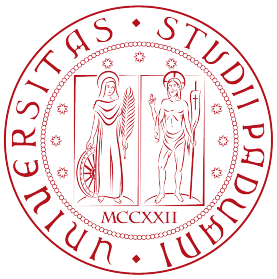
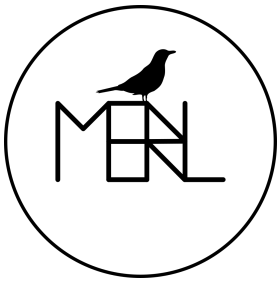


Università degli Studi di Padova
Laurea: Informatica
Corso: Ingegneria del Software
Anno Accademico: 2021/2022



Gruppo: MERL
Email: merlunipd@gmail.com

Manuale Sviluppatore

Informazioni sul documento

Versione	V1.0.0
Uso	Esterno
Data approvazione	08/06/2022
Distribuzione	Prof. *Vardanega Tullio* Prof. *Cardin Riccardo* *Zucchetti S.p.A.* Gruppo *MERL*

Registro delle Modifiche

Versione	Data	Autore	Verificatore	Modifica
v1.0.0	08/06/2022	Marco Mamprin	-	Approvazione
v0.0.3	06/05/2022	Emanuele Pase	Lorenzo Onelia	Aggiunta sezione "Strumenti per l'analisi e l'integrazione del codice"
v0.0.2	06/05/2022	Emanuele Pase	Lorenzo Onelia	Aggiunta sezione "Principali punti di estensione"
v0.0.1	04/05/2022	Emanuele Pase	Lorenzo Onelia	Aggiunti capitoli 'Introduzione', 'Tecnologie', 'Requisiti minimi di sistema', 'Architettura' e 'Installazione'
v0.0.0	03/05/2022	Lorenzo Onelia	Emanuele Pase	Creata prima struttura del documento

Indice

Elenco delle figure

Elenco delle tabelle

1. Introduzione

1.1 Scopo del documento

Lo scopo del *Manuale Sviluppatore* è quello di fornire una linea guida per gli sviluppatori che andranno a manuntenere o estendere il prodotto. Di seguito lo sviluppatore troverà tutte le informazioni riguardanti i linguaggi e le tecnologie e l'architettura utilizzate per la realizzazione del prodotto.

1.2 Scopo del prodotto

Il capitolato proposto dall'azienda *Zucchetti S.p.A.* ha come obiettivo quello di creare un'applicazione di visualizzazione di dati di login con numerose dimensioni che permettono di rintracciare eventuali anomalie a colpo d'occhio. Lo scopo del prodotto è quindi quello di fornire all'utente diversi tipi di visualizzazione di dati in modo da rendere più veloce ed efficace l'individuazione di anomalie.

1.3 Glossario

Al fine di evitare incomprensioni relative alla terminologia usata all'interno del documento, viene fornito un Glossario nel file *Glossario V2.0.0* in grado di dare una definizione precisa per ogni vocabolo potenzialmente ambiguo. Tali termini verranno evidenziati all'interno del documento con una G in pedice.

1.4 Riferimenti

1.4.1 Riferimenti normativi

- *Norme di Progetto V2.0.0*
- Capitolato d'appalto C5 - *Zucchetti S.p.A.*: *Login Warrior*
  https://www.math.unipd.it/~tullio/IS-1/2021/Progetto/C5.pdf

1.4.2 Riferimenti informativi

- Slide T9 del corso di Ingegneria del Software - Progettazione
  https://www.math.unipd.it/~tullio/IS-1/2021/Dispense/T09.pdf

- Slide P2 del corso di Ingegneria del Software - Diagramma delle classi https://www.math.unipd.it/~rcardin/swea/2021/Diagrammi%20delle%20Classi_4x4.pdf
- Slide P3 del corso di Ingegneria del Software - Gestione delle dipendenze https://www.math.unipd.it/~rcardin/swea/2022/Dependency%20Management%20in%20Object-Oriented%20Programming.pdf
- Slide P5 del corso di Ingegneria del Software - Diagramma di sequenza https://www.math.unipd.it/~rcardin/swea/2022/Diagrammi%20di%20Sequenza.pdf
- Slide L02 del corso di Ingegneria del Software - Pattern MVC e derivati https://www.math.unipd.it/~rcardin/sweb/2022/L02.pdf
- Slide L03 del corso di Ingegneria del Software - SOLID programming https://www.math.unipd.it/~rcardin/sweb/2022/L03.pdf
- API libreria per la visualizzazione dei grafici https://github.com/d3/d3/blob/main/API.md

2. Tecnologie

Tecnologia	Versione	Descrizione
Linguaggi		
HTML	5	Linguaggio di markup utilizzato per definire gli elementi dell'interfaccia.
CSS	3	Linguaggio utilizzato per la gestione dello stile degli elementi HTML.
Javascript	ES6	Linguaggio di programmazione ad alto livello, interpretato, multi-paradigma, con tipizzazione debole. Viene utilizzato dal motore del browser per eseguire codice da lato client. Utilizzati i *Moduli ES6* per gestire i file contenenti il codice Javascript.
Librerie		
D3	7.4.0	Libreria Javascript utilizzata per manipolare elementi del DOM_G in base a dati. Permette di creare visualizzazioni e grafici.
Strumenti		
NodeJS	17.2.0	Runtime costruito sul motore V8 di Google per l'esecuzione di codice JavaScript. Utilizzato per accedere a strumenti di supporto allo sviluppo (e.g. JestJS, ESLint) e per la definizione di piccoli script.
NPM	8.1.4	Package manager per la gestione di dipendenze di progetti NodeJS.
JestJS	27.5	Strumento per effettuare analisi dinamica di codice Javascript e per generare il code coverage.
ESLint	8.12	Strumento di analisi statica del codice. Viene utilizzato con le best practices configurate dallo standard *AirBnB*.

JSDocs	3.5.5	Linguaggio di markup che permette di annotare il codice sorgente Javascript e generare documentazione.
IndexedDB	3.0	API_G Javascript fornite dai browser per permettere il caching di dati da lato client.
Git	2.34.1	Strumento di controllo della versione distribuito. Utilizzato per gestire la repository remota su GitHub.

Tabella 2.1: Tabella delle tecnologie utilizzate

3. Requisiti Minimi di Sistema

3.1 Requisiti Minimi

Per far funzionare l'applicazione non ci sono particolare richieste, trattandosi di una Single-page Application.

3.2 Requisiti Consigliati

Per avere un'esperienza completa nell'uso dell'applicazione si consiglia d'installare nella propria macchina i seguenti software:

Software	Versione	Riferimenti per il download
Node.js	16.14.2	https://nodejs.org/en/
Npm	8.x	Integrato nel download di Node.js

Tabella 3.1: Tabella dei requisiti consigliati

3.3 Requisiti Hardware

Al fine di garantire prestazioni accettabili si consiglia di soddisfare i seguenti requisiti hardware:

Componente	Versione
Processore	Quad-Core
RAM	8GB

Tabella 3.2: Tabella dei requisiti hardware

3.4 Browser

I browser testati e resi compatibili con l'applicazione sono:

Browser	Versione
Chrome	99
Edge	99
Firefox	98
Opera	83
Safari	15.2

Tabella 3.3: Tabella dei browser testati e supportati

4. Installazione

Per utilizzare l'applicazione web è necessario:

- Clonare il repository;
- Avviare il server;
- Avviare la web app.

4.1 Clonare il repository

- Scaricare il codice come file .zip direttamente dal repository *login-warrior*:

  https://github.com/merlunipd/login-warrior

  oppure, avendo *Git* installato in locale, è possibile clonare il repository con il comando:

  ```
  git clone https://github.com/merlunipd/login-warrior
  ```

- Localizzare da terminale la cartella in cui è stato estratto/clonato il prodotto:

  ```
  cd percorso\LoginWarrior
  ```

4.2 Avviare il server

- Entrare nella cartella `login_warrior` con i seguenti comandi:

  ```
  cd src
  cd login_warrior
  ```

- In caso di primo avvio, per crea la cartella `node_modules` dove vengono installate tutte le dipendenza necessarie digitare:

  ```
  npm install
  ```

- Per listare gli script impostati digitare (**Opzionale**):

  ```
  npm run
  ```

- Per eseguire un server locale che permette l'accesso all'applicazione digitare:

  ```
  npm run server
  ```

4.3 Avviare la web app

Dopo aver avviato il server come spiegato nel passo precedente, l'applicazione sarà disponibile aprendo l'indirizzo fornito dal terminale:

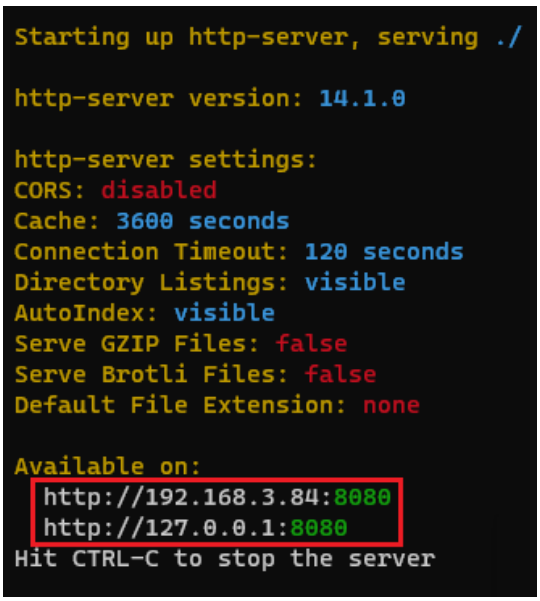


Figura 4.1: Avvio dell'applicazione

5. Strumenti per l'analisi e l'integrazione del codice

Strumento	Versione	Descrizione
Analisi statica		
ESLint	8.9.0	É uno strumento di analisi statica del codice, viene utilizzato per identificare pattern problematici all'interno di codice JavaScript. Compie sia dei check sulla qualità del codice, sia verifica l'aderenza a un particolare coding style. Il gruppo ha deciso di aderire al coding style definito da AirBnB.
Analisi dinamica		
Jest	27.5.1	É un framework di testing per JavaScript, permette di eseguire dei test automatici, definiti dall'utente, per controllare il corretto funzionamento di un programma o progetto.
Documentazione		
JSDocs	3.3.10	É un linguaggio di markup che permette di annotare il codice sorgente JavaScript. Comprende uno strumento che permette di generare automaticamente documentazione del codice in formato HTML o RTF.

Tabella 5.1: Strumenti per l'analisi e l'integrazine del codice

6. Architettura

6.1 Introduzione

L'architettura di *Login Warrior* è basata sul design pattern architetturale *Model-View-Controller*. Il gruppo ha sviluppato un controller per ognuna delle due pagine dell'applicazione, i quali devono gestire le interazioni dell'utente con la GUI_G.

Le viste corrispondono alle pagine dell'applicazione e sono quindi due: la pagina home nella quale verrà caricato il dataset_G o la sessione e visualizzata la lista dei grafici disponibili, e quella dove verrà effettivamente visualizzato il grafico con relativi filtri e personalizzazioni.

Il modello contiene i dati da visualizzare, che vengono presi dal file CSV_G e convertiti in oggetti di tipo *DataPoint* contenuti nell'oggetto *Dataset*.

Dato che viene utilizzato il servizio IndexedDB_G dei browser, il gruppo ha ritenuto che questo non facesse parte di nessuno dei tre componenti sopra descritti, e quindi ha deciso di separarlo mettendolo in *Services*. È comunque il controller che si occupa di gestirlo.

È stato scelto il design pattern MVC per i seguenti motivi:

- Favorisce la separazione tra $\textit{Business Logic}_G$ e $\textit{Presentation Logic}_G$, facendo comunicare modello e vista solo attraverso il controller;
- È adatto per le applicazioni che prevedono una *GUI* per l'interazione con l'utente.

6.2 Diagrammi delle classi

6.2.1 Model

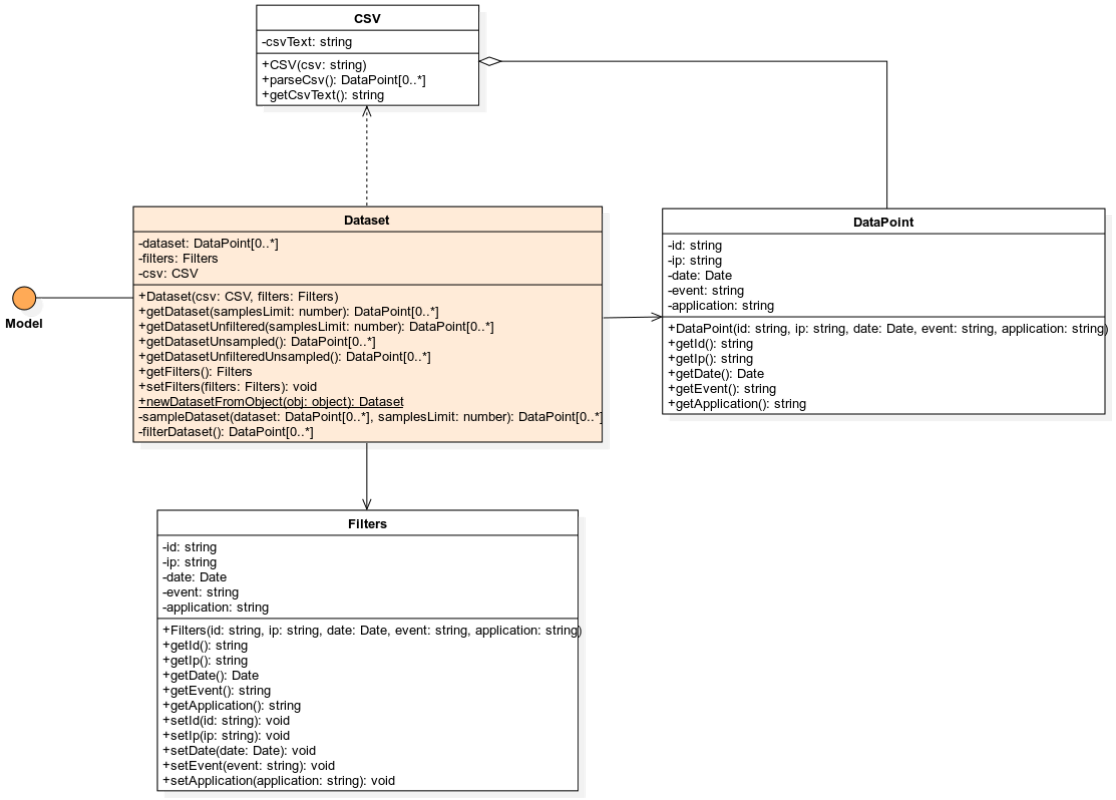


Figura 6.1: Diagramma delle classi riguardanti il Model

La funzione del modello è separare la logica dei dati dall'interfaccia.
Il diagramma delle classi del Model è costituito dall'interfaccia `Model` e dalle classi concrete `Dataset`, `Filters`, `DataPoint` e `CSV`. Nel dettaglio la funzione delle varie componenti del model è:

- `Filters`: è la classe che permette la gestione dei filtri applicati al grafico. Presenta dei metodi "get" e "set" per ogni campo dati presente, che permettono di recuperare oppure salvare i filtri applicati al grafico;
- `DataPoint`: è la classe che permette di salvare al suo interno le informazioni ottenute dalle tuple del file ".csv";
- `CSV`: è la classe che permette di salvare le informazioni presenti nel file ".csv" caricato, con il formato di array di `DataPoint`;
- `Dataset`: è la classe più importante del Modello in quanto, oltre a salvare al suo interno tutti i filtri applicati ai grafici, salva e gestisce tutte le informazioni lette dal file ".csv" caricato.
  La classe `Dataset` mette a disposizione differenti metodi che permettono di

ottenere e salvare i filtri, salvare le informazioni dei file ".csv" e recuperare tali informazioni con delle varianti (come privarle o meno di filtri e campionature).

6.2.2 View

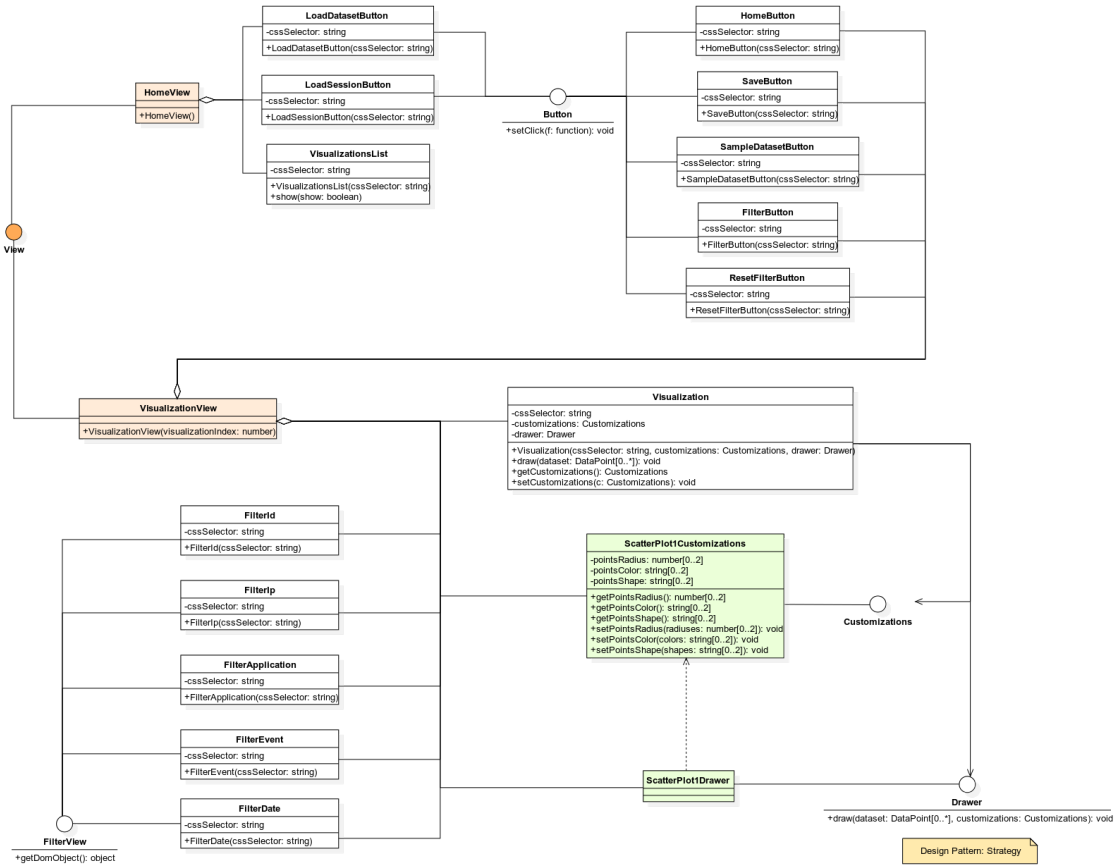


Figura 6.2: Diagramma delle classi riguardanti la View

Il diagramma delle classi della vista è diviso principalmente in due parti: la classe `HomeView` e `VisualizationView` che si occupano di creare tutti gli elementi che compongono le rispettive viste e implementano un'interfaccia comune `View`.

Nella parte superiore del grafico si può notare che tutti i bottoni presenti nell'applicazione implementano l'interfaccia `Button` che mette a disposizione il metodo `setClick(f: function)` il quale conterrà l'EventListener che verrà ridefinito da ogni bottone in base al suo compito.

In basso a sinistra si vede che i vari filtri impostabili implementano l'interfaccia `FilterView` che mette a disposizione il metodo `getDomObject()` il quale semplicemente restituisce l'elemento della DOM.

Come detto prima, la classe `VisualizationView` crea la classe `Visualization`, che si occupa di generare il grafico selezionato nella schermata home tramite il metodo `draw(dataset: Dataset)`, essa inoltre contiene i metodi che gestiscono le personalizzazioni dei grafici: `getCustomizations()` e `setCustomizations(c: Customizations)`.

Questa classe ha un riferimento alle interfacce `Drawer` e `Customizations`, dalle quali viene implementata una classe per ognuna possibile visualizzazione. In questo diagramma vengono inserite solo le classi Drawer e Customizations relative alla visualizzazione dello Scatter Plot$_G$ numero 1 per renderlo più leggibile, ma come detto ogni visualizzazione ha le sue.

6.2.3 Controller

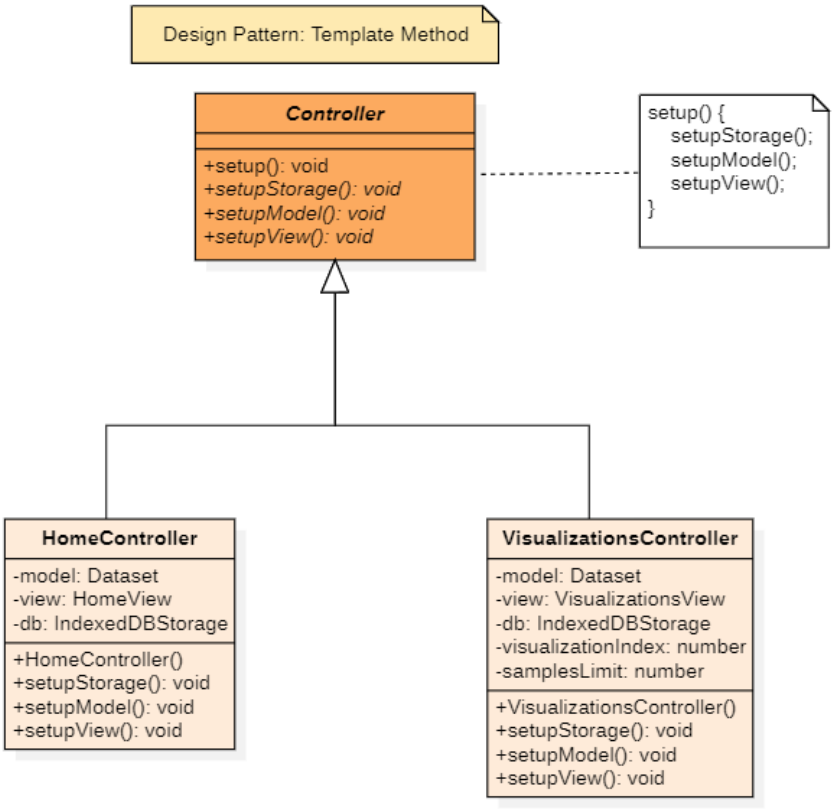


Figura 6.3: Diagramma delle classi riguardanti il Controller

Il controller agisce da intermediario tra vista e modello. Ha la funzione di soddisfare le richieste poste da parte dell'utente modificando le altre due componenti. Possiamo notare la presenza di una classe astratta chiamata `Controller` che definisce i valori e i metodi che le istanze dovranno presentare.

La scelta di usare due controller nell'applicazione è stata fatta per riuscire a gestire le pagine, con differenti funzionalità, in modo semplice e modulare permettendo una più semplice estendibilità e manutenibilità$_G$.
`HomeController` e `VisualizationsController` sono gli oggetti istanziabili con superclasse `Controller` e sono utilizzati nel seguente modo:

- `HomeController`: Ha la funzione di gestire l'interazione Model-View nella scheda iniziale;
- `VisualizationsController`: Ha la funzione di gestire l'interazione Model-View nella scheda in cui vengono visualizzati i grafici con i vari filtri.

Entrambe le classi precedentemente citate hanno il metodo comune `setup()` che permette l'inizializzazione della classe andando a chiamare altri tre metodi: `setupStorage()` che genera l'istanza della classe `IndexedDB`, `setupModel()` che istanzia il Modello e `setupView()` che crea la View. Completato il processo di inizializzazione l'applicazione sarà pronta a rispondere alle interazioni con l'utente.

6.3 Diagrammi di sequenza

6.3.1 Caricamento dataset

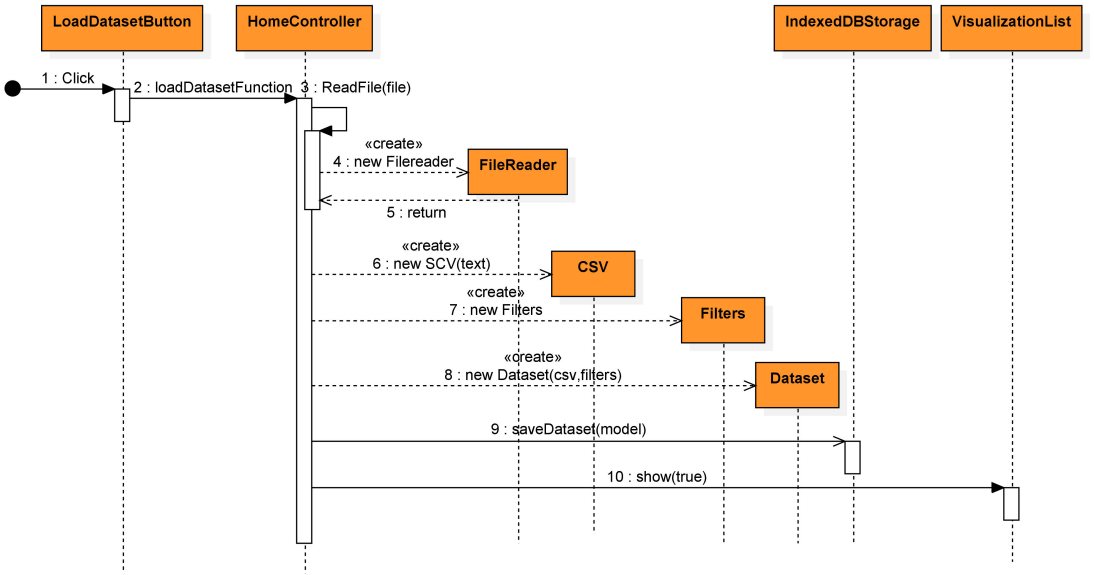


Figura 6.4: Diagramma di sequenza per il caricamento del dataset

La sequenza di azioni che portano al caricamento del dataset e alla conseguente visualizzazione dei vari grafici disponibili sono innescate dal "click" effettuato dall'utente sull'oggetto `LoadDatasetButton`.

Una volta cliccato `LoadDatasetButton` emetterà un segnale recepito, tramite gli EventListener, dalla classe `HomeController` che eseguirà la funzione `loadDatasetFunction()`. Tale funzione chiamerà a sua volta un'altra funzione appartenente alla classe `HomeController` chiamata `ReadFile()`. `ReadFile()` andrà semplicemente a creare un oggetto `FileReader` a cui verrà fatto leggere, sotto forma di testo, il dataset passato dall'utente.

Una volta terminata l'esecuzione della funzione `ReadFile()` continuerà l'esecuzione della funzione `LoadDatasetFunction()` andando a creare tre oggetti: `CSV` a cui verrà passato come parametro attuale ciò che è stato letto da `FileReader`, `Filters` inizializzato con parametri `null` e `Dataset` a cui vengono passati come parametri attuali gli oggetti precedentemente creati `CSV` e `FileReader`. Il passo successivo compiuto dalla funzione è chiamare il metodo `saveDataset()` dell'oggetto `IndexedDBStorage` a cui viene passato l'oggetto creato precedentemente `Dataset`

con lo scopo di salvarlo nel database esterno all'applicazione. Come ultimo passo viene chiamata la funzione `show()` dell'oggetto `VisualizationList` che permetterà la visualizzazione delle configurazioni dei grafici disponibili all'interno della HomePage.

6.3.2 Nuovo campionamento

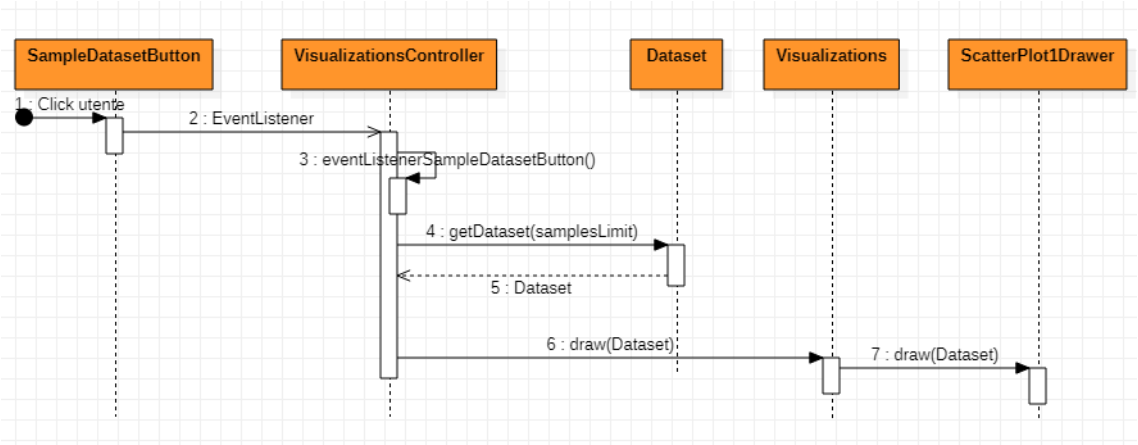


Figura 6.5: Diagramma di sequenza per nuovo campionamento nel grafico *ScatterPlot01*

La sequenza di azioni che portano ad eseguire un nuovo campionamento$_G$ e alla conseguente visualizzazione del grafico aggiornato (in questo caso *ScatterPlot01*)sono innescate dal "click" effettuato dall'utente sull'oggetto `SampleDatasetButton`.
Una volta cliccato `SampleDatasetButton` emetterà un segnale recepito, tramite gli EventListener, dalla classe `VisualizationsController` che richiamerà la classe `Dataset` (ovvero il modello) tramite il metodo `getDataset(samplesLimit)` il quale ritornerà un nuovo insieme di oggetti `DataPoint` contenuti appunto in `Dataset`. A questo punto il controller richiamerà la funzione `draw(Dataset)` sulla classe *ScatterPlot01Drawer* che genererà il nuovo grafico.

6.4 Design pattern utilizzati

6.4.1 Strategy

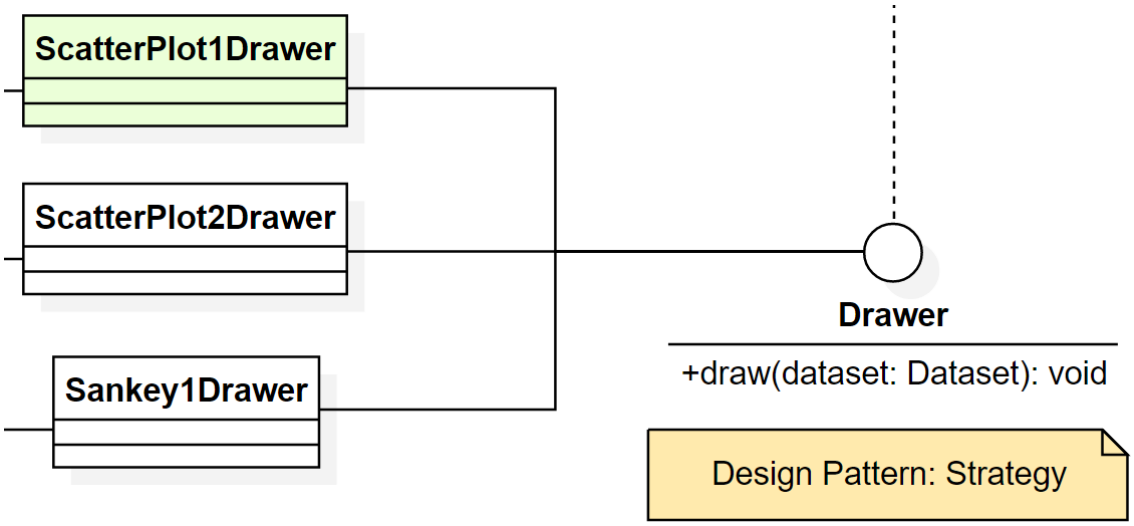


Figura 6.6: Diagramma Strategy pattern

Il nostro gruppo ha scelto di utilizzare il pattern *Strategy*$_G$ per la generazione dei grafici in quanto ci serviva un algoritmo il cui fine era il medesimo, ma che sfruttasse passi differenti per la rappresentazione delle differenti viste.

Come è possibile notare dall'immagine lo `Strategy` presenta un'interfaccia chiamata `Drawer` la quale dichiara un metodo `draw(dataset:Dataset)` che sarà definito in maniera differente dalle classi che implementeranno l'interfaccia, in questo modo sarà possibile generare grafici differenti.

Dall'immagine possiamo notare solamente tre differenti classi che implementano "Drawer" che sono `ScatterPlot1`, `ScatterPlot2` e `Sankey1Diagram`; questo è stato fatto per una questione di semplicità espositiva, in realtà ogni configurazione dei vari grafici avrà la propria classe specifica per la rappresentazione del dataset.

6.4.2 Template method

Il gruppo ha deciso di utilizzare il *Template Method*$_G$ come design pattern comportamentale nel *Controller* (per il diagramma vedi 6.2.3).
Il motivo è che i due controller implementano un algoritmo `setup()` che ha un flusso di esecuzione comune ad entrambi: in ordine vengono eseguiti `setupStorage()`, `setupModel()` e `setupView()` che rispettivamente si occupano di creare il database, il modello e la vista.
Ovviamente questi metodi avranno un comportamento diverso a seconda del controller in cui si trovano, ad esempio in `HomeController` il modello viene impostato dopo che l'utente ha caricato un file mentre in `VisualizationsController` il modello viene preso dal database del browser.

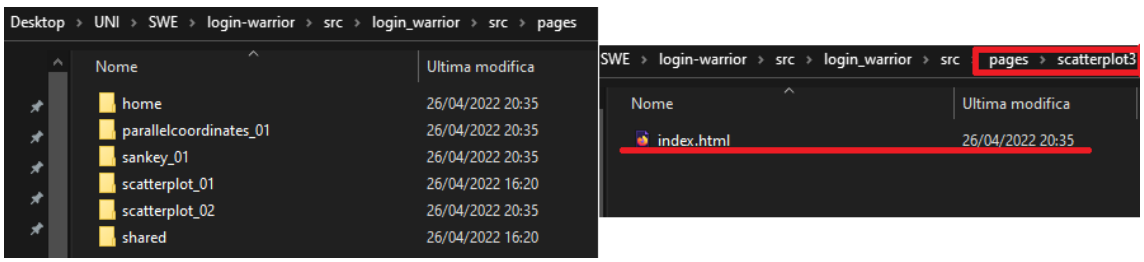
7. Principali punti di estensione

Il prodotto *Login-warrior* realizzato dal gruppo è pensato in modo da poter attuare estesioni e manutenzioni in futuro. La manutenzione del codice è favorita dall'utilizzo di software di analisi statica, come ESLint, che permettono di mantenere il codice privo di errori e uno stile di codifica coerente tra i vari file.

7.1 Aggiunta di un nuovo grafico

Il prodotto permette di visualizzare diverse tipologie di grafici generati dai dati caricati, ed è possibile aggiungerne di nuove:

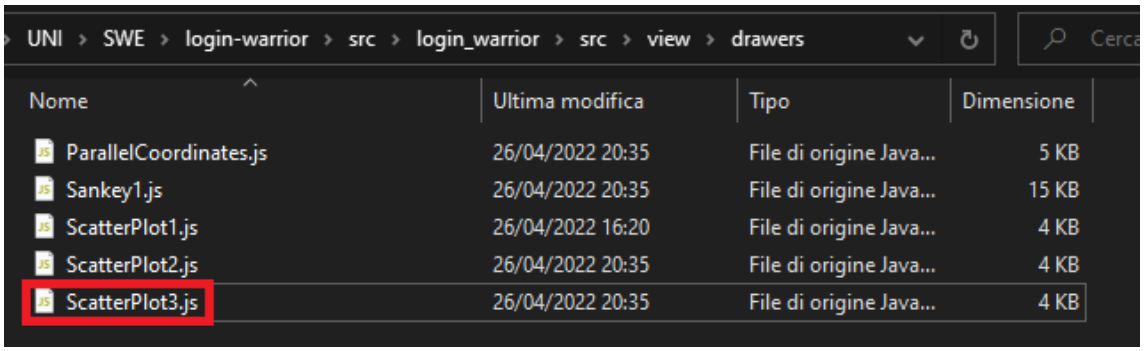
1. Dalla directory principale spostarsi nella cartella in cui ci sono le varie pagine dei grafici seguendo il perorso `src\login_warrior\src\pages`. In questa posizione creare e spostarsi all'interno di una nuova cartella (e.g. "ScatterPlot3"), creare un file *index.html* uguale a quello delle altre pagine a cui però si deve cambiare il titolo. Questa operazione serve per creare la struttura base per la visualizzazione del nuovo grafico;



2. Modificare il file *index.html* presente nella cartella "home", raggiungibile seguendo il percorso `src\login_warrior\src\pages` dalla directory principale del progetto, aggiungendo un elemento "#plot-entry" alla lista "#plot-list", controllando di indicare il percorso corretto al nuovo grafico in ".plot-link". Questo passo permette di vedere la scelta del nuovo grafico inserito all'interno della lista di grafici disponibili nella homepage;

```
<article class="plot-entry">
  <h2>Scatter Plot 3</h2>
    <p>Inserimento nuovo grafico</p>
    <div class="entry-info">
      <p><strong>X</strong>: utente</p>
      <p><strong>Y</strong>: ora</p>
      <p><strong>Colore</strong>: tipo di evento (login, errore)</p>
      <p><strong>Opacità</strong>: login 50% ed errore 100%</p>
      <p><strong>Testo (on hover)</strong>: informazioni (utente, ip, applicazione)</p>
    </div>
    <div class="plot-link-container">
      <a class="plot-link" href="../scatterplot3/">Visualizza</a>
    </div>
</article>
```

3. Spostarsi nella cartella "drawers" seguendo il percorso `src\login_warrior\src\view\drawers` dalla directory principale, creare una nuova classe che implementa l'interfaccia Drawer (e.g. ScatterPlot3). Questo passo permette di inserire i metodi necessari alla conversione del dataset in grafico;



4. Modificare il file "VisualizationView.js" seguendo il percorso `src\login_warrior\src\view` dalla cartella principale, nello specifico modificare lo switch del costruttore, aggiungendo un nuovo caso visualizationIndex (e.g. 5) e inizializzando la visualizzazione con il nuovo "Drawer"(e.g. ScatterPlot3). In questa fase è di fondamentale importanza importare il file presente nella cartella "drawers" creato nel punto precedente (e.g. `import ScatterPlot3 from ".\drawers\ScatterPlot3.js"`). Con questo passo si permette la visualizzazione del nuovo grafico nella lista dei grafici disponibili;

```
constructor(visualizationIndex) {
  switch (visualizationIndex) {
    case 1:
      this.visualization = new Visualization('#visualization', null, new ScatterPlot1());
      break;
    case 2:
      this.visualization = new Visualization('#visualization', null, new ScatterPlot2());
      break;
    case 3:
      this.visualization = new Visualization('#visualization', null, new ParallelCoordinates());
      break;
    case 4:
      this.visualization = new Visualization('#visualization', null, new Sankey1());
      break;
    case 5:
      this.visualization = new Visualization('#visualization', null, new ScatterPlot3());

    default:
      break;
  }
```

```
sers > emanu > Desktop > UNI > SWE > login-warrior > src > login_warrior > src > view >
import ScatterPlot1 from './drawers/ScatterPlot1.js';
import Sankey1 from './drawers/Sankey1.js';
import ParallelCoordinates from './drawers/ParallelCoordinates.js';
import ScatterPlot2 from './drawers/ScatterPlot2.js';
import ScatterPlot3 from "./drawers/ScatterPlot3.js";
```

5. Modificare il file "VisualizationsController.js" presente in `src\login_warrior\src\controller` aggiornando il metodo "viewsInfo()" aggiungendo un caso allo switch con il nome della pagina della nuova visualizzazione (e.g. scatterplot3) e il rispettivo visualizationIndex specificato nel punto precedente (e.g. 5, che è stato definito in `src\login_warrior\src\view \VisualizationView.js`). Come ultimo passo è necessario scegliere un numero massimo di punti per il campionamento andando a definire la variabile "samplesLimit". Con questo passo si va ad aggiornare il controller delle pagine che visualizzano i grafici, definendo le caratteristiche della nuova visualizzazione.

```
viewsInfo() {
  switch (this.getVisualizationName()) {
    case 'scatterplot_01':
      this.samplesLimit = 1000;
      this.visualizationIndex = 1;
      break;
    case 'scatterplot_02':
      this.samplesLimit = 1500;
      this.visualizationIndex = 2;
      break;
    case 'parallelcoordinates_01':
      this.samplesLimit = 1200;
      this.visualizationIndex = 3;
      break;
    case 'sankey_01':
      this.samplesLimit = 200;
      this.visualizationIndex = 4;
      break;

    case 'scatterplot3':
      this.samplesLimit = 10;
      this.visualizationIndex = 5;
      break;

    default:
      window.location.href = '../home';
      break;
  }
}
```

6. Come punto finale, se l'applicazione è stata utilizzata almeno una volta prima delle modifiche sopra riportate, pulire la cache del Browser utilizzato.

Figura 7.1: Risultato dell'aggiunta del grafico