



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
2 Background aziendale	5
2.1 Archiedo	5
2.2 Prodotto Software	5
2.2.1 Funzionalità principali	5
2.2.2 Obiettivi del software	6
2.2.3 Stakeholders	6
2.3 Ciclo di vita adottato	6
2.3.1 Plan-based vs Agile development	6
2.4 Formazione e Pair Programming	7
2.5 Problemi e valutazioni	7
3 Background tecnico	9
3.1 Introduzione	9
3.2 JavaScript e TypeScript	9
3.2.1 Le interfacce in TypeScript	10
3.3 React	11
3.3.1 I componenti	11
3.3.2 Le <i>props</i>	12
3.3.3 Gli hooks	12
3.4 Next.js	14
3.4.1 Rendering	14
3.4.2 App Routing	15
3.5 Next UI e Tailwind CSS	17
3.6 AWS: Amazon Web Service	17
4 Descrizione dei requisiti	19
4.1 Introduzione	19
4.2 Definizione dei requisiti del sistema	19
4.2.1 Gestione dei ruoli degli utenti e uso del contesto	20
4.2.2 Operazioni di CRUD	21
4.3 Diagrammi	24
4.3.1 Diagramma dei casi d'uso (Use Case Diagram)	24
4.3.2 Sequence diagram	25

5	Testing	27
5.1	La Libreria Cypress	27
5.1.1	Test end-to-end (E2E) e test dei componenti	27
5.1.2	Testing Unitario	28
5.1.3	Testing di Integrazione	28
5.1.4	Testing di Accettazione	28
5.2	Casi di Test	28
5.2.1	Test di Creazione Progetto	28
5.2.2	Test di Modifica Progetto	28
5.2.3	Test di Accesso Riservato	29
5.3	Risultati del Testing	29
5.4	Aspetti non testati e proposte di miglioramento	29
6	Conclusioni	31
6.1	Obiettivi Raggiunti	31
6.2	Lezioni apprese	31
6.3	Possibili Sviluppi Futuri	31
6.4	Conclusioni	32
	Bibliografia	33

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 stakeholder 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 già esistenti.

Capitolo 2

Background aziendale

Questo capitolo fornisce un quadro dettagliato del contesto aziendale di Archeido. Vengono esplorate le specializzazioni di Archeido 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 primo periodo 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 contribuisce alla creazione di un'interfaccia utente reattiva, di un'esperienza utente fluida e di una gestione robusta dei dati.

3.2 JavaScript e TypeScript

JavaScript [10] è un linguaggio di programmazione multi-paradigma, concepito per supportare l'orientamento agli eventi. È utilizzato sia per la programmazione lato client, principalmente nel contesto web, sia per la programmazione lato server, grazie all'uso di Node.js.

Tuttavia, 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**. 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.

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**.

Inoltre, è completamente compatibile con JavaScript, il che consente di integrare senza problemi codice esistente.

Un esempio pratico è la funzione `add`, che esegue la somma dei parametri `a` e `b`. In JavaScript, essendo un linguaggio a tipizzazione dinamica, la funzione accetta qualsiasi tipo di valore senza eseguire controlli. Al contrario, in TypeScript la chiamata `add("5", 10)` genera un errore di compilazione, poiché il linguaggio richiede esplicitamente che i parametri siano di tipo `number`.

```

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

Nell'esempio sopra, JavaScript non genera un errore perché applica 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.

3.2.1 Le interfacce in TypeScript

Durante lo sviluppo di questo progetto, si è dimostrata molto utile la definizione a priori di **interfacce** come linee guida per i modelli dati. Le interfacce 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.

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, rimuovendone le dipendenze da ciò che appare al suo esterno.
- **Composizione di componenti:** Una volta sviluppati diversi componenti, React consente di comporli insieme. Ad esempio, un componente **Card** potrebbe contenere tre componenti più semplici: un'immagine, un titolo e un pulsante. Tale **Card** potrebbe venir 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 della prop `name`, 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 invece viene passato "Bob" (attraverso `<Greeting name="Bob" />`), verrà stampato "Hello, Bob!".

3.3.3 Gli hooks

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 hooks 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 cambiano. 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 è 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 affrontare direttamente la causa del problema.
- **Effetti collaterali indesiderati:** L'uso eccessivo di questo hook può portare a effetti collaterali imprevedibili se le dipendenze non sono 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'uso di questo 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`. Questo 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

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

per ciascuna pagina. Questa strategia consente di migliorare le performance 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.
- **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.

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.

Sebbene l'hydration offra notevoli vantaggi in termini di prestazioni e SEO, durante la realizzazione di questo progetto si sono verificati diversi errori di hydration. Questi problemi erano principalmente dovuti a incongruenze tra l'HTML generato dal server e quello che il client si aspettava dopo l'esecuzione del 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 include il contenuto della pagina. La cartella **app** funge da radice del

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

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.

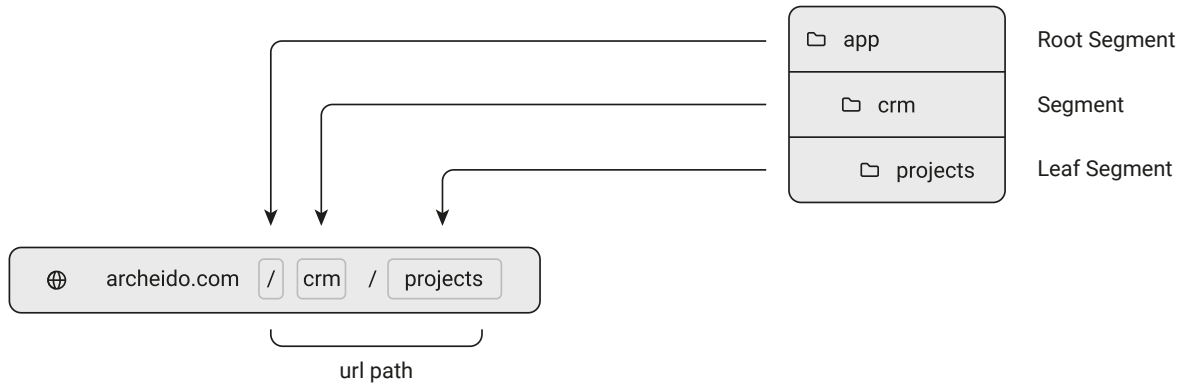


Figura 3.1: App routing [9] di Next.js

Un'altra funzionalità di Next.js è la possibilità di implementare 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. Per ottenere ciò, all'interno della cartella `/projects` si è creata un'ulteriore cartella `[projectId]` contenente un file `page.tsx` strutturato in questo modo:

```

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 3.8: 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

In questo progetto 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, modali, cards, chips, etc.) e stilizzati che semplificano la creazione di interfacce utente con un'estetica accattivante ma pulita ed intuitiva.

Un'altra libreria utilizzata per la progettazione del CRM di Archeido è Tailwind CSS: un framework CSS utility-first ³ che permette agli sviluppatori di creare interfacce utente in modo rapido e flessibile. Con Tailwind, è infatti possibile applicare stili direttamente nei file HTML o JSX, rendendo il processo di sviluppo più efficiente. 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 3.9: Parte del file `app.tsx` del CRM

3.6 AWS: Amazon Web Service

Nel contesto del progetto 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.

³È un approccio alla progettazione di interfacce utente in cui si utilizzano classi CSS per applicare stili direttamente agli elementi HTML, piuttosto che definire classi personalizzate o stili specifici per i componenti.

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

Listing 3.10: Parte del file `AuthenticatedLayout.tsx` del CRM



A screenshot of a login page. At the top center is a logo consisting of a hexagon with a blue and white geometric design. Below the logo is a white rectangular box with a thin gray border. Inside the box, the word "Email" is followed by a text input field containing the placeholder "Enter your Email". Below that, the word "Password" is followed by a text input field containing the placeholder "Enter your Password" and a toggle icon (an eye) to its right. Below the password field is a solid blue button with the text "Sign in" in white. At the bottom of the box is a blue hyperlink that says "Forgot your password?".

Figura 3.2: 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 `ADMIN`.
- **Accesso riservato ad utenti invitati:** Un nuovo utente che desidera utilizzare il CRM deve ricevere un invito via email 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, modificandone i campi sopracitati.
- **Cancellazione di un progetto:** l'utente può cancellare un progetto.

4.2.1 Gestione dei ruoli degli utenti e uso del contesto

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 avviene 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. Prima di tutto 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 chiamata `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 qui sotto.

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` creato 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 questa colonna 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 tale pagina è possibile effettuare modifiche o procedere all'eliminazione del progetto.

Per eliminare un progetto, si utilizza la funzione `deleteProject`, che richiede come argomento l'oggetto `Project` da eliminare, e tramite il suo ID, 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 fatto attraverso 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 };
```

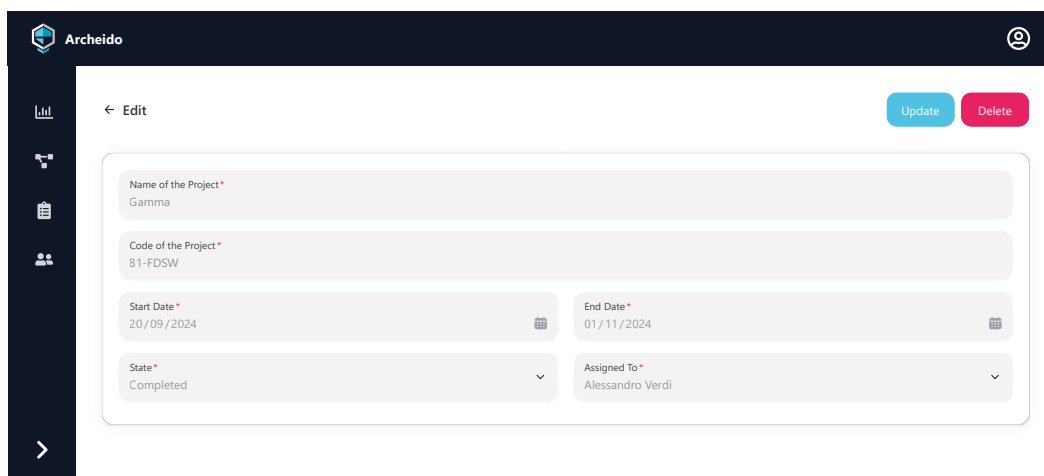
Listing 4.9: funzione `updateProject`The image shows a web application interface for 'Archeido'. On the left is a dark sidebar with navigation icons. The main content area is titled '← Edit' and contains a form for editing a project. The form has several input fields: 'Name of the Project*' with the value 'Gamma', 'Code of the Project*' with the value 'B1-FDSW', 'Start Date*' with the value '20/09/2024' and a calendar icon, 'End Date*' with the value '01/11/2024' and a calendar icon, 'State*' with a dropdown menu showing 'Completed', and 'Assigned To*' with a dropdown menu showing 'Alessandro Verdi'. At the top right of the form area 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 comprendono la creazione, lettura, aggiornamento e cancellazione di progetti, 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.

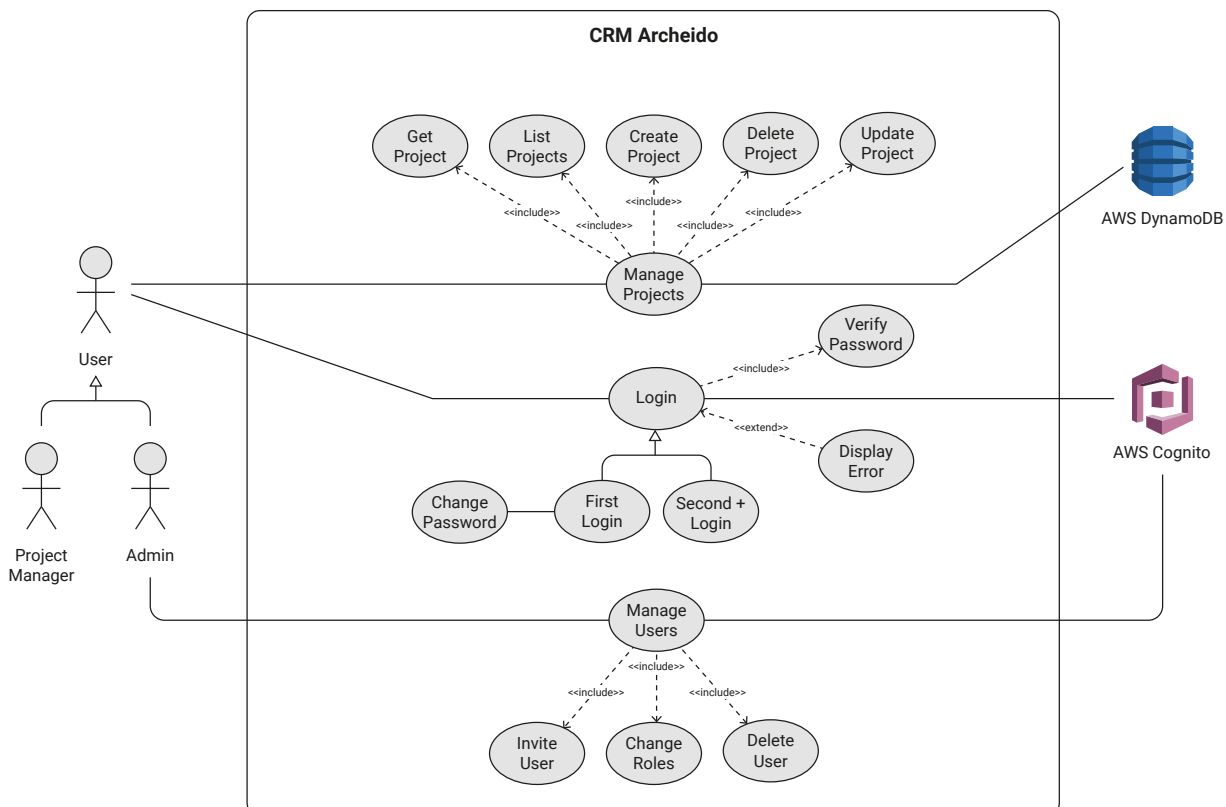


Figura 4.3: Use Case Diagram

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 (email e password) nella pagina di login, che inoltra i dati al pool di utenti Cognito per verificarne la validità. Se l'email 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 vedere 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 chiamate le funzioni `createProject` o `deleteProject`, rispettivamente. 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.

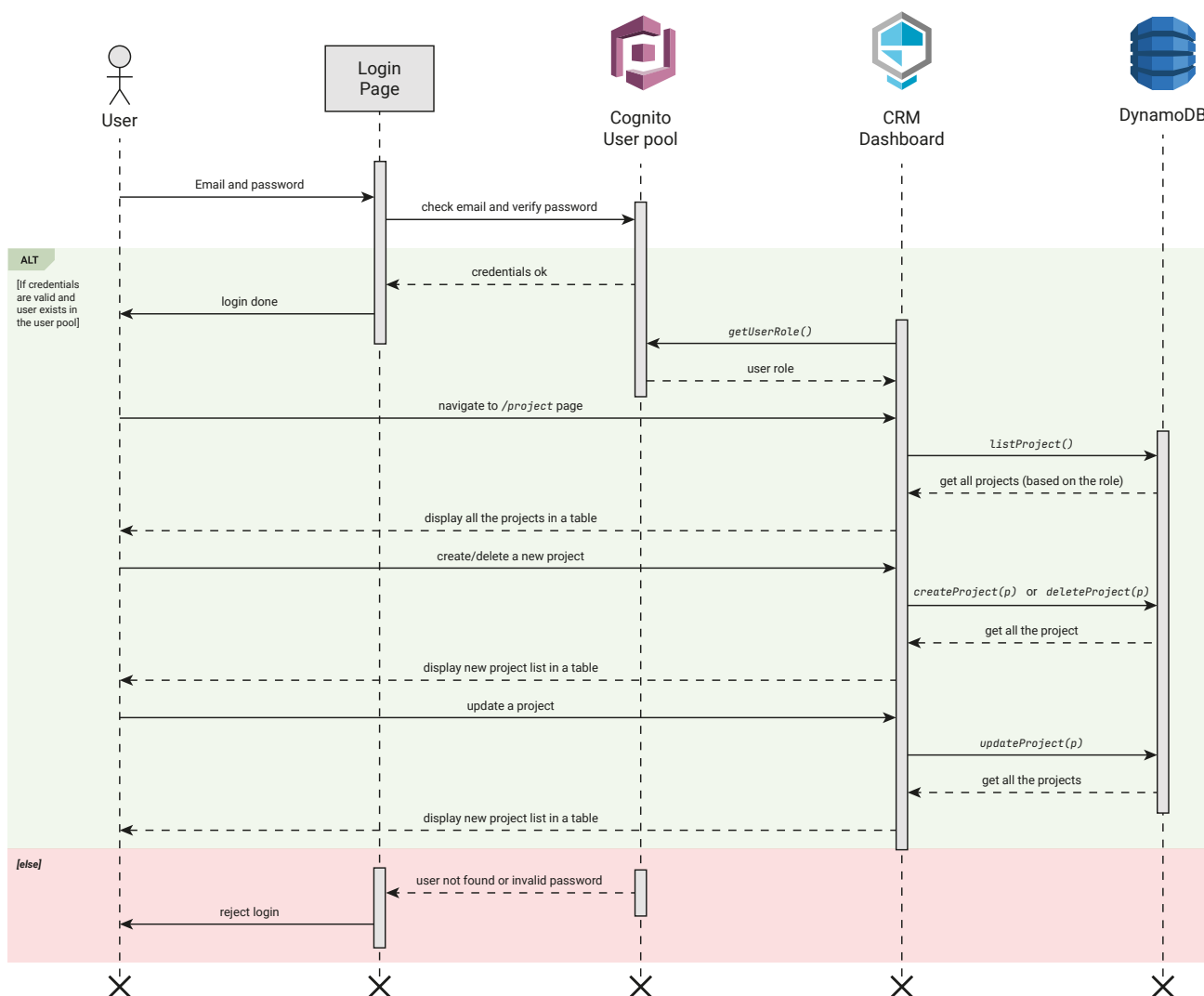


Figura 4.4: Sequence Diagram per il login e le operazioni CRUD sui progetti

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. Questo approccio ha portato a un'applicazione operativa e robusta, ma ha anche rivelato lacune significative in termini di accessibilità, un aspetto che merita una riflessione più approfondita, di cui se ne parlerà in seguito.

5.1 La Libreria Cypress

Cypress è un framework di testing end-to-end per applicazioni web, progettato per semplificare la scrittura, l'esecuzione e il debug dei test. Cypress consente di testare l'intero flusso di un'applicazione simulando le interazioni dell'utente con l'interfaccia, come clic, inserimenti di dati e navigazione tra le pagine. La sua sintassi è intuitiva e simile a JavaScript, rendendo la scrittura dei test accessibile anche a chi ha poca esperienza nel campo del testing, motivazione per la quale si è optato per lui in questo progetto.

Una delle caratteristiche distintive di Cypress è la sua interfaccia grafica (GUI), che permette di eseguire i test in tempo reale, mostrando il comportamento dell'applicazione e il risultato di ogni test mentre vengono eseguiti. Inoltre, offre strumenti di debugging integrati, che consentono agli sviluppatori di vedere esattamente cosa accade nel loro codice durante l'esecuzione dei test.

5.1.1 Test end-to-end (E2E) e test dei componenti

Cypress offre due modalità principali di testing: i test end-to-end (E2E) e i test dei componenti.

- I test end-to-end sono progettati per verificare il funzionamento dell'intera applicazione dall'inizio alla fine, simulando l'interazione dell'utente con l'interfaccia. Questa modalità consente di testare scenari complessi in cui più parti dell'applicazione sono coinvolte, garantendo che tutte le parti dell'applicazione funzionino correttamente e che l'esperienza utente sia fluida e priva di errori.
- D'altra parte, i test dei componenti si concentrano su singoli componenti dell'interfaccia utente, consentendo di verificare il comportamento e l'aspetto di ciascun elemento in isolamento. Utilizzando Cypress per i test dei componenti, è possibile testare come un componente reagisce a diverse props o stati, assicurandosi che funzioni correttamente in vari scenari. Questo approccio favorisce un ciclo

di sviluppo più rapido e consente di identificare e correggere i bug a livello di singolo componente prima che influenzino l'intera applicazione.

5.1.2 Testing Unitario

Il testing unitario è stato utilizzato per testare singole funzioni e componenti del sistema. Ogni funzione, come quelle di creazione, lettura, aggiornamento e cancellazione (CRUD) dei progetti, è stata testata in isolamento per garantire che operassero come previsto.

5.1.3 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.

5.1.4 Testing di Accettazione

Il testing di accettazione è stato realizzato coinvolgendo gli utenti finali nel processo. Gli utenti hanno testato le funzionalità del CRM per confermare che il sistema soddisfacesse i requisiti aziendali e che fosse user-friendly.

5.2 Casi di Test

I casi di test sono stati progettati per coprire tutte le funzionalità principali del sistema. Di seguito sono riportati alcuni esempi significativi:

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 e appare nell'elenco dei progetti.

5.2.2 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 e visualizzate correttamente.

5.2.3 Test di Accesso Riservato

- Obiettivo: Verificare che solo gli utenti autorizzati possano accedere a determinate funzionalità.
- Passi:
 1. Accedere come utente con ruolo `PROJECT MANAGER`.
 2. Tentare di accedere alla sezione "Users", scrivendo forzatamente nella barra degli url del browser l'url `/users`.
- Risultato Atteso: L'accesso viene negato e viene visualizzato un messaggio di errore.

5.3 Risultati del Testing

Il processo di testing ha portato a risultati positivi. Sono stati identificati e risolti alcuni bug minori, principalmente legati all'interfaccia utente e alla gestione degli stati. Le prestazioni del sistema sono state valutate positivamente, e le operazioni di CRUD hanno mostrato tempi di risposta rapidi e consistenti. Inoltre, il feedback degli utenti durante il testing di accettazione é stato positivo.

5.4 Aspetti non testati e proposte di miglioramento

Durante il processo di testing del sistema CRM sviluppato, ci si è concentrati prevalentemente sulla verifica della corretta implementazione delle funzionalità principali. Tuttavia, gli aspetti riguardanti l'accessibilità della web app non sono stati oggetto di un test approfondito.

Ad esempio, uno degli aspetti di accessibilità che non sono stati testati riguarda la navigazione tramite tastiera e la compatibilità con screen reader. La navigazione tramite tastiera rappresenta un elemento molto importante per gli utenti che non possono o preferiscono non utilizzare il mouse, garantendo l'accesso completo alle funzionalità dell'applicazione. Allo stesso modo, non sono stati condotti test di compatibilità con screen reader, strumenti essenziali per gli utenti non vedenti o ipovedenti.

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. 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.

Inoltre, l'adozione di metodologie Agile ha portato 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 in base alle esigenze emergenti.

6.3 Possibili Sviluppi Futuri

Il sistema CRM ha posto delle solide basi, ma ci sono opportunità di miglioramento e crescita:

- **Estensione dei Ruoli:** la possibilità di aggiungere ulteriori ruoli con permessi differenziati potrebbe ampliare l'utilizzo del sistema, favorendo una più ampia delega delle responsabilità.

- **Integrazione con Altri Servizi:** l'integrazione con altre API, come strumenti di gestione documentale o di reportistica avanzata, offrirebbe un supporto decisionale migliore agli utenti.
- **Analisi dei Dati:** implementare funzionalità di data analytics per generare report sulle performance dei progetti, aiutando l'azienda a monitorare l'efficienza operativa.
- **Potenziamento della Logica di Gestione degli Stati:** implementare una nuova logica per la gestione e il mantenimento degli stati attraverso l'uso di contesti e del dispatch di azioni. Questo approccio consentirebbe una gestione centralizzata e scalabile degli stati, semplificando la condivisione dei dati tra componenti e migliorando la modularità e la manutenzione del codice, aggiungendo ad esempio, un `loading` durante ogni cambio di stato/pagina.
- **Sviluppo di test per migliorare l'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. 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.
- **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. Questa implementazione permetterebbe di inviare inviti a nuovi utenti e di rimuovere quelli esistenti, semplificando il processo di amministrazione e migliorando significativamente l'usabilità del sistema. .

6.4 Conclusioni

Il progetto CRM realizzato durante il tirocinio rappresenta un efficace strumento di gestione dei progetti aziendali, grazie a una piattaforma sicura e facilmente scalabile. Questo progetto, oltre a consolidare competenze tecniche nella gestione e nello sviluppo di sistemi basati su cloud, ha aperto la strada a miglioramenti futuri e a possibili nuove funzionalità, rafforzando il contributo dell'applicazione nel supportare i processi aziendali.

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] Wikipedia contributors. Javascript, 2024. Ultima modifica il 22 settembre 2024. Accesso il 1 ottobre 2024.
- [11] Wikipedia contributors. Typescript, 2024. Ultima modifica il 27 settembre 2024. Accesso il 1 ottobre 2024.