ELABORATO DI APPLICAZIONI WEB DISTRIBUITE

PROGETTAZIONE E IMPLEMENTAZIONE DI UNA WEB APPLICATION PER LA GESTIONE DI UN BLOG

Dipartimento di Ingegneria Enzo Ferrari

Corso di Laurea Magistrale in Ingegneria Informatica

Elaborato di:
Gianluca Brilli

Prof. **Riccardo Lancellotti**

Anno Accademico 2017-2018

Indice

Introduzione

Il progetto presentato in questo documento si propone di creare una web application che permette la gestione di un **social blog**. Il servizio constente agli utenti registrati di creare un proprio blog personale, inerente ad una categoria definita in fase di creazione del blog. Tali utenti potranno poi aggiungere al proprio blog un numero arbitrario di post, marcati con dei tags. Gli utenti registrati possono inoltre personalizzare la propria pagina utente, andando ad inserire le proprie informazioni personali, quali username, email e nome, possono inoltre inserire una piccola descrizione personale.

Gli utenti registrati al sito possono poi commentare i post degli altri utenti e o rispondere ad eventuali commenti presenti. Questo documento è strutturato nel seguente modo:

1. *Capitolo 1 Descrizione delle Funzionalità*: descrizione delle principali attività che possono essere svolte all'interno della web application, in questa sezione si fanno a mostrare anche alcune immagini dell'applicazione stessa;

2. *Capitolo 2 Progettazione*: Si va a descrivere il design pattern (Model-View-Controller) che è stato utilizzato per progettare l'architettura dell'applicazione e si mostrano alcuni diagrammi UML che mostrano le fasi principali della progettazione della web application;

3. *Capitolo 3 Struttura del Progetto*: viene mostrato come è organizzata la directory principale del progetto;

4. *Capitolo 4 Tecnologie Utilizzate*: breve descrizione delle tecnologie che sono state utilizzate in questo progetto;

5. *Capitolo 5 Conclusione e Sviluppi Futuri*: in questo ultimo capitolo si andranno a riassumere i punti principali del progetto e si andranno a delineare i lavori che verranno svolti in futuro.

1. Descrizione delle Funzionalità

Nella presente sezione saranno dettagliate le funzionalità principali offerte dal servizio web sviluppato.

1.1 Sistema di Autenticazione

Come mostrato in Figura 1.1 l'applicazione mostra una pagina di login nella quale l'utente registrato può inserire le proprie credenziali per autenticarsi al sistema. Una volta immesso username e password l'utente sarà reindirizzato alla pagina principale dell'applicazione web.

In alternativa viene mostrato anche un pulsante di registrazione, per mezzo del quale un utente che ancora non possiede un account può crearsene uno, in modo tale da aver accesso ai servizi messi a disposizione dal sito web, come ad esempio creare un blog con al suo interno dei post o commentare i post degli altri blogger.

1.2 Homepage

La homepage mostrata in Figura 1.2 è chiaramente accessibile anche agli utenti non registrati e permette di visualizzare una situazione globale del sito web, è infatti possibile visualizzare i post più recenti ordinati per data di creazione con alcune informazioni al riguardo, quali ad esempio numero di commenti e visualizzazioni che ha raggiunto il determinato post.

Inoltre sulla destra è possibile vedere una lista dei commenti più recenti.

1.3 Profilo Utente

In Figura 1.3 viene mostrato come si presenta la pagina personale di un utente, in questo modo l'utente può condividere le informazioni che desidera col resto

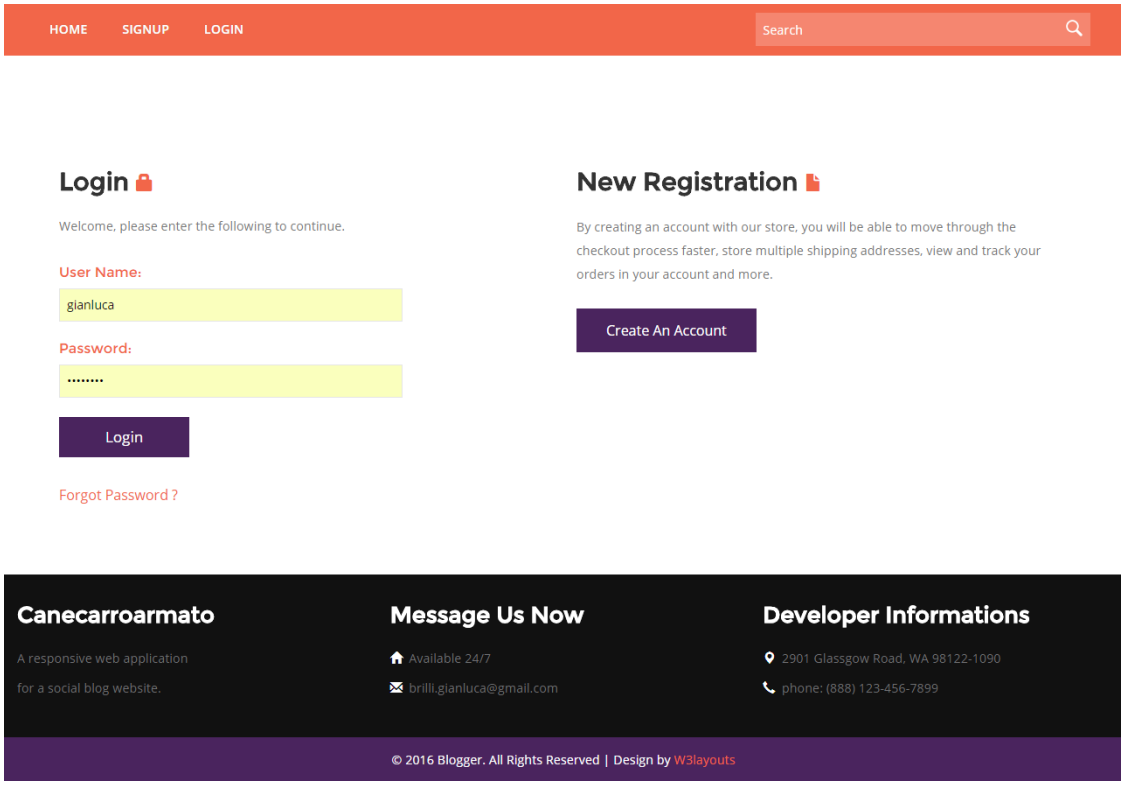


Figura 1.1: Autenticazione

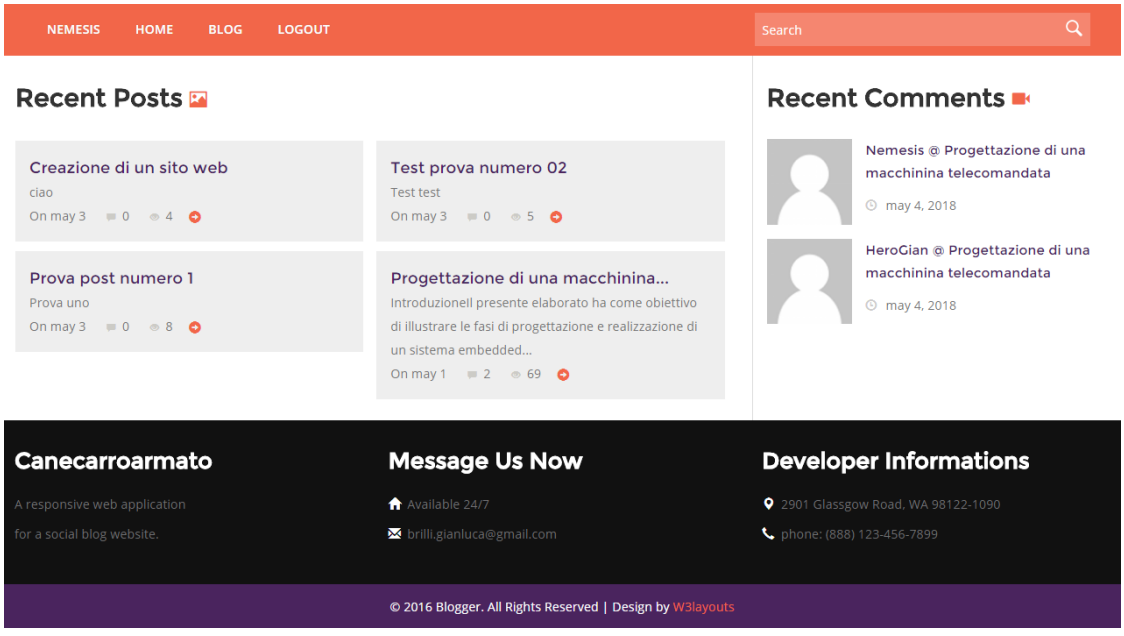


Figura 1.2: Homepage

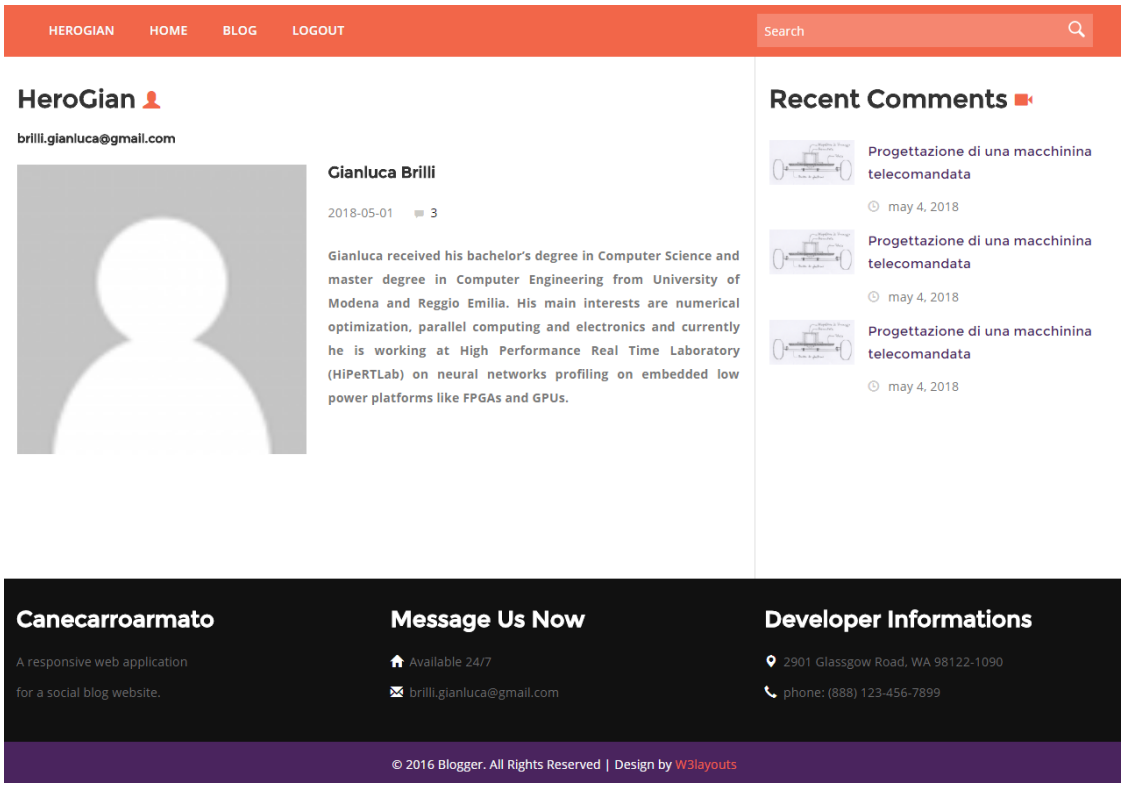


Figura 1.3: Profilo Utente

della community. Può inoltre inserire una breve descrizione per personalizzare ulteriormente la propria schermata.

E' inoltre presente una barra laterale che mostra alcuni commenti ordinati per data di pubblicazione dell'utente in oggetto.

1.4 Creazione di Blog e Post

Un utente registrato può inoltre decidere di creare un **blog personale**, assegnargli un **titolo** e indicare una **categoria** di riferimento che andrà ad accomunare i **post** che andrà successivamente a creare.

Ad esempio, con riferimento all'account di prova creato (*HeroGian*), gli è stato attribuito un blog dal titolo *Progetti Elettronica e Informatica* appartenente alla categoria *Science & Mathematics*, all'interno di questo blog è poi stato creato un post di prova intitolato *Progettazione di una macchinina telecomandata basata su microcontrollore STM32* ed a tale post sono stati attribuiti i **tags** *Elettronica, Informatica e STM32*.

La creazione effettiva di un post, avviene per mezzo della finestra modale riportata in Figura 1.4; particolare attenzione merita la parte inferiore della finestra, dove è presente un piccolo editor, grazie al quale l'utente può attribuire

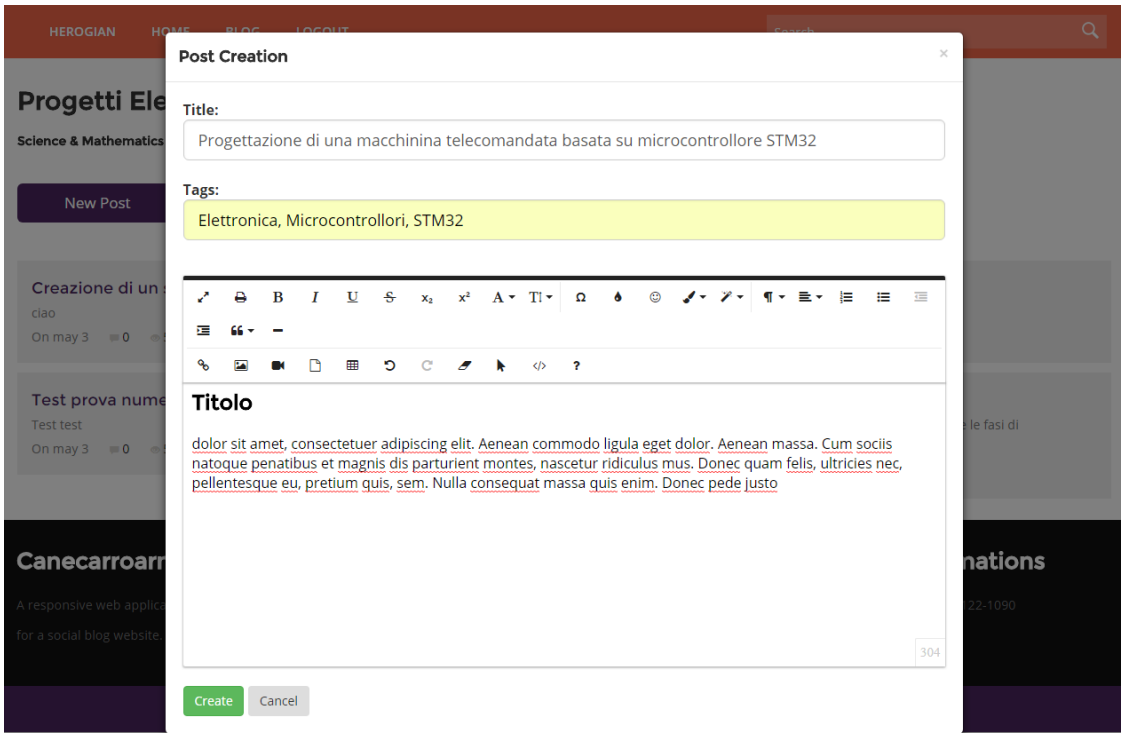


Figura 1.4: Creazione di un Post

uno stile al corpo del post che intende creare, può ad esempio inserire titoli, sottotitoli, immagini, tabelle, elenchi puntati e quant'altro.

Ultimata la creazione di un post, il blogger viene poi reindirizzato alla pagina che mostra la propria creazione, oltre al corpo del post è poi possibile visionare alcune statistiche al riguardo, quali ad esempio numero di commenti, data di creazione e numero totale di visualizzazioni che il post ha effettuato. Ultimo ma non meno importante è la barra laterale presente nella pagina di un post, essa mostra tutte i tag che sono stati attribuiti al post in oggetto ed una lista di post aventi il medesimo tag. In questo modo l'utente che segue post di un determinato blogger, può trovare utile e interessante visionare posts di altri bloggers che trattano il medesimo argomento.

1.5 Creazione di Commenti

Questa funzionalità assieme alla creazione dei post è una delle più importanti, in quanto permette un contatto tra un blogger e i propri followers. E' infatti possibile (per gli utenti registrati) andare a commentare un post direttamente nella pagina del post stesso; è inoltre possibile rispondere ad un commento, come viene mostrato in Figura 1.5.

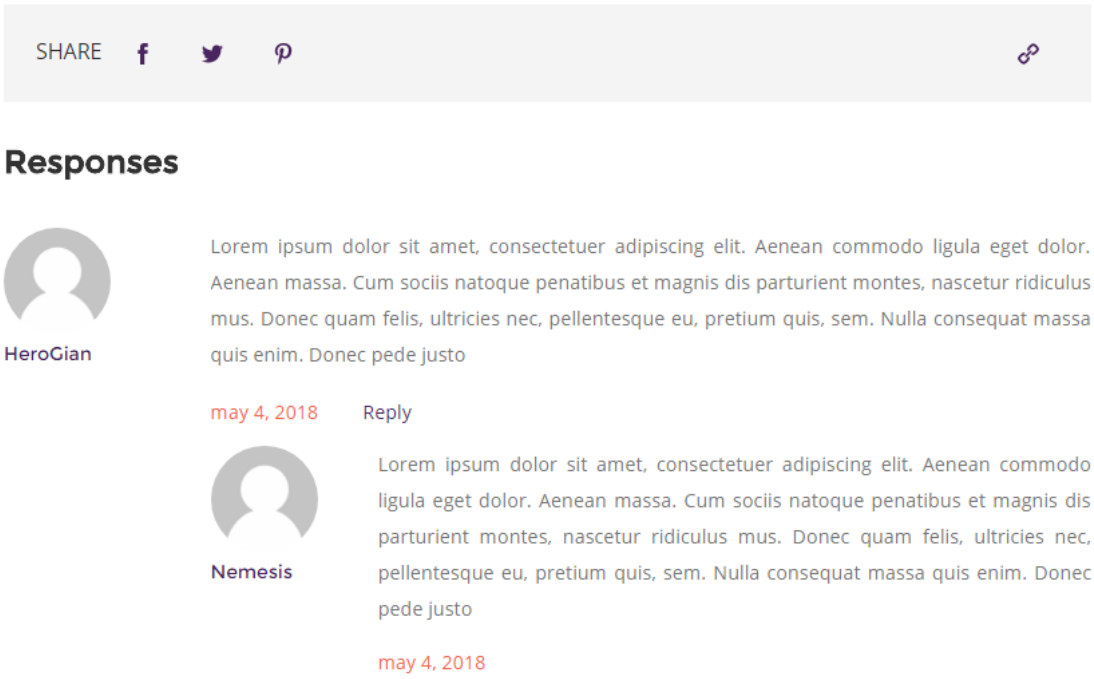


Figura 1.5: Commenti e Risposte a Commenti

1.6 Ricerca

L'ultima macro funzionalità descritta in questo documento è la funzione che permette di effettuare delle ricerche all'interno della web application. E' infatti possibile ricercare un post per mezzo della barra di ricerca posizionata in alto a destra, è necessario semplicemente scrivere ciò che si intende cercare ed il sistema effettuerà subito una ricerca all'interno del proprio database, in base al cambiamento della sequenza di caratteri digitati sulla barra di ricerca stessa, come viene mostrato in Figura 1.6.

Andando poi a cliccare sulla lente di ingrandimento o premendo invio, si viene reindirizzati alla pagina di ricerca principale, nella quale è possibile vedere più in grande le proprie ricerche.

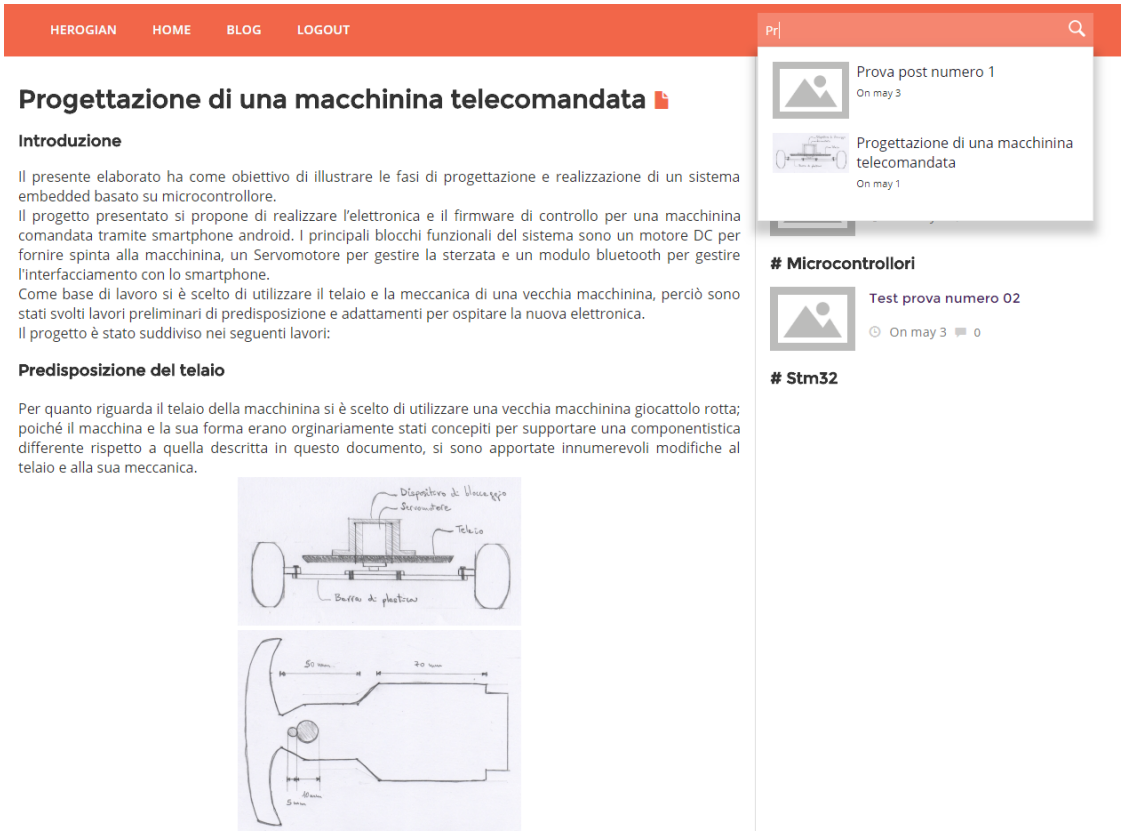


Figura 1.6: Dettaglio Post e Ricerca

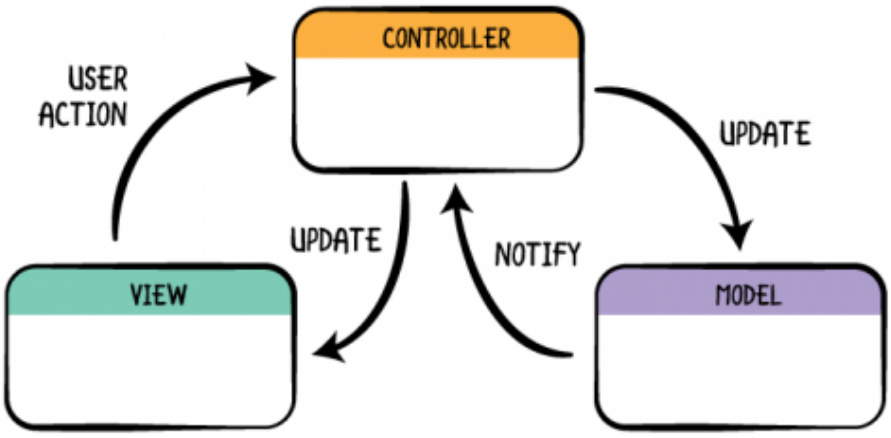
2. Progettazione

Al fine di rendere l'applicazione descritta il più modulare e scalabile possibile, si è deciso di progettare la sua architettura software tramite il design pattern **Model-View-Controller**, come descritto meglio nella seguente sezione.

2.1 Paradigma Model-View-Controller

Model-view-controller (MVC) è un **design pattern** architetturale molto diffuso nello sviluppo di sistemi software, in particolare nell'ambito della programmazione orientata agli oggetti, in grado di separare la presentation logic dalla business logic. Il pattern è basato sulla separazione dei compiti fra i componenti software che interpretano tre ruoli principali:

- **Model**: fornisce i metodi per accedere ai dati utili all'applicazione. Nell'applicazione sviluppata il model è rappresentato dalle classi contenute nel package *ml.canecarroarmato.model*;
- **View**: visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti. In questo caso le views sono rappresentate dai JSP contenuti nella directory *src/main/webapp/WEB-INF/views*;



- **Controller**: riceve i comandi dell'utente (in genere attraverso il view) e li attua modificando lo stato degli altri due componenti. Il controller è contenuto nel package *ml.canecarroarmato.controller*.

2.2 Diagramma delle Classi

Per mezzo del diagramma delle classi è mostrato nel dettaglio la struttura delle entità che modellano i dati all'interno dell'applicazione; saranno infine dettagliati gli attributi e i metodi più importanti delle classi presentate (per semplicità sono omessi dal diagramma i metodi get e set).

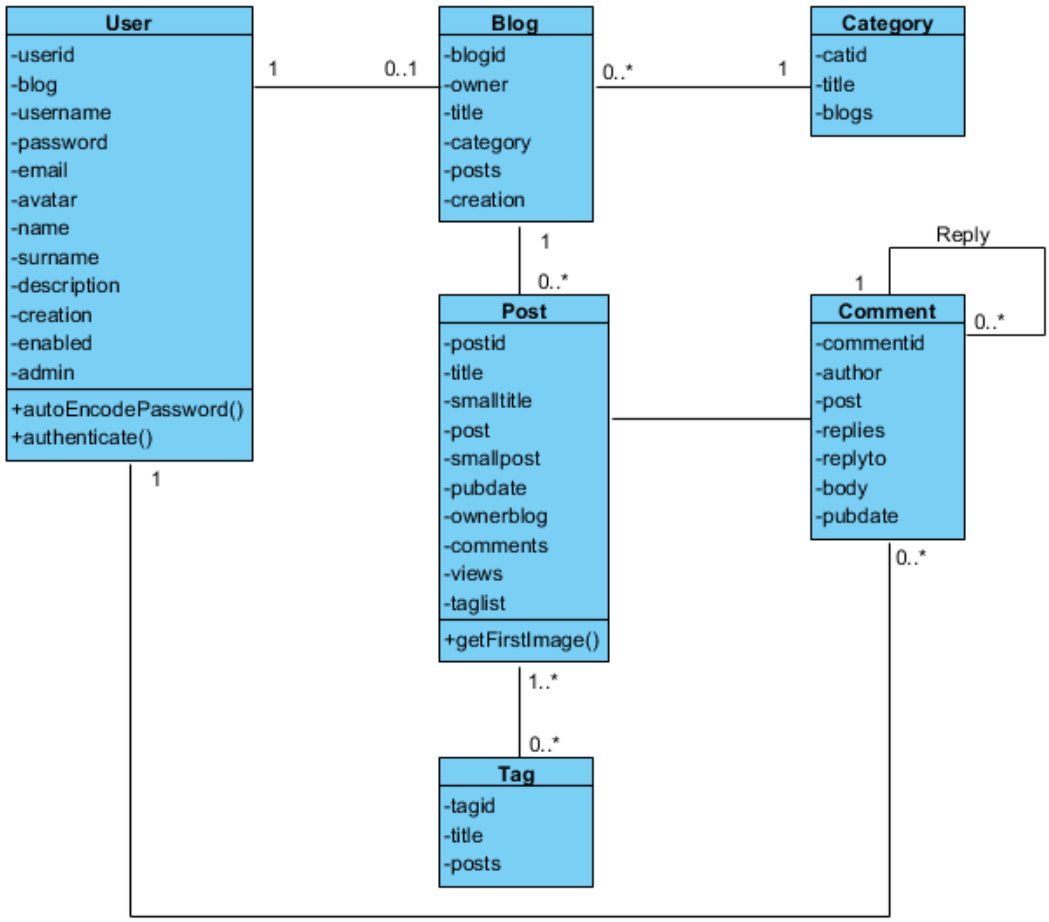


Figura 2.1: Diagramma delle Classi

I principali attributi utilizzati nelle entità modellate all'interno dell'applicazione sono listati di seguito, gli attributi non descritti qui di seguito sono autoesplicativi e non necessitano di ulteriori descrizioni.

Classe User

- *userid (Int)*: chiave primaria dell'entità;
- *blog (Blog)*: riferimento al blog creato, se ne possiede uno;
- *avatar (String)*: path relativo all'immagine personale;
- *enabled (Boolean)*: indica se l'utente è abilitato o meno a svolgere le proprie funzioni sull'applicazione (per usi futuri).
- *admin (Boolean)*: indica se l'utente è amministratore dell'applicazione o se è un utente normale (per usi futuri).

Classe Blog

- *blogid (Int)*: chiave primaria dell'entità.
- *owner (User)*: utente proprietario del blog.
- *category (Category)*: categoria di appartenenza del blog.
- *posts (List<Post>)*: lista di post che l'utente ha creato nel suo blog.

Classe Post

- *postid (Int)*: chiave primaria dell'entità.
- *smalltitle (String)*: sottotitolo ottenuto rimuovendo caratteri dal titolo originale.
- *post (String)*: corpo del messaggio, può includere testo normale, immagini, link, elenchi puntati, tabelle e molti altri elementi di stile.
- *smallpost (String)*: versione del corpo del messaggio avente un numero ridotto di caratteri e nella quale sono rimossi tutti gli elementi di stile.
- *ownerblog (Blog)*: blog di appartenenza del post.
- *comments (List<Comment>)*: lista di commenti al dato post.
- *views (Int)*: numero che indica quante volte è stato visto questo post.
- *taglist (List<Tag>)*: lista di tag con la quale il blogger ha marcato il proprio post.

Classe Comment

- *commentid (Int)*: chiave primaria dell'entità.
- *author (User)*: utente registrato che ha creato il commento.
- *post (Post)*: post di riferimento per il commento.
- *replies (List<Comment>)*: lista di commenti di risposta a questo commento.
- *replyto (Comment)*: commento al quale il commento in oggetto è risposta.
- *body (String)*: corpo del messaggio che compone il commento.

Infine trascurando i metodi get/set associati a ciascun attributo, i metodi più di interesse sono i seguenti:

Classe User

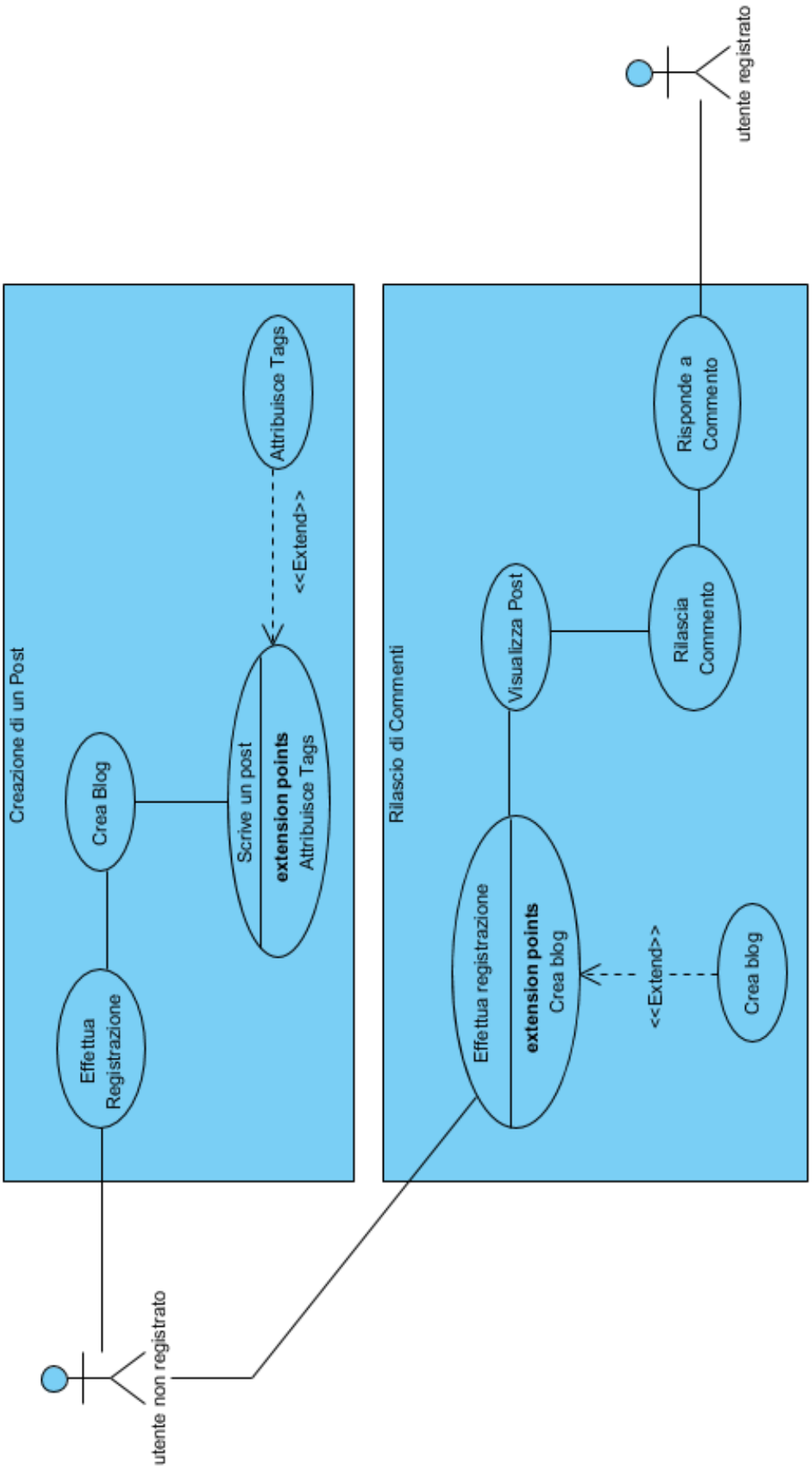
- *autoEncodePassword()*: applica una **funzione hash** all'attributo password dell'entità, in modo da rendere più sicuro il salvataggio dell'oggetto su memoria persistente.
- *authenticate(String password)*: controlla se la password passata come argomento è corretta ed in caso affermativo effettua l'autenticazione dell'utente.

Classe Post

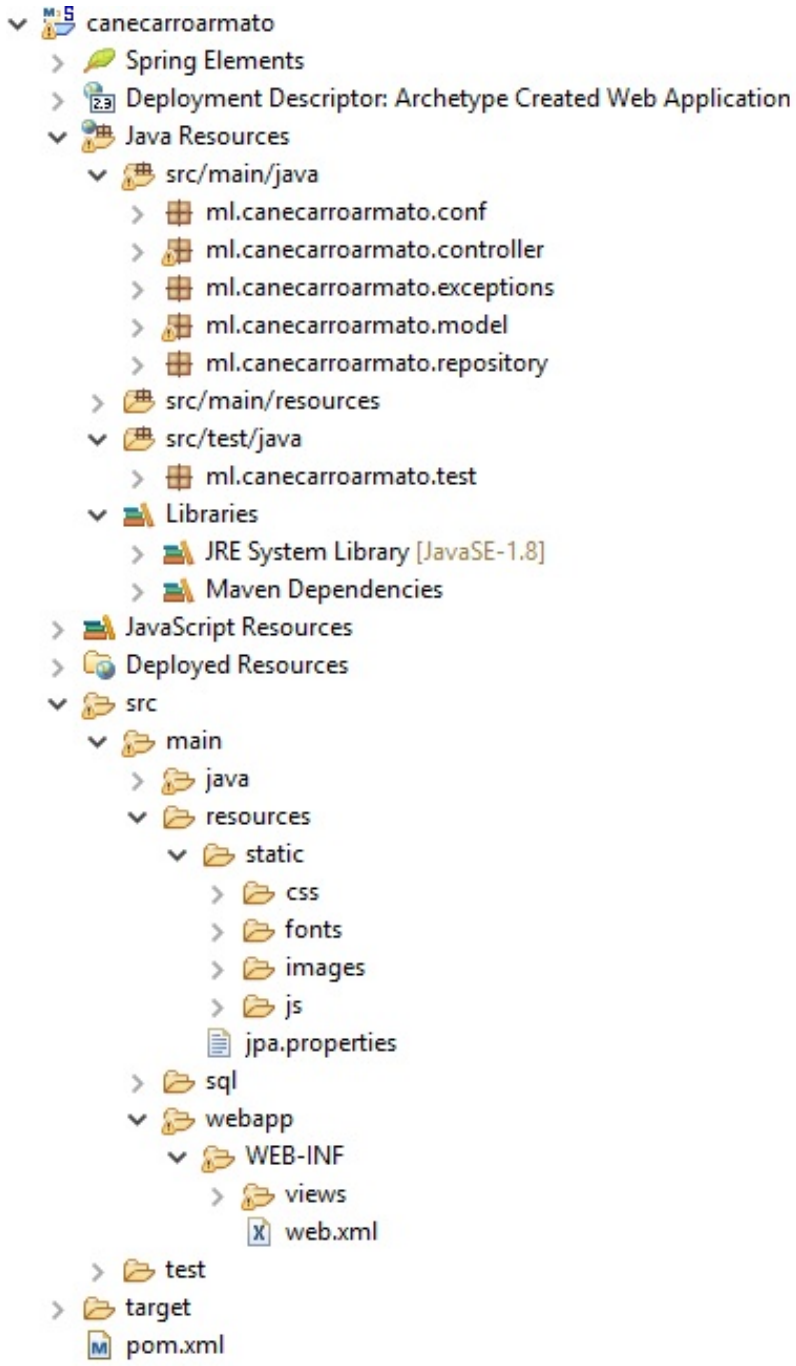
- *getFirstImage()*: va ad effettuare il parsing del corpo del post al fine di determinare se è presente o meno un'immagine, in caso affermativo va a ritornare l'url di tale immagine, mentre in caso contrario ritorna l'indirizzo di un'immagine di default salvata sul server. Ciò è utile per associare ad ogni post un'immagine che lo rappresenta.

2.3 Diagramma dei Casi d'Uso

Di seguito viene infine mostrato un diagramma che va a mostrare le interazioni tra gli attori che prendono parte al sistema ed il sistema stesso. In particolar modo viene preso in esame l'inserimento e la creazione di un post da parte di un generico utente non registrato al sistema e la creazione di un messaggio di risposta ad un post o ad un commento da parte di un utente non registrato.


utente registrato
Creazione di un Post
Attribuisce Tags
<<Extend>>
Crea Blog
Scrive un post
extension points
Attribuisce Tags
Effettua Registrazione
Rilascio di Commenti
Risponde a Commento
Visualizza Post
Rilascia Commento
Effettua registrazione
extension points
Crea blog
<<Extend>>
Crea blog
utente non registrato

3. Struttura del Progetto



La struttura del progetto è riportata in figura nella precedente pagina, i principali package contenuti nella directory del progetto sono i seguenti:

- *ml.canecarroarmato.conf*: in questo package sono contenute le classi Java per la configurazione del framework Spring.
- *ml.canecarroarmato.controller*: contiene le classi che si occupano di implementare i controller, ovvero quelle entità che restano in ascolto sugli url della web application e realizzano la business logic.
- *ml.canecarroarmato.exceptions*: contiene alcune classi che implementano delle eccezioni customizzate per l'applicazione.
- *ml.canecarroarmato.model*: contiene classi che implementano le entità di cui l'applicazione si compone, per mezzo di JPA (descritto in seguito) tali classi sono mappate sul database.
- *ml.canecarroarmato.repository*: sono interfaccie che ereditano da una classe padre chiamata *CrudRepository*, permettono di implementare operazioni di **Create, Read, Update e Delete (CRUD)** o di definire query personalizzate per mezzo del linguaggio HQL (descritto in seguito).
- *ml.canecarroarmato.test*: in questo package sono contenuti i vari test cases e test suites utilizzati per effettuare unit testing.

Altre directory degne di nota sono le seguenti:

- *src/main/resources/static/css*: contiene i fogli di stile (CSS) utilizzati dall'applicazione.
- *src/main/resources/static/fonts*: contiene alcuni fonts particolari che vengono impiegati all'interno del template utilizzato.
- *src/main/resources/static/images*: sono contenute le immagini di cui il template si compone, richiamate dall'html o dai css.
- *src/main/resources/static/js*: contiene i file JavaScript utilizzati per effetti grafici vari o per chiamate AJAX.
- *src/main/resources/sql*: contiene il codice SQL necessario alla creazione delle tabelle del database e dei dati di esempio per popolare la tabella Categories.

Per finire altre directory che meritano di essere menzionate in questa breve descrizione sono le seguenti:

- *src/main/webapp/WEB-INF/views*: contiene i file JSP che vengono impiegati come views per la web application sviluppata.

3.1 Maven: Gestore di Progetto

Apache Maven è uno strumento di gestione di progetti software basati su Java e build automation e per quanto riguarda le funzionalità è similare ad Apache Ant. Maven usa un costrutto conosciuto come **Project Object Model (POM)**; un file XML che descrive le dipendenze fra il progetto e le varie versioni di librerie necessarie nonché le dipendenze fra di esse. Maven effettua automaticamente il download di librerie Java e plug-in Maven dai vari repository definiti scaricandoli in locale o in un repository centralizzato lato sviluppo.

I file e le directory utilizzate da Maven sono:

- */pom.xml*: contiene le dipendenze e le librerie esterne utilizzate all'interno del progetto.
- */Java Resources/Libraries/Maven Dependancies*: directory in cui sono automaticamente scaricati i file JAR necessari al progetto e dichiarati nel pom.xml

Le dipendenze dichiarate all'interno del pom.xml e necessarie al corretto funzionamento della web application sono le seguenti:

- *junit, mockito-all e spring-test*: utilizzate per svolgere unit testing e mocking degli oggetti da testare;
- *javax.servlet-api*: api che permette l'utilizzo di alcuni metodi legati alle servlet.
- *spring-web* e *spring-webmvc*: dipendenze per il corretto funzionamento del framework Spring;
- *javax.servlet.jsp-api* e *jsp-template-inheritance*: dipendenze che permettono l'utilizzo della tecnologia Java Server Pages e per svolgere template inheritance;

- *javax.servlet.jsp.jstl-api e jstl*: api che permettono di utilizzare i vari tags messi a disposizione dalla tecnologia JSTL;
- *spring-data-jpa, hibernate-core, hibernate-entitymanager e hibernate-validator*: permettono il corretto funzionamento della Java Persistence, definita per mezzo della JPA e implementata da Hibernate; hibernate validator per implementare la validazione delle entità;
- *postgresql*: driver JDBC che permette la connessione della web application col database PostgreSQL;
- *spring-security-crypto*: permette l'utilizzo di funzioni hash, utilizzate per cifrare le password;
- *jackson-databind*: utilizzata per convertire un oggetto in una stringa JSON, viene utilizzata nei controller che gestiscono le chiamate AJAX.
- *json*: permette di manipolare oggetti JSON;
- *jsoup*: per parsing di stringhe HTML al fine di evitare attacchi di code injection;
- *imgscalr-lib*: libreria che permette la manipolazione delle immagini, utilizzata per scalare la dimensione delle immagini inserite dagli utenti;
- *commons-fileupload*: utilizzata per il caricamento di immagini sul server.

4. Tecnologie Utilizzate

4.1 Java 2 Enterprise Edition (J2EE)

La tecnologia Java 2 Enterprise Edition (J2EE) è la tecnologia di riferimento adottata nel progetto qui descritto, essa vanta di notevoli vantaggi, dal momento che è diventata negli anni sinonimo di sviluppo di applicazioni aziendali robuste, sicure ed efficienti. Queste caratteristiche la rendono tra le più importanti piattaforme tecnologiche di sviluppo, soprattutto in ambiti in cui la sicurezza e la robustezza sono vincoli imprescindibili (ad esempio applicazioni bancarie).

4.2 Presentation Logic

A questo livello viene definita la rappresentazione dei dati e l'interfaccia utente. È qui che usiamo in modo importante pagine e controlli Web e i dati che rappresentano sono forniti dai livelli inferiori.

4.2.1 Bootstrap

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

Per questo progetto si è deciso di utilizzare il template bootstrap *Blogger* fornito nel repository di template **W3layouts**, in questo modo si è incrementata la produttività nell'implementazione della web application, e si è evitato il tedioso lavoro di progettazione dell'interfaccia grafica.

4.2.2 JavaScript

In informatica 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.

Nel presente elaborato si è deciso di utilizzare la libreria **Jquery**, al fine di semplificare e velocizzare l'interazione e la manipolazione del **Document Object Model (DOM)**.

4.2.3 AJAX

AJAX, acronimo di **Asynchronous JavaScript and XML**, è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive (Rich Internet Application). 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.

Per la gestione dell'interazione asincrona per mezzo di AJAX si è deciso anche in questo caso di affidarsi alla libreria Jquery. In particolar modo all'interno di questo progetto si è deciso di utilizzare AJAX per effettuare richieste **GET** asincrone per la funzionalità di ricerca all'interno del database e per **POST** asincrone per l'inserimento di commenti e risposte ad un post. In questo modo si aumenta la reattività del sito e si va a migliorare l'esperienza utente.

4.2.4 Java Server Pages (JSP)

Una pagina JSP è un file che rappresenta una pagina Web di contenuto parzialmente o totalmente dinamico. Elaborando la pagina JSP, il motore JSP produce dinamicamente la pagina HTML finale che verrà presentata al web browser dell'utente. La pagina JSP può contenere tre tipi di elementi, a cui corrispondono tre diversi modi di elaborazione: contenuti statici, direttive e script.

Nel progetto qui descritto si è scelto di utilizzare la tecnologia JSP per implementare le views presentate all'utente, secondo il paradigma Model-View-Controller, come verrà dettagliato in seguito.

4.2.5 Java Standard Tag Library (JSTL)

JSTL è una libreria inclusa come componente della piattaforma software di sviluppo per applicazioni Web **Java EE**. È un'estensione di JSP ed incorpora un insieme di tag HTML definiti tramite file XML e programmati in linguaggio Java. Essa permette di interagire con gli oggetti passati alle views per mezzo

dei costrutti classici della programmazione, quali ad esempio loops, costrutti condizionali ecc.

4.3 Business Logic

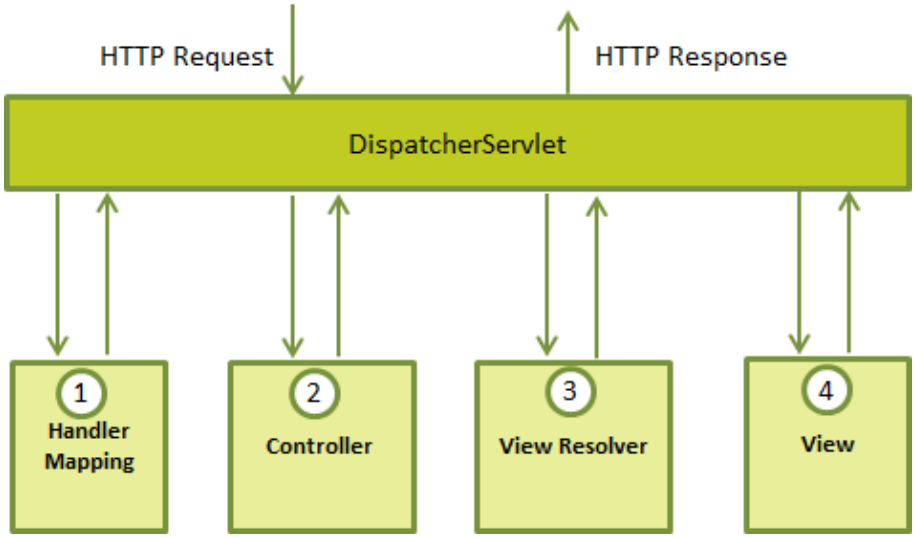
È il livello di descrizione delle entità logiche che descrivono i processi di business utilizzati all'interno dell'applicazione. Tali entità sono delle classi vere e proprie, con attributi e metodi utili a rappresentare la logica applicativa del proprio sito Web.

4.3.1 Spring MVC

Spring MVC è un web-framework fornito da Spring e basato sul design pattern **Model-View-Controller (MVC)**. Questo framework è creato sulla base di **JEE/Servlet** ed è di tipo **request-driven**. Questo significa che alcune Servlets ascoltano alcune porte, in attesa di richieste in entrata, ciò permette di fornire la particolare risorsa o dato che la richiesto.

In Spring MVC sono presenti componenti chiamati **DispatcherServlet** e **Controllers**, essi agiscono da Controller (inteso come Controller del design pattern MVC). Essi ricevono le richieste e decidono come esse saranno servite e che view presentare all'utente. Il Model in Spring MVC è rappresentato da particolari oggetti chiamati **Business objects** o **domain objects** e rappresentano il Model dell'applicazione, tali oggetti vengono invocati dal Controller e saranno poi passati ai JSP che agiranno da Views. Le principali componenti di cui Spring è formato sono le seguenti:

- **DispatcherServlet**: è il front controller di Spring MVC. Essa è una **HttpServlet** che si occupa di ricevere le richieste e ritornare le relative risposte.
- **MappingHandler**: un lavoro importante che deve essere svolto prima che la DispatcherServlet possa direzionare la richiesta al Controller, è determi-



nare a quale Controller reindirizzare la richiesta. La DispatcherServlet di Spring si affida ad un oggetto chiamato MappingHandler.

- **Controller**: si occupa di svolgere l'effettivo lavoro di processamento della richiesta, e rappresenta chiaramente il componente dove il programmatore svolgerà la maggior parte del proprio lavoro.
- **ViewResolver**: è il componente che si occupa di recuperare la particolare view da servire all'utente, in base a quanto richiesto dal Controller.
- **View**: rappresentate dai vari file JSP (che vengono compilati in HTML) e da eventuali classi per l'esportazione in formati diversi da HTML (PDF, XLS, CSV, ...).

4.4 Data Logic

È il livello di interazione con la base di dati legata al sito, dove si effettua fisicamente la connessione e si eseguono query di selezione, inserimento, aggiornamento o cancellazione.

4.4.1 Java Persistence API (JPA)

Con JEE 5 è stata definita anche la specifica per la persistenza dei dati con il nome di Java Persistence API. JPA si pone come livello di astrazione al di sopra delle API JDBC, rendendo il codice per la gestione dei dati **indipendente dal database** ed introducendo il linguaggio SQL-Like, che consente di effettuare interrogazioni sul modello Java e non direttamente sulla base dati.

Il mattone base è costituito dal concetto di Entity, essa non è altro che un **Plain Old Java Object (POJO)** da mappare su una o più tabelle del Database, in modo che i valori delle variabili di istanza dei suoi oggetti vengano salvati come record.

4.4.2 Hibernate

Hibernate ORM (H8) è una piattaforma che fornisce attraverso il proprio framework, un servizio di **Object relational mapping (ORM)**. In generale l'ORM è una tecnica di programmazione che permette di "mappare" oggetti (di un qualsiasi linguaggio Object-Oriented) su un Database Relazionale. Attraverso l'utilizzo di un framework come Hibernate lo sviluppatore non dovrà più preoccuparsi di gestire la persistenza dei dati e di conseguenza le procedure per le operazioni **CRUD** dei database (Create, Read, Update, Delete) saranno molto più semplici. Hibernate fornisce il mapping tra le classi Java e le tabelle di un database relazione e attraverso di esso viene gestito il salvataggio e la lettura dei dati. Hibernate definisce anche un linguaggio per interrogazioni orientato non più alle tabelle (come SQL) ma alle entità definite nel model, chiamato **Hibernate Query Language (HQL)**.

4.4.3 Java Database Connectivity (JDBC)

JDBC è un connettore (driver) per database che consente l'accesso e la gestione della persistenza dei dati sulle basi di dati da qualsiasi programma scritto con il linguaggio di programmazione Java, indipendentemente dal tipo di DBMS utilizzato. È costituito da un'API object oriented orientata ai database relazionali, raggruppata nel package java.sql, che serve ai client per connettersi a un database fornendo i metodi per interrogare e modificare i dati.

4.4.4 Database Management System (DBMS)

Per la gestione della persistenza dei dati si è deciso di affidarsi al DBMS **PostgreSQL** che rappresenta una reale alternativa sia rispetto ad altri prodotti liberi come MySQL, Firebird SQL e MaxDB che a quelli a codice chiuso come Oracle, IBM Informix o DB2 ed offre caratteristiche uniche nel suo genere che lo pongono per alcuni aspetti all'avanguardia nel settore dei database.

4.5 Unit Testing

Con Unit Test si intende un frammento di codice scritto per eseguire una specifica funzionalità e testarla, vengono considerate tipicamente funzioni piccole e/o metodi di classi. Le tipologie di Unit Testing sono innumerevoli, esistono ad esempio i *Performance Tests*, i *Behavioral Tests* e gli *State Tests*.

4.5.1 JUnit

JUnit è lo strumento principale in ambiente Java per eseguire Unit Tests, in questo caso si utilizzano classi esplicitamente dedicate a contenere i tests, in cui ogni metodo della classe creata rappresenta un test. All'interno dei metodi si fa uso si **assertions**, ovvero verifiche che una data condizione sia verificata.

In questo progetto, durante il suo ciclo di sviluppo si è fatto uso di tre tipologie di test:

- **Model Testing**: si sono testati alcuni dei metodi più critici delle classi che rappresentano le entità del database.
- **Controller Testing**: include tests svolti sui controller dell'applicazione, si è cercato di simulare le interazioni HTTP con i controller e tramite delle assertions, si è verificato il corretto funzionamento.
- **Database Testing**: includono test riguardanti il salvataggio e la gestione della persistenza delle entità utilizzate.

I singoli test sono stati poi raggruppati in tre **Test Suites** in base alla categoria di appartenenza ed una Suite generica che raggruppa ogni test e che permette la loro esecuzione in batch.

4.5.2 Mockito e MVCMock

Mockito e MVCMock sono rispettivamente due framework che permettono la creazione e la gestione di **Mock Objects**; questi framework sono utili se si intende

svolgere dei test più approfonditi che richiedono per esempio interazioni con entità più complesse quali ad esempio database, HttpServlets ecc. In questo caso vengono create delle *entità fittizie* o Mock, che permettono di svolgere il test desiderato.

Nel progetto presentato si sono utilizzati degli oggetti mock per il testing della Java Persistence API e delle operazioni CRUD sul database, in questo modo si sono potute testare le funzionalità senza andare a *sporcare* il database reale.

Si è poi fatto uso di oggetti mock HTTP (per mezzo di MVCMock) quali ad esempio HTTPServletRequests o Controllers della applicazione al fine di testare anche la risposta dei controllers alle richieste dell'utente.

4.6 Deployment

Questa sezione mostra brevemente come è avvenuta la fase di deployment (migrazione dal server di sviluppo) al server di produzione.

4.6.1 Macchina Server

Macchina server casalinga disponibile 24/7, con sistema operativo **Debian 9** con un **Web Application Server** in ascolto sulla **porta 8080**.

E' stato inoltre necessario configurare un corretto port forwarding sul router della rete LAN, in modo tale da permettere il redirezionamento del traffico HTTP alla macchina in oggetto. E' stato realizzato per mezzo del seguente comando **iptables** sul router:

```
1  # iptables -t nat -A PREROUTING -p tcp --dport 8080 -j DNAT
     --to-destination 192.168.1.254:8080
```

4.6.2 Hostname

Per la gestione dell'hostname si è deciso di utilizzare un servizio di **DNS Dinamico**, al fine di andare a mascherare l'indirizzo IP variabile offerto dal provider internet. In questo caso ci si è affidati a **Dynu** per la gestione del DNS Dinamico ed a **Freenom** per la creazione dell'hostname `canecarroarmato.ml`, per mezzo del **Top Level Domain** del Mali; si è deciso di procedere in questo modo in quanto il tutto è completamente gratuito.

4.6.3 Web Application Server

Come Web Application Server si è deciso di utilizzare **Apache Tomcat** che non è altro che un web server che agisce da **contenitore di Servlet** open source sviluppato dalla **Apache Software Foundation**. Implementa le specifiche Java Server Pages (JSP) e Servlet, fornendo quindi una piattaforma software per l'esecuzione di applicazioni Web sviluppate in linguaggio Java. La sua distribuzione standard include anche le funzionalità di web server tradizionale, che corrispondono al prodotto Apache.

La configurazione del server Tomcat utilizzata è molto basilare, per prima cosa si è creato un utente abilitato per la pagina di amministrazione di Tomcat ovvero: *localhost:8080/manager*, per mezzo del file *tomcat-users.xml* nella quale è stato creato un utente amministratore:

```
1   <role rolename="manager-gui"/>
2   <role rolename="manager-script"/>
3   <user username="gian" password="segreto" roles="manager-gui,
       manager-script" />
```

Dopo la creazione dell'utente, è stato svolto il deployment caricando sul server il progetto esportato in formato **Web application ARrchive (WAR)**.

5. Conclusione e Sviluppi Futuri

L'applicazione sviluppata e qui descritta presenta tutti i requisiti che sono stati definiti in fase di progetto e possiede le funzionalità principali necessarie ad un servizio di blogging, quali ad esempio: creazione di utenti, blog e post e possibilità di risposta ai post creati dai blogger.

Inoltre l'applicazione, nella sua versione beta, è già raggiungibile online all'indirizzo `http://canecarroarmato.ml:8080`

Si prevede in futuro di proseguire lo sviluppo del progetto ed esentedere le sue funzionalità, in particolare si valuterà l'integrazione delle seguenti funzionalità:

- suddivisione degli elementi dinamici presenti nella schermata principale in pagine numerate, che saranno gestite poi per mezzo di richieste GET asincrone;
- possibilità di modifica della descrizione utente e dei post creati dai bloggers;
- qualche sistema di personalizzazione del proprio blog;
- gestione di utenti abilitati e disabilitati, per mezzo di email di conferma al momento della iscrizione;
- implementazione di funzioni di amministrazione e moderazione dei contenuti presenti sul blog, si valuta anche l'integrazione di una pagina di amministrazione utile a controllare meglio i contenuti creati all'interno dell'applicazione;
- ottimizzazione di tutte le pagine del sito per dispositivi mobili e miglioramenti grafici.