



UNIVERSIDADE
DE VIGO

ESCOLA SUPERIOR DE ENXEÑARÍA INFORMÁTICA

Memoria do Traballo de Fin de Grao que presenta

D. Abel Fernández Nandín

para a obtención do Título de Graduado en Enxeñaría Informática

Asistente de Laboratorio para MALDI



Xuño, 2014

Traballo de Fin de Grao N°: EI 13/14 - 024

Titor/a: Miguel Reboiro Jato

Área de coñecemento: Linguaxes e Sistemas Informáticos

Departamento: Informática

Tabla de contenido

1	Introdución	9
1.1	Identificación do traballo	9
1.2	Organización da documentación	9
1.3	Marco da aplicación.....	9
1.4	Obxectivos da aplicación	10
1.5	Descrición técnica da aplicación.....	10
2	Metodoloxías.....	11
2.1	Metodoloxía Scrum.....	12
2.2	Historias de usuario.....	12
3	Tecnoloxías empregadas.....	13
3.1	ZK Framework	13
3.2	Spring Framework	13
3.3	Hibernate ORM.....	13
3.4	JUnit	13
3.5	DbUnit	13
3.6	Apache Maven.....	14
3.7	Kunagi.....	14
4	Planificación e Presuposto.....	14
4.1	Planificación.....	14
4.1.1	Planificación estimada.....	15
4.1.2	Seguimento temporal.....	18
4.2	Presuposto.....	21
4.2.1	Custo do hardware	21
4.2.2	Custo do software.....	21
4.2.3	Custo do persoal	21
4.2.4	Custo de impresión e encadernación	22
4.2.5	Custo total.....	22
5	Problemas atopados e solucións aportadas.....	22
6	Futuras ampliacións.....	23
6.1	Depositar mostras.....	23
6.2	Crear experimento avanzado	23
6.3	Subir resultados avanzado	24
6.4	Ordenar resultados avanzado	24

6.5	Deseñar padróns aleatoriamente.....	24
6.6	Modificar experimento.....	25
6.7	Datos e configuración persoais.....	25
6.8	Deseñar padróns manualmente avanzado.....	25
7	Conclusións.....	25
8	Bibliografía.....	26
9	Análise.....	29
9.1	Roles de usuario.....	29
9.2	Historias de usuario.....	29
9.2.1	HU01 - Conta persoal.....	29
9.2.2	HU02 - Listar experimentos.....	29
9.2.3	HU03 - Crear experimento básico.....	30
9.2.4	HU04 - Implementar probas de DAOs.....	30
9.2.5	HU05 - Implementar probas de Entidades e ViewModels.....	30
9.2.6	HU06 - Deseñar padróns manualmente básico.....	31
9.2.7	HU07 - Subir resultados básico.....	31
9.2.8	HU08 - Ordenar resultados básico.....	31
9.2.9	HU09 - Eliminar experimento.....	31
9.2.10	HU10 - Memoria TFG.....	32
9.2.11	HU11 - Manual técnico TFG.....	32
9.2.12	HU12 – Manual de usuario TFG.....	32
9.3	Planificación e seguimento das historias de usuario.....	32
10	Deseño.....	33
10.1	Arquitectura xeral de ZK.....	33
10.2	Arquitectura do sistema desenvolvido.....	34
10.2.1	Vista estática.....	36
10.2.1.1	Diagramas de clases.....	36
10.2.1.2	Descrición das clases.....	37
10.2.2	Vista dinámica.....	49
10.2.2.1	HU01 – Conta persoal.....	50
10.2.2.2	HU02 – Listar experimentos.....	52
10.2.2.3	Hu03 – Crear Experimento Básico.....	53
10.2.2.4	HU06 – Deseñar padróns manualmente básico.....	56
10.2.2.5	HU07 – Subir resultados básico.....	56
10.2.2.6	HU08 - Ordenar resultados básico.....	57

10.2.2.7	HU09 - Eliminar experimento	58
10.2.3	Probas	59
10.2.3.1	Probas de aceptación.....	59
10.2.3.2	Probas de unidade	59
11	Guía de instalación do software necesario.....	65
11.1	Instalar Java.....	65
11.2	Instalar e configurar Tomcat	65
11.3	Instalar e configurar MySQL.....	65
12	Guía de instalación do Asistente de Laboratorio para MALDI	66
13	Guía de uso do Asistente de Laboratorio para MALDI.....	67
13.1	Proceso de rexistro.....	67
13.2	Proceso de login	68
13.3	Páxina principal	68
13.4	Vista de edición do experimento	70
13.5	Vista da edición das placas do experimento	70
13.6	Vista de ordenado de arquivos.....	71

Índice de Figuras

Figura 1. Arquitectura xeral dunha aplicación ZK.....	11
Figura 2. Diagrama de Gantt da planificación estimada.....	17
Figura 3. Diagrama de Gantt do seguimento temporal.....	20
Figura 4. Arquitectura do patrón de deseño Model-View-ViewModel.	34
Figura 5. Arquitectura do Asistente de Laboratorio para MALDI.....	34
Figura 6. Diagrama xenérico coas chamadas entre clases máis comúns na aplicación.	35
Figura 7. Diagrama de clases das capas Entities - DAO - Service.....	36
Figura 8. Diagrama de clases das capas Service - ViewModel.....	37
Figura 9. Diagrama de clases do paquete IO para a ordenación de arquivos e creación de directorios.....	37
Figura 10. Diagrama de secuencia para a historia de usuario "Conta persoal".	50
Figura 11. Ref. Hu01 - Engadir usuario	51
Figura 12. Diagrama de secuencia para a historia de usuario "Listar experimentos"....	52
Figura 13. Diagrama de secuencia da historia de usuario "Crear experimento básico".	53
Figura 14. Ref. Hu03 - Engadir condición.....	54
Figura 15. Ref. Hu03 - Engadir mostra.	55
Figura 16. Ref. Hu03 - Engadir réplica.....	55
Figura 17. Diagrama de secuencia da historia de usuario "Deseñar padróns manualmente básico".....	56
Figura 18. Diagrama de secuencia da historia de usuario "Subir resultados básico".....	56
Figura 19. Diagrama de secuencia da historia de usuario "Ordenar resultados básico".	57
Figura 20. Diagrama de secuencia da historia de usuario "Eliminar experimento".	58
Figura 21. Configuración do usuario da web de administración.	65
Figura 22. Configuración do tamaño máximo de ficheiro a despreparar.....	65
Figura 23. Autenticación na web de administración de Tomcat.....	66
Figura 24. Xestor de aplicacións web de Tomcat.....	66
Figura 25. Menú para o despregue de aplicacións empaquetadas en WAR.	67
Figura 26. Primeira páxina da aplicación.....	67
Figura 27. Formulario de rexistro.....	68
Figura 28. Formulario de login.	68
Figura 29. Listado dos experimentos do usuario.....	69
Figura 30. Páxina de edición dos datos do experimento.....	69
Figura 31. Páxina de edición das placas do experimento.	70
Figura 32. Páxina de ordenado de arquivos.....	71
Figura 33. Estrutura de directorios a subir á aplicación.	71
Figura 34. Arquivos ordenados con condicións e mostrás.....	72
Figura 35. Arquivos ordenados con condicións.....	72
Figura 36. Arquivos ordenados con mostrás.	72
Figura 37. Arquivos ordenados sen condicións nin mostrás.	73

Índice de Táboas

Táboa 1. Estimación temporal.....	15
Táboa 2. Duración e datas estimadas das tarefas do proxecto.	16
Táboa 3. Dedicación temporal.	18
Táboa 4. Duración e datas de realización das tarefas do proxecto.	19
Táboa 5. Custo do hardware.....	21
Táboa 6. Custo do software.	21
Táboa 7. Custo do persoal.....	22
Táboa 8. Custo de impresión e encadernación.	22
Táboa 9. Custo total do proxecto.	22
Táboa 10. Planificación e seguimento das historias de usuario.....	33
Táboa 11. Caso de proba: UserTest.java.	59
Táboa 12. Caso de proba: ExperimentTest.java.	59
Táboa 13. Caso de proba: ConditionGroupTest.java.	60
Táboa 14. Caso de proba: SampleTest.java.	60
Táboa 15. Caso de proba: UserDAOTest.java.....	60
Táboa 16. Caso de proba: ExperimentDAOTest.java.....	61
Táboa 17. ConditionGroupDAOTest.java.	61
Táboa 18. Caso de proba: SampleDAOTest.java.	61
Táboa 19. Caso de proba: ReplicateDAOTest.java.....	62
Táboa 20. Caso de proba: ExperimentViewModelTest.java.	62
Táboa 21. BasicOutputSorterTest.java.	62
Táboa 22. ReplicatePathCreatorTest.java.	62

MEMORIA

1 Introducción

1.1 Identificación do traballo

Título: Asistente de Laboratorio para MALDI

Código: EI 13/14 - 024

Autor do TFG: Abel Fernández Nandín, 53186810-T

Titor do TFG: Dr. Miguel Reboiro Jato, 44477792-R

Curso: 2013/2014

Área de Linguaxes e Sistemas Informáticos

Departamento de Informática

Universidade de Vigo

1.2 Organización da documentación

Este traballo está documentado en tres partes:

- **Memoria:** documento formal no que se detalla todo o proceso de desenvolvemento do sistema, incluíndo as desviacións con respecto á planificación temporal e outros contratempos que puideran ter ocorrido.
- **Manual técnico:** documento formal que incluírá información puramente técnica referente á aplicación desenvolvida, co fin de facilitar as futuras tarefas de mantemento do sistema.
- **Manual de usuario:** documento informal orientado ao usuario no que se explicará como facer uso da ferramenta desenvolvida. Ademais, incluírá as instrucións necesarias para poder facer a instalación do sistema.

1.3 Marco da aplicación

Nos últimos anos a investigación en bioloxía sufriu unha importante revolución coa chegada da xenómica e da proteómica, que supuxeron o paso de facer investigacións sobre pequenos conxuntos de xenes ou proteínas a estudar miles destes compoñentes biolóxicos de forma simultánea, chegándose incluso a facer estudos de xenomas ou proteomas completos. Mentres que na xenómica nos atopamos cos microarrays de ADN e, máis recentemente, cos estudos de secuenciación masiva ou Next Generation Sequencing (NGS) como máximos expoñentes das técnicas de análise, na proteómica a técnica de maior relevancia é a espectrometría de masas.

Unha das técnicas máis destacadas de espectrometría de masas para proteómica é a denominada Matrix-Assisted Laser Desorption/Inonization (MALDI), especialmente na versión Time-Of-Flight (MALDI-TOF). Esta é unha técnica de ionización suave para a identificación e cuantificación de moléculas mediante a medición do seu tempo de voo a través dun campo electromagnético despois de seren ionizadas. Esta técnica resulta de especial interese pola súa capacidade para analizar biomoléculas, habitualmente

proteínas e péptidos, e moléculas orgánicas de maior tamaño. Ademais, destaca pola posibilidade de analizar decenas de mostras das que se obteñen medicións de centos de moléculas nunha única análise.

Un problema común á xenómica, proteómica e resto de ciencias ómicas é a gran cantidade de información xerada nunha única análise que dificulta a súa posterior análise e manexo. A única forma viable de realizar estas tarefas e mediante a aplicación de técnicas informáticas. Esta dependencia viuse reflectida nos últimos anos no importante auxe do campo da bioinformática.

Este traballo trata sobre a creación dunha ferramenta bioinformática de asistencia a investigadores durante o proceso de deposición de mostras en placas para a análise por MALDI-TOF. Este é un proceso que, habitualmente, consume unha gran cantidade de tempo, non só polo feito de ter que depositar unha gran cantidade de mostras, senón porque previamente é preciso rexistrar a posición de cada unha das mostras dentro da placa MALDI. Este proceso, que normalmente se fai de forma manual nun modelo en papel, serve para poder identificar a que mostra corresponde cada un dos ficheiros de resultados xerados pola análise, que tamén deberán ser ordenados de forma manual. Ademais, o feito de ter que facer este traballo manualmente leva a que os investigadores deposicionen as mostras de forma ordenada segundo a súa condición, algo que pode derivar na aparición de desviacións non desexadas nos resultados. Polo tanto, a posibilidade de automatizar tanto o proceso de creación de modelos de deposición de mostras, como a posterior organización dos ficheiros de resultados, preséntase de gran interese para os investigadores pola potencial redución do tempo necesario para preparar unha mostra e do número de erros de orixe humano.

1.4 Obxectivos da aplicación

O principal obxectivo deste traballo foi o desenvolvemento dunha aplicación que permitise deseñar como se faría a deposición de mostras dun experimento en placas MALDI e que facilitase a organización posterior dos resultados da análise.

Para acadar este obxectivo principal foi preciso superar os seguintes obxectivos previos:

- Permitir deseñar a forma na que se fará a deposición de mostras en placas MALDI. Este proceso debe resultar sinxelo e intuitivo para o usuario.
- Permitir a ordenación de mostras de xeito manual ou de forma automática baixo algún criterio determinado.
- Proporcionar un asistente que facilite a labor de deposición de mostras en placas MALDI no laboratorio segundo un deseño previo.
- Reorganizar os arquivos xerados polo MALDI para a súa análise posterior con ferramentas bioinformáticas.

1.5 Descrición técnica da aplicación

A aplicación que se desenvolveu está implementada sobre ZK, un *framework* para o desenvolvemento de aplicacións web en Java baseado en AJAX. A arquitectura ZK consta de dúas partes, cliente e servidor, tal e como se indica na Figura 1. A parte do cliente está formada polo explorador cliente, na que os usuarios interactúan cos *widgets*

propios de ZK, que se comunican co motor cliente de ZK. Este motor no cliente realiza peticións (e recibe respostas) AJAX ao servidor. Estas peticións son atendidas no servidor polo motor de actualización, que se comunica cos diferentes compoñentes de ZK para engadir eventos á cola de eventos. Deste xeito conséguese manter o estado dos compoñentes sincronizado entre o cliente e o servidor.

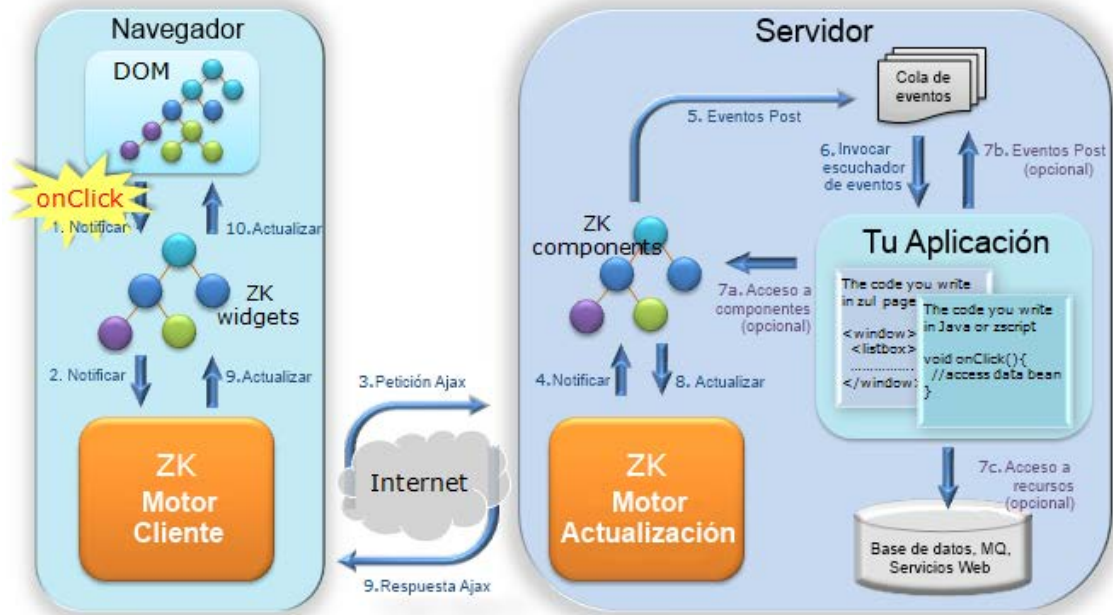


Figura 1. Arquitectura xeral dunha aplicación ZK¹.

Unha aplicación ZK pode acceder aos recursos do seu servidor, formar unha interface de usuario con compoñentes, escoitar as actividades do usuario, e despois manipular os compoñentes para actualizar a interface. Todas estas actividades pódense levar a cabo no servidor, e a sincronización do estado dos compoñentes entre o explorador cliente e o servidor é realizada por ZK automaticamente de forma transparente para o desenvolvedor.

Ao executarse no servidor, unha aplicación ZK ten acceso á pila completa de tecnoloxías Java habituais nos servidores de aplicacións. As tecnoloxías propias do usuario, como AJAX ou Server Push, son abstraídas en obxectos de evento. Por outra parte, a interface de usuario está composta de compoñentes POJO (Plain Old Java Objects), que fan que o seu manexo resulte sinxelo.

2 Metodoloxías

Para o proceso de implementación da aplicación empregouse a metodoloxía de desenvolvemento áxil Scrum adaptada ás restricións impostas pola normativa de TFGs á que está suxeito este traballo.

A decisión de empregar unha metodoloxías áxiles vén motivada por estar estas centradas na optimización do tempo de desenvolvemento, maximizando a labor creativa e minimizando aquelas tarefas que non aportan valor ao traballo. Por outra banda,

¹ Imaxe adaptada de http://books.zkoss.org/wiki/ZK_Developer%27s_Reference/Overture/Architecture_Overview

escolleuse Scrum por ser unha das metodoloxías áxiles máis utilizadas no ámbito empresarial, no que a diminución do tempo para a consecuente redución de custes é moi importante. Scrum ofrece, ademais, un marco de traballo sumamente flexible, algo que resulta de interese no presente traballo por ter este un perfil parcialmente exploratorio. Utilizáronse, como complemento a Scrum, as historias de usuario, que son a ferramenta de captura de requisitos máis habitual nas metodoloxías áxiles.

2.1 Metodoloxía Scrum

Scrum é unha metodoloxía áxil que está dirixida polo Backlog de Produto, o cal consiste nunha lista priorizada e completa de funcionalidades que se deben desenvolver para obter un produto. O feito de que a priorización sexa feita segundo o criterio do cliente fai que as características máis valiosas para este se desenvolvan en primeiro lugar. Por outra banda, é unha metodoloxía de desenvolvemento iterativa e incremental, na que cada iteración se desenvolve nun prazo de tempo fixo, denominado sprint, no que se implementan funcionalidades completas. O conxunto das funcionalidades desenvolvidas durante unha iteración constitúen, ao final da mesma, un incremento que dá lugar a un produto potencialmente publicable.

Como se comentou máis arriba, foi preciso aplicar certas adaptacións a esta metodoloxía para ter en conta as restricións impostas pola normativa á que está suxeito este traballo. A restrición máis importante é o feito de que este traballo ten carácter individual, xa que a metodoloxía Scrum está orientada a equipos compostos, normalmente, por entre 5 e 9 membros. Por este motivo, adoptáronse as seguintes adaptacións:

- O equipo estivo formado polo autor e o director deste traballo. Sendo o primeiro o responsable da parte relacionada co desenvolvemento do mesmo e o segundo da definición dos requisitos, actuando como cliente.
- As reunións diarias pasaron a realizarse dúas veces á semana. Posto que o obxectivo das mesmas é que todo o equipo de desenvolvemento estea ao tanto do traballo realizado por todos os membros, nun equipo tan reducido o posible beneficio vese diminuído e, polo tanto, a realización diaria suporía un custo de tempo excesivo.

2.2 Historias de usuario

As historias de usuario son unha ferramenta que describe unha funcionalidade que ten valor para o usuario ou cliente dunha aplicación software. Son a base para a definición das funcións que o sistema debe ofrecer, e facilitan a xestión dos requisitos. Deben escribirse de forma que se lles poda asignar un valor e que o desenvolvedor poida estimalas, e é habitual que evolucionen ó longo dun proxecto, de maneira que se dividan, desaparezan, ou aparezan outras novas. As historias de usuario deben presentar as seguintes cualidades:

- **Independentes:** debe evitarse crear dependencias entre historias de usuario, pois dificultan a planificación, estimación, e priorización.
- **Negociables:** non son contratos cerrados, son simples recordatorios dunha funcionalidade.

- **Valiosas:** deben xerar valor para o cliente, de forma que poida priorizar as tarefas segundo o seu valor.
- **Estimables:** os desenvolvedores deben ser capaces de estimar o tempo que lles levará completalas, de maneira que se poida realizar unha planificación.
- **Pequenas:** para unha correcta planificación é recomendable que o tamaño dunha historia estea entre medio día e dúas semanas.
- **Comprobables:** deben definirse probas, automáticas sempre que se poida, que o feito de superalas supoña que a funcionalidade foi desenvolvida con éxito.

3 Tecnoloxías empregadas

3.1 ZK Framework

ZK é un framework de código aberto escrito en Java, que permite a creación de interfaces gráficas de usuario para aplicacións web. O núcleo de ZK consiste nun mecanismo dirixido por eventos baseado en AJAX, 123 ficheiros XUL e 83 compoñentes basados en XHTML, e unha linguaxe de marcado para deseñar as interfaces de usuario. Os programadores deseñan as páxinas da aplicación en compoñentes XUL/XHTML enriquecidos, e os manipulan mediante eventos activados por actividades do usuario, dunha forma similar ó modelo de programación que se atopa nas aplicacións gráficas de escritorio.

3.2 Spring Framework

Spring é un *framework* para o desenvolvemento de aplicacións e un contedor de inversión de control para a plataforma Java. Entre as súas principais funcionalidades atópanse: inxección de dependencias, programación orientada a aspectos, servizos web *RESTful*, e soporte para JDBC, JPA e JMS.

3.3 Hibernate ORM

Hibernate é unha biblioteca de funcións para a linguaxe Java, que ofrece un *framework* para o mapeado dun modelo de dominio orientado a obxectos a unha base de datos tradicional. A característica principal de Hibernate é a de mapear clases Java a táboas dunha base de datos, e tipos de datos de Java a tipos de datos de SQL. Hibernate ofrece, ademais, capacidades para a xeración de sentencias SQL, liberando ao programador das responsabilidades da xestión do conxunto de resultados e da conversión de obxectos.

3.4 JUnit

JUnit é un framework de probas de unidade para a linguaxe de programación Java. Foi importante na evolución do desenvolvemento dirixido por probas (TDD), e pertence á familia de frameworks de probas de unidade coñecida como xUnit.

3.5 DbUnit

DbUnit é unha extensión de JUnit enfocada a proxectos nos que ten moita importancia a base de datos, facendo que esta teña un estado coñecido de antemán entre

as execucións de cada proba. Esta é unha forma excelente de evitar unha gran cantidade de problemas que poden ocorrer cando un caso de proba corrompe a base de datos e causa que as seguintes probas fallen.

3.6 Apache Maven

Maven é unha ferramenta de construción automática de software utilizada principalmente para proxectos Java. Maven encárgase de dous aspectos da construción de software: como se constrúe, e cales son as súas dependencias. Un ficheiro XML describe o proxecto software que se vai construír, as súas dependencias de outros módulos externos e compoñentes, orde de construción, directorios, e *plugins* que se poidan precisar.

3.7 Git e Github

Git é un sistema de control de versións distribuído. A selección deste sistema de control de versións sobre outros foi debida á súa sinxeleza, potencia e á posibilidade de integración con distintas contornas de desenvolvemento.

Por outra banda, escolleuse Github (<https://github.com/>) como repositorio remoto, pola súa popularidade e polas opcións que proporciona. Todo o código deste proxecto pódese atopar alocado en: <https://github.com/mirkeet/MALDILabAssistant>

3.8 Kunagi

Kunagi é unha ferramenta web para a xestión de proxectos e colaboración integrada baseada en Scrum. O feito de que sea integrada significa que permite ós usuarios xestionar todo o proxecto utilizando Kunagi como a única ferramenta. Para tal fin, complementase Scrum con outras boas prácticas para abranguer tódalas necesidades da xestión de proxectos.

Kunagi está enfocado, pero non limitado, a proxectos software. A natureza lixeira da xestión áxil de proxectos permite que poida ser utilizado tanto por profesionais como por non profesionais, causando unha carga de traballo adicional moi reducida no proxecto. De este modo, é adecuado para proxectos pequenos feitos por equipos remotos, repartidos polo mundo.

4 Planificación e Presuposto

4.1 Planificación

En Scrum asúmese a dificultade de realizar unha planificación realista ao comezo do proxecto, polo que se evita que esta sexa moi detallada. Pola contra, a planificación do proxecto distribúese ao longo do mesmo en varias reunións de planificación nas que participa todo o equipo. Estas reunións, principalmente, resúmense en reunións de planificación de publicación (*release*), na que se priorizan e seleccionan as funcionalidades que formarán parte da próxima publicación do sistema, e as reunións de planificación de *sprint*, nas que se deciden as funcionalidades a desenvolver no *sprint*. As reunións de publicación adóitanse celebrar cada tres ou seis meses, mentres que as de *sprint* se realizan ao comezo de cada iteración.

É importante ter en conta que na planificación de Scrum non se separan as distintas etapas de desenvolvemento de funcionalidades (p.ex. análise, deseño, implementación, etc.) como é habitual noutras metodoloxías. Pola contra, prefírese eliminar esta separación para favorecer que, por exemplo, o deseño, implementación e probas se realicen de forma conxunta e sinérxica.

4.1.1 Planificación estimada

Neste traballo considerouse que a primeira publicación do sistema tería lugar cando finalice o mesmo e que, polo tanto, todos os *sprints* realizados estarían asociados a esta publicación. En base a isto, as tarefas a realizar no presente proxecto por orde de planificación serían as seguintes:

1. Modelado do negocio no que se crearán as historias de usuario e dos criterios de aceptación das mesmas.
2. Investigación das posibles tecnoloxías a utilizar.
3. Planificación da publicación, na que se fará a estimación e priorización das historias de usuario.
4. O primeiro sprint.
5. O segundo sprint.
6. O terceiro e último sprint.

A elaboración da documentación realizárase durante toda a duración do proxecto. A duración estimada total foi de 300 horas, nas que se dedicarían 20 horas semanais, dando lugar a 15 semanas de desenvolvemento, tal e como se amosa na Táboa 1. Co fin de cumprir estas 20 horas semanais, dedicaríanse 4 horas ao día, 5 días a semana. A planificación anteriormente descrita recóllese na Táboa 2 e, de forma gráfica, na Figura 2.

Fase	Estimación temporal	
	(en días)	(en semanas)
Modelado de negocio	5	1
Investigación das posibles tecnoloxías a utilizar	5	1
Planificación da publicación	5	1
Sprint 1	20	4
Sprint 2	20	4
Sprint 3	20	4
TOTAL	75	15

Táboa 1. Estimación temporal.

☐ Proxecto	75d	17/02/2014	30/05/2014
Modelado de negocio	1s	17/02/2014	21/02/2014
Investigación das posibles tecnoloxías a utilizar	1s	24/02/2014	28/02/2014
Planificación da publicación	1s	03/03/2014	07/03/2014
Sprint 1	4s	10/03/2014	04/04/2014
Sprint 2	4s	07/04/2014	02/05/2014
Sprint 3	4s	05/05/2014	30/05/2014
☐ Documentación	75d	17/02/2014	30/05/2014
Manual técnico	15s	17/02/2014	30/05/2014
Manual de usuario	1s	26/05/2014	30/05/2014
Memoria	1s	26/05/2014	30/05/2014

Táboa 2. Duración e datas estimadas das tarefas do proxecto.

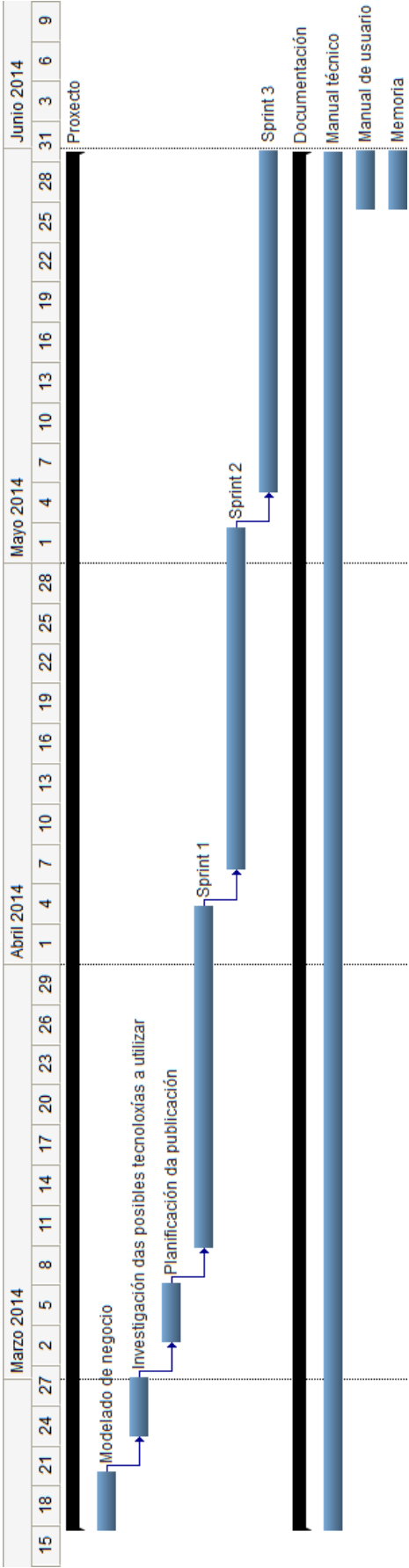


Figura 2. Diagrama de Gantt da planificación estimada.

4.1.2 Seguimento temporal

O comezo do primeiro sprint retrasouse do 10 ao 28 de marzo, o cal supuña aumentar o número de horas que habería que dedicar diariamente ou retrasar a data de finalización do traballo. Neste caso decidiuse retrasar a finalización do traballo ata o 25 de Xuño. Ademais, unha vez estimado o tamaño das historias de usuario, chegouse a conclusión de que sería máis conveniente facer 4 sprints de 3 semanas en vez de os 3 sprints de 4 semanas que se planificaran. Como consecuencia, as tarefas a realizar no presente proxecto por orde de planificación quedarían da seguinte maneira:

1. Modelado do negocio no que se crearán as historias de usuario e dos criterios de aceptación das mesmas.
2. Investigación das posibles tecnoloxías a utilizar.
3. Planificación da publicación, na que se fará a estimación e priorización das historias de usuario.
4. O primeiro sprint.
5. O segundo sprint.
6. O terceiro sprint.
7. O cuarto e último sprint

A elaboración da documentación realizaríase durante o cuarto e último sprint. A duración total foi de 310 horas, nas que se dedicarían aproximadamente 3 horas e 45 minutos diarios, dando lugar a 82 días de desenvolvemento, tal e como se amosa na Táboa 3. O seguimento anteriormente descrito recóllese na Táboa 4 e, de forma gráfica, na Figura 3.

Fase	Dedicación temporal	
	(en días)	(en semanas)
Modelado de negocio	5	1
Investigación das posibles tecnoloxías a utilizar	10	2
Planificación da publicación	5	1
Sprint 1	15	3
Sprint 2	17	3,4
Sprint 3	15	3
Sprint 4	15	3
TOTAL	82	16,4

Táboa 3. Dedicación temporal.

<input type="checkbox"/> Proxecto	82d	28/02/2014	23/06/2014
Modelado de negocio	5d	28/02/2014	06/03/2014
Investigación das posibles tecnoloxías a utilizar	10d	07/03/2014	20/03/2014
Planificación da publicación	5d	21/03/2014	27/03/2014
Sprint 1	15d	28/03/2014	17/04/2014
Sprint 2	17d	18/04/2014	12/05/2014
Sprint 3	15d	13/05/2014	02/06/2014
Sprint 4	15d	03/06/2014	23/06/2014

Táboa 4. Duración e datas de realización das tarefas do proxecto.

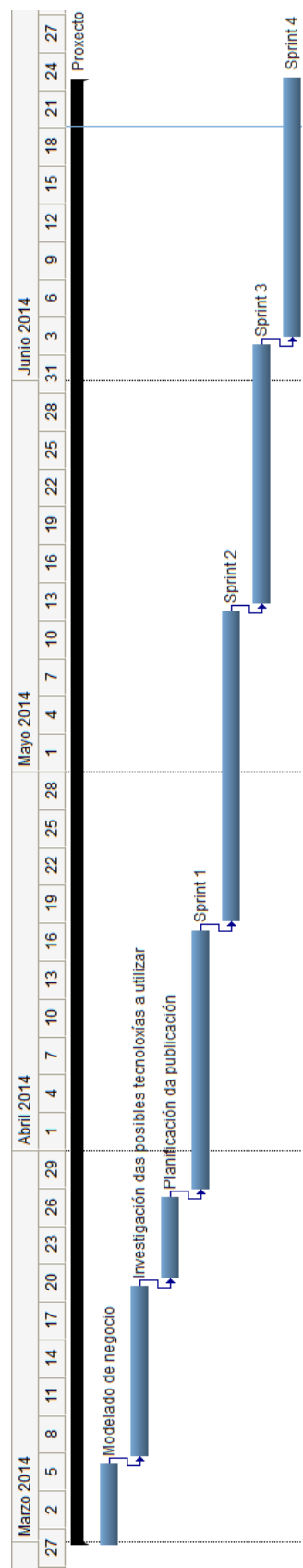


Figura 3. Diagrama de Gantt do seguimento temporal.

4.2 Presuposto

Os custos dun proxecto con estas características poden dividirse en custos de: hardware, software, persoal, impresión e encadernación; tal e como se amosa nas seguintes seccións.

4.2.1 Custo do hardware

A Táboa 5 describe o único compoñente hardware utilizado no traballo, un ordenador portátil. Dado que o hardware é susceptible de continuar a súa vida útil unha vez rematado o proxecto, calcúlase a súa amortización. Estímase unha vida útil de 4000 horas e o custo total é de 875 €; tendo en conta que o traballo tivo unha duración de 310 horas, a súa amortización é de 67,81 €.

Compoñente	Tipo	Custo total	Amortización
Ordenador portátil	Dell Inspiron 17R SE	875 €	67,81 €

Táboa 5. Custo do hardware.

4.2.2 Custo do software

Na Táboa 6 pódese observar un listado con todo o software utilizado na realización do proxecto. Cabe destacar que non houbo custo algún, ben por tratarse de software libre ou gratuito, ou ben por utilizar licenzas proporcionadas pola universidade.

Software	Custo
Windows 8 (Licenza da universidade)	0 €
VirtualBox	0 €
Linux Mint 16	0 €
Eclipse	0 €
GitHub	0 €
Kunagi	0 €
Gantter	0 €
Visual Paradigm (Licenza da universidade)	0 €
Microsoft Office 2013 (Licenza da universidade)	0 €
TOTAL	0 €

Táboa 6. Custo do software.

4.2.3 Custo do persoal

Para calcular o custo total do persoal do proxecto decidiuse establecer un prezo por hora de traballo de 15 €, tal e como se amosa na Táboa 7.

Perfil	Nº de Horas	Custo / hora	Custo total
Desenvolvedor Júnior	310	15 €	4650 €

Táboa 7. Custo do persoal.

4.2.4 Custo de impresión e encadernación

Os custos de impresión e encadernación do traballo pódense observar a continuación na Táboa 8.

Concepto	Custo
Impresión	50 €
Encadernación	40 €
TOTAL	90 €

Táboa 8. Custo de impresión e encadernación.

4.2.5 Custo total

Na Táboa 9 pódese observar o custo total do proxecto, o cal foi obtido sumando os custos parciais expostos anteriormente, e engadindo o custo de imprimir e encadernar a documentación.

Concepto	Custo
Hardware	67,81 €
Software	0 €
Persoal	4650 €
Impresión e encadernación	90 €
TOTAL	4807,81 €

Táboa 9. Custo total do proxecto.

5 Problemas atopados e solucións aportadas

Un dos obxectivos do traballo era “Proporcionar un asistente que facilite a labor de deposición de mostras en placas MALDI no laboratorio segundo un deseño previo”, o cal non se puido levar a cabo por falta de tempo. A solución é seguir o fluxo de traballo de Scrum, é dicir, deixalo recollido no *Backlog* de Produto para facelo no futuro.

Respecto aos aspectos técnicos, a maior dificultade foi a creación do compoñente gráfico para o deseño de placas. Para o seu desenvolvemento identificáronse tres posibles alternativas: *i)* crear un compoñente completamente novo, *ii)* crear un compoñente baseado no compoñente básico Grid ou *iii)* crear un compoñente baseado no compoñente avanzado Spreadsheet. A solución *i)* deixouse como última opción xa que, aínda que é a que proporciona maior flexibilidade, tamén é a de maior complexidade. Por outra banda, a solución *ii)* descartouse pola dificultade para realizar seleccións de celas neste compoñente. Por último, a opción finalmente escollida foi a *iii)*, xa que este compoñente proporcionaba tódalas funcionalidades requiridas. Aínda así, a adaptación

deste compoñente non resultou sinxela e tívose que crear un compoñente derivado e un *listener* de eventos especialmente adaptado para cubrir as necesidades do sistema.

Outros compoñentes cuxo desenvolvemento supuxo certas dificultades foron as árbores de elementos do experimento. Concretamente aquelas que mostran as condicións, mostras e réplicas relacionadas con un experimento nas vistas de edición dun experimento e de deseño de placas. En ZK non é posible actualizar un único elemento dunha árbore de forma directa, sendo necesario recargar toda a árbore cando se engade ou elimina un elemento. Isto facía que resultase pouco práctico para o usuario, pois con cada cambio a árbore recargábase e todos os nodos aparecían pechados. A solución para que a actualización fose máis natural foi combinar o uso dun ListModel (o modelo que usan os compoñentes que visualizan árbores) co patrón observador, o que permitiu un gran fino nas notificacións entre o modelo e o compoñente, evitando a recarga completa do modelo.

6 Futuras ampliacións

A continuación descríbense as historias de usuario que non foron completadas neste traballo pero que están recollidas no *Backlog* de Produto, cos seus correspondentes criterios de aceptación. Están descritas por orde de importancia.

6.1 Depositar mostras

“Como técnico quero que o sistema me guíe cando deposito as mostras dun experimento MALDI en placas para que me resulte máis sinxelo e cometa menos erros.”

Criterios de aceptación:

- O sistema, mostra a mostra, ensina a súa situación nas placas.
- Tódalas mostras foron recorridas ó final da asistencia.

6.2 Crear experimento avanzado

“Como técnico quero crear un experimento especificando o número de condicións, o número de réplicas por mostra, e de mostras por condición; para que preparar mostras e analizar con MALDI, e organizar os resultados da análise, resulte máis sinxelo e con menos erros.”

Criterios de aceptación:

- É posible crear experimentos con entre 1 e 10 condicións, entre 1 e 100 mostras por condición, e entre 1 e 10 réplicas por mostra, de modo que o tamaño de condición e mostra sexa o mesmo sempre.
- Non pode haber dúas condicións cos mesmo nome.
- Non pode haber dúas mostras co mesmo nome.
- Non pode haber dúas réplicas co mesmo nome.
- O nome das mostras e réplicas xérase de forma automática en base ao nome das condicións.

6.3 Subir resultados avanzado

“Como técnico quero subir os ficheiros de resultados dunha análise MALDI escollendo o formato de espectro e podendo asociar os ficheiros coas réplicas para poder ordenalos posteriormente.”

Criterios de aceptación:

- É posible seleccionar varios formatos de espectro (p. ex. CSV, mzML, mzXML, etc.).
- A asociación de ficheiros con réplicas farase mediante expresións regulares dentro das cales se poderán introducir, ó menos, os comodíns {pos}, {col}, {row}.

6.4 Ordenar resultados avanzado

“Como técnico quero poder ordenar os ficheiros dos espectros de forma automática e segundo os nomes ou ids das condicións, mostras, ou réplicas, e descargalos para poder analizar posteriormente dichos resultados.”

Criterios de aceptación:

- O usuario pode definir á árbore de directorios a crear utilizando os nomes ou ids de condición, mostra, e réplica. Por exemplo: {C:NAME}/{S:NAME}/{R:ID}.csv indica que se creará un directorio co nome de cada condición, dentro do cal se creará un directorio co nome de cada mostra da condición, dentro do cal estará o ficheiro de cada réplica que terá como nome identificador da réplica e como extensión “csv”.
- O resultado pódese descargar en formato TAR.

6.5 Diseñar padróns aleatoriamente

“Como técnico quero que o sistema deseñe os padróns dun experimento de forma automática colocando as réplicas de forma aleatoria para utilizar posteriormente o padrón como guía durante o depósito de mostras na placa real, e que o proceso de deseño me consuma pouco tempo.”

Criterios de aceptación:

- Tódalas réplicas se colocarán automaticamente e de forma aleatoria no padrón.
- Cada mostra e, por extensión, réplica terá o color no padrón asociado a súa condición.
- Crearase o número mínimo de placas necesario para conter tódalas mostras do experimento.
- O usuario debe ter a opción de que tódalas réplicas de cada mostra se coloquen no mesmo padrón.
- O deseño só se considera finalizado se tódalas réplicas de tódalas mostras foron colocadas nos padróns.

6.6 Modificar experimento

“Como técnico quero modificar os datos dun experimento para adaptalo ós posibles cambios que poidan xurdir durante a súa realización.”

Criterios de aceptación:

- Tódolos datos do experimento poden ser modificados.
- Se a modificación do experimento afecta ó tamaño dos padróns e existe un deseño, debe advertirse de que se perderá o deseño antes de facer efectiva a modificación.
- Se se elimina algunha das mostras ou condicións que foron colocadas nun deseño debe advertirse ó usuario antes de facer efectiva a modificación.

6.7 Datos e configuración persoais

“Como técnico quero ver e modificar os meus datos persoais e configuración por defecto para mantelos actualizados.”

Criterios de aceptación:

- Tódolos datos persoais do usuario son modificables.
- A modificación da configuración por defecto ten un efecto inmediato.

6.8 Diseñar padróns manualmente avanzado

“Como técnico quero facer o padrón dun experimento MALDI seleccionando manualmente a posición de cada réplica das mostras para utilizar posteriormente o padrón como guía durante o depósito de mostras na placa real.”

Criterios de aceptación:

- É posible colocar unha mostra, de modo que tódalas súas réplicas aínda non colocadas se sitúen no padrón.
- É posible seleccionar se a colocación das mostras se fai por filas ou por columnas.
- É posible eliminar unha condición do padrón, de forma que se eliminen tódalas súas mostras.
- É posible eliminar unha mostra do padrón, de forma que se eliminen tódalas súas réplicas.
- É posible eliminar unha réplica do padrón.
- É posible mostrar e configurar un “grid”. Isto é, unhas liñas de axuda horizontais cada X filas e verticais cada Y columnas.
- Cando se detecta unha colisión ó colocar unha mostra, se informará claramente ó usuario da mesma.

7 Conclusións

A finalidade deste proxecto era proporcionar unha ferramenta software para o deseño de placas MALDI que fose máis eficiente que facelo a man, en papel. Con tan modesto obxectivo é difícil dubidar do éxito do proxecto, pois efectivamente proporciona

tal ferramenta. Se ben non proporciona tódalas funcionalidades que se tivera pensado nun principio, ofrece o fluxo de traballo mínimo para que sexa utilizable, e constitúe unha boa base para a ampliación das súas funcionalidades, establecendo como punto de partida as historias de usuario anteriormente expostas no apartado de Futuras ampliacións.

No ámbito técnico este proxecto foi enriquecedor para min a nivel profesional, pois tiven a oportunidade de utilizar algunhas das tecnoloxías máis utilizadas no marco empresarial, como poden ser Spring, Hibernate, ou JUnit. Noustante, a maior dificultade coa que me atopei foi a integración de estas tecnoloxías cada vez que se avanzaba na implementación pois, en canto o fluxo de traballo se desviaba minimamente daquel para o que estaba pensado a tecnoloxía, a súa integración co resto volvíase moitísimo máis complicada. Isto tivo como consecuencia en varias ocasións o descarte dunha funcionalidade unha vez xa estaba implementada a súa lóxica, pois non se atopaba xeito de integrala co resto.

En canto ó proceso de desenvolvemento, no que cabe destacar o uso da metodoloxía Scrum, só podo dicir vantaxes con respecto as miñas experiencias anteriores con outras metodoloxías. Ofrece un fluxo de traballo extremadamente flexible, que nunca supuxo un obstáculo no desenvolvemento. Baixo o meu punto de vista, a gran vantaxe das metodoloxías áxiles é a honestidade, o feito de aceptar que un cliente quere unha solución a unha necesidade mediante un produto de calidade. De este xeito non se trata de enganalo mediante a firma de contratos que aseguren a inmutabilidade dos requisitos, senón de escoitalo ó longo do desenvolvemento para construír un produto que lle aporte o máximo valor posible.

8 Bibliografía

- [1] - <http://www.zkoss.org/documentation>. Documentación de ZK Framework.
- [2] - <https://spring.io/docs>. Documentación de Spring.
- [3] - <http://hibernate.org/orm/documentation>. Documentación de Hibernate.
- [4] - <http://www.manning.com/tahchiev>. TAHCHIEV, Petar. “JUnit in action” .
- [5] - <http://www.dbunit.org/project-info.html>. Documentación de dbUnit.
- [6] - <http://agilemanifesto.org/iso/es>. Manifesto Áxil.
- [7] - <https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/2013/Scrum-Guide-ES.pdf>. SCHWABER, Ken & SUTHERLAND, Jeff. La Guía de Scrum.
- [8] - COHN, Mike. User Stories Applied: For Agile Software Development.
- [9] – S. RUBIN, Kenneth. Essential Scrum.

MANUAL TÉCNICO

9 Análise

A continuación descríbense as funcionalidades da aplicación desenvolvida, documentadas en forma de historias de usuario, as cales son proporcionadas polo cliente. Nas seccións que aparecen a continuación descríbense os roles de usuario das historias, as propias historias cos seus criterios de aceptación, e a estimación e dedicación temporal de cada unha.

9.1 Roles de usuario

Neste proxecto identificáronse dous tipos de usuarios interesados e que, polo tanto, participan nas historias de usuario definidas. Os intereses e relación co sistema de cada tipo de usuarios son os seguintes:

- **Técnico de laboratorio (técnico):** é un usuario que fai análises con MALDI e quere usar a aplicación para planificar ditas análises. É probable que os seus coñecementos de informática sexan baixos e tan só teña experiencia no uso de ferramentas ofimáticas.
- **Programador:** abrangue tanto ó desenvolvedor do sistema como todos aqueles que no futuro vaian realizar tarefas de ampliación ou mantemento da aplicación, ou que queiran utilizar a súa API.
- **Alumno:** é o responsable de desenvolver o sistema. O seu interese no sistema está en que será entregado como Traballo de Fin de Grao (TFG) para completar os estudos de Graduado en Enxeñaría Informática. O motivo de que se inclúa como usuario é que certas historias de usuario estarán relacionadas coas esixencias do TFG (p. ex. documentar o sistema).

9.2 Historias de usuario

A continuación descríbense as historias de usuario completadas no traballo, cos seus correspondentes criterios de aceptación. Están descritas por orde de realización, que coincide coa orde de importancia.

9.2.1 HU01 - Conta persoal

“Como técnico quero ter unha conta persoal para que os datos dos meus experimentos sexan privados.”

Criterios de aceptación:

- Un usuario pode crear unha conta no sistema.
- Un usuario pode acceder á súa conta coas súas credenciais.
- Un usuario pode saír da súa conta, de modo que xa non se poida acceder ós seus datos.

9.2.2 HU02 - Listar experimentos

“Como técnico quero ver o listado dos meus experimentos para poder continuar traballando con eles ou revisar un traballo previo.”

Criterios de aceptación:

- Todos os experimentos do usuario aparecen inicialmente no listado.
- O listado amosa, polo menos, o título e a descrición de cada experimento.
- O listado pode filtrarse segundo o texto que se buscará no título dos experimentos.
- Ó seleccionar un experimento poderase continuar traballando con el, amosando os datos do experimento.

9.2.3 HU03 - Crear experimento básico

“Como técnico quero crear un experimento especificando as mostras e réplicas en tódalas condicións de forma individual para que preparar mostras e analizar con MALDI, e organizar os resultados da análise, resulte máis sinxelo e con menos erros.”

Criterios de aceptación:

- Cando se introducen os datos obrigatorios do experimento se crea na conta do usuario.
- Mentres non se introduzan tódolos datos obrigatorios dun experimento non é posible creado.
- É posible seleccionar o tamaño das placas en número de filas e columnas.
- É posible seleccionar o tipo de fila e columna entre numérico (1, 2, 3...), minúsculas (a, b, c...) ou maiúsculas (A, B, C...).
- Non pode haber dúas condicións cos mesmo nome.
- Non pode haber dúas mostras co mesmo nome.
- Non pode haber dúas réplicas co mesmo nome.
- O nome das condicións xérase automaticamente.
- O nome das mostras xérase automaticamente.
- O nome das réplicas xérase automaticamente.
- O usuario pode editar o nome das condicións, mostras, e réplicas.

9.2.4 HU04 - Implementar probas de DAOs

“Como programador quero ter probas de unidade do código de DAO para facilitar o seu posterior mantemento.”

Criterios de aceptación:

- Hai unha clase de proba por cada DAO, e todas as probas pasan correctamente.

9.2.5 HU05 - Implementar probas de Entidades e ViewModels

“Como programador quero ter probas de unidade do código de Entidades e ViewModels para facilitar o seu posterior mantemento”

Criterios de aceptación:

- Hai unha clase de proba por cada Entidade, e todas as probas pasan correctamente.
- Hai unha clase de proba por cada ViewModel, e todas as probas pasan correctamente.

9.2.6 HU06 - Diseñar padróns manualmente básico

“Como técnico quero facer o padrón dun experimento MALDI seleccionando manualmente a posición de cada réplica das mostras para utilizar posteriormente o padrón como guía durante o depósito de mostras na placa real.”

Criterios de aceptación:

- Unha mesma réplica non pode ser colocada varias veces no padrón.
- Cada mostra e, por extensión, réplica, terá a cor no padrón asociada a súa condición.
- Crearase o número mínimo de placas necesario para conter tódalas mostras do experimento.
- O deseño só se considera finalizado se tódalas réplicas de tódalas mostras de tódalas condicións han sido colocadas nos padróns.

9.2.7 HU07 - Subir resultados básico

“Como técnico quero subir os ficheiros de resultados dunha análise MALDI para poder ordenalos posteriormente.”

Criterios de aceptación:

- Os resultados pódense subir en ficheiros comprimidos en formato ZIP e TAR.
- O ficheiro comprimido debe conter un directorio por cada placa, dentro do cal se atopen os espectros xerados.
- O directorio de cada padrón ten que ter o mesmo nome que o padrón.
- Calquera error de formato se notificará de forma clara ó usuario.

9.2.8 HU08 - Ordenar resultados básico

“Como técnico quero poder ordenar os ficheiros dos espectros de forma automática e segundo distintos criterios e descargalos para poder analizar posteriormente ditos resultados.”

Criterios de aceptación:

- O usuario pode definir á árbore de directorios e crear ou non directorios para condicións e/ou mostras.
- O resultado pódese descargar en formato ZIP e TAR.

9.2.9 HU09 - Eliminar experimento

“Como técnico quero eliminar un experimento para manter a miña conta limpa de experimentos antigos ou que xa non vou realizar.”

Criterios de aceptación:

- Antes de eliminar un experimento pedirase confirmación ó usuario para evitar borrados accidentais.
- O experimento será eliminado e xa non aparecerá na conta do técnico.

9.2.10 HU10 - Memoria TFG

“Como alumno quero crear un documento onde se recolla todo o proceso de desenvolvemento en comparación co planificado, xunto coas conclusións e traballo futuro para cumprir coa normativa de TFG.”

Criterios de aceptación:

- A documentación está feita e revisada polo director.
- A documentación está impresa e encadernada.

9.2.11 HU11 - Manual técnico TFG

“Como alumno quero crear un documento onde se describa de forma técnica o sistema para cumprir coa normativa de TFG e para facilitar o futuro mantemento e ampliación do sistema.”

Criterios de aceptación:

- A documentación está feita e revisada polo director.
- A documentación está impresa e encadernada.

9.2.12 HU12 – Manual de usuario TFG

“Como alumno quero crear un documento onde se describa como utilizar o sistema para cumprir coa normativa de TFG e para facilitar o uso do mesmo aos usuarios.”

Criterios de aceptación:

- A documentación está feita e revisada polo director.
- A documentación está impresa e encadernada.
- A documentación está incluída no sistema como axuda para o usuario.

9.3 Planificación e seguimento das historias de usuario

Na Táboa 10, que se presenta a continuación, recóllense tódalas historias de usuario completadas no proxecto agrupadas por sprint, indicando para cada unha o número de horas que se estimou que levaría completalas, fronte ao número de horas finalmente dedicado.

Sprint	Historia de usuario	Nº de horas estimadas	Nº de horas dedicadas
1	HU01 - Conta persoal	8	8
	HU02 - Listar experimentos	8	10
	HU03 - Crear experimento básico	20	22
2	HU04 - Implementar probas de DAOs	4	12
	HU05 - Implementar probas de Entidades e ViewModels	4	15
	HU06 - Deseñar padróns manualmente básico	20	31
3	HU07 - Subir resultados básico	12	30

4	HU08 - Ordenar resultados	12	35
	HU09 - Eliminar experimento	2	1
	HU10 - Memoria TFG	20	20
	HU11 - Manual técnico TFG	20	26
	HU12 - Manual de usuario TFG	20	20

Táboa 10. Planificación e seguimento das historias de usuario.

10 Deseño

10.1 Arquitectura xeral de ZK

ZK ofrece soporte para o patrón de arquitectónico Model-View-ViewModel (MVVM) e, posto que o asistente de laboratorio para MALDI foi desenvolvido utilizando este patrón, cómpre describir o seu funcionamento, o cal se amosa de forma gráfica na Figura 4.

MVVM automatiza as tarefas de enlazamento de datos que os desenvolvedores terían que implementar nun controlador típico do patrón MVC, e divide a aplicación en catro partes:

- **Model:** está formado polos datos e as regras de negocio.
- **View:** constitúe a interface de usuario, é un ficheiro .zul que contén compoñentes ZK descritos nunha linguaxe XML propia de ZK: ZUL. Cando o usuario interacciona con ditos compoñentes activa eventos manexados polo binder.
- **ViewModel:** é unha abstracción da View, que contén o seu estado e comportamento. É responsable de amosar os datos do Model á View e ofrece as accións requeridas pola View.
- **Binder:** é o compoñente responsable de sincronizar os datos entre o ViewModel e a View, e xestiona eventos automaticamente segundo as expresións definidas para o enlazado de datos. Non é necesario controlar os compoñentes un mesmo. A diferenza do resto, este é un compoñente que proporciona o *framework*.

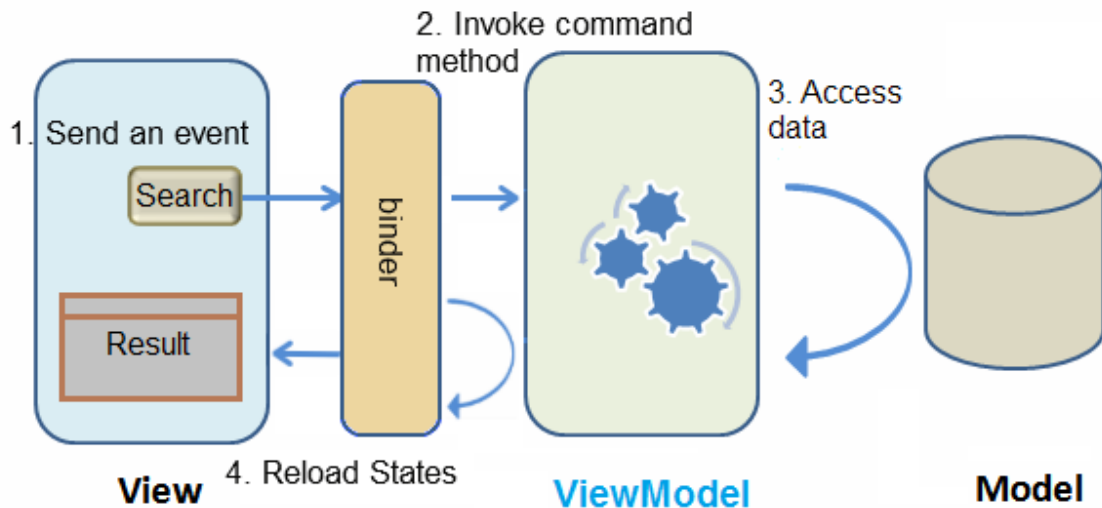


Figura 4. Arquitectura do patrón de deseño Model-View-ViewModel².

10.2 Arquitectura do sistema desenvolvido

A arquitectura do Asistente de Laboratorio para MALDI, a pesar de estar implementado en ZK, non se corresponde exactamente coa arquitectura do patrón Model-View-ViewModel. A diferencia radica nunha mellora do Model, que se ve dividido en Service, DAO, EntityManager, e Entities, tal e como se pode observar na Figura 5.

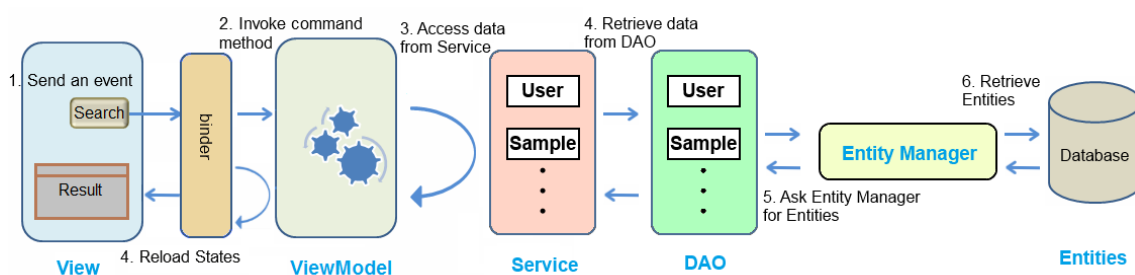


Figura 5. Arquitectura do Asistente de Laboratorio para MALDI.

De esta maneira, os ViewModels pasan a pedir os datos ós Servizos de cada entidade da base de datos, que os obteñen a partir dos seus correspondentes DAOs. Estes teñen acceso ó EntityManager, o cal abstrae consultas á base de datos, permitindo xestionar as Entities que nela se atopan.

Nas capas DAO - EntityManager - Entities utilízase a técnica Object-Relational Mapping(ORM) mediante a API de Persistencia de Java (JPA); máis concretamente, utilízase Hibernate como implementación de JPA. O que aporta esta técnica é a xestión das entidades da base de datos como se fosen obxectos Java, de maneira que se poidan administrar dende o propio linguaxe de programación.

Por outra banda, en tódalas capas lóxicas da aplicación (ViewModel - Service - DAO - EntityManager - Entities), se utiliza o patrón de Inxección de Dependencias, o cal aporta ó sistema Inversión de Control. De esta maneira, a responsabilidade de crear as

² Imaxe tomada de http://www.zkoss.org/zkdemo/getting_started/mvvm

dependencias pasa do desenvolvidor ó framework Spring, que é precisamente a librería que dota ó sistema desta Inversión de Control. Esta técnica aporta vantaxes substanciales ó desenvolvidor, como a prevención de erros cando se substitúe un módulo software por outro, ou para desacoplar a execución dunha tarefa da súa implementación.

Os diferentes fluxos de traballo que poden ter lugar no Asistente de Laboratorio para MALDI teñen moitas similitudes, porque a maioría involucran a tódalas capas da aplicación. Desta maneira, na posterior sección de Vista dinámica, será frecuente ver unha orde de execución de métodos similar a que se pode observar na Figura 6.

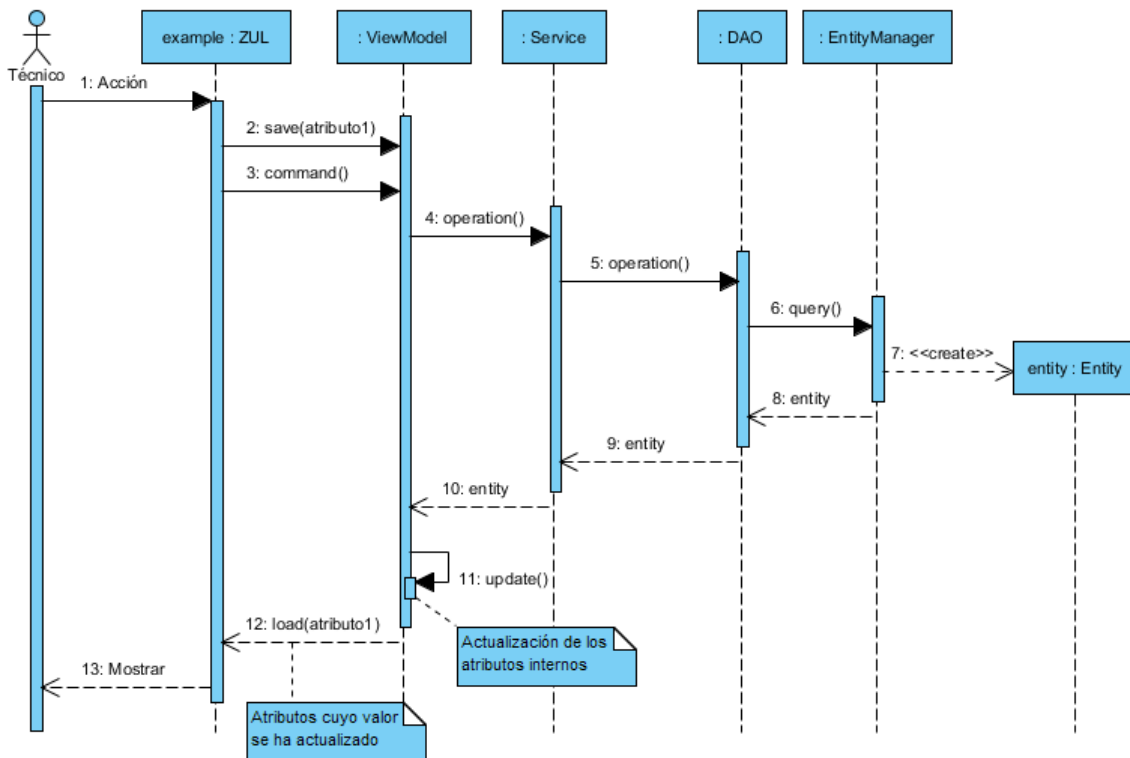


Figura 6. Diagrama xenérico coas chamadas entre clases máis comúns na aplicación.

De este xeito, o Técnico realiza unha acción na interface de usuario xerada polo arquivo ZUL, o cal executa eventos xestionados polo ViewModel. Neste exemplo, o ZUL actualiza o valor dun atributo do ViewModel e posteriormente executa un comando do mesmo. Dito comando chama a unha das operacións do Servizo, que a delega no seu correspondente DAO. Facendo uso do EntityManager, este pode consultar información da base de datos e construír un obxecto de entidade que a represente. Con esta entidade, o ViewModel actualiza un atributo interno en función dalgún criterio, e posteriormente actualiza aqueles compoñentes da interfaz ZUL que dependan del, cuxos cambios son xa visibles para o Técnico.

Hai un detalle a puntualizar, que se trata da existencia do Binder entre o ZUL e o ViewModel. Este é o responsable de realizar toda a parte lóxica do ZUL e de interactuar co ViewModel e é, polo tanto, o que realmente executa os métodos `save()` e `load()`. Dito comportamento non se reflicte no diagrama nin no funcionamento anteriormente expostos, pois é un aspecto da implementación interna de ZK que non aporta valor á comprensión do funcionamento da arquitectura da aplicación, que é a finalidade deste apartado.

10.2.1 Vista estática

A continuación se presenta unha vista estática do Asistente de Laboratorio para MALDI, mediante diagramas de clases e a descrición de cada unha.

10.2.1.1 Diagramas de clases

O primeiro diagrama, que se amosa na Figura 7, representa as relacións que se establecen entre as clases da capa Entities, da capa DAO, e da capa Service, que constitúen as capas máis internas da aplicación.

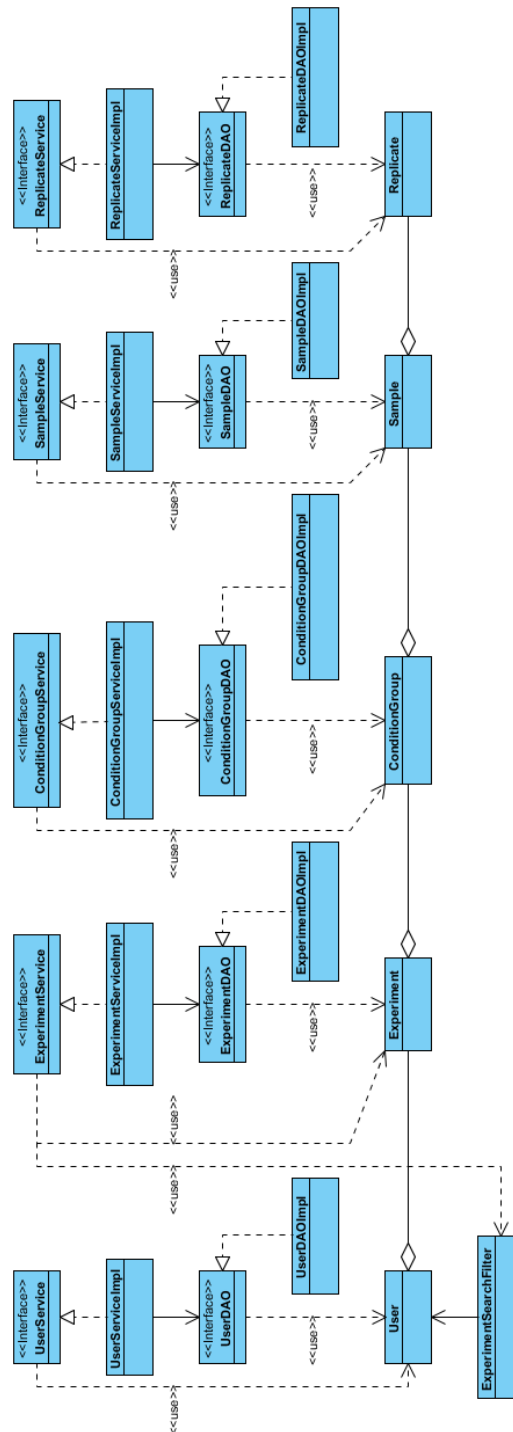


Figura 7. Diagrama de classes das camadas Entities - DAO - Service.

O seguinte abrangue as clases máis pretas ó usuario, pois mostra as relacións entre as clases ViewModel e os Servizos, tal e como aparece na Figura 8.

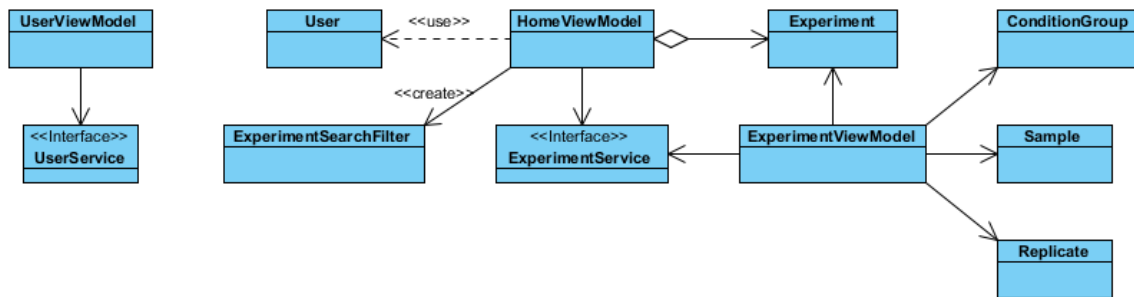


Figura 8. Diagrama de clases das capas Service - ViewModel.

Por último, na Figura 9 inclúese un diagrama adicional cuxo interese non radica na capa na que se atopa, senón que se achega pola interesante arquitectura do paquete IO.

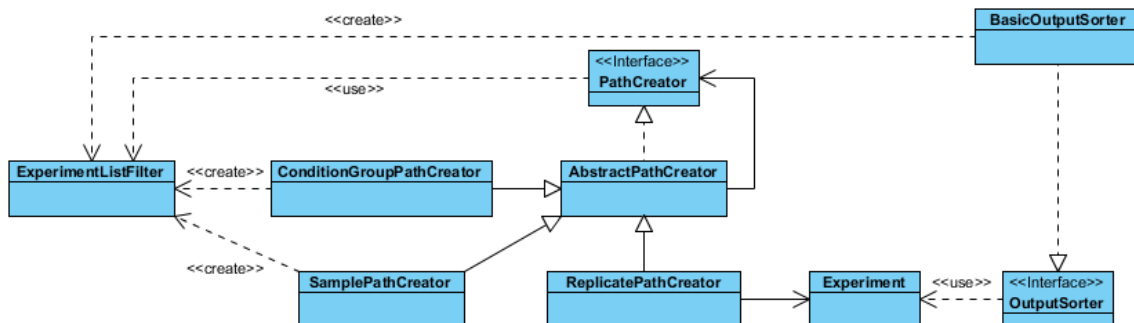
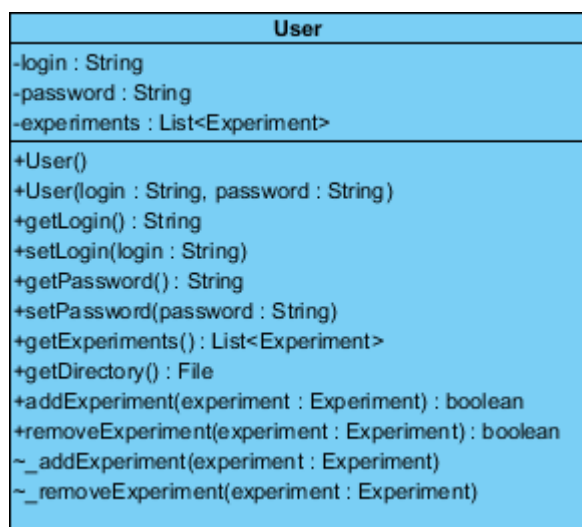


Figura 9. Diagrama de clases do paquete IO para a ordenación de arquivos e creación de directorios.

10.2.1.2 Descrición das clases



A clase User representa ao usuario do sistema, por iso almacena o seu login, password, e experimentos, e permite a xestión dos mesmos.

Experiment
-id : Integer -name : String -description : String -startDate : Date -endDate : Date -numRows : int -numCols : int -rowNameType : CellNameType -colNameType : CellNameType -user : User -conditions : List<ConditionGroup>
+Experiment() +getId() : Integer +setId(id : Integer) +getName() : String +setName(name : String) +getDescription() : String +setDescription(description : String) +getStartDate() : Date +setStartDate(startDate : Date) +getEndDate() : Date +setEndDate(endDate : Date) +getNumRows() : int +setNumRows(numRows : int) +getNumCols() : int +setNumCols(numCols : int) +getRowNameType() : CellNameType +setRowNameType(rowNameType : CellNameType) +getColNameType() : CellNameType +setColNameType(colNameType : CellNameType) +getUser() : User +countReplicates() : int +setUser(user : User) +getConditions() : List<ConditionGroup> +getReplicates(plateId : int) : List<Replicate> +getReplicates() : List<Replicate> +getSamples() : List<Sample> +getFile() : File +addCondition(condition : ConditionGroup) +removeCondition(condition : ConditionGroup) : boolean ~_addCondition(condition : ConditionGroup) ~_removeCondition(condition : ConditionGroup) +isOnPlate() : boolean +isMetadataComplete() : boolean +eachSampleHasReplicates() : boolean +placeReplicateAt(replicate : Replicate, plateId : int, row : int, col : int) +getReplicateAt(plateId : int, row : int, col : int) : Replicate -validatePlateLocation(plateId : int, row : int, col : int)

A clase Experiment contén tódolos datos necesarios para a creación dun experimento MALDI, o cal abrangue datos como a descrición e a data, así como a súa lista de condicións, mostras, e réplicas. Ofrece, ademais, toda clase de métodos para xestionar ditos atributos.

ConditionGroup
-id : Integer -name : String -experiment : Experiment -samples : List<Sample> -color : String
+ConditionGroup() +getId() : Integer +setId(id : Integer) +getName() : String +setName(name : String) +getExperiment() : Experiment +setExperiment(experiment : Experiment) +getSamples() : List<Samples> +getReplicates() : List<Replicate> +getColor() : String +setColor(color : String) +countReplicates() : int +addSample(sample : Sample) : boolean +removeSample(sample : Sample) : boolean ~_addSample(sample : Sample) ~_removeSample(sample : Sample) +isOnPlate() : boolean

A clase ConditionGroup representa unha condición dun experimento MALDI, da cal é interesante coñecer a lista de mostras e a cor que terá no padrón da placa. A súa lóxica permite modificar o experimento ó que pertence esta condición, así como as entidades que dela dependen.

Sample
-id : Integer -name : String -condition : ConditionGroup -replicates : List<Replicates>
+Sample() +getId() : Integer +setId(id : Integer) +getName() : String +setName(name : String) +getCondition() : ConditionGroup +setCondition(condition : ConditionGroup) +getReplicates() : List<Replicate> +countReplicates() : int +addReplicate(replicate : Replicate) : boolean +removeReplicate(replicate : Replicate) : boolean ~_addReplicate(replicate : Replicate) ~_removeReplicate(replicate : Replicate) +isOnPlate() : boolean +getColor() : String

A clase Sample correspóndese na realidade cunha mostra dun experimento MALDI. Trátase dunha entidade que pertence a unha condición e que posúe unha lista de réplicas, as cales poden ser modificadas a través dos métodos que se proveen.

Replicate
-id : Integer -name : String -plateId : Integer -col : Integer -row : Integer -sample : Sample
+Replicate() +getId() : Integer +setId(id : Integer) +getName() : String +setName(name : String) +getPlateId() : Integer +getCol() : Integer +getRow() : Integer +getSample() : Sample +setSample(sample : Sample) +isOnPlate() : boolean +isOnPlate(plateId : int) : boolean +isPlacedAt(plateId : int, row : int, col : int) : boolean +removeFromPlate() ~placeAtPlate(plateId : int, row : int, col : int) +getColor() : String

A clase Replicate representa a unha das réplicas dunha determinada mostra que reúne a información necesaria para ser colocada por un técnico nunha determinada placa, nun determinado número de fila e columna.

<<Interface>> UserDAO
+add(user : User) : User +get(login : String) : User +reload(user : User) : User

A interface UserDAO define o comportamento dun DAO para engadir ou recuperar un usuario da base de datos. O método reload() recupera os datos dun usuario tal e como se atopan na base de datos.

UserDAOImpl
-em : EntityManager +add(user : User) : User +get(login : String) : User +reload(user : User) : User

A clase UserDAOImpl implementa o contrato da interface UserDAO mediante o uso dun EntityManager.

<<Interface>> ExperimentDAO
+add(experiment : Experiment) : Experiment +get(experimentId : Integer) : Experiment +reload(experiment : Experiment) : Experiment +update(experiment : Experiment) : Experiment +delete(experiment : Experiment) +listFilter(filter : ExperimentSearchFilter) : List<Experiment>

A interface ExperimentDAO define o comportamento dun DAO para xestionar un experimento da base de datos. O método reload() recupera os datos dun experimento tal e como se atopan na base de datos.

ExperimentDAOImpl
-em : EntityManager
+add(experiment : Experiment) : Experiment +get(id : Integer) : Experiment +reload(experiment : Experiment) : Experiment +update(experiment : Experiment) : Experiment +delete(experiment : Experiment) +listFilter(filter : ExperimentSearchFilter) : List<Experiment>

A clase ExperimentDAOImpl implementa o contrato da interface ExperimentDAO mediante o uso dun EntityManager.

<<Interface>> ConditionGroupDAO
+add(condition : ConditionGroup) : ConditionGroup +get(id : Integer) : ConditionGroup +reload(condition : ConditionGroup) : ConditionGroup +update(condition : ConditionGroup) : ConditionGroup +delete(condition : ConditionGroup)

A interface ConditionGroupDAO define o comportamento dun DAO para xestionar unha condición da base de datos. O método reload() recupera os datos dunha condición tal e como se atopan na base de datos.

ConditionGroupDAOImpl
-em : EntityManager
+add(condition : ConditionGroup) : ConditionGroup +get(id : Integer) : ConditionGroup +update(condition : ConditionGroup) : ConditionGroup +delete(condition : ConditionGroup) +reload(condition : ConditionGroup) : ConditionGroup

A clase ConditionGroupDAOImpl implementa o contrato da interface ConditionGroupDAO mediante o uso dun EntityManager.

<<Interface>> SampleDAO
+add(sample : Sample) : Sample +get(id : Integer) : Sample +reload(sample : Sample) : Sample +update(sample : Sample) : Sample +delete(sample : Sample)

A interface SampleDAO define o comportamento dun DAO para xestionar unha mostra da base de datos. O método reload() recupera os datos dunha mostra tal e como se atopan na base de datos.

SampleDAOImpl
-em : EntityManager
+add(sample : Sample) : Sample +get(id : Integer) : Sample +update(sample : Sample) : Sample +delete(sample : Sample) +reload(sample : Sample) : Sample

A clase SampleDAOImpl implementa o contrato da interface SampleDAO mediante o uso dun EntityManager.

<<Interface>> ReplicateDAO
+add(replicate : Replicate) : Replicate +get(id : Integer) : Replicate +reload(replicate : Replicate) : Replicate +update(replicate : Replicate) : Replicate +delete(replicate : Replicate)

A interface ReplicateDAO define o comportamento dun DAO para xestionar unha réplica da base de datos. O método reload() recupera os datos dunha réplica tal e como se atopan na base de datos.

ReplicateDAOImpl
-em : EntityManager
+add(replicate : Replicate) : Replicate
+get(id : Integer) : Replicate
+update(replicate : Replicate) : Replicate
+delete(replicate : Replicate)
+reload(replicate : Replicate) : Replicate

A classe ReplicateDAOImpl implementa o contrato da interface ReplicateDAO mediante o uso dun EntityManager.

<<Interface>> UserService
+add(user : User) : User
+get(login : String) : User

A interface UserService define o comportamento dun Service para engadir ou recuperar un usuario da base de datos.

UserServiceImpl
-dao : UserDAO
+add(user : User) : User
+get(login : String) : User

A clase UserServiceImpl implementa o contrato da interface UserService mediante o uso dun DAO.

<<Interface>> ExperimentService
+add(experiment : Experiment) : Experiment
+get(id : Integer) : Experiment
+update(experiment : Experiment) : Experiment
+delete(experiment : Experiment)
+listFilter(filter : ExperimentSearchFilter) : List<Experiment>
+reload(reload : Experiment) : Experiment

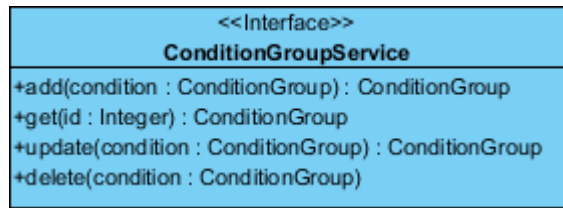
A interface ExperimentService define o comportamento dun Service para xestionar un experimento da base de datos. O método reload() recupera os datos dun experimento tal e como se atopan na base de datos.

ExperimentServiceImpl
-dao : ExperimentDAO
+add(experiment : Experiment) : Experiment
+get(id : Integer) : Experiment
+update(experiment : Experiment) : Experiment
+delete(experiment : Experiment)
+listFilter(filter : ExperimentSearchFilter) : List<Experiment>
+reload(experiment : Experiment) : Experiment

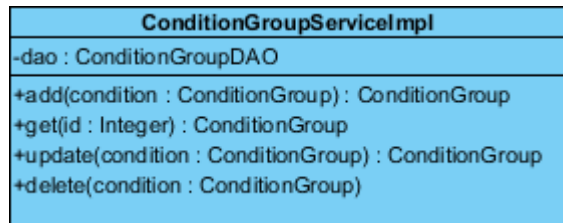
A clase ExperimentServiceImpl implementa o contrato da interface ExperimentService mediante o uso dun DAO.

ExperimentSearchFilter
~user : User
~name : String
+ExperimentSearchFilter(user : User)
+getUser() : User
+getName() : String
+setName(name : String)

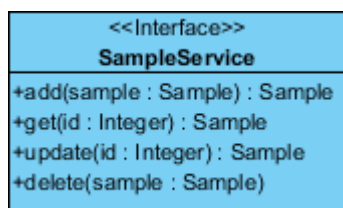
A clase ExperimentSearchFilter serve para filtrar os experimentos que aparecen na lista de HomeViewModel mediante a barra de busca.



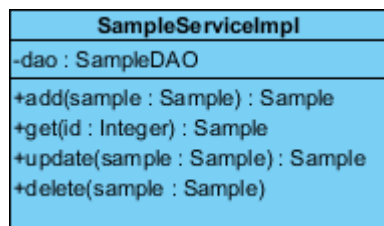
A interface ConditionGroupService define o comportamento dun Service para xestionar unha condición da base de datos.



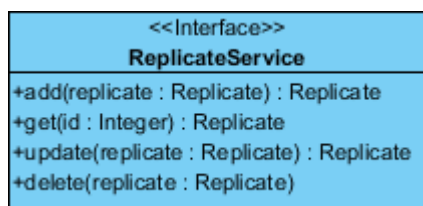
A clase ConditionGroupServiceImpl implementa o contrato da interface ConditionGroupService mediante o uso dun DAO.



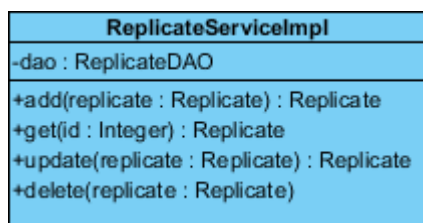
A interface SampleService define o comportamento dun Service para xestionar unha mostra da base de datos.



A clase SampleServiceImpl implementa o contrato da interface SampleService mediante o uso dun DAO.



A interface ReplicateService define o comportamento dun Service para xestionar unha réplica da base de datos.



A clase ReplicateServiceImpl implementa o contrato da interface ReplicateService mediante o uso dun DAO.

UserViewModel
-userService : UserService -signInUsername : String -signInPassword : String -signUpUsername : String -signUpPassword : String
+getSignInUsername() : String +setSignInUsername(signInUsername : String) +getSignInPassword() : String +setSignInPassword(signInPassword : String) +getSignUpUsername() : String +setSignUpUsername(signUpUsername : String) +getSignUpPassword() : String +setSignUpPassword(signUpPassword : String) +signIn() +signUp()

A clase UserViewModel é unha abstracción das vistas de login e rexistro, de xeito que almacena os seus datos e executa as súas lóxicas.

HomeViewModel
-experimentService : ExperimentService -experiments : List<Experiment> -filter : ExperimentSearchFilter
-getUser() : User +getExperimentModel() : List<Experiment> +getFilter() : ExperimentSearchFilter +edit(experiment : Experiment) +delete(experiment : Experiment) +add() +changeFilter()

A clase HomeViewModel é unha abstracción da vista home, do listado de experimentos; permite editar ou eliminar experimentos existentes, así como crear novos.

ExperimentViewModel
-experimentService : ExperimentService -outputSorter : OutputSorter -experiment : Experiment -selectedCondition : ConditionGroup -selectedSample : Sample -selectedReplicate : Replicate -uploadStatus : String -conditionChecked : boolean -sampleChecked : boolean -experimentFile : File -globalCommandListener : EventListener<Event>
+init() +selectedReplicateChanged(replicate : Replicate) +isMetadataCompleted() : boolean +isOnPlate() : boolean +getModel() : ExperimentListModel +getExperiment() : Experiment +setExperiment(experiment : Experiment) +getCellNameTypes() : CellNameType[] +getPlateIds() : List<Integer> +getPlateNames() : List<String> +getSelectedCondition() : ConditionGroup +setSelectedCondition(selectedCondition : ConditionGroup) +getSelectedSample() : Sample +setSelectedSample(selectedSample : Sample) +getUploadStatus() : String +setUploadStatus(uploadStatus : String) +isConditionChecked() : boolean +setConditionChecked(conditionChecked : boolean) +isSampleChecked() : boolean +setSampleChecked(sampleChecked : boolean) +getExperimentFile() : File +save() +reset() +exit() +cancel() +addCondition() +removeCondition(condition : ConditionGroup) +addSample(condition : ConditionGroup) +removeSample(condition : ConditionGroup, sample : Sample) +addReplicate(sample : Sample) +removeReplicate(sample : Sample, replicate : Replicate) +uploadFile(event : UploadEvent) +downloadFile() +isDirectoryStructureOk() : boolean -getPathRegex() : String -createTmpDirectory() : File

A clase ExperimentViewModel é unha abstracción das vistas de edición dos datos do experimento, da vista de deseño dos padróns do experimento, e da vista do ordenado de arquivos do experimento. Os datos da primeira vista almacénanse na propia entidade do experimento, Experiment. Por outra banda, na segunda vista, os atributos selectedCondition, selectedSample, e selectedReplicate, permiten coñecer a entidade seleccionada polo técnico, para despois pintala no padrón. Por último, ofrece o ordenado de arquivos e o seu comprimido/descomprimido.

ExperimentListModel
<<static>> -serialVersionUID : long -experiment : Experiment -condition : List<ConditionGroup>
+ExperimentListModel(experiment : Experiment) +getExperiment() : Experiment +getElementAt(index : int) : ConditionListModel +getSize() : int +update(o : Observable, arg : Object)

A clase ExperimentListModel serve para mostrar unha lista de experimentos nun compoñente Listbox, e facilita a súa actualización en tempo real.

ConditionListModel
<<static>> -serialVersionUID : long -condition : ConditionGroup
+ConditionListModel(condition : ConditionGroup) +getCondition() : ConditionGroup +getElementAt(index : int) : SampleListModel +getSize() : int +update(o : Observable, arg : Object)

A clase ConditionListModel sirve para mostrar unha lista de condicións nun compoñente Listbox, e facilita a súa actualización en tempo real.

SampleListModel
<<static>> -serialVersionUID : long -sample : Sample
+SampleListModel(sample : Sample) +getSample() : Sample +getElementAt(index : int) : Replicate +getSize() : int +update(o : Observable, arg : Object) -removeFromObserved()

A clase SampleListModel sirve para mostrar unha lista de mostras nun compoñente Listbox, e facilita a súa actualización en tempo real.

<<enumeration>> UploadStatusType
<<Constant>> -STOPPED <<Constant>> -IN_PROGRESS <<Constant>> -FINISHED <<Constant>> -ERROR -message : String
-UploadStatusType(message : String) +toString() : String

O enumerado UploadStatusType contén os diferentes estados que pode tomar o proceso de subida do arquivo comprimido a ordenar.

PlateEditorComposer
<<static>> -serialVersionUID +onCellClick(event : CellMouseEvent) <<static>> +isCtrlPressed(event : CellMouseEvent) : boolean

A clase PlateEditorComposer escoita os eventos que se producen no PlateEditor, co fin de pintar/borrar as celas que correspondan.

<<Interface>> OutputSorter
+checkPath(pathRegex : String) : boolean +sort(experiment : Experiment, datasetDirectory : File, pathRegex : String, outputDirectory : File)

A interface OutputSorter define o comportamento para ordenar un directorio de arquivos en base a un criterio aportado en forma de expresión regular.

BasicOutputSorter
<<static>> -directories : List<String>
+sort(experiment : Experiment, datasetDirectory : File, pathRegex : String, outputDirectory : File)
-createPathCreatorFromRegex(pathRegex : String, experiment : Experiment, outputDirectory : File) : PathCreator
-createFilterFromExperiment(experiment : Experiment) : ExperimentListFilter
+checkPath(pathRegex : String) : boolean
+generateMatches() : List<String>

A clase BasicOutputSorter implementa a interface OutputSorter, e permite crear PathCreator a partir dunha expresión regular.

<<Interface>> PathCreator
+create(baseDirectory : File, filter : ExperimentListFilter)
+setChild(child : PathCreator)

A interface PathCreator define o comportamento para crear directorios de acordo cun determinado filtro.

AbstractPathCreator
#child : PathCreator
+AbstractPathCreator()
+AbstractPathCreator(child : PathCreator)
+setChild(child : PathCreator)

A clase AbstractPathCreator xeneraliza unhas clases capaces de asignarse un PathCreator fillo.

ConditionGroupPathCreator
+ConditionGroupPathCreator()
+ConditionGroupPathCreator(child : PathCreator)
+create(baseDirectory : File, filter : ExperimentListFilter)
<<static>> +createFilter() : ExperimentListFilter

A clase ConditionGroupPathCreator crea os directorios intermedios correspondentes coas condicións do experimento.

SamplePathCreator
+SamplePathCreator()
+SamplePathCreator(child : PathCreator)
+create(baseDirectory : File, filter : ExperimentListFilter)
<<static>> +createFilter(sample : Sample) : ExperimentListFilter

A clase SamplePathCreator crea os directorios intermedios correspondentes coas mostras do experimento.

ReplicatePathCreator
-experiment = Experiment
-datasetDirectory : File
+ReplicatePathCreator(experiment : Experiment, filter : ExperimentListFilter)
+create(baseDirectory : File, filter : ExperimentListFilter)

A clase ReplicatePathCreator renomea os arquivos das réplicas segundo o especificado no experimento, e as coloca no directorio das súas correspondentes mostras se é que existen.

<<Interface>> ExperimentListFilter
+listConditions() : List<ConditionGroup>
+listSamples() : List<Sample>
+listReplicates() : List<Replicate>

A interface ExperimentListFilter define como listar condicións, mostras, ou réplicas, co fin de que cada PathCreator decida que listar en cada caso.

PlateEditor
<pre> <<static>> -serialVersionUID : long <<static>> -DEFAULT_SELECTED_CELL_COLOR : String -plateId : Integer -experiment : Experiment -selectedCondition : ConditionGroup -selectedSample : Sample -selectedReplicate : Replicate +getExperiment() : Experiment +setExperiment(experiment : Experiment) +getPlateId() : Integer +setPlateId(plateId : Integer) +getSelectedCondition() : ConditionGroup +setSelectedCondition(selectedCondition : ConditionGroup) +getSelectedSample() : Sample +setSelectedSample(selectedSample : Sample) +getSelectedReplicate() : Replicate +setSelectedReplicate(selectedReplicate : Replicate) -updateSelection() -updateLabels() +updateHighlightedCells() -getSelectedReplicates() : List<Replicate> -isSelected(replicate : Replicate) : boolean -cleanPlate() -highlightCells(tRow : int, lCol : int, bRow : int, rCol : int, color : String, selected : boolean) <<static>> -changeRangeBackground(range : Range, color : String, selected : boolean) -createTitles(type : CellNameType, num : int) : String </pre>

A clase PlateEditor é un compoñente propio que foi adaptado da Spreadsheet de ZK. Deste xeito se ten un compoñente totalmente personalizado para a colocación das réplicas.

<<final>> ColorUtils
<pre> <<static>> -THIRTY_COLORS : Color[] -ColorUtils() <<static>> +getThirtyColor(offset : int) : Color <<static>> +getThirtyColorHex(offset : int) : String <<static>> +colorToHexString(color : Color) : String <<static>> +hexStringColor(hexString : String) : Color <<static>> +invert(color : Color) : Color +invert(color : String) : String <<static>> +getBestContrast(color : String) : String <<static>> +getBestContrast(color : String) : Color </pre>

A clase ColorUtils é unha clase de utilidade que ten como finalidade a xeración de cores para representar as condicións do experimento nos padróns.

ConditionToSClassConverter
<pre> +coerceToBean(beanProp : String, component : Component, ctx : BindContext) : ConditionGroup +coerceToUi(compAttr : ConditionGroup, component : Component, ctx : BindContext) : String </pre>

A clase ConditionToSClassConverter convirte un obxecto ConditionGroup nun estilo CSS baseado na cor que ten asignada.

Esta clase está pensada para ser combinada co *binding* cando se usa o patrón MVVM.

SecurityInitiator
<pre> <<static>> -IGNORE_PAGES : Set<String> +doinit(page : Page, args : Map<String, Object>) </pre>

A clase SecurityInitiator comproba que non se accede a ningunha páxina na que se precisa estar autenticado sen estalo.

<<enumeration>> CellNameType
<<Constant>> -NUMERICAL <<Constant>> -LOWERCASE <<Constant>> -UPPERCASE
+indexToLabel(index : int) : String +createLabels(num : int) : String[] -inflate(str : StringBuilder) : StringBuilder

O enumerado CellNameType define os diferentes tipos de cela que pode haber, así como realizar conversións entre eles.

<<final>> Configuration
<<static>> -instance : Configuration -initialContext : InitialContext
<<static>> +getInstance() : Configuration -Configuration() -createInitialContext() -getConfigParams(param : String) : T +getDataDirectory() : File +getUsersDirectory() : File +getTmpDirectory()

A clase Configuration ten como finalidade obter valores de configuración, tales como os directorios por defecto dos datos, dos usuarios, ou do directorio temporal.

DataPackager
<<static>> +unpackData(media : Media, outputDirectory : File) <<static>> -unzipEntry(zipfile : ZipFile, entry : ZipEntry, outputDirectory : File) <<static>> +zipData(inputDir : File, zipFile : File) : File <<static>> -zipEntry(zipStream : ArchiveOutputStream, files : File[], path : String) <<static>> +createDirectory(dir : File)

A clase DataPackager ofrece métodos para desempaqetar en ZIP e TAR, e empaquetar en ZIP.

10.2.2 Vista dinámica

A continuación preséntase unha vista dinámica do Asistente de Laboratorio para MALDI mediante diagramas de secuencia. Estes diagramas representan o fluxo de traballo normal para cada unha das historias de usuario realizadas polo rol de Técnico.

10.2.2.1 HU01 – Conta pessoal

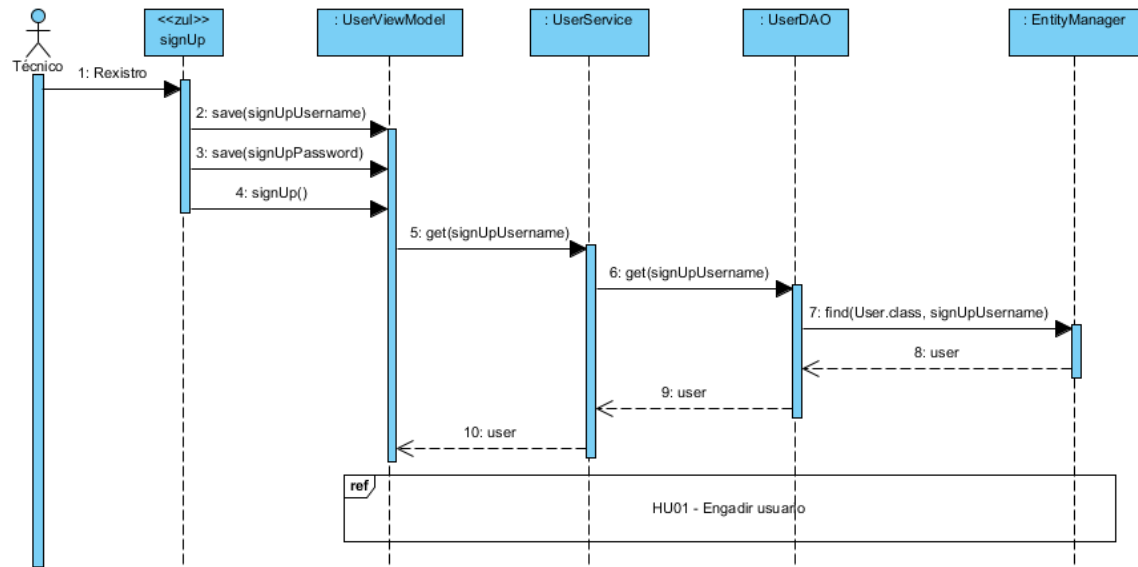


Figura 10. Diagrama de secuencia para a historia de usuario "Conta pessoal".

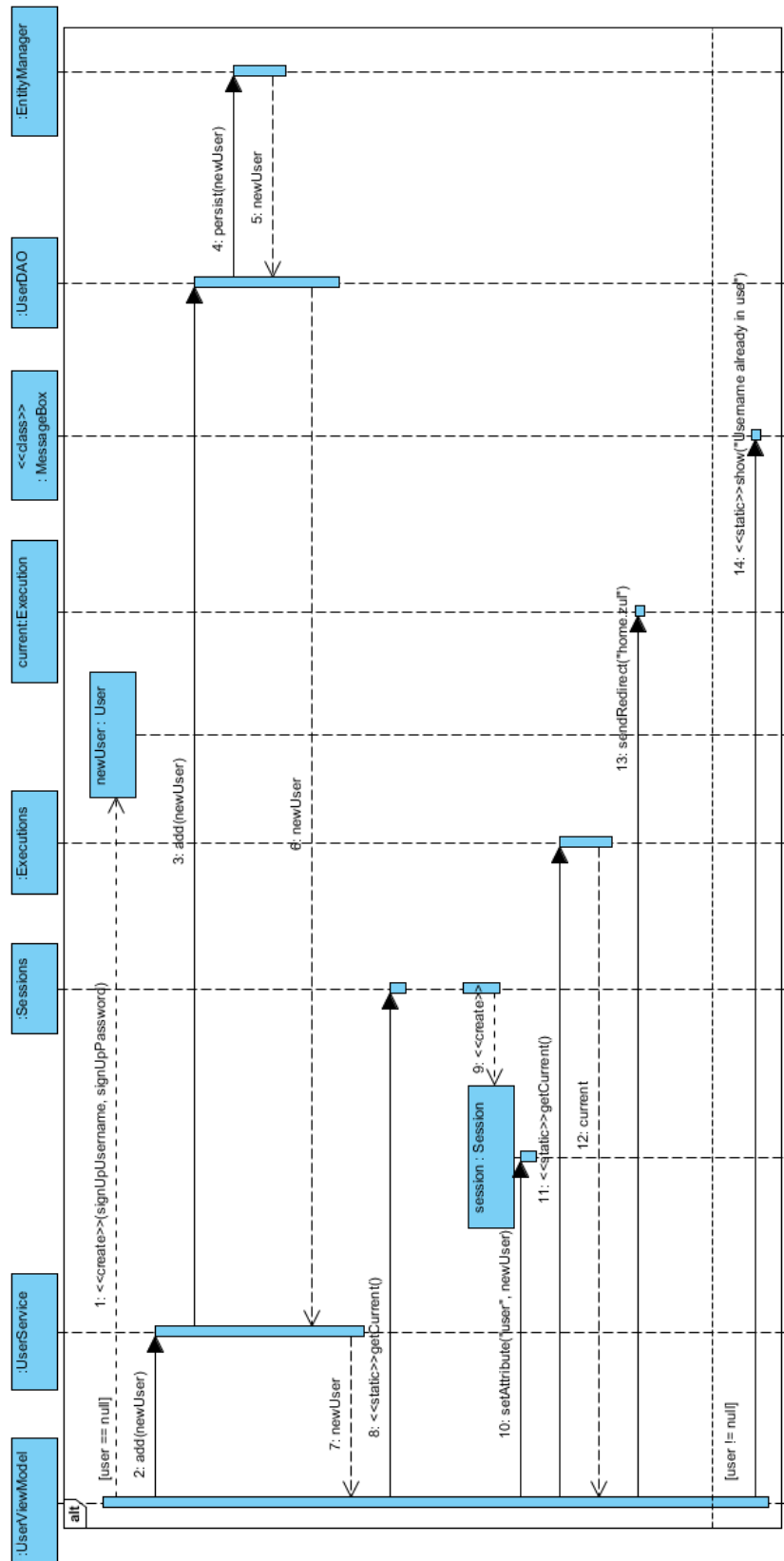


Figura 11. Ref. Hu01 - Engadir usuario

10.2.2.2 HU02 – Listar experimentos

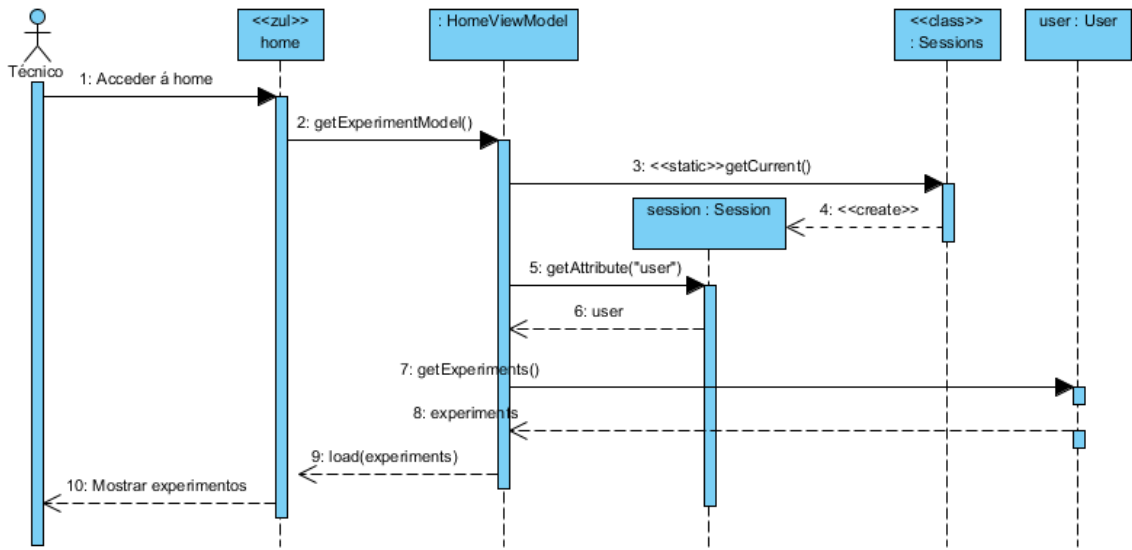


Figura 12. Diagrama de secuencia para a historia de usuario "Listar experimentos".

10.2.2.3 Hu03 – Crear Experimento Básico

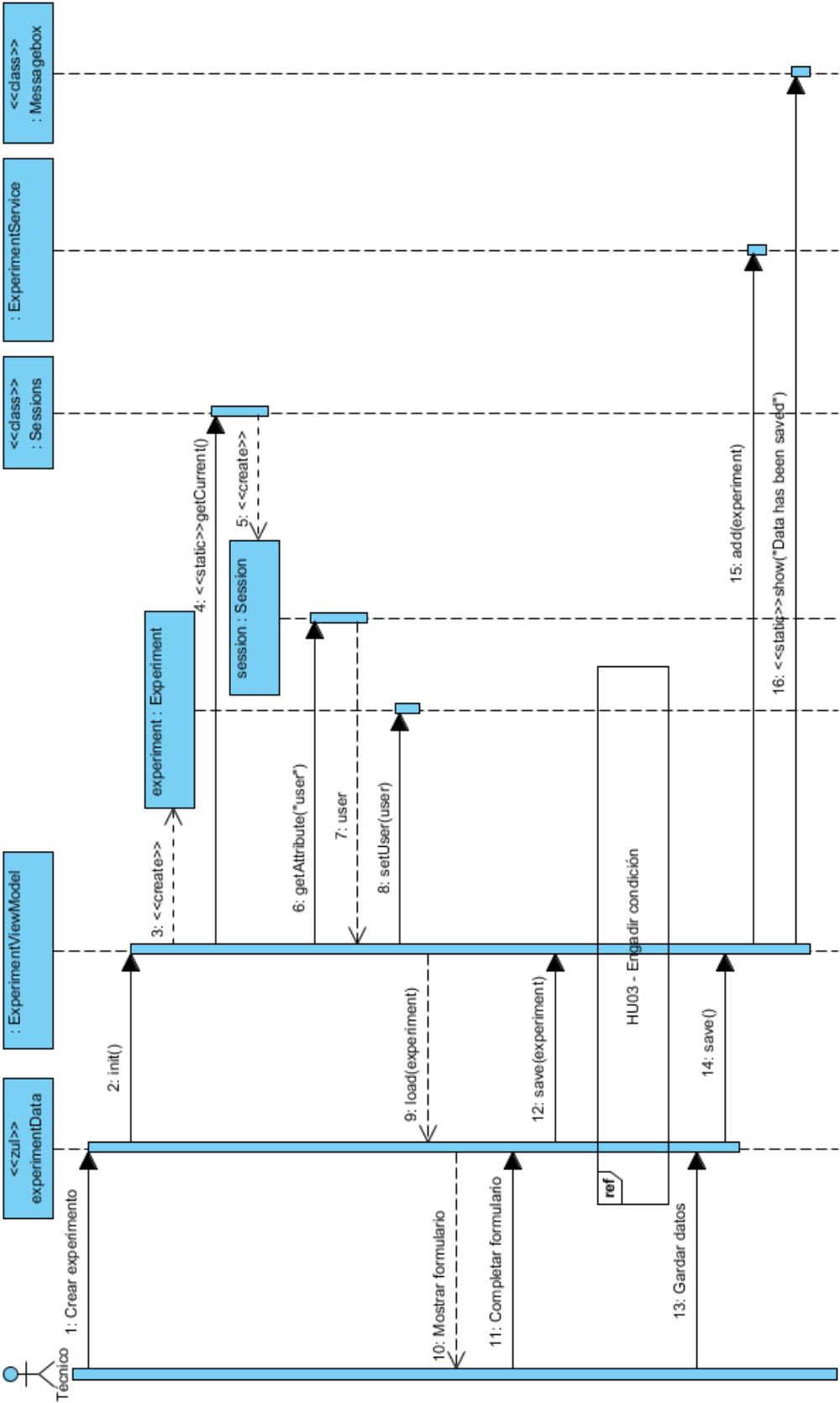


Figura 13. Diagrama de secuencia da historia de usuario "Crear experimento básico".

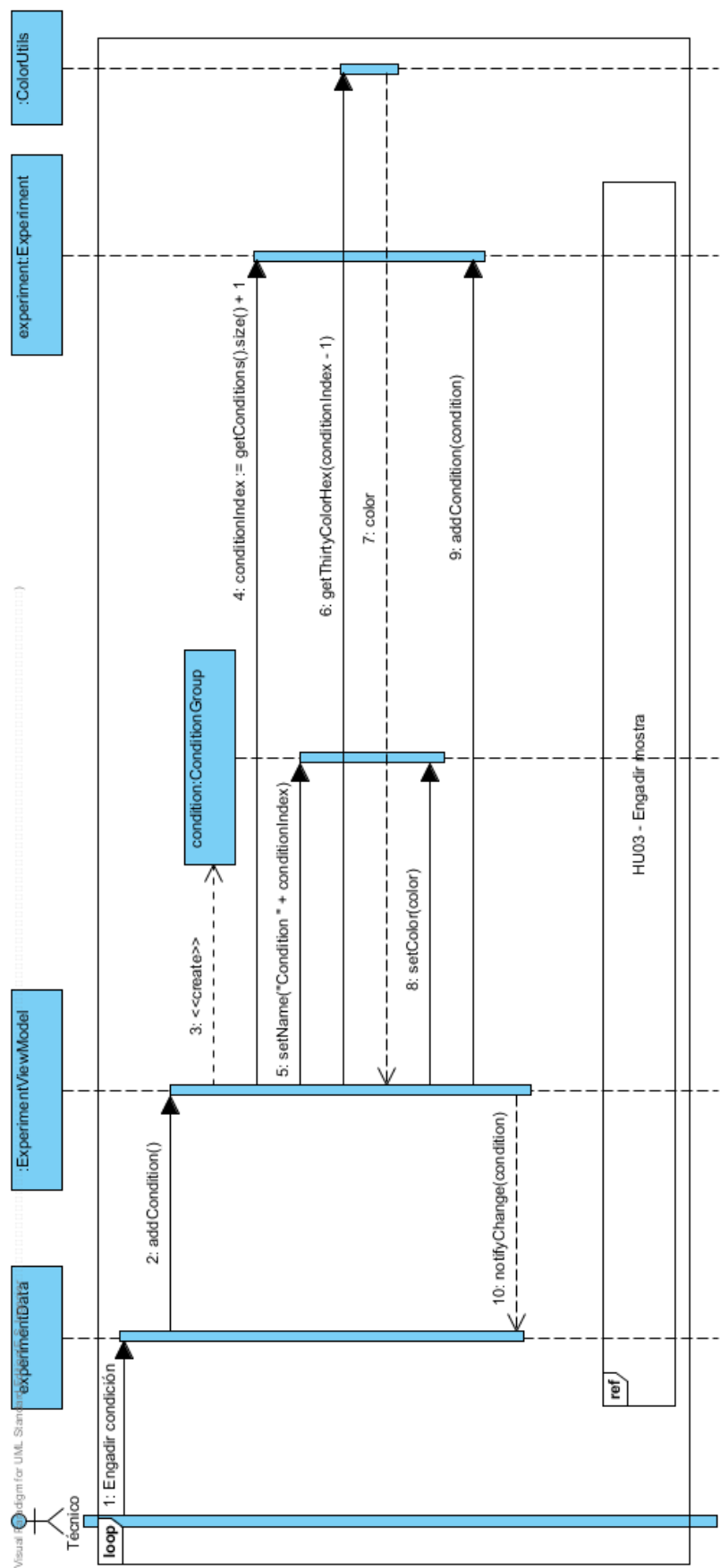


Figura 14. Ref. Hu03 - Engadir condición.

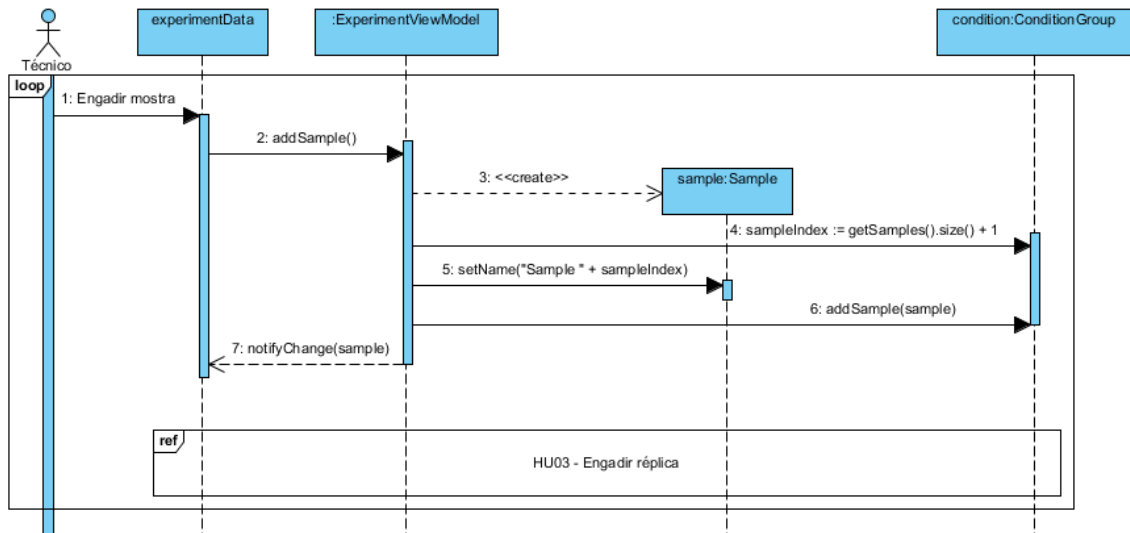


Figura 15. Ref. Hu03 - Engadir mostra.

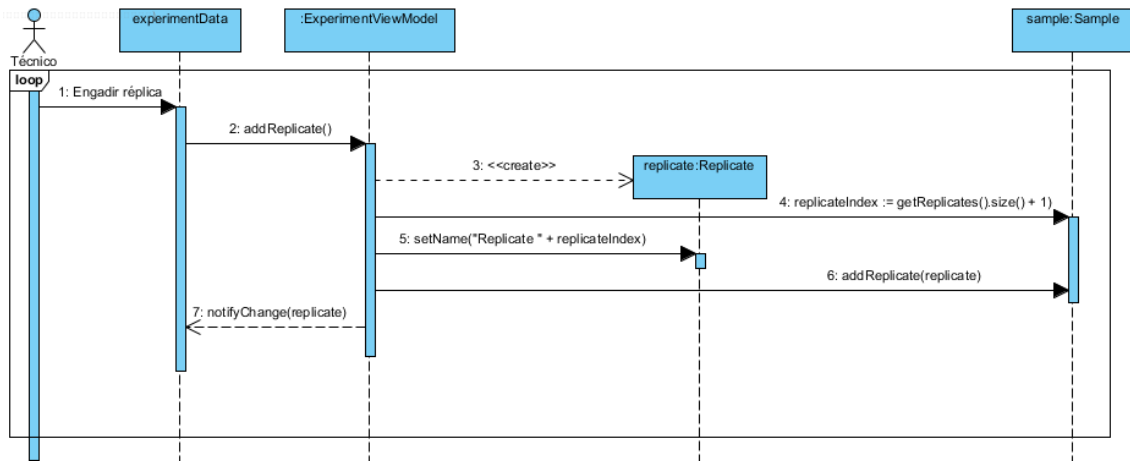


Figura 16. Ref. Hu03 - Engadir réplica.

10.2.2.4 HU06 – Diseñar padróns manualmente básico

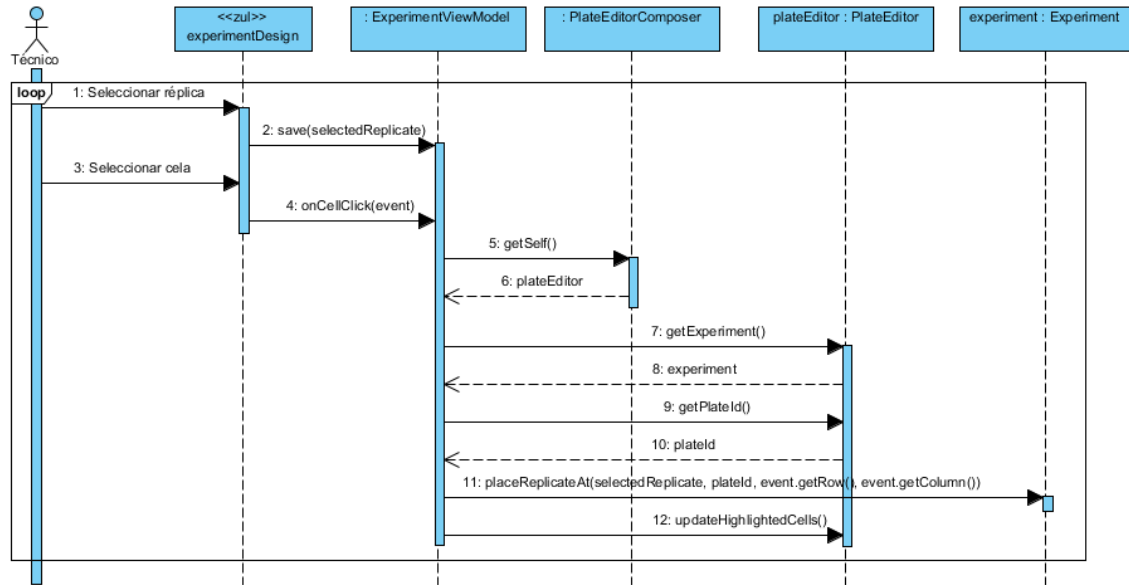


Figura 17. Diagrama de secuencia da historia de usuario "Diseñar padróns manualmente básico".

10.2.2.5 HU07 – Subir resultados básico

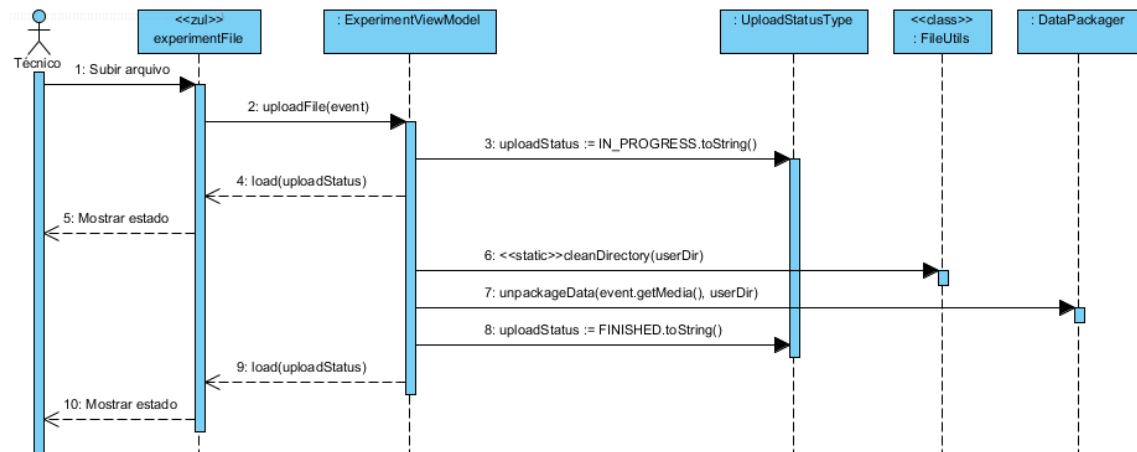


Figura 18. Diagrama de secuencia da historia de usuario "Subir resultados básico".

10.2.2.6 HU08 - Ordenar resultados básico

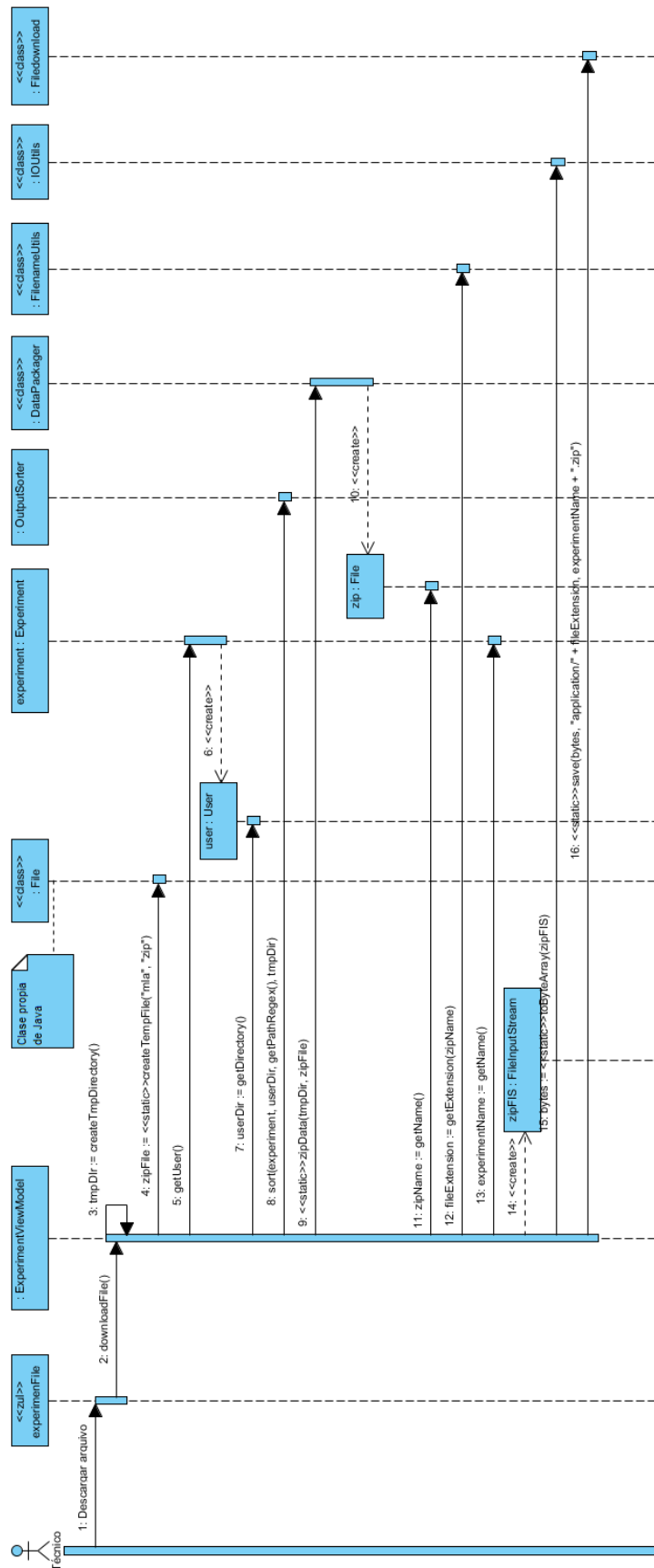


Figura 19. Diagrama de secuencia da historia de usuario "Ordenar resultados básico".

10.2.2.7 HU09 - Eliminar experimento

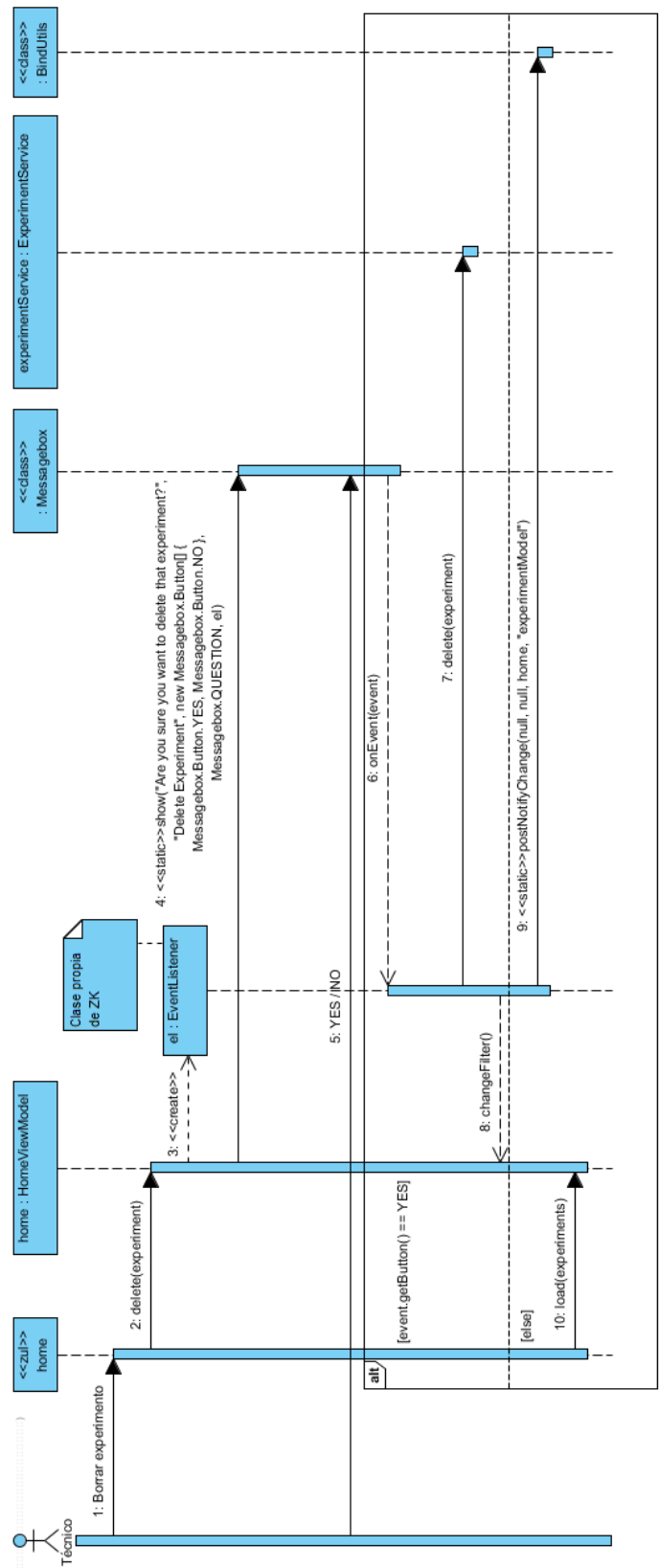


Figura 20. Diagrama de secuencia da historia de usuario "Eliminar experimento".

10.2.3 Probas

10.2.3.1 Probas de aceptación

As probas de aceptación de cada historia de usuario recólleas cada historia no apartado “Criterios de aceptación”, as cales se poden consultar na sección 9.2 Historias de usuario.

10.2.3.2 Probas de unidade

A continuación preséntanse os casos de proba que se implementaron no Asistente de Laboratorio para MALDI co fin de probar unitariamente os métodos de determinadas clases. Dado que estas probas de unidade se fixeron con JUnit, cada caso de proba correspóndese con unha clase Java e cada proba con un método. En tódolos casos se utiliza DbUnit para garantir que a base de datos non queda nun estado non desexado entre as execucións dos métodos, polo cal non se definen accións a realizar antes nin despois de cada proba. Todas estas probas poden atoparse como parte do código fonte da aplicación.

Caso de proba: UserTest.java	
Descrición	Executa as probas necesarias para que a lóxica da entidade User teña un comportamento esperado.
Probas	<ul style="list-style-type: none"> · Compróbase que se lanza unha excepción se se engade un experimento nulo. · Compróbase que se pode engadir un experimento. · Compróbase que se lanza unha excepción se se elimina un experimento nulo. · Compróbase que se pode eliminar un experimento.

Táboa 11. Caso de proba: UserTest.java.

Caso de proba: ExperimentTest.java	
Descrición	Executa as probas necesarias para que a lóxica da entidade Experiment teña un comportamento esperado.
Probas	<ul style="list-style-type: none"> · Compróbase que está na placa. · Compróbase que os metadatos están completos. · Compróbase que se poden contar as réplicas. · Compróbase que se poden obter as réplicas. · Compróbase que se poden obter as réplicas dada unha placa. · Compróbase que se lanza unha excepción se se engade unha condición nula. · Compróbase que se pode engadir unha condición. · Compróbase que se lanza unha excepción se se elimina un experimento nulo. · Compróbase que se pode eliminar unha condición.

Táboa 12. Caso de proba: ExperimentTest.java.

Caso de proba: ConditionGroupTest.java	
Descrición	Executa as probas necesarias para que a lóxica da entidade ConditionGroup teña un comportamento esperado.
Probas	<ul style="list-style-type: none"> · Compróbase que está na placa. · Compróbase que se poden contar as réplicas. · Compróbase que se lanza unha excepción se se engade unha mostra nula. · Compróbase que se pode engadir unha mostra. · Compróbase que se lanza unha excepción se se elimina unha mostra nula. · Compróbase que se pode eliminar unha mostra.

Táboa 13. Caso de proba: ConditionGroupTest.java.

Caso de proba: SampleTest.java	
Descrición	Executa as probas necesarias para que a lóxica da entidade Sample teña un comportamento esperado.
Probas	<ul style="list-style-type: none"> · Compróbase que está na placa. · Compróbase que se poden contar as réplicas. · Compróbase que se lanza unha excepción se se engade unha réplica nula. · Compróbase que se pode engadir unha réplica. · Compróbase que se lanza unha excepción se se elimina unha réplica nula. · Compróbase que se pode eliminar unha réplica.

Táboa 14. Caso de proba: SampleTest.java.

Caso de proba: UserDAOTest.java	
Descrición	Executa as probas necesarias para que a lóxica do DAO da entidade User teña un comportamento esperado.
Probas	<ul style="list-style-type: none"> · Compróbase que se pode engadir un usuario. · Compróbase que se pode recuperar un usuario. · Compróbase que se pode recargar un usuario.

Táboa 15. Caso de proba: UserDAOTest.java.

Caso de proba: ExperimentDAOTest.java	
Descrición	Executa as probas necesarias para que a lóxica do DAO da entidade Experiment teña un comportamento esperado.
Probas	<ul style="list-style-type: none"> · Compróbase que se pode engadir un experimento. · Compróbase que se pode recuperar un experimento. · Compróbase que se pode recargar un experimento. · Compróbase que se pode modificar un experimento. · Compróbase que se pode eliminar un experimento. · Compróbase que se poden filtrar resultados dunha lista de experimentos.

Táboa 16. Caso de proba: ExperimentDAOTest.java.

Caso de proba: ConditionGroupDAOTest.java	
Descrición	Executa as probas necesarias para que a lóxica do DAO da entidade ConditionGroup teña un comportamento esperado.
Probas	<ul style="list-style-type: none"> · Compróbase que se pode engadir unha condición. · Compróbase que se pode recuperar unha condición. · Compróbase que se pode recargar unha condición. · Compróbase que se pode modificar unha condición. · Compróbase que se pode eliminar unha condición.

Táboa 17. ConditionGroupDAOTest.java.

Caso de proba: SampleDAOTest.java	
Descrición	Executa as probas necesarias para que a lóxica do DAO da entidade Sample teña un comportamento esperado.
Probas	<ul style="list-style-type: none"> · Compróbase que se pode engadir unha mostra. · Compróbase que se pode recuperar unha mostra. · Compróbase que se pode recargar unha mostra. · Compróbase que se pode modificar unha mostra. · Compróbase que se pode eliminar unha mostra.

Táboa 18. Caso de proba: SampleDAOTest.java.

Caso de proba: ReplicateDAOTest.java	
Descrición	Executa as probas necesarias para que a lóxica do DAO da entidade Replicate teña un comportamento esperado.
Probas	<ul style="list-style-type: none"> · Compróbase que se pode engadir unha réplica. · Compróbase que se pode recuperar unha réplica. · Compróbase que se pode recargar unha réplica. · Compróbase que se pode modificar unha réplica. · Compróbase que se pode eliminar unha réplica.

Táboa 19. Caso de proba: ReplicateDAOTest.java.

Caso de proba: ExperimentViewModelTest.java	
Descrición	Executa as probas necesarias para que a lóxica do ExperimentViewModel que abstrae as vistas do experimento teña un comportamento esperado.
Probas	<ul style="list-style-type: none"> · Compróbase que o ExperimentViewModel pode recuperar os IDs das placas. · Compróbase que o ExperimentViewModel pode recuperar os nomes das placas.

Táboa 20. Caso de proba: ExperimentViewModelTest.java.

Caso de proba: BasicOutputSorterTest.java	
Descrición	Executa as probas necesarias para que a lóxica da clase de ordenación de arquivos BasicOutputSorter teña un comportamento esperado.
Probas	<ul style="list-style-type: none"> · Compróbase que BasicOutputSorter pode ordenar arquivos. · Compróbase que BasicOutputSorter pode validar unha ruta.

Táboa 21. BasicOutputSorterTest.java.

Caso de proba: ReplicatePathCreatorTest.java	
Descrición	Executa as probas necesarias para que a lóxica do creador de directorios ReplicatePathCreator teña un comportamento esperado.
Probas	<ul style="list-style-type: none"> · Compróbase que ReplicatePathCreator pode crear os directorios.

Táboa 22. ReplicatePathCreatorTest.java.

MANUAL DE USUARIO

11 Guía de instalación do software necesario

No apartado que se presenta a continuación proporciónase unha guía para a instalación e configuración do software que é necesario ter previamente ó despregue do Asistente de Laboratorio para MALDI. Durante a guía suporase que a instalación se realiza sobre un sistema operativo GNU / Linux Ubuntu moderno ou algún derivado.

11.1 Instalar Java

Para instalar Java, bastará con instalar o paquete `openjdk-7-jre` dende os repositorios da distribución GNU / Linux que esteamos a utilizar, da seguinte maneira:

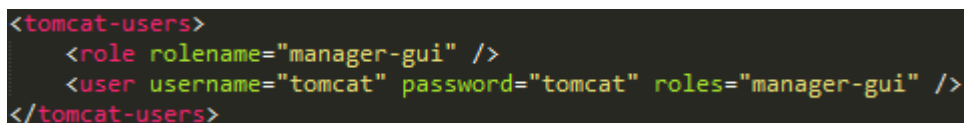
```
$ sudo apt-get install openjdk-7-jre
```

11.2 Instalar e configurar Tomcat

No caso de Tomcat cómpre instalar dous paquetes: `tomcat7`, e `tomcat7-admin`. O primeiro é o propio software Tomcat, e o segundo proporciona unha interface web para administralo. Polo tanto, executamos o seguinte comando na terminal:

```
$ sudo apt-get install tomcat7 tomcat7-admin
```

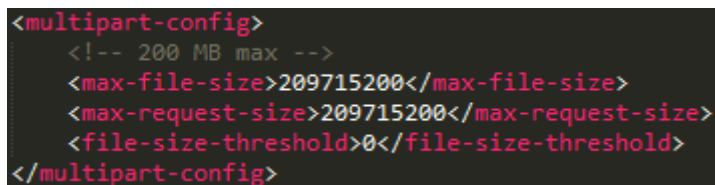
Unha vez instalado, hai que configuralo para podelo utilizar adecuadamente. Primeiro, é necesario engadir un usuario para a web de administración, editando o ficheiro `/etc/tomcat7/tomcat-users.xml` tal e como se amosa na Figura 21.



```
<tomcat-users>
  <role rolename="manager-gui" />
  <user username="tomcat" password="tomcat" roles="manager-gui" />
</tomcat-users>
```

Figura 21. Configuración do usuario da web de administración.

Ademais, hai un límite por defecto de 50 MB para os ficheiros que se despreguen en Tomcat, o cal debemos aumentar, pois o paquete `MALDILabAssistant.war` que contén o sistema desenvolvido pesa pouco máis de 100 MB. Na Figura 22 amósase a maneira de aumentar o límite a 200 MB dende o ficheiro `/usr/share/tomcat7-admin/manager/WEB-INF/web.xml`.



```
<multipart-config>
  <!-- 200 MB max -->
  <max-file-size>209715200</max-file-size>
  <max-request-size>209715200</max-request-size>
  <file-size-threshold>0</file-size-threshold>
</multipart-config>
```

Figura 22. Configuración do tamaño máximo de ficheiro a despregar.

11.3 Instalar e configurar MySQL

Para instalar MySQL dende os repositorios haberá que executar o seguinte comando:

```
$ sudo apt-get install mysql-client mysql-server
```

Posteriormente, hai que executar o script `createdb.sql` proporcionado no CD de instalación. Supoñendo que nos atopamos no mesmo directorio que o ficheiro, e que durante a instalación de MySQL se definiu un usuario “root” con contrasinal “root”, o comando a executar sería o seguinte:



```
$ mysql -u root -proot < createdb.sql
```

12 Guía de instalación do Asistente de Laboratorio para MALDI

Unha vez instalado o software necesario, o único paso para a instalación consiste en despregar o ficheiro `MALDILabAssistant.war` que se proporciona no CD de instalación. Para tal fin, cómpre abrir un explorador web e ir á dirección `localhost:8080/manager`, o cal fará que apareza unha ventá como a que se amosa na Figura 23.

Figura 23. Autenticación na web de administración de Tomcat.

Unha vez introducidos o usuario e contrasinal, que son os que se definiran na Figura 21, teremos acceso o panel de administración, que ten un aspecto como o da Figura 24.

Gestor de Aplicaciones Web de Tomcat

Mensaje: OK

Gestor					
Listar Aplicaciones	Ayuda HTML de Gestor	Ayuda de Gestor	Estado de Servidor		
Aplicaciones					
Trayectoria	Versión	Nombre a Mostrar	Ejecutándose	Sesiones	Comandos
/	Ninguno especificado		true	0	Arrancar <input type="button" value="Parar"/> <input type="button" value="Recargar"/> <input type="button" value="Replegar"/> <input type="button" value="Expirar sesiones"/> sin trabajar ≥ 30 minutos
/host-manager	Ninguno especificado	Tomcat Host Manager Application	true	0	Arrancar <input type="button" value="Parar"/> <input type="button" value="Recargar"/> <input type="button" value="Replegar"/> <input type="button" value="Expirar sesiones"/> sin trabajar ≥ 30 minutos
/manager	Ninguno especificado	Tomcat Manager Application	true	2	Arrancar <input type="button" value="Parar"/> <input type="button" value="Recargar"/> <input type="button" value="Replegar"/> <input type="button" value="Expirar sesiones"/> sin trabajar ≥ 30 minutos

Figura 24. Xestor de aplicacións web de Tomcat.

A continuación buscamos na web o panel “Desplegar”, e dentro, na parte de “Archivo WAR a desplegar”, poderemos seleccionar o arquivo `MALDILabAssistant.war`

proporcionado no CD de instalación mediante o botón “Examinar”, tal e como se amosa na Figura 25.

Figura 25. Menú para o despregue de aplicacións empaquetadas en WAR.

Por último, cómpre facer clic no botón “Desplegar” e, tras esperar un intre, poderemos acceder á aplicación a través da dirección localhost:8080/MALDILabAssistant, na cal se nos presentará unha web como a que amosa a Figura 26.

MALDI Lab Assistant

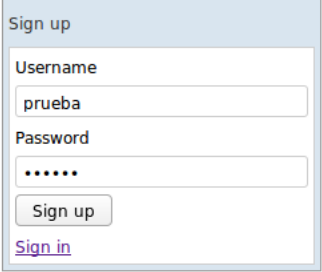
Figura 26. Primeira páxina da aplicación.

13 Guía de uso do Asistente de Laboratorio para MALDI

13.1 Proceso de rexistro

Para comezar a utilizar a aplicación é preciso dispoñer dunha conta de usuario, pois a ela estarán vinculados os experimentos que gardemos no futuro. Para rexistrarse basta con seguir o enlace “Sign up” que se pode observar na Figura 26, e completar o formulario de rexistro definindo un novo nome de usuario e contrasinal, como amosa a Figura 27.

MALDI Lab Assistant



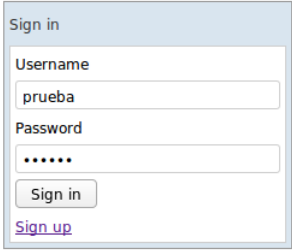
The screenshot shows a web form titled "Sign up" for the MALDI Lab Assistant application. It contains two input fields: "Username" with the text "prueba" and "Password" with masked characters "*****". Below the fields are two buttons: "Sign up" and a link "Sign in". At the bottom of the page, there is a footer with the text "2014 SING" followed by links "About us", "GitHub", and "Documentation".

Figura 27. Formulario de rexistro.

13.2 Proceso de login

Se xa estamos rexistrados, basta con introducir o noso nome de usuario e contrasinal no formulario de login inicial, tal e como se amosa na , para autenticarse.

MALDI Lab Assistant



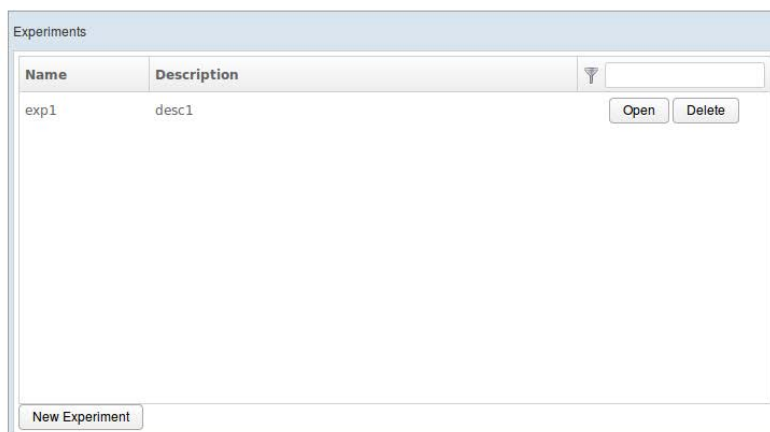
The screenshot shows a web form titled "Sign in" for the MALDI Lab Assistant application. It contains two input fields: "Username" with the text "prueba" and "Password" with masked characters "*****". Below the fields are two buttons: "Sign in" and a link "Sign up". At the bottom of the page, there is a footer with the text "2014 SING" followed by links "About us", "GitHub", and "Documentation".

Figura 28. Forumulario de login.

13.3 Páxina principal

Unha vez identificado na aplicación, pódese observar un listado de tódolos nosos experimentos, tal e como amosa a Figura 29.

MALDI Lab Assistant



[2014 SING](#) [About us](#) [GitHub](#) [Documentation](#)

Figura 29. Listado dos experimentos do usuario.

Pódense filtrar os experimentos que aparecen no listado escribindo na barra de busca da esquina superior dereita. Para cada experimento, podemos continuar a súa edición mediante o botón “Open”, ou eliminalo dándolle ó botón “Delete”. Se o que se desexa é crear un novo experimento, debe premerse o botón “New Experiment”, que nos levará á vista de edición dos datos do experimento, que se pode observar na Figura 30.

MALDI Lab Assistant

[2014 SING](#) [About us](#) [GitHub](#) [Documentation](#)

Figura 30. Páxina de edición dos datos do experimento.

13.4 Vista de edición do experimento

Nesta páxina pódese editar o nome, descrición e datas de inicio e fin do experimento a crear. Despois cómpre especificar o número de filas e columnas que ten a placa MALDI física sobre a que se porán as mostras, así como o seu tipo (numéricas, minúsculas, ou maiúsculas). Na táboa da parte inferior da páxina pódense engadir condicións ó experimento mediante o botón de “+” da esquerda, borrarlas co botón de “-” da dereita, e cambiar a súa cor mediante a paleta situada á esquerda do nome de cada unha. Cada condición pódese despregar premendo o botón con forma de frecha da columna esquerda, para engadirlle mostras de forma análoga a como se engaden as condicións. Así mesmo, pódenselle engadir réplicas a cada mostra de igual forma, despregando cada mostra e premendo no botón “+” debaixo de cada mostra.

Unha vez rematada a edición dos datos, prémese o botón “Save” para gardar os datos, e se todos están completos aparece a pestana “Design” que nos leva á vista de edición de placa que se amosa na Figura 31. Pódese, ademais, premer o botón “Reset” se queremos borrar os datos introducidos ou devolvelos a un estado anterior no caso de estar editando un experimento anteriormente creado.

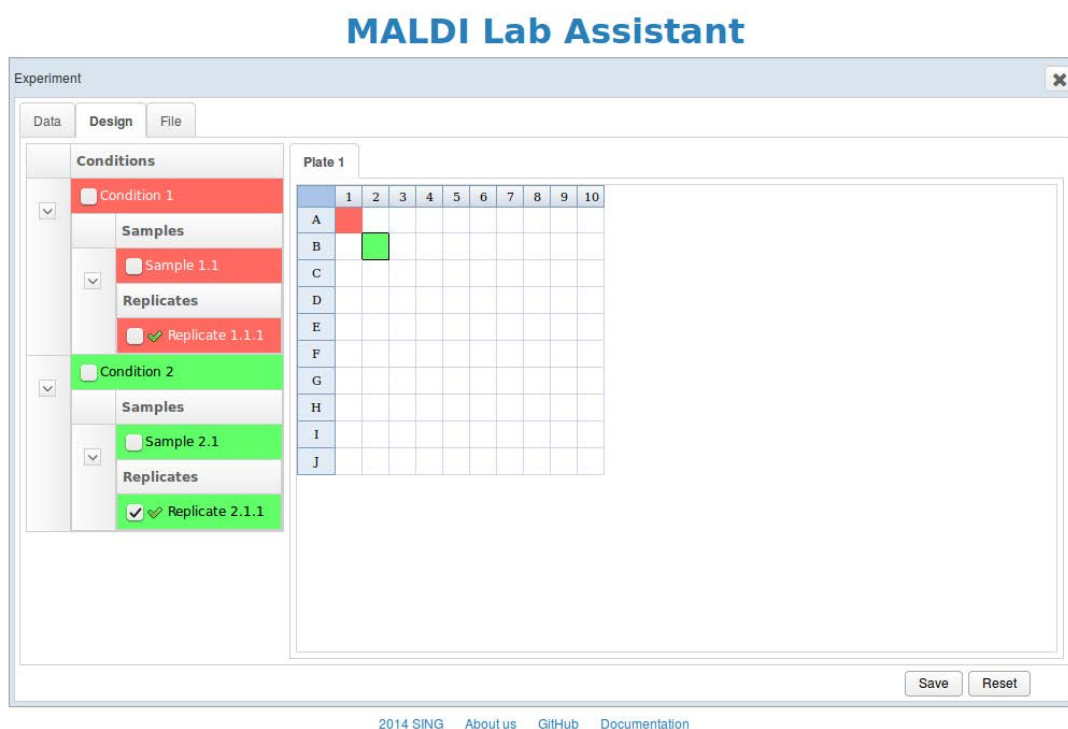


Figura 31. Páxina de edición das placas do experimento.

13.5 Vista da edición das placas do experimento

Na parte esquerda da aplicación obsérvase un listado de tódalas condicións, mostras e réplicas que se definiron previamente, mentres que na parte dereita hai unha matriz que representa unha placa MALDI, co tamaño especificado na vista de edición do experimento. No caso de que o número de réplicas fose maior que o número de celas da matriz, xéranse tantas placas como sexa preciso, e para editar cada unha se selecciona dende a barra de pestanas inmediatamente enriba da matriz.

Para colocar unha réplica no padrón da placa, basta con seleccionar o cadro á esquerda da réplica, facer clic na matriz, e despois facer clic na cela na que se queira colocar. Unha vez colocadas tódalas réplicas, e pulsado o botón “Save”, aparecerá a pestana “File”, que abrirá a interface de ordenado de arquivos, co aspecto que se amosa na Figura 32.



Figura 32. Páxina de ordenado de arquivos.

13.6 Vista de ordenado de arquivos

Co fin de ordenar os arquivos xerados polo APARATO, o primeiro é subir ditos arquivos comprimidos en ZIP ou TAR mediante o botón “Upload”. Dito arquivo debe ter a estrutura de directorios que se amosa na Figura 33.

```
[1]
|-> A1.csv
|-> A2.csv
|-> B6.csv
|-> D7.csv
[2]
|-> A2.csv
|-> C9.csv
|-> F6.csv
|-> G5.csv
```

Figura 33. Estrutura de directorios a subir á aplicación.

É dicir, un directorio por cada placa MALDI, cuxo nome sexa o ID da placa e, dentro de cada directorio, os arquivos csv cos datos de cada cela.

A continuación pódense elixir varias opcións para a ordenación dos arquivos, que consisten en crear ou non un directorio para condicións, e en crear ou non un directorio para mostradas, de forma que os datos que se poden observar na Figura 33 poderían quedar como os da Figura 34, Figura 35, Figura 36, ou Figura 37.

```
[Condición 1]
  |-> [Mostra 1]
        |-> Réplica 1.csv
        |-> Réplica 2.csv
  |-> [Mostra 2]
        |-> Réplica 3.csv
        |-> Réplica 4.csv
[Condición 2]
  |-> [Mostra 3]
        |-> Réplica 5.csv
        |-> Réplica 6.csv
  |-> [Mostra 4]
        |-> Réplica 7.csv
        |-> Réplica 8.csv
```

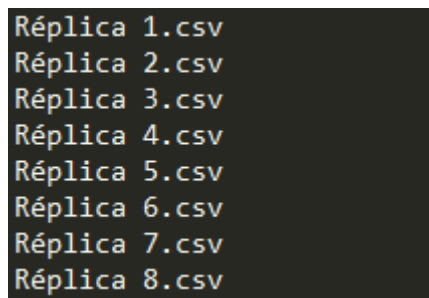
Figura 34. Archivos ordenados con condiciones e muestras.

```
[Condición 1]
  |-> Réplica 1.csv
  |-> Réplica 2.csv
  |-> Réplica 3.csv
  |-> Réplica 4.csv
[Condición 2]
  |-> Réplica 5.csv
  |-> Réplica 6.csv
  |-> Réplica 7.csv
  |-> Réplica 8.csv
```

Figura 35. Archivos ordenados con condiciones.

```
[Mostra 1]
  |-> Réplica 1.csv
  |-> Réplica 2.csv
[Mostra 2]
  |-> Réplica 3.csv
  |-> Réplica 4.csv
[Mostra 3]
  |-> Réplica 5.csv
  |-> Réplica 6.csv
[Mostra 4]
  |-> Réplica 7.csv
  |-> Réplica 8.csv
```

Figura 36. Archivos ordenados con muestras.



```
Réplica 1.csv  
Réplica 2.csv  
Réplica 3.csv  
Réplica 4.csv  
Réplica 5.csv  
Réplica 6.csv  
Réplica 7.csv  
Réplica 8.csv
```

Figura 37. Archivos ordenados sen condicións nin mostras.

Para rematar, tan só quedaría premer o botón “Download” para descargar un arquivo .zip coa nova estrutura de directorios ordenada.