NORRIS Framework

FLAMETECH Inc.

Specifica Tecnica

Informazioni sul documento

Versione	1.0.0
Redazione	Cardin Andrea Faggin Andrea Merlo Gianluca Sartor Michele Zanetti Davide
Verifica	Meneguzzo Francesco
Responsabile	Persegona Mattia
Uso	Esterno
Lista di distribuzione	**FlameTech Inc.** Prof. Vardanega Tullio Prof. Cardin Riccardo

Descrizione

Specifica tecnica e architettura del prodotto sviluppato dal gruppo **FlameTech Inc.** per la realizzazione del progetto Norris.

Stato	Modifica	Autore	Ruolo	Data	Versione
Approvato	Approvazione Documento	Persegona Mattia	Responsabile	2015/04/10	1.0.0
Verificato	Verifica Documento	Meneguzzo Francesco	Verificatore	2015/04/08	0.2.0
In Lavorazione	Correzione errori grammaticali	Faggin Andrea	Progettista	2015/04/08	0.1.1
Verificato	Verifica Documento	Meneguzzo Francesco	Verificatore	2015/04/07	0.1.0
In Lavorazione	Stesa sezione Stime di Fattibilitá	Merlo Gianluca	Progettista	2015/03/30	0.0.11
In Lavorazione	Stesa sezione Tracciamento	Cardin Andrea	Progettista	2015/03/24	0.0.10
In Lavorazione	Stesa sezione Design Pattern	Zanetti Davide	Progettista	2015/03/23	0.0.9
In Lavorazione	Stesa sezione Diagrammi di Attivitá	Sartor Michele	Progettista	2015/03/23	0.0.8
In Lavorazione	Termine stesura sezione Componenti e Classi	Merlo Gianluca	Progettista	2015/03/20	0.0.7
In Lavorazione	Stesa Appendice A	Zanetti Davide	Progettista	2015/03/17	0.0.6
In Lavorazione	Inizio stesura sezione Componenti e Classi	Merlo Gianluca	Progettista	2015/03/16	0.0.5
In Lavorazione	Stesa sezione Descrizione Architettura	Faggin Andrea	Progettista	2015/03/16	0.0.4
In Lavorazione	Stesa sezione Tecnologie Utilizzate	Cardin Andrea	Progettista	2015/03/11	0.0.3
In Lavorazione	Stesa sezione Introduzione	Sartor Michele	Progettista	2015/03/10	0.0.2
In Lavorazione	Inizio stesura scheletro documento	Faggin Andrea	Progettista	2015/03/10	0.0.1

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Il presente documento ha lo scopo di definire la progettazione ad alto livello del progetto Norris.
All'interno del documento verrà presentata l'architettura generale secondo la quale saranno organizzate le varie componenti software e i *design pattern*$_G$ adoperati per la creazione del prodotto. Sarà, inoltre, dettagliato il tracciamento tra le componenti software individuate ed i requisiti.

1.2 Scopo del prodotto

Lo scopo del prodotto è la realizzazione di un *framework*$_G$ per *Node.js*$_G$, compatibile con l'utilizzo standard dei *middleware*$_G$ di *Express*$_G$ in versione 4.x, per la realizzazione rapida di grafici aggiornabili in tempo reale.

1.3 Glossario

Per evitare ogni possibile ambiguità che potrebbe sorgere verrà allegato il *Glossario_ver3.0.0* dove verranno inseriti termini tecnici, acronimi, termini di dominio ed eventuali parole che potrebbero comportare delle incomprensioni o delle ambiguità nella lettura dei documenti. Per rendere la lettura più facile i termini verranno riportati in corsivo ed in pedice verrà posta una "G" maiuscola. (Esempio: *Android*$_G$).

1.4 Riferimenti

1.4.1 Normativi

- **Analisi dei Requisiti:** *AnalisiRequisiti_ver3.0.0*;
- **Norme di Progetto:** *NormeDiProgetto_ver2.0.0*;
- **Capitolato d'appalto C3 Norris:** Node Real-time Intelligence http://www.math.unipd.it/~tullio/IS-1/2014/Progetto/C3.pdf;
- **Verbale d'incontro con il Proponente** in data 2015/03/31.

1.4.2 Informativi

- Presentazione capitolato d'appalto: http://www.math.unipd.it/~tullio/IS-1/2014/Progetto/C3ps.pdf;
- Ingegneria del software - Ian Sommerville - 9a edizione (2011), Parte terza: Advance Software Engineering, Capitolo 18.3: Architectural patterns for distributed systems;
- Design Patterns - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides - 1a edizione italiana (2008);
- Node.js - Marc Wandschneider - 1a edizione (2013).

2 Tecnologie utilizzate

In questa sezione vengono descritte le tecnologie su cui si basa lo sviluppo del progetto. Per ognuna di esse vengono indicati la ragione del loro utilizzo, l'ambito di utilizzo delle tecnologia ed i vantaggi che ne derivano.
All'interno del *Glossario_ver3.0.0* saranno presenti ulteriori dettagli descrittivi sulle tecnologie.

2.1 *Node.js*$_G$

L'utilizzo di *Node.js*$_G$ è vincolato dalla richiesta del Proponente (requisito: RAV1); questo *framework*$_G$ verrà utilizzato per lo sviluppo della componente *back-end*$_G$.

Vantaggi:

- *Node.js*$_G$ è basato su un modello *event-driven*$_G$ con I/O non bloccante. Tale approccio risulta molto efficiente in situazioni critiche di elevato traffico di rete e in applicazioni real-time;
- Supporta l'organizzazione modulare dell'architettura tramite *npm*$_G$. Quest'ultimo infatti permette di importare facilmente i moduli necessari e di combinarli tra loro.

2.2 *Express*$_G$

L'utilizzo di *Express*$_G$ è vincolato dalla richiesta del Proponente (requisito: RAV2); questo *framework*$_G$ verrà utilizzato per la realizzazione dell'infrastruttura per la componente *back-end*$_G$.

Vantaggi:

- *Express*$_G$ semplifica l'uso di *Node.js*$_G$ e offre una migliore implementazione di alcuni aspetti chiave importanti per lo sviluppo del progetto;
- La creazione di *API*$_G$ è resa più facile e veloce grazie ai numerosi moduli e metodi di utilità messi a disposizione da *Express*$_G$.

2.3 *AngularJS*$_G$

Si è deciso di utilizzare *AngularJS*$_G$ come *framework*$_G$ per lo sviluppo della componente *front-end*$_G$.

Vantaggi:

- Incorpora il *MVC*$_G$ lato *client*$_G$; questo rende più facile l'organizzazione della logica dell'architettura complessiva;
- **Approccio dichiarativo:** l'integrazione di questo *framework*$_G$ avviene direttamente nel codice *HTML*$_G$ in modo chiaro e conciso. In particolare, i due aspetti più importanti di questo approccio sono:
  - Creazione di viste dinamiche che effettuano l'aggiornamento automatico tramite *two-way data binding*$_G$, rimuovendo allo sviluppatore parte della complessità della gestione di questo aspetto;

– Gestione delle dipendenze effettuata in maniera dichiarativa e quindi rende facilmente isolabili i comportamenti e le responsabilità dei singoli componenti, semplificando le procedure di test.

2.4 *Chart.js*$_G$

Si è deciso di utilizzare *Chart.js*$_G$ come libreria grafica per la creazione di grafici di tipo *Bar Chart*$_G$ e *Line Chart*$_G$ nella componente *front-end*$_G$.

Vantaggi:

- Non presenta dipendenze verso altri moduli o librerie.
- La libreria è divisa in moduli separati, permettendo di utilizzare solo i moduli necessari.
- I grafici creati sono naturalmente reattivi, il che permette loro di cambiare automaticamente dimensione e granularità a seconda della dimensione della finestra del browser. Questo permette di evitare la gestione di questo aspetto tramite eventuali fogli di stile.
- Supporta l'interattività dei grafici per eventuali sviluppi di funzionalità futuri.

2.5 *DataTables*$_G$

Si è deciso di utilizzare *DataTables*$_G$ come libreria grafica per la creazione di grafici di tipo *Table*$_G$ nella componente *front-end*$_G$.

Vantaggi:

- Supporta numerose tipologie di sorgenti di dati, rendendo la libreria facilmente integrabile nel sistema.
- Presenta funzioni di ricerca e di ordinamento già integrate nelle funzionalità disponibili.
- La presenza di varie opzioni di configurazione per le tabelle, oltre che di numerose estensioni, facilita eventuali sviluppi di funzionalità futuri.

2.6 *Google Maps API*$_G$

Si è deciso di utilizzare *Google Maps API*$_G$ come libreria grafica per la creazione di grafici di tipo *Map Chart*$_G$ nella componente *front-end*$_G$ e nell'applicativo *Android*$_G$.

Vantaggi:

- Diffusione ampia, largo supporto e documentazione ricca; questo rende preferibile l'utilizzo di questa libreria rispetto ad eventuali alternative.
- Presenta sia una libreria *JavaScript*$_G$ da utilizzare per la componente *front-end*$_G$, sia una libreria *Java*$_G$ per l'applicativo *Android*$_G$ utilizzabile ancora più semplicemente in quanto praticamente nativa.

L'utilizzo di questa tecnologia presenta anche un possibile svantaggio:

- L'utilizzo è gratuito esclusivamente con un volume di utenza ridotto, in particolare la richiesta giornaliera di caricamento delle mappe deve essere inferiore a 25000.

Tuttavia, data la natura principalmente dimostrativa del progetto, è stato stimato che un tale limite al volume di richieste è accettabile in quanto sarà difficilmente raggiungibile.
Nel caso si verifichi un futuro utilizzo di Norris che risulta particolarmente scalabile, fino al superamento del limite di richieste, sarà compito dello sviluppatore interessato preoccuparsi di questo aspetto.

2.7 *MPAndroidChart*$_G$

Si è deciso di utilizzare *MPAndroidChart*$_G$ come libreria grafica per la creazione di grafici di tipo *Bar Chart*$_G$ e *Line Chart*$_G$ nell'applicativo *Android*$_G$.

Vantaggi:

- Fornisce numerose funzionalità riguardanti la veste grafica e l'interazione con i grafici creati.
- Compatibile con le versioni di *Android*$_G$ richieste e semplice integrazione.

2.8 *Socket.io*$_G$

L'utilizzo di *Socket.io*$_G$ è vincolato dalla richiesta del Proponente (requisito: RAV3); questa libreria verrà utilizzata per la realizzazione della componente che gestisce le notifiche *push*$_G$.

Vantaggi:

- Semplifica le conoscenze necessarie per la gestione del protocollo *WebSocket*$_G$;
- Fornisce una libreria *JavaScript*$_G$ per la componente *back-end*$_G$ e una per la parte *front-end*$_G$, il che lo rende facilmente integrabile nel progetto.
- Sono disponibili diverse librerie *Java*$_G$ per l'applicativo *Android*$_G$, il che rimuove eventuali problemi di integrazione di librerie scritte in linguaggi diversi.

2.9 *JSON*$_G$

Si è deciso di utilizzare *JSON*$_G$ come formato per lo scambio dei dati tra *back-end*$_G$ e *front-end*$_G$ o applicativo *Android*$_G$.

Vantaggi:

- *JavaScript*$_G$ rende l'utilizzo di *JSON*$_G$ semplice ed immediato, praticamente escludendo ogni altra scelta da questa decisione.

2.10 *HTML5*$_G$

Si è deciso di utilizzare *HTML5*$_G$ come linguaggio per la creazione della struttura delle pagine web.

Vantaggi:

- Consente di integrare facilmente le funzionalità di *AngularJS*$_G$;
- Strutturato per offrire codice più pulito rispetto alle versioni precedenti;
- Compatibile con le versioni dei browser richiesti (requisiti: RAV4 e RAV5).

3 Descrizione architettura

3.1 Metodo e formalismo di specifica

Nell'esposizione dell'architettura di Norris si procederà con un approccio *top-down*$_G$, descrivendo l'architettura iniziando dal generale ed andando al particolare. Si procederà quindi alla descrizione dei *package*$_G$ e dei componenti, per poi descrivere nel dettaglio le singole classi, specificando per ognuna una descrizione, il suo utilizzo e le relazioni in ingresso ed in uscita. Successivamente si illustreranno degli esempi di uso dei *design pattern*$_G$ nell'architettura del sistema, rimandando la spiegazione generale degli stessi all'appendice *A - Descrizione Design pattern*$_G$
Per i diagrammi di *package*$_G$, di classe e di attività si utilizzerà il formalismo *UML*$_G$ 2.x. Nel riportare i diagrammi di *package*$_G$ e di classe si farà uso, dove appropriato, dei colori per aiutare la distinzione tra componenti diversi. Si noti in particolare che le classi di colore verde appartengono a librerie e componenti esterni e sono quindi da considerarsi fuori dai *package*$_G$, anche se riportate all'interno in alcuni diagrammi per maggior chiarezza.
Nel trattare i componenti, si chiarisce che sono da intendersi come *package*$_G$ e i due termini verranno quindi usati come sinonimi.
Le classi astratte potrebbero non contenere metodi astratti in quanto questi verranno specificati durante la fase D. Da notare, inoltre, che progettare il sistema con un'architettura ad oggetti classica non permette di rappresentare in modo naturale la gestione dinamica dei tipi e le caratteristiche tipiche degli stili di programmazione funzionali.
In certi casi, pertanto, è stato necessario introdurre interfacce e classi "fittizie", che non verranno codificate. Dato che questo introduce numerosi schematismi che appesantiscono i diagrammi e che non sono richiesti dal linguaggio di programmazione, si è cercato di limitarli soltanto ai casi in cui sono particolarmente utili.

3.2 Architettura generale

Il progetto è composto da tre parti: una componente *client*$_G$ web, costituita dal browser degli utenti che visualizzeranno le pagine *front-end*$_G$, una componente AndroidApp che costituisce l'applicazione per *smartphone*$_G$ *Android*$_G$ e una componente web *server*$_G$ costituita dalla sezione *back-end*$_G$ di Norris.

3.2.1 Interfaccia REST-like

Per l'interfaccia della componente *back-end*$_G$ di Norris si è scelto di utilizzare uno stile REST-like, ovvero basato sullo stile *REST*$_G$ ma modificato per permettere il recupero di una risorsa. I motivi che hanno spinto alla scelta di *REST*$_G$ sono:

- Semplicità di utilizzo;
- Facile integrazione con i *framework*$_G$ esistenti (*AngularJS*$_G$ e *Express*$_G$);
- Indipendenza dal linguaggio di programmazione utilizzato.

REST$_G$ utilizza il concetto di risorsa, ovvero un aggregato di dati con un nome (*URI*$_G$) e una rappresentazione, su cui è possibile invocare le operazioni *CRUD*$_G$ tramite la seguente corrispondenza:

Risorsa	***URI*$_G$ di una pagina** es. http://example.com/page/	***URI*$_G$ di un grafico** es. http://example.com/graph/
GET	Ritorna un oggetto pagina contenente grafici.	Ritorna i dati che rappresentano un oggetto grafico.
POST	Non usato	Non usato
PUT	Non usato	Non usato
DELETE	Non usato	Non usato

Tabella 2: Chiamate URI

Per il formato di rappresentazione dei dati è stato scelto *JSON*$_G$, in quanto si integra molto facilmente con i *framework*$_G$ utilizzati e con il linguaggio *JavaScript*$_G$, a differenza di XML o CSV che richiederebbero l'utilizzo di librerie specifiche.

3.2.2 Norris

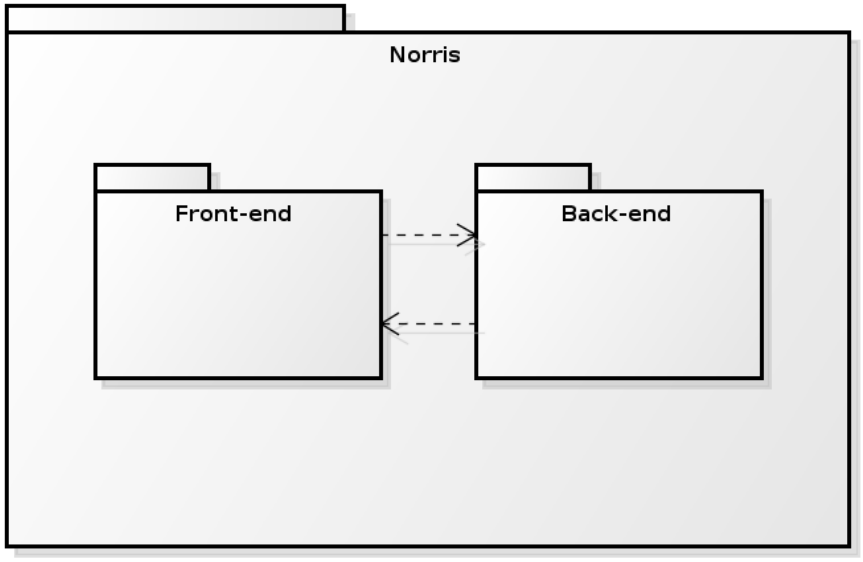

Figura 1: Diagramma dei *package*$_G$ Norris

L'architettura del sistema è divisa in due *package*$_G$ principali, a loro volta divisi in sotto *package*$_G$:

- **Back-end**: si occuperà della gestione logica del *framework*$_G$;
- **Front-end**: si occuperà di fornire un'interfaccia visualizzabile ai *client*$_G$ web.

3.2.2.1 Back-end

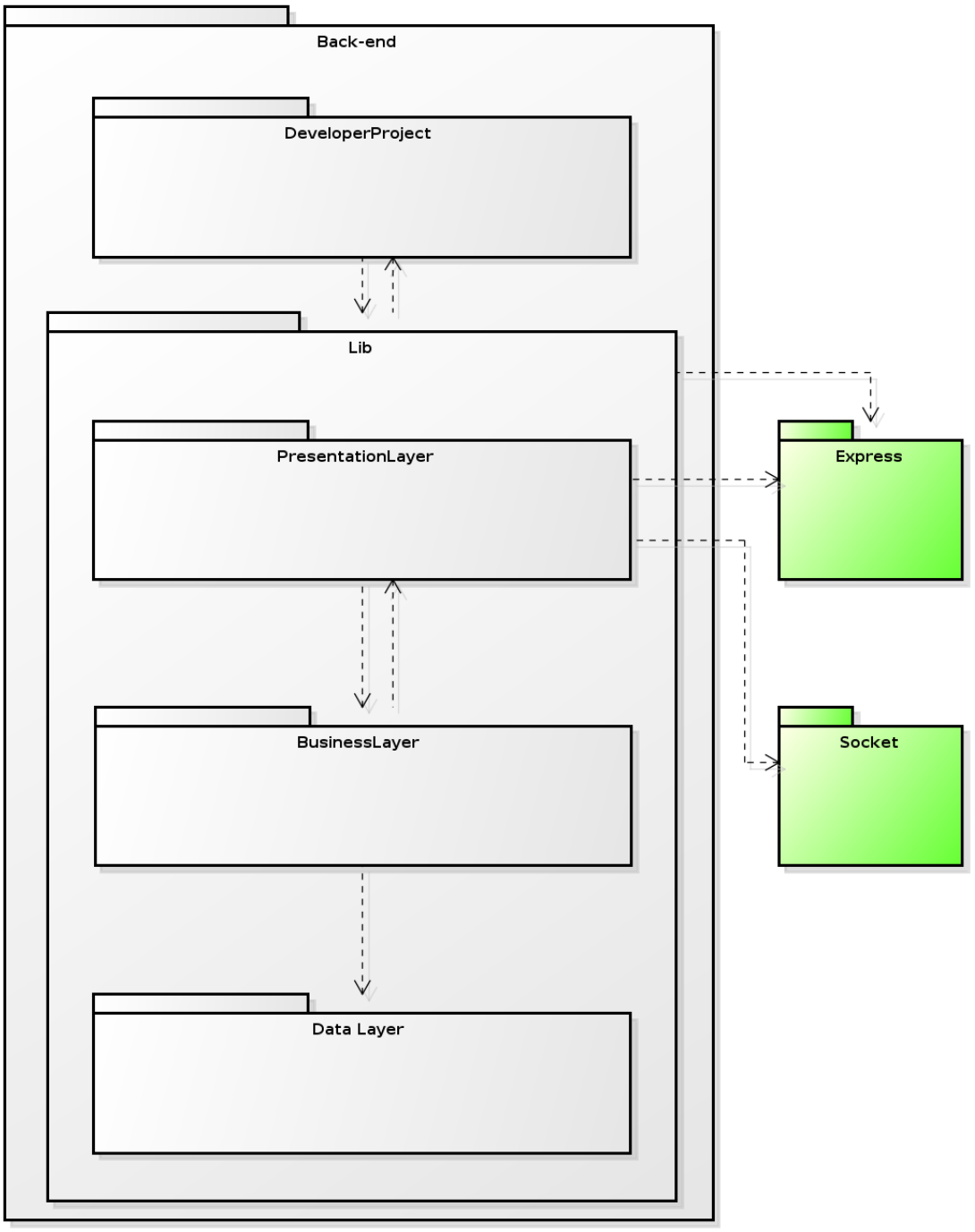

Figura 2: Diagramma dei *package*$_G$ Back-end

Il *package*$_G$ Back-end contiene le componenti necessarie al *framework*$_G$ per la gestione delle funzionalità sviluppatore, tra cui la creazione e la modifica di pagine e grafici, e le funzionalità di comunicazione con i *client*$_G$, quali browser web e applicativo *Android*$_G$.

L'architettura del *back-end*$_G$ è suddivisa in due *package*$_G$ principali:

- **DeveloperProject**: si occuperà di fornire la configurazione e avviare il *server*$_G$ web di Norris.

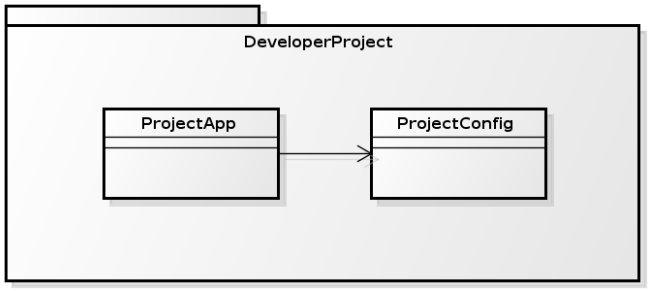

Figura 3: Diagramma del *package*$_G$ DeveloperProject

- **Lib**: costituisce la libreria principale del *framework*$_G$ Norris.
  Per il suo sviluppo è stato utilizzato il *design pattern*$_G$ *Three Tier Architecture*$_G$ in quanto ritenuto più vantaggioso per la suddivisione logica delle componenti presenti al suo interno.
  - **PresentationLayer**: in questo layer sono presenti le classi che consentiranno l'accesso alle diverse funzioni di Norris da parte dello sviluppatore. Gli utenti finali invieranno, tramite il browser, una richiesta che verrà presa in consegna dalla classe RequestHandler che si occuperà di indirizzarla, tramite il *Chain of Responsibility*$_G$, alla classe corretta. Lo sviluppatore accederà alla classe SourceHandler con la quale avrà la possibilità di richiamare funzionalità da altri layer. La classe LogPrinter si occuperà di eseguire la stampa dei log di errore, mentre la classe SocketHandler di inviare gli aggiornamenti in *push*$_G$ ai diversi *client*$_G$.
    Al di fuori da questo layer non sarà possibile accedere al *package*$_G$ Lib, sia per rispettare l'architettura del *design pattern*$_G$ utilizzato, sia per garantire un maggior grado di sicurezza e incapsulamento.

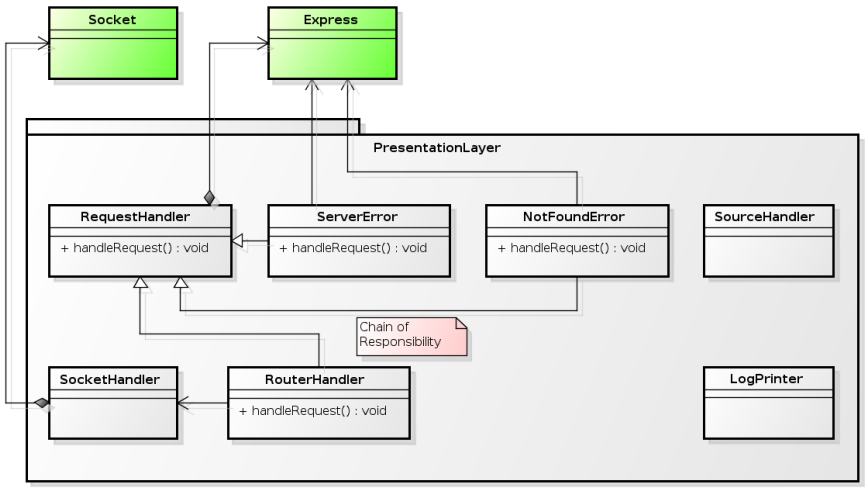

Figura 4: Diagramma del PresentationLayer

– **BusinessLayer**: in questo layer sono presenti le classi che si occuperanno di eseguire tutte le reali computazioni e operazioni logiche. La classe Router si occuperà della gestione della prima richiesta di pagina, le classi Builder e Updater si occuperanno delle operazioni eseguite dallo sviluppatore sui grafici, la prima in fase di creazione e la seconda in fasi successive. La classe UpdateObserver fa parte del *design pattern*$_G$ *Observer*$_G$ e si occuperà di notificare alla classe ClientUpdater che è stato eseguito un aggiornamento su un grafico ed è necessario eseguire un *push*$_G$ di quest'ultimo tramite *Socket*$_G$. La classe LogManager si occuperà di richiamare la classe LogPrinter in caso sia necessario stampare un log di errore.
Allo scopo di un corretto utilizzo della *Three Tier Architecture*$_G$ le operazioni che richiedono l'interazione fra PresentationLayer e DataLayer devono, in qualunque caso, passare per una o più classi di questo layer.

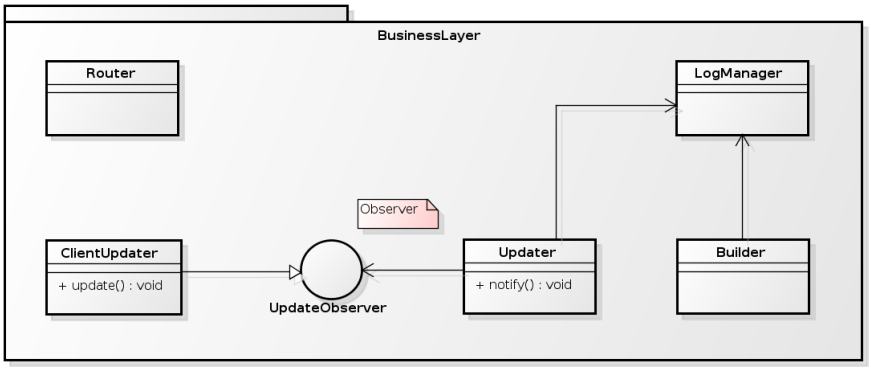

Figura 5: Diagramma del BusinessLayer

– **DataLayer**: in questo layer sono presenti le classi che rappresentano la struttura dati e gli oggetti veri e propri del Lib. La classe DataHandler contiene le funzioni che permetteranno allo sviluppatore di andare ad accedere alle funzioni di creazione sia della classe Graph che della classe Page. La prima classe è la factory del *design pattern*$_G$ *Factory Method*$_G$ utilizzato per la creazione delle classi successive rappresentanti i singoli grafici, la seconda rappresenta la classe che svolgerà la funzione di contenitore per gli oggetti ottenuti dalla prima. Delle ultime due classi restanti la classe Exception conterrà tutti i possibili errori che si possono riscontrare in fase di utilizzo delle classi presenti in DataLayer, mentre la classe ActivePage conterrà una lista di quali grafici attivi sono presenti all'interno del *server*$_G$ tracciati con i *client*$_G$ che ne stanno facendo uso.

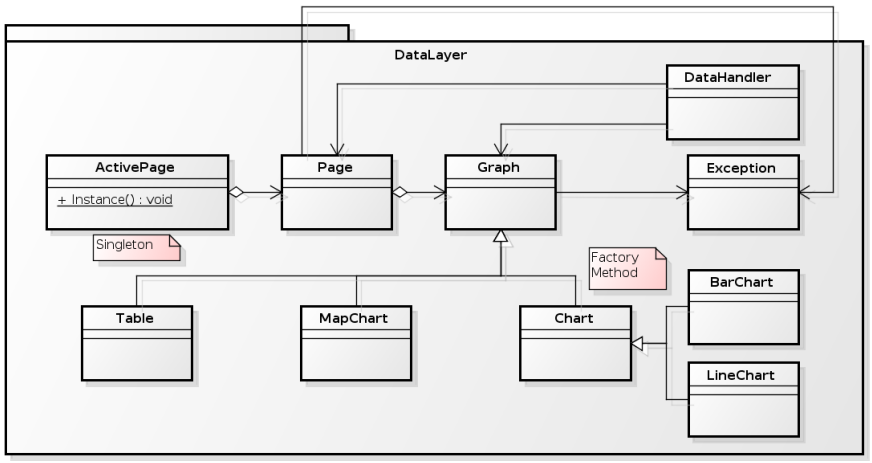

Figura 6: Diagramma del DataLayer

3.2.2.2 Front-end

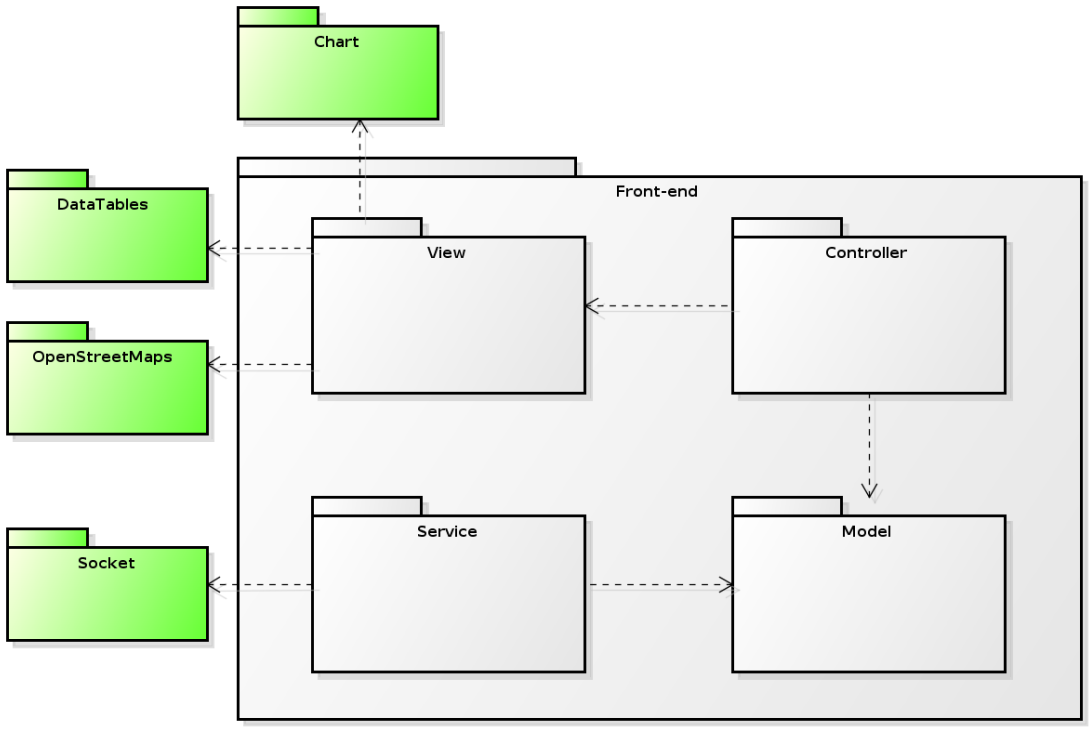

Figura 7: Diagramma dei *package*$_G$ Front-end

Questa parte dell'architettura si occupa di contenere la sezione *front-end*$_G$ Comprende il sottosistema che viene eseguito nei browser degli utenti e che fornisce l'interfaccia grafica all'utente finale che visualizzerà le pagine create dallo sviluppatore, ed è suddivisa in quattro *package*$_G$ principali:

- **View**: *package*$_G$ comprendente le classi che costituiscono la view relativa al *design pattern*$_G$ *MVC*$_G$ del componente Front-end. Ogni view rappresenta un tipo di grafico, che verrà popolato con i dati richiesti.

- **Controller**: *package*$_G$ comprendente le classi che costituiscono i controller relativi al *design pattern*$_G$ *MVC*$_G$ del componente Front-end. Ogni controller gestisce le operazioni e la logica applicativa riguardante un determinato tipo di grafico, e specifica quale view verrà aggiornata per la presentazione del grafico all'utente.
- **Model**: *package*$_G$ che comprende le classi dei modelli relativi al *design pattern*$_G$ *MVC*$_G$, dei dati utilizzati dal *front-end*$_G$. Servono a fornire al Controller e al Service i dati ricevuti dal *back-end*$_G$.
- **Service**: *package*$_G$comprendente le classi che descrivono i meccanismi con cui il *front-end*$_G$ può interfacciarsi con il *back-end*$_G$. Permette di ricevere i dati da inserire nel Model.

3.2.3 Applicativo *Android*$_G$

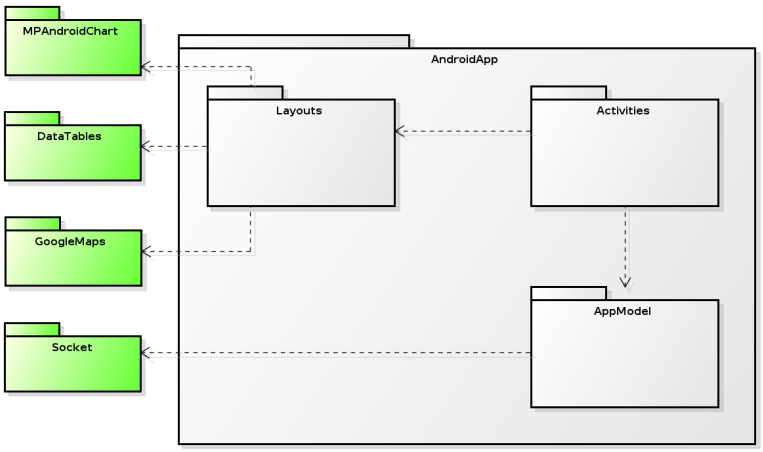

Figura 8: Diagramma dei *package*$_G$ applicazione *Android*$_G$

Questa parte contiene l'applicazione *Android*$_G$ ed è suddivisa in tre *package* rispecchiando il *design pattern*$_G$ *MVC*$_G$. In particolare il *package*$_G$ Layouts corrisponde alla view, AppModel corrisponde alla parte model ed il *package*$_G$ Activities corrisponde al controller.

- **Layouts**: *package*$_G$ comprendente le classi che costituiscono le singole view dell'applicazione. Ogni view rappresenta una sezione come ad esempio: la schermata principale, il menu, le impostazioni.
- **Activities**: *package*$_G$ comprendente le classi che gestiscono le operazioni e la logica applicativa. Ogni classe rappresenta una specifica attività che può essere eseguita dall'utilizzatore finale.
- **AppModel**: *package*$_G$ che comprende le classi dei modelli dei dati. Servono a fornire alle activities i dati necessari al funzionamento dell'applicazione e i dati ricevuti dal *back-end*$_G$ per la costruzione dei grafici.

4 Componenti e classi

Le componenti e le loro classi saranno, per garantire maggior chiarezza e facilità di lettura, suddivise tra *framework*$_G$ e applicazione *Android*$_G$.

4.1 Componente Norris *framework*$_G$

4.1.1 Norris

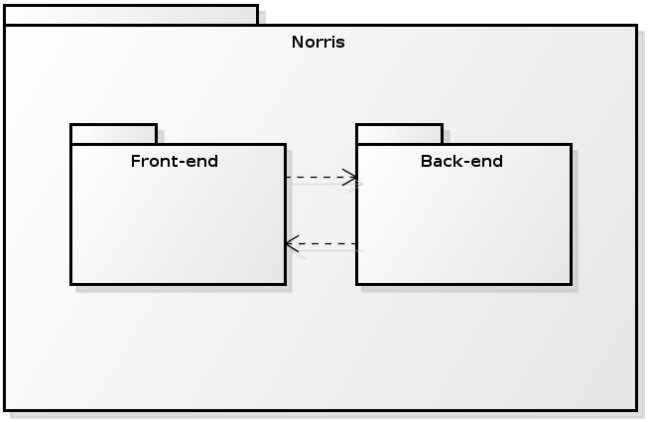

Figura 9: Diagramma del package Norris

4.1.1.1 Informazioni sul *package*$_G$

4.1.1.1.1 Descrizione

Questo *package*$_G$ contiene tutte le componenti del *framework*$_G$ Norris.

4.1.1.1.2 *Package*$_G$ contenuti

- Norris::Back-end;
- Norris::Front-end.

4.1.2 Norris::Back-end

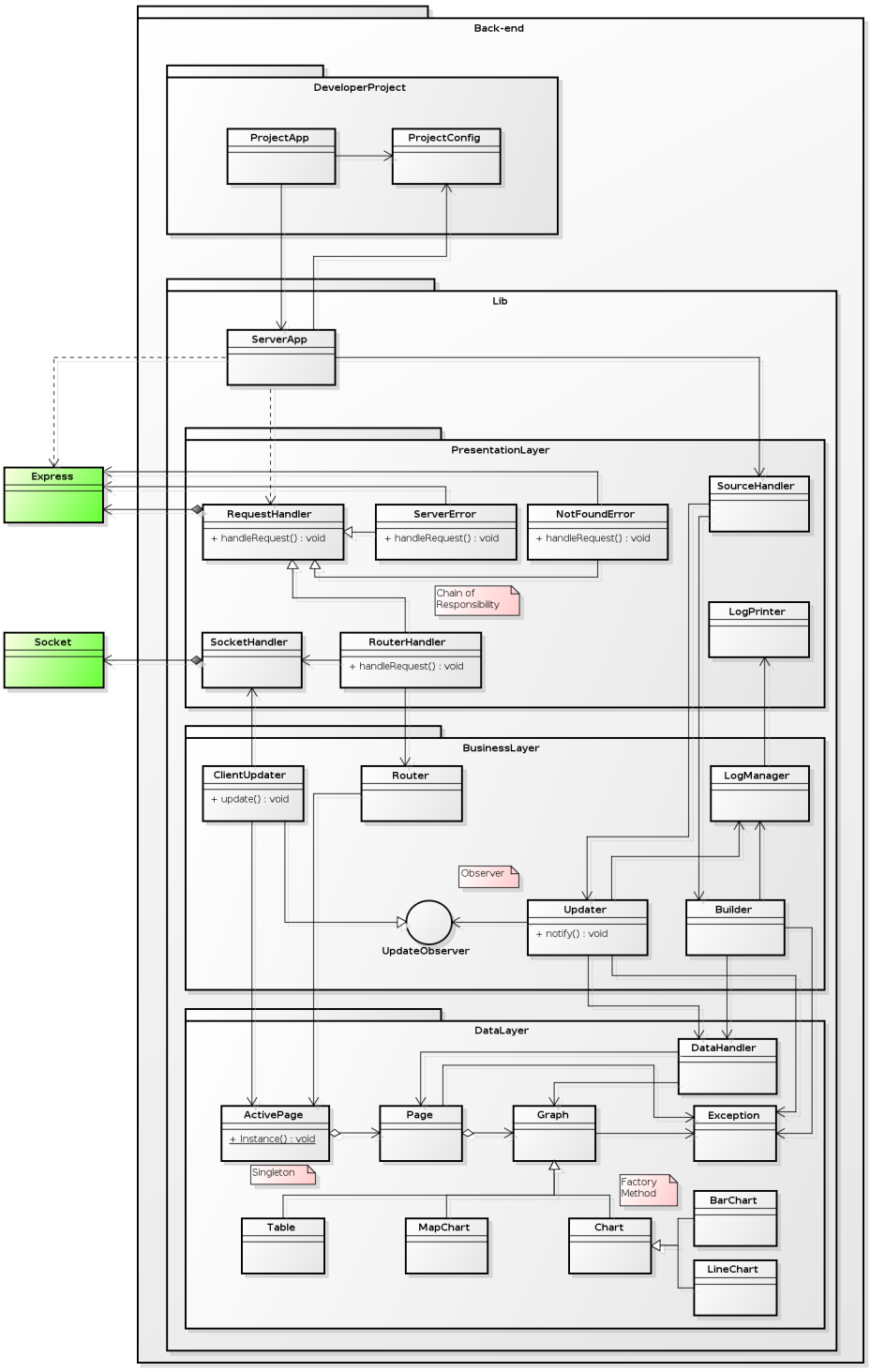

Figura 10: Diagramma delle classi Back-end

4.1.2.1 Informazioni sul *package*$_G$

4.1.2.1.1 Descrizione

Questo *package*$_G$ contiene tutte le classi per la gestione di Norris dal lato *server*$_G$.

4.1.2.1.2 *Package*$_G$ contenuti

- Norris::Back-end::DeveloperProject;
- Norris::Back-end::Lib.

4.1.3 Norris::Back-end::DeveloperProject

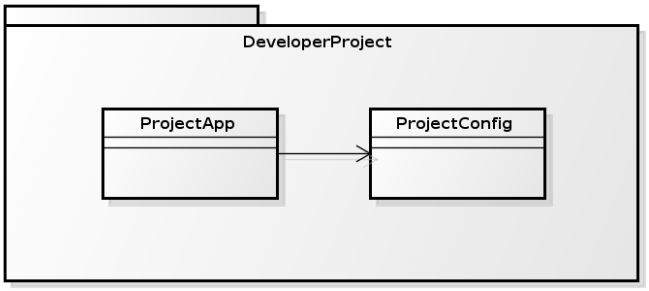

Figura 11: Componente DeveloperProject

4.1.3.1 Informazioni sul *package*$_G$

4.1.3.1.1 Descrizione

Questo *package*$_G$ contiene le classi che permettono allo sviluppatore di creare un nuovo progetto Norris e di configurarne le caratteristiche.

4.1.3.2 Classi

4.1.3.2.1 Norris::Back-end::DeveloperProject::ProjectApp

Descrizione

Questa classe è modificabile dallo sviluppatore, il quale si occupa di configurare e avviare il *server*$_G$ dell'applicazione.

Utilizzo

Implicitamente avvia il *server*$_G$ utilizzando la classe Lib::ServerApp, a cui passa i parametri di configurazione del progetto definiti con un oggetto della classe ProjectConfig.

Relazioni con altre classi

- **Norris::Back-end::DeveloperProject::ProjectConfig**

  Relazione uscente. Classe per la configurazione del progetto.
- **Norris::Back-end::Lib::ServerApp**

  Relazione uscente. Classe che attiva il *server*$_G$.

4.1.3.2.2 Norris::Back-end::DeveloperProject::ProjectConfig

Descrizione

Questa classe contiene la configurazione di un'istanza di Norris.

Utilizzo

Viene passata come parametro al costruttore della classe Lib::ServerApp per configurare l'applicazione.

Relazioni con altre classi

- **Norris::Back-end::DeveloperProject::ProjectApp**
  Relazione entrante. Classe per la configurazione del progetto.
- **Norris::Back-end::Lib::ServerApp**
  Relazione entrante. Classe per la configurazione del progetto.

4.1.4 Norris::Back-end::Lib

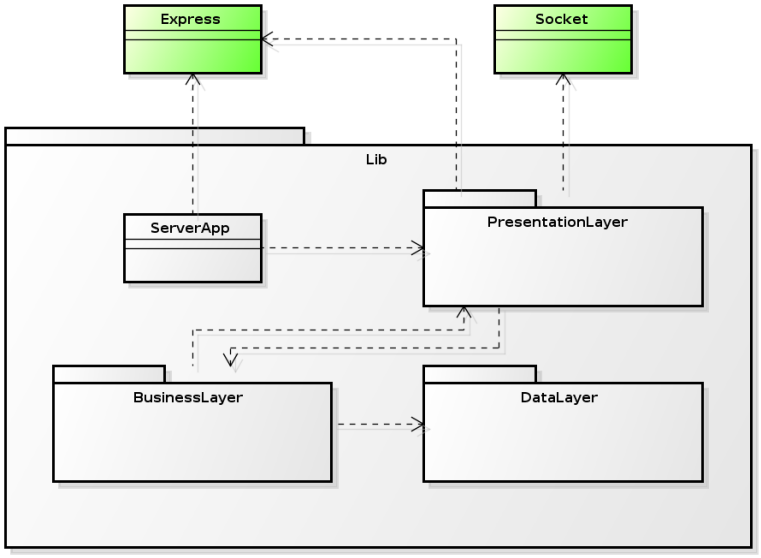

Figura 12: Componente Lib

4.1.4.1 Informazioni sul *package*$_G$

4.1.4.1.1 Descrizione

Questo *package*$_G$ contiene la libreria del *framework*$_G$ Norris.

4.1.4.1.2 *Package*$_G$ contenuti

- Norris::Back-end::Lib::BusinessLayer;
- Norris::Back-end::Lib::DataLayer;
- Norris::Back-end::Lib::PresentationLayer.

4.1.4.2 Classi

4.1.4.2.1 Norris::Back-end::Lib::ServerApp

Descrizione

Questa classe si occupa di avviare il *server*$_G$ e di invocare il *middleware*$_G$ di *Express*$_G$ permettendo cosi l'avvio dell'applicazione.

Utilizzo

Viene utilizzato per avviare l'applicazione. Internamente inizializza la catena di gestione delle chiamate utilizzando la classe PresentationLayer::RequestHandler.

Relazioni con altre classi

- **Norris::Back-end::DeveloperProject::ProjectApp**

  Relazione entrante. Classe che attiva il *server*$_G$.
- **Norris::Back-end::DeveloperProject::ProjectConfig**

  Relazione uscente. Classe per la configurazione del progetto.
- **Norris::Back-end::Lib::PresentationLayer::RequestHandler**

  Relazione uscente. Classe per la gestione delle richieste di pagina.
- **Norris::Back-end::Lib::PresentationLayer::SourceHandler**

  Relazione uscente. Classe per la gestione delle chiamate dello sviluppatore.

4.1.5 Norris::Back-end::Lib::BusinessLayer

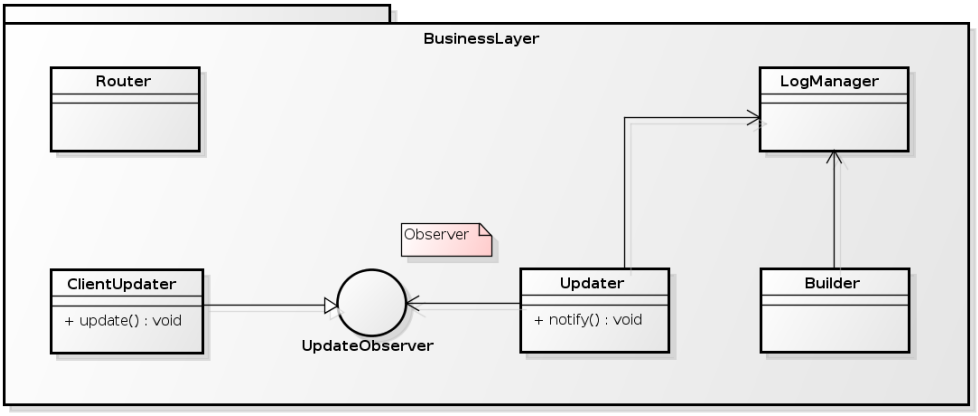

Figura 13: Componente BusinessLayer

4.1.5.1 Informazioni sul *package*$_G$

4.1.5.1.1 Descrizione

Questo *package*$_G$ contiene le classi che permettono al *framework*$_G$ di gestire la comunicazione tra PresentationLayer e DataLayer.

4.1.5.2 Classi

4.1.5.2.1 Norris::Back-end::Lib::BusinessLayer::Builder

Descrizione

Questa classe comprende i metodi per la creazione di grafici e pagine.

Utilizzo

Sarà utilizzata da PresentationLayer::SourceHandler per l'utilizzo delle API_G per la creazione dei grafici e delle pagine.

Relazioni con altre classi

- **Norris::Back-end::Lib::PresentationLayer::SourceHandler**
  Relazione entrante. Classe che effettua la costruzione di grafici e pagine.
- **Norris::Back-end::Lib::DataLayer::DataHandler**
  Relazione uscente. Classe per l'invocazione delle API_G.
- **Norris::Back-end::Lib::DataLayer::Exception**
  Relazione uscente. Classe che si occupa di gestire gli errori.
- **Norris::Back-end::Lib::BusinessLayer::LogManager**
  Relazione uscente. Classe per la gestione dei log di errore.

4.1.5.2.2 Norris::Back-end::Lib::BusinessLayer::ClientUpdater

Descrizione

Questa classe rappresenta la classe concreta del *design pattern*$_G$ *Observer*$_G$ e si occupa della gestione degli aggiornamenti dei grafici ai *client*$_G$.

Utilizzo

Sarà utilizzata dall'interfaccia UpdateObserver in seguito all'aggiornamento dei dati sui grafici.

Classi ereditate

- Norris::Back-end::Lib::BusinessLayer::UpdateObserver.

Relazioni con altre classi

- **Norris::Back-end::Lib::DataLayer::ActivePage**
  Relazione uscente. Classe mantiene la lista delle pagine attive.
- **Norris::Back-end::Lib::PresentationLayer::SocketHandler**
  Relazione uscente. Questa classe si occupa della gestione delle connessioni tramite *Socket.io*$_G$.

4.1.5.2.3 Norris::Back-end::Lib::BusinessLayer::LogManager

Descrizione

Questa classe comprende i metodi per la gestione del log degli errori.

Utilizzo

Sarà utilizzata dalle classi Updater e Builder.

Relazioni con altre classi

- **Norris::Back-end::Lib::BusinessLayer::Builder**
  Relazione entrante. Classe per la gestione dei log di errore.
- **Norris::Back-end::Lib::BusinessLayer::Updater**
  Relazione entrante. Classe per la gestione dei log di errore.
- **Norris::Back-end::Lib::PresentationLayer::LogPrinter**
  Relazione uscente. Classe per l'output dei log di errore.

4.1.5.2.4 Norris::Back-end::Lib::BusinessLayer::Router

Descrizione

Questa classe contiene i metodi che effettuano il *routing*$_G$ delle richieste provenienti dai *client*$_G$.

Utilizzo

Sarà utilizzata dalla classe PresentationLayer::RouterHandler per la gestione delle richieste valide.

Relazioni con altre classi

- **Norris::Back-end::Lib::PresentationLayer::RouterHandler**
  Relazione entrante. Classe che si occupa del *routing*$_G$ delle chiamate.
- **Norris::Back-end::Lib::DataLayer::ActivePage**
  Relazione uscente. Classe mantiene la lista delle pagine attive.

4.1.5.2.5 Norris::Back-end::Lib::BusinessLayer::UpdateObserver

Descrizione

Questa classe è l'interfaccia del *design pattern*$_G$ *Observer*$_G$ per l'aggiornamento dei *client*$_G$.

Utilizzo

Questa interfaccia sarà utilizzata dalla classe Updater.

Classi figlie

- Norris::Back-end::Lib::BusinessLayer::ClientUpdater.

Relazioni con altre classi

- **Norris::Back-end::Lib::BusinessLayer::Updater**
  Relazione entrante. Questa classe notifica gli aggiornamenti dei grafici.

4.1.5.2.6 Norris::Back-end::Lib::BusinessLayer::Updater

Descrizione

Questa classe comprende i metodi per la gestione degli aggiornamenti su grafici e pagine.

Utilizzo

Sarà utilizzata da PresentationLayer::SourceHandler per l'invocazione delle API_G per la gestione degli aggiornamenti su grafici e pagine.

Relazioni con altre classi

- **Norris::Back-end::Lib::PresentationLayer::SourceHandler**
  Relazione entrante. Classe che effettua l'aggiornamento di grafici e pagine.
- **Norris::Back-end::Lib::DataLayer::DataHandler**
  Relazione uscente. Classe per l'invocazione delle API_G.
- **Norris::Back-end::Lib::DataLayer::Exception**
  Relazione uscente. Classe che si occupa di gestire gli errori.
- **Norris::Back-end::Lib::BusinessLayer::LogManager**
  Relazione uscente. Classe per la gestione dei log di errore.
- **Norris::Back-end::Lib::BusinessLayer::UpdateObserver**
  Relazione uscente. Questa classe notifica gli aggiornamenti dei grafici.

4.1.6 Norris::Back-end::Lib::DataLayer

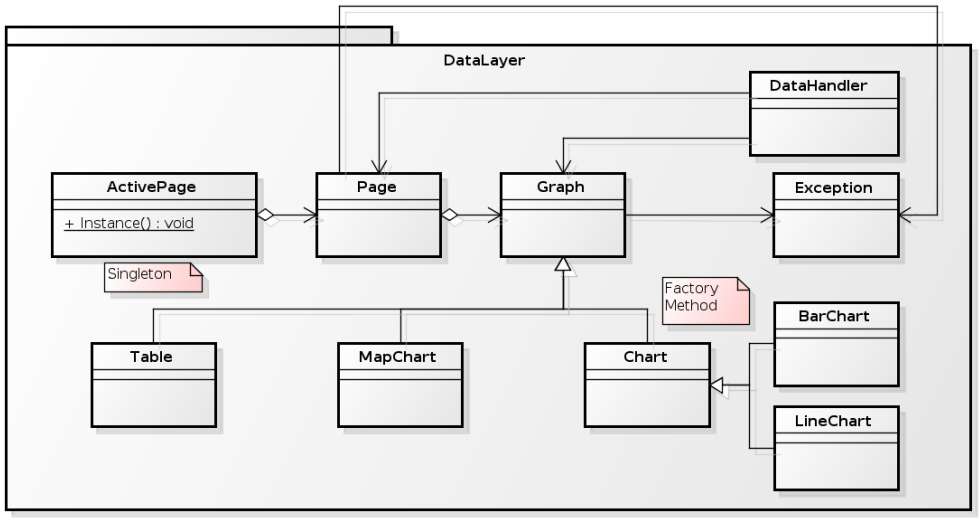

Figura 14: Componente DataLayer

4.1.6.1 Informazioni sul *package*$_G$

4.1.6.1.1 Descrizione

Questo *package*$_G$ include le classi che contengono e gestiscono la struttura dei dati.

4.1.6.2 Classi

4.1.6.2.1 Norris::Back-end::Lib::DataLayer::ActivePage

Descrizione

Questa classe contiene i dati delle pagine attualmente attive in una specifica istanza di Norris.

Utilizzo

Sarà utilizzata dalle classi BusinessLayer::Router e BusinessLayer::ClientUpdater per individuare la corrispondenza tra pagine e indirizzi URL_G.

Relazioni con altre classi

- **Norris::Back-end::Lib::BusinessLayer::ClientUpdater**
  Relazione entrante. Classe mantiene la lista delle pagine attive.
- **Norris::Back-end::Lib::BusinessLayer::Router**
  Relazione entrante. Classe mantiene la lista delle pagine attive.
- **Norris::Back-end::Lib::DataLayer::Page**
  Relazione uscente. Classe per l'invocazione delle API_G riguardanti le pagine.

4.1.6.2.2 Norris::Back-end::Lib::DataLayer::BarChart

Descrizione

Questa classe rappresenta il grafico di tipo *Bar Chart*$_G$, è derivata dalla classe Chart.

Utilizzo

Sarà utilizzata dallo sviluppatore per la creazione e la modifica di grafici di tipo *Bar Chart*$_G$.

Classi ereditate

- Norris::Back-end::Lib::DataLayer::Chart.

4.1.6.2.3 Norris::Back-end::Lib::DataLayer::Chart

Descrizione

Questa classe contiene le prorietà e i metodi comuni ai grafici di tipo *Bar Chart*$_G$ e *Line Chart*$_G$, è derivata dalla classe Graph.

Utilizzo

Sarà utilizzata dalle sottoclassi tramite la derivazione di dati e metodi.

Classi ereditate

- Norris::Back-end::Lib::DataLayer::Graph.

Classi figlie

- Norris::Back-end::Lib::DataLayer::BarChart;
- Norris::Back-end::Lib::DataLayer::LineChart.

4.1.6.2.4 Norris::Back-end::Lib::DataLayer::DataHandler

Descrizione

Questa classe contiene i metodi che verranno invocati dal BusinessLayer per l'utilizzo delle API_G.

Utilizzo

Sarà utilizzata dalle classi BusinessLayer::Updater e BusinessLayer::Builder per l'invocazione delle API_G di Norris.

Relazioni con altre classi

- **Norris::Back-end::Lib::BusinessLayer::Builder**

  Relazione entrante. Classe per l'invocazione delle API_G.
- **Norris::Back-end::Lib::BusinessLayer::Updater**

  Relazione entrante. Classe per l'invocazione delle API_G.
- **Norris::Back-end::Lib::DataLayer::Graph**

  Relazione uscente. Classe per l'invocazione delle API_G riguardanti i grafici.
- **Norris::Back-end::Lib::DataLayer::Page**

  Relazione uscente. Classe per l'invocazione delle API_G riguardanti le pagine.

4.1.6.2.5 Norris::Back-end::Lib::DataLayer::Exception

Descrizione

Questa classe contiene le eccezioni riguardanti le API_G del $framework_G$ Norris .

Utilizzo

Sarà utilizzata da tutte le classi che si occupano della creazione e della modifica di grafici e pagine nel caso si verifichino eccezioni.

Relazioni con altre classi

- **Norris::Back-end::Lib::BusinessLayer::Builder**

  Relazione entrante. Classe che si occupa di gestire gli errori.
- **Norris::Back-end::Lib::DataLayer::Graph**

  Relazione entrante. Classe che si occupa di gestire gli errori.
- **Norris::Back-end::Lib::DataLayer::Page**

  Relazione entrante. Classe che si occupa di gestire gli errori.
- **Norris::Back-end::Lib::BusinessLayer::Updater**

  Relazione entrante. Classe che si occupa di gestire gli errori.

4.1.6.2.6 Norris::Back-end::Lib::DataLayer::Graph

Descrizione

Questa classe contiene le proprietà e i metodi comuni a tutti i grafici.

Utilizzo

Sarà utilizzata dalle sottoclassi tramite la derivazione di dati e metodi.

Classi figlie

- Norris::Back-end::Lib::DataLayer::Chart;
- Norris::Back-end::Lib::DataLayer::MapChart;
- Norris::Back-end::Lib::DataLayer::Table.

Relazioni con altre classi

- **Norris::Back-end::Lib::DataLayer::DataHandler**
  Relazione entrante. Classe per l'invocazione delle API_G riguardanti i grafici.
- **Norris::Back-end::Lib::DataLayer::Page**
  Relazione entrante. Classe per l'invocazione delle API_G riguardanti i grafici.
- **Norris::Back-end::Lib::DataLayer::Exception**
  Relazione uscente. Classe che si occupa di gestire gli errori.

4.1.6.2.7 Norris::Back-end::Lib::DataLayer::LineChart

Descrizione

Questa classe rappresenta il grafico di tipo *Line Chart*$_G$, è derivata dalla classe Chart.

Utilizzo

Sarà utilizzata dallo sviluppatore per la creazione e la modifica di grafici di tipo *Line Chart*$_G$.

Classi ereditate

- Norris::Back-end::Lib::DataLayer::Chart.

4.1.6.2.8 Norris::Back-end::Lib::DataLayer::MapChart

Descrizione

Questa classe rappresenta il grafico di tipo *Map Chart*$_G$, è derivata dalla classe Graph.

Utilizzo

Sarà utilizzata dallo sviluppatore per la creazione e la modifica di grafici di tipo *Map Chart*$_G$.

Classi ereditate

- Norris::Back-end::Lib::DataLayer::Graph.

4.1.6.2.9 Norris::Back-end::Lib::DataLayer::Page

Descrizione

Questa classe contiene le proprietà e i metodi per la creazione e la modifica di una pagina.

Utilizzo

Sarà utilizzata dallo sviluppatore per la creazione e la modifica di pagine per la visualizzazione di grafici, inoltre si occuperà di fornire ad ActivePage le pagine web e i dati $JSON_G$ da restituire ai $client_G$ per risolvere la prima richiesta di pagina.

Relazioni con altre classi

- **Norris::Back-end::Lib::DataLayer::ActivePage**

  Relazione entrante. Classe per l'invocazione delle API_G riguardanti le pagine.
- **Norris::Back-end::Lib::DataLayer::DataHandler**

  Relazione entrante. Classe per l'invocazione delle API_G riguardanti le pagine.
- **Norris::Back-end::Lib::DataLayer::Exception**

  Relazione uscente. Classe che si occupa di gestire gli errori.
- **Norris::Back-end::Lib::DataLayer::Graph**

  Relazione uscente. Classe per l'invocazione delle API_G riguardanti i grafici.

4.1.6.2.10 Norris::Back-end::Lib::DataLayer::Table

Descrizione

Questa classe rappresenta il grafico di tipo $Table_G$, è derivata dalla classe Graph.

Utilizzo

Sarà utilizzata dallo sviluppatore per la creazione e la modifica di grafici di tipo $Table_G$.

Classi ereditate

- Norris::Back-end::Lib::DataLayer::Graph.

4.1.7 Norris::Back-end::Lib::PresentationLayer

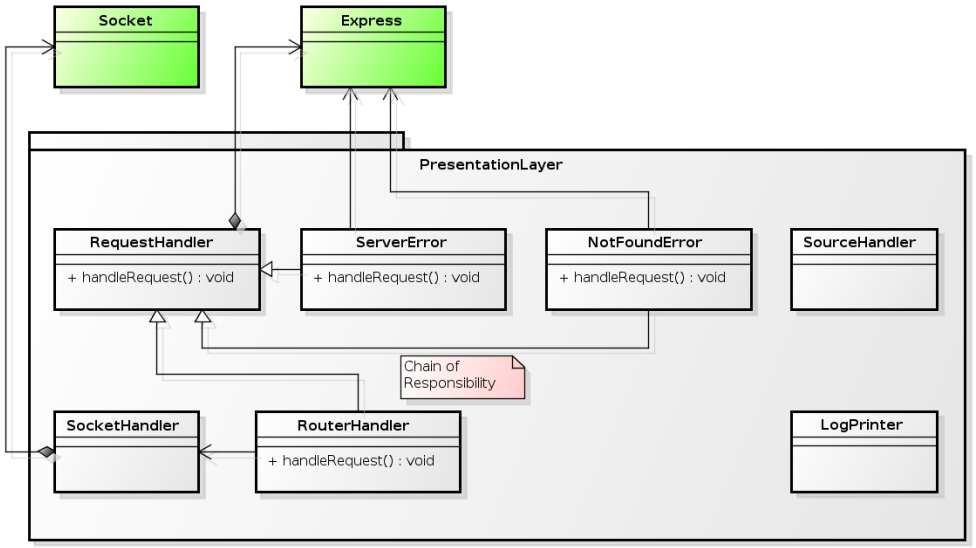

Figura 15: Componente PresentationLayer

4.1.7.1 Informazioni sul *package*$_G$

4.1.7.1.1 Descrizione

Questo *package*$_G$ contiene le classi che permettono al *framework*$_G$ la comunicazione con l'esterno rispettando la struttura della *Three Tier Architecture*$_G$.

4.1.7.2 Classi

4.1.7.2.1 Norris::Back-end::Lib::PresentationLayer::LogPrinter

Descrizione

Questa classe si occupa della gestione dell'output dei log di errore del *framework*$_G$ Norris.

Utilizzo

Sarà utilizzata da BusinessLayer::LogManager per la stampa del log degli errori.

Relazioni con altre classi

- **Norris::Back-end::Lib::BusinessLayer::LogManager**

  Relazione entrante. Classe per l'output dei log di errore.

4.1.7.2.2 Norris::Back-end::Lib::PresentationLayer::NotFoundError

Descrizione

Questa classe si occupa delle richieste di pagina che generano l'errore 404, è derivata dalla classe RequestHandler.

Utilizzo

Sarà utilizzata in caso di richieste di pagina non corrisposte.

Classi ereditate

– Norris::Back-end::Lib::PresentationLayer::RequestHandler.

4.1.7.2.3 Norris::Back-end::Lib::PresentationLayer::RequestHandler

Descrizione

Questa classe contiene i metodi atti alla gestione delle richieste da parte dei *client*$_G$, sta alla base della gerarchia che costituisce il *design pattern*$_G$ *Chain of Responsibility*$_G$.

Utilizzo

Sarà utilizzata per gestire e riconoscere le diverse richieste da parte dei *client*$_G$, in particolare attraverso la classe RouterHandler, da essa derivata. Gestirà gli eventuali errori attraverso le classi ServerError e NotFoundError, anch'esse parte della gerarchia.

Classi figlie

– Norris::Back-end::Lib::PresentationLayer::NotFoundError;
– Norris::Back-end::Lib::PresentationLayer::RouterHandler;
– Norris::Back-end::Lib::PresentationLayer::ServerError.

Relazioni con altre classi

– **Norris::Back-end::Lib::ServerApp**
  Relazione entrante. Classe per la gestione delle richieste di pagina.
– **Norris::Back-end::Lib::PresentationLayer::SocketHandler**
  Relazione uscente. Questa classe si occupa della gestione delle connessioni tramite *Socket.io*$_G$.

4.1.7.2.4 Norris::Back-end::Lib::PresentationLayer::RouterHandler

Descrizione

Questa classe contiene i metodi che gestiscono il *routing*$_G$ e le comunicazioni con *Socket.io*$_G$, è derivata da RequestHandler.

Utilizzo

Sarà utilizzata per la gestione delle richieste valide.

Classi ereditate

– Norris::Back-end::Lib::PresentationLayer::RequestHandler.

Relazioni con altre classi

– **Norris::Back-end::Lib::BusinessLayer::Router**
  Relazione uscente. Classe che si occupa del *routing*$_G$ delle chiamate.

4.1.7.2.5 Norris::Back-end::Lib::PresentationLayer::ServerError

Descrizione

Questa classe si occupa della gestione degli errori interni del $server_G$, è derivata da RequestHandler.

Utilizzo

Sarà utilizzata per la gestione degli errori interni del $server_G$.

Classi ereditate

– Norris::Back-end::Lib::PresentationLayer::RequestHandler.

4.1.7.2.6 Norris::Back-end::Lib::PresentationLayer::SocketHandler

Descrizione

Questa classe si occupa di gestire la comunicazione con $Socket.io_G$.

Utilizzo

Sarà utilizzata dalle classi RouterHandler e BusinessLayer::ClientUpdater per la comunicazione con i $client_G$.

Relazioni con altre classi

– **Norris::Back-end::Lib::BusinessLayer::ClientUpdater**
  Relazione entrante. Questa classe si occupa della gestione delle connessioni tramite $Socket.io_G$.
– **Norris::Back-end::Lib::PresentationLayer::RequestHandler**
  Relazione entrante. Questa classe si occupa della gestione delle connessioni tramite $Socket.io_G$.

4.1.7.2.7 Norris::Back-end::Lib::PresentationLayer::SourceHandler

Descrizione

Questa classe gestisce le operazioni che verranno effettuate dallo sviluppatore.

Utilizzo

Sarà utilizzata dallo sviluppatore per la creazione di pagine e grafici.

Relazioni con altre classi

– **Norris::Back-end::Lib::ServerApp**
  Relazione entrante. Classe per la gestione delle chiamate dello sviluppatore.
– **Norris::Back-end::Lib::BusinessLayer::Builder**
  Relazione uscente. Classe che effettua la costruzione di grafici e pagine.
– **Norris::Back-end::Lib::BusinessLayer::Updater**
  Relazione uscente. Classe che effettua l'aggiornamento di grafici e pagine.

4.1.8 Norris::Front-end

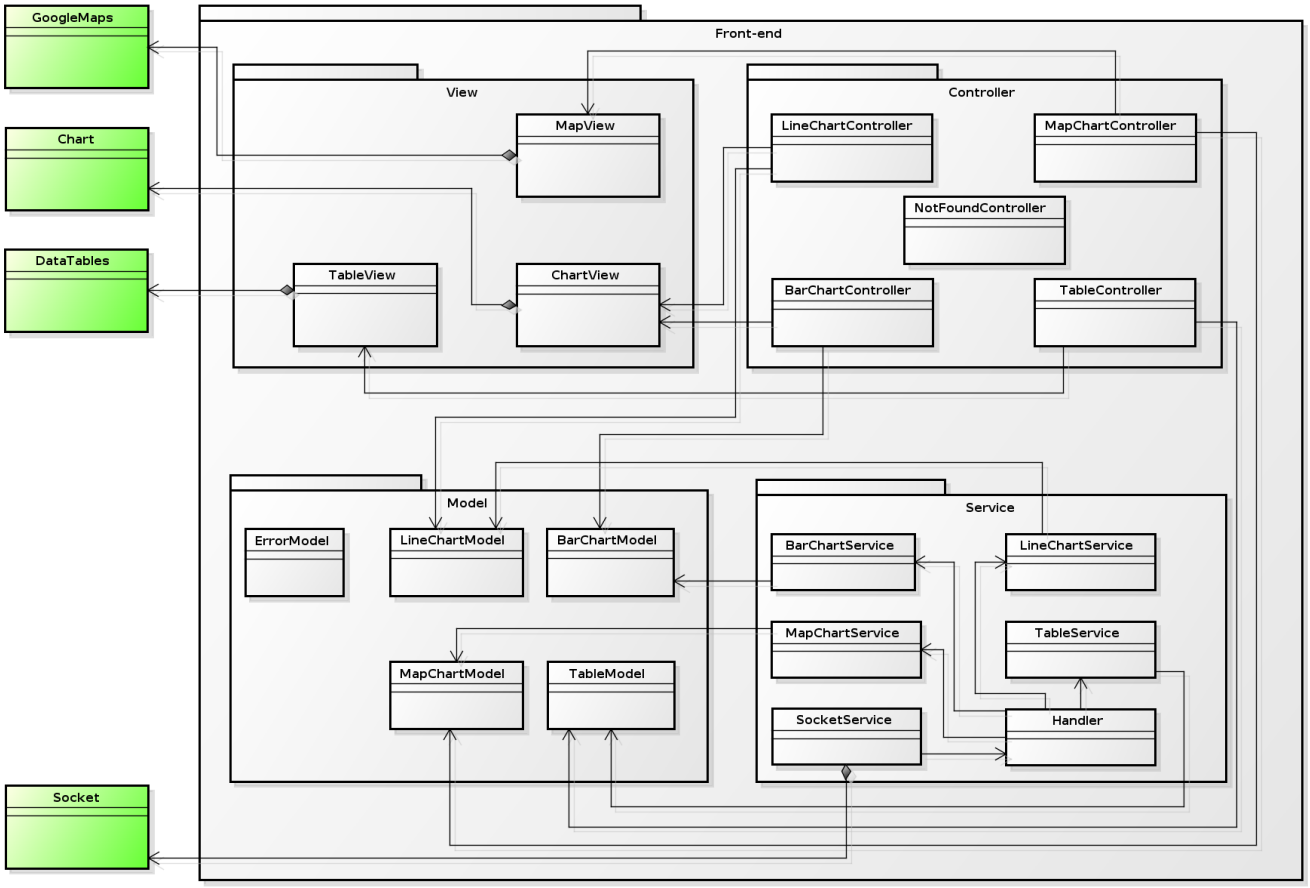

Figura 16: Diagramma delle classi Front-end

4.1.8.1 Informazioni sul *package*$_G$

4.1.8.1.1 Descrizione

Questo *package*$_G$ contiene tutte le classi che si occupano di gestire la visualizzazione dei grafici Norris dal browser.

4.1.8.1.2 *Package*$_G$ contenuti

- Norris::Front-end::Controller;
- Norris::Front-end::Model;
- Norris::Front-end::Service;
- Norris::Front-end::View.

4.1.9 Norris::Front-end::Controller

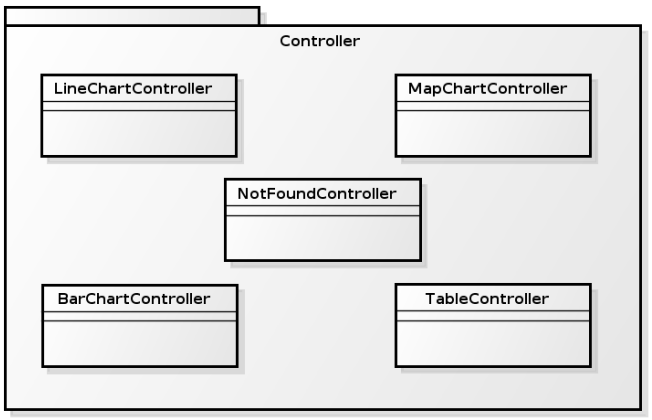

Figura 17: Componente Controller

4.1.9.1 Informazioni sul *package*$_G$

4.1.9.1.1 Descrizione

Questo *package*$_G$ contiene le classi per la creazione dei grafici e la gestione della pagina.

4.1.9.2 Classi

4.1.9.2.1 Norris::Front-end::Controller::BarChartController

Descrizione

Questa classe si occupa di rappresentare i grafici di tipo *Bar Chart*$_G$.

Utilizzo

Questa classe si occupa di disegnare il grafico tramite l'utilizzo della libreria *Chart.js*$_G$ e i dati gestiti tramite la classe Model::BarChartModel.

Relazioni con altre classi

- **Norris::Front-end::Model::BarChartModel**

  Relazione uscente. Classe che contiene i dati di grafici di tipo *Bar Chart*$_G$.
- **Norris::Front-end::View::ChartView**

  Relazione uscente. Classe che contiene la vista di grafici di tipo *Bar Chart*$_G$ e *Line Chart*$_G$.

4.1.9.2.2 Norris::Front-end::Controller::LineChartController

Descrizione

Questa classe si occupa di rappresentare i grafici di tipo *Line Chart*$_G$.

Utilizzo

Questa classe si occupa di disegnare il grafico tramite l'utilizzo della libreria *Chart.js*$_G$ e i dati gestiti tramite la classe Model::LineChartModel.

Relazioni con altre classi

- **Norris::Front-end::View::ChartView**

  Relazione uscente. Classe che contiene la vista di grafici di tipo *Bar Chart*$_G$ e *Line Chart*$_G$.
- **Norris::Front-end::Model::LineChartModel**

  Relazione uscente. Classe che contiene i dati di grafici di tipo *Line Chart*$_G$.

4.1.9.2.3 Norris::Front-end::Controller::MapChartController

Descrizione

Questa classe si occupa di rappresentare i grafici di tipo *Map Chart*$_G$.

Utilizzo

Questa classe si occupa di disegnare il grafico tramite l'utilizzo della libreria *Google Maps API*$_G$ e i dati gestiti tramite la classe Model::MapChartModel.

Relazioni con altre classi

- **Norris::Front-end::Model::MapChartModel**

  Relazione uscente. Classe che contiene i dati di grafici di tipo *Map Chart*$_G$.
- **Norris::Front-end::View::MapView**

  Relazione uscente. Classe che contiene la vista di grafici di tipo *Map Chart*$_G$.

4.1.9.2.4 Norris::Front-end::Controller::NotFoundController

Descrizione

Questa classe si occupa di gestire gli errori 404 lato *client*$_G$.

Utilizzo

Viene utilizzata da tutte le classi presenti all'interno del *package*$_G$ Front-end per gestire gli errori 404.

4.1.9.2.5 Norris::Front-end::Controller::TableController

Descrizione

Questa classe si occupa di rappresentare i grafici di tipo *Table*$_G$.

Utilizzo

Questa classe si occupa di disegnare il grafico tramite l'utilizzo della libreria *DataTables*$_G$ e i dati gestiti tramite la classe Model::TableModel.

Relazioni con altre classi

- **Norris::Front-end::Model::TableModel**

  Relazione uscente. Classe che contiene i dati di grafici di tipo $Table_G$.
- **Norris::Front-end::View::TableView**

  Relazione uscente. Classe che contiene la vista di grafici di tipo $Table_G$.

4.1.10 Norris::Front-end::Model

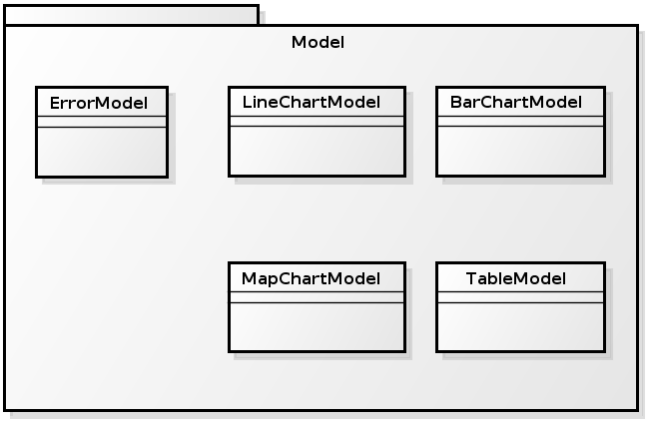

Figura 18: Componente Model

4.1.10.1 Informazioni sul *package*$_G$

4.1.10.1.1 Descrizione

Questo *package*$_G$ contiene le classi per mantenere e gestire i dati dei grafici presenti sulla pagina visualizzata dal *client*$_G$.

4.1.10.2 Classi

4.1.10.2.1 Norris::Front-end::Model::BarChartModel

Descrizione

Questa classe contiene i dati dei grafici di tipo *Bar Chart*$_G$.

Utilizzo

Sarà utilizzata dalle classi Service::BarChartService e Controller::BarChartController.

Relazioni con altre classi

- **Norris::Front-end::Controller::BarChartController**

  Relazione entrante. Classe che contiene i dati di grafici di tipo *Bar Chart*$_G$.
- **Norris::Front-end::Service::BarChartService**

  Relazione entrante. Classe che contiene i dati di grafici di tipo *Bar Chart*$_G$.

4.1.10.2.2 Norris::Front-end::Model::ErrorModel

Descrizione

Questa classe rappresenta la gestione degli errori che possono avvenire nel *client*$_G$.

Utilizzo

Viene utilizzata da tutte le classi presenti all'interno del *package*$_G$ Front-end per rappresentare e gestire gli errori.

4.1.10.2.3 Norris::Front-end::Model::LineChartModel

Descrizione

Questa classe contiene i dati dei grafici di tipo *Line Chart*$_G$.

Utilizzo

Sarà utilizzata dalle classi Service::LineChartService e Controller::LineChartController.

Relazioni con altre classi

- **Norris::Front-end::Controller::LineChartController**

  Relazione entrante. Classe che contiene i dati di grafici di tipo *Line Chart*$_G$.
- **Norris::Front-end::Service::LineChartService**

  Relazione entrante. Classe che contiene i dati di grafici di tipo *Line Chart*$_G$.

4.1.10.2.4 Norris::Front-end::Model::MapChartModel

Descrizione

Questa classe contiene i dati dei grafici di tipo *Map Chart*$_G$.

Utilizzo

Sarà utilizzata dalle classi Service::MapChartService e Controller::MapChartController.

Relazioni con altre classi

- **Norris::Front-end::Controller::MapChartController**

  Relazione entrante. Classe che contiene i dati di grafici di tipo *Map Chart*$_G$.
- **Norris::Front-end::Service::MapChartService**

  Relazione entrante. Classe che contiene i dati di grafici di tipo *Map Chart*$_G$.

4.1.10.2.5 Norris::Front-end::Model::TableModel

Descrizione

Questa classe contiene i dati dei grafici di tipo *Table*$_G$.

Utilizzo

Sarà utilizzata dalle classi Service::TableService e Controller::TableController.

Relazioni con altre classi

- **Norris::Front-end::Controller::TableController**
  Relazione entrante. Classe che contiene i dati di grafici di tipo *Table*$_G$.
- **Norris::Front-end::Service::TableService**
  Relazione entrante. Classe che contiene i dati di grafici di tipo *Table*$_G$.

4.1.11 Norris::Front-end::Service

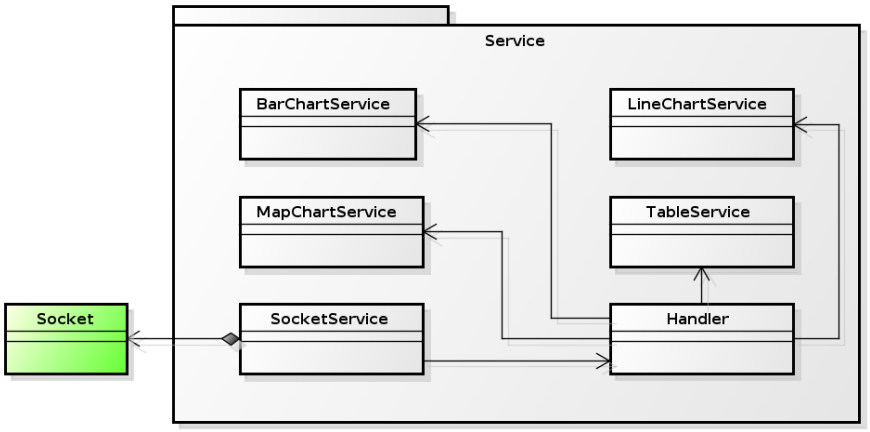

Figura 19: Componente Service

4.1.11.1 Informazioni sul *package*$_G$

4.1.11.1.1 Descrizione

Questo *package*$_G$ contiene le classi che permettono l'aggiornamento dei dati in seguito a notifiche da parte del *server*$_G$.

4.1.11.2 Classi

4.1.11.2.1 Norris::Front-end::Service::BarChartService

Descrizione

Questa classe si occupa di gestire i dati dei grafici di tipo *Bar Chart*$_G$.

Utilizzo

Questa classe si occupa di inserire ed aggiornare i dati gestiti tramite la classe Model::BarChartModel.

Relazioni con altre classi

- **Norris::Front-end::Service::Handler**

  Relazione entrante. Classe che si occupa della gestione di dei dati presenti in BarChartModel.

- **Norris::Front-end::Model::BarChartModel**

  Relazione uscente. Classe che contiene i dati di grafici di tipo *Bar Chart*$_G$.

4.1.11.2.2 Norris::Front-end::Service::Handler

Descrizione

Questa classi si occupa di suddividere i dati in arrivo dal *server*$_G$ tra gli specifici service.

Utilizzo

Sarà utilizzata per fornire ai service specifici di ogni tipo di grafico i dati corrispondenti forniti dalla classe SocketService.

Relazioni con altre classi

- **Norris::Front-end::Service::SocketService**

  Relazione entrante. Classe che gestisce i dati in arrivo sul textitclient$_G$.

- **Norris::Front-end::Service::BarChartService**

  Relazione uscente. Classe che si occupa della gestione di dei dati presenti in BarChartModel.

- **Norris::Front-end::Service::LineChartService**

  Relazione uscente. Classe che si occupa della gestione di dei dati presenti in LineChartModel.

- **Norris::Front-end::Service::MapChartService**

  Relazione uscente. Classe che si occupa della gestione di dei dati presenti in MapChartModel.

- **Norris::Front-end::Service::TableService**

  Relazione uscente. Classe che si occupa della gestione di dei dati presenti in TableModel.

4.1.11.2.3 Norris::Front-end::Service::LineChartService

Descrizione

Questa classe si occupa di gestire i dati dei grafici di tipo *Line Chart*$_G$.

Utilizzo

Questa classe si occupa di inserire ed aggiornare i dati gestiti tramite la classe Model::LineChartModel.

Relazioni con altre classi

- **Norris::Front-end::Service::Handler**

  Relazione entrante. Classe che si occupa della gestione di dei dati presenti in LineChartModel.

- **Norris::Front-end::Model::LineChartModel**

  Relazione uscente. Classe che contiene i dati di grafici di tipo *Line Chart*$_G$.

4.1.11.2.4 Norris::Front-end::Service::MapChartService

Descrizione

Questa classe si occupa di gestire i dati dei grafici di tipo *Map Chart*$_G$.

Utilizzo

Questa classe si occupa di inserire ed aggiornare i dati gestiti tramite la classe Model::MapChartModel.

Relazioni con altre classi

- **Norris::Front-end::Service::Handler**

  Relazione entrante. Classe che si occupa della gestione di dei dati presenti in MapChartModel.

- **Norris::Front-end::Model::MapChartModel**

  Relazione uscente. Classe che contiene i dati di grafici di tipo *Map Chart*$_G$.

4.1.11.2.5 Norris::Front-end::Service::SocketService

Descrizione

Questa classe si occupa di gestire le comunicazioni del *client*$_G$ con il *server*$_G$ tramite *Socket.io*$_G$.

Utilizzo

Sarà utilizzata per permettere al *client*$_G$ di ricevere i dati dal *server*$_G$ e fornirli alla classe Handler.

Relazioni con altre classi

- **Norris::Front-end::Service::Handler**

  Relazione uscente. Classe che gestisce i dati in arrivo sul textitclient$_G$.

4.1.11.2.6 Norris::Front-end::Service::TableService

Descrizione

Questa classe si occupa di gestire i dati dei grafici di tipo *Table*$_G$.

Utilizzo

Questa classe si occupa di inserire ed aggiornare i dati gestiti tramite la classe Model::TableModel.

Relazioni con altre classi

- **Norris::Front-end::Service::Handler**

  Relazione entrante. Classe che si occupa della gestione di dei dati presenti in TableModel.

- **Norris::Front-end::Model::TableModel**

  Relazione uscente. Classe che contiene i dati di grafici di tipo *Table*$_G$.

4.1.12 Norris::Front-end::View

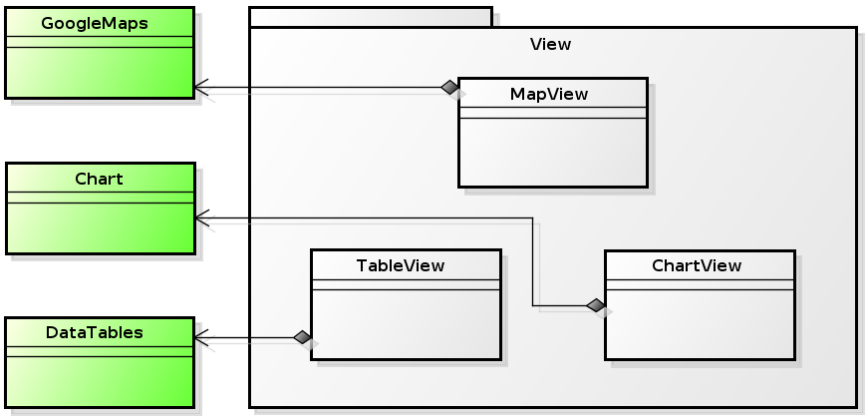

Figura 20: Componente View

4.1.12.1 Informazioni sul *package*$_G$

4.1.12.1.1 Descrizione

Questo *package*$_G$ contiene i *template*$_G$ per la visualizzazione dei grafici nelle pagine.

4.1.12.2 Classi

4.1.12.2.1 Norris::Front-end::View::ChartView

Descrizione

Questa classe contiene la rappresentazione dei grafici di tipo *Line Chart*$_G$ e di tipo *Bar Chart*$_G$.

Utilizzo

Sarà utilizzata dalle classi Controller::BarChartController e Controller::LineChartController.

Relazioni con altre classi

- **Norris::Front-end::Controller::BarChartController**

  Relazione entrante. Classe che contiene la vista di grafici di tipo *Bar Chart*$_G$ e *Line Chart*$_G$.

– **Norris::Front-end::Controller::LineChartController**

Relazione entrante. Classe che contiene la vista di grafici di tipo *Bar Chart*$_G$ e *Line Chart*$_G$.

4.1.12.2.2 Norris::Front-end::View::MapView

Descrizione

Questa classe contiene la rappresentazione dei grafici di tipo *Map Chart*$_G$.

Utilizzo

Sarà utilizzata dalla classe Controller::MapChartController.

Relazioni con altre classi

– **Norris::Front-end::Controller::MapChartController**

Relazione entrante. Classe che contiene la vista di grafici di tipo *Map Chart*$_G$.

4.1.12.2.3 Norris::Front-end::View::TableView

Descrizione

Questa classe contiene la rappresentazione dei grafici di tipo *Table*$_G$.

Utilizzo

Sarà utilizzata dalla classe Controller::TableController.

Relazioni con altre classi

– **Norris::Front-end::Controller::TableController**

Relazione entrante. Classe che contiene la vista di grafici di tipo *Table*$_G$.

4.2 Componente applicazione *Android*$_G$

4.2.1 AndroidApp

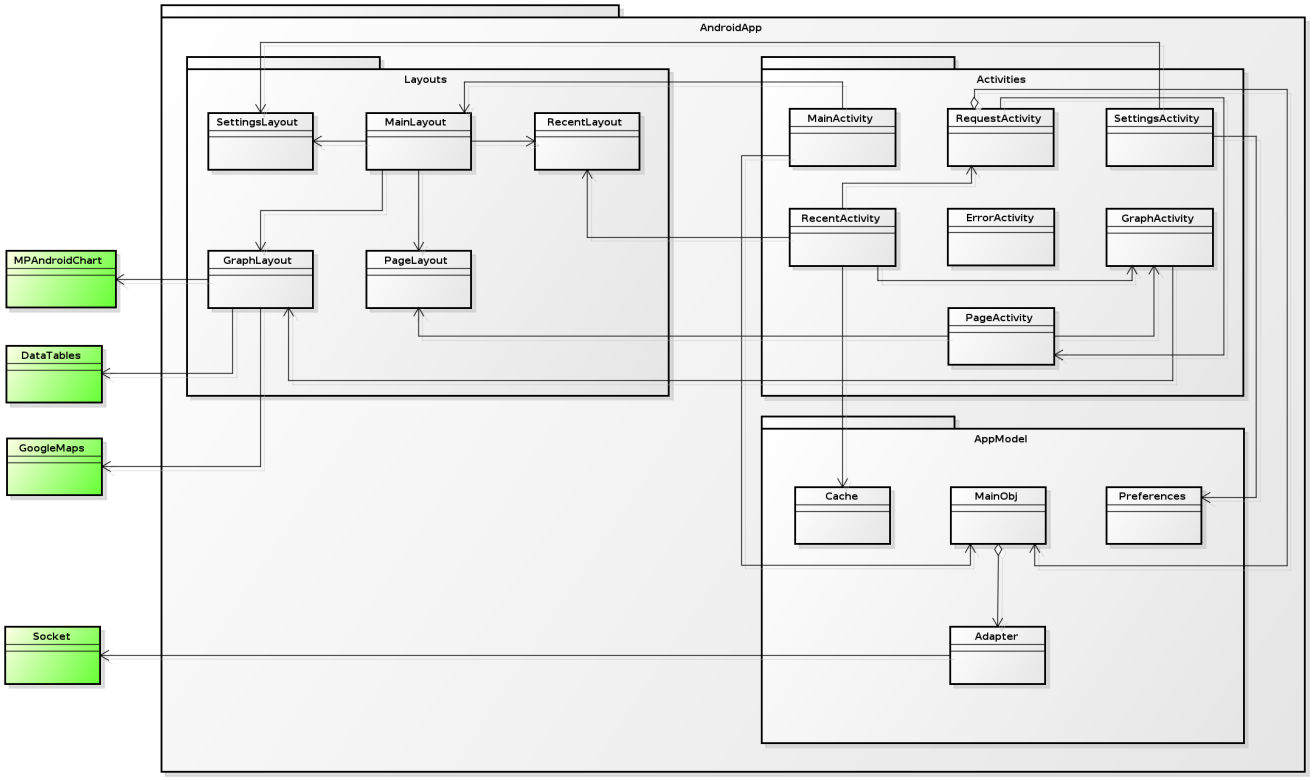

Figura 21: Diagramma delle classi AndroidApp

4.2.1.1 Informazioni sul *package*$_G$

4.2.1.1.1 Descrizione

Questo *package*$_G$ contiene tutte le classi che si occupano di gestire la visualizzazione dei grafici Norris nell'applicazione per *smartphone*$_G$ *Android*$_G$.

4.2.1.1.2 *Package*$_G$ contenuti

- AndroidApp::Activities;
- AndroidApp::AppModel;
- AndroidApp::Layouts.

4.2.2 AndroidApp::Activities

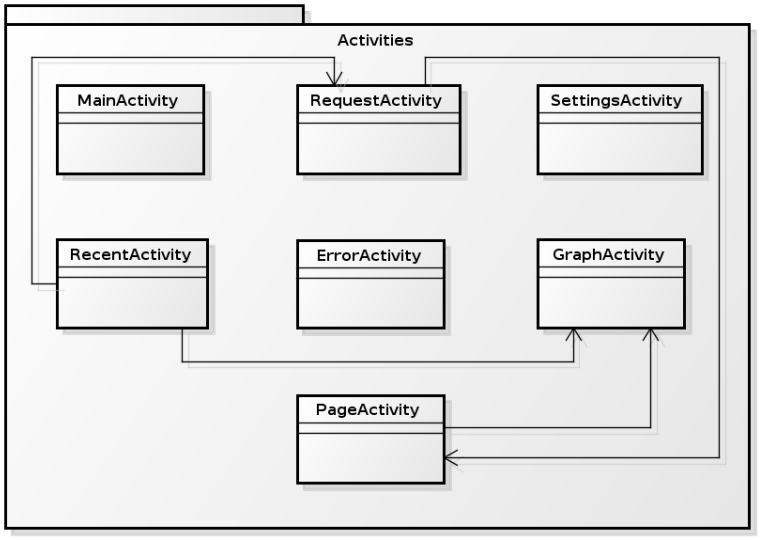

Figura 22: Componente Activities

4.2.2.1 Informazioni sul *package*$_G$

4.2.2.1.1 Descrizione

Questo *package*$_G$ contiene le classi per la gestione dell'applicazione.

4.2.2.2 Classi

4.2.2.2.1 AndroidApp::Activities::ErrorActivity

Descrizione

Questa classe contiene le activities per la gestione degli errori nell'applicazione.

Utilizzo

Questa classe si occupa degli errori nell'applicazione.

4.2.2.2.2 AndroidApp::Activities::GraphActivity

Descrizione

Questa classe contiene le activities per la gestione dei grafici visualizzati nell'applicazione.

Utilizzo

Questa classe si occupa della gestione dei grafici nell'applicazione.

Relazioni con altre classi

- **AndroidApp::Activities::PageActivity**

Relazione entrante. Classe che si occupa della gestione dei grafici nell'applicazione.

- **AndroidApp::Activities::RecentActivity**

  Relazione entrante. Classe che si occupa della gestione dei grafici nell'applicazione.

- **AndroidApp::Layouts::GraphLayout**

  Relazione uscente. Classe che contiene i layout dei grafici.

4.2.2.2.3 AndroidApp::Activities::MainActivity

Descrizione

Questa classe contiene le activities per la gestione dell'applicazione.

Utilizzo

Questa classe si occupa dell'avvio dell'applicazione.

Relazioni con altre classi

- **AndroidApp::Layouts::MainLayout**

  Relazione uscente. Classe che contiene il layout dell'applicazione.

- **AndroidApp::AppModel::MainObj**

  Relazione uscente. Classe che contiene i dati visualizzati dall'applicazione .

4.2.2.2.4 AndroidApp::Activities::PageActivity

Descrizione

Questa classe contiene le activities per la gestione delle pagine contenenti i grafici nell'applicazione.

Utilizzo

Questa classe si occupa della gestione delle pagine nell'applicazione.

Relazioni con altre classi

- **AndroidApp::Activities::RequestActivity**

  Relazione entrante. Classe che si occupa della gestione delle pagine contenenti i grafici nell'applicazione.

- **AndroidApp::Activities::GraphActivity**

  Relazione uscente. Classe che si occupa della gestione dei grafici nell'applicazione.

- **AndroidApp::Layouts::PageLayout**

  Relazione uscente. Classe che contiene il layout per la pagina contenente la lista dei grafici.

4.2.2.2.5 AndroidApp::Activities::RecentActivity

Descrizione

Questa classe contiene le activities per la gestione dei contenuti visualizzati recentemente tramite l'applicazione.

Utilizzo

Questa classe si occupa delle risorse recenti visualizzate nell'applicazione.

Relazioni con altre classi

- **AndroidApp::AppModel::Cache**

  Relazione uscente. Classe per la memorizzazione dei dati sui grafici recenti.

- **AndroidApp::Activities::GraphActivity**

  Relazione uscente. Classe che si occupa della gestione dei grafici nell'applicazione.

- **AndroidApp::Layouts::RecentLayout**

  Relazione uscente. Classe che contiene il layout con i grafici visti di recente.

- **AndroidApp::Activities::RequestActivity**

  Relazione uscente. Questa classe gestisce le richieste dei dati nell'applicazione.

4.2.2.2.6 AndroidApp::Activities::RequestActivity

Descrizione

Questa classe contiene le activities per la gestione delle richieste dei contenuti ad AppModel::MainObj.

Utilizzo

Questa classe si occupa di gestire le richieste dei dati nell'applicazione.

Relazioni con altre classi

- **AndroidApp::Activities::RecentActivity**

  Relazione entrante. Questa classe gestisce le richieste dei dati nell'applicazione.

- **AndroidApp::AppModel::MainObj**

  Relazione uscente. Classe che contiene i dati visualizzati dall'applicazione .

- **AndroidApp::Activities::PageActivity**

  Relazione uscente. Classe che si occupa della gestione delle pagine contenenti i grafici nell'applicazione.

4.2.2.2.7 AndroidApp::Activities::SettingsActivity

Descrizione

Questa classe contiene le activities per la gestione delle impostazioni dell'applicazione.

Utilizzo

Questa classe si occupa delle impostazioni dell'applicazione.

Relazioni con altre classi

- **AndroidApp::AppModel::Preferences**
  Relazione uscente. Classe che contiene le impostazioni scelte dall'utente.
- **AndroidApp::Layouts::SettingsLayout**
  Relazione uscente. Classe che contiene il layout del menu impostazioni.

4.2.3 AndroidApp::AppModel

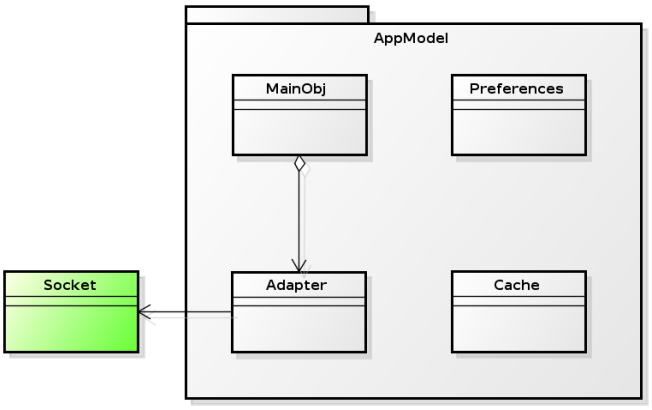

Figura 23: Componente AppModel

4.2.3.1 Informazioni sul *package*$_G$

4.2.3.1.1 Descrizione

Questo *package*$_G$ contienem le classi per mantenere e gestire i dati necessari all'applicazione.

4.2.3.2 Classi

4.2.3.2.1 AndroidApp::AppModel::Adapter

Descrizione

Questa classe si occupa del interazione dell'applicazione con il *server*$_G$ tramite *Socket.io*$_G$.

Utilizzo

Sarà utilizzata da MainObject per memorizzare i dati recenti.

Relazioni con altre classi

- **AndroidApp::AppModel::MainObj**
  Relazione entrante. Classe per l'interfacciamento a *Socket.io*$_G$.

4.2.3.2.2 AndroidApp::AppModel::Cache

Descrizione

Questa classe contiene i dati già richiesti e visualizzati dall'utente.

Utilizzo

Sarà utilizzata da Activities::RecentActivity per memorizzare i dati recenti.

Relazioni con altre classi

- **AndroidApp::Activities::RecentActivity**
  Relazione entrante. Classe per la memorizzazione dei dati sui grafici recenti.

4.2.3.2.3 AndroidApp::AppModel::MainObj

Descrizione

Questa classe si occupa della trasmissione dei dati dell'applicazione.

Utilizzo

Questa classe si occupa di gestire i dati dell'applicazione.

Relazioni con altre classi

- **AndroidApp::Activities::MainActivity**
  Relazione entrante. Classe che contiene i dati visualizzati dall'applicazione .
- **AndroidApp::Activities::RequestActivity**
  Relazione entrante. Classe che contiene i dati visualizzati dall'applicazione .
- **AndroidApp::AppModel::Adapter**
  Relazione uscente. Classe per l'interfacciamento a *Socket.io*$_G$.

4.2.3.2.4 AndroidApp::AppModel::Preferences

Descrizione

Questa classe contiene le impostazioni dell'applicazione selezionate dall'utente.

Utilizzo

Sarà utilizzata da Activities::SettingsActivity per memorizzare le preferenze.

Relazioni con altre classi

- **AndroidApp::Activities::SettingsActivity**
  Relazione entrante. Classe che contiene le impostazioni scelte dall'utente.

4.2.4 AndroidApp::Layouts

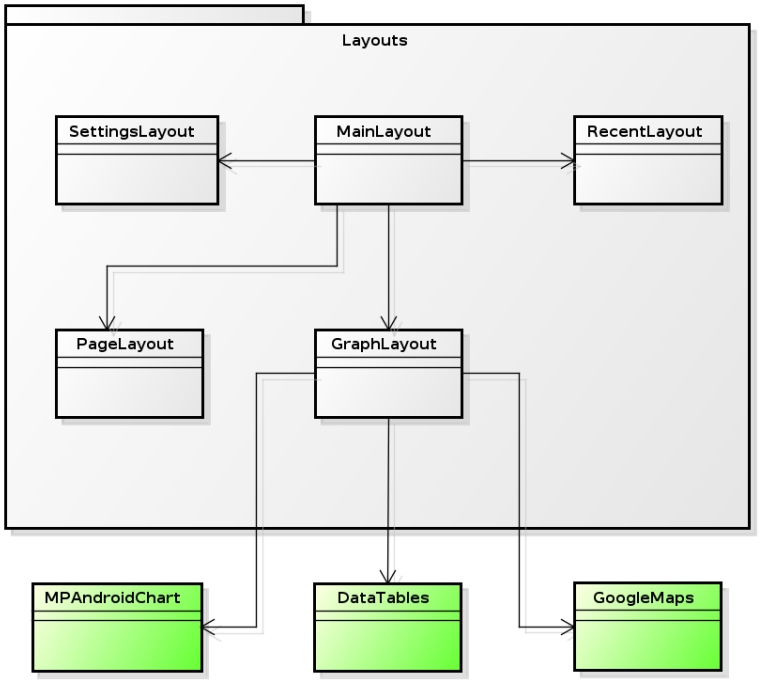

Figura 24: Componente Layouts

4.2.4.1 Informazioni sul *package*$_G$

4.2.4.1.1 Descrizione

Questo *package*$_G$ contiene le classi con le view usate dall'applicazione.

4.2.4.2 Classi

4.2.4.2.1 AndroidApp::Layouts::GraphLayout

Descrizione

Questa classe contiene il layout di visualizzazione del grafico selezionato dall'utente.

Utilizzo

Sarà utilizzata dalla classe Activities::GraphActivity per la gestione del layout di visualizzazione del grafico.

Relazioni con altre classi

- **AndroidApp::Activities::GraphActivity**

  Relazione entrante. Classe che contiene i layout dei grafici.
- **AndroidApp::Layouts::MainLayout**

  Relazione entrante. Classe che contiene i layout dei grafici.

4.2.4.2.2 AndroidApp::Layouts::MainLayout

Descrizione

Questa classe contiene il layout di base dell'applicazione.

Utilizzo

Sarà utilizzata dalla classe Activities::MainActivity per la gestione del layout.

Relazioni con altre classi

- **AndroidApp::Activities::MainActivity**

  Relazione entrante. Classe che contiene il layout dell'applicazione.
- **AndroidApp::Layouts::GraphLayout**

  Relazione uscente. Classe che contiene i layout dei grafici.
- **AndroidApp::Layouts::PageLayout**

  Relazione uscente. Classe che contiene il layout per la pagina contenente la lista dei grafici.
- **AndroidApp::Layouts::RecentLayout**

  Relazione uscente. Classe che contiene il layout con i grafici visti di recente.
- **AndroidApp::Layouts::SettingsLayout**

  Relazione uscente. Classe che contiene il layout del menu impostazioni.

4.2.4.2.3 AndroidApp::Layouts::PageLayout

Descrizione

Questa classe contiene il layout del menu che visualizzerà i dati della pagina scelta dall'utente.

Utilizzo

Sarà utilizzata dalla classe Activities::PageActivity per la gestione del layout di visualizzazione della pagina.

Relazioni con altre classi

- **AndroidApp::Layouts::MainLayout**

  Relazione entrante. Classe che contiene il layout per la pagina contenente la lista dei grafici.
- **AndroidApp::Activities::PageActivity**

  Relazione entrante. Classe che contiene il layout per la pagina contenente la lista dei grafici.

4.2.4.2.4 AndroidApp::Layouts::RecentLayout

Descrizione

Questa classe include il layout del menu contente i grafici recenti visualizzati tramite l'applicazione.

Utilizzo

Sarà utilizzata dalla classe Activities::RecentActivity per la gestione del layout del menu recenti.

Relazioni con altre classi

- **AndroidApp::Layouts::MainLayout**

  Relazione entrante. Classe che contiene il layout con i grafici visti di recente.

- **AndroidApp::Activities::RecentActivity**

  Relazione entrante. Classe che contiene il layout con i grafici visti di recente.

4.2.4.2.5 AndroidApp::Layouts::SettingsLayout

Descrizione

Questa classe include il layout del menu contenente le impostazioni dell'applicazione.

Utilizzo

Sarà utilizzata dalla classe Activities::SettingsActivity per la gestione del layout del menu impostazioni.

Relazioni con altre classi

- **AndroidApp::Layouts::MainLayout**

  Relazione entrante. Classe che contiene il layout del menu impostazioni.

- **AndroidApp::Activities::SettingsActivity**

  Relazione entrante. Classe che contiene il layout del menu impostazioni.

5 Comunicazione *client*$_G$-*server*$_G$

Vengono di seguito presentati i diagrammi di sequenza prodotti durante la fase di progettazione, i quali descrivono la gestione delle comunicazioni tra $server_G$ e $client_G$. Per illustrare le comunicazioni sono stati creati tre diagrammi: uno che rappresenta la prima richiesta di pagina, uno per la gestione della richiesta all'interno del $server_G$ ed infine un diagramma che rappresenta l'inoltro degli aggiornamenti dei dati.

5.1 Prima richiesta di pagina

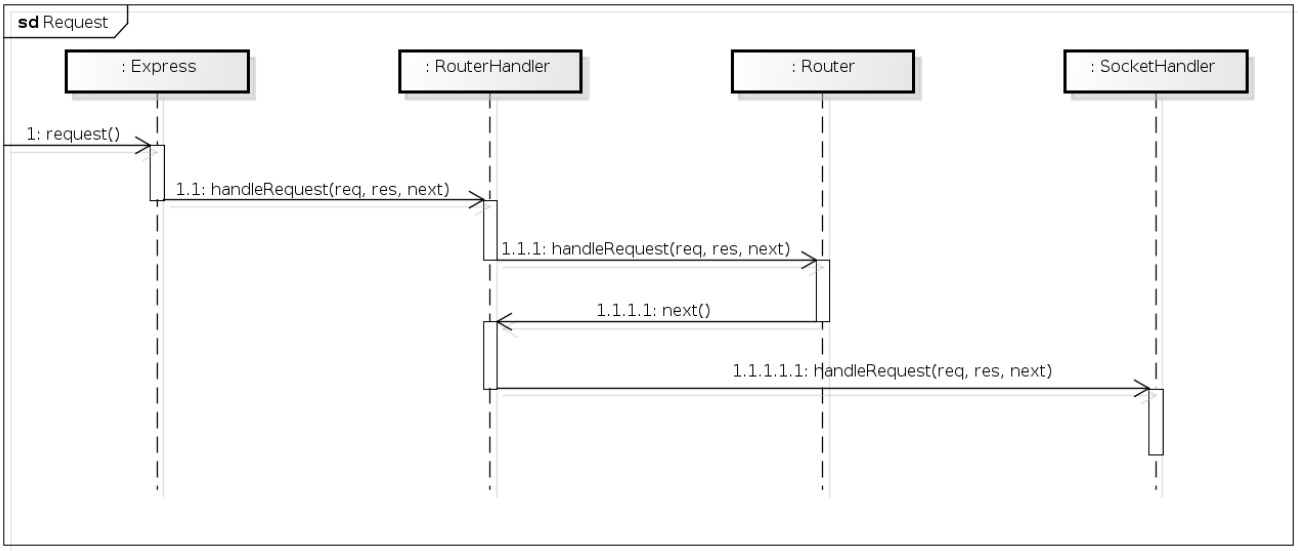

Figura 25: Diagramma di sequenza - Prima richiesta di pagina

La prima richiesta di pagina del $client_G$, sia esso un browser o l'applicazione $Android_G$, avviene tramite una chiamata $HTTP_G$ al $server_G$, il quale tramite l'utilizzo di $Express_G$ riceve la richiesta.
Una volta ricevuta, quest'ultima sarà analizzata e gestita tramite un'applicazione del *design pattern*$_G$ *chain of responsibility*$_G$ utilizzando le funzionalità di $Express_G$. Scendendo più nel dettaglio, dopo la ricezione la richiesta sarà analizzata da RouterHandler che verificherà se corrisponde all'indirizzo di una pagina valida. In caso affermativo sarà passata a Router che la elaborerà caricando i dati e restituendo la pagina al $client_G$ attraverso la classe SocketHandler, la quale creerà la connessione $WebSocket_G$ tramite $Socket.io_G$.

5.2 Prima richiesta di pagina non valida

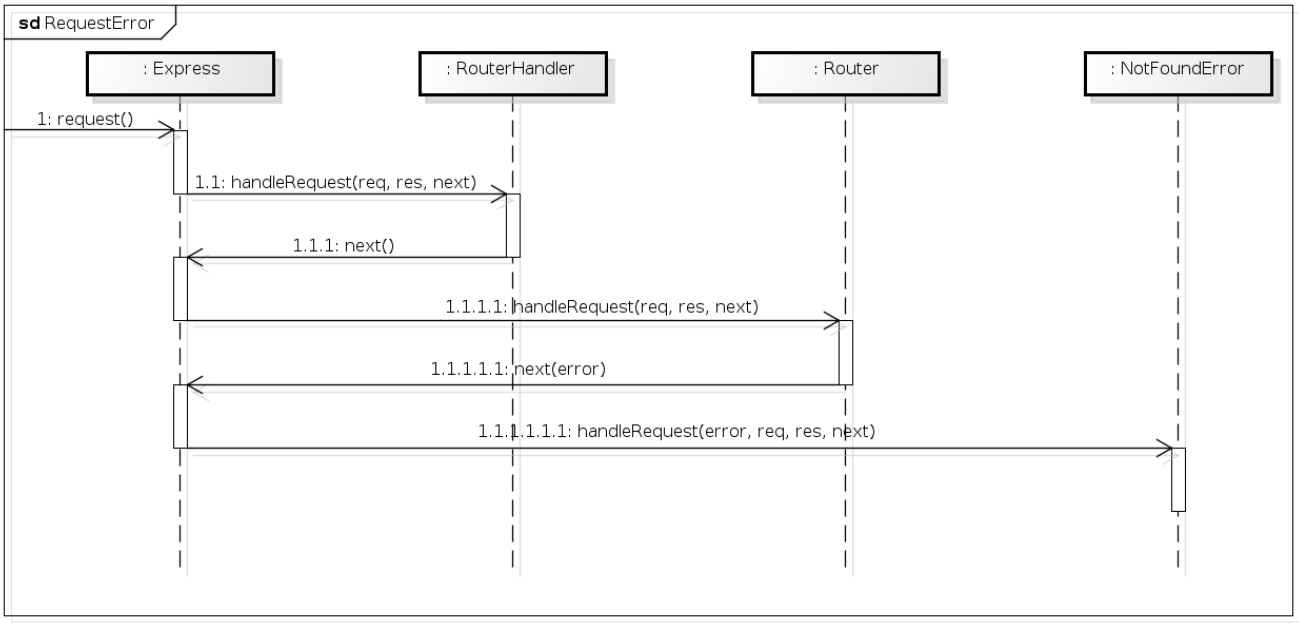

Figura 26: Diagramma di sequenza - Prima richiesta di pagina non valida

Nel caso venga effettuata una richiesta di pagina da un $client_G$ che corrisponda ad una pagina non presente nel $server_G$, la ricezione della richiesta sarà analizzata da RouterHandler, che la inoltrerà alla classe NotFoundError. Quest'ultima si occuperà di segnalare al richiedente della pagina la mancanza di una risorsa identificata da tale URL_G.

5.3 *Routing*$_G$

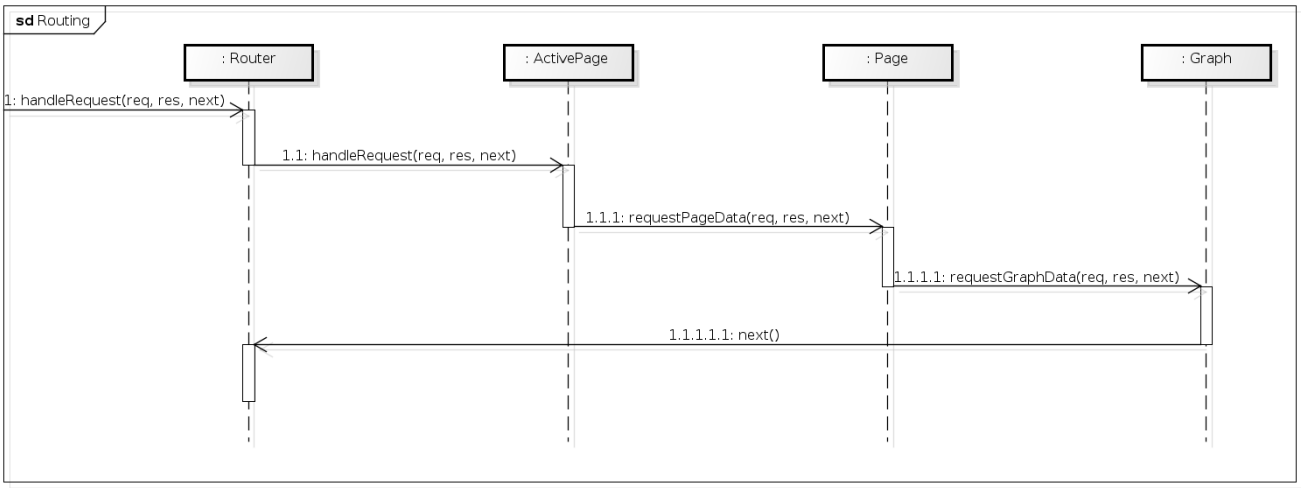

Figura 27: Diagramma di sequenza - *Routing*$_G$ di una richiesta

La gestione delle richieste di pagina valide partirà dalla classe Router che tramite il metodo HandleRequest trasmetterà la richiesta alla classe ActivePage. A questo punto

ActivePage selezionerà a quale pagina corrisponde l'URL_G richiesto, caricando così dalla classe Page la pagina che verrà restituita al $client_G$. Page a sua volta caricherà i grafici necessari dalla classe Graph.

5.4 Inoltro notifiche $push_G$

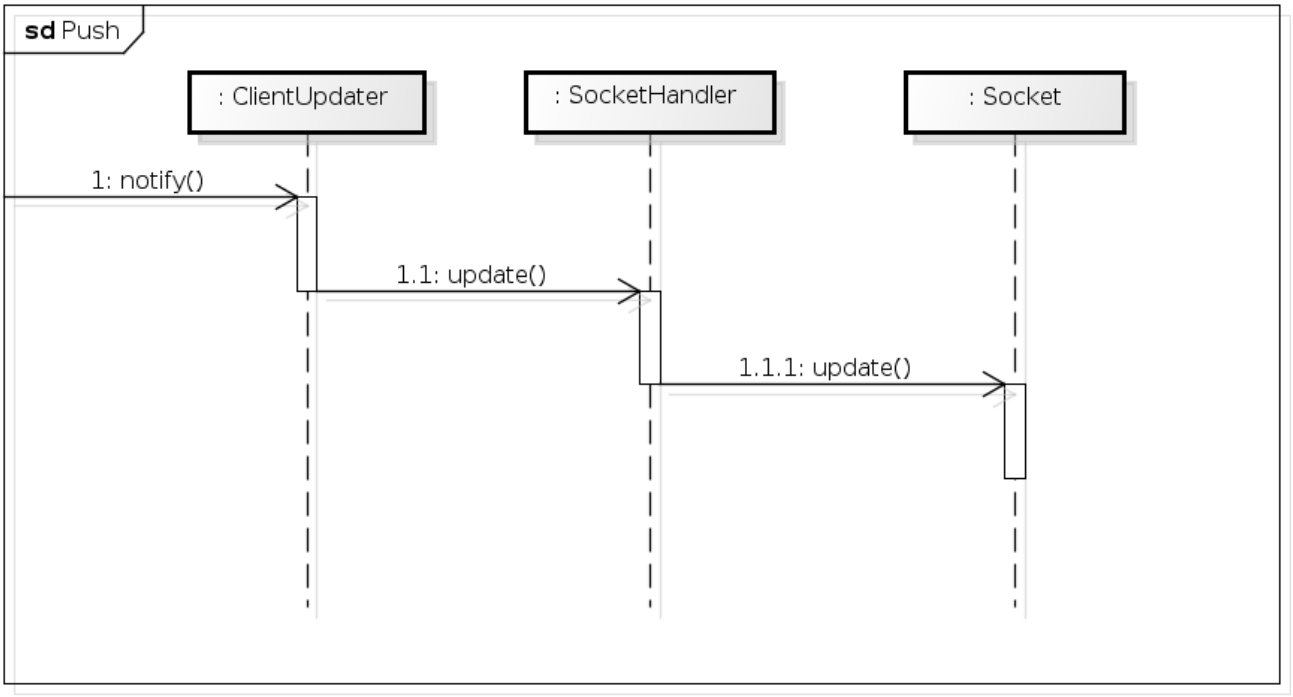

Figura 28: Diagramma di sequenza - Inoltro degli aggiornamenti

Qualora lo sviluppatore invocasse la funzione per l'aggiornamento dei dati di un grafico, la classe Updater tramite il metodo *notify()* si occuperà di segnalare a ClientUpdater l'avvenuta modifica. ClientUpdater invocherà a sua volta il metodo *update()* che segnalerà il cambiamento alla classe SocketHandler, la quale integrando le funzionalità di $Socket.io_G$ notificherà in modo automatico il cambiamento a tutti i $client_G$ che stanno visualizzando il grafico in questione.

6 Diagrammi di attività

Vengono di seguito illustrati i diagrammi di attività prodotti durante la fase di progettazione, i quali descrivono le iterazioni dell'utente finale e dello sviluppatore con il sistema Norris.
È stato ritenuto opportuno suddividere i diagrammi in tre categorie principali, in modo analogo a quanto fatto nella descrizione dei casi d'uso del documento *AnalisiRequisiti_ver3.0.0*:

- **Funzionalità Sviluppatore**: verranno illustrate le differenti possibilità di interazione fra lo sviluppatore e il *framework*$_G$ Norris;
- **Funzionalità Utente**: verranno illustrate le interazioni possibili operate da parte delle utente tramite il *client*$_G$ web;
- **Applicazione *Android*$_G$**: verranno mostrate le varie operazioni possibili da parte degli utenti tramite l'applicativo *Android*$_G$.

Inizialmente per ogni categoria verrà fornito uno schema ad alto livello, per poi scendere nel dettaglio tramite sotto-diagrammi più specifici. Le attività che verranno esplose sono marcate dal simbolo apposito.
Al fine di rendere il diagramma più facilmente leggibile, è stata considerata implicita la possibilità per l'utente di chiudere in qualsiasi momento la connessione al *server*$_G$, per esempio chiudendo la finestra del browser o terminando l'applicazione *Android*$_G$.

6.1 Funzionalità Sviluppatore

Vengono mostrate e descritte di seguito le operazioni che possono essere eseguite da uno sviluppatore mediante l'uso del *framework*$_G$ Norris:

6.1.1 Attività principali

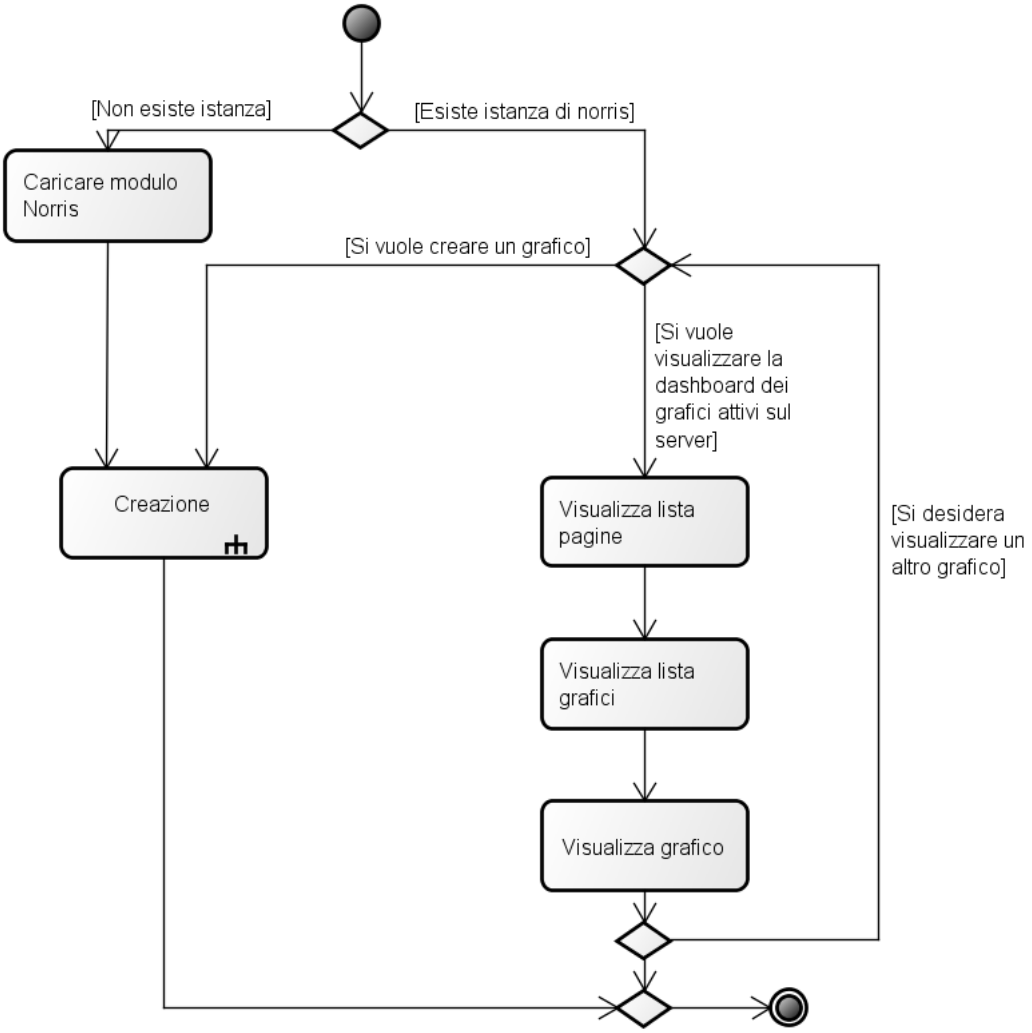

Figura 29: Diagramma di attività - Attività principali sviluppatore

Norris fornisce delle funzionalità a uno sviluppatore che volesse servirsi di grafici aggiornabili in tempo reale. Per fare uso delle funzionalità del *framework*$_G$, è necessario inizialmente importare il modulo nel proprio script. Successivamente sarà possibile accedere alle funzionalità di creazione di pagine e grafici, descritte in modo più approfondito nella sezione *6.1.2 - Attività di creazione*.
È disponibile anche una *dashboard*$_G$ web dei grafici associati a una particolare istanza del *framework*$_G$, che può essere consultata tramite browser, nella quale si possono effettuare le seguenti operazioni:

- Visualizzare la lista delle pagine attive;
- Visualizzare la lista dei grafici attivi;
- Visualizzare un singolo grafico.

L'operazione può essere ripetuta a seconda delle necessità dello sviluppatore.

6.1.2 Attività di creazione

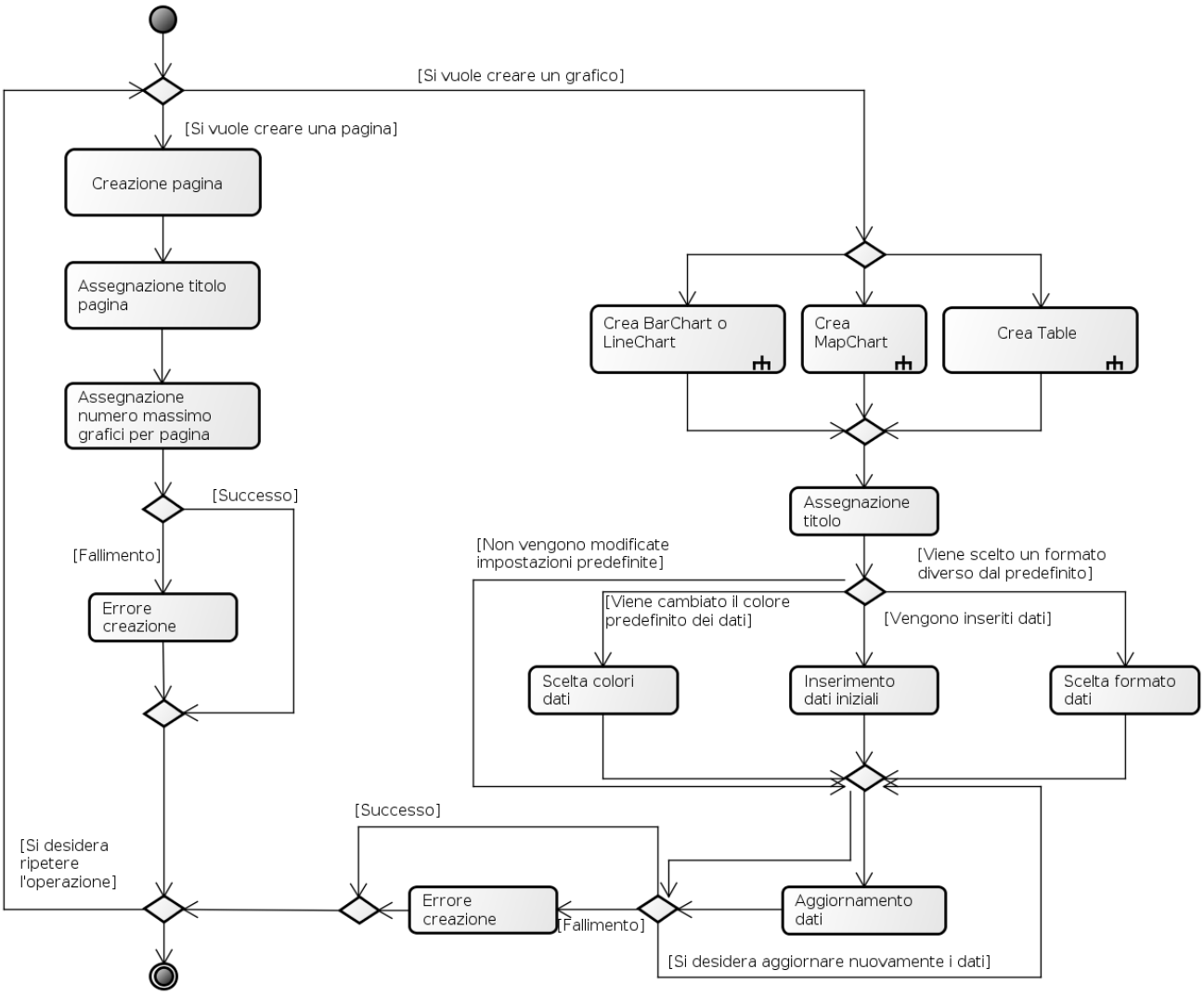

Figura 30: Diagramma di attività - Funzionalità di creazione

Dopo aver richiesto le funzionalità di Norris, lo sviluppatore può creare pagine e grafici mediante le apposite API_G. L'attività di creazione si distingue nelle due sotto attività di creazione pagina e grafico.
Per creare una pagina, lo sviluppatore deve istanziare un oggetto di tipo *page* e configurarne i parametri obbligatori, cioè il titolo e il massimo numero di grafici possibili contenuti nella stessa. Per creare un grafico invece, lo sviluppatore potrà scegliere il tipo di grafico da creare, invocando l'apposita funzione di creazione del tipo desiderato, analizzate in dettaglio nei diagrammi successivi.
È possibile modificare i parametri predefiniti comuni a tutti i tipi di grafico, nel caso non vengano specificati saranno assegnati valori di default. In caso di fallimento, si genererà un errore, altrimenti si potrà procedere con un' altra operazione, o ripetere una creazione.

6.1.3 Attività di creazione *Bar Chart*$_G$ o *Line Chart*$_G$

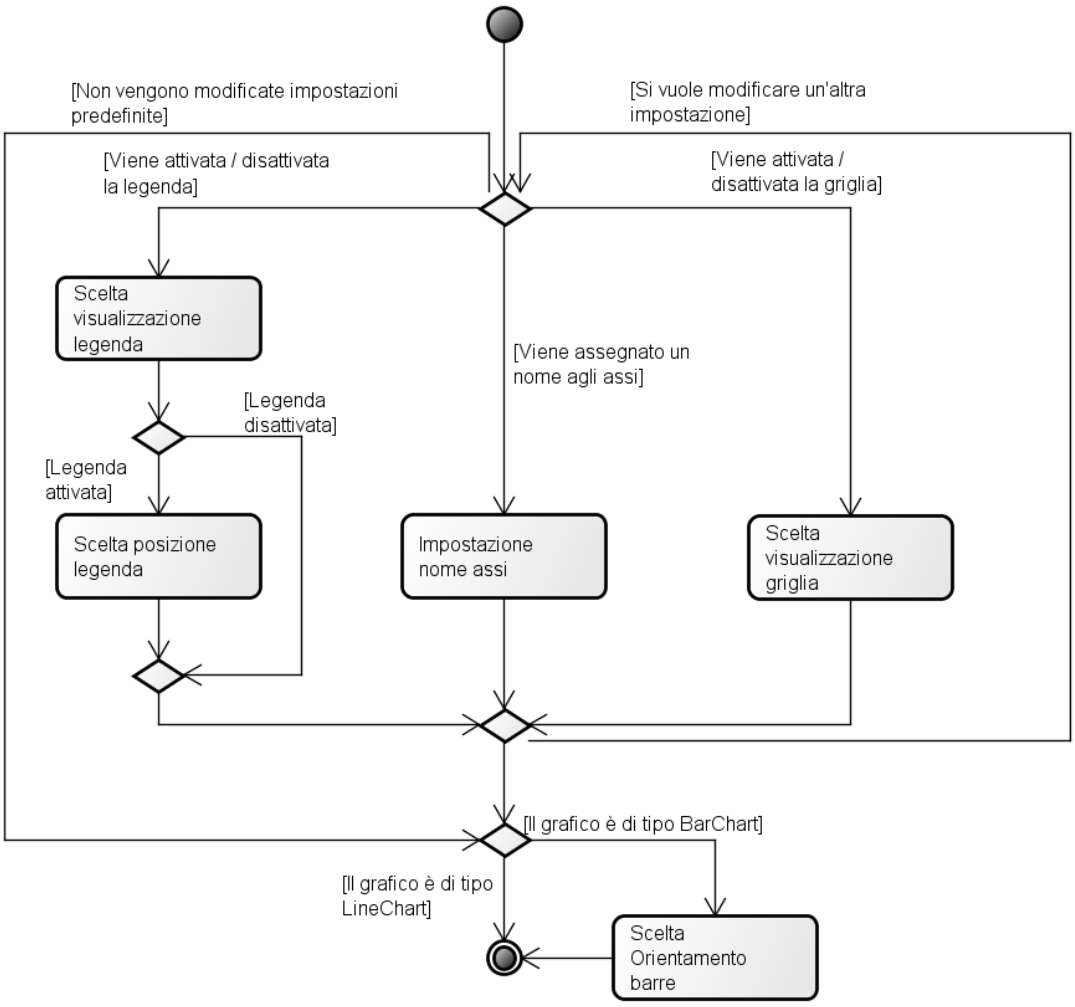

Figura 31: Diagramma di attività - Funzionalità di creazione *Bar Chart*$_G$ o *Line Chart*$_G$

Lo sviluppatore che volesse istanziare un grafico di tipo *Bar Chart*$_G$ o *Line Chart*$_G$ ha alcune proprietà che può configurare:

- Visualizzazione della legenda del grafico, e sua posizione in caso sia stata attivata;
- Etichette degli assi del grafico;
- Visualizzazione della griglia del grafico.

Per queste proprietà sono forniti dei parametri di default, e la loro configurazione è quindi da intendersi opzionale. Nel caso si tratti di un grafico di tipo *Bar Chart*$_G$ sarà possibile impostare anche l'orientamento delle barre del grafico.

6.1.4 Attività di creazione *Map Chart*$_G$

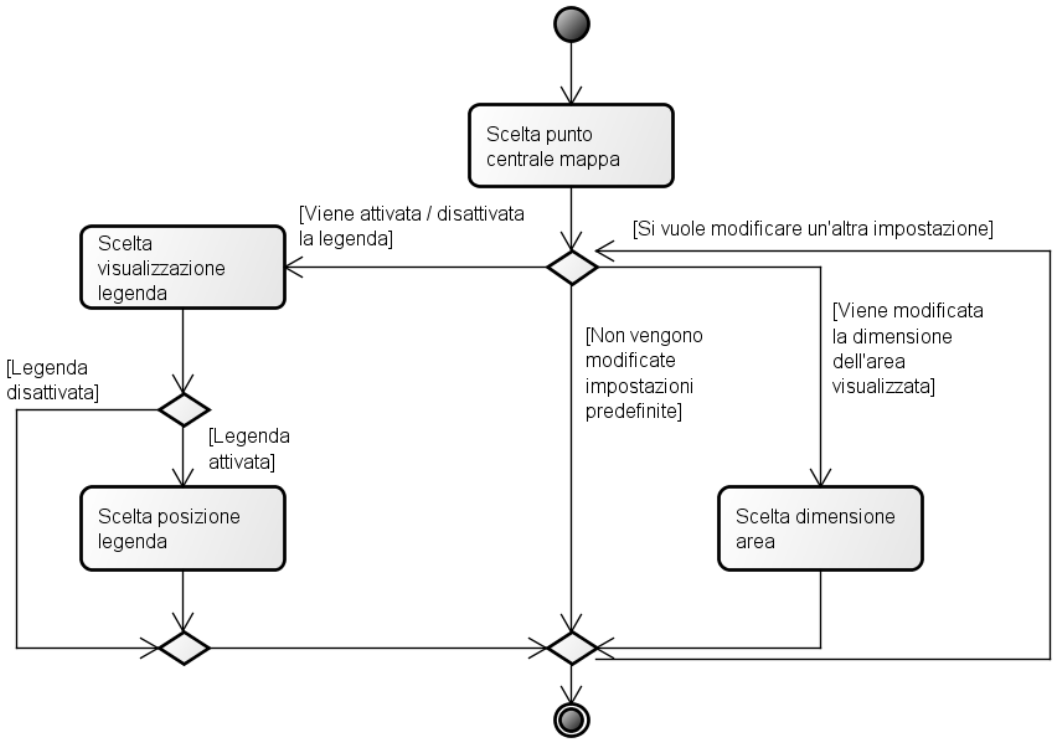

Figura 32: Diagramma di attività - Funzionalità di creazione *Map Chart*$_G$

Per istanziare un grafico di tipo *Map Chart*$_G$, è necessario fornire le coordinate del punto centrale della mappa che verrà visualizzata. Successivamente sarà possibile modificare i parametri di default relativi allo stato di:

- attivazione della legenda e relativa posizione;
- livello di zoom iniziale della mappa.

6.1.5 Attività di creazione *Table*$_G$

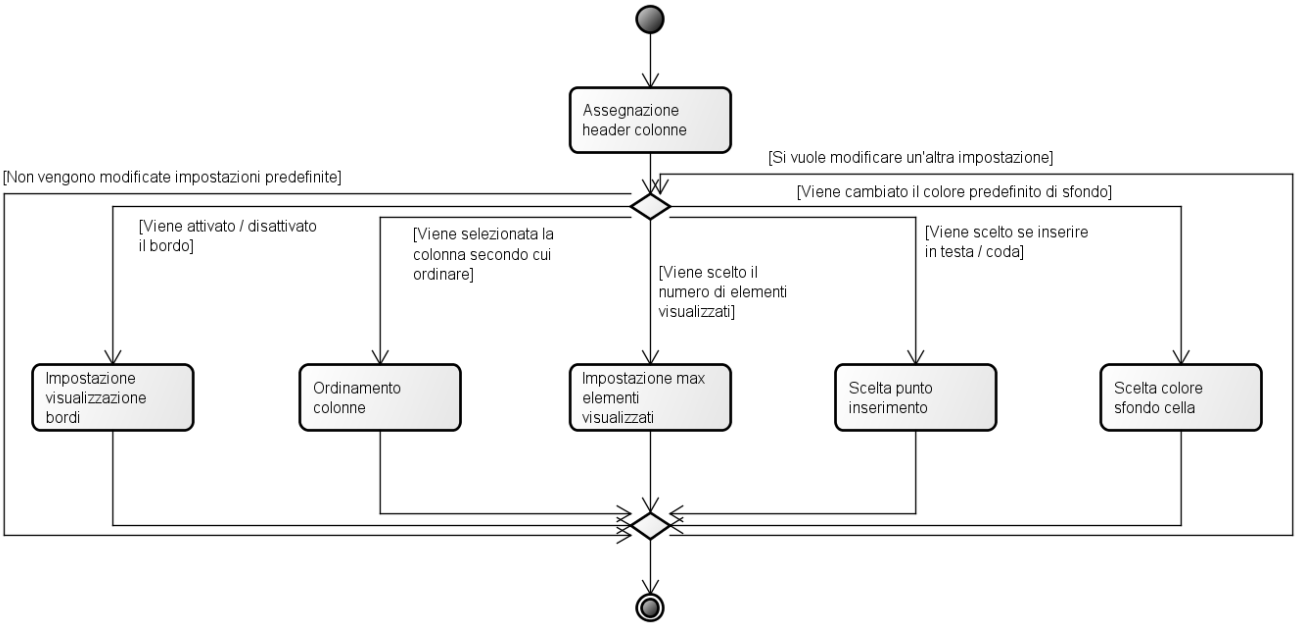

Figura 33: Diagramma di attività - Funzionalità di creazione *Table*$_G$

Per istanziare un grafico di tipo *Table*$_G$, è necessario fornire obbligatoriamente le intestazioni delle colonne della tabella. Sarà successivamente possibile modificare:

- la visualizzazione dei bordi della tabella;
- la colonna secondo la quale ordinare (in modo crescente o decrescente) i dati;
- il numero massimo di righe da mostrare simultaneamente;
- l'inserimento di nuove righe avviene in testa o in coda alla tabella;
- il colore di sfondo delle celle.

Per queste proprietà sono forniti dei parametri di default, e la loro configurazione è quindi da intendersi opzionale.

6.1.6 Attività di modifica

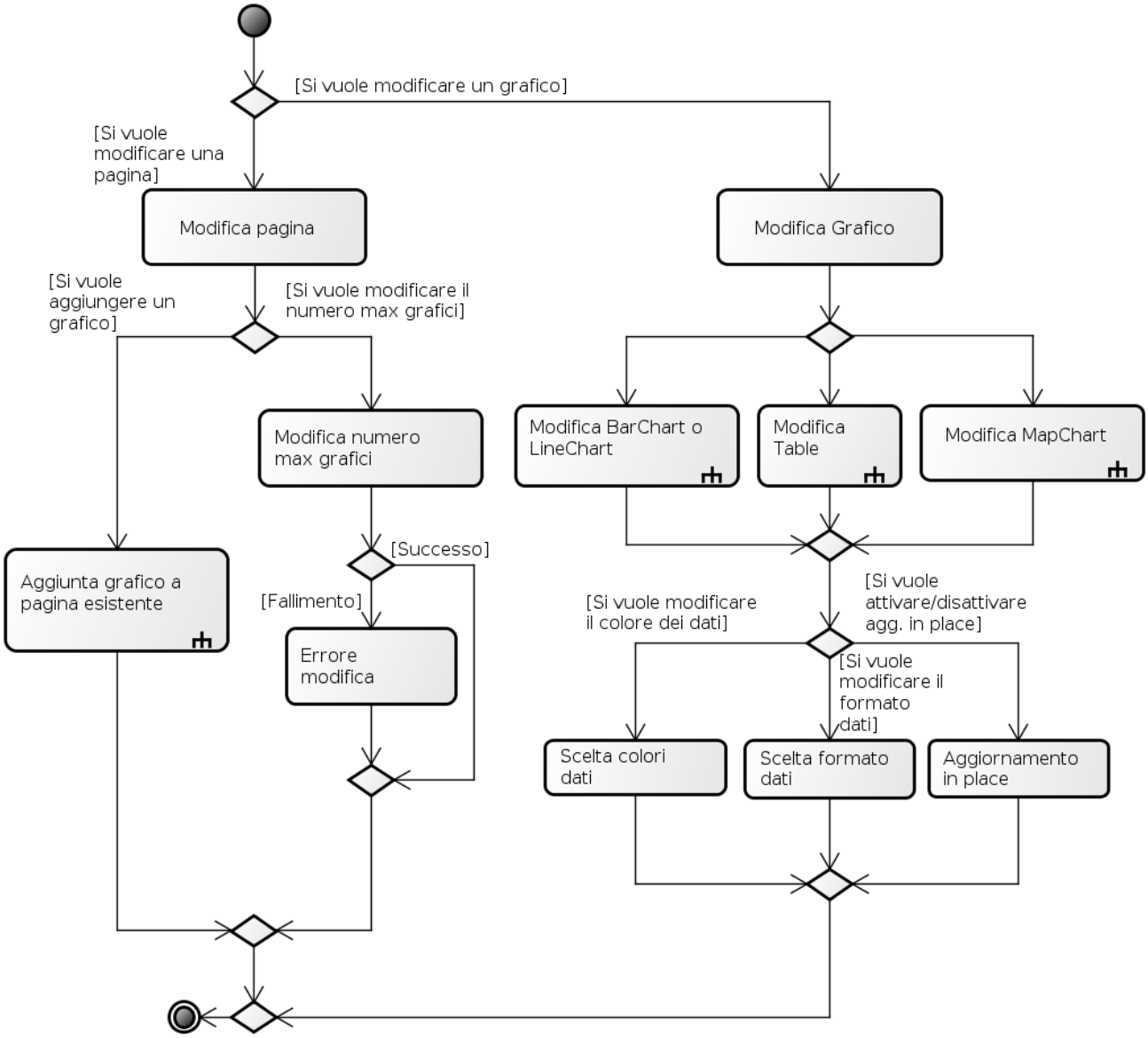

Figura 34: Diagramma di attività - Funzionalità di modifica

Dopo aver richiesto le funzionalità di Norris, lo sviluppatore può modificare pagine e grafici di un'istanza attiva mediante le apposite API_G. L'attività di modifica si distingue nelle due sotto attività di modifica pagina e grafico.
Di una pagina può essere modificato il numero massimo di grafici in essa contenuti, attività che provocherà un errore nel caso il numero venga ridotto al di sotto del numero di grafici già contenuti e attivi nella pagina stessa. È possibile anche, tramite le funzionalità di modifica, inserire un grafico precedentemente creato in una pagina.
Di un grafico invece, è possibile modificare sia i parametri specifici che quelli comuni a tutti i tipi.

6.1.7 Attività di modifica *Bar Chart*$_G$ o *Line Chart*$_G$

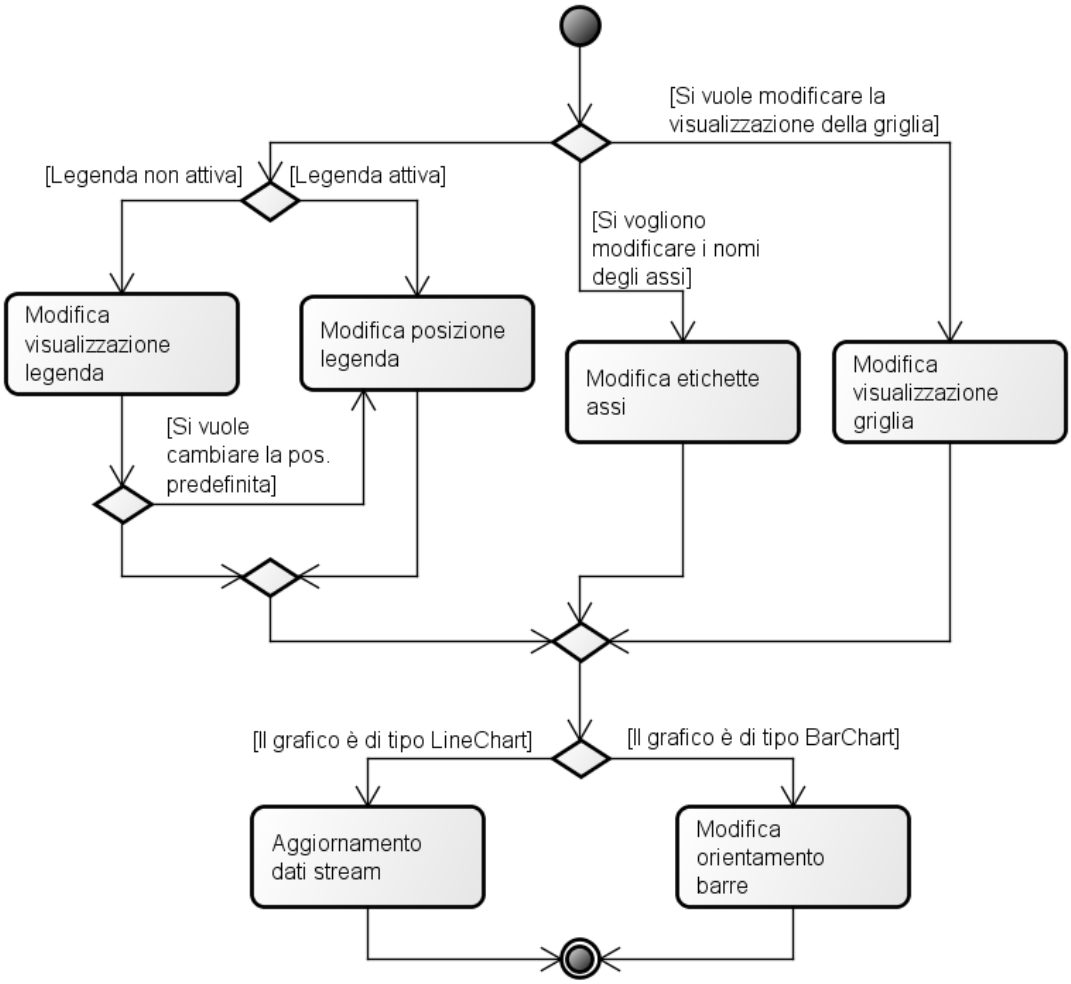

Figura 35: Diagramma di attività - Funzionalità di modifica *Bar Chart*$_G$ o *Line Chart*$_G$

Lo sviluppatore che volesse modificare le proprietà di un grafico di tipo *Bar Chart*$_G$ o *Line Chart*$_G$ può configurare:

- Visualizzazione della legenda del grafico, e sua posizione in caso sia stata attivata;
- Etichette degli assi del grafico;
- Visualizzazione della griglia del grafico.

Nel caso si tratti di un grafico di tipo *Bar Chart*$_G$ sarà possibile modificare anche l'orientamento delle barre del grafico.

6.1.8 Attività di modifica *Map Chart*$_G$

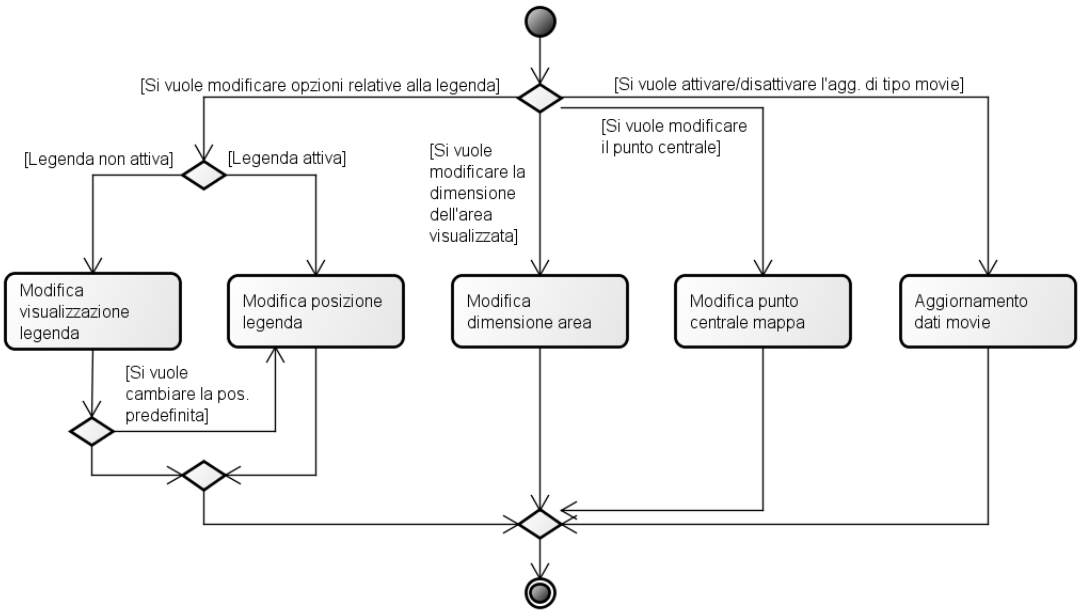

Figura 36: Diagramma di attività - Funzionalità di modifica *Map Chart*$_G$

Di un grafico di tipo *Map Chart*$_G$ è possibile modificare i parametri relativi a:

- stato di attivazione e relativa posizione della legenda;
- livello di zoom iniziale della mappa visualizzata;
- attivare o disattivare il metodo di aggiornamento movie.

6.1.9 Attività di modifica *Table*$_G$

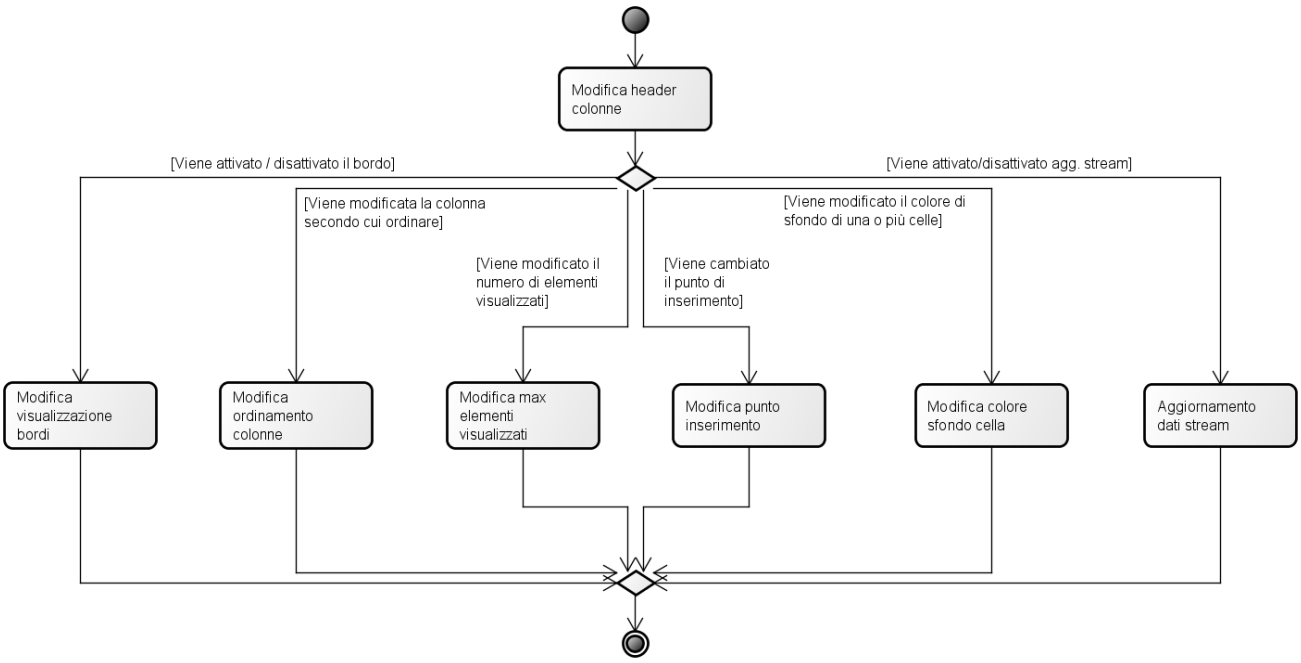

Figura 37: Diagramma di attività - Funzionalità di modifica *Table*$_G$

Di un grafico di tipo *Table*$_G$, è possibile modificare:

- la visualizzazione dei bordi della tabella;
- la colonna secondo la quale ordinare (in modo crescente o decrescente) i dati;
- il numero massimo di righe da mostrare simultaneamente;
- l'inserimento di nuove righe avviene in testa o in coda alla tabella;
- il colore di sfondo delle celle;
- lo stato di attivazione del metodo di aggiornamento stream.

Per queste proprietà sono forniti dei parametri di default, e la loro configurazione è quindi da intendersi opzionale.

6.1.10 Attività di aggiunta grafico a una pagina

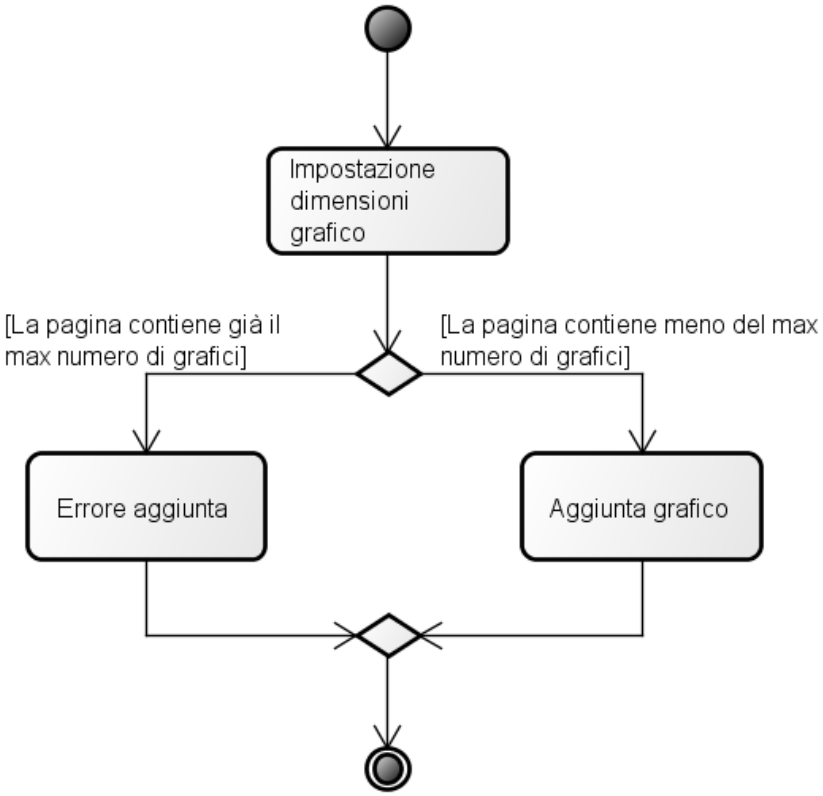

Figura 38: Diagramma di attività - Aggiunta grafico a una pagina

Lo sviluppatore, per aggiungere un grafico a una pagina che ha precedentemente creato, deve impostarne le dimensioni di visualizzazione. Il sistema verificherà poi se nella pagina sia disponibile spazio per il grafico, e effettuerà l'operazione solo nel caso questo sia possibile, mostrando altrimenti un messaggio di errore.

6.2 Funzionalità Utente

Vengono mostrate e descritte di seguito le operazioni che possono essere eseguite da un utente finale che usufruisce dei grafici creati mediante l'uso del *framework*$_G$ Norris:

6.2.1 Funzionalità generali

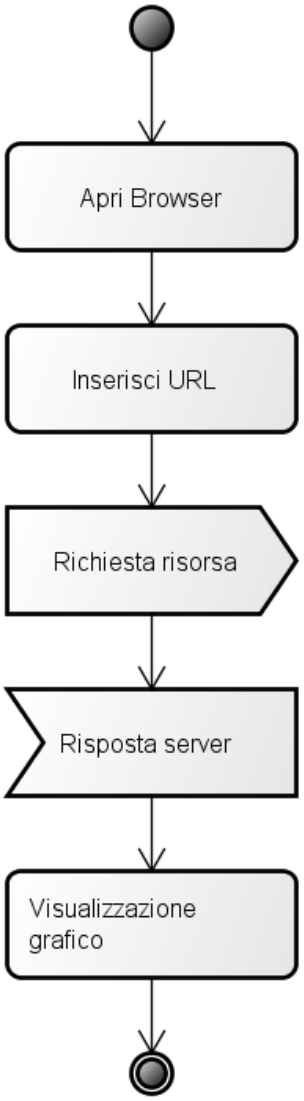

Figura 39: Diagramma di attività - Funzionalità

L'utente finale può visualizzare un grafico creato mediante il *framework*$_G$ Norris senza effettuare particolari operazioni. Il browser dell'utente effettuerà la chiamata $HTTP_G$ necessaria ad ottenere la risorsa specificata, aprendo il *socket*$_G$ necessario per ottenere gli aggiornamenti.

6.2.2 App *Android*$_G$

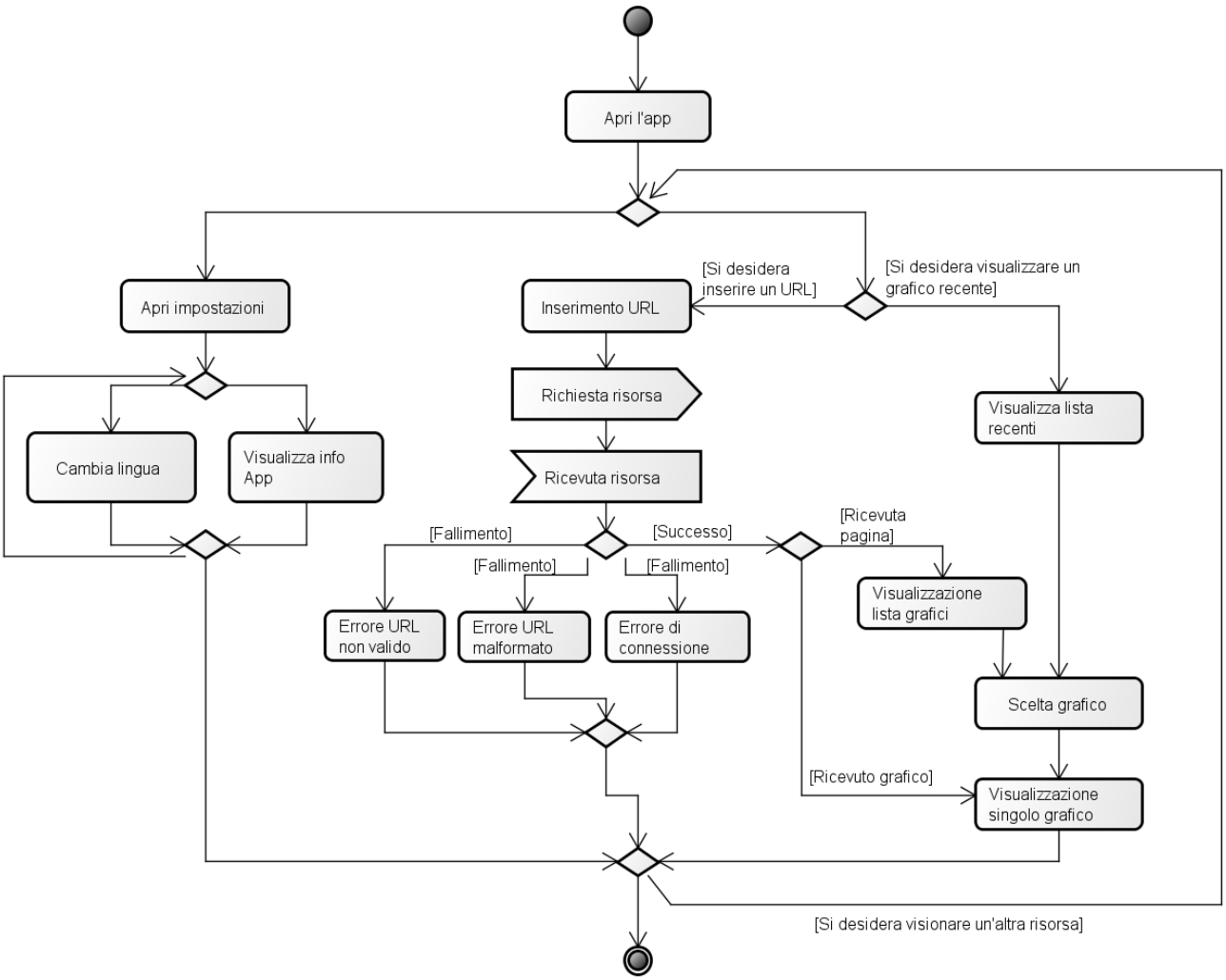

Figura 40: Diagramma di attività - App *Android*$_G$

Dopo aver aperto l'app, l'utente avrà la possibilità di inserire l'*URL*$_G$ di una risorsa Norris per visualizzarla, o di selezionare una risorsa precedentemente visualizzata, se disponibile, dalla lista Recenti. Nel primo caso, l'app effettuerà una richiesta *HTTP*$_G$ all'indirizzo specificato, che (se valido) ritornerà la risorsa richiesta. Se l'*URL*$_G$ inserita non fosse invece corretta, l'app restituirà errori specifici per varie cause:

- *URL*$_G$ non valido, nel caso non identificasse nessuna risorsa;
- *URL*$_G$ malformato, nel caso non corrispondesse alla forma di un *URL*$_G$ *HTTP*$_G$;
- Errore di connessione, nel caso la connessione fosse assente o fosse interrotta per motivi esterni.

Nel caso la risorsa ritornata invece fosse valida e fosse una pagina, l'app provvederà a mostrare una lista dei grafici in essa contenuti tra cui scegliere un singolo. Successivamente, o nel caso la risorsa ritornata fosse invece un grafico, l'app mostrerà direttamente quest'ultimo con l'orario di ultimo aggiornamento. Sarà in ogni momento

possibile tornare al passo precedente mediante il pulsante Back del sistema operativo *Android*$_G$. Dalla pagina iniziale sarà anche possibile accedere al menu delle impostazioni, in cui sarà possibile visualizzare delle informazioni sul gruppo **FlameTech Inc.** e le impostazioni sulla lingua. Sarà possibile scegliere tra inglese e italiano.

6.2.3 Caso applicativo APS Holding

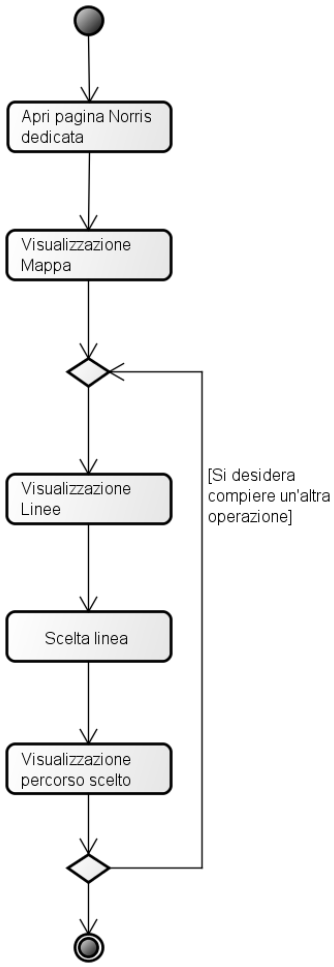

Figura 41: Diagramma di attività - Caso applicativo APS Holding

Per quanto concerne il caso applicativo, l'utente aprirà la pagina dedicata, e potrà visualizzare una mappa di Padova e un elenco di tutte le linee selezionabili. Sarà possibile selezionare le varie linee da una lista, e la mappa verrà di conseguenza aggiornata in tempo reale.
Sarà possibile ripetere l'operazione per ottenere dati di altre linee.

7 *Design Pattern*$_G$

Un *design pattern*$_G$ descrive soluzioni eleganti ai molteplici problemi ricorrenti che si presentano durante l'attività di progettazione. È fondamentale per qualsiasi progettista conoscere a fondo i *design pattern*$_G$ in quanto facilita l'attività di progettazione, favorisce la riusabilità e dà benefici enormi in termini di manutenibilità. Fondamentalmente possiamo suddividere i *design pattern*$_G$ in quattro categorie:

- ***Design pattern*$_G$ architetturali:** definiscono l'architettura dell'applicazione ad un alto livello di astrazione;
- ***Design pattern*$_G$ creazionali:** consentono di nascondere i costruttori delle classi, permettendo di creare oggetti senza conoscere la loro implementazione;
- ***Design pattern*$_G$ strutturali:** consentono di riutilizzare classi pre-esistenti, fornendo un'interfaccia adatta al riuso;
- ***Design pattern*$_G$ comportamentali:** definiscono soluzioni per le interazioni tra gli oggetti.

Per una descrizione generale e più approfondita dei *design pattern*$_G$ utilizzati si veda l'appendice *A - Descrizione Design pattern*$_G$.
Nella realizzazione del progetto Norris si è deciso di implementare i seguenti *design pattern*$_G$.

7.1 *Design pattern*$_G$ architetturali

7.1.1 *Three Tier Architecture*$_G$

- **Scopo dell'utilizzo:** suddividere l'architettura in tre diversi moduli o strati: interfaccia utente(input/output), logica funzionale e gestione dei dati persistenti, allo scopo di semplificarne l'utilizzo e la manutenzione. Il pattern impone comunicazione solo tra livelli adiacenti;
- **Contesto dell'utilizzo:** si è optato per questo *design pattern*$_G$ per l'architettura del *package*$_G$ Back-end::Lib in quanto ritenuto il più adatto e funzionale alle necessità progettuali.

7.1.2 *MVC*$_G$

- **Scopo dell'utilizzo:** è stato scelto il pattern *MVC*$_G$ per separare la logica dell'applicazione *Android*$_G$ dalla sua rappresentazione grafica;
- **Contesto dell'utilizzo:** il pattern *MVC*$_G$ è stato scelto per l'architettura generale dell'applicazione *Android*$_G$, in quanto è uno dei pattern più utilizzati per lo sviluppo di questo tipo di applicazioni, e si adatta alle necessità progettuali.

7.1.3 *MVW*$_G$

- **Scopo dell'utilizzo:** è un *design pattern*$_G$ tipico di *AngularJS*$_G$ che permette di separare la logica dell'applicazione dalla sua rappresentazione grafica. Simile al *MVC*$_G$, differisce da esso per una corrispondenza più diretta ed automatica tra la view ed il model. L'acronimo *MVW*$_G$ sta per Model-View-Whatever, dove Whatever secondo i progettisti di *AngularJS*$_G$ indica whatever works for you;

- **Contesto dell'utilizzo:** si è vista la necessità di utilizzarlo nella parte relativa al *front-end*$_G$ in quanto si userà il *framework*$_G$ *AngularJS*$_G$ per lo sviluppo.

7.2 *Design pattern*$_G$ creazionali

7.2.1 *Singleton*$_G$

- **Scopo dell'utilizzo:** assicurare che una classe abbia una sola istanza e fornire un punto d'accesso globale a tale istanza;
- **Contesto dell'utilizzo:** si è vista la necessità di utilizzarlo all'interno del *package*$_G$ DataLayer per quanto concerne la creazione della classe ActivePage in modo tale da garantire l'unicità di istanziazione di quest'ultima e di facilitarne l'accesso da parte del router.

7.2.2 *Factory Method*$_G$

- **Scopo dell'utilizzo:** definire un'interfaccia per la creazione di un oggetto, delegando alle sottoclassi la decisione sulla classe che deve essere effettivamente istanziata;
- **Contesto dell'utilizzo:** si è utilizzato questo specifico *design pattern*$_G$ per la gestione della creazione dei grafici, creando una classe Graph generica, successivamente ereditata dalle sottoclassi più specifiche.

7.3 *Design pattern*$_G$ comportamentali

7.3.1 *Chain of Responsibility*$_G$

- **Scopo dell'utilizzo:** evitare l'accoppiamento fra il mittente di una richiesta e il destinatario, consentendo a più di un singolo oggetto la possibilità di esaudire la richiesta tramite la concatenazione degli oggetti destinatari, permettendo così il passaggio della richiesta di oggetto in oggetto finché uno di questi riesce a soddisfarla;
- **Contesto dell'utilizzo:** nel progetto Norris il *design pattern*$_G$ *Chain of Responsibility*$_G$ è stato applicato lato *back-end*$_G$ per la gestione delle connessioni con i *client*$_G$. Le richieste giungeranno alla classe RequestHandler che definisce l'interfaccia comune e implementa il link da seguire nella catena. Le classi RequestHandler e RouterHandler si occuperanno di controllare se sono in grado di gestire la richiesta e, in caso contrario, quest'ultima sarà gestita dalle altre classi della catena ovvero NotFoundError o ServerError.

7.3.2 *Observer*$_G$

- **Scopo dell'utilizzo:** definire una dipendenza uno a molti fra oggetti in modo tale che, se un oggetto cambia il suo stato, tutti gli oggetti dipendenti da questo vengano informati e aggiornati automaticamente;
- **Contesto dell'utilizzo:** all'interno del progetto Norris l'*Observer*$_G$ è stato adoperato per segnalare alla classe ClientUpdater le operazioni eseguite dalla classe Updater.

8 Stime di fattibilità e di bisogno di risorse

L'architettura definita precedentemente ha raggiunto un livello di dettaglio sufficiente a fornire una stima di fattibilità e di bisogno di risorse.
L'analisi dell'architettura progettata ha permesso di constatare che le tecnologie che si è scelto di utilizzare risultano sufficientemente adeguate per la realizzazione del prodotto e per ricoprire le necessità progettuali. In particolare, nonostante il linguaggio *JavaScript*$_G$ non presenti direttamente i costrutti utilizzati da altri linguaggi di programmazione orientati agli oggetti, sono presenti soluzioni alternative per ottenere il medesimo comportamento.
La maggior parte delle tecnologie e degli strumenti sono poco conosciute ai membri del gruppo; questi si impegneranno a colmare le proprie lacune sugli argomenti interessati tramite materiale fornito dall'amministratore ed eventuali ulteriori approfondimenti personali.

9 Tracciamento

Seguono le tabelle di tracciamento tra componenti e requisiti. Per semplicità di lettura, requisiti associati a figli di un componente non sono riportati anche nel padre.

9.1 Tracciamento componenti - requisiti

Componente	Requisiti
Norris	RAF3.5 RAV1
Norris::Back-end	RAV1
Norris::Back-end::DeveloperProject	RBV8.1 RAV8.2
Norris::Back-end::Lib	RAF1
Norris::Back-end::Lib::BusinessLayer	RAF1.2 RBF1.4 RAF4 RAF4.1 RAF4.2

Componente	Requisiti
Norris::Back-end::Lib::DataLayer	RAF1.2.1 RAF1.2.2 RAF1.2.3 RAF1.3 RAF2 RAF2.1 RAF2.2 RAF2.3 RAF2.4 RAF3 RAF3.1 RAF3.2 RAF3.3 RAF3.4 RAF4.1.1 RAF4.1.10 RAF4.1.11 RAF4.1.12 RAF4.1.13 RAF4.1.14 RAF4.1.15 RAF4.1.16 RAF4.1.17 RAF4.1.2 RAF4.1.3 RAF4.1.4 RAF4.1.5 RAF4.1.6 RAF4.1.7 RAF4.1.8 RAF4.1.9 RAF4.2.1 RAF4.2.10 RAF4.2.11 RAF4.2.12 RAF4.2.13 RAF4.2.14 RAF4.2.2 RAF4.2.3 RAF4.2.4 RAF4.2.5 RAF4.2.6 RAF4.2.7 RAF4.2.8 RAF4.2.9

Componente	Requisiti
Norris::Back-end::Lib::PresentationLayer	RAF1 RAF1.1 RBF1.4 RAV2 RAV3
Norris::Front-end	RAF3.5 RAF5
Norris::Front-end::Controller	RAF3.5.1
Norris::Front-end::Model	RAF3.5.1
Norris::Front-end::Service	RAF3.5.1
Norris::Front-end::View	RAF3.5.1 RBF3.5.2 RAF5.1 RAF5.2 RAF5.2.1
AndroidApp	RAF3.5 RCF3.5.3 RAV9
AndroidApp::Activities	RCF3.5.3.1 RCF3.5.3.1.1 RCF3.5.3.1.2 RCF3.5.3.4.1 RCF3.5.3.4.2
AndroidApp::AppModel	RCF3.5.3.3
AndroidApp::Layouts	RCF3.5.3.2 RCF3.5.3.4

Tabella 3: Tabella Tracciamento Componenti - Requisiti

9.2 Tracciamento requisiti - componenti

Requisito	Componenti
RAF1	Norris::Back-end::Lib Norris::Back-end::Lib::PresentationLayer
RAF1.1	Norris::Back-end::Lib::PresentationLayer
RAF1.2	Norris::Back-end::Lib::BusinessLayer
RAF1.2.1	Norris::Back-end::Lib::DataLayer
RAF1.2.2	Norris::Back-end::Lib::DataLayer
RAF1.2.3	Norris::Back-end::Lib::DataLayer
RAF1.3	Norris::Back-end::Lib::DataLayer
RBF1.4	Norris::Back-end::Lib::BusinessLayer Norris::Back-end::Lib::PresentationLayer
RAF2	Norris::Back-end::Lib::DataLayer
RAF2.1	Norris::Back-end::Lib::DataLayer
RAF2.2	Norris::Back-end::Lib::DataLayer
RAF2.3	Norris::Back-end::Lib::DataLayer
RAF2.4	Norris::Back-end::Lib::DataLayer
RAF3	Norris::Back-end::Lib::DataLayer
RAF3.1	Norris::Back-end::Lib::DataLayer
RAF3.2	Norris::Back-end::Lib::DataLayer
RAF3.3	Norris::Back-end::Lib::DataLayer
RAF3.4	Norris::Back-end::Lib::DataLayer
RAF3.5	AndroidApp Norris Norris::Front-end
RAF3.5.1	Norris::Front-end::Controller Norris::Front-end::Model Norris::Front-end::Service Norris::Front-end::View
RBF3.5.2	Norris::Front-end::View

Requisito	Componenti
RCF3.5.3	AndroidApp
RCF3.5.3.1	AndroidApp::Activities
RCF3.5.3.1.1	AndroidApp::Activities
RCF3.5.3.1.2	AndroidApp::Activities
RCF3.5.3.2	AndroidApp::Layouts
RCF3.5.3.3	AndroidApp::AppModel
RCF3.5.3.4	AndroidApp::Layouts
RCF3.5.3.4.1	AndroidApp::Activities
RCF3.5.3.4.2	AndroidApp::Activities
RAF4	Norris::Back-end::Lib::BusinessLayer
RAF4.1	Norris::Back-end::Lib::BusinessLayer
RAF4.1.1	Norris::Back-end::Lib::DataLayer
RAF4.1.2	Norris::Back-end::Lib::DataLayer
RAF4.1.3	Norris::Back-end::Lib::DataLayer
RAF4.1.4	Norris::Back-end::Lib::DataLayer
RAF4.1.5	Norris::Back-end::Lib::DataLayer
RAF4.1.6	Norris::Back-end::Lib::DataLayer
RAF4.1.7	Norris::Back-end::Lib::DataLayer
RAF4.1.8	Norris::Back-end::Lib::DataLayer
RAF4.1.9	Norris::Back-end::Lib::DataLayer
RAF4.1.10	Norris::Back-end::Lib::DataLayer
RAF4.1.11	Norris::Back-end::Lib::DataLayer
RAF4.1.12	Norris::Back-end::Lib::DataLayer
RAF4.1.13	Norris::Back-end::Lib::DataLayer
RAF4.1.14	Norris::Back-end::Lib::DataLayer
RAF4.1.15	Norris::Back-end::Lib::DataLayer

Requisito	Componenti
RAF4.1.16	Norris::Back-end::Lib::DataLayer
RAF4.1.17	Norris::Back-end::Lib::DataLayer
RAF4.2	Norris::Back-end::Lib::BusinessLayer
RAF4.2.1	Norris::Back-end::Lib::DataLayer
RAF4.2.2	Norris::Back-end::Lib::DataLayer
RAF4.2.3	Norris::Back-end::Lib::DataLayer
RAF4.2.4	Norris::Back-end::Lib::DataLayer
RAF4.2.5	Norris::Back-end::Lib::DataLayer
RAF4.2.6	Norris::Back-end::Lib::DataLayer
RAF4.2.7	Norris::Back-end::Lib::DataLayer
RAF4.2.8	Norris::Back-end::Lib::DataLayer
RAF4.2.9	Norris::Back-end::Lib::DataLayer
RAF4.2.10	Norris::Back-end::Lib::DataLayer
RAF4.2.11	Norris::Back-end::Lib::DataLayer
RAF4.2.12	Norris::Back-end::Lib::DataLayer
RAF4.2.13	Norris::Back-end::Lib::DataLayer
RAF4.2.14	Norris::Back-end::Lib::DataLayer
RAF5	Norris::Front-end
RAF5.1	Norris::Front-end::View
RAF5.2	Norris::Front-end::View
RAF5.2.1	Norris::Front-end::View
RAV1	Norris Norris::Back-end
RAV2	Norris::Back-end::Lib::PresentationLayer
RAV3	Norris::Back-end::Lib::PresentationLayer
RBV8.1	Norris::Back-end::DeveloperProject

Requisito	Componenti
RAV8.2	Norris::Back-end::DeveloperProject
RAV9	AndroidApp

Tabella 4: Tabella Tracciamento Requisiti - Componenti

A Descrizione *Design pattern*$_G$

A.1 *Design pattern*$_G$ architetturali

A.1.1 *Three Tier Architecture*$_G$

- **Descrizione:** questo *design pattern*$_G$ indica una particolare architettura software di tipo multi-tier che prevede la suddivisione del sistema in tre diversi moduli o strati dedicati rispettivamente all'interfaccia utente, alla logica funzionale e alla gestione dei dati persistenti. Tali moduli interagiscono tra loro secondo le linee generali del paradigma *client*$_G$-*server*$_G$ utilizzando interfacce ben definite, in modo che ciascuno dei tre moduli possa essere modificato o sostituito indipendentemente dagli altri, conferendo quindi un alto grado di manutenibilità al sistema;
- **Motivazione:** il beneficio principale apportato da questo paradigma risiede nel fatto che ogni livello può essere cambiato o aggiornato senza dover propagare le modifiche ai livelli adiacenti. Tale vantaggio deriva principalmente dal raggruppamento delle varie funzionalità in gruppi disgiunti ma in stretta collaborazione tra loro e dal fatto che la comunicazione può avvenire solo tra livelli adiacenti. Tuttavia, la struttura di tale *design pattern*$_G$ risulta particolarmente calzante per dei servizi basati su architettura *client*$_G$-*server*$_G$, in quanto ogni livello non esiste semplicemente come raggruppamento logico a se stante, ma il suo ruolo viene adattato in relazione allo specifico ambiente di rete in cui viene eseguito: nel caso in cui la morfologia della rete dovesse cambiare, basterà aggiornare lo strato che opera in quel determinato ambiente;
- **Ambito applicativo:** il *Three Tier Architecture*$_G$ viene generalmente usato nella progettazione di *middleware*$_G$ o di applicazioni web.

A.1.2 *MVW*$_G$

- **Descrizione:** si tratta di un *design pattern*$_G$ tipico di *AngularJS*$_G$, che consente agli utilizzatori di integrare il Model-View con qualunque altra cosa si ritenga più appropriata, per l'appunto whatever works for you;
- **Motivazione:** è il *design pattern*$_G$ più adatto a lavorare con *AngularJS*$_G$ e permette di separare la logica dell'applicazione dalla sua rappresentazione grafica. Simile al *MVC*$_G$, differisce per una corrispondenza più diretta e automatica tra la view e il model;
- **Ambito applicativo:** questo *design pattern*$_G$ viene spesso usato quando si lavora con *AngularJS*$_G$.

A.1.3 *MVC*$_G$

- **Descrizione:** il *design pattern*$_G$ *MVC*$_G$ permette una completa disgiunzione tra le funzionalità di vista e di modello dei dati. Si individuano tre componenti:
  1. **Model:** rappresenta la logica di memorizzazione e recupero di dati utilizzati nel sistema;
  2. **View:** visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti esterni;

3. **Controller:** riceve i comandi dell'utente (in genere attraverso la View) e li attua modificando lo stato degli altri due componenti.

- **Motivazione:** tale *design pattern*$_G$ permette la separazione tra la modellazione del dominio, la presentazione e le azioni basate sugli input degli utenti suddividendoli in tre pacchetti separati. Questo schema implica anche la tradizionale separazione fra la logica applicativa, a carico del Controller e del Model, e l'interfaccia utente, a carico della View;

- **Ambito applicativo:** il pattern *MVC*$_G$ viene usato quando si vuole disaccoppiare View e Model instaurando un protocollo di sottoscrizione e notifica tra loro, o quando si vogliono agganciare più View ad un Model per fornire più rappresentazioni del Model stesso.

A.2 *Design pattern*$_G$ creazionali

A.2.1 *Singleton*$_G$

- **Descrizione:** il *design pattern*$_G$ *Singleton*$_G$ assicura che una classe abbia un'unica istanza e fornisce un punto d'accesso globale a tale istanza;

- **Motivazione:** garantire un risparmio di risorse fisiche poiché la creazione di un'istanza dell'oggetto in questione è rimandata fino al momento del suo effettivo utilizzo;

- **Ambito applicativo:** tale *design pattern*$_G$ viene utilizzato nei seguenti casi:

  1. si desidera avere la garanzia che nel sistema vi sia una sola istanza di oggetti di un determinato tipo, e che tali oggetti siano accessibili da un punto di accesso ben preciso;
  2. si desidera che il controllo di unicità di un oggetto non venga delegato ad altre entità;
  3. si desidera che più oggetti condividano un unico pool di dati.

A.2.2 *Factory Method*$_G$

- **Descrizione:** il *design pattern*$_G$ *Factory Method*$_G$ definisce un'interfaccia per la creazione di un oggetto, delegando alle sottoclassi la decisione sulla classe che deve essere istanziata. Inoltre, consente di deferire l'istanziazione di una classe alle sottoclassi;

- **Motivazione:** la creazione di un oggetto può spesso richiedere processi complessi la cui collocazione all'interno della classe di composizione potrebbe non essere appropriata.
  Essa può, inoltre, comportare duplicazione di codice, richiedere informazioni non accessibili alla classe di composizione o non fornire un sufficiente livello di astrazione. Il *Factory Method*$_G$ indirizza questi problemi definendo un metodo separato per la creazione degli oggetti. Tale metodo può essere ridefinito dalle sottoclassi per determinare il tipo derivato di prodotto che verrà effettivamente creato;

- **Ambito applicativo:** questo *design pattern*$_G$ viene utilizzato nei seguenti casi:

1. una classe non è in grado di sapere in anticipo le classi degli oggetti che deve creare;
2. una classe vuole che le sue sottoclassi scelgano gli oggetti da creare;
3. le classi delegano la responsabilità a una o più classi di supporto, e si vuole localizzare in un punto ben preciso la conoscenza di quale o quali classi di supporto vengano delegate.

A.3 *Design pattern*$_G$ comportamentali

A.3.1 *Chain of Responsibility*$_G$

- **Descrizione:** tale *design pattern*$_G$ si pone come scopo quello di disaccoppiare il mittente di una richiesta dal suo destinatario, consentendo che più di un singolo oggetto possa esaudire la richiesta, e di concatenare gli oggetti destinatari e passare la richiesta di oggetto in oggetto finché uno di questi non riesce ad esaudirla;
- **Motivazione:** data la sua particolare tipologia di costruzione, questo *design pattern*$_G$ offre l'opportunità di costruire una collezione di oggetti senza alcun tipo di accoppiamento fra loro. Questo permetterà di modificare la catena a seconda delle necessità in quanto il mittente non avrà alcun modo di sapere come verrà gestita la sua richiesta. Tale richiesta verrà presa in consegna da un handler comune e verrà passata fra tutti i possibili destinatari fino a quando un oggetto sarà in grado di gestirla in maniera appropriata. Il passaggio può avvenire in un solo verso, pertanto, in caso di richiesta non gestita, non si formerà un loop.
  Grazie a queste sue caratteristiche risulta essere uno dei *design pattern*$_G$ comportamentali più comuni durante l'utilizzo di *Express*;
- **Ambito applicativo:** è opportuno utilizzare questo *design pattern*$_G$ quando:
  1. più di un oggetto può gestire la richiesta ed il ricevente che la gestirà non è conosciuto a priori. Il ricevente dovrebbe essere scelto in modo automatico;
  2. si vuole passare una richiesta a uno dei molti oggetti, senza specificare esplicitamente il ricevente;
  3. l'insieme di oggetti che gestirà una richiesta deve essere definito dinamicamente.

A.3.2 *Observer*$_G$

- **Descrizione:** il *design pattern*$_G$ *Observer*$_G$ ha lo scopo di definire una dipendenza uno a molti fra oggetti, in modo tale che se un oggetto cambia il suo stato, tutti gli oggetti dipendenti da esso siano notificati e aggiornati automaticamente;
- **Motivazione:** un effetto collaterale comune al partizionare un sistema in insiemi di classi collaboranti è il bisogno di mantenere un alto livello di consistenza fra classi correlate. La consistenza non deve essere ottenuta a scapito dell'accoppiamento, che deve rimanere basso, altrimenti la riusabilità delle classi sarebbe ridotta. Il pattern *Observer*$_G$ dettaglia come impostare le relazioni di dipendenza tra le classi per ovviare a questo problema;
- **Ambito applicativo:** tale *design pattern*$_G$ viene utilizzato quando:

1. un'astrazione presenta due aspetti, di cui uno dipendente dall'altro. Incapsulando questi aspetti in due oggetti separati è possibile riusarli indipendentemente;
2. una modifica a un oggetto richiede modifiche ad altri oggetti che dipendono da questo, ma in generale non si conosce il numero degli oggetti dipendenti;
3. un oggetto ha bisogno di notificare ad altri oggetti senza conoscerne l'identità precisa. In altre parole si vuole mantenere un alto livello di disaccoppiamento.