

Definición del proyecto - Módulo 1 y 2

Realizado por:

Andres Felipe Ramirez Montana

Isabella Garces Acosta

Johan David Lozano Leiva

Maria Luisa Bautista Arango

Ingeniería de Sistemas y Computación



Universidad Nacional de Colombia
Sede Bogotá
Ingeniería de software I

Profesor:

Oscar Eduardo Alvarez

2. Redacción del levantamiento de requerimientos

Como estudiantes, hemos enfrentado múltiples problemas y una experiencia poco satisfactoria al buscar cursos de interés debido a la falta de claridad y facilidad en la plataforma. Algunos errores comunes incluyen bugs como el cierre inesperado de la sesión al interactuar con ciertos campos del formulario.

Además, hay problemas de forma, como la existencia de dos facultades o dos planes de estudio con nombres idénticos, lo que genera confusión al intentar seleccionar la opción correcta. También, la falta de explicaciones claras, dificulta entender su propósito, así como la cantidad excesiva de campos en los formularios abruma y complica aún más la experiencia de usuario.

Finalmente, algunas funcionalidades son redundantes, lo que incrementa la frustración al intentar buscar asignaturas. En general, la plataforma requiere mejoras significativas en términos de usabilidad, interfaz y experiencia del usuario.

Adicionalmente hemos visto que hay múltiples herramientas de aprendizaje en la universidad poco usadas o que muchos no conocen, como lo son Coursera ó actividades extracurriculares como semilleros, grupos estudiantiles u otros que la gente no se entera. Hay bastante desconocimiento y falta de difusión, así como falta de canales de comunicación para las actividades fuera de las curriculares que pueden contribuir al refuerzo del conocimiento y la parte práctica, o incluso que sean didácticas.

Por lo anterior, el objetivo principal es crear una aplicación web que permita a los estudiantes una experiencia de usuario optimizada que ayude a encontrar de una manera sencilla información relacionada con materias (empezando por las materias electivas de ingeniería de sistemas) y actividades extracurriculares.

Hemos evidenciado que los factores más relevantes de los estudiantes a la hora de decidir qué electiva ver son:

- Qué tan fácil es
- Qué tan relevante es para mis gustos en intereses
- Recomendaciones (muchos estudiantes eligen una electiva porque un compañero la recomendó)

La plataforma debe tener:

Un **Sistema de búsqueda optimizada de cursos (buscador)** que permita a los estudiantes encontrar materias electivas en una sola página web. El buscador debe incluir filtros limitados para número de créditos, facilidad y recomendación y debe ser capaz de mostrar información organizada y clara, evitando confusiones como nombres duplicados.

Una visualización de la **descripción y temario de las asignaturas** que incluya créditos, sus objetivos principales, los temas a tratar, facilidad de la materia, y recomendaciones o comentarios de otros estudiantes y las materias por las que se puede reemplazar. Esto ayudará a los estudiantes a tomar decisiones informadas sobre los cursos que desean inscribir.

Una opción de **filtro simplificado y priorizado** que incluyan sólo los campos esenciales para evitar abrumar al usuario, número de créditos y facilidad. Debe incluir filtros avanzados o campos adicionales que se puedan activar mediante íconos o menús desplegables, sin afectar la experiencia principal, por ejemplo si se quiere buscar por cantidad de créditos, por temática general de las materias.

Una **sección de búsqueda de cursos de Coursera** en donde pueda buscar de manera más fácil los cursos de su interés que le ofrece el convenio de la universidad con Coursera. Así mismo el usuario puede requerir de guardar los resultados de la búsqueda que le llamen la atención, es decir **agregar cursos a “favoritos”**

Una **Sección dedicada a actividades extracurriculares** para organizar las actividades extracurriculares como semilleros y grupos o talleres, en forma de canales de difusión que deben incluir:

- Una vista general con una descripción del grupo o seminario y un tablero para publicaciones y notificaciones de los administradores o miembros.
- Una opción para **unirse al canal**, que permita a los estudiantes interactuar con las publicaciones, crear sus propias publicaciones, acceder a un chat grupal, y habilitar notificaciones por correo institucional.

Una Gestión de administradores en canales que debe permitir que cada usuario pueda **crear un canal** de forma que cada canal cuente con un administrador principal (el creador del canal) con la capacidad de **designar a otros estudiantes como co-administradores**. Los administradores podrán gestionar publicaciones, y miembros, de manera que deben poder **eliminar un usuario** del canal. Adicionalmente serán los únicos con el permiso para **crear publicaciones** que se mostrarán en el feed.

Una **sección “Mis Canales”** en donde el usuario pueda consultar los canales a los que pertenece en el momento de la consulta.

Un **Sistema de notificaciones** relacionadas con publicaciones del canal, recibéndolas en su correo institucional registrado que los usuarios podrán habilitar o deshabilitar una vez se hayan unido al canal.

Un **Chat grupal integrado** en cada canal accesible solo para los miembros del canal. Este chat debe ser sencillo y eficiente, con funciones básicas como enviar mensajes de texto, enlaces, e imágenes.

Un **Sistema de búsqueda y exploración de canales** para explorar los canales disponibles. Los usuarios podrán buscar canales por categorías (temática, facultad, intereses) y visualizar información básica (nombre, descripción, número de miembros) antes de unirse.

Un **Inicio de sesión** con el correo institucional se debe requerir para poder unirse a los canales de difusión. Se debe poder hacer la autenticación del correo institucional, el sistema solo debe validar la autenticación si es institucional, los correos personales no serán válidos. De la misma manera se debe poder **Cerrar la Sesión**.

En cuanto a **requerimientos no funcionales**:

Usabilidad: La página debe ser entendible y fácil de usar.

Escalabilidad: La página debe ser escalable para evitar problemas de aumento de usuarios o incorporación rápida de nuevos elementos como añadir otras carreras al buscador.

Rendimiento: La página debe ser rápida, debe cargar y mostrar el catálogo de una manera rápida.

¿Qué esperamos del desarrollo del proyecto?

- **Isabella Garces:**

A través del desarrollo de este proyecto espero aprender más sobre la organización de proyectos a mediana escala que puedan requerir de la implementación de más de una herramienta de código que además requiera de una conexión de estas múltiples herramientas entre ellas. De manera que espero aprender a estructurarlo de manera que sea limpio y fácil de entender. Aprender esto a través del proyecto será de gran ayuda en mi formación profesional porque me sirve como una base para ser capaz de desarrollar productos eficientes y de buena calidad en un futuro.

- **Luisa Bautista:**

Por medio de este proyecto quiero poner en práctica los aprendizajes de la materia para saber cómo aplicarlos en código y proyectos reales y también ampliar mis conocimientos sobre diferentes tecnologías como varios frameworks con los que nunca he trabajado.

- **Andrés Ramirez:**

Con la realización de este proyecto, espero fortalecer tanto mis conocimientos técnicos como teóricos en ingeniería de software. Además, me entusiasma porque me permite acercarme a la realidad del mercado informático y al funcionamiento de entidades importantes. Tengo grandes expectativas, especialmente en lo referente a los frameworks y su impacto en el desarrollo de aplicaciones. Asimismo, me motiva contribuir a la comunidad universitaria, mejorando su experiencia y haciendo su estancia más agradable.

- **Johan Lozano:**

Quiero poner a prueba mis conocimientos en construcción de producto y desarrollar una herramienta útil para todos los estudiantes de la Universidad Nacional de Colombia

3. Análisis de requerimientos:

MosCow			
Must	Should	Could	Won't
→ Inicio sesión → Sección búsqueda materias (Buscador) → Sección búsqueda cursos de Coursera (Buscador)	→ Descripción y temario de las asignaturas → Crear anuncio/publicación en el feed del canal (para admin)	→ Agregar cursos de coursera a los favoritos → Sistema de notificaciones (de canales) → Rendimiento	→ Escalabilidad → Agregar administradores → Expulsar miembros → Usabilidad.

→ Formulario especializado para búsqueda de materias electivas → Sistema de búsqueda y exploración de canales → Crear canal → sección "mis canales"	→ Unirse a un canal → Chat grupal (para canales) → Cerrar sesión.		
--	---	--	--

Definición de tecnologías y descripción:

La estimación del tiempo necesario para cumplir con los requerimientos estará determinada por la complejidad de las tareas y la curva de aprendizaje asociada a las siguientes tecnologías:

Servidor: Java con framework Spring y algunos de sus proyectos ☕

Frontend: Astro 🌟 para interfaz dinámica, HTML/JavaScript 💻, Tailwind CSS 🎨

Base de Datos: Firebase (NoSQL) 🔥

Control de Versiones: GitHub 📊

Las tecnologías mencionadas son esenciales para el desarrollo de cualquier página o aplicación web, complementadas con el uso de Git para un eficiente control de versiones que optimiza el trabajo en equipo. Si bien su programación e implementación pueden implicar un alto nivel de complejidad, esta se puede mitigar significativamente con el apoyo de herramientas como la inteligencia artificial, foros de desarrollo y la documentación oficial.

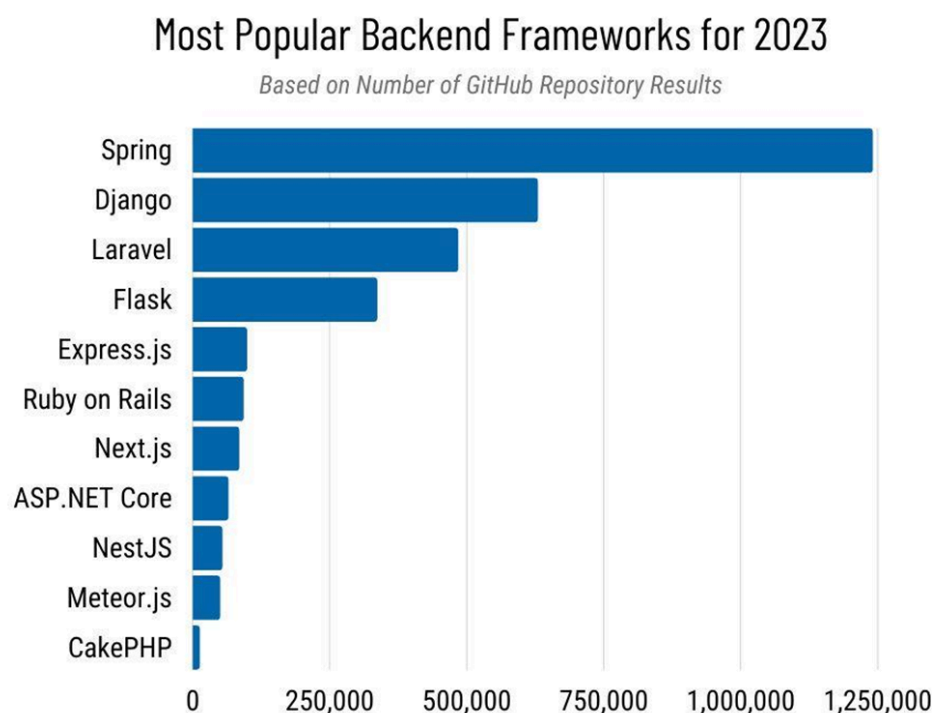
Un objetivo simple pero efectivo es mejorar el rendimiento de la búsqueda y filtro de las asignaturas sujetas a unos criterios impuestas por el usuario. Para esto, se va utilizar una base de datos basado en documentos (NoSQL) con el fin de mejorar la obtención de datos a la aplicación web.

El equipo de desarrollo estará conformado por cuatro integrantes, cada uno con habilidades específicas y experiencia en las tecnologías mencionadas. La selección de estas tecnologías se realizó considerando los conocimientos previos de cada miembro, la popularidad y la comunidad de soporte que las respalda, así como la disponibilidad de recursos e información en línea. Además, se tuvo en cuenta la

sinergia de las fortalezas individuales de cada miembro, garantizando que cada uno pueda aportar al proyecto desde su área de expertise.

Cada tecnología tiene un nivel de popularidad que varía según su complejidad y arquitectura, influyendo en la creación de aplicaciones robustas y seguras, así como en los lenguajes en los que están desarrolladas. En este tipo de proyectos, aspectos como la modularidad, la seguridad, la experiencia del usuario y una base de datos optimizada son clave para minimizar costos. Además, muchas de estas tecnologías ofrecen funcionalidades gratuitas y cuentan con una amplia disponibilidad de información en la web.

Basado en estadísticas para las tecnologías Back-end se tiene el siguiente gráfico:



Tomado de: <https://merehead.com/es/blog/marcos-de-back-end-2024/>

Para el desarrollo ágil de aplicaciones Java, Spring Boot sigue siendo una elección fuerte en los últimos años. Con un enfoque en la simplicidad y la facilidad de configuración, Spring Boot permite a los desarrolladores crear rápidamente aplicaciones empresariales robustas. La modularidad y la amplia comunidad de Spring brindan soporte y soluciones a una variedad de desafíos de desarrollo.

En cuanto a las tecnologías de Front-End, se destaca **Astro.js**, un generador de sitios estáticos que ha ganado popularidad gracias a su enfoque innovador y su capacidad para combinar la generación de sitios estáticos con la renderización del lado del servidor (SSR). Desarrollado por Fred K. Schott y un equipo de colaboradores, Astro comenzó como un generador de sitios estáticos y ha

evolucionado hacia un framework que ofrece un rendimiento excepcional y una experiencia de desarrollo optimizada.

Tomado:

<https://openwebinars.net/blog/astrojs-un-nuevo-paradigma-en-desarrollo-web/>

Algunos puntos importantes que impactan el desarrollo del software en un ambiente general son:





Experiencia técnica: Como principiantes en las tecnologías, es probable que tengamos habilidades técnicas aún en desarrollo, lo que puede aumentar la curva de aprendizaje al trabajar con tecnologías nuevas o complejas.











Disponibilidad de recursos: A menudo dependemos de recursos gratuitos o de código abierto, que pueden no ofrecer todas las funcionalidades necesarias.




Dependencia de revisiones o aprobaciones: Las decisiones clave suelen depender de la retroalimentación de profesores o tutores, lo que puede generar tiempos de espera. La falta de acceso rápido a expertos o mentores puede dificultar la resolución de problemas técnicos complejos.





Estimación de esfuerzo de Fibonacci:

Dado lo mencionado anteriormente, la estimación de tiempo para cada requerimiento funcional son los siguientes:

Requerimientos  Funcionales	Fibonacci
Inicio sesión 	La estimación de esfuerzo para este requerimiento se establece un número de 2 . Debido a aprender los diferentes proveedores de autenticación que proporciona framework.
Sección Búsqueda materias (Buscador) 	La complejidad va con el trato de los datos y los filtros necesarios, además en el front-end crear los otros apartados para cada materia encontrada. Por ende el número es 8 .
Sección búsqueda cursos de Coursera (Buscador) 	En este va relacionado el filtrado y el tratamiento de los datos en ambos apartados por ende el número es 8 .
Formulario especializado para búsqueda	El número que más se acerca es el 3 . Es bastante fácil construir el formulario, lo complicado es armar las consultas.

de materias. 	
Sistema de búsqueda y exploración de canales 	Al igual que los buscadores, interviene los filtros de datos sin comprometer mucho el rendimiento de la app por ende el número es 5 .
Crear canal (para admin de canal) 	El número que más relaciona la cantidad de esfuerzo, teniendo en cuenta la gestión, la administración del framework incluyendo las diferentes implementaciones para la modificación de la base de datos y la creación de relaciones, el número que mejor relaciona este esfuerzo es de 5 .
sección “mis canales” 	La complejidad en este caso va de hacer el filtrado de datos y las secciones de “unir” y luego hacer las modificaciones pertinentes por ende el número es 8 .
Descripción y temario de las asignaturas. 	Esta parte es sencilla, solo se busca la información y extraerla de la base de datos por ende el numero seria el 1 .
Crear anuncio/publicación en el feed del canal (para admin) 	Además que el administrador agregue información por el mismo, también podrá recurrir a la base de datos, la complejidad crece que tanta información quiere publicar y el tratamiento de la misma en el desarrollo por ende el número es 8 .
Unirse a un canal 	Las funcionalidades para este caso son bastante mínimas, como hacer la relación y el front traer el grupo a su apartado de grupos 3 .
Chat grupal (para canales) 	El nivel de complejidad depende de la tecnología, para este caso se dice que el número 13 . Además considerando la sincronización de mensajes desde varios ordenadores.
Cerrar sesión 	Así como guardar las sesiones, también se pueden destruir por ende el número es 1 .
Agregar cursos de Coursera a los favoritos 	La principal complejidad radica en el diseño de la interfaz, enfocándose en la experiencia del usuario, por lo que el número 8 juega un papel clave. Además, es fundamental que la información esté siempre disponible desde el servidor cuando el usuario acceda a la sección "Favoritos".

Sistema de notificaciones (de canales) 	La búsqueda de los módulos compatibles además de combinar la fácil implementación será un reto por ende el número es 3 .
Agregar administradores 	La principal complejidad técnica radica en la gestión de permisos, ya que al asignar un nuevo administrador, este deberá heredar todas las autorizaciones y capacidades del administrador existente. Por ende el número es 8 .
Expulsar miembros 	La complejidad en este requerimiento va con la definición de los filtros sin afectar el rendimiento de la app por ende el número es 5 .

Requerimientos NO funcionales 	Fibonacci
Usabilidad 	Es fundamental considerar el contexto del público objetivo, priorizando la facilidad de uso y una experiencia intuitiva. Por ende es 21 .
Rendimiento ⚡ 	La optimización del código, la sustitución de algoritmos y la gestión eficiente de la memoria representan uno de los mayores desafíos, por lo que se asigna el número 21 .
Escalabilidad 🌐 	Depende de las nuevas funcionalidades por ende el número es 34 .

4. Análisis de gestión de software

Alcance:

Para el alcance estimado del sistema, las funcionalidades que serán incluidas serán las clasificadas en Must y algunas del Should.

De manera que **serán incluidas en el alcance**:







- Inicio de sesión.
- Cerrar sesión.
- Búsqueda de materias y cursos (Buscador).
- Descripción y temario de las asignaturas.
- Formulario especializado para búsqueda de materias electivas.
- Sistema de búsqueda y exploración de canales.
- Creación de canales.
- Unirse a un canal
- Sección "Mis canales".
- Chat integrado






Por otro lado las que no se tienen previstas como parte del producto mínimo viable son: (En el orden en el que se podrían incluir si se alcanza el mvp con tiempo)







- Crear anuncios/publicaciones en el feed del canal (para administradores).
- Agregar administradores.
- Sistema de notificaciones de canales.
- Expulsar miembros.
- Agregar cursos de Coursera a favoritos.





Tiempo:





La siguiente tabla relaciona el tiempo con la diferentes descripciones de los requerimientos tanto funcionales como no funcionales:

Requerimientos Funcionales 	Descripción 	Estimación de tiempo 
Inicio sesión   	Basado en el inicio concurrente, para la app se va a tratar de que los usuarios inicien por su correo institucional de la UNAL. Este requerimiento requiere más trabajo por parte del servidor, aplicando filtros para darles los permisos necesarios.	4 días.

Sección Búsqueda materias (Buscador) 	<p>Una de las funcionalidades principales de la aplicación, este requerimiento va con el fin de ayudar a los estudiantes a buscar materias por menor entrada de parámetros. Este buscador, además de proporcionar la información de la materia, ayuda con una información detallada para ayudar a los estudiantes. Requiere del trabajo del back para la información y el Front para organizar los datos de tal manera que el usuario sea cómodo y fácil de entender.</p>	2-3 semanas
Sección búsqueda cursos de Coursera (Buscador) 	<p>Una extensión del buscador común, este se encarga de buscar cursos complementarios dependiendo de la necesidad del usuario, estará conectado con el convenio de la universidad. Para este caso requiere trabajo del back para aplicar el filtro de los datos y el front para darle la presentación al usuario.</p>	2-3 semanas
Formulario especializado para búsqueda de materias. 	<p>Utilizado para el modelo de consultas que tendrá que hacer el back-end. El usuario podrá buscar sus cursos dependiendo de las necesidades específicas. El front-end proporciona los campos y el back-end se encarga de construir la consulta.</p>	1-2 semanas
Sistema de búsqueda y exploración de canales 	<p>Se busca desarrollar una herramienta que permita a los estudiantes encontrar espacios específicos para ampliar sus conocimientos en un área de interés. Para ello, se implementará un backend encargado de la conexión con la base de datos y un frontend diseñado para presentar la información de manera accesible al usuario.</p>	1-2 semanas
Crear canal (para admin de canal) 	<p>Las personas interesadas en crear espacios para contribuir al conocimiento de los estudiantes podrán generar canales para compartir información sobre sus grupos, así como establecer un entorno donde los participantes puedan intercambiar conocimientos de manera efectiva.</p> <p>Este requerimiento requiere del back para hacer las respectivas relaciones en el</p>	4 días

	modelo de la base de datos.	
sección “mis canales” 	Parte específica que funciona por sesiones donde los usuarios podrán encontrar los canales en donde se unieron. Requiere trabajo del back para extraer las relaciones en el modelo de base de datos.	2 días
Descripción y temario de las asignaturas. 	Cuando el usuario utilice el buscador de materias de la Universidad, podrá acceder a una breve descripción junto con un temario, facilitando así su comprensión y mejorando su experiencia académica.	1 día
Crear anuncio/publicación en el feed del canal (para admin) 	Va con el objetivo de proporcionar o dar más contexto del canal a los participantes, facilitando la comprensión de la meta que tiene el canal. Para este caso se requiere trabajo para el front para los campos o un apartado para el ingreso de información para formar la publicación, por parte del back se hace las respectivas validaciones para que no todos tenga esa funcionalidad. Además, de guardar eso en la base de datos.	5 días
Unirse a un canal 	Será un pequeño botón donde el usuario podrá inscribirse al canal de su preferencia. El front proporciona el botón y el back se encargará de hacer la relación en la base de datos.	2 días
Chat grupal (para canales) 	Para este caso, se debe realizar primero una verificación del usuario al que se le enviará la información, asegurando que solo las personas que forman parte del grupo tengan acceso a los recursos correspondientes, mientras que el resto no podrá acceder. Un segundo aspecto clave es la sincronización de los sockets entre varios usuarios, así como la asignación de cada usuario a un grupo específico. Además, se debe permitir que el servidor añada imágenes o texto a las publicaciones.	5 días
Cerrar sesión 	La plataforma tendrá el tipo cierre de sesión, para proteger los datos del	1 día

	usuario.	
Agregar cursos de Coursera a los favoritos 	Esta funcionalidad está basada en la experiencia del usuario. Si el usuario podrá encontrar sus grupos que más le parecieron interesantes con solo un click.	3 días.
Sistema de notificaciones (de canales) 	Notificar a los usuarios de nuevas novedades a los usuarios inscritos. Por lo general hay módulos o librerías que realizan todo el trabajo pero en este caso se debe hacer una configuración adicional y es que debe sincronizar con otras funcionalidades de la aplicación, por ende se estima un total de 3 días.	3 días
Agregar administradores 	Esto va con la finalidad de compartir el mando de los grupos siempre y cuando los administradores lo consideren necesario. En este apartado del back-end tendrá que hacer los filtros y la modificación en la base de datos.	4-5 días
Expulsar miembros 	Sirve para mejorar la experiencia en los grupos, los administradores tendrán la posibilidad de expulsar si uno de los participantes no cumple con las reglas impuestas por los administradores. En este caso el back-end tendrá que hacer el update a la base, además poner sanciones como que el usuario nunca podrá entrar al grupo.	3 días.

Requerimientos NO funcionales 	Descripción 	Estimación de tiempo 
Usabilidad 	La usabilidad depende principalmente del trabajo del equipo frontend y de la creatividad del equipo en general. El objetivo es implementar una interfaz intuitiva que mejore la experiencia del usuario, asegurando que sea amigable mediante el uso de una paleta de colores adecuada,	1-2 semanas

	estructuras bien definidas y otros elementos visuales coherentes.	
Rendimiento ⚡🚀🔄📈	Para este requerimiento se necesita un revisión general del código, tanto de las librerías tanto del servidor o como el front, algoritmos utilizados para el tránsito de información en el software, etc. Esto requiere bastante tiempo ya que se deben hacer modificaciones y tener en cuenta la aparición de errores	3 semanas
Escalabilidad 🌐📊🔧📶	En este apartado, la estimación de tiempo se deja abierta, ya que dependerá de las necesidades futuras del proyecto y de su posible expansión dentro de la Universidad. El alcance del mismo puede variar según los requisitos adicionales que surjan durante su desarrollo	No aplica.

Costos:

Otro aspecto crucial son los costos asociados al desarrollo del proyecto. Aunque las tecnologías elegidas ofrecen numerosos recursos gratuitos, existen ciertos pilares fundamentales que generan costos adicionales. Estos incluyen los requisitos externos, como las herramientas de testing, servicios en la nube, herramientas de trabajo como IDEs, pruebas en servidores, el despliegue de la aplicación, y los costos relacionados con el uso de bases de datos, como en el caso de Firebase. Además, es necesario considerar los costos de personal, como el salario de los desarrolladores, diseñadores y otros roles involucrados en el proyecto, así como los gastos asociados a la infraestructura necesaria para el mantenimiento y operación del sistema. Es importante tener en cuenta estos gastos adicionales a lo largo del proceso de desarrollo para garantizar una planificación financiera adecuada y una ejecución eficiente del proyecto.

La siguiente tabla relaciona algunos costos aproximados en el desarrollo del proyecto:

Servicio	Prestador	Costo
Servicios Back-end	Desarrollador Junior back-end	Un desarrollador junior en Colombia estaría ganando un total de 2.275.000

		pesos colombianos mensuales.
Creación de interfaces.	Desarrollador Junior Front-end	Un desarrollador junior enfocado en el Front-end estaría ganando 1.650.000 pesos colombianos mensuales.
Servicios en la nube	Oracle OCI	Por el framework existen apartados en Oracle OCI que no tiene un costo adicional, sin embargo para el despliegue de aplicaciones web (kubernetes) si se quiere en un futuro el mantenimiento de la app se requiere un costo de \$415,00 cluster por hora. Además si el proyecto posee una API importante para su funcionamiento se requiere un costo de \$12.450,00 por millón de llamadas. Servicio serverless para ejecutar funciones sin administrar servidores se tiene funciones propia de Oracle y se requiere un costo de 0,0000002 \$ por las invocaciones de funciones.
Base de datos	Firebase	Para el almacenamiento de datos se tiene un costo de \$0.156 GIB/mes. Además para servicios para el despliegue de base de datos el primer GB de almacenamiento es gratuito; a partir de ahí, se cobra \$0.026 USD por GB al mes.
Herramientas de desarrollo y entornos de ejecución.	JetBrains	Para mejorar el desarrollo podemos utilizar la versión ultimate de los entornos de la empresa, pero para esto se debe pagar alrededor de 599 dólares para desbloquear la versión.
Especialización UX/UI	Diseñador UX/UI	Para esta profesión se tiene un salario promedio de 3.629.444 pesos colombianos mensuales.
Diseño de UX/UI y prototipado	Figma, Adobe XD, Google Analytics...	
Testing de aplicaciones Web	TestDevLab	La página no proporciona tarifas fijas. Todo depende de las necesidades del equipo.

Todos los precios mencionados son aproximaciones cercanas a la realidad. En cualquier proyecto de ingeniería de software, es fundamental realizar un estudio detallado de la inversión inicial, así como de los costos anuales asociados al desarrollo y mantenimiento. Además, es importante considerar escenarios futuros en los que pueda ser necesario realizar inversiones adicionales o recurrir a financiamiento externo para la expansión, actualización o escalabilidad del sistema.

Una planificación financiera adecuada permitirá anticiparse a estos posibles costos y garantizar la sostenibilidad del proyecto a largo plazo.

Parte 2

8. Diseño y Arquitectura:

- Arquitectura del sistema:

En el desarrollo de software, la elección de una arquitectura adecuada es un factor clave para el éxito de cualquier proyecto. Definir la estructura del sistema desde el inicio permite no solo satisfacer las necesidades actuales de la entidad, sino también garantizar su evolución a largo plazo.

Seleccionar una arquitectura bien diseñada ofrece múltiples ventajas, entre las que se destacan:

- **Escalabilidad:** Permite que el sistema crezca y se adapte a un aumento en la demanda sin comprometer su rendimiento.

- **Seguridad:** Facilita la implementación de buenas prácticas para proteger los datos y la integridad del software.
- **Modularidad:** Promueve una estructura organizada y flexible, facilitando el mantenimiento y la incorporación de nuevas funcionalidades.
- **Legibilidad del código:** Hace que el código sea más comprensible para otros desarrolladores.
- **Facilitación del trabajo en equipo:** Al seguir principios bien definidos, la colaboración entre diferentes áreas del desarrollo se vuelve más eficiente.

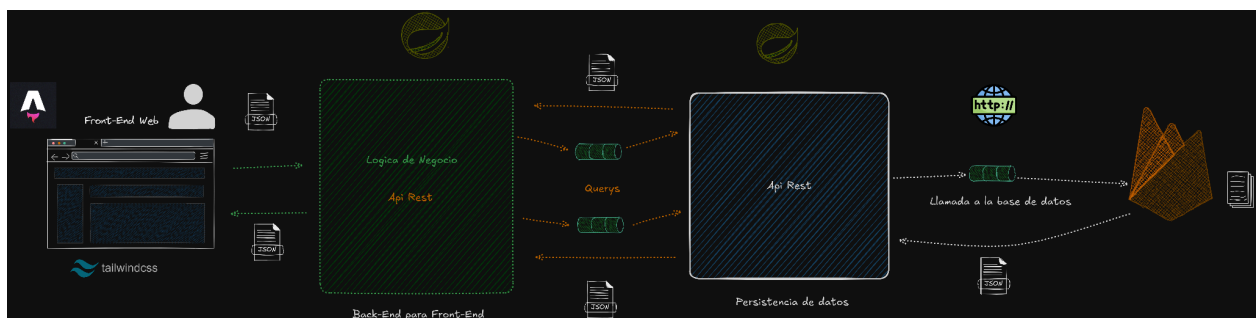
Una arquitectura bien pensada no solo optimiza el desempeño del software, sino que también mejora la productividad del equipo y reduce costos a largo plazo.

Para el proyecto, se ha optado por una arquitectura de microservicios, implementando una separación de responsabilidades basada en la filosofía de la arquitectura en capas. Cada módulo del sistema tendrá un propósito específico y, en conjunto, contribuirán al funcionamiento global de la aplicación web.

La comunicación entre estos módulos, así como con el **Front-end**, se realizará a través del protocolo **HTTP**, un estándar ampliamente utilizado en la industria para la conexión y el intercambio de información entre los diferentes componentes del sistema. Esta elección garantiza compatibilidad y facilidad de integración con otras tecnologías y servicios externos.

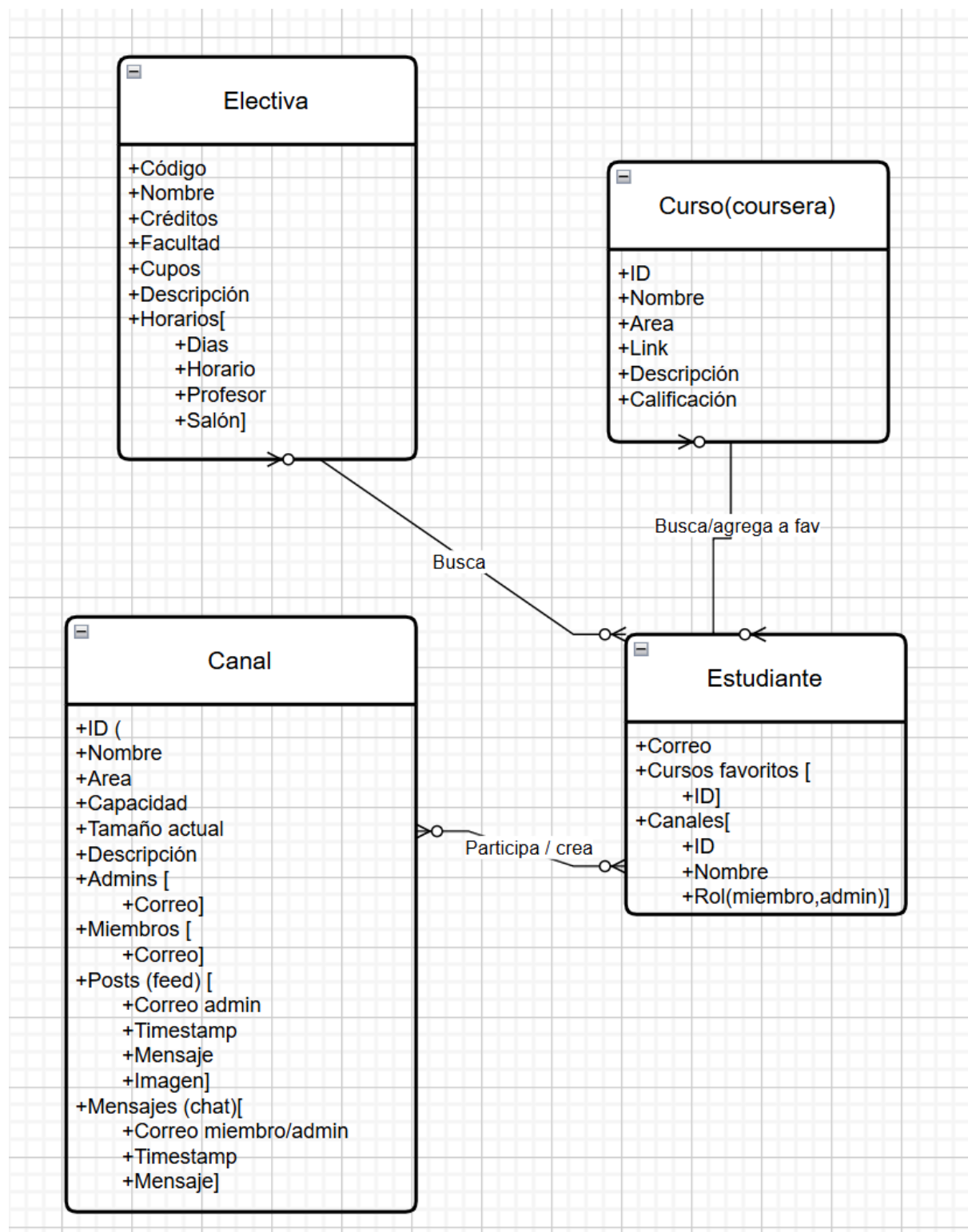
A continuación se muestra un gráfico de lo que se quiere implementar:

Link: <https://excalidraw.com/#room=8a629d936d45da706fba.ewCVZ5Bm2ZRI87GLZYEbRg>



En este caso solo se hará un microservicio y es el común en muchos, es la persistencia de datos, solo tendrá la función de hacer los CRUD dependiendo de las peticiones enviadas por el módulo "**Back-end para front-end**". El objetivo de esta separación es **reducir la complejidad del proyecto**, permitiendo un desarrollo más ágil y modular. Al delegar la persistencia de datos a un servicio independiente, se logra una mejor organización del código, mayor escalabilidad y la posibilidad de cambiar o mejorar la base de datos sin afectar al resto de la aplicación.

- **Diseño de bases de datos:**



Explicar las decisiones tomadas en el diseño del esquema de la base de datos (por ejemplo, normalización, índices, claves primarias y foráneas).

En el diagrama representamos gráficamente la estructura de nuestra base de datos. Aunque no es un diagrama de entidad-relación tradicional, ya que la base de datos no es relacional, buscamos mostrar las relaciones entre las diferentes colecciones.

El estudiante es el elemento central de la base de datos, ya que interactúa con los demás componentes. Además, incluimos referencias a otras colecciones mediante claves foráneas, evitando duplicación de datos. En estos casos, sólo almacenamos la clave primaria del objeto referenciado para recuperar su información desde su colección principal.

Para la identificación de los datos:

- **Electivas:** Se identifican mediante el *Código del SIA*, ya que es único.
- **Estudiantes:** Se identifican con su *correo electrónico*, un dato obligatorio y no repetido.
- **Cursos y canales:** Se les asigna un *ID único* generado.

Además, la base de datos incluye estructuras anidadas, como los mensajes en los chats y las publicaciones en el feed de los canales. Esta organización permite almacenar el historial completo sin necesidad de crear colecciones adicionales para cada canal.

Justificación:

Elegimos utilizar una base de datos no relacional en Firebase por varias razones según la estructura de nuestro proyecto. En primer lugar, Firebase ofrece bastante flexibilidad, permitiendo que cada documento pueda tener una estructura diferente y campos opcionales. Por ejemplo, un estudiante puede tener un conjunto de cursos favoritos que no necesariamente tienen todos los campos de otros estudiantes, y esto no requiere que modifiquemos el esquema global de la base de datos.

Además, Firebase facilita el almacenamiento de datos anidados, lo cual es especialmente útil en casos como los canales, donde cada uno tiene un chat único con mensajes que varían en contenido y fecha. En un modelo relacional, estos datos tendrían que dividirse en múltiples tablas y relaciones, pero con Firebase, podemos almacenar el historial de mensajes dentro del propio canal de forma eficiente, sin la necesidad de complejas uniones entre tablas. Esto no solo simplifica la estructura de la base de datos, sino que también mejora el rendimiento de las consultas al reducir la cantidad de operaciones necesarias para acceder a datos relacionados.

Por otro lado, este tipo de bases son más adaptables para escalarlas, lo que nos permite manejar grandes cantidades de usuarios, cursos y actividades sin complicaciones. A medida que el número de estudiantes, canales y mensajes crezca, Firebase puede distribuir y sincronizar estos datos de manera más eficiente.

Por último, Firebase permite la transferencia de datos por medio del formato JSON lo que facilita bastante las tareas de subir los datos a la base y también de ir a buscarlos allí ya que los datos que almacenamos en Firebase se pueden utilizar directamente en nuestra aplicación sin necesidad de realizar conversiones adicionales.

En resumen, Firebase nos permite una modelización flexible, eficiente y escalable de los datos de nuestro proyecto, lo que lo convierte en la opción ideal para manejar la información de estudiantes, cursos y canales en tiempo real.

9. Patrones de Diseño

Puesto que por el momento no contamos con código para el backend la identificación de patrones usados no es realmente posible, pero como grupo hay ciertos patrones que creemos serán convenientes de usar una vez tengamos dentro del proyecto el código funcional. Por otro lado, contamos con una parte del código del Frontend, del que aplicamos 1 patrón de Diseño.

Front

Composite Pattern

- **¿Qué problema resuelve?**

El Composite Pattern resuelve el problema de manejar estructuras jerárquicas y modulares en interfaces de usuario, permitiendo que los elementos de la UI (como botones, formularios y tarjetas) sean tratados de manera uniforme, independientemente de si son elementos individuales o grupos de elementos.

Este patrón facilita la reutilización de componentes, simplifica la gestión de la interfaz y mejora la mantenibilidad del código.

- **¿Por qué fue necesario en el proyecto?**

El proyecto tiene múltiples secciones con elementos visuales organizados jerárquicamente, como:

- Buscador de cursos con filtros, tarjetas de cursos.
- Canales de difusión con publicaciones, notificaciones y un chat grupal.








Sin Composite Pattern, el código de la interfaz sería más difícil de mantener, ya que cada componente estaría gestionado individualmente en lugar de formar parte de una estructura flexible y reutilizable. De manera que aplicando este patrón se logra modularidad, donde cada sección es un conjunto de componentes reutilizables, y escalabilidad en donde se pueden agregar nuevas funcionalidades sin afectar la estructura principal.

- **¿Cómo se implementó?**

Este se implementó usando “Astro.js” y de manera que el código estuviera organizado en una especie de capas y componentes como se puede ver en las imágenes.











[learnhub-front](#) / [src](#) / 

 **LozanoJohan** feat: added reviews to courses, ended course page & added mock db ✓

Name	Last commit message
 ..	
 assets	Initial commit from Astro
 components	feat: added reviews to courses, ended course page & added mock db
 data	feat: added reviews to courses, ended course page & added mock db
 layouts	feat: add login and started course page
 models	feat: added reviews to courses, ended course page & added mock db
 pages	feat: added reviews to courses, ended course page & added mock db




[learnhub-front](#) / [src](#) / [components](#) / 

 **LozanoJohan** feat: added reviews to courses, ended course page & added mock db ✓

Name	Last commit message
 ..	
 courses	feat: added reviews to courses, ended course page & added mock db
 login	feat: add login and started course page
 search	feat: add login and started course page
 AddReviewModal.astro	feat: add login and started course page
 Footer.astro	feat: add login and started course page
 Navbar.astro	feat: added reviews to courses, ended course page & added mock db
 RelatedCourses.astro	feat: add login and started course page
 Reviews.astro	feat: added reviews to courses, ended course page & added mock db
 ReviewsDisplay.astro	feat: added reviews to courses, ended course page & added mock db

[learnhub-front](#) / [src](#) / [components](#) / [search](#) / 

 **LozanoJohan** feat: add login and started course page ✓

Name	Last commit message
 ..	
 SearchSection.astro	feat: add login and started course page
 Searchbar.astro	feat: add search and card mockups

Back

Strategy

- ¿Qué problema resuelve?

El sistema de búsqueda de cursos necesita permitir a los usuarios aplicar diferentes filtros, de manera que la lógica será la misma pero la implementación para cada filtro debe ser distinta

- Cada usuario puede querer aplicar un solo filtro o combinar varios.
- Es necesario que los filtros sean flexibles y modificables en tiempo de ejecución sin alterar la lógica del buscador.

- Se requiere un diseño modular y reutilizable para evitar una gran cantidad de condicionales dentro de la lógica de búsqueda que se usaron por ejemplo en el Factory Method.

El patrón Strategy resuelve este problema al encapsular cada criterio de filtrado en una clase separada, permitiendo que el buscador cambie de estrategia dinámicamente sin afectar su implementación interna.

- **¿Por qué es necesario en el proyecto?**

En este sistema, los estudiantes pueden buscar cursos con distintos criterios y activar o desactivar filtros según sus preferencias.

- Sin Strategy, tendríamos un código con múltiples condicionales, lo que dificultaría la escalabilidad y mantenimiento.
- Con Strategy, cada filtro es independiente y se puede añadir o modificar sin afectar el buscador.
- Además, este enfoque permite a los usuarios cambiar los filtros en tiempo de ejecución sin necesidad de modificar el código del buscador.

Este patrón es clave para mantener la modularidad, flexibilidad y extensibilidad del sistema.

- **¿Cómo se implementaría?**

- Se definiría una **interfaz genérica de filtro** que estandariza la aplicación de cualquier criterio de búsqueda.
- Se crearían **clases concretas** para cada tipo de filtro (por créditos, temática, etc.), cada una implementando su propia lógica.
- Se diseñaría una **clase de búsqueda de cursos**, que permita seleccionar dinámicamente el filtro deseado y aplicarlo a la lista de cursos.
- En la página, los usuarios podrán elegir un filtro (o combinarlos) y realizar la búsqueda sin que el sistema requiera modificaciones internas.

Factory Method

- **¿Qué problema resuelve?**

El problema principal es que existen tres buscadores diferentes (cursos, electivas y actividades extracurriculares), los cuales utilizan un mismo método de búsqueda, pero consultan bases de datos distintas. Implementar cada buscador por separado implicaría duplicar código, lo que aumentaría la complejidad y dificultaría el mantenimiento.

Además algunos buscadores incluyen filtros mientras que otros no, por lo que se requiere una solución flexible que permita manejar estas diferencias sin generar estructuras innecesariamente rígidas.

- **¿Por qué es necesario en el proyecto?**

El sistema debe ofrecer una experiencia de búsqueda optimizada, con la capacidad de filtrar información en algunos casos y presentar resultados organizados. Dado que cada tipo de búsqueda tiene requisitos similares en términos de funcionalidad pero distintos en términos de origen de datos y filtros, es necesario porque:

- Se necesita una forma de estructurar el código que permita reutilizar la lógica común sin duplicar código y asegurando al mismo tiempo que cada buscador acceda a su respectiva fuente de datos.
 - El Factory Method permite instanciar el tipo de buscador adecuado según la necesidad del usuario, sin acoplar el código a una implementación específica. Esto facilita la escalabilidad del sistema.
- **¿Cómo se implementaría?**
 - Se definiría una clase base o interfaz para los buscadores, estableciendo un contrato común con métodos esenciales como buscar().
 - Se implementarían clases concretas para cada tipo de búsqueda (cursos, electivas y actividades extracurriculares), cada una con su propia fuente de datos y criterios de búsqueda.

Un Factory Method se encargaría de instanciar dinámicamente el tipo de buscador correspondiente según la pestaña activa en la interfaz de usuario. Cuando un usuario accede a una pestaña de búsqueda, el sistema invoca el Factory, que crea y devuelve la instancia adecuada sin que el frontend tenga que preocuparse por los detalles de implementación.