Manuale del Manutentore

Gruppo QuaranTeam - Progetto HD Viz

quaranteam2021@gmail.com

Informazioni sul documento	
Versione	2.0.0
Approvatore	Consalvo Federico
Redattori	Sinigaglia Matteo Veronese Luca
Verificatori	Chiarello Federico Gibellato Alice
Stato	Approvato
Uso	Esterno
Destinato a	Prof. Vardanega Tullio Prof. Cardin Riccardo Zucchetti S.p.A. QuaranTeam

Descrizione

Questo documento illlustra le tecnologie adottate e l'architettura del sistema agli sviluppatori interessati all'estensione e al mantenimento del software *HD Viz*.

Registro delle modifiche

Versione	Data	Descrizione	Autore	Verificatore
2.0.0	2021-06-08	Approvazione del documento.	Consalvo Federico	
1.0.3	2021-06-08	Modifica §7.	Veronese Luca	Gibellato Alice
1.0.2	2021-06-08	Modifica §4	Sinigaglia Matteo	Gibellato Alice
1.0.1	2021-05-28	Modifica §1.	Veronese Luca	Chiarello Federico
1.0.0	2021-05-09	Approvazione del documento.	Veronese Luca	
0.8.0	2021-05-04	Stesura §6.	Rech Elia	Mason Damiano
0.7.1	2021-04-28	Modifica §1.	Gibellato Alice	Consalvo Federico
0.7.0	2021-04-21	Stesura §4.	Sinigaglia Matteo	Mason Damiano
0.6.0	2021-04-13	Stesura §5.	Rech Elia	Consalvo Federico
0.5.0	2021-04-11	Stesura §A.	Sinigaglia Matteo	Chiarello Federico
0.4.0	2021-03-28	Stesura §3.	Sinigaglia Matteo	Consalvo Federico
0.3.0	2021-03-25	Stesura §2.	Rech Elia	Mason Damiano
0.2.0	2021-03-25	Stesura §1.	Gibellato Alice	Chiarello Federico
0.1.0	2021-03-25	Creazione del documento.	Veronese Luca	Chiarello Federico

Indice

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Questo documento ha lo scopo di fornire una guida agli sviluppatori interessati ad estendere o mantenere l'applicazione *HD Viz*. Il documento contiene l'analisi dell'architettura realizzata e delle scelte progettuali adottate.

1.2 Scopo del prodotto

HD Viz è un'applicazione web avente scopo di fornire uno strumento per la visualizzazione di dati con molte dimensioni a supporto della fase esplorativa dell'analisi dei dati. *HD Viz* è in grado di rappresentare dati che possono avere almeno 15 dimensioni e fornisce 4 diversi tipi di visualizzazione a tale scopo.

1.3 Glossario

Il documento contiene termini che possono presentare significati ambigui. Viene quindi fornito un glossario individuabile nell'appendice §A, all'interno del documento, contenente tutti i termini definiti ambigui e la loro spiegazione. Nel documento vengono identificati con una G a pedice.

1.4 Riferimenti

1.4.1 Riferimenti normativi

- **Capitolato$_G$ d'appalto C4 - *HD Viz*:**
  https://www.math.unipd.it/~tullio/IS-1/2020/Progetto/C4.pdf.

1.4.2 Riferimenti informativi

- **Slide del corso di Ingegneria del Software - Diagrammi delle classi**
  https://www.math.unipd.it/~rcardin/swea/2021/Diagrammi%20delle%20Classi_4x4.pdf;
- **Slide del corso di Ingegneria del Software - Diagrammi dei package**
  https://www.math.unipd.it/~rcardin/swea/2021/Diagrammi%20dei%20Package_4x4.pdf;
- **Slide del corso di Ingegneria del Software - Diagrammi di attività**
  https://www.math.unipd.it/~rcardin/swea/2021/Diagrammi%20di%20Attivit%c3%a0_4x4.pdf;
- **Slide 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;
- **Slide del corso di Ingegneria del Software - Design pattern comportamentali**
  https://www.math.unipd.it/~rcardin/swea/2021/Design%20Pattern%20Comportamentali_4x4.pdf;
- **Libri di Testo**:
  - Software Engineering (10th edition) - Ian Sommerville - Pearson Education - Global Edition
    Sezioni:

    * §6 - Architectural Design
- UML Distilled (3rd edition) - Martin Fowler - Addison Wesley
  Sezioni:
    * §3 - Class Diagrams: The Essentials;
    * §4 - Sequence Diagrams;
    * §5 - Class Diagrams: Advanced Concepts;
    * §7 - Package Diagrams.

2 Installazione

2.1 Requisiti

Sistema operativo

- Windows 10;
- Ubuntu 20.04 o superiore, o distribuzioni Linux derivate.

Browser

- **Mozilla Firefox** v85.0 o superiore;
- **Google Chrome** v87.0 o superiore.

Non è garantita la piena compatibilità per browser derivati dai suddetti o per versioni precedenti dei suddetti.

Requisiti software

- **Node.js** 14.16.0 o superiore;
- **npm** 7.6.3 o superiore;
- **PostgreSQL** 13.2 o superiore (opzionale[1]).

Requisiti hardware

- **RAM**: necessari almeno 4 GB, consigliati almeno 8 GB;
- **connessione a Internet**.

2.2 Installazione e avvio

2.2.1 Installazione

Il repository si trova all'indirizzo https://github.com/QuaranTeam2021/HD-Viz.
Per clonare il repository seguire i seguenti passaggi:

1. Aprire un terminale posizionato nella cartella in cui si desidera salvare il repository in locale;
2. Digitare il comando per la clonazione del repository:

```
git clone https://github.com/QuaranTeam2021/HD-Viz
```

[1] necessario solo per il collegamento con il database

2.2.2 Avvio del client

Una volta posizionati nella cartella `HD-Viz-main`:

1. Aprire un terminale posizionato nella cartella `client`;
2. Se si cerca di avviare per la prima volta *HD Viz*, digitare il comando:

```
npm install
```

3. Avviare il development server tramite il comando:

```
npm start
```

4. attendere l'apertura automatica del browser predefinito o immettere in un browser l'indirizzo:

```
http://localhost:3000/
```

2.2.3 Configurazione ed avvio del server (opzionale)

Il server viene sfruttato per interagire con un database esterno, pertanto la sua installazione ed esecuzione è opzionale. Il database utilizzato è `PostgreSQL`, scaricabile al seguente link:

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

Una volta posizionati in `HD-Viz-main`:

1. Se si cerca di avviare per la prima volta la componente server, aprire un terminale nella cartella `HD-Viz-main` e digitare il comando:

```
npm install
```

2. Aprire la cartella `server` e creare un file `.env` rispettando la seguente notazione:

```
HDVIZ_USER = postgres
HDVIZ_PASSWORD = postgres
HDVIZ_HOST = localhost
HDVIZ_PORT = 5432
HDVIZ_DATABASE = hdviz
```

   Inserire come valore dei campi HDVIZ_USER, HDVIZ_PASSWORD e HDVIZ_DATABASE rispettivamente il nome utente, la password e il nome del database$_G$ a cui ci si vuole connettere.

3. Spostarsi con il terminale nella cartella `server` e avviare il server$_G$ con il comando:

```
node server
```

Il server$_G$ sarà in esecuzione alla porta `5000`.

3 Tecnologie utilizzate

In questa sezione sono descritte brevemente le tecnologie utilizzate per lo sviluppo di *HD Viz*.

3.1 Linguaggi

Segue un elenco dei linguaggi di programmazione utilizzati:

- **JavaScript**$_G$: questo è il linguaggio alla base dell'applicazione essendo il linguaggio di base per qualunque sito web.

3.2 Framework e librerie

- **React.js**: libreria JavaScript$_G$ per la creazione di interfacce utente. Permette la renderizzazione delle componenti, rendendo semplice l'applicazione di modifiche ai grafici visualizzati;
  Link: https://it.reactjs.org/
- **Material-UI**: libreria di componenti utilizzata per facilitare la realizzazione di funzionalità lato front-end. Permette di personalizzare le componenti di React adottando come linea guida il linguaggio di design Material Design di Google;
  Link: https://material-ui.com/
- **D3.js**: questa libreria costituisce il cuore della visualizzazione dei grafici. D3.js è una libreria JavaScript$_G$ per la manipolazione di documenti basati sui dati (*Data-driven documents*), combinando componenti di visualizzazione guidate dai dati con la manipolazione del DOM$_G$;
  Link: https://d3js.org/
- **DruidJS**: questa libreria è stata scelta perché racchiude al suo interno numerosi algoritmi di riduzione e ha delle buone prestazioni se confrontata con altre librerie che si pongono lo stesso obiettivo;
  Link: https://renecutura.eu/pdfs/Druid.pdf
  Link: https://saehm.github.io/DruidJS/index.html
- **MobX.js**$_G$: utilizzato per l'implementazione del pattern observer$_G$, rende semplice, trasparente e scalabile la gestione dello stato di una applicazione web. MobX$_G$ fornisce una specifica integrazione per React;
  Link: https://mobx.js.org/about-this-documentation.html
- **Node.js**: framework che permette di utilizzare JavaScript$_G$ come linguaggio per la creazione di un server con cui interfacciarsi col database. Uno dei vantaggi di Node.js è la sua caratteristica di essere guidato da eventi asincroni non bloccanti;
  Link: https://nodejs.org/it/
- **Express**: framework per applicazioni web Node.js. Facilita la creazione di un server HTTP$_G$ e la definizione di middleware$_G$;
  Link: https://expressjs.com/it/
- **Node-postgres**: libreria utilizzata per interfacciarsi ad un database PostgreSQL.
  Link: https://node-postgres.com/

3.3 Database

- **PostgreSQL**: database relazionale open-source. Link: https://www.postgresql.org/

4 Architettura

La base dell'architettura per l'applicazione web *HD Viz* è rappresentata da MobX$_G$. Il vantaggio offerto da MobX$_G$ è la gestione automatica dello stato con un set minimale di istruzioni da implementare. MobX$_G$ e React, che è stato utilizzato per lo sviluppo della UI$_G$, sono comunemente usati insieme e per questo MobX$_G$ supporta una specifica integrazione con React che mette a disposizione un framework minimale per l'implementazione dell'*observer$_G$ pattern*.
Le componenti principali sono le seguenti:

- **Domain store**: i domain stores sono degli archivi che contengono i dati dell'applicazione. Nel nostro caso lo *Store* principale contiene dati che vengono caricati da database o da file locale. La componente *Store* è poi composta da altri oggetti di dominio al suo interno, come l'array di istanze *Graphs* che archivia lo stato di ogni diverso grafico richiesto sulla UI$_G$. Lo *Store* sarà quindi la componente di architettura che verrà resa *observable* (*subject*) da MobX$_G$;

- **Components**: queste sono le componenti React che compongono la UI$_G$. Sono organizzate secondo una struttura gerarchica, che rappresenta in quale punto vengono inserite nell'interfaccia. Il livello immediatamente sottostante alla componente radice, *App*, permette inoltre di individuare le diverse funzioni che l'applicazione fornisce (esplorazione di dati, gestione dei dataset salvati e manuale utente). La parte di esplorazione dei dati, funzionalità principale dell'applicazione, contiene il maggior numero di sotto-componenti. È proprio in queste che avviene l'integrazione tra le due librerie. Con il modulo *observer* è possibile inglobare specifiche componenti che necessitano di osservare lo stato dello *Store*. Non è quindi necessario rendere observer l'intera vista, ma solo le componenti necessarie;

- **Controllers**: diversamente da quanto previsto dall'utilizzo standard di MobX$_G$ con React abbiamo introdotto delle componenti controller che servono per trasformare gli input utente che arrivano dalle componenti React in dati che possano essere archiviati nello *Store*. Le componenti controller sono state divise per ruolo in modo da renderle più semplici e minimali senza includere funzionalità superflue.

Per rendere il tutto funzionante si fa utilizzo dei *React context*. È possibile vedere l'organizzazione delle componenti React come una struttura ad albero. Un contesto è visibile da tutte le componenti dell'applicazione una volta che viene passato grazie ad un *Provider* alla componente radice, che la rende disponibile a tutte le componenti discendenti da quest'ultima. Per utilizzare un contesto, che rimane quindi disponibile per tutta l'esecuzione, si utilizza *useContext()* offerta da React.
React mette a disposizione anche l'hook *useEffect* che può essere utilizzato per impostare dei *side effect* che devono verificarsi e che sono legati al ciclo di vita della componente React. L'utilizzo di *useEffect* richiede la specifica delle dipendenze, che possono essere lo *store context* o un suo particolare attributo, in modo tale che la componente reagisca al loro cambiamento di stato. La possibilità di definire come dipendenze gli attributi è fondamentale perché permette alle componenti di rirenderizzarsi solo in seguito a specifici cambiamenti.
In sintesi nell'architettura descritta è stato implementato un *observer$_G$ pattern* nel quale lo *Store* è un *subject* e diverse componenti di React sono *observers* che reagiscono al cambiamento di stato dello *Store*.

4.1 Diagramma dei package

Segue il diagramma dei package:

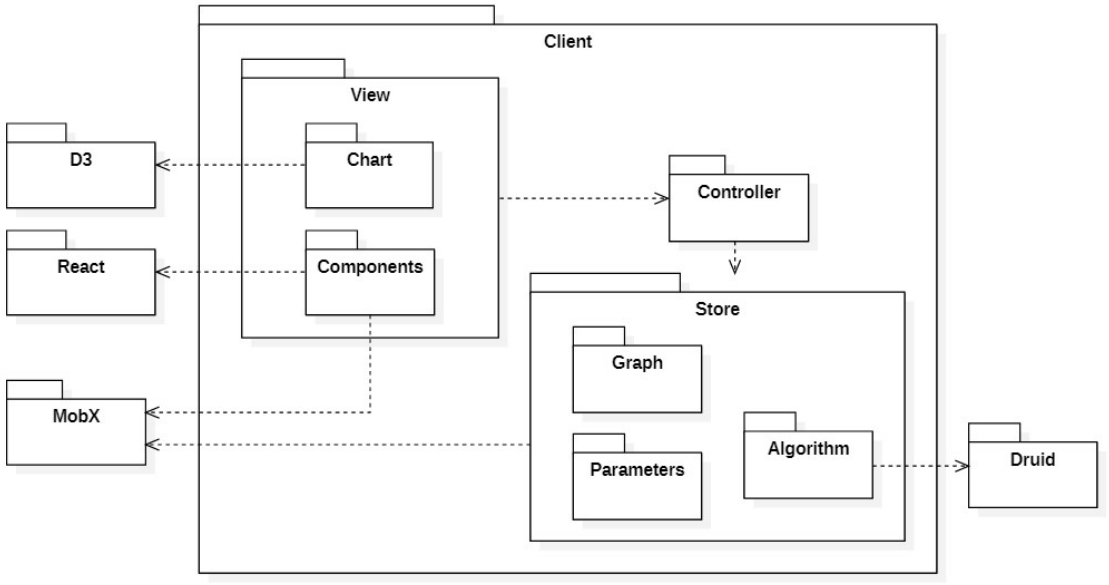

Figura 1: Diagramma dei package

Il diagramma dei package (Figura 1) evidenzia le dipendenze esterne della componente Client e la sua suddivisione interna in package. In particolare i package principali sono:

- **Store**: contenente le classi di business logic;
- **Controller**: contenente i controller, che fungono da mediatori tra la view e lo store garantendo la *Separation of Concerns*;
- **View**: contenente le componenti React e le classi per la generazione dei grafici;

4.2 Diagramma delle classi

Segue il diagramma della classi:

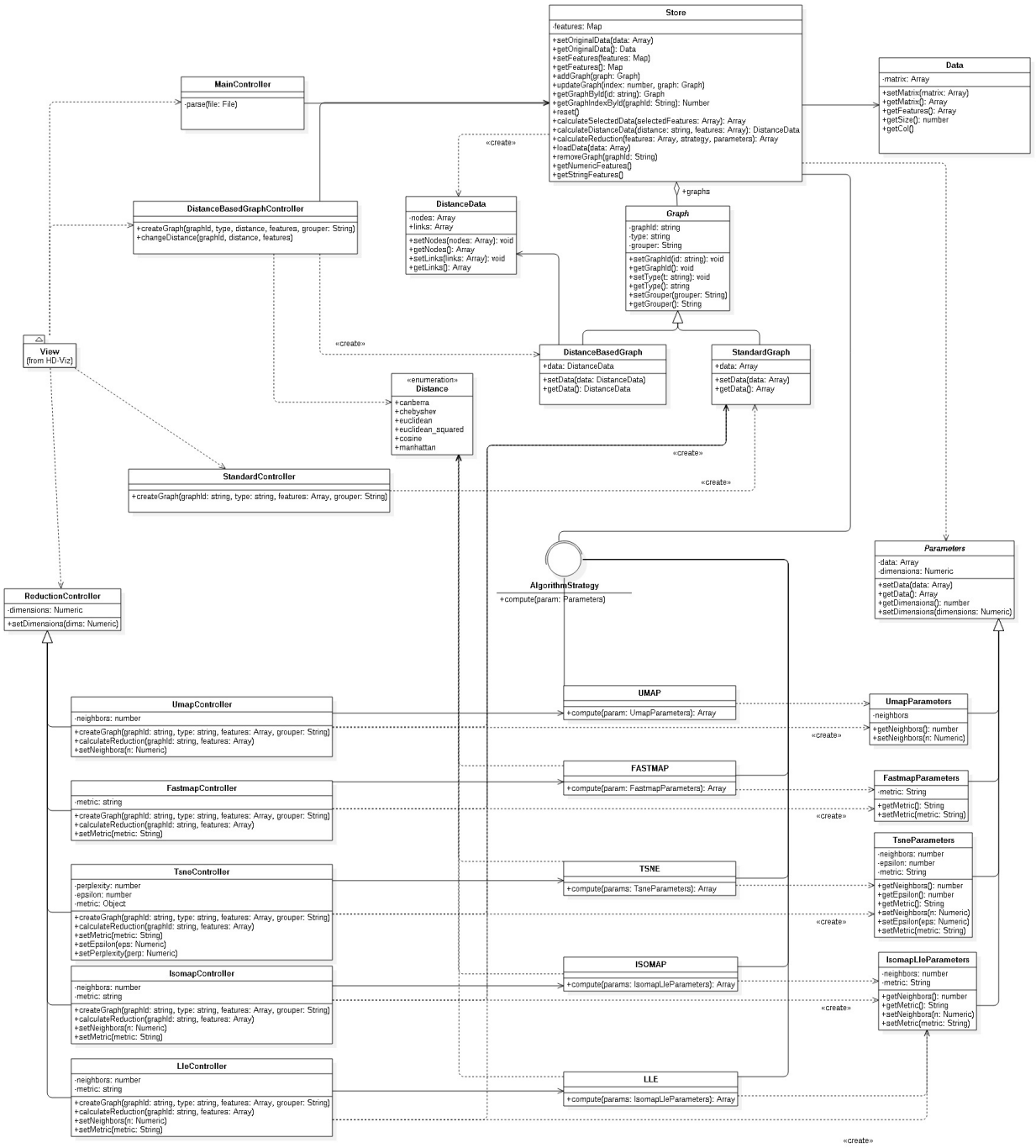

Figura 2: Diagramma delle classi

Il diagramma delle classi è composto principalmente dalla classe **Store**. Quest'ultima ha lo scopo di archiviare i dati caricati dall'utente e contiene tutti i dati relativi ai grafici visualizzati sulla vista,

fornendo i metodi per aggiornare e recuperare i dati. Le altre classi sviluppate sono:

- **Data**: definisce una struttura dati per contenere i dati caricati e fornisce i metodi per recuperare i dati e modificarli;
- **DistanceBasedGraph, StandardGraph**: contengono tutte le informazioni selezionate dall'utente per la creazione di ogni grafico, nonché i dati nel formato richiesto. Implementano la classe astratta **Graph**;
- **DistanceData**: è una struttura dati che contiene dati basati sulle distanze in due array *nodes, links*;
- **UmapParameters, FastmapParameters, IsomapLleParameters, TsneParameters**: contengono i parametri per lo specifico algoritmo cui sono dedicate. Estendono la classe astratta **Parameters**;
- **DatabaseLoaderController, DatabaseManagerController, DatabaseTablesController, LocalLoaderController, DistanceBasedGraphController, StandardController, UmapController, IsomapController, LleController, TsneController, FastmapController**: catturano gli input degli utenti e utilizzano i metodi implementati per modificare lo **Store**;
- **Distance**: enumerazione che contiene le possibili metriche di distanza della libreria DruidJS.

4.2.1 Design pattern notevoli

4.2.1.1 Observer pattern

È stata utilizzata la libreria MobX$_G$ per marcare la componente **Store** come *observable* e specifiche componenti React come *observer*. Questo ha permesso di definire degli hook *useEffect* che reagiscono a precisi cambiamenti dello **Store**. La realizzazione dell'*observer pattern* è possibile marcando la classe da osservare con *makeAutoObservable()* mentre le componenti React possono essere inglobate all'interno di *observer()*.

4.2.1.2 Strategy pattern

Per l'implementazione degli algoritmi di riduzione è stato utilizzato lo *Strategy$_G$ pattern*. La classe **AlgorithmStrategy** è un'interfaccia che viene implementata dalle classi **UMAP**, **LLE**, **TSNE**, **ISOMAP**, **FASTMAP**. Ognuna di queste classi implementa il metodo astratto *compute()* per calcolare la riduzione dimensionale sui dati. Ciascuna di queste classi è un template tipizzato su una specifica classe di tipo **Parameters**.

4.3 Diagrammi di sequenza

Seguono i diagrammi di sequenza per le funzionalità principali:

4.3.1 Caricamento file da locale

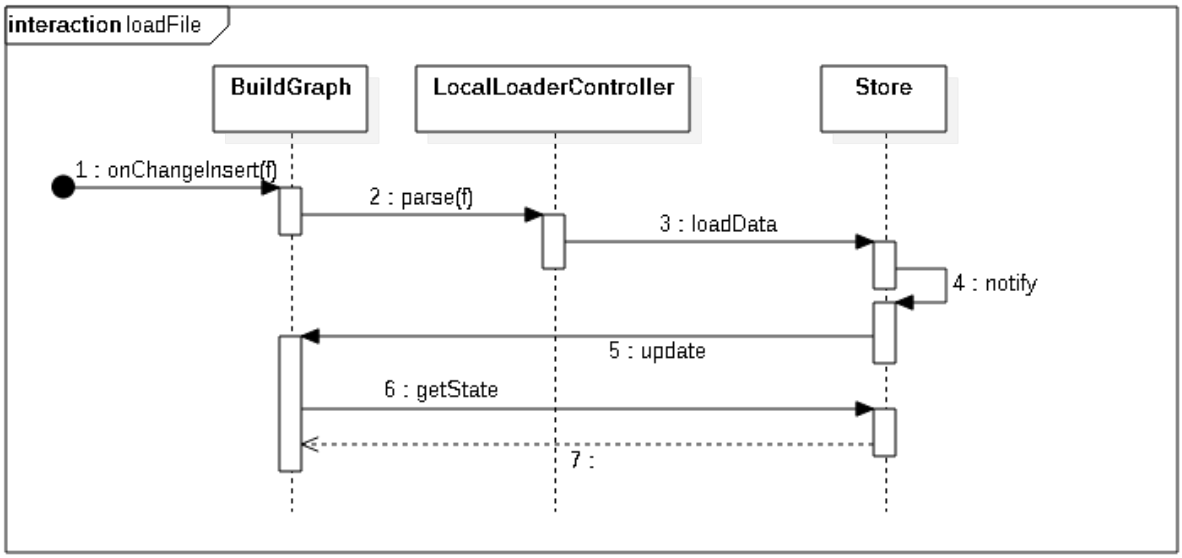

Figura 3: Diagramma di sequenza per il caricamento di un file da locale

Al verificarsi dell'evento *onChangeInsert* la componente *BuildGraph* della UI_G invoca il metodo *parse()* del controller *LocalLoaderController* del quale possiede un'istanza.
Il file caricato viene controllato in dimensione, subisce il parsing e viene memorizzato all'interno di un array. Le features individuate nei dati vengono salvare in una struttura dati *mappa <String, String>* per memorizzare la tipologia di ogni dimensione.
Successivamente il controller invoca *loadData()* della componente *Store*. Lo *Store* notifica tutti i sui osservatori grazie all'implementazione del pattern $observer_G$ realizzata con la libreria $MobX_G$.

4.3.2 Creazione di un grafico con riduzione dimensionale

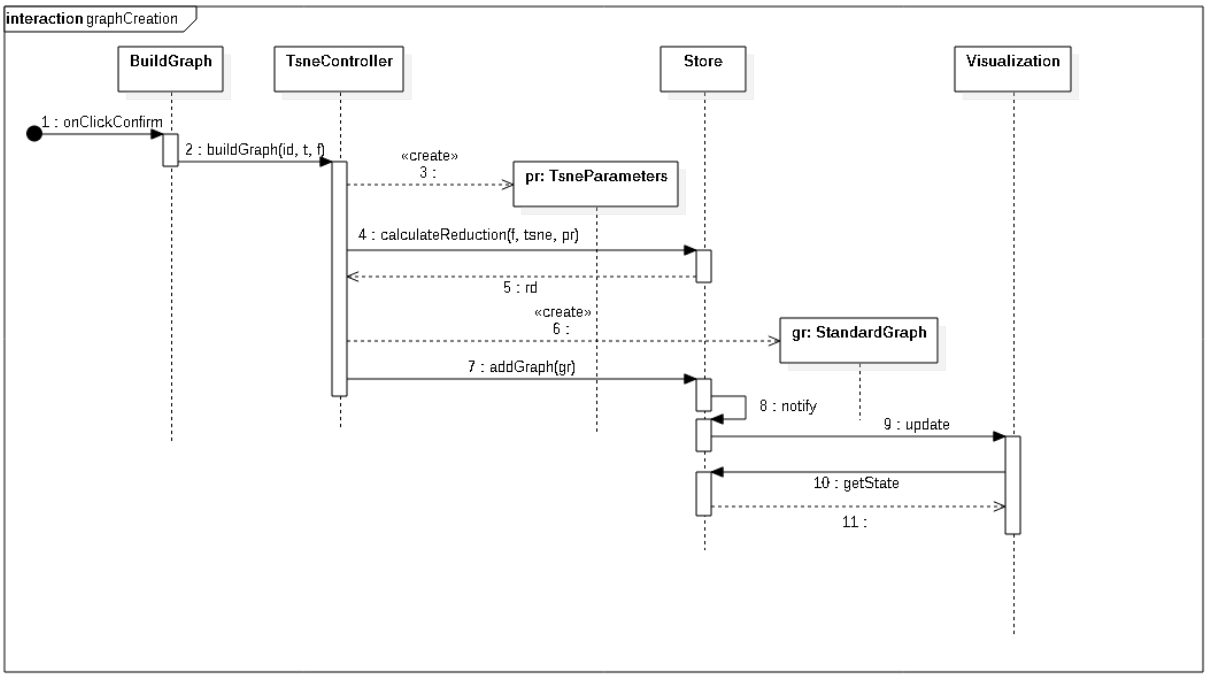

Figura 4: Diagramma di sequenza per la creazione di un grafico per il quale è richiesta riduzione dimensionale

Al verificarsi dell'evento *onClickConfirm*, la componente *BuildGraph* della UI_G invoca il metodo *buildGraph()* del controller *TsneController* del quale possiede un'istanza. In questo caso i controller derivano tutti da un'unica classe astratta *ReductionController*, e sfruttano così il polimorfismo per l'invocazione dei metodi.
Il controller crea l'istanza *TsneParameters* che viene passata come parametro per calcolare l'algoritmo. Successivamente viene chiamato *calculateReduction()* della classe *Store* per ottenere i dati ridotti da inserire nel grafico. Viene creata l'istanza per contenere un grafico *StandardGraph()* e successivamente si passa questa istanza come parametro del metodo *addGraph()* della classe *Store*. Il procedimento rimane invariato per la creazione di un qualunque altro grafico o algoritmo di riduzione. La componente *Visualization* è un observer dello *Store*, e verrà quindi notificata all'inserimento di un nuovo grafico.

4.4 View

La view si occupa di fornire una UI_G. L'interfaccia è stata implementata attraverso l'utilizzo delle librerie React e Material UI. È stato utilizzato il modulo `npm React-router-dom` per il rendering di pagine diverse senza ricaricare l'intero sito. Il router viene, infatti, implementato in un file $JavaScript_G$ chiamato **App.js**, il quale è responsabile della gestione dei componenti che devono essere renderizzati. La view è organizzata in quattro macro-componenti, le quali raccolgono più componenti. Seguono le componenti principali:

- **App**: essa permette di creare un'istanza dello **Store** e dei **Controller** e di passarne il contesto a tutte le sotto componenti. App.js include le seguenti sotto componenti: **Header**, **Help**, **BuildGraph**, **ManageDataset**, **Visualization**, **FeaturesGraph**;
- **BuildGraph**: la componente BuildGraph contiene tutte le funzionalità utili all'inserimento dei dati per poter visualizzare un grafico di *HD Viz*. Essa racchiude i seguenti componenti: **Insert**, **SelectGraph**, **SelectAlgorithm**, **SelectColumns**, **ButtonConfirm**;
- **Visualization**: contiene tutti i grafici generati l'utente, ognuno dei quali è contenuto in una componente **GraphContainer** dedicata;

5 Server

La parte server (opzionale per il funzionamento dell'applicazione) gestisce le query verso il database PostgreSQL ed il caricamento o la rimozione di dataset da esso, sfruttando la libreria node-postgres. Il framework Express.js gestisce le richieste HTTP$_G$ inviate dalla web app.

5.1 Gestione delle richieste

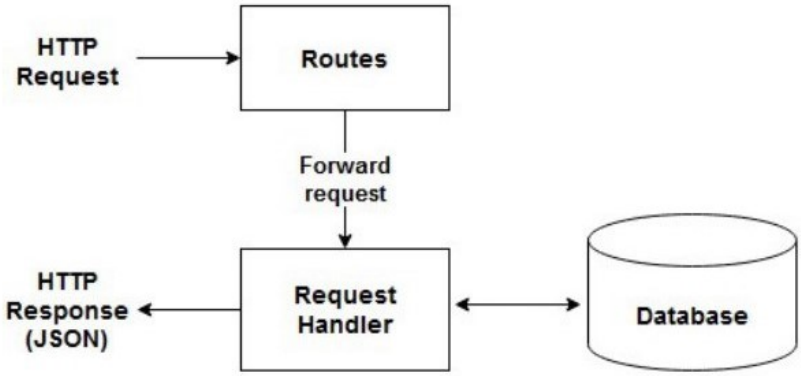

Figura 5: Schema di interazione con la parte server

La gestione di una richiesta HTTP$_G$ inviata dalla web app comprende i seguenti passaggi:

1. Le routes definite indirizzano le richieste della web app chiamando un'opportuna funzione middleware$_G$;
2. La funzione middleware$_G$ opera dei controlli sui parametri passati dal client tramite la richiesta HTTP$_G$. Nel caso di esito negativo provvede a ritornare un messaggio di errore significativo al client, in caso di esito positivo sfrutta la libreria node-postgres per effettuare le operazioni sul database collegato;
3. In base all'esito della query o dell'operazione sul database effettuata, viene ritornato il contenuto richiesto dal client o un messaggio di errore significativo.

5.2 Routing

Vengono utilizzati i seguenti percorsi di route:

- **(GET) /api/tableslist**: ritorna la lista dei dataset memorizzati nel database;
- **(GET) /api/getcontent/:table**: ritorna il dataset memorizzato nella tabella `table`;
- **(GET) /api/getcolnames/:table**: ritorna i nomi delle dimensioni del dataset `table`;
- **(GET) /api/getselectedcol/:table**: ritorna le dimensioni selezionate del dataset `table`;
- **(POST) /api/upload/:table**: carica nel database il dataset `table` estraendolo da un file in formato CSV$_G$ o TSV$_G$. Nel caso in cui non fosse presente il parametro `table`, il dataset viene salvato con il nome del file contenente i dati;

- **(DELETE) /api/delete/:table**: elimina dal database il dataset `table`;
- **(GET) /gettoken**: ritorna un token (JWT$_G$) alla componente client.

5.3 Sicurezza

La protezione dei percorsi di routes esposti viene garantita tramite l'uso di JWT$_G$, attraverso i seguenti passaggi:

1. La componente client dell'applicazione provvede ad effettuare una richiesta HTTP$_G$ alla route `/gettoken`, passando come parametri delle credenziali che ne consentano l'autenticazione;
2. La componente middleware$_G$ richiamata dalla route effettua un controllo sulle credenziali, se il controllo ha esito positivo provvede a generare un nuovo token che viene ritornato al client;
3. Il client salva il token che verrà quindi incluso nelle successive richieste HTTP$_G$;
4. Una funzione middleware$_G$ effettua un controllo sulle route `/api/`, relativamente alla presenza e alla validità del token, in caso di esito negativo provvede a ritornare un messaggio di errore al client, invece in caso di esito positivo ridirige la richiesta alla route opportuna.

6 Come estendere l'applicazione

Essendo *HD Viz* un'applicazione progettata cercando di rispettare al meglio l'*Open-closed principle*$_G$, è possibile estenderla in modo semplice.

6.1 Aggiunta grafici

Vi è possibilità di aggiungere nuovi grafici per la visualizzazione dati senza limiti di alcun tipo. Per aggiungere un nuovo grafico i passi da seguire sono i seguenti:

1. Inserire il codice D3.js per la creazione del grafico in un file con estensione *.js*. Inserire il suddetto file nella cartella `HD-Viz/client/src/view/chart`;
2. Aggiungere le opzioni necessarie nel form della componente *BuildGraph*;
3. Passo opzionale. Se il grafico implementato richiede un dataset con un formato diverso da quelli attualmente integrati sarà necessario implementare la classe astratta *Graph* con una nuova classe concreta in grado di archiviare i dati nel formato richiesto. In questo caso potrebbe essere necessario creare un nuovo controller che possa catturare gli input utenti e renderli utili all'archiviazione.

6.2 Aggiunta algoritmi di riduzione

Gli algoritmi di riduzione sono fondamentali per permettere la visualizzazione di dati con molte dimensioni. Si possono aggiungere algoritmi di riduzione che differiscono tra loro per risultati e performance. Per aggiungere un algoritmo di riduzione va eseguita la seguente procedura:

1. Implementare la classe astratta *Parameters* aggiungendo tutti i parametri necessari al nuovo algoritmo di riduzione;
2. Implementare l'interfaccia *AlgorithmStrategy* implementando il metodo *compute()*, tipizzata sulla classe precedentemente creata;
3. Creare un nuovo controller, implementando la classe astratta *ReductionController*, che contiene un'istanza della classe rappresentante il nuovo algoritmo. Implementare il metodo *createGraph()* per la creazione di un grafico al quale si passano i dati ridotti con il nuovo algoritmo. Questo controller deve possedere un attributo per ogni parametro necessario alla costruzione di un oggetto *Parameters*.

A Glossario

C

CSV

Acronimo di Comma-Separated Values, è un formato di file basato su file di testo utilizzato per l'importazione ed esportazione di una tabella di dati.

D

DOM

Acronimo di Document Object Model, è una forma di rappresentazione dei documenti strutturati come modello orientato agli oggetti.

G

GitHub

Servizio di hosting per progetti software, che implementa lo strumento di controllo versione distribuito Git.

H

HTTP

Acronimo di HyperText Transfer Protocol. È un insieme di regole che un server deve seguire quando si tratta della trasmissione di file attraverso il World Wide Web.

J

JavaScript

Linguaggio di scripting orientato agli oggetti e agli eventi, utilizzato nella programmazione web sia lato client che lato server.

JSON

Acronimo di JavaScript Object Notation. È un semplice formato per lo scambio di dati. Per le persone è facile da leggere e scrivere, mentre per le macchine risulta facile da generare e analizzarne la sintassi.

JWT

Acronimo di JSON Web Token, è un sistema di cifratura e di contratto in formato JSON_G per lo scambio di informazioni tra i vari servizi di un server. Vengono utilizzati nei web services e nelle web app per l'autenticazione dei client.

M

Middleware

Si intende il software che rende accessibile sul web risorse hardware o software che prima erano disponibili solo localmente o su reti non internet.

MobX

Libreria di gestione dello stato. Proprio come React, che utilizza un DOM_G virtuale per eseguire il rendering degli elementi dell'interfaccia utente nei browser, riducendo il numero di mutazioni DOM_G, MobX fa la stessa cosa ma nello stato dell'applicazione.

O

Observer

Pattern che definisce una dipendenza di tipo $1..n$ fra oggetti riflettendo la modifica di un oggetto su di quelli che sono a lui relazionati. Questo serve a mantenere consistenza nel sistema facendo in modo che in seguito ad una segnalazione avvengano degli aggiornamenti.

Open-closed principle

Principio della programmazione orientata agli oggetti che afferma che le entità (classi, moduli, funzioni) software dovrebbero essere aperte all'estensione, ma chiuse alle modifiche; in maniera tale che un'entità possa permettere che il suo comportamento sia modificato senza alterare il suo codice sorgente.

S

Separation of Concerns

Principio di progettazione che prevede la separazione di un programma in sezioni distinte.

Strategy

Pattern che definisce una famiglia di algoritmi che possono essere fra di loro interscambiabili. Viene utilizzato quando si hanno tipi diversi che differiscono tra di loro per il comportamento ma non per l'interfaccia.

T

TSV

Formato di file il cui nome sta per "Valori separati da tabulazione", e questi TSV file vengono creati e utilizzati da molte applicazioni per fogli di calcolo.

U

UI

Acronimo di User Interface. L'interfaccia utente è l'interfaccia visuale con la quale l'uomo interagisce con un computer o una macchina per portare a termine un'attività.