

Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática


DP2-Reporte de Test Student 2



Grado en Ingeniería Informática – Ingeniería del Software

Diseño y Pruebas II

Curso 2023 – 2024

<u>Group:</u>	C1.016
<u>Repository:</u>	https://github.com/luchersol/Acme-SF-D04
<u>Student #2</u>	
	
UVUS: marcallop7	
Name: Marina Calero López	
Email: marcallop7@alum.us.es	
<u>Date:</u>	Sevilla Mayo 27, 2024

Índice de contenido

1. Pruebas funcionales	2
2. Pruebas de rendimiento	11

1. Pruebas funcionales

1.1 Contract

List-Mine:

Safe:

Se realiza un listado de los contratos del client1 (quien sí posee proyectos).

```
@Service
public class ClientContractListMineService extends AbstractService<Client, Contract> {

    // Internal state -----

    @Autowired
    private ClientContractRepository repository;

    // AbstractService interface -----

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

    @Override
    public void load() {
        Collection<Contract> contract;
        int id;

        id = this.getRequest().getPrincipal().getAccountId();
        contract = this.repository.findContractByClientId(id);
        super.getBuffer().addData(contract);
    }

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

        Dataset dataset;

        dataset = super.unbind(object, "code", "providerName", "customerName");
        super.getResponse().addData(dataset);
    }
}
```

Show:

Safe:

Se muestran los detalles de un contrato de client1, tanto publicado como no publicado.

Hack:

Se intentan realizar los siguiente hacking:

- Sin haber iniciado sesión, en la URL la he cambiado para intentar ver los detalles de un contrato del client1.

- Una vez iniciada sesión, cambiar la URL para ver los detalles de los contratos tanto publicados como no publicados.

- Cambiar la URL para ver los detalles de un contrato que no me pertenece, en este caso poner el id=110 que es un contrato que pertenece al client1.

- Cambiar la URL para ver un contrato con un id que no existe.

```
@Service
public class ClientContractShowService extends AbstractService<Client, Contract> {

    // Internal state -----

    @Autowired
    private ClientContractRepository repository;

    @Autowired
    private MoneyExchangeService moneyExchange;

    // AbstractService interface -----

    @Override
    public void authorise() {
        boolean status;
        int id;
        Contract contract;
        Client client;

        id = super.getRequest().getData("id", int.class);
        contract = this.repository.findOneContractById(id);
        client = contract == null ? null : contract.getClient();
        status = contract != null && super.getRequest().getPrincipal().hasRole(client);

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

    @Override
    public void load() {
        Contract contract;
        int id;

        id = super.getRequest().getData("id", int.class);
        contract = this.repository.findOneContractById(id);

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

    @Override
    public void unbind(final Contract contract) {
        assert contract != null;

        Collection<Project> projectAllPublish;
        SelectChoices choicesProject;
        Dataset dataset;
        Money moneyExchange;

        projectAllPublish = this.repository.findAllProjectsPublish();
        choicesProject = SelectChoices.from(projectAllPublish, "code", contract.getProject());

        dataset = super.unbind(contract, "code", "instantiationMoment", "providerName", "customerName", "goal", "budget", "draftNode");
        dataset.put("project", choicesProject.getSelected().getKey());
        dataset.put("projects", choicesProject);
        moneyExchange = this.moneyExchange.computeMoneyExchange(contract.getBudget());
        dataset.put("moneyExchange", moneyExchange);

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

Create:

Safe:

Se crea un nuevo contrato desde client1.

Se intentan realizar las acciones correctas:

- Crear un contrato con un "code" válido.

- Crear un contrato con un "providerName" con menos de 74 caracteres.
- Crear un contrato con un "customerName" con menos de 74 caracteres.
- Crear un contrato con un "goal" con menos de 100 caracteres.
- Crear un contrato con un "budget" con una moneda válida y con un número positivo.

Se intentan realizar las acciones incorrectas:

- Crear un contrato con un "proyecto" al que no esté asociado a ese contrato.
- Crear un contrato para todos los campos vacíos.
- Crear un contrato con un "code" no válido.
- Crear un contrato con un "code" ya existente.
- Crear un contrato con un "providerName" con más de 75 caracteres.
- Crear un contrato con un "customerName" con más de 75 caracteres.
- Crear un contrato con un "goal" con más de 100 caracteres.
- Crear un contrato con un "budget" que no es válido, es decir, con caracteres.
- Crear un contrato con un "budget" con un número negativo.
- Crear un contrato con un "budget" con una moneda que no es válida.

```

1 @Service
2 public class ClientContractCreateService extends AbstractAntispamService<Client, Contract> {
3
4     // Internal state -----
5
6     @Autowired
7     private ClientContractRepository repository;
8
9     // AbstractService interface -----
10
11     @Override
12     public void authorise() {
13         super.getResponse().setAuthorised(true);
14     }
15
16     @Override
17     public void load() {
18         Contract contract;
19         Client client;
20         Date moment;
21
22         moment = MomentHelper.getCurrentMoment();
23
24         client = this.repository.findOneClientById(super.getRequest().getPrincipal().getActiveRoleId());
25
26         contract = new Contract();
27         contract.setInstantiationMoment(moment);
28         contract.setDraftMode(true);
29         contract.setClient(client);
30
31         super.getBuffer().addData(contract);
32     }
33
34     @Override
35     public void bind(final Contract contract) {
36         assert contract != null;
37
38         int projectId;
39         Project project;
40
41         projectId = super.getRequest().getData("project", int.class);
42         project = this.repository.findOneProjectById(projectId);
43
44         super.bind(contract, "code", "project", "providerName", "customerName", "goal", "budget");
45         contract.setProject(project);
46     }
47
48     @Override
49     public void validate(final Contract contract) {
50         assert contract != null;
51
52         boolean state;
53
54         if (!super.getBuffer().getErrors().hasErrors("code")) {
55             state = !this.repository.existsByCode(contract.getCode());
56             super.state(state, "code", "client.contract.form.error.code");
57         }
58
59         if (!super.getBuffer().getErrors().hasErrors("budget")) {
60             state = contract.getBudget().getAmount() >= 0;
61             super.state(state, "budget", "client.contract.form.error.budget");
62         }
63
64         if (!super.getBuffer().getErrors().hasErrors("project")) {
65             Boolean isDraftMode = this.repository.ProjectIsDraftMode(contract.getProject().getId());
66             super.state(!isDraftMode, "project", "client.contract.form.error.project");
67         }
68
69         super.validateSpam(contract);
70     }
71
72     @Override
73     public void perform(final Contract contract) {
74         assert contract != null;
75         //
76         //     Date moment;
77         //
78         //     moment = MomentHelper.getCurrentMoment();
79         //     contract.setInstantiationMoment(moment);
80
81         this.repository.save(contract);
82     }
83
84     @Override
85     public void unbind(final Contract contract) {
86         assert contract != null;
87
88         Collection<Project> projectAllPublish;
89         SelectChoices choices;
90         Dataset dataset;
91
92         projectAllPublish = this.repository.findAllProjectsPublish();
93
94         choices = SelectChoices.from(projectAllPublish, "code", contract.getProject());
95
96         dataset = super.unbind(contract, "code", "instantiationMoment", "project", "providerName", "customerName", "goal", "budget", "draftMode");
97         dataset.put("project", choices.getSelected().getKey());
98         dataset.put("projects", choices);
99
100         super.getResponse().addData(dataset);
101     }
102 }

```

Delete:

Safe:

Se borra un contrato no publicado de client1.

Hack:

Se intentan realizar los siguientes hacking:

- Sin haber iniciado sesión, en la URL la he cambiado para intentar borrar un contrato del client1.
- Una vez iniciada sesión, cambiar la URL para borrar contratos no publicados.
- Cambiar la URL para borrar contratos publicados.
- Cambiar la URL para borrar contrato con un id que no existe.
- Cambiar la URL para borrar contrato con un id que pertenece al client2, es decir, poner el id=110.

```
@Service
public class ClientContractDeleteService extends AbstractService<Client, Contract> {

    // Internal state -----

    @Autowired
    private ClientContractRepository repository;

    @Autowired
    private MoneyExchangeService moneyExchange;

    // AbstractService interface -----

    @Override
    public void authorise() {
        boolean status;
        int masterId;
        Contract contract;
        Client client;

        masterId = this.getRequest().getData("id", int.class);
        contract = this.repository.findOneContractById(masterId);
        client = contract == null ? null : contract.getClient();
        status = contract != null && contract.getDraftMode() && //
            super.getRequest().getPrincipal().hasRole(client);

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

    @Override
    public void load() {
        Contract contract;
        int id;

        id = super.getRequest().getData("id", int.class);
        contract = this.repository.findOneContractById(id);

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

    @Override
    public void bind(final Contract contract) {
        assert contract != null;

        int projectId;
        Project project;

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

        super.bind(contract, "code", "instantiationMoment", "providerName", "customerName", "goal", "budget");
        contract.setProject(project);
    }

    @Override
    public void validate(final Contract contract) {
        assert contract != null;
    }
}
```



```

@Override
public void perform(final Contract contract) {
    assert contract != null;

    Collection<ProgressLog> progressLogs;

    progressLogs = this.repository.findManyProgressLogsId(contract.getId());
    this.repository.deleteAll(progressLogs);
    this.repository.delete(contract);
}

@Override
public void unbind(final Contract contract) {
    assert contract != null;

    Collection<Project> projectAllPublish;
    SelectChoices choices;
    Dataset dataset;
    Money moneyExchange;

    projectAllPublish = this.repository.findAllProjectsPublish();
    choices = SelectChoices.from(projectAllPublish, "code", contract.getProject());

    dataset = super.unbind(contract, "code", "instantiationMoment", "providerName", "customerName", "goal", "budget", "draftMode");
    dataset.put("project", choices.getSelected().getKey());
    dataset.put("projects", choices);
    moneyExchange = this.moneyExchange.computeMoneyExchange(contract.getBudget());
    dataset.put("moneyExchange", moneyExchange);

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

```

Update:

Safe:

Se actualiza un contrato no publicado de client1.

Se intentan realizar las acciones correctas:

- Actualizar un contrato sin modificar nada.
- Actualizar un contrato con un "code" válido.
- Actualizar un contrato sin modificar el "code".
- Actualizar un contrato con un "providerName" con menos de 74 caracteres.
- Actualizar un contrato con un "customerName" con menos de 74 caracteres.
- Actualizar un contrato con un "goal" con menos de 100 caracteres.
- Actualizar un contrato con un "budget" con una moneda válida y con un número positivo.

Se intentan realizar las acciones incorrectas:

contrato.

- Actualizar un contrato con un "proyecto" al que no esté asociado a ese

- Actualizar un contrato para todos los campos vacíos.
- Actualizar un contrato con un "code" no válido.
- Actualizar un contrato con un "code" ya existente.
- Actualizar un contrato con un "providerName" con más de 75 caracteres.
- Actualizar un contrato con un "customerName" con más de 75 caracteres.
- Actualizar un contrato con un "goal" con más de 100 caracteres.
- Actualizar un contrato con un "budget" que no es válido, es decir, con caracteres.
- Actualizar un contrato con un "budget" con un número negativo.

- Actualizar un contrato con un “budget” con una moneda que no es válida.

Hack:

Se intentan realizar los siguientes hacking:

- Sin haber iniciado sesión, en la URL la he cambiado para intentar actualizar un contrato del client1.
- Una vez iniciada sesión, cambiar la URL para actualizar contratos no publicados.
- Cambiar la URL para actualizar los contratos publicados.
- Cambiar la URL para actualizar un contrato con un id que no existe.
- Cambiar la URL para actualizar un contrato con un id que pertenece al client2, es decir, poner el id=110.

```

1  @Service
2  public class ClientContractUpdateService extends AbstractAntiSpamService<Client, Contract> {
3
4      // Internal state -----
5
6      @Autowired
7      private ClientContractRepository repository;
8
9      @Autowired
10     private MoneyExchangeService moneyExchange;
11
12     // AbstractService interface -----
13
14     @Override
15     public void authorise() {
16         boolean status;
17         int id;
18         Contract contract;
19         Client client;
20
21         id = super.getRequest().getData("id", int.class);
22
23         contract = this.repository.findOneContractById(id);
24         client = contract == null ? null : contract.getClient();
25         status = contract != null && contract.getDraftMode() && super.getRequest().getPrincipal().hasRole(client);
26
27         super.getResponse().setAuthorised(status);
28     }
29
30     @Override
31     public void load() {
32         Client client;
33         Contract contract;
34         int clientId;
35
36         clientId = super.getRequest().getPrincipal().getActiveRoleId();
37         client = this.repository.findOneClientById(clientId);
38
39         contract = new Contract();
40         contract.setDraftMode(true);
41         contract.setClient(client);
42
43         super.getBuffer().addData(contract);
44     }
45
46     @Override
47     public void bind(final Contract contract) {
48         assert contract != null;
49
50         super.bind(contract, "code", "instantiationMoment", "project", "providerName", "customerName", "goal", "budget");
51     }
52 }

```

```

@Override
public void bind(final Contract contract) {
    assert contract != null;

    super.bind(contract, "code", "instantiationMoment", "project", "providerName", "customerName", "goal", "budget");
}

@Override
public void validate(final Contract contract) {
    assert contract != null;

    boolean state;

    if (!super.getBuffer().getErrors().hasErrors("code")) {
        state = !this.repository.existsOtherByCodeAndId(contract.getCode(), contract.getId());
        super.state(state, "code", "client.contract.form.error.code");
    }

    if (!super.getBuffer().getErrors().hasErrors("budget")) {
        state = contract.getBudget().getAmount() >= 0;
        super.state(state, "budget", "client.contract.form.error.budget");
    }

    if (!super.getBuffer().getErrors().hasErrors("budget")) {
        state = Arrays.asList(this.repository.findAcceptedCurrencies().split(",")).contains(contract.getBudget().getCurrency());
        super.state(state, "budget", "client.contract.form.error.invalid-currency");
    }

    if (!super.getBuffer().getErrors().hasErrors("project")) {}
    boolean isDraftMode = this.repository.ProjectIsDraftMode(contract.getProject().getId());
    super.state(isDraftMode, "project", "client.contract.form.error.project");
}

super.validateSpan(contract);
}

@Override
public void perform(final Contract contract) {
    assert contract != null;

    this.repository.save(contract);
}

@Override
public void unbind(final Contract contract) {
    assert contract != null;

    Collection<Project> projectAllPublish;
    SelectChoices choicesProject;
    Dataset dataset;
    Money moneyExchange;

    projectAllPublish = this.repository.findAllProjectsPublish();
    choicesProject = SelectChoices.from(projectAllPublish, "code", contract.getProject());

    dataset = super.unbind(contract, "code", "instantiationMoment", "providerName", "customerName", "goal", "budget", "draftMode");
    dataset.put("project", choicesProject.getSelected().getKey());
    dataset.put("projects", choicesProject);
    moneyExchange = this.moneyExchange.computeMoneyExchange(contract.getBudget());
    dataset.put("moneyExchange", moneyExchange);

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

```

Publish:

Safe:

Se publica un contrato de client1 que no tiene errores fatales, todas sus “progress-log” están publicadas y tiene al menos un “progress-log” asociada.

Se intentan realizar las acciones correctas:

- Publicar un contrato sin modificar nada.
- Publicar un contrato con un “code” válido.
- Publicar un contrato sin modificar el “code”.
- Publicar un contrato con un “providerName” con menos de 74 caracteres.
- Publicar un contrato con un “customerName” con menos de 74 caracteres.
- Publicar un contrato con un “goal” con menos de 100 caracteres.

- Publicar un contrato con un “budget” con una moneda válida y con un número positivo.

Se intentan realizar las acciones incorrectas:

- Publicar un contrato con un “proyecto” al que no esté asociado a ese contrato.
- Publicar un contrato para todos los campos vacíos.
- Publicar un contrato con un “code” no válido.
- Publicar un contrato con un “code” ya existente.
- Publicar un contrato con un “providerName” con más de 75 caracteres.
- Publicar un contrato con un “customerName” con más de 75 caracteres.
- Publicar un contrato con un “goal” con más de 100 caracteres.
- Publicar un contrato con un “budget” que no es válido, es decir, con caracteres.
- Publicar un contrato con un “budget” con un número negativo.
- Publicar un contrato con un “budget” con una moneda que no es válida.
- Publicar un contrato con un “budget” en el cual excede el coste total del proyecto asociado.
- Publicar un contrato con “progress-log” asociados sin estar publicados.
- Publicar un contrato con ningún “progress-log” asociado.

Hack:

Se intentan realizar los siguientes hacking:

- Sin haber iniciado sesión, en la URL la he cambiado para intentar publicar un contrato del client1.
- Una vez iniciada sesión, cambiar la URL para publicar contratos no publicados.

- Cambiar la URL para publicar los contratos publicados.
- Cambiar la URL para publicar un contrato con un id que no existe.
- Cambiar la URL para publicar un contrato con un id que pertenece al client2, es decir, poner el id=110.

```
@Service
public class ClientContractPublishService extends AbstractAntiSpamService<Client, Contract> {

    // Internal state -----

    @Autowired
    private ClientContractRepository repository;

    @Autowired
    private MoneyExchangeService moneyExchange;

    // AbstractService interface -----

    @Override
    public void authorise() {
        boolean status;
        int contractId;
        Contract contract;
        Client client;

        contractId = super.getRequest().getData("id", int.class);
        contract = this.repository.findOneContractById(contractId);
        client = contract == null ? null : contract.getClient();
        status = contract != null && contract.getDraftMode() && super.getRequest().getPrincipal().hasRole(client);

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

    @Override
    public void load() {
        Contract contract;
        int id;

        id = super.getRequest().getData("id", int.class);
        contract = this.repository.findOneContractById(id);

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

    @Override
    public void bind(final Contract contract) {
        assert contract != null;

        int projectId;
        Project project;

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

        super.bind(contract, "code", "instantiationMoment", "providerName", "customerName", "goal", "budget");
        contract.setProject(project);
    }

    @Override
    public void validate(final Contract contract) {
        assert contract != null;

        boolean state;

        if (!super.getBuffer().getErrors().hasErrors("code")) {
            state = !this.repository.existsOtherByCodeAndId(contract.getCode(), contract.getId());
            super.state(state, "code", "client.contract.form.error.code");
        }

        if (!super.getBuffer().getErrors().hasErrors("budget")) {
            state = contract.getBudget().getAmount() >= 0;
            super.state(state, "budget", "client.contract.form.error.budget");
        }

        super.state(!this.repository.findManyProgressLogByMasterId(contract.getId()).isEmpty(), "code", "client.contract.form.error.publish");
        boolean allContractInDraftMode = this.repository.areAllProgressLogPublished(contract.getId());
        super.state(allContractInDraftMode, "code", "client.contract.form.error.allpublish");

        if (!super.getBuffer().getErrors().hasErrors("budget")) {
            Collection<Money> budgets = this.repository.areAllBudgetContractExcedCostProject(contract.getProject().getId());
            state = budgets.stream().map(x -> x.getAmount()).mapToDouble(x -> x.doubleValue()).sum() + contract.getBudget().getAmount() < contract.getProject().getCost().getAmount();
            super.state(state, "budget", "client.contract.form.error.budgetExcedCostProject");
        }

        if (!super.getBuffer().getErrors().hasErrors("budget")) {
            state = Arrays.asList(this.repository.findAcceptedCurrencies().split(",")).contains(contract.getBudget().getCurrency());
            super.state(state, "budget", "client.contract.form.error.invalid-currency");
        }

        if (!super.getBuffer().getErrors().hasErrors("project")) {
            Boolean isDraftMode = this.repository.ProjectIsDraftMode(contract.getProject().getId());
            super.state(!isDraftMode, "project", "client.contract.form.error.project");
        }

        super.validateSpam(contract);
    }
}
```

```

@Override
public void perform(final Contract contract) {
    assert contract != null;

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

@Override
public void unbind(final Contract contract) {
    assert contract != null;

















    Collection<Project> projectAllPublish;
    SelectChoices choicesProject;
    Dataset dataset;
    Money moneyExchange;

    projectAllPublish = this.repository.findAllProjectsPublish();
    choicesProject = SelectChoices.from(projectAllPublish, "code", contract.getProject());

    dataset = super.unbind(contract, "code", "instantiationMoment", "providerName", "customerName", "goal", "budget", "draftMode");
    dataset.put("project", choicesProject.getSelected().getKey());
    dataset.put("projects", choicesProject);
    moneyExchange = this.moneyExchange.computeMoneyExchange(contract.getBudget());
    dataset.put("moneyExchange", moneyExchange);

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

```

▼  acme.features.client.contract	 95,0 %	1.355	72	1.427
>  ClientContractController.java	 100,0 %	36	0	36
>  ClientContractCreateService.java	 93,8 %	244	16	260
>  ClientContractDeleteService.java	 92,9 %	209	16	225
>  ClientContractListMineService.java	 92,6 %	50	4	54
>  ClientContractPublishService.java	 96,0 %	384	16	400
>  ClientContractShowService.java	 97,1 %	134	4	138
>  ClientContractUpdateService.java	 94,9 %	298	16	314

1.2 Progress Log

List:

Safe:

Se realiza un listado del registro de progreso de un contrato con registros de progresos asociadas y luego un contrato sin registros de progresos asociadas.

Hack:

Se intentan realizar los siguiente hacking:

- Sin haber iniciado sesión, cambiar la URL para mostrar el listado de los registros de progresos con un masterId del cliente1.
- Una vez iniciada sesión, cambiar la URL para mostrar el listado de los registros de progresos con un masterId del cliente1 de contratos publicados.

- Cambiar la URL para mostrar el listado de los registros de progresos con un masterId del cliente1 de contratos no publicados.
- Cambiar la URL para mostrar el listado de los registros de progresos con un masterId del cliente1 de contratos que no existen.
- Cambiar la URL para mostrar el listado de los registros de progresos con un masterId del cliente1 de contratos que no pertenecen a dicho contrato.

```

@Service
public class ClientProgressLogListService extends AbstractService<Client, ProgressLog> {

    // Internal state -----

    @Autowired
    private ClientProgressLogRepository repository;

    // AbstractService interface -----

    @Override
    public void authorise() {
        boolean status;
        int masterId;
        Contract contract;

        masterId = super.getRequest().getData("masterId", int.class);
        contract = this.repository.findOneContractById(masterId);
        status = contract != null && (!contract.getDraftMode() || super.getRequest().getPrincipal().hasRole(contract.getClient()));

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

    @Override
    public void load() {
        Collection<ProgressLog> progressLogs;
        int masterId;

        masterId = super.getRequest().getData("masterId", int.class);
        progressLogs = this.repository.findManyProgressLogsByMasterId(masterId);

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

    @Override
    public void unbind(final ProgressLog progressLog) {
        assert progressLog != null;

        Dataset dataset;

        dataset = super.unbind(progressLog, "recordId", "completeness", "comment", "registrationMoment", "responsiblePerson");

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

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

        int masterId;
        Contract contract;
        Boolean showCreate;

        masterId = super.getRequest().getData("masterId", int.class);
        contract = this.repository.findOneContractById(masterId);
        showCreate = super.getRequest().getPrincipal().hasRole(contract.getClient()) && contract.getDraftMode();

        super.getResponse().addGlobal("masterId", masterId);
        super.getResponse().addGlobal("showCreate", showCreate);
    }
}

```

Show:

Safe:

Se muestran los detalles de un registro de procesos de client1 que están tanto publicados como no publicados.

Hack:

Se intentan realizar los siguiente hacking:

- Sin haber iniciado sesión, cambiar la URL para mostrar los detalles de los registros de progresos con un id perteneciente al cliente1.
- Una vez iniciada sesión, cambiar la URL para mostrar los detalles de los registros de progresos publicados con un id perteneciente al cliente1.
- Cambiar la URL para mostrar los detalles de los registros de progresos no publicados con un id perteneciente al cliente1.
- Cambiar la URL para mostrar los detalles de los registros de progresos que no existen con un id perteneciente al cliente1.
- Cambiar la URL para mostrar los detalles de los registros de progresos con un id que no pertenece al cliente1.

```

4 @Service
5 public class ClientProgressLogShowService extends AbstractService<Client, ProgressLog> {
6
7     // Internal state -----
8
9     @Autowired
10    private ClientProgressLogRepository repository;
11
12    // AbstractService interface -----
13
14    @Override
15    public void authorise() {
16        boolean status;
17        int progressLogId;
18        Contract contract;
19
20        progressLogId = super.getRequest().getData("id", int.class);
21        contract = this.repository.findOneContractByProgressLogId(progressLogId);
22        status = contract != null && super.getRequest().getPrincipal().hasRole(contract.getClient());
23
24        super.getResponse().setAuthorised(status);
25    }
26
27    @Override
28    public void load() {
29        ProgressLog progressLog;
30        int id;
31
32        id = super.getRequest().getData("id", int.class);
33        progressLog = this.repository.findOneProgressLogById(id);
34
35        super.getBuffer().addData(progressLog);
36    }
37
38    @Override
39    public void unbind(final ProgressLog progressLogs) {
40        assert progressLogs != null;
41
42        Dataset dataset;
43
44        dataset = super.unbind(progressLogs, "recordId", "completeness", "comment", "registrationMoment", "responsiblePerson", "draftMode");
45        dataset.put("masterId", progressLogs.getId());
46        dataset.put("draftMode", progressLogs.isDraftMode());
47
48        super.getResponse().addData(dataset);
49    }
50 }

```

Create:

Safe:

Se crea un nuevo registro de progreso desde client1.

Se intentan realizar las acciones correctas:

- Crear un registro de progreso con un "recordId" con formato válido.
- Crear un registro de progreso con un "completeness" con un valor decimal entre 0 y 100.

- Crear un registro de progreso con un “comment” de 100 caracteres.
- Crear un registro de progreso con una “responsiblePerson” de 75 caracteres.

Se intentan realizar las acciones incorrectas:

- Crear un registro de progreso con todos los campos vacíos.
- Crear un registro de progreso con un “recordId” ya existente.
- Crear un registro de progreso con un “recordId” que tenga formato incorrecto .
- Crear un registro de progreso con un “completeness” con valor negativo.
- Crear un registro de progreso con un “completeness” con valor mayor que 100.
- Crear un registro de progreso con un “completeness” escribiendo una cadena de texto.
- Crear un registro de progreso con un “completeness” escribiendo una cadena de texto.
- Crear un registro de progreso con un “completeness” escribiendo un número entero.
- Crear un registro de progreso con un “comment” de más de 100 caracteres.
- Crear un registro de progreso con una “responsiblePerson” de más de 75 caracteres.

```

5
5 @Service
7 public class ClientProgressLogCreateService extends AbstractAntiSpamService<Client, ProgressLog> {
8
9     // Internal state -----
10
11     @Autowired
12     private ClientProgressLogRepository repository;
13
14     // AbstractService interface -----
15
16     @Override
17     public void authorise() {
18         boolean status;
19         int masterId;
20         Contract contract;
21
22         masterId = super.getRequest().getData("masterId", int.class);
23         contract = this.repository.findOneContractById(masterId);
24         status = contract != null && contract.getDraftMode() && super.getRequest().getPrincipal().hasRole(contract.getClient());
25
26         super.getResponse().setAuthorised(status);
27     }
28
29     @Override
30     public void load() {
31         ProgressLog progressLog;
32         int masterId;
33         Contract contract;
34
35         Date moment;
36
37         moment = MomentHelper.getCurrentMoment();
38
39         masterId = super.getRequest().getData("masterId", int.class);
40         contract = this.repository.findOneContractById(masterId);
41
42         progressLog = new ProgressLog();
43         progressLog.setRecordId("");
44         progressLog.setCompleteness(0.00);
45         progressLog.setComment("");
46         progressLog.setRegistrationMoment(moment);
47         progressLog.setResponsiblePerson("");
48         progressLog.setContract(contract);
49         progressLog.setDraftMode(true);
50         super.getBuffer().addData(progressLog);
51     }
52
53     @Override
54     public void bind(final ProgressLog progressLog) {
55         assert progressLog != null;
56
57         super.bind(progressLog, "recordId", "completeness", "comment", "responsiblePerson");
58     }
59
60     @Override
61     public void validate(final ProgressLog progressLog) {
62         assert progressLog != null;
63
64         boolean state;
65
66         if (!super.getBuffer().getErrors().hasErrors("recordId")) {
67             state = !this.repository.existsByCode(progressLog.getRecordId());
68             super.state(state, "recordId", "client.progress-log.form.error.code");
69         }
70
71         super.validateSpam(progressLog);
72     }
73
74     @Override
75     public void unbind(final ProgressLog progressLog) {
76         assert progressLog != null;
77
78         Dataset dataset;
79
80         dataset = super.unbind(progressLog, "recordId", "completeness", "comment", "registrationMoment", "responsiblePerson", "draftMode");
81         dataset.put("masterId", super.getRequest().getData("masterId", int.class));
82
83         super.getResponse().addData(dataset);
84     }
85 }

```

Delete:

Safe:

Se borra un registro de progreso no publicado de client1.

Hack:

Se intentan realizar los siguientes hacking:

- Sin tener la sesión iniciada, se cambia la URL para intentar eliminar un registro de progreso que pertenece a client1-
- Una vez iniciada la sesión, modificar la URL para borrar un registro de progreso no publicado.
- Modificar la URL para borrar un registro de progreso publicado.
- Modificar la URL para borrar un registro de progreso con un id que no existe.
- Modificar la URL para borrar un registro de progreso con un id que no pertenece al client1.

```
3 @Service
4 public class ClientProgressLogDeleteService extends AbstractService<Client, ProgressLog> {
5
6     // Internal state -----
7
8     @Autowired
9     private ClientProgressLogRepository repository;
10
11     // AbstractService interface -----
12
13     @Override
14     public void authorise() {
15         boolean status;
16         int progressLogId;
17         Contract contract;
18
19         progressLogId = super.getRequest().getData("id", int.class);
20         contract = this.repository.findOneContractByProgressLogId(progressLogId);
21         status = contract != null && contract.getDraftMode() && super.getRequest().getPrincipal().hasRole(contract.getClient());
22
23         super.getResponse().setAuthorised(status);
24     }
25
26     @Override
27     public void load() {
28         ProgressLog progressLog;
29         int id;
30
31         id = super.getRequest().getData("id", int.class);
32         progressLog = this.repository.findOneProgressLogById(id);
33
34         super.getBuffer().addData(progressLog);
35     }
36
37     @Override
38     public void bind(final ProgressLog progressLog) {
39         assert progressLog != null;
40
41         super.bind(progressLog, "recordId", "completeness", "comment", "registrationMoment", "responsiblePerson");
42     }
43
44     @Override
45     public void validate(final ProgressLog progressLog) {
46         assert progressLog != null;
47     }
48
49     @Override
50     public void perform(final ProgressLog progressLog) {
51         assert progressLog != null;
52
53         this.repository.delete(progressLog);
54     }
55
56     @Override
57     public void unbind(final ProgressLog progressLog) {
58         assert progressLog != null;
59
60         Dataset dataset;
61
62         dataset = super.unbind(progressLog, "recordId", "completeness", "comment", "registrationMoment", "responsiblePerson", "draftMode");
63         dataset.put("masterId", progressLog.getContract().getId());
64         dataset.put("draftMode", progressLog.getContract().getDraftMode());
65
66         super.getResponse().addData(dataset);
67     }
68 }
69
70 }
```

Update:

Safe:

Se actualiza un registro de progreso no publicado de client1.

Se intentan realizar las acciones correctas:

- Actualizar un registro de progreso con formato válido.
- Actualizar un registro de progreso con un "completeness" con un valor decimal entre 0 y 100.
- Actualizar un registro de progreso con un "comment" de 100 caracteres.
- Actualizar un registro de progreso con una "responsiblePerson" de 75 caracteres.

Se intentan realizar las acciones incorrectas:

- Actualizar un registro de progreso con todos los campos vacíos.
- Actualizar un registro de progreso con un "recordId" ya existente.
- Actualizar un registro de progreso con un "recordId" que tenga formato incorrecto .
- Actualizar un registro de progreso con un "completeness" con valor negativo.
- Actualizar un registro de progreso con un "completeness" con valor mayor que 100.
- Actualizar un registro de progreso con un "completeness" escribiendo una cadena de texto.
- Actualizar un registro de progreso con un "completeness" escribiendo una cadena de texto.
- Actualizar un registro de progreso con un "completeness" escribiendo un número entero.
- Actualizar un registro de progreso con un "comment" de más de 100 caracteres.

- Actualizar un registro de progreso con una “responsiblePerson” de más de 75 caracteres.

Hack:

Se intentan realizar los siguientes hacking:

- Sin tener la sesión iniciada, se cambia la URL para intentar actualizar un registro de progreso que pertenece a client1.

- Una vez iniciada la sesión, modificar la URL para actualizar un registro de progreso no publicado.

- Modificar la URL para actualizar un registro de progreso publicado.

- Modificar la URL para actualizar un registro de progreso con un id que no existe.

- Modificar la URL para actualizar un registro de progreso con un id que no pertenece al client1.

```
1 @Service
2 public class ClientProgressLogUpdateService extends AbstractAntiSpamService<Client, ProgressLog> {
3
4     // Internal state -----
5
6     @Autowired
7     private ClientProgressLogRepository repository;
8
9     // AbstractService interface -----
10
11     @Override
12     public void authorise() {
13         boolean status;
14         int progressLogId;
15         Contract contract;
16
17         progressLogId = super.getRequest().getData("id", int.class);
18         contract = this.repository.findOneContractByProgressLogId(progressLogId);
19         status = contract != null && contract.getDraftMode() && super.getRequest().getPrincipal().hasRole(contract.getClient());
20
21         super.getResponse().setAuthorised(status);
22     }
23
24     @Override
25     public void load() {
26         ProgressLog progressLog;
27         int id;
28
29         id = super.getRequest().getData("id", int.class);
30         progressLog = this.repository.findOneProgressLogById(id);
31
32         super.getBuffer().addData(progressLog);
33     }
34
35     @Override
36     public void bind(final ProgressLog progressLog) {
37         assert progressLog != null;
38
39         super.bind(progressLog, "recordId", "completeness", "comment", "registrationMoment", "responsiblePerson");
40     }
41
42     @Override
43     public void validate(final ProgressLog progressLog) {
44         assert progressLog != null;
45
46         boolean state;
47
48         if (!super.getBuffer().getErrors().hasErrors("recordId")) {
49             state = !this.repository.existsOtherByCodeAndId(progressLog.getRecordId(), progressLog.getId());
50             super.state(state, "recordId", "client.progress-log.form.error.code");
51         }
52
53         super.validateSpam(progressLog);
54     }
55 }
```

```

@Override
public void perform(final ProgressLog progressLog) {
    assert progressLog != null;

    this.repository.save(progressLog);
}

@Override
public void unbind(final ProgressLog progressLog) {
    assert progressLog != null;

    Dataset dataset;

    dataset = super.unbind(progressLog, "recordId", "completeness", "comment", "registrationMoment", "responsiblePerson", "draftMode");
    dataset.put("masterId", progressLog.getContract().getId());

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

```

Publish:

Safe:

Se publica un registro de progreso de client1.

Se intentan realizar las acciones correctas:

- Publicar un registro de progreso con formato válido.
- Publicar un registro de progreso con un “completeness” con un valor decimal entre 0 y 100.
- Publicar un registro de progreso con un “comment” de 100 caracteres.
- Publicar un registro de progreso con una “responsiblePerson” de 75 caracteres.

Se intentan realizar las acciones incorrectas:

















- Publicar un registro de progreso con todos los campos vacíos.
- Publicar un registro de progreso con un “recordId” ya existente.
- Publicar un registro de progreso con un “recordId” que tenga formato incorrecto .
- Publicar un registro de progreso con un “completeness” con valor negativo.
- Publicar un registro de progreso con un “completeness” con valor mayor que 100.
- Publicar un registro de progreso con un “completeness” escribiendo una cadena de texto.
- Publicar un registro de progreso con un “completeness” escribiendo una cadena de texto.

- Publicar un registro de progreso con un “completeness” escribiendo un número entero.
- Publicar un registro de progreso con un “comment” de más de 100 caracteres.
- Publicar un registro de progreso con una “responsiblePerson” de más de 75 caracteres.

Hack:

Se intentan realizar los siguientes hacking:

- Sin tener la sesión iniciada, se cambia la URL para intentar publicar un registro de progreso que pertenece a client1.
- Una vez iniciada la sesión, modificar la URL para publicar un registro de progreso no publicado.
- Modificar la URL para publicar un registro de progreso publicado.
- Modificar la URL para publicar un registro de progreso con un id que no existe.
- Modificar la URL para publicar un registro de progreso con un id que no pertenece al client1.

▼  acme.features.client.progressLogs	 92,8 %	974	76	1.050
>  ClientProgressLogController.java	 100,0 %	35	0	35
>  ClientProgressLogCreateService.java	 92,5 %	197	16	213
>  ClientProgressLogDeleteService.java	 90,4 %	151	16	167
>  ClientProgressLogListService.java	 94,4 %	135	8	143
>  ClientProgressLogPublishService.java	 91,7 %	177	16	193
>  ClientProgressLogShowService.java	 96,3 %	105	4	109
>  ClientProgressLogUpdateService.java	 91,6 %	174	16	190

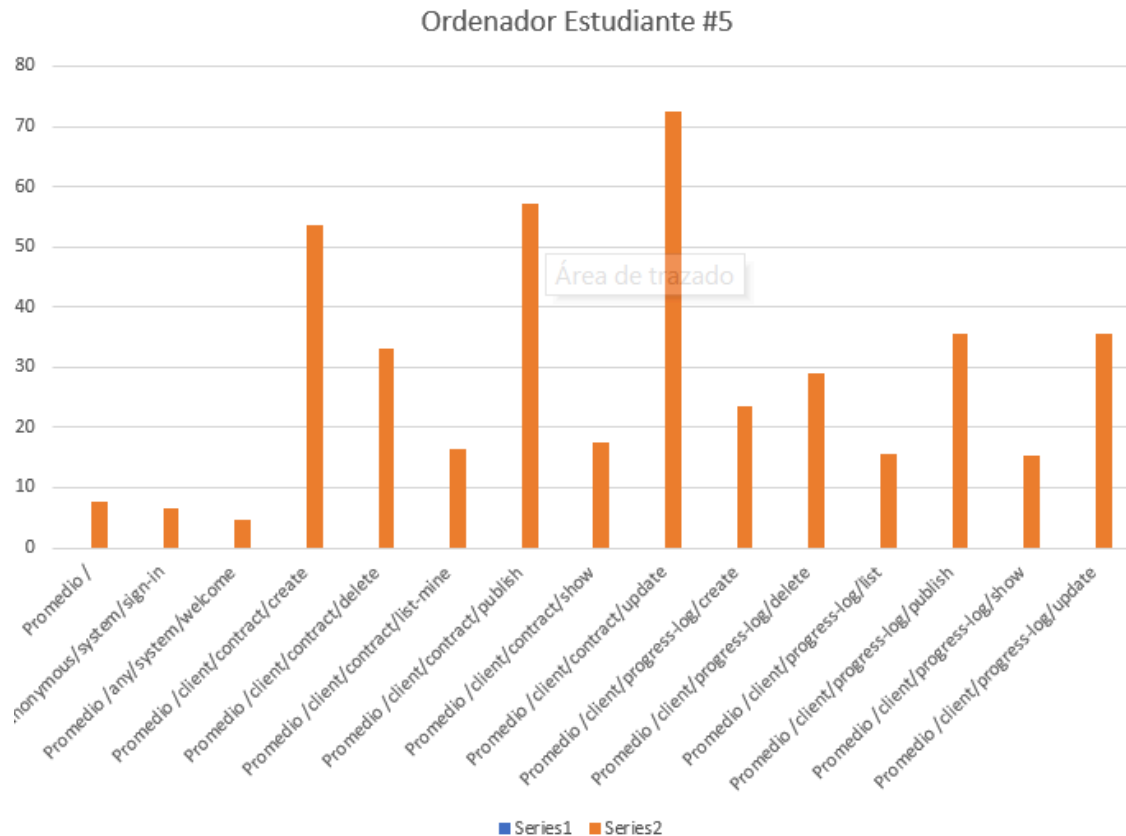
2. Pruebas de rendimiento

En esta sección hablaremos del rendimiento de nuestro código, analizando los métodos más costosos así como haciendo una comparativa de su ejecución en dos ordenadores diferentes.

Aquí podemos observar unas estadísticas generales del resultado de haber ejecutado los tests:

Rendimiento Estudiante #2

request-method	request-path	response-status	time
	Promedio /		5,46782593
	Promedio /anonymous/system/sign-in		4,94657592
	Promedio /any/system/welcome		3,22336093
	Promedio /client/contract/create		33,9284892
	Promedio /client/contract/delete		21,2813
	Promedio /client/contract/list-mine		11,2616826
	Promedio /client/contract/publish		37,7826931
	Promedio /client/contract/show		12,7409242
	Promedio /client/contract/update		45,0171629
	Promedio /client/progress-log/create		15,9832543
	Promedio /client/progress-log/delete		17,0035333
	Promedio /client/progress-log/list		10,0991808
	Promedio /client/progress-log/publish		24,8072186
	Promedio /client/progress-log/show		10,6880792
	Promedio /client/progress-log/update		23,5072839
	Promedio general		16,4507065



Ahora sí, podemos comentar los resultados de las estadísticas. Comenzamos diciendo que hemos superado el objetivo de tener un promedio inferior a 1 segundo, ya que obtuvimos una media de 16 milisegundos.

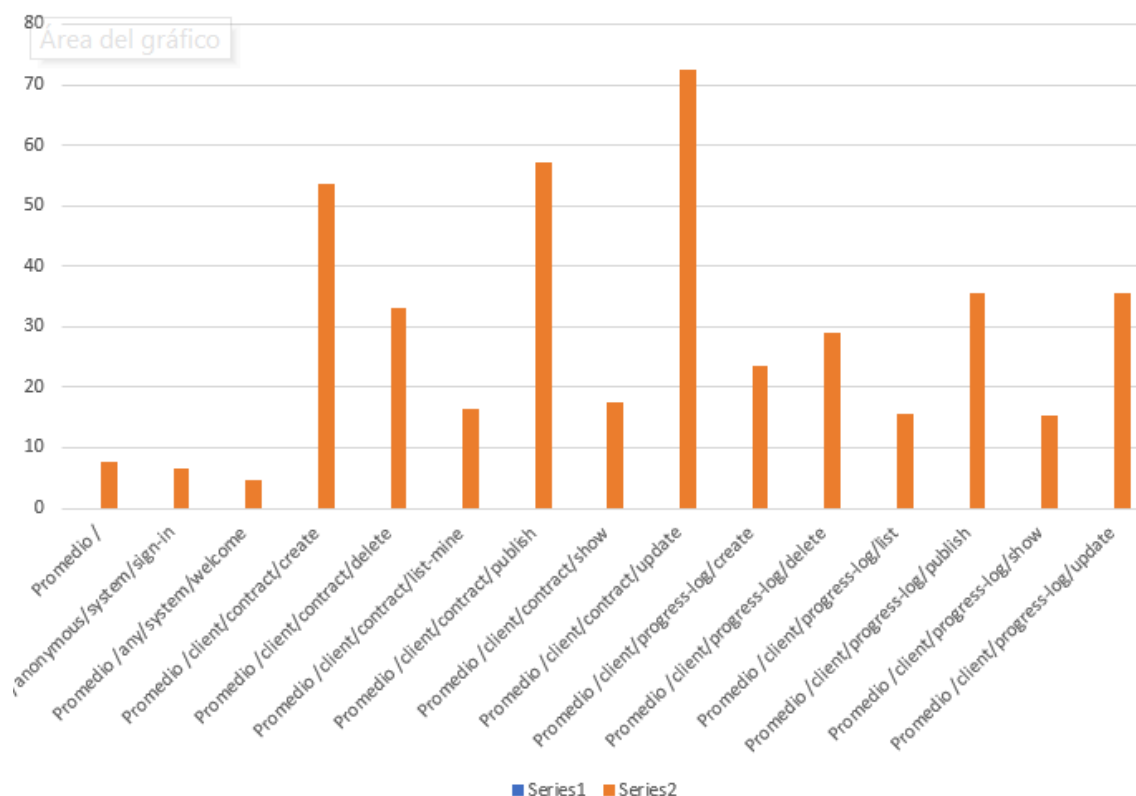
Por otra parte podemos ver que las funcionalidades que más tiempo requieren son el actualizado de un contrato (por todas las comprobaciones que hay que hacer y el hecho de tener que modificar una fila en base de datos). Los otros 2 más costosos son la creación y publicación del Contract, los cuáles tienen muchas validaciones por eso suponemos que tardan más. El resto de features tienen unos buenos tiempos.

También hemos analizado los tiempos en el equipo de otro miembro del grupo, en nuestro caso de Daniel Galván Cancio. Veamos los resultados:

Rendimiento Estudiante #5

request-method	request-path	response-status	time
	Promedio /		7,59848043
	Promedio /anonymous/system/sign-in		6,618356
	Promedio /any/system/welcome		4,52937174
	Promedio /client/contract/create		53,4779357
	Promedio /client/contract/delete		33,1484294
	Promedio /client/contract/list-mine		16,388215
	Promedio /client/contract/publish		57,0901345
	Promedio /client/contract/show		17,4717162
	Promedio /client/contract/update		72,336337
	Promedio /client/progress-log/create		23,4580167
	Promedio /client/progress-log/delete		28,983875
	Promedio /client/progress-log/list		15,4861528
	Promedio /client/progress-log/publish		35,5779063
	Promedio /client/progress-log/show		15,4248208
	Promedio /client/progress-log/update		35,492572
	Promedio general		24,7879175

Ordenador Estudiante #5



Estadísticas descriptivas

<i>Ordenador Estudiante #2</i>			<i>Ordenador Estudiante #5</i>		
Media	16,4507065		Media	24,7879175	
Error típico	0,77919057		Error típico	1,25638588	
Mediana	10,7862		Mediana	15,5776	
Moda	#N/D		Moda	#N/D	
Desviación estándar	16,9462754		Desviación estándar	27,3245876	
Varianza de la muestra	287,17625		Varianza de la muestra	746,633088	
Curtosis	19,6882228		Curtosis	20,0388737	
Coefficiente de asimetría	3,22853473		Coefficiente de asimetría	3,29463009	
Rango	174,015001		Rango	282,8229	
Mínimo	1,781		Mínimo	2,35	
Máximo	175,796001		Máximo	285,1729	
Suma	7781,18419		Suma	11724,685	
Cuenta	473		Cuenta	473	
Nivel de confianza(95,0%)	1,53111156		Nivel de confianza(95,0%)	2,46880161	
Interval(ms)	14,919595	17,9818181	Interval(ms)	22,3191159	27,2567192
Interval(s)	0,01491959	0,01798182	Interval(s)	0,02231912	0,02725672

Comparación de P-Value

Prueba z para medias de dos muestras		
	<i>Ordenador estudiante #2</i>	<i>Ordenador Estudiante #5</i>
Media	16,45070653	24,78791755
Varianza (conocida)	287,17625	746,633088
Observaciones	473	473
Diferencia hipotética de las medias	0	
z	-5,639377132	
P(Z<=z) una cola	8,53332E-09	
Valor crítico de z (una cola)	1,644853627	
Valor crítico de z (dos colas)	1,70666E-08	
Valor crítico de z (dos colas)	1,959963985	

Análisis

Observamos que los resultados si los medimos en segundos hay una diferencia notable, probablemente debida a la calidad de los equipos. Aún así en ambos se cumple el hecho de tener un valor por debajo del segundo.

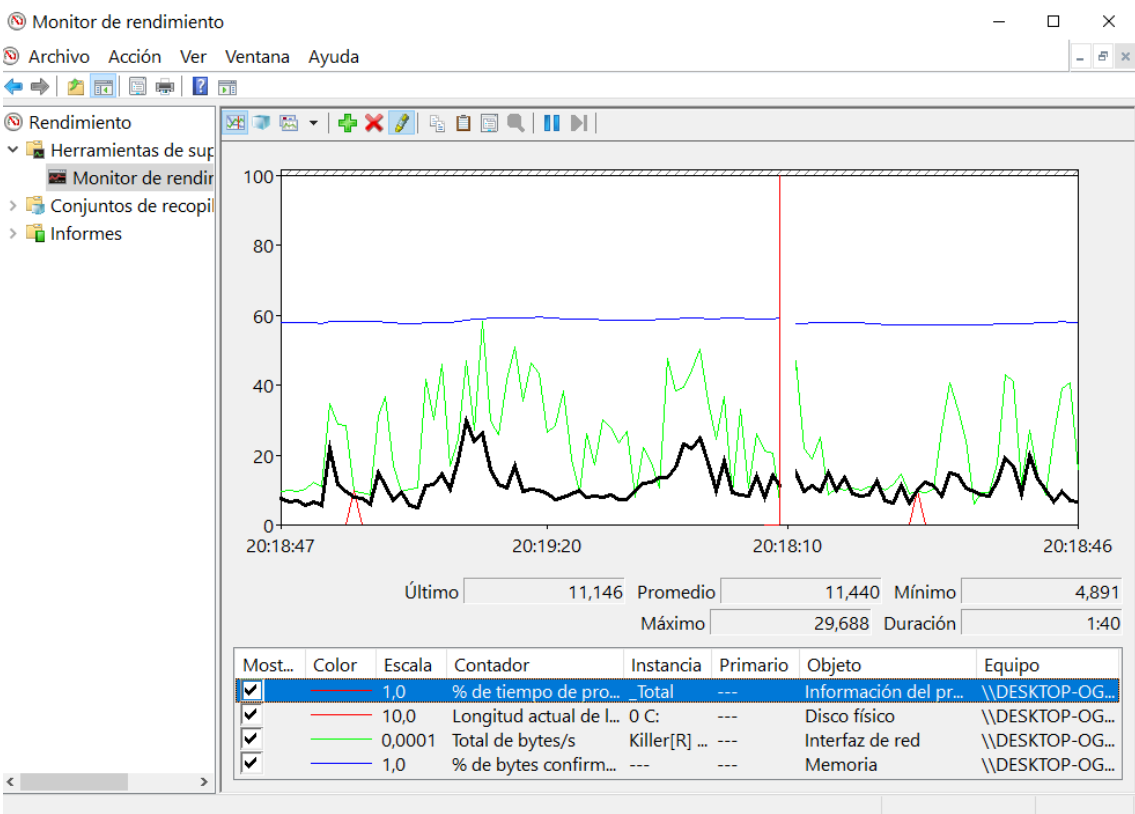
Para cerciorarnos, realizamos Z-Test. Se observa que el ordenador del estudiante #1 es mejor, pues el resultado de p-value es menor a 0.05 en el valor crítico de z (dos

colas).

Además, hemos usado visualVM para ver cuánto tardan en ejecutarse los métodos de nuestro proyecto.

Name	Self Time (CPU)	Total Time (CPU)
acme.features.client.progressLogs.ClientProgressLogUpdateService.validat	0,0 ms (- %)	103 ms (2,5 %)
acme.features.client.progressLogs.ClientProgressLogUpdateService.load ()	0,0 ms (- %)	99,7 ms (2,4 %)
acme.features.client.progressLogs.ClientProgressLogUpdateService.bind ()	0,0 ms (- %)	405 ms (9,6 %)
acme.features.client.progressLogs.ClientProgressLogShowService.authoris	0,0 ms (- %)	198 ms (4,7 %)
acme.features.client.progressLogs.ClientProgressLogPublishService.validat	0,0 ms (- %)	100 ms (2,4 %)
acme.features.client.progressLogs.ClientProgressLogPublishService.perform	0,0 ms (- %)	100 ms (2,4 %)
acme.features.client.progressLogs.ClientProgressLogPublishService.bind ()	0,0 ms (- %)	99,9 ms (2,4 %)
acme.features.client.progressLogs.ClientProgressLogPublishService.authori	0,0 ms (- %)	100 ms (2,4 %)
acme.features.client.progressLogs.ClientProgressLogListService.unbind ()	0,0 ms (- %)	0,0 ms (0 %)
acme.features.client.progressLogs.ClientProgressLogListService.load ()	0,0 ms (- %)	0,0 ms (0 %)
acme.features.client.progressLogs.ClientProgressLogDeleteService.load ()	0,0 ms (- %)	98,0 ms (2,3 %)
acme.features.client.progressLogs.ClientProgressLogDeleteService.bind ()	0,0 ms (- %)	98,2 ms (2,3 %)
acme.features.client.progressLogs.ClientProgressLogCreateService.validate	0,0 ms (- %)	98,3 ms (2,3 %)

Con respecto al estudio del hardware utilizando el monitor de rendimiento de windows, podemos determinar que no existen problemas ni cuellos de botella:



3. Bibliografía

No hay bibliografía presente para esta entrega.