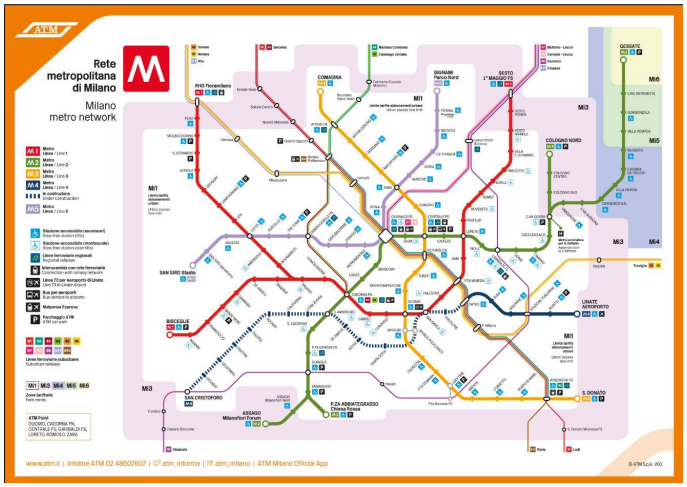
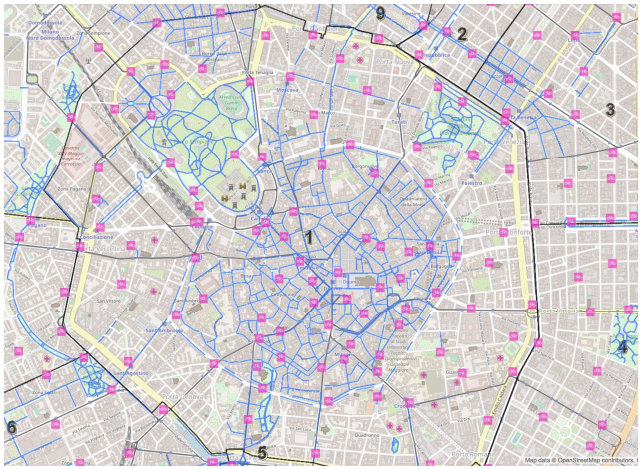
Geo Alley

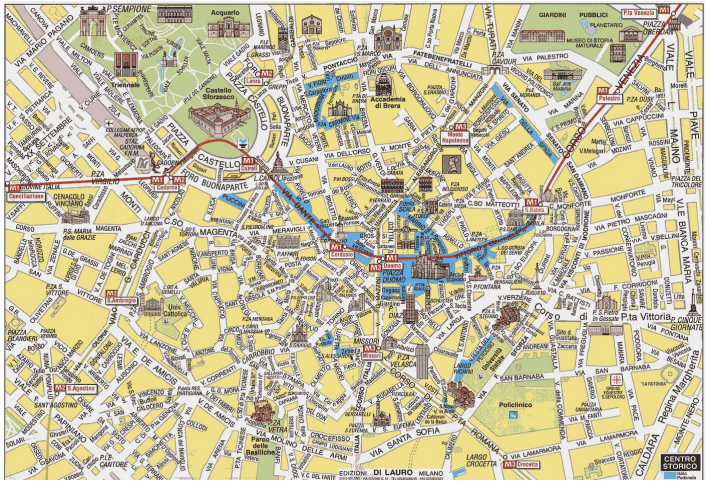
Matteo Gobbi Frattini

Dominio Applicativo

Online sono presenti numerose mappe relative ad ogni singolo luogo, ma sono spesso difficili da reperire senza passare attraverso numerosi siti

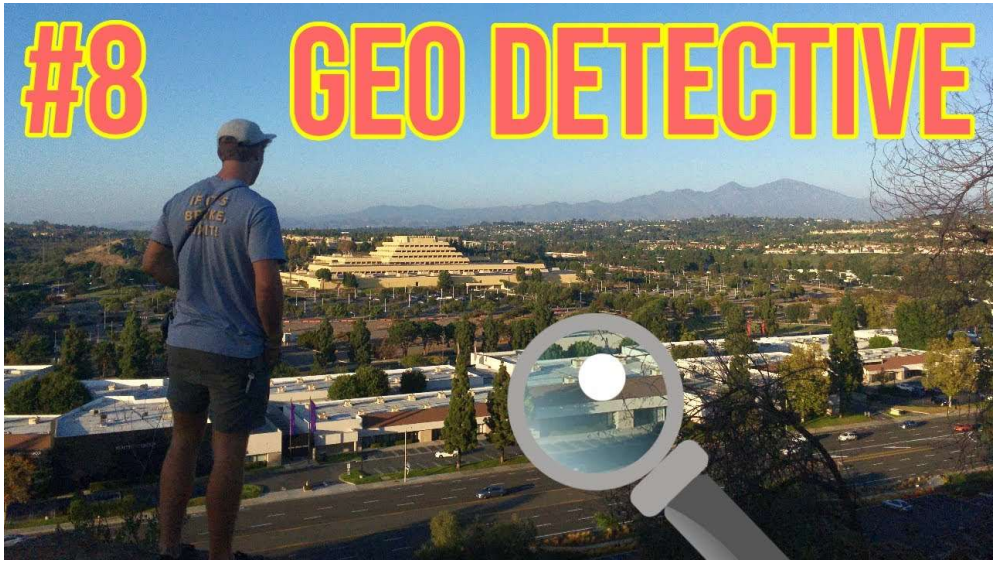






Dominio Applicativo

Si può sfruttare la potenziale presenza di appassionati di geografia sulla piattaforma



Dominio Applicativo

Geo Alley si rivolge a tre tipi di utenti principali:

- Coloro che per lavoro o necessità devono poter accedere facilmente a particolari tipi di mappe di determinati luoghi
- Appassionati di geografia che vogliono visualizzare mappe e provare a indovinare luoghi
- Semplici curiosi che hanno trovato una vecchia mappa o una vecchia foto e cercano aiuto nel trovare informazioni a riguardo

Funzionalità

Utente non registrato (ospite):

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

Funzionalità

Utente registrato:

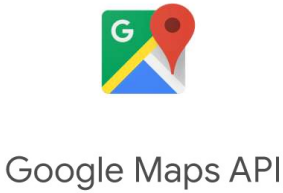
- Tutte le funzioni dell'utente ospite
- Effettuare il login
- Modificare le proprie informazioni
- Caricare mappe e foto
- Modificare ed eliminare le proprie mappe e foto
- Commentare
- Segnalare

Funzionalità

Utente admin:

- Tutte le funzioni dell’utente registrato
- Modificare ed eliminare tutte le mappe e foto
- Modificare i dati di tutti gli utenti
- Visualizzare l’elenco delle segnalazioni
- Chiudere le segnalazioni

Tecnologie Utilizzate



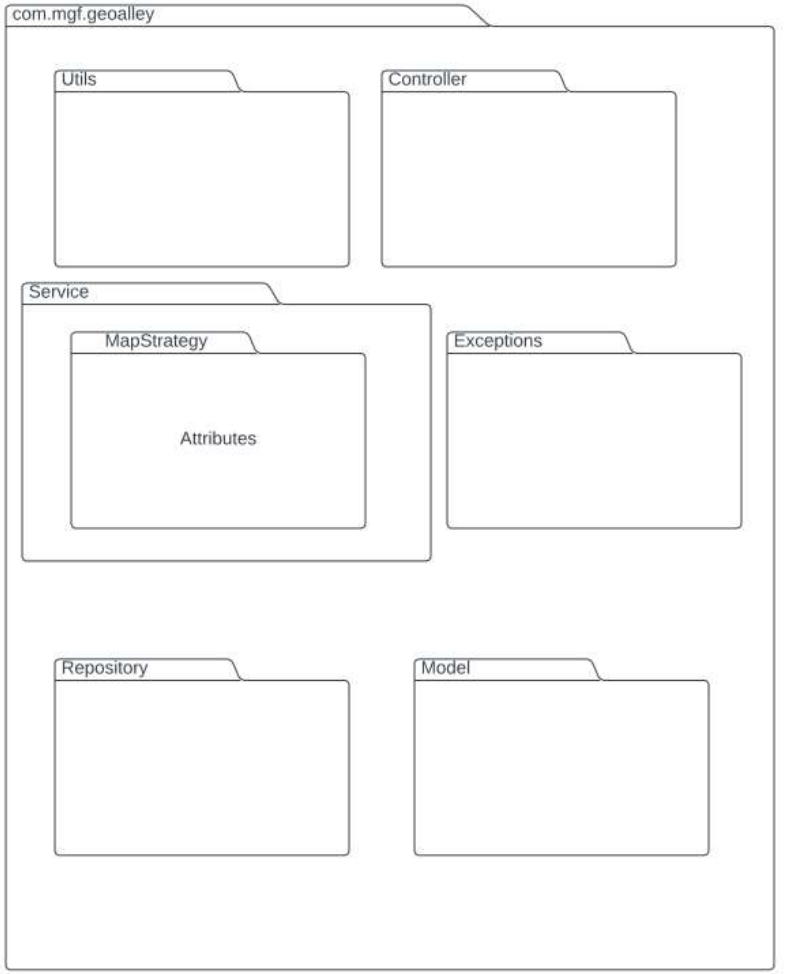
Architettura

Struttura classica di Spring Boot

- Controller
- Service
- Repository
- Model

Packages aggiuntivi:

- Utils
- Exceptions

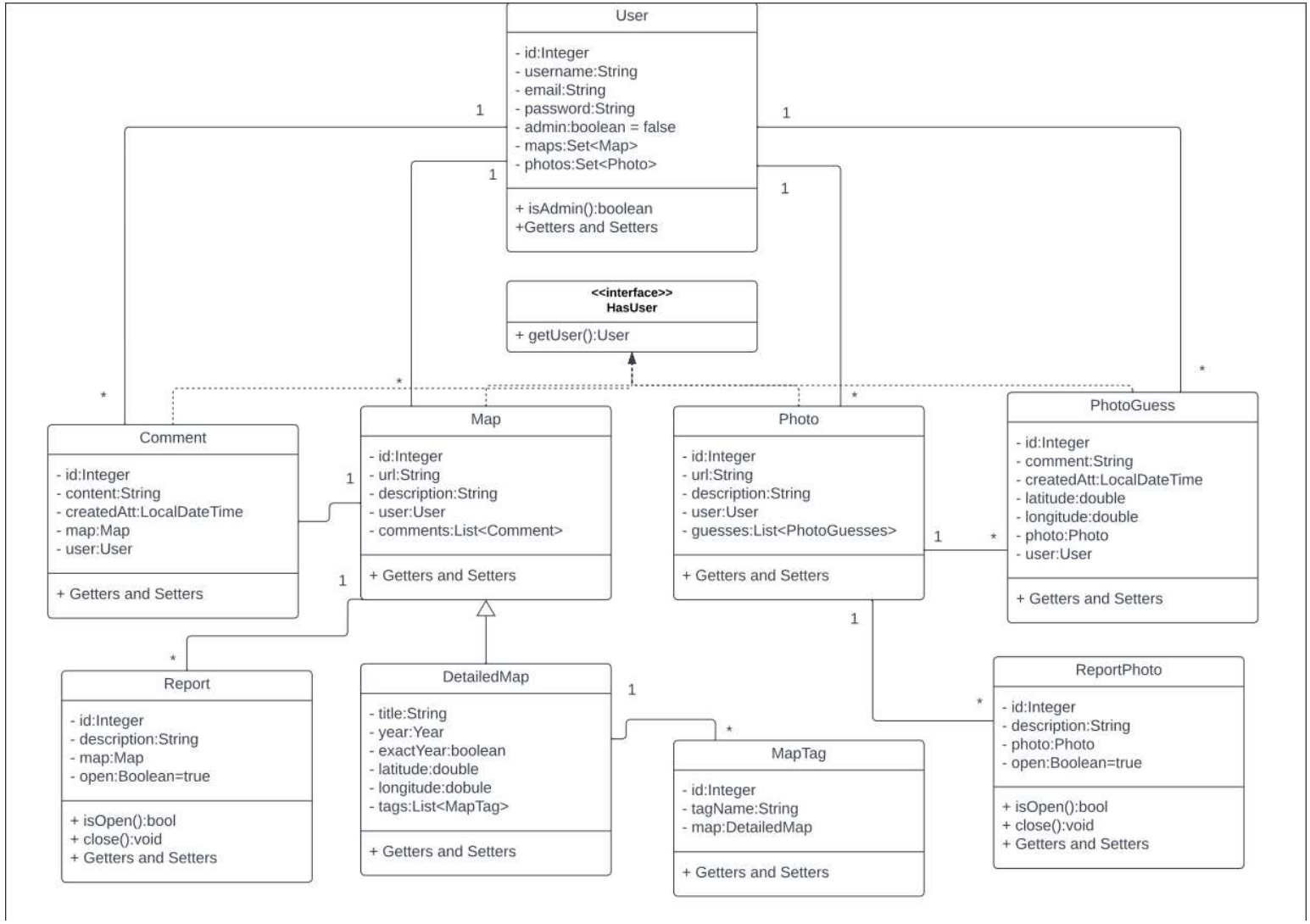


Utils

In questo package è contenuta una sola classe SecurityChecks, che offre dei metodi statici per i controlli di autenticazione utente.

```
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


User
- id:Integer
- username:String
- email:String
- password:String
- admin:boolean = false
- maps:Set<Map>
- photos:Set<Photo>
+ isAdmin():boolean
+Getters and Setters
<<interface>>
HasUser
+ getUser():User
Comment
- id:Integer
- content:String
- createdAtt:LocalDateTime
- map:Map
- user:User
+ Getters and Setters
Map
- id:Integer
- url:String
- description:String
- user:User
- comments:List<Comment>
+ Getters and Setters
Photo
- id:Integer
- url:String
- description:String
- user:User
- guesses:List<PhotoGuesses>
+ Getters and Setters
PhotoGuess
- id:Integer
- comment:String
- createdAtt:LocalDateTime
- latitude:double
- longitude:double
- photo:Photo
- user:User
+ Getters and Setters
Report
- id:Integer
- description:String
- map:Map
- open:Boolean=true
+ isOpen():bool
+ close():void
+ Getters and Setters
DetailedMap
- title:String
- year:Year
- exactYear:boolean
- latitude:double
- longitude:dobule
- tags:List<MapTag>
+ Getters and Setters
MapTag
- id:Integer
- tagName:String
- map:DetailedMap
+ Getters and Setters
ReportPhoto
- id:Integer
- description:String
- photo:Photo
- open:Boolean=true
+ isOpen():bool
+ close():void
+ Getters and Setters

Model

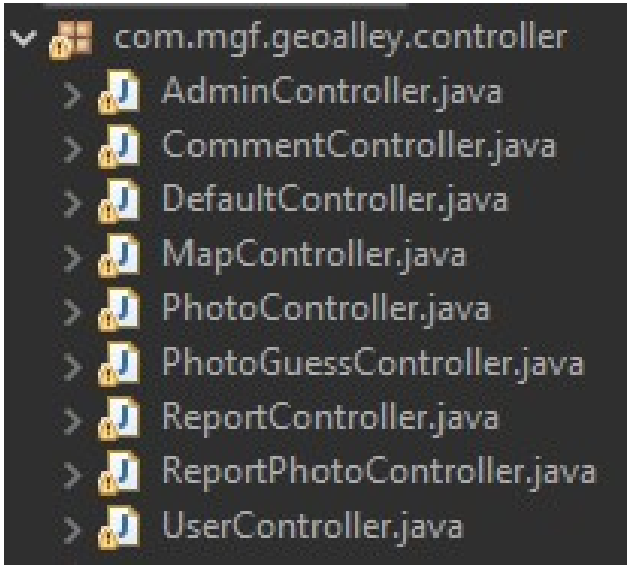
Ogni classe model è mappata nel database da JPA

```java
@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;
```

Controller

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

Nei controller avviene anche la maggior parte della gestione delle eccezioni, che vengono per lo più lanciate a livello Service

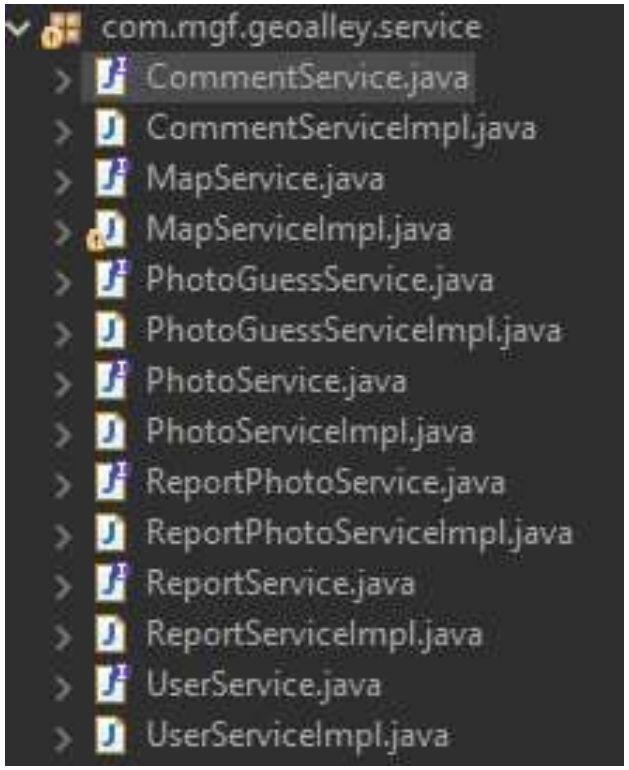



```
}catch(MapNotFoundException | UserUnauthorizedException e){
    redirectAttributes.addFlashAttribute("message", e.getMessage());
    return "redirect:/";
}catch (UserNotLoggedException e) {
    redirectAttributes.addFlashAttribute("message", "You must be logged to do that");
    return "redirect:/login";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

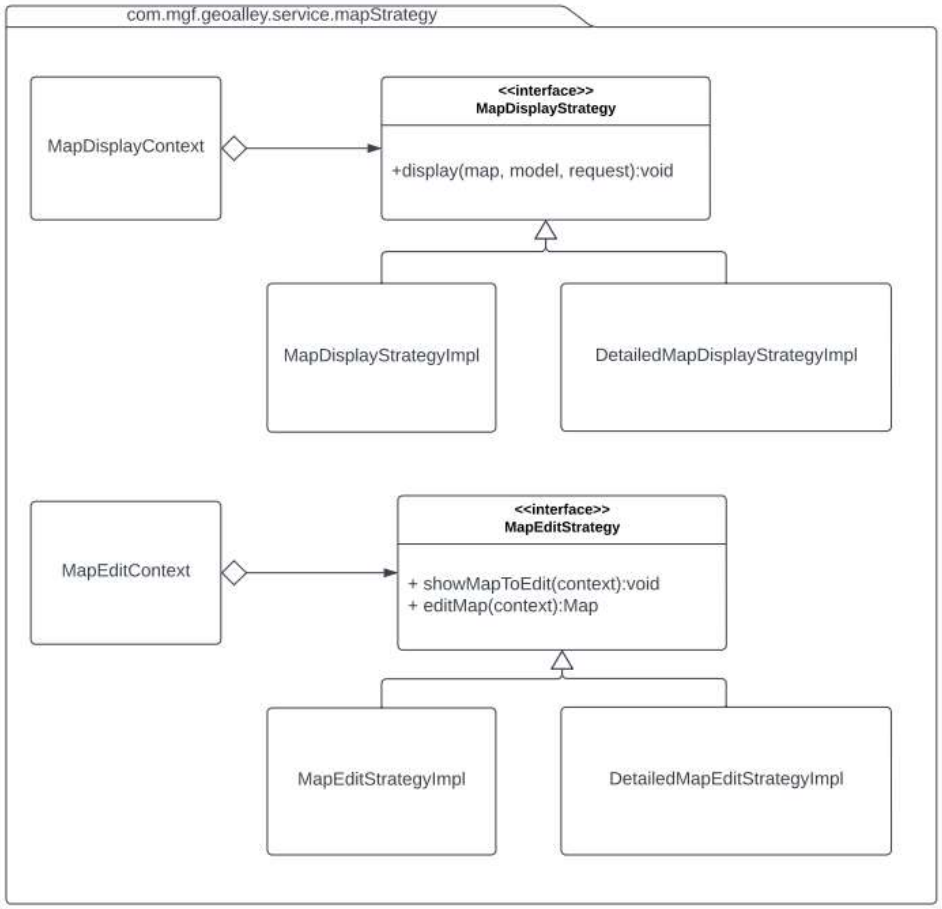
Service

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.

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.



mapStrategy

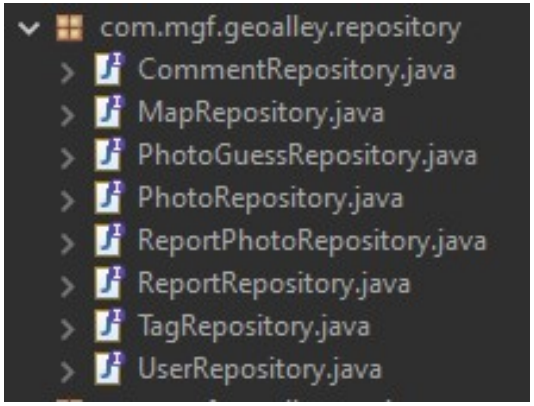


Il pattern strategy viene impiegato due volte:

- Per la logica di creazione della pagina in cui viene mostrata una mappa
- 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)

```
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 repository e tutte estendono JpaRepository.

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>

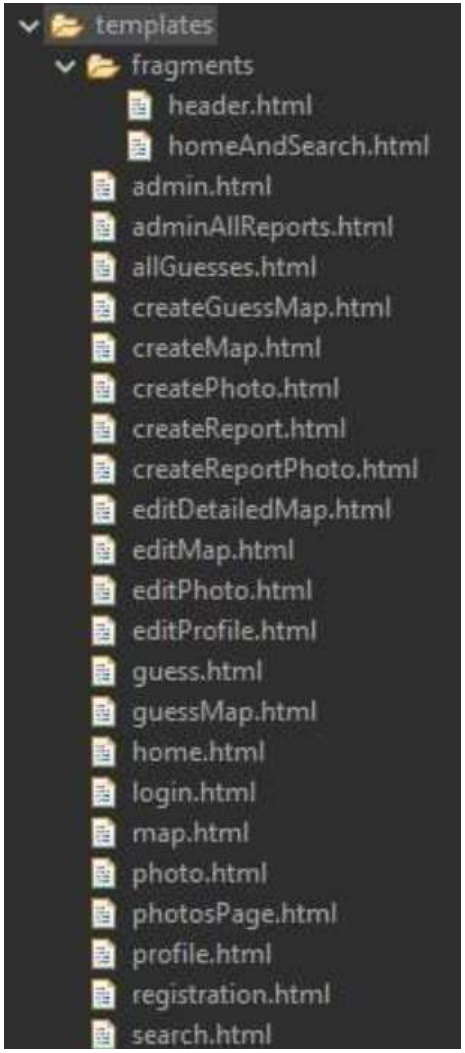
Nel Progetto si fa uso estensivo del pattern Optional, che permette di evitare il presentarsi di NullPointerException.

```
//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

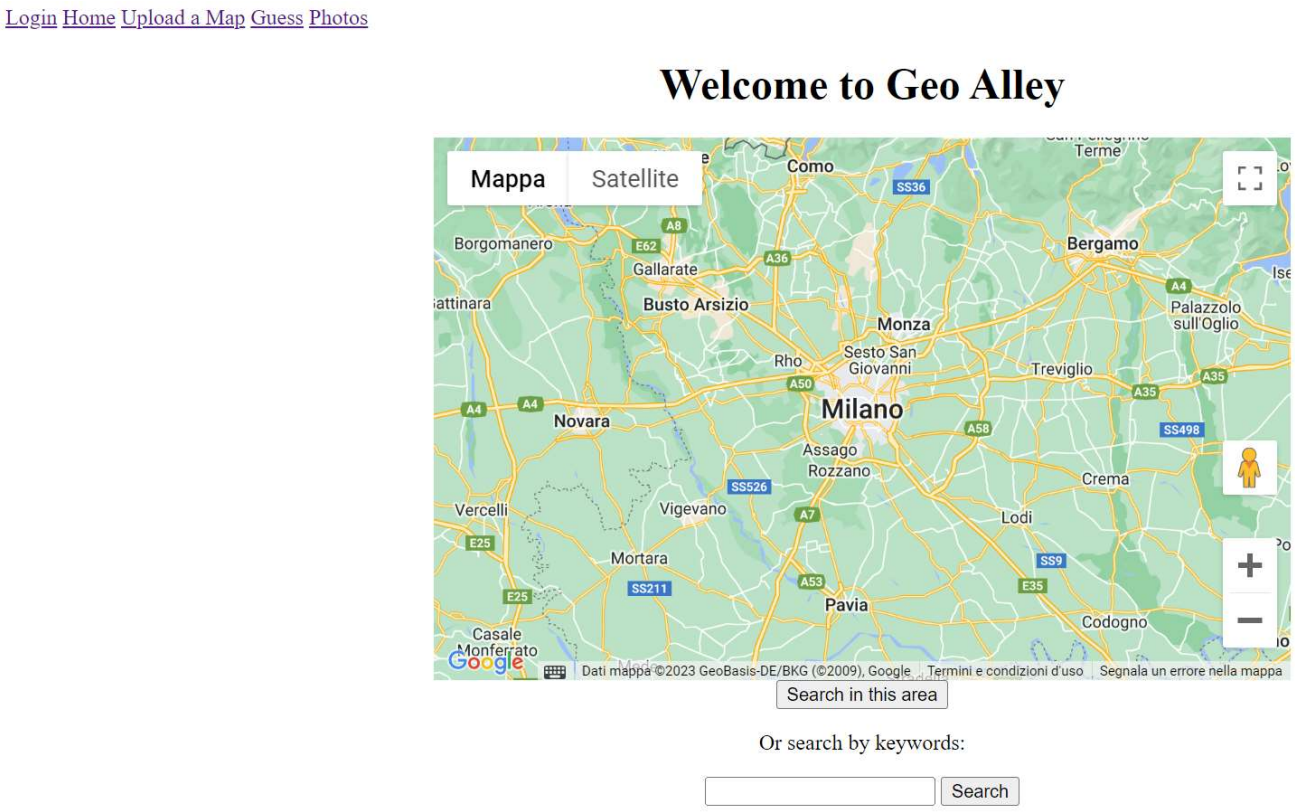
Il front end è sviluppato tramite template Thymeleaf e con l'utilizzo di qualche piccolo script JavaScript, soprattutto per l'interfacciamento con le API di Google Maps

```
//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";
```



Front End (Interfaccia Grafica)

L’interfaccia grafica è molto semplice e minimale perché sviluppata solo a scopo di esemplificare le funzionalità.



Testing

I test automatici svolti con Junit hanno riguardato le classi del package Service e del package Utils, raggiungendo una copertura sull'intero progetto del 30%

```java
@Test
public void testRemoveById_CommentRepositoryDeleteThrowsException() {

    final Optional<Comment> comment = Optional.of(new Comment(new User("username", "email", "password"),
            new Map("url", "description", new User("username", "email", "password")), "content"));
    when(mockCommentRepository.findById(0)).thenReturn(comment);

    doThrow(OptimisticLockingFailureException.class).when(mockCommentRepository).delete(any(Comment.class));

    assertThatThrownBy(() -> commentServiceImplUnderTest.removeById(0))
            .isInstanceOf(OptimisticLockingFailureException.class);
}

@Test
public void testFindById() throws Exception {

    final Optional<Comment> comment = Optional.of(new Comment(new User("username", "email", "password"),
            new Map("url", "description", new User("username", "email", "password")), "content"));
    when(mockCommentRepository.findById(0)).thenReturn(comment);

    final Comment result = commentServiceImplUnderTest.findById(0);

}

@Test
public void testFindById_CommentRepositoryReturnsAbsent() {

    when(mockCommentRepository.findById(0)).thenReturn(Optional.empty());

    assertThatThrownBy(() -> commentServiceImplUnderTest.findById(0)).isInstanceOf(CommentNotFoundException.class);
}
```