Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative

SUPSI

# Applicazione per l'acquisizione dati di un operatore manifatturiero

Studenti
Peter Catania
Marsildo Byketa

Relatore
Giuseppe Landolfi

Correlatore
Samuele Dell'Oca
Michel Rosselli

Committente
Giuseppe Landolfi

Corso di laurea
Ingegneria Informatica TP

Project Code
C10614

Anno
2022/2023

Data 08/03/23

# Indice

# Indice Delle figure

# Abstract (Italiano)

La raccolta di dati sullo stato psicofisico dell'operatore durante il processo produttivo può fornire informazioni preziose per migliorare le prestazioni e la sicurezza sul lavoro nell'ambito dell'Industry 5.0, un paradigma di produzione basato sulle risorse umane e sulla collaborazione tra uomo e macchina.

Il progetto ha come obiettivo lo sviluppo di un'applicazione in grado di acquisire dati dall'operatore manifatturiero durante le sue attività lavorative per alimentare un modello di Machine Learning esistente e integrarsi con le API della piattaforma HDT. L'obiettivo è di costruire un'applicazione che permetta agli operatori manifatturieri di inserire facilmente e in modo intuitivo i dati riguardanti la loro fatica e le attività svolte durante il lavoro, mantenendo al contempo un'interfaccia semplice e familiare.

L'applicazione è progettata per garantire la raccolta periodica di informazioni sulla fatica percepita dagli operatori durante l'esecuzione di specifiche attività. Questo viene fatto inviando notifiche push all'utente per richiedere i dati. Inoltre, l'applicazione registra il momento in cui un utente inizia e finisce un'attività e consente di salvare le informazioni raccolte in locale, garantendo la continuità nella raccolta dei dati anche in assenza di connessione o in caso di indisponibilità del server.

L'obiettivo principale del progetto è quello di fornire una soluzione efficiente per la raccolta di dati sull'operatore manifatturiero e sulle sue attività lavorative, che possono essere utilizzati per migliorare le prestazioni e la sicurezza sul lavoro. L'applicazione sarà facile da usare e fornirà una grande quantità di informazioni utili per il miglioramento continuo del processo produttivo.

# Abstract (Inglese)

Collecting data on the psychophysical state of the operator during the manufacturing process can provide valuable information to improve performance and workplace safety in the context of Industry 5.0, a manufacturing paradigm based on human resources and human-machine collaboration.

The project aims to develop an application that can capture data from the manufacturing operator during their work activities to feed an existing Machine Learning model and integrate with the HDT platform API. The goal is to build an application that allows manufacturing operators to easily and intuitively enter data regarding their fatigue and activities performed while working, while maintaining a simple and familiar interface.

The application is designed to ensure periodic collection of information about operators' perceived fatigue while performing specific tasks. This is done by sending push notifications to the user to request the data. In addition, the application records the time when a user starts and finishes a task and allows the collected information to be saved locally, ensuring continuity in data collection even when there is no connection or when the server is unavailable.

The main goal of the project is to provide an efficient solution for collecting data on the manufacturing operator and his work activities, which can be used to improve performance and safety at work. The application will be easy to use and provide a wealth of useful information for continuous improvement of the manufacturing process.

# Progetto Assegnato

Il progetto assegnato prevede lo sviluppo di un'applicazione per l'acquisizione di dati da un operatore di produzione. L'applicazione deve essere in grado di raccogliere informazioni relative al tipo di attività svolta dall'operatore e al suo stato psicofisico durante il lavoro. I dati raccolti saranno utilizzati per alimentare un modello di Machine Learning esistente e per integrarsi con le API di una piattaforma HDT.

Gli obiettivi principali del progetto sono:

- Sviluppare un'applicazione in grado di interagire facilmente con gli operatori del settore manifatturiero durante le loro attività lavorative.
- Raccogliere dati relativi al tipo di attività svolta dall'operatore e al suo stato psicofisico.

Per raggiungere questi obiettivi, i membri del team di sviluppo collaboreranno con i membri del progetto madre. Durante lo sviluppo dell'applicazione, i membri del team terranno conto delle esigenze dell'operatore di produzione, garantendo un'interazione facile e non invasiva, anche in condizioni di lavoro particolari. Inoltre, l'applicazione deve essere progettata per integrarsi con il modello di apprendimento automatico esistente e con le API della piattaforma HDT.

# Introduzione

La documentazione del progetto segue un approccio logico e strutturato per descrivere il lavoro svolto. Nel primo capitolo, viene presentato il problema e lo scopo del progetto, fornendo una motivazione dettagliata e spiegando il contesto in cui si inserisce.

Il secondo capitolo è dedicato alla valutazione delle varie alternative per risolvere il problema, con particolare attenzione alle scelte fatte riguardo ai dispositivi mobili. Qui viene descritta la prima attività di ricerca svolta per capire i vantaggi e gli svantaggi dei dispositivi mobili disponibili sul mercato.

Nel terzo capitolo, vengono elencate le scelte effettivamente fatte e viene spiegato il motivo per cui si è optato per quelle specifiche. Nel quarto capitolo, viene presentata la fase di progettazione del sistema, descrivendo in dettaglio come è stato pianificato e organizzato il lavoro per raggiungere gli obiettivi prefissati.

Il quinto capitolo si concentra sull'implementazione del sistema, descrivendo in dettaglio ciò che è stato ottenuto con lo sviluppo. Nel sesto capitolo, viene spiegato come effettuare il deploy del sistema, sia per la parte di frontend che per quella del backend.

Infine, il settimo e ultimo capitolo fornisce le conclusioni del lavoro svolto, facendo il punto della situazione con i risultati ottenuti e discutendo le considerazioni personali del team, i limiti del sistema e i possibili sviluppi futuri. La documentazione si conclude con una valutazione finale del progetto.

# 1 Motivazione e Contesto

Durante il 6 semestre del percorso di Bachelor in ingegneria informatica presso la SUPSI, gli studenti sono chiamati a svolgere un progetto di semestre per prepararsi al lavoro di diploma.

Il progetto è stato scelto tra le proposte fornite dal team che ha sviluppato la piattaforma HDT. L'obiettivo di questo progetto è di sviluppare un'applicazione in grado di acquisire dati dall'operatore manifatturiero durante le sue attività lavorative, al fine di alimentare un modello di Machine Learning esistente e integrarsi con le API della piattaforma HDT.

La raccolta di dati sullo stato psicofisico dell'operatore e sull'efficienza del processo produttivo può fornire informazioni preziose per migliorare le prestazioni e la sicurezza sul lavoro. Perciò è importante garantire la qualità e la periodicità dei dati raccolti.

Il team di sviluppo ha quindi come obbiettivo una stretta collaborazione con il team che ha sviluppato la piattaforma HDT per garantire la realizzazione di un'applicazione efficiente e facile da utilizzare, in grado di soddisfare le esigenze specifiche del contesto produttivo.

Nell'ambiente di lavoro degli operatori manifatturieri, ci sono diverse variabili da tenere in considerazione per la raccolta di dati. In primo luogo, l'ambiente può essere rumoroso, con macchinari in funzione e altri rumori di fondo che possono interferire con l'acquisizione dei dati. Inoltre, gli operatori potrebbero indossare guanti, tute o altri indumenti protettivi che potrebbero rendere difficile la raccolta dei dati tramite metodi tradizionali, come l'uso di tastiere o schermi touch.

## 1.1 Requisiti

Prima di poter accedere alle funzionalità dell'applicazione, l'utente deve essere registrato nel sistema attraverso l'assegnazione di un UUID. Ci sono tre modi per registrarsi: attraverso un UUID generato al momento della registrazione, scegliendo un UUID già presente nel sistema o inserendo un UUID valido per la registrazione.

L'obiettivo principale dell'applicazione è la raccolta delle informazioni sulla fatica percepita dagli operatori durante l'esecuzione di specifiche attività. Per garantire una raccolta periodica di queste informazioni, l'applicazione invierà una notifica push all'utente. Inoltre, nell'applicazione è prevista la registrazione del momento in cui un utente inizia e finisce un'attività.

È anche richiesto che l'applicazione sia in grado di salvare la raccolta di informazioni raccolte dagli operatori in maniera locale, poiché può succedere spesso che non ci sia connessione all'interno dell'ambiente di lavoro. In questo modo, le informazioni raccolte saranno comunque registrate sul dispositivo e verranno inoltrate al server quando la connessione sarà di nuovo disponibile.

## 1.2 Obbiettivo

L'obiettivo del progetto è di costruire un'applicazione che permetta agli operatori manifatturieri di inserire facilmente e in modo intuitivo i dati riguardanti la loro fatica e le attività svolte durante il lavoro. L'applicazione deve essere progettata in modo tale da richiedere il minimo sforzo all'utente, mantenendo al contempo un'interfaccia semplice e familiare. Inoltre, deve essere in grado di salvare i dati in locale, garantendo la continuità nella raccolta dei dati anche in assenza di connessione o in caso di indisponibilità del server.

Per garantire una buona soddisfazione sia dell'operatore che dei supervisori e direttori, l'applicazione deve notificare periodicamente l'utente della necessità di fornire i dati richiesti. In questo modo, l'operatore può concentrarsi sulla propria attività lavorativa senza dover preoccuparsi di ricordarsi di inserire i dati.

Inoltre, l'applicazione deve essere progettata per garantire la massima usabilità e l'intuitività dell'interfaccia utente, in modo da ridurre al minimo la necessità di istruzioni e formazione sull'uso dell'applicazione stessa.

# 2 Approccio al problema

In questo capitolo viene presentato l'approccio adottato per risolvere il problema della registrazione della fatica e dei tasks degli operatori. Prima di iniziare lo sviluppo del progetto, sono state effettuate delle ricerche per identificare le migliori soluzioni disponibili per la registrazione delle fatiche e dei tasks degli operatori durante le diverse attività lavorative. Il processo di ricerca ha permesso di comprendere le lacune delle soluzioni esistenti e quali si addicono allo scopo del progetto.

## 2.1 Ricerca

All'inizio del progetto, l'attenzione è stata concentrata sulla ricerca di una soluzione per l'acquisizione delle fatiche e dei task svolti dagli operatori in modo il meno invasivo possibile. Nell'ambito manifatturiero, la sicurezza e il benessere dei lavoratori sono una priorità, ma allo stesso tempo, l'efficienza e la produttività dell'azienda devono essere garantite. Per questo motivo, è stata effettuata una ricerca approfondita per individuare le migliori soluzioni disponibili per la registrazione delle attività lavorative degli operatori.

Grazie all'analisi dei diversi metodi esistenti, sono state individuate diverse opzioni per l'acquisizione delle fatiche degli operatori, ognuna delle quali ha i propri vantaggi e svantaggi. Tuttavia, l'obiettivo principale è stato sempre quello di garantire un'acquisizione efficace delle fatiche degli operatori, cercando di interferire il meno possibile con le attività lavorative quotidiane.

Durante la fase di ricerca, è stata data particolare attenzione alle tecnologie più recenti e all'innovazione in questo campo. Sono state analizzate le tecniche di monitoraggio dell'attività fisica, le applicazioni mobile per la registrazione dei task e le soluzioni per la registrazione automatica della fatica.

In definitiva, la ricerca ha permesso di individuare la soluzione migliore per l'acquisizione delle fatiche degli operatori, in modo da garantire un’acquisizione veloce e di qualità, tenendo conto anche del fatto che era necessario considerare le risorse limitate a disposizione e il tempo limitato per lo sviluppo del progetto.

## 2.2 Risultati ricerca

Nella ricerca sono state individuate due possibili soluzioni per la registrazione della fatica e dei tasks degli operatori. La prima soluzione prevede l'utilizzo di smartwatch abbinati ad uno smartphone. Grazie a questo sistema, gli operatori hanno la possibilità di scegliere il dispositivo più adatto per registrare i dati in base alla situazione in cui si trovano. Inoltre, lo smartwatch permette di notificare il lavoratore attraverso una vibrazione, che risulta molto comoda in quanto lo smartwatch viene indossato costantemente. Questa opzione offre un'elevata flessibilità e consente all'operatore di utilizzare lo strumento più adatto alle sue esigenze.

La seconda opzione ipotizzata prevede lo sviluppo di un'applicazione mobile da utilizzare su dispositivi Android e iOS. Questa applicazione sarebbe progettata per essere intuitiva, facile da utilizzare e con uno UI che facilità l’utilizzo tramite guanti, al fine di garantire una registrazione rapida e accurata dei dati relativi alla fatica e ai task dell'operatore. Tuttavia, prima di procedere all'implementazione, sarebbe necessario valutare la fattibilità tecnologica e la compatibilità dei dispositivi mobili sul mercato.

## 2.3 Scelta finale

La scelta finale è stata presa in considerazione dopo aver valutato attentamente le due possibili soluzioni proposte. Dopo un'attenta analisi, la decisione è stata quella di optare per lo sviluppo di un'applicazione smartphone *cross-platform* per la registrazione della fatica e dei tasks degli operatori. La scelta è stata motivata dalla maggiore comodità e facilità di utilizzo rispetto all'utilizzo di uno smartwatch. Inoltre, l'applicazione è stata progettata per essere altamente intuitiva e facile da usare, in modo da consentire una registrazione rapida ed efficiente dei dati.

Un ulteriore fattore che ha influenzato la scelta è la compatibilità con diversi dispositivi, poiché l'applicazione è *cross-platform* e può essere utilizzata su dispositivi Android e iOS. Questa scelta è stata fatta per garantire la massima compatibilità con i vari dispositivi utilizzati dagli operatori e per evitare qualsiasi problema di compatibilità che potrebbe compromettere l'efficienza del processo di registrazione dei dati.

L'applicazione sviluppata è in grado di adattarsi alle diverse esigenze degli operatori, consentendo loro di registrare la fatica e i tasks in modo rapido e accurato

# 3 Progettazione

Nella sezione di progettazione della documentazione, vengono illustrate le decisioni prese riguardanti l'implementazione del progetto corrente e le modalità di sviluppo adottate per realizzarlo. In particolare, viene fornita una panoramica sulle scelte tecniche fatte per garantire la funzionalità dell'applicazione sviluppata.

## 3.1 Architettura scelta

Il progetto è stato organizzato in tre componenti distinti, che verranno ora analizzati.

- **Frontend**: applicazione mobile, manda richieste al backend e si aspetta in ritorno dei dati in formato *JavaScript Object Notation* (JSON).
- **Backend**: riceve le richieste provenienti dall'applicazione mobile, verificare che siano autorizzate per essere elaborate, memorizzare i dati nel database e, una volta completato il processo, inviare i dati richiesti al client frontend nel formato *JSON*.
- **Database**: memorizza le informazioni necessarie per il funzionamento dell'applicazione, come ad esempio i dati degli utenti, le informazioni sulle attività svolte e i task assegnati. In questo contesto, il database lavora in stretta collaborazione con il server, il quale si occupa di elaborare le richieste e restituire i dati necessari in modo efficiente al client frontend.

Nel prossimo paragrafo (5.3), verranno esaminate le decisioni prese per ciascuna componente di questo progetto.

## 3.2 Scelte delle tecnologie

### 3.2.1

### 3.2.2 Database

Dopo aver discusso con i committenti del progetto e valutato varie opzioni disponibili, abbiamo deciso di utilizzare MongoDB come database per lo sviluppo del progetto.

- **MongoDB**

  MongoDB è un sistema di gestione di database non relazionali, noto anche come database *NoSQL*. Si basa sul modello di dati a documento, in cui i dati vengono organizzati in documenti *JSON* che possono contenere campi e valori variabili.

### 3.2.3 Backend

In questo progetto, il server (backend) deve avere il semplice compito di fornire l'accesso ai dati tramite una REST API, permettendo all'applicazione client (frontend) di consumare i dati forniti dal backend.

Dopo aver processato e salvato i dati in entrata, il backend deve poter restituirli al client nella forma appropriata quando richiesti, in linea con i requisiti del progetto.

L'attenzione è stata posta principalmente sulle funzionalità di design del software e sulla facilità di gestione della configurazione del server.

Le principali opzioni di linguaggio considerate erano Java, già utilizzato per un sistema di gestione del materiale didattico a supporto di corsi per la formazione Bachelor, e JavaScript, visto durante il percorso universitario nel corso di applicazione web 2.

Per entrambi i linguaggi esistono diversi framework disponibili per la creazione di API REST:

**1 Javascript**

Node.js è un runtime JavaScript che consente di eseguire codice JavaScript lato server. È basato sull'engine V8 di Google Chrome e fornisce un'infrastruttura per la creazione di applicazioni web scalabili e ad alta efficienza.

Express.js, o semplicemente Express, è un framework di applicazioni web back-end per la creazione di API RESTful con Node.js, rilasciato come software gratuito e open-source sotto la licenza MIT. È progettato per la creazione di applicazioni Web e API. Il suo obiettivo principale è di semplificare la creazione di applicazioni web e API RESTful, fornendo un'ampia gamma di funzionalità e una sintassi intuitiva e facile da usare. Express.js offre una serie di middleware integrati che semplificano la gestione di richieste HTTP, sessioni, autenticazione e autorizzazione.

**2 Java**

Spring Boot è un framework per lo sviluppo di applicazioni Java, spesso utilizzato per la creazione di API RESTful. Spring Boot semplifica notevolmente il processo di sviluppo di un'applicazione, fornendo una configurazione predefinita e un insieme di funzionalità chiave, come l'iniezione di dipendenze e la gestione dei database, riducendo il tempo di sviluppo e migliorando la produttività del programmatore.

Dopo aver valutato diverse opzioni per lo sviluppo del backend dell'applicazione per la registrazione della fatica e dei tasks degli operatori, abbiamo scelto di utilizzare Express.js con Node.js. Una delle ragioni principali per questa scelta è stata la facilità di integrazione tra il backend e il frontend dell'applicazione mobile, che utilizza lo stesso linguaggio. Questo ha permesso di semplificare il processo di sviluppo e manutenzione dell'applicazione, portando a una maggiore efficienza complessiva.

### 3.2.4 Frontend

La scelta è ricaduta sull'utilizzare React Native come tecnologia per lo sviluppo del frontend di un'applicazione mobile, scelta motivata dalla sua caratteristica cross-platform. Questa tecnologia consente di scrivere su un'unica base di codice in JavaScript e utilizzarla per creare app per diverse piattaforme, come iOS e Android, senza dover scrivere codice separato per ciascuna piattaforma.

Inoltre, per semplificare ulteriormente il processo di sviluppo di applicazioni mobili con React Native, abbiamo deciso di utilizzare Expo insieme a React Native. Expo è un insieme di strumenti open source che semplificano la creazione di app mobile.

#### 3.2.4.1 **React Native**

React Native è una tecnologia per lo sviluppo di applicazioni mobili che consente di utilizzare la libreria React per creare interfacce utente native per iOS e Android. React Native consente agli sviluppatori di scrivere codice una volta e utilizzarlo per creare app per entrambe le piattaforme, anziché dover scrivere codice separato per iOS e Android. Ciò consente di ridurre i costi di sviluppo e di accelerare il time-to-market. Inoltre, React Native offre un'esperienza utente nativa, con prestazioni e funzionalità simili a quelle delle app native scritte in Objective-C o Java.

#### 3.2.4.2 Expo

Expo è una piattaforma gratuita e *open-source* che permette di sviluppare applicazioni *mobile cross-platform* utilizzando tecnologie web come *React* e *React Native*. Expo semplifica il processo di sviluppo, eliminando la necessità di configurare e gestire manualmente il processo di compilazione delle applicazioni. Inoltre, Expo fornisce molte librerie e componenti già pronti all'uso, semplificando la scrittura di codice e accelerando lo sviluppo.

Tra i vantaggi di Expo ci sono la possibilità di testare l'applicazione in anteprima su dispositivi mobili reali durante lo sviluppo, la gestione semplificata dei permessi e delle *notifiche push*.

È stato scelto di utilizzare Expo per questo progetto principalmente per la sua facilità d'uso e la sua flessibilità. In particolare, l'utilizzo di Expo ha permesso di velocizzare lo sviluppo dell'applicazione, eliminando la necessità di configurare manualmente il processo di compilazione e semplificando la gestione delle dipendenze. Inoltre, Expo ha permesso di testare l'applicazione in anteprima su dispositivi mobili reali durante lo sviluppo, permettendo di individuare e risolvere eventuali problemi con maggiore facilità.

# 4 Implementazione

In questo capitolo viene illustrato il processo di implementazione del progetto, analizzando in modo dettagliato i componenti utilizzati sia nel backend che nel frontend. Si descrivono le strutture architetturali adottate e le librerie utilizzate, evidenziando le motivazioni dietro le scelte fatte.

## 4.1 Struttura database

La struttura del database è stata fatta utilizzando la libreria di **Mongoose**. Mongoose è una libreria di Object Data Modeling (**ODM**) per MongoDB e Node.js. Mongoose semplifica la gestione delle operazioni di database, fornendo un'interfaccia facile da usare per creare, leggere, aggiornare e cancellare documenti dal database MongoDB, nonché la definizione di schemi e modelli per i dati del database.

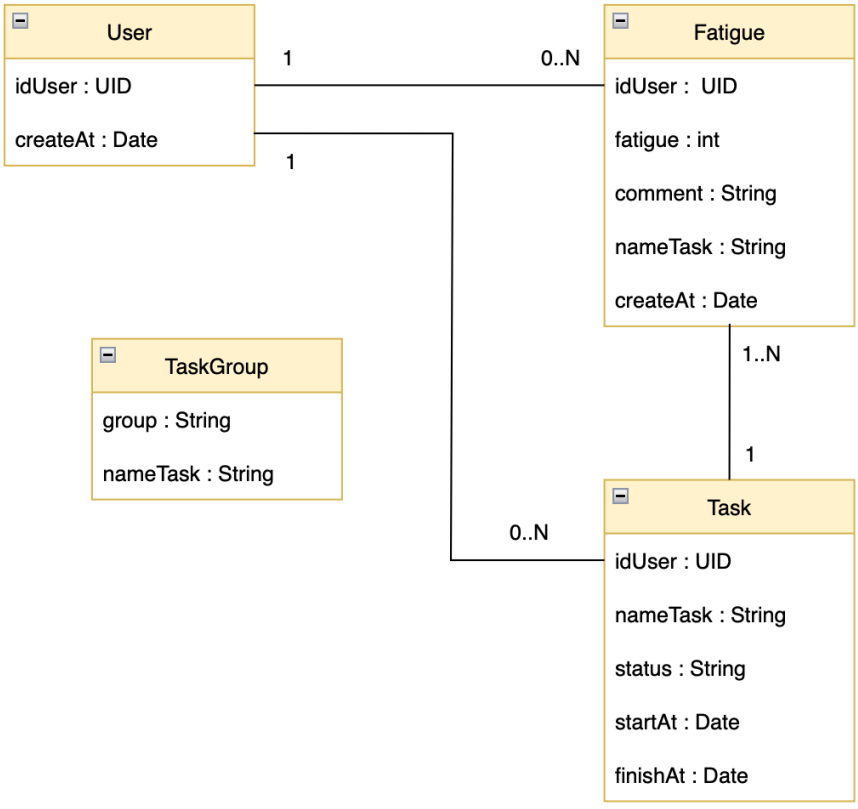



Figura 1 – Struttura database in mongoDB

Nel progetto si sono creati quattro modelli utilizzando Mongoose per semplificare la gestione delle operazioni di database con MongoDB e Node.js. Ogni modello ha una propria definizione dello schema che definisce la struttura dei dati che saranno memorizzati nella collezione, con i tipi di dati, le opzioni di validazione e di default per i campi.

Il modello "User" rappresenta un utente con un identificatore unico e una data di creazione. Il modello "Task" rappresenta un'attività con un nome, un commento, una data di inizio, uno stato e una data di fine. Il modello "TaskGroup" rappresenta un gruppo di attività con un nome. Il modello "Fatigue" rappresenta un livello di affaticamento per un utente, con un identificatore utente, un valore di affaticamento, un commento opzionale, una attività opzionale e una data di creazione.

Per creare le collections con Mongoose, è stato necessario uno schema per ogni modello da noi definito, dove lo schema definisce la struttura dei dati che saranno memorizzati nella collezione che include i tipi di dati, le opzioni di validazione e di default per i campi.

Infine, basterà richiamare la funzione model per creare la collection.

```
export const User = mongoose.model(“users”, userSchema);
```

## 4.2 Backend

Il backend del progetto è sviluppato in Node.js e utilizza il framework Express.js per la gestione delle richieste http (come spiegato nel capitolo 4.2). Il server è configurato per ascoltare sulla porta 4000 e si connette a un database MongoDB tramite il modulo Mongoose.

La struttura del progetto è organizzata in cartelle, ciascuna con uno specifico compito: la cartella "models" contiene i modelli Mongoose dei dati utilizzati nel progetto, la cartella "controllers" contiene le funzioni che gestiscono le richieste API, mentre la cartella "routes" contiene le definizioni delle route delle API.

La configurazione del server prevede l'utilizzo del modulo **Dotenv** per la gestione delle variabili di ambiente, come ad esempio la porta su cui il server deve ascoltare e la connessione al database MongoDB.

Infine, il server viene avviato mediante la chiamata al metodo "listen" di Express.js, che consente di mettere in ascolto il server sulla porta specificata e gestire le richieste HTTP in arrivo.

### 4.2.1 API backend

In questo paragrafo esploreremo il funzionamento dell'API, analizzando le diverse richieste e risposte che possono essere inviate e ricevute dal sistema. In particolare, ci concentreremo sulle funzionalità principali dell'API, tra cui l'accesso ai dati, l'autenticazione e registrazione degli utenti e la gestione degli errori.

### 4.2.2 Registrazione/Login

Per la creazione e verifica dell'utente, abbiamo utilizzato una singola *API* che gestisce entrambe le operazioni. Ciò significa che la stessa *API* viene utilizzata sia per creare un nuovo utente che per verificare se un utente esiste già.

L'endpoint utilizzato per l'invio della registrazione/login è il seguente: `api/v1/createUser`

Nel codice viene definita una *route* che accetta richieste **POST**. Quando viene inviata una richiesta POST con l'endpoint `api/v1/createUser`, viene eseguita la funzione *createUser*.

La funzione *createUser* inizia controllando se il parametro *idUser* è presente nel corpo della richiesta e se esiste già un utente con questo id nel database. Se l'utente esiste già, viene restituita una risposta con lo stato 200 e un messaggio di successo, altrimenti viene creato un nuovo utente con l'id specificato o un id generato casualmente, utilizzando il modello User.

La funzione salva il nuovo utente nel database e restituisce una risposta con lo stato 201 e l'oggetto user che rappresenta l'utente creato.

### 4.2.3 Inizio Task

Quando l'API REST riceve la richiesta all'endpoint `/api/v1/:idUser/startTask` viene richiamata funzione "startTask" dove viene estratto l'ID dell'utente dal parametro dinamico della richiesta e verifica che l'utente esista utilizzando una funzione asincrona "userExists(id)".

Se l'utente non esiste, viene restituita una risposta HTTP con stato 400 e un messaggio di errore. Invece se l'utente esiste verrà crea un nuovo oggetto Task utilizzando i dati estratti. L'oggetto Task viene impostato con lo stato "IN_PROGRESS" e viene salvato nel database utilizzando "await task.save()".

Infine, la funzione restituisce una risposta HTTP con stato 201 e il nuovo oggetto Task creato.

### 4.2.4 Fine Task

Questa è un'API che gestisce la richiesta per segnalare la conclusione di un task da parte di un utente identificato da un parametro di percorso **idUser**. L'end point è `/api/v1/:idUser/finishTask`.

Alla ricezione della richiesta, il server verifica se l'utente esiste nel database tramite la funzione **userExists()**. Se l'utente non esiste, il server restituisce una risposta di errore con un messaggio "User not exist" e lo stato HTTP 400.

Se l'utente esiste, il server cerca un task in corso associato all'utente tramite la funzione **findOneAndUpdate()** di Mongoose, e lo aggiorna con lo stato "DONE" e la data di completamento. Quindi il server restituisce una risposta di successo con lo stato HTTP 200 e il messaggio "Task update".

### 4.2.5 Aggiunta fatica

L'endpoint api/v1/:idUser/addFatigue permette di creare una nuova registrazione di fatica associata ad un utente e ad un'attività (task) in corso. La richiesta deve includere l'id dell'utente nella URL.

Inizialmente, viene controllato se l'utente esiste nel database. In caso contrario, viene restituita una risposta di errore con lo stato **400** e un messaggio "User not exist".

Se l'utente esiste, la richiesta deve includere un numero compreso da 0 a 10 di fatica e un eventuale commento. Viene quindi cercata l'attività (task) in corso dell'utente tramite una query al database che cerca un'attività in corso con lo stesso id dell'utente e lo stato "IN_PROGRESS".

Se l'attività è in corso, viene creata una nuova registrazione di fatica con l'id dell'utente, la quantità di fatica, il commento e il nome dell'attività. Viene poi salvata nel database e restituita come parte della risposta con lo stato 201 e un oggetto JSON che contiene i dettagli della registrazione di fatica creata (fatigue).

Se l'attività non è in corso, viene restituita una risposta di errore con lo stato 400 e un messaggio "Task not started".

### 4.2.6 Metodi GET

Nel backend sono stati implementati tre API che consentono di ottenere informazioni relative agli utenti, ai task e alle fatiche.

- **GET** /api/v1/getUser - restituisce la lista di tutti gli utenti registrati nel sistema.

  consente di ottenere la lista di tutti gli utenti registrati nel sistema. Quando viene effettuata una richiesta a questo endpoint, il server esegue una query al database per recuperare tutte le informazioni relative agli utenti e le restituisce sotto forma di array di oggetti JSON.

- **GET** /api/v1/users/:id/getTask - restituisce la lista di task dell'utente con l'ID specificato.

  permette di ottenere la lista di task di un utente specifico, identificato tramite il suo ID. Quando viene effettuata una richiesta a questo endpoint, il server esegue una query al database per recuperare tutte le informazioni sui task associati all'utente con l'ID specificato e le restituisce sotto forma di array di oggetti JSON.

- **GET** /api/v1/users/:id/getFatigue - restituisce la lista di fatiche registrate per l'utente con l'ID specificato.

  Questa API set permette di ottenere tutte le informazioni sulle fatiche registrate per un utente specifico. Per fare ciò, è necessario fornire l'ID dell'utente a cui le fatiche sono associate. Il server eseguirà quindi una query al database per recuperare tutte le informazioni sulle fatiche di quell'utente e le restituirà sotto forma di un elenco di oggetti JSON.

## 4.3 Frontend

La sezione front-end della documentazione del progetto è dedicata alla descrizione di come è stata progettata e implementata l'interfaccia utente dell'applicazione. Qui verranno discusse le scelte di design e di sviluppo relative alla parte visibile dell'applicazione, ovvero la parte con cui gli utenti interagiranno direttamente.

In questa sezione saranno illustrate le funzionalità principali dell'interfaccia utente, come i menu di navigazione, i pulsanti e le finestre di dialogo, oltre ai principali elementi grafici. Saranno inoltre fornite informazioni su come l'interfaccia utente è stata progettata per garantire la massima usabilità e intuitività possibile, al fine di rendere l'utilizzo dell'applicazione il più semplice e piacevole possibile per gli utenti.

### 4.3.1 Comunicazione con il backend

Nel progetto, è stata utilizzata la libreria *JavaScript Axios* per semplificare la comunicazione tra il *frontend* e il *backend*. *Axios* permette di eseguire richieste *HTTP* in modo asincrono dal *browser* o dal *server*, fornendo un'*API* semplice e intuitiva per creare richieste e gestire le risposte.

Sono state definite diverse funzioni all'interno del file "**requestManager.js**", che utilizzano *Axios* per effettuare richieste specifiche al *backend*. Ad esempio, la funzione *fetchUsers* utilizza *Axios* per fare una richiesta *GET* al backend e ottenere l'elenco degli utenti, mentre la funzione *createUser* fa una richiesta *POST* per creare un nuovo utente.

```
//post create user, and return idUser of the new user
export const createUser = async (id) => {
    try {
        let response = await axios.post(
            `${API_BASE_URL}createUser`,
            {
                idUser: id
            }
        );

        if (response.status !== 201 && response.status !== 200) {
            throw new Error("Failed to fetch users");
        }

        return response.data.user.idUser;
    } catch (err) {
        if(axios.isCancel(err)){
            console.log('Data fetching cancelled');
        }else{
            console.log("Error while creating user");
            console.log(err);
        }
    }
}
```

### 4.3.2 Connessione Internet

In generale, nell'applicazione è stato creato uno stato booleano per salvare se l'applicazione è connessa o meno a internet. Questo stato viene aggiornato utilizzando la libreria '**@react-native-community/netinfo**' e il metodo *NetInfo.addEventListener*, il quale permette di leggere lo stato della connessione dell'applicazione.

```
import NetInfo from '@react-native-community/netinfo';

const [isConnected, setIsConnected] = useState(false);

useEffect(() => {
    // Get if the user is connected to the internet
    const unsubscribe = NetInfo.addEventListener(state => {
        //console.log("Connection type", state.type);
        //console.log("Is connected?", state.isConnected);
        setIsConnected(state.isConnected);
    });

    return () => unsubscribe();
  }, []);
```

In questo modo, ogni volta che lo stato della connessione cambia, l'applicazione aggiorna il valore dello stato booleano. Questo stato viene poi passato attraverso un *context*, che rende il valore disponibile in tutta l'applicazione. In questo modo, l'applicazione può controllare se c'è una connessione internet disponibile per eseguire le operazioni necessarie.

```
return (
    <UserContext.Provider value={{userId, setUserId}}>
      <ConnectionContext.Provider value={{isConnected, setIsConnected}}>
        <HomeNavigation>
        </HomeNavigation>
      </ConnectionContext.Provider>
    </UserContext.Provider>
  );
```

### 4.3.3 Schermata di login

La schermata di login all'interno dell'applicazione è stata progettata per permettere all'utente di accedere ai propri dati personali. Inizialmente, l'utente deve identificarsi per poter accedere ai propri dati. Una volta eseguito l'accesso, non sarà più necessario ripetere questa procedura.

Nella schermata di login è presente un pulsante "**registrati**" per gli utenti che non hanno ancora un id associato, mentre gli utenti già registrati possono accedere utilizzando il tasto "**accedi**". Inoltre, c'è un menu a tendina che permette all'utente di scegliere tra gli ID disponibili o di inserirne uno manualmente. In caso di eventuali errori durante l'inserimento dell'ID, verrà visualizzato un messaggio di errore che fornisce all'utente informazioni sul problema riscontrato.

Gli utenti che hanno già effettuato l'accesso possono eseguire il logout semplicemente con uno swipe indietro o utilizzando il pulsante "**logout**" presente nell'angolo in alto a sinistra della schermata. In questo modo, gli utenti possono gestire il proprio accesso e uscire dall'applicazione in modo semplice e veloce.

In alto la schermata è presente un testo per debug, che non dovrebbe esserci nella versione di produzione dell'applicazione.


Figura 3 - Schermata Login


Figura 2 - Schermata Login (errore)

### 4.3.4 Sessione di login

Questa sezione di codice descrive il processo di login dell'utente nell'applicazione. La funzione **handleLogin** viene richiamata quando l'utente preme il bottone "Accedi" e inserisce un id corretto. In questa funzione, l'id dell'utente viene aggiornato nello stato globale dell'applicazione, che viene mantenuto all'interno di un contesto. Se l'utente non è ancora registrato, viene creato un nuovo utente nel database con l'id inserito dall'utente.

```
const handleLogin = () => {

    // update logged user
    setUserId(value);

    createUser(value)
      .then(navigation.navigate('Home', {userId: value}))
      .catch(err => {throw err});
  }
```

Per mantenere permanentemente l'id dell'utente loggato, anche dopo la chiusura dell'applicazione, viene utilizzato **AsyncStorage**. La funzione **setItem** viene utilizzata per salvare una stringa in locale sul dispositivo. Questo salvataggio viene effettuato automaticamente dopo che un nuovo valore di id utente è stato salvato.

```
  // when logged user is changed, save it in async storage
  // ang go to home screen
  useEffect(() => {
    if (userId) {
      AsyncStorage.setItem(LOGGED_USER_KEY, userId);
      navigation.navigate('Home');
    }
  }, [userId]);
```

Inoltre, se un id utente è già stato salvato in locale, l'utente verrà loggato automaticamente e portato alla pagina Home. Per fare ciò, viene utilizzata la funzione getItem di AsyncStorage nell'effetto collaterale useEffect che viene eseguito quando il componente viene montato. Se l'id è stato salvato, viene impostato nello stato globale dell'applicazione. Se l'id non è stato salvato in precedenza, l'utente dovrà effettuare il login manualmente.

```
// on mount, get users from local storage
 useEffect(() => {
   // get logged user from async storage
   AsyncStorage.getItem(LOGGED_USER_KEY)
     .then(res => {
       if (res !== null) {
         setUserId(res);
       }
     })
 }, []);
```

### 4.3.5 Salvataggio Locale

Il salvataggio in locale avviene attraverso la libreria **AsyncStorage** di React Native, che permette di memorizzare dati in modo asincrono e persistente. In particolare, vengono definite diverse funzioni all'interno del file **localStorage.js** per salvare i dati in locale, ad esempio: *saveStartTask*, *saveAddFatigue*, *saveEndTask* e *saveSurveyData*.

La funzione *saveStartTask* viene chiamata quando un'attività viene avviata. Prende in input il tipo di attività corrente, *currentActivity*, e un booleano *sentToServer* che indica se i dati relativi all'attività sono già stati inviati al server. Il metodo *getItem* di *AsyncStorage* viene utilizzato per recuperare i dati salvati in precedenza in locale. Se non ci sono dati, viene inizializzato un array vuoto. Il metodo JSON.parse viene utilizzato per convertire i dati in formato JSON in un oggetto JavaScript. Successivamente, l'ID dell'utente viene recuperato con il metodo getItem di AsyncStorage. Viene quindi costruito un oggetto che rappresenta l'istanza dell'attività corrente e viene aggiunto all'array dei dati. Infine, viene chiamato il metodo setItem di AsyncStorage per salvare i dati aggiornati in locale.

```
// save in localstorage that a task is started
export const saveStartTask = async (currentActivity, sentToServer) => {
    try {
        const data = await AsyncStorage.getItem(OFFLINE_DATA_KEY);
        const userId = await AsyncStorage.getItem(LOGGED_USER_KEY);
        let dataParsed = JSON.parse(data);

        if (!dataParsed)
            dataParsed = [];

        dataParsed.push(OFFLINE_DATA_START_TASK_ISTANCE(
            userId,
            currentActivity,
            Date.now(),
            sentToServer
        ));

        console.log("Save start task");
        console.log(dataParsed);
        await AsyncStorage.setItem(OFFLINE_DATA_KEY, JSON.stringify(dataParsed));
    } catch (error) {
        console.log("error while saving start task in local storage");
        console.log(error);
    }
}
```

### 4.3.6 Schermata Home

Nella schermata home, una volta eseguito il login, l'utente si trova di fronte a due tasti che consentono diverse azioni a seconda della scelta effettuata. Il primo tasto presente è denominato "**Quanto sei stanco?**" e, premendolo, viene avviato il sondaggio sulla fatica percepita, che porta l'utente alla schermata della fatica.

Il secondo tasto, invece, è denominato "**Inizia Attività**" e permette di avviare una nuova task. Una volta premuto, l'utente visualizza una schermata con tutte le attività di routine, tra cui può scegliere per iniziare una nuova task.

Una volta che l'utente ha avviato un task, il tasto "**Inizia Attività**" diventerà "Termina Task" sulla schermata home. In questo modo, l'utente potrà terminare la task corrente semplicemente premendo il tasto "**Termina Task**" e il task verrà interrotta, tornando alla schermata home.


Figura 4 - Schermata Home

### 4.3.7 Schermata Fatica

La schermata della fatica percepita ha lo scopo di permettere all'utente di esprimere la propria sensazione di stanchezza dopo aver svolto una determinata attività di routine. In particolare, l'utente può scegliere un valore compreso tra 0 e 10 per indicare la propria fatica percepita, dove 0 indica nessuna fatica e 10 indica una fatica estrema. Questo valore verrà poi associato all'attività svolta dall'utente così da permettere una valutazione complessiva del livello di sforzo e fatica nel corso del tempo.

#### 4.3.7.1 Lista responsive di bottoni

Inizialmente, avevamo pensato di creare una lista di numeri da **0** a **10**, grande quanto possibile per adattarsi allo schermo del cellulare. Quindi abbiamo deciso di adottare un approccio flessibile, creando una scala di valutazione della fatica che si adatta alle dimensioni dello schermo del dispositivo. In questo modo, l'utente potrà selezionare il proprio livello di fatica in modo semplice e intuitivo, indipendentemente dalle dimensioni del dispositivo utilizzato.

Tuttavia, abbiamo poi capito che questa soluzione potrebbe non essere ideale per i telefoni cellulari con schermi più piccoli. Infatti, la grafica risulterebbe compressa in una lista poco leggibile, con aree troppo ristrette.

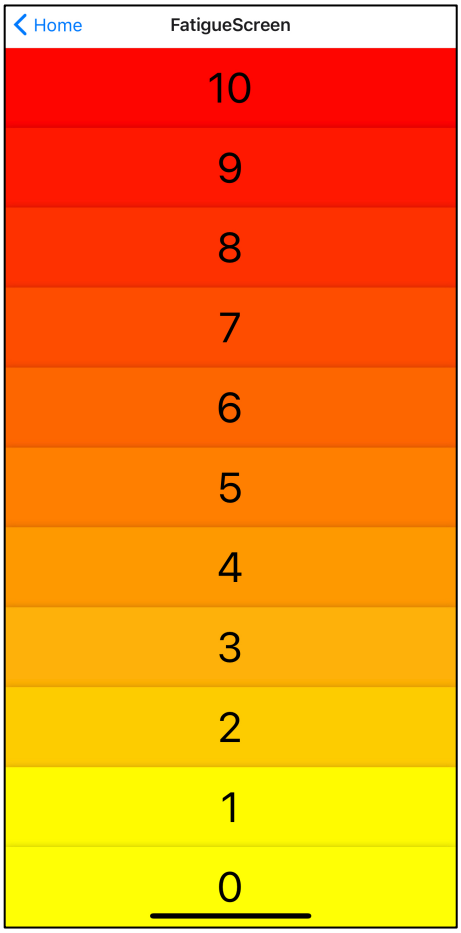



Figura 5 - Lista Responsive di Bottoni

#### 4.3.7.2 Lista responsive di bottoni su due colonne

Abbiamo deciso di adottare una seconda soluzione per la selezione del livello di fatica degli utenti. invece di creare solo una lista di opzioni, abbiamo scelto di creare una lista di bottoni responsive su due colonne. A differenza della lista responsive di bottoni, questo approccio consente di visualizzare le opzioni su bottoni cliccabili di dimensioni maggiori rispetto alla lista di numeri, poiché i bottoni sono disposti su due colonne.

Questa progettazione migliora l'usabilità sui dispositivi con schermi di dimensioni ridotte; tuttavia, la disposizione dei tasti potrebbe non essere ottimale. Per verificare tale aspetto sarebbe necessario condurre un test su larga scala coinvolgendo un numero significativo di utenti.


Figura 6 - Lista Responsive Doppia di Bottoni

#### 4.3.7.3 Miglioramento stile dei bottoni

Abbiamo deciso di apportare ulteriori miglioramenti alla nostra interfaccia utente, prendendo in considerazione sia il nostro gusto personale che la logica. Per questo motivo, abbiamo deciso di implementare un nuovo design per i pulsanti numerici presenti nella nostra applicazione. In particolare, abbiamo deciso di incrementare i numeri dal basso verso l'alto come su un tastierino numerico, al fine di rendere l'esperienza più intuitiva per gli utenti.

Inoltre, abbiamo aggiunto dei bordi ai rettangoli per farli sembrare più come dei veri e propri bottoni. Questo ha contribuito ad aumentare la chiarezza dell'interfaccia, poiché gli utenti possono riconoscere facilmente quali elementi sono cliccabili e quali no.

In conclusione, crediamo che questi miglioramenti abbiano reso l'interfaccia utente più intuitiva e facile da usare, e siamo soddisfatti dei risultati ottenuti.

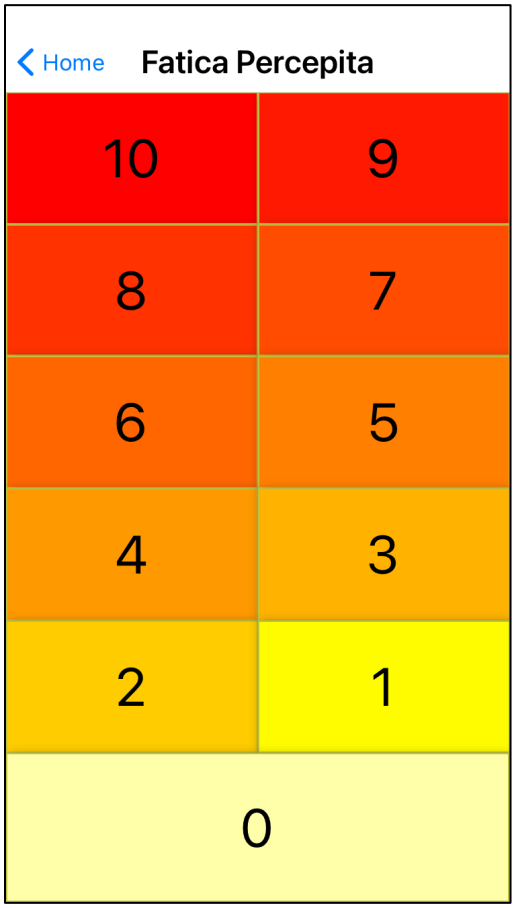



Figura 7 - Lista Responsive Doppia di Bottoni (Migliorata)

#### 4.3.7.4 Implementazione Generale Lista bottoni

La lista di bottoni è stata implementata utilizzando la libreria *React Native* e il componente *TouchableOpacity* per i bottoni. I valori di fatica sono stati memorizzati in un *array* di oggetti chiamato **fatigueValues**, in cui ogni oggetto rappresenta un valore di fatica e contiene una chiave (*key*) e un colore associati a quel valore.

```
const fatigueValues = [
  {key: '10', color: '#FF0000'},
  {key: '8', color: '#FF3200'},
  {key: '6', color: '#FF6600'},
  {key: '4', color: '#FF9900'},
  {key: '2', color: '#FFCC00'},
  {key: '9', color: '#FF1900'},
  {key: '7', color: '#FF4C00'},
  {key: '5', color: '#FF7F00'},
  {key: '3', color: '#FFB200'},
  {key: '1', color: '#FFFC00'},
  {key: '0', color: '#FFFFAA'},
];
```

La funzione FatigueListItem è stata creata per renderizzare ogni singolo pulsante di fatica, utilizzando la proprietà "item" (oggetto del valore di fatica) e la funzione "onFatiguePress" come proprietà del tocco per gestire il clic.

```
function FatigueListItem({ item, onFatiguePress}){

  return (
    <TouchableOpacity
      onPress={() => onFatiguePress(item)}
      style={[styles.item , {backgroundColor: item.color}]}
    >
      <Text style={styles.itemText}>{item.key}</Text>
    </TouchableOpacity>
  );
}
```

La funzione **FatigueDoubleListScreen** è stata creata per renderizzare due colonne di pulsanti di fatica. È stata utilizzata la funzione *map* per iterare su ogni valore di fatica e suddividerli in due colonne, una a sinistra e una a destra. Questa funzione passa la proprietà "item" alla funzione **FatigueListItem** per ogni valore di fatica nella mappatura.

Infine, quando l'utente preme su un pulsante di fatica, viene aperto un modale di conferma che mostra il valore di fatica selezionato e un campo di testo opzionale per i commenti. La funzione *handleModalConfirm* viene chiamata quando l'utente conferma la scelta di fatica e il modale viene chiuso. La funzione *handleModalCancel* viene chiamata quando l'utente annulla la scelta di fatica e il campo di testo opzionale viene ripulito.

### 4.3.8 Schermata Attività

Dopo aver inserito il livello di fatica percepita, l'utente verrà indirizzato alla schermata delle attività di routine, dove potrà scegliere quale attività sta svolgendo per associarla al valore di fatica inserito in precedenza.

Nella schermata attività, l'attività corrente viene visualizzata nel riquadro più grande, in modo da facilitare la scelta dell'utente. È anche possibile aggiungere una nuova attività che non era stata prevista, schiacciando sul riquadro rosso.

Dopo aver selezionato un'attività, all'utente verrà richiesto di confermare l'attività corrente e una volta verificata, verrà riportato alla schermata home.


Figura 9 - Schermata Attività


Figura 8 - Schermata Attività (Attività Improvvisa)

### 4.3.9 Notifiche

Per inviare le notifiche con Expo, è necessario utilizzare il modulo **Expo Notifications**. Questo modulo consente di inviare notifiche push ai dispositivi mobili utilizzando i servizi di notifica delle piattaforme Android e iOS.

Per utilizzare Expo Notifications, è necessario prima richiedere i permessi per le notifiche all'utente. Questo viene fatto utilizzando il metodo **Notifications.requestPermissionsAsync()**. Una volta richiamato all'interno di useEffect, se i permessi per le notifiche non sono abilitati, verrà visualizzato un pop-up per richiederli all'utente.

Per inviare effettivamente una notifica, è necessario utilizzare il metodo **Notifications.scheduleNotificationAsync()** o **Notifications.presentNotificationAsync()**, che richiedono un oggetto di configurazione per la notifica. Questo oggetto di configurazione include il titolo e il corpo della notifica, nonché eventuali dati aggiuntivi da inviare con la notifica.

```
useEffect(() => {
    // Request permission to send push notification
    Notifications.requestPermissionsAsync();

    // Get if the user is connected to the internet
    const unsubscribe = NetInfo.addEventListener(state => {
        //console.log("Connection type", state.type);
        //console.log("Is connected?", state.isConnected);
        setIsConnected(state.isConnected);
    });

    return () => unsubscribe();
  }, []);
```

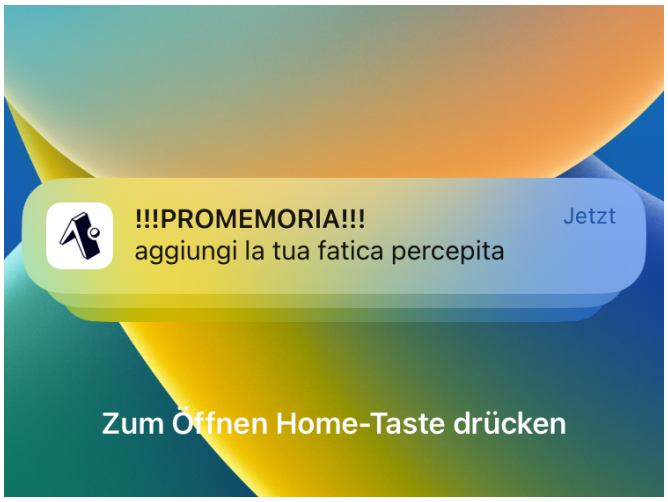



Figura 10 - Notifica Sondaggio Fatica

# 5 Deployment

In questo capitolo vedremo come effettuare il deploy sia per il frontend che per il backend, che è un processo fondamentale per rendere disponibile l'applicazione agli utenti finali.

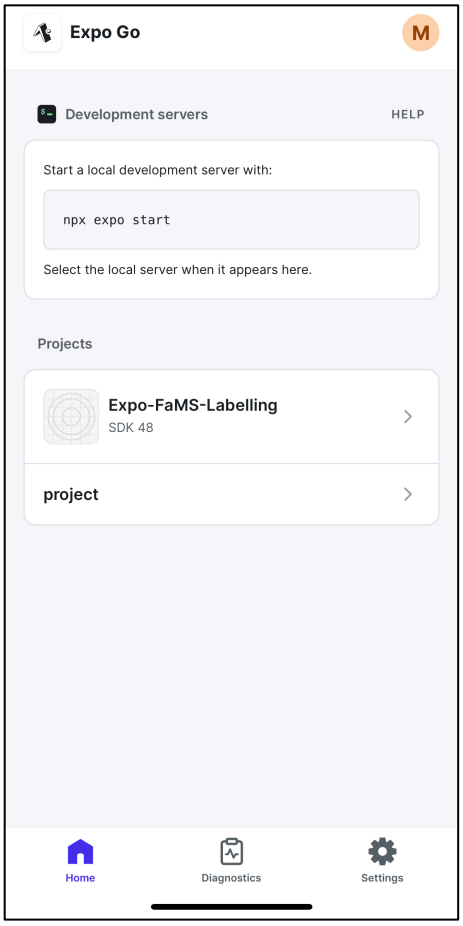
## 5.1 Frontend

Il deploy dell'applicazione in React Native verrà effettuata tramite Expo. In primo luogo, è necessario creare un account sul sito di Expo (www.expo.dev) per poter accedere ai servizi di pubblicazione dell'applicazione. Una volta registrato, basterà posizionarsi tramite terminale sulla cartella **Expo-FaMS-Labelling** del progetto.

Poi si dovranno eseguire alcuni comandi da terminale:

- "expo login" una volta eseguito il comando bisognerà autenticarsi con le credenziali dell'account Expo creato in precedenza.
- “expo publish” eseguito questo comando l'applicazione verrà caricata sul cloud di Expo e verrà resa disponibile per il download tramite l'applicazione ExpoGo.

Infine, basterà scaricare l'applicazione ExpoGo dallo store e accedere con le proprie credenziali Expo. Una volta autenticati, apparirà la scherma di home di ExpoGo (vedi Figura 10)




Applicazione per l'acquisizione dati di un operatore manifatturiero

Figura 11 - Home ExpoGo

Se i passi sono stati eseguiti correttamente l'applicazione sarà disponibile sotto la sezione Project con il nome Expo-FaMS-Labelling e basterà cliccarla per avviare l'applicazione. Per il corretto funzionamento dell'applicazione il Server(backend) dovrà essere in esecuzione.

## 5.2 Backend

Per la parte di backend/server, abbiamo deciso di utilizzare Docker per la distribuzione e il deploy del nostro software.

Docker ci consente di impacchettare il nostro software, insieme alle sue dipendenze, in un contenitore virtuale autonomo che può essere eseguito su qualsiasi macchina. In questo modo, il nostro software è isolato dal sistema host e le dipendenze sono gestite in modo centralizzato all'interno del contenitore, garantendo così un ambiente di esecuzione coerente e riproducibile.

Per configurare il contenitore, si è utilizzato due file fondamentali: Dockerfile e il docker-compose.yml.

Nel progetto il Dockerfile definisce la costruzione di un'immagine Docker che contiene l'applicazione Node.js e tutte le sue dipendenze, pronta per essere eseguita in un contenitore Docker. Mentre il docker-compose.yml determina due servizi, uno per MongoDB e uno per il server Node.js. Il servizio MongoDB utilizza un'immagine predefinita di MongoDB e definisce un volume per la persistenza dei dati. Il servizio Node.js è costruito dal Dockerfile nella directory corrente, esponendo la porta 4000 e dipende dal servizio MongoDB. Le variabili d'ambiente sono impostate per consentire alla app Node.js di connettersi al servizio MongoDB.

Per eseguire il deploy del nostro backend (server) basterà assicurarsi che Docker sia installato e in esecuzione sul proprio sistema.

Infine, molto semplicemente ci si dovrà spostare tramite terminale nella cartella in qui viene contenuto il Dockerfile ed eseguire il comando ***docker compose up***

# 6 Conclusioni

In quest'ultimo capitolo viene effettuata un'analisi della situazione attuale, valutando i risultati ottenuti a seguito dell'implementazione. In particolare, viene valutato se gli obiettivi prefissati sono stati raggiunti e se lo scopo generale del progetto è stato soddisfatto. Vengono inoltre espressi i commenti personali riguardo ai limiti dell'applicazione e alle possibili aree di sviluppo futuro. Infine, il capitolo si conclude con le conclusioni finali, riassumendo i punti chiave del progetto e le considerazioni generali sulle lezioni apprese e le eventuali implicazioni per il futuro.

## 6.1 Risultati Ottenuti

Durante lo sviluppo dell'applicazione, sono stati raggiunti diversi obiettivi. Innanzitutto, è stata creata un'interfaccia utente intuitiva ed esteticamente gradevole, che permette di inserire informazioni riguardanti l'inizio e la fine delle attività in modo semplice e veloce. Grazie all'utilizzo della libreria React, è stato possibile realizzare un'interfaccia reattiva e performante, che garantisce un'esperienza utente fluida e senza interruzioni.

Inoltre, sono state implementate le funzionalità di salvataggio in locale e su server, permettendo all'utente di riprendere in qualsiasi momento, anche in assenza di connessione, quello che stava facendo nell'applicazione. È stata inoltre effettuata una accurata gestione degli errori, con messaggi di errore chiari e comprensibili per l'utente.

In generale, il progetto ha raggiunto gli obiettivi prefissati, fornendo un'applicazione funzionale e intuitiva per la raccolta di informazioni sulla fatica e sulle attività. In particolare, l'interfaccia dell'applicazione, basata su **React**, si è dimostrata molto minimale ma efficace nel garantire un'esperienza utente fluida e semplice.

## 6.2 Considerazioni Personali

Ci riteniamo soddisfatti di aver sviluppato un'applicazione mobile, che ha soddisfatto i requisiti iniziali del progetto. Abbiamo creato un'applicazione che consente agli operatori di registrare le loro attività in modo semplice e intuitivo, migliorando l'efficienza e l'accuratezza dei dati raccolti.

Inoltre, il nostro team ha affrontato una sfida interessante poiché non avevamo mai utilizzato le tecnologie node.js e react native in precedenza. Grazie alla collaborazione e alla determinazione, siamo riusciti a imparare rapidamente e ad applicare queste tecnologie per creare l'applicazione.

Infine, questo progetto ci ha permesso di acquisire nuove competenze e di superare con successo le sfide tecnologiche incontrate lungo il percorso.

## 6.3 Possibili sviluppi futuri

In quanto progetto limitato dal tempo, ci sono alcune potenziali funzionalità che non abbiamo avuto la possibilità di implementare, ma che potrebbero apportare notevoli miglioramenti all'applicazione in termini di efficienza del lavoro degli operatori e qualità del servizio offerto. In questa sezione, daremo un'occhiata ad alcune di queste possibilità di sviluppo futuro.

- l'implementazione di un sintetizzatore vocale per la scrittura dei task e delle fatiche da parte degli operatori. Questo sarebbe particolarmente utile per coloro che hanno difficoltà nella digitazione sui dispositivi mobili a causa di accessori indossati (es. guanti).
- l'integrazione dell'applicazione con uno smartwatch. Ciò consentirebbe di registrare le fatiche e i task direttamente dallo smartwatch, fornendo un modo ancora più efficiente per i dipendenti di registrare il loro lavoro. L'integrazione con uno smartwatch potrebbe anche migliorare la sicurezza dei dipendenti, in quanto non sarebbero costretti a estrarre continuamente il telefono durante le attività lavorative. Le notifiche sarebbero più dirette visto che lo indosserebbero costantemente al polso, notificandoli immediatamente con la vibrazione.

## Conclusioni finali

In conclusione, l'applicazione sviluppata ha raggiunto gli obiettivi prefissati, fornendo agli operatori un modo semplice ed efficace per registrare le loro attività e la fatica. Grazie all'utilizzo delle tecnologie **React** e **Node.js**, l'interfaccia dell'applicazione si è dimostrata reattiva e performante, garantendo un'esperienza utente fluida e senza interruzioni. Inoltre, la gestione degli errori è stata accurata, con messaggi di errore chiari e comprensibili per l'utente. Il team di sviluppo ha superato con successo le sfide tecnologiche incontrate lungo il percorso, acquisendo nuove competenze lungo il processo. Ci sono alcune potenziali funzionalità che potrebbero essere implementate in futuro, come l'integrazione dell'applicazione con uno smartwatch e l'implementazione di un sintetizzatore vocale per la scrittura dei task e delle fatiche. Tali funzionalità potrebbero migliorare ulteriormente l'efficienza del lavoro degli operatori e la qualità del servizio offerto. In generale, il progetto ha fornito un'esperienza di apprendimento gratificante e ha permesso al team di sviluppo di applicare le proprie conoscenze e competenze per creare un'applicazione mobile funzionale e intuitiva.

# Bibliografia

https://springdoc.org (visitato il 04/05/2023)

https://it.wikipedia.org/wiki/Node.js (visitato il 04/05/2023)

https://mongoosejs.com (visitato il 04/05/2023)

https://reactnative.dev (visitato il 04/05/2023)

https://expo.dev (visitato il 04/05/2023)

https://expressjs.com (visitato il 04/05/2023)

https://www.mongodb.com (visitato il 04/05/2023)

# Glossario

**UUID**: sono stringhe di caratteri alfanumerici da 128-bit che vengono utilizzati per identificare univocamente un'entità o un oggetto in modo globale. In questo progetto sono stati utilizzati per identificare gli utenti in maniera univoca.

**HDT**: (Human Digital Twin) è un sistema di intelligenza artificiale e analisi dati che mira a creare una copia digitale di un operatore manifatturiero e delle sue attività lavorative.