

# Diseño y Pruebas II

## Testing Report

ACME SF-D04 - Luis García Parras



<b>Tabla de versiones</b>	<b>2</b>
<b>Tabla de revisiones</b>	<b>2</b>
<b>Resumen Ejecutivo</b>	<b>2</b>
<b>Introducción</b>	<b>3</b>
1. Contenido	3
<b>Conclusiones</b>	<b>5</b>
<b>Bibliography</b>	<b>6</b>

## Tabla de versiones

Versión	Fecha	Descripción
1.0	07/07/2024	Versión inicial del documento.

## Tabla de revisiones

N. Revisión	Fecha	Descripción
1	07/07/2024	D04

## Resumen Ejecutivo

El propósito de este informe es ofrecer una descripción detallada de los diferentes procedimientos seguidos que deben considerarse en el ámbito del testing formal del proyecto Acme-SF, desde la generación de suficientes datos de ejemplo, hasta la meticulosa comprobación de la correcta implementación de los requisitos funcionales y las distintas herramientas estadísticas para comparar los tiempos de ejecución entre peticiones, las diferencias del rendimiento entre dos ordenadores diferentes (antes de la mejora de índices), y por último la comparación

desde el ordenador de otro compañero del grupo para ver sus tiempos de ejecución.

En resumen, se ha utilizado un enfoque diligente para abordar y solucionar los errores encontrados durante el proceso de testeo con la finalidad de asegurar un producto de alto nivel con el que se satisfagan las expectativas del cliente y se han generado documentos que lo avalan.

## Introducción

El presente informe detalla los aspectos obtenidos de realizar los requisitos de testing 6 y 7.

En la primera sección, se describen los casos de prueba implementados, organizados por características, junto con evaluaciones sobre su eficacia para identificar errores. En la segunda sección, se incluyen gráficos relevantes y un intervalo de confianza del 95% para el tiempo de respuesta del proyecto ante las solicitudes funcionales, así como una comparación pertinente. Estos capítulos proporcionan una evaluación detallada de la calidad y eficiencia del proyecto en desarrollo, y finalmente, se presenta una conclusión del analista y la bibliografía utilizada.

## Contenido

Para cada característica se han realizado tanto pruebas .safe y pruebas .hack. Ahora vamos a ir desglosando según la entidad en cada uno de sus servicios como han ido yendo los resultados de la cobertura.

### ENTIDAD CONTRACT

#### *LIST:*

*Para el List hemos realizado pruebas safe que han consistido en ir listando los diferentes contratos y unas pruebas hack intentando acceder al listado de los contratos estando logueados con otro rol o sin estar logueados si quiera.*

*Este ha sido el resultado de la cobertura :*

>	ClientContractListAllService.java	94,6 %	70	4	74
---	-----------------------------------	--------	----	---	----

### SHOW:

*Para el show se han realizado pruebas safe que ha sido mostrar los detalles de cada uno de los contratos y unas hacking que consisten en acceder desde otros roles a los detalles del contrato de un cliente.*

*Este ha sido el resultado de la cobertura:*

>	 ClientContractShowService.java	 96,6 %	140	 5	145
---	------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------	-----	---------------------------------------------------------------------------------------	-----

### CREATE:

*Para el create se han realizado pruebas safe que ha sido ir probando los diferentes datos que ofrece el scrapbook a la hora de crear un contrato tanto casos positivos como negativos. En cuanto al hacking se ha intentado acceder a client/contract/create desde otros roles.*

*Este ha sido el resultado de la cobertura:*

>	 ClientContractCreateService.java	 94,0 %	300	 19	319
---	--------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------	-----	----------------------------------------------------------------------------------------	-----

#### UPDATE:

Para el update se han realizado pruebas safe que ha sido ir probando los diferentes datos que ofrece el scrapbook a la hora de modificar un contrato tanto casos positivos como negativos. En cuanto al hacking se ha intentado acceder a client/contract/update desde otros roles.

Este ha sido el resultado de la cobertura:

>	 ClientContractUpdateService.java	 93,6 %	291	20	311
---	--------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------	-----	----	-----

#### DELETE:

Para el delete se han realizado pruebas safe que ha sido ir eliminando diferentes contratos. En cuanto al hacking se ha intentado acceder a client/contract/delete desde otros roles.

Este ha sido el resultado de la cobertura:

>	 ClientContractDeleteService.java	 89,2 %	107	13	120
---	--------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------	-----	----	-----

#### PUBLISH:

Para el publish se han realizado pruebas safe que ha sido ir publicando diferentes contratos. En cuanto al hacking se ha intentado acceder a client/contract/publish desde otros roles.

Este ha sido el resultado de la cobertura:


>	 ClientContractPublishService.java	 92,9 %	262	20	282
---	-----------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------	-----	----	-----

### ENTIDAD PROGRESS LOG

#### LIST:

Para el List hemos realizado pruebas safe que han consistido en ir listando los diferentes progresos y unas pruebas hack intentando acceder al listado de los progresos estando logueados con otro rol o sin estar logueados si quiera.

Este ha sido el resultado de la cobertura :

>	 ClientProgressLogListAllService.java	 92,7 %	114	9	123
---	------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------	-----	---	-----

#### SHOW:

Para el show se han realizado pruebas safe que ha sido mostrar los detalles de cada uno de los progresos y unas hacking que consisten en acceder desde otros roles a los detalles del progresos de un cliente.

Este ha sido el resultado de la cobertura:

>	 ClientProgressLogShowService.java	 96,2 %	102	4	106
---	---------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------	-----	---	-----

#### CREATE:

Para el create se han realizado pruebas safe que ha sido ir probando los diferentes datos que ofrece el scrapbook a la hora de crear un progreso tanto casos positivos como negativos. En cuanto al hacking se ha intentado acceder a client/progress-log/create desde otros roles.

Este ha sido el resultado de la cobertura:

>	 ClientProgressLogCreateService.java	 92,1 %	234	20	254
---	-------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------	-----	----	-----

#### UPDATE:

Para el update se han realizado pruebas safe que ha sido ir probando los diferentes datos que ofrece el scrapbook a la hora de modificar un progreso tanto casos positivos como negativos. En cuanto al hacking se ha intentado acceder a client/progress-log/update desde otros roles.

Este ha sido el resultado de la cobertura:

>	 ClientProgressLogUpdateService.java	 91,2 %	248	24	272
---	-------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------	-----	----	-----

#### DELETE:

Para el delete se han realizado pruebas safe que ha sido ir eliminando diferentes progresos. En cuanto al hacking se ha intentado acceder a client/progress-log/delete desde otros roles.

Este ha sido el resultado de la cobertura:

>	 ClientProgressLogDeleteService.java	 88,9 %	168	21	189
---	-------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------	-----	----	-----

































## PUBLISH:

Para el publish se han realizado pruebas safe que ha sido ir publicando diferentes progresos. En cuanto al hacking se ha intentado acceder a `client/progress-log/publish` desde otros roles.

Este ha sido el resultado de la cobertura:

>	 ClientProgressLogPublishService.java	 90,2 %	157	17	174
---	------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------	-----	----	-----

## CONCLUSIÓN GENERAL:

▼	 acme.features.client.progresslog	 91,8 %	1.058	95	1.153
>	 ClientProgressLogUpdateService.java	 91,2 %	248	24	272
>	 ClientProgressLogDeleteService.java	 88,9 %	168	21	189
>	 ClientProgressLogCreateService.java	 92,1 %	234	20	254
>	 ClientProgressLogPublishService.java	 90,2 %	157	17	174
>	 ClientProgressLogListAllService.java	 92,7 %	114	9	123
>	 ClientProgressLogShowService.java	 96,2 %	102	4	106
>	 ClientProgressLogController.java	 100,0 %	35	0	35
▼	 acme.features.client.contract	 93,7 %	1.205	81	1.286
>	 ClientContractPublishService.java	 92,9 %	262	20	282
>	 ClientContractUpdateService.java	 93,6 %	291	20	311
>	 ClientContractCreateService.java	 94,0 %	300	19	319
>	 ClientContractDeleteService.java	 89,2 %	107	13	120
>	 ClientContractShowService.java	 96,6 %	140	5	145
>	 ClientContractListAllService.java	 94,6 %	70	4	74
>	 ClientContractController.java	 100,0 %	35	0	35

Podemos observar unos muy buenos resultados de cobertura lo que nos muestra que se han realizado unos buenos tests que han probado más del 90% de las instrucciones del proyecto.

Esto significa que se han verificado todas las validaciones y todos los posibles casos tanto positivos como negativos.

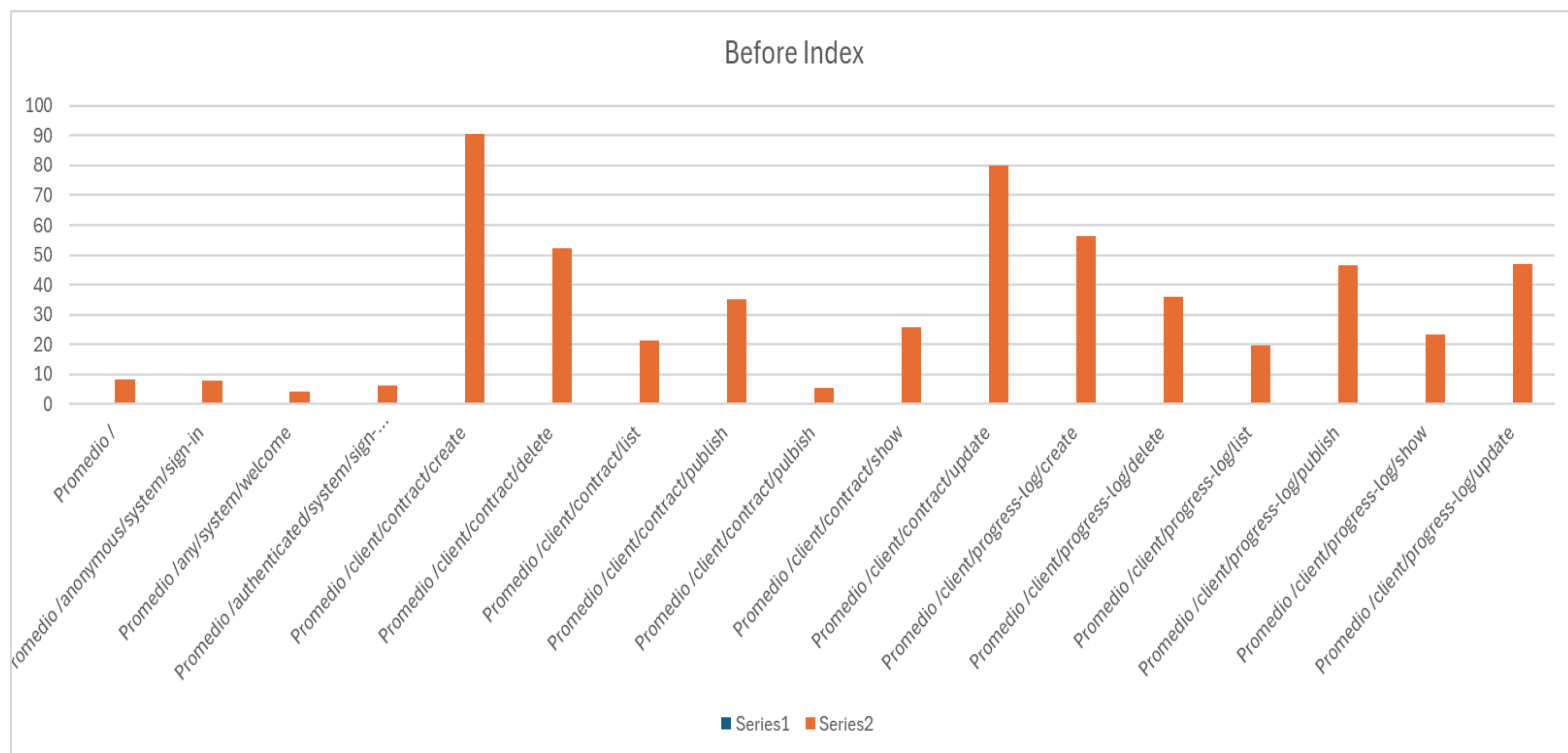
## Análisis de desempeño

Ahora vamos a comparar los resultados entre los tiempos de ejecución antes los índices y después de los índices.

Before				After		
Media	21,74661986			Media	20,4119961	
Error típico	1,508761709			Error típico	1,399984727	
Mediana	12,90785			Mediana	12,3623	
Moda	5,5492			Moda	3,0987	
Desviación estándar	29,64246433			Desviación estándar	27,50533572	
Varianza de la muestra	878,6756913			Varianza de la muestra	756,5434931	
Curtosis	31,12893227			Curtosis	32,84894314	
Coefficiente de asimetría	4,340689327			Coefficiente de asimetría	4,367819249	
Rango	316,1033			Rango	302,6414	
Mínimo	2,6877			Mínimo	2,5948	
Máximo	318,791			Máximo	305,2362	
Suma	8394,195265			Suma	7879,030494	
Cuenta	386			Cuenta	386	
Nivel de confianza(95,0%)	2,966444011			Nivel de confianza(95,0%)	2,752572712	
Interval(ms)	18,78017585	24,713064		Interval (ms)	17,65942339	23,164569
Interval(s)	0,018780176	0,0247131		Interval (s)	0,017659423	0,0231646

Vemos que con los índices se nota una pequeña disminución en los tiempos de ejecución pero no es nada bastante destacable.

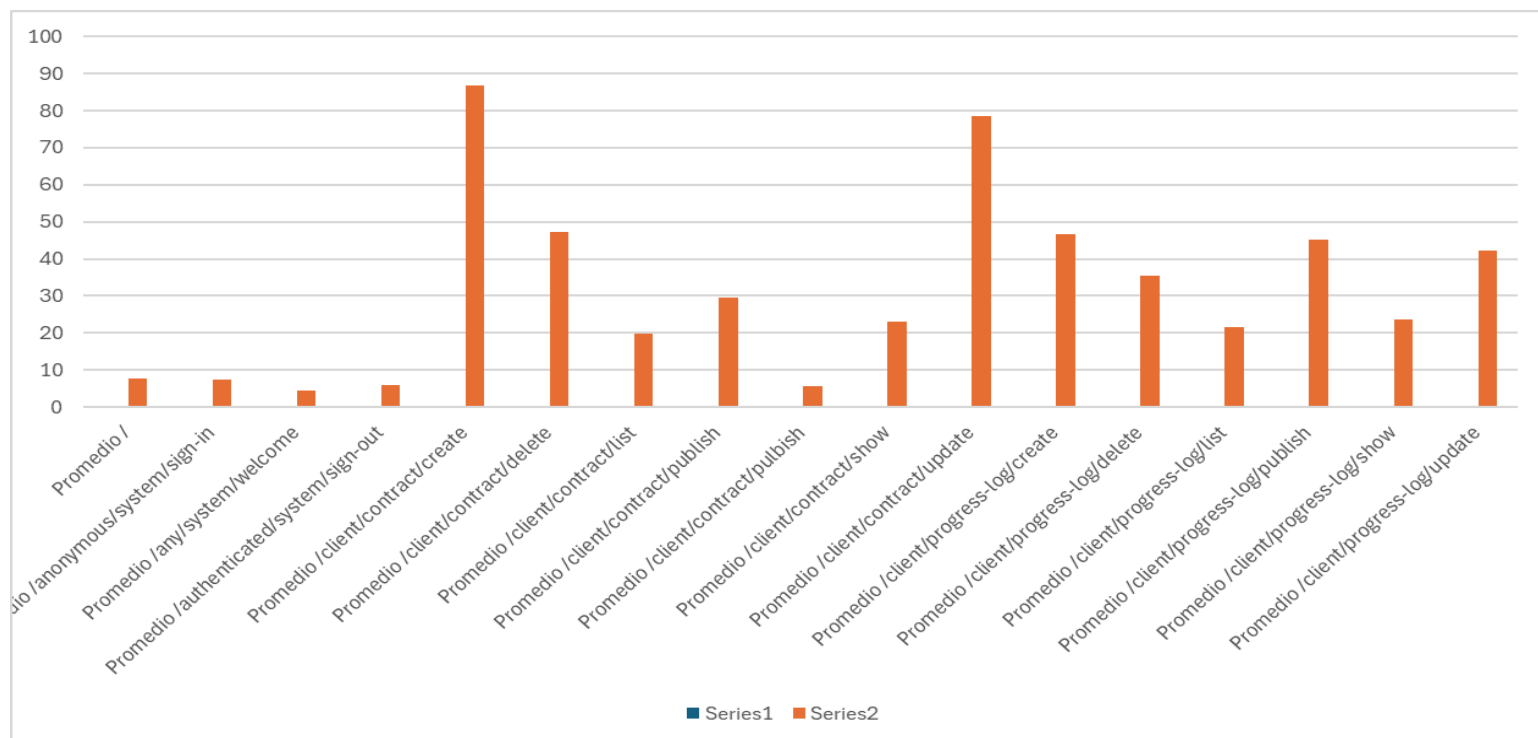
A continuación observamos las gráficas resultantes según las peticiones realizadas





Y después de los índices:

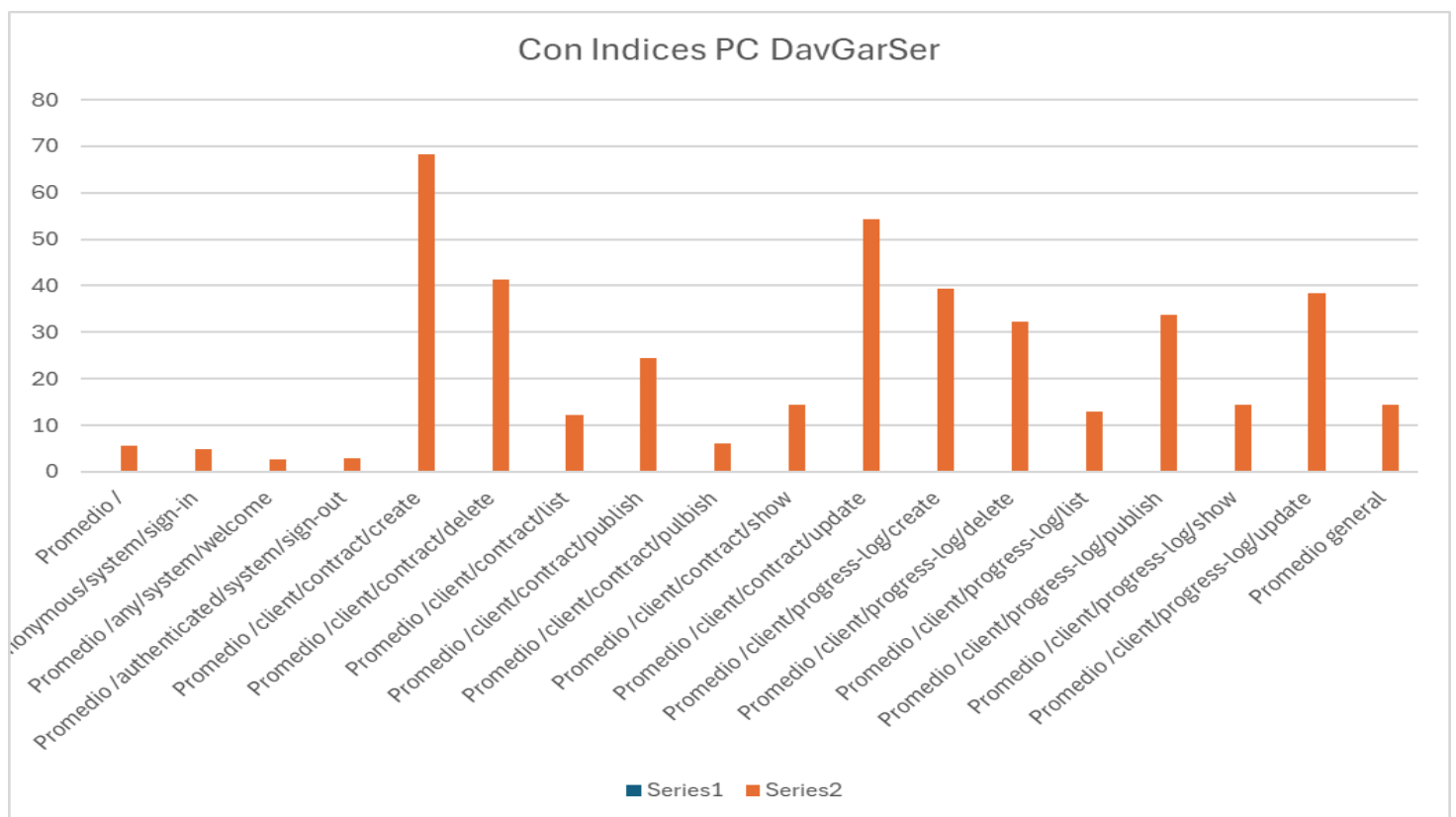
En cuanto a la comparación entre los resultados antes y después de usar los índices vemos que hay una pequeña mejoría en los servicios aunque realmente es algo bastante despreciable. Para ello vamos a hacer un análisis más exhaustivo mediante la prueba Z.



	202,7332	169,6255
Media	21,27652484	20,02442855
Varianza (conocida)	878,67	756,54
Observaciones	385	385
Diferencia hipotética de las medias	0	
z	0,607549022	
P(Z<=z) una cola	0,27174331	
Valor crítico de z (una cola)	1,644853627	
Valor crítico de z (dos colas)	0,54348662	
Valor crítico de z (dos colas)	1,959963985	

Sabiendo que tenemos un intervalo de confianza del 95 % podemos deducir que nuestro valor alpha es de 0.05 (1 - 0.95). Dicho esto podemos observar que nuestro valor Z es de 0.54 que al situarse en el intervalo (0.05,1) nos denota que los cambios son bastante irrelevantes y no han tenido repercusión en el contexto del análisis.

Ahora vemos los resultados de las pruebas en el PC de mi compañero de grupo David Gavira Serrano:



Podemos observar que hay una disminución destacable en cuanto a los tiempos medio de ejecución respecto a mi PC, podemos observar que el límite está en 70 es decir ninguna va

más allá del valor 70. Vamos a seguir analizando según los resultados con la prueba Z resultante.

Prueba z para medias de dos muestras		
	<i>Before (PC LUIS)</i>	<i>After ( PC Bonder)</i>
Media	20,4119961	14,67813489
Varianza (conocida)	409,54	756,54
Observaciones	386	385
Diferencia hipotética de las medias	0	
z	3,296180867	
P(Z<=z) una cola	0,000490044	
Valor crítico de z (una cola)	1,644853627	
Valor crítico de z (dos colas)	0,000980089	
Valor crítico de z (dos colas)	1,959963985	

Efectivamente se observa un valor Z entre el intervalo (0,0.05) lo que nos hace concluir que ha habido un cambio bastante destacable entre la ejecución en el ordenador de mi compañero y el mío.

He de aclarar que las pruebas y análisis se han realizado en las mismas condiciones ideales.

## Conclusiones

Gracias a este último entregable he podido cerciorarme que lo realizado desde el primer momento de esta asignatura es válido, es decir, he podido comprobar que todo lo realizado a lo largo de la asignatura es correcto, desde los form, hasta cada una de las características que he implementado para mis entidades. Ha sido un proceso meticuloso que también me ha hecho darme cuenta de errores que he ido cometiendo y posteriormente han sido testeados y analizados mediante varios métodos.

Puedo asegurar que el producto final es robusto y bastante fiable, aunque ha habido algunos problemas como el tema del coverage que no ha sido el mismo según donde se ha ejecutado.

También hemos tenido algunos problemas a la hora de mergear, pero todos como equipo hemos conseguido sacar el proyecto adelante.

Por último, he de decir, que este Deliverable es una parte fundamental del proyecto y no por ello menos importante. Protege la calidad del producto y la satisfacción del cliente.

## Bibliografía

Web de la universidad de Sevilla - <https://ev.us.es>

**ANEXO: A CONTINUACIÓN SE ADJUNTA CAPTURAS DE CADA RESULTADO DEL COVERAGE PARA CADA SERVICIO.**

```
1
2 package acme.features.client.contract;
3
4 import java.util.Collection;
5 import java.util.Date;
6
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.stereotype.Service;
9
10 import acme.client.data.models.Dataset;
11 import acme.client.helpers.MomentHelper;
12 import acme.client.services.AbstractService;
13 import acme.client.views.SelectChoices;
14 import acme.entities.contract.Contract;
15 import acme.entities.project.Project;
16 import acme.roles.Client;
17
18 @Service
19 public class ClientContractCreateService extends AbstractService<Client,
    Contract> {
20
21     @Autowired
22     private ClientContractRepository repository;
23
24
25     @Override
26     public void authorise() {
27         super.getResponse().setAuthorised(super.getRequest().getPrincipal
    ().hasRole(Client.class));
28     }
29
30     @Override
31     public void load() {
32         Contract object;
33         Client client;
34
35         client = this.repository.findClientByClientId(super.getRequest
    ().getPrincipal().getActiveRoleId());
36         object = new Contract();
37         object.setDraft(true);
38         object.setClient(client);
39
40         super.getBuffer().addData(object);
41     }
42
43     @Override
44     public void bind(final Contract contract) {
45         assert contract != null;
46         int projectId;
47         Project project;
```

```
48     projectId = super.getRequest().getData("project", int.class);
49     project = this.repository.findOneProjectById(projectId);
50     Date currentMoment = MomentHelper.getCurrentMoment();
51     Date moment = new Date(currentMoment.getTime());
52     super.bind(contract, "code", "moment", "providerName",
"customerName", "goals", "budget", "isDraft", "project");
53     contract.setProject(project);
54     contract.setMoment(moment);
55 }
56
57 @Override
58 public void validate(final Contract contract) {
59     assert contract != null;
60     Collection<Contract> contracts = null;
61     if (!super.getBuffer().getErrors().hasErrors("code")) {
62         contracts = this.repository.findAllContractsOfAClientById
(contract.getClient().getId());
63         boolean condition = contracts.contains(contract);
64         super.state(!condition, "code",
"client.contract.form.error.duplicated");
65     }
66     if (!super.getBuffer().getErrors().hasErrors("budget"))
67         super.state(contract.getBudget().getAmount() >= 0, "budget",
"client.contract.form.error.negative-budget");
68     if (!super.getBuffer().getErrors().hasErrors("budget")) {
69         contracts = this.repository.findAllContractsOfAClientById
(contract.getClient().getId());
70         double totalBudget = contracts.stream().mapToDouble(c ->
c.getBudget().getAmount()).sum();
71         double projectCost = contract.getProject().getCost().getAmount
();
72         totalBudget += contract.getBudget().getAmount();
73         super.state(totalBudget <= projectCost, "budget",
"client.contract.form.error.exceeds-project-cost");
74     }
75 }
76
77 @Override
78 public void perform(final Contract contract) {
79     assert contract != null;
80
81     this.repository.save(contract);
82 }
83
84 @Override
85 public void unbind(final Contract contract) {
86     assert contract != null;
87     boolean isDraft;
88     Collection<Project> projects;
```

```
90     projects = this.repository.findAllPublishedProjects();
91     SelectChoices choices;
92     choices = SelectChoices.from(projects, "title", contract.getProject
    ());
93     isDraft = contract.isDraft() == true;
94
95     Dataset dataset;
96
97     dataset = super.unbind(contract, "code", "moment", "providerName",
    "customerName", "goals", "budget", "isDraft", "project");
98     dataset.put("contractId", contract.getId());
99     dataset.put("isDraft", isDraft);
100    dataset.put("project", choices.getSelected().getKey());
101    dataset.put("projects", choices);
102
103    super.getResponse().addData(dataset);
104 }
105
106 }
107
```

```
1
2 package acme.features.client.contract;
3
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.stereotype.Service;
6
7 import acme.client.services.AbstractService;
8 import acme.entities.contract.Contract;
9 import acme.roles.Client;
10
11 @Service
12 public class ClientContractDeleteService extends AbstractService<Client,
    Contract> {
13
14     @Autowired
15     private ClientContractRepository repository;
16
17
18     @Override
19     public void authorise() {
20         boolean status;
21         int id = super.getRequest().getData("id", int.class);
22         Client client = this.repository.findClientByContractId(id);
23         status = super.getRequest().getPrincipal().getActiveRoleId() ==
    client.getId();
24         super.getResponse().setAuthorised(status);
25     }
26
27     @Override
28     public void load() {
29         Contract contract;
30         int id;
31
32         id = super.getRequest().getData("id", int.class);
33         contract = this.repository.findContractById(id);
34
35         super.getBuffer().addData(contract);
36     }
37
38     @Override
39     public void bind(final Contract contract) {
40         assert contract != null;
41
42         super.bind(contract, "code", "moment", "providerName",
    "customerName", "goals", "budget", "isDraft");
43     }
44
45     @Override
46     public void validate(final Contract contract) {
47         assert contract != null;
```



```
48     }
49
50     @Override
51     public void perform(final Contract contract) {
52         assert contract != null;
53         this.repository.deleteAll
54         (this.repository.findAllProgressLogsByContractId(contract.getId()));
55         this.repository.delete(contract);
56     }
57
58 }
59
```

```
1
2 package acme.features.client.contract;
3
4 import java.util.Collection;
5
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Service;
8
9 import acme.client.data.models.Dataset;
10 import acme.client.services.AbstractService;
11 import acme.entities.contract.Contract;
12 import acme.roles.Client;
13
14 @Service
15 public class ClientContractListAllService extends AbstractService<Client,
    Contract> {
16
17     // Internal state
18     -----
19     @Autowired
20     private ClientContractRepository repository;
21
22     // AbstractService interface
23     -----
24
25     @Override
26     public void authorise() {
27         super.getResponse().setAuthorised(super.getRequest().getPrincipal
    ().hasRole(Client.class));
28     }
29
30     @Override
31     public void load() {
32         int id = super.getRequest().getPrincipal().getActiveRoleId();
33         Collection<Contract> objects =
    this.repository.findAllContractsOfAClientById(id);
34         super.getBuffer().addData(objects);
35     }
36
37     @Override
38     public void unbind(final Contract object) {
39         assert object != null;
40
41         Dataset dataset;
42
43         dataset = super.unbind(object, "code", "moment", "providerName",
    "customerName", "budget", "goals", "project.title");
44
```

```
45     super.getResponse().addData(dataset);  
46 }  
47  
48 }  
49
```

```
1
2 package acme.features.client.contract;
3
4 import java.util.Collection;
5
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Service;
8
9 import acme.client.data.models.Dataset;
10 import acme.client.services.AbstractService;
11 import acme.entities.contract.Contract;
12 import acme.entities.project.Project;
13 import acme.roles.Client;
14
15 @Service
16 public class ClientContractPublishService extends AbstractService<Client,
    Contract> {
17
18     @Autowired
19     private ClientContractRepository repository;
20
21
22     @Override
23     public void authorise() {
24         boolean status;
25         int masterId;
26         Contract contract;
27         Client client;
28
29         masterId = super.getRequest().getData("id", int.class);
30         contract = this.repository.findContractById(masterId);
31         client = this.repository.findClientByContractId(masterId);
32         status = contract != null && contract.isDraft() && super.getRequest
    ().getPrincipal().hasRole(Client.class) && contract.getClient().equals
    (client);
33
34         super.getResponse().setAuthorised(status);
35
36     }
37
38     @Override
39     public void load() {
40         Contract contract;
41         int id;
42
43         id = super.getRequest().getData("id", int.class);
44         contract = this.repository.findContractById(id);
45
46         super.getBuffer().addData(contract);
47     }
```

```
48
49     @Override
50     public void bind(final Contract contract) {
51         assert contract != null;
52         int projectId;
53         Project project;
54
55         projectId = super.getRequest().getData("project", int.class);
56         project = this.repository.findOneProjectById(projectId);
57
58         super.bind(contract, "isDraft");
59         contract.setProject(project);
60     }
61
62     @Override
63     public void validate(final Contract contract) {
64         assert contract != null;
65         Collection<Contract> contracts = null;
66         if (!super.getBuffer().getErrors().hasErrors("code")) {
67             Contract existing;
68
69             existing = this.repository.findOneContractByCode
70                 (contract.getCode());
71
72             super.state(existing == null || existing.equals(contract),
73                 "recordId", "client.progresslog.form.error.duplicated");
74
75             if (!super.getBuffer().getErrors().hasErrors("budget"))
76                 super.state(contract.getBudget().getAmount() >= 0, "budget",
77                     "client.contract.form.error.negative-budget");
78             if (!super.getBuffer().getErrors().hasErrors("budget")) {
79                 contracts = this.repository.findAllContractsOfAClientById
80                     (contract.getClient().getId());
81                 double totalBudget = contracts.stream().mapToDouble(c ->
82                     c.getBudget().getAmount()).sum();
83                 double projectCost = contract.getProject().getCost().getAmount
84                     ();
85                 totalBudget += contract.getBudget().getAmount();
86                 super.state(totalBudget <= projectCost, "budget",
87                     "client.contract.form.error.exceeds-project-cost");
88             }
89         }
90     }
91
92     @Override
93     public void perform(final Contract contract) {
94         assert contract != null;
95
96         contract.setDraft(false);
97         this.repository.save(contract);
98     }
99 }
```

```
91     }
92     @Override
93     public void unbind(final Contract contract) {
94         assert contract != null;
95         boolean isDraft;
96
97         isDraft = contract.isDraft() == true;
98
99         Dataset dataset;
100
101         dataset = super.unbind(contract, "code", "moment", "providerName",
102             "customerName", "goals", "budget", "isDraft", "project");
103         dataset.put("isDraft", isDraft);
104
105         super.getResponse().addData(dataset);
106     }
107
108 }
109
```

```
1
2 package acme.features.client.contract;
3
4 import java.util.Collection;
5
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Service;
8
9 import acme.client.data.models.Dataset;
10 import acme.client.services.AbstractService;
11 import acme.client.views.SelectChoices;
12 import acme.entities.contract.Contract;
13 import acme.entities.project.Project;
14 import acme.roles.Client;
15
16 @Service
17 public class ClientContractShowService extends AbstractService<Client,
    Contract> {
18
19     @Autowired
20     private ClientContractRepository repository;
21
22
23     @Override
24     public void authorise() {
25         boolean status;
26         Contract contract;
27         int masterId;
28         Client client;
29
30         masterId = super.getRequest().getData("id", int.class);
31         contract = this.repository.findContractById(masterId);
32         client = this.repository.findClientByContractId(masterId);
33
34         status = super.getRequest().getPrincipal().hasRole(client) &&
            contract != null;
35
36         super.getResponse().setAuthorised(status);
37     }
38
39     @Override
40     public void load() {
41         int id;
42         id = super.getRequest().getData("id", int.class);
43         Contract contract = this.repository.findContractById(id);
44
45         super.getBuffer().addData(contract);
46     }
47
48     @Override
```

```
49     public void unbind(final Contract contract) {
50         assert contract != null;
51         boolean isDraft;
52         Collection<Project> projects;
53         projects = this.repository.findAllPublishedProjects();
54         SelectChoices choices;
55         choices = SelectChoices.from(projects, "title", contract.getProject
56         ());
57         isDraft = contract.isDraft() == true;
58         Dataset dataset;
59
60         dataset = super.unbind(contract, "code", "moment", "providerName",
61         "customerName", "goals", "budget", "isDraft", "project");
62         dataset.put("contractId", contract.getId());
63         dataset.put("isDraft", isDraft);
64         dataset.put("project", choices.getSelected().getKey());
65         dataset.put("projects", choices);
66         super.getResponse().addData(dataset);
67     }
68
69 }
70
```



```
1
2 package acme.features.client.contract;
3
4 import java.util.Collection;
5
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Service;
8
9 import acme.client.data.models.Dataset;
10 import acme.client.services.AbstractService;
11 import acme.client.views.SelectChoices;
12 import acme.entities.contract.Contract;
13 import acme.roles.Client;
14
15 @Service
16 public class ClientContractUpdateService extends AbstractService<Client,
    Contract> {
17
18     @Autowired
19     private ClientContractRepository repository;
20
21
22     @Override
23     public void authorise() {
24         boolean status;
25         int masterId;
26         Contract contract;
27         Client client;
28
29         masterId = super.getRequest().getData("id", int.class);
30         contract = this.repository.findContractById(masterId);
31         client = this.repository.findClientByContractId(masterId);
32         status = contract != null && contract.isDraft() && super.getRequest
    ().getPrincipal().hasRole(Client.class) && contract.getClient().equals
    (client);
33
34         super.getResponse().setAuthorised(status);
35     }
36
37     @Override
38     public void load() {
39         Contract contract;
40         int id;
41
42         id = super.getRequest().getData("id", int.class);
43         contract = this.repository.findContractById(id);
44
45         super.getBuffer().addData(contract);
46     }
47
```

```
48     @Override
49     public void bind(final Contract contract) {
50         assert contract != null;
51
52         super.bind(contract, "code", "moment", "providerName",
53             "customerName", "goals", "budget", "isDraft", "project");
54
55     @Override
56     public void validate(final Contract contract) {
57         assert contract != null;
58         Collection<Contract> contracts = null;
59         if (!super.getBuffer().getErrors().hasErrors("code")) {
60             Contract existing;
61
62             existing = this.repository.findOneContractByCode
63                 (contract.getCode());
64
65             super.state(existing == null || existing.equals(contract),
66                 "recordId", "client.progresslog.form.error.duplicated");
67
68             if (!super.getBuffer().getErrors().hasErrors("budget"))
69                 super.state(contract.getBudget().getAmount() >= 0, "budget",
70                     "client.contract.form.error.negative-budget");
71             if (!super.getBuffer().getErrors().hasErrors("budget")) {
72                 contracts = this.repository.findAllContractsOfAClientById
73                     (contract.getClient().getId());
74                 double totalBudget = contracts.stream().mapToDouble(c ->
75                     c.getBudget().getAmount()).sum();
76                 double projectCost = contract.getProject().getCost().getAmount
77                     ();
78                 totalBudget += contract.getBudget().getAmount();
79                 super.state(totalBudget <= projectCost, "budget",
80                     "client.contract.form.error.exceeds-project-cost");
81             }
82         }
83     }
84
85     @Override
86     public void perform(final Contract contract) {
87         assert contract != null;
88
89         this.repository.save(contract);
90     }
91
92     @Override
93     public void unbind(final Contract contract) {
94         assert contract != null;
95         boolean isDraft;
96         SelectChoices choices;
```

```
90     choices =
SelectChoices.from(this.repository.findAllPublishedProjects(), "title",
contract.getProject());
91     isDraft = contract.isDraft() == true;
92
93     Dataset dataset;
94
95     dataset = super.unbind(contract, "code", "moment", "providerName",
"customerName", "goals", "budget", "isDraft", "project");
96     dataset.put("contractId", contract.getId());
97     dataset.put("isDraft", isDraft);
98     dataset.put("projects", choices);
99
100     super.getResponse().addData(dataset);
101 }
102
103 }
104
```

```
1
2 package acme.features.client.contract;
3
4 import javax.annotation.PostConstruct;
5
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Controller;
8
9 import acme.client.controllers.AbstractController;
10 import acme.entities.contract.Contract;
11 import acme.roles.Client;
12
13 @Controller
14 public class ClientContractController extends AbstractController<Client,
    Contract> {
15
16     // hola
17     @Autowired
18     private ClientContractDeleteService deleteService;
19
20     @Autowired
21     private ClientContractCreateService createService;
22
23     @Autowired
24     private ClientContractUpdateService updateService;
25
26     @Autowired
27     private ClientContractShowService showService;
28
29     @Autowired
30     private ClientContractListAllService listService;
31
32     @Autowired
33     private ClientContractPublishService publishService;
34
35
36     @PostConstruct
37     protected void initialise() {
38
39         super.addBasicCommand("create", this.createService);
40         super.addBasicCommand("update", this.updateService);
41         super.addBasicCommand("show", this.showService);
42         super.addBasicCommand("list", this.listService);
43         super.addBasicCommand("delete", this.deleteService);
44         super.addCustomCommand("publish", "update", this.publishService);
45     }
46 }
47
```

```
1
2 package acme.features.client.progresslog;
3
4 import javax.annotation.PostConstruct;
5
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Controller;
8
9 import acme.client.controllers.AbstractController;
10 import acme.entities.contract.ProgressLog;
11 import acme.roles.Client;
12
13 @Controller
14 public class ClientProgressLogController extends AbstractController<Client,
    ProgressLog> {
15
16     @Autowired
17     private ClientProgressLogDeleteService deleteService;
18
19     @Autowired
20     private ClientProgressLogCreateService createService;
21
22     @Autowired
23     private ClientProgressLogUpdateService updateService;
24
25     @Autowired
26     private ClientProgressLogShowService showService;
27
28     @Autowired
29     private ClientProgressLogListAllService listService;
30
31     @Autowired
32     private ClientProgressLogPublishService publishService;
33
34
35     @PostConstruct
36     protected void initialise() {
37
38         super.addBasicCommand("create", this.createService);
39         super.addBasicCommand("update", this.updateService);
40         super.addBasicCommand("show", this.showService);
41         super.addBasicCommand("list", this.listService);
42         super.addBasicCommand("delete", this.deleteService);
43         super.addCustomCommand("publish", "update", this.publishService);
44     }
45 }
46
```

```
1
2 package acme.features.client.progresslog;
3
4 import java.util.Date;
5
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Service;
8
9 import acme.client.data.models.Dataset;
10 import acme.client.helpers.MomentHelper;
11 import acme.client.services.AbstractService;
12 import acme.entities.contract.Contract;
13 import acme.entities.contract.ProgressLog;
14 import acme.roles.Client;
15
16 @Service
17 public class ClientProgressLogCreateService extends AbstractService<Client,
    ProgressLog> {
18
19     @Autowired
20     private ClientProgressLogRepository repository;
21
22
23     @Override
24     public void authorise() {
25         super.getResponse().setAuthorised(super.getRequest().getPrincipal
    ().hasRole(Client.class));
26     }
27
28     @Override
29     public void load() {
30         final int masterId = super.getRequest().getData("masterId",
    int.class);
31         Contract contract;
32
33         contract = this.repository.findOneContractById(masterId);
34
35         ProgressLog progressLog = new ProgressLog();
36         progressLog.setDraft(true);
37         progressLog.setContract(contract);
38         super.getBuffer().addData(progressLog);
39     }
40
41     @Override
42     public void bind(final ProgressLog object) {
43         assert object != null;
44         Date currentMoment = MomentHelper.getCurrentMoment();
45         Date registrationMoment = new Date(currentMoment.getTime());
46         super.bind(object, "recordId", "completeness", "comment",
    "registrationMoment", "reponsiblePerson", "isDraft");
```

```
47     int masterId = super.getRequest().getData("masterId", int.class);
48     Contract contract = this.repository.findOneContractById(masterId);
49     object.setContract(contract);
50     object.setRegistrationMoment(registrationMoment);
51 }
52
53 @Override
54 public void validate(final ProgressLog object) {
55     assert object != null;
56
57     if (!super.getBuffer().getErrors().hasErrors("recordId")) {
58         ProgressLog existing;
59         existing = this.repository.findOneProgressLogByCode
60 (object.getRecordId());
61         super.state(existing == null, "recordId",
62 "client.progresslog.form.error.duplicated");
63     }
64     if (!super.getBuffer().getErrors().hasErrors("completeness")) {
65         Double currentCompleteness =
66 this.repository.findTotalCompletenessByContractIdExceptSelf
67 (object.getContract().getId(), object.getId());
68         if (currentCompleteness == null)
69             currentCompleteness = 0.0;
70         Double objectCompleteness = object.getCompleteness();
71         double totalCompleteness = currentCompleteness +
72 objectCompleteness;
73         super.state(totalCompleteness <= 100, "completeness",
74 "client.progresslog.form.error.completeness");
75     }
76 }
77
78 @Override
79 public void perform(final ProgressLog object) {
80     assert object != null;
81
82     this.repository.save(object);
83 }
84
85 @Override
86 public void unbind(final ProgressLog object) {
87     assert object != null;
88     Dataset dataset;
89
90     //SelectChoices contractChoices;
91     //Collection<Contract> contracts =
92 this.repository.findAllContractsByClientId(super.getRequest().getPrincipal
93 ().getActiveRoleId());
94
95     //contractChoices = SelectChoices.from(contracts, "code",
96 object.getContract());
```

```
88
89     dataset = super.unbind(object, "contract", "recordId",
    "completeness", "comment", "registrationMoment", "isDraft",
    "reponsiblePerson");
90     dataset.put("masterId", super.getRequest().getData("masterId",
    int.class));
91     dataset.put("isDraft", object.getContract().isDraft());
92     super.getResponse().addData(dataset);
93 }
94 }
95
```



```
1
2 package acme.features.client.progresslog;
3
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.stereotype.Service;
6
7 import acme.client.data.models.Dataset;
8 import acme.client.services.AbstractService;
9 import acme.entities.contract.Contract;
10 import acme.entities.contract.ProgressLog;
11 import acme.roles.Client;
12
13 @Service
14 public class ClientProgressLogDeleteService extends AbstractService<Client,
    ProgressLog> {
15
16     @Autowired
17     private ClientProgressLogRepository repository;
18
19
20     @Override
21     public void authorise() {
22         boolean status;
23         ProgressLog progressLog;
24         Contract contract;
25         Client client;
26         int id = super.getRequest().getData("id", int.class);
27         progressLog = this.repository.findOneProgressLogById(id);
28         contract = progressLog == null ? null : progressLog.getContract();
29         client = contract == null ? null : contract.getClient();
30         status = progressLog != null && progressLog.isDraft() && contract !=
    null && contract.isDraft() && super.getRequest().getPrincipal().hasRole
    (client) && contract.getClient().getId() == super.getRequest().getPrincipal
    ().getActiveRoleId();
31         super.getResponse().setAuthorised(status);
32     }
33
34     @Override
35     public void load() {
36         ProgressLog object;
37         int id;
38
39         id = super.getRequest().getData("id", int.class);
40         object = this.repository.findOneProgressLogById(id);
41         super.getBuffer().addData(object);
42     }
43
44     @Override
45     public void bind(final ProgressLog object) {
46         assert object != null;
```

```
47
48     super.bind(object, "recordId", "contract", "completeness",
"comment", "registrationMoment", "reponsiblePerson", "isDraft");
49 }
50 @Override
51 public void validate(final ProgressLog object) {
52     assert object != null;
53 }
54
55 @Override
56 public void perform(final ProgressLog object) {
57     assert object != null;
58
59     this.repository.delete(object);
60 }
61
62 @Override
63 public void unbind(final ProgressLog object) {
64     assert object != null;
65     Dataset dataset;
66
67     dataset = super.unbind(object, "recordId", "contract",
"completeness", "comment", "registrationMoment", "reponsiblePerson",
"isDraft");
68
69     super.getResponse().addData(dataset);
70 }
71 }
72
```

```
1
2 package acme.features.client.progresslog;
3
4 import java.util.Collection;
5
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Service;
8
9 import acme.client.data.models.Dataset;
10 import acme.client.services.AbstractService;
11 import acme.entities.contract.Contract;
12 import acme.entities.contract.ProgressLog;
13 import acme.roles.Client;
14
15 @Service
16 public class ClientProgressLogListAllService extends AbstractService<Client,
    ProgressLog> {
17
18     // Internal state
19     -----
20     @Autowired
21     private ClientProgressLogRepository repository;
22
23     // AbstractService interface
24     -----
25
26     @Override
27     public void authorise() {
28         super.getResponse().setAuthorised(super.getRequest().getPrincipal
    ().hasRole(Client.class));
29     }
30
31     @Override
32     public void load() {
33         Collection<ProgressLog> objects;
34         int masterId;
35         masterId = super.getRequest().getData("masterId", int.class);
36         objects = this.repository.findAllProgressLogsByContractId(masterId);
37         super.getBuffer().addData(objects);
38
39     }
40
41     @Override
42     public void unbind(final ProgressLog object) {
43         assert object != null;
44
45         Dataset dataset;
46
```

```
47     dataset = super.unbind(object, "recordId", "completeness",  
    "registrationMoment", "reponsiblePerson", "isDraft");  
48  
49     super.getResponse().addData(dataset);  
50 }  
51  
52 @Override  
53 public void unbind(final Collection<ProgressLog> objects) {  
54     assert objects != null;  
55  
56     int masterId;  
57     Contract contract;  
58     final boolean showCreate;  
59  
60     masterId = super.getRequest().getData("masterId", int.class);  
61     contract = this.repository.findOneContractById(masterId);  
62     showCreate = contract.isDraft() && super.getRequest().getPrincipal  
    ().hasRole(contract.getClient());  
63  
64     super.getResponse().addGlobal("masterId", masterId);  
65     super.getResponse().addGlobal("showCreate", showCreate);  
66     super.getResponse().addGlobal("isDraft", contract.isDraft());  
67 }  
68 }  
69
```

```
1
2 package acme.features.client.progresslog;
3
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.stereotype.Service;
6
7 import acme.client.data.models.Dataset;
8 import acme.client.services.AbstractService;
9 import acme.entities.contract.ProgressLog;
10 import acme.roles.Client;
11
12 @Service
13 public class ClientProgressLogPublishService extends AbstractService<Client,
    ProgressLog> {
14
15     @Autowired
16     private ClientProgressLogRepository repository;
17
18
19     @Override
20     public void authorise() {
21         boolean status;
22         int masterId;
23         ProgressLog progressLog;
24         masterId = super.getRequest().getData("id", int.class);
25         progressLog = this.repository.findOneProgressLogById(masterId);
26         status = progressLog != null && progressLog.isDraft() &&
            super.getRequest().getPrincipal().hasRole(Client.class) &&
            progressLog.getContract().getClient().getId() == super.getRequest
                ().getPrincipal().getActiveRoleId();
27
28         super.getResponse().setAuthorised(status);
29     }
30
31     @Override
32     public void load() {
33         ProgressLog object;
34         int id;
35
36         id = super.getRequest().getData("id", int.class);
37         object = this.repository.findOneProgressLogById(id);
38         super.getBuffer().addData(object);
39     }
40
41     @Override
42     public void bind(final ProgressLog object) {
43         assert object != null;
44
45         super.bind(object, "recordId", "completeness", "comment",
            "registrationMoment", "reponsiblePerson", "isDraft");
```

```
46 }
47
48 @Override
49 public void validate(final ProgressLog object) {
50     assert object != null;
51 }
52
53
54 @Override
55 public void perform(final ProgressLog progresslog) {
56     assert progresslog != null;
57
58     progresslog.setDraft(false);
59
60     this.repository.save(progresslog);
61 }
62
63 @Override
64 public void unbind(final ProgressLog object) {
65     assert object != null;
66     Dataset dataset;
67
68     dataset = super.unbind(object, "recordId", "completeness",
69 "comment", "registrationMoment", "reponsiblePerson", "isDraft");
70
71     dataset.put("isDraft", object.isDraft());
72
73     super.getResponse().addData(dataset);
74 }
75 }
76
```

```
1
2 package acme.features.client.progresslog;
3
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.stereotype.Service;
6
7 import acme.client.data.models.Dataset;
8 import acme.client.services.AbstractService;
9 import acme.entities.contract.Contract;
10 import acme.entities.contract.ProgressLog;
11 import acme.roles.Client;
12
13 @Service
14 public class ClientProgressLogShowService extends AbstractService<Client,
    ProgressLog> {
15
16     // Internal state
17     -----
18     @Autowired
19     private ClientProgressLogRepository repository;
20
21     // AbstractService interface
22     -----
23
24     @Override
25     public void authorise() {
26         boolean status;
27         int id;
28         Contract contract;
29
30         id = super.getRequest().getData("id", int.class);
31         contract = this.repository.findOneContractByProgressLogId(id);
32         status = contract != null && (!contract.isDraft() ||
    super.getRequest().getPrincipal().hasRole(contract.getClient()));
33
34         super.getResponse().setAuthorised(status);
35     }
36
37     @Override
38     public void load() {
39         ProgressLog object;
40         int id;
41
42         id = super.getRequest().getData("id", int.class);
43         object = this.repository.findOneProgressLogById(id);
44
45         super.getBuffer().addData(object);
46     }
```

```
47
48     @Override
49     public void unbind(final ProgressLog object) {
50         assert object != null;
51         Dataset dataset;
52
53         dataset = super.unbind(object, "recordId", "completeness",
54             "comment", "registrationMoment", "reponsiblePerson", "isDraft");
55         dataset.put("masterId", object.getContract().getId());
56
57         super.getResponse().addData(dataset);
58     }
59
60 }
61
```



```
1
2 package acme.features.client.progresslog;
3
4 import java.util.Collection;
5
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Service;
8
9 import acme.client.data.models.Dataset;
10 import acme.client.services.AbstractService;
11 import acme.client.views.SelectChoices;
12 import acme.entities.contract.Contract;
13 import acme.entities.contract.ProgressLog;
14 import acme.roles.Client;
15
16 @Service
17 public class ClientProgressLogUpdateService extends AbstractService<Client,
    ProgressLog> {
18
19     @Autowired
20     private ClientProgressLogRepository repository;
21
22
23     @Override
24     public void authorise() {
25         boolean status;
26         int progressLogId;
27         Contract contract;
28         ProgressLog progressLog;
29         progressLogId = super.getRequest().getData("id", int.class);
30         contract = this.repository.findOneContractByProgressLogId
    (progressLogId);
31         progressLog = this.repository.findOneProgressLogById
    (progressLogId);
32         status = progressLog != null && progressLog.isDraft() && contract !=
    null && super.getRequest().getPrincipal().hasRole(progressLog.getContract
    ().getClient());
33
34         super.getResponse().setAuthorised(status);
35     }
36
37     @Override
38     public void load() {
39         ProgressLog object;
40         int id;
41
42         id = super.getRequest().getData("id", int.class);
43         object = this.repository.findOneProgressLogById(id);
44         super.getBuffer().addData(object);
45     }
```

```
46
47     @Override
48     public void bind(final ProgressLog object) {
49         assert object != null;
50
51         super.bind(object, "recordId", "completeness", "comment",
52 "registrationMoment", "isDraft", "responsiblePerson");
53
54     @Override
55     public void validate(final ProgressLog object) {
56         assert object != null;
57
58         if (!super.getBuffer().getErrors().hasErrors("recordId")) {
59             ProgressLog existing;
60
61             existing = this.repository.findOneProgressLogByCode
62 (object.getRecordId());
63
64             super.state(existing == null || existing.equals(object),
65 "recordId", "client.progresslog.form.error.duplicated");
66
67             if (!super.getBuffer().getErrors().hasErrors("completeness")) {
68                 Double currentCompleteness =
69 this.repository.findTotalCompletenessByContractIdExceptSelf
70 (object.getContract().getId(), object.getId());
71
72                 if (currentCompleteness == null)
73                     currentCompleteness = 0.0;
74                 Double objectCompleteness = object.getCompleteness();
75                 double totalCompleteness = currentCompleteness +
76 objectCompleteness;
77                 super.state(totalCompleteness <= 100, "completeness",
78 "client.progresslog.form.error.completeness");
79             }
80         }
81
82     @Override
83     public void perform(final ProgressLog progresslog) {
84         assert progresslog != null;
85
86         this.repository.save(progresslog);
87     }
88
89     @Override
90     public void unbind(final ProgressLog object) {
91         assert object != null;
92         Dataset dataset;
93         boolean isDraft;
94         SelectChoices choices;
95         Collection<Contract> contracts =
```

```
    this.repository.findAllContractsByClientId(super.getRequest().getPrincipal  
        ().getActiveRoleId());  
89  
90    choices = SelectChoices.from(contracts, "code", object.getContract  
    ());  
91  
92    isDraft = object.isDraft() == true;  
93  
94    dataset = super.unbind(object, "recordId", "contract",  
    "completeness", "comment", "registrationMoment", "isDraft",  
    "reponsiblePerson");  
95    dataset.put("contract", choices.getSelected().getKey());  
96    dataset.put("contracts", choices);  
97    dataset.put("isDraft", isDraft);  
98    super.getResponse().addData(dataset);  
99    }  
100 }  
101
```