Informe de Pruebas

Capítulo 1: Pruebas Funcionales

Introducción

Este informe presenta los resultados de las pruebas funcionales y de rendimiento realizadas en el proyecto "Acme-SF" respecto al Student #1. Las pruebas funcionales verifican la efectividad en la detección de errores, mientras que las pruebas de rendimiento evalúan el tiempo de respuesta en diferentes computadoras. Los resultados ofrecen una visión sobre la estabilidad y eficiencia del proyecto.

Casos de Prueba Implementados para "Projects":

Característica 1: Listado de proyectos ("ManagerProjectListMineService.java")

- **Descripción:** Prueba la funcionalidad que permite a los gerentes listar sus propios proyectos almacenados en el sistema. Esta característica debe asegurarse de que solo se muestren los proyectos asociados con el manager autenticado.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 94,3%. Al ejecutar el "replayer" observamos que todas las líneas están en verde, por lo que se ha entrado y se ha ejecutado todo correctamente, de ahí el alto porcentaje.

Característica 2: Detalles de proyectos ("ManagerProjectShowService.java")

- **Descripción:** Prueba la funcionalidad que permite a los gerentes ver los detalles de sus propios proyectos almacenados en el sistema. Esta característica debe asegurarse de que solo se muestren los detalles de los proyectos asociados con el manager autenticado.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 93,6%. Al ejecutar el "replayer" observamos que hay algunas líneas en amarillo y en rojo:
    - Líneas amarillas:
        - **Línea 33:** `manager = project == null ? null : project.getManager();` La condición que siempre se cumple es que "project" siempre va a ser distinto a "null" por lo que siempre vamos a obtener un manager.
        - **Líneas 35 y 36:** `if (manager != null) status = super.getRequest().getPrincipal().hasRole(manager) && manager.getId() == managerRequestId;` Esta línea está en amarillo porque como manager siempre es distinto de null siempre va a entrar por este if y nunca por el else.

        - **Línea 56:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.
    - Líneas rojas:
        - **Línea 38:** `status = false;` Esta línea no se ejecuta debido a que nunca el manager va a ser null.

Característica 3: Crear proyecto ("ManagerProjectCreateService.java")

- **Descripción:** Prueba la funcionalidad que permite a los gerentes crear nuevos proyectos en el sistema. Esta característica debe asegurarse de que solo los gerentes autenticados puedan crear proyectos, que cuando se crea el proyecto no se ponga un código que tenga otro proyecto ya creado, que el título esté entre 0 y 75 caracteres, que el projectAbstract esté entre 0 y 100 caracteres, que el coste esté entre 0 y 10000 y que el link tenga longitud máxima 255 caracteres y formato URL y que los proyectos creados se asocien correctamente con el gerente autenticado. Todos los atributos son obligatorios menos el link.

- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 90,1%. Al ejecutar el "replayer" observamos que tenemos todas las líneas verdes menos todos los asserts, los cuales están en amarillo y siempre están en amarillo debido a que siempre son distintos de null.

Característica 4: Actualizar proyectos ("ManagerProjectUpdateService.java")

- **Descripción:** Prueba la funcionalidad que permite a los gerentes actualizar los detalles de sus propios proyectos almacenados en el sistema. Esta característica debe asegurarse de que solo los proyectos asociados con el gerente autenticado puedan ser actualizados por él, que cuando se actualice el proyecto no se ponga un código que tenga otro proyecto ya creado, que el título esté entre 0 y 75 caracteres, que el projectAbstract esté entre 0 y 100 caracteres, que el coste esté entre 0 y 10000 y que el link tenga longitud máxima 255 caracteres y formato URL y que los cambios se guarden correctamente en el sistema. Todos los atributos son obligatorios menos el link.

- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 90,7%. Al ejecutar el "replayer" observamos que hay algunas líneas en amarillo.
    - **Línea 32:** `manager = project == null ? null : project.getManager();` La condición que siempre se cumple es que "project" siempre va a ser distinto a "null" por lo que siempre vamos a obtener un manager.
    - **Línea 33:** `status = project != null && !project.isPublished() && super.getRequest().getPrincipal().hasRole(manager);` Esta línea está en amarillo debido a que como siempre el proyecto va a ser distinto de null, el proyecto no va a estar publidado debido a que sino no se podría actualizar el proyecto y siempre va a tener la persona el rol de manager pues status va a ser siempre true.
    - **Líneas 51,59,71,78:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.

- **Línea 65:** `super.state(existing == null || existing.equals(object), "code", "manager.project.form.error.duplicated");` En esta línea, el valor de "existing" siempre va a ser distinto de null (ya que siempre vamos a buscar un proyecto existente), por eso está en amarillo.

Característica 5: Publicar proyecto ("ManagerProjectPublishService.java")

- **Descripción:** Prueba la funcionalidad que permite a los gerentes publicar proyectos en el sistema. Esta característica debe asegurarse de que solo los gerentes autenticados puedan publicar proyectos, que el proyecto que se quiere publicar no esté ya publicado, que no tenga errores fatales, que cuando se actualice el proyecto no se ponga un código que tenga otro proyecto ya creado, que el título esté entre 0 y 75 caracteres, que el projectAbstract esté entre 0 y 100 caracteres, que el coste esté entre 0 y 10000 y que el link tenga longitud máxima 255 caracteres y formato URL, que además tenga al menos una historia de usuario asociada, que todas sus historias de usuario estén publicadas y que los cambios se guarden correctamente en el sistema. Todos los atributos son obligatorios menos el link.

- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 92%. Al ejecutar el "replayer" observamos que hay algunas líneas en amarillo.
  - **Línea 33:** `status = project != null && !project.isPublished() && super.getRequest().getPrincipal().hasRole(manager);` Esta línea está en amarillo debido a que como siempre el proyecto va a ser distinto de null, el proyecto no va a estar publidado debido a que sino no se podría actualizar el proyecto y siempre va a tener la persona el rol de manager pues status va a ser siempre true.
  - **Líneas 65,72,92,100:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.
  - **Línea 74:** `if (!super.getBuffer().getErrors().hasErrors("code")):` Esta línea está en amarillo debido a que no hay else y siempre o entra o no entra en el if.
  - **Línea 76:** `super.state(existing == null || existing.equals(object), "code", "manager.project.form.error.duplicated"); :` En esta línea, el valor de "existing" siempre va a ser distinto de null (ya que siempre vamos a buscar un proyecto existente), por eso está en amarillo.

Característica 6: Borrar proyecto ("ManagerProjectDeleteService.java")

- **Descripción:** Prueba la funcionalidad que permite a los gerentes eliminar proyectos del sistema. Esta característica debe asegurarse de que solo los gerentes autenticados puedan eliminar proyectos y que el proyecto que se quiere eliminar esté asociado con el gerente autenticado.

- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 65,7%. Al ejecutar el "replayer" observamos que hay algunas líneas en amarillo y en rojo.

  - Líneas amarillas:

- **Líneas 46 y 47:** `status = project != null && !project.isPublished() && super.getRequest().getPrincipal().hasRole(manager);` Esta línea está en amarillo debido a que como siempre el proyecto va a ser distinto de null, el proyecto no va a estar publidado debido a que sino no se podría actualizar el proyecto y siempre va a tener la persona el rol de manager pues status va a ser siempre true.

- **Líneas 65,72,77:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.

- Líneas rojas: se corresponden al método "unbind", este método no se ejecuta debido a que no se tiene que ejecutar, ya que sirve para mostrar los datos devuelve un error y en el delete no hay errores que devolver por tanto no se cargan los datos.

Casos de Prueba Implementados para "UserStory":

Característica 1: Listado de historias de usuario ("ManagerUserStoryListService.java")

- **Descripción:** Prueba la funcionalidad que permite a los gerentes listar sus propias historias de usuario almacenadas en el sistema. Esta característica debe asegurar que solo se muestren las historias de usuario asociadas con el gerente autenticado.

- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 90,8%. Al ejecutar el "replayer" observamos que hay algunas líneas en amarillo.

  - Líneas amarillas:

    - **Línea 36:** `manager = project == null ? null : project.getManager();` La condición que siempre se cumple es que "project" siempre va a ser distinto a "null" por lo que siempre vamos a obtener un manager.
    - **Líneas 38 y 39:** `if (manager != null) status = super.getRequest().getPrincipal().hasRole(manager) && manager.getId() == managerRequestId;` Esta línea está en amarillo porque como manager siempre es distinto de null siempre va a entrar por este if y nunca por el else.
    - **Líneas 59 y 70:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.
    - **Línea 78:** `showAdd = !project.isPublished() && super.getRequest().getPrincipal().hasRole(project.getManager());` Esta línea está en amarillo debido a que, llegados a este punto de la función, el rol siempre va a ser manager por lo que esa condición siempre será true.

  - Líneas rojas:

    - **Línea 41:** `status = false;` Esta línea no se ejecuta debido a que nunca el manager va a ser null.

Característica 2: Detalles de proyectos ("ManagerUserStoryShowService.java")

- **Descripción:** Prueba la funcionalidad que permite a los gerentes ver los detalles de sus propios proyectos almacenados en el sistema. Esta característica debe asegurarse de que solo se muestren los detalles de los proyectos asociados con el manager autenticado.

- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 93,9%. Al ejecutar el "replayer" observamos que hay algunas líneas en amarillo y en rojo:

    - Líneas amarillas:

        - **Línea 35:** `manager = userStory == null ? null : userStory.getManager();` La condición que siempre se cumple es que "userStory" siempre va a ser distinto a "null" por lo que siempre vamos a obtener un manager.

        - **Líneas 37 y 38:** `if (manager != null) status = super.getRequest().getPrincipal().hasRole(manager) && manager.getId() == managerRequestId;` Esta línea está en amarillo porque como manager siempre es distinto de null siempre va a entrar por este if y nunca por el else.

        - **Línea 58** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.

    - Líneas rojas:
        - **Línea 40:** `status = false;` Esta línea no se ejecuta debido a que nunca el manager va a ser null.

Característica 3: Crear historia de usuario ("ManagerUserStoryCreateService.java")

- **Descripción:** Prueba la funcionalidad que permite a los gerentes crear nuevas historias de usuario en el sistema. Esta característica debe asegurarse de que solo los gerentes autenticados puedan crear historias de usuario, que el título esté entre 0 y 75 caracteres, que la descripción esté entre 0 y 100 caracteres, que el coste estimado esté entre 1 y 10000, que los criterios de aceptación estén entre 0 y 100 caracteres, que el link tenga longitud máxima 255 caracteres y formato URL y que las historias de usuario creadas se asocien correctamente con el gerente autenticado. Todos los atributos son obligatorios menos el link.

- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 89,8%. Al ejecutar el "replayer" observamos que hay algunas líneas en amarillo:
    - **Líneas 45,52,57,64:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.

Característica 4: Actualizar historias de usuario ("ManagerUserStoryUpdateService.java")

- **Descripción:** Prueba la funcionalidad que permite a los gerentes actualizar los detalles de sus propias historias de usuario almacenadas en el sistema. Esta característica debe asegurarse de que solo las historias de usuario asociadas con el gerente autenticado puedan ser actualizadas por él, que el título tenga entre 0 y 75 caracteres, que la descripción esté entre 0 y 100 caracteres, que el coste esté entre 1 y 10000, y que el enlace tenga una longitud máxima de 255 caracteres y

formato URL. Además, los cambios deben guardarse correctamente en el sistema. Todos los atributos son obligatorios, excepto el enlace.

- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 89,9%. Al ejecutar el "replayer" observamos que hay algunas líneas en amarillo.
    - Líneas amarillas:
        - **Línea 35:** `manager = userStory == null ? null : userStory.getManager();` La condición que siempre se cumple es que "userStory" siempre va a ser distinto a "null" por lo que siempre vamos a obtener un manager.
        - **Línea 37:** `if (manager != null)` Esta línea está en amarillo porque como manager siempre es distinto de null siempre va a entrar por este if y nunca por el else.
        - **Línea 39:** `&& manager.getId() == managerRequestId;` Esta línea no debería estar en amarillo ya que he probado a realizar una petición update sobre una historia de usuario de un manager con otro manager.
        - **Líneas 59,66,71,78:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.
    - Líneas rojas:
        - **Línea 41:** `status = false;` Esta línea no se ejecuta debido a que nunca el manager va a ser null.

Característica 5: Publicar historia de usuario ("ManagerUserStoryPublishService.java")

- **Descripción:** Prueba la funcionalidad que permite a los gerentes publicar historias de usuario en el sistema. Esta característica debe asegurarse de que solo los gerentes autenticados puedan publicar historias de usuario, que la historia de usuario que se quiere publicar no esté ya publicada, que el título tenga entre 0 y 75 caracteres, que la descripción esté entre 0 y 100 caracteres, que el coste esté entre 1 y 10000 y que el enlace tenga una longitud máxima de 255 caracteres y formato URL. Además, los cambios deben guardarse correctamente en el sistema.
- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 90,1%. Al ejecutar el "replayer" observamos que hay algunas líneas en amarillo y en rojo:
    - Líneas amarillas:
        - **Línea 35:** `manager = userStory == null ? null : userStory.getManager();` La condición que siempre se cumple es que "userStory" siempre va a ser distinto a "null" por lo que siempre vamos a obtener un manager.
        - **Línea 37:** `if (manager != null)` Esta línea está en amarillo porque como manager siempre es distinto de null siempre va a entrar por este if y nunca por el else.

- **Línea 39:** `&& manager.getId() == managerRequestId;` Esta línea no debería estar en amarillo ya que he probado a realizar una petición publish sobre una historia de usuario de un manager con otro manager.
- **Líneas 59,66,71,79:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.

- Líneas rojas:

- **Línea 41:** `status = false;` Esta línea no se ejecuta debido a que nunca el manager va a ser null.

Característica 6: Borrar historia de usuario ("ManagerUserStoryDeleteService.java")

- **Descripción:** Prueba la funcionalidad que permite a los gerentes eliminar proyectos del sistema. Esta característica debe asegurarse de que solo los gerentes autenticados puedan eliminar proyectos y que el proyecto que se quiere eliminar esté asociado con el gerente autenticado.

- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 60,1%. Al ejecutar el "replayer" observamos que hay algunas líneas en amarillo y en rojo.

  - Líneas amarillas:

    - **Línea 38:** `manager = userStory == null ? null : userStory.getManager();` La condición que siempre se cumple es que "userStory" siempre va a ser distinto a "null" por lo que siempre vamos a obtener un manager.
    - **Línea 40:** `if (manager != null)` Esta línea está en amarillo porque como manager siempre es distinto de null siempre va a entrar por este if y nunca por el else.
    - **Línea 42:** `&& manager.getId() == managerRequestId;` Esta línea no debería estar en amarillo ya que he probado a realizar una petición delete sobre una historia de usuario de un manager con otro manager.
    - **Líneas 62,69,74:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.

  - Líneas rojas:

    - **Línea 44:** `status = false;` Esta línea no se ejecuta debido a que nunca el manager va a ser null.
    - **Método unbind:** Este método no se ejecuta debido a que no se tiene que ejecutar, ya que sirve para mostrar los datos devuelve un error y en el delete no hay errores que devolver por tanto no se cargan los datos.

Casos de Prueba Implementados para "ProjectUserStory":

Característica 1: Crear ProjectUserStory ("ManagerProjectUserStoryCreateService.java")

- **Descripción:** Prueba la funcionalidad que permite crear la relación entre historias de usuario y proyectos. Esta característica debe asegurarse de que solo se muestren, para la asociación a un proyecto determinado, las historias de usuario asociadas con el gerente autenticado. Además, no se podrán crear relaciones para proyectos que ya estén publicados.

- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 86,2%.

    - Líneas amarillas:

        - **Línea 38:** `manager = project == null ? null : userStory.getManager();` La condición que siempre se cumple es que "project" siempre va a ser distinto a "null" por lo que siempre vamos a obtener un manager.
        - **Línea 40:** `if (manager != null)` Esta línea está en amarillo debido a que el manager siempre va a ser distinto de null.
        - **Línea 42:** `&& manager.getId() == managerRequestId;` Esta línea está en amarillo porque no hay else, por lo que si entra en if lo ejecuta entero.
        - **Líneas 65,78,95,102:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.
        - **Línea 80:** `if (!super.getBuffer().getErrors().hasErrors("project"))` Esta línea está en amarillo porque como manager siempre es distinto de null siempre va a entrar por este if y nunca por el else.
        - **Línea 81:** `super.state(!object.getProject().isPublished(), "*", "manager.projectUserStory.form.error.published");` Esta línea está en amarillo porque como manager siempre es distinto de null siempre va a entrar por este if y nunca por el else.
        - **Línea 82:** `if (!super.getBuffer().getErrors().hasErrors("*") && !super.getBuffer().getErrors().hasErrors("userStory"))` Esta línea está en amarillo porque como manager siempre es distinto de null siempre va a entrar por este if y nunca por el else.
        - **Línea 84:** `super.state(existing == null, "*", "manager.projectUserStory.form.error.duplicatedRelation");` Esta línea está en amarillo porque como manager siempre es distinto de null siempre va a entrar por este if y nunca por el else.
        - **Línea 86:** `if (!super.getBuffer().getErrors().hasErrors("project") && !super.getBuffer().getErrors().hasErrors("userStory"))` Esta línea está en amarillo porque como manager siempre es distinto de null siempre va a entrar por este if y nunca por el else.
        - **Línea 89:** `super.state(projectManager.getId() == userStoryManager.getId(), "*", "manager.projectUserStory.form.error.sameManager");` Esta línea está en amarillo porque como manager siempre es distinto de null siempre va a entrar por este if y nunca por el else.
        - **Línea 128:** `if (object.getUserStory() == null)` : Esta línea está en amarillo ya que siempre entra por este if, y esto es porque siempre "object.getUserStory" es igual a null.

- **Línea 134:** `if (object.getUserStory() != null && object.getUserStory().getId() == us.getId())` Esta línea está en amarillo debido a que como "object.getUserStory" es siempre null, esta condición no se va a cumplir y por tanto nunca entra por el if y siempre entrará por el else.

- Líneas rojas:

  - **Línea 44:** `status = false;` Esta línea no se ejecuta debido a que nunca el manager va a ser null.
  - **Línea 131:** `choices.add("0", "-----", false);` Como "object.getUserStory" es siempre null, esta condición no se va a cumplir y por tanto nunca entra por el if y siempre entrará por el else.
  - **Línea 135:** `choices.add(Integer.toString(us.getId()), us.getTitle() + " - " + Integer.toString(us.getEstimatedCost()) + " - " + us.getPriority(), true);` Como "object.getUserStory" es siempre null, esta condición no se va a cumplir y por tanto nunca entra por el if y siempre entrará por el else.

Característica 2: Borrar ProjectUserStory ("ManagerProjectUserStoryDeleteService.java")

- **Descripción:** Prueba la funcionalidad que permite borrar la relación entre historias de usuario y proyectos. Esta característica debe asegurar que solo se puedan borrar aquellas historias que se correspondan con el manager autenticado y que están asociadas a un proyecto no publicado.

- **Efectividad:** En cuanto a la cobertura, observamos que tenemos un valor de 85,6%.

  - Líneas amarillas:

    - **Línea 38:** `manager = project == null ? null : userStory.getManager();` La condición que siempre se cumple es que "project" siempre va a ser distinto a "null" por lo que siempre vamos a obtener un manager.

    - **Línea 40:** `if (manager != null)` Esta línea está en amarillo porque como manager siempre es distinto de null siempre va a entrar por este if y nunca por el else.

    - **Línea 42:** `&& manager.getId() == managerRequestId;` Esta línea no debería estar en amarillo ya que he probado a realizar una petición show sobre un projectUserStory de un manager con otro manager.

    - **Líneas 65,78,94,103:** `assert object != null;` La condición que siempre se cumple es que "object" siempre va a ser distinto de null.

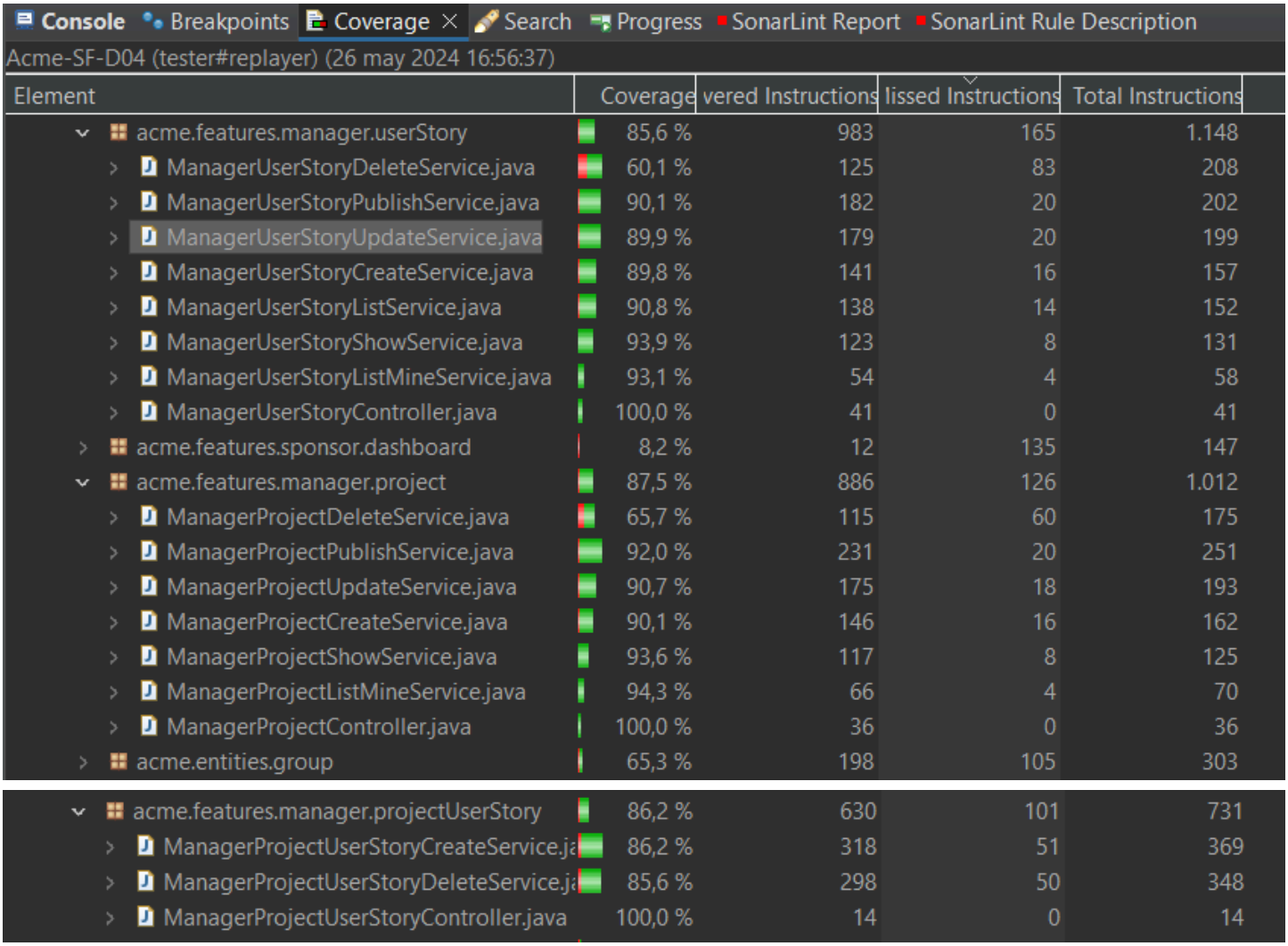
    - **Línea 80:** `if (!super.getBuffer().getErrors().hasErrors("userStory"))` Esta línea está en amarillo debido a que no hay else y por tanto siempre va a entrar en el if.

    - **Línea 83:** `if (!super.getBuffer().getErrors().hasErrors("project"))` Esta línea está en amarillo debido a que no hay else y por tanto siempre va a entrar en el if.

- **Línea 84:** `super.state(!object.getProject().isPublished(), "*", "manager.projectUserStory.form.error.published");` Esta línea está en amarillo debido a que no hay else y por tanto siempre va a entrar en el if.
- **Línea 86:** `if (!super.getBuffer().getErrors().hasErrors("*") && !super.getBuffer().getErrors().hasErrors("userStory"))` Esta línea está en amarillo debido a que no hay else y por tanto siempre va a entrar en el if.
- **Línea 88:** `super.state(existing != null, "*", "manager.projectUserStory.form.error.mustBeDuplicated");` Esta línea está en amarillo porque "existing" siempre va a ser distinto de null.
- **Línea 124:** `if (object.getUserStory() == null)` : ``: Esta línea está en amarillo ya que siempre entra por este if, y esto es porque siempre "object.getUserStory" es igual a null.
- **Línea 130:** `if (object.getUserStory() != null && object.getUserStory().getId() == us.getId())` Esta línea está en amarillo debido a que como "object.getUserStory" es siempre null, esta condición no se va a cumplir y por tanto nunca entra por el if y siempre entrará por el else.

- Líneas rojas:
  - **Línea 44:** `status = false;` Esta línea no se ejecuta debido a que nunca el manager va a ser null.
  - **Línea 127:** `choices.add("0", "-----", false);` Esta línea está en rojo debido a que como "object.getUserStory" es siempre null, esta condición siempre se va a cumplir y por tanto nunca entra por el else.
  - **Línea 131:** `choices.add(Integer.toString(us.getId()), us.getTitle() + " - " + Integer.toString(us.getEstimatedCost()) + " - " + us.getPriority(), true);` Esta línea está en rojo debido a que como "object.getUserStory" es siempre null, esta condición no se va a cumplir nunca y siempre se va a meter en el else.

A continuación proporciono unas capturas de la cobertura de mis pruebas:



Console | Breakpoints | Coverage | Search | Progress | SonarLint Report | SonarLint Rule Description

Acme-SF-D04 (tester#replayer) (26 may 2024 16:56:37)

Element	Coverage	vered Instructions	lissed Instructions	Total Instructions
acme.features.manager.userStory	85,6 %	983	165	1.148
ManagerUserStoryDeleteService.java	60,1 %	125	83	208
ManagerUserStoryPublishService.java	90,1 %	182	20	202
ManagerUserStoryUpdateService.java	89,9 %	179	20	199
ManagerUserStoryCreateService.java	89,8 %	141	16	157
ManagerUserStoryListService.java	90,8 %	138	14	152
ManagerUserStoryShowService.java	93,9 %	123	8	131
ManagerUserStoryListMineService.java	93,1 %	54	4	58
ManagerUserStoryController.java	100,0 %	41	0	41
acme.features.sponsor.dashboard	8,2 %	12	135	147
acme.features.manager.project	87,5 %	886	126	1.012
ManagerProjectDeleteService.java	65,7 %	115	60	175
ManagerProjectPublishService.java	92,0 %	231	20	251
ManagerProjectUpdateService.java	90,7 %	175	18	193
ManagerProjectCreateService.java	90,1 %	146	16	162
ManagerProjectShowService.java	93,6 %	117	8	125
ManagerProjectListMineService.java	94,3 %	66	4	70
ManagerProjectController.java	100,0 %	36	0	36
acme.entities.group	65,3 %	198	105	303
acme.features.manager.projectUserStory	86,2 %	630	101	731
ManagerProjectUserStoryCreateService.ja	86,2 %	318	51	369
ManagerProjectUserStoryDeleteService.ja	85,6 %	298	50	348
ManagerProjectUserStoryController.java	100,0 %	14	0	14

Capítulo 2: Pruebas de Rendimiento

Introducción

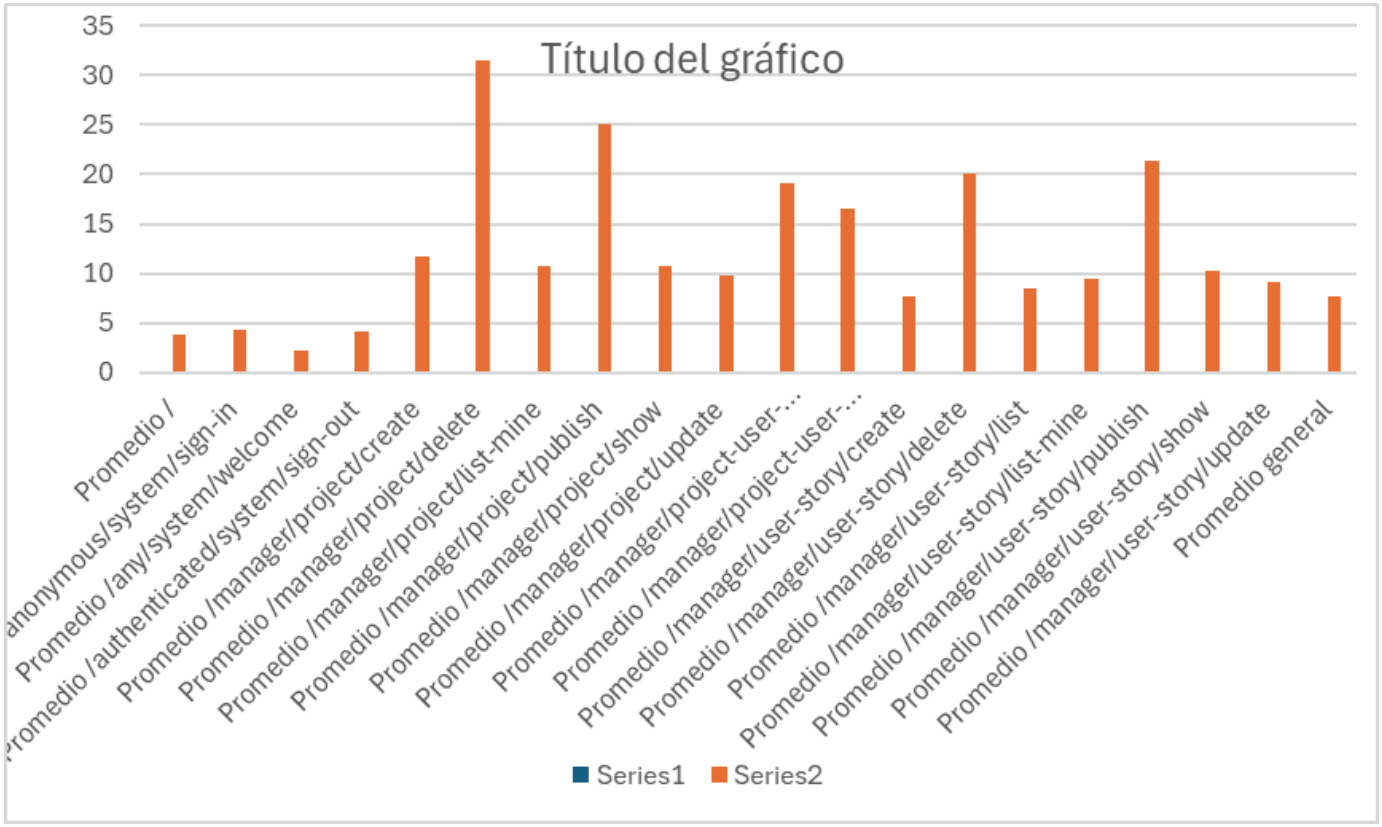
Este capítulo analiza las pruebas de rendimiento realizadas en el proyecto "Acme-SF" relacionadas con las características del manager. El objetivo es evaluar el tiempo de respuesta y la capacidad del sistema bajo diferentes cargas. Utilizamos una metodología que incluye la simulación de escenarios reales y el análisis estadístico de los resultados para garantizar la eficiencia y estabilidad del sistema.

Resultados de las Pruebas de Rendimiento

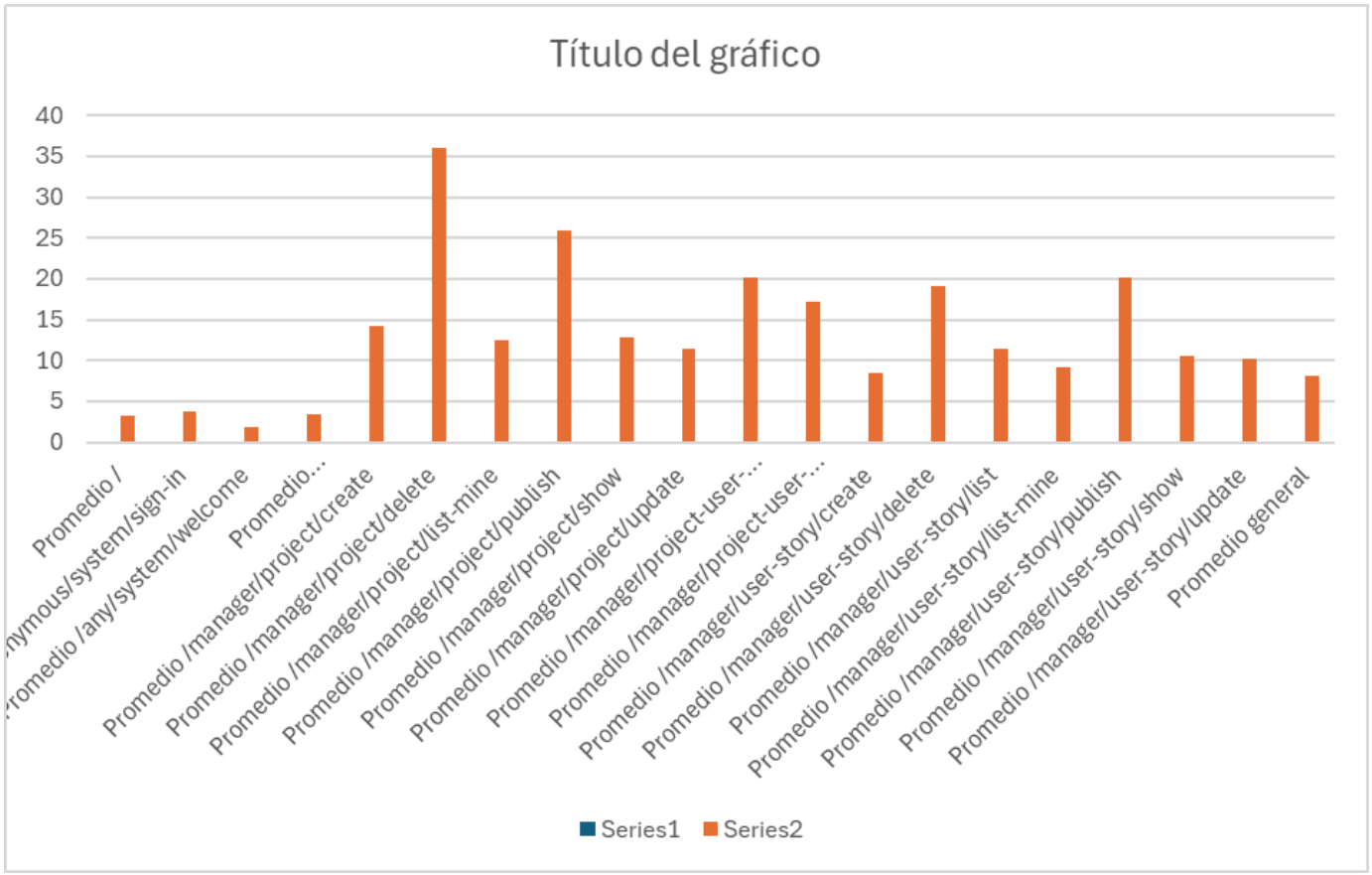
Gráficos

Gráfico del tiempo para la ejecución de las características del gerente sin la optimización en PC1.

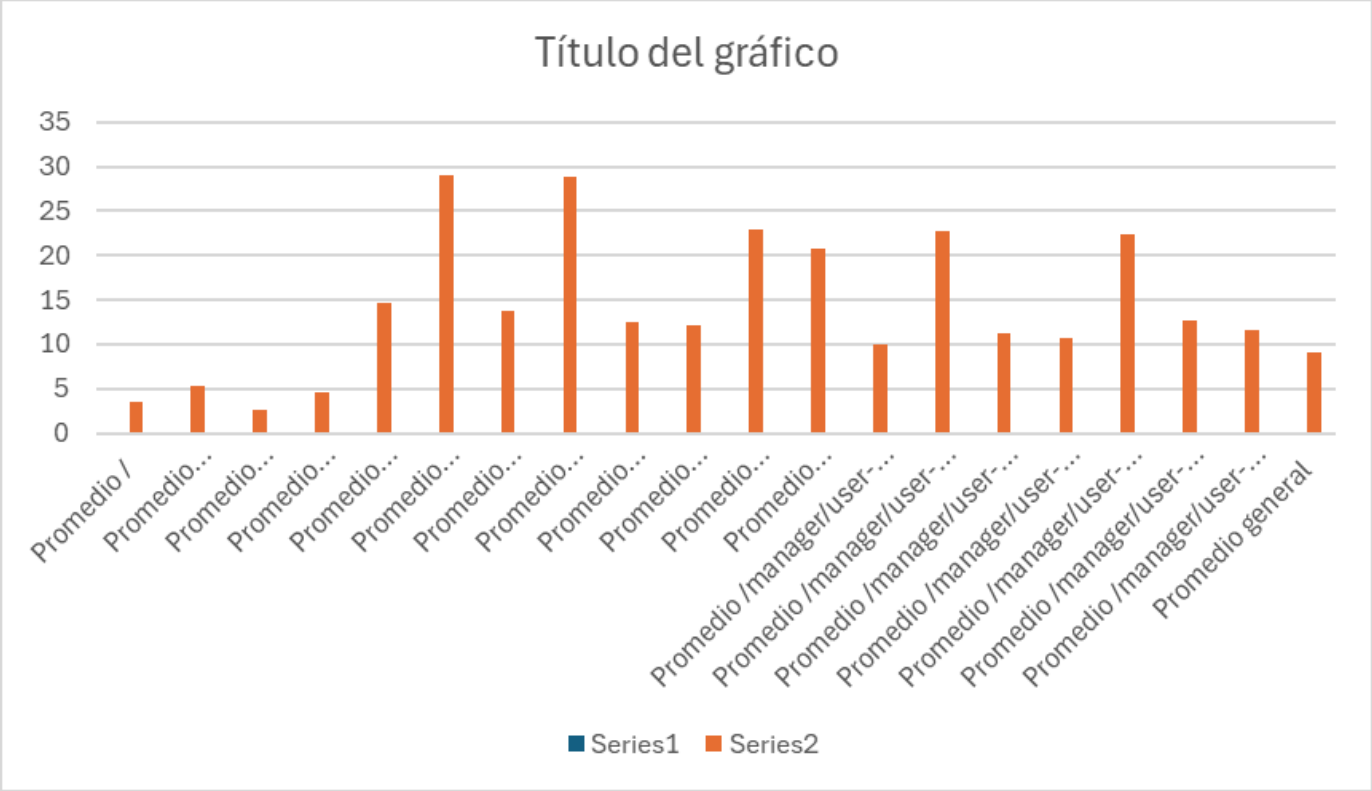
- **Gráfico 1:** Gráfico del tiempo para la ejecución de las características del gerente sin la optimización en PC1.



- **Gráfico 2:** Gráfico del tiempo para la ejecución de las características del gerente con la optimización en PC1.



- **Gráfico 3:** Gráfico del tiempo para la ejecución de las características del gerente con la optimización en PC2.



Intervalo de Confianza

Presentar el intervalo de confianza del 95% para el tiempo de respuesta en las dos computadoras.

- **PC1:**
  - **Nivel de Confianza sin la optimización:** 0,564382343
  - **Nivel de Confianza con la optimización:** 0,642458422
- **PC2:**
  - **Nivel de Confianza con la optimización:** 0,591105843

Contraste de Hipótesis

Presentar el contraste de hipótesis con un 95% de confianza sobre cuál computadora es más potente.

- **Resultado del Contraste de Hipótesis:**
  - Si observamos el "p-value" para "Before optimizaation with PC1" y "After optimization with PC1", su valor es de 0.205641004, como está contenido en el intervalo (0.05,1.00] entonces los cambios no dieron como resultado ninguna mejora significativa; los tiempos de muestra son diferentes, pero son globalmente iguales. De hecho, el tiempo de ejecución medio es mayor después de optimizar, esto es debido a que la optimización que hemos hecho tiene complejidad logaritmica respecto la complejidad lineal que tenemos antes de optimizar, al tener tan pocos datos nos empeora el rendimiento, por lo que esta optimización es ideal para tener una mayor cantidad de datos.
  - Si observamos el "p-value" para "After optimization with PC1" y "After optimization with PC2", su valor es de 0,033174903, como está contenido en el intervalo [0.00,0.05) y

observando los tiempos medios del PC1 y PC2, vemos que ha tenido una mejora de rendimiento en el PC2.