Universidad® de Medellín
Ciencia y Libertad

Chain Of Responsability

Diego Alejandro Durango Garro

Diseño de Software Flexible y Reusable
Especialización Ing. Software
Universidad de Medellín
2025-1

Universidad de Medellín
Ciencia y Libertad

Contenido

Análisis del problema 3
Diagrama de Clases UML 5
Diagrama Completo: 7
Identificación Patrones 7

Análisis del problema

Para este caso de estudio vamos a analizar párrafo por párrafo.

“ Supongan que están trabajando en un cliente que se conecta a un sistema para hacer órdenes en línea. Ustedes quieren restringir acceso al sistema para que solo los usuarios autenticados puedan crear órdenes. Además, los usuarios que tienen permisos administrativos deben tener acceso total al sistema. “

Para esto, ya que sabemos que es un sistema que se hace de órdenes en línea, vamos a crear una API para dichas ordenes o peticiones al sistema, para lo cual se necesita un modelo base para las peticiones (RequestModel). Que contendrá las credenciales de los usuarios Adicional, sabemos que tenemos que manejar usuarios autenticados o con tipos de permisos, para esto vamos a manejar un Handler para que valide esas credenciales (AuthHandler).

“Después de un rato de planeación, se dan cuenta de que estas verificaciones se deben realizar de forma secuencial. La aplicación intenta autenticar un usuario cuando recibe una solicitud que contiene las credenciales del usuario, sin embargo, si dichas credenciales no son correctas y la autenticación falla, no hay razón para proceder con otras verificaciones.”

Ya que nos indican que las verificaciones se deben de hacer de forma secuencial, vamos a usar un patrón de comportamiento llamado change of responsibility (cadena de responsabilidad). El cual nos va a permitir darle un orden a las verificaciones y hacer que la solicitud. Pase por cada una de la verificación. Para esto ya nuestra clase de petición y nuestra clase de autenticación. Estarían acorde para ese patrón, se usa la guía para este patrón.

(Chain of Responsibility)

“Durante las siguientes semanas, implementan algunas verificaciones adicionales:

Alguien les sugiere que no es seguro pasar datos crudos directo al sistema de solicitudes. Así que usted adiciona una validación extra para sanear los datos en la solicitud.”

Dado que ya decidimos o accionamos usar el patrón de cadena de responsabilidad. Simplemente agregar un manejador el cual haga el saneamiento de los datos (DataValidatorHandler). Después del procesamiento de la autenticación. Y esta clase no interfieren con el funcionamiento y autenticación, Ya que su funcionalidad única es para sanear los datos, También modificamos nuestro modelo (RequestModel), para que obtenga los datos que se desean sanear, agregando un nuevo campo que sea de data.

“Después, alguien nota que el sistema es vulnerable a ataques de ‘fuerza bruta’. Para evitar esto, usted adiciona una verificación que filtra solicitudes fallidas repetidas que vienen de la misma dirección IP.”

Agregaremos un tercer manejador (BruteForceHandler), el cual se encargue de validar. Los ataques de fuerza bruta dentro del sistema. Ya que indican que los ataques de fuerza bruta vienen de la misma dirección, IP, modificamos el modelo (RequestModel), para que agregue un nuevo que tenga la IP del solicitante.

"Alguien más sugiere que se podría incrementar la velocidad de respuesta del sistema si se retornan resultados 'cacheados' para solicitudes repetidas que contienen los mismos datos. Por lo tanto, ustedes adicionan otra verificación que deja pasar la solicitud por el sistema solo si no hay una respuesta adecuada cacheada."

Agregaremos otro manejador (CacheHandler). Que sería para los sería para los datos cacheados, el cual se encargaría de validar las respuestas anteriores a la misma solicitud si la respuesta ya se encuentra. Respondería del caché sin tener la procesar en caso de que no se encuentre, pues la procesaría.

"Ustedes deben diseñar el cliente siguiendo los pasos descritos en el enunciado y considerando que es muy probable que a futuro les pidan adicionar nuevas verificaciones o que dichas verificaciones se puedan reutilizar en otros clientes que requieran usar el sistema de órdenes en línea."

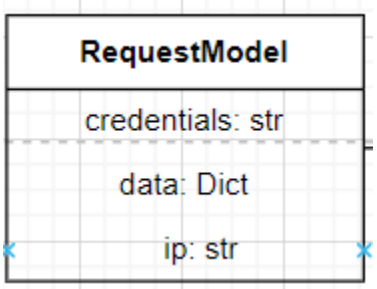
Dado que ya tenemos cuatro manejadores diferentes. Vamos a utilizar un manejador base (BaseHandler), el cual va a obtener un método abstracto, el cual se guiará el comportamiento de los otros manejadores y así tener un control. En caso tal se quieran agregar otras variaciones u otros manejadores, se use bajo el mismo modelo y no se afecte el funcionamiento de la aplicación. esto bajo el patrón creacional de Template Method.

Ya que obtuvimos toda la información del ejercicio y decidimos utilizar la cadena de responsabilidades. Debe crear una clase en la cual se asignen el orden de las validaciones o los verificadores que se van a usar. (OrderClientService)

Cómo se decidió utilizar una API para responder a las solicitudes, se va a hacer la implementación de forma fácil y sencilla. Explicativa en el lenguaje de programación Python con la librería de FASTAPI.

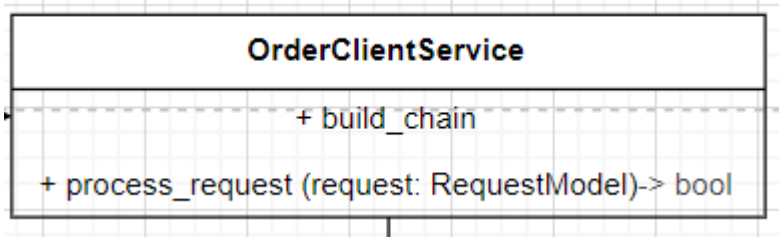
a. RequestModel:

Descrito los datos que debe llevar en el análisis del problema



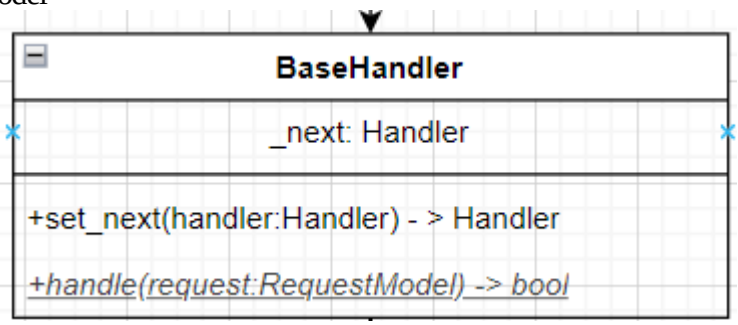
b. OrderClientService:

Se encarga de tener ele método build_chain que modera el orden de las verificaciones y procces_request que procesa la petición entrante recibiendo un RequestModel



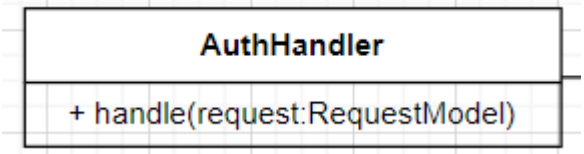
c. BaseHandler:

Contiene una variable Tipo Handler, método set_net que define el Handler siguiente y el método handle que es abstracto y se usa en los siguientes manejadores para proesar el RequestModel



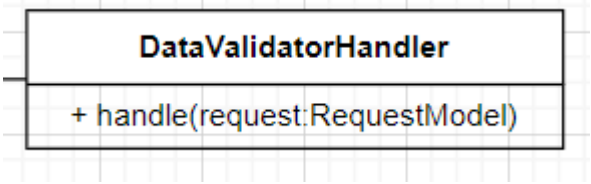
d. AuthHandler:

Contiene el método abstracto heredado, pero con la lógica propia de la autenticación



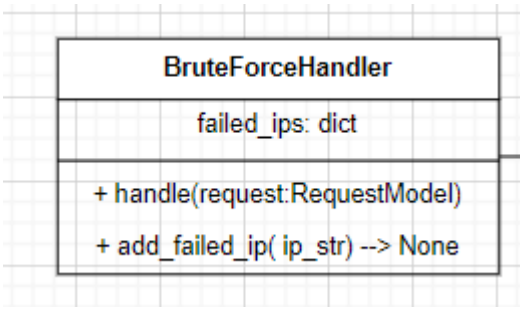
e. DataValidatorHandler:

Contiene el método abstracto heredado, pero con la lógica propia para sanear los datos



f. BruteForceHandler:

Contiene el método abstracto heredado, pero con la lógica propia de detectar un ataque de fuerza bruta



g. CacheHandler:

Contiene el método abstracto heredado, pero con la lógica propia para consultar la cache y retornar más rápidamente una respuesta

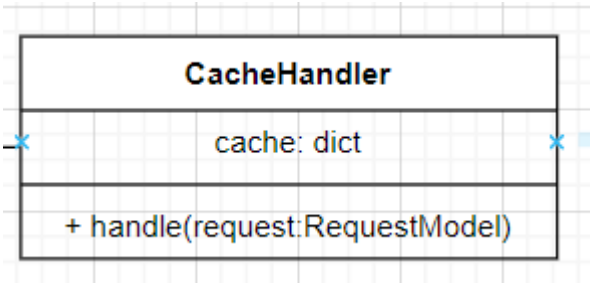
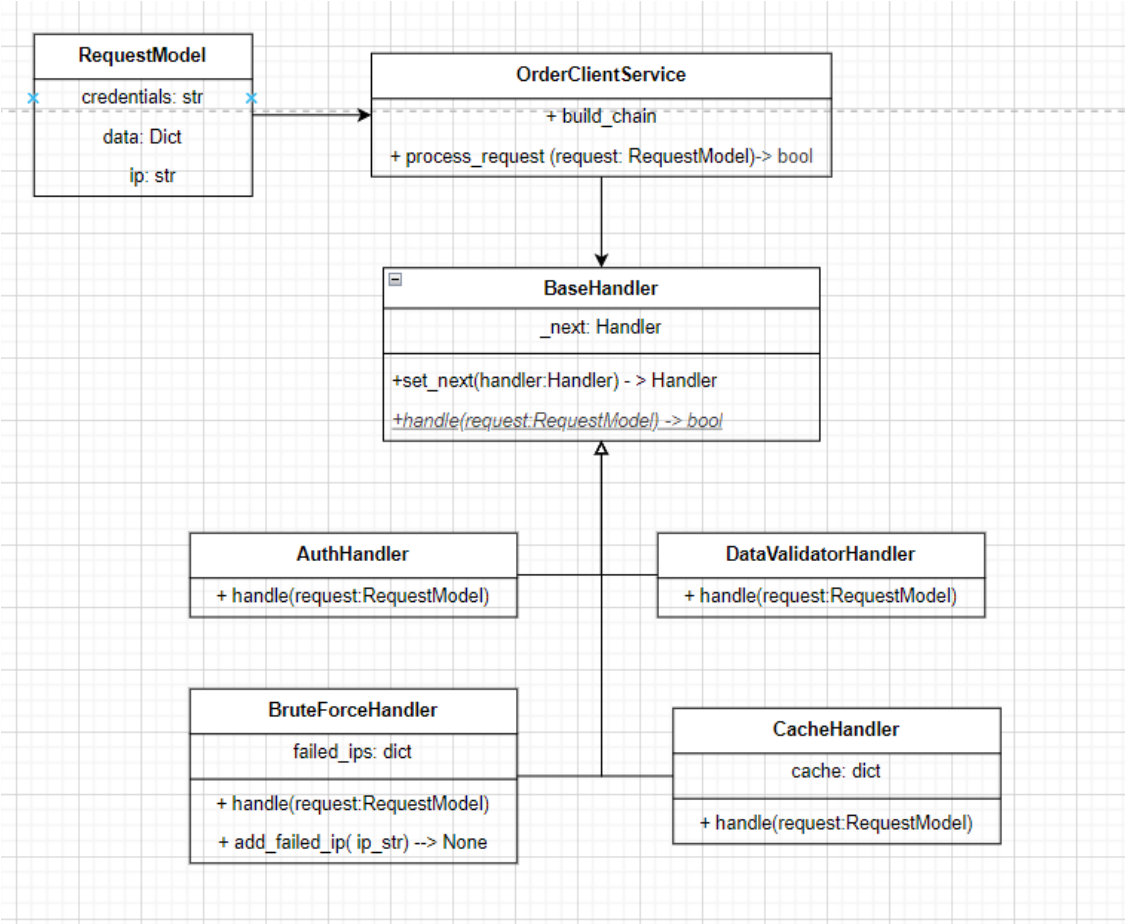


Diagrama Completo:



Identificación Patrones

Los patrones identificados dentro del ejercicios son Chain of Responsability como principal, elementos de Singleton, Template Method, Strategy:

a. **Chain of Responsability:** permite que la solicitud pase a través de la cadena de manejadores decide si se procesa la solicitud o si pasa al siguiente se ve en el código en la clase BaseHandler que es la base de los manejadores. Y OrderClientService que construye ese orden de los manejadores.
b. **Singleton**: no se ve de forma explícita, pero al implementar failed_ips en BruteForceHandler, se ve que los datos son compartidos en otro manejador.
c. **Template Method:** se denota que en BaseHandler se tiene un método base handle que las subclases sobrescriben para implementar su lógica especifica
d. **Strategy:** Cada Manejador implementa una estrategia especifica para manejar solicitudes , teniendo su propia lógica en el método heredado handle lo qe camia o extiende el comportamiento.