

SUPSI

App mobile per gestione timbrature e controllo qualità collaboratori

Studente/i

Walter Sostene Losa

Relatore

Andrea Baldassari

Correlatore

Matteo Besenzoni

Committente

**Progett SA - Impresa Generale
di Pulizie e Facility Management**

Corso di laurea

Ingegneria Informatica

Codice progetto

C10793

Anno

2023/2024

Data

20 agosto 2024

STUDENTSUPSI

Indice

1	Progetto Assegnato	3
1.1	Funzionalità per i Dipendenti	3
1.2	Funzionalità per i Responsabili	3
1.3	Funzionalità per i Clienti	4
1.4	Obiettivi del Sistema	4
2	Introduzione	5
3	Motivazione e Contesto	7
3.1	Obiettivo	8
4	Problema	9
5	Stato dell'arte	11
6	Approccio al problema	13
6.1	Gestione del progetto	13
6.2	Linguaggi di Programmazione	13
6.2.1	Backend	13
6.2.2	Frontend	14
6.3	Integrazione con Componenti del Dispositivo	15
6.4	Toolkits	15
7	Progettazione	17
7.1	Requisiti	17
7.1.1	Requisiti per gli Amministratori	17
7.1.2	Requisiti per i Dipendenti	17
7.1.3	Requisiti per i Responsabili (Supervisor)	18
7.1.4	Requisiti per i Clienti	18
7.2	Mockup delle Interfacce	18
7.2.1	Pagina di accesso	18
7.2.2	Pagina di registrazione delle timbrature	18

7.2.3	Pagina di gestione delle segnalazioni	18
7.2.4	Pagina di controllo qualità per i responsabili	18
7.2.5	Pagina di feedback dei clienti	18
8	Architettura	19
8.1	Visione Generale	19
8.2	Struttura del Codice	19
8.2.1	Frontend	19
8.2.2	Backend	20
8.3	Comunicazione Frontend-Backend	20
8.3.1	Comunicazione tramite API REST	21
8.3.2	Struttura delle Richieste e Risposte	21
8.3.3	Gestione degli Errori e Feedback	22
8.3.4	Autenticazione e Autorizzazione	22
8.3.5	Esempio di Flusso di Comunicazione	22
8.3.6	Sicurezza nella Gestione dei Dati Sensibili	22
8.4	Comunicazione Backend-Database	23
8.4.1	Operazioni CRUD tramite ORM	24
8.4.2	Gestione delle Relazioni tra Tabelle	24
8.4.3	Validazione dei Dati e Gestione degli Errori	25
8.4.4	Sicurezza e Ottimizzazione delle Query	25
8.4.5	Esempio di Flusso di Comunicazione	25
9	Implementazione	27
10	Test	29
11	Risultati	31
12	Conclusioni	33

Elenco delle figure

Elenco delle tabelle

Abstract

Capitolo 1

Progetto Assegnato

Il partner di progetto, un'importante azienda attiva nei servizi di pulizia e facility management, necessita di un sistema che permetta la gestione e il controllo qualità dei servizi erogati presso i clienti. Il sistema, accessibile tramite applicazione mobile, sarà utilizzato principalmente da tre attori: Dipendenti, Responsabili e Clienti.

1.1 Funzionalità per i Dipendenti

I Dipendenti avranno a disposizione le seguenti funzionalità:

- **Registrazione delle presenze:** I dipendenti possono registrare (timbrare) il loro arrivo e la loro partenza presso i clienti da cui si recano, tramite QR code o tag NFC dedicato.
- **Segnalazioni di anomalie:** Possono fare segnalazioni, allegando anche foto e video, presso i clienti, in caso di anomalie riscontrate durante il servizio.
- **Ordini di attrezzature e prodotti:** Possono fare un ordine delle attrezzature e dei prodotti che sono finiti o che sono in esaurimento, facilitando la gestione del materiale necessario per svolgere le loro mansioni.

1.2 Funzionalità per i Responsabili

I Responsabili avranno a disposizione le seguenti funzionalità:

- **Verifica delle timbrature:** Possono verificare le timbrature dei dipendenti di loro responsabilità ai fini di controllo qualità, assicurando che i servizi vengano svolti nei tempi previsti.
- **Segnalazioni di controllo qualità:** Possono fare segnalazioni per controllo qualità, direttamente al dipendente ed indicando il cliente, permettendo una rapida risoluzione di eventuali problemi riscontrati.

1.3 Funzionalità per i Clienti

I Clienti avranno a disposizione le seguenti funzionalità:

- **Segnalazioni di criticità:** Possono segnalare criticità nel servizio ricevuto, offrendo un feedback diretto all'azienda per migliorare la qualità del servizio.
- **Sondaggi sulla qualità del servizio:** Possono essere sottoposti a sondaggi sulla qualità del servizio ricevuto, fornendo dati utili per il miglioramento continuo.

1.4 Obiettivi del Sistema

Il sistema è progettato per raggiungere i seguenti obiettivi:

- **Miglioramento della gestione delle presenze:** Tramite l'uso di tecnologie come QR code e tag NFC, il sistema assicura una registrazione precisa e affidabile delle presenze dei dipendenti.
- **Aumento della trasparenza e della comunicazione:** Le funzionalità di segnalazione per dipendenti, responsabili e clienti aumentano la trasparenza e migliorano la comunicazione tra le parti coinvolte.
- **Ottimizzazione delle risorse:** La possibilità di ordinare attrezzature e prodotti direttamente tramite l'app facilita la gestione delle risorse, assicurando che i dipendenti abbiano sempre a disposizione il materiale necessario.
- **Controllo qualità continuo:** Le funzionalità di verifica e segnalazione per i responsabili permettono un controllo qualità continuo, garantendo un elevato standard del servizio erogato.

Capitolo 2

Introduzione

Il progetto nasce dalla necessità di Progett SA, un'importante azienda attiva nei servizi di pulizia e facility management, di migliorare la gestione e il controllo qualità dei servizi erogati presso i clienti. L'azienda desidera un sistema che permetta ai dipendenti di registrare le presenze presso i clienti, di segnalare anomalie riscontrate durante il servizio e di fare ordini di attrezzature e prodotti. I responsabili devono poter verificare le timbrature dei dipendenti di loro responsabilità e fare segnalazioni per controllo qualità. I clienti devono poter segnalare criticità nel servizio ricevuto e essere sottoposti a sondaggi sulla qualità del servizio. Il sistema deve garantire una registrazione precisa e affidabile delle presenze, aumentare la trasparenza e migliorare la comunicazione tra le parti coinvolte, facilitare la gestione delle risorse e assicurare un controllo qualità continuo.

Il sistema sarà accessibile tramite applicazione mobile e sarà utilizzato principalmente da tre attori: Dipendenti, Responsabili e Clienti. I Dipendenti potranno registrare le presenze, fare segnalazioni di anomalie e fare ordini di attrezzature e prodotti. I Responsabili potranno verificare le timbrature dei dipendenti e fare segnalazioni per controllo qualità. I Clienti potranno segnalare criticità e essere sottoposti a sondaggi sulla qualità del servizio.

Il Capitolo 2 del documento presenta un'analisi delle motivazioni e del contesto che hanno guidato lo sviluppo del progetto. Inoltre, vengono introdotti in modo preliminare gli obiettivi che costituiscono il nucleo centrale di questo lavoro.

todo: aggiungere una breve spiegazione di ogni capitolo

Capitolo 3

Motivazione e Contesto

In questo capitolo, vengono descritte le motivazioni accademiche che stanno alla base di questo lavoro di tesi, insieme al contesto in cui si inseriscono. In conclusione, vengono delineati gli obiettivi che guidano lo sviluppo di questo progetto.

Nell'ambito del corso di Bachelor in Ingegneria Informatica presso la Scuola Universitaria Professionale della Svizzera Italiana (SUPSI), ogni studente si trova di fronte al compito di completare un lavoro di diploma. Questo implica la scelta autonoma di un progetto tra quelli proposti da diversi relatori. Tali progetti abbracciano una vasta gamma di settori, spaziando tra software desktop standalone, applicazioni web, networking, intelligenza artificiale e molto altro. Ogni studente è tenuto a sviluppare il progetto nel corso della stagione estiva, per poi consegnarlo entro l'inizio di settembre. Lo scopo di questo progetto è applicare le tecniche, le strategie e i metodi di sviluppo appresi durante i tre anni di studi del percorso di Bachelor.

La scelta di svolgere questo particolare progetto è stata determinata da diversi fattori. In particolare, l'opportunità di sviluppare un'applicazione web è stata considerata un modo per scoprire meglio questo campo non essendo stato molto visto durante gli anni, offrendo uno spazio ideale per l'applicazione delle competenze acquisite. L'uso di un framework aggiornato, ha fornito un'opportunità unica per scoprire una nuova tecnologia ancora non molto diffusa nei software odierni.

Il progetto è seguito dai relatori Baldassari Andrea e Matteo Besenzoni, entrambi docenti e ricercatori presso la SUPSI. In particolare, il docente Andrea Baldassari ha fornito i requisiti e le specifiche man mano che il lavoro procedeva.

3.1 Obiettivo

L'obiettivo di questo progetto è sviluppare un'applicazione web per la gestione delle timbrature e il controllo qualità dei collaboratori di Progett SA. L'applicazione deve permettere ai collaboratori di timbrare in entrata e in uscita, di inserire le attività svolte durante la giornata e di caricare le foto dei lavori eseguiti. Inoltre, l'applicazione deve permettere ai responsabili di controllo qualità di visualizzare le timbrature e le attività svolte dai collaboratori, di valutare la qualità del lavoro svolto e di fornire feedback ai collaboratori.

Capitolo 4

Problema

Il problema affrontato nel contesto del progetto riguarda la necessità di sviluppare un sistema completo per la gestione delle timbrature e il controllo della qualità dei servizi offerti da un'azienda attiva nel settore della pulizia e del facility management. Attualmente, l'azienda si trova a dover affrontare diverse sfide legate alla gestione e al monitoraggio dei propri collaboratori e dei servizi erogati, tra cui:

- **Mancanza di un sistema di tracciamento efficiente:** L'assenza di un sistema integrato per la registrazione delle timbrature dei dipendenti presso i vari clienti rende difficile il monitoraggio in tempo reale delle attività lavorative e il controllo della qualità dei servizi erogati.
- **Difficoltà nella gestione delle segnalazioni:** Attualmente, le segnalazioni relative a anomalie o criticità del servizio devono essere gestite manualmente, il che comporta inefficienze operative e ritardi nella risoluzione dei problemi.
- **Carenza di trasparenza nelle comunicazioni:** I canali di comunicazione tra dipendenti, responsabili e clienti non sono ottimizzati, portando a una scarsa trasparenza e a possibili incomprensioni riguardo alle prestazioni del servizio e alle necessità dei clienti.
- **Inadeguatezza nella gestione delle scorte di attrezzature e prodotti:** La mancanza di un sistema automatizzato per la gestione delle richieste di rifornimento di attrezzature e prodotti può causare ritardi nel rifornimento, con conseguente impatto negativo sulla qualità del servizio.
- **Limitazioni nell'integrazione tecnologica:** L'integrazione con dispositivi mobili e tecnologie come NFC e QR code non è stata pienamente sfruttata, limitando la possibilità di utilizzare strumenti digitali avanzati per il controllo della qualità e la gestione delle attività lavorative.

Capitolo 5

Stato dell'arte

Capitolo 6

Approccio al problema

In questo capitolo, esamineremo in dettaglio l'approccio adottato per sviluppare una Progressive Web App (PWA) dedicata alla gestione e al controllo qualità dei servizi di pulizia e facility management. Illustreremo le scelte effettuate per quanto riguarda la gestione del progetto, i linguaggi di programmazione, i framework selezionati e l'integrazione con i componenti dei dispositivi mobili. Ogni decisione tecnica è stata presa con l'obiettivo di garantire prestazioni elevate, facilità di manutenzione e un'esperienza utente ottimale.

6.1 Gestione del progetto

La gestione del progetto è stata condotta utilizzando una metodologia Agile, in particolare il framework Scrum. Questa scelta è stata dettata dalla necessità di mantenere un'elevata flessibilità durante lo sviluppo, permettendo al team di adattarsi rapidamente ai cambiamenti nei requisiti del cliente. Ogni settimana è stato condotto uno sprint, al termine del quale si teneva una riunione con i committenti per presentare i progressi attraverso demo, raccogliere feedback e pianificare le attività per la settimana successiva.

Per la gestione del codice sorgente e delle attività del progetto, è stata utilizzata la piattaforma GitLab. GitLab ha offerto strumenti integrati per il versionamento del codice, la gestione delle issue, la pianificazione degli sprint e l'integrazione continua. Questi strumenti hanno facilitato una stretta collaborazione all'interno del team e una gestione efficace del progetto, assicurando che il codice fosse sempre aggiornato e che ogni membro del team fosse informato sugli sviluppi.

6.2 Linguaggi di Programmazione

6.2.1 Backend

Per lo sviluppo del backend, abbiamo scelto di utilizzare Node.js in combinazione con il framework Express e l'ORM Sequelize. La scelta di Node.js è stata guidata da diversi fattori

chiave:

- **Asincronicità e Performance:** Node.js è basato su un modello di I/O non bloccante, che consente di gestire un elevato numero di richieste simultanee con un'efficienza notevole. Questo è particolarmente vantaggioso per un'applicazione che deve interagire con dispositivi mobili e gestire numerose operazioni in tempo reale, come l'acquisizione di dati dai sensori dei dispositivi.
- **Ampio Ecosistema:** Node.js dispone di un vasto ecosistema di librerie e moduli disponibili attraverso npm (Node Package Manager). Questo ci ha permesso di accelerare lo sviluppo utilizzando pacchetti preesistenti per funzionalità comuni, riducendo il tempo necessario per implementare funzioni complesse.
- **Scalabilità:** La natura asincrona di Node.js lo rende particolarmente adatto per costruire applicazioni scalabili. Per un sistema come quello proposto, che potrebbe essere utilizzato da numerosi utenti contemporaneamente, la capacità di scalare orizzontalmente è cruciale.

Express, un framework minimalista per Node.js, è stato scelto per la sua semplicità e flessibilità nella gestione delle API RESTful. Express consente di strutturare l'applicazione in modo modulare, facilitando la manutenzione e l'espansione del codice nel tempo. Inoltre, con l'uso di Sequelize, un ORM per Node.js, è stato possibile gestire le interazioni con il database MySQL in modo sicuro ed efficiente, mappando le tabelle del database su oggetti JavaScript e semplificando le operazioni CRUD (Create, Read, Update, Delete).

6.2.2 Frontend

Per il frontend, è stato scelto React come framework principale per lo sviluppo dell'interfaccia utente. Le motivazioni alla base di questa scelta sono molteplici:

- **Componentizzazione:** React adotta un approccio basato su componenti, che permette di costruire interfacce utente modulari e riutilizzabili. Questa modularità non solo facilita lo sviluppo, ma rende anche più semplice la manutenzione e l'aggiornamento dell'applicazione nel tempo.
- **Performance e Virtual DOM:** React utilizza un Virtual DOM per ottimizzare il rendering dell'interfaccia utente. Questo riduce al minimo le operazioni di manipolazione del DOM reale, migliorando le prestazioni, soprattutto su dispositivi mobili, dove la potenza di calcolo può essere limitata.
- **Ecosistema e Community:** La vasta community di React e l'ampio ecosistema di librerie di supporto hanno fornito strumenti utili e best practices che hanno accelerato lo sviluppo e migliorato la qualità del codice.

Inoltre, per migliorare ulteriormente la velocità di sviluppo e la coerenza dell'interfaccia utente, è stata utilizzata la libreria **Ant Design (Antd)**. Antd è un set di componenti UI predefiniti per React, che offre una vasta gamma di elementi grafici stilisticamente coerenti e pronti all'uso. L'adozione di Antd ha permesso al team di:

- **Accelerare lo Sviluppo:** Utilizzando componenti predefiniti di alta qualità, il team ha potuto concentrarsi maggiormente sulle logiche di business piuttosto che sulla progettazione e implementazione di elementi grafici complessi.
- **Mantenere una UI Coerente:** Antd garantisce che l'interfaccia utente sia esteticamente coerente e intuitiva, migliorando l'esperienza utente complessiva.
- **Facile Personalizzazione:** Sebbene Antd offra componenti predefiniti, è altamente personalizzabile, consentendo al team di adattare l'aspetto e il comportamento dei componenti alle specifiche esigenze del progetto.

L'uso di React, insieme a Antd, ha inoltre facilitato l'integrazione con le API dei dispositivi, come la fotocamera e i sensori di posizione, rendendo possibile la realizzazione di funzionalità critiche come la scansione di QR code e la tracciabilità GPS.

6.3 Integrazione con Componenti del Dispositivo

Uno degli aspetti più innovativi dell'applicazione è la sua capacità di interagire direttamente con i componenti hardware dei dispositivi mobili. Grazie a librerie specifiche di React, l'applicazione può accedere alla fotocamera per scattare foto e scansionare codici QR, nonché raccogliere dati GPS per tracciare la posizione dei lavoratori in tempo reale. Questo livello di integrazione è stato fondamentale per garantire che il sistema potesse offrire funzionalità di controllo qualità avanzate, come la verifica dell'ubicazione del personale sul campo e la documentazione delle attività svolte.

6.4 Toolkits

Oltre ai linguaggi e framework descritti, sono stati utilizzati diversi toolkits per supportare lo sviluppo:

- **npm:** Il gestore di pacchetti npm è stato fondamentale per la gestione delle dipendenze e per l'installazione di librerie esterne che hanno velocizzato lo sviluppo.
- **Webpack:** Webpack è stato utilizzato per il bundling dei file JavaScript, ottimizzando le performance del frontend e migliorando i tempi di caricamento dell'applicazione.

- **GitLab CI/CD:** La pipeline CI/CD di GitLab ha automatizzato il processo di testing e deployment, garantendo che ogni modifica al codice fosse adeguatamente verificata prima di essere distribuita nell'ambiente di produzione.

L'insieme di questi strumenti ha permesso di creare un ambiente di sviluppo efficiente e affidabile, facilitando il rilascio continuo di nuove funzionalità e miglioramenti.

Capitolo 7

Progettazione

In questo capitolo viene descritta la fase di progettazione dell'applicativo, partendo dall'analisi dei requisiti necessari per poi passare alla descrizione delle funzionalità principali e dei mockup delle interfacce che saranno utilizzate dagli utenti.

7.1 Requisiti

Il sistema, accessibile tramite app, avrà principalmente quattro tipi di utenti: amministratori, dipendenti, responsabili (supervisor), e clienti. Di seguito sono elencati i requisiti specifici per ciascuno di questi ruoli.

7.1.1 Requisiti per gli Amministratori

Gli amministratori hanno il compito di gestire l'intero sistema, comprese le operazioni di gestione utenti e forniture. I requisiti specifici per questo ruolo includono:

- Creare, modificare e gestire gli account degli utenti (dipendenti, responsabili e clienti).
- Aggiungere, modificare e gestire le forniture disponibili nel database.
- Monitorare tutte le segnalazioni di controllo qualità effettuate dai clienti.
- Visualizzare e gestire tutti i servizi svolti dai dipendenti.

7.1.2 Requisiti per i Dipendenti

I dipendenti sono coloro che eseguono le attività operative presso i clienti. I requisiti specifici per questo ruolo includono:

- Registrare (timbrare) il loro arrivo e la loro partenza presso i clienti utilizzando QR code.

- Effettuare segnalazioni in caso di anomalie riscontrate presso i clienti.
- Ordinare attrezzature e prodotti che sono finiti o in esaurimento.

7.1.3 Requisiti per i Responsabili (Supervisor)

I responsabili, che sono dipendenti con funzioni aggiuntive, hanno il compito di controllare la qualità del lavoro svolto dai dipendenti sotto la loro responsabilità. I requisiti specifici per questo ruolo includono:

- Verificare le timbrature dei dipendenti di loro responsabilità ai fini del controllo qualità.

7.1.4 Requisiti per i Clienti

I clienti sono coloro che ricevono i servizi e possono fornire feedback sulla loro qualità. I requisiti specifici per questo ruolo includono:

- Segnalare criticità nel servizio ricevuto.

7.2 Mockup delle Interfacce

In questa sezione vengono descritti i mockup delle interfacce dell'applicazione, con particolare attenzione alle schermate più rilevanti per ciascun attore coinvolto (dipendenti, responsabili e clienti).

7.2.1 Pagina di accesso

7.2.2 Pagina di registrazione delle timbrature

7.2.3 Pagina di gestione delle segnalazioni

7.2.4 Pagina di controllo qualità per i responsabili

7.2.5 Pagina di feedback dei clienti

Capitolo 8

Architettura

In questo capitolo viene descritta l'architettura dell'applicativo. Iniziando con una visione generale di come sia strutturato l'applicativo, per poi passare ad una descrizione della struttura del codice, successivamente una spiegazione su come frontend e backend interagiscono tra di loro e come il backend comunica con il database.

8.1 Visione Generale

L'applicazione è progettata con un'architettura a componenti separati, suddivisa in tre parti principali:

- **Frontend:** Fornisce un'interfaccia intuitiva e interattiva per l'interazione con il sistema.
- **Backend:** Gestisce le richieste dal frontend, elabora i dati e comunica con il database.
- **Database MySQL:** Utilizzato per la persistenza dei dati, come informazioni sugli utenti, accessi e servizi.

8.2 Struttura del Codice

In questa sezione vengono discusse le scelte e le motivazioni che hanno portato a definire la struttura del codice dell'applicativo

8.2.1 Frontend

Lo sviluppo del frontend si è basato su convenzioni comuni utilizzate per sviluppare in *TypeScript* e *React*.

All'interno della cartella `frontend` è dunque presente una cartella `src` la quale contiene tutto il codice dell'applicativo web.

La cartella viene poi suddivisa in:

- `components`: collezione di componenti utilizzati all'interno dell'applicazione.
- `services`: collezione di servizi utilizzati per comunicare con il backend.
- `context`: collezione di contesti utilizzati per gestire lo stato dell'applicazione.
- `pages`: diverse pagine alle quali si può accedere attraverso l'applicazione.
- `utils`: file di utilità contenenti funzioni utilizzate più volte all'interno del frontend.

Oltre alle cartelle sopra elencate, sono presenti anche i file `App.tsx` e `index.tsx` i quali rappresentano rispettivamente il componente principale dell'applicazione e il file di entry point.

Infine è presente anche un file chiamato `type.ts` il quale contiene tutte le interfacce utilizzate all'interno del frontend.

8.2.2 Backend

Il progetto di backend è stato scritto in *Node.js* utilizzando il framework *Express*. Questo ha permesso di creare un server web che gestisce le richieste HTTP provenienti dal frontend e comunica con il database per recuperare e salvare i dati.

All'interno della cartella `backend` è presente una cartella `src` la quale contiene tutto il codice del server. In particolare, la cartella è suddivisa in:

- `db`: contiene i modelli e le configurazioni del database.
- `middleware`: contiene i middleware utilizzati per gestire le richieste HTTP.
- `repositories`: contiene i repository utilizzati per interagire con il database.
- `resources`: contiene le risorse che vengono inviate al frontend.
- `routes`: contiene i controller che gestiscono le richieste HTTP.
- `utils`: file di utilità contenenti funzioni utilizzate più volte all'interno del backend.

Anche in questo caso, oltre alle cartelle sopra elencate, sono presenti i file `app.ts` e `server.ts` i quali rappresentano rispettivamente il componente principale del server e il file di entry point.

8.3 Comunicazione Frontend-Backend

In questa sezione viene descritta l'interazione tra il frontend e il backend dell'applicazione, illustrando il flusso di comunicazione e le tecnologie utilizzate per garantire un'efficace integrazione tra i due componenti.

8.3.1 Comunicazione tramite API REST

Il frontend e il backend comunicano principalmente tramite API REST (Representational State Transfer). Le API REST sono endpoint esposti dal backend che il frontend può chiamare per inviare o ricevere dati. Questi endpoint sono progettati per seguire i principi REST, che includono l'uso di metodi HTTP standard come GET, POST, PUT e DELETE per operazioni CRUD (Create, Read, Update, Delete).

- **GET:** Utilizzato per recuperare dati dal backend. Ad esempio, il frontend può effettuare una richiesta GET per ottenere una lista di servizi o dettagli su un utente specifico.
- **POST:** Utilizzato per inviare nuovi dati al backend. Ad esempio, quando un utente registra una nuova timbratura, il frontend invia una richiesta POST al backend con i dettagli della timbratura.
- **PUT:** Utilizzato per aggiornare dati esistenti nel backend. Ad esempio, il frontend può inviare una richiesta PUT per aggiornare le informazioni di un servizio esistente.
- **DELETE:** Utilizzato per eliminare dati dal backend. Ad esempio, il frontend può effettuare una richiesta DELETE per rimuovere un utente.

8.3.2 Struttura delle Richieste e Risposte

Le richieste dal frontend al backend sono effettuate utilizzando la libreria `axios`. Ogni richiesta include un URL di destinazione (l'endpoint dell'API), un metodo HTTP, e, se necessario, un corpo della richiesta (payload) con i dati da inviare.

Le risposte del backend possono includere dati in formato JSON (JavaScript Object Notation) o oggetti JavaScript. Le risposte includono un codice di stato HTTP che indica l'esito dell'operazione (ad esempio, 200 OK, 404 Not Found, 500 Internal Server Error) e, se applicabile, i dati richiesti o un messaggio di errore.

- **Richiesta GET:** Il frontend invia una richiesta GET all'endpoint `/api/users` per recuperare la lista degli utenti. Il backend risponde con una Collection di `UserResource` contenente i dati degli utenti.
- **Richiesta POST:** Il frontend invia una richiesta POST all'endpoint `/api/time-tracking/new-time` con i dettagli della timbratura. Il backend elabora i dati e restituisce un JSON con un messaggio di conferma.
- **Richiesta PUT:** Il frontend invia una richiesta PUT all'endpoint `/api/time-tracking/stop-time/$timeTrackingId` per aggiornare i dettagli di un servizio con ID `timeTrackingId`. Il backend aggiorna il servizio con i dati aggiornati.

- **Richiesta DELETE:** Il frontend invia una richiesta DELETE all'endpoint `/api/users/$id` per eliminare un utente con ID `id`. Il backend elimina l'utente e restituisce una conferma di eliminazione.

8.3.3 Gestione degli Errori e Feedback

Durante la comunicazione tra frontend e backend, è essenziale gestire correttamente gli errori e fornire feedback agli utenti. Il backend deve restituire messaggi di errore chiari e informativi quando si verificano problemi, come dati non validi o errori di connessione.

Il frontend è responsabile della visualizzazione di questi messaggi di errore all'utente in modo chiaro e comprensibile.

8.3.4 Autenticazione e Autorizzazione

La comunicazione tra frontend e backend è protetta tramite meccanismi di autenticazione e autorizzazione. Il backend richiede che il frontend invii un token di autenticazione con ogni richiesta per verificare l'identità dell'utente e determinare i permessi di accesso.

Il frontend gestisce il processo di autenticazione, memorizza il token di autenticazione nei cookie, e lo include nelle intestazioni delle richieste successive. Questo garantisce che solo gli utenti autorizzati possano accedere a determinati endpoint e operazioni.

8.3.5 Esempio di Flusso di Comunicazione

1. **Richiesta di Autenticazione:** L'utente inserisce le proprie credenziali nel frontend. Il frontend invia una richiesta POST all'endpoint `/api/login` con le credenziali.
2. **Risposta di Autenticazione:** Il backend verifica le credenziali e, se valide, restituisce un token di autenticazione.
3. **Richiesta Dati Utente:** Nelle prossime richieste, il frontend include il token di autenticazione nelle intestazioni. Ad esempio, il frontend invia una richiesta GET all'endpoint `/api/user` per ottenere i dettagli del profilo utente.
4. **Risposta Dati Utente:** Il backend verifica il token prima di restituire i dati del profilo utente. Se il token è valido, il backend restituisce i dati richiesti.

Questo flusso di comunicazione assicura che il frontend e il backend lavorino insieme in modo sinergico, offrendo un'esperienza utente fluida e sicura.

8.3.6 Sicurezza nella Gestione dei Dati Sensibili

La sicurezza dei dati sensibili, come le password degli utenti, è una priorità assoluta nel sistema. Durante la comunicazione tra frontend e backend, i dati sensibili vengono protetti attraverso diversi meccanismi di sicurezza.

Le password degli utenti non vengono mai trasmesse in chiaro. Quando un utente invia la propria password tramite il frontend, questa viene inviata al backend utilizzando una connessione sicura tramite HTTPS (Hypertext Transfer Protocol Secure). HTTPS crittografa i dati in transito, rendendo estremamente difficile per eventuali malintenzionati intercettare o manipolare le informazioni trasmesse.

Nel backend, le password non vengono mai memorizzate in formato leggibile. Al momento della registrazione o dell'aggiornamento della password, il backend utilizza la libreria `bcrypt` per eseguire l'hashing della password. L'hashing è un processo che trasforma la password in una stringa di caratteri crittografata. Ad esempio, la seguente istruzione `bcrypt.hash(body.pw, 12)` genera un hash della password con un cost factor di 12, aumentando così la complessità computazionale richiesta per un eventuale attacco di forza bruta.

- **Hashing:** Quando il backend riceve la password in chiaro dal frontend, questa viene immediatamente sottoposta a un processo di hashing utilizzando l'algoritmo `bcrypt`. Questo processo assicura che la password originale non sia mai memorizzata nel database. Al suo posto, viene salvato solo l'hash della password.
- **Verifica della Password:** Quando un utente tenta di effettuare il login, il backend confronta la password inviata dal frontend con l'hash memorizzato nel database utilizzando il metodo `bcrypt.compare`. Se la password corrisponde all'hash memorizzato, l'autenticazione viene considerata valida.

Grazie a questo approccio, anche in caso di compromissione del database, le password degli utenti rimangono protette, poiché gli hash sono difficili da invertire. Inoltre, il cost factor di `bcrypt`, configurabile nel codice (in questo caso impostato a 12), assicura che il processo di hashing sia sufficientemente lento da rendere inefficaci gli attacchi basati su tecniche di brute force.

8.4 Comunicazione Backend-Database

La comunicazione tra il backend e il database avviene principalmente attraverso query SQL, che vengono generate ed eseguite dal backend per interagire con il database MySQL. Questa interazione consente al backend di gestire dati persistenti, come informazioni sugli utenti, le timbrature e i dettagli dei servizi.

Per facilitare e semplificare le operazioni di accesso al database, è stato utilizzato un ORM (Object-Relational Mapping) chiamato **Sequelize**. Questo strumento permette di gestire il database tramite codice JavaScript, senza dover scrivere manualmente query SQL, semplificando quindi l'accesso ai dati e la loro manipolazione.

8.4.1 Operazioni CRUD tramite ORM

L'interazione backend-database segue principalmente i principi CRUD (Create, Read, Update, Delete), che rappresentano le operazioni di base per la gestione dei dati nel database.

- **Create (Creare):** Quando il backend riceve una richiesta per creare una nuova risorsa (come un nuovo utente o una nuova timbratura), esso utilizza Sequelize per generare la corrispondente query SQL di inserimento. Questa query viene poi eseguita nel database MySQL per aggiungere il nuovo record.
- **Read (Leggere):** Il backend può inviare query di lettura al database per recuperare informazioni, come una lista di utenti o le timbrature di un dipendente. Sequelize converte le richieste di lettura del backend in query SQL SELECT, e i dati recuperati vengono restituiti come oggetti JavaScript pronti per essere utilizzati nel codice.
- **Update (Aggiornare):** Per aggiornare informazioni già esistenti, come modificare il profilo di un utente o aggiornare lo stato di un servizio, il backend invia query SQL di aggiornamento al database. Sequelize gestisce queste operazioni di aggiornamento in modo sicuro, traducendo le modifiche in query SQL UPDATE, e garantendo che solo i campi necessari siano modificati.
- **Delete (Eliminare):** Quando il backend deve rimuovere dati, ad esempio cancellare un utente o eliminare una segnalazione, Sequelize genera ed esegue query SQL DELETE che rimuovono i record dal database in modo sicuro.

8.4.2 Gestione delle Relazioni tra Tabelle

Uno dei vantaggi principali dell'utilizzo di un ORM come Sequelize è la capacità di gestire facilmente le relazioni tra le tabelle del database. Il sistema richiede la gestione di relazioni complesse tra diversi tipi di dati, come utenti, servizi e timbrature. Grazie a Sequelize, il backend può definire queste relazioni a livello di codice, rappresentando legami come "uno a molti" o "molti a molti".

Ad esempio:

- **Relazione Uno-a-Molti:** Un amministratore può gestire molti dipendenti, e un dipendente può avere molte timbrature. Sequelize permette di rappresentare queste relazioni utilizzando associazioni tra modelli, come la relazione tra il modello "User" e il modello "TimeTracking".
- **Relazione Molti-a-Molti:** I clienti possono avere rapporti con più servizi, e ogni servizio può essere assegnato a più clienti. Questo tipo di relazione può essere gestito tramite tabelle ponte, che Sequelize crea e gestisce automaticamente.

8.4.3 Validazione dei Dati e Gestione degli Errori

Durante l'interazione con il database, è importante garantire che i dati siano validi e consistenti. Sequelize offre funzionalità di validazione automatica per assicurarsi che i dati inviati al database rispettino determinati vincoli, come tipi di dato corretti e la presenza di campi obbligatori.

Inoltre, Sequelize gestisce anche gli errori generati durante le operazioni con il database, come violazioni di vincoli di unicità o errori di connessione. Il backend cattura questi errori e li gestisce in modo appropriato, inviando feedback dettagliati al frontend.

8.4.4 Sicurezza e Ottimizzazione delle Query

Per garantire la sicurezza delle operazioni sul database, Sequelize include meccanismi per prevenire attacchi comuni, come l'SQL injection. Le query generate automaticamente da Sequelize sono parametrizzate, il che impedisce l'inserimento di comandi SQL malevoli all'interno delle query.

Inoltre, Sequelize è ottimizzato per generare query SQL efficienti, minimizzando l'impatto sulle prestazioni del sistema anche quando si gestiscono grandi quantità di dati. Per query più complesse o personalizzate, è comunque possibile eseguire query SQL raw (non processate) per ottenere maggiore controllo e precisione.

8.4.5 Esempio di Flusso di Comunicazione

1. **Richiesta di Creazione Dati:** Quando un utente registra una nuova timbratura, il backend crea un nuovo record nel database utilizzando una query di inserimento generata da Sequelize.
2. **Richiesta di Lettura Dati:** Il backend effettua una query di selezione per ottenere tutte le timbrature associate a un dipendente. Sequelize converte la richiesta del backend in una query SQL SELECT ed esegue la query sul database MySQL.
3. **Risposta dal Database:** I dati vengono restituiti dal database sotto forma di oggetti JavaScript, che il backend invia al frontend.

Questo flusso garantisce che il backend possa interagire in modo efficiente con il database per memorizzare e recuperare informazioni, mantenendo il sistema sicuro e performante.

Capitolo 9

Implementazione

Capitolo 10

Test

Capitolo 11

Risultati

Capitolo 12

Conclusioni

[1]

Bibliografia

- [1] F. Blaabjerg, R. Teodorescu, M. Liserre, and A.V. Timbus. Overview of control and grid synchronization for distributed power generation systems. *Industrial Electronics, IEEE Transactions on*, 53(5):1398 –1409, oct. 2006.