Progetto Questionari 1 - Ingegneria del Software UNIMIB 2021/2022

Davide Costantini, Gianlorenzo Martini, Khalil Mohamed Khalil, Lorenzo Occhipinti, Luca Milazzo

30/01/2022

Contents

1 Visione

1.1 Introduzione

Il progetto UNIMIB Modules è un ambiente all'interno del quale gestire e compilare i questionari, dotata di alta usabilità, tolleranza ai guasti e performance.

1.2 Posizionamento

1.2.1 Formulazione del problema

I prodotti già esistenti che propongono servizi di questo tipo spesso mancano della componente comunitaria. Per esempio, non sempre è possibile costruire un questionario a partire dalle domande di altri utenti o creare una vera e propria bacheca pubblica per i questionari stessi.

1.2.2 Parti interessate

I destinatari del sistema possono appartenere ad una qualsiasi categoria di utente che sia in grado di navigare nel web e questo definisce un ampio spettro di copertura.

2 Analisi e progettazione

2.1 Glossario

Glossario

ID	Termine	Definizione
1	Utente	Un qualsiasi utente che utilizza il sistema.
2	Utente registrato	Un utente che possiede un account.
3	Utente non registrato	Un utente che non possiede un account.
4	Servizio email	Il sottosistema che permette l'invio di email.
5	Domanda	Un elemento testuale, con eventuale immagine. Può essere: - Aperta - Chiusa con scelta singola - Chiusa con scelta multipla
6	Riposta	Un elemento testuale associato ad una domanda
7	Questionario	Insieme di domande a cui gli utenti possono rispondere

2.2 Casi d'uso

In questa sezione sono presentati gli attori del sistema ed i relativi casi d'uso. Per uno di essi è anche riportata la sua descrizione dettagliata.

Attori del sistema

ID	Nome	Tipo
1	Utente registrato	Primario
2	Utente non registrato	Primario
3	Servizio email	Di Supporto

Casi d'uso - Formato breve

ID	Nome	Attore	Descrizione
1	Effettua Login	Utente registrato	L'utente, dopo aver inserito le sue credenziali verificate dal sistema, effettua l'accesso all'applicazione.
2	Effettua Logout	Utente registrato	L'utente registrato effettua il logout dal sistema.
3	Creazione domanda	Utente registrato	L'utente registrato crea domande, testuali o contenenti immagini, con risposte chiuse o aperte ed il sistema le memorizza .
4	Ricerca domanda	Utente registrato	L'utente cerca le domande presenti nel sistema e le visualizza.
5	Creazione questionario	Utente registrato	L'utente registrato crea un questionario, poi memorizzato dal sistema, partendo da domande già create.
6	Modifica domanda	Utente registrato	L'utente registrato modifica una domanda che ha creato.
7	Cancellazione domanda	Utente registrato	L'utente registrato cancella la domanda che ha creato.
8	Modifica questionario	Utente registrato	L'utente registrato modifica un questionario che ha creato.
9	Cancellazione questionario	Utente registrato	L'utente registrato cancella i questionari che ha creato.
10	Modifica risposta	Utente registrato	L'utente modifica le sue risposte ai questionari.
11	Cancellazione risposta	Utente registrato	L'utente elimina le sue risposte ai questionari.
12	Compilazione questionario	Utente registrato	L'utente compila i questionari inserendo delle risposte.
13	Notifica del completamento di un questionario	Servizio email	Il sistema invia una email all'utente in cui lo avvisa del completamento di un questionario con un PDF delle risposte date.
14	Ricerca di un questionario	Utente registrato Utente non registrato	L'utente può cercare un questionario tra quelli presenti nel sistema in base a un codice, a una parola presente nel questionario, ecc...
15	Effettua registrazione	Utente non registrato	L'utente effettua la registrazione nel sistema.

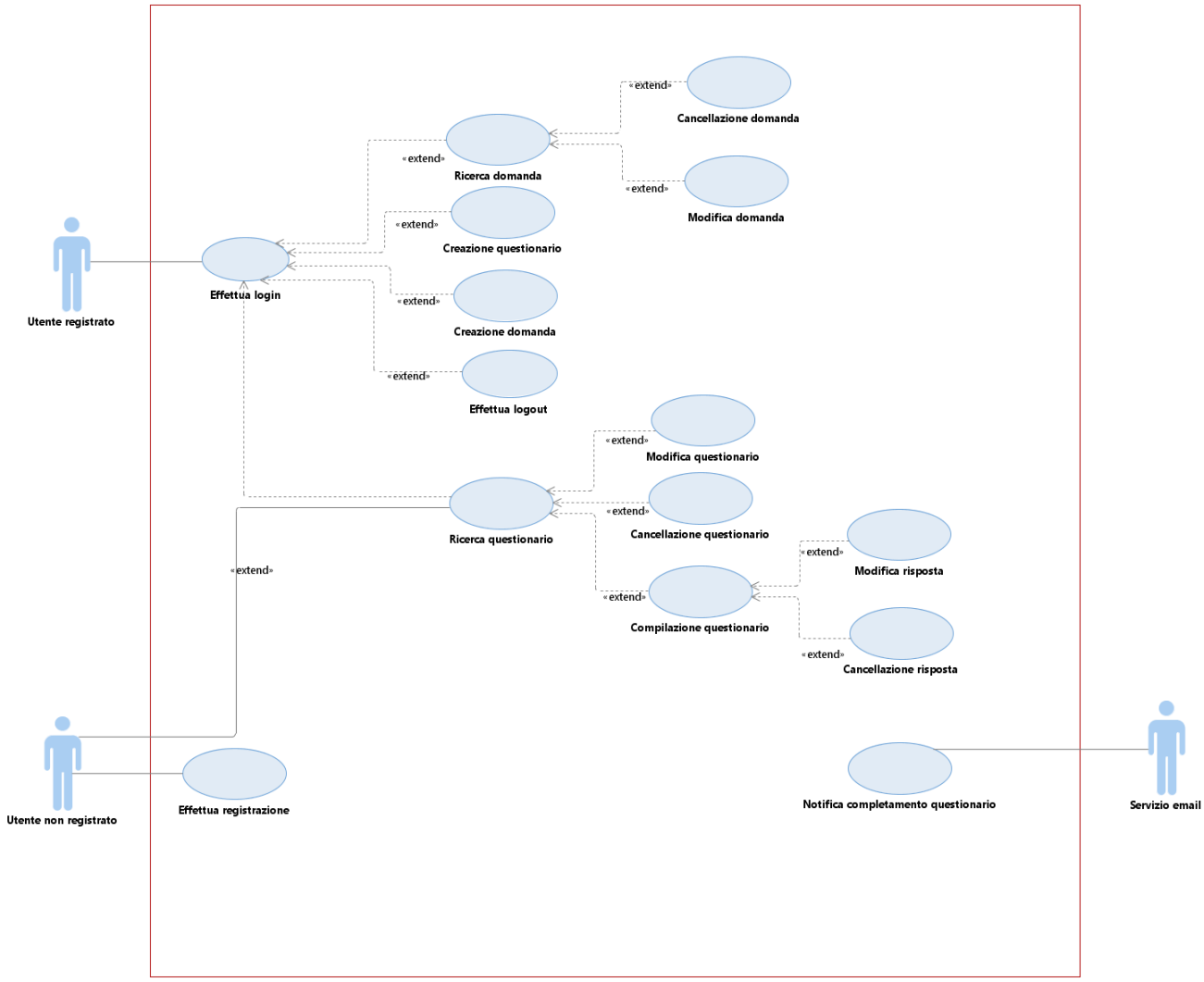

Fig. 1: Diagramma dei casi d'uso

2.2.1 Compila questionario come utente guest

- Nome: Compila questionario come utente guest
- Portata: Applicazione UNIMIBModules
- Livello: Obiettivo utente
- Attore primario: Utente non registrato

- Parti interessate:
  - Utente non registrato: Vuole compilare un questionario senza il bisogno di creare un account nel sistema. Vuole un codice utilizzabile per poter reprendere a compilare lo stesso questionario in un qualsiasi momento.
- Pre-condizioni: L'utente non ha effettuato l'accesso al sistema e si trova nella pagina Home.
- Garanzia di successo: La compilazione del questionario viene salvata nel sistema. L'utente ottiene un codice da utilizzare nella pagina Home per poter riprendere la stessa compilazione.
- Scenario principale di successo:
  - Il sistema carica i questionari
  - L'utente non registrato tramite l'interfaccia grafica della HomePage preme il pulsante "Compile" da uno dei questionari caricati.
  - Il sistema mostra all'utente non registrato un modale.
  - Il sistema restituisce all'utente non registrato il codice univoco del questionario che si vuole compilare.
  - L'utente non registrato inserisce l'email nell'apposito campo di testo che viene fornito nel modale
  - L'utente non registrato preme "Continue as Guest"
  - Il sistema registra l'email e il codice univoco in una nuova utenza nel database
  - Il sistema mostra all'utente non registrato la pagina di compilazione del questionario
  - All'avvenuta conferma della compilazione, il sistema invia una notifica di completamento all'email salvata precedentemente
  - Il sistema permette all'utente non registrato di poter scaricare il pdf riepilogativo del questionario compilato
  - Il sistema riporta l'utente non registrato alla HomePage
- Requestiti speciali: Il sistema deve essere completamente funzionante.
- Frequenza di ripetizione: Media

2.3 Requisiti non funzionali

ID	Descrizione	Tipo	Misura
1	Il sistema deve essere sempre raggiungibile	Di prodotto	Disponibilità
2	Il sistema deve essere in grado di gestire molti utenti contemporaneamente	Di prodotto	Efficienza
3	Il sistema deve garantire la persistenza dei dati	Di prodotto	Affidabilità
4	Il sistema deve garantire la sicurezza dei dati degli utenti	Di prodotto	Sicurezza
5	Il sistema deve garantire brevissime attese agli utenti per l'elaborazione di richieste	Di prodotto	Efficienza

2.4 SSD

Qui di seguito sono presenti gli SSD riguardanti i seguenti scenari:

- Creazione del questionario

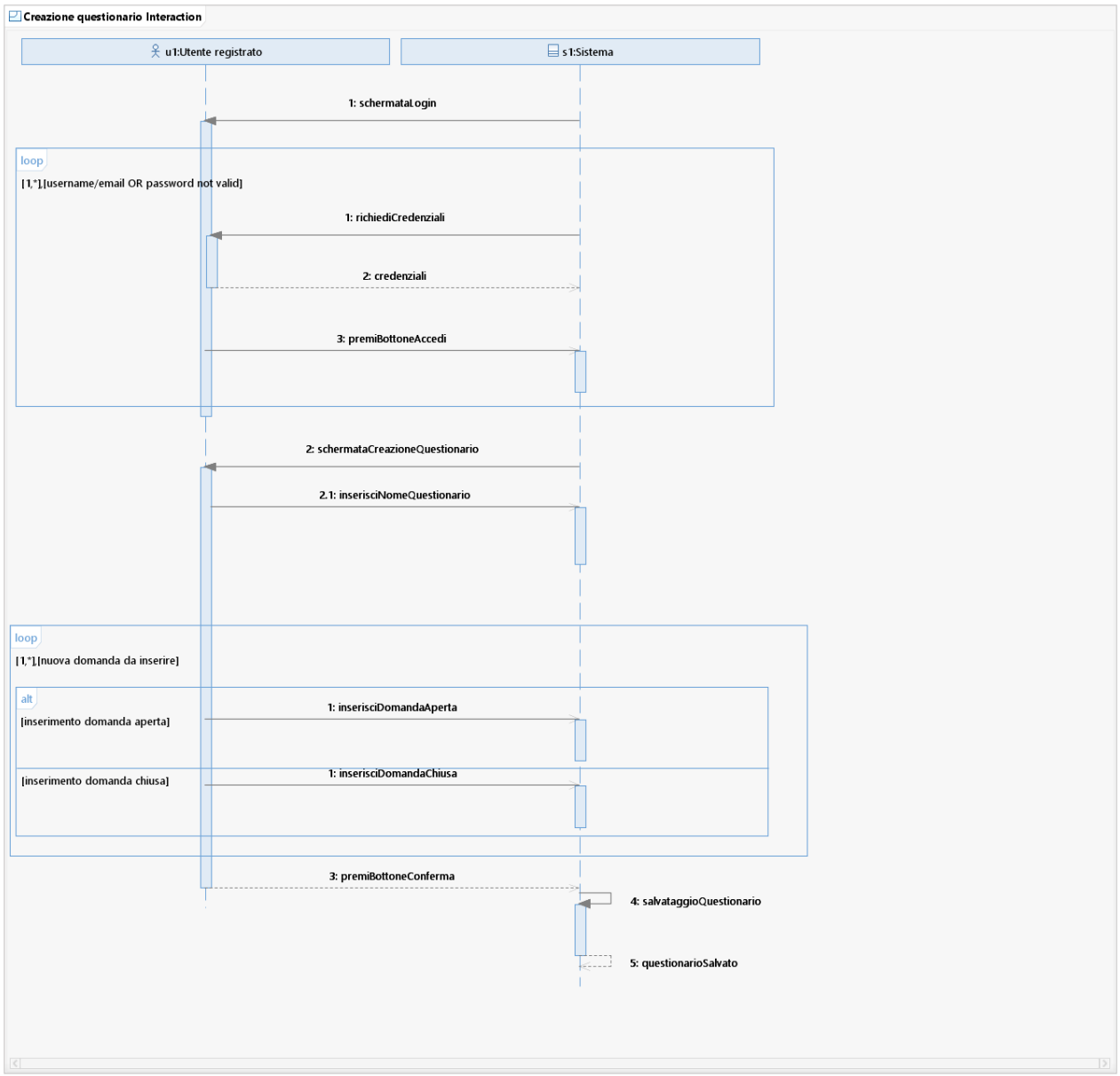

Fig. 2: SSD - Creazione del questionario

2.5 Modello di dominio

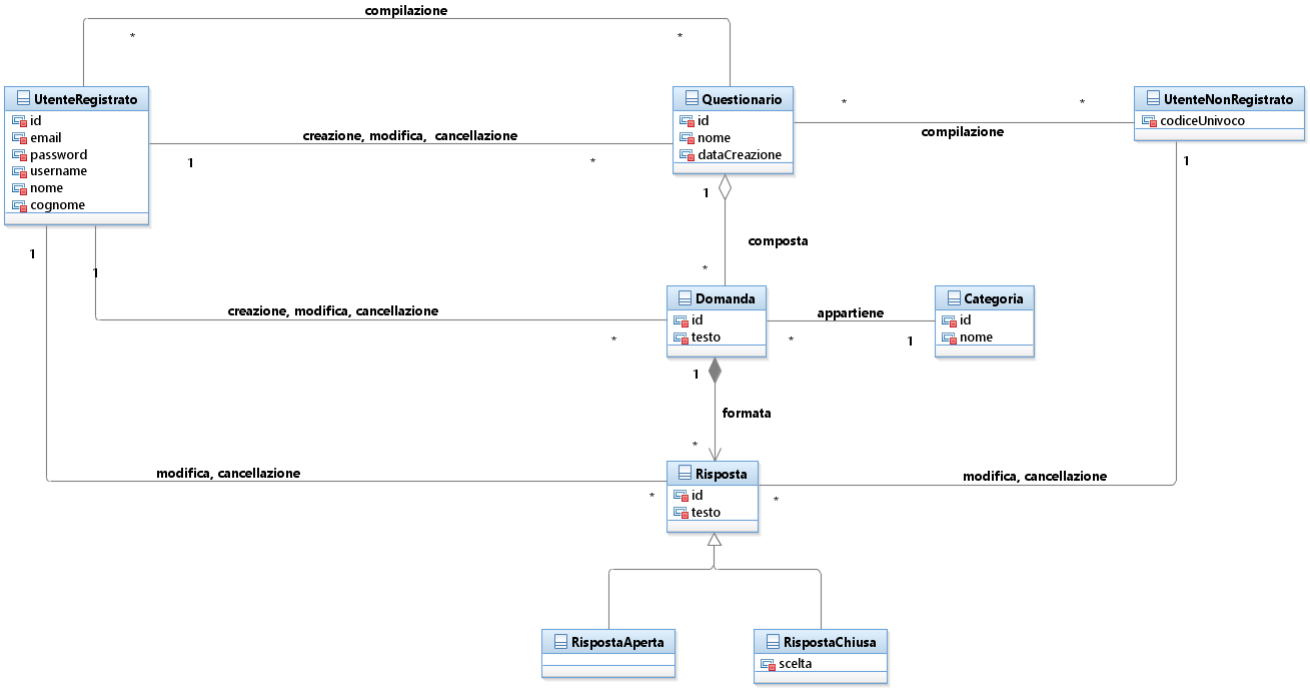

Fig. 3: Modello di dominio

2.6 Diagramma delle classi di progettazione

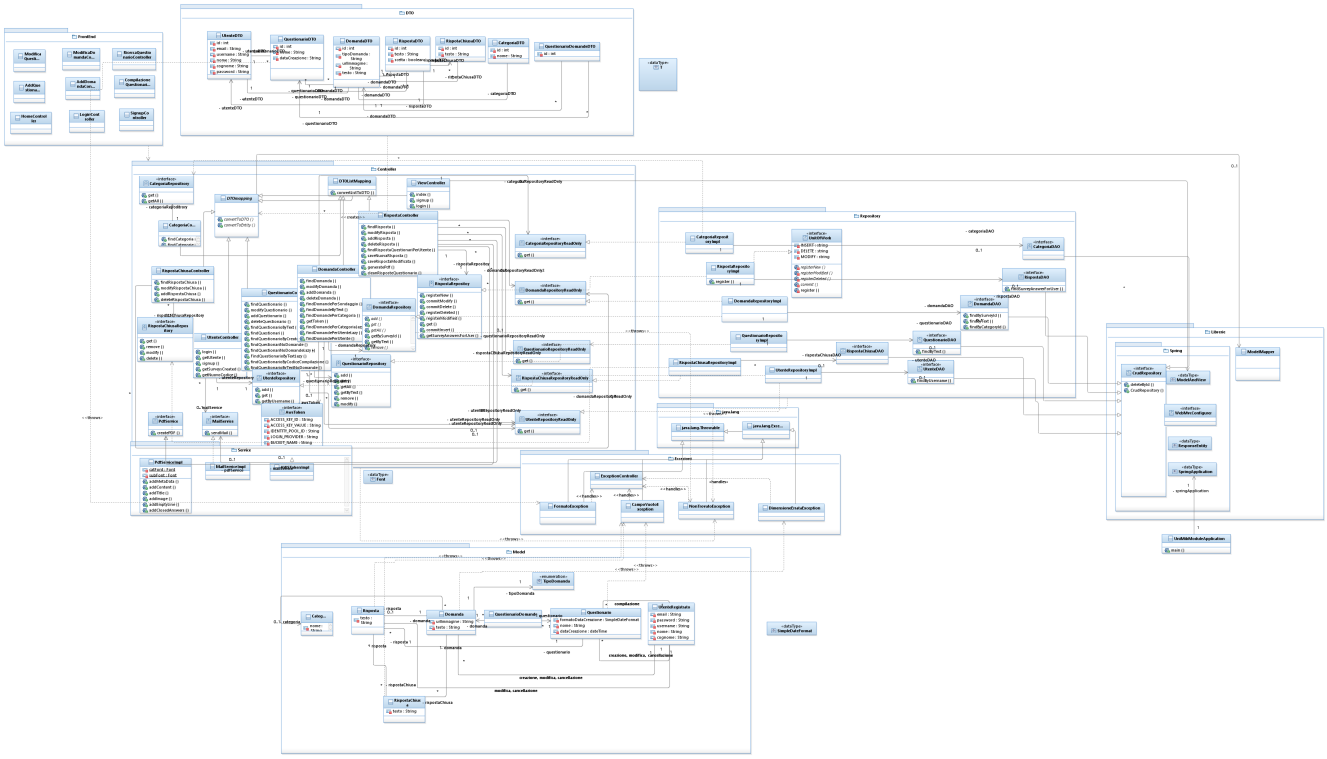

Fig. 4: Diagramma delle classi di progettazione

2.7 Diagrammi di sequenza

2.7.1 aggiugiDomanda

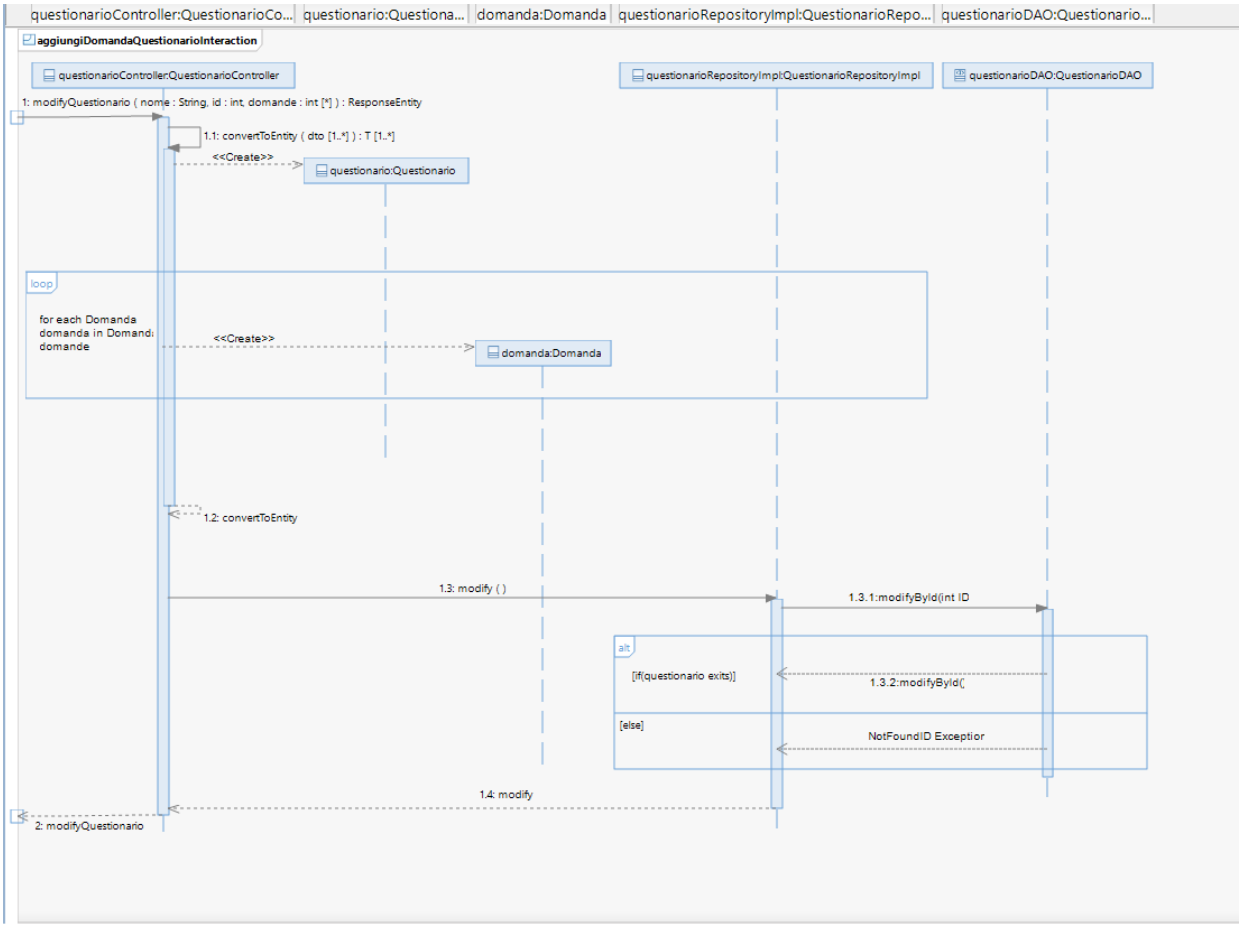

Fig. 5: Diagramma di sequenza aggiungiDomanda

2.7.2 eliminaDomanda

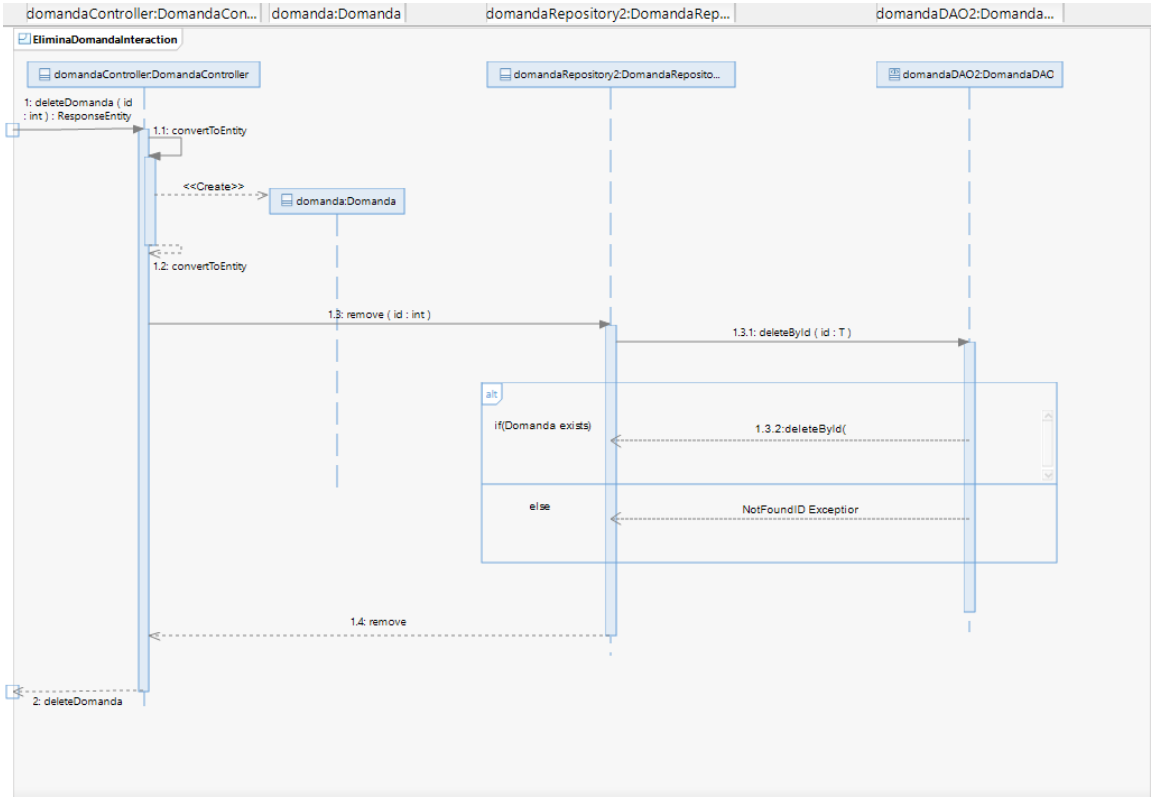

Fig. 6: Diagramma di sequenza eliminaDomanda

2.8 Diagrammi di stato

2.8.1 StateMachine Creazione Questionario

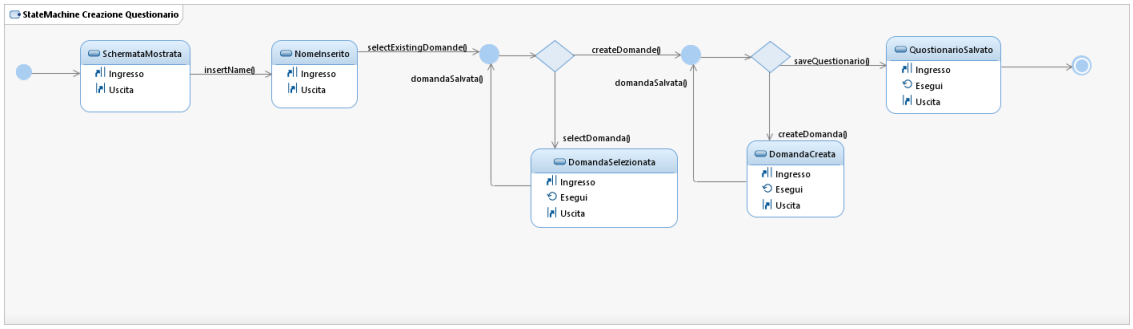

Fig. 7: Diagrammi di stato creazioneQuestionario

2.9 Diagrammi di attività

2.9.1 Creazione Questionario

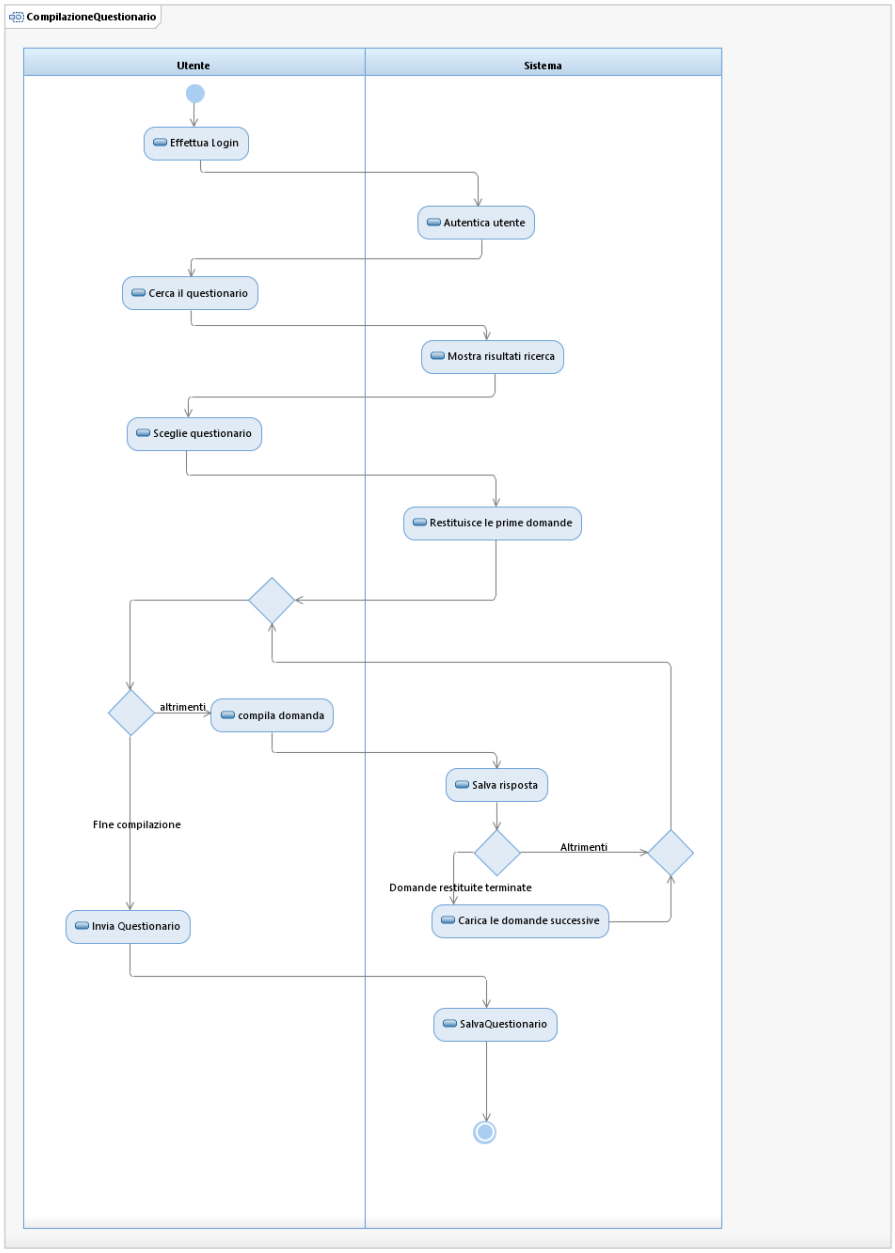

Fig. 8: Diagramma di attività creazioneQuestionario

2.10 Modello E-R

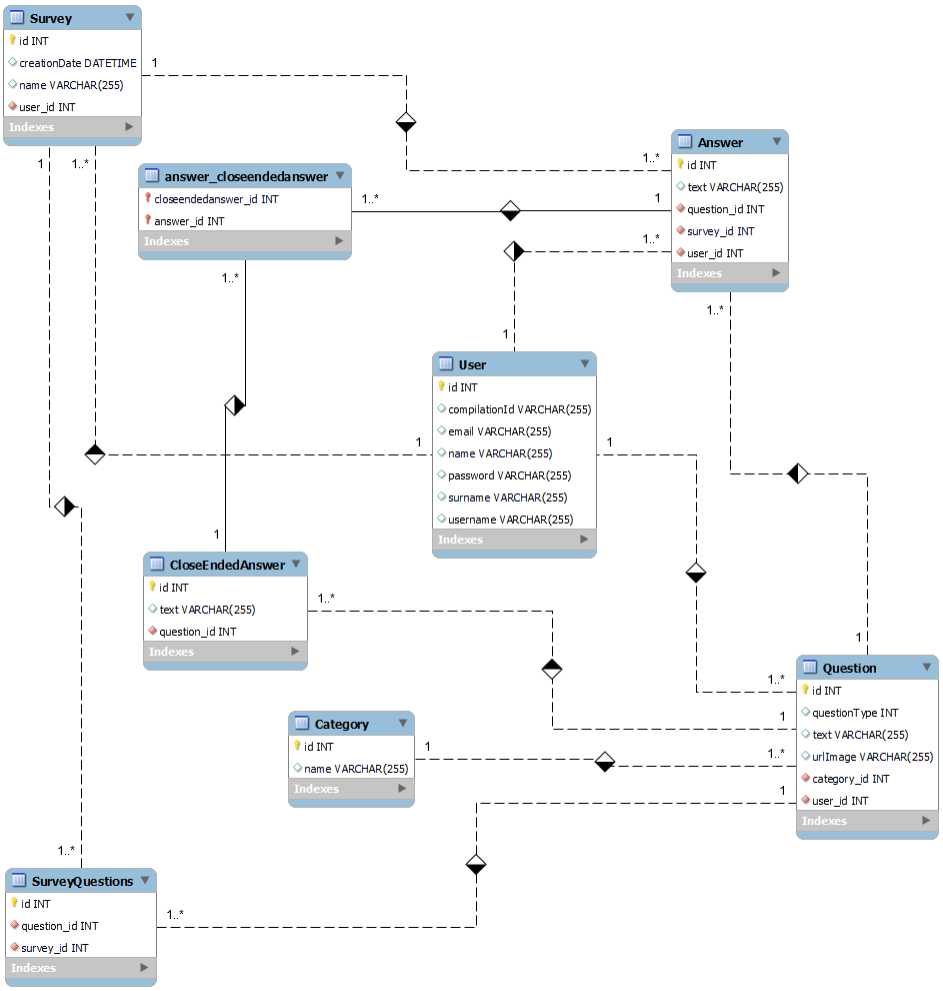

Fig. 9: Modello E-R

2.11 Diagramma dell'architettura software

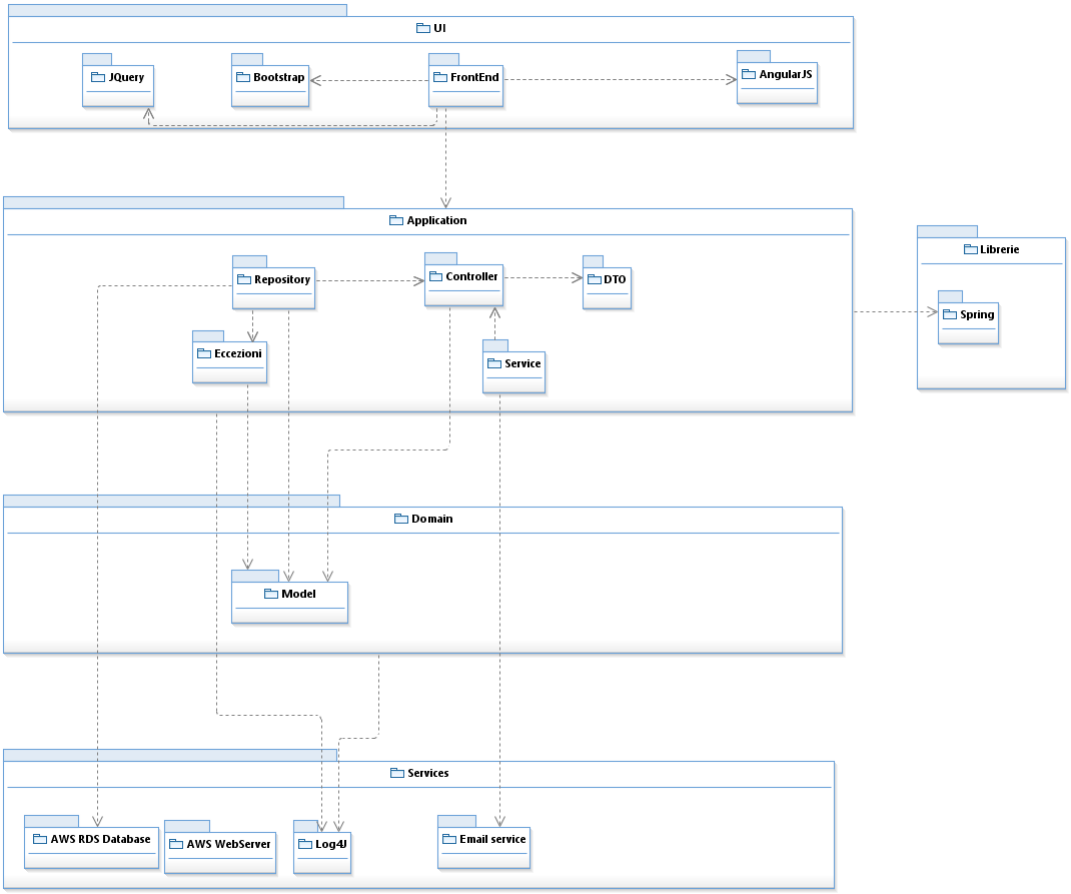

Fig. 10: Diagramma dell'architettura software

2.12 Design Principles

I Design Principles utilizzati durante la creazione del progetto.

- **Principio reuse/release equivalency**: L'unità di riutilizzo è l'unità di rilascio. I componenti rilasciati vengono seguiti da un sistema di tracking attraverso le versioni.
- **Principio di separazione degli interessi**: Ogni sezione del programma si occupa del suo interesse senza occuparsi di interessi non suoi.
- **Principio delle dipendenze acicliche**: Il grafo delle dipendenze di pacchetti non presenta cicli.

2.12.1 Principi SOLID

- **Principio di singola responsabilità**: Ogni elemento del programma ha una sola responsabilità.
- **Principio aperto/chiuso**: Ogni modulo è aperto ad estensioni ma chiuso a modifiche.
- **Principio di sostituzione di Liskov**: Gli oggetti di un sottotipo di un oggetto possono essere sostituiti dall'oggetto di cui sono sottotipo senza alterare la correttezza del programma.
- **Principio di segregazione delle interfacce**: Il client utilizza interfacce piccole e specifiche ma numerose per evitare dipendenza da metodi non utilizzati.
- **Principio di inversione delle dipendenze**: I moduli di alto e basso livello non dipendono tra di loro ma dipendono da astrazioni.

2.12.2 Principi PHAME

- **Principio di astrazione**: Alcune entità sono state generalizzate identificando caratteristiche importanti e comuni e ridotte eliminando dettagli non necessari
- **Principio di modularizzazione**: Il progetto è stato diviso in moduli che si occupano ciascuno della propria parte di processo
- **Principio di incapsulamento**: Sono stati nascosti i dettagli dell'implementazione e incapsulate le parti che potrebbero variare

2.13 Design Patterns

Design Patterns utilizzati durante la creazione del progetto.

2.13.1 Unit of Work

- **Nome**: Unit of Work
- **Classificazione**: Object-relational behavioural pattern
- **Applicabilità**: Quando l'utente sta compilando un questionario, le risposte non vengono inserite/modificate/eliminate appena l'utente conferma; vengono inserite nella Unit Of Work e vengono poi salvate quando l'utente termina la compilazione.
- **Partecipanti**:
  - UnitOfWork
  - RispostaRepositoryImpl
- **Scopo**: Tenere traccia di ciò che possa modificare il database durante una transazione business e quando è terminata si occupa di applicare tutte le modifiche.
- **Codice d'esempio**

```
/**
* The context of the UnitOfWork
*/
private final Map<String, List<Answer>> uofContext;

/**
* Registers <code>answer</code> on the specified
   <code>operation</code>.
* @param answer    the answer to be registered
* @param operation the operation to be performed on answer
*/
private void register(Answer answer, String operation) {

   List<Answer> answerToOperate =
      uofContext.computeIfAbsent(operation, k -> new
      ArrayList<>());
   answerToOperate.add(answer);
}
```

```
/**
* Adds <code>answer</code> to the elements to be inserted.
* @param answer the new Answer
* @see UnitOfWork#registerNew
*/
@Override
public void registerNew(Answer answer) {

   register(answer, UnitOfWork.INSERT);
}

/**
* Inserts the registered answers made by the user identified
    by <code>userId</code> on the survey identified by
* <code>surveyId</code>.
* @param   surveyId   the id of the survey
* @param   userId      the id of the user
*/
@Override
public void commitInsert(int surveyId, int userId) {

   if (uofContext.size() == 0 ||
       !uofContext.containsKey(UnitOfWorkOperations.INSERT.getValue()))
       {
      return;
   }

   List<Answer> answerList =
       uofContext.get(UnitOfWorkOperations.INSERT.getValue());
   answerList.stream()
   .filter(answer -> answer.getSurvey().getId() == surveyId &&
       answer.getUser().getId() == userId)
   .collect(Collectors.toList())
   .forEach(answer -> {
      add(answer);
      answerList.remove(answer);
   });
}
```

2.13.2 Data Mapper

- **Nome**: Data Mapper
- **Classificazione**: Data-source Architectural pattern
- **Applicabilità**: Si effettua il mapping tra le classi del modello e le tabelle del database
- **Partecipanti**:
  - CategoriaDAO
  - Categoria
  - RispostaDAO
  - Risposta
  - DomandaDAO
  - Domanda
  - QuestionarioDAO
  - Questionario
  - UtenteDAO
  - Utente
  - RispostaChiusaDAO
  - RispostaChiusa
- **Scopo**: Creare una relazione tra i dati del database e del dominio

2.13.3 Page Controller

- **Nome**: Page Controller
- **Classificazione**: Web presentation pattern
- **Applicabilità**: Si crea un oggetto che si occupa di gestire le richieste specifiche per ogni pagina del sito web.
- **Partecipanti**:
  - add-question.component

  - add-survey.component
  - compile-survey.component
  - edit-question.component
  - get-user.component
  - home.component
  - login-user.component
  - modify-survey.component
  - signup-user.component

- **Scopo**: Fare da tramite tra il back-end e la pagina web.

2.13.4 Lazy Loading

- **Nome**: Lazy Loading
- **Classificazione**: Object-relational behavioural pattern
- **Applicabilità**: Quando si desidera caricare tutti i questionari o tutte le domande di un questionario, queste non verranno mostrate tutte contemporaneamente ma si applicherà il lazy loading. Attraverso un offset si recupereranno un numero definito di domande alla volta.
- **Partecipanti**:
  - QuestionarioController
  - QuestionarioRepositoryImpl
  - DomandaController
  - DomandaRepositoryImpl
- **Scopo**: Rinvia l'inizializzazione di un oggeto fino a quando non è necessario
- **Codice d'esempio**

```
/**
* Finds all surveys without their questions.
*
* @return an HTTP response with status 200 if one survey
    exists at least.
```

```
* @throws NotFoundException
*/
@GetMapping("/findAllSurveysNoQuestionLazy")
public ResponseEntity<List<SurveyDTO>>
    findAllSurveysNoQuestionLazy(@RequestParam int offset,
@RequestParam int limit) throws NotFoundException {

   Iterable<Survey> surveys =
       surveyRepository.getAllLazy(offset, limit);
   List<SurveyDTO> surveysDTO = new ArrayList<>();
   for (Survey survey : surveys) {
      surveysDTO.add(convertToDTOAndSkipQuestions(survey));
   }
   return new ResponseEntity<>(surveysDTO, HttpStatus.OK);
}

/**
* Returns all surveys in the database with Lazy loading.
* @param offset
* @param limit
* @return a Set of Surveys
* @throws NotFoundException
*/
@Override
public Iterable<Survey> getAllLazy(int offset, int limit)
    throws NotFoundException {
   Iterable<Survey> surveys = surveyDAO.findAllLazy(offset,
       limit);
   if (IterableUtils.size(surveys) > 0) {
      return surveys;
   } else {
      throw new NotFoundException("No surveys exist.");
   }
}
```

2.13.5 Data Transfer Object

- **Nome**: Data Transfer Object
- **Classificazione**: Data
- **Applicabilità**: Quando si vogliono inviare dati al client o si ricevono dati dal client, questi vengono serializzati e trasmessi usando il corrispondente DTO.
- **Partecipanti**:
  - DTOMapping
  - DTOListMapping
  - UtenteDTO
  - UtenteController
  - QuestionarioDTO
  - QuestionarioController
  - DomandaDTO
  - Domanda Controller
  - RispostaDTO
  - RispostaController
  - RispostaChiusaDTO
  - RispostaChiusaController
  - CategoriaDTO
  - QuestionarioDomandeDTO
- **Scopo**: Serializzare e trasmettere i dati tra il client e il server per ridurre il numero di chiamate ai metodi
- **Codice d'esempio**

```
public class QuestionDTO {

  /**
  * Serialization of the id of the question.
  */
  @Getter private int id;
```

```
    /**
    * Serialization of the category of the question.
    */
    @Getter @Setter private CategoryDTO category;

    /**
    * Serialization of the image's url of the question.
    */
    @Getter @Setter private String urlImage;

    /**
    * Serialization of the text of the question.
    */
    @Getter @Setter private String text;

    /**
    * Modifies the id of the question, setting <code>id</code>
        as the new value.
    * @param id the new id value
    */
    public void setId(int id) {

       this.id = id;
    }

    /**
    * Modifies the id of the question, setting <code>id</code>
        as the new value.
    * @param id the new id value
    */
    public void setId(Object id) {

       this.id = (int) id;
    }
}

public class QuestionController extends
    DTOListMapping<Question, QuestionDTO>{

    public QuestionController(QuestionRepository
```

```
        questionRepository) {

        super(modelMapper);

        modelMapper.createTypeMap(Question.class,
            QuestionDTO.class)
        .addMappings(mapper -> {
            mapper.map(Question::getId, QuestionDTO::setId);
            mapper.map(Question::getUrlImage,
                QuestionDTO::setUrlImage);
            mapper.map(Question::getText, QuestionDTO::setText);
            mapper.map(Question::getCategory,
                QuestionDTO::setCategory);
        });

        modelMapper.createTypeMap(QuestionDTO.class,
            Question.class)
        .addMappings(mapper -> {
            mapper.map(QuestionDTO::getId, Question::setId);
            mapper.map(QuestionDTO::getUrlImage,
                Question::setUrlImage);
            mapper.map(QuestionDTO::getText, Question::setText);
            mapper.map(QuestionDTO::getQuestionType,
                Question::setQuestionType);
        });
    }
    ...
}
```

2.13.6 Data Access Object

- **Nome**: Data Access Object
- **Classificazione**: Data
- **Applicabilità**: Quando si vogliono effettuare operazioni su entità di una o più tabelle, lo si può fare attraverso i DAO, dove oltre le operazioni base sono state aggiunte in alcuni casi operazioni custom per interagire con il database.
- **Partecipanti**:
  - CategoriaDAO
  - CategoriaRepositoryImpl
  - RispostaDAO
  - RispostaRepositoryImpl
  - DomandaDAO
  - DomandaRepositoryImpl
  - QuestionarioDAO
  - QuestionarioRepositoryImpl
  - UtenteDAO
  - UtenteRepositoryImpl
  - RispostaChiusaDAO
  - RispostaChiusaRepositoryImpl
- **Scopo**: Rappresentare un'entità di una tabella di un database usato per stratificare e isolare l'accesso ad una tabella
- **Codice d'esempio**

```
public interface AnswerDAO extends CrudRepository<Answer,
    Integer> {

   @Query("SELECT a FROM Answer a WHERE a.survey.id = :surveyId
       AND a.user.id = :userId")
   Iterable<Answer> findSurveyAnswersForUser(@Param("surveyId")
       int surveyId, @Param("userId") int userId);
}
```

```
public class AnswerRepositoryImpl implements AnswerRepository,
    UnitOfWork<Answer> {

   /**
   * The instance of AnswerDAO that will be used to perform
       actions to the DB
   */
   private final AnswerDAO answerDAO;

   public Iterable<Answer> getSurveyAnswersForUser(int
       surveyId, int userId) {

      return answerDAO.findSurveyAnswersForUser(surveyId,
          userId);
   }
}
```

2.13.7 Repository

- **Nome**: Repository
- **Classificazione**: Data
- **Applicabilità**: I Repository permettono di fare da intermediario tra l'accesso ai dati che sia controller o DAO con il resto dell'applicazione
- **Partecipanti**:
  - RispostaRepositoryImpl
  - RispostaControllerImpl
  - UtenteRepositoryImpl
  - UtenteControllerImpl
  - DomandaRepositoryImpl
  - DomandaControllerImpl
  - QuestionarioRepositoryImpl
  - QuestionarioControllerImpl

- **Scopo**: Mediare tra la logica di accesso ai dati e il resto dell'applicazione
- **Codice d'esempio**

```
public interface CategoryRepository {

   /**
   * Finds the category identified by id in the database
   * @param  id  the id of the category to be found
   * @return     an instance of Category if there is a category
       identified by id, null otherwise
   */
   Category get(int id) throws NotFoundException;

   /**
   * Finds all the categories in the database
   * @return     all the instances of Category, null otherwise
   */
   Iterable<Category> getAll() throws NotFoundException;
}

/**
* Repository for the Category class. Adds business logic to
    Category instances before
* accessing the database via DAO.
* @author Lorenzo Occhipinti
* @version 0.4.1
*/
@Component("categoryRepository")
public class CategoryRepositoryImpl implements
    CategoryRepository, CategoryRepositoryReadOnly {

   /**
   * The instance of categoryDAO that will be used to perform
       actions to the DB
   */
   private final CategoryDAO categoryDAO;

   @Autowired
   public CategoryRepositoryImpl(CategoryDAO categoryDAO) {
      this.categoryDAO = categoryDAO;
```

```
}

/**
* Finds the category identified by id in the database
* @param  id  the id of the category to be found
* @return     an instance of Category if there is a category
    identified by id, null otherwise
* @see CategoryRepository#get(int id)
*/
@Override
public Category get(int id) throws NotFoundException {
   Optional<Category> category = categoryDAO.findById(id);
   try {
      return category.orElseThrow();
   }catch (NoSuchElementException ex){
      throw new NotFoundException("The category with id =
          "+id+" has not been found.");
   }
}

public class CategoryController extends
    DTOListMapping<Category, CategoryDTO>{

   /**
   * Instance of CategoryRepository that will be used to
       access the db.
   */
   private final CategoryRepository categoryRepository;


   @Autowired
   public CategoryController(CategoryRepository
       categoryRepository) {

      super(modelMapper);
      this.categoryRepository = categoryRepository;
   }

   /**
   * Gets the Category associated with the given id.
   * @param id the id of the category
```

```
     * @return     an HTTP response with status 200, 500
         otherwise
     * @throws NotFoundException
     */
     @GetMapping(path = "/findCategory/{id}")
     public ResponseEntity<CategoryDTO>
         findCategory(@PathVariable int id) throws
         NotFoundException {
        Category category = categoryRepository.get(id);
        return new ResponseEntity<>(convertToDTO(category),
            HttpStatus.OK);
     }
}
```

2.13.8 Façade

- **Nome**: Façade
- **Classificazione**: Data
- **Applicabilità**: Façade permette l'accesso ai vari controller Spring
- **Scopo**: Rappresenta il sistema complessivo, un oggetto radice, un dispositivo all'interno del quale viene eseguito il software, un punto di accesso al software o un sottosistema principale.

2.13.9 Valet Key

- **Nome**: Valet Key
- **Classificazione**: Security
- **Applicabilità**: Ogni domanda può contenere un'immagine e, per motivi di performance, è stato affidato al dispositivo client il compito di ottenere o inviare le immagini al service storage. Il valet key si occupa della sicurezza durante l'interazione tra il client e il data storage, ulteriori informazioni qui.
- **Partecipanti**:
  - AWSToken
  - AWSTokenImpl

– QuestionController

- **Scopo**: Gestire l'accesso a risorse protette da parte degli endpoint client, autenticati nel sistema, fornendogli il valet key.
- **Codice d'esempio**

```
public interface AWSToken {
   Region REGION = Region.getRegion(Regions.EU_CENTRAL_1);
   String ACCESS_KEY_ID = "ACCESS_KEY_ID_COGNITO";
   String ACCESS_KEY_VALUE = "SECRET_ACESS_KEY_COGNITO";
   String IDENTITY_POOL_ID =
       "eu-central-1:581b95ad-2144-4e38-b112-028abe2bac0a";
   String LOGIN_PROVIDER = "login.progettoquestionari.dev";
   String BUCKET_NAME = "questionari-images";
   /**
   * Get the User's token from AWS Cognito
   * @param idUser the id of the logged user.
   * @return
       GetOpenIdTokenForDeveloperIdentityResult instance
   */
   GetOpenIdTokenForDeveloperIdentityResult getToken(int
       idUser);
}

public class AWSTokenImpl implements AWSToken {

   /**
   * Get the User's token from AWS Cognito
   * @param idUser the id of the logged user.
   * @return
       GetOpenIdTokenForDeveloperIdentityResult instance
   */
   @Override
   public GetOpenIdTokenForDeveloperIdentityResult
       getToken(int idUser){
      AmazonCognitoIdentity identityClient = new
          AmazonCognitoIdentityClient(
      new BasicAWSCredentials(ACCESS_KEY_ID,
          ACCESS_KEY_VALUE)
      );
      identityClient.setRegion(REGION);
```

```
        GetOpenIdTokenForDeveloperIdentityRequest request =
        new GetOpenIdTokenForDeveloperIdentityRequest();
        request.setIdentityPoolId(IDENTITY_POOL_ID);
        HashMap<String,String> logins = new HashMap<>();
        logins.put(LOGIN_PROVIDER, ""+idUser);
        request.setLogins(logins);
        request.setTokenDuration(60 * 5L);
        GetOpenIdTokenForDeveloperIdentityResult response =
        identityClient.getOpenIdTokenForDeveloperIdentity(request);
        return response;
    }
}

@GetMapping(path = "/getToken/{id}")
public ResponseEntity<String> getToken(@PathVariable int id){
    GetOpenIdTokenForDeveloperIdentityResult response =
        awsToken.getToken(id);
    return new
        ResponseEntity<>("{\"token\":\""+response.getToken()+"\","
        +
        "\"identityToken\":\"" + response.getIdentityId() +"\"
            , " +
        "\"region\":\""+ AWSToken.REGION+"\", " +
        "\"identityPoolId\":\""+
            AWSToken.IDENTITY_POOL_ID+"\", " +
        "\"bucketName\":\""+ AWSToken.BUCKET_NAME+"\"}",
            HttpStatus.CREATED);
```

2.14 Anti-patterns

Usando il tool Understand abbiamo rilevato 2 external butterflies, nei package exception e in model. Non le abbiamo eliminate perchè ci avrebbe portato a creare codice duplicato.

Abbiamo in oltre rilevato un external breakable nel package controller che non abbiamo eliminato per lo stesso motivo delle external butterflies

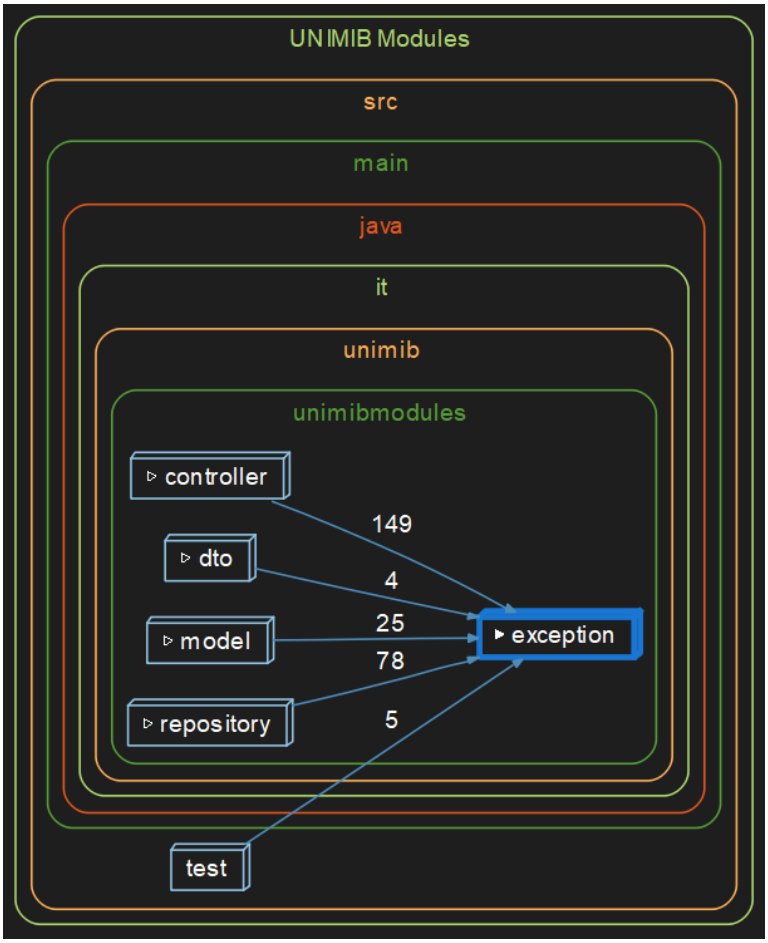

Fig. 11: Antipattern

2.15 Architettura di deployment

2.15.1 AWS - Amazon Web Services

L'intera infrastruttura dell'applicazione si basa sui servizi offerti da AWS (Amazon Web Services). AWS offre servizi di cloud computing, on-demand e pay-per-use, che possono appartenere alle classi IAAS (es. EC2, Load balancer), PAAS (es. Cloud9, Elastic Beanstalk) e SAAS (es. SNS). L'applicazione è raggiungibile al seguente link: UNIMIB Modules. In questa sezione dell'analisi sono descritti i servizi utilizzati in UNIMIB Modules.

- **RDS - Relational Database Service**

  RDS è stato utilizzato come database dell'applicazione. Inoltre, per poter garantire la persistenza dei dati è stato attivato il backup automatico. Un estensione dell'utilizzo attuale consiste nell'abilitazione dei replica sets per poter garantire ancora più affidabilità al sistema.

- **Elastic Beanstalk**

  Elastic Beanstalk si trova al centro dell'intera infrastruttura dell'applicazione. Esso è un servizio PAAS che facilita la distribuzione di applicazioni web andando ad astrarre la gestione delle istanze fisiche EC2, un servizio per l'utilizzo di macchine fisiche condivise nel cloud, e il processo stesso di installazione e distribuzione

delle versioni applicative. Nel caso di UnimibModules il servizio dispone un ambiente Java con auto scaling e load balancing. Il load balancing permette di distribuire le richieste su diverse istanze EC2 garantendo la disponibilità e tolleranza del sistema ad eventuali fault delle macchine fisiche. L'auto scaling interagisce con il load balancing andando ad aggiungere o rimuovere elementi dal pool di istanze EC2 per poter minimizzare l'utilizzo di forza computazionale nei momenti di basso carico e per poter rispondere al meglio nelle situazioni di carico opposte.

- **S3 - Simple Storage Service**

S3 è un servizio che permette l'archiviazione di oggetti all'interno di buckets, cartelle di un'entità di storage offerte da S3. Gli oggetti possono essere di qualsiasi tipo, ma nel caso di UnimibModules si tratta solamente di immagini riferite alle domande del sistema. S3 garantisce sicurezza negli accessi, disponibilità e persistenza dei dati tramite i backup automatici. Utilizzando questo servizio non è più necessario archiviare le immagini direttamente nel file system dell'istanza che esegue l'applicazione, ma si dividono le entità per migliorare in generale le performance applicative. Il bucket UnimibModules non è pubblico, quindi è protetto da policy di sicurezza AWS.

- **IAM - Identity and Access Management**

Nell'applicazione UnimibModules, IAM è stato utilizzato per poter creare la policy di accesso ad S3, sfruttata dal role associato all'identity pool di Cognito. La policy concede l'accesso al bucket S3 per le operaazioni di PUT, GET e DELETE.

- **Cognito**

AWS Cognito

Cognito permette di aggiungere strumenti di registrazione degli utenti, accesso e controllo degli accessi alle app Web e per dispositivi mobili. Tutto questo integrandosi con vari identity provider come google, facebook o cu stom. Un'altra sua applicazione riguarda la gestione degli accessi da parte degli utenti e risorse remote. Questo ultimo caso riguarda proprio il campo di utilizzo del servizio in UnimibModules. Per permettere agli utenti di comunicare direttamente con il bucket S3, per poter applicare il valet key, è necessario utilizzare Cognito come access manager. Si definisce quindi la differenza tra user-pool e identity-pool. Il primo è una directory che permette il login e signup degli utenti, come descritto precedentemente. Il secondo è utilizzato per gestire gli accessi degli user ai servizi AWS, appartenenti all'applicazione stessa, con credenziali temporanee. L'identity pool, identificato dall'identityPoolID, possiede un IAM role a cui è stata associata la policy precedentemente descritta. Questo ruolo rappresenta una matrice dalla quale andare a generare nuovi utenti temporanei. Per poter ottenere l'accesso diretto a un servizio è necessario eseguire il seguente work-flow:

- **1.** Autenticazione dello user con il server
- **2.** Il server comunica dell'avvenuto login dello user a Cognito per ottenere l'identity ID e il token di accesso OpenID settandone la durata.
- **3.** Per ottenere le vere e proprie credenziali lo user scambia il token ricevuto(token openID e identityID) con Cognito.
- **4.** Le credenziali sono utilizzate per accedere direttamente ai servizi AWS secondo i termini della policy dell'Identity pool.

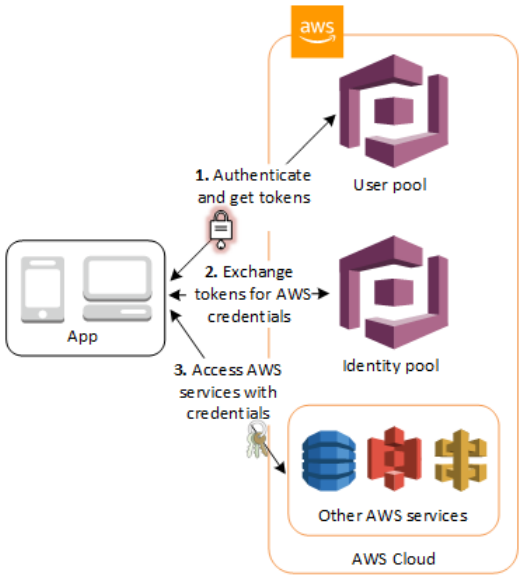

Fig. 12: Utilizzo combinato di user e identity pool

- **STS - Security Token Service**

STS viene utilizzato per l'effettiva generazione di chiavi di accesso temporanee a servizi AWS. Nel caso specifico viene sfruttato da Cognito per la creazione delle credenziali temporanee di S3.

2.15.2 Valet Key - Gestione immagini

Il Valet Key, introdotto nel capitolo dei pattern architetturali, è strettamente legato ai servizi AWS appena elencati.

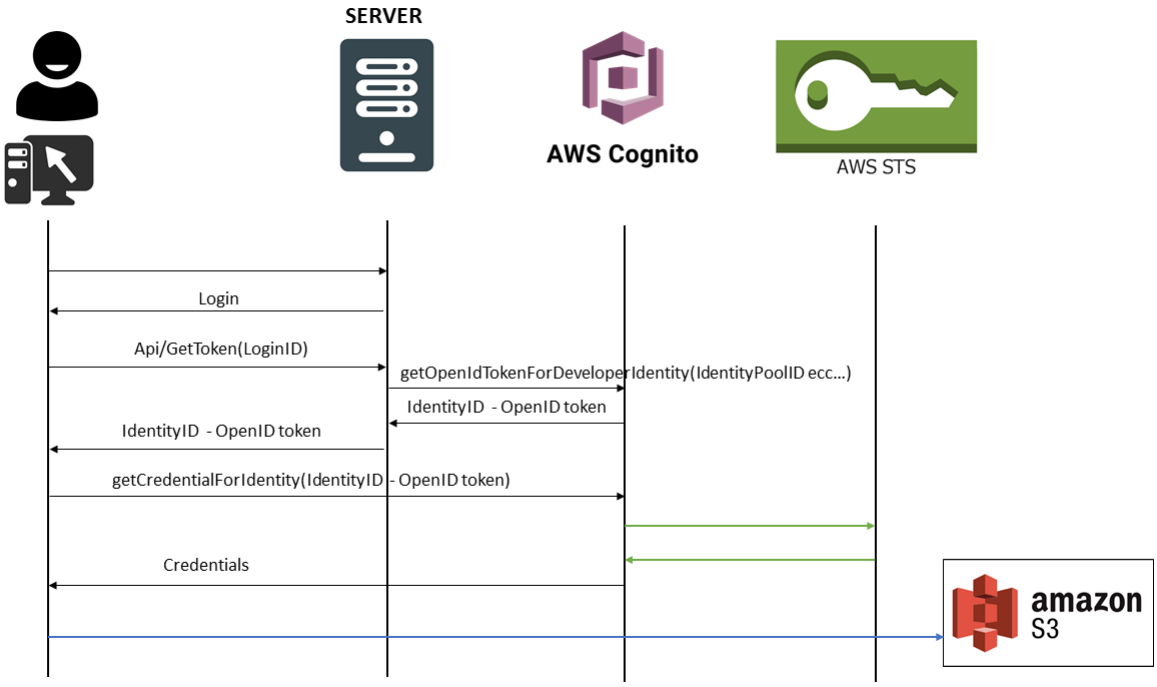

Fig. 13: Valet Key- Flusso operativo

Il pattern è stato scelto per deresponsabilizzare l'applicazione dal compito di gestire gli stream di dati (upload e download) inerenti alle immagini andando a mantenere la completa sicurezza dello storage remoto, cioè S3.
Viene definito il seguente flusso operativo:

- Login - Lo user deve essere loggato nell'applicazione per poter richiedere il valet key;
- Quando il codice client necessita, per esempio, di caricare un'immagine allegata ad una nuova domanda invia la prima richiesta all'API dell'applicazione chiamata GetToken, allegando l'ID utente per dimostrare l'effettivo passaggio dal punto precedente;

- Il server elabora la richiesta ed esegue la chiamata al metodo getOpenIdTokenForDeveloperIdentity tramite l'AWS SDK. Il server possiede le credenziali root dell'intero sistema quindi è in grado di avviare quest'ultima funzione passando come parametri l'ID dell'identity pool a cui si fa riferimento, la regione AWS di appartenenza e la durata del token.
- Il client ottiene dal server l'identityID (identifica un'istanza di un ruolo Cognito creato in AWS) e il token di accesso.
- Il client esegue la richiesta delle effettive credenziali temporanee direttamente a Cognito.
- Cognito delega il compito della creazione delle credenziali con policy pre-generata ad AWS STS.
- Il client utilizza le credenziali per interagire con il bucket S3.

3 Sviluppo

3.1 Piano dello sprint

È adottato un metodo di processo di sviluppo Agile seguendo le direttive dell' Unified Process. Il Product backlog contiene i task, normalmente associati ad un caso d'uso, da sviluppare nei vari sprint. Ogni sprint prevede le seguenti fasi:

- **Sprint meeting** per la composizione dello sprint backlog;
- **Analisi e progettazione** per aggiornare o creare componenti UML utili al task;
- **Bulding e testing** per lo sviluppo e testing del task;
- **Review e refactoring** per la revisione generale del lavoro effettuato e della qualità del codice (architectural smell, code smell ecc...);
- **Retrospective meeting** per chiudere con il team lo sprint presentando problemi, modifiche ecc...

3.2 GitHub Projects

Per la gestione delle attività abbiamo utilizzato GitHub Projects, che mette a disposizione una kanban board personalizzabile in cui inserire le attività. Queste attività possono essere convertite in issues in modo da poterle assegnare, mantenere uno stato chiuso/aperto, impostare un'etichetta (per esempio miglioramento, bug o documentazione) e associare una pull request.

3.3 CI/CD

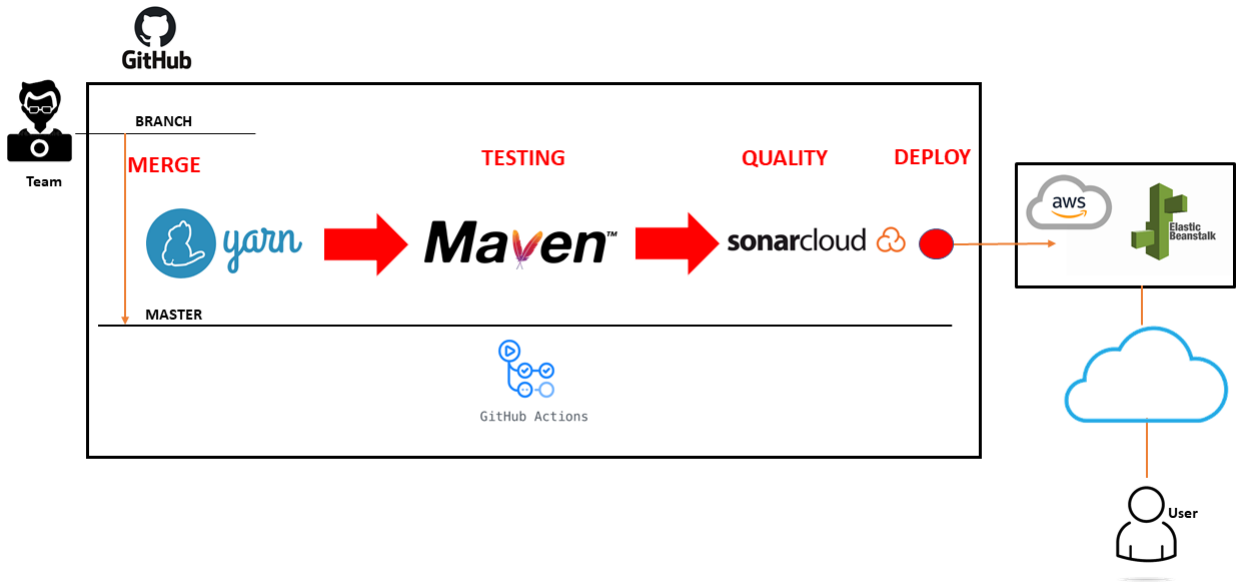

Fig. 14: CI/CD Workflow

3.4 Linguaggi

Il back-end è basato sul framework Java Spring. Per quanto riguarda il codice client è stato utilizzato AngularJs come middleware tra il DOM HTML e il server.

3.5 Best Practices

Durante la realizzazione del progetto, per la scrittura del codice sono state applicate diverse best practices per migliorare la qualità del codice e per renderlo più flessibile, riutilizzabile e mantenibile.

3.5.1 Lombok

Lombok è una libreria Java che sostituisce getter e setter delle classi attraverso l'ausilio di annotazioni. Lombok ci ha permesso di snellire le varie classi implementate.

```
public class Question {

   /**
   * The id of the answer.
   */
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   @Getter @Setter private int id;

   /**
   * The image's url of the question.
   */
   @Getter @Setter private String urlImage;

   /**
   * The text of the question.
   */
   @Getter @Setter private String text;
}
```

3.5.2 Enum al posto delle constanti

L'enum è un tipo di dato che permette ad una variabile di poter assumere un valore tra quelli definiti nell'insieme di costanti. Gli enum sono stati usati in 2 casi particolari:

- **Question type**: Il sito permette agli utenti registrati di poter creare domande che possono essere aperte, chiuse a scelta singola oppure chiuse a scelta multipla. L'enum, quindi, ha 3 valori: ENUM, MULTIPLECLOSED, MULTIPLEOPEN.
- **UnitOfWork operation**: L'enum in questione definisce le operazioni della Unit Of Work. Ogni operazione corrisponde ad un valore tra: CREATE, MODIFY, DELETE.

```
public enum QuestionType {
   OPEN,
```

```
    MULTIPLECLOSED,
    SINGLECLOSED
}

public enum UnitOfWorkOperations {

    INSERT("INSERT"),
    DELETE("DELETE"),
    MODIFY("MODIFY");

    @Getter private final String value;

    UnitOfWorkOperations(String value) {

        this.value = value;
    }
}
```

3.5.3 Lambdas e Streams

Alcune parti del codice sono state implementate attraverso l'utilizzo delle lambda expressions e degli stream. La soluzione è stata adottata per i principali vantaggi che offrono:

- Iterare su una sequenza mentre si eseguono operazioni, in una linea
- Filtrare facilmente gli elementi di una sequenza

```
public void commitInsert(int surveyId, int userId) {

   if (uofContext.size() == 0 ||
       !uofContext.containsKey(UnitOfWork.INSERT)) {
      return;
   }

   List<Answer> answerList = uofContext.get(UnitOfWork.INSERT);
   answerList.stream()
   .filter(answer -> answer.getSurvey().getId() == surveyId &&
       answer.getUser().getId() == userId)
   .collect(Collectors.toList())
   .forEach(answer -> {
      add(answer);
      answerList.remove(answer);
   });
}
```