Universidad de Las Américas

Facultad de Ingenierías y Ciencias Agropecuarias

Ingeniería de Softwate

Informe de laboratorio

1. **DATOS DEL ALUMNO: Camila Cabrera**

2. **TEMA DE LA PRÁCTICA:**

Curso Fundamentos en DevOps, APIs y Arquitectura de Microservicios

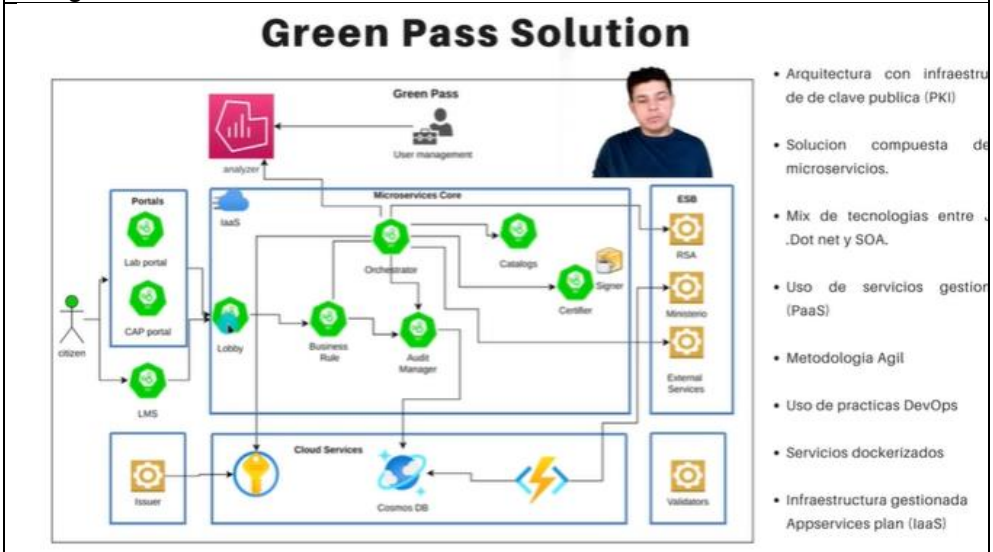
3. **OBJETIVO DE LA PRÁCITCA**
   - Culminar el curso sobre DevOps, entendiendo las bases y el caso aplicado (Pases de vacunas en la Unión Europea)

4. **OBJETIVOS ESPECÍFICOS**
   - Visualizar el curso
   - Realizar los ejercicios propuestos
   - Leer el material

5. **MATERIALES Y MÉTODOS**
   - Curso Udemy

6. **DESARROLLO DE LA PRÁCTICA Y RESULTADOS**

Sección 1: Antes de empezar: ¿Cómo es un proyecto real con Microservicios y DevOps?

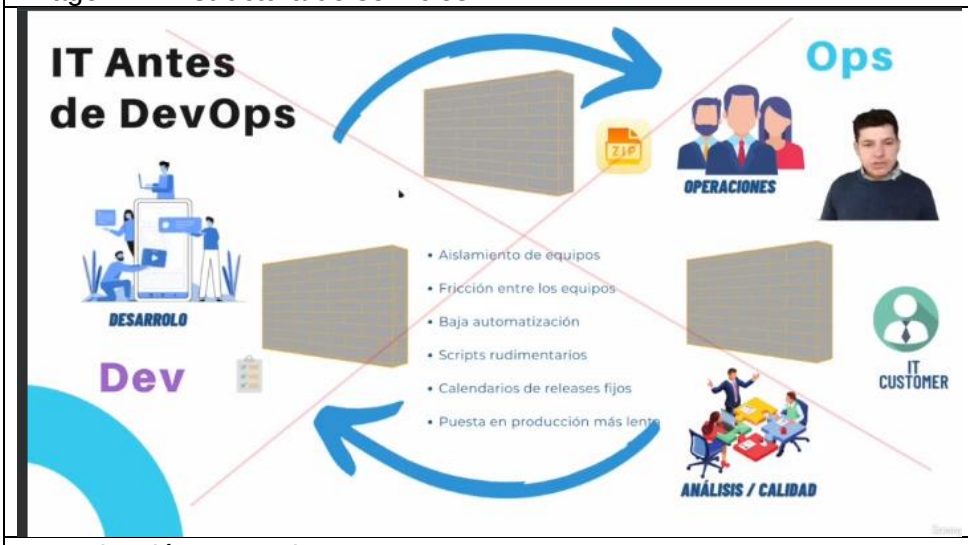
Imagen #1: Estructura de servicios

Estructura de la aplicación para pase de COVID-19

Durante esta sección se explicó que:

- El proyecto se planteó para la implementación de un pasaporte COVID en la unión europea, que certifica que una persona está libre de COVID-19, permitiéndole viajar y acceder a diferentes servicios. Este pasaporte se emite si la persona cumple con criterios como haber superado la enfermedad, estar vacunado o dar negativo en pruebas recientes. La implementación del proyecto implica la interacción con múltiples entidades y sistemas de información, con el objetivo de cumplir con regulaciones y emitir certificados de manera eficiente.
- En el proyecto optaron por una metodología ágil como DevOps y la arquitectura de microservicios para permitir el desarrollo rápido, iterativo y eficiente del sistema.
- Este pasaporte buscaba certificar a una persona respecto al COVID-19, permitiendo su acceso a diversos servicios y lugares.
- Se manejo una infraestructura de clave pública, donde las entidades emisoras generan llaves privadas para firmar los certificados, mientras que las llaves públicas se utilizan para validar la autenticidad de los certificados en repositorios centrales de la UE.
- Cada aplicación posee su propia funcionalidad específica y se comunica a través de APIs expuestas por microservicios independientes.

Sección 2: Fundamentos en DevOps, APIs y Arquitectura de Microservicios

Imagen #2: Estructura de servicios



Organización grupo de TI

Durante esta sección primero se explicó el índice de los temas:

- Fundamentos y su importancia en los proyectos.
- Definición de DevOps y sus características clave.
    - Uso de repositorios de Git y las metodologías de implementación como Tramp Boys y Git Flow.
- Análisis de la estructura típica de los departamentos de tecnología, resaltando roles específicos y desafíos comunes, como la falta de colaboración entre equipos y la necesidad de mantener la estabilidad en producción.

Problemas de departamentos de tecnología

- Aislamiento entre equipos: Los equipos están centrados en sus propias responsabilidades y no colaboran eficientemente. Esto conduce a fricciones y falta de comunicación.
- Bajo nivel de automatización: Los procesos son no eficientes. Hay falta de estandarización y documentación en los scripts utilizados para la automatización.
- Lentitud en los procesos de puesta en producción: Los procesos de despliegue suelen ser lentos debido a metodologías poco eficientes, como la planificación anual

basada en un calendario de lanzamientos predefinidos, lo que limita la flexibilidad y la velocidad en la implementación de cambios.

- DevOps es una metodología que busca unir a los equipos de desarrollo y operaciones dentro del departamento de tecnología, incentivando la colaboración para lograr resultados certeros.
- Se define como la unión de personas, procesos y tecnologías con el objetivo de aportar valor de forma continua a los clientes.
- Seleccionar herramientas de trabajo colaborativo, como Jira, Confluence, Trello, Microsoft Teams o Slack, que permitan una comunicación fluida y una gestión eficiente de tareas.
- Para establecer una vía de trabajo eficiente en DevOps, es necesario elegir una metodología de gestión de repositorios para el código, como GitFlow o Trunk-Based Development (TBD). Estas metodologías proporcionan estructuras y prácticas para administrar ramas y avances de trabajo.

Estilos de despliegue:

- Canary: Se despliega una nueva versión de la aplicación en un entorno paralelo.
- Blue Green: Redirección del tráfico se hace de manera completa en un momento determinado, evitando la necesidad de una transición gradual. Si hay problemas, se puede volver al entorno anterior fácilmente.
- Rolling Update: Ideal para entornos clusterizados

Sección 3: Fundamentos en microservicios

En esta unidad se identificaron los siguientes puntos:

- Los microservicios dividen la aplicación en componentes independientes que se ejecutan de forma sin dependencias y se comunican entre sí a través de interfaces. Los microservicios son una alternativa al enfoque monolítico en el desarrollo de software.
- Las principales características de los microservicios incluyen la independencia de implementación, el despliegue y la escalabilidad individual de cada servicio. También incluye la capacidad de utilizar diferentes tecnologías y herramientas para cada microservicio.
- Esto facilita la evolución y el mantenimiento de la aplicación a medida que los requisitos cambian considerar la escalabilidad para adaptarse al crecimiento del negocio y la demanda de usuarios. La escalabilidad se puede lograr mediante el escalado vertical, agregando recursos a una única máquina, o mediante el escalado horizontal, agregando nodos adicionales a la infraestructura.

- Cada microservicio expone su funcionalidad a través de una interfaz de programación de aplicaciones (API).

- Si un microservicio necesita interactuar con otro, lo hace exclusivamente a través de sus APIs

Sección 4: Diseño de soluciones de microservicios

- Se inicia con identificar los Building Blocks, que son los principales componentes de la solución y cómo se relacionan entre sí.
- Se usa diagramas de diseño para asegurar que todos tengan la misma comprensión de la solución que se está diseñando.
- Es importante entender el concepto de TDD o Design Driven Domain, que implica diseñar soluciones guiadas por los dominios y contextos delimitados de la aplicación. Este enfoque estructurado se basa en identificar las necesidades de los dominios y diseñar una solución que las satisfaga de manera óptima.
- Un dominio hablando de microservicios es el campo de acción objetivo.
- En la fase de análisis táctico del diseño, se delimitan los dominios y se identifican las entidades específicas, así como los microservicios que las manejarán.
- Cada microservicio tiene su propia entidad de datos específica para el dominio al que pertenece, y se comunica con su entorno a través de APIs.
- Tipos de microservicios son:
    - Servicios de entidades: Representan tablas maestras o de configuración
    - Servicios de composición: Se encargan de combinar información de múltiples sistemas o microservicios
    - Servicios de procesos: Realizan procesos específicos cuando se invocan con ciertos datos
    - Servicios de utilidades: Contienen funcionalidades comunes para todo el sistema, como configuraciones de acceso a bases de datos.
- Los patrones de diseño son fundamentales para crear soluciones con microservicios.
- Los patrones de microservicios se centran específicamente en el diseño de soluciones con arquitectura de microservicios
    - Patrón Event-driven: se basa en la comunicación entre microservicios a través de eventos.
    - Patrón Saga: Este patrón se utiliza para mantener la consistencia en operaciones distribuidas y transacciones entre múltiples microservicios.

Sección 5: Apis conceptualización

- Las APIs son interfaces de comunicación que permiten que los productos y servicios se comuniquen sin necesidad de conocer los detalles de su implementación.
    - SOAP: Es un protocolo basado en XML y orientado a servicios que define reglas y estándares para la construcción de APIs.
    - REST: No es un protocolo, sino un estilo de arquitectura que utiliza HTTP para la comunicación entre servicios. Permite trabajar con diferentes tipos de mensajes (JSON, XML, texto, binarios) y se basa en los verbos HTTP estándar (GET, POST, PUT, DELETE). Swagger/OpenAPI es una especificación comúnmente utilizada para diseñar APIs RESTful.
- API First se refiere a definir primero la API independientemente del lenguaje de programación utilizando una notación como JSON, y luego generar el código a partir de esta definición.
- API GETWAY componente de software que se sitúa entre los clientes y los microservicios, estableciendo políticas de seguridad y filtrando el tráfico de las peticiones.
- OIDC (OpenID Connect) se encarga de la autenticación, mientras que OAuth2 se centra en la autorización. Estos protocolos trabajan en conjunto para validar la identidad de los usuarios y determinar sus permisos de acceso a los recursos protegidos.
- OAuth2 es fundamental en el mundo de las APIs y los microservicios para implementar sistemas de seguridad.

Sección 5: Sección de práctica.

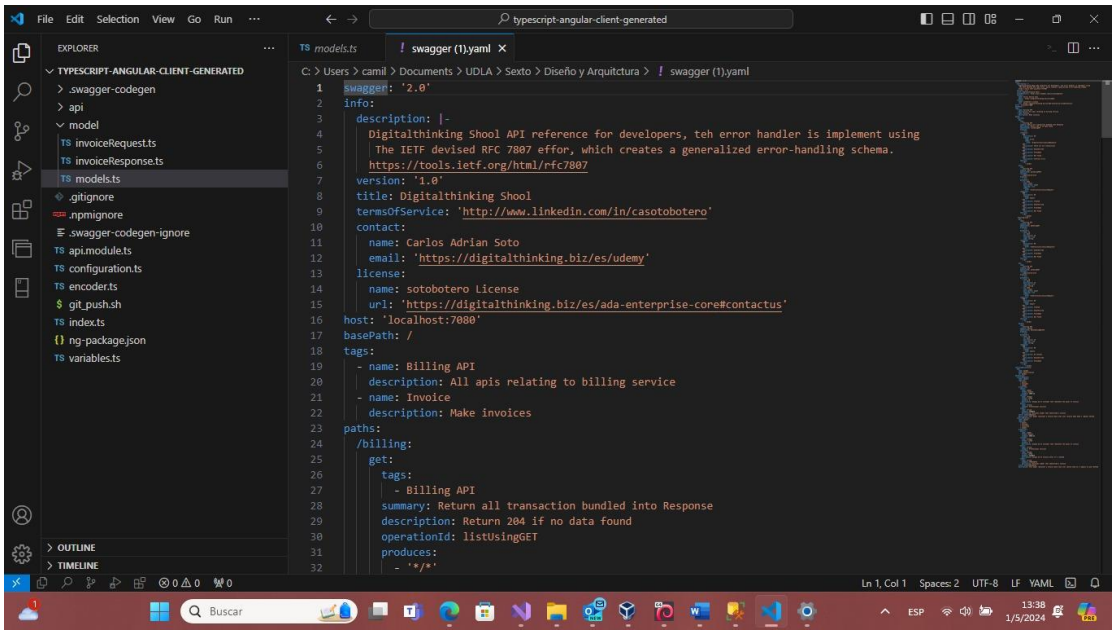
Durante esta sección se realizaron dos ejercicios para poner en práctica el conocimiento teórico.

- Para facilitar la definición de la API, se pueden utilizar herramientas como un editor de YAML. Y se deberían seguir los siguientes pasos:
    - Identificar los requisitos de la API
    - Definir las rutas (pads) y parámetros
    - Describir los componentes y modelos: Defina los tipos de datos y las estructuras de los objetos.

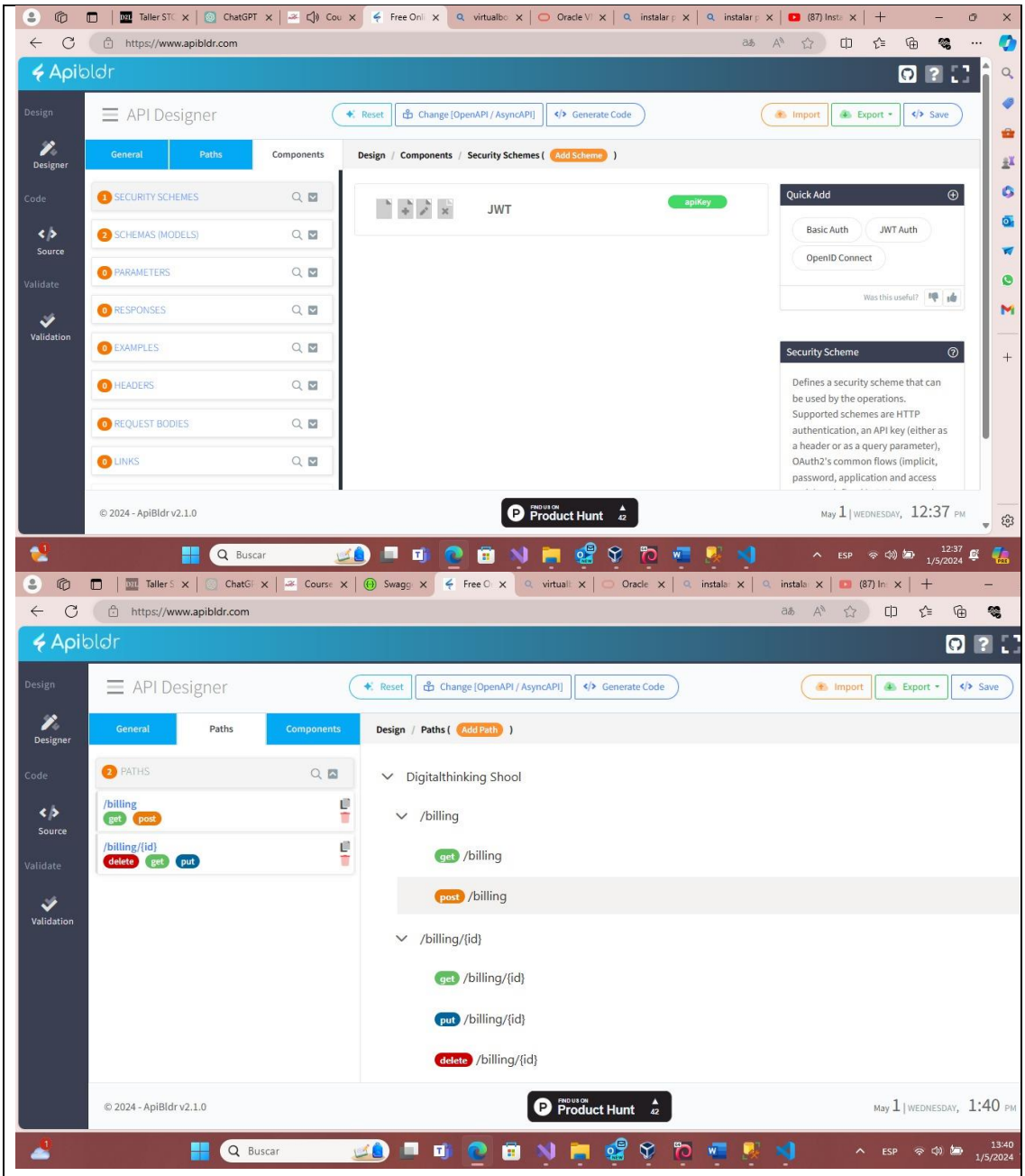
- Documentar la API: Proporcione documentación clara y detallada para la API, incluyendo la descripción de cada ruta.

- **RETO PRÁCTICO**

A continuación, se detalla paso a paso la resolución el ejercicio práctico:

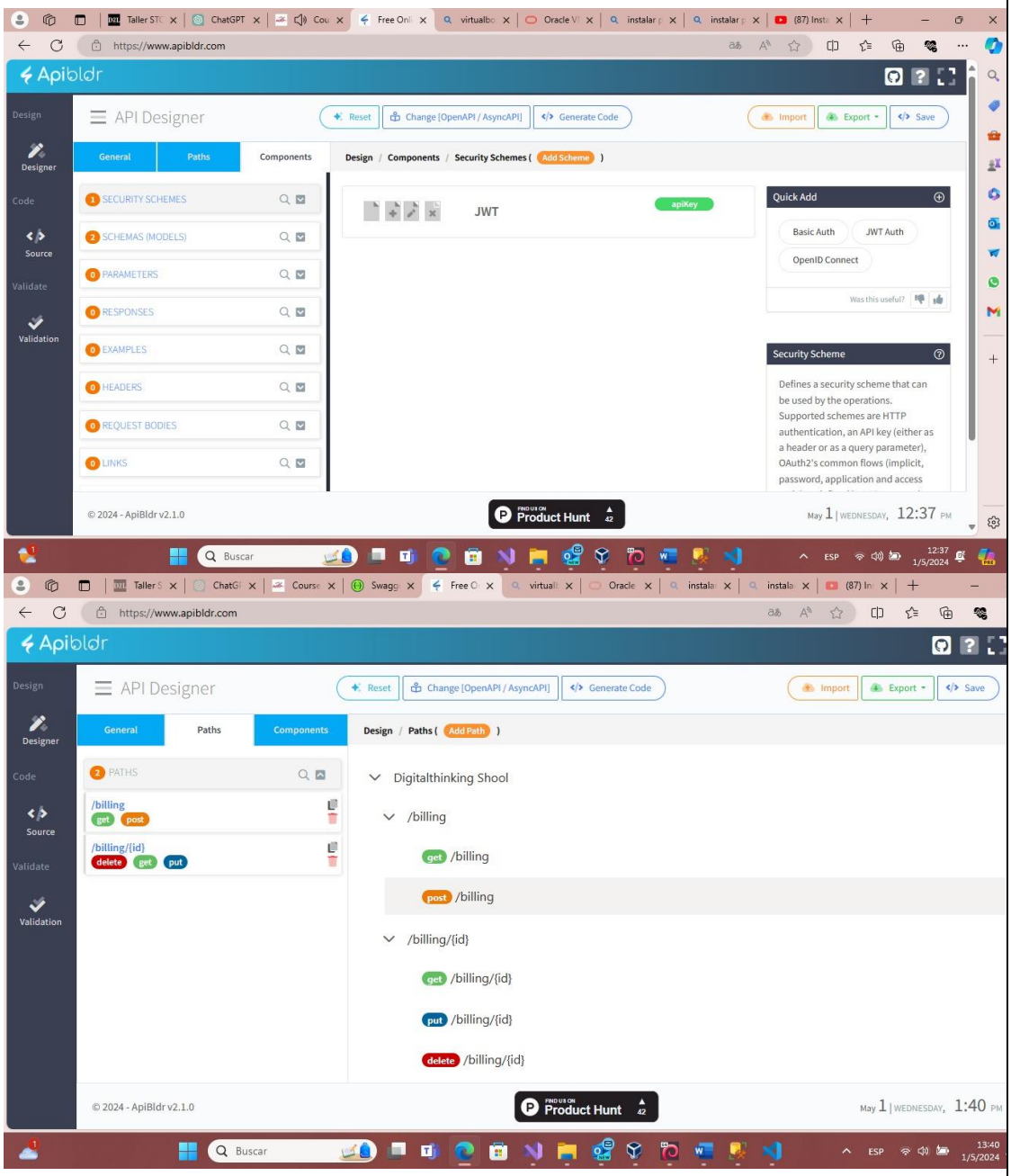
Imagén #1: Archivo YAML

Apertura de archivo .YAML

Imagén #2: uso Apibildr



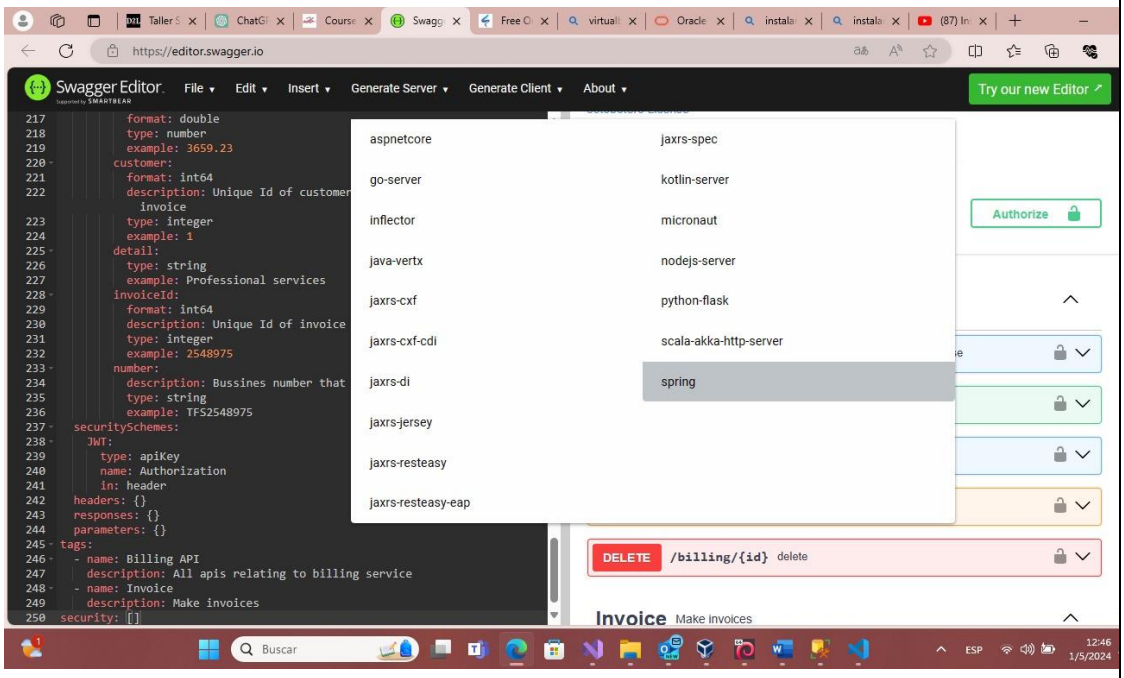
En base al archivo .YAML se construyo el diseño de la API usando la aplicación Apibildr.

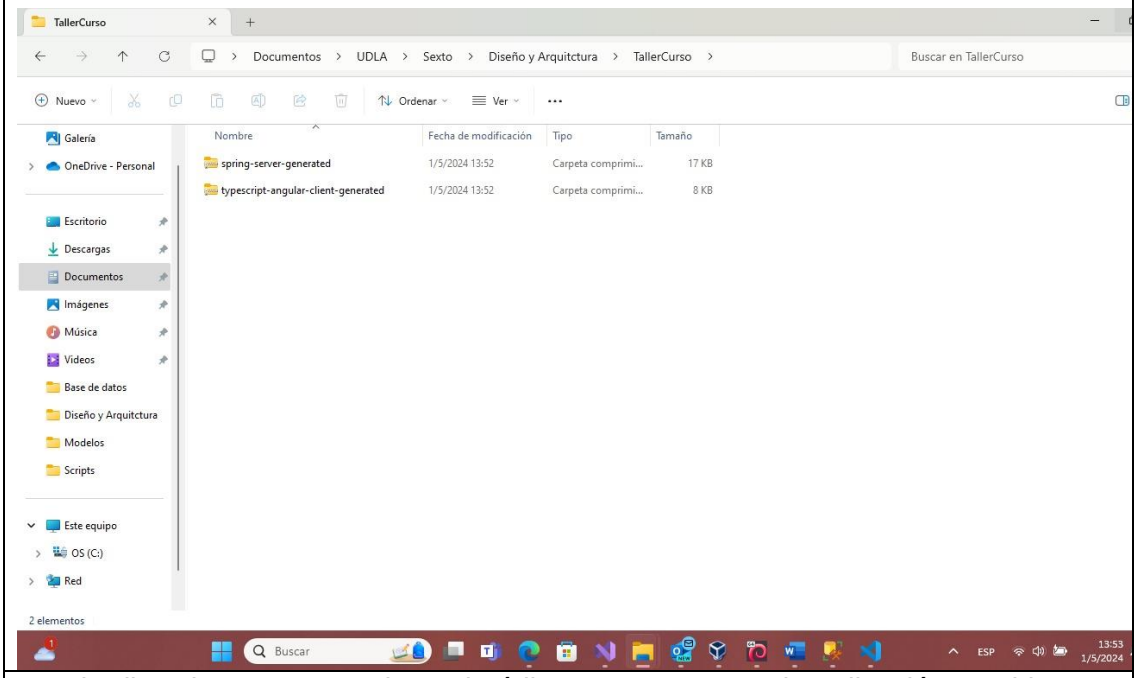
Imagén #3: uso Apibildr



En base al archivo. YAML se construyó el diseño de la API usando la aplicación Apibildr.

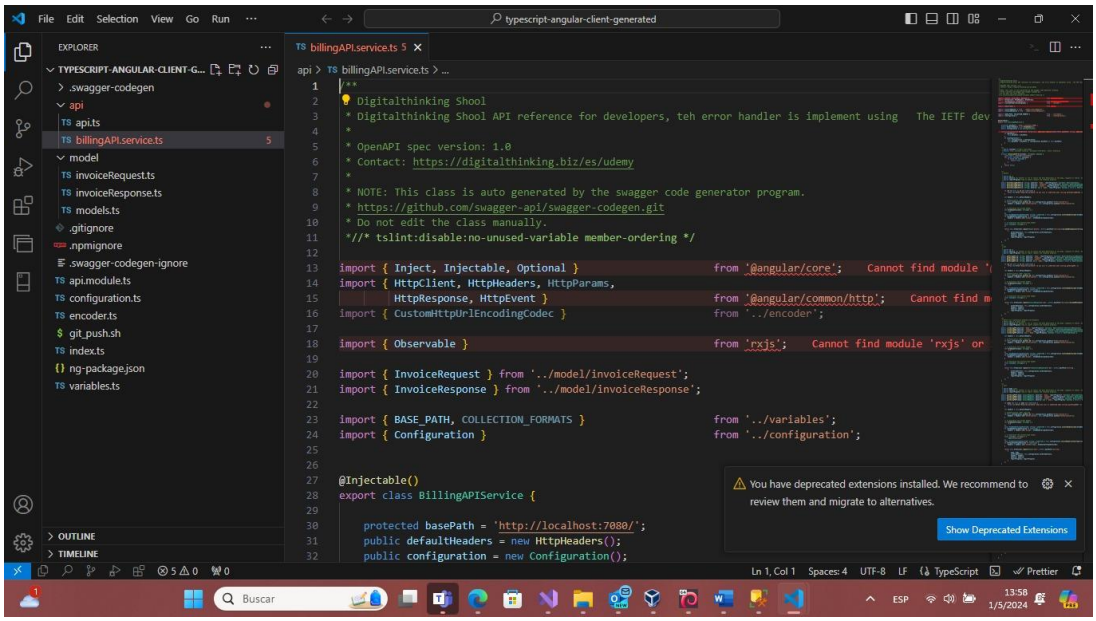
Imagén #4: SWAGER

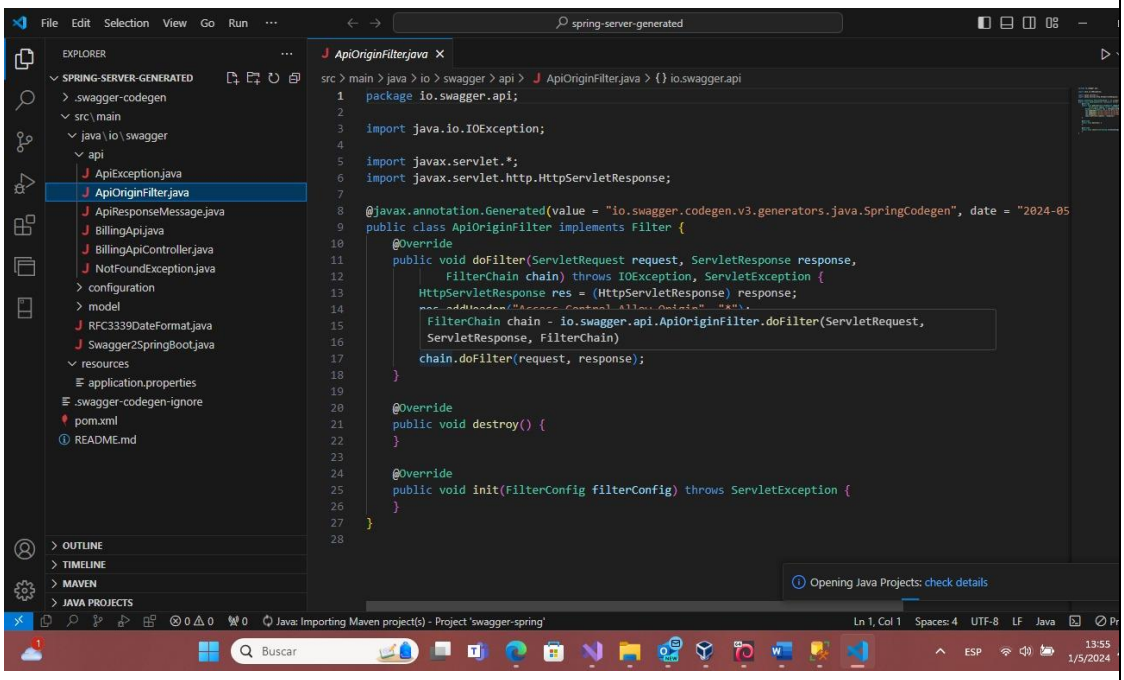




En el editor de sawer se coloca el código y se construye la aplicación servidor con spring y la aplicación cliente.

Imagén #5: Exploración del código





Finalmente se abrió cada proyecto navegado y se exploro la solución y como estaba formado.

7. OPINIÓN PERSONAL

- Este curso me pareció muy interesante porque pude aprender acerca de herramientas de generación de código automático que no sabía que existían y se podían usar en este contexto. Estas herramientas aceleran el proceso de desarrollo y reducen la carga de trabajo al automatizar tareas repetitivas. Una de las herramientas fue swager, mi experiencia trabajando con esta herramienta fue atreves de .net core y me pareció muy interesante ver como se puede interactuar con otros famework y lenguajes desde sawer para generar el código base. Una de las grandes ventajas que encontré respecto a esto es que al automatizar el proceso degeneración de código se puede reducir la probabilidad de introducir errores de tipo Humanos. Además, que fomenta la coherencia ya que se construye el código en base a patrones y estándares.
- Gracias al curso, comprendí de mejor manera el amplio alcance de las API y como se estructuran dependiendo de su función. También puede comprender la razón teórica detrás de los métodos HTTP y la construcción de las rutas, entendiendo que se puede definir rutas y métodos que permitan a los clientes acceder y manipular los datos de manera eficiente y segura. Es importante saber que al partir de la documentación de la API, esto permite que diferentes desarrolladore puedan hacer uso de esta y aprovechar sus funcionalidades de manera exitosa.
- Una característica importante que aprendí fue el concepto de la integración entre el fronent y el backend y la importancia de una comunicación afectiva. Entendí que un grupo de trabajo debe definir comó será la comunicación etre estos para aumentar el rendimiento del equipo. En el caso del ejemplo, aprendí cómo integrar el backend (en mi caso, un microservicio Springwood) con el frontend (una aplicación Angular) para crear una aplicación web completa. Esto incluye la comunicación entre los dos mediante solicitudes HTTP y la manipulación de datos en ambos extremos.
- Al realizar la tarea práctica, tuve la oportunidad de profundizar en la aplicación de los conceptos aprendidos durante el curso. Pude poner en práctica el diseño de APIs en la herramienta Apibldr y entender el manejo de solicitudes HTTP. Además, comprendí la arquitectura de una aplicación que va a manejar una API como punto transaccional.

8. ANEXOS

Anexo #1: Link Github Proyecto Servidor
CamilaACT/CursoUdemyServidor (github.com)

Anexo #2: Link Github Proyecto Cliente
CamilaACT/CursoUdemyCliente (github.com)

9. BIBLIOGRAFÍA

Curso Udemy:

Course: Fundamentos en DevOps, APIs y Arquitectura de Microservicios | Udemy Business