SWEG GROUP

Manuale Sviluppatore

Versione 1.0.0
Data di Rilascio 08/05/2017
Redazione Alberto Gelmi
Gianluca Crivellaro
Validazione Sebastiano Marchesini
Responsabile Alberto Gelmi
Uso Esterno
Destinato ItalianaSoftware S.r.l
Prof. Vardanega Tullio
Prof. Cardin Riccardo

Sommario
Manuale sviluppatore per applicazione APIM. Applicazione per vendere e consultare microservizi.

Registro Modifiche

Ver.	Modifica	Nome	Data
0.5.2	Aggiunta Parte del front end	Gianluca Crivellato	20/06/2017
0.5.1	Aggiunta Prima Pagina e Glossario Tutte le Figure Capitolo 3.1.1	Sebastiano Marchesini	20/06/2017
0.5.0	Verifica del documento Correzioni ortografiche e di layout del documento	Sebastiano Marchesini	20/06/2017
0.0.1	Creato documento Redatti cap.1, 2, 3, 4, 5	Alberto Gelmi	20/06/2017

Indice

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Questo documento offre le informazioni necessarie per modificare, ampliare ed estendere l'applicazione "API Market" allo scopo di aggiungere o adattare le sue funzionalità per renderle ulteriormente piacevoli all'utente.

1.2 Scopo del prodotto

API Market è una web app che permette all'utente di comprare o vendere dei microservizi, attraverso delle interfacce scritte in linguaggio Jolie. L'utente dunque potrà caricare sul portale web delle interfacce per i propri microservizi o acquistare delle chiavi per utilizzare quelli offerti da altri sviluppatori.

1.3 Argomenti trattati dal manuale sviluppatore

Questo manuale è diviso in sezioni che aiutano lo sviluppatore a comprendere in modo approfondito il prodotto software in ogni suo aspetto dal punto di vista architetturale e che forniscono il modo per estendere l'applicazione API Market.

1. **Ambiente di sviluppo**: in questa sezione sono fornite delle informazioni riguardanti l'ambiente di sviluppo (*Angular-Cli*$_g$) e la sua configurazione per avere accesso al codice di API Market.
2. **Architettura Software**: in questa sezione sono fornite delle informazioni sull'architettura dettagliata di API Market con particolare riguardo per le sue parti estensibili.
3. **Angular**: in questa sezione sono fornite delle informazioni sul il *framework*$_g$ Angular e il relativo linguaggio *TypeScript*$_g$ utilizzato per sviluppare il front end dell'applicazione.
4. **Jolie**: in questa sezione sono fornite delle informazioni riguardanti il linguaggio Jolie, utilizzato per sviluppare il Back end.
5. **Estendere API Market**: parte principale del documento dove vengono illustrati i più importanti punti di estensione.
6. **Test per le estensioni realizzate**: suggerimenti sulle verifiche da effettuare una volta implementato un nuovo prototipo di funzionalità.
7. **Note**

1.4 Glossario

Al fine di evitare ambiguità e ottimizzare la comprensione dei documenti, viene incluso un Glossario, nel quale saranno inseriti i termini tecnici, acronimi e parole che necessitano di essere chiarite.
Un glossario è una raccolta di termini di un ambito specifico e circoscritto. In questo caso per raccogliere termini desueti e specialistici inerenti al progetto.

2 Ambiente di sviluppo

2.1 Angular-cli

il front end è scritto usando l'ambiente di sviluppo *Angular-cli*, basta aprire *Angular-cli* dentro alla cartella del progetto per farlo girare.
Tramite *Angular-cli* è possibile estenderlo e modificarlo in maniera automatica.

2.1.1 Installazione

installabile attraverso npm con i seguenti comandi:

2.1.2 Compilazione

posizionarsi nella cartella del progetto attraverso la *shell*$_g$ e dare il comando `npm serve`

2.2 Repository

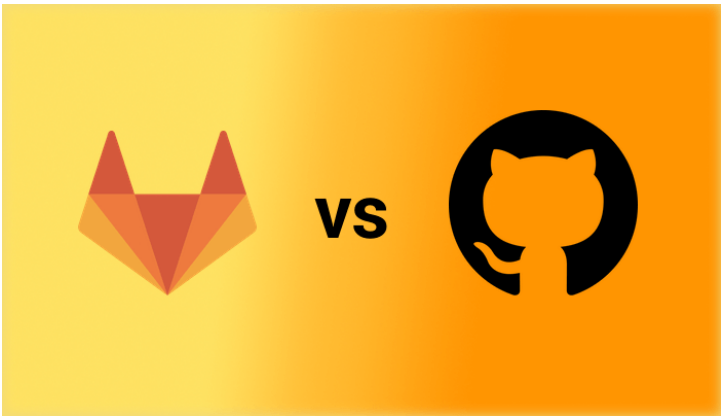

Figura 1: GitLab e GitHub

Per ottenere i *files* che costituiscono il progetto dal *repository*$_g$ presente su *GitLab*$_g$, dal progetto originale o un fork di esso, è necessario seguire il procedimento descritto nel paragrafo successivo.
Lo strumento *GitLab* rende private le repository, questo è stato scelto per non essere oggetto di spionaggio da parte di altri gruppi di progetto (in particolare per quanto riguarda la documentazione).
Per questo è stato reso pubblico uno spazio *GitHub*, apposito per la consultazione da parte degli enti interessati all link:

Repository di API Market: `https://github.com/SwegGroup/apimarket.git`

Per poter seguire i passi successivi, atti a configurare *APIMarket*$_g$ sul server, è necessario eseguire una *fork*$_g$ di una delle *repository*$_g$ ufficiale. Si consiglia di contattare il team *SWEg Group* tramite mail, per essere abilitati al lavoro su sistema *GitLab*. In quanto tale repository è maggiormente aggiornata.

2.3 Files del progetto

Di seguito sono riportati i percorsi relativi al progetto API Market che contengono il codice sorgente, i files *XML*$_g$ che riguardano l'interfaccia web e quelli che contengono i test di unità.

2.3.1 API Market: back-end

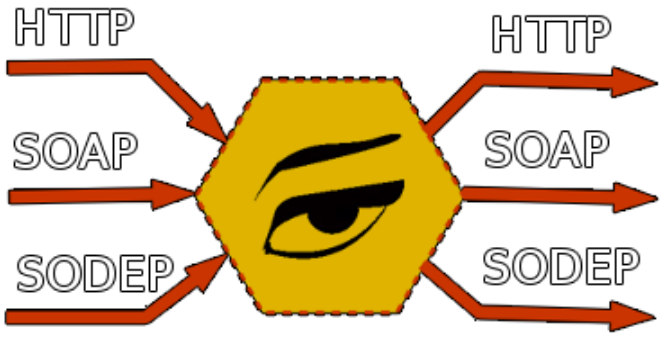

Figura 2: Jolie

Codice sorgente

1. **Microservizi Jolie:** `APIM/progetto/backend/Jolie`;
2. **Codice Java:** `APIM/progetto/backend/ApiM/src`;
3. **Microservizi Jolie:** `APIM/progetto/backend/Jolie`;

Figura 3: Test

Test

1. **Test di unità:** `APIM/progetto/backend/Jolie/UnitTest`.

3 Architettura Software

3.1 Architettura Generale

3.1.1 Front End

Dato che il prodotto finale è un'applicazione web single page, abbiamo deciso di implementare un design pattern che è più naturale programmare in questo tipo di situazioni, ossia il *Model-View-Controller (MVC)* in modo da agevolare il disaccoppiamento tra parte grafica e logica di *business*, facilitando quindi la creazione di test di unità.

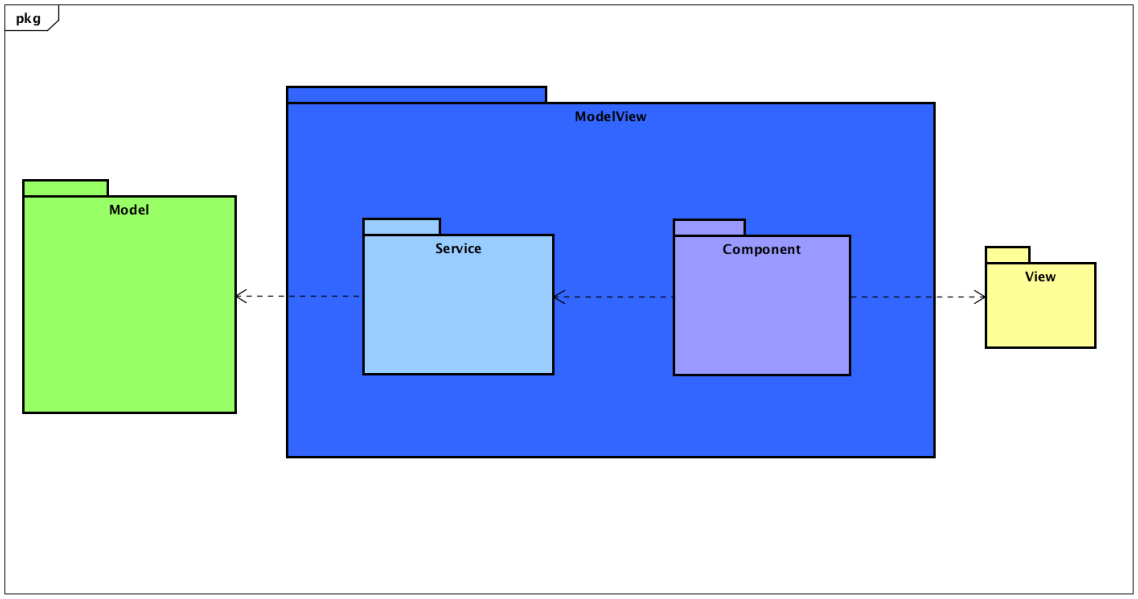

Figura 4: Front End Generale

L'intero progetto si poggia su dei tipo principali che grazie al *TypeScript* è possibile richiamare all'interno delle nostre componenti di Front-End.
Tali tipi sono richiamati anche nella base di dati e sono codificati nell'apposito *package*$_{\text{g}}$ `Model`.

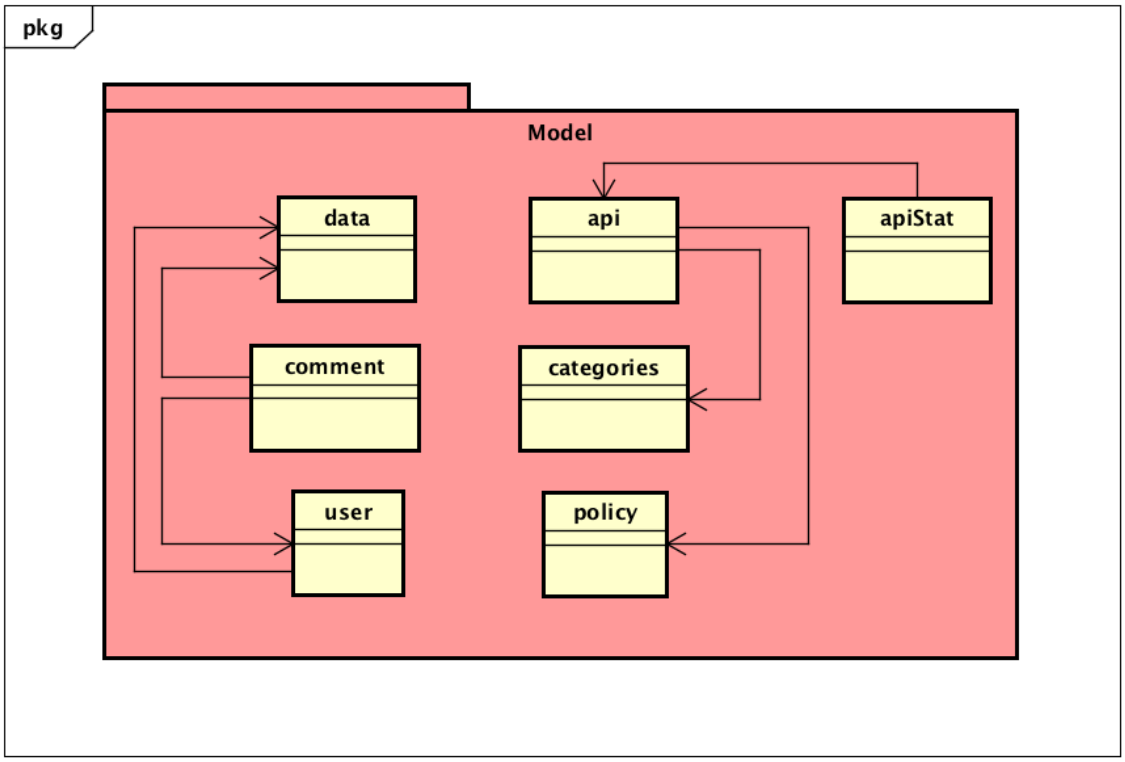

Figura 5: Modelli

- **Data**: rappresenta il formato data all'interno del Front-End;
- **Comment**: rappresenta i commenti che possono scrivere gli utenti;
- **User**: rappresenta il vero e proprio utente e tutti i suoi attributi;
- **Api**: sono i microservizi;
- **Categories**: sono le categorie che fanno riferimento i microservizi;
- **Policy**: di vendita suddivise a seconda della scelta;
- **ApiStat**: che riportano le statistiche di una data API.

3.1.2 Back End

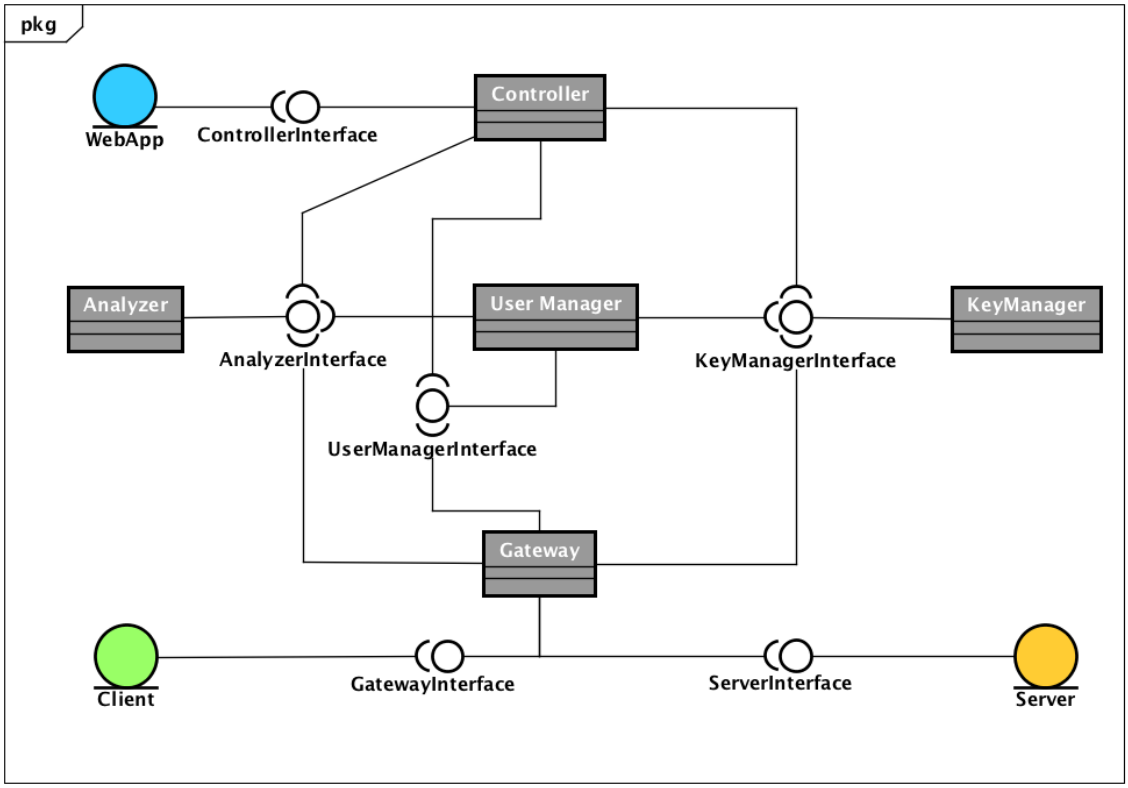

Figura 6: Back End Generale

Per quanto riguarda la parte server abbiamo adottato uno stile di programmazione orientato non agli oggetti, bensì ai microservizi, utilizzando il linguaggio *open source*$_g$ *Jolie*$_g$. I servizi principali sono:

1. **APIManager** il quale contiene le operazioni relative alle *API*$_g$;
2. **CommentManager** il quale contiene le operazioni relative ai commenti ;
3. **Gateway** il quale permette di inoltrare le chiamate degli utenti dal server centrale al quello dello sviluppatore contenente il microservizio richiesto;
4. **KeyManager** il quale contiene le operazioni relative alle *APIkeys*$_g$;
5. **UserManager** il quale contiene le operazioni relative agli utenti.

Ciascuno di questi può comunicare attraverso il protocollo *HTTP*$_g$ degli oggetti *JSON*$_g$ e quindi permette di essere integrato con la massima facilità con il Front End. Inoltre sono presenti anche dei microservizi ausiliari:

1. **Analyzer** contiene dei metodi per la raccolta di dati precisi relativi al tempo e al numero di byte usati da ogni microservizio.
2. **CourierGenerator** utilizzato per generare il file courier che viene *embeddato*$_g$ nel *gateway*$_g$, si usa per chiamare Analyzer e aggiornare i dati la base di dati, e genera l'interfaccia aggregata, ossia un'interfaccia modificata appropriata ad una chiamata al Gateway piuttosto che direttamente al microservizio dello sviluppatore.
3. **DAOs** raccolta di microservizi di accesso al database.

Per attivare tutti i microservizi è sufficiente lanciare lo script `./start.sh` presente nella cartella *Jolie*.

Durante l'installazione del server sulla propria macchina, è necessario ricordare che i microservizi utilizzano la cartella Uploads presente nella cartella Jolie per salvare i files dell'interfaccia, le immagini e la documentazione che vengono inviati dall'utente di API Market. Per questo motivo, nonostante i microservizi possano essere avviati da qualunque cartella, è necessario create un collegamento nella cartella principale del proprio server. Ad esempio su server XAMPP la cartella è XAMPP/htdocs.

4 Estendere API Market

Figura 7: Api

4.1 Estendere il front end

Grazie ad Angular-Cli estendere la parte di front-end è estremamente semplce, è sufficiente posizionarsi con il terminale nella cartella APIM che contiene APIMarket ed eseguire i comandi base descritti nella documentazione di Angular-Cli, che verranno qui riassunti:

- **Aggiungere un component:** ng generate component "nomeComponente";
- **Aggiungere un service:** ng generate service "nomeService";
- **Aggiungere un modulo:** ng generate module "nomeModulo";

Va ricordato che i service he vengono generati non sono già collegati al modulo, ma vanno aggiungi alla sezione providers del modulo stesso. Utilizzando il comando da terminale npm è possibile aggiungere numerose librerie al progetto seguendo i comandi forniti dal creatore della libreria.

4.2 Estendere il back end

4.2.1 Aggiungere una nuova operazione

Ogni microservizio che non sia un test di unità esegue in modalità concorrente, ossia qualunque richiesta riceva da qualsiasi utente, la esegue in modo parallelo. per tale motivo è necessario inserire una guardia attorno ad ogni nuova operazione inserita per fare in modo che esegua indipendentemente dalle altre.
Per fare ciò basta semplicemente inserire la nuova operazione all'interno di parentesi quadre. La prassi è che la *value* di richiesta sia denominata *request*, mentre quella di risposta sia *response*.

4.2.2 Aggiornare l'interfaccia

Ogni servizio X è accompagnato nella propria cartella da un file `XInterface.iol`. Questo file contiene le firme delle operazioni da estendere.
Per aggiungerne di nuove basta inserirle nell'elenco contenuto nel blocco Interface. Qualora fosse necessario definire un nuovo tipo, basta aggiungerlo in modo analogo a quelli definiti prima del blocco Interface. Occorre tuttavia fare attenzione per quanto riguarda i metodi dei servizi che devono comunicare con il *front-end* (I vari `.Manager`) poiché non è possibile specificare il valore del nodo radice di una *request*. A tale scopo si suggerisce di definire un figlio della radice chiamato `root` come da noi fatto ogni qualvolta necessario.

4.2.3 Aggiungere un servizio

Per aggiungere un ulteriore microservizio è sufficiente creare una nuova cartella e create i file del servizio e dell'interfaccia. Per praticità si consiglia di aggiungere anche un file di $deploy_g$ e di aggiornare il file `constants.iol` con i valori della Location

5 Verifica delle estensioni

Figura 8: Verify

Quando il progetto viene esteso con una nuova funzionalità, è necessario creare i relativi test di unità per verificare l'efficacia e la correttezza dell'estensione appena aggiunta.

5.1 front end

Per eseguire test sul codice Angular-cli mette a disposizione una serie di tools automatici creati da karma e richiamabili posizionandosi sulla cartella di APIMarket e digitando "ng test". Partiranno una serie di test di unità automatici che daranno il loro risultato su una nuova finestra del browser. Per eseguire test di copertura sarà necessario digitare il comando "ng test –code-coverage" e visitare il file index.php nella nuova cartella "coverage" creatasi nella root del progetto.

5.2 back end

Per eseguire un test su del codice Jolie è necessario creare manualmente un microservizio apposito. Questi sono raccolti nella cartella *Jolie/UnitTest* e hanno come scopo la verifica di ogni singola operazione di ogni microservizio ciascuno.

5.2.1 Aggiungere un test

Per aggiungere un nuovo test è sufficiente create un nuovo microservizio Jolie con una `outputPort` verso il servizio che si intende testare, chiamare l'operazione oggetto del test e controllare programmaticamente l'output, stampando a video l'errore in caso di fallimento.

5.2.2 Eseguire un test

Per eseguire un test è sufficiente aprire la shell nella cartella in cui il file del test è stato salvato ed eseguirlo come ogni altro servizio *Jolie*, ossia attraverso il comando `jolie servizio.ol`.

Glossario

A

Add-On : in campo informatico è un programma non autonomo che interagisce con un altro programma per ampliarne o estenderne le funzionalità originarie (specifica di plugin).

Angular-Cli : Ambiente di sviluppo, sviluppato dal team Angular, a riga di comando per creare la struttura di un'applicazione Angular 2 già configurata secondo le linee guida ufficiali.

API : È l'acronimo di Application Programming Interface, nonché di Application Program Interface.

APIkeys : È una chiave generata dal sistema che permette ad un utente di utilizzare una determinata API.

APIMarket (o APIM) : È l'acronimo di API Market ed è anche l'acronimo titolo del capitolato scelto.

D

Default : Scelta operativa elaborata da un sistema in assenza di istruzioni da parte dell'utente.

Deploy : diminutivo di deployment, indica la messa in campo o in atto (letteralmente, lo "spiegamento") di una soluzione. In informatica, in particolare, l'espressione può avere diversi significati a seconda del contesto.

E

Embedded : Nel progetto indica una parte riprogrammabile dal team per altri scopi, come la comunicazione Java e Jolie. Integrato nel sistema controlla ed è in grado di gestirne tutte o parte delle funzionalità richieste adattive.

F

Fork : Indica lo sviluppo di un nuovo progetto software che parte dal codice sorgente di un altro già esistente, ad opera di un programmatore.

Framework : Piattaforma che funge da strato intermedio tra un sistema operativo e il software che lo utilizza.

G

Gateway : Programma o computer che regola la comunicazione e lo scambio di dati fra due o più reti con protocolli diversi

GitLab : È una repository Git su base web. Gestita con wiki e con aggiunta di possibilità di tracciamento. Di vantaggio ha la possibilità di creare repository private e la lincenza è open source, sviluppato da GitLab Inc.

H

Header : Header bar si intende la barra situata in alto al nostro sito, dove presenta il menù con le principali informazioni. Nella parte di grafica e diretta verso l'utente finale l' header è di solito la parte più alta delle pagine web.

HTTP : HTTP (acronimo di HyperText Transfer Protocol, cioè "protocollo per il trasferimento dell'ipertesto") è il principale protocollo per lo scambio delle informazioni su Internet.

J

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...).

Jolie : È un linguaggio open source simile a Java e C. Con lo scopo di facilitare la progettazione dei microservizi.

JSON : JSON (JavaScript Object Notation) è un semplice formato per lo scambio di dati. ... JSON è un formato di testo completamente indipendente dal linguaggio di programmazione, ma utilizza convenzioni conosciute dai programmatori di linguaggi della famiglia del C, come C, C++, C#, Java, JavaScript, Perl, Python, e molti altri.

O

Open Source : indica un software di cui gli autori (più precisamente, i detentori dei diritti) rendono pubblico il codice sorgente, favorendone il libero studio e permettendo a programmatori indipendenti di apportarvi modifiche ed estensioni.

P

Package : Pacchetto. Di solito usato come termine specifico nel linguaggio Java.

R

Repository : È un ambiente di un sistema informativo, in cui vengono gestiti i metadati. Nel nostro contesto si utilizza spesso per indicare lo spazio GIT.

S

Shell : È la parte di un sistema operativo che permette agli utenti di interagire con il sistema stesso, impartendo comandi e richiedendo l'avvio di altri programmi. Insieme al kernel costituisce una delle componenti principali di un sistema operativo.

T

TypeScript : È un linguaggio di programmazione libero ed Open source sviluppato da Microsoft. Si tratta di un Super-set di JavaScript che basa le sue caratteristiche su ECMAScript 6, capo del progetto è Anders Hejlsberg.

X

XML : È un metalinguaggio per la definizione di linguaggi di markup, ovvero un linguaggio marcatore basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo.