Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica

Analisi e sviluppo di una applicazione web per la schedulazione e rendicontazione delle attività aziendali interne

Tesi di laurea

Relatore
Prof.Davide Bresolin

Laureando
Riccardo Pavan

Anno Accademico 2021-2022

Sommario

Questa tesi di laurea descrive il lavoro svolto e i risultati ottenuti durante il periodo di stage presso Wavelop Srls, con sede a Treviso.
L'obiettivo del tirocinio era lo sviluppo di una applicazione web dedicata alla gestione delle attività aziendali interne, in particolare a mostrarle in formato sia tabellare che grafico, permettendo di filtrarle in vari modi.
La tesi si propone inoltre di descrivere la metodologia di lavoro adottata e le tecnologie utilizzate per lo sviluppo della applicazione.

Ringraziamenti

Ringrazio il Prof. Davide Bresolin, relatore della mia tesi, per l'aiuto, i consigli e il supporto fornitomi durante la stesura del lavoro.

Ringrazio i miei genitori e familiari per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio.

Ringrazio i colleghi di Wavelop che mi hanno seguito durante lo stage e mi hanno assicurato un'ottima esperienza di tirocinio.

Ringrazio, infine, i miei amici universitari e non per essermi stati vicini durante questi anni e con cui ho condiviso bellissime esperienze.

Padova, Dicembre 2022 Riccardo Pavan

Indice

Elenco delle figure

Elenco delle tabelle

Elenco degli esempi di codice

Capitolo 1

Descrizione dello stage

1.1 L'azienda

Wavelop S.R.L.S. è un'azienda nata nel 2018 con sede a Treviso che si occupa di consulenza e di progetti web e mobile per piccole e medie imprese e che conta circa 10 dipendenti.
Dal 2020 propone progetti per studenti universitari interessati a un tirocinio formativo che verranno poi inseriti all'interno di un team di lavoro presso l'azienda.

1.2 Modello di svluppo

Wavelop segue un modello di sviluppo *agile* con metodologia *scrum*. Di seguito le caratteristiche che contraddistinguono questo metodo di lavoro:

- il lavoro viene diviso in archi temporali della durata di una settimana, denominati *sprint*;
- all'inizio di ogni *sprint* si tiene lo *sprint planning*, una riunione in cui si chiarisce quali obiettivi, espressi sotto forma di *user stories*, si mira a portare a termine;
- alla conclusione di ogni *sprint* si tiene invece lo *sprint review*, una riunione in cui si discute cosa si è fatto durante lo *sprint*, mostrando i risultati ottenuti, di solito tramite brevi demo;
- l'attività giornaliera inizia con la partecipazione allo *stand-up meeting*, una breve riunione della durata di circa 15 minuti, dove, a turno, ogni membro del team

espone brevemente cosa ha fatto il giorno precedente e cosa intende fare oggi, oltre che mettere al corrente tutti di eventuali ostacoli incontrati nello sviluppo.

Adottare una metodologia di questo tipo permette una comunicazione costante fra i membri del team e i clienti che porta a un tracciamento chiaro e preciso dell'avanzamento del progetto, mettendo subito in evidenza eventuali incomprensioni, permettendo quindi delle correzioni tempestive, in accordo col cliente.

1.3 Il progetto di stage

L'azienda ha già iniziato a sviluppare un'applcazione web, denominata *Timesheet*, destinata a uso interno, al fine di rendicontare e pianificare le attività dei vari componenti dell'azienda.
In particolare è già implementata la funzionalità di inserire attività singole nel database. Manca, invece, la possibilità di mostrare, filtrandole a seconda della volontà dell'utente, le varie attività memorizzate, ottenendo quindi dati utili come la quantità di ore dedicate a un cliente in un determinato arco di tempo.
Queste funzionalità dovranno essere raggruppate nella sezione *Reports* dell'applicazione. Lo scopo finale del progetto è quindi quello di completare la sezione *Reports*, sia per quanto riguarda la componente di frontend che per quella di backend.

1.4 Requisiti

Nel piano di lavoro sono stati redatti requisiti specifici che dovranno essere soddisfatti nel periodo di stage. Essi sono categorizzati in base alla importanza che ricoprono:

Obbligatori Requisiti ad alta priorità, la cui importanza è primaria per la riuscita del progetto:

- apprendimento delle tecnologie di sviluppo come React e Node.js e per il versionamento del progetto come git;
- gestione filtri avanzati di ricerca delle attività;
- visualizzazione a tabella delle attività filtrate;
- generazione file CSV delle attività filtrate.

Desiderabili Requisiti a media priorità, non necessari per il completamento dello stage ma che comunque, se soddisfatti, contribuiscono notevolmente alla buona riuscita del progetto:

- generazione file PDF delle attività filtrate;
- visualizzazione attività tramite grafico;
- salvataggio preset filtri per ricerche future.

Facoltativi Requisiti a bassa priorità, portano un valore aggiunto allo stage:

- visualizzazione widget laterale con statistiche relative all'utente;
- gestione responsive della piattaforma.

1.5 Pianificazione

Lo stage ha avuto luogo nel periodo di tempo dal 25 Luglio 2022 al 16 Settembre 2022, per un totale di 312 ore. Sono state inoltre definite delle attività specifiche per ogni settimana di stage, rappresentate di seguito:

Prima settimana - Dal 25 al 29 Luglio

- discussione requisiti relativi al sistema da sviluppare con le persone coinvolte nel progetto;
- introduzione alla cultura aziendale;
- formazione sulle tecnologie adottate;
- prendere confidenza con la struttura già esistente e assegnazione degli strumenti necessari;
- analisi dei requisiti.

Seconda settimana - Dall'1 al 5 Agosto

- analisi dei requisiti;
- progettazione architetturale.

Terza settimana - Dall'8 al 13 Agosto

- progettazione architetturale;
- analisi e definizione user stories per lo sprint successivo.

Dalla quarta alla ottava settimana - Dal 16 Agosto al 16 Settembre

- sviluppo delle user stories assegnate;
- sprint review con il referente e le altre persone coinvolte nel progetto;
- analisi e definizione user stories per lo sprint successivo.

Inoltre, nell'ultima settimana, è previsto un collaudo finale di ciò che è stato fatto.

1.6 Tecnologie utilizzate

È stato utilizzato lo stack tecnologico tipicamente scelto dall'azienda, composto da: *Ract*, *Node.js* e *MongoDB*, di seguito descritte nel dettaglio, una per una.

1.6.1 React

React[1] è una libreria Javascript open-source component-based per creare interfacce utente mantenuta da *Meta*. Adotta l'utilizzo di una sintassi che estende il linguaggio Javascript: il *Javascript Synstax Extension* (JSX), che rende semplice e intuitivo creare

[1] *Sito ufficiale di React*. URL: https://reactjs.org/.

i componenti che formano la pagina, come mostrato nell'esempio di codice 1.1 .

Esempio di codice 1.1: Esempio di utlizzo di codice JSX

```
1  const name = 'Riccardo Pavan';
2
3  const ShowName = () => {
4    return (
5      <div>Mi chiamo {name}</div>
6    );
7  };
```

Un'altra particolarità di React sono gli *hooks*, particolari funzioni che permettono di "agganciarsi" a varie funzionalità di *lifecycle* che React offre senza dovere usare classi.

1.6.2 Node.Js

Node.Js[2] è un *runtime environment* open-source basato su Javascript.
La sua architettura orientata agli eventi permette alle operazioni di input/output di essere asincrone ed è consona soprattutto per applicazioni scalabili e che necessitano di un grande *throughput* .
L'utilizzo di Node.Js permette di utilizzare Javascript anche lato server, permettendo quindi di usarlo come unico linguaggio nello sviluppo di un'applicazione web, dando vita al paradigma detto *Javascript everywhere*.

1.6.3 MongoDB

MongoDB[3] è un *database management system* open-source non relazionale, orientato ai documenti, questi ultimi in formato BSON, simile a JSON.
La struttura non relazionale permette una maggiore libertà nello strutturare il database, utile quando si ha a che fare con molti dati molto diversi fra loro e che tendono a cambiare nel tempo.

[2] *Sito ufficiale di Node.Js.* URL: https://nodejs.org/en/.
[3] *Sito ufficiale di MongoDB.* URL: https://www.mongodb.com/.

Capitolo 2

Gli sprint

2.1 Sprint 1

Il primo sprint è stato dedicato principalmente ad ambientarsi al clima aziendale e alle tecnologie che verranno utilizzate più avanti.
Ho quindi conosciuto gli altri membri del team e mi è stato fornito materiale da studiare inerente ai seguenti argomenti:

- metodologia agile e SCRUM, informazioni su termini come user stories ed epics;
- workflow da utilizzare nel repository condiviso, convenzioni su nomine dei commit;
- riunioni di routine da svolgere quotidianamente o settimanalmente;
- Javascript, React, CSS, Node.JS, MongoDB e programmazione asincrona;

Tutta la documentazione fornita si è rivelata essere pienamente comprensiva e utile, soprattutto per quanto riguarda le tecnologie, buona parte delle quali non avevo mai usato.

Oltre a formarmi sugli argomenti precedentemente elencati, ho anche impostato l'ambiente di lavoro necessario sia a far girare la app che a svilupparla.
In particolare ho utilizzato i seguenti strumenti:

- Docker, per poter utilizzare la immagine del database della applicazione;
- Studio 3T, una GUI per interfacciarsi a database MongoDB.

Tramite riga di comando potevo poi far girare sia la parte client che la parte server in locale, permettendomi di modificare codice e avere feedback visivo immediato.

Infine ho redatto una bozza di un documento atto a elencare le user stories per avere un'idea più chiara di ciò che avrei dovuto fare negli sprint successivi.
Seppure il documento non fosse ancora completo è risultato subito chiaro che sarebbe stata necessaria una libreria per creare grafici. Ho quindi dedicato l'ultimo giorno della settimana a ricercare delle librerie Javascript per grafici, evidenziandone vantaggi e difetti per potere poi giungere a una decisione con gli altri membri del team.

2.2 Sprint 2

Durante il secondo sprint ho redatto la versione conclusiva del documento relativo alle user stories, approvato poi dal mio relatore esterno.
All'inizio di ogni capitolo relativo a uno sprint indicherò le user stories su cui andrò a lavorare.

Per concludere la settimana ho lavorato a dei piccoli bugfix e features, anche per familiarizzare con la web app e i pattern utilizzati. In particolare ho:

- sistemato un problema con la immagine nella pagina di login, che non si rimpiccioliva nel caso la finestra facesse lo stesso;
- fatto sì che, nella pagina utente, nel caso quest'ultimo non la avesse inserita, venissero mostrate le iniziali dell'utente invece che nulla;
- implementato una finestra di dialogo che semplicemente chiede di confermare o annullare un'azione.

Seppure i primi due problemi fossero molto semplici e risolvibili principalmente con del codice CSS, creare la finestra di dialogo mi è stato utile sia a familiarizzare con React, che non avevo mai utilizzato, ma anche a capire meglio come fosse strutturata la applicazione.

2.2.1 Casi d'uso individuati

Ho inoltre creato dei diagrammi UML dei casi d'uso, di cui i più significativi sono elencati di seguito.

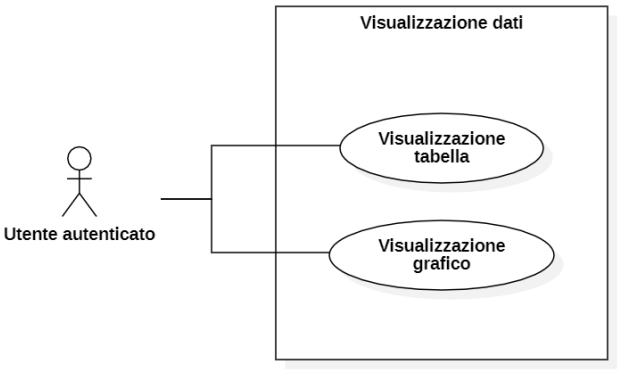

Figura 2.1: Use case per la visualizzazione dati

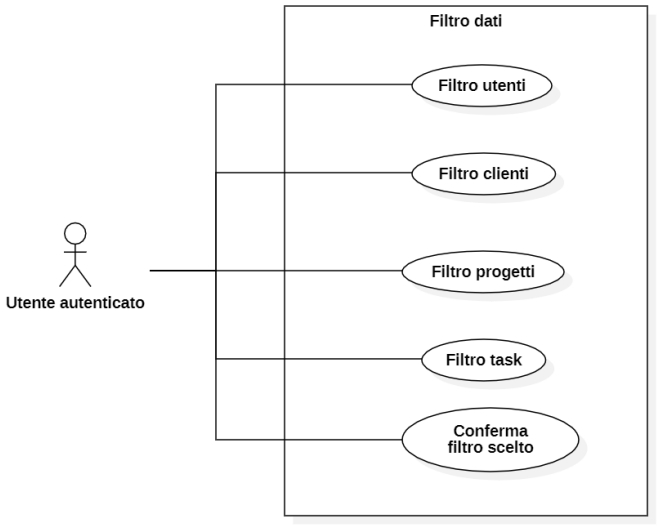

Figura 2.2: Use case per il filtro dati

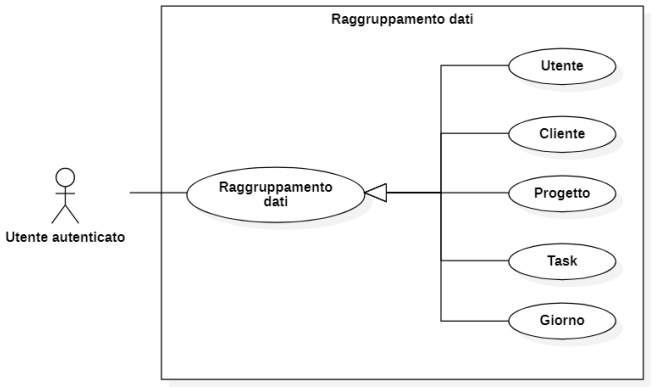

Figura 2.3: Use case per la visualizzazione dati

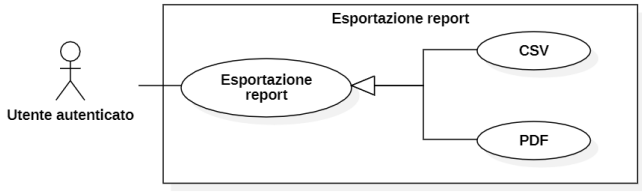

Figura 2.4: Use case per la visualizzazione dati

2.3 Sprint 3

2.3.1 User stories assegnate

Anche se inizialmente era un'attività prevista per la quarta settimana ho iniziato a lavorare effettivamente al progetto durante il terzo sprint.
Utilizzando il documento precedentemente menzionato previste mi sono state assegnate le user stories da completare durante la settimana. Ognuna di esse presenta i seguenti elementi:

- nome della user story;
- un numero intero positivo che indica il "peso" in termini temporali che ci si aspetta che la user story abbia. Questi numeri seguono l'andamento della serie di Fibonacci e sono stati utili per allocare meglio le varie user stories all'interno degli sprint. Questi numeri vanno solitamente da 1 (task da meno di un'ora) a non più di 13 (task per il quale può essere necessaria fino a una settimana);
- breve descrizione di ciò che si deve ottenere una volta conclusa la user story, dal punto di vista dell'utente;
- elenco di task, suddivisi tra frontend e backend, che dicono in modo più specifico e tecnico cosa fare per concludere la user story;

I quali verranno presentati nella forma:

Nome user story (Tempo previsto per concludere la user story)

Descrizione dela user story.

Tasks:

- Task1;
    - subTask1;

Visualizzazione tabella (5)

Come utente autenticato che si trova nella sezione "Reports", voglio poter vedere la tabella riassuntiva delle ore spese.

Tasks:

- implementare il componente principale della pagina Reports;
- implementare il componente della tabella, anche per quanto riguarda il suo stile;
- implementare la chiamata al backend per richiedere i dati da mostrare nella tabella;
- implementare la logica di query nel backend per soddisfare la richiesta dati.

Paginazione tabella (5)

Come utente autenticato che si trova nella sezione "Reports", voglio avere la tabella paginata, in modo da non avere troppe righe nella stessa schermata.

Tasks:

- modificare la chiamata al backend già implementata per paginare i dati;
- aggiungere bottoni per passare da una pagina all'altra della tabella;
- aggiungere opzione per decidere quante righe includere in una pagina.

2.3.2 Visualizzazione tabella

Innanzitutto ho dovuto aggiungere all'applicazione la sezione Reports in quanto, seppure fosse già presente la tab per raggiungerla, non c'erano nè il percorso nè il componente della pagina. A tal scopo ho aggiunto sia il percorso per raggiungere la sezione, `/reports`, al file contenente i routing privati dell'applicazione e ho creato il componente principale alla base della sezione Reports, per ora contenente solo il titolo. Per permettere l'ottenimento dei dati, a lato frontend ho fatto uso di tre degli hooks che React già offre di base: `useState`, `useEffect` e `useCallback`, come nell'esempio di codice 2.1:

Esempio di codice 2.1: Esempio di fetch dati da backend e memorizzazione in React state

```
1   const [data, setData] = useState([]);
2
3   const fetchData = useCallback(async () => {
4     const resultData = await
5     /* Logica per la richiesta dei dati */
6     if (resultData) {
7       setData(resultData);
8     }
9    })
10
11  useEffect(() => {
12    fetchData();
13  }, [fetchData]);
```

`useEffect` indica che `fetchData` verrà chiamata dopo ogni rendering del componente, la quale otterrà i dati da mostrare in formato tabellare, che verranno salvati nel React state `data` tramite `useState`. `useCallback` restituisce una funzione `fetchData` memoizzata, permettendo di non chiamarla nuovamente a ogni singolo re-rendering della pagina.
Per quanto riguarda la logica per la richiesta dei dati ho aggiunto un endpoint `/report` che risponde a una chiamata HTTP GET, ottenendo le *attività* di un utente. Un'attività è un oggetto composto dai seguenti campi:

- `id`: l'id univoco dell'attività;
- `user`
  - `id`: l'id univoco dell'utente;

    - `fullName`: il nome dell'utente;

- `project`
    - `id`: l'id univoco del progetto a cui l'attività partiene;
    - `name`: il nome del progetto;
- `client`
    - `id`: l'id univoco del cliente che commissiona il progetto;
    - `name`: il nome del cliente;
- `workspace`: il workspace dell'attività;
- `task`: il task che l'attività richiede;
- `work`: la durata in ore del'attività;
- `description`: breve descrizione dell'attività (opzionale);
- `day`: data in cui si è svolta l'attività;
- `workplace`: luogo in cui si è svolta l'attività.

Queste sono tutte le informazioni che la sezione report deve poter mostrare all'utente secondo i requisiti accordati.
L'endpoint `/reports` richiede inoltre dei query params aggiuntivi:

- `from`: data prima della quale non si vuole nessuna attività;
- `to`: data dopo della quale non si vuole nessuna attività;
- `sort`: può assumere il valore 1 o -1. Nel primo caso i risultati della query saranno ordinati in modo ascendente, altrimenti in modo discendente;
- `token`: il token di accesso dell'utente.

A lato backend ho aggiunto la query per ottenere i dati da database.

Manca solo mostrare i dati e, a tal scopo, ho creato due nuovi React components, `ReportTable` e `ReportRow`.
Il primo racchiude la struttura della tabella, con tanto di header, mentre il secondo costituisce una riga della stessa. A seconda di quanti elementi saranno ottenuti dal backend, `ReportTable` chiamerà `ReportRow` un numero adeguato di volte, creando una tabella.

2.3.3 Paginazone della tabella

Ora la tabella è visibile, ma per evitare lunghi scroll verticali è necessario creare una funzionalità di paginazione, suddividendo quindi la tabella in parti più piccole, navigabili tramite una apposita interfaccia.
Sono possibili diverse implementazioni per una funzionalità di questo tipo, in particolare:

- lato client, si richiedono tutti i dati con una sola chiamata, per poi mostrarne solo la quantità richiesta. Se l'utente vuole vedere altri dati è sufficiente prenderli da quelli già richiesti;

- lato server, si richiede una piccola quantità di dati, ad esempio solo 10 righe, che verranno poi mostrate. Se l'utente vuole vedere le successive 10 sarà necessario effettuare una nuova chiamata e poi mostrarle.

In accordo con il team, ho optato per la seconda opzione in quanto si è deciso di dare priorità a non sovraccaricare troppo il client, anche perchè è facile arrivare ad avere centinaia se non migliaia di attività anche in periodi di tempo relativamente brevi.
Ho quindi aggiunto due React components:

- `ReportTablePagination`: consiste in un menù a tendina che permette di scegliere quante righe la tabella potrà avere al massimo;

- `ReportTableNavigation`: consiste in un numero dinamico di bottoni che permettono le seguenti azioni:

  · andare alla prima/ultima pagina;

  · andare alla pagina precedente/successiva;

  · andare a una pagina specifica.

Al component principale `Report` ho aggiunto degli `useState` che permettono di implementare la funzionalità di paginazione:

- `[maxRows, setMaxRows]`: controlla il numero massimo di righe impostato;

- `[currentPage, setCurrentPage]`: controlla la pagina in cui l'utente si trova al momento;

- `[totalRows, setTotalRows]`: controlla il numero totale di entry che sarebbero restituite dalla query senza paginazione.

Le prime due sono impostate dall'utente mentre per ottenere l'ultima è stato sufficiente aggiungere un campo `count` all'oggetto restituito dalla query precedentemente fatta.
Sempre a questa query ho aggiunto due nuovi query params, calcolati lato client:

- `skip`: quante entry "saltare" quando vado a richiedere le attività. Corrisponde a 0 se la pagina corrente è la prima, `currentPage - 1 * maxRows` altrimenti;

- `limit`: quante entry posso ottenere al massimo. Corrisponde a `maxRows`.

2.3.4 Sprint review

La sprint review si è conclusa in modo positivo, senza particolari correzioni necessarie. Anche i tempi sono stati rispettati.
La figura 2.5 mostra la tabella ottenuta e il relativo footer per navigarla e paginarla.

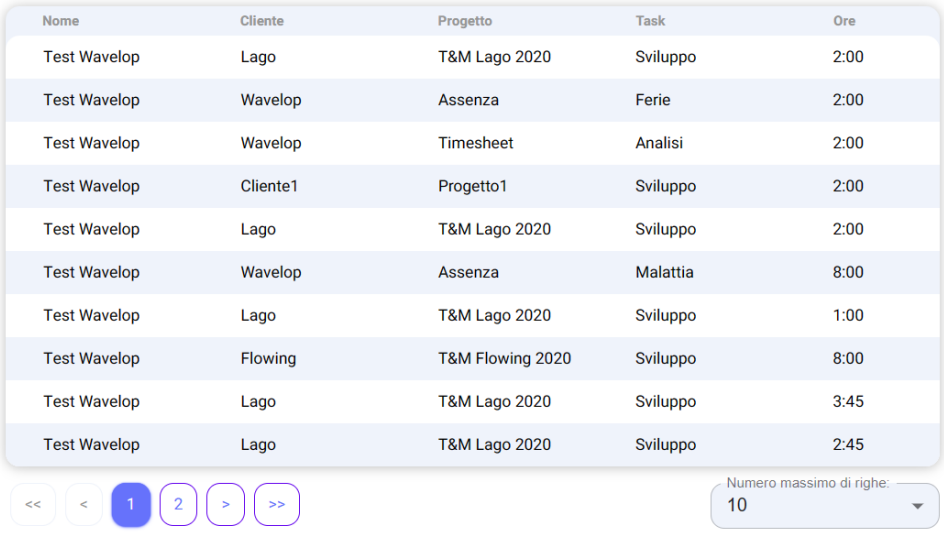

Figura 2.5: Tabella della sezione Reports

2.4 Sprint 4

2.4.1 User stories assegnate

Filtri utente, cliente, progetto e task (rispettivamente 5, 2, 2 e 2)

Come utente autenticato che si trova nella sezione "Reports", voglio poter scegliere uno o più utenti, clienti, progetti o task, anche contemporaneamente, e visualizzare le ore lavorative spese dal'utente o dedicate al cliente/progetto/task.

Tasks:

- Implementare il componente dei filtri;
  - implementare il componente principale;
  - implementare lo stile per i filtri;
- Implementare la logica di query nel backend per i filtri.

2.4.2 Filtri utente, cliente, progetto e task

A ciò che ho già fatto, è richiesto di aggiungere una toolbar, posizionata sopra la tabella, che permetta all'utente, tramite dei menù a tendina, di filtrare i dati che vengono mostrati, in particolare utenti, clienti, progetti e task.
Innanzitutto ho creato il component principale `ReportFilter` contenente la toolbar e i vari filtri.
Per creare elementi grafici come input, select, button ho utilizzato una librera grafica React già utilizzata nel resto della applicazione: MUI[1]. In questo modo, oltre a mantenere una consistenza di stile con il rest della app, è risultato più facile interagire

[1] *Sito ufficiale di MUI.* URL: https://mui.com/.

con questi elementi grafici (come tenere traccia di quali elementi sono selezionati nei select).
Successivamente, erano necessarie delle nuove richieste al backend per ottenere le liste di tutti gli utenti, clienti, progetti e task da mostrare nei rispettivi filtri. I vari endpoint erano già presenti, non ho quindi dovuto ancora toccare il backend per le richieste.
Avendo tutti i dati necessari, è stato sufficiente renderli disponibili al component principale `Reports` in modo da poterli usare nella richiesta per i dati già implementata.
In particolare ho aggiunto i seguenti queryparams:

- `users`: un array contenente gli utenti selezionati nei filtri;
- `clients`: un array contenente i clienti selezionati nei filtri;
- `projects`: un array contenente i progetti selezionati nei filtri;
- `tasks`: un array contenente i task selezionati nei filtri.

Ho modificato il backend di conseguenza, in modo che la query già implementata ottenesse solo le attività con, ad esempio, il campo `task` uguale a uno degli elementi presenti nell'array `tasks`. Con MongoDB questa è un'operazione piuttosto semplice in quanto basta utilizzare la keyword `$match` che permette appunto di ottenere documenti che rispettano una condizione, come mostrato nell'esempio di codice 2.2.

Esempio di codice 2.2: Esempio di filtro task lato backend con MongoDB

```
1  {
2    $match: tasks
3    ? { task: { $in: tasks } }
4    : { task: { $exists: true } },
5  },
```

In questo caso, se l'utente ha selezionato almeno un task, verrà eseguita la linea 3 del codice ottenendo solo i documenti con il campo `task` che compare in `tasks`, altrimenti verranno ottenuti tutti i documenti (linea 4).

2.4.3 Sprint review

La sprint review si è conclusa in modo positivo, senza particolari correzioni necessarie. Anche i tempi sono stati rispettati.
Nella figura 2.6 sono visibili i filtri realizzati, posizionati sopra la tabella.

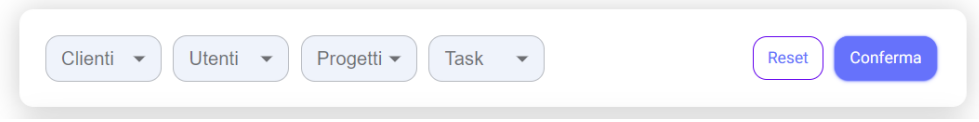

Figura 2.6: Filtri della tabella

2.5 Sprint 5

2.5.1 User stories assegnate

Filtro temporale (8)

Come utente autenticato che si trova nella sezione "Reports", voglio poter filtrare i dati visualizzati in modo che coprano solo un determinato lasso di tempo. In particolare voglio poter visualizzare i dati:

- del giorno corrente;
- della settimana corrente;
- del mese corrente;
- dell'anno corrente.

O, in alternativa, impostare un arco temporale personalizzato, con date d'inizio e fine a piacere.

Tasks:

- ricerca su librerie che offrano elementi datepicker consoni ai requisiti;
- implementare il componente del filtro temporale;
  - implementare il componente principale;
  - implementare lo stile per il filtro;
- aggiungere le opzioni per selezionare giorno/settimana/mese/anno corrente e arco temporale personalizzato;
- aggiornare la logica di query lato backend.

2.5.2 Filtro temporale

Con il team abbiamo concluso che la soluzione migliore per soddisfare questo requisito fosse l'aggiunta di un semplice bottone che, cliccato, espande un calendario dove l'utente può slezionare la data d'inizio e di fine dell'arco temporale per cui vuole filtrare i dati. Inoltre, a fianco al calendario, devono essere presenti delle opzioni per impostare rapidamente gli archi di tempo predefiniti.
Per semplificare le cose ho fatto una breve ricerca su delle librerie che offrissero dei datepicker adatti, finendo col scegliere `react-date-range`[2]. Questa scelta è dovuta principalmente che già di suo permette di soddisfare tutte le problematiche del requisito, eccetto quella di espandere il calendario da un bottone, che comunque è risolvibile in modo piuttosto semplice. È inoltre molto intuitiva e semplice da utilizzare ed è piacevole esteticamente.
Per quanto riguarda il lato backend non ho dovuto fare niente poichè avevo già integrato i parametri `from` e `to` nella query al database, sapendo che sarebbero serviti.

Infine, ho aggiunto un bottone per confermare i vari filtri per limitare il numero di richieste al backend. In questo modo verrà fatta una sola richiesta una volta che l'utente ha scelto tutti i filtri, invece che una dopo la selezione di ogni filtro.

2.5.3 Sprint review

La sprint review si è conclusa in modo positivo, senza particolari correzioni necessarie. Anche i tempi sono stati rispettati.

[2] *Repository GitHub di react-date-range.* URL: https://github.com/hypeserver/react-date-range.

2.6 Sprint 6

2.6.1 User stories assegnate

Raggruppamento dati (11)

Come utente autenticato che si trova nella sezione "Reports", voglio poter raggruppare i dati visualizzati per utente, cliente, progetto, task e data, con un massimo di 3 livelli di profondità.
Ad esempio, raggruppando per utente, la tabella mostrerà due colonne:

- `Utente`, con i nomi di ogni utente;
- `Ore`, con la somma delle ore delle attività di ogni utente.

Se poi voglio aggiungere un secondo livello di profondità al raggruppamento, ad esempio per cliente, la tabella mostrerà sempre la colonna `Utente`, ma questa volta, sotto a ogni nome, saranno presenti delle ulteriori righe con i nomi dei vari clienti per cui l'utente ha svolto attività, con relative somme di ore.

Tasks:

- implementare la funzionalità di raggruppamento;
- aggiungere la possibilità di raggruppamenti multipli;
- aggiungere una nuova sezione filtri dedicata ai raggruppamenti.

Esportazione report in formato CSV (3)

Come utente autenticato che si trova nella sezione "Reports", voglio poter esportare il report correntemente mostrato in un file CSV. Esso dovrà contenere tutti i dati mostrati nella tabella (anche eventuali dati presenti nelle pagine non correntemente mostrate).

Tasks:

- aggiungere opzione per eseportare la tabella in formato CSV;
- aggiungere logica per la generazione del file CSV lato backend.

2.6.2 Raggruppamento dati

La prima cosa che ho fatto è stata la parte di frontend, aggiungendo 3 menù a tendina, uno per ogni livello di raggruppamento, da cui scegliere il campo da raggruppare. Il secondo e terzo menù sono inizialmente nascosti e compaiono solo quando rispettivamente il primo e secondo raggruppamento sono stati scelti.
Lato frontend ho aggiornato la richiesta principale al backend aggiungendo 3 parametri: `groupOne`, `groupTwo` e `groupThree`. Essi sono stringhe che di default assumono il valore `none` ma, nel caso sia selezionato un raggruppamento da parte dell'utente, assumono il nome di quel raggruppamento, ad esempio `project`.
Ho inoltre creato un altro component per le righe "raggruppate", in modo da ottenere una struttura tale che, se l'utente ha scelto due raggruppamenti, vengano inizialmente

mostrate le righe del primo raggruppamento, potendo visualizzare quelle del secondo cliccando sulla rispettiva riga, come mostrato nella figura 2.7.

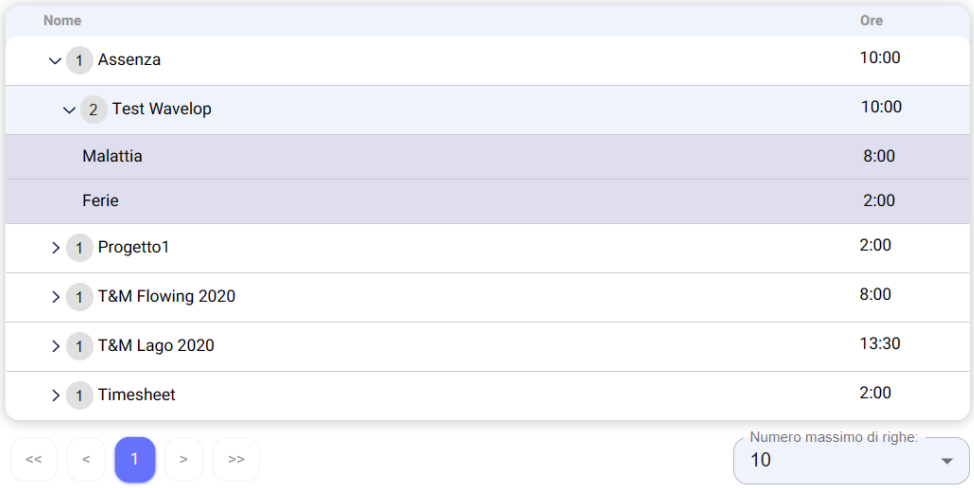

Figura 2.7: Tabella che mostra dati raggruppati

Lato backend ho semplicemente aggiornato la query già presente in modo che potesse raggruppare i dati ottenuti, se richiesto. In questo caso i dati vengono ritornati con la seguente struttura:

- `groupOne`
- `groupOneCount`
- `total`

Dove `groupOneCount` e `total` sono interi che rappresentano rispettivamente quanti elementi contiene il primo raggruppamento e la somma delle ore delle attività al suo interno. `groupOne` è invece un array contenenti oggetti con la seguente struttura:

- `name`: stringa contenente il nome della attività;
- `hours`: numero che rappresenta le ore dedicate all'attività;
- `groupTwo`: array contenente oggetti uguali a `groupOne`, eccetto relativi al secondo raggruppamento, più un campo `groupThree`.

2.6.3 Esportazione report in formato CSV

L'esportazione in formato CSV è risultata abbastanza rapida in quanto comportava solo ottenere i contenuti singoli dei campi richiesti dal frontend e porre tra di loro il carattere di separazione `","`.
L'unica cosa a cui prestare attenzione è la eventuale presenza di virgole o virgolette nei campi che possano interferire col processo di separare i campi. Per risolvere il problema basta circondare ogni campo tra virgolette.

Poi, tramite la funzione `join()`, ho unito tutti i campi di una riga ponendo fra loro il carattere `,` e separato le righe con il carattere di a capo `\n`.
Lato frontend ho poi aggiunto un bottone che permette di scaricare il file CSV così creato.

2.6.4 Sprint review

La sprint review si è conclusa in modo positivo, ma è emerso un piccolo bug nella selezione dei raggruppamenti: selezionando primo, secondo e terzo raggruppamento e poi togliendo il secondo o il primo, il terzo raggruppamento rimaneva selezionato. La sistemazione di questo bug è rimasta assegnata per lo sprint 7.

2.7 Sprint 7

2.7.1 User stories assegnate

Esportazione report in formato PDF (8)

Come utente autenticato che si trova nella sezione "Reports", voglio poter esportare il report correntemente mostrato in un file PDF. Esso dovrà contenere la tabella e il grafico mostrati nel report.

Tasks:

- aggiungere opzione per eseportare la tabella in formato PDF;
- aggiungere logica per la generazione del file PDF lato backend.

Inclusione/esclusione colonne (3)

Come utente autenticato che si trova nella sezione "Reports" e che sta esportando il proprio report, voglio poter selezionare quali colonne includere e quali no nel file PDF/CSV generato.

Tasks:

- Aggiungere un modale dopo avere selezionato il tipo di file da generare che permetta di scegliere quali colonne escludere;
- Aggiungere logica per riflettere le scelte fatte nel file generato.

2.7.2 Esportazione report in formato PDF

Per la creazione di file PDF mi è stato consigliato l'utilizzo di due librerie: puppeteer[3] e Handlebars.js[4].
La prima permette di utilizzare una versione *headless* (senza interfaccia utente) di un browser (in questo caso Chrome), la seconda, invece, mette a disposizione un linguaggio di templating HTML. L'idea è quindi di creare un template riutilizzabile per i report tramite Handlebars.js, per poi generarne un PDF tramite puppeteer, che

[3] *Pagina npm di puppeteer.* URL: https://www.npmjs.com/package/puppeteer.
[4] *Pagina npm di Handlebars.js.* URL: https://www.npmjs.com/package/handlebars.

verrà esportato.
Ho quindi creato il template tramite Handlebars.js che è risultato molto comodo e semplice grazie alle funzioni che la libreria mette a disposizione.

Esempio di codice 2.3: Esempio di utilizzo di Handlebars.js

```
1 {{#each people}}
2   <span>{{this.firstName}}</span>
3   {{#if this.lastName}}
4     <span>{{this.lastName}}</span>
5   {{/if}}
6   <br>
7 {{/each}}
```

L'esempio di codice 2.3 mostrato genererà, per ogni elemento presente in `people`, due tag `span` contenenti rispettivamente il valore della corrente iterazione di `people.firstName` e `people.lastName`, ma il secondo verrà generato solo se il campo `lastName` effettivamente esiste.
Una volta creato il template bisogna compilarlo tramite la funzione `compile()` di Handlebars.js per permettere a puppeteer di generare il relativo file PDF.

Esempio di codice 2.4: Esempio di utilizzo di puppeteer per la creazione di file PDF

```
1   const template = Handlebars.compile(htmlTemplate);
2   const html = template(data);
3   /*data contiene i dati che voglio mostrare nel report*/
4   const browser = await puppeteer.launch();
5   const page = await browser.newPage();
6
7   await page.setContent(html, { waitUntil: "domcontentloaded" });
8
9   const pdf = await page.pdf({
10     /*Opzioni di stile*/
11  });
12  await browser.close();
13  return pdf.toString("base64");
```

Nell'esempio di codice 2.4 si può vedere come il browser headless venga lanciato (riga 4), venga creata una nuova pagina (riga 5), ne venga impostato il contenuto con l'HTML generato (riga 7) e ne venga generato il PDF (riga 9).
Una volta generato il PDF, esso viene mandato al client per poi essere scaricato.

2.7.3 Inclusione/esclusione colonne

Per questa user story la gran parte del lavoro è stata fatta nel frontend: ho fatto in modo che, una volta selezionato che si vuole esportare il file come CSV/PDF si aprisse una modale in cui l'utente può selezionare quali colonne includere o escludere, tramite delle spunte, e poi ho aggiornato la chiamata al backend in modo che mandasse anche queste scelte.
Nel backend è stato sufficiente rendere condizionale la creazione di ogni colonna di dati.

2.7.4 Sprint review

La sprint review si è conclusa in modo positivo, con solo dei miglioramenti estetici da fare ai documenti PDF creati per il prossimo sprint. Anche la correzione fatta al bug sorto durante la sprint review precedente è stata soddisfacente.

2.8 Sprint 8

2.8.1 User stories assegnate

Visualizzazione grafico (8)

Come utente autenticato che si trova nella sezione "Reports", voglio poter vedere un istogramma che rappresenti graficamente i dati contenuti nella tabella.

Tasks:

- effettuare una ricerca su quale libreria Javascript possa essere più consona per creare il grafico richiesto;
- implementare il grafico.

2.8.2 Visualizzazione grafico

Per quanto riguarda la ricerca ho fatto un foglio Google simile a quello già fatto per i datepicker in cui ho raggruppato le librerie che mi sembravano migliori per svolgere l'incarico assegnato:

- Chart.js[5]
- D3.js[6]
- nivo[7]

Chart.js è, tra queste, la libreria più utilizzata e spicca per la sua semplicità di utilizzo e le molteplici tipologie di grafici preconfigurate. Anche il fatto che sia popolare è un punto a favore in quanto dispone di una community più ampia, che porta a trovare con più facilità soluzioni a eventuali problemi riscontrati.
Anche D3.js è una libreria piuttosto popolare e non è esclusivamente dedicata alla realizzazione di grafici ma, più in generale, alla manipolazione di elementi del DOM. È caratterizzata da avere un numero altissimo di funzionalità, che si traduce nella capacità di creare grafici complessi e per obiettivi specifici, al prezzo di avere una curva di apprendimento piuttosto ripida.
Nivo è basato su D3.js ma offre una buona riduzione di complessità rispetto a quest'ultimo, essendo pensato esclusivamente per la creazione di grafici.

La scelta finale è ricaduta su Chart.js in quanto il grafico da produrre era molto semplice e, essendo l'ultima settimana, non ci sarebbe stato tempo per familiarizzare con uno strumento complesso come D3.js.

[5] *Sito ufficiale di Chart.js.* URL: https://www.chartjs.org/.
[6] *Sito ufficiale di D3.js.* URL: https://d3js.org/.
[7] *Sito ufficiale di nivo.* URL: https://nivo.rocks/.

L'implementazione del grafico si è rivelata infatti piuttosto semplice, dovendo solo dichiarare delle variabili per i dati da mostrare e per delle opzioni aggiuntive del grafico:

Esempio di codice 2.5: Esempio di utilizzo di Chart.js

```
1   /*Opzioni aggiuntive del grafico*/
2   const options = {
3     responsive: true,
4     scales: {
5       y: {
6         min: 0,
7       }
8     }
9   };
10
11  /*Dichiarazione dati e relative labels*/
12  const data = {
13    labels: getLabels(),
14    datasets: [
15      {
16        data: getData(),
17      }
18    ]
19  };
20
21  return <Bar options={options} data={data} />;
```

Anche la struttura di come venivano restituiti i dati non è risultata problematica nè per i visualizzare i dati raggruppati nè per quelli non raggruppati.

2.8.3 Sprint review

Essendo la sprint review finale ho dimostrato brevemente le funzionalità sviluppate fino ad ora, ottenendo, anche per quanto riguarda il miglioramento estetico al file PDF assegnatomi lo scorso sprint, un feedback positivo.
Nella figura 2.8 viene mostrato il grafico creato durante lo sprint.

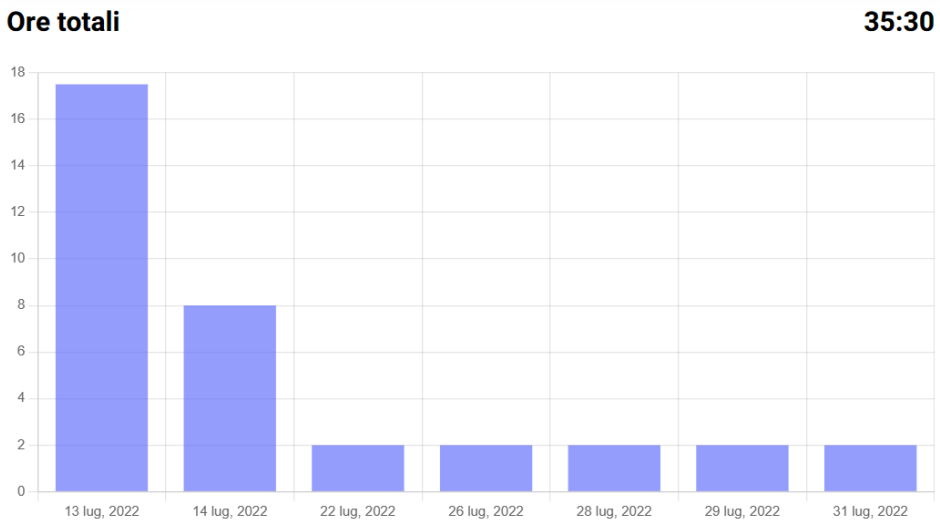

Figura 2.8: Grafico della sezione Reports

Capitolo 3

Conclusioni

3.1 Il risultato finale

Al termine di queste otto settimane ho portato a termine una sezione di una web app capabile di mostare all'utente dati relativi alle attività aziendali in formato sia tabellare che grafico e di esportarli.
Questi dati sono inoltre organizzabili in molti modi tramite le opzioni di filtro e di raggruppamento.

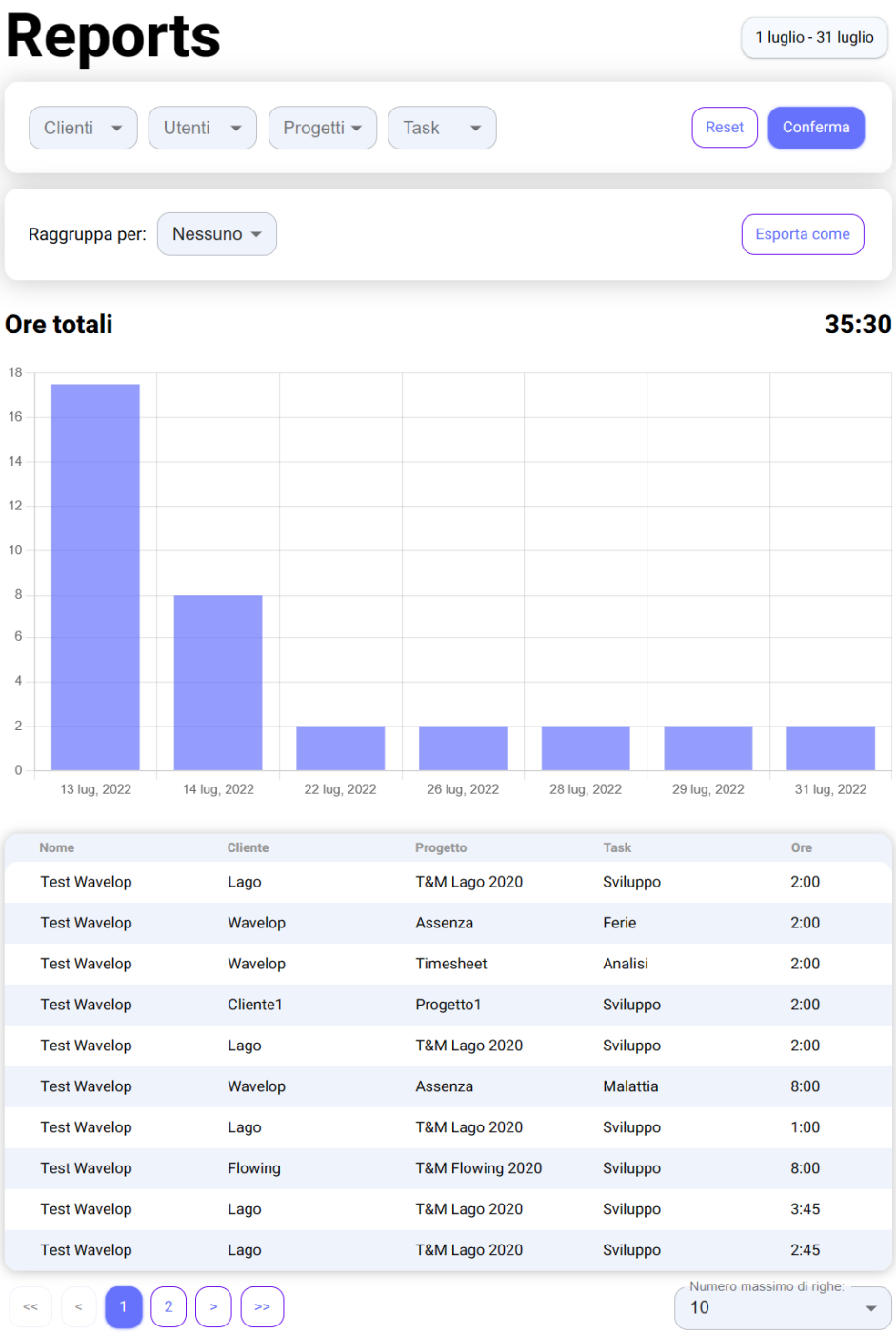

Figura 3.1: L'intera sezione Reports

3.2 Soddisfazione dei requisiti

La tabella seguente riporta lo stato di completamento dei requisiti elencati in §1.4.

Requisito	Importanza	Stato
Apprendimento delle tecnologie di sviluppo come React e Node.js e per il versionamento del progetto come git	Obbligatorio	Rispettato
Gestione filtri avanzati di ricerca delle attività	Obbligatorio	Rispettato
Visualizzazione a tabella delle attività filtrate	Obbligatorio	Rispettato
Generazione file CSV delle attività filtrate	Obbligatorio	Rispettato
Generazione file PDF delle attività filtrate	Desiderabile	Rispettato
Visualizzazione attività tramite grafico	Desiderabile	Rispettato
Salvataggio preset filtri per ricerche future	Desiderabile	Non Rispettato
Visualizzazione widget laterale con statistiche relative all'utente	Facoltativo	Non Rispettato
Gestione responsive della piattaforma	Facoltativo	Parzialmente rispettato

Tabella 3.1: Tabella dello stato di completamento dei requisiti

Seppure tutti i requisiti obbligatori sono stati tutti rispettati, ciò non è vero per alcuni desiderabili e facoltativi. Tale mancanza è principalmente dovuta al fatto che, durante lo stage, sono sorti altri requisiti che hanno avuto la precedenza su quelli non fatti (in particolare l'inclusione e l'esclusione delle colonne e il raggruppamento dati) e alla fine sono rimasti incompiuti. Ho inoltre segnato la gestione responsive della piattaforma come parzialmente rispettato poichè, andando avanti col progetto, ho gestito il responsive dei componenti che ho implementato, ma non c'è stato tempo di fare lo stesso per quelli rimanenti.

3.3 Valutazioni personali

Questa esperienza di stage è stata un'ottima opportunità per mettere in pratica ciò che ho appreso durante il mio percorso di laurea. In particolare da questa esperienza mi sono rimasti più di tutti questi insegnamenti:

- il lavorare in un ambiente aziendale, seguendo e rispettando dinamiche e metodologie ben definite e come influiscano positivamente sul lavoro;
- ho consolidato e arricchito le mie conoscenze sullo sviluppo di applicazioni web;
- ho imparato a utilizzare tecnologie che non avevo mai toccato prima d'ora e pratiche che avevo poco approfondito;
- ho imparato l'importanza della comunicazione e del confronto con i colleghi per collaborare alla risoluzione di un problema o al raggiungimento di un obiettivo, ma, allo stesso tempo, l'importanza di consultare la documentazione ufficiale per superare un ostacolo all'apparenza invalicabile.

Infine, voglio anche dire che questa è stata un'esperienza non solo utile ma anche piacevole, e che sono contento di finire il mio percorso universitario con questa opportunità.

Bibliografia

Siti web consultati

Sito ufficiale di React. URL: https://reactjs.org/ (cit. a p. 3).

Sito ufficiale di Node.Js. URL: https://nodejs.org/en/ (cit. a p. 4).

Sito ufficiale di MongoDB. URL: https://www.mongodb.com/ (cit. a p. 4).

Sito ufficiale di MUI. URL: https://mui.com/ (cit. a p. 12).

Repository GitHub di react-date-range. URL: https://github.com/hypeserver/react-date-range (cit. a p. 14).

Pagina npm di puppeteer. URL: https://www.npmjs.com/package/puppeteer (cit. a p. 17).

Pagina npm di Handlebars.js. URL: https://www.npmjs.com/package/handlebars (cit. a p. 17).

Sito ufficiale di Chart.js. URL: https://www.chartjs.org/ (cit. a p. 19).

Sito ufficiale di D3.js. URL: https://d3js.org/ (cit. a p. 19).

Sito ufficiale di nivo. URL: https://nivo.rocks/ (cit. a p. 19).

Manifesto Agile. URL: http://agilemanifesto.org/iso/it/.

Guida Scrum di Atlassian. URL: https://www.atlassian.com/agile/scrum.

Stackoverflow. URL: https://stackoverflow.com/.