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

Capitolo 1

L'azienda

1.1 Descrizione generale

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, conta circa 10 dipendenti e offre un ambiente lavorativo giovane e dinamico. Favorisce inoltre lo smartworking e la flessibilità degli orari.
Dal 2020 propone progetti per studenti universitari interessati ad 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.

Capitolo 2

Descrizione dello stage

2.1 Il progetto

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.

2.2 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 nececssari 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.

2.3 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 cultira 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;
- progetazione 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.

2.4 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.

2.4.1 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 i componenti che formano la pagina.

Esempio di utlizzo di codice JSX

```
const name = 'Riccardo Pavan';

const ShowName = () => {
  return (
    <div>Mi chiamo {name}</div>
  );
};
```

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.

2.4.2 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*.

2.4.3 MongoDB

MongoDB[3] è un *database manaegement 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.

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

Capitolo 3

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 porgrammazione 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 docmento 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.

Capitolo 4

Sprint 2

Durante il secondo sprint ho redatto la versione conclusiva del documento relativo alle user stories, approvato poi dal mio relatore esterno.
Ogni user story del documento presenta i seguenti elementi:

- nome della useer 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 utli per allocare meglio le varie user stories all'interno degli sprint;
- 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 speccifico e tecnico cosa fare per concludere la user story;

All'inizio di ogni capitolo relativo a uno sprint indicherò le user stories su cui andrò a lavorare.

Ho inoltre creato dei diagrammi UML dei casi d'uso. (Li metto qua?)

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 mostate 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.

Capitolo 5

Sprint 3

5.1 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, elencate di seguito con tanto di descrizione di task:

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 rigurda 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.

5.2 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 che segue:

Esempio di fetch dati da backend e memorizzazione in React state

```
const [data, setData] = useState([]);

const fetchData = useCallback(async () => {
  const resultData = await
  /* Logica per la richiesta dei dati */
  if (resultData) {
    setData(resultData);
  }
})

useEffect(() => {
  fetchData();
}, [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 ad ogni singolo re-rendering della pagina.
Per quanto riguarda la logica per la richiesta dei dati ho aggiunto un endpoint `/report` che risponde ad 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. TODO: [AGGIUNGERE ENDPOINT, CONTROLLER, SERVICE, REPOSITORY]

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 un riga della stessa. A seconda di quanti elementi saranno ottenuti dal backend, `ReportTable` chiamerà `ReportRow` un numero adeguato di volte, creando una tabella.

5.3 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 mostate. 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]`: TODO: [TERMINE MIGLIORE DI CONTROLLA] 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`.

TODO: [IMMAGINE FOOTER TABELLA]

5.4 Sprint review

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

Capitolo 6

Sprint 4

Descrizione del quarto sprint.

Capitolo 7

Sprint 5

Descrizione del quinto sprint.

Capitolo 8

Sprint 6

Descrizione del sesto sprint.

Capitolo 9

Sprint 7

Descrizione del settimo sprint.

Capitolo 10

Sprint 8

Descrizione dell'ottavo sprint.