ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria e Architettura
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Management by Objectives

Elaborato in
Applicazioni e servizi web

Autori
Davide Alpi
Angelo Parrinello
Paolo Penazzi

Anno Accademico 2022-2023

Introduzione

Il progetto consiste nella creazione di un software di Management By Objectives (MBO).

In sostanza, si tratta di un software che permette di definire obiettivi per i dipendenti di un'azienda, e di monitorarne il raggiungimento.

In base al raggiungimento degli obiettivi, ai dipendenti spetta un premio in denaro.

Requisiti

L'azienda utilizza attualmente un semplice foglio Excel, con una vista alquanto confusionaria, e senza la possibilità per ciascun dipendente di vedere solo i propri dati.

Il software deve permettere di definire obiettivi per l'anno corrente per i dipendenti di un'azienda, e di monitorarne il raggiungimento.

Un obiettivo è definito in questo modo:

- Nome, univoco e significativo
- Tipologia, gli obiettivi possono essere "Aziendale" (comune a tutti, come il fatturato), "Funzione" (relativo allo specifico ruolo del dipendente), o "Complementare" (un obiettivo raggiungibile dal ruolo del dipendente in collaborazione con altri dipartimenti aziendali), o "Soft Skills" (obiettivi di crescita personale).
- Base, il valore di partenza dell'obiettivo.
- 100%, il valore che l'obiettivo deve raggiungere entro fine anno per ottenere il minimo dei punti relativi a quest'obiettivo.
- 150%, il valore che l'obiettivo deve raggiungere entro fine anno per ottenere il massimo dei punti relativi a quest'obiettivo.
- Peso, il peso dell'obiettivo rispetto agli altri obiettivi. Gli gli obiettivi assegnati ad un dipendente deve sommare a 100.
- Uptodate, il valore attuale dell'obiettivo (i dati possono venire aggiornati man mano durante l'anno).

I dipendenti sono modellati in maniera molto semplice, con username, password, nome, cognome e ruolo aziendale. L'applicativo deve permettere al manager (super-user) di:

- Creare l'anagrafica dei dipendenti.
- Caricare gli obiettivi per l'anno corrente, assegnati a ciascun dipendente.
- Caricare i valori up-to-date, sulla base dei quali verrà calcolato il punteggio.

L'applicativo deve permettere al dipendente di:

- Vedere i propri obiettivi, e il loro stato di avanzamento.
- Simulare come cambierebbe il punteggio se l'obiettivo raggiungesse un certo valore.

Entrambe le tipologie di utenti chiaramente devono poter loggarsi nell'applicativo. Per il momento il software non deve permettere la registrazioni di nuovi utenti, che verranno configurati manualmente.

Design

Mockup

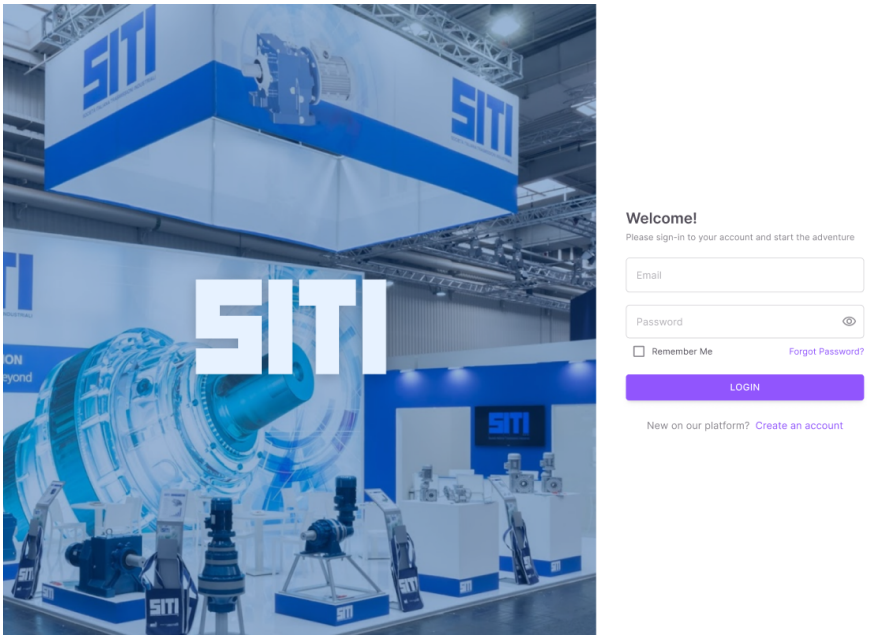

Figure 1: Schermata di login

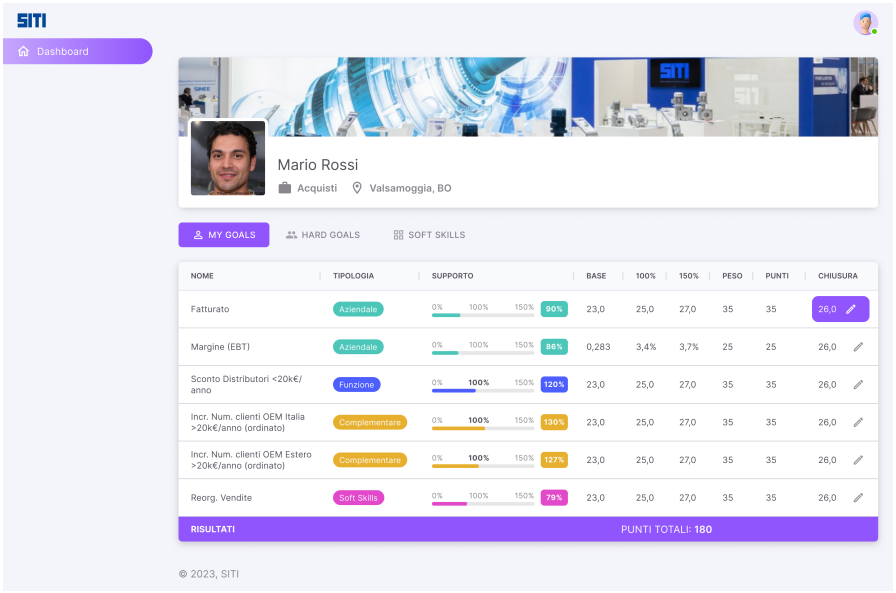

Figure 2: Dipendente - dashboard dei propri obiettivi

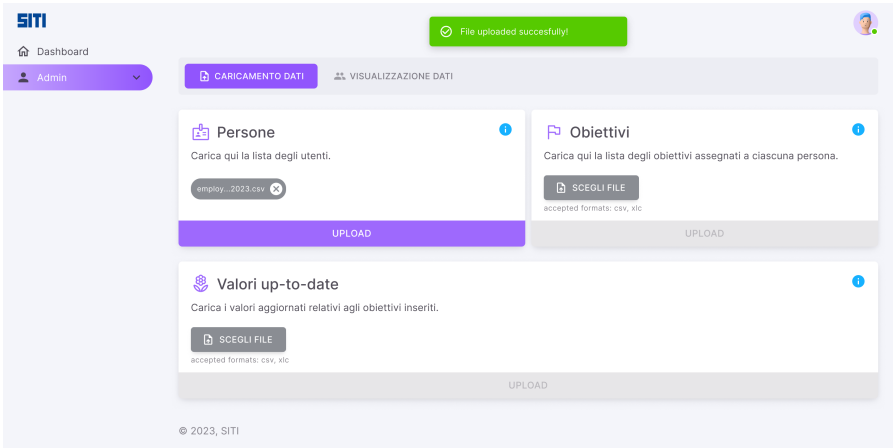

Figure 3: Admin - upload dei dati in formato csv

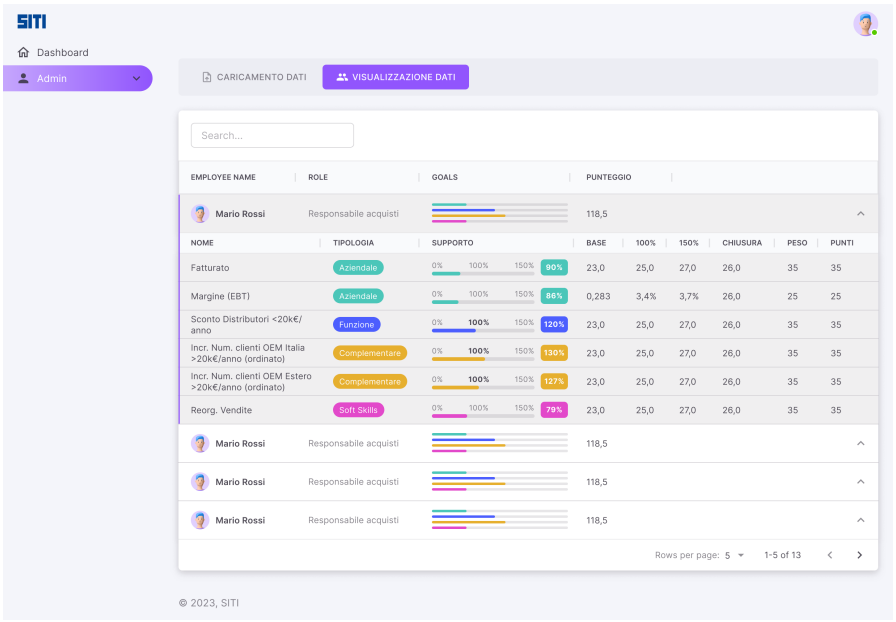

Figure 4: Admin - visione d'insieme su tutti i dipendenti

Il valore di chiusura è il valore raggiunto per l'obiettivo a fine anno. Di default è pari al valore base per l'anno corrente, finchè i dati up-to-date non vengono inseriti. Nei mockup non era stata prevista la colonna up-to-date. Il design finale dell'app consiste in:

- l'admin vede i valori up-to-date di ciascun dipendente
- i dipendenti nella propria dashboard, oltre alla colonna up-to-date, hanno a disposizione anche la colonna "chiusura", che è modificabile e consente di simulare valori possibili di chiusura e vedere come cambierebbe il punteggio.

Architettura

0.0.1 Database schemas

Il design del database è stato effettuato sulla base delle richieste del cliente. L'azienda ci ha fornito un file Excel contenente tutti i dati che dovevano essere inseriti nel database.

Sulla base di questo file, sono stati creati diversi schemi, ognuno dei quali rappresenta un entità del dominio.

- **User**: contiene i dati degli utenti, ovvero i dipendenti dell'azienda.
- **Credential**: contiene le credenziali degli utenti.

- **Objective**: contiene tutti gli obiettivi che l'azienda si pone.
- **Assigned**: contiene i valori dell'obiettivo, per ogni utente.
- **Default**: contiene i valori di default di ogni obiettivo.
- **Data**: contiene il valore corrente di un utente per un obiettivo.

Di seguito vengono mostrati gli schemi del database creati con Mongoose.

Users

```
const userSchema = new Schema({
    username: { type: String, required: true, unique: true},
    name: { type: String, required: true, },
    surname: { type: String, required: true },
    position: { type: String, required: true },
    manager: { type: String },
});
```

Da notare che l'unico campo non required è il manager. Abbiamo scelto di non renderlo obbligatorio in quanto ci potrebbero essere delle inconsistenze, o anche solo delle complicazioni, a gestire alcune situazioni. Inoltre, dopo aver parlato con gli stakeholders, è emerso che la gestione del suddetto campo non è fondamentale ai fini del progetto.

La chiave primaria di questo schema è il campo *username*, che è unico.

Credentials

```
const credentialsSchema = new Schema({
    username: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    role: { type: String, required: true }
});
```

Objectives

```
const objectivesSchema = new Schema({
    name: { type: String, required: true, unique: true },
    type: { type: String, required: true },
})
```

Questo schema è molto semplice: per ogni obiettivo si salvano nome e tipo. Il nome dell'obiettivo è la chiave primaria, ed è unico.

Assigned Objectives

```
const assignedObjectives = new Schema({
    user: { type: String, required: true },
    objective: { type: String, required: true },
    baseValue: { type: Number, required: true },
    value100: { type: Number, required: true },
    value150: { type: Number, required: true },
    weight: { type: Number, required: true },
    year: { type: Number, required: true}
})
```

Questo schema contiene i valori per ogni obiettivo di ogni utente. La chiave primaria è composta da *user* e *objective*, in quanto un utente per un determinato obiettivo può avere solo un valore. La semantica dei vari campi (value100, value150, etc..) deriva dai dati forniti dall'azienda.

Default values

```
const defaultsObjectivesSchema = new Schema({
    objective: { type: String, required: true, unique: true },
    defaultBaseValue: { type: Number, required: true },
    default100: { type: Number, required: true },
    default150: { type: Number, required: true },
    defaultWeight: { type: Number, required: true },
})
```

In questo schema, sono salvati i valori di default per ogni obiettivo. La chiave primaria è il nome dell'obiettivo. I valori di default sono stati forniti dall'azienda e sono utilizzati in caso non vengano assegnati dei valori specifici all'utente.

Data

```
const dataSchema = new Schema({
    objective: { type: String, required: true},
    username: { type: String, required: true },
    date: { type: Date, required: true},
    value: { type: Number, required: true },
})
```

Nel Data schema sono contenuti i valori reali di un utente per un obiettivo in una determinata data.

0.0.2 API

Le API utilizzano Next.js API routes. Questo framework permette di creare delle API RESTful in modo molto semplice.

Di seguito sono mostrate le routes create:

Endpoint	**Descrizione**
users/put	Aggiunge uno o più utenti al database (gli utenti già esistenti vengono aggiornati).
users/get	Recupera un utente dal database (nel caso non venga specificato uno username, ritorna tutti gli utenti).
users/delete	Rimuove un utente dal database e come manager di altri utenti, insieme agli obiettivi a lui assegnati.
users/update/manager	Aggiorna il manager di un utente.
users/update/position	Aggiorna la posizione di un utente.
objectives/put	Aggiunge uno o più obiettivi al database (gli obiettivi già esistenti vengono aggiornati).
objectives/get	Recupera un obiettivo dal database (nel caso non venga specificato un nome obiettivo, li ritorna tutti).
objectives/delete	Rimuove un obiettivo dal database.
assigned/put	Assegna un obiettivo ad un utente, o più obiettivi a più utenti.
assigned/get	Dato un utente, recupera tutti gli obiettivi a lui assegnati.
assigned/deleteAll	Rimuove tutti gli obiettivi assegnati.
data/put	Aggiunge (o aggiorna) i valori per un utente, relativi ad un obiettivo.
data/get	Recupera i valori di un utente per un obiettivo.
data/deleteAll	Rimuove tutti i valori.
auth/me	TODO
auth/login	TODO

Table 1: Descrizione delle operazioni API

Tecnologie

Nella seguente sezione verranno analizzate le principali tecnologie utilizzate per lo sviluppo del progetto. Per ogni tecnologia verrà fornita una breve descrizione e verranno elencati i vantaggi e gli svantaggi che hanno portato alla scelta fine. La scelta delle tecnologie è stata fatta tenendo conto delle conoscenze pregresse dei membri del gruppo e della loro disponibilità a imparare nuove tecnologie, così come della loro popolarità e della loro adozione in ambito aziendale. Inoltre, le scelte dello stack tecnologico hanno tenuto conto dei requisiti che ambo gli stakeholders (ricordiamo la dualità di questo progetto, università e azienda) hanno espresso.

TypeScript

Descrizione

TypeScript è un linguaggio di programmazione open-source sviluppato da Microsoft. È un super-set di JavaScript, ovvero un linguaggio che estende le funzionalità di un altro linguaggio, in questo caso JavaScript.

Vantaggi

Tra i vantaggi di TypeScript troviamo:

- **TypeScript è JavaScript**: essendo un super-set di JavaScript, TypeScript è compatibile con tutte le librerie JavaScript esistenti;
- **TypeScript è un linguaggio fortemente tipizzato**: questo permette di avere un codice più robusto e di trovare più facilmente errori durante la fase di sviluppo;
- **TypeScript è un linguaggio orientato agli oggetti**: questo permette di avere un codice più modulare e di avere una maggiore astrazione rispetto a JavaScript.

Svantaggi

Tra gli svantaggi di TypeScript troviamo:

- **TypeScript è un linguaggio fortemente tipizzato**: l'altro lato della medaglia è che il programmatore deve dichiarare il tipo di ogni variabile, questo può portare ad un aumento della verbosità del codice.

React

Descrizione

React.js è un framework e una libreria open-source sviluppata da Facebook, utilizzata per costruire in modo rapido ed efficiente interfacce utente interattive e applicazioni web con molto meno codice rispetto all'utilizzo di JavaScript "puro".

In React, lo sviluppo delle applicazioni viene fatto creando componenti riutilizzabili, simili a blocchi Lego indipendenti. Questi componenti rappresentano parti individuali di un'interfaccia finale che, quando combinati, costituiscono l'intera interfaccia utente dell'applicazione.

Vantaggi

Tra i vantaggi di React troviamo:

- **Componenti Riutilizzabili**: React permette la creazione di componenti personalizzabili e riutilizzabili, migliorando l'efficienza dello sviluppo;
- **Ampio ecosistema**: la comunità al supporto è molto ampia con molte risorse e librerie disponibili per estendere le funzionalità dell'applicazione;
- **Curva di apprendimento**: React è molto semplice da imparare, soprattutto per chi ha già esperienza con JavaScript;
- **SEO friendly**: è famoso per essere SEO friendly.

Svantaggi

Tra gli svantaggi di React troviamo:

- **JSX**: JSX è una sintassi che permette di scrivere codice HTML all'interno di JavaScript, questo può portare ad un aumento della verbosità del codice e c'è chi suggerisce che questo possa portare ad un aumento della ripidità della curva di apprendimento;
- **Documentazione**: la documentazione non è tra le più complete e chiare, complice anche la velocità con cui React evolve.

MUI

Descrizione

MUI è una libreria di componenti React che implementa il Material Design di Google. Questa libreria è stata scelta per la sua semplicità e per la sua popolarità.

Inoltre è stata utilizzata la libreria Materio (cartella @core del progetto), che include una serie di temi e componenti personalizzati per MUI.

Vantaggi

Tra i vantaggi di MUI troviamo:

- **Semplicità**: MUI è una libreria molto semplice da utilizzare e da configurare;
- **Popolarità**: MUI è una libreria molto popolare, questo permette di trovare facilmente soluzioni ai problemi che si possono incontrare durante lo sviluppo;
- **Documentazione**: MUI ha una documentazione molto chiara e completa.

Svantaggi

Tra gli svantaggi di MUI troviamo:

- **Personalizzazione**: La personalizzazione dei componenti alle volte può essere difficile.

Docker

Descrizione

Docker è una piattaforma open-source che permette di creare, testare e distribuire applicazioni in maniera semplice e veloce. Docker permette di creare dei container, ovvero degli ambienti virtuali isolati, in cui è possibile eseguire le applicazioni. Nel nostro caso è stato utilizzato per creare un container in cui è possibile eseguire l'applicativo sviluppato. Attraverso Docker è stato possibile eseguire il deploy dell'applicativo.

Vantaggi

Tra i vantaggi di Docker troviamo:

- **Semplicità**: Docker permette di creare e gestire i container in maniera molto semplice;
- **Portabilità**: Docker permette di creare dei container che possono essere eseguiti su qualsiasi sistema operativo.

Svantaggi

Tra gli svantaggi di Docker troviamo:

- **Risorse**: Docker utilizza molte risorse del sistema;
- **Curva di apprendimento**: Docker è una tecnologia molto ampia e complessa, questo può portare ad una curva di apprendimento molto ripida.

MongoDB

Descrizione

MongoDB è un database NoSQL flessibile che consente di gestire dati non strutturati o semi-strutturati.

Vantaggi

Tra i vantaggi di MongoDB troviamo:

- **Flessibilità**: MongoDB permette di gestire dati non strutturati o semi-strutturati;
- **Scalabilità**: MongoDB permette di scalare facilmente il database;
- **Agilità di sviluppo**: MongoDB permette di sviluppare applicazioni in maniera molto veloce, soprattutto grazie all'ottima integrazione con **Mongoose**.

Svantaggi

Tra gli svantaggi di MongoDB troviamo:

- **Dati duplicati**: può soffrire di problemi di dati duplicati;
- **Risorse**: MongoDB utilizza molte risorse del sistema, come ad esempio la memoria del sistema.

Mongoose

Descrizione

Mongoose è una libreria di ODM (Object-Document Mapping) per MongoDB e Node.js. Essenzialmente, Mongoose offre un set di strumenti per semplificare l'interazione con MongoDB, un database NoSQL orientato ai documenti, tramite una rappresentazione in stile oggetto dei dati.

Vantaggi

Tra i vantaggi di Mongoose troviamo:

- **Agilità di sviluppo**: Mongoose semplifica notevolmente lo sviluppo di applicazioni Node.js che utilizzano MongoDB come database di back-end. Fornisce una struttura chiara e coerente per definire schemi, gestire la validazione dei dati e creare query.

Svantaggi

Tra gli svantaggi di Mongoose troviamo:

- **Complessità aggiuntiva**: Mongoose aggiunge una certa complessità all'applicazione a causa della necessità di definire schemi e modelli. Questo potrebbe essere eccessivo per progetti molto semplici o prototipi.

Node.js

Descrizione

Node.js è un ambiente di runtime open-source, multi-piattaforma e orientato agli eventi per l'esecuzione di codice JavaScript lato server.

Vantaggi

Tra i vantaggi di Node.js troviamo:

- **Velocità**: Node.js è molto veloce grazie al suo modello di I/O asincrono;
- **Ampio ecosistema**: la comunità al supporto è molto ampia con molte risorse e librerie disponibili per estendere le funzionalità;
- **JavaScript Everywhere**: permette di utilizzare lo stesso linguaggio di programmazione (JavaScript) sia nel frontend che nel backend dell'applicazione, semplificando la condivisione del codice e la sincronizzazione tra il lato client e il lato server.

Svantaggi

Tra gli svantaggi di Node.js troviamo:

- **Single threaded**: sebbene sia in grado di gestire molte richieste simultaneamente, Node.js è single threaded, questo può portare ad un aumento del tempo di risposta in caso di richieste molto complesse;

- **Non adatto per calcoli intensivi**: non è la scelta migliore per applicazioni che richiedono calcoli intensivi o operazioni CPU-bound.

Next.js

Descrizione

Next.js è un framework web basato su React che offre un modo semplice ed efficiente per sviluppare applicazioni web reattive e ad alte prestazioni. La sua crescente popolarità nella comunità degli sviluppatori è attribuibile alle sue funzionalità avanzate e alla sua intuitiva facilità d'uso.

Vantaggi

Tra i vantaggi di Next.js troviamo:

- **Server-side rendering**: permette di eseguire il rendering delle pagine lato server, questo permette di avere un sito web più performante e più accessibile;
- **Static site generation**: permette di generare un sito web statico, questo permette di avere un sito web più performante e più sicuro;
- **Zero configuration**: è progettato per essere facile da configurare e richiede meno configurazioni rispetto ad altri framework.

Svantaggi

Tra gli svantaggi di Next.js troviamo:

- **No Dynamic Routing**: non è ready-to-use per quanto riguarda il dynamic routing, e per questo è necessario Node.js.

YARN

Descrizione

YARN è un package manager per Node.js che permette di gestire le dipendenze di un progetto. È stato scelto per la sua semplicità e per la sua popolarità.

Vantaggi

Tra i vantaggi di YARN troviamo:

- **Semplicità**: YARN è molto semplice da utilizzare;

- **Velocità**: è noto per le sue prestazioni migliorate rispetto a npm;
- **Riproducibilità**: YARN permette di bloccare le versioni delle dipendenze, questo permette di avere un ambiente di sviluppo riproducibile.

Svantaggi

Tra gli svantaggi di YARN troviamo:

- **Dimensioni**: il binario di YARN è molto più grande rispetto a quello di npm.

Axios

Descrizione

Axios è una libreria JavaScript che permette di effettuare richieste HTTP da Node.js o da browser.

Vantaggi

Tra i vantaggi di Axios troviamo:

- **Semplicità d'uso**: Axios fornisce un'API semplice e intuitiva per effettuare richieste HTTP, sia GET che POST, che si integra facilmente con il codice JavaScript esistente;
- **Supporto alle Promise**: utilizza le Promise per la gestione delle richieste asincrone, il che semplifica notevolmente la gestione del flusso di dati e delle operazioni.

Svantaggi

- **Dimensioni**: Axios è una libreria molto grande, questo può portare ad un aumento delle dimensioni del bundle finale.

CASL

Descrizione

CASL è una libreria JavaScript che permette di gestire i permessi in maniera semplice e dichiarativa. È quindi utilizzata per gestire il controllo degli accessi e l'autorizzazione in applicazioni web, attraverso la definzione di regole d'accesso dinamiche e controllare chi ha il permesso di eseguire determinate azioni all'interno dell'applicazione.

Vantaggi

Tra i vantaggi di CASL troviamo:

- **Controllo Fine-Grained degli Accessi**: consente di definire regole di accesso molto precise;
- **Integrazione**: si integra facilmente con diverse tecnologie e framework, come ad esempio React e Node.js;
- **Dinamic policies**: permette di definire regole di accesso dinamiche, che possono cambiare a seconda del contesto.

Svantaggi

Tra gli svantaggi di CASL troviamo:

- **Curva di apprendimento**: CASL è una libreria molto ampia e complessa, questo può portare ad una curva di apprendimento molto ripida, soprattutto per chi non è avezzo al controllo degli accessi;
- **Complessità aggiuntiva**: l'integrazione e lo sviluppo di applicazioni che impiegano CASL tende ad essere più complesso.

csv-parser

Descrizione

csv-parser è una libreria JavaScript che permette di leggere e parsare file CSV.

Vantaggi

Tra i vantaggi di csv-parser troviamo:

- **Semplicità**: è progettato per essere molto semplice da utilizzare;
- **Dimensioni**: è una libreria molto piccola.

Svantaggi

Tra gli svantaggi di csv-parser troviamo:

- **Limitato**: è una libreria molto limitata, infatti non contiene un numero elavato di features.

Codice

Credentials

Il Credentials schema include le funzioni per salare la password e per verificare la correttezza della password inserita dall'utente in fase di login.

```
  credentialsSchema.pre<ICredentials>('save', function (next) {
  const credentials = this as ICredentials;

  // only hash the password if it has been modified (or is new)
  if (!credentials.isModified('password')) return next();

  // generate a salt
  bcrypt.genSalt(SALT_WORK_FACTOR, function (err, salt) {
    if (err) return next(err);

    // hash the password using our new salt
    bcrypt.hash(credentials.password, salt, function (err, hash) {
      if (err) return next(err);

      // override the cleartext password with the hashed one
      credentials.password = hash;
      next();
    });
  });
  });

credentialsSchema.methods.comparePassword = function (candidatePassword: string, cb
  bcrypt.compare(candidatePassword, this.password, function (err, result) {
    if (err) return cb(err);
    cb(null, result === true);
  });
};

const Credentials = models.Credentials || model('Credentials', credentialsSchema,

export type { ICredentialsData };
export { Credentials };
```

Esempio di definizione API NextJS

Di seguito il file pages/api/data/get.ts, che definisce l'API per ottenere i dati da visualizzare nelle dashboard. L'API verrà automaticamente esposta da NextJS alla route /api/data/get e il codice dell'handler verrà automaticamente eseguito lato server.

```
import type { NextApiRequest, NextApiResponse } from "next/types";
import { Data } from "../../../@db/schemas/data"
import connectToDatabase from "../../../@db/database"

connectToDatabase()

export default async function handler(req: NextApiRequest, res: NextApiResponse) {
    if (req.method === 'GET') {
        try {
            const { objective, username } = req.query;
            const data = await Data.find({ objective: objective, username: username
            console.log('[SERVER] found data: \n' + data);
            res.status(200).json(data);
        } catch (error) {
            res.status(500).json({ error: 'Failed to retrieve data from the databas
        }
    } else {

        res.status(400).json({ error: 'Invalid request method' });
    }
}
```

Per rendere le API accessibili solo agli utenti loggati, è stato utilizzato il middleware di NextJS, che permette di eseguire del codice prima di eseguire l'handler dell'API. Il middleware controlla che l'utente sia loggato, e in caso contrario restituisce un errore 401 (Unauthorized). In futuro, questo middleware dovrà anche implementare una verifica che l'utente sia effettivamente autorizzato ad accedere ai dati richiesti (authorization).

```
import { NextApiRequest, NextApiResponse } from "next/types";
import jwt from "jsonwebtoken";

const jwtConfig = {
  secret: process.env.NEXT_PUBLIC_JWT_SECRET as string,
};
```

```
export const authMiddleware = (handler: (
  req: NextApiRequest,
  res: NextApiResponse
) => Promise<void>) => async (
  req: NextApiRequest,
  res: NextApiResponse
): Promise<void> => {
  try {
    const token = req.headers.authorization?.replace("Bearer ", "");

    if (!token) {
      return res.status(401).json({ error: "Authorization token not provided" });
    }

    jwt.verify(token, jwtConfig.secret, (err, decoded) => {
      if (err) {
        return res.status(401).json({ error: "Invalid authorization token" });
      }

      console.log(decoded)

      // If the token is valid, call the original API handler
      return handler(req, res);
    });
  } catch (error) {
    res.status(500).json({ error: "An error occurred during authentication" });
  }
};
```

Autenticazione

Per ottenere un codice più pulito, è stato fatto buon uso dei Context di React. Un ottimo esempio è l'AuthContext.tsx, che permette di accedere alle informazioni sulla sessione attuale a tutti i componenti figli di AuthProvider.

```
  const defaultProvider: AuthValuesType = {
  user: null,
  loading: true,
  setUser: () => null,
  setLoading: () => Boolean,
```

```
  login: () => Promise.resolve(),
  logout: () => Promise.resolve()
}

const AuthContext = createContext(defaultProvider)

type Props = {
  children: ReactNode
}

const AuthProvider = ({ children }: Props) => {
  // ** States
  const [user, setUser] = useState<UserDataType | null>(defaultProvider.user)
  const [loading, setLoading] = useState<boolean>(defaultProvider.loading)

  // ** Hooks
  const router = useRouter()

  useEffect(() => {
    const initAuth = async (): Promise<void> => {
      const storedToken = window.localStorage.getItem(authConfig.storageTokenKeyNa
      if (storedToken) {
        setLoading(true)
        await axios
          .get(authConfig.meEndpoint, {
            headers: {
              Authorization: storedToken
            }
          })
          .then(async response => {
            setLoading(false)
            setUser({ ...response.data.userData })

             // Set access token in localStorage
             window.localStorage.setItem(authConfig.storageTokenKeyName, response.
          })
          .catch(() => {
            localStorage.removeItem('userData')
            localStorage.removeItem('refreshToken')
            localStorage.removeItem('accessToken')
```

```
          setUser(null)
          setLoading(false)
          if (authConfig.onTokenExpiration === 'logout' && !router.pathname.inclu
            router.replace('/login')
          }
        })
    } else {
      setLoading(false)
    }
  }

  initAuth()
  // eslint-disable-next-line react-hooks/exhaustive-deps
}, [])

const handleLogin = (params: LoginParams, errorCallback?: ErrCallbackType) => {
  axios
    .post(authConfig.loginEndpoint, params)
    .then(async response => {
      params.rememberMe
        ? window.localStorage.setItem(authConfig.storageTokenKeyName, response.da
        : null
      const returnUrl = router.query.returnUrl

      setUser({ ...response.data.userData })
      params.rememberMe ? window.localStorage.setItem('userData', JSON.stringify

      const redirectURL = returnUrl && returnUrl !== '/' ? returnUrl : '/'

      router.replace(redirectURL as string)
    })

    .catch(err => {
      if (errorCallback) errorCallback(err)
    })
}

const handleLogout = () => {
  setUser(null)
  window.localStorage.removeItem('userData')
```

```
    window.localStorage.removeItem(authConfig.storageTokenKeyName)
    router.push('/login')
  }

  const values = {
    user,
    loading,
    setUser,
    setLoading,
    login: handleLogin,
    logout: handleLogout
  }

  return <AuthContext.Provider value={values}>{children}</AuthContext.Provider>
}

export { AuthContext, AuthProvider }
```

Test

Nella sezione seguente, verranno descritti i test condotti sul prodotto.

Inizialmente, le funzionalità del sistema sono state valutate dai membri del team utilizzando i browser più diffusi, tra cui Chrome e Firefox; in seguito, il portale è stato testato anche dagli utenti finali ovvero coloro che lo sfrutteranno nel day-by-day. L'obiettivo principale di questa fase di test era garantire la portabilità del sistema. A tal fine, sono stati eseguiti test su diverse piattaforme, inclusi dispositivi mobili. I test si sono concentrati sulla verifica che le funzionalità principali e i compiti fondamentali funzionassero in modo uniforme su tutti i browser e dispositivi. Inoltre, è stata prestata particolare attenzione alla corretta e comprensibile visualizzazione dei dati. Uno dei requisiti chiave del sistema è il monitoraggio dei dati, e questo aspetto doveva essere gestito in modo semplice e intuitivo.

Usability Test

Durante la fase di verifica del prodotto, il gruppo ha deciso di effettuare un test di usabilità per valutare la qualità del prodotto.

Il prodotto è stato dapprima testato dai membri del gruppo, che hanno navigato

in totale autonomia all'interno del sito, cercando di svolgere le operazioni più comuni. In particolare sono state testate:

- la navigazione all'interno del sito;
- le operazioni di login e logout;
- la visualizzazione delle dashboard;
- l'upload dei dati (file CSV).

Successivamente, il prodotto è stato testato dai destinatari del progetto, i quali hanno riprodotto i medesimi test.

I risultati ottenuti sono stati soddisfacenti. Tutti i membri del team hanno trovato ragionevole la navigabilità del sistema così come la visualizzazione e il rendimento grafico.
Al secondo gruppo di test, ovvero i fruitori reali dell'applicazione, non erano stati forniti particolari dettagli sul funzionamento del sistema, ma è stato detto loro di navigare liberamente all'interno del sito e di svolgere le operazioni che ritenevano più comuni.

In sintesi, tutti hanno trovato la disposizione degli elementi all'interno del sistema sempre adeguata e funzionale. Sono state sollevate alcune osservazioni riguardo a meri dettagli stilistici, nessuno di questi però andava a ledere l'usability del sito. Inolte, attraverso questo secondo test è nata poi la necessità di cambiare una parte della logica legata al caricamento dei dati (come spiegato nella sezione Design, con il riferimento alla colonna up-to-date).

Accessibilità, Performance e Ottimizzazione SEO

Per quanto riguarda il testing di accessibilità, performance e ottimizzazione SEO (sebbene quest'ultimo fosse poco rilevante per il progetto), è stato impiegato il tool **Lighthouse** di Google. In breve, è un tool gratuito per monitorare e ottimizzare le performance online, dal punto di vista della velocità e dell'user experience.

Anche in questo caso, il prodotto ha ottenuto risultati soddisfacenti. Dal punto di vista dell'accessibilità e dell'ottimizzazione SEO non sono stati riscontrati problemi, sono stati ottenuti punteggi molto alti sia nella versione desktop che nella versione mobile. Per quanto riguarda le performance, i punteggi ottenuti sono stati molto buoni con un leggero calo nella versione mobile.

Nota di merito, il sito si è rivelato molto accessibile e perfomante anche nel caso in cui le dashboard contenessero un numero elevato di dati.

Deployment

Docker offre un ambiente di containerizzazione che permette di confezionare l'applicazione web e il database MongoDB insieme, garantendo che siano consistenti e facili da distribuire su qualsiasi ambiente, indipendentemente dalle differenze di configurazione.

È stato quindi creato un Dockerfile per buildare l'immagine del server, e un docker-compose.yaml per orchestrare il deployment dell'applicazione web e del database.

Il Dockerfile è multistep per minimizzare il peso dell'immagine finale. La prima immagine è più pesante e contiene tutte le dipendenze necessarie per buildare, mentre la seconda solo quelle necessarie a runtime.

Il docker compose è attualmente configurato per l'utilizzo sul server nel quale è stato effettuato il deployment, ma può essere facilmente adattato per l'utilizzo in locale:

```
version: '3'

services:

  mongo:
    container_name: mongo
    image: mongo:4.4.18
    volumes:
      - mongodb_data:/data/db
    restart: always
    ports:
      - "27017:27017"

  server:
    container_name: server
    ports:
      - "3033:3033"
    build:
      context: ./
      dockerfile: Dockerfile
```

```
    depends_on:
      - mongo
    # environment:
      # - MONGODB=mongodb://mongo

volumes:
  mongodb_data:
```

0.1 Backend