Sistema antifurto per veicoli - FindME

Applicazioni e Servizi Web

Davide Crisante - 0001044805 {davide.crisante3@studio.unibo.it}

September 4, 2022

1 Introduzione

FindME è un servizio web che consente di monitorare la posizione del proprio veicolo grazie a un dispositivo fisico da installare all'interno della vettura.
L'hardware del dispositivo comprende:

- scheda SIM che permette la comunicazione via internet con il server;
- ricevitore GPS per determinarne la posizione;
- antenna per comunicare con un beacon che l'utente deve sempre portare con se quando utilizza il veicolo equipaggiato con il dispositivo FindME;
- batteria.

Il dispositivo manda il proprio stato (batteria residua, posizione, qualità del segnale GPS e internet) ai server FindME con cadenza regolare. Se il veicolo è in movimento e non viene individuato il beacon nelle vicinanze il dispositivo segnala il possibile furto al server e comunica il suo stato con una frequenza maggiore.
L'utente può monitorare lo stato del dispositivo dall'applicazione web e viene notificato in caso di presunto furto.

2 Requisiti

In accordo con le moderne tecniche di *Human-Computer Interaction* l'utente è stato considerato l'elemento fondamentale sul quale si è basata la progettazione dell'applicazione. Sono state quindi create delle **Personas** e degli **Scenarios** al fine di rappresentare un possibile gruppo di utenti di riferimento.

- **Marco**
  studente universitario, 22 anni. Al momento lavora come fattorino usando la sua automobile. Quando va al mare è costretto a parcheggiare la sua auto molto lontano dalla spiaggia a causa della difficoltà nel trovare parcheggi liberi vicini nei weekend. Ciò mette non poche preoccupazioni a Marco, che dipende economicamente dallo stato della sua vettura senza la quale non potrebbe lavorare. Da qui la necessità di poter monitorare lo stato della propria vettura e di essere allertato in tempi brevi in caso di furto.

- **Giuseppe**
  pensionato, 65 anni. Dopo una lunga vita lavorativa da meccanico, Giuseppe continua a nutrire la propria passione per i motori restaurando veicoli d'epoca. Dopo anni di fatiche è riuscito a mettere a nuovo la sua prima auto che adora guidare e sfoggiare quando si sposta. Essendo un'auto molto rara Giuseppe sa che potrebbe essere rubata da qualche mal intenzionato e per questo non si allontana mai troppo da essa. Per questo motivo vorrebbe un sistema in grado di calmare le sue preoccupazioni mentre si diverte con i suoi ex-colleghi al bar.

- **Lisa**
  parrucchiera, 32 anni, non ama le nuove tecnologie. Da anni lavora in un salone all'interno di un centro commerciale. Un giorno uscendo dal lavoro si accorge che la sua auto non era più dove l'aveva lasciata, la ricerca tramite le forze dell'ordine non è stata fruttuosa e ora deve acquistare un'auto nuova. Questa volta Lisa vorrebbe tutelarsi tramite un sistema capace di riferirle dove si trovi la sua auto cosicché in caso di necessità possa fornire indizi utili alle forze dell'ordine per aiutarla a ritrovare la sua auto. Non vuole però che la sua posizione venga tracciata e salvata perché ciò la metterebbe a disagio.

Dallo studio delle personas e dei relativi casi d'uso è stato possibile individuare i seguenti requisiti.

2.1 Funzionali

- registrarsi;
- effettuare login;
- visualizzare la posizione del dispositivo;

- visualizzare lo stato del dispositivo per rendersi conto sia della qualità del tracciamento che della necessità o meno di cambiare batteria;
- ricevere una notifica in caso di presunto furto;
- non immagazzinare dati sulla posizione non necessari.

Per mantenere il dispositivo operativo senza manutenzione il più a lungo possibile gli aggiornamenti possono non essere in tempo reale, ciò ci consente di utilizzare meno dati e meno energia.

2.2 Non funzionali

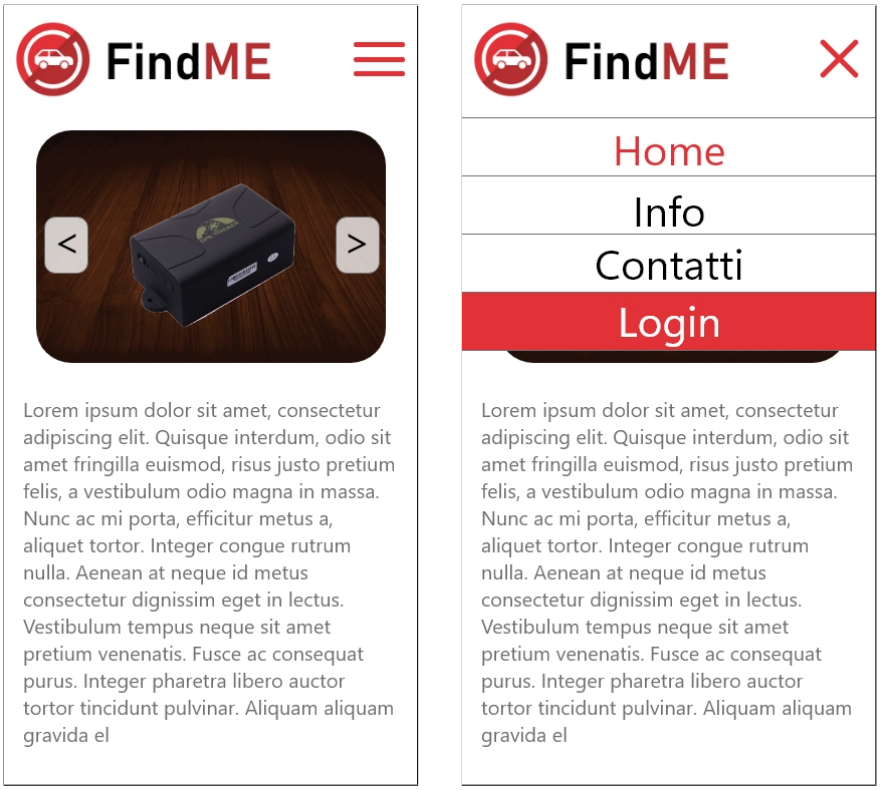
Le personas evidenziano il fatto che la principale fruizione del servizio verrebbe svolta per mezzo di un dispositivo mobile. Per questo la progettazione è stata svolta secondo l'approccio **Mobile First**, ideando quindi l'interfaccia per essere adeguata alla fruizione da dispositivi più limitanti (dispositivi mobili) in primo luogo e in secondo luogo per altri dispositivi (grazie all implementazione di un **design responsivo**).
Questo ha fatto sì che il design e i mockup siano stati incentrati sulla realizzazione di pagine adatte a dispositivi mobili da adattare successivamente a pagine più grandi come quelle di tablet o desktop.
Inoltre dati i nostri utenti di riferimento vogliamo un interfaccia semplice da utilizzare, intuitiva e reattiva.

3 Design

Il modello di design adottato è stato l' **User Centered Design** basato sulle **Personas** introdotte in precedenza. Il progetto è stato condotto seguendo una metodologia prevalentemente **Simil-Agile** caratterizzata da organizzazione ed esecuzione di *Sprint* per ogni funzionalità da implementare.

3.1 Design delle interfacce

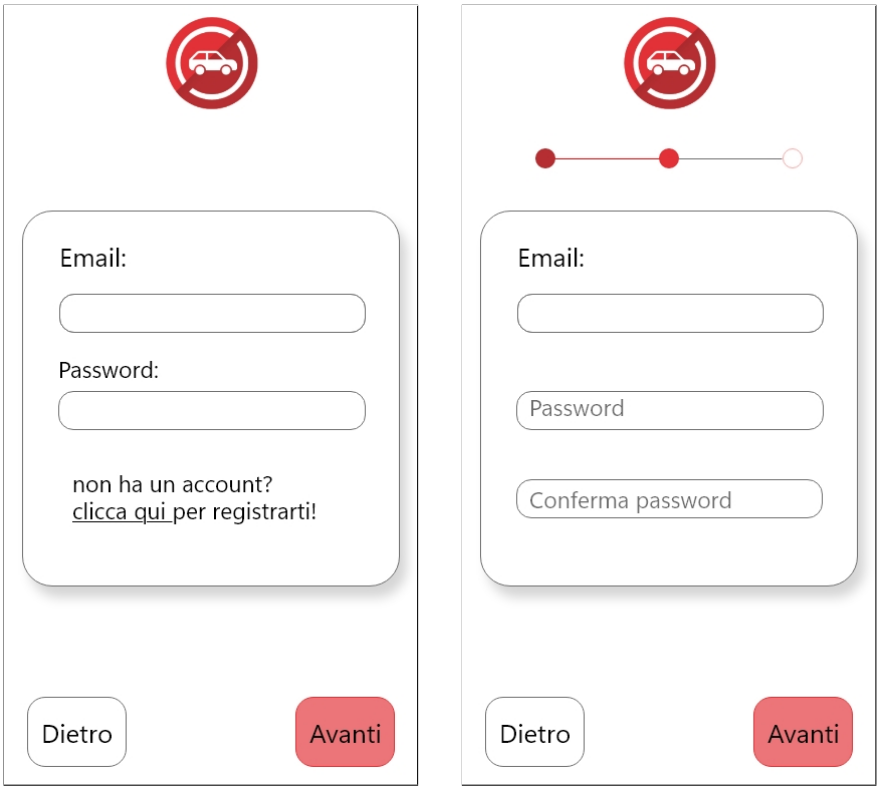
Il design delle interfacce è stato svolto cercando di sfruttare i principi **KISS** e **"Less is More"** al fine di realizzare un interfaccia semplice da fruire per gli utenti.
Di seguito sono riportati dei mockup realizzati per le sezioni ritenute più importanti dell interfaccia utente:



(a) Public home

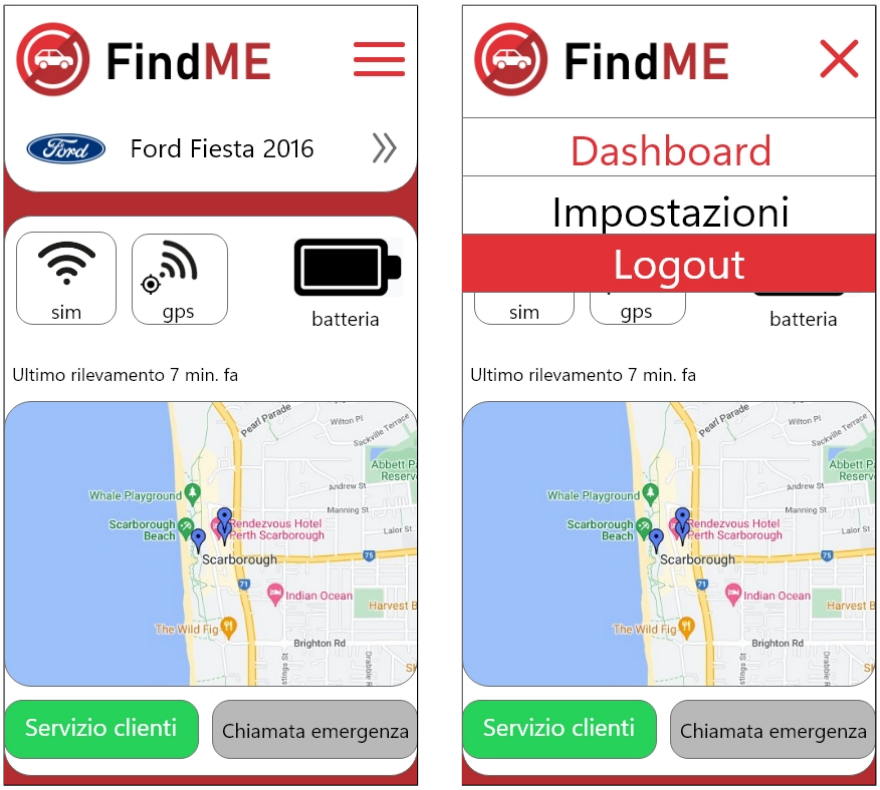
(b) Public home with menu opened

Figure 1: Public site home view



(a) Login page (b) Signin page

Figure 2: Login and signin view



(a) Dashboard home (b) Dashboard home with menu opened

Figure 3: Dashboard view

3.2 Design architetturale

Il sistema è composto da un servizio di backend che gestisce i dati e la business logic e un interfaccia di frontend per l'interazione con l'utente.

4 Tecnologie

Il sistema è stato realizzato utilizzando lo stack **MERN**.
Il backend è quindi basato su **Node.js** con l'impiego di **Express.js** per gestire le richieste e comunicare con il database non relazionale **MongoDB**. L'interfaccia utente è stata sviluppata con **React**.
La scelta di questa libreria è stata dettata dal fatto che nello studiare le tecnologie per il frontend proposti a lezione ho trovato più semplice apprendere il funzionamento di React.

4.1 Backend

Progettato per esporre un **API RESTful**.

Persistenza

Ottenuta tramite il database non relazionale MongoDB con il quale ci interfacciamo utilizzando la libreria **Mongoose**.
I dati immagazzinati sul database sono:

- ultimo stato noto dei dispositivi;
- storico degli stati dei dispositivi durante i presunti furti;
- dati personali utente;
- dati veicolo (tra cui il codice univoco del dispositivo a cui sono associati).

Autenticazione

Gestita tramite **JSON Web Token**.
In risposta a login o registrazione con esito positivo viene mandato al client un **JWT**. Successivamente il client dovrà mandare questo token per ogni richiesta che necessita di autenticare l'utente richiedente. Il token viene immagazzinato all'interno del browser e può essere usato fino alla scadenza di questo o finchè il server non dovesse cambiare *secret key*, quest'ultima viene generata casualmente all'avvio del server.

Comunicazione event-based

Vista la necessità di comunicare con gli utenti e con i dispositivi hardware in maniera event-based è stato deciso di usare la libreria **socket.io** presentata a lezione. Questa permette di comunicare tramite web socket. Nello specifico quando un dispositivo manda al server un aggiornamento riguardante il suo stato, questo viene salvato sulla memoria persistente (MongoDB) e successivamente inoltrato agli utenti associati a quel dispositivo che sono in ascolto.

Notifiche in caso di furto

Gestite tramite una API terza che permette di mandare email. Questa viene invocata tramite **Axios** inoltrando, come body, un html creato a partire da un template **PUG** contenente i dati dell'utente da notificare.

4.2 Frontend

Ho utilizzato diverse librerie e tecnologie:

- **SCSS**: ha facilitato molto la scrittura e il mantenimento dei fogli di stile, ho usato i mixins per creare in modo facile dei temi (chiaro e scuro);

- **axios**: per comunicare agevolmente con il backend tramite l'interfaccia REST;
- **socket.io-client**: per ricevere aggiornamenti sullo stato del dispositivo comunicando con il backend tramite web-socket;
- **react-router**: per gestire il routing all'interno dell'applicazione;
- **framer motion**: per gestire animazioni e transizioni tra schermate diverse;
- **react-google-maps/api**: per visualizzare la posizione del dispositivo su una mappa;

insieme ad altri componenti e librerie che hanno avuto un ruolo meno rilevante.

4.3 Dispositivo

Il dispositivo è stato simulato per mezzo di un applicazione scritta per **Java 17.0.2**.
Questa consente di modificare tutti i parametri che utilizzerebbe il vero dispositivo per comunicare:

- stato di batteria, segnale GPS e internet;
- presenza del beacon nelle vicinanze;
- se il veicolo è in movimento;
- l'indirizzo a cui mandare la richiesta tramite socket.io;
- il codice identificativo del dispositivo.

L'invio dello stato è gestito da un timer (la cui durata è modificabile). La posizione GPS che viene mandata è stata inserita arbitrariamente. Quando il dispositivo non è in allerta manda una posizione GPS fissa, quando è allertato manda ogni volta una posizione diversa seguendo l'ordine di una List riempita a mano in modo da simulare un percorso chiuso.

5 Codice

Riusabilità component:

Nel front-end il componente *'MainHeader'* è stato volutamente creato in modo tale da permetterne una grande riusabilità, questo infatti è presente in tutte le schermate eccetto quelle di login/signup. Il suo compito è quello di di visualizzare il menu e di gestire le transizioni tra le diverse schermate all'interno della stessa react-route.

Coerenza dati MongoDB:

Visto che la coerenza dei dati è fondamentale in questa applicazione è stata utilizzata la funzionalità *'Transaction'* offerta da MongoDB. Questa permette di fare commit delle modifiche durante la procedura di signin solo quando tutte le operazioni d'inserimento hanno avuto esito positivo, in caso contrario viene eseguito in automatico un rollback che invalida i nuovi dati e li rimuove dalle collezioni.

Sicurezza stati Socket.io:

Lo stato dei dispositivi (contenente anche le coordinate GPS) è un informazione sensibile e da proteggere. Questi dati vengono trasmessi mediante l'uso di *Socket.io*, il client:

- inizialmente richiede l'ultimo stato noto del dispositivo;
- successivamente si mette in attesa che il dispositivo mandi nuovi dati al server.

Nel primo caso il client viene autenticato per mezzo del JWT. Se l'autenticazione va a buon fine il socket viene inserito in una *room* contenente i socket associati a username ed email dell'utente (dati che identificano univocamente un account). Quando il server riceve aggiornamenti dal dispositivo, dopo averli salvati sul database, li inoltra a tutte le socket presenti nella room del proprietario del dispositivo previo controllo della validità dei token. Se il token di una o più socket della room scade, questa socket viene espulsa dalla room e la connessione viene interrotta.

Sincronizzazione temporale tra backend e dispositivi client:

Visto che il backend ed i client non sono sincronizzati temporalmente, nel component `'DeviceInfo'` durante la callback eseguita alla recezione di dati dal server, viene mascherata questa differenza per far sì che non sia visibile all'utente tramite la label indicante quando è stato ricevuto l'ultimo aggiornamento. La label non è comunque precisa visto che la sincronizzazione fatta non tiene conto del delay di trasmissione ma è comunque stato considerato abbastanza preciso ai fini della funzionalità in analisi. Una sincronizzazione vera e propria (es: tramite Lamport logical clock) è stata considerata non necessaria.

6 Test

Le funzionalità del sistema sono state testate su browser **Edge**, **Chrome** e **Firefox**.

L'accessibilità è stata verificata tramite **AChecker** e **W3C Validation Service**.
Le API RESTful sono state testate tramite l'uso di **Postman**.

6.1 Euristiche di Nielsen

L'usabilità dell'interfaccia è stata testata tramite la somministrazione di un questionario a 5 utenti reali.
Sono emerse le seguenti considerazioni:

- **Visibilità dello stato del sistema**: il sistema informa sempre l'utente di cosa sta accadendo;
- **Corrispondenza tra sistema e mondo reale**: il linguaggio dell'applicazione è semplice e adatta a tutti;
- **Controllo e libertà per l'utente**: le operazioni necessarie per portare a termine un task sono semplici da dedurre e il numero di operazioni è ridotto all essenziale;
- **Consistenza e standard**: consistenza nello stile visuale del sistema;
- **Prevenzione dall'errore**: l'utente non ha modo di incorrere in errori navigando l'interfaccia;
- **Riconoscimento più che ricordo**: i layout sono semplici, l'utente non deve adattarsi per fruire dell applicazione;
- **Flessibilità ed efficienza**: la fruizione dell' applicazione è molto semplice, non servono scorciatoie;
- **Estetica e progettazione minimalista**: design minimale e chiaro;
- **Facilità di riconoscimento, diagnosi e risoluzione dalle situazioni di errore**: qualora ci fossero errori o mal funzionamenti questi vengono notificati e spiegati all utente. Insieme alla notifica c'è anche un suggerimento per risolvere il problema;

6.2 Usability Test

La correttezza dell'utilizzo del sistema è stata testata dallo sviluppatore e dagli utenti coinvolti nella fase di testing precedentemente descritta.

6.3 Usability Test

La bontà dell'esperienza utente è stata valutata tramite la somministrazione di un **UEQ** reperito su ueq-online.org ai cinque utenti reali coinvolti. I risultati sono i seguenti:

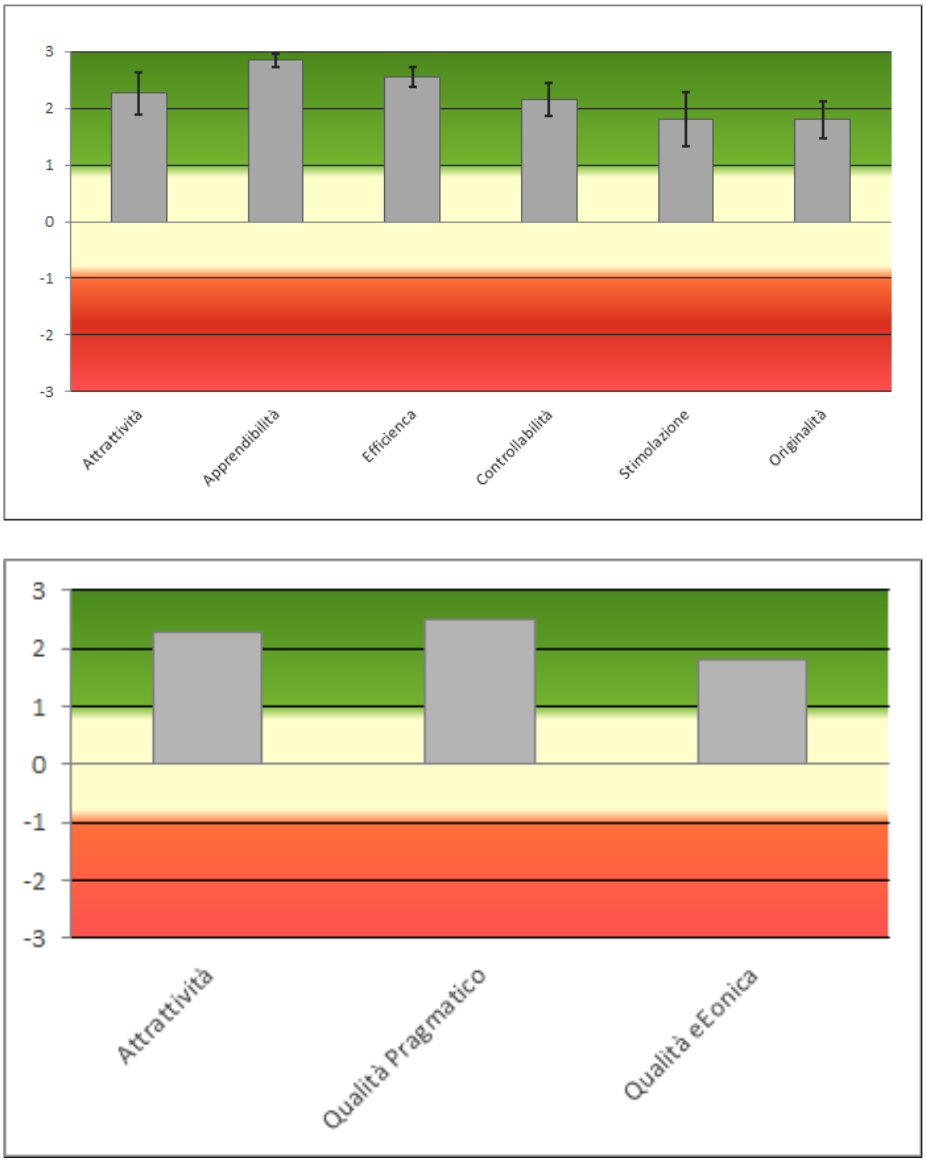


Figure 4: risultati UEQ

7 Deployment

L'applicazione è distribuita tramite l'utilizzo di **Docker**. Sono stati realizzati due container:

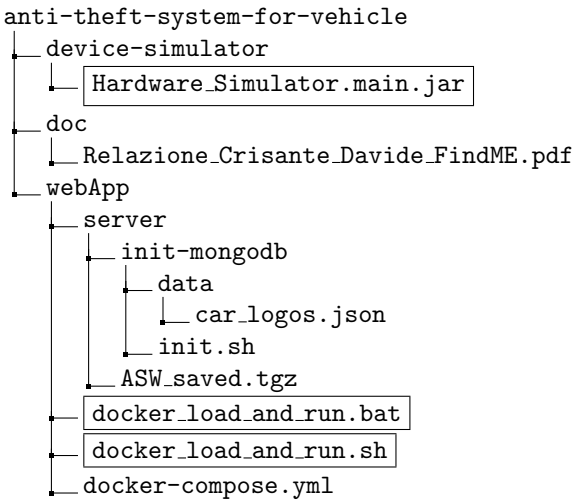
- `node`: backend NodeJs contenente anche la build del frontend da inviare a fronte di determinate richieste;
- `mongo`: database MongoDB. La collection `car_logos` che viene caricata in automatico al primo avvio del DB.

Tutto il materiale utilizzato per la creazione dei container è presente nel branch `main` di questa directory Bitbucket.

Setup

Eseguire il comando:
`git clone -b deploy https://davprs@bitbucket.org/davprs/anti-theft-system-for-vehicle.git`

La struttura della directory è la seguente:



I file evidenziati sono necessari per il setup e la fruizione del servizio.

In `device-simulator` è presente il simulatore di dispositivo in formato *jar*.
In `webApp` va eseguito lo script `docker_load_and_run.bat` (con Windows PowerShell) o `docker_load_and_run.sh` a seconda del sistema operativo (rispettivamente Windows o Linux/MacOS). Questi script:

- decomprimono l'archivio `ASW_saved.tgz` contenente i salvataggi dei due container in formato `.tar`;
- salvano i container come immagini Docker;

- eseguono i container.

Al termine dell esecuzione degli script basta aprire il browser e andare su `http://localhost:5000` per visualizzare la pagina web.
I container espongono le seguenti porte:

- **5000**: richieste gestite da NodeJs ed Express;
- **4000**: richieste gestite da Socket.io;
- **27017**: MongoDB.

8 Conclusioni

Il sistema è semplice da utilizzare e svolge tutti i compiti richiesti e specificati dalle personas.
Per mancanza di tempo alcune funzionalità aggiuntive sono state eliminate in quanto non essenziali.
Questo progetto si presta a molteplici estensioni e migliorie, quali ad esempio:

- supporto all'utilizzo di multipli dispositivi per account;
- aggiunta di ruoli diversi da quello di `user` (es: amministratore di una flotta di veicoli);
- visualizzatore storico furti con i relativi percorsi.

Commenti

Questo progetto mi ha permesso di approfondire la conoscenza delle tecnologie e metodologie per lo sviluppo di applicazioni web moderne. Nello specifico mi è stato possibile migliorare le mie capacità e conoscenze riguardanti Docker, SCSS, React e JavaScript in generale.