associato ai file che contengono codice TypeScript.

3.0.2 Intefaccia utente

L'interfaccia è stata progettata adoperando il software Adobe XD (*Experience Design*) e poi implementata mediante il framework Angular. L'obiettivo primario nella fase di progettazione è stato quello di ottenere un'interfaccia grafica il più intuitiva e semplice possibile. Per l'implementazione sono stati utilizzati i linguaggi HTML e CSS, rispettivamente per definire la struttura e lo stile grafico di ogni singolo componente, o in generale della pagina web.
Di seguito vengono mostrati alcuni esempi, in particolare la pagina di login e le pagine del timesheet con accesso manager ed impiegato.

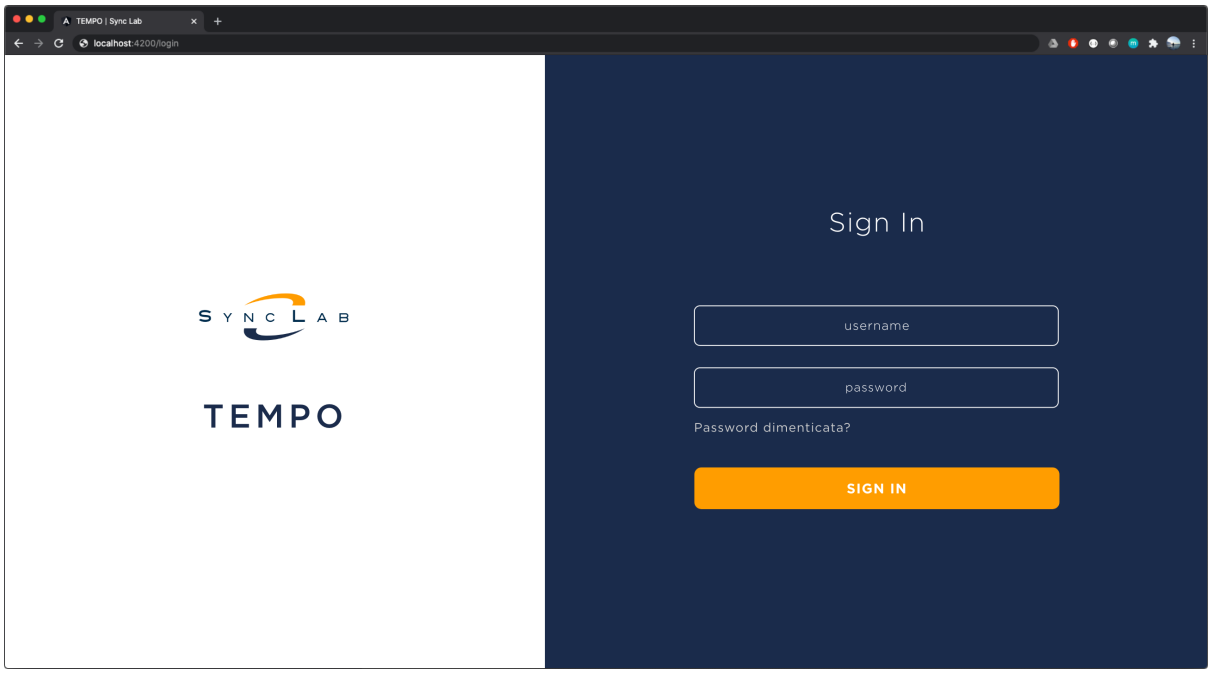

Figura 3.1: Pagina di login.

Osservando le figure 3.2 e 3.3, è possibile notare funzionalità aggiuntive per l'utente manager. Precisamente, in figura 3.2, sono presenti le sezioni "impiegati" e "risorse assegnate", che in figura 3.3 non compaiono. Tali funzionalità consentono al manager, rispettivamente, di gestire gli impiegati appartenenti alla sua sede e di gestire l'assegnamento di risorse a nuovi progetti, nonchè la creazione di nuovi codici commessa. Nella sezione archivio, vengono visualizzati i timesheet relativi ai mesi precedenti e si ha la possibilità di scaricarli in formato Excel.

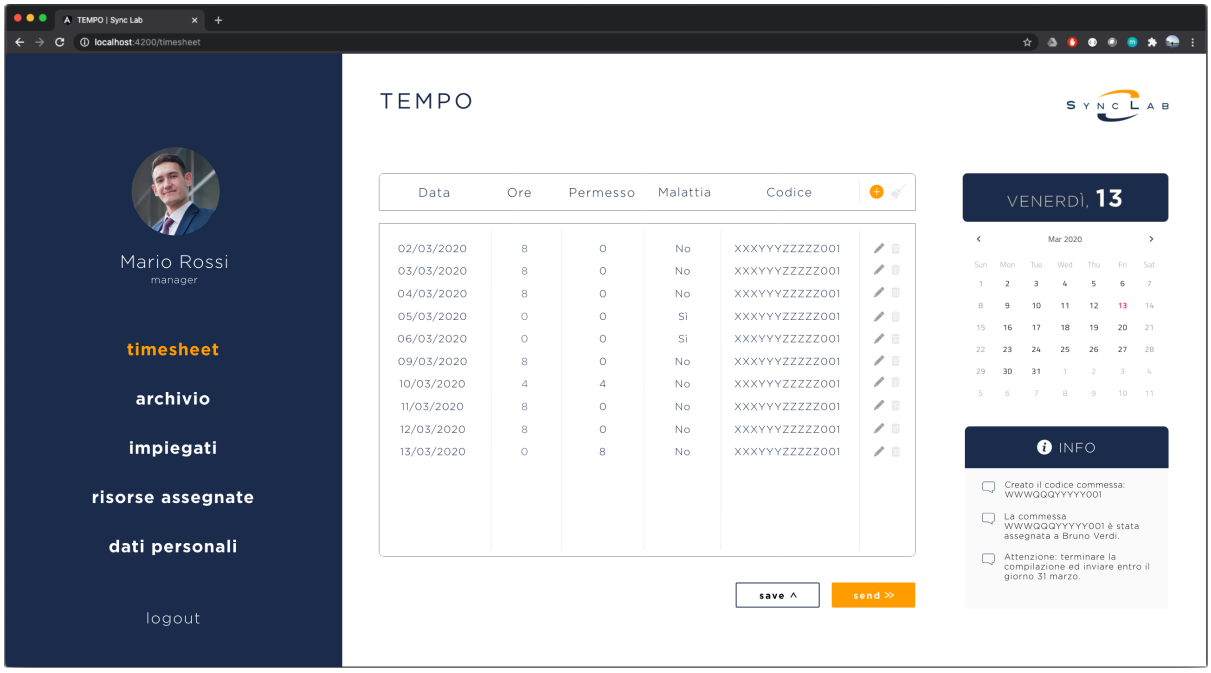

Figura 3.2: Sezione timesheet dell'interfaccia utente di un manager.

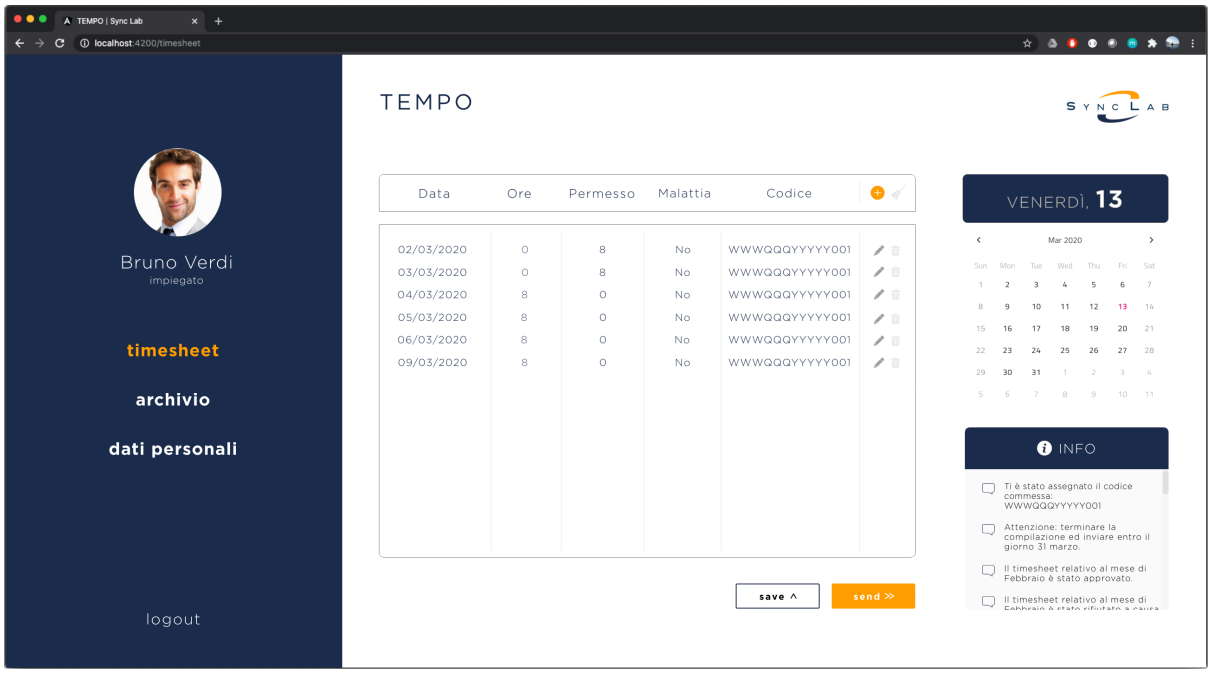

Figura 3.3: Sezione timesheet dell'interfaccia utente di un impiegato.

3.0.3 Server

Per quanto riguarda il server sono stati utilizzati i framework Spring[A.2], in particolare Spring MVC, e Hibernate[A.3], basati sul linguaggio di programmazione Java.

Spring è stato scelto poiché, oltre ad essere uno tra i più popolari framework per lo sviluppo di applicazioni Java Enterprise, permette di organizzare il codice secondo il pattern Model View Controller, citato in precedenza. Tale framework utilizza inoltre, il cosiddetto IoC (*Inversion of Control*), ossia un principio di programmazione che consiste nell'assegnare il controllo del flusso del programma al framework stesso. Questo agevola la creazione di software modulare, riusabile e semplice, eventualmente, da estendere. Oltre a quanto già detto, mediante Spring è stato possibile implementare un'architettura REST grazie all'uso di specifiche annotazioni Java, messe a disposizione dal framework, e definendo inoltre gli endopoint necessari per la localizzazione delle risorse da parte del client, il quale potrà accedervi mediante richieste HTTP.

In alternativa a Spring, si sarebbe potuto adoperare Struts, anch'esso un framework basato sul linguaggio Java e che permette la creazione di applicazioni web enterprise. In Struts, l'oggetto che si occupa di una richiesta e la instrada per l'ulteriore elaborazione è noto come *Action* e sostanzialmente corrisponde a quello che in Spring viene chiamato *Controller*. D'altra parte, Spring MVC è progettato attorno all'oggetto *DispatcherServlet*, ossia un *front controller*, responsabile della delegazione delle richieste agli specifici componenti (*controller*) dell'applicazione. Dunque nel momento in cui si effettua una richiesta, questa viene presa in carico dal DispatcherServlet, il quale a sua volta l'assegnerà al controller di competenza che si preoccuperà dell'elaborazione. La differenza sostanziale in questa fase, sta nel fatto che in Struts viene inizializzato un oggetto Action, ogniqualvolta viene fatta una richiesta, mentre in Spring, i controller vengono inizializzati solo una volta e salvati in memoria, quindi sempre disponibili e condivisi tra le varie richieste, tale gestione offre una maggiore efficienza. Inoltre, un'ulteriore differenza riguarda la flessibilità sulla definizione degli oggetti che compongono l'applicazione: in Spring esiste una separazione dei ruoli, ben definita, tra Controller, Model e View, ma non vengono poste particolari restrizioni sulla definizione degli oggetti; al contrario Struts forza l'uso delle Actions e degli ActionForms. Infine la testabilità di un'applicazione sviluppata mediante Spring, grazie anche al concetto di IoC, risulta più semplice. Per tali ragioni si è deciso di utilizzare Spring Framework.

Per quanto concerne la persistenza dei dati, ossia la loro manipolazione, è stato usato il framework Hibernate. Esso offre una serie di servizi ad alte prestazioni, legati all'interazione tra un'applicazione Java e un database relazionale. Precisamente Hibernate è uno strumento per l'*Object Relational Mapping* (ORM), ossia una tecnica di programmazione per il mappaggio delle classi Java alle tabelle di un database relazionale (e dati tipi Java ai tipi di dato del linguaggio SQL).

In aggiunta, fornisce funzionalità di query e recupero dei dati, in generale per la loro gestione e manipolazione. Questo è possibile poiché esso implementa quelle che sono le specifiche principali per ciò che riguarda la persistenza dei dati, ovvero quanto definito dalla *Java Persistence API* (JPA). I principali vantaggi di Hibernate sono i seguenti:

- open source e di dimensioni contenute;

- alte prestazioni, grazie all'uso di un doppio livello di memoria cache implementate all'interno del framework;
- query indipendenti dal database, questo grazie al linguaggio *Hibernate Query Language* (HQL), ossia una versione orientata agli oggetti del linguaggio SQL;
- creazione automatica delle tabelle del database, dunque non c'è la necessità di doverle creare manualmente.

Nello specifico, il codice è stato strutturato su tre livelli:

- Controller, il quale si occupa di reindirizzare le richieste provenieti dal client e recuperare i dati richiesti usufruendo del livello service.
- Service, contenente la logica di business dell'applicazione.
- Model, costituito da due ulteriori livelli:
  - DAO (*Data Access Object*), il quale contiene i metodi, implementati mediante l'uso del framework Hibernate, per la manipolazione dei dati.
  - Entity, il quale definisce la struttura delle entità che compongono l'applicazione.

3.1 Diagramma delle classi

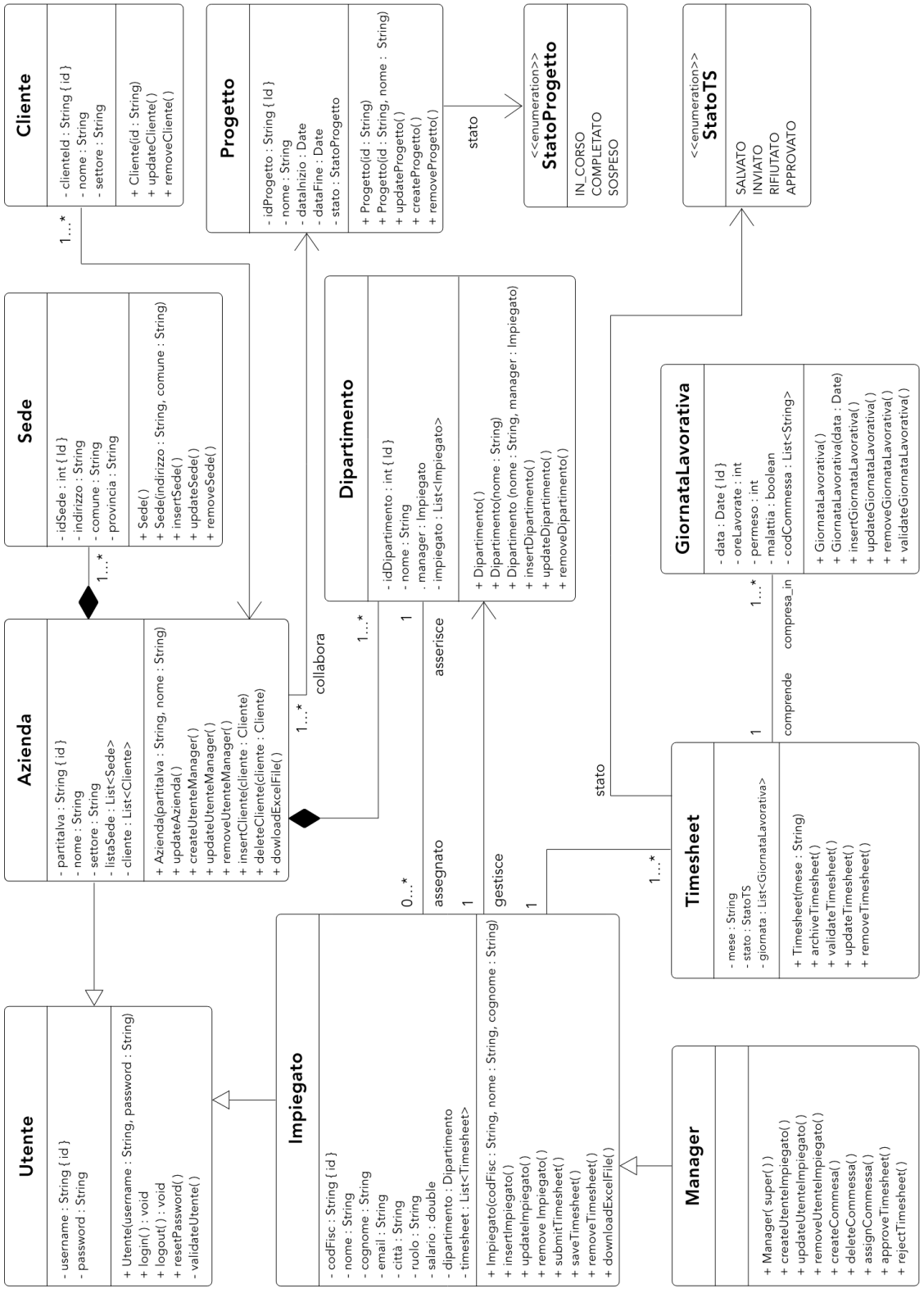

Figura 3.4: Diagramma delle classi.

3.2 Progettazione della base dati

Durante questa fase sono state individuate in primo luogo le entità fondamentali in funzione di quanto emerso nell'analisi dei requisiti. Dunque è stato costruito un diagramma *Entità-Relazione*, il quale rappresenta uno schema concettuale costituito da una serie di costrutti atti a descrivere il dominio preso in considerazione. Specificamente, i rettangoli rappresentano le *entità*, ossia una classe di oggetti del mondo reale (materiali o immateriali), e ognuna di esse possiede degli attributi significativi; i rombi invece, rappresentano una *relazione*, o associazione, la quale è caratterizzata da un nome (e da eventuali attributi se necessari) e che definisce un legame logico tra due o più entità.
Le entità individuate e le rispettive relazioni fra esse, vengono presentate in figura 3.5.

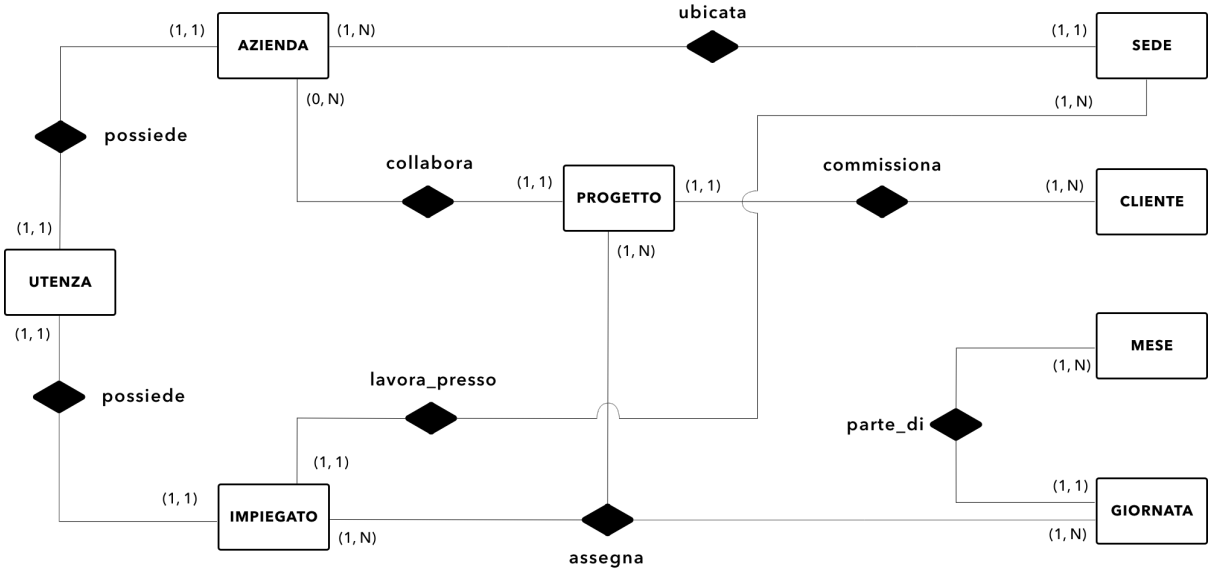

Figura 3.5: Diagramma E-R della base dati.

Successivamente, il diagramma E-R è stato tradotto in *modello relazionale*, ossia uno schema logico, dove il costrutto di base è la *relazione*, corrispondente ad una tabella della base dati. Per ogni relazione, o tabella, le informazioni vengono rappresentate per mezzo di tuple (righe di una tabella), le quali a loro volta sono formate da un insieme di attributi (colonne della tabella) che descrivono la relazione.
Di seguito, in figura 3.6, viene presentato un diagramma che raffigura le tabelle che costituisco la base dati dell'applicazione oggetto di questa relazione.

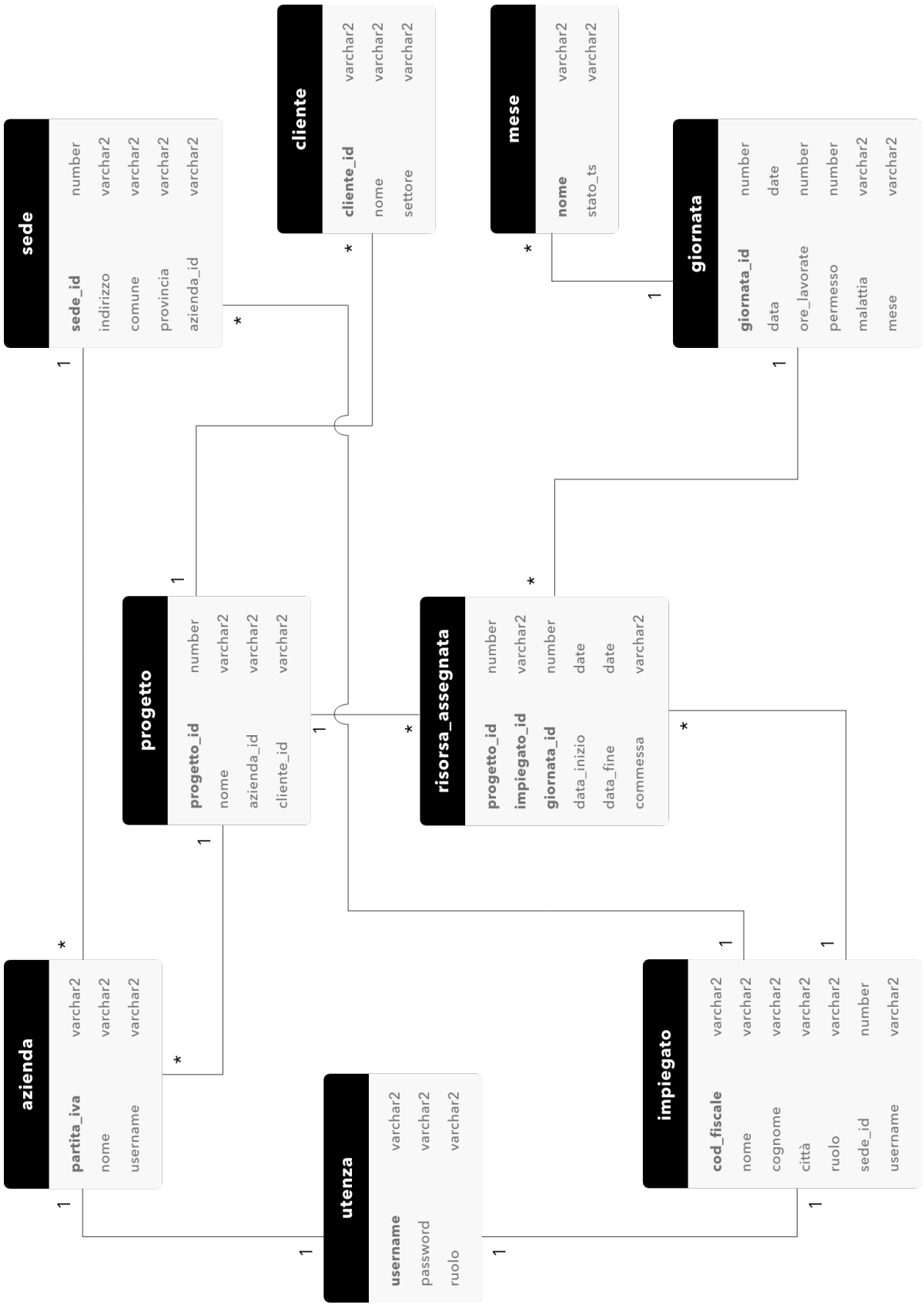

Figura 3.6: Diagramma delle tabelle che costituiscono la base dati.

Per la creazione delle tabelle è stato usato il linguaggio SQL. Nello specifico è stato creato uno script con tutte le istruzioni di DDL (*Data Definition Language*) necessarie per la creazione delle tabelle e per l'assegnamento dei vincoli di integrità. Di seguito uno estratto di tale script:

```
-- creazione della tabella IMPIEGATO senza vincoli di integrita
CREATE TABLE TEMPOAPP.IMPIEGATO (
    COD_FISC        VARCHAR2    NOT NULL,
    NOME            VARCHAR2        NULL,
    COGNOME         VARCHAR2        NULL,
    CITTA           VARCHAR2        NULL,
    EMAIL           VARCHAR2        NULL,
    RUOLO           VARCHAR2        NULL,
    SEDE_ID         NUMBER          NULL,
    USERNAME        NUMBER          NULL
)
/

-- aggiunta dei vincoli di integrita alla tabella IMPIEGATO
ALTER TABLE TEMPOAPP.IMPIEGATO ADD (
    CONSTRAINT IMP_SEDE_FK
    FOREIGN KEY (SEDE_ID)           REFERENCES TEMPOAPP.SEDE(ID),
    CONSTRAINT IMP_UTENTE_FK
    FOREIGN KEY (USERNAME)          REFERENCES TEMPOAPP.UTENTE(ID),
    CONSTRAINT EMAIL_UK
    UNIQUE (EMAIL),
    CONSTRAINT COD_FISC_PK
    PRIMARY KEY (COD_FISC)
);
```

Inoltre, la macchina utilizzata per ospitare tale base dati è un *Autonomous Data Warehouse* (ADW) Oracle, servizio fornito tramite cloud. Questa scelta deriva dalla necessità di avere una base dati Oracle, che fosse compatibile con il sistema operativo MacOS e che non richiedesse l'uso di una macchina virtuale eseguita in locale. Infine, per l'esecuzione degli script, e in generale per tutte le operazioni di interrogazione tramite il linguaggio SQL, è stato utilizzato il software Oracle SQL Developer.

Capitolo 4

Conclusioni e prospettive future

Il tirocinio che ha dato luogo alla creazione del prototipo oggetto di questa relazione è stato molto utile dal punto di vista professionale. L'accesso alla conoscenza di persone con molta esperienza in questo settore è sicuramente importante per la crescita personale, soprattutto dal punto di vista lavorativo. Durante il periodo di stage posso affermare di essere stato supportato in modo eccellente e istruito mediante corsi di formazione inerenti alle tecnologie utilizzate per questo progetto. Al termine del tirocinio, ho avuto e colto l'opportunità di continuare la mia esperienza in quest'azienda.

In conclusione, il prototipo progettato e implementato fornisce una soluzione per l'informatizzazione di un processo che attualmente risulta laborioso, quindi lento, e soggetto ad eventuali errori, che rallenterebbero ulteriormente il procedimento di rendicontazione dell'attività lavorativa di un dipendente.

Gli eventuali sviluppi futuri potrebbero riguardare la sicurezza, in particolare migliorare la gestione dell'autenticazione, per esempio criptando la password dell'utente con un opportuno algoritmo di hashing, diversamente da quanto viene fatto attualmente. Inoltre si potrebbe migliorare la gestione dei permessi di lavoro e malattia, introducendo la richiesta di giustificativi e certificati medici. Infine, sarebbe utile integrare una funzionalità che permette al dipendente di accedere alla sua busta paga, quindi visualizzarla e scaricarla localmente.

Appendice A

Tecnologie Utilizzate

A.1 Angular 8

Angular è un framework di progettazione di applicazioni e una piattaforma di sviluppo per la creazione di app a pagina singola efficienti e sofisticate, utilizzando come linguaggi HTML, CSS e TypeScript.

TypeScript è un linguaggio di programmazione open-source, sviluppato e mantenuto da Microsoft. È un soprainsieme di JavaScript (JS) a livello sintattico, con la differenza che è possibile utilizzare la tipizzazione statica per i dati.

Le funzionalità centrali e opzionali vengono implementate come set di librerie TypeScript, che possono essere importate nel progetto che si vuole creare. L'architettura di un'applicazione Angular, si basa su quattro concetti fondamentali, descritti di seguito.

A.1.1 Modulo

Un NgModule fornisce un contesto di compilazione per un insieme di componenti dedicato a un determinato dominio dell'applicazione, un workflow oppure un insieme di funzionalità strettamente correlate. Un NgModule può associare i suoi componenti al codice correlato, come i servizi, per formare delle unità funzionali. Un'applicazione Angular ha almeno un modulo radice, il quale è indispensabile per il suo avvio, e poi possiede altri diversi moduli ognuno con il proprio contesto.

A.1.2 Componente

Ogni applicazione Angular ha almeno un componente, quello radice, che collega una gerarchia di componenti con il DOM (*Document Object Model*) della pagina. Ciascun componente definisce una classe che contiene dati e logica dell'applicazione ed è associato a un modello HTML (e CSS) che definisce il livello di vista da visualizzare in un ambiente target, solitamente un browser web.

A.1.3 Service e dependency injection

Per i dati o la logica non associati ad una vista specifica e che si desidera condividere tra i componenti, si crea una classe service. Tali servizi possono essere iniettati all'interno dei componenti desiderati mediante il concetto di *dependency injection*, il quale consente di mantenere snelle ed efficienti le classi dei componenti e dunque rendendo il codice modulare, riutilizzabile e performante.

A.1.4 Routing

Il modulo router di Angular fornisce un servizio che consente di definire un percorso di navigazione tra i diversi stati dell'applicazione e visualizzarne le gerarchie. È modellato sulle convenzioni di navigazione del browser classiche, ossia l'inserimento di una URL nella barra degli indirizzi, click a collegamenti nella pagina, click sulle icone "indietro" e "avanti" del browser. Sostanzialmente il router esegue il mapping di percorsi simili ad una URL sulle viste (componenti), anziché direttamente sulle pagine. Quando un utente esegue un'azione, ad esempio facendo click su un collegamento, che carica una nuova pagina nel browser, il router intercetta il comportamento del browser e mostra o nasconde le gerarchie di viste.
Per definire le regole di navigazione, è sufficiente associare i percorsi di navigazione ai vari componenti. Un percorso utilizza una sintassi simile a un URL. È quindi possibile applicare la logica del programma per scegliere quali viste mostrare o nascondere, in risposta all'input dell'utente e alle proprie regole di accesso.

A.2 Spring Framework

Spring Framework fornisce un modello di programmazione e configurazione completo per le moderne applicazioni enterprise basate su Java, su qualsiasi tipo di piattaforma di distribuzione.

Un elemento chiave di Spring è il supporto infrastrutturale a livello di applicazione: Spring si concentra sulla gestione dell'infrastruttura delle applicazioni enterprise, in tal modo è possibile concentrarsi sulla logica di business sul livello applicazione, senza inutili legami con specifici ambienti di distribuzione.

A.2.1 Caratteristiche

Le principali caratteristiche di tale framework sono le seguenti:

- Spring consente di sviluppare applicazioni enterprise utilizzando i cosiddetti POJO (*Plain Old Java Object*). Il cui ulteriore vantaggio è che non è necessario un contenitore EJB (*Enterprise Java Bean*) come un application server, ma si ha la possibilità di utilizzare solo un contenitore servlet robusto come Tomcat o altri prodotti simili commerciali.

- Spring è organizzato in moduli. Anche se il numero di pacchetti e classi è notevole, è sufficiente preoccuparsi di quelli necessari per l'esecuzione dell'applicazione.
- Si integra facilmente con framework già esistenti.
- Gestione centralizzata delle eccezioni. Spring fornisce un'API conveniente per tradurre eccezioni specifiche per la tecnologia (generate ad esempio da JDBC, Hibernate o JDO) in eccezioni coerenti e non controllate.
- Dimensioni ridotte.
- Sfrutta il concetto di *dependency injection* e *Inversion of Control*.

A.3 Hibernate Framework

Hibernate è un framework Java che agevola lo sviluppo dell'applicazione per quanto riguarda l'interazione con il database. È uno strumento per l'ORM (Object Relational Mapping), open source e di dimensioni ridotte. Esso implementa le specifiche di JPA (Java Persistence API), una libreria Java per la persistenza dei dati.

A.3.1 Object Relational Mapping

Un'ORM è uno strumento che semplifica la creazione, la manipolazione dei dati e l'accesso ad essi. È una tecnica di programmazione utile per mappare gli oggetti di un linguaggio di programmazione come Java, nei dati memorizzati all'interno di tabelle di una base dati relazionale.

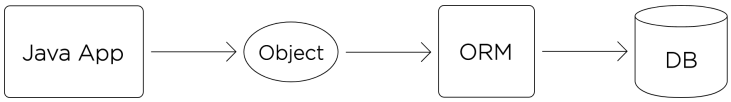

Figura A.1: Schema di alto livello di un'applicazione che sfrutta un ORM per l'interazione con un database relazionale.

Nel caso di Hibernate, quindi dell'utilizzo del linguaggio Java, viene utilizzato la JDBC (*Java Database Connectivity*) API per permettere l'interazione con la base dati.

A.3.2 Architettura del framework

L'architettura di Hibernate coinvolge diversi tipi di oggetti, quali un oggetto di persistenza (da memorizzare), session factory, transaction factory, connection factory, session, transaction, ecc.
In generale possiamo categorizzarla in quattro livelli:

- Applicazione Java.

- Framework Hibernate.
- Back-end API
- Base dati

Quanto appena detto, viene presentato in figura A.2

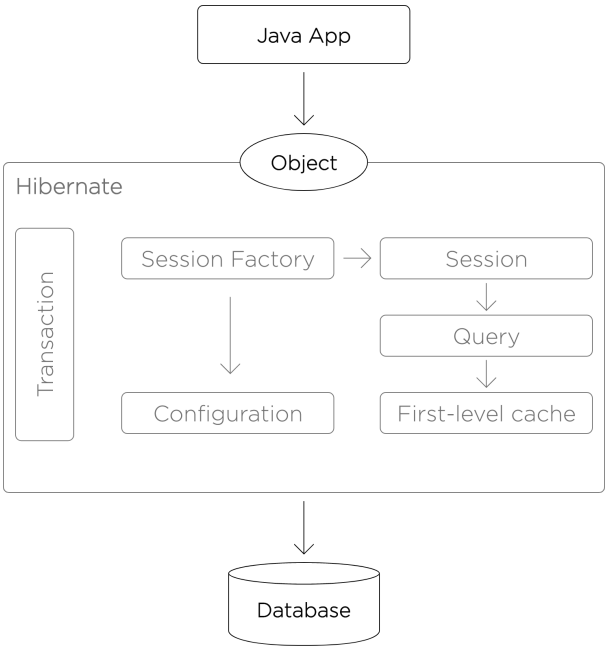

Figura A.2: Schema di alto livello di un'applicazione che adopera Hibernate per l'interazione con un database relazionale.