Scelte progettuali

Ticket System

Giugno 2020

Castaldo	Alessandro	M63000946
Allocca	Roberto	M63001045
Cesarano	Carmine	M63000948

Sommario

1. Processo di sviluppo

Nell'ambito di questo progetto è stato scelto un processo di sviluppo Agile, iterativo ed evolutivo di tipo UP. Dopo una prima **fase di ideazione**, durata 7 giorni, si sono susseguite **3 iterazioni** con una timebox fissata della durata di 10 giorni ciascuna.

FASE DI IDEAZIONE (08/05 – 14/05)

1. Workshop preliminare dei requisiti
2. Studio di fattibilità (analisi del framework)
3. Primi documenti di analisi (bozza SRS, Use case Diagram)
4. Prototipazione delle UI
5. Modellazione di una bozza del SDM
6. Definizione degli obiettivi per la fase successiva

ITERAZIONE 1 (15/05 – 24/05)

1. Raffinamento del SDM relativamente ai requisiti da implementare
2. Documenti di analisi comportamentale (sequence e activity) relativi ai requisiti di:
    a. Apertura ticket
    b. Visualizzazione dettagli
    c. Presa in carico ticket
    d. Chiusura ticket
3. Progettazione modello ER
4. Configurazione ambienti (application server Xampp, configurazione MySQL, configurazione framework Spring)
5. Design UI (riadattamento del template front-end):
    a. Pagina dashboard
    b. Pagina "dettagli ticket" utilizzata per visualizzare i dettagli, prendere in carico e chiudere un ticket
    c. Pagina "apri ticket" utilizzata per aprire un nuovo ticket.
6. Documenti di implementazione di dettaglio relativi ai requisiti da implementare (Class diagram "Controller", StateMachine, Sequence diagram, Package diagram)
7. Implementazione dei requisiti progettati
8. Testing delle funzionalità implementate
9. Build, deploy
10. Definizione degli obiettivi per la fase successiva

ITERAZIONE 2 (27/05 – 05/06)

1. Documenti di analisi comportamentale (sequence e activity) relativi ai requisiti:
    a. Visualizzazione lista cliente
    b. Visualizzazione lista ticket aperti
    c. Visualizzazione lista operatore
    d. Login
2. Configurazione aspetti di sicurezza del framework Spring legati al login
3. Design UI (riadattamento del template front-end):
    a. Liste ticket
    b. Pannello login
4. Documentazione di implementazione di dettaglio (sequence, raffinamento del package)
5. Implementazione dei requisiti progettati
6. Testing delle funzionalità implementate + testing di integrazione con le funzionalità implementate nell'iterazione precedente
7. Build, deploy

ITERAZIONE 3 (06/06 – 15/06)

1. Raffinamento documenti di analisi comportamentale di cui all'iterazione 1, con aggiunta delle funzionalità di notifica relative ai requisiti di:
    a. Presa in carico ticket
    b. Chiusura ticket
2. Configurazione aspetti relativi allo scambio di messaggi del framework Spring
3. Design UI (riadattamento del template):
    a. menu a tendina notifiche
4. Documentazione di implementazione di dettaglio (sequence, raffinamento del package)
5. Implementazione dei requisiti progettati
6. Testing delle funzionalità implementate + testing di integrazione con le funzionalità implementante nella iterazione precedente
7. Build, deploy

Al termine di ciascuna iterazione è resa dunque disponibile una versione del software opportunamente integrata, testata e funzionante. Per pulizia, è stato utilizzato un repository privato per mantenere traccia dei commit effettuati durante l'interazione, mentre nel repository pubblico sono stati effettuati soltanto i commit delle versioni funzionanti all'atto della consegna.

Repository di progetto: https://gitlab.com/gruppoPSSSTicketService/ticketsystemproject

1.1 Tecniche di sviluppo

Durante le sessioni di implementazione è stato fatto uso della tecnica di sviluppo agile "**Pair programming"** utilizzando la workstation di uno dei componenti a rotazione e condividendo lo schermo con i restanti due componenti del gruppo mediante la piattaforma *Teams*. Al termine di ogni sessione di sviluppo (giornata o metà giornata), è stato effettuato il commit della copia locale al workspace, su un repository privato.

1.2 Software a supporto dello sviluppo

Di seguito sono elencati i software utilizzati a supporto del processo di sviluppo software:

- **Microsoft Teams**: videoconferenza, condivisione schermo e scambio di materiale informativo
- **GitLab**: repository di progetto condivisa con tutti i componenti del gruppo
- **Xampp:** webserver che permette l'esecuzione di un application server Apache e il supporto al DBMS MySQL.
- **Visual Paradigm:** modellazione UML
- **Eclipse:** IDE a supporto dello sviluppo in Java
- **Balsamiq:** progettazione dei wireframe durante la fase di ideazione

1.3 Testing del software

In maniera continuativa, durante la fase di implementazione di ogni iterazione, è stato effettuato un **testing black box** sulle funzionalità via via implementate e rese eseguibili, utilizzando ove necessario, dei metodi stub per le precondizioni, e tenendo conto degli output attesi facendo riferimento ai requisiti definiti ad inizio iterazione.

2. Scelte di analisi del sistema software

2.1 Requisiti funzionali implementati

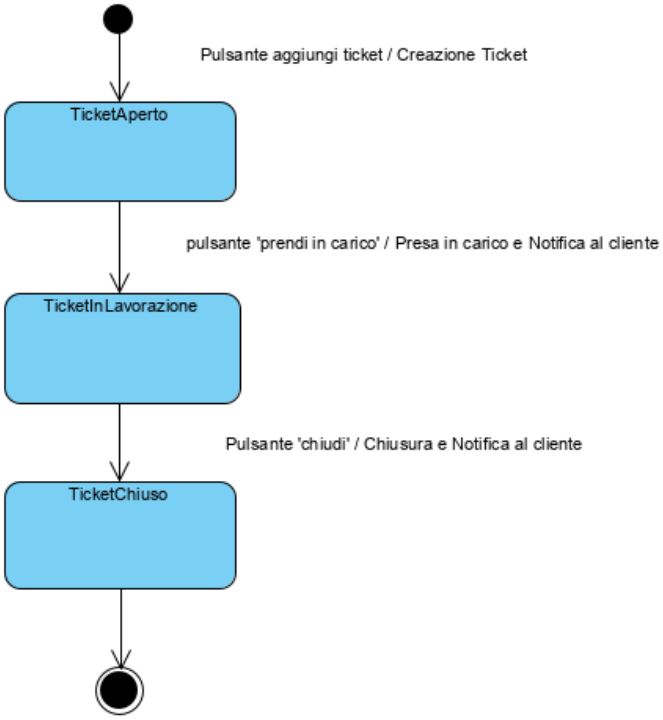
Di seguito sono riportati i casi d'uso del sistema implementati durante le tre iterazioni con le rispettive le priorità di sviluppo.

Caso d'uso	**Attore**	**Descrizione**	**Priorità (1-9)**	**Iterazione**
Apertura ticket	Cliente	Gestione apertura del ticket	9	1
Visualizzazione dettagli	Cliente, Operatore,Amministratore	Gestione visualizzazione dei dettagli ticket	9	1
Presa in carico	Operatore	Gestione presa in carico del ticket e creazione della notifica	8	1,3
Chiusura	Operatore	Gestione chiusura del ticket e creazione della notifica	8	1,3
Visualizzazione lista cliente	Cliente	Visualizzazione lista ticket aperti dal cliente	5	2
Visualizzazione lista ticket aperti	Operatore, Amministratore	Visualizzazione lista di tutti i ticket nello stato aperto	5	2
Visualizzazione lista operatore	Operatore	Visualizzazione lista di tutti i ticket presi in carico dall'operatore	5	2
Login	Cliente, Operatore	Login per clienti e operatori	4	2

Per una descrizione più dettagliata dei requisiti si rimanda al documento SRS Ticket System

2.2 Evoluzione dello stato del ticket

Relativamente alla gestione del ticket si è valutato di modellare la sua evoluzione con la macchina a stati finiti riportata in figura:



Il ticket può trovarsi in tre stati differenti:

- **Aperto**
- **In Lavorazione**
- **Chiuso**

Ad ogni transizione è associata un'azione precisa dettagliata nei Sequence Diagram di dettaglio opportunamente collegati. In particolare:

1. **Creazione**: All'atto della creazione, lo stato del ticket è Aperto.

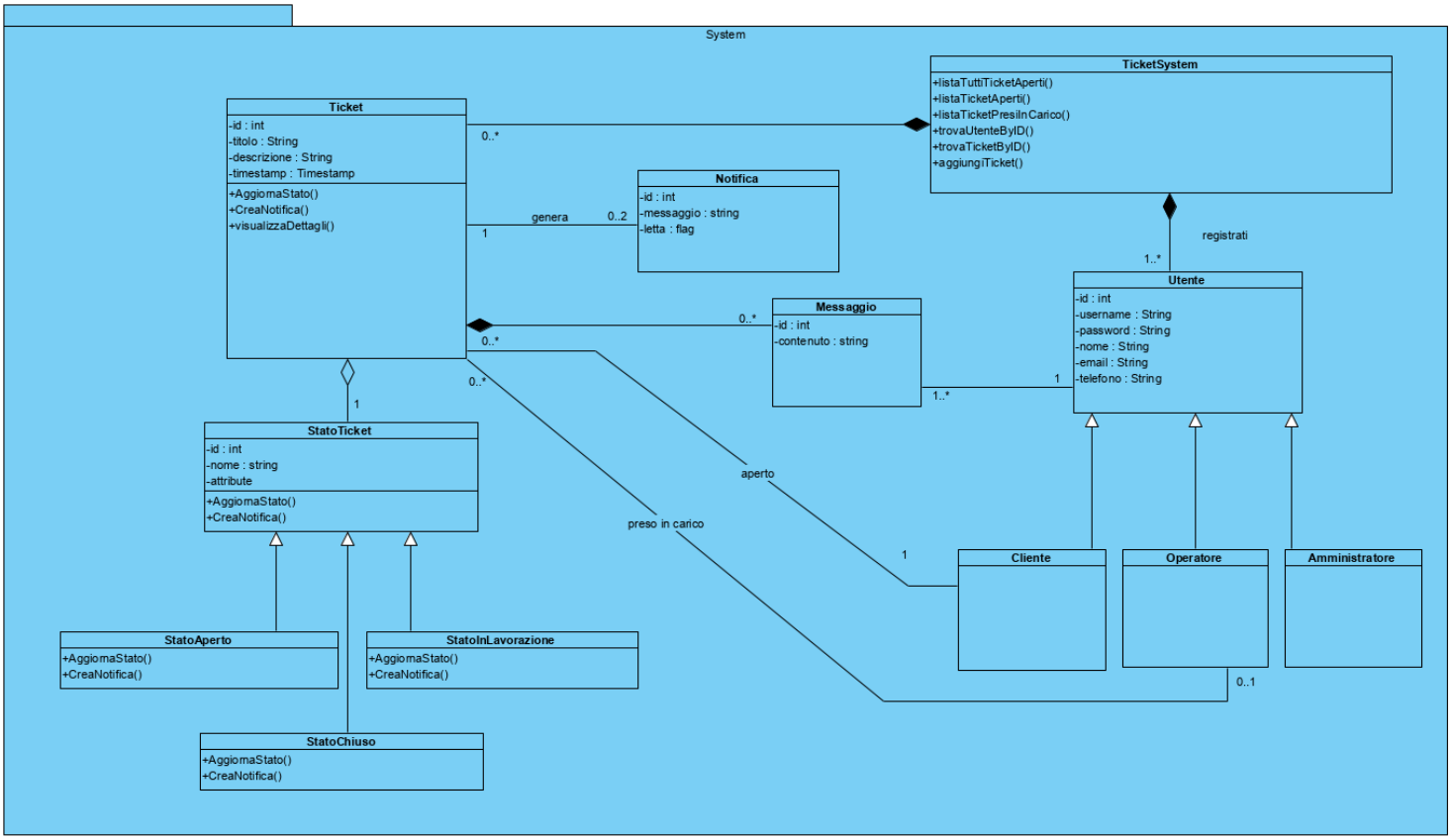
2. **TicketAperto->TicketInLavorazione**: il ticket viene preso in carico ed associato all'operatore e viene creata una notifica inviata all'utente.

3. **TicketInLavorazione->TicketChiuso**: il ticket viene chiuso dall'operatore e viene creata una notifica inviata all'utente.

Tale comportamento viene modellato utilizzando un design pattern State, associando al ticket uno stato e specializzandolo nelle tre modalità possibili. Ciò è chiarito meglio nel modello di dominio.

2.3 Domain model

Di seguito è mostrato il modello di dominio delle classi, così progettato nella fase di analisi:



Come si evince dal modello di dominio riportato, sono state effettuate le seguenti scelte di analisi:

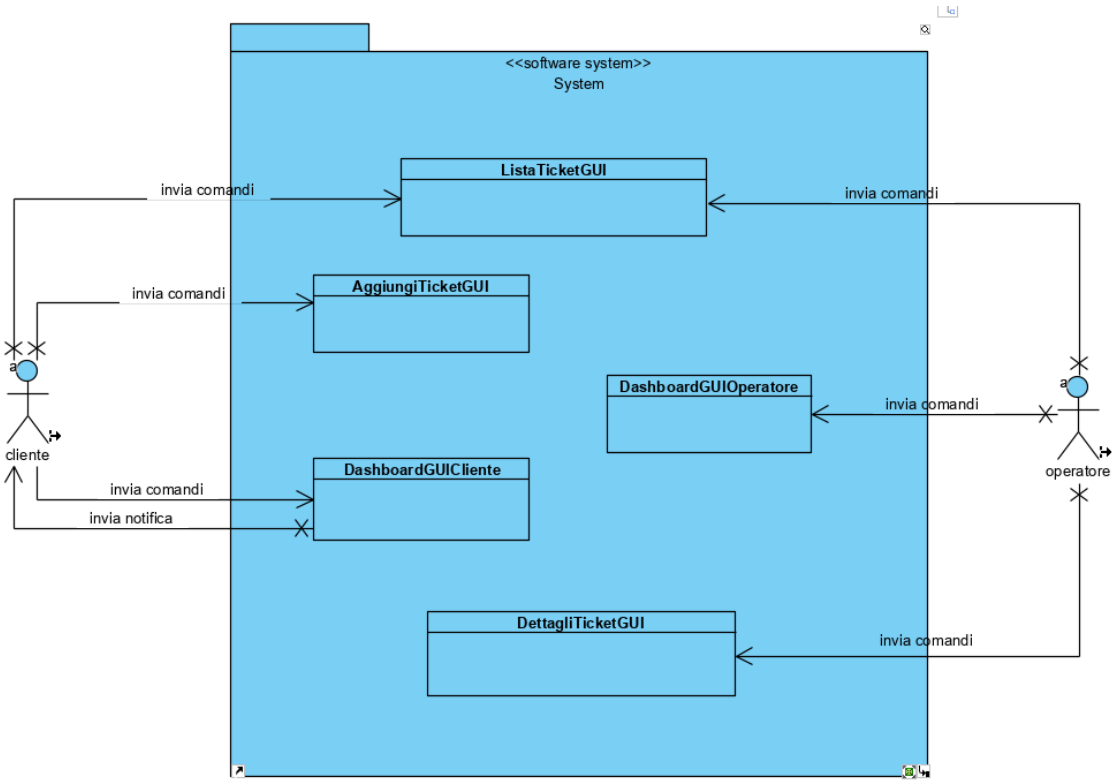
- E' stato applicato lo **State pattern** per la gestione dell'evoluzione di stato del ticket come precedentemente descritto;

- Si è resa necessaria una specializzazione dell'Utente in Cliente e Operatore per modellare la differenza di significato che essi ricoprono nell'ambito dell'associazione con il Ticket. In particolare un Ticket possiede un attributo cliente che rappresenta l'utente che lo ha aperto, e un attributo Operatore che rappresenta l'utente che lo ha preso in carico.

- E' stata introdotta la classe TicketSystem per ricoprire il ruolo di information expert degli utenti e dei ticket alla quale sono demandate le responsabilità di ricerca degli stessi e di aggiunta del ticket.

2.4 Diagramma di contesto

Di seguito è mostrato il diagramma di contesto con il quale vengono identificate le interfacce grafiche del sistema che permettono a ciascuna classe di utenti di interagire con il sistema.



2.5 Sequence e Activities diagram

Per meglio comprendere le scelte di analisi riguardanti il comportamento del sistema software si rimanda ai diagrammi comportamentali inclusi nel file .vpp allegato.

3. Scelte di design del sistema software

3.1 Framework Spring

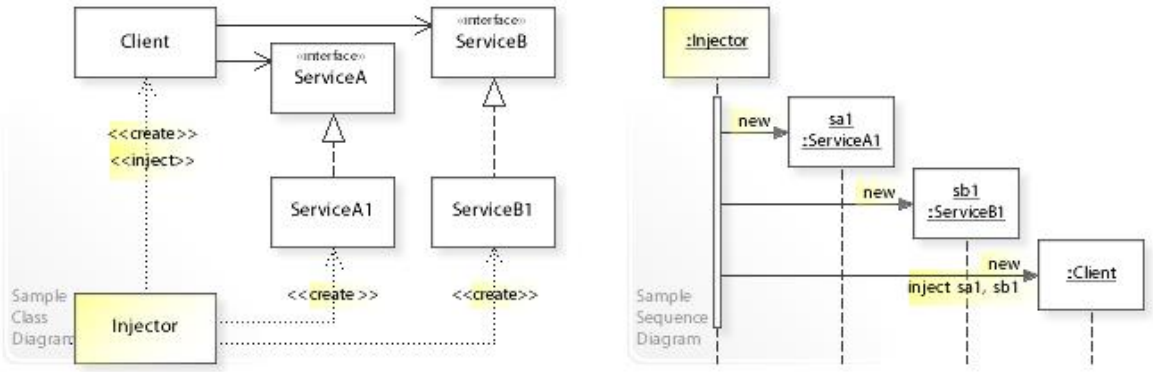
Per sviluppare lo strato di backend della web-application è stato utilizzato *Spring*, un framework open source per lo sviluppo di applicazioni su piattaforma Java.

L'ecosistema all'interno del quale le applicazioni Spring vivono viene definito *IoC container*. Quest'ultimo si basa sul principio di **Inversion of Control** (o IoC), tramite il quale, contrariamente a quanto accade utilizzando una libreria, in cui è il codice sviluppato a richiamare gli elementi definiti nella stessa, in questo caso il codice viene richiamato dai componenti del framework.

L'IoC container quindi si occupa di istanziare gli oggetti (beans) dichiarati nel progetto e di reperire ed iniettare tutte le dipendenze ad essi associate. Tali dipendenze possono essere i componenti del framework o altri beans dichiarati nel contesto applicativo.

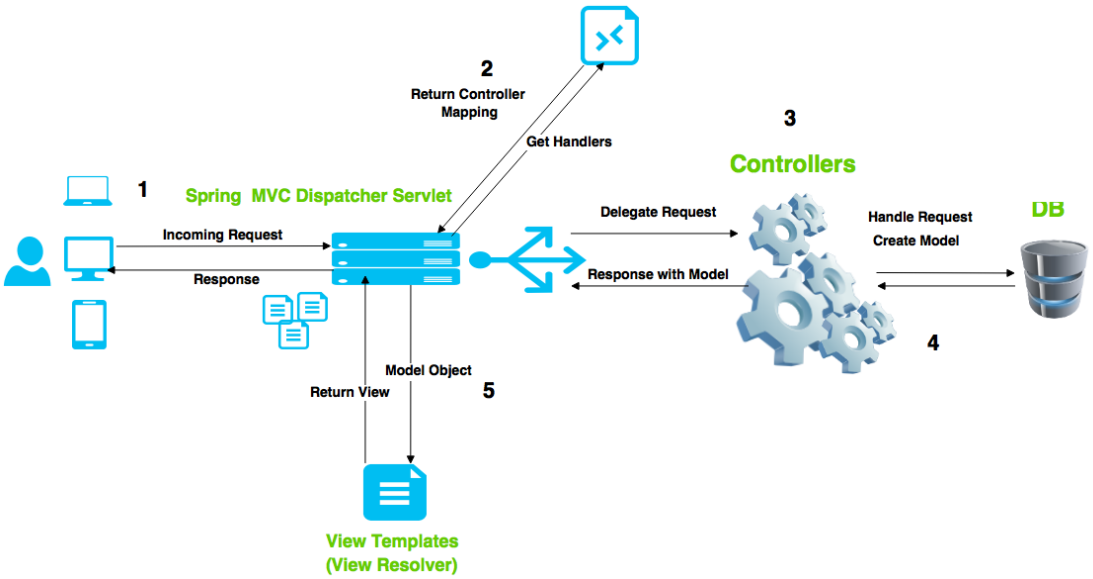
Le classi candidate ad essere Bean sono annotate come **Componenti** (Component), oppure specializzate come **Controller**, **Service** o **Repository** in base al layer in cui giacciono.

La tecnica utilizzata dal framework Spring per implementare il pattern dell'inversione di controllo è la **Dependency Injection** (DI). La DI prevede che tutti gli oggetti all'interno dell'applicazione accettino le dipendenze, ovvero gli oggetti di cui hanno bisogno, tramite costruttori o metodi setter. Non sono quindi gli stessi oggetti a creare le proprie dipendenze, ma esse vengono iniettate dall'esterno.

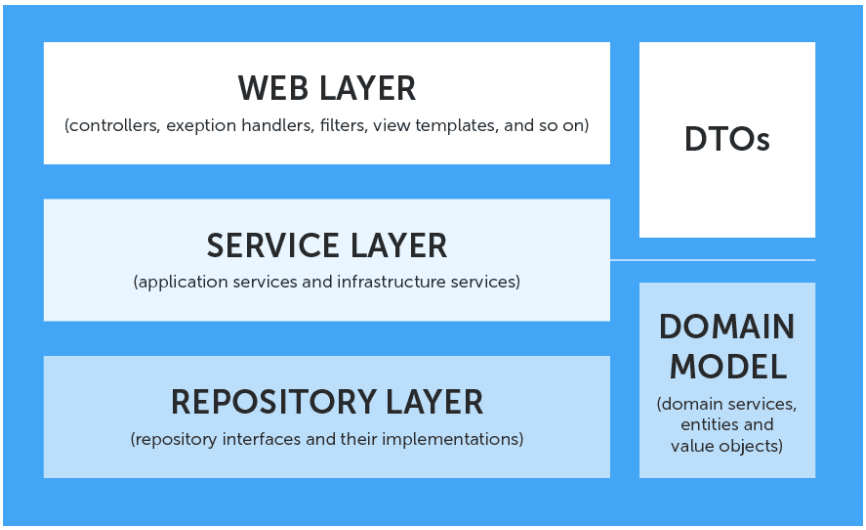


Il framework Spring permette lo sviluppo di applicazioni che seguono un pattern architetturale MVC. L'implementazione dell'MVC nelle applicazioni Web Spring prevede per lo strato Controller l'utilizzo di un altro pattern denominato "**Front Controller**". Tale pattern prevede un unico ingresso per le

chiamate HTTP rappresentato da una servlet, nella fattispecie chiamato **Dispatcher Servlet**. Il suo compito, dunque, è quello di prendere un URI in entrata e trovare il preciso gestore di tale richiesta (metodi sulle classi Controller) tramite l'**Handler Mapping**, un componente specializzato nell'interpretare le richieste e decidere a quale controller devono essere inoltrate. Il controller, dopo avere eseguito la sua logica applicativa ne restituisce i risultati al Dispatcher, che attraverso l'oggetto **View Resolver** fornisce la specifica View al client.



L'architettura di un'applicazione Spring si può schematizzare come un'architettura a livelli come mostrato di seguito:



- **Web Layer**: responsabile del processing degli input dell'utente e di restituire i valori corretti dell'elaborazione all'utente. Include dunque i controller (annotati come @Controller), che

implementano la logica di business, i gestori delle eccezioni, e le views. Dato che il web layer è l'entry point dell'applicazione, è necessario gestire correttamente gli accessi per proteggere le risorse.

- **Service Layer**: risiede al di sotto del web layer ed agisce come un interfaccia di transizione verso il livello sottostante, oltre che implementare parte della logica sui dati e la relativa sicurezza. I componenti di questo livello sono annotati come componenti di tipo @Service.
- **Repository Layer**: è il livello più basso dell'applicazione ed è responsabile di comunicare con il gestore della persistenza (DBMS). I componenti di questo livello sono annotati come Componenti di tipo @Repository.

3.2 Traduzione del Domain Model

La modellazione effettuata in fase di analisi è stata opportunamente adattata alle esigenze tecnologiche del framework Spring. In particolare, le modifiche apportate al modello di dominio, in fase di progettazione di dettaglio, risultano le seguenti:

- **Information Expert**: il ruolo di information expert, rivestito originariamente dalla classe TicketSystem diventa poco significativo con l'introduzione del service layer, all'interno del quale è presente una classe per ogni corrispettiva classe di model di cui si richiede persistenza, e che funge di fatto da information. Pertanto le navigabilità sono risolte evitando di sovraccaricare le classi del dominio con una funzione da information expert.

- **Utente**: la gerarchia che, nel modello di dominio, coinvolge la classe Utente, specializzandola in Cliente e Operatore, viene, in fase implementativa, risolta sopprimendo queste ultime due classi e introducendo un'ulteriore classe Ruolo con un id e una descrizione, associata all'Utente, cosicchè esso abbia un riferimento alla stessa. In caso contrario si sarebbe dovuto provvedere all'introduzione di un tipo enumerativo che modellasse i valori dell'attributo ruolo nella classe Utente, comprendendo un valore per ciascuna delle classi derivate e soppresse.

- **StatoTicket**: in fase implementativa, in ossequio alla modellazione di analisi, si procede a realizzare il ciclo di vita del ticket ricorrendo ad uno State pattern, utilizzando un'unica funzione, con comportamento polimorfico, definita nell'oggetto StatoTicket e ridefinita nelle derivate, per l'aggiornamento dello stato e la creazione della notifica.

3.3 Architettura software

3.3.1 Web controller layer

Si è optato, per la progettazione di questo livello, di designare un ridotto numero di classi controller al fine di realizzare una opportuna astrazione funzionale, un minimo grado di accoppiamento e un alto grado di coesione interna. I componenti di questo livello sono annotati come componenti di tipo **@Service**.

Questo livello espone all'esterno le funzionalità offerte dal sistema software e dunque include:

- **LoginController**: si occupa della gestione delle richieste utente relative a login e logout;
- **DashboardController**: Si occupa della gestione della richiesta che reindirizza alla homepage
- **TicketController:** si occupa della gestione delle richieste relative a creazione e aggiornamento dello stato del ticket, alla visualizzazione delle liste ticket e dei relativi dettagli e alla creazione della notifica.

Ognuna delle seguenti richieste è gestita da un metodo di uno dei Controller che provvede alla relativa gestione ed elaborazione (riportata nella colonna "Descrizione") e al reindirizzamento della corretta view.

Richiesta	**URL**	**Controller**	**Descrizione**
GET	/dashboard/index	DashboardController	Carica l'oggetto Account e la lista delle notifiche e reindirizza alla home page dell'utente loggato.
GET	/ticket/send	TicketController	Reindirizza alla pagina di "inserimento nuovo ticket"
POST	/ticket/send	TicketController	Provvede alla creazione e allo storage del nuovo oggetto Ticket
GET	/ticket/history_aperti	TicketController	Carica la lista di tutti i ticket aperti presenti nel sistema e reindirizza alla relativa pagina di visualizzazione.

GET	/ticket/history_cliente	TicketController	Carica la lista dei ticket aperti dal cliente loggato e reindirizza alla relativa pagina di visualizzazione.
GET	/ticket/history_operatore	TicketController	Carica la lista dei ticket presi in carico dall'operatore loggato e reindirizza alla relativa pagina di visualizzazione.
GET	/ticket/details/{id}	TicketController	Carica l'oggetto ticket avente l'id passato nella richiesta e reindirizza alla relativa pagina di dettaglio.
	/details/takecharge/{id}	TicketController	Permette il cambio di stato del ticket e il relativo salvataggio oltre che la creazione e il salvataggio della notifica e il suo invio al cliente.
GET	/login-panel/index	LoginController	Reindirizza al pannello di login
GET	/login-panel/login?error	LoginController	Reindirizza al pannello di login con un messaggio di errore nel caso di credenziali errate
GET	/login-panel/login?logout	LoginController	Reindirizza al pannello di login con un opportuno messaggio a seguito del logout
GET	/login-panel/accessDenied	LoginController	Reindirizza ad una pagina di errore nel caso in cui l'utente loggato non abbia i permessi per effettuare una determinata richiesta
GET	/login-panel/welcome	LoginController	Reindirizza alla dashboard a seguito del login

3.3.2 Service layer

Il Service rappresenta un layer di transizione verso il livello Repository, oltre che di gestione della logica sui dati. L'introduzione di questo layer è motivata dall'esigenza di rendere il layer sottostante completamente indipendente da qualsiasi tipo di logica. I componenti di questo livello sono annotati come componenti di tipo **@Service**.

Inoltre è utilizzato per astrarre meccanismi di gestione della sicurezza sui dati. Nella fattispecie, viene utilizzata una libreria del framework, **Spring Security**, che espone la classe *UserDetailsService*, opportunamente estesa dalla classe di service *UtenteService*. Ciò permette la gestione trasparente del meccanismo dei permessi in funzione del ruolo dell'utente.

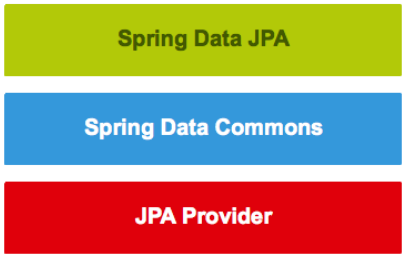
3.3.3 Repository layer

Il livello Repository si occupa della gestione della persistenza dei dati. A tal fine, si utilizzano le **Java Persistence API** (JPA), ovvero un framework che si occupa della gestione della persistenza dei dati di un DBMS relazionale nelle applicazioni che usano le piattaforme Java Platform, Standard Edition e Java Enterprise Edition.

Nell'ambito di questo progetto sono usate le **Spring Data JPA**, che implementano la specifica JPA tramite Hibernate, che funge da **JPA provider,** e consenteno l'interfacciamento con database relazionali.

Questa scelta determina due principali vantaggi

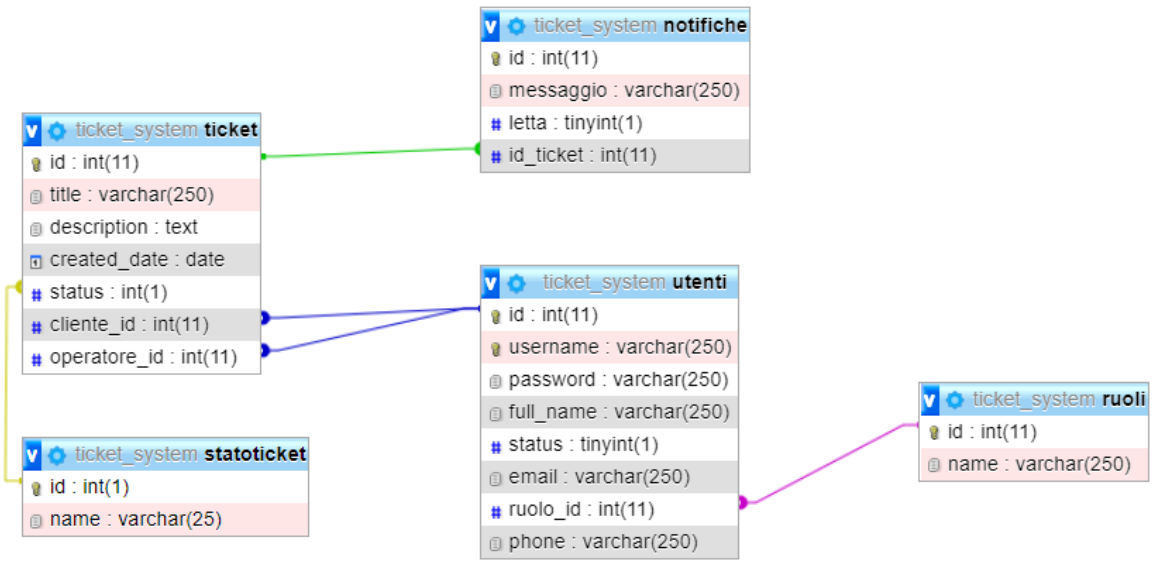
- Il software è indipendente dallo specifico DBMS utilizzato
- Semplifica l'implementazione di repositories JPA per l'accesso ai dati



I componenti di repository di questo livello sono annotati come componenti di tipo **@Repository**.

3.4 Altre scelte di progetto

3.4.1 Struttura del database



3.4.2 Gestione login/logout

Il framework Spring espone dei servizi a supporto del meccanismo di autenticazione utente e quindi necessari per gestire gli accessi dei diversi utenti anche in funzione del loro ruolo. In particolare, sono contenuti nel pacchetto "**spring-boot-starter-security**".
Estendendo opportunamente la classe WebSecurityConfigurerAdapter è possibile definire le regole di autorizzazione per i diversi tipi di utente:

1. *RUOLO_OPERATORE*: è autorizzato a chiedere la gestione delle richieste:
    a. `/dashboard/**`
    b. `/ticket/history_aperti`
    c. `/ticket/history_operatore`
    d. `/ticket/details`

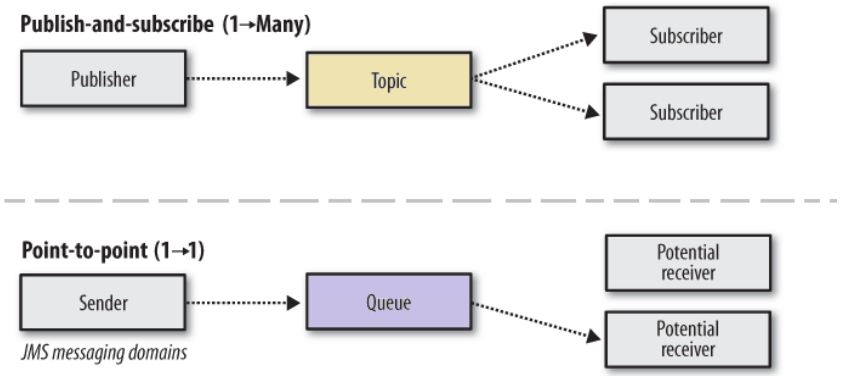
2. *RUOLO_CLIENTE*: è autorizzato a chiedere la gestione delle richieste
    a. `/dashboard/**`
    b. `/ticket/send`
    c. `/ticket/history_cliente`
    d. `/ticket/details`

Le credenziali degli utenti registrati sono mantenute nel database in formato crittografato. Il service relativo alla classe utente estende la classe astratta di Framework “UserDetailsService”. All’atto della pressione di un pulsante presente sulla pagina di login verrà reindirizzata la richiesta “/login/process-login”, gestita direttamente da Spring Security (opportunamente configurato in precedenza). Durante la gestione esso rileva la presenza di un’istanza “concreta” di UserDetailsService nel contesto dell’applicazione e utilizza il servizio per recuperare i dati necessari all’autenticazione. Se le credenziali sono corrette viene servita la richiesta “welcome”, viceversa l’errore viene gestito attraverso la richiesta “login?error” che restituisce un messaggio di errore.

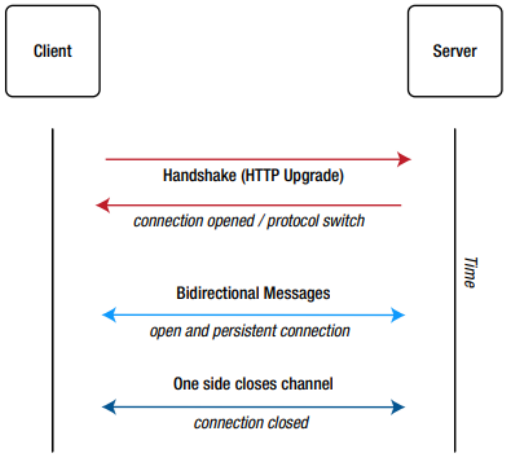
Spring Security inoltre prevede già l’esistenza di un gestore dell’URL “/process-logout” che consente all’utente di uscire dalla sessione corrente, riportandolo nuovamente sul pannello di login. Un pulsante nella dashboard utente è collegato alla suddetta richiesta.

3.4.3 Gestione delle notifiche

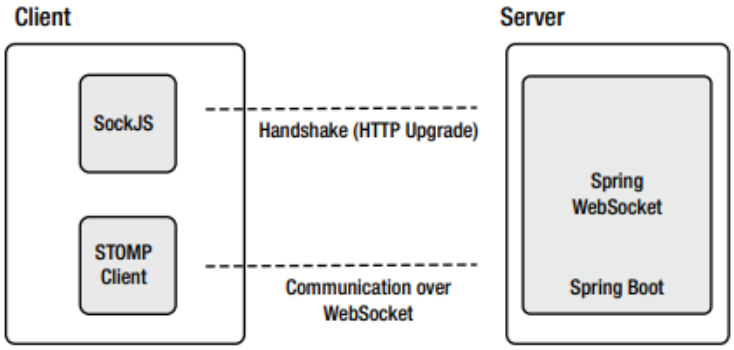
Per gestire il meccanismo delle notifiche, è stato usato un messaging pattern **point-to-point**, che modella una tipologia di messaggistica nella quale il canale assicura che un determinato messaggio possa essere consumato da un’unica entità ricevente. Tale gestione delle notifiche, dunque, non viene modellata mediante uno stile publish-subscribe, poichè nel caso specifico non vi è la necessità di inviare un messaggio in broadcast, essendo il destinatario della specifica notifica sempre unico.



Dal punto di vista implementativo, la gestione della comunicazione tra server e client è supportata dal meccanismo delle WebSocket. **WebSocket** è un protocollo di comunicazione full-duplex basato su TCP, che permette a due entità di essere trasmettitore e ricevente allo stesso tempo, attraverso un canale bidirezionale. Il protocollo inizia con un handshake (HTTP Request) e prosegue inviando messaggi come semplici frame su TCP.



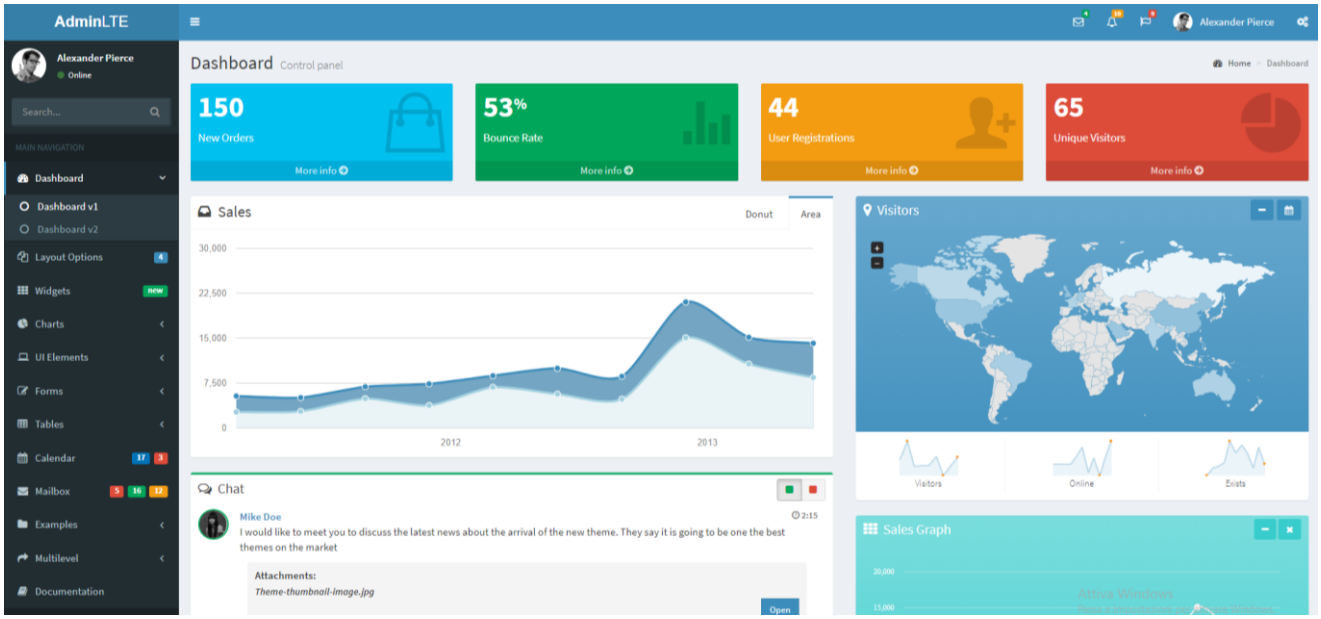
In particolare, viene utilizzato il modulo **spring-websocket** a supporto della gestione delle websocket server-side, mentre client-side, viene utilizzata la libreria javascript **SockJS**, inclusa nel file “stomp.js”.
In seguito all’handshake HTTP iniziale dove viene utilizzato SockJS, la comunicazione passa su TCP dove viene utilizzato per il formato dei frame scambiati, il protocollo **STOMP** (Simple/Streaming Text Oriented Message Protocol).



Si è scelto di rendere tutte le notifiche create dal sistema persistenti in modo tale da poter essere conservate per il cliente nel caso in cui esso non sia connesso al sistema nel momento in cui sono generate o nel caso in cui, pur essendo connesso, non ne abbia preso visione. Per questi motivi, le notifiche non prese in visione vengono riproposte ad ogni nuova sessione del cliente, poichè ricaricate al momento del login.

3.4.4 Frontend dinamico

Per quanto riguarda l'implementazione dei componenti di front-end, le dashboard dedicate all'utente 'cliente' e all'utente 'operatore', vengono ampiamente riutilizzati e riadattati i componenti del template open-source "*AdminLTE*". Si tratta di un template HTML ideato per dashboard e pannelli di controllo responsive basato sul framework *CSS Bootstrap3*, il cui design modulare permette una facile personalizzazione.



Il template è stato leggermente rimodulato in modo da presentare una struttura gerarchica adattiva. Se originariamente presentava, per ogni pagina da caricare, tutti i moduli dell'interfaccia, nella versione da noi adattata è presente uno scheletro, che esibisce sidebar e header, e un corpo che carica all'occorrenza la particolare view necessaria alla specifica funzionalità.

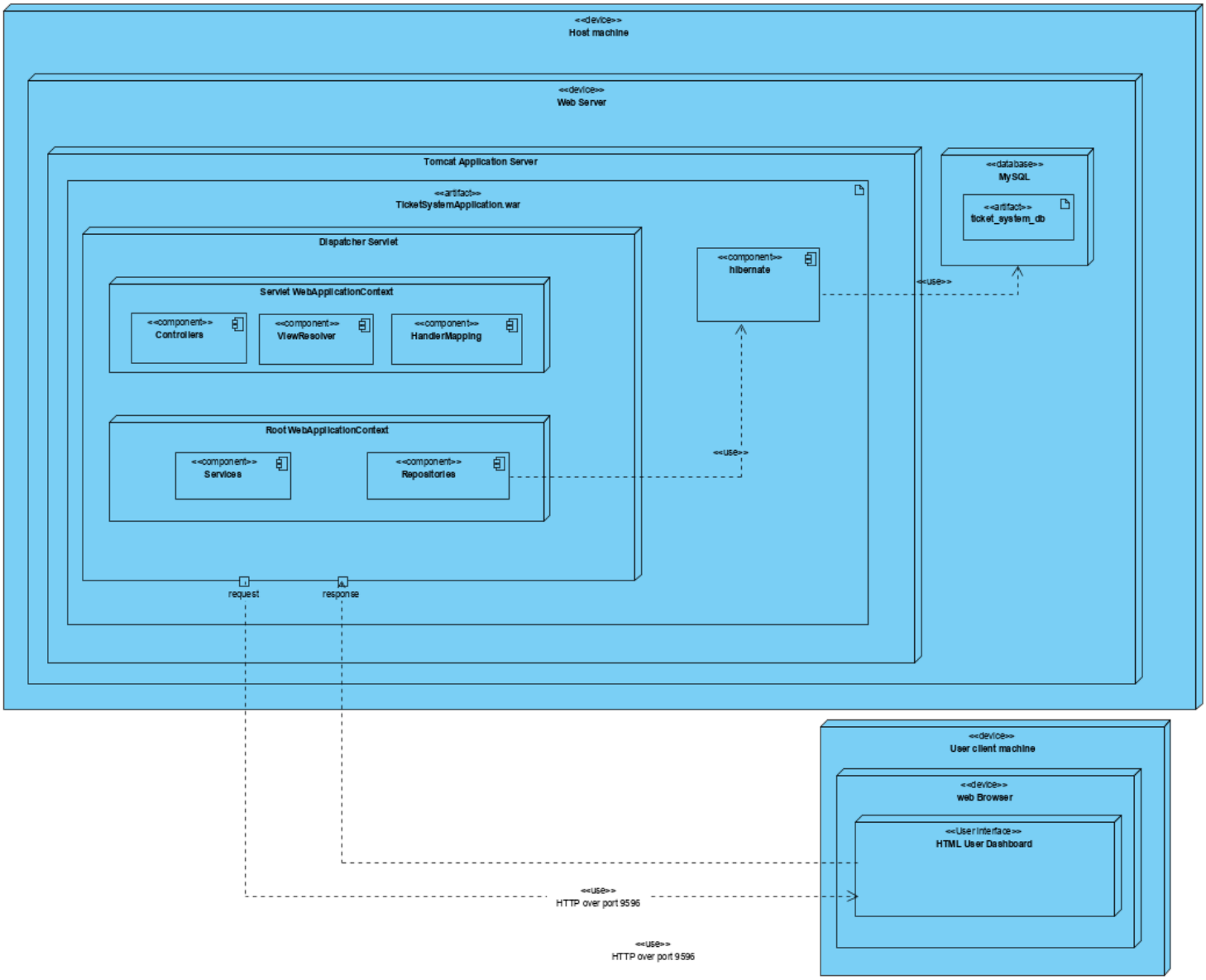
Il template inoltre è stato opportunamente modificato per essere adattivo rispetto al ruolo dell'utente che utilizza il sistema. Quindi, in generale, la view non è replicata staticamente per ogni diverso ruolo, ma è unica e contiene oggetti dinamici la cui visualizzazione dipende dal ruolo dell'utente loggato.

3.5 Deployment del sistema

3.5.1 Deployment Client Server

Il sistema software è dispiegato su due nodi:

- **Nodo Server**: dove è in esecuzione l'applicazione web Spring su webserver Apache e il Database MySQL, che implementa logica di controllo, elaborazione e persistenza
- **Nodo Client**: dove è in esecuzione l'interfaccia web su web browser che implementa unicamente la logica di presentazione



3.5.2 Gestione progetto software con Apache Maven

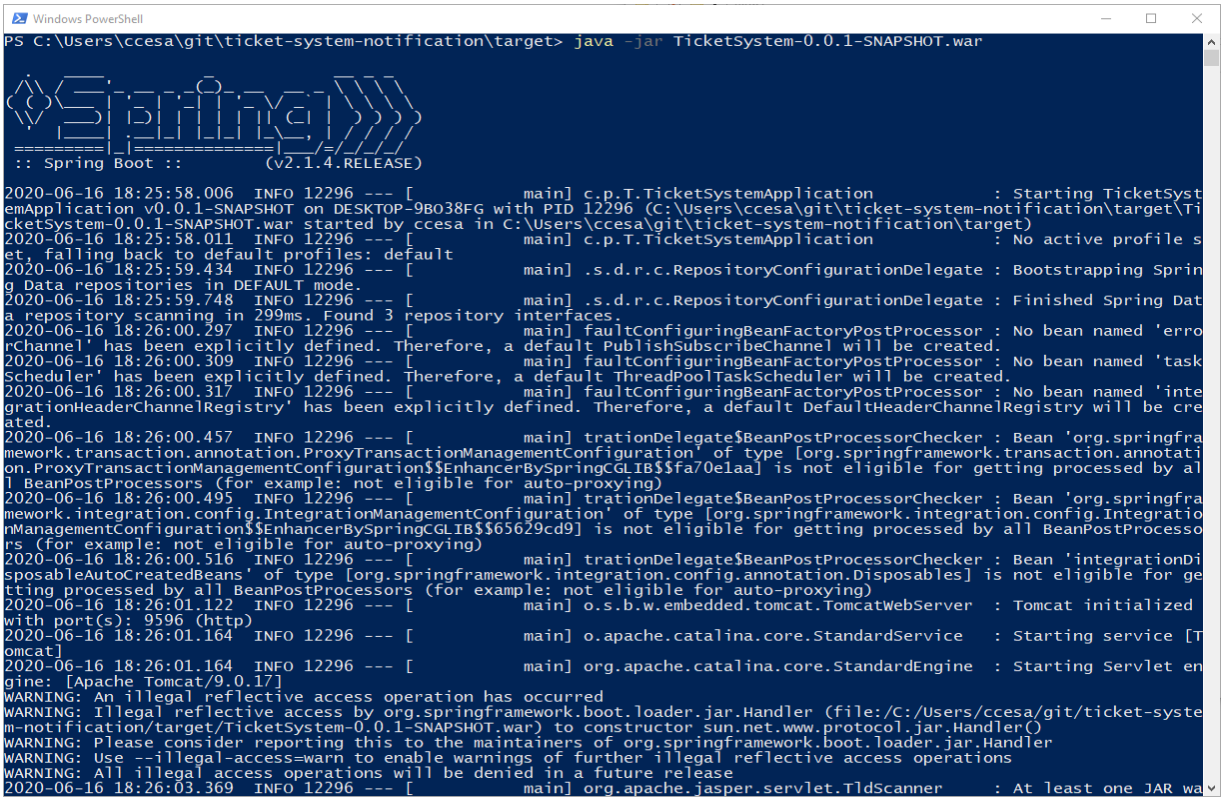
Apache Maven è uno strumento di gestione dei progetti che utilizza un **Project Object Model** (POM), ovvero un file XML che descrive le dipendenze fra il progetto e le varie versioni di librerie necessarie nonché le dipendenze fra di esse. Maven effettua automaticamente il download di librerie Java e plug-in Maven dai vari repository definiti scaricandoli in locale. Questo permette di recuperare in modo uniforme i diversi file JAR e di poter spostare il progetto indipendentemente da un ambiente all'altro avendo la sicurezza di utilizzare sempre le stesse versioni delle librerie.

4. Configurazione ed installazione del software

Al termine di ogni iterazione è stato effettuato il building della web app usufruendo del sistema di building Maven che genera l'archivio eseguibile .war tramite il comando "*clean install*"

```
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------< com.psss.TicketSystem:TicketSystem >----------------
[INFO] Building TicketSystem 0.0.1-SNAPSHOT
[INFO] --------------------------------[ war ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ TicketSystem ---
[INFO] Deleting C:\Users\ccesa\git\ticket-system-notification\target
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:resources (default-resources) @ TicketSystem ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO] Copying 7277 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:compile (default-compile) @ TicketSystem ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 24 source files to C:\Users\ccesa\git\ticket-system-notification\target\classes
[INFO] /C:/Users/ccesa/git/ticket-system-notification/src/main/java/com/psss/TicketSystem/configurations/WebSocketConfig.java: C:\Users\ccesa\git\ticket-system-notification\src\main\java\com\psss\TicketSystem\configurations\W
[INFO] /C:/Users/ccesa/git/ticket-system-notification/src/main/java/com/psss/TicketSystem/configurations/WebSocketConfig.java: Recompile with -Xlint:deprecation for details.
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:testResources (default-testResources) @ TicketSystem ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\ccesa\git\ticket-system-notification\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:testCompile (default-testCompile) @ TicketSystem ---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.22.1:test (default-test) @ TicketSystem ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-war-plugin:3.2.2:war (default-war) @ TicketSystem ---
[INFO] Packaging webapp
[INFO] Assembling webapp [TicketSystem] in [C:\Users\ccesa\git\ticket-system-notification\target\TicketSystem-0.0.1-SNAPSHOT]
[INFO] Processing war project
[INFO] Copying webapp resources [C:\Users\ccesa\git\ticket-system-notification\src\main\webapp]
[INFO] Webapp assembled in [76866 msecs]
[INFO] Building war: C:\Users\ccesa\git\ticket-system-notification\target\TicketSystem-0.0.1-SNAPSHOT.war
[INFO]
[INFO] --- spring-boot-maven-plugin:2.1.4.RELEASE:repackage (repackage) @ TicketSystem ---
[INFO] Replacing main artifact with repackaged archive
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ TicketSystem ---
[INFO] Installing C:\Users\ccesa\git\ticket-system-notification\target\TicketSystem-0.0.1-SNAPSHOT.war to C:\Users\ccesa\.m2\repository\com\psss\TicketSystem\TicketSystem\0.0.1-SNAPSHOT\TicketSystem-0.0.1-SNAPSHOT.war
[INFO] Installing C:\Users\ccesa\git\ticket-system-notification\pom.xml to C:\Users\ccesa\.m2\repository\com\psss\TicketSystem\TicketSystem\0.0.1-SNAPSHOT\TicketSystem-0.0.1-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  02:06 min
[INFO] Finished at: 2020-06-16T18:20:49+02:00
[INFO] ------------------------------------------------------------------------
```

L’esecuzione è subordinata all’opportuno settaggio dell’ambiente di esecuzione, attraverso la piattaforma Xampp, del webserver Apache e del DMBS MySQL sul quale è stato importato il database ‘ticket_system’.



Il sistema è fruibile mediante qualsiasi macchina collegata alla stessa rete locale dove è ospitato il web server virtuale, attraverso un qualunque web browser raggiungendo l’indirizzo ‘*/localhost:9596*’