Progetto: ***HDViz***

codebusterswe@gmail.com

Allegato Tecnico

Informazioni sul documento	
Versione	1.0.0-0.7
Approvatori	Zenere Marco
Redattori	Pirolo Alessandro Zenere Marco Rago Alessandro Safdari Hossain
Verificatori	Sassaro Giacomo Scialpi Paolo Baldisseri Michele
Uso	Esterno
Distribuzione	*Zucchetti* Prof. Vardanega Tullio Prof. Cardin Riccardo Gruppo *CodeBusters*

Descrizione

Questo documento racchiude tutte le scelte architetturali che il gruppo ha preso nella realizzazione del prodotto *HDViz*. Per la descrizione del prodotto sono utilizzati i diagrammi delle classi, dei package e di sequenza.

Indice

Elenco delle tabelle

1 Introduzione

1.1 Scopo del documento

Lo scopo di questo documento è descrivere e motivare tutte le scelte architetturali che il gruppo *CodeBusters* ha deciso di fare nella fase di progettazione e codifica del prodotto. Vengono quindi riportati i diagrammi delle classi, dei package e di sequenza per descrivere architettura e funzionalità principali del prodotto. È poi presente una sezione dedicata ai requisti che il gruppo è riuscito a soddisfare in ingresso alla RQ, così da fornire un'ampia visione sullo stato di avanzamento del lavoro.

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 capitolato C4 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 Riferimenti

1.3.1 Riferimenti normativi

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

1.3.2 Riferimenti informativi

- **Slide E1 del corso di Ingengeria del Software - Diagrammi delle classi e dei package**
- **Slide E2 del corso di Ingengeria del Software - Diagrammi delle attività e di sequenza**
- **Slide E10 del corso di Ingengeria del Software - Design pattern comportamentali**
  - Da slide 22 a slide 40 (Observer Pattern, Strategy Pattern)
- **Slide L02 del corso di Ingengeria del Software - Design pattern architetturali: Model View Controller e derivati**

2 Architettura del prodotto

2.1 Descrizione generale

Il pattern architetturale scelto dal gruppo per lo sviluppo del progetto è il Model-View-ViewModel. Il seguente pattern è tra i più diffusi nello sviluppo delle web application e permette di scrivere codice facilmente mantenibile e riusabile; questo è possibile grazie al forte disaccoppiamento che sussiste tra logica di presentazione e di business. Inoltre l'MVVM è risultato il più adatto per essere utilizzato con React, libreria impiegata per lo sviluppo dell'UI e che renderizza le componenti in base al loro stato interno.

- **Model**: questa porzione ricopre la logica di business dell'applicazione, ovvero la gestione dei dati di partenza, dimensioni e strutture create dall'utente ed infine le preferenze di visualizzazione dei grafici. Per una corretta separazione logica, il *Model* è stato suddiviso in tre parti: una dedicata ai dati e alle dimensioni (*DataSet.js*), una seconda per la gestione delle matrici delle distanze (*DistanceMatrices.js*) ed un'ultima dedicata alle preferenze dell'utente (*Preferences.js*);
- **ViewModel**: qui viene effettuato il binding tra *View* e *Model* ed è contenuta la loro logica;
- **View**: questa porzione gestisce la presentazione tramite una specifica gerarchia di componenti; ciascun componente contiene la logica strettamente legata alla sua visualizzazione e necessaria al mantenimento del proprio stato interno.

Il passaggio dei dati dal *Model* alle varie componenti grafiche avviene attraverso l'utilizzo di un *Context React*, al quale viene passato un'istanza del *ViewModel*. L'utilizzo di un *Context React* ci permette di accedere al valore corrente del *ViewModel* in qualsiasi porzione della *View*, senza doverlo passare di componente in componente attraverso le props (ossia gli argomenti dei componenti che compongono la vista). Nella radice dell'applicazione viene infatti creata un'istanza del *ViewModel*, che viene passata ad un *Context.Provider*, che fa da contenitore per tutta la *View*. All'interno di tale contenitore ogni componente può utilizzare un hook per accedere al *Context React* ed utilizzare il valore più recente del *ViewModel*.
È stato scelto di utilizzare un *Context React* per il passaggio dei dati in quanto la nostra applicazione è molto profonda e non risultava conveniente passare i dati per molti componenti rischiando, nel peggiore dei casi, di doverli utilizzare nell'ultimo della gerarchia.
Per poter fare in modo che una componente della *View* si renderizzi non solo al cambiamento del suo stato interno ma anche al cambiamento dei dati nel *Model*, abbiamo utilizzato la libreria *Mobx*. Questa ci permette di implementare l'**observer pattern**, non supportato di default da *React*. A tale scopo, Mobx permette di segnare delle classi (o attributi di esse) come "*observable*" e di costruire dei componenti della View come "*observer*". Quest'ultimi vengono automaticamente ri-renderizzati al cambiamento di un qualsiasi attributo *observable*.

2.2 Diagramma dei package

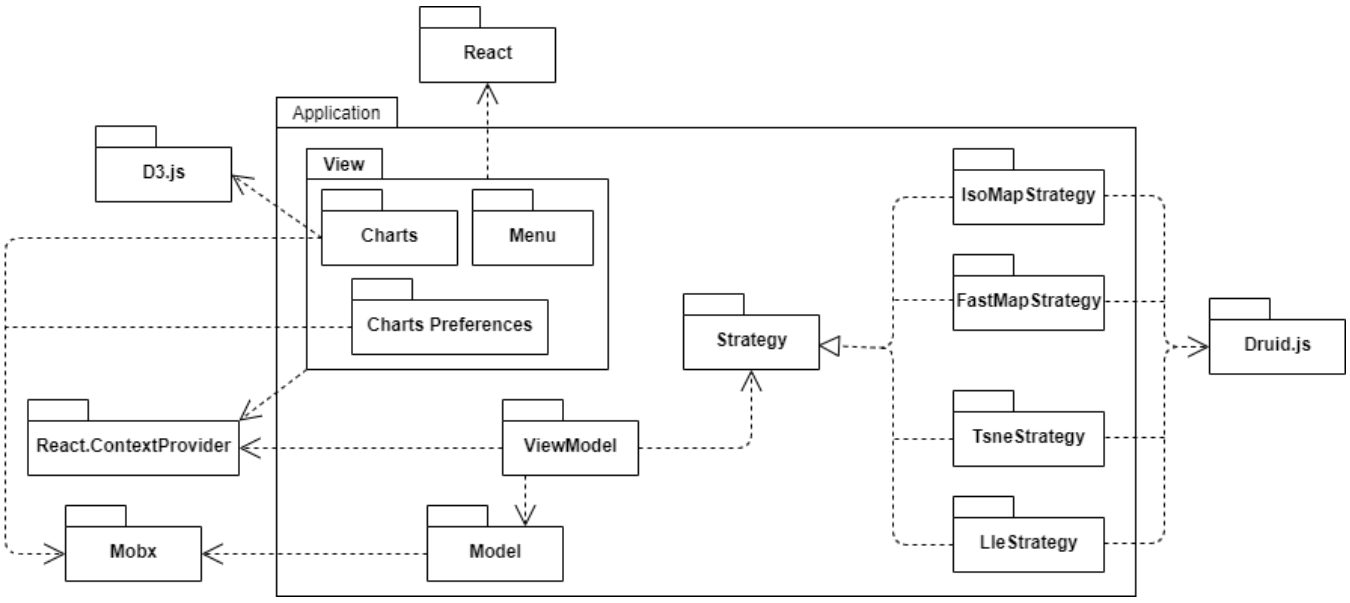

Figura 1: Diagramma dei package del client

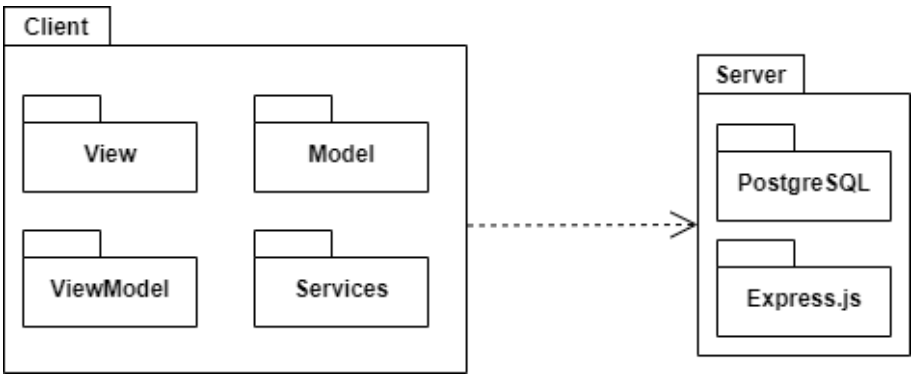

Figura 2: Diagramma dei package dell'applicazione

2.3 Diagrammi delle classi

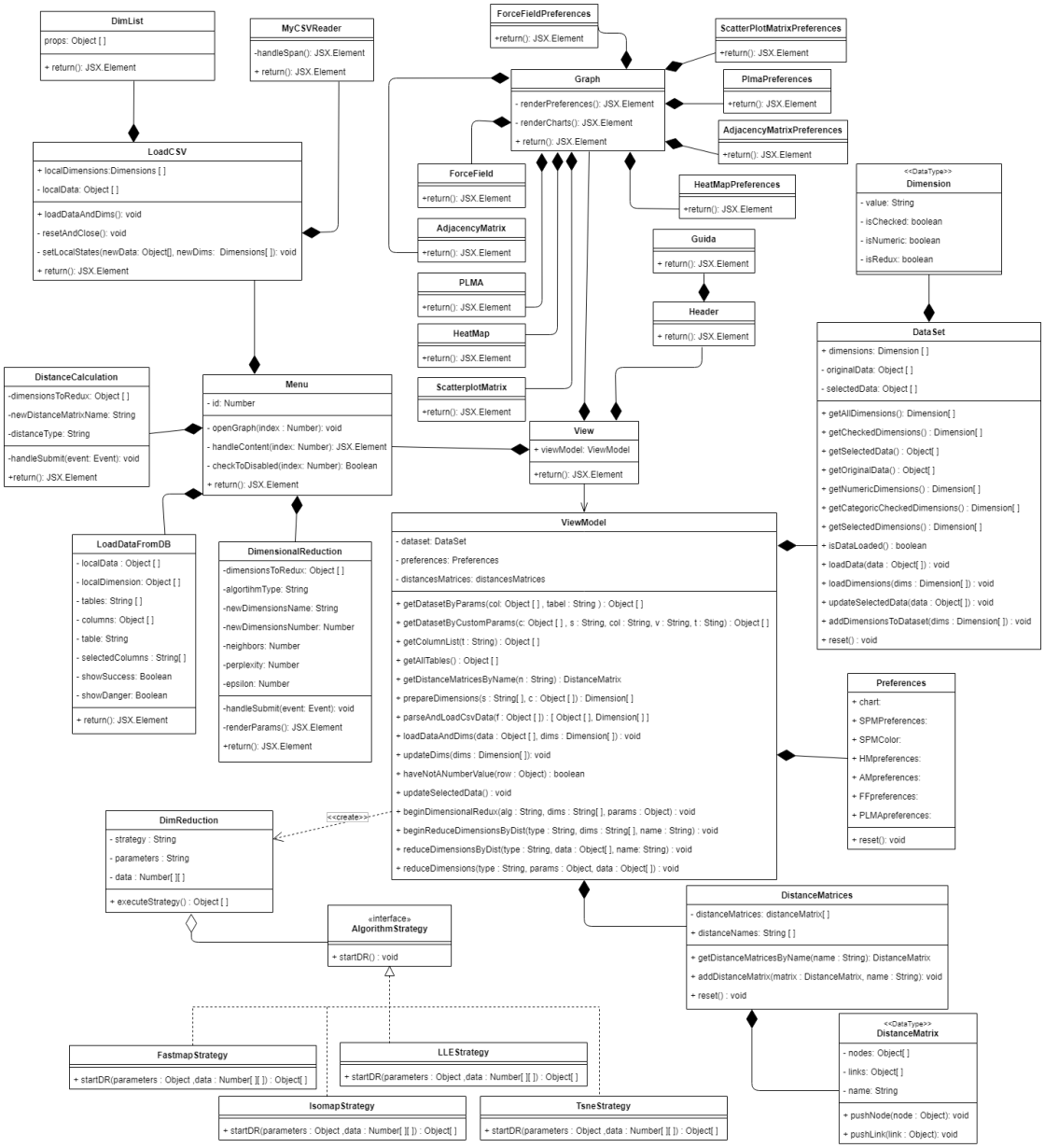

Figura 3: Diagramma delle classi generale

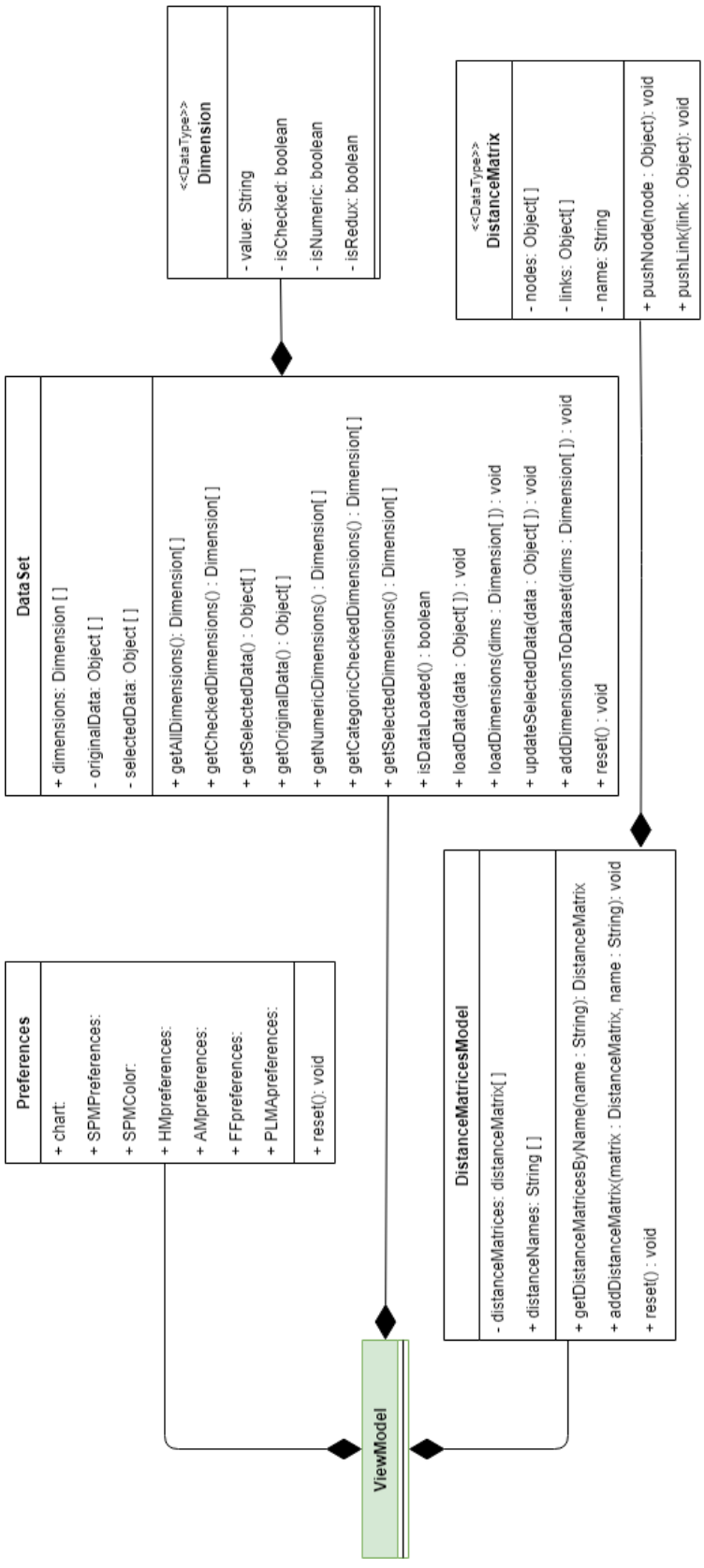

Figura 4: Diagramma delle classi, zoom sul modello

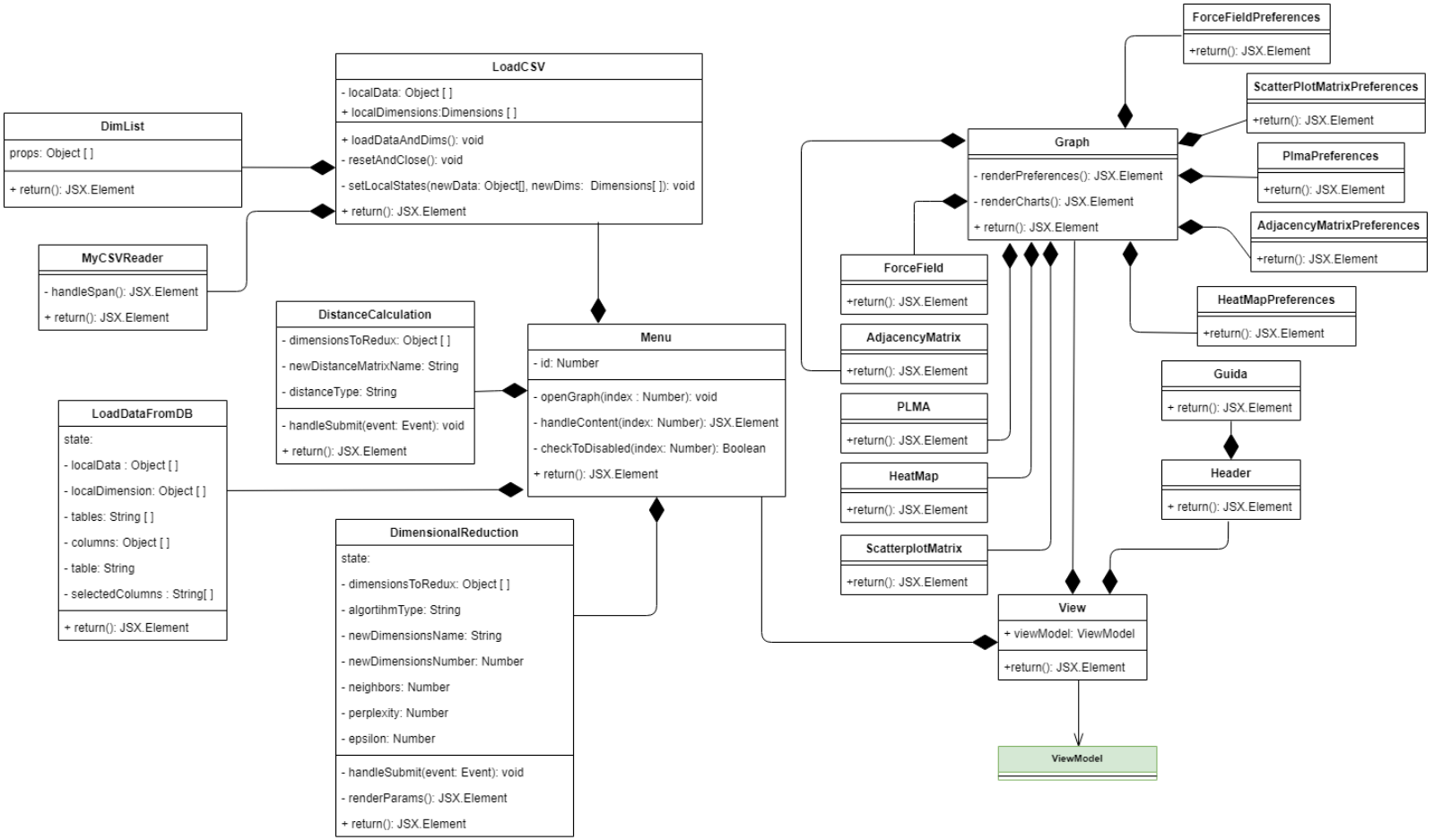

Figura 5: Diagramma delle classi, zoom sulla vista

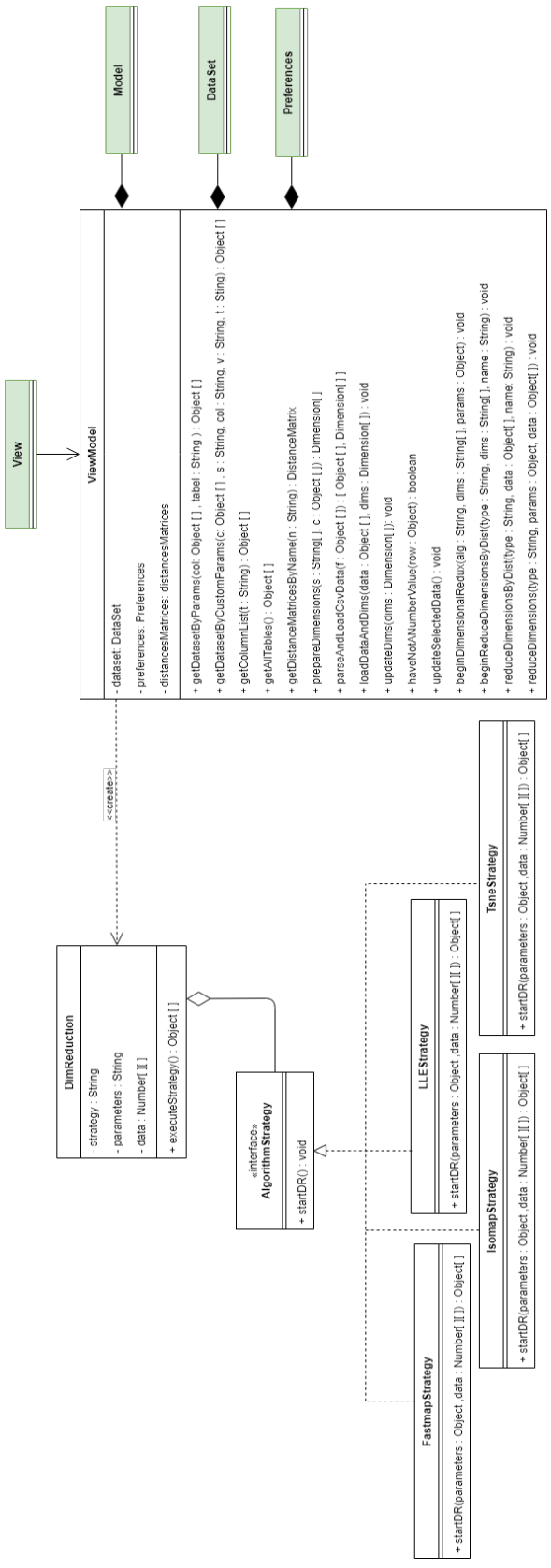

Figura 6: Diagramma delle classi, zoom sul view-model e implementazione del pattern strategy

2.4 Diagrammi di sequenza

Qui di seguito vengono rappresentati i diagrammi di sequenza per le 3 operazioni più importanti nel progetto:

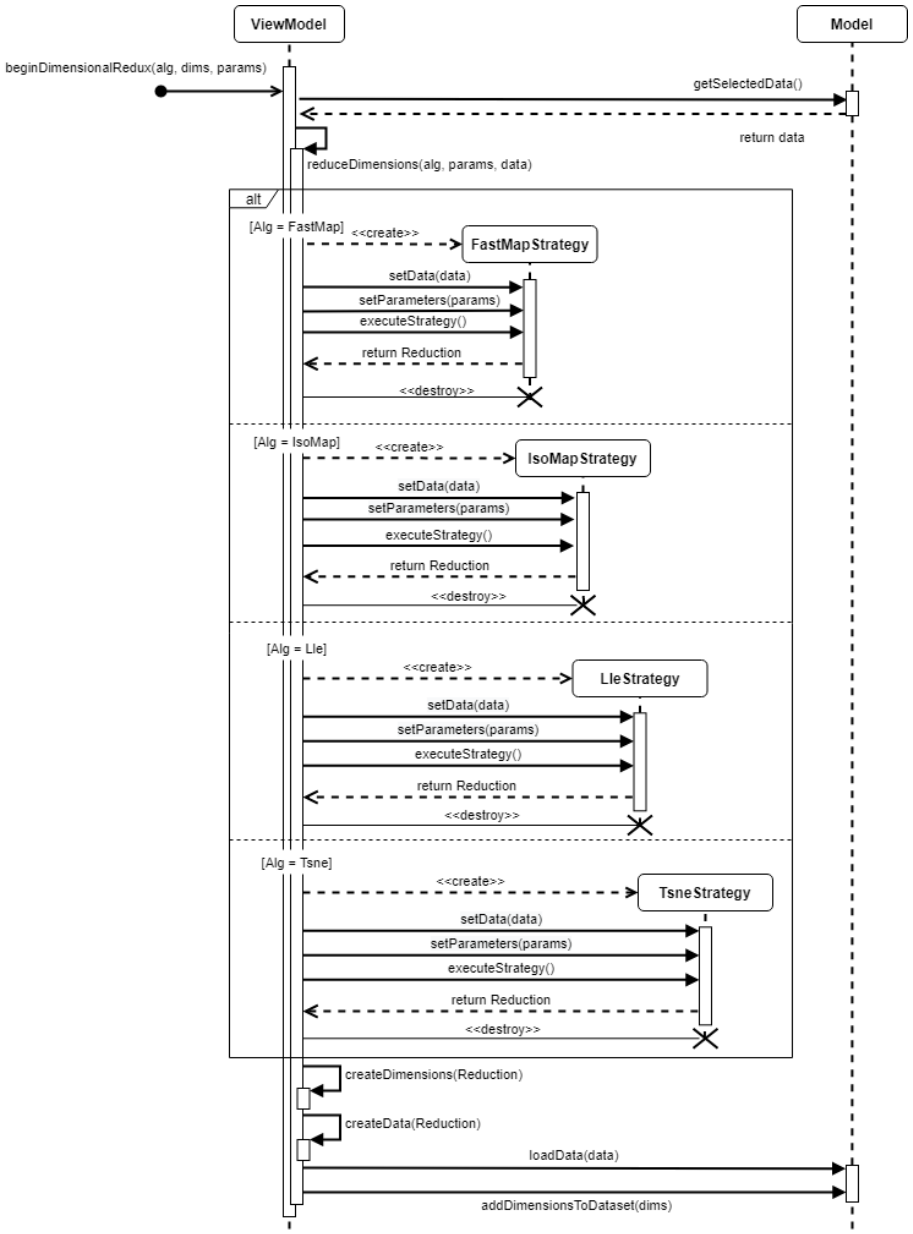

Figura 7: Diagramma di sequenza che modella il processo di riduzione dimensionale

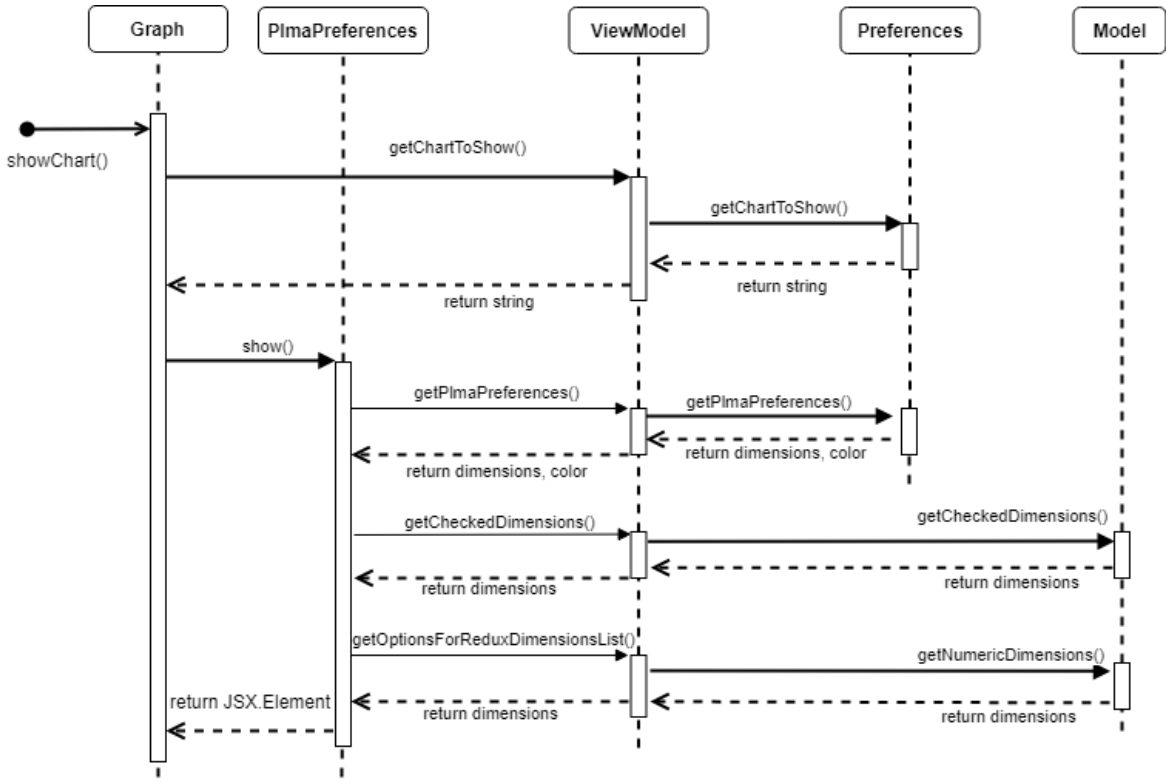

Figura 8: Diagramma di sequenza che modella il processo di visualizzazione delle preferenze per il grafico PLMA

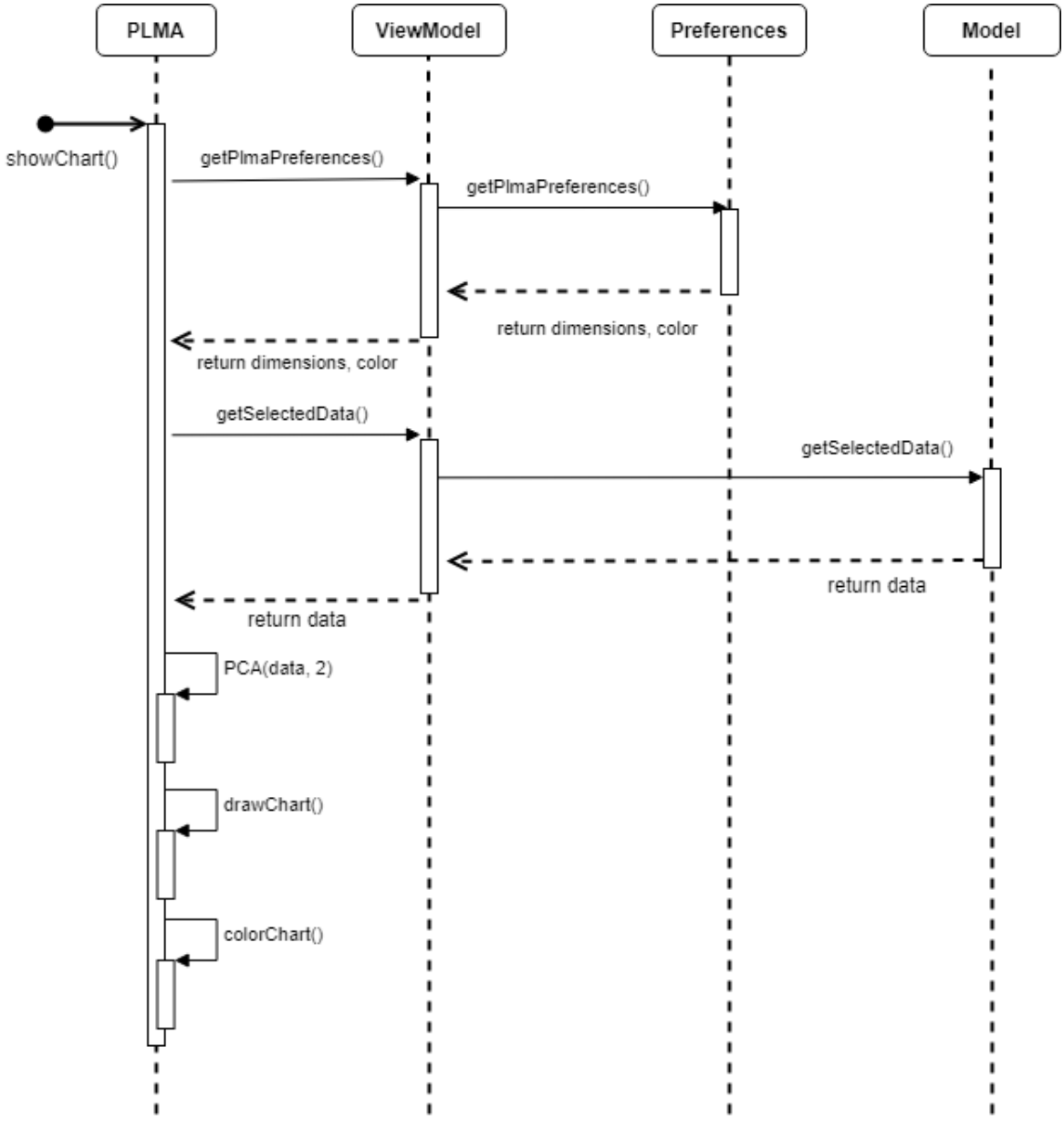

Figura 9: Diagramma di sequenza che modella il processo per la visualizzazione del grafico PLMA

2.5 Strategy Pattern

Per implementare il processo di riduzione dimensionale è stato utilizzato il design pattern **strategy**. Questo è stato possibile perché l'unica sostanziale differenza era determinata dalla tipologia di algoritmo applicato. Definire una famiglia di algoritmi e isolarli all'interno di un oggetto ci ha permesso di renderli interscambiabili dinamicamente ed evitare duplicazione di codice.

- **DimReduction.js**: rappresenta il *Context*, ovvero la classe concreta che invoca la *ConcreteStrategy* sotto richiesta del client;
- **AlgorithmStrategy.js**: interfaccia comune a tutti gli algoritmi ed utilizzata da *DimReduction* per settarli ed invocarli;
- **FastmapStrategy.js, IsomapStrategy.js, LLEStrategy.js, TsneStrategy.js**: rappresentano gli algoritmi concreti ed espongono l'interfaccia comune.

Nel nostro caso, il pattern si basa sul metodo `startDR()` che esegue la riduzione dimensionale sui dati scelti e ritorna le nuove dimensioni. La strategy viene passata come stringa e, una volta creata la classe concreta dell'algoritmo scelto, viene costruita l'apposita istanza dalla libreria *Druid.js*.
Eventuali integrazioni con ulteriori algoritmi di riduzione dimensionale potranno essere effettuate derivando nuove classi concrete dall'interfaccia esposta.

3 Requisiti soddisfatti

Seguendo quanto definito nel *Piano di Progetto v3.0.0-0.2* il gruppo è riuscito a soddisfare tutti i requisiti obbligatori. Arrivati a questo punto si è avviata anche la codifica delle funzionalità opzionali e desiderabili che proseguirà nel periodo successivo alla RQ.

3.1 Tabella del soddisfacimento dei requisiti

Codice	Classe	Stato
R1F1	OB	Soddisfatto
R1F1.1	OB	Soddisfatto
R1F1.2	OB	Soddisfatto
R2F2	DE	Soddisfatto
R1F3	OB	Soddisfatto
R2F4	DE	Non soddisfatto
R1F5	OB	Soddisfatto
R2F6	DE	Non soddisfatto
R1F7	OB	Soddisfatto
R1F7.1	OB	Soddisfatto
R1F7.1.1	OB	Soddisfatto
R2F7.1.2	DE	Non soddisfatto
R2F7.1.3	DE	Non soddisfatto
R1F7.1.4	OB	Soddisfatto
R1F7.1.5	OB	Soddisfatto
R1F7.2	OB	Soddisfatto
R1F7.2.1	OB	Soddisfatto
R1F7.3	OB	Soddisfatto
R3F7.3.1	DE	Soddisfatto
R3F7.3.2	OB	Soddisfatto
R3F7.3.3	OB	Soddisfatto
R1F7.4	OB	Soddisfatto
R2F7.4.1	DE	Soddisfatto

Continua nella pagina successiva...

R3F7.5	OP	Soddisfatto
R3F7.6	OP	Soddisfatto
R3F7.7	OB	Soddisfatto
R3F7.7.1	OB	Soddisfatto
R3F7.7.2	OB	Soddisfatto
R3F8	OP	Non soddisfatto
R3F9	OP	Non soddisfatto
R3F10	OP	Non soddisfatto
R2F11	DE	Non soddisfatto
R3F12	OP	Soddisfatto
R3F13	OP	Soddisfatto
R1F14	OB	Soddisfatto
R1F15	OB	Soddisfatto
R1F15.1	OB	Soddisfatto
R1F15.2	OB	Soddisfatto
R1F15.3	OB	Soddisfatto
R1F15.4	OB	Soddisfatto
R3F15.4.1	OP	Soddisfatto
R3F15.4.2	OP	Soddisfatto
R3F15.5	OP	Soddisfatto
R2F15.6	DE	Soddisfatto
R1F16	OB	Soddisfatto
R2F16.1	DE	Soddisfatto
R2F16.2	DE	Soddisfatto
R2F16.3	DE	Soddisfatto
R1F16.4	DE	Soddisfatto
R1F17	OB	Soddisfatto
R1F18	OB	Soddisfatto

Tabella 1: Tabella del soddisfacimento dei requisiti

3.2 Grafici del soddisfacimento dei requisiti

Riguardo ai requisiti funzionali del prodotto siamo arrivati ad una copertura del 84%; ossia sono stati soddisfatti 43 requisiti su 51.

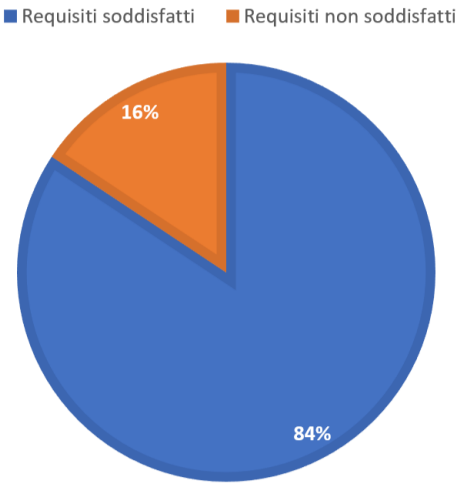

Figura 10: Percentuale dei requisiti funzionali soddisfatti

Riguardo invece ai soli requisiti obbligatori la copertura ha raggiunto il 100%, soddisfando quindi tutti i requisiti individuati.

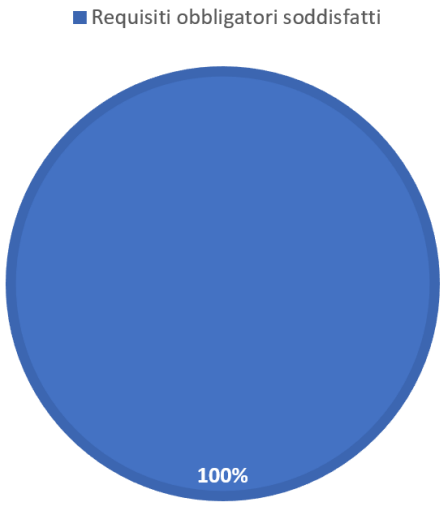

Figura 11: Percentuale dei requisiti funzionali obbligatori soddisfatti