Università degli Studi di Parma

Corso di Laurea in Ingegneria delle Tecnologie Informatiche

Sample Manager: Web App Gestionale per Magazzino

Tecnologie Internet

Studente: Federico Dachille
Matricola: 336853

Anno Accademico 2024/2025

Luglio 2025

Indice

Capitolo 1

Introduzione

1.1 Panoramica del progetto

Il presente progetto riguarda lo sviluppo di una web application finalizzata alla gestione del campionario aziendale nel settore della calzetteria. L'obiettivo principale è fornire agli utenti autorizzati uno strumento digitale per l'inserimento, l'organizzazione, la spedizione e la consultazione dei campioni stoccati in magazzino, ottimizzando i processi interni e migliorando l'accessibilità alle informazioni.

Capitolo 2

Analisi dei Requisiti

2.1 Requisiti funzionali

- Login e autenticazione
- Gestione dei ruoli (*user*, *admin*)
- Inserimento, modifica e cancellazione di campioni (con relative immagini)
- Visualizzazione del campionario
- Gestione delle taglie dei campioni
- Posizionamento in magazzino
- Creazione di spedizioni e visualizzazione dello storico
- Modifica dati utente (username, password)
- Gestione utenti (solo admin)

2.2 Requisiti non funzionali

- Interfaccia responsive
- Resistenza a disconnessioni di server e database
- Notifiche in tempo reale
- Sicurezza e protezione delle API
- Usabilità

2.3 Casi d'uso

Sono stati definiti casi d'uso principali per: autenticazione, inserimento/ricerca campioni, gestione dettagli e taglie, creazione spedizioni, consultazione storico, gestione profilo utente e amministrazione utenti.

2.3.1 UC01 - Login e Autenticazione

Attore Primario: Utente
Attori Secondari: Nessuno

Precondizioni

- L'attore è registrato nel sistema
- Il server e il database sono raggiungibili

Postcondizioni

- L'attore è autenticato e accede alla home
- I dati dell'attore sono caricati nella sessione

Scenario Principale

1. L'attore apre la pagina di login
2. Il sistema mostra il form di login
3. L'attore inserisce username e password
4. Il sistema verifica le credenziali
5. Il sistema imposta la sessione e reindirizza alla home
6. L'attore accede all'interfaccia utente personalizzata

Scenari Alternativi

3a. Se uno dei due campi è vuoto:

    1. Il sistema mostra un messaggio di avviso

4a. Se le credenziali sono errate:

    1. Il sistema mostra un messaggio di errore

2.3.2 UC02 - Inserimento Nuovo Campione

Attore Primario: Utente
Attori Secondari: Nessuno

Precondizioni

- L'attore è autenticato
- Il database è raggiungibile

Postcondizioni

- Un nuovo campione è registrato nel sistema
- I dati e l'immagine associata sono salvati nel database

Scenario Principale

1. L'attore accede alla pagina "Aggiungi campione"
2. Il sistema mostra il modulo per l'inserimento
3. L'attore compila i campi richiesti (codice, descrizione, immagine)
4. Il sistema verifica la validità dei dati
5. Il sistema salva il nuovo campione nel database
6. Il sistema mostra una conferma e il modulo viene svuotato per un'ulteriore inserimento

Scenari Alternativi

3a. Se il codice campione è già presente nel sistema:

    1. Il sistema mostra un messaggio di errore

3b. Se l'immagine non viene caricata:

    1. Il sistema consente l'invio ma salva il campione con riferimento ad un'immagine di default (_missing_image.png)

2.3.3 UC03 - Ricerca di un campione nella pagina home

Attore Primario: Utente
Attori Secondari: Nessuno

Precondizioni

- L'attore è autenticato
- Il database è raggiungibile

Postcondizioni

- Viene identificato il campione di cui vengono inseriti i dettagli
- Una card mostra il codice del campione e la sua immagine

Scenario Principale

1. L'attore accede alla homepage
2. Il sistema mostra una barra di ricerca e tutti i campioni registrati (card con una preview del codice e dell'immagine)
3. L'attore può digitare nella barra di ricerca il codice del campione oppure una parola chiave presente nella descrizione (fuzzy search); i risultati sono aggiornati ad ogni nuovo input
4. I campioni coerenti col testo digitato vengono portati in alto nella home
5. L'attore può cliccare sul campione desiderato per vederne i dettagli completi continua in UC04 - Visualizzazione e modifica dei dettagli di un campione]

Scenari Alternativi

2a. Se non sono ancora stati inseriti campioni nel sistema:

    1. Il sistema mostra un messaggio per avvertire l'attore del fatto che il database sia vuoto

3b. Se l'attore digita il testo in modo parziale o con errori di battitura:

    1. Il sistema mostra i risultati più coerenti col testo digitato

3c. Se la ricerca non porta a risultati:

    1. Il sistema mostra un messaggio per avvertire l'attore del mancato esito della ricerca

3d. Se l'attore elimina il testo digitato nella barra di ricerca:

    1. Il sistema mostrerà le cards nell'ordine in cui erano mostrate prima della ricerca

2.3.4 UC04 - Visualizzazione e modifica dei dettagli di un campione

Attore Primario: Utente
Attori Secondari: Nessuno

Precondizioni

- L'attore è autenticato
- Il database è raggiungibile

Postcondizioni

- I dati aggiunti o aggiornati sono salvati nel database

Scenario Principale

1. L'attore accede alla pagina dei dettagli di un certo campione
2. Il sistema mostra una schermata suddivisa in 4 sezioni:
   2.1 Dettagli del campione
   2.2 Posizione in magazzino tramite schema fisico di quest'ultimo
   2.3 Aggiunta di una nuova taglia
   2.4 Tabella delle taglie presenti (con relative quantità e posizione)
3. Nella sezione [2.1] l'attore visualizza e può modificare l'immagine, il codice e la descrizione del campione
4. Nella sezione [2.2] il sistema mostra uno schema che rappresenta gli scaffali su cui è stoccata la disponibilità del campione, evidenziando le sezioni in cui sono fisicamente posizionati
5. Nella sezione [2.3] l'attore ha la possibilità di aggiungere una nuova taglia del campione, mediante un modulo d'inserimento
6. Nella sezione [2.4] il sistema mostra una tabella delle taglie presenti; ogni riga mostra la scatola in cui è contenuta una certa taglia, la quantità di questa e la sua posizione; l'attore ha la possibilità di modificare i valori
7. Il sistema salva automaticamente le modifiche e aggiorna la vista in tempo reale

Scenari Alternativi

3a. Se l'attore modifica il codice inserendone uno identico a quello attuale:

1. Il sistema mostra un messaggio di avvertimento e non permette la modifica

5d. Se l'attore prova ad inserire una quantità negativa:

1. Il sistema impedisce la digitazione di caratteri diversi dai numeri

6b. Se non sono ancora presenti taglie del campione:

1. I blocchi [2.2] e [2.4] mostrano solamente un messaggio per avvertire l'attore della mancanza di record

6c. Se sono presenti più taglie nella stessa scatola (box) e l'attore ne modifica la posizione sugli scaffali:

1. Il sistema aggiorna in automatico tutti i record della tabella inerenti alla scatola (identificata per codice)

2.3.5 UC05 - Creazione di una spedizione

Attore Primario: Utente
Attori Secondari: Nessuno

Precondizioni

- L'attore è autenticato
- Il database è raggiungibile

Postcondizioni

- La spedizione creata è aggiunta al database
- Le quantità dei campioni spediti vengono sottratte dal database
- Vengono salvati automaticamente la data/ora di creazione e l'utente che ha effettuato la spedizione

Scenario Principale

1. L'attore accede alla pagina "Crea spedizione"
2. L'attore ha la possibilità di creare una nuova spedizione
3. Il modulo d'inserimento dei dati permette di scegliere:
    3.1 Destinatario
    3.2 Nome del corriere
    3.3 Aggiungere campioni alla spedizione, di cui si scelgono:
        i. Codice
        ii. Taglia inerente al campione selezionato
        iii. Scatola da cui vengono prelevati i campioni da spedire
        iv. Quantità della taglia

Scenari Alternativi

3a. Se l'attore crea una spedizione con un solo campione con dati incompleti:

1. Il sistema mostra un messaggio per avvertire l'attore di aggiungere almeno un campione con tutti i dati

3b. Se l'attore crea una spedizione e tra i campioni aggiunti ne appaiono con dati incompleti:

1. Il sistema non li aggiunge alla spedizione

3c. Se non sono ancora presenti taglie del campione:

1. Il sistema non mostrerà opzioni selezionabili

2.3.6 UC06 - Visualizzazione della cronologia delle spedizioni

Attore Primario: Utente
Attori Secondari: Nessuno

Precondizioni

- L'attore è autenticato
- Il database è raggiungibile

Postcondizioni

- L'attore visualizza l'elenco delle spedizioni effettuate
- L'interfaccia si aggiorna dinamicamente in base ai filtri selezionati

Scenario Principale

1. L'attore accede alla pagina "Cronologia spedizioni"
2. Il sistema mostra le spedizioni registrate in ordine cronologico dalla più recente e con tutti i dettagli
3. L'attore ha la possibilità di filtrare le spedizioni per destinatario, corriere, utente che ha creato la spedizione, codice campione e data di creazione della spedizione (nessuno, uno o più filtri)
   (a) Le spedizioni che rispettano i filtri vengono portate in alto e separate visivamente da quelle che non li rispettano

Scenari Alternativi

2a. Se non sono ancora state registrate spedizioni:

1. Il sistema mostra un messaggio per avvertire l'attore della mancanza di spedizioni registrate

3a. Se non sono presenti spedizioni che rispettano i filtri:

1. Il sistema mostra un messaggio per avvertire l'attore della mancanza match coi filtri

2.3.7 UC07 - Visualizzazione e modifica dei dati utente

Attore Primario: Utente
Attori Secondari: Nessuno

Precondizioni

- L'attore è autenticato
- Il database è raggiungibile

Postcondizioni

- L'attore visualizza i propri dati
- L'attore ha la possibilità di modificare il proprio username o la propria password

Scenario Principale

1. L'attore accede alla pagina "Profilo utente"
2. Il sistema mostra un modulo in cui viene visualizzato il nome utente e una sezione per la modifica della password
3. L'attore ha la possibilità cambiare il proprio nome utente
4. L'attore ha la possibilità di cambiare la propria password

Scenari Alternativi

3a. L'attore prova a cambiare il nome utente con uno identico a quello attuale:

1. Il sistema mostra un messaggio di avvertimento e non permette la modifica

4a. L'attore prova a cambiare la password inserendo in modo errato quella attuale:

1. Il sistema mostra un messaggio di avvertimento e non permette la modifica

4b. L'attore prova a cambiare la password con una identica a quello attuale:

1. Il sistema mostra un messaggio di avvertimento e non permette la modifica

2.3.8 UC08 - Registrazione di un nuovo utente

Attore Primario: Utente
Attori Secondari: Altro utente

Precondizioni

- L'attore principale è autenticato e ha permessi "admin"
- Il database è raggiungibile

Postcondizioni

- L'attore principale registra un nuovo utente
- L'attore secondario ha ora la possibilità di accedere alla piattaforma

Scenario Principale

1. L'attore principale accede alla pagina "Aggiungi utente"
2. Il sistema mostra un modulo in cui inserire il nome utente e le autorizzazioni (user o admin) del nuovo utente da registrare
3. L'utente viene registrato col nome utente inserito e password di default "azira"
4. L'attore secondario accede alla piattaforma e modifica la propria password

Scenari Alternativi

2a. L'attore principale prova ad aggiungere un utente con uno username già registrato:

1. Il sistema mostra un messaggio di avvertimento e non permette la registrazione

2.3.9 UC09 - Gestione utenti

Attore Primario: Utente
Attori Secondari: Altri Utenti

Precondizioni

- L'attore principale è autenticato e ha permessi "admin"
- Il database è raggiungibile

Postcondizioni

- Gli utenti selezionati sono stati modificati (autorizzazioni, eliminazione o reset password)
- L'attore secondario viene notificato in tempo reale delle modifiche ricevute

Scenario Principale

1. L'attore principale accede alla pagina "Gestisci utenti"
2. Il sistema mostra un modulo in cui ad ogni riga è presente:

2.1 Nome di un utente

2.2 Relativo ruolo (modificabile)

2.3 Azioni da compiere (eliminazione o reset della password)

Scenari Alternativi

2a. L'attore principale modifica il ruolo dell'attore secondario:

1. L'attore secondario viene notificato immediatamente della modifica

2b. L'attore principale diminuisce le autorizzazioni dell'attore secondario mentre quest'ultimo è in una pagina dedicata a soli amministratori:

1. L'attore secondario viene reindirizzato alla home

2c. L'attore principale elimina l'attore secondario mentre quest'ultimo è attivo sulla piattaforma:

1. L'attore secondario viene immediatamente notificato dell'eliminazione dell'account e successivamente reindirizzato alla pagina di login

Capitolo 3

Progettazione del Sistema

3.1 Architettura del Sistema

L'applicazione è strutturata secondo un'architettura client-server, dove il frontend, sviluppato in React.js, comunica con il backend, realizzato in Express.js, attraverso un'interfaccia RESTful. Il server espone una serie di API REST accessibili tramite i metodi HTTP standard (GET, POST, PUT, DELETE). Ogni risorsa del sistema (utenti, campioni, spedizioni) è mappata su un endpoint dedicato, e le operazioni CRUD vengono eseguite in modo coerente, seguendo le convenzioni REST (metodi HTTP, codici di stato, formato JSON), e risultano stateless dal punto di vista della comunicazione, pur essendo protette da controlli di sessione lato server.

Questo approccio garantisce modularità, scalabilità e separazione delle responsabilità tra le componenti client e server. Il client React quindi invia richieste al backend per ottenere o modificare i dati, aggiornando l'interfaccia in base alla risposta.

Accanto alla struttura RESTful, il sistema utilizza anche Socket.IO per abilitare comunicazioni in tempo reale tra client e server. Questo canale WebSocket viene utilizzato per notificare in tempo reale eventuali modifiche a tutti i client connessi, in modo tale che questi possano aggiornare i dati mostrati all'utente ed evitare duplicazioni o inconsistenza di azioni eseguite parallelamente su host diversi.

In questo modo si coniuga l'efficienza e la chiarezza del modello REST per le operazioni tradizionali con la reattività e la sincronia offerte dai WebSocket, ottenendo un'architettura robusta e adatta a scenari collaborativi multiutente.

3.2 Tecnologie utilizzate

3.2.1 Frontend: React.js, HTML, CSS, JSON

Per la parte client è stato utilizzato React.js, un framework moderno e diffuso per lo sviluppo di interfacce utente dinamiche. La scelta è motivata dalla necessità di una

gestione dello stato locale e globale reattiva e di un'interfaccia utente reattiva e responsive.

React consente di aggiornare la vista in modo efficiente al variare dei dati, senza necessità di ricaricare la pagina. Inoltre, la sua integrazione con strumenti come react-select, react-router-dom e l'uso dei React Hooks semplifica lo sviluppo mantenendo il codice modulare e leggibile.

Alla base di React vi sono le tecnologie HTML e CSS, utilizzate rispettivamente per la struttura e lo stile dell'interfaccia. L'HTML definisce la struttura semantica dei componenti, mentre il CSS è stato impiegato sia tramite file modulari che attraverso stili inline, curando anche la responsiveness per l'adattamento a tutti i formati di schermi.

Infine, per lo scambio dei dati tra client e server è stato utilizzato il formato JSON, nativamente supportato in JavaScript e impiegato come corpo dei messaggi in tutte le comunicazioni RESTful.

3.2.2 Backend: Express.js (Node.js)

Il backend è stato implementato con Express.js, un framework per Node.js. La scelta è dovuta alla sua flessibilità nella definizione delle route, al supporto nativo per le richieste asincrone, e alla compatibilità con librerie come bcrypt, express-session e cors.

Express permette di strutturare un server RESTful scalabile, che gestisce le API in modo efficiente, mantenendo separata la logica di business dalla logica di routing.

3.2.3 Database: MySQL

Il database scelto è MySQL, in quanto si adatta perfettamente alla struttura del sistema. I dati gestiti dall'applicazione (utenti, campioni, spedizioni, taglie, posizioni in magazzino) presentano relazioni ben definite e vincoli logici (es. un campione può avere più taglie, una spedizione è composta da più campioni, ecc.), che si modellano intuitivamente con tabelle collegate da chiavi primarie ed esterne. Questa struttura relazionale favorisce:

- Integrità referenziale, tramite vincoli tra entità
- Facilità nella progettazione del modello dati
- Semplicità di interrogazione con SQL, utile per filtrare, aggregare e unire dati

A differenza di un sistema NoSQL, dove la mancanza di schema rigido può introdurre inconsistenze o richiedere duplicazione dei dati, MySQL garantisce coerenza e controllo centralizzato delle relazioni, elementi fondamentali per un sistema di gestione strutturato come quello richiesto in questo progetto.

3.2.4 Comunicazione Real-Time: Socket.IO

Per la gestione delle notifiche in tempo reale tra utenti è stata adottata Socket.IO, una libreria che fornisce un canale di comunicazione WebSocket bidirezionale tra client e server.

Questa scelta consente al sistema di garantire sincronia tra le interfacce utente distribuite: ad esempio, la creazione o modifica di una spedizione viene propagata immediatamente a tutti gli utenti connessi, senza necessità di ricaricare la pagina.

Socket.IO offre un moderato utilizzo di banda e reattività immediata, due requisiti chiave per una web app destinata a uso collaborativo e simultaneo.

3.3 Modello dei dati

3.3.1 Diagramma ER del Database e Osservazioni

Il modello dei dati utilizzato si basa su un database relazionale, implementato tramite MySQL. Le entità principali del sistema sono: utenti, campioni, taglie dei campioni, scaffali, spedizioni e dettagli delle spedizioni.

Il diagramma E-R seguente rappresenta la struttura logica del database, evidenziando le relazioni tra le tabelle.

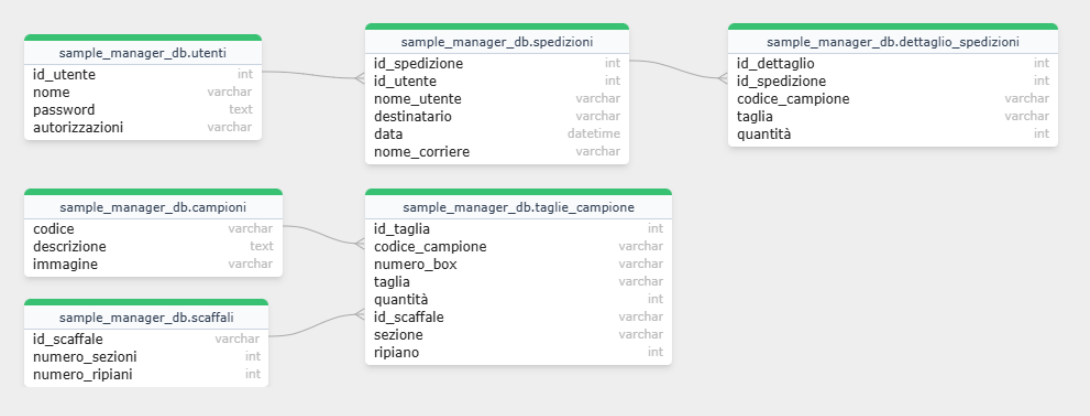

Figura 3.1: Diagramma E-R del Database

Sebbene il database sia costruito su basi relazionali, non è completamente normalizzato. Questa scelta è stata adottata intenzionalmente per privilegiare la semplicità del codice lato server e l'efficienza operativa. In particolare:

- Il campo numero_box è trattato come attributo delle taglie, senza separare le scatole come entità autonome.
- Il campo nome_utente nella tabella spedizioni è un attributo e non una foreign key su utenti, per semplificare le query.

- I campi codice_campione e taglia in dettaglio_spedizioni sono memorizzati direttamente come stringhe e non come riferimenti a chiavi primarie, per ridurre le necessità di join multiple nelle interrogazioni frequenti.

Nonostante questa parziale denormalizzazione, il database si dimostra efficace per lo scenario applicativo previsto, evitando inconsistenze attraverso controlli lato server e mantenendo buone prestazioni.

Capitolo 4

Implementazione

4.1 Struttura del codice

Il codice è suddiviso in due directory principali: client (frontend React) e server (backend Express/Socket.IO).

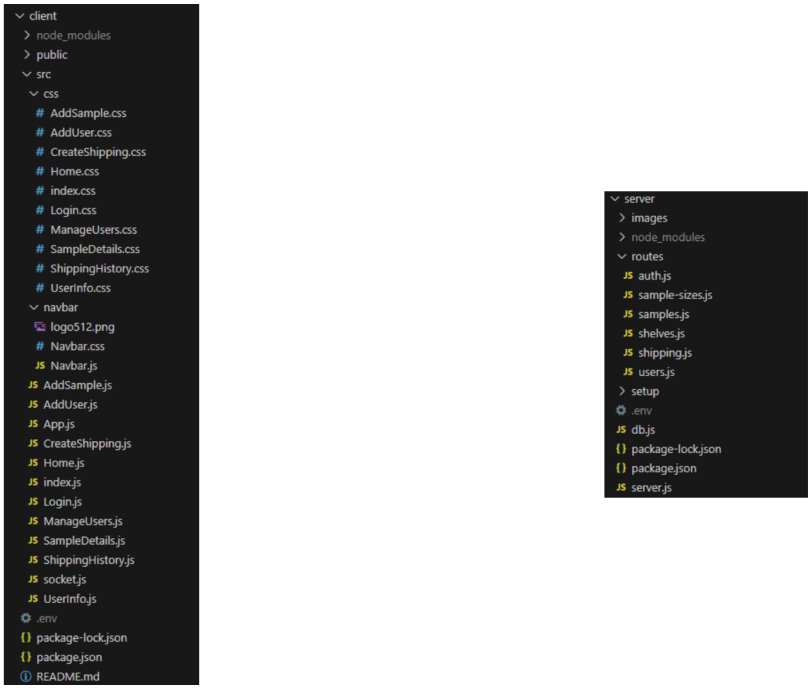

Figura 4.1: Struttura delle due directory

4.1.1 Client

La directory client contiene il codice e le dipendenze React relative al frontend dell'applicazione.

Le principali sub-directory e file sono i seguenti:

- **node_modules/** → raccoglie tutte le dipendenze installate, necessarie al funzionamento dell'applicazione React.
- **public/** → contiene i file statici accessibili direttamente dal browser (ad esempio index.html, immagini, favicon).
- **package.json / package-lock.json** → definiscono le dipendenze del progetto e i metadati dell'applicazione Node.js.
- **.env** → file di configurazione con le variabili d'ambiente.
- **src/** → include il codice sorgente dell'applicazione (.js e .css):
  - **css/** → directory che raccoglie i file CSS per la definizione della parte grafica.
  - **navbar/** → directory che contiene il codice relativo alla barra di navigazione condivisa tra tutte le pagine della web app.
  - **App.js** → componente principale che definisce la struttura generale dell'applicazione.
  - **index.js** → punto di ingresso dell'app React, che renderizza il componente App all'interno di index.html.
  - **socket.js** → modulo che inizializza la connessione WebSocket verso il backend, utilizzando l'indirizzo specificato nel file .env. Si basa sulla libreria socket.io-client, abilita l'invio automatico dei cookie e delle credenziali di sessione per supportare l'autenticazione e gestisce eventi standard della connessione (connect, disconnect, connect_error).
  - **AddSample.js** → pagina per l'aggiunta di un nuovo campione nel sistema.
  - **AddUser.js** → pagina per la registrazione di nuovi utenti, accessibile esclusivamente agli amministratori.
  - **CreateShipping.js** → pagina per la creazione del log di una spedizione, con dettagli relativi ai campioni spediti, alle quantità, al destinatario, al corriere e alla data.
  - **Home.js** → pagina principale della web app, che mostra l'elenco dei campioni disponibili.
  - **Login.js** → pagina di accesso alla web app, che consente all'utente di autenticarsi.
  - **ManageUsers.js** → pagina per la gestione degli utenti (solo per amministratori), che consente di modificare i permessi, reimpostare la password a quella di default o eliminare account.

  - **SampleDetails.js** → pagina che mostra i dettagli completi di un campione e permette di aggiungere e gestire le relative taglie.
  - **ShippingHistory.js** → pagina che elenca le spedizioni effettuate, con i relativi dettagli.
  - **UserInfo.js** → pagina che consente all'utente di modificare il proprio username e la propria password.

4.1.2 Server

La directory server contiene il codice e le dipendenze Node.js/Express per l'implementazione della logica backend dell'applicazione.

Le principali sub-directory e file sono i seguenti:

- **images/** → directory utilizzata per il salvataggio delle immagini caricate (foto dei campioni).
- **node_modules/** → raccoglie tutte le dipendenze installate, necessarie al funzionamento dell'applicazione Node/Express.
- **routes/** → contiene i moduli con le rotte REST del backend, suddivisi per ambito funzionale:
  - **auth.js** → gestisce l'autenticazione degli utenti (login, logout, verifica sessione).
  - **users.js** → gestisce gli utenti: registrazione, eliminazione, reset password, cambio ruolo e aggiornamento delle credenziali.
  - **samples.js** → gestisce i campioni (creazione, lettura, modifica e cancellazione), con supporto al caricamento immagini e notifiche in tempo reale tramite WebSocket.
  - **sample-sizes.js** → gestisce le taglie associate ai campioni, comprese le operazioni di aggiunta, modifica, eliminazione e propagazione degli aggiornamenti tra taglie collegate.
  - **shelves.js** → fornisce le API per il recupero degli scaffali dal database.
  - **shipping.js** → gestisce le spedizioni: creazione di nuovi log, aggiornamento delle quantità e recupero dello storico.
- **setupDB.js** → script che inizializza il database: crea le tabelle a partire dal file SQL, inserisce gli scaffali iniziali e crea l'utente amministratore di default.
- **sample_manager_db.sql** → script SQL che definisce lo schema del database (tabelle campioni, scaffali, taglie, utenti, spedizioni e dettagli spedizioni).

- **db.js** → modulo per la connessione al database MySQL. Gestisce la riconnessione automatica in caso di disconnessione e notifica i client tramite WebSocket sullo stato del DB.
- **server.js** → file principale del backend: configura l'applicazione Express, la gestione delle sessioni utente, l'inizializzazione di Socket.IO, il caricamento delle rotte e l'avvio del server HTTP sulla porta 5000.
- **package.json / package-lock.json** → definiscono le dipendenze del progetto e i metadati dell'applicazione Node.js.
- **.env** → file di configurazione con le variabili d'ambiente (credenziali DB, secret per le sessioni, URL del client).

4.2 Aspetti Tecnici Rilevanti

Nel backend sono state adottate diverse soluzioni per garantire robustezza, coerenza e interattività in tempo reale. I principali aspetti tecnici sono riportati di seguito.

4.2.1 Connessione in tempo reale (Socket.IO)

È stato integrato un sistema di notifiche WebSocket per mantenere il frontend sincronizzato con le operazioni di backend. Vengono emessi eventi dedicati per campioni (sample-added, sample-updated, sample-deleted), taglie (size-added, size-updated, size-deleted), spedizioni (shipping-created), utenti (user-added, user-renamed, user-deleted, role-updated) e stato del database (db-status). È stata inoltre prevista una mappatura utente → socket per permettere notifiche mirate, come nel caso del logout forzato di un utente eliminato.

4.2.2 Gestione resiliente del database

La connessione a MySQL è stata resa resiliente tramite un meccanismo di riconnessione automatica in caso di caduta. Lo stato della connessione viene notificato a tutti i client tramite eventi WebSocket.

4.2.3 Propagazione automatica delle modifiche "alle box"

In caso di aggiornamento di una taglia appartenente a una determinata box, la modifica viene propagata automaticamente a tutte le taglie con lo stesso numero_box. Successivamente, viene emesso un evento size-updated per ogni riga aggiornata, garantendo coerenza dei dati tra backend e frontend. È stato necessario introdurre questa soluzione poiché,

nel modello dati adottato, le box non sono gestite come entità indipendenti ma come proprietà delle taglie.

4.2.4 Gestione delle spedizioni e aggiornamento inventario

La creazione di una spedizione segue una sequenza strutturata: inserimento della spedizione, inserimento dei dettagli, decremento delle quantità in magazzino e notifica degli aggiornamenti al frontend. Durante questo processo, vengono emessi eventi WebSocket per aggiornare sia le quantità residue delle taglie coinvolte sia lo stato complessivo della spedizione.

4.2.5 Gestione delle immagini

Il caricamento delle immagini è stato gestito tramite multer, con salvataggio su filesystem e naming deterministico (codice_timestamp.ext). In caso di aggiornamento o cancellazione di un campione, le immagini obsolete vengono eliminate automaticamente, evitando la persistenza di file non più utilizzati.

Le immagini non vengono servite tramite API REST, ma sono esposte come file statici attraverso Express. Una volta salvate nella cartella /images, possono essere richieste dal frontend tramite URL dedicati (ad esempio http://localhost:5000/images/nomefile.ext). Questa soluzione semplifica la gestione e l'accesso ai contenuti multimediali, mantenendo le API REST focalizzate esclusivamente sui dati strutturati.

4.2.6 Autenticazione e autorizzazione

L'autenticazione è basata su sessione con hashing delle password tramite bcrypt. Un endpoint dedicato consente la verifica della sessione attiva. Il grado di autorizzazioni di un utente è gestito tramite ruoli (user, admin) e le operazioni riservate agli amministratori sono protette da controlli server-side. In caso di cambiamenti di ruolo o eliminazione di un utente, vengono inviati eventi in tempo reale al relativo socket.

4.2.7 Configurazione e deploy

Le informazioni sensibili (es. credenziali DB, secret delle sessioni, configurazione CORS) sono centralizzate nel file .env. Il server accetta richieste CORS con credenziali esclusivamente dall'origine configurata o da localhost:3000. È inoltre presente un file README con istruzioni per il setup e l'esecuzione locale del sistema.

4.3 Flusso di utilizzo dell'applicazione

Di seguito è descritto, a livello logico, il flusso tipico di utilizzo dal punto di vista di browser, frontend React, backend Express e database.

4.3.1 Apertura dell'applicazione e setup iniziale

1. L'utente raggiunge la web app via browser (origin frontend, es. http://192.168.X.Y:3000).
2. Il frontend invia richieste HTTP al backend (origin differente, es. http://192.168.X.Y:5000) con CORS abilitato e credenziali (cookie di sessione). Il backend accetta solo le origini previste e abilita le credenziali.
3. In parallelo viene stabilita la connessione WebSocket (Socket.IO) verso il backend; all'apertura, il server registra la connessione e mantiene una mappa utente→socket per eventuali notifiche mirate.
4. Lo stato di disponibilità del DB viene notificato in tempo reale ai client tramite evento db-status (es. in caso di disconnessione/riconnessione), così il frontend può mostrare un overlay UI se necessario.

4.3.2 Autenticazione e sessione

1. L'utente effettua il login inserendo credenziali; il backend verifica l'utente e la password (bcrypt) e, in caso positivo, crea la sessione server-side e imposta il cookie di sessione.
2. Un endpoint dedicato consente di verificare lo stato della sessione e riallineare i dati utente (id, nome, ruolo) nel client.
3. Dopo il login, il client può inviare al server l'identificativo utente sul canale WebSocket (evento di "register") per abilitare notifiche mirate (es. cambio ruolo, logout forzato).

4.3.3 Consultazione dei dati (campioni, taglie, scaffali, spedizioni)

1. Il frontend interroga gli endpoint REST per elenchi e dettagli:
   - Campioni (con immagini servite come statiche).
   - Taglie per campione.
   - Scaffali, sezioni e ripiani.
   - Storico spedizioni con relativi dettagli.

4.3.4 Aggiornamenti in tempo reale (push dal backend)

1. Le operazioni CRUD lato backend emettono eventi WebSocket dedicati (es. sample-added, size-updated, shipping-created). Il frontend riceve l'evento e aggiorna la UI senza ricaricare la pagina.

2. Per eventi che riguardano un singolo utente (es. cambio ruolo, eliminazione account), il server invia notifiche mirate al socket registrato.

4.3.5 Gestione campioni e immagini

1. In creazione/modifica, eventuali immagini del campione vengono caricate via multer e salvate su filesystem.

2. In aggiornamento/eliminazione, l'immagine precedente viene rimossa se non di default.

4.3.6 Creazione spedizione e aggiornamento inventario

1. Il backend inserisce i dettagli della spedizione; per ogni riga, decrementa la quantità su taglie_campione.

2. Dopo ciascun decremento la riga aggiornata viene emessa (size-updated); al termine viene inviato al frontend l'evento shipping-created.

4.3.7 Amministrazione utenti (solo admin)

1. Gli endpoint di gestione utenti (lista, cambio ruolo, reset password, eliminazione) sono protetti da controlli server-side sul ruolo.

2. In caso di cambio ruolo o eliminazione, il server notifica in tempo reale l'utente interessato.

4.3.8 Logout e chiusura sessione

1. Su richiesta di logout, la sessione server-side viene distrutta e il cookie invalidato. Se l'utente chiude o ricarica la pagina dopo il logout anche il WebSocket verrà chiuso (il server riceve l'evento disconnect e rimuove l'associazione utente→socket dalla mappa), altrimenti il server non forza subito la disconnessione.

4.4 Sicurezza, Integrità e Autenticazione

L'autenticazione degli utenti si basa su credenziali composte da username e password, che non vengono mai memorizzate in chiaro nel database: al momento della registrazione o del reset, le password vengono cifrate tramite l'algoritmo di hashing bcrypt. Il backend gestisce le sessioni con il middleware express-session, che associa all'utente un cookie di sessione e mantiene i dati relativi lato server; un endpoint dedicato consente inoltre al frontend di verificare in ogni momento lo stato della sessione.

Ogni utente è caratterizzato da un ruolo, che può essere user o admin. Le operazioni sensibili, come la gestione degli utenti, il cambio ruolo, il reset della password o l'eliminazione di un account, sono consentite esclusivamente agli amministratori, con controlli eseguiti lato server.

Le API integrano controlli di validazione sugli input per prevenire l'inserimento di dati incompleti o non validi, rispondendo con codici HTTP appropriati e messaggi esplicativi in caso di errore. Sono inoltre previsti controlli specifici per evitare duplicazioni, come la creazione di utenti già esistenti, e per mantenere consistenza nei dati, ad esempio con la propagazione automatica delle modifiche alle taglie.

Le informazioni sensibili, come le credenziali di accesso al database, le secret per la firma delle sessioni e le origini autorizzate per le richieste CORS, sono esternalizzate in un file .env. A livello di comunicazioni client-server, il backend abilita CORS unicamente verso l'origine del frontend (specificata nel .env), e consente l'invio delle credenziali necessarie per mantenere la sessione autenticata.

Capitolo 5

Testing

5.1 Strategia di Testing

Per il progetto non sono state adottate tecnologie di testing automatico. Trattandosi di un'applicazione gestionale, caratterizzata da un insieme limitato e ben definito di interazioni, è stata adottata una strategia di testing manuale.

Ogni funzionalità è stata verificata singolarmente, controllando che ad ogni azione corrispondesse l'effetto atteso sia lato frontend (come l'aggiornamento dell'interfaccia, messaggi di conferma o errore) sia lato backend (come la corretta modifica dei dati nel database e la coerenza degli stati mantenuti tramite WebSocket), anche con l'aiuto del logging.

In aggiunta, il progetto è stato utilizzato dall'azienda interessata, che ha fornito feedback diretti durante l'utilizzo. Questi riscontri hanno permesso di individuare eventuali problemi non emersi nei test iniziali e di migliorare l'usabilità, l'affidabilità e la coerenza del sistema.

Capitolo 6

Configurazione e Deployment

6.1 Ambiente di Sviluppo

Lo sviluppo del progetto è stato realizzato utilizzando Visual Studio Code come ambiente di programmazione principale, mentre per la gestione del database MySQL in locale è stato utilizzato MAMP.

L'ambiente di sviluppo è stato quindi costituito da:

- Frontend: React avviato in locale sulla porta 3000.
- Backend: Node.js con Express avviato in locale sulla porta 5000.
- Database: MySQL fornito da MAMP.

6.2 Ambiente di Produzione

L'ambiente di produzione è stato realizzato utilizzando un PC con sistema operativo Linux Mint configurato come server applicativo e database.

Questa scelta è risultata adeguata in quanto l'applicazione è destinata a un numero limitato di utenti interni all'azienda e non richiede quindi un'infrastruttura cloud o server dedicati di fascia alta.

Sul server sono stati installati:

- Node.js per l'esecuzione del backend basato su Express in ascolto sulla porta 5000.
- MySQL per la gestione del database.
- Frontend React distribuito in versione di produzione e servito dallo stesso server, così da rendere accessibile la web app tramite browser ai client della rete locale.

Questa configurazione consente di utilizzare la web app direttamente all'interno della rete aziendale, garantendo prestazioni sufficienti e semplicità di manutenzione, senza la necessità di un'infrastruttura più complessa.

6.3 Istruzioni per l'installazione

Le istruzioni complete sono riportate nel file README.md della repository GitHub del progetto: `https://github.com/fededachille/sample-manager`.

6.4 Dipendenze

Il progetto utilizza diverse librerie e framework per la gestione del frontend e del backend. Di seguito sono riportate le principali dipendenze adottate.

6.4.1 Frontend

- React: framework JavaScript utilizzato per la costruzione dell'interfaccia utente e per la logica frontend
- React Router: gestione della navigazione tra le pagine della web app.
- Socket.IO client: gestione della comunicazione in tempo reale con il server tramite WebSocket.
- Fuse.js: libreria utilizzata per implementare la ricerca fuzzy nella homepage.

6.4.2 Backend

- Express: framework Node.js per la gestione delle API REST.
- Socket.IO: per la comunicazione in tempo reale con i client.
- express-session: per la gestione delle sessioni utente tramite cookie.
- bcrypt: per l'hashing sicuro delle password.
- mysql2: driver per la connessione al database MySQL.
- multer: libreria per la gestione dell'upload delle immagini.
- dotenv: caricamento delle variabili di configurazione da file .env.

Capitolo 7

Valutazione e Risultati

Di seguito viene riportata la verifica dei requisiti funzionali e non funzionali individuati nella fase di analisi (2).

7.1 Verifica dei Requisiti

7.1.1 Requisiti funzionali

- Login e autenticazione → È presente una pagina di login che verifica le credenziali, crea una sessione lato server e consente l'accesso all'applicazione.
- Gestione dei ruoli (user e admin) → Ogni utente è associato a un ruolo, e le operazioni riservate agli amministratori sono protette da controlli lato server.
- Inserimento, modifica e cancellazione di campioni (con immagini) → È possibile gestire i campioni in tutte le operazioni CRUD, con caricamento e sostituzione delle immagini.
- Visualizzazione del campionario → La home page mostra l'elenco dei campioni disponibili con le relative informazioni.
- Gestione delle taglie dei campioni → Ogni campione può avere più taglie, con possibilità di aggiunta, modifica ed eliminazione.
- Posizionamento in magazzino → Il sistema offre un meccanismo di visualizzazione della posizione fisica dei campioni all'interno del magazzino.
- Creazione di spedizioni e visualizzazione di spedizioni passate → È possibile registrare nuove spedizioni, aggiornare le quantità in magazzino e consultare lo storico.
- Modifica dei propri dati utente (username e password) → Ogni utente autenticato può aggiornare le proprie credenziali tramite un'apposita sezione.

- Gestione utenti (solo admin) → L'amministratore può creare nuovi account, modificare i ruoli, resettare password o eliminare utenti.

7.1.2 Requisiti non funzionali

- Interfaccia responsive → L'interfaccia è stata sviluppata con CSS responsive, garantendo usabilità anche su dispositivi mobili.
- Resistenza alla disconnessione da server e database → Il backend gestisce la riconnessione al database e notifica ai client lo stato tramite overlay.
- Notifiche in tempo reale → Ogni operazione rilevante (aggiunta, modifica, eliminazione di entità) viene notificata ai client in tempo reale tramite WebSocket.
- Sicurezza e protezione delle chiamate API → Le password sono protette con bcrypt, le sessioni sono gestite lato server, e le API sono protette da controlli di ruolo e da CORS con credenziali.
- Utilizzo intuitivo per l'utente → L'interfaccia è stata progettata per essere semplice, con feedback immediati e messaggi di conferma o errore chiari.

7.2 Possibili miglioramenti

Durante lo sviluppo e l'utilizzo del sistema sono emerse alcune limitazioni che, pur non compromettendo il funzionamento generale, potrebbero essere affrontate in sviluppi futuri per aumentarne robustezza, scalabilità e usabilità.

- Assenza di test automatici: il testing è stato effettuato manualmente e tramite feedback dell'azienda. Per aumentare l'affidabilità e semplificare future modifiche, sarebbe utile integrare strumenti di test automatici.
- Deployment su singolo PC: l'ambiente di produzione è basato su un unico PC con Linux Mint. Sebbene sufficiente per l'uso corrente, la disponibilità e la scalabilità sono limitate.
- Gestione delle box come proprietà delle taglie: la scelta di modellare le box come attributo delle taglie ha reso necessaria la propagazione manuale delle modifiche. Una soluzione alternativa potrebbe essere l'introduzione di una tabella dedicata alle box, così da ridurre la logica applicativa, semplificare le query e garantire un database maggiormente normalizzato.
- Resilienza della connessione al database: il sistema è già in grado di gestire la riconnessione in caso di caduta, ma al momento i client vengono solo notificati con

un overlay. Un miglioramento possibile sarebbe l'introduzione di un sistema di memorizzazione locale temporanea delle operazioni, così da consentirne l'esecuzione automatica una volta ristabilita la connessione.

- Limitata gestione dello storico delle spedizioni: il sistema consente la sola consultazione delle spedizioni registrate, senza possibilità di modificarle o eliminarle.

Capitolo 8

Conclusioni

8.1 Riassunto del lavoro svolto

Il progetto ha avuto come obiettivo la realizzazione di una web application gestionale per la gestione del campionario in magazzino.

Durante lo sviluppo sono stati affrontati e raggiunti i seguenti traguardi principali:

- Implementazione del frontend in React, con interfaccia responsive e usabilità adatta al contesto aziendale.
- Sviluppo del backend in Node.js con Express, strutturato in moduli e arricchito da notifiche in tempo reale tramite Socket.IO.
- Gestione del database MySQL, con schema relazionale, vincoli di integrità e script di setup automatizzati.
- Autenticazione e autorizzazione basate su sessioni, con ruoli differenziati (user e admin) e protezione delle password tramite hashing con bcrypt.
- Gestione avanzata dei campioni e delle taglie, con propagazione automatica delle modifiche.
- Creazione e registrazione delle spedizioni, con aggiornamento automatico delle quantità a magazzino e storico consultabile.
- Caricamento e gestione delle immagini dei campioni, salvate su filesystem ed esposte tramite servizio statico.
- Deployment in ambiente aziendale, con configurazione su un server Linux Mint dedicato.
- Testing manuale e tramite feedback dell'azienda, che ha consentito di validare le funzionalità e migliorare l'usabilità del sistema.

Il lavoro svolto ha permesso di ottenere un'applicazione completa, funzionante e già operativa in un contesto reale, soddisfacendo i requisiti funzionali e non funzionali individuati in fase di analisi.

8.2 Competenze acquisite e considerazioni finali

Lo sviluppo di questo progetto ha rappresentato un percorso impegnativo ma estremamente formativo. Partendo dai requisiti iniziali, è stato possibile realizzare un'applicazione gestionale completa, capace di rispondere alle esigenze concrete dell'azienda.

Durante la progettazione e l'implementazione sono state affrontate sfide significative, come la definizione di uno schema dati coerente, la gestione della comunicazione in tempo reale, l'autenticazione basata su sessione e la propagazione automatica delle modifiche. Il superamento di questi ostacoli ha permesso di acquisire maggiore padronanza nello sviluppo full-stack e nella gestione dei problemi legati all'integrazione tra frontend, backend e database.

Il confronto diretto con l'azienda ha aggiunto ulteriore valore: i feedback ricevuti hanno consentito di migliorare l'usabilità, correggere criticità e adattare il sistema alle reali esigenze operative. Questo aspetto ha inoltre reso l'esperienza molto vicina al mondo professionale.

In conclusione, il progetto ha consentito non solo di consolidare conoscenze tecniche, ma anche di sviluppare competenze trasversali come problem solving, organizzazione del lavoro e comunicazione con gli utenti finali.