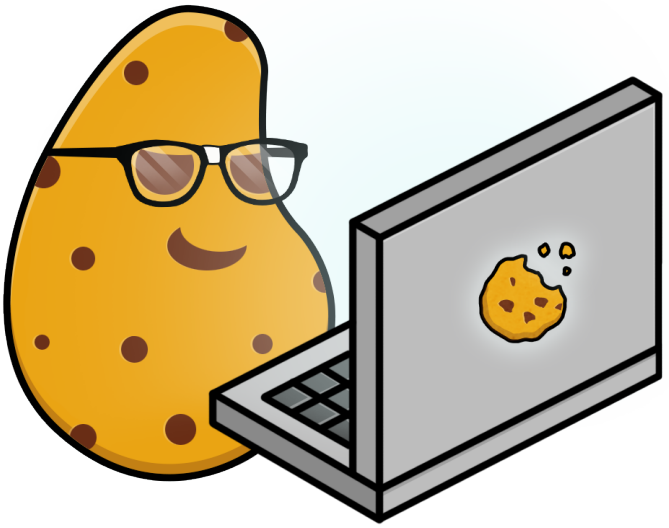

Manuale Manutentore

Gruppo DigitalCookies — Progetto SWEDesigner

digitalcookies.group@gmail.com

Informazioni sul documento

Versione	1.0.0
Redazione	Alberto Giudice, Alberto Rossetti
	Carlo Sindico, Alessia Bragagnolo
Verifica	Davide Albertini, Saverio Follador
Approvazione	Christian Cabrera
Uso	Esterno
Distribuzione	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Zucchetti S.p.A.
	Gruppo DigitalCookies

Descrizione

Documento contenente il Manuale Manutentore per il progetto *SWEDesigner* del gruppo DigitalCookies.

Registro delle modifiche

Versione	Data	Collaboratori	Ruolo	Descrizione
1.0.0	06-07-2017	Christian Cabrera	Responsabile	Approvazione documento
0.2.0	06-07-2017	Saverio Follador	Verificatore	Verifica documento
0.1.0	05-07-2017	Davide Albertini	Verificatore	Verifica documento
0.0.8	04-07-2017	Aleberto Giudice	Amministratore	Stesura A
0.0.7	04-07-2017	Alessia Bragagnolo	Programmatore	Stesura sezione 8 e 9
0.0.6	30-07-2017	Alberto Rossetti	Progettista	Stesura sezione 7
0.0.5	30-06-2017	Carlo Sindico	Progettista	Stesura sezione 5 e 6
0.0.4	29-06-2017	Alberto Giudice	Amministratore	Stesura sezione 4
0.0.3	29-06-2017	Alberto Rossetti	Progettista	Requisiti di sistema
0.0.2	28-06-2017	Alberto Rossetti	Progettista	Stesura sezione 2
0.0.1	28-06-2017	Saverio Follador	Amministratore	Creazione del template

Indice

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Il documento rappresenta il Manuale del Manutentore per l'applicazione *SWEDesigner* sviluppato dal gruppo DigitalCookies. Esso ha la finalità di spiegare al manutentore del software le modalità di installazione e di utilizzo, i requisiti necessari per poterlo utilizzare, le librerie e *framework*$_G$ esterni utilizzati per lo sviluppo dell'applicazione oltre alla struttura dei *package*$_G$ e delle *classi*$_G$ contenute al loro interno, così da aiutarlo nella ricerca delle varie funzionalità.

1.2 Scopo del prodotto

Lo scopo del prodotto è la progettazione di diagrammi *UML*$_G$ da cui sia possibile la generazione di codice *Java*$_G$. Il codice può essere generato dall'utente a partire dai diagrammi UML delle classi e da una versione modificata del diagramma delle *attività*$_G$. L'utente, interagendo con il sistema, è in grado di:

- delineare la struttura delle classi utilizzando lo standard UML;
- definire il corpo dei metodi di ogni classe sfruttando una versione modificata del diagramma delle attività;
- generare un applicativo scritto in codice Java a partire dai diagrammi sopracitati, modificarlo e salvarlo in locale.

All'utente è inoltre fornita una *libreria*$_G$ per generare con facilità diagrammi relativi al dominio dei giochi di carte e la possibilità di salvare in locale i propri progetti nel formato *JSON*$_G$.

1.3 Informazioni utili

Nell'andare a stilare il seguente manuale abbiamo assunto che l'utente a cui è rivolto *SWEDesigner* possegga delle conoscenze basilari nel campo della programmazione ad oggetti.
Per completezza, in appendice A, è possibile trovare un glossario che raccoglie termini tecnici o riguardanti particolari funzionalità di *SWEDesigner*. Per identificare i termini presenti a glossario, la loro prima occorrenza all'interno del documento è in corsivo e marcata con una G pedice.

2 Tecnologie utilizzate

2.1 Client

Il *Client*$_G$ di SWEDesigner è implementato con le tecnologie web richieste dal *capitolato*$_G$ d'appalto: HTML5, CSS3 e *JavaScript*$_G$. Il Client ha bisogno di alcune librerie JavaScript *open-source*$_G$, come richiesto dal capitolato.

2.1.1 GoJS

GoJS è una libreria JavaScript ricca di funzionalità, in quanto permette di realizzare diagrammi interattivi in maniera altamente personalizzabile. Tra le funzionalità più caratteristiche troviamo:

- creazione di elementi per diagrammi base (e.g. rettangoli, cerchi, commenti);
- creazione di elementi custom basati su *SVG*$_G$;
- creazione collegamenti personalizzabili (punta, etichette);
- inserimento, modifica di immagini;
- altamente *event-driven*$_G$;
- stampa dei diagrammi prodotti;
- zoom in/out;
- serializzazione/deserializzazione con *file*$_G$ JSON.

.

2.1.1.1 Vantaggi

- *HTML5*: *GoJS* richiede che una pagina HTML5 sia popolata da un semplice tag , che conterrà un diagramma;
- GoJS, per il suo normale utilizzo, non richiede alcuna libreria JavaScript o framework.

2.1.1.2 Svantaggi

- Il linguaggio di codifica risulta a volte complesso e articolato in particolare per la creazione di nodi;
- La documentazione fornita ha pochi esempi pratici e può risultare complessa di comprensione.

2.1.2 Angular2

Angular2$_G$ è un framework JavaScript, patrocinato da *Google*$_G$, utile a semplificare la realizzazione di applicazioni Web single page (*SPA*$_G$): favorisce un approccio dichiarativo allo *sviluppo*$_G$ client-side, migliore per la creazione di interfacce utente, laddove l'approccio imperativo è ideale per realizzare la logica applicativa.

2.1.2.1 Vantaggi

- ***Data-Binding*$_G$ Bidirezionale:** il data-binding bidirezionale di Angular2 gestisce la sincronizzazione tra il *DOM*$_G$ e il modello, e viceversa;
- ***MVVM*$_G$:** Angular2 permette una facile implementazione lato client del pattern MVVM;
- ***Dependency injection*$_G$:** Angular2 permette di descrivere in maniera dichiarativa quali sono le dipendenze che l'applicazione possiede, isolando i comportamenti e le responsabilità dei componenti e garantendo un facile rimpiazzo di quest'ultimi. Questo meccanismo favorisce inoltre la testabilità del codice dell'applicazione.

2.1.2.2 Svantaggi

- Il ciclo di vita di un'applicazione Angular2 è complesso. La compilazione e il link non sono intuitivi e in specifici casi possono essere confusi (ricorsione in compilazione, collisioni tra direttive).

2.1.3 HTML5

L'*HTML5* è un *linguaggio di markup*$_G$ per la strutturazione delle pagine web, pubblicato come *W3C*$_G$ *Recommendation* da ottobre 2014. Le novità introdotte dall'*HTML5* rispetto all'HTML 4 sono finalizzate soprattutto a migliorare il disaccoppiamento fra struttura, definita dal markup, caratteristiche di resa (tipo di carattere, colori, eccetera), definite dalle direttive di stile, e contenuti di una pagina web, definiti dal testo vero e proprio. Inoltre l'*HTML5* prevede il supporto per la memorizzazione locale di grandi quantità di dati scaricati dal web browser, per consentire l'utilizzo di applicazioni basate su web (come per esempio le caselle di posta di *Google* o altri servizi analoghi) anche in assenza di collegamento a Internet.

2.1.3.1 Vantaggi

- **Standard:** *HTML5* è uno standard *W3C*.

2.1.3.2 Svantaggi

- **Scarso supporto ai vecchi browser:** *HTML5* non è supportato dai browser più datati, per rimediare tali mancanze, sarà necessario utilizzare altre tecnologie di supporto.

2.1.4 CSS

CSS$_G$ (Cascading Style Sheets) è un linguaggio utile a descrivere la presentazione di un documento scritto in un linguaggio di markup; uno degli scopi principali che hanno spinto alla creazione di questo linguaggio è la necessità di separare la struttura del documento dalla sua presentazione rendendo cosi maggiormente accessibili le pagine dei siti web.

2.1.4.1 Vantaggi

- **Standard:** CSS è uno standard *W3C*;
- **Accessibilità:** la separazione tra struttura e presentazione consente di avere pagine molto più accessibili rispetto ad avere tutto in un unico file.

2.1.4.2 Svantaggi

- **Comportamento imprevedibile**: il comportamento delle regole CSS potrebbe variare a seconda dei browser utilizzati.

2.2 Server

2.2.1 Amazon Web Service

Il *front-end*$_G$ e il *back-end*$_G$ della nostra web-application sono ospitati su un *server*$_G$ *AWS*$_G$.

2.2.1.1 Vantaggi

- Facile da configurare e da utilizzare;
- Alto livello di sicurezza.

2.2.1.2 Svantaggi

- I server sono virtuali e non garantiscono delle prestazioni stabili quanto un server dedicato;

- I tipi di istanza sono rigidi e non prevedono elasticità per quanto riguarda le prestazioni offerte.

2.2.2 Node.js

Node.js$_G$ è un framework event-driven basato sul motore *JavaScript* V8 di Google *Chrome*$_G$, per piattaforme *UNIX*$_G$. Node.js usa un modello I/O non bloccante e ad eventi, risulta quindi un framework leggero ed efficiente per l'utilizzo server-side.

2.2.2.1 Vantaggi

- **Approccio asincrono**: grazie alla modalità event-driven node.js evita il modello basato su processi o *thread*$_G$ concorrenti utilizzato dai classici web server. Questa caratteristica permette una migliore efficienza in termini di prestazioni, sfruttando la gestione di altri processi durante le attese in maniera asincrona.
- **Architettura modulare**: node.js si appoggia allo strumento node package manager (*npm*$_G$) che permette un'organizzazione semplice del lavoro grazie alla combinazione di librerie con moduli importati. Infatti npm permette allo sviluppatore di accedere ai package messi a disposizione dalla community.

2.2.2.2 Svantaggi

- **Modello di programmazione asincrono**: il *programmatore*$_G$ che utilizza per la prima volta node.js deve adottare uno stile di programmazione asincrono, il quale risulta essere più complicato della programmazione di input e output lineare a blocchi; il codice prodotto risulta essere più disordinato e meno comprensibile e costringe lo sviluppatore a fare affidamento su *callback*$_G$ nidificate;

Le applicazioni Node.js vengono eseguite su un singolo thread, sebbene sia utilizzato un modello multi-thread per la gestione degli eventi legati a file e connessioni di rete.

2.2.3 Express

Express è un framework minimale per creare applicazioni web con Node.js. Express offre funzionalità che semplificano e aumentano le potenzialità di Node.js, fornendo una migliore implementazione del sistema di *routing*$_G$. Ciò è ottenuto incrementando le funzioni di richiesta e risposta, estendendole per una maggior flessibilità, integrando nuovi *middleware*$_G$ e agevolando la realizzazione delle viste.

Express non limita l'utente nella scelta del linguaggio di templating, lo aiuta a gestire le *route*$_G$, le *request*$_G$ e le *view*$_G$.

2.2.3.1 Vantaggi

- Molte funzionalità di routing, tra cui gestori separati per richieste PUT, GET e POST;
- Gestione dei file statici;

2.2.3.2 Svantaggi

- Creazione degli endpoint molto laboriosa;
- Refactoring del codice oneroso quando il progetto raggiunge dimensioni elevate.

2.2.4 MongoDB

MongoDB$_G$ è un *database*$_G$ *NoSQL*$_G$ open-source scalabile e altamente performante di tipo *document-oriented*$_G$, in cui i dati sono archiviati sotto forma di documenti in stile JSON con schemi dinamici che MongoDB chiama *BSON*$_G$, secondo una struttura semplice e potente.

2.2.4.1 Vantaggi

- Alte performance: non ci sono *join*$_G$ che possono rallentare le operazioni di lettura o scrittura. L'indicizzazione include gli indici di chiave anche sui documenti innestati e sugli array, permettendo una rapida interrogazione al database;
- Affidabilità: alto meccanismo di replicazione su server;
- Schemaless: non esiste nessuno schema, è più flessibile e può essere facilmente trasposto in un modello ad oggetti;
- Map-Reduce: Permette di processare parallelamente i dati.

2.2.4.2 Svantaggi

- Assenza di Join: non è possibile creare delle join come nei database relazionali, il che significa che quando si necessita di tale funzionalità bisogna creare più *query*$_G$ e fare il join dei dati manualmente con il codice. Quest'aspetto porta a creare codice non flessibile, in quanto le strutture dei dati potrebbero cambiare;
- Utilizzo della memoria: MongoDB tende ad utilizzare più memoria di quella necessaria, in quanto deve salvare il nome della chiave per ogni documento.

Altre funzionalità comprendono la possibilità di creare delle query ad hoc, l'Auto-Sharding, ovvero la capacità di scalare orizzontalmente e di aggiungere nuove macchine al database operativo. Inoltre, MongoDB supporta la definizione di *collection*$_G$ con una

dimensione fissata. Questo particolare tipo di collection mantiene l'ordine di inserimento e, una volta raggiunta la dimensione massima prefissata, si comporta come una lista circolare.

2.2.5 Mongoose

Mongoose è una libreria per interfacciarsi a MongoDB che permette di definire degli schemi per modellare i dati del database, imponendo una certa struttura per la creazione di nuovi *Document*$_G$. Inoltre, fornisce molti strumenti utili per la *validazione*$_G$ dei dati, per la definizione di query e per il cast dei tipi predefiniti.

2.2.5.1 Vantaggi

- La documentazione di *Mongoose* è ben fornita e descrive le API_G in maniera completa, fornendo degli *snippet*$_G$ del codice sorgente;
- Integra direttamente la validazione dei dati.

2.2.5.2 Svantaggi

- L'utilizzo di schemi è in contrasto con lo scopo di avere un database NoSQL.

3 Requisiti di sistema

SWEDesigner è una applicazione web funzionante su tutti i dispositivi desktop. Non è presente una versione mobile.

3.1 Dispositivi supportati

L'applicazione *SWEDesigner* è compatibile con i seguenti sistemi operativi desktop:

- Microsoft Windows 10
- Apple OSX Sierra
- Ubuntu 16.04 LTS 64 bit
- Manjaro 17

3.2 Browser supportati

Di seguito viene fornito un breve elenco delle versioni minime di tutti i browser sui quali il funzionamento del nostro prodotto è garantito:

- Google Chrome 50
- *Mozilla Firefox*$_G$ 48
- *Safari*$_G$ 9.1

Per rendere effettivo il funzionamento su tali browser, deve essere abilitato JavaScript.

4 Configurazione ambiente di lavoro

4.1 Installazione prerequisiti

Per poter utilizzare l'applicazione SWEDesigner è necessario soddisfare alcuni prerequisiti di installazione.

4.1.1 Installazione Node.js

Il seguente link porta alla pagina dove poter scaricare ed installare Node.js: https://nodejs.org/it/.

4.1.2 Installazione Npm

Con l'installazione di Node.js assicurarsi che si sia installato anche Npm digitando da terminale il comando `npm -v` che in caso positivo restituisce la versione presente sul proprio pc. In caso contrario procedere all'installazione manuale tramite il seguente link: https://www.npmjs.com/get-npm.

4.1.3 Installazione angular CLI

Angular cli è possibile installarlo mediante le istruzioni illustrate al seguente link: https://www.npmjs.com/package/angular-cli#installation.

4.2 Configurazione dell'applicazione SWEDesigner

Per prima cosa è necessario clonare la seguente repository:
https://github.com/DigitalCookiesGroup.
Successivamente spostarsi all'interno della cartella Front-End e aggiornare le dipendenze mediante il comando da terminale `npm install`. Effettuare la build di SWEDesigner tramite il comando `npm run build`. Una volta effettuata questa operazione copiare il contenuto della cartella dist che si sarà creata in Back-End/public. In seguito spostarsi nella cartella Back-End e aggiornare le dipendenze del Back-End tramite il comando `npm install`. Infine avviare il server tramite il comando `npm start`.

5 Architettura

5.1 Metodo e formalismo

L'architettura dell'applicazione verrà illustrata seguendo un approccio *top-down*$_G$, quindi descrivendo l'architettura dal generale al particolare. Verrano descritti per primi i package e i componenti, e poi nel dettaglio le classi in essi contenute fornendo una loro descrizione, specificandone l'utilizzo ed evidenziandone le relazioni con le atre classi.

5.1.1 Informazioni generali

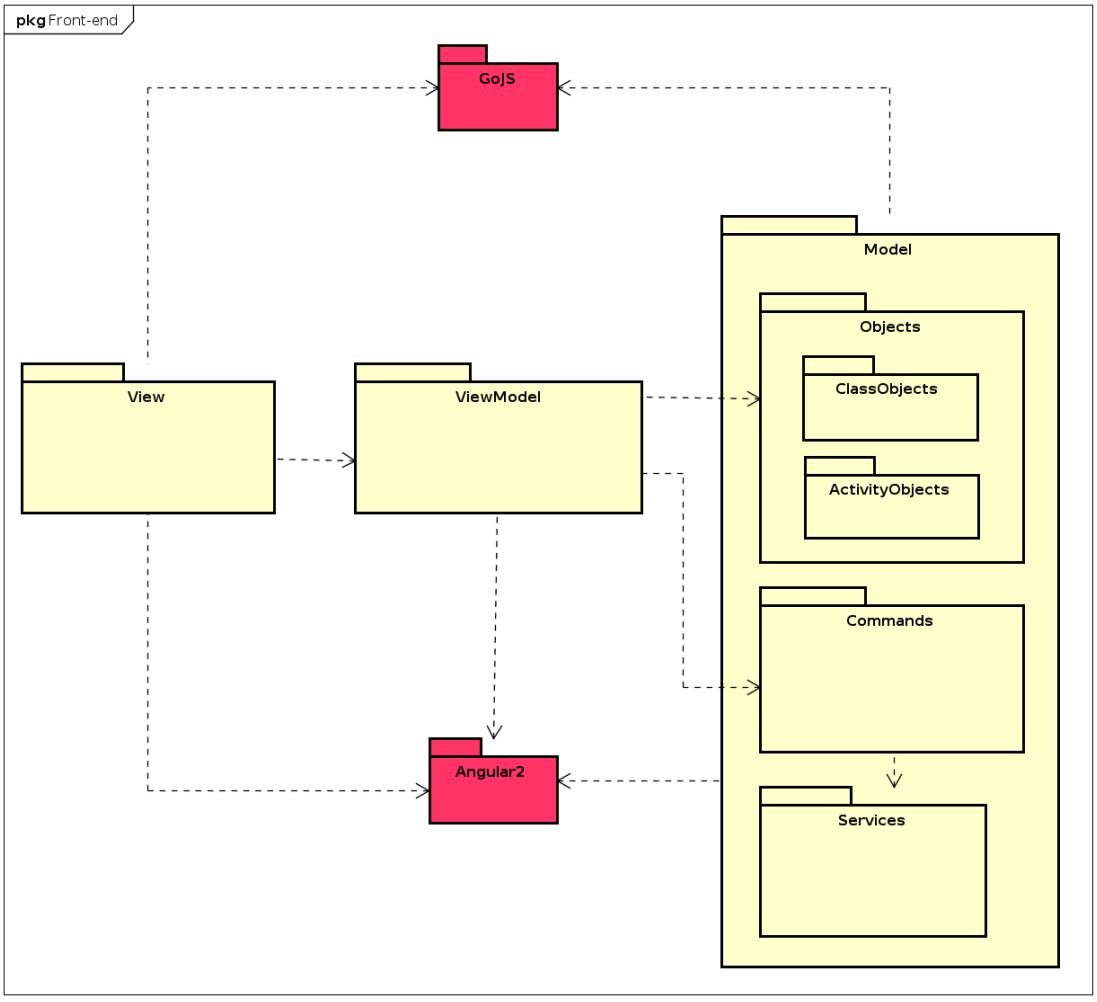

Figura 1: Front-end

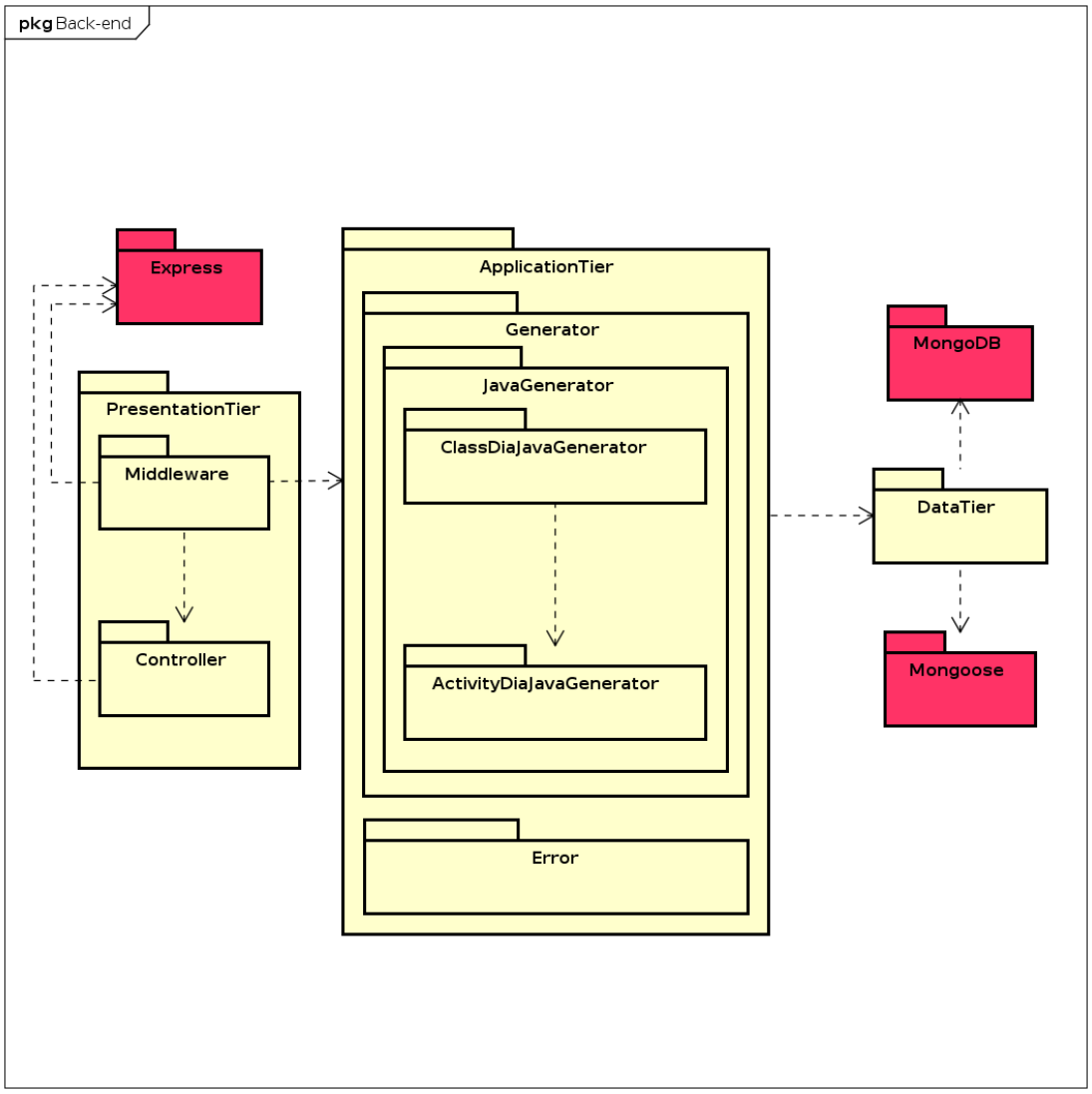

Figura 2: Back-end

- **Descrizione**: architettura ad alto livello dell'applicazione SWEDesigner. L'architettura è suddivisa in due applicazioni distinte che comunicano tra di loro. Il front-end che è stato pensato come una Single Page Application scritta nei linguaggi HTML5, CSS3 e JavaScript e sviluppata seguendo il design pattern MVVM. Tramite il front-end l'utente potrà creare diagrammi delle classi, delle attività e generare codice a partire da essi visualizzandolo su un editor online con la possibilità di scaricarlo come file. Il back-end è stato sviluppato seguendo il pattern *Three-Tier*$_G$. Si occupa di gestire la ricezione e l'invio dei file JSON, la manipolazione di quest'ultimi per la generazione del codice, la gestione degli errori e il

recupero dei dati da libreria.

- **Packages contenuti**:
  - `SWEDesigner::Front-end`: package contenente le classi ed i package che compongono il front-end di SWEDesigner;
  - `SWEDesigner::Back-end`: package contenente le classi ed i package che compongono il back-end di SWEDesigner;
- **Librerie e framework**:
  - GoJs: libreria JavaScript ricca di funzionalità, in quanto permette di realizzare diagrammi interattivi in maniera altamente personalizzabile;
  - Angular2: framework JavaScript, patrocinato da Google, utile a semplificare la realizzazione di applicazioni Web single page;
  - Express: framework minimale per creare applicazioni web con Node.js. Express offre funzionalità che semplificano e aumentano le potenzialità di Node.js, fornendo una migliore implementazione del sistema di routing;
  - Mongoose: libreria per interfacciarsi a MongoDB che permette di definire degli schemi per modellare i dati del database, imponendo una certa struttura per la creazione di nuovi Document.

6 SWEDesigner::Front-End

6.1 SWEDesigner::Front-End::View

- **Descrizione**: questo package contiene le classi che rappresentano l'editor dei diagrammi delle classi, delle attività e l'editor che permette di modificare il codice prodotto dai diagrammi precedentemente creati;
- **Classi contenute**:
  - `SingePageApp`: questa classe gestisce la visualizzazione dell'interfaccia grafica di SWEDesigner. Contiene le classi riguardanti l'editor dei diagrammi e l'editor del codice generato;
  - `CodeEditor`: questa classe gestisce la visualizzazione dell'interfaccia grafica dell'editor del codice generato dal Back-end;
  - `DiagramEditor`: questa interfaccia rappresenta la struttura per ciascun editor di diagramma;
  - `ClassDiagramEditor`: questa classe gestisce la visualizzazione dell'interfaccia grafica dell'editor del diagramma delle classi;
  - `ActivityDiagramEditor`: questa classe gestisce la visualizzazione dell'interfaccia grafica dell'editor del diagramma delle attività;
  - `DiagramPalette`: questa interfaccia rappresenta la struttura per ciascuna palette del diagramma;
  - `ClassDiagramPalette`: questa classe gestisce la visualizzazione dell'interfaccia grafica della palette del diagramma delle classi;
  - `ActivityDiagramPalette`: questa classe gestisce la visualizzazione dell'interfaccia grafica delle palette del diagramma delle attività;
  - `DiagramFactory`: questa interfaccia rappresenta la struttura per le operazioni di creazione dei diagrammi;
  - `ClassFactory`: questa classe implementa le operazioni di creazione dei componenti del diagramma delle classi;
  - `ActivityFactory`: questa classe implementa le operazioni di creazione dei componenti del diagramma delle attività.

6.2 SWEDesigner::Front-End::ViewModel

- **Descrizione**: questo package contiene tutte le classi necessarie a regolare la comunicazione e l'interazione tra i package `Front-end::View` e `Front-end::Model` e funge da responsabile per la gestione logica della `Front-end::View`;
- **Classi contenute**:
  - `EditorObjController`: questa classe gestisce le richieste degli editor dei diagrammi interfacciandosi con la parte del `Front-end::Model`;
  - `PaletteObjController`: questa classe gestisce le richieste della palette dei diagrammi interfacciandosi con la parte del `Front-end::Model`;
  - `CodeCommandController`: questa classe gestisce le richieste dell'editor del codice generato interfacciandosi con la parte del `Front-end::Model`;
  - `PaletteCommandController`: questa classe gestisce le richieste della palette dei diagrammi interfacciandosi con la parte del `Front-end::Model`.s

6.3 SWEDesigner::Front-End::Model

- **Descrizione**: questo package contiene i modelli dei blocchi utilizzati dalla `Front-end::View`, i comandi che permettono la richiesta di generazione codice e richiesta di template;
- **Package contenuti**:
  - `Front-end::Model::Objects`
  - `Front-end::Model::Objects::ClassObjects`
  - `Front-end::Model::Objects::ActivityObjects`
  - `Front-end::Model::Commands`
  - `Front-end::Model::Services`

6.4 SWEDesigner::Front-End::Model::Objects

- **Descrizione**: questo package contiene tutti i blocchi che costituiscono il diagramma delle classi `Front-End::Model::Objects::ClassObjects` ed il diagramma delle attività `Front-End::Model::Objects::ActivityObjects`;
- **Package contenuti**:
  - `Front-end::Model::Objects::ClassObjects`
  - `Front-end::Model::Objects::ActivityObjects`

- **Classi contenute**:
  - `BaseDiaObj`: questa classe astratta rappresenta un contratto comune tra le classi `Front-end::Model::Objects::ClassObjects::ClassDiaObj` e `Front-end::Model::Objects::ActivityObjects::ActivityDiaObj`.

6.5 SWEDesigner::Front-End::Model::Objects::ClassObjects

- **Descrizione**: questo package contiene tutti i blocchi che vanno a comporre il diagramma delle classi;
- **Classi contenute**:
  - `ClassDiaObj`: questa classe astratta rappresenta un contratto comune tra le classi che rappresentano i blocchi del diagramma delle classi;
  - `Class`: questa classe rappresenta un blocco classe del diagramma delle classi;
  - `Comment`: questa classe rappresenta un blocco commento del diagramma delle classi;
  - `Relation`: questa classe astratta rappresenta un contratto comune tra le classi che rappresentano le relazioni del diagramma delle classi;
  - `Dependency`: questa classe rappresenta una relazione di tipo dipendenza del diagramma delle classi;
  - `Association`: questa classe rappresenta una relazione di tipo associazione del diagramma delle classi;
  - `Aggregation`: questa classe rappresenta una relazione di tipo aggregazione del diagramma delle classi;
  - `Composition`: questa classe rappresenta una relazione di tipo composizione del diagramma delle classi;
  - `Generalization`: questa classe rappresenta una relazione di tipo generalizzazione del diagramma delle classi.

6.6 SWEDesigner::Front-End::Model::Objects::ActivityObjects

- **Descrizione**: questo package contiene tutti i blocchi che vanno a comporre il diagramma delle attività;
- **Classi contenute**:
  - `ActivityDiaObj`: questa classe astratta rappresenta un contratto comune tra le classi che rappresentano i blocchi del diagramma delle attività;

- `VariableObj`: questa classe rappresenta un blocco variabile del diagramma delle attività;
- `ExistingVariableObj`: questa classe rappresenta un blocco variabile esistente del diagramma delle attività;
- `methodCallObj`: questa classe rappresenta un blocco chiamata del metodo del diagramma delle attività;
- `CycleObj`: questa classe rappresenta un blocco ciclo del diagramma delle attività;
- `ifElseObj`: questa classe rappresenta un blocco if/else del diagramma delle attività;
- `OperatorObj`: questa classe rappresenta un blocco operatore del diagramma delle attività;
- `StepObj`: questa classe rappresenta un blocco avanzamento del diagramma delle attività;
- `JollyObj`: questa classe rappresenta un blocco jolly del diagramma delle attività.

6.7 SWEDesigner::Front-End::Model::Commands

- **Descrizione**: questo package contiene i comandi relativi al progetto di richiesta codice e richiesta template;
- **Classi contenute**:
  - `Command`: questa classe astratta rappresenta un contratto comune per l'esecuzione di un comando che viene chiamato dalle classi `PaletteCommandController` o `CodeCommandController` quando l'utente decide di richiedere un template o la generazione di codice;
  - `RequestCode`: questa classe rappresenta un comando di richiesta generazione codice del progetto richiesto dall'utente;
  - `GetTemplate`: questa classe rappresenta un comando di richiesta template richiesto dall'utente.

6.8 SWEDesigner::Front-End::Model::Services

- **Descrizione**: questo package contiene tutte le classi necessarie per gestire la comunicazione tra Front-end e Back-end;
- **Classi contenute**:

– `ServerConnector`: questa classe gestisce le comunicazioni con l'applicazione back-end, sfruttando le funzionalità `http` offerte dal framework Angular2.

7 SWEDesigner::Back-End

7.1 SWEDesigner::Back-end::PresentationTier

- **Descrizione**:
  Questo package contiene le classi che gestiscono le richieste da parte del client.
- **Package contenuti**:
  - `Back-end::PresentationTier::Middleware`
  - `Back-end::PresentationTier::Controller`

7.2 SWEDesigner::Back-end::PresentationTier::Middleware

- **Descrizione**:
  Questo package contiene le classi per la gestione delle comunicazioni tra server e client, e per la gestione dei possibili errori.
- **Classi contenute:**
  - **MiddlewareLoader**:
    Questa classe astratta definisce un contratto comune per tutte le richieste REST provenienti dal client;
  - **ErrorHandler**:
    Questa classe gestisce gli errori generati nei precedenti middleware.
  - **Router**:
    Classe che si occupa della richiesta di risorse in base all'URI ricevuto e ad invocare l'opportuno metodo sulla classe `ApplicationController`;
  - **NotFoundHandler**:
    Classe che si occupa della gestione dell'errore di pagina non trovata.

7.3 SWEDesigner::Back-end::PresentationTier::Controller

- **Descrizione**:
  Questo package contiene la classe che gestisce la richiesta della index-page.
- **Classi contenute:**
  - **IndexGiver**:
    Questa classe gestisce la richiesta della single page da parte del client.

7.4 SWEDesigner::Back-end::ApplicationTier

- **Descrizione**:
  Questo package contiene tutte le componenti inerenti la *business logic*$_G$ dell'applicazione Back-end, scritte in JavaScript.
- **Package contenuti**:
  - Back-end::Generator
  - Back-end::Error
- **Classi contenute:**
  - **ApplicationController**:
    Questa classe gestisce le comunicazioni tra i package della struttura three-tier.

7.5 SWEDesigner::Back-end::ApplicationTier::Generator

- **Descrizione**:
  Questo package contiene le classi necessarie per trasformare gli oggetti JSON in codice sorgente di un un particolare linguaggio target.
- **Classi contenute:**
  - **BaseGenerator**:
    Questa interfaccia rappresenta un contratto comune a tutte le classi che generano codice.

7.6 SWEDesigner::Back-end::ApplicationTier::Generator::JavaGenerator

- **Descrizione**:
  Questo package contiene le classi relative alla generazione di codice Java partendo dal file JSON ricavato dai diagrammi delle classi e delle attività.
- **Package contenuti**:
  - `Back-end::Generator::JavaGenerator::ClassDiaJavaGenerator`
  - `Back-end::Generator::JavaGenerator::ActivityDiaJavaGenerator`
- **Classi contenute:**
  - **JavaGenerator**:
    Questa classe si occupa della generazione di codice Java partendo dal file JSON ricavato dai diagrammi delle classi e delle attività.

7.7 SWEDesigner::Back-end::ApplicationTier::Generator::JavaGenerator ::ClassDiaJavaGenerator

- **Descrizione**:
  Questo package contiene le classi relative alla generazione di codice Java partendo dal file JSON ricavato dai diagrammi delle classi.
- **Classi contenute:**
  - **ClassDiaJavaGenerator**:
    Questa classe si occupa della generazione di codice Java di un diagramma delle classi partendo dal file JSON.
  - **ClassJavaGenerator**:
    Questa classe si occupa della generazione del codice Java di una classe partendo dal file JSON.
  - **AttributeJavaGenerator**:
    Questa classe si occupa della generazione del codice Java di un attributo partendo dal file JSON.
  - **MethodJavaGenerator**:
    Questa classe si occupa della generazione del codice Java di un metodo partendo dal file JSON.
  - **RelationJavaGenerator**:
    Questa classe si occupa della generazione del codice Java di una relazione partendo dal file JSON.
  - **CommentJavaGenerator**:
    Questa classe si occupa della generazione del codice Java di un commento partendo dal file JSON.

7.8 SWEDesigner::Back-end::ApplicationTier::Generator::JavaGenerator ::ActivityDiaJavaGenerator

- **Descrizione**:
  Questo package contiene le classi relative alla generazione di codice Java partendo dal file JSON ricavato dai diagrammi delle attività.
- **Classi contenute:**
  - **ActivityDiaJavaGenerator**:
    Questa classe si occupa della generazione di codice Java di un diagramma delle attività partendo dal file JSON.

- **InstructionJavaGenerator**:
  Questa classe astratta rappresenta un contratto comune a tutte le classi che generano codice Java relativo a blocchi del diagramma delle attività.
- **VariableJavaGenerator**:
  Questa classe si occupa della generazione del codice Java di un blocco variabile del diagramma delle attività partendo dal file JSON.
- **VariableConnectJavaGenerator**:
  Questa classe si occupa della generazione del codice Java di un blocco connessione variabile esistente del diagramma delle attività partendo dal file JSON.
- **MethodCallJavaGenerator**:
  Questa classe si occupa della generazione del codice Java di un blocco chiamata del metodo del diagramma delle attività partendo dal file JSON.
- **JollyJavaGenerator**:
  Questa classe si occupa della generazione del codice Java di un blocco jolly del diagramma delle attività partendo dal file JSON.
- **StepJavaGenerator**:
  Questa classe si occupa della generazione del codice Java di un blocco avanzamento del diagramma delle attività partendo dal file JSON.
- **CycleJavaGenerator**:
  Questa classe si occupa della generazione del codice Java di un blocco ciclo del diagramma delle attività partendo dal file JSON.
- **OperatorJavaGenerator**:
  Questa classe si occupa della generazione del codice Java di un blocco operatore del diagramma delle attività partendo dal file JSON.
- **IfElseJavaGenerator**:
  Questa classe si occupa della generazione del codice Java di un blocco if/else del diagramma delle attività partendo dal file JSON.

7.9 SWEDesigner::Back-end::ApplicationTier::Error

- **Descrizione**:
  Questo package contiene la classe che gestisce la visualizzazione degli errori.
- **Classi contenute:**
  - **ErrorApplication**:
    Questa classe rappresenta un errore che può verificarsi nel `Back-end`.

7.10 SWEDesigner::Back-end::DataTier

- **Descrizione**:
  Questo package contiene la classe che gestisce il recupero dei dati di libreria interfacciandosi con un database MongoDB, tramite la libreria Mongoose.
- **Classi contenute:**
  - **Template**:
    Questa classe gestisce la richiesta di un template JSON da parte del client richiedendolo al Database.

8 Estensione delle funzionalità

8.1 Login tramite GitHub o altre piattaforme

Uno degli obiettivi principali dell'applicazione SWEDesigner è di poter essere utilizzata su qualunque piattaforma web con facilità con la possibilità di recuperare i propri progetti precedentemente salvati. Per questo, una delle funzionalità desiderabili è permettere all'utente di non perdere tempo durante la fase di registrazione, ma di poterlo fare appoggiandosi a piattaforme come $GitHub_G$, $Gitlab_G$ o altri repository. L'utente avrà così la possibilità di caricare un progetto salvato e di salvare il progetto corrente su un proprio repository.

8.2 Generazione nuovi linguaggi

Il servizio SWEDesigner è stato sviluppato per poter generare in futuro più linguaggi di programmazione. Tali linguaggi sono identificati da package interni al package Generator, che possono essere aggiunti nel tempo.

8.3 Aggiunta elementi a libreria

Uno dei servizi aggiuntivi offerti dall'applicazione SWEDesigner è la presenza di una lista di classi e pattern specifica per generare con facilità diagrammi relativi al dominio dei giochi di carte. L'utente potrà inserire le classi o il design pattern, selezionando fra quelli forniti dalla libreria. Queste liste potranno essere successivamente aggiornate e/o modificate interfacciandosi con un database MongoDB, tramite libreria Mongoose. Il Database viene gestito dalla classe `Back-end::DataTier::Template` che esegue la richiesta di un template tramite delle query utilizzando la libreria Mongoose.
I JSON all'interno della tabella Template saranno distinguibili tramite quattro campi:

- Id;
- Titolo;
- Categoria;
- Corpo.

Infine per visualizzare il JSON ricevuto da libreria basterà eseguire il comando `console.log()` del JSON stesso da terminale.

8.4 Aggiunta nuovi diagrammi

Attualmente l'applicazione SWEDesigner fornisce la possibilità di creare un software di costruzione di diagrammi UML con relativa generazione di codice Java. Il codice può essere generato dall'utente a partire dai diagrammi UML delle classi e da una versione modificata del diagramma delle attività. Sarà possibile in futuro raffinare la generazione di codice introducendo nuovi diagrammi, come ad esempio il diagramma di sequenza per permettere di generare codice concorrente.

8.5 Aggiunta nuovi blocchi

Il prodotto SWEDesigner utilizza una struttura a blocchi per permettere un maggiore dialogo tra il diagramma delle classi e il diagramma delle attività. Il diagramma è pertanto formato dal seguente insieme di blocchi modificati:

- Variabile;
- Chiamata del metodo;
- Ciclo;
- If/else;
- Operatore;
- Avanzamento;
- Jolly.

Successivamente sarà possibile aggiungere nuovi blocchi in base alle necessità richieste dall'utente.

9 Estensione del codice

9.1 Generazione nuovi linguaggi

Al fine di garantire la generazione di codice di nuovi linguaggi di programmazione all'applicazione SWEDesigner, sarà necessario introdurre nuovi package relativi a tali linguaggi all'interno del package `Back-end::ApplicationTier::Generator`. Questa interfaccia rappresenta un contratto comune a tutte le classi che generano codice. Rappresenta l'interfaccia Strategy del design pattern Strategy. Bisognerà creare il parser specifico per il linguaggio selezionato. Si avrà inoltre bisogno di introdurre un opportuno controllo sul metodo `+getGeneratedCode` contenuto nella classe `Back-end::ApplicationTier::ApplicationController` per effettuare la richiesta del tipo di linguaggio richiesto. Occorre fare riferimento a tale classe per aggiungere o modificare tale controllo poichè si occupa dell'utilizzo della classe `Back-end::ApplicationTier ::Generator::BaseGenerator` per invocare l'algoritmo di generazione codice nel linguaggio voluto attraverso le sue sottoclassi. Infine per gestire le richieste provenienti dal client, per ciascun linguaggio sarà opportuno creare una richiesta REST per invocare l'opportuno metodo sulla classe `Back-end::ApplicationTier::ApplicationController`.

9.2 Aggiunta nuovi diagrammi e blocchi

Attualmente l'applicazione SWEDesigner fornisce la possibilità di creare un software di costruzione di diagrammi UML a partire dai diagrammi UML delle classi e da una versione modificata del diagramma delle attività. Per introdurre nuovi diagrammi sarà necessario introdurre nuove classi all'interno del package `Front-end::View` relative al nuovo diagramma introdotto, ai menù e le palette laterali che fanno parte dell'editor del diagramma. Per aggiungere i blocchi relativi ad un nuovo diagramma, sarà necessario creare un nuovo package all'interno del package `FrontEnd::Model::Objects`. Questo package conterrà tutti i blocchi che vanno a comporre il diagramma in questione. Diversamente, per aggiungere nuovi blocchi ad un diagramma già esistente, basterà creare le nuove classi all'interno del package del diagramma stesso, con gli opportuni metodi. In entrambi casi si avrà bisogno di creare o aggiornare i parser nel `Back-end` con le informazioni dei nuovi blocchi aggiunti.

A Glossario

A

Angular2
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
Con application programming interface (in acronimo API, in italiano interfaccia di programmazione di un'applicazione), si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per la realizzazione di un determinato compito all'interno di un certo programma.
Attività
Si riferisce al diagramma delle attività che è definito all'interno dello Unified Modeling Language (UML) che definisce le attività da svolgere per realizzare una data funzionalità.
AWS
Servizio di elaborazione serverless che esegue il codice dato dall'utente in risposta a determinati eventi e gestisce automaticamente le risorse di elaborazione in uso al posto dell'utente.

B

Back-end
Una applicazione back end è un programma con il quale l'utente interagisce indirettamente, in generale attraverso l'utilizzo di una applicazione front-end. In una struttura client/server il back-end è il server.
Browser
Particolare programma per navigare in Internet che inoltra la richiesta di un documento alla rete e ne consente la visualizzazione una volta arrivato.
BSON
Rappresentazione in formato binario di documenti Json. Estende il modello Json per fornire tipi di dati aggiuntivi, campi ordinati e per essere efficienti per la codifica e la decodifica in diverse lingue.

C

Callback
Funzione o "blocco di codice" che viene passata come parametro ad un'altra funzione. In particolare, quando ci si riferisce alla callback richiamata da una funzione, la callback viene passata come parametro alla funzione chiamante.
Capitolato
Documento tecnico, in genere allegato ad un contratto di appalto, che vi fa riferimento per definire in quella sede le specifiche tecniche delle opere che andranno ad eseguirsi

per effetto del contratto stesso, di cui è solamente parte integrante.

Chrome
Navigatore web sviluppato da Google, basato, a partire dalla versione 28, sul motore di esecuzione Blink (precedentemente sfruttava WebKit).

Classe
Costrutto di un linguaggio di programmazione usato come modello per creare oggetti. Il modello comprende attributi e metodi che saranno condivisi da tutti gli oggetti creati (istanze) a partire dalla classe.

Client
Componente che accede ai servizi o alle risorse di un'altra componente detta server.

Collection
Equivalenti alle tabelle dei database relazionali. In MongoDB rappresentano collezionidi documenti BSON.

CSS
Il Cascaded Style Sheet, è un linguaggio che permette di definire tutte le proprietà di stile e formattazione di una pagina web in maniera modulare, tenendo questa parte della progettazione di un sito separata da quella relativa al contenuto.

D

Data-Binding
Possibilità di collegare oggetti tra di loro in modo che uno rifletta i cambiamenti dell'altro.

Database
Archivio di dati strutturato in modo da razionalizzare la gestione e l'aggiornamento delle informazioni e da permettere lo svolgimento di ricerche complesse.

Dependency Injection
Tecnica con la quale si può attuare l'inversione del controllo. Essa prende il controllo su tutti gli aspetti di creazione degli oggetti e delle loro dipendenze.

Document-oriented
Riferito a basi di dati orientate ai documenti. Ovvero non memorizzano i dati in tabelle con campi uniformi per ogni record come nei database relazionali, ma ogni record è memorizzato come un documento che possiede determinate caratteristiche. Qualsiasi numero di campi con qualsiasi lunghezza può essere aggiunto al documento. I campi possono anche contenere pezzi multipli di dati.

Document
File che utilizza MongoDB con estensione .Bson.

DOM
(letteralmente modello ad oggetti del documento) Forma di rappresentazione dei documenti strutturati come modello orientato agli oggetti.

E

Event-driven
Termine associato a certi linguaggi di programmazione che fanno parte della programmazione ad eventi. Mentre in un programma tradizionale l'esecuzione delle istruzioni segue percorsi fissi, che si ramificano soltanto in punti ben determinati predefiniti dal programmatore, nei programmi scritti utilizzando la tecnica a eventi il flusso del programma è largamente determinato dal verificarsi di eventi esterni.

F

File
Viene utilizzato per riferirsi a un contenitore di informazioni/dati in formato digitale, tipicamente presenti su un supporto digitale di memorizzazione opportunamente formattato in un determinato file system.
Firefox
Web browser libero e multipiattaforma, mantenuto da Mozilla Foundation. È nato nel 2002 con il nome "Phoenix" dai membri della comunità Mozilla che volevano un browser stand-alone piuttosto che il raggruppamento Mozilla Application Suite. Gecko è il suo motore di rendering, supportando gran parte dei nuovi standard web oltre ad alcune caratteristiche che sono state progettate come estensioni a questi ultimi.
Framework
Architettura logica di supporto (spesso un'implementazione logica di un particolare design pattern) su cui un software può essere progettato e realizzato, spesso facilitandone lo sviluppo da parte del programmatore.
Front-end
Tale termine denota la parte visibile all'utente e con cui egli può interagire (interfaccia utente).

G

Google
Motore di ricerca per Internet il cui dominio è stato registrato il 15 settembre 1997. Successivamente, il 4 settembre 1998, è stata fondata la società Google Inc. Oltre a catalogare e indicizzare le risorse del World Wide Web, Google Search si occupa anche di foto, newsgroup, notizie, mappe (Google Maps), email (Gmail), shopping, traduzioni, video e programmi creati da Google Inc.

L

Libreria
Insieme di funzioni o strutture dati predisposte per essere connesse ad un programma software attraverso opportuno collegamento.
Linguaggio di Markup
Insieme di regole che descrivono i meccanismi di rappresentazione (strutturali, semantici o presentazionali) di un testo che, utilizzando convenzioni standardizzate, sono utilizza-

bili su più supporti.

M

Middleware
Insieme di programmi informatici che fungono da intermediari tra diverse applicazioni e componenti software. Sono spesso utilizzati come supporto per sistemi distribuiti complessi con architetture multitier. L'integrazione dei processi e dei servizi, residenti su sistemi con tecnologie e architetture diverse, è un'altra funzione delle applicazioni middleware.
MongoDB
DBMS non relazionale, orientato ai documenti. Classificato come un database di tipo NoSQL.
MVVM
Pattern software architetturale o schema di progettazione software.

- **Model**: rappresenta il punto di accesso ai dati. Trattasi di una o più classi che leggono dati dal DB, oppure da un servizio Web di qualsivoglia natura;
- **View**: rappresenta la vista dell'applicazione, l'interfaccia grafica che mostrerà i dati;
- **View model**: è il punto di incontro tra la View e il Model: i dati ricevuti da quest'ultimo sono elaborati per essere presentati e passati alla View.

Il fulcro del funzionamento di questo pattern è la creazione di un componente, il ViewModel appunto, che rappresenta tutte le informazioni e i comportamenti della corrispondente View. La View si limita infatti, a visualizzare graficamente quanto esposto dal ViewModel, a riflettere in esso i suoi cambi di stato oppure ad attivarne dei comportamenti.

N

Node.js
Piattaforma event-driven per il motore JavaScript V8, su piattaforme UNIX like.
NoSQL
Movimento che promuove sistemi software dove la persistenza dei dati è caratterizzata dal fatto di non utilizzare il modello relazionale, di solito usato dai database tradizionali.
Npm
Principale software utilizzato per maneggiare i moduli di Node.js e consente di condividere il codice per problemi tipici tra gli sviluppatori JavaScript.

O

Open-source
Software non protetto da copyright e liberamente modificabile dall'utente.

J

Java
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.
JavaScript
Linguaggio di Scripting lato-client, che viene interpretato dal browser.
Join
Operazione che realizza la congiunzione dell'algebra relazionale.
JSON
Formato adatto all'interscambio di dati fra applicazioni client-server.

P

Package
Nel linguaggio di programmazione Java, si tratta di un meccanismo per organizzare classi in gruppi logici, principalmente (ma non solo) in modo da definire namespace distinti per diversi contesti. Il package ha lo scopo di riunire classi (o entità analoghe, quali interfacce ed enumerazioni) logicamente correlate. Per esempio, le librerie standard Java sono organizzate in un sistema di package che comprende per esempio elementi strutturali del linguaggio, servizi di rete e così via.
Programmatore
Chi implementa una parte della soluzione dei progettisti.

Q

Query
Interrogazione di un database per estrarre o aggiornare i dati che soddisfano un certo criterio di ricerca.

R

Request
Richesta inviata da una componente ad un'altra.
Route
Blocchi di indirizzi IP contigui che compongono gli elementi delle tabelle di instradamento.
Routing
Instradamento effettuato a livello di rete. Nel caso tipico di IP, i router usano tabelle di instradamento i cui elementi sono blocchi di indirizzi IP contigui, che sono detti route o rotte. Questo metodo è pertanto più scalabile, in quanto un singolo elemento della tabella di instradamento può gestire un numero anche molto alto di host.

S

Safari
Applicazione e browser web sviluppato dalla Apple Inc. per i sistemi operativi iOS e macOS.

Server
Componente o sottosistema informatico di elaborazione e gestione del traffico di informazioni che fornisce, a livello logico e fisico, un qualunque tipo di servizio ad altre componenti (tipicamente chiamate clients, cioè clienti) che ne fanno richiesta attraverso una rete di computer, all'interno di un sistema informatico o anche direttamente in locale su un computer.

Snippet
Frammenti ed esempi di codice sorgente, di solito distribuiti nel pubblico dominio o come freeware.

SPA
Applicazione web o un sito web che può essere usato o consultato su una singola pagina web con l'obiettivo di fornire una esperienza utente più fluida e simile alle applicazioni desktop dei sistemi operativi tradizionali.

SVG
Linguaggio standardizzato dal W3C per la descrizione di immagini 2D composte da forme in grafica vettoriali, immagini bitmap e testo. SVG prevede diversi attributi per ogni oggetto, inerenti lo stile, la formattazione, l'animazione, etc.

Sviluppo
Il termine sviluppo (usato in ambito informatico) identifica una attività o una serie di attività mirate a costruire (sviluppare appunto) un programma. Lo sviluppo avviene scrivendo una serie di istruzioni (codice) in un determinato linguaggio di programmazione. Il processo di sviluppo si articola nelle seguenti fasi:

- Analisi: questa attività ha lo scopo di raccogliere tutti i requisiti che devono essere soddisfatti dal software da realizzare;
- Progettazione: questa attività ha lo scopo di definire il linguaggio di programmazione, le regole per la scrittura del codice e tutte le direttive tecniche che devono essere seguite dal team di sviluppo durante la fase di realizzazione;
- Realizzazione: questa è l'attività di scrittura vera e propria. In generale ciascun componente del team di sviluppo al termine della scrittura di una singola parte del software ne fa anche una verifica denominata "unit test". In questa fase sono comprese le "classiche" attività del programmatore: codifica, assemblaggio e compilazione (se il linguaggio non è del tipo interpretato), ed infine, memorizzazione dell'eseguibile in una determinata libreria o directory, definita in fase di progettazione;
- Test: tutte le parti del software vengono assemblate e testate insieme per verificare che ciò che è stato realizzato soddisfa esattamente i requisiti raccolti in fase di analisi;

- Rilascio: il programma viene pubblicato e quindi è pronto per essere installato.

T

Thread
Un thread o thread di esecuzione, in informatica, è una suddivisione di un processo in due o più filoni o sottoprocessi, che vengono eseguiti concorrentemente da un sistema di elaborazione monoprocessore (multithreading) o multiprocessore o multicore.
Three-tier
Particolare architettura software di tipo multi-tier per l'esecuzione di un'applicazione web che prevede la suddivisione dell'applicazione in tre diversi moduli o strati dedicati rispettivamente alla interfaccia utente, alla logica funzionale (business logic) e alla gestione dei dati persistenti.
Top-down
Modello in cui si formula inizialmente una visione generale del sistema ovvero se ne descrive la finalità principale senza scendere nel dettaglio delle sue parti. Ogni parte del sistema è successivamente rifinita aggiungendo maggiori dettagli della progettazione. Ogni nuova parte così ottenuta può quindi essere nuovamente rifinita, specificando ulteriori dettagli, finché la specifica completa è sufficientemente dettagliata da validare il modello.

U

UML
(Linguaggio di Modellazione Unificato) Linguaggio di modellazione e specifica basato sul paradigma orientato agli oggetti.
UNIX
Sistema operativo portabile per computer inizialmente sviluppato da un gruppo di ricerca dei laboratori AT & T e Bell Laboratories.

V

Validazione
Controllo effettuato sul software, per controllare se tutti i requisiti previsti sono stati coperti.
View
Visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti.

W

W3C
La parola W3C significa World Wide Web Consortium, cioè il Consorzio Internazionale che si occupa di definire linee guida e standard non proprietari, le tecnologie, i protocolli e tutto quanto necessario per lo sviluppo del Web.