Università degli Studi di Padova

BugPharma - *Login Warrior*

E-mail: bugpharma10@gmail.com

Manuale del Manutentore

Versione	2.0.0
Approvazione	Nicholas Sertori
Redazione	Michele Masetto, Sara Nanni, Silvia Giro, Nicla Faccioli, Andrea Salmaso, Lorenzo Piran, Nicholas Sertori
Verifica	Silvia Giro
Stato	Approvato
Uso	Esterno
Distribuzione	Prof. Vardanega Tullio Prof. Cardin Riccardo Zucchetti S.p.A. Gruppo BugPharma

Registro delle modifiche

Versione	Data	Descrizione	Autore	Ruolo
2.0.0	09/08/2022	Approvazione del documento	Nicholas Sertori	Responsabile
1.1.0	08/08/2022	Revisione del documento	Silvia Giro	Verificatore
1.0.3	29/07/2022	Aggiornamento diagrammi delle classi e di sequenza	Michele Masetto, Silvia Giro	Progettista, Verificatore
1.0.2	20/07/2022	Modifica nomi classi e verifica	Andrea Salmaso, Nicholas Sertori	Progettista, Verificatore
1.0.1	12/07/2022	Modifica \$6 e verifica	Andrea Salmaso, Nicholas Sertori	Progettista, Verificatore
1.0.0	21/06/2022	Approvazione del documento	Michele Masetto	Responsabile
0.2.0	20/06/2022	Revisione del documento	Andrea Salmaso, Silvia Giro	Verificatori
0.1.2	19/06/2022	Stesura \$7 e verifica	Nicholas Sertori, Lorenzo Piran	Progettista, Verificatore
0.1.1	18/06/2022	Stesura \$6 e verifica	Nicholas Sertori, Nicla Faccioli	Progettista, Verificatore
0.1.0	07/05/2022	Verifica complessiva del documento	Nicla Faccioli	Verificatore
0.0.4	30/04/2022	Modifica \$5 e verifica	Michele Masetto, Silvia Giro	Amministratore, Verificatore
0.0.4	25/04/2022	Stesura \$3, \$4 e verifica	Sara Nanni, Andrea Salmaso	Responsabile, Verificatore
0.0.3	20/04/2022	Stesura \$2, \$5 e verifica	Sara Nanni, Michele Masetto, Silvia Giro	Responsabile, Amministratore, Verificatore
0.0.2	19/04/2022	Stesura \$1 e verifica	Michele Masetto, Nicholas Sertori	Amministratore, Verificatore
0.0.1	19/04/2022	Creazione documento	Michele Masetto	Amministratore

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Questo documento ha lo scopo di servire da linea guida per gli sviluppatori che andranno ad estendere o manutenere il prodotto. Di seguito lo sviluppatore troverà tutte le informazioni riguardanti i linguaggi e le tecnologie utilizzate, l'architettura del sistema e le scelte progettuali effettuate per il prodotto. Questo documento ha anche il fine di illustrare le procedure per l'installazione e lo sviluppo in locale.

1.2 Scopo del prodotto

Le applicazioni cloud in tecnologia web stanno gradualmente sostituendo le applicazioni tradizionali "on premise": portano diversi vantaggi, soprattutto in termini di *TCO*$_{\text{G}}$ (Total Cost of Ownership), in quanto funzionano con un semplice *browser*$_{\text{G}}$ web e con una connessione ad Internet veloce e con bassa latenza. Nonostante questo, presentano la criticità di poter subire attacchi da parte di script kiddies, hacker e criminali informatici: essendo infatti sistemi esposti in rete, possono ricevere per loro natura connessioni da tutto il mondo, anche da utenti che non sono esattamente coloro per cui sono state rilasciate. Nasce quindi la necessità di distinguere un utente malintenzionato da uno legittimo nel suo accedere ad applicazioni di questo tipo, così da migliorarne l'esperienza d'uso, attraverso un sistema di analisi esplorativa dei dati ottenuti dai *login*$_{\text{G}}$.

Il capitolato C5, *Login Warrior*, pone come obiettivo la realizzazione di un'applicazione di visualizzazione di dati di login a supporto della fase esplorativa, *EDA*$_{\text{G}}$ (Exploratory Data Analysis), attraverso grafici di varia tipologia, quali:

- Scatter Plot;
- Parallel Coordinates;
- Force-Directed Graph;
- Sankey Diagram.

L'utente dovrà quindi essere in possesso di un file `.CSV` contenente il *dataset*$_{\text{G}}$ che potrà essere caricato nell'applicazione. Tale applicazione sarà fruibile attraverso un browser in grado di supportare le tecnologie *HTML5*$_{\text{G}}$, *CSS*$_{\text{G}}$ e *JavaScript*$_{\text{G}}$.

1.3 Glossario

Al fine di evitare possibili ambiguità relative al linguaggio utilizzato nei documenti, viene fornito il *Glossario v2.0.0* nel quale sono contenute le definizioni di termini aventi uno specifico significato. Tali termini, ove necessario, sono segnati in corsivo e marcati con una $_{\text{G}}$ a pedice.

1.4 Maturità del documento

Il presente documento è stato redatto con un approccio incrementale al fine di poter trattare nuove o ricorrenti questioni in modo rapido ed efficiente, sulla base di decisioni concordate tra tutti i membri del gruppo. Nella sua attuale versione, il documento viene considerato completo.

1.5 Riferimenti

1.5.1 Riferimenti normativi

- *Norme di Progetto v3.0.0*;
- Capitolato d'appalto C5 - Login Warrior:
  `https://www.math.unipd.it/~tullio/IS-1/2021/Progetto/C5.pdf`
- Slide PD2 del corso di Ingegneria del Software - Regolamento del Progetto Didattico:
  `https://www.math.unipd.it/~tullio/IS-1/2021/Dispense/PD2.pdf`

1.5.2 Riferimenti informativi

- Slide P02 del corso di Ingegneria del Software - Diagrammi delle classi:
  `https://www.math.unipd.it/~rcardin/swea/2021/Diagrammi%20delle%20Classi_4x4.pdf`
- Slide P02 del corso di Ingegneria del Software - Diagrammi dei package:
  `https://www.math.unipd.it/~rcardin/swea/2021/Diagrammi%20dei%20Package_4x4.pdf`
- Slide P05 del corso di Ingegneria del Software - Diagrammi di sequenza:
  `https://www.math.unipd.it/~rcardin/swea/2022/Diagrammi%20di%20Sequenza.pdf`
- Component Based Architecture - Wikipedia:
  `https://en.wikipedia.org/wiki/Component-based_software_engineering`
- Component Based Architecture:
  `https://www.tutorialspoint.com/software_architecture_design/component_based_architecture.htm`
- Documentazione Redux - Fundamentals:
  `https://redux.js.org/tutorials/fundamentals/part-1-overview`
- Documentazione Redux - App Structure:
  `https://redux.js.org/tutorials/essentials/part-2-app-structure`

2 Tecnologie coinvolte

Tecnologia	Versione	Descrizione
Linguaggi		
JavaScript	ES6	Utilizzato per la creazione di effetti dinamici e interattivi tramite eventi invocati dall'utente. E' il linguaggio della libreria *React*$_G$.
HTML	5	Linguaggio di markup utilizzato insieme a React per impostare la struttura delle pagine web.
CSS	3	Utilizzato per la formattazione e la definizione dello stile dei documenti HTML.
Strumenti		
Npm	7.x	Gestore di pacchetti.
Librerie e framework		
React	18.0.x	Libreria JavaScript per la creazione di componenti grafiche scelta per facilitare lo sviluppo del front-end ed ottenere migliori performance grazie al suo metodo di renderizzazione delle componenti grafiche.
D3.js$_G$	6.x	Libreria per creare visualizzazioni dinamiche ed interattive partendo da dati organizzati.
React Redux	8.0.x	Libreria JavaScript per gestire e centralizzare lo stato dell'applicazione.

Tabella 1: Tecnologie coinvolte

3 Configurazione

In questa sezione sono elencati i requisiti minimi necessari per l'utilizzo dell'applicazione *Login Warrior*, seguiti successivamente dalla guida all'installazione del prodotto in locale direttamente dal *repository*$_G$ pubblico *GitHub*$_G$ del gruppo *BugPharma*.

3.1 Requisiti di sistema

Per far sì che le operazioni di installazione e avvio del prodotto avvengano correttamente e che si possa aver accesso a tutte le funzionalità, è necessario aver installato nella propria macchina i seguenti software:

Software	Versione	Riferimento per il download
Node.js	17.4.0	`https://nodejs.org/it`

Tabella 2: Requisiti di sistema

3.2 Requisiti hardware

Per ottenere prestazioni accettabili dell'applicazione in uso è consigliato l'utilizzo dei seguenti componenti hardware:

Componente	Requisito
Processore	Quad-Core 3,2 GHz
Ram	8GB DDR4

Tabella 3: Requisiti hardware minimi

3.3 Requisiti browser

L'applicazione è stata testata e resa compatibile con le ultime versioni dei browser maggiormente utilizzati:

Browser	Versione
Chrome	98.0
Edge	97.0
Mozilla Firefox	97.0
Safari	15.0
Opera	83.0

Tabella 4: Browser e versioni compatibili

4 Installazione

Per usufruire dell'applicazione web è necessario svolgere i seguenti passi:

- Clonare il repository **(Obbligario)**;
- Avviare la web app **(Obbligatorio)**.

4.1 Clonare il repository

1. Scaricare il codice come file .zip direttamente dal repository denominato BugPharma-Login-Warrior:

   `https://github.com/BugPharma/BugPharma-Login-Warrior`

   Oppure, con Git installato in locale, è possibile clonare il repository utilizzando il comando :

   `git clone https://github.com/BugPharma/BugPharma-Login-Warrior`

2. Localizzare da terminale la cartella in cui è stato estratto/clonato il prodotto:
   cd percorso\BugPharma-Login-Warrior

4.2 Avvio della web app

Attualmente l'applicazione è ancora in fase di sviluppo. Di conseguenza i passi per l'avvio sono i seguenti:

1. Nel caso di primo avvio eseguire il comando:

   `cd BugPharma-Login-Warrior;`

2. Eseguire il comando:

   `npm install --force;`

3. Per avviare l'applicazione digitare:

   `npm start`

5 Strumenti per l'analisi del codice

Strumento	Versione	Descrizione
Analisi statica		
Prettier	9.5.x	Strumento per la formattazione automatica del codice.
SonarCloud	-	Servizio web per eseguire controlli di qualità del codice presente all'interno del repository, riportando tutti i risultati in un cruscotto.
SonarLint	3.5.x	Estensione per IDE che permette di individuare problematiche del codice mentre lo si scrive.
Analisi dinamica		
Jest	28.0.x	Framework di testing utilizzato per l'analisi dinamica del codice.
React Testing Library	12.1.x	Libreria utilizzata per testare le componenti scritte in React.
Puppeteer	-	Libreria di Node.js per effettuare test di sistema di tipo E2E.
Codecov	-	Servizio per l'analisi della code coverage del codice.
GitHub Actions	-	Strumento fornito da GitHub che permette di definire workflows personalizzati all'interno di repository.

Tabella 5: Strumenti utilizzati per l'analisi del codice

6 Architettura

L'architettura di *Login Warrior* è basata sul design pattern Component-Based-Architecture (CBA_G).

Il principio fondante della CBA è di decomporre il sistema in componenti logiche o funzionali che contengano ciascuna i metodi e le proprietà necessarie alla stessa. Ciascuna componente contiene tutti i dettagli implementativi necessari alla stessa e può comunicare con ciascuna delle altre attraverso una interfaccia . Tutto ciò ha come obiettivo quello di realizzare codice che sia il più modulare e riutilizzabile possibile.

Il gruppo ha scelto di orientarsi su questa tipologia di architettura in concordanza con l'utilizzo del framework React per la realizzazione della web-app. Anche React infatti si basa sul concetto di componente e risulta quindi più naturale utilizzare questo tipo di architettura rispetto ad altri pattern (come Model-View-*), che richiederebbero forzature e una conseguente deviazione dai principi base di funzionamento di React.

6.1 Diagrammi dei package

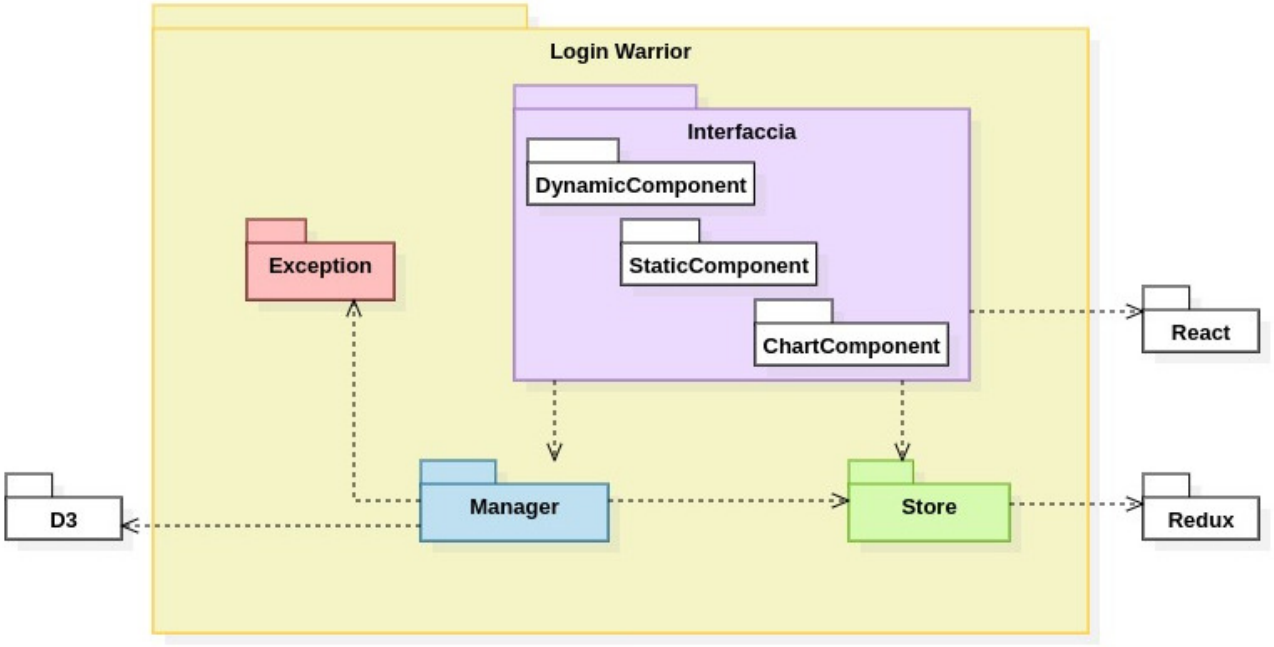

Figura 1: Diagramma dei package

Il diagramma sopra riportato rappresenta la struttura dell'applicazione a livello di packages. In particolare:

- **Vista**: ha una dipendenza verso $React_G$ poichè si è scelto di implementare l'interfaccia dell'applicazione utilizzando tale framework di $JavaScript_G$. Al suo interno si divide a sua volta in **DynamicComponent**, **StaticComponent** e **ChartComponent**;
- **D3**: libreria di JavaScript utilizzata per la visualizzazione dei grafici;
- **Redux**: libreria Javascript per gestire e centralizzare lo stato dell'applicazione;
- **Exception**: package contenente la gestione degli errori.

6.2 Diagrammi delle classi

6.2.1 Interfaccia utente

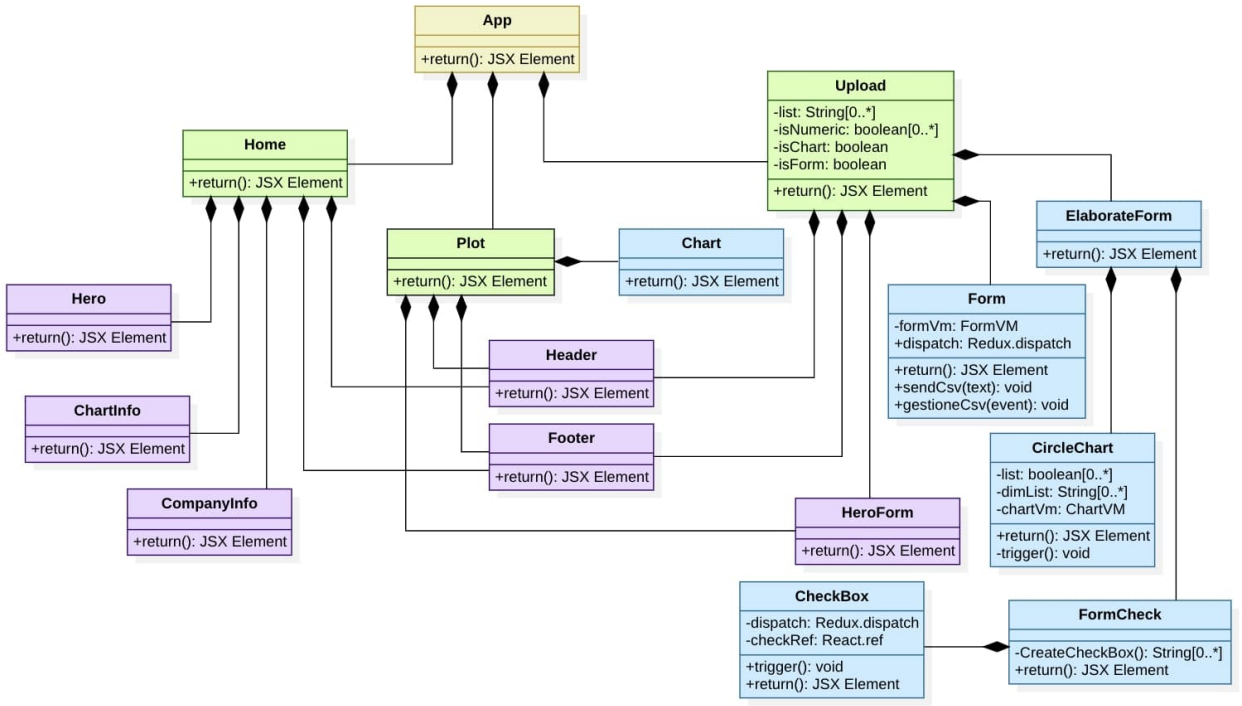

Figura 2: Diagramma delle classi dell'interfaccia utente

Il diagramma delle classi sopra riportato rappresenta tutte le componenti React utilizzate per la creazione dell'interfaccia utente di *Login Warrior*. In particolare:

- `App`: componente principale. Si occupa della creazione delle componenti corrispondenti alle pagine dell'applicazione (Home, Upload e Plot);
- `Home`: pagina completamente statica, semplice landing page;
- `Upload`: corrisponde alla pagina da cui l'utente può accedere alle funzionalità di caricamento di un file `.csv`, scelta delle dimensioni e scelta della tipologia di grafico da visualizzare;
- `Plot`: è la pagina in cui viene mostrato il grafico prodotto e in cui l'utente può personalizzarlo;
- `Form`: componente per mostrare un form che permetta l'inserimento del file contente i dati da elaborare;
- `ElaborateForm`: componente che si occupa di creare le varie istanze necessarie di `CircleChart` e quella di `FormCheck`;
- `Chart`: componente che contiene un unico tag che viene utilizzato per inserire il grafico all'interno della pagina una volta che esso è stato creato.

- `FormCheck`: componente che si occupa della creazione e visualizzazione delle componenti `CheckBox` necessarie.
- `CheckBox`: singolo checkbox. Ogni volta che viene cliccato, esso invia il proprio stato al modello.

Infine, le componenti di colore azzurro all'interno del diagramma si limitano a rappresentare una specifica parte statica di interfaccia utente.

6.2.2 Store Redux

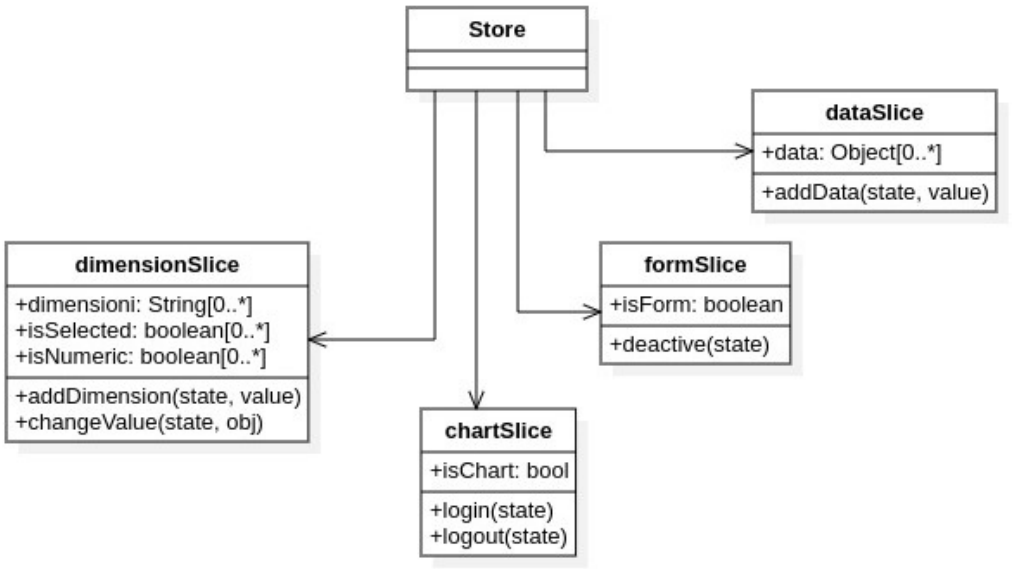

Figura 3: Diagramma delle classi del modello

Per gestire in modo più corretto e agevole lo stato globale dell'applicazione, il gruppo ha deciso di utilizzare la libreria Redux di Javascript. Redux organizza i dati all'interno di $store_{\mathrm{G}}$: oggetti Javascript con alcune particolari funzionalità che li rendono più manutenibili e testabili rispetto a semplici oggetti globali. In particolare:

- Non è possibile modificare direttamente lo stato contenuto all'interno di uno store;
- L'unico modo per modificare lo stato è creando un oggetto $action_{\mathrm{G}}$ che descriva "qualcosa accaduto all'interno dell'applicazione". Tale azione viene comunicata allo store;
- Quando questo accade, lo store esegue una funzione $reducer_{\mathrm{G}}$ con il compito di calcolare il nuovo stato in base allo stato precedente ed all'azione comunicata;
- Infine lo store notifica i subscribers del cambiamento nello stato in modo tale che l'interfaccia possa aggiornarsi con i nuovi dati.

Lo store inoltre è suddiviso in $slice_{\mathrm{G}}$, ognuna contenente parti diverse dello stato con relativi reducer e action.

Login Warrior è costituito da un unico `store` suddiviso in 4 diverse `slice`:

- `dimensionSlice`: formata da tre diversi array:

  - uno contenente i nomi di tutte le dimensioni estratte dal file `csv`;
  - uno contenente le dimensioni scelte dall'utente per la creazione del grafico;
  - uno per determinare determinare quali campi siano numerici.

- `chartSlice`: conserva lo stato della parte di interfaccia contenente il grafico, ovvero se sia visualizzata oppure no;
- `dataSlice`: conserva i dati risultanti dal parsing (in formato `JSON`) all'interno di un array;
- `formSlice`: conserva lo stato della parte di vista contenente il form, ovvero se sia visualizzata oppure no.

6.2.3 Manager

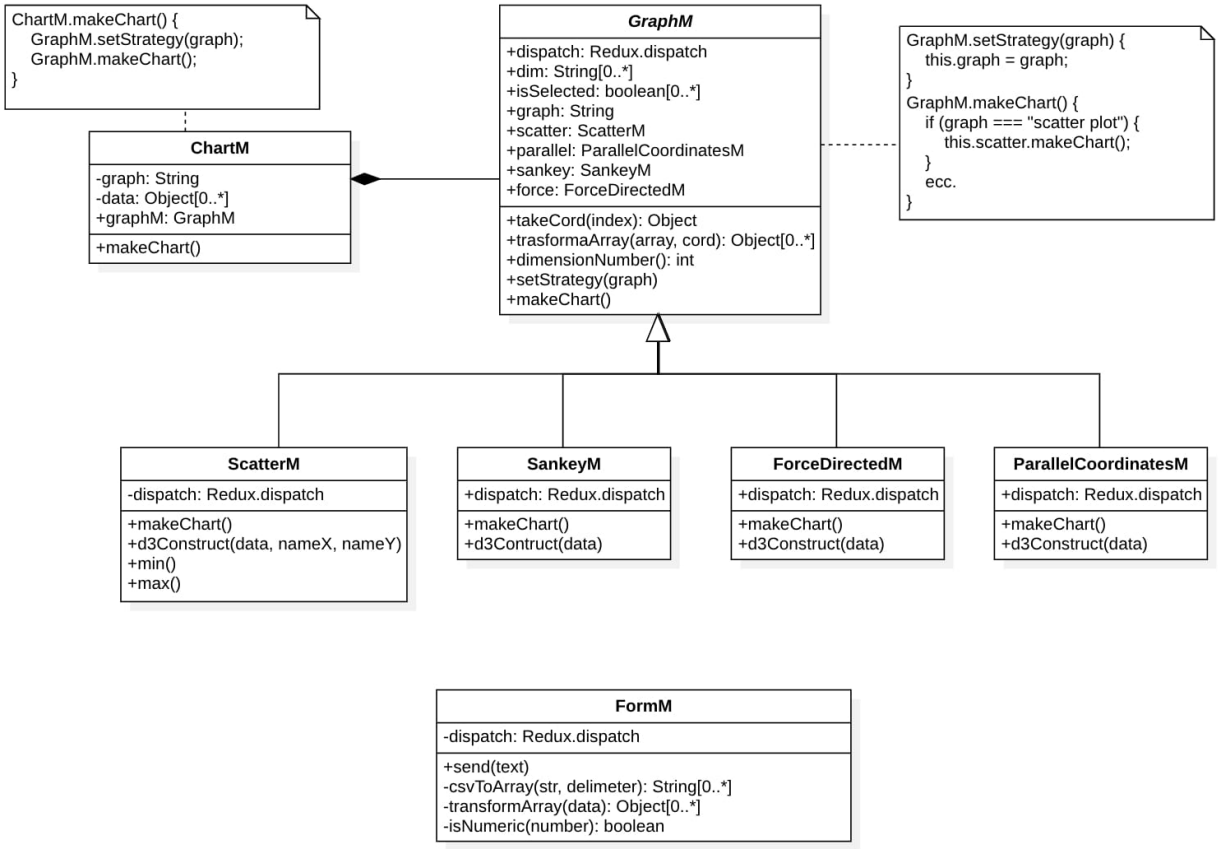

Figura 4: Diagramma delle classi dei ViewModel

I manager si occupano dell'elaborazione dei dati e della effettiva costruzione dei grafici. In *Login Warrior* è stata utilizzata la seguente suddivisione di responsabilità:

- `ChartM`: riceve le informazioni sul grafico da creare e si occupa di comunicarle nel modo corretto a `GraphM` in modo tale da poter sfruttare il design pattern Strategy;

- `GraphM`: classe astratta e interfaccia dello Strategy. Si occupa di prendere le informazioni sul grafico da creare e comunicarle allo specifico grafico selezionato dall'utente;
- `ScatterM`: contiene il codice specifico per la costruzione del diagramma Scatter Plot;
- `SankeyM`: contiene il codice specifico per la costruzione del diagramma Sankey;
- `ParallelM`: contiene il codice specifico per la costruzione del diagramma Parallel Coordinates;
- `Force`: contiene il codice specifico per la costruzione del diagramma Force Directed;
- `FormM`: riceve i dati in formato `.csv` e si occupa di elaborarli. In particolare:
  - Fa il parsing dei dati;
  - Salva i nomi delle dimensioni nello store dedicato;
  - Salva i dati nello store dedicato sotto forma di array. Tale array avrà un elemento in formato `JSON` per ciascuna riga del file inserito dall'utente.

6.3 Diagrammi di sequenza

6.3.1 Caricamento dati tramite file `csv`

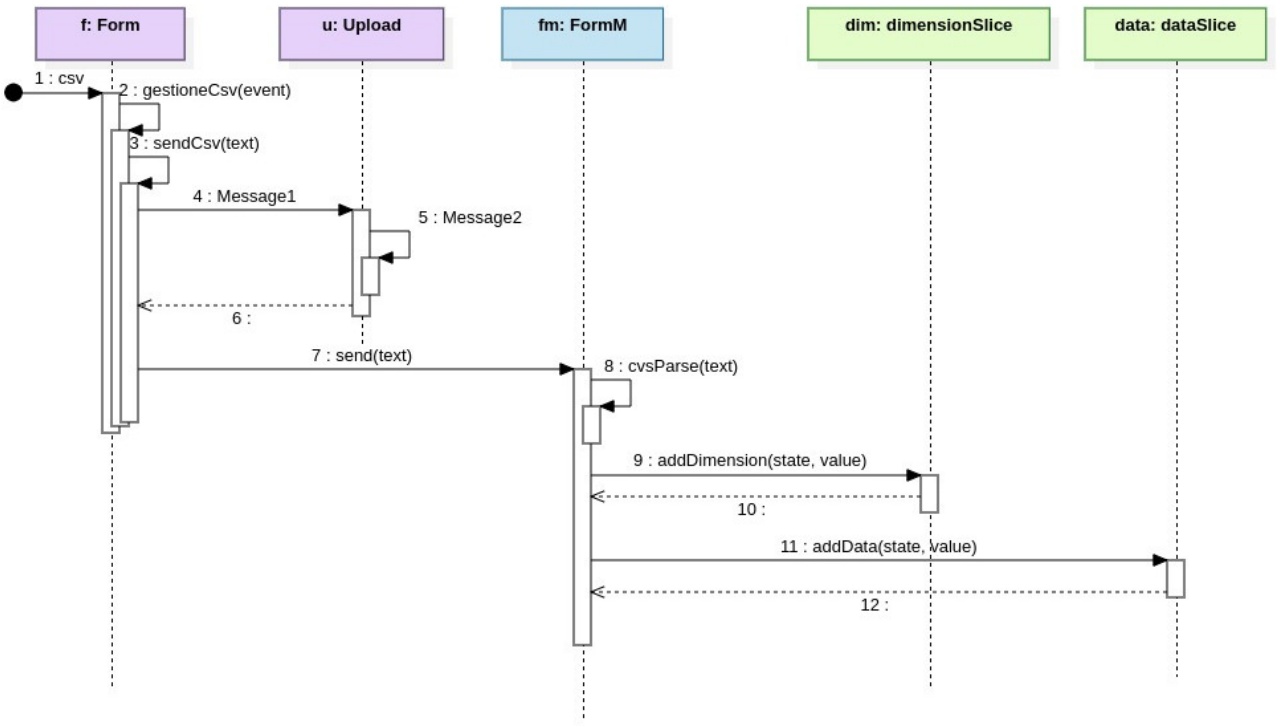

Figura 5: Caricamento del file `csv` e salvataggio dei dati nello Store

Il grafico sopra riportato si compone delle seguenti attività:

1. `Form` è la componente attraverso cui l'utente può inserire il proprio `csv`. Una volta caricato, `change()` si occupa di cambiare un parametro all'interno della pagina `Upload` (`setIsForm(boolean)`) per fare in modo che venga nascosto il form di caricamento e venga mostrata al suo posto la lista di checkbox formata dalle intestazioni delle dimensioni contenute nel file caricato.

2. `send(text)` invia il file a `FormM`, il cui compito è quello di trasformare il testo letto dal `csv` in un array di oggetti.

3. Successivamente viene estrapolata la prima riga del file, contenente le intestazioni delle colonne. Tale prima riga e il resto dei dati vengono infine salvati separatamente all'interno dello store (`addDimension(state, value)` per la prima riga, `addData(state,value)` per il resto dei dati).

6.3.2 Creazione della lista di checkbox delle dimensioni

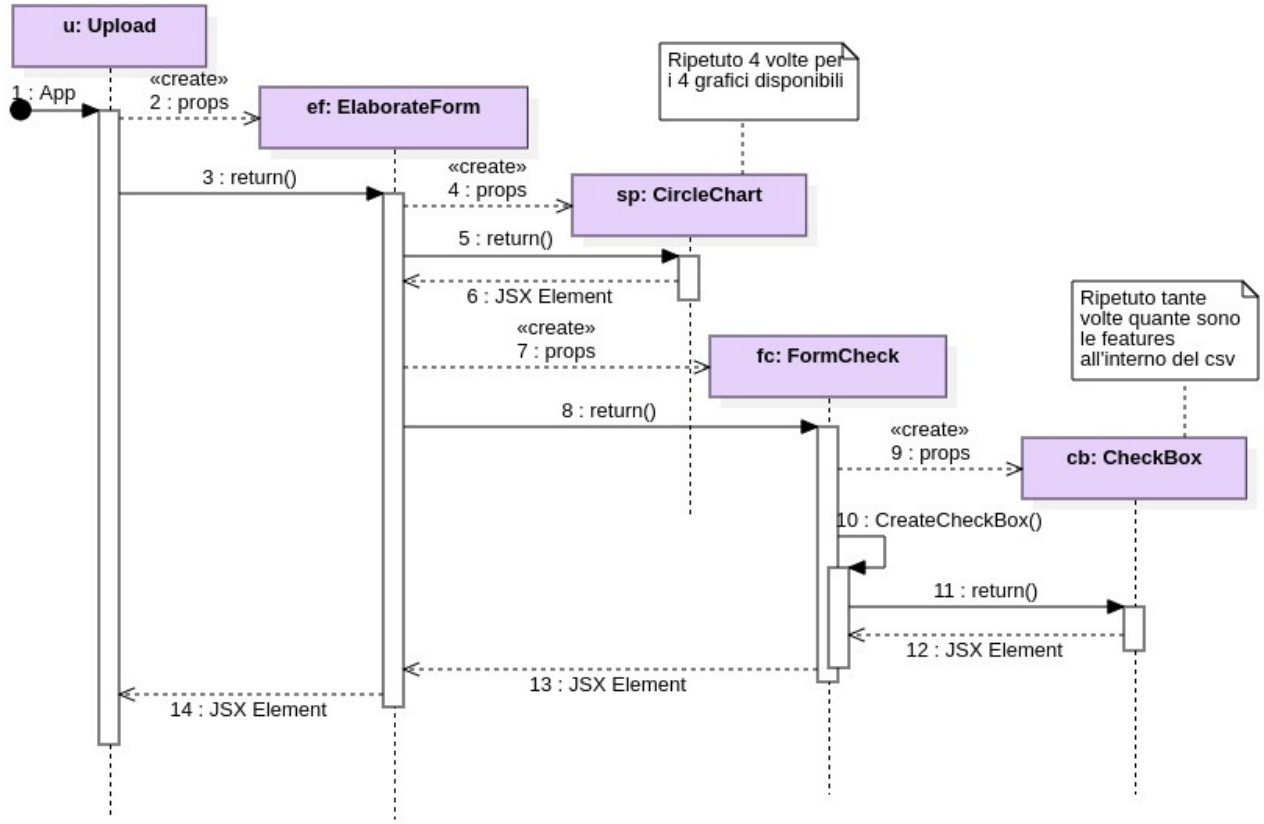

Figura 6: Creazione della lista di checkbox delle dimensioni

Il grafico sopra riportato si compone delle seguenti attività:

1. `Upload` crea e mostra `ElaborateForm`, secondo le regole di funzionamento di React;

2. `ElaborateForm` crea a sua volta le diverse componenti del form:

   - `FormCheck`: componente per la visualizzazione della lista di checkbox;
   - `CircleChart`: pulsanti, uno per ogni grafico. Il loro click comporta una serie di operazioni per visualizzare lo specifico grafico di cui è stato premuto il pulsante.

3. `FormCheck` crea infine un'istanza di `CheckBox` per ciascuna delle dimensioni salvate nello store `dimensionsSlice` utilizzando `CreateCheckBox()`.

6.3.3 Creazione del grafico

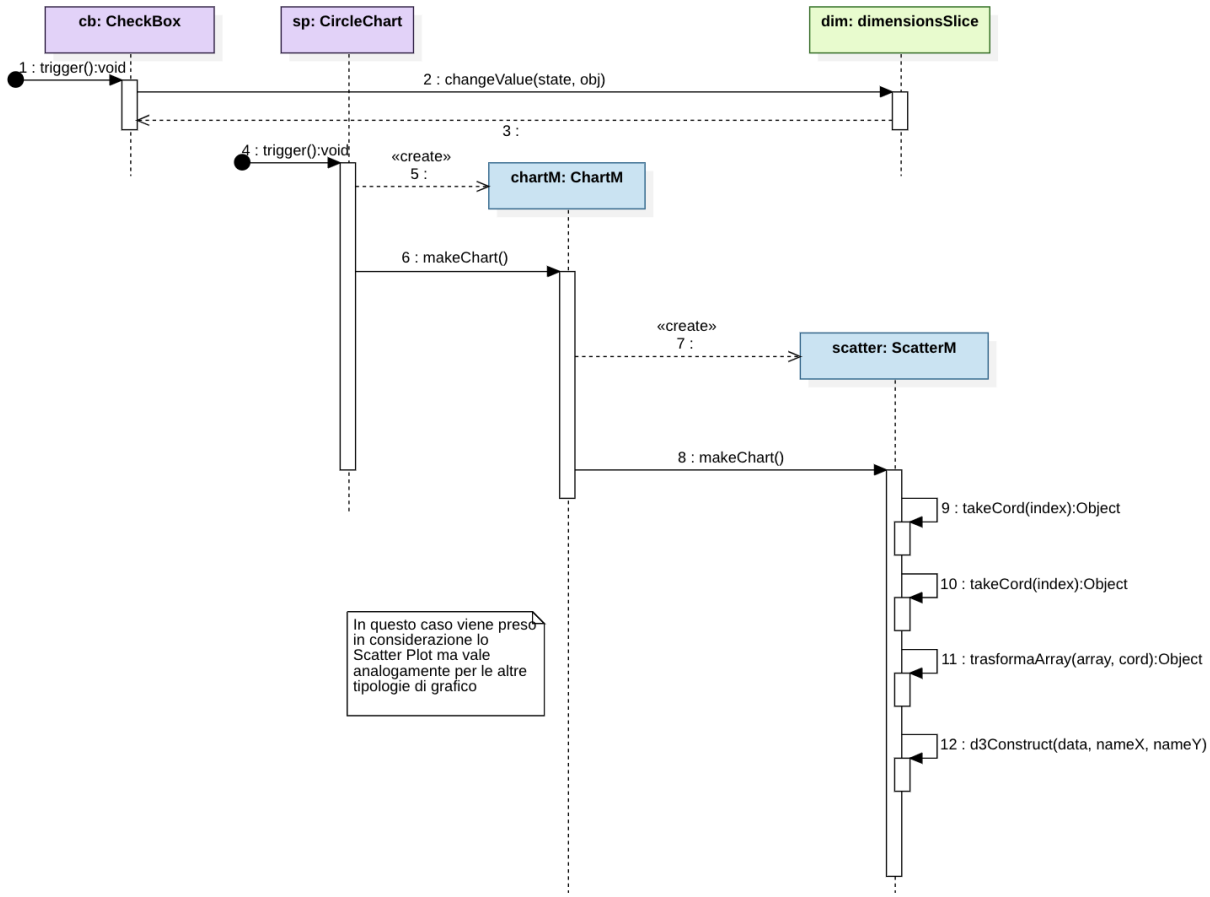

Figura 7: Creazione del grafico

Il grafico sopra riportato si compone delle seguenti attività:

1. Nel momento in cui viene cliccato uno specifico `CheckBox`, viene immediatamente aggiornato lo stato all'interno dello `store`, in particolare del `dimensionsSlice`.
2. Una volta selezionate le dimensioni, l'utente preme sull'elemento `CircleChart` corrispondente al grafico che vuole visualizzare.
3. `CircleChart` crea la propria istanza di `ChartM` e ne utilizza la funzione `makeChart()`.
4. `ChartM` crea l'istanza di `GraphM`, che crea a sua volta le istanze dei quattro grafici possibili. Nell'esempio è riportato solamente il caso dello Scatter Plot (`ScatterM`);
5. `ChartM` viene costruito passandogli come parametro una stringa per comunicare quale tipo di grafico tra quelli disponibili è stato selezionato per la visualizzazione dei dati. Questo per sopperire al fatto che il linguaggio JavaScript non è tipizzato e di conseguenza non è possibile utilizzare il subtyping per realizzare un vero e proprio design pattern Strategy;
6. Successivamente `ChartM` chiama la funzione `makeChart()` di `GraphM` che si occuperà a sua volta di chiamare la funzione `makeChart()` del grafico scelto;

7. Infine il manager specifico del grafico, in questo caso `ScatterM`, procederà all'effettiva creazione del grafico chiamando il metodo `d3Construct(data, nameX, nameY)` utilizzando la libreria `d3.js`.

7 Estensione del prodotto

7.1 Visualizzazione dei dati

Login Warrior mette a disposizione quattro diverse tipologie di grafico per la visualizzazione dei dati inseriti. È possibile aggiungerne ulteriori seguendo i seguenti passi:

1. Creare la classe manager del nuovo grafico da inserire. Al suo interno dovrà essere inserita tutta la logica di creazione e personalizzazione del grafico. Per permettere al grafico di essere visualizzato nella pagina React basterà utilizzare `d3.js` per selezionare ed appendere il grafico stesso all'elemento con id `mydataviz`;
2. Aggiungere a `GraphM` il campo dati relativo alla classe manager del nuovo grafico;
3. Modificare la classe `CircleChart` in modo tale che venga mostrato anche il pulsante per il nuovo grafico inserito tra quelli a disposizione dell'utente.