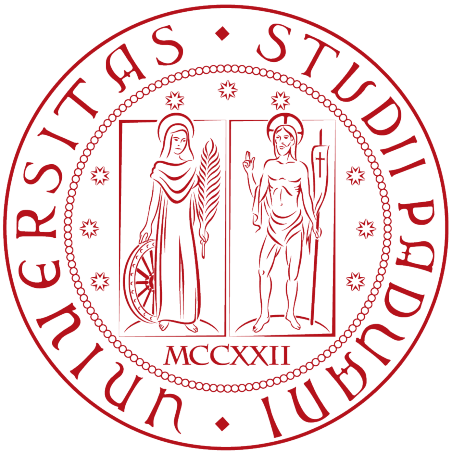
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Una estensione Chrome per la gestione di attività lavorative

Tesi di laurea triennale

Relatore

Prof.Tullio Vardanega

Laureando

Fincato Alessandro

Anno Accademico 2021-2022

Per la mia famiglia
e per me stesso.

Sommario

Il presente documento fornisce una descrizione del lavoro svolto durante il periodo di *stage*, della durata di circa trecentodieci ore, dal laureando Fincato Alessandro presso l'azienda Wavelop Srls.
Il progetto prevedeva l'analisi, la progettazione e l'implementazione di una estensione per *Google Chrome* che permetta la pianificazione e la consuntivazione di attività lavorative svolte dai dipendenti. Il tutto utilizzando tecnologie attuali come *ReactJS*, *TypeScript*, *HTML5* e *CSS*.

Ringraziamenti

Per prima cosa, vorrei esprimere la mia gratitudine al Professore Tullio Vardanega, relatore della mia tesi, per l'importante supporto fornitomi durante la stesura del lavoro.

Ringrazio con affetto la mia famiglia per il grande supporto in questi anni e per aver sempre creduto in me nonostante tutto.

Infine, ringrazio tutti i miei amici per le esperienze vissute insieme senza le quali non sarei quello che sono.

Padova, Dicembre 2022 Fincato Alessandro

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Contesto aziendale

Questo capitolo tratta dell'azienda nella quale ho svolto lo stage, Wavelop Srls. Lo scopo è esporne la mission, il dominio applicativo e l'organizzazione interna, compresa di tutti i processi e le tecnologie utilizzate in maniera trasversale nella produzione di software.

1.1 L'azienda

Wavelop Srls nasce con l'obiettivo di progettare e sviluppare soluzioni *web* e *mobile* lavorando a stretto contatto con il cliente ed utilizzando le migliori tecnologie disponibili nel mercato. Sia il cliente una *startup* o una grande impresa, cerca sempre di mettere l'utente al primo posto, creando *design user-centered* che possano dare la miglior *user-experience*.
Alla base della filosofia aziendale vi sono la comunicazione con il cliente, la metodologia *Agile Scrum* e l'utilizzo delle tecnologie più recenti. La stessa origine del nome emerge dalla volontà di rimanere costantemente aggiornati ed aperti alle innovazioni tecnologiche. Questo permette di lavorare ai progetti più disparati, studiando le soluzioni migliori e applicando i *design pattern* più adatti, in modo da produrre del *software* di alta qualità, facile da mantenere ed estendere.

1.2 Organizzazione aziendale

Le figure che si trovano all'interno dell'azienda sono:

* **Amministratore**: ruolo ricoperto dai *co-founder* dell'azienda, la sua responsabilità è mantenere un rapporto costante con i clienti e gestire il gruppo di sviluppatori. Nello specifico deve capire le richieste del cliente, cercando di coinvolgerlo il più possibile durante l'intero progetto, e gestire ed aiutare il *team* nell'implementazione della metodologia *Scrum*, pianificandone il lavoro nel modo più efficiente possibile;

* **Sviluppatore**: il compito principale di ogni sviluppatore è collaborare, seguendo quanto pianificato con gli amministratori, al fine di produrre un prodotto di qualità che rispetti le richieste fatte dal cliente e dagli *stakeholders*. All'interno dell'azienda si trovano diverse figure di sviluppatori:

  - **Full stack developer**: è la figura maggiormente presente nell'azienda, si tratta di sviluppatori con una conoscenza generale di tecnologie diverse, che possono progettare sia la parte *server* che *client* di un prodotto *software*;
  - **Frontend developer**: sviluppatore specializzato soprattutto nelle tecnologie relative alla parte *frontend* del prodotto, ovvero tutto ciò che fa riferimento alla parte *client*.

Tale organizzazione interna è rispecchiata dai ruoli all'interno della metodologia *Scrum*, adottata dall'azienda. Questi sono:

* **Product owner**: colui che ha il compito di capire le richieste del cliente e degli *stakeholders*, assegnarci un livello di priorità a seconda del valore di mercato e prendere decisioni necessarie al completamento del progetto e alla gestione del *backlog*. Questo ruolo viene svolto dagli amministratori;

* **Scrum master**: colui che ha il compito di aiutare il *team* ad implementare lo *Scrum*, è responsabile del raggiungimento degli obiettivi e degli eventi *Scrum*. Anche questo ruolo viene svolto dagli amministratori;

* **Development *team***: è l'insieme degli sviluppatori che collaborano per produrre del *software* di qualità che rispetti i requisiti stabiliti. Questo ruolo è svolto dagli sviluppatori assegnati ad un gruppo che lavora su un determinato progetto.

1.3 Processi aziendali

1.3.1 Modello di ciclo di vita del *software*

Per lo sviluppo di prodotti *software* l'azienda utilizza una metodologia *Agile* basata su *Scrum*. I modelli di sviluppo agili si basano principalmente su quattro aspetti chiave:

* Dare priorità a del *software* efficiente anziché a della documentazione esaustiva;

* Preferire una collaborazione con il cliente anziché una negoziazione dei contratti;

* Valorizzare maggiormente gli individui e le interazioni anziché i processi e gli strumenti;

* Essere sempre pronti ad affrontare un cambiamento.

Il *framework Scrum* è stato ideato da Ken Schwaber e Jeff Sutherland con l'obiettivo di rendere il lavoro dei *team* altamente produttivo e funzionante nella consegna di valore per il cliente finale. *Scrum* è un *framework* euristico: si basa sull'apprendimento continuo e l'adattamento a fattori variabili [1]. L'idea di fondo è che il *team* non dispone di tutte le conoscenze sin dall'inizio e per questo non farà altro che evolversi man mano che acquisisce esperienza. *Scrum* è strutturato per aiutare i *team* ad adattarsi in modo naturale alle condizioni ed esigenze di clienti e *stakeholders* che possono mutare nel tempo, con una ridefinizione delle priorità e cicli di rilascio brevi, che consentono al *team* di perseguire un miglioramento continuo durante l'intero avanzamento del progetto. I tre aspetti chiave alla base del *framework* sono:

[1] Drumond, C. (s.d.). "Scrum: che cos'è, come funziona e perché è strepitoso". Atlassian. https://www.atlassian.com/it/agile/scrum

* **Trasparenza**: richiede che tutte le persone coinvolte nel progetto ne conoscano lo scopo e possano sapere a cosa stanno lavorando gli altri componenti. Ciò implica che il lavoro e le relative misure di *performance* siano visibili a tutti a qualsiasi livello organizzativo;

* **Ispezione**: implica che, successivamente ad un incremento o ad una iterazione, vengano svolte le relative valutazioni rispetto alle metriche di misurazione decise e stabilite a priori;

* **Adattamento**: è la conseguenza dell'ispezione, significa che il *team* di sviluppo deve ri-pianificare l'avanzamento del progetto in base ai risultati dell'ispezione in modo da apportare un miglioramento continuo delle proprie *performance*;

Affinché l'utilizzo di *Scrum* funzioni, vi è la necessità di formare il *Team Scrum* associando tra di loro ruoli, eventi, regole e artefatti. Ognuno di questi aspetti ha uno scopo ben preciso e ricopre un ruolo essenziale per il successo dell'utilizzo del *framework*. Le regole legano insieme gli eventi, mentre i ruoli e gli artefatti governano le relazioni e le interazioni tra essi, pur essendo queste variabili a seconda del progetto.

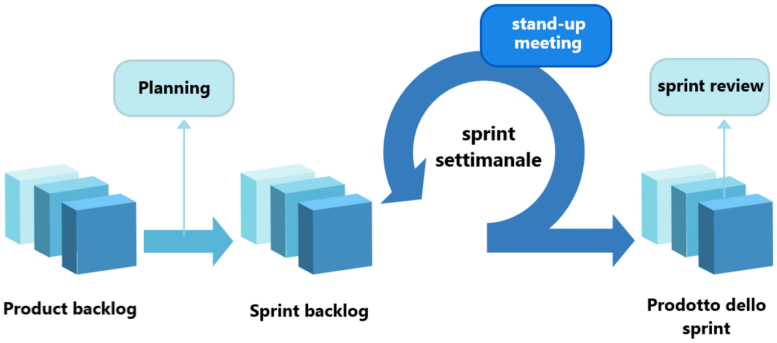


Figura 1.1: Processo *Scrum*, tratto da: "*Agile Scrum Process Flow Diagram*", Slideegg.

L'unità base temporale dello sviluppo all'interno di *Scrum* è lo *sprint*. Questo è un periodo di durata fissa che va da una a quattro settimane a seconda delle esigenze ed al modo di lavorare del *team*. Nel caso dell'azienda, è stato deciso di adottare degli *sprint* della durata di una settimana.
Facendo riferimento alla Figura 1.1, che rappresenta gli elementi e le attività che compongono uno *sprint* in Wavelop Srls, possiamo notare come ognuno sia preceduto da una riunione di pianificazione. Questo è un evento collaborativo durante il quale il *team* risponde a due domande fondamentali: a cosa serve lo *sprint* e come verrà svolto il lavoro. La scelta degli elementi di lavoro giusti per uno *sprint* è frutto di un impegno collaborativo, viene discusso l'obiettivo da raggiungere e gli elementi del *backlog* di prodotto che devono essere completati per raggiungere tale obiettivo. Successivamente, il *team* esegue una stima temporale dei vari elementi e definisce il *backlog* dello *sprint*, che contiene tutti gli elementi del *backlog* da implementare in quel determinato periodo di tempo. Alla fine della pianificazione dello *sprint*, il *team* è pronto a iniziare a lavorare sul *backlog* di quest'ultimo.
Durante lo *sprint*, quotidianamente, il *team* esegue uno *stand-up meeting* per parlare

dell'avanzamento del lavoro. Il fine principale di questo evento è di far emergere eventuali difficoltà e problemi bloccanti che compromettono la capacità del *team* di raggiungere gli obiettivi stabiliti.
Al termine di ogni *sprint* viene eseguita la riunione di revisione e retrospettiva dell'avanzamento. Nella prima parte di tale riunione vi è l'opportunità di esporre quanto è stato implementato durante lo *sprint*, successivamente avviene un'analisi di quest'ultimo in modo da poterne identificare i lati positivi, da mantenere, e quelli meno positivi, da migliorare. Lo sviluppo è di durata fissa in modo da poter terminare lo *sprint* alla data prefissata, reinserendo eventuali elementi non completati nel *backlog* di prodotto.

I principali vantaggi riscontrati dall'azienda nell'utilizzo del *framework Scrum* per lo sviluppo *software* sono:

* Adattabile alle diverse esigenze specifiche del *team*;
* Semplifica la pianificazione dello sviluppo di grandi progetti;
* Permette di garantire qualità sin dall'inizio;
* Lo sviluppo viene basato sulla comunicazione continua con clienti e *stakeholders*;
* Permette di affrontare eventuali cambiamenti in maniera più semplice.

Di conseguenza vi sono però anche dei lati negativi nell'utilizzo di *Scrum*, che sono stati presi in considerazione nella scelta del modello di sviluppo da utilizzare e che sono controllati durante l'intero avanzamento del progetto:

* Difficoltà nel definire una data precisa per il termine del progetto;
* Forte necessità di collaborazione da parte dei componenti del *team*.

1.3.2 Strumenti a supporto dei processi

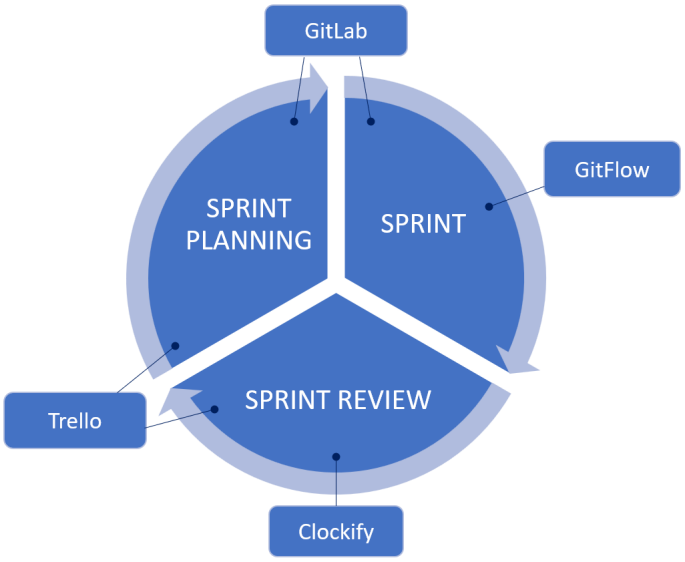


Figura 1.2: Strumenti a supporto dei processi nel flusso di lavoro.

Wavelop Srls ha definito vari strumenti e norme che svolgono un ruolo di supporto nelle diverse attività di progetto. Questi, facendo riferimento alla Figura 1.2, sono integrati nel flusso di lavoro e perciò permettono sia di adottare una metodologia *Agile Scrum* che di svolgere in modo analogo le diverse attività di gestione e organizzazione in ogni progetto. Tali strumenti riguardano infatti vari aspetti trasversali del flusso di lavoro, ovvero:

* **Versionamento**: per fissare una metodologia di sviluppo e di gestione del codice comune per tutti i progetti. Di supporto alla fase di sviluppo;
* **Framework Scrum**: per fissare uno strumento comune a tutti i progetti per la gestione dei *task* e l'applicazione del *framework Scrum*. Di supporto soprattutto alle attività di pianificazione dello *sprint* e di organizzazione del lavoro durante quest'ultimo;
* **Pianificazione**: per stabilire lo strumento da utilizzare e gli obiettivi da perseguire nella pianificazione generale dei progetti e per il controllo di eventuali criticità. Di supporto durante la pianificazione e la revisione dello *sprint*;
* **Rendicontazione**: per definire gli strumenti e le norme, comuni per ogni progetto, necessarie per la rendicontazione delle ore di lavoro. Svolge un ruolo di supporto soprattutto per la revisione dello *sprint*.

1.3.2.1 Versionamento

Come sistema di versionamento l'azienda utilizza *Git*, un sistema *software* di controllo di versione distribuito, utilizzabile direttamente da linea di comando, creato da Linus Torvalds. Per gestire le *repository Git* relative ai diversi progetti ai quali lavora, viene utilizzata la piattaforma *web open source GitLab*. La scelta è ricaduta su tale servizio soprattutto grazie alla possibilità di unire la gestione del versionamento e la pianificazione di parte delle attività di progetto all'interno della stessa piattaforma, rendendo così il lavoro più efficiente e meno complesso.

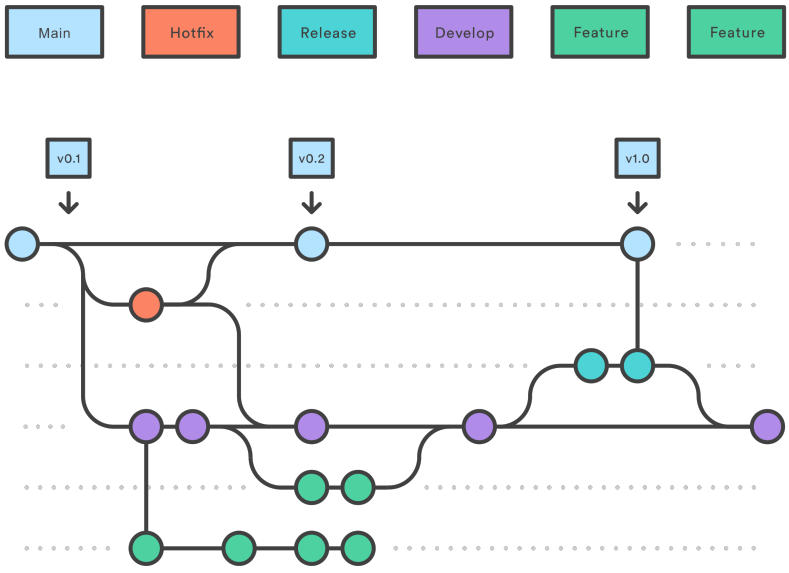


Figura 1.3: "*Gitflow Workflow | Atlassian Git Tutorial*", Atlassian.

Per l'organizzazione del versionamento, Wavelop Srls adotta come modello di *branching Gitflow* (Figura 1.3), che permette di avere una storia implementativa più comprensibile e pulita. Tale modello di *branching* consiste nel creare, a partire dal ramo principale dello sviluppo, un *branch* per ogni nuova *feature* del prodotto che si vuole implementare, ritardandone così il *merging* finché non è conclusa e verificata totalmente.
La differenza principale rispetto agli altri modelli di sviluppo è che assegna un ruolo specifico ai diversi *branch* e definisce quando e come questi debbano interagire tra di loro. Di fatto, oltre al *branch* di sviluppo, dal quale partono i vari *feature branch*, vi sono altri *branch* con scopi ben specifici, ovvero:

* **Produzione**: contiene la versione corrente del codice;
* **Aggiornamento rapido**: utilizzato nel caso in cui si renda necessario realizzare *patch* della versione in produzione, non è necessario modificare il ramo di sviluppo, ma è sufficiente creare un nuovo *branch* direttamente dal ramo di produzione;
* **Release**: utilizzato ogni volta che una nuova *release* del codice è pronta per essere integrata all'interno del *branch* di produzione. Questo *branch* parte dal ramo di sviluppo ed ha lo scopo di preparare il codice implementato per essere pubblicato, successivamente viene unito sia al *branch* di produzione che a quello di sviluppo per indicare che è stato distribuito.

1.3.2.2 Framework Scrum

A supporto della metodologia agile, Wavelop Srls ha scelto di utilizzare la piattaforma *GitLab* per l'implementazione del *framework Scrum*. Tale piattaforma permette diverse funzionalità utili nell'ambito del *project managment*:

* Definire i *task* da inserire nel *backlog* del prodotto;
* *Issue* e *Bug tracking*;
* Creare *feature branch*, relativi a specifici *task*, ai quali faranno riferimento durante lo sviluppo.

Inoltre, i vantaggi dati dall'utilizzo di questo strumento sono:

* La pianificazione del progetto diventa più chiara ed efficiente;
* Disponibilità di una comoda interfaccia grafica *drag and drop*.

1.3.2.3 Pianificazione

Per la pianificazione del'organizzazione del *team* l'azienda utilizza *Trello*, un *software* gestionale in stile *Kanban* basato sul *web*. Gli obiettivi dietro all'utilizzo di questo strumento sono:

* Esporre eventuali criticità su progetti;
* Esporre eventuali criticità personali;
* Punto della situazione aziendale;
* Pianificare eventuali migliorie, progetti o opportunità.

Questa riunione viene svolta una volta a settimana in modo da garantire trasparenza riguardo l'andamento generale del lavoro.

1.3.2.4 Consuntivazione

Per la consuntivazione Wavelop Srls utilizza due strumenti diversi: *Clockify* e *GitLab*. Il primo è quello principale, utilizzato in qualsiasi progetto e generalmente da solo. Si tratta di un *software* di monitoraggio del tempo che consente il tracciamento delle ore di lavoro che un componente del *team* ha dedicato per ciascuna attività di progetto alla quale ha lavorato. Nello specifico, per ogni progetto va indicata l'attività che si è svolta e le ore di lavoro impiegate. Successivamente, al termine della settimana di lavoro, si avrà un *report* che indica quante ore sono state utilizzate per ogni progetto e per quali attività.
In alcuni progetti può essere richiesto di rendicontare anche le ore che un componente ha dedicato per ogni specifico *task*. Per fare ciò viene utilizzata la funzionalità di tracciamento del tempo presente direttamente su *GitLab*, che consente di inserire tale informazione come campo per ogni *task*.

1.4 Tecnologie utilizzate

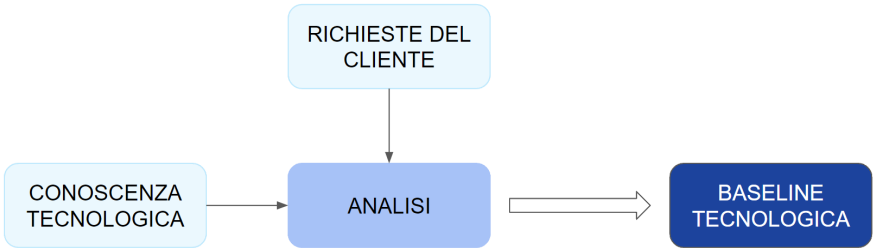


Figura 1.4: Flusso di selezione delle tecnologie

Questa sezione presenta le tecnologie che costituiscono la conoscenza tecnologica aziendale. Di fatto non vengono utilizzate tutte in ogni progetto, bensì vengono scelte le più adatte per ogni specifico caso. Come illustrato nella Figura 1.4, a seguito delle richieste del cliente e di una prima fase di analisi, vengono decise le tecnologie più adatte allo sviluppo del prodotto richiesto stabilendone così la *baseline* tecnologica, ovvero l'insieme di base delle tecnologie che verranno utilizzate durante il progetto. Successivamente a tale scelta si potrà iniziare ad analizzare, progettare e sviluppare la soluzione ottimale al problema del cliente.

1.4.1 React

React.js è una libreria *JavaScript* sviluppata da *Facebook*. Il suo scopo è permettere agli sviluppatori di creare in modo facile e veloce delle applicazioni *web*. Ogni prodotto che utilizza *React* è composto da componenti riutilizzabili che costituiscono parti dell'interfaccia utente. Avere questi componenti riutilizzabili facilita lo sviluppo perché non dobbiamo ripetere il codice, ma solamente creare la sua logica e importare il componente dove è necessario. *React* è anche un'applicazione a pagina singola. Quindi, invece di inviare una richiesta al *server* ogni volta che una nuova pagina deve essere renderizzata, il contenuto di questa viene caricato direttamente dai componenti *React*. Questo porta a un *rendering* più veloce senza ricaricare la pagina.

1.4.2 Angular

Angular è un *framework Javascript open source*, basato su *HTML* e *TypeScript*. Lanciato e mantenuto da *Google*, il suo scopo principale è creare applicazioni a pagina singola. Fornisce funzionalità fondamentali e aggiuntive sotto forma di un insieme di librerie *TypeScript* che vengono caricate nelle applicazioni. Inoltre, permette agli utenti di costruire applicazioni scalabili e facilmente gestibili.
A seguito di continue migliorie che l'hanno reso sempre più utile per diversi progetti *web* è stato creato *Angular 2.0*, che presenta molte nuove funzionalità ed elementi in aggiunta ai vantaggi già esistenti di *AngularJS*. Questa nuova versione di *Angular* è stata creata da zero per eliminare molti limiti e difetti del vecchio *AngularJS*.

1.4.3 Vue

Vue è un *framework open source JavaScript* che fornisce potenti strumenti *web* per lo sviluppo di interfacce utente e applicazioni a pagina singola. Utilizza un pattern architetturale *model-vie-viewmodel* (MVVM) ed è riconosciuto come *framework JavaScript* dinamico e progressivo, in quanto permette di creare interfacce utente *progressive*, che consentono la modifica del codice dell'applicazione senza impattare su alcuna funzionalità essenziale. La notevole flessibilità di *Vue* consente di aggiungere all'applicazione *web* moduli e componenti visivi personalizzati.

1.4.4 MongoDB

MongoDB è un *database NoSQL open source*. In quanto *database* non relazionale, è in grado di elaborare dati strutturati, semi-strutturati e non strutturati. Utilizza un modello di dati non relazionale e orientato ai documenti e un linguaggio di *query* non strutturato. Si tratta di una tecnologia altamente flessibile che consente di combinare e memorizzare più tipi di dati. Inoltre, memorizza e gestisce quantità di dati maggiori rispetto ai *database* relazionali tradizionali. *MongoDB* utilizza un formato di *storage* dei documenti definito *BSON*, che è una forma binaria di *JSON* (*JavaScript Object Notation*) in grado di ospitare più tipi di dati.Gli oggetti di dati sono memorizzati in raccolte e documenti, invece che nelle tabelle e righe utilizzate nei *database* relazionali tradizionali. Le raccolte comprendono insiemi di documenti, che sono l'equivalente delle tabelle di un *database* relazionale. I documenti sono costituiti da coppie chiave-valore, che sono l'unità di base dei dati in *MongoDB*.

1.4.5 Node.js

Node.js è un ambiente di esecuzione che permette di eseguire codice *Javascript* come un qualsiasi linguaggio di programmazione. Facendo ciò può essere utilizzato per svolgere qualsiasi tipo di programma: da elaborazioni statistiche o scientifiche a interazioni con rete e *database*, fino all'utilizzo come *server*. Si tratta di un prodotto *open source*, gratuito e multipiattaforma, utilizzabile su qualsiasi sistema operativo, e seguito da una delle più folte e attive comunità di sviluppatori al mondo. Inoltre *Node.js* si basa su un meccanismo di reazione agli eventi che gli permette, al contempo, di consumare poche risorse ed essere opportunamente reattivo quando serve.

1.4.6 Java

Java è un linguaggio di programmazione orientato agli oggetti, basato sulle classi e tipizzato staticamente. Viene utilizzato per lo sviluppo *web* ed in particolare per le applicazioni *web client-server*. *Java* è progettato per avere il minor numero possibile di dipendenze di implementazione, per consentire agli sviluppatori di "scrivere una volta, eseguire ovunque". Infatti il codice è multipiattaforma e, una volta eseguito su una, non ha bisogno di essere ricompilato per essere eseguito su un'altra.
Oltre ad essere il linguaggio di programmazione, *Java* fa anche riferimento all'intero ecosistema che ci gravita attorno, composto da tre componenti fondamentali:

* **Java Virtual Machine (JVM)**: ambiente di esecuzione virtuale, indipendente dalla piattaforma, che converte il *bytecode Java* in linguaggio macchina e lo esegue;
* **Java Runtime Environment (JRE)**: ambiente *runtime* necessario per eseguire programmi e applicazioni *Java*;
* **Java Development Kit (JDK)**: componente principale dell'ambiente *Java* che contiene *JRE* insieme al compilatore *Java*, al *debugger Java* e ad altre classi.

1.4.7 TypeScript

TypeScript è un linguaggio di programmazione *open source* sviluppato da *Microsoft*. Nello specifico, è un *superset* di *JavaScript*, che aggiunge tipi, classi, interfacce e moduli opzionali al *JavaScript* tradizionale. Si tratta di un linguaggio tipizzato, ovvero aggiunge definizioni di tipo statico: i tipi consentono di descrivere la forma di un oggetto, documentandolo meglio e consentendo a *TypeScript* di verificare che il codice funzioni correttamente. Uno dei punti di forza di *TypeScript*, che lo differenzia dai concorrenti, è il fatto di essere un *superset*, di conseguenza qualsiasi codice scritto in *JavaScript* è anche compatibile con la sintassi e la semantica *TypeScript*.

1.4.8 PHP

PHP è un linguaggio di *scripting open source* interpretato lato *server*, usato per lo più nello sviluppo *web*, ma può essere usato anche per scrivere *script* da riga di comando o applicazioni *stand-alone* con interfaccia grafica. *PHP* sta per *PHP: Hypertext Preprocessor*. Si tratta di un linguaggio fondamentalmente di alto livello a tipizzazione debole e che, dalla versione 5, supporta il paradigma di programmazione ad oggetti.È stato progettato con lo scopo di creare pagine *web* dinamiche, integrate efficacemente con i *database*. Inoltre, può essere incorporato direttamente all'interno del codice *HTML* di una pagina *web*. Queste caratteristiche rendono *PHP* un linguaggio estremamente diffuso nella programmazione *web*, e parte integrante di *stack* come *LAMP*, *WAMP* e *MAMP*.

1.4.9 AWS

Amazon Web Services, Inc. (nota con la sigla AWS) è un'azienda statunitense di proprietà del gruppo *Amazon*, che fornisce servizi di *cloud computing* su un'omonima piattaforma *on demand*. Offre oltre 200 prodotti, fornendo soluzioni con caratteristiche di *high availability*, ridondanza e sicurezza, in cui il costo finale deriva dalla combinazione di tipo e quantità di risorse utilizzate, caratteristiche scelte dall'utente, tempo di utilizzo e performance desiderate.

1.4.10 Ionic

Ionic è un *framework HTML5 open source*, usato per scrivere applicazioni *mobile* ibride con tecnologie *web* come *HTML*, *JavaScript*, *CSS* e *SASS*. Permette di creare applicazioni *web progressive* (PWAs, *Progressive Web Apps*) multipiattaforma. Questa tecnologia offre una vasta libreria *front end* di componenti per l'interfaccia grafica, ottimizzati per *mobile* e compatibili con qualsiasi *framework JavaScript*, come *Angular*, *React* e *Vue*. *Ionic* presenta inoltre una interfaccia a riga di comando (*Ionic CLI*) funzionale alla attività di creazione, *testing* e distribuzione delle *app*. Il *team Ionic* ha inoltre elaborato un proprio *runtime* di *app* multipiattaforma, chiamato *Capacitor*, come alternativa a *Cordova*, che invece era stato alla base della prima versione di *Ionic*.

1.4.11 Organizzazione delle tecnologie all'interno del prodotto

Lo scopo principale di Wavelop Srls è sviluppare soluzioni *web* e *mobile* utilizzando tecnologie innovative e che rispettino le esigenze dell'utente finale. Le tecnologie principali utilizzate per la creazione di *web app* e *mobile* sono:

* *React, Angular* oppure *Vue*;
* *Ionic*;
* *Java*;
* *TypeScript*;
* *Node.js*.

In alcuni casi è necessario sviluppare anche un sistema di supporto a queste tecnologie, che interagisca con esse, così da andare a creare un dualismo di tipo *client/server*. Per creare le applicazioni *backend*, che si occupano della gestione ed elaborazione dei dati, Wavelop Srls utilizza:

* *MongoDB*;
* *PHP*;
* *AWS*.

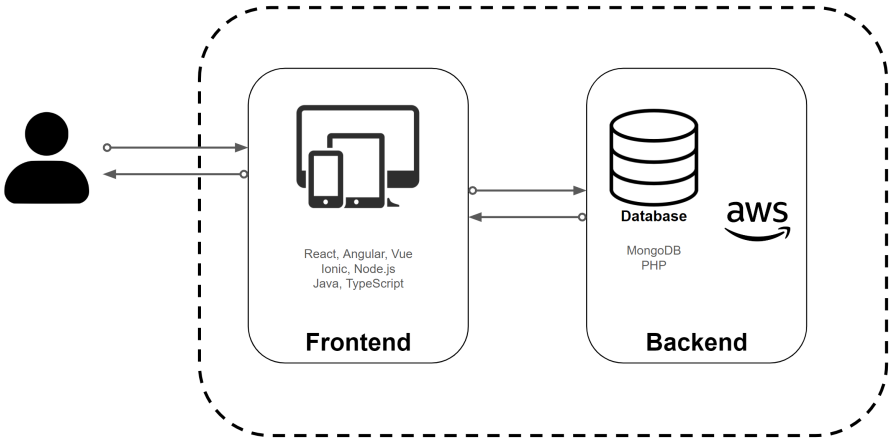


Figura 1.5: Architettura delle tecnologie del prodotto

1.5 Propensione all'innovazione

Wavelop Srls pone sempre tra i suoi obiettivi principali quello di utilizzare le tecnologie più recenti e migliori sul mercato, dimostrando così un atteggiamento proattivo verso l'innovazione. In ogni progetto cerca sempre di acquisire esperienze nuove adottando approcci innovativi, ma perseguendo sempre la qualità in ciò che produce. Ne sono la conferma l'utilizzo di:

* **Metodologia Agile**: per gestire il lavoro in modo efficace ed efficiente con le risorse limitate che l'azienda;
* **Git flow**: per scrivere il codice in modo collaborativo;
* **Tecnologie diverse**: in modo tale da non specializzarsi solamente nell'utilizzo di alcune tecnologie, ma rimanere sempre aggiornati riguardo nuove possibili soluzioni ed al passo con quanto richiesto dal mercato. Di fatto, oltre all'utilizzo delle tecnologie presentate nella sezione §1.4, Wavelop Srls è sempre alla ricerca di novità studiandole e sperimentandole internamente in previsione di possibili utilizzi in progetti futuri.

Capitolo 2

Il progetto nella strategia aziendale

Questo capitolo tratta il contesto nel quale avviene il mio progetto di stage. Verranno descritti il rapporto generale che l'azienda ha con l'accoglienza e le proposte di stage, il progetto a monte del quale il mio tirocinio si colloca ed i vari vincoli e obiettivi ad esso associati.

2.1 L'azienda e gli stage

Wavelop Srls si affaccia agli *stage* partecipando a vari eventi, in modo da far conoscere la propria realtà ed entrare in contatto con più persone possibile. L'evento principale al quale partecipa è STAGE-IT, un evento annuale promosso da Assindustria Venetocentro in collaborazione con l'Università di Padova che favorisce l'incontro tra aziende con progetti innovativi in ambito *IT* e studenti dei corsi di laurea in Informatica, Ingegneria Informatica e Statistica.
Data l'affluenza a tali eventi, per scegliere il candidato ideale l'azienda ha deciso di sottoporre ad un breve *test* di programmazione tutte le figure che ritengono interessanti. Questo *test* consiste in tre quesiti generici da risolvere in poche linee di codice. Inoltre, per una migliore valutazione, viene registrata ogni azione di scrittura e cancellazione delle risposte. Ciò permette di valutare non solo la risposta finale, ma anche il procedimento che ha portato il candidato a tale soluzione, così da avere una valutazione più precisa riguardo l'idoneità o meno di quest'ultimo di partecipare allo *stage*. La scelta finale è influenzata anche dall'opportunità di poter, in futuro, allargare il gruppo di collaboratori con delle figure giovani, più che come possibilità di avere della semplice manodopera per lo svolgimento di vari progetti.
Wavelop Srls ha a disposizione più progetti sui quali possono lavorare gli stagisti. Questi riguardano sia progetti interni di cui vanno ampliate le funzionalità, che progetti esterni che vanno mantenuti o sviluppati tramite delle nuove tecnologie. Durante il periodo di *stage* l'azienda cerca di trovare con il tirocinante il giusto equilibrio tra autonomia e supporto. Quest'ultimo è dato soprattutto per la pianificazione del lavoro, per la verifica del codice prodotto e per l'introduzione alle nuove tecnologie che andranno utilizzate. Questa modalità di interazione lascia la libertà al tirocinante di ambientarsi e di studiare effettivamente come funzionano le diverse tecnologie, in modo tale da concentrarsi soprattutto nel produrre del codice di qualità entro quanto pianificato.

2.2 Il progetto Timesheet

Come esposto in §1.3.2.4, la rendicontazione delle ore di lavoro nell'azienda avviene tramite *Clockify* e *GitLab*. L'utilizzo di queste tecnologie rende però tale attività complessa e poco efficiente, soprattutto quando bisogna utilizzare entrambi i servizi. Per questo Wavelop Srls ha deciso di sviluppare un prodotto per l'uso interno che permetta di raggruppare in un'unica piattaforma entrambe le tipologie di rendicontazione (generale per le attività e specifica per i singoli *task*), includendo inoltre funzionalità di *scheduling* e gestione dei *team* assegnati ai diversi progetti. Ciò permette di rendere tali attività più efficienti e semplici da eseguire. Tale progetto è *Timesheet*.
Quest'ultimo, nello stato precedente al mio *stage*, permetteva le seguenti funzionalità:

* Accedere alla piattaforma mediante *SSO* (*Single-Sign-On*), un processo di autenticazione che consente l'accesso a più risorse, servizi digitali e applicazioni con un solo set di credenziali;
* Rendicontare e schedulare le attività svolte su clienti e progetti inseriti nel *database*;
* Gestire i diversi clienti e progetti nel *database*;
* Visualizzare ed esportare dei *report* personalizzati, ad esempio filtrando alcuni campi in modo tale da raggruppare diversamente le attività svolte.

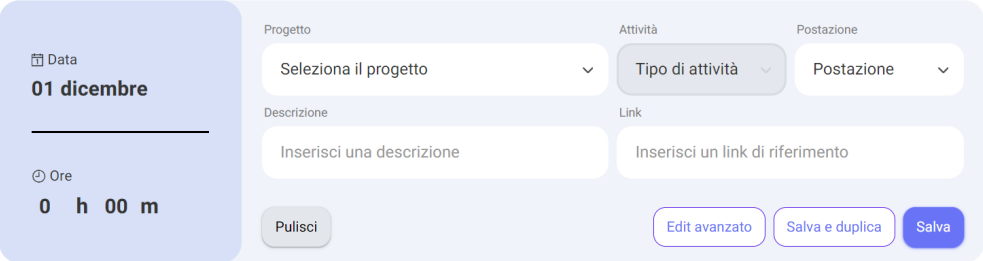


Figura 2.1: *Form* di inserimento di un'attività in *Timesheet*

La funzionalità principale dell'applicativo è poter rendicontare e schedulare le diverse attività. Per fare ciò, dalla schermata principale, è possibile salvare una qualsiasi attività tramite il *form* esposto dalla Figura 2.1 definendone:

* La data;
* Le ore di lavoro impegnate;
* Il progetto a cui fa riferimento l'attività;
* Il tipo di attività svolta;
* Il luogo di lavoro;
* La descrizione breve dell'attività svolta, che però non è obbligatoria;
* Il *link* alla pagina dell'attività che si è svolta (ad esempio la pagina del *task* su *GitLab*), non è obbligatorio.

Durante il mio *stage* sono state implementate le seguenti funzionalità:

* Gestire gli utenti assegnati ad un progetto;
* È stata creata una estensione per *Google Chrome* che permette di rendicontare e schedulare le attività direttamente dalla pagina del *task* al quale si è lavorato.

L'applicativo allo stato attuale necessita ancora di migliorie, sia a livello di codice che di funzionalità, per essere integrato totalmente nel *workflow* aziendale. Successivamente a questo sviluppo rimane comunque aperta la possibilità di commercializzare tale prodotto, dato che può essere molto utile non solo per Wavelop Srls, ma anche per altre realtà.

2.3 Il mio stage

2.3.1 Obiettivi

Wavelop Srls, all'inizio di ogni *stage*, prefissa degli obiettivi da perseguire necessari sia allo stagista, per poter capire su quali aspetti del progetto concentrarsi maggiormente, che all'azienda, per poter valutare il lavoro del tirocinante e capire se può essere o meno inserito all'interno dell'organico in futuro. Tali obiettivi sono divisi in: formativi, funzionali e di integrazione.

Obiettivi Formativi

Sono gli obiettivi che riguardano l'apprendimento delle principali tecnologie utilizzate dal *team*. Nel mio caso è stato richiesto l'ampliamento delle conoscenze riguardanti alcune tecnologie utilizzate dall'azienda e soprattutto delle metodologie agili e delle norme utilizzate nello sviluppo dei progetti. Nello specifico, gli obiettivi formativi stabiliti per il mio *stage* sono stati:

* Ampliare le nozioni riguardo la metodologia di sviluppo *agile*;
* Aumentare le capacità di analisi e sviluppo di un progetto
* Apprendere nuovi concetti legati al linguaggio *Typescript*/*Javascript*;
* Acquisire nuove competenze riguardo l'utilizzo delle tecnologie: *React*, *Node.js*, *MongoDB* e *Docker*;
* Prendere dimestichezza nella gestione del versionamento del progetto tramite *Git* e la piattaforma *GitLab*;
* Assimilare le nozioni riguardanti la creazione e pacchettizzazione di una estensione per *Google Chrome*.

Obiettivi Funzionali

Sono gli obiettivi che riguardano la soddisfazione dei requisiti obbligatori, desiderabili e facoltativi della proposta di *stage*. Fanno riferimento ai prodotti attesi a seguito dell'attività di tirocinio. Nello specifico del mio *stage*, gli obiettivi funzionali stabiliti sono stati:

* **Obiettivi Obbligatori**:
  - Assegnare un utente ad un progetto;
  - Rimuovere un utente da un progetto;
  - Visualizzare gli utenti assegnati ad uno o più progetti;
  - Creare estensione per *Google Chrome*;
  - *Login* tramite l'estensione;
  - Visualizzazione dei progetti assegnati;
  - *Scheduling* di una attività collegata alla pagina corrente.
* **Obiettivi Desiderabili**:
  - Gestione dei ruoli degli utenti all'interno dell'applicativo. Ogni utente, in base al proprio ruolo, potrà vedere solamente determinate sezioni;
  - Memorizzazione dei metadati del *link* associato ad una attività in base alla sorgente, ad esempio *Gitlab*, *Trello* e *OTRS*;
  - Implementazione di *Feature Flag* per la gestione di contenuti *premium* in abbonamento.
* **Obiettivi Facoltativi**:
  - Multilingua;
  - Integrazione dell'applicativo a un sistema di fatturazione elettronica;
  - *Refactor* del *form*.

Obiettivi di Integrazione

Sono gli obiettivi che riguardano le capacità dello stagista di integrarsi all'interno dell'organico aziendale. Questo aspetto è ritenuto di notevole importanza, anche a discapito di maggiori capacità individuali della persona a livello di conoscenze tecniche. L'azienda ritiene di maggiore importanza avere una risorsa che sappia interagire positivamente con le altre, piuttosto che avere una persona capace ma poco collaborativa, che rischia di danneggiare l'ambiente di lavoro.

2.3.2 Prodotti Attesi

Oltre agli obiettivi da perseguire, Wavelop Srls definisce anche i prodotti attesi al termine di ogni *stage*. Questi hanno lo scopo di definire il prodotto finale che si dovrebbe avere una volta terminata l'attività di progetto. Inoltre, a partire da questi, sono definiti gli obiettivi esposti nella sezione §2.1, ad eccezione di quelli che riguardano aspetti teorici come l'acquisizione di competenze oppure di integrazione con il gruppo. Per il mio *stage* sono stati definiti i seguenti prodotti attesi:

* Estensione *Chrome* per la schedulazione di attività;
* Sviluppo visibilità entità Utente e Progetto ;
* Produzione test di unità per *controllers API*;
* Documentazione con analisi e progettazione di *evolutive business* e richieste di miglioramento;

Per ogni prodotto atteso è definita una versione minima ed una massima in modo tale da poterne valutare il livello di sviluppo:

* **Versione minima**: tutti gli obiettivi obbligatori riferiti al prodotto atteso sono stati raggiunti;
* **Versione massima**: tutti gli obiettivi riferiti al prodotto atteso, quindi sia obbligatori che facoltativi e desiderabili, sono stati raggiunti.

2.4 Obiettivi personali

Gli obiettivi che mi hanno portato a scegliere di fare lo *stage* presso l'azienda Wavelop Srls sono stati:

* **Apprendere diverse nuove tecnologie**: il progetto proposto dall'azienda prevedeva l'utilizzo di diverse tecnologie, sia per la parte di *frontend* che per quella di *backend*. Ciò è stata un'ottima opportunità per poter fare esperienza con più tecnologie e linguaggi di programmazione che sono molto richiesti nel mondo del lavoro e di cui non ho ancora molta esperienza;
* **Azienda piccola e giovane**: essere in un contesto piccolo ma comunque stimolante ha influenzato positivamente la mia scelta. Ho trovato la possibilità di confrontarmi facilmente con tutte le diverse figure che lavorano all'interno dell'azienda molto importante, soprattutto per poter vedere i diversi punti di vista ed imparare nuovi approcci alle problematiche che mi venivano poste;
* **Metodologie agili**: trattandosi di un *team* che utilizza una metodologia *Agile* per lo sviluppo dei progetti, ho avuto la possibilità di apprendere gli strumenti e le tecniche necessarie per svolgere questo tipo di attività;
* **Fare esperienza della vita aziendale**: Wavelop Srls è un'agenzia *IT* che lavora a più progetti contemporaneamente con le tecnologie più disparate, a differenza di altre aziende più statiche. Questo mi ha dato la possibilità di vedere le diverse dinamiche di lavoro che avvengono sia internamente all'azienda che esternamente, con i diversi clienti e le varie richieste da soddisfare;
* **Possibilità di proseguire con il *team***: ho considerato questo aspetto importante. Soprattutto per la garanzia di lavorare in un'azienda che avesse lo scopo di istruire e formare al meglio una persona, anziché utilizzarla solamente come risorsa temporanea.

Capitolo 3

Attività di stage

Questo capitolo espone un'analisi approfondita del progetto di stage, presentandone la pianificazione adottata, gli studi delle tecnologie e le scelte che ho preso durante lo sviluppo. Vengono descritte le difficoltà affrontate e l'implementazione effetiva della soluzione finale sia da un punto di vista generale, ad alto spettro, che più specifico.

3.1 Pianificazione

3.1.1 Organizzazione generale

Lo *stage* ha avuto una durata di circa 310 ore di lavoro, distribuite in 8 settimane tra lunedì 19 Settembre 2022 e venerdì 11 Novembre 2022. Queste settimane combaciano con la durata degli *sprint* adottata dall'azienda. La pianificazione generale delle attività da svolgere durante questo periodo è la seguente:

* **Prima settimana**: nella quale è stata fatta un'introduzione alla cultura aziendale ed al progetto, discutendone più approfonditamente i requisiti e le richieste relative al sistema da sviluppare. Inoltre è stata fatta della formazione riguardo le tecnologie da adottare ed è stato configurato l'ambiente di lavoro con tutti gli strumenti necessari;

* **Seconda settimana**: nella quale è stata fatta l'analisi dei requisiti del progetto ed iniziata la progettazione architetturale;

* **Terza settimana**: nella quale sono state terminate tutte le attività preliminari al progetto ed è stato definito il *backlog* iniziale del prodotto. Successivamente, è stato definito il *backlog* dello *sprint* della settimana successiva assieme al referente;

* **Quarta settimana - Ottava settimana**: in ciascuna settimana è stata svolta l'implementazione dei *task* presenti nel *backlog* dello *sprint*. Al termine di ogni periodo è stata fatta la *sprint review* con il referente ed altre persone coinvolte nel progetto per valutare l'avanzamento eseguito, testare quanto sviluppato e pianificare il *backlog* dello *sprint* successivo.

Di seguito, nella Figura 3.1 ho riportato un Gantt che mostra la divisione delle varie attività durante il progetto:

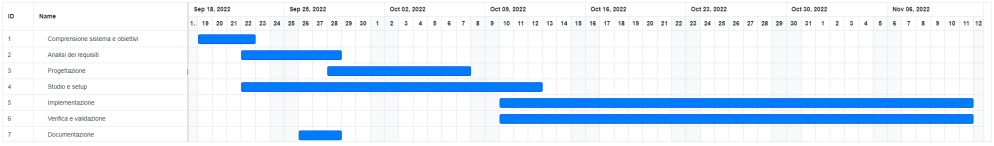


Figura 3.1: Divisione delle attività svolte durante il progetto

3.1.2 Attività

Nello specifico, le attività svolte durante il progetto sono state:

* **Comprensione del sistema e degli obiettivi**: in quest'attività ho esaminato lo stato dell' applicativo principale, cercando di comprenderne i meccanismi e gli strumenti utilizzati. Inoltre, ho preso familiarità con il *framework React*, *TypeScript* e *Node.js*, ovvero gli strumenti più importanti per lo sviluppo del progetto proposto. Infine, ho analizzato gli obiettivi richiesti e studiato le metodologie utilizzate da Wavelop Srls per definire i requisiti del progetto e per gestire il versionamento del codice;

* **Analisi dei requisiti**: affiancato al *tutor* aziendale, in questa attività ho determinato i requisiti da soddisfare durante il progetto tramite l'utilizzo di *user stories* anziché dei casi d'uso, tale scelta è spiegata nella sezione §3.3.2;

* **Progettazione**: sempre affiancato dal *tutor* aziendale, in questa attività ho delineato l'architettura generale e descritto più dettagliatamente le componenti ed i comportamenti richiesti dall'estensione. In questo periodo, inoltre, ho definito l'organizzazione delle componenti nell'estensione, cercando di mantenere un'estetica simile a *Timesheet*;

* **Studio e *setup* dell'ambente di sviluppo**: in questa attività, per strutturare l'applicazione correttamente, ho utilizzato un *boilerplate* pubblicato su *GitHub*, ovvero una *repository* contenente il codice e l'organizzazione base dei *file* necessari per il giusto funzionamento dell'estensione;

* **Implementazione**: in questa attività ho implementato tutto quello stabilito nella analisi dei requisiti e nella progettazione, seguendo un avanzamento settimanale;

* **Test e validazione**: una volta terminata la codifica, per ogni periodo, ho eseguito attività di verifica del codice scritto per accertarne la correttezza. L'attività qui descritta consiste principalmente in verifica statica del codice e prove di funzionamento;

* **Documentazione**: purtroppo questa attività non ha avuto molto spazio all'interno dello sviluppo del progetto vista la grande mole di *software* da implementare.

Nel dettaglio, le ore di lavoro utilizzate per ciascuna attività sono riportate nella seguente tabella:

Attività	Ore di lavoro
Comprensione del sistema e degli obiettivi	20
Analisi dei requisiti	40
Progettazione	40
Studio e *setup* dell'ambente di sviluppo	20
Implementazione	130
Test e validazione	52
Documentazione	10
Totale	**312**

Tabella 3.1: Attività svolte con le ore di lavoro corrispondenti.

3.2 Avvicinamento al progetto

3.2.1 Studio delle tecnologie e delle norme

Durante il mio percorso accademico, specialmente nel corso di Ingegneria del *Software*, mi sono imbattuto nell'utilizzo di tecnologie riguardanti l'ambito *web* per lo sviluppo di applicazioni. Tale esperienza però, nonostante la conoscenza pregressa anche di *HTML* e *CSS*, non è stata sufficiente con quanto necessario per lo sviluppo del progetto proposto. Colmare queste lacune, almeno in modo teorico, è stato il principale obiettivo dei primi giorni dello *stage*, in modo tale da poter iniziare il prima possibile la progettazione e lo sviluppo del prodotto finale richiesto.

Le tecnologie che ho studiato maggiormente durante questa prima fase sono:

* **React**: è una delle tecnologie maggiormente usate per lo sviluppo di applicazioni *frontend*. È stata scelta come tecnologia da utilizzare per lo sviluppo di *Timesheet* e, per mantenere un linea comune nello sviluppo dell'estensione, ho deciso di utilizzarla anche per il progetto di *stage*. Il principale vantaggio dato dall'utilizzo di questa tecnologia è facilitare lo sviluppo a componenti, approccio che si sposa bene con la metodologia di sviluppo *Agile Scum* adottata da Wavelop Srls.
  Di fatto, la principale *feature* utilizzata è stata quella dei ***Feature Components***. Introdotti in *React16*, sono delle semplici funzioni *javascript* che accettano informazioni in forma di *prop* e ritornano un *React Element*, che non è altro che del semplice codice *HTML*. Vengono sempre più utilizzati nelle applicazioni che utilizzano *React* anche grazie alla possibilità di utilizzare dei *lifecycle methods* al loro interno, ovvero dei metodi che, rimanendo in ascolto di eventuali modifiche al componente durante tutto il ciclo di vita di quest'ultimo, ne permettono l'aggiornamento;

* **TypeScript**: negli ultimi anni *TypeScript* è diventato sempre più popolare tra gli sviluppatori *frontend*. I principali vantaggi dati dall'utilizzo di questo linguaggio di programmazione sono:

  – Rende il codice più facile da leggere e mantenere, rendendo così minore la probabilità di *bug* e più efficiente l'attività di sviluppo;

  - Con la costante crescita in popolarità di *TypeScript*, aumentano anche le librerie esterne supportate dal linguaggio di *default*. Questo ne rende più facile l'utilizzo;
  - Consente di adattare progetti iniziati con *JavaScript* a *TypeScript* in modo graduale, aumentando man mano la copertura di quest'ultimo;
  - Permette una migliore *IntelliSense*, una forma di completamento automatico resa popolare da *Visual Studio* (ambiente di sviluppo creato da *Microsoft*). Questo è utile anche come documentazione per i nomi delle variabili, delle funzioni e dei metodi usando metadati e *reflection*;
  - Permette l'utilizzo di tipi statici, in modo tale da riuscire ad individuare prima gli errori presenti nel codice.

  Anche questo strumento ho deciso di utilizzarlo per via del suo impiego in *Timesheet*, in modo tale da mantenere una linea comune non solo nel linguaggio utilizzato, ma anche nelle metodologie e le convenzioni utilizzate;

* **Node.js**: è stata utilizzata per lo sviluppo di *Timesheet* grazie ad alcuni vantaggi che offre, ovvero:
  - Data la presenza sia di una parte *frontend* che *backend* nell'applicativo, permette di evitare l'utilizzo di un linguaggio di programmazione separato lato *server*;
  - Permette di sfruttare i vantaggi del *Node Package Manager* (*NPM*), che consente la gestione delle dipendenze dei vari strumenti del progetto;
  - Con la sua architettura basata su eventi, permette una sincronizzazione veloce tra il lato *server* ed il lato *client*. Diventa così una soluzione ideale per le applicazioni in tempo reale, come *Timesheet*.

Questi strumenti, insieme alle basi *HTML* e *CSS*, costituiscono un insieme di tecnologie che permette di sviluppare applicazioni *frontend*. Nel mio caso tale applicazione è l'estensione per *Google Chrome*. Nello specifico, *React* definisce un *framework* che permette di velocizzare e rendere più semplice il processo di sviluppo; all'interno del *framework* utilizziamo *TypeScript*, che comprende *Javascript* al quale aggiunge delle funzionalità utili per lo sviluppo di applicazioni *web*, e *Node.js* per la gestione delle varie dipendenze esterne del codice.

3.2.2 Configurazione dell'ambiente di lavoro

Lo sviluppo di una estensione per *Google Chrome* è simile allo sviluppo di una pagina *web* statica. Di fatto i linguaggi necessari sono *HTML*, *CSS* e *JavaScript*. Per il progetto però, l'obiettivo è sviluppare l'estensione utilizzando *React*, *TypeScript* e *Node.js*, in modo tale da poterne sfruttare i vantaggi e le potenzialità. Per fare ciò ho utilizzato una *repository* su *GitHub*, sviluppata da Michael Xieyang Liu, contenente del codice *boilerplate*, del codice standard che può essere riutilizzato per più progetti. Lo scopo principale di tale codice è definire la struttura principale di un'estensione per *Google Chrome* che non sia solo una pagina statica, ma che utilizzi le seguenti tecnologie:

* **Chrome Extension Manifest V3**: è il cuore dell'estensione, viene utilizzato dal *browser* per conoscere le informazioni di base dell'estensione, come i *file* più importanti, i permessi e le capacità che potrebbe utilizzare;

* **React 17**;
* **Webpack 5**: è uno *static module bundler* che ha il compito di processare l'estensione. Partendo dai *file* dell'estensione, crea dei grafi di dipendenza da uno o più *entry points* e combina i vari moduli necessari in uno o più *bundle*, ovvero degli *assets* statici necessari;
* **Webpack Dev Server 4**: inizializzato con *npm*, permette l'*auto reload* delle nuove *feature* implementate nel codice. Questo rende estremamente più veloce l'attività di sviluppo;
* **React Hot Loader**: permette l'aggiornamento delle modifiche nel codice in "tempo reale" sull'estensione caricata. Come *Webpack Dev Server*, rende l'attività di sviluppo molto più efficiente e meno complicata;
* **TypeScript**.

Queste tecnologie mi hanno permesso, da un lato, di sviluppare l'estensione tramite *React* e gli altri strumenti utilizzati per lo sviluppo di applicazioni *web* (*TypeScript e React Hot Loader*), dall'altro di convertire quanto sviluppato nei *file* corretti che *Google Chrome* utilizza per pubblicare l'estensione. La Figura 3.2 rappresenta la divisione delle tecnologie nei due ruoli principali:

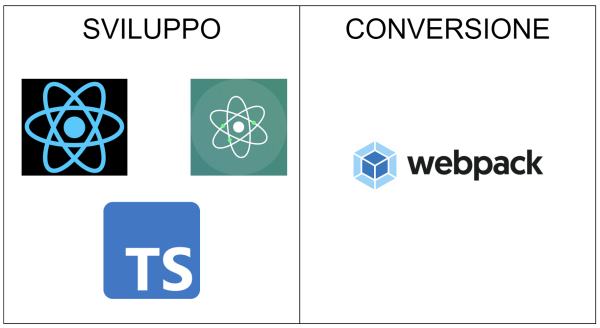


Figura 3.2: Divisione delle tecnologie del *boilerplate* per ruolo nello sviluppo.

La struttura base dell'estensione è la seguente:

```
chrome-extension/
|__utils/
|__src/
|   |__assets/
|   |__pages/
|   |   |__Background/
|   |   |__Content/
|   |   |__Devtools/
|   |   |__Newtab/
|   |   |__Options/
|   |   |__Panel/
|   |   |__Popup/
|__package.json
|__README.md
|__webpack.config.json
|__tsconfig.json
```

La cartella *utils* contiene tutti i *file* di utilità e di configurazione necessari per far funzionare il progetto. La cartella *src* contiene, invece, l'applicazione vera e propria. All'interno sono presenti: il *file manifest*, contenente la configurazione di base dell'estensione, e le due sottocartelle *assets* e *pages*. La prima contiene tutti i *file* di supporto che non hanno una posizione definita all'interno dell'applicazione. *Pages*, invece, contiene le cartelle che si riferiscono alle diverse pagine utilizzate dall'estensione. In particolare:

* **Popup**: contiene i *file* relativi alla pagina principale dell'estensione, quella che l'utente vede durante il suo utilizzo;
* **Background**: contiene i *file* relativi ai *service workers*. Questi sono degli *script* che sono eseguiti in *background*, separati dall'estensione, e che forniscono *feature* che non necessitano di una pagina *web* o di un'interazione con l'utente;
* **Newtab**: contiene i *file* relativi alla pagina da visualizzare ogni qual volta l'utente apre una nuova *tab*;
* **Options**: contiene i *file* relativi alla pagina opzioni dell'estensione;
* **Devtools**: contiene i *file* relativi alla pagina accessibile tramite *devtools*;
* **Panel**: contiene i *file* relativi alla pagina visibile dal *devtools panel*;
* **Content**: contiene i *file* che vengono eseguiti nel contesto della *web page*. Usando il *DOM* permettono di leggerne i dettagli, fare delle modifiche e passare delle informazioni all'estensione padre. Possono inoltre accedere alle *Chrome API* usate dal padre tramite lo scambio di messaggi con l'estensione.

Come detto precedentemente questi *file* sono presenti nell'estensione, ma ciò non significa che vengano utilizzati tutti. Il *manifest* ha lo scopo di definire quali *file* utilizzare (e quali non utilizzare) nella configurazione di base dell'estensione, che viene utilizzata da *Google Chrome*. Tale concetto viene esposto nella Figura 3.3.
Infine, essendo tutti i componenti scritti in *JavaScript*, li ho convertiti in *TypeScript* in modo tale da utilizzare tutti gli strumenti prefissati per il progetto.

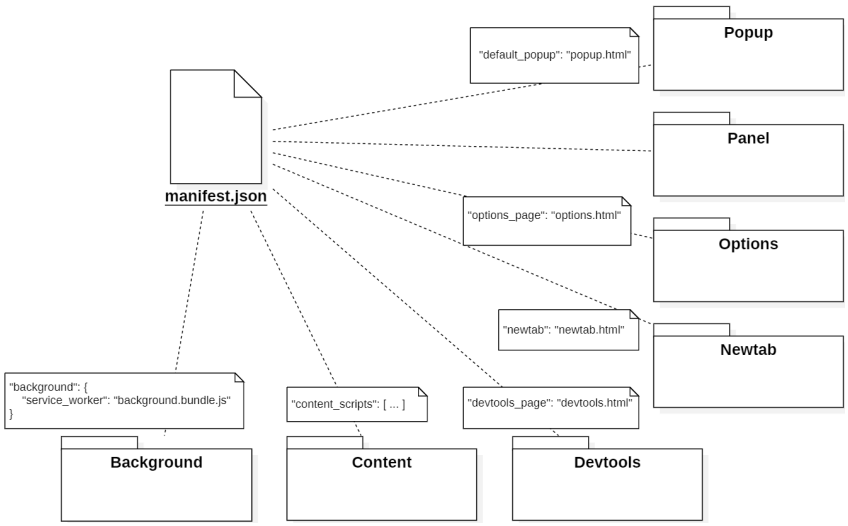


Figura 3.3: Organizzazione delle pagine dell'estensione.

3.3 Analisi dei requisiti

3.3.1 Attori principali

La Figura 3.4 riporta la gerarchia degli attori. Dopo aver analizzato lo scopo e le funzionalità dell'applicativo ho individuato tre tipi principali di attori:

* **Utente generico**: questo utente ha preso per la prima volta conoscenza dell'applicativo, per questo motivo non ha ancora un *account* e, di conseguenza, non può utilizzare alcuna funzionalità. L'unica cosa che vede all'apertura di *Timesheet* o dell'estensione è la richiesta di effettuare la registrazione alla piattaforma;

* **Utente registrato ma non autenticato**: questo utente ha già effettuato la registrazione delle proprie credenziali nel sistema, però non ha ancora effettuato il *login* all'applicativo. Pertanto la pagina e l'estensione visualizzeranno solamente la richiesta di effettuare tale *login*;

* **Utente autenticato**: questo utente ha effettuato il *login* all'applicativo ed ora ha a disposizione tutte le sue funzionalità, sia della piattaforma che dell'estensione.

Al momento questa lista è ancora incompleta, essendo l'applicativo principale ancora in una fase di sviluppo non finale, non presenta alcuna differenza tra gli utenti autenticati. In futuro un'obiettivo di *Timesheet* sarà avere utenti con diversi tipi di autorizzazioni per ogni progetto al quale lavorano.

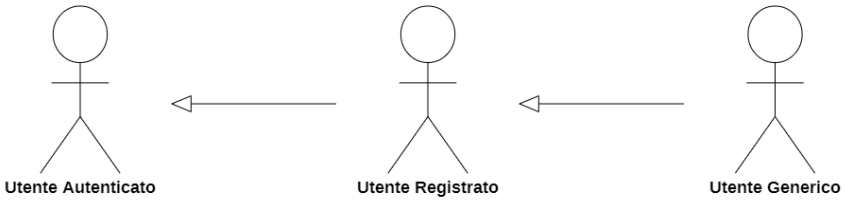


Figura 3.4: Gerarchia degli attori principali.

3.3.2 User stories

Per l'analisi dei requisiti e delle funzionalità del progetto di *stage*, al posto dei casi d'uso, ho utilizzato le *user stories*, lo strumento utilizzato da Wavelop Srls per l'analisi dei requisiti di ogni progetto al quale lavora.

Le *user stories* hanno lo scopo aiutare nella descrizione in dettaglio delle funzionalità che hanno valore per l'utente di un prodotto *software*, piuttosto che documentarlo. Ogni *user story* è composta da tre aspetti fondamentali:

* **Carta**: una descrizione della storia utile per la pianificazione e come promemoria;

* **Conversazione**: una conversazione riguardo la storia, utile per esporre tutti i dettagli riguardanti quest'ultima;

* **Conferma**: dei *test* che documentino i dettagli e che possano essere usati per determinare il completamento della storia in questione.

Un aspetto molto importante, che il *team* di sviluppo deve definire fin dall'inizio, è la grandezza delle storie e la loro profondità. Questo perché un utilizzo scorretto delle *user stories* può portare alla mancanza di dettagli e, di conseguenza, ad sviluppo incompleto delle funzionalità richieste. Le storie possono essere definite anche in insiemi di *user stories* correlate tra di loro, questi gruppi vengono definiti *epic*. Di fatto, è preferibile avere più storie separate rispetto ad un'unica grande storia che risulta troppo grande, in modo tale da poter dividere meglio il lavoro nei diversi avanzamenti del progetto.
Un progetto *story-driven* ed una metodologia di sviluppo *agile* condividono la stessa filosofia di coinvolgere il cliente continuamente durante tutto il progetto. Infatti le storie vengono inizialmente definite durante un *workshop* apposito, ma possono essere scritte in qualsiasi momento durante il progetto. Il processo di pianificazione e sviluppo tramite l'utilizzo di storie è quindi il seguente:

1. *Workshop* iniziale, durante il quale vengono scritte più storie possibili;
2. Il *team* di sviluppo stima la grandezza di ciascuna storia;
3. Viene decisa la grandezza, fissa, delle iterazioni che può andare da una a quattro settimane;
4. Si può iniziare lo sviluppo. Al termine di ogni periodo il *team* di sviluppo ha il compito di presentare del codice funzionante per alcuni aspetti del prodotto finale. Il cliente rimane coinvolto durante ogni iterazione, parlando con gli sviluppatori riguardo le storie pianificate per quel periodo.

Si può facilmente notare come questo tipo di sviluppo condivida molti aspetti con la metodologia *agile*. Primo su tutti è il coinvolgimento del cliente durante tutto il progetto, ma anche la pianificazione dello sviluppo in periodi da una a quattro settimane e la revisione a fine periodo di quanto implementato.
Vi sono quindi diversi modi per definire i requisiti che il prodotto finale di un progetto deve rispettare. I vantaggi dell'usare le *user stories* sono i seguenti:

* **Enfatizzano la comunicazione verbale**: lo scopo delle *user stories* non è documentare ogni minimo dettaglio riguardo una determinata funzionalità, bensì scrivere delle frasi semplici e concise che ci ricordino di discuterne con il cliente in prima persona, facilitando notevolmente la comunicazione;
* **Sono comprensibili da chiunque**: le storie sono concise e sono sempre scritte mostrando il valore di una funzionalità dal punto di vista del cliente o dell'utente, sono sempre facilmente comprensibili sia dal cliente che dagli sviluppatori;
* **Sono della giusta dimensione per la pianificazione**: non vi è una grandezza definita ed obbligatoria. Ogni *team* adatta le storie in base alle proprie capacità e necessità, in modo tale da semplificare la pianificazione e lo sviluppo;
* **Funzionano con modelli di ciclo di vita iterativi**: non vi è la necessità di scrivere tutte le storie prima di iniziare lo sviluppo. È possibile scrivere alcune storie, implementarle, testarle e poi ripetere il procedimento tutte le volte che è necessario. Le storie posso essere scritte a qualsiasi livello di dettaglio si ritenga appropriato. Quindi le storie funzionano bene per lo sviluppo iterativo a causa di quanto sia facile iterate sulle storie stesse;

* **Incoraggiano il rinvio dei dettagli**: non bisogna per forza scrivere tutti i dettagli di una storia già dall'inizio. Man mano che si avanza nello sviluppo vengono specificate le storie necessarie;

* **Supportano design opportunistici**: sviluppare un prodotto [/software] con un approccio *top-down* è molto improbabile per via di variabili come ciò che desidera il cliente, specifici dettagli implementativi che si conoscono solamente più avanti nello sviluppo, oppure eventuali errori.
  Poiché non può esistere un processo di sviluppo che proceda in un percorso strettamente lineare dai requisiti di alto livello al codice, le *user stories* consentono facilmente ad un *team* di cambiare tra requisiti di alto e basso livello a seconda delle necessità;

* **Incoraggiano la partecipazione nella progettazione**: la grande accessibilità delle storie incoraggia il cliente ad essere più partecipativo alla progettazione del *software*. Inoltre, il cliente può imparare a definire le proprie richieste in modo da facilitare la stesura delle storie per aiutare gli sviluppatori. Questo porta ad una maggiore partecipazione degli sviluppatori stessi che, di conseguenza, cercheranno di coinvolgere ancora di più i clienti, creando così un circolo virtuoso;

* **Costruiscono una conoscenza tacita**: data l'enfasi delle storie ad una comunicazione faccia a faccia, le storie favoriscono la crescita di conoscenza tra il *team* ed il cliente man mano che le discussioni ed il coinvolgimento aumenta.

3.3.3 Definizione e sviluppo delle *user stories*

Durante tutto lo *stage* ho utilizzato le *User Stories* per definire i requisiti ed i *task* da implementare. All'inizio del progetto, durante l'attività di analisi dei requisiti, ho definito la versione iniziale di tutte le storie riguardanti le funzionalità richieste. Ad esempio, ho definito la seguente *user story* riguardante il *setup* e la visualizzazione del *form* di rendicontazione:

> ***Titolo**: Visualizzazione dell'estensione*
> ***Descrizione**: Come utente autenticato voglio poter visualizzare il form per la rendicontazione dell'attività.*

da notare come, trattandosi di una storia che avrei affrontato più avanti nello sviluppo, non ho definito ancora alcun *task* necessario per il suo completamento.

In seguito, durante l'attività di implementazione, prima di iniziare il vero e proprio sviluppo di ciascuna funzionalità ho discusso con il *tutor* aziendale per definire più dettagli possibili relativi alla storia, sia riguardo al *design* finale che alla *user experience*. Con questi dettagli aggiuntivi ho definito i diversi *task* associati alla storia e, solo successivamente, svolto l'effettivo sviluppo della nuova funzionalità. Prendendo l'esempio fatto precedentemente, a seguito delle discussioni con il *tutor* aziendale e degli studi su *Formik* (descritto in §3.5.3) svolti durante il terzo *sprint*, ho modificato la descrizione della storia ed aggiunto i *task* principali sui quali basare lo sviluppo della funzionalità:

> ***Titolo**: Visualizzazione dell'estensione*
> ***Descrizione**: Come utente autenticato e acceduto al pop-up voglio poter visualizzare il form per la rendicontazione dell'attività.*

***Task**:*

- *Definire componente Formik;*
- *Definire valori iniziali campi dato;*
- *Definire wrapper campi dato;*
- *Definire logica di validazione;*
- *Definire componente stile*

Infine, al termine dell'implementazione di ogni storia, ho svolto una revisione di quanto prodotto con il *tutor* aziendale. Insieme, abbiamo analizzato i *task* svolti relativi alla storia implementata; nello specifico abbiamo analizzato la forma che ho utilizzato, come sono arrivato a quella determinata soluzione ed eventuali accorgimenti da tenere a mente per gli avanzamenti futuri. Successivamente, dopo aver verificato l'effettivo funzionamento di quanto prodotto, passavo alla storia seguente nello *sprint backlog*.
Al termine dello *stage*, le storie definitive sono state utilizzate per confermare l'effettivo completamento delle funzionalità richieste.

3.3.4 Analisi dei rischi

Durante lo sviluppo del progetto vi è la possibilità di incorrere in problemi che rallentano e rendono più complesso il lavoro, ma che possono essere previsti e per questo controllati. Perciò, già durante le prime fasi di analisi del progetto, ho studiato i principali fattori di rischio. La procedura per l'individuazione e la gestione dei rischi è composta principalmente dalle seguenti due fasi:

- **Identificazione:** dove vengono individuati i vari fattori problematici che possono rappresentare dei rischi nel proseguimento del progetto;
- **Monitoraggio:** attività continua per tutta la vita del progetto volta ad evitare che queste complicazioni abbiano luogo e, nel caso peggiore, permetterne una gestione rapida ed efficace.

Quest'ultima fase è resa decisamente più facile dalla pianificazione del lavoro in *sprint* settimanali. Ciò permette, in primo luogo, di avere un monitoraggio pressoché continuo, visto l'avanzamento del lavoro a piccoli passi e la presenza degli *stand-up meeting* giornalieri; inoltre, le *sprint review* hanno anche l'obiettivo di analizzare ed individuare eventuali rischi e problemi individuati durante lo sviluppo, così da gestirli il prima possibile ed evitare che causassero problemi ben più gravi.

I rischi possono essere di due categorie:

- **Tecnologico**: ovvero riguardante le tecnologie da implementare e le prassi da utilizzare nel farlo. Questo tipo di rischi è stato mitigato dalla disponibilità del *tutor aziendale* e dagli *stand-up meeting* giornalieri, nei quali ho potuto esprimere eventuali perplessità ed eventuali rallentamenti;
- **Organizzativo**: relativi alla stima ed all'assegnazione dei *task* ad un determinato *sprint*. Questo tipo di rischi è stato mitigato soprattutto dalle *sprint review*, che hanno l'obiettivo di analizzare l'avanzamento appena eseguito, valutarne l'organizzazione ed aggiornare le stime dei *task* ancora da completare, in modo tale da poter pianificare al meglio i futuri avanzamenti.

3.4 Progettazione architetturale

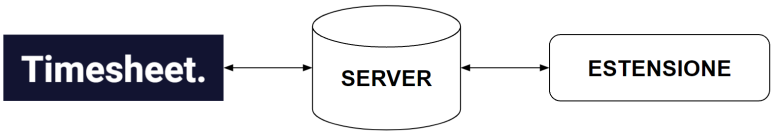


Figura 3.5: Architettura generale dell'intero sistema *Timesheet*.

Come mostrato nella Figura 3.5, l'estensione vive nel dominio dell'applicativo come un elemento a se stante. Non comunica direttamente con la piattaforma principale, ma solamente con il *server*, nel quale inserisce le diverse attività da rendicontare e da pianificare. Queste saranno successivamente visibili in *Timesheet*. L'invio di queste attività viene fatto tramite le *API*, definite in un periodo precedente al mio *stage*.

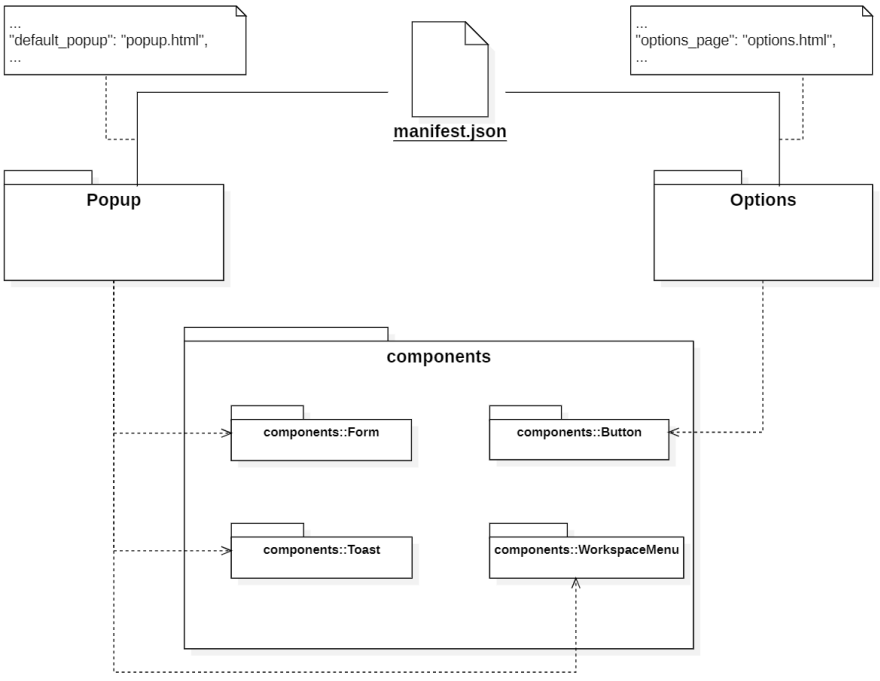


Figura 3.6: Architettura generale dell'estensione.

L'architettura generale dell'estensione, come mostra la Figura 3.6 si basa principalmente sul *Manifest*. Questo ha il compito di comunicare al *browser* la configurazione di base dell'estensione, ovvero indica quali *file* utilizzare per i diversi compiti e quali permessi e capacità sono necessari all'estensione perché funzioni correttamente. Non tutte le diverse componenti dell'estensione sono state necessarie per il mio progetto, ho ritenuto utili per il momento soltanto le componenti di:

* **Popup**: definisce l'estensione vera e propria, ovvero ciò che l'utente visualizza durante il suo utilizzo;

* **Options**: definisce la pagina di opzioni dell'estensione, nella quale possono essere inserite delle funzionalità extra-estensione. Nel mio caso la pagina opzioni è stata utilizzata per effettuare il *login*, *logout* e per accedere direttamente all'area di gestione del proprio profilo in *Timesheet*.

Ciascuna di queste componenti principali viene quindi vista come un elemento indipendente, nel quale vengono utilizzati componenti base per definirne le diverse parti. Questi componenti base sono generici, in modo tale da poter essere riutilizzati il più possibile. Facendo riferimento sempre alla Figura 3.6, tali componenti base sono:

* **Button**: utilizzando per definire ogni bottone utilizzato all'interno dell'estensione. Nello specifico è stato utilizzato un bottone per effettuare tutte le seguenti funzionalità:
  - Effettuare il *login* all'estensione;
  - Effettuare il *logout* dall'estensione;
  - Accedere alla propria pagina personale in *Timesheet*;
  - Accedere alla pagina delle opzioni direttamente dall'estensione;
  - Salvare l'attività inserita e chiudere l'estensione;
  - Salvare l'attività inserita e continuare la rendicontazione;
  - Cancellare le informazioni dell'attività inserite.
* **Form**: contiene i diversi componenti utilizzati per la definizione del *form* di rendicontazione, tali componenti sono:
  - *Input*: utilizzato per la descrizione dell'attività, per la quale bisogna inserire del testo;
  - *Date Picker*: utilizzato per selezionare la data nella quale si è lavorato sull'attività che si vuole rendicontare;
  - *Field*: componente necessario per utilizzare i vantaggi della libreria *Formik*, descritta nella sezione §3.5.3. Il suo scopo principale è fare da contenitore al componente del *form* in modo tale da mostrarne eventuali errori durante la validazione dei dati inseriti;
  - *Select*: utilizzato per selezionare una tra più possibilità disponibili per un campo dati.
* **Toast**: utilizzato per definire dei *dialog* di successo oppure di errore che vengono mostrati nell'estensione al salvataggio dell'attività da rendicontare;
* **Workspace Menu**: utilizzato per definire il menù di selezione dei *workspace*. Nello specifico, è possibile selezionare un *workspace* di lavoro tra quelli disponibili all'utente oppure accedere direttamente alla pagina di gestione dei propri *workspace* nell'applicazione principale.

3.5 Progettazione in dettaglio e codifica

Questa sezione tratta la spiegazione dettagliata di ciascun aspetto principale del progetto, esponendo la motivazione di eventuali scelte fatte.

3.5.1 Login

Uno degli obiettivi principali del progetto è effettuare il *login* all'estensione tramite *SSO*. Il *Single-Sign-On* è una funzione di gestione degli accessi che consente agli utenti di accedere con un unico *set* di credenziali di identità a diversi *account*, *software*, sistemi e risorse. L'*SSO* è un accordo di gestione federata delle identità (*FIM*) che riguarda nello specifico la gestione degli accessi. La federazione delle identità permette l'interoperabilità consentendo all'utente di poter utilizzare un singolo *set* di credenziali di accesso per accedere ad applicazioni, sistemi e servizi diversi. L'*SSO* garantisce che le credenziali dell'utente non siano condivise con l'applicazione, il sistema o con il servizio a cui si accede con uno schema di autorizzazione aperta (*OAuth*) che sostituisce le informazioni di accesso dell'utente con un *token* per accedere.
Come piattaforma di gestione delle identità ho utilizzato *Auth0*, principalmente perché è lo stesso strumento di autenticazione che viene utilizzato in *Timesheet*. La Figura 3.7 descrive il processo di autenticazione utilizzando *Auth0* dove, essendo l'estensione un elemento a se stante nel contesto dell'applicativo *Timesheet*, possiamo vedere il percorso relativo al *domain1.com* come il processo di autenticazione eseguito dalla piattaforma, mentre possiamo vedere il percorso relativo al *domain2.com* come il processo di autenticazione eseguito dall'estensione.

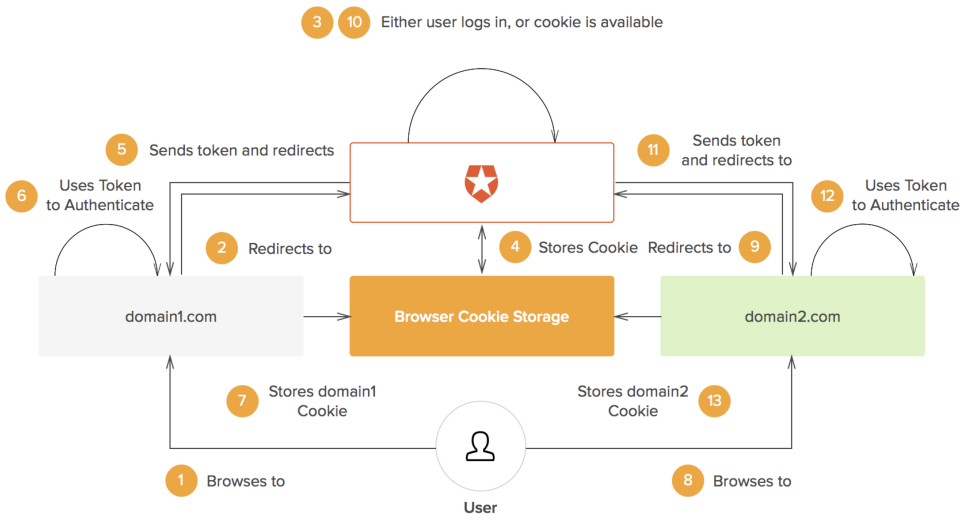


Figura 3.7: Peyrott, S. (2022, 4 febbraio). What Is and How Does Single Sign-On Authentication Work?

Per implementare *Auth0* all'interno dell'estensione, e *React*, ho utilizzato una libreria *Auth0 React SDK* (*auth0-react.js*) sviluppata direttamente da *Auth0*. Questa mette a disposizione un *custom hook* e altri componenti di alto livello in modo tale da poter implementare il *login* tramite *SSO*. Il componente principale utilizzato è *Auth0Provider* che permette di salvare lo stato di autenticazione dell'utente e lo stato di *Auth0*, se è disponibile o meno. Inoltre, questo componente, mette a disposizione

dei metodi di supporto per effettuare il *login* ed il *logout* dell'utente ma non solo, tutti questi metodi sono accessibili tramite lo *useAuth0() hook*.

Durante lo sviluppo di questa funzionalità ho avuto non pochi problemi con il decidere come effettuare il *login* dall'estensione. Il principale problema è che dal *popup* dell'estensione non è possibile aprire nuove finestre nel *browser* oppure reindirizzare la finestra corrente per utilizzare il servizio di *Auth0*. La soluzione alla quale sono arrivato insieme al *tutor* aziendale è utilizzare la pagina delle opzioni per effettuare il *login* ed il *logout*. All'interno dell'estensione, invece, comunico con i sistemi di *Auth0* solamente per avere lo stato di autenticazione dell'utente ed eventualmente richiederne il *token* di accesso, utile per comunicare con il *server* e ottenere le informazioni necessarie sull'utente in questione.

Il processo finale di *login* tramite l'estensione è quindi il seguente:

1. Accedo all'estensione che mi richiede di effettuare il *login*;
2. Vengo reindirizzato alla pagina delle opzioni, dove posso effettuare il *login*;
3. Vengo reindirizzato nuovamente al servizio di autenticazione tramite *SSO* di *Auth0*;
4. Completata la procedura di *login* vengo reindirizzato alla pagina delle opzioni;
5. Adesso sono autenticato e, la prossima volta che utilizzerò l'estensione, visualizzerò il *form* di rendicontazione;

3.5.2 Popup

La pagina di *Popup* è l'elemento principale dell'estensione, la pagina con cui l'utente interagisce direttamente quando utilizza l'estensione. Gli elementi della pagina cambiano a seconda che l'utente sia autenticato o meno. Essendo questa componente contenuta in un *Auth0Provider*, descritto in §3.5.1, è possibile utilizzare lo stato di autenticazione per verificare che l'utente abbia effettuato il *login* (tramite il metodo di supporto *isAuthenticated*) e, successivamente, il metodo di supporto *getAccessTokenSilently* per comunicare direttamente con il servizio di *Auth0* ed ottenere il *token* di accesso dell'utente. Questo *token* è necessario per comunicare con il *server* ed ottenere le informazioni relative all'utente.
Nel caso l'utente non avesse effettuato il *login*, può visualizzare solamente il bottone per eseguire la procedura di autenticazione (come mostrato nella Figura 3.11). Questo, come accennato in §3.5.1, apre una nuova finestra nel *browser* con la pagina delle opzioni dell'estensione, dalla quale è possibile effettuare il processo di *login*.
Altrimenti, se l'utente ha eseguito l'accesso, può vedere le due parti principali dell'estensione (come mostrato nella Figura 3.12), ovvero:

- **Navbar**: contenente il logo dell'applicativo, il bottone per venire reindirizzati alla pagina delle opzioni ed il menu di selezione del *workspace*, dal quale l'utente può selezionare uno dei suoi *workspace* di lavoro oppure venire reindirizzato alla piattaforma principale per eliminarne o crearne di nuovi;
- **Form di rendicontazione**: utilizza la libreria *Formik*, descritta nella sezione §3.5.3;

Dopo ever inserito i dati nel *form* e premuto il bottone di salvataggio le informazioni vengono preparate in un oggetto che successivamente viene utilizzato per definire la *query* che verrà inviata al *server* tramite *API* per il salvataggio dell'attività.
Infine, l'utente visualizzerà un *dialog* che lo informerà del successo o meno del salvataggio.

3.5.3 Formik

Formik è una libreria per *React* che ha lo scopo di aiutare nella creazione e gestione di *form*. Nello specifico è utile soprattutto per tre aspetti altrimenti complicati da gestire:

* Inserire ed estrarre i valori dallo stato del *form*;
* Validare le informazioni e visualizzare eventuali messaggi di errore;
* Gestire la *submit* del *form*.

Questa libreria cerca di unire tutti questi aspetti in un unico posto, in modo tale da mantenere il codice organizzato e renderne la verifica ed il *refactoring* meno complessi.
Nel mio progetto ho utilizzato la libreria *Formik* per definire il *form* di rendicontazione.
Nella Figura 3.8 possiamo vedere uno schema generale della struttura del form che ho definito per la rendicontazione delle attività.

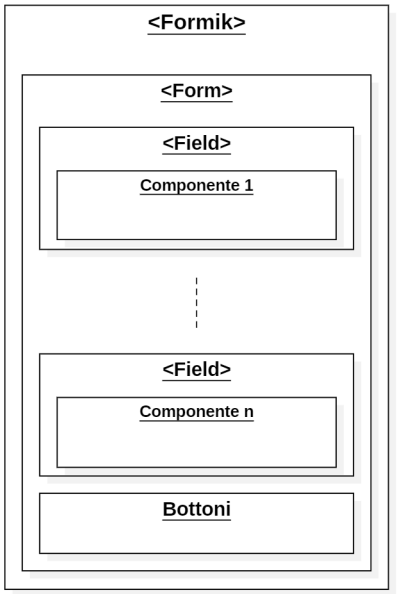


Figura 3.8: Struttura generale delle componenti del *form* di rendicontazione

Questo schema è composto dai seguenti elementi:

* **Formik**: componente che aiuta nella creazione dei *form*. Utilizza un *render props pattern* e si occupa di eseguire tutte le operazioni di gestione del *form*, come ad esempio validare i dati inseriti, generare i valori iniziali dei campi, inviare i valori inseriti e gestire gli eventuali errori. Ha quindi lo scopo di generare e gestire lo stato del *form*, mettendo a disposizione vari metodi di supporto utilizzabili al suo interno;

* **Form**: componente che contiene i diversi campi dati ed i bottoni di invio e cancellazione del *form*. Si collega automaticamente agli *hooks* di *Formik*: *handleSubmit* e *handleReset*. Inoltre tutti i *props* di *Formik* sono passati di *default*;

* **Field**: componente che ha il compito di fare da *wrapper* ai componenti di base in modo da gestirne la validazione e poterne mostrare eventuali errori;

* **Componenti base**: relativi a ciascun campo del form, ad esempio *select* oppure *input*.

3.5.4 Metadati

Uno degli obiettivi del progetto è, in fase di rendicontazione dell'attività, salvare anche dei metadati ricavabili dalla pagina in cui mi trovo. Lo scopo dietro all'ottenimento di queste informazioni aggiuntive è poter avere dei *report* di analisi del lavoro più ricchi di dati e quindi più utili per eventuali analisi al fine di migliorare il lavoro. L'importante in questo progetto, più che definire una lista completa di metadati e fornire supporto per tutti gli strumenti utilizzati, era di definire la struttura di base per il salvataggio di queste informazioni aggiuntive. Per questo mi sono concentrato solamente su alcune informazioni ricavabili da *GitLab*. Nonostante ciò ho comunque dato molta importanza all'estensibilità di ciò che ho implementato, proprio perché so che uno dei futuri obiettivi per il progetto successivo al mio *stage* sarà supportare altri strumenti e non solamente *GitLab*.
Per fare ciò ho definito una gerarchia di tipi di metadato, come mostra la Figura. Per creare il giusto oggetto contenente i metadati relativi all'attività, ho utilizzato il *design pattern* creazionale *builder*. Questo ha l'obiettivo di istanziare il costruttore corretto che, successivamente, andrà a creare il giusto tipo di metadato, come viene mostrato nella seguente Figura.

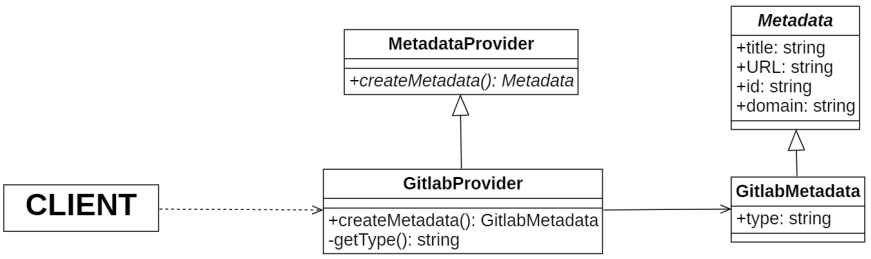


Figura 3.9: Diagramma UML della struttura dei metadati

3.6 Verifica e Validazione

La verifica e la validazione del progetto si sono basate sull'analisi statica e dinamica del codice prodotto ad ogni iterazione. L'analisi statica è il processo di valutazione di un sistema o di un suo componente basato sulla sua forma, sulla sua struttura, sul suo contenuto o sulla documentazione di riferimento. Ciò significa che la valutazione avviene a prescindere dall'esecuzione del sistema o dell'oggetto che si sta testando.

Questo tipo di analisi è stato il primo controllo che veniva effettuato sul codice con l'intento di riscontrare anomalie nel prodotto *software*. Durante il mio *stage*, al termine dell'implementazione di ogni funzionalità, effettuavo insieme al *tutor* aziendale una *code review* nella quale discutevamo di come avevo implementato la funzionalità in questione, analizzando i *file* prodotti ed il codice scritto. Questa operazione ha portato alla luce varie volte dei difetti, talvolta di tipo logico (ad esempio operazioni eseguite in modo scorretto), altre invece legati alle *best practice* che vengono usate da Wavelop Srls (ad esempio il non utilizzo di unità di misura comuni a tutto il progetto).
L'analisi dinamica è il processo di valutazione di un sistema o di un componente durante la sua esecuzione. È una delle parti essenziali del processo di verifica. Questa analisi viene eseguita principalmente tramite l'esecuzione di *test*, ma per il mio *stage* mi sono limitato al collaudo delle funzionalità implementate, in quanto l'azienda non ha richiesto l'implementazione di *test* specifici e anche perché il tempo per riuscire a svolgerli tutti non era sufficiente. Questi collaudi hanno avuto lo scopo di verificare tutti i possibili utilizzi dell'estensione, analizzandone il comportamento nella maggior parte di casi possibili. Durante il mio *stage* veniva svolto sia al termine di ogni implementazione insieme al *tutor* aziendale, che ad ogni *sprint review* facendo vedere il comportamento dell'intero sistema, comprensivo delle nuove implementazioni, a tutte le persone coinvolte nel progetto.
La Figura 3.10 riporta i diversi tipi di analisi effettuati e la loro organizzazione all'interno dell'attività di sviluppo.

Figura 3.10: Analisi e collaudi nell'attività di sviluppo, tratta da: "*Sprint Scrum: tutto ciò che devi sapere*", Atlassian

3.7 Attualizzazione dei rischi

Durante il progetto ho effettivamente riscontrato qualche piccolo problema, sia dal punto di vista tecnologico che organizzativo. Nello specifico:

* Nella quinta settimana non sono riuscito a terminare tutti i *task* assegnati allo *sprint*. Ho fatto un'analisi approfondita insieme al *tutor* aziendale delle possibili motivazioni e come risolvere il problema. La motivazione principale che abbiamo individuato è stata la scorretta stima del tempo necessario per eseguire i *task* del *backlog* dello *sprint*. Questa è data soprattutto dal non aver preso in considerazione eventuali rallentamenti che avrei potuto avere con l'utilizzo di alcune nuove tecnologie.

A seguito di quest'analisi ho aggiornato le stime dei *task* del *backlog* di prodotto che sarebbero stati assegnati allo *sprint* successivo e definito un limite di tempo massimo ai rallentamenti che affronto oltre il quale avrei dovuto obbligatoriamente fare nota al *tutor* aziendale. In questo modo ho fatto esperienza provando a risolvere da solo eventuali problematiche, ma allo stesso tempo non ho perso troppo tempo che altrimenti avrebbe intralciato la pianificazione.

* Ho avuto difficoltà nell'implementazione del processo di autenticazione. Questo è dato sia dalla poca esperienza che avevo con le tecnologie utilizzate, ma soprattutto con il poco supporto che c'è da parte di *Auth0* per le estensioni *Chrome*. Questo poco supporto è risultato in poche soluzioni possibili, tutte molto complesse e certe volte obbligate per via di funzionalità che non funzionano nell'estensione. Ad esempio dall'estensione non è possibile aprire una nuova finestra nel *browser*, ma solamente aprire la pagina delle opzioni.
  Per risolvere questa problematica ho chiesto l'aiuto del *tutor* aziendale con il quale ho definito una soluzione accettabile che si adatta agli obiettivi iniziali.

3.8 Copertura dei requisiti

Le funzionalità prodotte durante l'attività di progetto si possono riassumere in:

* Gestione degli utenti assegnati ai diversi progetti;
* *Login* con *SSO* effettuabile tramite l'estensione;
* Rendicontazione e pianificazione delle attività tramite l'estensione;
* Informazioni aggiuntive nella pagina delle attività;
* Salvataggio di informazioni aggiuntive dalla pagina dell'attività rendicontata.

Il progetto che ho svolto non è stato pensato per essere completato durante lo *stage*, ma per un periodo ben più lungo, andando di pari passo all'applicativo principale. Pertanto l'obiettivo finale, più che produrre del codice definitivo, era studiare le funzionalità implementabili e le eventuali problematiche nello sviluppo di un'estensione, andando a definire la struttura iniziale sulla quale si baserà l'estensione completa. Di conseguenza durante il progetto non ho fatto *testing*, che andrà invece svolto più avanti.

3.8.1 Quantità dei prodotti

Per fare luce sulla sulla dimensione del progetto e sulla quantità dei prodotti, durante il mio *stage* ho prodotto circa 3000 linee di codice. Inoltre, la struttura finale delle cartelle prodotta è la seguente:

```
chrome-extension/
|__utils/
|__pem/
|__src/
|  |__assets/
|  |__components/
|  |  |__Button/
|  |  |__Form/
|  |  |__Toast/
```

```
|  |  |__WorkspaceMenu/
|  |__config/
|  |__i18n/
|  |__models/
|  |__pages/
|  |  |__Options/
|  |  |__Popup/
|  |__services/
|  |__theme/
|  |__utils/
|__package.json
|__README.md
|__webpack.config.json
|__tsconfig.json
```

dove le cartelle:

* **assets**: contiene tutti i *file* di supporto all'interfaccia dell'estensione, come i font e le immagini;

* **components**: contiene i diversi componenti base che vengono riutilizzati tra le diverse pagine;

* **config**: contiene i *file* che definiscono i diversi indirizzi delle *API* con i quali l'estensione invia le varie richieste al *server*. Questi indirizzi al momento non sono diversificati, ma in futuro verranno divisi tra produzione e sviluppo una volta che l'estensione e l'applicazione principale verranno pubblicate;

* **i18n**: contiene tutti i *file* necessari per le diverse traduzioni dell'interfaccia. Al momento l'estensione è stata progettata per essere utilizzata interamente sia in italiano che in lingua inglese;

* **models**: contiene tutti i modelli, ovvero le diverse interfacce dei tipi, delle entità e dei servizi;

* **pages**: ognuna delle cartelle al suo interno definisce una pagina utilizzata dall'estensione. Per il mio progetto, è stata prevista soltanto l'implementazione delle pagine *Popup* e *Options*, relative rispettivamente alla parte principale con cui l'utente interagisce e alla pagina delle impostazioni;

* **services**: contiene tutti i servizi utilizzati dall'estensione per recuperare le informazioni necessarie dal *server*. Ad esempio, per le attività, sono stati implementati i servizi di gestione base: salvataggio, cancellazione, lettura e aggiornamento. Per utilizzarli è necessario avere l'*id* utente, il *token* di accesso e l'oggetto attività da inviare (nel caso volessimo aggiornare oppure salvare un'attività) che viene convertito nel *body* della chiamata *API* al *server*;

* **theme**: contiene tutti i *file* necessari per definire il tema dell'estensione. Questo tema viene usato per uniformare tra di loro le diverse pagine al livello estetico (ad esempio i codici colore, font, grandezza del carattere e unità di spaziatura);

* **utils**: contiene tutti i *file* di utilità e di configurazione necessari per far funzionare il progetto.

3.8.2 Qualità dei prodotti

Per valutare la qualità di quanto ho prodotto, nonostante la sola presenza di analisi statica e collaudi che hanno limitato gli aspetti da valutare, ho utilizzato le seguenti metriche:

Metrica	Valore massimo	Valore raggiunto
Copertura funzionalità	100%	80%
Copertura requisiti	100%	93%
Parametri per metodo	4	4
Nidificazione	5	8
Codice per metodo	50	25

Tabella 3.2: Metriche di qualità

dove:

* **Copertura funzionalità**: misura la percentuale di funzionalità implementate rispetto a quelle stabilite all'inizio dello *stage*;
* **Copertura requisiti**: misura la percentuale di requisiti soddisfatti rispetto a quelli stabiliti all'inizio dello *stage*;
* **Parametri per metodo**: misura il numero massimo di parametri passati ad un metodo;
* **Nidificazione**: misura il livello massimo della nidificazione nel codice;
* **Codice per metodo**: misura il numero massimo di linee di codice in un metodo.

La nidificazione è di molto maggiore rispetto al valore massimo soprattutto per via del *form* di rendicontazione che è composto da molti componenti nidificati tra di loro; ciò porta facilmente ad un innalzamento di tale valore.

3.9 Visione d'insieme

Il principale scopo dell'estensione è consentire agli sviluppatori di un *team* di rendicontare e pianificare le proprie ore di lavoro per specifici *task*. Al momento è pensata per essere usata nel *workflow* aziendale in Wavelop Srls.
L'estensione mette a disposizione le seguenti funzionalità:

* **Utilizzabile in *Google Chrome***: gli sviluppatori possono scaricare l'estensione dalla *repository* interna dell'azienda, seguire le istruzioni per la compilazione dei *file* ed, infine, caricare quanto prodotto nel *browser*. Da qui saranno in grado di utilizzare l'estensione da una qualsiasi pagina di *Google Chrome* e rendicontare o pianificare le proprie attività;
* ***Login* tramite *SSO***: al primo accesso all'estensione, l'utente dovrà effettuare il processo di autenticazione che utilizza un servizio di *Single-Sign-On* che ne facilità il completamento;

* ***Form* di Rendicontazione**: nella sezione principale dell'estensione, dopo aver eseguito l'accesso, è possibile utilizzare il *form* di rendicontazione sia per rendicontare le attività eseguite (quindi riferite al passato) che pianificare delle ore di lavoro per delle attività (quindi riferite al futuro). Questa funzionalità è usufruibile da qualsiasi pagina del *browser*; è comunque consigliato farlo direttamente dalla pagina dell'attività in questione, in modo tale da salvare anche il riferimento diretto alla pagina e rendere le attività future più efficienti;

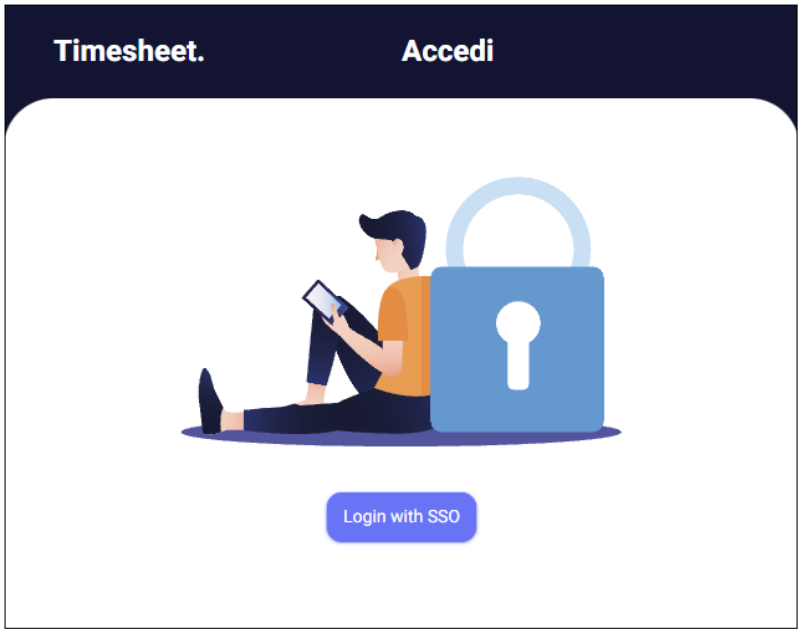


Figura 3.11: Interfaccia di Login dell'estensione

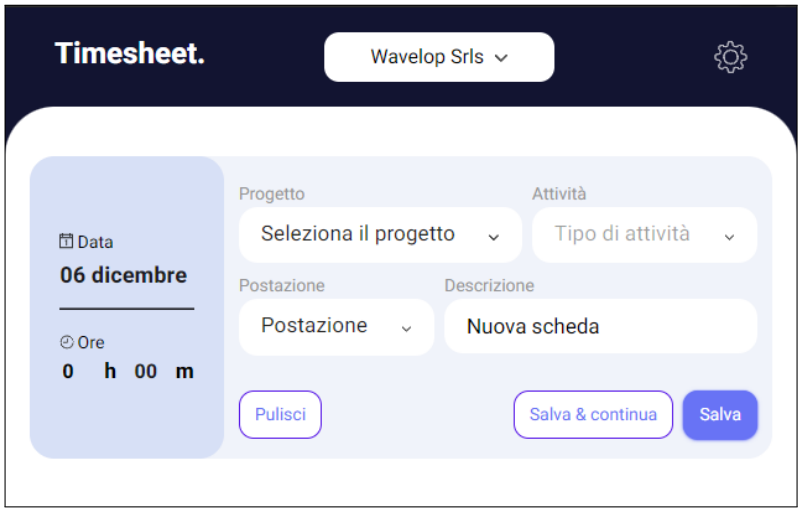


Figura 3.12: Interfaccia principale dell'estensione

Capitolo 4

Valutazione retrospettiva

Questo capitolo ha l'obiettivo di esporre l'analisi e la valutazione dello stage svolto. Inoltre verranno discusse le mie considerazioni personali rispetto alle aspettative e rispetto alla differenza tra le competenze accademiche e quelle utili nel mondo del lavoro.

4.1 Resoconto degli obiettivi

4.1.1 Obiettivi funzionali

Nella Tabella 4.1 ho riportato la lista degli obiettivi funzionali stabiliti con l'azienda ed esposti in §2.3.1. Per ogni obiettivo ho indicato il tipo, ovvero se si tratta di un obiettivo obbligatorio, desiderabile o facoltativo, e l'esito, ovvero se è stato soddisfatto o meno.

Obiettivo	**Tipo**	**Esito**
Apprendimento delle tecnologie	Obbligatorio	Soddisfatto
Assegnare un utente ad un progetto	Obbligatorio	Soddisfatto
Rimuovere un utente da un progetto	Obbligatorio	Soddisfatto
Visualizzare gli utenti di un progetto	Obbligatorio	Soddisfatto
Creare una estensione per *Google Chrome*	Obbligatorio	Soddisfatto
Login tramite l'estensione	Obbligatorio	Soddisfatto
Visualizzazione dei progetti assegnati	Obbligatorio	Soddisfatto
Scheduling dell'attività relativa alla pagina corrente	Obbligatorio	Soddisfatto
Gestione dei ruoli degli utenti	Desiderabile	Non Soddisfatto
Memorizzazione metadati di un'attività in base alla sorgente	Desiderabile	Soddisfatto
Implementazione di *Feature Flag*	Desiderabile	Non Soddisfatto

Obiettivo	Tipo	Esito
Multilingua	Facoltativo	Soddisfatto
Integrazione ad un sistema di fatturazione elettronica	Facoltativo	Non Soddisfatto
Refactor form	Facoltativo	Soddisfatto

Tabella 4.1: Soddisfacimento degli obiettivi funzionali dello *stage*

Dalla tabella è possibile vedere come ho soddisfatto tutti gli obiettivi obbligatori, mentre nel complesso ho soddisfatto 11 obiettivi sui 14 totali, ovvero circa l'80% dei requisiti. Si può inoltre notare come ho soddisfatto alcuni obiettivi facoltativi a discapito di altri desiderabili, nonostante fossero meno importanti. Ciò è avvenuto a seguito di discussioni fatte durante lo sviluppo del progetto, dove è stato deciso di dare maggiore priorità ad alcuni obiettivi anziché ad altri.

4.1.2 Obiettivi personali

Gli obiettivi personali che mi ero prefissato all'inizio dello *stage* (discussi in §2.4) erano per la quasi totalità legati all'acquisire esperienza. Tali obiettivi sono stati pienamente soddisfatti. Wavelop Srls si è rivelato un ambiente lavorativo stimolante, che mette insieme persone giovani ed appassionate dello sviluppo *software*. Questo mi ha permesso di instaurare nuovi rapporti ed ha esaltato l'intera esperienza all'interno dell'azienda.
Ho avuto la possibilità di migliorare le mie conoscenze riguardo all'utilizzo di *React* e *Node.js*; tecnologie che avevo utilizzato durante il corso di Ingegneria del *Software*, ma che durante questi due mesi ho potuto davvero approfondire ed imparare, anche grazie al costante supporto del *tutor* aziendale. Inoltre, ho avuto l'opportunità di apprendere tecnologie nuove come *MongoDB* e *TypeScript*, entrambe molto importanti nel contesto dello sviluppo di applicazioni *web*.
Questo *stage* mi ha permesso non solo di acquisire ulteriori conoscenze tecnologiche, ma anche di poter vivere per la prima volta il mondo del lavoro all'interno di un'azienda *software*. Questo riguarda soprattutto le dinamiche interne all'azienda, come gli *stand-up meeting* e le *review* settimanali. Ho trovato molto formativo poter vedere la pianificazione e la gestione del lavoro all'interno dei diversi *team*, come ad esempio la divisione e l'assegnazione della responsabilità ad ogni singolo componente del gruppo, oppure le discussioni riguardo specifici aspetti di progettazione e *design*.
Tutto ciò mi ha dato sicuramente l'occasione di accrescere la mia esperienza sia dal punto di vista professionale, che da quello personale.

4.2 Bilancio formativo

Lo stage mi ha permesso di apprendere cose nuove ed innovative, mai utilizzate in precedenza. La mia formazione si è arricchita delle seguenti tecnologie:

* **React**: prima dello *stage* avevo utilizzato questo *framework* solamente durante il corso di Ingegneria del *Software*. Però, ora che ho avuto la possibilità di ampliare le mie conoscenze a riguardo, ho potuto vedere la diversa metodologia di utilizzo rispetto al primo approccio, notandone tutti i vantaggi;

* **TypeScript**: ho imparato da zero questo nuovo linguaggio. Ciò mi ha dato inoltre la possibilità non solo di capirne vari aspetti e conoscere nuove metodologie di programmazione, ma anche di accrescere le mie conoscenze riguardo a *JavaScript*, il linguaggio su cui si basa *TypeScript*;

* **Node.js**: prima dello *stage* avevo avuto la possibilità di utilizzare questa tecnologia durante il corso di Ingegneria del *Software*. Ora però ho avuto l'occasione di ampliarne le conoscenze per l'utilizzo lato *server*;

* **MongoDB**: ho potuto imparare come utilizzare un *database* non relazionale, effettuando *query* e studiandone la gestione dei dati.

Ma non solo, vi sono anche metodologie e nuovi strumenti che hanno decisamente arricchito il mio bagaglio di conoscenze:

* **GitLab**: precedentemente allo *stage* avevo avuto esperienza solamente con *GitHub*. Ho avuto così la possibilità di studiarne una valida alternativa, analizzandone le principali differenze, i punti di forza e quelli di debolezza;

* **GitFlow**: ho potuto utilizzare per la prima volta questa metodologia di sviluppo software. Prima dello *stage* avevo sempre fatto uso di metodologie più semplici, ma definitivamente meno efficienti, di sviluppo e gestione del versionamento del codice;

* **Metodologia *Agile Scrum***: non avendo mai avuto la possibilità di essere inserito all'interno di un contesto lavorativo prima dello *stage*, la mia conoscenza riguardo il *project management* era isolata solamente al progetto di Ingegneria del *Software* che utilizza un approccio completamente diverso. Grazie a ciò ho potuto imparare questa nuova tecnica ed ho avuto l'occasione di metterla in atto apprendendone molte cose.

4.3 Considerazioni sul rapporto tra università e lavoro

In questa sezione sono esposte le principali differenze che ho riscontrato tra le competenze accademiche e quelle richieste nel mondo del lavoro.

4.3.1 Ruolo dello stage

Ritengo che lo *stage* rivesta un ruolo molto importante di tramite tra il mondo universitario e quello del lavoro. Le tecnologie ed i metodi usati dalle aziende sono fondamentalmente diverse da quelle proposte ed insegnate in ambiente universitario. Lo *stage* è stata la possibilità di poter sperimentare questi nuovi concetti e vedere in prima persona la sostanziale differenza che sta alla base del mondo accademico e quello lavorativo. In quest'ultimo infatti l'aspetto fondamentale è quello economico; l'adattamento alle esigenze del mercato e la tempestività nel fornire soluzioni di qualità e generare guadagni sono gli obiettivi principali in un'azienda. Per ciò sono dell'idea che lo *stage* assuma un ruolo di grande valore all'interno della carriera universitaria. Avere la possibilità di entrare gradualmente dall'ambiente universitario a quello del lavoro rende il passaggio tra i due molto più semplice.

4.3.2 Considerazioni sui contenuti mancanti

Alcune delle competenze utilizzate durante lo *stage* facevano parte del mio bagaglio culturale. Questo è dato soprattutto dal loro uso durante il corso di Ingegneria del *Software*, anche se in modo tutt'altro che approfondito. Molto spesso però i temi trattati durante le diverse discussioni non facevano parte delle conoscenze apprese durante la mia carriera universitaria. Ho dovuto dedicare non poco tempo per comprendere ed utilizzare tecnologie che durante il corso di studi sono state solamente accennate o svolte superficialmente, per poterle sfruttare durante lo sviluppo del progetto. Nello specifico, alcune mancanze che ho riscontrato nel Corso di laurea in Informatica sono:

* **Applicazioni *web***: il mio *stage* ha in pratica toccato con mano tale aspetto. Lo sviluppo delle *web app* sta ormai diventando l'approccio di base per lo sviluppo di siti *web* di nuova generazione. Pertanto sono dell'idea che sarebbe necessario il suo inserimento all'interno dei temi trattati dal corso;

* **Linguaggi moderni**: durante la laurea triennale gran parte del tempo vengono studiati linguaggi come *C++* oppure *Java*. Durante lo *stage* ho però notato lo scarso utilizzo di questi in un progetto, ai quali sono invece preferiti linguaggi più moderni come *TypeScript*, *JavaScript* oppure i linguaggi funzionali;

* ***Design* grafico**: questo aspetto non è mai stato trattato durante il corso perché considerato di secondo piano. Ritengo invece che il saper sviluppare un'interfaccia grafica sia un aspetto non banale e per nulla da trascurare.

Dal punto di vista di studente vorrei esporre ulteriori aspetti che penso potrebbero alzare il livello del corso di laurea, che rimane comunque di spessore:

* I *design pattern* dovrebbero essere studiati molto prima rispetto a quanto fatto finora. Questo permetterebbe sia di imparare fin da subito il modo corretto di pensare e progettare del *software* che di poter fare più pratica nel loro utilizzo durante i vari progetti durante il corso;

* Incentivare maggiormente il lavoro di gruppo. Questo porterebbe gli studenti a prendere pratica con le dinamiche di lavoro all'interno di un gruppo e con le diverse metodologie utilizzabili

* Cercare di seguire, per quanto possibile, l'innovazione tecnologica, definendo dei corsi riguardo l'utilizzo di tecnologie moderne.

Infine, quello che l'ambiente universitario ha potuto sicuramente darmi sono le metodologie di base da utilizzare per affrontare la velocità del progresso tecnologico. Queste mi hanno permesso, durante lo *stage*, di adattarmi velocemente alle nuove tecnologie e metodologie utilizzate da Wavelop Srls. Il Corso di laurea in Informatica, perciò, riesce a plasmare in modo corretto lo studente, fornendogli gli strumenti per restare costantemente aggiornato. Penso che questo sia, dopotutto, l'aspetto più importante.