

Machinevisor

Applicazioni e Servizi Web

Eleonora Bertoni - 0001057555 {eleonora.bertoni3@studio.unibo.it}
Elisa Albertini - 0001050864 {elisa.albertini3@studio.unibo.it}
Denys Grushchak - 0001027862 {denys.grushchak@studio.unibo.it}

01 Agosto 2022

Introduzione

Machinevisor è un'applicazione web che permette di monitorare e controllare il corretto funzionamento dei macchinari in un impianto industriale. Grazie alla visualizzazione della pianta dei macchinari, dei dati e dei grafici aggiornati in real time l'utente potrà supervisionare l'impianto. Il sistema permette inoltre di rilevare gli stati dei macchinari tra cui situazioni di allarme quando i sensori rilevano dati anomali.

Capitolo 1

Requisiti

1.1 Requisiti funzionali

Funzionalità:

- Visualizzazione della pianta dei macchinari
 - Visualizzazione dello stato del macchinario real time tramite differenti colori. È possibile capire quando è spento, acceso o in allarme.
 - È possibile cliccare gli elementi della pianta corrispondenti a un macchinario e, tramite un piccolo menù, arrivare alla pagina con i dettagli.
- Visualizzazione di dati di funzionamento dei macchinari in real time
- Modalità di controllo dei macchinari
 - Accensione e spegnimento
 - Scelta della modalità di lavoro
 - Scelta del periodo di invio dei dati
- Visualizzazione grafici
 - Grafici per singoli macchinari
 - Grafici aggregati
- Visualizzazione statistiche aggregate
- Sistema di simulazione dei sensori collegati ai macchinari
- Gestione di un database per mantenere lo storico dei dati tra cui memorizzazione dei dati dei sensori (log)
- Realizzazione di un modulo di anomaly detection
- Registrazione e login degli utenti
- Visualizzazione della pagina con i dati dell'utente

1.2 Requisiti non funzionali

- Rendere l'interfaccia utente accessibile, usabile e intuitiva
- Rendere l'interfaccia responsive
- Implementare meccanismi di sicurezza

Capitolo 2

Design

Ci si è basati su User Centered Design individuando gli utenti target dell'applicazione e sviluppando la Personas. Questo passo è stato molto utile per individuare le principali funzionalità del sistema e per capire come impostare l'interfaccia grafica in modo da soddisfare l'utente in termini di accessibilità e usabilità. Nello sviluppo sono stati sempre tenuti in considerazione i principi **KISS** e **DRY** per realizzare codice di buona qualità.

2.1 Definizione della Personas

Personas: Mario

Mario è un operaio di un'industria che ha un insieme di macchinari che vanno osservati e gestiti.

- È necessario che Mario sia in grado di controllare tutti i macchinari da remoto in modo da evitare di avvicinarsi ai macchinari in funzione e di aumentare la velocità di risposta in caso di anomalie. Quindi l'azienda ha richiesto la realizzazione di una web app che possa aiutare Mario a svolgere il suo lavoro collegando tutti i macchinari in rete
- Mario è stato assunto da poco quindi ha ancora difficoltà nel distinguere i vari problemi che possono avere i macchinari quindi sarà scopo dell'app aiutarlo a individuare la causa delle anomalie.
- Mario per ulteriori dettagli potrà accedere allo storico dei log dei macchinari
- Mario vuole usare l'applicazione anche per consultare le statistiche sulla fabbrica e sui singoli macchinari periodicamente
- Mario desidera un'app che gli permetta facilmente di controllare i macchinari e riduca la possibilità di suoi errori

2.2 Design del frontend

2.2.1 Mockup

Di seguito verranno riportati i mockup realizzati in fase di analisi dei requisiti del progetto per essere utilizzati come base per l'interfaccia utente. Abbiamo deciso di utilizzare principalmente il formato mobile per seguire la regola “mobile first”. La versione desktop è stata derivata.

Prima di iniziare con lo sviluppo del frontend, i mockup sono stati revisionati da un focus group di cinque persone che li ha approvati. Come strumento è stato utilizzato Figma tramite cui è stata realizzata anche la loro versione prototipata in modo che gli utenti del focus group potessero utilizzarla in modo interattivo.

- **Login:** La pagina di login è stata realizzata con una card centrale contenente i vari controlli. In fase di sviluppo questa pagina è stata utilizzata come riferimento anche per la realizzazione della pagina di registrazione utente.
- **Home:** Nella pagina della home sono presenti la cartina dell'impianto e l'overview dei dati registrati dai macchinari. Per ogni macchinario presente nella cartina, una volta cliccato, appare un menù con il suo nome e un bottone che porta alla pagina specifica.
- **Macchinario:** I mockup del macchinario, realizzati anche in versione desktop data la quantità di elementi presenti, presenta i vari controlli utilizzati per la gestione dei macchinari, come l'interruttore di accensione e spegnimento etc., e riporta sia i dati del macchinario sia i valori raccolti costantemente dai sensori ad esso collegati. Quando un valore registrato è critico viene sottolineato di rosso, mentre allo stato normale è verde.

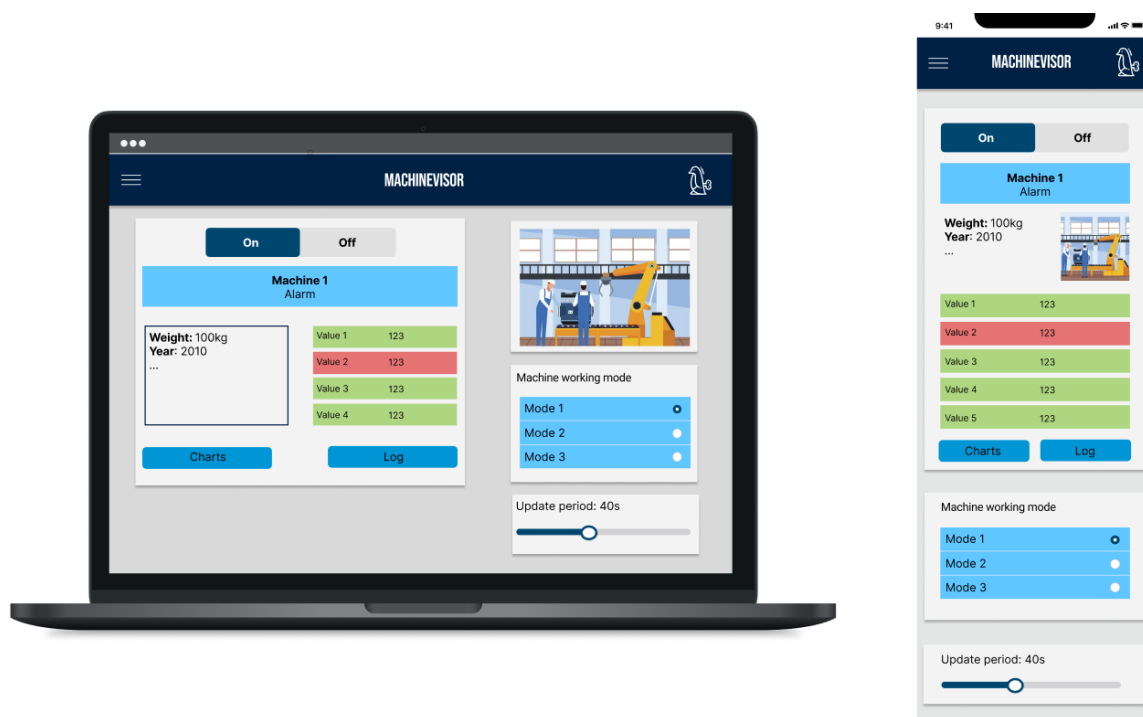


Figura 2.1: Mockup della pagina del macchinario. A sinistra la versione desktop e a destra la versione mobile

- **Grafici e Log:** Queste due pagine sono state realizzate in maniera simile. I dati dei log vengono contenuti in singole card in modo che la loro lettura sia più scorrevole, come i grafici.
- **Utente:** La pagina utente contiene i dati anagrafici e la schedule settimanale dell'utente.

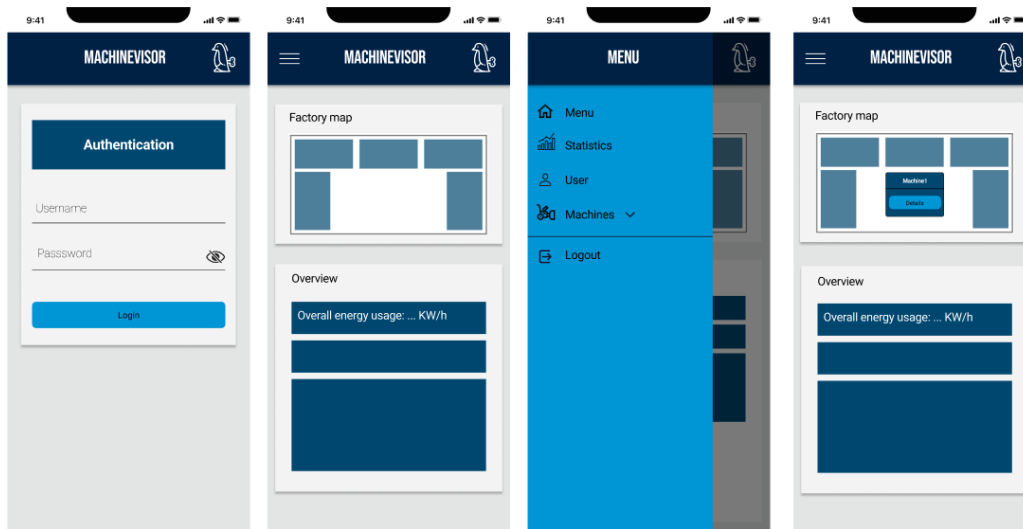


Figura 2.2: Da sinistra verso destra i mockup della pagina di login, home, home con menù aperto e home dopo click sulla mappa

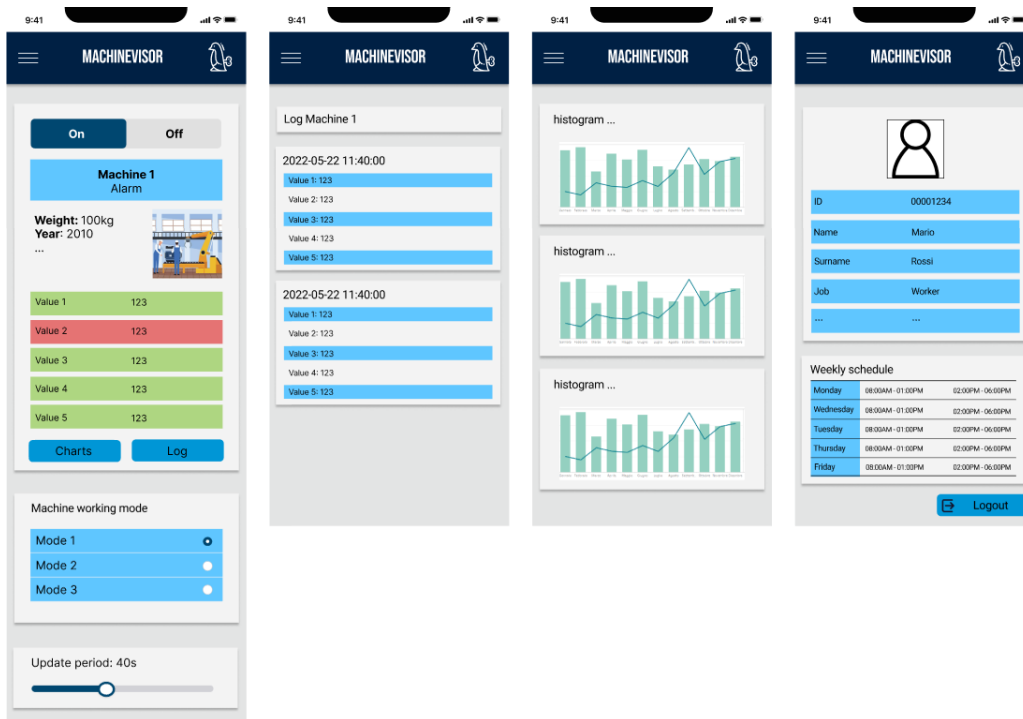


Figura 2.3: Da sinistra verso destra i mockup della pagina del macchinario, log, grafici e utente

2.2.2 Frontend

Il frontend è stato strutturato cercando di rendere i vari elementi il più modulari possibile. Per questo motivo sono stati realizzati vari sotto-package.

Components

Contiene tutti i componenti Angular che sono stati richiamati all'interno dei componenti che svolgono il ruolo di pagina.

La divisione dei vari componenti è stata fatta in base alle funzionalità che essi avrebbero avuto all'interno delle varie pagine. Abbiamo scelto di fare questa divisione aggiuntiva, rispetto a quella delle pagine, in modo che fosse più semplice testare, modificare e aggiungere funzionalità rispettando il principio della modularità già citato precedentemente.

In questa sezione vogliamo sottolineare tre componenti in particolare che sono navbar, footer e menù. Essi sono fissi in tutte le pagine dell'applicativo, anche se le loro funzionalità possono essere variabili in base al fatto che l'utente sia autenticato o meno.

- **Navbar:** la navbar presenta il nome dell'app, il logo, che se cliccato permette di navigare alla pagina di home e l'hamburger che fa aprire il menù.
- **Footer:** il footer contiene contatti e link utili all'utente.
- **Menu:** permette all'utente di navigare con comodità all'interno dell'applicazione e permette anche di fare logout.

Pages

Contiene tutti i componenti che svolgono il ruolo di pagina, racchiudendo al loro interno gli elementi del package *components*.

Le rotte per le varie pagine sono state gestite all'interno del modulo app-routing. Per impedire agli utenti di accedere a pagine a cui non dovrebbero, sono state usate delle guardie che verranno spiegate nel dettaglio successivamente. La seguente divisione è stata dedotta naturalmente dalla fase di analisi dei requisiti:

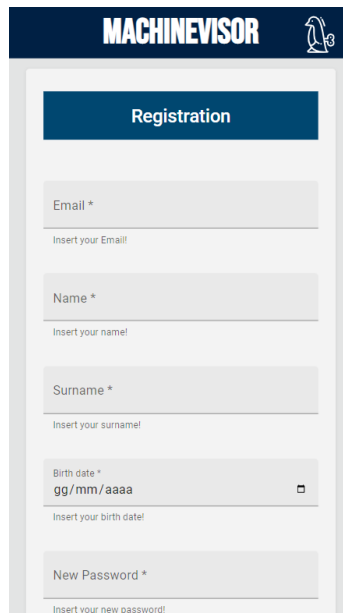
- **Registration:** La pagina della registrazione contiene tutti i campi utili all'inserimento nel database di un nuovo utente e contiene anche i vari controlli sugli input. L'utente deve compilare necessariamente tutti i campi, la data di nascita non può essere successiva alla data corrente, la password deve essere di almeno otto caratteri e contenere una lettera maiuscola, una minuscola, un numero e, infine, la password di conferma deve corrispondere a quella precedentemente inserita. Una volta sottomesso il form, nel caso l'email sia già presente nel database, all'utente verrà segnalato l'errore e dovrà ricompilare il form. Altrimenti la registrazione andrà a buon fine e l'utente verrà loggato automaticamente all'interno dell'applicazione.
- **Login:** La pagina di login permette all'utente di inserire le proprie informazioni per autenticarsi e di navigare alla pagina della registrazione. Nell'eventualità che le credenziali sottomesse non siano corrette, viene mostrato un errore all'utente che sarà invitato a reinserirle da capo. Se il login va a buon fine l'utente verrà reindirizzato alla pagina di home. Dato che l'utente non è loggato in questa fase del suo utilizzo dell'app, abbiamo utilizzato una guardia per impedire all'utente di accedere, tramite URL, alle pagine dell'applicazione che

richiedono l'autenticazione in quanto contenenti dati sensibili. Per questo stesso motivo non è presente neanche l'icona dell'hamburger (in modo che non si possano carpire informazioni private dal menù stesso) e l'icona in alto a destra non permette la navigazione alla pagina di home.

- **Home:** Nella pagina di home sono presenti la cartina dell'impianto e l'overview. La cartina presenta i vari macchinari colorati in base allo stato che hanno al momento e si aggiornano continuamente (i colori relativi ai vari stati sono indicati nella legenda al di sotto della mappa stessa). Cliccando sui vari macchinari viene mostrato un piccolo menù con il nome del macchinario e un bottone che permette di navigare alla specifica pagina. L'overview, invece, presenta dei dati di interesse campionati su 12 h. Per navigare a questa pagina si può usare sia il menù sia cliccare sull'icona in alto a destra.
- **Macchinario:** La pagina dei singoli macchinari presenta vari elementi: l'interruttore per l'accensione e lo spegnimento (per lo spegnimento l'utente dovrà confermare di voler eseguire l'azione tramite una dialog), le informazioni sul macchinario stesso, come l'immagine, il nome, il peso etc., tutti i dati aggiornati in tempo reale raccolti dai sensori ad esso collegati, come stato, temperatura, consumo energetico etc., i controlli per cambiare la modalità e il periodo corrente. Sono inoltre presenti due bottoni che permettono la navigazione alla pagina dei relativi grafici e log. Le pagine dei singoli macchinari sono raggiungibili sia cliccando sulla piantina dell'impianto (nella pagina di home) sia grazie al menù con apposita tendina.
- **Log:** La pagina dei log mostra i singoli report dei sensori collegati ad un macchinario. Tramite un menù a tendina è possibile selezionare se si vogliono vedere i log in tempo reale o si vogliono vedere gli ultimi n log prodotti.
- **Charts:** La pagina dei grafici del macchinario mostra una selezione di grafici real time dei valori più rappresentativi raccolti dai sensori.
- **Statistics:** La pagina delle statistiche, raggiungibile tramite il menù, mostra dei grafici sull'andamento generale di tutti i macchinari dell'impianto.
- **User:** La pagina dell'utente permette all'utente loggato di vedere i propri dati anagrafici e la propria schedule settimanale. La pagina permette anche di fare logout (cosa che può essere fatta anche dal menù).

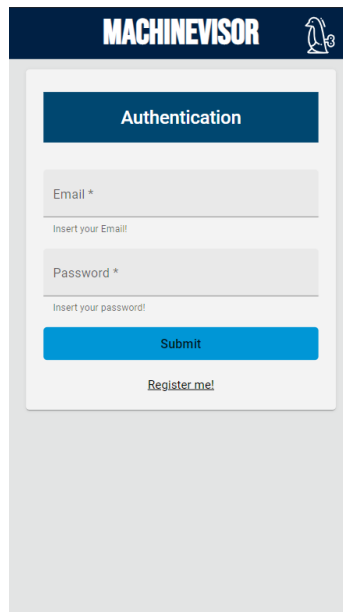
Tutte queste pagine hanno una guardia che impedisce all'utente di accedere, tramite URL, alle pagine di login e registrazione se prima non si è fatto logout.

Tutti i componenti Angular realizzati presentano la classica divisione in file HTML, scss e TypeScript. Inoltre, per realizzarli sono stati utilizzati principalmente componenti di Angular Material, in modo che lo stile generale dell'applicazione fosse mantenuto ovunque. Tutti i componenti Material utilizzati sono stati importati in un modulo apposito.



The registration form is titled "Registration" in a dark blue header. It contains five input fields: "Email *" with placeholder "Insert your Email!", "Name *" with placeholder "Insert your name!", "Surname *" with placeholder "Insert your surname!", "Birth date *" with placeholder "gg/mm/aaaa" and a calendar icon, and "New Password *" with placeholder "Insert your new password!".

Figura 2.4: Nella pagina di registrazione è possibile inserire i dati e procedere con la registrazione



The authentication form is titled "Authentication" in a dark blue header. It contains two input fields: "Email *" with placeholder "Insert your Email!" and "Password *" with placeholder "Insert your password!". Below the fields is a blue "Submit" button and a link "Register me!".

Figura 2.5: Nella pagina di login è possibile inserire le proprie credenziali e cliccare su submit per la verifica. Cliccando su "register me!" si viene indirizzati alla pagina di registrazione

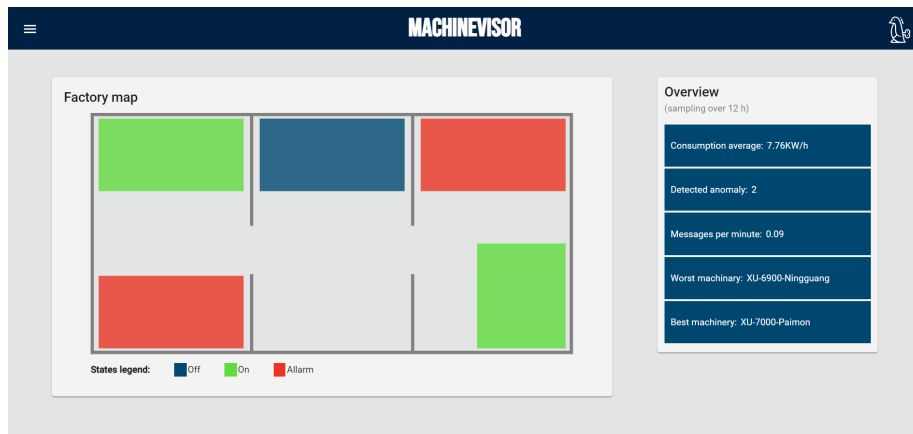


Figura 2.6: Nella pagina di home si può visualizzare la pianta dei macchinari con legenda annessa. Si può leggere l'overview e si può aprire il menu cliccando sull'hamburger

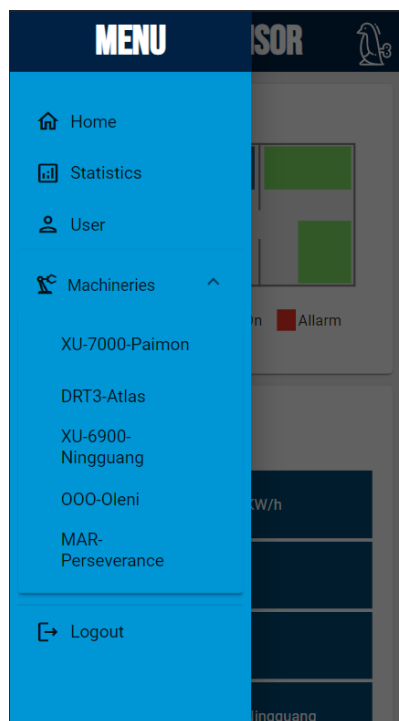


Figura 2.7: Una volta aperto, il menu permette di navigare facilmente le pagine dell'applicazione e di fare logout

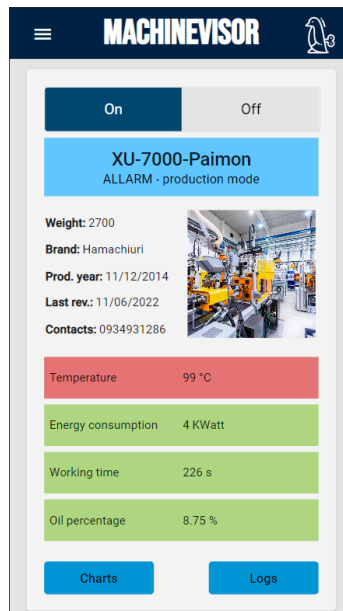


Figura 2.8: La pagina del macchinario presenta un bottone per regolare l'accensione, dalla card si possono visualizzare i valori e raggiungere le pagine di Charts e Logs

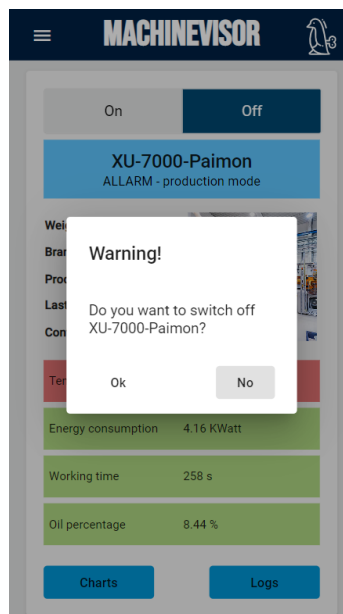


Figura 2.9: La schermata mostra il popup di conferma

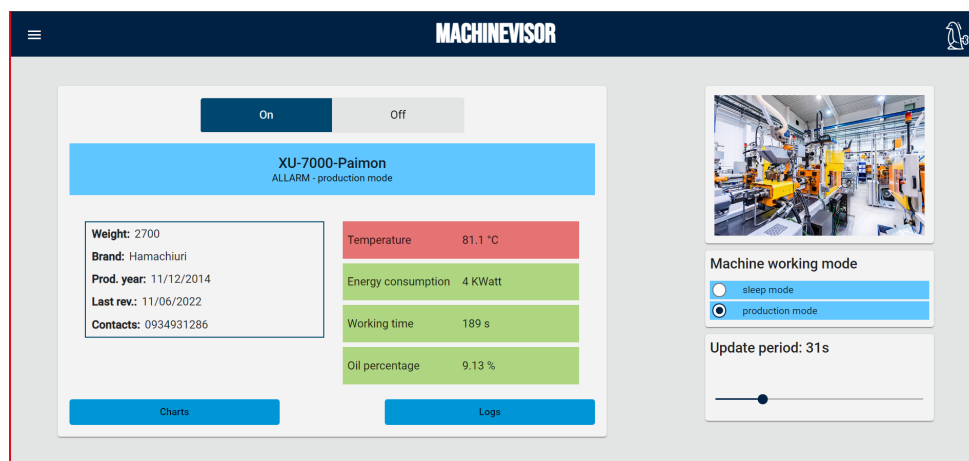


Figura 2.10: La schermata mostra il layout desktop

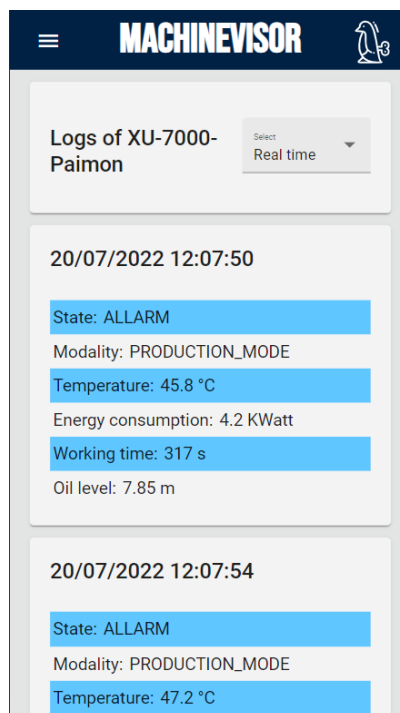


Figura 2.11: Nella pagina di log è presente il menù a tendina e le card con i dati dei log

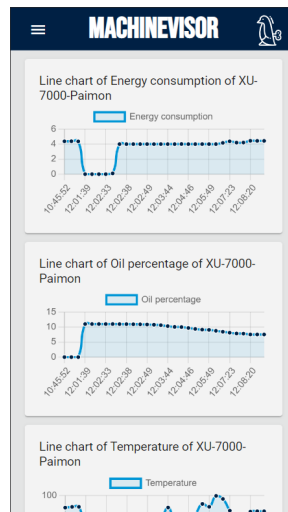


Figura 2.12: Sono presenti i grafici del consumo di energia, livello dell'olio e temperatura del singolo macchinario

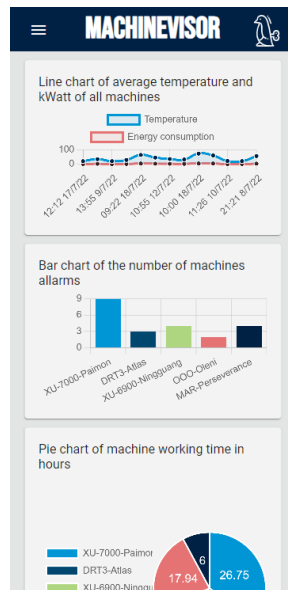


Figura 2.13: Sono presenti i grafici aggregati di temperatura e consumo, del numero di allarmi e del tempo di lavoro

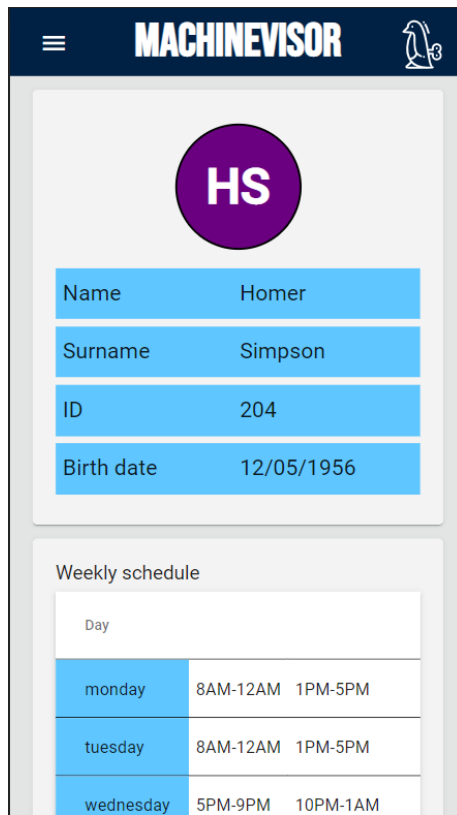


Figura 2.14: Dalla pagina utente è possibile visualizzare le informazioni e l'orario di lavoro

Utilities

- **dataInterfaces**: contiene le dichiarazioni delle interfacce relative alle varie entità del sistema.
- **guards**: contiene le guardie utilizzate per impedire agli utenti di navigare in pagine in cui non hanno accesso. Le guardie di Angular servono a definire se una determinata rotta è attiva o meno. Nel caso un utente voglia utilizzare una rotta non attiva esso viene reindirizzato a un'altra pagina.
 - **authGuard**: non permette di accedere a determinate pagine dell'applicazione se l'utente non è autenticato. La pagina di default, in questo caso, è la pagina di login.
 - **logoutGuard**: non permette all'utente di accedere alla pagina di login e registrazione se l'utente non ha prima fatto logout. La pagina di default, in questo caso, è la pagina di home.
- **services**: contiene i vari servizi utilizzati per svolgere precise funzionalità all'interno del sistema.

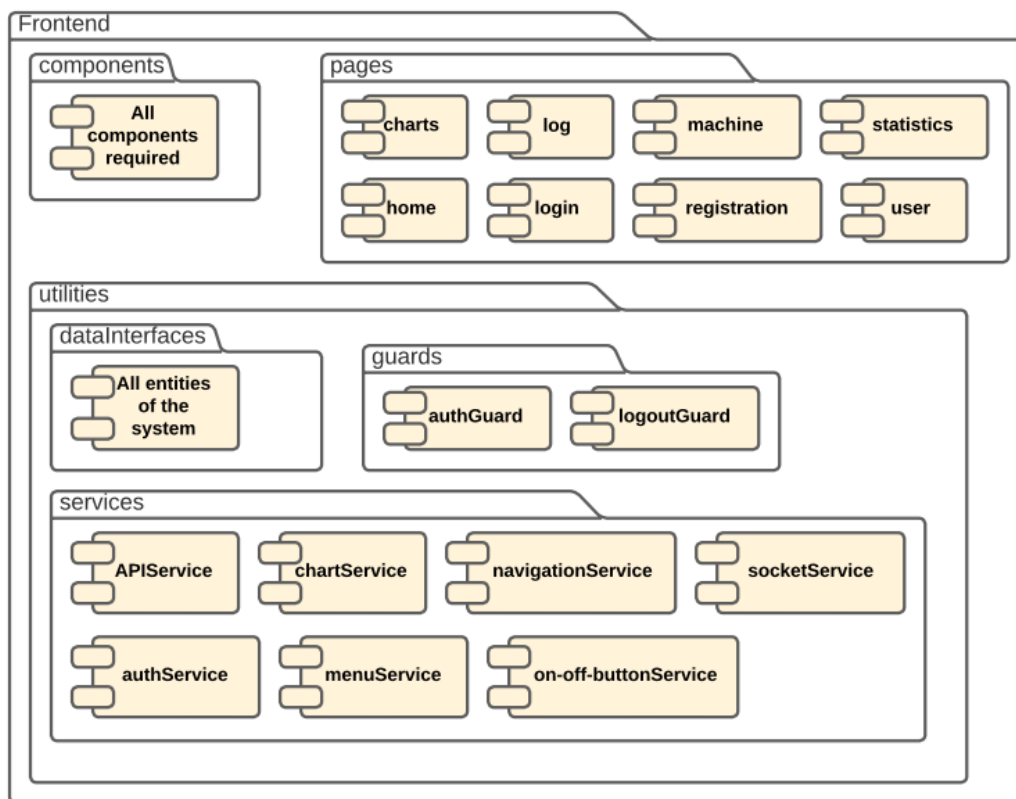


Figura 2.15: Diagramma dei package dei componenti del frontend

2.2.3 Accessibilità e Usabilità

Il sito è stato realizzato in modo che fosse usabile da tutti gli utenti e per questo motivo si è prestata particolare attenzione alla fase dei mockup, realizzandoli con cura e producendone una funzione prototipata che gli utenti potessero davvero utilizzare. La versione utilizzata dal focus group è disponibile al seguente [link](#).

Per quanto riguarda l'accessibilità, in primis, i colori della palette sono stati scelti mediante un tool di Google chiamato Google Color che, scelti due colori principali, crea una palette accessibile e indica colore e grandezza delle scritte in modo che siano leggibili. Con il tool [Contrast Checker](#) è stato effettuato un controllo aggiuntivo dei colori per verificare che rispettassero lo standard WCAG AAA.

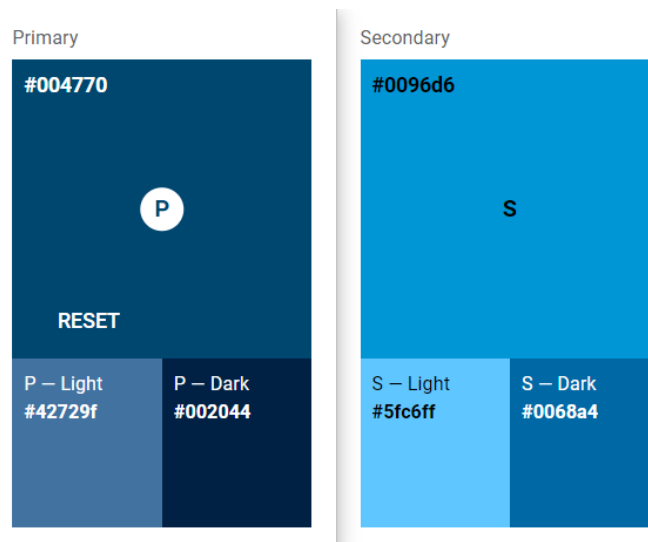


Figura 2.16: Palette dei colori dell'applicazione

Per quanto riguarda i vari componenti, invece, è stata controllata la sezione accessibilità di ogni componente Material utilizzato e si sono fatte tutte le aggiunte necessarie a rendere i componenti più accessibili (es. aria-label nei button).

I grafici sono stati resi accessibili tramite la generazione di una fonte alternativa: è possibile ottenere la stessa informazione come tabella contenente i dati rappresentati nei grafici in modo da essere leggibili dagli screen reader.

2.3 Design dell'application server

L'architettura del server backend aderisce al modello REST (Representational State Transfer) ad accensione delle comunicazioni asincrone mediante un socket.

2.3.1 Comunicazioni REST

Il server definisce cinque gruppi di risorse principali:

- **auth** - per l'autenticazione dell'utente
- **machines** - per ottenere l'informazione principale che riguarda le macchine collegate al sistema
- **users** - per ottenere l'informazione sullo user
- **overview** - per ottenere un riassunto sui dati raccolti
- **statistics** - per ottenere i dati utili ai grafici delle statistiche aggregate

2.3.2 Comunicazione tramite Socket

Le API del socket sono basati sul modello publish-subscriber, ossia tutte le API sono suddivise nei canali dove si può pubblicare i messaggi o sottoscrivere per ricevere le nuove pubblicazioni. L'application server mette a disposizione i seguenti canali di comunicazione:

- Publish
 - `machines/period` - permette di cambiare il periodo di reportistica di un macchinario
 - `machines/subscribe` - permette di sottoscrivere agli aggiornamenti degli stati dei macchinari
 - `machines/unsubscribe` - permette di disiscrivere dagli aggiornamenti degli stati dei macchinari
 - `machines/modality` - permette di cambiare la modalità di lavoro di un macchinario
 - `machines/state` - permette di spegnere ed accendere un macchinario
- Subscribe
 - `update` - permette di ricevere aggiornamenti dello stato del macchinario che è stato sottoscritto dal client
 - `periodUpdate` - permette di essere notificato in caso di cambiamenti del periodo di reportistica dei macchinari
 - `exception` - un canale per ricevere le informazioni sulle eccezioni

Un client per ottenere aggiornamenti in real time di un macchinario deve sottoscrivere al canale `machines/subscribe` specificando id di un macchinario e da quel punto l'application server invierà a quel client i dati rilevati dai sensori del macchinario quando quelli saranno ricevuti dall'application server. Tra altro l'application server memorizza in database tutti i dati inviati dai macchinari. Il processo di sottoscrizione e ricevimento dei dati si può vedere nella fig. 2.17

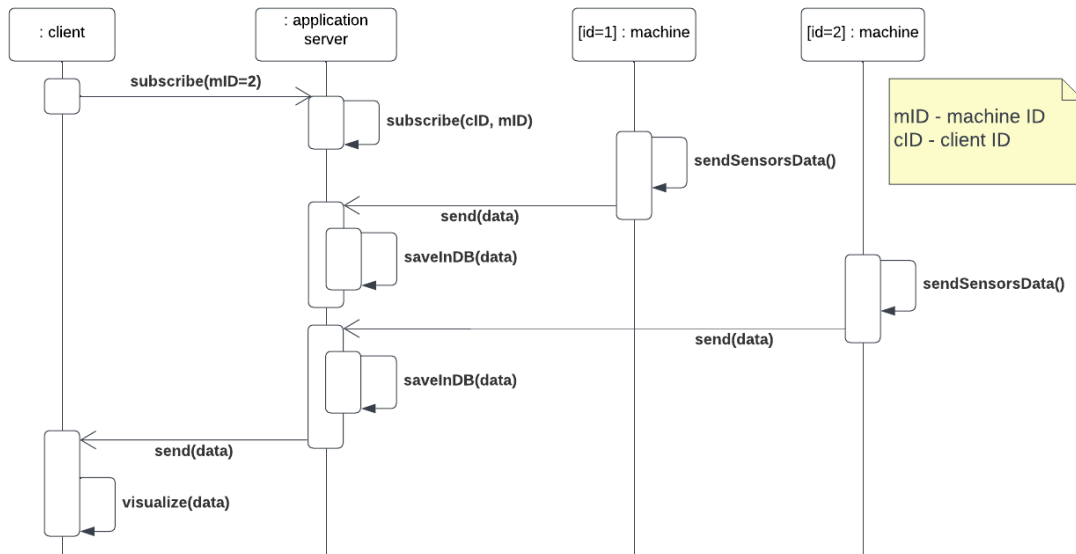


Figura 2.17: Rappresentazione della sottoscrizione di un client agli aggiornamenti dello stato del macchinario con id 2

2.4 Design dei macchinari

I macchinari vengono simulati ed eseguiti separatamente dall'application server comunicando tramite socket per ricevere i comandi e inviare dati rilevati dai sensori virtuali. I macchinari possono avere diversi insiemi di modalità di lavoro (es. sleep mode, energy economy mode ecc.) e tipi di sensori che rilevano i dati simulati.

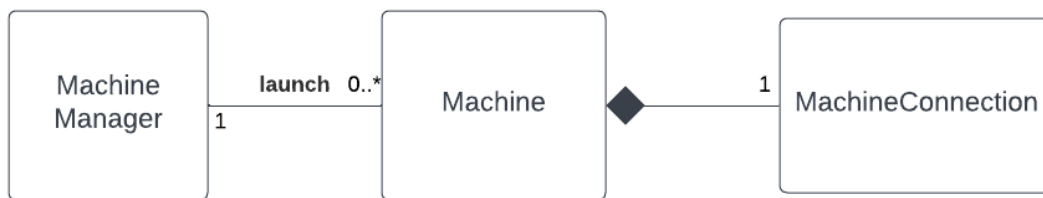


Figura 2.18: Rappresentazione semplificata del modulo di simulazione dei macchinari

Simulazione dei macchinari è composto da tre componenti fig. 2.18:

- Machine Manager - crea ed esegue i macchinari
- Machine - simulazione di un macchinario che periodicamente rileva i dati dai sensori e li manda all'application server
- MachineConnection - gestisce la connessione e i messaggi ricevuti tramite socket

La simulazione del rilevamento dei dati è realizzata in maniera realistica:

- La temperatura cresce se il macchinario è acceso ed è in modalità di produzione,
- La temperatura cala se il macchinario è in sleep mode o è spento
- La quantità di energia consumata dipende dalla modalità di lavoro del macchinario
- Se un macchinario è dotato del sensore che misura la quantità d'olio disponibile, allora quello inizierà lentamente a calare quando partirà la produzione.

2.5 Sicurezza

Json Web Token

Per permettere di eseguire solo le operazioni autorizzate è stato utilizzato *Json Web Token (JWT)* per accedere alle API, sia REST che Socket.io. Nel nostro sistema un utente che effettua il login con successo riceve un token temporaneo e questo, per tutto il periodo di durata, permette di chiamare API negando, invece, il permesso quando il token è assente. Se il token scade mentre l'utente sta navigando l'applicazione, viene reindirizzato alla pagina di login.

Validazione e hashing della password

Il sistema effettua la verifica della sicurezza della password delle nuove registrazioni sia client side che server side. La password deve avere queste proprietà:

- Minimo 8 caratteri
- Massimo 100 caratteri
- Avere almeno un carattere minuscolo e un carattere maiuscolo
- Avere almeno una cifra

Prima della registrazione viene scelto il salt e calcolato l'hash della password. L'hash e il salt vengono salvati nel DB.

Secure Sockets Layer(SSL)

Per proteggere i pacchetti da intercettazione viene utilizzata la crittografia asimmetrica. L'application server e il server web forniscono le chiavi pubbliche ai client per poter utilizzare un canale di comunicazione protetto. Al fine dell'elaborato i certificati non sono firmati da un *Certificate Authority*.

2.6 Tecnologie

Il progetto è basato sullo stack MEAN (MongoDB, Express, Angular, Node) che è ottimizzato in modo da permettere di velocizzare e rendere più semplice (user-friendly) la creazione di applicazioni Web robuste e facilmente mantenibili. La sua principale caratteristica è l'uso di "javascript everywhere" e di un database non relazionale.

2.6.1 Node

MEAN usa Node.js come ambiente di esecuzione delle applicazioni server-side eseguendo il codice JS al di fuori del browser. Node.js permette di avere Web application con connessioni real-time, two-way, dove non solo il client, ma anche il server può iniziare una connessione. La sua architettura event-driven permette di gestire un elevato numero di connessioni simultanee.

Nel progetto noi usiamo due server creati con Node.js: web server di Angular e application server.

2.6.2 Express

Express è un Node.js web application framework server side, minimale e flessibile che significativamente semplifica l'implementazione di API REST. Il layer di funzionalità fornito non "oscura" le funzionalità di node.js. Nel progetto abbiamo pienamente sfruttato la comodità di sviluppo delle API REST e della creazione e gestione dei middleware del server.

2.6.3 MongoDB

MongoDB è un database document-oriented che sostituisce il concetto di riga con un modello più flessibile: il *documento*. Questo permette di rappresentare relazioni gerarchiche complesse in un unico documento. Tra altro in MongoDB non esiste un schema predefinito.

Abbiamo sfruttato la flessibilità di MongoDB soprattutto per salvare i dati dei log dei macchinari che possono avere diversi set di campi che dipendono dai tipi di sensori installati nei macchinari.

2.6.4 Angular

Angular è una piattaforma di sviluppo di Single Page Application basata su Typescript. Come piattaforma Angular include:

- Un framework basato su componenti per realizzare applicazioni web scalabili
- Una collezione di librerie ben integrate che ricopre una vasta varietà di funzionalità includendo routing, comunicazioni client-server e tante altre
- Un insieme di strumenti per aiutare lo sviluppatore a costruire, testare e aggiornare il codice

La nostra scelta è ricaduta su Angular in quanto è il framework nativamente più completo, con la sua struttura "all-in-one" che mette a disposizione molte funzionalità integrate. Inoltre, Angular permette di avere una curva di apprendimento migliore di quella di Vue e React e di realizzare applicazioni con un diverso livello di complessità.

2.6.5 Angular Material

Angular Material è una libreria realizzata da Google che, mettendo a disposizione elementi del Material Design, permette di utilizzare un unico stile grafico per ogni componente. Questo ci ha permesso di mantenere lo stile generale migliorando il *"look and feel"* dell'applicazione.

Inoltre, mette a disposizione componenti accessibili per tutti e accuratamente testati per assicurare ottime performance e affidabilità. Il tutto mettendo a disposizione strumenti che permettono agli sviluppatori di realizzare dei componenti personalizzabili con pattern di interazione condivisi.

2.6.6 SCSS

Abbiamo scelto SCSS per lo stile del progetto Angular in quanto permette di scrivere CSS sfruttando i vantaggi di variabili, funzioni e permettendo l'organizzazione del foglio di stile in più file. Lo abbiamo utilizzato per gestire la palette di colori definendoli come variabili all'interno di un foglio di stile che abbiamo importato nei diversi componenti. Inoltre, la definizione di variabili ci ha permesso di modificare facilmente lo stile delle card presenti nella maggior parte delle pagine della nostra applicazione web.

2.6.7 Altre tecnologie per lo sviluppo

- **Mongoose** è una libreria per lavorare con MongoDB ed è basata su schemi che modellano i dati delle applicazioni. Include il casting dei tipi, la convalidazione dei dati, la creazione di query, gli hook di business logic e altro.
- **Jest** è un framework JavaScript per il testing incentrato sulla semplicità. Lo abbiamo utilizzato per verificare il corretto funzionamento delle API REST
- **Socket.io** è una libreria JavaScript per le real-time web application, che abilita la comunicazione bidirezionale in tempo reale tra client e server. È composta da due componenti aventi la stessa API, una gira client-side mentre l'altra server-side
- **Json Web Token (JWT)** è stato sfruttato per la generazione e verifica di token di autenticazione.
- **pbkdf2-password** - libreria per generare hash e salt delle password. Inoltre, è possibile in un modo semplice effettuare un controllo della validità degli hash.
- **Chart.js** - libreria per creazione dei chart
- **password-validator** - libreria che semplifica la validazione delle password

2.6.8 Tecnologie per la documentazione

Per documentare le API REST è stato utilizzato il formato OpenAPI 3.0 e come l'host per le API si è scelto di utilizzare Swagger [3].

Postman [2] è stato utilizzato per generare le chiamate REST partendo dalla documentazione OpenAPI 3.0.

OpenAPI non è progettato per descrivere le API asincrone per questo motivo per documentare le API di SocketIO è stato necessario trovare un'altra tecnologia: AsyncAPI [\[1\]](#) che utilizza una sintassi basata su OpenAPI. Come host per la documentazione asincrona si è utilizzato l'application server del progetto stesso perché il gruppo di AsyncAPI non forniscono un host gratuito come viene fatto dal gruppo di Swagger.

Capitolo 3

Codice

3.1 Gestione degli errori

Per rispettare il modello REST la gestione degli errori sfrutta fortemente i codici di stato HTTP. Per esempio la rotta `/users/userId` potrebbe restituire i seguenti codici HTTP:

- 200 (OK) - in caso di successo
- 400 (BAD REQUEST) - se lo `userId` fornito ha un formato incorretto
- 401 (UNAUTHORIZED) - se l'utente non è stato autenticato
- 404 (NOT FOUND) - se l'utente non è stato trovato
- 500 (INTERNAL SERVER ERROR) - tipicamente accade se ci sono dei problemi con la connessione con il database

Nel listing 3.1 vediamo l'implementazione della rotta `/users/userId`. Si può notare la gestione di quasi tutti codici degli errori specificati precedentemente. Non è visibile solo l'invio dell'errore 401 che viene gestito da un middleware di autenticazione.

Listing 3.1: ll

```
1  ...
2  const db_service: DBService = new DBService_mongo();
3
4  router.get('/:userId', (req:Request, res:Response) => {
5      if(isNumber(req.params.userId)){
6          let promise = db_service.getUser(+req.params.userId);
7          promise.then((user) =>{
8              if(user != null){
9                  res.json(user);
10             }else{
11                 res.status(status.NOT_FOUND).send(makeErr("Not found", 'User not
12                     found'));
13             }
14         }).catch((err)=> res.status(status.INTERNAL_SERVER_ERROR).send(makeErr("
15             ServerError", err)));
16     }else{
```

```

15         res.status(status.BAD_REQUEST)
16         .json(makeErr("Bad request", "UserId is not a number"));
17     }
18 });

```

3.2 Generazione di query dinamiche

Considerando il fatto che ogni macchinario potrebbe avere diversi tipi di sensori, per poter visualizzare i dati nei chart è necessario costruire una query dinamica che richiede solo i dati che riguardano i sensori di un determinato macchinario. Il codice è rappresentato nel listing 3.2 dove nelle linee 5-6 si può vedere la generazione di un raggruppamento che in seguito viene utilizzato nell'aggregazione di MongoDB.

Listing 3.2: Il codice genera una query per richiedere i dati dei chart di un determinato macchinario

```

1 public getMachineCharts(machine_id: number, values: string[]): Promise<any []>{
2     return new Promise((resolve, reject) => {
3         if(!this.isConnected()) reject(this.getErrorDBNotConnected());
4         let gr: {[k: string]: any} = {}
5         gr['_id'] = "$machine_id";
6         values.forEach(e => gr[e] = {"$push": {value: ("$" + e), label: "$timestamp"}})
7         let query = [
8             {$match: {machine_id: machine_id}},
9             {$sort: {timestamp: -1}},
10            {$limit: 30},
11            {$sort: {timestamp: 1}},
12            {$group: gr},
13            {$project: {_id: 0}}
14        ]
15        Log.aggregate(query).then(ris =>{
16            resolve(ris)
17        })
18    })
19 }

```

3.3 Generazione delle tabelle accessibili

Per rendere i grafici del sito accessibili ai non vedenti, per ogni elemento canvas di html, dove vengono visualizzati i chart, viene generata una tabella accessibile con tutti i dati contenuti nel grafico.

Il codice di tale generazione è mostrato nel listing 3.6. La linea 6 genera le colonne basandosi sul numero dei tipi di dati nel dataset del chart. Le linee 10-13 generano le righe della tabella, si può notare che vengono utilizzati gli attributi **headers** e **scope** che permettono di rendere la tabella accessibile.

Listing 3.3: Il codice permette di generare una tabella accessibile partendo dai dati presenti nel chart

```

1 <table>
2   <caption>Chart data alternative representation</caption>
3   <thead>

```

```

4     <tr>
5       <th id="label">Label</th>
6       <th *ngFor="let dataset of data?.datasets" id="{{dataset.label}}" scope="col" >
          {{dataset.label}}</th>
7     </tr>
8   </thead>
9   <tbody>
10    <tr *ngFor="let label of data?.labels; let i = index">
11      <th id="label-{{i}}" scope="row" headers="label">{{label}}</th>
12      <td *ngFor="let dataset of data?.datasets" headers="label-{{i}}" {{dataset.label
          }}">{{dataset.data[i]}}</td>
13    </tr>
14  </tbody>
15 </table>

```

3.4 Generazione dinamica di elementi

Il codice riportato permette grazie a ***ngFor** di istanziare dinamicamente un insieme di elementi tutti uguali che si differenziano solo per i valori contenuti. Inoltre, **ngClass** permette di assegnare la classe controllando una condizione. Nel nostro caso è stata utile per aggiornare in tempo reale lo stile dei valori inviati dai sensori delle macchine.

Listing 3.4: Il codice permette di generare dinamicamente degli elementi con classe variabile

```

1 <div class="values">
2   <ul>
3     <li *ngFor="let elem of socketData"
4
5       [ngClass]="{...}">
6
7     <p class="value-key">{{elem.key}}</p><p class="value-val">{{elem.val}} {{elem.
          unit}}</p>
8
9     </li>
10   </ul>
11 </div>

```

3.5 Visualizzazione dinamica di componenti in base al Viewport

Per poter realizzare un layout della pagina più soddisfacente per i diversi Viewport abbiamo fatto in modo che alcuni elementi potessero essere visualizzati solo per una specifica dimensione dello schermo.

Listing 3.5: Il codice permette di renderizzare il componente solo se la condizione è vera

```

1 <app-machine-image *ngIf="!isMobile" [machineID]="machineID" class="info"></
  app-machine-image>

```

Listing 3.6: Il codice permette di modificare in modo reattivo, in relazione alla Viewport, il valore di una variabile

```
1 @HostListener('window:resize', ['$event'])
2   onWindowResize() {
3     this.isMobile = window.innerWidth <= DESKTOP_SIZE;
4   }
```

Capitolo 4

Test

4.1 Test Frontend

Il sistema è stato testato dai membri del gruppo principalmente sul browser Chrome. In particolare abbiamo verificato che i dati si aggiornassero in tempo reale e che la modifica dei colori nella piantina fosse rapida in modo da non infastidire l'utente. Sono state provate diverse sequenze di azioni per controllare la corretta gestione degli errori come la richiesta di una pagina il cui accesso non è autorizzato o la registrazione con input non accettabili.

Per garantire buona usabilità dell'applicazione l'interfaccia è stata valutata rispettando le seguenti euristiche di Nielsen:

- **Visibilità dello stato del sistema:** Il sistema tiene informato l'utente sullo stato corrente visualizzando cambiamenti di stato dei macchinari, aggiornamenti dei grafici e dei dati riducendo al minimo le latenze percepite dall'utente.
- **Corrispondenza tra sistema e mondo reale:** Nel menu sono state scelte icone semplici e ben conosciute, ad esempio, per la home è stata inserita una casetta.
- **Controllo e libertà:** Una volta effettuato il login, da ogni pagina è possibile accedere a un menù che permette di navigare agevolmente l'applicazione. Inoltre, le diverse pagine sono raggiungibili con un esiguo numero di click.
- **Consistenza e standard:** I colori presenti nella palette sono stati ripetuti per tutti i componenti all'interno delle diverse pagine. Inoltre, elementi come i bottoni mantengono lo stesso stile.
- **Prevenzione dell'errore:** L'interruttore di spegnimento del macchinario una volta premuto apre una finestra che richiede la conferma dell'azione, utile se il bottone è stato premuto per sbaglio.
- **Riconoscimento anzichè ricordo:** Nei grafici sono state inserite label che in modo veloce aiutano a capire la tipologia dei dati. Per la pianta, in cui sono stati utilizzati dei colori, è stata inserita una legenda.

- **Design e estetica minimalista:** Il design è minimale e elementi tra loro collegati sono stati raggruppati in modo da impedire che l'utente si confonda.
- **Aiuto all'utente:** Nella registrazione eventuali errori come la scelta di una password debole vengono mostrati con un messaggio rosso sotto al campo interessato. Inoltre, è richiesta la conferma per lo spegnimento di un macchinario.

4.2 Test Backend

Per le API-REST è stata realizzata una serie di test automatici Jest che, dopo aver effettuato il login, chiamano le API in modo da testarne il corretto funzionamento.

Tramite Postman sono state generate le chiamate a API partendo dalla documentazione scritta in OpenAPI 3.0. Queste chiamate sono state sfruttate per verificare il corretto funzionamento delle API nella fase di sviluppo.

Capitolo 5

Deployment

Per la fase di deployment si è fatto uso di **Docker**. L'applicazione è stata strutturata in tre container: application server, server web e MongoDB.

Utilizziamo una suddivisione tra application server e web server (Angular) per motivi di modularità. In questo modo è possibile la creazione di diverse interfacce grafiche senza legarsi all'implementazione proposta con Angular e, qualora si volesse realizzare una applicazione mobile, questa dovrà interagire solamente con l'application server. Proprio per avere questo migliore isolamento del sistema si è preferito separare i server.

Per far comunicare i container viene utilizzata una rete virtuale chiamata *interna*. L'esecuzione dell'intero sistema e la popolazione del database viene fatto automaticamente grazie a `docker-compose`

5.1 Setup

1. Scaricare il codice del progetto dalla repository
2. Buildare il progetto con `docker-compose build`
3. Lanciare i container con `docker-compose up`
4. Chrome potrebbe dare dei problemi a causa dell'uso di un certificato SSL non firmato. Per questo motivo è necessario permettere al browser di accedere alla pagine web con tale certificato:
 - Aprire `https://localhost:8080` e permettere l'accesso al browser. Chiudere la pagina
 - Aprire `https://localhost:4200` e permettere l'accesso al browser

5.2 Deploy della documentazione

- La documentazione delle API REST si può consultare su [Swagger](#)
- La documentazione delle API asincrone di Socker.io è consultabile nell'application server del progetto stesso. Per aprirla:

- Se il server è attivo, aprire la pagina `https://localhost:8080/doc`
- Altrimenti aprire con il browser il file `./public/doc/index.html`

Capitolo 6

Conclusioni

La realizzazione dell'elaborato ci ha permesso di sperimentare nuove metodologie e tecnologie apprese durante il corso. In particolare, lavorando con Angular, abbiamo appreso i vantaggi della divisione in componenti che possono essere personalizzati e utilizzati in più punti dell'applicazione. Siamo stati messi alla prova con l'utilizzo di Socket.io per gli aggiornamenti in real time. Abbiamo imparato a usare la sintassi OpenAPI 3.0 per descrivere le API-REST, questa potrebbe essere utilizzata da altri sistemi per generare in modo automatico le API. Un altro punto critico è stato quello del deployment dell'elaborato su docker reso difficile dalla struttura dell'applicazione e dalla necessità di caricare in anticipo i dati. Siamo complessivamente soddisfatti di aver portato a termine un elaborato di questo livello di complessità e quantità di tecnologie richieste.

Bibliografia

- [1] Asyncapi web site. <https://www.asyncapi.com>.
- [2] Postman web site. <https://www.postman.com>.
- [3] Swagger web site. <https://swagger.io>.