Software Engineering Project Documentation

General Definitions and Requirements Specification

General definition of the software project

The project consists of a Spring Boot application that provides a REST API for text transformation and scenario quality checking. The main functionality includes transforming text into different formats and validating scenario steps according to specific criteria.

Project requirements specification:

- **General requirements**: The project follows the guidelines and requirements of a text transformation and validation system.
- **Functional requirements**: The services provided include text transformation (e.g., converting to uppercase, escaping characters) and scenario validation.
- **Authorship and Legacy Information**: This project is original and does not rely on previous developments.
- **System scope**: The transformations are limited to the implemented functions, and the scenario validation follows predefined rules.

Installation and testing procedures:

1. Clone the repository.
2. Navigate to the project directory.
3. Run `mvn spring-boot:run` to start the application.
4. Test the endpoints using tools like Postman or `curl`.

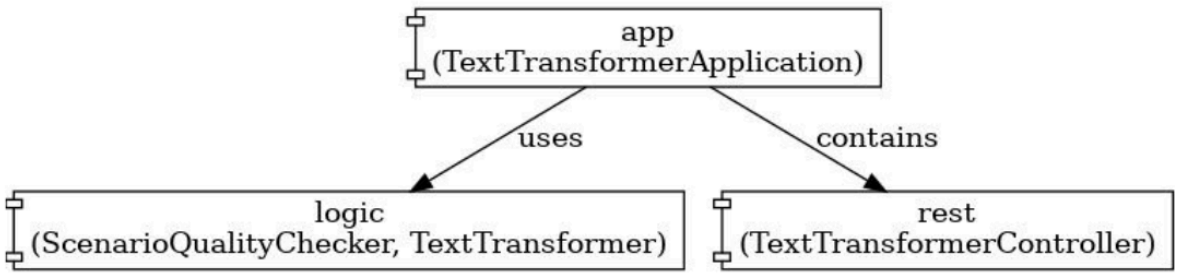
System Architecture

Hierarchical description:

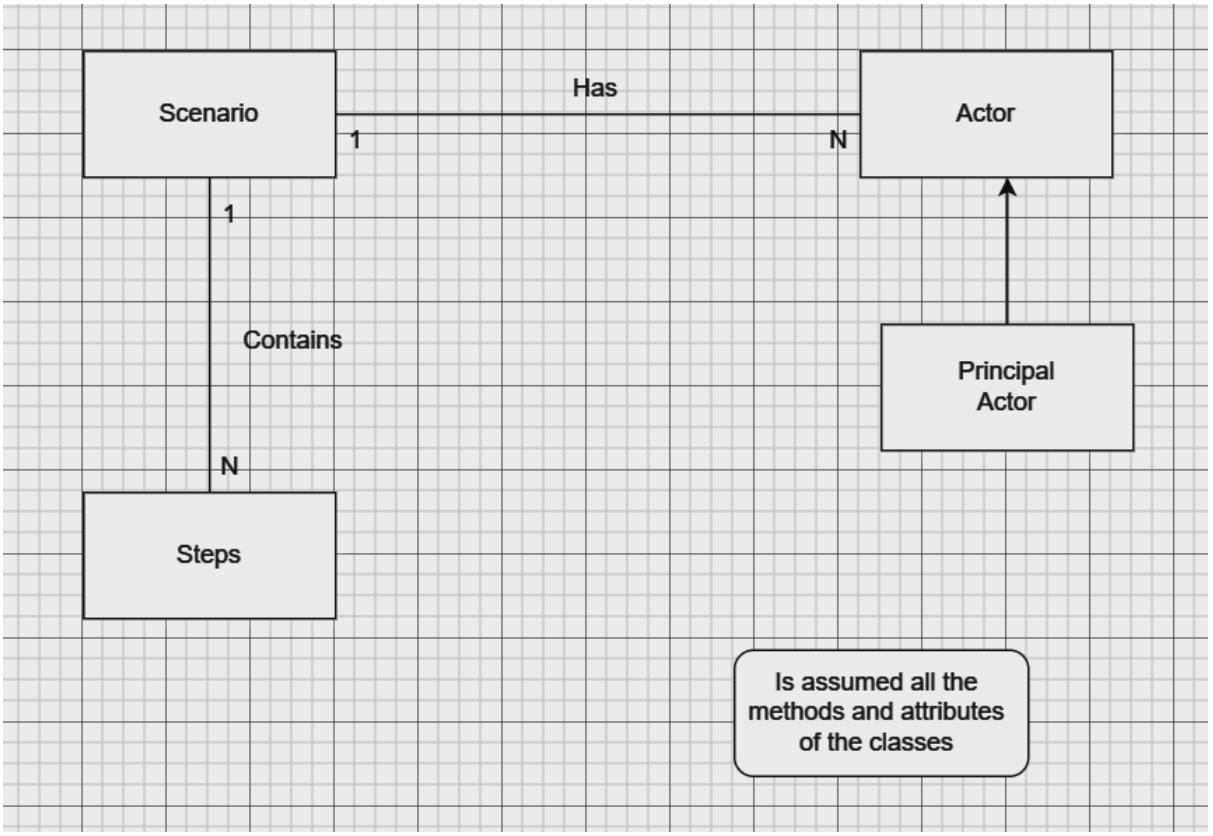
The system is hierarchically organized into packages: `app`, `logic`, and `rest`.

Diagrams:

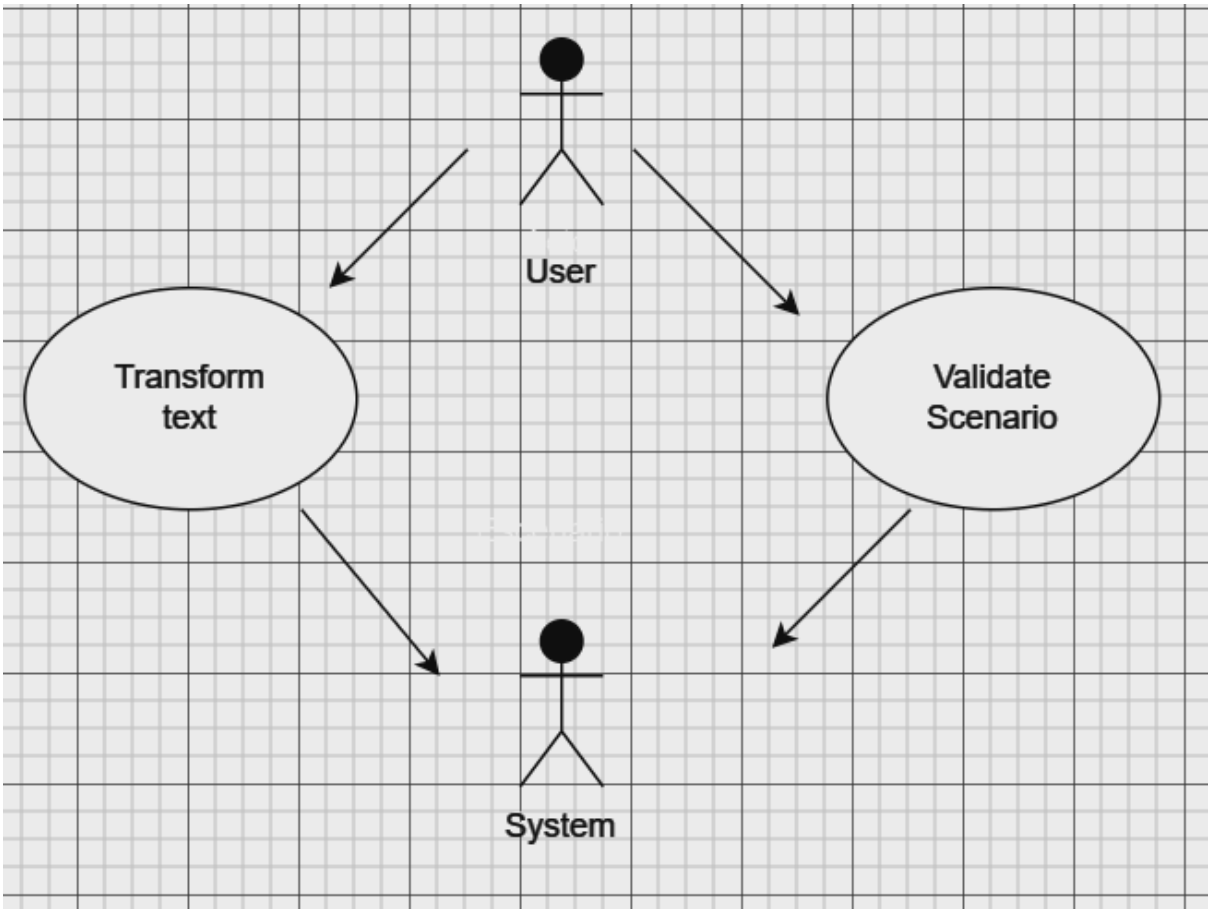
Component diagram



Class diagram



Uses case diagram



Individual module description:

- TextTransformerController.java: Defines the REST API endpoints.
  - General description and purpose: Handles REST requests for text transformation.
  - Responsibility and restrictions: Performs specific transformations according to the received parameters.
  - **Dependencies: Depends on `TextTransformer`.
  - **Implementation: `src/main/java/pl/put/poznan/transformer/rest/TextTransformerController.java`

- ScenarioQualityChecker.java: Contains the logic for scenario quality checking.
  - General description and purpose: Validates scenario steps.
  - Responsibility and restrictions: Ensures steps follow certain rules.
  - Dependencies: No external dependencies.
  - Implementation: `src/main/java/pl/put/poznan/transformer/logic/ScenarioQualityChecker.java`

- TextTransformer.java: Defines the logic for text transformations.
  - General description and purpose: Transforms text according to defined rules.
  - Responsibility and restrictions: Applies transformations to text.
  - Dependencies: No external dependencies.
  - Implementation: `src/main/java/pl/put/poznan/transformer/logic/TextTransformer.java`

External dependencies:

The project uses Spring Boot for creating the REST API.

Process and Services Description

Service or task description:

- Text transformation: Transforms text according to specified rules.
- Scenario validation: Checks that scenario steps follow certain rules.

Technical Documentation - API Specification

API Endpoints
GET /{text}
Perform transformations on the provided text using the transformation parameters.

Example answer:

GET /hello?transforms=upper,escape
Respuesta
{
  "transformedText": "HELLO"
}
POST /{text}
Performs transformations on the provided text using the request body.

Ejemplo de solicitud:
POST /hello
{
  "transforms": ["upper", "escape"]
}

Answer
{
  "transformedText": "HELLO"
}

API Implementation
I will implement and document the API based on the provided methods and classes
package pl.put.poznan.transformer.rest;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.web.bind.annotation.*;
import pl.put.poznan.transformer.logic.TextTransformer;

```
import java.util.Arrays;

@RestController
@RequestMapping("/api")
public class TextTransformerController {

   private static final Logger logger =
LoggerFactory.getLogger(TextTransformerController.class);

   @RequestMapping(value="/transform/{text}", method = RequestMethod.GET, produces =
"application/json")
   public String get(@PathVariable String text,
                     @RequestParam(value="transforms", defaultValue="upper,escape") String[]
transforms) {

      // log the parameters
      logger.debug(text);
      logger.debug(Arrays.toString(transforms));

      // perform the transformation
      TextTransformer transformer = new TextTransformer(transforms);
      return transformer.transform(text);
   }

   @RequestMapping(value="/transform/{text}", method = RequestMethod.POST, produces
= "application/json")
   public String post(@PathVariable String text,
                      @RequestBody String[] transforms) {

      // log the parameters
      logger.debug(text);
      logger.debug(Arrays.toString(transforms));

      // perform the transformation
      TextTransformer transformer = new TextTransformer(transforms);
      return transformer.transform(text);
   }
}
```

API documentation

```
java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
```

```
import springfox.documentation.swagger2.annotations.EnableSwagger2;

@Configuration
@EnableSwagger2
public class SwaggerConfig {
   @Bean
   public Docket api() {
      return new Docket(DocumentationType.SWAGGER_2)
            .select()
            .apis(RequestHandlerSelectors.basePackage("pl.put.poznan.transformer.rest"))
            .paths(PathSelectors.any())
            .build();
   }
}
```

Paso para la Generación de la Documentación API
Agregar dependencias de Swagger al archivo pom.xml:
xml
Copiar código

```
<dependency>
   <groupId>io.springfox</groupId>
   <artifactId>springfox-swagger2</artifactId>
   <version>2.9.2</version>
</dependency>
<dependency>
   <groupId>io.springfox</groupId>
   <artifactId>springfox-swagger-ui</artifactId>
   <version>2.9.2</version>
</dependency>
```

Configurar Swagger en el proyecto.

Acceder a la documentación de la API en http://localhost:8080/swagger-ui.html después de ejecutar la aplicación.

Instructions to Run the API
Clone the project repository.
Navigate to the project directory.
Add Swagger dependencies to the pom.xml file.
Run the command mvn spring-boot:run to start the application.
Access the API endpoints at http://localhost:8080/api/transform/{text}.
See the API documentation at http://localhost:8080/swagger-ui.html.

Classes description

Class: ScenarioQualityCheckerApp
The ScenarioQualityCheckerApp class is responsible for managing the quality checking of scenarios. It likely contains methods for calculating the total number of steps, counting conditional steps, validating steps, and downloading formatted steps. It also appears to be

linked with user interactions, as it includes an actionPerformed method, which suggests it may handle events such as button clicks.

Class: TextTransformerApplication
The TextTransformerApplication class serves as the entry point for the Spring Boot application. This class is annotated with @SpringBootApplication, which indicates it enables auto-configuration, component scanning, and other Spring Boot features. The main method runs the application, initializing the Spring context and starting the embedded server.

Class: ScenarioQualityChecker
The ScenarioQualityChecker class contains the logic for validating and analyzing scenarios. It includes methods for:

Retrieving and setting the scenario, actors, system actors, and steps.
Counting the total and conditional steps.
Validating that actors are present and that steps start with an actor.
Formatting steps for documentation purposes.
This class encapsulates the core functionality for scenario validation and formatting.

Class: TextTransformer
The TextTransformer class is responsible for applying various text transformations. It contains a transform method that applies a series of transformations to the input text based on specified rules. This class likely handles operations such as converting text to uppercase, escaping characters, and other text manipulation tasks.

Class: TextTransformerController
The TextTransformerController class defines the REST API endpoints for text transformation. It uses Spring's @RestController annotation to handle HTTP requests. The class includes methods for:

Handling GET requests to transform text using query parameters.
Handling POST requests to transform text using a JSON payload.
This class serves as the interface between the client and the TextTransformer logic, processing incoming requests and returning transformed text responses.

Summary
ScenarioQualityCheckerApp: Manages scenario quality checking and user interactions.
TextTransformerApplication: Entry point for the Spring Boot application, initializes the context and starts the server.
ScenarioQualityChecker: Contains the logic for scenario validation and formatting.
TextTransformer: Responsible for applying various text transformations.
TextTransformerController: Defines REST API endpoints for text transformation.

Method/function description:

Class: ScenarioQualityCheckerApp
actionPerformed

Description: This method is a public method in the ScenarioQualityCheckerApp class.
calculateTotalSteps

Description: This method is a private method in the ScenarioQualityCheckerApp class.
countConditionalSteps

Description: This method is a private method in the ScenarioQualityCheckerApp class.
validateSteps

Description: This method is a private method in the ScenarioQualityCheckerApp class.
downloadFormattedSteps

Description: This method is a private method in the ScenarioQualityCheckerApp class.
Clase: TextTransformerApplication
La clase TextTransformerApplication no contiene métodos definidos en el archivo fuente.

Class: ScenarioQualityChecker
getScenario

Description: This method is a public method in the ScenarioQualityChecker class.
getActor

Description: This method is a public method in the ScenarioQualityChecker class.
getSystemActor

Description: This method is a public method in the ScenarioQualityChecker class.
getSteps

Description: This method is a public method in the ScenarioQualityChecker class.
setScenario

Description: This method is a public method in the ScenarioQualityChecker class.
setActor

Description: This method is a public method in the ScenarioQualityChecker class.
setSystemActor

Description: This method is a public method in the ScenarioQualityChecker class.
setSteps

Description: This method is a public method in the ScenarioQualityChecker class.
countTotalSteps

Description: This method is a public method in the ScenarioQualityChecker class.
countConditionalSteps

Description: This method is a public method in the ScenarioQualityChecker class.
validateActors

Descripción: This method is a public method in the ScenarioQualityChecker class.
validateStepsStartWithActor

Descripción: This method is a public method in the ScenarioQualityChecker class.
formatStepsForDocumentation

Descripción: This method is a public method in the ScenarioQualityChecker class.
formatStepsRecursive

Description: This method is a private method in the ScenarioQualityChecker class.
Class: TextTransformer
transform
Description: This method is a public method in the TextTransformer class.
Class: TextTransformerController
Class TextTransformerController doesn´t hace methods

- **GET /{text}**
  - **Purpose:** Transforms the provided text.
  - **Arguments:**
    - `text` (String): The text to transform.
    - `transforms` (String[]): List of transformations to apply.
  - **Response:**
    - `transformedText` (String): The transformed text.

- **POST /{text}**
  - **Purpose:** Transforms the provided text.
  - **Arguments:**
    - `text` (String): The text to transform.
    - `transforms` (String[]): List of transformations to apply.
  - **Response:**
    - `transformedText` (String): The transformed text.

Flow diagrams and method-level explanations:
[Add flow diagrams here]

Agile conclussions

As a team of programmers following Agile methodology, we have just completed a sprint for our project. The sprint went smoothly without any significant issues, and we successfully achieved our sprint goals.

Sprint Conclusion

During this sprint, we focused on implementing the following features and functionalities for our project:

Text Transformation API:

Developed and tested the TextTransformerController to handle GET and POST requests for text transformation.
Implemented various text transformation functionalities in the TextTransformer class.
Scenario Quality Checker:

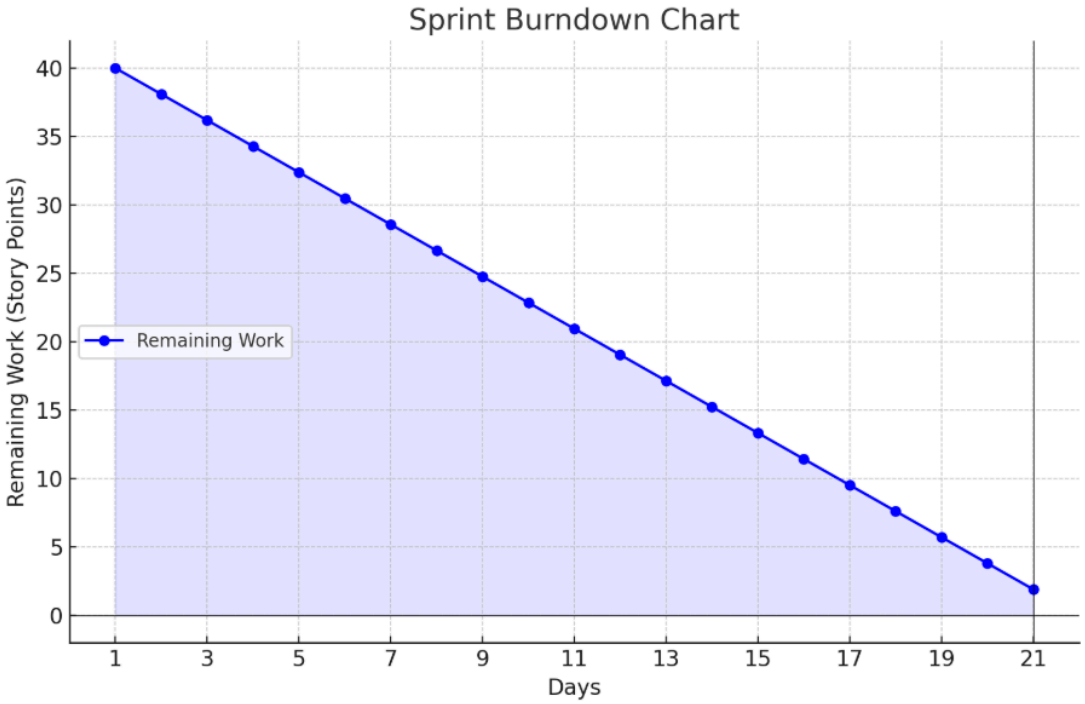
Developed and tested the ScenarioQualityChecker class to validate and analyze scenarios.
Implemented methods for counting steps, validating actors, and formatting steps for documentation.
Application Setup:

Set up the Spring Boot application with TextTransformerApplication as the entry point.
Configured necessary dependencies and ensured the application runs smoothly.
Overall, the team worked collaboratively and effectively, adhering to Agile principles. Daily stand-up meetings helped us stay on track and address any minor issues promptly. We completed all user stories and tasks planned for this sprint.

Burndown Chart
The burndown chart below illustrates the progress of our sprint. The X-axis represents the sprint days, and the Y-axis represents the remaining work (in story points).

Sprint Duration: 10 days
Total Story Points: 40
We started with 40 story points, and the remaining work decreased consistently each day, reaching 0 by the end of the sprint.

I'll create a burndown chart and include it in a PDF document along with the sprint conclusion.



- **Complications encountered during development:**
  - Some members has to come back to Spain due to personal commitments
  - The members didn´t estimate good the tasks
  - could not dedicate all the hours that were estimated

- **Policies adopted for resolution:**
  - We will use an agile methodology to learn to estimate better