



ugr | Universidad
de Granada

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA

Pradobot

Bot de Telegram para Moodle

Autor
Luis Gil Guijarro

Directores
Nombre Apellido1 Apellido2 (tutor1)
Nombre Apellido1 Apellido2 (tutor2)



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, 1 Septiembre de 2017

Pradobot: *Bot* de Telegram para Moodle

Luis Gil Guijarro

Palabras clave: Bot, Telegram, Moodle, Docencia, Chat, Mensaje

Resumen

Hoy en día los bots están surgiendo para asistir en múltiples tareas y una plataforma que presta un fuerte apoyo a estos es Telegram. El objetivo del proyecto es utilizar un *bot* de Telegram y la plataforma Moodle para facilitar las tareas relacionadas con la docencia a profesores y alumnos.

Pradobot: Telegram bot for Moodle

Luis Gil Guijarro

Keywords: Telegram, Bot, Moodle, Educative, Message, Chat

Abstract

Nowadays bots are making their own way onto our lives. From the robot voice that guides you when you give the telephone company a call to Youtube where bots supervise chats ensuring people don't employ bad words they can be used for a lot of useful tasks.

Recently they have reached Telegram. Since October 2016 Telegram has released an API where programmers and enthusiasts can build their own bot for a lot of different purposes: greeting people who enter in a chat, looking for gifs, looking for music, travel plans, film recommendations... but in educative tasks, they are rarely used.

The immediate nature of instant messages and the widespread use of programs like Whatssap and Telegram can captivate students and professors to use them for educative purposes contributing in their way to reach their career goals. Believing in this idea the aim of this work is to build a Telegram bot that assists university courses in some of their day-to-day problems like helping solve doubts, providing students with the date of the upcoming delivers, getting their marks, managing professors academic tutoring...

So where are we going to find all the students and professor? We are going to use Moodle which is one of the most widely used educative platform in the university world.

Moodle since version 2.0 provides an API that can be accessed using REST and enables external applications to interact with it so they can make ordinary tasks like enrolling students, getting marks, adding resources to courses... Configuring Moodle correctly can be a tricky topic but knowing that security is a must we are going to try to develop a Moodle special configuration so the Moodle courses that use our bot don't disturb the others courses which don't.

We will use the special graphical keyboards and the menus that Telegram provides to bots to interact with users. With these special keyboards bots can easily indicate the user all the tasks enabled to them. We are also using one-step commandos in group chats instead of menus so they provide users with useful information causing the less noisy possible.

We are going to employ collaborative technology like github and git as version control system so other people could contribute to help us develop our bot and hopefully making us useful suggestions that enable us to build a quality product. In addition we will employ technology like Vagrant to

automatically build virtual machines and Ansible to provision this virtual machines to ease the problem of configuring and installing our bot. We will write unit tests because we want to make sure adding new features don't compromise the overall stability of the project.

Finally we will use the Ruby programming language to develop the bot. Even though it is not as popular as it used to be it has proven to have enough tools to ensure the construction of successful projects. Once it is finished and because we believe in open source software we will release it under the MIT licence one of the less restrictive licences.

Yo, **Luis Gil Guijarro**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 44066149-N, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Luis Gil Guijarro

Granada a 1 de Septiembre de 2017 .

D. **Nombre Apellido1 Apellido2 (tutor1)**, Profesor del Área de XXXX del Departamento YYYY de la Universidad de Granada.

D. **Nombre Apellido1 Apellido2 (tutor2)**, Profesor del Área de XXXX del Departamento YYYY de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Pradobot: Bot de Telegram para Moodle***, ha sido realizado bajo su supervisión por **Luis Gil Guijarro**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a X de mes de 201 .

Los directores:

Nombre Apellido1 Apellido2 (tutor1)
Apellido2 (tutor2)

Nombre Apellido1 Apellido2 (tutor1)

Agradecimientos

Poner aquí agradecimientos... También a todo profesor que ha desmanejado mis dudas en su despacho y a aquellos que no me han bajado la nota al ir después del examen.

Índice general

1.	1
1.1. Introducción	1
1.1.1. Análisis preliminar. Estudio viabilidad del proyecto.	1
1.1.2. Descripción general del sistema y objetivos	2
2.	5
2.1. Planificación	5
2.1.1. Lista actividades y sus fases	5
2.1.2. Presupuesto	8
3.	9
3.1. Ingeniería de requisitos	9
3.1.1. Descripción de los implicados.	9
3.1.2. Especificación de requisitos	13
3.2. Casos de uso	16
3.2.1. Diagrama casos uso	16
3.2.2. Descripción casos uso	17
3.2.3. Diagrama de paquetes	59
3.2.4. Diagramas de actividad	60
4.	69
4.1. Análisis de los requisitos	69
4.1.1. Modelo conceptual	69
4.1.2. Diagramas secuencia	70
4.1.3. Contratos	79
5.	93
5.1. Diseño	93
5.1.1. Diseño de la interacción con el usuario	93
5.1.2. Diseño de la estructura	94
5.1.3. Diseño de la configuración de Moodle	103
5.1.4. Diseño de la base de datos	111
5.1.5. Diseño de los tests	112

6.	115
6.1. Implementación	115
6.1.1. Programa	115
6.1.2. Base de datos	118
6.1.3. Moodle	119
6.1.4. Despliegue	119
6.1.5. Aprivisionamiento	121
6.1.6. Control ejecución	124
6.1.7. Tests	125
7.	131
7.1. Pruebas	131
7.1.1. Prueba despliegue	131
7.1.2. Tests	133
7.1.3. Rendimiento	134
7.2. Conclusiones	137
8.	139
8.1. Glosario de términos	139
Bibliografía	143

Índice de figuras

2.1. Planificación temporal actividades y sus fases	7
3.1. Diagrama de los casos de uso más relevantes del sistema	16
3.2. Diagrama de los casos de uso más relevantes del sistema	59
3.3. Diagrama actividad CU-1.1 Clasificar mensaje	60
3.4. Diagrama actividad CU-1.2 Clasificar mensaje privado	61
3.5. Diagrama actividad CU-7.1, CU-7.2, CU-7.3 Mostrar solución duda, mostrar dudas resueltas y entregas por chat grupal	61
3.6. Diagrama actividad de CU-4.3 Solicitar tutoría	62
3.7. Diagrama actividad conjunto de CU-4.4 Aceptar petición tutoría, CU-4.5 Denegar petición tutoría	63
3.8. Diagrama actividad de CU-4.6 Ver información tutoría	64
3.9. Diagrama actividad de CU-4.7 Ver peticiones realizadas	65
3.10. Diagrama actividad de CU-5.1 Crear duda	65
3.11. Diagrama actividad conjunto de CU-5.2 Ver dudas sin solución, CU-5.3 Ver mis dudas, CU-5.4 Ver dudas resueltas,	66
3.12. Diagrama actividad conjunto de CU-5.5 Crear respuesta a duda, CU-5.6 Ver respuesta duda, CU-5.7 Ver solución duda, CU-5.8 Borrar mi duda, CU-5.9 Elegir solución mi duda, CU-10 Borrar duda, CU-11 Elegir solución a duda	66
3.13. Diagrama actividad CU-6.1 Ver entregas	67
3.14. Diagrama actividad CU-6.2 Mostrar calificaciones	67
4.1. Diagrama conceptual del sistema	69
4.2. DS: Cambiar curso (CU-1.4)	70
4.3. DS: Aceptar denegar peticiones (CU-4.4, CU-4.5)	71
4.4. DS: Ver información tutoría (CU-4.6)	71
4.5. DS: Establecer tutoría (CU-4.1)	72
4.6. DS: Borrar tutoría (CU-4.2)	72
4.7. DS: Solicitar asistir tutoría (CU-4.3)	73
4.8. DS: Ver solicitudes realizadas (CU-4.7)	73
4.9. DS: Crear duda (CU-5.1)	74
4.10. DS: Ver dudas sin solución (CU-5.2)	74

4.11. DS: Ver mis dudas (CU-5.3)	75
4.12. DS: Ver dudas con solución (CU-5.4)	75
4.13. DS: Crear respuesta (CU-5.5)	76
4.14. DS: Ver respuestas (CU-5.6)	76
4.15. DS: Ver solución a duda (CU-5.7)	77
4.16. DS: Borrar duda (CU-5.8, CU-5.10)	77
4.17. DS: Elegir solución duda (CU-5.9, CU-5.11)	78
4.18. DS: Ver proximas entregas (CU-6.1)	78
4.19. DS: Consultar calificaciones entregas (CU-6.2)	79
5.1. Arriba menú tipo Inline, abajo menú tipo teclado	94
5.2. Diagrama secuencia recibir_mensaje clase Mensajero: el bot recibe un nuevo mensaje	96
5.3. Diagrama secuencia recibir_mensaje clase ManejadorMensajesChatsPrivados	96
5.4. Diagrama secuencia añadir_nuevo_usuario clase ManejadorMensajesChatsPrivados	97
5.5. Diagrama secuencia obtener_menu_inicio clase ManejadorMensajesChatsPrivados	97
5.6. Diagrama secuencia ejecutar_menu_inicio clase ManejadorMensajesChatsPrivados	98
5.7. Diagrama secuencia muestra ejecución thread iniciado por ManejadorMensajesChatsPrivados	98
5.8. Diagrama comunicación recibir_mensaje clase SolicitarTutoria	101
5.9. Diagrama comunicación mostrar_tutorías clase SolicitarTutoria	101
5.10. Diagrama comunicación solicitar_tutoria clase SolicitarTutoria	102
5.11. Diagrama de clases del sistema	102
5.12. Habilitando protocolo REST en Moodle	104
5.13. Creando rol webservices_student	105
5.14. Creando webservice llamado webservices_bot	107
5.15. Añadiendo funciones a el webservice llamado webservices_bot	107
5.16. Asignado el rol webservices_bot al usuario creado en Moodle para el bot	108
5.17. Asignado el rol webservices_bot al usuario creado en Moodle para el bot	108
5.18. Creando token para el usuario en Moodle del bot	109
5.19. Añadiendo al usuario bot al curso curso7	109
5.20. Llamando a algunas de las funciones de webservices_bot a través del navegador web	109
5.21. Diagrama ER de la base de datos	111
5.22. Esquema base datos en tercera forma normal	112
6.1. Uso del método SendMessage por parte del bot	116
6.2.	117

6.3. Utilizando aws-cli para generar claves efímeras para el despliegue	120
7.1. Ejecutando los tests de la aplicación	131
7.2. Finalización de aprovisionamiento con Ansible	132
7.3. Iniciamos la ejecución de la aplicación utilizando Capistrano	132
7.4. Detenemos la ejecución de la aplicación	133
7.5. Tras ejecutar los tests de la aplicación	133
7.6. Reporte de cobertura de código de los tests	134
7.7. Código utilizado para la medición del tiempo que se tarda en procesar un mensaje	135
7.8. Tiempo que tarda pradobot en procesar mensajes procedentes de un chat privado y de un chat grupal	135
7.9. Envío de mensaje utilizando telegram-cli	136

Índice de cuadros

3.1.	Curso normal de CU-1.1. Clasificar mensaje.	18
3.2.	Cursos alternos de CU-1.1. Clasificar mensaje.	18
3.3.	Curso normal de CU-1.2. Clasificar mensaje privado.	19
3.4.	Cursos alternos de CU-1.2. Clasificar mensaje privado.	19
3.5.	Curso normal de CU-1.3. Inicializar usuario.	20
3.6.	Cursos alternos de CU-1.3. Inicializar usuario.	20
3.7.	Curso normal de CU-1.4. Cambiar de curso.	21
3.8.	Curso normal de CU-2. Dar de alta usuario en el sistema. . .	23
3.9.	Cursos alternos de CU-2. Dar de alta usuario en el sistema. .	24
3.10.	Curso normal de CU-3.1. Cambiar a menú entregas.	24
3.11.	Curso normal de CU-3.2. Cambiar a menú tutorías.	25
3.12.	Curso normal de CU-3.3. Cambiar a menú tutorías.	26
3.13.	Curso normal de CU-3.4. Cambiar a menú chat.	27
3.14.	Curso normal de CU-3.5. Cambiar a menú dudas.	28
3.15.	Curso normal de CU-3.6. Volver al menú anterior.	29
3.16.	Curso normal de CU-4.1 Establecer tutoría.	31
3.17.	Cursos alternos de CU-4.1. Establecer tutoría.	32
3.18.	Curso normal de CU-4.2. Borrar tutoría	33
3.19.	Cursos alternos de CU-4.2. Borrar tutoría.	34
3.20.	Curso normal de CU-4.3. Solicitar tutoría.	35
3.21.	Cursos alternos de CU-4.3. Solicitar tutoría.	36
3.22.	Curso normal de CU-4.4 Aceptar petición tutoría.	37
3.23.	Cursos alternos de CU-4.4. Aceptar petición tutoría.	37
3.24.	Curso normal de CU-4.5 Denegar petición tutoría.	38
3.25.	Cursos alternos de CU-4.5 Denegar petición tutoría.	38
3.26.	Curso normal de CU-4.6 Ver información tutoría.	39
3.27.	Cursos alternos de CU-4.6 Ver información tutoría.	40
3.28.	Curso normal de CU-4.7. Ver solicitudes a tutoría realizadas.	40
3.29.	Cursos alternos de CU-4.7 Ver solicitudes a tutoría realizadas.	41
3.30.	Curso normal de CU-5.1. Crear duda.	41
3.31.	Cursos alternos de CU-5.1 Crear duda.	42
3.32.	Curso normal de CU-5.2. Ver dudas.	42
3.33.	Cursos alternos de CU-5.2 Ver dudas.	42

3.34. Curso normal de CU-5.3. Ver mis dudas.	43
3.35. Cursos alternos de CU-5.3 Ver mis dudas.	43
3.36. Curso normal de CU-5.4. Ver dudas resueltas.	44
3.37. Cursos alternos de CU-5.4 Ver dudas resueltas.	44
3.38. Curso normal de CU-5.5. Ver dudas resueltas	45
3.39. Curso normal de CU-5.6. Ver respuestas duda.	46
3.40. Cursos alternos de CU-5.6 Ver respuestas duda.	47
3.41. Curso normal de CU-5.7. Ver solución a duda.	47
3.42. Cursos alternos de CU-5.7 Ver solución a duda.	47
3.43. Curso normal de CU-5.8. Borra mi duda.	48
3.44. Cursos alternos de CU-5.8 Borrar mi duda.	48
3.45. Curso normal de CU-5.9. Elegir solución mi duda..	49
3.46. Curso normal de CU-5.10. Borrar duda.	50
3.47. Cursos alternos de CU-5.9 Borrar duda.	50
3.48. Curso normal de CU-5.11. Elegir solución a duda.	51
3.49. Curso normal de CU-6.1. Ver entregas.	52
3.50. Cursos alternos de CU-6.1 Ver entregas.	53
3.51. Curso normal de CU-6.2. Consultar calificaciones.	53
3.52. Cursos alternos de CU-6.2 Consultar calificaciones.	54
3.53. Curso normal de CU-6.3. Consultar dudas chat grupo.	55
3.54. Cursos alternos de CU-6.3. Consultar dudas chat grupo.	55
3.55. Curso normal de CU-7.1. Mostrar solución duda chat grupal.	56
3.56. Curso normal de CU-7.2. Mostrar próximas entregas chat grupal.	57
3.57. Curso normal de CU-7.3. Mostrar dudas resueltas.	58

Capítulo 1

1.1. Introducción

Hoy en día aplicaciones como WhatsApp o Telegram son usadas de manera habitual por cualquier persona no siendo la excepción los estudiantes y profesores de la universidad.

Habiendo lanzado hace poco más de un año Telegram el soporte para utilizar *bots* (del inglés robot) y conociéndose el uso que se da hoy en día a los *bots* para la asistencia a distintas tareas, resulta interesante crear uno diseñado explícitamente para asistir en la gestión docente de una asignatura, de tal forma, que tareas básicas como la solicitud de tutorías, entregas de asignaturas, notas sea algo sencillo de hacer con este bot.

1.1.1. Análisis preliminar. Estudio viabilidad del proyecto.

Ante todo, lo primero es analizar si es posible, la realización de este proyecto o, en todo caso, ver si es necesario introducir algún matiz. Primero vamos a analizar la parte técnica, es decir, si es realizable un bot de Telegram y que este interactúe con una instancia de Moodle. Voy a listar los puntos que permiten afirmar, bajo mi punto de vista, que se trata de algo perfectamente realizable:

- Telegram cuenta con una API que da amplias funcionalidades a un programa para comunicarse con un usuario que contacte con él, bien sea a través de mensajes de texto, imágenes, iconos, teclados gráficos... todo esto proporciona un entorno rico para crear una interacción sencilla y fluida entre un usuario y el programa, permitiendo al bot asistir al usuario en alguna tarea. En nuestro caso concreto esta tarea consistiría en permitir realizar labores típicas relacionadas con la docencia tales como crear/solicitar tutorías, consultar notas, resolver dudas...
- Moodle desde la versión 2.0 permite el acceso a datos de una instancia

de Moodle utilizando una interfaz tipo REST, el acceso a estos datos se realiza a través de una serie de funciones llamadas *webservice functions*. Es más, Moodle permite filtrar qué usuarios pueden acceder a qué funciones y por tanto a qué datos de una instancia de Moodle teniendo así un control sobre qué usuarios pueden utilizar la API REST y para qué datos.

Con estos dos puntos ya se puede afirmar que es posible construir un programa que utilice Telegram como *bot* y a su vez, tenga acceso a datos de una instancia de Moodle de manera controlada. Ahora bien tiene: ¿Tiene sentido?

Utilizar un chat de Telegram para un curso permite a los estudiantes y al profesor compartir cualquier información relacionada con el curso de forma rápida y sencilla: cambios de aula, recursos de interés, preguntar dudas... el *bot* aporta riqueza a esta comunicación almacenando, por ejemplo, todos aquellos recursos de interés y que no se pierdan en el flujo de una conversación en un chat grupal. También permitiría guardar todas aquellas dudas que generan los alumnos a lo largo del curso junto con sus respuestas, o incluso, un alumno podría ver cuanta gente tiene por delante para asistir a una tutoría y así solicitar la tutoría para el día que mejor le convenga.

Viendo que tiene viabilidad técnica y habiendo enumerado alguna de las cualidades prácticas que puede aportar, paso a perfilar el proyecto en las siguientes secciones.

1.1.2. Descripción general del sistema y objetivos

El sistema hará uso, por una parte, de la plataforma Moodle para obtener datos relacionados con una asignatura (nº de alumnos apuntados, entregas abiertas, notas, plazos...) y, por otro lado, de Telegram para asistir a un usuario en tareas vinculadas con la docencia (petición de tutorías, gestión de dudas...)

Además se utilizarán los chats grupales de Telegram relacionados con una asignatura para dar información de interés general para todos los integrantes de esa asignatura.

A modo de resumen, los principales objetivos que se pretenden alcanzar son:

OBJ-1 Uso de Telegram y Moodle para asistir en la tareas relacionadas con la docencia a profesores tales como: tutorías, gestión de dudas..

OBJ-2 Uso de Telegram y Moodle para facilitar a los alumnos las actividades relacionadas con su participación en una asignatura: entregas

abiertas, plazos, notas..

OBJ-3 Asistencia en chats grupales de Telegram relacionados con una asignatura mediante el aporte de información relevante para el conjunto del alumnado.

Como objetivo secundario destacaríamos:

OBJ-4 Correcta configuración Moodle para el uso de la aplicación.

Capítulo 2

2.1. Planificación

En este apartado se muestra la planificación diseñada para el desarrollo del proyecto.

Como este es un proyecto fácilmente subdivisible en diferentes partes y al que se le pueden ir añadiendo nuevas funciones conforme se completa una, se ha decidido emplear el desarrollo iterativo para su realización.

2.1.1. Lista actividades y sus fases

- **Estudio bots Telegram.**

- Contenido: se analiza la capacidad de la API de Telegram, se ve el funcionamiento de los bots creados hasta la fecha.
- Estimación: 10h.

- **Estudio Moodle.**

- Contenido: se estudia el funcionamiento interno de Moodle y el funcionamiento de su API.
- Tiempo empleado: 15h.

- **Configuración Moodle.**

- Contenido: se prueban diferentes formas de configurar una instancia de Moodle.
- Tiempo empleado: 20h.

- **Prototipo pruebas Moodle.**

- Contenido: se comprueba el funcionamiento de la API de Moodle conforme se prueban diferentes configuraciones.

- Tiempo empleado: 8h.

- **Entregas chat privado.**

- Contenido: primera iteración se busca que el bot pueda acceder a la API de Moodle y mostrar información sobre las entregas a usuarios.
- Fases.
 - Especificación:15h.
 - Análisis:20h.
 - Diseño:25h
 - Implementación: 35h.
 - Pruebas:10h.

- **Entregas chat grupal.**

- Contenido: segunda iteración se distingue entre chat grupal y privado, se permite mostrar datos de entregas a través de un chat grupal.
- Fases.
 - Especificación:5h.
 - Análisis:7h.
 - Diseño:15h
 - Implementación: 15h.
 - Pruebas:4h.

- **Tutorías.**

- Contenido: tercera iteración se añade la posibilidad de que los estudiantes puedan solicitar tutorías y los profesores gestionarlas.
- Fases.
 - Especificación:10h.
 - Análisis:10h.
 - Diseño:15h
 - Implementación: 25h.
 - Pruebas:5h.

- **Dudas.**

- Contenido: cuarta iteración se permite la gestión de dudas de un curso.
- Fases.
 - Especificación: 15h.

- Análisis: 15h.
- Diseño: 20h
- Implementación: 40h.
- Pruebas: 10h.

■ Ajustes finales.

- Contenido: se ordena la documentación generada, se limpian las posibles asperezas surgidas en el desarrollo con el fin de tener la aplicación lista.
- Duración: 30h.

La planificación temporal seguida para desarrollar las siguientes actividades podemos apreciarla de manera visual en el siguiente diagrama de Gantt:

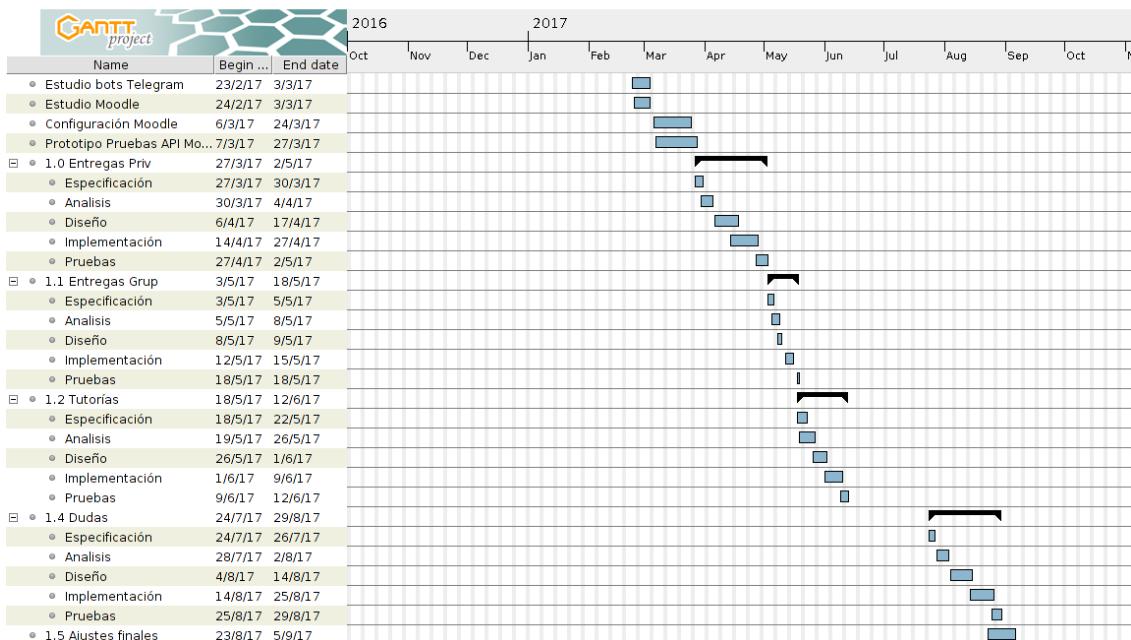


Figura 2.1: Planificación temporal actividades y sus fases

La memoria que compone este TFG la he ido realizando incrementalmente conforme desarrollaba el programa. Las etapas (especificación, análisis, diseño..) que componen las diferentes actividades mostradas, están en su respectiva sección del documento.

2.1.2. Presupuesto

Todos los programas utilizados para el desarrollo y funcionamiento del proyecto son software libre por lo que no hay gastos generados por la comprar licencias. Para el desarrollo del presente proyecto sería necesario un equipo informático estandar y para su funcionamiento podemos alojarlo en un servicio de cloud hosting:

- Cloud hosting VMware
 - Uso: 20 %.
 - Sistema operativo: CentOS.
 - Una IP pública.
 - 1GB RAM
 - Precio: **25€/mes**
- Ordenador portatil para desarrollo
 - Modelo: ASUSK-541
 - Procesador: Intel i5. 2.5GHz.
 - RAM: 12GB
 - Almacenamiento: 1TB
 - Precio **700€**

Además se incluye en el zip del trabajo un pdf llamado “Derecho Informatico: Ejemplo contrato desarrollo software pradobot” que aplica el derecho al desarrollo de pradobot.

Capítulo 3

3.1. Ingeniería de requisitos

En esta sección hablamos, en primer lugar, de los principales actores implicados en el sistema y de sus necesidades, tras lo cual, mostramos los requisitos funcionales, no funcionales y de información del sistema.

3.1.1. Descripción de los implicados.

Los principales implicados en el sistema son los profesores y los estudiantes.

Nombre	Descripción	Tipo	Responsabilidad
Estudiante	Usuario de Telegram que además está apuntado a un curso en la plataforma Moodle con rol de estudiante.	Usuario producto.	Acceso a información relacionada con los cursos en los que se encuentra apuntado, genera dudas y asiste a tutorías.
Profesor	Usuario de Telegram apuntado a un curso en la plataforma Moodle con rol de profesor.	Usuario producto	Responsable de una asignatura y de sus tutorías.
Usuario registrado	Usuario de Telegram que se ha identificado respecto al <i>bot</i> y éste lo tiene registrado.	Usuario producto	Acceso a funciones comunes para todos los usuarios que se registren en el <i>bot</i> .
Usuario sin identificar	Usuario de telegram que no se ha dado de alta respecto al <i>bot</i> .	Usuario producto	
Usuario chat grupal Telegram	Cualquier usuario de un chat de telegram asociado a un curso.	Usuario producto	Realiza consultas al <i>bot</i> con interacciones tipo usuario pregunta, bot responde, fin interacción.

Estudiante

Descripción	Usuario registrado en cursos de un servidor de Moodle con rol de estudiante, que accede a información de estos cursos, tales como entregas que ha realizado o que tiene que realizar y a su vez produce información relacionada con los cursos como peticiones a tutorías.
Tipo	Utiliza el sistema de forma directa y aporta información adicional relacionada con los cursos.
Responsabilidad	Acceder a la información del curso. Ver notas de sus entregas en un curso. Solicitar tutorías. Plantear dudas.
Criterios de éxito	Que el sistema le permita realizar sus actividades de la forma más sencilla posible.
Implicación	Utilizará el sistema de forma activa.
Comentarios/Cuestiones	

Usuario registrado

Descripción	Cualquier usuario que se ha registrado con respecto al bot
Tipo	Utiliza el sistema de forma directa y aporta información adicional relacionada con los cursos.
Responsabilidad	Plantear dudas.
Criterios de éxito	Poder tener un manejo de las dudas asociadas a un curso lo más simple posible.
Implicación	Utilizará el sistema de forma activa.
Comentarios/Cuestiones	Representa las actividades comunes que pueden realizar utilizando el bot los profesores y estudiantes

Profesor

Descripción	Usuario registrado en cursos de un servidor de Moodle con rol de profesor que está a cargo de la gestión de una serie de asignaturas, gestiona todo lo relacionado con éstas tal y como recursos (pdfs, urls, imágenes..), entregas, dudas y tutorías.
Tipo	Utiliza el sistema de forma directa y aporta información “indirecta” o adicional relacionada con los cursos.
Responsabilidad	Gestionar la información del curso. Ver entregas de un curso. Realizar entregas de un curso. Gestión tutorías. Plantear dudas.
Criterios de éxito	Que el sistema le permita realizar sus actividades de la forma más sencilla posible.
Implicación	Utilizará el sistema de forma activa.
Comentarios/Cuestiones	

Usuario chat grupal Telegram

Descripción	Cualquier usuario de un chat de Telegram en cuyo interior se encuentre el <i>bot</i> y que este chat esté asociado a un curso de Moodle.
Tipo	Utiliza el sistema para obtener información general de un curso.
Responsabilidad	Aportar información de interés general para el curso. Consultar información general del curso: preguntar fecha de próximas entregas de una asignatura, consultar dudas del curso.
Criterios de éxito	El sistema muestra claramente que acciones puede realizar y cual será el resultado de estas.
Implicación	Utilizará el sistema de forma activa.
Comentarios/Cuestiones	Al ser un chat donde participa mucha gente, las interacciones del <i>bot</i> tienen que ser breves y contundentes para evitar que genere ruido.

3.1.1.1. Necesidades principales de los implicados

Necesidad	Prioridad	Problema	Solución actual.	Solución propuesta.
Petición tutorías	Alta	Poder solicitar una tutoría a un profesor de una asignatura	Buscar el correo electrónico del profesor responsable de la asignatura y mandarle un correo electrónico al profesor	Solicitar al <i>bot</i> una tutoría para una asignatura y éste guarda las solicitudes y notifica al profesor
Gestionar tutorías	Alta	Gestionar las tutorías de las que es responsable y conocer quienes quieren asistir a ellas	Llevar una lista mentalmente o en el correo de estudiantes que han solicitado asistir a una de sus tutorías	Permitir al profesor definir tutorías a través del <i>bot</i> visibles para los estudiantes de sus cursos que pueden realizar solicitudes a ellas llevándose un recuento de las solicitudes realizadas
Resolver dudas curso	Alta	Resolver una duda sobre cómo actuar ante un problema surgido durante el desarrollo de las actividades de un curso	Buscar el correo del profesor, mandarle correo electrónico ó contactar con algún conocido y preguntarle	Guardar la duda del usuario y permitir que mediante el <i>bot</i> los usuarios de un curso aporten soluciones.
Conocer información acerca de las entregas	Alta	Poder conocer el estado de las entregas para un curso	Acceder a Moodle y para cada curso en el que se está matriculado ver si hay alguna actividad abierta	Indicando al <i>bot</i> que quiere saber las entregas de un curso, cuales de ellas están abiertas, si alguna tiene notas..

3.1.2. Especificación de requisitos

3.1.2.1. Requisitos funcionales

Debido a que se pueden incluir muchas funcionalidades y que el tiempo es limitado, las funciones a implementar por el sistema para cubrir las necesidades de los usuarios son:

- **RF-1** Acceso usuarios.
 - **RF-1.1** El sistema debe permitir a un usuario registrado en la instancia de moodle utilizar el *bot*.
- **RF-2** El *bot* debe permitir a los usuarios que lo usan especificar el curso sobre el cual tendrán efecto sus acciones.
- **RF-3** El *bot* debe dar a un usuario información acerca de las entregas para los cursos a los cuales tiene acceso el usuario:
 - **RF-3.1** Proporcionar a un estudiante información acerca de las entregas que se encuentran abiertas tales como: la fecha de entrega, descripción de qué hay que hacer en la entrega si la hubiera o cuántos días faltan para la entrega.
 - **RF-3.2** Información acerca de las notas que tiene para las entregas.
- **RF-4** Gestión de las dudas de un curso.
 - **RF-4.1** El sistema debe permitir a un profesor o estudiante crear dudas relacionadas con un curso.
 - **RF-4.2** El sistema debe permitir al usuario que crea la duda o al profesor marcar como resuelta una duda.
 - **RF-4.3** Cualquier usuario con acceso al curso debe poder ver las dudas para ese curso.
 - **RF-4.4** Cualquier usuario con acceso al curso debe poder aportar respuestas a una duda.
- **RF-5** Gestión de tutorías.
 - **RF-5.1** El profesor de un curso puede:
 - **RF-5.1.1** Definir las tutorías para ese curso.
 - **RF-5.1.2** Aprobar o denegar las peticiones que reciba de tutorías.
 - **RF-5.1.3** Conocer la cola de peticiones aprobadas para una tutoría.

- **RF-5.2** El estudiante de un curso puede:
 - **RF-5.2.1** Conocer las tutorías del profesor asociado a un curso.
 - **RF-5.2.2** Solicitar asistir a una tutoría de un curso.
 - **RF-5.2.3** Conocer el estado de sus solicitudes.
- **RF-6** Chats grupales de Telegram.
 - **RF-6.1** El profesor de un curso puede asociar un chat de grupo de Telegram a un curso de los que es responsable.
 - **RF-6.2** El bot tiene que aportar información general relevante para un curso en un chat grupal.
 - **RF-6.2.1** Fechas de las entregas abiertas para el curso asociado al chat.
 - **RF-6.2.2** Información acerca de una entrega en concreto.
 - **RF-6.2.3** Dudas actuales asociadas con el curso.
 - **RF-6.2.4** Tutorías del profesor.

3.1.2.2. Requisitos no funcionales

Empaquetamiento

- **RNF-1** El sistema debe poder aprovisionarse automáticamente.
- **RNF-2** El sistema debe poder desplegarse automáticamente a través de ssh.

Seguridad

- **RNF-3** Toda la comunicación entre el sistema y la instancia de Moodle debe estar cifrada utilizando ssl.

Interfaz

- **RNF-4** El sistema debe proporcionar al usuario menús gráficos para toda acción que requiere más de dos pasos.

Legales

- **RNF-5** El *bot* tiene que tener una licencia libre.

3.1.2.3. Requisitos de información

- **RI-1.** Usuario Moodle.

Almacenamos información de un usuario de Moodle.

Contenido: email, token, id_moodle

- **RI-2.** Cursos.

Información sobre un curso de Moodle.

Contenido: nombre, id_moodle_curso

- **RI-3.** Usuario Telegram.

Datos asociados a un usuario de Telegram.

Contenido: nombre_usuario, id_telegram

- **RI-4.** Cursos.

Información sobre un curso de Moodle.

Contenido: nombre, id_moodle_curso

- **RI-5.** Dudas.

Datos de las dudas de un curso.

Contenido: contenido

- **RI-6.** Respuestas.

Datos de las respuestas que tienen las dudas.

Contenido: contenido.

- **RI-7.** Tutorías.

Información asociada a las tutorías de un profesor.

Contenido: día semana, hora.

- **RI-8.** Tutorías.

Información asociada a las tutorías de un profesor.

Contenido: día semana, hora.

- **RI-9.** Peticiones.

Datos relacionados con las peticiones a tutorías.

Contenido: hora solicitud.

- **RI-10.** Chat Telegram.

Datos asociados a un chat de Telegram.

Contenido: nombre_chat, id_chat_telegram

3.2. Casos de uso

En esta sección procedemos a mostrar los casos de uso más relevantes del sistema. Primero mostramos el diagrama de casos de uso para después desarrollar algunos de manera más básica y otros de forma más extendida.

3.2.1. Diagrama casos uso

A través diagrama de casos de usos podemos obtener una primera aproximación de como interaccionan con el sistema los distintos actores que hemos descrito anteriormente.

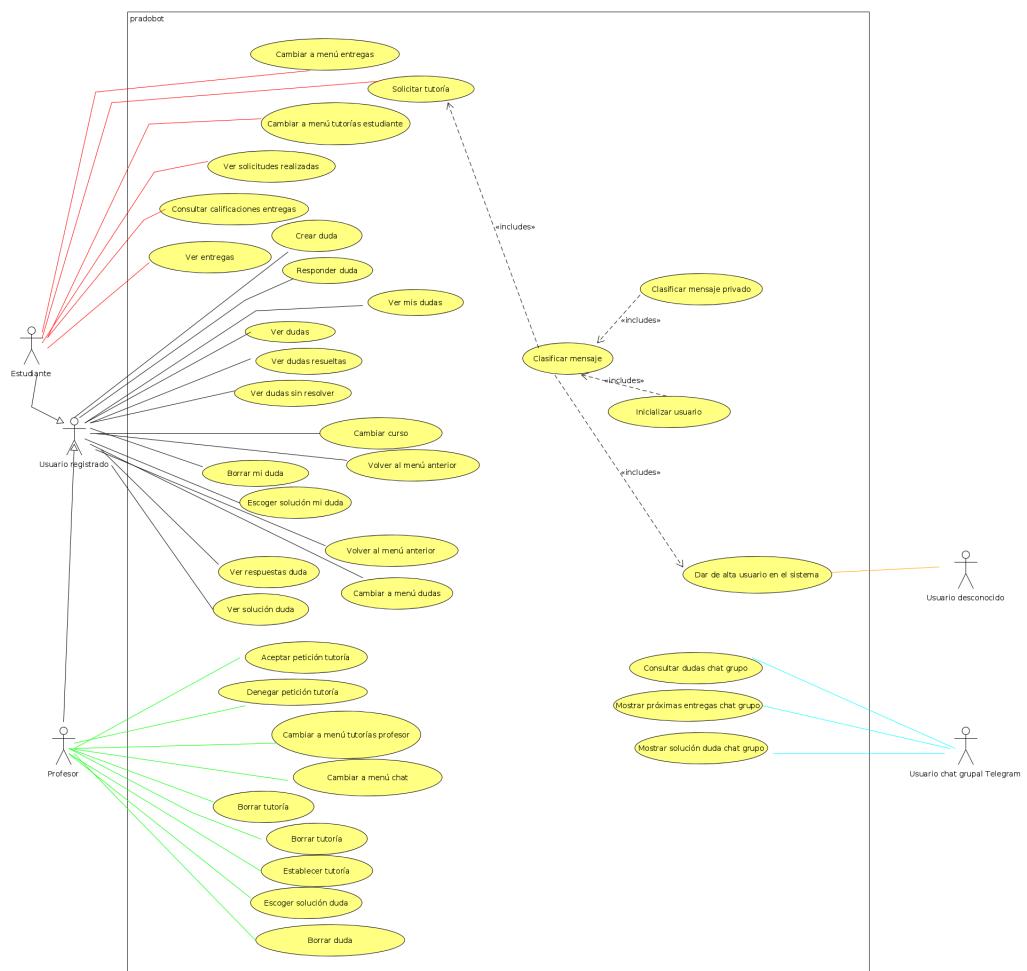


Figura 3.1: Diagrama de los casos de uso más relevantes del sistema

Podemos observar las funciones comunes que comparten todos los usu-

rios registrados del sistema, así como las que son específicas de cada rol. Por ejemplo, algo exclusivo del profesor es el establecimiento de las tutorías, mientras que algo único que puede hacer el estudiante es solicitarlas.

3.2.2. Descripción casos uso

He optado por realizar dos tipos de CUs unos más cercanos a los CU Reales, que muestran algunos detalles más cercanos al diseño y otros, bastante más abstractos, cuyo enfoque está en mostrar la interacción del usuario con el programa. El objetivo, es que haya equilibrio entre abstracción y las peculiaridades de un programa tipo bot, cuyo funcionamiento depende fuertemente del contexto en el que se recibe un mensaje del usuario.

CU-1.1	Clasificar mensaje.
Actores	Estudiante, Profesor
Tipo	Primario, Real
Referencias	CU-1.2 (Clasificar mensaje privado)
Precondición	Bot haya recibido un mensaje
Postcondición	
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito

Clasificar un mensaje recibido desde Telegram para determinar si procede de un chat individual o de uno grupal.

Resumen

El bot recibe un mensaje y lo clasifica dependiendo de si procede de un chat privado o de uno grupal.

Curso normal		
Actor	Sistema	
	1	El bot recibe el mensaje.
	2	El bot comprueba si el mensaje procede de un chat grupal o de uno privado.
	3	El bot clasifica el mensaje como privado.
	4	Incluir(CU-1.2, Clasificar mensaje privado)

Cuadro 3.1: Curso normal de CU-1.1. Clasificar mensaje.

Cursos alternos	
3a	El bot clasifica el mensaje como grupal Incluir(CU-1.4, Clasificar mensaje grupal) Fin CU.

Cuadro 3.2: Cursos alternos de CU-1.1. Clasificar mensaje.

CU-1.2	Clasificar mensaje privado
Actores	Estudiante, Profesor
Tipo	Primario, Real
Referencias	CU-1.3 (Inicializar usuario)
Precondición	El bot haya recibido un mensaje. El mensaje debe proceder desde un chat privado de Telegram
Postcondición	
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito
Clasificar un mensaje procedente de un chat privado

Resumen
El bot recibe un mensaje determinando el tipo de usuario que le manda el mensaje y si es la primera vez que recibe un mensaje de este usuario

Curso normal	
Actor	Sistema
	1 Obtiene todos los usuarios que han usado el bot desde que éste empezó a funcionar
	2 Obtiene al usuario del mensaje.
	3 Comprueba si el usuario que le ha mandado el mensaje está entre ellos.
	4 Comprueba si ha acabado de procesarse el último mensaje que envió el usuario.
	5 Obtiene el menú devuelto por el último menú utilizado por el usuario.
	6 Pasa el nuevo mensaje del usuario al menú devuelto.

Cuadro 3.3: Curso normal de CU-1.2. Clasificar mensaje privado.

Cursos alternos	
3a	El usuario no le ha mandado un mensaje al bot desde que éste empezó a funcionar . Incluir (CU-1.3, inicializar usuario) . Fin CU.
4a	La acción que ejecutó el usuario en su último mensaje no se ha acabado de ejecutar . Bot le pide que espere . Fin CU.

Cuadro 3.4: Cursos alternos de CU-1.2. Clasificar mensaje privado.

CU-1.3	Inicializar usuario
Actores	Estudiante, Profesor
Tipo	Primario, Real
Referencias	
Precondición	El usuario que le manda el mensaje al bot no le haya mandado un mensaje anteriormente desde que éste empezó a ejecutarse
Postcondición	El usuario será añadido a la lista de usuarios que están utilizando el bot
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17
Propósito	
Asignarle un menú de inicio al usuario que le ha mandado el mensaje	

Resumen	
El bot determina si el usuario está registrado en el sistema y le asigna un menú de inicio.	

Curso normal	
Actor	Sistema
	1 Bot comprueba si el usuario está registrado en el sistema.
	2 El bot obtiene una lista de cursos en los que se encuentra matriculado el usuario.
	3 El bot asigna como curso activo del usuario al primer curso de la lista.
	4 El bot obtiene el menú inicial del profesor.
	5 El bot le muestra al usuario el menú principal del profesor.

Cuadro 3.5: Curso normal de CU-1.3. Inicializar usuario.

Cursos alternos	
2a	El usuario no está registrado en el sistema. Obtiene el menú encargado de iniciar usuarios desconocidos. Muestra este menú al usuario. Fin CU.
4a	El usuario es un estudiante. Obtiene el menú inicial del estudiante y se lo muestra. Fin CU.

Cuadro 3.6: Cursos alternos de CU-1.3. Inicializar usuario.

CU-1.4	Cambiar curso.
Actores	Usuario Registrado
Tipo	Primario, Real
Referencias	CU-1.1 (Clasificar mensaje)
Precondición	
Postcondición	El usuario habrá cambiado el curso sobre el que actúan los mensajes que le envíe al bot.
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito	
Que un usuario que utilice el bot pueda variar el curso sobre el cual actúan sus mensajes	

Resumen	
El usuario selecciona en el menú que le esté mostrando el bot, que desea cambiar de curso. El bot le muestra los cursos accesibles para el usuario, el usuario escoge uno y el bot cambia el curso del usuario.	

Curso normal		
	Actor	Sistema
1	El usuario selecciona en el menú que tenga activo que desea cambiar de curso.	
		2 Incluir(CU-1.1, Clasificar mensaje)
		3 El bot detecta que el usuario ha pulsado cambiar de curso.
		4 El bot obtiene todos los cursos para los cuales está dado de alta el usuario.
		5 El bot muestra todos los cursos al usuario.
6	El usuario elige un curso .	
		7 Incluir(CU-1.1, Clasificar mensaje)
		8 El bot identifica al curso seleccionado por el usuario.
		9 El bot cambia el curso para el usuario.
		10 El bot informa al usuario del cambio de curso.

Cuadro 3.7: Curso normal de CU-1.4. Cambiar de curso.

CU-2	Dar de alta usuario en el sistema.
Actores	Usuario sin identificar
Tipo	Primario, Real
Referencias	CU-1.1 (Clasificar mensaje)
Precondición	El usuario no este registrado en el sistema.
Postcondición	Se habrá registrado un nuevo usuario en el sistema.
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito	
Para utilizar el bot es necesario que el usuario se identifique como usuario autorizado, para lo cual es necesario que pruebe que tiene acceso a los recursos de la instancia de Moodle.	

Resumen	
El usuario manda un mensaje al bot y éste le pide que se identifique introduciendo el email y contraseña que utiliza en Moodle, el usuario los introduce y el bot registra al usuario	

Curso normal	
Actor	Sistema
1 El usuario manda un mensaje al bot.	2 Incluir(CU-1.1, Clasificar mensaje)
	3 El bot le solicita que introduzca el email que utiliza en Moodle.
4 El usuario manda un mensaje al bot indicando su email.	5 Incluir(CU-1.1, Clasificar mensaje)
	6 El bot le solicita que introduzca la contraseña que utiliza para Moodle.
7 El usuario introduce la contraseña.	8 Incluir(CU-1.1, Clasificar mensaje privado)
	9 El bot solicita la token de usuario a la instancia de Moodle .
	10 El bot solicita los cursos accesibles para el usuario a la instancia de Moodle.
	11 El bot registra al usuario en el sistema.
	12 El bot establece como curso activo al primero de los obtenidos.
	13 El bot obtiene el menú principal del profesor .
	14 El bot establece el menú principal del profesor como menú a mostrar al usuario.
	15 El bot le muestra el menú principal al usuario.
	16 El bot le indica que que puede empezar a usarlo.

Cuadro 3.8: Curso normal de CU-2. Dar de alta usuario en el sistema.

Cursos alternos	
10a	Moodle le dice que los datos introducidos son erroneos . Bot manda mensaje de error al usuario. Fin CU.
11a	El usuario no puede acceder a ningún curso a través de la API de Moodle, bot manda mensaje de error . Fin CU.
13a	El usuario es un estudiante el bot obtiene el menú principal para los estudiantes. Se lo muestra al estudiante. Indica que puede empezar a utilizar el bot. Fin CU.

Cuadro 3.9: Cursos alternos de CU-2. Dar de alta usuario en el sistema.

CU-3.1	Cambiar a menú entregas
Actores	Estudiante
Tipo	Primario, Esencial
Referencias	
Precondición	Estudiante se encuentre en el menú principal del estudiante.
Postcondición	
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito
Que el estudiante pueda cambiar desde el menú principal del estudiante al de entregas.

Resumen
El estudiante le indica al bot que desea cambiar al menú de entregas y el bot lo cambia

Curso normal			
Actor		Sistema	
1	El estudiante selecciona al menú de entregas.		
		2	El bot muestra al estudiante el menú de entregas.

Cuadro 3.10: Curso normal de CU-3.1. Cambiar a menú entregas.

CU-3.2	Cambiar a menú tutorías
Actores	Estudiante
Tipo	Primario, Esencial
Referencias	
Precondición	Estudiante se encuentre en el menú principal del estudiante.
Postcondición	El estudiante ahora se encontrará en el menú de tutorías
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito

Que el estudiante pueda cambiar desde el menú principal del estudiante al menú de tutorías de estudiantes.

Resumen

El estudiante le indica al bot que desea cambiar al menú de tutorías y el bot lo cambia

Curso normal			
Actor		Sistema	
1	El estudiante selecciona al menú de tutorías.		
		2	El bot cambia al estudiante el menú de tutorías.
		3	El bot muestra al estudiante el menú de tutorías.

Cuadro 3.11: Curso normal de CU-3.2. Cambiar a menú tutorías.

CU-3.3	Cambiar a menú tutorías
Actores	Profesor
Tipo	Primario, Esencial
Referencias	
Precondición	Profesor esté en el menú principal de profesores.
Postcondición	El profesor estará en el menú de tutorías para profesores
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito
Que el profesor pueda cambiar desde el menú principal de los profesores al menú de tutorías para los profesores

Resumen
El profesor le indica al bot que lo cambie al menú de tutorías y el bot lo cambia

Curso normal		
	Actor	Sistema
1	El profesor selecciona el menú de tutorías.	
		2 El bot cambia al profesor al menú de tutorías.
		3 El bot muestra al profesor el menú de tutorías.

Cuadro 3.12: Curso normal de CU-3.3. Cambiar a menú tutorías.

CU-3.4	Cambiar a menú chat
Actores	Profesor
Tipo	Primario, Esencial
Referencias	
Precondición	Profesor esté en el menú principal de profesores.
Postcondición	El profesor estará en el menú de chat
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito

Que el profesor pueda cambiar desde el menú principal de los profesores al menú de chat de Telegram

Resumen

El profesor le indica al bot que desea ir al menú de chat y el bot lo cambia

Curso normal		
Actor		Sistema
1	El profesor selecciona el menú de chat.	
		2 El bot cambia al profesor al menú de chat.
		3 El bot muestra al profesor el menú de chat al profesor..

Cuadro 3.13: Curso normal de CU-3.4. Cambiar a menú chat.

CU-3.5	Cambiar a menú dudas
Actores	Usuario registrado
Tipo	Primario, Esencial
Referencias	
Precondición	Profesor esté en el menú principal de profesores.
Postcondición	El usuario se encontrará en el menú de dudas
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito
Que un usuario pueda acceder al menú de dudas.

Resumen
El usuario indica al bot que desea acceder al menú de dudas, el bot lo cambia y se lo muestra

Curso normal		
	Actor	Sistema
1	El usuario selecciona el menú de dudas.	
		2 El bot cambia al usuario al menú de dudas.
		3 El bot muestra al usuario el menú de dudas.

Cuadro 3.14: Curso normal de CU-3.5. Cambiar a menú dudas.

CU-3.6	Volver al menú anterior
Actores	Usuario registrado
Tipo	Primario, Esencial
Referencias	
Precondición	El usuario no se encuentre en su menú principal
Postcondición	El usuario se encontrará en el menú inmediatamente anterior que haya visitado
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito
Que un usuario pueda volver al menú anterior

Resumen
El usuario indica al bot que desea volver para atrás, el bot ve en qué menú está el usuario y lo cambia al menú anterior

Curso normal			
Actor		Sistema	
1	El usuario indica que quiere volver al menú anterior.		
		2	El bot obtiene el menú del usuario.
		3	El bot obtiene el menú anterior al que se encuentra el usuario.
		4	El bot cambia al usuario de menú .
		5	El bot muestra al usuario el menú al que acaba de cambiarlo .

Cuadro 3.15: Curso normal de CU-3.6. Volver al menú anterior.

CU-4.1	Establecer tutoría
Actores	Profesor
Tipo	Primario, Real
Referencias	CU.1.1
Precondición	El menú activo para el profesor sea el menú principal del profesor
Postcondición	Se habrá creado una nueva tutoría.
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito
Que un profesor pueda crear una nueva tutoría

Resumen
El profesor selecciona el menú de tutorías, el bot muestra al profesor el menú de tutorías, el profesor selecciona que desea crear una nueva tutoría, el bot le pide que introduzca el día de la semana de la tutoría y la hora, el profesor lo introduce y el bot registra la nueva tutoría.

Curso normal		
	Actor	Sistema
1	El profesor selecciona el menú de tutorías.	
		2 Incluir(CU-1.1, Clasificar mensaje)
		3 El bot detecta que el profesor ha seleccionado el menú de tutorías.
		4 El bot muestra al profesor las opciones del menú de tutorias.
		5 El profesor selecciona la opción de crear una tutoría.
		6 Incluir(CU-1.1, Clasificar mensaje)
		7 El bot detecta que el profesor ha seleccionado la opción de crear una tutoría.
		8 El bot solicita al profesor que introduzca el día de la semana en la cual quiere establecer la nueva tutoría.
9	El profesor manda un mensaje indicando el día de la semana elegido.	
		10 Incluir(CU-1.1, Clasificar mensaje)
		11 El bot obtiene la última opción pulsada para el menú de tutorías del profesor.
		12 El bot obtiene el último paso realizado para la opción de crear tutorías.
		13 El bot almacena el día introducido.
		14 El bot solicita que introduzca la hora.
15	El profesor introduce la hora.	
		16 Incluir(CU-1.1, Clasificar mensaje)
		17 El bot obtiene la última opción pulsada para el menú de tutorías del profesor.
		18 El bot obtiene el último paso realizado para la opción de crear tutorías.
		19 El bot comprueba la hora introducida.
		20 El bot crea una nueva tutoría con la fecha y hora introducidos.
		21 El bot notifica al usuario que se ha creado con éxito la tutoría.

Cuadro 3.16: Curso normal de CU-4.1 Establecer tutoría.

Cursos alternos	
10a	Introduce un día inexistente . Bot manda mensaje de error al usuario. Fin CU.
17a	El usuario introduce una hora inválida bot manda mensaje de error . Fin CU.

Cuadro 3.17: Cursos alternos de CU-4.1. Establecer tutoria.

CU-4.2	Borrar tutoría
Actores	Profesor
Tipo	Primario, Real
Referencias	CU-1.1
Precondición	El menú activo para el profesor sea el menú principal del profesor
Postcondición	Se habrá creado una nueva tutoría.
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito
Que un profesor pueda borrar una tutoría creada por él.

Resumen
Desde el menú de tutorías el profesor selecciona la opción de borrar tutorías, el bot le muestra las tutorías que ha creado, el profesor selecciona una y el bot procede a borrarlas.

Curso normal		
	Actor	Sistema
1	El profesor selecciona el menú de tutorías.	2 Incluir(CU-1.1, Clasificar mensaje)
		3 El bot detecta que el profesor ha seleccionado el menú de tutorías.
		4 El bot muestra al profesor las opciones del menú de tutorias.
5	El profesor selecciona la opción de borrar tutorías.	6 Incluir(CU-1.1, Clasificar mensaje)
		7 El bot detecta que el profesor ha seleccionado la opción de borrar tutoría.
		8 El bot obtiene las tutorias creadas por el profesor.
		9 El bot muestra las tutorías al profesor.
10	El profesor elige una.	11 Incluir(CU-1.1, Clasificar mensaje)
		11 El bot obtiene la última opción pulsada para el menú de tutorías del profesor.
		12 El bot obtiene el último paso realizado para la opción de borrar tutorías.
		13 El bot obtiene del mensaje la tutoría elegida por el profesor.
		14 El bot procede a borrar la tutoría elegida por el profesor.
		15 El indica al profesor que ha borrado la tutoría elegida.

Cuadro 3.18: Curso normal de CU-4.2. Borrar tutoría

Cursos alternos	
9a	El profesor no ha creado ninguna tutoría . Bot manda mensaje de error al profesor. Fin CU.

Cuadro 3.19: Cursos alternos de CU-4.2. Borrar tutoría.

CU-4.3	Solicitar tutoría
Actores	Profesor
Tipo	Primario, Real
Referencias	
Precondición	El menú activo para el estudiante sea el menú principal del estudiante
Postcondición	Se habrá creado una nueva solicitud para la tutoría elegida por el estudiante.
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito
Que un estudiante pueda solicitar asistir a una tutoría de un profesor

Resumen
El estudiante selecciona la opción de realizar petición tutoría en el menú de tutorías para los estudiantes, el bot le muestra las tutorías creadas por los profesores de los cursos en los que está registrado, el estudiante selecciona una y el bot registra la selección.

Curso normal		
	Actor	Sistema
1	El estudiante selecciona el menú de tutorías desde su menú principal.	
		2 Incluir(CU-1.1, Clasificar mensaje)
		3 El bot detecta la selección de menú del estudiante.
		4 El bot muestra al estudiante el menú de tutorías de los estudiantes.
5	El estudiante selecciona la opción de realizar petición tutoría.	
		6 Incluir(CU-1.1, Clasificar mensaje)
		7 El bot detecta la opción elegida para el último menú mostrado al estudiante.
		8 El bot obtiene los cursos en los cuales se encuentra registrado el estudiante.
		9 El bot obtiene las tutorías creadas por los profesores responsables de los cursos obtenidos.
		9 El bot muestra las tutorías al estudiante.
10	El estudiante selecciona una.	
		11 Incluir(CU-1.1, Clasificar mensaje)
		12 El bot detecta la opción elegida para el último menú mostrado al estudiante.
		12 El bot obtiene la tutoría escogida por el estudiante del mensaje.
		13 El bot crea una petición para la tutoría escogida.
		14 El bot le indica al estudiante que su petición se ha registrado correctamente.

Cuadro 3.20: Curso normal de CU-4.3. Solicitar tutoría.

Cursos alternos	
9a	No hay tutorías creadas . Bot manda mensaje de error al profesor. Fin CU.

Cuadro 3.21: Cursos alternos de CU-4.3. Solicitar tutoría.

CU-4.4	Aceptar petición tutoría
Actores	Profesor
Tipo	Primario, Esencial
Referencias	
Precondición	El menú activo para el profesor sea el menú de tutorías del profesor
Postcondición	Una petición para una tutoría del profesor habrá sido aceptada.
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito
Que un profesor pueda aceptar las peticiones de asistencia que se hacen a sus tutorías

Resumen
El profesor indica al bot que desea ver las peticiones que se han hecho a sus tutorías, bot muestra tutorías, profesor selecciona una, bot muestra las peticiones, profesor selecciona una y le indica al bot que la acepte

Curso normal		
	Actor	Sistema
1	El profesor indica que desea ver las peticiones realizadas a una de sus tutorías	
		2 El bot obtiene las tutorías del profesor.
		3 El bot muestra las tutorías al profesor.
4	El profesor selecciona una de las tutorías.	
5	El bot obtiene las peticiones realizadas para la tutoría elegida.	
		6 El bot muestra las peticiones al profesor.
7	Profesor selecciona una petición.	
8	Profesor indica que desea aceptar la petición elegida.	
		9 El bot cambia el estado de la petición elegida a aprobada.

Cuadro 3.22: Curso normal de CU-4.4 Aceptar petición tutoría.

Cursos alternos	
3a	No hay tutorías creadas . Bot manda mensaje de error al profesor. Fin CU.

Cuadro 3.23: Cursos alternos de CU-4.4. Aceptar petición tutoría.

CU-4.5	Denegar petición tutoría
Actores	Profesor
Tipo	Primario, Esencial
Referencias	
Precondición	El menú activo para el profesor sea el menú de tutorías del profesor
Postcondición	Se habrá denegado una petición a una tutoría creada por el profesor.
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito
Que un profesor pueda rechazar alguna de las peticiones de asistencia que recibe una de sus tutorías

Resumen
El profesor indica al bot que desea ver las peticiones que se han hecho a sus tutorías, bot muestra tutorías, profesor selecciona una, bot muestra las peticiones, profesor selecciona una y le indica al bot que la rechaza, bot la marca como rechazada

Curso normal		
	Actor	Sistema
1	El profesor indica que desea ver las peticiones realizadas a una de sus tutorías	
		2 El bot obtiene las tutorías del profesor.
		3 El bot muestra las tutorías al profesor.
4	El profesor selecciona una de las tutorías.	
5	El bot obtiene las peticiones realizadas para la tutoría elegida.	
		6 El bot muestra las peticiones al profesor.
7	Profesor selecciona una petición.	
8	Profesor indica que desea rechazar la petición elegida.	
		9 El bot cambia el estado de la petición elegida a rechazada.

Cuadro 3.24: Curso normal de CU-4.5 Denegar petición tutoría.

Cursos alternos	
3a	No hay tutorías creadas . Bot manda mensaje de error al profesor. Fin CU.

Cuadro 3.25: Cursos alternos de CU-4.5 Denegar petición tutoría.

Propósito
Permitir a un usuario registrado en el sistema poder crear una duda asociada a un curso

CU-4.6	Ver información tutoría
Actores	Profesor
Tipo	Primario, Esencial
Referencias	
Precondición	El menú activo para el profesor sea el menú de tutorías del profesor
Postcondición	U
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito	
Que un profesor pueda ver la cola de asistencia a una de sus tutorías	

Resumen	
El profesor indica al bot que le muestre información acerca de una tutoría, el bot obtiene las peticiones realizadas a esa tutoría, el bot obtiene la fecha de la tutoría y se la manda al profesor	

Curso normal		
	Actor	Sistema
1	El profesor indica que desea ver que tutorías tiene creadas.	
		2 El bot obtiene las tutorías del profesor.
		3 El bot muestra las tutorías al profesor.
4	El profesor indica que quiere ver la cola para una tutoría.	
5	El bot obtiene las peticiones aceptadas para la tutoría elegida.	
		6 El bot muestra las peticiones al profesor.

Cuadro 3.26: Curso normal de CU-4.6 Ver información tutoría.

CU-5.2	Ver dudas sin solución
Actores	Usuario registrado
Tipo	Primario, Esencial
Referencias	
Precondición	El usuario se encuentre en el menú de dudas.
Postcondición	
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Cursos alternos	
3a	No hay tutorías creadas . Bot manda mensaje de error al profesor. Fin CU.
5a	No hay peticiones aceptadas . Bot manda mensaje de error al profesor. Fin CU.

Cuadro 3.27: Cursos alternos de CU-4.6 Ver información tutoría.

CU-4.7	Ver solicitudes realizadas
Actores	Estudiante
Tipo	Primario, Esencial
Referencias	
Precondición	Estudiante se encuentre en el menú de tutorías para estudiante.
Postcondición	
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito
Permitir a un estudiante ver las peticiones a tutorías que ha realizado hasta el momento

Resumen
El estudiante le indica al bot que desea conocer las peticiones a tutorías que ha realizado hasta el momento y éste se las muestra

Curso normal		
	Actor	Sistema
3	El estudiante indica al bot que desea conocer el estado de las peticiones que ha realizado a tutorías.	
		2 El bot obtiene todas las peticiones que ha realizado el estudiante.
3	El bot le muestra al estudiante todas las peticiones que ha realizado hasta el momento junto con el estado de dichas peticiones.	

Cuadro 3.28: Curso normal de CU-4.7. Ver solicitudes a tutoría realizadas.

Cursos alternos	
2.a	El estudiante no ha realizado ninguna petición. Muestra mensaje error indicando esto. FIN CU.

Cuadro 3.29: Cursos alternos de CU-4.7 Ver solicitudes a tutoría realizadas.

CU-5.1	Crear duda
Actores	Usuario registrado
Tipo	Primario, Esencial
Referencias	
Precondición	El usuario registrado se encuentre en su menú principal
Postcondición	Una nueva duda habrá sido añadida al sistema
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Resumen	
	El usuario registrado indica al bot que quiere crear una nueva duda , éste le dice que introduzca la duda y el bot la crea tras ser introducida por el usuario

Curso normal		
	Actor	Sistema
1	El usuario registrado indica que quiere crear una nueva duda.	
		2 El bot le manda un mensaje indicando que introduzca la duda.
3	El usuario registrado introduce el texto de la duda.	
		4 El bot le pide que confirme que si de verdad quiere crear la duda.
5	El usuario registrado le manda un mensaje diciendo que si la quiere crear.	
		6 El bot crea la duda.
		7 El bot manda mensaje confirmando el correcto registro de la duda.

Cuadro 3.30: Curso normal de CU-5.1. Crear duda.

Cursos alternos	
5.a	El usuario indica que no. Bot no crea la duda . FIN CU.

Cuadro 3.31: Cursos alternos de CU-5.1 Crear duda.

Propósito	
	Permitir a un usuario conocer todas aquellas dudas que aún no hayan sido resueltas

Resumen	
	El usuario indica que quiere conocer aquellas dudas pendientes de solucionarse para el curso en el que está, el bot obtiene las dudas sin solución y se las manda al usuario.

Curso normal		
	Actor	Sistema
1	El usuario registrado indica al bot que le muestre las dudas sin solución.	
		2 El bot obtiene todas las dudas sin solución que tiene el curso en el que se encuentra el usuario registrado.
3	El bot muestra al usuario registrado las dudas.	

Cuadro 3.32: Curso normal de CU-5.2. Ver dudas.

Cursos alternos	
2.a	No hay dudas sin solución para el curso activo. Bot indica que no hay dudas al usuario . FIN CU.

Cuadro 3.33: Cursos alternos de CU-5.2 Ver dudas.

Propósito	
	Permitir a un usuario ver las dudas que él ha creado

CU-5.3	Ver mis dudas
Actores	Usuario registrado
Tipo	Primario, Esencial
Referencias	
Precondición	El usuario se encuentre en el menú de dudas.
Postcondición	
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Resumen

El usuario indica que quiere ver las dudas que ha creado, el bot las obtiene y se las muestra

Curso normal		
	Actor	Sistema
1	El usuario registrado solicita al bot que le muestre sus dudas.	
		2 El bot obtiene todas las dudas del usuario para el curso concreto en el que está ahora mismo.
3	El bot muestra al usuario registrado las dudas.	

Cuadro 3.34: Curso normal de CU-5.3. Ver mis dudas.

Cursos alternos	
2.a	El usuario no ha creado ninguna duda para el curso activo. Bot indica que no hay dudas al usuario . FIN CU.

Cuadro 3.35: Cursos alternos de CU-5.3 Ver mis dudas.

Propósito
Permitir a un usuario ver las dudas que tienen solución para el curso en el que se encuentre

CU-5.4	Ver dudas resueltas
Actores	Usuario registrado
Tipo	Primario, Esencial
Referencias	
Precondición	El usuario se encuentre en el menú de dudas.
Postcondición	
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Resumen	
El usuario indica que quiere aquellas dudas solucionadas para el curso en el que está	

Curso normal		
	Actor	Sistema
1	El usuario registrado solicita al bot que le muestre aquellas dudas que tienen solución para el curso en el que está.	
		2 El bot obtiene todas las dudas con solución para el curso concreto en el que está ahora mismo.
3	El bot muestra al usuario registrado las dudas.	

Cuadro 3.36: Curso normal de CU-5.4. Ver dudas resueltas.

Cursos alternos	
2.a	No hay dudas con solución para el curso activo. Bot indica que no hay dudas al usuario . FIN CU.

Cuadro 3.37: Cursos alternos de CU-5.4 Ver dudas resueltas.

Propósito	
Permitir a un usuario registrado responder a las dudas planteadas en un curso	

CU-5.5	Crear respuesta a duda
Actores	Usuario registrado
Tipo	Primario, Esencial
Referencias	
Precondición	El usuario se encuentre en el menú de dudas.
Postcondición	Se habrá añadido una nueva respuesta asociada a la duda elegida.
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Resumen

El usuario registrado indica al bot que quiere añadir una respuesta a la duda seleccionada, el bot le pide que introduzca su respuesta y ésta se almacena.

Curso normal		
	Actor	Sistema
1	El usuario registrado indica que quiere responder una duda.	
2	El usuario envía la duda al bot.	
		3 El bot le pide que introduzca la respuesta
4	El usuario registrado introduce la respuesta.	
		5 El bot asocia la respuesta introducida con la duda enviada por el usuario.
		6 El bot confirma al usuario que se ha almacenado la respuesta correctamente.

Cuadro 3.38: Curso normal de CU-5.5. Ver dudas resueltas

CU-5.6	Ver respuestas duda
Actores	Usuario registrado
Tipo	Primario, Esencial
Referencias	
Precondición	El usuario se encuentre en el menú de dudas.
Postcondición	
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito

Permitir a un usuario registrado conocer las respuestas asociadas a una duda.

Resumen

El usuario registrado indica que quiere obtener las respuestas asociadas a una duda, el bot las obtiene y se las envía al usuario.

Curso normal		
	Actor	Sistema
1	El usuario indica que quiere conocer las respuestas a una duda.	
		2 El bot solicita la duda.
3	El usuario introduce la duda.	
		4 El bot solicita la respuesta.
5	El usuario introduce la respuesta.	
		6 El bot guarda la respuesta.
		7 El bot asocia la respuesta con la duda.

Cuadro 3.39: Curso normal de CU-5.6. Ver respuestas duda.

Cursos alternos	
3.a	No hay respuestas para la duda escogida. Bot manda mensaje al usuario indicando esto . FIN CU.

Cuadro 3.40: Cursos alternos de CU-5.6 Ver respuestas duda.

CU-5.7	Ver solución a duda
Actores	Usuario registrado
Tipo	Primario, Esencial
Referencias	
Precondición	El usuario se encuentre en el menú de dudas.
Postcondición	
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito	
Permitir a un usuario registrado ver la solución de una duda	

Resumen	
El usuario indica que quiere conocer la solución a una duda, el bot obtiene la solución y se la muestra	

Curso normal			
Actor		Sistema	
1	El usuario registrado indica que quiere ver la solución de una duda.		
		2	El bot solicita la duda al usuario
3	El usuario envía la duda al bot.		
		4	El bot obtiene la solución a la duda.
		5	El bot envía la solución al usuario.

Cuadro 3.41: Curso normal de CU-5.7. Ver solución a duda.

Cursos alternos	
4.a	La duda escogida no tiene solución asociada. Bot manda mensaje indicando esto.

Cuadro 3.42: Cursos alternos de CU-5.7 Ver solución a duda.

CU-5.8	Borrar mi duda.
Actores	Usuario registrado
Tipo	Primario, Esencial
Referencias	
Precondición	El usuario se encuentre en el menú de dudas.
Postcondición	
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito
Permitir a un usuario eliminar una duda que haya creado.

Resumen
El usuario indica al bot que desea eliminar la duda que ha seleccionado, el bot le pide que lo confirme y procede a borrar la duda

Curso normal			
Actor		Sistema	
1	El usuario le indica al bot que quiere eliminar una duda que ha creado.		
2		El bot le pide que introduzca la duda	
3	El usuario introduce la duda.		
4		El bot le pide que confirme que desea borrar la duda.	
5	El usuario le dice que está seguro de que desea eliminar la duda.		

Cuadro 3.43: Curso normal de CU-5.8. Borra mi duda.

Cursos alternos	
5.a	El usuario dice que no está seguro Bot no borra la duda. Fin CU.

Cuadro 3.44: Cursos alternos de CU-5.8 Borrar mi duda.

CU-5.9	Elegir solución mi duda.
Actores	Usuario registrado
Tipo	Primario, Esencial
Referencias	
Precondición	El usuario se encuentre en el menú de dudas.
Postcondición	
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito
Permitir que un usuario pueda elegir entre las respuestas a una duda la que considera que resuelve la misma.

Resumen
El usuario registrado indica al bot que quiere seleccionar una de las respuestas que ha tenido su duda como solución a ésta, le indica la respuesta y la duda al bot y éste la asocia como solución a la duda.

Curso normal			
Actor		Sistema	
1	El usuario indica al bot que quiere asociar una respuesta como solución a una duda que ha creado.		
		2	El bot le solicita que introduzca la respuesta y la duda.
3	El usuario introduce la respuesta y la duda.		
		4	El bot asocia la respuesta como solución a la duda.
		5	El bot asocia la respuesta escogida como solución a la duda seleccionada.

Cuadro 3.45: Curso normal de CU-5.9. Elegir solución mi duda..

Propósito
Permitir a un profesor borrar una duda del curso

CU-5.10	Borrar duda
Actores	Profesor
Tipo	Primario, Esencial
Referencias	
Precondición	El profesor se encuentre en el menú de dudas para los usuarios registrados.
Postcondición	Se habrá borrado una duda del sistema
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Resumen	
El profesor indica al bot que desea borrar la duda seleccionada, el bot confirma el deseo de borrar la duda y procede a borrarla.	

Curso normal			
Actor		Sistema	
1	El profesor indica al bot que quiere que borre una duda .		
		2	El bot le solicita que introduzca la duda
3	El profesor introduce la duda.		
		4	El bot le pregunta si está si esta seguro.
5	El profesor indica que si.		
		6	La duda se borra del sistema.

Cuadro 3.46: Curso normal de CU-5.10. Borrar duda.

Cursos alternos	
5.a	El profesor dice que no está seguro. Bot no borra la duda. Fin CU.

Cuadro 3.47: Cursos alternos de CU-5.9 Borrar duda.

Propósito	
Permitir a un profesor establecer una respuesta como solución a cualquier duda planteada en el curso.	

CU-5.11	Elegir solución a duda
Actores	Profesor
Tipo	Primario, Esencial
Referencias	
Precondición	El profesor se encuentre en el menú de dudas
Postcondición	La duda seleccionada por el profesor tendrá asociada una respuesta como solución
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Resumen

El profesor indica al bot que desea establecer una respuesta como solución a la duda elegida, el bot le proporciona las respuestas, el profesor escoge la que será la solución a la duda y el bot la asocia como solución a la duda.

Curso normal			
Actor		Sistema	
1	El profesor indica al bot que desea escoger una respuesta como solución a una duda.		
		2	El bot le solicita que introduzca la duda.
3	El profesor introduce la duda.		
		4	El bot le solicita que introduzca la respuesta.
5	El profesor proporciona al bot la respuesta que quiere que sea solución a la duda seleccionada.		
		6	El bot asocia la respuesta como solución a la duda.

Cuadro 3.48: Curso normal de CU-5.11. Elegir solución a duda.

CU-6.1	Ver entregas
Actores	Estudiante
Tipo	Primario, Esencial
Referencias	
Precondición	El estudiante se encuentra en el menú de entregas.
Postcondición	
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito
Permitir a un estudiante consultar información acerca de las entregas del curso.

Resumen
El estudiante indica que quiere ver qué entregas hay para el curso en el que está, el bot le muestra las entregas, el estudiante selecciona una y el bot le proporciona más información acerca de ésta

Curso normal		
	Actor	Sistema
1	El estudiante le indica al bot que le muestre las entregas que hay para el curso.	
		2 El bot obtiene el curso activo del estudiante.
		3 El bot obtiene las entregas abiertas del curso.
		4 El bot muestra las entregas al estudiante.
5	El estudiante solicita información acerca de una.	
		6 El bot obtiene información adicional de la entrega elegida.
		7 El bot muestra la información recopilada al estudiante.
8	El usuario registrado le manda un mensaje diciendo que sí la quiere crear.	

Cuadro 3.49: Curso normal de CU-6.1. Ver entregas.

Cursos alternos	
4.a	No hay entregas abiertas para el curso. Bot manda mensaje error . FIN CU.

Cuadro 3.50: Cursos alternos de CU-6.1 Ver entregas.

CU-6.2	Consultar calificaciones
Actores	Estudiante
Tipo	Primario, Esencial
Referencias	
Precondición	El estudiante se encuentra en el menú de entregas.
Postcondición	
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito
Permitir a un estudiante consultar las calificaciones de sus entregas

Resumen
El estudiante expresa que quiere conocer las notas que tiene para las entregas realizadas en el curso, bot obtiene las notas de las entregas y se las muestra al estudiante

Curso normal		
	Actor	Sistema
1	El estudiante indica al bot que le muestre qué calificaciones tienen las entregas que ha realizado.	
		2 El bot obtiene el curso activo del estudiante.
		3 El bot obtiene las entregas del curso realizadas por el estudiante.
		4 El bot muestra las notas de las entregas al estudiante.

Cuadro 3.51: Curso normal de CU-6.2. Consultar calificaciones.

Cursos alternos	
4.a	No hay entregas abiertas para el curso. Bot manda mensaje error . FIN CU.

Cuadro 3.52: Cursos alternos de CU-6.2 Consultar calificaciones.

CU-6.3	Consultar dudas chat grupo.
Actores	Usuario chat grupal Telegram
Tipo	Primario, Esencial
Referencias	
Precondición	El usuario se encuentre en un chat de Telegram asociado con un curso para el cual el profesor responsable ha establecido ese chat como el chat del curso. El bot tiene que encontrarse en ese chat.
Postcondición	
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito
Permitir a un usuario de un chat grupal mostrar las dudas resueltas de un curso por el chat asociado a éste.

Resumen	
El usuario manda el comando asociado a mostrar dudas resueltas al bot por el chat grupal, éste las obtiene y las muestra.	

Curso normal	
Actor	Sistema
1 El usuario manda el comando asociado a mostrar dudas resueltas por el chat grupal.	
	2 El bot obtiene el curso asociado al chat grupal
	3 El bot obtiene las dudas resueltas asociadas al curso.
	4 El bot muestra las dudas por el chat grupal.

Cuadro 3.53: Curso normal de CU-6.3. Consultar dudas chat grupo.

Cursos alternos	
3.a	No hay dudas resueltas para el curso. Indica que no hay dudas resueltas por el chat grupal Fin CU.

Cuadro 3.54: Cursos alternos de CU-6.3. Consultar dudas chat grupo.

CU-7.1	Mostrar solución duda chat grupal.
Actores	Usuario chat grupal Telegram
Tipo	Primario, Esencial
Referencias	
Precondición	Que el bot se encuentre en el chat grupal. Que la duda indicada tenga solución.
Postcondición	
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito

Poder mostrar la solución a una duda resuelta para un curso dado de alta en el bot por el chat grupal que tiene asociado.

Resumen

El usuario manda el comando por el chat grupal indicándose en este comando la duda cuya solución quiere que se muestre, el bot obtiene la solución a la duda y la muestra

Curso normal		
	Actor	Sistema
1	El usuario manda por el chat grupal el comando que le indica al bot que muestre la solución para una duda que se indica también en este comando.	
		2 El bot obtiene el curso asociado al chat grupal
		3 El bot obtiene las dudas resueltas asociadas al curso.
		4 El bot obtiene la solución a la duda.
		5 El bot muestra la duda junto con su solución por el chat grupal.

Cuadro 3.55: Curso normal de CU-7.1. Mostrar solución duda chat grupal.

CU-7.2	Mostrar próximas entregas chat grupal.
Actores	Usuario chat grupal Telegram
Tipo	Primario, Esencial
Referencias	
Precondición	Que el bot se encuentre en el chat grupal.
Postcondición	
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito
Un usuario de un chat grupal en el que se encuentra el bot pueda solicitar que éste muestre las próximas del curso asociado a éste en el chat.

Resumen
El usuario manda un comando por el chat grupal que le indica al bot que muestre las próximas entregas del curso, el bot obtiene las entregas y las muestra por el chat.

Curso normal		
	Actor	Sistema
1	El usuario solicita por el chat al bot que muestre las entregas.	
		2 El bot obtiene el curso asociado al chat grupal
		3 El bot obtiene las próximas entregas del curso.
		4 El bot muestra por el chat grupal las fechas de las entregas obtenidas.

Cuadro 3.56: Curso normal de CU-7.2. Mostrar próximas entregas chat grupal.

CU-7.3	Mostrar dudas resueltas.
Actores	Usuario chat grupal Telegram
Tipo	Primario, Esencial
Referencias	
Precondición	Que el bot se encuentre en el chat grupal.
Postcondición	
Autor	Luis Gil Guijarro
Versión	1.0
Fecha	10-05-17

Propósito
Mostrar las dudas resueltas de un curso por su chat de Telegram asociado.

Resumen
El usuario manda un comando por el chat grupal que le indica al bot que muestre las próximas entregas del curso, el bot obtiene las entregas y las muestra por el chat.

Curso normal		
	Actor	Sistema
1	El usuario solicita por el chat al bot que muestre las dudas resueltas del curso.	
		2 El bot obtiene el curso asociado al chat grupal
		3 El bot obtiene las dudas resueltas del curso.
		4 El bot muestra por el chat grupal las dudas.

Cuadro 3.57: Curso normal de CU-7.3. Mostrar dudas resueltas.

3.2.3. Diagrama de paquetes

En el diagrama de paquetes siguiente podemos ver un esbozo de la estructura lógica del sistema.

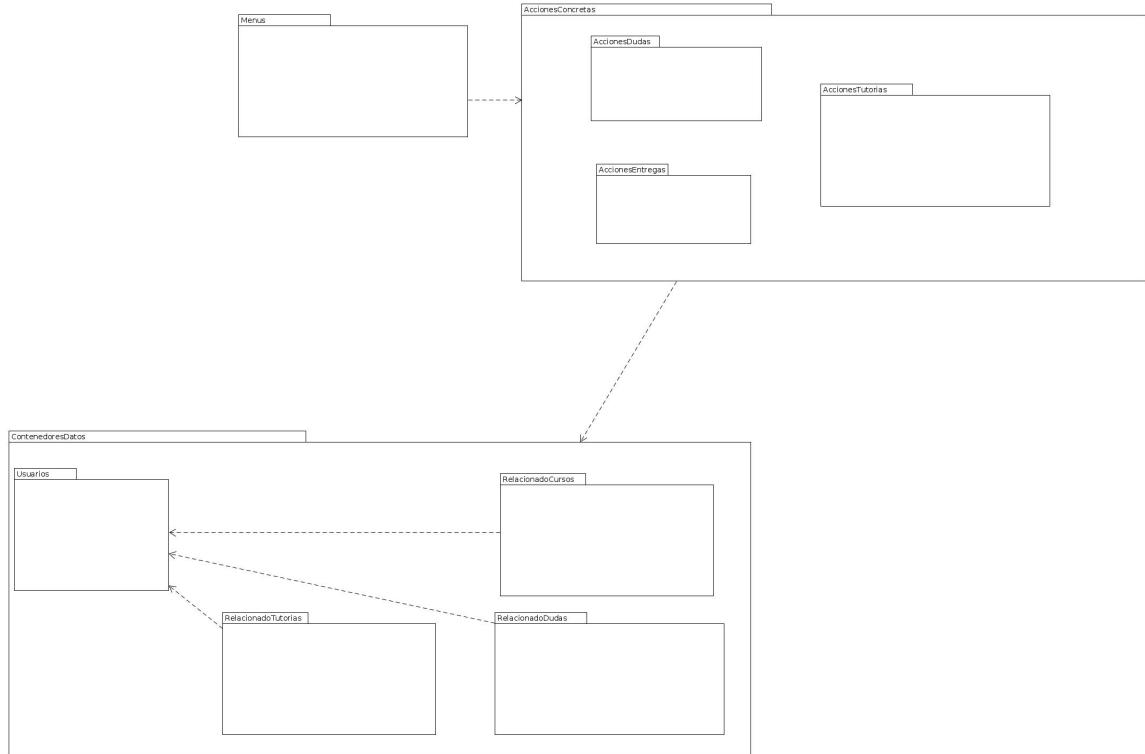


Figura 3.2: Diagrama de los casos de uso más relevantes del sistema

3.2.4. Diagramas de actividad

A continuación mostramos los diagramas de actividad para representar el flujo de procesos contenidos en los CU.

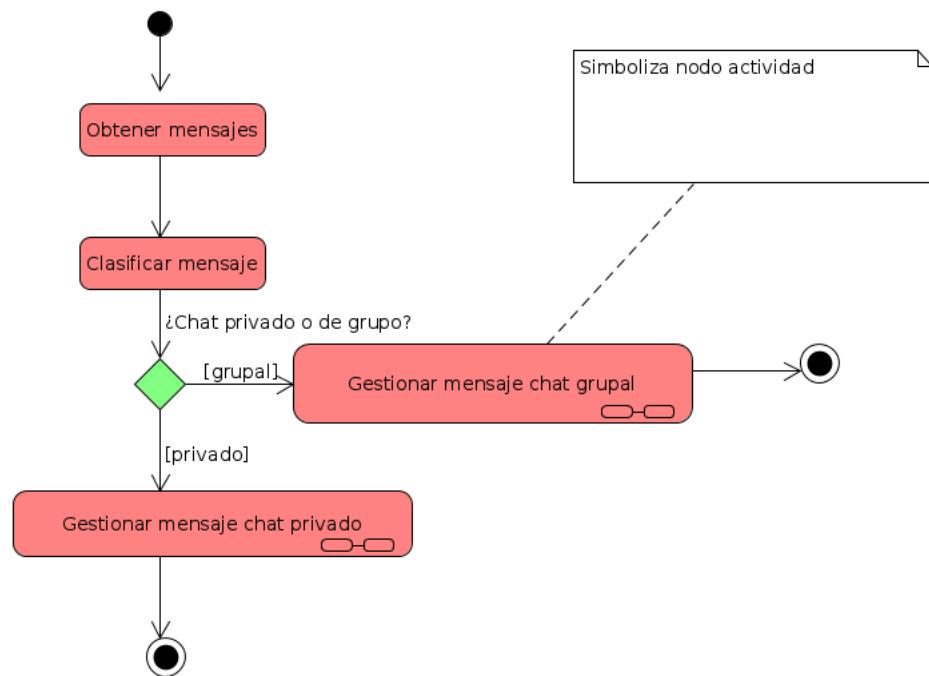


Figura 3.3: Diagrama actividad CU-1.1 Clasificar mensaje

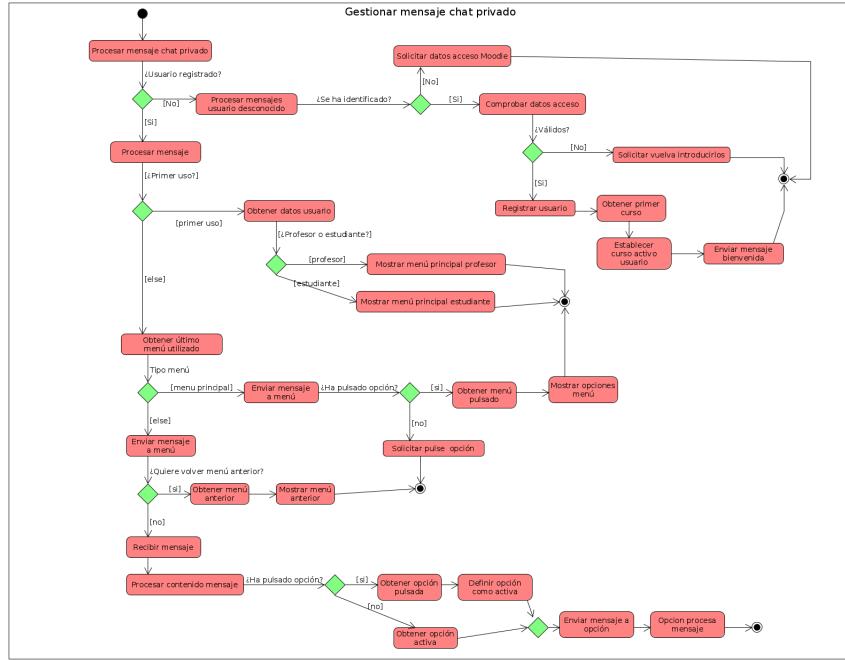


Figura 3.4: Diagrama actividad CU-1.2 Clasificar mensaje privado

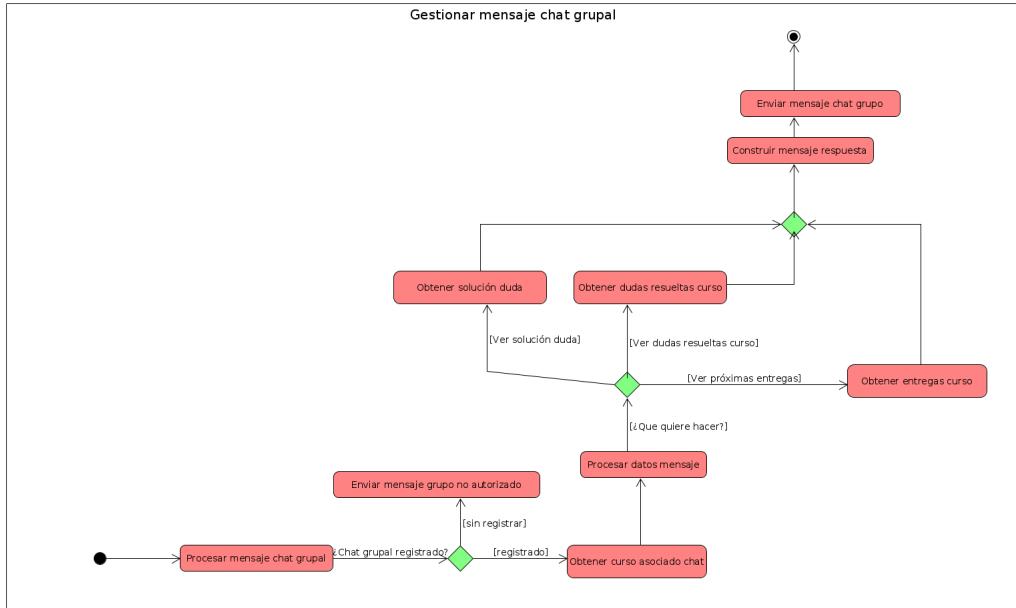


Figura 3.5: Diagrama actividad CU-7.1, CU-7.2, CU-7.3 Mostrar solución duda, mostrar dudas resueltas y entregas por chat grupal

A continuación mostramos los diagramas de actividad de los CU relacionados con las tutorías.

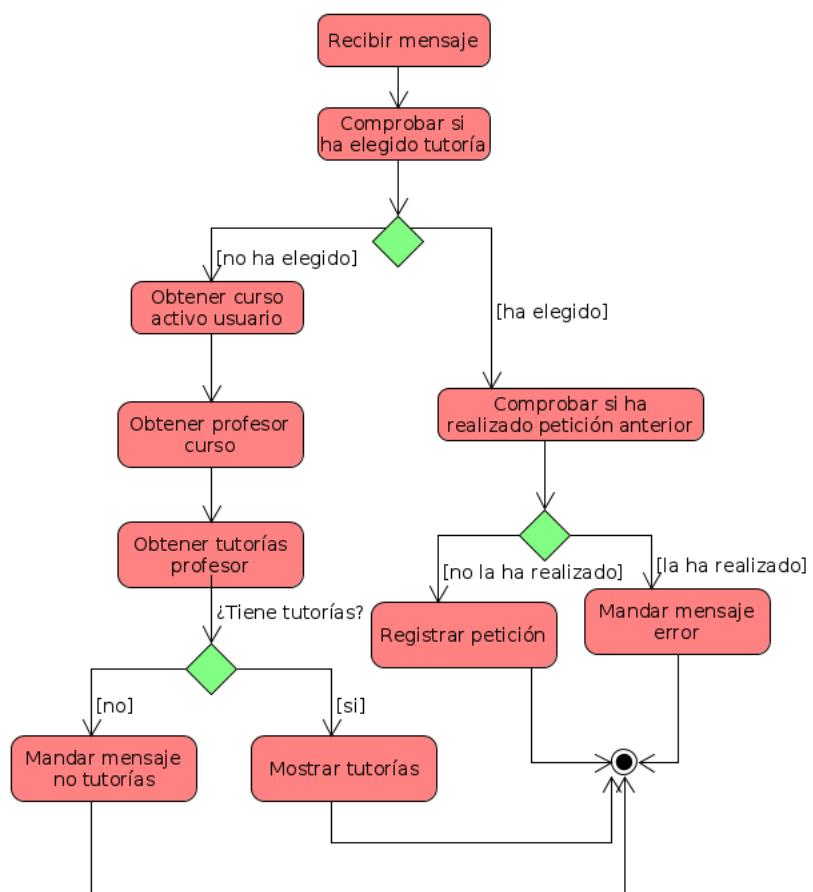


Figura 3.6: Diagrama actividad de CU-4.3 Solicitar tutoría

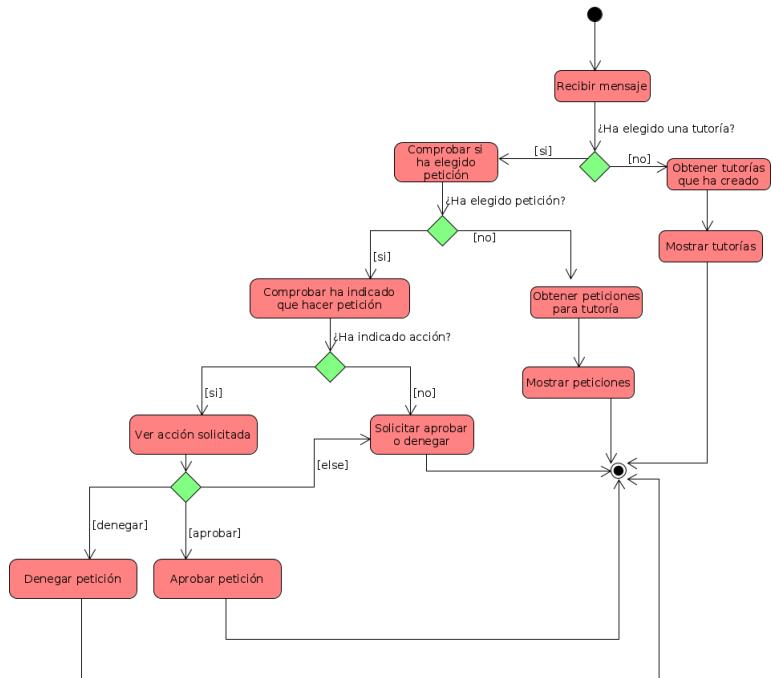


Figura 3.7: Diagrama actividad conjunto de CU-4.4 Aceptar petición tutoría, CU-4.5 Denegar petición tutoría

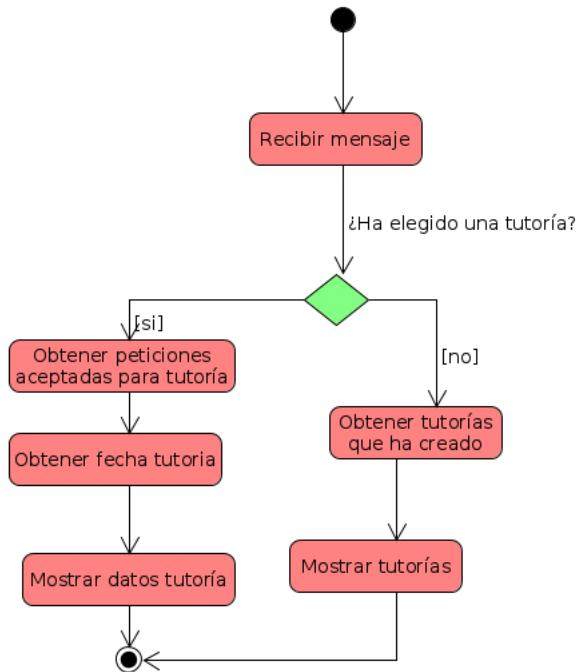


Figura 3.8: Diagrama actividad de CU-4.6 Ver información tutoría

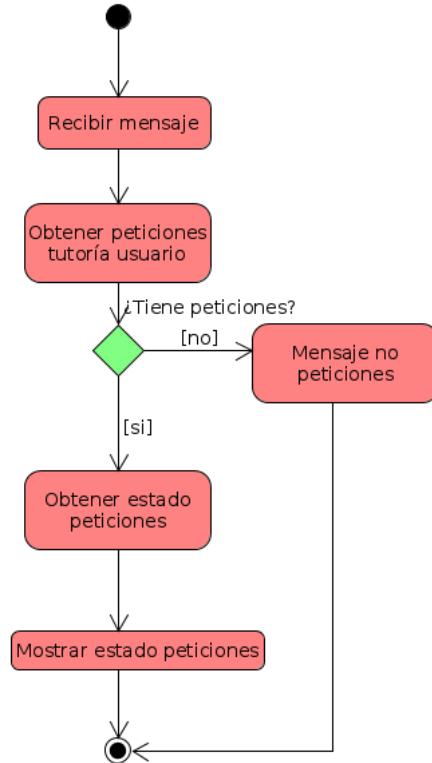


Figura 3.9: Diagrama actividad de CU-4.7 Ver peticiones realizadas

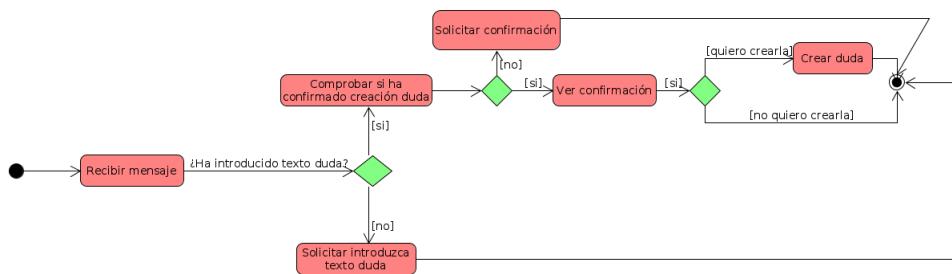


Figura 3.10: Diagrama actividad de CU-5.1 Crear duda

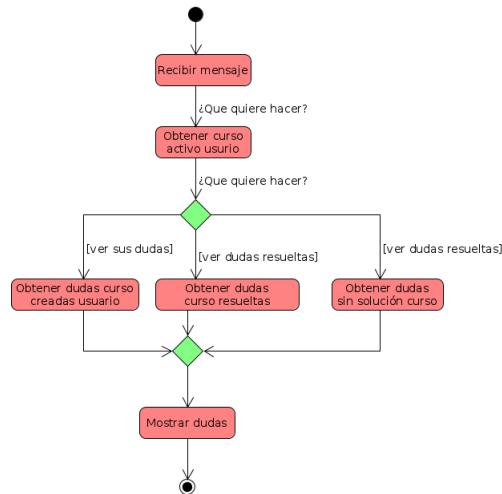


Figura 3.11: Diagrama actividad conjunto de CU-5.2 Ver dudas sin solución, CU-5.3 Ver mis dudas, CU-5.4 Ver dudas resueltas,

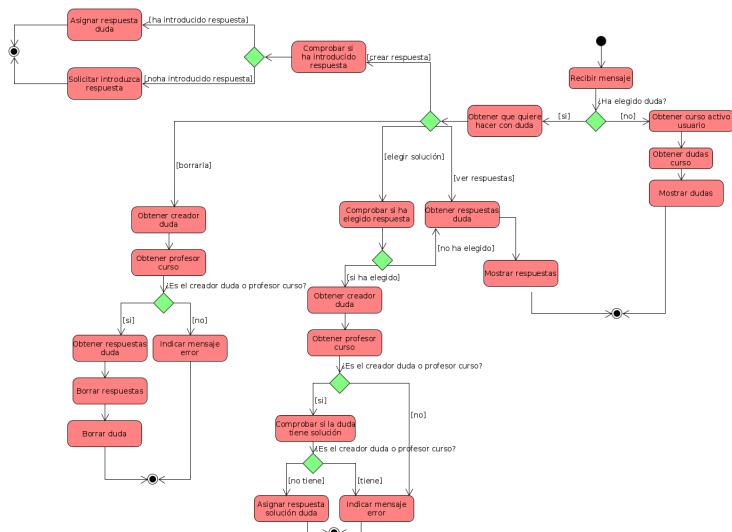


Figura 3.12: Diagrama actividad conjunto de CU-5.5 Crear respuesta a duda, CU-5.6 Ver respuesta duda, CU-5.7 Ver solución duda, CU-5.8 Borrar mi duda, CU-5.9 Elegir solución mi duda, CU-10 Borrar duda, CU-11 Elegir solución a duda

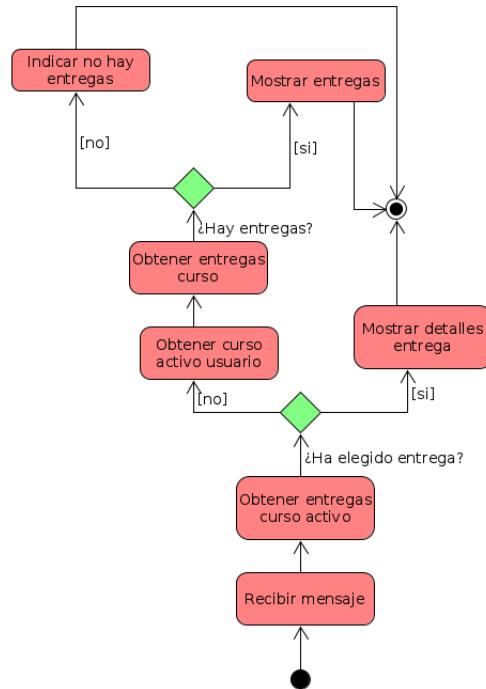


Figura 3.13: Diagrama actividad CU-6.1 Ver entregas

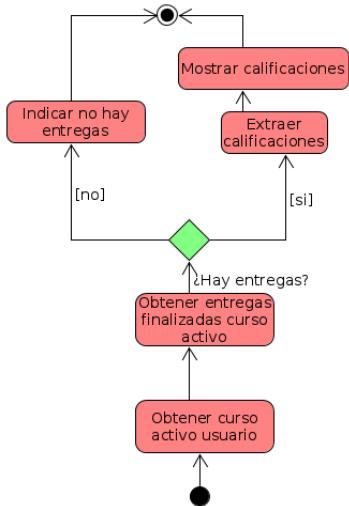


Figura 3.14: Diagrama actividad CU-6.2 Mostrar calificaciones

Capítulo 4

4.1. Análisis de los requisitos

4.1.1. Modelo conceptual

A continuación podemos ver los principales conceptos que comprende el sistema y cómo se relacionan. De la interfaz y sus conceptos hablaremos en el diseño de la interfaz.

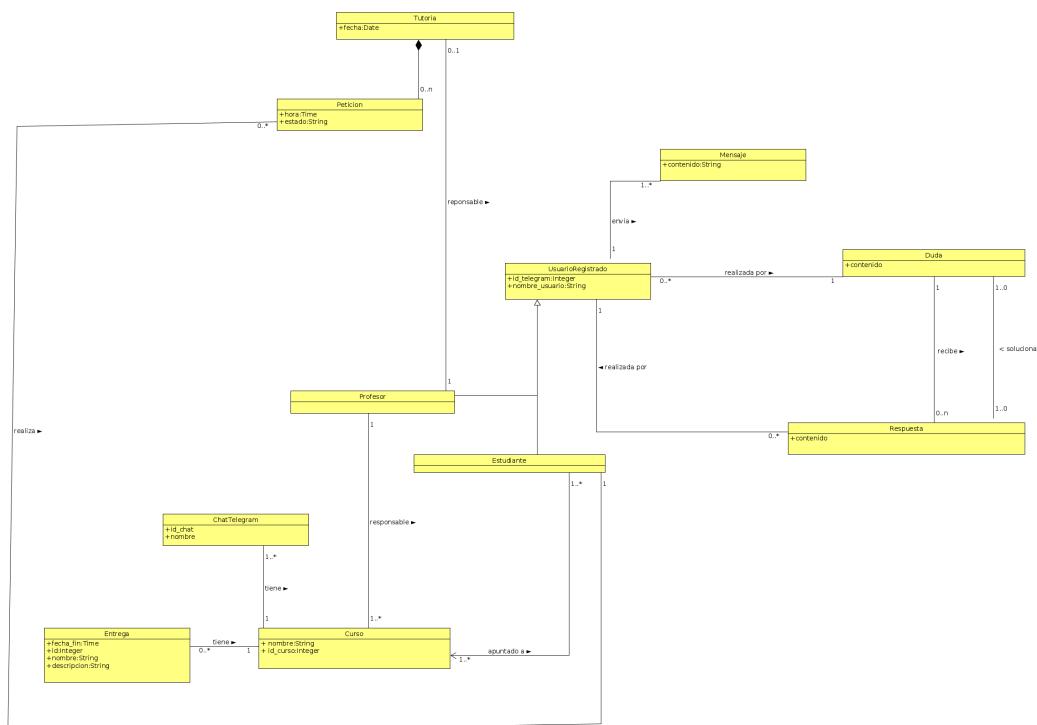


Figura 4.1: Diagrama conceptual del sistema

Presenta cierto parecido con el diagrama de casos de uso: profesor y estudiante heredan de usuario registrado que tienen relación con las dudas, profesor responsable de las tutorías y el estudiante realiza peticiones a éstas.

4.1.2. Diagramas secuencia

La meta buscada en esta sección es mostrar las interacciones entre los objetos que provocan la realización de algunos de los CUs más importantes por parte de los actores. Ahora mismo hay una clase “pradobot” que hace de mediadora entre el usuario y el resto de clases del sistema. Esta clase dirime qué es lo que quiere el usuario y lleva a cabo las interacciones necesarias con el resto de clases para cumplir el objetivo del usuario. Todas estas operaciones exigidas por el usuario se traducirán en interfaces cuyo diseño se tratará en la sección de diseño interacción usuario.

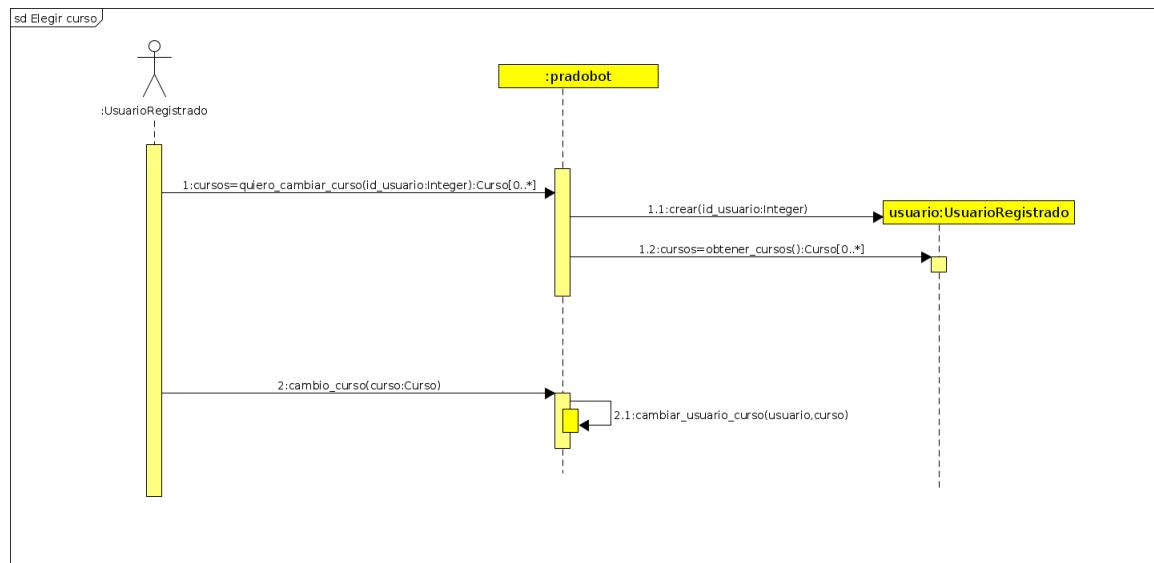


Figura 4.2: DS: Cambiar curso (CU-1.4)

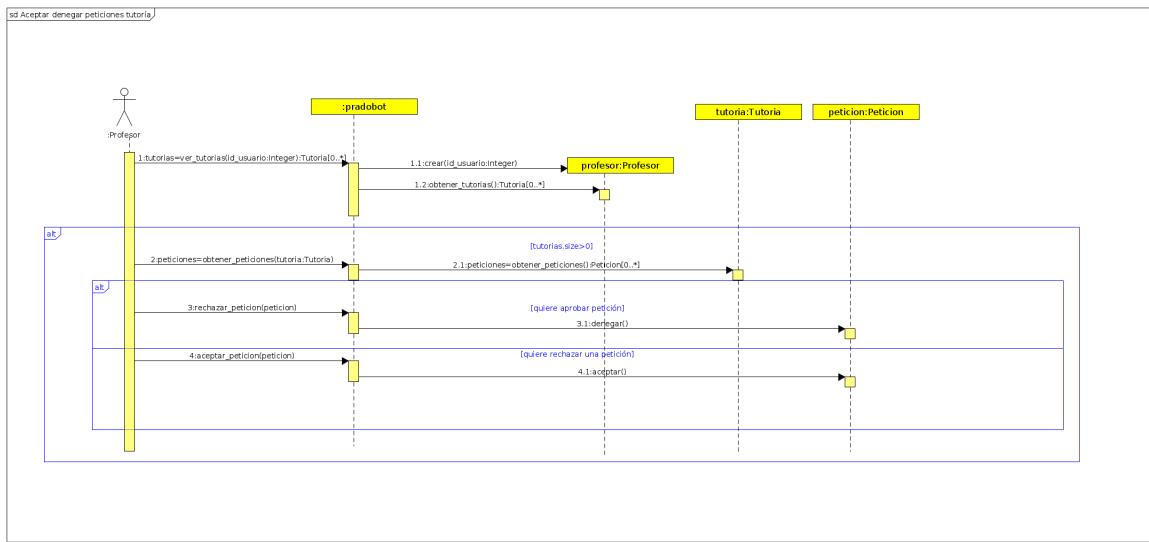


Figura 4.3: DS: Aceptar denegar peticiones (CU-4.4, CU-4.5)

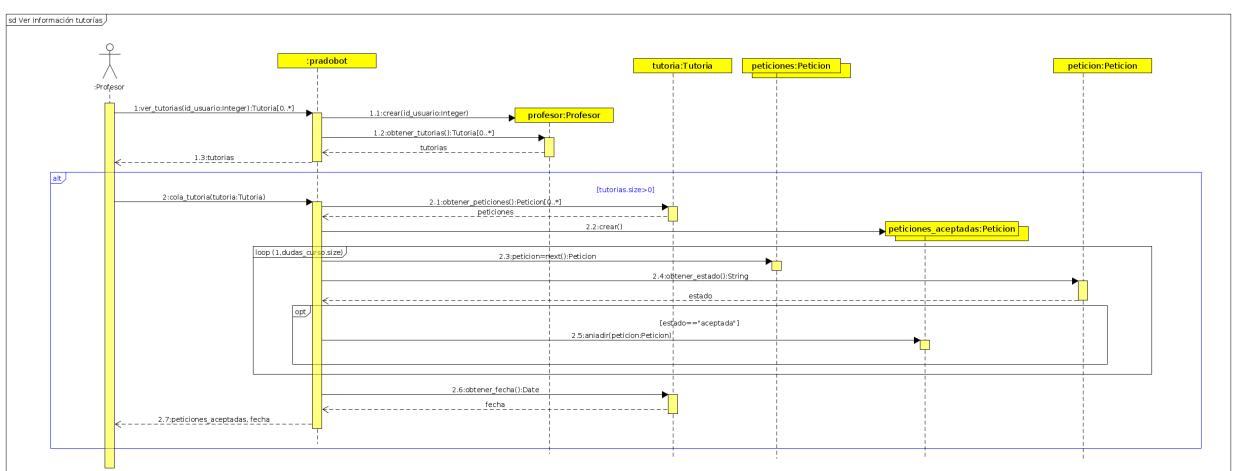


Figura 4.4: DS: Ver información tutoria (CU-4.6)

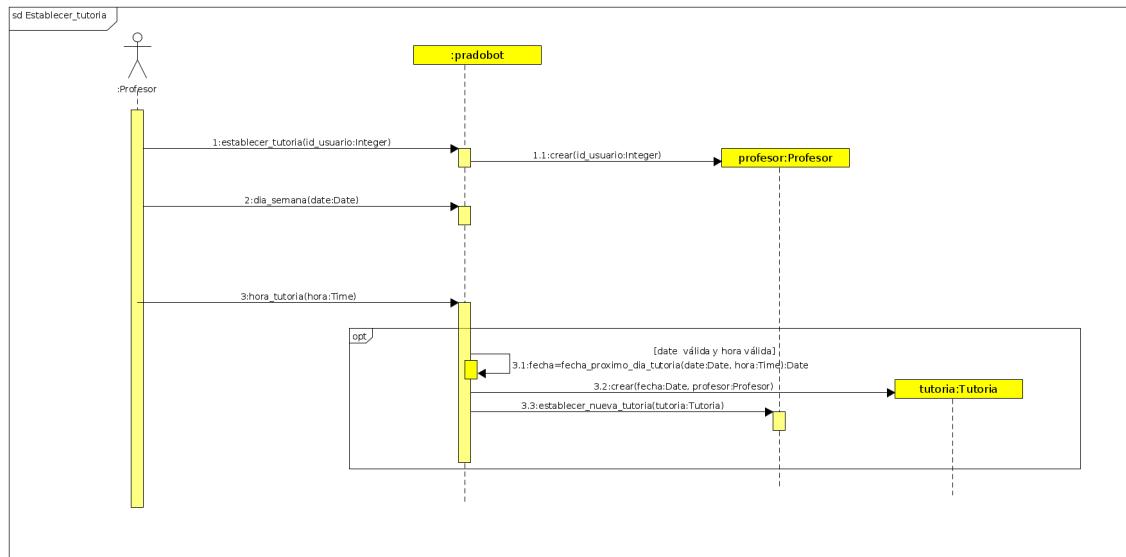


Figura 4.5: DS: Establecer tutoría (CU-4.1)

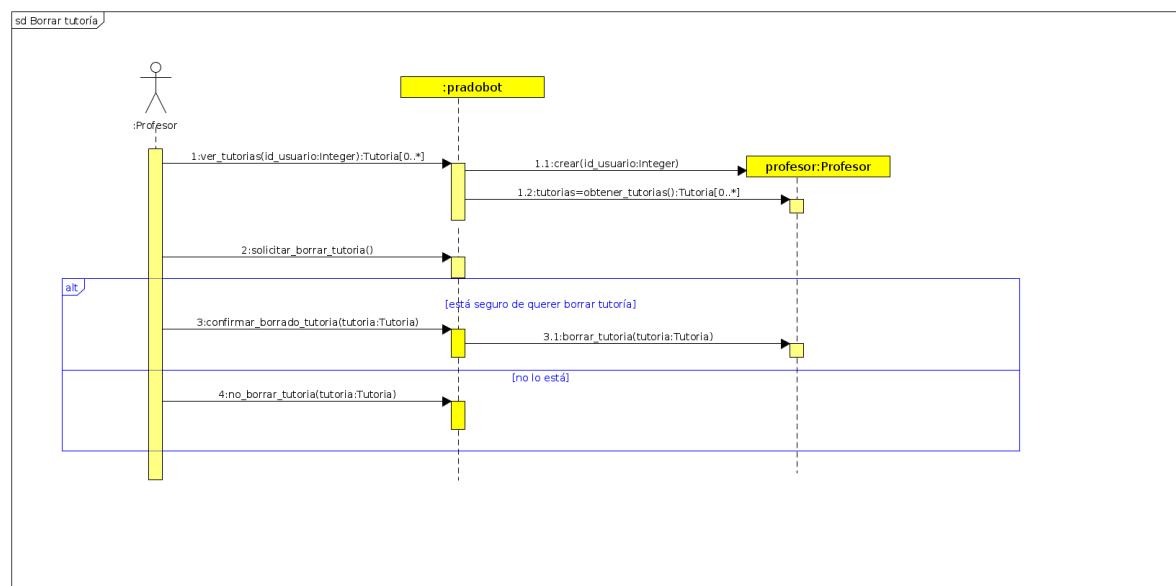


Figura 4.6: DS: Borrar tutoría (CU-4.2)

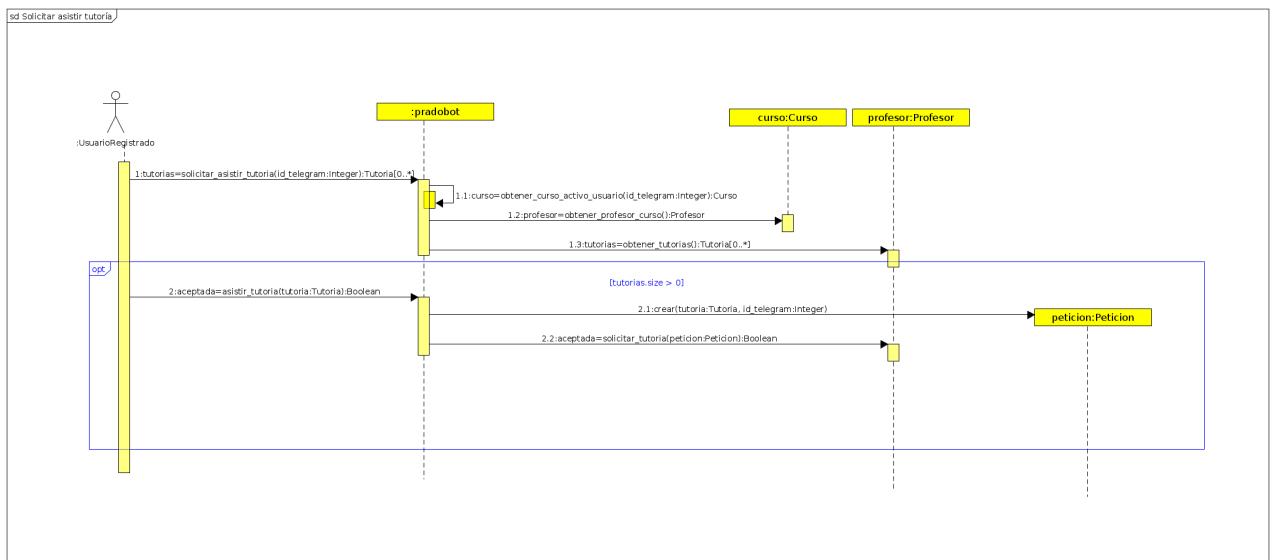


Figura 4.7: DS: Solicitar asistir tutoría (CU-4.3)

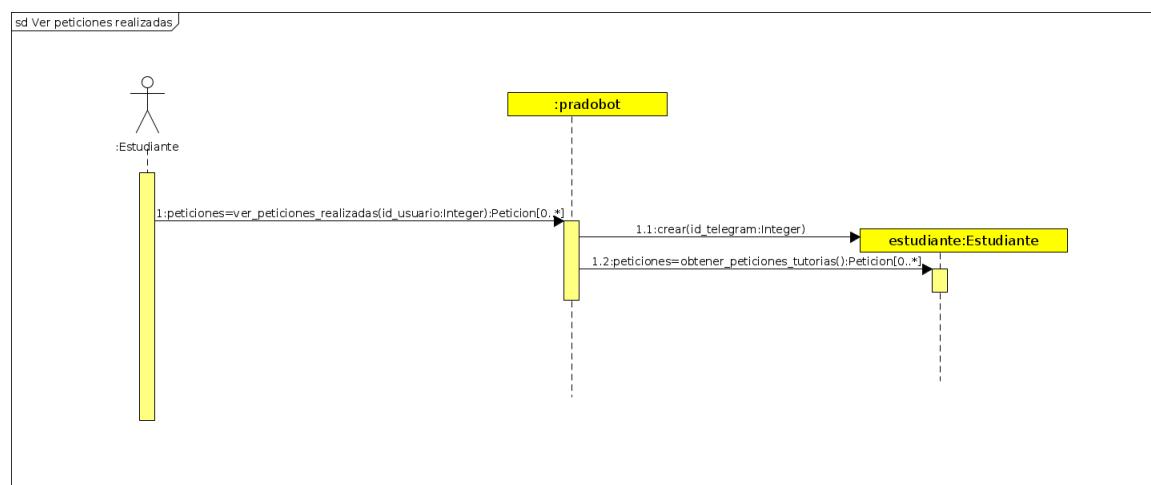


Figura 4.8: DS: Ver solicitudes realizadas (CU-4.7)

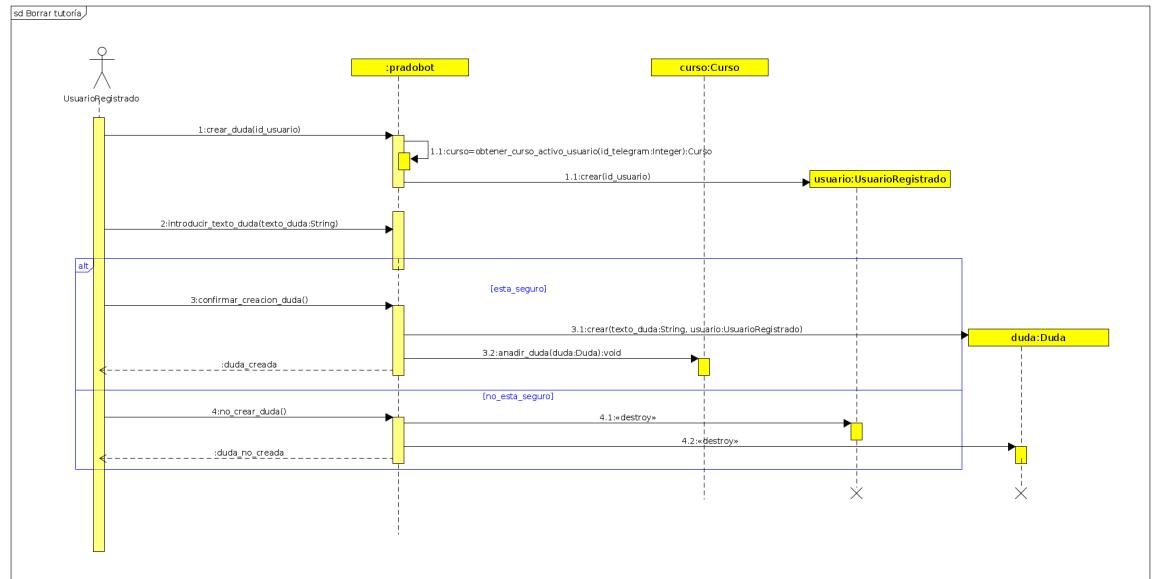


Figura 4.9: DS: Crear duda (CU-5.1)

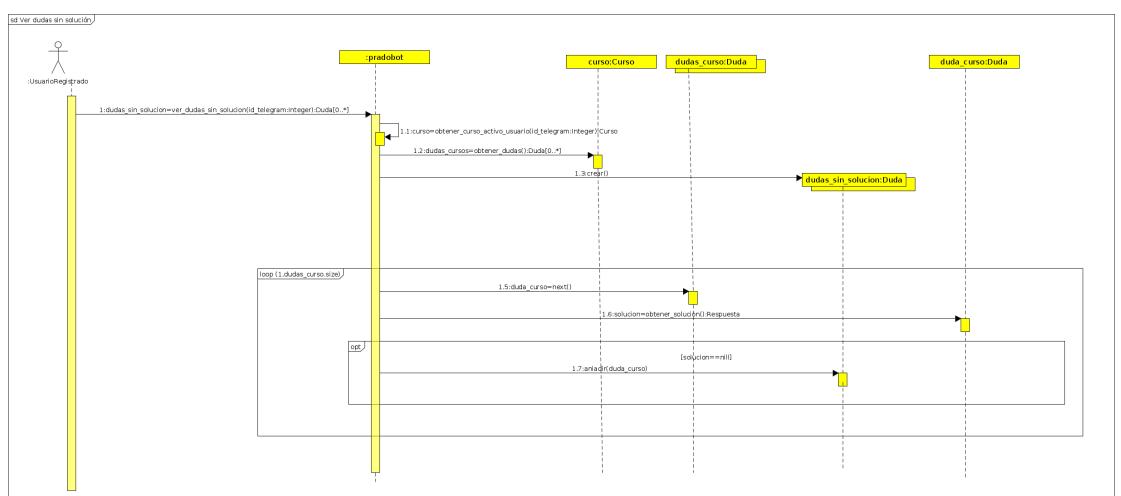


Figura 4.10: DS: Ver dudas sin solución (CU-5.2)

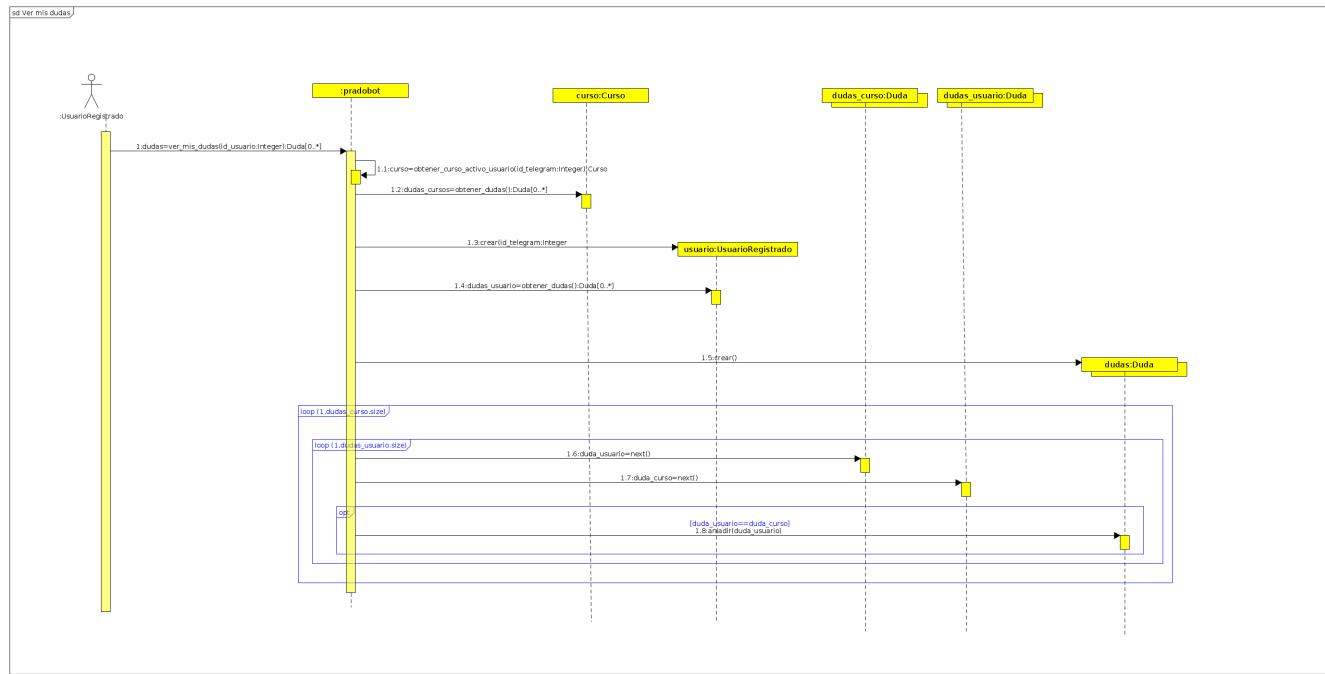


Figura 4.11: DS: Ver mis dudas (CU-5.3)

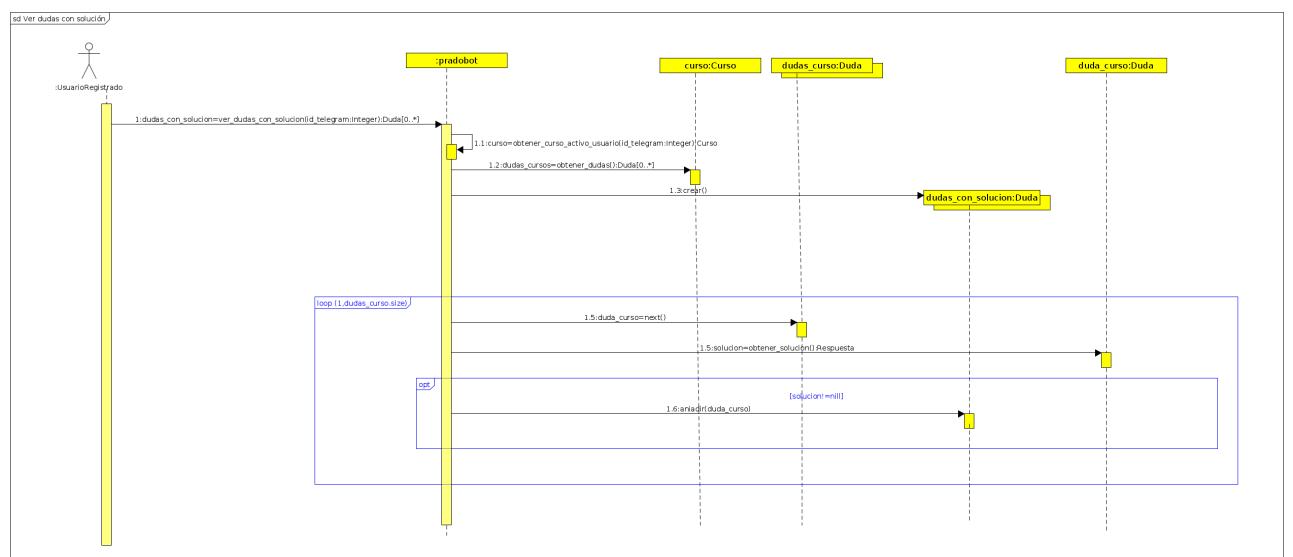


Figura 4.12: DS: Ver dudas con solución (CU-5.4)

4.1. Análisis de los requisitos

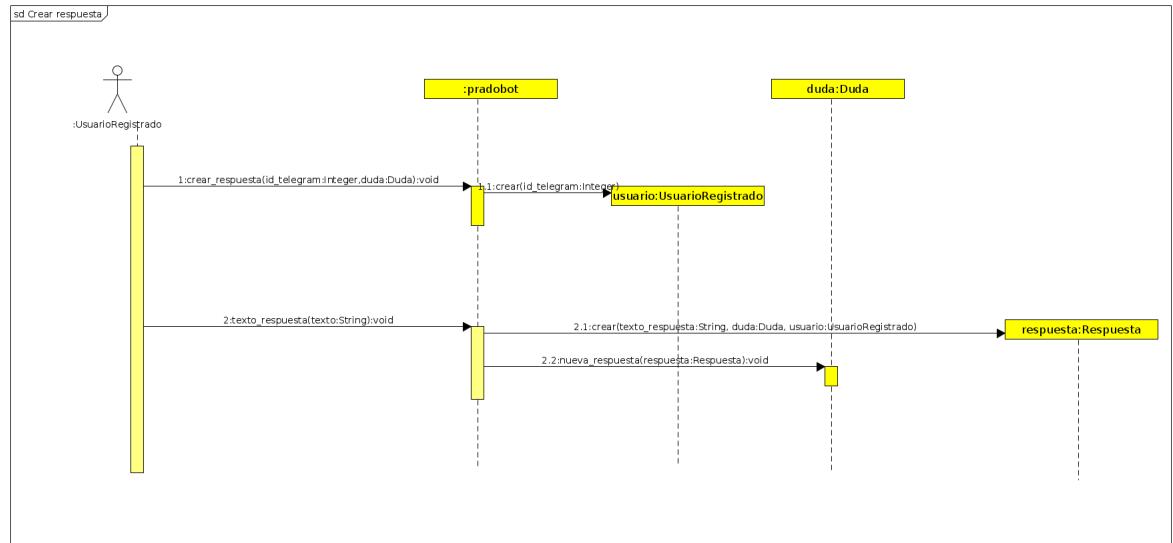


Figura 4.13: DS: Crear respuesta (CU-5.5)

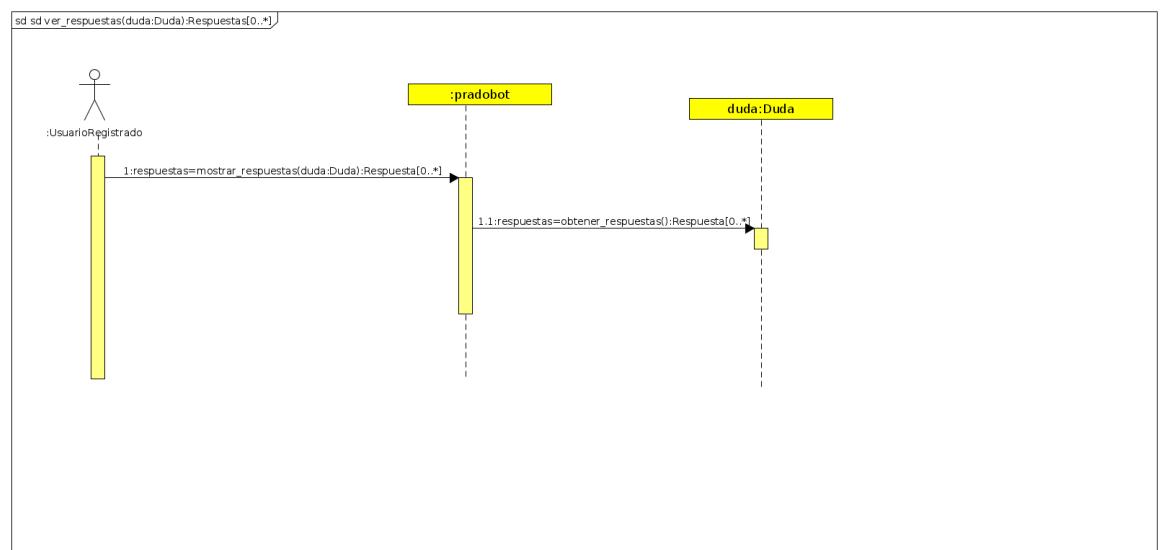


Figura 4.14: DS: Ver respuestas (CU-5.6)

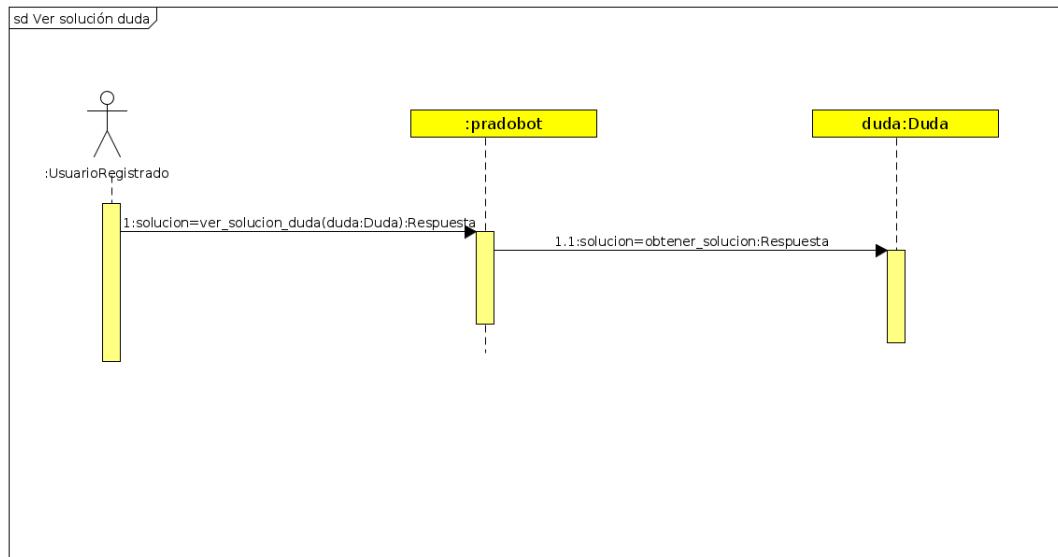


Figura 4.15: DS: Ver solución a duda (CU-5.7)

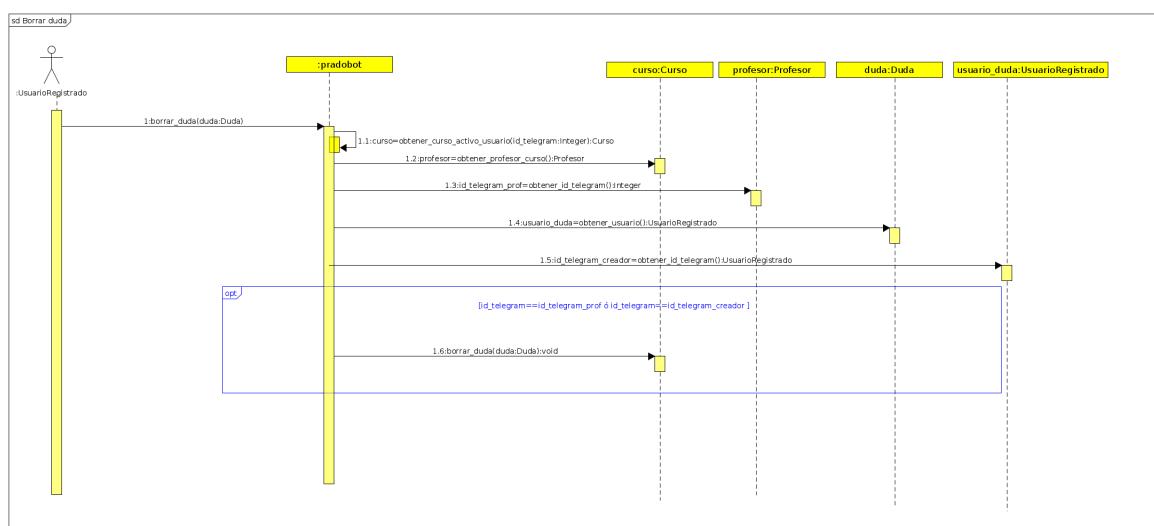


Figura 4.16: DS: Borrar duda (CU-5.8, CU-5.10)

4.1. Análisis de los requisitos

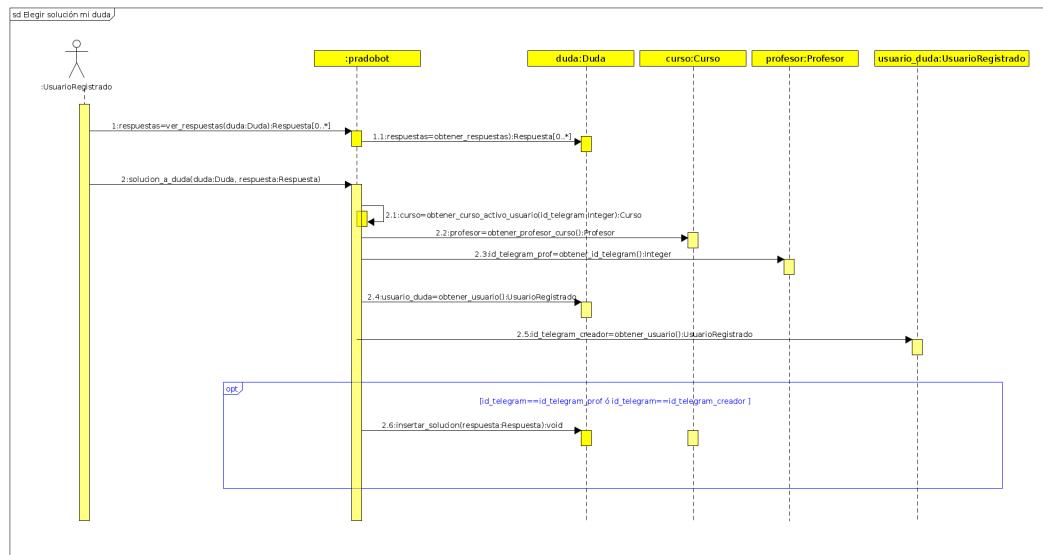


Figura 4.17: DS: Elegir solución duda (CU-5.9, CU-5.11)

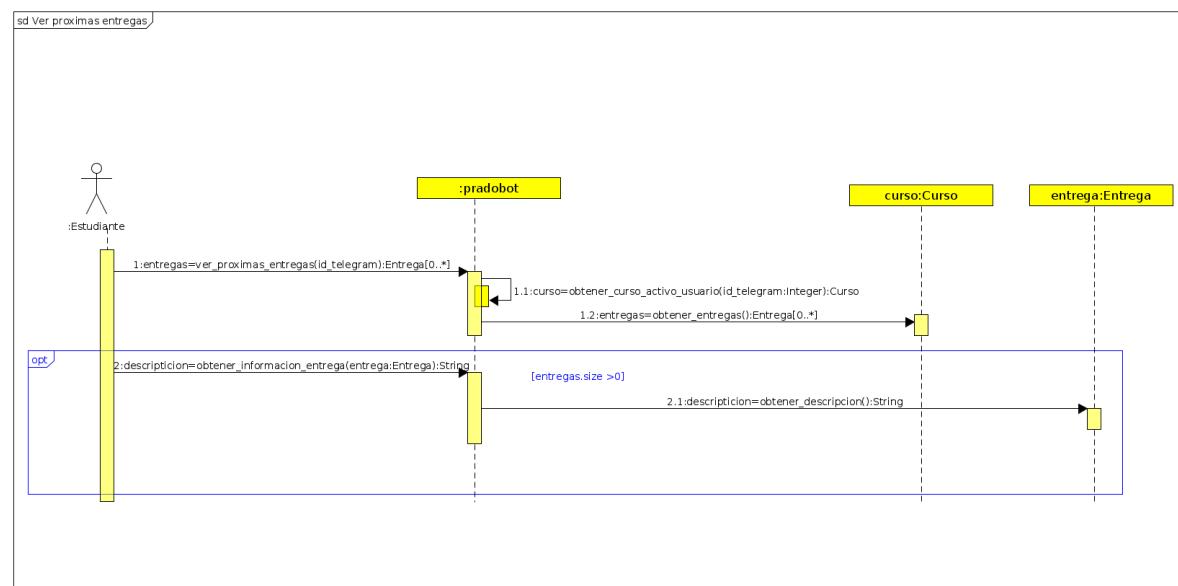


Figura 4.18: DS: Ver proximas entregas (CU-6.1)

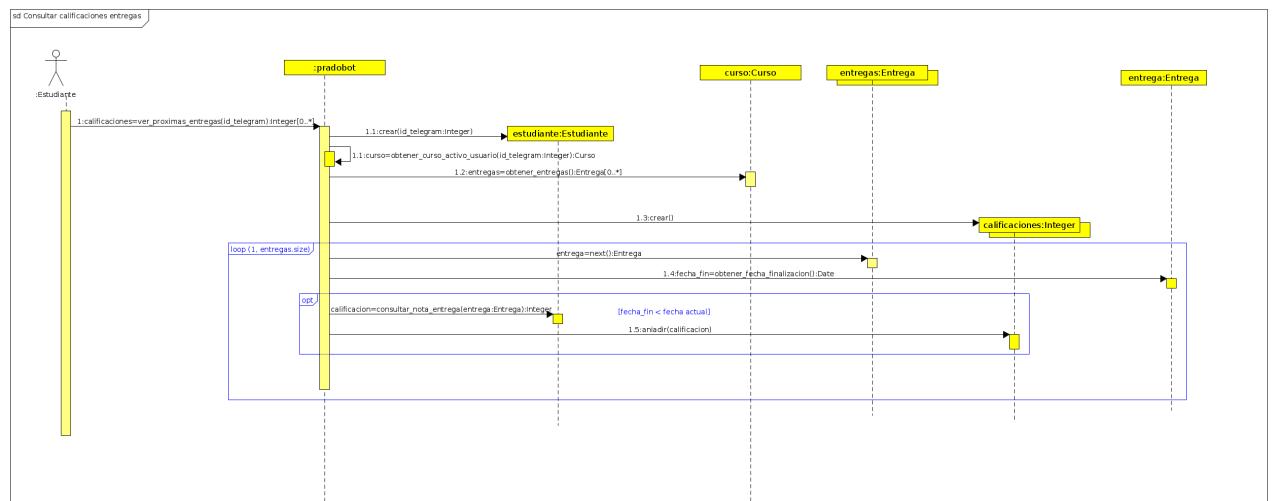


Figura 4.19: DS: Consultar calificaciones entregas (CU-6.2)

4.1.3. Contratos

Nombre	tutorias=obtener_tutorias()
Responsabilidad	Obteniene las tutorías de un Profesor.
Tipo	Profesor
Notas	
Excepciones	<ul style="list-style-type: none"> ▪ Que no tenga tutorías el objeto profesor.
Salida	Para cada objeto tipo Tutoria contenido en tutorias se proporciona: <ul style="list-style-type: none"> ▪ «El profesor que la ha creado» ▪ «Fecha tutoría»
Precondiciones	
Poscondiciones	

Nombre	peticiones=obtener_peticiones()
Responsabilidad	Obtiene las peticiones que ha recibido una Tutoria
Tipo	Tutoria
Notas	
Excepciones	<ul style="list-style-type: none"> ▪ Que no haya tenido ninguna petición la tutoría.
Salida	Para cada objeto Peticion contenido en peticiones se proporciona: <ul style="list-style-type: none"> ▪ La tutoría a la que se ha realizado la petición ▪ El estudiante que ha realizado la petición
Precondiciones	
Poscondiciones	

Nombre	denegar()
Responsabilidad	Cambia el estado de una petición a rechazada
Tipo	Peticion
Notas	
Excepciones	
Salida	
Precondiciones	
Poscondiciones	Fue modificado el objeto de la clase Peticion cambiándose el valor del atributo estado a “rechazada”.

Nombre	aceptar()
Responsabilidad	Cambia el estado de una Peticion a aceptada
Tipo	Peticion
Notas	
Excepciones	
Salida	
Precondiciones	
Poscondiciones	El valor del atributo estado del objeto de la clase Peticion fue modificado a “aceptada”.

Nombre	profesor=obtener_profesor_curso()
Responsabilidad	Obteniene a el Profesor responsable de un Curso
Tipo	Curso
Notas	
Excepciones	
Salida	<p>Para el objeto profesor de la clase Profesor contenido en profesor se proporciona</p> <ul style="list-style-type: none"> ▪ id_telegram
Precondiciones	
Poscondiciones	.

Nombre	aceptada=solicitar_tutoria(peticion)
Responsabilidad	Realiza una Peticion a una tutoría de un Profesor.
Tipo	Profesor
Notas	
Excepciones	<ul style="list-style-type: none"> ▪ El estudiante que realiza la petición ya haya realizado otra a la misma tutoría. ▪ El objeto estudiante de la clase Estudiante contenido en petición no existe.
Salida	Para el objeto aceptada de la clase Boolean se proporciona el valor true o false.
Precondiciones	
Poscondiciones	<ul style="list-style-type: none"> ▪ Fue creado un enlace entre el objeto peticion de la clase Peticion y el objeto de la clase Tutoria

Nombre	id_telegram=obtener_telegram()
Responsabilidad	Obteniene el identificador de Telegram un Profesor
Tipo	Profesor
Notas	
Excepciones	
Salida	Devuelve identificador numérico en el objeto de la clase Integer contenido en id_telegram.
Precondiciones	
Poscondiciones	.

Nombre	usuario_duda=obtener_usuario()
Responsabilidad	Obteniene al UsuarioRegistrado creador de una Duda
Tipo	Profesor
Notas	
Excepciones	
Salida	Para el objeto UsuarioRegistrado contenido en usuario_duda se proporciona: <ul style="list-style-type: none"> ■ id_telegram
Precondiciones	
Poscondiciones	.

Nombre	id_telegram=obtener_telegram()
Responsabilidad	Obtiene el identificador numérico de un UsuarioRegistrado
Tipo	UsuarioRegistrado
Notas	
Excepciones	
Salida	Devuelve identificador numérico para el objeto id_telegram de la clase Integer.
Precondiciones	
Poscondiciones	

Nombre	borrar_duda(duda)
Responsabilidad	Borra una Duda de un Curso
Tipo	Curso
Notas	
Excepciones	<ul style="list-style-type: none"> ▪ Que no exista el objeto duda de la clase Duda.
Salida	
Precondiciones	
Poscondiciones	<ul style="list-style-type: none"> ▪ Fue borrado el enlace entre el objeto duda y el el objeto de la clase Curso . ▪ Para cada objeto r de la clase Respuesta relacionado con el objeto duda de la clase Duda: <ul style="list-style-type: none"> • Fue borrado el enlace entre r y duda. • Fue borrado el objeto r. ▪ Fue borrado el enlace entre duda y el objeto de clase UsuarioRegistrado relacionado con duda. ▪ Fue borrado el objeto duda.

Nombre	borrar_tutoria(tutoria)
Responsabilidad	Elimina una Tutoría de un Profesor
Tipo	Profesor
Notas	
Excepciones	<ul style="list-style-type: none"> ▪ Que no exista el objeto tutoria de la clase Tutoria.
Salida	
Precondiciones	
Poscondiciones	<ul style="list-style-type: none"> ▪ Fue borrado el enlace entre el objeto tutoria y el objeto de la clase Profesor . ▪ Para cada objeto p de la clase Petición relacionado con el objeto tutoria de la clase Tutoria: <ul style="list-style-type: none"> • Fue borrado el enlace entre p y tutoria. • Fue borrado el enlace entre p y el objeto de la clase Estudiante con el que está relacionado. • Fue borrado el objeto p. ▪ Fue borrado el objeto tutoria

Nombre	entregas=obtener_entregas()
Responsabilidad	Obtener las entregas de un Curso
Tipo	Profesor
Notas	
Excepciones	<ul style="list-style-type: none"> ▪ No tenga ninguna entrega el objeto de la clase Curso.
Salida	<p>Para cada objeto Entrega contenido en entregas se proporciona:</p> <ul style="list-style-type: none"> ▪ fecha_fin ▪ identificador ▪ nombre
Precondiciones	
Poscondiciones	

Nombre	fecha_fin=obtener_fecha_finalizacion()
Responsabilidad	Obteniene la fecha finalización de una Entrega
Tipo	Entrega
Notas	
Excepciones	
Salida	Se devuelve el valor del atributo fecha_fin del objeto de la clase Entrega en el objeto tipo Date contenido en fecha_fin
Precondiciones	
Poscondiciones	

Nombre	aniadir_duda(duda)
Responsabilidad	Añade una nueva Duda al Curso
Tipo	Curso
Notas	
Excepciones	
Salida	
Precondiciones	
Poscondiciones	<ul style="list-style-type: none"> ▪ Fue creado un enlace entre el objeto duda de la clase Duda y el objeto de la clase Curso.

Nombre	nueva_respuesta(respuesta)
Responsabilidad	Añade una nueva Respuesta a una Duda
Tipo	Duda
Notas	
Excepciones	
Salida	
Precondiciones	
Poscondiciones	Fue creado un enlace entre el objeto respuesta de la clase Respuesta y el objeto de la clase Duda.

Nombre	cursos=obtener_cursos()
Responsabilidad	Obtiene los cursos en los que está dado de alta un Usuario-Registrado
Tipo	UsuarioRegistrado
Notas	
Excepciones	
Salida	<p>Para cada objeto Curso contenido en cursos se proporciona:</p> <ul style="list-style-type: none"> ■ id_curso ■ nombre_curso
Precondiciones	
Poscondiciones	

Nombre	respuestas=obtener_respuestas()
Responsabilidad	Obtiene las respuestas que tiene una Duda
Tipo	Duda
Notas	
Excepciones	<ul style="list-style-type: none"> ■ Que no haya tenido ninguna respuesta la duda.
Salida	<p>Para cada objeto Respuesta contenido en respuestas se proporciona:</p> <ul style="list-style-type: none"> ■ contenido ■ usuario ■ duda
Precondiciones	
Poscondiciones	

Nombre	insertar_solucion(respuesta:Respuesta)
Responsabilidad	Asigna una Respuesta como solución a una Duda
Tipo	Duda
Notas	
Excepciones	<ul style="list-style-type: none">▪ El objeto Duda ya tenga solución.
Salida	
Precondiciones	
Poscondiciones	Fue creado un nuevo enlace entre el objeto respuesta de la clase Respuesta y el objeto de la clase Duda.

Nombre	solucion=obtener_solucion()
Responsabilidad	Obtiene el objeto Respuesta que resuelve una Duda
Tipo	Duda
Notas	
Excepciones	<ul style="list-style-type: none"> ■ Objeto de la clase Duda no tenga ningún objeto de la clase Respuesta que sea su solución.
Salida	<p>Para el objeto de la clase Respuesta contenido en respuesta se proporciona:</p> <ul style="list-style-type: none"> ■ contenido ■ usuario ■ duda
Precondiciones	
Poscondiciones	

Nombre	dudas_curso=obtener_dudas()
Responsabilidad	Obtiene las dudas asociadas a un Curso
Tipo	Curso
Notas	
Excepciones	<ul style="list-style-type: none"> ■ Que el objeto Curso no tenga ninguna duda asociada.
Salida	<p>Para cada objeto Duda contenido en dudas se proporciona:</p> <ul style="list-style-type: none"> ■ contenido ■ usuario
Precondiciones	
Poscondiciones	

Nombre	estado=obtener_estado()
Responsabilidad	Obtiene el estado de una Peticion
Tipo	Peticion
Notas	
Excepciones	
Salida	Se proporciona el valor del atributo estado de un objeto de la clase Peticion en el objeto estado de la clase String.
Precondiciones	
Poscondiciones	

Nombre	fecha=obtener_fecha()
Responsabilidad	Proporciona la fecha que tiene una Tutoria.
Tipo	Tutoria
Notas	
Excepciones	
Salida	Se proporciona el valor del atributo fecha de un objeto de la clase Tutoria en el objeto fecha de la clase Date.
Precondiciones	
Poscondiciones	

Nombre	dudas=obtener_dudas()
Responsabilidad	Obtiene las dudas creadas por un UsuarioRegistrado
Tipo	UsuarioRegistrado
Notas	
Excepciones	
Salida	Para cada objeto de la clase duda contenido en dudas se proporciona: <ul style="list-style-type: none"> ■ contenido ■ usuario
Precondiciones	
Poscondiciones	

Nombre	peticiones=obtener_peticiones_tutoria()
Responsabilidad	Obtiene las peticiones realizadas por un Estudiante a una Tutoría
Tipo	Estudiante
Notas	
Excepciones	<ul style="list-style-type: none"> ■ El estudiante no haya realizado ninguna petición a una tutoría del profesor.
Salida	<p>Para cada objeto Peticion contenido en peticiones se proporciona:</p> <ul style="list-style-type: none"> ■ tutoria ■ estudiante ■ hora
Precondiciones	
Poscondiciones	

Nombre	descripcion=obtener_descripcion()
Responsabilidad	Obtiene la descripción de una entrega.
Tipo	Estudiante
Notas	
Excepciones	<ul style="list-style-type: none"> ■ La entrega no tenga descripción.
Salida	Se proporciona el valor del atributo descripción del objeto de la clase entrega en el objeto descripcion de la clase String.
Precondiciones	
Poscondiciones	

Nombre	establecer_nueva_tutoria(tutoria))
Responsabilidad	Crea una nueva tutoría para el profesor
Tipo	UsuarioRegistrado
Notas	
Excepciones	<ul style="list-style-type: none"> ■ Objeto de la clase Tutoría con los mismos datos ya existe.
Salida	
Precondiciones	
Poscondiciones	Fue creado un enlace entre el objeto tutoria de la clase tutoria y el objeto de la clase Profesor.

Nombre	establecer_nueva_tutoria(tutoria))
Responsabilidad	Crea una nueva tutoría para el profesor
Tipo	UsuarioRegistrado
Notas	
Excepciones	<ul style="list-style-type: none">■ Objeto de la clase Tutoria con los mismos datos ya existe.
Salida	
Precondiciones	
Poscondiciones	Fue creado un enlace entre el objeto tutoria de la clase tutoria y el objeto de la clase Profesor.

Capítulo 5

5.1. Diseño

Hasta ahora, se ha mostrado de una forma muy abstracta, cómo interacciona el usuario con el *bot*, por lo que vamos a abordar en primer lugar, cómo se puede hacer la interacción usuario-*bot* antes de empezar a dar más forma al programa.

5.1.1. Diseño de la interacción con el usuario

5.1.1.1. Chats grupales

Conocer qué quiere un usuario por un chat grupal es fácil, ya que el *bot* solo es invocado cuando recibe un mensaje en el formato “/quiero_hacer_algo”¹ es decir, que si alguien escribe /temperatura Estambul el bot recibe /temperatura Estambul pero si escriben temperatura Roma el *bot* no recibe ningún mensaje. Esto hace que sea fácil para un usuario indicar al *bot* qué quiere por un chat grupal ya que solamente le hace falta conocer los comandos tipo /comando xxx que soporta el *bot* y lo que hacen.

5.1.1.2. Chats privados

Más peliaguda es la interacción entre el usuario y el *bot* en un chat privado ya que, al contrario que en los CUs donde el actor era un usuario de un chat grupal, en la que toda interacción se reducía a un solo paso por parte del usuario en los CUs de los usuarios de los chats privados la realización de una acción requiere de varios pasos y utilizar el formato /asignar_solucion_duda id_duda 7 id_respuesta 9 queda sumamente críptico y poco amigable. Por todo lo anteriormente dicho se ha decidido utilizar los menús gráficos que nos proporciona Telegram:

¹En realidad si pones como moderador a un bot de un chat puede recibir todos los mensajes de un chat

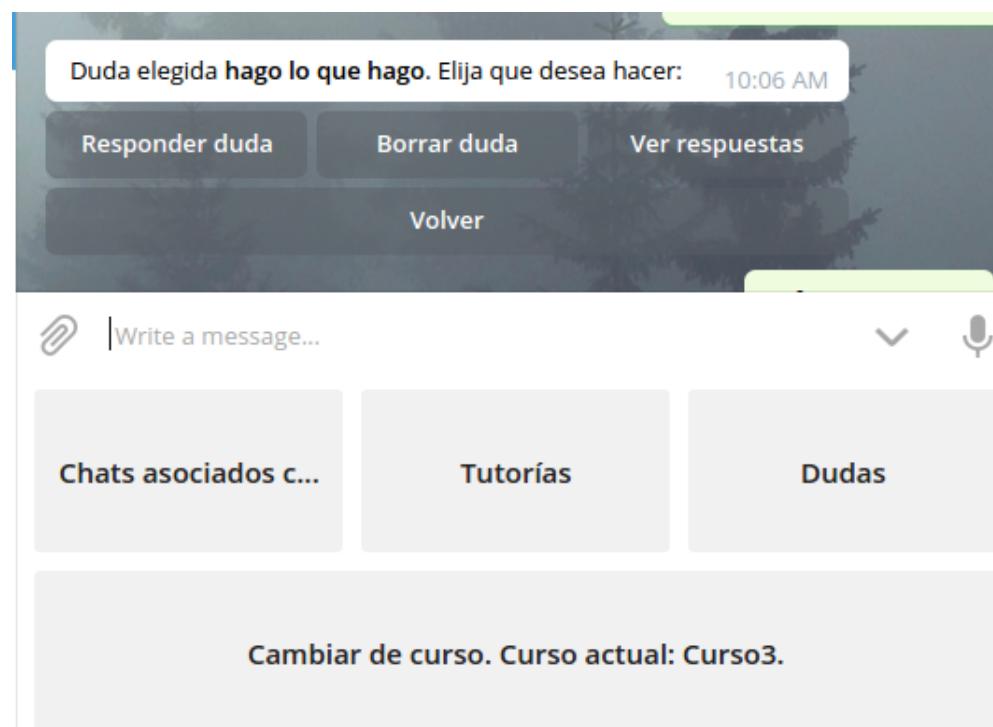


Figura 5.1: Arriba menú tipo Inline, abajo menú tipo teclado

Estos menús tienen la peculiaridad de que cuando un usuario pulsa una tecla el bot recibe un mensaje tipo “callback” (caso menús Inline) o de texto (menús tipo teclado) que identifica a la tecla pulsada dentro del menú. Dicho lo cual, si se realiza un buen diseño de los menús, tanto el usuario como el bot pueden comprender perfectamente qué es lo que quiere el uno del otro.

No se ha pensado en el uso de menús en los chats grupales porque pienso que generarían un ruido innecesario y que al ser interacciones “de un solo paso” por parte del usuario son innecesarios.

5.1.2. Diseño de la estructura

Comenzamos a refinar el sistema para que éste empiece a estar lo suficientemente definido como para llevar a cabo su implementación. Hemos hablado brevemente del enrutamiento de mensajes en los CUs Reales y en los diagramas anteriores hemos mostrado una clase “pradobot” que es la encargada de conocer qué es lo que el usuario quiere y llevar a cabo las acciones pertinentes. Vamos a empezar a repartir las responsabilidades de dicha clase utilizando diferentes patrones de diseño.

5.1.2.1. Primera capa: controladores

El bot es un programa que constantemente está escuchando por si Telegram le envía un nuevo mensaje y una vez recibido realiza una acción u otra. Como hemos indicado en el apartado de diseño de interacción de usuario, la forma en que el bot se relaciona con un usuario entre un chat privado y uno grupal es muy diferente requiriendo la gestión de acciones que hacen uso de múltiples pasos mediante menús para los usuarios de un chat privado ,mientras que las acciones que ejecutan los usuarios de los chats grupales solamente son de un paso y sin menús.

Esto nos lleva a aplicar el patrón “**Controlador**” y distribuir las responsabilidades entre tres clases:

- **Mensajero:** su función es la de recibir mensajes, ver si proceden de un chat privado o grupal y transferir el mensaje a la clase responsable del manejo de mensajes procedentes de chats privados o a la de chats grupales.
- **ManejadorMensajesChatsPrivados:** Recibe mensajes procedentes de un chat privado y lleva a cabo las siguientes funciones:
 - Ver si el usuario que le habla está registrado en el sistema.
 - Controlar si el sistema ha acabado ya de procesar el último mensaje que le mandó el usuario.
 - Entregar el mensaje que le envía el usuario a una capa inferior para que ésta lleve a cabo una acción con el mensaje.

Esta clase no tiene la responsabilidad de ver qué se hace con el mensaje sino que lo transfiere a otra clase para que ésta lo procese.

- **ManejadorMensajesChatsGrupales:** Recibe mensajes procedentes de un chat grupal. Sus funciones a destacar son:
 - Controlar si los mensajes proceden de chats asociados a cursos del sistema.
 - Llevar a cabo el procesamiento del mensaje, lo cual desencadena en una acción.

Como esta clase es la que conoce qué chat corresponde a qué curso y las interacciones con los chats grupales son tan básicas, se le ha asignado también la responsabilidad de ejecutar lo que quiere el usuario en el mensaje.

Podemos ver el funcionamiento de esta primera capa en las siguientes capturas:

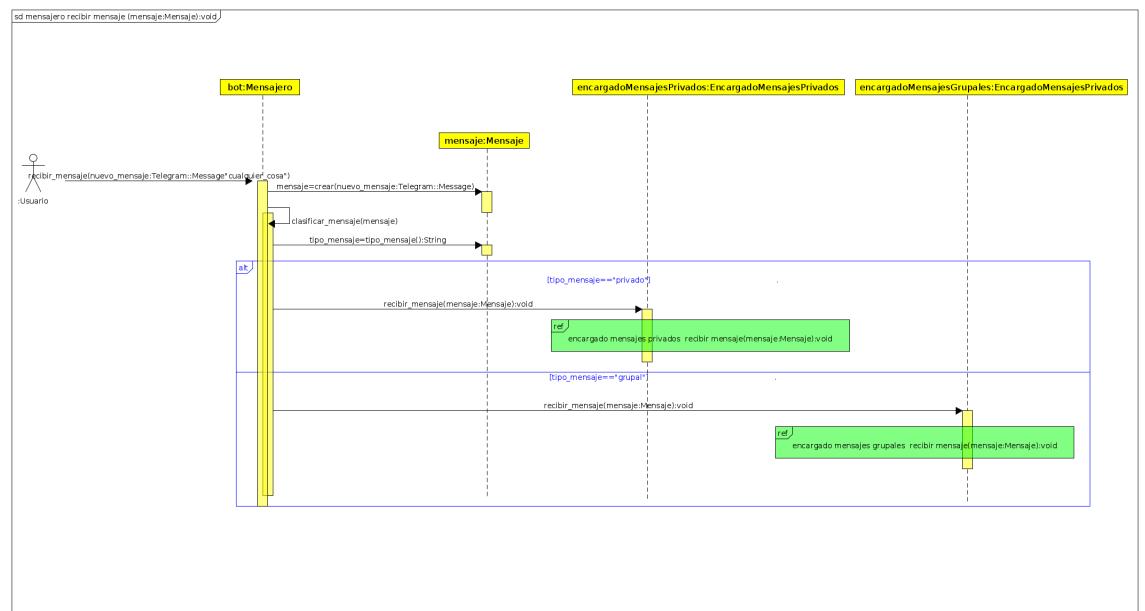


Figura 5.2: Diagrama secuencia recibir_mensaje clase Mensajero: el bot recibe un nuevo mensaje

He preferido mostrar el funcionamiento de como opera el ManejadorMensajesPrivados debido a que es bastante más complejo que el de chats grupales:

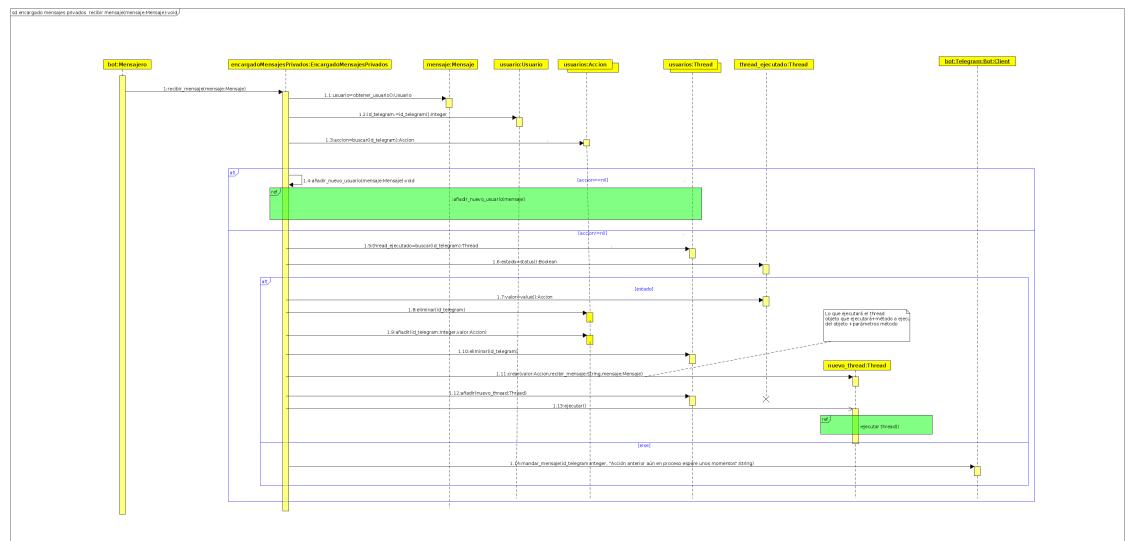


Figura 5.3: Diagrama secuencia recibir_mensaje clase ManejadorMensajes-ChatsPrivados

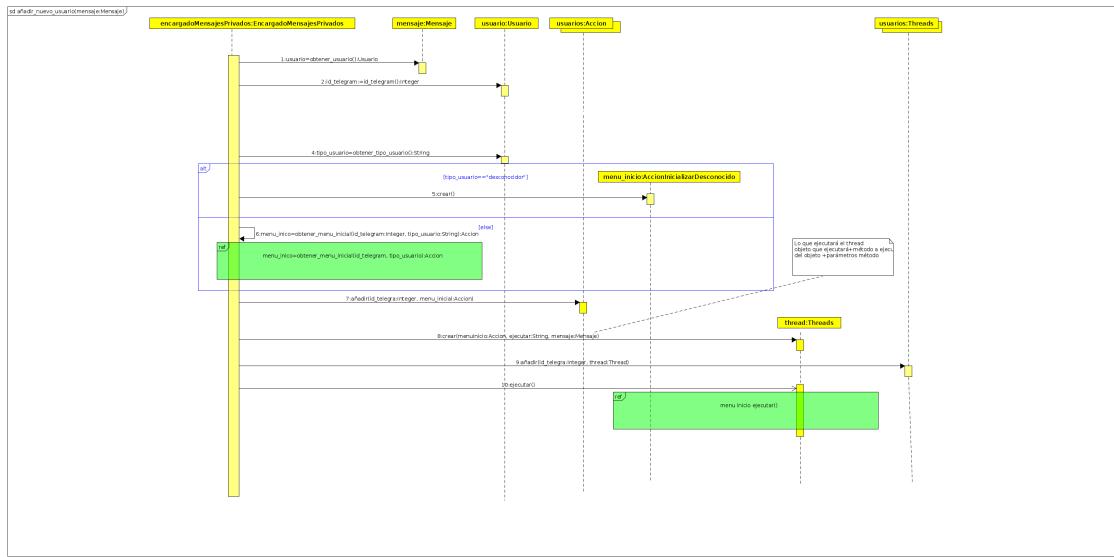


Figura 5.4: Diagrama secuencia añadir_nuevo_usuario clase ManejadorMensajesChatsPrivados

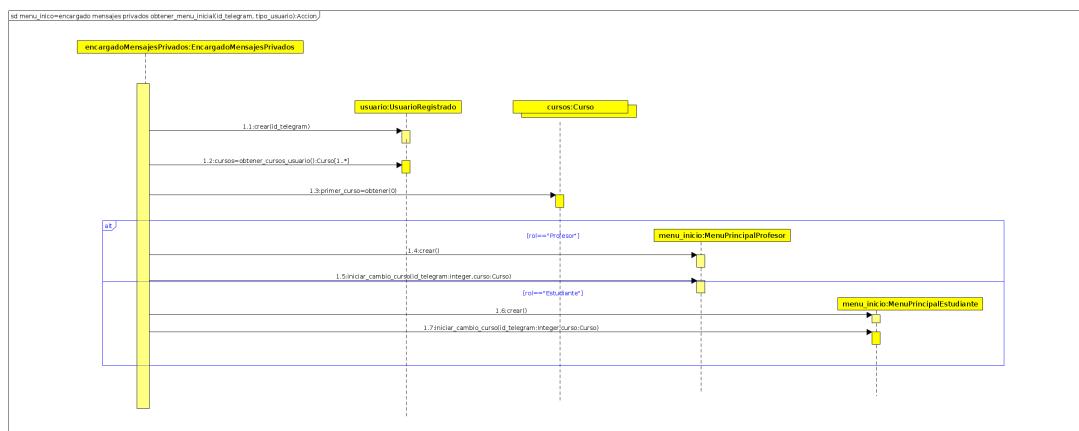


Figura 5.5: Diagrama secuencia obtener_menu_inicio clase ManejadorMensajesChatsPrivados

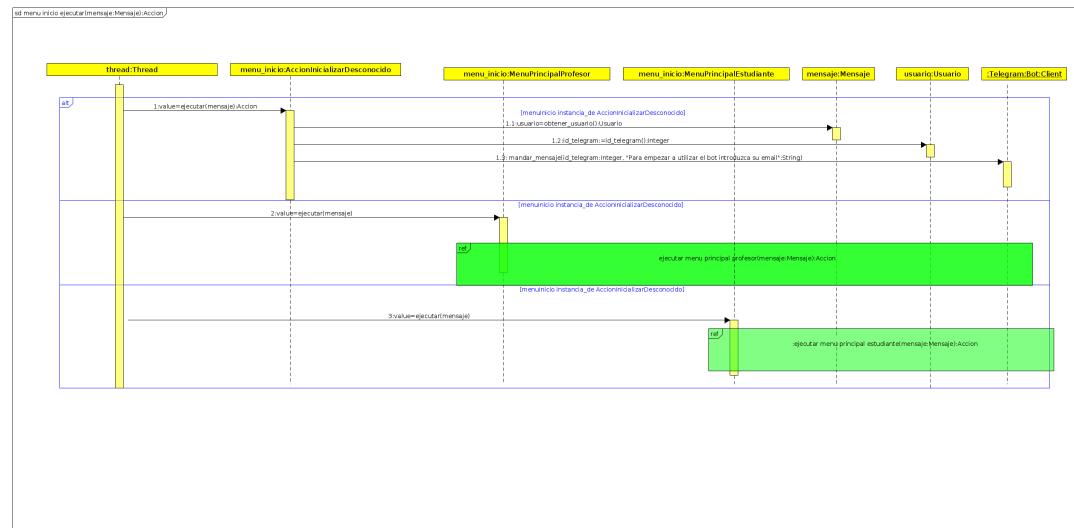


Figura 5.6: Diagrama secuencia ejecutar_menu_inicio clase ManejadorMensajesChatsPrivados

En la siguiente captura se puede apreciar el polimorfismo. Explicaremos su uso en el programa en la siguiente sección.

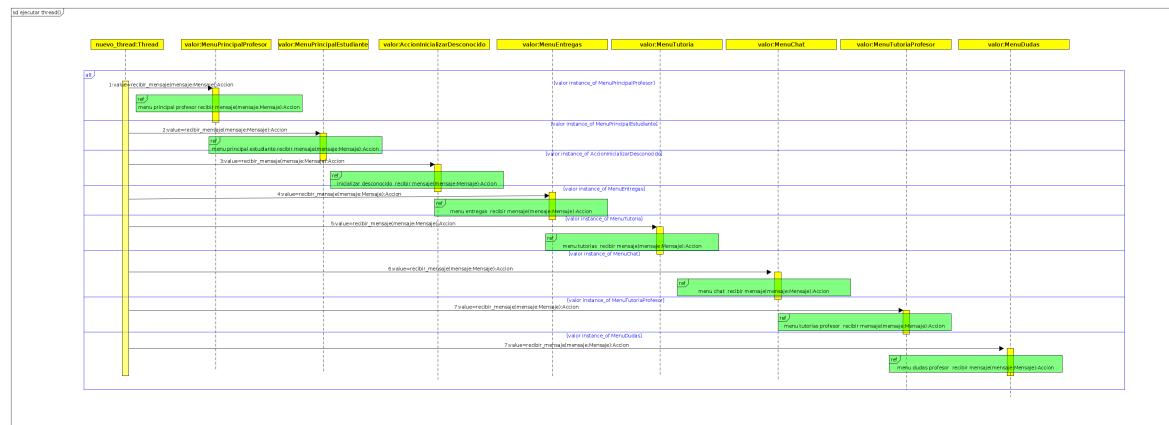


Figura 5.7: Diagrama secuencia muestra ejecución thread iniciado por ManejadorMensajesChatsPrivados

Segunda capa: menús

ManejadorMensajesChatsPrivados le pasa el mensaje a otra clase que es la encargada de ver qué quiere el usuario con el mensaje. Debido al diseño mediante menús que se ha decidido hacer para interaccionar con el

usuario, la clase que realmente sabe qué es lo que quiere hacer el usuario con su mensaje es aquella que controla al menú que está viendo el usuario ya que ésta es la que conoce las opciones contenidas en el menú. El mensaje que recibe un menú puede indicar tres cosas:

- Selección de una nueva opción del menú.
- Indicar que se quiere cambiar de menú.
- El mensaje no está destinado al menú sino a la última opción que pulsó el usuario en el menú.

El procesamiento de un mensaje por parte de una clase menú siempre devuelve otra clase menú que será la que reciba el proximo mensaje de ManejadorMensajesChatsPrivados. Puede devolverse a ella misma o a su menú padre. He aquí el **Polimorfismo**. La jerarquía de menús puede descomponerse en:

- MenuPrincipalEstudiante: El menú que vé un estudiante la primera vez que le manda un mensaje al bot. Contiene:
 - MenuTutorias
 - MenuEntregas
 - MenuDudas
- MenuPrincipalProfesor: Igual que el menú anterior pero para el profesor. Contiene:
 - MenuTutoriasProfesor
 - MenuChatTelegram
 - MenuDudas
- MenuTutoriasProfesor: Contiene las opciones:
 - Ver cola de tutorías
 - Crear tutoría
 - Borrar tutoría.
- MenuChatTelegram: Contiene las opciones:
 - Asociar curso a chat.
- MenuTutorias: Permite al estudiante elegir entre:
 - Solicitar asistencia tutoría.
 - Ver estado de sus solicitudes.

- MenuEntregas: Permite al estudiante:
 - Obtener información de próximas entregas.
 - Ver calificaciones.
- MenuDudas: contiene las opciones:
 - Crear duda.
 - Opción que permite:
 - Ver dudas con solución.
 - Ver dudas sin solución.
 - Ver dudas del usuario.
 - Ver respuestas dudas.
 - Seleccionar una respuesta como solución a duda.
 - Borrar duda.

La jerarquía de menús se ha diseñado con especial énfasis en que las acciones que realizan algo similar estén contenidas en el mismo menú. Aquellas opciones que permiten más de una opción han sido anidadas así para evitar la repetición de acciones por parte de un usuario. Ejemplo:

Si un usuario selecciona una duda se le muestre que puede borrar la duda, ver sus respuestas, asignarle una solución... y no al revés, es decir, tener un menú con la opción borrar duda, ver respuestas, asignarle solución y que nada más pulsar cualquiera de ellas te solicite que elijas la duda.

Todos los menús, además, contienen la opción de cambiar de curso y la de volver al menú anterior (menos los menús principales ya que no tienen menú anterior).

5.1.2.2. Tercera capa: opciones de menús

Cada opción de un menú está representada por una clase, que es la que conoce los pasos que tiene que realizar un usuario para llevar a cabo la acción que representa, así como de mediar entre el usuario y los “objetos” que intervienen en esa acción. Recibe los mensajes procedentes del menú que la contiene y de ellos extrae la información que necesita.

5.1.2.3. Cuarta capa: contenedores de datos

El objetivo de las clases de esta capa es la de hacer de árbitro entre los datos (Moodle y la base de datos) y el resto del sistema. Son las únicas que utilizan la API de Moodle cuando es necesario y hacen llamadas a la base

de datos cuando se les solicita un dato de la clase que simbolizan (ejemplo: dudas creadas por un usuario).

Para ilustrar las interacciones entre la tercera y cuarta capas podemos ver el siguiente diagrama de comunicación de la opción solicitar tutoría que recibe mensaje de menú MenuTutorias:

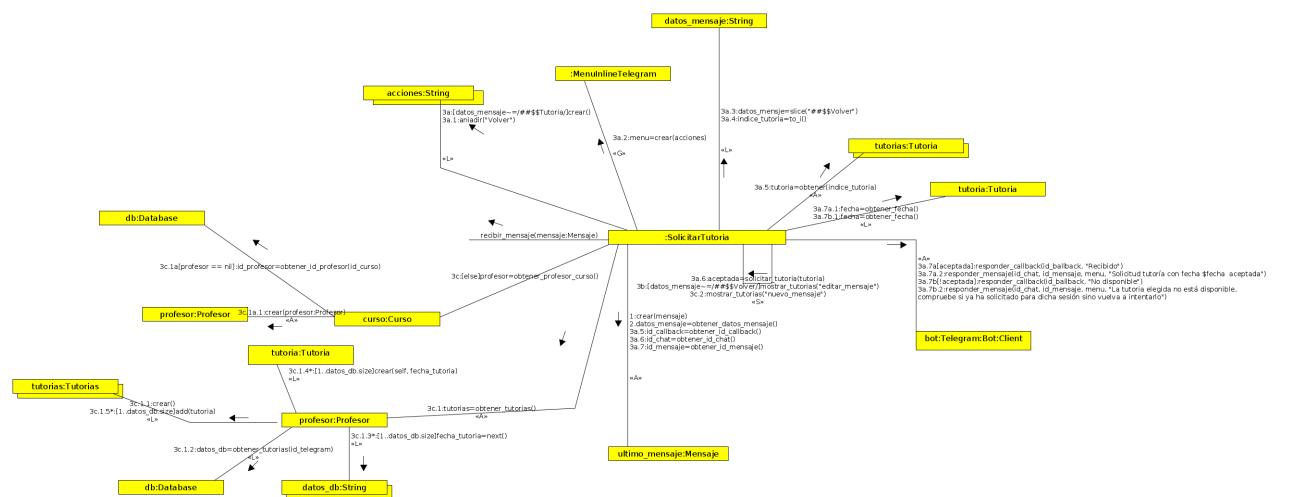


Figura 5.8: Diagrama comunicación recibir_mensaje clase SolicitarTutoria

En el diagrama se puede observar como la clase **SolicitarTutoria** crea y envía diferentes mensajes al usuario empleando para el contenido de estos información de las clases **Profesor** y **Tutoría**.

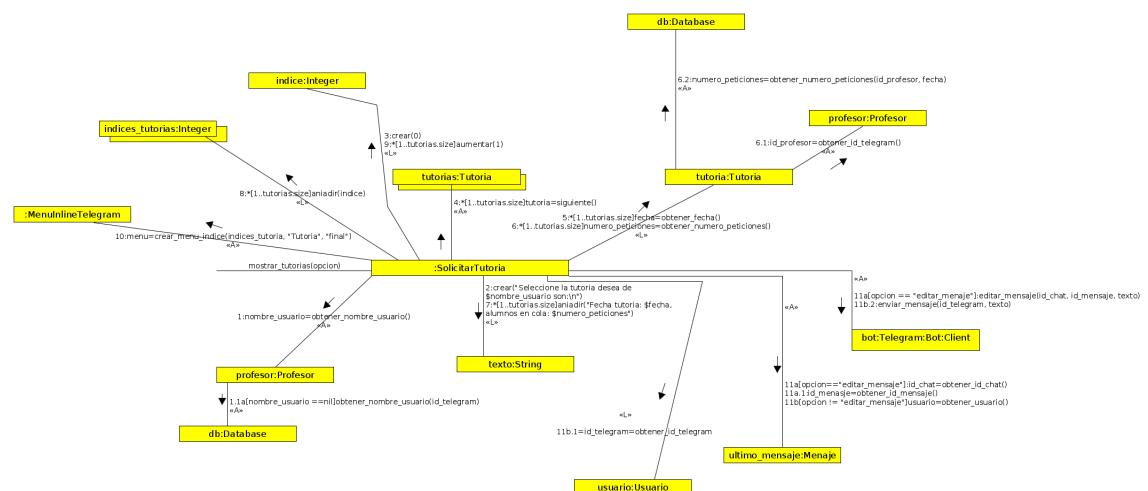


Figura 5.9: Diagrama comunicación mostrar_tutorias clase SolicitarTutoria

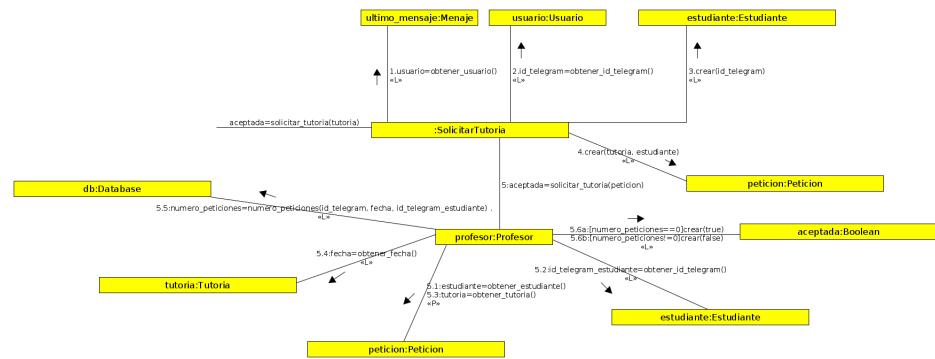


Figura 5.10: Diagrama comunicación solicitar_tutoria clase SolicitarTutoria

El diagrama de clases obtenido al final para la aplicación sale demasiado grande para que sea fácilmente visible en el pdf. Se puede encontrar dentro de la carpeta diagramas con el nombre `diagrama_clases.png`.

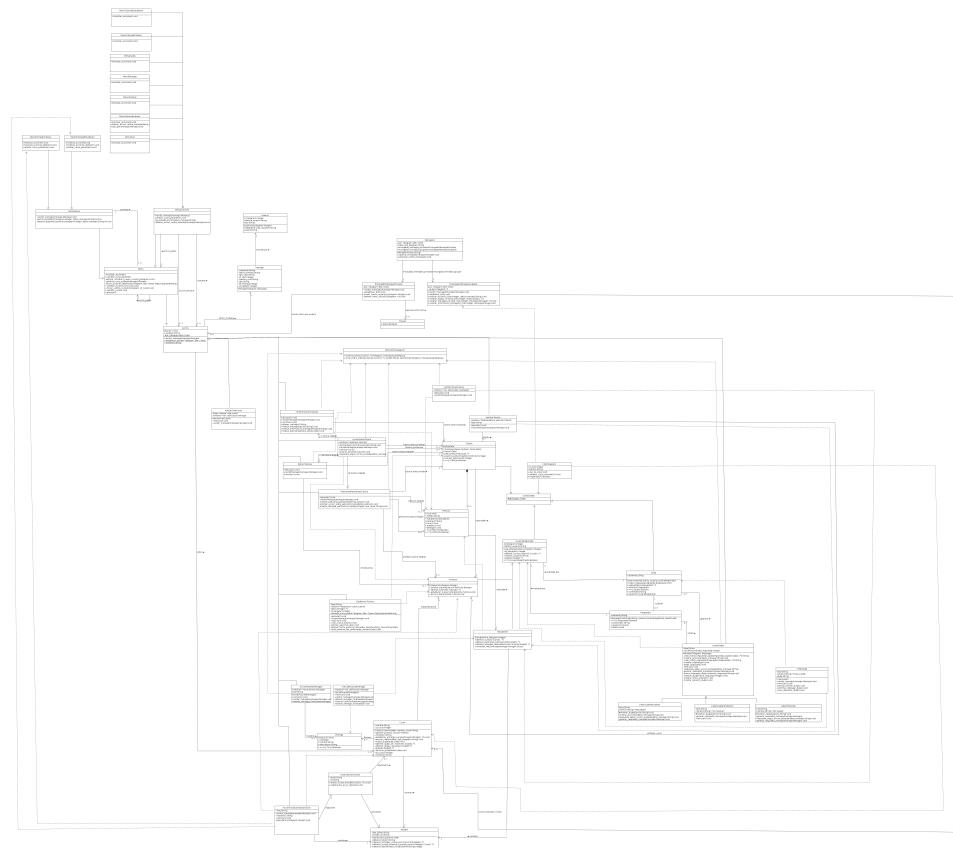


Figura 5.11: Diagrama de clases del sistema

5.1.3. Diseño de la configuración de Moodle

Moodle, como bien se mencionaba en la introducción, proporciona una API accesible a través de lo que ellos llaman *web services* que permiten especificar las funciones que se pueden llamar de la API y por parte de qué usuarios en qué contexto. A los usuarios se le asigna un rol que es al que se le autoriza a usar un protocolo para comunicarse con los *web services*, siendo también necesario autorizar individualmente a cada usuario a que usen un *web service*. Los protocolos soportados son SOAP, REST o XML-RPC, nosotros vamos a utilizar REST por ser el más sencillo de manejar.

Para el diseño de la configuración de Moodle hay que tener en cuenta lo que necesita el *bot* de Moodle:

1. *Bot* necesita que Moodle verifique que los datos que introduce el usuario al registrarse son correctos.
2. *Bot* necesita saber si el usuario que se da de alta es un profesor o un estudiante para darle acceso a una parte u otra de acciones.
3. Tiene que conocer los cursos en los que está matriculado un profesor para que cuando éste se dé de alta, poder registrar estos cursos como usables en el *bot* y, al menos, también tiene que poder saber a qué cursos registrados en el *bot* tiene acceso un estudiante en Moodle.
4. Necesita tener acceso a las entregas abiertas para los cursos que tiene registrados así como a datos como su fecha y si un estudiante lo solicita, a la nota de este estudiante.

Para conseguir configurar Moodle de forma que el *bot* pueda realizar los puntos descritos arriba y que el uso del mismo implique los menos privilegios posibles para evitar cualquier problema con la seguridad y el funcionamiento de la instancia de Moodle, he optado por un enfoque en el cual se empieza con ningún permiso e ir dando poco a poco hasta llegar a lo mínimo necesario.

Los pasos que he seguido son los siguientes:

1. **Habilitar los webservices**, para lo cual hay que irse al apartado de “Plugins” dentro de “Site Administration” buscar “Web services” y darle a habilitar.
2. **Habilitar el protocolo a utilizar**: dentro de Web services buscamos «Manage protocols» y habilitamos REST.

The screenshot shows the 'Manage protocols' page in Moodle. At the top, a green banner says 'Changes saved'. Below it, the title 'Manage protocols' is displayed. A sub-header 'Active web service protocols' is followed by a table. The table has columns for 'Protocol', 'Version', 'Enable', and 'Settings'. Three rows are listed: 'REST protocol' (version 2016120500, enable icon is a person), 'SOAP protocol' (version 2016120500, enable icon is a gear), and 'XML-RPC protocol' (version 2016120500, enable icon is a gear). A note below the table says 'For security reasons, only protocols that are in use should be enabled.' At the bottom left, there are links for 'Web services' and 'Advanced settings'.

Figura 5.12: Habilitando protocolo REST en Moodle

3. Crear tres roles con contexto de sistema con los siguientes permisos mínimos:

Nombre rol	Permisos
webservices_bot	<ul style="list-style-type: none"> ■ moodle/course:viewparticipants ■ moodle/user:viewdetails ■ moodle/user:viewhiddendetails ■ moodle/course:useremail ■ moodle/user:update
webservices_estudiante	<ul style="list-style-type: none"> ■ mod/assign:view
webservice_profesor	

Manage roles Allow role assignments Allow role overrides Allow role switches

Adding a new role ?

Create this role Cancel

Short name ? webservice_estudiante

Custom full name ? webservies_estudiante

Custom description ?

Role archetype ? None

Context types where this role may be assigned

- System
- User
- Category
- Course
- Activity module
- Block

Allow role assignments

Figura 5.13: Creando rol webservices_student

Estos permisos son los imprescindibles para poder utilizar las funciones descritas en la siguiente tabla. Si no fueran roles con contexto de sistema entonces no tendrían acceso a los *web services*.

4. Crear los siguientes webservices marcando “enable” y “Authorized users only” con las siguientes funciones cada uno:

Webservice	Funciones
webservices_bot	<ul style="list-style-type: none"> ■ core_enrol_get_users_courses: Permite al bot obtener los cursos de un usuario cuando éste se identifica ante el bot por primera vez.
webservices_estudiante	<ul style="list-style-type: none"> ■ core_user_get_users_by_field: Permite obtener el identificador de moodle del usuario recién registrado, necesario para conocer las notas de su entrega. ■ mod_assign_get_assignments: Permite al bot obtener las entregas abiertas para un curso junto con detalles como su fecha.
webservice_profesor	<ul style="list-style-type: none"> ■ mod_assign_get_assignments: Permite al estudiante ver las entregas para los cursos a los cuales tiene acceso. ■ mod_assign_get_submission_status: Un estudiante puede con su token, id de moodle e id entrega puede ver que calificación tiene en la entrega.

New Site

Dashboard / Site administration / Plugins / Web services / External services / External service

External service

Name	<input type="text" value="webservice_bot"/>
Short name	<input type="text" value="webservice_bot"/>
<input checked="" type="checkbox"/> Enabled	
<input checked="" type="checkbox"/> Authorised users only <small>(?)</small>	

Show more...

Add service Cancel

There are required fields in this form marked *.

Figura 5.14: Creando webservice llamado webservices_bot

New Site

Dashboard / Site administration / Plugins / Web services / External services / Functions

Add functions to the service "webservice_bot"

Function	Description	Required capabilities	Edit
core_enrol_get_users_courses	Get the list of courses where a user is enrolled in	moodle/course:viewparticipants	Remove
core_user_get_users_by_field	Retrieve users' information for a specified unique field - If you want to do a user search, use core_user_get_users()	moodle/user:viewdetails, moodle/user:viewhiddendetails, moodle/course:uservisit, moodle/user:update	Remove
mod_assign_get_assignments	Returns the courses and assignments for the users capability		Remove

Add functions

Figura 5.15: Añadiendo funciones a el webservice llamado webservices_bot

5. Crear un usuario en Moodle para el bot.
6. Asignar el rol de webservices_estudiante a los usuarios de moodle que quieren utilizar el bot como estudiantes, el webservices_profesor a los profesores y el webservice_bot al usuario que se ha creado para el

bot. Como son roles de sistema es necesario irse al apartado **Site Administration >Users >Assign system roles services**:

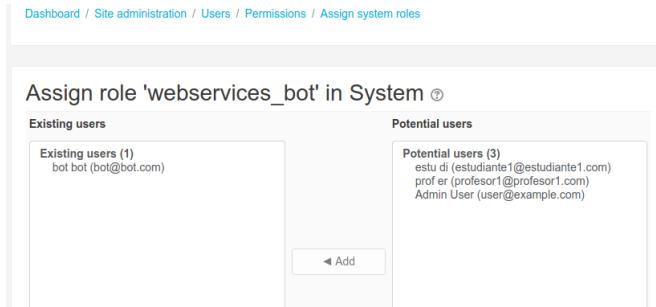


Figura 5.16: Asignado el rol webservices_bot al usuario creado en Moodle para el bot

7. Añadir a los usuarios a los que se les ha asignado como rol webservices_estudiante como usuarios autorizados a usar el recien creado webservices_estudiante, a los que tienen rol webservices_profesor añadir como usuarios autorizados al webservice webservices_profesor y al usuario del bot con rol webservices_bot añadirlo como usuario autorizado webservice_bot.

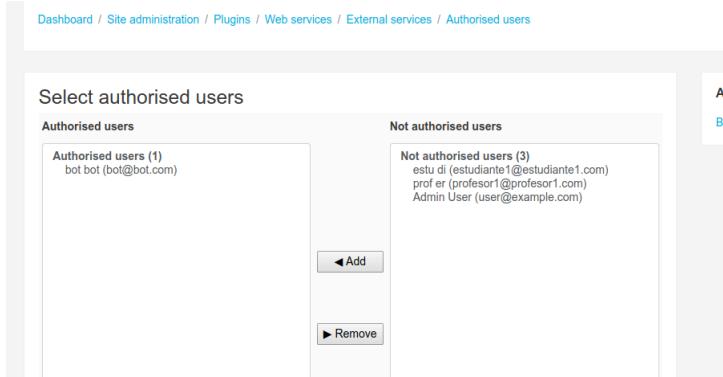


Figura 5.17: Asignado el rol webservices_bot al usuario creado en Moodle para el bot

8. Crear token para el usuario que se ha creado en Moodle para el bot. Esto se puede hacer bajo el apartado Web services buscando **Manage tokens**.

The screenshot shows the 'Manage tokens' page in Moodle. It has a header with 'Dashboard / Site administration / Plugins / Web services / Manage tokens'. Below the header is a table with columns: Token, User, Service, IP restriction, Valid until, and Operation. There is one row visible with the token '0621070a9c45', user 'bot bot', service 'webservices_bot', and an 'Delete' button.

Figura 5.18: Creando token para el usuario en Moodle del bot

9. Añadir en Moodle al usuario creado para el bot a cada curso que se quiera que tenga acceso el bot.

The screenshot shows the 'Enrolled users' page for the course 'curso7'. The title bar says 'curso7: 3 enrolled users'. The page includes a search bar, filters for group ('All participants'), status ('All'), and roles ('Student'). It lists two users: 'bot' and 'estu di'. The 'bot' user is shown with the role 'Student' and 'webservices_bot'. The 'estu di' user is also listed with the same role and role. Both users have 'Manual enrolments from Friday, 25 August 2017, 9:18 AM'.

Figura 5.19: Añadiendo al usuario bot al curso curso7

Una vez hecho esto podemos comprobar si tenemos acceso a las funciones de los *web services* a través de REST utilizando un navegador web:

The screenshot shows a browser window with two tabs. Both tabs are pointing to the URL `http://192.168.1.63/webservice/rest/server.php?wstoken=XXXXXXXXXX`. The top tab's content is a JSON response for course 5, listing course details like id, shortname, fullname, and enrollment count. The bottom tab's content is a JSON response for course 0, listing course details for 'curso0'.

```
[{"id":5,"shortname":"curso7","fullname":"curso7","enrolledusercount":3,"idnumber":"","visible":1,"summary":"","summaryformat":1,"format":"topics","showgrades":true,"lang":"","enablecompletion":false,"category":1}, {"id":2,"shortname":"curso0","fullname":"curso0","enrolledusercount":3,"idnumber":"","visible":1,"summary":"","summaryformat":1,"format":"topics","showgrades":true,"lang":"","enablecompletion":false,"category":1}]
```

Figura 5.20: Llamando a algunas de las funciones de webservices_bot a través del navegador web

Todo el proceso anteriormente descrito se puede encontrar en la sección **Site Administration >Plugins >Web services** de cualquier instancia de Moodle.

5.1.3.1. Justificación

Se han creado tres *web services* con tres roles diferentes ya que hay tres tipos de usuarios y cada uno necesita acceder a un conjunto de funciones diferentes. Si tuvieramos un solo rol y un solo *web service* entonces algunos usuarios tendrían más permisos de los necesarios y acceso a funciones de la API que no les hacen falta.

Ejemplo: al *bot* no le hace falta hacer llamadas a la función que obtiene calificaciones de la entrega y por tanto mejor no darle acceso.

Cada usuario del sistema en el momento que introduce sus datos de Moodle para empezar a usar el bot éste obtiene una token de usuario y con esa token el usuario solamente tiene acceso a su información individual (caso de los estudiante que cuando llaman a `mod_assign_get_submission_status` les devuelve datos de las entregas del usuario cuya token se utiliza para llamar al *web service*.)

Hace falta un token aparte para el *bot* porque éste necesita conocer información común de un curso como las entregas que hay abiertas, si no tiene token entonces no hay acceso. Además si no creáramos un usuario aparte para el *bot* entonces no se puede controlar a qué cursos tienen acceso los usuarios del *bot*. Obligando a añadir al bot a un curso de Moodle hacemos que los usuarios que soliciten al bot usarlo sólamente tengan acceso a aquellos cursos en los que el bot ha sido añadido. Esto es importante ya que dando rol con contexto de sistema a un usuario hacemos que tenga acceso con su token a todos los cursos en los que están matriculados en Moodle (limitado a las funciones permitidas en su webservice y a los permisos de su rol) tanto si el profesor responsable de ese curso quiere o no quiere utilizar los webservices de Moodle para ese curso.

5.1.4. Diseño de la base de datos

La estructura lógica de la base de datos se puede observar en el siguiente diagrama ER:

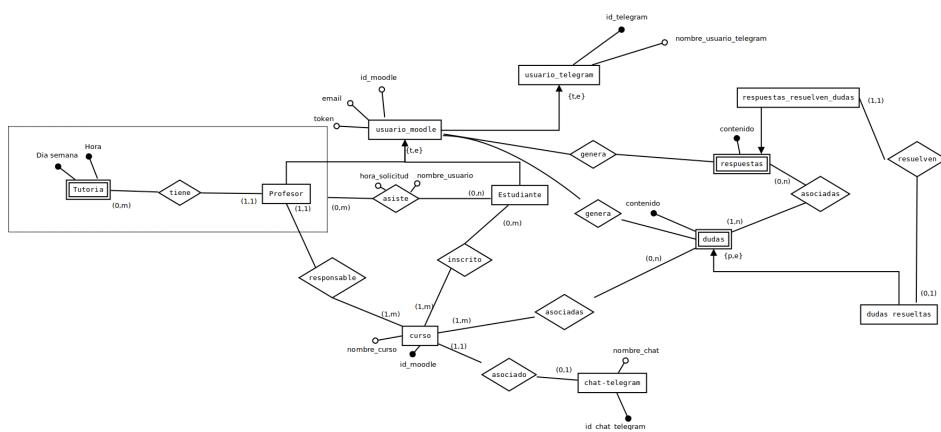


Figura 5.21: Diagrama ER de la base de datos

Se pueden apreciar las necesidades de almacenamiento que va a tener nuestra aplicación y podemos extraer el esqueleto de la base de datos con sus relaciones, claves externas, primarias...

Tras realizar el paso a tablas de este diagrama y efectuar el paso a la tercera forma normal obtenemos el siguiente esquema de la base de datos:

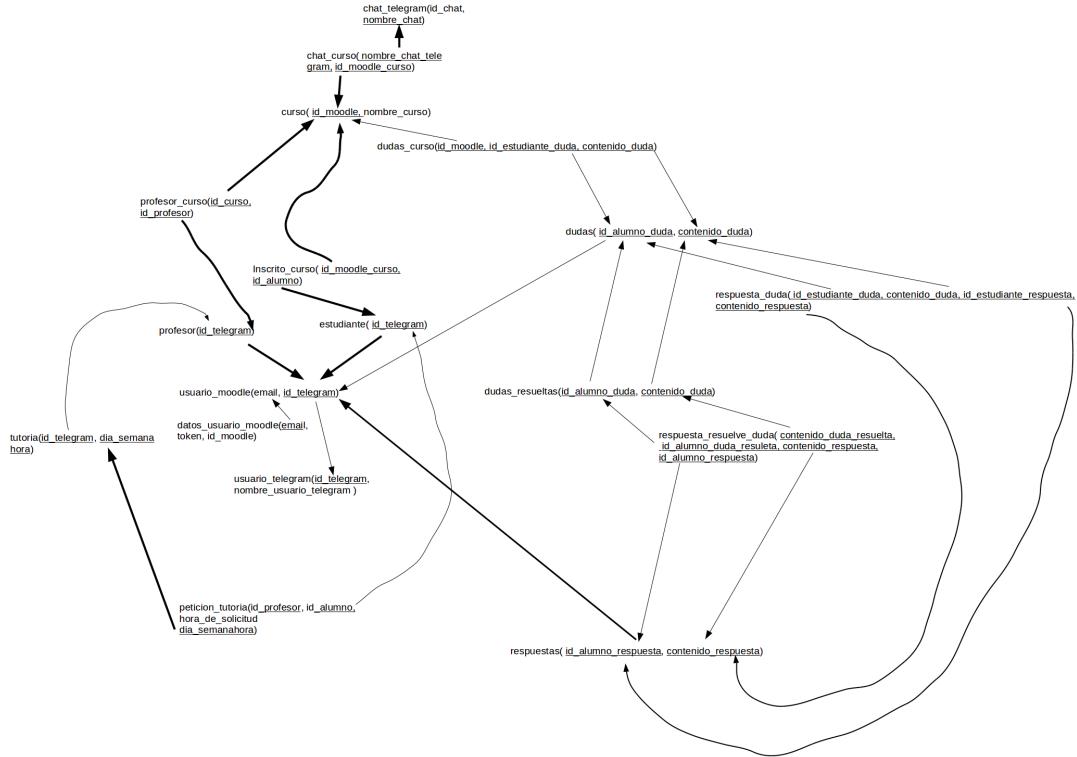


Figura 5.22: Esquema base datos en tercera forma normal

5.1.5. Diseño de los tests

En el desarrollo de la aplicación se está siguiendo la programación orientada a objetos como paradigma de programación. Para verificar el correcto funcionamiento de las clases que la componen se hace uso de test unitarios que prueban la “interfaz” o parte pública de la clase proporcionándole datos de prueba y comprobando si los resultados devueltos por ésta son los esperados.

A la hora de diseñar estos tests unitarios hay que tener en cuenta que básicamente hay dos tipos de clases: aquellas que reciben un mensaje y realizan algo con este (bien sea extraer los datos de éste y realizar una acción o entregar el mensaje a la clase responsable) y las que acceden a la

base de datos y Moodle. Para los tests de las primeras se hará uso de stubs para simular clases con una serie de métodos predefinidos que devuelven datos ideales y de mocks para monitorizar si la clase bajo prueba realiza las acciones deseadas cuando se le entrega un mensaje con unas características predeterminadas.

Para las segundas debido a la dificultad de simular una base de datos se optará por crear una base de datos “desechable” en la cual se introducen una serie de datos prefijados y se prueba el comportamiento de la interfaz pública de estas clases, bien llamando a sus métodos públicos y comprobando el resultado devuelto, o bien comprobando el efecto sobre la base de datos.

Capítulo 6

6.1. Implementación

6.1.1. Programa

A la hora de implementar el programa una variable importante es la facilidad con la que se puede trabajar con la API para bots de Telegram. Debido a ello, y como quería que el proyecto fuera el desarrollo de pradobot y no de un programa para trabajar con la API de Telegram, opté por el uso de una librería que facilitara la tarea. En el momento de empezar la implementación las más populares eran:

- Para NodeJS: <https://github.com/yagop/node-telegram-bot-api>
- Para Python: <https://github.com/python-telegram-bot/python-telegram-bot>
- Para Golang: <https://github.com/go-telegram-bot-api/telegram-bot-api>
- Java: <https://github.com/rubenlagus/TelegramBots>
- Ruby: <https://github.com/atipugin/telegram-bot-ruby>

Teniendo conocimiento de Ruby y habiendo probado en alguna asignatura el correcto funcionamiento de la librería de Ruby, decidí usarla. Esta librería te permite hacer uso de los métodos de la API para bots de Telegram con un formato *camel_case* es decir las palabras de los métodos en minúscula y separadas por una barra baja. Por lo demás acepta los mismos parámetros:

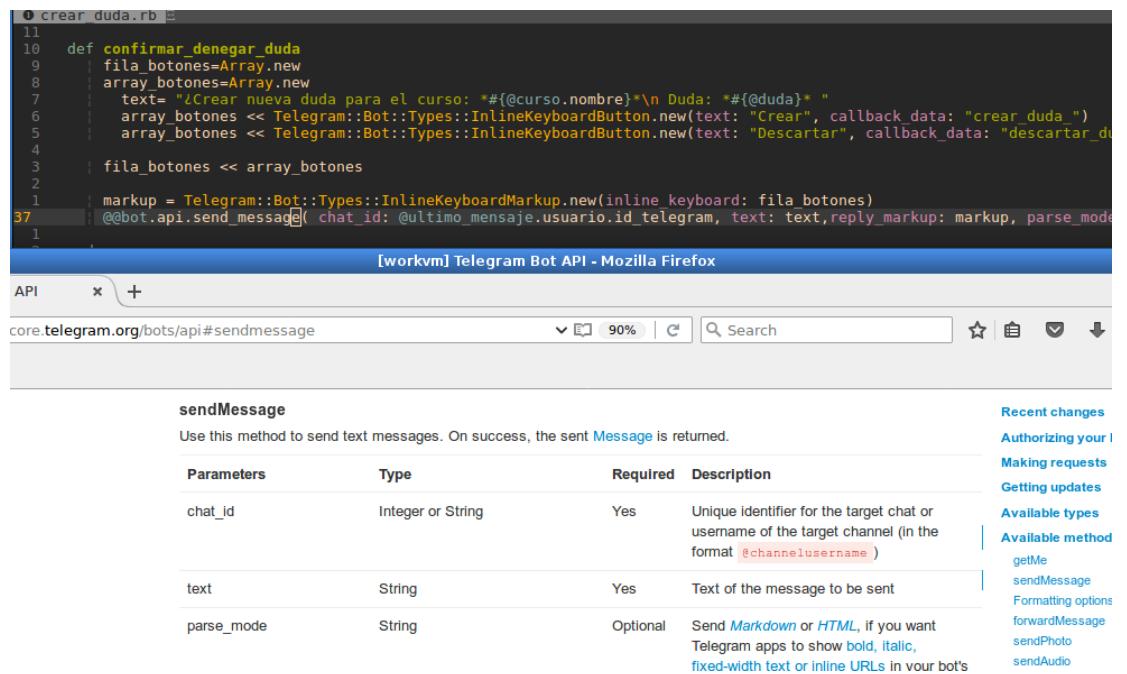


Figura 6.1: Uso del método SendMessage por parte del bot

El único inconveniente que tiene es que solamente puede haber una instancia del cliente que recibe mensajes. Esto es debido a que para que un bot de Telegram reciba los mensajes que le llegan tiene dos alternativas *long polling* y *Webhook* :

- *Long polling*: El bot periódicamente llama a los servidores de Telegram utilizando un método que devuelve los 100 primeros mensajes sin contestar que tiene el bot. Más lento pero mucho más fácil de utilizar.
- *Webhook*: Hace falta implementar un servidor con una IP pública o un nombre de dominio e indicarle a Telegram que cuando se le manda un mensaje al bot éste lo redirija a este servidor. Rápido pero más complicado de montar.

El bot ha sido implementado haciendo uso de long polling y la implementación que realiza la librería utilizada solamente deja que haya una instancia del bot **escuchando**:

```

① test_rapidez_mensaje.rb ② mensajero.rb ③ encargado_mensajes_pri
  9  #
  8  #Recibe los mensajes de parte de Telegram destinados al bot
  7  #
  6  def empezar_recibir_mensajes
  5    @bot.run(@token_bot_telegram) do |botox|
  4      Accion.establecer_bot(botox)
  3      @encargado_mensajes_privados.establecer_bot(botox)
  2      @encargado_mensajes_grupales.establecer_bot(botox)
  1      botox.listen do |message|
  55    begin
  1      mensaje=Mensaje.new(message)
  2      clasificar_mensaje(mensaje)
  3    end
  4    end
  5  end
  6
  7
  8
  9 end

```

Figura 6.2:

Lo cual a efectos prácticos implica que solamente puede haber un único

:

```

botox.listen do |message|
  begin
    .....
  end
end

```

Una vez tienes el programa escrito en ruby que es capaz de obtener los mensajes que se le envían al bot por Telegram y de interactuar con los usuarios quedan dos partes más: la base de datos y la instancia de Moodle.

6.1.1.1. Git-Github

Con la intención de que la construcción de la aplicación fuera lo más transparente posible y de dar la posibilidad de que otras personas pudieran contribuir a ella, se ha hecho uso de la plataforma de desarrollo colaborativo Github y del sistema de control de versiones Git durante todo el proceso de implementación de la aplicación.

Github permite, entre otras cosas, ver los cambios realizados en el código

desde el primer momento, la creación de *issues* que pueden ser utilizados para indicar algún fallo en la aplicación y junto con git permite tener varias versiones del código.

Todo el código del programa se puede encontrar en <https://github.com/LuisGi93/pradobot>.

6.1.2. Base de datos

Como base de datos he utilizado PostgreSQL principalmente porque es *opensource*, fácil de instalar y no he visto en ningún sitio que tenga diferencias de rendimiento significativas comparada con otras bases de datos importantes. Aún así debido al uso de una librería de ruby llamada Sequel, que hace de adaptador entre la aplicación y la base de datos, se podría utilizar cualquier otra base de datos.

Sequel hace uso de ORM para la comunicación con la base de datos. También ofrece una capa de abstracción entre aplicación y el manejo de todo lo relacionado con la interacción sobre la base de datos. Algunas de las características relevantes para su uso:

- Ofrece configurar el número de conexiones disponibles para acceder a la base de datos.[15]
- Una instancia de la “conexión” con la base de datos hace uso de manera transparente para el programador de múltiples hebras para utilizar las conexiones disponibles de la base de datos. Lo cual implica que una misma variable puede ser usada a lo largo del programa para realizar consultas con la base de dato sin que una consulta paralice al programa ya que serán ejecutadas por diferentes hebras.
- Las conexiones descritas en el primer punto se levantan en el momento en que se lanza la primera consulta [16] quedando latentes a la espera de ser usadas por las siguientes consultas. Es decir no se abre una sesión con la base de datos, se realiza la consulta y se cierra la conexión, sino que está diseñado para que una misma conexión sea usada por diferentes consultas. Una correcta configuración del número de conexiones que se quiere que se tengan disponibles para la base de datos y del número de hebras a usar sobre esas conexiones es importante para un uso óptimo de la base de datos. Pradobot es un prototipo, y si se quisiera utilizar en un ambiente “real”, esta sería una de las cosas que habría que mirar con lupa.
- La abstracción que te ofrece el uso de ORM simplifica mucho el trabajar con la base de datos.

6.1.3. Moodle

Durante el desarrollo del programa se ha utilizado la versión 3.2.1 de Moodle. Aunque también se ha probado con la versión 3.0.10 y sea posiblemente la versión mínima soportada por el programa ya que las anteriores versiones no tienen algunas de las funciones de la API de Moodle utilizadas por el programa.

6.1.4. Despliegue

Para facilitar la instalación de la aplicación incluimos un fichero “Vagrantfile” que permite la creación de una máquina virtual con Ubuntu 14.04 en Amazon AWS con pradobot y todas sus dependencias instaladas en ella. El contenido del Vagrantfile es el siguiente:

Para poder utilizar el Vagrantfile necesitamos tres programas:

1. Instalar programa vagrant para nuestra distribución.
2. aws-cli: programa que nos permite conectarnos con nuestra cuenta en amazon aws y que genera credenciales para Vagrantfile (pip install awscli) .
3. Plugin aws para vagrant (vagrant plugin install vagrant-aws)

El contenido de un Vagrantfile es similar al siguiente:

```
Vagrant.configure("2") do |config|
  config.vm.box = "dummy"

  config.vm.define "pradobot-aws" do |host|
    host.vm.hostname = "pradobot-aws"
  end
  config.vm.provider :aws do |aws, override|
    aws.access_key_id = "xxx"
    aws.secret_access_key = "xxx"
    aws.session_token = "xxx"
    aws.keypair_name = "pradobotllave"
    aws.region= "us-west-2"
    aws.security_groups = "migruposeguro"
    aws.instance_type= 't2.micro'

    aws.ami = "ami-d57dcfb5"
```

```

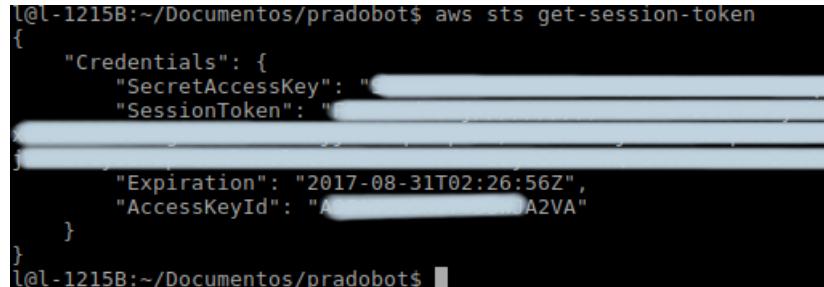
override.ssh.username = "ubuntu"
override.ssh.private_key_path = "pradobotllave.pem"
end

config.vm.provision :ansible do |ansible|
  ansible.playbook = "pradobot.yml"
  ansible.force_remote_user= true
end
end

```

El Vagrantfile cuenta con los siguientes elementos:

- **aws.secret_access_key, aws.session_token:** Claves efímeras que nos autorizan para crear máquinas virtuales en nuestra cuenta durante 48 horas. Generadas utilizando aws-cli:



```

l@l-1215B:~/Documentos/pradobot$ aws sts get-session-token
{
    "Credentials": {
        "SecretAccessKey": "XXXXXXXXXXXXXX",
        "SessionToken": "A3VAA2VA",
        "Expiration": "2017-08-31T02:26:56Z",
        "AccessKeyId": "AXXXXXXXXXXXXXXA2VA"
    }
}
l@l-1215B:~/Documentos/pradobot$ 

```

Figura 6.3: Utilizando aws-cli para generar claves efímeras para el despliegue

- **aws.keypair_name:** Generada utilizando:

```

aws ec2 create-key-pair --key-name pradobotllave
--query 'KeyMaterial' --output text >
pradobotllave.pem

```

- **aws.security_groups:** Grupo de seguridad en Amazon AWS en podemos especificar que puertos de entrada/salida están habilitados en la máquina virtual y que IPs aceptamos.
- **aws.region,aws-instance-type,aws.ami:** Especifican la región, las características del servidor donde se ejecuta la máquina virtual y el sistema operativo.
- Por último **config.vm.provision** especifica que queremos usar Ansible para aprovisionar la máquina virtual:

- **ansible.playbook**: script que utilizará ansible para aprovisionar la máquina virtual.

6.1.5. Aprivisionamiento

El aprovisionamiento de la máquina virtual creada por vagrant se hace utilizando el programa Ansible. Ansible hace uso de ssh y de un tipo de archivos llamados “playbooks” en las cuales se especifican las órdenes necesarias para el aprovisionamiento de la máquina virtual. El playbook utilizado por pradobot es el siguiente:

```
---
---
- hosts: all
  user: ubuntu
  sudo: yes
  roles:
    - { role: rvm_io.ruby,
        tags: ruby,
        rvm1_rubies: ['ruby-2.3.1'],
        rvm1_user: 'ubuntu',
        become: true
    }
  tasks:
#Por alguna razon instala todos los paquetes menos libpq-dev
  - name: Instalamos paquetes necesarios
    become: true
    action: >
      {{ ansible_pkg_mgr }} name= {{ item }} state=installed
      update_cache=yes
    with_items:
      - build-essential
      - ruby-dev
      - libpq-dev
      - ruby
      - git
      - libgdbm-dev
      - libncurses5-dev
```

```

- automake
- libtool
- bison
- libffi-dev

- name: instalamos git
  apt:
    name: git
    state: present

- name: instalamos libpq-dev
  apt:
    name: libpq-dev
    state: present

- name: clonamos repo de la web
  become: true
  become_user: ubuntu
  shell: git clone https://github.com/LuisGi93/pradobot.git
  args:
    creates: pradobot
    executable: /bin/bash

- name: Instalamos el proyecto
  become: true
  become_user: ubuntu
  shell: source ~/.rvm/scripts/rvm && bundle install
  args:
    chdir: pradobot
    executable: /bin/bash

      - name: Creamos las tablas de la base de datos
        become: true
        become_user: ubuntu
        shell: source ~/.rvm/scripts/rvm && export
          URL_DATABASE_TRAVIS="" && ruby
          config/primer_inicio_aplicacion.rb
        args:
          chdir: pradobot
          executable: /bin/bash

```

En el cual podemos distinguir las siguientes partes relevantes:

- **hosts, user:** Indicamos el nombre de la máquina a la que se va a conectar Ansible utilizando ssh.
- **user:** nombre del usuario con el cual se inicia sesión en dicha máquina utilizan ssh.
- **sudo:** Indicamos si se va utilizar sudo o no.
- **roles:** Indicamos que para ejecutarse este playbook se necesitan una serie de archivos, en este caso rvm que explico más abajo para que sirve.
- **tasks:** Tareas a ejecutar por Ansible secuencialmente, algunos de los elementos que pueden tener son:
 - **name:** nombre descriptivo.
 - **become:** implica el uso de sudo.
 - **become_user:** implica el usuario a utilizar.
 - **shell:** el comando se tiene que ejecutar en una shell.
 - **apt:** requiere el uso del administrador de paquetes.

El script en primer lugar instala los paquetes necesarios, tras lo cual clona el proyecto desde github creando el directorio pradobot y procede a instalar las dependencias de pradobot utilizando RVM y Bundler.

RVM es un programa que permite la ejecución de otro programa escrito en ruby en el entorno de ruby deseado, en mi caso pradobot necesita la versión 2.3.1 de ruby que no se encuentra en los repositorios de Ubuntu. Bundler automatiza la instalación de las librerías que necesita una aplicación de ruby para lo cual hace uso de un archivo llamado Gemfile:

```
source "https://rubygems.org"
gem 'sequel'
gem 'pg'
gem 'rake'
gem 'rspec'
gem 'telegram-bot-ruby'
gem 'json'
gem 'rufus-scheduler'
gem 'activesupport'
gem 'daemons'
gem 'typhoeus'
group :test do
  gem 'rake'
end
```

En el se especifican las librerías a obtener, su versión si se quiere y de donde obtenerlas. Por último inicia la ejecución de pradobot.

6.1.6. Control ejecución

Una vez hemos creado la máquina con Vagrant y haberla aprovisionado utilizando Ansible, tenemos que poder controlar el inicio y fin de ejecución de la aplicación de manera automática. Para ello vamos a utilizar una herramienta llamada Capistrano. Capistrano permite la ejecución de comandos remotos a través de ssh. En primer lugar lo instalamos haciendo `gem install capistrano` tras lo cual ejecutamos la orden `cap install` en el directorio de nuestro proyecto. Esto provocará la generación de una serie de archivos. Buscamos “config/deploy.rb” y le añadimos información para la autenticación via ssh:

```
set :ssh_options, {
  forward_agent: true,
  auth_methods: ["publickey"],
  keys: ["path_a_llave_pem/pradobotllave.pem"]
}
```

En Capistrano se definen ”tareas” donde defines que es lo que quieres que haga capistrano. Para establecer la tarea creamos un fichero en lib/capistrano/tasks acabado en .rake y en el defino la siguiente tarea:

```
require "resolv-replace.rb"

role :yoquese, "yoquese"

namespace :pradobot do
  desc "Iniciamos la aplicacion"

  task :daemon_start do
    on "ubuntu@ip.us-west-2.compute.amazonaws.com" do
      execute 'source ~/.rvm/scripts/rvm && export
TOKEN_BOT_MOODLE="" && export MOODLE_HOST="" &&
export TOKEN_BOT="" && export URL_DATABASE_TRAVIS=""
&& nohup ruby pradobot/bin/run.rb && true & '
    end
  end
end
```

```

desc "Paramos la aplicacion"
task :daemon_stop do
  on "ubuntu@ip.us-west-2.compute.amazonaws.com" do
    execute 'pkill ruby'
  end
end
end

```

Con lo cual podemos ejecutar el comando `cap production pradobot:daemon_start` para iniciar su ejecución y pararlo utilizando `cap production pradobot:daemon_stop`.

Para el inicio de la ejecución se hace uso del comando nohup, que desliga a un proceso de la terminal de tal forma que cuando se lanza el programa en segundo plano (`ruby pradobot/bin/run.rb && true &`) su ejecución no termina al finalizar la ejecución de la orden por ssh.

6.1.7. Tests

Los tests se han implementado utilizando Rake. Rake es una herramienta que permite entre otras cosas ejecutar tests de ruby para lo cual necesita disponer de un archivo llamado Rakefile en el cual se le indica dónde puede encontrar los tests y qué pasos hay que dar para ejecutarlos. El Rakefile de pradobot es el siguiente:

```

require 'rake/testtask'
require 'rspec/core/rake_task'
require_relative 'config/crear_tablas_bd'
require 'sequel'

namespace :tasks do
  namespace :db do
    namespace :test do

      desc "Creamos base de datos test"
      task :crear do
        db=Sequel.connect(ENV['URL_DATABASE'])
        begin
          db.run "CREATE DATABASE bd_prueba"
        rescue Sequel::Error
          db.run "DROP DATABASE bd_prueba"
        end
      end
    end
  end
end

```

```

    db.run "CREATE DATABASE bd_prueba"
  end
  db.disconnect
  db=Sequel.connect(ENV['URL_DATABASE']+ '/bd_prueba')
  crear_tablas(db)
  db.disconnect
  ENV['URL_DATABASE_ORIGINAL']=ENV['URL_DATABASE']
  ENV['URL_DATABASE']=ENV['URL_DATABASE_PRUEBA']
end

RSpec::Core::RakeTask.new(:tests_bd) do |t|
  t.pattern = Dir.glob('test/test_*.rb')

  t.rspec_opts = '--format documentation'
end

desc "Borramos la base de datos"
task :destruir do
  ENV['URL_DATABASE']=ENV['URL_DATABASE_ORIGINAL']
  db=Sequel.connect(ENV['URL_DATABASE'])
  db.run "DROP DATABASE bd_prueba"
  db.disconnect
end

end

end

desc "Ejecutamos los test sobre la base de datos"
task :default => ['tasks:db:test:crear',
  'tasks:db:test:tests_bd', 'tasks:db:test:destruir' ]

```

En él se especifican las tareas que tiene que ejecutar Rake:

- **task :default:** Es la tarea que se llama por defecto subdividida en tres tareas:
 1. **tasks:db:test:crear :** Crea una base de datos de prueba e inser-

tamos las tablas que componen la aplicación.

2. **tasks:db:test:tests_db** : Indica a Rake que ejecute como tests todos los archivos contenidos en la carpeta test/ y que empiecen por test_.
3. **tasks:db:test:destruir**: Destruye la base de datos creada en el paso 1.

Para la realización de los tests he hecho uso de la librería RSpec que proporciona las herramientas necesarias para la realización de los mocks y los stubs. Un ejemplo de test puede ser:

```
describe CrearDuda do
  before(:each) do
    @stub_bot = double('bot')

    Accion.establecer_bot(@stub_bot)

    allow(@stub_mensaje).to receive_message_chain(:usuario,
      :id_telegram) { 66 }

    @stub_curso = double('curso')
    allow(@stub_curso).to receive(:nombre) { 'nombre del
      curso' }
    allow(@stub_curso).to receive(:id_moodle) { 5 }
    stub_padre = double('accion_padre')
    allow(stub_padre).to receive(:cambiar_curso)
    allow(stub_padre).to receive(:cambiar_curso_parientes)
    allow(stub_padre).to receive(:curso) { @stub_curso }
    @accion = MenuDudas.new(stub_padre)
    @accion.cambiar_curso(@stub_curso)
  end

  it 'Cuando recibe el primer mensaje muestra mensaje
    explicativo de que realiza' do
    allow(@stub_mensaje).to receive(:datos_mensaje) { 'Nueva
      duda' }
    expect(@stub_bot).to receive_message_chain(:api,
      :send_message) { |arg1|
      arg1.keys.should_not include(:reply_markup)
      arg1[:text].should eq("Escriba a continuaci\otilden la
        duda que desea crear relacionada con *nombre del
        curso*: \n")
    }
  end
end
```

```

@accion.recibir_mensaje(@stub_mensaje)
end

it 'Debe mandar mensaje que incluya el texto de la duda y
el curso con opciones para confirmar la creaci\otilden
de la duda' do
  allow(@stub_mensaje).to receive(:datos_mensaje) { 'Nueva
    duda' }

  allow(@stub_bot).to receive_message_chain(:api,
    :send_message)
  @accion.recibir_mensaje(@stub_mensaje)

  allow(@stub_mensaje).to receive(:datos_mensaje) {
    'Contenido nueva duda' }

  expect(@stub_bot).to receive_message_chain(:api,
    :send_message) { |arg1|
    arg1.keys.should include(:chat_id, :text, :reply_markup)
    arg1[:text].should include('nombre del curso')
    arg1[:text].should include('Contenido nueva duda')
    arg1[:reply_markup].should
      be_instance_of(Telegram::Bot::Types::InlineKeyboardMarkup)
  }
  @accion.recibir_mensaje(@stub_mensaje)
end

.....
end

```

Al principio se define stub_bot que actúa a la vez como mock y stub para, a continuación, especificarse los métodos que tiene que aceptar mediante la orden `allow(stub).to receive(nombre método)[datos respuesta]`.

En esta fragmento de código podemos encontrar dos tests que vienen compredidos entre el `it .. do` y el `end` que lo cierra. En los dos tests mostrados se definen mediante la orden `expect(stub).to receive(nombre método)[argumentos]` las expectativas del test.

- En el primero el objeto stub_bot espera que se llame al método `stub_bot.api.send_message` que se le pase como argumento un objetivo tipo hash que entre sus llaves no tenga una llamada `:reply_markup` y sí una llave llamada `text` que tenga el valor de `.Escriba a continuación la duda que desea crear relacionada con *nombre del curso*: ".`

- El segundo test es parecido solo que ahora se observa el comportamiento de la clase cuando ha recibido su segundo mensaje y se comprueba qué le manda al bot como respuesta de este segundo mensaje.

Por último para ver la cobertura sobre el código del programa que tienen nuestros tests y comprobar en qué puntos de la aplicación faltan pruebas unitarias por realizar se ha hecho de una librería llamada **SimpleCov**.

Capítulo 7

7.1. Pruebas

7.1.1. Prueba despliegue

Para probar la creación de la máquina virtual en Amazon AWS vamos a mostrar los resultados de ejecutar el Vagrantfile mostrado anteriormente. Es necesario ejecutar la orden `vagrant up --provider=aws`:

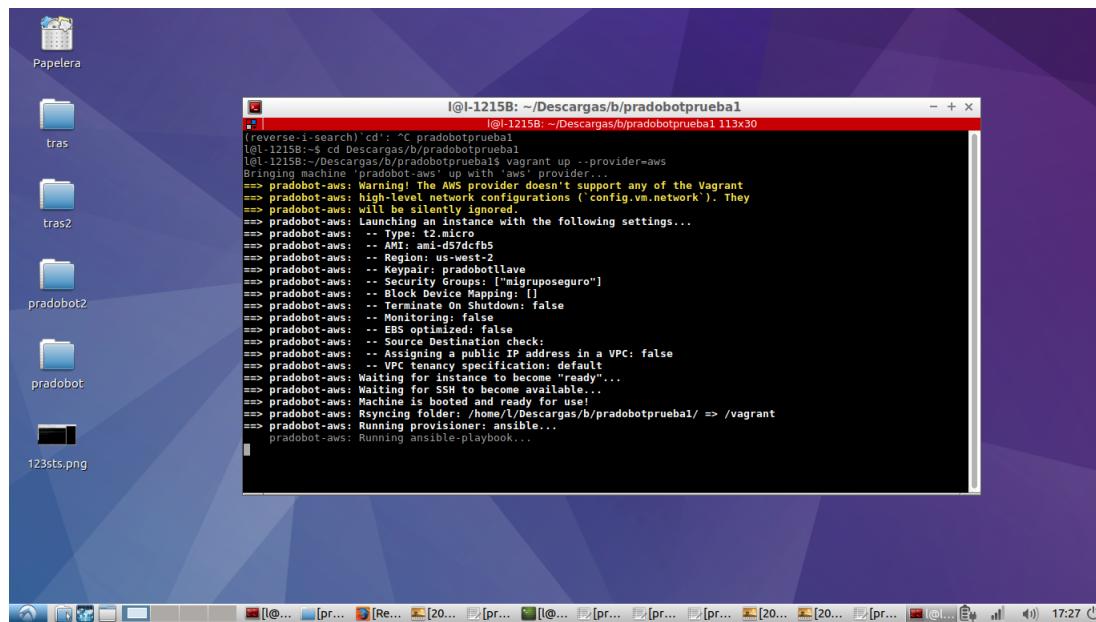


Figura 7.1: Ejecutando los tests de la aplicación

En la salida de la orden se pueden ver las características de la máquina virtual creada tras lo cual una vez está habilitado ssh se ejecuta el playbook de Ansible:

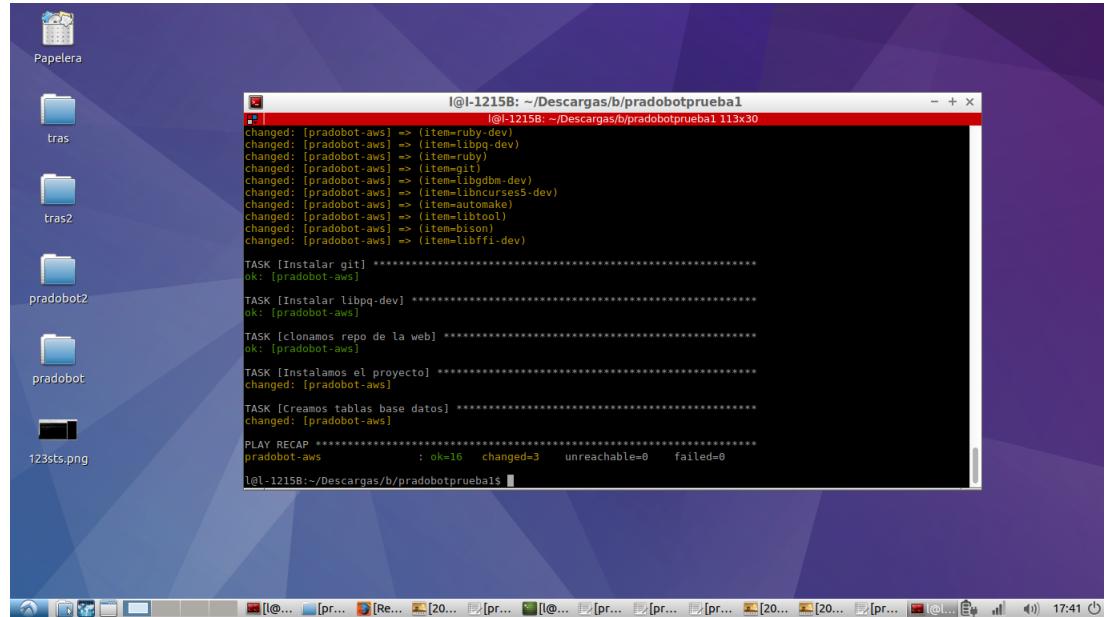


Figura 7.2: Finalización de aprovisionamiento con Ansible

Por último ejecutamos Capistrano para iniciar y parar la aplicación:

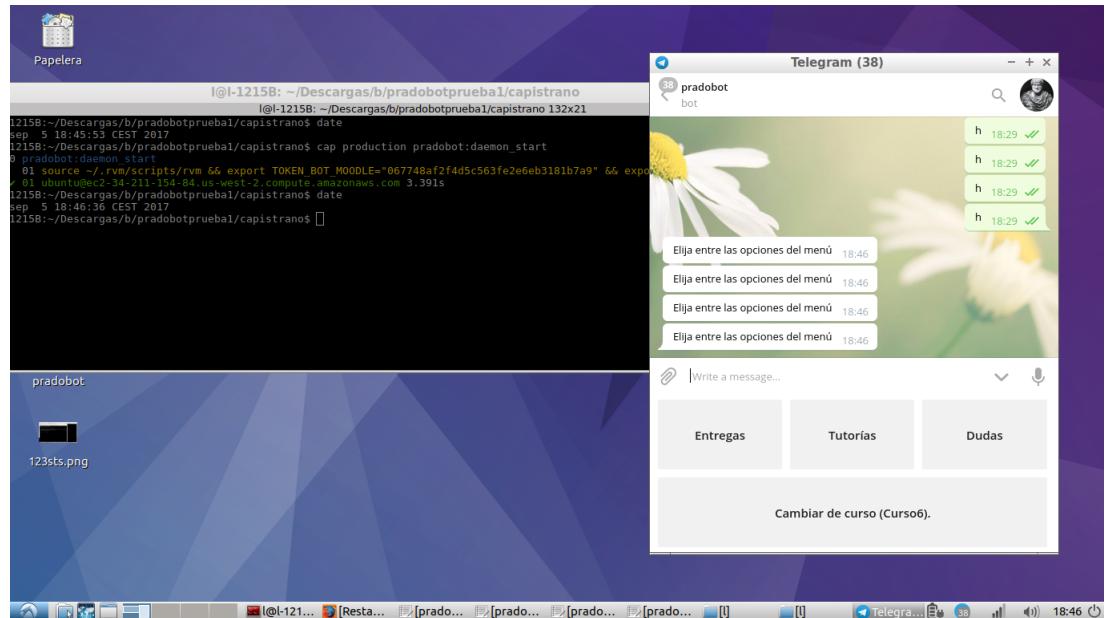


Figura 7.3: Iniciamos la ejecución de la aplicación utilizando Capistrano

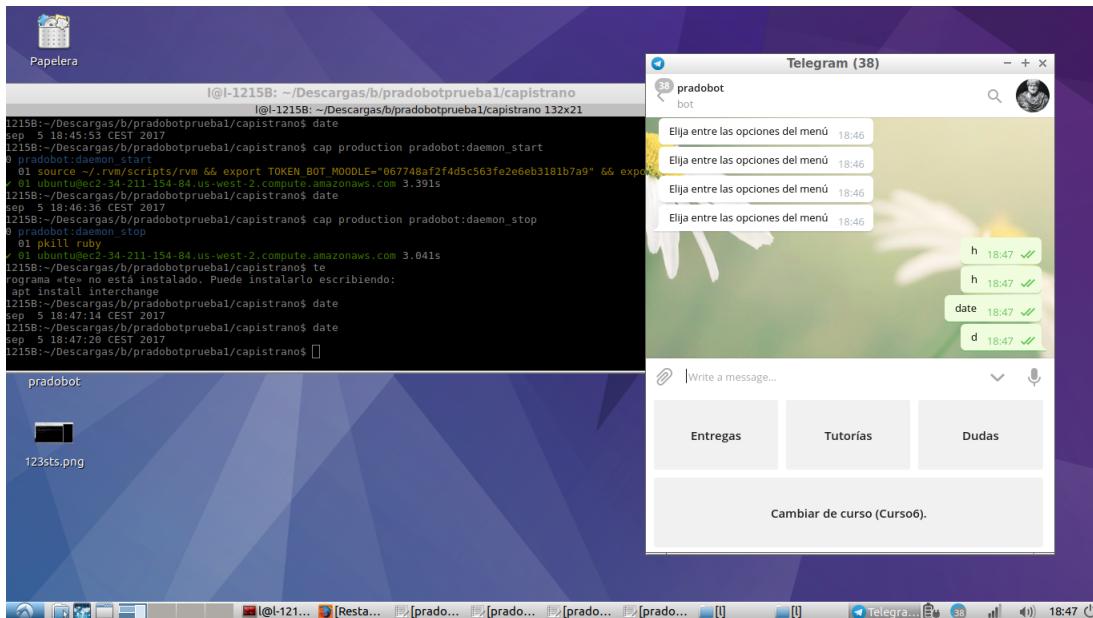


Figura 7.4: Detenemos la ejecución de la aplicación

Capistrano puede que al hacer el `cap production pradobot:daemon_start` se quede paralizado siendo necesario hacer un control-C para que se salga de la terminal. La aplicación se seguirá ejecutando normalmente.

7.1.2. Tests

Para ejecutar los tests unitarios que componen la aplicación es necesario ejecutar la orden `rake` en el directorio en el cual se encuentra el Rakefile. Esto hará que rake lea los ficheros especificados en el Rakefile y ejecute los tests contenidos en ellos uno a uno:

```
Finished in 1 minute 52.29 seconds (files took 0.59027 seconds to load)
156 examples, 0 failures

Coverage report generated for Unit Tests to /home/user/Documents/Universidad/TFG/pradobot/coverage. 3253 / 3493 LOC (93.13%) covered.
user@workvm:~/Documents/Universidad/TFG/pradobot$
```

Figura 7.5: Tras ejecutar los tests de la aplicación

Como podemos ver en la imagen superior se muestra en verde cada test que ha pasado satisfactoriamente dando al final un resumen del número de tests ejecutados (156 examples) y de cuántos han fallado (0 failures) indicando además cuánto tiempo se han tardado en ejecutar los tests. También se muestra el porcentaje de código que cubren nuestros tests (93.13 %) ge-

nerandose un reporte en formato html que permite ver qué líneas de código no se han cubierto:

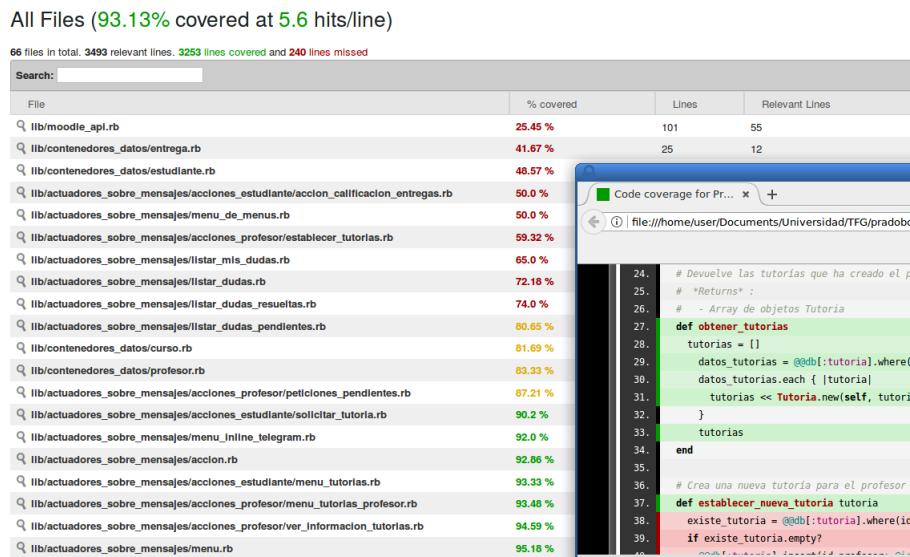
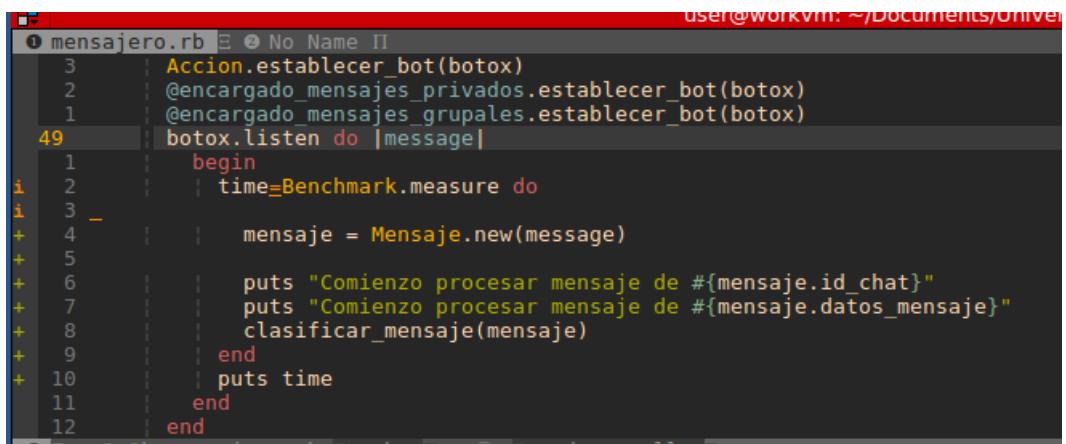


Figura 7.6: Reporte de cobertura de código de los tests

7.1.3. Rendimiento

Realizar test de rendimiento sobre el programa es complejo, ya que cualquier mensaje que se le mande al bot implicará llamadas a la base de datos y hacer mocks y stubs solamente es práctico si se va a realizar un test sobre una parte del sistema. Aún así vamos a probar a mandar varios mensajes desde chats privados y grupales midiendo el tiempo que se tarda en procesar los mensajes. El código utilizado en la medición es el siguiente:



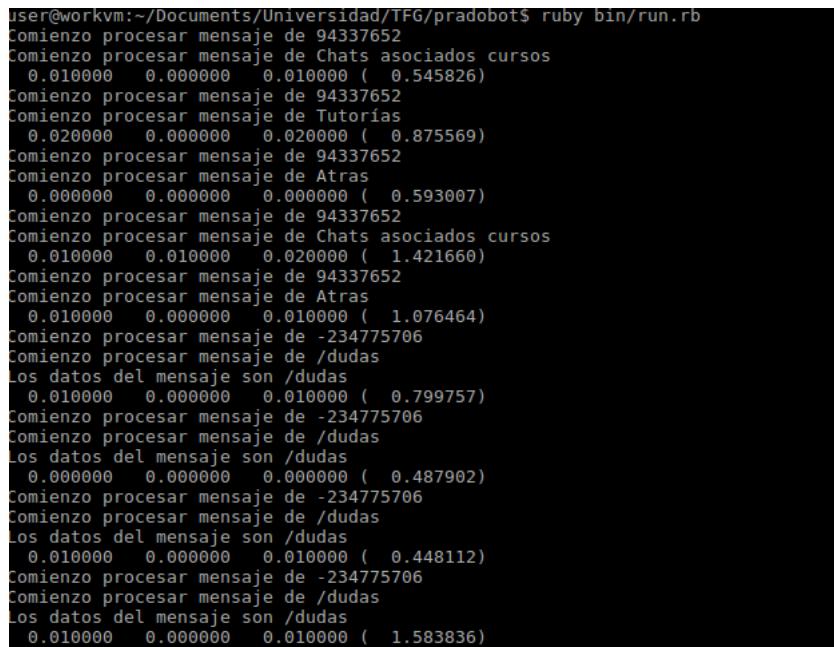
```

user@workvm: ~/Documents/Universidad/TFG/pradobot$ ruby bin/run.rb
1 mensajero.rb E @ No Name II
  3   | Accion.establecer_bot(botox)
  2   | @encargado_mensajes_privados.establecer_bot(botox)
  1   | @encargado_mensajes_grupales.establecer_bot(botox)
49  | botox.listen do |message|
  1   |   begin
i  2   |     time=Benchmark.measure do
i  3   |
+  4   |       mensaje = Mensaje.new(message)
+  5   |
+  6       puts "Comienzo procesar mensaje de #{mensaje.id_chat}"
+  7       puts "Comienzo procesar mensaje de #{mensaje.datos_mensaje}"
+  8       clasificar_mensaje(mensaje)
+  9       end
+ 10      puts time
  11    end
  12  end

```

Figura 7.7: Código utilizado para la medición del tiempo que se tarda en procesar un mensaje

Se ha utilizado la función `measure` del módulo de benchmarking de ruby `Benchmark`. Esta imprime el tiempo de CPU en espacio de usuario (*CPU time*), tiempo en el kernel realizando operaciones para nuestro programa (*system CPU time*), la suma de los dos y después el tiempo que se ha empleado en total (*elapsed real time*). Los tiempos obtenidos son los siguientes:



```

user@workvm:~/Documents/Universidad/TFG/pradobot$ ruby bin/run.rb
Comienzo procesar mensaje de 94337652
Comienzo procesar mensaje de Chats asociados cursos
  0.010000  0.000000  0.010000 ( 0.545826)
Comienzo procesar mensaje de 94337652
Comienzo procesar mensaje de Tutorías
  0.020000  0.000000  0.020000 ( 0.875569)
Comienzo procesar mensaje de 94337652
Comienzo procesar mensaje de Atras
  0.000000  0.000000  0.000000 ( 0.593007)
Comienzo procesar mensaje de 94337652
Comienzo procesar mensaje de Chats asociados cursos
  0.010000  0.010000  0.020000 ( 1.421660)
Comienzo procesar mensaje de 94337652
Comienzo procesar mensaje de Atras
  0.010000  0.000000  0.010000 ( 1.076464)
Comienzo procesar mensaje de -234775706
Comienzo procesar mensaje de /dudas
Los datos del mensaje son /dudas
  0.010000  0.000000  0.010000 ( 0.799757)
Comienzo procesar mensaje de -234775706
Comienzo procesar mensaje de /dudas
Los datos del mensaje son /dudas
  0.000000  0.000000  0.000000 ( 0.487902)
Comienzo procesar mensaje de -234775706
Comienzo procesar mensaje de /dudas
Los datos del mensaje son /dudas
  0.010000  0.000000  0.010000 ( 0.448112)
Comienzo procesar mensaje de -234775706
Comienzo procesar mensaje de /dudas
Los datos del mensaje son /dudas
  0.010000  0.000000  0.010000 ( 1.583836)

```

Figura 7.8: Tiempo que tarda pradobot en procesar mensajes procedentes de un chat privado y de unchat grupal

Como podemos observar los tiempos de CPU suelen rondar los 0.1 segundos, mientras que el tiempo real empleado es de entre 0.8-2 segundos. Esta diferencia es bastante significativa. Vamos a probar a mandar mensajes con otro programa “telegram-cli”. Este programa permite mandar mensajes a un chat de Telegram desde la terminal:

```
user@workvm:~/Downloads/b/b/tg/bin$ time ./telegram-cli -W -k server.pub -e "msg grupotesttf2 3"
Telegram-cli version 1.4.1, Copyright (C) 2013-2015 Vitaly Valtman
Telegram-cli comes with ABSOLUTELY NO WARRANTY; for details type `show_license'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show_license' for details.
Telegram-cli uses libtgc version 2.1.0
Telegram-cli includes software developed by the OpenSSL Project
for use in the OpenSSL Toolkit. (http://www.openssl.org/)
I: config dir=/home/user/.telegram-cli
[11:20] grupotesttf2 Luis Gil >>> 3
[11:20] grupotesttf2 Luis Gil >>> 3
> All done. Exit
halt

real    0m1.03s
user    0m0.013s
sys     0m0.025s
user@workvm:~/Downloads/b/b/tg/bin$ time ./telegram-cli -W -k server.pub -e "msg grupotesttf2 3"
Telegram-cli version 1.4.1, Copyright (C) 2013-2015 Vitaly Valtman
Telegram-cli comes with ABSOLUTELY NO WARRANTY; for details type `show_license'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show_license' for details.
Telegram-cli uses libtgc version 2.1.0
Telegram-cli includes software developed by the OpenSSL Project
for use in the OpenSSL Toolkit. (http://www.openssl.org/)
I: config dir=/home/user/.telegram-cli
[11:20] grupotesttf2 Luis Gil >>> 3
[11:20] grupotesttf2 Luis Gil >>> 3
> All done. Exit
halt

real    0m1.152s
user    0m0.009s
sys     0m0.017s
user@workvm:~/Downloads/b/b/tg/bin$ time ./telegram-cli -W -k server.pub -e "msg grupotesttf2 3"
Telegram-cli version 1.4.1, Copyright (C) 2013-2015 Vitaly Valtman
Telegram-cli comes with ABSOLUTELY NO WARRANTY; for details type `show_license'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show_license' for details.
Telegram-cli uses libtgc version 2.1.0
Telegram-cli includes software developed by the OpenSSL Project
for use in the OpenSSL Toolkit. (http://www.openssl.org/)
I: config dir=/home/user/.telegram-cli
[11:20] grupotesttf2 Luis Gil >>> 3
[11:20] grupotesttf2 Luis Gil >>> 3
> All done. Exit
halt

real    0m1.007s
user    0m0.007s
sys     0m0.021s
user@workvm:~/Downloads/b/b/tg/bin$
```

Figura 7.9: Envío de mensaje utilizando telegram-cli

Se puede apreciar que la diferencia de tiempo entre el tiempo de CPU total y el tiempo real sigue siendo significativa. El porqué de esta diferencia pienso que es debido a la latencia que supone recibir y enviar un mensaje a los servidores de Telegram, es decir, en el procesamiento del mensaje llega un punto en que se inicia el envío del mensaje de respuesta a Telegram, el sistema operativo, mientras los datos transitan por la red, pausa el programa transfiriéndole el control de la CPU a otro proceso, se completa el envío y nuestro programa finaliza con el procesamiento del mensaje. Lo cual implica que la velocidad con la que el usuario ve una respuesta por parte del bot no es tanto la tardanza de procesar el mensaje sino de la rapidez con la que se envían y reciben los mensajes pudiendo decirse, por tanto, que si quisieramos

que nuestro bot respondiera rápidamente no tendríamos que comprar un procesador más potente sino una conexión a internet más veloz.

7.2. Conclusiones

Tras realizar el proyecto y revisar los objetivos, la impresión que me queda es que Moodle y Telegram pueden llegar a ser muy útiles trabajando juntos. Si bien con algún matiz, ya que Moodle me parece muy rígido en cuanto al modo de configurarlo para poder usar su API.

El uso de Telegram facilita el contacto entre los estudiantes y los profesores de un curso, pudiendo llegar a ser el bot una herramienta que ayude, en gran medida, en todas aquellas tareas que se deriven de esta comunicación. Debido a la falta de tiempo hay muchas funcionalidades interesantes pensadas que no se han podido implementar como por ejemplo:

- Permitir imágenes en las dudas y las respuestas.
- Permitir a un profesor configurar funciones en el bot como:
 - Mensaje bienvenida cuando un estudiante entra al chat del curso.
 - Mostrar avisos cuánto faltan x horas para que finalice la entrega de algún hito del curso.
 - Permitir programar que muestre un mensaje a las x horas del día y.
- El bot notifique al estudiante cuando se ha aprobado/denegado su petición.
- Permitir al bot “grabar” la duda directamente del chat grupal, es decir, con un mecanismo tipo *duda* *¿Cursiva o entrecomiñado para los extranjerismos?* y grabar todas las respuestas que se dieran en el chat hasta pararlo con un *fin_duda*.
- El bot pudiera guardar recursos de interés compartidos por los estudiantes/profesores dentro del chat grupal.

Como última reflexión decir que Moodle no es imprescindible. La mayoría de las funcionalidades descritas más arriba no necesitan necesariamente a Moodle. Moodle aporta **control** e **información**, es el que aporta al bot quién es el responsable de qué curso y, por tanto, el responsable de indicarle al bot cuál es el chat en el que está **autorizado** a compartir la información del curso. Sin una fuente, el bot no sabe discernir dónde puede revelar esta o aquella información. Resulta muy interesante explorar otras posibles fuentes de información, como por ejemplo Google spreadsheets, Github o similares.

Capítulo 8

8.1. Glosario de términos

Bot: Programa informático que efectua tareas repetitivas a través de internet[26][27].

Telegram: Servicio de mensajería instantánea por internet.

Moodle: Plataforma educativa diseñada para la gestión de cursos online.

Interfaz: Capa intermedia entre dos sistemas que utilizan para comunicarse.

Desarrollo iterativo: Metodología de desarrollo de software que implica agrupar el desarrollo del software en un conjunto de etapas repetitivas.[28], [29]

SSH: Protocolo para la comunicación segura entre computadores[25]

SSL: Protocolo para la transmisión de manera segura de información entre dos aplicaciones

Token: Identificador que se le da a un usuario para facilitar el proceso de autenticación.

API: interfaz de programación de aplicaciones, conjunto de funciones que ofrece una biblioteca o aplicación con el fin de que sea usado por otro software como una capa de abstracción.

REST: Transferencia de Estado Representacional, estilo de arquitectura de software para sistemas distribuidos. Actualmente el término se usa para describir cualquier interfaz que permita la modificación de la forma textual de recursos web a través de un conjunto de operaciones predefinidas y sin estado.[30][31]

pradobot: Nombre del programa que estamos desarrollando.

Polimorfismo: Capacidad de enviar el mismo tipo de mensajes a objetos de diferente clase.

ORM: Es una técnica que permite realizar consultas y manipular datos de la base de datos realizando operaciones sobre “objetos” que simbolizan la estructura de los datos.[24]

Issue: Un issue de Github es como una llamada de atención sobre un repositorio de github.

Repositorio: De forma muy simple se puede describir Github como una página web que permite el desarrollo colaborativo de software siendo el repositorio de un proyecto el lugar donde se almacena su código.

Stub: Simulan un objeto con una serie de métodos que devuelven datos predefinidos..[32]

Mock: Parecido a los stubs solo que sus métodos no tiene un comportamiento predefinido. Tienen programados una serie de expectativas como que se llame x veces a este método con tales parámetros y se suelen utilizar para si el uso de la clase que simulan es el esperado en los tests.

Bibliografía

Ruby

- [1] “Ruby docs”. 20/03/2017.ruby-doc.org/

Moodle

- [2] “Get users token”. 05/03/2017.<https://moodle.org/mod/forum/discuss.php?d=193857>
- [3] “How to get a user token from a external aplciation”.05/03/2017. https://docs.moodle.org/31/en/Web_services_FAQ#How_to_get_a_user_token_from_an_external_application.3F
- [4] “Creating a web service client”. 06/03/2017.https://docs.moodle.org/dev/Creating_a_web_service_client#How_to_get_a_user_token
- [5] “Add new user”. https://docs.moodle.org/32/en/Add_a_new_user
- [6] “Step by step installation guide moodle ubuntu”. 07/03/2017.https://docs.moodle.org/26/en/Step-by-step_Installation_Guide_for_Ubuntu
- [7] “Roles and capabilities”. 07/03/2017.https://docs.moodle.org/19/en/Roles_and_capabilities
- [8] “Webservice API functions”. 06/03/2017. https://docs.moodle.org/dev/Web_service_API_functions

- [9] “Webservices”. 06/03/2017. https://docs.moodle.org/dev/Web_services

UML

- [10] “UML basics: The sequence diagram”. 09/04/2017. <https://www.ibm.com/developerworks/rational/library/3101.html>
- [11] “Differences between sequence diagrams and collaboration diagrams”. 09/04/2017. <https://www-01.ibm.com/support/docview.wss?uid=swg21123475>
- [12] “UML 2 Sequence Diagrams: An Agile Introduction”. 09/04/2017. <http://www.agilemodeling.com/artifacts/sequenceDiagram.htm>

Lecturas sobre bots

- [13] “Server-side architecture when bots invade”. 01/03/2017. <https://medium.com/@JonathanZWhite/server-side-infrastructure-when-bots-invade-a2252e9d4bc9>
- [14] “Server-side architecture when bots invade”. 01/03/2017. <https://medium.com/@surmenok/chatbot-architecture-496f5bf820ed>

Sequel

- [15] “Dataset filtering”. 01/05/2017 . http://sequel.jeremyevans.net/rdoc/files/doc/dataset_filtering_rdoc.html
- [16] “ConnectionPool”. 01/05/2017 . <http://www.rubydoc.info/github/evanfarrar/opensprints/Sequel/ConnectionPool>
- [17] “ThreadedConnectionPool”. 01/05/2017 . http://sequel.jeremyevans.net/rdoc/classes/Sequel/ThreadedConnectionPool.html#attribute-i-available_connections

Tests

- [18] “ RSpec Expectations 3.6”. 10/05/2017 . <https://relishapp.com/rspec/rspec-expectations/docs/built-in-matchers>
- [19] “ RSpec Expectations 3.6”. 10/05/2017 . <https://dzone.com/articles/why-shouldnt-i-test-privates>

Telegram

- [20] “Telegram”. 01/03/2017 . <https://core.telegram.org/bots>
- [21] “Telegram bot API”. 01/03/2017 . <https://core.telegram.org/bots/api#message>
- [22] “Telegram bot updates”. 01/03/2017 . <https://core.telegram.org/bots/api#getupdates>
- [23] “Long polling vs webhooks”. 01/03/2017 . <https://github.com/python-telegram-bot/python-telegram-bot/wiki/Webhooks#polling-vs-webhook>

Definiciones

- [24] “Visual paradigm ORM”. 01/09/2017 . https://www.visual-paradigm.com/support/documents/vpuserguide/3563/3581/85424_whatisObject.html
- [25] “SSH”. 01/09/2017. <http://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/ch-ssh.html>
- [26] “Bot”. 01/09/2017 . <https://es.wikipedia.org/wiki/Bot>
- [27] “Malicious bots”. 01/09/2017. <https://books.google.com/books?id=nmgK7KcibSUC>
- [28] “Desarrollo iterativo y creciente”. 01/09/2017. https://es.wikipedia.org/wiki/Desarrollo_iterativo_y_creciente
- [29] “Proceso de Desarrollo Iterativo”. 01/09/2017 . <http://fernandosoriano.com.ar/?p=13>
- [30] “REST”. 01/09/2017 . <https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest>
- [31] “Representational state transfer”. 01/09/2017 . https://en.wikipedia.org/wiki/Representational_state_transfer
- [32] “Pruebas unitarias”. 01/09/2017 . <https://itblogsogeti.com/2015/03/26/desarrollo-pruebas-unitarias-trinitario-gomez-sogeti/>

Otro material

- Consultas Stack OverFlow.
- Alguna cosilla latex <https://github.com/germaaan/TFG>.
- Material de las asignaturas **Fundamentos de Ingeniería del Software**, **Programación Orientada a Objetos**, **Diseño de Interfaces de Usuario** , **Diseño de Interfaces de Usuario** , **Diseño y Desarrollo de Sistemas de Información**, **Diseño y Desarrollo de Sistemas de Información**, **Seguridad en Sistemas Operativos**, **Derecho Informático**, (**Ingeniería de Servidores e Infraestructura Virtual** impartidas en **Grado en Ingeniería Informática** en la **Universidad de Granada**.

