Arquitectura del Sistema
Testify

OSLO

Ojeda Valeria – Sly Eduardo
Levipichun Emilio – Oyarzo Malena

La Arquitectura del software, comprende el conjunto de elementos estáticos, propios del diseño intelectual del sistema, que definen y dan forma tanto al código fuente, como al comportamiento del software en tiempo de ejecución.

Naturalmente este diseño arquitectónico ha de ajustarse a las necesidades y requisitos del proyecto. Este documento describe en términos generales, las ideas principales detrás de la arquitectura escogida para el mismo.

Tabla de contenido

Arquitectura del Sistema

Introducción

El presente Documento de Arquitectura de Software tiene como objetivo proporcionar una visión completa y estructurada de la arquitectura del sistema Testify. Este sistema está diseñado para gestionar proyectos, roles, permisos y escenarios (a modo de casos de prueba) dentro de un entorno orientado a la evaluación y el seguimiento de pruebas de software. El documento incluye una descripción detallada de la estructura del sistema, los componentes clave, las tecnologías empleadas y los flujos de interacción entre los diferentes elementos del sistema.

Propósito

El propósito de este documento es formalizar la arquitectura de software de Testify, brindando una guía clara para los desarrolladores, arquitectos de sistemas y otros miembros del equipo. Proporciona una base de referencia para el diseño, desarrollo y mantenimiento del sistema, y asegura que todos los involucrados compartan una comprensión común de los componentes y las interacciones dentro del sistema. Además, describe cómo se interrelacionan los diferentes módulos (frontend, backend, base de datos) y las capas de la arquitectura. Este documento está dirigido a desarrolladores, testers, administradores de sistemas y cualquier otra persona encargada de trabajar en el sistema, ya sea en su implementación o en su posterior evolución.

Alcance

El Documento de Arquitectura de Software de Testify aplica a todo el ciclo de vida del desarrollo del sistema, incluyendo las fases de diseño, desarrollo, pruebas y mantenimiento. Cubre la arquitectura tanto del frontend, desarrollado en Angular 18, como del backend, implementado en Spring Boot con persistencia de datos en MySQL. También abarca los aspectos de seguridad relacionados con la autenticación y autorización, mediante el uso de Spring Security y la API de Google para OAuth2.

Definiciones, Acrónimos, y Abreviaturas

API: Interfaz de Programación de Aplicaciones.

CRUD: Operaciones de Crear, Leer, Actualizar y Eliminar.

OAuth2: Protocolo abierto estándar para autenticación y autorización.

SPA: Single Page Application, aplicable a aplicaciones web de una sola página.

MySQL: Sistema de gestión de bases de datos relacional.

REST: Transferencia de Estado Representacional (Representational State Transfer), estilo de arquitectura para sistemas distribuidos.

Referencias

Documento de Requerimientos del Sistema Testify, OSLO, Septiembre 2024.

E202-OSLO-Modelo de Casos de Uso, OSLO, Septiembre 2024.

E209-OSLO-Modelo de Datos, OSLO, Septiembre 2024.

E105-OSLO-Herramientas y Tecnologías, OSLO, Septiembre 2024.

Documento de Arquitectura de Sistema, OSLO, Octubre 2024.

Prototipo Funcional Testify, OSLO, Octubre 2024.

Panorama General

Este documento está organizado en secciones que detallan la arquitectura general del sistema, las capas que lo componen, los componentes clave en cada una de estas capas y las interacciones entre ellos. Además, se describen las tecnologías empleadas y los diagramas que representan la estructura y comportamiento del sistema. La arquitectura de Testify sigue el modelo cliente-servidor con una separación clara entre el frontend (Angular) y el backend (Spring Boot), comunicados a través de una API REST.

Representación Arquitectónica

La arquitectura de software de Testify sigue el modelo clásico de arquitectura en capas, donde cada componente tiene responsabilidades claramente definidas, facilitando la separación de preocupaciones y permitiendo una mayor flexibilidad en su desarrollo y mantenimiento.

- Vista de Caso de Uso: La arquitectura refleja los requerimientos funcionales detallados en el modelo de casos de uso, donde actores como el Administrador, Gestor de Pruebas, Tester e Invitado interactúan con el sistema para realizar acciones como gestionar proyectos, roles, permisos y escenarios.
- Vista Lógica: El sistema está dividido en tres capas principales:
    - Capa de Presentación (Frontend): Desarrollada en Angular 18, se encarga de la interacción con el usuario a través de componentes visuales como el navbar, dropdown de proyectos, y formularios para la asignación de roles.
    - Capa de Servicio (Backend): Desarrollada en Spring Boot, proporciona servicios REST que gestionan la lógica de negocio, la seguridad, y la persistencia de datos a través de entidades como Usuario o Proyecto.
    - Capa de Persistencia (Base de Datos): Utiliza MySQL para almacenar los datos del sistema, incluyendo usuarios, roles, proyectos y asignaciones. La interacción con la base de datos está gestionada a través de Hibernate.

- Vista de Procesos: El flujo de procesos refleja cómo los usuarios interactúan con el sistema. Un usuario autenticado puede gestionar proyectos y roles, asignar permisos, ejecutar casos de prueba y generar reportes. Estas acciones siguen un ciclo de CRUD, con cada interacción validada y autenticada mediante Spring Security.

- Vista de Liberación: Describe la configuración del sistema en los entornos de desarrollo, pruebas y producción. El despliegue se realiza en un servidor Tomcat embebido en Spring Boot para el backend y un servidor web estático para el frontend, con la base de datos MySQL gestionada en entornos separados para pruebas y producción.

- Vista de Implementación: La estructura de carpetas del sistema sigue una organización clara:

src/main/java/com/oslo/testify: Contiene las clases Java del backend.

src/main/webapp/app/: Carpeta para el frontend en Angular.

El código está organizado en controladores, servicios, repositorios y entidades para una mayor modularidad y facilidad de mantenimiento.

Objetivos Arquitectónicos y Restricciones

Los objetivos arquitectónicos de Testify están orientados a garantizar un sistema robusto, escalable y seguro, que facilite la gestión de proyectos y casos de prueba en el contexto del desarrollo de software. Entre los principales objetivos y restricciones se destacan:

- Seguridad: Se ha implementado Spring Security para gestionar la autenticación y autorización del sistema, asegurando que solo usuarios autorizados puedan acceder a los recursos según los roles asignados (Administrador, Gestor de Prueba, Tester e Invitado). Además, se utiliza la API de Google OAuth2 para el inicio de sesión seguro.

- Garantía de Privacidad: Los datos sensibles de los usuarios y proyectos se manejan de acuerdo con las mejores prácticas de privacidad y seguridad, con acceso restringido a los roles adecuados.

- Portabilidad: El sistema se basa en tecnologías que garantizan su portabilidad en distintos entornos, como Spring Boot y Angular, ambos multiplataforma. Esto permite que el sistema se despliegue fácilmente en diferentes infraestructuras con mínimas modificaciones.

- Reuso: La modularidad del sistema asegura que componentes como los servicios REST y los controladores de roles y proyectos puedan ser reutilizados y mantenidos de manera independiente.

- Restricciones Especiales: La elección de las tecnologías, como MySQL para la base de datos y el uso de Spring Boot para el backend, está condicionada por la necesidad de soportar altos volúmenes de datos y garantizar un rendimiento óptimo. La arquitectura se ha diseñado para garantizar que el sistema pueda manejar de manera eficiente múltiples proyectos y casos de prueba asignados a distintos usuarios. Además, el sistema está diseñado solo para ser utilizado en entornos de escritorio, según lo establecido en los requerimientos no funcionales.

Objetivos Generales

El proyecto Testify tiene como objetivo principal desarrollar una plataforma robusta para la gestión y seguimiento de pruebas de software en proyectos colaborativos. El sistema está diseñado para facilitar la administración de escenarios, asignaciones de roles y permisos, y la generación de reportes relacionados con el estado de los proyectos. La motivación detrás de este desarrollo surge de la necesidad de una herramienta que permita la gestión eficiente de casos de prueba, asegurando que cada usuario tenga acceso a la información y funcionalidades que se ajusten a su rol, ya sea Administrador, Gestor de Prueba, Tester o Invitado.

Este proyecto busca no solo proporcionar una herramienta funcional, sino también mejorar la trazabilidad en la gestión de pruebas, optimizando el ciclo de vida del desarrollo de software.

Objetivos Específicos

- Gestión de Proyectos y Casos de Prueba: Implementar funcionalidades que permitan la creación, modificación, eliminación y consulta de proyectos y escenarios de prueba. Cada proyecto tendrá asignadas iteraciones, que a su vez contendrán escenarios de prueba con roles específicos asignados.

- Asignación de Roles y Permisos: Establecer un sistema de gestión de roles que permita asignar permisos específicos a los usuarios dentro de los proyectos. Los roles como Administrador, Gestor de Pruebas y Tester estarán claramente definidos, y los permisos se asignarán según las necesidades de cada proyecto.

- Seguridad y Control de Acceso: Asegurar que el sistema gestione la autenticación y autorización de usuarios mediante Spring Security y la API de Google OAuth2. Se debe implementar un control de acceso basado en roles que proteja los recursos críticos del sistema y garantice que solo los usuarios autorizados puedan acceder a la información.

- Generación de Informes: Facilitar la exportación de informes sobre el estado de los proyectos, permitiendo a los gestores y administradores visualizar el progreso de las pruebas y generar documentos en formato PDF.

Descripción de Procesos

El flujo de información en Testify sigue una arquitectura en capas que separa claramente la lógica de presentación, negocio y persistencia. El sistema está diseñado para permitir la interacción de diferentes actores, como administradores, gestores de pruebas y testers, quienes realizan tareas específicas según sus roles.

- Proceso de Gestión de Casos de Prueba: El Gestor de Pruebas crea y asigna escenarios de prueba dentro de un proyecto. Los testers acceden a los escenarios asignados, ejecutan las pruebas y reportan los resultados. El sistema permite gestionar múltiples iteraciones dentro de un proyecto, con cada iteración agrupando un conjunto de escenarios.

- Proceso de Asignación de Roles: Los administradores del sistema pueden asignar o modificar los roles de los usuarios para proyectos específicos. Esta asignación se gestiona a través de la entidad RoleAssignment, que vincula a los usuarios con roles y proyectos específicos. La seguridad de acceso a estas funciones es controlada mediante Spring Security.

- Proceso de Autenticación y Autorización: Todos los usuarios deben autenticarse a través de la API de Google OAuth2 o mediante autenticación básica HTTP. Una vez autenticados, los usuarios interactúan con el sistema según los permisos definidos por su rol.

- Proceso de Generación de Informes: El Gestor de Pruebas o Administrador puede generar informes sobre el estado de los proyectos, incluyendo detalles sobre los casos de prueba completados, en proceso o pendientes. Estos informes se exportan en formato PDF, permitiendo su distribución a otros miembros del equipo o interesados.

Estos procesos están diseñados para garantizar una interacción fluida y segura entre los distintos actores del sistema, asegurando que la información fluya de manera eficiente y que cada usuario tenga acceso a las funcionalidades que le corresponden según su rol.

Vista de Caso de Uso

En Testify, los casos de uso representan las interacciones clave que los diferentes actores realizan con el sistema para cumplir con los objetivos funcionales. A continuación, se enumeran los casos de uso más relevantes, ya que estos abarcan gran parte de la funcionalidad central del sistema:

- CRUD de Proyectos (CU09): Los administradores pueden crear, modificar, eliminar y consultar proyectos dentro del sistema. Este caso de uso es clave para la gestión de proyectos y para asegurar que los proyectos sean administrados adecuadamente en función de las necesidades de la organización(E202-OSLO-Modelo de Cas…).
- Gestión de Escenarios de Prueba (CU01): Los gestores de prueba pueden gestionar escenarios dentro de los proyectos, asignando escenarios a diferentes testers, actualizando su estado y agregando comentarios o documentación adicional.
- Asignación de Roles (CU14): Los administradores pueden gestionar los roles asignados a los usuarios, asegurando que cada usuario tenga los permisos adecuados en cada proyecto. Este caso de uso cubre gran parte de la lógica de seguridad y autorización dentro del sistema.
- Exportación de Informes (CU06): El sistema permite generar y exportar informes sobre el estado de los proyectos y las pruebas ejecutadas, los cuales pueden ser distribuidos entre los interesados. Este caso de uso es fundamental para la trazabilidad y seguimiento del progreso de los proyectos.
- Autenticación y Autorización: A través de Spring Security y la API de Google OAuth2, los usuarios acceden al sistema y realizan acciones basadas en sus roles asignados. Este proceso es fundamental para garantizar la seguridad del sistema y proteger los recursos de acceso no autorizado.

Descripción de los Actores

- Administrador: Es el actor con el acceso más completo al sistema. El Administrador puede gestionar todos los aspectos del sistema, desde la creación de usuarios hasta la asignación de roles y la administración de proyectos y escenarios. Su función es crítica para asegurar el correcto funcionamiento y control del sistema.

- Gestor de Prueba: El Gestor de Prueba tiene la responsabilidad de administrar los proyectos asignados a su usuario, gestionar los escenarios de prueba y supervisar el progreso de las pruebas. Este actor tiene acceso limitado a los proyectos asignados y solo puede gestionar los escenarios dentro de esos proyectos.

- Tester: El Tester es responsable de ejecutar las pruebas asignadas a los escenarios de prueba dentro de los proyectos. Su función incluye actualizar el estado de los escenarios, agregar comentarios y adjuntar documentos para respaldar los resultados de las pruebas.

- Invitado: El Invitado tiene un rol muy limitado, ya que solo puede consultar información básica sobre los proyectos y escenarios de prueba. Este actor no tiene permisos para modificar ni ejecutar acciones dentro del sistema.

Contexto del sistema

Testify es un sistema que opera en un entorno de gestión de proyectos de pruebas de software. En este contexto, se integra con diversos actores que realizan actividades clave para el desarrollo y la evaluación de proyectos. El sistema se comunica con otros sistemas o subsistemas a través de servicios web, utilizando una API REST que permite gestionar los proyectos, usuarios, roles y permisos.

Además, el sistema interactúa con la API de Google OAuth2 para la autenticación de usuarios, lo que garantiza un acceso seguro y controlado a los recursos del sistema(E105-OSLO-Herramientas …)(Documento de Arquitectu…). También se integra con el servidor de aplicaciones Tomcat, donde se despliega el backend en Spring Boot, mientras que el frontend desarrollado en Angular se despliega de forma separada en un servidor web estático (Documento de Arquitectu…).

Si bien Testify es un sistema autónomo en su gestión de proyectos de pruebas, su arquitectura modular permite la integración futura con otros sistemas relacionados con la gestión de calidad y desarrollo de software. Esto se puede visualizar mediante un diagrama de contexto, que ilustra la interacción entre el sistema principal, los actores, y los servicios externos, asegurando que cada subsistema interactúa de manera eficiente y segura.

Vista Lógica

La vista lógica de Testify describe cómo está estructurado el sistema en términos de clases y componentes, organizados en paquetes de diseño dentro del backend y frontend. Cada uno de estos paquetes cumple con responsabilidades específicas y contribuye al flujo general de la arquitectura del sistema.

- Capa de Presentación (Frontend): Desarrollada en Angular 18, esta capa maneja la interacción del usuario y está compuesta por componentes clave como:
  - AppComponent: Componente principal que gestiona la navegación.
  - NavbarComponent: Controla la navegación entre secciones y filtra los accesos según el rol del usuario.
  - UpdateRoleAssignmentComponent: Interfaz para la asignación de roles a los usuarios en los proyectos.
- Capa de Servicio (Backend): Implementada en Spring Boot, esta capa gestiona la lógica de negocio, conectando el frontend con la base de datos y ofreciendo servicios a través de APIs REST. Los principales paquetes de esta capa incluyen:
  - Controladores REST: UserController, ProjectController, RoleController, que manejan las operaciones CRUD para usuarios, proyectos y roles(Documento de Arquitectu…).
  - Servicios: UserService, ProjectService, RoleService, que implementan la lógica de negocio.
  - Seguridad: SecurityConfig, que gestiona la autenticación y autorización a través de Spring Security.
- Capa de Persistencia: Utiliza Hibernate y JPA para mapear objetos Java a tablas en la base de datos MySQL. Las entidades principales incluyen User, Project, Role, y RoleAssignment.

Perspectiva General

El modelo de diseño de Testify sigue una arquitectura en capas claramente definida, con el frontend y backend interconectados a través de APIs REST. Estas capas son:

- Capa de Presentación: Maneja la interacción con el usuario.
- Capa de Servicio: Gestiona la lógica de negocio y la comunicación con la base de datos.
- Capa de Persistencia: Responsable de la gestión de datos y operaciones CRUD.

Paquetes de Diseño importantes arquitectónicamente

- Paquete de Controladores (controller): Contiene las clases que gestionan las peticiones HTTP y ejecutan la lógica necesaria para interactuar con el frontend.
- UserController: Gestiona las operaciones relacionadas con los usuarios (alta, baja, modificación, autenticación).
- RoleController: Maneja la asignación de roles y permisos.
- Paquete de Servicios (service): Implementa la lógica de negocio y se comunica con los repositorios para la gestión de datos.
- UserService: Gestiona la creación y autenticación de usuarios.
- RoleService: Administra los roles y sus permisos.
- Paquete de Seguridad (config): Contiene la configuración de seguridad del sistema.
- SecurityConfig: Define las reglas de autorización y autenticación mediante Spring Security.

Vista de Procesos

El sistema Testify se organiza mediante procesos ligeros que se gestionan a través de hilos en el backend. Los controladores REST reciben peticiones del frontend y los servicios correspondientes ejecutan la lógica de negocio para procesar los datos y responder al cliente. Cada proceso involucra la autenticación y autorización del usuario antes de acceder a los recursos protegidos. La comunicación se realiza mediante HTTP, utilizando APIs REST, y las respuestas se devuelven en formato JSON.

Vista de Liberación

El software se despliega en un entorno de producción con los siguientes componentes físicos:

- Backend: Desplegado en un servidor con Spring Boot utilizando Tomcat como servidor de aplicaciones.
- Frontend: Desplegado en un servidor web estático que sirve la aplicación Angular.
- Base de Datos: El sistema utiliza una instancia de MySQL para la persistencia de datos, alojada en un servidor separado. La comunicación entre los servidores se realiza a través de una LAN, y el acceso a la aplicación se realiza mediante la web utilizando navegadores.

Vista de Implementación

El modelo de implementación sigue una estructura basada en capas. El software está dividido en:

- Capa de Presentación (Angular): Desplegada en un servidor web estático.
- Capa de Servicio (Spring Boot): Desplegada en un servidor Tomcat embebido, con servicios REST que comunican el frontend con el backend.
- Base de Datos (MySQL): Maneja la persistencia de datos de los usuarios, roles y proyectos.

Capas

- Capa de Presentación: Contiene los componentes Angular responsables de la interfaz gráfica y la interacción con el usuario.
- Capa de Servicio: Gestiona la lógica de negocio y conecta el frontend con la base de datos a través de APIs REST.
- Capa de Persistencia: Utiliza Hibernate y JPA para mapear las entidades Java a las tablas de la base de datos MySQL.

Tamaño y Rendimiento

El sistema está diseñado para gestionar múltiples proyectos y asignaciones de roles sin comprometer el rendimiento. Para ello, se utilizan optimizaciones en las consultas a la base de datos mediante Hibernate, y se minimizan las cargas de red con respuestas eficientes en formato JSON.

Calidad

La arquitectura del software contribuye a la extensibilidad, ya que nuevas funcionalidades pueden añadirse sin modificar el sistema existente. La fiabilidad se asegura mediante pruebas unitarias e integradas antes del despliegue. La portabilidad está garantizada por el uso de tecnologías multiplataforma como Angular y Spring Boot. Además, se han implementado estrictas políticas de seguridad con Spring Security para garantizar la privacidad y el acceso controlado.

Diagramas

Diagramas de Despliegue

El diagrama de despliegue de Testify ofrece una visión clara de cómo se distribuyen físicamente los componentes del sistema en diferentes nodos (máquinas o servidores). En términos simples, muestra cómo las partes del sistema se encuentran desplegadas en la infraestructura y cómo interactúan entre sí.

Descripción de los componentes:

Cliente:

- Qué es: Este componente representa cualquier dispositivo desde el cual un usuario accede al sistema Testify, como un ordenador, laptop, o incluso tablet, usando un navegador web (por ejemplo, Chrome, Firefox).
- Función: El cliente es el punto de entrada donde el usuario interactúa con la interfaz gráfica. A través del navegador, el usuario envía peticiones al servidor (para acceder a los proyectos, gestionar roles, etc.).
- Interacción: El cliente envía peticiones HTTP (solicitudes) al Servidor Web, y recibe respuestas HTTP de vuelta.

Servidor Web:

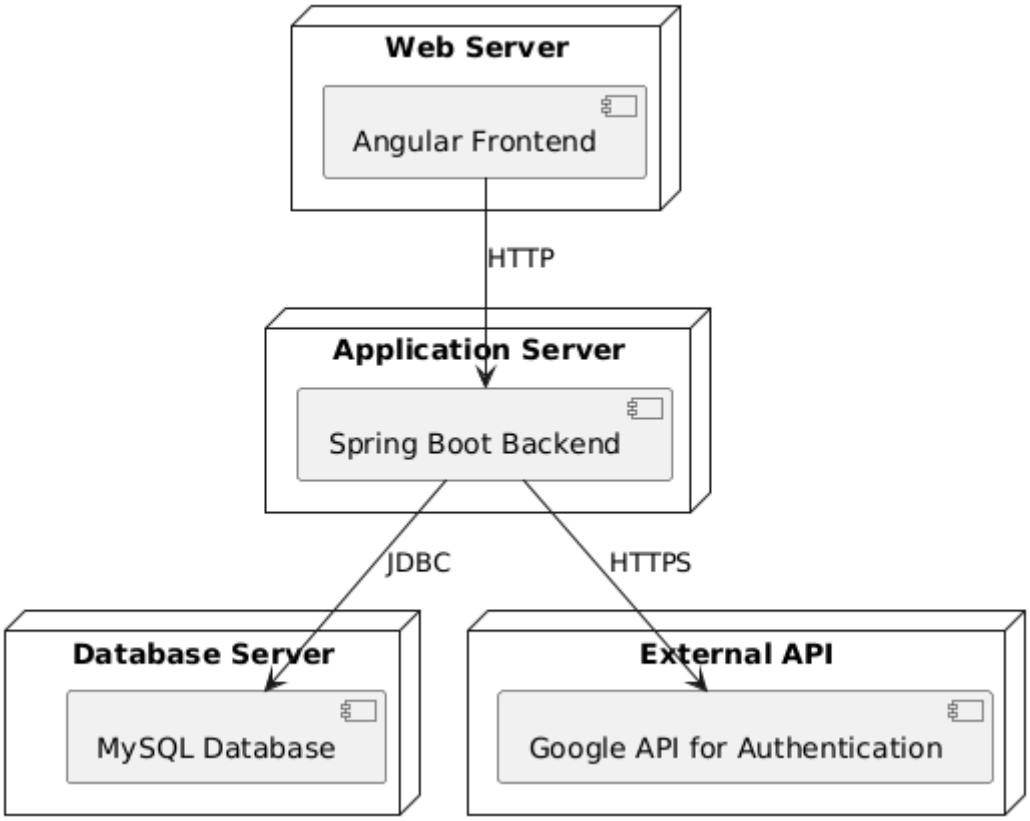
- Qué es: Es el servidor donde está alojada tanto la interfaz gráfica (Frontend) como la lógica de negocio (Backend). El Frontend está desarrollado en Angular, y el Backend en Spring Boot. Estos dos componentes trabajan juntos, pero se sirven desde el mismo servidor web.
- Función:
    - Frontend: Se encarga de gestionar la interfaz de usuario. Cuando un usuario interactúa con el sistema (por ejemplo, creando un nuevo proyecto o asignando un rol), esas acciones se envían al servidor como peticiones.
    - Backend: Se encarga de procesar las peticiones enviadas desde el frontend. Realiza las operaciones de negocio, como validar datos, gestionar usuarios, roles, proyectos, y luego devolver los resultados al frontend.
- Interacción:
    - El Frontend maneja la interacción con el usuario, mostrando información y capturando las entradas (formulario de login, lista de proyectos, etc.).
    - El Frontend se comunica con el Backend a través de una API REST, haciendo peticiones (por ejemplo, crear, leer, actualizar, o eliminar datos).
    - El Backend procesa estas peticiones, ejecuta la lógica de negocio, y luego envía las respuestas al Frontend en formato JSON, que se encarga de actualizar la interfaz del usuario.

Base de Datos (MySQL):

- Qué es: Es el servidor de base de datos donde se almacenan todos los datos persistentes

del sistema. Esto incluye información de usuarios, roles, proyectos, permisos, etc.

- Función: Almacena y gestiona la información relacionada con los proyectos, roles, usuarios y demás entidades. El Backend consulta esta base de datos cada vez que se necesita acceder a la información o realizar cambios en los datos (por ejemplo, cuando se añade un nuevo rol o se actualiza un proyecto).
- Interacción: El Backend se comunica directamente con la base de datos usando consultas SQL (usando Hibernate y JPA). Estas consultas son enviadas desde el Backend para recuperar o modificar los datos que se solicitan a través de las peticiones del cliente.



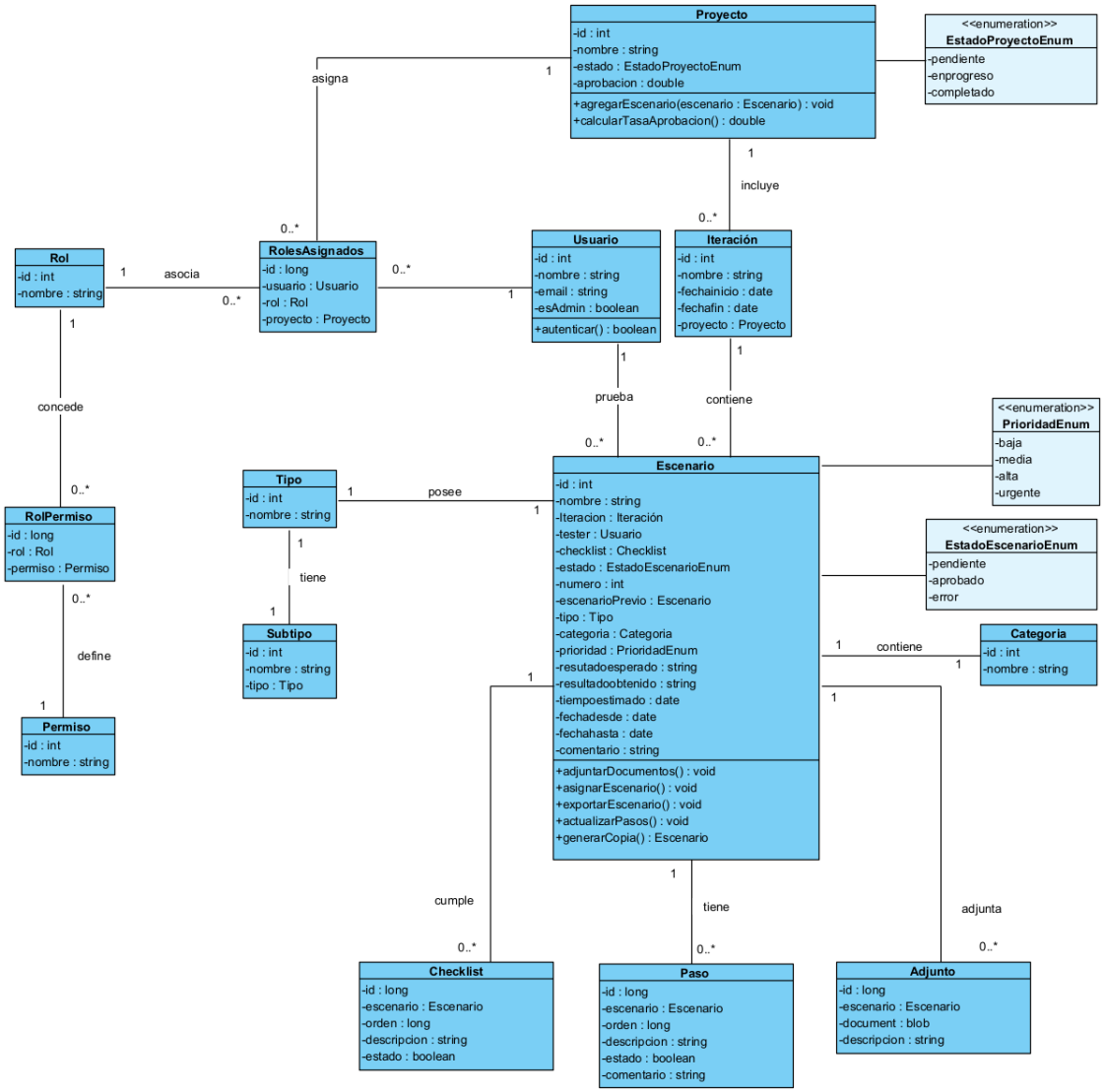
Flujo de interacción:

1. Cliente a Servidor Web: El usuario, a través del navegador web, interactúa con la interfaz de Angular en el Servidor Web. Las interacciones, como hacer clic en un botón o enviar un formulario, generan peticiones HTTP que se envían al Servidor Web.
2. Frontend a Backend: Una vez que el Frontend recibe la solicitud del cliente, genera una petición a la API REST en el Backend de Spring Boot. Esto puede ser, por ejemplo, una solicitud para ver la lista de proyectos asignados al usuario o modificar un caso de prueba.
3. Backend a Base de Datos: El Backend en Spring Boot procesa la solicitud, y si necesita acceder o modificar los datos, se comunica con la Base de Datos MySQL mediante consultas SQL (a través de Hibernate). Los datos recuperados de la base de datos son procesados y enviados de vuelta al Frontend en respuesta.
4. Backend a API: El Backend envía una solicitud HTTPS a la API de Google con los detalles

del usuario (por ejemplo, un token de autorización o un código de acceso). La Google API procesa la solicitud y responde con la información de autenticación del usuario, como un token de acceso válido o la confirmación de autenticación. El Backend recibe esta respuesta y la utiliza para continuar con el flujo de la aplicación, como permitir el acceso al sistema o asignar permisos de usuario basados en la autenticación. Si la autenticación es exitosa, el Backend puede permitir que el usuario continúe con sus operaciones, como la gestión de proyectos, pruebas, o roles dentro de Testify. Esta respuesta de la API se puede enviar de vuelta al Frontend como parte de la respuesta

completa al usuario.

Diagrama de Clases



Modelo Vista-Controlador