



UNIVERSITÀ
DEGLI STUDI
DI UDINE
HIC SUNT FUTURA

DIPARTIMENTO DI SCIENZE MATEMATICHE, INFORMATICHE E FISICHE

TESI DI LAUREA IN
INFORMATICA

Integrazione di Tecnologie Moderne per un CRM Efficiente: Un Caso Studio su Next.js

CANDIDATO

Alessandro Gerotto

RELATORE

Prof. Vincenzo Riccio

TUTOR AZIENDALE

Yannick Ponte

Anno accademico 2023-2024

CONTATTI DELL'ISTITUTO

Dipartimento di Scienze Matematiche, Informatiche e Fisiche

Università degli Studi di Udine

Via delle Scienze, 206

33100 Udine — Italia

+39 0432 558400

<https://www.dmif.uniud.it/>

Indice

Indice	iii
1 Introduzione	3
1.1 Introduzione generale	3
1.2 Struttura della tesi	3
1.3 L'importanza di un CRM	4
1.3.1 Vantaggi dell'uso di un CRM	4
1.3.2 Alternative ai sistemi CRM	4
1.4 L'importanza di una soluzione cloud-native	4
1.4.1 Conclusione	5
2 Background aziendale	7
2.1 Archiedo	7
2.2 Prodotto software	7
2.2.1 Funzionalità principali	7
2.2.2 Obiettivi del software	8
2.2.3 Stakeholders	8
2.3 Ciclo di vita adottato	8
2.3.1 Plan-based vs Agile development	8
2.4 Formazione e Pair Programming	9
2.5 Problemi e valutazioni	9
3 Background tecnico	11
3.1 Introduzione	11
3.2 JavaScript e TypeScript	11
3.2.1 Le interfacce in TypeScript	12
3.3 React	13
3.3.1 I componenti	14
3.3.2 Le <i>props</i>	14
3.3.3 Gli hook	15
3.4 Next.js	17
3.4.1 Rendering	17
3.4.2 App routing	19
3.5 Next UI e Tailwind CSS	21
3.5.1 Next UI	21
3.5.2 Tailwind CSS	22
3.6 AWS: Amazon Web Service	22
4 Descrizione dei requisiti	25
4.1 Introduzione	25
4.2 Definizione dei requisiti del sistema	25
4.2.1 Gestione dei ruoli degli utenti	26

4.2.2	Operazioni di CRUD	27
4.3	Diagrammi	30
4.3.1	Diagramma dei casi d'uso (Use Case Diagram)	30
4.3.2	Sequence diagram	31
5	Testing	33
5.1	Tipologie di test condotti	33
5.1.1	Testing unitario	33
5.1.2	Testing di integrazione	34
5.1.3	Testing di accettazione	34
5.2	Casi di test	34
5.2.1	Test di creazione progetto	34
5.2.2	Test di cancellazione progetto attraverso la pagina dei progetti	35
5.2.3	Test di cancellazione progetto attraverso la pagina del dettaglio	35
5.2.4	Test di modifica progetto	35
5.2.5	Test di accesso riservato	36
5.3	Risultati	36
5.4	Aspetti non testati e proposte di miglioramento	36
5.4.1	Test di sicurezza	37
5.4.2	Test di accessibilità	37
6	Conclusioni	39
6.1	Obiettivi raggiunti	39
6.2	Lezioni apprese	39
6.3	Possibili sviluppi futuri	40
6.3.1	Potenziamento della logica di gestione degli stati	40
6.3.2	Analisi dei dati	41
6.3.3	Sviluppo di test sulla sicurezza e sull'accessibilità	41
6.3.4	Implementazione della gestione utenti	41
7	Ringraziamenti	43
	Bibliografia	45

Capitolo 1

Introduzione

1.1 Introduzione generale

Il presente lavoro di tesi si propone di analizzare il progetto di un software CRM (Customer Relationship Management) realizzato durante il periodo di tirocinio presso Archeido. L'obiettivo principale del progetto è stato quello di sviluppare una soluzione software efficace e innovativa, in grado di rispondere alle esigenze di gestione delle relazioni con i clienti.

Più nel dettaglio, lo scopo di questo progetto è sviluppare un CRM per gestire e monitorare i progetti in corso all'interno dell'azienda. Quando viene commissionato un nuovo progetto, questo viene registrato nella dashboard del CRM, dove è possibile tenere traccia delle tempistiche, dello stato di avanzamento e delle persone assegnate al progetto. Il sistema consente di monitorare l'intero ciclo di vita del progetto, dalla fase iniziale di pianificazione fino alla consegna finale, garantendo una gestione più efficiente delle risorse e una maggiore trasparenza nel processo operativo.

Per raggiungere questo obiettivo, è stato adottato un approccio di sviluppo basato su metodologie agili, che hanno consentito di adattare rapidamente le funzionalità del software alle necessità emergenti durante il processo di sviluppo.

1.2 Struttura della tesi

Nelle sezioni seguenti, si fornirà un'analisi dettagliata del prodotto software sviluppato, evidenziando le sue funzionalità principali, gli obiettivi prefissati e gli stakeholders coinvolti. Inoltre, verrà discusso il ciclo di vita adottato per il progetto, i problemi riscontrati e le valutazioni effettuate, nonché il percorso di formazione che ha affiancato il tirocinio. Successivamente, sarà fornita una panoramica delle tecnologie utilizzate e dei requisiti funzionali del sistema, accompagnata da diagrammi esplicativi per facilitare la comprensione del lavoro svolto.

Infine, saranno discussi i metodi di testing impiegati, le aree che hanno ricevuto maggior attenzione e quelle che necessiterebbero di ulteriori verifiche. In prospettiva, saranno esplorate le opportunità di ampliamento del progetto, come l'integrazione di nuove funzionalità o l'ottimizzazione di quelle preesistenti.

1.3 L'importanza di un CRM

Un sistema CRM (Customer Relationship Management) è un software progettato per supportare le aziende nella gestione delle relazioni con i clienti, fornendo una piattaforma centralizzata per raccogliere e organizzare informazioni sui clienti, le interazioni e le attività correlate. L'importanza di adottare un CRM è evidente in diversi contesti aziendali, poiché consente di migliorare la qualità delle relazioni con la clientela, ottimizzare i processi interni e supportare strategie di crescita orientate ai dati.

1.3.1 Vantaggi dell'uso di un CRM

L'adozione di un CRM comporta una serie di vantaggi strategici e operativi per le aziende. Tra i principali vantaggi, possiamo identificare i seguenti:

- **Centralizzazione e accessibilità dei dati:** un CRM permette di raccogliere tutte le informazioni sui clienti e sui progetti in un unico sistema accessibile ai vari dipartimenti aziendali. Ciò consente una maggiore coordinazione tra team (ad esempio vendite, marketing e assistenza).
- **Facilità di monitoraggio dei progetti:** un CRM permette di monitorare l'avanzamento di ogni progetto, assegnare e verificare compiti e tempistiche, e garantire che ogni fase venga portata a termine nei tempi previsti.
- **Miglioramento dell'efficienza operativa:** grazie alla possibilità di monitorare e automatizzare le attività, un CRM riduce il tempo necessario per gestire progetti e interazioni, abbassando i rischi di errore umano e velocizzando i processi decisionali.
- **Miglioramento della fidelizzazione e soddisfazione del cliente:** un CRM permette di gestire in modo personalizzato le interazioni con ogni cliente, aumentando la capacità di rispondere alle loro esigenze. Ciò contribuisce a costruire rapporti più duraturi e a migliorare il tasso di fidelizzazione.

1.3.2 Alternative ai sistemi CRM

Oltre ai software CRM progettati per la gestione delle relazioni con i clienti, le aziende possono considerare altre soluzioni per organizzare dati e processi. I fogli di calcolo, come Excel o Google Sheets, rappresentano una soluzione economica, flessibile e semplice, adatta a piccoli team o aziende in fase iniziale, ma limitata in scalabilità e automazione. I sistemi ERP, invece, offrono una gestione aziendale completa, che include funzionalità per contabilità, produzione e, in alcuni casi, CRM. Tuttavia, un ERP può risultare complesso e costoso, soprattutto se le aziende necessitano solo di strumenti per la gestione dei clienti. Infine, piattaforme di gestione progetti come Trello, Asana o Monday permettono di organizzare e monitorare le attività interne, ma non offrono le funzionalità avanzate di un CRM, come il tracciamento delle interazioni e l'analisi delle vendite.

1.4 L'importanza di una soluzione cloud-native

Adottare un CRM cloud-native offre accesso continuo ai dati da qualsiasi dispositivo, favorendo la collaborazione e il lavoro remoto. La scalabilità del cloud permette di adattare le risorse senza costi infrastrutturali, mentre la manutenzione e la sicurezza sono gestite dal fornitore, riducendo oneri per

l'azienda. Inoltre, le soluzioni cloud-native si integrano facilmente con altre applicazioni, centralizzando i dati e automatizzando i processi, migliorando l'efficienza operativa dell'azienda.

1.4.1 Conclusione

La scelta di un sistema CRM è, in definitiva, determinata dalle esigenze aziendali specifiche e dal livello di complessità delle relazioni con i clienti che si intende gestire. Un CRM rappresenta un'opzione fondamentale per le aziende che desiderano migliorare l'organizzazione e la qualità delle proprie relazioni commerciali, con un vantaggio competitivo legato alla possibilità di creare esperienze cliente più personalizzate e gestire i progetti in modo più efficiente.

Capitolo 2

Background aziendale

Questo capitolo fornisce un quadro dettagliato del contesto aziendale di Archeido. Vengono esplorate le specializzazioni di tale azienda nell'ingegneria del software e nella trasformazione digitale, ponendo l'accento sull'approccio cloud-native e sull'uso delle metodologie Agile.

Si presenterà il CRM come una soluzione centrale per il monitoraggio dei progetti, evidenziando le sue funzionalità principali e gli obiettivi strategici che intende perseguire. Inoltre, verranno identificati gli stakeholders coinvolti nel progetto e analizzato il ciclo di vita del software adottato, mettendo in luce l'importanza della flessibilità e dell'adattamento.

Infine, saranno discussi i problemi affrontati e le decisioni critiche riguardanti le tecnologie utilizzate e il percorso di formazione intrapreso per garantire un'efficace implementazione del sistema.

2.1 Archiedo

Archeido [1] è un'azienda specializzata nell'ingegneria del software, focalizzata sulla creazione e implementazione di soluzioni su architetture cloud-native e infrastrutture Cloud, utilizzando principalmente Amazon Web Services. La missione di Archeido è supportare le aziende durante il processo di trasformazione digitale e adozione del Cloud, offrendo un'assistenza sia tecnica che strategica con massima trasparenza. L'azienda adotta e si basa sulle metodologie Agile, garantendo un processo di sviluppo rapido, sicuro ed efficiente.

2.2 Prodotto software

Il software sviluppato e descritto in questa tesi è un CRM (Customer Relationship Management), progettato per servire come una dashboard centrale per il monitoraggio dei progetti attualmente in corso, come archivio completo di quelli già completati e come piattaforma di pianificazione per i progetti futuri.

2.2.1 Funzionalità principali

Il CRM offre una gestione completa dei progetti, facilitandone le operazioni di creazione, lettura, aggiornamento e cancellazione (CRUD). Inoltre, consente di monitorare gli utenti autorizzati ad accedere alla dashboard, i quali vengono creati tramite inviti generati e condivisi da un super amministratore. A ciascun utente viene assegnato un ruolo specifico, che definisce le regole di visibilità e le autorizzazioni di

accesso alle informazioni presenti nella dashboard. Infine, il sistema consente di valutare le performances aziendali, fornendo dati e metriche utili per analizzare l'efficacia dei progetti e il rendimento complessivo del team.

2.2.2 Obiettivi del software

Il CRM ha come obiettivo principale valutare le stime concordate con i clienti, consentendo ai project manager di confrontare le previsioni iniziali stabilite insieme al cliente con i risultati effettivi ottenuti a progetto completato. In questo modo, è possibile identificare eventuali scostamenti e adottare misure correttive tempestive per garantire il successo del progetto. Il CRM sviluppato è dunque uno strumento di monitoraggio che fornisce un supporto strategico volto all'ottimizzazione delle relazioni con i clienti.

2.2.3 Stakeholders

Gli stakeholders di questo progetto includono:

- I **Super Amministratori**, che gestiscono la creazione degli utenti, assegnano loro permessi e regole di visibilità e hanno accesso completo a tutte le sezioni del CRM.
- I **Project Manager**, i quali ricevono un account per accedere alla dashboard e possono visualizzare solo i progetti a cui sono stati assegnati, limitando l'accesso ad alcune sezioni.
- I **Clienti**, le cui esigenze influenzano l'uso del CRM, poiché il sistema è progettato per rispondere meglio alle loro necessità.

2.3 Ciclo di vita adottato

Archeido adotta un ciclo di vita del software basato su metodologie agili, che consentono di gestire progetti in modo iterativo e incrementale. Lo sviluppo Agile si caratterizza per un approccio flessibile, orientato a fornire rapidamente software funzionante al cliente, garantendo aggiornamenti frequenti e miglioramenti continui. In questo contesto, le fasi di specifica, progettazione, sviluppo, validazione ed evoluzione si susseguono in un processo fluido e non fisso, caratterizzato da una continua negoziazione dei requisiti e raccolta di feedback.

2.3.1 Plan-based vs Agile development

La differenza tra Agile e lo sviluppo plan-based risiede principalmente nell'approccio alla gestione dei progetti e nella flessibilità di risposta ai cambiamenti. Nel modello plan-based, il processo di sviluppo è caratterizzato da una pianificazione dettagliata e rigida delle fasi del progetto, dove i requisiti vengono definiti all'inizio e rimangono relativamente statici per tutta la durata del ciclo di vita del progetto. Al contrario, le metodologie Agile, come Scrum e Kanban, pongono l'accento sulla collaborazione continua tra i membri del team e gli stakeholder, consentendo una maggiore adattabilità. I requisiti non sono definitivi e possono evolvere nel corso del progetto, permettendo al team di rispondere rapidamente alle esigenze emergenti.

2.4 Formazione e Pair Programming

Il periodo iniziale di questo tirocinio è stato dedicato alla formazione, durante la quale è stata approfondita la programmazione funzionale in JavaScript [4], acquisendo le competenze necessarie per scrivere codice più efficiente, leggibile e mantenibile. In seguito, si è analizzato React [7], una libreria JavaScript sviluppata da Facebook per la creazione di interfacce utente interattive e componenti riutilizzabili, facilitando lo sviluppo di applicazioni web complesse. Infine, è stato esplorato Next.js, un framework che estende le capacità di React, permettendo il rendering lato server e facilitando la creazione di applicazioni web performanti.

Una parte fondamentale della formazione è stata lo sviluppo prevalentemente in modalità di **Pair Programming**, una pratica essenziale della metodologia **Extreme Programming (XP)**, che ha velocizzato e semplificato l'apprendimento grazie alla collaborazione diretta tra i membri del team. In questo approccio, due programmatori lavorano insieme su un singolo computer: uno scrive il codice (driver), mentre l'altro lo esamina e fornisce suggerimenti (observer o navigator), favorendo un continuo scambio di conoscenze e un confronto immediato sulle soluzioni.

L'adozione di XP ha migliorato significativamente la qualità del codice, grazie al feedback immediato e all'integrazione di pratiche agili come il **refactoring continuo** e la **revisione condivisa** del lavoro. Questo metodo ha permesso di mettere in pratica fin da subito le nozioni apprese nei corsi di JavaScript funzionale e React sopramenzionati, consentendo di affinare rapidamente le competenze tecniche e di adottare efficacemente pratiche di **clean code**, garantendo uno sviluppo più strutturato, di qualità e conforme alle best practices della scrittura del codice adottate in Archeido.

2.5 Problemi e valutazioni

Nelle prime fasi dello sviluppo, è stata condotta un'analisi approfondita per valutare quale tra le diverse tecnologie disponibili e librerie grafiche fosse più adatta per la realizzazione del progetto. Alla fine, si è deciso di utilizzare Next.js [8] e l'omonima libreria grafica NextUI [5].

Un altro aspetto cruciale nella fase di sviluppo è stata la scelta tra l'utilizzo di JavaScript puro e l'adozione di TypeScript. Questa decisione ha comportato una valutazione approfondita dei vantaggi e degli svantaggi di entrambe le opzioni, considerando fattori come la tipizzazione statica, la manutenzione del codice e la scalabilità del progetto, approfonditi ulteriormente nelle sezioni seguenti.

Capitolo 3

Background tecnico

3.1 Introduzione

In questo capitolo, verrà presentato il contesto tecnico del progetto, descrivendo nel dettaglio le tecnologie principali utilizzate: TypeScript, React, Next.js, AWS, Next UI e Tailwind CSS.

Queste tecnologie rappresentano la base dell'architettura del sistema e sono state selezionate per le loro caratteristiche che consentono di sviluppare un CRM moderno, efficiente e altamente scalabile. Verranno analizzati i punti di forza di ciascuna tecnologia, con un focus su come ognuna di esse contribuisca alla creazione di un'interfaccia utente reattiva, di un'esperienza utente fluida e di una gestione robusta dei dati.

3.2 JavaScript e TypeScript

Nel contesto di questo progetto si è optato per l'utilizzo di **TypeScript** [11]. Questa scelta è stata effettuata durante la fase iniziale di analisi dei requisiti e il motivo principale è stato che TypeScript presenta un **sistema di tipi statici** rispetto a JavaScript, che invece ne utilizza uno a tipizzazione dinamica. Questo significa che i tipi delle variabili, delle funzioni e delle proprietà vengono definiti al momento della compilazione e non cambiano durante l'esecuzione, permettendo di rilevare gli errori in fase di compilazione piuttosto che durante l'esecuzione. Tuttavia, il browser non può interpretare direttamente il codice TypeScript, il quale però è completamente compatibile con JavaScript. Infatti, prima di poter essere eseguito nel browser, il codice TypeScript deve essere convertito in JavaScript tramite un processo di compilazione. Questo processo è tipicamente gestito da strumenti come il TypeScript Compiler (tsc), Webpack, Babel e altri bundler che supportano TypeScript. Durante la compilazione, TypeScript verifica la correttezza dei tipi e successivamente trasforma il codice TypeScript in JavaScript, che può essere eseguito da qualsiasi browser. Un esempio pratico in cui JavaScript risulta problematico è il seguente:

```
1 const add = (a, b) => {  
2   return a + b;  
3 }  
4 add("5", 10); // -> "510"
```

Listing 3.1: Coercion in JavaScript

```
1 const add = (a: number, b: number) => {  
2   return a + b;  
3 }  
4 add("5", 10); -> // Errore
```

Listing 3.2: Errore in TypeScript

Qui viene dichiarata la funzione `add`, la quale esegue la somma degli argomenti `a` e `b` che riceve come input. In JavaScript, essendo un linguaggio a tipizzazione dinamica, non viene generato alcun errore perché viene applicata automaticamente la **coercion**, ovvero la conversione implicita di un tipo di dato in un altro per eseguire l'operazione richiesta. Nel caso di `add("5", 10)`, il numero 10 viene infatti convertito in una stringa "10" per consentire la concatenazione con "5", producendo il risultato "510". Questo accade in quanto l'operatore `+` in JavaScript somma i valori se entrambi sono numeri, ma effettua la concatenazione se almeno uno dei due è una stringa. Al contrario, in TypeScript la chiamata `add("5", 10)` provoca un errore di compilazione, poiché il linguaggio richiede esplicitamente che i parametri siano di tipo `number`. Per questo motivo, TypeScript offre un notevole vantaggio, poiché impedisce le conversioni implicite di tipi, che spesso sono causa di comportamenti imprevedibili.

3.2.1 Le interfacce in TypeScript

Un altro punto a favore viene evidenziato in contesti dove le applicazioni sono più complesse: TypeScript offre un'architettura di codice più chiara e strutturata, mettendo a disposizione le **interfacce**. Queste ultime fungono da *blueprint*, definendo i tipi esplicitamente. Ad esempio, per poter creare un nuovo progetto all'interno del CRM, è necessario che esso segua una struttura definita. Attraverso l'uso delle interfacce, è possibile stabilire chiaramente quali proprietà e quali tipi di dati devono essere presenti, garantendo così la corretta integrazione e funzionamento all'interno del sistema. Durante l'analisi dei requisiti, è emerso che ogni progetto registrato deve includere i seguenti campi:

- **ID univoco**: necessario per distinguere i diversi progetti.
- **Codice progetto**: può non essere univoco.
- **Nome del progetto**: la denominazione assegnata al progetto.
- **Data di inizio**: corrisponde alla data in cui il team inizia a lavorare attivamente sul progetto.
- **Data di fine**: corrisponde alla data prevista per la consegna finale del progetto.
- **Project Manager assegnati**: i responsabili del progetto.
- **Stato del progetto**: può essere uno tra i seguenti: `Not Started`, `In Progress` o `Completed`.

Questa specifica ha consentito di definire un'interfaccia `Project`, garantendo che chiunque la implementi debba rispettare lo schema riportato di seguito:

```
1 export interface Project {  
2     id: string;  
3     code: string;  
4     name: string;  
5     startdate: string | number;  
6     enddate: string | number;  
7     state: string;  
8     assignedTo: string;  
9 }
```

Listing 3.3: Interfaccia di progetto

Inizialmente, non era previsto che un progetto potesse essere assegnato a specifici Project Manager, quindi il sistema era configurato per consentire a tutti gli utenti di accedere a ogni progetto. Grazie a TypeScript, l'aggiornamento del sistema è stato notevolmente semplificato: modificando l'interfaccia, gli errori di compilazione sono stati segnalati automaticamente in tutte le parti del codice interessate, facilitando l'individuazione delle sezioni da aggiornare. Questo ha reso il processo di adeguamento più rapido e preciso, assicurando al contempo la qualità del software.

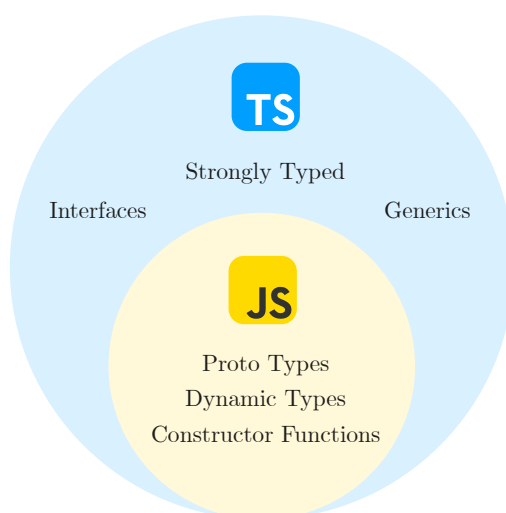


Figura 3.1: TypeScript e JavaScript

3.3 React

React è una libreria JavaScript per la creazione di interfacce utente reattive e componenti riutilizzabili. React consente di costruire interfacce utente utilizzando i cosiddetti **componenti** sfruttando **JSX** (o **TSX** nel caso di TypeScript), una sintassi di estensione per JavaScript che permette di scrivere codice che combina quest'ultimo con HTML, in un formato che è più leggibile e comprensibile. Nell'esempio sotto riportato, il componente **Greeting** viene inizializzato usando JavaScript, ma restituisce un codice HTML:

```
1  const name = 'Alice';
2  const Greeting = () => {
3    return (
4      <h1>
5        Hello, {name}!
6      </h1>
7    )
8  }
```

Listing 3.4: Esempio di codice React

3.3.1 I componenti

Un componente React è una funzione JavaScript che rappresenta una parte riutilizzabile e autonoma di un'interfaccia utente in un'applicazione React. I componenti permettono di scomporre l'interfaccia in parti più piccole e gestibili, promuovendo la leggibilità, la manutenzione e il riuso del codice. Questo approccio basato sui componenti e sulla continua estrazione del codice in componenti più semplici garantisce diversi vantaggi:

- **Riuso:** un componente può essere utilizzato in diverse parti dell'applicazione, riducendo la duplicazione del codice.
- **Isolamento:** ogni componente ha il proprio stato e le proprie proprietà, rendendo il singolo componente utilizzabile e testabile in ambienti isolati, rimuovendo le dipendenze da quanto appare al suo esterno.
- **Composizione di componenti:** una volta sviluppati diversi componenti, React consente di comporli insieme. Ad esempio, un componente `Card` potrebbe contenerne a sua volta tre più semplici: un'immagine, un titolo e un pulsante. Tale `Card` potrebbe venire poi usata molteplici volte all'interno di un componente più complesso, come un `Carousel`.

3.3.2 Le *props*

Le props permettono a un componente genitore di fornire informazioni immutabili a un componente figlio, rendendo quest'ultimo più dinamico e flessibile.

```
1 import React from 'react';
2 import Greeting from './Greeting';
3
4 const App = () => {
5   return (
6     <div>
7       <Greeting name= "Alice" />
8       <Greeting name= "Bob" />
9     </div>
10   );
11 };
```

Listing 3.5: Componente padre

```
1 import React from 'react';
2
3 const Greeting = ({ name }) => {
4   return (
5     <h1>
6       Hello, {name}!
7     </h1>
8   );
9 };
10
11 export default Greeting;
```

Listing 3.6: Componente figlio

In questo esempio, il componente `Greeting` (sulla destra) è una funzione che accetta la prop `name` e restituisce il testo “Hello,” seguito dal valore di tale prop, ottenuta dal componente padre.

Per esempio, quando viene passato “Alice” come valore, come accade in `<Greeting name="Alice" />`, il risultato renderizzato sarà un'intestazione `h1` che riporta “Hello, Alice!”, mentre se viene passato “Bob” (attraverso `<Greeting name="Bob" />`), verrà stampato “Hello, Bob!”.

3.3.3 Gli hook

Gli hook sono una caratteristica di React introdotta con la versione 16.8 che consente di utilizzare lo stato e altre funzionalità di React nei componenti funzionali, senza la necessità di convertire il componente in una classe. Gli hook forniscono un modo per gestire lo stato locale, eseguire effetti collaterali, e utilizzare contesti, tra le altre operazioni, rendendo il codice più pulito e riutilizzabile.

Gli hook più comuni includono `useState`, che permette di mantenere e modificare lo stato locale all'interno di un componente, e `useEffect`, che consente di gestire effetti collaterali, come il caricamento di dati o la registrazione di eventi. Più nel dettaglio, `useState` restituisce un array composto da due elementi: il valore attuale dello stato (`selectedProject`) e una funzione per aggiornare tale valore (`setSelectedProject`). `useState` accetta un argomento, il quale rappresenta lo stato iniziale.

```
const [selectedProject, setSelectedProject] = useState<Project>()
```

`useEffect` invece, è un hook di React utilizzato per gestire **effetti collaterali** all'interno di un componente, come operazioni di recupero dati, manipolazione del DOM o registrazione di eventi. Accetta due argomenti:

- Una **funzione** che contiene il codice da eseguire `() => ...`;
- Un **array** di dipendenze che specifica i valori da monitorare `[...]`. Quando una di queste dipendenze subisce una modifica, il corpo di `useEffect` viene eseguito. Tale esecuzione spesso genera effetti collaterali, con conseguente cambiamento di stato. Quando ciò accade, si verifica un rendering aggiornato del componente. Se, come in questo caso, il secondo parametro è un array vuoto, la funzione passata come primo argomento all'hook verrà eseguita soltanto la prima volta.

Un esempio d'uso dei due hook sopracitati è il seguente, in cui si definisce una funzione chiamata `foo`. All'interno di questa funzione, viene utilizzato l'hook `useState` per creare una variabile di stato denominata `selectedProject` e la relativa funzione di aggiornamento `setSelectedProject`. L'inizializzazione di `selectedProject` è impostata su un oggetto vuoto, indicando che inizialmente non contiene dati.

Successivamente, viene impiegato l'hook `useEffect`. In questo caso, l'array di dipendenze è vuoto, il che significa che la funzione sarà eseguita solo una volta, quando il componente viene montato per la prima volta. All'interno della funzione `useEffect`, viene chiamata `getProjectFromDatabase(1)`, il quale recupera il progetto identificato dall'ID 1. Una volta ottenuto il progetto, viene utilizzata `setSelectedProject` per aggiornare lo stato di `selectedProject` con il progetto recuperato.

```
1 const foo = () => {
2   const [selectedProject, setSelectedProject] = useState<Project>({});
3
4   useEffect(() => {
5     const myProject = getProjectFromDatabase(1)
6     setSelectedProject(myProject)
7   }, []);
8 }
```

Listing 3.7: `useEffect` e `useState` per ottenere e salvare un progetto in uno stato locale

Un altro hook molto popolare è `useMemo`. Tale hook viene utilizzato per memorizzare i risultati di una funzione e restituisce un valore memorizzato che cambia solo quando le sue dipendenze variano. Questo assicura che i calcoli costosi non vengano ripetuti a ogni rendering, ottimizzando così le prestazioni dei componenti React. Tuttavia, l'uso di `useMemo` dovrebbe essere impiegato solo quando si tratta di calcoli che richiedono molto tempo o risorse e i risultati non cambiano tra i rendering, come ad esempio:

- **Cicli annidati:** l'uso di `useMemo` permette di evitare l'esecuzione ripetuta dei cicli annidati ad ogni render, migliorando l'efficienza. Questo può essere il caso, ad esempio, in cui un componente React deve iterare su una matrice bidimensionale per calcolare dei risultati. Senza `useMemo`, ogni volta che il componente si renderizza, il ciclo annidato viene eseguito di nuovo, anche se i dati non sono cambiati;
- **Operazioni ricorsive e calcoli matematici complessi:** `useMemo` memorizza il risultato di operazioni ricorsive complesse, prevenendo ricalcoli non necessari. Un esempio potrebbe essere un componente React che esegue una funzione ricorsiva per calcolare il numero di Fibonacci. Se `useMemo` non venisse usato, la funzione verrebbe rieseguita completamente ad ogni render, anche se il numero d'ingresso rimane lo stesso;
- **Trasformazioni di grandi quantità di dati:** ad esempio, quando si lavora con array di dimensioni significative, `useMemo` garantisce che la trasformazione venga ricalcolata solo per i dati modificati rispetto alla precedente esecuzione.

Tuttavia, un uso eccessivo di questo hook può comportare gravi problemi, come evidenziato da Edvins Antonovs nel suo articolo intitolato ‘`useMemo overdose`’ [2] e come illustrato da Giuseppe Funicello durante il ReactJS Day 2024 svolto a Verona il 25/10/2024 [3]. Un abuso di `useMemo` può portare a:

- **Ottimizzazione prematura:** nella maggior parte dei casi, quando un componente richiede un notevole potere di calcolo e impiega molto tempo per elaborarsi, si tende a utilizzare questo hook per affrontare il collo di bottiglia, anziché gestire direttamente la causa del problema.
- **Effetti collaterali indesiderati:** l'uso eccessivo di tale hook può portare a effetti collaterali imprevisti se le dipendenze non vengono gestite correttamente. Se le dipendenze sono fornite in modo accurato, si possono evitare situazioni in cui il valore memorizzato non si aggiorna quando dovrebbe, portando a risultati obsoleti o errati.
- **Complessità non necessaria:** un uso eccessivo di questo hook aumenta la ridondanza nel codice, rendendolo più difficile da comprendere, oscurando la logica reale del componente.

Considerate queste osservazioni, si è optato per evitare l'impiego del corrente hook all'interno di questo progetto, specialmente in vista dell'aggiornamento a **React 19**, che porterà con sé il nuovo React Compiler. Questo compilatore consente di eliminare del tutto la necessità di utilizzare hook come il sopracitato `useMemo`. Ciò avviene perché, sfruttando la sua conoscenza di JavaScript e delle regole di React, il compilatore memoizza automaticamente valori o gruppi di valori all'interno dei componenti e degli hook. Qualora vengano rilevate violazioni delle regole, il compilatore salterà automaticamente quei componenti o hook specifici, continuando a elaborare in modo sicuro il resto del codice.

3.4 Next.js

Next.js [9] è un *framework*¹ costruito sopra React, progettato per la creazione di applicazioni web full-stack. Mentre React si concentra principalmente sulla costruzione di interfacce utente tramite componenti, Next.js estende queste funzionalità offrendo un insieme di strumenti e ottimizzazioni che semplificano lo sviluppo di applicazioni complesse. Ad esempio, Next.js supporta il rendering lato server (SSR) e la generazione di siti statici (SSG), che migliorano le prestazioni e l'indicizzazione sui motori di ricerca (SEO).

3.4.1 Rendering

In React, il rendering delle pagine avviene completamente nel browser tramite Client-Side Rendering (CSR). Quando un utente richiede una pagina, il server invia un documento HTML minimale, che di solito è molto scarno e non contiene il contenuto della pagina. Solo dopo che il codice JavaScript è stato eseguito nel browser, viene effettuata una chiamata per recuperare i dati, generando così il contenuto della pagina. Questo approccio può portare a tempi di caricamento iniziali più lunghi, poiché l'utente deve attendere il caricamento e l'esecuzione del codice JavaScript prima di vedere il contenuto.

Next.js ottimizza le funzionalità di React grazie al **pre-rendering** di ogni pagina. A differenza del tradizionale rendering effettuato tramite JavaScript lato client, Next.js genera in anticipo il codice HTML per ciascuna pagina. Questa strategia consente di migliorare le performances e l'esperienza utente. In particolare, Next.js implementa due forme di pre-rendering per affrontare le problematiche tipiche del rendering in React:

- **Static Generation (SSG)**: il codice HTML delle pagine che utilizzano la generazione statica viene generato una sola volta durante il processo di compilazione e riutilizzato per ogni richiesta successiva. Questa caratteristica migliora notevolmente le prestazioni, poiché il contenuto è pronto per essere servito immediatamente. È buona norma adottare la generazione statica quando possibile, soprattutto per le pagine che non richiedono aggiornamenti frequenti o dati in tempo reale, in quanto possono essere pre-renderizzate.

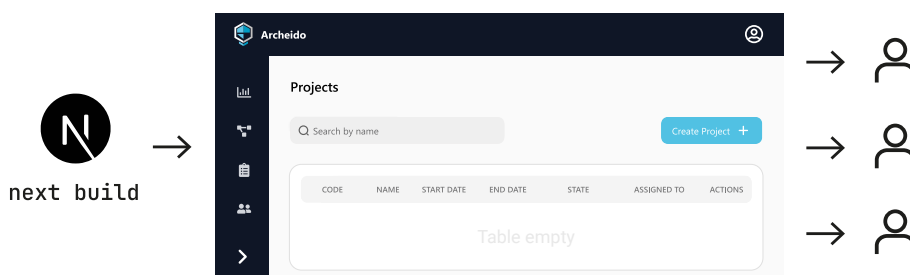


Figura 3.2: Static Generation

¹Un insieme di strumenti, librerie e regole che forniscono una struttura di base per lo sviluppo di software, facilitando il lavoro dei programmatori.

- **Server-side Rendering (SSR o Dynamic Rendering):** il rendering lato server consente di generare dinamicamente il codice HTML per ogni pagina al momento della richiesta. Questo approccio è cruciale per le pagine che richiedono dati in tempo reale o informazioni che cambiano frequentemente. Con il Server-Side Rendering, ogni volta che un utente richiede una pagina, il server elabora le informazioni necessarie e genera il codice HTML. Una volta completato il rendering, l'HTML viene inviato al browser, assicurando che l'utente riceva immediatamente una pagina completamente renderizzata e aggiornata.

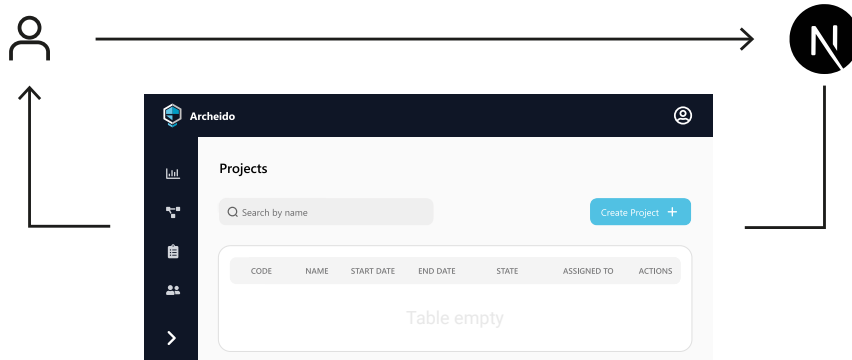
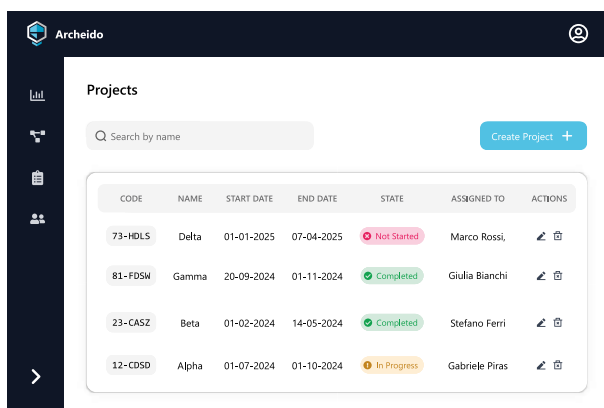


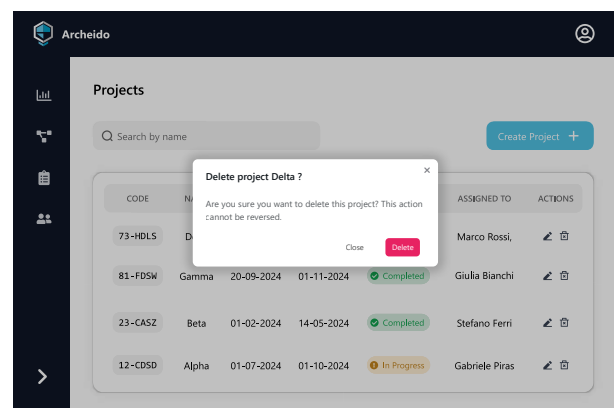
Figura 3.3: Server-side Rendering

Come già descritto in precedenza, quando una pagina web è generata attraverso il rendering lato server, il server invia al browser un documento HTML già pronto, completo di contenuto e struttura. Tuttavia, affinché l'applicazione diventi interattiva e possa rispondere agli eventi (come click, input dell'utente, ecc.), è necessario che il codice JavaScript venga caricato e eseguito nel browser.

Dunque, una volta che il codice JavaScript è disponibile, viene eseguito l'**hydration**, ossia un processo che comporta l'associazione dei gestori di eventi e l'attivazione della logica dei componenti, trasformando l'HTML statico in un'applicazione interattiva.



(a) **JavaScript non ancora caricato:** facendo clic sul pulsante di eliminazione di un progetto, non si verifica alcuna azione.



(b) **JavaScript caricato:** facendo clic sul pulsante di eliminazione di un progetto, si apre una finestra modale per confermare l'eliminazione.

Figura 3.4: Prima e dopo l'hydration

Sebbene l'hydration offra notevoli vantaggi in termini di prestazioni e SEO, durante la realizzazione di questo progetto si sono verificati diversi errori dovuti ad esso, evidenziati specialmente durante la fase di testing. Questi problemi erano principalmente dovuti a incongruenze tra l'HTML generato dal server e quello che il client si aspettava dopo l'esecuzione del codice JavaScript. In particolare, un componente utilizzava dati che cambiavano frequentemente, causando stati non sincronizzati tra server e client. La soluzione a questo problema è consistita nel trasferire la logica relativa agli stati problematici e non sincronizzati all'interno di un contesto, permettendo così di mantenere uno stato coerente tra server e client.

3.4.2 App routing

Il routing in React gestisce la navigazione tra diverse pagine o componenti all'interno di un'applicazione. Utilizza librerie come **React Router** per definire percorsi e associare URL specifici a componenti. Quando un utente naviga a un URL, React Router mostra il componente corrispondente senza ricaricare l'intera pagina, permettendo un'esperienza utente fluida e dinamica. In Next.js, il sistema di routing è reso notevolmente più semplice e intuitivo, poiché si basa sulla **struttura del file system**. Le cartelle rappresentano i percorsi nell'applicazione web, e ogni cartella dedicata a una pagina deve contenere un file chiamato `page.tsx`², che ne include il contenuto. La cartella `app` funge da radice del progetto. Ad esempio, nel CRM è stata creata una cartella `projects`, al cui interno si trova il file `page.tsx`. In questo file, è stata definita ed esportata una funzione che restituisce un elemento TSX. Questo componente verrà visualizzato quando si visiterà il percorso `./projects` nella pagina web.

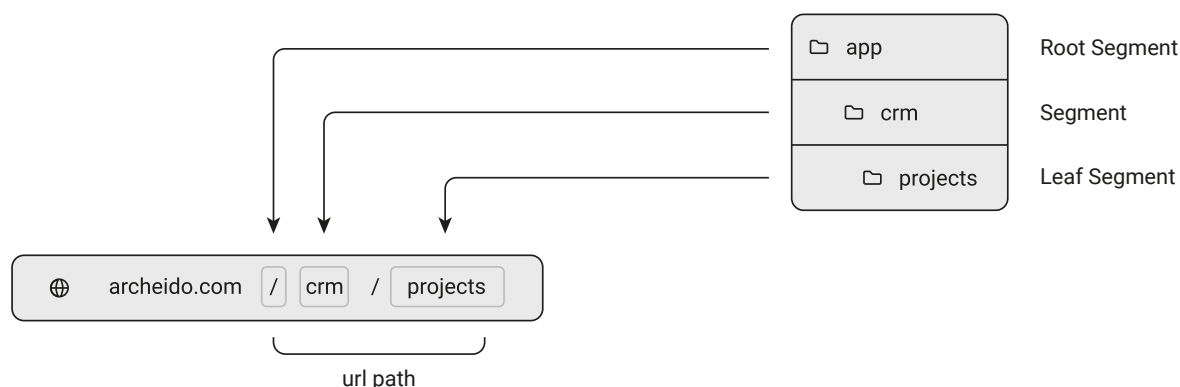


Figura 3.5: App routing [9] di Next.js

La gestione del routing è stata implementata utilizzando un hook specifico chiamato `useRouter`. Questo hook, fornito da Next.js, permette di accedere al router e gestire la navigazione all'interno dell'applicazione. Per utilizzarlo, è sufficiente importarlo con la seguente istruzione:

```
import { useRouter } from 'next/router';
```

Dopodiché, può essere dichiarato all'interno del componente con la seguente sintassi:

²L'estensione è `.tsx` perché si sta usando TypeScript. Con JavaScript sarebbe `.jsx`.

```
const router = useRouter();
```

Utilizzando la costante `router`, è possibile accedere facilmente a numerose informazioni e funzionalità legate alla navigazione. In particolare, si possono ottenere:

- Il **percorso attuale**: con `router.pathname`, che restituisce il percorso della route corrente senza includere i parametri della query. Questo può essere utile per sapere quale pagina è attualmente visualizzata e per applicare logiche condizionali basate sulla posizione dell'utente nell'applicazione;
- I **parametri della query string**: tramite `router.query`, che fornisce un oggetto contenente i parametri della query presenti nell'URL. Questa funzionalità è utile per leggere dati dinamici dalla route, come identificatori o filtri passati tramite l'URL;
- **Funzioni per navigare a nuove route e gestione della cronologia del browser**: ad esempio, `router.push()` consente di navigare verso un nuovo percorso specificato, permettendo reindirizzamenti programmatici all'interno dell'applicazione. Un'alternativa è `router.replace()`, che, a differenza di `push()`, sostituisce la voce corrente nella cronologia del browser, evitando di aggiungerne una nuova. Infine, `router.back()` offre la possibilità di tornare alla pagina precedente, simulando il comportamento del pulsante "indietro" del browser.

Nel contesto della web app sviluppata, questo hook è stato usato molte volte nel seguente modo:

```
1 import { useRouter } from 'next/router';
2 const MyComponent = () => {
3   const router = useRouter();
4   const handleNavigation = () => {
5     router.push('/new-page'); // Reindirizza a "/new-page"
6   };
7   return (
8     <div>
9       <p>Current path: {router.pathname}</p>
10      <p>Query params: {router.query}</p>
11      <button onClick={handleNavigation}>Go to new page</button>
12    </div>
13  );
14 };
```

Listing 3.8: `useRouter` per la navigazione tra URL

Un altro aspetto chiave di Next.js è la possibilità di implementare facilmente il **routing dinamico**. Questa funzionalità è stata adottata nella pagina `./projects` dove, selezionando un progetto, si accede ai relativi dettagli, con un reindirizzamento a una pagina avente un URL del tipo `/projects/ID_PROGETTO`. Naturalmente, ogni progetto ha un URL univoco, il che rende funzionale la navigazione e il caricamento delle informazioni specifiche relative a ciascun progetto. Per ottenere ciò, all'interno della cartella `/projects` è stata creata un'ulteriore cartella denominata `[projectId]`. Questa convenzione di denominazione, utilizzando le parentesi quadre, indica a Next.js che il file al suo interno rappresenta una *route dinamica*. All'interno di questa cartella, è presente un file chiamato `page.tsx`, che funge da componente per la pagina dei dettagli del progetto.

```

1 const page = ({ params }: { params: { projectId: string } }) => {
2   const [selectedProject, setSelectedProject] = useState<Project>({});
3
4   useEffect(() => {
5     getProject(params.projectId).then((selectedProject: any) => {
6       setSelectedProject(selectedProject as Project);
7     });
8   }, []);
9
10  return <div>Selected project: {params.slug}</div>
11 };

```

Listing 3.9: Routing dinamico con Next.js

Quando il componente viene montato, il codice esegue un effetto collaterale utilizzando l'hook `useEffect`. Questo effetto invoca la funzione `getProject`, passando il `projectId` come argomento per recuperare i dettagli del progetto corrispondente. Una volta ottenuto il progetto, lo stato `selectedProject` viene aggiornato con i dati ricevuti. Infine, il componente restituisce un elemento `<div>` che visualizza un messaggio con il slug presente in `params`. Questo messaggio indica quale progetto è stato selezionato.

3.5 Next UI e Tailwind CSS

3.5.1 Next UI

In questo lavoro si è deciso di utilizzare Next UI [5], una libreria di componenti UI progettata specificamente per Next.js. Essa offre una serie di componenti predefiniti (come bottoni, calendari, modali, cards, chips, etc.) e stilizzati che semplificano la creazione di interfacce utente. Inoltre, è possibile importare solo i componenti necessari e non tutta la suite, riducendo il peso complessivo dell'applicazione. Ad esempio, se si volesse importare un componente relativo ad un calendario, sarebbe sufficiente scrivere il relativo codice:

```

1 import {Calendar} from "@nextui-org/react";
2
3 const date = parseDate("2020-02-03");
4
5 export default function App() {
6   return (
7     <Calendar
8       aria-label="Date"
9       defaultValue={date}
10    />
11   );
12 }

```

Listing 3.10: Uso di un componente NextUI

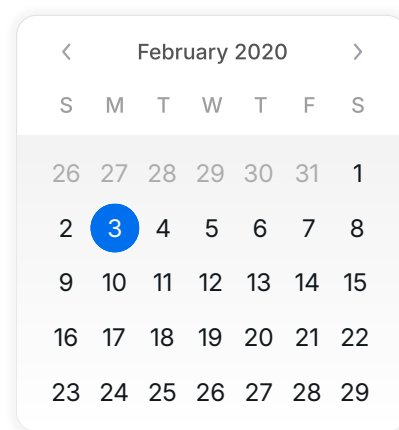


Figura 3.6: Calendario di NextUI

3.5.2 Tailwind CSS

Un'altra libreria utilizzata per la progettazione del CRM di Archeido è Tailwind CSS: un framework CSS *utility-first* che permette agli sviluppatori di creare interfacce utente in modo rapido e flessibile. Il termine *utility-first* si riferisce a un approccio relativo alla progettazione delle interfacce utente che prevede l'uso di classi CSS predefinite per applicare gli stili direttamente agli elementi HTML, anziché creare classi personalizzate o definire stili specifici per i componenti. Un esempio è il seguente:

```

1 <main className = "flex flex-col h-screen">
2   <NavBar className = "w-screen" signOut={signOut} />
3   <div className = "flex flex-row h-full">
4     <div className = "col-auto">
5       <SideBar />
6     </div>
7     <div className = "col-auto p-8 grow">{children}</div>
8   </div>
9 </main>

```

Listing 3.11: Parte del file `app.tsx` del CRM

Questo codice rappresenta la struttura di un componente React che definisce la disposizione generale delle pagine del progetto. L'elemento `<main>` ha una classe che lo rende un contenitore flessibile e verticale, con un'altezza che occupa l'intero schermo. All'interno di questo contenitore, è presente una barra di navigazione `<NavBar>` che si estende per tutta la larghezza della pagina e riceve una proprietà `signOut` per la gestione del logout. Sotto la barra di navigazione, vi è un contenitore diviso orizzontalmente in due sezioni. La prima sezione contiene una barra laterale `<SideBar>` che occupa spazio in base al suo contenuto, mentre la seconda sezione si estende e cresce per occupare il resto dello spazio disponibile. Quest'ultima include un'area di contenuto, rappresentata da `children`, che è un segnaposto per i contenuti passati come figli al componente principale.

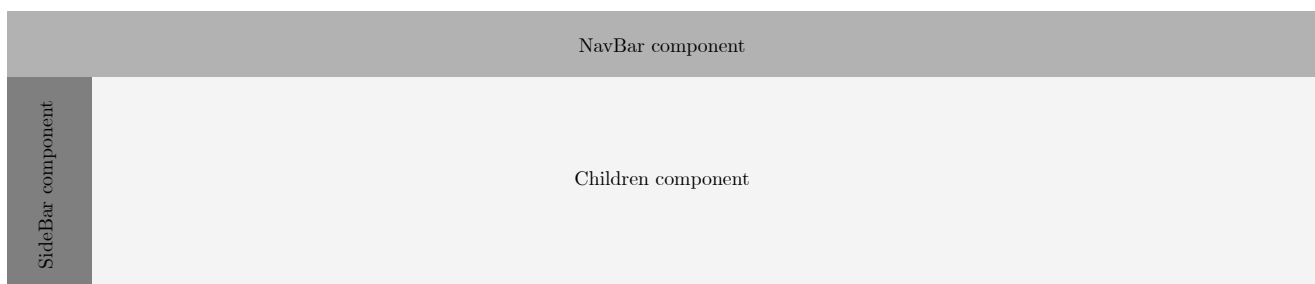


Figura 3.7: Layout generico delle pagine, ottenuto utilizzando Tailwind CSS

3.6 AWS: Amazon Web Service

Nel contesto del CRM per la gestione dei progetti, è stato scelto Amazon Web Services (AWS) come piattaforma di hosting e gestione delle funzionalità del sistema. AWS offre una vasta gamma di servizi cloud che si integrano in modo efficiente per garantire scalabilità, sicurezza e affidabilità.

Per il deployment e la gestione del front-end del CRM, è stato impiegato **AWS Amplify**, un servizio che permette di ospitare applicazioni web con un'integrazione continua (CI/CD). Amplify è stato inizialmente collegato al repository **BitBucket**, il quale contiene il codice sorgente del progetto. Ogni volta che vengono effettuate modifiche al codice nel repository, Amplify esegue automaticamente un nuovo deployment.

Per la gestione dell'autenticazione degli utenti, è stato invece utilizzato **AWS Cognito**, il quale offre una gestione sicura delle credenziali, supportando funzionalità come la multi-factor authentication (MFA) e la rotazione automatica delle chiavi. Il sistema di login del CRM è progettato in modo tale da non permettere la registrazione autonoma degli utenti. Infatti, gli amministratori sono gli unici autorizzati a creare nuovi account e fornire le credenziali di accesso. Questo avviene direttamente dalla dashboard di Cognito, in quanto nel CRM non è stata sviluppata una funzionalità per rendere possibile l'invito di nuovi utenti, essendo uno scenario che trova impiego in pochi casi. Per implementare l'autenticazione, AWS mette a disposizione un componente **Authenticator**, il quale prende la prop `hideSignUp`, che permette di nascondere l'opzione per la registrazione.

```
1 const AuthenticatedLayout = ({ children }: any) => {  
2   return (  
3     <Authenticator hideSignUp components={components}>  
4       {(props) => <App {...props}>{children}</App>  
5     </Authenticator>  
6   );  
7 };
```

Listing 3.12: Parte del file `AuthenticatedLayout.tsx` del CRM



Email

Password


 
[Sign in](#)
[Forgot your password?](#)

Figura 3.8: Pagina di login

Capitolo 4

Descrizione dei requisiti

4.1 Introduzione

In questo capitolo si descriveranno i requisiti del sistema CRM per la gestione dei progetti, sviluppato durante il tirocinio. Verranno utilizzati diversi strumenti di modellazione per rappresentare i requisiti funzionali e non funzionali del sistema, tra cui diagrammi e pseudocodice per i casi più rilevanti.

4.2 Definizione dei requisiti del sistema

Il sistema CRM è stato progettato per supportare la gestione di progetti aziendali, offrendo funzionalità che spaziano dalla creazione e monitoraggio dei progetti, alla gestione degli utenti. I principali requisiti funzionali delineati durante la fase di raccolta dei requisiti includono:

- **Gestione utenti e ruoli:** gli utenti con ruolo `ADMIN` possono accedere e apportare modifiche a tutte le sezioni della dashboard: è loro compito assegnare un progetto a uno o più project manager. Gli utenti con ruolo `PROJECT_MANAGER` possono visionare, modificare e/o cancellare solo i progetti a cui sono stati assegnati da un utente con ruolo `ADMIN`. Inoltre, la sezione utenti, disponibile al link `./users`, è visibile solo ad utenti con ruolo di amministratore.
- **Accesso riservato ad utenti invitati:** un nuovo utente che desidera utilizzare il CRM deve ricevere un invito via e-mail contenente una password temporanea, inviato da un amministratore. Al primo accesso, l'utente sarà obbligato a cambiare la password temporanea.
- **Creazione di un progetto:** l'utente può creare un progetto, specificando dettagli come `codice`, `nome`, `data_di_inizio`, `data_di_fine`, `stato` (il quale può assumere i valori di `non_iniziato`, `in_corso` o `completato`).
- **Modifica di un progetto:** l'utente può modificare un progetto esistente, cambiandone i campi sopracitati.
- **Cancellazione di un progetto:** l'utente può cancellare un progetto.

4.2.1 Gestione dei ruoli degli utenti

Per implementare le regole di visibilità in base ai ruoli degli utenti, è stato aggiunto un campo custom dalla dashboard di Cognito: tale campo è stato definito come `ROLE` e può assumere solo i due valori sopracitati. Un utente che accede al CRM, viene memorizzato in un Context ¹ globale, il quale “wrappa” l’intera applicazione. Tale contesto può essere poi ottenuto in ogni componente della web app.

Un esempio di utilizzo si può trovare nel componente `SideBar`. Una sidebar è un pannello laterale in un’interfaccia utente che offre accesso rapido finalizzato alla navigazione. In questo caso, il contenuto della sidebar varia in base al ruolo dell’utente attualmente loggato: se l’utente è un admin, ha accesso a tutte le sezioni: Performance, Projects, Orders e Users. Al contrario, se l’utente è un project manager, la sezione Users non sarà visibile. Per gestire le sezioni visibili nella sidebar, si utilizza un array di oggetti, il quale contiene l’insieme di sezioni visibili dall’utente con meno permessi, quindi project manager.

```

1  const menusFields = [
2    {
3      title: "Performances",
4      src: "performances",
5      icon: <FaRegChartBar className = "scale-150" />,
6      visibility: "ALL",
7      gap: false
8    },
9    {
10     title: "Projects",
11     src: "projects",
12     icon: <FaProjectDiagram className = "scale-150" />,
13     visibility: "ALL",
14     gap: false
15   },
16   {
17     title: "Orders",
18     src: "orders",
19     icon: <FaClipboardList className = "scale-150" />,
20     visibility: "ALL",
21     gap: false
22   },
23 ];

```

Listing 4.1: Campi default visibili nella sidebar

Viene successivamente estratto il ruolo dell’utente loggato dal contesto dell’applicazione:

```

1  const user = useContext(AuthContext);
2  const role = user["custom:role"];

```

Listing 4.2: Parte del file `AuthenticatedLayout.tsx` del CRM

¹è uno strumento (un oggetto) che consente di condividere dati (come lo stato o le funzioni) tra componenti senza dover passare esplicitamente le props attraverso ogni livello dell’albero dei componenti. Ogni volta che un campo del contesto varia, la schermata viene ri-renderizzata.

In questo modo, la costante `role` contiene la stringa "ADMIN" o la stringa "PROJECT_MANAGER". Tale valore, viene usato in seguito per decidere se mostrare o meno la sezione Orders:

```

1 role === "ADMIN"
2   ? [...Menus, {
3     title: "Users",
4     src: "users",
5     icon: <FaUserFriends className = "scale-150" />,
6     visibility: "ADMIN",
7     gap: false
8   }]
9   : Menus;
```

Listing 4.3: Verifica ruolo

Questo codice verifica se il ruolo dell'utente che ha effettuato il login è "ADMIN". Se è così, crea una nuova lista `Menus` che include gli elementi esistenti, aggiungendo un nuovo oggetto per la voce di menu "Users". Contrariamente, se l'utente non è un admin, la lista `Menus` rimane invariata.

L'uso dell'operatore di spread `...Menus` garantisce che la lista originale non venga modificata, mantenendo l'approccio immutabile tipico della programmazione funzionale.

4.2.2 Operazioni di CRUD

Le operazioni CRUD sono le quattro azioni fondamentali per la gestione dei dati in un database: Create (creare nuovi dati), Read (leggere o visualizzare i dati), Update (aggiornare i dati esistenti) e Delete (eliminare i dati). Il CRM supporta tutte e quattro queste operazioni, implementate seguendo la guida [6] sulla documentazione di Amplify. Innanzitutto è stato creato il **modello dati** reattivo ai progetti

```

1 const schema = a
2 .schema({
3   Project: a.model({
4     code: a.string(),
5     name: a.string(),
6     startdate: a.string(),
7     enddate: a.string(),
8     state: a.string(),
9     assignedTo: a.string(),
10  }),
11 })
```

Listing 4.4: Parte del file `amplify/data/resource.ts`

Ogni volta che si definisce un modello con `a.model()`, vengono automaticamente create le seguenti risorse nel cloud:

- Una tabella DynamoDB per memorizzare i record.
- API per query che permettono di creare, leggere (list/get), modificare ed eliminare i record.
- Campi `createdAt` e `updatedAt` per tracciare quando ogni record è stato creato o aggiornato.
- API in tempo reale per iscriversi agli eventi di creazione, aggiornamento ed eliminazione dei record.

Una volta completata questa operazione, è stata creata una cartella `api` nella root del progetto. Questa cartella ha lo scopo di raggruppare tutti i file relativi alle chiamate API ed è stata pensata per mantenere il progetto ben organizzato, estraendo tutte le funzioni che effettuano chiamate al database. Al suo interno, è stata generata un'altra cartella denominata `Project`, che a sua volta contiene il file `endpoints.ts`. Quest'ultimo file si occupa di definire ed esportare le funzioni relative alle operazioni CRUD.

Le prime funzioni definite per gestire le operazioni di CRUD sono `getProject` e `listProject`. La funzione `getProject` accetta come argomento un ID di tipo stringa, effettua una richiesta al database e restituisce l'oggetto `Project` corrispondente. La funzione `listProject`, invece, recupera e restituisce l'intera lista dei progetti presenti nel database.

```
1 const getProject = async (id: string) => {
2   const { data: selectedProject }: Project = await client.models.Project.get({
3     id
4   });
5   return selectedProject;
6 };
```

Listing 4.5: funzione `getProject`

```
1 const listProject = async () => {
2   const { data: projects } = await client.models.Projects.list();
3   return projects;
4 };
```

Listing 4.6: funzione `listProject`

Graficamente, le due funzioni menzionate sono responsabili dell'estrazione di tutti i dati necessari dal database per popolare la tabella presente all'URL `/projects`, mostrata nell'immagine sottostante.

CODE	NAME	START DATE	END DATE	STATE	ASSIGNED TO	ACTIONS
73-HDLS	Delta	01-01-2025	07-04-2025	Not Started	Marco Rossi, Giulia Bianchi	
81-FDSW	Gamma	20-09-2024	01-11-2024	Completed	Alessandro Verdi	
23-CASZ	Beta	01-02-2024	14-05-2024	Completed	Federica Marconi, Stefano Ferri	
12-CDSD	Alpha	01-07-2024	01-10-2024	In Progress	Gabriele Piras	

Figura 4.1: Pagina dei progetti

A questo punto, se si clicca sul pulsante “Create Project” viene caricato l'URL `./projects/create`, il quale visualizza un modulo vuoto per l'inserimento dei dati del progetto. Per la realizzazione e la

gestione di questo modulo, è stata utilizzata la libreria **Formik**. Dopo aver compilato tutti i campi del modulo, premendo il pulsante “Create” viene invocata la funzione `createProject`, che riceve come input l’oggetto `Project` generato tramite il riempimento dei campi e lo invia al database per la sua creazione.

```
1 const createProject = async (projectToCreate: Project) => {  
2   await client.models.Project.create(project);  
3 };
```

Listing 4.7: funzione `createProject`

La tabella dei progetti in `./projects` presenta una colonna denominata **ACTIONS**, in cui sono visibili due icone. Quando si preme l’icona a forma di cestino, si apre una modale di *danger* che consente di confermare l’eliminazione del progetto. D’altro canto, se si seleziona l’icona a forma di matita, viene caricato l’URL della pagina del progetto con la seguente forma: `./projects/project_id`. In quest’ultima è possibile effettuare modifiche o procedere all’eliminazione del progetto.

Per cancellare un progetto, si utilizza la funzione `deleteProject`, che richiede come argomento l’oggetto `Project` da eliminare, e tramite il suo ID, ne esegue la cancellazione:

```
1 const deleteProject = async (projectToDelete: Project) => {  
2   const { id } = projectToDelete;  
3   await client.models.Project.delete({ id });  
4 };
```

Listing 4.8: funzione `deleteProject`

L’aggiornamento di un progetto esistente viene eseguito mediante la funzione `updateProject`, che accetta come parametro un oggetto `Project` e aggiorna il database con i nuovi dati forniti:

```
1 const updateProject = async (projectToUpdate: Project) => {  
2   await client.models.Project.update(projectToUpdate);  
3 };
```

Listing 4.9: funzione `updateProject`

The screenshot shows the 'Archeido' application interface. On the left is a dark sidebar with navigation icons. The main content area is titled 'Edit' and contains a form for editing a project. The form has the following fields:

- Name of the Project***: Text input with value 'Gamma'.
- Code of the Project***: Text input with value 'B1-FDSW'.
- Start Date***: Date picker showing '20/09/2024'.
- End Date***: Date picker showing '01/11/2024'.
- State***: Dropdown menu with value 'Completed'.
- Assigned To***: Dropdown menu with value 'Alessandro Verdi'.

At the top right of the form are two buttons: 'Update' (blue) and 'Delete' (red).

Figura 4.2: Pagina dei progetti

4.3 Diagrammi

4.3.1 Diagramma dei casi d'uso (Use Case Diagram)

Il diagramma nella figura mostra il sistema CRM con i diversi attori coinvolti, sia umani che esterni. Nello specifico:

- gli **attori umani** includono “User”, “Project Manager” e “Admin” ;
- i **servizi esterni** sono “AWS DynamoDB” e “AWS Cognito”.

Gli attori umani interagiscono con il sistema principalmente per la gestione dei progetti e degli utenti. Le operazioni di gestione dei progetti ne comprendono la creazione, lettura, aggiornamento e cancellazione, raggruppate sotto il caso d'uso generale “Manage Projects”.

In modo simile (ma limitato agli attori Admin), la gestione degli utenti è modellata attraverso il caso d'uso “Manage Users”, che include funzionalità come invitare nuovi utenti, modificare i ruoli e cancellare utenti già registrati.

Il processo di autenticazione è diviso in due casi d'uso principali: il “First Login” per il primo accesso e “Second + Login” per i successivi. Questi casi d'uso si estendono con la gestione degli errori e la verifica delle password. Inoltre, è presente un caso d'uso separato per il cambio della password, “Change Password”. Il sistema interagisce con “AWS Cognito” per gestire l'autenticazione e con “AWS DynamoDB” per eseguire operazioni sui dati dei progetti.

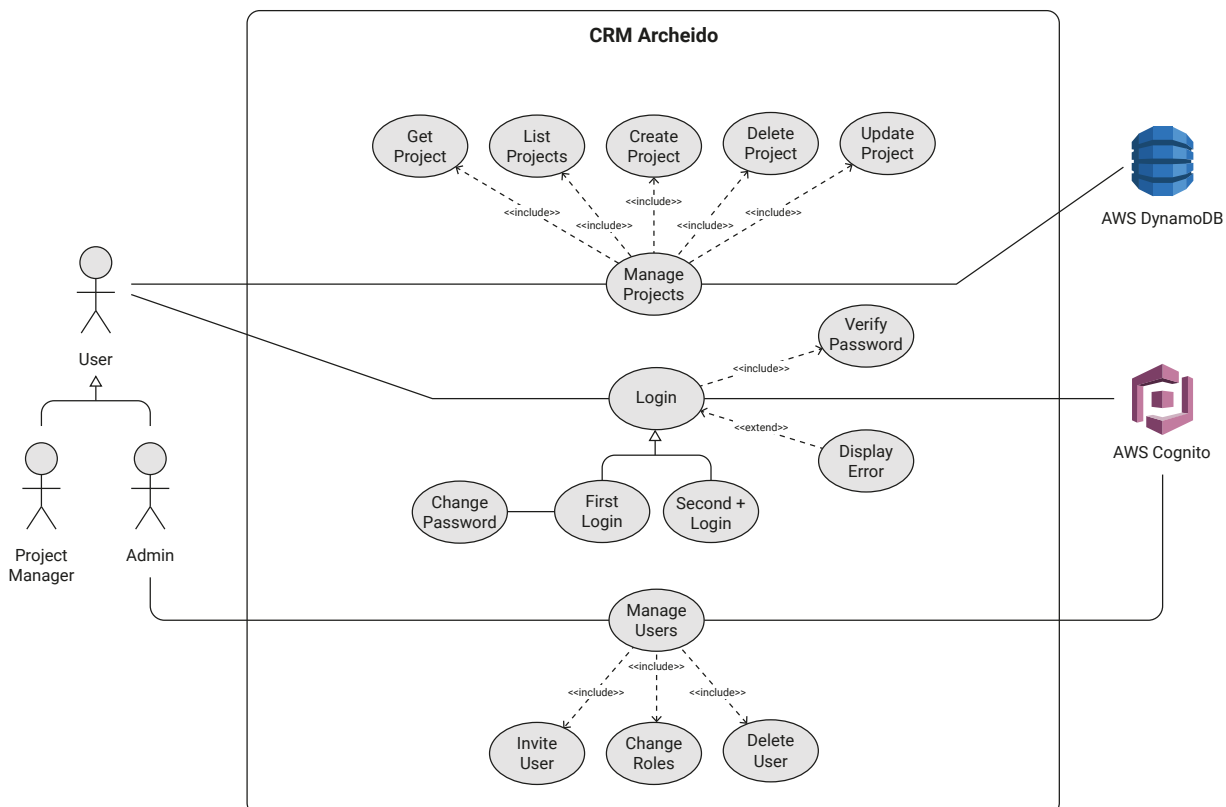


Figura 4.3: Use Case Diagram

4.3.2 Sequence diagram

Il diagramma di sequenza illustra le interazioni tra l'utente, il sistema di autenticazione Cognito, la dashboard del CRM e il database DynamoDB nella gestione dei progetti. Inizia con l'utente che inserisce le credenziali (e-mail e password) nella pagina di login, che inoltra i dati al pool di utenti Cognito per verificarne la validità. Se l'e-mail è presente e la password corretta, il client riceve una risposta di successo, confermando l'accesso; in caso contrario, viene visualizzato un messaggio di errore. Dopo un accesso riuscito, l'utente naviga alla pagina dei progetti (`/projects`), dove può visualizzare l'elenco di tutti i progetti disponibili. La dashboard del CRM richiede i dati al database DynamoDB tramite la funzione `listProject`, e, una volta ottenuta la risposta, i progetti vengono mostrati in una tabella. Quando l'utente desidera creare, modificare o eliminare un progetto, invia una richiesta tramite la dashboard. Per creare un nuovo progetto o cancellarne uno esistente, vengono invocate rispettivamente le funzioni `createProject` o `deleteProject`. Dopo l'esecuzione di queste operazioni, la dashboard aggiorna la lista dei progetti per mostrare i dati più recenti. Analogamente, se l'utente sceglie di aggiornare un progetto, viene invocata la funzione `updateProject` e i dati modificati vengono salvati nel database, con la lista dei progetti che si ricarica per riflettere le modifiche.

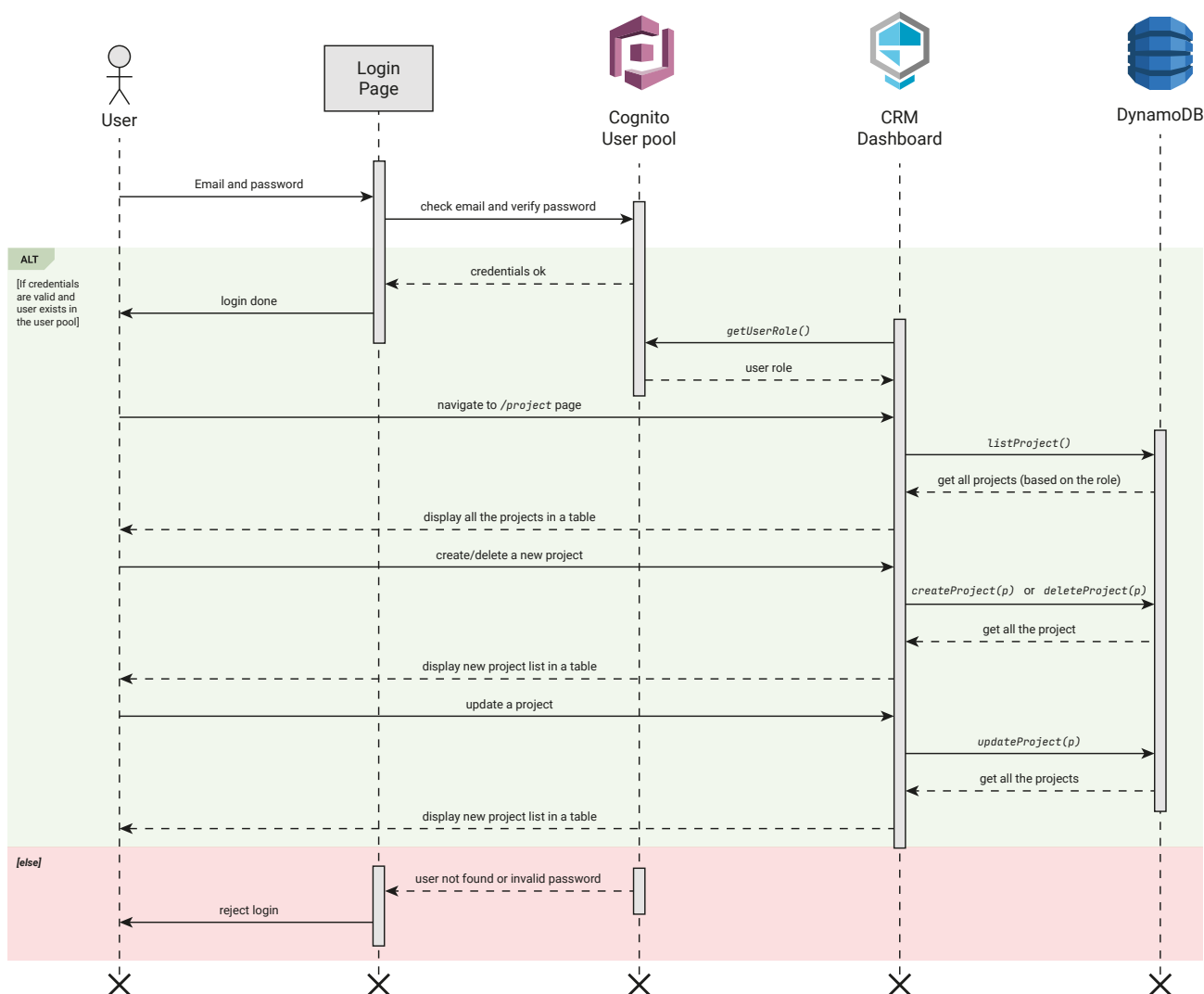


Figura 4.4: Sequence Diagram per il login e le operazioni CRUD sui progetti

Capitolo 5

Testing

Le attività di testing si sono concentrate principalmente sulla verifica delle funzionalità principali dell'applicazione e sulla gestione degli accessi in base ai ruoli definiti, assicurando che ogni componente critico funzionasse correttamente e che l'assegnazione dei permessi rispettasse i criteri di sicurezza e operatività previsti. Tuttavia, il focus limitato a questi aspetti ha fatto emergere alcune lacune significative, in particolare per quanto riguarda l'accessibilità e l'inclusività dell'applicazione. Questi aspetti, sebbene non compromettano l'efficienza di base, indicano la necessità di una riflessione più approfondita e di un'analisi dettagliata, che verrà discussa in un secondo momento per evidenziare come migliorare l'esperienza complessiva degli utenti e garantire che l'applicazione sia fruibile da persone con esigenze diverse.

5.1 Tipologie di test condotti

Diversi tipi di test sono stati implementati per valutare le funzionalità, l'affidabilità e l'usabilità del sistema in modo progressivo e complementare. Dapprima, il testing unitario ha consentito la verifica puntuale delle singole funzioni e componenti del codice, garantendo la loro operatività in un contesto isolato.

Successivamente, si è effettuato il testing di integrazione per analizzare la cooperazione tra i vari moduli, verificando l'interazione tra frontend e backend, nonché l'affidabilità delle chiamate ai servizi esterni come `DynamoDB` e `AWS Cognito`.

Infine, il testing di accettazione ha coinvolto gli utenti finali, admin e project managers, fornendo un feedback per confermare che il sistema rispondesse alle esigenze aziendali e offrisse un'esperienza user-friendly.

5.1.1 Testing unitario

Il testing unitario è stato utilizzato per testare singole funzioni e componenti del sistema in un ambiente controllato, verificando che ciascun elemento funzionasse correttamente e in modo indipendente dagli altri. Ogni funzione, come quelle di creazione, lettura, aggiornamento e cancellazione dei progetti, è stata verificata in isolamento per assicurare che operasse come previsto e che le risposte fossero accurate e coerenti. In secondo luogo, il testing unitario ha reso più semplice il refactoring del codice, garantendo che i miglioramenti e le modifiche non compromettessero le funzionalità esistenti. La copertura dei test

ha contribuito a costruire una base solida per i successivi livelli di test, come il testing di integrazione, dove si è verificata l'interazione tra più componenti del sistema per confermare che collaborassero senza problemi.

5.1.2 Testing di integrazione

Il testing di integrazione è stato condotto per verificare l'interazione tra diversi moduli del sistema, in particolare tra il frontend e il backend. Questo ha incluso test per garantire che le chiamate API al database `DynamoDB` restituissero i risultati attesi e che il sistema di autenticazione `AWS Cognito` funzionasse correttamente.

5.1.3 Testing di accettazione

Il testing di accettazione è stato realizzato coinvolgendo direttamente gli utenti finali nel processo per simulare scenari di utilizzo reale e ottenere un feedback autentico sull'efficacia del sistema. Gli utenti hanno avuto l'opportunità di testare le funzionalità principali del CRM, valutando sia la correttezza tecnica delle operazioni, sia la fluidità e l'intuitività dell'interfaccia. Durante queste sessioni, sono stati verificati aspetti come la facilità di navigazione, la chiarezza dei flussi di lavoro e i tempi di esecuzione delle varie operazioni. Inoltre, questo tipo di testing ha garantito che il CRM rispondesse pienamente ai requisiti aziendali, rispettando i requisiti funzionali, definiti nelle prime fasi del progetto, e di performance.

5.2 Casi di test

I casi di test sono stati progettati con l'obiettivo di coprire in modo completo e dettagliato tutte le funzionalità principali del sistema, oltre a garantire la conformità ai requisiti funzionali stabiliti durante la fase di pianificazione iniziale del progetto. Questi casi di test sono stati sviluppati per esplorare scenari reali di utilizzo in modo da assicurare la robustezza e l'affidabilità dell'applicazione. L'attenzione è stata rivolta a verificare che ogni aspetto del sistema rispondesse correttamente agli input previsti e non, mantenendo un comportamento coerente e sicuro. Di seguito vengono riportati alcuni esempi significativi di casi di test implementati:

5.2.1 Test di creazione progetto

- **Obiettivo:** verificare che un progetto possa essere creato con successo.
- **Passi:**
 1. Accedere come utente con ruolo `ADMIN`.
 2. Navigare alla pagina di creazione del progetto.
 3. Compilare il modulo con dati validi.
 4. Cliccare sul pulsante "Create".
- **Risultato Atteso:** il progetto viene creato correttamente e compare nel relativo elenco dei progetti. Inoltre, anche dopo il refresh della pagina o dopo aver effettuato il logout e successivamente il login, il progetto rimane visibile e preservato nella lista, confermando la persistenza dei dati.

5.2.2 Test di cancellazione progetto attraverso la pagina dei progetti

- **Obiettivo:** Verificare che un progetto possa essere cancellato correttamente direttamente dall'elenco dei progetti, senza la necessità di accedere alla pagina di dettaglio del progetto stesso.
- **Passi:**
 1. Accedere come utente con ruolo **ADMIN**.
 2. Navigare alla pagina contenente la tabella con l'elenco dei progetti.
 3. Selezionare un progetto esistente dall'elenco e, facendo clic sull'icona a forma di cestino nella colonna delle azioni, verificare che si apra una finestra modale di conferma per la cancellazione.
 4. Cliccare sul pulsante "Delete" nella modale.
- **Risultato Atteso:** il progetto viene rimosso con successo dall'elenco dei progetti. Inoltre, anche dopo un refresh della pagina o dopo aver effettuato il logout e poi il login, il progetto non deve più comparire nella lista.

5.2.3 Test di cancellazione progetto attraverso la pagina del dettaglio

- **Obiettivo:** verificare che un progetto possa essere cancellato correttamente dalla pagina del dettaglio di tale progetto.
- **Passi:**
 1. Accedere al sistema con un account avente ruolo **ADMIN**.
 2. Navigare alla pagina contenente la tabella con l'elenco dei progetti.
 3. Selezionare il progetto che si desidera eliminare, cliccando sull'icona a forma di matita nella colonna delle azioni per accedere alla pagina di dettaglio.
 4. Attendere il caricamento della pagina e, una volta completato, premere il pulsante "Delete". Assicurarsi che appaia una finestra modale di conferma per la cancellazione.
 5. Confermare l'azione cliccando nuovamente sul pulsante "Delete" nella modale di conferma.
- **Risultato Atteso:** il progetto viene rimosso con successo dall'elenco. Inoltre, anche dopo un refresh della pagina o dopo aver effettuato il logout e poi il login, il progetto non deve più comparire nella lista.

5.2.4 Test di modifica progetto

- **Obiettivo:** assicurarsi che un progetto esistente possa essere modificato.
- **Passi:**
 1. Accedere come **PROJECT MANAGER**.
 2. Selezionare un progetto dall'elenco.
 3. Modificare i dettagli del progetto.
 4. Cliccare sul pulsante "Update".

- **Risultato Atteso:** le modifiche vengono salvate con successo e visualizzate correttamente, assicurando che gli aggiornamenti siano immediatamente riflessi nell'interfaccia e persistano nel sistema.

5.2.5 Test di accesso riservato

- **Obiettivo:** assicurarsi che solo gli utenti autorizzati possano accedere a determinate funzionalità del sistema, come la visualizzazione dell'icona relativa alla pagina “Users” nella barra laterale della dashboard o la navigazione all'URL `/users`.
- **Passi:**
 1. Accedere come utente con ruolo `PROJECT MANAGER`.
 2. Verificare la mancanza della sezione “Users” nella barra laterale.
 3. Tentare di accedere alla sezione “Users”, scrivendo forzatamente nella barra degli URL del browser l'URL `/users`.
- **Risultato Atteso:** l'accesso viene bloccato e l'utente visualizza un messaggio di errore che informa sull'assenza delle autorizzazioni necessarie per accedere alla sezione “Users”.

5.3 Risultati

Il processo di testing ha prodotto risultati complessivamente positivi. Durante le fasi di verifica, sono stati identificati e risolti diversi bug minori, prevalentemente associati all'interfaccia utente e alla gestione degli stati dell'applicazione. Relativamente a quest'ultimi, particolarmente significativo è stato il problema dell'hydration, una criticità emersa durante i test e dettagliatamente descritta nella sezione 3.4.1. Questa problematica ha influito sul modo in cui i dati venivano sincronizzati tra il server e il client, causando alcune anomalie visive e interattive nell'interfaccia.

In aggiunta, grazie alle metodologie Agile, ogni ciclo di testing ha permesso di coinvolgere attivamente gli utenti finali del sistema, garantendo che le loro esigenze e aspettative venissero tempestivamente integrate nel processo di sviluppo. L'adozione di un approccio iterativo ha semplificato l'identificazione delle aree di miglioramento, consentendo di allineare il prodotto alle reali esigenze e desideri degli utenti finali. La continua iterazione e il monitoraggio delle prestazioni del software, hanno contribuito a mantenere un elevato standard di qualità, assicurando che l'applicazione rispondesse ai requisiti iniziali e fosse anche capace di adattarsi a esigenze future e feedback emersi durante le fasi di testing.

Infine, le prestazioni del sistema sono state valutate favorevolmente e le operazioni di CRUD hanno mostrato tempi di risposta rapidi e consistenti e il feedback degli utenti durante il testing di accettazione è stato positivo.

5.4 Aspetti non testati e proposte di miglioramento

Durante il processo di testing del sistema CRM sviluppato, l'attenzione si è focalizzata principalmente sulla verifica della corretta implementazione delle funzionalità principali e sulla gestione delle operazioni

di accesso. Tuttavia, alcune aree critiche, come la sicurezza o l'accessibilità, non hanno ricevuto un'analisi dettagliata e approfondita.

5.4.1 Test di sicurezza

Durante il processo di testing, poiché il CRM era destinato ad essere un progetto interno all'azienda, la sicurezza del sistema non è stata analizzata in modo approfondito, evidenziando così una significativa lacuna da considerare. La mancanza di test di sicurezza potrebbe esporre l'applicazione a rischi significativi, come furti di dati, compromissioni della privacy e danni reputazionali. Per affrontare queste preoccupazioni, è fondamentale implementare un programma di test di sicurezza che sia continuativo, in modo da rilevare e risolvere le vulnerabilità man mano che emergono.

5.4.2 Test di accessibilità

In particolare, non sono stati condotti test specifici sulla navigazione tramite tastiera, un aspetto molto importante per garantire l'accessibilità agli utenti che, per ragioni di disabilità o preferenze personali, non utilizzano il mouse. Inoltre, non sono stati eseguiti test di compatibilità con screen reader. La mancanza di test in questo ambito potrebbe limitare l'uso del sistema a una parte significativa degli utenti, nello specifico non vedenti o ipovedenti, i quali si affidano a tecnologie assistive per interagire con le applicazioni web.

Per migliorare l'accessibilità e garantire che il CRM sia inclusivo, si propone di implementare in futuro una serie di test dedicati alla navigazione tramite tastiera e alla compatibilità con i principali screen reader. Attraverso la scrittura di test mirati al garantire l'accessibilità della web app, è possibile garantire che il sistema non solo soddisfi i requisiti funzionali, ma anche i principi di inclusività, rendendo l'esperienza utente davvero completa e soddisfacente per tutte le tipologie di utenti.

Capitolo 6

Conclusioni

6.1 Obiettivi raggiunti

Il sistema CRM per la gestione dei progetti sviluppato durante il tirocinio ha soddisfatto gli obiettivi prefissati, fornendo una soluzione completa e scalabile per la gestione di progetti e utenti all'interno di un'organizzazione. Il progetto ha permesso l'implementazione di funzionalità chiave, tra cui:

- La gestione centralizzata di utenti e ruoli, con accessi e permessi controllati basati sui ruoli `ADMIN` e `PROJECT MANAGER`, per una maggiore sicurezza e personalizzazione dell'esperienza.
- La creazione, modifica e cancellazione di progetti, con tracciamento dello stato e assegnazione dei responsabili, rendendo agevole il monitoraggio e la gestione delle attività.
- L'uso di strumenti come `AWS Cognito` e `DynamoDB`, che hanno consentito la gestione degli utenti e delle operazioni CRUD, supportando le necessità di autenticazione e archiviazione dei dati.

6.2 Lezioni apprese

Lo sviluppo del sistema CRM ha fornito numerose lezioni preziose, specialmente nell'ambito dell'uso di tecnologie come JavaScript, TypeScript, React e Next.js. L'integrazione di queste tecnologie ha permesso di costruire un'applicazione robusta e reattiva, ma ha anche comportato alcune sfide tecniche significative. L'uso di React, ad esempio, ha facilitato la creazione di interfacce utente dinamiche e modulari, ma ha reso necessaria una gestione accurata dello stato e delle proprietà per evitare problemi di performance e di rendering. Allo stesso modo, l'adozione di TypeScript ha offerto vantaggi in termini di tipizzazione statica, contribuendo a prevenire errori durante lo sviluppo, ma ha richiesto un'attenzione particolare alla definizione delle interfacce e delle strutture dei dati.

Le sfide tecniche affrontate durante lo sviluppo hanno ulteriormente sottolineato l'importanza di una pianificazione adeguata e di valutazioni rigorose per garantire la scalabilità futura del sistema. Dedicar tempo all'inizio del processo per costruire fondamenta solide e scalabili si è dimostrato essenziale, evitando così l'implementazione affrettata di funzionalità che potrebbero portare a problemi di scarsa scalabilità in futuro. In questo contesto, Next.js ha rivelato il suo valore, facilitando la gestione del rendering lato server e ottimizzando le prestazioni complessive dell'applicazione.

Inoltre, l'adozione di metodologie Agile ha mostrato numerosi benefici al progetto. Grazie alla suddivisione del lavoro in iterazioni brevi e incrementali, si è reso possibile ricevere feedback continui e apportare modifiche rapide e “leggere” in base alle esigenze emergenti. Inoltre, le retrospettive regolari hanno favorito un ambiente di apprendimento continuo, dove le sfide e le soluzioni trovate sono state condivise. In conclusione, l'esperienza di sviluppo del sistema CRM non solo ha evidenziato l'importanza di una solida pianificazione e di una gestione attenta delle tecnologie utilizzate, ma ha anche dimostrato come le metodologie Agile possano facilitare un approccio flessibile e reattivo, fondamentale per il successo di progetti complessi. Questo ha portato a un prodotto finale funzionale, scalabile e in grado di rispondere in modo efficace alle esigenze dei suoi utenti.

6.3 Possibili sviluppi futuri

Il sistema CRM ha fornito solide fondamenta, tuttavia esistono ulteriori opportunità di miglioramento e crescita. Di seguito verranno delineate alcune di queste possibilità.

6.3.1 Potenziamiento della logica di gestione degli stati

Al momento, tutti gli stati di questa applicazione vengono gestiti nei componenti stessi attraverso l'hook `useState`. Tuttavia, questa modalità di gestione presenta svantaggi significativi, specialmente in contesti complessi. Ciò include difficoltà nella gestione dello stato globale, che porta a eccessivo **prop drilling**¹, e complicazioni nella tracciabilità degli stati con l'aumento della complessità dell'applicazione. La gestione isolata di ciascuno stato può causare problemi di aggiornamenti concorrenti e inconsistenze nel rendering. Inoltre, la mancanza di centralizzazione rende difficile avere una visione complessiva dello stato dell'applicazione e complica i test automatizzati. Per queste ragioni, uno sviluppo futuro per questo progetto potrebbe essere quello di implementare una nuova logica per la gestione e il mantenimento degli stati attraverso l'utilizzo di contesti, reducer e il dispatch di azioni.

Per iniziare, si potrebbe creare una cartella denominata `Context` e spostare al suo interno le logiche esistenti relative al contesto globale (ad esempio reperire i dati dell'utente loggato). In aggiunta, si potrebbe realizzare file separati relativi ai contesti, come `Project.tsx` e `Users.tsx`, i quali “wrappe-rebbero” l'intera applicazione garantendo che le strutture dati all'interno dei contesti specifici siano accessibili in modo globale. Ogni contesto dovrà includere una costante `initState`, che definirà i dati iniziali e la struttura del contesto stesso. In questo nuovo approccio, le modifiche di stato non avverranno più tramite l'hook `useState`, ma piuttosto attraverso il dispatch di un'azione, che modificherà lo stato del contesto. Ogni volta che ciò si verificherà, l'applicazione verrà renderizzata nuovamente, proprio come accade con `useState`. Sfruttando questa metodologia, si potrà beneficiare di una gestione centralizzata e scalabile degli stati, semplificando la condivisione dei dati tra i componenti. Inoltre, contribuirà a migliorare la modularità e la manutenzione del codice, consentendo l'integrazione di funzionalità come un indicatore di `loading` durante i cambiamenti di stato o di pagina.

¹È il processo di passaggio di props attraverso diversi livelli di componenti in React, fino a raggiungere il componente che ne ha bisogno. Questo rende il codice meno leggibile e più difficile da gestire, specialmente in applicazioni complesse con molti livelli di nidificazione.

6.3.2 Analisi dei dati

Si potrebbe implementare funzionalità di data analytics per generare report dettagliati sulle performance dei progetti. Queste funzionalità consentirebbero di raccogliere e analizzare dati significativi riguardanti le tempistiche di completamento, l'utilizzo delle risorse (ad esempio il tempo totale speso su un certo progetto rispetto a quello stimato inizialmente) e la soddisfazione dei clienti. Queste analisi potrebbero essere integrate nella pagina `/performances`, già presente nella struttura del progetto. All'interno di questa pagina, dovrebbe essere possibile visualizzare report interattivi e grafici che riassumono le performance dei progetti nel tempo, facilitando l'accesso a informazioni chiave per i manager e i team di progetto.

6.3.3 Sviluppo di test sulla sicurezza e sull'accessibilità

Si propone di includere test mirati nei seguenti ambiti: innanzitutto, la navigazione tramite tastiera, tramite test automatizzati che verifichino la navigabilità dell'interfaccia, con particolare attenzione agli elementi interattivi come pulsanti, link e moduli di input. Un'altra proposta è l'implementazione di test di compatibilità con screen reader comuni per assicurare che i contenuti siano esposti in modo comprensibile e sequenziale e che l'esperienza utente sia coerente. Altri test utili includono il controllo del contrasto cromatico, per assicurarsi che le scelte di colore siano conformi agli standard WCAG 2.1[10]. Inoltre, un'analisi approfondita delle etichette e delle descrizioni alternative sarebbe fondamentale per garantire che ogni immagine o icona disponga di un testo alternativo adeguato.

Si suggerisce inoltre di implementare test di sicurezza per identificare e mitigare vulnerabilità nel software. Questo include test di penetrazione per simulare attacchi esterni e valutare la robustezza dell'applicazione. È importante anche eseguire analisi di vulnerabilità per individuare potenziali punti deboli. Ulteriori test devono riguardare la gestione delle sessioni, per assicurare che i dati degli utenti siano protetti e che le sessioni scadano in modo appropriato.

6.3.4 Implementazione della gestione utenti

Attualmente, la pagina dedicata agli utenti in `/users` consente esclusivamente la visualizzazione degli utenti registrati, senza offrire la possibilità di invitare nuovi utenti o di modificare i loro attributi. Una possibile nuova funzionalità potrebbe prevedere l'integrazione di una gestione completa degli utenti direttamente all'interno della dashboard del CRM, centralizzando così tutte le operazioni in un'unica interfaccia anziché attraverso la dashboard di Cognito.

Capitolo 7

Ringraziamenti

Grazie

Bibliografia

- [1] Archeido. Archeido - soluzioni software innovative, 2024. Accesso: 29 Settembre 2024.
- [2] Edvins. Usememo overdose. 2024. Accessed: 2024-10-27.
- [3] Giuseppe Funicello. Giuseppe funicello at reactjs day 2024, 2024. Presented: 2024-10-27.
- [4] KnowThen. Functional programming for beginners with javascript, 2024. Accesso: 29 Settembre 2024.
- [5] NextUI. Nextui - a react ui library, 2024. Accesso: 29 Settembre 2024.
- [6] Amazon Web Services. Aws amplify documentation for next.js, 2024. Accessed: 2024-10-06.
- [7] Udemy. Modern react with redux [2024 update], 2024. Accesso: 29 Settembre 2024.
- [8] Vercel. Next.js - the react framework, 2024. Accesso: 29 Settembre 2024.
- [9] Vercel. Next.js documentation, 2024. Accessed: 2024-10-01.
- [10] World Wide Web Consortium (W3C). Web content accessibility guidelines (wcag) 2.1, 2018. Accessed: 2024-11-03.
- [11] Wikipedia contributors. Typescript, 2024. Ultima modifica il 27 settembre 2024. Accesso il 1 ottobre 2024.