DOCUMENTO ESTERNO

Carbon7team

carbon7team@gmail.com

08 Maggio 2022

Organizzazione github: Carbon7team

Manuale dello Sviluppatore

v0.0.3

Redattori	**Revisori**
Andrea Polato Filippo Brugnolaro	

Sommario

Documento gestionale Esterno relativo al Manuale dello Sviluppatore$_G$ del Carbon7team

Storico modifiche al documento

Legenda:

- +: Prima redazione di contenuto
- #: Estensione di contenuto
- [n]: Sezione n del documento

Versione	Operazione	Autore	Verficatore	Data
0.0.3	# Tecnologie adottate [2] # Installazione di Remote Support [4.3] + Remote Support [5.1] + Model [5.1.1] + View [5.1.2]	Filippo Brugnolaro		2022/05/09
0.0.2	+ Introduzione [1] + Tecnologie adottate [2] + Configurazione [3] + Installazione [4]	Andrea Polato	Filippo Brugnolaro	2022/04/04
0.0.1	Generazione Documento	Andrea Polato	-	24/03/2022

Indice

1 Introduzione

1.1 Scopo del documento

Il documento descrive le scelte architetturali prese da Carbon7team nella realizzazione del prodotto in questione. Il fine del documento è quello di fornire una panoramica dettagliata delle tecnologie utilizzate e delle scelte implementative, per garantire la manutenibilità agli sviluppatori che in futuro prenderanno in carico questo prodotto.

1.2 Scopo del prodotto

Il capitolato C6 prevede lo sviluppo di un sistema di assistenza, che garantisca supporto tecnico all'utente utilizzatore di dispositivi UPS. Tale sistema prevede sia la creazione di un applicativo, per monitorare lo stato del dispositivo UPS, sia di una piattaforma di sostegno che aiuti il tecnico nello svolgimento della sua mansione di assistenza.

1.3 Glossario

Per assicurare la massima trasparenza e fruibilità del documento, il *Carbon7team* ha deciso di stilare il *Glossario*. Qui verranno inserti tutti i termini ambigui o relativi all'attività del progetto, che il gruppo individua come degni di nota. I termini qui presenti saranno identificati attraverso una 'G' a pedice.

1.4 Riferimenti

1.4.1 Riferimenti normativi

- *NormeDiProgetto_2.0.0*$_G$
- Regolamento del progetto didattico - dispense
- Capitolato C6: Smart4Energy

1.4.2 Riferimenti informativi

- Design pattern architetturali - dispense
- Design pattern creazionali - dispense
- Design pattern strutturali - dispense
- Design pattern comportamentali - dispense
- Pattern MVC e derivati - dispense
- SOLID programming - dispense

2 Tecnologie adottate

Tecnologia - Versione	**Descrizione**
Linguaggi	
Kotlin - x.x.x	Linguaggio di programmazione utilizzato per realizzare l'applicazione mobile Virtual Display
JavaScript - x.x.x	linguaggio di programmazione orientato agli eventi utilizzato nella realizzazione del nodo server e di Remote Support.
Framework	
React - 17.0.2	Libreria utilizzata per la realizzazione dell'interfaccia utente di Remote Support.
PeerJS - 1.3.2	
Socket.io-Client - 4.4.1	
Express - x.x.x	
Junit - x.x.x	Framework di testing utilizzato nello sviluppo di Virtual Display.
Jest - 27.5.1	Framework di testing utilizzato nello sviluppo di Remote Support.
Strumenti	
NPM - 7.24.2	Strumento di gestione dei pacchetti JavaScript utilizzati.
React-Select - 5.3.0	
Node.js - x.x.x	Runtime che permette di eseguire moduli JavaScript.
Heroku - x.x.x	
Gradle - x.x.x	Strumento per l'automazione dello sviluppo di Virtual Display.
PostrgreSQL - x.x.x	Database utilizzato per immagazzinare i dati degli utenti.

3 Configurazione

Di seguito vengono riportati i requisiti hardware e software necessari all'utilizzo degli strumenti sfruttati nella realizzazione del prodotto finale.

3.1 Requisiti hardware - Virtual Display

Si necessità di un PC con:

- Windows 8 (o successivo) 64-bit;
- 8 GB RAM;
- 8 GB minimo di spazio su disco (IDE + Android SDK + Android Emulator);
- CPU con supporto alla virtualizzazione;
- schermo con risoluzione minima di 1280x800.

3.2 Requisiti software - Virtual Display

- Android Studio Bumblebee 2021.1.1 - download;
- Simulatore Modbus - rilasciato da SOCOMEC®, link per il download non disponibile.

3.3 Requisiti hardware - Remote Support

-

3.4 Requisiti software - Remote Support

- Node.js - download;
- NPM - installazione tramite terminale.

3.5 Altri requisiti

- Una porta di rete libera per l'esecuzione del simulatore (predefinita: 8888);
- Un server raggiungibile da entrambi il Virtual Display e il Remote Support per instaurare la comunicazione tra le parti e recuperare i dati di login.

4 Installazione

Il prodotto finale è diviso in 3 repository differenti, in modo da fornire solo la parte interessata nel caso in cui si disponesse già delle altre. Di seguito sono riportati i link ad esse:

- Virtual Display;
- Remote Support;
- Nodo server.

4.1 Installazione di Virtual Display

- Clonare la relativa repository;
- Eseguire l'emulatore del modbus:
  - Dalla directory del simulatore, entrare nella cartella `bin` e avviare un terminale;
  - Digitare il comando `.\ModbusSlave.exe -d . --dump`;
  - Se si desidera avere informazioni sui comandi disponili, digitare: `.\ModbusSlave.exe --help`;
- Aprire Android Studio e aprire il progetto contenuto nella cartella di clonazione;
- Eseguire l'emulazione dell'applicazione;
- In alternativa, da Android Studio, generare il pacchetto .apk e installarlo su un dispositivo Android fisico o emulato;

4.2 Installazione del server

- Clonare la relativa repository;
- Aprire il terminale e posizionarsi nella cartella contente index.js;
- Eseguire il comando `node index.js`.

4.3 Installazione di Remote Support

- Clonare la relativa repository;
- Posizionarsi all'interno della cartella principale;
- Installare i moduli e le dipendenze necessari tramite il comando `npm install`;
- Posizionarsi all'interno della cartella `src`, contenente il file `index.js`;
- Eseguire il comando `npm start`.

5 Progettazione architetturale

5.1 Remote Support

Per quanto riguarda la parte del Remote Support, a livello prettamente architetturale, è stato utilizzato un approccio **Model-View**.
Il motivo sta nel fatto che, nella libreria di React, le componenti funzionali sono delle funzioni stateless dove si va a inserire tramite delle feature tipiche di React lo stato e le funzionalità della componente.
Inoltre è stata utilizzata una libreria esterna, *MobX*, che ha svolto il compito di implementare il **design pattern observer**, utile sia per la visualizzazione dei dati in real time ricevuti tramite la connessione peer-to-peer che per la condivisione di variabili di stato che legano componenti che non hanno nessuna correlazione tra loro.
Diretta conseguenza dell'utilizzo di questa libreria è l'attuazione del **RootStore pattern**, contenente i 2 contenitori rispettivamente di *DatasetStore* e *StateUIStore*.

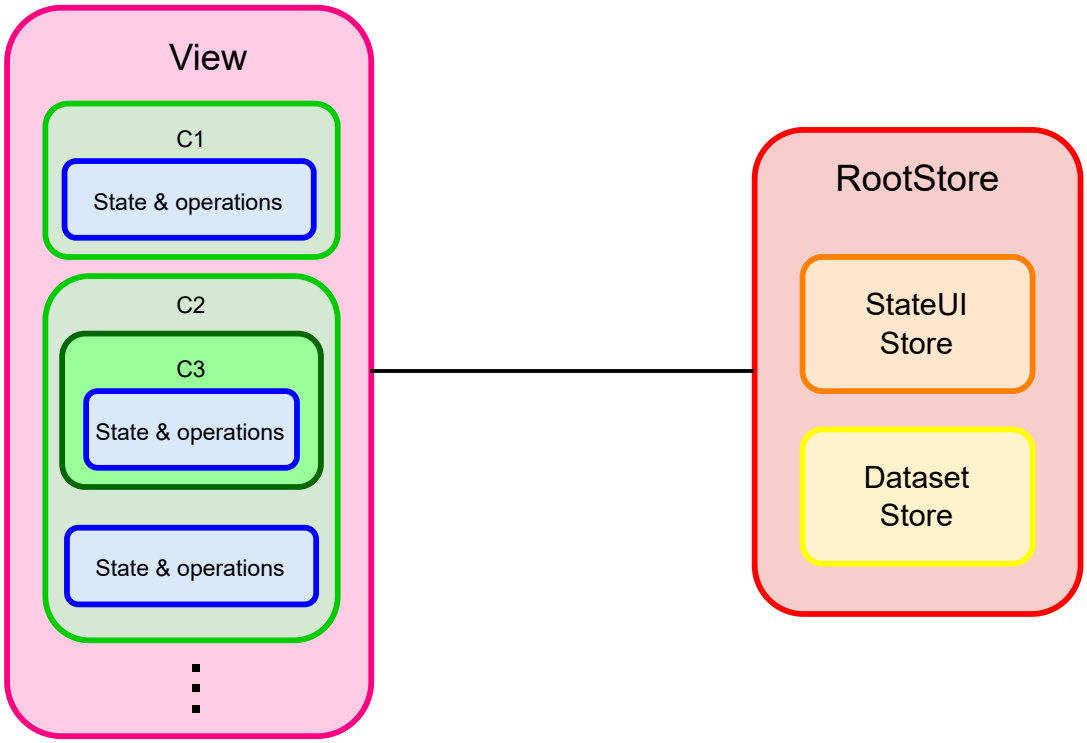

Figura 1: Architettura

Legenda:

- *Cx*: componente x, ognuna può essere formata da più componenti più piccole

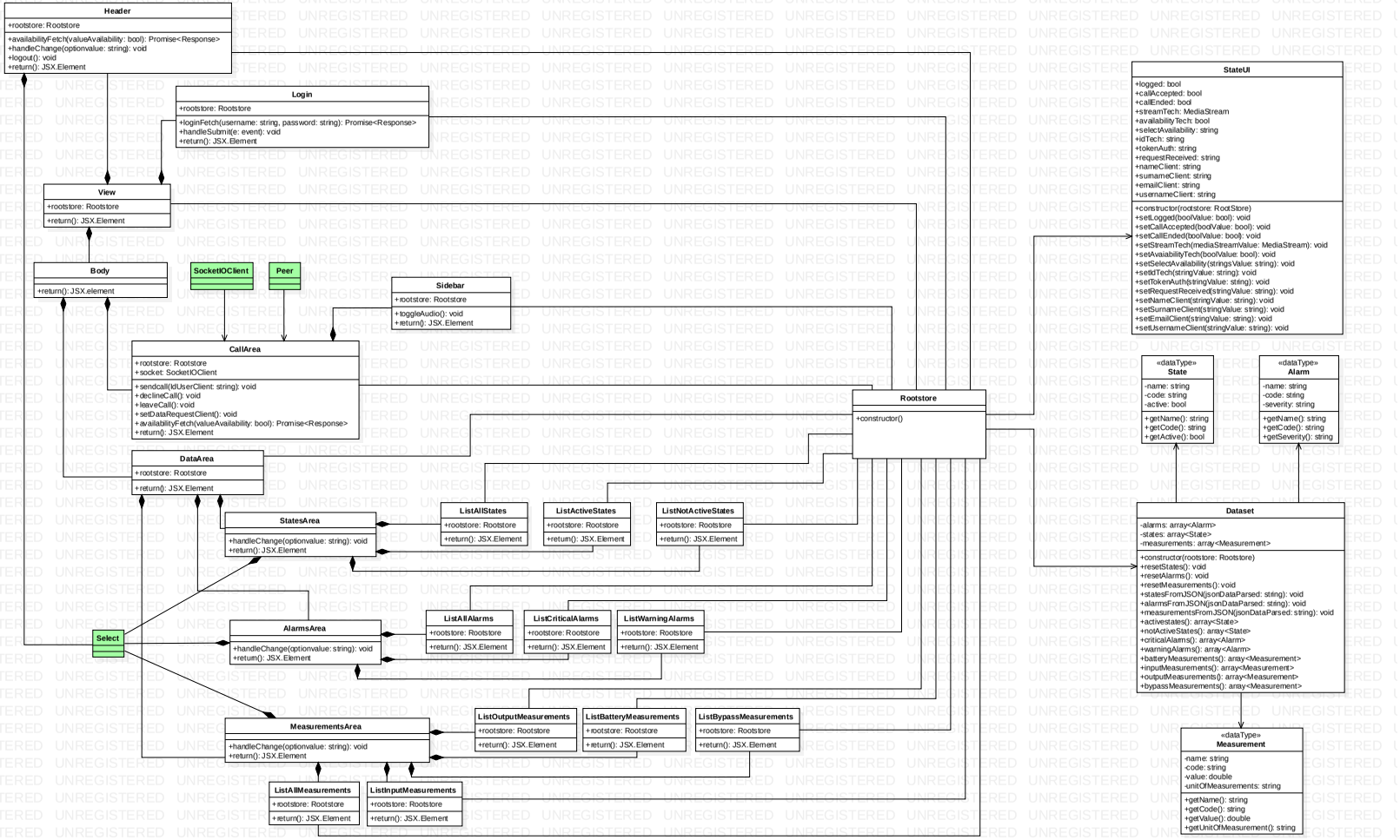

Figura 2: Diagramma delle classi Remote Support

5.1.1 Model

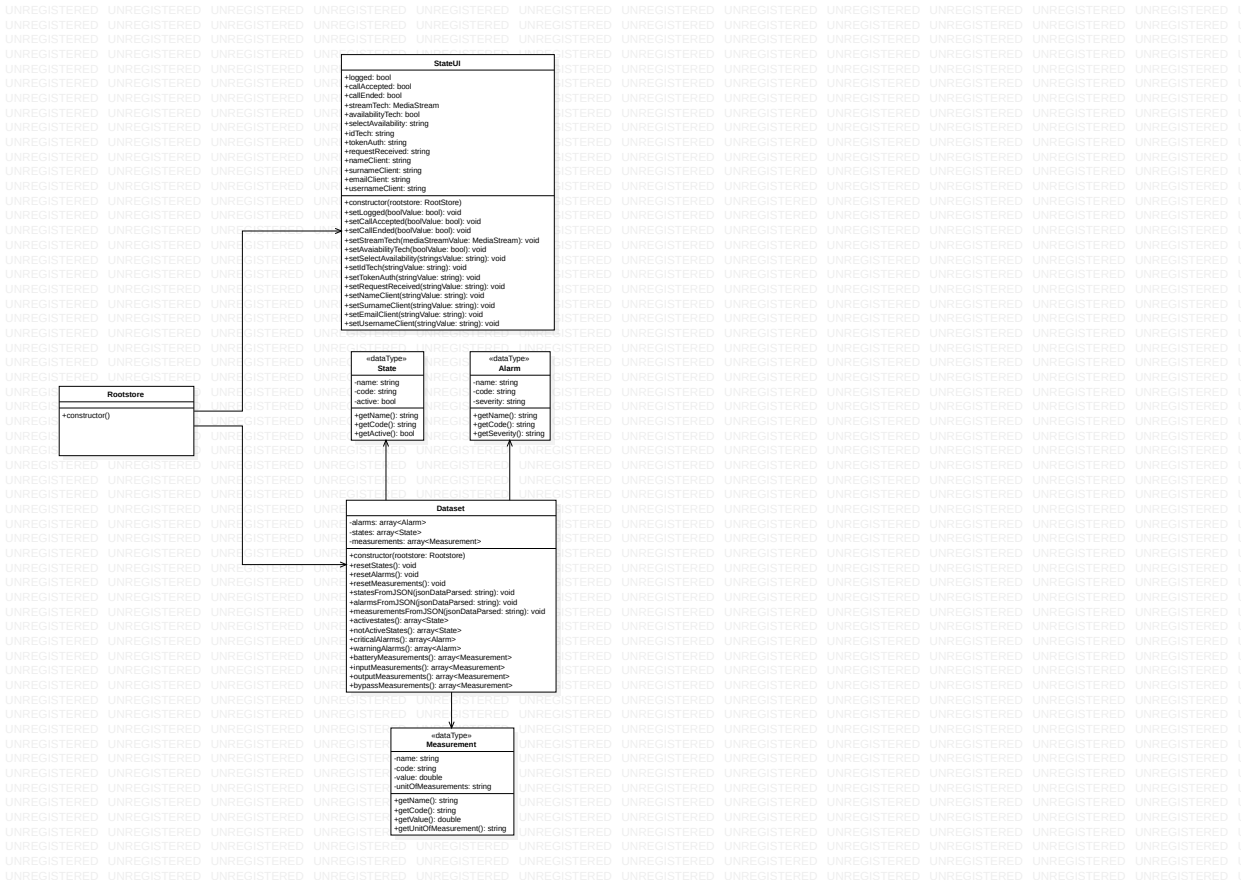

Figura 3: Diagramma delle classi Remote Support, zoom sul Model

Il diagramma delle classi del *Model* è costituito da un **RootStore**, il quale istanzia i due store utilizzati nel Remote Support:

- *StateUI Store*: contiene le variabili di stato che sono condivise tra componenti che non sono direttamente correlate tra loro (esempio: variabili per la renderizzazione condizionale).
- *Dataset Store*: contiene i dati che vengono presi dalle componenti della View, dotato di metodi per il recupero e l'aggiornamento degli stessi.

State, **Alarm** e **Measurement** sono i tipi che rappresentano le varie tipologie di dati.
Gli store contengono gli attributi observable che, nel momento in cui vengono modificati, grazie all'utilizzo di *MobX*, causano la re-renderizzazione dei componenti observer della vista.

5.1.2 View

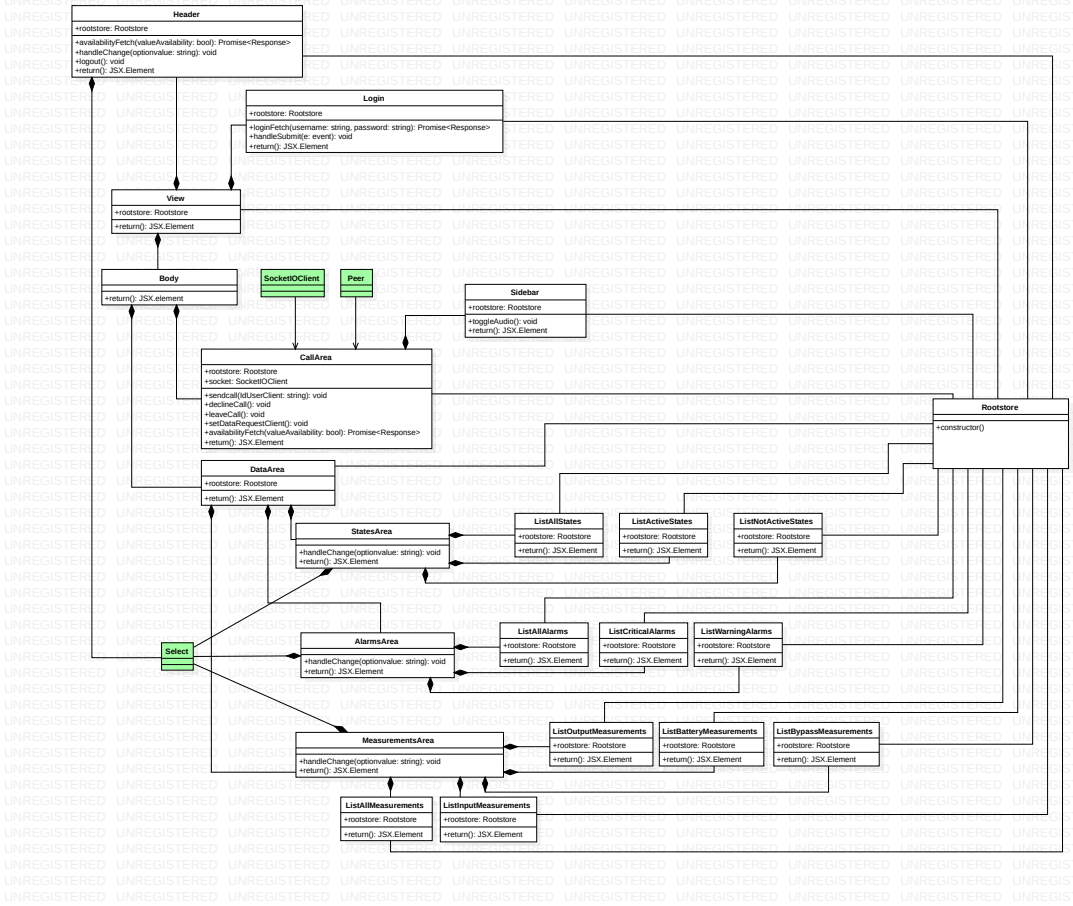

Figura 4: Diagramma delle classi Remote Support, zoom sulla View

Il diagramma delle classi della vista è costituito da tutti i componenti React che compongono l'interfaccia utente della UI. Visitando dall'alto la gerarchia di componenti, il padre è **View**, il quale crea l'**Header** e i due componenti principali **Login** e **Body**, tutti renderizzati condizionalmente in base al fatto che il tecnico abbia effettuato o meno l'accesso correttamente.
Riguardo a quest'ultimi due componenti:

- *Login*: rappresenta il punto di accesso per il tecnico
- *Body*: contiene una *DataArea* per la rappresentazione dei dati con i relativi filtri e una *CallArea* per la visualizzazione dello stato della chiamata

Tutte le classi sono degli **observer** verso il **Rootstore**, dunque faranno un re-render in caso le variabili cambino di valore.
In verde sono evidenziate le classi che hanno dipendenze che però provengono da librerie esterne.