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 María Portela Huerta

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

Capítulo 1 – Pruebas funcionales

Sponsorship

Tras ejecutar todos los test, se puede observar que para sponsorship se cubre el 89,9%, valor que está muy próximo de la recomendación mínima del 90% que debería cubrir al menos todos los test. Pero, no lo hace por el delete, cuyo unbind baja.

acme.features.sponsor.sponsorship	89,9 %	1.493	168	1.661
SponsorSponsorshipDeleteService.java	57,0 %	138	104	242
SponsorSponsorshipPublishService.java	95,6 %	410	19	429
SponsorSponsorshipUpdateService.java	94,9 %	354	19	373
SponsorSponsorshipCreateService.java	95,5 %	340	16	356
SponsorSponsorshipShowService.java	96,3 %	158	6	164
SponsorSponsorshipListMineService.java	93,4 %	57	4	61
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, y que el status tampoco se puede poner en verde, porque hay un caso que nunca se puede probar.

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

    masterId = super.getRequest().getData("id", int.class);
    sponsorship = this.repository.findOneSponsorshipById(masterId);
    sponsor = sponsorship == null ? null : sponsorship.getSponsor();

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

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

Voy a empezar hablando por el UpdateService.

Para el update service podemos observar que todo está en verde, menos alguna validación, que se ha probado y funciona igual en otros casos. Supongo que se me habrá pasado probarlo aquí, porque en los otros (como el create) funciona.

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

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

        existing = this.repository.findOneSponsorshipByCode(object.getCode());
        super.state(existing == null || existing.equals(object), "code", "sponsor.sponsorship.form.error.duplicated");
    }

    if (!super.getBuffer().getErrors().hasErrors("project")) {
        Collection<Project> myProjects;

        myProjects = this.repository.findManyProjects();

        super.state(myProjects.contains(object.getProject()), "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.isAfter(object.getEndingExecutionPeriod(), minimumDeadline), "endingExecutionPeriod", "sponsor.sponsorship.form.error.too-clos
    }

    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")) {
        String[] acceptedCurrencies = this.repository.findAcceptedCurrencies().split(";");
        super.state(Stream.of(acceptedCurrencies).anyMatch(c -> c.equals(object.getAmount().getCurrency())), //
            "amount", "sponsor.sponsorship.form.error.currency-not-valid");

        super.state(object.getAmount().getAmount() > 0 && object.getAmount().getAmount() <= 1000000, "amount", "sponsor.sponsorship.form.error.negative-amount"
    }
}
```

A continuación, vamos como el ShowService, el cual está todo en verde, menos el showInvoice que está implementado así para los casos de hackeo.

```
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;

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

    sponsorId = super.getRequest().getPrincipal().getActiveRoleId();
    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);
    dataset.put("showInvoices", sponsorId == object.getSponsor().getId());

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

Continuamos con el publishService.

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

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

        existing = this.repository.findOneSponsorshipByCode(object.getCode());
        super.state(existing == null || existing.equals(object), "code", "sponsor.sponsorship.form.error.duplicated");
    }

    if (!super.getBuffer().getErrors().hasErrors("project")) {
        Collection<Project> myProjects;

        myProjects = this.repository.findManyProjects();

        super.state(myProjects.contains(object.getProject()), "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(), 7, ChronoUnit.DAYS);
        super.state(MomentHelper.isAfter(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")) {
        String[] acceptedCurrencies = this.repository.findAcceptedCurrencies().split(";");
        super.state(Stream.of(acceptedCurrencies).anyMatch(c -> c.equals(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("amount")) {
        Double sumTotalAmountsInvoices;

        sumTotalAmountsInvoices = this.repository.findManyInvoicesBySponsorshipId(object.getId()).stream().map(Invoice::totalAmount).mapToDouble(Money::getAmount).sum();
        sumTotalAmountsInvoices = Math.round(sumTotalAmountsInvoices * 100) / 100.0;
        super.state(sumTotalAmountsInvoices.equals(object.getAmount().getAmount()), "*", "sponsor.sponsorship.form.error.not-total-amount-invoices");
    }

    {
        boolean allInvoicesPublished;

        allInvoicesPublished = this.repository.findManyInvoicesBySponsorshipId(object.getId()).stream().allMatch(Invoice::isPublished);
        super.state(allInvoicesPublished, "*", "sponsor.sponsorship.form.error.not-all-invoices-published");
    }
}
```

```
    @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);
    }
}
```

Como se puede comprobar todo en verde.

Seguimos con el ListService.

```
25      @Override
26      public void authorise() {
27          super.getResponse().setAuthorised(true);
28      }
29
30      @Override
31      public void load() {
32          Collection<Sponsorship> objects;
33          int sponsorId;
34
35          sponsorId = super.getRequest().getPrincipal().getActiveRoleId();
36          objects = this.repository.findManySponsorshipsBySponsorId(sponsorId);
37
38          super.getBuffer().addData(objects);
39      }
40
41      @Override
42      public void unbind(final Sponsorship object) {
43          assert object != null;
44
45          Dataset dataset;
46
47          dataset = super.unbind(object, "code", "amount", "type");
48          dataset.put("project", object.getProject().getTitle());
49
50          super.getResponse().addData(dataset);
51      }
52  }
```

Está todo en verde.

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.

```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", "moment", "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

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

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

            existing = this.repository.findOneSponsorshipByCode(object.getCode());
            super.state(existing == null, "code", "sponsor.sponsorship.form.error.duplicated");
        }

        if (!super.getBuffer().getErrors().hasErrors("project")) {
            Collection<Project> myProjects;

            myProjects = this.repository.findManyProjects();
            super.state(myProjects.contains(object.getProject()), "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.isAfter(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")) {
            String[] acceptedCurrencies = this.repository.findAcceptedCurrencies().split(";");
            super.state(Stream.of(acceptedCurrencies).anyMatch(c -> c.equals(object.getAmount().getCurrency())), //
                "amount", "sponsor.sponsorship.form.error.currency-not-valid");

            super.state(object.getAmount().getAmount() > 0 && object.getAmount().getAmount() <= 1000000, "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());

        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. Al final de este capítulo se muestra una imagen con la batería de datos que se ha usado para probar todo el proyecto. 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 créate.

Invoice

En esta ocasión, casi superamos el umbral del 90% sin embargo, se el porcentaje ha disminuido debido a que en el método delete no es posible acceder al unbind.

acme.features.sponsor.invoice	89,4 %	1.183	141	1.324
SponsorInvoiceDeleteService.java	57,1 %	97	73	170
SponsorInvoiceCreateService.java	94,1 %	273	17	290
SponsorInvoicePublishService.java	93,9 %	263	17	280
SponsorInvoiceUpdateService.java	93,9 %	260	17	277
SponsorInvoiceListService.java	92,3 %	144	12	156
SponsorInvoiceShowService.java	95,7 %	111	5	116
SponsorInvoiceController.java	100,0 %	35	0	35

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

```
59    @Override
60    public void validate(final Invoice object) {
61        assert object != null;
62
63        if (!super.getBuffer().getErrors().hasErrors("code")) {
64            Invoice existing;
65
66            existing = this.repository.findOneInvoiceByCode(object.getCode());
67            super.state(existing == null || existing.equals(object), "code", "sponsor.invoice.form.error.duplicated");
68        }
69
70        if (!super.getBuffer().getErrors().hasErrors("dueDate")) {
71            Date minimumDeadline;
72
73            minimumDeadline = MomentHelper.deltaFromMoment(object.getRegistrationTime(), 1, ChronoUnit.MONTHS);
74            super.state(MomentHelper.isAfter(object.getDueDate(), minimumDeadline), "dueDate", "sponsor.invoice.form.error.too-close");
75        }
76
77        if (!super.getBuffer().getErrors().hasErrors("quantity")) {
78            String[] acceptedCurrencies = this.repository.findAcceptedCurrencies().split(";");
79            super.state(Stream.of(acceptedCurrencies).anyMatch(c -> c.equals(object.getQuantity().getCurrency())), //
80                "quantity", "sponsor.invoice.form.error.quantity-not-valid");
81
82            super.state(object.getQuantity().getAmount() > 0, "quantity", "sponsor.invoice.form.error.negative-amount");
83        }
84    }
```

```
59    @Override
60    public void validate(final Invoice object) {
61        assert object != null;
62
63        if (!super.getBuffer().getErrors().hasErrors("code")) {
64            Invoice existing;
65
66            existing = this.repository.findOneInvoiceByCode(object.getCode());
67            super.state(existing == null || existing.equals(object), "code", "sponsor.invoice.form.error.duplicated");
68        }
69
70        if (!super.getBuffer().getErrors().hasErrors("dueDate")) {
71            Date minimumDeadline;
72
73            minimumDeadline = MomentHelper.deltaFromMoment(object.getRegistrationTime(), 1, ChronoUnit.MONTHS);
74            super.state(MomentHelper.isAfter(object.getDueDate(), minimumDeadline), "dueDate", "sponsor.invoice.form.error.too-close");
75        }
76
77        if (!super.getBuffer().getErrors().hasErrors("quantity")) {
78            String[] acceptedCurrencies = this.repository.findAcceptedCurrencies().split(";");
79            super.state(Stream.of(acceptedCurrencies).anyMatch(c -> c.equals(object.getQuantity().getCurrency())), //
80                "quantity", "sponsor.invoice.form.error.quantity-not-valid");
81
82            super.state(object.getQuantity().getAmount() > 0, "quantity", "sponsor.invoice.form.error.negative-amount");
83        }
84    }
```

Podemos observar que está todo en verde y comprobado perfectamente.

Pasamos con el showService que está todo en verde también.

```
30          sponsorshipId = super.getRequest().getData("id", int.class);
31          sponsorship = this.repository.findOneSponsorshipByInvoiceId(sponsorshipId);
32          status = sponsorship != null && (sponsorship.isPublished() || super.getRequest().getPrincipal().hasRole(sponsorship.getSponsor()));
33
34          super.getResponse().setAuthorised(status);
35      }
36
37      @Override
38      public void load() {
39          Invoice object;
40          int id;
41
42          id = super.getRequest().getData("id", int.class);
43          object = this.repository.findOneInvoiceById(id);
44
45          super.getBuffer().addData(object);
46      }
47
48      @Override
49      public void unbind(final Invoice object) {
50          assert object != null;
51
52          Dataset dataset;
53
54          dataset = super.unbind(object, "code", "registrationTime", "dueDate", "quantity", "tax", "link", "isPublished");
55          dataset.put("totalAmount", object.totalAmount());
56          dataset.put("masterId", object.getSponsorship().getId());
57
58          super.getResponse().addData(dataset);
59      }
```

Seguimos con el publishService, que es exactamente igual que el updateService y que también es igual que el createService, el cual tiene todo en verde..

```
59    @Override
60    public void validate(final Invoice object) {
61        assert object != null;
62
63        if (!super.getBuffer().getErrors().hasErrors("code")) {
64            Invoice existing;
65
66            existing = this.repository.findOneInvoiceByCode(object.getCode());
67            super.state(existing == null || existing.equals(object), "code", "sponsor.invoice.form.error.duplicated");
68        }
69
70        if (!super.getBuffer().getErrors().hasErrors("dueDate")) {
71            Date minimumDeadline;
72
73            minimumDeadline = MomentHelper.deltaFromMoment(object.getRegistrationTime(), 1, ChronoUnit.MONTHS);
74            super.state(MomentHelper.isAfter(object.getDueDate(), minimumDeadline), "dueDate", "sponsor.invoice.form.error.too-close");
75        }
76
77        if (!super.getBuffer().getErrors().hasErrors("quantity")) {
78            String[] acceptedCurrencies = this.repository.findAcceptedCurrencies().split(";");
79            super.state(Stream.of(acceptedCurrencies).anyMatch(c -> c.equals(object.getQuantity().getCurrency())), //
80                "quantity", "sponsor.invoice.form.error.quantity-not-valid");
81
82            super.state(object.getQuantity().getAmount() > 0, "quantity", "sponsor.invoice.form.error.negative-amount");
83        }
84    }
```

```
86    @Override
87    public void perform(final Invoice object) {
88        assert object != null;
89
90        object.setPublished(true);
91        this.repository.save(object);
92    }
93
94    @Override
95    public void unbind(final Invoice object) {
96        assert object != null;
97
98        Dataset dataset;
99
100       dataset = super.unbind(object, "code", "registrationTime", "dueDate", "quantity", "tax", "link");
101       dataset.put("totalAmount", object.totalAmount());
102       dataset.put("masterId", object.getSponsorship().getId());
103       dataset.put("isPublished", object.getSponsorship().isPublished());
104
105       super.getResponse().addData(dataset);
106   }
107}
```

```
45        masterId = super.getRequest().getData("masterId", int.class);
46        objects = this.repository.findManyTrainingSessionByTrainingModuleId(masterId);
47
48        super.getBuffer().addData(objects);
49    }
50
51    @Override
52    public void unbind(final TrainingSession object) {
53        assert object != null;
54
55        Dataset dataset;
56
57        dataset = super.unbind(object, "code", "location", "contactEmail");
58
59        if (object.isDraftMode()) {
60            Locale local = super.getRequest().getLocale();
61            dataset.put("draftMode", local.equals(Locale.ENGLISH) ? "Yes" : "Sí");
62        } else
63            dataset.put("draftMode", "No");
64
65        super.getResponse().addData(dataset);
66    }
67
68    @Override
69    public void unbind(final Collection<TrainingSession> object) {
70        assert object != null;
71
72        int masterId;
73
74        masterId = super.getRequest().getData("masterId", int.class);
75
76        super.getResponse().addGlobal("masterId", masterId);
```

Por último, tenemos el deleteService, que como se puede observar, nos baja el porcentaje porque nunca llega al unbind.

```java
public void authorise() {
    boolean status;
    int masterId;
    Invoice sponsorship;

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

    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");
    dataset.put("totalAmount", object.totalAmount());
    dataset.put("masterId", object.getSponsorship().getId());
    dataset.put("isPublished", object.getSponsorship().isPublished());

    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 se muestra una imagen con la batería de datos que se ha usado para probar todo el proyecto. 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.

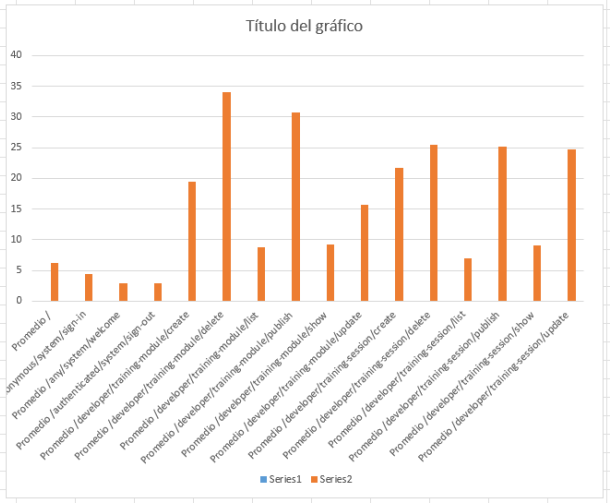
Para terminar este capítulo me gustaría mostrar todos los datos que se han usado para las pruebas. He usado el Excel que se ha añadido en la actualización del framework para tomar de ahí los datos.

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.

request-method	request-path	response-status	time
	Promedio /		6,241785
	Promedio /anonymous/system/sign-in		4,37312
	Promedio /any/system/welcome		2,847834
	Promedio /authenticated/system/sign-out		2,90015
	Promedio /developer/training-module/create		19,43748
	Promedio /developer/training-module/delete		33,98215
	Promedio /developer/training-module/list		8,807479
	Promedio /developer/training-module/publish		30,77915
	Promedio /developer/training-module/show		9,205264
	Promedio /developer/training-module/update		15,75239
	Promedio /developer/training-session/create		21,74964
	Promedio /developer/training-session/delete		25,44175
	Promedio /developer/training-session/list		6,951381
	Promedio /developer/training-session/publish		25,22635
	Promedio /developer/training-session/show		9,100666
	Promedio /developer/training-session/update		24,6721
	Promedio general		12,64411



Como se puede observar, ningún promedio supera los 50 ms, lo que es un resultado muy bueno, porque nos indica que las búsquedas se realizan de manera rápida. En este apartado tengo que comentar que la primera vez que realicé el test, observé que había un par de métodos que llegaban a los 50 ms, así que gracias a eso pude darme cuenta de que el código que había implementado no estaba refactorizado correctamente, porque me traía colecciones de datos enteras, y luego recorría todos esos datos para obtener los resultados, lo que en una base de datos mucho más grande provocaría mucha perdida de tiempo, así que opté por solucionar el código y poner todo lo necesario directamente en las querys.

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	12,6441063		Media	12,4331975	
Error típico	0,38597407		Error típico	0,38456645	
Mediana	8,2804		Mediana	8,0893	
Moda	8,2325		Moda	7,7308	
Desviación estándar	12,0890627		Desviación estándar	11,6961319	
Varianza de la muestra	146,145437		Varianza de la muestra	136,799501	
Curtosis	61,492175		Curtosis	34,7543068	
Coeficiente de asimetría	5,41108282		Coeficiente de asimetría	4,00865021	
Rango	185,0061		Rango	155,4288	
Mínimo	1,5008		Mínimo	1,661	
Máximo	186,5069		Máximo	157,0898	
Suma	12103,8683		Suma	20957,7077	
Cuenta	941		Cuenta	941	
Nivel de confianza(95,0%)	0,75743073		Nivel de confianza(95,0%)	0,75472499	
Interval (ms)	11,8866756	13,401537	Interval (ms)	11,6784725	13,1879225
Interval (s)	0,01188668	0,01340154	Interval (s)	0,01167847	0,01318792

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	12,64410632	12,4331975
Varianza (conocida)	146,145437	136,799501
Observaciones	941	941
Diferencia hipotética de las medias	0	
z	0,387091388	
P(Z<=z) una cola	0,349344277	
Valor crítico de z (una cola)	1,644853627	
Valor crítico de z (dos colas)	0,698688553	
Valor crítico de z (dos colas)	1,959963985	

Podemos observar que Alpha es 0.05, y que el p-value es 0.698… 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.

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

Before PC2			*After PC2*		
Media	12,7755109		Media	12,5370975	
Error típico	0,39163019		Error típico	0,38735775	
Mediana	8,3936		Mediana	8,1438	
Moda	9,6781		Moda	1,9404	
Desviación estándar	12,2662177		Desviación estándar	11,7810262	
Varianza de la muestra	150,460096		Varianza de la muestra	138,792577	
Curtosis	63,5362241		Curtosis	34,237388	
Coeficiente de asimetría	5,50562157		Coeficiente de asimetría	3,97976665	
Rango	190,5563		Rango	155,4288	
Mínimo	1,5458		Mínimo	1,661	
Máximo	192,1021		Máximo	157,0898	
Suma	12532,7762		Suma	12296,8152	
Cuenta	935		Cuenta	935	
Nivel de confianza(95,0%)	0,76853023		Nivel de confianza(95,0%)	0,76020303	
Interval (ms)	12,0069807	13,5440411	Interval (ms)	11,7768945	13,2973005
Interval (s)	0,01200698	0,01354404	Interval (s)	0,01177689	0,0132973

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		
	192,1021	*122,8571*
Media	12,5925246	12,4177036
Varianza (conocida)	150,460096	138,792577
Observaciones	930	930
Diferencia hipotética de las medias	0	
z	0,31720738	
P(Z<=z) una cola	0,37554313	
Valor crítico de z (una cola)	1,64485363	
Valor crítico de z (dos colas)	0,75108625	
Valor crítico de z (dos colas)	1,95996398	

Podemos observar que Alpha es 0.05, y que el p-value es 0.751… 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**, ninguno de los PCs muestra una diferencia significativa 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 ninguno 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.