

CODEBUSTERS

Progetto: *HD Viz*
codebusterswe@gmail.com

Manuale Sviluppatore

Informazioni sul documento

Versione	1.0.0-1.7
Approvatori	Rago Alessandro Pirolo Alessandro
Redattori	Zenere Marco Rago Alessandro Safdari Hossain
Verificatori	Sassaro Giacomo Scialpi Paolo Baldisseri Michele
Uso	Interno <i>Zucchetti</i>
Distribuzione	Prof. Vardanega Tullio Prof. Cardin Riccardo Gruppo <i>CodeBusters</i>

Descrizione

Questo documento racchiude tutte le informazioni necessarie per l'estensione e la manutenzione del prodotto *HD Viz*.

Registro delle modifiche

Versione	Data	Nominativo	Ruolo	Descrizione
1.0.0-1.7	23-04-2021	Rago Alessandro	Responsabile	Approvazione del documento
0.2.0-1.7	20-04-2021	Scialpi Paolo	Verificatore	Verifica complessiva del documento
0.1.4-1.7	18-04-2021	Zenere Marco, Sassaro Giacomo	Programmatore, Verificatore	Stesura della sezione §A e verifica
0.1.3-1.7	17-04-2021	Pirolo Alessandro, Baldisseri Michele	Programmatore, Verificatore	Stesura delle sezioni §6, §7 e verifica
0.1.2-1.7	16-04-2021	Safdari Hossain, Sassaro Giacomo	Programmatore, Verificatore	Stesura della sezione §5.3, §5.4 e verifica
0.1.1-1.7	16-04-2021	Zenere Marco, Scialpi Paolo	Programmatore, Verificatore	Stesura delle sezioni §5.1, §5.2 e verifica
0.1.0-0.5	21-03-2021	Sassaro Giacomo	Verificatore	Verifica complessiva del documento
0.0.4-0.4	20-03-2021	Pirolo Alessandro, Baldisseri Michele	Programmatore, Verificatore	Stesura della sezione §4 e verifica
0.0.3-0.3	16-03-2021	Rago Alessandro, Sassaro Giacomo	Programmatore, Verificatore	Stesura delle sezioni §2, §3 e verifica
0.0.2-0.3	16-03-2021	Safdari Hossain, Baldisseri Michele	Amministratore, Verificatore	Stesura della sezione §1 e verifica
0.0.1-0.3	13-03-2021	Rago Alessandro, Scialpi Paolo	Amministratore, Verificatore	Creazione scheletro documento

Indice

1	Introduzione	5
1.1	Scopo del documento	5
1.2	Scopo del prodotto	5
1.3	Glossario	5
1.4	Riferimenti	5
1.4.1	Riferimenti normativi	5
1.4.2	Riferimenti informativi	5
1.4.3	Riferimenti legali	6
2	Tecnologie coinvolte	7
3	Configurazione	8
3.1	Requisiti di sistema	8
3.2	Requisiti hardware	8
3.3	Browser	9
4	Installazione	10
4.1	Clonare la repository	10
4.2	Importare il database	10
4.3	Avviare il server	10
4.4	Avviare la web app	11
5	Strumenti per l'analisi e l'integrazione del codice	12
6	Architettura	13
6.1	Diagrammi dei package	14
6.1.1	Client side	14
6.1.2	Connessione tra client e server side	15
6.2	Diagrammi delle classi	16
6.2.1	View	16
6.2.2	Model	17
6.3	Diagrammi di sequenza	18
6.4	Architettura di dettaglio: Strategy pattern	21
7	Server Side	22
8	Principali punti di estensione	23
8.1	Riduzione dimensionale	23
8.2	Calcolo della distanza	23
8.3	Visualizzazione dei dati	24
A	Glossario	26

Elenco delle tabelle

1	Tecnologie coinvolte	8
2	Requisiti di sistema	8
3	Requisiti hardware	8
4	Prestazioni ottimali	9
5	Browser e versioni compatibili	9
6	Strumenti per l'analisi e l'integrazione del codice	12

Elenco delle figure

1	Avvio dell'applicazione	11
2	Model-View-ViewModel di <i>HDViz</i>	13
3	Diagramma dei package client side	14
4	Connessione tra client side e server side	15
5	Diagramma delle classi della vista	16
6	Diagramma delle classi del modello	17
7	Diagramma di sequenza che modella il processo di riduzione dimensionale con algoritmo LLE	18
8	Diagramma di sequenza che modella il processo di visualizzazione del grafico PLMA e del relativo box di preferenze	19
9	Strategy pattern per la scelta dell'algoritmo di riduzione dimensionale	21
10	Credenziali di default	22
11	Oggetto contenente i diversi tipi di distanza	23
12	Codice per l'input della funzione di calcolo della distanza	23
13	Oggetto contenente i diversi tipi di visualizzazione	24
14	Switch per le voci del menu dedicate ai grafici	24
15	Switch per il rendering delle preferenze	25
16	Switch per il rendering del grafico	25

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 mantenere il prodotto *HD Viz*. Di seguito lo sviluppatore troverà nel documento 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

Oggigiorno anche i programmi più tradizionali gestiscono e memorizzano una grande mole di dati; di conseguenza servono software in grado di eseguire un'analisi e un'interpretazione delle informazioni. Il prodotto *HD Viz* ha come obiettivo quello di creare un'applicazione di visualizzazione di dati con numerose dimensioni in modo da renderle comprensibili all'occhio umano. Lo scopo del prodotto sarà quello di fornire all'utente diversi tipi di visualizzazioni e di algoritmi per la riduzione dimensionale in modo che, attraverso un processo esplorativo, l'utilizzatore del prodotto possa studiare tali dati ed evidenziarne degli eventuali cluster.

1.3 Glossario

Per evitare ambiguità relative alle terminologie utilizzate, queste verranno evidenziati da una 'G' ad apice e riportate nel glossario presente nell'appendice §A.

1.4 Riferimenti

1.4.1 Riferimenti normativi

- Capitolato d'appalto C4 - HD Viz: visualizzazione di dati multidimensionali:
<https://www.math.unipd.it/~tullio/IS-1/2020/Progetto/C4.pdf>

1.4.2 Riferimenti informativi

- Slide E1 del corso di Ingegneria del Software - Diagrammi delle classi e dei package
https://www.math.unipd.it/~rcardin/swea/2021/Diagrammi%20delle%20Classi_4x4.pdf

- Proprietà, slide 11 - 15;
- Operazioni, slide 15 - 17;
- Relazioni di dipendenza, slide 20 - 24.

https://www.math.unipd.it/~rcardin/swea/2021/Diagrammi%20dei%20Package_4x4.pdf

- Dipendenze, slide 6 - 15.

- **Slide E2 del corso di Ingegneria del Software - Diagrammi di sequenza**
https://www.math.unipd.it/~rcardin/swea/2021/Diagrammi%20di%20Sequenza_4x4.pdf
 - Messaggi, slide 9 - 16;
 - Cicli e condizione, slide 19 - 21.
- **Slide E7 del corso di Ingegneria del Software - Principi di programmazione SOLID**
https://www.math.unipd.it/~rcardin/swea/2021/SOLID%20Principles%20of%20Object-Oriented%20Design_4x4.pdf
 - Single responsibility principle, slide 5 - 9;
 - Open-closed principle, slide 11 - 17.
- **Slide E10 del corso di Ingegneria del Software - Design pattern comportamentali**
https://www.math.unipd.it/~rcardin/swea/2021/Design%20Pattern%20Comportamentali_4x4.pdf
 - Observer Pattern, slide 22-31;
 - Strategy Pattern, slide 32-40.
- **Slide L02 del corso di Ingegneria del Software - Design pattern architetturali: Model View Controller e derivati**
<https://www.math.unipd.it/~rcardin/sweb/2020/L02.pdf>
 - MVVM, slide 29-40.
- **API libreria per la visualizzazione dei grafici:**
<https://github.com/d3/d3/blob/master/API.md>
- **Libreria per gli algoritmi di riduzione dimensionale:**
<https://github.com/saehm/DruidJS>
- **Libreria per l'implementazione dell'observer pattern:**
<https://mobx.js.org/README.html>
- **Node.js API:**
<https://nodejs.org/dist/latest-v16.x/docs/api/>
- **PostgreSQL API:**
<https://www.postgresql.org/about/news/postgresql-restful-api-1616/>

1.4.3 Riferimenti legali

- **Licenza MIT:**
<https://opensource.org/licenses/MIT>

2 Tecnologie coinvolte

Tecnologia	Versione	Descrizione
Linguaggi		
JavaScript	ES6 ^G	Utilizzato per la creazione di effetti dinamici e interattivi tramite eventi invocati dall'utente. È il linguaggio della libreria React ^G .
HTML	5	Utilizzato nel progetto assieme a React per definire l'interfaccia grafica.
CSS	3	Utilizzato per definire la formattazione dei documenti HTML5 e lo stile.
Strumenti		
PostgreSQL	13.x	DBMS ^G ad oggetti utilizzato nel progetto per il caricamento nell'applicazione di basi di dati preesistenti.
Npm	7.x	Gestore di pacchetti utilizzato per effettuare le operazioni di build del codice.
Babel	7.11.0	Transcompiler JavaScript utilizzato per convertire il codice ECMAScript 2015+ in una versione retro compatibile per browser non aggiornati.
Librerie e framework		
React	17.0.1	Libreria per la creazione di UI scelta per facilitare lo sviluppo del front-end e avere performance migliori grazie al suo metodo di renderizzazione dei componenti grafici.
D3.js	6.x	Libreria per creare visualizzazioni dinamiche ed interattive partendo da dati organizzati.
Node.js	14.16.0	Runtime system orientato agli eventi scelto come strumento per l'utilizzo di Javascript lato server.
React Bootstrap	1.5.2	Framework ^G che fornisce componenti React con uno stile già integrato e che permette la creazione di applicazioni web responsive.
MobX	6.1.x	Libreria per React che permette la gestione dello state ^G dei componenti e l'implementazione del design pattern Observer.
Druid.js	0.3.5	Libreria usata per implementare il processo di riduzione dimensionale attraverso algoritmi lineari e non lineari.

Continua nella pagina successiva...

ml-distance.js	3.0.0	Libreria usata per implementare il processo di calcolo delle distanze.
Express	4.17.1	Framework ^G per Node.js che fa da intermediario tra web app e database, agevolandone il collegamento.

Tabella 1: Tecnologie coinvolte

3 Configurazione

In questa sezione vengono trattati in primo luogo i requisiti minimi necessari per l'utilizzo dell'applicazione *HDViz* e successivamente come poter installare il prodotto in locale direttamente dal repository^G pubblico GitHub^G del gruppo *CodeBusters*.

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 avere nella propria macchina i seguenti software.

Software	Versione	Riferimento per il download
Node.js	14.16.x	https://nodejs.org/it/
Npm	7.x	Integrato nel download di Node.js
PostgreSQL	13.x	https://www.postgresql.org/download/

Tabella 2: Requisiti di sistema

3.2 Requisiti hardware

Per avere delle prestazioni accettabili dell'applicazione è preferibile avere almeno i seguenti componenti hardware.

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

Tabella 3: Requisiti hardware

La connessione internet ideale dovrebbe essere di almeno 80Mb/s in download per assicurare i seguenti tempi di risposta (in secondi) della web app.

Tempo	Operazione
2	Caricamento di un dataset di 2Mb;
7	Applicazione di un algoritmo di riduzione dimensionale con 4 dimensioni e 500 punti in totale;
2	Visualizzazione di un grafico con 500 punti.

Tabella 4: Prestazioni ottimali

3.3 Browser

L'applicazione è stata testata e quindi resa compatibile con le ultime versioni dei browser maggiormente utilizzati al momento.

Browser	Versione
Chrome	87
Edge	79
Mozilla Firefox	84
Safari	13.1

Tabella 5: Browser e versioni compatibili

4 Installazione

Per utilizzare l'applicazione web è necessario:

- Clonare la repository (**Obbligatorio**)
- Importare il database (*Opzionale*)
- Avviare il server (*Opzionale*)
- Avviare la web app (**Obbligatorio**)

4.1 Clonare la repository

1. Scaricare il codice come file .zip direttamente dal repository CodeBusters-HDViz:

<https://github.com/CodeBusterswe/CodeBusters-HDViz>

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

git clone https://github.com/CodeBusterswe/CodeBusters-HDViz

2. Localizzare da terminale la cartella in cui si è stato estratto/clonato il prodotto:

cd percorso\CodeBusters-HDViz

4.2 Importare il database

1. Installare PostgreSQL e pgAdmin dal sito :

<https://www.postgresql.org/download/>

2. Creare un nuovo database
3. Fare il restore del database utilizzando il file "dumbDB" presente in "./server"

4.3 Avviare il server

1. Entrare nella cartella "server"

cd server

2. Nel caso di primo avvio, digitare:

npm install

3. Controllare che i valori impostati nel file `"/config/default.js"` corrispondano a quelli impostati in PostgreSQL:
 - `'USERNAME'` : nome utente dell'account;
 - `'PASSWORD'` : password dell'account;
 - `'HOST'` : porta di PostgreSQL;
 - `'DB_NAME'` : nome del db precedentemente creato;
4. Avviare il server con il comando:

`npm start`

4.4 Avviare la web app

1. Entrare nella cartella `"client"`

`cd client`

2. Nel caso di primo avvio digitare il comando:

`npm install`

3. Fare la build dell'app con il comando:

`npm run build`

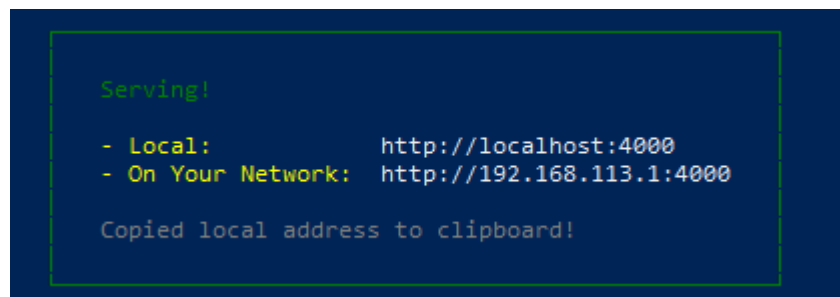
4. Nel caso di primo avvio dell'applicazione sul sistema, digitare:

`npm install -g serve`

5. Digitare poi:

`serve -s build -l 4000`

L'applicazione sarà disponibile aprendo l'indirizzo fornito dal terminale.



```
Serving!
- Local:      http://localhost:4000
- On Your Network: http://192.168.113.1:4000
Copied local address to clipboard!
```

Figura 1: Avvio dell'applicazione

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

Strumento	Versione	Descrizione
Analisi statica		
ESLint	7.15.0	Strumento di analisi statica utilizzato per la segnalazione degli errori di sintassi, per avere regole d'indentazione uguali in tutti i file e per notifiche altri problemi comuni.
Analisi dinamica		
Jest	26.x	Framework ^G di test utilizzato per l'analisi dinamica del codice.
React Testing Library	11.2.x	Libreria usata per testare lo stato interno delle componenti React.
Continuous Integration		
GitHub Actions	-	Strumento fornito da GitHub per l'implementazione della CI/CD all'interno del proprio repository.
SonarCloud	-	Piattaforma per il controllo continuo della qualità e manutenibilità del codice.
CodeCov	-	Piattaforma per il controllo del code coverage.

Tabella 6: Strumenti per l'analisi e l'integrazione del codice

6 Architettura

L'architettura di *HDViz* è basata sul design pattern architetturale^G *Model-View-ViewModel (MVVM)*^G, derivato dal più comune *Model-View-Controller (MVC)*^G. Abbiamo quindi sviluppato un *ViewModel* per ogni componente React della *View* che necessita di interagire con il *Model*, e delegato a tali *View-Model* la modifica del *Model* in base agli input dell'utente. Il modello corrisponde al *RootStore*, che istanzia i tre diversi store utilizzati in *HD Viz*.

È stato scelto il design MVVM per i seguenti motivi:

- Favorisce la separazione tra *business logic*^G e *presentation logic*^G, facendo comunicare *Model* e *View* solo attraverso un *ViewModel*;
- Permette di non avere un unico controller con cui dover gestire tutta l'*application logic*^G. Essa è infatti contenuta nei vari *ViewModel* dei componenti della vista, fornendo diversi vantaggi:
 - Minor numero di conflitti in fase di codifica (non si deve accedere ad uno stesso file dove è contenuta tutta la logica);
 - Performance migliori (viene renderizzato solo il componente che effettivamente subisce modifiche del proprio stato interno^G, gestito anch'esso nel rispettivo *ViewModel*).
- Adatto per le web application la cui interfaccia utente è sviluppata con React.

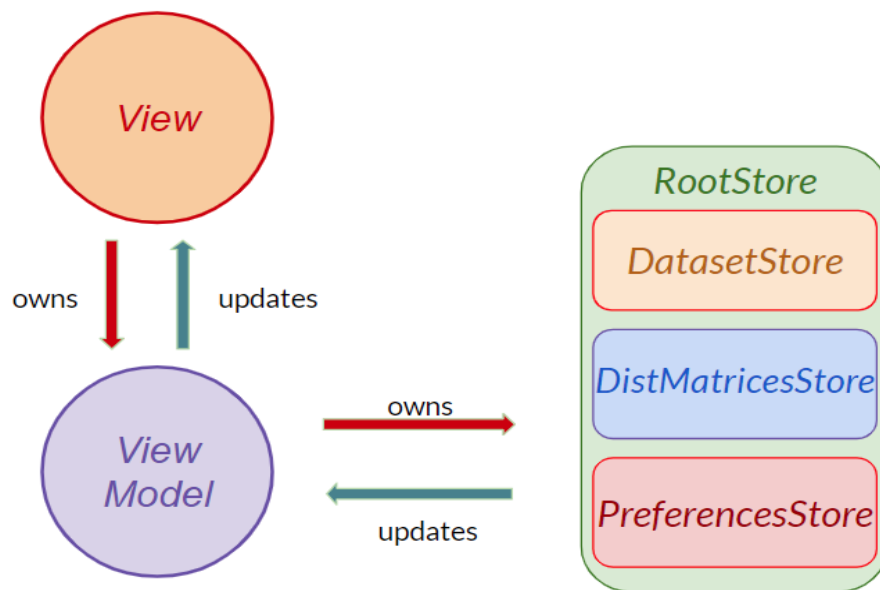


Figura 2: Model-View-ViewModel di *HDViz*

6.1 Diagrammi dei package

6.1.1 Client side

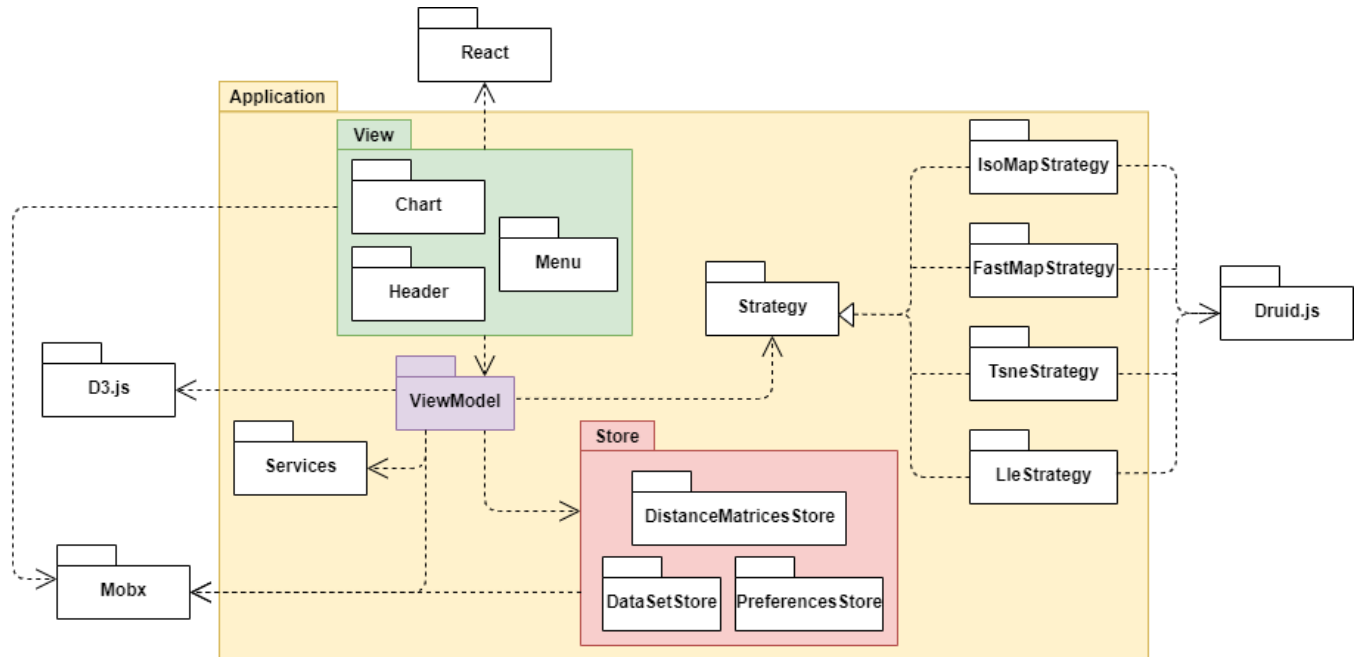


Figura 3: Diagramma dei package client side

Il diagramma sopra riportato descrive a livello di package le parti che compongono il lato client dell'applicazione. In particolare si può notare che:

- *View* e *Store* (ossia il modello) non siano direttamente collegati, ma il passaggio delle informazioni avviene solo attraverso un *ViewModel*, unico per ciascun componente della vista che richieda una connessione allo store;
- MobX ha il ruolo fondamentale di applicare il design pattern Observer^G, ossia permette di rendere i dati contenuti nei vari store observable^G e i componenti della vista che li utilizzano observer^G (per esempio i grafici);
- Druid.js è importata solo nelle classi concrete degli algoritmi implementati (IsoMap, FastMap, Tsne, LLE) per eseguire la riduzione dimensionale e D3.js solo nei *ViewModel* delle varie visualizzazioni (Scatterplot Matrix, HeatMap, Adjacency Matrix, Force Field, PLMA);
- La maggior parte delle dipendenze partono dai diversi *ViewModel* verso le altre parti della web application, lasciando che la *View* si occupi solo di ritornare le componenti che compongono la vista.

6.1.2 Connessione tra client e server side

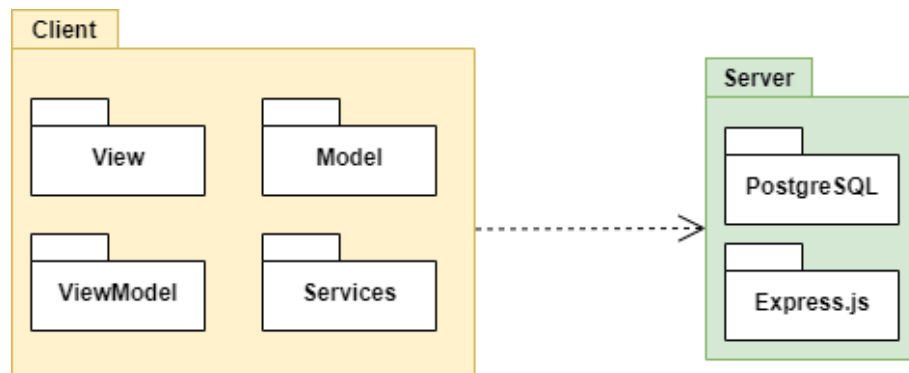


Figura 4: Connessione tra client side e server side

La connessione tra client e server è così gestita:

- Il server fornisce l'interfaccia per l'interrogazione del database attraverso delle API costruite con Express.js;
- Il client sfrutta Axios^G per la connessione al server;
- Grazie ad alcune funzioni richiamate nel *ViewModel* dedicato al caricamento dei dati dal database, vengono inviate le query di selezione al server che si occupa di interrogare il DB, il quale restituisce il risultato dell'interrogazione.

6.2 Diagrammi delle classi

6.2.1 View

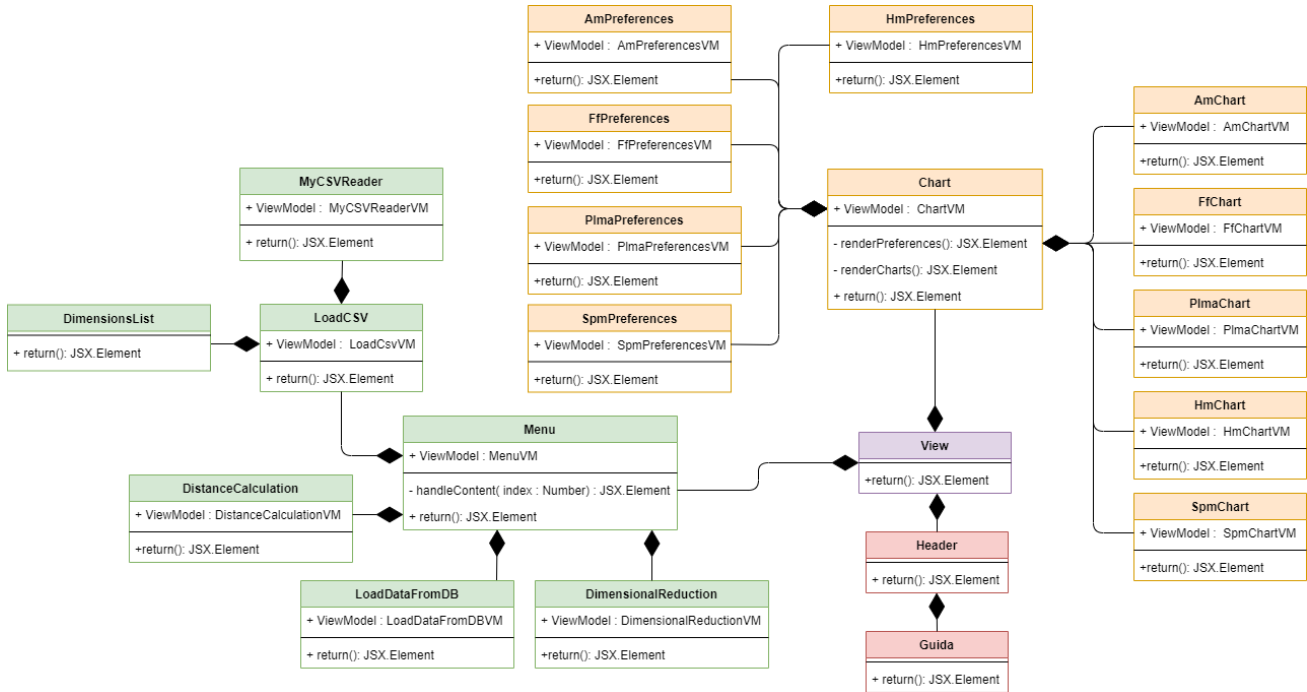


Figura 5: Diagramma delle classi della vista

Il diagramma delle classi della vista è costituito da tutti i componenti React che compongono l'interfaccia utente di *HD Viz*. Visitando dall'alto la gerarchia di componenti, il padre è **View**, il quale crea l'**Header** e i più importanti **Menu** e **Chart**. Riguardo a quest'ultimi due componenti:

- Menu rappresenta per l'utente il punto d'accesso a tutte le funzionalità fornite dall'applicazione. Da qui sono infatti creati i vari modal^G per il caricamento dei dati da file CSV (**LoadCSV**) e dal database (**LoadDataFromDB**), per il calcolo della distanza (**DistanceCalculation**) e per la riduzione dimensionale (**DimensionalReduction**);
- Chart è il componente che crea i grafici e le relative form per settarne le preferenze. La scelta del grafico avviene nel Menu e comprende Scatterplot Matrix, Adjacency Matrix, Force Field, Heatmap e PLMA.

La maggior parte dei componenti hanno un attributo `ViewModel`: questo indica che hanno una dipendenza verso di esso, usata per ottenere le informazioni di cui hanno bisogno contenute negli store (per esempio i dati o le dimensioni), oppure semplicemente come contenitore della propria *presentation logic*.

In questo modo la vista risulta essere completamente separata dalla *business logic* e si occupa solo di ritornare i vari elementi HTML che la costituiscono.

6.2.2 Model

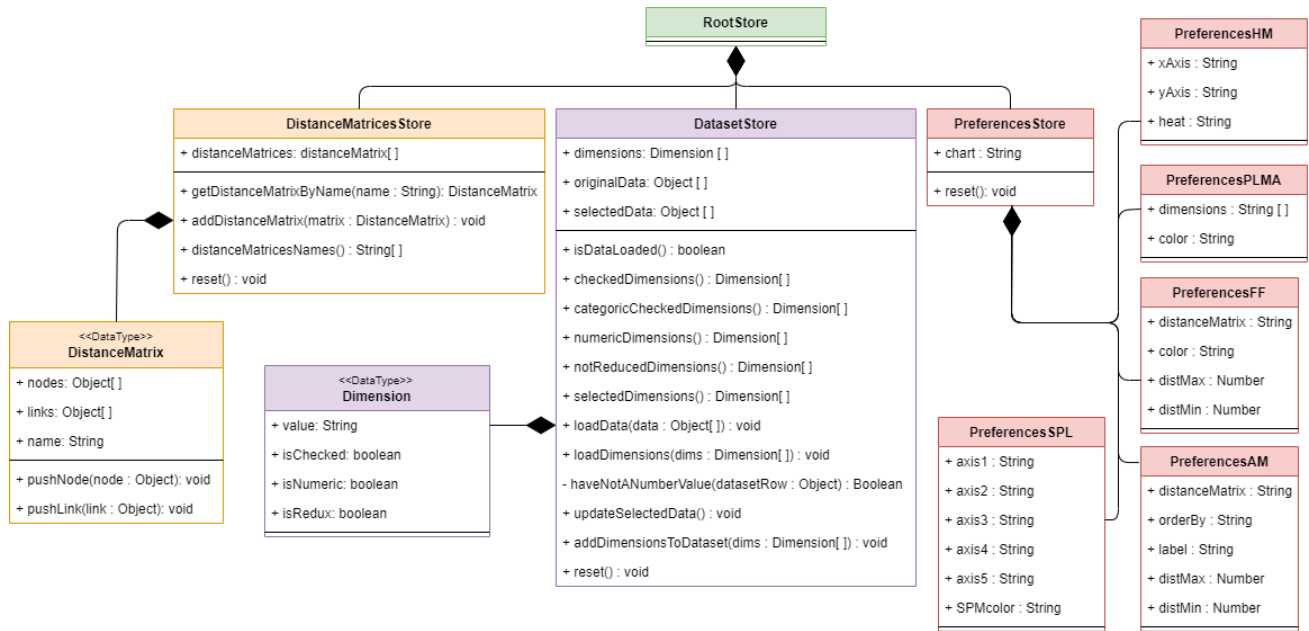


Figura 6: Diagramma delle classi del modello

Il diagramma delle classi del modello è costituito da un **RootStore**, il quale istanzia i tre store utilizzati in *HD Viz*:

- **DatasetStore** che contiene i dati caricati e le dimensioni da visualizzare, fornendo i metodi per recuperarli e aggiornarli;
- **PreferencesStore** che conserva le preferenze di visualizzazione di tutti i grafici utilizzati dall'utente;
- **DistanceMatricesStore** che contiene le matrici delle distanze.

Dimension e **DistanceMatrix** sono i tipi che definiscono rispettivamente le dimensioni e le matrici delle distanze create dall'utente attraverso le varie funzioni di calcolo della distanza fornite (Euclidea, Chebyshev, Manhattan e Canberra).

PreferencesSPM, **PreferencesFF**, **PreferencesAM**, **PreferencesPLMA**, **PreferencesHM** sono i tipi che definiscono le preferenze dei diversi grafici. Gli store contengono gli attributi observable^G che, nel momento in cui vengono modificati, grazie all'utilizzo di MobX, causano la rirenderizzazione dei componenti observer^G della vista.

6.3 Diagrammi di sequenza

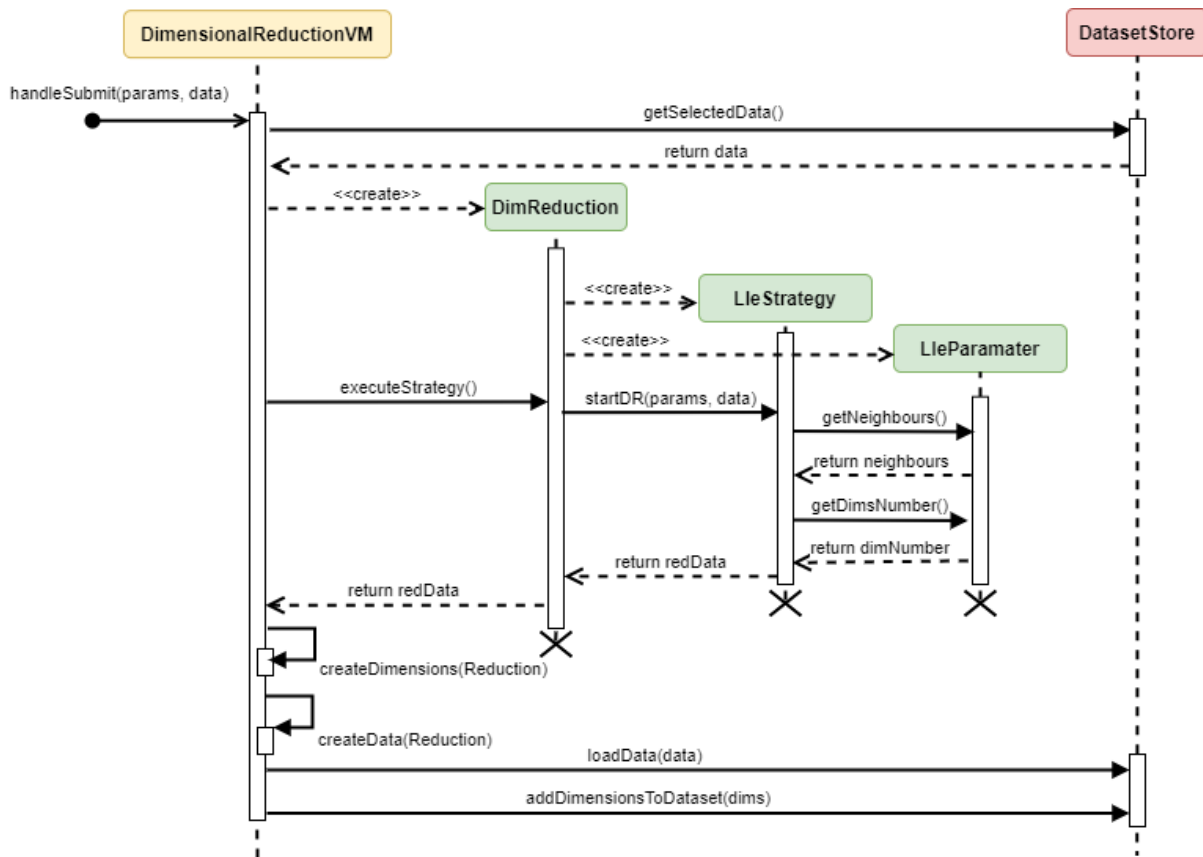


Figura 7: Diagramma di sequenza che modella il processo di riduzione dimensionale con algoritmo LLE

Il diagramma di sequenza sopra riportato è così descritto:

1. **DimensionalReductionVM**, che corrisponde al *ViewModel* della parte di vista dedicata alla riduzione dimensionale, riceve l'algoritmo e i relativi valori dei parametri impostati dall'utente, prende i dati su cui eseguire la riduzione dal **DatasetStore** e crea il contesto, ossia **DimReduction**;
2. **DimReduction** si occupa di creare un'istanza della classe concreta dell'algoritmo scelto dall'utente (nell'esempio LLE, definito in **LLEStrategy**) e un'istanza della classe dei corrispettivi parametri (ossia **LLEParameter**), utilizzando i valori passatigli per settarne i vari attributi;
3. Chiama poi **startDR()** sull'algoritmo, fornendo i dati e l'istanza della classe dei parametri. I valori di quest'ultimi vengono presi dall'algoritmo attraverso dei getter (nell'esempio **getNeighbours()** e **getDimensionsNumber()**) verso l'istanza di **LLEParameter**;
4. **startDR()** a questo punto può eseguire la riduzione dimensionale utilizzando i metodi forniti dalla libreria *Druid.js* e ritornare un oggetto con dentro i nuovi dati e le nuove dimensioni;
5. In **DimensionalReductionVM** vengono estrapolati i dati e le dimensioni dall'oggetto e caricate nel **DatasetStore**.

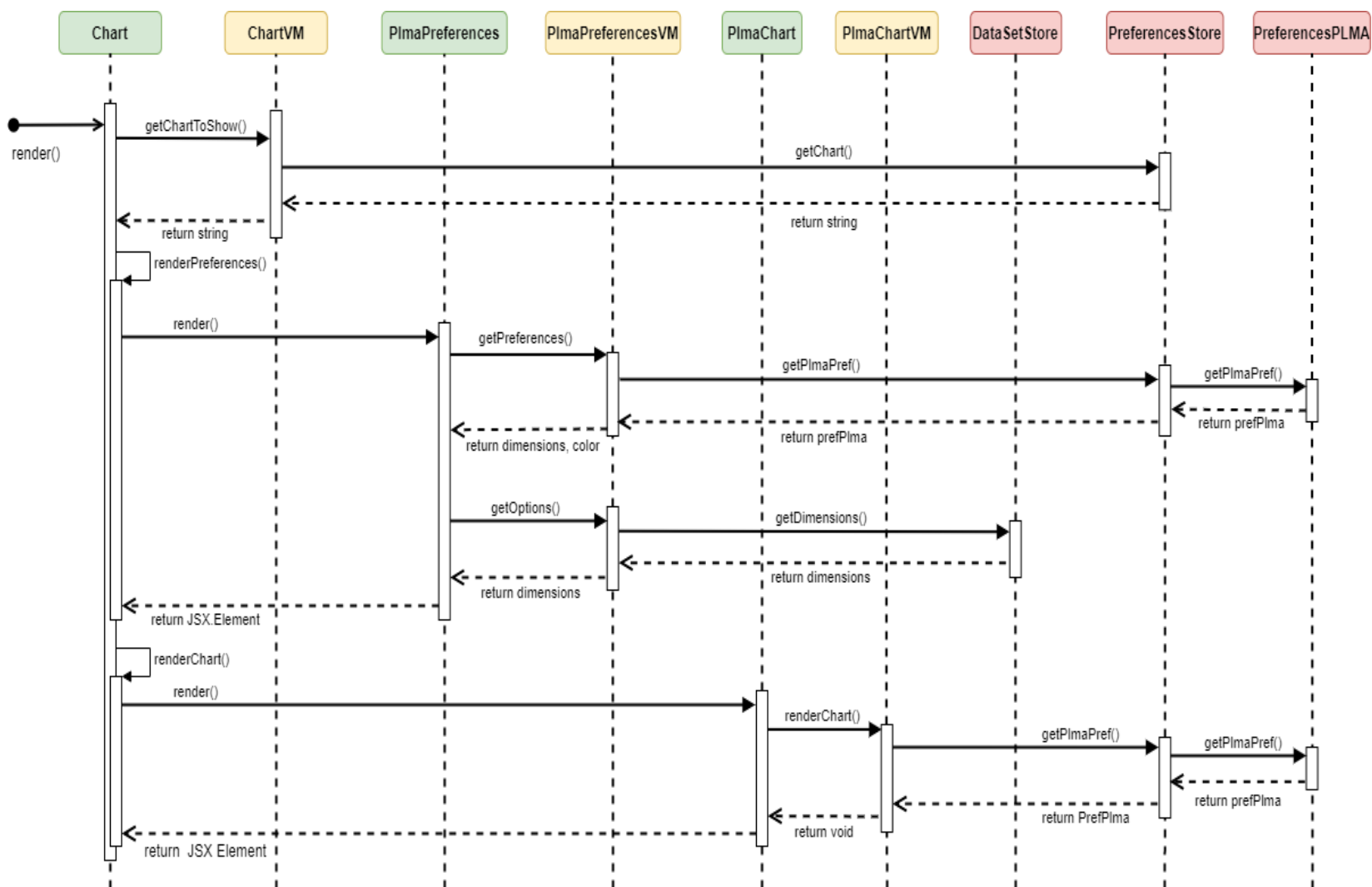


Figura 8: Diagramma di sequenza che modella il processo di visualizzazione del grafico PLMA e del relativo box di preferenze

Il diagramma di sequenza sopra riportato è così descritto:

- 1) **Chart** è il componente React che renderizza la parte di vista contenente il grafico e il relativo box di preferenze. Con `getChartToShow()` e `getChart()` di **ChartVM** (il corrispettivo *ViewModel*) seleziona dal **PreferencesStore** la stringa che definisce quale box visualizzare (in questo esempio "PLMA");
- 2) Sono quindi renderizzate le preferenze (**PlmaPreferences**), ossia la form utilizzabile dall'utente per modificare le caratteristiche di visualizzazione del PLMA. Inizialmente ogni suo campo input è vuoto e quindi non verrà visualizzato il grafico;
- 3) Ogni volta che l'utente modifica tale form avviene una rirenderizzazione del box delle preferenze e del grafico. Tale modifica comporta la chiamata al **PlmaPreferencesVM** (*ViewModel* delle preferenze del PLMA) che si occupa di prelevare dai vari store (**PreferencesStore** e **DatasetStore**) le informazioni necessarie per cambiare in real-time la vista;
- 4) È quindi renderizzato il PLMA (in **PLMAChart**) con le varie modifiche di visualizzazione impostate dall'utente.

6.4 Architettura di dettaglio: Strategy pattern

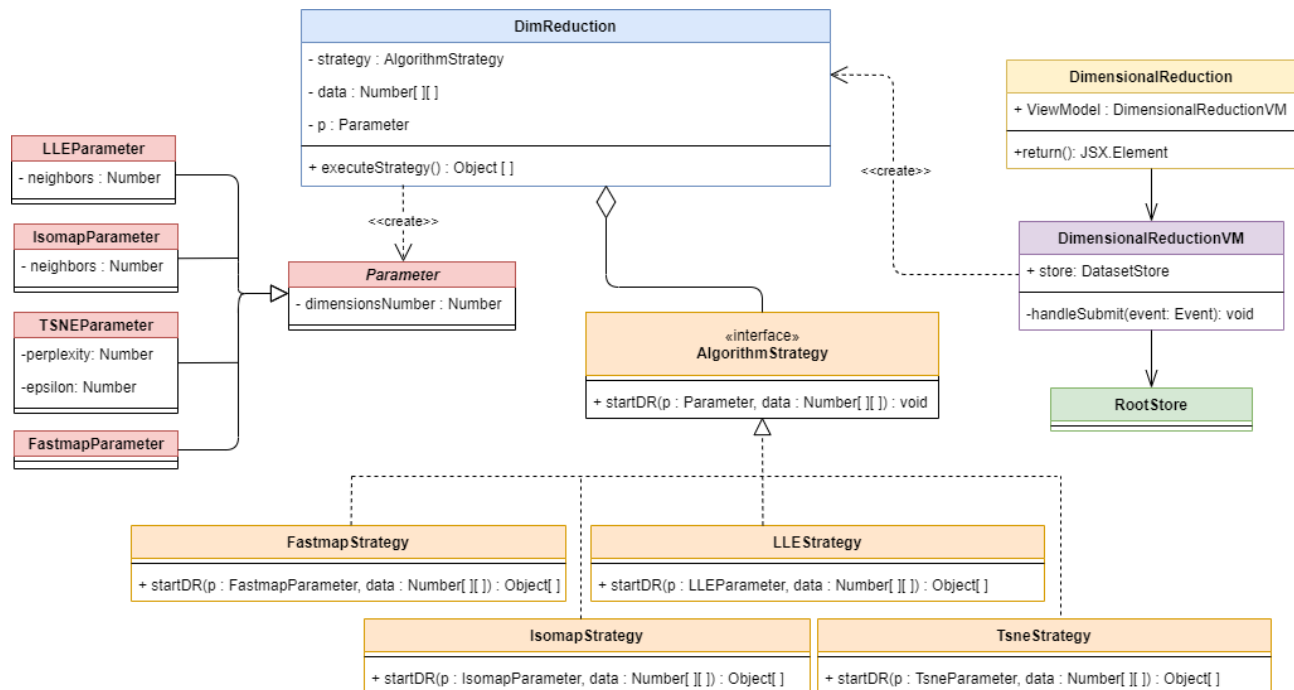


Figura 9: Strategy pattern per la scelta dell'algoritmo di riduzione dimensionale

Nella figura sopra è mostrata la funzionalità di riduzione dimensionale sui dati caricati dall'utente. Per l'implementazione dei vari algoritmi viene utilizzato lo strategy pattern. In particolare:

- **DimensionalReduction** è il componente React che si occupa solo di ritornare gli elementi HTML che compongono questa parte di vista;
- **DimensionalReductionVM** è il rispettivo view-model che ne contiene tutta la logica. Questo prende i dati dal RootStore (nello specifico dal *DatasetStore*) e crea il contesto DimReduction;
- **DimReduction** si occupa di creare i parametri con i valori impostati dall'utente dalla vista. A seconda del valore del suo attributo **strategy** crea la corretta classe di parametri e chiama attraverso **executeStrategy()** il metodo **startDR()** sull'algoritmo scelto;
- **AlgorithmStrategy** è l'interfaccia implementata dalle famiglia di classi concrete degli algoritmi, facilitandone l'estensibilità.

Infine **startDR()** esegue la riduzione dimensionale sui dati utilizzando i metodi forniti dalla libreria Druid.js e ritorna le nuove dimensioni.

7 Server Side

HD Viz è una web application che non richiede un'ampia comunicazione con il lato server. Esso è infatti utilizzato solo per l'interrogazione del database PostgreSQL, per permettere all'utente di caricare i suoi dati contenuti nel database.

Per questo motivo è stato deciso di non adottare una specifica architettura lato server.

I file dedicati sono contenuti nella cartella **server**, così organizzati:

- **test** è la sottocartella contenente il file **server.test.js** utile a controllare che la connessione con il server avvenga correttamente;
- **config** è la sottocartella contenente i file per la configurazione del database. All'interno del file **default.js** è infatti possibile cambiare i dati di accesso al DB con le proprie credenziali (figura 10)
- **api** è la sottocartella contenente il file **DataSet.js**. Al suo interno sono presenti le API create con Express.js per la comunicazione del lato client dell'applicazione con il server sviluppato con Node.js.

```
module.exports={
  'USER_NAME':'postgres',
  'PASSWORD':'admin',
  'HOST':5400,
  'DB_NAME':'demoDatabase',
  'PORT':5000,
}
```

Figura 10: Credenziali di default

8 Principali punti di estensione

HD Viz è una web app progettata pensando anche ad eventuali operazioni future di manutenzione o estensione del prodotto. Rispetto alla manutenzione del codice essa è favorita da una codifica coerente in tutti i file (grazie all'utilizzo di ESLint per l'analisi statica) e l'aggiunta di commenti ove necessario. Riguardo l'estensibilità dell'applicazione sono stati individuati principalmente tre funzionalità che potranno essere ampliate senza grandi difficoltà.

8.1 Riduzione dimensionale

Come detto in precedenza questa funzionalità è stata implementata con l'utilizzo dello strategy pattern. Questo permette una facile manutenzione delle classi concrete degli algoritmi già presenti e permette l'estensione con altri algoritmi di riduzione dimensionale, senza la necessità che essi siano presenti nella libreria *Druid.js* (attualmente utilizzata a questo scopo). L'implementazione del metodo `startDR()` in un nuovo algoritmo è infatti indipendente dalle altre già preesistenti. È necessario però aggiungere anche una classe per contenere i corrispondenti parametri, la quale estende la classe astratta *Parameter* (fare riferimento al diagramma dello strategy pattern alla sezione §6.4).

8.2 Calcolo della distanza

Il calcolo della distanza risulta facilmente estendibile con nuove funzioni. È infatti sufficiente seguire i seguenti passi:

- 1) Aggiungere i nuovi tipi di distanza nell'oggetto `DistanceType` (riportato in figura 11) presente nel file `utils.js`;
- 2) Aggiungere le corrispondenti `option` all'interno della form di selezione (riportata in figura 12). Essa si trova nel file `DistanceCalculation.js` all'interno del `return()` del componente React.

L'unico vincolo è che, per non dover cambiare nient'altro nel codice, le nuove funzioni da aggiungere siano presenti all'interno della libreria *ml-distance*, utilizzata a tale scopo. Essa ne fornisce molte, identificabili nella documentazione ufficiale:

<https://www.npmjs.com/package/ml-distance>

```
export const DistanceType = {  
  Euclidean: "euclidean",  
  Manhattan: "manhattan",  
  Canberra: "canberra",  
  Chebyshev: "chebyshev"  
};
```

Figura 11: Oggetto contenente i diversi tipi di distanza

```
<Form.Control  
  as="select"  
  custom  
  value={distanceType}  
  onChange={handleChangeDistanceType}>  
  <option value={DistanceType.Euclidean}>EUCLIDEA</option>  
  <option value={DistanceType.Canberra}>CANBERRA</option>  
  <option value={DistanceType.Chebyshev}>CHEBYSHEV</option>  
  <option value={DistanceType.Manhattan}>MANHATTAN</option>  
</Form.Control>
```

Figura 12: Codice per l'input della funzione di calcolo della distanza

8.3 Visualizzazione dei dati

HD Viz fornisce cinque diversi tipi di visualizzazione dei dati caricati. È possibile aggiungerne di nuovi seguendo i seguenti passi:

- 1) Aggiungere i nuovi tipi di visualizzazione nell'oggetto `VisualizationType` (riportato in figura 13) presente nel file `utils.js`;
- 2) Aggiungere nel menu la corrispondente voce per avviare la visualizzazione. In particolare bisogna aggiungere dei "case" nello switch dedicato (riportato in figura 14) che si trova in `MenuVM.js`;
- 3) A questo punto bisogna creare i file per le preferenze e per il grafico stesso. In particolare è necessario aggiungere i file:

- **NuovaVisualizzazionePreferences.js** e il corrispondente *ViewModel* nella cartella al percorso:

`./src/components/UI/graphUI/preferences ;`

- **NuovaVisualizzazione.js** e il corrispondente *ViewModel* nella cartella al percorso:

`./src/components/UI/graphUI/charts .`

- 4) Per mostrare a schermo le nuove preferenze e i nuovi grafici è infine necessario aggiungere i "case" agli switch dedicati alla loro renderizzazione (riportati nelle figure 15 e 16) nel file `Chart.js`.

```
export const VisualizationType = {
  ScatterPlotMatrix: "scatterPlotMatrix",
  HeatMap: "heatMap",
  ForceField: "forceField",
  AdjacencyMatrix: "adjacencyMatrix",
  PLMA: "plma"
};
```

Figura 13: Oggetto contenente i diversi tipi di visualizzazione

```
showChart = index => {
  switch(index) {
    case 5:
      this.preferencesStore.chart = VisualizationType.ScatterPlotMatrix;
      break;
    case 6:
      this.preferencesStore.chart = VisualizationType.AdjacencyMatrix;
      break;
    case 7:
      this.preferencesStore.chart = VisualizationType.HeatMap;
      break;
    case 8:
      this.preferencesStore.chart = VisualizationType.ForceField;
      break;
    case 9:
      this.preferencesStore.chart = VisualizationType.PLMA;
      break;
    default:
  }
}
```

Figura 14: Switch per le voci del menu dedicate ai grafici

```
function renderPreferences(){
  switch(chartToShow){
    case VisualizationType.ScatterPlotMatrix:
      return <ScatterPlotMatrixPreferences/>;
    case VisualizationType.AdjacencyMatrix:
      return <AdjacencyMatrixPreferences/>;
    case VisualizationType.ForceField:
      return <ForceFieldPreferences/>;
    case VisualizationType.HeatMap:
      return <HeatMapPreferences/>;
    case VisualizationType.PLMA:
      return <PlmaPreferences/>;
    default:
      return null;
  }
}
```

Figura 15: Switch per il rendering delle preferenze

```
function renderCharts(){
  switch(chartToShow){
    case VisualizationType.ScatterPlotMatrix:
      return <ScatterPlotMatrix/>;
    case VisualizationType.AdjacencyMatrix:
      return <AdjacencyMatrix/>;
    case VisualizationType.ForceField:
      return <ForceField/>;
    case VisualizationType.HeatMap:
      return <HeatMap/>;
    case VisualizationType.PLMA:
      return <Plma/>;
    default:
      return null;
  }
}
```

Figura 16: Switch per il rendering del grafico

A Glossario

A

Application logic

Rappresenta tutta logica della vista. Ogni elemento grafico con cui l'utente può interagire (un pulsante, un campo input, una checkbox) possiede una logica interna che permette di far eseguire ciò che è stato richiesto dall'interfaccia.

Axios

Axios è una libreria che permette di effettuare chiamate ajax in maniera facile e intuitiva, evitando di utilizzare vanilla JavaScript che non permette il totale controllo sulla gestione di tali chiamate. In *HD Viz* è utilizzata per la connessione client/server.

B

Business logic

Rappresenta la logica che dà il valore al prodotto. Essa costituisce i metodi/funzioni in grado, per esempio, di mettere in comunicazione la web app con il database e quindi di estrapolarne i dati, gestirli, modificarli e restituirli a chi li deve utilizzare.

C

Code Coverage

È la percentuale di linee di codice verificate attraverso test di unità rispetto quelle totali che compongono un prodotto software. Maggiore è tale percentuale, minore è la probabilità di rilevare bug o errori in fase di rilascio.

Context Provider

È il componente fornito da React utilizzato per consumare i dati contenuti nel Context React^G attraverso dei componenti denominati "consumatori".

Context React

È un metodo fornito da React per evitare di dover passare dati (variabili, costanti, funzioni ecc.) da componente padre a componente figlio attraverso le props^G. Permette di inserire all'interno di questo contesto un set di dati utilizzabili ovunque semplicemente chiamando l'hook `useContext()` (da non confondere con variabili globali).

Continuos Integration

È una pratica che si applica in contesti in cui lo sviluppo di un prodotto software avviene attraverso un sistema di controllo versione. L'obiettivo del team nell'applicare questa pratica è essere sempre tutti allineati alla stessa versione e avere in ogni momento un prodotto funzionante.

D

DBMS

Acronimo di Database Management System. È un sistema software che consente la creazione, la manipolazione e l'interrogazione in modo efficiente di un database. Un esempio è PostgreSQL.

E

ES6

Acronimo di ECMAScript 6. È una versione di JavaScript che ha introdotto molti cambiamenti rispetto alla precedente, tra cui arrow functions, le keywords "let" e "const", "spread" e "rest" operator.

F

Framework

Architettura di supporto a un software, che ne facilita l'utilizzo a un programmatore. Solitamente tale architettura è composta da una serie di librerie contenenti classi astratte facilmente implementabili.

G

Git

Software di controllo di versione distribuito utilizzabile da interfaccia a riga di comando, creato da Linus Torvalds nel 2005.

GitHub

Servizio di hosting per sviluppatori. Fornisce uno strumento di controllo versione e permette lo sviluppo distribuito del software.

J

JSX

Sintassi utilizzata in React per unire i linguaggi HTML e JavaScript. Questa permette per esempio di rendere elementi HTML dinamici, applicandoci condizioni tipiche del linguaggio JS.

M

Model-View-Controller(MVC)

Pattern architetturale molto diffuso nello sviluppo di software. Consiste nel dividere il prodotto in tre parti fondamentali: il modello (che contiene la business logic), il controller (che contiene l'application logic) e la vista (che contiene la presentation logic). Il controller si occupa di far comunicare vista e modello, ricevendo gli input dell'utente attraverso la vista e avvisando il modello dei cambiamenti avvenuti. La view in questo caso è in grado di visualizzare direttamente i dati contenuti nel modello.

Model-View-ViewModel(MVVM)

Pattern architetturale derivato dal Model-View-Controller(MVC)^G che prevede l'utilizzo di un View-Model per far comunicare vista e modello. In questo caso la vista non ha alcuna comunicazione con il modello quindi il disaccoppiamento è maggiore rispetto a MVC.

O

Observable

È l'oggetto osservato nel design pattern Observer^G, detto anche "Subject".

Observer

È l'oggetto osservatore nel design pattern Observer^G.

Observer design pattern

È un design pattern software utilizzato per far sì che un oggetto (o nel caso di React un componente) observable^G ("osservato") notifichi modifiche interne all'observer^G ("osservatore"), che reagirà di conseguenza, solitamente chiamando un suo metodo interno.

P

Pattern architetturale

È una modellazione architetturale di un prodotto software. Ne esistono diversi e permettono di separare il comportamento delle componenti che lo compongono, aumentando il disaccoppiamento e favorendo lo unit-test delle varie componenti.

Presentation logic

Rappresenta come i componenti grafici vengono visualizzati nell'interfaccia grafica di una applicazione web.

Props

Sono i dati che vengono passati dal componente padre al componente figlio. Si possono paragonare agli argomenti che si passano ad una normale funzione o metodo.

R

Repository

Ambiente di un sistema informativo, in cui vengono gestiti i metadati, attraverso tabelle relazionali; l'insieme di tabelle, regole e motori di calcolo tramite cui si gestiscono i metadati prende il nome di metabase.

S

Stato interno/State

Un componente React può possedere uno stato interno (*statefull*) oppure no (*stateless*). Questo stato è un insieme più o meno numeroso di dati, i quali possono essere modificati attraverso interazioni dell'utente con questo componente (nella vista). La modifica di tali dati causa la rirenderizzare del componente stesso e dei suoi figli.

Strategy design pattern

Design pattern che definisce una famiglia di algoritmi che possono essere fra di loro interscambiabili. Essi infatti differiscono per il comportamento ma non per l'interfaccia.