Università Degli Studi di Modena e Reggio Emilia

Dipartimento di Ingegneria “Enzo Ferrari”

Corso di Laurea in Ingegneria Informatica

Struttura generica di bot: il Qbot

Relatori:
Prof.ssa Laura Po
Prof Domenico Beneventano

Candidato:
Marianna Ferrari

Anno Accademico 2016-2017

Indice

Indice delle figure

1. Abstract

Questa tesi vuole parlare di una struttura generica di bot.

L'argomento in questione, è stato studiato e sviluppato presso l'azienda Quix srl di Appalto di Soliera, durante il tirocinio universitario previsto dal mio piano degli studi.

Il progetto è stato scritto, quasi interamente, in Java, utilizzando le tecnologie (vari framework, IDE, linguaggi di supporto …) che verranno brevemente riassunte nei prossimi capitoli.

L'elaborato si compone di una panoramica sulla nascita dei bot, il loro possibile sviluppo futuro, gli strumenti che mi hanno permesso, con il supporto del mio tutor, di poter arrivare al risultato finale, e infine, una descrizione dell'applicazione in questione, il cui nome è Qbot.

Inizialmente pensato come applicativo per un caso reale (applicazione specifica per un'azienda del territorio emiliano), questo progetto, si è poi trasformato in un prodotto aziendale che può essere specializzato per qualsiasi richiesta futura.

Per quanto riguarda l'applicazione di messaggistica scelta, il progetto si basa su Telegram, ma, in quanto struttura generica, è stato pensato per essere facilmente integrato con altre applicazioni con la semplice aggiunta di classi e una modifica del codice minima.

2. Introduzione

2.1. Cosa sono i Chatbot?

I Chat Bot, o Chatbot, sono programmi che possono interagire, attraverso una chat, con l'essere umano, simulandone il comportamento. Viene quindi instaurata una conversazione tra essere umano e robot.

Fin dai primi sviluppi della scienza informatica, in collaborazione con altre discipline, gli studiosi hanno cercato di riprodurre, attraverso l'ausilio di macchine, i processi cognitivi tipicamente umani. Data la loro complessità, è ovvio che, in questo caso, non si possa parlare di una simulazione soddisfacente del comportamento proprio delle persone, ma ci si può comunque iniziare a riferire, in modo molto blando, al concetto di intelligenza artificiale (AI).

Il Bot, una volta sviluppato uno schema, può eseguirlo e mostrare il suo funzionamento. I Bot possono fare qualsiasi cosa, dal rispondere ai messaggi in modo automatico a permettere acquisti online, ricevere news di qualsiasi genere, condizioni meteo, visualizzare video musicali o addirittura possono consentire di sfidare gli amici a giochi in rete (Gamebot), tutto esclusivamente tramite chat.

Esistono svariate piattaforme che implementano la possibilità di utilizzare i Bot, tra queste ricordiamo: Telegram, Skype, Messenger, Slack, SMS, email. I Bot permettono di utilizzare queste piattaforme, applicazioni già conosciute e usate dall'utente per altre funzionalità (come la messaggistica), per svolgere al loro interno le funzioni più disparate evitando all'utente di usare e, spesso e volentieri, installare ulteriori applicazioni sul proprio dispositivo.

2.2. La storia dei bot

2.2.1. Il test di Turing

Anche se complessa e molto sfumata, l'inizio della storia dei Bot viene fatto coincidere con gli anni Cinquanta, quando l'informatico britannico Alan Turing teorizzò un test per capire se una macchina fosse in grado di imitare il comportamento umano: analizzando una conversazione tra un individuo e un computer, una persona esterna doveva capire quale dei due fosse l'automa, se la serie di scambi era tale da impedirne il riconoscimento, la macchina aveva superato il test di Turing.

2.2.2. ELIZA

Il primo Bot ad avvicinarsi al superamento del test sopracitato è stato ELIZA, creato da Joseph Weizenbaum nel 1966, un programma che fingeva di essere un psicoterapeuta e che regolava le risposte in base a ciò che veniva scritto dall'interlocutore. ELIZA, dopo essere stata interpellata, iniziava la conversazione chiedendo quale fosse il problema e, in base alla risposta, analizzando alcune parole chiave, era in grado di rispondere. ELIZA era quindi in grado di creare conversazioni verosimili in un ambito piuttosto ristretto, cosa che però permise al primo Bot di superare il test.

Nel 1973 Don Danglow scrisse una versione più sviluppata di ELIZA, chiamata Ecala e negli anni seguenti furono realizzati molti programmi basati su entrambi.

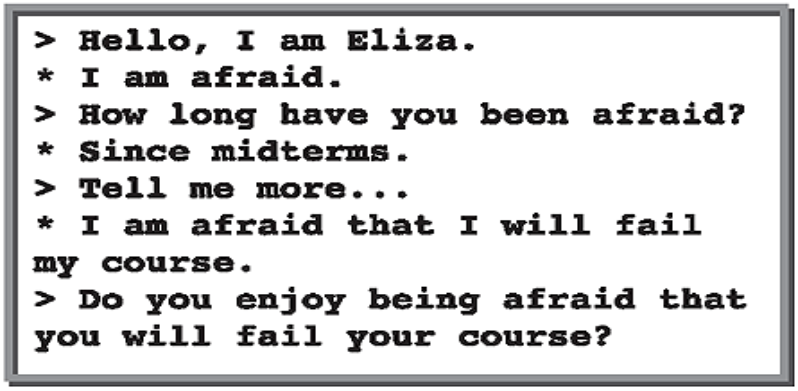

Figura 1 – ELIZA

2.2.2.1. Curiosità su ELIZA

- ELIZA fu chiamato così in onore di Eliza Doolittle, la fioraia, dal modo di espressione incolto e dialettale, protagonista della commedia Pigmalione (1912) di George Bernard Shaw, che, grazie alla ripetizione delle forme corrette di pronuncia, imparò il raffinato modo di esprimersi delle classi più agiate.

- Dal nome ELIZA è stato coniato il cosiddetto “Effetto ELIZA”, ossia il fenomeno psicologico che attribuisce a un computer, maggiore intelligenza di quanta in realtà possegga.

2.2.3. A.L.I.C.E.

A.L.I.C.E. (Artificial Linguistic Internet Computer Entity) nasce nel 1995 ad opera di Richard Wallace, poi riscritto nel 1998 in Java (Program D). Ispirato al suo predecessore ELIZA, è un NLP (Natural Language Processor) Chatbot. Il programma usa uno schema ispirato all’XML chiamato AIML (Artificial Intelligence Markup Language) che permette al Bot di avere una conoscenza di base sufficiente a una conversazione verosimile. Nel linguaggio AIML vengono immesse frasi preposte suddivise per categoria. Queste frasi sono scritte in modo da soddisfare le più comuni domande fatte in una conversazione

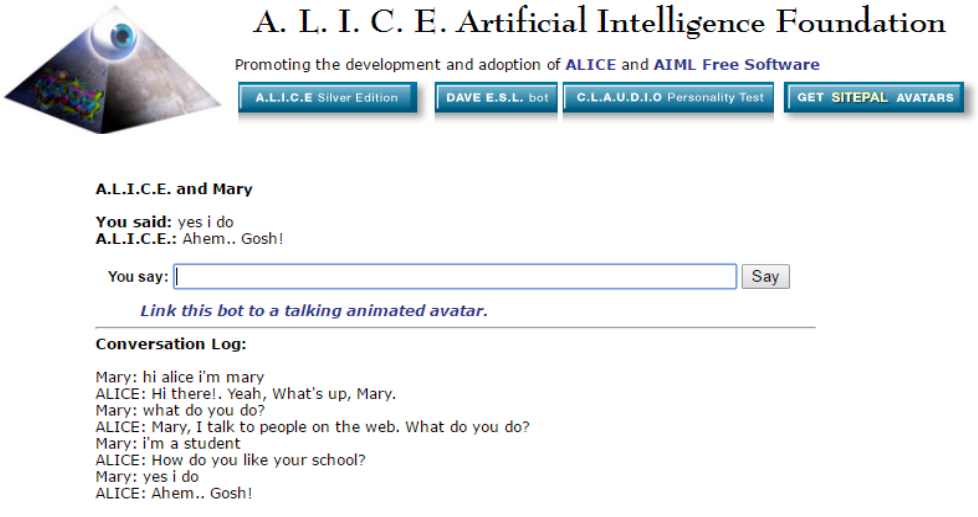

Figura 2 - A.L.I.C.E.

2.2.3.1. Curiosità su A.L.I.C.E.

- A.L.I.C.E. ha vinto il Loebner Prize tre volte (negli anni 2000, 2001, 2004).
- Spike Jonze ha citato A.L.I.C.E. come l'ispirazione per il suo film "Her" (vincitore del premio Oscar 2014 come miglior sceneggiatura originale e altri numerosi premi) in un articolo del New Yorker intitolato "Can Humans Fall in Love with Bots?" (Possono gli esseri umani innamorarsi di un robot?).

2.3. Il futuro dei bot

Dopo l'inizio promettente, l'arrivo dei social network, nei primi anni 2000, e, in seguito delle app, ha fatto vacillare il futuro di questa tecnologia fintanto da mettere quasi del tutto fuori circolazione i Bot.

I progressi nei sistemi di intelligenza artificiale e, soprattutto, la grandissima diffusione delle applicazioni di messaggistica, hanno poi riportato in auge i Bot come soluzione a svariati svantaggi dovuti al funzionamento del sistema dei dispositivi mobili e delle applicazioni.

Secondo ComeScore, in media l'80% del tempo trascorso sullo smartphone è impiegato nell'utilizzo di tre sole applicazioni, più o meno sempre le stesse, con l'abbandono quasi completo delle altre. Queste tre applicazioni, quasi sempre, coincidono con i principali promotori dei Bot, nonché principali app di messaggistica. Lo sviluppo dei Bot permetterebbe di sfruttare queste applicazioni come "canali" di interazione automatica con l'utente per qualsivoglia funzionalità (meteo, shopping online, news, pubblicità personalizzata sul cliente …).

I Chatbot sono visti, dagli osservatori più entusiasti, come il futuro dell'interazione con lo smartphone/pc.

Secondo gli scettici, invece, non avranno molto futuro. Le principali motivazioni alla base di questo scetticismo, si riferiscono al rapporto tra Bot e intelligenza artificiale. Il concetto di AI ha molte cose in comune con i Bot, ma non necessariamente un Bot deve saper rispondere a qualsiasi richiesta. Tipicamente un Bot ha uno scopo principale (o un numero ridotto di scopi), per cui le sue doti di AI vengono limitate a quel solo ambito, cercando di farlo rendere al meglio.

La preoccupazione nasce dal fatto che senza una buona intelligenza artificiale, i Bot possano risultare “stupidi”, rigidi e macchinosi. Il rischio è, quindi, quello che gli utenti non riescano a vedere l’utilità di una tecnologia che, tra qualche anno, potrebbe essere rivoluzionaria.

2.4. Un esempio di Bot: il TrackBot

TrackBot è un Bot di Telegram in grado di tracciare spedizioni, esattamente come si farebbe sui vari siti dei corrieri. TrackBot supporta numerosi corrieri che agiscono anche in Italia (BRT, FedEx, UPS, DHL, GLS, SDA, TNT, Nexive e Poste Italiane …).
È un servizio completamente gratuito che può essere utilizzato su qualsiasi client di Telegram (mobile, desktop e web).

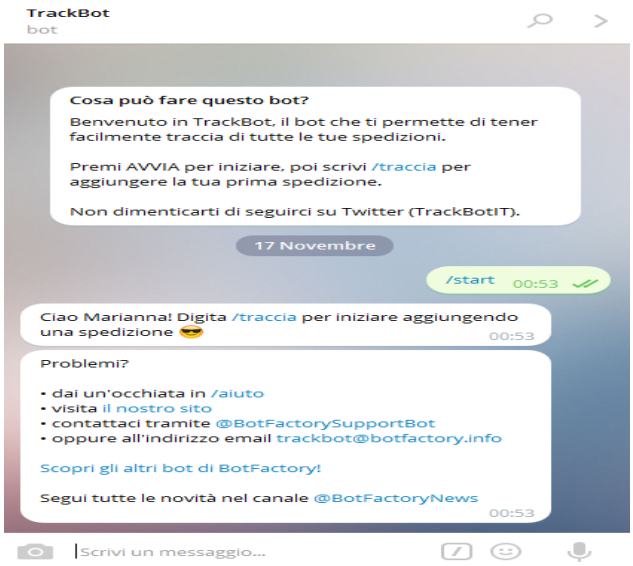

Figura 3 – TrackBot

2.5. Il Qbot

Figura 4 - logo Qbot

Il compito principale dell'applicazione Qbot è la risposta automatica a richieste inviate tramite applicazioni di messaggistica. L'idea principale è riuscire a svolgere le stesse funzioni di una qualsiasi applicazione web, ma attraverso richieste e risposte ricevute tramite chat.

Il Qbot prevede una procedura di login tramite CAS e controllo della presenza dell'utente nel database. È stato progettato per poter scegliere se utilizzare Webhook o una modalità polling, in modo da soddisfare qualsiasi tipo di richiesta.

L'implementazione presentata in questo elaborato, sarà basata su Telegram e svolgerà semplici funzioni quali la ripetizione del messaggio ricevuto e poco più, infatti, l'obbiettivo del progetto non era lo svolgimento di compiti complicati da parte del bot che implementa il Qbot, ma la creazione di una struttura generica che possa poi essere implementata per qualsiasi tipo di utilizzo.

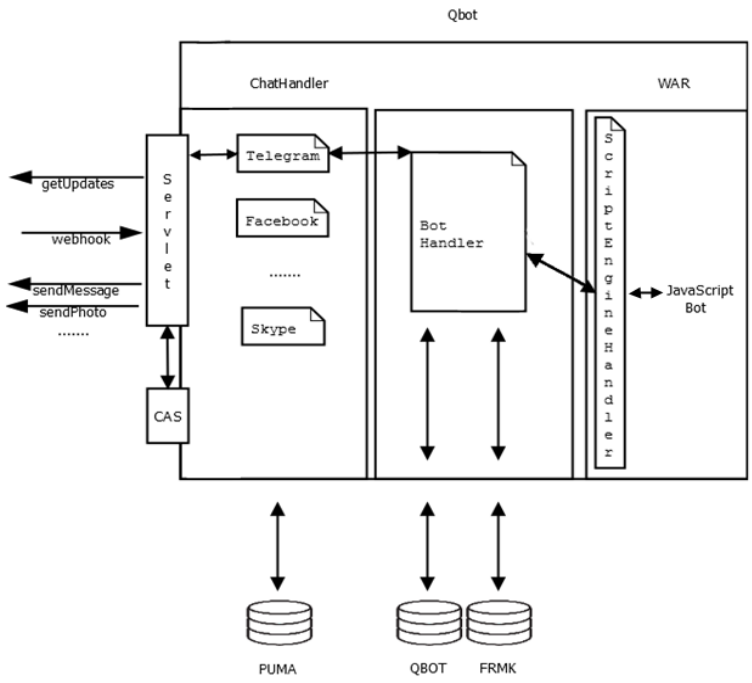

Figura 5 - struttura Qbot

3. Tecnologie utilizzate

3.1 Introduzione

In questo capitolo verranno descritte le seguenti tecologie utilizzate nel progetto Qbot:

- Java: l'intero progetto è stato scritto utilizzando il linguaggio ad oggetti Java (per la precisione jdk1.8.0_66);
- Eclipse: per la programmazione è stato impiegato l'ambiente di sviluppo integrato multi-linguaggio e multipiattaforma Eclipse;
- Apache Tomcat: come contenitore Servlet, in questo contesto, e come piattaforma di esecuzione di applicazioni Web è stato usato Apache Tomcat (versione 7.0.68);
- XML: l' XML è il metalinguaggio per la definizione di markup, che è stato utilizzato per la scrittura dei file (come ad esempio server.xml, web.xml e struts.xml) che definiscono le caratteristiche implementate delle varie tecnologie;
- JSP: le pagine JSP sono state impiegate nell'implementazione dell'interfaccia grafica per la gestione dell'elenco di Bot e BotChannel associati del Qbot.
- MVC Pattern: l'intero progetto segue il pattern MVC, suddividendo Model, View e Controller;
- CSS: i fogli CSS scritti e uniti al progetto hanno il compito di definire la grafica vera e propria delle pagine JSP;
- HTML: utilizzato all'interno delle pagine JSP;
- Servlet: usata per processare le richieste http ricevute dal Bot, in questo progetto ne è stata implementata una che svolge il compito di "ponte" tra la struttura scritta in java e il Bot vero e proprio;
- Apache Maven: usato per la gestione del progetto (attraverso il POM, ad esempio, sono state gestite le dipendenze);
- Apache Struts: framework che segue il pattern MVC, utilizzato in questo contesto per la definizione delle Action, come ad esempio quella di Login;
- Java Spring: framework impiegato, in questo progetto, ad esempio, per la definizione dei DataSources e dei DAO;
- MySQL: i database utilizzati sul RDBMS MySQL;

- SQL: linguaggio di interrogazione di DB utilizzato per la creazione e modifica dei DB MySQL;
- JavaScript: linguaggio di scripting orientato ad oggetti che è stato utilizzato, in questo progetto, per la definizion vera e propria del comportamento del Bot;
- Angular JS: utilizzato nelle pagine JSP come ng-controller per la definizione del comportamento dinamico delle pagine;
- Chiamate REST: utilizzate per il trasferimento dati su protocollo HTTP (ad esempio per l'invio dei messaggi tramite Bot o il reperimento degli aggiornamenti);
- Protocollo HTTP: protocollo a livello applicativo usato per la trasmissione di informazioni (sendMessage, getUpdates, setWebhook…);
- Bootstrap: usato nelle pagine JSP;
- Json: formato utilizzato per lo scambio di informazioni tra la struttura e il Bot;
- Telegram: applicazione di messaggistica su cui ci si è basati per la semplice implementazione del Bot vero e proprio.

3.2. JAVA

La maggior parte del progetto è stata scritta in Java.

Java è un linguaggio di programmazione orientato ad oggetti a tipizzazione statica, progettato per essere il più possibile indipendente dalla piattaforma di esecuzione. La peculiarità di questo linguaggio è la presenza della JVM (Java Virtual Machine), nata per interpretare il byte code, un linguaggio intermedio non destinato ad essere eseguito direttamente dall'hardware. Il byte code permette quindi di eseguire lo stesso codice su più piattaforme senza trasferire il sorgente, unico requisito di sistema sarà quindi la presenza della JVM. Questo è il principio fondamentale del linguaggio, espresso dal motto "*write once, run anywhere*" (WORA).

Altro grande vantaggio del linguaggio Java è il sistema di gestione della memoria, chiamato Garbage Collection. Il Garbage Collector libera il programmatore dal compito della gestione della memoria, assegnandola e rilasciandola automaticamente a seconda delle esigenze.

Java è stato creato per soddisfare cinque obiettivi primari:

- Essere semplice, orientato a oggetti e familiare;

- Essere robusto e sicuro;
- Essere indipendente dalla piattaforma;
- Contenere strumenti e librerie per il networking;
- Essere progettato per l'esecuzione di codice da sorgenti remote in modo sicuro.

Queste sono tra le principali ragioni della grande diffusione di Java, è infatti uno dei linguaggi di programmazione più utilizzati al mondo, specialmente per applicazioni client/server.

3.2.1. La storia di Java

I cambiamenti del linguaggio sono formalizzati nelle specifiche di linguaggio definite da un documento chiamato *The Java Language Specification* (spesso abbreviato in JLS) che integra i cambiamenti tramite le *Java Specification Request*. La prima edizione del documento è stata pubblicata nel 1996. Da allora il linguaggio ha subito numerose modifiche e integrazioni, aggiunte di volta in volta nelle edizioni successive. Ad oggi, la versione più recente delle specifiche è la *Java SE 8 Edition* (quarta).

Java è stato annunciato ufficialmente il 23 Maggio 1995 a SunWorld, occasione in cui, oltre alla prima JVM, è stato rilasciato anche il JDK 1.0 (Java Developer Kit). JDK 1.0 includeva già l'ambiente di esecuzione (JVM + librerie, chiamato JRE, Java Runtime Enviroment), il compilatore (Java to byte code, javac) e i tool di sviluppo.

Java 2 (versiona Java 1.2) è stata rilasciata nel 1998 ed è considerata un nuovo inizio per questo linguaggio. I maggiori cambiamenti furono:

- La riscrittura del sistema di event handling (aggiunta dell'Event Listener);
- Introduzione del JIT (Just In Time Compiler);
- Introduzione di nuove distribuzioni cablate sulle necessità del programmatore:
    1. J2EE (Java 2 Enterprise Edition), destinata allo sviluppo Enterprise.
    2. J2ME (Java 2 Micro Edition), destinata allo sviluppo di dispositivi embedded e mobile.
    3. J2SE (Java 2 Standard Edition), distribuzione classica.

L'attuale versione del linguaggio è Java 8 (Java 1.8). Negli anni si sono aggiunte molte funzionalità tra cui:

- JDPA - Java Platform Debugger Architecture (da Java 3);
- XML processing (da Java 4);
- Logging (da Java 4);
- Regular Expression (da Java 4);
- Web Services (Java 6);
- Scripting (da Java 6).

3.3. Ambiente di sviluppo: Eclipse

L'ambiente di sviluppo utilizzato per l'intero progetto è Eclipse, software libero, distribuito sotto i termini della Eclipse Public License e disponibile per le piattaforme più utilizzate (Linux, macOS, Windows …).

Eclipse è un ambiente di sviluppo integrato (IDE) multilinguaggio e multipiattaforma. Eclipse è scritto in linguaggio Java, la piattaforma di sviluppo è incentrata sull'utilizzo di plug-in, componenti software ideati per uno specifico scopo, che possono essere sviluppati e modificati da qualunque programmatore.

Nella versione base è possibile programmare in Java, usufruendo di funzioni di aiuto (completamento automatico, suggerimento dei tipi dei parametri dei metodi, accesso diretto a CVS).

Eclipse è stato creato dalla Eclipse Foundation, organizzazione non-profit fondata nel 2001 da società quali Borland, IBM, Red Hat, SUSE, HP, Fujitsu e altre.

Esistono diverse versioni di Eclipse, quella utilizzata nel progetto è Eclipse Mars (2015).

3.4. Apache Tomcat

Apache Tomcat è un application server (più precisamente contenitore servlet) open source, sviluppato dalla Apache Software Foundation. Tomcat fornisce una piattaforma software per l'esecuzione di applicazioni Web sviluppate in linguaggio Java, implementando le specifiche Java Server Pages (JSP) e Servlet.

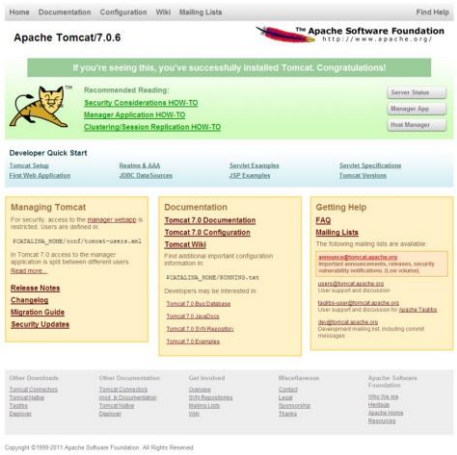

Figura 6 - Apache Tomcat/7.0.6

Da Tomcat versione 4.x, è stato distribuito con :

- Catalina (contenitore di servlet): in Tomcat un elemento del Reame rappresenta un database di username, password e ruoli assegnati all'utente. Differenti implementazioni del Reame permettono a Catalina di essere integrato in ambienti in cui queste informazioni di autenticazione sono già presenti e di utilizzarle per implementare la "Container Managed Security".
- Coyote (connettore HTTP) : supporta HTTP1.1 per il web server o per il contenitore di applicazioni, ascolta le connessioni in entrata su una specifica porta TCP sul server (di default la 8080, nel progetto la 8082) e inoltra la richiesta al Tomcat Engine che processa la richiesta per poi poter dare risposta al client richiedente.
- Jasper (motore JSP): analizza i file JSP per compilarli in codice Java come servlets (gestite poi da Catalina).

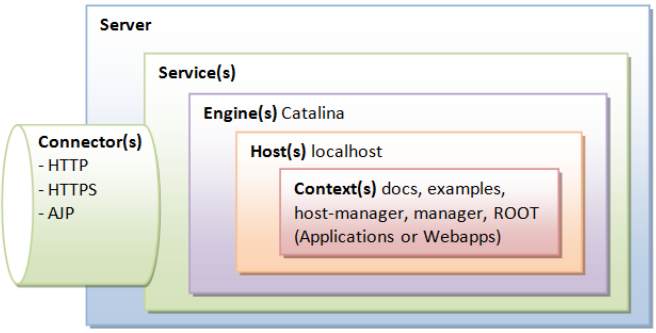

Figura 7 - struttura Apache Tomcat

3.4.1. File server.xml

```
<?xml version='1.0' encoding='utf-8'?>
<Server port="8005" shutdown="SHUTDOWN">
  <Listener className="org.apache.catalina.core.JasperListener" />
  <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />
  <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
  <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
  <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />

  <GlobalNamingResources>
    <Resource name="UserDatabase" auth="Container"
              type="org.apache.catalina.UserDatabase"
              description="User database that can be updated and saved"
              factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
              pathname="conf/tomcat-users.xml" />
  </GlobalNamingResources>

  <Service name="Catalina">
    <Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
    <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />

    <Engine name="Catalina" defaultHost="localhost">

      <Realm className="org.apache.catalina.realm.LockOutRealm">
        <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
               resourceName="UserDatabase"/>
      </Realm>

      <Host name="localhost"  appBase="webapps"
            unpackWARs="true" autoDeploy="true">
        <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
               prefix="localhost_access_log." suffix=".txt"
               pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />
      </Host>
    </Engine>
  </Service>
</Server>
```

Figura 8 - esempio di file server.xml

I principali elementi sono:

- <server>: rappresenta un'istanza di Tomcat.it, può contenere uno o più servizi, ognuno dei quali avrà i propri <engine> e <connector>;
- <listener> : elemento che ascolta e risponde a uno specifico evento. Esistono diversi tipi di ascoltatori tra cui:
    - JasperListener : responsabile della ricompilazione delle pagine JSP;
    - GlobalRecourcesLifecycleListener : abilita le risorse globali e rende possibile lì utilizzo di JINDI per accedere alle risorse come i database.
- <globalNamingRosources> : definisce il JINDI (Java Naming and Directory Interface) che consente all'utente l'accesso al database;

- <service> : associa uno o più connector a un Engine. Di default viene specificato “Catalina” a cui vengono collegati due Connectors: HTTP (porta 8080) e AJP (porta 8009);
- <connector> : associato a una porta TCP per la gestione delle comunicazione tra service e client.

3.4.2. File web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="qbot" ... >
    <context-param>
        <param-name>context-path</param-name>
        <param-value>/qbot</param-value>
    </context-param>
    <filter>
        <filter-name>struts</filter-name>
        <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>struts</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    <welcome-file-list>
        <welcome-file>Index.jsp</welcome-file>
    </welcome-file-list>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <resource-ref>
        <description>Reference to Collection database</description>
        <res-ref-name>jdbc/qbot</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>
    <servlet>
        <servlet-name>InitServlet</servlet-name>
        <servlet-class>it.quix.framework.web.servlet.InitServlet</servlet-class>
        <load-on-startup>5</load-on-startup>
    </servlet>
    <servlet>
        <servlet-name>BotServlet</servlet-name>
        <servlet-class>it.quix.qbot.web.servlet.BotServlet</servlet-class>
        <load-on-startup>5</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>InitServlet</servlet-name>
        <url-pattern>/InitServlet</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>BotServlet</servlet-name>
        <url-pattern>/channel/*</url-pattern>
    </servlet-mapping>
</web-app>
```

Figura 9 - esempio file web.xml

Il file web.xml contiene diversi elementi, tra cui:

- <web-app>: è il tag container di livello superiore, non ci può essere nulla al di fuori di questo tag;

- <context-param>: contiene i parametri dell'intero contesto o della singola applicazione definiti tramite i tag <description>, <param-name> e <param-value>. Si può usare, ad esempio, per settare una mail a cui inviare i messaggi di errore;
- <filter>: contiene i filtri per l'applicazione e gli eventuali parametri. Si può usare, ad esempio, per settare l'utilizzo di struts;
- <filter-mapping>: contiene la mappatura del filtro (url) o il nome della servlet. Più <filter-mapping> possono essere definiti per un unico filtro. Il <filter-mapping> richiede due parametri: il <filter-name> e <url-pattern> o < servlet-name >. Se si usa <servlet-name>, il filtro verrà chiamato ogni volta che la specifica servlet verrà invocata. Il <filter-mapping> deve essere definito dopo la definizione del tag <filter>;
- <welcome-file-list>: contiene la lista dei <welcome-file> in ordine di preferenza.
- <welcome-file>: specifica il path relativo di un singolo welcome file. Solitamente coincide con un file nominato "index.jsp" o con una servlet. I <welcome-file> vengono visualizzati nel caso in cui non venga specificato uno specifico parametro nell'URL;
- <listener>: definisce un ascoltatore per la web application. Ogni ascoltatore si riferisce a veri eventi (di applicazione, di sessione …).
- <resource-ref>: definisce i riferimenti ai database o Datasource e a che livello possono essere utilizzati.
- <servlet>: specifica le servlet Java e i loro parametri. Questa sezione deve assolutamente esistere per ogni servlet, insieme alla sezione <servlet-mapping>, per avvisare Tomcat dell'esistenza della servlet. Specifica almeno due parametri:
  - <servlet-name>: il nome della servlet che serve per collegare la servlet a quella specificata nel <servlet-mapping>;
  - <servlet-class>: nome della classe della servlet.
- <servlet-mapping>: specifica l'URL per una servlet specificata nel tag <servlet>. Questa sezione deve essere assolutamente dichiarata dopo il tag <servlet>. Richiede due parametri:
  - <servlet-name>: richiama il <servlet-name> della sezione <servlet>;
  - <servlet-url>: mappatura della servlet.

3.5. XML

XML (eXtensible Markup Language) è un metalinguaggio per la definizione di linguaggi di markup (come HTML) che permette di creare tag personalizzati.

Per essere correttamente interpretato, un documento XML, deve essere ben formattato, deve quindi dichiarare:

- Un prologo : prima istruzione del documento
  Ad esempio: <?xml version=”1.0” encoding=”UTF-8”?>
- Un unico elemento root : che conterrà tutti gli altri elementi
  Ad esempio: <web-app …>
- Tutti i tag devono essere aperti e chiusi.

L’XML, a parte la presenza dei <tag>, non è assolutamente simile all’HTML. L’HTML è un linguaggio di markup e definisce la formattazione di pagine web, l’XML è, invece, un metalinguaggio per la creazione di nuovi linguaggi di markup.

3.6. JSP

JSP (Java Server Pages) è un linguaggio flessibile e multipiattaforma in grado di generare pagine dinamiche lato server che utilizzano la sintassi Java. Una pagina JSP è costituita da markup (X)HTML (all’interno dei classici tag <html></html>) frammentato da sezioni di codice JavaScript (all’interno dei tag <script></script>) e/o Java (all’interno dei tag <% codice_Java %>), permette quindi modifiche alle parti sviluppate in Java/JavaScript lasciando inalterata la struttura HTML o viceversa. Essendo basate sulla tecnologia Java, le pagine JSP ereditano i vantaggi garantiti dalla metodologia object oriented e dalla portabilità multipiattaforma (utile anche in caso di presenza di sistemi operativi diversi tra loro). Le JSP si occupano della logica di presentazione (tipicamente secondo il pattern MVC – Model View Controller) e si basa su un insieme di tag (<nome_tag>).

La tecnologia JSP è strettamente correlata a quella delle servlet, infatti, all’atto della prima invocazione, vengono tradotte automaticamente da un compilatore JSP in

servlet, motivo per cui, l’uso di questa tecnologia, richiede la presenza, sul Web server, di un servlet container dotato di motore JSP (come ad esempio Apache Tomcat).

Le pagine JSP possono includere, oltre a script e codice Java, altre pagine JSP, tag propri della tecnologia struts, angular, fogli di stile CSS (sia scritti direttamente all'interno della pagina o esterni, inclusi tramite apposito tag all'interno del tag <head></head>) e altre tecnologie.

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
        <title>Una semplice pagina JSP di esempio</title>

        <!-- foglio di stile css scritto direttamente nel file -->
        <style type="text/css">
            .big {
                font-family: helvetica, arial, sans-serif;
                font-weight: bold;
                font-size: 20px;
            }

            .small {
                font-size: 20px;
                text-align: center;
            }
    </head>
    <body>
        <p class="big">PAGINA JSP DI ESEMPIO</p>

        <!-- tabella -->
        <table style= "border: 6px outset;">
            <tr>
                <td style = "background-color: black;">
                    <p class = "small" style ="color: cyan;">
                        <%=new java.util.Date()%>
                    </p>
                </td>
                <td>
                    <p class = "big">

                        <!-- parte Java, scriptlet -->
                        <%
                            String result= request.getParameter("firstName");
                            if( result!= null) {
                        %>

                        <h1> Hello <%=result%> </h1>

                        <%
                            //continuo scriptlet
                            else {
                        %>

                        <h1> Inserire nome </h1>

                        <%
                            //continuo scriptlet
                            }
                        %>
                    </p>
                </td>
            </tr>
        </table>
    </body>
</html>
```

Figura 10 - esempio JSP

3.7. MVC pattern

MVC (Model-View-Controller) è un pattern architetturale in grado di separare il codice in blocchi dalle funzionalità ben distinte (logica di presentazione e logica di business) permettendo allo sviluppatore di concentrarsi su un problema specifico alla volta.

Il componente centrale del MVC, il Model, avendo al suo interno i metodi di accesso ai dati, cattura il comportamento dell'applicazione in termini di dominio del problema, gestendo direttamente i dati, la logica e le regole applicative.

La View, che può essere una qualsiasi rappresentazione in output di informazioni (pagina HTML, diagramma, tabella…), si occupa di visualizzare i dati dell'utente e gestisce l'interazione tra quest'ultimo e l'infrastruttura sottostante.

Il Controller, accetta l'input attraverso la View e esegue operazioni che possono interessare il Model e che portano, quasi certamente, a un cambiamento di stato della View.

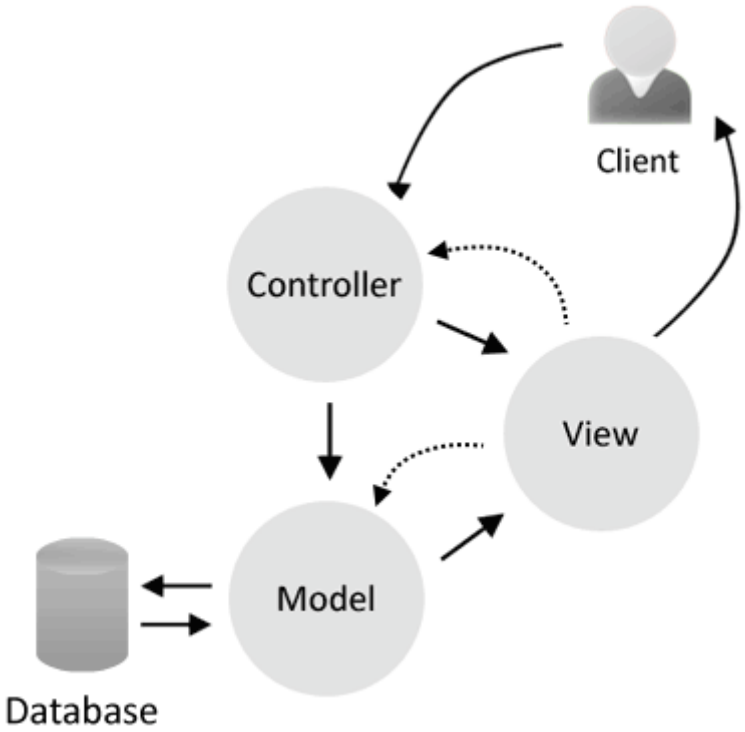

Figura 11 - MVC pattern

3.8. CSS

Il CSS (Cascading Style Sheet) è un linguaggio per la formattazione di documenti HTML, XHTML e XML.

L'introduzione di CSS ha reso possibile la separazione dei contenuti delle pagine HTML dalla loro grafica, permettendo un codice più chiaro e più facilmente riutilizzabile.

L'inserimento del codice CSS può avvenire in 3 modi diversi:

- Inserendo nel tag <head> della pagina HTML un collegamento a un foglio di stile esterno (ad esempio il file *style_example.css*) tramite la direttiva import, che viene utilizzata anche all'interno degli stessi file di stile per importarne altri. Questa soluzione consente il massimo riutilizzo del codice possibile.

```
1  <html>
2      <head>
3          <title>ExampleStyle</title>
4          <link rel="stylesheet" type="text/css" href="style_example.css"/>
5      </head>
6
7      <body>
8          ....
9      </body>
10 </html>
11
12 oppure
13
14 <html>
15     <head>
16         <title>ExampleStyle</title>
17         <style type="text/css">
18         @import url(style_example.css);
19         </style>
20     </head>
21
22     <body>
23         ....
24     </body>
25 </html>
26
27
```

Figura 12 - primo metodo inserimento CSS in HTML

- Inserendo, sempre all'interno del tag <head> tra gli specifici tag <style> e </style> il codice CSS. Questo metodo consente un riuso minore rispetto alla soluzione precedente, dato che, il codice, può essere utilizzato solo all'interno del file in cui è dichiarato.

```
13
14  <html>
15      <head>
16          <title>ExampleStyle</title>
17          <style type="text/css">
18              <!--codice css -->
19              .nome_classe1 {
20                  <!-- definizione stile classe -->
21              }
22              body {
23                  <!-- definizione stile body -->
24              }
25              ...
26          </style>
27      </head>
28
29      <body>
30          ....
31      </body>
32  </html>
33
```

Figura 13 - secondo metodo inserimento CSS in HTML

- La terza soluzione, la peggiore dal punto di vista della divisione del codice, prevede il codice CSS in linea all'interno degli elementi.

```
1   <html>
2       <head>
3           <title>ExampleStyle</title>
4       </head>
5
6       <body style = "dichiarazione CSS del body">
7           ....
8           <tag style = "dichiarazione CSS del tag qualsiasi sia il tag">
9               ...
10          </tag>
11          ....
12      </body>
13  </html>
14
```

Figura 14 - terzo metodo inserimento CSS in HTML

Un foglio CSS è strutturato come una sequenza di regole definite da un selettore (predicato che individua uno o più elementi del documento HTML) e un blocco di dichiarazioni (ogni dichiarazione, che è separata dalle altre con un punto e virgola, è costituita da una proprietà e il suo relativo valore) racchiuso tra parentesi graffe.

```css
@CHARSET "ISO-8859-1";

.frameworkLeftMenuCell {
    background-color: #27ae60;

}

.box-header {
    display:inline;
}

.typeChat {
    font-size: 30px;

}

.framework-sidebar-menu-element-link {
    height: 30px;
}

.logo {
    display:inline;
    height: 128px;
    width: 128px;
    margin: 10px;
}

.containerLaterale{
    display:inline;
}
```

Figura 15 - esempio di foglio CSS

3.9. HTML

HTML (HyperText Markup Language) è il principale linguaggio (non è un linguaggio di programmazione, ma di markup) di pubblicazione di pagine Web e di realizzazione di contenuti e applicazioni mobile. Questo linguaggio permette di indicare quali elementi visualizzare e come disporli all'interno di una pagina. Le indicazioni vengono date attraverso appositi tag racchiusi tra parentesi angolari (<nome_tag>) e sono contenute all'interno di un file HTML (con estensione *.html* o *.htm*). In quanto linguaggio di markup, non possiede i costrutti propri della programmazione, quali iterazioni (ad esempio for, while, do-while in Java) o condizioni (ad esempio if, if-else in Java), si parla quindi di linguaggio dichiarativo che indica cosa far comparire sulla pagina, dove farlo comparire e in quale sequenza.

Ora come ora, l' HTML delega lo stile e l'aspetto della pagina al CSS, occupandosi solamente della definizione degli elementi presenti, del collegamento ad altre pagine (link tramite il tag <a href="pagina_linkata"…/>), della creazione di form e così via.

La versione attuale è HTML 5, versione che nasce per espandere le capacità di questo linguaggio al mondo delle applicazioni, sia desktop che mobile. HTML 5 include, oltre alla sintassi di markup, una serie di API che consentono la gestione di network, multimedia e hardware dei dispositivi (video, audio, monitoraggio batteria …).

Un documento HTML inizia con una dichiarazione del tipo di documento, una stringa che indica la sintassi e la versione con cui è scritto il file, e che permette al browser di identificare le regole di visualizzazione del documento. Dopo la dichiarazione del tipo di documento, presenta una struttura ad albero annidato composta da diverse sezioni delimitate da tag opportuni che contengono a loro volta sottosezioni minori.

La struttura più esterna è quella delimitata dal tag <html></html> che indica inizio e fine del documento.

All'interno dei tag <html> vi sono sempre due sezioni:

- L'intestazione (header), delimitata dai tag <head></head>, che contiene informazioni di controllo normalmente non visualizzate dal browser e il titolo (tra i tag <title></title>) della pagina.

- La sezione del corpo, delimitata dai tag <body></body>, che contiene la vera e propria struttura della pagina (testo, immagini, link…).

I tag dell'header sono diversi da quelli del body, essendo destinati a scopi differenti, sono cioè informazioni di controllo e di servizio. Tra questi ci sono:

- metadata : fornisce informazioni utili ad applicazioni esterne (o al browser), come ad esempio la codifica dei caratteri;
- metadata di tipo HTTP-equiv : controlla le informazioni aggiuntive nel protocollo HTTP;
- script : inserimento di script (codice eseguibile) usati dal documento;
- CSS locali : informazioni di stile;

- title : titolo associato alla pagina e visualizzato nella finestra principale del browser.

```html
<html>
    <head>
        <link rel="stylesheet" type="text/css" href="style-ct.css" />
        <title>TITOLO</title>
    </head>
    <body>
        <div class="container">
            <div class="header">
                <h1>TITOLO</h1>
            </div>
            <div class="ciao">
                <div class="img">
                    <a target="_blank" href="fjords.jpg">
                        <img src=http://esempio_immagine.com alt="Fjords" width="300" height="200">
                    </a>
                    <div class="desc">EVENTI</div>
                </div>
                <div class="img">
                    <a target="_blank" href="forest.jpg">
                        <img src=http://esempio_immagine.com alt="Forest" width="300" height="200">
                    </a>
                    <div class="desc">FOTO</div>
                </div>
                <div class="img">
                    <a target="_blank" href="lights.jpg">
                        <img src=http://esempio_immagine.com alt="Northern Lights" width="300" height="200">
                    </a>
                    <div class="desc">ISCRIZIONE</div>
                </div>
                <div class="img">
                    <a target="_blank" href="mountains.jpg">
                        <img src=http://esempio_immagine.com alt="Mountains" width="300" height="200">
                    </a>
                    <div class="desc">AREA RISERVATA</div>
                </div>
            </div>
            <div class="login">
    <!--            <form action="demo_form.asp">
                    USERNAME: <input type="text" name="username" value="username"><br>
                    PASSWORD: <input type="text" name="password" value="password"><br>
                    <br/>
                    <input type="submit" value="LOGIN">
                </form> -->
            </div>
        </div>
    </body>
</html>
```

Figura 16 - esempio pagina HTML

3.10. Servlet

Una servlet è un software scritto in Java, gestito da un “container” (il servlet container, o servlet engine, è un’estensione di un web server che fornisce l’ambiente di esecuzione ad una Servlet. Un esempio di container è Apache Tomcat), in grado di estendere le funzionalità e capacità di un server. Le servlet sono principalmente utilizzate per implementare web application, applicazioni accessibili tramite browser (Web dinamico).

Essendo classi Java, sono del tutto integrabili con il resto della tecnologia Java, per cui ereditano i vantaggi propri di questo linguaggio, tra cui:

- Efficienza: il codice di inizializzazione della servlet viene eseguito un’unica volta, la prima volta che viene richiesta dal server; una volta caricata, ogni nuova richiesta si traduce in quasi esclusivamente in una chiamata a un metodo (maggiore velocità);

- Persistenza: la servlet, dopo il caricamento, rimane in memoria mantenendo informazioni per le successive richieste;
- Portabilità: le servlet possono essere velocemente “portate” da una piattaforma a un’altra;
- Affidabilità: dovuta alla possibilità di utilizzare JDK evolute, alla gestione delle eccezioni in caso di errore, alla quantità abbondante di librerie di supporto;
- Estensibilità: basata su un linguaggio orientato ad oggetti, per cui facilmente estensibile;
- Sicurezza: fornita dal web-server;
- Scalabilità: la servlet e il web possono girare tranquillamente su macchine diverse senza comprometterne le prestazioni;
- Compatibilità: le servlet sono supportate dalla maggior parte dei web-server.

Una servlet deve implementare l’interfaccia *Servlet* e può derivare dalla classe *GenericServlet* o da *HTTPServlet* (package *javax.servlet* e *javax.servlet.HTTP*), ereditando così i metodi base (doGet, doPost, init ecc…) per la gestione dell’interazione con un client via HTTP. La servlet ha quindi a disposizione un framework di classi Java che offre interfacce orientate ad oggetti che incapsulano la comunicazione tra client e server (request, response).

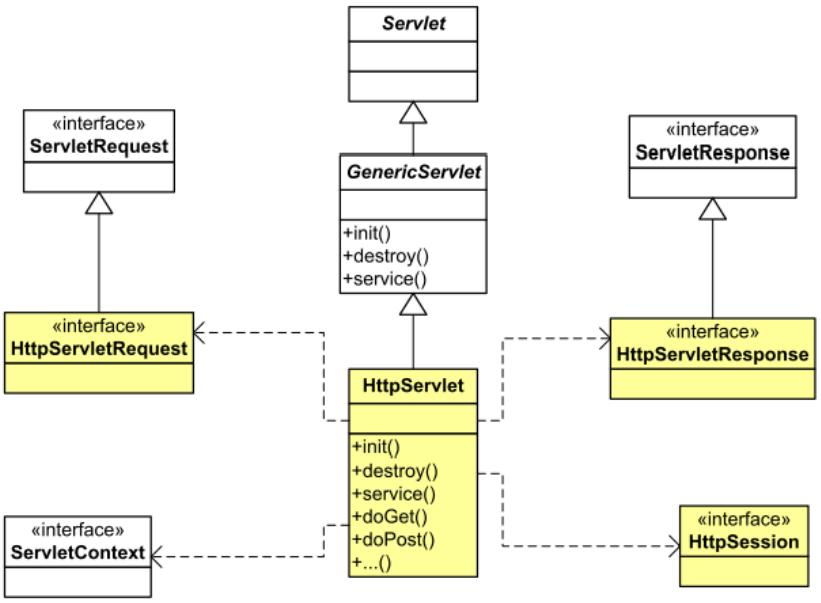

Figura 17 - gerarchia classi/interfacce

3.10.1. Il funzionamento delle Servlet

La Servlet interagisce con un web Client attraverso il paradigma di comunicazione request/response:

a. La servlet viene invocata attraverso una richiesta HTTP di un client ad un web server;
b. Il web server, quindi, scarica la servlet opportuna (solamente la prima volta) e poi crea un thread per eseguirla;
c. Il container esegue la servlet richiesta;
d. La servlet genera la risposta;
e. La risposta viene restituita al client attraverso una pagina HTML.

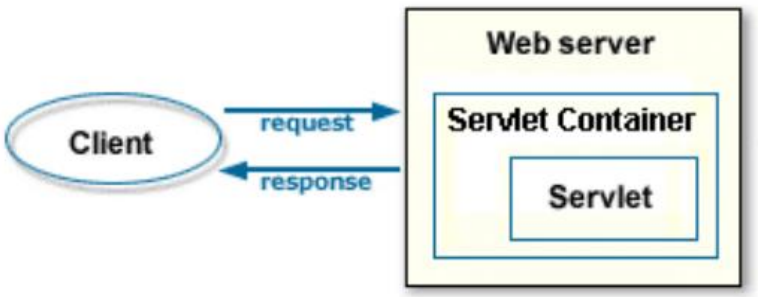

Figura 18 - paradigma client/server applicato alle servlet

3.10.2. Servlet Lifecycle

- Caricamento e installazione:
  Servlet MyServlet = new HTTPServlet();
- Inizializzazione:
  MyServlet.init(ServletConfig);
- Request Handling:
  MyServlet.service(request, response);
- Fine del servizio:
  MyServlet.destroy();

Il metodo init() viene eseguito una solo volta per ogni servlet, permette di accedere a risorse utili per le funzioni della servlet (ad esempio la connessione al database).

I metodi service(), doGet(), doPost() possono essere implementati dal programmatore:

- Service() per default delega l'esequzione al metodo indicato nella richiesta HTTP del client, le più comuni sono POST e GET, gestite dai metodi doPost() e doGet();
- doGet() : chiamato in risposta a una richiesta GET, viene usato per per ottenere dal server qualsiasi informazione, nella forma di entità, come definita nel campo Request-URI;
- doPost(): chiamato in risposta a una richiesta POST, è generalmente utilizzato lì dove siano necessarie operazioni in cui è richiesto il trasferimento di dati al
- server affinché siano processati. Una richiesta di tipo POST non utilizza il campo Request-URI per trasferire i parametri, bensì invia un pacchetto nella forma di message-header con relativo entity-header. Il risultato di una operazione POST non produce necessariamente entità, ma può anche generare semplicemente uno status-code (ad esempio 200 OK).

Esistono altri metodi meno utilizzati:

- doDelete(): chiamato in risposta a una richiesta DELETE, è usato per cancellare un file dal server. Non è disponibile per tutti i server e comporta rischi per la sicurezza;
- do Head(): chiamato in risposta a una richiesta HEAD, è usato quando il client vuole conoscere solamente l'header della risposta (ad esempio la validità di un URI o il tipo di contenuto);
- doOptions(): chiamato in risposta a una richiesta OPTIONS, ritorna al client le opzioni supportate dal server (ad esempio la versione del protocollo o i metodi supportati);
- doPut(): chiamato in risposta a una richiesta PUT, è usato per memorizzare un file nel server, come doDelete(), potrebbe non essere disponibile per tutti i server;
- doTrace(): chiamato in risposta a una richiesta TRACE, viene usato per il debugging, l'implementazione di questo metodo ritorna automaticamente un documento HTML contenente informazioni dell'header della pagina richiesta.

Per ogni richiesta viene creato un Thread che può essere riutilizzato nel caso in cui lo stesso client richieda più volta la stessa servlet.

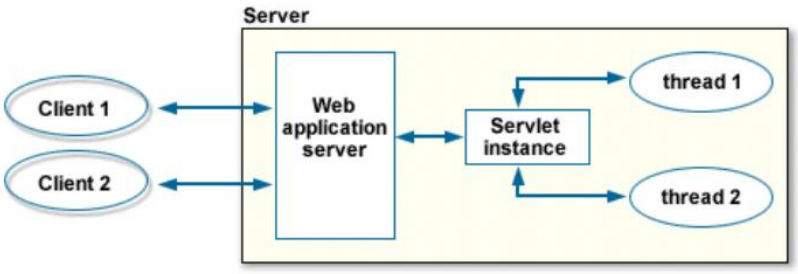

Figura 19 - creazione Thread per la servlet

3.11. Apache Maven

Maven è uno strumento per la gestione dei progetti Java e la compilazione del codice. Permette:

- Un'organizzazione pulita dei progetti;
- La standardizzazione della struttura di un progetto;
- La risoluzione della dipendenze (attraverso l'uso del repository maven o altri presenti sul web, configurati in appositi file);
- L'automatizzazione dei test;
- L'uso dei plugin.

I componenti principali di questo strumento sono:

- pom.xml: è un file di configurazione per plugin, dipendenze e repository;
- goals: sono funzioni eseguibili su singoli progetti;
- settings.xml: file di configurazione per repository locali, proxy, profili e sicurezza. Può essere specificato a due livelli, o per un singolo utente, o a livello di maven (quindi per tutti gli utenti maven).

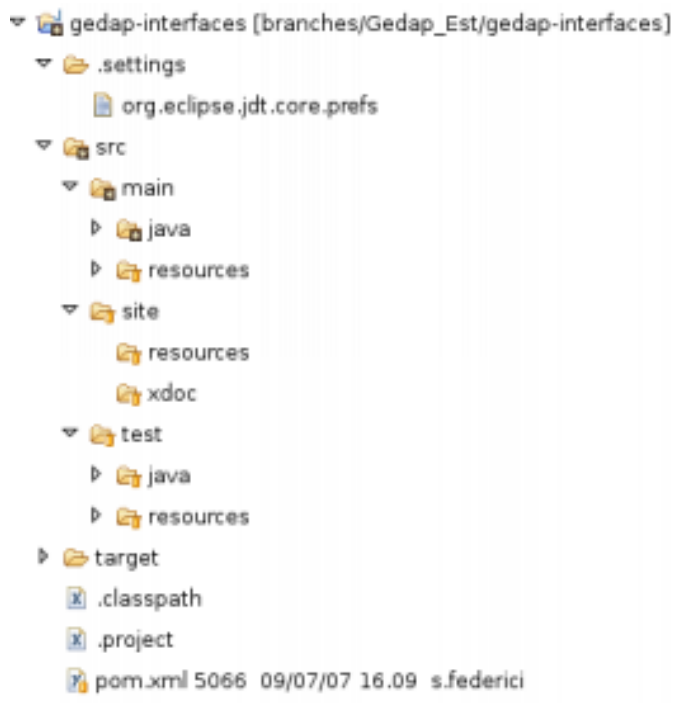

Figura 20 - struttura progetto generata da maven

3.11.1. POM (Project Object Model)

Il file pom.xml è composto da:

- <groupId> : gruppo associato al progetto, di solito nella forma “it.gruppo” (ad esempio “it.quix”);
- <artifactId> : ID del progetto (ad esempio “qbot”);
- <version> : versione del progetto;
- <packaging> : il tipo di pacchetto che si vuole avere (ad esempio war o jar);
- <dependencies> : insieme delle dipendenze denotate dal tag <dependency>;
- <dependency> : definisce una dipendenza specificando il <groupId>, <artifactId> e <version> della dipendenza, per ogni dipendenza è possibile anche definire uno <scope>, che può essere:
    - Compile (default): le dipendenze sono disponibile in ogni classpath del progetto;
    - Provided: come compile, ma prevede che a runtime le dipendenze siano disponibili solo dall’ambiente di esecuzione;
    - Runtime: le dipendenze sono disponibili sono in esecuzione;
    - Test: le dipendenze sono richieste solo per la compilazione e l’esecuzione;

    - System: la dipendenza non viene recuperata tramite repository, ma ne viene dichiarata esplicitamente la posizione locale.
- <repositories> : insieme dei repositories remoti, directories strutturate destinate alla gestione delle librerie. Maven ne crea uno in locale sotto la home dell'utente (evitando così anche inutili accessi alla rete), nel pom vengono definiti quelli remoti specificandone <id>, <name> e <url>.
- <build> : informazioni relative al processo di build, come ad esempio la struttura del progetto, la directory del file sorgente e la configurazione dei vari plug-in. Può essere usato in due diversi modi dal pom:
    - Project build: elemento che contiene le informazioni tipicamente usate durante il processo di build vero e proprio;
    - Profile build: alterazioni delle precedenti informazioni o di quelle di default.

  Gli elementi contenuti nel <build> sono:

    - <default Goal>: il goal o la frase da eseguire nel caso in cui non ne venga specificato alcuno;
    - <directory>: directory dove vengono memorizzati i file prodotti dal processo di build (di default *target*);
    - <finalName>: nome del prodotto di Maven (di default nome_artifact-numero_versione);
    - <filter>: fa riferimento a proprietà definiti in file esterni (con estensione .properties);
    - <resources>: risorse, file che non sono compilati, ma comunque inglobati nel progetto generato (ad esempio i file di configurazione di spring);
    - <plugin>: sezione che include i plug-in che si vogliono usare di cui vengono specificati groupId, artifactId, version e altre informazioni.
- <profiles>: insieme dei <profile>, elemento che consente di variare opportune impostazioni in funzione dell'ambiente di build. Tramite il tag <activation> possono essere attivati specificandone ad esempio jdk e sistema operativo;
- Altri elementi.

```
1.  <project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.   <groupId>it.cosenonjaviste</groupId>
5.   <artifactId>javaModule</artifactId>
6.   <version>1.0</version>
7.   <packaging>jar</packaging>
8.   <name>javaModule</name>
9.   <url>http://maven.apache.org</url>
10.  <dependencies>
11.   <dependency>
12.    <groupId>junit</groupId>
13.    <artifactId>junit</artifactId>
14.    <version>3.8.1</version>
15.    <scope>test</scope>
16.   </dependency>
17.  </dependencies>
18. </project>
```

Figura 21 - esempio di pom.xml

3.11.2. Plugin & Goal

Un goal è una funzione che può essere eseguita sul progetto. Può essere sia specifico per il progetto dove è incluso, che riusabile.

I plugin sono invece goal riutilizzabili in tutti i progetti.

Maven ha diversi plugin built-in disponibili, tra cui i principali sono:

- Clean: cancella i compilati del progetto
- Compiler: compila i file sorgenti
- Deploy: deposita il pacchetto generato nel repository remoto
- Install: deposita il pacchetto generato nel repository locale
- Site: genera la documentazione del progetto
- Archetype: genera la struttura di un progetto a partire da un template

3.12. Apache Struts

Apache Struts è un framework open source per lo sviluppo di web application su piattaforma Java EE.

Il framework di struts è un'implementazione server-side del design pattern MVC.

Questo framework riceve e gestisce tutte le richieste client per poi smistare il flusso applicativo in base alle configurazioni scritte all'interno di un apposito file XML (struts-config.xml) letto in fase di start-up dell'applicazione.

Realizzare un'applicazione con il supporto di struts ha diversi vantaggi:

- Rapidità di sviluppo: è possibile sviluppare in parallelo le varie parti dell'applicazione, logica di business e di view;
- Mantenibilità: l'applicazione è costituita da livelli logici ben distinti, per cui le modifiche rimangono circoscritte al modulo in cui vengono effettuate;
- Modularità e riusabilità: i diversi ruoli dell'applicazione vengono affidati a componenti diversi, questo permette uno sviluppo modulare e più facilmente riutilizzabile;
- Gestione automatica dei pool delle connessioni al database;

3.12.1. Componenti principali

Struts è un insieme di classi e interfacce che costituiscono lo scheletro delle applicazioni web. I principali elementi di questo scheletro sono:

- ActionServlet: servlet di controllo (Controller) che gestisce tutte le richieste dell'applicazione. Essendo una servlet, estende la classe javax.servlet.HTTP.HTTPServlet e implementa quindi tutti i metodi di lifecycle di questa classe;
- Struts-config.xml: in questo file xml vengono definiti tutti gli elementi dell'applicazione e le associazioni. Viene letto in fase di start-up dell'applicazione della ActionServlet;
- Action: classi a cui la servlet delega l'elaborazione delle action;
- ActionMapping: contiene gli oggetti associati a una Action nello struts-config (ad esempio gli ActionForward);
- ActionForm: fanno riferimento ad uno specifico form e vengono popolati automaticamente dal framework con i dati della request HTTP;
- ActionForward: contengono i path ai quali la servlet inoltra il flusso in base alla logica di apllicazione;
- Custom-tags: tag particolari forniti dal framework che permettono lo svolgimento dei compiti più comuni delle pagine JSP.

3.12.2. Struts Lifecycle

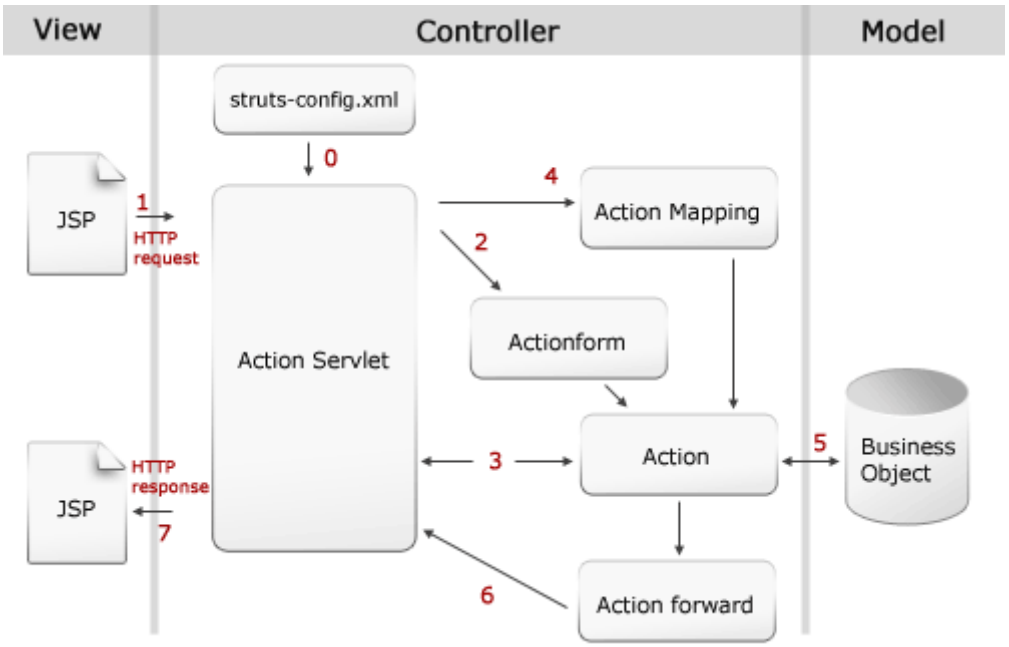

Figura 22 - flusso elaborativo di Struts

- I dati di configurazione vengono letti dalla servlet in fase di startup dal file struts-config.xml (0);
- Il client invia una richiesta HTTP (1);
- La richiesta viene ricevuta dalla servlet che provvede a popolare l'ActionForm e l'ActionMapping con i dati della request (2 - 4);
- La servlet delega l'elaborazione alla relativa Action (3) passandola in input HTTPRequest e HTTPResponse, ActionForm e ActionMapping;
- L'Action compie la logica di business e rende permanente lo stato dell'applicazione dialogando con il Model (5);
- L'Action, al termine dell'elaborazione, restituisce alla servlet un ActionForward (6) contenente il path della vista da fornire;
- L'Action esegue il forward della vista specificata nell'ActionForward (7).

3.12.3. File struts-config.xml

Il file struts-config.xml viene utilizzato da Struts per la configurazione e l'avvio di tutti i componenti necessari all'applicazione. Questo file permette di specificare il comportamento dei componenti del framework senza che venga inserito nel codice dell'applicazione.

La struttura del file vede il tag-root <struts-config> all'interno del quale ci sono 5 sezioni:

- <global-forwards> : in questo tag si creano associazioni globali tra particolari azioni del controller e i relativi percorsi (view)

```
<global-forwards>
    <forward
    className="classe che estende il bean di configurazione (di default org.apache.struts.action.ForwardConfig)"
    name="nome unico che serve a riferirsi a questo forward (attributo obbligatorio)"
    path="URI verso cui deve avvenire il forward (attributo obbligatorio che deve iniziare con '/')"
     />
</global-forward>
```

Figura 23 - <global-forwards>

- <form-beans> : definisce i form bean, classi che contengono i dati inseriti in un form di una pagina JSP

```
<form-beans>
    <form-bean
    className="classe dell'applicazione che sostituisce il bean di configurazione standard di Struts"
    name="nome unico che serve a riferirsi a questo form bean (attributo obbligatorio)"
    type="nome di una classe Java che estende ActionForm di Struts"
    />
</form-beans>
```

Figura 24 - <form-beans>

- <action-mappings> : definisce le Action. Per ogni azione inserisco sia l'elemento che i forward dell'azione

```
<action-mappings>
   <action
   path="percorso request inviata, quindi nome azione (attributo obbligatorio, inizia con '/')
   name="nome del form bean associato all'azione (facoltativo)
   input="path form bean (obbligatorio se presente name)
   validate="boolean che indica se il metodo validate() del form bean deve essere eseguito (di default true)
   scope="visibilità del form bean ('request' se relativo alla richiesta o 'session' se relativo alla sessione, facoltativo)
   type="elaborazione della request, classe Java che estende Action (presente in relazione a forward e include)"
   include="path di una servlet o JSP da includere nella response (presente in relazione a forward e type)"
   attribute="nome di un attributo necessario all'accesso di un form bean"
   parameter="informazioni extra per la action (recuperato nell'action con getParameter(), facoltativo)"
   className="classe alternativa per mapping e configurazione action (default ActionMapping)"
   >
   <forward ....(vedi forward)/>
   </action>
</action-mappings>
```

Figura 25 - <action-mappings>

- <message-resources> : classi per la gestione di messaggi in modo unificato nell'applicazione, utili per applicazioni multilingua. Nelle view si fa riferimento a delle key che sono associate ai messaggi corrispondenti

```
<message-resources parameter="MessageResources" />
```

Figura 26 - <message-resources>

- <plug-in> : dichiarazione dei plugin usati nell'applicazione. Il più importante è il validator, che permette di validare i parametri in un form

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
        <set-property  property="pathnames"
                       value="/org/apache/struts/validator/validator-rules.xml,
                              /WEB-INF/validation.xml" />
</plug-in>
```

Figura 27 - dichiarazione plug-in validator

3.13. Apache Struts2: Principali cambiamenti rispetto alla prima versione

- Web.xml
  - Struts: il controllo affidato a una servlet (di default è la ActionServlet, ma è possibile dichiararne una personalizzata)
  - Struts 2: il controllo viene affidato a un filtro (di default FilterDispatcher, ma è possibile definirne uno personalizzato)
- URI pattern
  - Struts: di default si utilizza il pattern *.do per identificare una richiesta che la ActionServlet deve prendere in carico
  - Struts 2: di default si utilizza il pattern *.action per identificare una richiesta che il FilterDispatcher deve prendere in carico
- File di configurazione
  - Struts: di default è il file struts-config.xml posizionato allo stesso livello del file web.xml
  - Struts 2: di default è il file struts.xml posizionato nella directory del classpath
- Mapping delle Action
  - Struts: definito nel file di configurazione nel tag <action-mapping>
  - Struts 2: automaticamente generato dalla concatenazione “package-nome_action” (valori definiti nel file di configurazione)
- Proprietà delle Action
  - Struts: proprietà delle action definite nella classe ActionForm (insieme ai metodi get e set)
  - Struts 2: proprietà definite direttamente nella Action (insieme ai metodi get e set)
- Action
  - Struts: una Action estende la classe org.apache.struts.action.Action
  - Struts 2: una Action implementa l’interfaccia com.opensymphony.xwork2.Action o estende la classe com.opensymphony.xwork2.ActionSupport (che a sua volta implementa Action)

- Metodo execute
  - Struts: il metodo execute di una Action restituisce un ActionForward
  - Struts 2: il metodo execute di una Action restituisce una stringa, è il controller che definisce, in base alla stringa restituita, qual è la vista da richiamare
- TagLibrary
  - Struts: disponibili diverse tagLibrary suddivise per tipo
  - Struts 2: disponibile un'unica tagLibrary che fornisce sia operazioni di logica che di rendering HTML
- Interceptor
  - Struts: non presenti
  - Struts 2: gli Interceptor sono classi stateless che possono essere invocate prima o dopo una Action o un insieme di Action. Svolgono lo stesso compito dei filtri. Il framework dispone di diversi Interceptor predefiniti che possono essere inseriti in uno stack (l'esecuzione segue la politica FIFO: first in, first out)

3.14. Java Spring

Gli stessi autori di Spring lo definiscono come:

"un framework open source nato con l'intento di gestire la complessità nello sviluppo di applicazioni enterprise".

I principali vantaggi offerti da Spring possono essere riassunti nei seguenti punti:

- È un framework leggero che non sconvolge l'architettura pre-esistente di un progetto. Grazie alla sua architettura modulare, può essere utilizzato per intero o solo in parte ed è facilmente integrabile con altri framework, come per esempio Struts;
- È un lightweight container e può essere visto come una alternativa o un completamento di J2EE (Java2 Enterprise Edition). Propone un modello semplice e leggero anche grazie all'utilizzo di tecnologie come l'Inversion of Control e l'Aspect Oriented.

- Mette a disposizione un insieme completo di strumenti per la gestione dell'intera complessità di un progetto software fornendone un approccio

semplificato (ad esempio per accesso ai database, gestione dipendenze, testing…).

L'utilizzo di Spring permette l'esclusione di parti del framework non necessarie all'applicazione grazie alla sua natura modulare, che si compone principalmente di cinque livelli:

- Core Container: è l'elemento su cui è costruito il framework. Contiene i moduli di Core e Beans che sono responsabili della funzionalità di IoC (Inversion of Control) e Dependency Injection e hanno come compito principale la creazione, manipolazione e gestione di qualsiasi oggetti, che nel contesto di Spring viene chiamato bean. Contiene inoltre il modulo Context che estende i servizi base del Core, aggiungendone funzionalità, tra cui JINDI e supporto agli eventi. Include infine il modulo Expression Language che fornisce un linguaggio per interrogare e modificare oggetti a runtime.
- Data Access/Integration: fornisce l'accesso al database tramite tecnologie eterogenee (ad esempio JDBC, Hibernate..) nascondendone la complessità delle API.
- Spring AOP: aggiunge al framework la funzionalità della programmazione Aspect Oriented facilitando le operazioni trasversali tra più oggetti (ad esempio gestiosce le transazioni).
- Web: gestisce le caratteristiche Web del framework mettendo a disposizione una implementazione del pattern MVC (Spring MVC Framework).
- Testing: Spring mette a disposizione un ambiente di testing molto potente.

Figura 28 - moduli Spring

Ogni configurazione (data source, action, DAO, altri beans…) viene fatta in un file XML (può contenere importazioni di altri file XML).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans  xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:context="http://www.springframework.org/schema/context" xsi:schemaLocation="
    http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
    http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- file included -->
    <aop:aspectj-autoproxy />
    <context:annotation-config />
    <import resource="classpath:qbot-ds-spring.xml" />
    <import resource="classpath:qbot-dao-spring.xml" />
    <import resource="classpath:qbot-validator-spring.xml" />
    <import resource="classpath:qbot-config-spring.xml" />

    <!-- Aspects -->
    <bean id="userContextAspect" class="it.quix.framework.core.aspect.UserContextAspect">
        <property name="order" value="1" />
    </bean>
    <tx:annotation-driven transaction-manager="transactionManager" order="2" proxy-target-class="true" />

    <!-- jdbc spring -->
    <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager" >
        <property name="dataSource" ref="qbotDataSource" />
    </bean>

    <!-- beans application -->
    <bean id="userContext" class="it.quix.qbot.core.model.QbotUserContext" scope="prototype">
        <property name="pumaDataSource" ref="qbotPumaDataSource" />
        <property name="commonsDataSource" ref="qbotCommonsDataSource" />
    </bean>
    <bean id="userContextFactory" class="it.quix.framework.core.model.UserContextFactory">
        <property name="userContextClass" value="it.quix.qbot.core.model.QbotUserContext" />
    </bean>
    <bean id="validatorFactory" class="it.quix.qbot.core.validation.ValidatorFactory" />
    <bean id="constraintFactory" class="it.quix.framework.core.validation.ConstraintFactoryImpl" />
    <bean id="qbotManager" class="it.quix.qbot.core.manager.QbotManager" />
    <bean id="botHandler" class="it.quix.qbot.core.handler.BotHandler" />
    <bean id="scriptEngineHandler" class="it.quix.qbot.core.handler.ScriptEngineHandler" />
</beans>
```

Figura 29 - esempio di file di configurazione Spring

3.15. MySQL

I database sono strutture organizzate che memorizzano permanentemente i dati, in modo da poterli elaborare (leggere, aggiungere, aggiornare, cancellare) successivamente.

MySQL è un RDBMS (Relational DataBase Management System) open source, composto da un client a linea di comando e un server.

Un DBMS (e quindi anche un RDBMS) è un software di tipo server avente il compito di gestire uno o più database.

Un RDBMS è un DBMS che si riferisce alla teoria relazionale. Nel modello relazionale, i dati all'interno di un database, sono organizzati in diverse tabelle che sono, a loro volta, in relazione tra loro. Per interrogare le tabelle e quindi interagire con i dati, MySQL utilizza SQL, un linguaggio di interrogazione strutturata.

I database costituiscono un importante strumento anche nello sviluppo di web application, dato che è grazie a questi che, le pagine web, acquisiscono memoria persistente.

3.16. SQL

SQL (Structured Query Language) è un linguaggio standardizzato per l'interrogazione e la gestione di database (DB) basati sul modello relazionale, progettato per:

- Creare, cancellare e modificare database (DDL – Data Definition Language);
- Inserire, modificare e gestire dati memorizzati (DML – Data Manipulation Language);
- Interrogare i dati memorizzati attraverso query (DQL – Data Query Language);
- Gestire gli utenti e i permessi (DCL – Data Control Language);
- Controllare i supporti (memorie di massa) dove vengono memorizzati i dati (DMCL – Device Media Control Language).

3.16.1. DDL: comandi base

Il DDL fornisce i comandi per agire sullo schema di un database permettendo, quindi, di definirne la struttura e l'organizzazione logica dei dati contenuti.

I principali comandi sono:

- CREATE DATABASE : permette di creare un nuovo database, se è presente l'istruzione IF NOT EXIST, il DB verrà creato solo nel caso in cui non esista già.
- ALTER DATABASE : permette di modificare un database
- DROP DATABASE : permette di eliminare un database. Questo comando non esiste nello standard SQL, ma la maggior parte dei DBMS lo implementa.
- CREATE TABLE: permette di creare una nuova tabella;
- ALTER TABLE: permette di modificare una tabella. Esistono diversi operatori che possono specificare questo comando:
  - ADD: dà istruzioni per l'inserimento di un nuovo elemento (ad esempio inserisce una nuova colonna);
  - MODIFY: dà istruzioni per la modifica di un elemento già esistente in quella tabella;
  - DROP: dà istruzioni per l'eliminazione di un elemento già esistente nella tabella (quale ad esempio una colonna);
- DROP TABLE: permette l'eliminazione di una tabella, essendo un'operazione irreversibile, provoca la perdita di tutti i dati della tabella;
- RENAME: permette di rinominare una tabella.

```
CREATE TABLE indirizzi (
   id serial PRIMARY KEY,
   nome varchar(40) NOT NULL,
   cognome varchar(40) NOT NULL,
   indirizzo varchar(40) DEFAULT 'sconosciuto',
   telefono varchar(40) NOT NULL,
   id_comune int references (ammcom.id),
   data_nascita date
);
```

Figura 30 - esempio di comando DDL

3.16.2. DML: comandi base

Il DML fornisce i comandi per l'inserimento, l'aggiornamento, la modifica e l'eliminazione dei dati all'interno le tabelle dei DB (create in precedenza, insieme al DB, utilizzando i comandi DDL).

I comandi principali sono:

- INSERT: inserisce i dati nelle tabelle. Le colonne di destinazione possono essere o meno dichiarate nel comando, se non vengono dichiarate, è necessario passare al comando un valore per ogni colonna nell'esatto ordine delle colonne stesse;
- UPDATE … SET: modifica i dati delle tabelle. La parola chiave SET indica il campo da modificare e il suo nuovo valore. L'UPDATE può essere invocato in modo generico o specificando una condizione (tramite la parola chiave WHERE);
- DELETE: elimina i dati dalle tabelle. Come UPDATE, anche DELETE permette un'eliminazione generica o con condizione.

```
INSERT INTO Indirizzi (
    nome,
    cognome,
   indirizzo,
    telefono,
   compleanno
 )
 VALUES (
    'Gigi',
    'Marzullo',
    'Via Teulada 66 Roma',
    '06 1234567',
   '1915/03/26'
 );
```

Figura 31 - esempio comando DML

3.16.3. DQL: comandi base

Il DQL , in quanto linguaggio di interrogazione vero e proprio, comprende tutti quei comandi per leggere e elaborare i dati presenti sul DB (inseriti attraverso il DML in strutture create dal DDL).

Il comando del DQL è il SELECT, comando che estrae i dati, in modo mirato dal DB.

```
SELECT [ ALL | DISTINCT |     TOP ] lista_elementi_selezione
FROM lista_riferimenti_tabella
[ WHERE espressione_condizionale ]
[ GROUP BY lista_colonne [HAVING Condizione] ]
[ ORDER BY lista_colonne ];
```

Figura 32 - struttura comando SELECT

Una forma di SELECT composto tra diverse tabelle con uno o più campi comuni si ottiene attraverso il JOIN.

3.16.4. DCL: comandi base

Questi comandi servono a dare o revocare i permessi agli utenti. Tali permessi sono necessari per poter utilizzare sia i comandi DML che DDL e DCL.

I comandi di questa sezione sono:

- GRANT : concede uno o più permessi a un determinato utente;
- REVOKE : revoca uno o più permessi a un determinato utente.

3.17. CRUD

CRUD (Create Read Update Delete) è l'acronimo per le quattro funzioni base della memorizzazione persistente (interazione con database). A volte, CRUD, viene usato anche per la descrizione delle convenzioni dell'interfaccia utente che facilitano la visualizzazione, la ricerca e l'aggiornamento delle informazioni (ci si riferisce a CRUD spesso e volentieri anche nell'ambito delle chiamate REST). Varianti di CRUD, a volte utilizzate, sono BREAD (Browse Read Edit Add Delete) e MADS (Modify Add Delete Show).

Operazione	SQL	HTTP
Create	INSERT	PUT/POST
Read	SELECT	GET
Update	UPDATE	POST/PUT/PATCH
Delete	DELETE	DELETE

Tabella 1 – CRUD

3.18. DAO

Il DAO (Data Access Object) è un pattern architetturale per la gestione della persistenza. In pratica, è una classe (Java) con relativi metodi, attraverso cui si interroga il database e che, quindi, incapsula le modalità di comunicazione con quest'ultimo. I metodi del DAO corrispondono alle query e verranno chiamati esclusivamente dal Manager dell'applicazione, una classe della business logic. In particolare conterrà le funzionalità di base: CRUD (Create, Read, Update, Delete).

Il vantaggio relativo all'utilizzo del DAO è il mantenimento di una rigida separazione tra le componenti dell'applicazione (soprattutto se basata sul paradigma MVC).

Nel caso di interrogazioni che coinvolgono più di una classe di dominio (classi a cui si riferisce il DAO e che corrispondono alle tabelle del DB), si definiscono nuove classi di accesso ai dati, dette Data Control Service (DCS), anch'esse parte delle persistence classes.

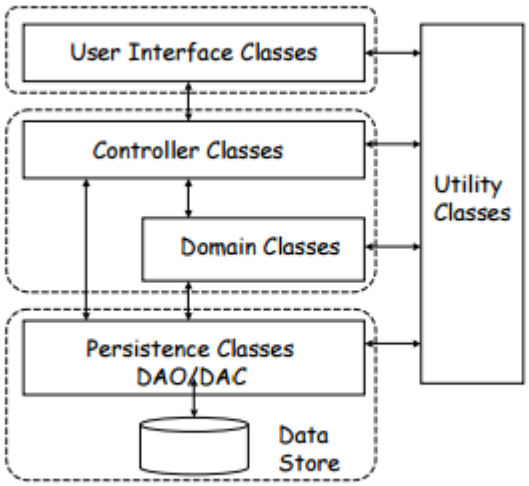

Figura 33- architettura con DAO e DCS

Utilizzando Java Spring, anche il DAO viene dichiarato come bean, permettendo la connessione al Datasource e di conseguenza al database.

```
<bean id="botChannelDAO" class="it.quix.qbot.core.dao.BotChannelDAO" >
    <constructor-arg ref="qbotDataSource" ></constructor-arg>
</bean>

<bean id="calendarDAO" class="it.quix.qbot.core.dao.CalendarDAO" >
    <constructor-arg ref="qbotCalendarDataSource" ></constructor-arg>
</bean>
```

Figura 34 - dichiarazione bean DAO

3.19. Datasource

Il Datasource, o pool di connessioni, permette il disaccoppiamento tra applicazione (classi JDBC) e sorgente dati, affidando la gestione delle connessioni all'application server che ne garantirà le connessioni ai database che ne faranno richiesta.

La creazione del Datasource, è un'operazione descrittiva realizzata tramite file xml. Nel file verranno dichiarate le credenziali di accesso, ma anche il numero massimo di connessioni contemporanee, il numero minimo di connessioni da mantenere nel pool e tutte le caratteristiche necessarie a ottimizzare il ciclo di vita dell'applicazione.

```
<bean id="qbotDataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName">
        <value>java:comp/env/jdbc/qbot</value>
    </property>
</bean>
<bean id="qbotCalendarDataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName">
        <value>java:comp/env/jdbc/calendar</value>
    </property>
</bean>
```

Figura 35 - dichiarazione bean Datasource

```
<GlobalNamingResources>
      <Resource auth="Container" driverClassName="com.mysql.jdbc.Driver"
      validationQuery="select 1" maxActive="100" maxIdle="30"
      maxWait="10000" name="jdbc/puma"  type="javax.sql.DataSource"
      url="jdbc:mysql://srvsqldev:3306/quixpro"
      username="mysql"
      password="mysql"
      />

      <Resource auth="Container" driverClassName="com.mysql.jdbc.Driver"
      validationQuery="select 1" maxActive="100" maxIdle="30"
      maxWait="10000" name="jdbc/calendar"  type="javax.sql.DataSource"
      url="jdbc:mysql://localhost:3306/calendar"
      username="root"
      password="******"
      />
</GlobalNamingResources>
```

Figura 36 - dichiarazione in risorse globali di server.xml dei Datasource

```
<Context docBase="${workspaceDirectory}\qbot\src\main\webapp" path="/qbot">
    <Loader className="it.quix.tomcat.VirtualWebappLoader"
            virtualClasspath="${workspaceDirectory}\qbot\target\classes\;${workspaceDirectory}\qbot\target\qbot\WEB-INF\lib\*.jar"/>
    <ResourceLink name="jdbc/qbot" global="jdbc/qbot" type="javax.sql.DataSource"/>
    <ResourceLink name="jdbc/commons" global="jdbc/framework" type="javax.sql.DataSource"/>
    <ResourceLink name="jdbc/puma" global="jdbc/puma" type="javax.sql.DataSource"/>
    <ResourceLink name="jdbc/framework" global="jdbc/framework" type="javax.sql.DataSource"/>
    <ResourceLink name="jdbc/calendar" global="jdbc/calendar" type="javax.sql.DataSource"/>
</Context>
```

Figura 37 - dichiarazione contesto server.xml

3.20. JavaScript

JavaScript (abbreviato spesso con JS) è un linguaggio di scripting interpretato e dinamico client-side per pagine web. JS viene eseguito direttamente lato client dalla pagina web e può anche essere utilizzato per delinearne il design e stabilirne il comportamento nel caso di un particolare evento (scatenato dall'utente).

JavaScript permette di programmare sia in modo procedurale, sia orientato agli oggetti consentendo, tra le altre cose, la creazione a runtime di oggetti, elenchi di parametri, funzioni, variabili, script dinamici (tramite il comando eval) e il recupero del codice sorgente.

JavaScript può essere sia incluso in una pagina HTML (tramite tag <script>), che dichiarato in un file a parte (con estensione .js).

```
<html>
    <head>
        <title>La mia prima applicazione JavaScript</title>
    </head>
    <body>
        <script>
            document.write("Hello, world!");
        </script>
    </body>
</html>
```

Figura 38 - esempio di Js interno a HTML

```
12 function impostaCookie (nome, valore, percorso, scadenza) {
13    valore=escape(valore);
14
15    if (scadenza == "") {
16       var oggi = new Date();
17       oggi.setMonth(oggi.getMonth()+6);
18       scadenza=oggi.toGMTString();
19    }
20    if (percorso!="")
21       percorso= ";Path=" + percorso;
22
23    document.cookie = nome + "=" + valore + ";expires=" + scadenza + percorso;
24 }
25
```

Figura 39 - esempio codice JavaScript

3.21. Angular JS

AngularJS è un framework JavaScript per lo sviluppo di applicazioni Web client side.

Tra le principali funzionalità di Angular ci sono:

- Binding bidirezionale (two-way binding);
- Dependency injection (pattern che implementa la Inversion of Control per risolvere le dipendenze);
- Supporto al pattern MVC;
- Supporto ai moduli;
- Separazione delle competenze;
- Testabilità del codice;
- Riusabilità dei componenti.

Angular non è una semplice libreria, ma un completo framework per lo sviluppo di applicazioni JavaScript. La differenza principale tra libreria e framework può essere riassunta così:

	Libreria	**Framework**
Cos'è?	Collezione di funzioni specializzate per un determinato compito (ad esempio il supporto di uno specifico pattern).	Infrastruttura predisposta alla realizzazione di una applicazione secondo un determinato approccio.
Come si usa?	Il codice del programmatore decide se e quando utilizzare le funzioni messe a disposizione dalla libreria.	Il codice del programmatore si inserisce in questa infrastruttura per implementare il comportamento specifico dell'applicazione (è il framework che invoca il codice).

Tabella 2 – differenza libreria/framework

Nella realizzazione di un'applicazione, Angular propone una struttura modulare composta da diversi componenti, ognuno dei quali svolge una ben determinata funzione, seguendo il principio della separazione delle competenze. I componenti principali sono:

- View: interfaccia grafica generata a partire da un template HTML elaborato da Angular;
- Controller: oggetto JavaScript che espone dati e funzioni a una view;
- Filtro: funzione che formatta il valore di un'espressione per la visualizzazione di una view (formattazione della data, ad esempio);
- Direttiva: componente che estende l'HTML con tag e attributi personalizzati (unico componente che può manipolare il DOM via JavaScript);
- Servizio: oggetto che fornisce funzionalità indipendenti dall'interfaccia (view), ad esempio l'accesso al server HTTP.

Questo framework lavora leggendo prima di tutto la pagina HTML, che al suo interno ha degli attributi addizionali (ad esempio “ng-controller“) all’interno dei tag che vengono interpretati da Angular come dei comandi veri e propri.

3.21.1. Attributi e elementi di Angular

Gli attributi “ng” definiti da Angular sono delle direttive che definiscono il comportamento di un elemento HTML della pagina.

Molto importanti tra questi attributi sono:

- ng-app: definisce il contesto dell’applicazione, cioè indica ad Angular quale elemento interpretare come view della nostra applicazione. Ogni elemento della pagina può essere definito come contesto attraverso questo attributo;
- ng-model: utilizzato sull’elemento “input” e definisce un modello di dati associato all’elemento di input. Questo meccanismo viene chiamato data binding (meccanismo di sincronizzazione automatica dei dati tra model e view. Essendo, in Angular, bidirezionale, ogni modifica al model si riflette sulla view e viceversa);

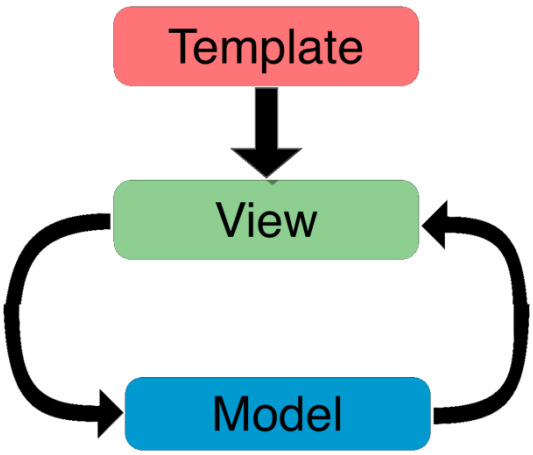

Figura 40 - two-way data binding

- {{espressione}}: valuta il valore corrente del modello dichiarato da ng-model;

- $scope: è un parametro passato dal framework al controller e condiviso con la view. Ha il compito di definire il modello dei dati e di esporlo alla view, in modo che, tutte le proprietà e i metodi di questo parametro, siano accessibili dalla view;

```
<script type="text/javascript">
angular.module("myApp", [])
    .controller("userController",
        function($scope) {
            $scope.utente = { nome: "Mario", cognome: "Rossi"};
            $scope.saluta = function() {
                return "Buongiorno " +
                       $scope.utente.nome + " " +
                       $scope.utente.cognome + "!"
            };
        });
</script>
```

Figura 41 - esempio utilizzo $scope

- Controller: viene definito da un costruttore JavaScript. Ha il compito di inizializzare l'oggetto $scope e attribuirgli il comportamento definendone i metodi;
- ng-controller: associa il controller alla view;
- ng-include: permette di includere, nella corrente view, altri file HTML (che a loro volta possono contenere codice Angular);
- ng-repeat: permette di generare un elenco di elementi HTML a partire da una collezione di dati (lista, array..). Inoltre, definisce una variabile di scope $index che rappresenta l'indice corrente inizializzata a zero. È utile anche nella creazione di tabelle, nella visualizzazione di mappe e altro.

```
$scope.persone = [
    { nome: "Andrea",   cognome: "Rossi",  citta: "Roma"    },
    { nome: "Marco",    cognome: "Verdi",  citta: "Milano"  },
    { nome: "Giovanni", cognome: "Neri",   citta: "Napoli"  },
    { nome: "Roberto",  cognome: "Gialli", citta: "Palermo" }
];
<ul>
    <li ng-repeat="persona in persone">
        {{persona.nome}} {{persona.cognome}}
    </li>
</ul>
```

Figura 42 - esempio di utilizzo ng-repeat

3.22. Chiamate REST

REST (REpresentational State Transfer) è un tipo di architettura software per la progettazione di applicazioni di rete, basata su un protocollo di comunicazione stateless, client/server e cacheble.

Le applicazione basate su REST vengono chiamate RESTful e utilizzano richieste HTTP per inviare, modificare e cancellare dati, effettuare query e per tutte e quattro le operazioni CRUD.

I principali vantaggi sono:

- Indipendenza dalla piattaforma;
- Indipendenza dal linguaggio;
- Basato su un protocollo di comunicazione standard (HTTP);
- Utilizzo anche in presenza di firewall (utilizza la porta 80).

REST non offre in alcun modo metodi per la gestione della sicurezza, per questo, spesso e volentieri, viene inclusa nell'header HTTP (ad esempio tramite l'inserimento di token tra i parametri). Per quanto riguarda la crittografia, REST può essere incluso nell'header di una richiesta HTTPS (secure sockets).

3.23. Il protocollo HTTP

HTTP (HyperText Transfer Protocol) è il protocollo principalmente usato come sistemata di trasferimento delle informazioni sul web (tipica architettura client/server). Le specifiche di questo protocollo sono gestite dal World Wide Web Consortium (W3C).

Il server HTTP, tipicamente, rimane in ascolto delle richieste dei client sulla porta 80. A livello di trasporto la richiesta viene gestita tramite protocollo TCP.

Il client, cioè il browser, esegue una richiesta tramite richiesta HTTP, e il server, la macchina su cui risiede il sito web, restituisce la risposta (risposta HTTP), una volta soddisfatta la richiesta, la connessione instaurata viene chiusa.

3.23.1. Request HTTP

```
GET /wiki.com/Pagina_principale HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.2;
Linux) (KHTML, like Gecko)
Accept: text/html, image/jpeg, image/png, text/*,
image/*, */*
Accept-Encoding: x-gzip, x-deflate, gzip, deflate,
identity
Accept-Charset: iso-8859-1, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: it.wikipedia.org
```

Figura 43 - esempio richiesta HTTP

Il messaggio è composto di tre parti:

- Request line
- Header
- Body

La request line si compone di:

- Metodo: i più utilizzati sono GET (per l'ottenimento del contenuto della risorsa indicata), POST (per l'invio di informazioni al server) e HEAD (simile a GET, ma usato per conoscere solo l'inestazione);
- URI: Uniform Resource Identifier, indica, in modo univoco, l'oggetto della richiesta;
- Versione del protocollo (ad esmpio, HTTP 1.1).

Per quanto riguarda gli header, i più comuni sono:

- Host: nome del server a cui si riferisce l'URL. È obbligatorio nelle richieste HTTP 1.1;
- User-Agent: identifica il tipo di client (ad esempio browser).

3.23.2. Response HTTP

```
HTTP/1.0 200 OK
Date: Mon, 28 Jun 2004 10:47:31 GMT
Server: Apache/1.3.29 (Unix) PHP/4.3.4
X-Powered-By: PHP/4.3.4
Vary: Accept-Encoding,Cookie
Cache-Control: private, s-maxage=0, max-age=0, must-
revalidate
Content-Language: it
Content-Type: text/html; charset=utf-8
Age: 7673
X-Cache: HIT from wikipedia.org
Connection: close
```

Figura 44 - esempio di risposta HTTP

Il messaggio di risposta è di tipo testuale ed è composto da tre parti:

- Status line
- Header
- Body

La riga di intestazione riporta un codice a tre cifre seguito da un breve messaggio, e riassume, in modo conciso, l'esito della richiesta:

- 1xx: messaggi informativi
- 2xx: messaggi di successo
- 3xx: messaggi di ridirezione
- 4xx: messaggi di errore client (richiesta sbagliata)
- 5xx: messaggi di errore server (problema interno al server, la richiesta non può comunque essere soddisfatta)

Per quanto riguarda gli header di risposta, i più comuni sono:

- Server: tipo e versione del server
- Content-Type: tipo del contenuto restituito (tipi MIME – Multimedia Internet Mail Extension) tra cui text/html, per i documenti HTML, e application/json per i documenti JSON

3.24. Bootstrap

Bootstrap è un insieme di elementi grafici, di impaginazione e stilistici (framework) che consentono di costruire pagine HTML completamente responsive, coerenti e funzionali. Il principale vantaggio di Bootstrap è l'adattamento alle dimensioni del dispositivo utilizzato che, attualmente, può essere di qualsiasi natura, dallo smartphone al classico personal computer.

Figura 45 – dispositivi moderni

I componenti di questo framework possono essere suddivisi in quattro classi:

- Impalcatura: contiene gli elementi CSS che permettono di definire il layout della pagina. Segue il Grid system, una griglia in cui possono essere definite righe e colonne che permette di incasellare i contenuti;
- CSS base: definisce gli stili predefiniti per i diversi elementi;
- Componenti: comprende gruppi di pulsanti, barre di navigazione, menù a discesa, un set di icone e tanto altro;
- JavaScript: contiene diversi plug-in jQuery per realizzare effetti comuni quali transizioni, popup, tab…

3.25. JSON

JSON (JavaScript Object Notation) è un tipo di formato molto utilizzato in ambito client/server per lo scambio di dati basato su JavaScript, ma completamente indipendente da questo.

Un oggetto JSON si presenta come gli oggetti letterali di JS.

```
{
  "home": "Html.it",
  "link": "http://www.html.it",
  "argomento": "Standard del web",
  "aree": [
    {
      "area": "CSS",
      "url": "http://css.html.it"
    },
    {
      "area": "Basic",
      "url": "http://basic.html.it"
    }
  ]
}
```

Figura 46 - definizione oggetto JSON

Possiamo quindi definirlo come una serie di coppie chiave/valore seprate da virgole e racchiuse tra parentesi graffe.

La chiave è definita come una stringa, il valore può essere qualsiasi tipo elementare o anche un array (delimitato dalle parentesi quadre) di oggetti JSON, o un singolo oggetto JSON (delimitato anch'esso da parentesi graffe).

L'utilizzo del formato JSON con Java è estremamente semplice: esistono librerie apposite per la serializzazione e la deserializzazione di questi elementi in classi che rispecchiano la struttura dell'oggetto.

```java
public class JsonObject{

    private String home;
    private String link;
    private String argomento;
    private List<Area> aree;

    //metodi get e set che consentono l'accesso ai dati

    public void setHome ( String home ) {
        this.home = home;
    }

    public String getHome ( ) {
        return this.home;
    }

    public void setLink ( String link ) {
        this.link = link;
    }

    public String getLink ( ) {
        return this.link;
    }

    public void setArgomento ( String argomento ) {
        this.argomento = argomento;
    }

    public String getArgomento ( ) {
        return this.argomento;
    }

    public void setAree ( List<Area> aree ) {
        this.aree = aree;
    }

    public List<Area> getAree ( ) {
        return this.aree;
    }
}
```

```java
public class Area{

    private String area;
    private String url;

    //metodi get e set che consentono l'accesso ai dati

    public void setArea ( String area ) {
        this.area = area;
    }

    public String getArea ( ) {
        return this.area;
    }

    public void setUrl ( String url ) {
        this.url = url;
    }

    public String getUrl ( ) {
        return this.url;
    }
}
```

Figura 47 – classe Java in cui deserializzare JSON figura 44

3.26. Telegram

Telegram è un'applicazione di messaggistica basata su velocità e sicurezza. Può essere utilizzata contemporaneamente su più dispositivi, infatti, a differenza di altre applicazioni di messaggistica (come, ad esempio, Whatsapp), è basata su cloud. Questa applicazione può essere utilizzata su smartphone, tablet e computer grazie alle app iOS, Android, il client web messo a disposizione e le applicazioni desktop supportate da Windows, OSX e linux.

Caratteristica importante, per il progetto descritto nei prossimi paragrafi e che ha spinto verso l'adozione di Telegram (per una prima implementazione di Qbot, che comunque è una struttura generica di bot), è il supporto ai bot, specifica che è propria di molte applicazioni di messaggistica, ma non di tutte (ad esempio Whatsapp non li supporta).

4. Il Qbot

4.2. I Database

In questo progetto vengono interrogati tre database: PUMA, FRMK, QBOT.

- PUMA: database in cui sono registrati gli utenti con i relativi dati, ad esempio username, password (criptata), ruoli (ad esempio amministratore), permessi (ad esempio creazione altri utenti), nome, cognome …
- FRMK: database utilizzato dal framework Quix, insieme dei framework visti nei capitoli precedenti
- QBOT: database creato per l'applicazione.

In tutti e tre i database si è usato MySQL.

Il database QBOT si compone di 4 tabelle riportate e spiegate di seguito:

- QBOT_BOT: tabella che si riferisce al model "Bot". Include:
  - idBot: identificatore univoco del bot in questione;
  - name: nome del bot;
  - jsScript: JavaScript serializzato che descrive il comportamento del bot;
  - createBy: creatore del bot;
  - createDate: data di creazione;
  - updateBy: nome di chi ha attuato l'ultimo aggiornamento;
  - updateDate: data dell'ultimo aggiornamento.

  Salva nel DB l'insieme dei Bot.

- QBOT_BOT_CHANNEL: fa riferimento al model "BotChannel". Include:
  - Id: identificatore univoco del BotChannel;
  - idBot: identificatore del Bot a cui è collegato;
  - name: nome del BotChannel;
  - type: tipo del canale (Telegram, Skype, Messanger Facebook …);
  - createBy: creatore del bot;
  - createDate: data di creazione;
  - updateBy: nome di chi ha attuato l'ultimo aggiornamento;
  - updateDate: data dell'ultimo aggiornamento;

- offset: intero che identifica il prossimo messaggio ricevuto dal Bot che deve essere processato.

Salva nel DB l'insieme dei canali dei Bot, ogni canale fa riferimento a un solo Bot e lo specializza (ad esempio in un Bot Telegram).

- QBOT_CHANNEL_REGISTRY: si riferisce al model "Channel_registry". Include:
  - Id: identificatore univoco del Channel_registry;
  - idBot: bot a cui fa riferimento;
  - username: username dell'utente;
  - channelType: equivalente al type del BotChannel;
  - idChannel: id della chat a cui si riferisce;
  - type: tipologia della chat (private, group …);
  - createDate: data di creazione;
  - lastAccesDate: data di ultimo accesso;
  - loginCounter: numero di accessi dell'utente;
  - authCode: codice che da l'autorizzazione all'utente di comunicare con il bot, viene assegnato dopo il login;
  - authCodeValidityDate: scadenza authCode dopo la quale bisogna rifare login;
  - session: sessione serializzata.

  Salva su DB le chat vere e proprie del QBOT_CHANNEL.

- QBOT_LOG_HISTORY: si riferisce al model "LogHistory" e include:
  - Id: identificatore univoco del "LogHistory"
  - idBot: bot a cui fa riferimento;
  - idChannelRegistry: Channel_registry a cui si riferisce;
  - eventDate: data dell'evento;
  - level: livello dell'evento;
  - message: messaggio dell'evento;
  - exception: eccezione scaturita dall'evento.

  Salva su DB i log delle chat, cioè ogni messaggio scambiato, o ogni eccezione sollevata.

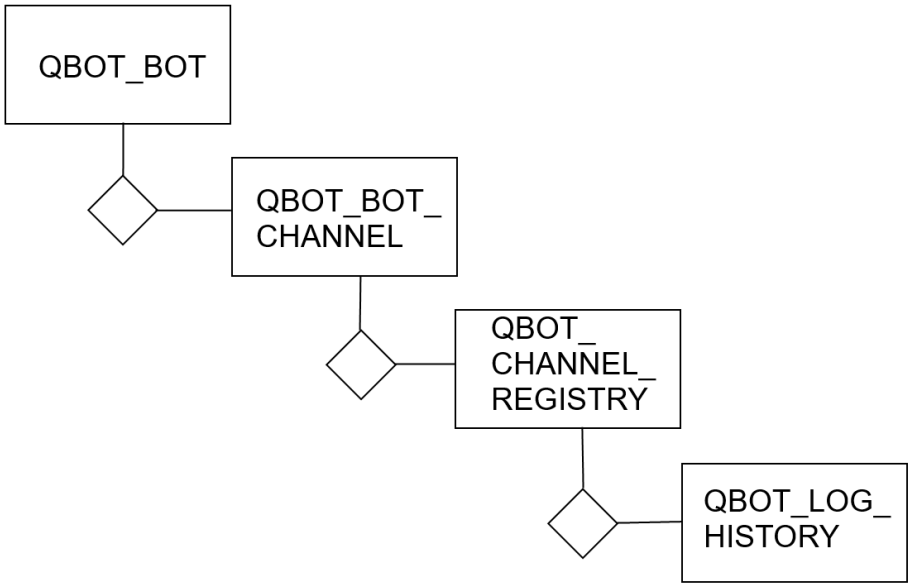

Figura 48 - struttura DB QBOT

```
create table QBOT_BOT
(
idBot nvarchar(36) not null,
name nvarchar(50),
jsScript longtext,
createBy nvarchar(50),
createDate date,
updateBy nvarchar(50),
updateDate date,
primary key(id)
);
```

```
create table QBOT_BOT_CHANNEL
(
id nvarchar(36) not null,
idBot nvarchar(36) not null,
name nvarchar(50),
type nvarchar(50),
createBy nvarchar(50),
createDate date,
updateBy nvarchar(50),
updateDate date,
offset int,
primary key(id)
);
```

```
create table QBOT_CHANNEL_REGISTRY
(
id nvarchar(36) not null,
idBot nvarchar(36) not null,
username nvarchar(50),
channelType nvarchar(50),
idChannel nvarchar(50) not null,
type nvarchar(50),
createDate date,
lastAccessDate date,
loginCounter integer,
authCode nvarchar(36),
authCodeValidityDate date,
session longtext,
primary key(id)
);
```

```
create table QBOT_LOG_HISTORY
(
id bigint auto_increment,
idBot nvarchar(36) not null,
idChannelRegistry nvarchar(36),
eventDate date,
level nvarchar(50),
message longtext,
exception nvarchar(50),
primary key(id)
);
```

Figura 49 - istruzioni creazione tabelle QBOT DB

I database, per poter essere interrogati, in questo progetto, sono stati inseriti nel server.xml. L'unico modo in cui sono stati interrogati è tramite le classi DAO e QbotManager che verranno analizzate nel prossimo capitolo.

4.3. Interrogazione del database tramite QbotManager e DAO

Grazie alle tecnologie utilizzate e spiegate nei capitoli precedenti, l'interazione con il database è stata gestita in modo piuttosto semplice.

Per prima cosa sono stati dichiarati il context e le resources (incluso il massimo numero di connessioni aperte) nel server.xml, per permettere a Tomcat di gestire le connessioni tramite DataSource. Il server.xml permette la gestione di interazioni con multipli database; nel caso di questo progetto, tre diversi DB sono stati interrogati (uno per il framework Quix, uno per la gestione degli utenti, uno per il Qbot vero e proprio).

Per quanto riguarda DAO, DataSources, Manager e Validatori, è stata gestita con l'aiuto dei bean di Spring tramite diversi file xml, poi importati nel file "qbot-spring-xml". Si è scelto di utilizzare diversi file xml per mantenere continuità con i progetti aziendali e per agevolare la modularizzazione del progetto.

Nel concreto sono stati utilizzati:

- un file xml per la dichiarazione dei bean DAO con relative relazioni con i DataSource;
- un file xml per la dichiarazione dei Datasource;
- un file xml per la configurazione della view (utilizzato in altri contesti del progetto);
- un file xml per i bean applicativi come lo UserContext, i Manager e gli Handler (più avanti saranno spiegati meglio);
- infine, un file xml per la validazione.

Ogni bean dichiarato nei file appena citati, tranne i bean DataSource, fa riferimento a una classe Java. Verranno qui brevemente esposte, le classi dei bean DAO e del Manager, classe a cui viene delegata l'interazione con il DAO.

4.3.1. I DAO

Ogni tabella costruita nei vari DataBase viene interrogata tramite DAO. Ogni DAO implementa, tipicamente, le operazioni CRUD, inoltre vengono specificate altre query che possano essere utili nell'applicazione.

Per quanto riguarda questo progetto sono stati scritti:

- BotDAO: classe java che si occupa delle principali query al database QBOT per quanto riguarda la tabella QBOT_BOT;
- BotChannelDAO: DAO che implementa le query verso la tabella QBOT_BOT_CHANNEL del DB QBOT;
- Channel_registryDAO: per quanto riguarda le interrogazioni della tabella QBOT_CHANNEL_REGISTRY;
- LogHistoryDAO: classe che interroga la tabella QBOT_LOG_HISTORY.

```
public Bot get(String idBot) throws DAOFinderException {
        Connection connection = null;
        PreparedStatement statement = null;
        ResultSet rs = null;
        try {
            // Compose the select query
            StringBuilder query = new StringBuilder(EOL);
            query.append("SELECT * FROM QBOT_BOT ").append(EOL);
            query.append("WHERE IDBOT = ?  ").append(EOL);
            // Query logging
            if (queryLog.isInfoEnabled()) {
                queryLog.info(query);
            }
            // Get connection
            connection = getConnection();
            // Prepare the statement
            statement = connection.prepareStatement(query.toString());
            // Set the parameters
            int p = 1;
            // Set the primary key
            super.setParameterString(statement, p++, idBot);

            // Execute the query
            long startTime = System.currentTimeMillis();
            rs = statement.executeQuery();
            long endTime = System.currentTimeMillis();
            long time = endTime - startTime;
            String msgTime = FrameworkStringUtils.concat("Query time: ", time);
            if (queryLog.isDebugEnabled()) {
                queryLog.debug(msgTime);
            }
            if (rs.next()) {
                Bot botModel = buildModelFromResultSet(rs);
                return botModel;
            }
            throw new DAOFinderException(FrameworkStringUtils.concat("Cannot find Bot on database with [idBot = ", idBot, "]"));

        } catch (SQLException ex) {
            String msg = FrameworkStringUtils.concat("Error on method get(String idBot) for Bot on database with [idBot = ", idBot, "]");
            if (log.isErrorEnabled()) {
                log.error(msg);
            }
            throw new SystemException(msg, ex);
        } finally {
            closeResultSet(rs);
            closeStatement(statement);
            closeConnection(connection);
        }
    }
```

Figura 50 - esempio di metodo di BotDAO

Come si può notare nella *Figura 50*, ogni metodo del DAO gestisce la connessione al DB. Apertura e chiusura della connessione, in realtà sono fittizie, vengono, infatti, delegate al DataSource che, invece di chiudere e aprire una nuova connessione per ogni chiamata, provvederà a mantenere in standby un pool di connessioni (come dichiarato nel server.xml) da cui poi prelevarne una da consegnare al DAO.

In realtà il DAO non viene interrogato direttamente dal Manager, ma dal DAOFactory, una classe che ha come variabili i vari DAO direttamente iniettati come resources (tramite l'annotazione *@Resources(name="nome_bean")*, essendo stati dichiarati come bean nei file di configurazione di Spring).

```
public class DAOFactory extends AbstractDaoFactory {

    @Resource(name = "botDAO")
    private BotDAO botDAO;

    @Resource(name = "botChannelDAO")
    private BotChannelDAO botChannelDAO;

    @Resource(name = "channel_registryDAO")
    private Channel_registryDAO channel_registryDAO;

    @Resource(name = "logHistoryDAO")
    private LogHistoryDAO logHistoryDAO;


    /* getter e setter per ogni variabile */

    //  ..........

}
```

Figura 51 - DAOFactory

4.3.2. Il QbotManager

Il Manager, chiamato QbotManager nel progetto descritto, è la classe che interagisce con il DAOFactory.

Ogni metodo del Manager, chiama un diverso metodo dei DAO, in modo da poter interagire con questi. I metodi del QbotManager, sono stati annotati con l'annotazione *@Transactional* di Spring, in modo che venga effettuato automaticamente il rollback in caso di eccezione durante la loro esecuzione, permettendo così l'integrità dei database.

4.4. I model

I model del progetto possono essere suddivisi in tre grandi categorie:

- I DBModel, che rispecchiano le tabelle del DB QBOT (Bot, BotChannel, Channel_registry e LogHistoryBot)
- Gli AbstractModel, utilizzati dai metodi degli Handler (UserBot, MessageBot, UpdateBot, WebhookInfoBot …)
- I TelegramModel che estendono le classi astratte dei model sopracitati e che rispecchiano le API di Telegram (TelegramUser, TelegramMessage, TelegramUpdate, TelegramWebhookInfo …)

La struttura prevede un futuro inserimento di altre categorie di model, come ad esempio i MessengerModel che estenderanno, come i TelegramModel, gli AbstractModel, seguendo, però, le API di Messenger Facebook.

Il primo grande gruppo, come già detto, rispecchia le tabelle del DB QBOT e serve per il salvataggio, recupero e update delle informazioni del DB.

Il secondo verrà utilizzato dal BotHandler, ChatHandler e ScriptEngineHandler per la gestione astratta, essendo Qbot una struttura generica di Bot, delle principali funzioni di un qualsiasi bot, come ad esempio l'invio dei messaggi, dei documenti, delle foto, il reperimento degli update (in caso di polling) o il controllo delle informazioni di webhook (url, certificato …).

Il terzo gruppo sarà utilizzato nel caso di un BotChannel di tipo Telegram, per poter ricevere informazioni, poter inviare messaggi (inclusi messaggi con foto, documenti, bottoni, tastiere …) e poter deserializzare, tramite il JSONDeserializer di flexjson, le informazioni ricevute (ad esempio i TelegramUpdate).

```java
@Table(name="QBOT_BOT")
public class Bot {

    @Column(length=50)
    @Id
    public String id;

    @Column(length=50)
    public String name;

    @Column(length=50)
    public String type;

    @Column
    @Lob
    public String jscode;

    @Column(length=50)
    public String createBy;

    @Column
    @Temporal(TemporalType.TIMESTAMP)
    public Date createDate;

    @Column(length=50)
    public String updateBy;

    @Column
    @Temporal(TemporalType.TIMESTAMP)
    public Date updateDate;

    @Column
    public Integer offset;

}
```

Figura 52 - esempio di DBModel

```java
public abstract class ChatBot {

    public abstract Boolean getAll_members_are_administrators();

    public abstract void setAll_members_are_administrators(Boolean all_members_are_administrators);

    public abstract Integer getId() ;

    public abstract void setId(Integer id);

    public abstract String getType();

    public abstract void setType(String type) ;

    public abstract String getTitle();

    public abstract void setTitle(String title);

    public abstract String getUsername();

    public abstract void setUsername(String username);

    public abstract String getFirst_name();

    public abstract void setFirst_name(String first_name);

    public abstract String getLast_name();

    public abstract void setLast_name(String last_name);

}
```

Figura 53 - esempio di AbstractModel

```java
import it.quix.qbot.core.model.ChatBot;

public class TelegramChat extends ChatBot {

    private Integer id;

    private String type;

    private String title;

    private String username;

    private String first_name;

    private String last_name;

    private Boolean all_members_are_administrators;

    // @Override
    // getter and setter

    @Override
    public String toString() {
        return "Chat{"
                + "id=" + id
                + ", type='" + type + '\''
                + ", title='" + title + '\''
                + ", username='" + username + '\''
                + ", first_name='" + first_name + '\''
                + ", last_name='" + last_name + '\''
                + ", all_members_are_administrator=" + all_members_are_administrators
                + '}';
    }
}
```

Figura 54 - esempio di TelegramModel

4.5. Webhook e Long Polling

4.5.1. Long Polling

Il Long Polling, tecnica che può essere utilizzata con la piattaforma Telegram (ad esempio con la piattaforma Messenger FB può essere utilizzato solamente il Webhook), è una variante del tradizionale polling, processo in cui il dispositivo di controllo aspetta un segnale dall'esterno in modo sincrono e interrogando ogni "tot" di tempo il dispositivo esterno per sapere se esistono degli aggiornamenti.

Il Long Polling richiede informazioni al server esattamente come nel caso del Polling, ma con l'aspettativa che il server possa anche non rispondere immediatamente, per cui si "mette in attesa" della risposta. Nel momento in cui la risposta viene ricevuta, viene effettuata una nuova richiesta al server, attraverso il metodo getUpdates().

4.5.2. Webhook

È un metodo per alterare il comportamento di una pagina o applicazione Web con chiamate di ritorno personalizzate (callback).

Quando l'evento accade, la fonte fa una richiesta HTTP alla URI configurata per il webhook.

I webhook vengono spesso utilizzati, come in questo caso, per scatenare l'aggiornamento di sistemi in modo continuativo e asincrono.

Per settare il webhook, in questo progetto, sono stati necessari un indirizzo HTTPS e una chiave pubblica di certificato.

Il Bot implementato fa uso della tecnica di Long Polling.

4.6. TelegramChatHandler

Il TelegramChatHandler è una classe che estende la classe astratta ChatHandler implementando le API Telegram. In quanto tale, implementa tutti i metodi, nell'ambito di Telegram, che servono per:

- La ricezione degli update

- Il controllo delle webhookInfo
- L'invio dei messaggi
- L'invio di foto, documenti, sticker, audio, video, contatti, luoghi, contatti …
- L'aggiunta di tastiere
- L'impostazione del webhook
- …

Per una descrizione più dettagliata vedere il capitolo sul funzionamento del bot.

4.7. ScriptEngineHandler

Si può definire lo ScriptEngineHandler come una classe intermedia tra l'intera architettura Java costruita e il vero e proprio comportamento del Bot, definito invece in JavaScript, per questo è una delle classi di maggiore importanza della struttura.

Questa classe ha un unico grande metodo, che permette l'invocazione e la compilazione degli script JavaScript salvati in memoria (all'interno del DB QBOT nella colonna JsScript della tabella QBOT_BOT).

4.8. Funzionamento

Per quanto riguarda il funzionamento del Qbot, ci si focalizzerà su una breve descrizione della creazione del bot, del reperimento degli updates e della procedura di login; per quanto riguarda gli altri metodi implementati si rimanda alle API Telegram, che sono pubblicamente distribuite.

Per prima cosa si è provveduto alla creazione del Bot Telegram scrivendo al BotFather, un Bot che permette la creazione di altri Bot. Il BotFather, attraverso vari comandi, setta il nome, lo username e il token, un identificatore univoco del Bot creato.

Ogni richiesta effettuata è una chiamata REST (sono supportate sia le GET che le POST) del tipo:

https://api.telegram.org/bot<token>/METHOD_NAME

Ci sono quattro modi diversi di passare parametri nelle richieste:

- Attraverso l'url (Query String);
- Application/x-www-form-urlencoded;
- Application/Json (ad eccezione dell'upload dei file);
- Multipart/form-data (per l'upload dei file).

Le risposte contengono un oggetto JSON composto da due coppie chiave-valore:

{

'ok':'true', (oppure 'false')

'result': {

…. (contenuto del risultato, solitamente un altro oggetto JSON)…

}

}

Esistono due diversi modi di impostare il Bot:

- Ricezione degli update in pollling: il Bot interroga ciclicamente il sistema per sapere se è arrivata qualche richiesta;

- Ricezione degli aggiornamenti tramite webhook: si imposta un webhook, un url intermedio che riceve gli aggiornamenti e che li spedisce al Bot. In questo caso il metodo getUpdates non funziona.

Per settare il webhook, si utilizza il metodo setWebhook proprio delle API di Telegram. Questo metodo richiede diversi parametri:

- url: una Stringa che indica un url HTTPS a cui spedire gli updates, per rimuovere il webhook si utilizza una stringa vuota;
- certificate: chiave pubblica del certificato che permette il controllo del certificato in uso;
- max_connections: numero intero che indica il numero massimo di connessioni HTTPS al webhook , consentito simultaneamente per la consegna degli aggiornamenti (un numero elevato permetterà una velocità maggiore, un numero inferiore, permetterà di limitare il carico sul server del bot;
- allow_updates: un array di stringhe che indica i tipi di aggiornamenti che si desidera ricevere per il bot.

Nel caso in cui si decida di lavorare in polling, gli aggiornamenti vengono richiesti tramite il metodo getUpdates, con i seguenti parametri:

- offset: indica l'identificatore del primo aggiornamento che si intende ricevere, se non impostato verranno ricevuti tutti gli aggiornamenti delle ultime ventiquattro ore;
- limit: il limite di update che possono essere ricevuti, di default vale cento;
- timeout: timeout in secondi per il polling;
- allowed_updates: array di stringhe che indica, come nel caso del webhook, i tipi di aggiornamenti che sono accettati.

Una volta ricevuti gli aggiornamenti, in uno o nell'altro modo, la Servlet passa la gestione al ChatHandler, che nel caso specifico è il TelegramChatHandler (oggetto che si occupa anche del reperimento degli updates) che, una volta spacchettato il messaggio, passa la gestione al BotHandler. Il BotHandler dopo aver salvato BotChannel, Channel_registry e LogHistory nel DB, passa la gestione al ScriptEngineHandler che renderà eseguibile lo script salvato nella tabella QBOT_BOT. Infine verrà eseguito lo script, richiamando i metodi del BotHandler, quali, ad esempio, sendLogin, sendMessage o sendDocument (che poi richiameranno i metodi propri del ChatHandler).

Nel caso di primo contatto con il Bot nell'ultima ora, verrà inviato il sendLogin, un metodo che richiederà all'utente di eseguire il login tramite un messaggio con tastiera incorporata, come mostrato nella Figura 56.

Figura 55 - messaggio di login

La richiesta di login viene inviata ogni volta che scade l'authCode, un token che viene rilasciato nel momento in cui si effettua il login correttamente.

Cliccando il pulsante sottostante il messaggio, si apre una pagina in un browser esterno (ad esempio Chrome), adibita al login attraverso CAS (Central Authentication Service).

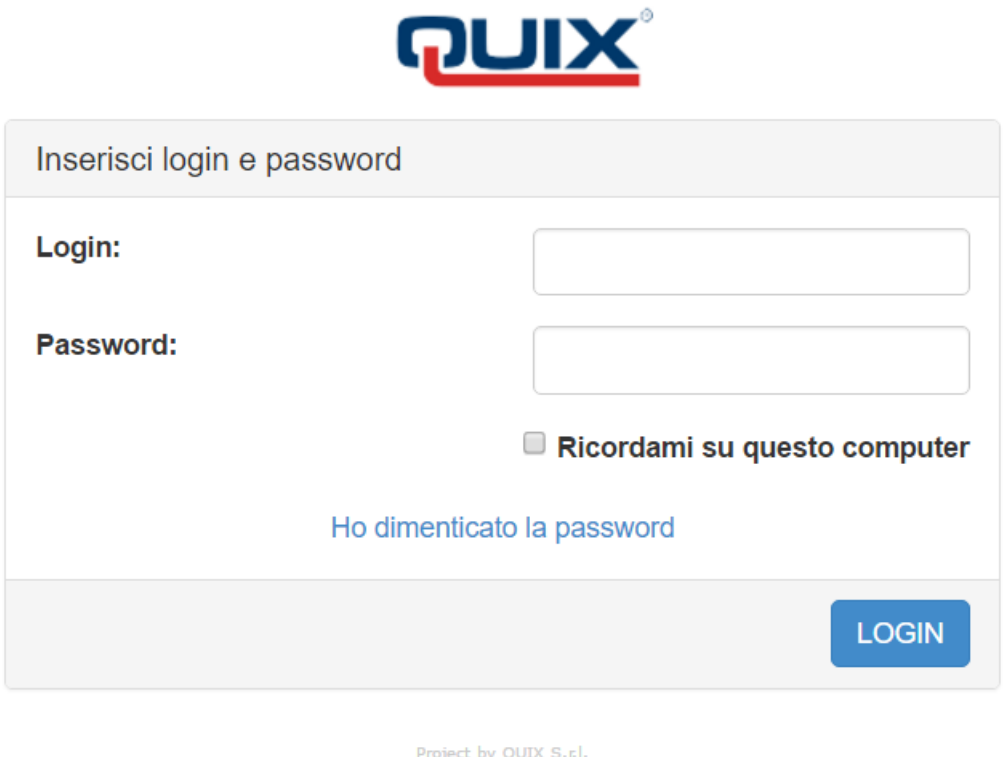

Figura 56 - pagina di login tramite CAS

Una volta effettuato il login, la Action HomeActionLogin (dichiarata tramite Struts2) genera un nuovo authCode (impostando la authCodeValidityDate all'ora corrente

più un'ora), salva BotChannel, Channel_registry e LogHistory e, infine, fa in modo che il Bot invii un messaggio di benvenuto all'utente.

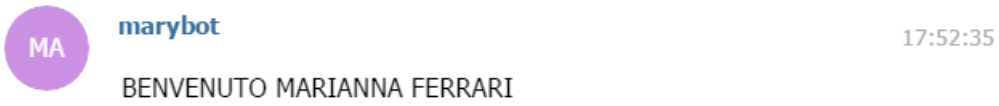

Figura 57 - messaggio di benvenuto

4.9. Le pagine JSP

Si è provveduto anche all'implementazione di una interfaccia grafica, scritta tramite l'utilizzo di pagine JSP, per la gestione dell'elenco di Bot e BotChannel associati del Qbot.

Come Home Page viene visualizzato l'elenco dei Bot con relativo elenco dei Botchannel associati (vengono visualizzati tipo e nome), tasto di creazione di un nuovo BotChannel associato, nome del creatore del Bot, nome dell'ultimo utente che ha aggiornato il Bot e tasto di modifica del Bot.

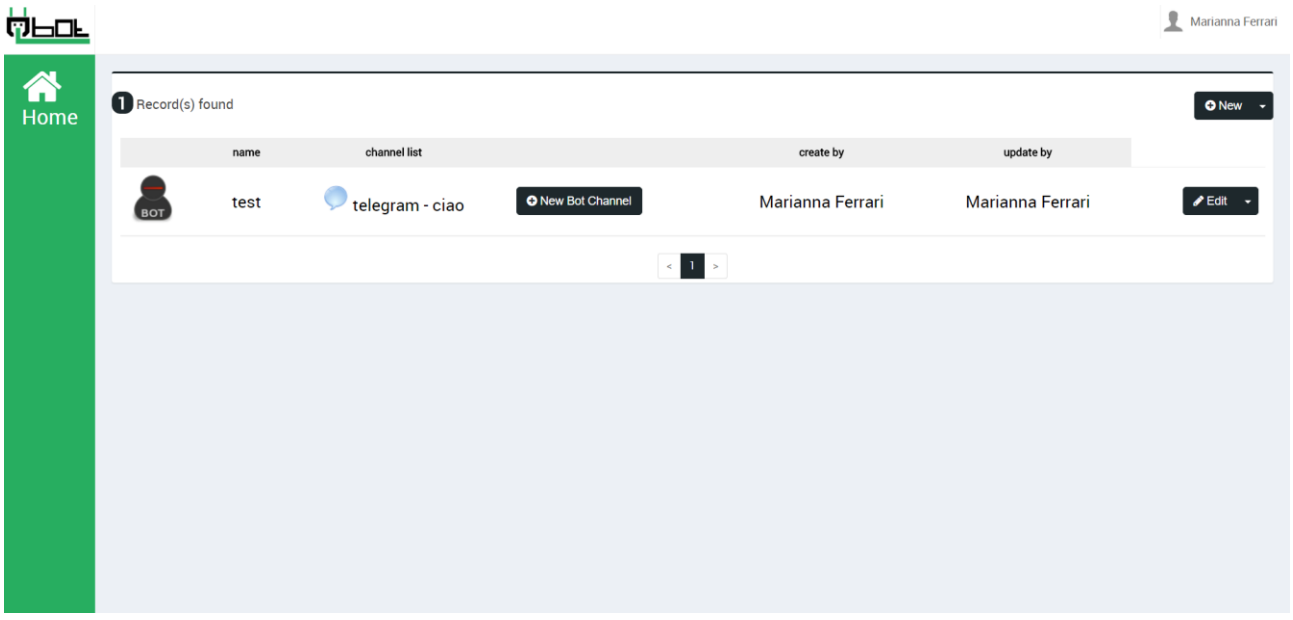

Figura 58 - Home Page

Cliccando il tasto “New Bot Channel”, si entra nella pagina di edit del Bot Channel per la creazione di un nuovo canale associato al Bot. Nella pagina vengono visualizzate diverse textArea:

- Name: nome del BotChannel;
- CreateBy: nome del creatore, questo campo sarà riempito automaticamente dal sistema;
- CreateDate: data di creazione, questo campo sarà riempito automaticamente dal sistema;
- UpdateBy: nome dell’utente che ha apportato l’ultima modifica, questo campo sarà riempito automaticamente dal sistema;
- UpdateDate: data dell’ultimo aggiornamento apportato, questo campo sarà riempito automaticamente dal sistema;
- Type tipo del canale (Telegram, Messenger, Skype), campo obbligatorio;
- Offset: offset degli update da ricevere;
- idBot: identificatore del Bot a cui è associato, questo campo viene riempito automaticamente e non è editabile.

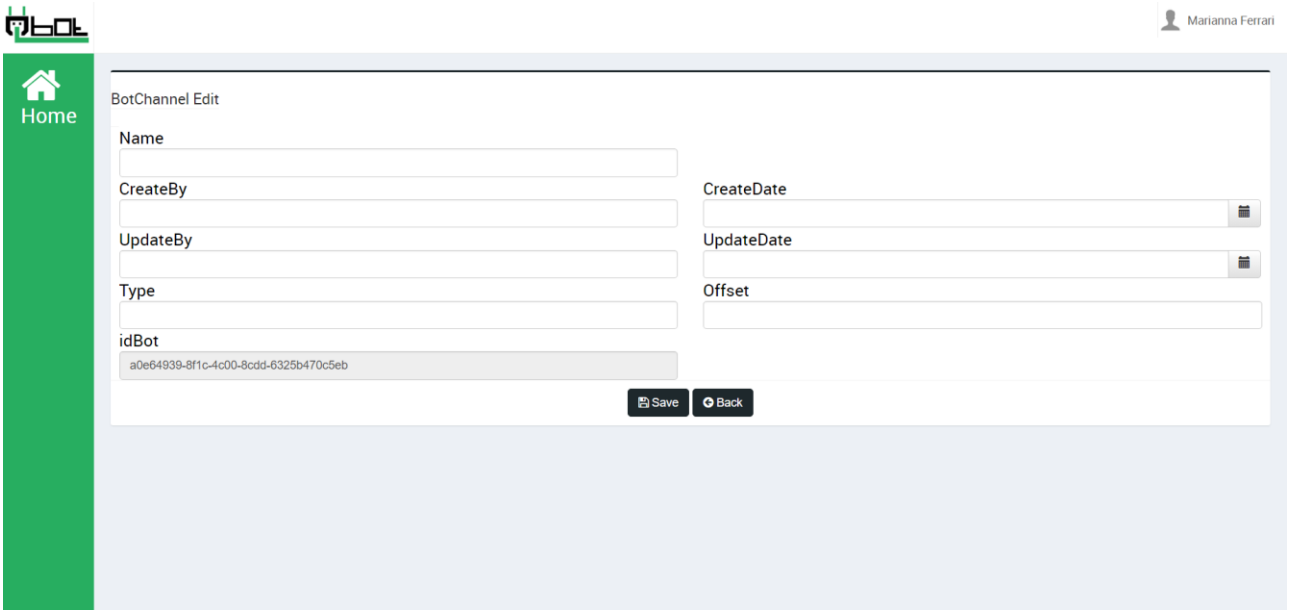

Figura 59 - Bot Channel Edit Page

Nel caso in cui si decida di editare il Bot, cliccando sul bottone della Home Page, Edit, verrà visualizzata la pagina raffigurata nella Figura 61.

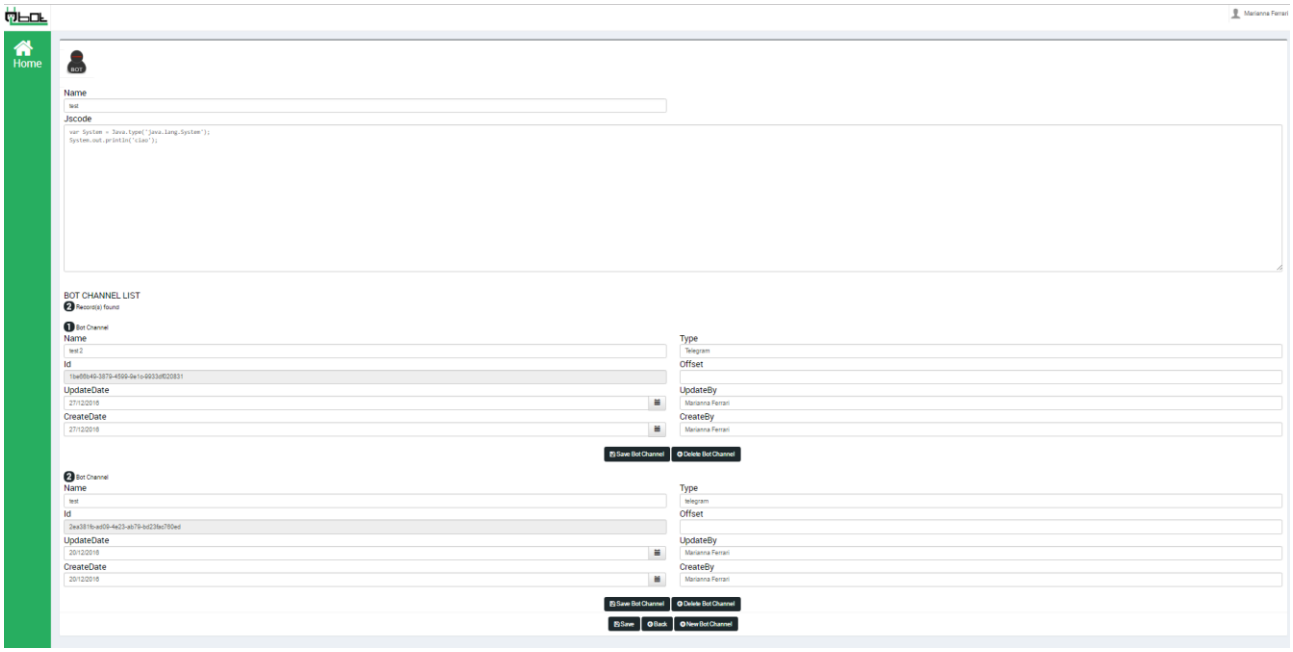

Figura 60 - Edit Bot Page

Come si può notare dalla Figura 61, questa pagina contiene:

- Name: nome del Bot;
- Jscode: codice JavaScript che verrà eseguito;
- Elenco dei BotChannel associati.

Per ogni modifica al BotChannel è possibile salvare l'elemento, in più è possibile cancellare un BotChannel, crearne uno nuovo, salvare l'intero contesto oppure eliminare tutte le modifiche apportate tornando alla pagina principale attraverso il tasto "back".

L'intera grafica dell'interfaccia del progetto è stata dichiarata in un file CSS, in modo da poterla modificare con il minimo cambiamento di codice delle pagine.

I bottoni scatenano le azioni dichiarate, grazie ad AngularJs, nel controller. In questo progetto si è scelto di usare un unico controller, qxBotController per tutte le pagine sopracitate. L'utilizzo di Angular, nel contesto di questo progetto, permette di elencare i vari BotChannel con relative caratteristiche, senza alcuna fatica.

4.10. Semplice implementazione JavaScript

```
function bot(contextBot, messageEntity) {
    var Request = Java.type('org.apache.http.client.fluent.Request');

    if(contextBot.getSession().get('user')==null ){

        //***** PROCEDURA DI LOGIN *****//

        contextBot.getBotHandler().sendMessage(contextBot, 'Benvenuto nel Bot di Demo di Quix. I servizi sono disponibili solo sotto autenticazione.');
        contextBot.getBotHandler().sendLogin(contextBot, 'Premi il tasto login per accedere.', 'EFFETTUA LOGIN');

    } else {

        //***** LOGIN EFFETTUATO *****//

        //***** PRIMO MESSAGGIO DOPO IL LOGIN *****//

        if(contextBot.isLoginJustMade() || messageEntity.getText() == '/login') {
            contextBot.getBotHandler().sendMessage(contextBot, 'Benvenuto ' + contextBot.getSession().get('user').getFullName());
            menu(contextBot);
            return;
          }

        //***** CONTROLLO SU TESTO RICEVUTO E CONSEGUENTI AZIONI *****//

        if(messageEntity.getType()=="TEXT") {

            var text = messageEntity.getText();

            if( text == '/ciao')
                contextBot.getBotHandler().sendMessage(contextBot, 'ciao a te, ' + contextBot.getSession().get('user').getFullName() + '!!');

            //***** ESEMPIO DI MESSAGGIO CON INLINEBUTTON *****//
            else if( text == '/cena')
                contextBot.getBotHandler().sendInlineButton(contextBot, 'vuoi uscire a cena cone me, ' + contextBot.getSession().get('user').getFullName() + '?',
                [[new KeyboardButtonBot('Si','','si'), new KeyboardButtonBot('No','','no')]]);

            //***** PROCEDURA DI LOGOUT *****//
            else if( text == '/logout')
                contextBot.getBotHandler().logout(contextBot, 'Arrivederci, ' + contextBot.getSession().get('user').getFullName() +
                    '! Scrivi un nuovo messaggio qualsiasi per avviare nuovamente la procedura di login.');

            //***** MESSAGGIO DI DEFAULT *****//
            else{
                contextBot.getBotHandler().sendMessage(contextBot, 'Non capisco cosa mi stai chiedendo');
                contextBot.getBotHandler().sendMessage(contextBot, 'scegliere comando da tastiera.');
            }
        }

        //***** CALLBACK QUERY DOPO L'INVITO A CENA *****//

        if(messageEntity.getType()=='CALLBACK') {

            var callback = messageEntity.getData();
            if(callback == 'si')
                contextBot.getBotHandler().sendMessage(contextBot, 'Perfetto! Non vedo l'ora di uscire con te');
            else if(callback == 'no')
                contextBot.getBotHandler().sendMessage(contextBot, 'Peccato, sarà per un\'altra volta');
        }
    }
}

//***** FUNZIONE MENU CON TASTIERA IN BASSO *****//

function menu(contextBot) {
    contextBot.getBotHandler().sendMessage(contextBot, 'All\'interno di questo bot sono disponibili i seguenti servizi: ');
    contextBot.getBotHandler().sendMessage(contextBot, '/ciao spedisce un normale messaggio di testo con saluto');
    contextBot.getBotHandler().sendMessage(contextBot, '/cena spedisce un messaggio di testo con InlineButton');
    contextBot.getBotHandler().sendMessage(contextBot, '/logout effettua il logout dal sistema');
    var KeyboardButtonBot = Java.type('it.quix.qbot.core.model.view.KeyboardButtonBot');
    contextBot.getBotHandler().sendMessageWithKeyBoard(contextBot,
        'Seleziona un elemento scrivendo il comando o selezionandolo dalla tastiera, o inviami un altro tipo di messaggio',
        [[new KeyboardButtonBot('/ciao'), new KeyboardButtonBot('/cena')],
        [new KeyboardButtonBot('/logout')]]);
}
```

Figura 61 - JavaScript di un semplice Bot Telegram

La Figura 61 rappresenta un semplice script di un Bot Telegram.

Il Bot di cui sopra, per prima cosa controlla l'accesso dell'utente. Nel caso in cui l'utente non sia ancora loggato, il Bot, spedisce un messaggio di login con un InlineButton collegato a url per login su CAS.

Dopo aver effettuato il login, l'utente riceve un messaggio di benvenuto con la presentazione del menu dei possibili comandi (presenti nella tastiera in basso).

Esistono diversi comandi:

- /ciao: comando che porta all'invio di un semplice messaggio con saluto;
- /cena: comando che causa l'invio di un messaggio con InlineButton con callback associata;
- /logout: comando per logout.

Ogni InlineButton del messaggio di risposta al comando "/cena" è associato a una callback. Una callback è come un comando scaturito automaticamente dalla risposta a un altro messaggio. Entrambe le callback di questo Bot inviano un semplice messaggio di risposta.

Il Bot descritto fa uso di Long Polling ed è implementato su piattaforma Telegram.

Non esiste ora come ora un esempio concreto da poter mostrare in sede di discussione, in quanto il prodotto Qbot è attualmente installato in server di test di clienti dell'azienda.

5. Ringraziamenti

Ringrazio in primo luogo la professoressa Po e il professor Beneventano per l'aiuto datomi per la stesura della tesi per avermi proposto il tirocinio nell'azienda Quix srl.

Un grazie immenso ai ragazzi della Quix srl, presso cui ho fatto il tirocinio, che mi hanno insegnato tanto e regalato momenti stupendi.

Ringrazio la mia famiglia per il sostegno e le opportunità datemi in questi anni.

Un grazie ai miei amici storici (Leo, Enri, Maso, War, Cavikkia, Glo e tutti gli altri) che non mi dimenticano mai, anche se ormai sono 20 anni che ci conosciamo.

Come non ringraziare il mio gruppo di ballo Country Tex che mi tiene compagnia quasi ogni sera tra balli e risate.

In ultimo, ma non per importanza, un grazie speciale al mio sorriso e amico più grande che anche a distanza non smette mai di darmi coraggio e credere in me.

E perché no, un sincero ringraziamento anche a chi mi ha messo i bastoni tra le ruote e mi ha permesso di riuscire in questa impresa nonostante tutto.

6. Bibliografia / Sitografia

www.softfobia.com

www.ilpost.it

www.udemy.com

www.core.telegram.org

www.html.it

www.stackoverflow.com

www.struts.apache.org/docs

www.it.wikipedia.org

www.alice.pandorabots.com

www.tomcat.apache.org/tomcat-7.0-doc/

www.w3schools.com

www.helloservlet.altervista.org

www.codingjam.it

www.wiki.metawerx.net

www.developer.mozilla.org

www.mrwebmaster.it

www.docs.angularjs.org/api

www.json.org

www.telegram.org