Desenvolvendo o Sorting Hat: uma Ferramenta para Caracterização de Arquitetura Baseada em Serviços

Erick Rodrigues de Santana[1], **Thatiane de Oliveira Rosa**[1], **João Francisco Lino Daniel**[1], **Alfredo Goldman**[1]

[1] Departamento de Ciência da Computação
Instituto de Matemática e Estatística – Universidade de São Paulo (IME/USP)
Rua do Matão, 1010 – CEP 05508-090 – São Paulo – SP – Brasil

erick.rodrigues.santana@usp.br,

{thatiane,joaofran,gold}@ime.usp.br

Abstract. *Developing large software systems is not trivial, as these systems must satisfy quality attributes such as scalability and maintainability. Therefore, it is important that they have an architecture that favors the fulfillment of these attributes. Understanding the architecture of a software, knowing its structural aspects and patterns, is essential to be able to satisfy the desirable quality requirements. However, there is little support for characterizing and evaluating systems architecture, especially those with service-based architecture. This paper presents the implementation progress of the undergraduate research project regarding Sorting Hat, a tool that assists in the process of characterizing the architecture of service-based systems.*

Resumo. *Desenvolver grandes sistemas de software não é trivial, uma vez que esses sistemas devem possuir atributos de qualidade tais como escalabilidade e manutenabilidade. Diante disso, é importante que eles possuam uma arquitetura que favoreça o atendimento a esses atributos. Compreender a arquitetura de um software – seus aspectos estruturais e padrões – é fundamental para conseguir satisfazer os requisitos de qualidade desejáveis. Entretanto, existe pouco suporte para caracterizar e avaliar a arquitetura de sistemas, especialmente daqueles baseados em serviços. Este artigo apresenta o progresso da Iniciação Científica no desenvolvimento do Sorting Hat, uma ferramenta que auxilia no processo de caracterização da arquitetura de sistemas baseados em serviços.*

1. Introdução

A arquitetura de software é uma área que estuda conceitos e práticas importantes para garantir a qualidade e sucesso de um sistema de software. Uma boa arquitetura permite ter uma visão ampla sobre um sistema de software e sua futura evolução, estimar os custos de infraestrutura e prazos de entrega, dita a estrutura dos times, além de ser base para ensinar novos membros da organização sobre o sistema [Bass et al. 2013]. Já um software desenvolvido sem um planejamento arquitetural adequado poderá ser difícil de ser mantido durante sua evolução. Uma arquitetura pobre é o principal contribuinte para o aparecimento de elementos do software que impedem a capacidade dos desenvolvedores de entender o software [Fowler 2019].

Mesmo com o planejamento de uma arquitetura, pode ser um desafio para os arquitetos e engenheiros manterem o sistema em concordância com ela. Ao longo do processo de desenvolvimento, podem acontecer desvios e erosões arquiteturais, ou seja, decisões na implementação que ferem a arquitetura planejada [Perry and Wolf 1992]. Dessa forma, é valioso manter o registro das evoluções e da arquitetura concreta do sistema, para identificar esses desvios e tomar as decisões corretas.

Existem estilos arquiteturais que podem guiar as equipes no processo de desenvolvimento. O estilo arquitetural de microsserviços é um desses e vem sendo amplamente difundido no contexto de desenvolvimento de software. Embora esse estilo possua benefícios, existem desafios enfrentados em sua implementação. Esses desafios estão fortemente ligados à intrínseca complexidade de aplicações baseadas em microsserviços [Soldani et al. 2018]. Como exemplos, podem ser citadas as dificuldades na determinação da granularidade dos microsserviços e o gerenciamento de armazenamentos distribuídos [Soldani et al. 2018]. Nesse cenário, os desvios e erosões podem ser potencializados, de modo que caracterizar a arquitetura concreta do sistema seja trabalhoso e custoso para a equipe de desenvolvimento.

Este artigo apresenta o progresso da Iniciação Científica no desenvolvimento do Sorting Hat, uma ferramenta em desenvolvimento para caracterização de arquitetura baseada em serviços, que possibilita a coleta automatizada de dados e a exibição de métricas de sistemas de microsserviços, auxiliando no processo de tomada de decisão dos arquitetos em prol da evolução de seus sistemas.

O restante deste artigo está estruturado da seguinte forma: a Seção 2 apresenta um contexto mais detalhado dos conceitos envolvidos no projeto. A Seção 3 explica os objetivos do projeto, bem como a metodologia adotada para atingi-los; em particular, é explicitado o escopo desenvolvido durante a IC. A Seção 4 mostra detalhadamente o estado atual da ferramenta em desenvolvimento, evidenciando seus aspectos arquiteturais e estruturais. A Seção 5 apresenta trabalhos relacionados a este e suas diferenças de abordagens. Por fim, a Seção 6 apresenta as conclusões obtidas, ameaças à validade da ferramenta e trabalhos a serem realizados futuramente.

2. Contexto

Arquitetura de software é o conjunto de estruturas necessárias para a compreensão de um sistema, que abrange elementos do software, a relação entre eles e suas propriedades [Bass et al. 2013]. É uma disciplina dentro da Engenharia de Software que serve de ponte entre os requisitos do sistema – funcionais e não-funcionais – e a implementação do software. O processo de desenvolvimento de software guiado por uma arquitetura planejada facilita aspectos como compreensão, desenvolvimento, manutenção e evolução do sistema, bem como auxilia na tomada de decisões da equipe de desenvolvimento.

Um papel importante a ser desempenhado pelos membros de uma equipe de desenvolvimento de software é o de arquiteto de software. Ele é responsável por conhecer o domínio do negócio, entender o processo de desenvolvimento, conhecer tecnologias e metodologias, além de ter boas habilidades com programação. Esses conhecimentos são essenciais, pois o arquiteto de software também é responsável por tomar decisões em contextos incertos, que afetam a estrutura e a qualidade do sistema.

Para guiar os arquitetos e engenheiros de software na concepção dos elementos do

sistema e suas iterações, é comum a adoção de estilos arquiteturais.

> *"Um estilo arquitetural, então, define uma família de tais sistemas em termos de um padrão de organização estrutural. Mais especificamente, um estilo arquitetural determina o vocabulário de componentes e conectores que podem ser usados em instâncias desse estilo, junto com um conjunto de restrições sobre como eles podem ser combinados."* [Garlan and Shaw 1994]. Tradução própria

O estilo arquitetural de microsserviços é descrito como uma maneira de projetar software como um conjunto de pequenos serviços implantados independentemente, cada um executando em seu próprio processo e se comunicando por mecanismos leves, preferencialmente de caráter assíncrono [Fowler and Lewis 2014]. De acordo com [Newman 2015], algumas das principais características desse estilo arquitetural são as seguintes:

- Serviços pequenos e limitados por domínio do negócio;
- Baixo acoplamento entre serviços;
- Serviços tolerantes a falhas;
- Heterogeneidade tecnológica;
- Infraestrutura de implantação automatizada.

Contudo, desenvolver sistemas seguindo esse estilo arquitetural não é trivial. Alguns questionamentos levantados por [Newman 2015] são:

- Como definir o tamanho e as responsabilidades de cada serviço?
- Como garantir a consistência de dados entre os serviços?
- Como manter um baixo nível de acoplamento entre os serviços?

Esses questionamentos podem levar a incertezas no momento de implementação do sistema. Nesse sentido, podem ocorrer desvios arquiteturais, que são incongruências causadas no momento de implementar a arquitetura planejada [Perry and Wolf 1992]. Para restabelecer a arquitetura planejada, existe o conceito de Recuperação Arquitetural, que consiste em usar técnicas de engenharia reversa para extrair a arquitetura implementada a partir de artefatos de código [de Silva and Balasubramaniam 2012].

Para avaliar a arquitetura atual de sistemas de microsserviços, foi desenvolvido por [Rosa et al. 2020] um modelo de caracterização de arquiteturas baseadas em serviços. Este modelo, que está em constante evolução, atualmente tem como ponto central as seguintes dimensões:

- **Tamanho dos módulos**: o modelo considera que um módulo é uma unidade de implantação de serviços, ou seja, o tamanho de um módulo é a quantidade de serviços que o compõe;
- **Compartilhamento de bases de dados entre módulos**: o modelo considera a distribuição de bases de dados entre módulos, indicando se uma base é utilizada por um ou mais módulos;
- **Acoplamento síncrono entre serviços**: mede o grau de dependência síncrona entre os serviços de um sistema, contabilizando o número de comunicações síncronas entre serviços;
- **Acoplamento assíncrono entre serviços**: mede o grau de dependência assíncrona entre os serviços de um sistema, contabilizando o número de comunicações assíncronas entre serviços.

3. Proposta

O objetivo deste artigo é apresentar o progresso da implementação de uma ferramenta para coleta e exibição de métricas de sistemas que utilizam o estilo arquitetural de microsserviços. Esta ferramenta mostrará o estado atual da arquitetura do sistema e aplicará algumas métricas do modelo de [Rosa et al. 2020] para relacionar a arquitetura com o estilo arquitetural de microsserviços. Dessa forma, ela poderá auxiliar na tomada de decisão dos arquitetos de software. Além do aluno de IC, o trabalho envolve uma equipe composta pela aluna de doutorado Thatiane de Oliveira Rosa – responsável por desenvolver o modelo de caracterização utilizado pela plataforma – pelo aluno de mestrado João Francisco Lino Daniel – um dos desenvolvedores da ferramenta junto ao aluno de IC – e pelo Prof. Dr. Alfredo Goldman – orientador da IC. O escopo da IC abrange a parte de visualização de resultados e de coleta automatizada de dados, que serão discutidos na seção 4.

Para atingir os objetivos deste trabalho de IC, as seguintes etapas foram propostas:

1. **Estudo de literatura**: Estudar os conceitos relacionados a arquitetura de software e estilos arquiteturais, além de investigar diferentes padrões arquiteturais adotados para o desenvolvimento de sistemas que têm arquitetura baseada em serviços;
2. **Codificação do Frontend**: Desenvolver o MVP (*minimum viable prototype*, inglês para protótipo viável mínimo) do *frontend* da plataforma. Com esse MVP, é possível ver as métricas de um sistema;
3. **Integrar dados ao MVP**: Integrar ao MVP dados coletados manualmente, para validar as implementações feitas;
4. **Codificação do protótipo**: Desenvolvimento de um protótipo de coletor automatizado com escopo reduzido e bem definido. Este protótipo tem como finalidade estudar estratégias de coleta automatizada de sistemas de microsserviços.
5. **Integração do coletor com o Frontend**: Integrar o protótipo de coleta automatizada com o *frontend*. Dessa forma, os resultados apresentados pelo coletor poderão ser validados.

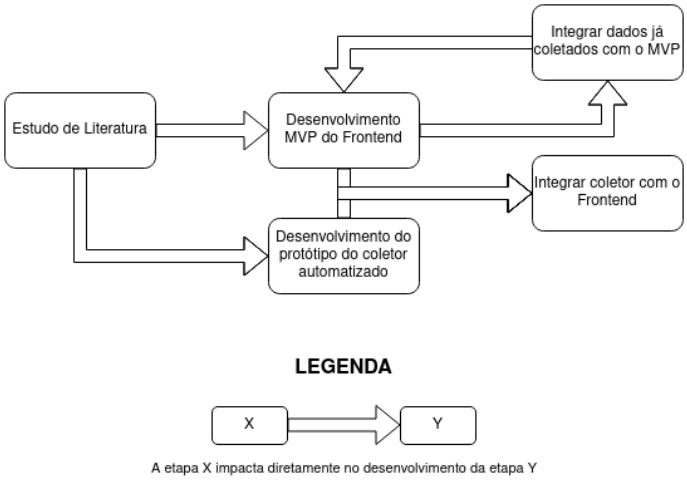

Figura 1. Metodologia proposta

Um esquema visual do fluxo de realização das etapas e da relação entre cada uma destas pode ser visto na Figura 1. É possível ver que o desenvolvimento do MVP

e a integração dos dados com o mesmo é um processo cíclico, pois a integração ajuda a validar a implementação feita, de modo que, caso os resultados não estejam corretos, mudanças deverão ser feitas no *frontend*.

4. Estado Atual

Esta seção apresenta o estado atual do desenvolvimento do Sorting Hat. A Subseção 4.1 apresenta o *frontend*, focado na exibição de métricas de sistemas, enquanto a Subseção 4.2 apresenta o desenvolvimento de um protótipo de coletor automatizado de dados.

4.1. Frontend

O Sorting Hat está em desenvolvimento guiado por protótipos evolutivos. O primeiro MVP do *frontend* pode ser acessado em `https://the-sortinghat-front.herokuapp.com/`. Esse MVP é focado na exibição das métricas de um sistema.

Atualmente, o Sorting Hat conta somente com dados do InterSCity – uma plataforma de suporte para cidades inteligentes desenvolvido com o estilo arquitetural de microsserviços. Esses dados foram coletados em um estudo de caso para validar o modelo de [Rosa et al. 2020]. Neste estudo, foram coletados alguns dados pertinentes como: os serviços e as operações realizadas pelos mesmos, bases de dados e as comunicações síncronas e assíncronas realizadas. Esses dados foram obtidos através de inspeções manuais no código-fonte do InterSCity. Dessa forma, foi possível desenvolver o *frontend* sem que a ferramenta de coleta estivesse pronta.

O *frontend* do Sorting Hat é composto por 3 níveis de visualização: um sistema, um módulo de um sistema e um serviço de um módulo. Para cada um desses três níveis existe uma página. As páginas de um sistema e de um módulo possuem duas formas de visualização: a visualização em grafo, que mostra as comunicações síncronas e assíncronas entre os módulos ou serviços; e a lista de métricas, que mostra todas as métricas pertinentes ao respectivo nível de visualização. Exemplos destas visualizações são mostradas nas Figuras 2 e 3.

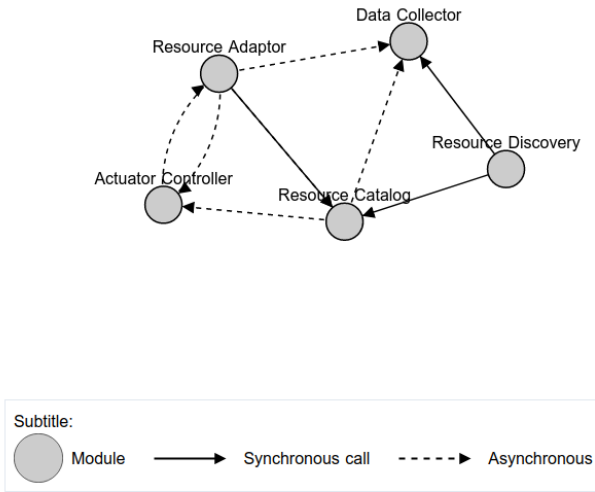

Figura 2. Visualização em grafos dos módulos do InterSCity.

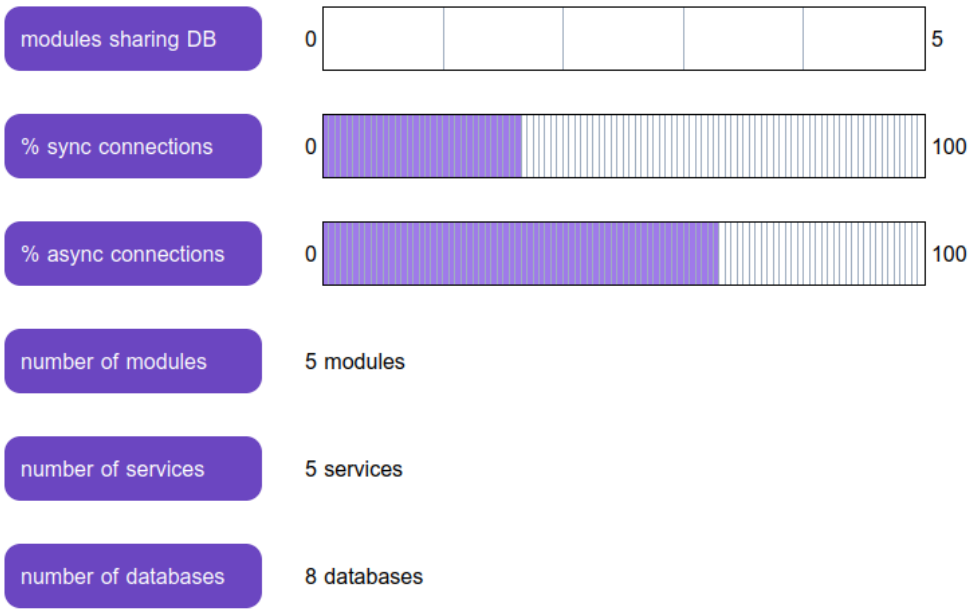

Figura 3. Visualização da lista de métricas do InterSCity.

A visualização em grafos explicita as conexões síncronas e, ao observar isso, os arquitetos podem tomar decisões de mantê-las ou de migrá-las para assíncronas, que são mais condizentes com o estilo arquitetural de microsserviços. Comunicações síncronas podem gerar dependência entre os serviços em tempo de execução e aumentar o acoplamento entre os mesmos.

A visualização da lista de métricas também pode auxiliar os arquitetos na tomada de decisões. Por exemplo, a Figura 3 ilustra a métrica *modules sharing DB*, que diz a quantidade de módulos que compartilham um mesmo banco de dados. Um dos padrões que guiam o estilo arquitetural de microsserviços é o *Database per service*, que diz que cada serviço deve possuir sua própria base de dados privada. O compartilhamento de bases de dados pode gerar inconsistências, indo de desencontro com as práticas adotadas no estilo arquitetural em questão. Dessa forma, caso a Figura 3 mostrasse que existem módulos compartilhando banco de dados, os arquitetos poderiam decidir remover esses compartilhamentos ou mantê-los caso necessário.

É importante notar que, apesar de não coletar dados de sistemas automaticamente, o *frontend* gera essas visualizações que podem ser pertinentes aos arquitetos na tomada de decisões futuras.

A arquitetura do *frontend* da plataforma é representada na Figura 4. Ela ilustra uma visão geral da estrutura de páginas, mostrando o fluxo de navegação do usuário. A Figura 4 também apresenta os componentes que as páginas utilizam, bem como a interação do *frontend* com elementos externos, como o *backend*, responsável por obter os dados dos sistemas disponíveis na plataforma.

Para desenvolvê-lo, foi utilizado o Nuxt.js, um framework para desenvolvimento Web que permite a construção de SPAs (*single-page applications*, traduzido para aplicações de página-única), uma forma moderna de desenvolver sites em que a navegação entre páginas é feita de maneira mais fluida, sem a necessidade do *browser* realizar uma

nova requisição para o servidor onde o site está hospedado a cada mudança de página. A construção de SPAs também é baseada em componentes, uma parte da aplicação que é comum a várias páginas e que, portanto, pode ser reutilizada. Além disso, o Nuxt.js permite ao desenvolvedor utilizar *plugins* – bibliotecas ou módulos externos à aplicação e que são disponibilizados a ela.

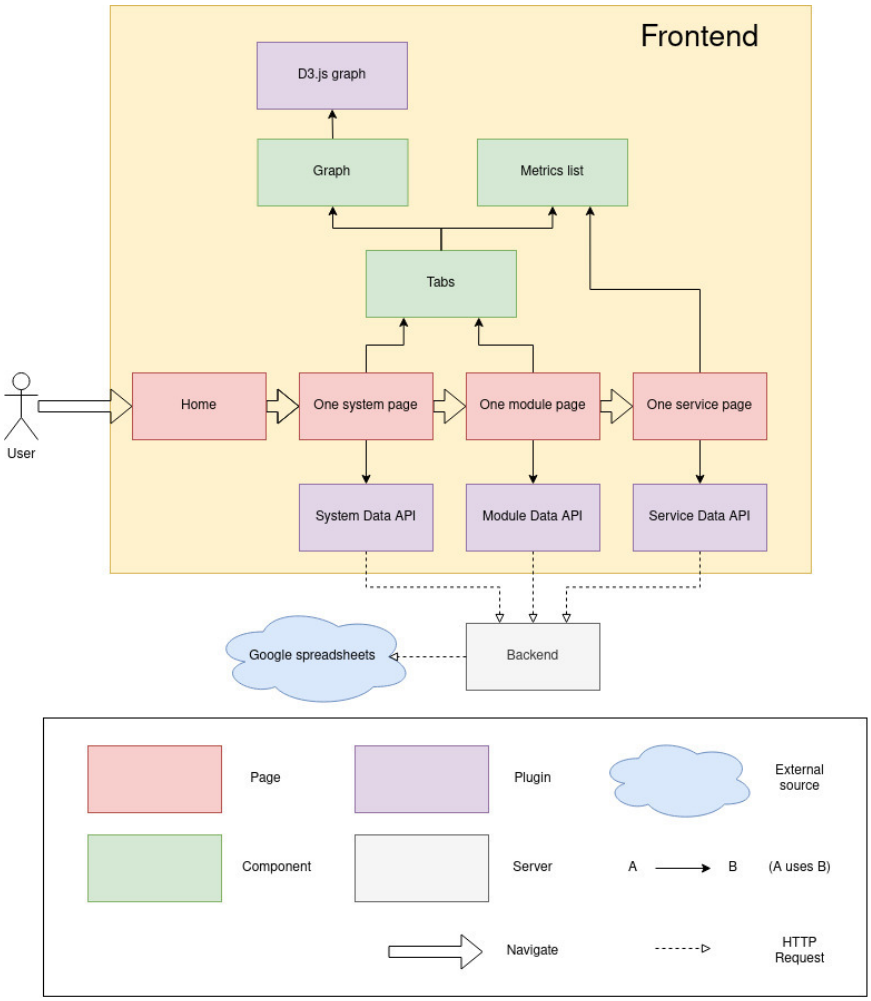

Figura 4. Arquitetura do frontend da plataforma.

Na Figura 4, estão representadas as três páginas mencionadas anteriormente, que mostram os diferentes níveis de visualização. As páginas de um sistema e um módulo utilizam os componentes de grafo e métricas. Para a visualização em grafo, foi utilizada como *plugin* a biblioteca D3.js, que permite a construção de diversas formas de visualização a partir dos dados brutos que a aplicação pretende mostrar.

Nesse MVP, somente os dados do InterSCity estão disponíveis. Esses dados foram armazenados em planilhas do Google, por motivos de facilidade. Para a plataforma obtê-los, foram criadas algumas abstrações: cada uma das três páginas utiliza um *plugin* para requisitar os dados do *backend* e transformá-los para a forma que as páginas entendem. Dessa forma, cada um dos *plugins* atua como uma abstração do servidor. O *backend* tem um objetivo similar: requisita os dados das planilhas do Google e os transforma para objetos que são entendidos por cada um dos *plugins*. Todas essas abstrações foram

criadas com objetivo de modularizar a aplicação, provendo ganhos de testabilidade, de desacoplamento e de escalabilidade.

A plataforma tem licença de código aberto e o código-fonte está disponível em: `https://github.com/the-sortinghat/frontend`. O *frontend* conta atualmente com 2 desenvolvedores ativos e mais de 2700 linhas de código escritas. O MVP do *frontend* foi apresentado na quarta edição do Workshop do InterSCity.

4.2. Protótipo do Coletor Automatizado

Um protótipo de sistema de coleta automática de dados está sendo desenvolvido. Ele possui um escopo reduzido e bem definido e tem como finalidade ser um objeto de estudo de estratégias de coleta automatizada de dados.

O protótipo se propõe a identificar o padrão de microsserviços *Database per service*, em que cada serviço possui sua própria base de dados que só ele tem acesso. Como entrada de dados, é necessário somente a URL de um repositório do Github que contenha um arquivo do *docker-compose*, que descreve os contêineres Docker que representam serviços e banco de dados de um dado sistema. A partir desse arquivo, o protótipo, em uma série de etapas, conseguirá identificar quais os usos de banco de dados por cada serviço, verificando assim se o sistema de microsserviços utiliza o padrão *Database per service*.

Uma visão geral da arquitetura do coletor é mostrada na Figura 5.

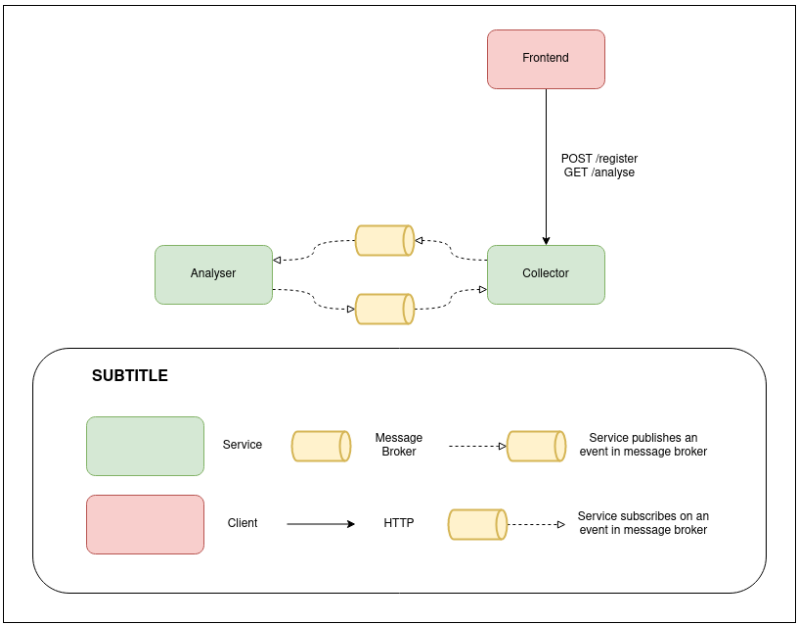

Figura 5. Arquitetura do protótipo do coletor automatizado.

No `Frontend`, o usuário consegue enviar a URL do repositório do Github desejado, além de conseguir ver os resultados analisados pelo protótipo. O `Collector`, por sua vez, expõe duas rotas: `POST /register` recebe a URL enviada pelo `Frontend` e registra o sistema, enviando os dados de serviços e bancos de dados para o `Analyser`, responsável por identificar o *Database per service*. Após terminar de analisar os dados, o

`Analyser` envia os resultados ao `Collector`. O `Frontend` consegue ter acesso a esses dados ao fazer uma requisição `GET /analyze`. É importante notar que o protótipo utiliza a arquitetura de microsserviços. O Collector e o Analyzer são serviços que se comunicam de maneira assíncrona, publicando e escutando eventos em um *Message Broker*.

O coletor automatizado ainda não atingiu a maturidade do primeiro MVP e, portanto, não está disponível para uso. Porém, tem licença de código aberto e o código-fonte pode ser acessado em: `https://github.com/the-sortinghat/ProtoINCT`. O coletor conta atualmente com 3 desenvolvedores ativos e mais de 2000 linhas de código escritas.

5. Trabalhos Relacionados

O Sorting Hat está sendo desenvolvido com base no modelo de [Rosa et al. 2020]. Nesse sentido, existem alguns trabalhos já feitos que se aproximam do apresentado neste artigo e do modelo proposto por [Rosa et al. 2020].

O MicroART é um protótipo implementado com base em uma abordagem de recuperação de arquitetura de sistemas de microsserviços e tem como objetivo auxiliar no entendimento da complexidade da arquitetura desses sistema. Essa abordagem foi desenvolvida por [Granchelli et al. 2017], que coletaram e analisaram dados de métricas estáticas e dinâmicas do sistema de *benchmark* Acme Air.

O MAAT (*Microservice Architecture Analysis Tool*) é uma ferramenta que possibilita a visualização de arquitetura de sistemas de microsserviços. Foi desenvolvida com base na abordagem de [Engel et al. 2018], que é fundamentada em princípios como acoplamento flexível e serviços pequenos. A partir da definição desses princípios, foram derivadas as métricas a serem adotadas para avaliar a arquitetura e identificar os pontos críticos que requerem atenção.

Dessa forma, é possível ver que, apesar de semelhantes, os estudos discutidos não utilizam a abordagem da plataforma Sorting Hat. A ferramenta apresentada neste artigo visa essencialmente caracterizar a arquitetura de sistemas de microsserviços considerando diferentes aspectos estruturais e métricas interoperáveis e viáveis para o contexto de cada dimensão proposta no modelo de [Rosa et al. 2020]. Além disso, nenhum dos estudos discutidos apresenta algum indício de coleta automatizada de dados de sistemas de microsserviços, o que é um diferencial proposto pela ferramenta Sorting Hat.

6. Conclusões

Neste artigo foi mostrado o Sorting Hat, uma ferramenta que auxilia a exibição de métricas de sistemas que utilizam o estilo arquitetural de microsserviços. Esta ferramenta mostrará o estado atual da arquitetura do sistema e aplicará algumas métricas do modelo de [Rosa et al. 2020] para relacionar a arquitetura com o estilo arquitetural de microsserviços. Dessa forma, ela poderá auxiliar na tomada de decisão dos arquitetos de software. O desenvolvimento do *frontend* da plataforma e do protótipo do coletor estão dentro do contexto de IC.

Com os dados coletados manualmente do InterSCity, o MVP do Frontend se mostrou eficaz na exibição dinâmica das relações de módulos e serviços do sistema, bem como das principais métricas do mesmo. Dessa forma, já é possível realizar experimentos com a

equipe de desenvolvedores do InterSCity para validar os resultados obtidos. Com relação ao protótipo do coletor automatizado de dados, ainda não é possível validá-lo e obter conclusões acerca dos resultados produzidos, pois ele está em fase de desenvolvimento e com um escopo reduzido.

Como só existem dados do InterSCity disponíveis, ainda não é possível ter certeza se o MVP do Frontend conseguirá efetivamente exibir métricas de outros sistemas, o que é uma ameaça à validade do projeto.

Como trabalhos futuros a serem realizados, temos o término do protótipo do coletor automatizado. Após isso, será feita uma expansão no escopo do coletor, com a finalidade de aplicar novas estratégias de coleta de dados. Em paralelo a isso, mudanças pontuais serão feitas no *frontend*. Além disso, novos estudos sobre visualização de dados serão realizados a fim de encontrar maneiras melhores de se exibir as métricas dos sistemas da plataforma. Por fim, novos dados de sistemas serão coletados – de maneira manual – para validar as novas implementações.

Referências

Bass, L., Clements, P., and Kazman, R. (2013). *Software Architecture in Practice*. Addison-Wesley, 3rd edition.

de Silva, L. and Balasubramaniam, D. (2012). *Controlling software architecture erosion: A survey*. Journal of Systems and Software.

Engel, T., Langermeier, M., Bauer, B., and Hofmann, A. (2018). *Evaluation of Microservice Architectures: A Metric and Tool-Based Approach*. in Information Systems in the Big Data Era.

Fowler, M. (2019). *Software Architecture Guide*. Url: `https://martinfowler.com/architecture/`.

Fowler, M. and Lewis, J. (2014). *Microservices*. Url: `https://martinfowler.com/articles/microservices.html`.

Garlan, D. and Shaw, M. (1994). An introduction to software architecture. *Carnegie Mellon University*, (CMU-CS-94-166).

Granchelli, G., Cardarelli, M., Francesco, P. D., Malavolta, I., Iovino, L., and Salle, A. D. (2017). *Towards Recovering the Software Architecture of Microservice-Based Systems*. IEEE International Conference on Software Architecture Workshops (ICSAW 2017).

Newman, S. (2015). *Building Microservices: Designing Fine-Grained System*. O'Reilly Media, 1st edition.

Perry, D. E. and Wolf, A. L. (1992). *Foundations for the Study of Software Architecture*. ACM SIGSOFT Software Engineering Notes.

Rosa, T., Goldman, A., and Guerra, E. (2020). *Modelo para Caracterização e Evolução de Sistemas com Arquitetura Baseada em Serviços*. Workshop de Teses e Dissertações do CBSoft - WTDSoft 2020.

Soldani, J., Tamburri, D. A., and Heuvel, W.-J. V. D. (2018). The pains and gains of microservices: A systematic grey literature review. *Journal of Systems and Software*, 146.