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

Prevediamo la realizzazione di un'applicazione web chiamata UNIMIB Questionari, un ambiente all'interno del quale gestire e compilare i questionari, dotata di alta usabilità, tolleranza ai guasti, sicurezza e performance.

1.2 Posizionamento

1.2.1 Forumlazione 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'effettivo invio di email.
5	Domanda	Un elemento testuale o multimediale (immagine) contenente delle risposte.
6	Riposta	Associata ad una domanda può essere: - Aperta con eventuale numero massimo e/o minimo di caratteri - Chiuse con scelte multiple
7	Questionario	minimo e massimo di caratteri, chiusa con scelte multiple

2.2 Casi d'uso

In questa sezione sono presentati gli attori del sistema ed i relativi casi d'uso. Per alcuni di essi sono riportate anche le loro descrizioni dettagliate.

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 esterno 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.

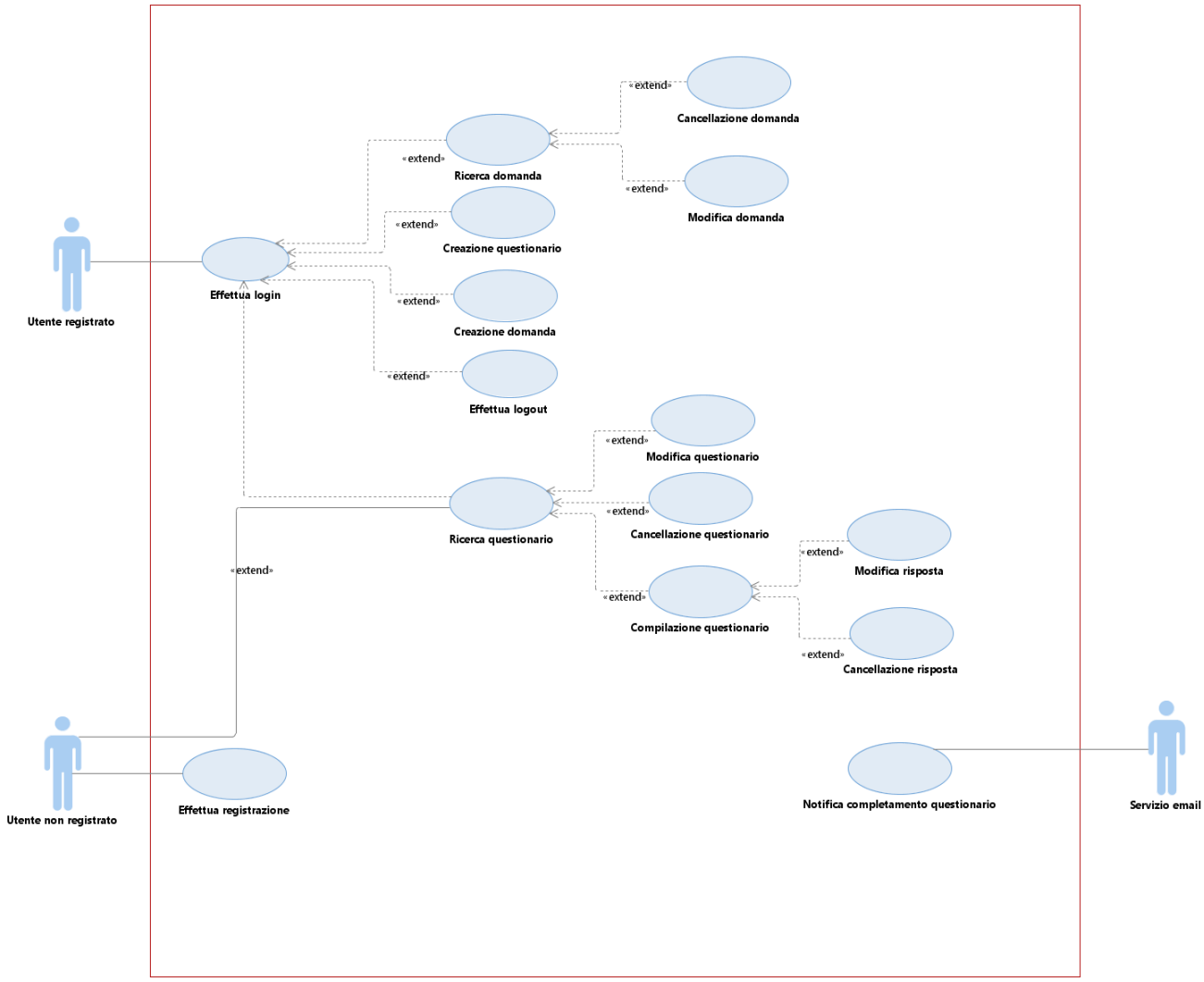

Fig. 1: Diagramma dei casi d'uso

2.3 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 Design Principles

Design Principles utilizzati durante la creazione del progetto.

- **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 inversione delle dipendenze**: I moduli di alto e basso livello non dipendono tra di loro ma dipendono da astrazioni.
- **Principio di segregazione delle interfacce**: Il client utilizza interfacce piccole e specifiche ma numerose per evitare dipendenza da metodi non utilizzati.
- **Principio delle dipendenze acicliche**: Il grafo delle dipendenze di pacchetti non presenta cicli.

2.5 SSD

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

- Creazione del questionario

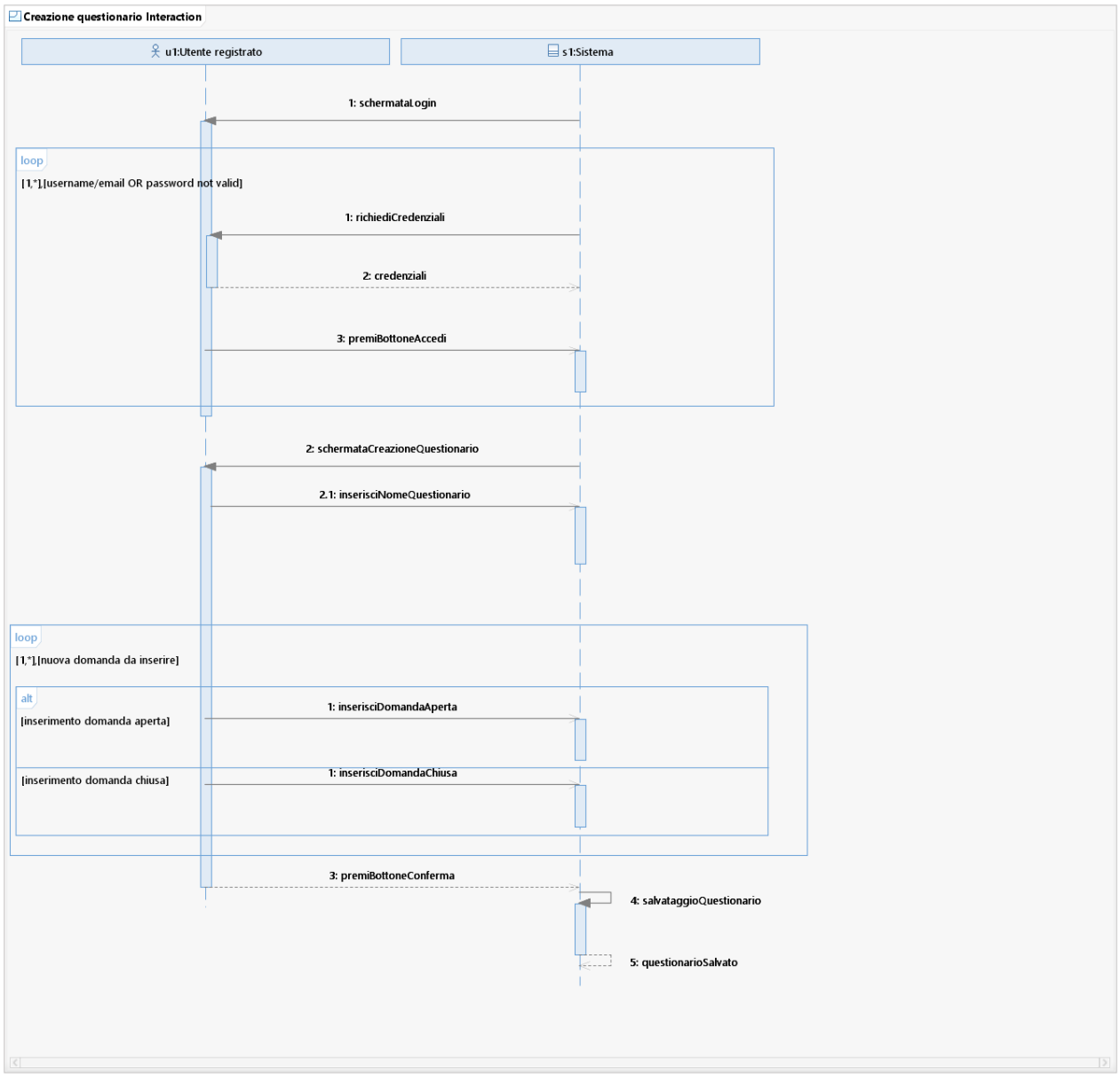

Fig. 2: SSD - Creazione del questionario

2.6 Modello di dominio

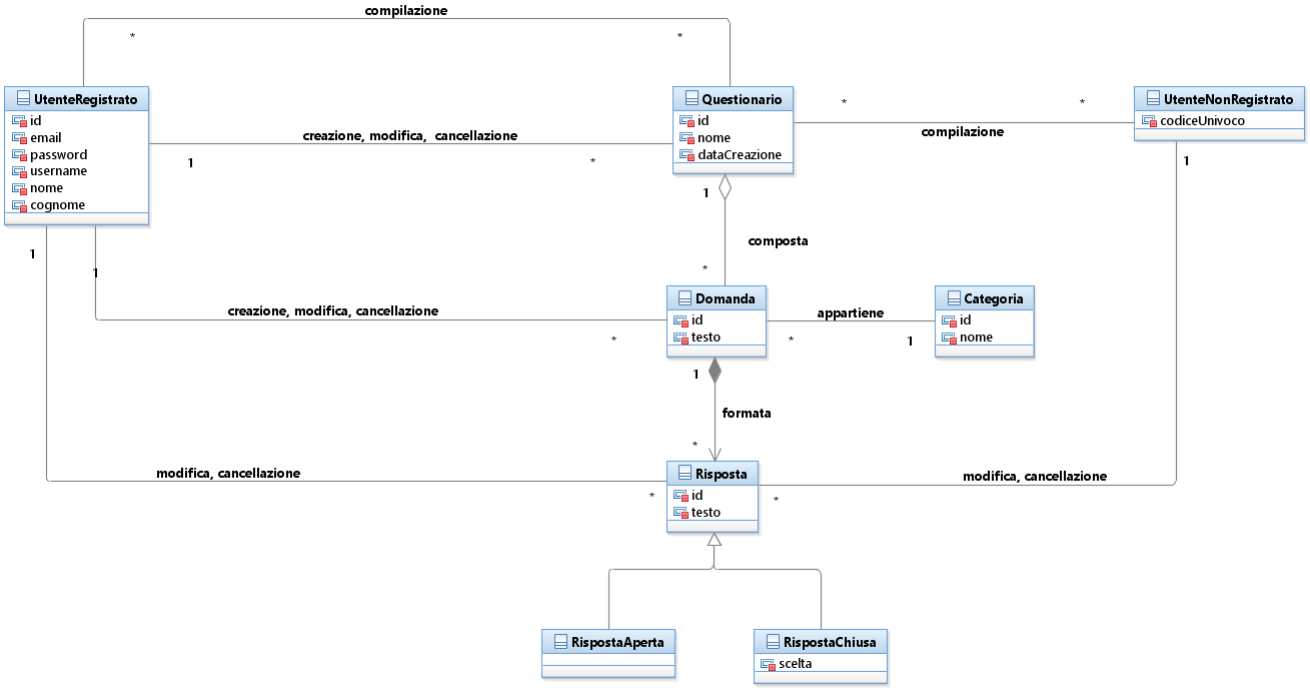

Fig. 3: Modello di dominio

2.7 Diagramma delle classi di progettazione

2.8 Diagrammi di sequenza

2.8.1 aggiugiDomanda

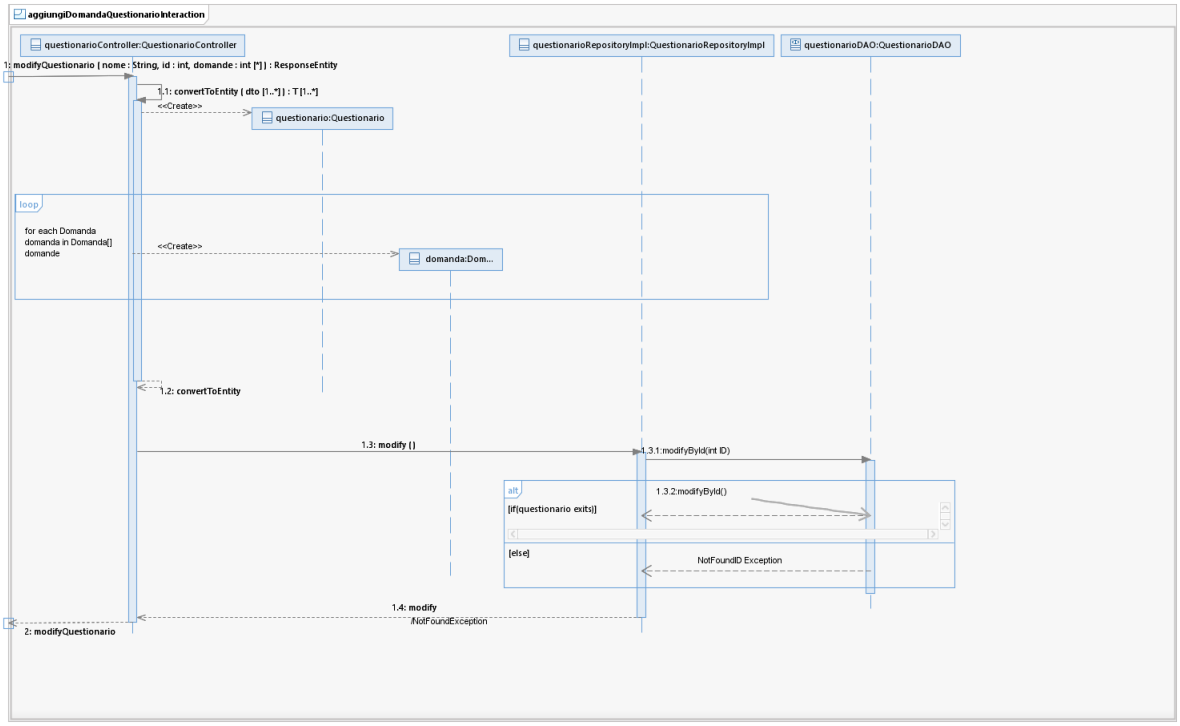

Fig. 4: Diagramma di sequenza aggiungiDomanda

2.8.2 eliminaDomanda

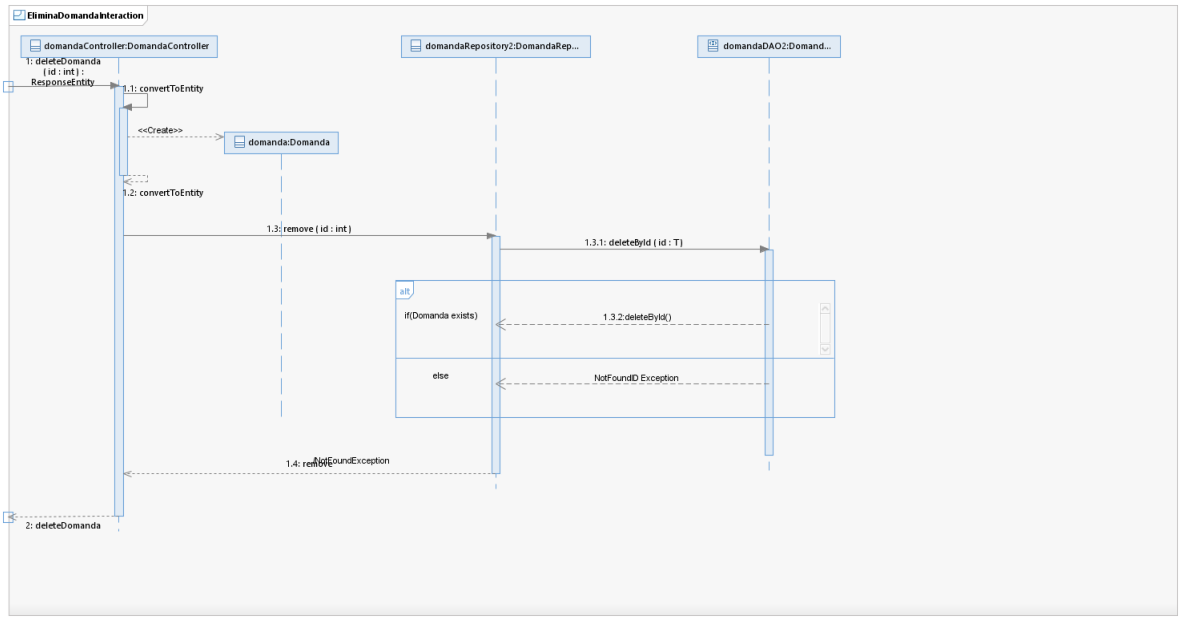

Fig. 5: Diagramma di sequenza eliminaDomanda

2.9 Diagrammi di stato

2.9.1 StateMachine Creazione Questionario

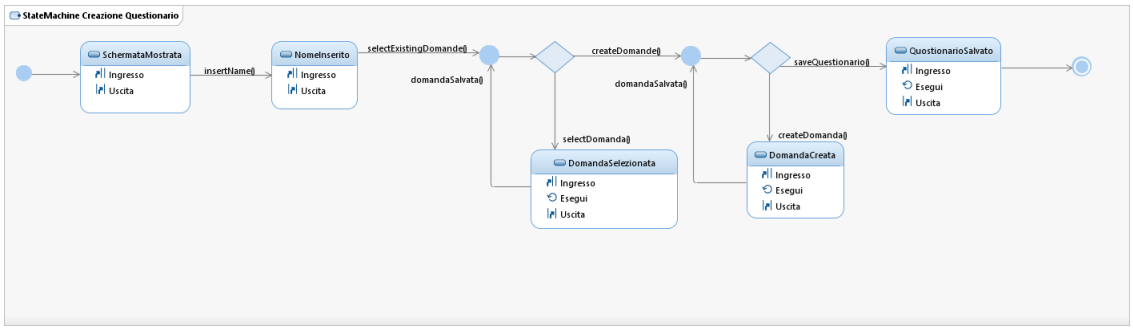

Fig. 6: Diagrammi di stato creazioneQuestionario

2.10 Diagrammi di attività

2.10.1 Creazione Questionario

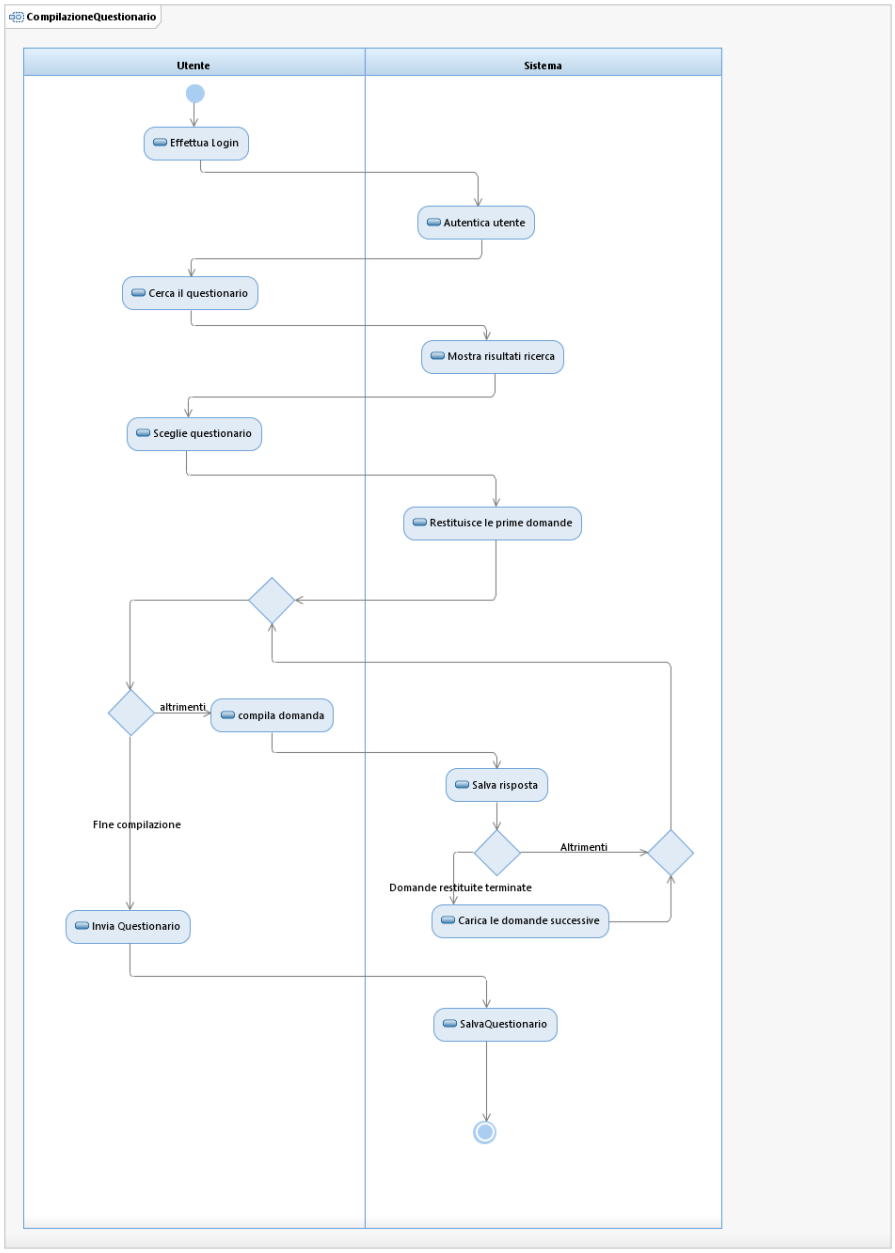

Fig. 7: Diagramma di attività creazioneQuestionario

2.11 Diagramma dell'architettura software

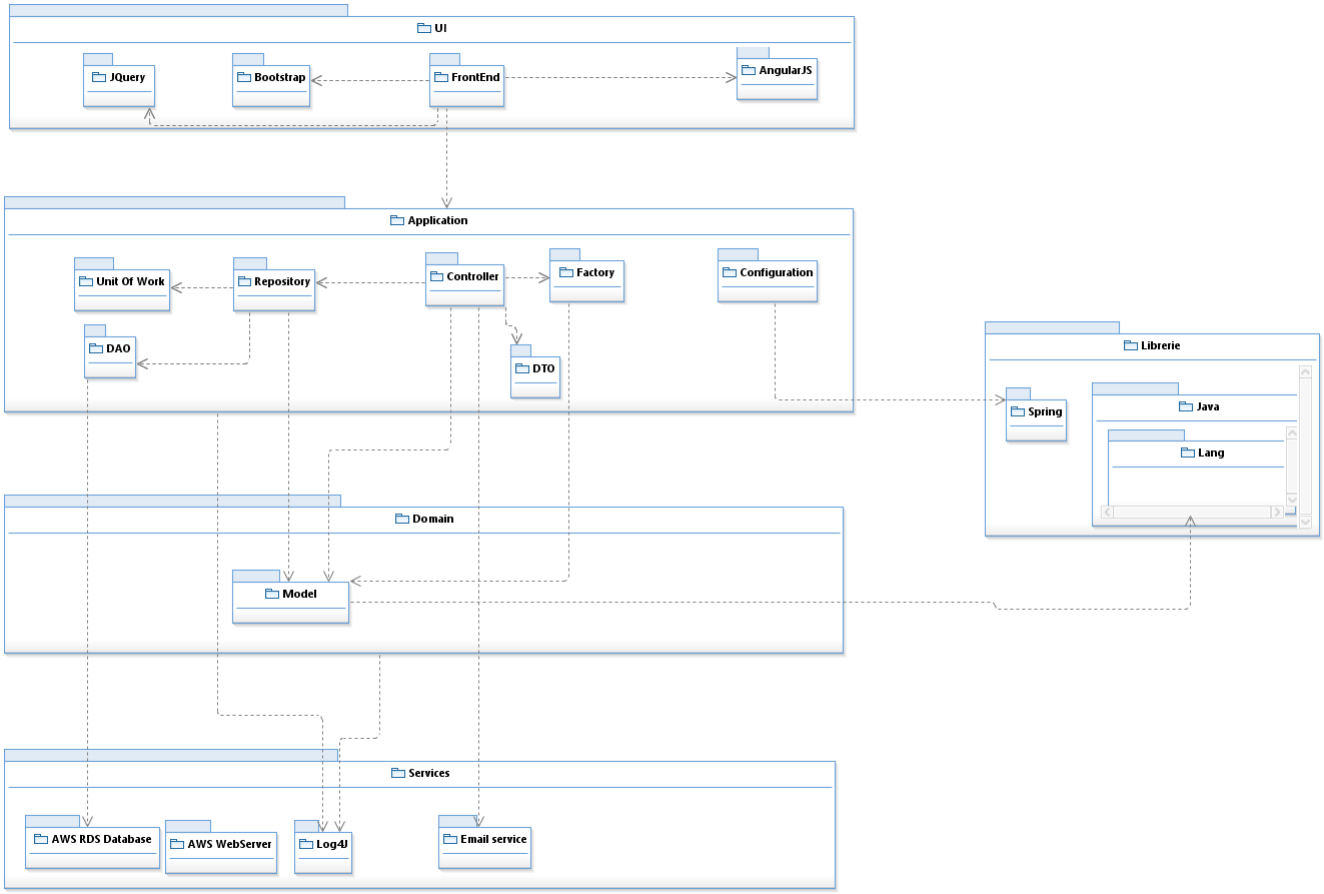

Fig. 8: Diagramma dell'architettura software

2.12 Design Patterns

Design Patterns utilizzati durante la creazione del progetto.

2.12.1 Architectural Patterns

- **Unit of Work**
  - **Nome**: Unit of Work
  - **Classificazione**: Architectural

– **Applicabilità**: Nel caso di UnimibModules 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**: Classe che si occupa di 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) {

logger.debug("Registering Answer with id {} for insert in
```

```
    context.", answer.getId());
register(answer, UnitOfWork.INSERT);
}

/**
* Adds <code>answer</code> to the elements to be modified.
* @param answer the answer that will replace the Answer with the
    same id
* @see UnitOfWork#registerModified
*/
@Override
public void registerModified(Answer answer) {

logger.debug("Registering Answer with id {} for modify in
    context.", answer.getId());
register(answer, UnitOfWork.MODIFY);
}
```

- **Data Mapper**
  - **Nome**: Data Mapper
  - **Classificazione**: Architectural
  - **Applicabilità**: Nel caso di UnimibModules, si effettua il mapping per la conversione dei dati sia da un recupero dati dal database e sia per un invio dati verso il database
  - **Partecipanti**:
    - CategoriaDAO
    - Categoria
    - RispostaDAO
    - Risposta
    - DomandaDAO
    - Domanda
    - QuestionarioDAO
    - Questionario
    - UtenteDAO
    - Utente

        * RispostaChiusaDAO
        * RispostaChiusa
    - **Scopo**: Classe che si occupa della conversione dei dati tra database e dominio

- **Front Controller**: Oggetto che fornisce un punto centralizzato per la gestione delle richieste

- **Lazy Loading**
    - **Nome**: Data Mapper
    - **Classificazione**: Architectural
    - **Applicabilità**: Nel caso di UnimibModules, quando si desidera caricare tutti i questionario o tutte le domande di 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
* @see it.unimib.unimibmodules.exception.NotFoundException
* @see it.unimib.unimibmodules.exception.ExceptionController
#handleNotFoundException
*/
@GetMapping("/findAllSurveysNoQuestionLazy")
public ResponseEntity<List<SurveyDTO>>
    findAllSurveysNoQuestionLazy(@RequestParam int offset,
   @RequestParam int limit) throws NotFoundException {
```

```
Iterable<Survey> surveys = surveyRepository.getAllLazy(offset,
    limit);
logger.debug("Retrieved all Surveys.");
List<SurveyDTO> surveysDTO = new ArrayList<>();
for (Survey survey : surveys) {
   surveysDTO.add(convertToDTOAndSkipQuestions(survey));
}
logger.debug("Retrieved {} surveys.", surveysDTO.size());
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
* @see SurveyRepository#getAll()
*/
@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.");
}
}
```

2.12.2 Data Patterns

- **Data Transfer Object**
  - **Nome**: Data Transfer Object
  - **Classificazione**: Data
  - **Applicabilità**: Nel caso di UnimibModules, quando si vogliono inviare dati al client o si ricevono dati dal client, questi vengono trasportati attraverso il corrispondente DTO.

- **Partecipanti**:
  - DTOMapping
  - UtenteDTO
  - UtenteController
  - QuestionarioDTO
  - QuestionarioController
  - DomandaDTO
  - Domanda Controller
  - RispostaDTO
  - RispostaController
  - RispostaChiusaDTO
  - RispostaChiusaController
  - CategoriaDTO
  - QuestionarioDomandeDTO
- **Scopo**: Oggetto che trasporta data tra i processi per poter ridurre il numero di chiamate ai metod
- **Codice d'esempio**

```
public abstract class DTOMapping<M, T> {

/**
 * The instance of modelMapper that will be used to convert
    Question to QuestionDTO and vice versa.
 */
protected final ModelMapper modelMapper;

@Autowired
protected DTOMapping(ModelMapper modelMapper) {

   this.modelMapper = modelMapper;
   modelMapper.getConfiguration().
   setMatchingStrategy(MatchingStrategies.STANDARD);
   modelMapper.getConfiguration().setImplicitMappingEnabled(false);
}

/**
 * Converts an instance of M to an instance of T
```

```
 * @param  value an instance of M
 * @return      an instance of T, containing the serialized data
    of value
 */
public abstract T convertToDTO(M value);

/**
 * Converts an instance of T to an instance of M
 * @param  dto   an instance of T
 * @return    an instance of M, containing the deserialized data of
    T
 * @throws FormatException
 * @throws EmptyFieldException when one of the required field is
    empty
 * @throws NotFoundException when one of the queries fails
 * @throws IncorrectSizeException
 */
public abstract M convertToEntity(T dto) throws FormatException,
   EmptyFieldException, NotFoundException, IncorrectSizeException;
}

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);
        });
}
```

- **Data Access Object**
  - **Nome**: Data Access Object
  - **Classificazione**: Data
  - **Applicabilità**: Nel caso di UnimibModules, 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**: Oggetto che rappresenta 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);
    }
}
```

- **Repository**
  - **Nome**: Repository
  - **Classificazione**: Data
  - **Applicabilità**: Nel caso di UnimibModules, le interfacce Repository permettono di fare da intermediario tra l'accesso ai dati che sia controller o DAO con il resto dell'applicazione
  - **Partecipanti**:
    * RispostaRepository
    * RispostaController
    * UtenteRepository
    * UtenteController

    - DomandaRepository
    - DomandaController
    - QuestionarioRepository
    - QuestionarioController
- **Scopo**: Interfaccia che si occupa di mediare tra la logica di accesso 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;
}

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);
    }
}
```

- **Façade**
  - **Nome**: Façade
  - **Classificazione**: Data
  - **Applicabilità**: Nel caso di UnimibModules, 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.

2.12.3 Security Patterns

- **Valet Key**
  - **Nome**: Valet Key
  - **Classificazione**: Security
  - **Applicabilità**: Nel caso di UnimibModules, 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 valtet key si occupa della sicurezza durante l'interazione tra il client e il data storage.

– **Partecipanti**: DA SISTEMARE + LINK SEZIONE AWS
  * 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);
   logger.debug("Get token for user {}", id);
   return new
       ResponseEntity<>("{\"token\":\""+response.getToken()+"\"," +
         "\"identityToken\":\"" + response.getIdentityId() +"\" , "
            +
         "\"region\":\""+ AWSToken.REGION+"\", " +
         "\"identityPoolId\":\""+ AWSToken.IDENTITY_POOL_ID+"\", " +
         "\"bucketName\":\""+ AWSToken.BUCKET_NAME+"\"}",
            HttpStatus.CREATED);
```

2.13 Architettura di deployment

2.13.1 Diagramma di deployment

2.13.2 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 segue link: UnimibModules. In questa sezione dell'analisi sono descritti i servizi utilizzati in UnimibModules.

- **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 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. 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**

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 costum. 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.** Server: Comunicazione dell'avvenuto logging 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.

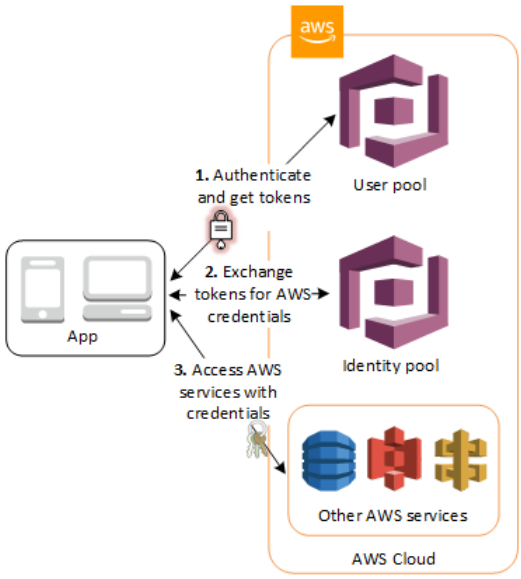

Fig. 9: Utilizzo combinato di user e identity pool

- **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.

2.13.3 Valet Key - Gestione immagini

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

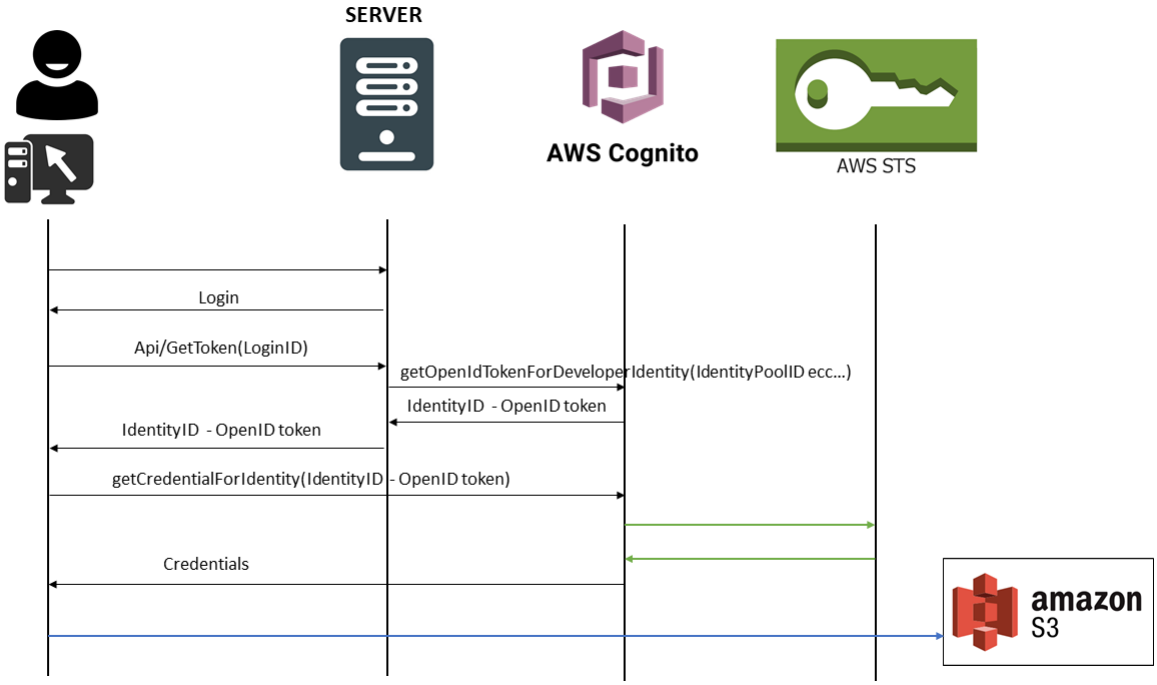

Fig. 10: Valet Key- Flusso operativo

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.

2.14 Modello E-R

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 Linguaggi

L'applicazione web è interamente basata sul framework Java Spring. Per quanto riguarda il codice client è stato utilizzato AngularJs come middleware tra il DOM HTML e il server.

3.3 Workflow per la Continuos Integration

3.4 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.

3.4.1 Lombok

Lombok è una libreria Java che sostituisce pezzi di codice ridondanti attraverso l'ausilio di annotazioni. Nel caso di UnimibModules, Lombok è stato utilizzato per evitare di scrivere getter e setter per ogni attributo all'interno delle 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;
}
```

3.4.2 Enum instead of int constants

L'enum è un tipo di dato che permette ad una variabile di poter assumere un valore tra quelli predefiniti nell'insieme di costanti. Nel caso di UnimibModules, 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;
   }
}
```

3.4.3 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);
          });
    logger.debug("Inserted registered answers for user {} and survey {}.",
        userId, surveyId);
}
```