UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DESARROLLO DE UN NUEVO SERVICIO DE NOTIFICACIONES PARA LA PLATAFORMA DE APOYO A LA GESTIÓN Y APRENDIZAJE U-CURSOS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERA CIVIL EN COMPUTACIÓN

VALERIA CONSTANZA LEONOR LEÓN GONZÁLEZ

PROFESORA GUÍA:
JOCELYN SIMMONDS WAGEMANN

PROFESOR CO-GUÍA:
WILLY MAIKOWSKI CORREA

MIEMBROS DE LA COMISIÓN:
JOSÉ PINO URTUBIA
DANIEL PEROVICH GEROSA

SANTIAGO DE CHILE
2023

Resumen

En la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile, el Centro Tecnológico Ucampus dispone de dos plataformas de apoyo a la docencia y gestión curricular a la comunidad, U-Cursos y Ucampus respectivamente. U-Cursos es el sitio más utilizado por los estudiantes y profesores a lo largo del año académico, y desde el 2020 ha presentado un aumento considerable de los usuarios concurrentes, lo que ha generado problemas y caídas dentro de las plataformas. Una de las causas de esas caídas es que el sistema de notificaciones se encuentra acoplado al controlador principal de la plataforma, conocido como `KERNEL`, por lo que un problema en las notificaciones conlleva a la caída completa del sistema.

Debido a lo anterior, se propuso separar dichos funcionamientos. Para ello, se investigaron herramientas disponibles en el mercado que pudieran reemplazar a las notificaciones de U-Cursos, también se analizaron librerías alternativas que pudieran proveer dicha funcionalidad pero con mayor control de desarrollo y, por último, se averiguó sobre opciones para darle mayor funcionalidad al sistema actual. De acuerdo al funcionamiento actual de la plataforma, a que las alternativas implicaban un costo monetario extra, a la falta de implementación de estructuras que se acoplaran a la plataforma en las librerías encontradas y que el sistema actual poseía una librería abstraída llamada `pubsub`, se decidió no utilizar ninguna herramienta externa, sino más bien brindarle asincronía y robustez a la implementación actual.

Para ello, se diseñó una estrategia diferenciada para los dos tipos de consultas que recibe el sistema actualmente. Para el caso de lectura de datos, estas debían efectuarse desde el lado del cliente con *JQuery*, y para el caso de escritura de datos, se utilizó encolamiento de solicitudes a partir de una librería ya existente en Ucampus. Además, se desarrollaron múltiples *endpoints* en un módulo de U-Cursos con un funcionamiento separado del `KERNEL`. Por otra parte, se eliminaron las más de 100 líneas de código donde se utilizaba el sistema de notificaciones en la ejecución normal del sitio.

Con todo lo diseñado e implementado, se logró cumplir con todos los objetivos a excepción de la validación del sistema en las mismas condiciones en que el año 2020 se presentaban caídas. Aun así, se logró implementar una ejecución no bloqueante del `KERNEL` y comprobar que aunque la base de datos de Notificaciones estuviese caída, la conexión asíncrona y el encolamiento de las tareas permiten la disponibilidad del resto de la plataforma. En conclusión, se considera que si bien se poseen mejoras a futuro, se adquiere un excelente punto de partida para el nuevo servicio, con posibilidad de extensión por parte del Centro Ucampus, permitiendo separar y mejorar el sistema de notificaciones de U-Cursos, y en consecuencia, logrando una plataforma más estable.

A mi yo interior: lo logramos.

Agradecimientos

Primeramente, me gustaría agradecer a mis padres por apoyarme todos estos años en mis decisiones, a pesar de la distancia y de tener que dejarme ir para cumplir mis sueños, gracias por siempre darme una palabra de aliento y entregarme todo su amor incluso en los tiempos más difíciles. A mis hermanas, que han sido y siguen siendo mis modelos a seguir y que me tomaron de la mano a su manera para guiarme donde estoy.

A mi mamá, Natalia, por siempre estar presente en mi corazón. Donde quieras que estés, gracias por cuidarme y permitir que tu recuerdo siga aquí.

Al resto de mi familia, tíos y primos, por escuchar y crecer juntos, creando una red de apoyo para toda la vida.

A todos mis amigos de esta linda etapa universitaria, por todas las juntas de risas y estudios, por su ayuda y cariño, porque hicieron de esta etapa una etapa mucho mejor de lo jamás hubiese esperado. Y a aquellos que han estado conmigo desde el principio y siguen estando. A Canela y Bianka, quienes me han ayudado en las etapas difíciles y en las peores, espero algún día poder retribuir todo su cariño.

A Bastián, por aceptarme en todas mis facetas y darme la contención y el cariño que necesitaba en algunos de los momentos más duros de mi carrera y de mi vida. Gracias por ayudarme a avanzar y por darme ánimos y confianza durante toda esta memoria. Definitivamente no hubiese llegado hasta aquí sin ti. Y gracias a su familia, por acogerme como una más de los suyos.

A mis profesores guías, Jocelyn y Willy, por confiar en mí y apoyarme en cada paso de este camino, recordándome que yo también debo confiar en mí misma. Gracias por su comprensión y dedicación.

Al Centro Tecnológico Ucampus y su equipo, por brindarme su confianza para un proyecto tan especial como lo fue esta memoria. Y a Cecilia y Matías, por hacer más entretenidas las tardes en la oficina.

A mis compañeros de vida peludos, mis gatos y perros, por contenerme en su mar de pelos y ronroneos cada vez que volvía a casa. Esto también es por ustedes.

Y finalmente, a mí misma, por llegar hasta aquí, por seguir avanzando cuando todo se veía negro. Gracias por ser tan valiente y no rendirte. Lo logramos.

Tabla de Contenido

1. Introducción **1**

1.1. Problemática y Contexto 1

1.2. Objetivos 3

1.2.1. Objetivo General 3

1.2.2. Objetivos específicos 3

1.3. Solución Propuesta 3

1.4. Metodología 3

1.5. Estructura del documento 5

2. Estado del Arte **6**

2.1. Sistemas de notificaciones en el mercado 6

2.1.1. Firebase 6

2.1.2. *Amazon SNS* 7

2.1.3. Ably 8

2.2. Exploración de librerías alternativas 9

2.2.1. PubSubJS 9

2.2.2. Redis Pub/Sub 10

2.2.3. Laravel PubSub 11

2.3. Llamadas asíncronas en PHP 11

2.3.1. Spatie/Async 11

2.3.2. Revolt 12

3. Análisis y Diseño **13**

3.1. Estructura de U-Cursos 13

3.1.1. Arquitectura 13

3.1.2. Estándar Ucampus para API's 14

3.2. Sistema de notificaciones actual 15

3.2.1. Funcionamiento del sistema 15

3.2.2. Limitaciones de este sistema 17

3.2.3. Arquitectura del sistema 18

3.3. Comparación de alternativas 19

3.3.1. Sistemas de notificaciones en el mercado 19

3.3.2. Librerías Pubsub alternativas 23

3.3.3. Llamadas asíncronas en PHP 23

3.4. Diseño de nuevo módulo de notificaciones 24

3.4.1. Creación módulo de notificaciones 25

4. Implementación **31**

4.1. Separación notificaciones de U-Cursos 31

4.2. Implementación de endpoints de la API 32

4.2.1. Endpoints de lectura 32

4.2.2. Endpoints de escritura 34

4.3. Encolamiento de solicitudes 36

4.3.1. Base de Datos 36

4.3.2. Clase Tasks2 36

4.3.3. Archivos Batch 37

4.4. Llamadas asíncronas con JQuery 38

4.4.1. Módulo Canales 38

4.4.2. Módulo Notificaciones 41

4.5. Conexión a los endpoints 43

4.5.1. Endpoints de lectura 43
4.5.2. Endpoints de escritura 44

5. Validaciones 45

5.1. Revisión de Código 45
5.2. Pruebas de funcionalidad 46
5.3. Pruebas de carga y robustez 49

6. Conclusión 51

6.1. Trabajo realizado 51
6.2. Cumplimiento de objetivos 52
6.3. Mejoras a futuro 54
6.4. Aprendizaje personal 55

Bibliografía 57

Anexos 58

A. Documentación 59

A.1. Usos de la librería PubSub de Ucampus en U-Cursos 59
A.1.1. Módulo Kernel 59
A.1.2. Módulo Canales 60
A.1.3. Módulo Oauth 62
A.1.4. Módulo Servicios 62
A.1.5. Módulo Tareas 62
A.1.6. Módulo Uchile 63
A.1.7. Módulo Actas 63
A.1.8. Módulo Calendario 63
A.1.9. Módulo Datos Institución 63
A.1.10. Módulo Correo 63

A.1.11. Módulo Dropbox 64
A.1.12. Módulo Encuestas 64
A.1.13. Módulo Foro 64
A.1.14. Módulo Horario 64
A.2. Ejemplo de uso de Revolt 64

Índice de Tablas

3.1. Cuadro resumen de tablas de la base de datos y su contenido 18

3.2. Flujo lógico de funciones y tablas en la base de datos a las que acceden . . . 19

3.3. Comparación de alternativas 20

Índice de Ilustraciones

2.1. Diagrama del funcionamiento de Firebase CM. Fuente: Elaboración propia. . 7

2.2. Diagrama del funcionamiento de Amazon SNS. Fuente: Elaboración propia. . 8

2.3. Diagrama del funcionamiento de Ably. Fuente: Elaboración propia. 9

3.1. Diagrama de la estructura modular de U-Cursos. Fuente: Elaboración propia. 14

3.2. Módulos para el curso CC6909-5 Trabajo de Título, U-Cursos 14

3.3. Configuración de notificaciones para cada canal disponible de un usuario . . 16

3.4. Flujo de envío de mensaje y guardado de notificaciones. Fuente: Elaboración propia. 17

3.5. Diagrama alto nivel del módulo Notificaciones en U-Cursos. Fuente: Elaboración propia. 24

3.6. Diagrama alto nivel de los principales requisitos del sistema. Fuente: Elaboración propia. 25

3.7. Diagrama de los componentes del módulo Notificaciones. Fuente: Elaboración propia. 26

3.8. Diagrama del flujo del encolamiento de tareas 29

4.1. Estructura tabla `TASKS2` 37

4.2. Interfaz del módulo Notificaciones 41

5.1. Encolamiento de tareas al publicar un mensaje en Foro Institucional 47

5.2. Encolamiento de tareas al publicar un mensaje en Foro Institucional 48

5.3. Interfaz módulo Notificaciones con base de datos no disponible 50

5.4. Endpoint `pendientes` al ser consultado desde el navegador con base de datos no disponible 50

Capítulo 1

Introducción

En este capítulo se entregará el contexto, la problemática y los objetivos a resolver a lo largo de este trabajo de memoria.

En la sección 1.1 se detallará el problema y contexto actual, en la sección 1.2 se presentarán el objetivo general y los objetivos específicos de este proyecto. En la sección 1.3 se dará una idea de la solución propuesta al problema, finalizando con la metodología de trabajo y la estructura del documento en las secciones 1.4 y 1.5 respectivamente.

1.1. Problemática y Contexto

El Centro Tecnológico Ucampus[1] es un centro de tecnologías de información nacido dentro de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile, que desarrolla plataformas de apoyo a la gestión docente para automatizar procesos de Educación Superior. Distintas universidades comparten el mecanismo de gestión de Ucampus, como la Universidad de Aysén o la Universidad de O'Higgins.

Existen distintas áreas dentro del centro que permiten su funcionamiento. Específicamente, para el desarrollo de las herramientas del LMS (*Learning Management System*: sistema de gestión de aprendizaje, en español, es un software utilizado para administrar actividades no presenciales de una institución u organización.) se tiene el equipo de Desarrolladores, encargado además de otras plataformas como lo son la Plataforma Digital de Participación Popular de la Convención Constitucional[2] o la plataforma del programa Súbete de Movilidad Nacional[3] de las universidades estatales.

La plataforma de U-Cursos es utilizada principalmente por la Universidad de Chile como plataforma de apoyo a la docencia, además del uso de Ucampus para gestión. U-Cursos cuenta con sus versiones de aplicación web y móvil, que permiten a los usuarios mayor flexibilidad para estar al tanto de las novedades de sus cursos inscritos, comunidades e instituciones.

[1]Sitio web: `https://ucampus.cl`, consultado el 25 de agosto de 2022
[2]Sitio web: `https://www.chileconvencion.cl`, consultado el 11 de septiembre de 2022
[3]Sitio web: `https://www.movilidadnacional.cl`, consultado el 11 de septiembre de 2022

Entre las funciones más destacadas y utilizadas de las plataformas se encuentran los módulos para entregar tareas, compartir material docente, revisar notas, calendario, foros de discusión, enlaces, entre muchos otros. Todos estos módulos permiten entregar notificaciones a los usuarios cada vez que hay cambios en el contenido de un módulo.

Desde el 2020, y debido a la pandemia, se implementaron nuevos módulos para apoyar el paso a la docencia remota, como el módulo de Clase Virtual. Esto, sumado al aumento en la matrícula que ha presentado la universidad los últimos años, produjo un alza de aproximadamente un 300 % en los usuarios recurrentes de la plataforma. Por otra parte, los bloques horarios en donde se tienen una mayor cantidad de clases dictadas, corresponden a los días martes y jueves a las 10:15 hrs. Estos factores, en conjunto a la implementación actual del sistema de notificaciones, provocaban caídas de la plataforma en dichos bloques debido a la masiva notificación de usuarios con el mensaje *"Ha comenzado una nueva clase virtual"*.

La implementación de las notificaciones actualmente utiliza parte del controlador principal[4] de U-Cursos para encargarse de agregar entradas y hacer consultas a la base de datos, la cual posee un tamaño considerable que provoca que las consultas puedan demorar mucho e incluso caerse. Sumando lo anterior, algo tan simple como las notificaciones de usuario pueden hacer que el controlador de la plataforma deje de responder y por lo tanto, que se caiga U-Cursos.

Este no es un problema menor, ya que una notificación no deja que los estudiantes entreguen tareas o deja de funcionar U-Test[5] en medio de una evaluación ya que se cayó U-Cursos. En otras palabras, al ser U-Cursos la principal herramienta utilizada para el apoyo a la gestión y aprendizaje, lugar en donde se concentran evaluaciones e informaciones relevantes para las comunidades, una caída de la plataforma genera retrasos en entrega de información, retrasos en evaluaciones o coordinaciones al interior de cursos y comunidades.

Es por ello que se requiere una mejor solución para un sistema de notificaciones que permita manejar de mejor manera las consultas a la base de datos, sin poner en riesgo la integridad y disponibilidad del servicio, permitiendo a los usuarios una mejor experiencia en períodos de alta demanda; ya que, si bien se han generado soluciones parche a este problema, no son suficientes considerando que la demanda de uso está en constante crecimiento. Se busca que dicho sistema se independice del controlador de U-Cursos, como un servicio externo a la plataforma que funcione mediante consultas dinámicas, similar a Ajax por ejemplo, para que de esta forma si la consulta se cae, el controlador decida qué hacer en lugar de dejar de funcionar.

[4]Controlador: dentro de la estructura de un software, es la rutina o programa que se encarga de manejar subrutinas o eventos y comunica distintas capas, principalmente datos e interfaces, entre otras funciones.

[5]Módulo de U-Cursos que permite al cuerpo docente tomar evaluaciones de manera síncrona remota a todos los integrantes de un curso.

1.2. Objetivos

1.2.1. Objetivo General

El objetivo de este trabajo de memoria es diseñar e implementar un sistema de notificaciones para las plataformas de Ucampus, que esté separado de la ejecución principal de las mismas, de manera que permita integridad y disponibilidad en tiempos de alta demanda. Por lo tanto, se definiría como un trabajo exitoso si se logra que los sistemas no se vean afectados por un bajo rendimiento del servicio de notificaciones.

1.2.2. Objetivos específicos

1. Identificar el lugar de la estructura donde se produce la unión del sistema de notificaciones con el resto de la plataforma para mitigar problemas que puedan producirse al separar los servicios.
2. Identificar las tecnologías que se ajustan de mejor manera al desarrollo de un nuevo sistema de notificaciones.
3. Rediseñar la funcionalidad de notificaciones de manera independiente al resto de los servicios.
4. Diseñar un proceso de migración entre las plataformas actuales y el nuevo servicio.
5. Implementar lo diseñado al momento.
6. Validar el rendimiento de las plataformas en sus condiciones de uso actuales, es decir, con un flujo aproximado de 7,000 usuarios en los bloques martes y jueves a las 10:15hrs.

1.3. Solución Propuesta

La solución propuesta para esta problemática es una primera versión de un microservicio de notificaciones para U-Cursos, manteniendo las funcionalidades ya existentes de la librería actualmente utilizada para ello.

Este microservicio debe ser externo a las actuales plataformas de Ucampus, principalmente externo a U-Cursos, y debe permitir generar las conexiones con dichos sistemas y el servicio en términos de datos de forma independiente, como una forma de asegurar continuidad del resto de los servicios en caso de caída en el sistema de notificaciones.

1.4. Metodología

Primeramente, para poder estructurar y acotar el alcance del proyecto, se realizó una investigación de las tecnologías ya presentes en el mercado que abordan y ofrecen servicios

de notificaciones, comparando la conveniencia de cada alternativa. La principal preocupación en herramientas de terceros, como *Firebase*, *Ably* o *Amazon SNS* viene dada por los cobros adicionales por volúmenes de datos. Se elige entonces realizar un primer prototipo del sistema de notificaciones propio de Ucampus.

Luego se analizaron las opciones disponibles de librerías que podría utilizar el sistema y cómo se mantendrían las funcionalidades ya existentes que dependen de las notificaciones dentro de *U-Cursos*, ya que uno de los objetivos específicos es precisamente que la plataforma mantenga todas sus funcionalidades actuales. Estudiando estas alternativas y además la estructura interna de la librería propia que ya se estaba utilizando, se decidió continuar con la librería del sistema actual y enfocar el trabajo en separar las estructuras involucradas que pasarían a formar parte del nuevo microservicio de notificaciones.

Siguiendo esta línea, se realiza entonces el estudio del resto de la estructura y el flujo de las notificaciones dentro de la plataforma, aislando los usos de la librería encargada de las notificaciones y obteniendo una instancia de la plataforma que mantiene su funcionamiento sin uso de notificaciones. Paralelamente, se estudiaron *frameworks* para creación de API's en lenguaje *PHP*, entre ellos Laravel y el método propio de Ucampus, decidiendo que este último era preferible para el desarrollo del microservicio.

Posteriormente, se estableció que el nuevo servicio de notificaciones correspondería a un módulo de U-Cursos, por lo que se comenzó con el diseño lógico de los componentes que harían parte del módulo y permitirían la conexión con el mismo. Se establecen los *endpoints* a implementar y el tipo de manejo que se le dará a las solicitudes que llegan a cada uno de ellos, para posteriormente implementar dichos componentes y funcionalidades.

Lo anterior requirió movilizar librerías desde el módulo `KERNEL` hacia el módulo Notificaciones, para poder dar soporte al manejo de datos y conexión con la base de datos `PUBSUB` al nuevo módulo. Además, se debió crear una nueva tabla `TASKS2` en la base de datos, para poder guardar las solicitudes de ingreso de información que luego deben ser ejecutadas, esto como parte de la funcionalidad de encolamiento de solicitudes.

Una vez se tuvieron implementados los *enddpoints* y el encolamiento de tareas, se procede a conectar algunos módulos que hacen uso de notificaciones al nuevo módulo, como Foro y Canales, para validar que el funcionamiento de estos se mantuviera dada la nueva conexión. Estas conexiones se implementaron de forma asíncrona mediante consultas web con *JQuery* desde el *frontend* cuando los módulos solo consultan información a la base de datos. Aquellas consultas que se realizan para ingresar nueva información, pasan por el encolamiento de tareas para ser procesadas de forma asíncrona.

Finalmente, se realizaron distintas validaciones al módulo implementado. En primer lugar, se realizó una validación junto al profesor guía Willy Maikowski que permitió estandarizar el código escrito, documentarlo y analizar los casos de uso que permitía el módulo hasta ese momento. Una vez completada la implementación y conexión parcial, se realizaron validaciones de funcionalidad y robustez del módulo, comprobando el flujo que realiza el módulo al recibir los distintos tipos de consultas en condiciones normales y cuando la base de datos no se encuentra disponible.

1.5. Estructura del documento

Este documento se compone de diferentes capítulos y secciones que permiten guiar el entendimiento del proyecto realizado. En el Capítulo 2, se describen algunos sistemas de notificaciones más conocidos disponibles en mercado, como también librerías ocupadas para implementar esta funcionalidad bajo el paradigma Publicar/Suscribir en sistemas, terminando con una discusión acerca de las herramientas disponibles para realizar llamadas asíncronas en *PHP*.

En el Capítulo 3, primero se describe la arquitectura de U-Cursos y cómo funciona el sistema de notificaciones al momento de iniciar este trabajo de memoria. Después se presenta una comparación de los sistemas de notificaciones y herramientas descritos en el Capítulo 2, y en base a esto, se presenta el diseño del sistema de notificaciones a desarrollar en este proyecto.

En el Capítulo 4 se explican los principales aspectos de la implementación de este sistema de notificaciones: la refactorización de U-Cursos para poder remover el sistema existente de notificaciones, la implementación de los nuevos *endpoints* del nuevo sistema y el desarrollo de varios mecanismos que permiten mayor robustez de la solución frente a los escenarios típicos de uso del sistema.

Finalmente, en el Capítulo 5 se presentan las distintas validaciones del sistema y el Capítulo 6 cierra con una discusión de las conclusiones, el trabajo a futuro y los aprendizajes personales a partir de este trabajo de memoria.

Capítulo 2

Estado del Arte

En este capítulo se presentan diferentes opciones y alternativas a herramientas que actualmente utiliza el sistema de notificaciones de U-Cursos.

En la sección 2.1 se presentan opciones de sistemas de notificaciones en el mercado, en la sección 2.2 se muestran alternativas de librerías Pub/Sub relevantes para este proyecto y en la sección 2.3 se presentan alternativas para realizar llamadas asíncronas en *PHP*.

2.1. Sistemas de notificaciones en el mercado

A continuación, se presentan algunos sistemas de notificaciones disponibles en el mercado que siguen el paradigma Publicar/Suscribir.

2.1.1. Firebase

Firebase Cloud Messaging (FCM) es un servicio enfocado en el envío de notificaciones *push*, aquellas notificaciones emergentes en dispositivos móviles o de escritorio informando de un cambio en algún lugar de la aplicación. Posee un sistema para el guardado de mensajes y contenido de las notificaciones en formato NoSQL [20], mediante archivos que contienen la información por un período máximo de 4 semanas.

El funcionamiento de *Firebase CM* es ligeramente diferente al que se tiene actualmente en U-Cursos para el manejo de publicaciones en canales y suscripciones, ya que al ser su enfoque el envío de la notificación, prioriza la masividad y la entrega, más que la segmentación de los usuarios y suscripciones a diferentes canales. Posee configuraciones que permiten tener diferentes temas o tópicos a los que los usuarios se suscriben, pero para cada notificación diferente se debe configurar el contenido del mensaje, ya que se define como una nueva campaña. Todas estas configuraciones pueden ser manejadas desde el *dashboard* de la plataforma o directamente en el código fuente del *front* de la aplicación web o móvil.

Una vez realizado el diseño de la notificación a entregar, esta se envía hasta a 500 usuarios a la vez, realizando más llamadas de envío en caso de superar ese límite de entrega de notificaciones.

En el caso de que la información relativa a notificaciones se quisiera almacenar con algún servicio como *Firebase Realtime Database* (FRD), que permite la actualización de los datos guardados de manera automática en todos los dispositivos conectados a la aplicación, se debe considerar que existe costo asociado al tamaño de almacenamiento utilizado.

En caso de no optar por *Firebase Realtime Database* y mantener la base de datos actual, *Firebase CM* no soluciona de por sí el problema de las consultas a dicha base que bloquean el controlador, por lo que con cualquiera de estas opciones se debe diseñar e implementar una forma de manejar los llamados para evitar la caída de la plataforma.

El flujo de este proceso se puede observar en la figura 2.1, basada en el diagrama disponible en el sitio oficial de *Firebase CM*[9].

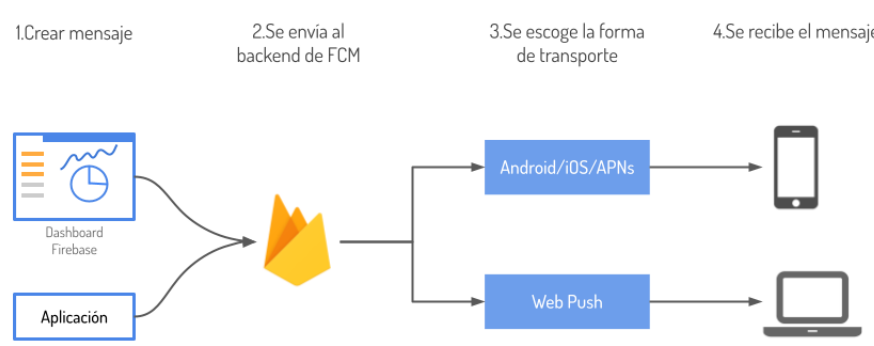


Figura 2.1: Diagrama del funcionamiento de Firebase CM. Fuente: Elaboración propia.

2.1.2. *Amazon SNS*

El caso de *Amazon SNS* es similar al de *Firebase Cloud Messaging*, pero difiere en algunos detalles. Como *Firebase CM*, *Amazon SNS* es un sistema que también se enfoca en el envío de las notificaciones *push* para aplicaciones web y móviles, y pueden agruparse las notificaciones en *tópicos* para segmentar dispositivos y suscriptores, con una API de tipo Pub/Sub (publicar-suscribir).

Amazon SNS posee un sistema para reprocesar las notificaciones que no fueron entregadas debido a la desconexión del dispositivo y así reenviarlas o llevarlas a análisis en otras herramientas. En caso de querer almacenar la información de las notificaciones, también se debe implementar la lógica de guardado de dicha información en una base de datos propia

o con algún servicio de *Amazon*, como *Amazon Simple Storage System* (S3), ya que *SNS* no guarda la información.

Una diferencia de *Amazon SNS* con respecto a *Firebase CM*, es que posee una cuota gratuita de 1M de solicitudes al mes y luego se paga en relación a la cantidad extra utilizada de acuerdo a los criterios fijados por *Amazon*, que pueden ser ambiguos en cuánto al cálculo de dicho uso extra. *Amazon* S3 funciona con el mismo sistema.

En la figura 2.2 se puede observar el flujo de envío de notificaciones en alto nivel que ofrece *Amazon SNS*, basado en el diagrama disponible en la página web oficial del servicio[5].

Figura 2.2: Diagrama del funcionamiento de Amazon SNS. Fuente: Elaboración propia.

2.1.3. Ably

Ably [3] es un sistema de notificaciones que funciona como PaaS (*Platform-as-a-Service*) similar a los vistos anteriormente, pero que tiene diferentes planes de precios establecidos para los distintos estándares de uso. También funciona con el mecanismo Pub/Sub, tal como *Amazon SNS* y se enfoca en la entrega de mensajes en tiempo real brindando soporte y rendimiento para sistemas con gran cantidad de usuarios.

Ably plantea la definición, tanto para canales como usuarios, del concepto *peak*, que se traduce como canales utilizados simultáneamente o usuarios que generan una conexión simultánea a la API de *Ably*. 200 usuarios peak son 200 conexiones simultáneas a *Ably* en un canal. 200 canales peak son 200 canales utilizados a la vez.

Las tres modalidades de servicio son el plan gratuito, plan prepago y plan personalizado, cuyos detalles en relación a precios se discutirán más adelante en la Sección 3.3: Comparación de alternativas. Todos ellos incluyen cuotas de solicitudes y almacenamiento de datos.

Para el caso particular de las características gratuitas, *Ably* guarda los mensajes solo por

24hrs, por lo que al igual que para *Firebase CM* y *Amazon SNS*, se debe trabajar en la lógica de guardado de notificaciones si se quiere mantener información en la base de datos actual.

En la figura 2.3 se puede observar el flujo de envío de notificaciones a un alto nivel de *Ably*, basado en el diagrama disponible en el sitio web oficial de *Ably*[3]. En esta figura las flechas azules representan una publicación de notificaciones y las grises representan el recibo de una notificación.

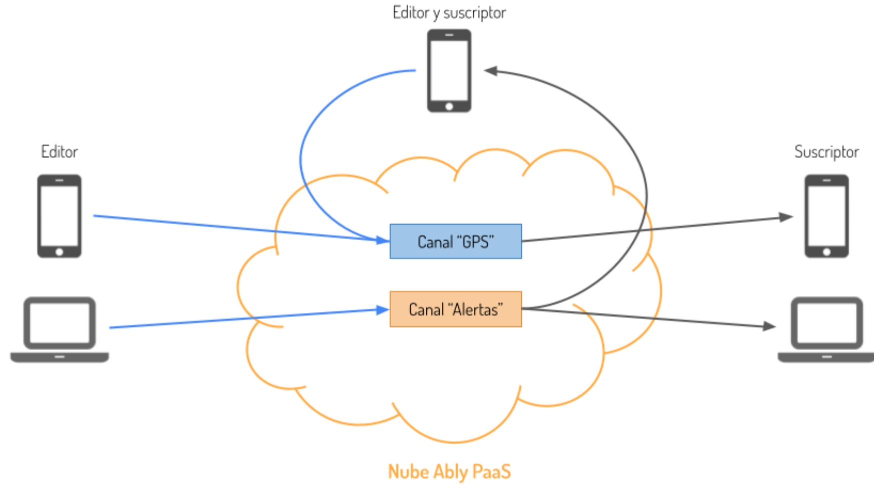


Figura 2.3: Diagrama del funcionamiento de Ably. Fuente: Elaboración propia.

2.2. Exploración de librerías alternativas

Además de buscar sistemas ya disponibles en el mercado que pudiesen suplir el modelo actual de notificaciones, se barajaron opciones que pudiesen reemplazar las funcionalidades de la librería `pubsub` que utiliza el `KERNEL`para el manejo de información de la base de datos, ya que la actual librería es extensa y comprende variados métodos de lectura y escritura.

A continuación se presentan algunas opciones consideradas.

2.2.1. PubSubJS

PubSubJS[17] es una librería *Javascript* bajo el modelo Publicar/Suscribir, cuyos métodos están orientados de manera similar a *Amazon SNS*, esto es, con enfoque en usuarios suscritos a diferentes tópicos y métodos simples para la extracción de la información, además de comunicación síncrona. La librería ha sido usada por múltiples usuarios dentro de *GitHub*, con 464 *forks* y 4,6k estrellas de puntuación.

En el código 2.1 se pueden observar algunos métodos de esta librería, que corresponden a aquellos que permiten publicar y suscribir usuarios.

```
1 import PubSub from 'pubsub-js'
2 var mySubscriber = function (msg, data) {
3     console.log( msg, data );
4 };
5
6 var token = PubSub.subscribe('MY TOPIC', mySubscriber);
7
8 PubSub.publish('MY TOPIC', 'hello world!');
```

Listing 2.1: Usos básicos de la librería PubSubJS, fuente: https://github.com/mroderick/PubSubJS

La librería utiliza objetos como estructura para almacenar información previo a su guardado en la base de datos, de la que se crea una instancia en la línea 1, representando un suscriptor. El método suscribir recibe como parámetros el *string* con el tópico y el objeto suscriptor, retornando un *token* de identificación, lo que se puede observar en la línea 6; este *token* permite cancelar la suscripción del tópico a futuro. El método `publish` permite publicar un mensaje en un tópico, los que deben ser entregados como parámetros, tal como se ilustra en la línea 8.

El proceso para que un mensaje de notificación llegue a un usuario es más extenso e involucra varios métodos que deben ser utilizados para que la notificación efectivamente sea entregada.

2.2.2. Redis Pub/Sub

Redis Pub/Sub[15] es una librería de *Redis* que permite utilizar el paradigma Pub/Sub dentro de una aplicación, orientado al almacenamiento de datos en memoria. La librería viene incluida al instalar *Redis* y posee el enfoque de suscriptores a diferentes canales.

Para poder utilizar los métodos de la librería se necesitan ingresar las instrucciones por línea de comando. Para ser utilizado en una aplicación, se deben definir funciones que permitan formatear la data manejada a nivel lógico a lo pedido y especificado por el comando que debe ser llamado.

En el código 2.2 se puede observar la definición de métodos que permiten utilizar los comandos para suscribir un usuario a un canal y para publicar mensajes en dichos canales.

```
1   def subscribe(*channels, &blk, user)
2     channels.each { |c| @blocks[c.to_s] = blk }
3     call_command('subscribe', *channels, user)
4   end
5
6   def publish(channel, msg)
7     call_command('publish', channel, msg)
8   end
```

Listing 2.2: Ejemplo en alto nivel de uso de la librería Redis Pub/Sub, fuente: https://gist.github.com/pietern/348262

2.2.3. Laravel PubSub

Laravel PubSub[18] es una librería para ser utilizada en el *framework Laravel*, y que se conecta bien con otras tecnologías como *Redis*. Su orientación es a suscriptores en canales, similar a la librería anterior. En particular, esta librería no ha sido utilizada por muchos usuarios en *GitHub*, con solo 39 *forks* y 67 estrellas de puntuación.

En el código 2.3 se tiene un ejemplo de uso de los principales métodos de la librería. En la línea 1 se define el objeto de tipo *pubsub* que contiene los métodos a utilizar, como lo son `publish` y `subscribe`. Para el caso de este último, no se tiene claro el uso de la función entregada como parámetro al método, por lo que se requiere más documentación y estudio al respecto para adaptar su uso en caso de ser necesario.

```
1 $pubsub = app('pubsub');
2
3 $pubsub->publish('channel_name', 'message');
4 $pubsub->subscribe('channel_name', function ($message) {
5     var_dump($message);
6 });
```

Listing 2.3: Ejemplo en alto nivel de uso de la librería Laravel PubSub, fuente: `https://github.com/Superbalist/laravel-pubsub/`

Por su parte, la librería local `pubsub` del `KERNEL` sigue siendo una opción a considerar, ya que actualmente es utilizada por la plataforma, como se explicará en mayor detalle en la sección 3.2.

2.3. Llamadas asíncronas en PHP

Otro elemento de búsqueda para este proyecto fueron las opciones para realizar llamadas web asíncronas dentro de los lenguajes que utiliza U-Cursos, específicamente *PHP*, que corresponde a un lenguaje compilado, por lo que el servidor compila el resultado de una consulta antes de enviarlo al cliente. Esto hace en principio difícil realizar llamadas asíncronas que requieran una respuesta posterior al compilado.

A continuación se presentan algunos de los recursos encontrados.

2.3.1. Spatie/Async

Async[1] forma parte del paquete *Spatie* para *PHP*, que permite realizar llamadas asíncronas mediante la ejecución de diferentes procesos en paralelo.

En el código 2.4 se tiene un ejemplo de uso de este paquete. Para poder utilizarlo, se necesita definir un elemento `Pool`, lo que sucede en la línea 2, que es el objeto encargado de manejar los *threads* o procesos necesarios para la ejecución de funciones, en este caso, asíncronas. Entre las líneas 4 y 11 se tiene un ciclo *for* que para cada conjunto de datos que se le entrega realiza alguna acción, que al ser completada se maneja según su ejecución haya

sido exitosa o haya fallado. Al terminar el ciclo, el objeto `pool` entra en espera hasta volver a ser utilizado.

```
1 use Spatie\Async\Pool;
2 $pool = Pool::create();
3
4 foreach ($things as $thing) {
5     $pool->add(function () use ($thing) {
6         // Do a thing
7     })->then(function ($output) {
8         // Handle success
9     })->catch(function (Throwable $exception) {
10        // Handle exception
11    });
12 }
13
14 $pool->wait();
```

Listing 2.4: Ejemplo en alto nivel de uso del paquete Spatie/Async, fuente: `https://github.com/spatie/async`

2.3.2. Revolt

Revolt[2] es un paquete para *PHP* que permite añadir un *scheduler* o controlador, también llamado *event loop* para aplicaciones concurrentes, esto es, que necesiten introducir *threads* en su ejecución. En su caso particular, *Revolt* permite realizar llamadas asíncronas sin necesidad de establecer funciones *callback* o promesas al término de ejecución de la llamada. Para ello utiliza el paradigma de multitarea cooperativa, en donde los procesos ceden voluntariamente el control periódicamente o cuando están en estado inactivo para permitir la ejecución de múltiples procesos al mismo tiempo.

Es compatible con otras estructuras de *PHP* que permiten ejecución de múltiples procesos, como *fibers*[1], o aquellas que permiten conexión a *sockets*, como *streams*[2], entre otras compatibilidades.

Ejemplos particulares y explicativos de uso pueden encontrarse en los anexos de este documento, sección A.2.

[1]Sitio web documentación: `https://www.php.net/manual/en/language.fibers.php`
[2]Sitio web documentación `https://www.php.net/manual/en/intro.stream.php`

Capítulo 3

Análisis y Diseño

En este capítulo se profundizará en el análisis de la arquitectura de la aplicación actual en la sección 3.1 y el sistema de notificaciones en la sección 3.2. En la sección 3.3 se realizará una comparación entre las alternativas estudiadas para la solución del problema, para finalmente detallar el diseño de un nuevo microservicio de notificaciones en la sección 3.4.

3.1. Estructura de U-Cursos

U-Cursos es una plataforma desarrollada por Ucampus para la gestión de aprendizaje, *LMS*. Su estructura se basa en módulos y sigue al paradigma llamado *modelo-vista-controlador*, que permite separar las diferentes lógicas de la aplicación de acuerdo a las responsabilidades de cada componente.

3.1.1. Arquitectura

Como se mencionó anteriormente, U-Cursos presenta sus funcionalidades a través de módulos, que corresponden a unidades independientes entre sí, cada una con su propio controlador, vistas y base de datos. Sin embargo, estos pequeños módulos, a su vez, son administrados por el controlador principal de la aplicación, que también es un módulo, denominado `KERNEL`. Este módulo se encarga de manejar la información de gestión de permisos, *data* sensible, control de la ruta, entre otros, esto es, funciones e información transversal al resto de los módulos.

En la figura 3.1 se puede observar la estructura general de cómo se comunican y estructuran estos módulos. Cada módulo tiene conexión a su propia base de datos independiente, y a su vez, cada módulo tiene comunicación con el `KERNEL`, que maneja su propia base de datos. Esta comunicación pudiese ser bidireccional, pero dependerá del módulo y se diagraman en una sola dirección para establecer jerarquía.

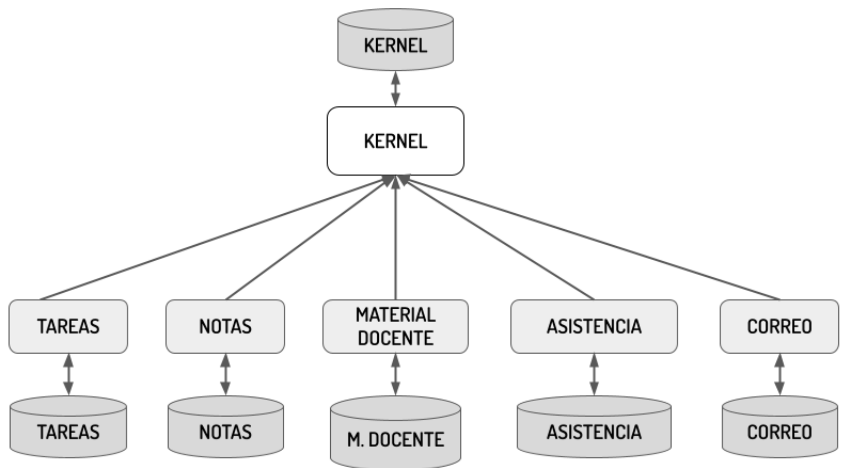


Figura 3.1: Diagrama de la estructura modular de U-Cursos. Fuente: Elaboración propia.

Desde el punto de vista de su uso, los módulos son las secciones o herramientas que pueden activarse en un usuario, curso, comunidad o institución por sus administradores. Algunos ejemplos de ellos serían los módulos Notas, Foro o Novedades de un determinado curso. En la figura 3.2 pueden observarse los distintos módulos disponibles para el curso CC6909-5 Trabajo de Título.

Figura 3.2: Módulos para el curso CC6909-5 Trabajo de Título, U-Cursos

Por otro lado, cada módulo puede tener sus propios *endpoints*, archivos *batch*, archivos *cron*, librerías, etc. que le permiten, en pocas palabras ser una unidad relativamente independiente de las otras, como una arquitectura de microservicios, con la diferencia de que los módulos no están separados en contenedores propios sino que son parte de una misma instancia de la plataforma a nivel macro.

3.1.2. Estándar Ucampus para API's

Ucampus tiene definido su propio proceso de creación de API's para los diferentes módulos de la plataforma. Para ello, se crea una carpeta `api` dentro de las dependencias del módulo, en donde se alojarán los archivos que correspondan a cada *endpoint*.

Cuando se realiza una *request* a un *endpoint* del módulo, esta posee una estructura particular en donde se indica el módulo, el *endpoint* y los parámetros necesarios para la consulta en la url, por lo que el controlador procesa este formato y llama al *endpoint* correspondiente. Una dirección de consulta podría verse como sigue:

$$.../api/0/usuario/nombre_usuario/pubsub/pendientes?pers_id = random\¬... \quad (3.1)$$

donde, desde *api* y en orden se indica que se debe consultar la api del módulo, entre los archivos en su versión 0, para el usuario correspondiente, en el módulo *pubsub*, el *endpoint* `pendientes` y todo lo que viene luego del símbolo ? son los parámetros que recibe el *endpoint*.

Cada *endpoint* puede realizar los cálculos o acciones para los que fue definido sin una estructura secuencial específica pero sí debe retornar al término con la función `KERNEL::streamAPI()`, que corresponde a un método del `KERNEL` para generar la respuesta a la *request* que llamó al *endpoint* en un principio. Permite también establecer el estado de la *request*, esto es, el estado de respuesta HTTP, el cual en general toma alguno de los siguientes valores:

- **200:** estado de respuesta satisfactoria.
- **400-499:** estado de error por parte del cliente.
- **500-599:** estado de error de parte del servidor.

3.2. Sistema de notificaciones actual

El sistema de notificaciones de U-Cursos conceptualmente funciona similar a como lo hacen los servicios de suscripciones en otras plataformas.

3.2.1. Funcionamiento del sistema

Para poder explicar de mejor manera el funcionamiento, se necesita aclarar y definir algunos conceptos:

- **Suscripción:** Una suscripción corresponde a la disposición de un usuario para recibir notificaciones de los cambios hechos en el contenido de un módulo.
- **Canales:** Los canales dentro de U-Cursos corresponden a un identificador de un módulo, compuesto por un tipo (usuario, curso o institución), un número identificador y el tipo de módulo (foro, notas, enlaces, etc). Para cada módulo distinto existe un canal y permite identificar los distintos módulos para los distintos cursos, comunidades e instituciones.

- **Notificaciones:** Dentro de U-Cursos, existen dos tipos de notificaciones: las notificaciones de la aplicación web, correspondientes a un pequeño ícono indicando la cantidad para cada curso, comunidad o institución; y las notificaciones *push*, que corresponden a las notificaciones para dispositivos móviles o web manejadas por el navegador.

En resumen, cada módulo en U-Cursos posee su propio canal a modo de identificador. Las suscripciones de los usuarios se pueden activar o desactivar para cada canal disponible desde la pestaña *Canales* en el perfil personal de U-Cursos, además de poder distinguir entre las notificaciones web y las notificaciones *push*, tal como se muestra en la Figura 3.3. Por defecto, todos los usuarios tienen suscripción automática para cada uno de los canales disponibles en sus cursos, comunidades e instituciones.

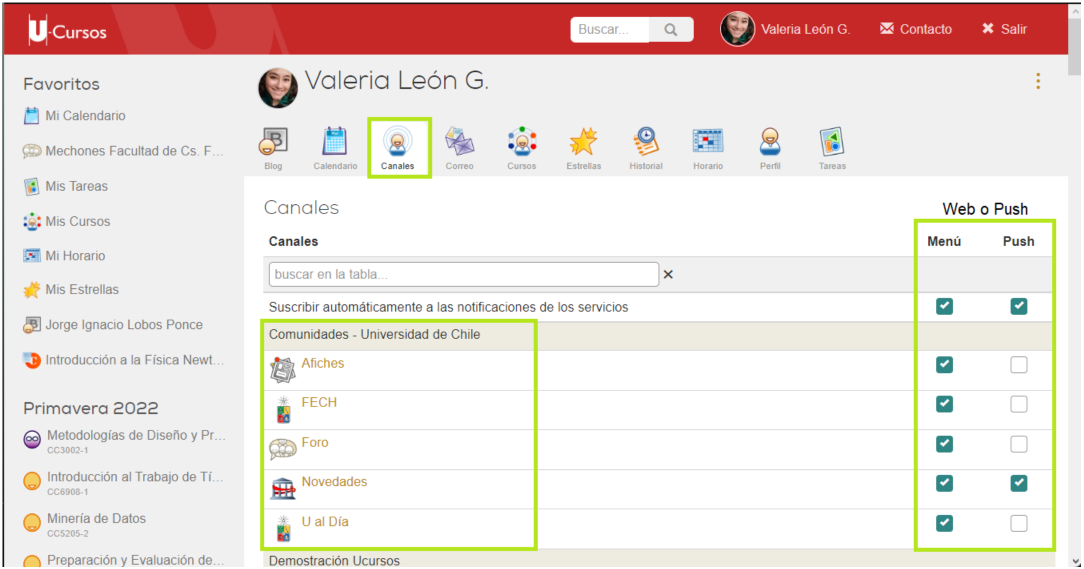


Figura 3.3: Configuración de notificaciones para cada canal disponible de un usuario

Si por ejemplo un usuario escribe un mensaje en el foro institucional, el módulo Foro se encarga de guardar el mensaje en su respectiva base de datos y posteriormente llama a una función del `KERNEL`para iniciar el guardado de la notificación. El `KERNEL`llama entonces a la librería `pubsub`, que es la que se encarga de realizar el almacenamiento en la base de datos `PUBSUB` de las notificaciones. Todas estas acciones suceden de manera secuencial en el código. Esto puede observarse de manera simplificada en la figura 3.4.

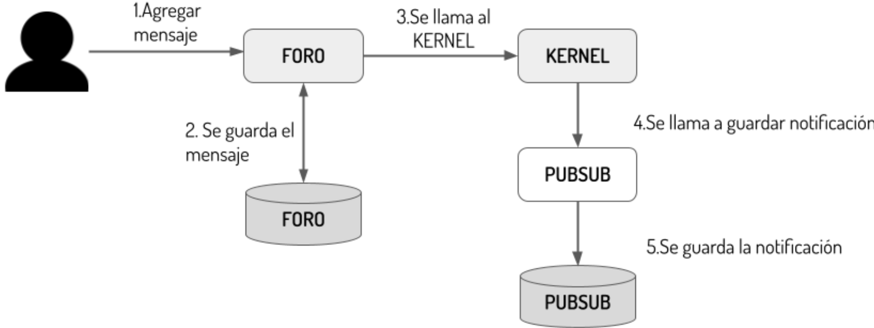


Figura 3.4: Flujo de envío de mensaje y guardado de notificaciones. Fuente: Elaboración propia.

3.2.2. Limitaciones de este sistema

Debido a la pandemia de COVID-19, gran parte de las actividades docentes sufrieron una migración a modalidad virtual, para lo cual se desarrollaron nuevos módulos dentro de la plataforma, como el módulo de Clase Virtual. Además, U-Cursos presentó un incremento abrupto de usuarios concurrentes en los últimos años, pasando de 7,000 a 22,000 usuarios. Lo mismo sucede al tener cursos con mayor cantidad de integrantes.

Dado lo anterior, durante la pandemia se presentaron varias caídas de la plataforma, debido a que una consulta específica a las tablas de suscripciones en la base de datos tomaba demasiado tiempo en responder, ya que esta base de datos tenía un tamaño aproximado de 11GB. Estas consultas paralizaban el controlador y afectaban el funcionamiento de los servidores que dan soporte a U-Cursos al tener que notificar a miles de usuarios sobre el comienzo de una nueva clase virtual, por ejemplo, en los horarios más concurridos, como lo son martes y jueves a las 10:15hrs. El equipo de desarrollo estimaba aproximadamente 15,000 usuarios simultáneos que daban paso a estos incidentes.

Siguiendo el mismo ejemplo de un párrafo anterior, si un usuario quería publicar un mensaje en el Foro, debido a la secuencia de acciones, en el último paso donde se almacena la notificación se bloqueaba la ejecución porque la base de datos demoraba en responder. Ya que las llamadas al `KERNEL`para consultar sobre notificaciones son globales y no específicas de ese módulo, todos los usuarios que ingresaran a la plataforma se verían afectados por la demora o caída del servicio. En concreto, U-Cursos estaba caído.

En el transcurso de estos acontecimientos, se generó una solución parche para el problema, la cual consistió en separar la tabla `SUSCRIPCIONES` en tablas diferenciadas por el dígito verificador de los usuarios, además de aislar la base de datos completa del sistema

de suscripciones en un servidor distinto al resto de la plataforma, disminuyendo la carga al resto de los servidores. No obstante, esto no fue suficiente para evitar caídas y fue necesario además configurar un tiempo máximo de ejecución para la consulta, lo que logró evitar las caídas de las plataformas, pero las plataformas demoraban en mostrar contenido en períodos de alta carga y finalmente no se muestra nada asociado a notificaciones si la consulta excedió el tiempo de respuesta.

Sin embargo, ninguna de estas soluciones es óptima ni estable, por lo que se requiere diseñar e implementar un nuevo sistema de notificaciones, que sea independiente a la plataforma en términos de recursos del controlador. Idealmente, similar a un micro-servicio que sea accesible mediante llamadas dinámicas que no bloqueen la plataforma. Además, no se cuentan con estadísticas asociadas a estas caídas ya que recién el año 2022 se comenzó a guardar datos de historial de incidentes.

En el resto del documento, se hablará indistintamente del sistema, servicio o librería actual de notificaciones refiriéndonos al mismo esquema explicado recientemente.

3.2.3. Arquitectura del sistema

Como se mencionó en otras secciones anteriores, el principal módulo de software de la plataforma de interés para este trabajo es el controlador principal y cómo se relaciona con la base de datos encargada del sistema de notificaciones.

Primeramente, en la Tabla 3.1 se muestra un resumen de la base de datos y sus tablas, con una descripción de lo que se guarda en cada una de ellas.

Tabla	**Datos**
Canales	Guarda todos los canales de cada módulo, con su respectivo identificador
Configuraciones	Configuraciones de cada usuario con respecto al recibo de notificaciones
Mensajes	Guarda las notificaciones y su contenido, autor y canal correspondiente
Suscripciones	Guarda las suscripciones de un usuario a un canal

Tabla 3.1: Cuadro resumen de tablas de la base de datos y su contenido

El controlador posee una clase relacionada llamada *PubSub*, que contiene las funciones encargadas de acceder a la base de datos para cada funcionalidad necesaria, ya sea para consultar, agregar o actualizar información. A continuación se presenta un cuadro en donde se observa cada función o grupos de funciones con su tarea asociada y las tablas de la base de datos que se consultan.

Función	Funcionalidad	Tabla en la Base de Datos
timeline() readSub()	Consultar información para mostrarla en el historial de un usuario al ingresar a U-Cursos	**Lectura:** 1. Mensajes 2. Suscripciones 3. Canales
pub()	Ingresar un nuevo mensaje que genera una notificación para el canal correspondiente y sus suscriptores	**Lectura:** 1. Canales **Escritura:** 1. Canales 2. Mensajes
readPub() cntPub()	Consultar información para ser mostrada por cada canal	**Lectura:** 1. Mensajes 2. Suscripciones 3. Canales
sub() initCanales()	Crear un canal e inicializar suscripciones en él	**Escritura:** 1. Canales 2. Suscripciones

Tabla 3.2: Flujo lógico de funciones y tablas en la base de datos a las que acceden

La Tabla 3.2 es una referencia a nivel macro de algunas funciones y su relación con las tablas de datos, pero existen las contrapartes en funcionalidad para eliminar publicaciones, eliminar suscripciones, limpiar y modificar configuraciones, entre otras, que no se añadieron para mantener más limpio el flujo general.

3.3. Comparación de alternativas

En esta sección, se discuten las ventajas y desventajas de distintas alternativas de solución, presentando en primer lugar un cuadro resumen de las alternativas con los indicadores utilizados para compararlas, para posteriormente ahondar en los detalles de cada una.

3.3.1. Sistemas de notificaciones en el mercado

A continuación se presenta la comparación de las alternativas detalladas en la Sección 2.1, enfocada en los indicadores de costo extra por uso o licencias, funcionalidad, mantención y extensibilidad a modo de resumen general.

Indicador	Firebase	Amazon	Ably	Microservicio
Costo de uso	Ninguno	Ambiguo	$\approx$ \$107	Ninguno
Funcionalidad	Enfoque en envío Díficil manejo de canales	Enfoque en envío Difícil manejo de canales	Enfoque en envío Soporte de canales	Enfoque en envío y guardar notificaciones Soporte de canales
Mantención	Potencialmente solo configuración inicial	Potencialmente solo configuración inicial	Potencialmente solo configuración inicial	Actualizaciones manuales y soporte
Extensibilidad	Sin acceso a código fuente	Sin acceso a código fuente	Sin acceso a código fuente	Con acceso a código fuente

Tabla 3.3: Comparación de alternativas

En la Tabla 3.3, el apartado de Funcionalidad hace referencia al soporte de funcionalidades que actualmente el servicio de notificaciones de U-Cursos posee. A su vez, la extensibilidad hace referencia a poder añadir nuevas funcionalidades a futuro del sistema utilizado. A excepción de un microservicio propio, no es posible añadir características al servicio, sino que debe hacerse en el *back-end* de U-Cursos.

En adición a estos indicadores también se tuvo en cuenta los riesgos de mantener información relacionada a los usuarios de las plataformas de Ucampus en herramientas de terceros, además del nivel de seguridad en las conexiones con dichos servicios. Esto debido a que si bien la información de notificaciones de un usuario por sí misma no es una información sensible si no se cuenta con el identificador de una persona, al poseerlo puede revelar información respecto a los cursos, las comunidades o instituciones a las que está ligado, además de las actividades o novedades dentro de un canal particular. Esta información es sensible y confidencial para la Universidad, por lo que si las conexiones no son seguras o si la tercera parte sufre ataques de filtración de datos, esta información podría ser expuesta y traer consecuencias graves a la institución.

Si bien Ucampus al hacerse cargo de estos datos en sus sistemas propios no está exento a estos riesgos, *Amazon SNS* ha tenido multas por no cumplimiento de las normativas de la *GDPR*[1][13], además de numerosas filtraciones y ataques en los últimos años. Por su parte, una mala configuración de *Firebase* en una aplicación puede romper con los sistemas de permisos de la misma, permitiendo a usuarios sin autenticar tener acceso a la base de datos completa al conectarse a una API, como lo muestra el estudio realizado por un equipo de *Comparitech* en mayo de 2021[8].

Por lo anterior, sumado al alcance de dichas plataformas y la magnitud de las consecuencias para los usuarios si se llegase a filtrar la información, es que se prefiere un manejo local de Ucampus de la información, al menos para esta primera iteración del proyecto.

Firebase

Como se mencionó en la Sección 2.1: Sistemas de notificaciones en el mercado, *Firebase*, de momento, es un servicio gratuito. Los posibles costos vienen asociados a querer utilizar

[1]Reglamento General de Protección de Datos europeo, sitio web: `https://gdpr-info.eu`

Firebase Realtime Database, ya que en caso de tener aproximadamente 12GB de información almacenado, similar al tamaño de la base de datos actual de U-Cursos para el sistema de notificaciones, el costo es de aproximadamente USD$ 55 mensuales.

Se podría querer utilizar esta tecnología debido a que se integra de buena forma con *FCM* y otras herramientas de Google. Además FRD está optimizado y enfocado en la actualización en tiempo real de los datos en todos los dispositivos suscritos en *FCM*.

En relación a las funcionalidades permitidas por *FCM* para soportar las características actuales de U-Cursos, el manejo del sistema de canales es complicado, ya que la segmentación ofrecida por *FCM* no contempla la existencia de miles de canales diferentes, que requeriría una mayor cantidad de tablas en *Firebase* o un diseño diferente de sistema de entrega de notificaciones. Por otra parte, al querer extender y añadir nuevas funcionalidades en el sistema de notificaciones, se debe hacer desde el backend de U-Cursos necesariamente.

Amazon SNS

Como se mencionó anteriormente, *Amazon SNS* solo posee 1M de solicitudes gratuitas al mes, definiendo solicitud como notificaciones *push* móviles o paquetes de 64KB de datos enviados, 1.000 notificaciones por e-mail y 100.000 notificaciones vía HTTPS. Sín embargo, también pueden aplicar cargos por transferencia de datos, solicitudes a la API, entre otros.

Considerando el envío de al menos 1M de notificaciones push móviles, 1M de notificaciones push web y 10.000 e-mails, puede tenerse un cobro extra de un poco más de USD$ 1 para utilizar esas cantidades. Pero en otros apartados del sistema de cobros se menciona que las solicitudes a la API como publicaciones de tópicos tienen costo adicionales por millón de solicitudes extra de USD$ 0,50 y USD$ 0,0272 por cada 1GB de datos cargados, mientras que los mensajes de suscripción tienen un costo adicional menor, pero que también se factura por millón de solicitudes y GB de carga [6].

En cuanto a cantidad de notificaciones, podríamos no asumir cargos extras para el caso de U-Cursos, dada la cantidad de usuarios de la plataforma, que ronda aproximadamente los 50.000, al menos en dispositivos móviles de tipo Android. Esta cantidad podría generar cerca del millón de notificaciones mensuales. Lo anterior considerando que el tamaño de dichas notificaciones se mantiene bajo los 64KB, pero, como puede observarse, es difícil hacer un cálculo incluso aproximado de costos, debido a lo volátil que puede ser la cantidad de notificaciones enviadas al mes en U-Cursos.

Por otra parte, al igual que con *FCM*, el manejo del sistema Pub/Sub con numerosa cantidad de canales es complicado y requeriría de lógica adicional a la actual dentro de U-Cursos. Lo mismo en caso de querer añadir nuevas características al sistema de notificaciones.

Ably

Para este caso, se detallarán a detalle los tres planes disponibles y los límites de uso de acuerdo a la información entregada por la propia página de precios de *Ably* [4].

El plan gratuito incluye 6M de mensajes enviados, 200 canales *peak* y 200 usuarios *peak*. El plan prepago (*Pay-As-You-Go)* tiene cobro adicional por millón de mensaje y miles de canales y usuarios *peak*. Finalmente, el plan personalizado para empresas no especifica un precio estandar debido a que se ajusta a las necesidades acordadas entre la empresa y *Ably*, pero los números de los parámetros anteriores son ilimitados y el soporte técnico es en modalidad 24/7, mientras para los dos planes más económicos se contemplan las horas laborales de UK/US este.

Para el guardado de notificaciones, el plan gratuito solo guarda los mensajes por 24hrs, por lo que al igual que para *FCM* y *Amazon SNS*, se debe trabajar en la lógica de guardado de notificaciones si se quiere mantener información en la base de datos actual. Este guardado temporal suma a la cuota mensual de mensajes gratuitos. Además, al igual que *Amazon SNS*, un mensaje es en realidad medido por tamaño, 2KiB[2], por lo que una notificación de 50KiB en un canal con 100 suscriptores cuenta como 25 mensajes publicados y 2.500 mensajes a enviar a los usuarios.

Por otra parte, se deben tomar muy en cuenta las conexiones *peak*, ya que tomando como base los números que se manejaban en pandemia con clases virtuales solo en FCFM, se tendrían 6.441 usuarios con cálculos y conexiones simultáneas, lo que en *Ably* serían USD$ 75 extras al mes [4], además de una base de aproximadamente USD$ 32 por 1.000 canales y 1.000 usuarios *peak*. Un gasto extra que podría incluso ser mayor si se hiciese un cálculo exacto de la cantidad de notificaciones que se manejan también en otros canales, como foros institucionales y comunidades con usuarios activos. Por ejemplo, solo en FCFM pueden llegar a haber 100 mensajes nuevos en el foro mensualmente, con sus interacciones. Incluso si la persona no tiene activadas las notificaciones, estas deben mostrarse en la barra lateral de la aplicación, por lo que de todas maneras son ingresadas como notificaciones a enviar en la base de datos.

Finalmente, igual que para *Amazon SNS* y *FCM*, las características adicionales que se quieran introducir a la aplicación deberán ser manejadas por el backend de U-Cursos en su totalidad. Por otro lado, una ventaja de *Ably* en relación a las opciones anteriores, es que este si posee soporte y estructura para una gran cantidad de canales y conexiones simultáneas, pero con un alto precio en comparación a *Amazon SNS* y *FCM*.

Microservicio propio de notificaciones

Para el caso de un microservicio propio, se propone que al menos la primera versión corresponda a un módulo de usuario de U-Cursos, es decir, un módulo accesible desde el perfil de un usuario. Con esta estructura, no se tienen gastos adicionales por concepto de licencias o suscripciones, ya sea a un servicio de notificaciones o un servicio de almacenamiento, ya que se utilizarán herramientas ya disponibles, bases de datos propias y librerías de libre uso.

En caso de querer añadir nuevas características al sistema de notificaciones, queda a criterio del equipo de desarrollo de Ucampus extender el microservicio o el *backend* de U-Cursos, al tener acceso a ambas opciones. Sin embargo, se deben realizar las mantenciones o actualizaciones necesarias de manera manual, es decir, estas tareas quedan a cargo de

[2]KibiByte: unidad de medida que corresponde a 1024B o $2^{10} bytes$. [19]

Ucampus y no de un tercero.

En cuanto a las funcionalidades ya existentes, es posible trasladar casi por completo el código ya existente y reemplazar las llamadas a funciones desde el controlador por llamadas asíncronas a una API propia del sistema de notificaciones independiente.

3.3.2. Librerías Pubsub alternativas

Para el caso de las librerías opcionales, se decidió no hacer uso de ninguna de ellas, pues el hacerlo requería un estudio más detallado de las estructuras y metodologías de cada una, además de su respectiva conexión a la base de datos. El tiempo requerido para ello escapa el alcance del proyecto, además el problema principal que se trata de solucionar es el uso que el `KERNEL` y el resto de los módulos le da a la librería actual, no la conexión entre la librería y la base de datos.

Por otro lado, los métodos que poseen las librerías consultadas no son los suficientes para suplir todas las responsabilidades con las que cuenta actualmente la librería, que soporta un manejo diferenciado de notificadores y métodos de obtención de notificaciones, además de diferentes operaciones de las que se encarga la librería `pubsub` cuando un usuario inicia sesión. Esto implicaría una adaptación de cualquier librería a los usos actuales de las notificaciones en U-Cursos, pues el objetivo de este trabajo es asegurar que las funcionalidades actuales del sistema se mantienen aún haciendo cambios al sistema de notificaciones.

Por lo anterior, es que se decide trabajar con la librería actual y adaptar las llamadas y usos que se le dan a ella, además del controlador encargado, en lugar de adaptar una externa.

3.3.3. Llamadas asíncronas en PHP

Para el caso de las llamadas asíncronas desde el lado del servidor con *PHP* se decide que, si bien los métodos y librerías encontradas podrían ser buenos candidatos para asegurar la asincronía que se espera del sistema, incluir estas nuevas herramientas no asegura un buen rendimiento del sistema por sí solo. Por otra parte, ya existen estructuras del sistema de notificaciones que se encuentran separadas del resto de la infraestructura de la plataforma, por lo que las llamadas asíncronas como única medida contribuyen a parchar el estado actual del servicio en lugar de generar una solución a largo plazo para el manejo de notificaciones.

No obstante, el requerimiento de llamadas asíncronas se mantiene, pero se decide utilizarlas en aquellos lugares donde se requiere información desde la base de datos por el lado del cliente, utilizando *JQuery* para realizar las consultas en conjunto con *JavaScript* para actualizar la información desplegada. Por su parte, se decide que para ingresar información a la base de datos de parte del servidor el modelo debe poseer un sistema de respuesta rápida que no bloquee el `KERNEL` por mucho tiempo.

3.4. Diseño de nuevo módulo de notificaciones

A partir de lo comentado en la sección anterior, es que se decide entonces diseñar un microservicio propio para la plataforma, ya que el desarrollo, mejoramiento y mantención de este queda a cargo del equipo de Ucampus. Por tanto, es importante que esté conformado por herramientas y lenguajes familiares. En particular, este microservicio corresponderá a un módulo de U-Cursos encargado especialmente de manejar las notificaciones de usuario.

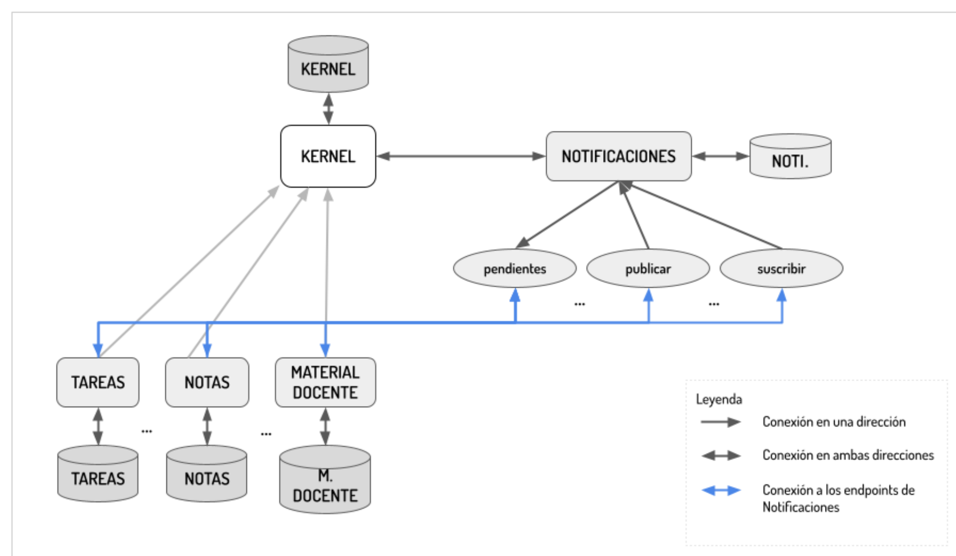


Figura 3.5: Diagrama alto nivel del módulo Notificaciones en U-Cursos. Fuente: Elaboración propia.

En la figura 3.5 se puede observar el esquema lógico propuesto en un alto nivel, donde se tiene que el servicio de notificaciones es un módulo de U-Cursos que puede ser consultado por otros módulos e incluso el `KERNEL`, en alguno de sus *endpoints*, como lo son `pendientes`, `suscribir` y `publicar`, conexión representada por las flechas azules, que indican una solicitud web al *endpoint* correspondiente. Estos *endpoints* ingresan o leen información del módulo, quien se conecta a la base de datos de Notificaciones.

A grandes rasgos, este nuevo módulo debe cumplir con asincronía y robustez, permitiendo que la plataforma mantenga disponibilidad incluso en momentos de alto estrés y, específicamente, en momentos de caída en el servicio de notificaciones.

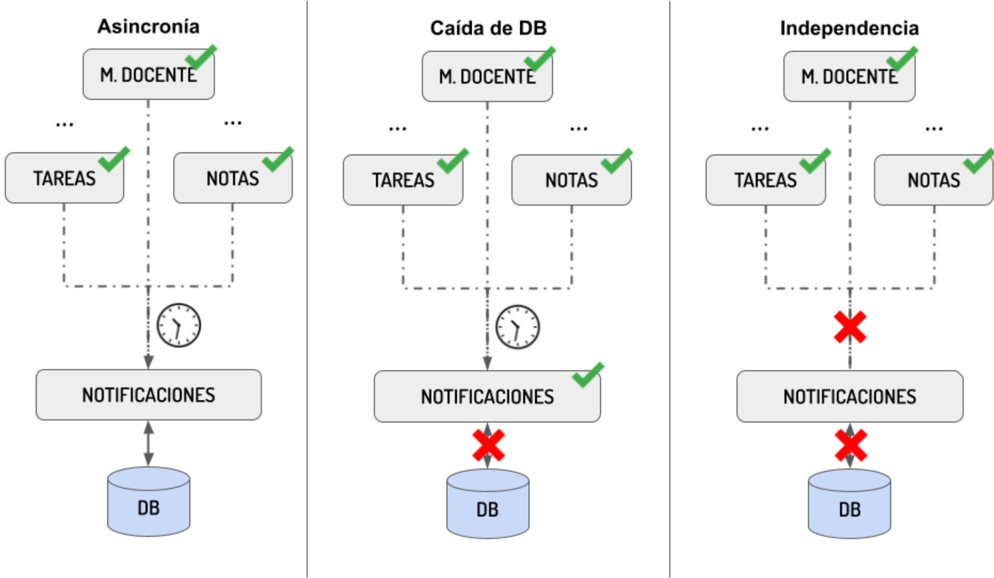


Figura 3.6: Diagrama alto nivel de los principales requisitos del sistema. Fuente: Elaboración propia.

En la figura 3.6 es posible distinguir los tres principales requisitos en los que se enfocó el diseño e implementación del nuevo servicio de notificaciones del sistema. Los relojes entre las conexiones de otros módulos y el sistema corresponden a consultas no bloqueantes, ya sean asíncronas o de rápida respuesta. Las flechas verdes significan disponibilidad del módulo o conexión correspondiente y las cruces rojas indican lo contrario. En primer lugar, se necesita que el servicio pueda ser consultado sin bloquear el funcionamiento del resto de los módulos que lo consultan, incluso aunque se deba esperar por cierto tiempo la respuesta a dichas consultas, esto es, asincronía. En segundo lugar, es requisito que si la base de datos del módulo no está disponible, el resto de los módulos consultantes y la navegación entre ellos esté disponible. Finalmente, se requiere que el sistema sea independiente a los demás módulos y que incluso estos pudiesen mantener sus funcionalidades sin un servicio de notificaciones disponible.

En pos de cumplir estos requisitos, se decide que un nuevo módulo permite desarrollar las funcionalidades solicitadas en una primera versión, sin perjuicio de que a futuro puede ser incluso una instancia distinta a U-Cursos.

3.4.1. Creación módulo de notificaciones

El nuevo módulo notificaciones se encarga entonces de manejar las solicitudes de lectura y escritura a la base de datos, retornando la información cuando sea necesario y gestionando el ingreso de información en los otros casos. De acuerdo a lo explicado a lo largo de las secciones anteriores, se decide manejar las solicitudes de lectura y escritura de manera diferenciada, las primeras desde el lado del cliente y las segundas desde el lado del servidor. Dado esto existen dos necesidades claras: que las consultas de parte del cliente sean seguras y que las consultas del lado del servidor sean robustas.

Para suplir las necesidades recién descritas, se necesita de un *buffer* de encolamiento de tareas para robustez, lo que se logra con una librería `tasks` ya existente en el sistema. Y por otro lado, debido a que se posee la misma sesión del usuario desde el lado del cliente y a su vez esto es un módulo perteneciente a U-Cursos, la seguridad de dicha conexión ya estaría provista por el `KERNEL`.

De manera concreta, el módulo poseerá 6 *endpoints* que cubren una gran parte de los casos de uso del servicio, un *buffer* de encolamiento de tareas a realizar y una interfaz que muestra los mensajes pendientes del usuario cuya sesión se encuentra inicializada en el navegador. Además, este módulo poseerá y hará uso de la librería `pubsub`, antes perteneciente al `KERNEL` , y una réplica de la librería `tasks` para el manejo del encolamiento, entre otros componentes.

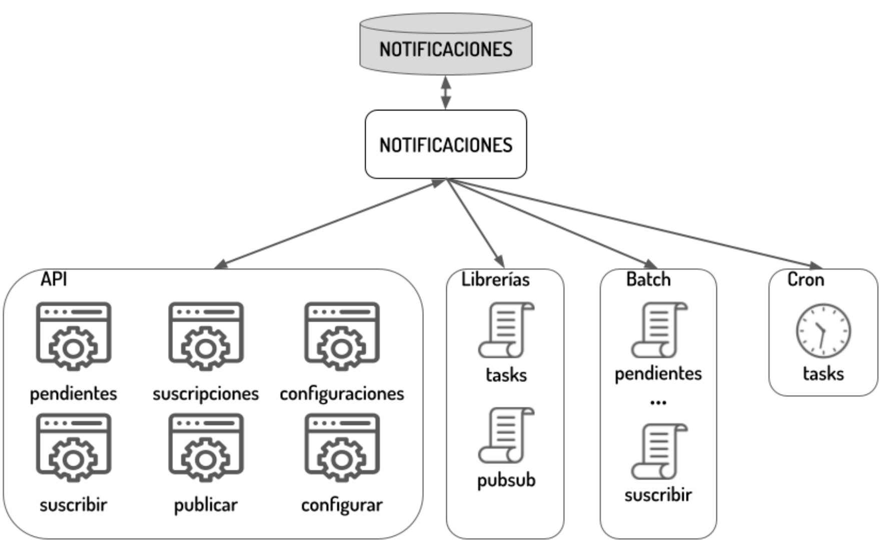


Figura 3.7: Diagrama de los componentes del módulo Notificaciones. Fuente: Elaboración propia.

En la figura 3.7 es posible observar los diferentes componentes que harán parte del módulo Notificaciones. El primero de ellos corresponde a la API del módulo, que constará de 6 *endpoints*, una para cada funcionalidad principal del servicio actual de notificaciones que deben mantenerse. Luego se tienen las librerías del módulo, las cuales corresponden a `pubsub` y `tasks` que se encargan del manejo de datos desde y hacia la base de datos. En tercer lugar se tienen los archivos `batch`, que como se explicará más adelante en detalle, hacen parte del sistema de encolamiento de tareas del módulo, siendo los archivos que finalmente ejecutarán las instrucciones de ingreso de nueva información a la base de datos `PUBSUB`. Por último se tiene un archivo `crontab` que es el indicado de correr las tareas pendientes del módulo.

Además, el módulo cuenta con un componente `web` que contiene archivos de configuración para el controlador del módulo y código *JavaScript* y una carpeta `template` con archivos *HTML* que hacen parte de la interfaz gráfica del módulo.

En las subsecciones siguientes se explicará en profundidad la composición y funcionamiento lógico del módulo, además se mencionarán los requisitos de arquitectura física del

servicio.

Arquitectura lógica

De manera lógica, este módulo posee *endpoints* que permiten hacer consultas síncronas o asíncronas, un encolamiento de solicitudes y una interfaz simple de mensajes pendientes. Se discutirá cada uno de ellos a continuación.

Endpoints

A partir del estudio del sistema actual realizado en la sección 3.2, el cual permitió establecer las funcionalidades que cumple dicho sistema, es que se definen dos tipos de *endpoint* **lectura** y **escritura**.

- **Endpoints de escritura:** son *endpoints* que se llaman para poder ingresar nueva información a la base de datos, por lo que su acceso y uso debiese ser solo desde el lado del servidor. Al estar el *backend* de la plataforma en lenguaje *PHP* principalmente, no es posible realizar llamadas asíncronas a estos *endpoints*. Para ello es que se establece que el módulo posea un *buffer* de solicitudes, que permita una respuesta rápida a la solicitud de escritura sin dejar bloqueado el controlador en la misma. El diseño de este *buffer* se explicará con mayor detalle más adelante.
- **Endpoints de lectura:** son *endpoints* que se consultan para extraer información de la base de datos, por lo que su acceso y uso puede realizarse desde el lado del cliente mediante llamadas asíncronas con *jQuery/Ajax*, permitiendo que el controlador no se bloquee en la consulta, pero el módulo consultante pueda esperar en paralelo y desplegar la información cuando esta esté disponible.

A partir de esta categorización de *endpoints* es que se establecen 6 funcionalidades que posee actualmente la librería PubSub y que deben mantenerse para el correcto funcionamiento de la plataforma. Por ello, para cada una de las funcionalidades se crea un `endpoint` asociado, los que son:

- **Pendientes:** este *endpoint* apunta a la funcionalidad de mostrar la información de las notificaciones pendientes al ingresar a U-Cursos en el módulo Canales, por lo que va a buscar todas los mensajes de todos los canales a los que el usuario está suscrito, retornando un *JSON* con la información.
- **Publicar:** este *endpoint* es uno de los principales y más utilizados, ya que se encarga de escribir en la base de datos las nuevas notificaciones generadas en el resto de los módulos. Encola las solicitudes en el *buffer* y retorna el estado de éxito o fracaso de la *request*.
- **Suscribir:** este *endpoint* se utiliza principalmente al inicio de cada período académico, ya que a diferencia de otras plataformas la suscripción a canales es automática de acuerdo a los cursos inscritos. Sin embargo, en la opción Configurar del módulo Canales

es posible gestionar el estado de las suscripciones para recibir o no las notificaciones asociadas a cada canal. Este *endpoint* también es de escritura, y al igual que `publicar` encola las solicitudes y retorna el estado de éxito o fracaso de la *request*.

- **Configurar:** este *endpoint* permite cambiar la configuración de las notificaciones para cada notificador posible. Por ejemplo, configurar si se quieren recibir notificaciones *push* o *HTML*. Es un *endpoint* de escritura, pero es el único de ellos que no encola solicitudes, pues su uso es poco recurrente y no crítico.
- **Configuraciones:** este *endpoint* permite consultar y extraer la información de las configuraciones de notificaciones asociadas a un usuario. Es un *endpoint* de lectura y funciona de la misma manera que `pendientes`.
- **Suscripciones:** este *endpoint* permite consultar todas las suscripciones y su estado de un usuario. Al igual que `configuraciones` es un *endpoint* de lectura que puede ser consultado asíncronamente.

Todos estos *endpoints* se alojan en una carpeta `api` dentro del módulo Notificaciones.

Buffer de solicitudes

El *buffer* de solicitudes está pensado para que el sistema sea capaz de recibir solicitudes de escritura a la base de datos que serán completadas a medida que el módulo tenga la capacidad de hacerlo, pero al momento de consultar un *endpoint* de escritura, el servicio tome la solicitud, la encole y rápidamente pueda responder de vuelta la consulta al módulo respectivo, sin bloquear el controlador por mucho tiempo. Además, el poseer un *buffer* de encolamiento permite que incluso si la base de datos está caída, se tenga un respaldo de las solicitudes de escritura que deben ser agregadas.

Para incorporar esta funcionalidad al módulo se replicó una librería que ya poseía el módulo `KERNEL`: las tareas. Estas tareas son exclusivas para el módulo Notificaciones, independientes del funcionamiento de las tareas para el `KERNEL`, poseen su propia tabla dentro de la base de datos y para poder realizarse se tiene un archivo que corre automáticamente cada cierto período de tiempo, denominado `cron`.

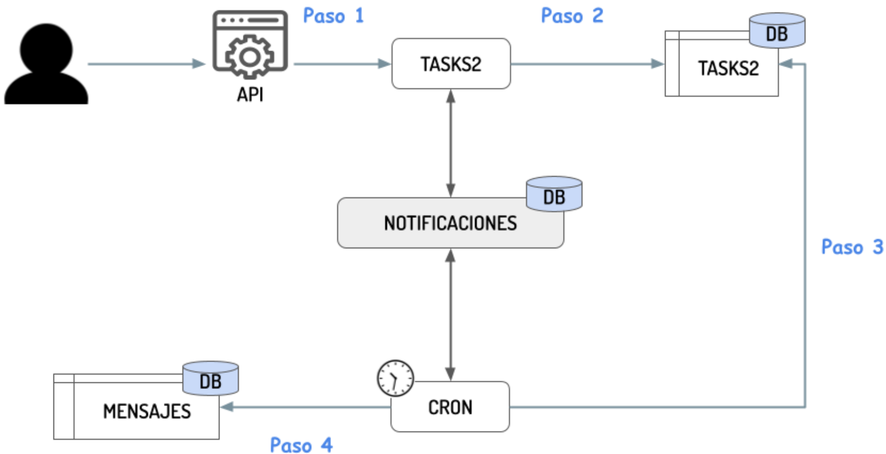


Figura 3.8: Diagrama del flujo del encolamiento de tareas

En la figura 3.8 se puede observar el flujo general del funcionamiento de las tareas. Cuando un usuario realiza una consulta a un *endpoint* de escritura este llama a la clase `tasks2` para que añada una tarea a la tabla, lo que sucede en los pasos 1 y 2 del diagrama. El paso 3 sucede cada cierto período de tiempo, donde un archivo `cron` extrae las tareas que deben realizarse desde la tabla `TAKS2` y va, en el paso 4, a agregar nuevos mensajes en la tabla `MENSAJES` de la base de datos `PUBSUB` para este caso, pero podría ser cualquiera de las otras tablas dentro de la base.

De manera más explícita el encolamiento de tareas posee los siguientes componentes:

- **Clase `tasks2`:** Esta clase corresponde a la librería que posee los métodos para manejar las tareas en la base de datos. Estos son:
  - `get`: método para extraer tareas de la tabla. Dependiendo de los parámetros entregados pueden traerse todas las tareas, acotadas por identificador o por estado, que puede ser pendiente o ejecutando.
  - `del`: método para eliminar la tarea especificada de la tabla de datos, lo que sucede una vez que la tarea concluye su ejecución.
  - `add`: método para añadir una tarea a la tabla. Recibe como parámetro la ruta al archivo que representa la tarea a ejecutar, por ejemplo, un *script* para publicar mensajes.
  - `upd`: método que permite actualizar una tarea, recibiendo como parámetro la ruta al *script* a ejecutar y el estado al que se actualiza la tarea.
  - `enEjecucion`: método para consultar la tarea que actualmente posee el estado `EJECUTANDO`.
- **Archivos /batch:** corresponden a los *scripts* que serán ejecutados por el `cron`. Estos archivos son 2 y equivalen a 2 de los *endpoints* de escritura que se tienen: `publicar` y

`suscribir`. La única diferencia en contenido de estos archivos respecto a los *endpoints* es que en estos efectivamente se llaman a los métodos respectivos de la librería `pubsub` que permiten añadir información correspondiente a la base de datos.

- **Cron:** es un archivo que correo automáticamente cada cierto período de tiempo configurable. Este archivo es el encargado de ir a buscar tareas a la base de datos, ejecutarlas y eliminarlas de los pendientes.

- **Tabla TASKS2:** corresponde a la tabla añadida a la base de datos para el manejo de las tareas. Sus atributos son:
  - `TAS_ID`: identificador de cada tarea y llave primaria de la tabla.
  - `TAS_URL`: ruta del archivo a ejecutar.
  - `TAS_ESTADO`: estado en que se encuentra una tarea, los valores posibles son `PENDIENTE` o `EJECUTANDO`.
  - `TAS_FECHA_M`: corresponde a la fecha de última modificación de la tarea.
  - `TAS_FECHA_C`: corresponde a la fecha de creación de la tarea.

Interfaz Gráfica del Módulo Debido a la naturaleza de este módulo, no posee interfaces gráficas complejas ni está enfocado en usabilidad de usuario. Sin embargo, se añade al diseño una interfaz simple que permite consultar los mensajes pendientes del usuario, en lugar de realizarlo en el módulo Canales antes mencionado. Lo anterior con la finalidad de poder obtener las notificaciones de un usuario de manera más directa, pudiendo a futuro incluso adquirir la responsabilidad del módulo Canales respecto a las notificaciones, canales y suscripciones de un usuario.

Arquitectura física

Actualmente, la arquitectura física se compone del *stack* tecnológico de Ucampus, que vendrían siendo *PHP, JavaScript, CSS y HTML* para el *frontend*, *PHP* para el *backend*, además de los servidores a su cargo. La base de datos de Notificaciones ya se encuentra separada en un servidor distinto al resto de las instancias de la plataforma y corresponde a una base de tipo *MySQL*.

El *stack* anterior se mantiene para el módulo de notificaciones. Sin embargo, al momento de su implantación, es importante que se realicen los cambios necesario en la funcionalidad `tasks` de forma que su tabla asociada quede en una base de datos diferente a la del resto de `PubSub`.

Capítulo 4

Implementación

En este capítulo se hablará sobre la implementación del sistema desarrollado, especificando los cambios realizados al resto de los módulos en la sección 4.1, el funcionamiento de los *endpoints* en la sección 4.2, el encolamiento de solicitudes en la sección 4.3, el uso de llamadas asíncronas al módulo en la sección 4.4 y finalmente el protocolo de conexión a los *endpoints* en la sección 4.5.

4.1. Separación notificaciones de U-Cursos

Primeramente, se debieron comentar para dejar sin efecto, todas las llamadas a la instancia `PubSub` dentro de todos los módulos y además eliminar la importación de la librería `pubsub` en las configuraciones de la aplicación, consiguiendo una versión de la plataforma funcional sin servicio de notificaciones.

Dichas instancias se dejaron en formato comentario en lenguaje *PHP* debido a que se contabilizaron aproximadamente 100 de ellas a lo largo de todos los módulos con diferentes usos y procesamientos de los datos consultados, por lo que se decidió dejar constancia de ellas dentro del mismo código para su posterior reemplazo con el método de conexión correspondiente al *endpoint* consultado.

Se tuvo especial cuidado al realizar este proceso dentro del módulo `KERNEL` debido a su importancia para el resto de la aplicación, asegurando su correcta funcionalidad sin las llamadas a los métodos de la librería `pubsub`.

En el anexo A.1 se puede encontrar un listado exhaustivo de todas las instancias y llamadas a la librería encontradas a lo largo de los módulos que conforman U-Cursos. En la siguiente sección se explicará la implementación de los endpoints de la API y el resto de las funcionalidades.

4.2. Implementación de endpoints de la API

Como se comentó en la sección 3.4, para el manejo de solicitudes relacionadas a notificaciones se implementaron seis diferentes *endpoints*, los cuales permiten escribir o leer en la base de datos y se muestran a continuación.

4.2.1. Endpoints de lectura

Dentro de esta categoría se encuentran los *endpoints* `pendientes.0.php, configuraciones .0.php y suscripciones.0.php`. Se encargan esencialmente de ir a leer información a la base de datos para luego retornarla en formato JSON.

En el código 4.1 podemos ver la estructura general de un *endpoint* de lectura de la base de datos, en donde se tienen cuatro secciones del proceso de lectura: primero se extraen los parámetros necesarios de la *request*, luego se realiza la consulta correspondiente, se procesan y formatean los datos de ser necesarios y finalmente se envían en la respuesta de la *request* mediante el método `KERNEL::streamAPI()`.

En este caso particular, el *endpoint* `pendientes` retorna la información de las notificaciones pendientes de un usuario.

```
1 <?php
2 require_once( '../include/pubsub.class.php' );
3 require_once('../web/config.php');
4
5 $pers_id = $_REQUEST['pers_id'] ?? $_SESSION['kernel']['persona']['pers_id'];
6 $not_id = $_REQUEST['not_id'] ? $_REQUEST['not_id'] : 'HTML';
7
8 $todos_mensajes = PubSub::readSub( $pers_id, $not_id );
9
10 foreach( $todos_mensajes as $id => $m ) {
11   if( $m['canal']['servicio']['C_EXT_URL'] ) unset( $todos_mensajes[$id] );
12   if( ! ( intval( isset( $m['canal']['servicio']['visible'] ) ) ) ) unset( $todos_mensajes[
      $id] );
13 }
14
15 $canales = UTIL::agrupar( $todos_mensajes, 'can_id' );
16 foreach( $canales as $can_id => $ms ) {
17   $m = reset( $ms );
18   $servicio = (array)$m['canal']['servicio'];
19   $canales[$can_id] = [
20     'n' => count( $ms ),
21     't' => $servicio['titulo'],
22     'i' => $servicio['img_url'],
23     'u' => $servicio['url'],
24     'l' => $m['canal']['url'],
25     'm' => $ms,
26   ];
27 }
28
29 KERNEL::streamAPI( $canales, NULL, JSON_FORCE_OBJECT );
```

Listing 4.1: Endpoint `pendientes.0.php`

De manera más específica, lo que se tiene en el código 4.1, es la extracción de los parámetros de la *request*, esto en las líneas 5 a 6. Luego, en la línea 8 se realiza la consulta a la base

de datos, llamando a una instancia propia del módulo de la librería `pubsub` que mediante la función `readSub` retorna todos los mensajes pendientes asociados a una persona y notificador. Desde la línea 10 a la 13 se realiza un filtro de los mensajes, excluyendo los mensajes relacionados a canales externos, como noticias. Al tener estos mensajes filtrados, se normalizan en un diccionario ordenándolos por cada canal, para que la información consultada sea más fácil de acceder por los módulos que se conectan al *endpoint*, lo que puede verse entre las líneas 15 y 27. Finalmente, se retorna todo el diccionario en formato JSON como respuesta a la request recibida.

Continuando, en el código 4.2 se puede ver el *endpoint* de configuraciones, que permite traer las configuraciones de notificaciones asociadas a una persona.

```
1 <?php
2 require_once( '../include/pubsub.class.php' );
3
4 $pers_id = $_REQUEST['pers_id'];
5
6 $conf = PubSub::getconf( $pers_id );
7 if( ! $conf ) KERNEL::streamAPI( '0' );
8
9 KERNEL::streamAPI( $conf, NULL, JSON_FORCE_OBJECT );
```

Listing 4.2: Endpoint `configuraciones.0.php`

Más detalladamente, en la línea 4 se extrae el identificador de una persona de la *request* para poder consultar, en la línea 6, la configuración de notificaciones de la persona. En la línea 7 se valida que el diccionario venga vacío, en cuyo caso se retorna una *request* con datos vacíos y estado 200. De otro modo, se retorna el diccionario, como se ve en la línea 9.

Finalmente en el código 4.3 se tiene el *endpoint* `suscripciones`, que permite consultar todos los canales a los que está suscrita una persona en un formato específico, como móvil, e-mail o web. Este *endpoint* tiene una estructura muy similar al de configuraciones, difieren principalmente en la función que consulta a la base de datos.

```
1 <?php
2 require_once( '../include/pubsub.class.php' );
3
4 $pers_id = $_REQUEST['pers_id'];
5 $notificadores = explode( ',', $_REQUEST['nots'] );
6
7 $suscripciones = PubSub::getsub( $pers_id, $notificadores );
8 if( ! $suscripciones ) KERNEL::streamAPI( '0' );
9
10 KERNEL::streamAPI( $suscripciones, NULL, JSON_FORCE_OBJECT );
```

Listing 4.3: Endpoint `suscripciones.0.php`

En las líneas 4 y 5 se extraen los parámetros de la *request* para ser entregados, en la línea 7, a la función que retorna la información. En la línea 8 se verifica que el diccionario esté vacío, caso en el que se retorna una *request* con estado 200 y data vacía. En la línea 10 se retorna el diccionario en formato JSON en caso de que este contenga información.

4.2.2. Endpoints de escritura

En esta sección, se encuentran los *endpoints* `publicar.0.php, configurar.0.php y suscribir.0.php`. Estos se encargan de recibir información y colocarla en lista de espera para ser enviada a la base de datos.

La estructura general de estos *endpoints* se compone de cuatro secciones para el proceso de encolamiento: primero se extraen los parámetros entregados en la *request*, se crea la url que será necesario entregarle al archivo *batch* que será computado posteriormente, luego se encola esta tarea y finalmente se retorna una respuesta a la solicitud con `KERNEL::streamAPI()`.

En el código 4.4 se puede ver el *endpoint* `publicar` que permite publicar un mensaje en el canal especificado.

```
1 <?php
2 require_once( '../include/pubsub.class.php' );
3 require_once( '../include/task2.class.php' );
4 require('../web/config.php');
5
6 $pers_id = $_REQUEST['pers_id'] ? $_REQUEST['pers_id'] :  $_SESSION['kernel']['persona']['
    pers_id'];
7 $canal = $_REQUEST['can_id'];
8 $id_objeto = $_REQUEST['id_objeto'];
9 $mensaje = $_REQUEST['mensaje'];
10 if( ! $pers_id || ! $canal || ! $id_objeto || ! $mensaje ) KERNEL::streamAPI( '0' );
11
12 $datos_extra = $_REQUEST['datos_extra'] ?? '{}';
13 $fecha = $_REQUEST['fecha'] ?? '';
14 $force_push = $_REQUEST['force_push'] ?? 0;
15
16 $url = $ruta_web.'/publicar?';
17
18 $url .= http_build_query( [
19   'pers_id' => $pers_id,
20   'can_id' => $canal,
21   'id_objeto' => $id_objeto,
22   'mensaje' => $mensaje,
23   'datos_extra' => $datos_extra,
24   'fecha' => $fecha,
25   'force_push' => $force_push,
26 ] );
27
28 Task2::add( $url );
29 KERNEL::streamAPI( '1' );
```

Listing 4.4: Endpoint `publicar.0.php`

Más en detalle, en las líneas 6 a 9 se extraen los parámetros de la *request*, que son el identificador de una persona, el canal donde se publicará el mensaje, el identificador del mensaje u objeto que genera la notificación y el mensaje que se utilizará en el cuerpo de la notificación. Por ejemplo, el identificador del mensaje puede ser la llave de un mensaje en el foro y el mensaje enviado al *endpoint* `publicar` es el texto *"Alguien ha respondido este mensaje en el foro"*. En la línea 10 se valida que todos estos parámetros existan dentro de la *request*, de lo contrario se retorna un estado 200 con data vacía. Luego, en las líneas 12 a 14 se extraen parámetros opcionales en caso de existir, como lo son datos extra, fecha y opciones para las funciones que recibirán dichos datos.

Como se mencionó en la sección 3.4, los *endpoints* de escritura encolan la tareas guardando la ruta al controlador de la tarea correspondiente. Por ello, en la línea 16 se define la ruta

para el controlador de publicar. En las líneas 18 a 26 se ordenan los parámetros que serán utilizados posteriormente por el controlador para guardar los datos en la base de datos. Posteriormente, en la línea 26 se añade una nueva tarea con la ruta definida y finalmente retornamos un estado 200.

En el código 4.5 se puede observar el *endpoint* `configurar`, que permite modificar el estado de suscripción de una persona a un tipo de notificaciones.

```
1 <?php
2 require_once( '../include/pubsub.class.php' );
3
4 $pers_id = $_REQUEST['pers_id'];
5 $not_id = $_REQUEST['not_id'];
6 $estado = $_REQUEST['estado'];
7
8 PubSub::conf( $pers_id, $not_id, $estado );
9
10 KERNEL::streamAPI( '1' );
```

Listing 4.5: Endpoint `configurar.0.php`

Más específicamente, en las líneas 4 a 6 se extraen de la *request* los parámetros identificador de la persona, identificador del tipo de notificación y el estado para ser utilizados por la función que escribe en la base de datos en la línea 8. Este *endpoint* es el único de escritura que no encola la solicitud, ya que como se explicó en la sección 3.4 el uso que se le da no es crítico ni recurrente.

Finalmente, se retorna un estado 200 como respuesta en la línea 10.

En el código 4.6 se tiene el *endpoint* `suscribir`, el cual permite a un usuario cambiar el estado de suscripción a un canal y tipo de notificación especificado.

```
1 <?php
2 require_once( '../include/pubsub.class.php' );
3 require_once( '../include/task2.class.php' );
4 require('../web/config.php');
5
6 $pers_id = $_REQUEST['pers_id'] ?? $_SESSION['modulo']['pers_id'];
7 $not_id = $_REQUEST['not_id'] ? $_REQUEST['not_id'] : 'PUSH';
8
9 $can_id = $_REQUEST['can_id'];
10 $estado = (int)$_REQUEST['estado'];
11
12 $suscripciones = PubSub::getsub( $pers_id );
13 $suscripcion = (array)$suscripciones[$not_id][$can_id];
14
15 if( ! $suscripcion ) KERNEL::streamAPI( '0' );
16
17 $url = $ruta_web.'/suscribir?';
18 $url .= http_build_query( [
19   'pers_id' => $pers_id,
20   'not_id' => $not_id,
21   'can_id' => $can_id,
22   'estado' => $estado,
23 ] );
24
25 if( $_REQUEST['init'] ) $url .= '&init=1';
26
27 Task2::add( $url );
28
29 KERNEL::streamAPI( '1' );
```

Listing 4.6: Endpoint `suscribir.0.php`

En este caso, en las líneas 6 a 10 se extraen de la *request* los parámetros identificador de persona, identificador de notificación, identificador de canal y el estado de suscripción. En la línea 12 se extraen de la base de datos las suscripciones actuales de la persona y en la línea 13 se define la existencia dentro de esas suscripciones de la suscripción al canal que se quiere modificar. En la línea 14 verificamos que esa suscripción no esté vacía o que sea un objeto no nulo, ya que en caso de serlo, el usuario no debiese tener acceso al objeto consultado y por lo tanto se retorna un estado 200 con data vacía.

Si la suscripción existe, entonces definimos la ruta al controlador de suscribir y en las líneas 17 a 22 agregamos los parámetros que se utilizarán posteriormente. En la línea 24 se verifica si la *request* posee el parámetro `init`, que indicará al controlador que debe ejecutar la función de inicialización de canales al actualizar la información en la base de datos, agregando entonces el parámetro a la ruta. En la línea 26 se agrega la tarea con la ruta definida y finalmente en la línea 28 se retorna la respuesta con estado 200 al cliente.

A continuación, se presentará la implementación del encolamiento de tareas de escritura a la base de datos.

4.3. Encolamiento de solicitudes

El encolamiento de solicitudes es un concepto previo a un *buffer* para el microservicio, que responde a la necesidad de contar con un mecanismo que permita recibir solicitudes de escritura a la base de datos incluso en momentos de lentitud o caída de la misma.

Para ello, se reutilizó código relacionado a una funcionalidad ya existente en U-Cursos: las Tareas. Esta funcionalidad permite a la aplicación en general poder guardar tareas a realizar en algún momento, lo que ocurre mediante archivos de automatización, denominados `cron` o `crontab`. Por tanto, se hizo una adaptación específica para el módulo de la funcionalidad *tasks*.

4.3.1. Base de Datos

Para dar soporte a esta característica, fue necesario agregar una nueva tabla `TASKS2` a la base de datos `$UC_PUBSUB$`, cuya estructura se puede observar en la figura 4.1 y que fue explicada en la sección 3.4.

La utilización y conexión a esta tabla está a cargo de la clase `tasks2`, cuyos métodos fueron vistos con mayor detalle en la sección 3.4.

4.3.2. Clase Tasks2

Las modificaciones a la clase `tasks2` vienen dadas solo por la modificación del nombre de la base de datos a la cual se conecta la clase, cambiando a `pubsub` en la conexión para cada

	Field	Type
☐	TAS_ID	int(11)
☐	TAS_URL	varchar(255)
☐	TAS_ESTADO	int(11)
☐	TAS_FECHA_M	datetime
☐	TAS_FECHA_C	datetime

Figura 4.1: Estructura tabla `TASKS2`

uno de los métodos de la clase.

Como se comentó en la sección 4.2.2, el método `add()` de esta clase se utiliza para poder agregar las solicitudes de escritura a la base de datos a la cola, para ser gestionadas posteriormente por el controlador de tareas al ejecutar los archivos `cron`. Si bien en caso de caída de la base de datos, esta cola no está activa, permite un nivel rápido de respuesta a las *requests* recibidas.

4.3.3. Archivos Batch

Estos archivos corresponden a aquellos que serán finalmente llamados a ejecutarse a medida que el archivo `cron` avance en la ejecución de tareas. Se implementan dos de ellos, para los *endpoints* que efectivamente encolan tareas: `suscribir` y `publicar`.

Como fue mencionado anteriormente, estos archivos no difieren de gran manera respecto a los archivos de los *endpoints*, reemplazando el paso de encolamiento por una llamada a la librería `pubsub` del módulo notificaciones.

En el código 4.7 se puede observar el *batch* para suscribir un usuario a un canal especificado.

```php
1 <?php
2 require_once( '../include/pubsub.class.php' );
3 require('../web/config.php');
4
5 $rut = urldecode( $_REQUEST['pers_id'] );
6 $not_id = $_REQUEST['not_id'] ? urldecode( $_REQUEST['not_id'] ) : 'PUSH';
7 $can_id = urldecode( $_REQUEST['can_id'] );
8 $estado = (int) urldecode( $_REQUEST['estado'] );
9
10 if( ! ($pers_id || $not_id || $can_id ) ) KERNEL::streamAPI( '0' );
11
12 $suscripciones = PubSub::getsub( $rut );
13 $suscripcion = (array) $suscripciones[$not_id][$can_id];
14 if( ! $suscripcion ) KERNEL::streamAPI( '0' );
15
16 PubSub::sub( $rut, $not_id, $can_id, $estado );
17
18 if( $_REQUEST['init'] ) PubSub::initCanales();
19
20 KERNEL::streamAPI( '1' );
```

Listing 4.7: Archivo *batch* `suscribir.php`

La diferencia principal entre este código y aquel para el *endpoint* `suscribir` se encuentra en la línea 16, en donde este archivo realiza una llamada al método `sub()` de la librería `pubsub` y en la línea 18, en caso de existir el parámetro necesario, realiza además una llamada al método `initcanales()`. Las líneas anteriores solo realizan extracción de parámetros de la url y la línea posterior retorna una respuesta a la *request* que llama al archivo.

En el código 4.8 se tiene el *batch* para publicar mensajes en un canal especificado.

```
1 <?php
2 require_once( '../include/pubsub.class.php' );
3
4 $pers_id = $_REQUEST['pers_id'] ? urldecode( $_REQUEST['pers_id'] ) :  $_SESSION['kernel']['
     persona']['pers_id'];
5 $canal = urldecode($_REQUEST['can_id']);
6 $id_objeto = urldecode($_REQUEST['id_objeto']);
7 $mensaje = urldecode($_REQUEST['mensaje']);
8 if( ! ($pers_id || $canal || $id_mensaje || $mensaje) ) KERNEL::streamAPI( '0' );
9
10 $datos_extra = $_REQUEST['datos_extra'] ? UTIL::json_decode($_REQUEST['datos_extra']) : [];
11 $fecha = $_REQUEST['fecha'] ? urldecode($_REQUEST['fecha']) : '';
12 $force_push = $_REQUEST['force_push'] ? urldecode($_REQUEST['force_push']) : FALSE;
13
14 PubSub::pub( $canal, $id_objeto, $pers_id, $mensaje, $datos_extra, $fecha, $force_push );
15
16 KERNEL::streamAPI( '1' );
```

Listing 4.8: Archivo *batch* `publicar.php`

Nuevamente, la principal diferencia entre este código y el *endpoint* `publicar` se localiza en la línea 14, donde en lugar de encolar la tarea de publicar se realiza una llamada al método `pub()` de la líbrería `pubsub` para luego retornar con una respuesta exitosa a la *request* que realiza la llamada al archivo.

Por su parte, como se discutió anteriormente, el *endpoint* `configurar` no realiza encolamiento de tareas y por lo tanto no tiene un archivo *batch* correspondiente debido a que su uso no es concurrente ni crítico para el sistema.

A continuación, se hablará acerca de las llamadas asíncronas al módulo Notificaciones mediante *JQuery*.

4.4. Llamadas asíncronas con JQuery

Como se explicó en la sección 3.4, la primera aproximación de asincronía para el sistema fue la implementación de llamadas y consultas mediante *JQuery* desde *frontend* en los módulos Notificaciones y Canales.

4.4.1. Módulo Canales

Actualmente, Canales es un módulo dentro de U-Cursos que muestra los primeros 200 mensajes de un usuario, ordenados por fecha y destacando aquellos no leídos. Esta acción es precisamente una de las problemáticas explicadas en la sección 1.1, ya que es el módulo que U-Cursos muestra por defecto al ingresar a la plataforma.

Además, es en este módulo donde se configura el recibo de notificaciones para un usuario, como se explicó en la sección 3.2: Sistema de notificaciones actual.

Las modificaciones que se realizaron en este módulo están relacionadas a la extracción y actualización de las configuraciones de notificaciones para un usuario, mediante el uso de los *endpoints* `configuraciones, suscripciones, configurar y suscribir`, todo esto previa validación del estado de la base de datos mediante llamada asíncrona.

En el código 4.9 se puede observar la validación realizada mediante una llamada *JQuery* para conocer el estado de la base de datos en un archivo *javascript* que se carga al importar el template.

```
var promise = $.ajax({
    url: ruta_api + username +"/pubsub/configuraciones&pers_id="+ pers,
    success: function(response){
        var data = response
    },
    error: function(response, status, xhr){
        document.getElementById("cargando").remove()
        var error = response.responseJSON.error
        var errordiv = '<div class="mascota"><div class="info">'+ error +'</div></div>'
        document.getElementById("error").innerHTML = errordiv
    }
})

promise.then( function(){
    document.getElementById("cargando").remove()
    var url = ruta_api + username +"/mis_canales/configurar_validado"
    window.location.replace( url )
} )
```

Listing 4.9: *Script* que realiza una consulta asíncrona a la BD para el módulo Canales

Entre las líneas 1 y 12 se tiene la llamada asíncrona, donde en la línea 2 se define la url a llamar, en las líneas 3 a 4 lo que sucede si la llamada tiene éxito, que sería guardar la respuesta en la variable `data`. En caso de que la llamada retorne un error, en la línea 7 eliminamos el objeto del *HTML* que muestra el mensaje de carga de datos, en la línea 8 extraemos el mensaje de error de la respuesta del servidor, en la línea 9 definimos el código *HTML* para el objeto que contiene el error y en la línea 10 lo agregamos al DOM.

En caso de que la *request* sea exitosa, entonces redirigimos a la ventana que permite configurar las notificaciones para cada canal, en las líneas 15 a 17.

En el código 4.10 se puede observar un extracto del código del controlador de la interfaz de configuración de notificaciones, que contiene parte de lo modificado en este período de trabajo.

```
<?php
require( 'config.php' );

$notificadores = [
  'HTML' => 'Menu',
  'RSS' => 'RSS Personal',
  'DROPBOX' => 'Dropbox',
  'PUSH' => 'Push',
];

if( ! KERNEL::merged() ) {
  if( SISTEMA == 'fcfm' ) $notificadores = [ 'HTML' => 'Menu' ];
}

```

```
15 $conf = UTIL::json_decode( UTIL::wget($pubsub_url."configuraciones&pers_id=".$pers_id) );
16
17 foreach( array_keys( $conf ) as $not_id ) if( ! $notificadores[$not_id] ) unset( $conf[
     $not_id] );
18 $s_conf = implode(',', array_keys($conf));
19 $s_conf = urlencode($s_conf);
20
21 $suscripciones = UTIL::json_decode(UTIL::wget($pubsub_url."suscripciones&pers_id=".$pers_id.
     "&nots=".$s_conf));
22
23 $can_id = $_REQUEST['can_id'];
24 $not_id = $_REQUEST['notificador'];
25
26 if( $_REQUEST['accion'] == 'cambiar' ) {
27   KERNEL::log();
28   $estado = ( ! $suscripciones[$not_id][$can_id] || ! $suscripciones[$not_id][$can_id]['
     estado'] ) ? 1 : 0;
29   $parametros = [
30     'pers_id' => urlencode( $pers_id ),
31     'not_id' =>  $not_id,
32     'can_id' => urlencode( $can_id ),
33     'estado' => urlencode( $estado ),
34     'init' => 1,
35   ];
36
37   UTIL::wget($pubsub_url."suscribir", $parametros);
38   exit( '1;' );
39
40 } elseif( $_REQUEST['accion'] == 'conf' ) {
41   KERNEL::log();
42   $estado = ! $conf[$not_id];
43   $param = [
44     'pers_id' => $pers_id,
45     'not_id' => $not_id,
46     'estado' => $estadp,
47   ];
48   UTIL::wget($pubsub_url."configurar", $param);
49   exit( '1;' );
50
51 [...]
52 }
```

Listing 4.10: Extracto del controlador de configuraciones del módulo Canales

Más específicamente, entre las líneas 5 a 14 se define la variable `notificadores` que contiene los diferentes formatos de notificaciones que pueden configurarse. En la línea 16 se extraen las configuraciones de una persona desde el *endpoint* `configuraciones` en formato arreglo. En la línea 18 se eliminan del arreglo aquellos notificadores que no pertenecen a los definidos previamente, para luego en las líneas 19 y 20 extraer solo las llaves de dicho arreglo, que es utilizado en la línea 22 para extraer desde el *endpoint* `suscripciones` los datos de suscripción del usuario.

En las líneas 24 y 25 se extraen desde la *request* los identificadores de canal y notificador. Estos parámetros serán usados al existir una acción indicada en la *request*. En la línea 27 se consulta si dicha acción es "cambiar", de ser así, entre las líneas 28 y 36 se definen los parámetros que serán enviados en la línea 38 al *endpoint* `suscribir` para actualizar las suscripciones a notificaciones especificadas y en la línea 39 sale del ciclo.

En caso de que la acción de la *request* sea "conf", entre las líneas 42 y 48 se definen los parámetros que por su parte serán enviados al *endpoint* `configurar` en la línea 49, para finalmente salir del ciclo.

4.4.2. Módulo Notificaciones

Como se explicó en la sección 3.4, es el nuevo módulo de notificaciones el que se encargará de mostrar la información de los mensajes sin leer al ser consultado y utiliza el *endpoint* `pendientes` para extraer la información desde la base de datos.

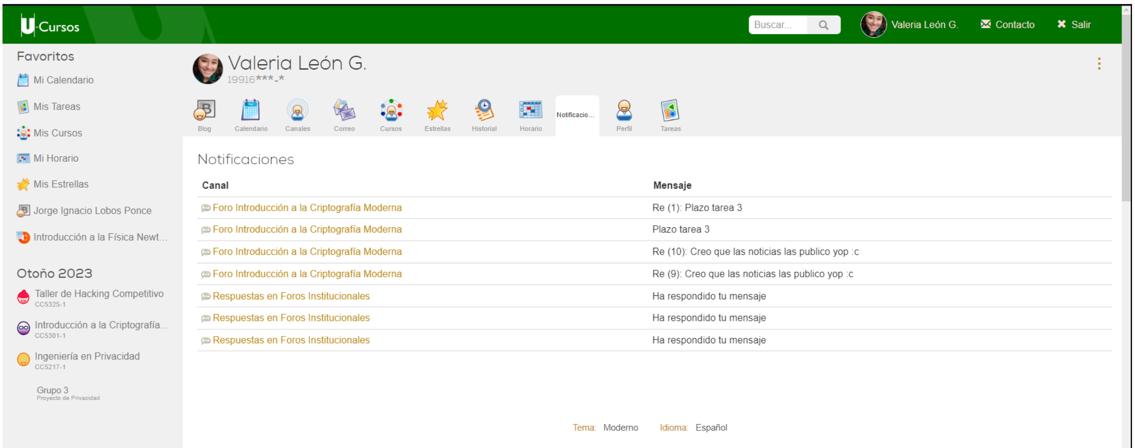


Figura 4.2: Interfaz del módulo Notificaciones

En la figura 4.2 se puede observar la interfaz del módulo Notificaciones para mostrar los mensajes pendientes del usuario, en donde se tienen notificaciones de todos los canales al que el usuario está suscrito y que poseen algún mensaje pendiente. Por ejemplo, para este usuario se tienen respuestas a mensajes en el foro institucional y en el foro del curso Introducción a la Criptografía Moderna. Al hacer click se redirige al usuario al canal correspondiente a la notificación.

Esta interfaz recibe los datos desde el *backend*, que a su vez consulta el *endpoint* `pendientes` para enviarlos al *template* de manera asíncrona mediante *JQuery*.

En el código 4.11 se puede observar el funcionamiento del controlador de la interfaz del módulo.

```
1 <?php
2 require( 'config.php' );
3
4 $username = $_SESSION['kernel']['persona']['username'];
5 $pers_id = $_SESSION['kernel']['persona']['pers_id'];
6
7 KERNEL::head();
8 require( template( '../template/index.html' ) );
9 KERNEL::foot();
```

Listing 4.11: *Backend* index módulo Notificaciones

Más en detalle, en las líneas 4 y 5 se definen variables para consultar la API de pendientes, lo que se realiza en el *frontend* mediante una llamada asíncrona. En la línea 7 se realiza la llamada para importar los menús principal y lateral de la plataforma, en la línea 8 importamos el *template* `index.html` para cargar el contenido del módulo y en la línea 9 se importan las opciones del pie de página.

En el código 4.12 se puede observar el *script* que realiza la validación mediante *JQuery* y extrae los datos del *endpoint* `pendientes` al importar el template desde el *backend*.

```
1 function pendientes( username, pers ) {
2     var url = ruta_api + username + "/pubsub/pendientes&pers_id=" + pers
3     var error_no_pendientes = '<div class="mascota"><div class="info">No hay mensajes en el
      historial</div></div>'
4
5     $.ajax( url, {
6         success: function(response){
7             var len = Object.keys(response).length
8             var tabla = document.getElementById("notificaciones")
9             var cargando =  document.getElementById("cargando")
10
11            if(len == 0 ){
12                tabla.remove()
13                cargando.remove()
14                document.getElementById("error").innerHTML = error_no_pendientes
15            }else{
16                cargando.remove()
17                for(var canal in response){ //cada canal
18                    var conteo = response[canal].n
19                    var titulo = response[canal].t
20                    var imagen = response[canal].i
21                    var url = response[canal].u
22                    var contenido = response[canal].m
23
24                    for(var mensaje in contenido ){ //cada mensaje
25                        var row = tabla.insertRow()
26                        var col1 = row.insertCell(0)
27                        var col2 = row.insertCell(1)
28
29                        contenidoMensaje = contenido[mensaje].mensaje
30                        var img = '<img src="'+imagen+'" alt="'+titulo+'" class="icono"/>'
31                        var a = '<a href="'+url+'">'+titulo+'</a>'
32                        col1.innerHTML=  img + a
33                        col2.innerHTML=  contenidoMensaje
34                    }
35
36                }
37            }
38        },
39        error: function(response, status, xhr){
40            var tabla = document.getElementById("notificaciones")
41            var cargando =  document.getElementById("cargando")
42            var errordiv = ''
43            var error = response.responseJSON.error
44            tabla.remove()
45            cargando.remove()
46            errordiv = '<div class="mascota"><div class="info">'+ error +'</div></div>'
47            document.getElementById("error").innerHTML = errordiv
48        }
49    })
50
51
52 }
53
54 pendientes( username, pers )
```

Listing 4.12: *Script* que valida y carga datos para el módulo Notificaciones

Entrando en detalles, se define una nueva función `pendientes` que recibe identificadores de usuario. En las líneas 2 y 3 se definen la ruta a consultar y un string de error. En la línea 5 se realiza la consulta con *ajax*. El caso de éxito en la consulta se define entre las líneas 6 y 33. Si la consulta retorna sin mensajes pendientes, entonces se despliega el mensaje *"No hay mensajes en el historial"* junto a la mascota en la interfaz del módulo, lo que sucede entre las líneas 11 y 14. Por su parte, en caso de existir información dentro de la respuesta, se extraen

todos los mensajes junto a su información y se despliegan en una tabla en la interfaz, lo que se realiza entre las líneas 15 a 33.

Entre las líneas 39 y 47 se tiene el comportamiento en caso de que exista un error en la interfaz, que sería limpiar los elementos del *HTML* de carga y agregar una sección con el error enviado desde la API.

Finalmente, en la línea 54 se llama a esta función con los parámetros entregados en un *script* embebido en *HTML* al ser compilada la página en el servidor para que la consulta se realice al momento de carga de la página desde el lado del cliente.

4.5. Conexión a los endpoints

Como se comentó en secciones anteriores, la conexión a los *endpoints* anteriormente descritos puede realizarse de dos formas: mediante llamadas asíncronas por *JQuery* o llamadas que entran a ser encoladas.

Ambos tipos llamadas corresponden a consultas web a una url, y como se explicó en la sección 3.1.2, esta dirección posee una estructura particular. Para efectos de mostrar la forma de conexión para cada caso, la url puede obviarse y asumirse conocida.

4.5.1. Endpoints de lectura

Las conexiones a un *endpoint* de lectura están diseñadas para ser realizadas por medio de *JQuery* desde el *frontend*, más específicamente, se recomienda utilizar la estructura provista por el método *Ajax*. En el código 4.12 de la sección anterior se puede observar un ejemplo de una conexión de este estilo.

En el código 4.13 se puede observar el formato de utilización *Ajax* para una llamada a un *endpoint* de lectura en alto, la cual se recomienda emplear dentro de una función de *JavaScript*.

```
1 function conexion( params ) {
2     var url = params.url
3 
4     $.ajax( url, {
5         success: function(response){
6             //Aca se maneja lo que sucede con la respuesta en caso de que la consulta sea
    exitosa
7         },
8         error: function(response, status){
9             //Aca se maneja la logica en caso de recibir un error como respuesta
10         }
11     })
12 }
```

Listing 4.13: Estándar en alto nivel de conexión a *endpoint* de lectura

Más en detalle, es posible definir una función que reciba ciertos parámetros, entre ellos la url a consultar. Luego se crea la llamada con *Ajax*, lo que se puede ver en la línea 4, donde

entregamos la url como primer parámetro y un diccionario con los casos de éxito y error en la consulta como segundo argumento. Cada uno de estos puede ejecutar una función, como se tiene en las líneas 5 y 8. En caso de éxito, se recibe un parámetro *response*, que contiene la data de la consulta realizada. Por su parte, en caso de error también se recibe una *response* con data, además del estado de la consulta *status* que puede ser uno de los valores mencionados en 3.1.2. Cabe destacar que *Ajax* soporta otros argumentos, pero se puede seleccionar aquellos de interés para el manejo que se le quiera dar a cada caso.

Lo que ocurra dentro de cada función queda a criterio de las necesidades del módulo que realiza la consulta. Sin embargo, la ventaja de utilizar este tipo de llamadas es que el resto de recursos dentro de la plataforma sigue disponible mientras la consulta se realiza de manera asíncrona.

4.5.2. Endpoints de escritura

Para el caso de los *endpoints* de escritura, su conexión está pensada para realizarse desde el *backend* de la plataforma, por lo que debiese estar en los archivos *PHP* del controlador del módulo que realiza la consulta.

En el código 4.14 se puede observar a un alto nivel el estándar de conexión a un *endpoint* de escritura desde el *backend* de algún módulo.

```
1  ...
2  $parametros = [
3          ...
4    'pers_id' => $pers_id,
5    'not_id' =>  $not_id,
6    'can_id' => $can_id ,
7          ...
8  ];
9
10 $response = UTIL::wget($url, $parametros);
11 ...
```

Listing 4.14: Estándar en alto nivel de conexión a *endpoint* de escritura

Los puntos suspensivos indican que puede haber código antes o después de las líneas explicitadas. Se recomienda definir un *array* de *PHP* con los parámetros que se le entregarán al *endpoint* a consultar, esto entre las líneas 2 y 8, para luego realizar la llamada mediante el método `UTIL::wget()` perteneciente al `KERNEL`, en la línea 10. Este método es el encargado de ejecutar la consulta y recibe como parámetros la url al *endpoint* y los argumentos de la consulta. La respuesta puede ser guardada en una variable para ser utilizada o simplemente ejecutar el método sin asignarlo, para proseguir con el resto del código en el archivo.

Para efectos de este trabajo, se realizaron las conexiones parciales de los módulos Foro y Canales como parte de los casos de uso de los *endpoints* implementados.

A continuación, se presentarán las diferentes validaciones realizadas al módulo para evaluar su desempeño y el cumplimiento de objetivos.

Capítulo 5

Validaciones

Para comprobar el cumplimiento de los objetivos establecidos en la sección 1.2, se realizó en primer lugar una instancia de revisión de código, que será explicada en la sección 5.1, y posteriormente algunas pruebas de funcionalidad, carga y robustez, lo que será profundizado en las secciones 5.2 y 5.3.

5.1. Revisión de Código

En Ucampus, una revisión de código corresponde a una instancia en la que analiza todo el código lógico que compone, en este caso, el módulo, con la intención de ajustarlo a los estándares manejados por el Centro, asegurando que posea documentación, que las metodologías utilizadas sean óptimas y la sintaxis utilizada sea representativa del código en caso de ser necesario.

La revisión de código para el módulo Notificaciones fue realizada el día 11 de mayo de 2023, participó el profesor co-guía Willy Maikowski, quien era parte del equipo de desarrollo de Ucampus al momento de realizar este trabajo de memoria, y abarcó la implementación de los 6 *endpoints* y el archivo `index.php`, correspondiente a la interfaz gráfica del módulo.

Para cada uno de los archivos revisados se añadió la documentación correspondiente, además de analizar la lógica de cada uno y mapear la funcionalidad a la que apunta cumplir el *endpoint*, asegurando que los *endpoints* de escritura cumplieran con encolar las tareas y los de lectura retornaran la información adecuadamente.

La funcionalidad de encolamiento de tareas ya se encontraba implementada a la fecha, pero los archivos correspondientes no fueron incluidos en la revisión de código debido a que su implementación consistió en replicar código ya existente en el módulo `KERNEL`, por tanto su lógica no difiere del estándar.

En el caso de los archivos *JavaScript* relativos a consultas asíncronas, tampoco fueron incluidos en la revisión de código ya que fueron implementados de forma posterior a la instancia, por lo que no se encuentran validados bajos los estándares de sintaxis de Ucampus.

5.2. Pruebas de funcionalidad

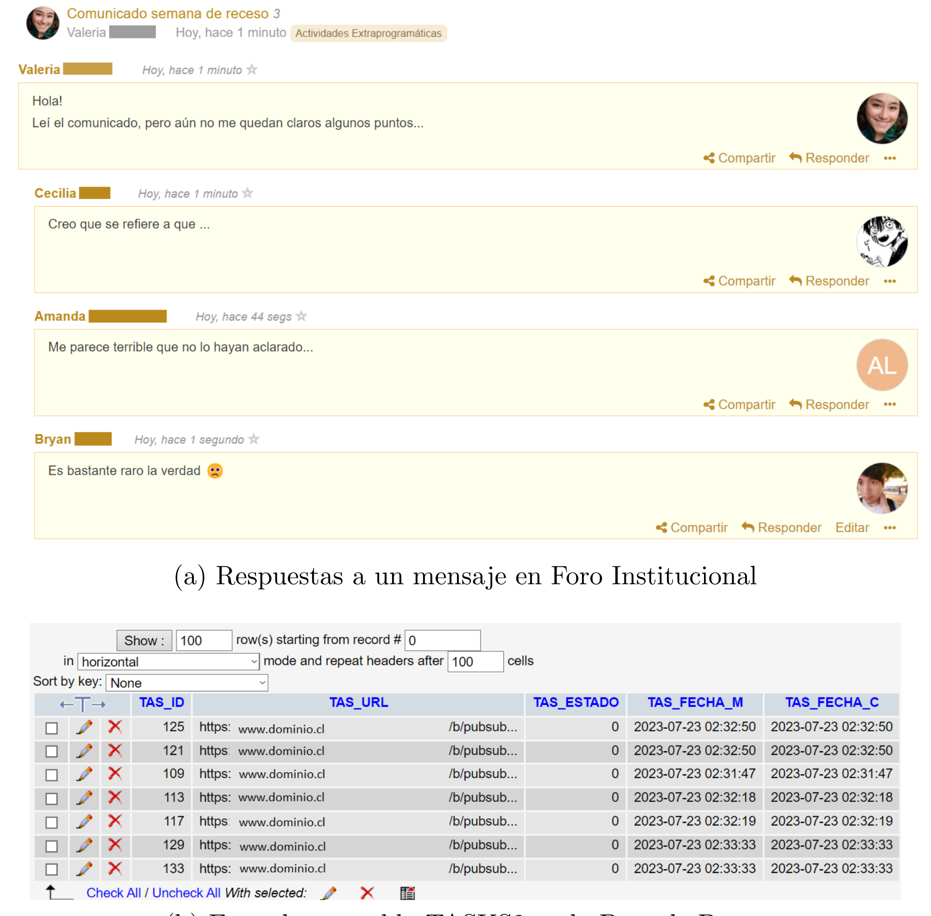
Las pruebas de funcionalidad están orientadas a establecer si el diseño planteado cumple con lo propuesto una vez implementado, lo que permite detectar situaciones que el diseño pasa por alto o si lo implementado funciona como se propuso.

Para el módulo Notificaciones, en la sección 1.2 se establecen los objetivos específicos 4 y 5, relacionados al diseño e implementación del proceso de conexión entre los módulos actuales y el nuevo servicio. Este proceso de conexión consta de dos tipos, como fue explicado en las secciones anteriores; encolamiento de tareas para las solicitudes de escritura y llamadas mediante *JQuery* para las consultas de lectura a la base de datos.

Para validar la funcionalidad del encolamiento de tareas del módulo, se generan notificaciones a un usuario a partir de respuestas a mensajes en el módulo Foro Institucional, lo que debiese generar un encolamiento de tareas que se traduce en nuevas entradas a la base de datos en la tabla `TASKS2` y luego de su ejecución, produce nuevos mensajes pendientes al consultar el módulo Notificaciones.

En la figura 5.1 se puede observar el proceso descrito en el párrafo anterior. En la subfigura 5.1a se tiene un hilo en el Foro Institucional, con varias respuestas al mensaje inicial. Cada uno de estos mensajes genera dos notificaciones: una notificación específica a la persona autora del mensaje que se está respondiendo y una notificación para todas las personas que están suscritas al canal. Luego, en la subfigura 5.1b se observan las tareas encoladas en la base de datos luego de que estos mensajes fueron creados. En este caso, son 7 tareas por lo mencionado anteriormente: 3 notificaciones para el usuario autor del hilo y 4 notificaciones de nuevos mensajes en el Foro Institucional.

Finalmente, en la subfigura 5.1c se tienen las notificaciones del autor del hilo al consultar su módulo Notificaciones, donde se observan las 7 notificaciones correspondientes.



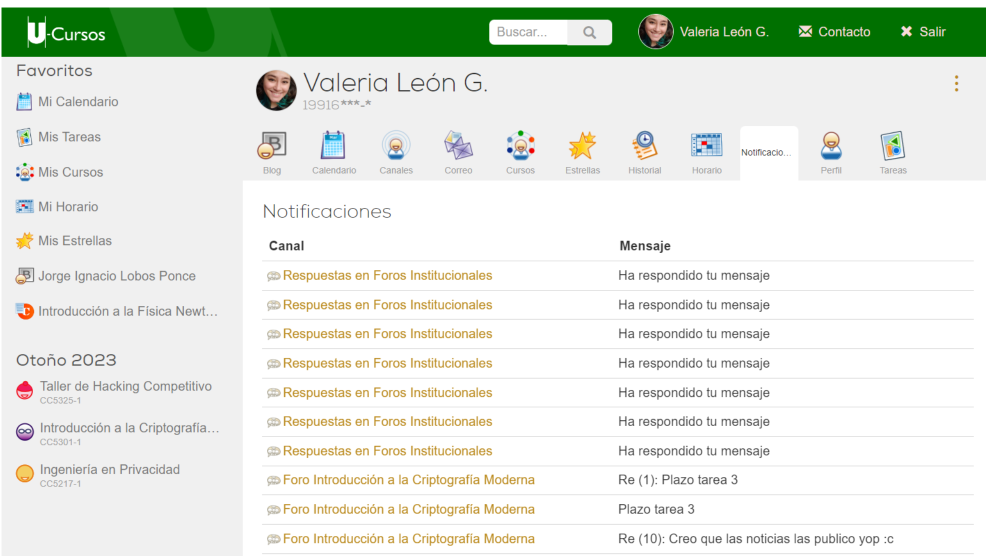
(a) Respuestas a un mensaje en Foro Institucional

Show : 100 row(s) starting from record # 0
in horizontal mode and repeat headers after 100 cells
Sort by key: None

	TAS_ID	TAS_URL	TAS_ESTADO	TAS_FECHA_M	TAS_FECHA_C
	125	https: www.dominio.cl /b/pubsub...	0	2023-07-23 02:32:50	2023-07-23 02:32:50
	121	https www.dominio.cl /b/pubsub...	0	2023-07-23 02:32:50	2023-07-23 02:32:50
	109	https: www.dominio.cl /b/pubsub...	0	2023-07-23 02:31:47	2023-07-23 02:31:47
	113	https: www.dominio.cl /b/pubsub...	0	2023-07-23 02:32:18	2023-07-23 02:32:18
	117	https www.dominio.cl /b/pubsub...	0	2023-07-23 02:32:19	2023-07-23 02:32:19
	129	https: www.dominio.cl /b/pubsub...	0	2023-07-23 02:33:33	2023-07-23 02:33:33
	133	https: www.dominio.cl /b/pubsub...	0	2023-07-23 02:33:33	2023-07-23 02:33:33

Check All / Uncheck All *With selected:*

(b) Entradas en tabla TASKS2 en la Base de Datos



(c) Notificaciones del usuario al consultar el módulo Notificaciones

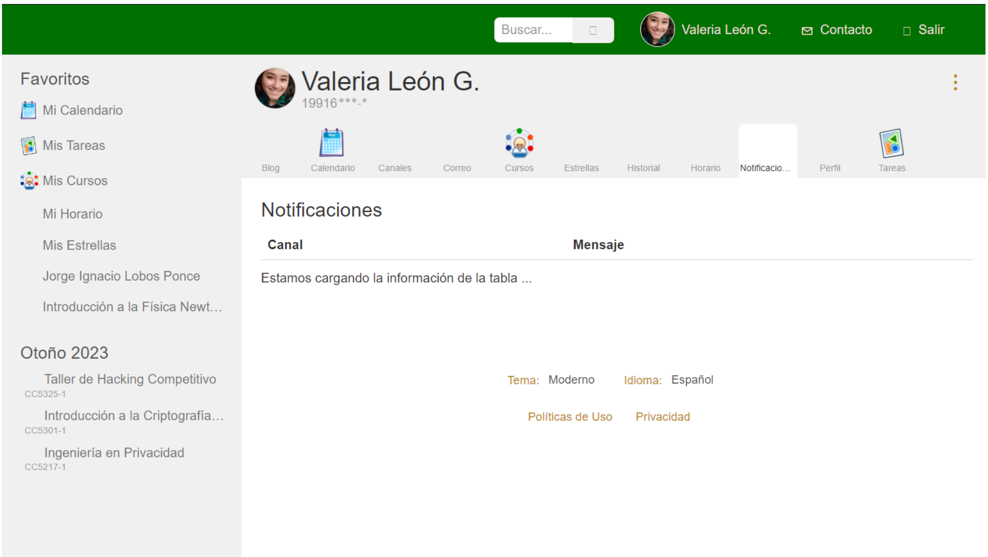
Figura 5.1: Encolamiento de tareas al publicar un mensaje en Foro Institucional

Para validar la conexión al módulo de manera asíncrona mediante *JQuery*, se ingresa al módulo con una conexión lenta, configuración que puede realizarse desde las opciones de desarrollador del navegador utilizado, en el apartado Red. Esto permite poder observar los

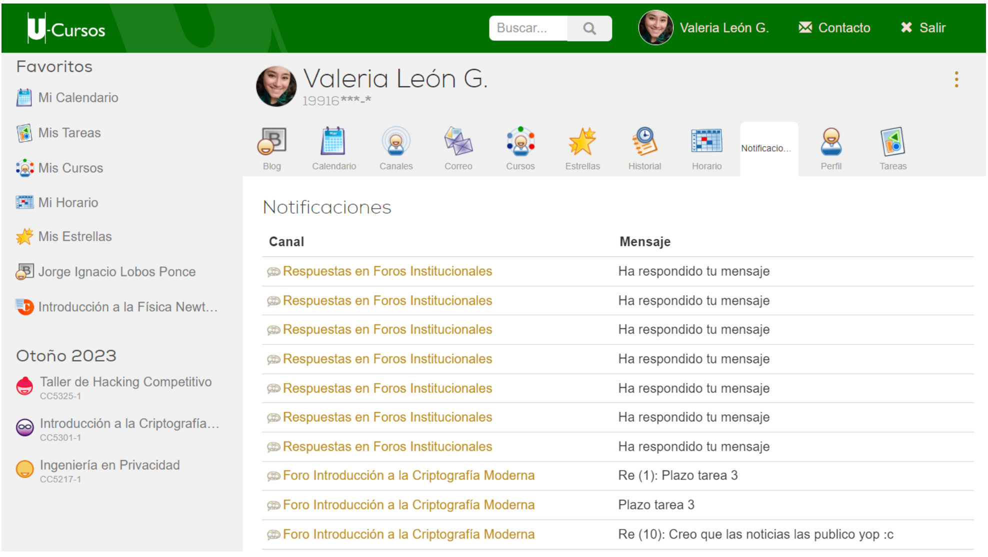
diferentes estados de carga de una página con mayor detalle y también observar las consultas que se están realizando para obtener la información necesaria.

En la figura 5.2, se tiene el proceso de carga del módulo Notificaciones. Al ingresar, previo a que se realice la consulta al *endpoint* `pendientes`, se tiene lo mostrado en la subfigura 5.2a, en donde se observa el mensaje *"Estamos cargando la información de la tabla..."*. Se puede notar que falta cargar otros recursos de la página, como los iconos del resto de los módulos, el logo de U-Cursos y las fuentes de los textos.

Luego de que la consulta es completada, la información se procesa y ordena en una tabla, como puede observarse en la subfigura 5.2b, además del resto de los recursos que se cargan de forma similar.



(a) Mensaje de carga desplegado al ingresar al módulo Notificaciones



(b) Despliegue de información en el módulo Notificaciones

Figura 5.2: Encolamiento de tareas al publicar un mensaje en Foro Institucional

Para el caso de los objetivos relativos al diseño, se puede decir que fueron cumplidos, ya que se tiene un producto mínimo viable capaz de administrar las notificaciones de la plataforma de manera independiente al módulo `KERNEL`, sin impactar de forma negativa al resto de los servicios más allá de la no disponibilidad de las notificaciones a los usuarios, con las tecnologías apropiadas para el alcance de tiempo del proyecto.

Además, también se especifican las formas de conexión al nuevo servicio, cumpliendo con diseñar un proceso de migración entre las plataformas actuales y el nuevo servicio.

5.3. Pruebas de carga y robustez

Las pruebas de carga y robustez son utilizadas para medir o comprobar el rendimiento de un sistema bajo ciertas condiciones, lo que permite anticipar medidas de acción, realizar optimizaciones para ajustar el rendimiento u observar el comportamiento general del mismo.

Como se propuso en la sección 1.2, el objetivo específico número 6 tiene relación con validar el rendimiento de las plataformas en sus condiciones de uso actuales con un flujo aproximado de 7,000 usuarios simultáneos. Esto con la finalidad de asegurar que el servicio de notificaciones no representara problemas para el resto de las funcionalidades de la plataforma como lo hace actualmente.

Debido a que durante el período de desarrollo de este trabajo de memoria, el Centro Tecnológico Ucampus tuvo participación en el proceso de desarrollo y mantención de la plataforma de Secretaría de Participación Ciudadana[1] para este segundo proceso constitucional, no era posible la realización de pruebas de cargas que representaran una falta de disponibilidad del ambiente de desarrollo durante ciertos períodos de tiempo.

Por lo anterior, es que se decide entonces validar la disponibilidad de la plataforma cuando el sistema de notificaciones no está disponible, ya sea por error en la consulta realizada o por una caída en la base de datos. Esto también nos permite validar el objetivo específico número 3, relacionado a la independencia del nuevo módulo del resto de la plataforma.

Para ello se agregó en cada *endpoint* una línea al principio de cada archivo que retornara un error al compilar, que es lo que internamente sucede si una conexión a módulo no está disponible. Es decir, al tratar de conectarse a un *endpoint* lo que devuelve el servidor es un error, que es lo que sucederá cuando efectivamente la base de datos esté caída o no pueda dar respuesta a las consultas generadas.

En la figura 5.3 se puede observar el mensaje de error obtenido al consultar el módulo de Notificaciones desde U-Cursos, desde la interfaz en la aplicación. Por su parte, en la figura 5.4 se tiene la respuesta del *endpoint* `pendientes` al ser consultado directamente desde el navegador cuando la base de datos está caída.

[1] Sitio Web: `https://ucampus.quieroparticipar.cl/m/iniciativas/iniciativas`, consultado el 15/07/2023

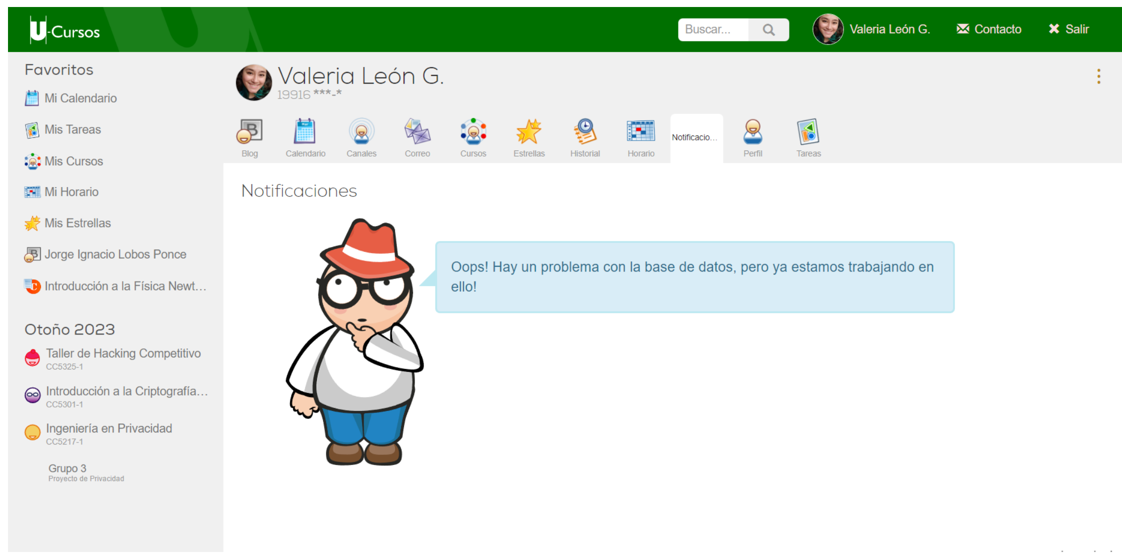


Figura 5.3: Interfaz módulo Notificaciones con base de datos no disponible

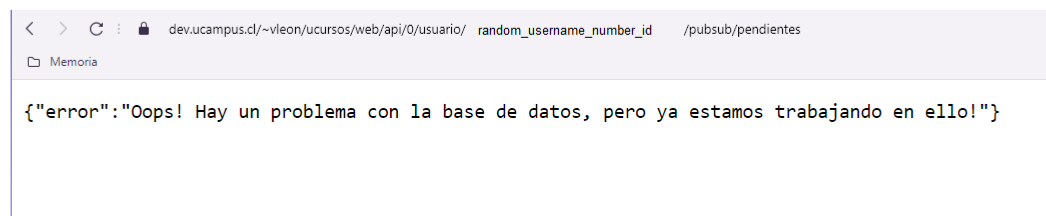


Figura 5.4: Endpoint `pendientes` al ser consultado desde el navegador con base de datos no disponible

Lo anterior fue realizado para cada uno de los *endpoints*, asegurando que aquellos que poseen una conexión al módulo Notificaciones mediante algún *endpoint*, como el módulo Canales y Foro, siguiesen disponibles aún sin generar notificaciones o que mostrasen el error correspondiente. En la figura 5.3 se puede observar el mensaje de error al ingresar a la opción Configurar del Módulo Canales cuando la base de datos de notificaciones está caída y en la figura 5.4 se tiene lo que se despliega al consultar directamente el *endpoint* desde un navegador.

Dadas estas pruebas, fue posible comprobar el cumplimiento parcial del objetivo específico número 6, ya que la plataforma sigue disponible incluso cuando no hay servicio de notificaciones disponible, con lo que un usuario puede navegar entre módulos, revisar el resto de la información en U-Cursos y hacer uso de sus funcionalidades aunque no sea notificado al momento de cambios que se hayan realizados en la plataforma.

Capítulo 6

Conclusión

Un sistema de notificaciones es una parte importante en una aplicación como U-Cursos, donde las interacciones entre los usuarios que conforman la comunidad de la Universidad son recurrentes y relevantes para la misma, en un contexto educativo. Es por ello, que este sistema no puede ni debería interferir de manera significativa en el desempeño del resto de los componentes de la plataforma, principalmente en el funcionamiento de su controlador principal.

A partir de las secciones revisadas a lo largo de este informe, se pueden definir el cumplimiento de objetivos en la sección 6.2, las conclusiones respecto al trabajo realizado en la sección 6.1, las ideas de mejoras al sistema en la sección 6.3 y finalmente aquellas relacionadas al aprendizaje personal en la sección 6.4.

6.1. Trabajo realizado

Durante este período de trabajo, se logró diseñar e implementar un nuevo módulo de Notificaciones independiente al controlador principal de la plataforma U-cursos. Las principales características de este módulo son sus *endpoints* para manejo de información de notificaciones y el encolamiento de solicitudes de escritura a la base de datos.

Primeramente, para poder realizar un correcto diseño e implementación de un nuevo módulo de Notificaciones, fue necesario, en primera instancia, realizar un estudio a profundidad de la plataforma, identificando los puntos en el código fuente en el que se hace uso de la librería `PubSub` y dejándolos sin efecto sobre la ejecución, incluyendo el módulo `KERNEL`. Dichos puntos se encuentran documentados en el anexo A.1 y permitieron establecer el alcance e importancia de las funcionalidades que actualmente provee la librería, lo que influyó en el posterior diseño del módulo implementado.

Pasando a la etapa de diseño y desarrollo, los diferentes *endpoints* implementados, descritos en las secciones 3 y 4, permiten añadir y leer notificaciones cuando sea requerido, además de cambiar las configuraciones de suscripción de los diferentes usuarios y canales. Los *endpoints* fueron desarrollados con el estándar de Ucampus para API's, por lo que su estructura

es conocida para el equipo de desarrollo.

Por su parte, también se diseñaron modos de conexión para los dos tipos de *endpoints* que se poseen, lectura y escritura. Se estableció una estructura para conexión a los *endpoints* de lectura mediante solicitudes *Ajax* desde el *frontend* de la plataforma, donde se maneja el despliegue de información. A su vez, para los *endpoints* de escritura a la base de datos, se establece una conexión mediante consulta web desde el *backend* con *PHP*, las cuales se encolan por orden de llegada y permiten al controlador principal no bloquear su ejecución esperando a que la solicitud completa se ingrese a la base de datos. En su lugar, el encolamiento permite recibir las solicitudes y dejar que el módulo de Notificaciones se encargue de escribir en la base de datos mediante archivos *cron*, ejecutados periódicamente.

De las conexiones documentadas, solo se llevaron a cabo alrededor del 11 % del total, que corresponden a conexiones realizadas en los módulos Canales y Foro, con el propósito de validar funcionalidad y robustez del sistema con el nuevo módulo implementado.

Por la naturaleza de este proyecto, no fueron necesarios cambios o diseño de interfaces de usuario, sin embargo se crea una página de inicio para el módulo Notificaciones desde donde un usuario puede consultar sus notificaciones pendientes en formato tabla y acceder a cada una de ellas en su módulo correspondiente. Lo anterior también fue implementado siguiendo el estándar de Ucampus en cuanto a lenguajes y estructuras.

Cabe destacar que este trabajo es el primero en realizar una manipulación del código del controlador principal, en términos de funcionalidades que este abarca. Esto es un riesgo en sí debido a que, como se explicó a lo largo de este informe, es el `KERNEL` quién manipula la información de permisos, flujo entre módulos, acceso a datos de sesión, entre otras responsabilidades, por lo que cualquier cambio se debe trazar para no generar afectaciones en la plataforma.

Concluyendo, este trabajo es un buen punto de partida para mejorar el servicio de notificaciones de la plataforma U-Cursos, que permitirá tener una plataforma más estable, teniendo en cuenta el aumento sostenido de usuarios que se ha presentado a lo largo de los últimos años debido al aumento de estudiantes en la universidad. Además, su mantención y extensión queda a cargo del equipo de desarrollo de Ucampus, lo que se consideró al momento de tomar las decisiones de diseño del sistema.

6.2. Cumplimiento de objetivos

De acuerdo a todo lo expuesto a lo largo de este informe, se puede realizar un análisis del cumplimiento de los objetivos específicos propuestos en la sección 1.2.2.

A partir del trabajo realizado, es posible decir que el objetivo general se encuentra logrado, siendo este "Diseñar e implementar un sistema de notificaciones para las plataformas de Ucampus, que esté separado de la ejecución principal de las mismas, de manera que permita integridad y disponibilidad en tiempos de alta demanda".

Para el objetivo específico número 1, es posible decir que se encuentra cumplido a caba-

lidad, ya que se encontraron todos los usos de la librería `PubSub` en la instalación disponible para el desarrollo del trabajo propuesto, que incluía todos los módulos disponibles para la plataforma, además del `KERNEL` y otros módulos necesarios para el funcionamiento de la plataforma. Estos usos se encuentran listados en el anexo A.1, destacando en qué línea del archivo correspondiente es posible ubicarlo y qué función del la librería se utiliza.

El objetivo número 2 también fue cumplido satisfactoriamente, ya que se establecieron cuáles eran las tecnologías disponibles que mejor se adaptaban al propósito del nuevo sistema de notificaciones y su posterior uso y mantenimiento. Fue necesario considerar y analizar cómo herramientas externas a las ya utilizadas en Ucampus podían ayudar a que el nuevo módulo funcionara con las características deseadas, contrastando cada una ellas respecto a criterios de costo, funcionalidad, mantención y extensibillidad. Esto para posteriormente decidir la opción que mejor se adaptaba a la solución y al tiempo disponible para el desarrollo del proyecto.

Para el objetivo número 3, efectivamente se rediseñó la funcionalidad de notificaciones de forma independiente al resto de los servicios, esto como un nuevo módulo dentro de la plataforma. Este nuevo módulo se diseñó como una microservicio, lo que fue explicado en el capítulo 3, con sus propios *endpoints* para lectura y escritura de información en la base de datos. Además de añadir al diseño una cola de encolamiento de solicitudes de escritura a la base de datos, ya que estas solicitudes debiesen manejarse desde el *backend* de la plataforma por seguridad de la información. Por lo tanto, puede considerarse este objetivo como logrado.

Por su parte, el objetivo específico número 4, que se refiere al diseño proceso de migración entre la plataforma y el nuevo microservicio, puede considerarse cumplido, ya que se especifican los modos de conexión entre el nuevo módulo y el resto de servicios de la plataforma, detallados en la sección 4.5, diferenciados por tipo de *endpoint* a ser utilizado.

En el caso del objetivo número 5 se puede considerar parcialmente logrado, esto debido a que se implementa un módulo de notificaciones independiente al resto de los módulos de la plataforma, especialmente del `KERNEL` de U-cursos, que corresponde a la implementación del objetivo específico número 3. Sin embargo, si bien se diseñó y ejemplificó el proceso de conexión para cada tipo de *endpoint*, las conexiones listadas no fueron realizadas en su totalidad. Aproximadamente un 11 % del total (10 de 94) fueron implementadas para efectos de validar la funcionalidad y robustez del nuevo módulo. Por ello, la implementación completa del proceso de migración se considera incompleta.

Finalmente, el objetivo específico número 6 se considera logrado parcialmente, ya que no se pudieron realizar pruebas de carga para las condiciones de uso indicadas, debido a que los servidores de desarrollo no podían verse imposibilitados o caídos durante la realización de este proyecto. Lo anterior se debía a que el Centro Tecnológico Ucampus era parte del desarrollo de dos mecanismos de participación ciudadana para el nuevo proceso constituyente, Audiencias Públicas[1] e Iniciativa Popular de Norma[2]. Sin embargo, se realizaron validaciones de funcionalidad y rendimiento simulando una caída del sistema de notificaciones, lo que resultó en un flujo constante entre los diferentes módulos, con acceso al resto de la información

[1] Sitio Web: `https://audiencias.scanner.unholster.com/regiones/inicio`, consultado el 01/09/2023.

[2] Sitio Web: `https://ucampus.quieroparticipar.cl/m/iniciativas/iniciativas`, consultado el 01/09/2023.

y servicios pero sin notificaciones disponibles.

6.3. Mejoras a futuro

Dentro de las principales mejoras a futuro propuestas para este trabajo, destacan la conexión con el resto de los módulos y el encolamiento de tareas.

Si bien se diseñó un proceso de conexión entre el resto de los módulos y las notificaciones de acuerdo a los tipos de solicitudes que se realizan a la librería `pubsub`, dichas conexiones no fueron realizadas en su totalidad. Como se explicó en la sección 5.3, las conexiones realizadas parcialmente corresponden a los módulos Canales y Foro, que representan aproximadamente el 11 % del total, en los lugares que permitían comprobar los casos de usos principales de los *endpoints* a consultar. No obstante, se documentaron al menos otras 90 conexiones en diferentes módulos, incluido el `KERNEL` de U-Cursos.

Al momento de realizar el proceso de conexión de cada uso de la librería, es necesario estudiar cada uno de ellos en cuanto a cómo se extraen los datos y cómo se procesan para ser utilizados, ya que las funcionalidades relacionadas debiesen mantenerse. Además, se debe configurar la dirección de la solicitud al *endpoint* correspondiente. En caso de que se realicen *requests* de lectura desde el *backend*, es necesario generar la estructura necesaria para migrar la solicitud al *frontend*. Este proceso al completo puede incluso tomar el tiempo requerido para un trabajo de memoria diferente.

Además, se propone una validación más exhaustiva del módulo en condiciones de uso de alta demanda, para poder determinar de mejor forma la robustez del sistema y determinar cambios o mejoras necesarias a su estructura.

Para el encolamiento de tareas, se sugiere que al momento de implantar los cambios realizados, se separen las bases de datos de notificaciones y tareas del módulo, ya que lo ideal es que el encolamiento funcione incluso si las notificaciones no están disponibles, para poder mantener solicitudes que deban ser ingresadas cuando el servicio de notificaciones vuelva a su funcionamiento.

Otras mejoras propuestas van en línea a respaldar la seguridad de las llamadas a los diferentes *endpoints*, ya que si bien U-Cursos no permite realizar consultas a un *endpoint* si no se está autenticado en la plataforma, se debe asegurar la autorización de un usuario para realizar solicitudes y no permitir que un usuario pueda obtener información de otro.

Finalmente, se propone un estudio más extensivo de los usos actuales de la librería `pubsub` para poder migrar las responsabilidades de notificaciones al nuevo módulo, como el envío de mensajes por los diferentes notificadores. Esto en pos de independizar y centralizar todo lo relacionado a notificaciones en un solo módulo, ya que, por ejemplo, el envío de notificaciones vía correo lo maneja un módulo diferente al nuevo módulo implementado en este trabajo.

6.4. Aprendizaje personal

En última instancia, se pasará a discutir y reflexionar de manera personal acerca del trabajo plasmado en el presente informe.

De manera más técnica, el proyecto realizado le permitió a la estudiante mejorar las habilidades de producción de código que se adapte a un estándar previamente establecido por el cliente, lo que permite una mejor comprensión futura, extensibilidad y mantenimiento que se otorgue a la plataforma. Se destaca la aplicación de conocimientos adquiridos sobre análisis y estudio de opciones para el diseño de un sistema de software que cumpla con los requisitos y objetivos establecidos para su funcionamiento, decisiones que pueden ser validadas posteriormente con el resto del equipo de desarrollo, adaptando la solución a las necesidades del problema.

Se reconoce la importancia de una planificación del trabajo para el cumplimiento de objetivos, pero también de poder adaptar una hoja de ruta a medida que nuevas ideas surgen al estudiar las posibles soluciones. Es necesario investigar y analizar diferentes alternativas para tomar decisiones informadas, que pueden incluir la modificación de la planificación previa, sobretodo en un proyecto con tiempo acotado como lo fue este trabajo.

Siguiendo esta línea, no se puede restar importancia al estudio de los recursos actuales que se poseen y de la plataforma a intervenir, pues permiten delimitar de mejor manera el tiempo y esfuerzos requeridos para llevar a cabo la solución que se propone. Por lo tanto, el análisis, diseño e implementación de un sistema poseen la misma importancia a lo largo del proceso para que este pueda materializarse. Y si bien a veces se desea realizar más durante el plazo de trabajo establecido, es necesario visualizar cuándo dichos esfuerzos escapan de los recursos que se poseen.

Esta experiencia también le permitió a la estudiante consolidar variados conocimientos adquiridos a lo largo de estos años de estudio, pudiendo asumir un proyecto con una problemática real pero cuya propuesta era abstracta y concretarlo en un solución que sirve como punto de partida a un sistema más definitivo.

Es por ello que se considera el objetivo principal logrado, ya que se materializó un nuevo sistema de Notificaciones independiente para la plataforma, con el fin de lograr un servicio más estable para todos sus usuarios.

Bibliografía

[1] Github - spatie/async: Easily run code asynchronously. Disponible en `https://github.com/spatie/async`. Revisado el 2023/07/24. Última versión: 2022/06/21.

[2] Revolt - the rock-solid event loop for php — revolt php. Disponible en `https://revolt.run/`. Revisado el 2023/07/24.

[3] Colaboradores Ably. Key concepts. Disponible en `https://ably.com/docs/key-concepts`. Revisado el 04/12/2022.

[4] Colaboradores Ably. Pricing. Disponible en `https://ably.com/pricing`. Revisado el 04/12/2022.

[5] Colaboradores Amazon. Amazon simple notification system. Disponible en `https://aws.amazon.com/es/sns`. Revisado el 24/09/2022.

[6] Colaboradores Amazon. Precios de amazon sns. Disponible en `https://aws.amazon.com/es/sns/pricing/`. Revisado el 04/12/2022.

[7] Colaboradores Amazon. ¿qué son los microservicios? Disponible en `https://aws.amazon.com/es/microservices`. Revisado el 05/12/2022.

[8] Paul Bischoff. 24,000 android apps expose user data through firebase blunders. Disponible en `https://www.comparitech.com/blog/information-security/firebase-misconfiguration-report/`, 2021. Revisado el 2023/07/20.

[9] Colaboradores Firebase. Firebase cloud messaging. Disponible en `https://firebase.google.com/products/cloud-messaging`. Revisado el 24/09/2022.

[10] Colaboradores Firebase. Firebase realtime database. Disponible en `https://firebase.google.com/products/realtime-database`. Revisado el 02/12/2022.

[11] Colaboradores Go4It. Principales herramientas para el desarrollo de apis. Disponible en `https://go4it.solutions/es/blog/principales-herramientas-para-el-desarrollo-de-apis`. Revisado el 05/12/2022.

[12] Colaboradores Red Hat. ¿qué son y para qué sirven los microservicios? Disponible en `https://www.redhat.com/es/topics/microservices`. Revisado el 05/12/2022.

[13] Michael X. Heiligenstein. Amazon data breaches: Full timeline through 2023. Disponible en `https://firewalltimes.com/amazon-data-breach-timeline/`, 2023. Revisado el 2023/07/19.

[14] Colaboradores NetApp. ¿qué son los contenedores? Disponible en https://www.netapp.com/es/devops-solutions/what-are-containers/. Revisado el 05/12/2022.

[15] Colaboradores Redis. Redis pub/sub. Disponible en https://redis.io/docs/interact/pubsub/#format-of-pushed-messages. Revisado el 2023/07/20.

[16] Chris Richardson and Floyd Smith. Microservices from design to deployment. Disponible en https://www.nginx.com/blog/introduction-to-microservices/, 2016. Revisado el 24/09/2022.

[17] Morgan Roderick. Github - mroderick/pubsubjs: Dependency free publish/subscribe for javascript. Disponible en https://github.com/mroderick/PubSubJS/. Revisado el 2023/07/20. Última versión: 2018/12/28.

[18] Superbalist. Github - superbalist/laravel-pubsub: A pub-sub abstraction for laravel. Disponible en https://github.com/Superbalist/laravel-pubsub. Revisado el 2023/07/20.

[19] Colaboradores Wikipedia. Kibibyte. Disponible en https://es.wikipedia.org/wiki/Kibibyte. Revisado el 05/12/2022.

[20] Colaboradores Wikipedia. Nosql. Disponible en https://es.wikipedia.org/wiki/NoSQL. Revisado el 02/12/2022.

[21] Colaboradores Wikipedia. Sistema de gestión de aprendizaje. Disponible en https://es.wikipedia.org/wiki/Sistema_de_gestin_de_aprendizaje, 2022. Revisado el 11/09/2022.

ANEXOS

Anexo A

Documentación

A.1. Usos de la librería PubSub de Ucampus en U-Cursos

Se listarán para cada módulo, los archivos que contienen usos de la librería `pubsub` de Ucampus. Para cada archivo se presenta un listado con la función que se utiliza en cada línea de código relacionada a la librería.

Cabe destacar que estos son los usos directos para cada módulo de la librería `pubsub`, pero cada módulo utiliza alguna vez la función del `KERNEL addMensaje()`, la cual llama a la librería para ingresar la notificación generada. Para efecto de este trabajo, solo se considera la definición de dicha función como un uso de la librería pues es esta la que finalmente realiza la llamada a algún método de `pubsub`.

A.1.1. Módulo Kernel

- `carga_script.php`:
  - Línea 33: pubsub::updsub())
  - Línea 64: pubsub::updsub()
- `ucursos.class.php`:
  - Línea 1985: pubsub::initcanales()
  - Línea 2632: pubsub::pub()
  - Línea 2638: pubsub::unpub()
  - Línea 2705: pubsub:: getinfocanalext()
  - Línea 2889: pubsub::cntsub()
  - Línea 2923: pubsub::getsub()
- `menu_usuario.0.php`:

- Línea 20: pubsub::cntsub()
- Línea 27: pubsub::cntsub()

- `menu.0.php`:
  - Línea 24: pubsub::cntsub()
- `caducar_permanentes.php`:
  - Línea 107: pubsub::pub()
- `kernel.class.php`:
  - Línea 1910: pubsub::initcanales()
  - Línea 2101: pubsub::cntsub()
  - Línea 2149: pubsub::getsub()
  - Línea 2254: pubsub::pub()
  - Línea 2260: pubsub::unpub()
- `rewrite_ucampus.php`:
  - Línea 84: pubsub::updsub()
  - Línea 209: pubsub::updsub()

A.1.2. Módulo Canales

- `mis_canales.php`:
  - Línea 7: pubsub::pub()
  - Línea 8: pubsub::unpub()
  - Línea 12: pubsub::readsub()
  - Línea 19: pubsub::updsub()
  - Línea 23: pubsub::updsub()
- `index.php`:
  - Línea 20: pubsub::cntpub()
  - Línea 22: pubsub::readPub()
  - Línea 23: pubsub::getsub()
- `feed.0.php`:
  - Línea 21: pubsub::timeline()
- `leer.0.php`:
  - Línea 20: pubsub::updsub()
- `pendientes.0.php`:

- Línea 19: pubsub::$timeout
- Líneas 20: pubsub::readSub()

- `registrar.0.php`:
  - Línea 34: pubsub::getconf()
  - Línea 37: pubsub::conf()
- `suscribir.0.php`:
  - Línea 23: pubsub::getsub()
  - Línea 27: pubsub::sub()
- `suscripciones.0.php`:
  - Línea 34: pubsub::getsub()
- `unsuscribe.php`:
  - Línea 10: pubsub::getconf()
  - Línea 11: pubsub::conf()
  - Línea 14: pubsub::getsub()
  - Línea 16: pubsub::sub()
- `upd_feed_rss.php`:
  - Línea 164: pubsub::pub()
- `cambiar.php`:
  - Línea 9: pubsub::getsub()
  - Línea 10: pubsub::conf()
  - Línea 13: pubsub::sub()
- `configurar.php`:
  - Línea 16: pubsub::getconf()
  - Línea 22: pubsub::getsub()
  - Línea 29: pubsub::sub()
  - Línea 30: pubsub::initcanales()
  - Línea 35: pubsub::conf()
- `rss20_usuario.php`:
  - Línea 10: pubsub::getconf()
  - Línea 13: pubsub::getsub()
  - Línea 15: pubsub::sub()
  - Línea 17: pubsub::conf()
  - Línea 39: pubsub::readsub()

  - Línea 58: pubsub::updsub()

- `rss20.php`:
  - Línea 12: pubsub::readPub()

A.1.3. Módulo Oauth

- `configurar.php`:
  - Línea 17: pubsub::_expandir()
  - Línea 19: pubsub::getsub()
  - Línea 20: pubsub::getconf()
  - Línea 28: pubsub::sub()()
  - Línea 34: pubsub::conf()
  - Línea 35: pubsub::sub()
- `oauth.class.php`:
  - Línea 162: pubsub::getsub()
  - Línea 421: pubsub::getsub()
  - Línea 422: pubsub::sub()

A.1.4. Módulo Servicios

- `datos.php`:
  - Línea 20: pubsub::pub()
- `integrantes.php`:
  - Línea 35: pubsub::pub()
  - Línea 50: pubsub::pub()
- `visibilidades.php`:
  - Línea 44: pubsub::pub()
- `wizard.php`:
  - Línea 78: pubsub::pub()

A.1.5. Módulo Tareas

- `feedback.php`:
  - Línea 32: pubsub::pub()

A.1.6. Módulo Uchile

- `menu_usuario.0.php`:
  - Línea 20: pubsub::cntsub()
  - Línea 27: pubsub::cntsub()
- `funciones_inicio.php`:
  - Línea 82: pubsub::_fillmensajes()
  - Línea 89: pubsub::_normMensajes()

A.1.7. Módulo Actas

- `index.php`:
  - Línea 64: pubsub::pub()

A.1.8. Módulo Calendario

- `config.php`:
  - Línea 33: pubsub::getsub()
  - Línea 46: pubsub::_expandir()
- `api.php`:
  - Línea 466: pubsub::touchcanal()

A.1.9. Módulo Datos Institución

- `funciones.php`:
  - Línea 33: pubsub::readPub()

A.1.10. Módulo Correo

- `mensajes.php`:
  - Línea 190: pubsub::pub()
- `enviar_programados.php`:
  - Línea 89: pubsub::pub()

A.1.11. Módulo Dropbox

- `dropbox_subir.php`:
  - Línea 6: pubsub::readnotificador()
  - Línea 85: pubsub::unsub()
  - Línea 197: pubsub::pub()
  - Línea 215: pubsub::updsub()
- `configurar.php`:
  - Línea 6: pubsub::getsub()
  - Línea 7: pubsub::getconf()
  - Línea 17: pubsub::sub()
  - Línea 22: pubsub::conf()

A.1.12. Módulo Encuestas

- `funciones.php`:
  - Línea 861: pubsub::pub()

A.1.13. Módulo Foro

- `mensaje.php`:
  - Línea 87: pubsub::getsub()
  - Línea 91: pubsub::pub()

A.1.14. Módulo Horario

- `upd_horarios_simbad.php`:
  - Línea 321: pubsub::pub()

A.2. Ejemplo de uso de Revolt

Este ejemplo de uso se extrajo directamente desde el sitio web oficial del paquete[2]. En él se muestra el uso de algunos métodos que se encuentran disponibles para utilizar la herramienta. Este paquete permite inicializar, repetir funciones, añadir retardo y suspender el *scheduler* o *event loop* utilizado para paralelizar procesos.

```php
<?php

require __DIR__ . '/vendor/autoload.php';

use Revolt\EventLoop;

$suspension = EventLoop::getSuspension();

$repeatId = EventLoop::repeat(1, function (): void {
    print '++ Executing callback created by EventLoop::repeat()' . PHP_EOL;
});

EventLoop::delay(5, function () use ($suspension, $repeatId): void {
    print '++ Executing callback created by EventLoop::delay()' . PHP_EOL;

    EventLoop::cancel($repeatId);
    $suspension->resume(null);

    print '++ Suspension::resume() is async!' . PHP_EOL;
});

print '++ Suspending to event loop...' . PHP_EOL;

$suspension->suspend();

print '++ Script end' . PHP_EOL;
```

Listing A.1: Ejemplo en alto nivel de uso del paquete Revolt, fuente `https://revolt.run/fundamentals`