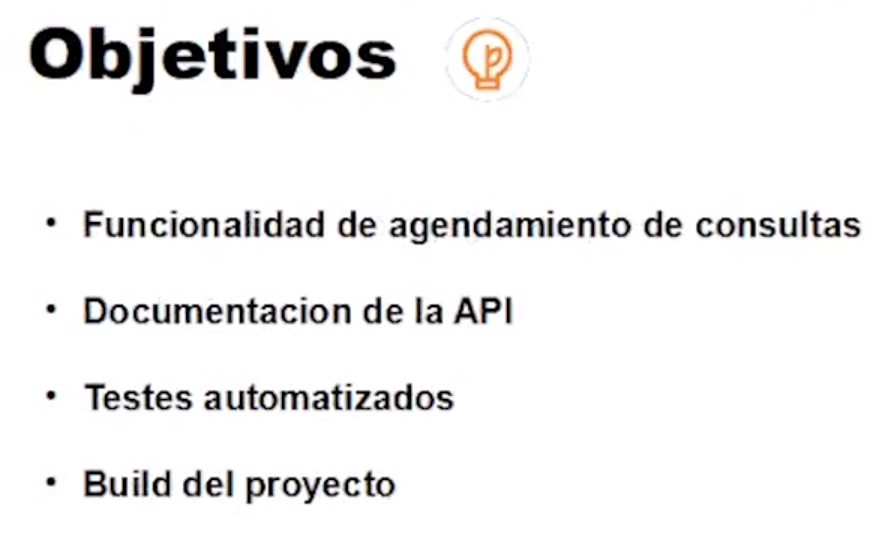
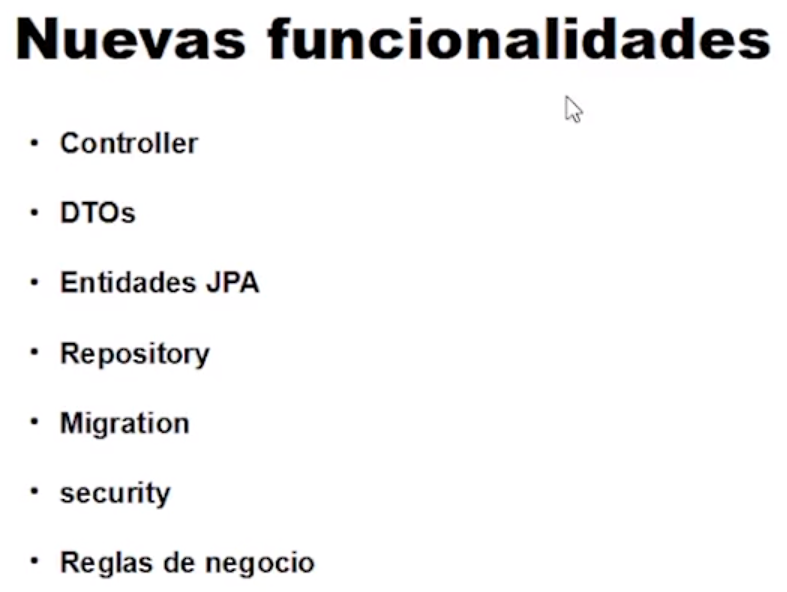
Curso de  
**Spring Boot 3: documentar, probar y preparar una API para su implementación**

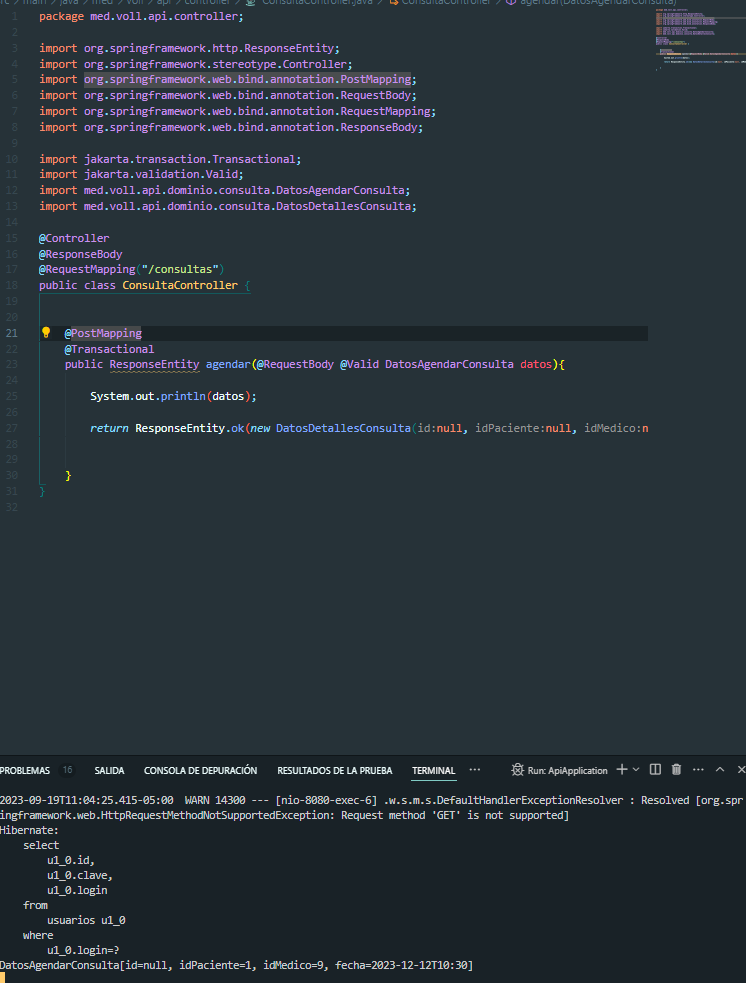
****

**AGENDAMIENTO DE CONSULTAS:**

Nueva funcionalidad:

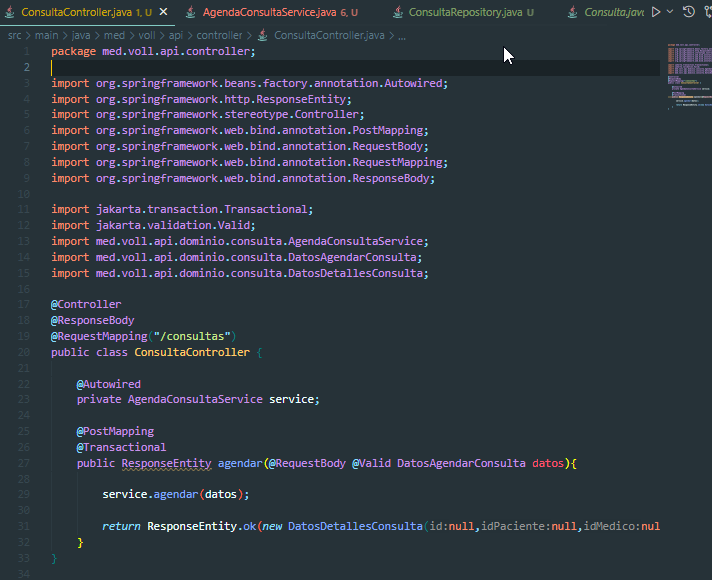


Consulta Controller:

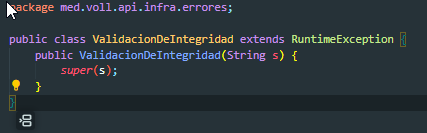


Clase Service:

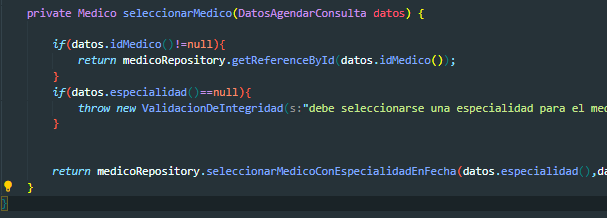
CONTROLLER:

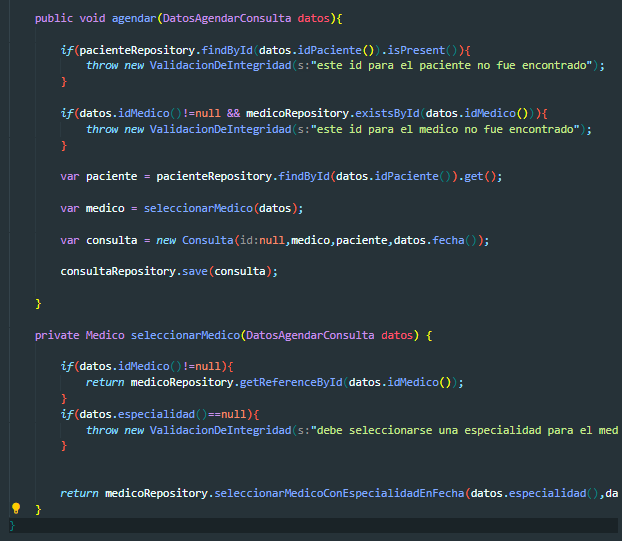


Validaciones de integridad:



Selección medico aleatorio:





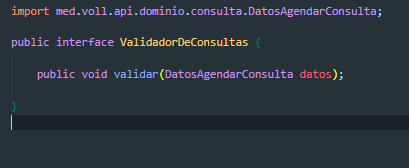
Lo que Aprendimos:

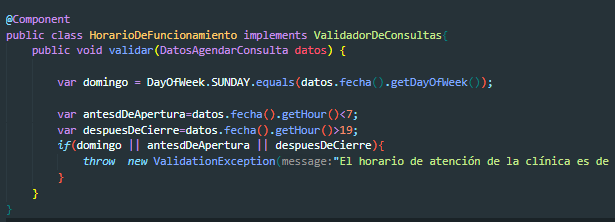
Lo que aprendimos en esta aula:

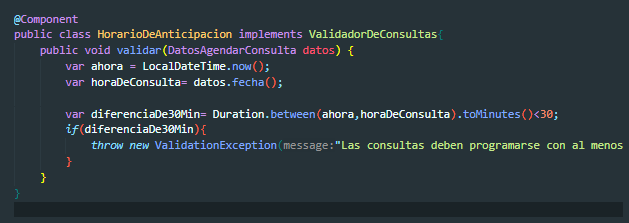
* Implementar una nueva funcionalidad en el proyecto;
* Evaluar cuando es necesario crear una clase de Servicio en la aplicación;
* Crear una clase de Servicio, con el objetivo de aislar códigos de reglas de negocio, utilizando para ello la anotación @Service;
* Implementar un algoritmo para la funcionalidad de programación de consultas;
* Realizar validaciones de integridad de la información que llega a la API;
* Implemente una consulta JPQL (Lenguaje de consulta de persistencia de Java) compleja en una interfaz de repositorio, utilizando la anotación @Query.

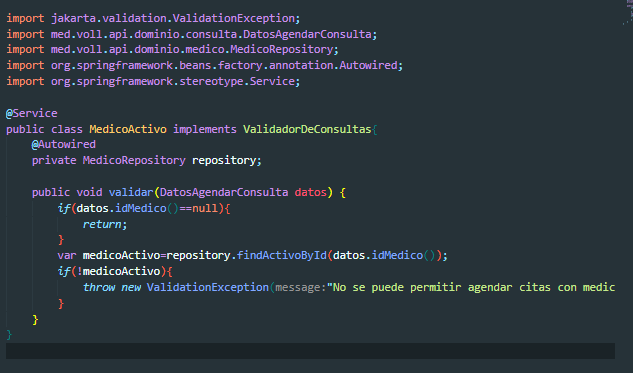
**REGLAS DEL NEGOCIO:**

Clases de validación:

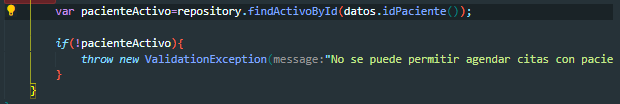


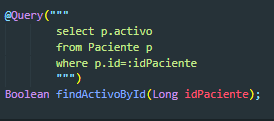


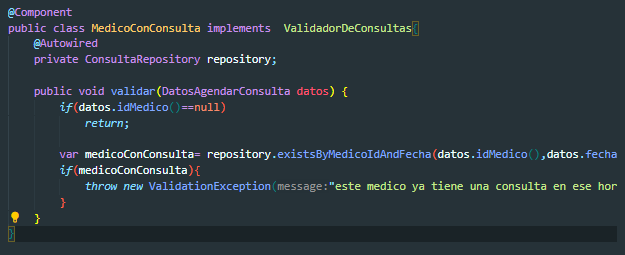


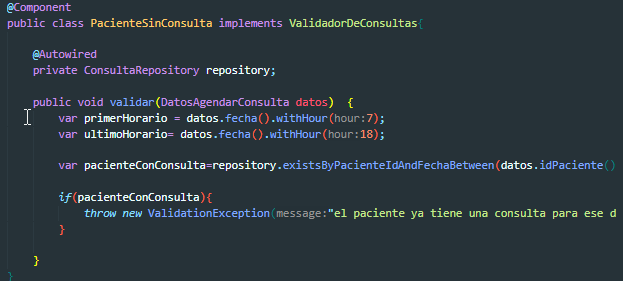


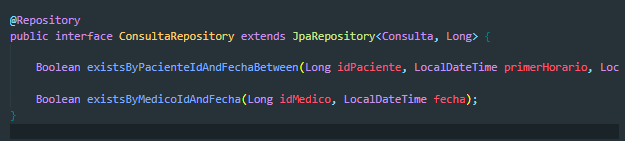
Validación:



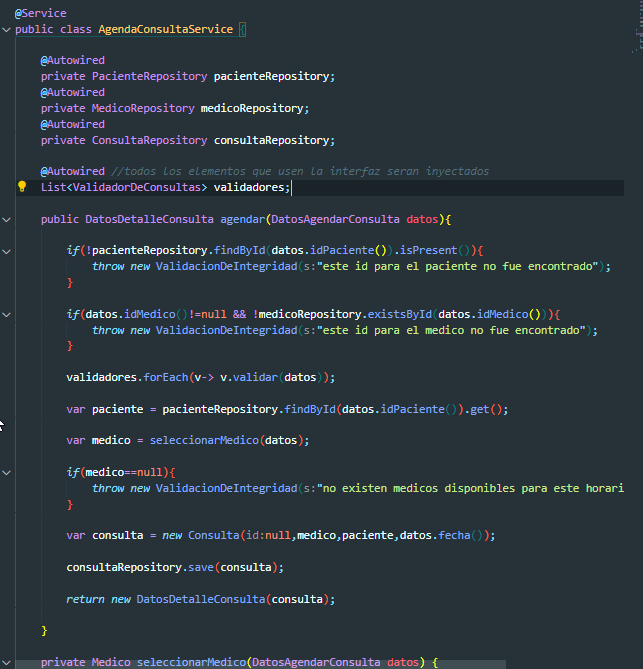


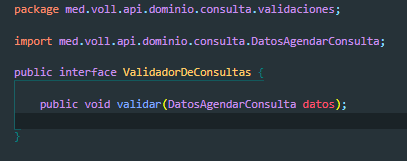






Aplicando principios SOLID:



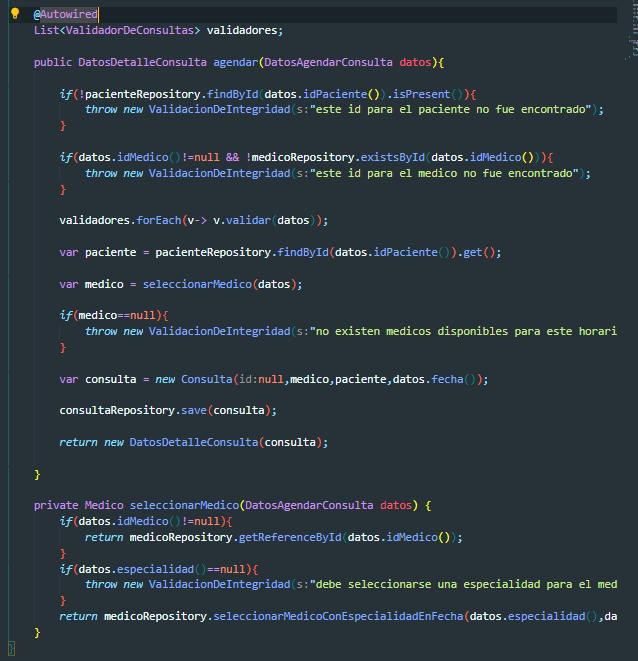


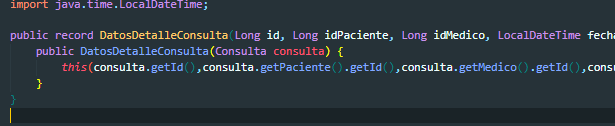
SOLID es un acrónimo que representa cinco principios de programación:

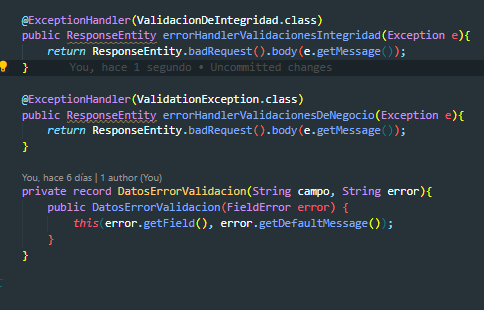
* Principio de Responsabilidad Única (Single Responsibility Principle)
* Principio Abierto-Cerrado (Open-Closed Principle)
* Principio de Sustitución de Liskov (Liskov Substitution Principle)
* Principio de Segregación de Interfaces (Interface Segregation Principle)
* Principio de Inversión de Dependencia (Dependency Inversion Principle)

Cada principio representa una buena práctica de programación que, cuando se aplica en una aplicación, facilita mucho su mantenimiento y extensión. Estos principios fueron creados por Robert Martin, conocido como Uncle Bob, en su artículo Design Principles and Design Patterns.

Testeando la agenda:







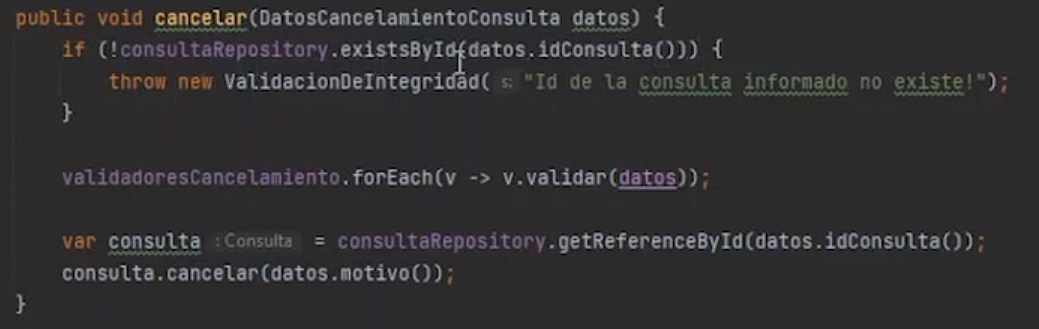
Lo que Aprendimos:

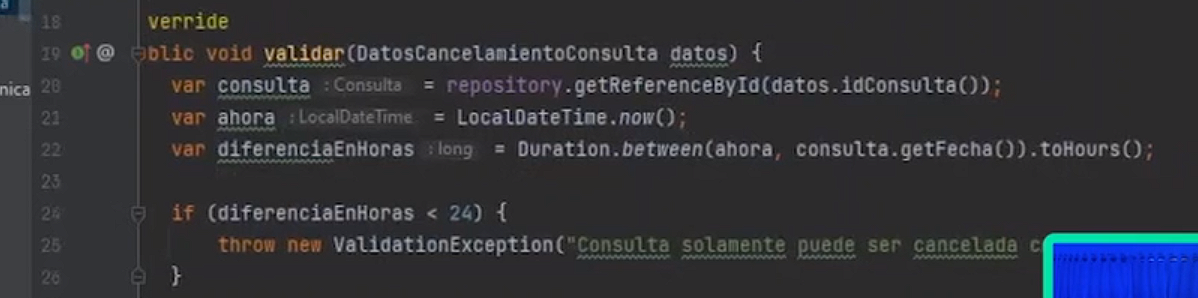
Lo que aprendimos en esta aula:

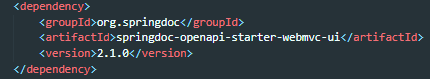
* Aislar los códigos de validación de reglas comerciales en clases separadas, utilizando la anotación @Component de Spring;
* Finalizar la implementación del algoritmo de programación de consultas;
* Usar los principios SOLID para hacer que el código de la funcionalidad de programación de citas sea más fácil de entender, evolucionar y probar.

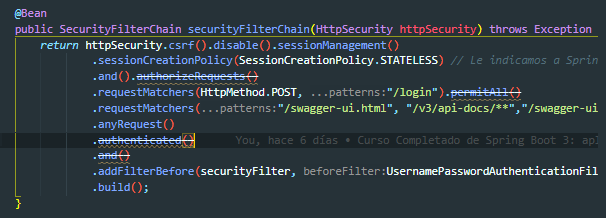
**DOCUMENTACION DE LA API:**

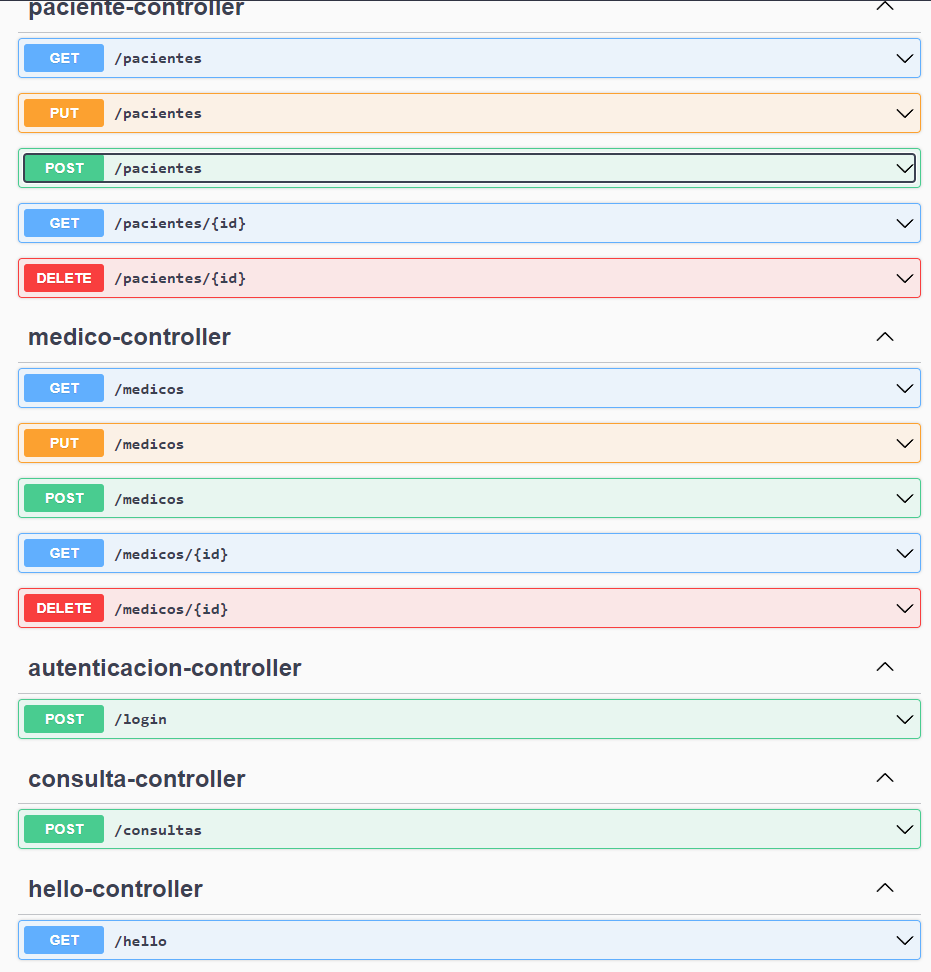
Documentando con SpringDoc:











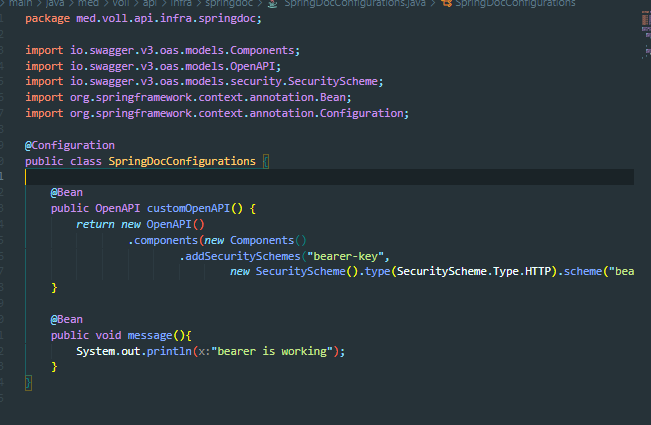
La documentación es algo muy importante en un proyecto, especialmente si se trata de una API Rest, ya que en este caso podemos tener varios clientes que necesiten comunicarse con ella y necesiten documentación que les enseñe cómo realizar esta comunicación de manera correcta.

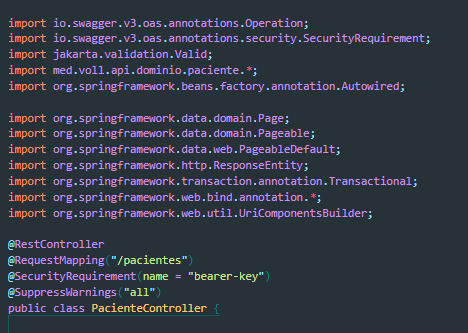
Durante mucho tiempo no existió un formato estándar para documentar una API Rest, hasta que en 2010 surgió un proyecto conocido como Swagger, cuyo objetivo era ser una especificación open source para el diseño de APIs Rest. Después de un tiempo, se desarrollaron algunas herramientas para ayudar a los desarrolladores a implementar, visualizar y probar sus APIs, como Swagger UI, Swagger Editor y Swagger Codegen, lo que lo convirtió en un proyecto muy popular y utilizado en todo el mundo.

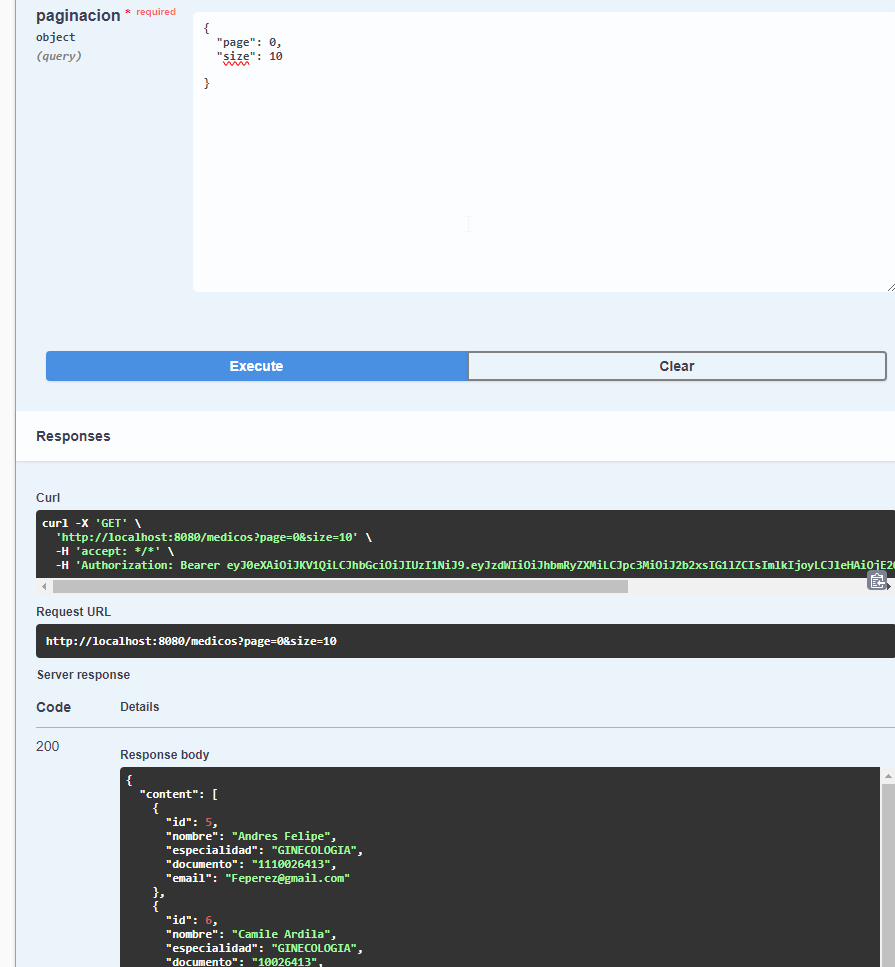
En 2015, Swagger fue comprado por la empresa SmartBear Software, que donó la parte de la especificación a la fundación Linux. A su vez, la fundación renombró el proyecto a OpenAPI. Después de esto, se creó la OpenAPI Initiative, una organización centrada en el desarrollo y la evolución de la especificación OpenAPI de manera abierta y transparente.

OpenAPI es actualmente la especificación más utilizada y también la principal para documentar una API Rest. La documentación sigue un patrón que puede ser descrito en formato YAML o JSON, lo que facilita la creación de herramientas que puedan leer dichos archivos y automatizar la creación de documentación, así como la generación de código para el consumo de una API.

JWT en a la documentación:







En el video anterior, vimos que es posible personalizar la documentación generada por SpringDoc para incluir el token de autenticación. Además del token, podemos incluir otras informaciones en la documentación que forman parte de la especificación OpenAPI, como la descripción de la API, información de contacto y su licencia de uso.

Estas configuraciones se deben hacer en el objeto OpenAPI, que se configuró en la clase SpringDocConfigurations de nuestro proyecto:

@Bean

public OpenAPI customOpenAPI() {

**return** **new** **OpenAPI**()

.components(new Components()

.addSecuritySchemes("bearer-key",

new SecurityScheme()

.type(SecurityScheme.Type.HTTP)

.scheme("bearer")

.bearerFormat("JWT")))

.info(new Info()

.title("API Voll.med")

.description("API Rest de la aplicación Voll.med, que contiene las funcionalidades de CRUD de médicos y pacientes, así como programación y cancelación de consultas.")

.contact(new Contact()

.name("Equipo Backend")

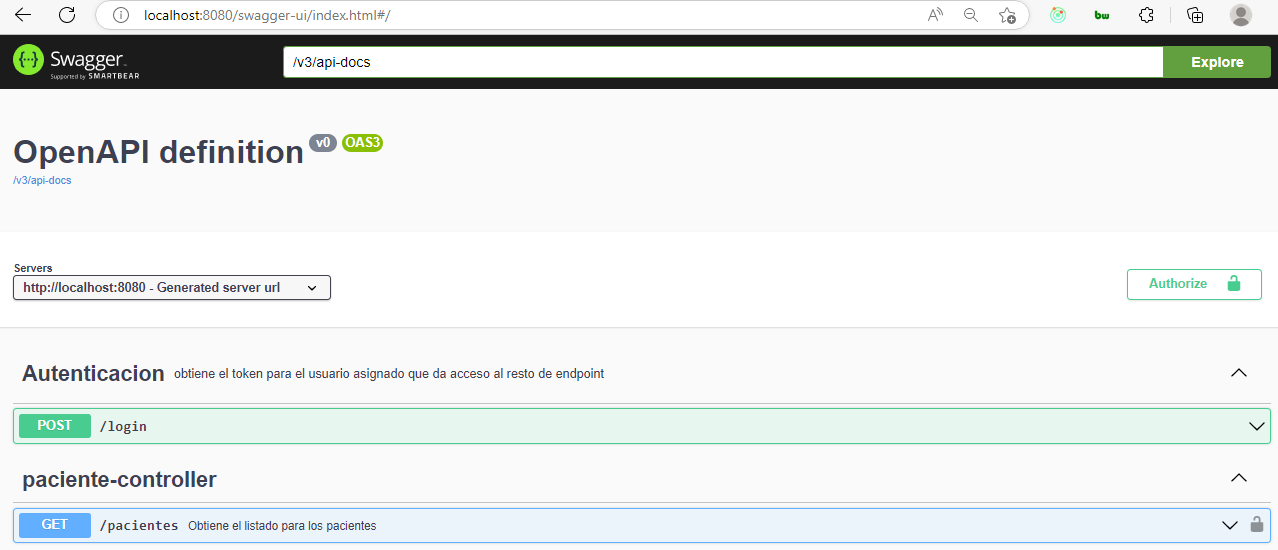
.email("backend@voll.med"))

.license(new License()

.name("Apache 2.0")

.url("http://voll.med/api/licencia"))); }COPIA EL CÓDIGO

En el código anterior, después de la configuración del token JWT, se agregaron las informaciones de la API. Al acceder nuevamente a la página de Swagger UI, se mostrarán estas informaciones, como se muestra en la imagen a continuación:



Para obtener más detalles sobre qué informaciones se pueden configurar en la documentación de la API, consulte la especificación OpenAPI en el sitio web oficial de la iniciativa.

Lo que Aprendimos:

Lo que aprendimos en esta aula:

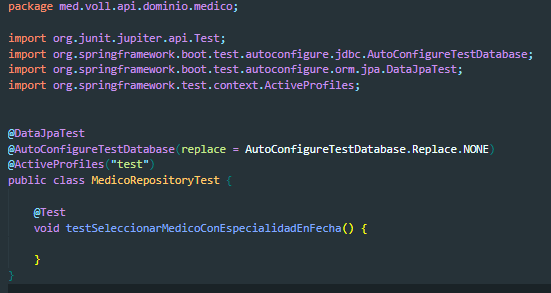
* Agregar la biblioteca SpringDoc al proyecto para que genere automáticamente la documentación de la API;
* Analizar la documentación de SpringDoc para comprender cómo configurarlo en un proyecto;
* Acceder a las direcciones que brindan la documentación de la API en formatos yaml y html;
* Utilizar la interfaz de usuario de Swagger para visualizar y probar una API Rest;
* Configurar JWT en la documentación generada por SpringDoc.

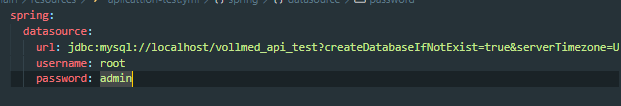
**TEST AUTOMATIZDOS:**

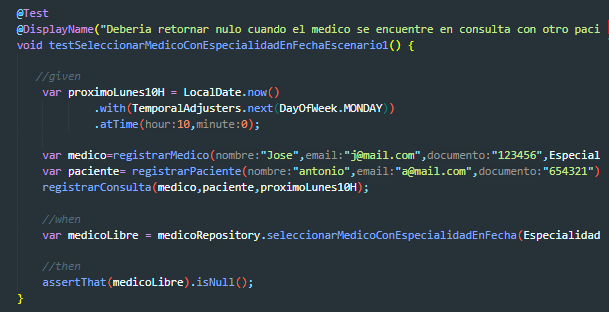
Testes con Spring Boot:



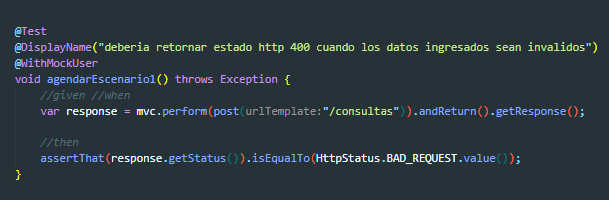
Configurando el banco de datos de Test:



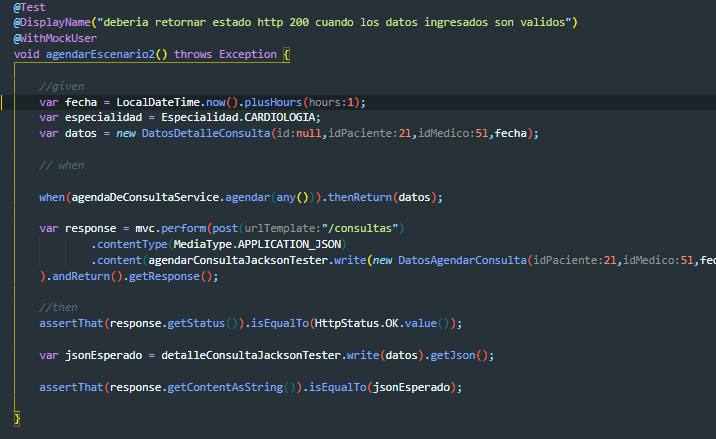


Testeando el Repository:

Testeando el Error 400:



Testeando el Error 200:



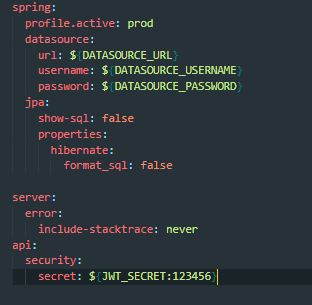
Lo que Aprendimos:

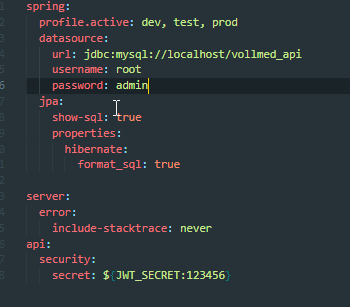
Lo que aprendimos en esta aula:

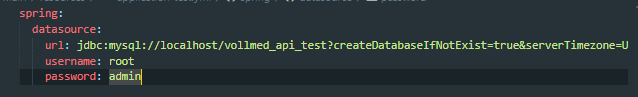
* Escribir pruebas automatizadas en una aplicación con Spring Boot;
* Escribir pruebas automatizadas para una interfaz de Repositorio, siguiendo la estrategia de utilizar la misma base de datos que utiliza la aplicación;
* Sobreescribir las propiedades del archivo application.properties, creando otro archivo llamado application-test.properties que se carga solo cuando se ejecutan pruebas, usando la anotación @ActiveProfiles para eso;
* Escribir pruebas automatizadas para una clase de controlador, utilizando la clase MockMvc para simular solicitudes de API;
* Probar los escenarios de error 400 y código 200 al probar una clase de controlador.

**BUILD DEL PROYECTO:**

Build con Maven:







Ejecutando vía terminal:

Lo que Aprendimos:

Lo que aprendimos en esta aula:

* Realizar el build de una aplicación con Spring Boot;
* Utilizar archivos de propiedades específicos para cada perfil, cambiando las propiedades que deben modificarse en cada archivo;
* Configurar información confidencial de la aplicación, como datos de acceso a la base de datos, a través de variables de entorno;
* Crear el proyecto a través de Maven;
* Ejecutar la aplicación a través de la terminal, con el comando java -jar, pasando las variables de entorno como parámetros.