Documento Arquitetural

Arquitetura de Microsserviços	
Arquiteto	**Arquiteto**
Maycon da Silva Moreira	José Francisco de Oliveira Junior
maycondasilvamoreira@hotmail.com	joseph.francisco.oliveira@gmail.com
(62) 9 9545-2562	62 9 8204-0541

Objetivo deste Documento

Este documento tem como objetivo descrever as principais decisões de projeto tomadas pela equipe de desenvolvimento e os critérios considerados durante a tomada destas decisões. Além de reunir informações necessárias ao controle das atividades de arquitetura, oferecendo uma visão macro dos requisitos arquiteturais e não funcionais.

Histórico de Revisão

Data	Demanda	Autor	Descrição	Versão
04/03/2019	000	Maycon da Silva Moreira	Criação do documento arquitetural	001

Sumário

1. INTRODUÇÃO

1.1 Finalidade

Este documento fornece uma visão arquitetural da arquitetura de microsserviços, usando diversas visões de arquitetura para *representar* diferentes aspectos do sistema. O objetivo deste documento é capturar e comunicar as decisões arquiteturais significativas que foram tomadas em relação à arquitetura.

O documento irá adotar uma estrutura baseada na visão "4+1" de modelo de arquitetura [KRU41].

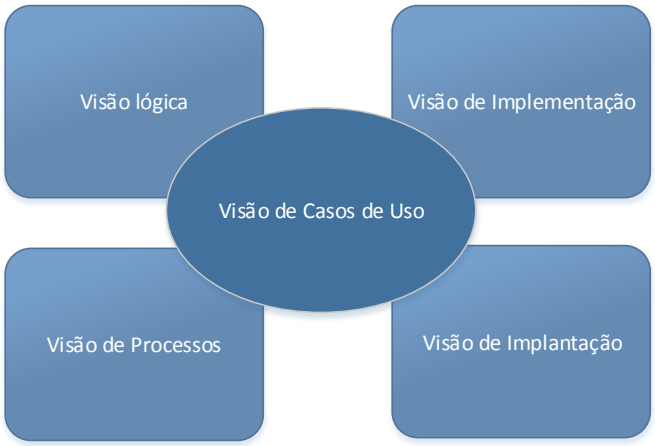


Figura 1 – Arquitetura 4+1

1.2 Escopo

Este documento de arquitetura de software descreve os padrões de software, seus componentes, plataformas de desenvolvimento, plataformas de hardware, softwares de desenvolvimento, servidores de aplicação, servidores de banco de dados, sistemas operacionais, linguagem de programação, tecnologias, frameworks e APIs.

Também são descritos neste "Documento Arquitetural de Microsserviços" a descrição dos focos e sistemáticas arquiteturais, descrição das camadas que é composto o modelo arquitetural e requisitos de integrações.

É também escopo deste documento orientar as pessoas que utilizaram esta arquitetura de microsserviços, oferecendo diretrizes quanto às tecnologias a serem utilizadas neste projeto, assim como seu padrão de utilização.

1.3 Fora do escopo

Este documento arquitetural não fará qualquer alusão a novos serviços que venham a ser integrados a essa arquitetura.

1.4 Definições, Acrônimos e Abreviações.

QoS – Quality of Service, ou qualidade de serviço. Termo utilizado para descrever um conjunto de qualidades que descrevem as requisitos não-funcionais de um sistema, como performance, disponibilidade e escalabilidade[QOS].

1.5 Referências

[KRU41]: The "4+1" view model of software architecture, Philippe Kruchten, November 1995, http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/Pbk4p1.pdf

[QOS] https://docs.oracle.com/cd/E19636-01/819-2326/6n4kfe7dj/index.html

GUEDES, Gilleanes T. A. **UML 2: uma abordagem prática**. 2º edição. São Paulo, Novatec Editora Ltda, 2011.

GOMES, Vanessa. **Métricas de qualidade de software.** 2016. Disponível em: <https://www.tiespecialistas.com.br/metricas-de-qualidade-de-software/>. Acessado em 09/05/2019.

MINISTÉRIO DA SAÚDE, DATASUS. **Documento de Arquitetura de Software - Datasus - Ministério da Saúde.** 2016. Disponível em: <http://datasus.saude.gov.br/images/MDSF/MDSoftware/Artefatos/Arquitetura/MDS_DAS_Documento_Arquitetura_Software2.docx>. Acessado em 01/03/2019.

2. REPRESENTAÇÃO ARQUITETURAL

Este documento irá detalhar as visões baseado no modelo "4+1" [KRU41]. As visões utilizadas no documento serão:

- Visão Lógica: Têm como publico alvo os analistas, ela Abrange as classes, interfaces e colaborações que formam o vocabulário do problema e de sua solução;
- Visão de Implementação: Têm como publico alvo os programadores, ela Abrange os componentes e os artefatos utilizados para a montagem e fornecimento da arquitetura de microsserviços;

- Visão do Processo: Têm como publico alvo os integradores, ela Mostra o fluxo de controle entre as várias partes, Desempenho, Escalabilidade, Concorrência;
- Visão de Implantação: Tem como publico alvo a gerencia de configuração, ela Abrange os nós que formam a topologia de hardware em que o sistema é executado.

2.1 Objetivo e restrições arquiteturais

A arquitetura proposta tem como objetivo de ser uma arquitetura referencial para o desenvolvimento de novos sistemas nos cursos de bacharelado em computação. Esta por sua vez utiliza o padrão de microsserviços.

2.2 Critérios da Avaliação Arquitetural

Os critérios utilizados para a seleção da solução arquitetural foram:

- **Resiliência** é a capacidade do software de manter um nível específico de performance em caso de falhas ou de violações em sua interface específica;
- **Interoperabilidade** é a capacidade do software de interagir com um ou mais sistemas específicos;
- **Portabilidade Adaptabilidade** é a capacidade do software de ser adaptado a ambientes diferentes sem a aplicação de ações ou outros meios que não aqueles previamente estabelecidos;
- **Flexibilidade** diz respeito a como a aplicação é capaz de se adequar às mudanças solicitadas na aplicação;

2.3 Tecnologias definidas para o desenvolvimento da arquitetura de microsserviços

Através do estudo bibliográfico foi identificado que é possível desenvolver a arquitetura de microsserviços em diferentes linguagens de programação e utilizando inúmeras tecnologias.

Foram escolhidas às tecnologias que serão citas a seguir por possuírem uma documentação ampla, e por já possuírem soluções que resolvem vários problemas já conhecidos durante o desenvolvimento de uma arquitetura de microsserviços. Essas tecnologias serão citadas a seguir:

- Linguagem de programação: Java 1.8.0_171, vendor: Oracle Corporation;
- Framework: Spring;
- Banco de dados: MySQL 5.6;
- Documentação da API: OpenAPI 3.0;

- Gerenciado de dependência: Maven 3.5.2;

Esta arquitetura de microsserviços pode ser decomposta em quatro principais responsabilidades, utilizando algumas tecnologias para promover QoS para aplicações distribuídas em rede. Abaixo são apresentadas as camadas e suas principais tecnologias:

- Componente de borda, utilizando à tecnologia Zuul;
- Componente de descoberta de serviços, utilizando a tecnologia Eureka;
- Componente de tolerância à falhas, utilizando o Hystrix;
- Camada de autorização e autenticação, utilizando spring security;
- Camada física de dados, utilizando o MySQL;
- Cada serviço possui uma arquitetura em três camadas, sendo:
- Camada de serviço, usando HTTP /JAX-RS;
- Camada de lógica de negócio;
- Camada de acesso a dados, usando JPA/Hibernate

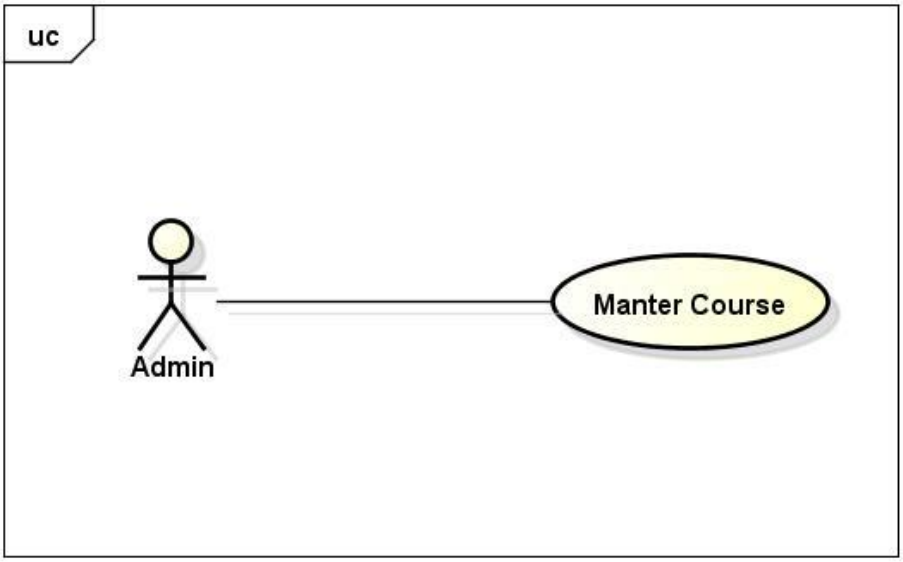
3. VISÃO DE CASOS DE USO

Esta seção mostra os casos de uso dos dois microsserviços desenvolvidos dentro da arquitetura.

3.1 Caso de Uso Course

Ao examinarmos o caso de uso ilustrado na figura 01, percebemos que o ator Admin utiliza de alguma forma, o serviço manter Course, nele é possível cadastrar Course, atualizar Course e excluir Course.

Figura 01 – Diagrama de caso de uso Course

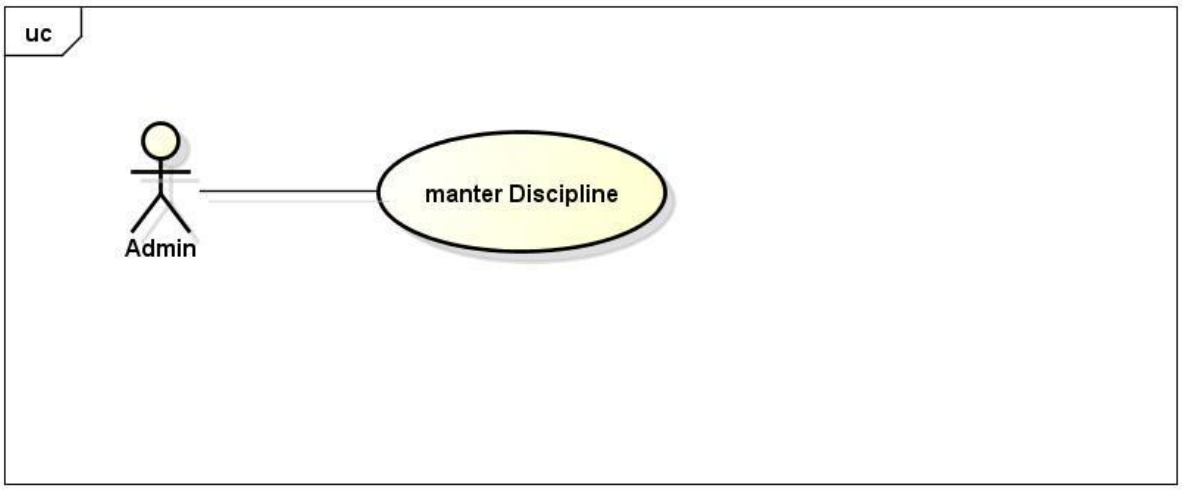


Fonte: Produção Própria dos Autores

3.2 Caso de Uso Discipline

Ao examinarmos o caso de uso ilustrado na figura 02, percebemos que o ator Admin utiliza de alguma forma, o serviço manter Discipline, nele é possível cadastrar Discipline, editar Discipline, excluir Discipline. No entanto para adicionar uma disciplina, obrigatoriamente deverá verificar se o ID do Course é valido.

Figura 02 – Diagrama de caso de uso Discipline



Fonte: Produção Própria dos Autores

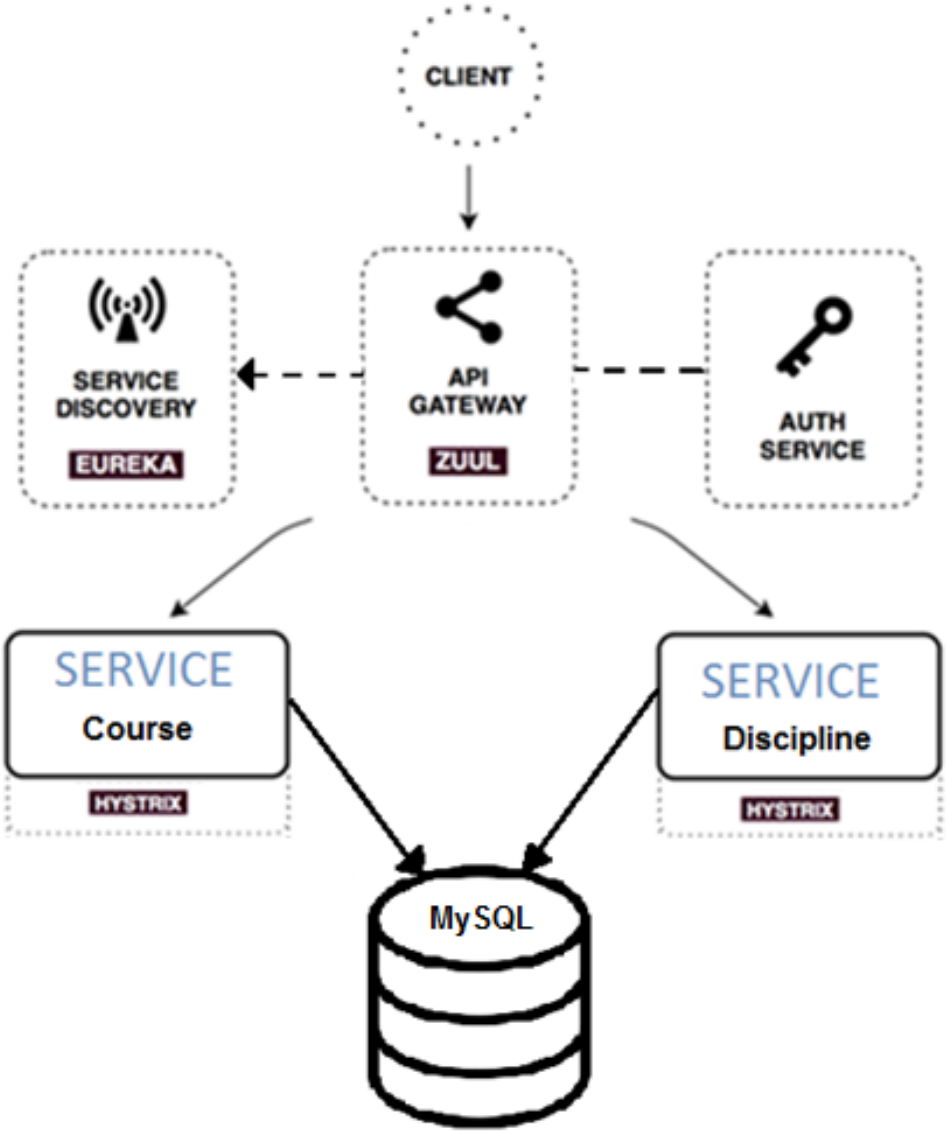
4. VISÃO LÓGICA

Nesse tópico é descrito a visão logica da arquitetura, tais como as classes mais importantes, sua organização em pacotes de serviços e subsistemas e a organização desses subsistemas em camadas.

Também descreve as realizações dos casos de uso dos microsserviços implementados. Diagramas de classes e sequência devem ser incluídos para ilustrar os relacionamentos entre as classes significativas na arquitetura, subsistemas, pacotes e camadas.

Na figura 03 demonstra a estrutura da arquitetura de microsserviços que será utilizada no presente projeto. O fluxo de dados desta arquitetura começa no *Client* após ter feito a autenticação, por exemplo, o mesmo quer acessar o *Service* Course, para isso o *Client* necessita acessar a camada Gateway, que verificará se existe tal serviço, se o serviço existir, o Gateway acessa a camada *Auth Server* e verifica se o *Client* possui permissão para acessar o *Service* Semear, se possuir, então o *Gateway* acessará a camada *Discovery* para pegar o endereço do *Service* Course, depois disso a camada *Gateway* acessará o Service Course e retornando-o para o Client.

Figura 03 - Estrutura da arquitetura de Microsserviços



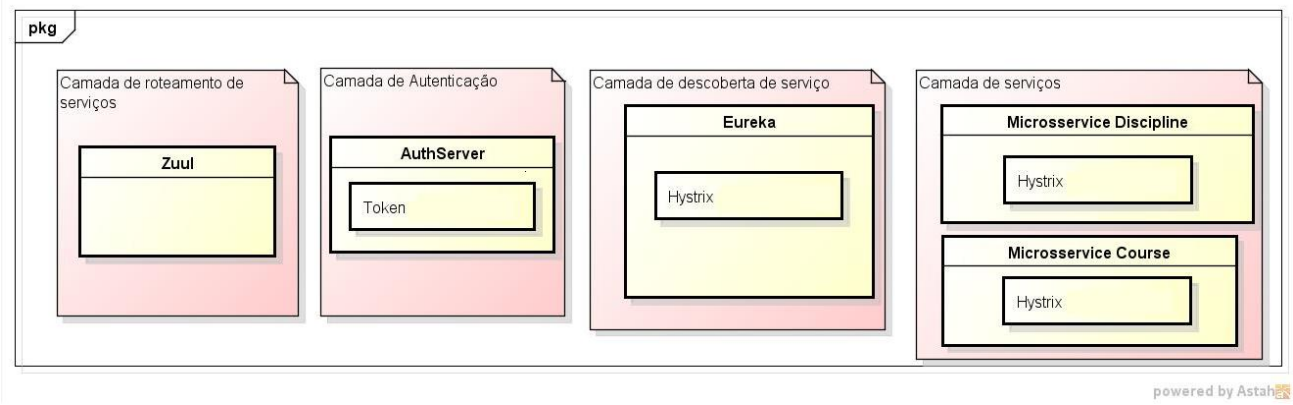
Fonte: Produção Própria dos Autores

4.1 Visão Geral – pacotes e camadas

Na figura 04 demonstra o diagrama de camada, este apresenta as quatro camadas que a arquitetura foi dividida, são elas:

- A camada de entrada, que tem a função de rotear as requisições para os serviços.
- A camada de autenticação, que tem a função de gerenciar toda a parte de autenticação e de segurança da arquitetura.
- A camada de descoberta de serviço, essa tem a função de registrar os serviços dentro da arquitetura.
- E por ultimo a camada de serviços, essa é composta pelos serviços que são integrados a essa arquitetura.

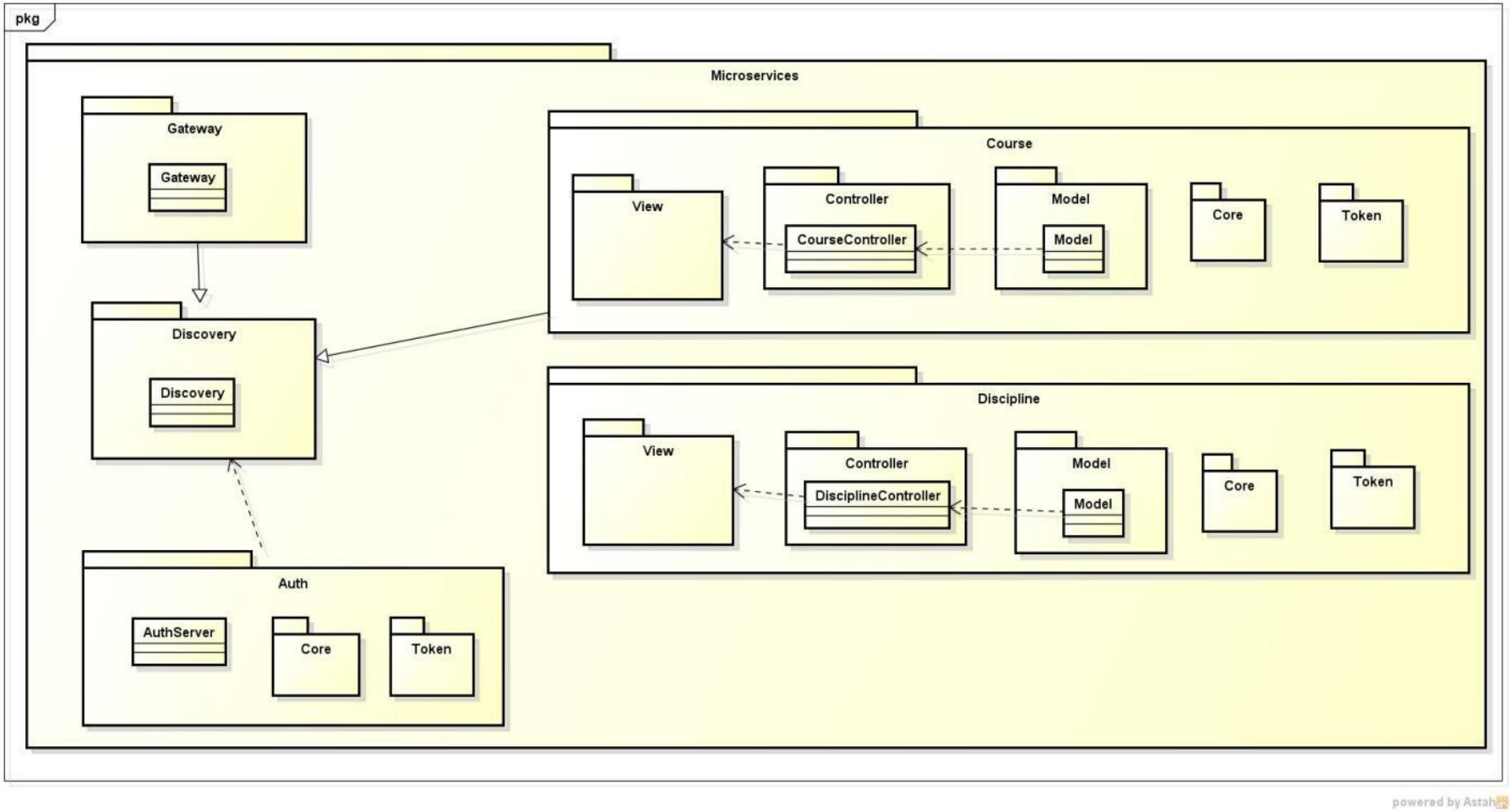
Figura 04 – Diagrama de camadas da arquitetura de Microsserviços



Fonte: Produção Própria dos Autores

Na figura 05 mostra o diagrama de pacotes, nele podemos ver a organização dos pacotes e algumas dependências entre eles, tais como o pacote gateway que necessita do Discovery, dos serviços course discipline e Auth, que também necessitam do Discovery.

Figura 05 – Diagrama de pacotes da arquitetura de Microsserviços

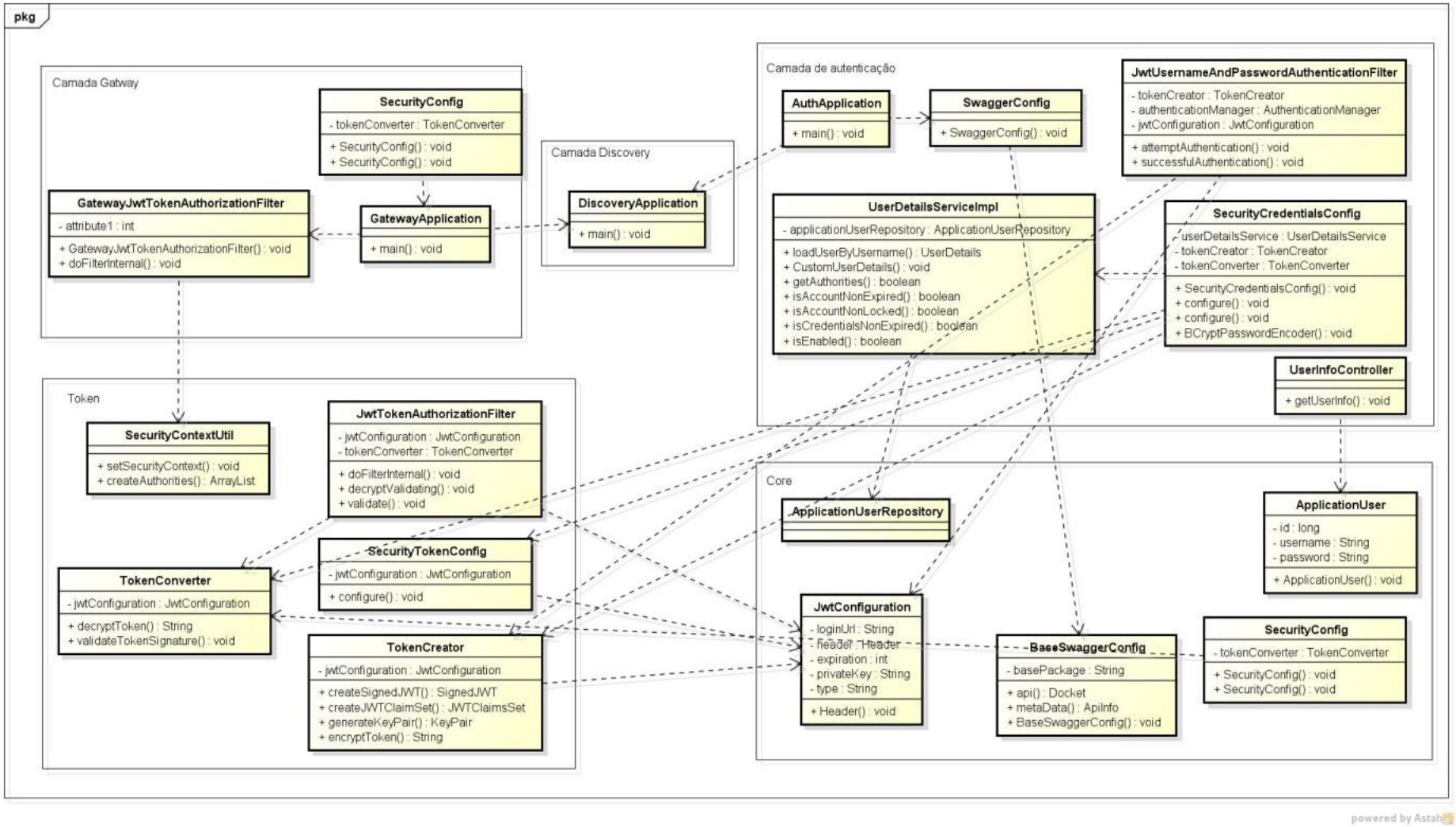


Fonte: Produção Própria dos Autores

5. VISÃO DE IMPLEMENTAÇÃO

No diagrama de Classes abaixo, na figura 06 é mostrado às classes que compõem a arquitetura e seus respectivos atributos e métodos e como eles se relacionam. Elas estão separadas em cinco partes para ficar mais fácil de entender o seu relacionamento, são elas Gateway, Discovery, Auth, Core, Token.

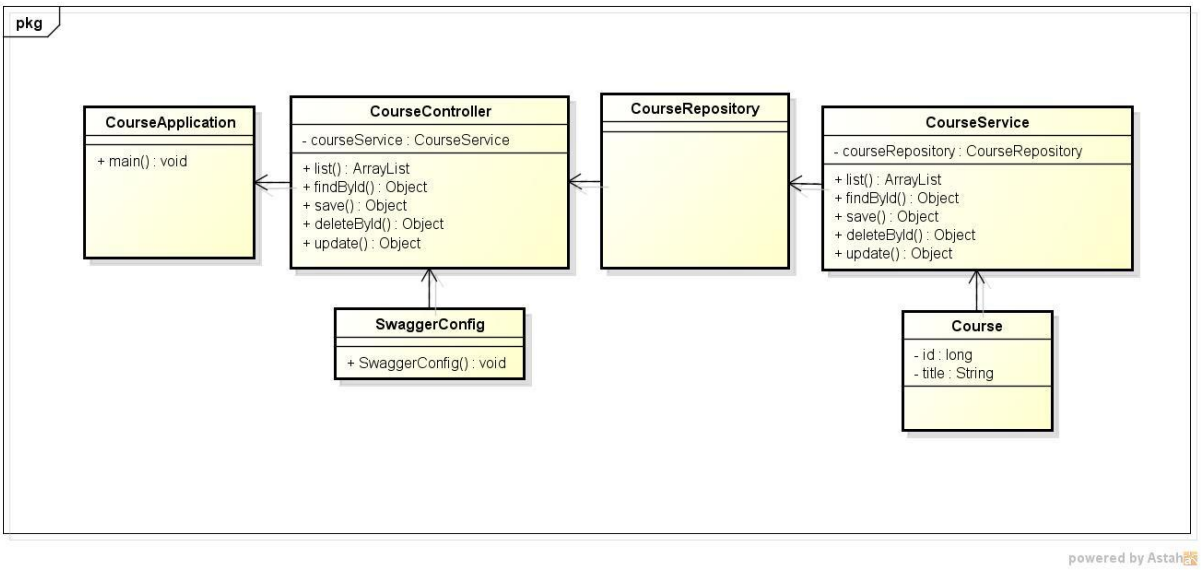
Figura 06– Diagrama de classe arquitetura de Microsserviços



Fonte: Produção Própria dos Autore

O diagrama de Classes exemplificado pela figura 07 compõe às classes do microsserviço Course e seus respectivos atributos, métodos e seus relacionamentos.

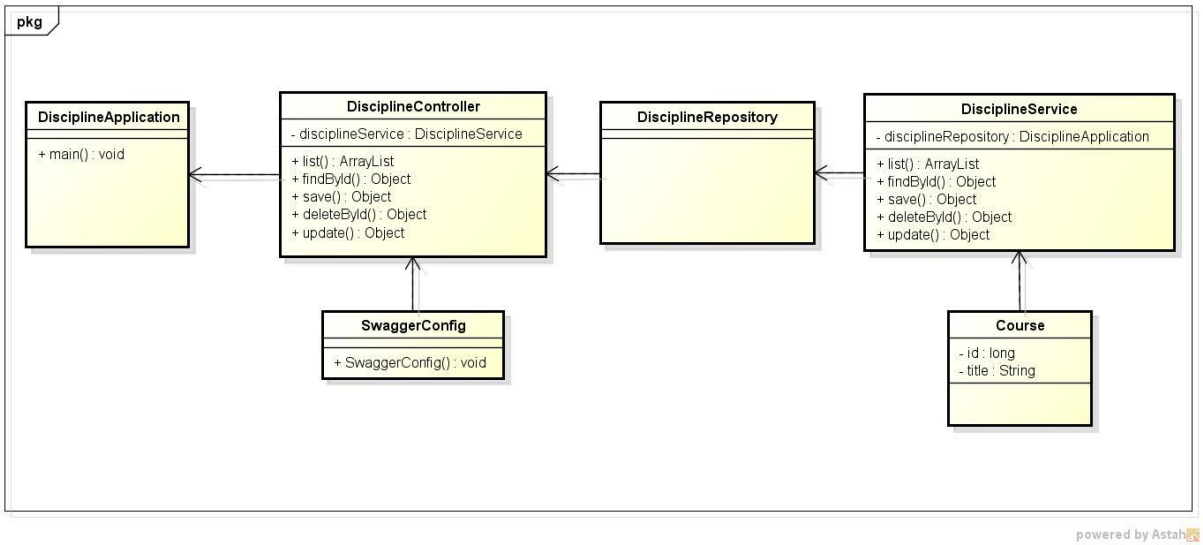
Figura 07 – Diagrama de classe microsserviço Course



Fonte: Produção Própria dos Autores

O diagrama de Classes exemplificado pela figura 08 compõe às classes do microsserviço Discipline e seus respectivos atributos, métodos e seus relacionamentos.

Figura 08 – Diagrama de classe microsserviço Discipline



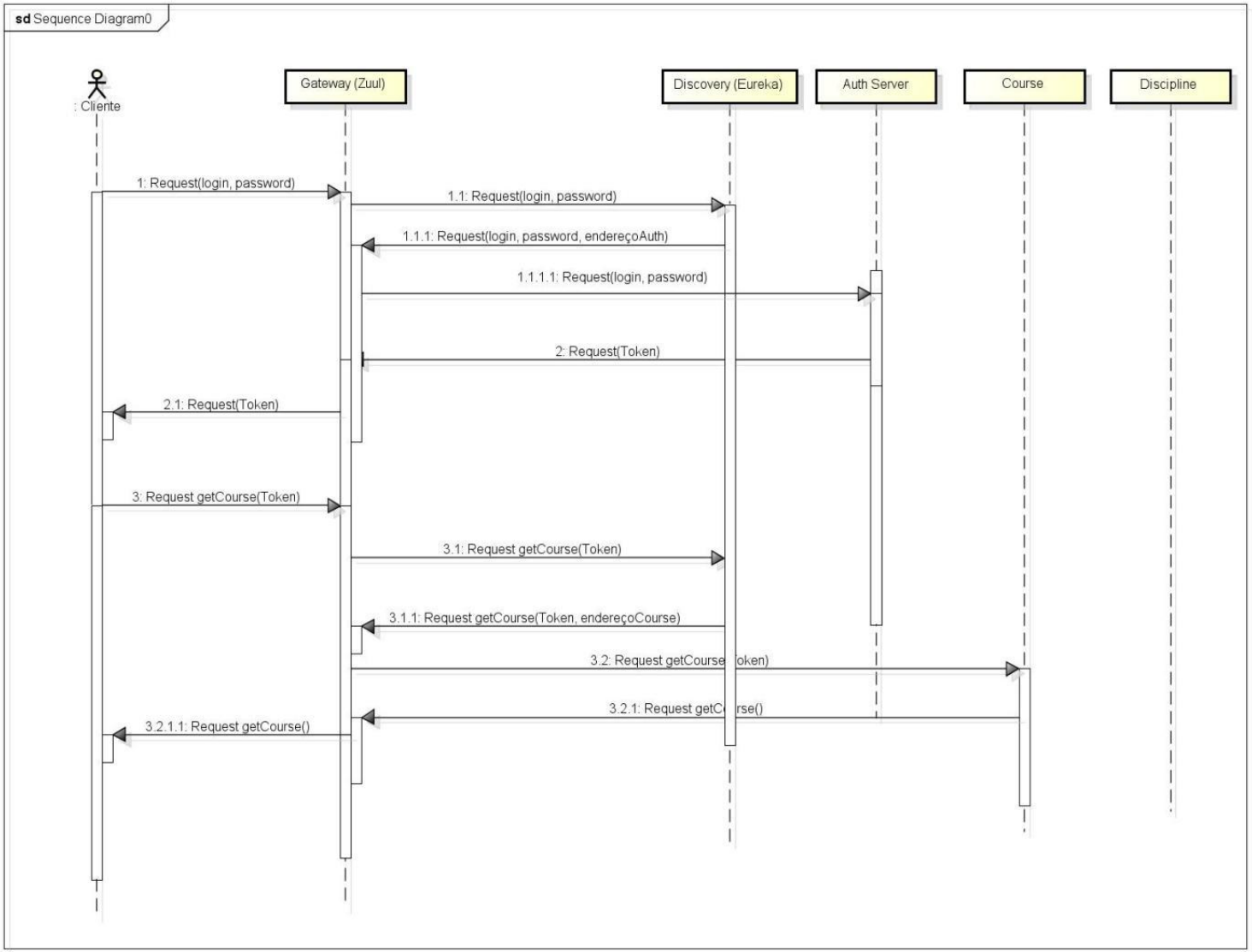
Fonte: Produção Própria dos Autores

6. Diagrama de sequencia

O diagrama de sequencia nesse documento arquitetural representa as interações entre os serviços dentro da arquitetura de microsserviços. Na figura 09 é possível ver o fluxo de dados que começa no cliente. Nesse diagrama o cliente quer todos os registros do serviço Course.

1. Cliente informa login e senha para autenticar.
2. Com essa requisição o Gateway solicita para o Discovery o endereço do Auth Server.
3. O Discovery retorna o endereço para o Gateway.
4. O gateway faz uma requisição para o Auth Server solicitando o Token JWT.
5. O Auth Server verifica se o cliente tem permissão para acessar o serviço, se possuir então é retornado um token para o gateway.
6. O gateway retorna o Token para o cliente.
7. O cliente faz uma requisição para o gateway para acessar o serviço Course.
8. O gateway vai no Discovery e verifica o endereço que o serviço esta localizado.
9. O Discovery retorna o endereço para o gateway.
10. O gateway envia uma requisição para o serviço Course passando o token como parâmetro.
11. No Course verifica se o Token é valido, caso seja retorna as informações solicitadas pelo cliente.

Figura 09 – Diagrama de sequencia da arquitetura de Microsserviços

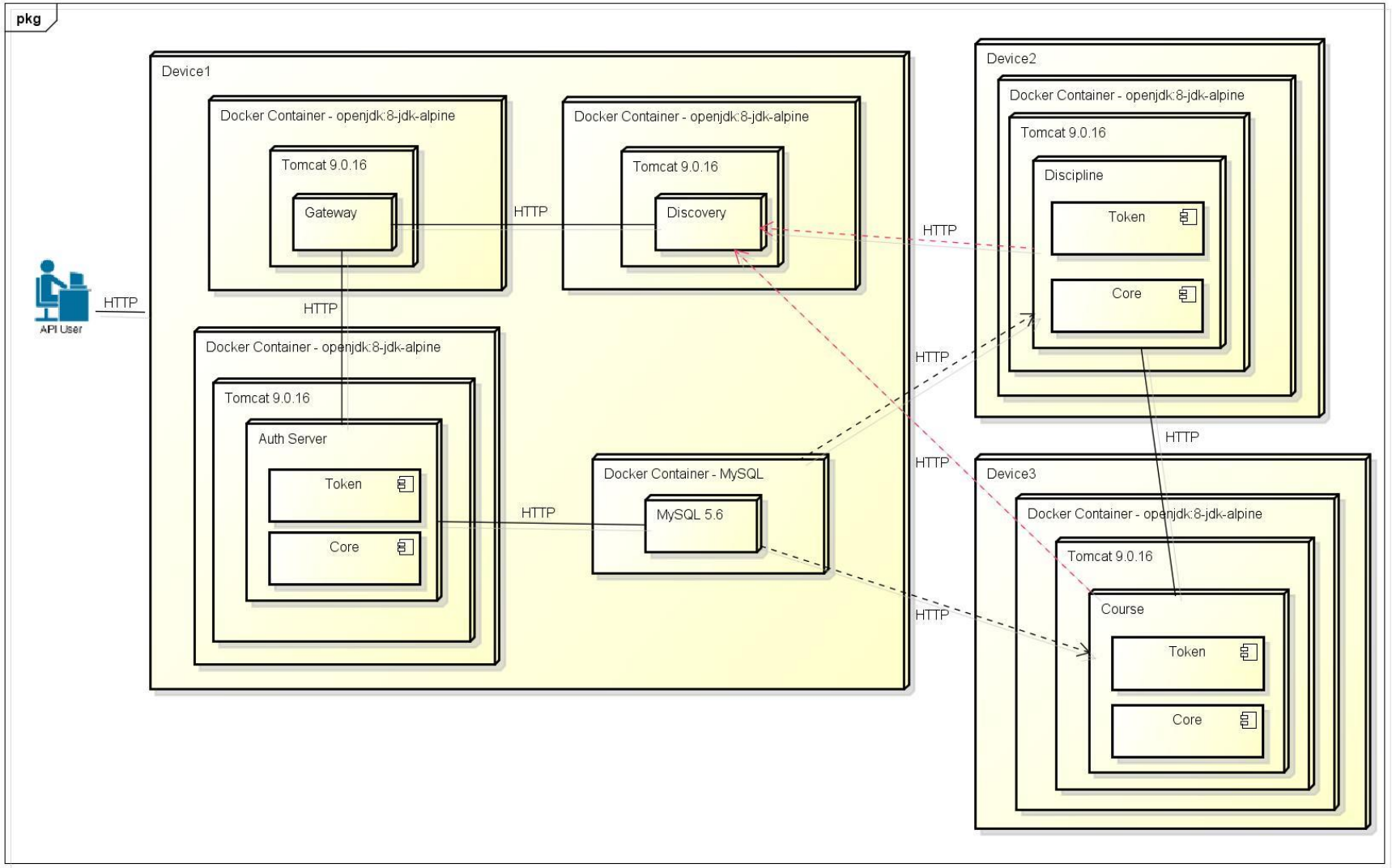


Fonte: Produção Própria dos Auto

7. VISÃO DE IMPLANTAÇÃO

No diagrama de visão de implantação são mostrados os nodos físicos, as configurações e os artefatos que serão implantados. Na figura 10 pode-se observar que existem três *Device* que são os servidores que irão rodar o sistema, em cada servidor esta sendo executada a ferramenta Docker, que cria contêineres com a imagem *openjdk 8-jdk-alpine*. Na imagem esta rodando o Tomcat 9.0.16, que executa o serviço web desenvolvido em Java, tais como Discovery, Gateway, Auth Server entre outros. Já no container banco de dados está rodando a imagem MySQL. A comunicação entre esses serviços é feito através do protocolo HTTP, utilizando REST.

Figura 10 – Diagrama de implantação da arquitetura de Microsserviços



Fonte: Produção Própria dos Autores

7.1 Como executar a arquitetura de microsserviços.

Nesse tópico será descrito o passo a passo para subir a arquitetura de microsserviços. O primeiro passo é instalar o docker e o docker compose. O site oficial do Docker disponibiliza o passo-a-passo para a instalação em sua plataforma de preferencia, seja ele Linux, Mac ou Windows. Segue o link: <https://docs.docker.com/install/>.

Após sua instalação, será necessário fazer o download do projeto disponibilizado no GitLab, o link é: <https://gitlab.com/Joseph.francisco/tcc-arquitetura-de-microsservicos>.

Após o download do projeto, deverão ser configuradas as portas de cada serviço para que não haja nenhum conflito com as portas já em uso em sua máquina. Depois de executar as etapas anteriores, é necessário executar o comando abaixo para fazer o build do projeto.

mvn –U –DskipTests=true clean package dockerfile:build

Após fazer o build do projeto, falta apenas executar o comando no terminal dentro da pasta raiz do projeto baixado para subir o projeto:

docker-compose -f application.yml up

Após a conclusão da execução do comando acima, em outro terminal excute o comando para listar os contêineres que estão sendo executados:

docker ps –a

O resultado deverá ser algo semelhante ao resultado mostrado na imagem 11.

Figura 11 – Lista os contêineres em execução

Fonte: Produção Própria dos Autores

A figura 11 mostra o resultado do comando executado anteriormente. Em caso de sucesso a coluna STATUS deverá mostrar o status UP. Pronto, agora o sistema deve estar rodando e pronto para ser usado.

7.2 Integrando novos serviços na arquitetura de microsserviços

Nesse tópico será descrito como integrar novos serviços a essa arquitetura, o passo a passo do que precisa ser feito, tais como: as configurações que deverão ser feitas, qual protocolo deve ser utilizado para comunicação, os parâmetros que precisam ser enviados e recebidos por esse novo microsserviço dentre outros.

Vale ressaltar que os novos serviços adicionados a esta arquitetura são serviços independentes, sendo desnecessária qualquer alteração nesta arquitetura, pois todas as configurações necessárias devem ser feitas nos serviços que serão adicionados. Essas configurações são descritas abaixo.

Para exemplificar a forma de adicionar um novo serviço para que este fique disponível para os demais serviços desta arquitetura, será utilizado o serviço Course desenvolvido em JAVA, entretanto os novos serviços poderão ser implementados em qualquer linguagem, contudo estes devem se comunicar utilizando REST. Apos a criação do serviço Course foram adicionadas algumas configurações e dependências ao serviço.

Para que os serviços adicionados estejam disponíveis para os demais serviços, será necessário que estes se registrem no componente de descoberta de serviços Discovery. Para isso duas configurações devem ser feitas, a primeira é exemplificado na linha 16 da figura 12, o serviço deve se habilitar como um cliente do Eureka.

Figura 12 – Habilitando o Eureka Client

CourseApplication.java

```
1  package microservice.course;
2
3  import microservice.core.property.JwtConfiguration;
4  import org.springframework.boot.SpringApplication;
5  import org.springframework.boot.autoconfigure.SpringBootApplication;
6  import org.springframework.boot.autoconfigure.domain.EntityScan;
7  import org.springframework.boot.context.properties.EnableConfigurationProperties;
8  import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
9  import org.springframework.context.annotation.ComponentScan;
10 import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
11
12 @SpringBootApplication
13 @EntityScan({"microservice.core.model"})
14 @EnableJpaRepositories({"microservice.core.repository"})
15 @EnableConfigurationProperties(value = JwtConfiguration.class)
16 @EnableEurekaClient
17 @ComponentScan("microservice")
18 public class CourseApplication {
19
20     public static void main(String[] args) { SpringApplication.run(CourseApplication.class, args); }
23
24 }
25
```

Fonte: Produção Própria dos Autores

Na segunda configuração deve ser informada a localização do Eureka, esta é exemplificada na figura 13.

Figura 13 – Configuração do endereço Eureka

```
eureka:
  instance:
    prefer-ip-address: true
  client:
    service-url:
      defaultZone: http://discovery:8081/eureka/
    register-with-eureka: true
```

Fonte: Produção Própria dos Autores

Para a utilização do serviço de autenticação e segurança desta arquitetura, o serviço adicionado deve ter implementado uma classe que recebe e valida o token, segundo os padrões da especificação JWT/OAuth2.0 com algoritmo (JWSAlgorithm.RS256), que é enviado em cada requisição, para isso ele deve ativar as configurações do JWT como ilustrado na linha 15 da figura 14.

As configurações do JWT devem contemplar propriedades mostradas na figura 14.

Figura 14 – Configurações do JWT/OAuth2.0

JwtConfiguration.java

```
1  package microservice.core.property;
2
3  import lombok.Getter;
4  import lombok.Setter;
5  import lombok.ToString;
6  import org.springframework.beans.factory.annotation.Value;
7  import org.springframework.boot.context.properties.ConfigurationProperties;
8  import org.springframework.boot.context.properties.NestedConfigurationProperty;
9  import org.springframework.context.annotation.Configuration;
10
11 @Configuration
12 @ConfigurationProperties(prefix = "jwt.config")
13 @Getter
14 @Setter
15 @ToString
16 public class JwtConfiguration {
17
18     @Value("#{environment.JWT_PRIVATE_KEY}")
19     private String privateKey;
20
21     private String loginUrl = "/login/**";
22     @NestedConfigurationProperty
23     private Header header = new Header();
24     private int expiration = 3600;
25     private String type = "encrypted";
26
27     @Getter
28     @Setter
29     public static class Header {
30         private String name = "Authorization";
31         private String prefix = "Bearer ";
32     }
33 }
34
```

Fonte: Produção Própria dos Autores

Para a decodificação do token é necessária uma chave privada, ela está armazenada na variável de ambiente JWT_PRIVATE_KEY, por questões de segurança ela não é disponibilizada.

Os módulos Token e Core devem ser adicionados como dependências no arquivo pom.xml como ilustrado na figura 15 ou implementados no novo serviço adicionado, as descrições destes módulos estão nos subtópicos 4.2.6 - Módulo Token e no 4.2.7 - Módulo Core.

Figura 15 – Dependências dos módulos Token e Core

```
    <properties>
        <java.version>1.8</java.version>
        <microservice-core.version>1.0-SNAPSHOT</microservice-core.version>
        <microservice-token.version>1.0-SNAPSHOT</microservice-token.version>
        <docker.version>1.4.2</docker.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>microservice</groupId>
            <artifactId>token</artifactId>
            <version>${microservice-token.version}</version>
        </dependency>
        <dependency>
            <groupId>microservice</groupId>
            <artifactId>core</artifactId>
            <version>${microservice-core.version}</version>
        </dependency>
    </dependencies>
```

Fonte: Produção Própria dos Autores

8. QUALIDADE

Item	Solução
Portabilidade	É utilizado o Docker que permite rodar as aplicações e suas dependências em um container portátil, podendo ser executado em diferentes plataformas.
Segurança	É utilizado o token JWT que é recebido após o cliente ser autenticado, este possui uma assinatura que impedi adulterações.
Interoperabilidade	É utilizado o Rest para fornecer interoperabilidade entre vários tipos de sistemas.
Resiliência	Foi utilizado o componente Hystrix para alcançar a resiliência.