DP2 2024

Acme Software Factory

Repositorio: https://github.com/DP2-2024-C1-029/Acme-Software-Factory.git

Miembro:

- José María Portela Huerta (josporhue@alum.us.es)

Tutor: José González Enríquez

GRUPO C1.029

27/05/2024

Versión 1.0

Índice

Historial de versiones

Fecha	Versión	Descripción	Entrega
27/05/2024	V1.0	Inicio del documento	D04
08/07/2024	V2.0	Revisión y cambios para la segunda Convocatoria	D04 2nd Call

Capítulo 1 – Pruebas funcionales

Sponsorship

Tras ejecutar todos los test, se puede observar que para sponsorship se cubre el 90,2%, valor que está por encima de la recomendación mínima del 90% que debería cubrir al menos todos los test. Igualmente, este porcentaje debería ser mayor si quitáramos del delete el unbind, método que nunca se va a ejecutar al no mostrarse errores al eliminar.

acme.features.sponsor.sponsorship	90,2 %	1.444	157	1.601
SponsorSponsorshipUpdateService.java	95,7 %	352	16	368
SponsorSponsorshipShowService.java	97,3 %	142	4	146
SponsorSponsorshipPublishService.java	96,1 %	398	16	414
SponsorSponsorshipListMineService.java	93,4 %	57	4	61
SponsorSponsorshipDeleteService.java	56,5 %	131	101	232
SponsorSponsorshipCreateService.java	95,3 %	328	16	344
SponsorSponsorshipController.java	100,0 %	36	0	36

En primer lugar, para no repetirlo durante todo el documento, se va a comentar que las líneas que los “assert” siempre aparecen en amarillo. Esto se debe a que, como nos comentan en las clases teóricas, es recomendable añadirlos al inicio de los proyectos software con el fin de asegurarnos que estamos trabajando con elementos no nulos.

```
assert object != null;
```

Ahora, voy a empezar hablando por el UpdateService.

Para el update service podemos observar que todo está en verde. Esto lo conseguimos en esta clase y en el resto de ellas, porque las pruebas de hacking nos ayudan a pasar por esas condiciones (del status) por las que normalmente un usuario normal no pasaría.

```
@Override
public void authorise() {
    boolean status;
    int masterId;
    Sponsorship sponsorship;

    masterId = super.getRequest().getData("id", int.class);
    sponsorship = this.repository.findOneSponsorshipById(masterId);

    status = sponsorship != null && super.getRequest().getPrincipal().hasRole(sponsorship.getSponsor()) && !sponsorship.isPublished();

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

@Override
public void load() {
    Sponsorship sponsorship;
    int id;

    id = super.getRequest().getData("id", int.class);
    sponsorship = this.repository.findOneSponsorshipById(id);

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

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

    int projectId;
    Project project;

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

    super.bind(object, "code", "initialExecutionPeriod", "endingExecutionPeriod", "amount", "type", "email", "link");
    object.setProject(project);
}
```

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

    if (!super.getBuffer().getErrors().hasErrors("code"))
        super.state(this.repository.notExistsDuplicatedCodeExceptThisByIdLike(object.getCode(), object.getId()), "code", "sponsor.sponsorship.form.error.duplic

    if (!super.getBuffer().getErrors().hasErrors("project")) {
        boolean existsProject;

        existsProject = this.repository.existsValidProject(object.getProject().getId());
        super.state(existsProject, "project", "sponsor.sponsorship.form.error.not-exists");
    }

    if (!super.getBuffer().getErrors().hasErrors("endingExecutionPeriod") && !super.getBuffer().getErrors().hasErrors("initialExecutionPeriod")) {
        Date minimumDeadline;

        minimumDeadline = MomentHelper.deltaFromMoment(object.getInitialExecutionPeriod(), 1, ChronoUnit.MONTHS);
        super.state(MomentHelper.isAfterOrEqual(object.getEndingExecutionPeriod(), minimumDeadline), "endingExecutionPeriod", "sponsor.sponsorship.form.error.t
    }

    if (!super.getBuffer().getErrors().hasErrors("initialExecutionPeriod"))
        super.state(MomentHelper.isAfter(object.getInitialExecutionPeriod(), object.getMoment()), "initialExecutionPeriod", "sponsor.sponsorship.form.error.not

    if (!super.getBuffer().getErrors().hasErrors("amount")) {
        super.state(this.repository.isAmongAcceptedCurrencies(object.getAmount().getCurrency()), "amount", "sponsor.sponsorship.form.error.currency-not-valid")

        super.state(object.getAmount().getAmount() > 0, "amount", "sponsor.sponsorship.form.error.negative-amount");

        if (this.repository.existsInvoicesPublishedOfSponsorship(object.getId())) {
            Boolean resultado = this.repository.existsSponsorshipByIdWithItsCurrencyLike(object.getId(), object.getAmount().getCurrency());
            super.state(resultado, "amount", "sponsor.sponsorship.form.error.cannot-change-currency");
        }
    }
}
```

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

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

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

    Collection<Project> projects;
    SelectChoices choicesProject;
    SelectChoices choicesType;
    Dataset dataset;

    choicesType = SelectChoices.from(SponsorType.class, object.getType());
    projects = this.repository.findManyProjects();
    choicesProject = SelectChoices.from(projects, "title", object.getProject());

    dataset = super.unbind(object, "code", "moment", "initialExecutionPeriod", "endingExecutionPeriod", "amount", "type", "email", "link", "isPublished");
    dataset.put("type", choicesType.getSelected().getKey());
    dataset.put("types", choicesType);
    dataset.put("project", choicesProject.getSelected().getKey());
    dataset.put("projects", choicesProject);

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

A continuación, vamos como el ShowService, el cual está también todo en verde. Para evitar repetirme por todo el documento, concreto ahora que solo se resaltará si ha habido algún caso en que no hemos conseguido todo verde.

```
@Override
public void authorise() {
    boolean status;
    int sponsorshipId;
    Sponsorship sponsorship;

    sponsorshipId = super.getRequest().getData("id", int.class);
    sponsorship = this.repository.findOneSponsorshipById(sponsorshipId);
    status = sponsorship != null && super.getRequest().getPrincipal().hasRole(sponsorship.getSponsor());

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

@Override
public void load() {
    Sponsorship object;
    int sponsorId;

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

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

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

    Collection<Project> projects;
    SelectChoices choicesType;
    SelectChoices choicesProject;
    Dataset dataset;

    choicesType = SelectChoices.from(SponsorType.class, object.getType());
    projects = this.repository.findManyProjects();
    choicesProject = SelectChoices.from(projects, "title", object.getProject());

    dataset = super.unbind(object, "code", "moment", "initialExecutionPeriod", "endingExecutionPeriod", "amount", "type", "email", "link", "isPublished");
    dataset.put("type", choicesType.getSelected().getKey());
    dataset.put("types", choicesType);
    dataset.put("project", choicesProject.getSelected().getKey());
    dataset.put("projects", choicesProject);

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

Continuamos con el publishService.

```
@Override
public void authorise() {
    boolean status;
    int masterId;
    Sponsorship sponsorship;

    masterId = super.getRequest().getData("id", int.class);
    sponsorship = this.repository.findOneSponsorshipById(masterId);

    status = sponsorship != null && super.getRequest().getPrincipal().hasRole(sponsorship.getSponsor()) && !sponsorship.isPublished();

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

@Override
public void load() {
    Sponsorship sponsorship;
    int id;

    id = super.getRequest().getData("id", int.class);
    sponsorship = this.repository.findOneSponsorshipById(id);

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

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

    int projectId;
    Project project;

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

    super.bind(object, "code", "initialExecutionPeriod", "endingExecutionPeriod", "amount", "type", "email", "link");
    object.setProject(project);
}
```

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

    if (!super.getBuffer().getErrors().hasErrors("code"))
        super.state(this.repository.notExistsDuplicatedCodeExceptThisByIdLike(object.getCode(), object.getId()), "code", "sponsor.sponsorship.form.error.duplicated");

    if (!super.getBuffer().getErrors().hasErrors("project")) {
        boolean myProject;

        myProject = this.repository.existsValidProject(object.getProject().getId());
        super.state(myProject, "project", "sponsor.sponsorship.form.error.not-exists");
    }

    if (!super.getBuffer().getErrors().hasErrors("endingExecutionPeriod") && !super.getBuffer().getErrors().hasErrors("initialExecutionPeriod")) {
        Date minimumDeadline;

        minimumDeadline = MomentHelper.deltaFromMoment(object.getInitialExecutionPeriod(), 1, ChronoUnit.MONTHS);
        super.state(MomentHelper.isAfterOrEqual(object.getEndingExecutionPeriod(), minimumDeadline), "endingExecutionPeriod", "sponsor.sponsorship.form.error.too-close");
    }

    if (!super.getBuffer().getErrors().hasErrors("initialExecutionPeriod"))
        super.state(MomentHelper.isAfter(object.getInitialExecutionPeriod(), object.getMoment()), "initialExecutionPeriod", "sponsor.sponsorship.form.error.not-after");

    if (!super.getBuffer().getErrors().hasErrors("amount")) {
        super.state(this.repository.isAmongAcceptedCurrencies(object.getAmount().getCurrency()), "amount", "sponsor.sponsorship.form.error.currency-not-valid");

        super.state(object.getAmount().getAmount() > 0, "amount", "sponsor.sponsorship.form.error.negative-amount");

    }

    if (!super.getBuffer().getErrors().hasErrors()) {
        //          boolean allInvoicesPublished;

        super.state(this.repository.existsInvoicesOfSponsorship(object.getId()), "*", "sponsor.sponsorship.form.error.none-invoices-asociated");

        //          allInvoicesPublished = this.repository.findManyInvoicesNotPublished(object.getId()).isEmpty();
        //          super.state(allInvoicesPublished, "*", "sponsor.sponsorship.form.error.not-all-invoices-published");
        super.state(this.repository.notAllInvoicesArePublishedOfSponsorship(object.getId()), "*", "sponsor.sponsorship.form.error.not-all-invoices-published");
    }

    if (!super.getBuffer().getErrors().hasErrors()) {
        Double sumTotalAmountsInvoices;

        sumTotalAmountsInvoices = this.repository.sumQuantityOfInvoicesBySponsorshipId(object.getId());
        super.state(object.getAmount().getAmount().equals(sumTotalAmountsInvoices), "*", "sponsor.sponsorship.form.error.not-total-amount-invoices");

        // Sabemos seguro que todas las invoices están públicas, por lo que obtendremos en este punto siempre un booleano
        boolean sameCurrency = this.repository.existsSponsorshipByIdWithItsCurrencyLike(object.getId(), object.getAmount().getCurrency());
        super.state(sameCurrency, "amount", "sponsor.sponsorship.form.error.cannot-change-currency");
    }
}
@Override
public void perform(final Sponsorship object) {
    assert object != null;

    object.setPublished(true);
    this.repository.save(object);
}

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

    Collection<Project> projects;
    SelectChoices choicesProject;
    SelectChoices choicesType;
    Dataset dataset;

    choicesType = SelectChoices.from(SponsorType.class, object.getType());
    projects = this.repository.findManyProjects();
    choicesProject = SelectChoices.from(projects, "title", object.getProject());

    dataset = super.unbind(object, "code", "moment", "initialExecutionPeriod", "endingExecutionPeriod", "amount", "type", "email", "link", "isPublished");
    dataset.put("type", choicesType.getSelected().getKey());
    dataset.put("types", choicesType);
    dataset.put("project", choicesProject.getSelected().getKey());
    dataset.put("projects", choicesProject);

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

Seguimos con el ListService.

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

@Override
public void load() {
    Collection<Sponsorship> objects;
    int sponsorId;

    sponsorId = super.getRequest().getPrincipal().getActiveRoleId();
    objects = this.repository.findManySponsorshipsBySponsorId(sponsorId);

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

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

    Dataset dataset;

    dataset = super.unbind(object, "code", "amount", "type");
    dataset.put("project", object.getProject().getTitle());

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

También tenemos el deleteService, que está todo en verde, menos el unbind porque siempre que un usuario quiera eliminar un objeto, que pueda eliminar, lo va a eliminar. Y, si no pudiera, saltaría un panic de no autorizado. Por eso está en rojo.

```
@Override
public void authorise() {
    boolean status;
    int masterId;
    Sponsorship sponsorship;

    masterId = super.getRequest().getData("id", int.class);
    sponsorship = this.repository.findOneSponsorshipById(masterId);

    status = sponsorship != null && super.getRequest().getPrincipal().hasRole(sponsorship.getSponsor()) && !sponsorship.isPublished();

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

@Override
public void load() {
    Sponsorship sponsorship;
    int id;

    id = super.getRequest().getData("id", int.class);
    sponsorship = this.repository.findOneSponsorshipById(id);

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

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

    int projectId;
    Project project;

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

    super.bind(object, "code", "initialExecutionPeriod", "endingExecutionPeriod", "amount", "type", "email", "link");
    object.setProject(project);
}
@Override
public void validate(final Sponsorship object) {
    assert object != null;
}

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

    Collection<Invoice> invoices;

    invoices = this.repository.findManyInvoicesBySponsorshipId(object.getId());

    this.repository.deleteAll(invoices);
    this.repository.delete(object);
}

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

    Collection<Project> projects;
    SelectChoices choicesProject;
    SelectChoices choicesType;
    Dataset dataset;

    choicesType = SelectChoices.from(SponsorType.class, object.getType());
    projects = this.repository.findManyProjects();
    choicesProject = SelectChoices.from(projects, "title", object.getProject());

    dataset = super.unbind(object, "code", "moment", "initialExecutionPeriod", "endingExecutionPeriod", "amount", "type", "email", "link");
    dataset.put("type", choicesType.getSelected().getKey());
    dataset.put("types", choicesType);
    dataset.put("project", choicesProject.getSelected().getKey());
    dataset.put("projects", choicesProject);

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

Por último tenemos el createService. Aquí mencionar que se ha tomado una decisión de diseño en el authorise, que es para evitar un hacking de que actualice otro elemento por el id y la versión, saltándose todas las validaciones que se encuentran en el update y publish, no

se permite el acceso si el elemento creado tiene un id distinto a 0 (en caso de que se esté haciendo post. Si no, no tendrá dicho campo id).

```
@Override
public void authorise() {
    boolean hasIdSponsorship;

    hasIdSponsorship = super.getRequest().hasData("id", Integer.class);

    super.getResponse().setAuthorised(!hasIdSponsorship || super.getRequest().getData("id", int.class).equals(0));
}

@Override
public void load() {
    Sponsorship sponsorship;
    Sponsor sponsor;

    sponsor = this.repository.findOneSponsorById(super.getRequest().getPrincipal().getActiveRoleId());
    sponsorship = new Sponsorship();
    sponsorship.setPublished(false);
    sponsorship.setSponsor(sponsor);
    sponsorship.setMoment(MomentHelper.deltaFromCurrentMoment(-1, ChronoUnit.MILLIS));

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

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

    int projectId;
    Project project;

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

    super.bind(object, "code", "initialExecutionPeriod", "endingExecutionPeriod", "amount", "type", "email", "link");
    object.setProject(project);
}

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

    if (!super.getBuffer().getErrors().hasErrors("code"))
        super.state(this.repository.notExistsDuplicatedCodeLike(object.getCode()), "code", "sponsor.sponsorship.form.error.duplicated");

    if (!super.getBuffer().getErrors().hasErrors("project")) {
        boolean myProject;

        myProject = this.repository.existsValidProject(object.getProject().getId());
        super.state(myProject, "project", "sponsor.sponsorship.form.error.not-exists");
        //          super.state(object.getProject().isDraftMode(), "project", "sponsor.sponsorship.form.error.not-exists");
    }

    if (!super.getBuffer().getErrors().hasErrors("endingExecutionPeriod") && !super.getBuffer().getErrors().hasErrors("initialExecutionPeriod")) {
        Date minimumDeadline;

        minimumDeadline = MomentHelper.deltaFromMoment(object.getInitialExecutionPeriod(), 1, ChronoUnit.MONTHS);
        super.state(MomentHelper.isAfterOrEqual(object.getEndingExecutionPeriod(), minimumDeadline), "endingExecutionPeriod", "sponsor.sponsorship.form.error.too-close");
    }

    if (!super.getBuffer().getErrors().hasErrors("initialExecutionPeriod"))
        super.state(MomentHelper.isAfter(object.getInitialExecutionPeriod(), object.getMoment()), "initialExecutionPeriod", "sponsor.sponsorship.form.error.not-after");

    if (!super.getBuffer().getErrors().hasErrors("amount")) {
        super.state(this.repository.isAmongAcceptedCurrencies(object.getAmount().getCurrency()), "amount", "sponsor.sponsorship.form.error.currency-not-valid");

        super.state(object.getAmount().getAmount() > 0, "amount", "sponsor.sponsorship.form.error.negative-amount");
    }
}
@Override
public void perform(final Sponsorship object) {
    assert object != null;

    this.repository.save(object);
}

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

    Collection<Project> projects;
    SelectChoices choicesProject;
    SelectChoices choicesType;
    Dataset dataset;

    choicesType = SelectChoices.from(SponsorType.class, object.getType());
    projects = this.repository.findManyProjects();
    choicesProject = SelectChoices.from(projects, "title", object.getProject());
    //      if (projects.contains(object.getProject()))
    //          choicesProject = SelectChoices.from(projects, "title", object.getProject());
    //      else
    //          choicesProject = SelectChoices.from(projects, "title", null);

    dataset = super.unbind(object, "code", "moment", "initialExecutionPeriod", "endingExecutionPeriod", "amount", "type", "email", "link", "isPublished");
    dataset.put("type", choicesType.getSelected().getKey());
    dataset.put("types", choicesType);
    dataset.put("project", choicesProject.getSelected().getKey());
    dataset.put("projects", choicesProject);

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

Como conclusión se puede sacar que todo el “Sponsorship” ha sido probado de manera muy exhaustiva, probando todas las validaciones posibles. Para concluir, decir que en la tabla Excel añadida entre los reportes, llamada DatosPruebaTesting.xlsx, se encuentran todos los datos de pruebas, tanto de casos positivos y negativos en la hoja Sponsorships, como de

hacking en la hoja HACK-Sponsorships. Se añaden ahí todos los ejemplos y no en este documento porque se ilustran mejor los datos probados con una descripción en este formato, lo cuál debe hacer más sencillo visualizar todos los casos de prueba. A mí, en particular, me sirvió para darme cuenta de si me faltaba algún caso alguna vez, y hacía más mecánico el repetir los tests si se me había olvidado hacer algún detalle en alguno de ellos.

Invoice

En esta ocasión 90% por los pelos, con el porcentaje disminuido de nuevo porque en el método delete no es posible acceder al unbind, como antes.

acme.features.sponsor.invoice	90,0 %	1.175	130	1.305
SponsorInvoiceUpdateService.java	93,5 %	230	16	246
SponsorInvoiceShowService.java	96,5 %	109	4	113
SponsorInvoicePublishService.java	95,0 %	301	16	317
SponsorInvoiceListService.java	93,2 %	136	10	146
SponsorInvoiceDeleteService.java	59,0 %	98	68	166
SponsorInvoiceCreateService.java	94,3 %	266	16	282
SponsorInvoiceController.java	100,0 %	35	0	35

Como en el apartado anterior, vamos a comenzar con el updateService.

```
@Override
public void authorise() {
    boolean status;
    int masterId;
    Invoice invoice;

    masterId = super.getRequest().getData("id", int.class);
    invoice = this.repository.findOneInvoiceById(masterId);
    status = invoice != null && super.getRequest().getPrincipal().hasRole(invoice.getSponsorship().getSponsor()) && !invoice.isPublished();

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

@Override
public void load() {
    Invoice object;
    int masterId;

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

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

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

    super.bind(object, "code", "dueDate", "quantity", "tax", "link");
}
```

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

    if (!super.getBuffer().getErrors().hasErrors("code"))
        super.state(this.repository.notExistsDuplicatedCodeExceptThisIdLike(object.getCode(), object.getId()), "code", "sponsor.invoice.form.error.duplicated");

    if (!super.getBuffer().getErrors().hasErrors("dueDate")) {
        Date minimumDeadline;

        minimumDeadline = MomentHelper.deltaFromMoment(object.getRegistrationTime(), 1, ChronoUnit.MONTHS);
        super.state(MomentHelper.isAfter(object.getDueDate(), minimumDeadline), "dueDate", "sponsor.invoice.form.error.too-close");
    }

    if (!super.getBuffer().getErrors().hasErrors("quantity")) {
        super.state(this.repository.isAmongAcceptedCurrencies(object.getQuantity().getCurrency()), "quantity", "sponsor.invoice.form.error.quantity-not-valid");

        super.state(object.getQuantity().getAmount() > 0, "quantity", "sponsor.invoice.form.error.negative-amount");
    }
}

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

    this.repository.save(object);
}

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

    Dataset dataset;

    dataset = super.unbind(object, "code", "registrationTime", "dueDate", "quantity", "tax", "link", "isPublished");
    dataset.put("totalAmount", object.totalAmount());
    dataset.put("masterId", object.getSponsorship().getId());

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

Pasamos con el showService.

```
@Override
public void authorise() {
    boolean status;
    int invoiceId;
    Sponsorship sponsorship;

    invoiceId = super.getRequest().getData("id", int.class);
    sponsorship = this.repository.findOneSponsorshipByInvoiceId(invoiceId);
    status = sponsorship != null && super.getRequest().getPrincipal().hasRole(sponsorship.getSponsor());

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

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

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

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

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

    Dataset dataset;

    dataset = super.unbind(object, "code", "registrationTime", "dueDate", "quantity", "tax", "link", "isPublished");
    dataset.put("totalAmount", object.totalAmount());
    dataset.put("masterId", object.getSponsorship().getId());

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

Seguimos con el publishService.

```java
@Override
public void authorise() {
    boolean status;
    int masterId;
    Invoice invoice;

    masterId = super.getRequest().getData("id", int.class);
    invoice = this.repository.findOneInvoiceById(masterId);
    status = invoice != null && super.getRequest().getPrincipal().hasRole(invoice.getSponsorship().getSponsor()) && !invoice.isPublished();

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

@Override
public void load() {
    Invoice object;
    int masterId;

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

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

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

    super.bind(object, "code", "dueDate", "quantity", "tax", "link");
}
@Override
public void validate(final Invoice object) {
    assert object != null;

    if (!super.getBuffer().getErrors().hasErrors("code"))
        super.state(this.repository.notExistsDuplicatedCodeExceptThisIdLike(object.getCode(), object.getId()), "code", "sponsor.invoice.form.error.duplicated");

    if (!super.getBuffer().getErrors().hasErrors("dueDate")) {
        Date minimumDeadline;

        minimumDeadline = MomentHelper.deltaFromMoment(object.getRegistrationTime(), 1, ChronoUnit.MONTHS);
        super.state(MomentHelper.isAfter(object.getDueDate(), minimumDeadline), "dueDate", "sponsor.invoice.form.error.too-close");
    }

    if (!super.getBuffer().getErrors().hasErrors("quantity")) {
        super.state(this.repository.isAmongAcceptedCurrencies(object.getQuantity().getCurrency()), "quantity", "sponsor.invoice.form.error.quantity-not-valid");

        super.state(object.getQuantity().getAmount() > 0, "quantity", "sponsor.invoice.form.error.negative-amount");
    }

    if (!super.getBuffer().getErrors().hasErrors("quantity")) {
        // Para comprobar que tiene la currency del resto de invoices publicadas, y si no de la del sponsorship
        super.state(object.getSponsorship().getAmount().getCurrency().equals(object.getQuantity().getCurrency()), "quantity", "sponsor.invoice.form.error.currency-diferent-from-sponsorship");

        // Para comprobar que tiene la suma de quantity menor o igual que la amount del sponsorship
        super.state(this.calculateIsLowerOrEqualCuantityToSponsorship(object), "quantity", "sponsor.invoice.form.error.more-than-sponsorship");
    }
}

private boolean calculateIsLowerOrEqualCuantityToSponsorship(final Invoice object) {
    double cuantityOfNewInvoice;
    Double sumOfCuantitiesPublished;

    cuantityOfNewInvoice = object.getQuantity().getAmount() * (100.0 + object.getTax()) / 100;
    sumOfCuantitiesPublished = this.repository.sumOfQuantitiesOfInvoicesOfSponsorship(object.getSponsorship().getId());

    if (sumOfCuantitiesPublished == null)
        sumOfCuantitiesPublished = 0.0;

    return sumOfCuantitiesPublished + cuantityOfNewInvoice <= object.getSponsorship().getAmount().getAmount();
}
@Override
public void perform(final Invoice object) {
    assert object != null;

    object.setPublished(true);
    this.repository.save(object);
}

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

    Dataset dataset;

    dataset = super.unbind(object, "code", "registrationTime", "dueDate", "quantity", "tax", "link", "isPublished");
    dataset.put("totalAmount", object.totalAmount());
    dataset.put("masterId", object.getSponsorship().getId());

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

Luego está el listService.

```
@Override
public void authorise() {
    boolean status;
    int masterId;
    Sponsorship sponsorship;

    masterId = super.getRequest().getData("masterId", int.class);
    sponsorship = this.repository.findOneSponsorshipById(masterId);
    status = sponsorship != null && super.getRequest().getPrincipal().hasRole(sponsorship.getSponsor());

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

@Override
public void load() {
    Collection<Invoice> objects;
    int sponsorId;

    sponsorId = super.getRequest().getData("masterId", int.class);
    objects = this.repository.findManyInvoicesByMasterId(sponsorId);

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

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

    Dataset dataset;
    String trueValue;
    String falseValue;

    trueValue = super.getRequest().getLocale().toString().equals("es") ? "Sí" : "Yes";
    falseValue = "No";

    dataset = super.unbind(object, "code", "dueDate");
    dataset.put("isPublished", object.isPublished() ? trueValue : falseValue);
    dataset.put("totalAmount", object.totalAmount());

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

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

    int masterId;
    Sponsorship sponsorship;

    masterId = super.getRequest().getData("masterId", int.class);
    sponsorship = this.repository.findOneSponsorshipById(masterId);

    super.getResponse().addGlobal("masterId", masterId);
    super.getResponse().addGlobal("showCreate", !sponsorship.isPublished());
}
```

También tenemos el deleteService, con su unbind en rojo por lo explicado anteriormente.

```
@Override
public void authorise() {
    boolean status;
    int masterId;
    Invoice invoice;

    masterId = super.getRequest().getData("id", int.class);
    invoice = this.repository.findOneInvoiceById(masterId);
    status = invoice != null && super.getRequest().getPrincipal().hasRole(invoice.getSponsorship().getSponsor()) && !invoice.isPublished();

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

@Override
public void load() {
    Invoice object;
    int masterId;

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

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

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

    super.bind(object, "code", "dueDate", "quantity", "tax", "link");
}

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

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

    this.repository.delete(object);
}

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

    Dataset dataset;

    dataset = super.unbind(object, "code", "dueDate", "quantity", "tax", "link", "isPublished");
    dataset.put("totalAmount", object.totalAmount());
    dataset.put("masterId", object.getSponsorship().getId());

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

Y, para terminar, el createService.

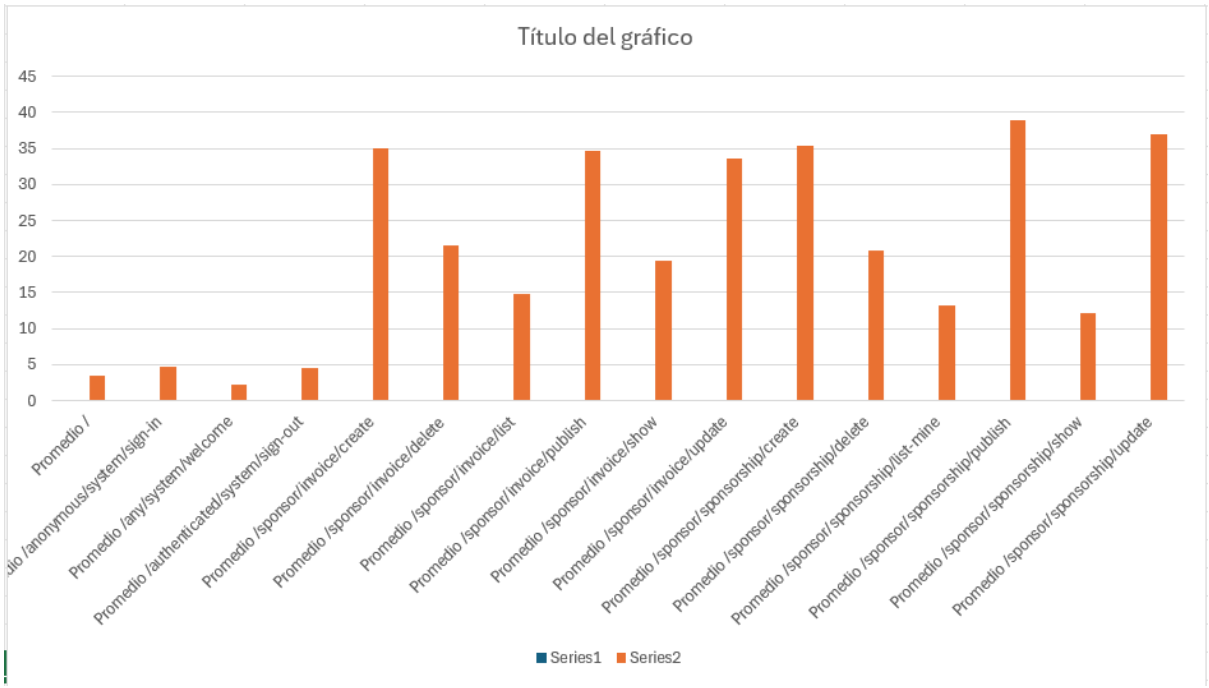
```
@Override
public void authorise() {
    boolean hasIdInvoice;
    boolean status;
    int masterId;
    Sponsorship sponsorship;

    masterId = super.getRequest().getData("masterId", int.class);
    sponsorship = this.repository.findOneSponsorshipById(masterId);
    status = sponsorship != null && super.getRequest().getPrincipal().hasRole(sponsorship.getSponsor()) && !sponsorship.isPublished();

    // Prevenir errores de hacking por id
    hasIdInvoice = super.getRequest().hasData("id", Integer.class);
    status = status && (!hasIdInvoice || super.getRequest().getData("id", int.class).equals(0));

    super.getResponse().setAuthorised(status);

}

@Override
public void load() {
    Invoice object;
    int masterId;
    Sponsorship invoice;

    masterId = super.getRequest().getData("masterId", int.class);
    invoice = this.repository.findOneSponsorshipById(masterId);

    object = new Invoice();
    object.setSponsorship(invoice);
    object.setPublished(false);
    object.setRegistrationTime(MomentHelper.deltaFromCurrentMoment(-1, ChronoUnit.MILLIS));

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

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

    super.bind(object, "code", "dueDate", "quantity", "tax", "link");
}
@Override
public void validate(final Invoice object) {
    assert object != null;

    if (!super.getBuffer().getErrors().hasErrors("code"))
        super.state(this.repository.notExistsDuplicatedCodeLike(object.getCode()), "code", "sponsor.invoice.form.error.duplicated");

    if (!super.getBuffer().getErrors().hasErrors("dueDate")) {
        Date minimumDeadline;

        minimumDeadline = MomentHelper.deltaFromMoment(object.getRegistrationTime(), 1, ChronoUnit.MONTHS);
        super.state(MomentHelper.isAfterOrEqual(object.getDueDate(), minimumDeadline), "dueDate", "sponsor.invoice.form.error.too-close");
    }

    if (!super.getBuffer().getErrors().hasErrors("quantity")) {
        super.state(this.repository.isAmongAcceptedCurrencies(object.getQuantity().getCurrency()), "quantity", "sponsor.invoice.form.error.quantity-not-valid");

        super.state(object.getQuantity().getAmount() > 0, "quantity", "sponsor.invoice.form.error.negative-amount");
    }
}

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

    this.repository.save(object);
}

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

    Dataset dataset;

    dataset = super.unbind(object, "code", "registrationTime", "dueDate", "quantity", "tax", "link", "isPublished");
    dataset.put("totalAmount", object.totalAmount());
    dataset.put("masterId", object.getSponsorship().getId());

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

Como conclusión se puede sacar que todo el “Invoice” ha sido probado de manera muy exhaustiva, probando todas las validaciones posibles. Al final de este capítulo comento de nuevo la tabla de Excel, con la batería de datos que se ha usado para probar todo el proyecto. También tiene datos para Invoice y sus hacking. Se ha de comentar que no solo se ha hecho una petición, si no que todas las operaciones se han repetido muchas veces, en algunas ocasiones se ha probado entre 30-40 veces como pueden ser los casos del publish, update o create.

Capítulo 2 – Pruebas de desempeño

El desarrollo del software se ha ejecutado durante todo el cuatrimestre en el PC1. Obteniendo los resultados de ejecutar el replayer en eclipse, nos genera una batería de datos, los cuales, analizándolos mediante las técnicas enseñadas en clase, hemos podido obtener resultaros claros.

Vamos a empezar por los promedios de los resultados de búsqueda.



Como se puede observar, ningún promedio supera los 40 ms, lo que es un resultado muy bueno, porque nos indica que las búsquedas se realizan de manera rápida. Comentar que esta foto es después de refactorizar y corregir el código, dejando todo en las querys. Esto ha ayudado a que el sistema haya podido optimizarse y no acumule retrasos en entornos más grandes.

A continuación, vamos a observar el intervalo de confianza para **el PC 1** tanto antes de introducir índices, como después de meter índices.

Before				*After*		
Media	20,4509999			Media	18,0554033	
Error típico	0,86566384			Error típico	0,77277437	
Mediana	10,1189			Mediana	8,3056	
Moda	3,238			Moda	1,978	
Desviación estándar	22,7556006			Desviación estándar	20,3138261	
Varianza de la muestra	517,817357			Varianza de la muestra	412,651533	
Curtosis	5,623617			Curtosis	9,0822933	
Coeficiente de asimetría	1,99472178			Coeficiente de asimetría	2,18230811	
Rango	165,7957			Rango	191,1809	
Mínimo	1,5029			Mínimo	1,0182	
Máximo	167,2986			Máximo	192,1991	
Suma	14131,6409			Suma	12476,2837	
Cuenta	691			Cuenta	691	
Nivel de confianza(95,0%)	1,69965131			Nivel de confianza(95,0%)	1,51727137	
Interval (ms)	18,7513485	22,1506512		Interval (ms)	16,538132	19,5726747
Interval (s)	0,01875135	0,02215065		Interval (s)	0,01653813	0,01957267

Con estos datos se ha realizado un Z-Test, el cual se muestra a continuación.

Prueba z para medias de dos muestras		
	Before	*After*
Media	20,4509999	18,0554033
Varianza (conocida)	517,817357	412,651533
Observaciones	691	691
Diferencia hipotética de las medias	0	
z	2,06443765	
P(Z<=z) una cola	0,01948812	
Valor crítico de z (una cola)	1,64485363	
Valor crítico de z (dos colas)	0,03897624	
Valor crítico de z (dos colas)	1,95996398	

Podemos observar que Alpha es 0.05, y que el p-value es 0.03897…, por lo que podemos decir que los cambios **sí** dieron como resultado cambios importantes (no demasiado porque están algo próximos a Alpha, pero no está junto a éste); los tiempos de muestreo son diferentes, siendo la media de después mejor. Por lo que concluimos que mejora el sistema con índices.

<u>También se ha replicado estas pruebas en otro ordenador</u> (PC2 – características similares) y he obtenido los siguientes resultados:

Before				*After*		
Media	20,6735026			Media	20,63319442	
Error típico	0,875640632			Error típico	0,87811966	
Mediana	10,248576			Mediana	10,324321	
Moda	#N/D			Moda	3,17324	
Desviación estándar	23,01785921			Desviación estándar	23,08302512	
Varianza de la muestra	529,8218428			Varianza de la muestra	532,8260486	
Curtosis	5,372744716			Curtosis	6,085358591	
Coeficiente de asimetría	1,975264212			Coeficiente de asimetría	2,051549425	
Rango	162,434699			Rango	174,16368	
Mínimo	1,517929			Mínimo	1,49985	
Máximo	163,952628			Máximo	175,66353	
Suma	14285,3903			Suma	14257,53734	
Cuenta	691			Cuenta	691	
Nivel de confianza(95,0%)	1,719239818			Nivel de confianza(95,0%)	1,724107161	
Interval (ms)	18,95426278	22,3927424		Interval (ms)	18,90908726	22,3573016
Interval (s)	0,018954263	0,02239274		Interval (s)	0,018909087	0,0223573

Con estos resultados se ha hecho de nuevo un **Z-Test (PC 2)** para analizar correctamente los datos, y he obtenido:

Prueba z para medias de dos muestras		
	Variable 1	*Variable 2*
Media	20,6450196	18,2320567
Varianza (conocida)	529,821843	532,826049
Observaciones	691	691
Diferencia hipotética de las medias	0	
z	1,94578551	
P(Z<=z) una cola	0,02584025	
Valor crítico de z (una cola)	1,64485363	
Valor crítico de z (dos colas)	0,05168051	
Valor crítico de z (dos colas)	1,95996398	

Podemos observar que Alpha es 0.05, y que el p-value es 0.05168051… por lo que podemos decir que los cambios no dieron como resultado ninguna mejora significativa; los tiempos de muestreo son diferentes, pero son globalmente iguales.

Como **conclusión**, solo el PC1 muestra una diferencia significativa (ligeramente) en el rendimiento (antes y después) a un nivel de variación del 5%. Por lo tanto, las diferencias observadas en las medias no son estadísticamente significativas, lo que sugiere que solo el primero de los PCs es concluyentemente más rápido o lento que el otro según los datos proporcionados.

Bibliografía

Diapositivas de Diseño y Pruebas 2 – Universidad de Sevilla.