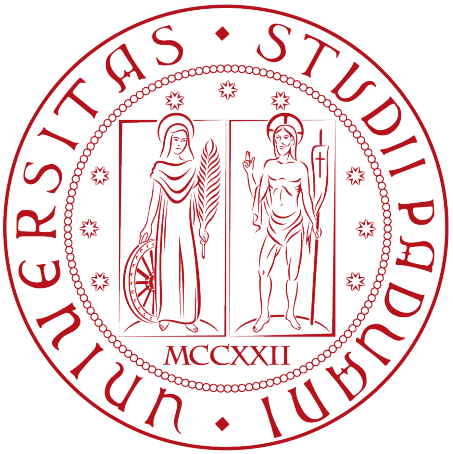
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Una applicazione web per la recensione di prodotti

Tesi di laurea

Relatore

Prof.Tullio Vardanega

Laureando

Matteo Tossuto

Anno Accademico 2021-2022

Matteo Tossuto: *Una applicazione web per la recensione di prodotti,* Tesi di laurea, © Dicembre 2022.

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di *stage*, della durata di circa trecento ore, dal laureando Matteo Tossuto presso l'azienda Synthema Artificial Intelligence - S.AI S.r.l. Il progetto di *stage* consisteva nella realizzazione di una Progressive Web App (PWA), in React. Tale applicazione doveva garantire mediante la scansione di un codice a barre, presente sulla confezione di un prodotto, o attraverso la sua digitazione, all'utente della piattaforma di poter raggiungere una pagina contenente informazioni aggiuntive sui prodotti. L'obiettivo del progetto era quello di sperimentare con varie tecnologie, individuandone i pregi e difetti e decidere quali utilizzare e quali eliminare. Tutta l'attività, le problematiche sorte e quanto prodotto nello *stage* sarà esposto in modo dettagliato nei capitoli che compongono il presente documento.

Ringraziamenti

Vorrei ringraziare di cuore la mia meravigliosa ragazza, che mi ha sempre supportato e sopportato lungo tutto il mio percorso di studi, grazie alla quale ora sono qua, che mi è stata accanto nelle brutte giornate e alla quale devo molto.

Desidero ringraziare la mia famiglia per avermi supportato e sopportato lungo il mio percorso di studi, per gli spazi e il tempo che mi hanno lasciato per dedicarmi allo studio, al lavoro e alla tesi.

Desidero ringraziare i miei nonni che mi hanno insegnato molte cose, mi hanno sempre aiutato e a cui devo molto.

Vorrei ringraziare tutti i miei amici poiché è anche grazie a loro e a ciò che abbiamo passato assieme che ora sono chi sono.

Desidero ringraziare ed esprimere la mia gratitudine al Prof. Tullio Vardanega, relatore della mia tesi, per avermi seguito ed aiutato durante lo stage, per l'aiuto che mi ha dato durante la stesura della tesi e la pazienza che ha avuto con me durante il nostro percorso assieme.

Infine, vorrei ringraziare Giulio Paci, tutor aziendale durante il mio stage, e i miei colleghi per essere sempre stati disponibili e per avermi insegnato molto.

Padova, Dicembre 2022 Matteo Tossuto

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

L'azienda

1.1 Presentazione

Synthema Artificial Intelligence (S.AI) è una *start-up* innovativa che si occupa di ricerca, progettazione, sviluppo, commercializzazione e manutenzione di prodotti e servizi innovativi ad alto valore tecnologico, basati sull'*Internet of Things*, la *blockchain* e su tecniche di Intelligenza Artificiale (in particolare reti neurali profonde), per l'analisi integrata e la comprensione di dati multimodali da fonti eterogenee (linguaggio naturale sia scritto che parlato, audio, immagini, video, dati generati da sensori) e per la gestione dei *workflow*, sia nel settore pubblico che privato. Inoltre l'azienda progetta e realizza applicazioni online, accessibili via *Web*, che consentono di semplificare e automatizzare la gestione di procedure amministrative di varia natura, interagendo tramite *web* anche con eventuali utenti esterni. Le soluzioni offerte sono rivolte ad aziende ed enti sia pubblici che privati e sono sempre personalizzate in base alle esigenze. Per garantire tale personalizzazione, l'azienda supporta i clienti nel tempo, offrendo la possibilità di adeguare le applicazioni a seguito di evoluzioni di procedure, modalità operative, esigenze o modifiche normative. L'azienda è specializzata nella gestione dei concorsi per l'accesso ai Corsi di Laurea a numero chiuso. Il sistema consente di gestire facilmente tutte le varie fasi del processo: dall'iscrizione dei candidati all'acquisizione e alla correzione ottica degli elaborati con *test* a risposte multiple e la generazione automatica di verbali e graduatorie. S.AI si occupa anche di ricerca e sviluppo ed è attualmente impegnata nella preparazione di proposte di progetto nelle aree dell'*emotion recognition* e *cognitive computing* e il suo *team* di ricerca ha partecipato come leader in diversi progetti europei.

1.2 Metodologia di sviluppo

Per sviluppare *software*, Synthema Artificial Intelligence usa il *framework Scrum*. Si tratta di una metodologia incrementale, la quale suddivide l'intero progetto in blocchi rapidi di lavoro (*Sprint*) alla fine di ciascuno dei quali viene creato un incremento del *software*. Ogni *Sprint* è preceduto da una riunione di pianificazione in cui vengono identificati gli obiettivi (*Sprint Backlog*) e vengono stimati i tempi. Gli *Sprint* hanno una durata fissa da una a due settimane. Al termine di ogni *Sprint*, il *team* di sviluppo consegna una versione potenzialmente completa e funzionale del prodotto, che rispetta gli avanzamenti decisi nella riunione di pianificazione.

I *task* che non vengono completati al termine dello *Sprint*, vengono spostati nel *Product Backlog* per essere completati negli *Sprint* successivi. Questa metodologia consente di monitorare e adattare più facilmente la direzione del progetto e per venire in contro alle richieste dell'azienda cliente.

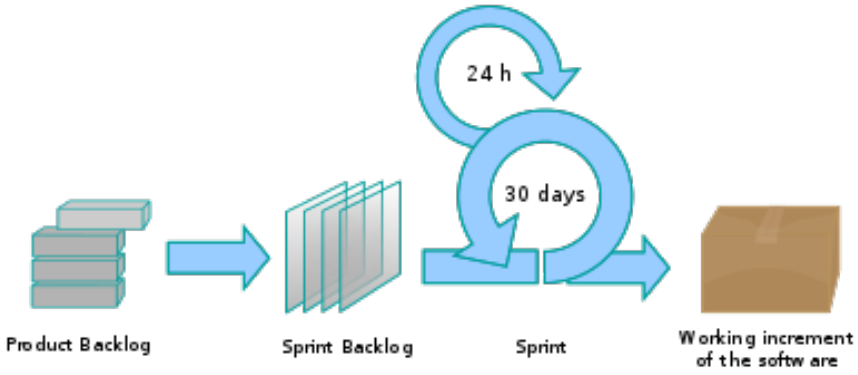


Figura 1.1: Metodologia *Scrum* (https://wikipedia.org)

L'azienda utilizza *Jira* per creare e tracciare i vari *Sprint* e i *task* da svolgere o svolti; ossia su *Jira* sono presenti il *Product Backlog* e il *Sprint Backlog*; ovviamente *Sprint* e *task* sono suddivisi in base all'ambito di sviluppo e al progetto a cui sono correlati. Attraverso *Jitsi Meet* vengono tenute le riunioni di pianificazione e le riunioni di fine *Sprint*. La piattaforma viene anche usata dall'azienda per discutere con i clienti.
Slack viene utilizzato per una comunicazione veloce tra i membri del *team*, per discutere dei *ticket* o di eventuali problemi. Inoltre tramite un sistema automatico di notifiche collegato a *Jira*, l'intero *team* viene informato sul procedimento del lavoro e sullo stato di ogni *ticket*, in modo da avere una visione più ampia del progetto e poter intervenire in caso di necessità o di richiesta specifica. Alla fine di ogni *Sprint* una versione funzionale è reperibile da un *repository* aziendale su *GitLab* che viene utilizzato come sistema di versionamento.
Eventuali richieste o aggiunte sono prima discusse tra il responsabile del progetto e i clienti tramite *Jitsi Meet*. Una volta che le richieste sono state concordate, gli adattamenti necessari saranno aggiunte allo *Sprint* successivo se uno è già in corso.

Per produrre *software* l'azienda usa diversi strumenti di sviluppo e comunicazione in comune tra i diversi ambiti di interesse e lavoro trattati. Mentre usa differenti tecnologie in base agli ambiti di sviluppo. Nella prossima sezione vedremo le tecnologie utilizzate per produrre *Web Application* e gli strumenti in comune usati dall'azienda.

1.3 Tecnologie utilizzate

Le tecnologie seguenti vengono usate dall'azienda per produrre *Web Application*, come nel caso del mio progetto:

* **React**: Utilizzato solo recentemente dall'azienda, sembra molto promettente e l'azienda sembra già sicura di questa tecnologia;

* **Node.js**: Anche questo utilizzato solo recentemente dall'azienda insieme a *React* e il suo utilizzo o meno dipende dall'uso di *React*;
* **Django**: Utilizzato da sempre dall'azienda per creare e mantenere *API*;
* **Docker**: Utilizzato dall'azienda in tutti gli ambiti di sviluppo per poter creare un ambiente di sviluppo indipendente.

In questi progetti *Docker* viene utilizzato per creare un ambiente di sviluppo indipendente in cui viene utilizzato *Django* per progettare le *API* e *React* e *Node.js* per la codifica e implementazione delle soluzioni progettate.
Nel caso del mio progetto invece era stata adottata una soluzione più semplice e veloce: *Docker* veniva utilizzato per ricreare una copia del *server* da cui andavo a prendere i dati dei prodotti o a caricarli. Invece il *frontend* e il *backend* li facevo girare tramite *localhost* e tutte le operazioni riguardanti le recensioni, l'autenticazione e registrazione avveniva puramente tra loro due.

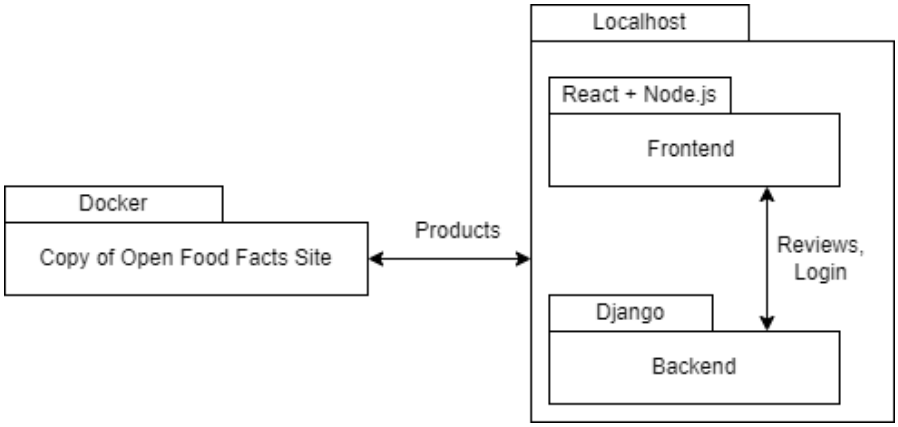


Figura 1.2: Architettura del progetto

1.3.1 React

React è una libreria *JavaScript* per lo sviluppo di interfacce utente, *open source*, creato e mantenuto principalmente da *Meta* (ex *Facebook*). Consente l'utilizzo dei linguaggi "classici" per la creazione di una *Web App* (*JavaScript*, *HTML*, *PHP*, ecc...) ma anche una nuova sintassi: *JSX* che è un estensione della sintassi *JavaScript* che produce "elementi *React*" che verranno poi renderizzati nel *DOM*. Da notare che *JSX* è in pratica zucchero sintattico e può essere sostituito da *JavaScript* andando però così a perdere diversi vantaggi dell'utilizzare *React*. *React* permette lo sviluppo di una Single Page Application (SPA) che funziona in tutti i *browser* moderni, inclusi quelli *mobile*, si occupa del *rendering* dei dati sul *DOM* ed è in grado di interfacciarsi con altre librerie e/o *framework*.

Tra i vari vantaggi che offre *React* ci sono:

* Possibilità di creare componenti grandi o piccoli da riutilizzare per evitare duplicazione di codice;

* Possibilità di usare assieme *JavaScript* e codice *HTML* ossia il linguaggio *JSX*;
* Vasta e attiva *community*;
* Permette l'utilizzo di un estensione *web* per facilitare lo sviluppo e il *test* tramite *DevTools* di *Google Chrome*;
* Grande disponibilità di moduli che offrono diversi servizi;
* Diverse librerie supportate per facilitare lo sviluppo sotto diversi aspetti.

```
import Spinner from "./Spinner";

function ProductList() {

    const [searchParams] = useSearchParams();
    const searchName = searchParams.get('search_name');
    const navigate = useNavigate();

    const handleReturn = () => {
        navigate("/");
    }

    const { count, productList } = useGetProductListBySearchNameQuery(searchName, {
        selectFromResult: ({ currentData }) => ({
            count: currentData?.count,
            productList: currentData?.products,
        }),
    });

    if (!productList) return <Spinner />

    return (
        <>
            <MainNavBar search />
            <Container component='main' maxWidth='100%' sx={{ pt: 6 }} align='-webkit-center'>
                <Typography variant="h3">
                    Product List: {count} products found
                </Typography>
                <Box>
                    {count === 0 &&
                        <Button variant="contained" onClick={handleReturn} sx={{ mt: 3 }}>
                            Cercare un altro prodotto
                        </Button>
                    }
```

Figura 1.3: Esempio di un componente *React*

1.3.2 Node.js

Node.js è un sistema a *runtime, open source,* multipiattaforma orientato agli eventi per l'esecuzione di codice *JavaScript*. Usato insieme a *React* permette tramite *NPM*, un *package manager*, l'installazione di *packages* o moduli. Tali moduli sono librerie *JavaScript* che introducono o facilitano funzionalità utilizzabili dalla *Web App*. *Node.js* è progettato per creare applicazioni di rete scalabili: permette di gestire molte connessioni contemporaneamente; ad ogni connessioni viene chiamata la *callback* ma se non c'è nulla da fare *Node.js* rimarrà inattivo. Inoltre quasi nessuna funzione in *Node.js* esegue direttamente *I/O*, quindi il processo non si blocca mai tranne quando l'*I/O* viene eseguito utilizzando i metodi sincroni della libreria *standard* di *Node.js*.

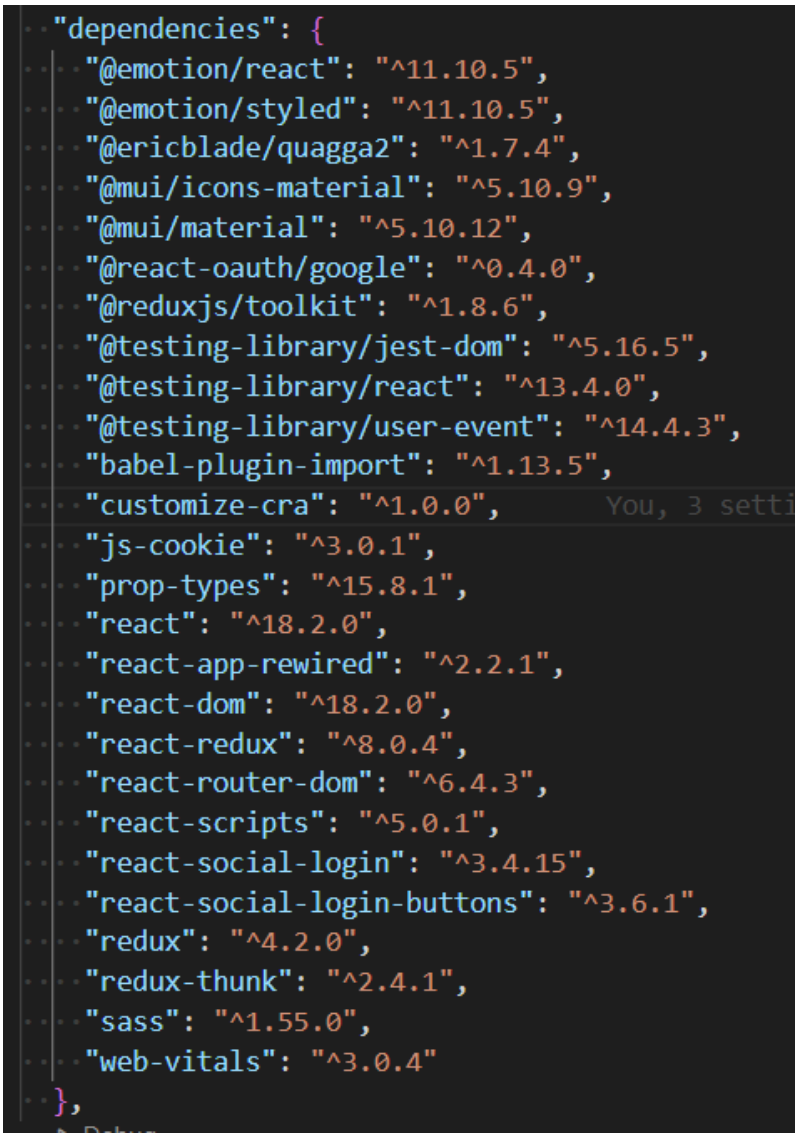


Figura 1.4: Moduli installati tramite *Node.js*

1.3.3 Django

Django è un *web framework* scritto in *Python*, per lo sviluppo *back-end* di *Web App*. Si occupa di gran parte della seccature per lo sviluppo *web*, permettendo allo sviluppatore di occuparsi dello sviluppo senza dover reinventare la ruota. *Django* è *open source*, è presente una vasta e attiva *community* e fornisce una completa e buona documentazione. Utilizzare *Django* permette di avere *Web App*:

* Complete;
* Versatili;
* Sicure;
* Scalabili;
* Mantenibili;
* Portabili.

Django permette anche le creazione e gestione di *API*, *endpoint* e *query*.

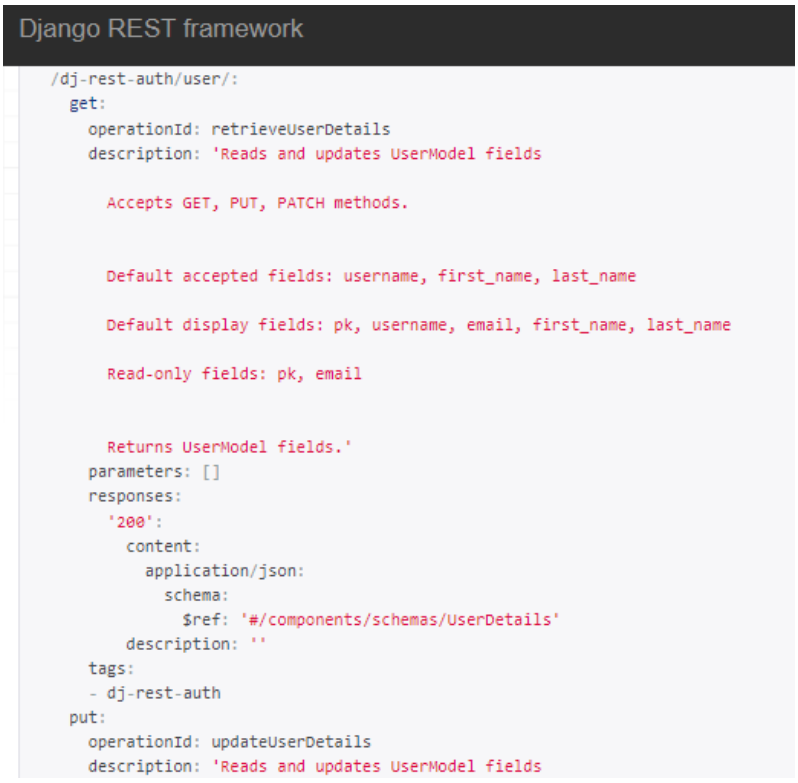


Figura 1.5: Esempio di *API* creata tramite *Django*

1.3.4 Docker

Docker permette la creazione e utilizzo di *Container*, ossia pacchetti di *software* leggeri, autonomi ed eseguibili che contengono tutto il necessario per eseguire un'applicazione: codice, eventuali strumenti, librerie, configurazioni e impostazioni. Caratteristica principale è che i *Container* isolano il *software* dall'ambiente esterno, assicurando che funzioni in modo uniforme nonostante eventuali differenze; come differenze possibili tra ambienti di sviluppo e di produzione. Usato molto spesso dall'azienda poiché permette di configurare e utilizzare diversi servizi insieme. Nel caso di questo progetto, ho utilizzato *Docker* autonomamente, per simulare il sito da cui l'applicativo prende i dati sui prodotti o dove carica o modifica eventuali prodotti. Tra i vari vantaggi di *Docker* c'è la possibilità di creare dei *volumi*, ossia *directory* locali nel computer, in cui viene salvato lo stato dei vari servizi, così si possono "spegnere" e "accendere" i *Container* senza perdita di dati.

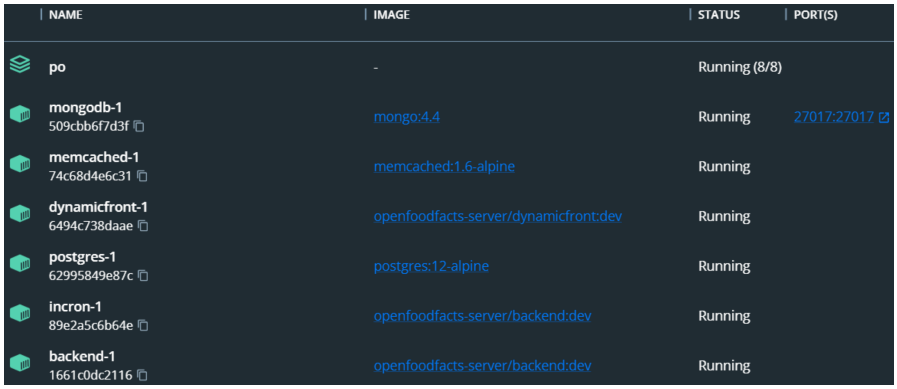


Figura 1.6: Esempio di *Container* utilizzati durante il progetto

1.4 Strumenti di sviluppo

Questi strumenti vengono usati dall'azienda per sviluppare le varie applicazioni e coordinare il *team* e il lavoro.
Vengono usati in tutti gli ambiti di sviluppo in cui opera l'azienda e, tolto *Visual Studio Code* che viene utilizzato solo durante la fase di progettazione e maggiormente durante la fase di programmazione, gli altri due strumenti, *GitLab* e *Jira*, sono presenti durante tutto il flusso di lavoro e ciclo di vita del *software*.

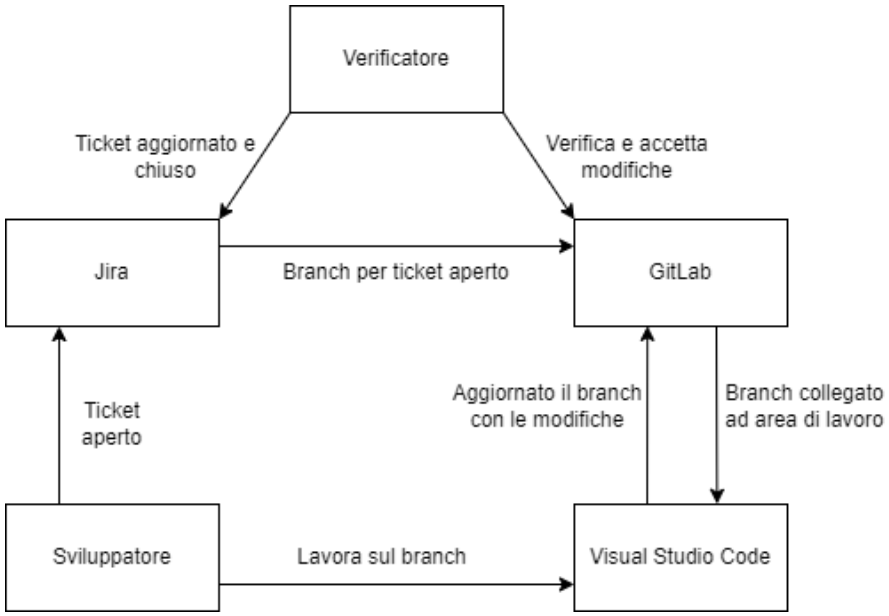


Figura 1.7: *Workflow* del lavoro tramite strumenti di sviluppo

1.4.1 Visual Studio Code

Visual Studio Code è un editor di codice sorgente autonomo sviluppato da *Microsoft*. Di base supporta *JavaScript* e alcuni linguaggi per lo sviluppo di *Web App*, ma grazie al ricco *marketplace* è possibile scaricare diverse e numerose estensioni per poter utilizzare *Visual Studio Code* con ulteriori linguaggi.
Le funzionalità che *Visual Studio Code* offre sono:

* *IntelliSense*, una tecnologia che velocizza la scrittura di codice offrendo suggerimenti in caso di dubbi;
* Un *debbuger* avanzato, che consente di sospendere l'esecuzione del codice per esaminare un *bug*;
* Una sezione dedicata per eseguire i *test* ed esaminare gli eventuali errori sistemando il codice velocemente;
* Un'integrazione nativa per i programmi di controllo versione come *Git*;
* Il supporto per la collaborazione, fra colleghi, rapida tramite *Live Share*;
* *Editor* altamente estendibile e personalizzabile grazie al ricco *marketplace*;
* Possibilità di usare direttamente *Azure* (*software* sempre di *Microsoft*).

1.4.2 GitLab

GitLab è una piattaforma *web open source*, pensato per l'utilizzo della pratica *DevOps*, che permette la gestione di *repository Git*, gestione di *ticket* ed è facilmente integrabile con diversi servizi di *CI* e *CD* o *software* come *Docker*. *GitLab* mette a disposizione diverse funzionalità a seconda del tipo di sottoscrizione e del prezzo pagato. È comunque

possibile utilizzarlo gratuitamente. Durante lo *stage*, per utilizzare al meglio il servizio offerto da *GitLab*, ho deciso di utilizzare *Git* da *console* di commando direttamente da *Visual Studio Code*. Le caratteristiche principali nell'utilizzo di *GitLab* e di *Git* sono:

* Supporto allo sviluppo non lineare attraverso l'utilizzo di *branch* e *merge*;
* Sviluppo distribuito. Ogni sviluppatore ha una copia locale dell'intera cronologia di sviluppo;
* Rapidità nell'esecuzione, essendo scritto in linguaggio *C*.

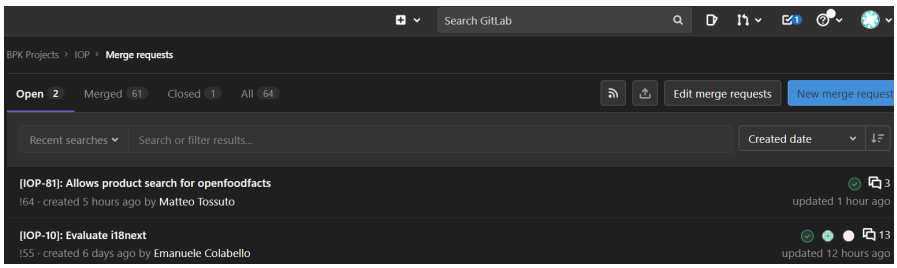


Figura 1.8: Interfaccia di *GitLab* per la gestione delle richieste di *merge*

1.4.3 Jira

Jira è un *software* proprietario sviluppato da *Atlassian* e gratuito per *team* fino a 10 membri; ideato per gestire e tracciare *Issue*, *bug* o *task* tramite *ticket* e pensato per la gestione di progetti sviluppati con metodi agili. Integrabile con diversi servizi, tra cui lo strumento di comunicazione *Slack*. Punto di forza di *Jira* è l'estrema personalizzazione dei *ticket* ma anche di modelli come *Scrum* o *Kanban*.

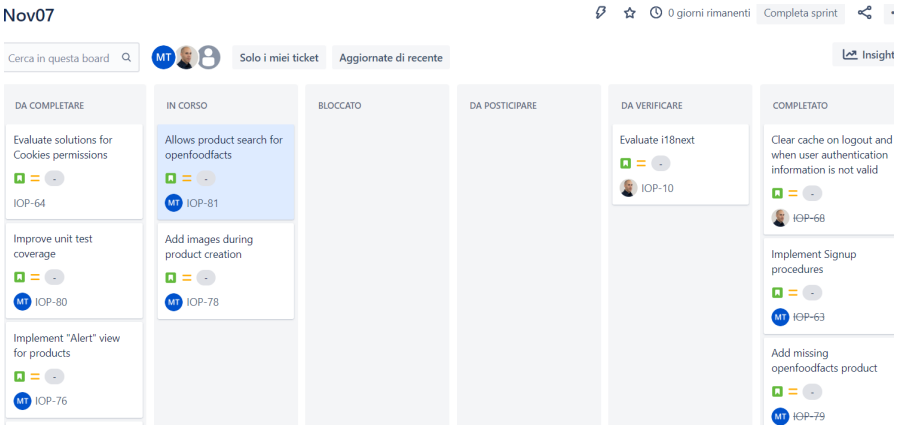


Figura 1.9: Esempio di *workflow* dei *ticket* su *Jira*

1.5 Strumenti di comunicazione

Questi strumenti vengono usati dall'azienda per la comunicazione, *meeting* e l'invio di notifiche relative al lavoro.
Vengono usati in tutti gli ambiti di sviluppo in cui opera l'azienda e sono presenti durante tutto il flusso di lavoro e ciclo di vita del *software*.

1.5.1 Slack

Slack è un *software* di messaggistica ideato principalmente per ambiente aziendali e offre *feature* di tipo *IRC*:

* Canali: ossia *chat room* permanenti suddivisi in base alle necessità;
* Messaggi diretti: per comunicazione 1 a 1.

Oltre a ciò permette:

* Incontri audio e video tra singoli individui o gruppi;
* Alta integrabilità con diversi *software* o servizi come *Jira* o *Google Docs* (ad esempio ricevere le notifiche da *Jira*);
* Alta personalizzazione per gestione di notifiche o stato dell'utente;
* Permette la connessione di aree di lavoro di aziende diverse per una comunicazione più facile e veloce;
* Permette l'automatizzazione di diverse azioni.

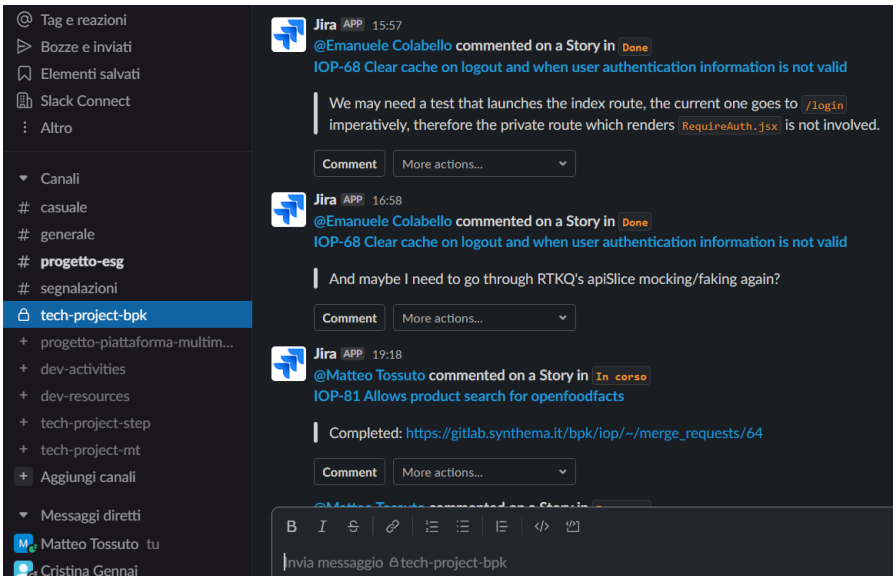


Figura 1.10: Interfaccia di *Slack* ed esempio notifiche da *Jira* su apposito canale

1.5.2 Jitsi Meet

Jitsi Meet è un sistema gratuito e *open-source* per effettuare videochiamate. È possibile utilizzarlo senza doversi registrare o installare alcun *software* sul proprio computer. È utilizzabile dal *browser*, dandogli i permessi necessari per gestire il microfono e la *webcam* del computer. È pienamente compatibile con tutti i principali sistemi operativi desktop: *Windows*, *macOS* e *Linux*. Inoltre, sono presenti anche versioni per piattaforme mobili, come *iOS* e *Android*, e anche queste versioni sono totalmente gratuite. Permette agli utenti di usare una *chat* di testo o di condividere simultaneamente più schermi. Inoltre permette l'integrazione con alcuni servizi: *Google*, *Microsoft* e *Slack*.

1.6 Propensione all'innovazione

L'azienda è perennemente alla ricerca di innovazione per i propri prodotti e la propria offerta tecnologica. Da un punto di vista delle tecnologie utilizzate stavano rinnovando un po' tutto, ma allo tempo sono costretti a mantenere vari *software* che utilizzano tecnologie ormai obsolete a causa di problemi di dipendenze o richieste del cliente. Da un punto di vista organizzativo, essendo ancora una *Start-Up*, cercano di mettere in piedi delle buone pratiche di gestione delle attività. In generale, l'azienda punta ad innovarsi tecnologicamente seguendo le novità di mercato. Prima però, esegue analisi di costo e creazione di prototipi e, se vanno a buon fine, vengono incorporate nella *suite* aziendale. Essendo un azienda relativamente giovane è molto probabile che le strategie di innovazione andranno a cambiare con il tempo, con un eventuale cambio di clientela o causa di un cambio di obiettivi.

Capitolo 2

Lo *stage*

2.1 Visione dell'azienda degli *stage*

S.AI vede gli *stage* come un opportunità per farsi conoscere dai giovani laureandi e allo stesso tempo cercare potenziali futuri sviluppatori da assumere ed integrare nei propri *team*. Dato che l'azienda ha un accordo di *stage* sia con l'Università di Padova, sia con l'Università di Pisa, (luogo di sede principale dell'azienda) ha una maggiore possibilità di farsi conoscere dai laureandi e assumere nuovi sviluppatori. Inoltre, permettendo di lavorare da remoto permettono agli studenti di godere di una maggiore flessibilità per quanto riguarda spostamenti, gestione del tempo o di altri impegni universitari, dandoli così un motivo in più per accettare lo *stage* proposto dall'azienda.
Gli ambiti in cui offrono di fare *stage*, sono lo sviluppo di *Web Application* o progetti correlati all'intelligenza artificiale; ovviamente la disponibilità o meno di progetti nei diversi ambiti dipende dai progetti che l'azienda sta svolgendo o che deve far partire. Nello specifico, per lo sviluppo delle *Web Application*, agli stagisti vengono affidati prevalentemente compiti o ruoli relativi al *frontend*. Ossia la creazione di interfacce grafiche, del loro funzionamento logico e di collegarlo con un eventuale *database* o *backend* già impostato tramite *API* e fornite da un altro membro del *team* che si occupa dello sviluppo lato *backend*. La scelta di far svolgere attività relativamente legate al *frontend*, è data dal fatto che lo sviluppo lato *backend*, sempre nell'ambito delle *Web Application*, non richiede troppo lavoro, dato anche il fatto che si cerca di mantenere, il più possibile, un ambiente comune tra i vari *backend* di diverse *Web Application*. Inoltre, al momento, il *team* ha già personale che si occupa dello sviluppo lato *backend* mentre è in carenza di figure che si occupano dello sviluppo lato *frontend*.

2.2 Introduzione al progetto

Lo scopo del progetto era realizzare un'applicazione *web*, nello specifico una PWA, ovvero un'applicazione che si può utilizzare sia nel *browser* sia come un'applicazione normale. Concetto del progetto era di creare una *landing page* per prodotti alimentari dove, l'utente, tramite scansione del codice a barre presente sul prodotto, potesse avere accesso ad una serie di informazioni, inserire recensioni o visualizzare recensioni di altri clienti. L'applicazione doveva consentire all'utente non registrato la possibilità di ricercare prodotti e visualizzare informazioni e recensioni. Mentre all'utente registrato, tramite *social*, era consentito anche di lasciare recensioni. Per poter integrare le diverse

funzioni necessarie ho utilizzato diversi moduli *Node.js* studiati e decisi insieme al responsabile del progetto. Tali moduli hanno semplificato il lavoro andando a ridurre il codice necessario per introdurre diverse *feature*. L'applicazione interagiva con un *database* esterno per il recupero dei dati sui prodotti, mentre interagiva con un *database* interno per registrare nuovi utenti o il *login* di utenti già registrati, la creazione, modifica, rimozione o recupero delle recensioni. Per configurare tali interazioni è stato usato il *framework Django* e sono stati configurati e implementati diversi *endpoint*.

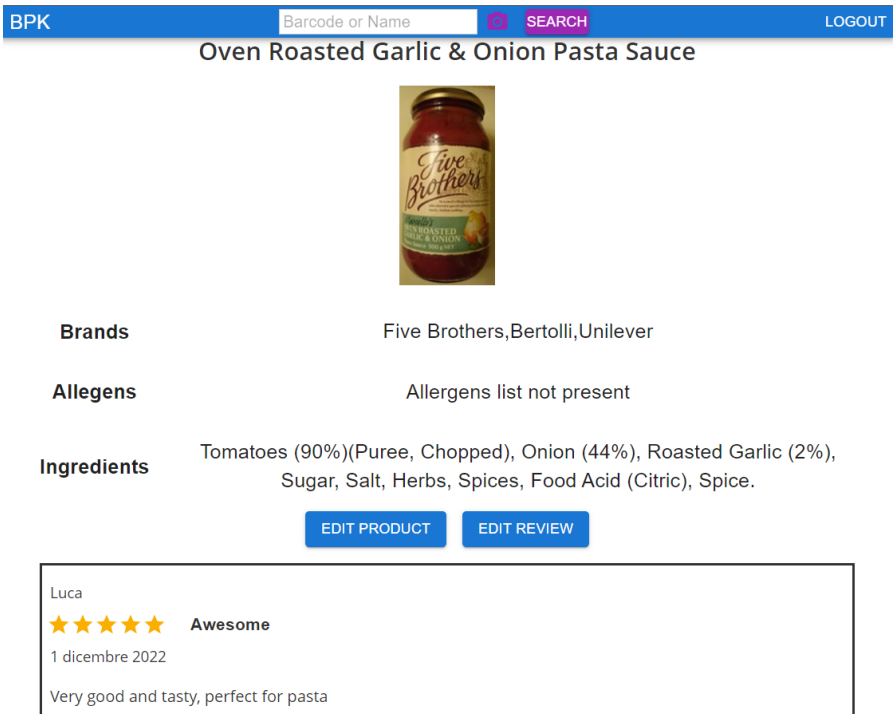


Figura 2.1: Pagina in cui l'utente atterra dopo aver ricercato un prodotto o eseguito una scansione del codice a barre con successo. Realizzazione finale dell'idea.

2.3 Importanza del progetto per l'azienda

Tale progetto è piuttosto importante per S.AI. È molto ampio e coinvolge varie aree di BPK. S.AI già collabora allo sviluppo del *software* gestionale di BPK, ha partecipato attivamente alla definizione del progetto da me svolto nella sua interezza ed è responsabile diretta di alcune componenti, fra cui la componente di *data exchange* (una *web app* dedicata agli utenti finali) e una componente di analisi che deve scandagliare diverse fonti per fornire informazioni utili ai produttori dei vari prodotti. La *web app* per gli utenti finali, così come definita nel progetto, doveva fungere in primo luogo da *landing page* per l'utente quando questo scansiona un codice a barre di un prodotto. In questa *landing page* dovevano essere mostrati vari dati deducibili dal codice scansionato o da altre fonti. Il progetto della *community platform* è un primo prototipo di questa *landing page*. Alle versioni successive, sviluppate dopo la conclusione del mio *stage*, si dovrebbero aggiungere informazioni e dovrebbero essere aggiunti metodi per scansionare un prodotto (per es., usando *QR code* o *NFC* o *watermarks*).

2.4 Obiettivi del progetto

Il *tutor* aziendale che è il responsabile del progetto, dopo un analisi di ciò che doveva garantire l'applicativo, ha individuato una serie di obiettivi suddividendoli in tre categorie: obbligatori, desiderabili e opzionali.

Obiettivi obbligatori
Visualizzazione di dettaglio dei prodotti
Autenticazione mediante *social*
Possibilità di *editing* delle informazioni di dettaglio dei prodotti
Possibilità di scrivere recensioni

Tabella 2.1: Obiettivi obbligatori

Obiettivi desiderabili
Ricerca attraverso scansione di un codice a barre/*QR code*
Test di unità esaustivi

Tabella 2.2: Obiettivi desiderabili

Obiettivi opzionali
Possibilità di cercare prodotti per stringa
Possibilità di configurare *alert* relativi ai dettagli di un prodotto
Acquisizione immagini per il dettaglio dei prodotti
Acquisizione immagini per le recensioni

Tabella 2.3: Obiettivi opzionali

2.5 Vincoli del progetto

2.5.1 Vincoli temporali

Lo *stage* è durato 8 settimane(lunedì-venerdì) con giornate lavorative di 8 ore. Prima di iniziare lo *stage*, il responsabile del progetto ha redatto un piano di lavoro. Il piano è stato suddiviso nel seguente modo:

* **Settimana 1:** Studio della libreria *React*, dell'utilizzo di *Node.js* per installare moduli per la *Web App* e studio iniziale della libreria *Redux*. Creazione di un prototipo di *Web App* in *React* a cui aggiungere le future funzionalità;

* **Settimana 2:** Studio più approfondito della libreria *Redux* e delle *API* per la ricerca di prodotti. Studio, progettazione e implementazione dell'interfaccia di accesso/registrazione tramite *social network*;

* **Settimana 3:** Approfondimento libreria *Redux*; progettazione e implementazione ricerca per codice dei prodotti e la vista in dettaglio di essi;

* **Settimana 4:** Ultimazione studio libreria *Redux* e raffinamento dei processi di *accesso/registrazione*; studio, progettazione e implementazione *scanner* per ricerca prodotti tramite codice a barre. Studio *API* per modificare dettagli prodotto, studio, progettazione e implementazione *editor* recensioni;
* **Settimana 5-6:** Miglioramento ricerca prodotti e vista dettagli di essi; implementazione *editor* recensioni ed *editor* informazioni sui prodotti ;
* **Settimana 7:** Raffinamento *editor* informazioni sui prodotti ed *editor* recensioni;
* **Settimana 8:** Raffinamento *editor* recensioni.

Il lavoro svolto veniva tracciato tramite *commit* su *GitLab* e commenti o chiusura tramite *ticket* di *Jira*.

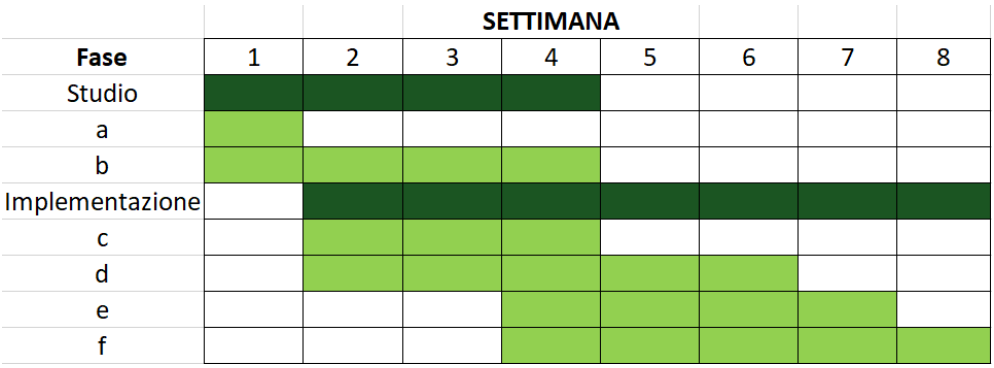


Figura 2.2: Piano di lavoro settimanale

2.6 Lavoro remoto

Come consigliato dal *tutor* aziendale, ho svolto lo *stage* interamente da remoto senza mai andare alla seda aziendale, dove non avrei trovato quasi nessuno. Tutto il *team* lavora da remoto e tramite le applicazioni citate sopra, la coordinazione e pianificazione del lavoro è sempre stata alla massima efficienza. Anche se da remoto, non mi sono mai sentito solo o abbandonato, dato che che in caso di aiuto o per 5 minuti di pausa c'era sempre qualcuno disponibile per scambiare due messaggi o fare una chiamata vocale. Inoltre, tramite stato personale di *Slack*, si può indicare agli altri la propria situazione lavorativa: al lavoro, a pranzo, in pausa, ecc.

2.7 Scelta del progetto

Provenendo da un istituto alberghiero non avevo mai fatto *stage* relativi all'ambito informatico e questa è stata la mia prima esperienza nel settore.
All'inizio ho cercato uno *stage* che avesse progetti riguardanti l'intelligenza artificiale, ma quelli che c'erano o erano al completo o troppo distanti da Vicenza. Ho scelto S.AI come azienda perché interessato dai vari progetti e ambiti seguiti dall'azienda, tra cui intelligenza artificiale ma di cui non erano presenti progetti al momento dello *stage*. Un altro motivo per cui ho scelto tale azienda era il progetto su cui ho lavorato, dato che lo

sviluppo di *Web App* mi era interessato parecchio anche durante gli studi all'università. Il fatto che proponessero l'utilizzo di *React* unito all'uso di moduli tramite *Node.js* e all'utilizzo della libreria *Redux* mi ha interessato molto poiché sono tecnologie oramai usate, diffuse e che volevo apprendere. La mia mancanza di "esperienza" non è stato di nessun problema per l'azienda e il *tutor* è stato felice che avessi seguito un corso per l'utilizzo di *Git* e avessi sperimentato la pratica dello *Scrum* durante il progetto di *Ingegneria del Software*. Gli obiettivi personali che mi sono posto di raggiungere durante lo *stage* sono i seguenti:

* Apprendimento del *framework React*;
* Apprendimento della libreria *Redux*;
* Consolidamento delle buone pratiche di lavoro e degli *standard* tecnologici e di codifica;
* Consolidamento della pratica dello *Scrum*, dei suoi vantaggi e dei suoi metodi di lavoro;
* Capacità di lavorare in autonomia e di valutare attentamente e logicamente eventuali scelte;
* Valutazione se il percorso di studi e lavoro futuro è stata effettivamente una buona scelta;
* Valutazione di quanto utili possano essere state le nozioni apprese durante il percorso universitario.

Capitolo 3

Resoconto del lavoro svolto

3.1 Attività di studio

Il tempo che ho dedicato allo studio delle tecnologie durante lo *stage* può essere suddiviso in: tempo dedicato durante la prima settimana e tempo dedicato durante il resto dello *stage*. Ho dedicato la prima settimana di *stage* ad uno studio iniziale di alcune delle tecnologie che ho usato le settimane seguenti. Mentre, durante lo *stage*, ho continuato a studiare alcune tecnologie per approfondire alcuni argomenti o poiché venivano introdotte nuove tecnologie.

3.1.1 Prima settimana dedicata allo studio

Prima di tutto ho studiato la libreria *React*, leggendo la documentazione ufficiale e seguendo dei corsi intensivi sulle caratteristiche base e i vantaggi che offriva. Ho accompagnato lo studio della libreria alla pratica, andando a creare un primo prototipo del sito nel quale testavo le varie cose che studiavo. Questo prototipo è stato la base per le *feature* che ho integrato in seguito. Oltre a *React*, ho cominciato uno studio basilare di *Redux* e delle sue funzioni combinate con *React*. Lo studio di tale libreria è continuata nelle settimana seguenti, per approfondire alcuni argomenti non troppo chiari all'inizio, ma che poi sono stati più facili da trattare e necessari da approfondire per implementare alcune *feature*. Durante questa settimana ho anche studiato le *feature* offerte e i vantaggi ottenuti dagli strumenti *Jira* e *GitLab*.

3.1.2 Studio durante lo *stage*

Durante lo *stage* ho dedicato più volte tempo ad approfondire funzionalità offerte da alcuni servizi o a studiare nuove tecnologie introdotte durante lo sviluppo per facilitare il lavoro di codifica, facilitare alcuni aspetti del progetto o rispettare alcuni *standard*. Tra le varie cose che ho studiato durante la progettazione o codifica sono:

* Durante la seconda settimana ho dovuto implementare l'autenticazione tramite *social network*. Per questo ho dovuto studiare la documentazione sia di *Facebook* che di *Google* per poter poi procedere alla implementazione e configurazione dei processi di autenticazione.
  Tale studio è servito sia a me che all'intero *team*, dato che non era ancora stata implementata nessuna funzione del genere;

* Durante lo *stage* ho dovuto utilizzare diversi moduli per implementare funzionalità utili o vantaggiose. Ogni volta che ne implementavo una dovevo studiarne le *feature* offerte e le modalità d'utilizzo. Uno dei moduli più ostici da capire e implementare a dovere è stato il modulo per effettuare scansioni dei codici a barre;

* ***Material UI***: Libreria per *React* che permette di utilizzare e implementare il *Material Design* di *Google*. Include una raccolta completa di componenti grafici pronti per l'utilizzo, permettendo allo sviluppatore di concentrarsi sul codice senza dover reinventare componenti grafici o di preoccuparsi dell'aspetto del sito. Inoltre la libreria offre un ampia personalizzazione dei componenti grafici per potersi adattare ai diversi gusti o esigenze;

* ***ESlint***: Per lo sviluppo di applicazioni *web*, l'azienda si basa sulla configurazione di *ESlint* di *AirBnb* che rispecchia lo *standard JavaScript* per quanto riguarda la formattazione e il codice. Ho dovuto studiare le varie regole, motivazioni di alcuni divieti e possibili eccezioni alle regole;

* ***RTK Query***: *Feature* offerta dalla libreria *Redux* che permette al sito o all'utente di interagire in diversi modi con un *server* o con un *database*;

* ***Jest*** e ***Testing Library***: Spesso usati insieme quando si sviluppa in *React* e permettono di implementare vari tipi di *test* per testare l'applicazione in modo semplice e chiaro. *Jest* è una libreria per testare *JavaScript* mentre *Testing Library* è una libreria per testare le componenti grafiche.

3.1.3 Risultati della attività di studio

Dall'attività di studio della prima settimana e dalle varie attività di studio svolte durante lo *stage*, ho imparato diverse cose che riporto di seguito:

* Utilizzo avanzato del linguaggio *React* e dei suoi vantaggi per la creazione di componenti *frontend*;

* Utilizzo avanzato dello strumento *Redux* per potenziare le possibilità dei componenti *React*;

* Utilizzo approfondito degli strumenti *Jira* e *GitLab*;

* Come implementare in *React* e configurare l'autenticazione tramite *Facebook* e *Google*;

* Utilizzo intermedio della libreria *Material UI* per la creazione della grafica del *frontend*;

* Utilizzo e configurazione base delle regole decise tramite *ESlint*;

* Utilizzo buono della *feature RTK Query* offerta da *Rexud*;

* Utilizzo base delle librerie *Jest* e *Testing Library* per l'implementazione di *test*.

3.2 Flusso di lavoro

Avendo seguito una metodologia di sviluppo *Scrum*, durante lo *stage* ho ripetuto più volte le attività di progettazione e codifica invece che una volta sola. Il mio flusso di lavoro durante lo *stage* ha seguito *Sprint* di durata settimanale, con incontro settimanale fissato a lunedì per discutere con il resto del *team* cosa era stato fatto, *ticket* rimasti in sospeso ed eventuali *ticket* bloccati ed il motivo per cui erano bloccati. Durante questo incontro si decideva gli obiettivi minimi dello *Sprint* successivo e si davano eventuali *feedback* sul metodo di lavoro o sulla settimana di lavoro trascorsa. Ogni *ticket* seguiva il seguente flusso di lavoro: ogni nuovo*ticket* si trovava nello stato *Da completare* e potrebbe essere già stato assegnato a qualcuno; in caso contrario se un *ticket* rientra nella propria area di competenza e si è liberi ci si può assegnare i *ticket* non assegnati. Se lo sviluppatore comincia uno dei *ticket* a lui assegnati, il *ticket* viene spostato nello stato *In Corso*. Una volta che lo sviluppatore ha terminato il proprio lavoro, sposta il *ticket* nello stato *Da Verificare*, tramite commento nel *ticket* segnala che è possibile effettuare la revisione e la assegna a uno degli altri membri. Ad inizio *stage* questo ruolo ero ricoperto quasi esclusivamente dal responsabile di progetto. Il verificatore poteva richiedere eventuali cambiamenti o segnalava errori e in questo caso spostava il *ticket* dallo stato *Da verificare* allo stato *Da completare*, in modo da seguire la logica del flusso di lavoro, e segnalava la richiesta di cambiamenti tramite commento nel *ticket*. Se invece la verifica andava a buon fine, il *ticket* veniva spostato dallo stato *Da verificare* allo stato *Done* e il verificatore notificava l'avvenuta chiusura del *ticket* tramite commento. Per ogni *ticket* esiste un corrispettivo *branch* su cui lavorare e una volta chiuso il *ticket*, ne viene fatto il *merge* nel *main* ed eliminato.

3.3 Progettazione

Dopo aver svolto gli studi iniziali necessari ho cominciato a sviluppare l'applicazione. Come già detto in precedenza, c'è stato un ciclo continuo di progettazione e codifica come succede con i flussi di lavoro di tipo *SCRUM*. In seguito andremmo a vedere alcune delle progettazioni principali o che hanno richiesto più tempo o impegno.

3.3.1 Progettazione dei componenti

Per quanto riguarda i componenti *React*, la loro progettazione seguiva sempre il seguente flusso di lavoro:

* **Decisione del nome**: Anche se sembra un'operazione banale il nome deve far capire, a chiunque legga il codice, lo scopo del componente/classe e l'ambito o l'oggetto di riguardo (ad esempio recensioni o prodotti);
* **Moduli esterni**: Decidere se e quali moduli esterni siano necessari per implementare tale componente o la *feature* che offre. Tale necessità dipende dal fatto o meno che la *feature* sia creabile anche senza componenti esterni o se il lavoro dello sviluppatore viene facilitato o ridotto. Nel considerare i vari moduli si considerava l'ultimo aggiornamento eseguito, la frequenza di aggiornamenti, eventuali *bug* riportati dagli altri utenti ed popolarità del modulo;
* **Eventuale scomposizione**: Decidere se il componente è abbastanza atomico; in caso contrario decidere in quali e quanti componenti scomporlo od utilizzare classi puramente in *JavaScript* per implementare funzioni da utilizzare nel componente.

3.3.2 Progettazione del *login* tramite *Facebook* e *Google*

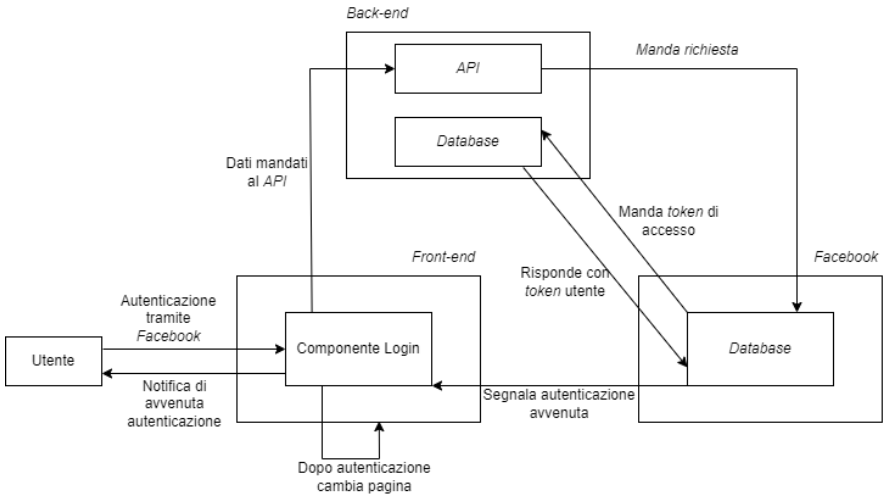


Figura 3.1: *Workflow* di un'autenticazione tramite *Facebook* (versione finale)

È stata la prima pagina e *feature* che avevo progettato e implementato durante la seconda settimana di *stage*. Mi aveva richiesto alcuni giorni per poter testare e configurare correttamente la funzione di *login* sul sito di sviluppo dei *social network* e ideare come implementare tale funzione nell'applicativo. Alla fine dell'implementazione di tale *feature*, avevo redatto per i miei colleghi un *README* che ho posto all'interno della *repository* del progetto, in modo che potessero consultarlo in futuro. Entrambi i servizi avevano richiesto passaggi simili, ossia:

* Se non posseduto, creazione di un *account Google* o *Facebook Developers*;
* Creazione di un applicazione sul sito di sviluppo per *developers* dei *social network*;
* Aggiunta del servizio di *login*, sul sito di sviluppo, con aggiunta di permessi e domini accettati;
* Configurazione lato *backend*, tramite *Django*, delle impostazioni necessarie.

La configurazione di questi servizi le avevo svolte inizialmente tramite *account* personale, per non "sporcare" la storia di configurazione con i *test*. Dopodiché, una volta assicuratomi del giusto procedimento da seguire e configurazione da applicare, avevo utilizzato un *account* aziendale per configurare l'applicativo. Il *login* tramite *Google* è stato semplice e abbastanza immediato. Invece il *login* tramite *Facebook* ha richiesto più tempo e un paio di modifiche extra lato *backend*. Tali modifiche riguardavano l'utilizzo e il *test* del *login* tramite *https* e non *http*, con l'aggiunta di un *file .env* e installazione di dipendenze. Tali modifiche si sono rese necessarie poiché *Facebook*, anche solo per il *test*, è molto restrittivo e richiede, senza eccezioni, l'accesso al suo servizio di *login* solo tramite *https*. Non avendo un *server* aziendale su cui far girare l'applicativo ma solo tramite *localhost*, le modifiche riportate prima sono state l'unica soluzione. La prima progettazione di tale *feature* era abbastanza base, relativa alla sola pagina di *login* e non causava effetti che si propagavano al resto dell'applicativo. Se il *login* avveniva con successo, il *form* di *login* veniva sostituito da un messaggio di

benvenuto. Dopodiché, ho progettato tale *feature* in modo che il *login* fosse effettivo in tutte le pagine dell'applicazione e impedisse l'accesso ad alcune pagine o funzionalità a cui potevano accedere solo gli utenti autenticati.

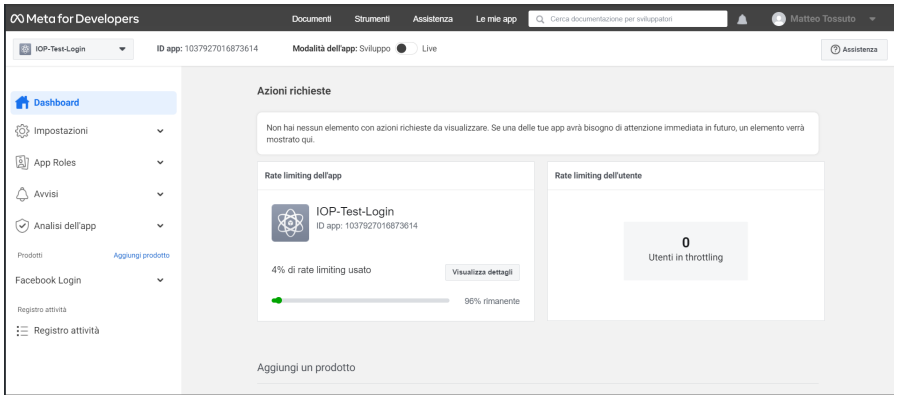


Figura 3.2: Sito sviluppatore *Facebook*

3.3.3 Progettazione della *feature* di *scanner*

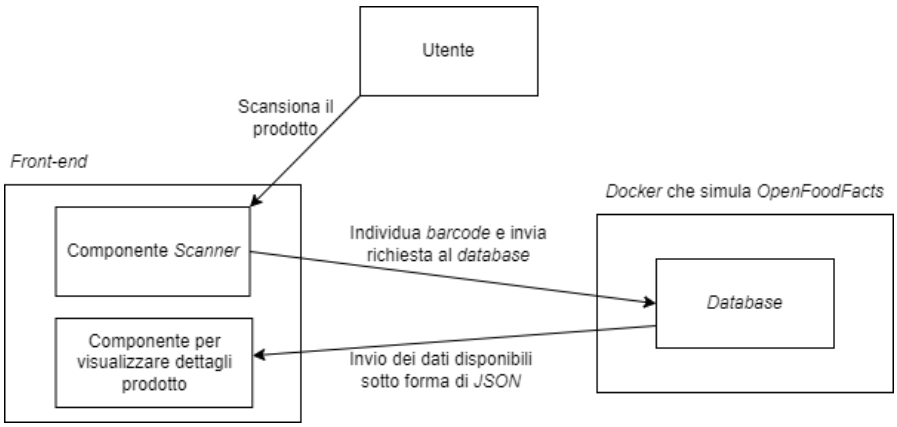


Figura 3.3: *Workflow* di una scansione di un codice a barre

È stata la *feature* più ardua e importante che ho dovuto progettare e implementare; mi aveva richiesto quasi una settimana intera di tempo tra progettazione e implementazione. In prima battuta ho dovuto cercare uno o più moduli che permettessero di utilizzare le fotocamere di computer e telefono, effettuare scansioni e recuperare un determinato tipo di codice a barre. Molti dei pacchetti erano vecchi o non aggiornati, ma alla fine sono riuscito a trovarne uno. Dopodiché, una volta studiata la documentazione, creata per una vecchia versione di *React*, ho dovuto capire come poter utilizzare le funzioni offerte da tale modulo con la nuova versione di *React*. Questo per poter mantenere tutte le componenti aggiornate all'ultima versione, seguire gli *standard* e poter utilizzare tutti i vantaggi che l'ultima versione comporta.
Dato che questa *feature* l'avevo implementata durante la seconda settimana, non avevo ancora abbastanza padronanza di *React*. Allora, insieme al mio *tutor*, abbiamo deciso di

Figura 3.4: Tentativo di scansione del codice a barre di un prodotto tramite *scanner* che ho implementato

progettare tale *feature* con il vecchio modello di *React*, in modo da dover solo adattare la documentazione del modulo al caso d'uso del progetto e creare un *poc* iniziale con la progettazione di due componenti. Durante la seconda metà dello *stage*, ho deciso di riprendere in mano tale *feature* e progettarla in modo che rispettasse le nuovi versioni di *React* e passare da uno stato di *poc* a uno stato ben definito e funzionale. Durante questa progettazione mi sono reso conto dell'avanzamento che avevo fatto, poiché è stato quasi immediato progettare tale *feature* in maniera ottimale passando addirittura dall'utilizzo di due componenti a uno solo e progettando le seguenti *feature* aggiuntive:

* Pulsante per chiudere la finestra per la scansione che chiude sia la finestra che la *webcam* (nel *poc* veniva chiusa solo la finestra e non la *webcam* che rimaneva invece attiva);

* Pulsante che permette all'utente di passare dalla fotocamera frontale del telefono alla fotocamera posteriore e viceversa. Inoltre, se sono presenti più fotocamere

posteriori viene presa quelli più adatta e funzionante, ossia che non presenta distorsioni.

3.3.4 Progettazione del *refactor* del codice per implementare *Material UI* ed *Eslint*

Durante lo svolgimento del progetto ci sono state delle aggiunte che mi hanno portato a riprogettare il codice ossia a svolgere un *refactor* di esso. Tali aggiunte sono state:

* ***Material UI***: per un miglioramento grafico del sito;
* **Configurazione di *ESlint***: per adattare il codice a *standard* di formattazione e di codifica.

Nel primo caso, con *Material UI*, ho dovuto riprogettare il codice per poter successivamente sostituire i normali *tag HTML*, con i *tag di Material UI*. Questo primo *refactor* non ha richiesto troppo tempo, lo avevo svolto alla fine del primo mese di *stage* e mi ha portato a riprogettare interi componenti, andando però a renderli più semplici e a rimuovere opzioni di stile dal *file CSS*. Il secondo caso, con *ESlint*, mi ha portato a riprogettare parti di componenti, componenti interi o classi *JavaScript* per poter adattare il codice agli *standard*, rispettare le regole di formattazione, evitare alcuni errori di codifica comuni e ambiguità che potevano portare ad errori.

3.4 Codifica

Dopo ogni attività di progettazione passavo all'attività di codifica e implementazione della soluzione ideata.
Grazie a tutto il lavoro di studio svolto inizialmente e durante lo svolgimento dello *stage* e grazie ad una buona progettazione svolta ad ogni *Sprint*, ogni attività di codifica non ha avuto grosse criticità e il lavoro è stato svolto nei tempi previsti dello *Sprint*. A volte il lavoro terminava addirittura prima permettendomi cosi di iniziare a documentarmi e/o progettare per ulteriori *ticket* non previsti per lo *Sprint* in corso. In seguito andremmo a vedere alcune delle soluzioni principali o che hanno richiesto più tempo.

3.4.1 Implementazione dei componenti

Per quanto riguarda i componenti *React*, la loro implementazione seguiva sempre il seguente flusso di lavoro:

* **Moduli Esterni**: Installazione, tramite *Node.js*, dei moduli scelti durante la progettazione e seguente importazione di tali moduli nel componente. L'ordine delle importazioni doveva rispettare rigide regole di *ESlint*: prima i moduli esterni e poi quelli interni, in ordine alfabetico;
* **Implementazione della soluzione**: Si passava poi alla stesura vera e propria della soluzione individuata, ponendo attenzione a rispettare le regole di *ESlint* ed utilizzando, ove possibile, il modulo *Material UI* invece dei *tag HTML* per la componente grafica;
* **Sottocomponenti**: Durante la stesura della soluzione e del componente, andavo a implementare gli eventuali sottocomponenti individuati durante la progettazione

del componente principale. La stesura dei sottocomponenti seguiva la stessa procedura per l'implementazione del componente principale.

3.4.2 Implementazione della logica di *login*

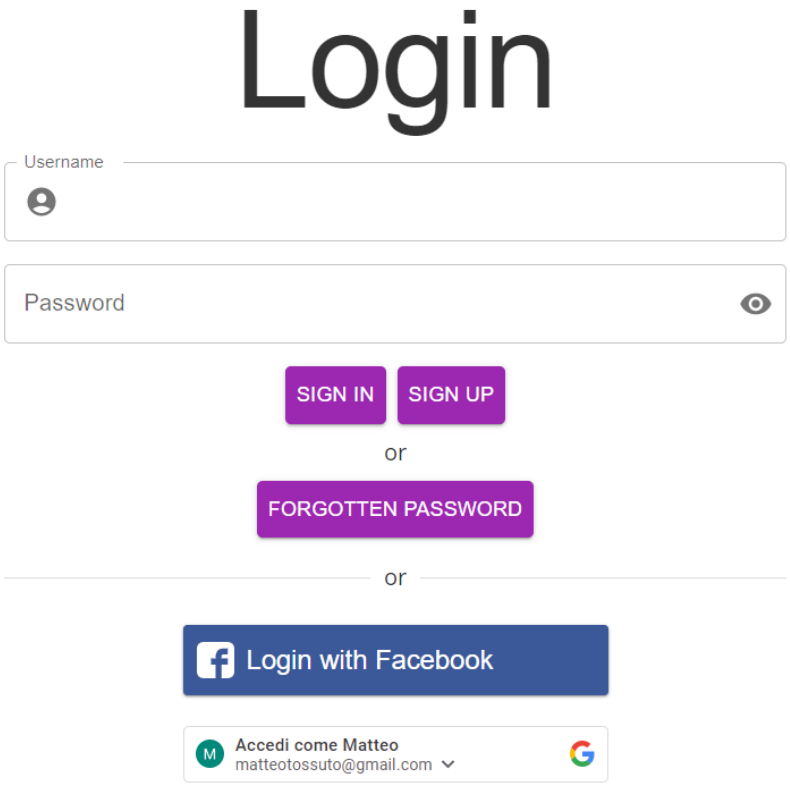


Figura 3.5: Versione finale della pagina di autenticazione e registrazione

La pagina di registrazione e autenticazione è stato il componente che ha subito più modifiche e miglioramenti durante lo *stage*. Possiamo paragonare l'evoluzione del componente di *login* all'evoluzione del mio apprendimento di *React* e di *Redux*.
Inizialmente, avevo implementato l'autenticazione tramite *Facebook* e il processo influenzava solo la pagina di *login* e non il resto del sito. Questo, poiché essendo ancora all'inizio del progetto, non avevo ancora abbastanza conoscenza per utilizzare *Redux* e le sue funzionalità per propagare lo stato di autenticazione al resto del sito. Dopodiché ho implementato anche l'autenticazione tramite *Google*.

L'implementazione dell'autenticazione di *Facebook* è stata più impegnativa dell'implementazione dell'autenticazione tramite *Google*; questo poiché *Facebook* è molto restrittiva e non permette di utilizzare i suoi servizi tramite protocollo *HTTP* ma solo tramite protocollo *HTTPS*. Mentre il mio responsabile ha aggiunto la possibilità di utilizzare il *backend* in *localhost* con protocollo *HTTPS*, io ho aggiunto la possibilità di utilizzare il *frontend*, in *localhost*, tramite protocollo *HTTPS*.
Per fare ciò, ho dovuto semplicemente creare un *file .env* dove salvare l'opzione di

volere utilizzare il protocollo *HTTPS*. Inoltre, anche se andava fatto durante la progettazione, ho dovuto completare la configurazione dell'applicazione presente sul sito sviluppatore di *Facebook*, poiché nella documentazione ufficiale non erano riportate alcune impostazioni importanti da configurare.

A metà del primo mese di *stage*, avendo finalmente approfondito per bene *Redux*, ho potuto implementare la soluzione per propagare al resto del sito l'effetto di un autenticazione da parte di un utente. Reindirizzandolo, una volta autenticato, alla pagina dalla quale aveva richiesto il *login* o permettendoli l'accesso alle aree riservate solo agli utenti autenticati.

Verso la fine dello *stage*, a causa di una modifica negli obiettivi di progetto di cui parleremo più avanti, ho introdotto le *feature* di registrazione e *reset* della *password*; andando cosi a rendere completo, sotto ogni aspetto, il componente di registrazione e autenticazione.

3.4.3 Implementazione delle *API* lato *frontend*

```
const apiReviewsSlice = apiSlice.injectEndpoints({
    endpoints: builder => ({
        getReviews: builder.query({
            query: ({ ean, mine }) => ({
                url: `/community/products/${ean}/reviews/`,
                params: {
                    mine,
                },
            }),
            providesTags: ['Reviews'],
        }),
        addNewReview: builder.mutation({
            query: ({ payload, ean }) => ({
                url: `/community/products/${ean}/reviews/`,
                method: 'POST',
                body: payload,
                headers: {
                    'Content-Type': 'application/json',
                    'X-CSRFToken': Cookies.get('csrftoken'),
                },
            }),
            invalidatesTags: ['Reviews'],
        }),
```

Figura 3.6: Esempi di *query*

Per poter implementare le *API* ho dovuto prima terminare lo studio di *Redux* e dello strumento da esso offerto: *RTK Query*.

Tale strumento facilita la realizzazione di *query* per estrarre o manipolare diversi dati; allo sviluppatore basta dichiarare gli *endpoint delle API*, un eventuale *URL* di base per ridurre il codice, dichiarare il tipo di operazione se diversa da quella di *get*, dichiarare eventuali parametri o *body* della *query*, includere credenziali se richieste ed infine eventuali *headers* come ad esempio il passaggio di *cookies*, il tipo di dato passato o in che forma ci si aspetta di ricevere la risposta (es. *JSON*). Inoltre è possibile, tramite utilizzo di *tags*, dopo una *query*, resettare le operazioni di altre *query* e azionarle nuovamente in automatico.

3.4.4 *SLOC*

Per poter misurare la dimensione del progetto ho scelto di utilizzare la metrica *SLOC (source lines of code)*. Tale metrica è utilizzata per stabilire la complessità di un *software* e per stimare le risorse necessarie per la sua produzione e mantenimento.
Per misurare *SLOC* e il numero di *file* ho utilizzato un estensione di *Visual Studio Code*: ***VS Code Counter***.

Segue una tabella con il valore *SLOC* e il numero di *file* suddivisi per linguaggio e ruolo nel progetto.

Linguaggio e Ruolo	***File***	***SLOC***
***React* per componenti grafici**	26	1918
***JavaScript* per configurazione**	3	127
***JavaScript* per funzioni**	1	30
JavaScript* per *API	9	306
CSS	1	80
***JSON* per localizzazione**	2	528
Totale senza *test*	42	3295
Test	8	305
Totale	50	3600

Tabella 3.1: *SLOC* e numero *file*. Il valore *SLOC* rappresenta il massimo valore tra i file.

3.5 Test

L'attività di *test* non era presente nei primi *Sprint*, almeno non in forma automatica, poiché prima di ogni *commit*, da un *branch* di un *ticket* al *branch* principale, controllavo a mano ogni *feature* implementata e questo ha comunque aiutato a scoprire alcuni *bug*.
L'implementazione dei *test* è diventata parte dello *Sprint* dal secondo mese di progetto in poi. Questo poiché prima non avevo abbastanza conoscenze di *React* e inoltre ho dovuto studiare le librerie *Jest* e *Testing Library*.
Ho implementato un *test* per controllare la singola classe in puro *JavaScript*, con una copertura del 100%, e ho implementato dei *test* per la maggior parte dei componenti *React*, con una copertura accettabile.
La mancanza di alcuni *test* è dovuta al fatto che, insieme ad un mio collega, stavamo ancora valutando alcuni moduli per poter simulare le *query* necessarie e fare un *mock* dei dati.

3.5.1 Test *JavaScript*

Ho testato l'unica classe in puro *JavaScript* presente. Ho creato un *test* d'unità per ogni funzione che la classe implementava e testavo più scenari possibili.

```
test('check if image exist or not', () => {
  //arrange
  const data = 'http://images.openfoodfacts.localhost/images/products/541/080/395/0689/front_fr.10.200.jpg';
  const emptyData = '';
  const notFoundImage = 'https://i.imgur.com/QtFrFJj.png';
  //act
  let output = CheckDataProduct.checkImage(data);
  expect(output).toBe(data);
  output = CheckDataProduct.checkImage(emptyData);
  expect(output).toBe(notFoundImage);
});

test('check if productCode is a barcode or a word', () => {
  //arrange
  const barcode = '5410803950689';
  const barcodeWithSpace = ' 5410803950689 ';
  const word = 'pasta';
  const wordWithSpace = ' pasta ';
  //act
  let output = CheckDataProduct.isProductCode(barcode);
  expect(output).toBe(true);
  output = CheckDataProduct.isProductCode(word);
  expect(output).toBe(false);
  output = CheckDataProduct.isProductCode(barcodeWithSpace);
  expect(output).toBe(true);
  output = CheckDataProduct.isProductCode(wordWithSpace);
  expect(output).toBe(false);
});
```

Figura 3.7: Esempio di un *test* d'unità per *JavaScript*

3.5.2 Test componenti *React*

Per testare i componenti *React* ho creato dei *test* d'unità che verificassero la corretta generazione della pagina *web* dal codice *React*. Prima di tutto il *test* andava a fare un *render* del sito e poi verificava la presenza di testi, bottoni o altri elementi grafici.

```
test('renders EditOrCreateReview page', () => {
  // Arrange
  render(
    <Provider store={store}>
      <Router>
        <Routes>
          <Route path="/" element={<EditOrCreateReview />} />
        </Routes>
      </Router>
    </Provider>
  );

  //Assertions
  expect(screen.getByText('reviews.form.heading')).toBeInTheDocument();
  expect(screen.getByText('reviews.form.fields.rating')).toBeInTheDocument();
  expect(screen.getByText('reviews.form.fields.title')).toBeInTheDocument();
  expect(screen.getByText('reviews.form.fields.description')).toBeInTheDocument();
  expect(screen.getByText('common.actions.send')).toBeInTheDocument();
  expect(screen.getByText('reviews.form.return')).toBeInTheDocument();
  expect(screen.getByRole('button', { name: 'common.actions.send' })).toBeInTheDocument();
  expect(screen.getByRole('button', { name: 'reviews.form.return' })).toBeInTheDocument();
```

Figura 3.8: Esempio di un *test* d'unità per una componente *React*

3.5.3 Risultati dei test

Di seguito una tabella contenente il numero di *file* in *React* e *JavaScript* testati, la copertura media e la media dei *test* d'unità implementati per componente/classe.

linguaggio	***File***	**Copertura**	***Test* d'unità**
React	22	67%	1
JavaScript	1	100%	4

Tabella 3.2: Tabella copertura *test* per linguaggio utilizzato

3.6 Modifica agli obiettivi del progetto

Durante il secondo mese di *stage*, ho concordato, con il responsabile, alcune modifiche agli obiettivi del progetto.

* **Obiettivi obbligatori**: Abbiamo aggiunto i seguenti obiettivi:
  - Registrazione classica al sito;
  - Autenticazione tramite nome utente e *password*.
* **Obiettivi desiderabili**: Abbiamo aggiunto i seguenti obiettivi:
  - *Reset* della *password*;
  - Possibilità di aggiungere prodotti non presenti nel *database* esterno;
  - Localizzazione in inglese e in italiano del sito.

Con le modifiche appena descritte la situazione definitiva per gli obiettivi di progetto era la seguente:

* **Obiettivi Obbligatori**: Sono passati da 4 a 6;
* **Obiettivi Desiderabili**: Sono passati da 2 a 5;
* **Obiettivi Opzionali**: Sono rimasti invariati.

3.7 Risultati

Al termine del periodo di *stage*, l'azienda e il responsabile del progetto sono rimasti soddisfatti del prodotto finale, contenti sia per la quantità che per la qualità del lavoro svolto.
Il prodotto ha raggiunto una maturità tale che il responsabile del progetto aveva pianificato di creare una struttura per poter mostrare o far provare una *demo* dell'applicativo al cliente.

Di seguito è presente una tabella che mostra il numero di requisiti soddisfati o meno, suddivisi per priorità.

Requisiti	Soddisfatti	Non soddisfatti
Obbligatori	6	0
Desiderabili	5	0
Opzionali	1	3

Tabella 3.3: Tabella soddisfacimento requisiti

Gli obiettivi opzionali non soddisfatti sono rimasti incompleti per le seguenti ragioni:

* **Possibilità di configurare *alert* relativi ai dettagli di un prodotto**: Obiettivo troppo avanzato per lo stato del prodotto e quindi mancata configurazione lato *backend*;
* **Acquisizione immagini per il dettaglio del prodotto**: *Bug* con la *API* fornita dal servizio esterno. Ho comunicato con il *team* di sviluppo di tale servizio (*OpenFoodFacts*) tramite apposito canale *Slack*; alla fine sono riusciti a risalire al problema e lo hanno sistemato ma oramai ero all'ultimo giorno di *stage*;
* **Acquisizione immagini per le recensioni**: Mancata configurazione lato *backend*.

Anche se tali requisiti non sono stati soddisfatti, il *tutor* e l'azienda hanno ritenuto il prodotto completo, almeno come prototipo, e soddisfacente.
Inoltre la presenza di requisiti non soddisfatti non aveva influenzato la *demo* prevista dal responsabile del progetto.

Capitolo 4

Valutazione retrospettiva

4.1 Soddisfacimento obiettivi

Lo *stage* si è concluso il 21 novembre 2022, ho soddisfatto tutti i requisiti obbligatori e desiderabili e solo 1 su 4 dei requisiti opzionali (tabella 3.3). Tali requisiti sono stati individuati inizialmente e poi modificati dal responsabile del progetto. Di seguito sono riportati i requisiti soddisfatti con un breve commento:

* **Obbligatori**
  - **Autenticazione mediante *social***: È stata la prima *feature* che ho implementato; ho apportato diverse modifiche e miglioramenti nel corso dello *stage*;
  - **Visualizzazione di dettaglio dei prodotti**: È stata la terza *feature* che ho implementato ed è stato semplice. Mi ha richiesto la creazione di una classe *JavaScript* per la gestione e scelta dei dati da visualizzare;

Figura 4.1: Esempio di un prodotto trovato ma con informazioni mancanti

  - **Possibilità di editing delle informazioni di dettaglio dei prodotti**: È stata tra le ultime *feature* che ho implementato; per tale operazione ho usato delle *API* fornite dal sito *OpenFoodFacts*;

– **Possibilità di scrivere recensioni**: È stata la quarta *feature* che ho implementato ed non è stato troppo complicato; è stato il primo caso in cui ho dovuto interfacciarmi con le *API* fornite dal nostro *backend* e quindi dell'utilizzo dello strumento *RTK Query*;

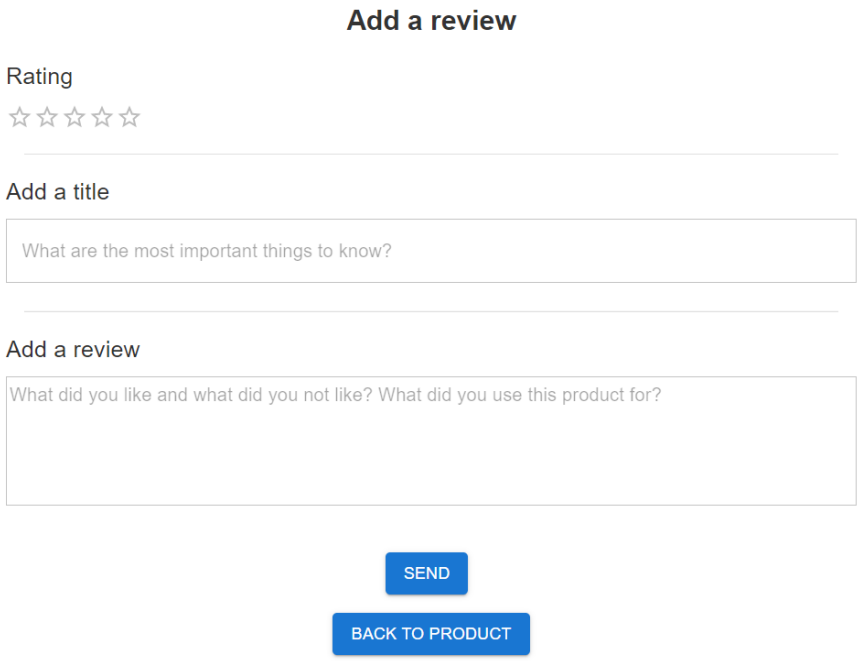


Figura 4.2: *Form* per l'aggiunta di recensioni

– **Registrazione classica al sito**: È stato uno dei requisiti aggiunti e quindi che ho implementato verso la fine del progetto;
– **Autenticazione tramite nome utente e *password***: È stato uno dei requisiti aggiunti durante lo *stage* e quindi che ho implementato verso la fine del progetto; tale requisito è figlio del requisito precedente.

∗ **Desiderabili**

– **Ricerca attraverso scansione di un codice a barre/*QR code***: È stata la seconda *feature* che ho implementato e tra le più impegnative, specialmente sotto l'aspetto della progettazione. Mi ha richiesto molto lavoro di studio dei moduli e progettazione della soluzione, che ha portato ad un primo *poc*; ulteriore lavoro di progettazione nella seconda metà dello *stage* mi ha permesso di portare la *feature* ad uno stato definitivo e ottimale;
– ***Test* di unità esaustivi**: Per implementare tale *feature* ho dovuto studiare ed utilizzare le librerie *Jest* e *Testing Library*; anche se la copertura dei *test* implementati non è totale, il lavoro che ho svolto è stato ritenuto soddisfacente e il requisito catalogato come completato dal responsabile del progetto;
– ***Reset* della *password***: Tale requisito è stato creato dopo l'aggiunta del requisito obbligatorio **Registrazione classica al sito** per una maggiore completezza, sicurezza e comodità per l'utente finale;

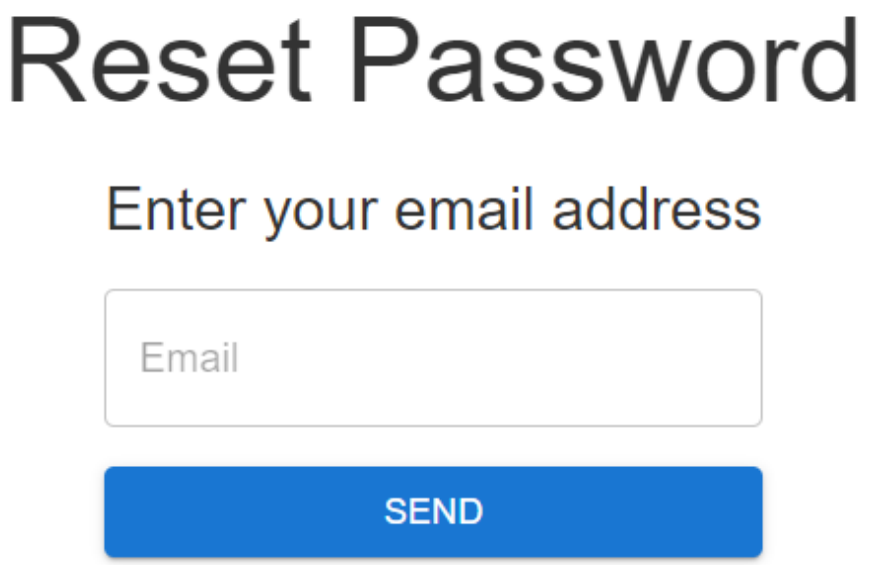


Figura 4.3: *Form* per richiedere *email* di cambio *password*

– **Possibilità di aggiungere prodotti non presenti nel *database* esterno**: Data la maturità del progetto verso fine *stage* è stato deciso di aggiungere questo requisito e farci aiutare dagli utenti a riempire il *database* di *OpenFoodFacts*;

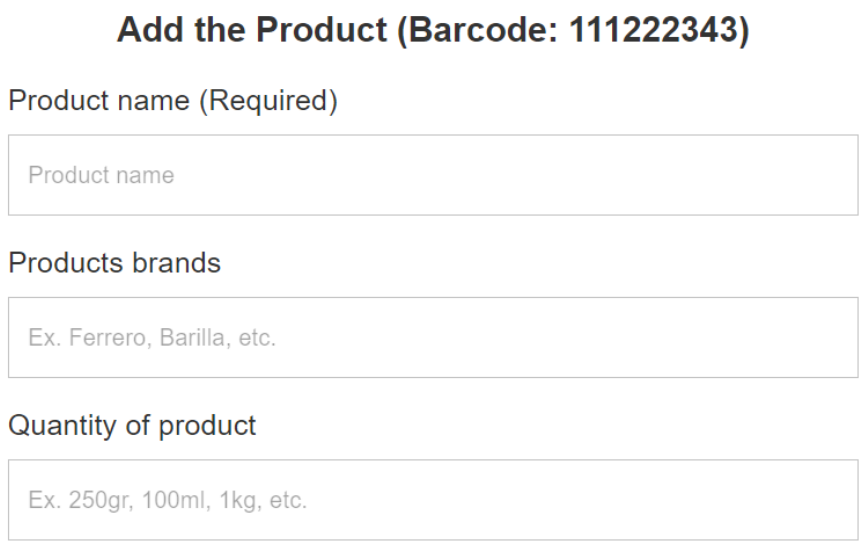


Figura 4.4: Parte di *form* per l'aggiunta o modifica dei dati del prodotto

– **Localizzazione in inglese e in italiano del sito**: Anche questo requisito è stato aggiunto una volta visto la maturità raggiunta dal progetto; l'idea

era di permettere anche ad utenti non residenti in Italia di poter utilizzare l'applicativo.

* **Opzionali**
  - **Possibilità di cercare prodotti per stringa**: Ho implementato tale requisito per poter offrire un alternativa di ricerca all'utente finale, nel caso in cui non potesse o non volesse utilizzare la ricerca tramite scansione del codice a barre. Requisito abbastanza semplice, che ho implementato tramite *API* fornite dal sito *OpenFoodFacts*.

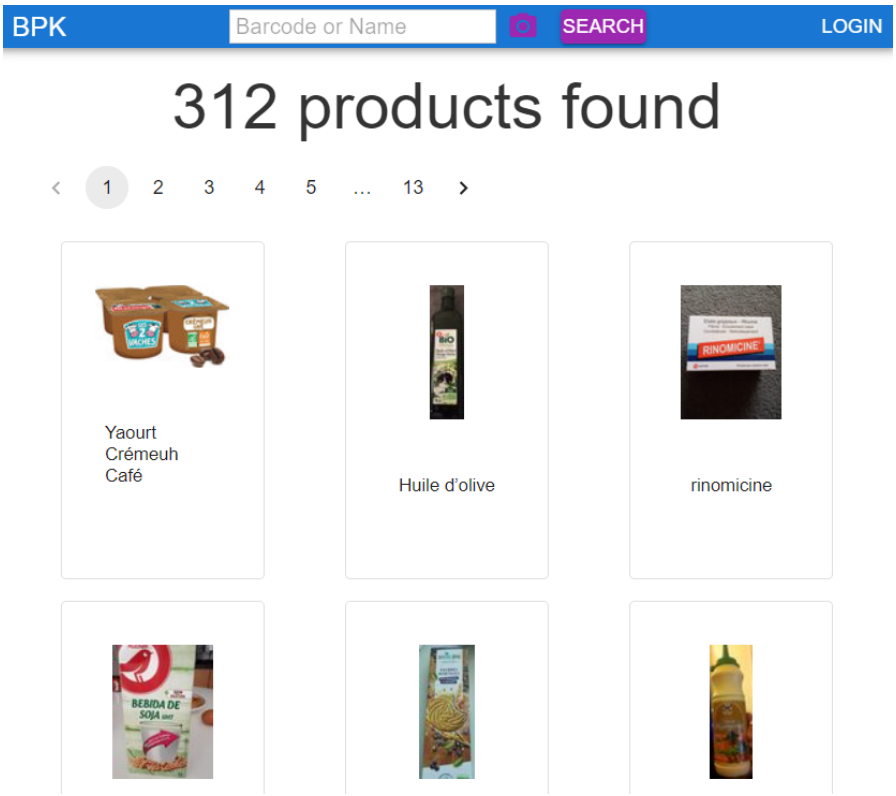


Figura 4.5: Pagina che mostra i risultati di una ricerca per stringa

L'aver soddisfatto quasi tutti i requisiti mi ha reso molto contento, anche se avrei voluto soddisfare anche quelli opzionali. Anche il mio *tutor*/responsabile del progetto ne è rimasto molto soddisfatto e contento.
Inoltre, dopo questo progetto, l'azienda ha visto il potenziale effettivo di *React* insieme a *Node.js* e ha già scelto di utilizzarli anche in altri progetti.

4.2 Conoscenze ottenute

Prima dell'inizio dello *stage*, le mie uniche conoscenze riguardo agli argomenti trattati o tecnologie usate durante il progetto, riguardavano *Git*, *Visual Studio Code* e il framework *Scrum*. Grazie allo *stage*, ho potuto ampliare di molto le mie conoscenze riguardo all'utilizzo di diverse tecnologie e strumenti, alla loro configurazione e al loro utilizzo per lo sviluppo di *Web Application* e la gestione e coordinazione del lavoro e del *team* di lavoro. Tali tecnologie e strumenti sono riportati di seguito:

* **React**: Ho fatto mio il linguaggio *React* con tutti i vantaggi e comodità che comporta; ho imparato a sfruttarlo in diversi ambiti dello sviluppo di *Web Application* e a gestire diversi scenari. Inoltre ho imparato come creare *file* di configurazione per configurare variabili d'ambiente per la sola *Web Application* e *file* di configurazione per creare *script* da utilizzare tramite *React*;
* **Node.js**: Ho imparato come utilizzare al meglio *Node.js* insieme a *React*, nello specifico come scaricare e gestire eventuali moduli da utilizzare;
* **Django**: Ho imparato a navigare l'interfaccia del *framework*, cercare le *API* create e testarle. Inoltre sono in grado di accedere alle sezione *admin* e manipolare i dati, ossia inserirli, modificarli ed eliminarli. Non ho imparato a creare e implementare le *API* ma questo esulava dal mio ruolo nel progetto;
* **Docker**: Non ho del tutto fatto mio *Docker*, solo alcuni aspetti base e fondamentali. Sono in grado di avviare, gestire e configurare in modo limitato i container, posso configurare in modo limitato le immagini e so dove cercare e reperire alcune informazioni essenziali;
* **GitLab**: Ho fatto mio tale servizio. In particolare la creazione e gestione dei *branch* e delle *merge request*.
* **Jira**: Ho fatto mio tale servizio. In particolare le creazione e gestione dei **ticket** e degli *sprint*.
* **Slack**: Ho imparato ad usare molto bene tale *software*, anche se alcune *feature* che offre non le ho mai usate. Le cose principali che ho imparato sono la creazione ed uso di canali appositi, chiamate vocali rapide tra 2 o più persone, possibilità di configurare uno stato per comunicare la propria situazione lavorativa al *team*, utilizzo di messaggi rapidi o di metodi per citare le persone.
* **Jitsi Meet**: Ho imparato ad usare l'applicazione in modo basilare. In particolare so come avviare o creare un *meeting*, utilizzare video, audio, *chat* di testo e condividere lo schermo.

4.3 Valutazione personale

Mi ritengo veramente soddisfatto dallo svolgimento di questo *stage*, sia dal punto di vista professionale che personale. Il progetto che ho svolto è stato molto interessante e stimolante e questo mi ha confermato che ho fatto la scelta giusta sia per quanto riguarda l'ambito di *stage*, che come percorso di studi intrapreso.
Tale progetto e gli strumenti utilizzati sono attuali e sempre più diffusi, perciò questa esperienza mi tornerà molto utile. Per quanto mi riguarda, ad aver svolto lo *stage* da remoto mi son trovato molto bene per vari motivi, elencati di seguito:

- **Supporto costante**: Tramite piattaforma *Slack* e tramite servizio *Jitsi* mi è sempre stato fornito supporto durante l'orario lavorativo;
- **Comunicazione veloce**: Grazie alla piattaforma *Slack*, in caso di messaggio di testo rapido ad un collega, esso mi rispondeva in modo celere e chiaro;
- **Stato di lavoro**: Tramite una funzione di *Slack* mi era possibile segnalare il mio stato lavorativo come facevano gli altri, in modo da avere sempre un idea generale dell'andamento del lavoro e della situazione generale del *team*;
- **Confronto con i colleghi**: Anche se sembra banale, poter parlare e comunicare con i colleghi per un confronto sul lavoro, per 5 minuti di chiacchierata o una discussione dopo orario lavorativo, è stato fondamentale perché mi ha fatto sentire veramente come se fossi in un azienda a lavorare con altre persone con cui poter interagire o scambiare idee ed opinioni.
- **Risparmio e riuso del tempo**: Aver potuto svolgere lo *stage* da casa mi ha permesso di risparmiare minimo due ore al giorno che erano altrimenti destinate agli spostamenti. Queste ore che recuperavo le impiegavo per leggere documentazione, approfondire vari argomenti o tecnologie.

In linea generale i corsi di studio che ho seguito mi hanno dato le conoscenze, le metodologie, gli strumenti e il pensiero logico per affrontare diversi problemi, anche se di linguaggi o ambienti di sviluppo diversi e le capacità base per entrare nel mondo del lavoro. Inoltre, alcuni corsi svolti all'università mi hanno permesso di affrontare lo *stage* in maniera più efficiente e organizzata; in particolare il corso di ingegneria del *software* mi ha permesso di mettere in pratica il *framework Scrum* in maniera ottimale fin dall'inizio e grazie al corso di tecnologie *web* avevo già una base per quanto riguarda *HTML*, *CSS* e *JavaScript*.
Devo però dire che all'università non vengono insegnate le tecnologie o *framework* del momento ma è anche vero che l'informatica è un mondo tanto vasto e non sempre è possibile insegnare le ultime novità. Sarebbe utile e interessante poterle insegnare invitando sviluppatori competenti e capaci.
Ho intrapreso questo corso di studi appena finito le superiori, per le quali avevo scelto un diverso ambito di studio, perché era un settore che mi è sempre piaciuto e interessato. Alla fine di questi tre anni posso dire di essere abbastanza soddisfatto di come si è svolta la mia carriera scolastica e di com'è strutturato il percorso di studi. Avrei, però, preferito svolgere più progetti pratici come accompagnamento alla teoria e suggerirei a tutti gli studenti di informatica di seguire il corso opzionale (che dovrebbe essere reso obbligatorio) *"Tecnologie Open Source"*; questo poiché insegna diverse metodologie di approccio e strumenti oramai utilizzati spesso dalle aziende per lo sviluppo di *software* tramite metodologia *Scrum*.
Per concludere, ritengo che lo *stage* da svolgere, alla fine della propria carriera universitaria, sia un evento molto importante per lo studente per diversi motivi elencati di seguito:

- Testare le proprie capacità;
- Testare la propria maturità;
- Provare cosa significa lavorare nel settore che è stato scelto o a cui si è interessati;
- Decidere se continuare a lavorare o iniziare la magistrale;
- Capire se l'ambito in cui si è scelto di fare *stage* è realmente di proprio interesse.

Glossario

PWA Con il termine *PWA, Progressive Web App* si intendono tutte le applicazioni web che vengono sviluppate e caricate come normali pagine web, ma che si comportano in modo simile alle applicazioni native. 39

SPA Con il termine *SPA, Single Page Application* si intende un'applicazione web che interagisce con l'utente aggiornando dinamicamente la pagina corrente con i nuovi dati in arrivo dal web server. L'obiettivo è avere transizioni più veloci tra una pagina e l'altra e far sembrare il sito come un'app nativa. 39

Acronimi

PWA Progressive Web App. iii, 13

SPA Single Page Application. 3

Bibliografia

Siti web consultati

Airbnb Style Guide JavaScript. URL: https://github.com/airbnb/javascript.

Airbnb Style Guide React. URL: https://github.com/airbnb/javascript/tree/master/react.

Django. URL: https://www.djangoproject.com/.

Django Documentation. URL: https://docs.djangoproject.com/en/4.1/.

Docker. URL: https://www.docker.com/.

Docker Documentation. URL: https://docs.docker.com/.

Docker Start Guide OpenFoodFacts. URL: https://openfoodfacts.github.io/openfoodfacts-server/introduction/dev-environment-quick-start-guide/.

ESLint. URL: https://eslint.org/.

Facebook Developers. URL: https://developers.facebook.com/?locale=it_IT.

Facebook Developers Documentation - Login. URL: https://developers.facebook.com/docs/facebook-login/.

GitLab. URL: https://about.gitlab.com/.

Google Developers. URL: https://console.cloud.google.com/.

Jest. URL: https://jestjs.io/.

Jest Documentation React. URL: https://jestjs.io/docs/tutorial-react.

Jira. URL: https://www.atlassian.com/it/software/jira.

Jira Features. URL: https://www.atlassian.com/it/software/jira/features.

Jitsi. URL: https://meet.jit.si/.

Material-UI. URL: https://mui.com/material-ui/.

Modulo Scanner. URL: https://www.npmjs.com/package/@ericblade/quagga2.

Node.js. URL: https://nodejs.org/en/.

npm. URL: https://www.npmjs.com/.

OpenFoodFacts. URL: https://it.openfoodfacts.org/.

OpenFoodFacts API Documentation. URL: https://wiki.openfoodfacts.org/API.

React. URL: https://it.reactjs.org/.

React Crash Course 2021. URL: https://www.youtube.com/watch?v=w7ejDZ8SWv8.

React Documentation. URL: https://it.reactjs.org/docs/getting-started.html.

Redux. URL: https://redux.js.org/.

Redux Documentation. URL: https://redux.js.org/introduction/getting-started.

Redux Toolkit. URL: https://redux-toolkit.js.org/.

Redux Toolkit Documentation. URL: https://redux-toolkit.js.org/introduction/getting-started.

Redux Tutorial. URL: https://www.youtube.com/watch?v=93p3LxR9xfM.

RTK Query. URL: https://redux-toolkit.js.org/rtk-query/overview.

Slack. URL: https://slack.com/intl/it-it.

Slack Features. URL: https://slack.com/intl/it-it/features.

Testing Library. URL: https://testing-library.com/.

Testing Library Documentation React. URL: https://testing-library.com/docs/react-testing-library/intro.

Visual Studio Code. URL: https://code.visualstudio.com/.

Visual Studio Code Documentation. URL: https://code.visualstudio.com/docs.

Visual Studio Code Extension - VS Code Counter. URL: https://marketplace.visualstudio.com/items?itemName=uctakeoff.vscode-counter.

Visual Studio Code Marketplace. URL: https://marketplace.visualstudio.com/VSCode.