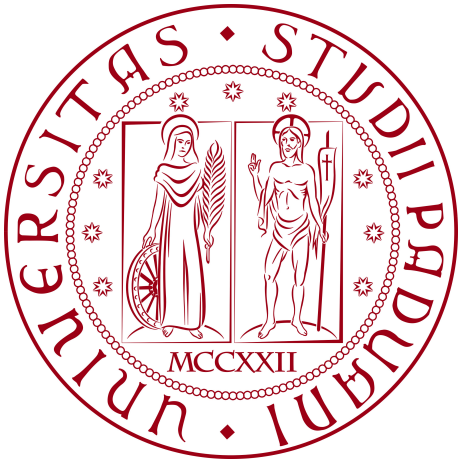
**Università degli Studi di Padova**

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica




# Sviluppo e gestione di un'app multipiattaforma in monorepo: Caso studio presso UNOX S.p.A.

*Tesi di Laurea Triennale*

*Relatore*

Prof. Da San Martino Giovanni

*Laureando*

Bobirica Andrei Cristian

Matricola 1224449

---

Anno Accademico 2023-2024

*"C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off."*

— Bjarne Stroustrup.

# Ringraziamenti

Desidero esprimere la mia gratitudine al professor Da San Martino Giovanni, mio relatore, per l'aiuto e il sostegno che mi ha dato durante la stesura dell'elaborato.

Un grazie di cuore ai miei genitori, che mi hanno sempre supportato e incoraggiato in ogni fase della mia vita. Senza di loro, nulla di tutto questo sarebbe stato possibile.

Un ringraziamento speciale va alle mie maestre delle elementari, Ornella e Luigina. Quando sono arrivato in Italia all'età di cinque anni, non conoscevo la lingua e mi trovavo di fronte a un nuovo mondo. Loro mi hanno accolto con affetto e pazienza, aiutandomi a integrarmi, a imparare l'italiano e a sentirmi parte di questa nuova realtà. Grazie al loro sostegno e alla loro dedizione, ho potuto costruire le basi per il mio percorso educativo e personale.

//todo amici

Padova, Luglio 2024 *Bobirica Andrei Cristian*

# Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di trecentoventi ore, dal laureando Bobirica Andrei Cristian presso l'azienda UNOX S.p.A.
L'obiettivo dello stage era la realizzazione di un'applicazione multipiattaforma che riuscisse a garantire compatibilità con IOS, Android e Web.
La sfida nella realizzazione di questa app è stata integrarla con un Design System$_G$ già esistente e in una monorepo$_G$ dove era già presente un'altra applicazione.
In questo documento si potrà esaminare l'analisi tecnica effettuata per l'applicazione, ma anche le problematiche riscontrate nella realizzazione e i spunti di riflessione che ne conseguono.

# Indice

# Elenco delle figure

# Elenco delle tabelle

# Elenco dei codici sorgenti

# Acronimi e abbreviazioni

**API** Application Program Interface. 14

**DDC** Data Driven Cooking. 4, 6

**RMA** Return Merchandise Authorization. 7

# Glossario

**Backend** //todo Backend Description. 6, 11, 16

**cross-platform** //todo crpl Description. 1

**DDC Service** //todo DDC SERVICE Description. 4, 11, 13, 16, 25, 28

**Design Pattern** //todo Design Pattern Description. 2

**Design System** //todo Design System Description. iv, 4, 6, 9, 10

**E2E** //todo e2e Description. 8

**ECN** //todo ECN Description. 22

**Firmware** //todo Firmware Description. 23

**Frontend** //todo Frontend Description. 11

**monorepo** //todo monorepo Description. iv, 1, 2, 4–6, 13

**Nav Bar** //todo Nav Bar Description. 23

**Product Code** //todo Product Code Description. 22

**DevOps** //todo DevOps Description. 11

**Repo** //todo Repository Description. 13

**RMA** //todo RMA Description. 7, 8

**Serviced Oven** //todo Serviced Oven Description. 19

**Tab Bar** //todo Tab Bar Description. 23

# Capitolo 1

# Introduzione

## 1.1 Convenzioni tipografiche

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*$_{G}$;
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

## 1.2 Organizzazione del testo

Questa tesi è strutturata per fornire una visione dettagliata e comprensibile dell'esperienza di stage presso UNOX S.p.A., focalizzandosi sullo sviluppo di un'applicazione cross-platform$_{G}$ e sullo studio di un ambiente di sviluppo in monorepo$_{G}$ . La suddivisione dei capitoli permette di seguire il percorso progettuale in modo chiaro e logico, dal contesto aziendale alle conclusioni finali. Di seguito è riportata l'organizzazione del testo:

**Introduzione:** Il capitolo introduttivo presenta una panoramica dell'azienda UNOX S.p.A., il contesto dello stage, e fornisce una descrizione dettagliata dell'organizzazione della tesi.

**Descrizione dello stage:** In questo capitolo viene descritto il progetto di stage, inclusi gli obiettivi tecnici e professionali, le attività svolte, la pianificazione dettagliata e l'analisi preventiva dei rischi.

**Tecnologie utilizzate:** Questo capitolo elenca e descrive le tecnologie impiegate durante lo sviluppo dell'applicazione.

**Analisi dei requisiti:** In questa sezione vengono descritti i casi d'uso, il monitoraggio dei requisiti e le tabelle che specificano le funzioni principali dell'applicazione.

**Progettazione e codifica:** In questo capitolo verranno esaminati i Design Pattern$_{G}$ adottati, esemplificati con parti significative di codice, insieme a descrizioni dettagliate di alcune funzionalità chiave sviluppate.

**Studio fattibilità app in monorepo:** Questo capitolo esplora la fattibilità dello sviluppo dell'applicazione in un ambiente monorepo$_{G}$ , descrivendo l'organizzazione iniziale, le problematiche rilevate, le soluzioni proposte e gli strumenti utilizzati per la gestione delle dipendenze.

**Conclusioni:** Il capitolo conclusivo presenta un consuntivo finale del lavoro svolto, una valutazione del raggiungimento degli obiettivi prefissati, le conoscenze acquisite durante lo stage, e una riflessione personale sull'esperienza complessiva.

**Bibliografia e Sitografia:** Infine, vengono elencate le fonti bibliografiche e sitografiche consultate per la redazione della tesi.

## 1.3 L'azienda

UNOX S.p.A.è un'azienda leader nel settore della produzione di forni professionali per la ristorazione, fondata nel 1990 e situata a Cadoneghe, in provincia di Padova, Italia. Riconosciuta a livello internazionale per la qualità, l'affidabilità e l'innovazione dei suoi prodotti, UNOX è all'avanguardia nella tecnologia di cottura intelligente, che integra connettività avanzata e automazione.

### Mission e Vision

La mission di UNOX S.p.A.è quella di contribuire al successo dei propri clienti offrendo soluzioni innovative e di alta qualità che migliorano le prestazioni e l'efficienza delle loro cucine. L'azienda si impegna a fornire prodotti che combinano tecnologia avanzata e facilità d'uso, garantendo al contempo sostenibilità ambientale e risparmio energetico. La vision di UNOX si concentra sull'essere il punto di riferimento per l'innovazione nel settore della ristorazione professionale. L'azienda punta a creare valore attraverso lo sviluppo continuo di tecnologie all'avanguardia e il miglioramento costante dei propri prodotti e servizi.

### Prodotti e Servizi

UNOX offre una vasta gamma di forni professionali, noti per la loro efficienza, versatilità e innovazione tecnologica. I prodotti principali includono:

- Forni a convezione: Forni che utilizzano l'aria calda per cuocere il cibo in modo uniforme e veloce.
- Forni a vapore: Forni che utilizzano il vapore per cucinare in modo sano e preservare le proprietà nutrizionali degli alimenti.
- Sistemi di cottura intelligenti: Tecnologie integrate che permettono il controllo preciso dei processi di cottura e l'automazione delle operazioni.
- Forni combinati: Forni che combinano cottura a vapore e a convezione, ideali per una varietà di preparazioni culinarie.

### Connettività e Innovazione

UNOX S.p.A.è pioniera nell'integrazione della connettività nei suoi prodotti, offrendo soluzioni che permettono il monitoraggio e il controllo remoto dei forni attraverso piattaforme digitali. L'azienda ha sviluppato il progetto Data Driven Cooking (DDC)$_G$ , una piattaforma che utilizza i dati raccolti dai forni per ottimizzare i processi di cottura e fornire suggerimenti personalizzati agli chef.

## 1.4 Lo stage

L'offerta di stage mi ha immediatamente intrigato per il suo focus su un progetto specifico vitale per l'azienda, piuttosto che un semplice esercizio accademico. Questo aspetto ha richiesto un impegno significativo e una collaborazione intensa con diversi team per portare a termine un progetto cruciale per l'azienda. Nel contesto aziendale esistono due app chiamate DDC$_G$ e DDC Service$_G$ .

- DDC ha come utilizzatori i proprietari dei forni che utilizzano questa app per le funzionalità connesse dei loro dispositivi.
- DDC Service ha come utilizzatori personale tecnico, personale responsabile di manutenzione dei forni e utenti addetti al *Service*.

Per rispondere alle esigenze aziendali, è stato necessario avviare lo sviluppo di una nuova app DDC Service$_G$ con una prospettiva moderna e che sia multi-piattaforma. Durante il mio stage presso UNOX S.p.A., ho lavorato principalmente sull'avvio dello sviluppo di questa app, il mio obiettivo principale è stato ristrutturare e sviluppare completamente da zero DDC Service precedentemente limitata alla piattaforma *Web*. Ho esteso le funzionalità di DDC Service per renderla compatibile con dispositivi Android, iOS e Web, integrando questa nuova versione nell'esistente monorepo$_G$ di DDC$_G$ . Questo approccio ha permesso di condividere il Design System$_G$ e sfruttare l'infrastruttura esistente per ottimizzare l'efficienza e la manutenibilità del codice. Durante il periodo di stage, ho collaborato attivamente con il team di sviluppo, design e progettazione per implementare le prime funzionalità richieste per l'applicazione, rispettando le linee guida e assicurando la compatibilità

su tutte le piattaforme *target*. Questa esperienza mi ha fornito competenze pratiche nello sviluppo software multi-piattaforma e una comprensione approfondita della progettazione scalabile e della gestione delle risorse tecniche in un ambiente monorepo.

# Capitolo 2

# Descrizione dello stage

## 2.1 Pianificazione

### 2.1.1 Attività

La seguente pianificazione delle attività è stata inizialmente delineata nel piano di lavoro. Tuttavia, durante lo stage, alcune attività sono state modificate sia per una maggiore comprensione emersa dall'analisi dei requisiti, sia per cambiamenti negli obiettivi da realizzare.

**Prima Settimana (40 ore)**

- Incontro con le persone coinvolte nel progetto per discutere i requisiti e le richieste relative al sistema da sviluppare.
- Verifica delle credenziali e degli strumenti di lavoro assegnati.
- Presa visione dell'infrastruttura esistente, in particolare della app DDC$_{G}$ , del suo Design System$_{G}$ e della monorepo$_{G}$ esistente.
- Formazione sulle tecnologie adottate.

**Seconda Settimana (40 ore)**

- Studio del software Backend$_{G}$ esistente con cui l'applicazione si integrerà.

- Avvio dello sviluppo dell'applicazione, definizione dell'architettura, dello *stack* di navigazione e implementazione della funzionalità di autenticazione.

**Terza Settimana (40 ore)**

- Continuazione dello sviluppo dell'architettura dell'applicazione, inclusi il *login* e lo stack di navigazione principale.

**Quarta Settimana (40 ore)**

- Sviluppo della funzionalità consultazione Prodotto, detta *Product Page*

**Quinta Settimana (40 ore)**

- Continuazione dello sviluppo delle funzionalità di consultazione Prodotto.

**Sesta Settimana (40 ore)**

- Implementazione nella *Product Page* delle funzionalità di visualizzazione manuali, ricambistica e *Tech and Docs*
- Sviluppa della funzionalità consultazione *Serviced Oven*

**Settima Settimana (40 ore)**

- Sviluppo della funzionalità di gestione del flusso Return Merchandise Authorization$_G$.

**Ottava Settimana - Conclusione (40 ore)**

- Continuazione dello sviluppo della funzionalità di gestione del flusso Return Merchandise Authorization (RMA).
- Test e ottimizzazione dell'applicazione con il personale aziendale.

| Durata in ore | Descrizione dell'attività |
|---|---|
| 40 | Inserimento in azienda |
| 24 | Studio Backend esistente |
| 56 | Sviluppo architettura applicazione |
| 80 | Sviluppo della funzionalità *Product Page* |
| 40 | Sviluppo della funzionalità *Serviced Oven Page* |
| 60 | Sviluppo funzionalità flusso *RMA* |
| 20 | Test e ottimizzazione della applicazione |
| **Totale ore** | **320** |

**Tabella 2.1:** Suddivisione delle ore di lavoro per le attività di progetto.

### 2.1.2 Obbiettivi

#### Obiettivi obbligatori

- **Architettura dell'applicazione:** Definizione dello scheletro e dell'architettura dell'applicazione, compresa la navigazione.
- **Autenticazione:** Implementazione della funzionalità di autenticazione (*SignIn, SignUp, Recover Password*)
- ***Product Page*:** Sviluppo della funzionalità consultazione Prodotto e delle funzionalità di visualizzazione manuali, ricambistica e *Tech and Docs*
- ***Serviced Oven*:** Sviluppa della funzionalità consultazione dei propri forni in *service* detti *Serviced Oven*
- **Test piattaforme:** Esecuzione di test sulla piattaforma web e mobile per garantire la massima portabilità del codice

#### Obiettivi desiderabili

- **RMA:** Sviluppo della funzionalità di gestione del flusso Return Merchandise Authorization$_G$.

#### Obiettivi facoltativi

- **Test E2E:** Creazione di test automatizzati e2e$_G$per verificare le varie componenti dell'applicazione, per massimizzare l'efficienza del processo di *testing*.

### 2.1.3 Vincoli

Durante lo sviluppo del progetto, sono stati identificati vari vincoli che hanno influenzato il contesto operativo e le decisioni progettuali. Questi vincoli hanno avuto un impatto significativo sulle scelte effettuate e sull'approccio adottato per la realizzazione dell'applicazione. I principali vincoli sono suddivisibili in categorie come vincoli aziendali, tecnologici, temporali e di design.

#### Vincoli tecnologici

Un vincolo importante riguardava l'adozione delle tecnologie già utilizzate da UNOX S.p.A.L'applicazione doveva essere sviluppata utilizzando strumenti e tecnologie in uso all'interno dell'azienda per assicurare l'integrazione e la coerenza con l'ecosistema tecnologico esistente.

#### Vincoli temporali

Un vincolo temporale significativo era la data di conclusione dello stage, fissata per il 7 giugno 2024. Questa scadenza ha imposto un termine rigido per il completamento del progetto, richiedendo una gestione attenta del tempo e delle risorse per rispettare il limite prestabilito.

#### Vincoli di design

I vincoli di design includevano il rispetto del Design System$_G$aziendale e delle specifiche grafiche fornite dall'azienda. L'applicazione doveva essere allineata al *Design System* esistente e rispettare le palette di colori e le linee guida visive stabilite dall'azienda.

## 2.2 Analisi preventiva dei rischi

Durante la fase di analisi iniziale, sono stati individuati alcuni possibili rischi che avrebbero potuto causare problemi nel corso del progetto. Per affrontarli, sono state elaborate delle possibili soluzioni.

**1. Inesperienza tecnologica**

**Descrizione:** Era previsto l'utilizzo di tecnologie mai utilizzate prima, il che poteva causare rallentamenti nello sviluppo dell'applicazione.

**Soluzione:** L'azienda ha programmato un periodo di circa una settimana dedicato allo studio autonomo delle tecnologie, utilizzando tutorial e risorse interne.

**2. Difficoltà nel soddisfare le esigenze di Design**

**Descrizione:** Inizialmente era previsto realizzare le funzionalità richieste senza dare peso al design e alla parte grafica dell'app. Tuttavia, è emersa la necessità di cooperare con il team di design per seguire le loro linee guida.

**Soluzione:** Si è utilizzato il Design System$_G$ già esistente, adattandolo dove necessario, e si è dato del tempo per imparare a utilizzare nuovi strumenti come Figma.

**3. Interpretazione dei requisiti**

**Descrizione:** I requisiti avrebbero potuto subire aggiornamenti in corso d'opera a causa della difficoltà d'individuare con facilità se un requisito fosse realizzabile o meno.

**Soluzione:** Sono stati pianificati meeting regolari con i team coinvolti per discutere e individuare soluzioni, semplificando la realizzazione dei requisiti proposti.

# Capitolo 3

# Tecnologie utilizzate

Questo capitolo esplora le tecnologie chiave adottate nel contesto dello sviluppo dell'applicazione DDC Service$_G$ . Vengono presentati i linguaggi di programmazione, i framework, gli strumenti di sviluppo, le piattaforme di collaborazione e gestione, oltre alle soluzioni *cloud* e DevOps$_G$ impiegate per supportare e ottimizzare il processo di sviluppo. Ogni tecnologia è discussa nel contesto del suo ruolo nell'ecosistema di sviluppo dell'applicazione, evidenziando come contribuisca alla scalabilità, alla manutenibilità e alla coerenza del codice, nonché al miglioramento complessivo dell'efficienza operativa dell'azienda.

## 3.1 Linguaggi di programmazione

- **TypeScript**: è un linguaggio di programmazione open-source sviluppato da Microsoft. TypeScript è un superset di JavaScript che aggiunge la tipizzazione statica opzionale e altre funzionalità moderne, rendendo il codice più robusto e manutenibile. Questo linguaggio è stato utilizzato per lo sviluppo dell'applicazione *DDC Service*, sia per la parte Frontend$_G$ che per l'integrazione con i servizi Backend$_G$ , garantendo una maggiore affidabilità e scalabilità del codice.

## 3.2 Framework in uso

- **Expo**: è un *framework open-source* per la creazione di applicazioni *React Native*. Facilita lo sviluppo di applicazioni mobili fornendo strumenti e librerie preconfigurate. Expo è stato utilizzato per lo sviluppo delle applicazioni *Android* e *iOS*, permettendo di scrivere il codice una sola volta e distribuirlo su entrambe le piattaforme in modo efficiente.

- **Next.js**: un *framework* di sviluppo *React* per la creazione di applicazioni *Web*. Supporta il *rendering* lato *server* e la generazione di siti statici, migliorando così le prestazioni e l'ottimizzazione per i motori di ricerca (*SEO*).

- **React**: è una libreria *JavaScript* per la costruzione di interfacce utente sviluppata da *Facebook*. React si distingue per la sua architettura basata su componenti e l'uso del *virtual DOM*, che rendono lo sviluppo di interfacce utente reattive ed efficienti. In particolare, React è stato utilizzato come base per le applicazioni, consentendo la creazione di componenti riutilizzabili che migliorano la coerenza e la manutenibilità del codice.

- **React Native**: è un framework open-source per lo sviluppo di applicazioni mobili creato da *Facebook*. React Native permette di utilizzare React e *TypeScript* per costruire applicazioni native per *iOS* e *Android*. Unox utilizza React Native per sviluppare applicazioni mobili, permettendo al team di scrivere il codice una sola volta e distribuirlo su entrambe le piattaforme, semplificando e ottimizzando gli sforzi di sviluppo.

- **NodeJS**: è una piattaforma di *runtime open-source* basata su *JavaScript V8* di *Chrome*, progettata per costruire applicazioni di rete veloci e scalabili. Unox utilizza NodeJS per l'esecuzione del codice *TypeScript* e come gestore di pacchetti. Facilita le operazioni lato *server*, consentendo una gestione efficiente di compiti come il *rendering* del server e lo sviluppo di *API*, migliorando le prestazioni e la scalabilità dell'applicazione.

## 3.3 Tecnologie per monorepo

- **NPM Workspaces**: una funzionalità di NPM che consente di gestire più pacchetti all'interno di un unico Repository$_G$ . NPM Workspaces è stato utilizzato per organizzare i vari pacchetti del progetto DDC Service$_G$ , semplificando la gestione delle dipendenze e migliorando l'efficienza dello sviluppo.
- **NX**: un set di strumenti per la gestione di monorepo$_G$ che facilita lo sviluppo, il test e la manutenzione di applicazioni e librerie su larga scala. Nel progetto *DDC Service*, NX è stato implementato nella parte finale per la gestione del *monorepo*, per organizzare e gestire le dipendenze del codice e migliorando la modularità e la coerenza del progetto.

## 3.4 Librerie utilizzate

- **Solito**: una libreria che permette di condividere il codice tra applicazioni *Next* ed *Expo*, riducendo la duplicazione del codice e semplificando la manutenzione. Nel contesto del progetto DDC Service$_G$ , Solito è stato impiegato per ottimizzare la condivisione del codice tra le piattaforme web e mobile, implementando una gestione del routing comune tra le pagine. Ciò ha permesso di mantenere una struttura di navigazione coerente e una logica di gestione dei percorsi uniforme, migliorando l'esperienza dell'utente e semplificando lo sviluppo e la manutenzione dell'applicazione su entrambe le piattaforme.
- **Moti**: una libreria di animazioni per *React Native*$_G$che facilita la creazione di animazioni complesse e fluide. È stata utilizzata nel progetto DDC Service$_G$ per migliorare l'interazione dell'utente e l'aspetto visivo delle applicazioni mobili.
- **Dripsy**: Una libreria per la gestione del stile di UI per React Native e Web. Dripsy permette di definire uno stile una sola volta e eseguirlo ovunque, supportando la creazione di interfacce responsive che si adattano automaticamente a diverse dimensioni di schermo. È compatibile con Expo, Vanilla React Native e Next.js, offrendo un supporto completo per TypeScript e facilitando

l'implementazione di temi personalizzati e varianti di tema. Con una semplice Application Program Interface (API)$_G$ , è possibile definire stili tematici e responsivi in una sola riga di codice. Supporta anche modalità scura e personalizzazione dei colori.

## 3.5 Strumenti di sviluppo

- **Visual Studio Code**: un editor di codice sorgente sviluppato da *Microsoft*, altamente estensibile e utilizzato per una varietà di linguaggi di programmazione.
- **Xcode**: un ambiente di sviluppo integrato (*IDE*) di *Apple* per *macOS*, utilizzato per sviluppare software per *iOS*, *macOS*, *watchOS* e *tvOS*.
- **Android Studio**: un *IDE* ufficiale per lo sviluppo di applicazioni *Android*, fornito da *Google*. Viene utilizzato per scrivere, eseguire il debug e testare le applicazioni *Android*.
- **Prettier**: uno strumento di formattazione del codice che aiuta a mantenere uno stile di codice coerente in tutti i progetti.
- **Cocoapods**: un gestore di dipendenze per *Swift* e *Objective-C Cocoa projects*. Viene utilizzato per integrare librerie di terze parti nei progetti *iOS*.

## Piattaforme di collaborazione e gestione

- **Git**: viene utilizzato come sistema di controllo delle versioni per il tracciamento delle modifiche al codice sorgente.
- **Microsoft Teams**: adottato come strumento di comunicazione e collaborazione in tempo reale all'interno dell'azienda, facilitando le discussioni, le videochiamate e la condivisione di documenti. Viene utilizzato anche per la calendarizzazione di eventi e meeting.

## Piattaforme cloud e DevOps

- **Microsoft Azure**: una piattaforma cloud utilizzata per l'hosting di applicazioni, servizi e dati aziendali. Viene utilizzato da Unox per l'hosting di alcuni dei servizi principali. Dalla suite di Azure, viene utilizzato anche *Azure DevOps* per la gestione delle attività di sviluppo software, tra cui la gestione dei repository Git, delle build e delle attività.

- **AWS**: *Amazon Web Services* (*AWS*) è un altro servizio cloud utilizzato per le risorse di calcolo, archiviazione e servizi di rete. Alcuni dei servizi secondari di Unox sono ospitati su AWS.

- **Amplify**: è una piattaforma di sviluppo di applicazioni cloud che facilita l'integraazione di funzionalità come autenticazione, *API*, storage e altro ancora. In questo progetto, Amplify è stato utilizzato per scaricare automaticamente le chiavi di accesso, migliorando la sicurezza e semplificando la gestione delle credenziali.

## Strumenti di design

- **Figma**: uno strumento di design collaborativo utilizzato per la progettazione delle interfacce utente. Facilita la collaborazione tra designer e sviluppatori e permette di creare e condividere facilmente prototipi e design.

# Capitolo 4

# Analisi dei requisiti

## 4.1 Caratteristiche degli utenti

Per esaminare la *user base* dell'app DDC Service$_G$ bisogna tenere in considerazione che tutti i dati di cui l'applicazione fa uso saranno reperibili dal Backend$_G$ associato all'applicazione. Le basi di dati con cui il *backend* andrà a comunicare saranno popolate da piattaforme esterne ignote alla app in questione. Per tanto il sistema di autenticazione non dovrà tenere in considerazione di tipologie di utenti speciali come *Admin* o *SysAdmin*.

Questi utenti possono far parte del personale Unox oppure possono essere professionisti con cui Unox collabora. Sono persone che richiedono di accedere alla app sia da *Desktop* tramite *web-app*, mentre sono in ufficio per esempio, che da app sul proprio *smartphone* mentre sono in mobilità. La user base della app è composta da personale tecnico, personale responsabile alla vendita e utenti addetti al *Service*. Non esiste però la necessità lato autenticazione di rendere il processo di registrazione o di *login* diverso in base alla funzione del utente o in base alla sua appartenenza. In linea generica tutti gli utenti registrati hanno gli stessi permessi. In futuro l'applicazione necessiterà di un sistema per rendere un utente registrato verificato o meno, però per questo progetto questa richiesta non è una necessità.

## 4.2 Vincoli generali

Date le premesse sulle caratteristiche degli utenti della applicazione, definiremo quindi solo due categorie di utenti: Utente Autenticato, Utente Non Autenticato. Ho individuato i seguenti attori:

**Utente Non Autenticato** L'Utente Non Autenticato è una tipologia di utente che o non si è ancora registrato e quindi non possiede delle credenziali oppure possiede delle credenziali ma deve ancora far il processo di *login* detto *signin*.

**Utente Autenticato** L'Utente Autenticato è una tipologia di utente che possiede delle credenziali valide e che ha completato con successo il processo di *login* detto *signin*.

## 4.3 Casi d'uso

I seguenti casi d'uso sono state formulati solo per le funzionalità che sono state realmente realizzate in quanto le funzionalità opzionali per le quali non si ha avuto tempo non sono state tenute in considerazione per l'analisi dei requisiti.

**UC0: Autenticazione**

**Attori Principali:** Utente Non Autenticato
**Precondizioni:** Un Utente Non Autenticato vuole accedere alle funzionalità della app
**Descrizione:** Questo scenario descrive un utente che vuole accedere alla funzionalità della app ma che non ne ha i permessi, per tanto deve fare la registrazione (*singup*) o il *login* (*singin*)
**Postcondizioni:** l'utente accede al processo di registrazione o di *login*

**UC0.1: Registrazione**

**Attori Principali:** Utente Non Autenticato
**Precondizioni:** Un Utente Non Autenticato NON possiede delle credenziali per

poter accedere alla app

**Descrizione:** Questo scenario descrive un utente che non possiede credenziali di autenticazione e che non riesce ad accedere alle funzionalità della app, per tanto intraprende il processo di registrazione con cui alla fine riuscirà ad avere delle credenziali valide.

**Postcondizioni:** L'utente ora ha delle credenziali valide con cui poter effettuare il *login* e diventare un Utente Autenticato

**Scenario Alternativo:** Se l'utente non fornisce con correttezza tutti i dati necessari alla registrazione, visualizza un messaggio di errore.

#### UC0.2: Login

**Attori Principali:** Utente Non Autenticato

**Precondizioni:** Un Utente non Autenticato che possiede delle credenziali valide per l'autenticazione.

**Descrizione:** Tramite questo scenario l'utente fornisce le sue credenziali per l'autenticazione e completa il processo di *login*.

**Postcondizioni:** L'utente è diventato un Utente Autenticato

**Scenario Alternativo:** Se l'utente fornisce delle credenziali sbagliate, visualizza un messaggio di errore.

#### UC0.3: Recover Password

**Attori Principali:** Utente Non Autenticato

**Precondizioni:** Un Utente non Autenticato che possiede delle credenziali NON valide per l'autenticazione.

**Descrizione:** Tramite questo scenario l'utente fornisce le sue credenziali in maniera parziale e tramite un processo riesce a ripristinarle per poter avere delle credenziali valide.

**Postcondizioni:** L'utente ha delle credenziali valide.

#### UC1: Visualizzazione Prodotto

**Attori Principali:** Utente Autenticato

**Precondizioni:** Un utente ha completato con successo il processo di autenticazione.

**Descrizione:** L'utente visualizza le informazioni di un determinato prodotto incluse le sue parti commerciali e i dettagli tecnici a lui associati.

**Postcondizioni:** L'utente ha visualizzato le informazioni collegate al prodotto richiesto.

**Scenario Alternativo:** Se il prodotto richiesto non è esistente, viene visualizzato un messaggio di errore.

#### UC2: Visualizzazione Serviced Oven

**Attori Principali:** Utente Autenticato

**Precondizioni:** Un utente ha completato con successo il processo di autenticazione e ha tra i suoi forni alcuni dispositivi categorizzati come *Serviced Oven*, cioè forni a cui l'utente presta assistenza o manutenzione.

**Descrizione:** L'utente visualizza le informazioni di un determinato Serviced Oven$_{G}$ di cui ne ha i permessi.

**Postcondizioni:** L'utente ha visualizzato le informazioni collegate al *Serviced Oven* richiesto.

**Scenario Alternativo:** Se il prodotto richiesto non è esistente, viene visualizzato un messaggio di errore.

## 4.4 Tracciamento dei requisiti

Da un'attenta analisi dei requisiti e degli use case effettuata sul progetto è stata stilata la tabella che traccia i requisiti in rapporto agli use case.
Sono stati individuati diversi tipi di requisiti e si è quindi fatto utilizzo di un codice identificativo per distinguerli.
Il codice dei requisiti, dove ogni requisito è identificato con il carattere **R**, è così strutturato:

**F**: Funzionale.

**Q**: Qualitativo.

**V**: Di vincolo.

**N**: Obbligatorio (necessario).

**D**: Desiderabile.

Nelle tabelle 4.1, 4.2 e 4.3 sono riassunti i requisiti e il loro tracciamento con gli use case delineati in fase di analisi.

## 4.5 Tabelle dei requisiti

| Requisito | Descrizione | Use Case |
|---|---|---|
| RFN-1 | La funzionalità di autenticazione deve essere accessibile solo a Utenti Non Autenticati | UC0 |
| Continua nella prossima pagina... | | |

**Tabella 4.1 – Continuo della tabella**

| Requisito | Descrizione | Use Case |
|---|---|---|
| RFN-2 | L'utente deve essere in grado di accedere alla pagina registrazione detta *signup*. Questa pagina deve dare la possibilità di inserire i campi *Name, Surname, Email, Password, Confirm Password, Company Name, Business Type, Address, Phone Number, Contry, Language* e scelte *checkbox* GDPR e consenso Marketing | UC0.1 |
| RFN-2.1 | Nel *form* per la registrazione devono esserci filtri per il controllo dei dati immessi come *input* | UC0.1 |
| RFN-2.2 | Una volta immessi i dati nel *form* della pagina di registrazione questa deve mostrare un messaggio che indica al utente di attivare l'account con la conferma della *email* | UC0.1 |
| RFD-2.2.1 | Una volta confermata la email il processo di registrazione è completato, è richiesto che la app faccia automaticamente il *login* con l'account appena creato, quindi ad ogni registrazione avviene il *login* automatico | UC0.1 |
| RFN-3 | Nella pagina di *login* l'utente presenta le sue credenziali nel apposito *form*, cioè *email e password* per poter diventare un Utente Autenticato | UC0.2 |
| RFN-3.1 | Nella pagina di *login* nel caso in cui l'utente abbia inserito le credenziali non valide deve apparire un messaggio di errore | UC0.2 |

Continua nella prossima pagina...

**Tabella 4.1 – Continuo della tabella**

| Requisito | Descrizione | Use Case |
|---|---|---|
| RFN-4 | Deve esistere una pagina collegata alla pagina di *login* che permetta di fare il processo di ripristino *password*, detto *Recover Password*, con questo processo l'utente inserisce la propria *email* e dopo un processo di verifica gli viene confermato il ripristino della *password* | UC0.3 |
| RFN-5 | La funzionalità di Prodotti definisce la *feature* di visualizzazione di prodotti di Unox, sono tutti i prodotti esistenti e sono oggetti astratti da catalogo, non macchine reali, è richiesto solo la visualizzazione di un singolo prodotto nella sua *Product Page* | UC1 |
| RFN-5.1 | Un prodotto deve essere identificato dal proprio Product Code$_G$ , se lo si identifica solo con il *product code* si intende che si vuole visualizzare il prodotto con l'ultimo ECN$_G$ | UC1 |
| RFN-5.2 | Nel caso in cui si identifichi un prodotto tramite il suo seriale, si intende che si vuole una istanza di un prodotto, quindi un prodotto fisico realmente esistente. Questa metodologia permette di identificare un prodotto con un *ecng* specifico. | UC1 |
| Continua nella prossima pagina... | | |

**Tabella 4.1 – Continuo della tabella**

| Requisito | Descrizione | Use Case |
|---|---|---|
| RFN-5.3 | Nella pagina visualizzazione prodotto deve essere possibile visualizzare informazioni riguardanti le specifiche del prodotto, i documenti al prodotto collegati ed i *files* scaricabili come Firmware$_G$ o modelli 3D. | UC1 |
| RFN-6 | Deve essere implementata la *feature* Service, in particolare *Serviced Oven*, è richiesto che l'utente visualizzi un singolo suo prodotto identificato con un seriale | UC2 |
| RFN-6.1 | Nella pagina di *Serviced Oven* si devono poter visualizzare le informazioni riguardanti la connessione, si deve poter avere riferimenti alla pagina Prodotto del forno, e poter visualizzare e modificare le informazioni aggiuntive legate al prodotto, in particolare informazioni legate al cliente presso cui il forno è installato. | UC2 |
| RFN-7 | Per tutte le piattaforme deve essere realizzato un sistema di navigazione che permetta di cambiare le sezioni della app | UC0, UC1, UC2 |
| RFN-7.1 | Per le piattaforme mobile è prevista la realizzazione di una Tab Bar$_G$ invece per la piattaforma *web* è prevista la realizzazione di una Nav Bar$_G$ | UC0, UC1, UC2 |

**Tabella 4.1:** Tabella del tracciamento dei requisiti funzionali.

| Requisito | Descrizione | Use Case |
|---|---|---|
| RQN-1 | La navigazione della app deve essere funzionale sia da *mobile* che da *web*, permettendo di indentificare con facilità il corretto flusso di funzionamento della app indipendentemente dalla piattaforma in uso. | - |
| RQN-2 | Il codice prodotto deve essere scalabile e manutenibile, rispettando i design pattern attualmente già utilizzati nel ecosistema Unox. | - |
| RQN-3 | Tutte le funzionalità realizzate devono essere testate su tutte le piattaforme di destinazione garantendo il loro funzionaento. | - |

**Tabella 4.2:** Tabella del tracciamento dei requisiti qualitativi.

| Requisito | Descrizione | Use Case |
|---|---|---|
| RVN-1 | Le pagine e i componenti della applicazione devono essere realizzati tenendo conto delle indicazioni date dal team di Design, rispettando un *Design System* esistente e le convenzioni date per stili grafici compresi UI/UX | - |
| RVN-2 | Le feature realizzate per la applicazione devono essere compatibili con tutte le piattaforme di destinazione. | - |

**Tabella 4.3:** Tabella del tracciamento dei requisiti di vincolo.

# Capitolo 5

# Progettazione e Codifica

In questo capitolo verranno descritti i processi di progettazione e codifica utilizzati nello sviluppo dell'applicazione DDC Service$_G$ . Si esamineranno l'architettura dell'applicazione, le tecnologie utilizzate per il frontend e il backend, i protocolli di comunicazione, l'autenticazione e l'architettura a componenti.

## 5.1 Architettura dell'Applicazione

### 5.1.1 Backend e Frontend

In questa sezione, verranno descritti il backend e il frontend dell'applicazione DDC Service$_G$ . Si analizzeranno le differenze tra le due parti, le tecnologie utilizzate e le responsabilità di ciascuna componente.

#### Responsabilità del Backend

Il backend dell'applicazione è scritto in Node.js utilizzando TypeScript, una scelta che combina la flessibilità e la velocità di Node.js con la robustezza della tipizzazione statica di TypeScript. Il backend ha diverse responsabilità cruciali:

- **Gestione dei Dati**: Il backend gestisce l'archiviazione, il recupero e la manipolazione dei dati. Utilizza diversi Database per memorizzare informazioni strutturate. Inoltre, il backend si interfaccia con servizi di cloud storage come Amazon S3 di AWS per memorizzare file e oggetti di grandi dimensioni, garantendo alta disponibilità e durabilità.

- **Autenticazione**: L'autenticazione degli utenti è un'altra responsabilità fondamentale del backend. L'autenticazione gestita dal backend offre un livello di sicurezza maggiore per diverse ragioni:

  - **Controllo Centralizzato:** Il backend funge da punto centrale per la verifica delle credenziali degli utenti, garantendo che tutti i tentativi di accesso siano gestiti in modo uniforme e sicuro. Questo riduce il rischio di inconsistenze e vulnerabilità che potrebbero sorgere se l'autenticazione fosse distribuita su più client.
  - **Protezione dei Segreti:** Nel backend, è possibile mantenere i segreti e le chiavi di crittografia al sicuro, lontano dai dispositivi degli utenti dove potrebbero essere più facilmente compromessi. Ciò include la gestione sicura delle password, che vengono memorizzate come hash sicuri, e la generazione di token di accesso.
  - **Validazione e Scadenza dei Token:** Il backend può gestire la generazione, la validazione e la scadenza dei token di accesso JWT, assicurando che solo i token validi e non scaduti possano essere utilizzati per accedere alle risorse protette. Questo meccanismo previene l'accesso non autorizzato e garantisce che gli utenti debbano autenticarsi periodicamente.
  - **Log e Monitoraggio:** Il backend può registrare tutte le attività di autenticazione, facilitando il monitoraggio e l'analisi dei tentativi di accesso. Questo aiuta a identificare e rispondere rapidamente a potenziali attacchi, come tentativi di forza bruta o accessi sospetti.

  Questi vantaggi fanno sì che l'autenticazione gestita dal backend sia un componente cruciale per la sicurezza complessiva dell'applicazione, proteggendo le informazioni sensibili degli utenti e garantendo l'integrità e la riservatezza dei dati.

- **Logica di Business**: Il backend contiene la logica di business dell'applicazione, questa logica è separata in moduli e servizi, rendendo il codice più organizzato e manutenibile. I servizi nel backend sono responsabili di eseguire

operazioni come la validazione dei dati, l'elaborazione delle transazioni, e la comunicazione con servizi esterni.

- **API**: Il backend espone API REST e GraphQL che permettono al frontend di interagire con i dati e le funzionalità dell'applicazione. Le API REST sono utilizzate per operazioni standard CRUD (Create, Read, Update, Delete) e seguono una struttura basata su risorse. GraphQL, d'altra parte, offre una maggiore flessibilità permettendo al frontend di specificare esattamente quali dati necessita, riducendo così il sovraccarico di rete e migliorando l'efficienza.

#### Responsabilità del Frontend

Il frontend dell'applicazione è la parte visibile e interattiva che gli utenti utilizzano. È sviluppato utilizzando Expo e Next.js, con Expo destinato alle piattaforme mobili e Next.js alle piattaforme web. Il linguaggio di programmazione utilizzato per il frontend è TypeScript. Le principali responsabilità del frontend includono:

- **Interfaccia Utente (UI)**: Il frontend è responsabile della presentazione visiva e dell'interazione dell'utente con l'applicazione. Utilizza componenti React per costruire un'interfaccia modulare e riutilizzabile. Ogni componente rappresenta una parte della UI, come bottoni, moduli di input, e layout di pagina.

- **Gestione dello Stato**: La gestione dello stato è un aspetto critico del frontend. Utilizzando Redux, una libreria per la gestione dello stato, l'applicazione mantiene uno stato globale che può essere condiviso tra vari componenti.

- **Interazione con le API**: Il frontend interagisce con il backend attraverso le API REST e GraphQL. Utilizzando librerie come Axios o Fetch, il frontend invia richieste HTTP al backend per recuperare, creare, aggiornare o eliminare dati. Le risposte del backend sono quindi utilizzate per aggiornare l'interfaccia utente, rendendo i dati dinamicamente disponibili agli utenti.

- **Navigazione e Routing**: Il frontend gestisce la navigazione tra le diverse pagine dell'applicazione, permettendo agli utenti di spostarsi fluidamente all'interno dell'interfaccia. Questo include la definizione di percorsi e la

gestione delle transizioni tra le pagine, assicurando una suddivisione logica dell'applicazione in sezioni distinte e accessibili.

### Differenze tra Backend e Frontend

Sebbene il backend e il frontend siano strettamente collegati, essi hanno ruoli e responsabilità distinti:

- **Tecnologie**: Il backend è sviluppato in Node.js, focalizzandosi sulla gestione dei dati, la logica di business e le API. Il frontend, invece, è sviluppato utilizzando Expo per le piattaforme mobili e Next.js per le piattaforme web, concentrandosi sull'interfaccia utente e l'esperienza dell'utente.
- **Responsabilità**:
  - **Backend**: Gestisce la logica di business, l'autenticazione, la gestione dei dati e la comunicazione con i servizi esterni.
  - **Frontend**: Si occupa della presentazione visiva, l'interazione dell'utente, la gestione dello stato e la navigazione.
- **Interazione**: Il backend e il frontend comunicano tramite API, dove il backend fornisce i dati e le funzionalità necessarie, mentre il frontend li utilizza per creare un'interfaccia utente interattiva e dinamica.

### Flusso di Lavoro Complessivo

Il flusso di lavoro tipico dell'applicazione prevede che l'utente interagisca con il frontend, che a sua volta invia richieste al backend. Il backend elabora queste richieste, esegue la logica di business necessaria, interagisce con i database e i servizi esterni, e infine restituisce una risposta al frontend. Il frontend quindi aggiorna l'interfaccia utente in base alla risposta ricevuta, garantendo un'esperienza utente fluida e interattiva. In sintesi, il backend e il frontend dell'applicazione DDC Service$_G$ lavorano insieme per fornire un servizio completo ed efficiente, ognuno con ruoli e responsabilità specifici ma complementari. Questa architettura modulare e separata permette di sviluppare, mantenere e scalare l'applicazione in modo efficace.

### 5.1.2 Struttura delle Applicazioni

La struttura complessiva delle applicazioni è organizzata in modo da mantenere il codice manutenibile e modulare. Questo approccio consente di gestire facilmente la complessità del progetto, facilitando lo sviluppo, la collaborazione e la scalabilità. La disposizione delle directory e dei file è progettata per riflettere la suddivisione logica delle funzionalità, garantendo che ogni componente del sistema sia isolato e riutilizzabile.

#### Monorepo

L'utilizzo di una monorepo è una strategia che permette di gestire tutto il codice del progetto in un unico repository. Questo approccio offre numerosi vantaggi, tra cui:

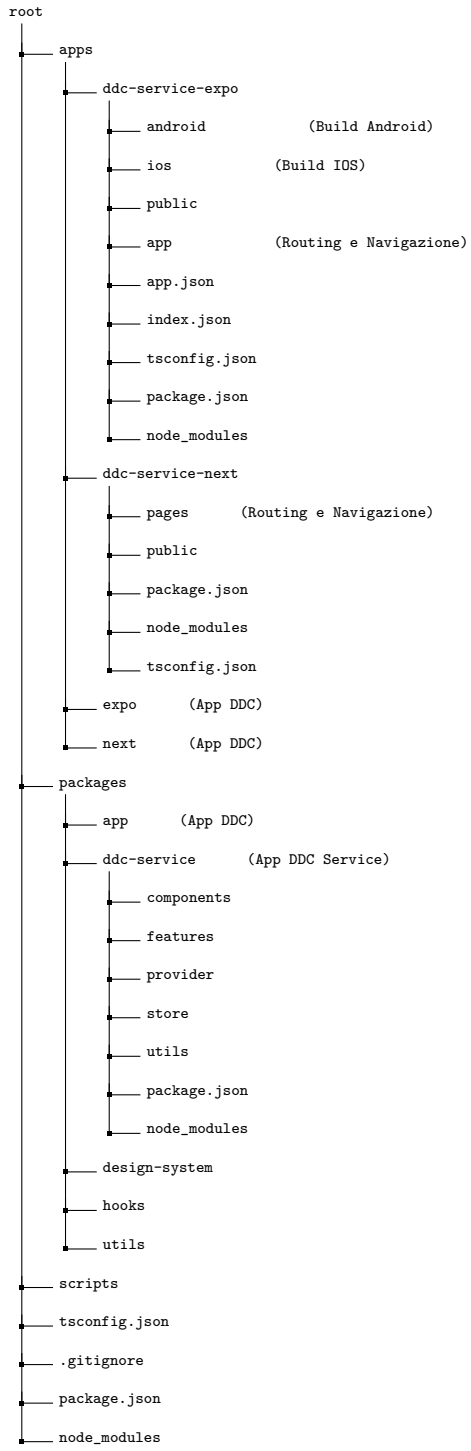
- **Condivisione del Codice**: Facilita la condivisione di moduli e librerie comuni tra diverse parti dell'applicazione, evitando duplicazioni e incoerenze. Questo approccio è particolarmente utile per il progetto DDC Service, poiché consente di riutilizzare parti comuni tra l'applicazione DDC e DDC Service, come il Design System, garantendo una coerenza visiva e funzionale tra le due applicazioni.
- **Gestione delle Dipendenze**: Permette di avere una gestione centralizzata delle dipendenze, semplificando l'aggiornamento e la sincronizzazione delle librerie utilizzate. Questo è essenziale per mantenere un ambiente di sviluppo stabile e aggiornato, riducendo i conflitti e le incompatibilità tra le diverse parti del progetto.
- **Collaborazione**: Migliora la collaborazione tra i team, offrendo una visione unificata del progetto e facilitando la revisione del codice. I team possono lavorare contemporaneamente su diverse parti dell'applicazione, beneficiando della trasparenza e della coerenza del codice condiviso.
- **Strumenti di Build e CI/CD**: Consente di configurare strumenti di build e pipeline CI/CD in modo centralizzato, ottimizzando i processi di integrazione

e distribuzione continua. Questo permette di automatizzare i test, il rilascio e il deploy delle applicazioni DDC e DDC Service, assicurando che le nuove funzionalità e le correzioni di bug siano implementate rapidamente e senza intoppi.

Per questo progetto si è deciso di utilizzare la tecnologia monorepo presente con *npm workspaces*. Questo primo approccio è stato scelto in quanto è stata la configurazione già addottata per DDC, tuttavia, come verrà analizzato nel capitolo successivo, ci sono soluzioni migliore.

#### 5.1.2.1 Struttura delle directory

Di seguito descritta la struttura complessiva delle applicazioni, spiegando come sono organizzate le directory e i file al fine di mantenere il codice manutenibile e modulare. La struttura del progetto è organizzata come segue:



#### 5.1.2.2 Visione Generale

La struttura del progetto è organizzata per supportare lo sviluppo e la gestione di quattro applicazioni principali: due applicazioni DDC (una per Expo e una per Next) e due applicazioni DDC Service (una per Expo e una per Next). La suddivisione in diverse directory facilita la condivisione del codice, delle risorse e delle configurazioni tra le applicazioni, migliorando la manutenibilità e la modularità del progetto.

Le directory principali sono suddivise come segue:

- **apps**: Contiene le applicazioni specifiche per DDC e DDC Service. Ogni applicazione ha una directory separata per le versioni Expo e Next. Le applicazioni contenute qui sono la struttura portante rispettivamente di Next ed Expo, e il codice contenuto dentro queste cartelle si occupa solamente di navigazione e routing.
- **packages**: Contiene i pacchetti condivisi tra le applicazioni, inclusi componenti, utils, hook e il design system. il package utils contiene funzioni utili per tutte le applicazioni che possono essere riutilizzate in ambito sviluppo multipiattaforma.
- **scripts**: Contiene script per automatizzare varie attività di sviluppo e di deployment.
- **Configurazioni**: File di configurazione globali, come `tsconfig.json`, `package.json` e `.gitignore`.

#### 5.1.2.3 Descrizione delle Directory

- **apps** La directory `apps` contiene le applicazioni specifiche per DDC e DDC Service, organizzate come segue:
  - **ddc-service-expo**: Contiene la versione Expo dell'applicazione DDC Service. Include le directory per Android e iOS, oltre a file di configurazione specifici per Expo.

  - **ddc-service-next**: Contiene la versione Next dell'applicazione DDC Service. Include le pagine dell'applicazione e i file di configurazione specifici per Next.js.
  - **expo (DDC)**: Contiene la versione Expo dell'applicazione DDC.
  - **next (DDC)**: Contiene la versione Next dell'applicazione DDC.

- **packages** La directory `packages` contiene i pacchetti condivisi tra le applicazioni, organizzati come segue:
  - **app**: Pacchetto contenente le applicazioni e le pagine condivise tra le versioni Expo e Next dell'applicazione DDC.
  - **ddc-service**: Pacchetto contenente componenti, funzionalità, provider, store e utilità specifici per l'applicazione DDC Service.
  - **design-system**: Pacchetto contenente il design system condiviso tra le applicazioni, incluso il tema dell'applicazione e i componenti di base.
  - **hooks**: Pacchetto contenente hook condivisi tra le applicazioni.
  - **utils**: Pacchetto contenente utilità condivise tra le applicazioni.

- **scripts** La directory `scripts` contiene script per automatizzare varie attività di sviluppo, come l'installazione delle dipendenze, la costruzione e il deployment delle applicazioni.

- **Configurazioni** Nella root del progetto si trovano vari file di configurazione globali, tra cui:
  - `postinstall.json`: Script eseguiti dopo l'installazione delle dipendenze.
  - `tsconfig.json`: Configurazione TypeScript condivisa tra tutte le applicazioni e i pacchetti.
  - `.gitignore`: File che specifica i file e le directory da ignorare nel controllo di versione.
  - `package.json`: Gestione delle dipendenze e script di progetto.
  - `node_modules`: Directory generata automaticamente che contiene tutte le dipendenze installate.

#### 5.1.2.4 Gestione delle Dipendenze

All'interno del file *package.json* situato nella root del progetto, sono configurati diversi parametri che definiscono la struttura della monorepo. In particolare, c'è una suddivisione tra due principali categorie di *workspaces*: *apps* e *packages*. Per *workspace* ci si riferisce alle cartelle contenute all'interno delle directory *apps* e *packages*. Ad esempio, *ddc-service-expo* è considerato un workspace. Questa suddivisione permette una gestione organizzata e modulare del codice, facilitando la manutenzione e la scalabilità del progetto. Per quanto riguarda la gestione delle dipendenze, quando si utilizza un *package* esterno, è necessario installarlo nel corretto *workspace*. Ogni *workspace* ha il proprio file *package.json*, che contiene solo le dipendenze specifiche necessarie per quel workspace. Questa separazione garantisce che ogni parte del progetto abbia accesso solo alle librerie di cui ha bisogno, evitando conflitti e riducendo il rischio di dipendenze non utilizzate. Inoltre, ogni *workspace* ha una propria cartella *node_modules* dedicata, che contiene tutte le librerie e le dipendenze installate specificamente per quel workspace. Questa configurazione consente di mantenere un ambiente di sviluppo isolato per ciascun workspace, facilitando la gestione e l'aggiornamento delle dipendenze in modo indipendente. In sintesi, l'uso di *workspaces* e la gestione decentralizzata delle dipendenze attraverso file *package.json* separati e cartelle *node_modules* dedicate contribuiscono a rendere la monorepo più organizzata, modulare e manutenibile. Questa struttura permette ai team di sviluppo di lavorare in modo più efficiente, riducendo i tempi di integrazione e semplificando il processo di aggiornamento delle librerie.

## 5.2 Comunicazione

La comunicazione tra il frontend e il backend è un aspetto cruciale nello sviluppo di applicazioni moderne. Questo capitolo descrive i protocolli di comunicazione utilizzati, come REST e GraphQL, e l'uso di strumenti di code generation per mantenere il codice tipizzato e sincronizzato con lo schema GraphQL.

### 5.2.1 Protocolli di Comunicazione

Per garantire una comunicazione efficace e sicura tra il frontend e il backend, sono stati adottati diversi protocolli di comunicazione. Tra i principali protocolli utilizzati troviamo REST e GraphQL, ciascuno con specifici casi d'uso e vantaggi.

#### REST

Le API REST sono utilizzate principalmente per l'autenticazione degli utenti. REST è un'architettura leggera che utilizza i metodi HTTP standard (GET, POST, PUT, DELETE) per eseguire operazioni CRUD (Create, Read, Update, Delete) sui dati. Un esempio di endpoint REST per l'autenticazione potrebbe essere:

```
POST /api/auth/login
{
    "username": "example_user",
    "password": "example_password"
}
```

Questo endpoint accetta le credenziali dell'utente e, se valide, restituisce un token JWT che viene utilizzato per autenticare le richieste successive.

#### GraphQL

GraphQL è un linguaggio di query per le API che offre una maggiore flessibilità rispetto a REST. In DDC Service, una volta ottenuto il token JWT tramite REST, GraphQL viene utilizzato per interrogare il backend. I vantaggi di GraphQL includono la possibilità di richiedere esattamente i dati necessari e la riduzione del numero di richieste necessarie per ottenere tutte le informazioni richieste.

Esempio generico di una query GraphQL:

```
query {
    user(id: "123") {
        id
        name
        email
```

```
    }
}
```

Esempio generico di una mutazione GraphQL:

```
mutation {
    updateUser(id: "123", input: {name: "New Name"}) {
        id
        name
        email
    }
}
```

GraphQL è stato scelto per la sua efficienza nella gestione delle richieste complesse e per la sua capacità di evolversi senza influenzare le versioni precedenti dell'API.

### 5.2.2 GraphQL Codegen e RTK Query

L'utilizzo combinato di strumenti di code generation per GraphQL, come GraphQL Code Generator, e Redux Toolkit Query (RTK Query) è fondamentale per mantenere il codice tipizzato e sincronizzato con lo schema GraphQL. GraphQL Code Generator genera automaticamente tipi TypeScript per le query, le mutazioni e i frammenti definiti, mentre RTK Query facilita l'integrazione di queste query nel sistema di gestione dello stato. Questa combinazione migliora la sicurezza del tipo e riduce gli errori di runtime, oltre a fornire un'architettura chiara e scalabile per la gestione delle API GraphQL nel frontend.

#### Configurazione e Utilizzo

La configurazione di GraphQL Codegen nel progetto è semplice e diretta. I file .graphql, contenenti tutte le query da eseguire, sono memorizzati all'interno della cartella *graphql/documents*.

Di seguito è riportato un esempio di configurazione nel file *codegen.config.ts*:

```
import { CodegenConfig } from '@graphql-codegen/cli'

const config: CodegenConfig = {
schema: './store/graphql/schema.json',
documents: ['./store/graphql/documents/**/*.graphql'],
ignoreNoDocuments: true, // for better experience with the watcher
generates: {
    './store/graphql/queryBaseApi.ts': {
    plugins: [
        'typescript',
        'typescript-operations',
        'typescript-resolvers',
        {
        'typescript-rtk-query': {
            importBaseApiFrom: 'ddc-service/store/api/baseApi',
            importBaseApiAlternateName: 'baseApi',
            exportHooks: true,
        },
        },
    ],
    },
},
}

export default config
```

**Codice 5.1:** Configurazione GraphQL Codegen: codegen.config.ts

Questo file di configurazione specifica l'endpoint dello schema GraphQL (*schema.json*), i documenti GraphQL (*documents*) da cui generare il codice, e i plugin da utilizzare per la generazione. Inoltre, viene generato un file chiamato *queryBaseApi.ts*, che contiene tutte le query e le mutazioni definite nei file .graphql.

Una volta configurato, è possibile eseguire il comando di generazione:

`graphql-codegen`

Il file *queryBaseApi.ts* generato viene utilizzato per integrare le query e le mutazioni GraphQL nel codice del frontend, sfruttando Redux per la gestione della cache.

Queste API sono cacheate su Redux e possono essere chiamate semplicemente con:

```
import { useGetOvensQuery }
from 'ddc-service/store/graphql/queryBaseApi'
```

Per utilizzare la query:

```
const { data, isFetching, isLoading, isSuccess,
isError, isUninitialized, error, refetch } = useGetOvensQuery({})
```

**Codice 5.2:** Utilizz API GraphQL

- **data**: contiene i dati recuperati dalla query.
- **isFetching**: booleano che indica se la query è attualmente in fase di recupero.
- **isLoading**: booleano che indica se la query è in fase di caricamento iniziale.
- **isSuccess**: booleano che indica se la query è stata completata con successo.
- **isError**: booleano che indica se c'è stato un errore nell'esecuzione della query.
- **isUninitialized**: booleano che indica se la query non è stata ancora eseguita.
- **error**: contiene informazioni sull'errore se la query ha fallito.
- **refetch**: funzione che permette di eseguire nuovamente la query.

L'utilizzo di GraphQL Codegen garantisce che il codice rimanga sincronizzato con lo schema GraphQL, riducendo il rischio di errori e migliorando la produttività del team di sviluppo.

### 5.2.3 Autenticazione

Descrivere il sistema di autenticazione utilizzato nell'applicazione. Includere dettagli sui flussi di autenticazione, i token JWT e le strategie di sicurezza implementate.

#### Flussi di Autenticazione

Descrivere i vari flussi di autenticazione, come l'accesso tramite username e password, l'autenticazione a due fattori, e il rinnovo dei token.

#### Gestione Cookie

#### Gestione dei Token JWT

Dettagliare l'uso di JSON Web Tokens (JWT) per l'autenticazione, inclusi i processi di generazione, validazione e gestione dei token.

## 5.3 Architettura a Componenti

### 5.3.1 Componenti di Base Design System

Descrivere i componenti di base dell'applicazione e come vengono creati utilizzando React. Includere esempi di componenti comuni come bottoni, input e layout.

### 5.3.2 Componenti Compositi

Descrivere i componenti compositi, che combinano i componenti di base per formare parti più complesse dell'interfaccia utente.

#### Esempi di Componenti Compositi

Fornire esempi di componenti compositi come moduli di login, tabelle di dati e dashboard.

### 5.3.3 Gestione dello Stato

Descrivere come viene gestito lo stato dell'applicazione utilizzando Redux o un altro sistema di gestione dello stato. Includere dettagli su azioni, riduttori e middleware.

#### Azioni e Riduttori

Descrivere il ruolo delle azioni e dei riduttori nella gestione dello stato, includendo esempi pratici.

#### Middleware

Descrivere l'uso dei middleware per gestire operazioni asincrone e altri effetti collaterali nello stato dell'applicazione.

### 5.3.4 Stilizzazione dei Componenti

Descrivere le tecniche di stilizzazione utilizzate per i componenti React, come CSS-in-JS, Styled Components, e altri approcci moderni.

#### Esempi di Stilizzazione

Fornire esempi di stilizzazione dei componenti, spiegando come vengono applicati gli stili in modo modulare e riutilizzabile.

# Capitolo 6

# Studio fattibilità app in monorepo

## 6.1 Descrizione della monorepo

## 6.2 Descrizione routing e navigazione

## 6.3 Problemi riscontrati

### 6.3.1 Versionamento dipendenze

### 6.3.2 Isolamento delle dipendenze

# Capitolo 7

# Conclusioni

## 7.1 Consuntivo finale

## 7.2 Raggiungimento degli obiettivi

## 7.3 Conoscenze acquisite

## 7.4 Valutazione personale

# Bibliografia

## Testi

James P. Womack, Daniel T. Jones. *Lean Thinking, Second Editon*. Simon & Schuster, Inc., 2010.

## Articoli

Einstein, Albert, Boris Podolsky e Nathan Rosen. «Can Quantum-Mechanical Description of Physical Reality be Considered Complete?» In: *Physical Review* 47.10 (1935), pp. 777–780. DOI: 10.1103/PhysRev.47.777.

# Sitografia

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