



**UNIVERSITÀ
DI TRENTO**



Ardeeno WebApp

T41-SE22

D4-Documento di Sviluppo

v1.0

alessandro.manfucci@studenti.unitn.it enrico.cescato@studenti.unitn.it
m.sottocornola-1@studenti.unitn.it

29-12-2022

Indice

1	BackEnd	2
1.1	Struttura	2
1.2	Dependencies	3
1.3	Database	3
1.4	APIs	5
1.4.1	Resources Extraction from the Class Diagram	5
1.4.2	Resources Models	7
1.4.3	Sviluppo	7
1.4.4	Documentazione OpenAPI 3.0	7
1.4.5	Testing Jest	8
1.4.6	.env	9
2	FrontEnd	9
2.1	Struttura	10
2.2	Dependencies	11
2.3	User Flows	11
2.4	Sviluppo	12
2.4.1	ActionModal.js	13
2.4.2	Home.js	13
2.4.3	Login.js	13
2.4.4	MyAccount.js	13
2.4.5	MyImpianti.js	13
2.4.6	Dashboard.js	13
2.4.7	Heatmap.js	13
2.5	Schermate	14
3	GitHub Repository	16
4	Deployment	17
4.1	FrontEnd	17
4.2	BackEnd	17

Abstract

In questo primo sprint di sviluppo (D4v1.0), ci si limita ai Requisiti Funzionali:

- [RF1 Visualizzazione presentazione](#)
- [RF4 Registrazione](#) (la sola parte BackEnd)
- [RF5 Autenticazione](#)
- [RF5.1 Logout](#)
- [RF8 Visualizzazione dati personali](#)
- [RF11 Visualizzazione impianti acquistati](#)
- [RF11.1 Visualizzazione singolo impianto \(Dashboard\)](#)
- [RF12 Visualizzazione misurazioni su heatmap](#)

Dato che **non** si implementa il [RF6 Conferma indirizzo email](#) l'applicazione prototipo considera sempre confermato l'indirizzo email di ogni account.

Si veda il Documento di Specifica per i casi d'uso associati a questi RF – che verranno quindi implementati.

Per i Requisiti Non Funzionali ci si limita ad:

- [RNF2 Sicurezza](#) la sola proprietà 'Password salvata non in chiaro'
- [RNF3 Portabilità](#) le sole coppie (S.O., Browser): (Ubuntu 20.04, Firefox 108), (Android 9, Firefox 108)

Il progetto è stato suddiviso in due repositories: `ardeeno-frontend`, contenente il progetto React, ed `ardeeno-backend`, il quale si interfaccia con `cloud.mongodb.com` e fornisce le API al FrontEnd.

1 BackEnd

Il BackEnd è stato realizzato con il runtime environment `Node.js` sul repository [t41-se22/ardeeno-backend](#)

1.1 Struttura

```
t41-se22/ardeeno-backend/  
|---controllers/  
|---coverage/  
|---middleware/  
|---models/  
|---node_modules/  
|---routes/  
|---schemas/  
|---tests/  
|---utils/  
|---.env  
|---.gitignore  
|---app.js  
|---package.json  
|---Procfile  
|---README.md  
|---server.js  
|---swagger3.json
```

La struttura del BackEnd è stata suddivisa in più cartelle, in particolare:

- **controllers/**: contiene le funzioni che costituiscono le vere e proprie API
- **coverage/**: contiene le informazioni sulla copertura del codice garantita dai test **Jest**
- **middleware/**: contiene quelle funzioni comuni a tutte le API – autorizzazione e controllo del token – come dal paradigma di sviluppo di **Node**

- **models/**: contiene i **Model** degli **Schemas** utilizzati sul database **mongodb**
- **routes/**: contiene gli end-point delle API
- **schemas/**: contiene gli **Schemas** utilizzati sul database **mongodb**
- **tests/**: contiene i file di test **Jest** ed il file di configurazione del mock-database
- **app.js**: contiene la configurazione degli endpoint tramite **express** e la configurazione della connessione a **mongodb** tramite **mongoose**
- **Procfile**: contiene la configurazione di **Heroku**
- **server.js**: contiene la configurazione del server di rete
- **swagger3.json**: contiene la documentazione Open API 3.0 in formato **json**

1.2 Dependencies

```
"dependencies": {
  "bcrypt": "^5.1.0",
  "cors": "^2.8.5",
  "dotenv": "^16.0.3",
  "express": "^4.18.2",
  "http-status-codes": "^2.2.0",
  "jsonwebtoken": "^8.5.1",
  "mongodb-memory-server": "^8.10.2",
  "mongoose": "^6.8.0",
  "swagger-ui-express": "^4.6.0"
},
"devDependencies": {
  "jest": "^29.3.1",
  "supertest": "^6.3.3"
}
```

Si descrivono nello specifico le librerie più rilevanti:

- **bcrypt**: libreria che fornisce funzioni di salted-hashing con multipli hashing-rounds, in maniera stateless
- **cors**: libreria per gestire il Cross-Origin Resource Sharing – necessaria per adempiere al protocollo HTTP
- **dotenv**: libreria che carica come variabili globali le costanti in **.env**
- **http-status-codes**: libreria che contiene come costanti i codici di stato http più rilevanti
- **jsonwebtoken**: libreria che fornisce funzioni di generazione di token json come dall'[RFC 7519](#)
- **mongodb-memory-server**: libreria che permette la creazione di un database **mongodb** in memoria RAM – utile per creare mock-database da utilizzare nella fase di testing
- **swagger-ui-express**: libreria che genera a partire da un file **.json/.yaml** pagine **html** per la documentazione delle API
- **jest**: libreria che effettua il testing dell'applicazione – si utilizza senza **babel**, dunque per default non supporta i moduli js ES6
- **supertest**: libreria che mette a disposizione utili convenience-methods per inoltrare richieste http e fare assertions sulle risposte http; utilizzata per effettuare il testing dell'applicazione

1.3 Database

Il BackEnd si interfaccia con un database **mongodb** in hosting su [MongoDB Atlas](#). MongoDB è un database document-based, non relazionale. Si utilizza **mongoose** per interfacciarsi con il database in maniera più strutturata, definendo degli **Schemas**. Il database **ardeeno-db** è suddiviso in più collections, per ognuna delle quali si definisce uno Schema **mongoose**:

- **Utenti**

```
{
  email: {type:String, required:true, unique:true},
  password: {type:String, required:true},
}
```

```

indirizzo: {type:String, required:true},
nome: {type:String, required:true},
cognome: {type:String, required:true},
telefono: {type:String, required:true, unique:true},
ruolo: {type:String,
  enum:['cliente', 'tecnico',
    'supervisore', 'amministratore'],
  default:'cliente'},
isEmailConfermata: {type:Boolean, default:true},
impiantiAcquistati: {type:[{
  type: mongoose.Schema.Types.ObjectId,
  ref: 'Impianto'}], default:[]},
cf: {type:String, unique:true, sparse:true},
isDimesso: {type:Boolean, default:false}
}

```

- Modelli

```

{
  nome: {type:String, required:true, unique:true},
  tipo: {type:String, required:true},
  immagine: {type:String, required:true},
  costo: {type:Map, required:true},//in euro, senza iva
  numSensori: {type:Number, required:true},
  superficie: {type:Number, required:true},//consigliata in km^2
  pi: {type:Number, required:true},
  parametri: {type:[{type:Map}], required:true},
}

```

- Impianti

```

{
  modello: {type: mongoose.Schema.Types.ObjectId, ref:'Modello', required:true},
  indirizzo: {type:String, required:true},
  lat: {type:Number, required:true},
  long: {type:Number, required:true},
  fattura: {type:String, required:true},
  superficie: {type:Number, required:true},//effettiva in km^2
  dataAcquisto: {type:Date, required:true},
  dataDismissione: {type:Date},
  isDimesso: {type:Boolean, default:false},
  sensori: {type:[{type: mongoose.Schema.Types.ObjectId, ref:'Sensore'}], default:[]}
}

```

- Sensori

```

{
  impianto: {type: mongoose.Schema.Types.ObjectId, ref:'Impianto', required:true},
  lat: {type:Number, required:true},
  long: {type:Number, required:true},
  dataDismissione: {type:Date},
  isDimesso: {type:Boolean, default:false}
}

```

- Snapshots_<impianto._id>

In questo caso, per ogni Impianto si crea una nuova collections con lo stesso Schema mongoose; questo

perché le query sugli Snapshots sono sempre – nel nostro contesto – su un solo impianto. Inoltre, si stima dai RNF un utilizzo con numerosi impianti, che producono una notevole quantità di snapshot – questa struttura aiuta a mantenere efficienza. Le collections create sono di tipo **capped**, ovvero hanno un massimo numero di documenti; questo, come dalla documentazione ([link](#)), mantiene l'ordine di inserimento dei documenti – che avviene nel nostro contesto in maniera ordinata, e dunque non si necessita di un riordinamento al momento della query.

```
{
  impianto: {type: mongoose.Schema.Types.ObjectId,
    ref: 'Impianto', required: true},
  date: {type: Date, required: true, unique: true}
}
```

- **Misurazioni_<impianto._id>**

Come nel caso precedente, anche per le Misurazioni si crea una nuova collection per ogni Impianto con lo stesso Schema mongoose, di tipo **capped**.

```
{
  sensore: {type: mongoose.Schema.Types.ObjectId,
    ref: 'Sensore', required: true},
  date: {type: Date, required: true},
  valori: {type: Map, required: true}
}
```

1.4 APIs

1.4.1 Resources Extraction from the Class Diagram

Si omettono quelle risorse non necessarie per implementare i RF scelti per questo sprint (D4v1).

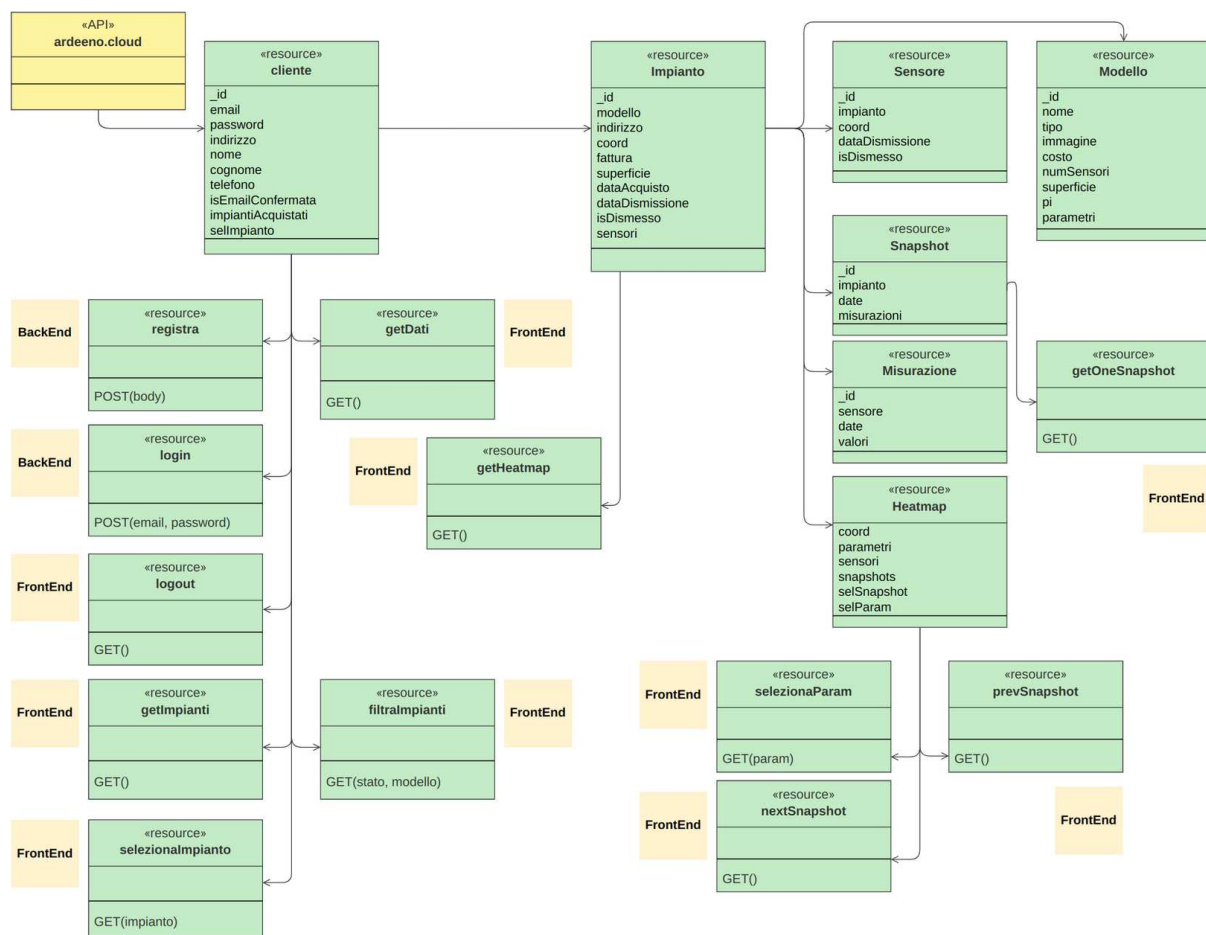


Figura 1: Resources Extraction

1.4.2 Resources Models

Non si indicano gli header in richieste/risposte; il token di accesso è inviato come parametro header `x-access-token` ed è necessario in `getDati`, `getImpianti`, `getHeatmap`, `getOneSnapshot`. Ogni risposta comprende l'header `x-token-status`, con valori `['empty', 'valid', 'expired']`.

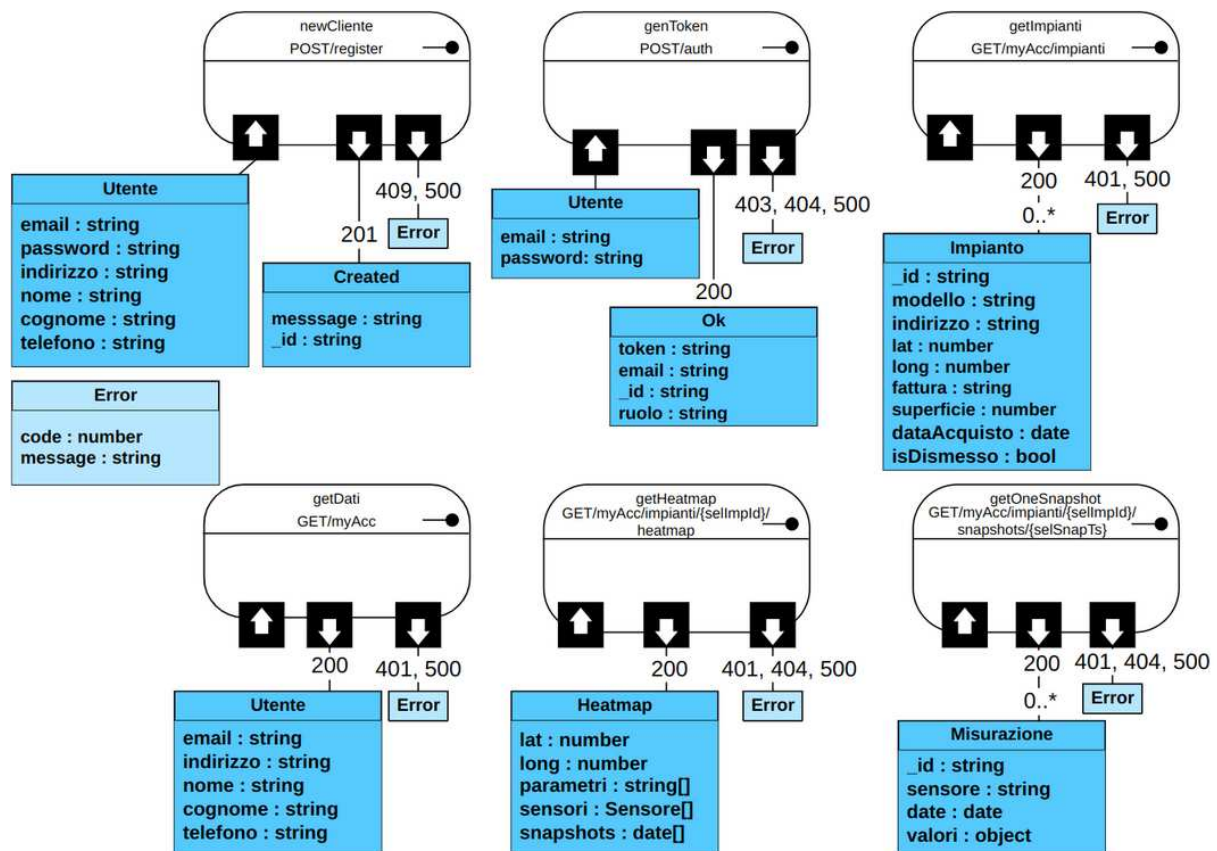


Figura 2: Resources Models

1.4.3 Sviluppo

Si è utilizzato uno stile di programmazione con `async/await` e gestione degli errori – anche asincroni – con `try/catch`.

1.4.4 Documentazione OpenAPI 3.0

Le API sono state documentate seguendo lo standard Open API 3.0 e sono visibili ad api.ardeeno.cloud/api-docs. Si documentano tutti i possibili messaggi di errore nella sezione `examples`. Si danno inoltre `examples` per i messaggi di richiesta della funzione `genToken` in `POST /auth`; si descrivono poi gli Schemas `Mongoose`. Sono presenti quattro configurazioni Server:

- `http://localhost:8080` da utilizzare se si esegue il Backend in locale
- `http://api.ardeeno.cloud` da utilizzare se connessi al Backend deployed su Heroku.
- `http://api.ardeeno.best` da utilizzare se connessi al Backend deployed su Heroku – ridondante.
- `http://ardeeno-backend.herokuapp.com` da utilizzare se connessi al Backend deployed su Heroku – ridondante.

L'applicazione prototipo dispone di un Utente pre-registrato, a cui sono associati due Impianti:

- **Panarotta SP11:** Impianto con sensori, snapshots e misurazioni (auto generati con [t41-se22/ardeeno-datalayer](https://t41-se22.ardeeno-datalayer))
- **Val Borzago:** Impianto **senza** sensori, snapshots o misurazioni

Le credenziali dell'Utente sono:

email: mario.rossi@gmail.com

password: password

Le credenziali sono appunto visibili sugli `examples` di `genToken` (api.ardeeno.cloud).

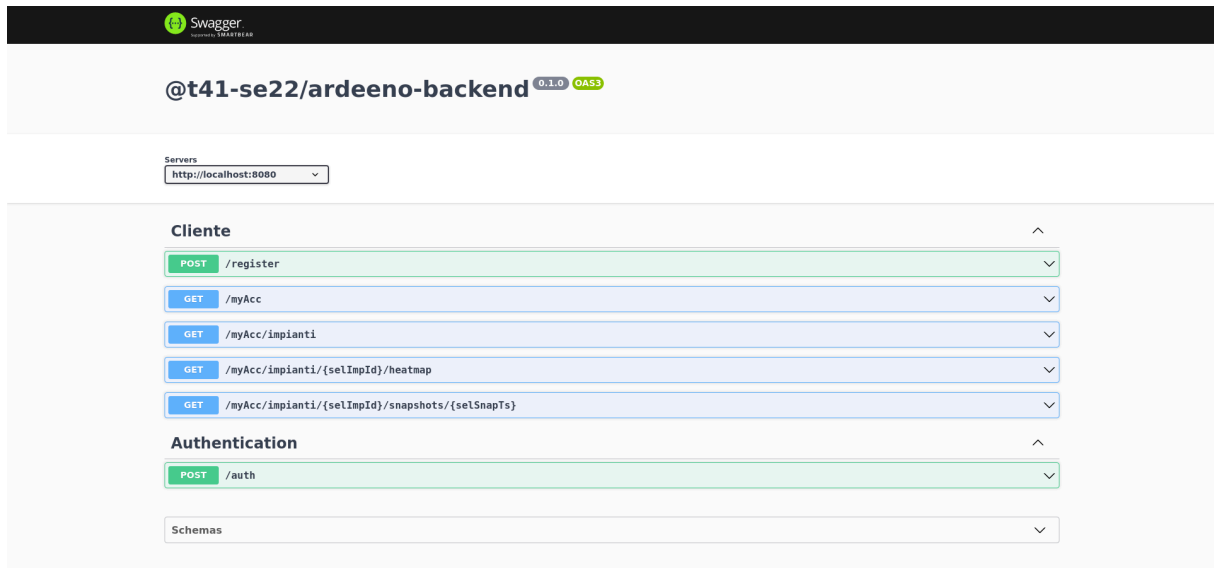


Figura 3: Anteprima Documentazione

1.4.5 Testing Jest

Il Testing è stato gestito con la libreria `Jest` ed `supertest`. Inoltre, si è utilizzato `mongodb-memory-server` per creare – al momento del testing – un mock-database con dati sempre costanti, così da fare assertions direttamente sul body delle risposte.

Per ogni API definita, si sono effettuati tutti i casi di test sul successo (2xx SUCCESS) e sugli errori dell'Utente (4xx CLIENT ERROR) (a meno del token non valido, funzionalità presente in ogni risposta tramite header `x-token-status`) – ma non sui possibili errori interni (5xx SERVER ERROR). Inoltre, si sono testate quelle sole risposte definite esplicitamente nello sviluppo, e non quelle gestite implicitamente da `Node.js`.

Nel particolare:

- **POST /auth**
 - 201 CREATED
 - 403 FORBIDDEN: Wrong Password code 1
 - 404 NOT FOUND: Utente Not Found code 0
- **POST /register**
 - 200 OK
 - 409 CONFLICT: Email already used code 0
 - 409 CONFLICT: Telefono already used code 1
- **GET /myAcc**
 - 200 OK
- **GET /myAcc/impianti**
 - 200 OK
- **GET /myAcc/impianti/:selImpId/heatmap**
 - 200 OK
 - 404 NOT FOUND: No such impianto for this user code 1
- **GET /myAcc/impianti/:selImpId/snapshots/:selSnapTs**
 - 200 OK
 - 404 NOT FOUND: No such impianto for this user code 1
 - 404 NOT FOUND: No such snapshot code 3

```

ardeeno-backend$ npm test

> @t41-se22/ardeeno-backend@0.1.0 test
> jest --coverage

PASS tests/token.test.js
PASS tests/cliente.test.js

Test Suites: 2 passed, 2 total
Tests:       13 passed, 13 total
Snapshots:   0 total
Time:        5.73 s
Ran all test suites.

```

Dal report sul coverage di Jest si nota che si sono coperti il 57.99% dei branch – quelli non coperti sono appunto quei branch che gestiscono gli errori 5xx SERVER ERROR, i quali sono principalmente gli errori del server mongodb. Infatti i test coprono il 89.34% degli Statements, il 100.00% delle funzioni ed il 89.34% delle linee di codice.

All files

89.34% Statements 453/507 57.89% Branches 33/57 100% Functions 19/19 89.34% Lines 453/507

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File		Statements	Branches	Functions	Lines
ardeeno-backend	<div><div></div></div>	86.56%	58/67	66.66%	4/6
ardeeno-backend/controllers	<div><div></div></div>	80.9%	178/220	52.5%	21/40
ardeeno-backend/middleware	<div><div></div></div>	94.23%	49/52	70%	7/10
ardeeno-backend/models	<div><div></div></div>	100%	62/62	100%	0/0
ardeeno-backend/routes	<div><div></div></div>	100%	31/31	100%	0/0
ardeeno-backend/schemas	<div><div></div></div>	100%	21/21	100%	0/0
ardeeno-backend/tests/database	<div><div></div></div>	100%	54/54	100%	1/1

Figura 4: Coverage

1.4.6 .env

Per testare manualmente il codice in locale, si lascia la configurazione .env – che non è presente per motivi di sicurezza su git.

```

DB_HOST="cluster0.uztny9.mongodb.net/Cluster0"
DB_USER=root
DB_PASS=i64uohew4uSH1UWa
PORT=8080
MONGODB_URI="mongodb+srv://root:i64uohew4uSH1UWa@cluster0.uztny9.mongodb.net/ardeeno-db"
SUPER_SECRET="someSecret"

```

2 FrontEnd

Il FrontEnd è stato realizzato con React 18, seguendo i suoi principi (*The React Way*) ed utilizzando la libreria [coreui-free-react-admin-template](#). La WebApp è quindi una Single Page Application.

2.1 Struttura

```
t41-se22/ardeeno-frontend/
|---node_modules/
|---public/
|---src/
|   |---assets/
|   |   |---brand/
|   |   |---fonts/
|   |   |---images/
|   |---components/
|   |   |---AppContent.js
|   |   ...
|   |---scss/
|   |---views/
|   |   |---appContent/
|   |   |---pages/
|   |   |---AppLayout.js
|   |---_nav.js
|   |---App.js
|   |---index.js
|   |---store.js
|---.env
|---.eslintrc.json
|---.gitignore
|---.jsconfig.json
|---LICENSE-COREUI-FREE
|---package.json
|---Procfile
|---README.md
```

La struttura del FrontEnd è suddivisa in più cartelle, in particolare:

- **public/**: contiene il file `index.html` scaricato inizialmente dal browser, che si occupa di configurare il runtime-environment `React`
- **src/**: contiene i file `React`
- **src/components/**: contiene le componenti `React` dell'`AppLayout`
- **src/components/AppContent.js**: contiene la componente `React AppContent`, che effettua il routing tra i contenuti dell'applicazione
- **src/views/**: contiene le viste `React` – dove una vista è un contenuto dell'applicazione o una pagina a sé stante dall'`AppLayout`
- **src/views/AppLayout.js**: contiene la componente `React AppLayout`, che crea il layout istanziando `Sidebar`, `Header`, `Footer` ed `AppContent`
- **src/views/appContent/**: contiene le viste caricate da `AppContent` – ovvero l'applicazione vera e propria
- **src/views/pages/**: contiene le pagine a sé stanti dall'`AppLayout`, che sono *esterne* all'applicazione (`Login`, `Page401`, `Page404`, ...)
- **src/_nav.js**: file javascript contenente costanti per la configurazione dinamica della `Sidebar` (a seconda del tipo di `Utente`)
- **src/App.js**: contiene la componente `React App`, che esporta la variabile di ambiente `API_URL`, fornisce il contesto `loggedUser`, `selImp` (a partire dai *persistent data* di `localforage`) ed effettua il routing verso l'`AppLayout` e verso le `pages` esterne ad `AppLayout`
- **src/index.js**: file javascript che configura il contesto `redux` ed istanzia la componente `React` radice `App`
- **.eslintrc.json**: contiene le impostazioni per il linting dei file `React` ed `JSX`
- **Procfile**: contiene la configurazione di `Heroku`

2.2 Dependencies

```
"dependencies": {
  "@coreui/coreui": "^4.2.1",
  "@coreui/icons": "^2.1.0",
  "@coreui/icons-react": "^2.1.0",
  "@coreui/react": "^4.3.1",
  "@coreui/utils": "^1.3.1",
  "axios": "^1.2.1",
  "core-js": "^3.24.1",
  "deck.gl": "^8.8.20",
  "localforage": "^1.10.0",
  "prop-types": "^15.8.1",
  "react": "^18.2.0",
  "react-app-polyfill": "^3.0.0",
  "react-dom": "^18.2.0",
  "react-leaflet": "^4.2.0",
  "react-map-gl": "^5.3.0",
  "react-redux": "^8.0.2",
  "react-router-dom": "^6.3.0",
  "react-scripts": "5.0.1",
  "redux": "4.2.0",
  "sass": "^1.54.4",
  "simplebar-react": "^2.4.1"
},
"devDependencies": {
  "eslint": "^8.30.0",
  "eslint-plugin-react": "^7.31.11",
  "eslint-plugin-react-hooks": "^4.6.0"
}
```

Oltre a **coreui** ed alle sue dependencies (**@coreui/...**, **react-redux**, **redux**, **sass**, **simplebar-react**) si sono utilizzate ulteriori librerie, che si descrivono nello specifico:

- **axios**: libreria che mette a disposizione convenience-methods per effettuare richieste http AJAX
- **deck.gl**: libreria che aggrega e renderizza Big Data su mappe geografiche, in maniera bidimensionale ma anche tridimensionale
- **localforage**: libreria che, appoggiandosi su **localstorage**, mette a disposizione funzioni per il salvataggio di dati sul browser; è stata utilizzata per mantenere *persistent data* dell'Utente Autenticato
- **prop-types**: libreria che permette di documentare le **props** delle componenti React
- **react-leaflet**: libreria che permette la renderizzazione di mappe OpenStreetMap
- **react-map-gl**: libreria che permette la renderizzazione di mappe – necessaria la versione (deprecata) ^5 per **deck.gl**

2.3 User Flows



Figura 5: legenda

Si descrive un'user flow per il Cliente, che comprende: login, logout, visualizzazione degli impianti acquistati, selezione di un Impianto e visualizzazione delle misurazioni su Heatmap.

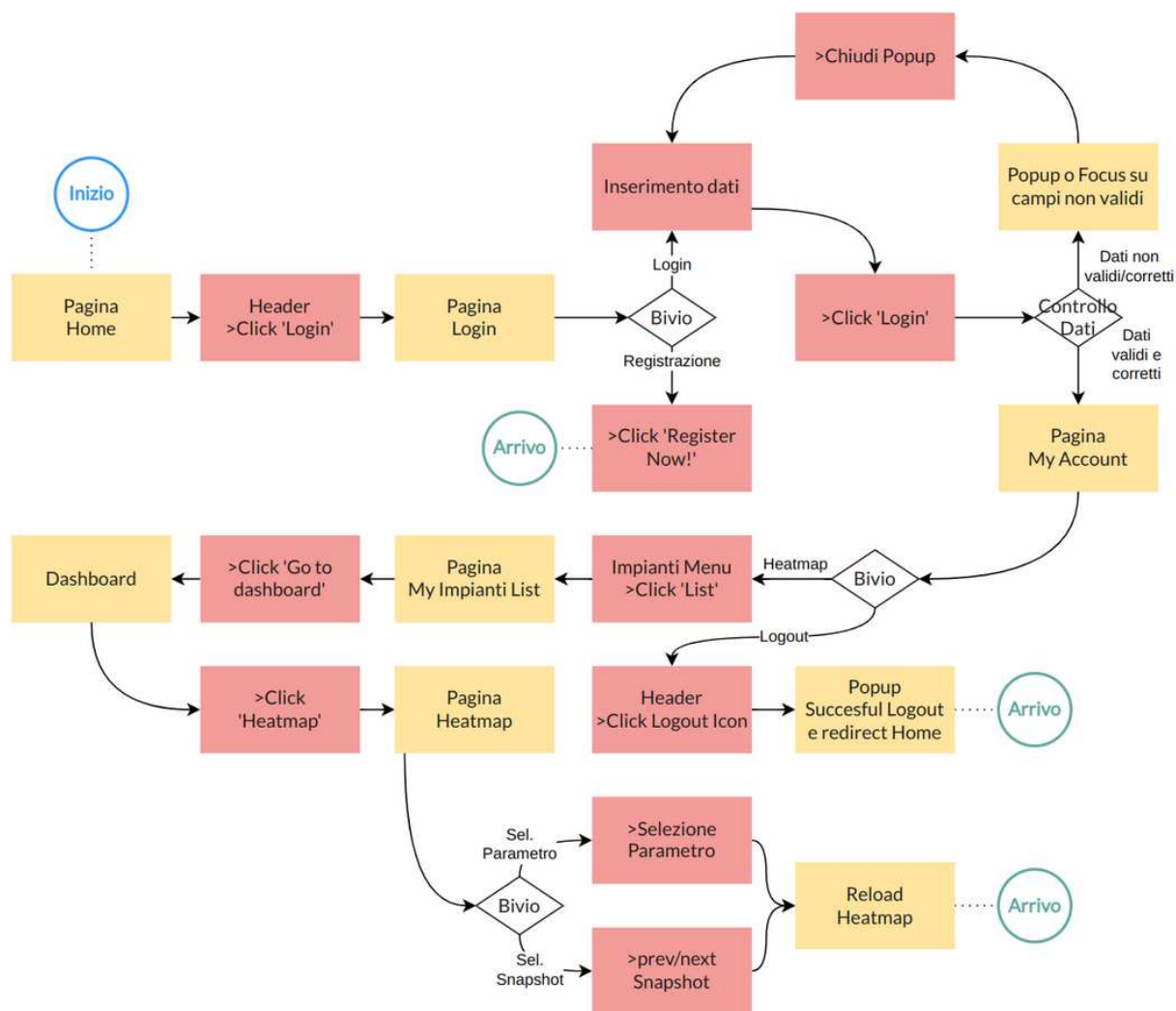


Figura 6: User Flow Cliente

2.4 Sviluppo

Per quanto riguarda lo sviluppo, si è utilizzato uno stile **React** con *function components* ed *stateful function components (hooks)*. Si sono gestiti sia i casi d'usi corretti, che le exceptions (tutti i code di errore descritti dal BackEnd, BackEnd non raggiungibile, ...). Inoltre, si sono ricompilati i fogli di stile **scss** introducendo il Font-Family **Lato** e la palette di colori **ardeeno**, come dai mock-up presentati nel D1-Documento di Progetto. Infine, l'applicazione garantisce compatibilità Desktop e Mobile (**RNF3 Portabilità**), grazie ad un corretto uso delle componenti **Row**, **Col** e differenziando il motore di rendering della **Heatmap.js** a seconda del tipo di dispositivo (Mobile, Desktop); l'applicazione è stata quindi testata su Desktop (Ubuntu 20.04, Firefox 108), Tablet (Android 12, Firefox 108) ed Smartphone (Android 9, Firefox 108).

Il routing ha la seguente struttura ¹:

```

/ ·> /home
/home -> Home.js
/MyAccount -> MyAccount.js
/MyImpianti ·> /MyImpianti/list
/MyImpianti/list -> MyImpianti.js
/MyImpianti/Dashboard -> Dashboard.js

```

¹Con **react-router** 6.3, le routes sono per default final-slash insensitive.

```

/MyImpianti/Heatmap -> Heatmap.js

/login -> Login.js
/401 -> Page401.js
/404 -> Page404.js
/500 -> Page500.js
/* ·> /404

```

Si descrivono ora brevemente le componenti sviluppate ad-hoc per **ardeeno-frontend**.

2.4.1 ActionModal.js

Componente sviluppata per mostrare all'Utente i messaggi di errore.

```

ActionModal.propTypes = {
  title:string,
  body:string,
  onClose:func
}

```

2.4.2 Home.js

Componente sviluppata per implementare [RF1 Visualizzazione Presentazione](#); si utilizza **react-leaflet** per mostrare la mappa con le sedi dell'azienda, connettendosi a un **TileProvider** OSM. Cliccando sul marker si visualizza l'indirizzo della sede.

2.4.3 Login.js

Componente sviluppata per implementare [RF5 Autenticazione](#) in tutte le sue exceptions. I dati dell'Utente Autenticato sono memorizzati in **localStorage** e nel **AppContext** React. La funzione [RF5.1 Logout](#) è accessibile dall'**AppHeader** in qualunque pagina dell'applicazione.

2.4.4 MyAccount.js

Componente sviluppata per implementare [RF8 Visualizzazione dati personali](#).

2.4.5 MyImpianti.js

Componente sviluppata per implementare [RF11 Visualizzazione impianti acquistati](#) – con ordinamento e filtraggio. Al primo accesso, per accedere alla Dashboard ed alla Heatmap è necessario selezionare un Impianto cliccando sul link **Go to Dashboard** – se ciò non è fatto, la Dashboard e la Heatmap fanno un redirect ad **/MyImpianti/list**. I dati dell'Impianto selezionato sono memorizzati in **localStorage** e nel **AppContext** React.

2.4.6 Dashboard.js

Componente sviluppata per implementare [RF11.1 Visualizzazione singolo impianto \(Dashboard\)](#); contiene link ridondanti per le funzioni disponibili (Heatmap).

2.4.7 Heatmap.js

Componente sviluppata per implementare [RF12 Visualizzazione misurazioni su heatmap](#). Si utilizza **deck.gl** per renderizzare la heatmap sopra ad una **react-map-gl**, che reperisce le mappe da un **TileProvider** OSM (**deck.gl** supporta lo stile dichiarativo di React). Come dalle specifiche, si memorizzano le misurazioni del solo snapshot selezionato, dunque con **nextSnapshot()**, **prevSnapshot()** si effettua una richiesta **ajax** al BackEnd per reperire le misurazioni del dato snapshot. Come da specifiche si memorizzano tutti gli snapshots dell'Impianto (il solo **date**). In un futuro *sprint* potrebbe essere utile (per adempiere al [RNF4 Prestazioni](#)) implementare la paginazione degli snapshots – in quel caso si dovrebbe memorizzare una sola pagina di snapshot, e richiedere la pagina successiva/precedente *on-demand*.

2.5 Schermate

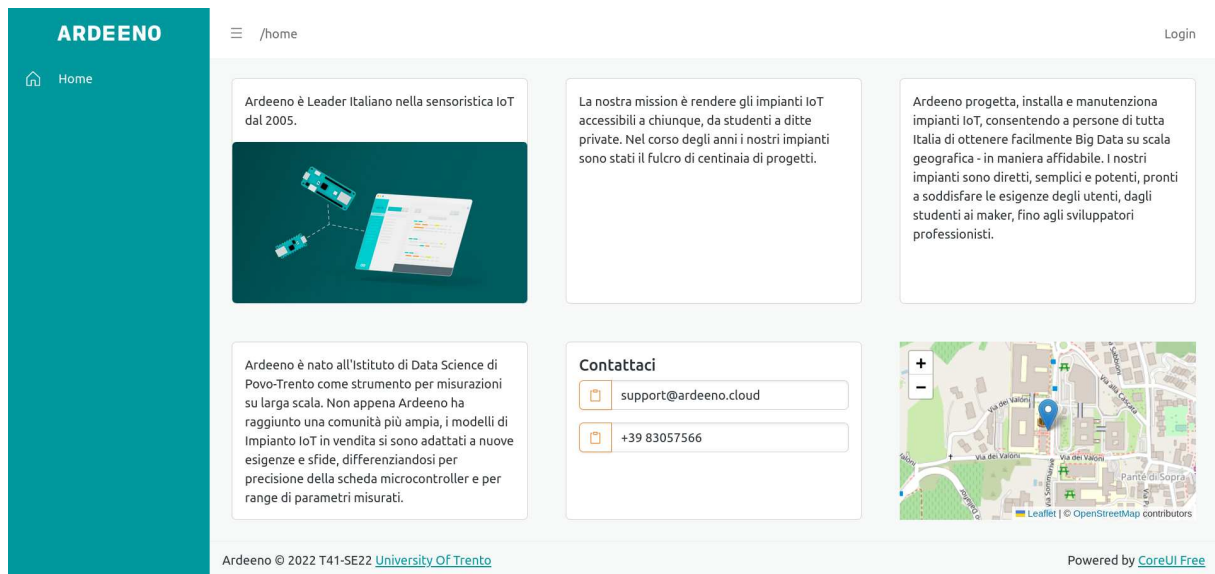


Figura 7: Home

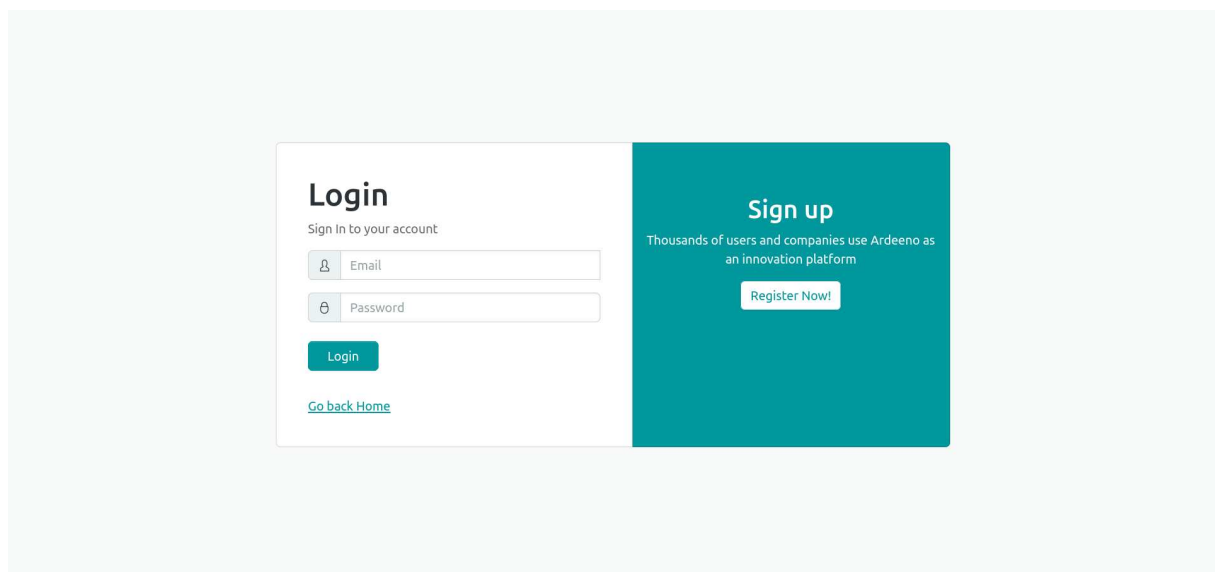


Figura 8: Login

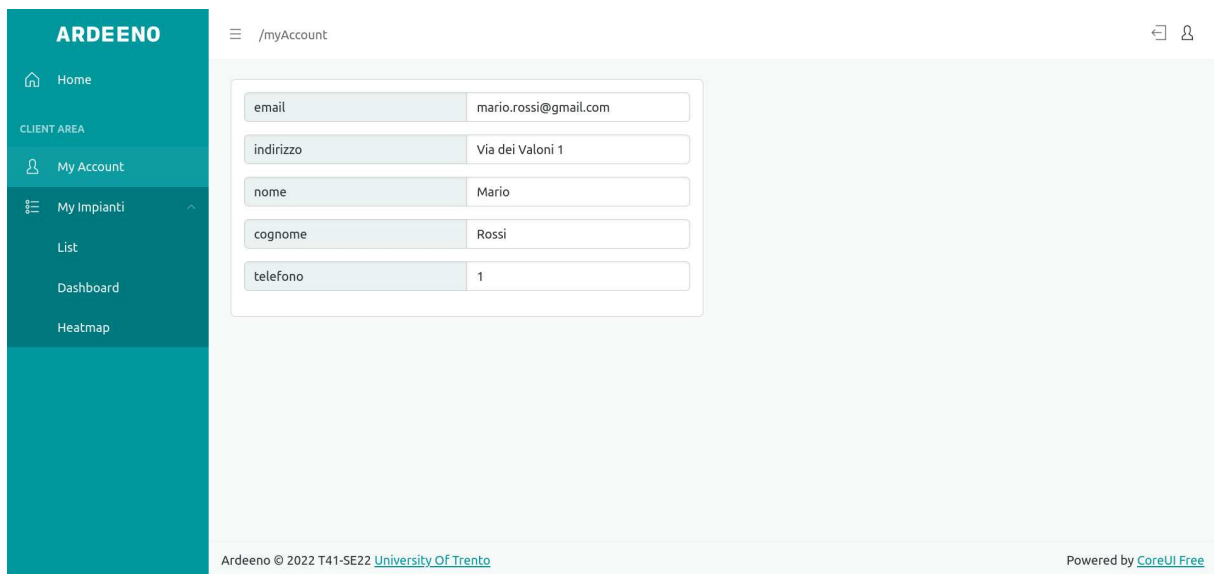


Figura 9: MyAccount

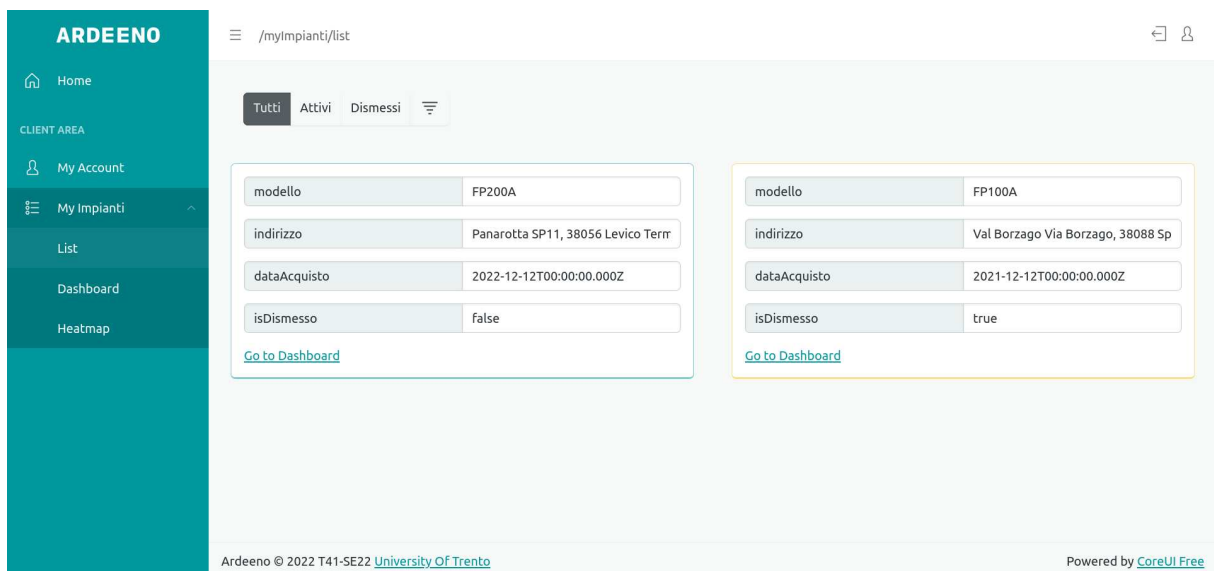


Figura 10: MyImpianti

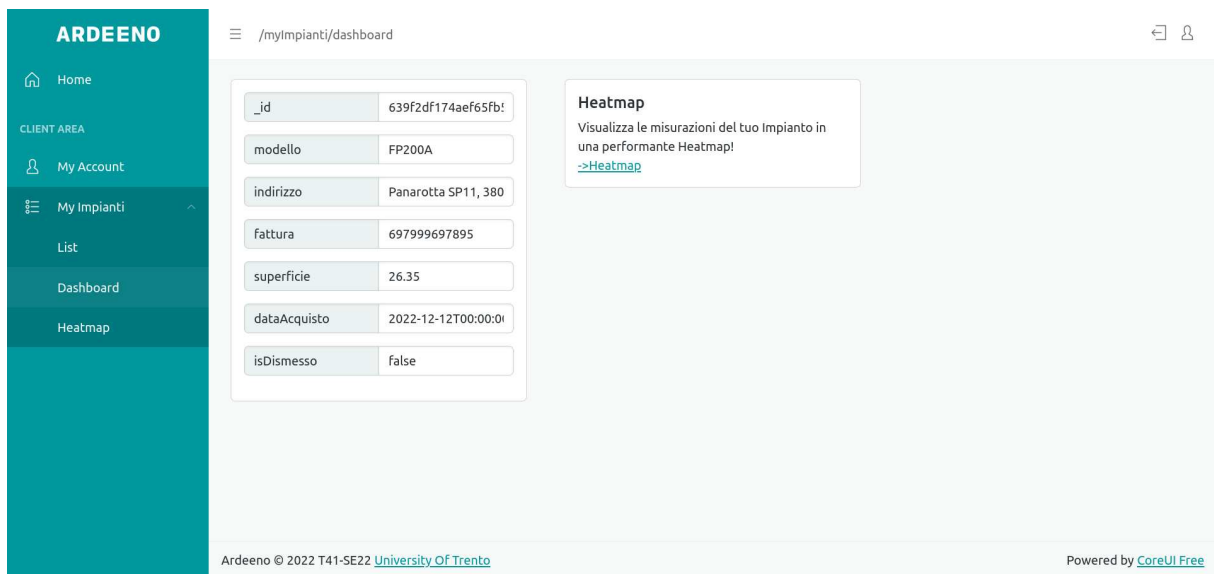


Figura 11: Dashboard

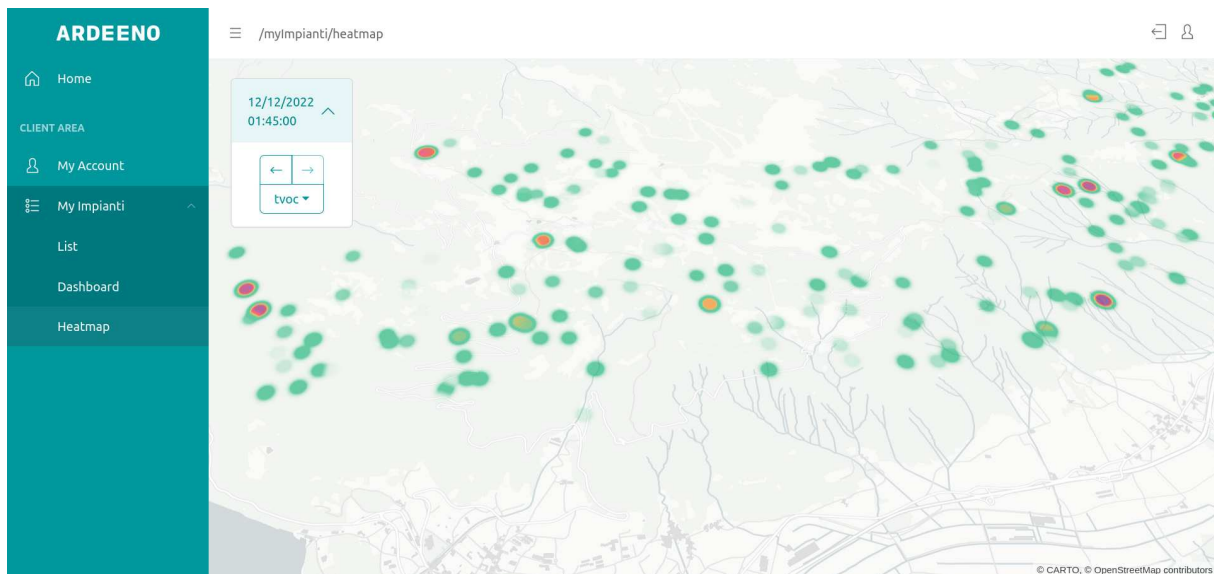
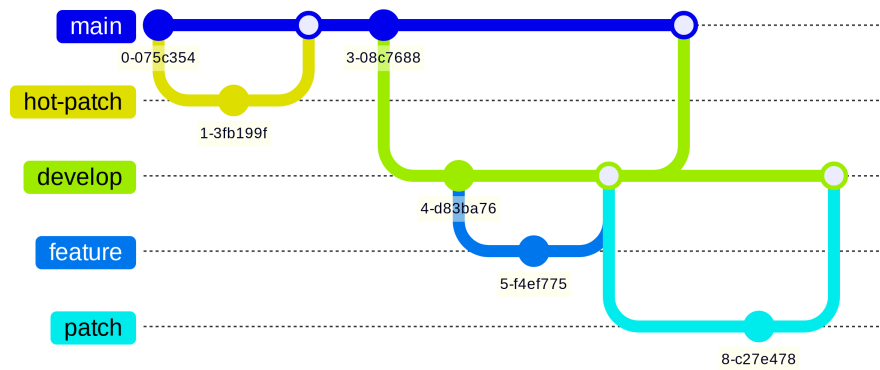


Figura 12: Heatmap

3 GitHub Repository

Per il BackEnd il flusso di lavoro è stato organizzato su più branch:



Inoltre, sempre per il BackEnd, si sono utilizzate delle GitHub Actions per automatizzare il testing Jest su pull request/push per **develop** e **main** (Continuous Integration).

Per il FrontEnd si è usato un solo branch **main**.

4 Deployment

Si è utilizzato Heroku per avere un Fast Deployment agli URL:

4.1 FrontEnd

- www.ardeeno.cloud
- www.ardeeno.best
- ardeeno-frontend.herokuapp.com

4.2 BackEnd

- api.ardeeno.cloud
- api.ardeeno.best
- ardeeno-backend.herokuapp.com

Il Fast Deployment è stato configurato attraverso Heroku e non tramite le GitHub Actions

I custom domain names sono stati impostati tramite la CLI di Heroku.

Per **ardeeno-backend** è stato sufficiente un *EcoDyno*, mentre per **ardeeno-frontend** è stato necessario utilizzare uno *Standard-2X Dyno* con 1GB di memoria RAM.

App connected to GitHub

Code diffs, manual and auto deploys are available for this app.

Connected to [T41-SE22/ardeeno-backend](#) by [almanfu](#) [Disconnect...](#)

Releases in the [activity feed](#) link to GitHub to view commit diffs

Automatically deploys from [main](#)

Automatic deploys

Enables a chosen branch to be automatically deployed to this app.

You can now change your main deploy branch from "master" to "main" for both manual and automatic deploys, please follow the instructions [here](#).

Automatic deploys from [main](#) are enabled

Every push to [main](#) will deploy a new version of this app. **Deploys happen automatically:** be sure that this branch in GitHub is always in a deployable state and any tests have passed before you push. [Learn more](#)

☒ Wait for CI to pass before deploy

Only enable this option if you have a Continuous Integration service configured on your repo.

[Disable Automatic Deploys](#)

Figura 13: CD backend

App connected to GitHub

Code diffs, manual and auto deploys are available for this app.

Connected to [T41-SE22/ardeeno-frontend](#) by [almanfu](#) [Disconnect...](#)

Releases in the [activity feed](#) link to GitHub to view commit diffs

Automatically deploys from [main](#)

Automatic deploys

Enables a chosen branch to be automatically deployed to this app.

You can now change your main deploy branch from "master" to "main" for both manual and automatic deploys, please follow the instructions [here](#).

Automatic deploys from [main](#) are enabled

Every push to [main](#) will deploy a new version of this app. **Deploys happen automatically:** be sure that this branch in GitHub is always in a deployable state and any tests have passed before you push. [Learn more](#)

☐ Wait for CI to pass before deploy

Only enable this option if you have a Continuous Integration service configured on your repo.

[Disable Automatic Deploys](#)

Figura 14: CD frontend

Domains

You can add custom domains to any Heroku app, then visit [Configuring DNS](#) to setup your DNS target.

You have multiple custom domains enabled

[Add domain](#)

Domain Name	DNS Target ?
api.ardeeno.best	skeletal-mole-3b2hqtq...
api.ardeeno.cloud	trapezoidal-oviraptor-db...

Figura 15: DNS backend

Domains

You can add custom domains to any Heroku app, then visit [Configuring DNS](#) to setup your DNS target.

You have multiple custom domains enabled

[Add domain](#)

Domain Name	DNS Target ?
www.ardeeno.best	cellular-gibbon-19ahs8zcb...
www.ardeeno.cloud	floral-krill-uawlc6k0chw0x...

Figura 16: DNS frontend