UA/DETI • 41951: Análise de Sistemas

António Caetano (114094), Carlos Verenzuela (114597), Hugo Dias (114142), Rúben Gomes (113435)

Grupo 305, v2024/03/18.

RELATÓRIO

Lab 5: Vistas de arquitetura

Exercício 5.1

A)

Neste Diagrama de componentes de uma aplicação de gestão de blogs vemos que o componente Log4j fornece o serviço para fazer logging (“Logger”) ao BlogDataSource. Este vai fornecer DataSources para o ConversionManagement.

Este, por sua vez, fornece os serviços de FeedProvider e DisplayConverter, que vão ser aproveitados pelo BroadcastEngine e pelo BlogViewer, respetivamente.

B)

Os logs são registos de atividades decorrentes do uso do programa (Log4j é uma biblioteca de registo específica) neste caso de uma aplicação Java.

Estas bibliotecas podem ser referenciadas usando coordenadas de dependência em projetos Java que utilizam gerenciadores de dependência, como Maven ou Gradle.

Têm muitas utilidades, como controlar a saída de registos das suas aplicações de forma flexível e configurável, permite aos programadores gerir grandes quantidades de informação de forma fácil, com um desempenho eficiente, entre outras coisas.

No geral, o Log4j é uma ferramenta poderosa que simplifica o processo de registro de mensagens de log em aplicativos Java, fornecendo aos desenvolvedores controle e flexibilidade para gerenciar eficientemente o registro de eventos e diagnosticar problemas de aplicativos.

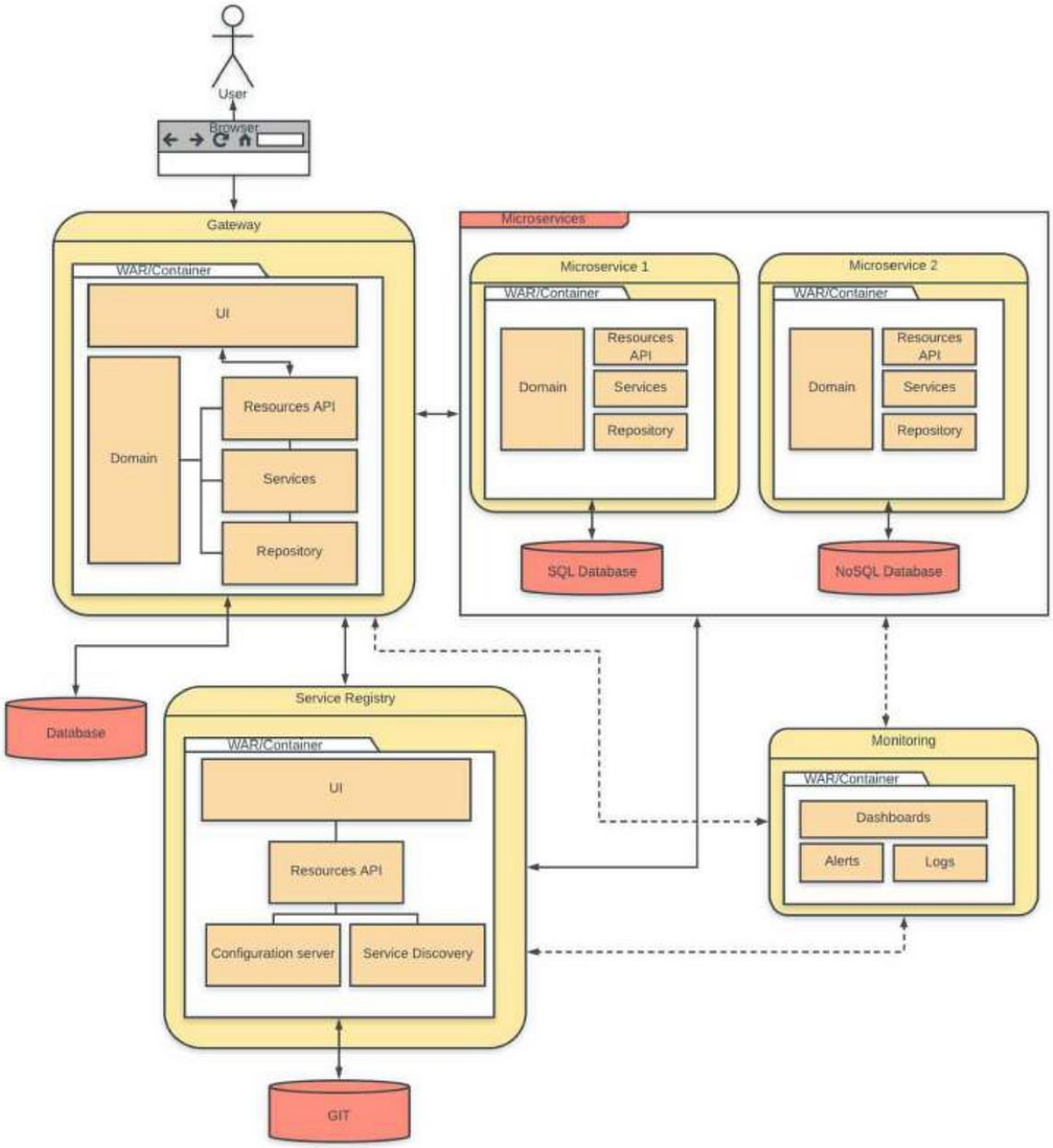
Para adicioná-lo num ficheiro de configuração Gradle (build.gradle):

```
dependencies {
  implementation ‘org.apache.logging.log4j:log4j-core:2.17.1’
}
```

Sendo a versão ‘2.17.1’ a mais recente disponível até o momento.

Exercício 5.2

A) Para este exercício foi considerado o seguinte diagrama, que demonstra uma aplicação web. Este esquema encontra-se abaixo:



Fonte: https://miro.medium.com/v2/resize:fit:1200/1*Bu4lUFyaMcSl_ynyc_IQw.png

Facilmente pode-se identificar os 4 grandes serviços, sendo estes a **Gateway**, os **Microservices**, o **Service Registry** e **Monitoring**. Também temos o User e o Browser, em que o User é a camada mais alta, e logo a seguir vem o Browser (camada de GUI). O Browser apenas tem acesso ao User. Apenas o Browser tem acesso à Gateway, e não ao contrário.

Um aspeto a notar é que todos os serviços existentes estão dentro de containers. Isto visa a isolar o sistema em si que provém o serviço, de modo a não comprometer o mesmo.

Outro grande aspeto é que todos os serviços estão interligados entre si, com acesso bidirecional. Os acessos da parte de Monitoring são a tracejado, que simboliza um acesso não influencial nos resultados, como acontece entre os outros serviços, pois apenas monitoriza o que está a acontecer.

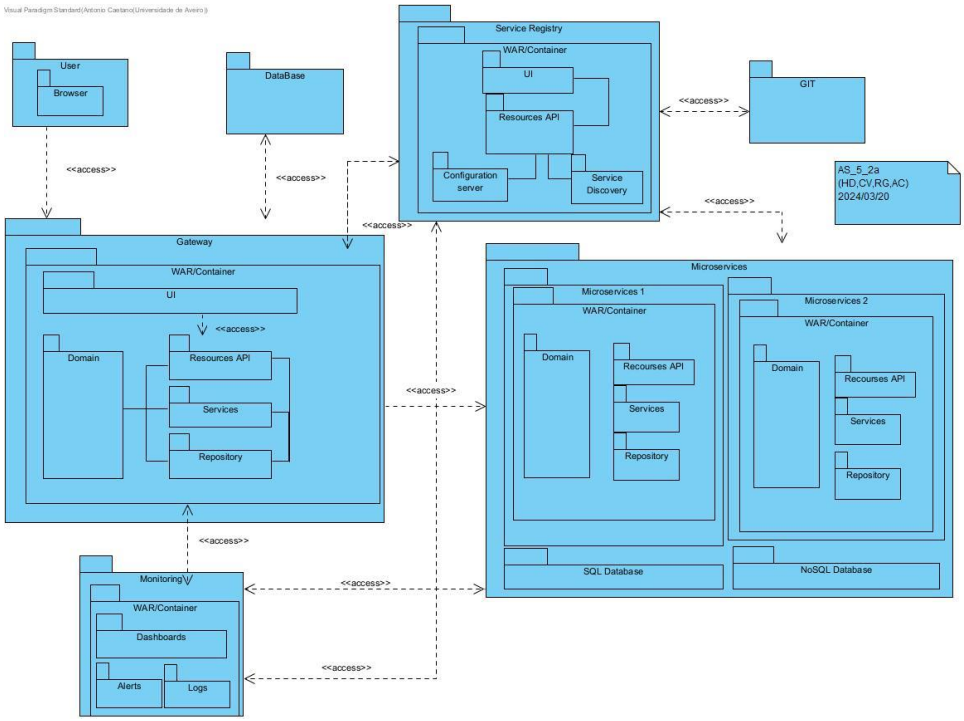
A **Gateway** é o que contém a UI (User Interface), que será interpretada pelo Browser. Esta UI tem acesso aos recursos da API. O Domain do site tem acesso aos recursos da API, Serviços e ao Repositório. Estes todos têm acesso bidirecional à base de dados.

Também podem existir vários **microserviços**. Estes servem para processar os domínios, recursos da API, Serviços e Repositórios. Cada um deles tem acesso bidirecional a uma base de dados individual para cada microserviço.

O **Service Registry** guarda em cache e gere o que é proveniente dos microserviços, escolhendo qual deles está mais atualizado e/ou mais apto para a Gateway utilizar. ISot é feito juntamente com o Configuration Server e Service Discovery. Este está ligado a um repositório git, que será usuado na Gateway.

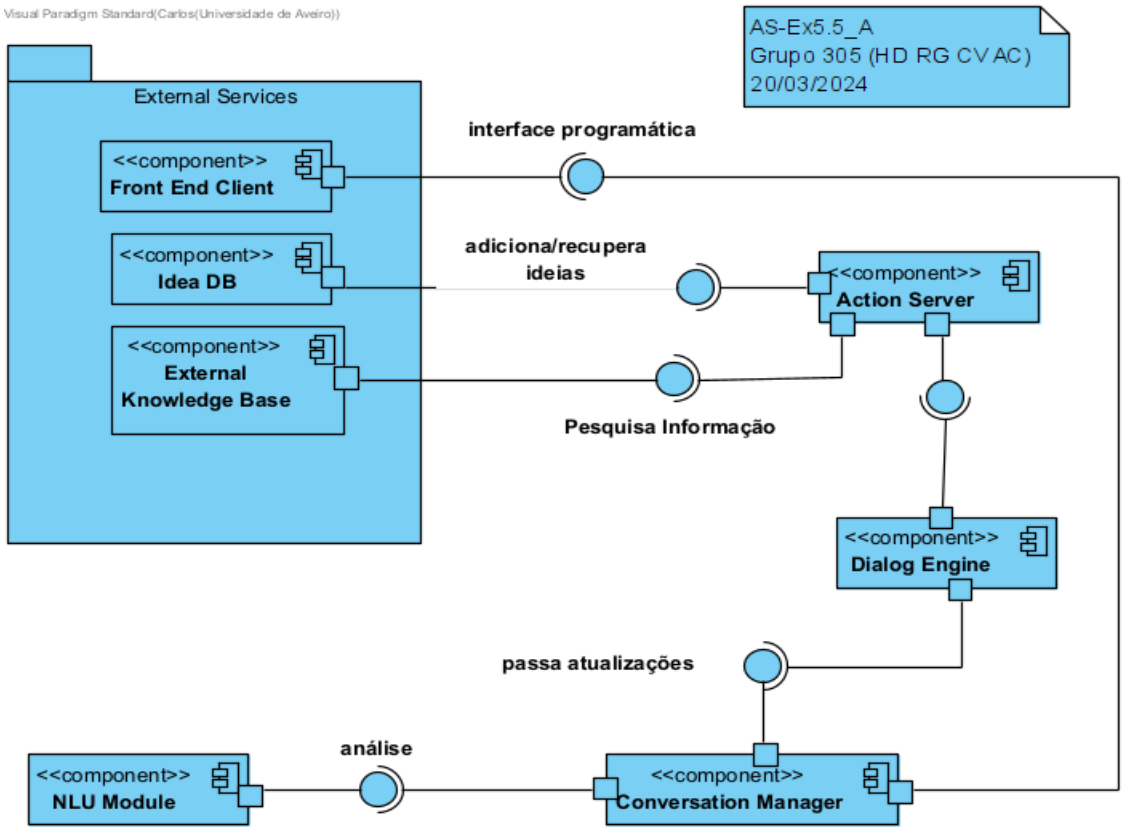
Quanto ao **Monitoring**, este é considerado um serviço que não impacta o funcionamento, mas que tem acesso a todos os serviços, de forma a poder guardar os *logs* e alertas que podem ocorrer em cada serviço. Estes alertas podem ser consultados pelo gestor do serviço através de um eventual Dashboard.

B) Quanto à importação e adaptação do diagrama para UML, este é apresentado em baixo. Não existem grandes mudanças entre ambos, pois foi tido em conta o aspeto/mensagem original do diagrama.


Visual Paradigm Standard(Antonio Caetano(Universidade de Aveiro))
User
Browser
DataBase
Service Registry
WAR/Container
UI
Resources API
Configuration server
Service Discovery
GIT
<<access>>
AS_5_2a
(HD,CV,RG,AC)
2024/03/20
Gateway
WAR/Container
UI
Domain
Resources API
Services
Repository
Microservices
Microservices 1
WAR/Container
Domain
Recourses API
Services
Repository
Microservices 2
WAR/Container
Domain
Recourses API
Services
Repository
SQL Database
NoSQL Database
Monitoring
WAR/Container
Dashboards
Alerts
Logs

Exercício 5.3

A)



O sistema de Chatbot para participação cidadã consiste nos seguintes componentes:

Front End Client: Ponto de entrada para os usuários interagirem com o Chatbot.

Conversation Manager: Gerencia sessões de conversação, coordena interações entre os módulos e oferece uma interface para os clientes manterem uma conversa.

NLU Module: Interpreta a linguagem natural das mensagens dos usuários.

Dialog Engine: Prevê a próxima ação com base na intenção extraída pelo NLU Module.

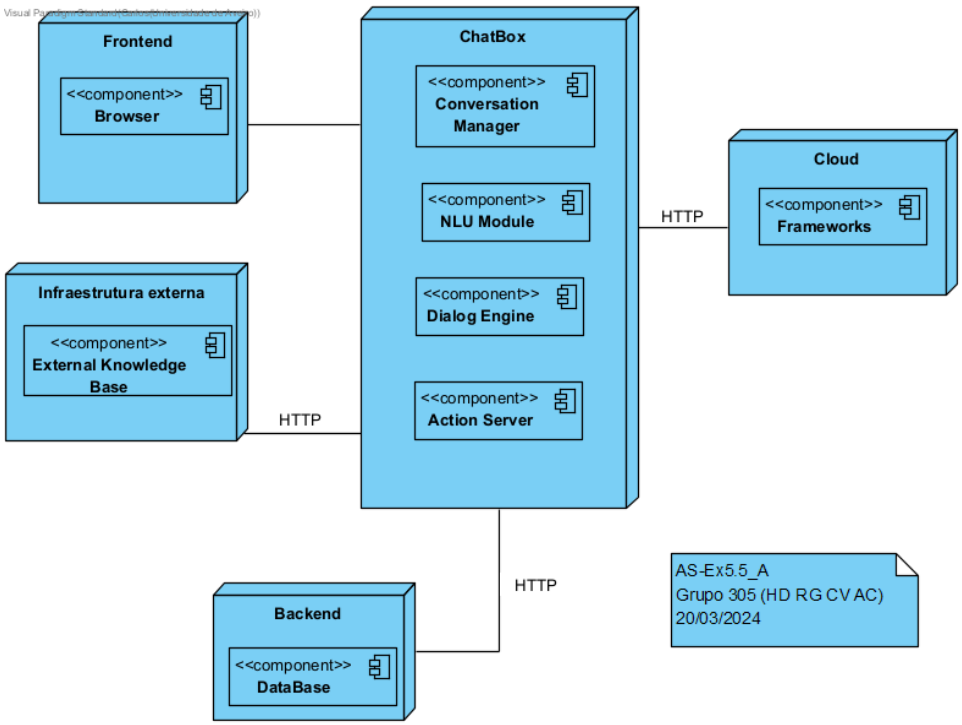
Action Server: Interage com serviços externos, acessa a Idea DB para adicionar e recuperar ideias e consulta a External Knowledge Base para obter informações.

Idea DB: Armazena ideias coletadas dos cidadãos.

External Knowledge Base: Repositório de informações usado pelo Chatbot para fornecer respostas e treinar modelos de IA.

O fluxo de interações começa com o Front End Client, passa pelo Conversation Manager, NLU Module, Dialog Engine e Action Server, conforme necessário, para responder às mensagens dos usuários e executar ações específicas. O Conversation Manager coordena todas as interações e mantém o controle sobre o fluxo da conversa.

B)



Os usuários finais acessam o sistema através de um navegador, interagindo com o Front End Client, que por sua vez se comunica com os outros componentes. Cada componente está implantado em um servidor dedicado, e a comunicação entre eles é realizada

principalmente via HTTP. O NLU Module pode acessar serviços de apoio na nuvem através de REST API.