ODD: Object Design Documentation

ScheduFIRE

Riferimento	SDD: System Design Documentation
Versione	1.1
Data	13/12
Destinatario	Top Management
Presentato da	Annunziata Giusy, Bombardelli Emanuele, Bruno Biagio, Cipolletta Ciro, Giuliano Alfredo, Labanca Nicola, Perillo Francesca, Sottile Eugenio
Approvato da	

Revision History

Data	Versione	Cambiamenti	Autori
12/12/2019	0.1	Aggiunta e revisione paragrafi 1.1, 1.2 e 1.3	Bombardelli Emanuele Giuliano Alfredo
13/12/2019	0.2	Stesura completa e Quality Revision	Bombardelli Emanuele Bruno Biagio Cipolletta Ciro Labanca Nicola Sottile Eugenio
18/12/2019	1.0	Reiterazione dei Diagrams	Sottile Eugenio Giuliano Alfredo
13/01/2020	1.1	Modifica dei componenti off the shelf	Emanuele Bombardelli

Sommario

1. Introduzione

1.1 Design Trade-Offs

Nel fare l'Object Design del software, si sono individuati i seguenti trade-offs:

Comprare vs Programmare:
Nel decidere se comprare o programmare specifiche funzioni nel programma (con preferenza specifica sulla seconda per testing e debugging più semplice ed affidabile), si cercherà di capire se il componente off-the-shelf in questione renderà buona parte del lavoro completo, altrimenti si preferirà programmare da zero, poiché è più facile costruire dalla base e non aggiungere e rimuovere funzionalità su di una componente con la quale non si ha confidenza.

Comprensibilità vs Tempo:
Si farà enfasi al produrre una documentazione ed un codice comprensibile per rendere il lavoro in parallelo il più semplice possibile ed il prodotto più manutenibile, anche se questo renderà il tutto più costoso in termini di tempo. Nel paragrafo 1.3 verranno mostrati le linee guida da rispettare.

Manutenibilità vs Performance:
La manutenibilità è chiave per il software proposto, in quanto si è prevista l'aggiunta di diverse funzionalità in futuro, e perché il software proposto deve essere affidabile il più possibile in ogni momento del suo ciclo di vita, considerato che dovrà essere proposto ad un servizio ministeriale. Per questo motivo la manutenibilità è preferita alle performance.

1.2 Componenti off-the-shelf

Per il software che si vuole realizzare faremo uso di componenti *off-the-shelf*, ossia componenti già disponibili sul mercato. Si è deciso di utilizzare *Apache Maven* come gestore delle Dependencies per evitare problemi di compatibilità nel lavorare al progetto (tra i quali import di vari file .jar). Per selezionare periodi di tempo in modo interattivo si è deciso di utilizzare Litepicker (link). Per mandare le mail si utilizzerà *Javax Mail*. Lato front-end, per il passaggio al back-end, si utilizzeranno JQuery e JSON. Infine, per il testing, si utilizzerà *JUnit Jupiter* (JUnit 5) e i mock offerti dallo *Spring Framework*.

Come server si è deciso di optare per *Apache Tomcat 9*.

Per la gestione dei dati persistenti, si utilizzerà *MySQL*.

1.3 Linee guida per la documentazione dell'interfaccia

HTML

Per rendere più comprensibile il codice tra i membri dello sviluppo Front-End, si seguirà questo esempio:

```
<!DOCTYPE html>
<html>
<head>
<!-- Corpo dell'head -->
</head>
<body>
        <!-- Corpo del body -->
        <p>Questa stringa è particolarmente lunga quindi va scritta
                su più linee per rendere il codice utilizzabile e
                leggibile su ogni schermo e    macchina. Notare come le
                linee successive alla prima iniziano allineate al primo
                TAB dopo l'istruzione di inizio.</p>
        <div>
                <div>
                        <p>I blocchi innestati ad altri blocchi scendono
                                a cascata utilizzando incrementalmente il
                                TAB</p>
                </div>
        </div>
</body>
</html>
```

Java e Javadoc

Il seguente blocco di codice esempio include come vanno identate le classi ed i relativi Javadoc.

```
package example;

import package-da-importare;
import classe-da-importare;

/**
 * Breve descrizione della classe. Per il Javadoc verrà usata la
 * lingua italiana, mentre nel codice è indifferente la lingua,
 * con preferenza per la lingua inglese per permettere lo
 * sviluppo anche a livello internazionale. Notare come il primo
 * commento inserito è dopo tutti gli import.
 * @author Autore della classe
 *
 */
public class Class {
        //Commento in-line di tipo 1.

        /*
         * Commento in-line di tipo 2.
         * Questo tipo di commento è da usare per commenti su più linee
         */

        //Variabili di istanza
        private tipo_parametro1 parametro1;
        private tipo_parametro2 parametro2;

        //Costanti
        private static final int PARAMETRO_STATICO = 1;
        public static final int COSTANTE_PUBLIC = 2;

        //Costruttore/i
```

```
        /**
         * Descrizione del primo costruttore
         */
        public Class() {
                //Corpo del costruttore
        }

        /**
         * Descrizione del secondo costruttore
         * @param parametro1 cos'è il primo parametro?
         * @param parametro2 ed il secondo?
         * @param parametro3 il terzo invece?
         */
        public Class(tipo_parametro parametro1, tipo_parametro parametro2,
                        tipo_parametro parametro3 ...){
                //Corpo del secondo costruttore
        }

        /**
         * Descrizione del metodo
         * @param parametro1 quale parametro si deve passare al metodo?
         * @return cosa restituisce il metodo al suo completamento?
         */
        public return_type method1(tipo_parametro parametro1) {
                private_method(parametro1);
                return return_value;
        }

        //per i metodi private non è richiesta documentazione
        private return_type private_method(tipo_parametro parametro1) {
                //Corpo del metodo private
                return return_value;
        }

        /**
         * Descrizione del metodo
         * @return cosa restituisce il metodo?
         * @throws Exception quando viene lanciata l'eccezione?
         */
        public return_type method2() throws Exception {
                //Notare come ogni condizione dell'IF sia racchiusa tra parentesi
                if(((1 < 2) && (2 < 3)) || (3 > 4)) {
                        throw new Exception();
                }
                //Notare anche come è identato il FOR
                for(int i = 0; i < 10; i++) {
                        //Corpo del FOR
                }
                return return_value;
        }
}
```

Ulteriori linee guida:

1. Una inizializzazione per linea:

```
//SBAGLIATO
int a = 10, b = 15;
//CORRETTO
int a = 10;
int b = 15;
```

2. Le stringhe e le espressioni troppo lunghe (indicativamente più di 65/70 caratteri) vanno divise su più linee:

```
//SBAGLIATO
String s = "Lorem ipsum dolor sit amet consectetur adipiscing elit. Mauris sodales
nunc efficitur, condimentum diam at, porttitor dolor"
//CORRETTO
String s = "Lorem ipsum dolor sit amet consectetur adipiscing elit. Mauris"
                + " nunc efficitur, condimentum diam at, porttitor dolor";
```

3. I metodi non devono essere troppo grandi (indicativamente più di 30 linee di codice) per evitare illeggibilità del codice. Se un metodo è troppo grande, va diviso in più metodi private.

1.4 Design Pattern

I design pattern descrivono un concetto che può essere definito come "una soluzione progettuale generale ad un problema ricorrente". Si tratta di una descrizione o di un modello logico da applicare per la risoluzione di un problema che può presentarsi in diverse situazioni durante le fasi di progettazione e sviluppo del sistema software.

I design pattern presi in considerazione per il sistema ScheduFIRE sono sostanzialmente due: MVC e Singleton.

1.4.1 MVC

Il design pattern MVC è un pattern architetturale che consente la suddivisione del sistema in tre blocchi principali. Grazie a questa suddivisione si è in grado di separare la logica di presentazione dei dati dalla logica di business.

Il Model fornisce i metodi di accesso ai dati persistenti, il View si occupa dell'interazione con l'utente e della presentazione dei dati prelevati dal Model, il Controller riceve i comandi dell'utente attraverso il View e modifica lo stato di quest'ultimo e del Model.

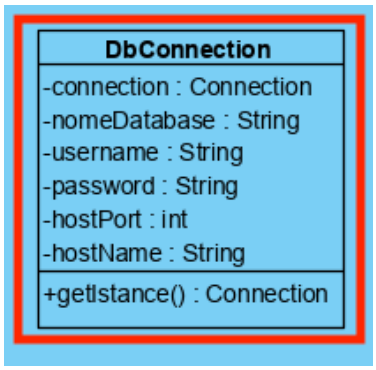
Nel sistema ScheduFIRE le classi sono state suddivise in package aventi come nome, la dicitura del blocco a cui si riferiscono.

1.4.2 Singleton

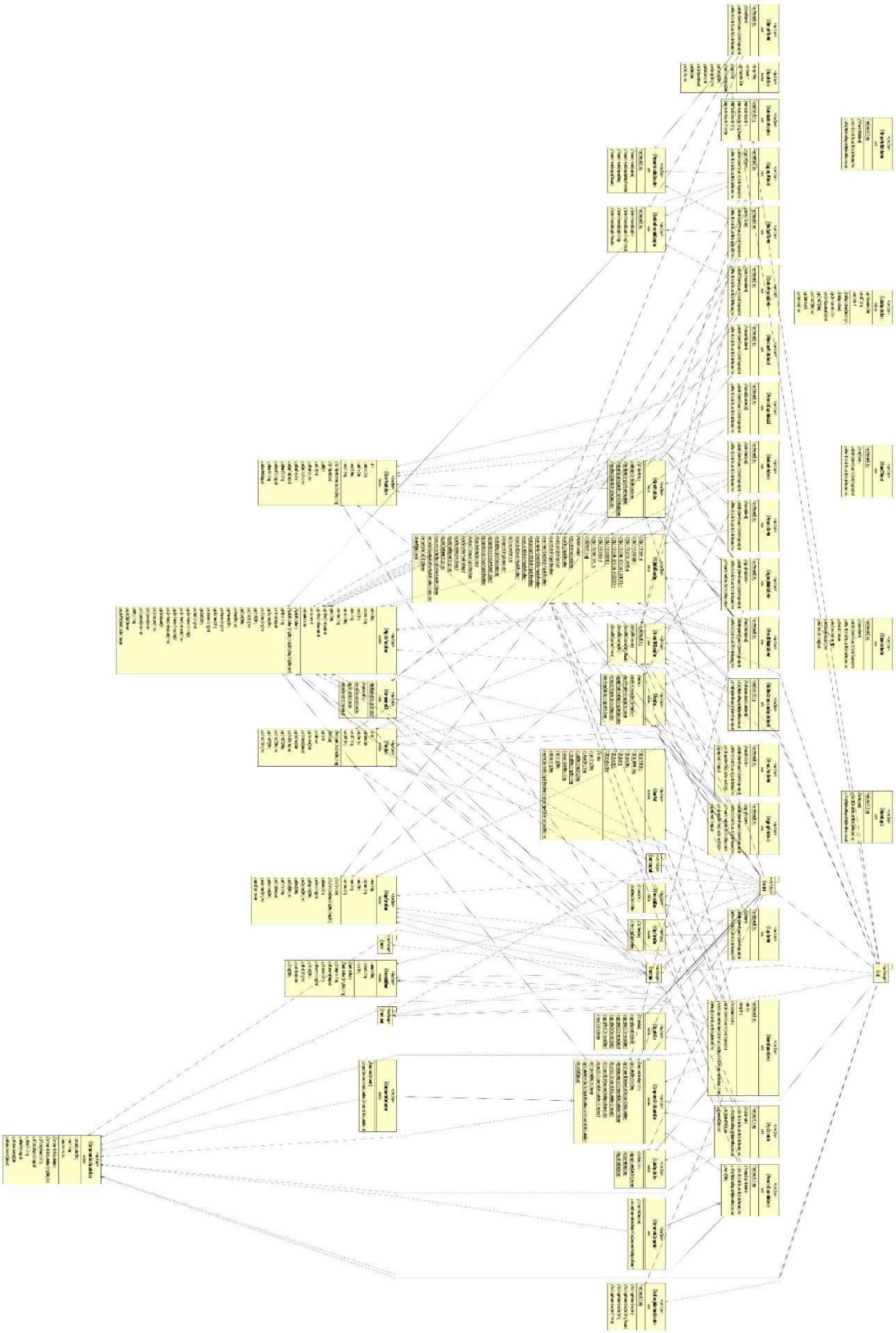
Il design pattern singleton è un pattern creazionale che ha lo scopo di garantire che di una determinata classe venga creata una ed una sola istanza e, di fornire un punto di accesso globale a tale istanza.

E' stata progettata una classe **DbConnection** avente come compito fondamentale la connessione al database, con un'operazione fine a ritornare l'istanza di connessione.

Per evitare la perdita di efficienza dovuta alla creazione di più istanze di questa classe si è deciso di renderla un Singleton.



Vista completa del sistema nella pagina che segue.



1.5 Definizioni, acronimi e abbreviazioni

HTML: HyperText Markup Language. Linguaggio di markup utilizzato per lo sviluppo di pagina web.

CSS: acronimo di Cascading Style Sheets è un linguaggio usato per definire la formattazione delle pagine HTML.

JS: JavaScript. E' un linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client per la creazione di effetti dinamici interattivi.

JQuery: è una libreria JavaScript per applicazioni web.

Apache Tomcat: è un web server open source. Fornisce una piattaforma software per l'esecuzione di applicazioni Web sviluppate in linguaggio Java.

Off-The-Shelf: Servizi esterni al sistema di cui viene fatto utilizzo.

Bootstrap: è una raccolta di strumenti liberi per la creazione di siti e applicazioni per il web.

Framework: supporta lo sviluppo di piattaforme web dinamiche, applicazioni e servizi web. Alleggerisce il lavoro associato allo sviluppo delle attività più comuni di un'applicazione web da parte dello sviluppatore.

Java: linguaggio di programmazione ad alto livello.

Javadoc: applicativo utilizzato per la generazione automatica della documentazione del codice sorgente scritto in linguaggio Java.

Design Pattern: descrizione o modello logico da applicare per la risoluzione di un problema che può presentarsi in diverse situazioni durante le fasi di progettazione e sviluppo software, ancor prima della definizione dell'algoritmo risolutivo della parte computazionale.

MVC: acronimo di Model-view-controller, è un pattern architetturale molto diffuso nello sviluppo di sistemi software.

JSP: acronimo di JavaScripting Prepocessor. E' una tecnologia di programmazione web in Java per lo sviluppo della logica di presentazione di applicazioni web.

Servlet: oggetti scritti in linguaggio Java che operano all'interno di un server web.

Exception handler: costrutto dei linguaggi di programmazione, progettato per gestire errori a runtime o altri problemi che avvengono durante l'esecuzione di un programma su di un computer.

1.6 Riferimenti

ScheduFIRE_RAD_v.1.7;

ScheduFIRE_SDD_V_0.5;

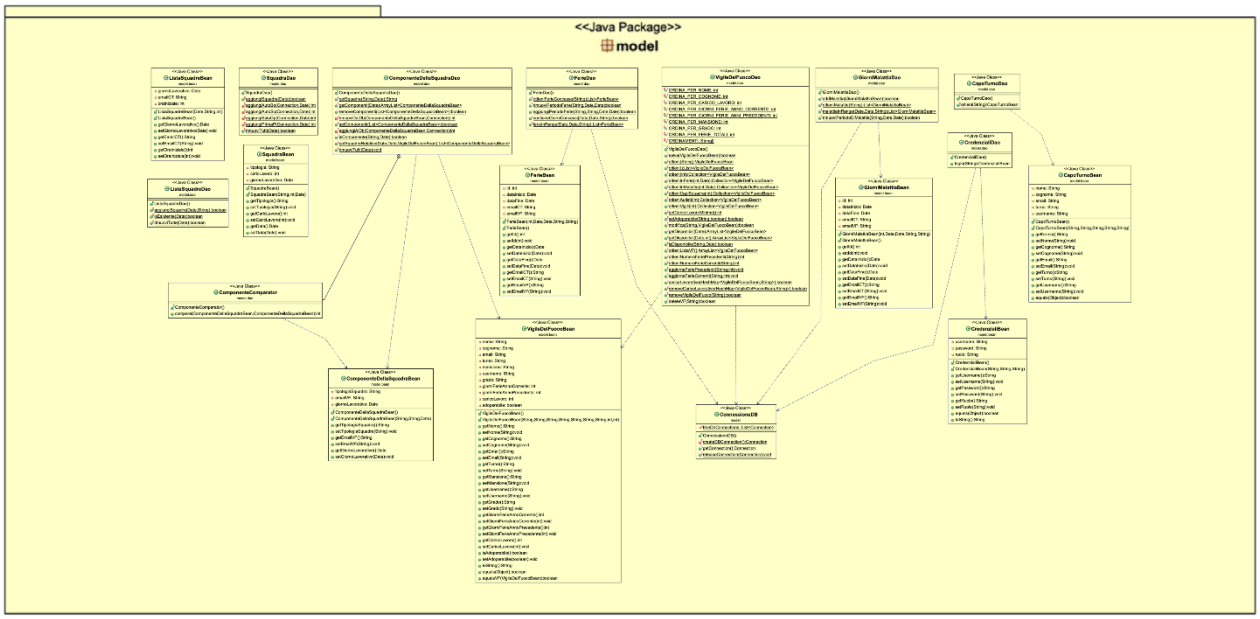
Slide del corso, presenti sulla piattaforma e-learning;

Libro: Object-Oriented Software Engineering (Using UML, Patterns and Java), Third Edition.
Autori: Bernd Bruegge & Allen H. Dutoit.

2. Packages

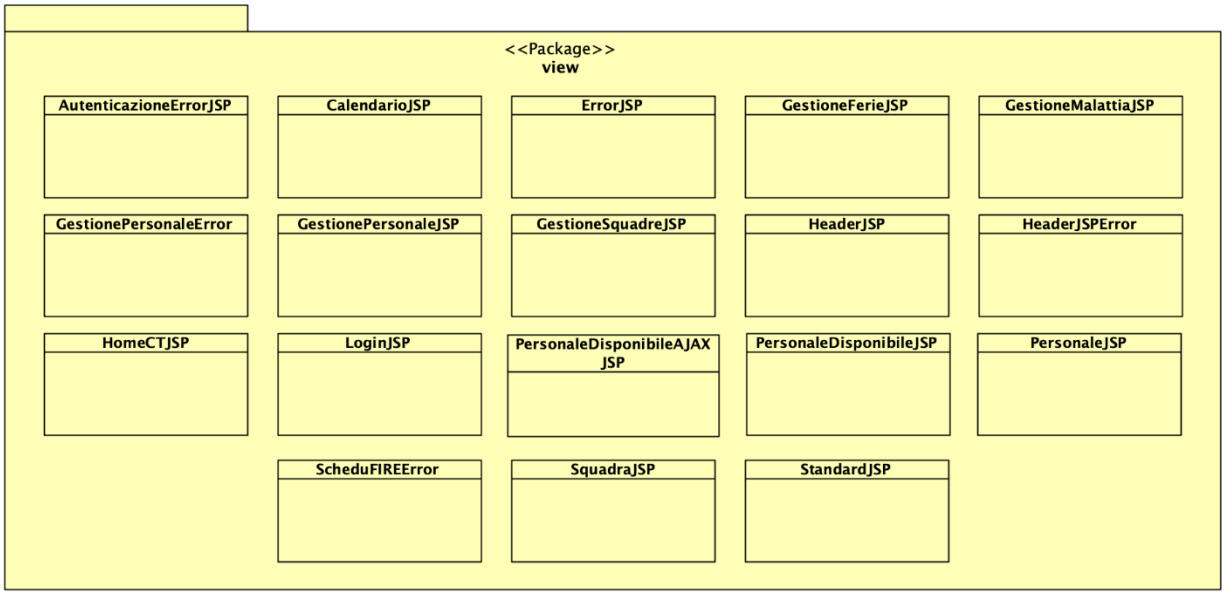
2.1 Model

Il package Model racchiude le classi utilizzate per la gestione dei dati presenti nel database; per compiere tale gestione, le classi avranno metodi di accesso e manipolazione dei dati. Queste classi modelleranno l'astrazione delle informazioni derivanti dal dominio applicativo. Le classi presenti in questo package sono: CapoTurno, VigileDelFuoco, Credenziali, Squadra, ComposizioneSquadra, FaParte, GiorniMalattia, Ferie.



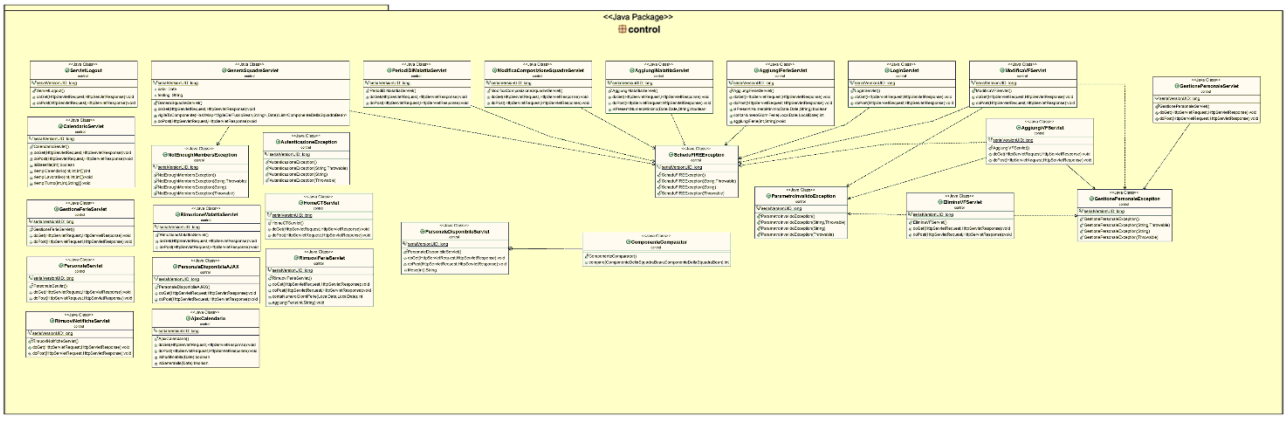
2.2 View

Il package View è composto da due ulteriori package: CapoTurno e VigileDelFuoco, distinguendo così le classi utilizzate per la vista dell'utente Capo Turno, da quelle dell'utente Vigile del Fuoco. Inoltre, è presente la classe LoginJSP che implementa la vista dell'accesso al sistema, comune a tutti gli utenti. Il Package CapoTurno conterrà la classe HomeSiteJSP che implementa la pagina iniziale del sistema, dalla quale si potrà accedere: alla pagina di gestione squadra, implementata dalla classe GestioneSquadraJSP; alla pagina di gestione delle ferie, implementata dalla classe GestioneFerieJSP; alla pagina di visualizzazione del calendario, implementata dalla classe VisualizzaCalendarioJSP; alla pagina di gestione del personale, implementata dalla classe GestionePersonaleJSP; alla pagina di gestione dei giorni di malattia, implementata dalla classe GestioneMalattiaJSP; alla pagine relativa al personale disponibile, implementata dalla classe PersonaleDisponibileJSP. Nel package VigileDelFuoco vi saranno le sole classi VisualizzaCalendarioJSP e VisualizzaTurnoJSP. Queste classi si interfacceranno con il package Controller per inoltrare le richieste e per ricevere le risposte dal server.



2.3 Controller

Il package controller si occupa, tramite le classi che lo implementano, di processare le richieste dell'utente. Esso è composto dalle seguenti servlet: LoginLogoutServlet, che si occupa di processare la richiesta di accesso e di uscita dal sistema; GeneraSquadreServlet, che si occupa di generare le squadre di soccorso dei VVF per i successivi giorni lavorativi; PersonaleDisponibileServlet, che si occupa di fornire la lista dei VVF disponibili; VisualizzaCalendarioServlet, che si occupa di ottenere le informazioni per costruire una vista dei giorni di lavoro; VisualizzaSquadreServlet, che si occupa di ottenere le informazioni relative alle squadre di un determinato giorno lavorativo; AggiungiVfServlet, che si occupa di inserire nel database un nuovo VF; ModificaVfServlet, che si occupa di modificare le informazioni di un VF presente nel database; EliminaVfServlet, che si occupa di rendere indisponibile un VF presente nel database; ConcediGiorniMalattiaServlet, che si occupa di concedere ai VVF dei giorni di degenza; AggiungiFerieServlet, che si occupa di assegnare ferie ai VVF; RimuoviFerieServlet, che si occupa di cancellare un periodo di ferie assegnato ad un VF. Per effettuare tali operazioni, le Servlet si servono della classe DbConnection, che realizza una connessione verso il database.



3. Class interfaces

Nome classe	LoginJSP

Descrizione	Questa classe rappresenta la vista della funzionalità del Login.
Pre-condizione	Context LoginJSP: validate(username, password); pre: username != null && password != null && email.equals(db.email) && password.equals(db.password)
Post-condizione	Accesso al sistema.
Invarianti	

Nome classe	HomeSiteJSP
Descrizione	Questa classe rappresenta la vista della pagina iniziale del CT e può reindirizzare a GestioneSquadraJSP, GestioneFerieJSP, VisualizzaCalendarioJSP, GestionPersonaleJSP, GestioneMalattiaJSP, PersonaleDisponibileJSP.
Pre-condizione	
Post-condizione	
Invarianti	

Nome classe	GestioneSquadraJSP
Descrizione	Questa classe rappresenta la vista della funzionalità di gestione delle squadre.
Pre-condizione	Context GestioneSquadraJSP: GestioneSquadraJSP()
Post-condizione	
Invarianti	

Nome classe	GestioneFerieJSP
Descrizione	Questa classe rappresenta la vista della funzionalità di gestione delle ferie.
Pre-condizione	Context GestioneFerieJSP: GestioneFerieJSP()
Post-condizione	
Invarianti	

Nome classe	VisualizzaCalendarioJSP
Descrizione	Questa classe rappresenta la vista della funzionalità di visualizzazione del calendario.
Pre-condizione	Context VisualizzaCalendarioJSP: VisualizzaCalendarioJSP()
Post-condizione	
Invarianti	

Nome classe	GestionePersonaleJSP
Descrizione	Questa classe rappresenta la vista della funzionalità di gestione del personale.

Pre-condizione	Context GestionePersonaleJSP: GestionePersonaleJSP()
Post-condizione	
Invarianti	

Nome classe	**GestoneMalattiaJSP**
Descrizione	Questa classe rappresenta la vista della funzionalità di gestione dei giorni di malattia.
Pre-condizione	Context GestioneMalattiaJSP: GestioneMalattiaJSP()
Post-condizione	
Invarianti	

Nome classe	**PersonaleDisponibileJSP**
Descrizione	Questa classe rappresenta la vista della funzionalità di visualizzazione del personale disponibile.
Pre-condizione	Context PersonaleDisponibileJSP: PersonaleDisponibileJSP()
Post-condizione	
Invarianti	

Nome classe	**VisualizzaTurnoJSP**
Descrizione	Questa classe rappresenta la vista della funzionalità di visualizzazione delle squadre di un turno.
Pre-condizione	Context VisualizzaTurnoJSP: VisualizzaTurnoJSP()
Post-condizione	
Invarianti	

Nome classe	**AggiungiVfJSP**
Descrizione	Questa classe rappresenta la vista della funzionalità di aggiunta di un VF al sistema.
Pre-condizione	Context AggiungiVfJSP: validate(nome, cognome, email, mansione, ferieCorrenti, feriePrecedenti) Pre: nome != null && cognome != null && email != null && mansione != null
Post-condizione	Il VF viene aggiunto al sistema.
Invarianti	

Nome classe	**ModificaVfJSP**
Descrizione	Questa classe rappresenta la vista della funzionalità di modifica di un VF nel sistema.

Pre-condizione	Context ModificaVfJSP: validate(nome, cognome, email, mansione, ferieCorrenti, feriePrecedenti) Pre: nome != null && cognome != null && email != null && mansione != null
Post-condizione	I dati del VF vengono modificati.
Invarianti	

Nome classe	EliminaVfJSP
Descrizione	Questa classe rappresenta la vista della funzionalità di rendere indisponibile un VF dal sistema.
Pre-condizione	Context EliminaVfJSP: EliminaVfJSP()
Post-condizione	Il VF viene reso indisponibile.
Invarianti	

4. Class Diagram

Viste le dimensioni del Class Diagram, questo sarà allegato nel documento ScheduFIRE_Class Diagram

5. Glossario

HTML: HyperText Markup Language. Linguaggio di markup utilizzato per lo sviluppo di pagina web.

CSS: acronimo di Cascading Style Sheets è un linguaggio usato per definire la formattazione delle pagine HTML.

JS: JavaScript. È un linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client per la creazione di effetti dinamici interattivi.

JQuery: è una libreria JavaScript per applicazioni web.

Apache Tomcat: è un web server open source. Fornisce una piattaforma software per l'esecuzione di applicazioni Web sviluppate in linguaggio Java.

Off-The-Shelf: Servizi esterni al sistema di cui viene fatto utilizzo.

Bootstrap: è una raccolta di strumenti liberi per la creazione di siti e applicazioni per il web.

Framework: supporta lo sviluppo di piattaforme web dinamiche, applicazioni e servizi web. Alleggerisce il lavoro associato allo sviluppo delle attività più comuni di un'applicazione web da parte dello sviluppatore.

Java: linguaggio di programmazione ad alto livello.

Javadoc: applicativo utilizzato per la generazione automatica della documentazione del codice sorgente scritto in linguaggio Java.

Design Pattern: descrizione o modello logico da applicare per la risoluzione di un problema che può presentarsi in diverse situazioni durante le fasi di progettazione e sviluppo software, ancor prima della definizione dell'algoritmo risolutivo della parte computazionale.

MVC: acronimo di Model-view-controller, è un pattern architetturale molto diffuso nello sviluppo di sistemi software.

JSP: acronimo di JavaScripting Prepocessor. È una tecnologia di programmazione web in Java per lo sviluppo della logica di presentazione di applicazioni web.

Servlet: oggetti scritti in linguaggio Java che operano all'interno di un server web.

Exception handler: costrutto dei linguaggi di programmazione, progettato per gestire errori a runtime o altri problemi che avvengono durante l'esecuzione di un programma su di un computer.