Geo Alley,

progetto prova finale di ingegneria del software

Matteo Gobbi Frattini
Cod. P: 10581031
Matricola: 886787
Data: 07/02/2023

Sommario

Dominio Applicativo

Benché online siano reperibili numerose mappe relative ad ogni singolo luogo, rappresentative di aspetti differenti (strade, trasporti, servici, eccetera) esse si possono trovare su vari siti diversi, e spesso risulta difficile per un utente reperire informazioni complete senza dover passare attraverso numerosi siti per la sua ricerca.

Ad esempio, un utente che sta organizzando un viaggio in una certa città o regione potrebbe avere la necessità di visualizzare diverse mappe, come quella dei luoghi di interesse, quella del trasporto pubblico, quella dei parcheggi, eccetera, e per farlo dovrebbe passare dal sito turistico, da quello dell'agenzia dei trasporti, eccetera.

La soluzione proposta gli consentirebbe di effettuare la ricerca con semplicità e immediatezza, poiché basterebbe cercare il luogo di interesse e gli comparirebbero tutte le mappe disponibili filtrabili per tag.

Pertanto lo scopo di questa applicazione sarebbe quella di poter accedere a tutte le mappe relative a un luogo da una singola applicazione di semplice utilizzo.

Inoltre, sfruttando la potenziale presenza di appassionati di geografia sulla piattaforma, si vuole dare agli utenti la possibilità di caricare sia mappe di cui non si conosce con esattezza il luogo rappresentate e/o il periodo storico, sia immagini di cui non si conosce la collocazione geografica.

Negli ultimi, nella nicchia degli appassionati di geografia, si è diffuso il trend di tentare di localizzare il luogo in cui sono state scattate le fotografie basandosi sugli elementi visibile sullo sfondo.

Geo Alley è quindi la soluzione ideale sia per appassionati di geografia che per semplici utenti interessati a determinati luoghi per diversi motivi, siano essi di viaggio, di ricerca o di lavoro, che con facilità riescono a ottenere le informazioni che cercano.

Ricapitolando, quindi, Geo Alley si rivolge a tre tipi di utenti principali:

- Coloro che per lavoro o necessità devono poter accedere a mappe a particolari tipi di mappe di determinati luoghi (ad esempio lo scrittore che intende ambientare il suo romanzo nella Milano del 1920 può trovare utile l'accesso a mappe di quel periodo per rendere la storia più realistica)
- Appassionati di geografia che vogliono visualizzare e confrontare mappe di luoghi, e "mettersi in gioco" tentando di indovinare luoghi e periodi storici di mappe e immagini caricate dalla terza categoria di utenti
- Semplici curiosi che hanno trovato una vecchia mappa e vogliono sapere a che anno risale, una vecchia foto e desiderano conoscere il luogo dove è stata scattata, oppure da una foto fatta in volo aereo vogliono sapere la località ripresa

Tecnologie Utilizzate

La soluzione è basata su una web application che integra in un unico componente le funzioni di Frontend e Backend, scritta in linguaggio Java e che si appoggia al Framework Spring Boot con integrazione della componente Thymeleaf.

L'ambiente di sviluppo utilizzato è Eclipse, con l'utilizzo di Apache Tomcat e Maven

La scelta dell'utilizzo di Spring Boot è stata dettata dal fatto che consente di concentrarsi sulla progettazione della logica applicativa automatizzando la realizzazione della base dell'applicazione come l'accesso al database.

La scelta dell'utilizzo Thymeleaf è data dal fatto che dà la possibilità di creare dei template HTML che si adattano ai dati da mostrare all'utente passati tramite l'oggetto Model.

Per la memorizzazione e gestione dei dati si è usato un database MySQL a cui ci si interfaccia per mezzo dell'implementazione di Hibernate JPA, che consente tramite annotations di automatizzare la scrittura delle query e la creazione delle tabelle.

Per semplificare all'utente la visualizzazione e l'inserimento di coordinate geografiche, che vengono usate estensivamente all'interno dell'applicazione vengono usate le API di Google Maps, che permettono di visualizzare una mappa di Google Maps in html e interagire con essa.

Requisiti funzionali

L'applicazione è accessibile da utenti sia registrati che non (con funzionalità limitate). L'idea è che chiunque deve essere in grado di ricercare e visualizzare le mappe e le loro informazioni senza il bisogno di essere registrati, appunto perché lo scopo dell'applicazione è rendere facilmente reperibili e accessibili le mappe.

Per poter caricare una mappa o una foto, poter commentare e leggere i commenti e poter segnalare contenuti errati è però necessario essere loggati con il proprio account.

Vi è inoltre una terza categoria di utenti, gli admin, che hanno possibilità di visualizzazione, modifica e cancellazione tu tutto e hanno a disposizione una schermata aggiuntiva con l'elenco delle segnalazioni.

Requisiti funzionali per categoria di utenti

Utente non registrato (**ospite**):

- Ricercare mappe tramite parole chiave
- Ricercare mappe tramite mappa
- Filtrare i risultati della ricerca per tag
- Registrarsi

Utente **registrato**:

- Tutte le funzioni dell'utente ospite
- Effettuare il Login
- Modificare le proprie informazioni
- Caricare mappe con tutti i dettagli (pronte per essere ricercate)
- Caricare mappe incomplete per chiederne i dettagli ad altri utenti
- Modificare ed eliminare le proprie mappe
- Commentare le mappe proprie e altrui
- Eliminare i propri commenti e i commenti dalle proprie mappe
- Caricare foto per chiedere ad altri utenti il luogo dello scatto
- Modificare ed eliminare le proprie foto
- Rispondere alle richieste di "aiuto foto" con la localizzazione che pensa
- Segnalare mappe o foto

Utente **admin**:

- Tutte le funzioni dell'utente registrato
- Modificare ed eliminare tutte le mappe e foto
- Modificare i dati degli utenti
- Eliminare tutti i commenti
- Visualizzare l'elenco delle segnalazioni aperte
- Chiudere le segnalazioni

Use case diagram

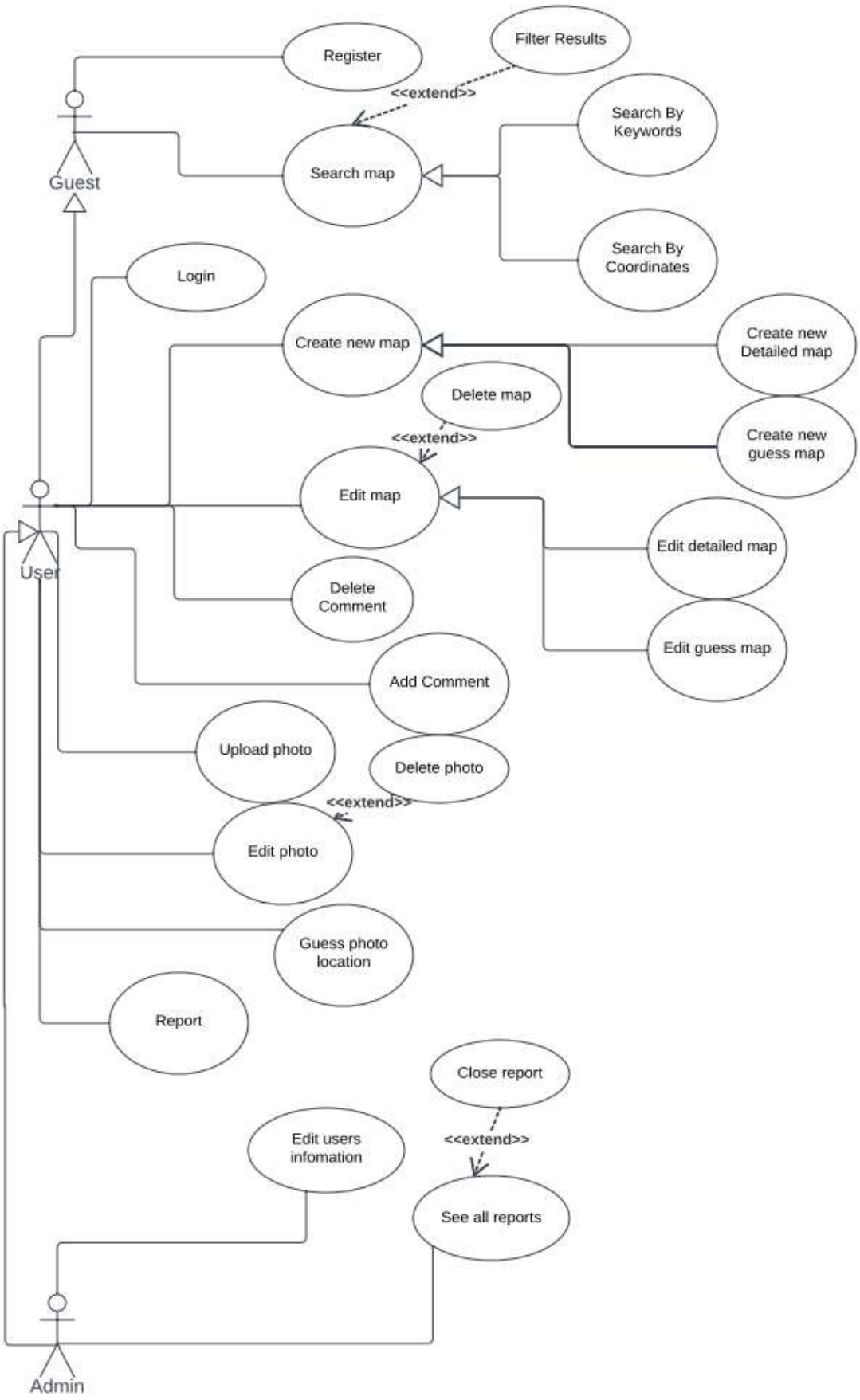
Nella prossima pagina è presento lo Use Case Diagram dell'applicazione, che sintetizza l'elenco dei requisiti funzionali. Si vedono i tre tipi di utenti, ognuno con le funzioni che può svolgere.

Si vede come admin generalizza user (che rappresenta l'utente registrato) che a sua volta generalizza guest (l'utente non registrato). Questo perché, come detto sopra, admin può svolgere tutte le funzioni di user, che a sua volta può svolgere tutte le funzioni di guest.

Inoltre ci sono altre tre relazioni di ereditarietà: La ricerca viene specializzata da ricerca per parole chiave e ricerca per coordinate, mentre caricamento e modifica mappa vengono specializzati da caricamento e modifica di mappa dettagliata o mappa incompleta.

La relazione di estensione tra la ricerca e il filtraggio dei risultati è dovuta al fatto che per poter filtrare i risultati è necessario fare una ricerca, ma non è detto che per ogni ricerca si vogliano filtrare i risultati.

Ragionamento simile per quanto riguarda la relazione di estensione tra modifica e cancellazione di mappa e foto, in quanto per poter eseguire la cancellazione bisogna accedere alla pagina di modifica.



Use Case Diagram

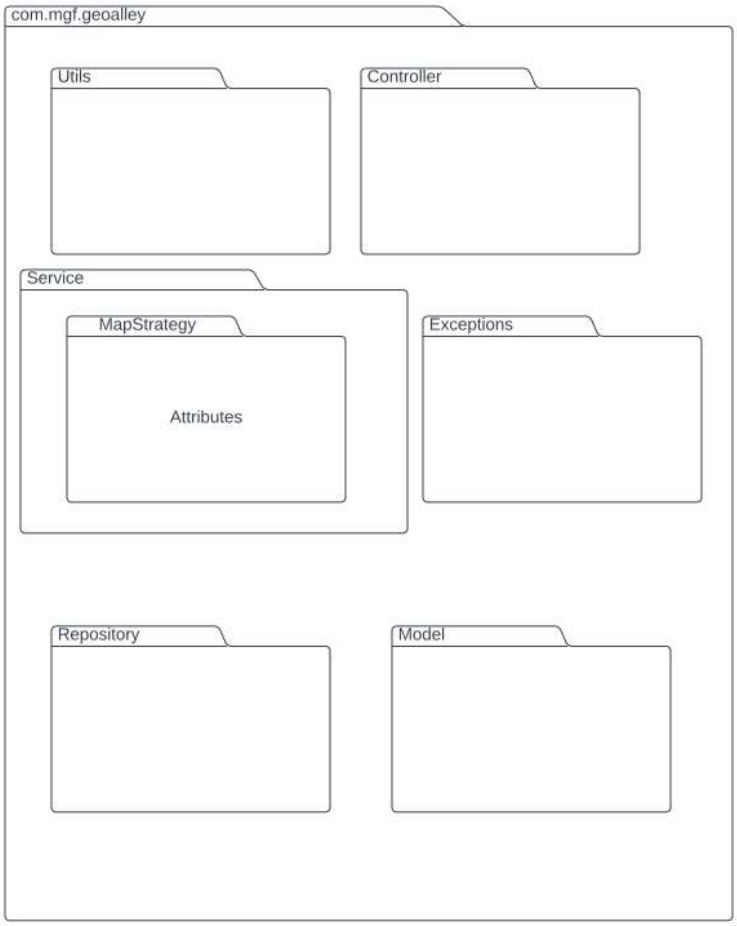
Architettura

Il sistema è organizzato secondo la logica classica di Spring Boot, secondo la quale le classi sono raggruppate in quattro gruppi principali:

- Controller, che ricevono le richieste dal browser e le gestiscono, chiamando metodi del Service
- Service, che gestiscono la logica applicativa
- Repository, che fanno da interfaccia tra i Service e il database
- Model, che sono le entità del progetto

Ho deciso di seguire questa struttura, creando un package per ognuno dei quattro gruppi, a cui ho aggiunto un package utils e un package exceptions per le eccezioni.
All'interno del package Service ho creato un package "mapStrategy" che include le classi utilizzate per l'utilizzo del pattern Strategy.

La struttura dei package è quindi riassunta dal seguente diagramma:



Utils

In questo package è contenuta una sola classe SecurityChecks, che offre dei metodi statici per i controlli di autenticazione utente. Questi metodi controllano se l'utente è loggato, se è un admin e i permessi di modifica di mappe, foto e quant'altro.

Per poter riutilizzare lo stesso metodo che controlla se l'utente è il proprietario di una risorsa senza doverlo riscrivere per ogni metodo ho usato i tipi generici: ogni model che ha tra gli attributi un utente proprietario implementa l'interfaccia "HasUser", che ha un unico metodo "getUser()". In questo modo il metodo checkOwnership e quelli che ne derivano sono applicabili a qualsiasi tipo di oggetto.

```
public static <T> boolean checkOwnership(User user, T object) {

    if(object instanceof HasUser) {
        HasUser obj =(HasUser) object;
        return obj.getUser().getId().equals(user.getId());
    }
    else
        return false;
}
```

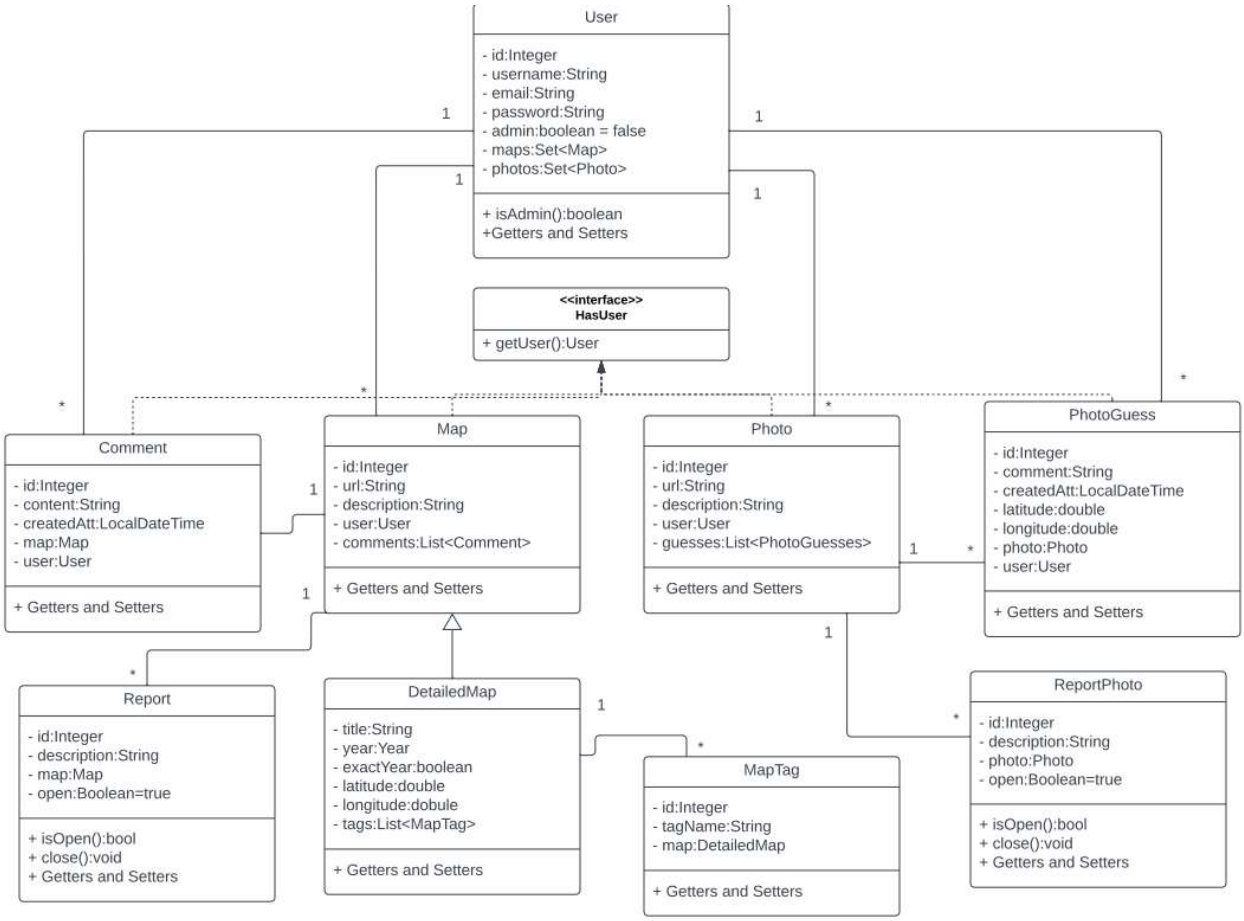
Model

In questo package sono presenti tutte le classi che sono mappate nel database da JPA, seguendo il class diagram alla pagina seguente.

Ogni classe è appropriatamente annotata per rendere il database più fedele possibile al modello.

```
@Entity
@Inheritance(strategy=InheritanceType.JOINED) //Relazione 1-1 ereditata in una seconda tabella nel database
@Table(name="map")
public class Map implements HasUser{

    //Id autogenerato dal database con auto-increment
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    private String url;
    private String description;

    @ManyToOne
    @JoinColumn(name="user_id")
    private User user;

    @OneToMany(mappedBy="map", cascade = CascadeType.REMOVE)
    private List<Comment> comments;
```

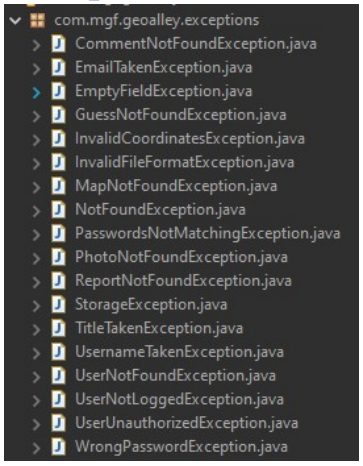
Esempio di annotazioni JPA



Si vede come le classi che hanno un attributo User implementano l'interfaccia HasUser, che permette l'utilizzo dei tipi generici nei metodi dei SecurityChecks.

Nella classe User esiste un attributo booleano "admin", che viene inizializzato a falso nel costruttore e non può essere modificato in alcun modo (manca il setter). Il motivo di questa scelta è l'idea che qualsiasi account venga creato tramite richieste dal browser sia un account non admin, e l'unico modo per creare account admin sia l'accesso diretto al database.

Exceptions



Questo package contiene le classi relative alle eccezioni che vengono lanciate e gestite estensivamente all'interno del progetto.

L'unica relazione tra di esse è data dalla classe "NotFoundException", che viene estesa da tutte le altre classi NotFound, ovvero:

MapNotFoundException
GuessNotFoundException
UserNotFoundException
PhotoNotfoundException
CommentNotFoundException
ReportNotFoundException

Controller

In questo package sono presenti tutte le classi che rispondono alle richieste http e restituiscono il template html appropriato.

C'è sostanzialmente un controller per ogni entità del package "model", oltre a un "DefaultController" che risponde alla richiesta di generare la homepage, la schermata di registrazione, la schermata di login e il logout.

I controller sono:

- AdminController, che risponde alle richieste di visualizzazione della pagina "admin" con l'elenco dei report aperti e degli utenti, e della pagina "allReports" con l'elenco di tutti i report aperti e chiusi
- CommentController, che risponde alle richieste di creazione e cancellazione dei commenti
- DefaultController, descritto in precedenza
- MapController, il più complesso, che risponde alle richieste di creazione, modifica, cancellazione, visualizzazione e ricerca di mappe (sia dettagliate che non)
- PhotoController, che risponde alle richieste di creazione, modifica, cancellazione e visualizzazione delle fotografie
- PhotoGuessController, che risponde alle richieste di creazione, visualizzazione e cancellazione di "guess", ovvero delle segnalazioni degli utenti dei luoghi in cui pensano siano state scattate le foto
- ReportController, che risponde alle richieste di creazione, visualizzazione e chiusura delle segnalazioni sulle mappe
- ReportPhotoController, che risponde alle richieste di creazione, visualizzazione e chiusura delle segnalazioni sulle foto
- UserController, che risponde alle richieste di registrazione, login, e visualizzazione e modifica del profilo

Nei controller si trova anche la gestione delle eccezioni, che vengono per lo più lanciate a livello Service, e reindirizzano l'utente alla pagina opportuna facendo utilizzo di RedirectAttributes.

Esempio di gestione delle eccezioni

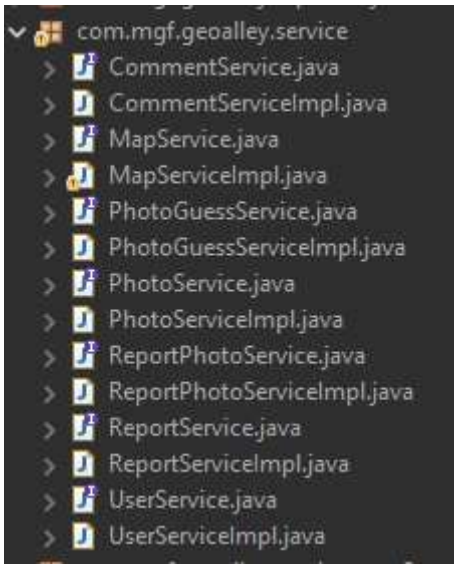
```
}catch(MapNotFoundException | UserUnauthorizedException e){
    redirectAttributes.addFlashAttribute("message", e.getMessage());
    return "redirect:/";
}catch (UserNotLoggedException e) {
    redirectAttributes.addFlashAttribute("message", "You must be logged to do that");
    return "redirect:/login";
} catch (EmptyFieldException | InvalidCoordinatesException | TitleTakenException e) {
    redirectAttributes.addFlashAttribute("message", e.getMessage());
    return editMapShow(id, model, request, redirectAttributes);
}
```

Service

Nel package Service sono presenti tutte le classi che implementano la logica applicativa vera e propria, che vengono chiamate dai controller e si interfacciano con i repositories.

Anche in questo caso si segue la classica disposizione in classi di interfaccia e classi di implementazione vera e propria.

Spesso alle funzioni dei service vengono passate anche la richiesta http (per effettuare i controlli sull'utente loggato e per ricevere i parametri della richiesta) e quasi sempre viene passato l'oggetto Model, a cui il service aggiunge gli attributi che serviranno al template Thymeleaf per generare l'opportuna pagina HTML.



L'unico controller a non avere un service dedicato è l'AdminController, perché i suoi metodi servono solo a generare le pagine dedicate all'admin, e per farlo utilizzano metodi di UserService e ReportService.

MapStrategy

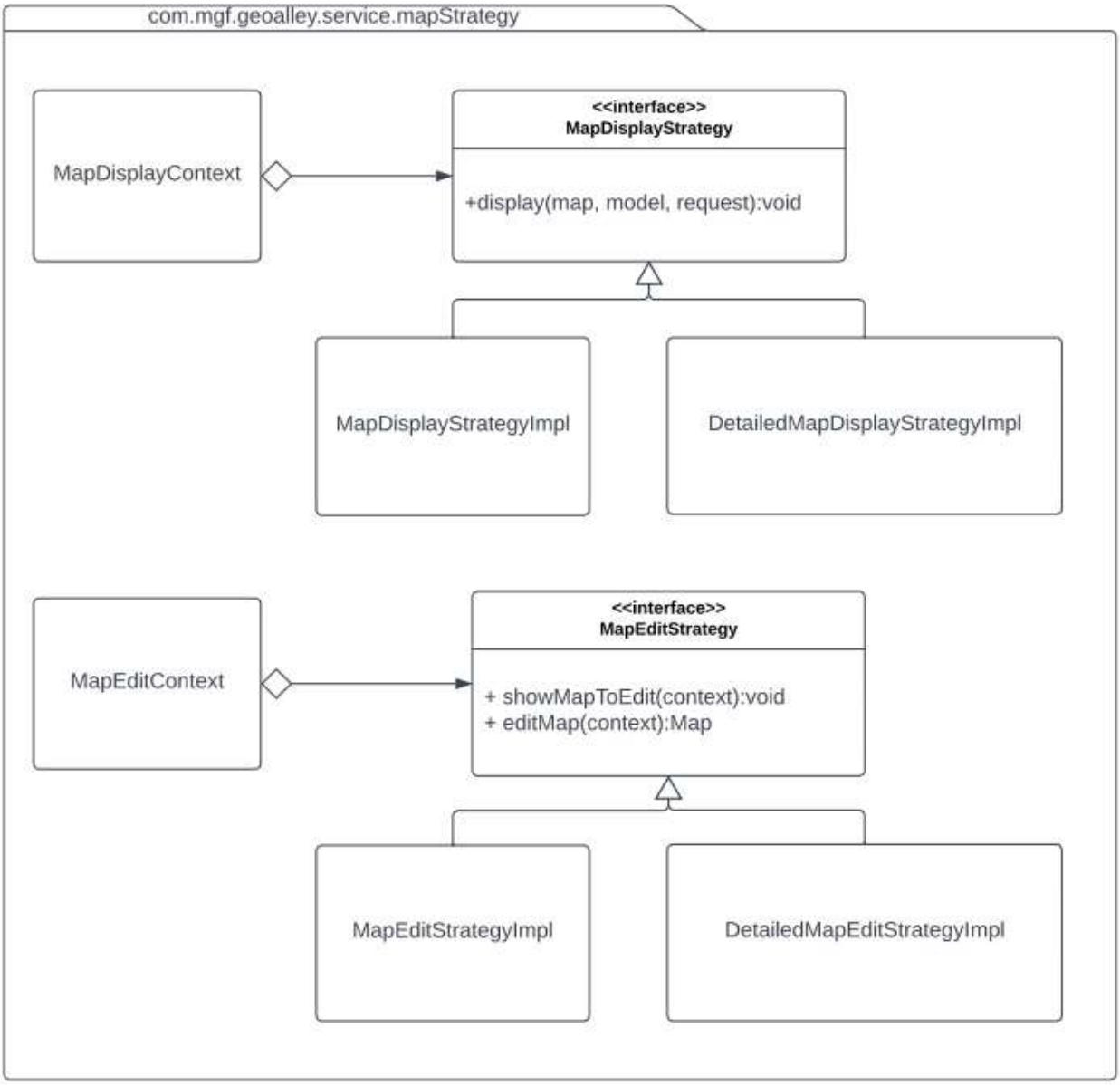
All'interno del package Service si trova anche un altro package, chiamato mapStrategy, in cui sono racchiuse le classi e le interfacce che servono a impiegare il pattern strategy.

Tale pattern viene impiegato due volte:

- Per la logica di creazione della pagina in cui viene mostrata una mappa, in risposta alla richiesta a /map/{idMappa}, in cui non si sa a priori se la mappa richiesta è una mappa dettagliata o meno
- Per la logica di modifica di una mappa (sia per visualizzare la pagina con il form per la modifica, sia per effettuare la modifica vera e propria), perché non si sa a priori se la mappa da modificare è una mappa dettagliata o meno.

Nella pagina seguente è possibile vedere il class diagram di questo package. In particolare MapEditContext racchiude tutti gli attributi che servono in entrambe le strategie, e dispone quindi di quattro costruttori in base agli attributi necessari.

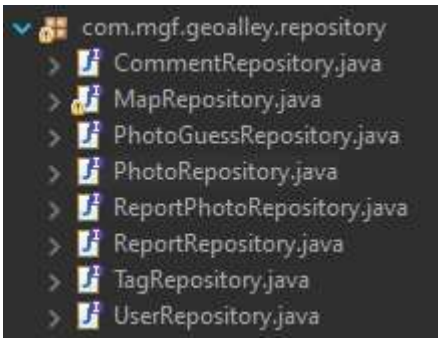
```
private MapEditStrategy editStrategy;
private Integer id;
private Map map;
private MapRepository mapRepository;
private TagRepository tagRepository;
private List<MapTag> tags;
private Model model;
private HttpServletRequest request;
//Attributo che indica se una mappa non dettagliata deve essere trasformata in una dettagliata
private boolean transform = false;
```



Nel MapService, quando bisogna chiamare in causa le classi del mapStrategy, si controlla di che oggetto è istanza la mappa in questione, si inizializza il context, si imposta la strategy esi chiama il metodo

```
//Preparazione context con gli attributi che serviranno e della strategy in base al tipo di mappa
if(map instanceof DetailedMap) {
    context = new MapEditContext(map, model, tagRepository);
    context.setStrategy(new DetailedMapEditStrategyImpl());
    showDetailed=true;
}
else {
    context = new MapEditContext(map, model);
    context.setStrategy(new MapEditStrategyImpl());
    showDetailed=false;
}

context.showMapToEdit();
return showDetailed;
```

Repository



Questo package contiene le classi che si interfacciano con il database. È standard nella sua progettazione, ovvero ogni componente del Model ha una propria interfaccia e tutte estendono JpaRepository.

Oltre ai metodi offerti di base da JpaRepository ho aggiunto diversi metodi per ogni classe tramite nomenclatura standard Jpa, e un metodo nel mapRepository con custom query.

Metodi tramite nomenclatura standard Jpa

```
public Optional<User> findByUsername(String username);

public Optional<User> findByEmail(String email);

public Optional<User> findByUsernameAndPassword(String username, String password);
```

Metodo con custom query

```
//Ritorna tutte le mabbe che non sono dettagliate
@Query("SELECT m FROM Map m WHERE m.id NOT IN (SELECT dm.id FROM DetailedMap dm)")
List<Map> findAllGuessMaps();
```

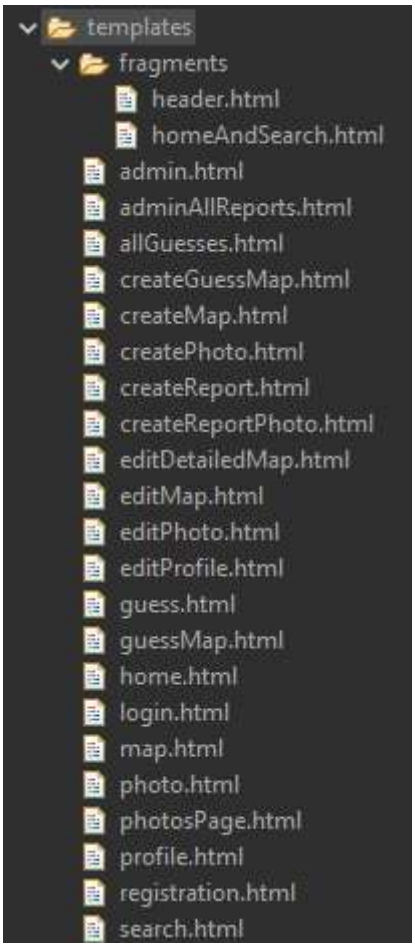
Pattern Optional<Type>

Nel Progetto si fa uso estensivo del pattern Optional, che permette di evitare il presentarsi di NullPointerException. Questo pattern è reso necessario dal fatto che alcuni metodi dei repository che ricercano un elemento per id, o per qualche altro parametro, possono non trovare nessun elemento corrispondente sul database, e quindi ritornare null.

Impiegando il pattern Optional rende più semplice gestire queste situazioni senza dover passare dalle eccezioni, ma verificando se l'oggetto Optional ritornato è presente oppure no.

```
//Metodo che controlla l'esistenza della mappa sul database e, se esiste, la ritorna
public Map findById(Integer id) throws MapNotFoundException {

    Optional<Map> mapOptional = mapRepository.findById(id);
    if(mapOptional.isPresent())
        return mapOptional.get();
    else
        throw new MapNotFoundException("Map does not exist");
}
```

Front End



Il front end di questa applicazione è sviluppato solo a scopo esemplificativo e di testing delle funzionalità. Un vero sviluppo della parte più grafica del front end deve essere immaginato come commissionato a terzi, quindi l'approccio che ho tenuto nello sviluppo è stato di evitare di usare codice CSS (se non in alcuni casi, usandolo direttamente nei tag HTML per rendere più leggibili le cose).

In generale è sviluppato tramite template Thymeleaf, che permette di passare oggetti all'html, e con qualche piccolo script JavaScript, soprattutto per l'interfacciamento con le API di Google Maps.

Gli attributi da passare ai template vengono caricati sul model con il comando model.AddAttribute sia dai controller che dai service.

```
//Preparazione degli attributi da mostrare nella pagina
List<User> users = userService.getAllUsers();
List<Report> reports = reportService.getOpenReports();
List<ReportPhoto> reportPhotos = reportPhotoService.getOpenReports();
model.addAttribute("users", users);
model.addAttribute("reports", reports);
model.addAttribute("reportPhotos", reportPhotos);

return "admin";
```

Esempio di caricamento di oggetti sul model

Interfaccia Grafica

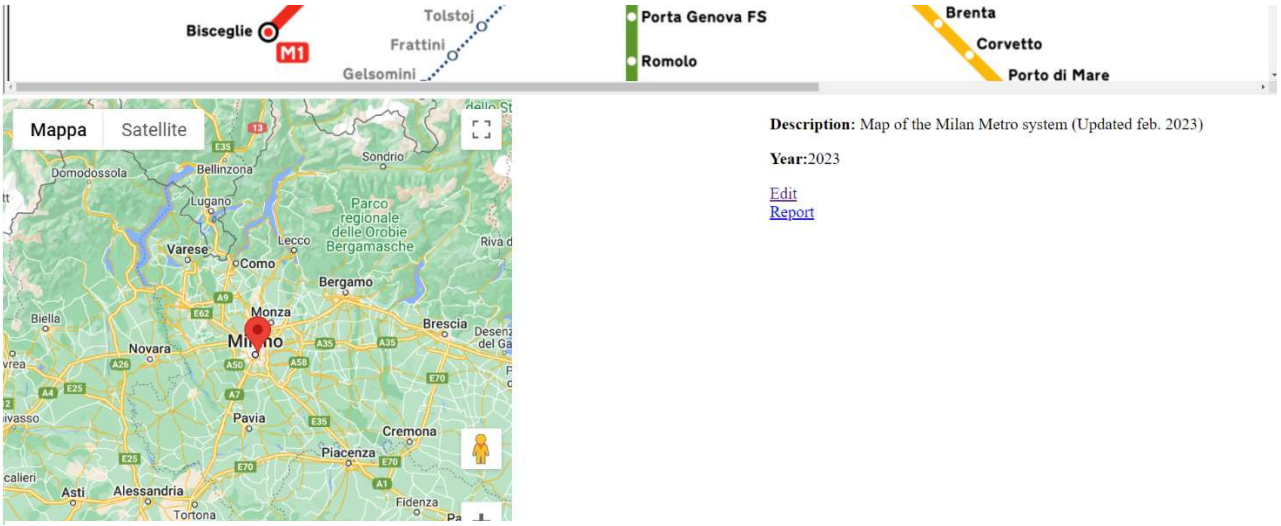
Come già detto l'interfaccia grafica è molto semplice e minimale perché sviluppata solo a scopo di esemplificare le funzionalità. L'unica parte che si può vedere come definitiva senza dover passare per terze parti è l'esperienza utente nell'interfacciamento con le mappe di Google Map.

Infatti è possibile dalla home ricercare mappe "per zona", inquadrando sulla mappa un'area e cliccando "cerca in questa zona".

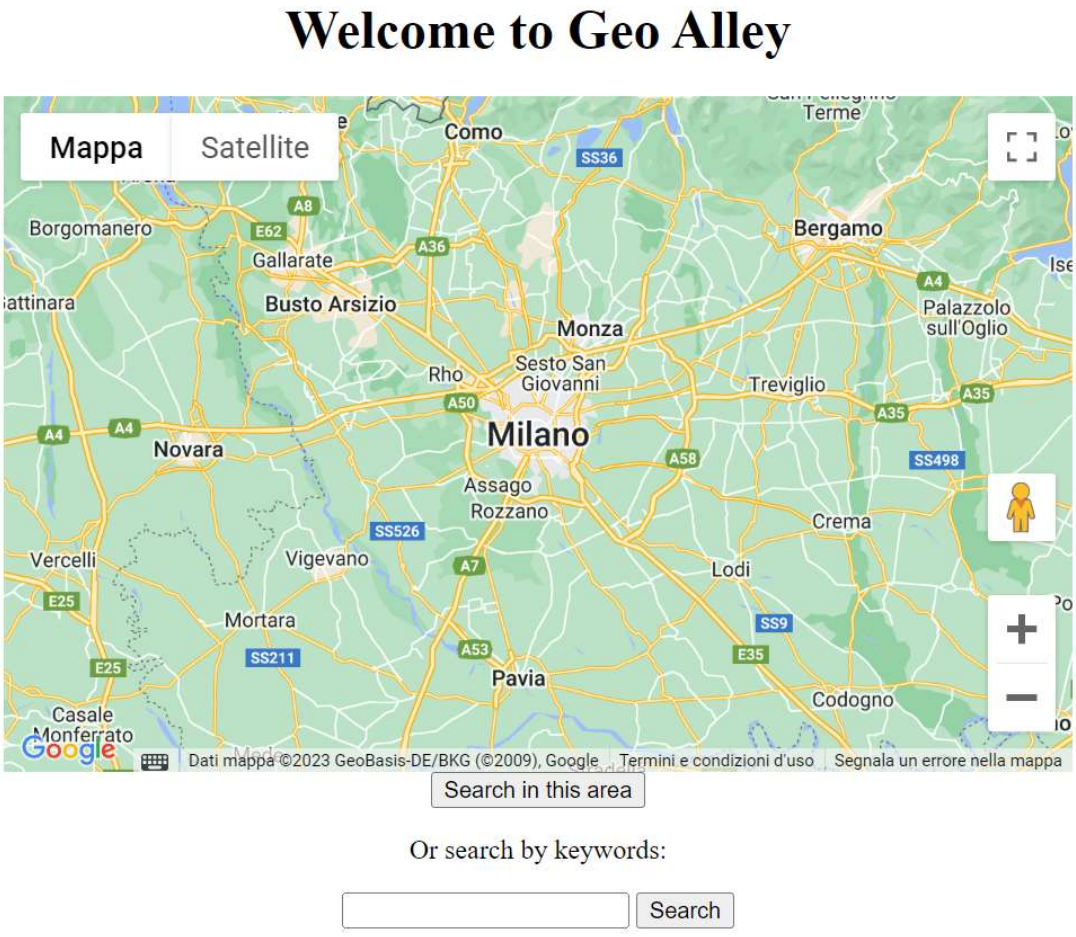
Inoltre, al momento della creazione di una mappa, è reso semplice e immediato l'inserimento delle coordinate: è sufficiente cliccare su un punto della mappa e le coordinate si compileranno da sole.

Un altro utilizzo delle mappe di Google Map è quello di poter visualizzare semplicemente, all'apertura di una delle mappe salvate, la posizione dell'area rappresentata.

Nella pagina seguente si vedono gli esempi di utilizzo di Google Maps nell'interfaccia grafica.



Visualizzazione posizione sotto la mappa della metro di Milano



Ricerca tramite mappa

```
<title>Map Details</title>
</head>
<body>

<div th:insert="fragments/header :: block-header"></div>

<h3 th:text="${map.title}"></h3>
<div th:if="${message}" th:text="${message}"></div>
<div style="display: inline-block;">
    Tags: <span th:each="tag : ${tags}" th:text="${tag.tagName} + ' - '">Tags: </span>
</div>


<iframe th:src="'http://localhost:8080/' + ${map.url}" width="100%" height="500"></iframe>

<div style="display: flex; width:100%">
    <div id="map" style=" float: left; width: 500px; height: 500px;"></div>

    <div style="position: absolute; right: 0; width: 40%;">
        <p><strong>Description:</strong> <span th:text="${map.description}"></span></p>
        <p><strong>Year:</strong><span th:if="${!map.exactYear}">Circa </span><span th:text="${map.year}"></span></p>
        <a th:if="${auth}" th:href="@{/edit/{id}(id=${map.id})}">Edit</a><br>
        <a th:if="${session.user}" th:href="@{/report/{id}(id=${map.id})}">Report</a>
    </div>
</div>

<h4>Comments</h4>
<div th:if="${session.user}">
    <div th:if="${commentMessage}" th:text="${commentMessage}"></div>
    <div th:if="${map.comments.size() > 0}">
        <ul>
```

Esempio di template Thymeleaf

Testing

Premettendo che il codice è stato meticolosamente testato fin dalla fase di scrittura del codice, testando singolarmente ogni nuova funzionalità tramite browser, la fase di Unit Testing finale è stata svolta come segue:

Il package repository si è assunto come funzionante in quanto formato da metodi “preconfezionati” e forniti da Jpa;

Il package model si è assunto come funzionante in quanto formato da soli metodi getter e setter;

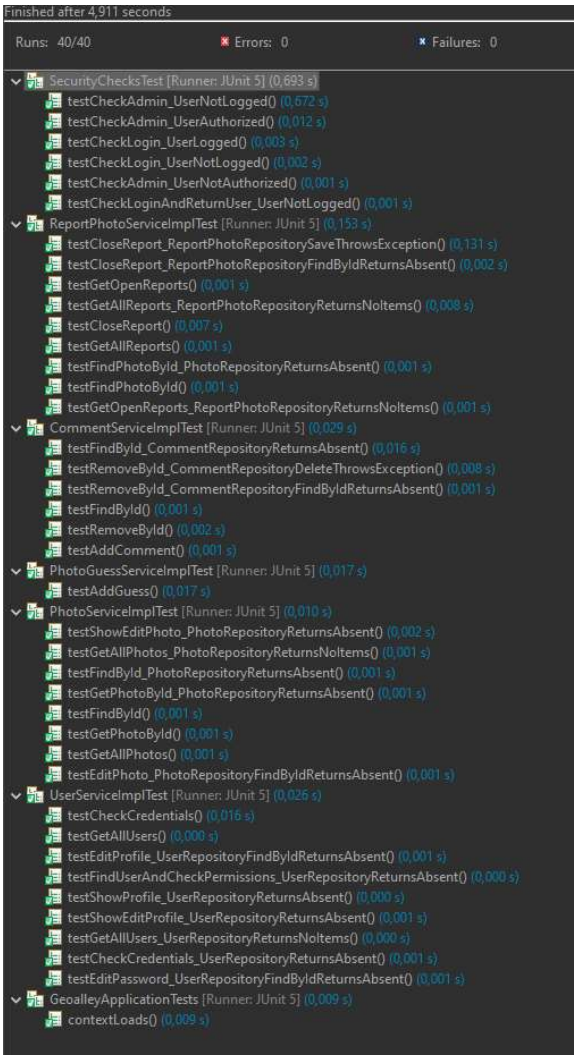
Il package service è stato testato con Junit su classi e servizi a campione, sfruttando Mockito.

Il package utils è stato testato con Junit sull’interezza della sua unica classe, vista l’importanza della gestione dei permessi

Il package controller è stato testato manualmente tramite interfaccia grafica su browser, scelta dettata dal fatto che i controller rispondono alle richieste http e generano le pagine html corrispondenti.

Risultati dei test

I risultati dei test automatici svolti con Junit sono riassunti to nella seguente immagine, e hanno raggiunto una copertura sull’intero progetto del 30%



I test manuali per i controller si sono svolti usando tre tipologie di utenti, di cui uno amministratore.

Ogni pagina e richiesta è stata testata anche ricorrendo alla funzione “ispeziona elemento” di chrome per inviare campi vuoti o errati.

L’utilizzo di tre utenti distinti ha consentito di testare i permessi di modifica e eliminazione (permessi riservati all’amministratore e al proprietario della risorsa).

Ogni singolo scenario di utilizzo e funzionalità è stata testata con tutti e tre i tipi di utente, raggiungendo così una copertura soddisfacente.