FsyManager

Titolo del progetto: FsyManager
Alunno/a: Dyuman Bulloni
Classe: Info 4
Anno scolastico: 2018/2019
Docente responsabile: Adriano Barchi

1 Introduzione

1.1 Informazioni sul progetto

Allievo: Dyuman Bulloni
Docente Responsabile:

- Adriano Barchi

Scuola: Scuola d'Arti Mestieri Trevano.
Sezione: Informatica.
Materia: Modulo Progetti Individuali
Data Inizio Progetto: 08.01.2019
Data Consegna Progetto: 10.04.2019

1.2 Abstract

Initial Situation
The project is born with the objective to replace completely an operational site (www.fsy-europe.com).
It will be used Spring Boot for programming, a Java Framework for Web MVC.
The work will be to create a site to manage registration and subscription of users for the fsy events, where the administrators have a significant role, monitoring the process and inform the user when he inserted wrong values.
The pages will be Multilanguage.
All data of users will be manipulate by the user itself and see it by admins, counsellors and assistants.
Actuation
I worked with Spring Boot for the main code, so using classes, controllers and file jsp (views).
For the web pages in jsp I used the HTML structure, JavaScript and Ajax logic, Bootstrap and free templates for graphics.
The pages are hosting in Heroku, a free web page for hosting much things (Spring Boot included), meanwhile for the Database I used ClearDb, an add-on of Heroku for MySql databases.
Results
The implementation of registration and subscription is completed, but the control of admins not, so the user will not know if the values he inserted are right or not.
The Multilanguage modality is on only for the login/registration page, and only for Italian and English.
I remove the pdf files from the project, so I ignored that phase of the subscription and the exportation of the file is available only for csv format.

1.3 Scopo

Il progetto è nato con lo scopo di creare un sistema di registrazione e iscrizione ad un evento, organizzato dalla Fsy, indirizzato a ragazzi dai 14 ai 18 anni che si riunisce annualmente in vari luoghi, il tutto chiaramente gestito da maggiorenni.
Questo lavoro è collegato a dei progetti proposti nello scorso semestre. Se i miei colleghi si son dedicati infatti alla gestione dei viaggi, all'assegnazione delle stanze e all'assegnazione degli utenti, il mio compito è quello di creare le pagine principali, che poi sfrutteranno i loro lavori (che rimangono quindi non veramente inerenti con le specifiche del lavoro), e fare in modo che queste pagine siano già predisposte ad eventuali cambiamenti voluti dagli amministratori.

Analisi

1.4 Analisi del dominio

La situazione iniziale non prevede alcuna base, se non quella del sito ufficiale di Fsy, il quale però non è gestito al meglio.
Sono già state create tramite altri progetti delle parti di questo nuovo sito, e questo progetto nasce proprio con lo scopo di unificare il tutto e avere, a linea teorica, un prodotto completo.
Purtroppo però gli scorsi progetti riguardanti altre parti non sono stati portati a termine, e quindi fino a quando rimarranno così tutto il lavoro rimane incompleto, e quindi più adatto ad un esercizio per testare le proprie abilità che a un prodotto vendibile.
Questo lavoro ha come scopo il sostituire o comunque sistemare il sito già presente e creato l'anno scorso per il gruppo Fsy. Questo infatti presenta un'ottima grafica ma una pessima implementazione di codice, rimanendo fin troppo grezzo e fine a sé stesso, impedendo quindi di effettuare dei miglioramenti.
L'applicativo web nasce per essere utilizzato da coloro che gestiscono gli eventi Fsy e coloro che vogliono non solo saperne di più riguardo a questa fantastica iniziativa, ma anche dar loro la possibilità di iscriversi e gestire le loro esigenze (mediche, alimentari, ecc.).

1.5 Analisi dei costi e benefici

Risorsa	**Costo**
1 Persona(50CHF) * 155 ore	7750 CHF
TOT:	**7750 CHF**

1.6 Analisi e specifica dei requisiti

ID: REQ-01	
Nome	Pagina di registrazione
Priorità	1
Versione	1.0
Note	
Sotto requisiti	
001	Si necessita una maschera di registrazione.
002	I dati obbligatori sono nome, cognome, password, categoria dell'utente e indirizzo email.
003	Gli utenti registrabili sono gli assistenti, consiglieri o partecipanti.
004	Dovrà essere gestito un controllo sulla validità dei dati inseriti.
005	Dovrà essere inviata un'email di conferma all'utente appena registrato.
006	Una volta che l'utente avrà confermato la validità della sua email, verrà inviata un'email agli amministratori.

ID: REQ-02	
Nome	Pagina per l'inserimento dei dati personali
Priorità	1
Versione	1.0
Note	
Sotto requisiti	
001	Una volta registrato con successo, l'utente avrà modo di inserire dati più specifici riguardo alla sua persona.
002	I dati che un partecipante dovrà inserire obbligatoriamente sono disponibili negli allegati.
005	I dati saranno poi inviati tramite email agli amministratori, i quali dovranno controllare che i dati siano pertinenti alla richiesta.
006	L'amministratore dovrà quindi confermare l'iscrizione dell'utente, il quale finalmente potrà passare all'ultima fase della registrazione all'evento.

ID: REQ-03	
Nome	Documenti PDF
Priorità	1
Versione	1.0
Note	
Sotto requisiti	
001	Ultima fase della registrazione all'evento. L'utente avrà la possibilità di scaricare dei file PDF.
002	Questi file PDF per i partecipanti saranno come quelli presenti negli allegati e dovranno essere compilati a mano in quanto necessitano di essere firmati.
003	Questi file PDF per gli assistenti saranno come quelli presenti negli allegati e dovranno essere compilati a mano in quanto necessitano di essere firmati.
004	Questi file PDF per i consiglieri saranno come quelli presenti negli allegati e dovranno essere compilati a mano in quanto necessitano di essere firmati.
005	I fogli andranno poi scannerizzati dall'utente e caricati all'interno del sito.
006	Gli amministratori verranno informati tramite email del caricamento, e a questo punto potranno controllare che i dati siano pertinenti e che sia tutto in ordine a livello legale.
007	Una volta confermata dagli amministratori, l'iscrizione verrà approvata e verrà inviata un'email di conferma al partecipante
008	L'utente che non avrà scaricato i file sarà ad uno stato inferiore del procedimento rispetto a chi invece lo ha fatto, seppur non ricaricandoli.

ID: REQ-04	
Nome	Pagine homes
Priorità	1
Versione	1.0
Note	
Sotto requisiti	
001	Queste dovranno avere una struttura che possa fare da base per spiegare cosa è l'evento, la sua location, ecc.
002	Deve essere implementata la funzione multilingua per tutte le pagine del progetto (incluse quelle di registrazione).

ID: REQ-05	
Nome	Gestione dati
Priorità	1
Versione	1.0
Note	
Sotto requisiti	
001	I dati verranno salvati all'interno di un database MySQL.
002	Verranno forniti dei dati iniziali tramite CSV. Deve quindi deve essere gestito il loro salvataggio.
003	Dovrà essere possibile stampare dei PDF contenenti i dati, filtrati tramite la richiesta dell'amministratore o assistenti.
004	I dati dovranno essere possibili da esportare in un file CSV.
005	Per l'amministratore deve essere implementata la possibilità di inserire nuovi parametri alla tabella.
006	I nuovi parametri andranno gestiti, venendo richiesti da quel momento in avanti al momento della registrazione.

1.7 Pianificazione

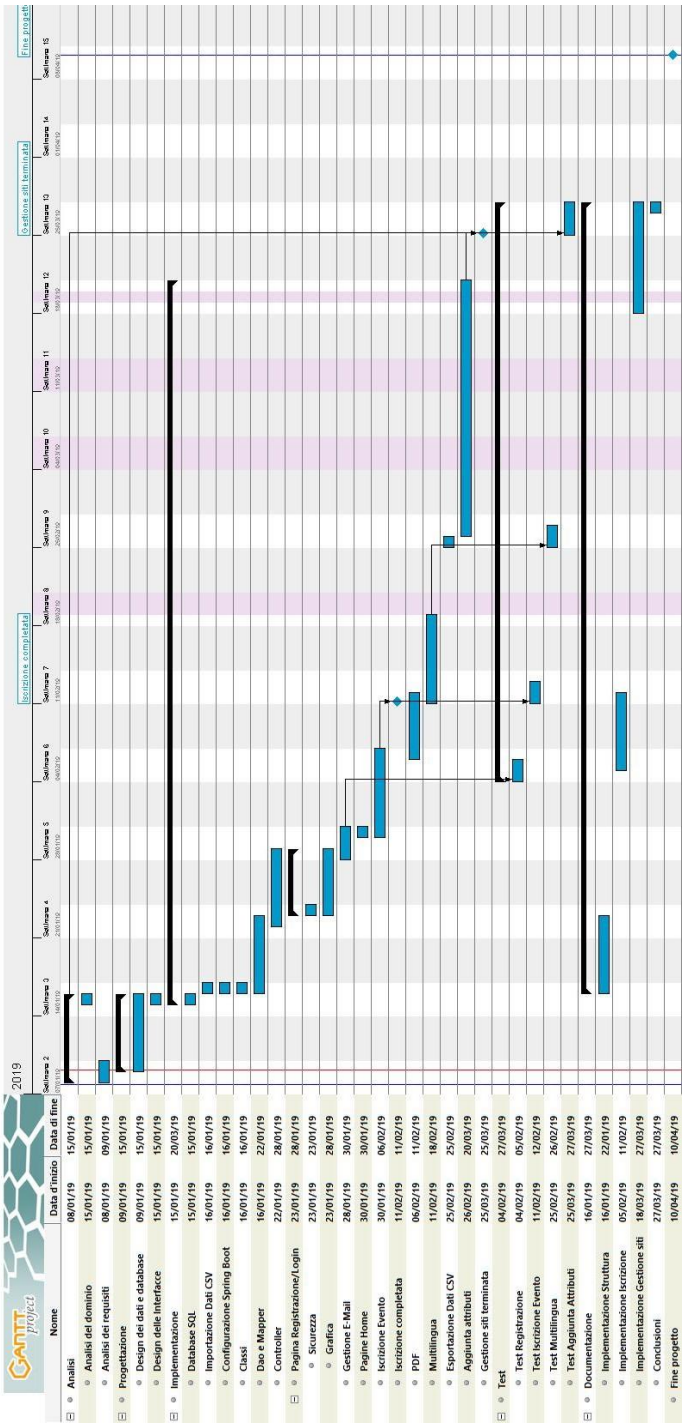

Figura 1 Gantt Preventivo

Il Gantt preventivo prevede un'analisi rapida ed efficace e una lavorazione partendo dalle basi dei dati, ovvero il database, per poi passare all'interfaccia di registrazione e login, con tutti i controlli del caso, e l'utilizzo delle email, fondamentali per tutti i passaggi. Solo una volta terminato ciò si passerà al lavoro vero e proprio.

1.8 Analisi dei mezzi

1.8.1.1 Software

Heroku usato come web hosting per l'applicativo.
ClearDB, add-on di Heroku utilizzato come hosting del database.
Java 8 Update 144, compilatore e linguaggio utilizzato.
Google Chrome 69.0.3497, browser utilizzato per navigare su internet.
Microsoft Word 2013, utilizzato per diari e documentazione.
IntelliJ IDEA 2018.2.3 Free Trial, utilizzato per lavorare con Spring.
GitHub Desktop 1.6.5, utilizzato per effettuare i backup del progetto su GitHub e per utilizzare Heroku (che appunto è collegato alla cartella GitHub).
7-Zip 18.05, programma utilizzato per estrarre i file scaricati in versione zip.
TomCat 8.5.34 Server necessario per lavorare con Spring, integrato con IntelliJ IDEA.

1.8.2 Hardware

1 PC S.O. Windows 10

2 Spring Boot

Per questo progetto ho deciso di utilizzare Spring Boot. La scelta è derivata più che altro dal voler mettere in pratica il linguaggio Spring (che altro non è che Java) imparato recentemente tramite l'esperienza lavorativa effettuata alla fine del terzo anno scolastico e messo in pratica nello scorso progetto.
Di seguito una breve guida sul funzionamento di questo linguaggio e di come è stato strutturato tutto il progetto. Quest'ultimo funziona tramite TomCat Server e, per favorire l'uso semplice delle librerie, Maven.

2.1.1.1 Configurazione

Per avere la struttura base, è sufficiente creare un applicativo Spring Boot (tramite questo sito è tutto automatizzato e facilitato: https://start.spring.io/).
La classe principale permette all'applicativo di essere lanciato al momento dell'avvio del TomCat Server.

```
6  @SpringBootApplication
7  public class FsyManagerApplication {
8
9      public static void main(String[] args) {
10         SpringApplication.run(FsyManagerApplication.class, args);
11     }
12
13 }
```

Figura 2 Classe Spring Boot d'avvio

Grazie a Maven, possiamo subito importare le librerie di cui abbiam bisogno (presenti di default utilizzando il link indicato prima).

2.1.1.2 Controller

Questi permettono di interpretare le richieste effettuate tramite browser. Per impostarli è sufficiente controllare di avere la seguente linea di codice, chiaramente con il package dove verranno poi contenuti i controller.

```
12    @ComponentScan("ch.samt.bulloni.FsyManager.Controller")
```

Figura 3 Configurazione Posizione Controller

Son sempre in ascolto e permettono ai metodi di essere mappati e richiamati tramite richiesta.
Per definire una classe come metodo, basta definire in questo modo la classe:

```
32    @Controller
33    @RequestMapping("/")
34    public class FsyController {
```

Figura 4 Configurazione Classe Controller

È possibile avere più controller. Per farlo, è sufficiente definire la mappatura richiesta in modo diverso. Ricordare bene che per richiamare le mappature dei metodi interni è necessaria come prima cosa la mappatura del controller.
Per mappare un metodo, è necessario impostarli nel seguente modo:

```
199    @RequestMapping(value = "/helloWorld", method = RequestMethod.GET)
```

Figura 5 Mappatura di un metodo

Il primo parametro definisce la mappatura, mentre il secondo permette di variare il metodo utilizzato per effettuare la richiesta.
Le richieste più utilizzate son quella GET (di default), che indica semplicemente la richiesta tramite browser. È poi usata molto anche la richiesta POST, quando si desidera inviare dei dati in modo più nascosto, o comunque una grande mole di dati.

```
@RequestMapping(value = "/helloWorld", method = RequestMethod.GET)
public ModelAndView helloWorld(ModelAndView model){
    String message = "hello";
    model.addObject( attributeName: "message", message);
    model.setViewName("home");
    return model;
}
```

Figura 6 Esempio Configurazione richiesta GET

L'oggetto ritornato corrisponde all'insieme di model (e quindi tutti gli oggetti aggiunti) e il nome del file view.
è comunque possibile definire dei parametri nelle richieste GET, separati da '/' e impostati nel seguente modo:

```
@RequestMapping(value = "/helloWorld/{message}", method = RequestMethod.GET)
public ModelAndView helloWorld(ModelAndView model, @PathVariable String message){
    model.addObject( attributeName: "message", message);
    model.setViewName("home");
    return model;
}
```

Figura 7 Esempi richieste GET con parametri

È fondamentale che il nome della variabile e quello della mappatura siano uguali. Il tipo della variabile può essere soltanto Stringa. Eventuali cast e errori andranno gestiti quindi di conseguenza.

Per le richieste post invece, è necessario usare una classe aggiuntiva, HttpServletRequest. Nella richiesta post, tutti i valori mappati tramite l'attributo "name" son inviati tramite la richiesta e contenuti nell'oggetto. A questo punto, per estrapolarli, è semplicemente necessario sapere in che modo son stati nominati.
Vengono tutti salvati come stringa. Nel caso esistano più tag con lo stesso nome, utilizzare il metodo a riga 42 per ottenere tutti i valori.

```
@RequestMapping(value = "/helloWorld", method = RequestMethod.POST)
public ModelAndView helloWorld(ModelAndView model, HttpServletRequest request){
    model.addObject( attributeName: "names", request.getParameterValues( s: "names"));
    model.addObject( attributeName: "surname", request.getParameter( s: "surname"));
    model.setViewName("home");
    return model;
}
```

Figura 8 Esempio Metodo richiesta POST

È anche possibile definire i metodi di richiesta come HEAD, DELETE e altri.
Chiaramente tutte queste richieste possono essere effettuate tramite Ajax, e quindi una struttura sincrona. Ricordarsi tuttavia che la pagina non mostrerebbe immediatamente i risultati, in quanto è comunque necessario che si ricarichi il sito.

2.1.1.3 Classi

Le classi definite all'interno della struttura fanno da tramite con il database implementato (in questo caso, SQL). È possibile definirle in maniera più stretta, legandole alle tabelle. Tuttavia, per permetterne un uso più ampio e maggiore malleabilità, è stato deciso di non implementare quel tipo di logica.

2.1.1.4 Dao

Questi sono il tramite principale con il database. Definendo correttamente quest'ultimo, le azioni possibili sul nostro database saranno perfettamente gestite e controllate, evitando SQL Injections o richieste non consone.
Va prima di tutto definito il Bean (oggetto che tramite il sistema @AutoWired si imposta autonomamente) che permetterà di interfacciarsi al database, ovvero BasicDataSource. Quest'ultimo userà poi l'oggetto JdbcTemplate per tutte le operazioni.

```
25      @Bean
26      public BasicDataSource herokuDataSource(){
27          BasicDataSource basicDataSource = new BasicDataSource();
28          basicDataSource.setDriverClassName("com.mysql.jdbc.Driver");
29          String username = "b35854f78a276c";
30          String password = [redacted];
31          String dbUrl = "jdbc:mysql://eu-cdbr-west-02.cleardb.net/heroku_d933dd3f20a51de";
32
33          basicDataSource.setUrl(dbUrl);
34          basicDataSource.setUsername(username);
35          basicDataSource.setPassword(password);
36
37          return basicDataSource;
38      }
39
40      @Bean
41      public JdbcTemplate toJdbc(){
42          JdbcTemplate jdbc = new JdbcTemplate(herokuDataSource());
43          return jdbc;
44      }
```

Figura 9 Configurazione Collegamnento con DB

All'interno dei propri Dao è ora sufficiente inserire le seguenti linee:

```
30      @Autowired
31      private JdbcTemplate jdbcTemplate;
32
33      public UserDao(JdbcTemplate jdbcTemplate) { this.jdbcTemplate = jdbcTemplate; }
```

Figura 10 Istanziazione del collegamento DB

Tutti le operazioni vengon quindi gestite definendo le stringhe dell'operazione:
È quindi possibile effettuare tutte le operazioni del caso, le manipolazioni di dati si basano tutte sulla seguente logica (stringa sql impostata correttamente e poi la chiamata al metodo execute).

```
37      public boolean createUser(User u){
38          if(u != null) {
39              if(getUserByIds(u.getName(), u.getSurname(), u.getBirthDate()) != null){
40                  return false;
41              }
42              String id = u.getName()+"."+u.getSurname()+"."+dateFormat(u.getBirthDate());
43              String sql = "insert into user (id, name, surname, birthDate, email, password, link) " +
44                      "values (?,?,?,?,?,?,?)";
45              return this.doChangeFilter(sql, id, u.getName(), u.getSurname(),
46                      dateFormat(u.getBirthDate()), u.getEmail(), u.getPassword(), u.getLink());
47          }
48          return true;
49      }
```

Figura 11 Esempio di metodo Dao

Per la lettura dei dati invece la questione si complica un attimo, in quanto i dati estrapolati si ha bisogno di ottenerli in maniera ordinata. Rientrano dunque in gioco le classi create in precedenza. Assegnando i valori del risultato filtrandoli e inserendoli all'interno della classe, è possibile ottenere degli oggetti contenenti i dati.
Usando le preparedStatement di java, è necessario questo metodo:

```
314     private boolean doChangeFilter(String sql, String... params){
315         return jdbcTemplate.execute(sql,new PreparedStatementCallback<Boolean>(){
316             @Override
317             public Boolean doInPreparedStatement(PreparedStatement ps) throws SQLException, DataAccessException {
318                 for(int i = 0; i < params.length; i++){
319                     ps.setString( parameterIndex: i+1, params[i]);
320                 }
321                 return !ps.execute();
322             }
323         });
324     }
```

Figura 12 PreparedStatement azione di modifica

Per richieste select specifiche, è sufficiente impostare la stringa sql e passarla come valore assieme al numero di parametri (corrispondenti ai '?') al metodo che esegue la richiesta.

```java
public List<User> getUsers() {
    String sql = "select *from user";
    List<User> us = this.selectUser(sql);
    if(us != null){
        if(us.size() > 0){
            return us;
        }
    }
    return null;
}
private List<User> selectUser(String sql){
    List <User> us = jdbcTemplate.query(sql, new UserMapper());
    return us;
}
```

Figura 13 Lettura dei dati da DB

```java
public class UserMapper implements RowMapper<User> {
    public User mapRow(ResultSet rs, int rowNum) throws SQLException {
        User us = new User();
        us.setName(rs.getString( columnLabel: "name"));
        us.setSurname(rs.getString( columnLabel: "surname"));
        us.setBirthDate(LocalDate.parse(rs.getString( columnLabel: "birthDate")));
        us.setEmail(rs.getString( columnLabel: "email"));
        us.setGender(rs.getString( columnLabel: "gender").equals("1"));
        us.setRole(rs.getString( columnLabel: "role"));
        us.setLink(rs.getString( columnLabel: "link"));
        int tot = rs.getMetaData().getColumnCount();
        List<String[]> values = new ArrayList<>();
        for(int i = 1; i <= tot; i++){
            if(rs.getMetaData().getColumnName(i) != null){
                values.add(new String[]{rs.getMetaData().getColumnName(i), rs.getString(i)});
            }
        }
        us.setValues(values);
        return us;
    }
}
```

Figura 14 Mapper dei dati DB e una classe

Chiaramente i parametri devono essere il nome della colonna in stringa. Nel caso il dato non sia una stringa ma per esempio un intero, è presente per la classe ResultSet anche il metodo getInt e altri campi. Anche se è consigliato comunque estrarre prima di tutto la stringa per poi al massimo fare un parsing al tipo di dato desiderato.

2.1.1.5 Pagine JSP

Queste son il risultato che verrà mostrato all'utente. Per essere utilizzate, bisogna controllare che l'applicazione sappia dove sono i file e che quindi siano presenti le seguenti linee di codice all'interno di un file di configurazione:

```
23      @Override
24      public void configureDefaultServletHandling(
25              DefaultServletHandlerConfigurer configurer) {
26          configurer.enable();
27      }
28      @Bean
29      public ViewResolver getViewResolver() {
30          InternalResourceViewResolver resolver = new InternalResourceViewResolver();
31          resolver.setPrefix("/WEB-INF/jsp/");
32          resolver.setSuffix(".jsp");
33          return resolver;
34      }
```

Figura 15 Configurazione della locazione e il tipo delle View

Tramite il suffisso, ora quando le view verranno date (ritornate tramite stringa) sarà sufficiente inserire il nome del file senza il suffisso ".jsp".
La pagina utilizza quindi una classica struttura HTML, con la possibilità di usare anche CSS, JavaScript, JQuery e simili. Ha però qualche funzione extra.
La più utilizzata altro non è che il sistema per usare i dati mappati o definiti tramite tools aggiuntivi.
Funziona come un classico template. Definendo ${nome_var}, è possibile ottenere il dato e perfino usare della logica java all'interno delle parentesi graffe, come un assegnazione, un'operazione ternaria o il richiamo di un metodo (nel caso non sia implementabile, darà chiaramente errore).

```
22      ${message}
```

Figura 16 Esempio di variabile usata dai file JSP

Come accennato poco fa, son infatti disponibili dei tools che aggiungono dei tag (lista completa: https://www.tutorialspoint.com/jsp/jsp_standard_tag_library.htm). Essi permettono di implementare logica all'interno della pagina e non solo in una piccola parte di essa.
Per creare le variabili direttamente all'interno della pagina infatti, si utilizza il tag c nel seguente modo:

```
23      <c:set var="i" value="${0}"></c:set>
24      <c:set var="i" value="${i+1}"></c:set>
```

Figura 17 Esempio di creazione e modifica variabile JSP

Nel corso di questo progetto ne verrà utilizzato solamente uno, ovvero quello che permette di usare foreach e choose (variante migliore del classico if), nonché variabili interne alla pagina.
Per aggiungere questa libreria è seguente inserire questa linea di codice nella pagina:

```
1       <%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>
```

Figura 18 Libreria per utilizzare tag aggiuntivi JSP

Il foreach richiede un oggetto contenente più oggetti come "item" e li estrapola uno a uno per inserirli all'interno del nome definito in "var" (sebbene questa vada definita in stringa, per utilizzarne il valore sarà necessario usare ${nome_var}).

```
23      <c:forEach items="${names}" var="name">
24          <p>${name}</p>
25      </c:forEach>
```

Figura 19 Esempio ForEach

Un altro fondamentale utilizzo di questo tool prevede il tag choose.
Questo funziona come un normale if, dove per ogni if viene utilizzato un when. L'else corrisponde invece ad un otherwise. Il when va utilizzato usando test="${variabile booleana oppure una logica che ne riporti una}".

```
28  <c:choose>
29      <c:when test="${checked}">
30          <c:set var="admin" value="${''}"></c:set>
31      </c:when>
32      <c:when test="${write}">
33          <c:set var="wPermission" value="${''}"></c:set>
34      </c:when>
35  </c:choose>
```

Figura 20 Esempio Choose

2.1.1.6 Librerie / file esterni

Questi andranno all'interno della cartella WEB-INF (o webapps, a differenza della struttura). È consigliato strutturare il tutto all'interno di una cartella, solitamente denominata "resources", e inserire queste linee di codice:

```
15  @Override
16  public void addResourceHandlers(final ResourceHandlerRegistry registry) {
17      registry
18              .addResourceHandler( ...pathPatterns: "/resources/**")
19              .addResourceLocations("/WEB-INF/resources/");
20  }
```

Figura 21 Configurazione utilizzo risorse esterne

In questo modo, i file verranno visualizzati e usati correttamente.

3 Progettazione

3.1 Design dell'architettura del sistema

3.1.1 Struttura delle cartelle

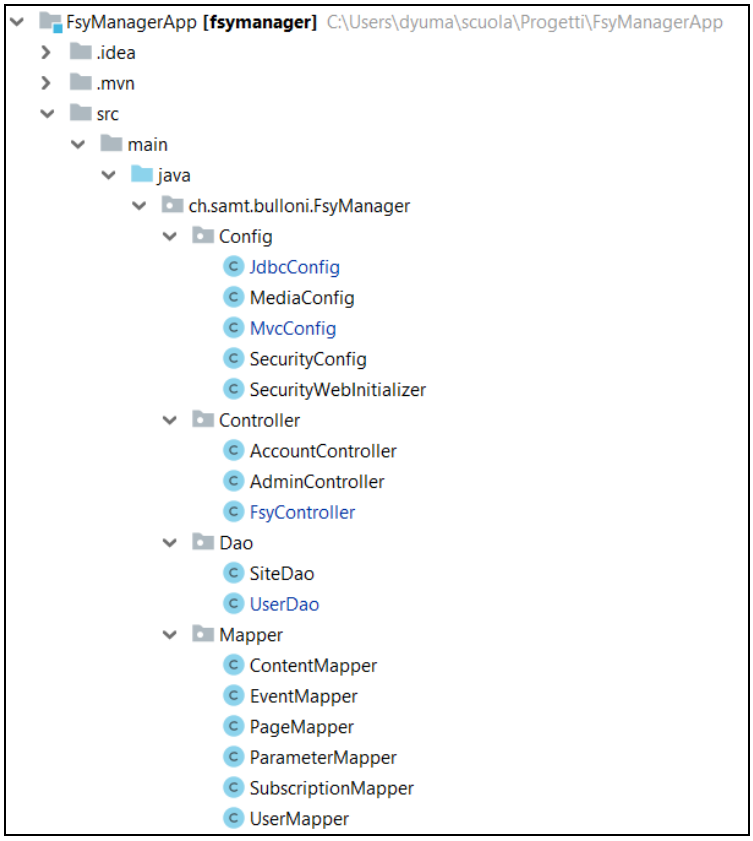

Figura 22 Struttura cartelle 1

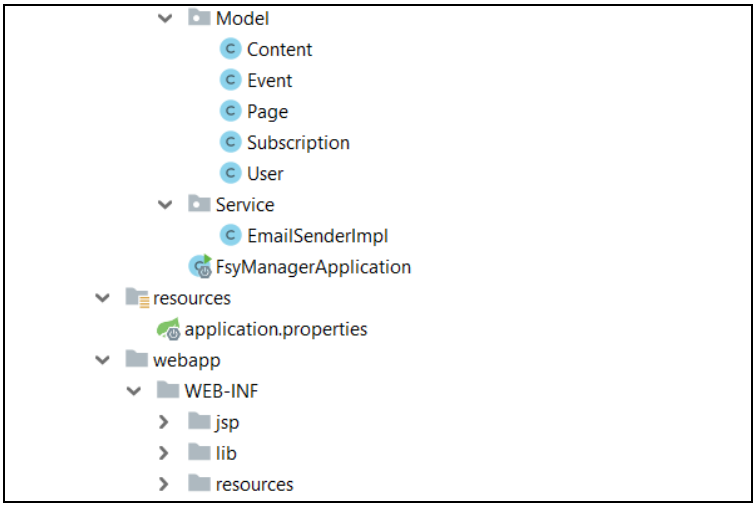

Figura 23 Struttura cartelle 2

AdminController si occupa di tutte le operazioni riguardanti i privilegi da admin.

FsyController si occupa di tutta la gestione del sito anche senza dover essere loggati (e esegue login e registrazione), così come la funzione del filtro.
AccountController si occupa di tutte le operazioni di gestione account e iscrizioni dedicate.
I Dao non hanno particolari funzioni, servono da tramite per operare con il database e contengono solamente i metodi che son stati ritenuti necessari per lavorare.
I Mapper hanno semplicemente la funzione di supportare i Dao, strutturando i dati ottenuti in maniera consona alle classi contenute nel Model.
Quest'ultime, oltre che servire da contenitori per i dati delle rispettive tabelle, hanno attributi aggiuntivi, che hanno l'obiettivo di facilitare il passaggio di dati potendo mostrare più dettagli tramite le pagine web.
I file Config riguardano configurazioni necessarie al progetto, verranno ripresi nell'implementazione.
L'unico file nella cartella servlet riguarda l'utilizzo delle email.
Le risorse statiche non sono state utilizzate, mentre application.properties è anch'esso un file di configurazione, ma è stato soppiantato dalla configurazione tramite Bean presente nei file Java di configurazione.

3.1.2 Struttura delle risorse

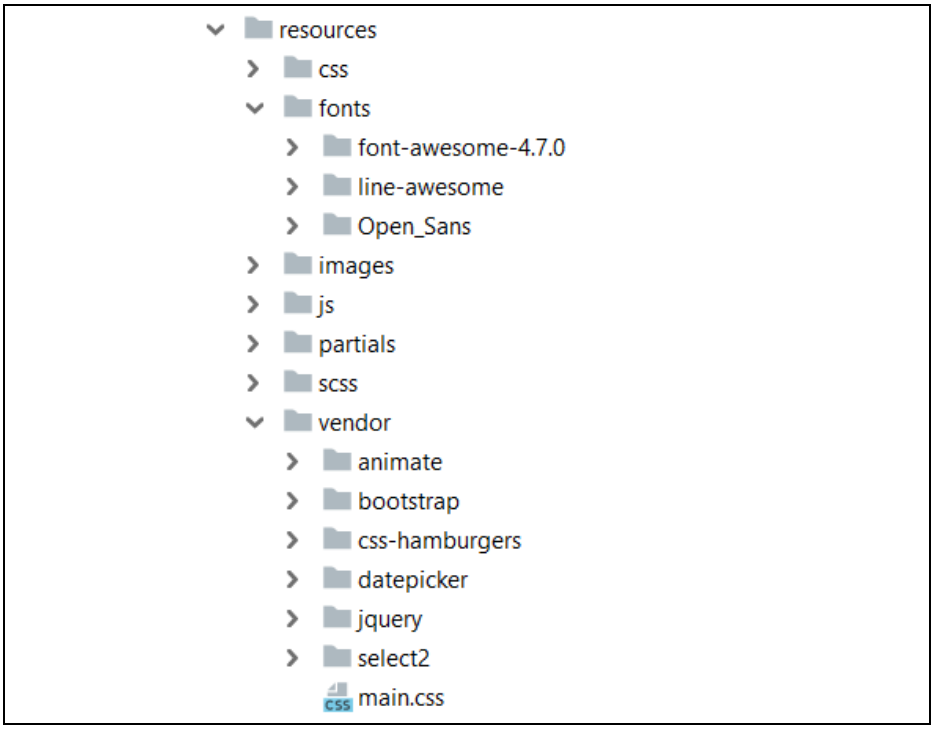

Figura 24 Struttura delle risorse

Nella cartella webapp sono presenti tutti i file che dovranno essere disponibili nel web.
Le pagine JSP, così come le librerie che necessita di utilizzare e eventuali file CSS e JS. In questo progetto Definire le risorse (immagini, CSS e JS) all'interno di una cartella come resources permette poi di escludere quest'ultima dal controllo di Spring Security.

3.1.3 Struttura delle librerie

Figura 25 Struttura librerie

Queste sono le librerie che, per ragioni ancora non chiare, non son potute essere importate tramite Maven (di cui il file è pom.xml). Perlopiù quelle che han dato problemi son infatti riguardanti Spring Security, il connettore con i database mysql e l'utilizzo delle email. Le restanti son più per funzioni minori, come la creazione di una stringa casuale per i link dei database.
Son poi state importate le seguenti librerie tramite Maven:
Tutte queste librerie son necessarie a Spring Boot per partire e gestire tutte le funzioni utilizzate dal progetto.

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web-services</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>

        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.tomcat.embed</groupId>
        <artifactId>tomcat-embed-jasper</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-web</artifactId>
    </dependency>
```

Figura 26 Struttura librerie Maven 1

```
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
    </dependency>


    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-core</artifactId>
        </dependency>


        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
        </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-mail</artifactId>
    </dependency>

    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
    </dependency>
```

Figura 27 Struttura librerie Maven 2

```
88      <dependency>
89          <groupId>org.springframework</groupId>
90          <artifactId>spring-core</artifactId>
91      </dependency>
92      <dependency>
93          <groupId>org.springframework</groupId>
94          <artifactId>spring-context</artifactId>
95      </dependency>
96      <dependency>
97          <groupId>org.springframework</groupId>
98          <artifactId>spring-jdbc</artifactId>
99      </dependency>
100     <dependency>
101         <groupId>org.springframework</groupId>
102         <artifactId>spring-beans</artifactId>
103     </dependency>
104     <dependency>
105         <groupId>org.springframework</groupId>
106         <artifactId>spring-aop</artifactId>
107     </dependency>
108     <dependency>
109         <groupId>org.springframework</groupId>
110         <artifactId>spring-tx</artifactId>
111     </dependency>
112     <dependency>
113         <groupId>org.springframework</groupId>
114         <artifactId>spring-expression</artifactId>
115     </dependency>
116     <dependency>
117         <groupId>org.springframework</groupId>
118         <artifactId>spring-web</artifactId>
119     </dependency>
```

Figura 28 Struttura librerie Maven 3

```
121     <dependency>
122         <groupId>javax.servlet</groupId>
123         <artifactId>jstl</artifactId>
124         <version>1.2</version>
125     </dependency>
126     <dependency>
127         <groupId>org.jetbrains</groupId>
128         <artifactId>annotations</artifactId>
129         <version>RELEASE</version>
130         <scope>compile</scope>
131     </dependency>
132
133
134
135     <dependency>
136         <groupId>commons-dbcp</groupId>
137         <artifactId>commons-dbcp</artifactId>
138     </dependency>
139     <dependency>
140         <groupId>commons-dbcp</groupId>
141         <artifactId>commons-dbcp</artifactId>
142         <version>1.2.2</version>
143     </dependency>
```

Figura 29 Struttura librerie Maven 4

Va inoltre inserito il seguente codice (fuori dalle dipendenze) per fare in modo che Maven si avvii da solo all'interno dell'hosting esterno.

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-dependency-plugin</artifactId>
            <executions>
                <execution>
                    <phase>package</phase>
                    <goals><goal>copy</goal></goals>
                    <configuration>
                        <artifactItems>
                            <artifactItem>
                                <groupId>com.github.jsimone</groupId>
                                <artifactId>webapp-runner</artifactId>
                                <version>9.0.13.0</version>
                                <destFileName>webapp-runner.jar</destFileName>
                            </artifactItem>
                        </artifactItems>
                    </configuration>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

Figura 30 Struttura Build Maven per hosting esterno

3.2 Design dei dati e database

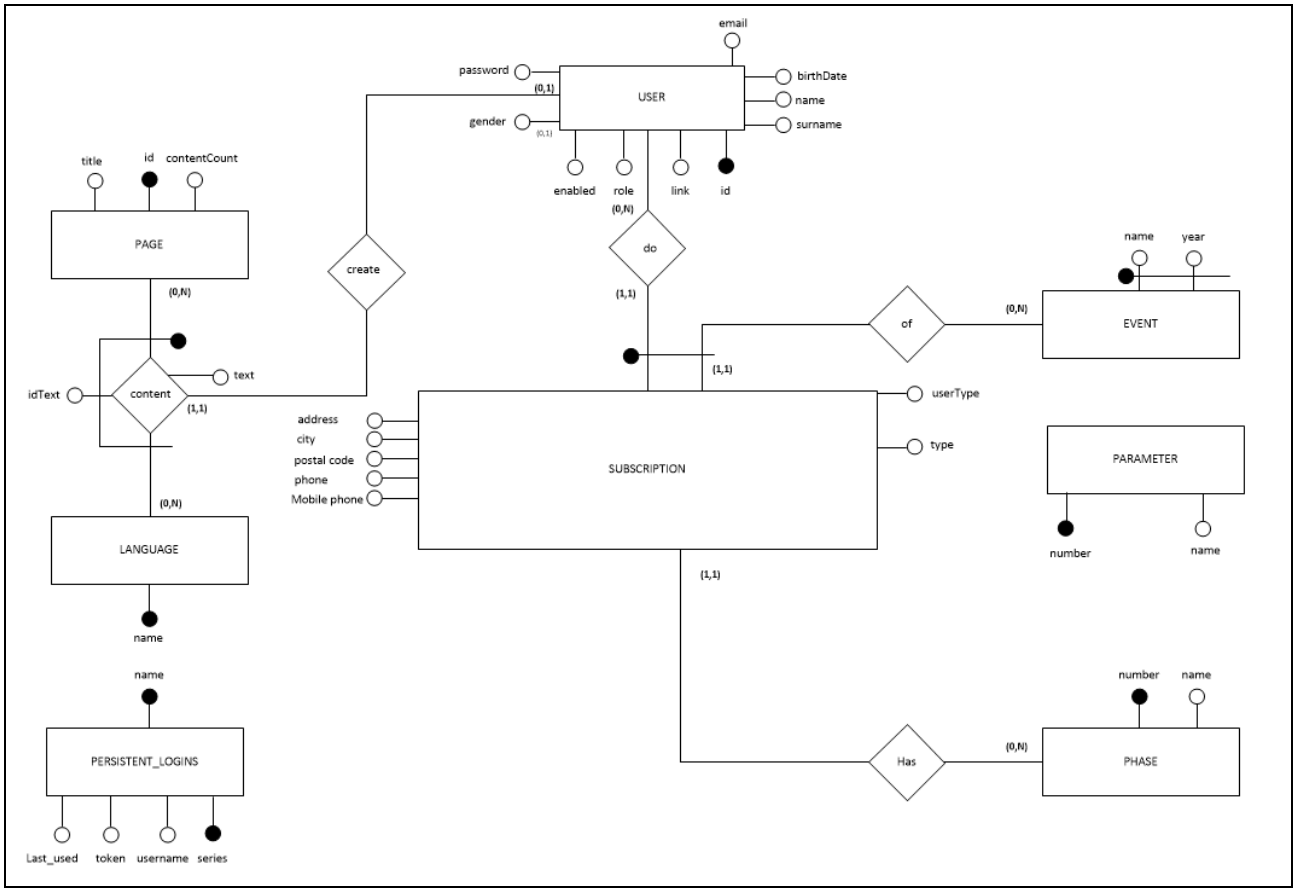

Figura 31 Schema E-R

Il database finale è privo di moltissimi campi, che son stati rimossi per praticità di lavoro.
User consiste nell'utente e i suoi dati.
Subscription sono le varie iscrizioni effettuate.
Event son tutti gli eventi organizzati da fsy.
Parameter son i parametri presenti in subscription.
Phase son tutte le fasi delle iscrizioni.
Persistent_logins viene utilizzato per salvare i cookie per la funzione "ricordami".
Language son le varie lingue presenti.
Content è il contenuto della pagina nelle varie lingue.
Page rappresenta le pagine presenti nel progetto.

4 Implementazione

4.1 Email

Una parte fondamentale è senza dubbio l'utilizzo delle email.
Come prima cosa va definito il bean che permetterà al programma di accedere all'email creata appositamente per questo progetto per poi inviare le email necessarie.

```
10  @Configuration
11  public class MediaConfig {
12
13      @Bean
14      public JavaMailSender getJavaMailSender() {
15          JavaMailSenderImpl mailSender = new JavaMailSenderImpl();
16          mailSender.setHost("smtp.gmail.com");
17          mailSender.setPort(587);
18          mailSender.setUsername("samt.fsymanager@gmail.com");
19          mailSender.setPassword[redacted]');
20          Properties props = mailSender.getJavaMailProperties();
21          props.put("mail.transport.protocol", "smtp");
22          props.put("mail.smtp.auth", "true");
23          props.put("mail.smtp.starttls.enable", "true");
24          props.put("mail.smtp.starttls.required", "false");
25          props.put("mail.debug", "true");
26          props.put("mail.smtp.socketFactory.class", "javax.net.ssl.SSLSocketFactory");
27
28          return mailSender;
29      }
30
31
32  }
```

Figura 32 Mail 1

Definendo questa classe infatti, utilizzando le funzioni basilari della classe MimeMessage, è possibile inviare le email. A questo punto sarà sufficiente istanziare un oggetto EmailSenderImpl e richiamare la funzione sendingMail con i relativi parametri. L'email sarà inviata senza alcun particolare problema, a patto che la porta tcmp 465 sia disponibile nella rete (cosa che non ho chiesto per il progetto visto quanto in là è stata completata questa funzione).

```java
@Service
public class EmailSenderImpl {

    @Autowired
    private JavaMailSender javaMailSender;

    public void sendingMail(String to, String subject, String body){
        MimeMessage message=javaMailSender.createMimeMessage();
        MimeMessageHelper helper;
        try {
            helper=new MimeMessageHelper(message, multipart: true);
            helper.setTo(to);
            helper.setSubject(subject);
            helper.setText(body, html: true);
            javaMailSender.send(message);

        } catch (javax.mail.MessagingException e) {
            e.printStackTrace();
        }
    }
}
```

Figura 33 Classe per inviare email

4.2 Spring Security

Spring Security consiste in tutta la sicurezza dell'applicativo.
Tramite database viene controllato id e ruolo, confrontandone poi la password cifrata tramite una classe Java.

```java
@Configuration
@EnableWebMvcSecurity
@EnableGlobalMethodSecurity(securedEnabled = true)
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    BasicDataSource herokuDataSource;

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth.jdbcAuthentication().dataSource(herokuDataSource)
                .authoritiesByUsernameQuery("select id, role FROM user where id=?")
                .usersByUsernameQuery("select id,password,enabled FROM user where id=?")
                .passwordEncoder(passwordEncoder());
    }
    @Bean
    public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }

    @Override
    public void configure(WebSecurity web) throws Exception {
        web.ignoring().antMatchers( ...antPatterns: "/resources/**");
    }
```

Figura 34 Spring Security 1

Il filtro consiste nell'impedire alcune richieste URL obbligando il login o impedendone proprio l'accesso.

Per fare ciò è necessario il metodo configure, tramite il quale vengono definiti i vari parametri di gestione login tramite Spring Security e i vari url che richiedono elementi specifici, come essere autenticati oppure avere un ruolo definito.
Il secondo metodo serve per implementare la funzione “Ricordami”, che fa riferimento alla tabella persistent_logins nel quale vengono salvati come token i cookie di accesso, permettendo così di fornirlo quando necessario.

```
45      @Override
46      protected void configure(HttpSecurity http) throws Exception {
47          http
48                  .authorizeRequests() ExpressionInterceptUrlRegistry
49                  .antMatchers( ...antPatterns: "/admin/**").access( attribute: "hasRole('ROLE_ADMIN')")
50                  .antMatchers( ...antPatterns: "/account/**").access( attribute: "isAuthenticated()") Ex
51                  .antMatchers( ...antPatterns: "/users/**").access( attribute: "hasRole('ROLE_ADMIN')||'
52                  "hasRole('ROLE_ASS')||hasRole('ROLE_COU')") ExpressionUrlAuthorizationConfigurer<HttpSecu
53                  .and() HttpSecurity
54                  .formLogin().loginPage("/login").failureUrl("/login?error") FormLoginConfigurer<H
55                  .usernameParameter("username").passwordParameter("password") FormLoginConfigurer<
56                  .and() HttpSecurity
57                  .rememberMe() RememberMeConfigurer<HttpSecurity>
58                  .tokenValiditySeconds(24 * 60 * 60) // expired time = 1 day
59                  .tokenRepository(persistentTokenRepository()) RememberMeConfigurer<HttpSecurity>
60                  .and() HttpSecurity
61                  .logout().logoutSuccessUrl("/login?logout") LogoutConfigurer<HttpSecurity>
62                  .and() HttpSecurity
63                  .csrf().disable() HttpSecurity
64                  .exceptionHandling().accessDeniedPage("/403") ExceptionHandlingConfigurer<HttpSecurity>
65                  .and() HttpSecurity
66                  .sessionManagement() SessionManagementConfigurer<HttpSecurity>
67                  .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED) SessionManagementConfigu
68                  .and() HttpSecurity
69                  .headers().disable().headers() HeadersConfigurer<HttpSecurity>
70                  .frameOptions().disable();
71      }
72
73      @Bean
74      public PersistentTokenRepository persistentTokenRepository() {
75          JdbcTokenRepositoryImpl tokenRepository = new JdbcTokenRepositoryImpl();
76          tokenRepository.setDataSource(herokuDataSource);
77          return tokenRepository;
78      }
```

Figura 35 Spring Security 2

È necessaria inoltre l’implementazione di una classe aggiuntiva, che consente l’utilizzo di Spring Security:

```
5   public class SecurityWebInitializer extends AbstractSecurityWebApplicationInitializer {}
```

Figura 36 Spring Security 3

4.3 Gestione lingue

4.3.1 Spring Boot

La ricezione della richiesta POST è gestita nel modo seguente:

```
206     @RequestMapping(value = "/language/{language}/{page}/{db}", method = RequestMethod.POST)
207     public void updateDataLanguage(
208             @PathVariable String page, @PathVariable String language, @PathVariable String db,
209             HttpServletRequest request){
210         try{
211             int idPage = Integer.parseInt(page);
212             Page p = siteDao.getPageById(idPage);
213             if(p!=null) {
214                 boolean save = db.equals("true");
215                 String[] contents = request.getParameterValues( s: "contents");
216                 User u = getLoggedUser();
217                 if(u!=null){
218                     siteDao.saveLanguage(p, language, contents, save, u.getLink());
219                 }
220             }
221         }catch (NumberFormatException ex){
222             System.err.println("ERROR");
223         }
224     }
```

Figura 37 Richiesta POST linguaggi

Il metodo che viene richiamato è il seguente:

```
91      public boolean saveLanguage(Page p, String lan, String[] contents, boolean definitive, String user){
92          String isUser = "";
93          if(p != null) {
94              if(getPageById(p.getId()) == null){
95                  System.err.println("Page not found");
96                  return false;
97              }
98              String sql;
99              List<Content> cUser = getContentsForPageAndUser(p.getId(), lan, user);
100             for(int i = 0; i < contents.length; i++){
101                 if(getContent(p.getId(), i, lan, user) == null){
102                     sql = "insert into content (text, idPage, idText, language, linkUser) values (?,?,?,?,?)";
103                 }else{
104                     sql = "update content set text=? where idPage=? && idText=? && language=? && linkUser=?";
105                 }
106                 this.doChangeFilter(sql,  ...params: contents[i],p.getId()+"", i+"",lan, (definitive?"":user));
107             }
108             return true;
109         }
110         System.err.println("page not passed");
111         return false;
112
113     }
```

Figura 38 Dao per salvare linguaggi

4.3.2 Pagina JSP

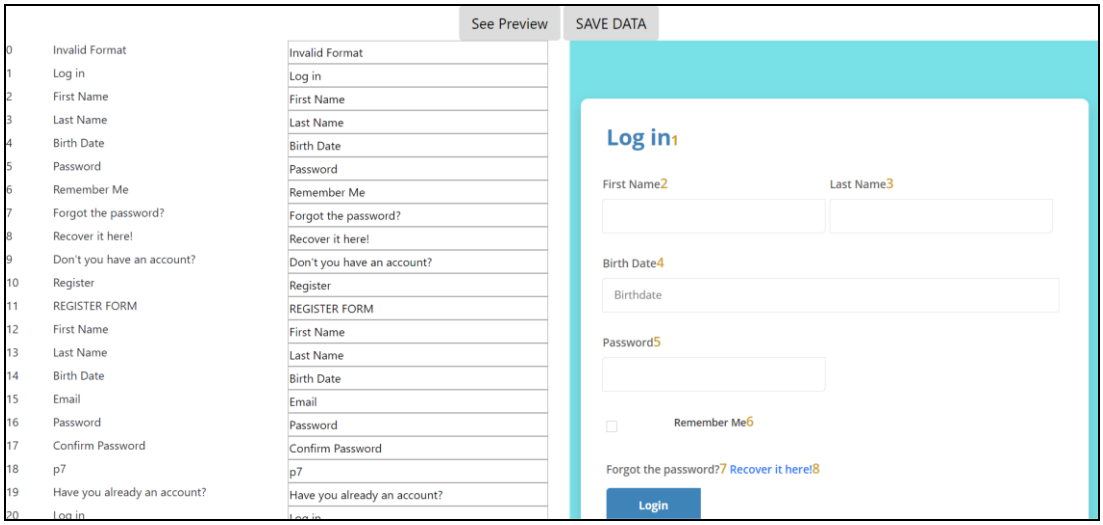

Figura 39 Pagina JSP per lingue

Il codice per creare ciò è veramente semplice:

```
<body>
<div class="row">
    <div class="col-sm-12 text-center btn-group-lg">
        <button class="btn" onclick="updateDB(false)">See Preview</button>
        <button class="btn" onclick="updateDB(true)">SAVE DATA</button>
    </div>
</div>
<div class="row">

    <div class="col-md-6">
        <form id="dataForm">
            <c:forEach begin="0" end="${page.contentCount-1}" varStatus="loop">
                <div class="row">
                    <div class="col-sm-1"><label>${loop.index}</label></div>
                    <div class="col-sm-5"><label class="fullSize">${contentsPrim.get(loop.index).text}</label></div>
                    <div class="col-sm-6">
                        <input class="fullSize" type="text" name="contents" value="${contents.get(loop.index).text}">
                    </div>
                </div>
            </c:forEach>
        </form>

    </div>
    <div class="col-md-6">
        <div class="embed-responsive embed-responsive-1by1">
            <iframe id="previewFrame" class="embed-responsive-item" src="${preview}">

            </iframe>
        </div>

    </div>
</div>
</body>
```

Figura 40 Pagina JSP linguaggi

Il salvataggio avviene tramite richiesta Ajax, differenziando tramite un booleano se il valore va salvato a livello di amministratore oppure generale:

```
function updateDB(saved) {
    var urlS = window.location.href + "/" + saved;
    console.log(urlS);
    $.ajax( url: {
        type: "POST",
        url: urlS,
        data: $("#dataForm").serialize(),
        error: function () {
            if (saved) {
                console.log(self.location);
                location = self.location;
            } else {
                document.getElementById( elementId: 'previewFrame').contentDocument.location.reload( forcedReload: true);
            }
        }
    });
    console.log(saved);
}
```

Figura 41 Js linguaggi

4.4 Gestione account

Il metodo in cui i dati vengono estrapolati è il seguente:

4.5 Pagina liste

I file Csv vengono salvati grazie a questo metodo:

```
@RequestMapping(value = "/users/download/{classD}/{fileName}/{type}/{phases}/{par1}/{par2}",
        produces = "application/csv")
public void exportFile(HttpServletResponse response, @PathVariable String classD, @PathVariable String fileName,
                       @PathVariable String type, @PathVariable String phases,
                       @PathVariable String par1, @PathVariable String par2) throws IOException {
    ServletOutputStream writer = response.getOutputStream();
    if (classD.equalsIgnoreCase( anotherString: "S")) {
        List<Subscription> subs = userDao.getSubscriptionsByEventAndPhases(par1, par2, phases);
        if (subs != null) {
            if (subs.size() > 0) {
                if (type.equalsIgnoreCase( anotherString: "pdf")) {

                } else if (type.equalsIgnoreCase( anotherString: "csv")) {
                    response.setContentType("application/csv");
                    response.setHeader( s: "content-disposition", s1: "attachment;filename =" + fileName + ".csv");
                    String separator = ";";
                    writer.println(subs.get(0).toCSV( key: true, separator));
                    writer.println();
                    for (Subscription s : subs) {
                        writer.println(s.toCSV( key: false, separator));
                    }
                }
            }
        }
```

Figura 42 Creazione File csv

Lo stesso procedimento è usato per gli utenti.

4.6 Gestione Parametri per Admin

La pagina è stata definita velocemente e presenta semplicemente una lista dei parametri con la possibilità di aggiungerne uno tramite campo input text e bottone che effettua la richiesta GET.

Il salvataggio viene eseguito nel seguente modo:

```
235     public boolean addSubscriptionColoumn(String content) {
236         if(!isAParameter(content)){
237             String c = "_" + content;
238             String sql2 = "insert into parameter(name) values (?)";
239             this.doChangeFilter(sql2, c);
240             String sql = "ALTER TABLE subscription add " + c + " varchar(250)";
241             return this.doChangeFilter(sql);
242         }
243         return false;
244     }
```

Figura 43 Salvataggio parametri extra

5 Test

5.1 Protocollo di test

Test Case: **Riferimento**:	TC-001	**Nome:**	Registrazione sito
Descrizione:	Registrarsi in modo da poter utilizzare le funzionalità base del sito.		
Prerequisiti:	Un computer, accesso a internet.		
Procedura:	Andare sulla pagina https://fsy-manager.herokuapp.com/login. Registrarsi inserendo le proprie credenziali e accettando le condizioni dell'account (al momento sostituite da una stringa "p7"). Abilitare l'account. Per farlo, controllare l'indirizzo email utilizzato per la registrazione e aprire il link che viene inviato tramite posta elettronica. A questo punto, effettuare il login.		
Risultati attesi:	Reindirizzamento sulla pagina Home, contenenti le informazioni base e la possiiblità di iscriversi agli eventi, così come di tenere d'occhio le iscrizioni già presenti. Gli admin ricevono un'email per segnalare loro che un nuovo utente si è registrato.		

Test Case: **Riferimento**:	TC-002	Nome:	Gestione errori registrazione/login
Descrizione:	Controllare che non sia possibile aggirare il sistema, inserendo valori non accettabili.		
Prerequisiti:	Un computer, accesso a internet.		
Procedura:	Andare sulla pagina https://fsy-manager.herokuapp.com/login. Per tutti i campi input, provare a forzare dei valori non accettabili, come numeri per nomi o una struttura per l'email non conforme.		
Risultati attesi:	Reindirizzamento sulla pagina Home, contenenti le informazioni base e la possiiblità di iscriversi agli eventi, così come di tenere d'occhio le iscrizioni già presenti.		

Test Case: **Riferimento**:	TC-003	**Nome:**	Funzione Ricordami
Descrizione:	Controllare che la spunta ricordami serva effettivamente		
Prerequisiti:	Un computer, accesso a internet.		
Procedura:	1. Andare sulla pagina https://fsy-manager.herokuapp.com/login. 2. Effettuare un login con un qualsiasi utente mettendo la spunta su ricordami. 3. Chiudere la pagina, riaprirla e controllare che l'utente sia rimasto loggato.		
Risultati attesi:	L'utente è loggato con 2 cookie utilizzati, uno per la sessione di login e l'altro necessario alla funzione ricordami.		

Test Case: **Riferimento**:	TC-004	**Nome:**	Iscrizione a un nuovo evento
Descrizione:	Registrarsi in modo da poter utilizzare le funzionalità base del sito.		
Prerequisiti:	Un computer, accesso a internet, un account al sito.		
Procedura:	1. Andare sulla pagina https://fsy-manager.herokuapp.com/login. 2. Effettuare un login e tramite la pagina home, registrarsi ad un evento tramite il link apposito. 3. Compilare i campi come si desidera per poi salvare tramite il pulsante update.		
Risultati attesi:	La nuova iscrizione è ora disponibile nell'elenco delle proprie iscrizioni. Gli amministratori ricevono un'email che li invita a controllare I dati inseriti dall'utente.		

Test Case: **Riferimento**:	TC-005	**Nome:**	Modifica iscrizione
Descrizione:	Modificare un'iscrizione già creata		
Prerequisiti:	Un computer, accesso a internet, un account al sito.		
Procedura:	1. Andare sulla pagina https://fsy-manager.herokuapp.com/login. 2. Effettuare un login e tramite la pagina home, registrarsi ad un evento tramite il link apposite (carica I dati) oppure andare su check it nell'elenco delle proprie iscrizioni. 3. Compilare i campi come si desidera per poi salvare tramite il pulsante update.		
Risultati attesi:	L'iscrizione è stata aggiornata con successo e gli amministratori son stati avvisati tramite email. Gli amministratori ricevono un'email che li invita a controllare I dati inseriti dall'utente.		

Test Case: **Riferimento**:	TC-006	**Nome:**	Eliminazione iscrizione
Descrizione:	Eliminare un'iscrizione già creata		
Prerequisiti:	Un computer, accesso a internet, un account al sito, un'iscrizione da eliminare.		
Procedura:	1. Andare sulla pagina https://fsy-manager.herokuapp.com/login. 2. Effettuare un login e tramite la pagina home, eliminare un'iscrizione tramite tasto apposito e confermare l'operazione nel modal che esce.		
Risultati attesi:	L'iscrizione è stata eliminate e non è più visibile all'interno della propria lista. Gli amministratori ricevono un'email che li avvisa della disiscrizione dell'utente dall'evento.		

Test Case: **Riferimento**:	TC-007	**Nome:**	Pagina account in lettura
Descrizione:	Leggere I dati di un utente senza poterli modificare.		
Prerequisiti:	Un computer, accesso a internet, un account utente e un account per leggere (assistente, consigliere o admin), un'iscrizione presente.		
Procedura:	1. Andare sulla pagina https://fsy-manager.herokuapp.com/login e loggare tramite un account abilitato a leggere i dati. 2. Tramite la pagina https://fsy-manager.herokuapp.com/users andare a visualizzare i dati di un utente.		
Risultati attesi:	I dati sono visibili ma non modificabili.		

Test Case: **Riferimento**:	TC-008	**Nome:**	Pagina account per Admins
Descrizione:	Leggere I dati di un utente		
Prerequisiti:	Un computer, accesso a internet, un account utente e un account admin, un'iscrizione presente.		
Procedura:	1. Andare sulla pagina https://fsy-manager.herokuapp.com/login e loggare tramite un account admin. 2. Tramite la pagina https://fsy-manager.herokuapp.com/users andare a visualizzare i dati di un utente.		
Risultati attesi:	I dati sono visibili ma non modificabili. Sono presenti campi aggiuntivi checkbox oltre che un campo per inserire note extra da parte dell'utente.		

Test Case: **Riferimento**:	TC-009	**Nome:**	Segnalazione correzione Admins
Descrizione:	Segnalare correttezza valori oppure eventuali valori da modificare all'utente		
Prerequisiti:	Un computer, accesso a internet, un account utente e un account admin, un'iscrizione presente.		
Procedura:	1. Andare sulla pagina https://fsy-manager.herokuapp.com/login e loggare tramite un account admin. 2. Tramite la pagina https://fsy-manager.herokuapp.com/users andare a visualizzare i dati di un utente. 3. Modificare i checkbox e inserire una stringa all'interno del campo note dedicato.		
Risultati attesi:	L'utente riceve un'email che è tutto a posto oppure con I valori da modifcare con le note definite dall'admin. Gli altri admin ricevono un'email per segnalare loro che il caso è stato gestito, indicando anche quale admin ha eseguito l'azione.		

Test Case: **Riferimento**:	TC-010	**Nome:**	Pagina filtri
Descrizione:	Visualizzare iscrizioni e utenti in maniera comoda.		
Prerequisiti:	Un computer, accesso a internet, un account col permesso di leggere, qualche utente e qualche iscrizione per provare i filtri.		
Procedura:	1. Andare sulla pagina https://fsy-manager.herokuapp.com/login e loggare tramite un account col permesso di lettura. 2. Tramite la pagina https://fsy-manager.herokuapp.com/users andare a visualizzare i dati. 3. Provare il filtro a lato e vederne i risultati. 4. Effettuare l'operazione sia per il filtro subscriptions che quello users.		
Risultati attesi:	La pagina si aggiorna di conseguenza mostrando solamente I valori inerenti al nuovo filtro.		

Test Case: **Riferimento**:	TC-011	**Nome:**	Scaricare file CSV o PDF filtrati
Descrizione:	Esportare le liste di dati di iscrizioni e utenti in format CSV e PDF.		
Prerequisiti:	Un computer, accesso a internet, un account col permesso di leggere, qualche utente e qualche iscrizione per provare i filtri.		
Procedura:	1. Andare sulla pagina https://fsy-manager.herokuapp.com/login e loggare tramite un account col permesso di lettura. 2. Tramite la pagina https://fsy-manager.herokuapp.com/users andare a visualizzare i dati. 3. Provare il filtro a lato. 4. Scaricare il file tramite il bottone apposito. 5. Provare entrambi i formati file (csv e pdf). 6. Effettuare l'operazione sia per il filtro subscriptions che quello users.		
Risultati attesi:	I file si scaricano correttamente e contengono solamente i valori filtrati dall'utente.		

Test Case: **Riferimento**:	TC-012	**Nome:**	Promozione di un utente
Descrizione:	Modificare I permessi di un utente è disponibile solamente agli amministratori.		
Prerequisiti:	Un computer, accesso a internet, un account utente e un account admin.		
Procedura:	1. Andare sulla pagina https://fsy-manager.herokuapp.com/login e loggare tramite un account admin. 2. Tramite la pagina https://fsy-manager.herokuapp.com/users andare a visualizzare i dati di un utente. 3. Promuovere un utente qualsiasi ad un nuovo ruolo.		
Risultati attesi:	L'utente ora avrà I permessi modificati e potrà effettuare più o meno funzioni a dipendenza del gruppo (user, assistant, counselor, admin).		

Test Case: **Riferimento**:	TC-013	**Nome:**	Aggiunta parametri
Descrizione:	Aggiungere parametric extra all'iscrizione.		
Prerequisiti:	Un computer, accesso a internet, un account admin, un'iscrizione presente.		
Procedura:	1. Andare sulla pagina https://fsy-manager.herokuapp.com/login e loggare tramite un account admin. 2. Tramite la pagina https://fsy-manager.herokuapp.com/admin/parameters andare a visualizzare i parametri presenti. 3. Aggiungere tramite input text e bottone apposito un nuovo parametro.		
Risultati attesi:	Il parametro sarà aggiunto alla lista presente. I parametri aggiunti hanno "_" davanti al nome per indicare il fatto che non fanno parte dei parametri originali.		

Test Case: **Riferimento**:	TC-014	**Nome:**	Visualizzazione traduzioni
Descrizione:	Salvare la traduzione per l'amministratore.		
Prerequisiti:	Un computer, accesso a internet, un account admin.		
Procedura:	1. Andare sulla pagina https://fsy-manager.herokuapp.com/login/{lingua da visualizzare} 2. Controllare più lingue (implementate: ITA e ENG).		
Risultati attesi:	I vari testi vengono caricati in base alla traduzione presente nel database, permettendo così di cambiare lingua. La lingua di default è l'inglese.		

Test Case: **Riferimento**:	TC-015	**Nome:**	Modifica temporanea lingua pagina Linguaggi
Descrizione:	Salvare la traduzione per l'amministratore.		
Prerequisiti:	Un computer, accesso a internet, un account admin.		
Procedura:	Andare sulla pagina https://fsy-manager.herokuapp.com/login e loggare tramite un account admin. Tramite la pagina https://fsy-manager.herokuapp.com/admin/language/{lingua da modificare}/{numero della pagina} per modificare la traduzione. Modificare qualche valore della pagina. Vedere la preview del sito. Chiudere la pagina e riaprirla. Andare a visualizzare la pagina effettiva del quale si è creato/modificato le traduzioni.		
Risultati attesi:	Nell'iframe a lato è visibile la preview e anche tornando alla pagina l'amministratore mantiene i suoi progressi. Andando sulla pagina effettiva (per esempio di login), la traduzione effettuata non è visibile.		

Test Case: **Riferimento**:	TC-016	**Nome:**	Modifica definitiva lingua pagina Linguaggi
Descrizione:	Salvare la traduzione per l'intero sito.		
Prerequisiti:	Un computer, accesso a internet, un account admin.		
Procedura:	Andare sulla pagina https://fsy-manager.herokuapp.com/login e loggare tramite un account admin. Tramite la pagina https://fsy-manager.herokuapp.com/admin/language/{lingua da modificare}/{numero della pagina} per modificare la traduzione. Modificare qualche valore della pagina. Salvare i dati. Chiudere la pagina e riaprirla. Andare a visualizzare la pagina effettiva del quale si è creato/modificato le traduzioni.		
Risultati attesi:	Nell'iframe a lato è visibile la preview e anche tornando alla pagina l'amministratore mantiene i suoi progressi. Andando sulla pagina effettiva (per esempio di login), la traduzione effettuata è visibile per tutti.		

5.2 Risultati test

Numero Test	Risultato Test	Possibili Soluzioni
TC-001, Registrazione sito	Funziona come previsto nei risultati attesi.	-
TC-002, Gestione errori registrazione/login	Funziona come previsto nei risultati attesi, se non per il fatto che come controllo non limita veramente l'utente, in quanto tiene conto solamente dell'accettabilità dell'ultimo valore (e password)	Aggiunta di un controllo di tutti i campi che blocca le operazioni fino a quando è presente almeno un campo con valore errato.
TC-003, Funzione Ricordami	Funziona come previsto nei risultati attesi.	-
TC-004, Iscrizione a un nuovo evento	Funziona come previsto nei risultati attesi.	-
TC-005, Modifica iscrizione	Funziona come previsto nei risultati attesi.	-
TC-006, Eliminazione iscrizione	Funziona come previsto nei risultati attesi.	-
TC-007, Pagina account in lettura	Funziona come previsto nei risultati attesi.	-
TC-008, Pagina account per Admins	Funziona come previsto nei risultati attesi.	-
TC-009, Segnalazione correzione Admins	Non funzionante.	Si viene a generare un errore al momento dell'estrapolazione dati. Dovrei analizzare meglio quella parte di codice per capire dove trova errori.
TC-010, Pagina filtri	Ha problemi, visto che a volte carica solo determinati filtri.	Dovrei capire meglio il funzionamento della parte di JS che gestisce il filtro per risolvere il problema.

TC-011, Scaricare file CSV filtrati	Funziona come previsto nei risultati attesi per quanto riguarda i file csv, mentre quelli pdf non sono stati implementati.	Gestire correttamente la creazione e scaricamento del file dovrebbe essere un'unione tra lo scaricamento del csv e la creazione di un file PDF, cosa su cui mi son documentato solamente in parte.
TC-012, Promozione di un utente	Funziona come previsto nei risultati attesi.	-
TC-013, Aggiunta parametri	Funziona come previsto nei risultati attesi.	-
TC-014, Visualizzazione traduzioni	Funziona come previsto nei risultati attesi per quanto riguarda la pagina di login.	Implementare la funzione nelle altre pagine dovrebbe essere semplice, visto che la struttura già presuppone l'inserimento di un valore per indicare la pagina sul quale si vogliono effettuare/leggere le traduzioni.
TC-015, Modifica temporanea lingua pagina Linguaggi	Funziona come previsto nei risultati attesi per quanto riguarda la pagina di login.	Implementare la funzione nelle altre pagine dovrebbe essere semplice, visto che la struttura già presuppone l'inserimento di un valore per indicare la pagina sul quale si vogliono effettuare/leggere le traduzioni.
TC-016, Modifica definitiva lingua pagina Linguaggi	Funziona come previsto nei risultati attesi.	-

5.3 Mancanze/limitazioni conosciute

Purtroppo non son riuscito a completare molte funzionalità.
Partendo con ordine, abbiamo la password dimenticata nel login, niente di così difficile ma alla quale non ho dedicato tempo, la risposta degli admin riguardanti i parametri da mettere a posto, la quale era nella mia testa praticamente completata ma ora non funziona e non ho tempo di capirne il motivo.
I parametri extra vengono solamente aggiunti, non cancellati o modificati, mentre invece le lingue al momento son implementate solamente per la pagina di login e solo per la lingua inglese e italiano, rendendo di fatto la cosa ad un passo dal poter essere implementata ma non abbastanza da poterla definire tale.
Manca inoltre (e questa forse è la mancanza più grande) tutta la parte riguardanti i file PDF, sia per quanto riguarda download e upload da parte dell'utente in modo da permettergli di firmare e simili sia per le liste filtrabili e scaricabili da admin, assistenti e consiglieri.
Per quanto riguarda il filtro, funziona correttamente al momento in cui si scarica ma non sulla pagina, e questo è dovuto ad un codice JS non proprio intuitivo per estrapolare l'errore e risolverlo (è un template che ho trovato su internet).

5.4 Consuntivo

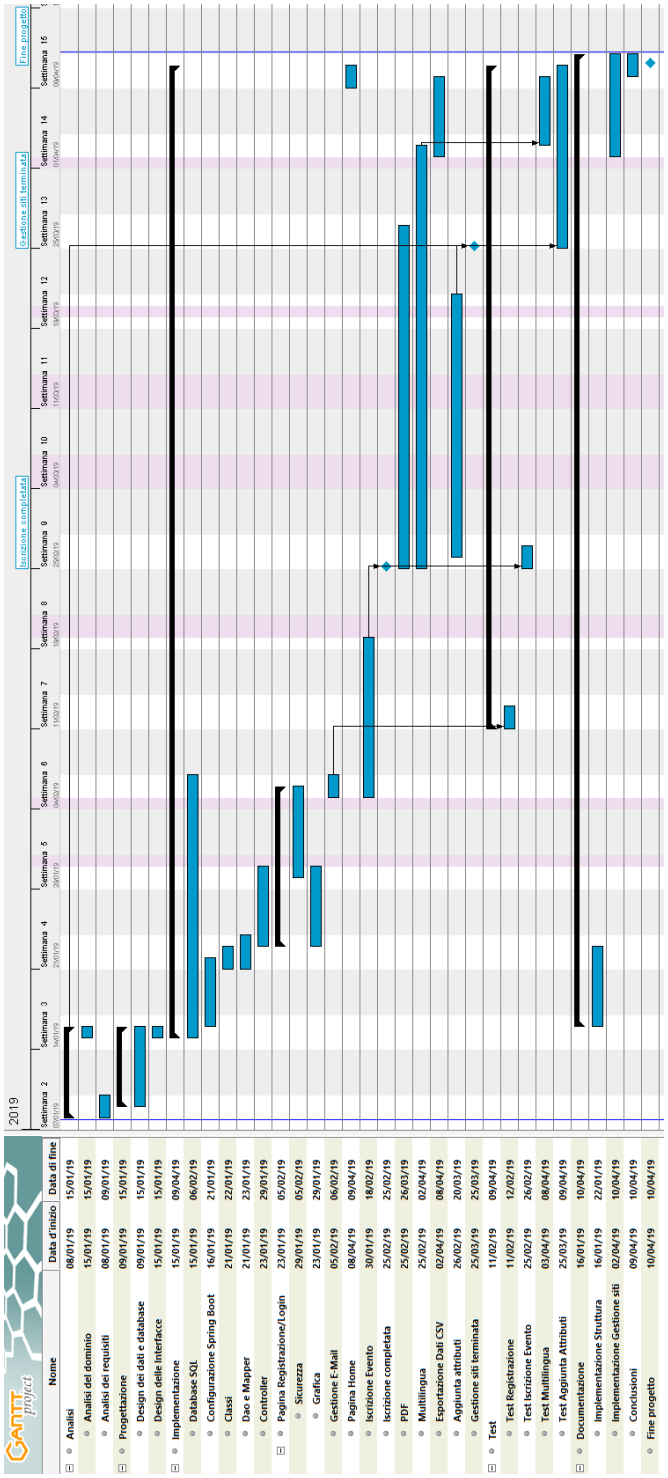

Figura 32 Gantt Consuntivo

Il progetto ha subito molti rallentamenti dovuti principalmente a perdita di ore per malattia e il mancato recupero a casa. Si è tutto concentrato nelle ultime settimane, dove ho cercato di portare a termine il maggior numero di funzioni sacrificando così controlli, grafica e commenti.

6 Conclusioni

Quali sono le implicazioni della mia soluzione? Che impatto avrà? Cambierà il mondo? È un successo importante? È solo un'aggiunta marginale o è semplicemente servita per scoprire che questo percorso è stato una perdita di tempo? I risultati ottenuti sono generali, facilmente generalizzabili o sono specifici di un caso particolare? ecc

6.1 Sviluppi futuri

Le migliorie principali da applicare riguardano le mancanze attuali, in quanto era un progetto bello corposo e con una visione completa di quello che era la gestione del sito.
Una possibile miglioria, abbozzata solamente nella mia testa e che ho capito presto essere difficile da realizzare, è un gestionale per quanto riguarda la creazione delle pagine, assegnando così url e strutturando tag html e contenuti in base a quanto faceva l'admin.

6.2 Considerazioni personali

Nel corso di questo progetto ho potuto consolidare ulteriormente il sistema Spring Boot riguardo alla programmazione in ambito Web. Questo mi ha permesso di avere più spazio di manovra e in generale più flessibilità, sebbene molte cose, avendole già sviluppate in un modo specifico in passato, non sono andato a cambiarle (vedi ad esempio Dao, nonostante tutti i sistemi per comunicare al DB).
Purtroppo la cosa più grande che ho imparato è che la gestione del tempo non è da sottovalutare. Mi ero previsto due settimane cuscinetto, che sono finite per essere state consumate da periodi di malattia. Non recuperando quelle ore di progetto se non in minima parte, sento di non essere riuscito a sfruttare il tempo per completare o quasi il progetto, traguardo che mi è sembrato fattibile fino all'ultima settimana, dove invece purtroppo mi son dovuto arrendere al fatto di un lavoro incompleto e pieno di sbavature.

7 Glossario

- Spring: Framework di Java per l'implementazione Web.
- Java: Linguaggio di programmazione
- Framework: Architettura di sviluppo sul quale ci si basa un software. Scheletro di programmazione.
- Spring Boot: Avanzamento del framework di Java, che permette una configurazione
- MVC: Framework che utilizza il sistema Model View Controller, che separa appunto la struttura tra model e quindi oggetti, view e quindi siti web, controller e quindi collegamento tra oggetti e pagine web.
- HTML: HyperText Markup Language. Scheletro di un sito, basato su numerosi tag per strutturare il contenuto di una pagina.
- CSS: Cascading Style Sheets, utilizzato per la grafica e il posizionamento della pagina web.
- Bootstrap: Libreria prefatta di css e js comoda da utilizzare.
- JS: JavaScript, gestisce piccole parti di logica del sito web, permettendo di mallearne il contenuto.
- JSP: JavaServer Pages, pagine web che utilizzano normalmente html, css e js con la possibilità di aggiungere codice Java tramite tag appositi.
- Dao: Collegamento tra DB e le classi, permettendo di effettuare query sul database.
- DB: DataBase, Banca dati contententi tutti i dati che vengono utilizzati.
- Query: richiesta al database tramite il quale è possibile salvare, modificare, eliminare o leggere i dati presenti nel DB.
- E-R: Entità-Relazione, schema tramite il quale è possibile avere un'idea precisa della struttura del database.
- Gantt: Progettazione in ore o giorni del lavoro da effettuare.

8 Bibliografia

8.1 Bibliografia per articoli di riviste:

Non è stata utilizzata alcun articolo per lo sviluppo del progetto.

8.2 Bibliografia per libri

Non è stata utilizzata alcun libro per lo sviluppo del progetto.

8.3 Sitografia

https://www.w3schools.com/, *W3Schools*
https://stackoverflow.com/ , *StackOverflow*
https://www.baeldung.com/, *Baeldung*
https://www.mkyong.com/spring-boot/, *Mkyong*
https://spring.io/blog/, *Spring Documentation*
StackOverflow è la principale fonte di soluzioni a problemi trovati riguardanti la programmazione. I restanti sono stati utilizzati per comprendere e ricordare come strutturare la programmazione Spring Boot.

9 Didascalia

10 Allegati

- Allegato A: Diari di Progetto