NetBreak

Progetto API Market

Manuale Manutentore

Informazioni sul documento

Nome del documento	ManualeManutentore 1_0_0.pdf
Data di creazione	15 maggio 2017
Ultima modifica	16 giugno 2017
Versione	1.0.0
Stato	Approvato
Redatto da	Marco Casagrande Dan Serbanoiu Davide Scarparo Alberto Nicolè
Verificato da	Andrea Scalabrin Nicolò Scapin
Approvato da	Davide Scarparo
Uso	Esterno
Distribuzione	NetBreak
Destinato a	Prof. Tullio Vardanega, Prof. Riccardo Cardin, NetBreak
Email di riferimento	netbreakswe@gmail.com

Abstract

Documento contenente il Manuale Manutentore per l'implementazione di nuove funzionalità e/o l'ordinaria amministrazione e manutenzione della piattaforma API Market.

Changelog

Versione	Data	Autore	Ruolo	Descrizione
1.0.0	2017-06-16	Davide Scarparo	Responsabile	Approvazione documento
0.2.0	2017-06-15	Nicolò Scapin	Verificatore	Verifica documento
0.1.3	2017-06-08	Andrea Scalabrin	Verificatore	Stesura della sezione#7: "Packages Back-end"
0.1.2	2017-06-06	Andrea Scalabrin	Verificatore	Stesura delle sezioni #5:"Architettura generale" e #6: "Packages Front-end"
0.1.1	2017-06-01	Andrea Scalabrin	Verificatore	Stesura della sezione #4: "Configurazioni software"
0.1.0	2017-05-27	Andrea Scalabrin	Verificatore	Verifica documento
0.0.4	2017-05-20	Alberto Nicolè	Verificatore	Stesura della sezione #3: "Tecnologie"
0.0.3	2017-05-17	Nicolò Scapin	Verificatore	Stesura della sezione #2: "requisiti di sistema"
0.0.2	2017-05-15	Andrea Scalabrin	Verificatore	Stesura della sezione #1: "Introduzione"
0.0.1	2017-05-15	Andrea Scalabrin	Verificatore	Creazione template

Indice

1 Introduzione

1.1 Scopo del documento

Il documento rappresenta il Manuale per il manutentore che installerà ed effettuerà le manutenzioni al software *API Market*$_G$ sviluppato dal gruppo NetBreak. Saranno presentate le modalità di utilizzo e la descrizione in dettaglio dell'applicazione in un ottica di manutenzione esterna o aggiunta di nuove funzionalità.

1.2 Scopo del prodotto

Lo scopo del prodotto è la realizzazione di un API Market per l'acquisto e la vendita di *microservizi*$_G$. Il sistema offrirà la possibilità di registrare nuove *API*$_G$ per la vendita, permetterà la consultazione e la ricerca di API ai potenziali acquirenti, gestendo i permessi di accesso ed utilizzo tramite creazione e controllo di relative *API key*$_G$. Il sistema, oltre alla *web app*$_G$ stessa, sarà corredato di un *API Gateway*$_G$ per la gestione delle richieste e il controllo delle chiavi, e fornirà funzionalità avanzate di statistiche per il gestore della piattaforma e per i fornitori dei microservizi.

1.3 Riferimenti informativi

- **Ingegneria del software - Ian Sommerville - 8a edizione:**
  Porzione dedicata alla Progettazione Architetturale (Capitolo 11)
- **Slides del corso - Design patterns strutturali**
  `http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/E04.pdf`
- **Slides del corso - Design patterns creazionali**
  `http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/E05.pdf`
- **Slides del corso - Design patterns architetturali**
  - `http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/E08.pdf`
  - `http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/E09.pdf`
- **Slides del corso - Diagrammi dei packages**
  `http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/E02b.pdf`
- **Slides del corso - Diagrammi delle classi**
  `http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/E02a.pdf`
- **Slides del corso - Diagrammi di sequenza**
  `http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/E03a.pdf`
- **Slides del corso - Diagrammi di attività**
  `http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/E03b.pdf`
- **HTML5**
  `https://www.w3.org/TR/html5/`
- **Bootstrap CSS Framework**
  `http://getbootstrap.com/`
- **AngularJS**
  `https://www.angularjs.org/`
- **JQuery JavaScript Library**
  `https://jquery.com/`

- **Jolie Programming Language**
  `http://www.jolie-lang.org/`
- **Java Programming Language**
  `https://www.oracle.com/it/java/index.html`
- **MySQL Database**
  `https://www.mysql.it/`
- **Chris Richardson**
  - `http://microservices.io`
  - `http://microservices.io/patterns/microservices.html`
- **Martin Fowler**
  - `https://martinfowler.com/articles/microservices.html`
  - `https://martinfowler.com/microservices/`
- **IBM**
  `http://www.ibm.com/developerworks/websphere/library/techarticles/1601_clark-trs/1601_clark.html`
- **NGINX**
  - `https://www.nginx.com/blog/introduction-to-microservices/`
  - `https://www.nginx.com/blog/building-microservices-using-an-api-gateway/`

1.4 Glossario

Per semplificare la consultazione e disambiguare alcune terminologie tecniche, le voci indicate con la lettera *G* a pedice sono descritte approfonditamente nel documento Glossario 2_0_0.pdf e specificate solo alla prima occorrenza all'interno del suddetto documento.

2 Requisiti di sistema

2.1 Categorie di utenza

Sono presenti due categorie di utilizzo per l'utente. Esse permettono di accedere in modo differente alle varie aree del sito. Riassumiamo nell'elenco sottostante le categorie presentate:

- **Utente non autenticato**: rappresenta l'utente che visita il sito per la prima volta oppure quando non ha effettuato il login nella piattaforma. All'utente non autenticato è permessa la visualizzazione dei prodotti inseriti nella piattaforma, così come la ricerca. Essi però non hanno accesso alle aree e funzionalità riservate;
- **Utente autenticato**: un utente appartenente a questa categoria possiede le funzionalità di ricerca come per l'utenza non autenticata. Inoltre, ha a disposizione il proprio profilo utente ed è abilitato all'acquisto dei servizi offerti nel sito, con l'accesso alle relative funzionalità statistiche.
- **Utente autenticato (sviluppatore)**: questa categoria di utente è una sottoclasse dell'utente autenticato, ma risulta abilitato all'inserimento e gestione dei propri servizi per la vendita ad altri utenti della piattaforma.

2.2 Requisiti di sistema

APIM, essendo un applicazione web-based, richiede l'accesso ad internet per poter essere utilizzato. La compatibilità *browser*$_G$ è riassumibile nell'elenco sottostante, fermo restando che seppur non garantita potrebbe esserci compatibilità con versioni antecedenti a quelle indicate. L'utente per usufruire della piattaforma non deve installare alcun componente, ma deve attivare *javascript*$_G$ sul browser web scelto per usufruire dei contenuti della *web app*$_G$. L'hardware del dispositivo non è oggetto di vincolo all'utilizzo.

2.2.1 Browser supportati

Il software API Market supporta i browser a partire dalle versioni: *Google Chrome*$_G$ 56.0, Mozilla Firefox 51.0, Safari 10.0, Microsoft Edge 38.0, Android browser 5.1 e Safari per *iOS*$_G$ 10.

3 Tecnologie

3.1 AngularJS

Per lo sviluppo della parte *Front-End*$_G$ dell'applicazione si è scelto di usare il *framework*$_G$ *JavaScript*$_G$ *AngularJS*$_G$, che permette lo sviluppo di applicazioni in singola pagina. Esso consente di utilizzare HTML$_G$ come linguaggio di template e permette di estenderne la sintassi per esprimere i componenti dell'applicazione in modo chiaro e conciso.

3.2 Node.js

La Web app necessita di *node.js*$_G$ per essere eseguita. Esso permette di realizzare applicazioni Web, come appunto API Market, utilizzando il linguaggio JavaScript, tipicamente client-side, per la scrittura server-side. La caratteristica principale di Node.js risiede nella possibilità che offre di accedere alle risorse del sistema operativo in modalità *event-driven*$_G$ non sfruttando il classico modello basato su processi o thread concorrenti, utilizzato dai classici web server. In particolare, si utilizzano diversi pacchetti npm dall'ecosistema Node.js quali Http-server e Bower.

3.3 MySQL

MySQL o *Database MySQL*$_G$ è il più diffuso Relational Database Management system (RDBMS) composto da un client a riga di comando e un server. Entrambi i software sono disponibili sia per sistemi Unix e Unix-like che per Windows; le piattaforme principali di riferimento sono Linux e Oracle Solaris. Il software supporta numerosi applicativi per la corretta gestione dei database ad esso associato, ed è rilasciato con licenza Open Source da Oracle.

3.4 Jolie

Jolie$_G$ è il primo linguaggio esplicitamente orientato ai microservizi. E' un linguaggio open-source utilizzato per lo sviluppo back-end a microservizi di API Market. Ogni microservizio del progetto è stato sviluppato a partire da questo linguaggio e integrato con opportune librerie Java. Jolie infatti supporta nativamente un integrazione con Java, dal quale deriva direttamente.

3.5 Java

Java$_G$ è un linguaggio di programmazione ad alto livello, orientato agli oggetti e a tipizzazione statica, specificatamente progettato per essere il più possibile indipendente dalla piattaforma di esecuzione. Vista la diffusione di Java e l'integrazione nativa con il linguaggio Jolie, esso si presta per l'integrazione di librerie di terze parti, quali ad esempio controlli JDBC per la connessione ai database.

3.6 Javascript

JavaScript è un linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client per la creazione, in siti web e applicazioni web, di effetti dinamici interattivi tramite funzioni di script invocate da eventi innescati a loro volta in vari modi dall'utente sulla pagina web in uso (mouse, tastiera, caricamento della pagina ecc...). Tali funzioni di script possono essere opportunamente inserite in file HTML, in pagine JSP o in appositi file separati con estensione .js poi richiamati nella logica di business.

3.7 JQuery

jQuery$_G$ è una libreria JavaScript per applicazioni web. Nasce con l'obiettivo di semplificare la selezione, la manipolazione, la gestione degli eventi e l'animazione di elementi DOM in pagine HTML, nonché implementare funzionalità AJAX.

3.8 Bootstrap 3

Bootstrap 3$_G$ è una raccolta di strumenti per la creazione di siti e applicazioni Web. Essa contiene modelli di progettazione basati su HTML e *CSS*$_G$, sia per la tipografia, che per le varie componenti dell'interfaccia, come moduli, così come alcune estensioni opzionali di JavaScript.

4 Configurazione software

4.1 Pre-requisiti

4.2 Installazione di Git

4.2.1 Windows 10

Scaricare l'eseguibile di msysGit dal sito: http://msysgit.github.io/ e procedere con l'installazione.

4.2.2 Mac OSX

Eseguire da terminale il seguente comando, dopo aver installato Homebrew (http:// brew.sh/):

```
$ brew install git
```

4.2.3 Ubuntu 16.04 LTS

Eseguire da terminale il seguente comando:

```
$ apt install git
```

4.2.4 Installazione node.js

Scaricare l'installer corretto in base al proprio sistema operativo, reperibile alla pagina ufficiale di download di Node.js (https://nodejs.org/it/download/). Procedere dunquecon l'installazione di Node.js.

4.2.5 Download dei file da GitHub

Eseguire da terminale il seguente comando:

```
$  git clone --recursive-submodules https://github.com/netbreakswe/APIM_Source
```

4.2.6 Installazione Java JDK

Scaricare l'eseguibile compatibile con il proprio terminale di JDK8 dal sito:

```
http://www.oracle.com/technetwork/java/javase/downloads/
```

e procedere con l'installazione.

4.2.7 Installazione Jolie

Prima di eseguire questo passaggio è importante aver eseguito quanto descritto nella sezione precedente.
Scaricare l'eseguibile di Jolie dal sito: http://www.jolie-lang.org/downloads.html.

4.2.7.1 Windows

Aprire un terminale con privilegi di amministratore nella cartella del download ed eseguire:

```
java -jar jolie-1.6.0.jar
```

4.2.7.2 Linux/MacOS

Aprire un terminale nella cartella del download ed eseguire:

```
sudo java -jar jolie-1.6.0.jar
```

4.3 Installazione Dipendenze WebApp

Aprire il terminale nella cartella con il seguente percorso: `*Download del punto 5.2.5*/Frontend` ed eseguire i seguenti comandi in sequenza:

```
npm install -g bower@latest
```

```
npm install -g http-server
```

```
npm install -g angular-seed@latest
```

```
npm install
```

4.4 Installazione piattaforma

In questa sezione è spiegato come avviare i microservizi neccessari per una corretta fruizione della web app e del gateway. Si suppone che la cartella di partenza sia la cartella scaricata al punto 5.2.5 del manuale corrente.

4.4.1 Microservizi relativi alla Web App

Di seguito saranno elencati i microservizi da avviare per una corretta funzione della web app. L'ordine di avvio non è rilevante e può avvenire in ordine casuale.

4.4.1.1 Microservizio per gestione utenti

Per avviare il microservizio relativo alla gestione utenti è necesario recarsi nella cartella:

```
\Backend\services\usersDB
```

ed eseguire da terminale il comando:

```
jolie user_db.ol
```

4.4.1.2 Microservizio per il Service Level Agreement

Per avviare il microservizio relativo al Service Level Agreement è necesario recarsi nella cartella:

```
\Backend\services\slaDB
```

ed eseguire da terminale il comando:

```
jolie sla_db.ol
```

4.4.1.3 Microservizio per la gestione di file

Per avviare il microservizio relativo alla gestione di file è necesario recarsi nella cartella:

```
\Backend\services\filehandlerDB
```

ed eseguire da terminale il comando:

```
jolie filehandler.ol
```

4.4.1.4 Microservizio per la gestione di microservizi

Per avviare il microservizio relativo alla gestione di microservizi è necesario recarsi nella cartella:

```
\Backend\services\microservicesDB
```

ed eseguire da terminale il comando:

```
jolie microservices_db.ol
```

4.4.1.5 Microservizio per la gestione delle transazioni

Per avviare il microservizio relativo alla gestione delle transazioni è necesario recarsi nella cartella:

```
\Backend\services\transactionsDB
```

ed eseguire da terminale il comando:

```
jolie transactions_db.ol
```

4.4.2 Microservizi relativi all'API Gateway

Di seguito saranno elencati i microservizi da avviare per una corretta funzione dell'API Gateway. L'ordine di avvio è rilevante e come prerequisito si richiede di aver avviato tutti i servizi elencati al punto 5.4.1.

4.4.2.1 Microservizio per gestione microservizi del gateway

Per avviare il microservizio relativo alla gestione delle transazioni è necesario recarsi nella cartella:

```
\Backend\services\microservicesDB
```

ed eseguire da terminale il comando:

```
jolie serviceinteractionhandler.ol
```

4.4.2.2 Avvio del gateway

Per avviare il microservizio relativo alla gestione delle transazioni è necesario recarsi nella cartella:

```
\Backend\gateway
```

ed eseguire da terminale il comando:

```
jolie gateway.ol
```

4.5 Configurazione database

Per la configurazione dei database, è necessario avere accessi amministrativi al database che si intende utilizzare. Tramite un tool grafico (ad esempio PhpMyAdmin) o riga di comando, è necessario creare 5 database corrispondenti a ciascun microservizio.
Successivamente, in ciascun database deve venir eseguita un importazione dei file .sql forniti in dotazione con il software. Tale procedimento crea le tabelle necessarie per la corretta funzionalità dei servizi.
Prima dell'avvio dell'applicazione, è necessario modificare i file interfaccia relativi ai servizi backend, affinchè essi puntino al database corrispondente. I file da modificare sono i seguenti:

`filehandler.ol` `microservices_db.ol` `sla_db.ol` `transactions_db.ol` `user_db.ol`

Per la corretta impostazione, è necessario cercare le voci relative ai dati di connessione:

`.host` `.driver` `.port` `.database` `.username` `.password`

In particolare, le diciture sopraelencate corrispondono ai seguenti valori:

- **.host**: L'URL del database, al quale il servizio effettuerà la connessione
- **.driver**: Il driver JDBC da utilizzare. In particolare, nel caso di installazione come da specifiche, dovrà essere utilizzato il driver `mysql` .
- **.port**: La porta di connessione al database. Di default tale valore è 3306 per i database MySQL.
- **.database**: Il nome del database corrispondente al servizio considerato. Si presti particolare attenzione ad associare il servizio con l'opportuno database che ospita le tabelle corrispondenti.
- **.username**: L'username con diritti di lettura e scrittura sul database
- **.password**: La password per l'accesso al database.

4.6 Avvio dell'applicazione

Aprire il terminale nella cartella con il seguente percorso:

```
\Frontend
```

ed eseguire il seguente comando:

```
npm start
```

5 Architettura generale

L'applicativo API Market è strutturato in un lato front-end ed un lato back-end, come mostrato dal diagramma seguente:

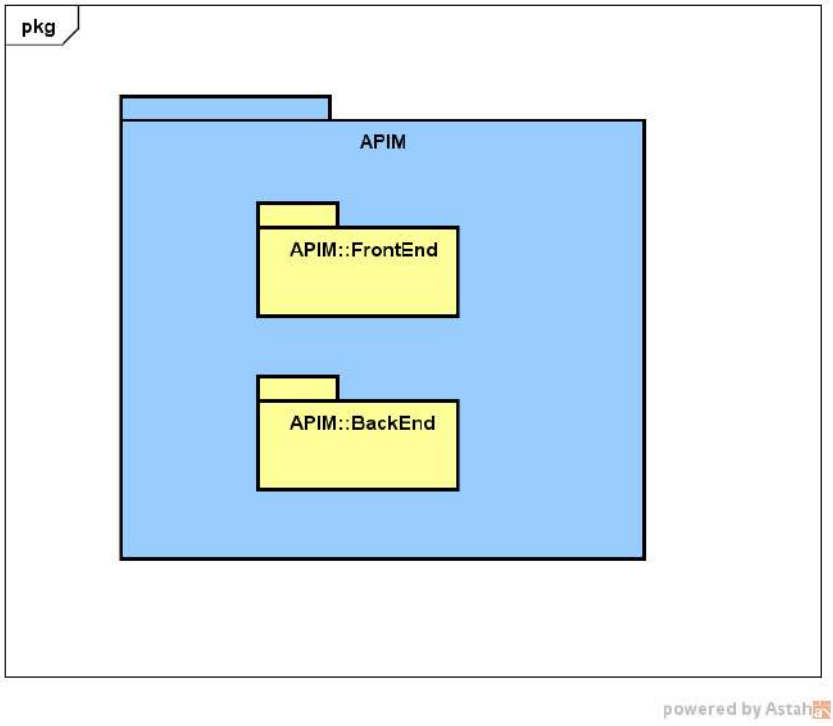

Figura 1: Package APIM

5.1 Front-end

Nella parte front-end sono presenti i package *node_ modules* e *App*, che derivano dall'architettura dei componenti e framework scelti (AngularJS).

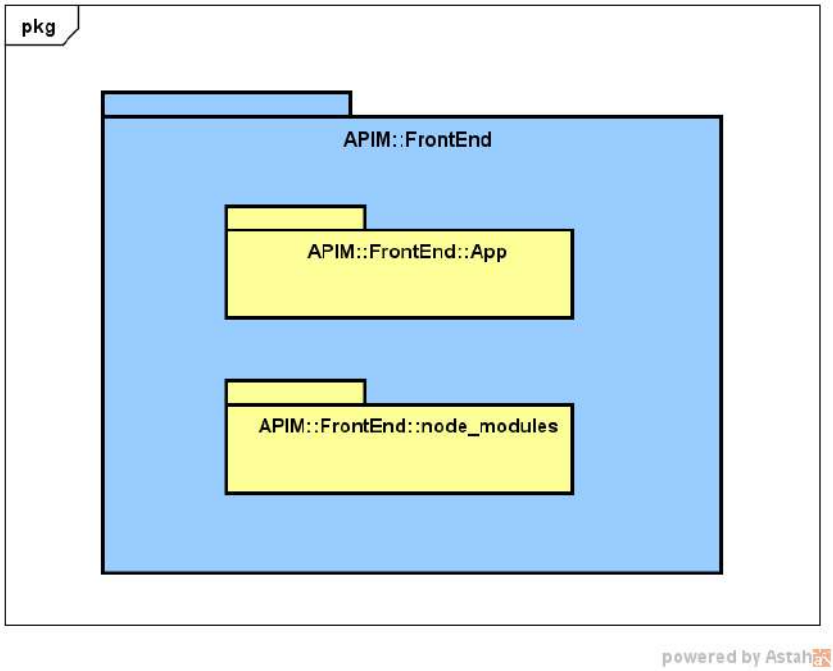

Figura 2: Package APIM::FrontEnd

- **Padre**: APIM;
- **Descrizione**: package contenente le componenti del front-end dell'applicazione;
- **Package contenuti**:
  - ***node_ modules***: package contenente i riferimenti alle librerie esterne di JavaScript;
  - ***App***: package contenente le componenti principali dell'applicazione.

5.2 Back-end

Nella parte back-end sono presenti i package *Gateway* e *Services* strutturati secondo un'architettura a microservizi, i quali hanno una dipendenza verso l'interfaccia *ServiceInteractionHandler*.

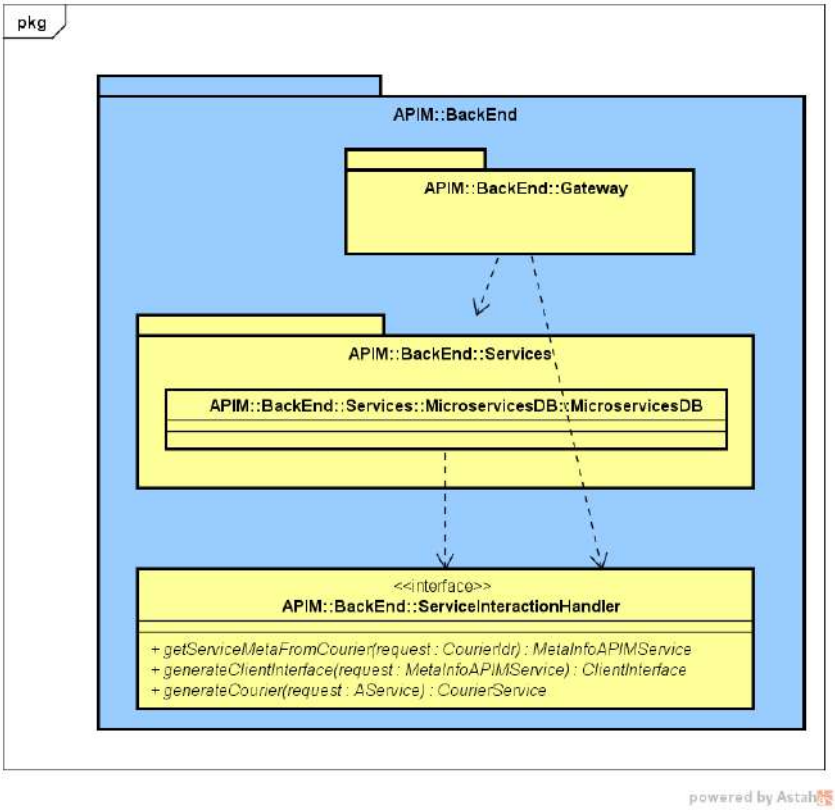

Figura 3: Package APIM::BackEnd

- **Padre**: APIM;
- **Descrizione**: package contenente le componenti del back-end dell'applicazione;
- **Package contenuti**:
  - ***Gateway***: package contenente le classi e le interfacce per il funzionamento dell'API Gateway;
  - ***Services***: package contenente tutti i microservizi per le comunicazioni con i database.
- **Classi contenute**:
  - ***ServiceInteractionHandler***: interfaccia per la gestione delle comunicazioni dell'API Gateway e dei *services*.

Nelle sezioni verranno presentati i packages in maniera dettagliata.

6 Packages Front-end

6.1 node_modules

- **Padre**: FrontEnd;
- **Descrizione**: package che raccoglie le librerie esterne di JavaScript, installate tramite *npm* di Node.js, e fornisce le funzionalità necessarie alla parte di front-end dell'applicazione;
- **Relazioni d'uso con altri componenti**: questo package si relaziona con i *controllers* presenti nel sub-package *Controllers* del package *App*.

6.2 App

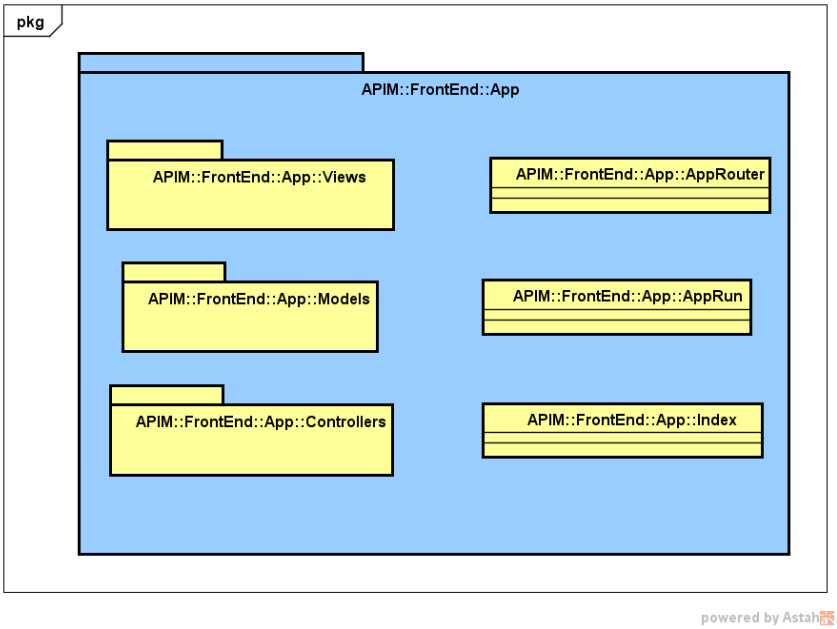

Figura 4: Package APIM::FrontEnd::App

- **Padre**: FrontEnd;
- **Descrizione**: package che raccoglie le componenti principali del front-end dell'applicazione web, secondo il pattern architetturale Model-View-Controller (MVC);
- **Relazioni d'uso con altri componenti**: questo package si relaziona con le librerie presenti nel sub-package *node_modules* del package padre;
- **Package contenuti**:
  - ***Views***: package contenente le *views* del front-end dell'applicazione;
  - ***Models***: package contenente i *models* del front-end dell'applicazione;
  - ***Controllers***: package contenente i *controllers* del front-end dell'applicazione.
- **Classi contenute**:
  - ***AppRun***: classe che istanzia l'applicazione;
  - ***AppRouter***: classe che gestisce i routes dell'applicazione;
  - ***Index***: view generale dell'applicazione (single page app).

6.2.1 AppRun

- **Padre**: App;
- **Descrizione**: classe che istanzia l'applicazione e che viene utilizzata per indicare le dipendenze tra l'applicazione e i packages esterni.

6.2.2 AppRouter

- **Padre**: App;
- **Descrizione**: classe che gestisce i routes dell'applicazione al fine di associare ad ogni route un controller e una view (associa un URL alle varie view dell'applicazione).

6.2.3 Index

- **Padre**: App;
- **Descrizione**: classe che rappresenta la view generale dell'applicazione e che contiene gli elementi che saranno presenti in ogni pagina dell'applicazione.

6.2.4 Views

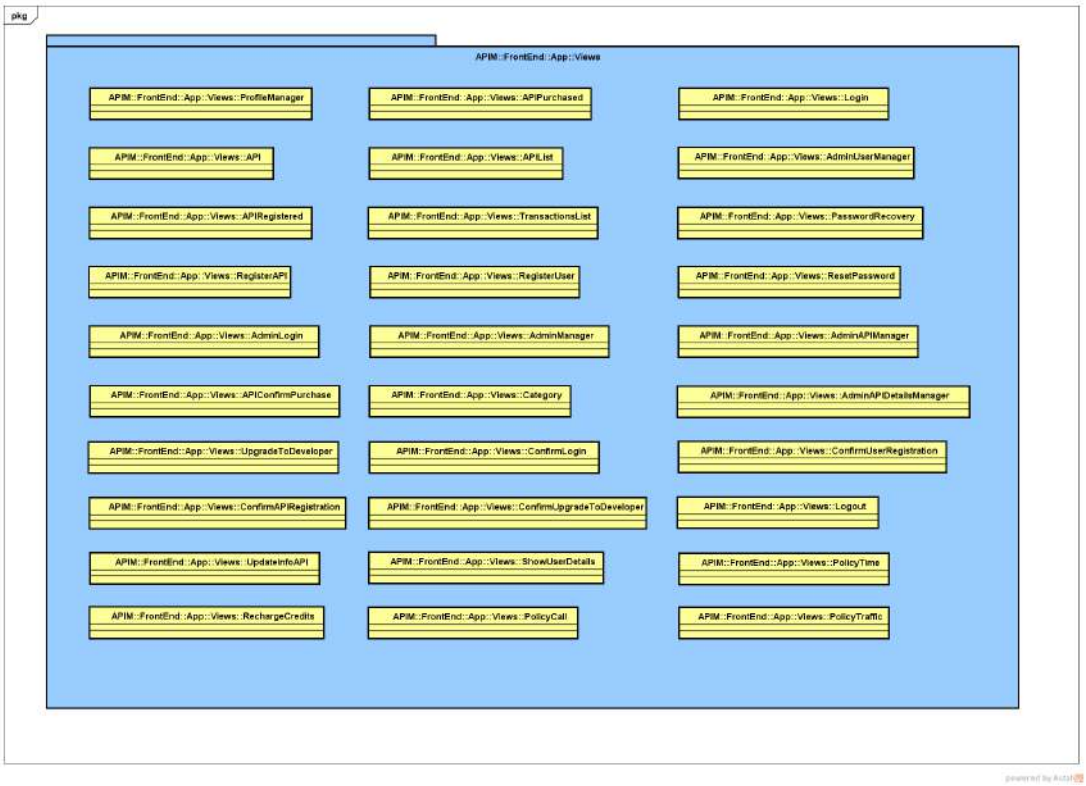

Figura 5: Package APIM::FrontEnd::App::Views

- **Padre**: App;
- **Descrizione**: package contenente tutte le classi che rappresentano i vari template HTML per le pagine web dell'applicazione;
- **Relazioni d'uso con altri componenti**: questo package si relaziona con i *controllers* presenti nel package *Controllers*;

- **Classi contenute**:
  - ***RegisterUser***: view contenente il form dedicato alla registrazione di un utente, il quale può inserire i campi necessari e registrarsi così alla piattaforma. Contiene, inoltre, un link alla pagina di login.
    Si relaziona con le seguenti componenti: *UserRegistrationController*.
  - ***Login***: view contenente il form necessario affinchè l'utente possa effettuare il login ed autenticarsi al sistema. Contiene, inoltre, un link alla pagina di registrazione e uno alla pagina per il recupero della password.
    Si relaziona con le seguenti componenti: *LoginController*.
  - ***PasswordRecovery***: view contenente il form dedicato al recupero della password di un utente, il quale può inserire l'indirizzo email e ricevere una nuova password con la quale autenticarsi al sistema. Contiene, inoltre, un link alla pagina di login.
    Si relaziona con le seguenti componenti: *PasswordRecoveryController*.
  - ***API***: view contenente i risultati della ricerca effettuata, che permette di selezionare un risultato presente al suo interno.
    Si relaziona con le seguenti componenti: *APIController*.
  - ***ProfileManager***: view contenente le informazioni del profilo personale di un utente registrato. Contiene, inoltre, l'informazione relativa al saldo del proprio conto virtuale.
    Si relaziona con le seguenti componenti: *ProfileManagerController*.
  - ***ResetPassword***: view contenente il form dedicato al cambio di password di un utente autenticato, il quale può inserire la nuova password che intende utilizzare per i futuri login al sistema.
    Si relaziona con le seguenti componenti: *ResetPasswordController*.
  - ***RegisterAPI***: view contenente il form per l'inserimento di una API da parte di un utente sviluppatore. Lo sviluppatore può inserire tutti i dati relativi al microservizio che intende esporre sul marketplace.
    Si relaziona con le seguenti componenti: *APIRegistrationController*.
  - ***PolicyCall***: view contenente le informazioni relative alla policy per chiamate.
    Si relaziona con le seguenti componenti: *PolicyCallController*.
  - ***PolicyTime***: view contenente le informazioni relative alla policy per tempo.
    Si relaziona con le seguenti componenti: *PolicyTimeController*.
  - ***PolicyTraffic***: view contenente le informazioni relative alla policy per traffico dati.
    Si relaziona con le seguenti componenti: *PolicyTrafficController*.
  - ***APIRegistered***: view contenente le informazioni di una API registrata sulla piattaforma.
    Si relaziona con le seguenti componenti: *APIRegisteredController*.
  - ***APIPurchased***: view contenente le informazioni delle API acquistate da un cliente della piattaforma API Market.
    Si relaziona con le seguenti componenti: *APIPurchasedController*.
  - ***APIList***: view contenente l'elenco delle API disponibili sul marketplace API Market.
    Si relaziona con le seguenti componenti: *APIListController*.
  - ***TransactionsList***: view contenente l'elenco delle transazioni effettuate da un utente sul marketplace API Market.
    Si relaziona con le seguenti componenti: *TransactionsListController*.
  - ***APIConfirmPurchase***: view contenente la conferma di un acquisto di una API.
    Si relaziona con le seguenti componenti: *APIConfirmPurchaseController*.

- ***AdminManager***: view contenente le operazioni per la gestione del profilo amministratore API Market.
  Si relaziona con le seguenti componenti: *AdminManagerController*.
- ***Category***: view contenente l'elenco delle categoria nell'API Market.
  Si relaziona con le seguenti componenti: *CategoryController*.
- ***ConfirmUpgradeToDeveloper***: view contenente la conferma dell'upgrade di un utente a sviluppatore nell'API Market.
  Si relaziona con le seguenti componenti: *ConfirmUpgradeToDeveloperController*.
- ***ConfirmLogin***: view contenente la conferma di login all'API Market.
  Si relaziona con le seguenti componenti: *ConfirmLoginController*.
- ***ConfirmUserRegistration***: view contenente la conferma di registrazione all'API Market.
  Si relaziona con le seguenti componenti: *ConfirmUserRegistrationController*.
- ***ConfirmAPIRegistration***: view contenente la conferma di registrazione di una API all'API Market.
  Si relaziona con le seguenti componenti: *ConfirmAPIRegistrationController*.
- ***UpgradeToDeveloper***: view contenente il modulo per diventare sviluppatore all'interno dell'API Market.
  Si relaziona con le seguenti componenti: *UpgradeToDeveloperController*.
- ***AdminAPIManager***: view contenente le operazioni dell'amministratore sulle API dell'API Market.
  Si relaziona con le seguenti componenti: *AdminAPIManagerController*.
- ***AdminAPIDetailsManager***: view contenente le statistiche di una API dell'API Market.
  Si relaziona con le seguenti componenti: *AdminAPIDetailsManagerController*.
- ***AdminUserManager***: view contenente le operazioni di un ammistratore sugli utenti dell'API Market.
  Si relaziona con le seguenti componenti: *AdminUserManagerController*.
- ***AdminLogin***: view contenente il form necessario affinchè l'amministratore possa effettuare il login ed autenticarsi al sistema.
  Si relaziona con le seguenti componenti: *AdminLoginController*.
- ***Logout***: view contenente la conferma di logout dall'API Market.
  Si relaziona con le seguenti componenti: *LogoutController*.
- ***UpdateInfoAPI***: view contenente il form dedicato modifica delle informazioni di una API presente nel marketplace API Market.
  Si relaziona con le seguenti componenti: *UpdateInfoAPIController*.
- ***RechargeCredits***: view contenente la possibilità di scegliere quanti crediti ricaricare sul conto personale tra i tagli disponibili.
  Si relaziona con le seguenti componenti: *RechargeCreditsController*.

  item ***ShowUserDetalis***: view contenente le informazioni personali di un utente visualizzabili da altri fruitori del marketplace API Market.
  Si relaziona con le seguenti componenti: *ShowUserDetalisController*.
- ***AdminModeration***: view contenente il form dedicato alla moderazione di un utente o di una API/microservizio da parte di un amministratore della piattaforma API Market.
  Si relaziona con le seguenti componenti: *AdminModerationController*.

6.2.5 Models

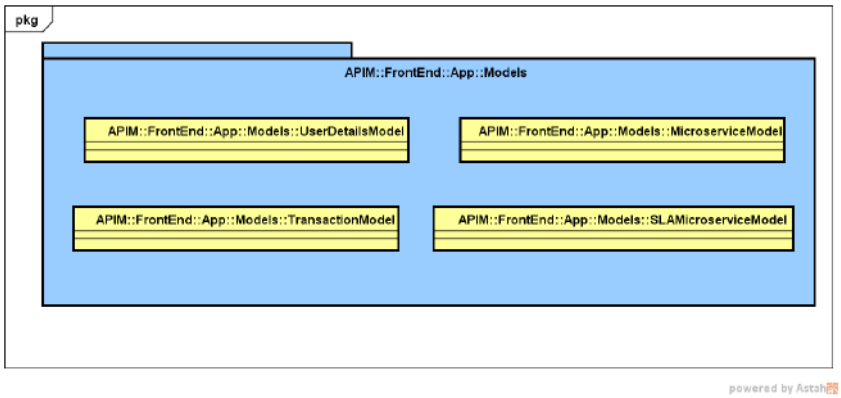

Figura 6: Package APIM::FrontEnd::App::Models

- **Padre**: App;
- **Descrizione**: package che contiene le classi che definiscono la business logic dell'applicazione;
- **Relazioni d'uso con altri componenti**: questo package si relaziona con il package *Controllers*;
- **Classi contenute**:
  - ***UserDetailsModel***: classe che rappresenta un utente e che contiene tutte le informazioni necessarie alla presentazione del contenuto di un utente, sia nella visualizzazione che nella gestione di un profilo.
    Si relaziona con le seguenti componenti: *LoginController*, *APIController*, *ProfileManagerController*, *ResetPasswordController*, *PasswordRecoveryController*.
  - ***MicroserviceModel***: classe che rappresenta un microservizio e che contiene tutte le informazioni necessarie alla presentazione del contenuto di un microservizio, sia nella visualizzazione che nella gestione.
    Si relaziona con le seguenti componenti: *APIRegiteredController*, *APIRegistrationController*, *PolicyCallController*, *PolicyTimeController*, *PolicyTrafficController*, *APIPurchasedController*, *APIListController*.
  - ***TransactionModel***: classe che rappresenta una transazione avvenuta e che contiene tutte le informazioni necessarie alla presentazione del contenuto di una transazione, sia nella visualizzazione che nella gestione.
    Si relaziona con le seguenti componenti: *TransactionsListController*, *APIPurchasedController*.
  - ***SLAMicroserviceModel***: classe che rappresenta la SLA di un microservizio e che contiene tutte le informazioni necessarie alla presentazione del contenuto di SLA di un microservizio, sia nella visualizzazione che nella gestione.
    Si relaziona con le seguenti componenti: *PolicyCallController*, *PolicyTimeController*, *PolicyTrafficController*, *APIRegistrationController*.

6.2.6 Controllers

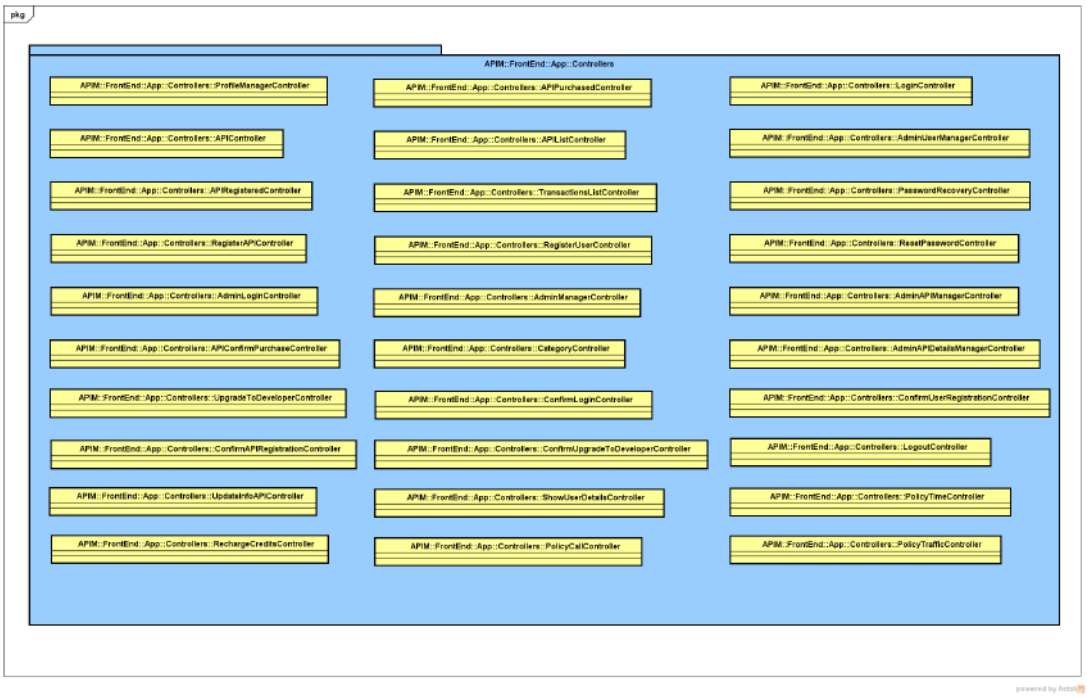

Figura 7: Package APIM::FrontEnd::App::Controllers

- **Padre**: App;
- **Descrizione**: package che contiene i *controllers* individuati per la parte front-end dell'applicazione, i quali consentono la gestione delle azioni utente dell'applicazione web;
- **Relazioni d'uso con altri componenti**: questo package si relaziona con i package *Views* e *Models*.
- **Classi contenute**:
  - ***RegisterUserController***: classe che permette di gestire la registrazione di un utente al sistema, fornendone le funzionalità preposte.
    Si relaziona con le seguenti componenti: *RegisterUser*.
  - ***LoginController***: classe che permette di gestire il login di un utente alla piattaforma API Market, fornendo le funzionalità di autenticazione al sistema, compresa la gestione di situazioni di errore di autenticazione.
    Si relaziona con le seguenti componenti: *UserDetailsModel*, *Login*.
  - ***PasswordRecoveryController***: classe che permette di gestire il ripristino della password dimenticata da un utente, fornendo tutte le funzionalità per il recupero della password dopo aver verificato l'identità dell'utente.
    Si relaziona con le seguenti componenti: *UserDetailsModel*, *PasswordRecovery*.
  - ***APIController***: classe che permette di gestire la ricerca di microservizi all'interno del marketplace API Market, fornendo all'utente le funzionalità di ricerca tramite categorie e keywords per sviluppatori e microservizi.
    Si relaziona con le seguenti componenti: *API*.
  - ***ProfileManagerController***: classe che permette di gestire il profilo personale di un utente, fornendo le funzionalità all'utente per poter modificare i propri dati.
    Si relaziona con le seguenti componenti: *UserDetailsModel*, *ProfileManager*.

- ***ResetPasswordController***: classe che permette di gestire il cambio password di un utente autenticato al sistema, fornendo le funzionalità per il salvataggio di una nuova password.
  Si relaziona con le seguenti componenti: *UserDetailsModel*, *ResetPassword*.
- ***RegisterAPIController***: classe che permette di gestire l'inserimento di una API, fornendo tutte le funzionalità atte alla corretta esposizione di un microservizio di uno sviluppatore, utente della piattaforma API Market.
  Si relaziona con le seguenti componenti: *MicroserviceModel*, *RegisterAPI*.
- ***PolicyCallController***: classe che permette di gestire le policy di vendita dei microservizi per chiamate.
  Si relaziona con le seguenti componenti: *PolicyCall*, *SLAMicroserviceModel*.
- ***PolicyTimeController***: classe che permette di gestire le policy di vendita dei microservizi per tempo.
  Si relaziona con le seguenti componenti: *PolicyTime*, *SLAMicroserviceModel*.
- ***PolicyTrafficController***: classe che permette di gestire le policy di vendita dei microservizi per traffico dati.
  Si relaziona con le seguenti componenti: *PolicyTraffic*, *SLAMicroserviceModel*.
- ***APIRegisteredController***: classe che permette di gestire le informazioni di una API precedentemente inserita.
  Si relaziona con le seguenti componenti: *MicroserviceModel*, *APIRegistered*.
- ***APIPurchasedController***: classe che permette di gestire l'acquisto e le relative informazioni di una API, fornendo l'API Key per l'utilizzo del cliente.
  Si relaziona con le seguenti componenti: *APIPurchased*, *MicroserviceModel*, *TransactionModel*.
- ***APIListController***: classe che permette di gestire l'elenco dei microservizi presenti sul marketplace API Market.
  Si relaziona con le seguenti componenti: *APIList*, *MicroserviceModel*.
- ***TransactionsListController***: classe che permette di gestire lo storico delle transazioni di un utente del marketplace API Market.
  Si relaziona con le seguenti componenti: *TransactionsList*, *TransactionsModel*.
- ***AdminManagerController***: classe che permette di gestire il profilo di un amministratore della piattaforma API Market, fornendo le funzionalità per poter modificare i propri dati e moderare utenti ed API.
  Si relaziona con le seguenti componenti: *AdminManager*.
- ***CategoryController***: classe che permette di gestire l'elenco delle categoria nell'API Market.
  Si relaziona con le seguenti componenti: *Category*.
- ***ConfirmUpgradeToDeveloperController***: classe che permette di gestire la conferma dell'upgrade di un utente a sviluppatore nell'API Market.
  Si relaziona con le seguenti componenti: *ConfirmUpgradeToDeveloper*.
- ***ConfirmLoginController***: classe che permette di gestire la conferma di login all'API Market.
  Si relaziona con le seguenti componenti: *ConfirmLogin*.
- ***ConfirmUserRegistrationController***: classe che permette di gestire la conferma di registrazione all'API Market.
  Si relaziona con le seguenti componenti: *ConfirmUserRegistration*.
- ***ConfirmAPIRegistrationController***: classe che permette di gestire la conferma di registrazione di una API all'API Market.
  Si relaziona con le seguenti componenti: *ConfirmAPIRegistration*.

- ***UpgradeToDeveloperController***: classe che permette di gestire il modulo per diventare sviluppatore all'interno dell'API Market.
  Si relaziona con le seguenti componenti: *UpgradeToDeveloper*.
- ***AdminAPIManagerController***: classe che permette di gestire le operazioni dell'amministratore sulle API dell'API Market.
  Si relaziona con le seguenti componenti: *AdminAPIManager*.
- ***AdminAPIDetailsManagerController***: classe che permette di gestire le statistiche di una API dell'API Market.
  Si relaziona con le seguenti componenti: *AdminAPIDetailsManager*.
- ***AdminUserManagerController***: classe che permette di gestire le operazioni di un ammistratore sugli utenti dell'API Market.
  Si relaziona con le seguenti componenti: *AdminUserManager*.
- ***AdminLoginController***: classe che permette di gestire il form necessario affinchè l'amministratore possa effettuare il login ed autenticarsi al sistema.
  Si relaziona con le seguenti componenti: *AdminLogin*.
- ***LogoutController***: classe che permette di gestire la conferma di logout dall'API Market.
  Si relaziona con le seguenti componenti: *Logout*.
- ***UpdateInfoAPIController***: classe che permette di gestire il form dedicato modifica delle informazioni di una API presente nel marketplace API Market.
  Si relaziona con le seguenti componenti: *UpdateInfoAPI*.
- ***RechargeCreditsController***: classe che permette di gestire la possibilità di scegliere quanti crediti ricaricare sul conto personale tra i tagli disponibili.
  Si relaziona con le seguenti componenti: *RechargeCredits*.
- ***ShowUserDetalisController***: classe che permette di gestire le informazioni personali di un utente visualizzabili da altri fruitori del marketplace API Market.
  Si relaziona con le seguenti componenti: *ShowUserDetalis*
- ***AdminModerationController***: classe che permette di gestire la moderazione di un utente (cliente o sviluppatore che sia) e di API, fornendo le funzionalità per la sospensione e rimozione.
  Si relaziona con le seguenti componenti: *AdminModeration*.

7 Packages Back-end

7.1 Gateway

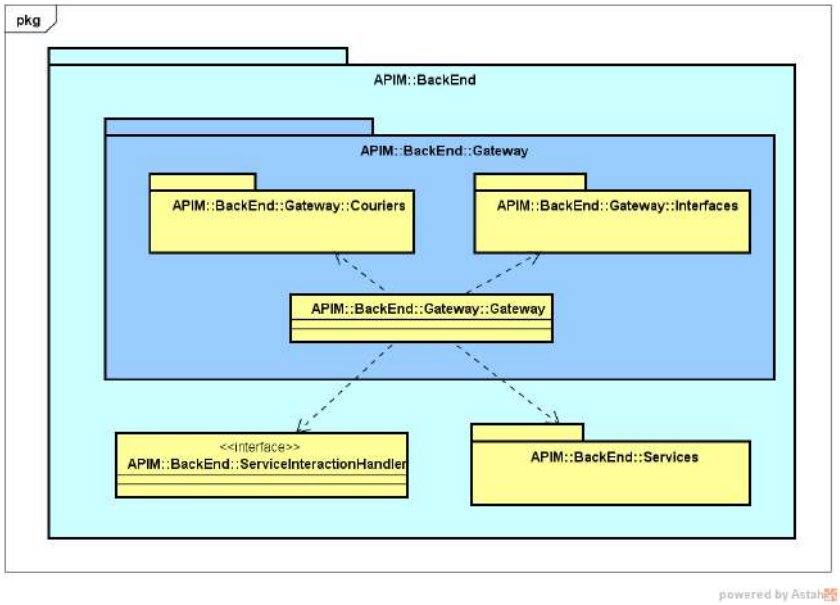

Figura 8: Package APIM::BackEnd::Gateway

- **Padre**: BackEnd;
- **Descrizione**: package contenente la componente principale del lato back-end. Pur essendo integrato nella sistema, è un modulo che svolge funzioni separate alla piattaforma vera e propria: infatti, l'API Gateway si occupa di controllare e reindirizzare le chiamate effettuate dai clienti ai microservizi (prodotti) inseriti nel marketplace API Market, verificandone la validità (credenziali, chiave) e monitorandone l'uso (SLA, utilizzi rimanenti).
- **Relazioni d'uso con altri componenti**: la classe *Gateway* comunica con le classi contenute all'interno del package *Services*. Inoltre, necessita delle interfacce contenute nel package *Interfaces* e crea le sessioni *couriers* Jolie, archiviandole nel package *Couriers*.
- **Package contenuti**:
  - ***Couriers***: package contenente l'archivio delle sessioni couriers dei microservizi Jolie. Le sessioni couriers vengono create dal gateway ed utilizzate da *ServiceInteractionHandler*. Inoltre, forniscono i dati necessari al funzionamento del package *Interfaces*. Le sessioni *couriers* permettono l'overloading dei messaggi inviati nelle chiamate dei microservizi al gateway, così da allegare alla richiesta le informazioni per raggiungere il microservizio target.
  - ***Interfaces***: package contenente le interfacce necessarie al funzionamento dell'API Gateway. Si occupa dei dati riguardanti le chiamate ai microservizi, in particolar modo, il tipo di operazione richiesta e le informazioni riguardanti l'interfaccia del microservizio target.
- **Classi contenute**:
  - ***Gateway***: classe che rappresenta la struttura dell'API Gateway della piattaforma API Market.

7.2 Services

Tale package contiene tutti i servizi appartenenti al lato back-end della piattaforma, eccezion fatta per il gateway già descritto nella sezione soprastante. *Services* contiene tutte le classi di comunicazione con i database che permetteranno il corretto funzionamento dell'applicazione.

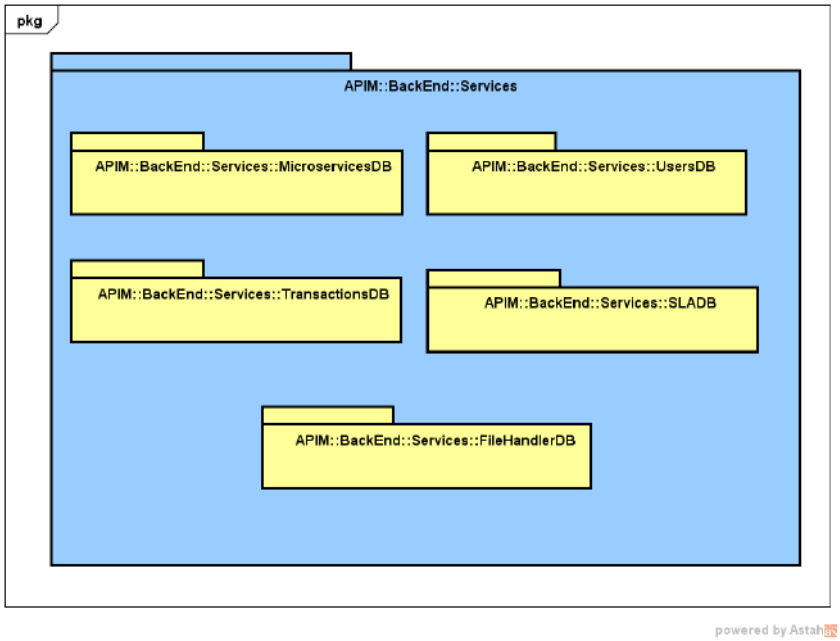

Figura 9: Package APIM::BackEnd::Services

- **Padre**: BackEnd;
- **Descrizione**: package che contiene tutti i package relativi ai microservizi del back-end dell'applicazione;
- **Relazioni d'uso con altri componenti**: questo package si relaziona con l'API Gateway;
- **Package contenuti**:
  - ***UsersDB***: package contenente le classi che permettono la comunicazione con il database relativo agli utenti del sistema;
  - ***MicroservicesDB***: package contenente le classi che permettono la comunicazione con il database relativo ai microservizi registrati sulla piattaforma;
  - ***TransactionsDB***: package contenente le classi che permettono la comunicazione con il database relativo alle transazioni degli utenti del sistema;
  - ***SLADB***: package contenente le classi che permettono la comunicazione con il database relativo alle informazioni di SLA dei microservizi utilizzati tramite l'API Gateway della piattaforma API Market;
  - ***FileHandlerDB***: package contenente le classi che permettono la gestione dei file.

7.2.1 UsersDB

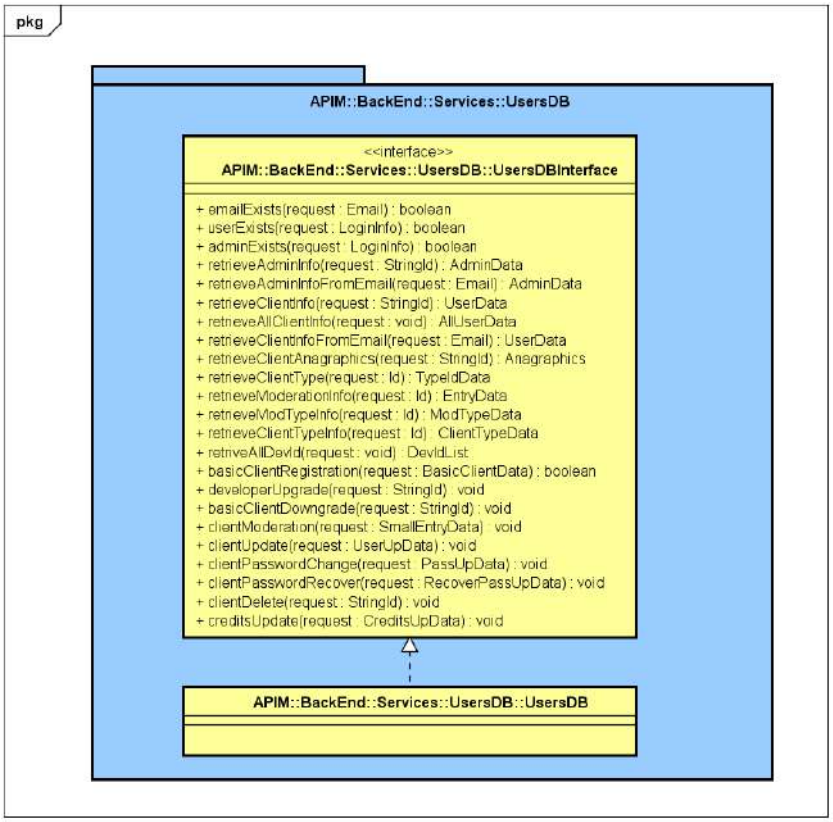

Figura 10: Package APIM::BackEnd::Services::UsersDB

- **Padre**: Services;
- **Descrizione**: package che contiene le classi di comunicazione con il database dell'applicazione relativo alle informazioni (anagrafiche e personali) degli utenti (admin, clienti, sviluppatori) del sistema;
- **Relazioni d'uso con altri componenti**: comunica con il *Gateway* per permettere l'identificazione, l'autenticazione e la gestione degli utenti, clienti e sviluppatori della piattaforma API Market;
- **Classi contenute**:
  - ***UsersDBInterface***: interfaccia che raccoglie tutte le *operation* riguardanti il database degli utenti;
  - ***UsersDB***: classe che implementa l'interfaccia *UsersDBInterface*.

7.2.2 MicroservicesDB

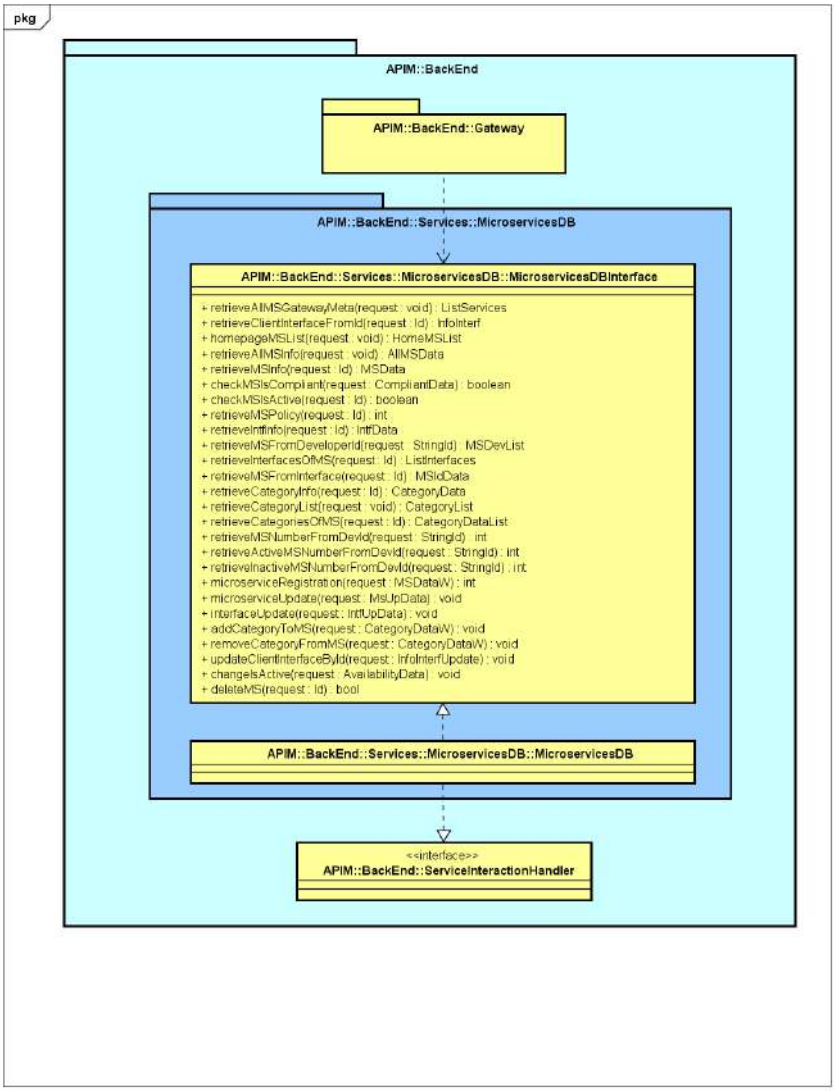

Figura 11: Package APIM::BackEnd::Services::MicroservicesDB

- **Padre**: Services;
- **Descrizione**: package che contiene le classi di comunicazione con il database dell'applicazione relativo ai microservizi registrati nella piattaforma;
- **Relazioni d'uso con altri componenti**: comunica con il *Gateway* per permettere l'identificazione e l'utilizzo dei microservizi, e garantire correttezza, efficienza e validità delle chiamate;
- **Classi contenute**:
  - ***MicroservicesDBInterface***: interfaccia che raccoglie tutte le *operation* riguardanti il database dei microservizi;
  - ***MicroservicesDB***: classe che implementa le interfacce *MicroservicesDBInterface* e *ServiceInteractionHandler*.

7.2.3 TransactionsDB

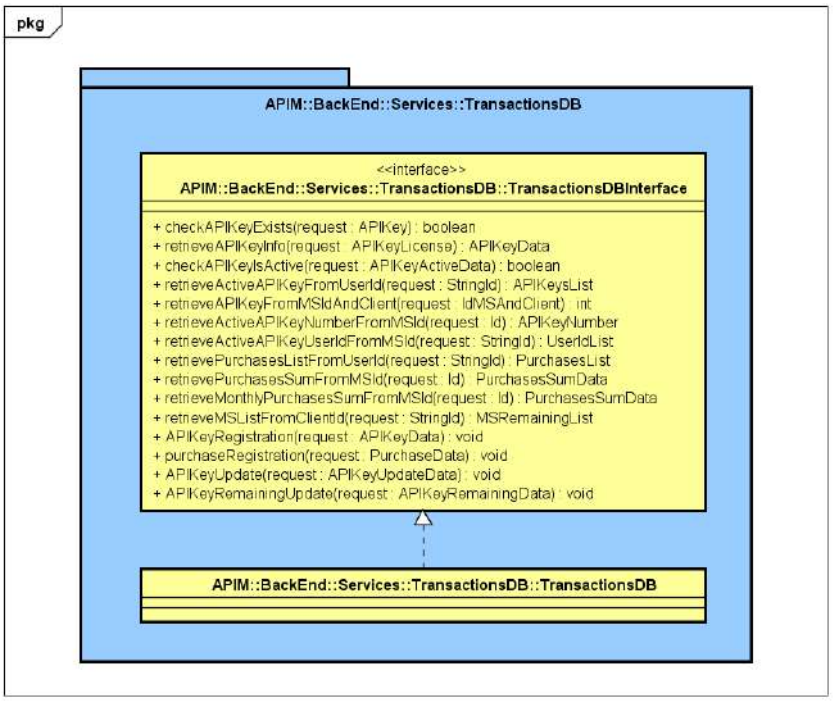

Figura 12: Package APIM::BackEnd::Services::TransactionsDB

- **Padre**: Services;
- **Descrizione**: package che contiene le classi di comunicazione con il database dell'applicazione che si occupa di immagazzinare i dati sulle transazioni;
- **Relazioni d'uso con altri componenti**: comunica con il *Gateway* per verificare la validità delle API Key e/o aggiornare i dati relativi (usi residui della chiave) alle chiamate ai microservizi effettuate;
- **Classi contenute**:
  - ***TransactionsDBInterface***: interfaccia che raccoglie tutte le *operation* riguardanti il database delle transazioni;
  - ***TransactionsDB***: classe che implementa l'interfaccia *TransactionsDBInterface*.

7.2.4 SLADB

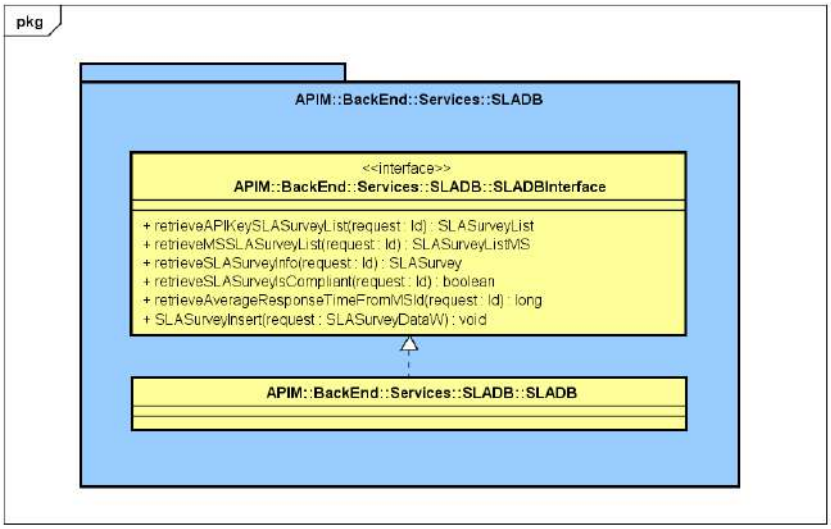

Figura 13: Package APIM::BackEnd::Services::SLADB

- **Padre**: Services;
- **Descrizione**: package che contiene le classi di comunicazione con il database che si occupa di immagazzinare e trattare i dati relativi al Service Level Agreement (SLA);
- **Relazioni d'uso con altri componenti**: comunica con il *Gateway* per aggiornare i dati delle performance di risposta di ogni microservizio utilizzato;
- **Classi contenute**:
  - ***SLADBInterface***: interfaccia che raccoglie tutte le *operation* riguardanti il database delle informazioni di SLA;
  - ***SLADB***: classe che implementa l'interfaccia *SLADBInterface*.

7.2.5 FileHandlerDB

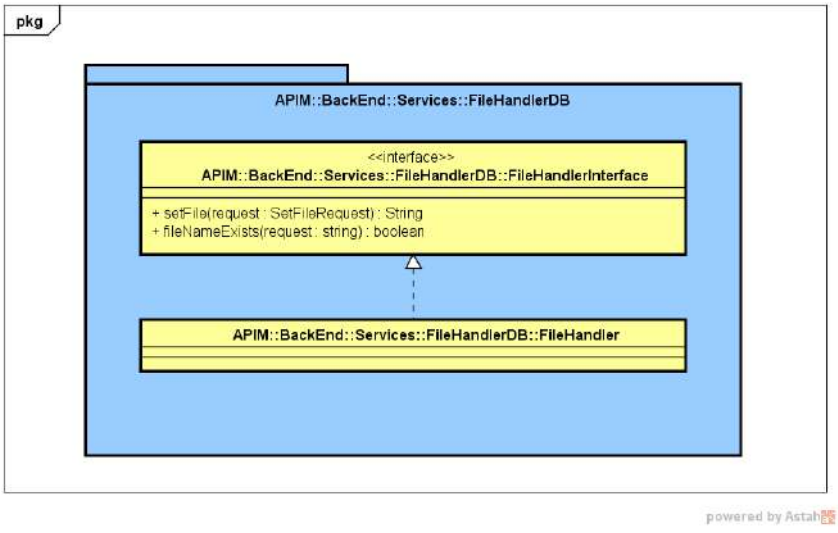

Figura 14: Package APIM::BackEnd::Services::FileHandlerDB

- **Padre**: Services;
- **Descrizione**: package che si occupa della gestione dei file, e presenta le classi di comunicazione con il database per tale finalità;
- **Relazioni d'uso con altri componenti**: comunica con il *Gateway* per recuperare i file legati alle interfacce dei microservizi;
- **Classi contenute**:
  - ***FileHandlerInterface***: interfaccia che raccoglie tutte le *operation* riguardanti la gestione dei file;
  - ***FileHandler***: classe che implementa l'interfaccia *FileHandlerInterface*.

8 Implementazione funzionalità

Per ragioni di tempo e di competenze, alcune funzionalità del software API Market non sono state implementate. In questa sezione vengono elencati tutti i punti di possibile estensione delle funzionalità e un loro possibile sviluppo.

8.1 Web App

8.1.1 Autenticazione tramite social network

La Web App è stata sviluppata per essere utilizzata un numero elevato di persone e permettere il login tramite siti esterni come Facebook e Twitter snelirebbe la procedura di registrazione. Facebook è attualmente il più grande social network del mondo e rende disponibile il login tramite un SDK proprietario basato su Javascript. Facebook Login con l'SDK di Facebook per JavaScript consente agli utenti di accedere al sito web con le loro credenziali di Facebook.

8.2 Rating e recensioni delle API

Una funzionalità molto diffusa nei market e in altre tipologlie di siti che offrono prodotti e servizi è la possibilità di recensire un prodotto, luogo o servizio. L'inserimento di una piattaforma di valutazione all'interno di API Market aiuterebbe il potenziale cliente a valutare la bontà dell'API e capire punti di forza o debolezza che l'autore non ha elencato. Solitamente ad ogni recensione è assegnata una valutazione sintetica da uno a cinque, dove uno è pessimo e cinque è ottimo, che permetterebbe di ordinare le API in base alle recensioni lasciate dagli acquirenti.

8.2.1 Q & A

L'introduzione di una sezione di domande e risposte è un modo per dare l'opportunità a potenziali di clienti di avere una risposta ad una specifica domanda da parte dell'autore, riguardante l'API pubblicata. Se questo sistema venisse implementato ridurrebbe l'insoddisfazione da parte del cliente nel caso l'API non svolgesse l'azione che si era prefissato.

8.2.2 Aggiunta tipologie account

Si è scelto di realizzare tre tipologie di account, quelle fondamentali per la corretta fruizione dell'API Market, ma è possibile aggiungerne altre, ad esempio potrebbe essere interessante l'aggiunta di tipologie agevolate nell'acquisto e/o fruizione delle API, come studenti e professori associati a scuole e università tramite l'inserimento di un codice personale.

8.2.3 Aggiunta modalità di pagamento

Attualmente l'unica modalità di acquisto è tramite crediti che è possibile acquistare tramite la piattaforma Paypal. Un Market che dispone di solo un metodo di pagamento può essere limitativo e quindi l'aggiunta di altri metodi di pagamento può ampliare il bacino di utenti.

8.3 Gateway

8.3.1 Service Level Agreement

Il Service Level Agreement è un insieme di caratteristiche che un API deve rispettare. Al momento del rilascio, si è scelto di verificare se l'API rispetta il tempo massimo di risposta indicato al momento dell'acquisto. Questa scelta è stata fatta per non sovracaricare il gateway ma in futuro in una soluzione con gateway distribuito, sarà possibile accumulare più dati per creare una profilazione maggiore riguardante una API. Tra le caratteristiche che si posso aggiungere

ci sono la dimensione dei dati inviata e ricevuta, la differenza di dimensione tra i dati inviati e ricevuti, la velocità di trasmissione.

8.3.2 Gateway distribuito

Durante la progettazione si è scelto di realizzare un gateway centralizzato, attraverso il quale passano tutte le richieste alle API. Si è tenuto conto che in futuro il gateway potrebbe essere distribuito, ovvero più gateway distribuiti. Questa soluzione distribuirebbe le richieste evitando di sovracaricare un unico gateway con troppe richieste e di conseguenza un rallentamento generale dell'infrastruttura. Un gateway distribuito permetterebbe di ampliare i dati ottenuti dal Service Level Agreement(SLA), limitati a solo il tempo di risposta in un gateway centralizzato.

A Glossario

A.1 A

- **AJAX:** In informatica AJAX, acronimo di Asynchronous JavaScript and XML, è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive. Lo sviluppo di applicazioni HTML con AJAX si basa su uno scambio di dati in background fra web browser e server, che consente l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente. AJAX è asincrono nel senso che i dati extra sono richiesti al server e caricati in background senza interferire con il comportamento della pagina esistente. Normalmente le funzioni richiamate sono scritte con il linguaggio JavaScript. Tuttavia, e a dispetto del nome, l'uso di JavaScript e di XML non è obbligatorio, come non è detto che le richieste di caricamento debbano essere necessariamente asincrone.
- **AngularJS:** è un framework web open source principalmente sviluppato da Google e dalla comunità di sviluppatori individuali che ruotano intorno al framework nato per affrontare le molte difficoltà incontrate nello sviluppo di applicazioni singola pagina.
- **API Gateway:** strumento che filtra e reindirizza le richieste utente per le varie API, fornendo il servizio, anche se questo non è presente sul server del marketplace.
- **API Key:** codice che identifica univocamente una specifica API e funge da token segreto che ne regola l'accesso e l'utilizzo.
- **API Market:** piattaforma che permette il commercio di API.

A.2 B

- **Bootstrap 3**: framework gratuito per il front-end web, open source, per la progettazione di siti e applicazioni web. Contiene template CSS per la maggior parte delle componenti grafiche di un'interfaccia utente ed estensioni JavaScript. A differenza di molti altri framework web, Bootstrap si occupa solo della parte front-end.

- **Browser**: il web browser, o più semplicemente browser, è un'applicazione per il recupero, la presentazione e la navigazione di risorse web. Tali risorse, come pagine web, immagini o video, sono messe a disposizione sulla World Wide Web, la rete globale che si appoggia su Internet, o su una rete locale, o sullo stesso computer dove il browser è in esecuzione. Il programma implementa da un lato le funzionalità di client per il protocollo HTTP, che regola lo scaricamento delle risorse dai server web a partire dal loro indirizzo URL e dall'altro quelle di visualizzazione dei contenuti ipertestuali, solitamente all'interno di documenti HTML, e di riproduzioni multimediali.

A.3 E

- **Event-driven**: programmazione guidata dagli eventi o semplicemente, programmazione a eventi.

A.4 H

- **HTML**: acronimo per HyperText Markup Language (traduzione letterale: linguaggio a marcatori per ipertesti), è il linguaggio di markup solitamente usato per la formattazione e impaginazione di documenti ipertestuali disponibili nel World Wide Web sotto forma di pagine web. È un linguaggio di pubblico dominio, la cui sintassi è stabilita dal World Wide Web Consortium (W3C).

A.5 J

- **Java**: è un linguaggio di programmazione orientato agli oggetti a tipizzazione statica, specificatamente progettato per essere il più possibile indipendente dalla piattaforma di esecuzione.
- **JavaScript**: linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione web lato client per la creazione di effetti dinamici interattivi, tramite funzioni di script invocate da eventi innescati a loro volta in vari modi dall'utente sulla pagina web in uso (mouse, tastiera, caricamento della pagina, etc.).
- **Jolie**: acronimo per Java Orchestration Language Interpreter Engine, è un linguaggio di programmazione open source per lo sviluppo di applicazioni distribuite basate su microservizi. Nel paradigma a microservizi proposto da Jolie, ogni programma è un servizio che può comunicare con altri programmi tramite lo scambio di messaggi attraverso la rete. Jolie sfrutta un interprete implementato in linguaggio Java ed è, inoltre, supportato da più sistemi operativi, quali Linux, OS X e Windows.
- **jQuery**: libreria JavaScript per applicazioni web che semplifica la selezione, la manipolazione, la gestione degli eventi e l'animazione di elementi DOM in pagine HTML, ed implementa funzionalità AJAX. È un framework gratuito, distribuito sotto i termini della licenza MIT.

A.6 W

- **Web app**: abbreviazione per applicazione web.