Testing Report

Diseño y pruebas II

Sprint 4

Versión 1.0

Fecha 20/05/2024

Preparado por:
Miguel Hernández Sánchez (C1.028)

Repositorio:
DP2-c1-028/Acme-SF-D04 (github.com)

Índice

Índice **2**
1. Functional testing **4**
1.1. Operaciones del gestor en proyectos 4
Listado de proyectos 4
Coverage 4
Analyse 5
Detalles de un proyecto 5
Coverage 5
Analyse 6
Crear un proyecto 6
Coverage 6
Analyse 7
Actualizar un proyecto 7
Coverage 8
Analyse 9
Publicar proyectos 10
Coverage 10
Analyse 12
Borrar proyecto 12
Coverage 12
Analyse 13
1.2. Operaciones del gestor en las historias de usuario 13
Listado de las historias de usuario 13
Coverage 14
Analyse 15
Detalles de una historia de usuario 15
Coverage 15
Analyse 16
Crear una historia de usuario 16
Coverage 17
Analyse 18
Actualizar una historia de usuario 18
Coverage 19
Analyse 20
Publicar una historia de usuario 20
Coverage 20
Analyse 21
Borrar una historia de usuario 21
Coverage 22
Analyse 23
1.3. Operaciones del gestor en la entidad intermedia entre los proyectos y las historias

de usuario 23
Listado de la entidad intermedia 23
Coverage 23
Analyse 24
Detalles de la entidad intermedia 24
Coverage 24
Analyse 25
Crear entidad intermedia 25
Coverage 26
Analyse 27
Actualizar entidad intermedia 27
Coverage 28
Analyse 29
Borrar entidad intermedia 30
Coverage 30
Analyse 31
2. Performance testing **31**
Estadísticas de rendimiento desde mi equipo 31
Comparativa entre equipos 33
Comparativa con índices 35

1. Functional testing

1.1. Operaciones del gestor en proyectos

Listado de proyectos

Para el listado del proyecto se realizaron dos archivos de testing: *list.safe* y *list.hack*. En el archivo .safe se probó listar los proyectos de los usuarios con rol de gestor para cubrir que todos los elementos de los datos de prueba se mostrasen en la pantalla. En el archivo .hack se probó realizar la funcionalidad con otro rol.

Coverage

```
@Service
public class ManagerProjectListService extends AbstractService<Manager, Project> {

    // Internal state ---------------------------------------------------------

    @Autowired
    private ManagerProjectRepository repository;

    // AbstractService interface ----------------------------------------------


    @Override
    public void authorise() {
        super.getResponse().setAuthorised(true);
    }

    @Override
    public void load() {
        Collection<Project> objects;
        int managerId;

        managerId = super.getRequest().getPrincipal().getActiveRoleId();

        objects = this.repository.findProjectsByManagerId(managerId);

        super.getBuffer().addData(objects);
    }

    @Override
    public void unbind(final Project object) {
        assert object != null;

        Dataset dataset;

        dataset = super.unbind(object, "title", "code", "abstractText", "cost", "link");

        super.getResponse().addData(dataset);
    }

}
```

Como se puede observar, el coverage es totalmente completo exceptuando un assert el cual no podemos probar porque es una situación del framework el cual no podemos controlar.

Analyse

En el análisis se puede observar, se prueban todos los datos de pruebas existentes para los proyectos y por lo tanto se podría decir que son completos.

Detalles de un proyecto

Para los detalles de un proyecto se realizaron dos archivos de testing: *show.safe* y *show.hack*. En el archivo .safe se probó mostrar los datos de todos los proyectos de los datos de prueba y en el archivo .hack se intentó acceder con un rol inadecuado y con otro usuario del mismo rol realizando GET hacking.

Coverage

```
@Service
public class ManagerProjectShowService extends AbstractService<Manager, Project> {

    // Internal state ---------------------------------------------------------

    @Autowired
    private ManagerProjectRepository repository;

    // AbstractService interface ----------------------------------------------


    @Override
    public void authorise() {
        boolean status;
        int id;
        int managerId;
        Project project;

        id = super.getRequest().getData("id", int.class);
        project = this.repository.findOneProjectById(id);

        managerId = super.getRequest().getPrincipal().getActiveRoleId();

        status = managerId == project.getManager().getId();

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        Project object;
        int id;

        id = super.getRequest().getData("id", int.class);
        object = this.repository.findOneProjectById(id);

        super.getBuffer().addData(object);
    }

    @Override
    public void unbind(final Project object) {
        assert object != null;

        Dataset dataset;

        dataset = super.unbind(object, "title", "code", "abstractText", "cost", "link", "draftMode");

        super.getResponse().addData(dataset);
    }
}
```

Como se puede observar, se ha realizado un buen testeo ya que la única línea en la cual no se han recorrido todos los casos es en la que no podemos ya que es un caso que no controlamos nosotros.

Analyse

En el análisis se puede observar que se han mostrado todos los datos del archivo de datos de prueba.

Crear un proyecto

Para la creación de un proyecto se realizaron dos archivos de testing: *create.safe* y *create.hack*. En el archivo .safe se probaron todos los casos negativos según la metodología dada en clase y posteriormente se procedió a probar los datos positivos usando el rango mínimo y máximo de los valores que se podían tomar. En el archivo .hack se intentó acceder con un rol inadecuado y a crear proyectos que incluían SQL y HTML injection dando como resultado que no se ejecutaron las instrucciones maliciosas y por lo tanto sacando como conclusión de que el código no es vulnerable.

Coverage

```java
@Service
public class ManagerProjectCreateService extends AbstractService<Manager, Project> {
    // Internal state ---------------------------------------------------------

    @Autowired
    private ManagerProjectRepository        repository;

    @Autowired
    private SystemConfigurationRepository   systemConfigurationRepository;

    // AbstractService interface ----------------------------------------------


    @Override
    public void authorise() {
        super.getResponse().setAuthorised(true);
    }

    @Override
    public void load() {
        Project object;

        object = new Project();
        Integer managerId = super.getRequest().getPrincipal().getActiveRoleId();
        Manager manager = this.repository.findOneManagerById(managerId);
        object.setManager(manager);
        object.setDraftMode(true);
        object.setHasFatalError(false);

        super.getBuffer().addData(object);
    }

    @Override
    public void bind(final Project object) {
        assert object != null;

        super.bind(object, "title", "code", "abstractText", "cost", "link");
    }
```

```
    @Override
    public void validate(final Project object) {
        assert object != null;

        if (!super.getBuffer().getErrors().hasErrors("code")) {

            Project projectSameCode = this.repository.findOneProjectByCode(object.getCode());

            super.state(projectSameCode == null, "code", "manager.project.form.error.code");
        }

        if (!super.getBuffer().getErrors().hasErrors("cost") && object.getCost() != null)
            super.state(object.getCost().getAmount() >= 0, "cost", "manager.project.form.error.cost-negative");

        if (!super.getBuffer().getErrors().hasErrors("cost"))
            super.state(object.getCost() != null, "cost", "manager.project.form.error.cost-null");

        if (!super.getBuffer().getErrors().hasErrors("cost") && object.getCost() != null) {
            String symbol = object.getCost().getCurrency();
            boolean existsCurrency = this.systemConfigurationRepository.existsCurrency(symbol);
            super.state(existsCurrency, "cost", "manager.project.form.error.not-valid-currency");
        }

        if (!super.getBuffer().getErrors().hasErrors("cost") && object.getCost() != null)
            super.state(object.getCost().getAmount() <= 1000000, "cost", "manager.project.form.error.not-valid-currency");

    }

    @Override
    public void perform(final Project object) {
        assert object != null;

        this.repository.save(object);
    }

    @Override
    public void unbind(final Project object) {
        assert object != null;

        Dataset dataset;

        dataset = super.unbind(object, "title", "code", "abstractText", "cost", "link");

        super.getResponse().addData(dataset);
    }
}
```

Como podemos observar, ignorando las líneas con *assert* las cuales no podemos probar, en el validador se encuentran algunos *if* los cuales están amarillos indicando que no han pasado por todos los caminos posibles. Esto se debe a que tienen un apartado extra en la condición que comprueba que el objeto no tiene como valor nulo al atributo del cuál se va a comprobar dicha validación. Este código es necesario ya que sino se comprueba esto daría un error de pánico al intentar acceder y comprobar un valor el cual es nulo. Por otro lado, en las pruebas no se es posible pasar por los 4 estados del if ya que uno de ellos por el propio funcionamiento de java, es decir, si la primera parte de una operación AND es falsa, el resto no se ejecuta, por lo tanto no existiría la posibilidad de que la primera parte sea falsa y la segunda verdadera.

Analyse

En el análisis se puede observar que se prueban todos los casos dentro del rango positivo como tanto los casos negativos y los de hacking.

Actualizar un proyecto

Para la actualización de un proyecto se realizaron dos archivos de testing: *update.safe* y *update.hack*. En el archivo .safe se probaron todos los casos negativos según

la metodología dada en clase y posteriormente se procedió a probar los datos positivos usando el rango mínimo y máximo de los valores que se podían tomar. En el archivo .hack se intentó acceder con un rol inadecuado y con un usuario al que no le pertenecía el proyecto, y por último a actualizar proyectos que incluían SQL y HTML injection dando como resultado que no se ejecutaron las instrucciones maliciosas y por lo tanto sacando como conclusión de que el código no es vulnerable.

Coverage

```
@Service
public class ManagerProjectUpdateService extends AbstractService<Manager, Project> {
    // Internal state ---------------------------------------------------------

    @Autowired
    private ManagerProjectRepository        repository;

    @Autowired
    private SystemConfigurationRepository   systemConfigurationRepository;

    // AbstractService interface ----------------------------------------------


    @Override
    public void authorise() {
        boolean status;
        int id;
        int managerId;
        Project project;

        id = super.getRequest().getData("id", int.class);
        project = this.repository.findOneProjectById(id);

        managerId = super.getRequest().getPrincipal().getActiveRoleId();

        status = managerId == project.getManager().getId() && project.isDraftMode();

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        Project object;
        Integer id;

        id = super.getRequest().getData("id", int.class);
        object = this.repository.findOneProjectById(id);

        super.getBuffer().addData(object);
    }
```

```
    @Override
    public void bind(final Project object) {
        assert object != null;

        Integer managerId = super.getRequest().getPrincipal().getActiveRoleId();
        Manager manager = this.repository.findOneManagerById(managerId);
        object.setManager(manager);
        super.bind(object, "title", "code", "abstractText", "cost", "link");
    }

    @Override
    public void validate(final Project object) {
        assert object != null;

        if (!super.getBuffer().getErrors().hasErrors("code")) {

            Project projectSameCode = this.repository.findOneProjectByCode(object.getCode());

            if (projectSameCode != null)
                super.state(projectSameCode.getId() == object.getId(), "code", "manager.project.form.error.code");
        }

        if (!super.getBuffer().getErrors().hasErrors("cost") && object.getCost() != null)
            super.state(object.getCost().getAmount() >= 0, "cost", "manager.project.form.error.cost-negative");

        if (!super.getBuffer().getErrors().hasErrors("cost"))
            super.state(object.getCost() != null, "cost", "manager.project.form.error.cost-null");

        if (!super.getBuffer().getErrors().hasErrors("cost") && object.getCost() != null) {
            String symbol = object.getCost().getCurrency();
            boolean existsCurrency = this.systemConfigurationRepository.existsCurrency(symbol);
            super.state(existsCurrency, "cost", "manager.project.form.error.not-valid-currency");
        }

        if (!super.getBuffer().getErrors().hasErrors("cost") && object.getCost() != null)
            super.state(object.getCost().getAmount() <= 1000000, "cost", "manager.project.form.error.not-valid-currency");

    }

    @Override
    public void perform(final Project object) {
        assert object != null;

        this.repository.save(object);
    }
```

```
    @Override
    public void unbind(final Project object) {
        assert object != null;

        Dataset dataset;

        dataset = super.unbind(object, "title", "code", "abstractText", "cost", "link", "draftMode")

        super.getResponse().addData(dataset);
    }
}
```

Como podemos observar, ignorando las líneas con *assert* las cuales no podemos probar, en el validador se encuentran algunos *if* los cuales están amarillos indicando que no han pasado por todos los caminos posibles. Esto se debe a que tienen un apartado extra en la condición que comprueba que el objeto no tiene como valor nulo al atributo del cuál se va a comprobar dicha validación. Este código es necesario ya que sino se comprueba esto daría un error de pánico al intentar acceder y comprobar un valor el cual es nulo. Por otro lado, en las pruebas no se es posible pasar por los 4 estados del if ya que uno de ellos por el propio funcionamiento de java, es decir, si la primera parte de una operación AND es falsa, el resto no se ejecuta, por lo tanto no existiría la posibilidad de que la primera parte sea falsa y la segunda verdadera.

Además hay un camino en el *authorise* que no se llega a probar. Esto es debido a que es necesario de POST hacking para probarlo.

Analyse

En el análisis se puede observar que se prueban todos los casos dentro del rango positivo como tanto los casos negativos y los de hacking.

Publicar proyectos

Para la publicación de un proyecto se realizó un archivo de testing: *publish.safe*. Se probaron todos los casos negativos según la metodología dada en clase y las validaciones necesarias para que un proyecto sea publicado.

Coverage

```java
@Service
public class ManagerProjectPublishService extends AbstractService<Manager, Project> {
    // Internal state ---------------------------------------------------------

    @Autowired
    private ManagerProjectRepository        repository;

    @Autowired
    private SystemConfigurationRepository   systemConfigurationRepository;

    // AbstractService interface ----------------------------------------------


    @Override
    public void authorise() {
        boolean status;
        int id;
        int managerId;
        Project project;

        id = super.getRequest().getData("id", int.class);
        project = this.repository.findOneProjectById(id);

        managerId = super.getRequest().getPrincipal().getActiveRoleId();

        status = managerId == project.getManager().getId() && project.isDraftMode();

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        Project object;
        int id;

        id = super.getRequest().getData("id", int.class);
        object = this.repository.findOneProjectById(id);

        super.getBuffer().addData(object);
    }
```

```
    @Override
    public void bind(final Project object) {
        assert object != null;

        super.bind(object, "title", "code", "abstractText", "cost", "link");
    }

    @Override
    public void validate(final Project object) {
        assert object != null;

        if (!super.getBuffer().getErrors().hasErrors("hasFatalError"))
            super.state(!object.isHasFatalError(), "*", "manager.project.form.error.hasFatalError");

        if (!super.getBuffer().getErrors().hasErrors("userStory")) {
            Collection<UserStory> userStories = this.repository.findUserStoryByProjectId(object.getId());
            super.state(!userStories.isEmpty(), "*", "manager.project.form.error.atLeastOneUserStory");
            boolean userStoriesPublished = userStories.stream().allMatch(u -> !u.isDraftMode());
            super.state(userStoriesPublished, "*", "manager.project.form.error.userStoriesPublished");
        }

        if (!super.getBuffer().getErrors().hasErrors("cost") && object.getCost() != null)
            super.state(object.getCost().getAmount() >= 0, "cost", "manager.project.form.error.cost-negative");

        if (!super.getBuffer().getErrors().hasErrors("code")) {

            Project projectSameCode = this.repository.findOneProjectByCode(object.getCode());

            if (projectSameCode != null)
                super.state(projectSameCode.getId() == object.getId(), "code", "manager.project.form.error.code");
        }

        if (!super.getBuffer().getErrors().hasErrors("cost"))
            super.state(object.getCost() != null, "cost", "manager.project.form.error.cost-null");

        if (!super.getBuffer().getErrors().hasErrors("cost") && object.getCost() != null) {
            String symbol = object.getCost().getCurrency();
            boolean existsCurrency = this.systemConfigurationRepository.existsCurrency(symbol);
            super.state(existsCurrency, "cost", "manager.project.form.error.not-valid-currency");
        }

        if (!super.getBuffer().getErrors().hasErrors("cost") && object.getCost() != null)
            super.state(object.getCost().getAmount() <= 1000000, "cost", "manager.project.form.error.not-valid-currency");
    }

    @Override
    public void perform(final Project object) {
        assert object != null;

        object.setDraftMode(false);
        this.repository.save(object);
    }

    @Override
    public void unbind(final Project object) {
        assert object != null;

        Dataset dataset;

        dataset = super.unbind(object, "title", "code", "abstractText", "cost", "link", "draftMode");

        super.getResponse().addData(dataset);
    }
}
```

Como podemos observar, ignorando las líneas con *assert* las cuales no podemos probar, en el validador se encuentran algunos *if* los cuales están amarillos indicando que no han pasado por todos los caminos posibles. Esto se debe a que tienen un apartado extra en la condición que comprueba que el objeto no tiene como valor nulo al atributo del cuál se va a comprobar dicha validación. Este código es necesario ya que sino se comprueba esto daría un error de pánico al intentar acceder y comprobar un valor el cual es nulo. Por otro lado, en las pruebas no se es posible pasar por los 4 estados del if ya que uno de ellos por el propio funcionamiento de java, es decir, si la primera parte de una operación AND es falsa, el resto no se ejecuta, por lo tanto no existiría la posibilidad de que la primera parte sea falsa y la segunda verdadera.

Además hay un validador que nunca ha dado error ya que es de un atributo que no se encuentra en el formulario y por lo tanto no se puede probar con GET hacking. Sin embargo si lo añadiésemos con un POST hacking el validador debería de hacer su trabajo ya que no se pueden publicar proyectos con errores fatales.

Por último hay un camino en el *authorise* que no se llega a probar. Esto es debido a que es necesario de POST hacking para probarlo.

Analyse

En el análisis se puede observar que se probaron los casos negativos al tener un comportamiento de *update* y los casos en los que los validadores no son válidos.

Borrar proyecto

Para el borrado de un proyecto se realizó un archivo de testing: *delete.safe*. En el archivo .safe se probó a borrar un proyecto. Como no hay casos negativos solo se ha probado el caso positivo.

Coverage

```
@Service
public class ManagerProjectDeleteService extends AbstractService<Manager, Project> {
    // Internal state ---------------------------------------------------------

    @Autowired
    private ManagerProjectRepository repository;

    // AbstractService interface ----------------------------------------------


    @Override
    public void authorise() {
        boolean status;
        int id;
        int managerId;
        Project project;

        id = super.getRequest().getData("id", int.class);
        project = this.repository.findOneProjectById(id);

        managerId = super.getRequest().getPrincipal().getActiveRoleId();

        status = managerId == project.getManager().getId() && project.isDraftMode();

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        Project object;
        int id;

        id = super.getRequest().getData("id", int.class);
        object = this.repository.findOneProjectById(id);

        super.getBuffer().addData(object);
    }

    @Override
    public void bind(final Project object) {
        assert object != null;

        super.bind(object, "title", "code", "abstractText", "cost", "link");
    }

  @Override
  public void validate(final Project object) {
      assert object != null;

  }
```

```
    @Override
    public void perform(final Project object) {
        assert object != null;

        Collection<UserStoryProject> relations = this.repository.findRelationsByProjectId(object.getId());

        this.repository.deleteAll(relations);

        this.repository.delete(object);
    }

    @Override
    public void unbind(final Project object) {
        assert object != null;

        Dataset dataset;

        dataset = super.unbind(object, "title", "code", "abstractText", "cost", "link");

        super.getResponse().addData(dataset);
    }
}
```

Como podemos observar, ignorando los *assert* los cuales no podemos probar ya que es algo que no controlamos, hay un camino en el *authorise* que no se llega a probar. Esto es debido a que es necesario de POST hacking para probarlo.

Además el unbind tampoco se prueba ya que se ejecutaría en caso de que haya un pánico a la hora de realizar el delete, como por ejemplo un error con la base de datos y por lo tanto es un caso de prueba que no podemos cubrir.

Analyse

En el análisis se puede observar que el resultado de las llamadas realizadas para borrar el proyecto.

1.2. Operaciones del gestor en las historias de usuario

Listado de las historias de usuario

Para el listado de las historias de usuario se realizaron dos archivos de testing: *list.safe* y *list.hack*. En el archivo .safe se probó listar las historias de usuarios de los usuarios con rol de gestor para cubrir que todos los elementos de los datos de prueba se mostrasen en la pantalla. En el archivo .hack se probó realizar la funcionalidad con otro rol.

Además se probó accediendo directamente desde el menú de navegación y desde dentro de un proyecto para ver sus historias de usuario relacionadas.

Coverage

```
@Service
public class ManagerUserStoryListService extends AbstractService<Manager, UserStory> {
    // Internal state ---------------------------------------------------------

    @Autowired
    private ManagerUserStoryRepository repository;

    // AbstractService interface ----------------------------------------------


    @Override
    public void authorise() {

        boolean status = true;

        if (super.getRequest().hasData("projectId")) {
            int projectId;
            int managerId;
            Project project;

            projectId = super.getRequest().getData("projectId", int.class);
            project = this.repository.findOneProjectById(projectId);

            managerId = super.getRequest().getPrincipal().getActiveRoleId();

            status = managerId == project.getManager().getId();
        }

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        Collection<UserStory> objects;
        int projectId;
        int managerId;

        managerId = super.getRequest().getPrincipal().getActiveRoleId();

        if (super.getRequest().hasData("projectId")) {
            projectId = super.getRequest().getData("projectId", int.class);
            objects = this.repository.findAllUserStoriesByProjectId(projectId);
        } else
            objects = this.repository.findAllUserStories(managerId);

        super.getBuffer().addData(objects);
    }
```

```
    @Override
    public void unbind(final UserStory object) {
        assert object != null;

        Dataset dataset;

        dataset = super.unbind(object, "title", "description", "estimatedCost", "priority", "link", "acceptanceCriteria");

        super.getResponse().addData(dataset);
    }

    @Override
    public void unbind(final Collection<UserStory> objects) {
        assert objects != null;

        if (super.getRequest().hasData("projectId")) {
            int projectId;
            Project project;

            projectId = super.getRequest().getData("projectId", int.class);
            project = this.repository.findOneProjectById(projectId);

            super.getResponse().addGlobal("projectId", projectId);
            super.getResponse().addGlobal("canCreate", project.isDraftMode());
        }
    }

}
```

Como se puede observar, el coverage es totalmente completo exceptuando un assert el cual no podemos probar porque es una situación del framework el cual no podemos controlar.

Analyse

En el análisis se puede observar que se prueban todos los datos de pruebas existentes para las historias de usuario y por lo tanto se podría decir que son completos.

Detalles de una historia de usuario

Para los detalles de una historia de usuario se realizaron dos archivos de testing: *show.safe* y *show.hack*. En el archivo .safe se probó mostrar los datos de todos las historias de usuario de los datos de prueba y en el archivo .hack se intentó acceder con un rol inadecuado y con otro usuario del mismo rol realizando GET hacking.

Coverage

```
@Service
public class ManagerUserStoryShowService extends AbstractService<Manager, UserStory> {

    // Internal state ---------------------------------------------------------

    @Autowired
    private ManagerUserStoryRepository repository;

    // AbstractService interface ----------------------------------------------


    @Override
    public void authorise() {
        boolean status;
        int id;
        int managerId;
        UserStory userStory;

        id = super.getRequest().getData("id", int.class);
        userStory = this.repository.findOneUserStoryById(id);

        managerId = super.getRequest().getPrincipal().getActiveRoleId();

        status = managerId == userStory.getManager().getId();

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        UserStory object;
        int id;

        id = super.getRequest().getData("id", int.class);
        object = this.repository.findOneUserStoryById(id);

        super.getBuffer().addData(object);
    }


    @Override
    public void unbind(final UserStory object) {
        assert object != null;

        Dataset dataset;
        SelectChoices choices;

        choices = SelectChoices.from(Priority.class, object.getPriority());

        dataset = super.unbind(object, "title", "description", "estimatedCost", "priority", "link", "acceptanceCriteria", "draftMode");
        dataset.put("priorities", choices);

        super.getResponse().addData(dataset);
    }

}
```

Como se puede observar, se ha realizado un buen testeo ya que la única línea en la cual no se han recorrido todos los casos es en la que no podemos ya que es un caso que no controlamos nosotros.

Analyse

Como se puede observar, se prueban todos los datos de pruebas existentes para las historias de usuario y por lo tanto se podría decir que son completos.

Crear una historia de usuario

Para la creación de una historia de usuario se realizaron tres archivos de testing: *create.safe, create-extra.safe* y *create.hack*. En el archivo .safe se probaron todos los casos negativos según la metodología dada en clase y posteriormente se procedió a probar los datos positivos usando el rango mínimo y máximo de los valores que se podían tomar. En el archivo extra.safe se probó un caso que faltó en el primer .safe y para no repetir el test entero ya que es de grandes dimensiones se complementó con este. En el archivo .hack se intentó acceder con un rol inadecuado y a crear historias de usuario que incluían SQL y HTML injection dando como resultado que no se ejecutaron las instrucciones maliciosas y por lo tanto sacando como conclusión de que el código no es vulnerable.

También se probó a crear una historia de usuario tanto dentro como fuera de un proyecto.

Coverage

```
@Service
public class ManagerUserStoryCreateService extends AbstractService<Manager, UserStory> {
    // Internal state ---------------------------------------------------------

    @Autowired
    private ManagerUserStoryRepository repository;

    // AbstractService interface ----------------------------------------------


    @Override
    public void authorise() {

        boolean status = true;

        if (super.getRequest().hasData("projectId")) {
            Integer projectId;
            Project project;
            int managerId;

            projectId = super.getRequest().getData("projectId", int.class);
            project = this.repository.findOneProjectById(projectId);

            managerId = super.getRequest().getPrincipal().getActiveRoleId();

            status = managerId == project.getManager().getId() && project.isDraftMode();

        }

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        UserStory object;

        object = new UserStory();
        Integer managerId = super.getRequest().getPrincipal().getActiveRoleId();
        Manager manager = this.repository.findOneManagerById(managerId);

        object.setManager(manager);
        object.setDraftMode(true);

        super.getBuffer().addData(object);
    }

    @Override
    public void bind(final UserStory object) {
        assert object != null;

        super.bind(object, "title", "description", "estimatedCost", "priority", "link", "acceptanceCriteria");
    }

    @Override
    public void validate(final UserStory object) {
        assert object != null;

        if (super.getRequest().hasData("projectId") && !super.getBuffer().getErrors().hasErrors("published project")) {
            Integer projectId;
            Project project;

            projectId = super.getRequest().getData("projectId", int.class);
            project = this.repository.findOneProjectById(projectId);

            super.state(project.isDraftMode(), "*", "manager.user-story.form.error.project-published");
        }
        if (!super.getBuffer().getErrors().hasErrors("estimatedCost")) {
            double maxDouble = Double.MAX_VALUE;
            super.state(object.getEstimatedCost() < maxDouble, "cost", "manager.project.form.error.not-valid-currency");
        }

    }

    @Override
    public void perform(final UserStory object) {
        assert object != null;

        this.repository.save(object);

        if (super.getRequest().hasData("projectId")) {
            Integer projectId;
            Project project;

            projectId = super.getRequest().getData("projectId", int.class);
            project = this.repository.findOneProjectById(projectId);

            UserStoryProject usp = new UserStoryProject();
            usp.setProject(project);
            usp.setUserStory(object);
            this.repository.save(usp);
        }

    }
```

```java
    @Override
    public void unbind(final UserStory object) {
        assert object != null;

        Dataset dataset;
        SelectChoices choices;

        choices = SelectChoices.from(Priority.class, object.getPriority());

        dataset = super.unbind(object, "title", "description", "estimatedCost", "priority", "link", "acceptanceCriteria", "draftMode");
        dataset.put("priorities", choices);

        if (super.getRequest().hasData("projectId"))
            dataset.put("projectId", super.getRequest().getData("projectId", int.class));

        super.getResponse().addData(dataset);
    }
}
```

Como podemos observar, ignorando las líneas con *assert* las cuales no podemos probar, en el validador se encuentran algunos *if* los cuales están amarillos indicando que no han pasado por todos los caminos posibles. Esto se debe a que tienen un apartado extra en la condición que comprueba que el objeto no tiene como valor nulo al atributo del cuál se va a comprobar dicha validación. Este código es necesario ya que sino se comprueba esto daría un error de pánico al intentar acceder y comprobar un valor el cual es nulo. Por otro lado, en las pruebas no se es posible pasar por los 4 estados del if ya que uno de ellos por el propio funcionamiento de java, es decir, si la primera parte de una operación AND es falsa, el resto no se ejecuta, por lo tanto no existiría la posibilidad de que la primera parte sea falsa y la segunda verdadera.

Por último hay un camino en el *authorise* que no se llega a probar. Esto es debido a que es necesario de POST hacking para probarlo.

Analyse

En el análisis se puede observar que se prueban todos los casos dentro del rango positivo como tanto los casos negativos y los de hacking.

Actualizar una historia de usuario

Para la actualización de una historia de usuario se realizaron tres archivos de testing: *update.safe, update-extra.safe* y *update.hack*. En el archivo .safe se probaron todos los casos negativos según la metodología dada en clase y posteriormente se procedió a probar los datos positivos usando el rango mínimo y máximo de los valores que se podían tomar. En el archivo extra.safe se probó un caso que faltó en el primer .safe y para no repetir el test entero ya que es de grandes dimensiones se complementó con este. En el archivo .hack se intentó acceder con un rol inadecuado y con un usuario al que no le pertenecía el proyecto, y por último a actualizar historias de usuario que incluían SQL y HTML injection dando como resultado que no se ejecutaron las instrucciones maliciosas y por lo tanto sacando como conclusión de que el código no es vulnerable.

Coverage

```
@Service
public class ManagerUserStoryUpdateService extends AbstractService<Manager, UserStory> {
    // Internal state ---------------------------------------------------------

    @Autowired
    private ManagerUserStoryRepository repository;

    // AbstractService interface ----------------------------------------------


    @Override
    public void authorise() {
        boolean status;
        int id;
        int managerId;
        UserStory userStory;

        id = super.getRequest().getData("id", int.class);
        userStory = this.repository.findOneUserStoryById(id);

        managerId = super.getRequest().getPrincipal().getActiveRoleId();

        status = managerId == userStory.getManager().getId() && userStory.isDraftMode();

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        UserStory object;
        Integer id;

        id = super.getRequest().getData("id", int.class);
        object = this.repository.findOneUserStoryById(id);

        super.getBuffer().addData(object);
    }

    @Override
    public void bind(final UserStory object) {
        assert object != null;

        Integer managerId = super.getRequest().getPrincipal().getActiveRoleId();
        Manager manager = this.repository.findOneManagerById(managerId);
        object.setManager(manager);
        super.bind(object, "title", "description", "estimatedCost", "priority", "link", "acceptanceCriteria");
    }

    @Override
    public void validate(final UserStory object) {
        assert object != null;

        if (!super.getBuffer().getErrors().hasErrors("estimatedCost")) {
            double maxDouble = Double.MAX_VALUE;
            super.state(object.getEstimatedCost() < maxDouble, "cost", "manager.project.form.error.not-valid-currency");
        }
    }

    @Override
    public void perform(final UserStory object) {
        assert object != null;

        this.repository.save(object);
    }

    @Override
    public void unbind(final UserStory object) {
        assert object != null;

        Dataset dataset;
        SelectChoices choices;
        choices = SelectChoices.from(Priority.class, object.getPriority());

        dataset = super.unbind(object, "title", "description", "estimatedCost", "priority", "link", "acceptanceCriteria", "draftMode");
        dataset.put("priorities", choices);

        super.getResponse().addData(dataset);
    }
}
```

Como podemos observar, ignorando los *assert* los cuales no podemos probar ya que es algo que no controlamos, hay un camino en el *authorise* que no se llega a probar. Esto es debido a que es necesario de POST hacking para probarlo.

Analyse

En el análisis se puede observar que se prueban todos los casos dentro del rango positivo como tanto los casos negativos y los de hacking.

Publicar una historia de usuario

Para la publicación de una historia de usuario se realizaron dos archivos de testing: *publish.safe,* y *publish-extra.safe*. Se probaron todos los casos negativos según la metodología dada en clase. En el archivo extra.safe se probó un caso que faltó en el primer .safe y para no repetir el test entero ya que es de grandes dimensiones se complementó con este.

Coverage

```java
@Service
public class ManagerUserStoryPublishService extends AbstractService<Manager, UserStory> {
    // Internal state ---------------------------------------------------------

    @Autowired
    private ManagerUserStoryRepository repository;

    // AbstractService interface ----------------------------------------------


    @Override
    public void authorise() {
        boolean status;
        int id;
        int managerId;
        UserStory userStory;

        id = super.getRequest().getData("id", int.class);
        userStory = this.repository.findOneUserStoryById(id);

        managerId = super.getRequest().getPrincipal().getActiveRoleId();

        status = managerId == userStory.getManager().getId() && userStory.isDraftMode();

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        UserStory object;
        int id;

        id = super.getRequest().getData("id", int.class);
        object = this.repository.findOneUserStoryById(id);

        super.getBuffer().addData(object);
    }

    @Override
    public void bind(final UserStory object) {
        assert object != null;

        super.bind(object, "title", "description", "estimatedCost", "priority", "link", "acceptanceCriteria");
    }

    @Override
    public void validate(final UserStory object) {
        assert object != null;

        if (!super.getBuffer().getErrors().hasErrors("estimatedCost")) {
            double maxDouble = Double.MAX_VALUE;
            super.state(object.getEstimatedCost() < maxDouble, "cost", "manager.project.form.error.not-valid-currency");
        }
    }

    @Override
    public void perform(final UserStory object) {
        assert object != null;

        object.setDraftMode(false);
        this.repository.save(object);
    }
```

```
    @Override
    public void unbind(final UserStory object) {
        assert object != null;

        Dataset dataset;
        SelectChoices choices;
        choices = SelectChoices.from(Priority.class, object.getPriority());

        dataset = super.unbind(object, "title", "description", "estimatedCost", "priority", "link", "acceptanceCriteria", "draftMode");
        dataset.put("priorities", choices);

        super.getResponse().addData(dataset);
    }
}
```

Como podemos observar, ignorando los *assert* los cuales no podemos probar ya que es algo que no controlamos, hay un camino en el *authorise* que no se llega a probar. Esto es debido a que es necesario de POST hacking para probarlo.

Analyse

En el análisis se puede observar que se probaron los casos negativos al tener un comportamiento de *update*.

Borrar una historia de usuario

Para el borrado de una historia de usuario se realizó un archivo de testing: *delete.safe*. En el archivo .safe se probó a borrar una historia de usuario. Como no hay casos negativos solo se ha probado el caso positivo.

Coverage

```
@Service
public class ManagerUserStoryDeleteService extends AbstractService<Manager, UserStory> {
    // Internal state ---------------------------------------------------------

    @Autowired
    private ManagerUserStoryRepository repository;

    // AbstractService interface ----------------------------------------------


    @Override
    public void authorise() {
        boolean status;
        int id;
        int managerId;
        UserStory userStory;

        id = super.getRequest().getData("id", int.class);
        userStory = this.repository.findOneUserStoryById(id);

        managerId = super.getRequest().getPrincipal().getActiveRoleId();

        status = managerId == userStory.getManager().getId() && userStory.isDraftMode();

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        UserStory object;
        int id;

        id = super.getRequest().getData("id", int.class);
        object = this.repository.findOneUserStoryById(id);

        super.getBuffer().addData(object);
    }

    @Override
    public void bind(final UserStory object) {
        assert object != null;

        super.bind(object, "title", "description", "estimatedCost", "priority", "link", "acceptanceCriteria");
    }

    @Override
    public void validate(final UserStory object) {
        assert object != null;

    }
```

```
    @Override
    public void validate(final UserStory object) {
        assert object != null;

    }

    @Override
    public void perform(final UserStory object) {
        assert object != null;

        Collection<UserStoryProject> relations = this.repository.findAllRelationsByUserStoryId(object.getId());

        this.repository.deleteAll(relations);

        this.repository.delete(object);
    }

    @Override
    public void unbind(final UserStory object) {
        assert object != null;

        Dataset dataset;

        dataset = super.unbind(object, "title", "description", "estimatedCost", "priority", "link", "acceptanceCriteria", "draftMode");

        super.getResponse().addData(dataset);
    }
}
```

Como podemos observar, ignorando los *assert* los cuales no podemos probar ya que es algo que no controlamos, hay un camino en el *authorise* que no se llega a probar. Esto es debido a que es necesario de POST hacking para probarlo.

Además el unbind tampoco se prueba ya que se ejecutaría en caso de que haya un pánico a la hora de realizar el delete, como por ejemplo un error con la base de datos y por lo tanto es un caso de prueba que no podemos cubrir.

Analyse

En el análisis se puede observar que el resultado de las llamadas realizadas para borrar la historia de usuario.

1.3. Operaciones del gestor en la entidad intermedia entre los proyectos y las historias de usuario

Listado de la entidad intermedia

Para el listado de la entidad intermedia se realizaron dos archivos de testing: *list.safe* y *list.hack*. En el archivo .safe se probó listar las entidades intermedias de los usuarios con rol de gestor para cubrir que todos los elementos de los datos de prueba se mostrasen en la pantalla. En el archivo .hack se probó realizar la funcionalidad con otro rol.

Coverage

```
@Service
public class ManagerUserStoryProjectListService extends AbstractService<Manager, UserStoryProject> {

    // Internal state ---------------------------------------------------------

    @Autowired
    private ManagerUserStoryProjectRepository repository;

    // AbstractService interface ----------------------------------------------


    @Override
    public void authorise() {
        super.getResponse().setAuthorised(true);
    }

    @Override
    public void load() {
        Collection<UserStoryProject> objects;
        int managerId;

        managerId = super.getRequest().getPrincipal().getActiveRoleId();

        objects = this.repository.findAllRelationsByManager(managerId);

        super.getBuffer().addData(objects);
    }

    @Override
    public void unbind(final UserStoryProject object) {
        assert object != null;

        Dataset dataset;

        dataset = new Dataset();

        dataset.put("project", object.getProject().getCode());
        dataset.put("userStory", object.getUserStory().getTitle());
        dataset.put("id", object.getId());

        super.getResponse().addData(dataset);
    }

}
```

Como se puede observar, el coverage es totalmente completo exceptuando un assert el cual no podemos probar porque es una situación del framework el cual no podemos controlar.

Analyse

En el análisis se puede observar que se prueban todos los datos de pruebas existentes para las entidades intermedias y por lo tanto se podría decir que son completos.

Detalles de la entidad intermedia

Para los detalles de una entidad intermedia se realizaron dos archivos de testing: *show.safe* y *show.hack*. En el archivo .safe se probó mostrar los datos de todos las entidades intermedias de los datos de prueba y en el archivo .hack se intentó acceder con un rol inadecuado y con otro usuario del mismo rol realizando GET hacking.

Coverage

```
@Service
public class ManagerUserStoryProjectShowService extends AbstractService<Manager, UserStoryProject> {

    // Internal state ---------------------------------------------------------

    @Autowired
    private ManagerUserStoryProjectRepository repository;

    // AbstractService interface ----------------------------------------------


    @Override
    public void authorise() {
        boolean status;
        int id;
        int managerId;
        UserStoryProject userStoryProject;

        id = super.getRequest().getData("id", int.class);
        userStoryProject = this.repository.findOneUserStoryProjectById(id);

        managerId = super.getRequest().getPrincipal().getActiveRoleId();

        status = managerId == userStoryProject.getProject().getManager().getId();

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        UserStoryProject object;
        int id;

        id = super.getRequest().getData("id", int.class);
        object = this.repository.findOneUserStoryProjectById(id);

        super.getBuffer().addData(object);
    }
```

```java
@Override
public void unbind(final UserStoryProject object) {
    assert object != null;

    Dataset dataset;
    SelectChoices choicesProject;
    SelectChoices choicesUserStory;
    Project preselectedProject;
    UserStory preselectedUserStory;
    int managerId;

    managerId = super.getRequest().getPrincipal().getActiveRoleId();

    List<Project> projectsOwned = this.repository.findAllProjectsOwnedAndNotPublished(managerId).stream().toList();
    List<UserStory> userStoriesOwned = this.repository.findAllUserStoriesOwned(managerId).stream().toList();

    if (!object.getProject().isDraftMode()) {
        projectsOwned = new ArrayList<>();
        projectsOwned.add(object.getProject());

        userStoriesOwned = new ArrayList<>();
        userStoriesOwned.add(object.getUserStory());

    }

    preselectedProject = object.getProject();
    preselectedUserStory = object.getUserStory();

    choicesProject = SelectChoices.from(projectsOwned, "code", preselectedProject);
    choicesUserStory = SelectChoices.from(userStoriesOwned, "title", preselectedUserStory);

    dataset = new Dataset();

    String projectCode = preselectedProject.getCode();
    String userStoryTitle = preselectedUserStory.getTitle();

    dataset.put("project", projectCode);
    dataset.put("userStory", userStoryTitle);
    dataset.put("projects", choicesProject);
    dataset.put("userStories", choicesUserStory);
    dataset.put("id", object.getId());
    dataset.put("version", object.getVersion());
    dataset.put("draftMode", object.getProject().isDraftMode());

    super.getResponse().addData(dataset);
}
```

Como se puede observar, el coverage es totalmente completo exceptuando un assert el cual no podemos probar porque es una situación del framework el cual no podemos controlar.

Analyse

En el análisis se puede observar que se prueban todos los datos de pruebas existentes para las entidades intermedias y por lo tanto se podría decir que son completos.

Crear entidad intermedia

Para la creación de una entidad intermedia se realizaron tres archivos de testing: *create.safe, create-extra.safe* y *create.hack*. En el archivo .safe se probaron todos los casos negativos según la metodología dada en clase y posteriormente se procedió a probar los datos positivos usando el rango mínimo y máximo de los valores que se podían tomar. En el archivo extra.safe se probó un caso que faltó en el primer .safe y para no repetir el test entero ya que es de grandes dimensiones se complementó con este. En el archivo .hack se intentó acceder con un rol inadecuado y a crear entidades intermedias que incluían SQL y HTML injection dando como resultado que no se ejecutaron las instrucciones maliciosas y por lo tanto sacando como conclusión de que el código no es vulnerable.

Coverage

```java
@Service
public class ManagerUserStoryProjectCreateService extends AbstractService<Manager, UserStoryProject> {
    // Internal state ---------------------------------------------------------

    @Autowired
    private ManagerUserStoryProjectRepository repository;

    // AbstractService interface ----------------------------------------------


    @Override
    public void authorise() {
        super.getResponse().setAuthorised(true);
    }

    @Override
    public void load() {
        UserStoryProject object;

        object = new UserStoryProject();
        super.getBuffer().addData(object);
    }

    @Override
    public void bind(final UserStoryProject object) {
        assert object != null;

        super.bind(object, "project", "userStory");
    }

    @Override
    public void validate(final UserStoryProject object) {
        assert object != null;
        int managerId;

        managerId = super.getRequest().getPrincipal().getActiveRoleId();

        if (!super.getBuffer().getErrors().hasErrors("userStoryProject") && object.getProject() != null && object.getUserStory() != null) {

            UserStoryProject uspSameData = this.repository.findOneUserStoryProjectByProjectAndUserStory(object.getProject().getId(), object.getUserStory().getId());

            super.state(uspSameData == null, "*", "manager.user-story-project.form.error.same");
        }

        if (!super.getBuffer().getErrors().hasErrors("project") && object.getProject() != null)
            super.state(object.getProject().getManager().getId() == managerId, "*", "manager.user-story-project.form.error.project-owned");

        if (!super.getBuffer().getErrors().hasErrors("userStory") && object.getUserStory() != null)
            super.state(object.getUserStory().getManager().getId() == managerId, "*", "manager.user-story-project.form.error.user-story-owned");

    }


    @Override
    public void perform(final UserStoryProject object) {
        assert object != null;

        this.repository.save(object);
    }

    @Override
    public void unbind(final UserStoryProject object) {
        assert object != null;

        Dataset dataset;
        SelectChoices choicesProject;
        SelectChoices choicesUserStory;
        int managerId;

        managerId = super.getRequest().getPrincipal().getActiveRoleId();

        Collection<Project> projectsOwned = this.repository.findAllProjectsOwnedAndNotPublished(managerId);
        Collection<UserStory> userStoriesOwned = this.repository.findAllUserStoriesOwned(managerId);

        Project preselectedProject = projectsOwned.stream().toList().isEmpty() ? null : projectsOwned.stream().toList().get(0);

        UserStory preselectedUserStory = userStoriesOwned.stream().toList().isEmpty() ? null : userStoriesOwned.stream().toList().get(0);

        choicesProject = SelectChoices.from(projectsOwned, "code", preselectedProject);
        choicesUserStory = SelectChoices.from(userStoriesOwned, "title", preselectedUserStory);

        dataset = new Dataset();

        String projectCode = preselectedProject != null ? preselectedProject.getCode() : null;
        String userStoryTitle = preselectedUserStory != null ? preselectedUserStory.getTitle() : null;

        dataset.put("project", projectCode);
        dataset.put("userStory", userStoryTitle);
        dataset.put("projects", choicesProject);
        dataset.put("userStories", choicesUserStory);
        dataset.put("id", object.getId());
        dataset.put("version", object.getVersion());

        super.getResponse().addData(dataset);
    }
```

Como podemos observar, ignorando las líneas con *assert* las cuales no podemos probar, en el validador se encuentran algunos *if* los cuales están amarillos indicando que no han pasado por todos los caminos posibles. Esto se debe a que tienen un apartado extra en la condición que comprueba que el objeto no tiene como valor nulo al atributo del cuál se va a comprobar dicha validación. Este código es necesario ya que sino se comprueba esto daría un error de pánico al intentar acceder y comprobar un valor el cual es nulo. Por otro

lado, en las pruebas no se es posible pasar por los 4 estados del if ya que uno de ellos por el propio funcionamiento de java, es decir, si la primera parte de una operación AND es falsa, el resto no se ejecuta, por lo tanto no existiría la posibilidad de que la primera parte sea falsa y la segunda verdadera.

Además hay validadores que nunca ha dado error ya que solo se podrían probar con POST hacking.

Por último, en el unbind hay expresiones booleanas que no se llegan a probar todos los casos. Esto es debido a que son útiles para cuando la aplicación no tiene ningún tipo de dato, es decir, cuando corre con solo los datos iniciales. Como las pruebas se realizan con datos de testeo no pasan por todas las posibilidades.

Analyse

En el análisis se puede observar que se prueban todos los casos dentro del rango positivo como tanto los casos negativos y los de hacking.

Actualizar entidad intermedia

Para la actualización de una entidad intermedia se realizaron tres archivos de testing: *update.safe, update-extra.safe* y *update.hack*. En el archivo .safe se probaron todos los casos negativos según la metodología dada en clase y posteriormente se procedió a probar los datos positivos usando el rango mínimo y máximo de los valores que se podían tomar. En el archivo extra.safe se probó un caso que faltó en el primer .safe y para no repetir el test entero ya que es de grandes dimensiones se complementó con este. En el archivo .hack se intentó acceder con un rol inadecuado y con un usuario al que no le pertenecía el proyecto.

Coverage

```
@Service
public class ManagerUserStoryProjectUpdateService extends AbstractService<Manager, UserStoryProject> {
    // Internal state ---------------------------------------------------------

    @Autowired
    private ManagerUserStoryProjectRepository repository;

    // AbstractService interface ----------------------------------------------


    @Override
    public void authorise() {
        boolean status;
        int id;
        int managerId;
        UserStoryProject userStoryProject;

        id = super.getRequest().getData("id", int.class);
        userStoryProject = this.repository.findOneUserStoryProjectById(id);

        managerId = super.getRequest().getPrincipal().getActiveRoleId();

        status = managerId == userStoryProject.getProject().getManager().getId() && userStoryProject.getProject().isDraftMode();

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        UserStoryProject object;
        int id;

        id = super.getRequest().getData("id", int.class);
        object = this.repository.findOneUserStoryProjectById(id);

        super.getBuffer().addData(object);
    }

    @Override
    public void bind(final UserStoryProject object) {
        assert object != null;

        super.bind(object, "project", "userStory");
    }


    @Override
    public void validate(final UserStoryProject object) {
        assert object != null;
        int managerId;

        managerId = super.getRequest().getPrincipal().getActiveRoleId();

        if (!super.getBuffer().getErrors().hasErrors("userStoryProject") && object.getProject() != null && object.getUserStory() != null) {

            UserStoryProject uspSameData = this.repository.findOneUserStoryProjectByProjectAndUserStory(object.getProject().getId(), object.getUserStory().getId());

            if (uspSameData != null)
                super.state(uspSameData.getId() == object.getId(), "*", "manager.user-story-project.form.error.same");
        }

        if (!super.getBuffer().getErrors().hasErrors("project") && object.getProject() != null)
            super.state(object.getProject().getManager().getId() == managerId, "*", "manager.user-story-project.form.error.project-owned");

        if (!super.getBuffer().getErrors().hasErrors("userStory") && object.getUserStory() != null)
            super.state(object.getUserStory().getManager().getId() == managerId, "*", "manager.user-story-project.form.error.user-story-owned");

    }
```

```java
@Override
public void unbind(final UserStoryProject object) {
    assert object != null;

    Dataset dataset;
    SelectChoices choicesProject;
    SelectChoices choicesUserStory;
    int managerId;

    managerId = super.getRequest().getPrincipal().getActiveRoleId();

    Collection<Project> projectsOwned = this.repository.findAllProjectsOwnedAndNotPublished(managerId);
    Collection<UserStory> userStoriesOwned = this.repository.findAllUserStoriesOwned(managerId);

    Project preselectedProject = projectsOwned.stream().toList().isEmpty() ? null : projectsOwned.stream().toList().get(0);

    UserStory preselectedUserStory = userStoriesOwned.stream().toList().isEmpty() ? null : userStoriesOwned.stream().toList().get(0);

    choicesProject = SelectChoices.from(projectsOwned, "code", preselectedProject);
    choicesUserStory = SelectChoices.from(userStoriesOwned, "title", preselectedUserStory);

    dataset = new Dataset();

    String projectCode = preselectedProject != null ? preselectedProject.getCode() : null;
    String userStoryTitle = preselectedUserStory != null ? preselectedUserStory.getTitle() : null;

    dataset.put("project", projectCode);
    dataset.put("userStory", userStoryTitle);
    dataset.put("projects", choicesProject);
    dataset.put("userStories", choicesUserStory);
    dataset.put("id", object.getId());
    dataset.put("version", object.getVersion());
    dataset.put("draftMode", true);

    super.getResponse().addData(dataset);
}
```

Como podemos observar, ignorando las líneas con *assert* las cuales no podemos probar, en el validador se encuentran algunos *if* los cuales están amarillos indicando que no han pasado por todos los caminos posibles. Esto se debe a que tienen un apartado extra en la condición que comprueba que el objeto no tiene como valor nulo al atributo del cuál se va a comprobar dicha validación. Este código es necesario ya que sino se comprueba esto daría un error de pánico al intentar acceder y comprobar un valor el cual es nulo. Por otro lado, en las pruebas no se es posible pasar por los 4 estados del if ya que uno de ellos por el propio funcionamiento de java, es decir, si la primera parte de una operación AND es falsa, el resto no se ejecuta, por lo tanto no existiría la posibilidad de que la primera parte sea falsa y la segunda verdadera.

Además hay validadores que nunca ha dado error ya que solo se podrían probar con POST hacking.

También, hay un camino en el *authorise* que no se llega a probar. Esto es debido a que es necesario de POST hacking para probarlo.

Por último, en el unbind hay expresiones booleanas que no se llegan a probar todos los casos. Esto es debido a que son útiles para cuando la aplicación no tiene ningún tipo de dato, es decir, cuando corre con solo los datos iniciales. Como las pruebas se realizan con datos de testeo no pasan por todas las posibilidades.

Analyse

En el análisis se puede observar que se prueban todos los casos dentro del rango positivo como tanto los casos negativos y los de hacking.

Borrar entidad intermedia

Para el borrado de una entidad intermedia se realizó un archivo de testing: *delete.safe*. En el archivo .safe se probó a borrar una entidad intermedia. Como no hay casos negativos solo se ha probado el caso positivo.

Coverage

```
@Service
public class ManagerUserStoryProjectDeleteService extends AbstractService<Manager, UserStoryProject> {
    // Internal state ---------------------------------------------------------

    @Autowired
    private ManagerUserStoryProjectRepository repository;

    // AbstractService interface ----------------------------------------------


    @Override
    public void authorise() {
        boolean status;
        int id;
        int managerId;
        UserStoryProject userStoryProject;

        id = super.getRequest().getData("id", int.class);
        userStoryProject = this.repository.findOneUserStoryProjectById(id);

        managerId = super.getRequest().getPrincipal().getActiveRoleId();

        status = managerId == userStoryProject.getProject().getManager().getId() && userStoryProject.getProject().isDraftMode();

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        UserStoryProject object;
        int id;

        id = super.getRequest().getData("id", int.class);
        object = this.repository.findOneUserStoryProjectById(id);

        super.getBuffer().addData(object);
    }

    @Override
    public void bind(final UserStoryProject object) {
        assert object != null;

        super.bind(object, "project", "userStory");
    }

    @Override
    public void validate(final UserStoryProject object) {
        assert object != null;

    }
```

```java
    @Override
    public void perform(final UserStoryProject object) {
        assert object != null;

        this.repository.delete(object);
    }

    @Override
    public void unbind(final UserStoryProject object) {
        assert object != null;

        Dataset dataset;
        SelectChoices choicesProject;
        SelectChoices choicesUserStory;
        int managerId;

        managerId = super.getRequest().getPrincipal().getActiveRoleId();

        Collection<Project> projectsOwned = this.repository.findAllProjectsOwnedAndNotPublished(managerId);
        Collection<UserStory> userStoriesOwned = this.repository.findAllUserStoriesOwned(managerId);

        choicesProject = SelectChoices.from(projectsOwned, "code", object.getProject());
        choicesUserStory = SelectChoices.from(userStoriesOwned, "title", object.getUserStory());

        dataset = new Dataset();

        dataset.put("project", object.getProject().getCode());
        dataset.put("userStory", object.getUserStory().getTitle());
        dataset.put("projects", choicesProject);
        dataset.put("userStories", choicesUserStory);

        super.getResponse().addData(dataset);
    }
}
```

Como podemos observar, ignorando los *assert* los cuales no podemos probar ya que es algo que no controlamos, hay un camino en el *authorise* que no se llega a probar. Esto es debido a que es necesario de POST hacking para probarlo.

Además el unbind tampoco se prueba ya que se ejecutaría en caso de que haya un pánico a la hora de realizar el delete, como por ejemplo un error con la base de datos y por lo tanto es un caso de prueba que no podemos cubrir.

Analyse

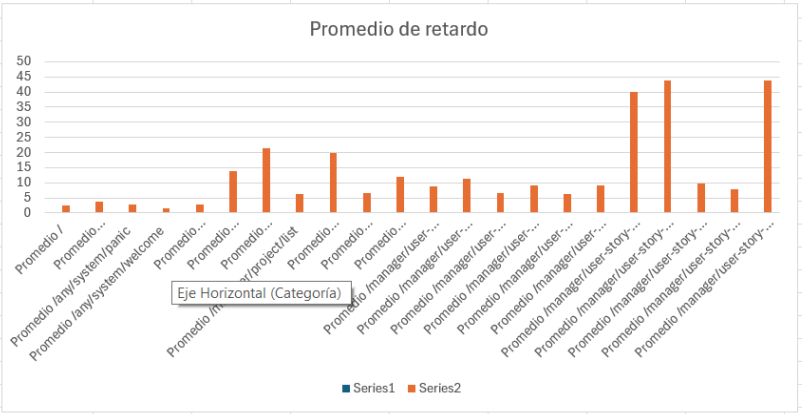
En el análisis se puede observar que el resultado de las llamadas realizadas para borrar el proyecto.

2. Performance testing

Estadísticas de rendimiento desde mi equipo

Para las estadísticas de rendimiento, se tomó una traza con más de 12.000 líneas obtenida al ejecutar todos los tests referentes al Student 1. Tras la limpieza definida en las diapositivas, la muestra llega a 1051 líneas, teniendo más de 50 entradas por *feature* las cuáles son necesarias para que los cálculos estadísticos sean significativos.

request-path	response-status	time
Promedio /		2,55143552
Promedio /anonymous/system/sign-in		3,71077188
Promedio /any/system/panic		2,9271
Promedio /any/system/welcome		1,4840285
Promedio /authenticated/system/sign-out		2,90239412
Promedio /manager/project/create		13,8402263
Promedio /manager/project/delete		21,5352
Promedio /manager/project/list		6,16170833
Promedio /manager/project/publish		19,9861938
Promedio /manager/project/show		6,49327105
Promedio /manager/project/update		12,0501425
Promedio /manager/user-story/create		8,73473333
Promedio /manager/user-story/delete		11,1915
Promedio /manager/user-story/list		6,676114
Promedio /manager/user-story/publish		9,02172
Promedio /manager/user-story/show		6,40918551
Promedio /manager/user-story/update		9,19139487
Promedio /manager/user-story-project/create		39,9315286
Promedio /manager/user-story-project/delete		43,6927
Promedio /manager/user-story-project/list		9,61791154
Promedio /manager/user-story-project/show		7,9608963
Promedio /manager/user-story-project/update		43,9441571
		0
		0
Promedio general		6,25700124



Como se puede observar, el promedio es de 0.006 segundos lo cual está bastante bien. La ruta que más rápido funciona es */any/system/welcome* el cual tiene sentido ya que no requiere de una llamada con la base de datos. La ruta que más lenta funciona es */manager/user-story-project/update*.

Estadísticas				
		Interval(ms)	6,78183885	5,75599981
Media	6,26891933	Interval(s)	0,00678184	0,005756
Error típico	0,26139649			
Mediana	4,718			
Moda	2,2787			
Desviación es	8,47021443			
Varianza de la	71,7445326			
Curtosis	27,8122392			
Coeficiente de	4,40052838			
Rango	101,5309			
Mínimo	0,8315			
Máximo	102,3624			
Suma	6582,3653			
Cuenta	1050			
Nivel de confia	0,51291952			

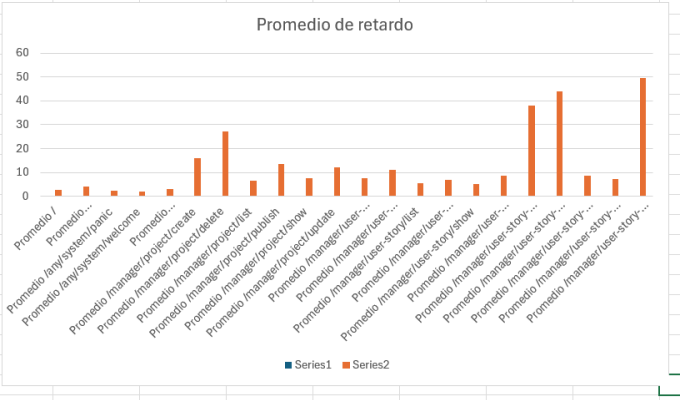
Por último, mi intervalo de la media con 95% del nivel de confianza es [5.755999981,6.78183885].

Comparativa entre equipos

Para este apartado comparé mis resultados con el equipo de Gonzalo Navas Remmers (Student 3).

Estos son los promedios de retardo de los tests ejecutados desde el equipo de Gonzalo:

Promedio /	2,62552191
Promedio /anonymous/system/sign-in	3,92115521
Promedio /any/system/panic	2,4347
Promedio /any/system/welcome	2,09108348
Promedio /authenticated/system/sign-out	3,00891875
Promedio /manager/project/create	16,044889
Promedio /manager/project/delete	27,1197
Promedio /manager/project/list	6,3768815
Promedio /manager/project/publish	13,3808688
Promedio /manager/project/show	7,73976552
Promedio /manager/project/update	12,0100098
Promedio /manager/user-story/create	7,57915828
Promedio /manager/user-story/delete	11,085
Promedio /manager/user-story/list	5,41064404
Promedio /manager/user-story/publish	6,80775993
Promedio /manager/user-story/show	5,17279859
Promedio /manager/user-story/update	8,5524129
Promedio /manager/user-story-project/create	38,0653857
Promedio /manager/user-story-project/delete	44,1487
Promedio /manager/user-story-project/list	8,61431131
Promedio /manager/user-story-project/show	7,21594826
Promedio /manager/user-story-project/update	49,5305857
Promedio general	5,97730135



Miguel - Student 1				*Gonzalo - Student 3*		
Media	6,26891933			Media	5,97730135	
Error típico	0,26139649			Error típico	0,27168672	
Mediana	4,718			Mediana	4,3116	
Moda	2,2787			Moda	2,1248	
Desviación es	8,47021443			Desviación es	7,77517835	
Varianza de la	71,7445326			Varianza de la	60,4533984	
Curtosis	27,8122392			Curtosis	27,9926212	
Coeficiente de	4,40052838			Coeficiente de	4,72819134	
Rango	101,5309			Rango	71,456702	
Mínimo	0,8315			Mínimo	1,342999	
Máximo	102,3624			Máximo	72,799701	
Suma	6582,3653			Suma	4895,4098	
Cuenta	1050			Cuenta	819	
Nivel de confia	0,51291952			Nivel de confia	0,53328525	
Intervalo(ms)	6,78183885	5,75599981		Intervalo(ms)	6,51058659	5,4440161
Intervalo(s)	0,00678184	0,005756		Intervalo(s)	0,00651059	0,00544402

Estas son las estadísticas obtenidas de las trazas al correr mis tests en los dos equipos. Se puede observar que mi media teniendo en cuenta el nivel de confianza es mayor que la de Gonzalo.

Prueba z para medias de dos muestras		
	Miguel	*Gonzalo*
Media	6,257001236	5,97730135
Varianza (conocida)	71,7445326	60,4533984
Observaciones	1052	819
Diferencia hipotética de las medias	0	
z	0,742215343	
P(Z<=z) una cola	0,228978437	
Valor crítico de z (una cola)	1,644853627	
Valor crítico de z (dos colas)	0,457956874	
Valor crítico de z (dos colas)	1,959963985	

Al realizar las pruebas z, se puede observar que el valor crítico de z (dos colas) es mayor que alpha (0.05) y que por lo tanto no hay un cambio significativo entre ambos equipos.

Comparativa con índices

Para mejorar el rendimiento del sistema se procedió a la inclusión de índices. Estos son los resultados de la comparativa:

Antes			*Después*		
Media	6,26891933		Media	6,20623014	
Error típico	0,26139649		Error típico	0,25878253	
Mediana	4,718		Mediana	4,67082	
Moda	2,2787		Moda	2,255913	
Desviación es	8,47021443		Desviación es	8,38551229	
Varianza de la	71,7445326		Varianza de la	70,3168164	
Curtosis	27,8122392		Curtosis	27,8122392	
Coeficiente de	4,40052838		Coeficiente de	4,40052838	
Rango	101,5309		Rango	100,515591	
Mínimo	0,8315		Mínimo	0,823185	
Máximo	102,3624		Máximo	101,338776	
Suma	6582,3653		Suma	6516,54165	
Cuenta	1050		Cuenta	1050	
Nivel de confia	0,51291952		Nivel de confia	0,50779032	
Intervalo(ms)	6,78183885	5,75599981	Intervalo(ms)	6,71402046	5,69843982
Intervalo(s)	0,00678184	0,005756	Intervalo(s)	0,00671402	0,00569844

No parecen haber grandes diferencias entre el uso o no uso de índices al menos en este caso específico.

Prueba z para medias de dos muestras		
	70,0646	*69,363954*
Media	6,20810362	6,14602259
Varianza (conocida)	71,7445326	70,3168164
Observaciones	1049	1049
Diferencia hipotética de las medias	0	
z	0,16869753	
P(Z<=z) una cola	0,43301728	
Valor crítico de z (una cola)	1,64485363	
Valor crítico de z (dos colas)	0,86603456	
Valor crítico de z (dos colas)	1,95996398	

En cuanto al valor *p*, supera el umbral de 0.05, por lo tanto no ha habido un cambio significativo.

Ya que la inclusión de índices resultó en fallas en los tests y ha conllevado a una baja mejora en el rendimiento, se tomó la decisión de eliminarlos del código.