

INSTITUTO TECNOLÓGICO DE BUENOS AIRES – ITBA
ESCUELA DE INGENIERÍA Y GESTIÓN

THULIUM

DB Evaluation Web App

DOCENTE/S TITULAR/ES O TUTOR/ES: Soliani, Valeria

AUTOR/ES:

Goffan, Martín Alexis (Leg. N° 55431)

Horvat, Eric Nahuel (Leg. N° 55564)

Pascale, Juan Martín (Leg. N° 55027)

**TRABAJO FINAL PRESENTADO PARA LA OBTENCIÓN DEL TÍTULO DE
INGENIERO EN INFORMÁTICA**

Lugar: Av Eduardo Madero 399, C.A.B.A., Argentina

Fecha: 10 de julio de 2019

Índice

Índice	1
1. Abstract	6
2. Introducción	6
3. Justificación	7
4. Marco Teórico	8
4.1. Usuario	8
4.1.1. Usabilidad	8
4.2. Base de datos	9
4.2.1. DBMS	9
4.2.2. Modelo de datos	10
4.2.2.1. Modelo relacional	10
4.2.2.1.1. SQL (Structured Query Language)	10
4.2.2.1.2. Data Definition Language (DDL)	10
4.2.2.1.3. Data Manipulation Language (DML)	11
4.2.2.2. Modelo no relacional (NoSQL)	11
4.2.2.2.1. Clave-Valor	12
4.2.2.2.2. Documentos	12
4.2.2.2.3. Grafos	13
4.2.2.2.4. In-memory	14
4.2.2.2.5. De columnas	14
4.2.2.2.6. De búsqueda	15
4.3. Sistema de gestión de aprendizaje	15
4.4. Autenticación	16
4.4.1. Sistema de autenticación	16
4.4.2. Protocolo Demming - Sacco	17
4.4.3. OAuth2	17
4.4.3.1. Roles	17
4.4.3.2. Flujo de autenticación	18
4.4.4. JSON Web Token	19
5. Estado del Arte	20
5.1. SQL Fiddle [4]	20
5.2. Lagunita [5]	21
5.3. PgExercises [6]	22
5.4. Coding Ground [7]	23
5.5. DataBricks [8]	23
5.6. HackerRank [9]	24

5.7. Cloud9 [10]	25
5.8. Facebook Query Tool [11]	25
5.9. Compose [12]	25
6. Alcance	25
6.1. Requisitos funcionales	26
6.1.1 Características Generales	26
6.1.2. Modalidad playground	27
6.1.3. Modalidad autoevaluación	27
6.1.4. Modalidad examen	27
6.2. Requisitos no funcionales	28
6.3. Cambios en el alcance	28
7. Metodología de Desarrollo	29
7.1. Línea de Tiempo	29
7.2. Ciclo de Vida	29
7.3. Gestión de Cambios	30
8. Casos de Uso	30
8.1. CU1: Alumno / Profesor quiere ejecutar una query	31
8.2. CU2: Alumno / Profesor crea nueva pestaña	32
8.3. CU3: Alumno / Profesor quiere hacer login	33
8.4. CU4: Alumno quiere resolver ejercicios	34
8.5. CU5: Alumno resuelve examen	35
8.6. CU6: Profesor quiere subir un dataset	36
8.7. CU7: Profesor quiere subir un examen	37
8.8. CU8: Profesor revisa un examen	39
9. Arquitectura General	40
9.1. Funcionalidad Requerida	40
9.2. Atributos de Calidad	40
9.2.1. Tolerancia a Fallas + Disponibilidad	40
9.2.2. Interoperabilidad	41
9.2.3. Performance	41
9.2.4. Escalabilidad	41
9.2.5. Usabilidad	41
9.3. Arquitectura candidata	41
9.3.1 Cliente	42
9.3.1.1. Webapp	42
9.3.2. Servidor	42
9.3.2.1. Load Balancer	42
9.3.2.2. Core API	43
9.3.2.3. Websocket API	43
9.3.2.4. Jobs	43

9.3.3. Base de datos interna	44
9.3.4. Dockerizing	44
9.3.5. Kubernetes (k8s)	45
9.4. Verificando atributos de calidad	46
9.4.1. Tolerancia a Fallas + Disponibilidad	46
9.4.2. Interoperabilidad	46
9.4.3. Performance	46
9.4.4. Escalabilidad	46
9.4.5. Usabilidad	46
9.5. Puntos críticos	47
10. Implementación	47
10.1. Roles	47
10.2. Modelado	48
10.2.1. Usuarios	48
10.2.2. Motor	48
10.2.3. Dataset	48
10.2.4. Dataset Ítem	49
10.2.5. Entrada de dataset	49
10.2.6. Pregunta de examen	49
10.2.7. Examen	49
10.2.8. File	50
10.2.9. Respuesta de examen	50
10.2.10. Sesión	50
10.2.11. Instancia de dataset	50
10.2.12. Jobs	51
10.3. Diseño de Módulos	51
10.3.1. Módulos estáticos	52
10.3.2. Base	52
10.3.3. API	52
10.3.4. Internal	53
10.3.5. Jobs	53
10.3.6. Storage	54
10.3.7. Webapp	54
10.3.8. Websocket	55
10.4. Seguridad	56
10.5. API	56
10.5.1. Autenticación e integración con Blackboard	56
10.5.2. Proxy para Blackboard	57
10.5.3. GraphQL	57
10.5.3.1. Entidades de GraphQL	58
10.6. Ejemplo de comunicación entre módulos	59

10.7. Ejecución de queries	59
10.7.1. Dataset instance	59
10.7.2. Parseo de queries, renaming de las tablas	60
10.8. Integración con Blackboard	61
11. Producto final	62
11.1. Pantalla de Login	62
11.2. Pantalla en Modo Playground	63
11.3. Pantalla para crear nuevos archivos	64
11.4. Pantalla para la creación de un dataset	64
11.4.1. Paso 1: Elección de un paradigma	65
11.4.2. Paso 2: Subida de CSV	65
11.4.3. Paso 3: Revisión de los Datos	66
11.4.4. Paso 4: Selección de Permisos	67
11.5. Ejecución de una Query	67
11.6. Creación de un examen	68
11.6.1. Paso 1: Vista de un curso	68
11.6.2. Paso 2: Formulario de creación de un examen	68
11.6.3. Paso 3: Creación de Preguntas	69
11.6.4. Paso 4: El examen se muestra en Campus	69
11.7. Modalidad Examen	70
11.7.1. Paso 1: Identificación de examen vigente	70
11.7.2. Paso 2: Vista del curso	70
11.7.3. Paso 3: Vista del examen	70
11.7.4. Paso 4: Vista de la consigna y envío de respuesta	71
11.8. Corrección de exámenes	71
12. Conclusiones	72
13. Trabajo futuro	72
14. Referencias	73
15. Anexos	74
15.1. Anexo A: Diagramas de actividad de los Casos de uso	74
CU1: Alumno / Profesor quiere ejecutar query	74
CU2: Alumno / Profesor crea nueva pestaña	75
CU3: Alumno / Profesor quiere hacer login	76
CU7: Profesor quiere subir un examen	76
CU8: Profesor revisa un examen	77
15.2 Anexo B: Archivos de módulo de configuración	77
app.json	77
app.secure.json	79
15.3 Anexo C: Rutas de la API	80

1. Abstract

Los motores de base de datos son una herramienta indispensable en el desarrollo de cualquier sistema informático. Por otro lado, las instituciones educativas recaen en sistemas LMS (Learning Management System) para la gestión de su operatoria diaria. En este trabajo proponemos crear una plataforma que ofrezca facilidades en cuanto al uso de motores de base de datos mediante una aplicación web que se conecta con un sistema LMS para extender sus funcionalidades. Esta plataforma permite solucionar un problema intrínseco de los motores de base de datos que es el tiempo y esfuerzo necesario hasta que efectivamente el motor está listo para su uso.

Para realizar esto realizamos un análisis de plataformas similares y relacionadas en el ámbito de lo educacional con respecto a motores de base de datos y plataformas de bases de datos como servicio para extraer ventajas, desventajas y problemas. Exploramos las plataformas y obtuvimos información suficiente para identificar las características más adecuadas y beneficiosas para el desarrollo de la plataforma.

La plataforma propuesta está orientada a resolver problemas existentes que tienen alumnos y profesores en instituciones educativas. Desde el punto de vista del alumno, interactuar con un entorno amigable en el cual se dispone de distintos motores de bases de datos ubicados en la nube, a través de una interfaz web. Por otro lado, proveer a los docentes con facilidades para llevar a cabo evaluaciones, correcciones y distintos tipos de ejercitación. Para ello, definimos una infraestructura escalable que soporta una carga de usuarios determinada y comunicación bilateral con el servicio universitario de autenticación, evaluaciones y cualquier otro tipo de información intercambiable.

2. Introducción

Desde hace tiempo que la instalación de motores de base de datos resulta una actividad poco trivial donde, en ocasiones, es necesaria la participación de un tercero, encargado de los sistemas o infraestructura de la institución. No sólo eso, sino que la instalación es altamente dependiente de la plataforma en la que se instala y de la versión a instalar del motor de base de datos. Esto provoca que, en instituciones educativas, aquellos que son los usuarios finales del motor de base de datos, que llamaremos “alumnos”, encuentren constantemente dificultades para practicar lo que están intentado aprender y en lo que serán evaluados. En ocasiones, incluso pueden hasta aprender de modo incorrecto, dado que cierta consigna puede conseguirse de dos formas distintas dependiendo la versión del motor.

En paralelo, esta complejidad también se extiende a aquellos encargados de evaluar este tipo de trabajos, que llamaremos “profesores”. Los profesores deben pensar cómo llevar a cabo estas evaluaciones, porque lograr que se instale el software del motor de base de datos en todos los equipos disponibles para la evaluación es una tarea grande que puede fallar en el momento de la evaluación y, además, se complementa el hecho de que la tarea de recopilar las respuestas de los alumnos en una examinación de estas características no es simple, por lo que se suele recaer en

la modalidad escrita en papel. Finalmente, el profesor debe reproducir lo escrito en cada uno de los papeles para verificar la correctitud de la respuesta.

Los sistemas LMS más utilizados del mercado suelen ser desarrollados por empresas grandes que tienen productos muy maduros y probados en muchas instituciones. Estas empresas suelen vender las implementaciones para cada institución en particular. Si bien los sistemas LMS suelen tener funcionalidades para evaluar a través de la plataforma, existen limitaciones en cuanto a mecanismos para evaluar. Es común encontrar en estos sistemas formas para tomar evaluaciones multiple choice, verdadero/falso o respuestas escritas, entre otros de los métodos más comunes. Recientemente, los sistemas LMS comenzaron a proveer herramientas para que abiertamente cada institución pueda realizar su integración con su sistema LMS.

Este proyecto se basa en el desarrollo e implementación de una plataforma a la que denominamos **Thulium**, resultado del estudio de plataformas similares y análisis del presente problema. El presente informe pretende detallar los pasos que hemos realizado como así también algunos conceptos que ayudan a comprender el contexto. En la primera parte hacemos un extenso panorama del marco teórico necesario para comprender el problema en su completitud, además, como se trata de una plataforma extensible y genérica, incluimos todos los aspectos relacionados. Luego, analizamos plataformas existentes para entender las características de estos productos y utilizarlas en nuestro favor. En la siguiente sección acordamos y seteamos más claramente los objetivos para lograr el proyecto se adapte a ser un trabajo realizable en el contexto de una materia de la carrera. Allí mismo mostramos una línea de tiempo descriptiva de las etapas para lograr el desarrollo de la plataforma. Finalmente, contamos en profundidad el análisis realizado para lograr la mejor implementación que resuelva el problema. En esta sección se explican las decisiones tomadas con respecto del diseño, implementación y puesta en marcha necesarias para llevar a cabo el desarrollo de la plataforma

3. Justificación

Si bien en la actualidad el Instituto Tecnológico de Buenos Aires (ITBA) cuenta con un sistema LMS, no cuenta con la posibilidad de realizar evaluaciones interactivas y autocorregibles de consultas y operaciones de base de datos.

Las bases de datos son un aspecto extremadamente importante en la carrera de los ingenieros informáticos, y como en su naturaleza son complejas, mayormente requieren demasiado trabajo de configuración y recursos antes de estar en condiciones de ser utilizadas. En otras palabras, el trabajo de configuración o creación de datasets de trabajo es a veces más tedioso que el trabajo de ejecución de queries en sí, y es necesario realizar muchas operaciones antes de poder realmente empezar a trabajar con una base de datos.

Ante esta necesidad, se decidió investigar la factibilidad de crear una plataforma disponible para la comunidad ITBA capaz de ser manejada tanto por alumnos como por profesores. Esta plataforma debe ser capaz de integrarse al sistema LMS existente y permitir llevar a cabo operaciones de bases de datos sin demasiada configuración, realizar evaluaciones que

involucren a las mismas, cargar datasets fácilmente, y realizar distintas actividades cuyos resultados puedan ser cargados en dicho sistema.

El trabajo se divide en dos etapas: análisis y desarrollo. Cada una de estas etapas se ejecutará en dos oportunidades durante el transcurso del proyecto. Durante el período de análisis se presenta un marco teórico y se analizan y comparan distintas aplicaciones que ofrecen facilidades que resultan interesantes para la propuesta de este proyecto final en el estado del arte. Esto tiene como fin decidir qué facilidades utilizar de cada sistema existente. En la etapa de desarrollo, se presenta una implementación de un sistema que abstrae el uso de bases de datos e implementa las facilidades elegidas.

4. Marco Teórico

En esta sección se describen teórica y brevemente los conceptos de conocimiento requerido para la comprensión del informe. A continuación se enumeran los conceptos:

- Base de datos, motor de DB.
- Modelo de datos, relacional y no relacional.
- Sistema de gestión de aprendizaje.
- Autenticación.

Todos los conceptos tratados en esta sección tienen un alto nivel de profundidad y especificación. Más aún, existe conexión con aspectos técnicamente más profundos; pero no es el objetivo del trabajo explicarlos.

4.1. Usuario

Un usuario es un individuo que utiliza una computadora, sistema operativo, servicio o cualquier sistema. Se utiliza para clasificar a diferentes privilegios o permisos a los que tiene acceso un usuario o grupo de usuarios, para interactuar o ejecutar con la computadora o con programas ya instalados.

4.1.1. Usabilidad

“Grado de eficacia (effectiveness), eficiencia (efficiency) y satisfacción con la que usuarios específicos pueden lograr objetivos específicos en contextos de uso específicos.” **ISO 9241 - 11**

Donde se define:

- *Eficacia*: La exactitud con la cual los usuarios logran objetivo específicos.
- *Eficiencia*: Los recursos utilizados en relación a la exactitud con la cual el usuario logra los objetivos.
- *Satisfacción*: Libertad de incomodidad y actitud positiva ante el uso del producto.
- *Contexto de uso*: Características del usuario, tareas y el entorno.
- *Objetivo*: resultado esperado.

- *Tarea:* Actividades requeridas para lograr un objetivo.

4.2. Base de datos

Colección de datos relacionados, que cumple con las siguientes propiedades:

- Representa algún aspecto del mundo real (proviene de alguna fuente o mini-universo).
- Tiene coherencia lógica con algún significado inherente (no son números aleatorios).
- Se diseña, construye y se completa con datos para un propósito específico. Puede tener diferentes grupos de usuarios cada uno con intereses distintos y además cuenta con ciertas aplicaciones preconcebidas para lograr cierto objetivo.

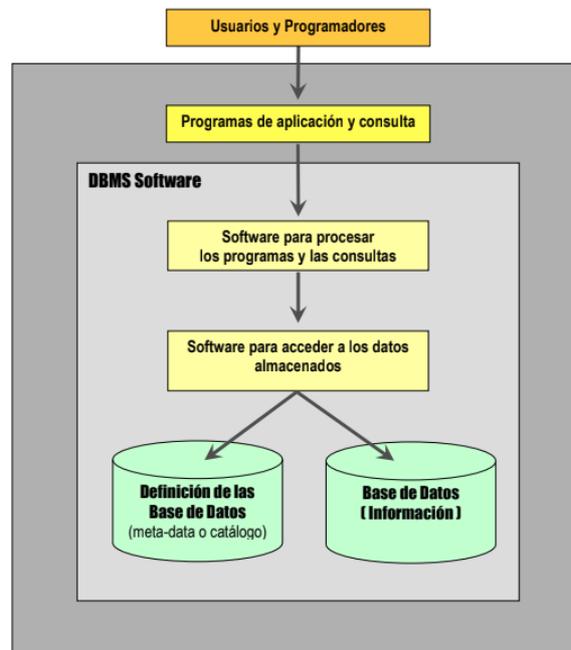


Fig 1: DBMS

4.2.1. DBMS

Es una colección de programas que permiten crear y mantener una Base de Datos.

Su creación implica definir los tipos de datos, estructura y restricciones de los datos que se almacenen. Mantenerla implica actualizar sus datos para que reflejen los cambios del mini-universo que representa, generar reportes y poder consultar la información que almacena.

Un DBMS no sólo contiene los datos de la base sino la completa definición de la misma, la cual se guarda en su catálogo (catalog). Dicho catálogo posee la estructura subyacente, el formato y tipo de cada ítem, y restricciones de los datos (si las hubiera).

En general el catálogo es usado por el software DBMS, pero podría ser consultado por los usuarios que desearan conocer la estructura subyacente.

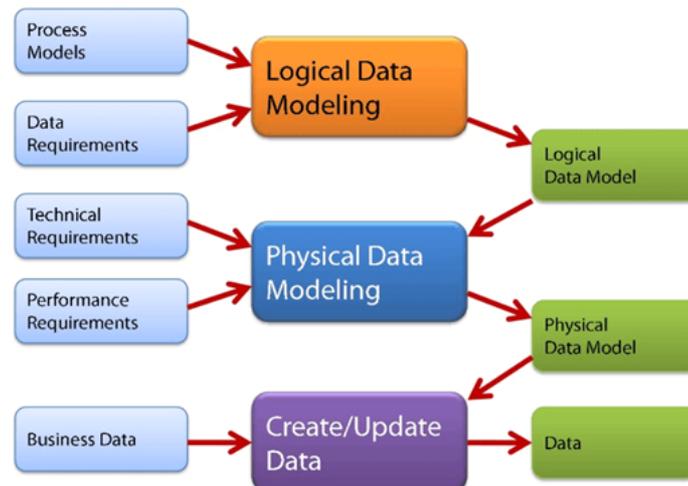


Fig 2: Data modeling

4.2.2. Modelo de datos

Conjunto de conceptos que se utilizan para describir la estructura de la base de datos (tipos de datos, relaciones y ciertas restricciones).

4.2.2.1. Modelo relacional

En este modelo todos los datos son almacenados en relaciones, y como cada relación es un conjunto de datos. El orden en el que estos se almacenen no tiene relevancia. La información puede ser recuperada o almacenada por medio de consultas que ofrecen una amplia flexibilidad para poder administrar la información.

El núcleo del modelo relacional lo constituyen las relaciones, que están formadas por atributos. Cada atributo toma sus valores de un dominio que restringen los valores que podrán tomar las tuplas en los mismos. Las restricciones del modelo relacional constituyen una forma fuerte de validación que libera a los programas que acceden a bases de datos de tener que ser ellos los que deban realizar los chequeos.

4.2.2.1.1. SQL (Structured Query Language)

SQL es un lenguaje de consulta híbrido entre álgebra relacional y cálculo relacional. Por un lado, desde la sintaxis del lenguaje, es decir, desde el punto de vista del programador, está fuertemente influido por el cálculo relacional. Por otro lado, desde su implementación, es decir, desde el punto de vista de los motores DBMS, se utiliza el Álgebra Relacional para reescribir la consulta SQL original y generar una equivalente pero más eficiente.

4.2.2.1.2. Data Definition Language (DDL)

Un lenguaje de definición de datos es un lenguaje proporcionado por el DBMS que permite a los programadores de la misma llevar a cabo las tareas de definición de las estructuras que almacenarán los datos así como de los procedimientos o funciones que permitan consultarlos.

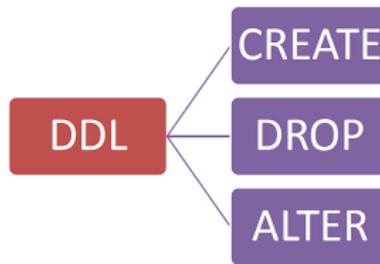


Fig 3: DDL

4.2.2.1.3. Data Manipulation Language (DML)

Un lenguaje de manipulación de datos es un lenguaje proporcionado por los DBMS que permite a los usuarios de la misma llevar a cabo las tareas de consulta o modificación de los datos contenidos en las Bases de Datos del Sistema Gestor de Bases de Datos.

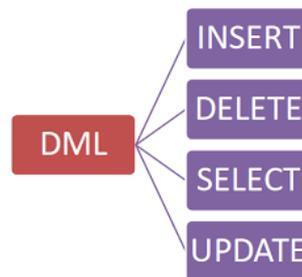


Fig 4: DML

4.2.2.2. Modelo no relacional (NoSQL)

Las bases de datos [NoSQL](#) [1] utilizan una variedad de modelos de datos para acceder y administrar datos, como documentos, gráficos, clave-valor, in-memory y de búsqueda. Estos tipos de bases de datos están optimizados específicamente para aplicaciones que requieren grandes volúmenes de datos, baja latencia y modelos de datos flexibles, lo que se logra mediante la flexibilización de algunas de las restricciones de coherencia de datos en otras bases de datos.

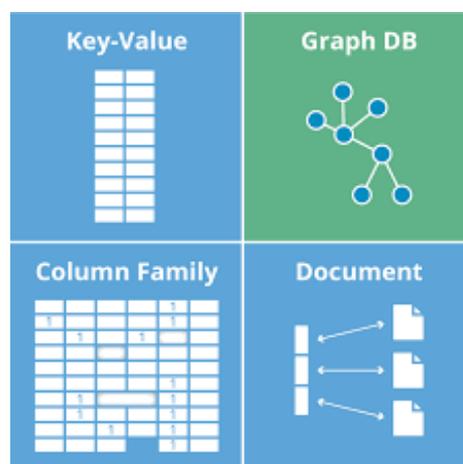


Fig 5: NoSQL

A diferencia del modelo relacional, las bases de datos no relacionales aportan:

- *Flexibilidad:* generalmente ofrecen esquemas flexibles que permiten un desarrollo más rápido y más iterativo. El modelo de datos flexible hace que las bases de datos NoSQL sean ideales para datos semiestructurados y no estructurados.
- *Escalabilidad:* generalmente están diseñadas para escalar usando clústeres distribuidos de hardware en lugar de escalar añadiendo servidores caros y sólidos. Algunos proveedores de la nube manejan estas operaciones fuera del alcance del usuario, como un servicio completamente administrado.
- *Alto rendimiento:* están optimizadas para modelos de datos específicos (como documentos, clave-valor y gráficos) y patrones de acceso que permiten un mayor rendimiento, en lugar de lograr una funcionalidad similar con bases de datos relacionales.
- *Altamente funcional:* proporcionan APIs altamente funcionales y tipos de datos que están diseñados específicamente para cada uno de sus respectivos modelos de datos.

4.2.2.2.1. Clave-Valor

Una base de datos clave-valor es un tipo de base de datos no relacional que utiliza un método simple de clave-valor para almacenar datos. Una base de datos clave-valor almacena datos como un conjunto de pares clave-valor en los que cada clave sirve como un identificador único. Tanto las claves como los valores pueden ser cualquier cosa, desde objetos simples hasta objetos compuestos complejos. Las bases de datos clave-valor son altamente divisibles y permiten el escalado horizontal a escalas que otros tipos de bases de datos no pueden alcanzar. Generalmente, a la hora de dividir los datos en nodos de un cluster se utilizan claves de partición (Partition Key).

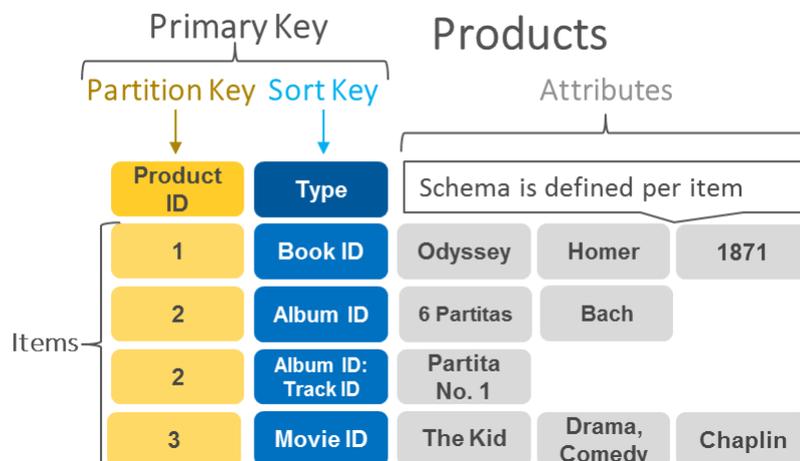


Fig 6: Base de datos de clave-valor escalada horizontalmente

4.2.2.2.2. Documentos

Una base de datos de documentos es un tipo de base de datos no relacional que ha sido diseñada para almacenar y consultar datos como documentos de tipo JSON. Las bases de datos de documentos facilitan a los desarrolladores el almacenamiento y la consulta de datos en una base de datos mediante el mismo formato de modelo de documentos que emplean en el código

de aplicación. La naturaleza flexible, semi-estructurada y jerárquica de los documentos y las bases de datos de documentos permite que evolucionen según las necesidades de las aplicaciones. El modelo de documentos funciona bien con casos de uso como catálogos, perfiles de usuario y sistemas de administración de contenido en los que cada documento es único y evoluciona con el tiempo. Las bases de datos de documentos permiten una indexación fácil, potentes consultas ad hoc y análisis de colecciones de documentos.



Fig 7: Base de datos de documentos en comparación con el modelo relacional

4.2.2.2.3. Grafos

Las bases de datos de grafos están diseñadas expresamente para almacenar relaciones y navegar por ellas. Las relaciones son ciudadanos de primera clase en las bases de datos de grafos, y la mayor parte del valor de las bases de datos de grafos deriva de estas relaciones. Las bases de datos de grafos usan nodos para almacenar entidades de datos y bordes para almacenar relaciones entre entidades. Un borde siempre tiene un nodo de inicio, un nodo final, un tipo y una dirección, y puede describir las relaciones principales y secundarias, las acciones, la propiedad y cosas similares. No hay límite para la cantidad y el tipo de relaciones que un nodo puede tener.

Un grafo de una base de datos de grafos se puede desplazar a lo largo de tipos de bordes específicos, o por todo el grafo. En las bases de datos de grafos, atravesar las uniones o las relaciones es muy rápido porque las relaciones entre los nodos no se calculan en los tiempos de consulta, sino que se mantienen en la base de datos. Presentan ventajas con respecto a los casos de uso como las redes sociales, los motores de recomendaciones y la detección del fraude, donde es necesario crear relaciones entre los datos y consultarlas rápidamente.

La figura 8 es un ejemplo de un grafo de red social. Dadas las personas (nodos) y sus relaciones (bordes), es posible averiguar quiénes son los "amigos de los amigos" de una persona en particular, por ejemplo, los amigos de los amigos de Jack.

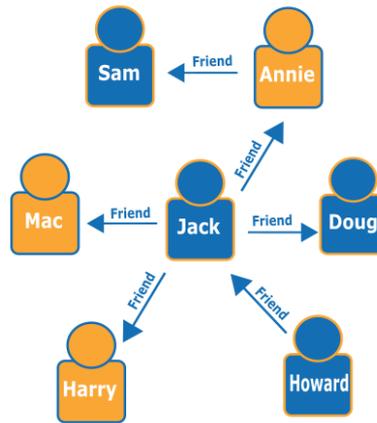


Fig 8: Red social basada en una base de datos de grafos

4.2.2.2.4. In-memory

Una base de datos en memoria es un tipo de base de datos no relacional que recae en la memoria para el almacenamiento de datos, a comparación del resto de las base de datos que lo guardan datos en discos o SDDs. Estas bases de datos son diseñadas para lograr tiempo de respuesta minimal, al eliminar el acceso a disco (Overhead IO).

Al tener los datos guardados y manejados exclusivamente en la memoria principal, hay un riesgo de los mismos se pierdan con un error de proceso o de servidor. Las base de datos en memoria pueden persistir los datos en discos a través de guardados de logs o snapshots de las mismas, pero en general son volátiles y cuando se apagan o el proceso que las administra deja de correr, los datos se pierden.

Las base de datos en memoria son ideales para aplicaciones que requieren tiempos de respuesta en orden de microsegundos y pueden tener grandes picos de tráfico en cualquier momento.

4.2.2.2.5. De columnas

Una base de datos columnar está optimizada para leer y escribir columnas de datos en lugar de filas. El almacenamiento basado en columnas para las tablas de bases de datos es un factor importante en el desempeño de las consultas analíticas, ya que reduce notablemente los requisitos globales de E/S del disco (IO), y reduce la cantidad de datos que hay que cargar desde el mismo.

De la misma forma que otras bases de datos NoSQL, las bases de datos columnares están diseñadas para reducir la escala utilizando clusters distribuidos de hardware de bajo coste para aumentar el desempeño, de manera que resultan ideales para el almacenamiento de datos y el procesamiento de Big Data.

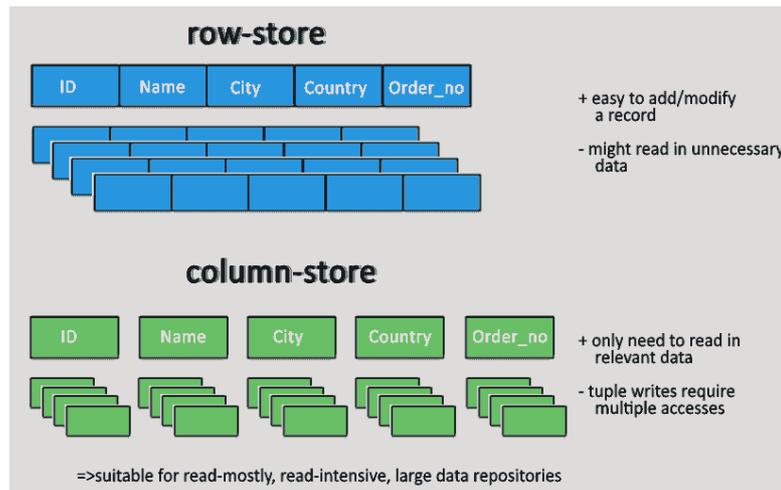


Fig 9: Base de datos de columnas

4.2.2.2.6. De búsqueda

Una base de datos de búsqueda es un tipo de base de datos no relacional que se especializa en buscar el contenido de los datos. Estas bases de datos usan índices para categorizar características similares a lo largo de los datos y facilitar la capacidad de búsqueda. Las bases de datos de búsqueda están optimizadas para tratar con datos que pueden ser grandes, semi-estructurados, sin estructura, y suelen ofrecer métodos específicos como expresiones de búsqueda complejas, ranking de resultados de búsqueda o full-text search.

4.3. Sistema de gestión de aprendizaje

Un sistema de gestión de aprendizaje (SGA; en inglés, learning management system o LMS) es un software instalado en un servidor web que se emplea para administrar, distribuir y controlar las actividades de formación no presencial (o aprendizaje electrónico) de una institución u organización. Permitiendo un trabajo de forma asíncrona entre los participantes.

Las principales funciones del sistema de gestión de aprendizaje son: gestionar usuarios, recursos así como materiales y actividades de formación, administrar el acceso, controlar y hacer seguimiento del proceso de aprendizaje, realizar evaluaciones, generar informes, gestionar servicios de comunicación como foros de discusión, videoconferencias, entre otros.

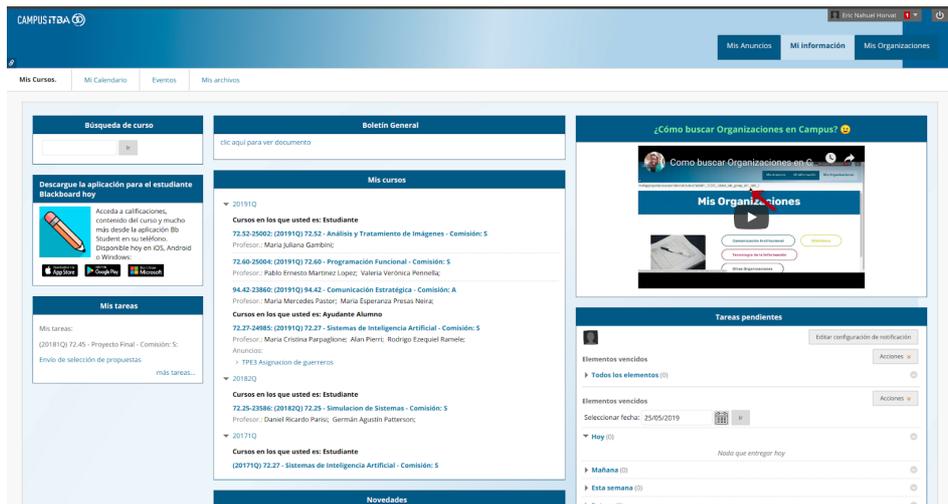


Fig 10: Campus, sistema de gestión de aprendizaje del ITBA

4.4. Autenticación

Procedimiento que permite asegurar que un usuario de un servicio es auténtico o quien dice ser.

4.4.1. Sistema de autenticación

Compuesto por:

- $A = \{ a \}$ - Información de autenticación
 - La designación de la entidad e información provista
- $C = \{ c \}$ - Información complementaria
 - Contiene la especificación de un principal e información almacenada por el sistema
- $F = \{ F: A \rightarrow C \}$ - Funciones de complementación
 - Deriva información complementaria
- $L = \{ L: A \times C \rightarrow \{ 0, 1 \} \}$ - Funciones de autenticación
 - Determina si un par A,C son una asociación válida
- $S = \{ s \}$ - funciones de selección
 - Funciones que permiten crear y actualizar A y C

Al momento de autenticar se debe tener cierta información para la misma, esta debe ser provista por una entidad externa, puede poseer diferentes grados de confianza y venir de diversas fuentes:

- Algo que conozco
- Algo que tengo
- Algo que soy
- Dónde estoy (contexto)

4.4.2. Protocolo Demming - Sacco

Protocolo que sirve para comunicar dos usuarios a través de un canal cifrado basado en la confianza de una entidad centralizada, llamada KDC (Key Distribution System). Está basada en los siguientes pasos:

- Se asume que todo usuario se comunica de manera segura con el KDC (C).
1. A desea comunicarse con B, por lo que se comunica con el KDC, diciendo quién es, con quien quiere comunicarse y determinada información.
 2. El KDC le responde bajo la clave pública de A, que el mensaje es para A, que quiere hablar con B y la información mencionada en el paso anterior. Además, le agrega la clave o secreto para la próxima comunicación con B y el mensaje que debe enviarle a B.
 3. A le manda a B un mensaje cifrado con la clave pública de B que recibió en el paso anterior del KDC. El mismo contiene que debe comunicarse con A, la clave de la comunicación y un timestamp (por cuestiones de seguridad para que no haya un ataque de repetición de mensaje).
 4. B le responde a A y comienza la comunicación a través de la clave provista por el KDC.

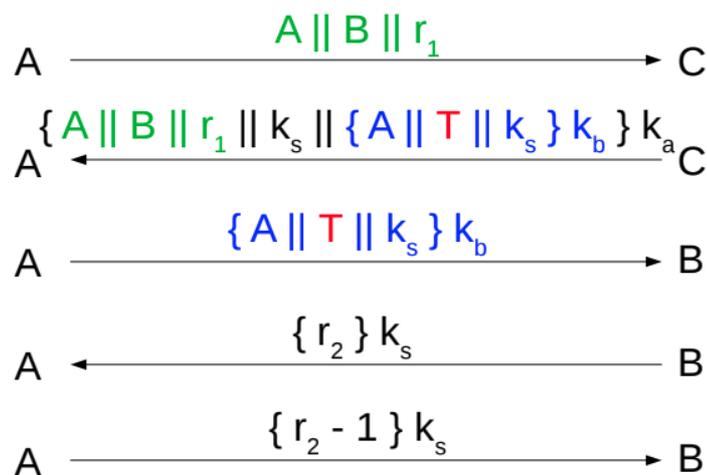


Fig 11: Protocolo Demming Sacco

4.4.3. OAuth2

[OAuth2](#) [2] es un framework de autorización que le permite a las aplicaciones obtener acceso limitado a cuentas de usuario en un servicio HTTP. Delega la autenticación del usuario al servicio que aloja la cuenta del mismo y autoriza a las aplicaciones de terceros el acceso a dicha cuenta de usuario. OAuth2 proporciona flujos de autorización para aplicaciones web y de escritorio; y dispositivos móviles.

4.4.3.1. Roles

El flujo de OAuth2 se basa en cuatro roles:

- *Propietario del recurso:* El propietario del recurso es el "usuario" que da la autorización a una aplicación, para acceder a su cuenta. El acceso de la aplicación a la cuenta del

usuario se limita al "alcance" de la autorización otorgada (e.g. acceso de lectura o escritura).

- *Cliente*: es la aplicación que desea acceder a la cuenta del usuario. Antes de que pueda hacerlo, debe ser autorizado por el usuario, y dicha autorización debe ser validada por la API.
- *Servidor de recursos*: aloja las cuentas de usuario protegidas.
- *Servidor de autorización*: verifica la identidad del usuario y luego genera tokens de acceso a la aplicación.

Desde el punto de vista del desarrollador de una aplicación, la API del servicio atiende tanto a los roles de recursos como a los de autorización. Nos referiremos a ambos roles combinados, como al rol de servicio o de API. Este rol funciona como un KDC en el protocolo Demming - Sacco, que atribuye tokens.

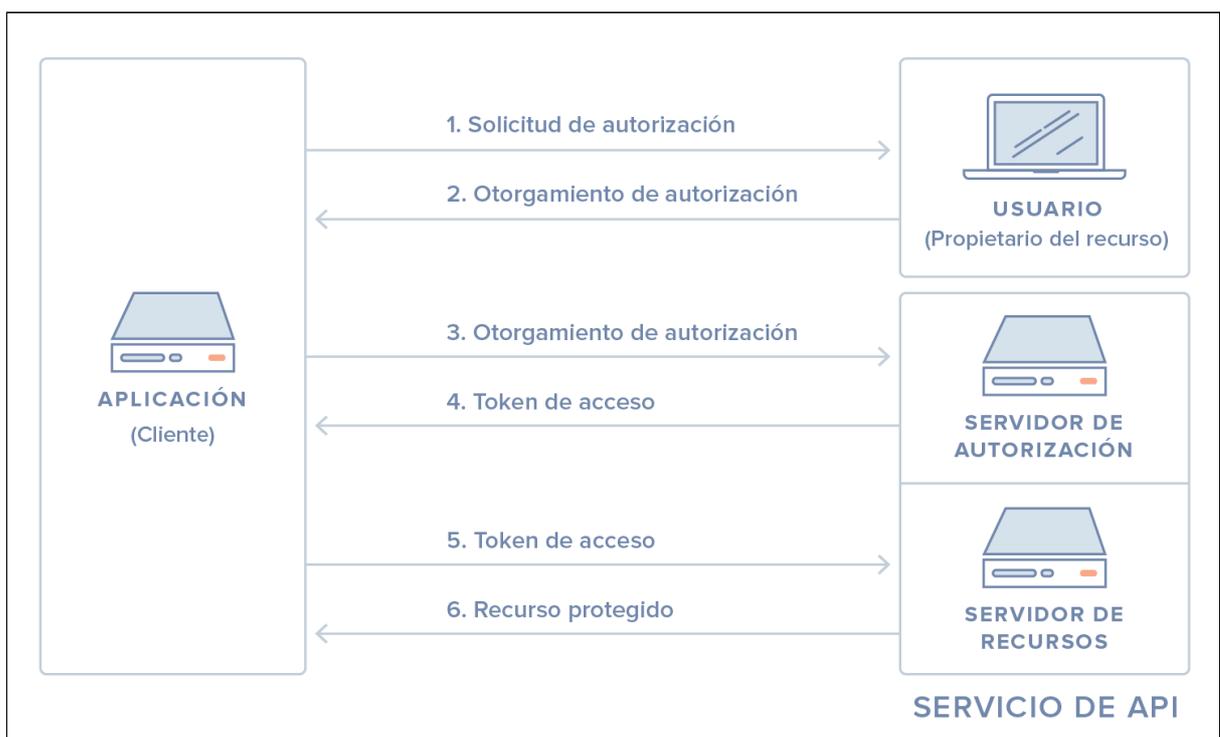


Fig 12: Flujo de OAuth2

4.4.3.2. Flujo de autenticación

El flujo de OAuth2, representado en la figura 12, consiste en:

1. La aplicación solicita autorización para acceder a los recursos de servicio del usuario.
2. Si el usuario autoriza la solicitud, la aplicación recibe la autorización.
3. La aplicación solicita al servidor de autorización (API), presentando la autenticación de su identidad y la autorización otorgada. La aplicación solicita al servidor de autorización (API) un token de acceso presentando la autenticación de su propia identidad y la autorización otorgada.
4. Si la identidad de la aplicación es autenticada y la autorización es válida, el servidor de autorización (API) emite un token de acceso a la aplicación. La autorización finaliza.

5. La aplicación solicita el recurso al servidor de recursos (API) y presenta el token de acceso para autenticarse.
6. Si el token de acceso es válido, el servidor de recursos (API) provee el recurso a la aplicación.

4.4.4. JSON Web Token

Un [JSON Web Token \(JWT\)](#) [3] es un objeto JSON definido en el RFC 7519. Es una forma segura de representar un conjunto de información de forma segura entre dos partes. El token se comprende de un encabezado (header), un contenido (payload) y una firma (signature). Es decir tiene el siguiente formato:

```
header.payload.signature
```

A modo de ejemplo, para ilustrar cómo se usan los JWT, en la figura 13, se muestra un diagrama con tres entidades que son: el usuario, el servidor de autenticación y el servidor de la aplicación. El servidor de autenticación provee de un JWT al usuario, una vez que se autentica correctamente y con ese JWT el usuario puede comunicarse de forma segura con la aplicación.

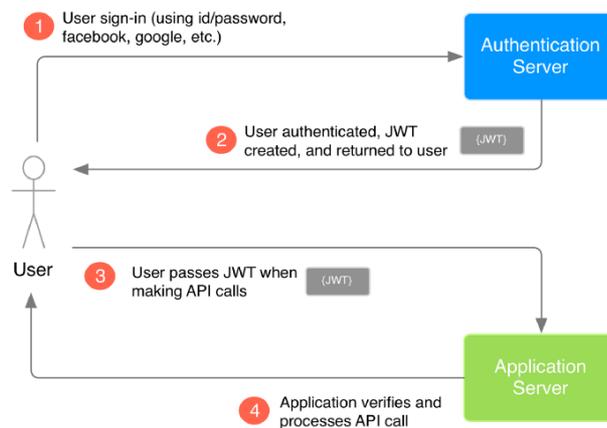


Fig 13: Autenticación e interacción con JWT

Para construir un JWT se generan dos objetos JSON: el header y el payload. El header contiene información sobre cómo se debe computar la firma. El formato del header es el siguiente:

```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

El campo *typ* define que el objeto es un JWT y el campo *alg* define qué algoritmo de hashing se utilizó para crear el JWT.

El payload es un objeto JSON que contiene la información que se desea guardar en el JWT. Existen campos estandarizados enumerados a continuación:

Campo	Significado	description
iss	Issuer (Emisor)	Identifica el emisor del JWT.
sub	Sujeto (Subject)	Identifica al sujeto identificado por el JWT.
aud	Audience (Audiencia)	Identifica los destinatarios a los que se dirige el JWT.
exp	Expiration Time (Tiempo de Expiración)	Determina la caducidad del JWT. Dice a partir de cuándo el JWT no debe ser aceptado para su procesamiento.
nbf	Not Before (No antes de)	Determina a partir de qué momento el JWT puede ser aceptado para procesarse.
iat	Issued at (Fecha de Emisión)	Determina cuándo se emitió el JWT.
jti	JWT ID	Identificador único del JWT, inclusive entre distintos emisores.

La firma se computa de acuerdo al siguiente pseudocódigo:

```
data = base64urlEncode(header) + "." + base64urlEncode(payload)
hashedData = hash(data, secret)
signature = base64urlEncode(hashedData)
```

JWT se utiliza para probar que la información enviada fue creada por una fuente auténtica. El JWT se codifica en base 64 y se firma, por lo que no está encriptado. La finalidad de la firma es poder verificar la autenticidad de un JWT. Para verificar un JWT existe un secreto compartido entre el servidor de autenticación y el servidor de la aplicación. Entonces, el servidor de la aplicación puede verificar si un JWT fue firmado con la misma clave que aquella del servidor de autenticación, si la firma no es válida entonces se descarta el pedido por no estar autenticado.

5. Estado del Arte

En esta sección se enumeran distintos tipos de servicios online que ofrecen características que nuestro sistema debería contener.

5.1. [SQL Fiddle](#) [4]

Características:

- Ejecución de queries, a base de la construcción del schema con DDL.
- Ejecución de queries, a partir de la entrada manual de la misma.
- Permite ver el plan de ejecución de la DB.
- Permite ver un cuadro de resultado.

- Posee una herramienta básica para convertir texto a DDL (sin ayuda de uso, no intuitiva; y poca, impráctica e incompleta documentación en un repositorio de github).

Esta plataforma resulta interesante dado que permite ver el plan de ejecución, y los resultados de la query. La idea de texto a DDL es buena, pero al ser difícil de manejar, resulta una complicación más que una ayuda.

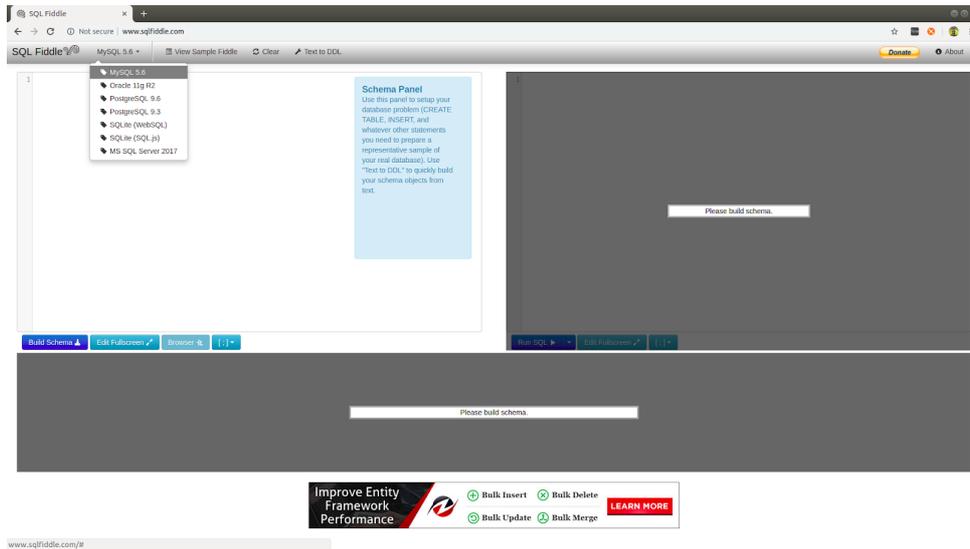


Fig 14: SQL Fiddle

5.2. [Lagunita](#) [5]

Características:

- Ejecución de queries de consulta, a partir de entrada manual.
- Comparación de resultados obtenidos contra resultados esperados.
- Se interpretan los errores y se muestra un mensaje acorde (fácil debido a limitación).
- Solo permite una ejecución a la vez.
- DDL no modifica la DB ni devuelve error, parece ser estático.

Esta plataforma sirve como ejemplo para tomar exámenes, aunque no es útil el hecho de que no permita ejecutar queries DDL, ni avisar que los cambios no se hicieron. La indicación de errores es valiosa de la manera implementada.

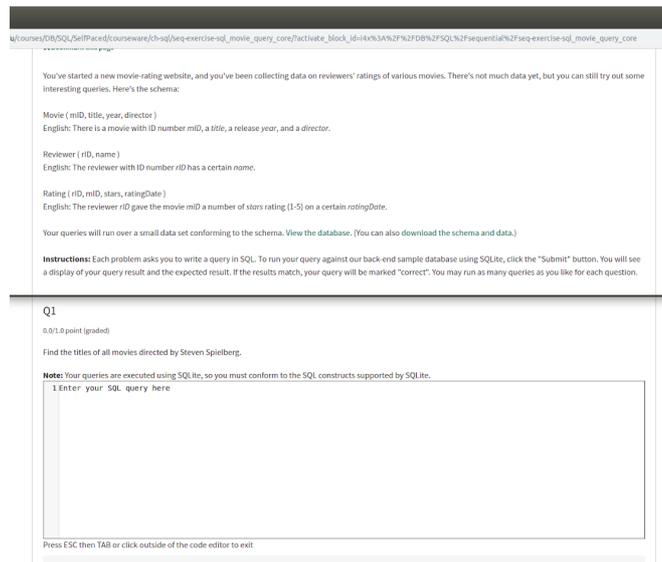


Fig 15: Lagunita

5.3. [PgExercises](#) [6]

Características:

- Ejecución de queries de consulta, a partir de entrada manual.
- Comparación de resultados obtenidos contra resultados esperados.
- Se interpreta los errores y muestra un mensaje acorde (fácil debido a limitación).
- Solo permite una ejecución a la vez.
- DDL modifica la DB solo en los ejercicios que lo requieren.
- Tiene preguntas tipo examen.

Al igual que Lagunita, resulta una buena inspiración a seguir, tanto en modo de examen, donde agrega preguntas que no están en formato de base de datos, como en muestra de errores. Además, permite queries de modificación de DB con DDL, algo que debemos implementar en este proyecto final.

Insert calculated data into a table

Question

Let's try adding the spa to the facilities table again. This time, though, we want to automatically generate the value for the next field, rather than specifying it as a constant. Use the following values for everything else:

- Name: 'Spa', membercost: 20, guestcost: 30, initialoutlay: 100000, monthlymaintenance: 800.

Expected Results

facid	name	membercost	guestcost	initialoutlay	monthlymaintenance
0	Tennis Court 1	5	25	10000	200
1	Tennis Court 2	5	25	8000	200
2	Badminton Court	0	15.5	4000	50
3	Table Tennis	0	5	320	10
4	Massage Room 1	35	80	4000	3000
5	Massage Room 2	35	80	4000	3000
6	Squash Court	3.5	17.5	5000	80
7	Snooker Table	0	5	450	15
8	Pool Table	0	5	400	15
9	Spa	20	30	100000	800

Answers and Discussion

Fig 16: PgExercises

5.4. [Coding Ground](#) [7]

Características:

- Ofrece acceso a una terminal.
- Permite hacer todo tipo de consultas (DDL y DML).

Coding Ground es útil en cuanto a la libertad de las queries, aunque en nuestro proyecto es necesario limitar a los alumnos en cuanto a algunas acciones. Al ser terminal, no posee una buena UX.

The screenshot shows the Coding Ground interface. On the left, a terminal window displays the following SQL code:

```
-- Create the schema
CREATE SCHEMA schema;
-- Create the table
CREATE TABLE schema.tables (
  name VARCHAR(255) NOT NULL,
  version VARCHAR(255) NOT NULL,
  PRIMARY KEY (name, version)
);
-- Insert data
INSERT INTO schema.tables (name, version) VALUES ('table', 'v1');
-- Query the data
SELECT * FROM schema.tables;
```

On the right, a sidebar contains the following navigation options:

- Hacer un Curriculum
- View CV Templates
- Plantillas Curriculum Vitae
- CV en Español
- Modelos de Curriculum

Fig 17: Coding Ground

5.5. [DataBricks](#) [8]

Características:

- Optimiza un contenedor de Spark [7].
- Provee:
 - Conjunto de herramientas orientadas a cloud databases.
 - Facilidades que permiten analizar Big Data mediante un click.
 - Dashboards interactivos con comandos que se pueden importar/exportar.
- Permite programar la ejecución de tareas.
- Crea multi-stage pipelines para el control de estructuras del lenguaje de programación que se usa.

DataBricks es una plataforma muy completa, no limitada a base de datos, que permite herramientas interesantes, como Big Data en Bases de Datos en la nube. Es un modelo a seguir en cuanto a escalabilidad, y opciones de importación de datos.

5.6. [HackerRank](#) [9]

Características:

- Plataforma que propone problemas a resolver.
- Ejecuta distintos entornos de lenguajes de programación.
- Posee un IDE integrado en la UI.
- Corre tests cases con sus respectivas entradas y salidas esperadas correspondientes.
- El programa, dependiendo del lenguaje de programación, tiene un límite de tiempo en ejecución.

HackerRank es la plataforma de excelencia para evaluaciones. El IDE, la interoperabilidad de lenguajes, los tiempos límites de ejecución y el control bajo tests de entradas y salidas (tanto visibles como ocultos), son características en la que se basó Thulium.

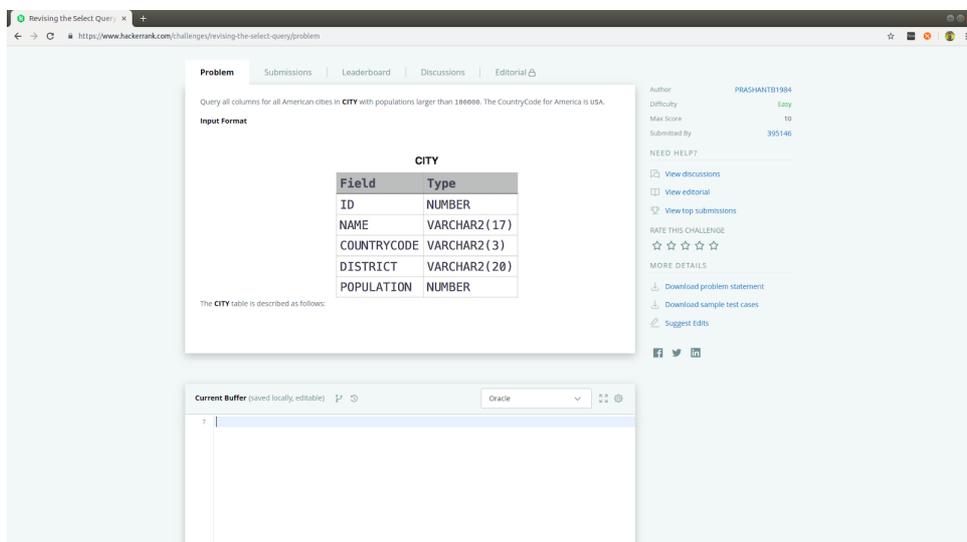


Fig 18: Hacker Rank

5.7. [Cloud9](#) [10]

Características:

- Se ejecuta en una máquina virtual.
- Ofrece acceso a una terminal en la que se puede usar git, apt y otros tipos de comandos de consola de linux para setear el entorno, por lo que fácilmente permite instalar una base de datos.
- Provee un subdominio que permite escuchar un servidor en un puerto.

Esta plataforma sirve como ejemplo de cómo encapsular y separar los entornos, de forma tal que las acciones de un usuario no afecten a los demás, característica obligatoria en nuestra implementación.

5.8. [Facebook Query Tool](#) [11]

Características:

- Permite pedir un access token y permisos para interactuar con el backend de Facebook.
- Tiene 2 modalidades mediante el estilo “API” y “FQL” (Facebook Query Language), que es un DSL creado sobre SQL.

Esta herramienta de Facebook aporta su método de autenticación a través de tokens. Además, permite la utilización y parseo para highlighting de queries de lenguajes basado en SQL.

5.9. [Compose](#) [12]

Características:

- Ofrece una base de datos como un servicio.
- Permite montar varios motores de base de datos y su interfaz web permite realizar queries uno por uno sobre los distintos motores.

Compose tiene como característica destacada la interoperabilidad de motores de DB.

6. Alcance

El objetivo principal de este Proyecto Final, se basa en crear una plataforma web, llamada Thulium que permitirá a alumnos y profesores interactuar con distintos motores de bases de datos.

Para ambos debe ser fácil interactuar con la plataforma, es decir, el diseño de la misma debe estar orientado a la UX de los usuarios. Los usuarios deben tener recursos disponibles para, de forma sencilla y amigable, ejecutar consultas de tipo SQL.

La autenticación debe ser sencilla. Al principio del desarrollo del Proyecto Final, una versión stand-alone, de fácil y rápida implementación, para luego implementar una integración con Campus, como se menciona en la sección 10.5.1.

Thulium debe poder ser utilizado en cualquier materia de la universidad, por lo que debe poseer interoperabilidad entre bases de datos relacionales y no relacionales.

Desde el punto de vista del alumno, Thulium debe poder garantizar la interacción, mediante un entorno amigable, con distintos motores de DB (que se mencionarán más adelante) para realizar distintos ejercicios, como prácticas libres. Es requisito que los usuarios no se vean afectados por las acciones de otros usuarios. A su vez, ofrecer mecanismos usables para rendir exámenes, ya que no debería agregar dificultades no inherentes al examen en sí.

Por otro lado, Thulium debe proveer a los docentes con herramientas para crear y tomar exámenes, realizar correcciones y crear distintos tipos de ejercicios. Todo esto debe poder hacerse de manera intuitiva, sin que el usuario se vea frustrado.

En resumen, los objetivos para este proyecto final son los siguientes:

- Desarrollo de una aplicación web funcional, construido sobre un sistema escalable, con una lógica que permite ejecutar operaciones sobre motores de bases de datos.
- Lograr un diseño amigable y usable.
- Cargar datasets desde un CSV.
- Tomar exámenes, y realizar distintos tipos de ejercicios, respetando cierta lógica de permisos.
- Integrar con el sistema de gestión académica de ITBA, Campus, que permite autenticarse en la plataforma utilizando las credenciales del mismo, y realizar un manejo de cierta información de los usuarios en conjunto.
- Establecer las bases para poder conectarse con Campus ITBA. En otras palabras, el valor de este proyecto no solamente se encuentra en la implementación de la aplicación, sino que también en un trabajo de investigación que permita que, a partir de ahora, futuros desarrollos de otras aplicaciones también se puedan integrar con Campus.
- Lograr que la aplicación implementada abstraiga una numerosa cantidad de features que se encuentran en los motores de Postgres, MySQL y MongoDB.

6.1. Requisitos funcionales

Los requisitos funcionales de Thulium fueron definidos según tres modos posibles: playground, auto-evaluación y examen teniendo en mente dos tipos de usuario: alumno y profesor.

6.1.1 Características Generales

A continuación se describen las características generales que debe presentar la aplicación web para los distintos roles:

Alumno:

- Proveer un editor SQL con syntax highlighting y predicción de query.
- Autenticación: por campus o por google.
- Obtener un resumen de resultados, tiempo de ejecución, y cualquier característica destacable luego de una query.
- Poder ejecutar y detener una query con tan solo un botón (UX).

Profesor:

- Heredar todas las características del rol alumno.
- Import/Export de datos (En CSV)
- ABM datasets y exámenes.

6.1.2. Modalidad playground

En este modo, el usuario puede interactuar con el motor de base de datos seleccionado. Se pensaron los siguiente requisitos:

- Debe tener una pantalla de configuración inicial para elegir motor, dataset, y demás configuración, basándonos en Compose.
- Poder cambiar motor, simil a la implementación de SQLFiddle, un dropdown de opciones disponibles en una fixed bar, ubicada en el sector superior. En el resto de los modos esto no debería ser modificable.
- Poder elegir un dataset precargado.
- Importar un Dataset.
- Poder realizar consultas sobre el Dataset.

6.1.3. Modalidad autoevaluación

El docente debe poder cargar conjuntos de datos en modo guia de ejercicios. El alumno debe poder acceder a estos datos, y modificarlos si el docente así lo quisiera, en un entorno local (es decir, en un entorno propio del usuario y que no afecte a otros).

- Se eligen las preguntas y los casos de prueba para cada pregunta. Cada caso de prueba consta de un resultado esperado al ejecutar una consulta, que se obtiene de una query armada por el docente.
- La modalidad puede o no tener fecha de expiración, y no tiene límite de tiempo.
- Se debe poder importar y exportar datasets en CSV, extrayendo las consultas DDL de los headers del csv, o de manera manual.

6.1.4. Modalidad examen

El docente puede cargar conjuntos de datos, uno visible por el alumno, que puede ejecutar queries para probar resultados, y uno oculto y opcional, que sólo devuelve si es el resultado esperado o no.

- Cada examen debe tener un tiempo de ejecución controlado desde el server (no desde el UI). Cada examen puede tener corrección numérica, en base a 10.
- Restricción para que la consulta corra en menos de una cierta cantidad de tiempo e implementación de otras restricciones, como memoria usada o cantidad de tablas consultadas.
- Herramientas para el análisis de query.
- Las queries tienen un puntaje de correctitud, según devuelven la BD/tabla en el estado esperado. Sin embargo, en el caso de que sea un resultado satisfactorio, estará disponible un modo de corrección manual, para análisis de la misma. Esto se pensó de manera tal que se pueda evaluar performance, verificar que no se haya mockeado la respuesta, y prescindir del manejo de cuestiones no automatizables. Además, es posible disponer de una devolución de errores para guiar al usuario.

6.2. Requisitos no funcionales

- Integración con campus - A definir.
- Postgres, Oracle, Redis, algún motor No-SQL.
- Reportes, email y/o descarga de resultados de exámenes, obtención de estadísticas de resultados de exámenes.

6.3. Cambios en el alcance

Durante el desarrollo de Thulium, se fue cumpliendo la planificación del desarrollo según el calendario establecido. Sin embargo, algunos cambios en el roadmap, como el proceso de investigación e implementación de la integración con Campus, provocaron varios cambios en los features finales que Thulium presenta. El motivo principal es debido a la dificultad que implicó esta integración. Los features no implementados se listan a continuación:

- Motores de base de datos no relacionales en base al modelo genérico de un motor de base de datos. Implementación de vistas y otras facilidades que ofrecen algunos motores NoSQL, como por ejemplo MongoDB, cuya implementación estaba estipulada y se dejó de lado debido a cambios en el roadmap.
- Importación de un dataset por parte de un alumno, basado en lo ya implementado, dándole acceso únicamente a alumnos del mismo curso o al alumno que lo creó.
- Estadísticas, reporting o cualquier tipo de análisis complementario respecto a los resultados. Se sugiere la implementación del stack ELK (ElasticSearch, Logstash, Kibana), que permitirá obtener métricas sobre la performance general de los alumnos, que son valiosas para el profesor y la universidad. Además, permitirá obtener métricas sobre el funcionamiento general del sistema.
- Modalidad de autoevaluación, basado en código preexistente. Consiste en preguntas de tipo query, con posibilidad de tener un dataset oculto y ejecutarse fuera de la modalidad de examen.

Nota: Como hubo objetivos que no se alcanzaron debido a dificultades inesperadas de la integración con Campus, se detalló un roadmap tentativo de objetivos a cumplir en un trabajo futuro (Sección 13).

7. Metodología de Desarrollo

7.1. Línea de Tiempo

En la figura 19, se representan los periodos de tiempos tomados para la investigación y el desarrollo durante todo el proyecto. Estos períodos representan cuál fue el principal enfoque del mismo, pero no indican que se haya dedicado exclusivamente a esa tarea. El proyecto empezó en abril de 2018, terminando su desarrollo en julio de 2019.

Los periodos están clasificados según el siguiente criterio:

- Tutor: Etapa inicial, que consistió en encontrar un tutor para el desarrollo del proyecto. Durante el mismo se diagramó y estableció la primer idea del proyecto.
- Análisis: Etapas de investigación y refinamiento de objetivos.
 - Análisis 1: Determinación e investigación de los objetivos iniciales, con resultados principalmente en las secciones anteriores.
 - Análisis 2: Investigación sobre campus.
- Desarrollo: Etapas de implementación y desarrollo del proyecto, siendo descritas en las secciones 9 y 10.

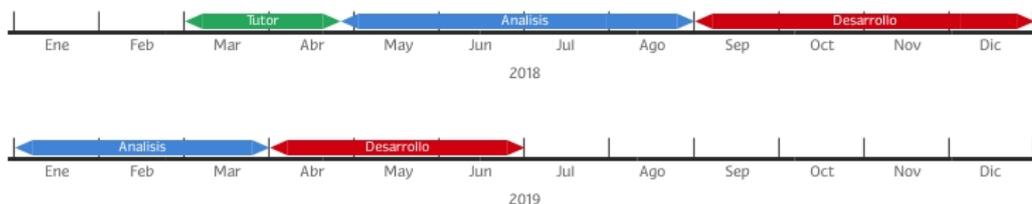


Fig 19: Timeline

Aclaración: Donde se indica que la obtención de resultado o descripción de algún periodo se encuentra en alguna sección, se hace mención a cuándo fue el principal desarrollo de esta sección; ya que todo el proceso fue implementado con integración continua, incluida la documentación y el presente informe.

7.2. Ciclo de Vida

Por las características del proyecto y del equipo se optó por un modelo de ciclo de vida del software en espiral.



Fig 20: Espiral con etapas de trabajo

El desarrollo o modelo en espiral es un enfoque de desarrollo de software que puede ser considerado como una respuesta a los inconvenientes del desarrollo en cascada. El modelo en espiral describe el ciclo de vida de un software por medio de espirales, que se repiten hasta que se puede entregar el producto terminado. El desarrollo en espiral también se conoce como desarrollo o modelo incremental. El producto se trabaja continuamente y las mejoras a menudo tienen lugar en pasos muy pequeños.

Una característica clave del desarrollo en espiral es la minimización de los riesgos en el desarrollo de software, lo que podría resultar en más esfuerzo y un lanzamiento tardío. Estos riesgos son contrarrestados por el enfoque incremental, haciendo primero prototipos, que luego pasan al menos una vez, por las fases de desarrollo de software.

7.3. Gestión de Cambios

Como herramienta de gestión de cambios se utilizó git, en conjunto con un repositorio en Bitbucket donde se aloja el código. Esto permitió el seguimiento de los cambios a través de Issues y Milestones, la revisión del código a través de Pull Requests y facilitó el versionado.

8. Casos de Uso

Para la representación de las funcionalidades desarrolladas en Thulium, se han elaborado algunos casos de uso. Los mismos se encuentran en las secciones subsiguientes, siendo estos:

- Alumno / Profesor quiere ejecutar una query.
- Alumno / Profesor quiere cambiar de motor.
- Alumno / Profesor quiere hacer login.
- Alumno quiere resolver ejercicios.
- Alumno tiene examen.
- Profesor quiere subir un dataset.
- Profesor quiere subir un examen.
- Profesor revisa un examen.

Algunos de estos casos de uso son bastante simples y no tienen diagrama de actividad; aquellos que sí los tienen, se encuentran en el Anexo A.

8.1. CU1: Alumno / Profesor quiere ejecutar una query

1. Descripción

Este caso de uso representa la funcionalidad más básica y principal del proyecto final. A través del sistema, se ejecutan consultas en alguno de los múltiples motores de bases de datos disponibles en el sistema.

2. Contexto

a. Actor

El agente que inicia este caso de uso es un usuario del sistema, puede ser cualquiera de los siguientes roles:

- Alumno: puede consultar y crear bases de datos que tenga disponibles.
- Profesor: puede consultar y crear bases de datos que tenga disponibles.

b. Activación

El caso de uso se inicia con la intención del usuario de ejecutar una consulta.

c. Precondición

Para ejecutar este caso, se debe haber previamente seleccionado el motor correspondiente a la consulta que se desea ejecutar.

d. Postcondición

Al finalizar de ejecutar este caso, se obtendrá el resultado de la consulta ejecutada.

3. Escenario básico

Acción del Actor	Respuesta del Sistema
A1: Usuario escribe consulta.	R1: Highlighting correspondiente de una consulta de bases de datos.
A2: Clickeo en ejecución de consulta.	R2: Devolución de datos correspondientes a la misma.

Tabla 1: Ejecución de query - Escenario básico

4. Escenarios secundarios

a. Flujos Alternativos

Acción del Actor	Respuesta del Sistema
	R2.1: En caso de no poder ejecutar la consulta (falta de permisos, etc), se muestra el mensaje de error.

Tabla 2: Ejecución de query - Flujos alternativos

8.2. CU2: Alumno / Profesor crea nueva pestaña

1. Descripción

Este caso de uso representa una de las funcionalidades más prácticas del proyecto final. En el cliente web del sistema, se puede elegir entre diversos motores de bases de datos sobre los cuales se ejecutan las consultas, esto se logra en modo playground al abrir una nueva pestaña.

2. Contexto

a. Actor

El agente que inicia este caso de uso es un usuario del sistema, puede ser cualquiera de los siguientes roles:

- Alumno: puede elegir entre todos los motores permitidos.
- Profesor: puede elegir entre todos los motores permitidos.

b. Activación

El caso de uso se inicia con la intención del usuario de abrir una nueva pestaña.

c. Precondición

Para ejecutar este caso, se debe cumplir previamente estar autenticado en el sistema.

d. Postcondición

Al finalizar la ejecución de este caso, se obtendrá una nueva pestaña dentro del sistema, que ejecutará las consultas sobre el motor deseado.

3. Escenario básico

Acción del Actor	Respuesta del Sistema
A1: Usuario selecciona el botón de nueva pestaña.	R1: El cliente web muestra un modal, desplegando opciones.
A2: Usuario ingresa nombre, motor y dataset del archivo a crear.	R2: El cliente web muestra la nueva pestaña.

Tabla 3: Creación de pestaña - Escenario básico

4. Escenarios secundarios

a. Flujos Alternativos

Acción del Actor	Respuesta del Sistema
	R2.1: En caso de que el dataset no esté montado internamente, se lo monta en el motor indicado.

Tabla 4: Creación de pestaña - Flujos alternativos

8.3. CU3: Alumno / Profesor quiere hacer login

1. Descripción

Este caso de uso representa una funcionalidad esencial del proyecto final, donde un usuario accede al sistema, a través de un sistema autenticación.

2. Contexto

a. Actor

El agente que inicia este caso de uso es un usuario del sistema, puede ser cualquiera de los siguientes roles:

- Alumno.
- Profesor.

b. Activación

El caso de uso se inicia con la intención del usuario de iniciar sesión.

c. Precondición

Para ejecutar este caso, se debe cumplir previamente que se haya dado permiso a Thulium para autenticarse a través de Campus de ITBA.

d. Postcondición

Al finalizar de ejecutar este caso, el usuario estará logueado y podrá interactuar con el sistema.

3. Escenario básico

Acción del Actor	Respuesta del Sistema
A1: Usuario ingresa a la url de Thulium	R1: El cliente web pide credenciales
A2: Usuario ingresa sus credenciales	R2: El cliente web muestra su home

Tabla 5: Login - Escenario básico

4. Escenarios secundarios

a. Flujos Alternativos

Acción del Actor	Respuesta del Sistema
	R2.1: Ante credenciales incorrectas, se vuelve a pedir las mismas indicando un mensaje de error.

Tabla 6: Login - Flujos alternativos

8.4. CU4: Alumno quiere resolver ejercicios

1. Descripción

Este caso de uso representa la primer funcionalidad principal y simple del proyecto final, donde un Alumno puede resolver un ejercicio dado.

2. Contexto

a. Actor

El agente que inicia este caso de uso es un usuario del sistema con el rol de Alumno.

b. Activación

El caso de uso se inicia con la intención de un alumno de resolver un ejercicio.

c. Precondición

Para ejecutar este caso, se debe cumplir previamente que el docente haya cargado la consulta y datasets para ejecutar para ejecutar la/s query/ies. Además, el alumno debe estar autenticado.

d. Postcondición

Al finalizar de ejecutar este caso, el usuario habrá resuelto el ejercicio propuesto.

3. Escenario básico

Acción del Actor	Respuesta del Sistema
A1: Alumno selecciona el ejercicio a realizar	R1: El sistema le muestra la consigna, y el espacio para correr una query.
A2: El alumno ejecuta una query (Ver Caso de uso N° 1)	R2: El sistema devuelve resultado
A3: El alumno cliquea indicando que quiere enviar la query actual como solución.	R3: El sistema compara la query propuesta contra el dataset propuesto e indica si la respuesta es correcta o no. Mostrando el resultado obtenido.

Tabla 7: Resolución de ejercicios - Escenario básico

4. Escenarios secundarios

a. Flujos Alternativos

Acción del Actor	Respuesta del Sistema
A3.1: Si la query es de tipo True/False, el alumno indica la opción que considera correcta.	R3.1: El sistema indica correctitud de la respuesta seleccionada.
	R3.2: Si hay un dataset oculto, el sistema corre la query contra el mismo y

	solo indica si obtiene el resultado esperado.
--	-----------------------------------------------

Tabla 8: Resolución de ejercicios - Flujos alternativos

8.5. CU5: Alumno resuelve examen

1. Descripción

Este caso de uso representa la segunda funcionalidad principal y más compleja del proyecto final, donde un Alumno debe resolver un examen dado.

2. Contexto

a. Actor

El agente que inicia este caso de uso es un usuario del sistema con el rol de Alumno.

b. Activación

El caso de uso se inicia cuando el periodo de examen inicia.

c. Precondición

Para ejecutar este caso, se debe cumplir previamente que el Profesor haya creado un examen, y el mismo entre en período de actividad.

d. Postcondición

Al finalizar de ejecutar este caso, el alumno habrá concluido un examen.

3. Escenario básico

Acción del Actor	Respuesta del Sistema
A1: Se loguea al sistema.	R1: Muestra el home de Thulium, indicando la existencia de un examen activo en una materia.
A2: Selecciona la materia correspondiente.	R2: Muestra la lista de exámenes.
A3: Selecciona el examen en cuestión.	R3: Muestra el examen
A4: Escribe queries para responder la consigna y las ejecuta.	R4: Muestra resultados (Ver caso de uso 1)
A5: Desarrolla respuesta y envía	R5: Notifica sobre el estado del envío de respuesta

Tabla 9: Alumno resuelve examen - Escenario básico

4. Escenarios secundarios
 - a. Flujos Alternativos

Acción del Actor	Respuesta del Sistema
	R1.1: En caso de venir desde el link al examen desde Campus, lleva directamente al examen
A4.1: Selecciona otro ejercicio	R4.2: Muestra el contenido del mismo
A5.1: No envía respuesta y termina el tiempo de examen.	R5.1: Se envía respuesta parcial.

Tabla 10: Alumno resuelve examen - Flujos alternativos

8.6. CU6: Profesor quiere subir un dataset

1. Descripción

Este caso de uso representa cómo un docente sube como contenido un conjunto de tablas a partir de subidas de datasets.
2. Contexto
 - a. Actor

El agente que inicia este caso de uso es un usuario del sistema con el rol de Profesor.
 - b. Activación

El caso de uso se inicia con la intención de un profesor de subir un dataset a la plataforma.
 - c. Precondición

Para ejecutar este caso, se debe cumplir previamente que el Profesor se haya autenticado en el sistema.
 - d. Postcondición

Al finalizar de ejecutar este caso, habrá un dataset nuevo subido en Thulium.
3. Escenario básico

Acción del Actor	Respuesta del Sistema
A1: El profesor selecciona la opción crear nuevo dataset.	R1: El sistema pregunta si se desea elegir un dataset de un modelo SQL o NoSQL (no implementado).
A2: El profesor elige SQL y elige avanzar al siguiente paso.	R2: Se abre el panel de nuevo dataset.

<p>A3: El profesor elige:</p> <ul style="list-style-type: none"> • Nombre. • Dataset de examen [Si/No]. • Una o más colecciones. <p>El profesor elige pasar al siguiente paso.</p>	<p>R3: Se abre un nuevo panel con una preview de todos los datos del dataset.</p>
<p>A4: El profesor chequea la correctitud de los datos y elige seguir al siguiente paso.</p>	<p>R4: El sistema abre el panel de permisos.</p>
<p>A5: El profesor elige qué permisos van a tener los alumnos, y elige subir el examen.</p>	<p>R5: Hace los últimos procesamientos y crea el dataset.</p>

Tabla 11: Dataset upload - Escenario básico

4. Escenarios secundarios
 - a. Flujos Alternativos

Acción del Actor	Respuesta del Sistema
<p>A3.1: Agregar una colección está compuesto de:</p> <ul style="list-style-type: none"> • Archivo. • Primera fila indica nombre de tablas. <p>Luego, indica que desea procesar los archivos.</p>	<p>R3.1: Chequea el formato y notifica que está todo bien.</p>
<p>A3.1.1: El Profesor puede indicar un archivo de tabla oculta en el caso de ser un dataset de examen.</p>	
	<p>R3.1.2: Si el formato es incorrecto lo indica con un mensaje claro.</p>
<p>A4.1: El profesor desea cambiar un dato, o tipo de dato y lo modifica</p>	

Tabla 12: Dataset upload - Flujos alternativos

8.7. CU7: Profesor quiere subir un examen

1. Descripción

Este caso de uso representa cómo un docente sube como contenido un examen nuevo en base a datasets subidos según caso de uso N°6.

2. Contexto

a. Actor

El agente que inicia este caso de uso es un usuario del sistema con el rol de profesor.

b. Activación

El caso de uso se inicia con la intención de un profesor de subir un examen a la plataforma.

c. Precondición

Para ejecutar este caso, se debe cumplir previamente que el Profesor se haya autenticado en el sistema, y haya subido el dataset pertinente.

d. Postcondición

Al finalizar de ejecutar este caso, habrá un examen nuevo subido en Thulium y en Campus.

3. Escenario básico

Acción del Actor	Respuesta del Sistema
A1: El profesor selecciona la materia correspondiente.	R1: Muestra la lista de exámenes existentes y el botón "New Exam"
A2: El profesor selecciona el botón de nuevo examen	R2: Aparece el formulario de alta del examen
A3: El profesor elige: <ul style="list-style-type: none">• Título del examen• Sección correspondiente• Fecha y hora de inicio y fin• Una o más preguntas Luego selecciona crear	R3: Se crea el examen en Thulium, y se en Campus.

Tabla 13: Alta de un examen - Escenario básico

4. Escenarios secundarios

a. Flujos Alternativos

Acción del Actor	Respuesta del Sistema
A3.1: Agregar una colección está compuesto de: <ul style="list-style-type: none">• Nombre• Motor• Dataset• Consigna• Tipo de pregunta• Respuesta Luego seleccionar salvar.	R3.1: La pregunta a sido almacenada.

A3.1.1: En caso de ser Verdadero o Falso, se debe seleccionar la respuesta correcta.	
A3.1.2: En caso de ser query, se debe seleccionar la query que ocasiona el set de tuplas correcto.	
A3.1.3: En caso de ser multiple choice, se debe seleccionar la respuesta correcta.	

Tabla 14: Alta de un examen - Flujos alternativos

8.8. CU8: Profesor revisa un examen

1. Descripción

Este caso de uso representa cómo un docente evalúa el contenido de un examen.

2. Contexto

a. Actor

El agente que inicia este caso de uso es un usuario del sistema con el rol de Profesor.

b. Activación

El caso de uso se inicia con la intención de un profesor de corregir un examen de la plataforma.

c. Precondición

Para ejecutar este caso, se debe cumplir previamente que el profesor se haya autenticado en el sistema, y haya concluido el periodo de activación del examen.

d. Postcondición

La nota del examen se encontrará publicada en Campus.

3. Escenario básico

Acción del Actor	Respuesta del Sistema
A1: El profesor selecciona la materia del examen en cuestión.	R1: Muestra la lista de exámenes existentes y el botón "New Exam"
A2: El profesor selecciona el examen correspondiente.	R2: Se muestra una tabla, con una columna por cada pregunta.
A3: El docente puede ver las respuestas de autoevaluación corregidas y puede corregir aquellas manuales. Elige el link "Send to Campus".	R3: Se publica la nota en Campus

Tabla 15: Revisión de un examen - Escenario básico

4. Escenarios secundarios
 - a. Flujos Alternativos

Acción del Actor	Respuesta del Sistema
A3.1: En caso de ser query, se puede evaluar manualmente la misma	R3.1: Agrega la valoración de este punto a la consigna final
A3.2: En caso de ser de texto libre, el profesor debe evaluar manualmente la misma	R3.2: Agrega la valoración de este punto a la consigna final

Tabla 16: Revisión de un examen - Flujos alternativos

9. Arquitectura General

En esta sección se detalla cómo fue pensada la arquitectura estructural del servicio implementado. Primero, se describe la funcionalidad requerida de esta estructura y los atributos de calidad en orden prioritario. Luego, describimos dicha arquitectura y cómo verifican los atributos de calidad antes mencionados. Finalmente se revisan los puntos críticos de la misma.

9.1. Funcionalidad Requerida

Si bien los detalles de la funcionalidad están descritos en la sección 10, se listan a continuación las características más importantes del sistema:

- Elección de motor de base de datos.
- Elección de dataset sobre motor de base de datos.
- Poder soportar múltiples roles tales como alumno y profesor.
- Tener una modalidad de evaluación.

9.2. Atributos de Calidad

A continuación, se listan los atributos de calidad que se deben garantizar en el sistema, en orden de importancia: Tolerancia a fallas y disponibilidad, interopeabilidad, performance, escalabilidad y usabilidad. En las subsiguientes secciones describimos estos atributos de calidad en profundidad.

9.2.1. Tolerancia a Fallas + Disponibilidad

Si bien no es necesario que el sistema esté disponible las 24 horas, los 7 días de la semana, el sistema atraviesa periodos donde es crítico que esté disponible. En los periodos donde el sistema se encuentra estresado por la evolución de uno o más exámenes en simultáneo el sistema debe garantizar disponibilidad a través de la tolerancia a fallas. El eje del sistema es la comunicación con múltiples motores de base de datos, es decir, una numerosa fuente de comunicaciones proclive a errores.

9.2.2. Interoperabilidad

Otra característica distinguible del sistema es la capacidad de comunicarse con múltiples motores de base de datos. Por un lado, el sistema debe poder ser extensible en la cantidad de motores de base de datos con los que se comunica, es decir, incorporar un nuevo motor de base de datos debería implicar un proceso simple. Por el otro, los motores de base de datos se encuentran en constante actualización y cambiando sus capacidades. El sistema debe poder responder bien ante estos cambios siempre y cuando dichos cambios sean retrocompatibles.

9.2.3. Performance

Las consultas de base de datos pueden ser operaciones costosas. El sistema debe poder manejar concurrentemente un gran número de consultas de todo tipo sin dejar de estar disponible. Más allá de que eventualmente la consulta sea resuelta por el motor de base de datos, el pre procesamiento y post procesamiento de los datos debe ser eficiente y suficiente veloz para todo tipo y tamaño de respuesta de consultas.

9.2.4. Escalabilidad

El sistema está pensado para admitir grandes cargas en cortos periodos de tiempo. Tiene que estar preparado para aumentar sus capacidades a medida que esto sucede, y esto no debe afectar la performance del sistema.

9.2.5. Usabilidad

A pesar de que los usuarios de esta plataforma van a ser personas técnicas, se apunta a construir una herramienta para el trabajo de los profesores y una herramienta para el aprendizaje de los alumnos. Es por esto que la herramienta debe brindar vistas simples e interactivas para que puedan desarrollar su operatoria.

Es claro que cualquiera de los usuarios puede acceder a una CLI para obtener el mismo resultado. Es una característica clave y uno de los propósitos de la herramienta que esto no suceda.

Aclaración

El hecho de que un atributo de calidad no esté en la lista no significa que no se vaya a considerar en la arquitectura, sino que se considera que tiene una relevancia menor a la de los mencionados y justificados anteriormente.

9.3. Arquitectura candidata

La arquitectura a implementar será del modelo de arquitectura de Hierarchical Layer, subdividida en 3 capas, pensada como una serie de llamadas entre capas contiguas.

Las capas propuestas son Cliente, Servidor y Persistencia, que en vista general posee el siguiente diagrama. La capas pueden estar divididas en distintas subcapas, comulgando con un tipo de arquitectura orientado a microservicios.

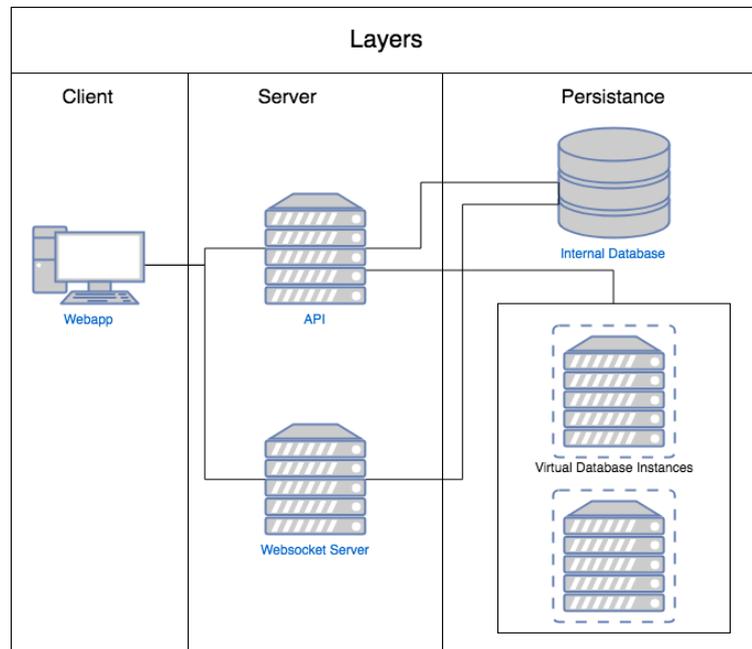


Fig 21: Capas de la arquitectura candidata. La cola de mensajes es parte del Websocket server.

9.3.1 Cliente

La capa cliente está constituida por el software de la webapp que se ejecuta en los navegadores de los usuarios.

9.3.1.1. Webapp

El desarrollo de la webapp se realizará con el framework [React](#) [13] y las herramientas de su entorno. La decisión se tomó no solo basada en la cantidad de proyectos que existen en la comunidad actualmente sino que también debido a la experiencia que tiene el equipo de desarrollo en la tecnología, que permitirá tener una buena velocidad de desarrollo desde el principio.

Se utilizará React como framework para la programación de las vistas y [Redux](#) [14] para el manejo del estado de la aplicación y control de las acciones. Además, se utilizará HTTPS para comunicarse con el servidor, manejando usuarios y sesiones.

9.3.2. Servidor

9.3.2.1. Load Balancer

Para aprovechar el procesamiento de múltiples instancias del servidor es posible utilizar load balancers que mediante el algoritmo Round Robin repartirán la carga del sistema entre todas las réplicas de los servidores. Para nuestro sistema no es necesario ningún algoritmo de IP hashing para garantizar sticky sessions.

Por default no se accionará ningún load balancer dado que el volumen de usuarios esperado probablemente no lo amerite. De todos modos, el sistema es compatible con Kubernetes (k8s), cuyo funcionamiento e integración con el proyecto será explicado más adelante. Esto permitirá la implementación de load balancers escalables en cuestión de segundos.

9.3.2.2. Core API

La Core API es un componente del sistema que precisa de una alta disponibilidad para procesar la cantidad de requests que tomará el sistema. Resulta imprescindible que la tecnología a utilizar en este servidor sea no bloqueante, es decir, no debe dejar de atender requests de otros clientes por el hecho de estar procesando. En otras palabras, la comunicación con la capa de persistencia debe ser asíncrona. Es por esto que escogimos NodeJS [15] como framework para el servidor, pues otros frameworks son naturalmente sincrónicos y se bloquean ante el IO, mientras que Node es por esencia no bloqueante y event-driven I/O, siendo además liviano y eficiente para operaciones en las que se recibe volúmenes de datos intensivamente en tiempo real. Este framework ayuda a crear aplicaciones que manejen un gran número de conexiones simultáneas con gran rendimiento, lo que aporta escalabilidad.

La Core API declara endpoints REST para los datasets, consultas, autenticación e instancias de los motores.

9.3.2.3. Websocket API

La Websocket API se utilizara para alivianar la carga de la Core API en términos de transmisión de datos. Esto es un recaudo para queries que tomen mucho tiempo o transmiten mucha información. Además funciona como una subarquitectura Publisher/Subscriber para las consultas, liberando al server de esperar consultas largas y procesar timeouts.

El motivo para elegir esta arquitectura fue no sólo evitar comprometer el servicio de API, sino que también el hecho de que las queries no necesariamente tienen que ser ejecutadas en tiempo real. A partir de esta subarquitectura se las ejecuta a través del servidor de background jobs.

El proceso se desempeña de la siguiente manera: un usuario autenticado envía un request a al servicio de API. El servidor API lo valida, realiza un pre-análisis del request y lo reenvía al servicio de websockets. Dicho servicio, le devuelve un websocket a API que será reenviado al cliente, en donde deberá escuchar la respuesta en el momento en el que su request haya sido procesado.

9.3.2.4. Jobs

Jobs es un servicio que está constantemente esperando nuevas tareas de ejecución de queries. Las queries son ejecutadas a modo de background jobs. Cada vez que el servidor recibe un request pidiendo que se procese una query, se crea un job y se lo encola. La cola funciona en memoria y la tecnología utilizada para el procesamiento es [ZeroMQ](#) [16]. Cada vez que el servidor detecta un nuevo job, lo desencola, lo procesa y cuando obtiene el resultado, responde a

través de un websocket. El servidor a va procesar jobs continuamente hasta que la cola esté vacía.

El SLA (Service Level Agreement) establecido para queries no es tiempo real, sino que es best effort. Los motivos por los cuales se decidió implementar un sistema que encole queries en vez de uno que resuelva queries en tiempo real fueron los siguientes:

- Es más defensivo en términos de arquitectura, ya que se evita ejecutar varias queries a la vez en la base de datos.
- Es más barato, es posible atender varios pedidos sin necesidad de escalar.
- Es más fácil implementar retries, volviendo a encolar los pedidos que fallen.
- Se puede lograr una performance parecida a real-time simplemente escalando el pool de workers de Jobs.
- En términos de mantenibilidad, es posible implementar un sistema que escale en base al tamaño de la cola en un futuro.

9.3.3. Base de datos interna

La base de datos interna guardará toda la información de los usuarios, queries, sesiones y otros modelos relevantes. Se distingue de las bases externas ya que estas ejecutan queries y guardan información volátil. Se decidió utilizar una base de datos NoSQL por la amplia cantidad de eventos que pueden llegar a registrarse. No solamente se trata del volumen de información sino que también del formato que la misma posee. La naturaleza de la información de un evento reúne algunas características esenciales como un timestamp de cuándo sucedió, el tipo de evento del cual se trata y alguna información adicional que permita identificar al evento. Además es imprescindible agregar información adicional sobre el contexto de cada evento, y debido a la variabilidad que existe en la forma de los eventos, no es conveniente estar limitado a un esquema SQL. En otras palabras, las bases de datos NoSQL se adecúan mejor al comportamiento planteado y esperado por parte de Core API.

Entre las opciones disponibles en el mercado, se encuentra [MongoDB](#) [17], [Cassandra](#) [18], [CouchDB](#)[19], entre muchas otras. Se decidió que MongoDB es el motor de base de datos NoSQL que se adapta mejor a las necesidades del proyecto. En primera instancia, en comparación con Cassandra, MongoDB no impone ninguna restricción sobre el dinamismo de los campos a utilizar.

MongoDB posee librerías como [Mongoose](#) [20] que resuelven problemas como la subdivisión de los documentos en colecciones. En nuestro caso, tendríamos distintas colecciones como usuarios, sesiones, etc; que de otra forma deberíamos enmarcarlas en un tipo de documento. Esta falta de separación, no sólo afecta la programación sino que también repercute en la performance de la base de datos en cuestiones como índices, ya que dependiendo del contenido del documento, el formato de la información puede ser completamente distinto.

9.3.4. Dockerizing

Los distintos microservicios van a correr en contenedores de [Docker](#) [21]. A la hora de generar las imágenes, se crea un virtual environment en la que se copian todos los archivos correspondientes al microservicio. Para la ejecución simplemente se le indica a Docker un

comando para ejecutar.

Si un microservicio consta de más de un subservicio, cada microservicio corre en un contenedor separado. Todos los contenedores corren en una misma network virtual y pueden comunicarse entre sí.

Docker no instala un kernel desde cero, sino que simula un kernel utilizando la máquina host. Además, posee otros features como caché a la hora de construir una imagen. Esto permite:

- Realizar una rápida instalación de cualquier versión de un sistema operativo para utilizar en el contenedor.
- Realizar un desarrollo independiente de un kernel particular.
- Realizar migraciones/actualizaciones a una velocidad acelerada.

9.3.5. Kubernetes (k8s)

[Kubernetes](#) [22] es un orquestador de contenedores pensado para facilitar el deployment de las aplicaciones. Podemos decir que el codebase es “Kubernetes compatible” ya que todo lo que corre en un contenedor es compatible (Docker).

La unidad mínima en k8s es el pod y está compuesta de uno o más contenedores que corren en una misma network. El layout pensado para cada microservicio es:

- Un deployment que ejecute uno o más contenedores.
- Un service que permita resolver el nombre del servicio a la hora de querer comunicarse con otro servicio.
- Un horizontal pod autoscaler que permita escalar en base al uso de CPU y así garantizar disponibilidad.

Las motivaciones para utilizar kubernetes son las siguientes:

- Health check y self healing: Kubernetes utiliza readiness probe y health probe. Health probe realiza un request a cada pod en el deployment para revisar que esté vivo. Si el pod no responde, se mata y se vuelve a instanciar (self healing). Readiness probe, por su parte, no lanza la versión de deployment hasta que el contenedor no se ejecuta correctamente.
- Deployments seguros y revisioning: El hecho de tener readiness probe implica no lanzar una nueva revisión del deployment hasta que esta no se ha inicializado correctamente. Esto significa que un deployment no mata el servicio (ya que k8s deja corriendo la revisión anterior), con lo cual, los deployments son confiables.
- Autoscaling: Al tener horizontal pod autoscalers, es posible agrandar el cluster de computadoras en donde corren los servidores en base al uso de CPU. Se setea un “target CPU usage” y si el uso de CPU por parte del servicio es mayor, se crean nuevos pods. Si los pods no entran en las computadoras existentes, se agregan nuevas computadoras al cluster. Cuando el uso de CPU disminuye, ocurre el efecto inverso, se eliminan pods y se reagrupan en menos computadoras.

Nota: En caso de no disponer de recursos para utilizar un servicio de Kubernetes en la nube, es posible lograr este mismo efecto teniendo computadoras físicas y usando tecnologías como OpenStack.

9.4. Verificando atributos de calidad

9.4.1. Tolerancia a Fallas + Disponibilidad

Se garantiza la disponibilidad en momentos clave, ya que se eliminaron varios puntos de falla, por ejemplo al utilizar un modelo que procesa queries a través de jobs. Hay muy bajo riesgo de quedarse sin suministro eléctrico, ya que el servicio se encontrará hospedado en un data center tier 3. Todos los servidores y bases de datos se encuentran replicados. Además los servidores reciben los request de un Load Balancer para que la carga sea lo más uniformemente distribuida posible entre ellos, garantizando conectividad.

9.4.2. Interoperabilidad

A partir de la implementación de Managers de DB y las correspondientes librerías para su manejo, se puede implementar cualquier tipo de DB.

9.4.3. Performance

El hecho de utilizar una base de datos interna NoSQL, en particular MongoDB garantiza la performance en este caso debido a que, por ejemplo, los objetos serán insertados en la base de datos sin validar tipos, en comparación con una base de datos relacional. La validación de tipos se realizará previa a la inserción. Además, el load balancer distribuye la carga en las múltiples instancias para alivianar la carga y la decisión de utilizar un framework que realice las operaciones de IO de modo no bloqueante permite aprovechar cada ciclo de CPU a procesar pedidos y sin esperar. A su vez, obtener el costo como preprocesamiento de una consulta y utilizar colas de prioridades en base a este costo, nos permite manejar cualquier tipo de consulta según la carga del servidor.

9.4.4. Escalabilidad

Se puede escalar horizontalmente debido a que contamos con load balancers, que distribuyen la carga entre todos los servidores que se vayan agregando. En cuanto a la escalabilidad en las bases de datos; la DB interna es fácilmente escalable pues es sencillo agregar más nodos al cluster NoSQL. El servidor central se encarga de distribuir la información entre las mismas; mientras que las DB de consulta pueden escalar tanto verticalmente; como horizontalmente utilizando un nodo como master.

9.4.5. Usabilidad

Para lograr usabilidad se respetaran las prácticas de UCD (Diseño Centrado en Usuario) utilizando botones con iconos representativos y de colores que identifiquen la acción a realizar en los reportes e indicadores apropiados. Por ejemplo, cuando se espera la respuesta de una consulta, se informa al usuario a través de un spinner de la misma manera en que Google Drive

lo hace al comprimir un archivo. Además, se utilizan otras técnicas de HCI para lograr que quien use el sistema logre sentirse cómodo con el mismo. Un ejemplo de esto es que el cliente tendrá la sintaxis del código con highlighting de las palabras reservadas en SQL.

9.5. Puntos críticos

La API es un claro punto crítico donde van a llegar la mayoría de requests de los usuarios. Si bien este servicio tiene poco procesamiento, en caso de estar caído no se cumple la función primordial del sistema, que es registrar eventos. Por este motivo, es recomendable utilizar múltiples réplicas de este proceso.

El hecho de que la base de datos interna esté sobrecargada es otro claro SPOF (Single Point of Failure), ya que de no tener la información de los usuarios disponible, estos no podrán iniciar sesión.

A su vez, las bases de datos que ejecutan las consultas pueden llegar a fallar. Esto no implicaría un fallo total de la aplicación, sino que simplemente dejaría inutilizable temporalmente el menú de ejecución de consultas para el motor que falle. De todas formas, esto se previene utilizando Sharding y Replica Sets. Si un nodo del replica set fallase, habría otro con la misma información. Si un nodo master fallase, entonces un slave sería promovido a master. Para perder toda la información de un shard deberían caerse todos los servidores dentro del réplica set.

10. Implementación

En esta sección se describen las distintas vistas y definiciones en el desarrollo del TPF; siendo estos: los roles conceptuales, los casos de uso, el diseño de la arquitectura, el modelado y el diseño del código en los módulos.

10.1. Roles

Conceptualmente se han definido roles según los permisos, accesos y vistas a los cuales se tendrá acceso dentro de Thulium.

- Alumno: Es aquel que tendrá mayor contacto con Thulium. Podrá ejecutar consultas en múltiples motores de bases de datos, realizar ejercicios y evaluaciones.
- Profesor: Es la autoridad de control con respecto a los exámenes y ejercicios. Consecuentemente, podrá crear ambos elementos y configurarlos dentro de Thulium.
- Administrador: Es la autoridad respecto de la creación y asignación de los roles de los usuarios.

10.2. Modelado

En esta sección se describe cada uno de los modelos implementados, como así también los campos que los componen.

Nota: El tipo Mixed significa que el atributo puede ser de cualquier tipo.

10.2.1. Usuarios

El modelo de los usuarios está basado en los siguientes campos:

- `first_name` y `last_name` (String): Nombre y Apellido.
- `bb_*`: Campos de representación para la conexión con blackboard, siendo los siguientes:
 - `bb_id` (String).
 - `bb_username` (String).
 - `bb_access_token` (String).
 - `bb_refresh_token` (String).
 - `bb_token_expiry` (Date).

Además, por retrocompatibilidad a la no integración con campus, se tiene los siguientes campos:

- `email` (String): campo por el cual se identifica al usuario, actualmente este dato se obtiene de blackboard.
- `hash` y `salt` (String): campos derivados de seguridad para el almacenamiento de contraseñas.
- `role` (String): campo para identificar a usuarios y profesores, actualmente este comportamiento varía según el rol que provee Blackboard.

10.2.2. Motor

El modelo de un motor de base de datos está compuesto por los siguientes campos:

- `title` (String): Título de la tabla.
- `exam` (Boolean): Indica si el motor está habilitado para uso.

10.2.3. Dataset

El modelo de datasets se compone de los siguientes campos:

- **`title`** (String): Título del dataset.
- **`publisher`** (Usuario): Usuario que ha publicado este dataset.
- **`paradigm`** (String): Indica si es SQL o NoSQL.
- **`exam`** (Boolean): Indica si es un dataset de examen.
- **`reduced`** (Dataset): Si es un dataset de examen, y es el no oculto; apunta a su correspondiente dataset oculto.

- **full** (Dataset): Si es un dataset de examen, y es el oculto; apunta a su correspondiente dataset.
- **default** (Boolean): Indica si es el dataset default, en el resto de los datasets este campo no aparece.
- **actions** (Mapa de Boolean): Este mapa contiene los permisos como clave e indica si se tiene ese permiso.

10.2.4. Dataset Ítem

Este modelo es una representación de una tabla, que puede ser modelada como un ítem de un dataset. Sus campos son:

- **title** (String): Título de la tabla
- **dataset** (Dataset): Dataset al que pertenece.
- **headers** (Mapa de String): Este mapa contiene el nombre de las columnas como claves, y el tipo como valor, los mismos pueden ser: Integer, String o Float.

10.2.5. Entrada de dataset

Este modelo es nuestra representación de una fila dentro de una tabla, que puede ser modelada como un subítem de un dataset item. Sus campos son:

- **dataset** (Dataset): Dataset al que pertenece.
- **dataset_item** (Dataset Item): Dataset Ítem al que pertenece.
- **index** (Number): Al ser nuestra base de datos interna NoSQL, no garantiza el orden de inserción, por lo que agregamos el mismo.
- **data** (Array de tipo Mixed): Conjunto de los datos de la fila.

10.2.6. Pregunta de examen

Cada pregunta está compuesta por los siguientes campos:

- **content** (String): Texto de la pregunta.
- **type** (String): Tipo de la pregunta, su valor auto-descriptivo puede ser:
 - true-false.
 - multiple-choice.
 - written-answer.
 - query-response.
- **dataset** (Dataset): Dataset de la pregunta.
- **engine** (Motor): Motor de la pregunta
- **correct_answer** (Mixed): Respuesta correcta, que varía según el tipo.
- **options** (Mixed): Opciones de respuesta, que varía según el tipo.

10.2.7. Examen

Los campos del modelo de los exámenes son los siguientes:

- **title** (String): Título del examen.
- **questions** (Array de Preguntas): Conjunto de preguntas del examen.

- **owner** (Usuario): Usuario que creó este examen.
- **contentId** (String): Campo requerido por campus para identificar el ítem que aparece en la sección “Material Didáctico” dentro de Campus.
- **gradeColumnId** (String): Campo requerido por campus para identificar el examen dentro de Campus.

10.2.8. File

Este modelo representa las pestañas del usuario, compuesto por los siguientes campos:

- **title** (String): Título de la pestaña.
- **content** (String): Contenido de la pestaña.
- **owner** (Usuario): Usuario dueño de este File.
- **engine** (Motor): Motor a usar en esta pestaña.
- **dataset** (Dataset): Dataset de esta pestaña.
- **exam** (Examen): Examen de esta pestaña. Puede ser indefinido si no es de ningún examen.
- **session** (Sesión): Sesión al que pertenece la pestaña.

10.2.9. Respuesta de examen

Este modelo representa cada respuesta de un alumno a una pregunta en un examen. Sus campos son:

- **exam** (Examen): Examen al que pertenece. Puede ser indefinido si no es de ningún examen.
- **user** (Usuario): Usuario que responde esta respuesta.
- **file** (File): File de esta pregunta.
- **question** (Pregunta): Preguntas del examen a la que se está respondiendo.
- **response** (String): Texto que representa la respuesta del alumno.
- **review** (Boolean): Indica la corrección del docente.
- **hint** (Boolean): Indica si esta respuesta puede tener una ayuda del sistema.

10.2.10. Sesión

Los campos del modelo de las sesiones concurrentes son:

- **owner** (Usuario): Usuario de esta sesión.
- **files** (Array de File): Conjunto de Files del modo playground.
- **exam** (Examen): Si está definido, es el examen de esta sesión.
- **examFiles** (Array de File): Conjunto de Files del modo examen.
- **ws** (Array de String): Conjunto de identificador de websockets de esta sesión.

10.2.11. Instancia de dataset

Este modelo representa cuales datasets están montados en los motores de base de datos, compuesto por los siguientes campos:

- **owner** (Usuario): Usuario que tiene acceso a esta instancia.
- **dataset** (Dataset): Dataset al que instancia.
- **exam** (Examen): Examen al que pertenece. Puede ser indefinido si no es de ningún examen.
- **engine** (Motor): Motor que usa esta instancia.
- **tables** (Mapa de String): Esto hace un mapeo del nombre original de la tabla a como lo llamamos en nuestro caso.

10.2.12. Jobs

Los jobs están modelados según los siguientes campos:

- **status** (String): El estado en el que se encuentra el job, puede ser cualquiera de los siguientes:
 - *pending*: Creado.
 - *received*: Recibido por la cola de mensajes.
 - *completed*: Terminado con éxito.
 - *failed*: Terminado con alguna falla.
- **key** (String): Tipo de trabajo a ser procesado.
- **params** (Mixed): Parámetros que varían según el tipo **key**.
- **scope** (Array de String): Conjunto de identificadores de websocket a quienes se debe notificar el resultado de este job.
- **received** (Date): En que momento fue recibido.
- **completed** (Date): En que momento fue completado.
- **failed** (Date): En que momento fallo.
- **error** (Mixed): Error a ser interpretado por otro componente.

10.3. Diseño de Módulos

En esta sección se detallan los diferentes módulos desarrollados para la implementación de Thulium. El mismo, como fue mencionado en la sección 9.3.2.2, fue desarrollado en NodeJS como lenguaje de backend, debido a su fácil manejo de asincronicidad y su event loop. Cada módulo representa un paquete de Node, siendo estos los módulos disponibles:

- API.
- Assets.
- Base.
- Config.
- Internal.
- Jobs.
- Storage.
- WebApp.
- WebSocket.

10.3.1. Módulos estáticos

Existen módulos que contienen archivos estáticos y fueron detallados en varias secciones de código, para una correcta modularización de código, estos son:

1. **Assets:** Contiene archivos estáticos que son utilizados en todo el código, por ejemplo los logos.
2. **Configuración:** Contiene dos JSON, **app.json** y **app.secure.json**. El primero contiene las configuraciones por default, exceptuando contraseñas. Estas son públicas y se encuentran en el repositorio del proyecto. El segundo contiene las configuraciones seguras, como credenciales, puertos o urls; que a su vez pisan las propuestas en app.json, y no es seguro que sean públicas. En el anexo B se encuentran los archivos utilizados para la etapa de desarrollo.

10.3.2. Base

Este módulo se encarga de leer los archivos de configuración, generar una configuración del sistema y hacerla disponible al resto de los módulos. A su vez, provee métodos genéricos útiles en el resto de la aplicación según en qué ambiente se está ejecutando el código, es decir, desarrollo, staging, producción, etc.

10.3.3. API

Este módulo funciona como un servidor, el cual responde a cada solicitud externa como una API REST versionada. Esto significa dos cosas: primero, que al ser una API REST, responde según los métodos HTTP, en formato JSON; y segundo, que al ser versionada, puede haber cambios a futuro y proveer retrocompatibilidad. El componente principal es [ExpressJS](#) [23], que es el front-controller que gestiona todos los requests que llegan al servidor.

El proceso de inicio del servidor de esta API es el siguiente:

1. Conexión a la base interna de Mongo y configuración según los módulos Configuración y Base (secciones 9.3.3, 10.3.1 y 10.3.2, respectivamente).
2. Proceso de Bootstrapping:
 - a. Configuración según los módulos ya mencionados. Se es administrador, ya que es una buena práctica y permite realizar los siguientes pasos.
 - b. Levantar los motores bases de datos.
 - c. Crear un dataset lógico, donde se es dueño al ser el administrador.
3. Proceso de servidor web
 - a. Levanta el servidor web, utilizando express.
 - b. Crea todas las rutas base del servidor, que están disponibles en el anexo C.

Para el control flow del API utilizamos la librería [async](#) [24], que nos permite poder manejar el flujo de toda la app a pesar de haber llamadas remotas asincrónicas.

10.3.4 Internal

Este módulo se encarga de implementar internamente la abstracción de los modelos propuestos, como por ejemplo exámenes o datasets, y su comportamiento. Para ello se utiliza mongoose, el ORM de Mongo, que está basado en programación orientada a aspectos (AOP), la cual establece que el comportamiento del programa, en nuestro caso el comportamiento de los modelos, están basados en los aspectos de lo último ejecutado. Este módulo expone al resto los objetos con los cuales puede interactuar.

Para la mayoría de los modelos, este módulo expone su alta, baja, modificación y búsqueda. En el caso de los File y Sessions posee un comportamiento particular, ya que a través del AOP, si un usuario inicia una nueva sesión, le da todos los archivos que ya tiene abiertos; y permite guardar temporalmente los activos cuando entra en modo examen.

10.3.5. Jobs

Este módulo se encarga de recibir solicitudes de tareas a través de una cola de mensajes, ejecutarlas y pasar la respuesta. Esto permite tener un servidor distinto de aquel que se encarga de gestionar la comunicación con el cliente. Esta implementación tiene por objetivo no demorar comunicación entre el cliente y el servidor, ya que la misma funciona sobre el protocolo HTTP, y es necesario tolerar timeouts elevados para poder procesar las queries de forma asíncrona. Una vez ejecutado el job, se puede notificar al cliente el resultado del mismo a través de un web sockets.

ZeroMQ es la cola de mensajes seleccionada para la implementación. La misma, funciona de manera simple a través de Sockets, de 3 maneras diferentes según el tipo de mensajes:

- Request / Response: Funciona en base a una solicitud y respuesta inmediata. Fue implementada para la comunicación hacia el server de Jobs, pero no fue utilizada.
- Push / Pull: El emisor envía la información cuando desea, mientras que el receptor hace pull de la información cuando le es posible. Fue implementada para la comunicación hacia el server.
- Publisher / Subscriber: El emisor avisa que va a publicar información sobre un tópico, el receptor avisa que desea recibir información sobre ese tópico. Se implementó para la emisión de información de cada Job.

El contenido de la información desde o hacia este servidor está en formato JSON. En el caso de que la información recibida no cumpla con el formato esperado, el mensaje será descartado.

El formato de cada mensaje está compuesto por:

- Identificador (Id): Cada trabajo a procesar se encuentra almacenado en la base de datos interna, por lo que con obtener el identificador, es posible buscarla, obtener los parámetros de la misma y ejecutar la tarea. A su vez, es posible actualizar los datos de la misma, permitiendo en un futuro poder sacar estadísticas.
- Job: Describe el tipo de solicitud de la cual se trata. Si no se encuentra, se descarta el mensaje. Los tipos de trabajos son:

- Crear una instancia de un dataset: Busca los parámetros, busca el dataset y le da la instrucción al módulo de storage de que cree la instancia.
- Ejecutar una query: Se busca la instancia del motor y se le pide al módulo de storage que ejecute la query a través del Executor.
- Comparar con el dataset oculto: Busca el examen, luego controla que el tipo sea el correcto (Query / Response), crea ambos Dataset instance, ejecuta la query en ambos en paralelo, y ejecuta la rutina de comparación con los resultados. Guarda en la respuesta en un hint para luego informar que está bien o mal.

Finalmente, cuando la tarea termina su ejecución, debe avisar a quien corresponda que la misma ha concluido.

10.3.6. Storage

Cada implementación de un motor soportado por Thulium se encuentra en este módulo. Estos deben implementar algunos métodos mínimos basados en un contrato, siendo los expuestos estos:

Tipos:

```

type QueryCallback
  alias function (Error, QueryResults): Any
type CreateDatasetOptions
  shape { items: [DatasetItem], nonce?: Any }
type CreateDatasetResult
  shape { engine: String, tables: Map(String => String) }
type CreateDatasetCallback
  alias function (Error, CreateDatasetResult): Any

```

Contrato:

```

function config(c: Configuration): Any
function id(): String
function query(q: String, cb: QueryCallback): Any
function createDataset(opts: CreateDatasetOptions, cb:
CreateDatasetCallback): Any

```

Tabla 17 - Contrato para implementar un motor

10.3.7. Webapp

Este módulo contiene toda la aplicación web, implementada en React y JavaScript. La misma permite ejecutar localmente en modo desarrollo y actualizarse en tiempo real, como así también compilar y generar los archivos necesarios para ejecutar y servir en un servidor externo. Este proceso se realiza utilizando [Webpack](#) [25].

Se hace uso de la técnica code splitting, que permite separar la aplicación en partes y no tener que cargar el código de un Modal hasta que no haya sido clickeado el botón pertinente, lo que resulta en que se haya cargado solo el código necesario.

Se realiza un control de si el usuario está autenticado o no, y luego se chequea la existencia del token en el localStorage o en las cookies como así también el estado de autorización dentro de la aplicación. En el caso de que el token exista, se busca el perfil a partir de la API de Blackboard.

Como el usuario ya se encuentra autorizado, se inicia un nuevo proceso de bootstrapping:

- Caso playground: Empieza el booting/loading, a partir de Redux se maneja del estado de la aplicación, donde el usuario puede ejecutar acciones causando nuevas acciones, o cambiando el estado global de la misma, impactando en todos los componentes que están suscritos a estos cambios. Por ejemplo: el cambio de estar logeado a no logeado, implica mostrar la pantalla de login.

En paralelo se buscan los motores, y a través de [GraphQL](#) [26] se consigue el membership (tabla que relaciona a una persona con un curso) y luego se buscan los cursos del alumno. Finalmente, se obtienen todas las columnas (exámenes); la sesión (archivos abiertos, permitiendo abrir en otra computadora los mismos archivos); y los datasets.

Una vez obtenida la sesión y el web token, se abre el websocket, y automáticamente se conecta (ya que el mismo se encuentra autenticado).

Existe un handler de respuestas que se encarga de manejar tanto la completitud como el fallo de una query a través de la persistencia de esta información dentro del estado. Cuando termina de procesarse una solicitud, el resultado se muestra en la UI.

- Caso examen: Se trata de buscar el examen, los motores y los datasets a partir de la información obtenida desde la llamada al endpoint correspondiente enviando el id del examen en la query de la URL. Se da la instrucción de obtener los datos de el examen correspondiente, y cambiar al modo examen.

Luego, se conecta al websocket, y se carga el examen. Este proceso está dividido en las siguientes subtareas:

- Buscar el examen al servidor.
- Buscar la session, con motivo de obtener los datos faltantes.
- Realizar un swap de archivos, guardando los archivos que se encuentren abiertos y empezar el proceso de creación de los correspondientes al examen.
- Buscar con el parámetro *expect*, cuáles instancias deben ser creadas.
- Realizar una prueba del websocket para corroborar si está disponible.

Cuando estos pasos hayan finalizado, significa que se está esperando nuevos eventos ya que el bootstrapping ha finalizado.

10.3.8. WebSocket

Este módulo se encarga de proveer un servidor de websockets, a partir del cual se establece una comunicación con cliente para solicitudes de timeouts grandes, para los cuales HTTP no está pensado.

Para la comunicación de respuestas, cada websocket tiene un identificador, y cada mensaje que recibe el servidor de websockets contiene un campo *scope*, el cual indica por cuáles

websockets se debe notificar una respuesta. Si este campo se encuentra vacío, no notifica a nadie. Si se dispone de 1 o más identificadores, se les notifica a todos.

El proceso de verificación de clientes consiste en:

1. Corroborar el campo origin por cuestiones de seguridad.
2. Validar el token, si no es válido, se descarta.
3. Busca usuario y session y guardarlo en la instancia correspondiente.

Los mensajes se encuentran en formato JSON. Estos vienen contenidos en texto plano, por lo que deben ser decodificados.

Los Topic a los que los mismos se suscriben son:

- *create dataset instance*
- *create dataset instance:error*
- *execute query*
- *execute query:error*
- *compare with reduced*
- *compare with reduced:error*

10.4. Seguridad

Para garantizar la seguridad de la plataforma se tomaron medidas en todas las capas. Las comunicaciones entre el cliente y la API y el WebSocket Server se realizan mediante canales autenticados por HTTPS, de modo tal que todas las comunicaciones se encuentran encriptadas. Además, ambas, la API y el WebSocket Server sólo aceptan conexiones de orígenes validados de acuerdo al ambiente de desarrollo, por último. Ambas, la API y el WebSocket Server validan sus conexiones mediante JWT con un secreto compartido entre ambos servidores. Los servidores aceptan conexiones si un JWT valido se encuentra en el header de HTTP *Authorization*, como header HTTP *X-Access-Token* o si se encuentra como parametro de la query como *token* (<https://thulium.xyz?token=XXX>).

10.5. API

10.5.1. Autenticación e integración con Blackboard

El servidor en su ruta */auth* presenta los endpoints disponibles para la autenticación de usuarios. Permite autenticación por usuario y contraseña, donde si el usuario presenta credenciales válidas entonces estas se intercambian por un JWT.

El servidor también permite delegar la validez de un usuario a Blackboard, es decir, si un usuario es válido en Blackboard entonces también es válido en Thulium.

El servidor API de Thulium es un intermediario entre el cliente web y Blackboard, ya que habilita a estos a comunicarse entre sí. Para esto, Blackboard ofrece un proceso de autenticación

a través de OAuth2 muy estricto, que se concluye una vez que el usuario se loguea y el mismo haya dado permiso desde Blackboard, obteniéndose así un token para poder enviar información. En caso de ser necesario, este token es renovable.

En este proceso se dispone de un endpoint en la API que recibe un código (correspondiente a aquel del flujo OAuth2) con el cual se hace un llamado a Blackboard con ese código y otros parámetros del flujo OAuth2 para intercambiar ese código por un token para poder interactuar con la API REST de Blackboard. Una vez que se recibe el token de acceso a Blackboard, este se almacena y se utiliza para pedir a Blackboard los detalles del usuario autenticado. Con esta información disponible, se almacena en la base de datos interna y luego se genera un token validado por Thulium para entregarle al cliente para que pueda comunicarse con la API de Thulium.

De esta manera, delegando la autenticación e identificación a Blackboard, se logró ser un intermediario entre ambos.

10.5.2. Proxy para Blackboard

La API cuenta con un endpoint */learn*. Este endpoint se utiliza para comunicarse con la API REST de Blackboard. Este endpoint intercepta las llamadas a la API de Blackboard y realiza, a partir de las credenciales presentadas, un intercambio de tokens entre el token presentado ante la API de Thulium y el token guardado anteriormente para realizar pedidos ante Blackboard.

La intercepción permite agregar información a los endpoints con los que se le realiza una consulta a Blackboard, por ejemplo, cuando se desea crear un examen en Campus, se agregan un link a Thulium referenciando al examen correspondiente.

Por sobre los detalles de implementación, el endpoint funciona como un reverse proxy a la API de Blackboard. Es decir, por ejemplo, los requests que llegan a la API de Thulium como *GET /core/v1/learn/v1/courses*, terminan respondiendo lo mismo que respondería */learn/v1/courses* de la API de Blackboard.

10.5.3. GraphQL

La API de Blackboard está altamente desacoplada y para cargar la información necesaria para la vista del cliente es necesario realizar muchos pedidos a la API de Blackboard. Como por una cuestión de seguridad el cliente no debe conocer el token de acceso a la API de Blackboard, es necesario que el pedido pase por los servidores de Thulium. Aún así existe un problema que es que el cliente debe realizar muchos requests, que se traduce a muchos manejos de errores y latencia.

Optamos por desarrollar una interfaz en GraphQL para reducir todos esos pedidos en uno sólo a la API de Thulium. El hecho de realizar un solo request a la API de Thulium y múltiples a la API de Blackboard es una decisión conveniente en términos de transferencia y latencia, ya que Blackboard y Thulium están alojados en AWS. De esta manera, todos los requests se realizan desde una menor distancia física. Esto, además, permite manejar errores. Por ejemplo, si algo devuelve error porque no existe en Campus, se maneja ese error almacenando un valor acorde.

10.5.3.1. Entidades de GraphQL

Se lista los tipos utilizados por GraphQL para mapear las entidades de Blackboard en Thulium:

- **AdaptiveRelease**: Concepto de **desde** y **hasta**, para la disponibilidad de un contenido, sus campos son:
 - **start** (String)
 - **end** (String)
- **Availability**: Concepto de disponibilidad de un contenido, sus campos son:
 - **available** (String)
 - **allowGuests** (Boolean): Acceso a invitados
 - **adaptiveRelease** (AdaptiveRelease): Tiempo de disponibilidad
- **Content**: Contenido de Blackboard:
 - **parentId** (String): Identificador de quien contiene a este elemento.
 - **title** (String): Título de elemento.
 - **body** (String): Contenido en sí mismo.
 - **description** (String): Descripción del contenido.
 - **position** (Integer): Índice del elemento en la lista de contenidos.
 - **hasChildren** (Boolean): Indica si contiene elementos en su interior.
 - **hasGradeBookColumns** (Boolean): Indica si contiene calificaciones.
 - **availability** (Availability): Disponibilidad del elemento
- **CourseGrade**: Calificación:
 - **displayName** (String): Nombre a mostrar.
 - **description** (String): Descripción de la calificación.
 - **contentId** (String): Identificador del contenido de la calificación en Campus.
 - **thuliumId** (String): Identificador del contenido de la calificación en Thulium.
 - **content** (Content): Contenido de la calificación.
- **Course**: Curso de una materia:
 - **name** (String): Nombre del curso.
 - **grades** (Array de CourseGrade): Conjunto de calificaciones del curso.
 - **contents** (Array de Content): Conjunto de contenido del curso.
- **Membership**: Relación de pertenencia entre usuario y curso:
 - **userId** (String): Identificador del usuario.
 - **courseId** (String): Identificador del curso.
 - **courseRoleId** (String): Identificador del rol del usuario en el curso.
 - **course** (Course): Curso en cuestión.
- **Query**: Punto inicial de GraphQL, con el cual se puede pedir las membership del usuario autenticado:

- **memberships** (Array de Membership): Conjunto de relación a cursos.

10.6. Ejemplo de comunicación entre módulos

Con los módulos descritos previamente, se establece un circuito de información. Por ejemplo, para crear instancias de varios datasets para un examen:

1. La app del cliente realiza un pedido de carga de un examen.
2. El módulo de API recibe ese request.
3. A partir de la información necesaria, se envía por la cola de mensajes la tarea que crea los datasets hacia server de jobs.
4. Se avisa al cliente cómo identificar la tarea que se corresponde con la solicitud propuesta.
5. Cuando una instancia de dataset finalmente es creada, el job correspondiente notifica al server de websockets que su tarea fue finalizada.
6. El server de websockets le avisa al cliente que la instancia de dataset fue creada.
7. El cliente, cuando todas las instancias de dataset hayan sido creadas, cambiará la UI, y el usuario seguirá interactuando con Thulium.

10.7. Ejecución de queries

10.7.1. Dataset instance

Un requerimiento primordial a la hora de tener en cuenta el manejo de datasets, es que las acciones de un usuario no tienen que afectar al resto de los usuarios. En otras palabras, las alteraciones que un usuario provoque sobre un dataset debería verlas solo dicho usuario. Con el motivo de solucionar este problema, se pensó la estructura del dataset instance.

El dataset instance es una abstracción de un dataset que guarda un estado interno con los cambios que el usuario ejecuta sobre dicho dataset.

Es transparente para el usuario y el mismo no percibe su existencia. En términos simples, el dataset es un molde a partir del cual se generan instancias de dicho dataset para distintos usuarios. Para cada par usuario-dataset, puede existir un y sólo un dataset instance.

El dataset instance es por naturaleza stateful y borrarlo implica que el usuario vuelva el estado de un dataset donde está trabajando a las condiciones predeterminadas. Por ejemplo, si un usuario ejecuta consultas que agregan filas y columnas en un dataset, dichas consultas afectan al dataset instance y no al dataset per se; si se elimina ese dataset instance y se crea uno nuevo, a los ojos del usuario el dataset presentará la cantidad de filas y columnas que tenía originalmente.

El proceso de instanciación de un dataset consiste la creación de tablas lógicas en la base de datos interna, con referencias e información necesaria para acceder a una tabla física

almacenada en el motor de base de datos. Se crea una tabla física por cada dataset item (representación lógica de una tabla) que posea el dataset. Una vez creados, se obtienen todas las entradas correspondientes a cada dataset item (registros) y se insertan en la tabla física.

10.7.2. Parseo de queries, renaming de las tablas

Uno de los desafíos más grandes de Thulium es el manejo básico de las consultas SQL ejecutadas por los usuarios. Existen determinados requerimientos de permisos y garantías que es necesario ofrecerle a los usuarios.

Por ejemplo:

- Si un usuario crea una tabla llamada “users” en la base de datos de postgres, el resto de los usuarios también debería poder crear una tabla con el mismo nombre y esas tablas deberían ser capaces de convivir en la misma base de datos. Además, si cada usuario ejecuta una consulta que afecte a la tabla “users”, dicha consulta debería afectar a la tabla perteneciente al usuario que la ejecutó.
- La división de permisos debería ser efectiva de manera que los usuarios no vean información de tablas que no les pertenecen.

Para lograr esto, es necesario evitar ejecutar consultas directamente en la base de datos por parte de los usuarios, y realizar algún tipo de metodología ingenieril entre medio.

Thulium ofrece una lógica que actúa como un proxy entre las consultas de los usuarios para cumplir con todos estos requerimientos.

Para cumplir con la división de permisos, implementamos la estructura dataset instance, explicada en la sección 10.7.1 y corroboramos que la tabla pertenezca al usuario. Además, el cliente envía al servidor el identificador del dataset sobre el cual se está ejecutando la consulta.

En cuanto a ofrecer la posibilidad de que tablas con el mismo nombre convivan en la misma base de datos, se implementó un sistema de renaming de tablas. Cada vez que un usuario crea o utiliza un dataset, se deben crear tablas físicas en la base de datos que se pretende utilizar. Como no es posible guardar tablas con el mismo nombre, se genera un nombre con el formato “table_<id>”, donde el id es un unique ID generado en base parámetros únicos del usuario (como su id). A su vez, en la base de datos interna de Thulium, se crea una denominada tabla lógica, que guarda información necesaria para acceder a la tabla física. La información de la tabla lógica es almacenada en el dataset instance.

Con esta implementación es posible crear tablas con el mismo nombre, pero es necesario que la ejecución de queries sea coherente con este sistema. Esto significa que, por ejemplo, una vez creada la tabla “users”, el usuario debe poder referirse a la misma como “users” y no como “table_<id>” durante una consulta. El procedimiento a seguir es el siguiente:

- El usuario ejecuta una consulta y el cliente envía al servidor la consulta junto con otra metadata (como el dataset a usar).
- El servidor busca el dataset instance correspondiente al usuario y se realizan los chequeos correspondientes de seguridad.

- Se parsea la query y se genera el árbol AST de la query. De esta manera, es posible descartar aquellas queries sean sintácticamente incorrectas.
- El dataset instance posee un mapa cuya clave es el nombre lógico de la tabla y su valor es el nombre físico de la tabla. En el ejemplo, si se pide el valor para la clave “users”, se obtiene “table_<id>”.
- Los nombres lógicos de las tablas en la query son reemplazados por los nombres físicos.
- El árbol AST se vuelve a traducir en una consulta SQL.
- La consulta pasa a la siguiente capa en donde se la despacha como un Job.

Por ejemplo, si la consulta ejecutada por un usuario es “SELECT * FROM users;”, al concluir este proceso, la misma será internamente traducida a “SELECT * FROM table_<id>;”

La limitación de este sistema es que sólo se pueden ejecutar queries que involucren tablas existentes dentro de un mismo dataset. Es una decisión que se tomó basada en la lógica que implementa [Google Cloud BigQuery](#) [27] con sus tablas y datasets.

```
{
  "_id" : ObjectId("5d10fede90eb1f0ced617fb2"),
  "dataset" : ObjectId("5d10fc95b7a3de0cecff34de"),
  "owner" : ObjectId("5d03b4ec19df5d160c564893"),
  "engine" : "psql",
  "tables" : {
    "datos" : "table_ecd29be22af89c362f90f435e1bd14a4"
  },
  "exam" : ObjectId("5d10fe91b7a3de0cecff34f9"),
  "__v" : 0
}
```

Fig 22 - Dataset instance y su mapeo de la tabla lógica “datos” a la física.

10.8. Integración con Blackboard

Para la integración con Blackboard se realizaron las siguientes tareas:

1. Registrar la aplicación en <https://developer.blackboard.com>.
2. En Blackboard registrar la aplicación para autorizarla.
3. Implementar la autenticación descrita en la sección 10.5.1
4. Implementar las interacciones descritas las secciones en 10.5.1 y 10.5.2.

Luego, la integración recae en realizar los llamados adecuados a la API de acuerdo a lo necesario. Los endpoint que ofrece la API de Blackboard son los siguientes: <https://developer.blackboard.com/portal/displayApi>.

11. Producto final

11.1. Pantalla de Login

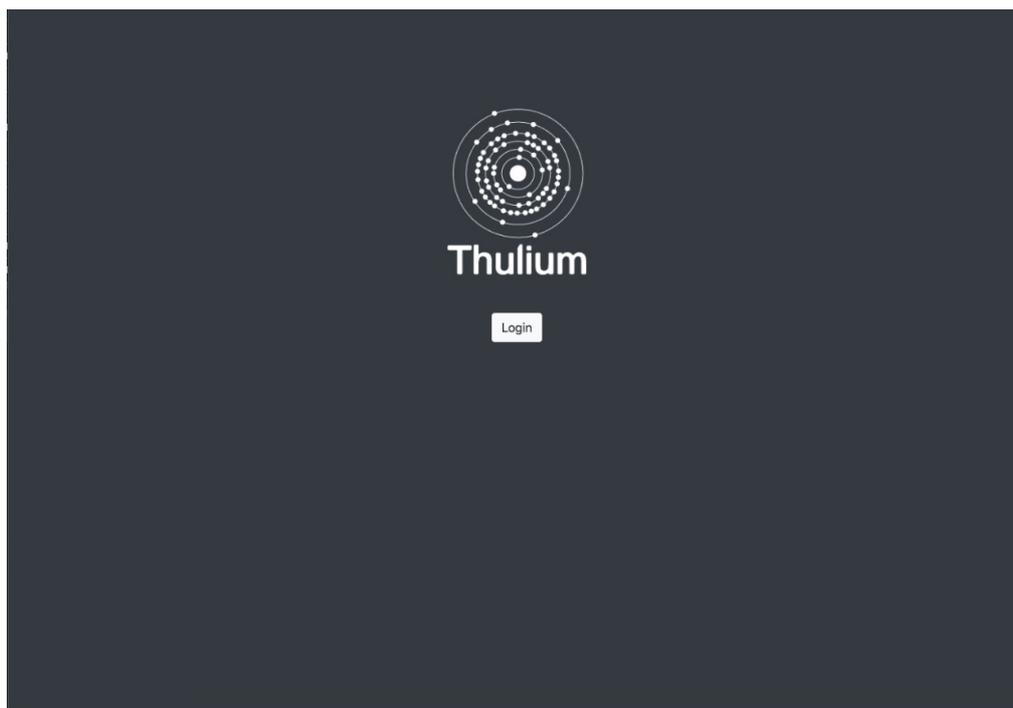


Fig 23: Pantalla de login

En esta pantalla se muestra cuando el usuario no está autenticado con la aplicación. Se presenta el logo de Thulium y el botón para iniciar sesión.

11.2. Pantalla en Modo Playground

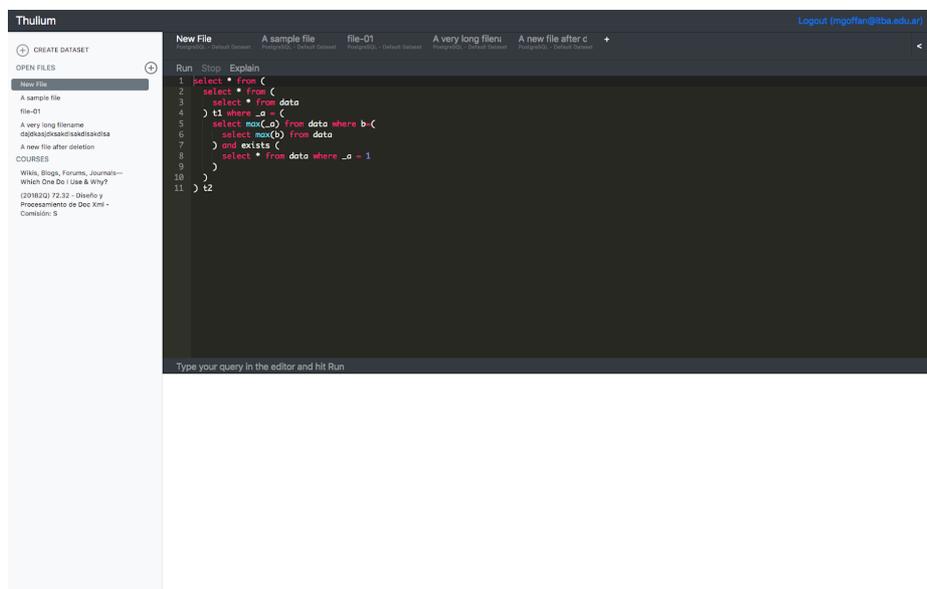


Fig 24: Pantalla en modo playground

En esta pantalla se distinguen las siguientes secciones:

- **Barra superior:** esta contiene el isotipo identificador por la izquierda y los datos del usuario autenticado por la derecha.
- **Barra lateral:** contiene el botón para crear un nuevo dataset, el listado de archivos abiertos y el listado de cursos a los que se encuentra inscripto.
- **Barra de archivos abiertos:** se muestran los archivos abiertos, con su nombre, motor y dataset correspondientes. Por la derecha existen botones para navegar la zona de overflow y poder tener múltiples archivos abiertos. Además, tiene un botón para crear nuevos archivos
- **Barra de acción:** Es aquella que tiene los botones para correr, frenar y explicar queries.
- **Barra de estado:** Muestra la información sobre la última query ejecutada.
- **Caja de resultados:** Muestra en una grilla los resultados de la última query ejecutada.

11.3. Pantalla para crear nuevos archivos

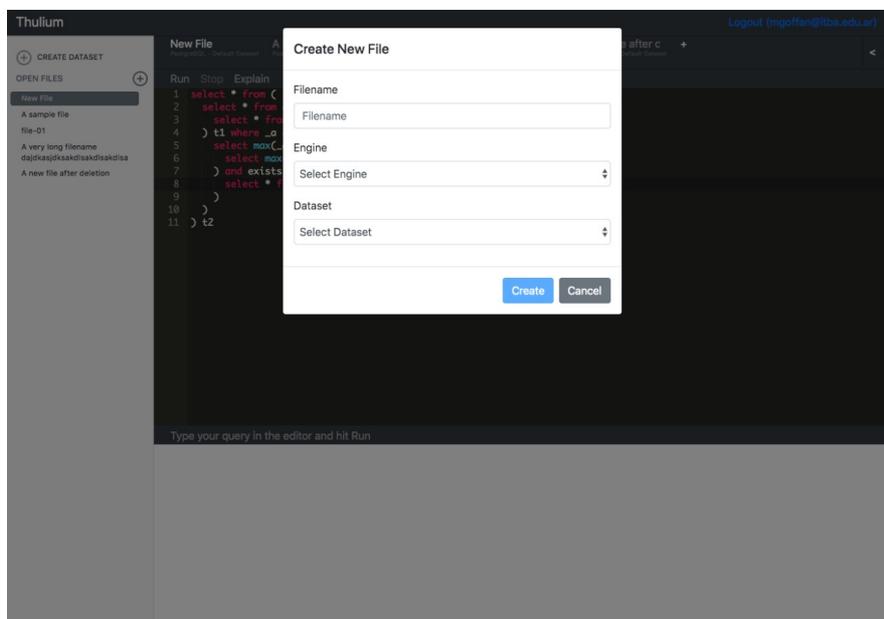


Fig 25: Creación de nuevo archivo

Para crear nuevos archivos, se abre en forma de “modal” el formulario que pide un nombre de archivo, un motor y un dataset.

11.4. Pantalla para la creación de un dataset

Para crear un dataset se muestran un conjunto de “modal dialogs” que a modo de “wizard”, en pasos, facilitan la creación de un dataset.

11.4.1. Paso 1: Elección de un paradigma

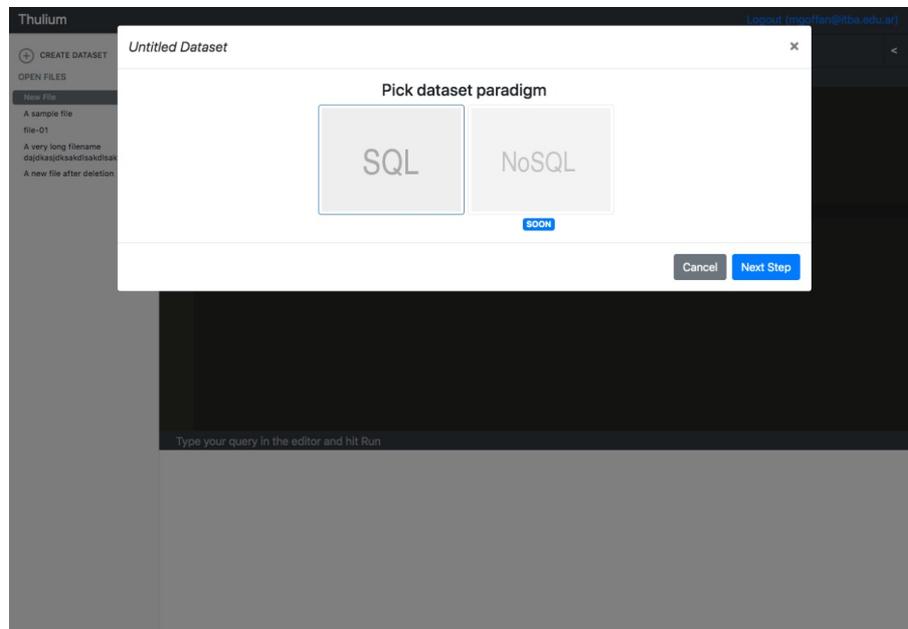


Fig 26 - Profesor quiere subir un dataset - Paradigma

En esta pantalla se elige entre SQL y NoSQL. Para el presente desarrollo NoSQL no se encuentra disponible. Además, se puede cambiar el título de un dataset haciendo click en la parte superior donde por defecto dice "Untitled Dataset" y una vez que se cambió el nombre haciendo enter para que tome efecto el cambio.

11.4.2. Paso 2: Subida de CSV

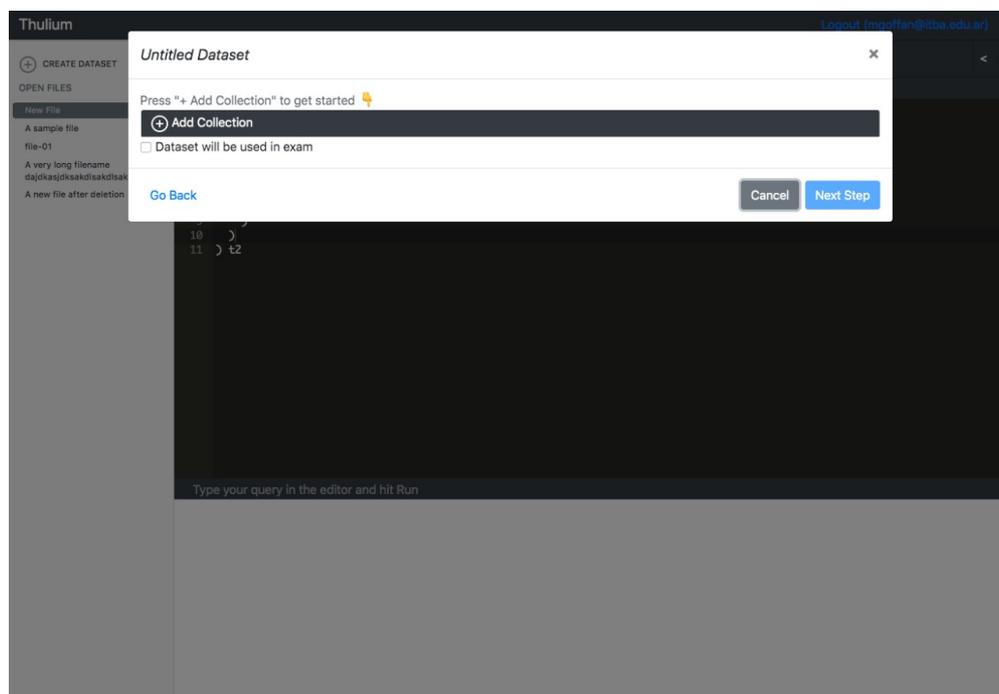


Fig 27 - Profesor quiere subir un dataset - Collection

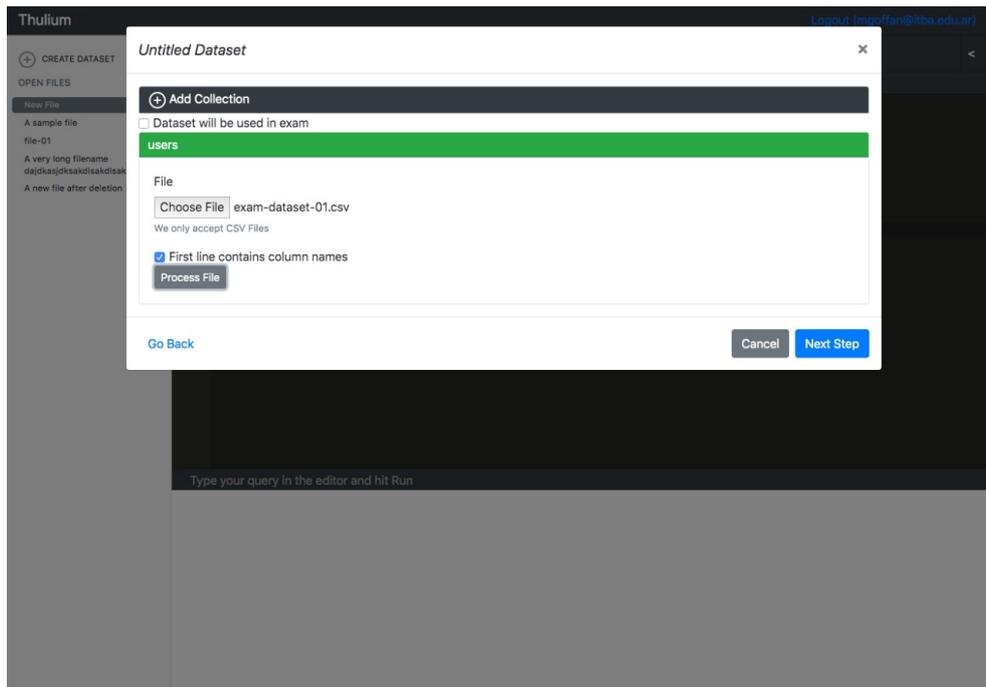


Fig 28 - Profesor quiere subir un dataset - Archivo subido

En esta parte se suben las colecciones de datos deseadas, se les asigna un título a cada colección y se sube el archivo CSV que corresponde a cada colección. Una vez que se procesa el archivo, cada colección se muestra en verde y se puede seguir al próximo paso.

11.4.3. Paso 3: Revisión de los Datos

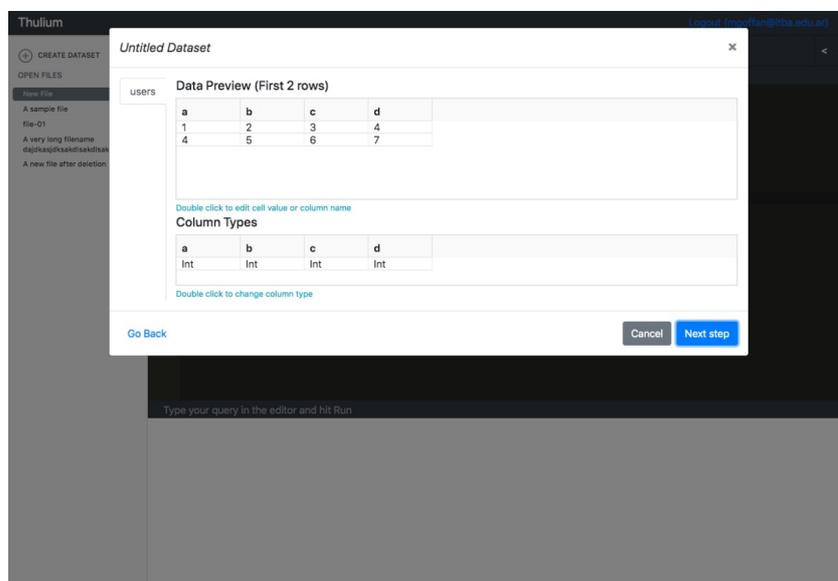


Fig 29 - Profesor quiere subir un dataset - Edición final

En esta etapa se revisan los datos y se puede cambiar el tipo de datos que muestra cada columna. Los tipos de datos se intuyen a partir de tomar una muestra de los datos.

11.4.4. Paso 4: Selección de Permisos

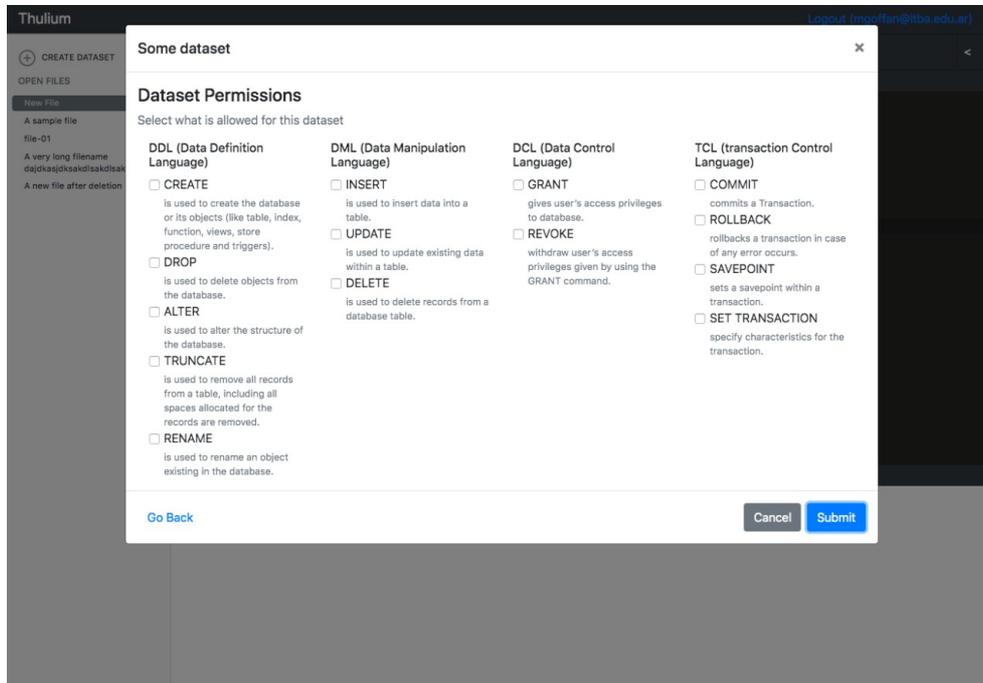


Fig 30 - Profesor quiere subir un dataset - Permisos

En esta etapa se pueden seleccionar las acciones que se pueden realizar en el dataset.

11.5. Ejecución de una Query

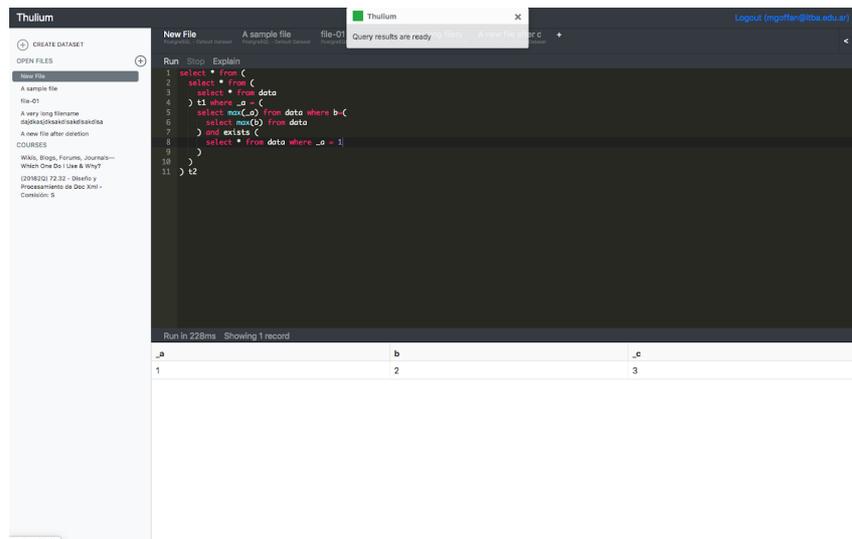


Fig 31 - Profesor quiere subir un dataset - Permisos

En esta pantalla se muestran los resultados de una query y un cartel de éxito indicando que se ejecutó exitosamente la query.

11.6. Creación de un examen

11.6.1. Paso 1: Vista de un curso

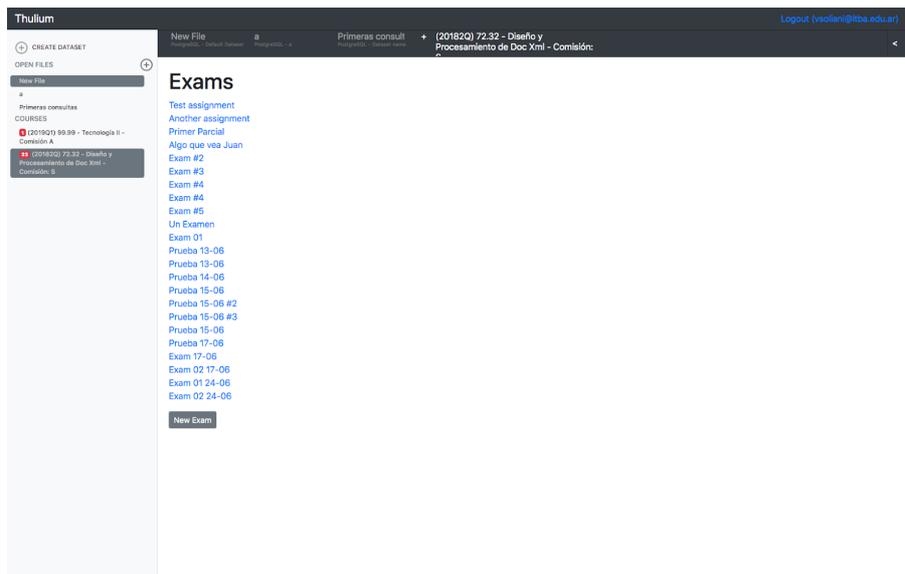


Fig 32 - Creación de un examen - Pantalla principal

En esta pantalla se listan todos los exámenes disponibles en un curso. Para crear un nuevo examen es necesario hacer click en “New Exam”.

11.6.2. Paso 2: Formulario de creación de un examen

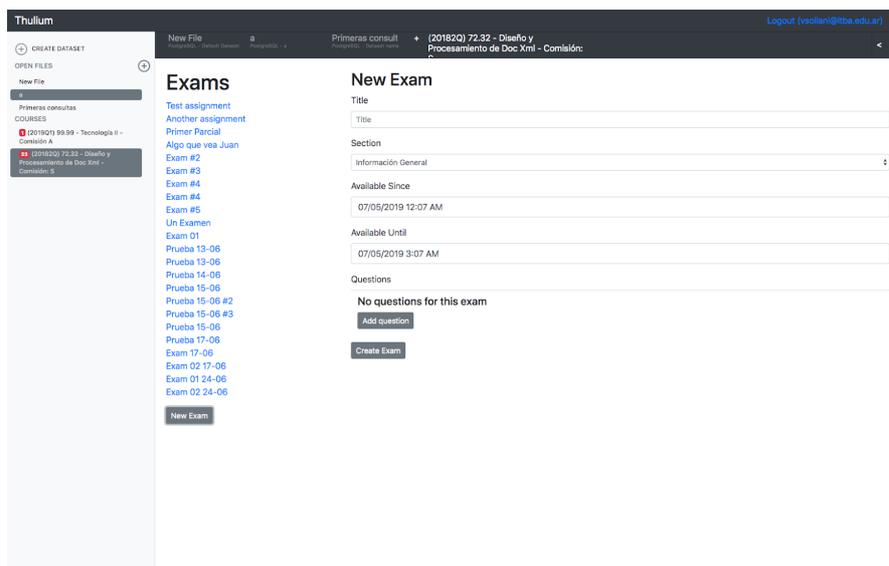


Fig 33 - Creación de un examen - Información básica

En esta pantalla se muestran los campos necesarios en el formulario para crear el examen.

11.6.3. Paso 3: Creación de Preguntas

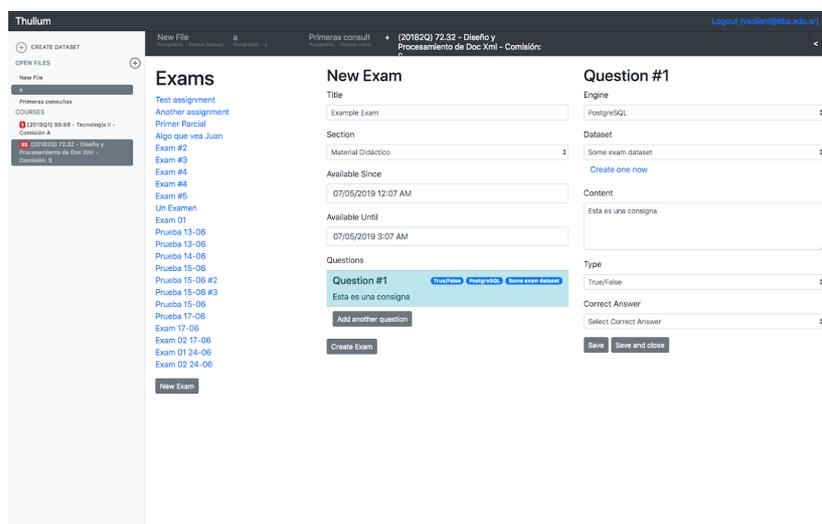


Fig 34 - Creación de un examen - Creación de preguntas del examen

Se ingresan los campos necesarios para cada una de las preguntas y se van guardando en el examen. Cuando todos los campos están completos haciendo click en “Create Exam” se crea el examen.

11.6.4. Paso 4: El examen se muestra en Campus

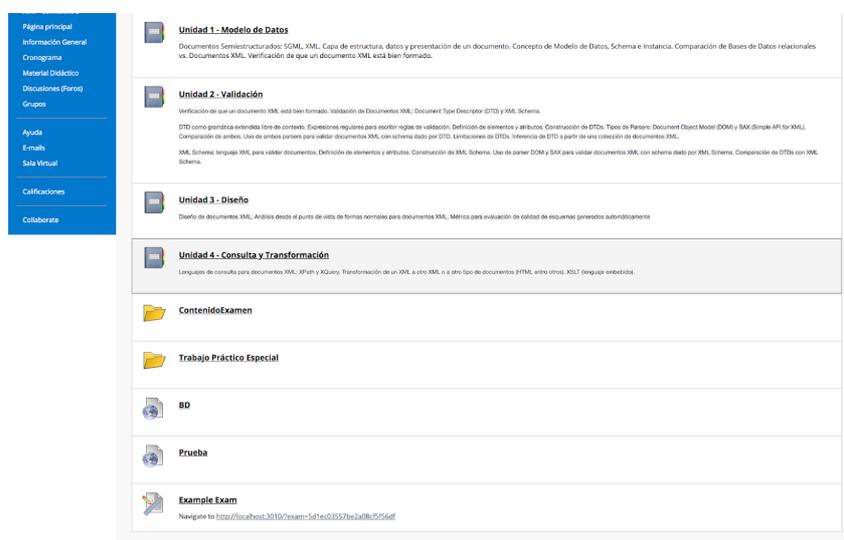


Fig 35 - Creación de un examen - El examen se muestra en campus

Como alumno se puede ver que la actividad está creada en Campus en la sección de Material Didáctico

11.7. Modalidad Examen

11.7.1. Paso 1: Identificación de examen vigente

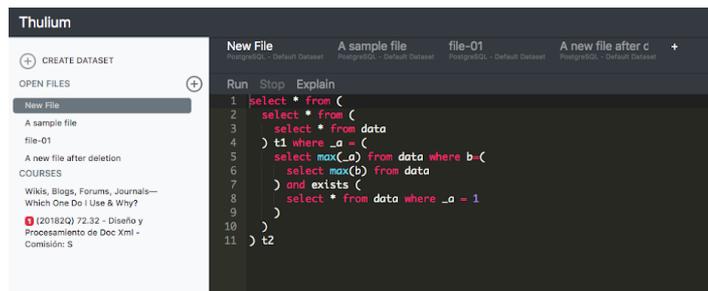


Fig 36 - Modalidad Examen - Identificación del examen vigente

Se muestra con un número en rojo, al lado del curso, la cantidad de exámenes disponibles que el alumno tiene para el curso.

11.7.2. Paso 2: Vista del curso

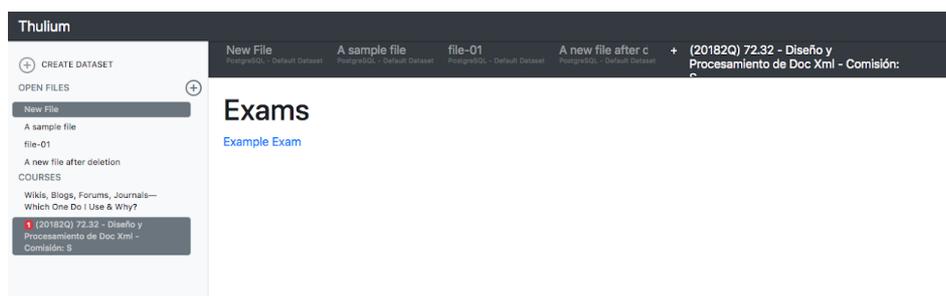


Fig 37 - Modalidad Examen - Vista del curso

Se listan los exámenes disponibles en el curso. Haciendo click se accede al examen.

11.7.3. Paso 3: Vista del examen

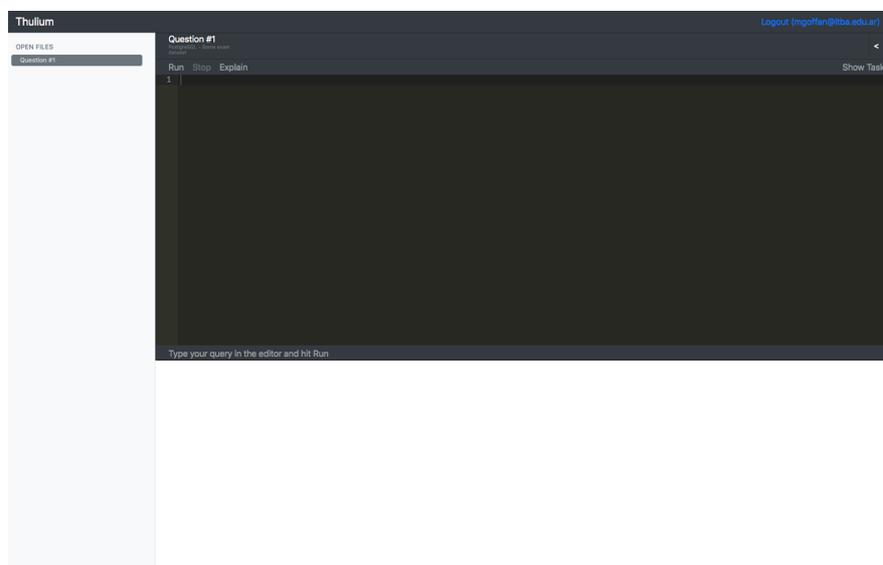


Fig 38 - Modalidad Examen - Vista del examen

Se destaca en esta pantalla, que la mayoría de las funcionalidades disponibles en modo playground no son accesibles. Se crean automáticamente un archivo por consigna y es todo lo que se puede hacer en esta modalidad.

11.7.4. Paso 4: Vista de la consigna y envío de respuesta

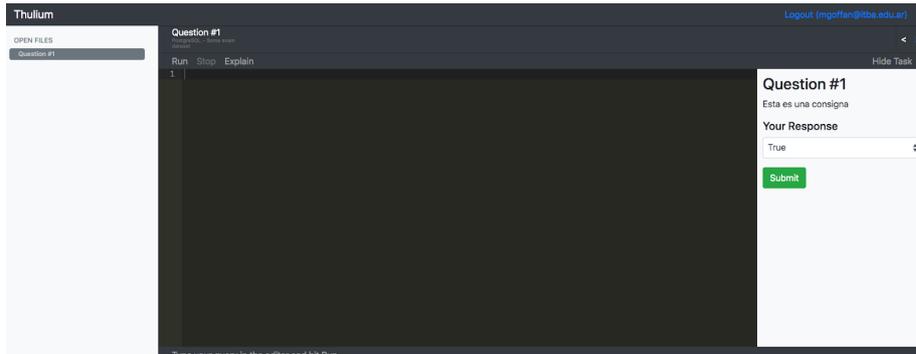


Fig 39 - Modalidad Examen - Vista de la consigna y envío de respuesta

Se muestra de modo poco obstrusivo la consigna y la posibilidad de enviar la respuesta para la pregunta en cuestión.

11.8. Corrección de exámenes

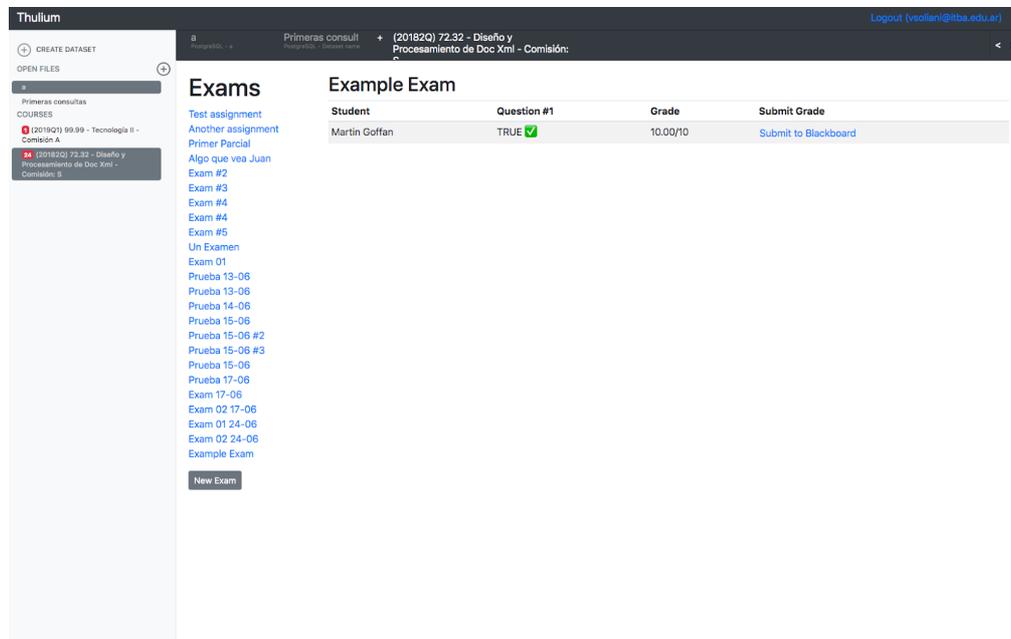


Fig 40 - Modalidad Examen - Corrección del examen

Accediendo al curso y seleccionando el examen, se muestra la grilla con los resultados de cada pregunta para cada alumno que envió alguna respuesta. En caso de la respuesta sea auto-correctible entonces se indica con si la respuesta es correcta, en caso contrario se indica con . Para las respuestas de tipo query, entonces el resultado de la validación se muestra

como 👍 para un resultado positivo y con ☹️ en caso que sea negativo. Este tipo de respuestas son una ayuda para el profesor que cuenta con un “modal” en el cual se despliega la respuesta del alumno para que pueda cerciorarse de que la respuesta sea efectivamente correcta o incorrecta. Por último, apretando el botón “Submit to Blackboard” envía la calificación del alumno a Campus.

12. Conclusiones

Como resultado del proceso de desarrollo, es posible concluir que se lograron los objetivos propuestos al inicio de desarrollo del Proyecto Final. A su vez, se logró realizar la prueba de dos casos de exámenes previamente tomados en la universidad. El primero, es el caso de un examen tomado durante el primer cuatrimestre de 2019 en la materia de Bases de Datos I. El segundo, es un examen simple en Campus ITBA. Ambos casos de prueba han sido satisfactorios.

13. Trabajo futuro

Si bien el desarrollo del trabajo fue intensivo, existen muchas implementaciones de features que sería conveniente tener pero que exceden el alcance de este Trabajo Práctico Final. A continuación se detalla un roadmap sugerido para continuar el desarrollo de este proyecto en un futuro:

- Implementar objetivos no alcanzados:
 - Motores de base de datos no relacionales, en base al modelo genérico de un motor de base de datos. implementación de vistas y otras facilidades que ofrecen algunos motores NoSQL, como por ejemplo Mongo, cuya implementación se dejó de lado debido a cambios en el roadmap, y CouchDB.
 - Importación de un dataset por parte de un alumno, basado en lo ya implementado, dándole acceso únicamente a alumnos del mismo curso o al alumno que lo creo.
 - Estadísticas, reporting o cualquier tipo de análisis complementario respecto a los resultados. Se sugiere la implementación del stack ELK (ElasticSearch, Logstash, Kibana), que permitirá obtener métricas sobre la performance general de los alumnos, que son valubles para el profesor y la universidad. Además, permitirá obtener métricas sobre el funcionamiento general del sistema.
 - Modalidad de autoevaluación, basado en código preexistente. Consiste en preguntas de tipo query, con posibilidad de tener un dataset oculto y ejecutarse fuera de la modalidad de examen.
- Realizar pruebas de implementación en exámenes reales o con alumnos de grado en prácticas de materias acordes. Estas pruebas deberían ser tanto de performance, como stress de red.
- Hacer un sistema que permita que el sistema de Jobs sea compatible con un scaling en base al tamaño de la cola.

14. Referencias

1. <https://aws.amazon.com/es/nosql/> , última revisión: 05/07/19
2. <https://oauth.net/2/> , última revisión: 05/07/19
3. <https://jwt.io/> , última revisión: 05/07/19
4. <http://sqlfiddle.com/> , última revisión: 06/06/19
5. <https://lagunita.stanford.edu/courses/DB/SQL/SelfPaced/course/> , última revisión: 06/06/19
6. <https://pgexercises.com> , última revisión: 06/06/19
7. <https://www.tutorialspoint.com/codingground.htm> , última revisión: 06/06/19
8. <https://databricks.com/> , última revisión: 06/06/19
9. www.hackerrank.com , última revisión: 06/06/19
10. <https://aws.amazon.com/cloud9/?origin=c9io> , última revisión: 06/06/19
11. <https://developers.facebook.com/> , última revisión: 06/06/19
12. <https://www.compose.com/> , última revisión: 06/06/19
13. <https://reactjs.org/> , última revisión: 05/07/19
14. <https://redux.js.org/> , última revisión: 05/07/19
15. <https://nodejs.org/en/> , última revisión: 05/07/19
16. <http://zeromq.org/> , última revisión: 05/07/19
17. <https://www.mongodb.com/> , última revisión: 05/07/19
18. <http://cassandra.apache.org/> , última revisión: 05/07/19
19. <http://couchdb.apache.org/> , última revisión: 05/07/19
20. <https://mongoosejs.com/> , última revisión: 05/07/19
21. <https://www.docker.com/> , última revisión: 05/07/19
22. <https://kubernetes.io/> , última revisión: 05/07/19
23. <https://expressjs.com/> , última revisión: 05/07/19
24. <https://caolan.github.io/async/v3/> , última revisión: 05/07/19
25. <https://webpack.js.org/> , última revisión: 05/07/19
26. <https://graphql.org/learn/> , última revisión: 05/07/19
27. <https://console.cloud.google.com/> , última revisión: 05/07/19
28. Material didáctico de la cátedra de Base de Datos I.
 - a. Garcia-Molina, H., Ullman, J. D. & Widom, J. (2008). Database Systems: The Complete Book. Prentice Hall, 2nd Ed.
 - b. Elmasri, R. & Shamkant Navathe, S. (2010). Fundamentals of Database Systems. Addison Wesley.
29. Material didáctico de la cátedra de Criptografía y Seguridad.
 - a. BISHOP, Matt. Computer Security – Art and Science. Boston: Addison-Wesley, 2004. 1136p.
 - b. KATZ, Jonathan, LINDELL, Yehuda. Introduction to Modern Cryptography. Chapman & Hall/CRC Cryptography and Network Security Series, 2007. 552p.
30. Material didáctico de la cátedra de Interacción Hombre Computadora..
 - a. Jokela T, Iivari N., Matero J. & Karukka M. (2003). The standard of user-centered design and the standard definition of usability: analyzing ISO 13407 against ISO

9241-11. Proceedings of the Latin American conference on Human-computer interaction, CLIHC '03, Rio de Janeiro, Brasil.

b. Hilbert D. M. & Redmiles D. F. (2000). Extracting usability information from user interface events. ACM Comput. Surv, 32(4).

31. <https://github.com/jpascale/pg-query-parser> , última revisión: 05/07/19

32. <https://www.blackboard.com/index.html> , última revisión: 05/07/19

33. <https://developer.blackboard.com/> , última revisión: 05/07/19

34. <https://community.blackboard.com/login.jspa> , última revisión: 05/07/19

15. Anexos

15.1. Anexo A: Diagramas de actividad de los Casos de uso

1. CU1: Alumno / Profesor quiere ejecutar query

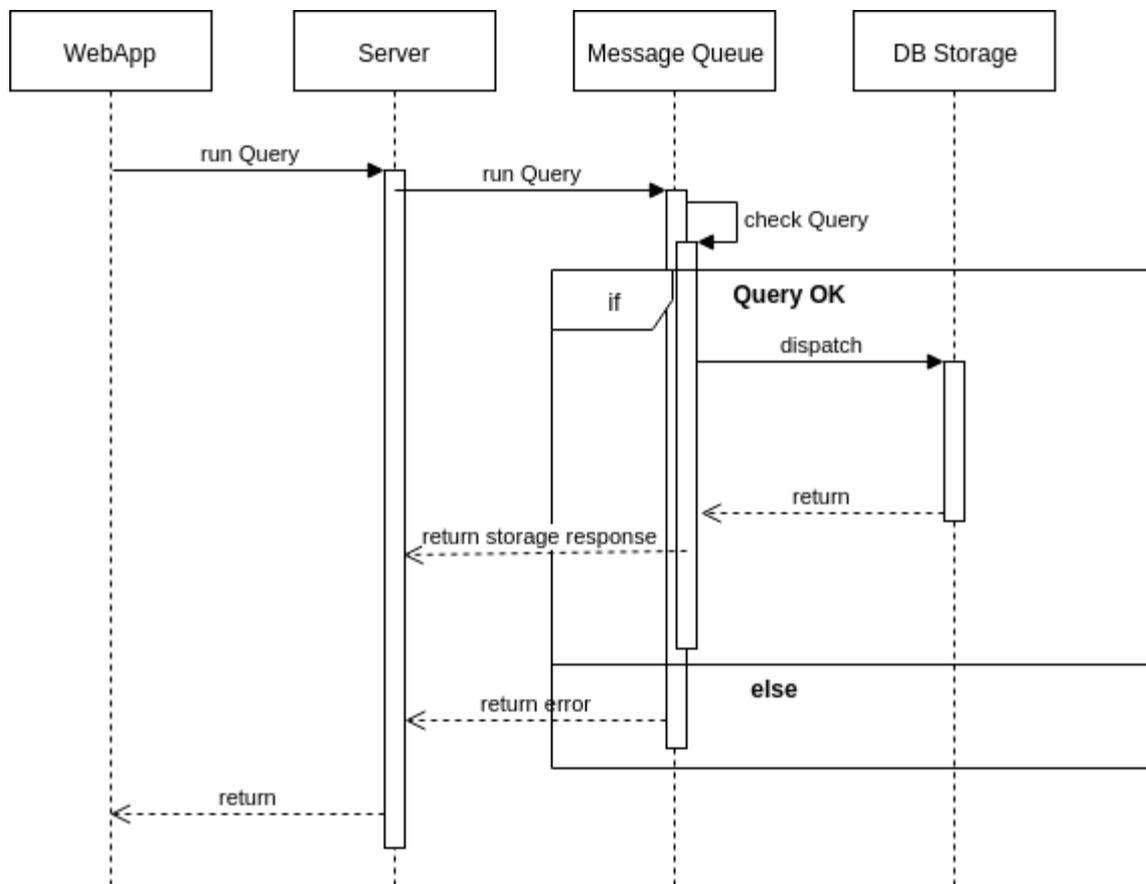


Fig 41: Alumno / Profesor quiere ejecutar query

2. CU2: Alumno / Profesor crea nueva pestaña

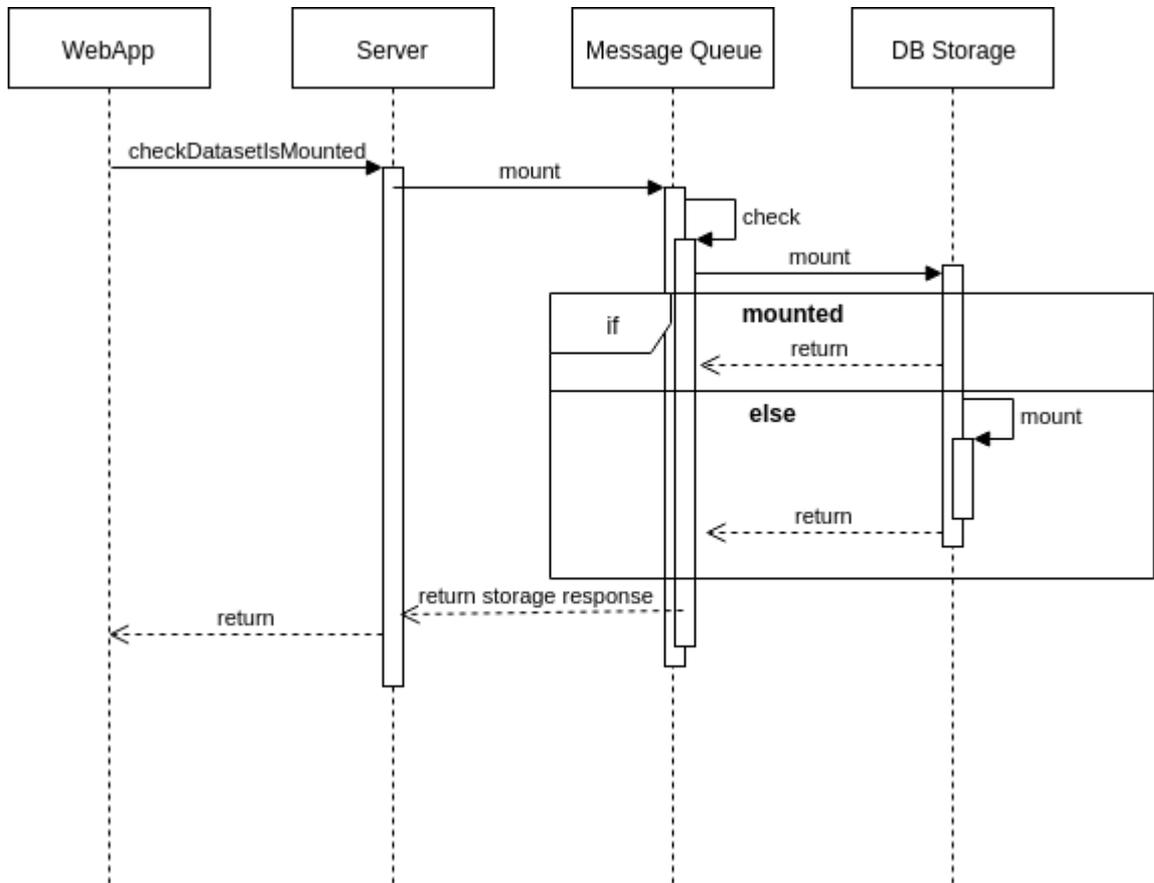


Fig 42: Alumno / Profesor crea una nueva pestaña

3. CU3: Alumno / Profesor quiere hacer login

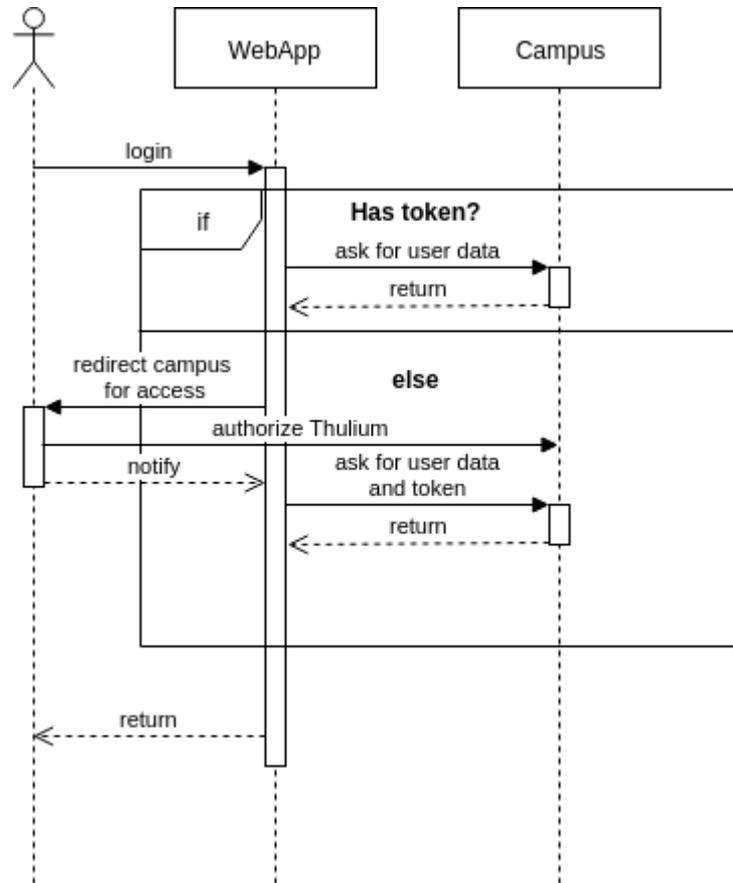


Fig 43: Alumno / Profesor quiere hacer login

4. CU7: Profesor quiere subir un examen

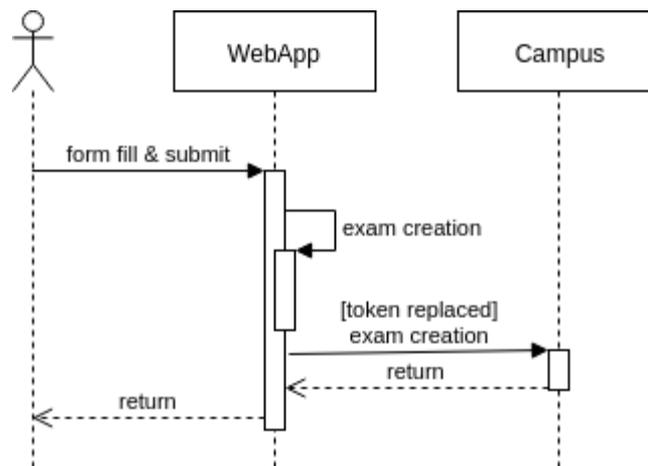


Fig 44: Profesor quiere subir un examen

5. CU8: Profesor revisa un examen

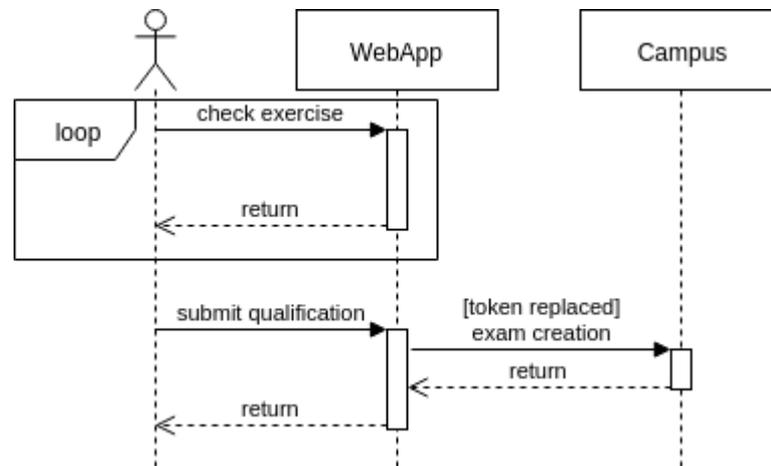


Fig 45: Profesor revisa un examen

15.2 Anexo B: Archivos de módulo de configuración

1. app.json

```
{
  "development": {
    "blackboard": {
      "REDIRECT_URI":
"http://localhost:3000/core/v1/auth/social/blackboard"
    },
    "engines": [
      {
        "id": "psql",
        "title": "PostgreSQL",
        "enabled": true
      },
      {
        "id": "mongo",
        "title": "MongoDB",
        "enabled": false
      },
      {
        "id": "mysql",
        "title": "MySQL",
        "enabled": true
      }
    ],
    "defaultEngine": "psql",
    "defaultDataset": {
      "title": "Default Dataset",
      "paradigm": "sql",
      "items": [
        {
          "title": "data",
          "data": [
            [
              "1",
              "2",
              "3"
            ]
          ]
        }
      ]
    }
  }
}
```

```

    ],
    "headers": [
      "a",
      "b",
      "c"
    ],
    "types": [
      "Int",
      "Int",
      "Int"
    ]
  }
],
"actions": {
  "can_create": false,
  "can_drop": false,
  "can_alter": false,
  "can_truncate": false,
  "can_rename": false,
  "can_insert": false,
  "can_update": false,
  "can_delete": false,
  "can_grant": false,
  "can_revoke": false,
  "can_commit": false,
  "can_rollback": false,
  "can_savepoint": false,
  "can_set_transaction": false
}
},
"production": {
  "blackboard": {
    "REDIRECT_URI":
"https://\api.thulium.xyz\core\v1\auth\social\blackboard"
  },
  "engines": [
    {
      "id": "psql",
      "title": "PostgreSQL",
      "enabled": true
    },
    {
      "id": "mongo",
      "title": "MongoDB",
      "enabled": false
    },
    {
      "id": "mysql",
      "title": "MySQL",
      "enabled": true
    }
  ],
  "defaultEngine": "psql",
  "defaultDataset": {
    "title": "Default Dataset",
    "paradigm": "sql",
    "items": [
      {
        "title": "data",
        "data": [
          "1",
          "2",
          "3"
        ]
      }
    ]
  }
}

```



```

    "mq": {
      "host": "127.0.0.1",
      "port": 13001
    },
    "pubsub": {
      "host": "127.0.0.1",
      "port": 14001
    },
    "reqres": {
      "host": "127.0.0.1",
      "port": 15001
    },
    "postgres": {
      "host": "35.164.108.20",
      "port": 5432,
      "user": "dev",
      "password": "dev",
      "database": "dev"
    },
    "mysql": {
      "host": "35.164.108.20",
      "port": 3306,
      "user": "thulium",
      "password": "thulium",
      "database": "thulium"
    }
  },
  "secret":
  "5LQCVbjhpSL4dM1VR4dLReogd09TbZailVMumuWEw6qr0T8tzmlHc9cjKmbR8y5NNPam8rBKe/yw+qvt
  2Sbq5g==",
  "jwt": {
    "audience": "https://thulium.xyz",
    "issuer": "https://thulium.xyz"
  }
}

```

Tabla 19: *app.secure.json* - Archivo con las credenciales, permite utilizar distintas credenciales dependiendo del entorno en donde se ejecute la aplicación. No es seguro commitearlo.

15.3 Anexo C: Rutas de la API

```

GET    /core/v1/session/mine
GET    /core/v1/auth/social/blackboard
POST   /core/v1/auth/register
POST   /core/v1/auth/login
GET    /core/v1/auth/profile
GET    /core/v1/engines
GET    /core/v1/engines/:id([a-f0-9]+)
POST   /core/v1/files
PATCH /core/v1/files/:id([0-9a-f]+)
POST   /core/v1/datasets
GET    /core/v1/datasets/all
GET    /core/v1/datasets/datasets
GET    /core/v1/datasets/instances

```

POST	/core/v1/datasets/create
POST	/core/v1/exams/:id([a-f0-9]+)/load
POST	/core/v1/exams/:eid([a-f0-9]+)/response/:qid([a-f0-9]+)
GET	/core/v1/exams/:id([a-f0-9]+)/responses
PATCH	/core/v1/exams/:id([a-f0-9]+)/review

Tabla 20: Rutas de la API

16. Glosario

- Abstract Syntax Tree (AST): Estructura que permite describir código de programación o queries de bases de datos en forma de árbol a partir de la composición de términos a partir de la sintaxis de la misma.
- Credencial: Una credencial o token, es una prueba de identidad respecto a un usuario.
- Flujo de control: El flujo de control (o control flow) hace referencia a la especificación del orden en que se ejecutan o evalúan las declaraciones individuales, las instrucciones o las llamadas a funciones de un programa.
- Front Controller Pattern: Patrón de diseño principalmente para WebApps, separando en 4 componentes principales: Un Controlador (de requests), la vista, un dispatcher (para el manejo de vistas) y helpers (para manejar el comportamiento de los modelos).
- Programación orientada a aspectos: El principal objetivo de la POA es la separación de las funcionalidades dentro del sistema: Por un lado funcionalidades comunes utilizadas a lo largo de la aplicación; y por otro, las funcionalidades propias de cada módulo. Cada funcionalidad común se encapsula en una entidad.
- Experiencia del usuario (User Experience, UX): Es el concepto que explica cómo el usuario percibe la usabilidad de un servicio.
- WebSocket: el RPC (remote process call) que proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP.