Arquiteturas de Sistemas Computacionais

Professor Rodrigo Bossini

Conteúdo

Capítulo 1

Introdução

Neste material são abordados conceitos relacionados à **Arquitetura de Software**. O primeiro passo é fazer uma definição precisa, apropriada para o contexto desejado. Essa definição caracteriza aquilo que se espera de um **Arquiteto de Software**. A seguir, em uma abordagem prática, o material trata dos seguintes tópicos.

- Padrões Arquiteturais. Padrão Arquitetural REST.
- Serviços Restful.
- Arquitetura cliente/servidor.
- Microsserviços.
- Containers e orquestração.

Capítulo 2

Arquitetura de Software

Há diferentes definições para **Arquitetura de Software**. Diferentes autores produzem definições sutilmente diferentes. É óbvio que a definição deve estar de acordo com aquilo que se pretende abordar no texto. Há, também, as definições advindas do mercado de trabalho. Elas são importantes pois caracterizam as funções que espera-se sejam desempenhadas por um **Arquiteto de Software**. Não se trata de uma ciência exata e não há definição correta ou incorreta. Há definições diferentes, cada uma apropriada para um contexto específico.

2.1 Definição acadêmica A seguir, apresentamos algumas definições para Arquitetura de Software oferecidas por diferentes autores.

Definição

Segundo Armando Fox et. al., a **Arquitetura de Software** descreve como os sub-sistemas que constituem um software interagem entre si a fim de tornar disponíveis os requisitos funcionais e não funcionais[2].

Definição

Segundo Len Bass et. al., a **Arquitetura de Software** de um sistema é o conjunto de estruturas necessário para que se possa refletir sobre ele, o que abrange elementos de software, relações entre eles e as propriedades de ambos.[1].

Definição

Segundo Mark Richards, et. al. a **Arquitetura de Software** é caracterizada pela **estrutura** do sistema, combinada com as **características arquiteturais** às quais o sistema deve dar suporte, pelas **decisões arquiteturais** e pelos **princípios de design**. Nesta definição

- a **estrutura do sistema** diz respeito ao estilo arquitetural utilizado em sua implementação (como **microsserviços**, em **camadas** etc).
- as **características arquiteturais** definem aquilo que é fundamental para o sucesso do sistema, como **escalabilidade**, **tolerância a falhas**, **desempenho** etc.
- as **decisões arquiteturais** definem as regras segundo as quais o sistema deve ser construído. Por exemplo, pode-se especificar que somente componentes da camada de serviço têm acesso direto à base de dados.
- os **princípios de design** são semelhantes às decisões arquiteturais. Entretanto, não são regras absolutas. Tratam-se apenas de recomendações que podem ser aplicadas pelos desenvolvedores em situações específicas. Por exemplo, fazer uso de troca de mensagens assíncronas entre serviços para melhorar o desempenho, sempre que possível.

Finalmente, Martin Fowler escreveu um famoso artigo chamado **Who Needs an Architect?**[3] que vale olhar. Parte do texto chega à seguinte conclusão.

Definição

A **Arquitetura de Software** é caracterizada pelas coisas importantes. Seja lá o que elas forem.

2.2 Definição no mercado de trabalho O profissional "Arquiteto de Software" tem grande demanda no mercado. Buscas em portais de vagas conhecidos trazem inúmeras oportunidades para esse tipo de profissional, algumas vezes com sutis variações no nome. Ocorre que as necessidades de cada empresa variam em função da natureza de suas atividades. Por isso, **caracterizar a Arquitetura de Software com base naquilo que se vê no mercado é tão dependente de contexto** quanto o que ocorre na definição mais acadêmica. Vejamos alguns

exemplos de oportunidades para esse tipo de profissional, especificamente no Brasil[1].

- O Arquiteto de Soluções e Softwares deve ter experiência em múltiplos ambientes de hardware e software e estar confortável com ambientes de sistemas heterogêneos complexos. Deve ser um tecnocrata sênior altamente experiente na liderança e definição arquitetural de soluções e softwares alinhadas às necessidades de negócio. O profissional deve ter capacidade de partilhar e comunicar ideias com clareza, oralmente e por escrito à equipe executiva, aos patrocinadores e as equipes técnicas envolvidas no projeto.

- Necessário sólida experiência em desenvolvimento (fullstack), arquitetura e processos em várias tecnologias. Vivências como líder de time, prática com arquiteto de aplicações para soluções WEB e de Micros serviços. Conhecimento em arquitetura Cloud (Azure/ AWS). Conhecimentos em ASP.NET e C, outras API Web.

- Desenvolvemos soluções digitais sob medida para o negócio do cliente. Nossa metodologia de trabalho tem sua base na Experiência do Usuário (UX) e o Design Thinking para a concepção de soluções, desenvolvimento pautado nas metodologias ágeis com entregáveis recorrentes resultando em soluções robustas, de alta performance, escaláveis e compliance com requisitos de segurança. Buscamos um profissional que terá responsabilidades desde nível de negócio (Engenharia de Software) até Arquitetura de Software e DevOps/Infra com mais de 3 anos de experiência na função.

- Experiência em definição de arquitetura de soluções de alta disponibilidade (HA), escaláveis e microsserviços, desenvolvimento de interfaces através da construção de API. Linguagens: NodeJS, ReactJS, React Native. Lógica de programação avançada. Metodologias ágeis. Acompanhar o mercado e as novas tecnologias. Hands-on na fase de codificação.

Note como a definição também pode variar bastante. Algumas delas, observe, têm bastante relação com as definições encontradas em livros. Note também que, muitas vezes, um profissional que levaria o título de **Desenvolvedor Sênior** em uma empresa pode, em outra, ser considerado um **Arquiteto de Software** ou **Arquiteto de Sistemas**. O mesmo ocorre com outros títulos como **Engenheiro de Software** e similares. Muitas oportunidades deixam claro que o profissional deve ter **condições de colocar a mão na massa!**. Ou seja, conhecer diversas linguagens de programação e, de fato, programar. Isso não é diferente com

[1] Busca realizada no site APINFO em fevereiro de 2021.

oportunidades encontradas em outros países[2]. Veja alguns exemplos.

- Docker/Kubernetes, Cloud (AWS, GCP), Data Engineering, DevOps, ML background, Python, ML Ops tools - ML Flow/Kubeflow, ML frameworks - TensorFlow/PyTorch/Keras, ML Testing. Experienced in Python, SQL. Experience with Hadoop infra and NoSQL databases like Cassandra, MongoDB. Experience in building docker container-based solutions – Doker, Kubernetes.

- Experience with cloud/SaaS, big data, or analytics and Machine Learning. Excellent communication skills: Demonstrated ability to present to all levels of leadership, including executives. Expertise with modern technology stacks, microservices, public cloud and programming languages. Familiarity with the FinTech and how technology can be used to solve problems in the personal finance space. Expertise with Agile Development, SCRUM, or Extreme Programming methodologies.

- Application Architect on a team; primary role to design and develop secure web-based applications. Establish work necessary for a complete product; break work down into discrete tasks and deliverables. Code, troubleshoot, test, and maintain core product software and databases, to ensure strong optimization and secure functionality. Knowledge of software architecture and design patterns, and the ability to apply them. Experience in web-based technologies like Microsoft.Net, ASP.NET, JavaScript, C, VB.Net, Web API and complementary business layer and front-end technologies. Strong problem-solving skills.

[2]Busca realizada no site INDEED em fevereiro de 2021.

Professor Rodrigo Bossini
professorbossini@gmail.com

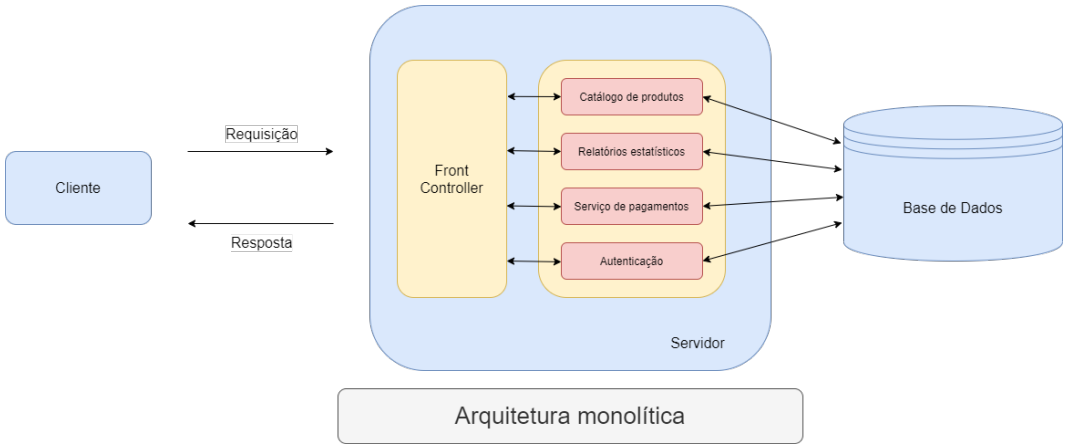
Capítulo 3

Microsserviços

Neste capítulo trataremos de um dos tipos de arquiteturas mais utilizados atualmente: a **Arquitetura baseada em microsserviços**.

3.1 Arquitetura Monolítica Para entender o que é uma **Arquitetura baseada em microsserviços**, vamos falar de um outro tipo de arquitetura, muitas vezes denominada **Arquitetura Monolítica**. Esse tipo de arquitetura foi muito utilizado nas últimas décadas e ainda o é nos dias atuais. Muito embora o seu uso possa ser empregado ao mesmo tempo em que boas práticas de programação - como o uso de **Design Patterns** - são utilizadas, alguns problemas são inerentes à sua natureza. A Figura 3.1.1 mostra um exemplo típico.

Figura 3.1.1



A aplicação exibida na Figura 3.1.1 possui diferentes funcionalidades.

- Exibição de catálogo de produtos.
- Geração de relatórios estatísticos.
- Gerenciamento e realização de pagamentos.
- Serviço de autenticação.

Ela está relativamente bem organizada. Possivelmente foi implementada utilizando-se algum padrão composto como o **MVC** ou **MVVM**. Seus componentes de software internos são, possivelmente, altamente coesos e pouco acoplados. Entretanto, a visão geral do sistema mostra um problema que pode ser grave. Todas as funcionalidades fazer parte de uma coisa só: caso seja necessário fazer ajustes no serviço de pagamento, por exemplo, é necessário fazer uma nova implantação do sistema inteiro. Caso o serviço de autenticação deixe de funcionar, possivelmente a aplicação inteira ficará indisponível.

3.2 Arquitetura baseada em microsserviços Podemos, assim, definir o conceito de **Arquitetura baseada em microsserviços**.

Definição

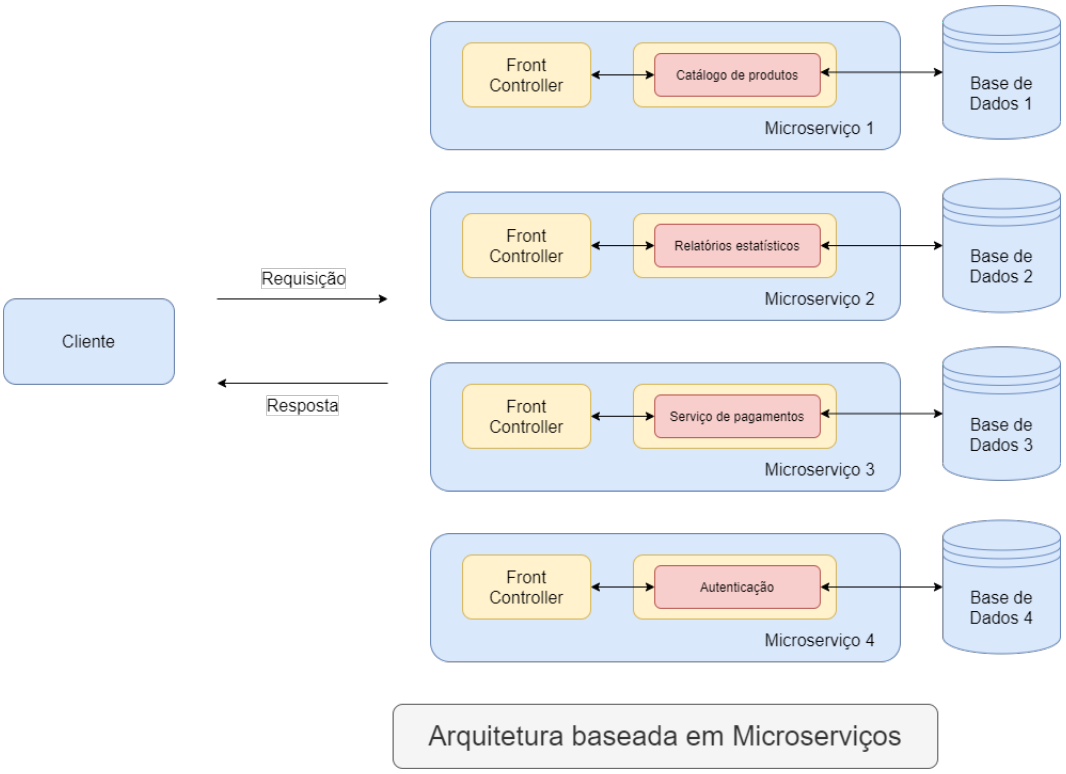
Uma **Arquitetura baseada em microsserviços** é constituída de pequenos serviços independentes, cada qual responsável por uma única funcionalidade do sistema. Os microsserviços se comunicam entre si por meio de uma interface bem definida. Quando um microsserviço fica indisponível, os demais continuam operando normalmente. Em geral, um microsserviço é desenvolvido e mantido por uma única equipe de desenvolvimento.

Os serviços de **Computação em Nuvem** estão intimamente relacionados ao uso de arquiteturas baseadas em microsserviços. Visite o Link 3.2.1 para conhecer a definição proposta pela **Amazon**.

Link 3.2.1
https://aws.amazon.com/pt/microservices/

A Figura 3.2.1 mostra a ideia geral de um sistema cuja implementação utiliza a Arquitetura baseada em microsserviços.

Figura 3.2.1

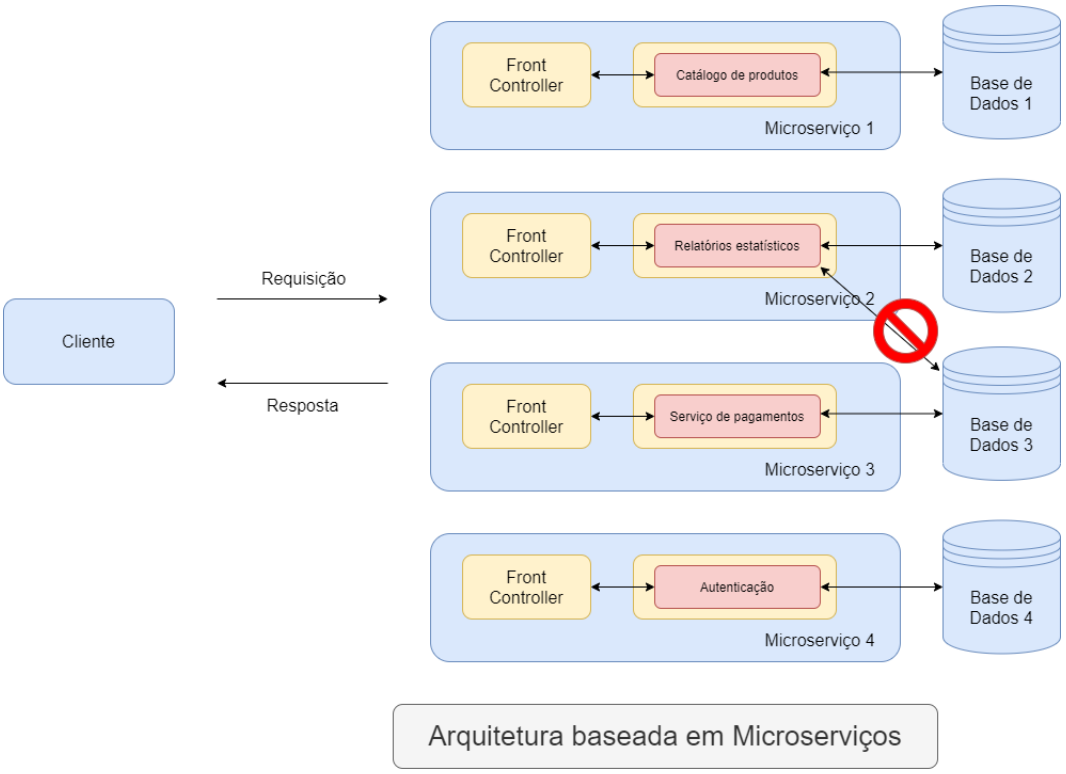


3.2.1 E os dados? Devido à independência entre os microsserviços sugerida pelas definições apresentadas, o uso de uma base de dados independente para cada um deles parece natural. Idealmente, **um microsserviço jamais acessa a base de dados de outro**. Isso ocorre pois **o esquema de cada base pode ser alterado a qualquer momento** e, além disso, cada microsserviço pode ter um **tipo de esquema (relacional ou NoSQL, por exemplo) mais apropriado** para a sua finalidade.

Entretanto, essa abordagem tem aspectos que podem comprometer a desejada simplicidade para o desenvolvimento do sistema. Considere, ainda, a proposta de arquitetura da Figura 3.2.2.

Professor Rodrigo Bossini
professorbossini@gmail.com

Figura 3.2.2



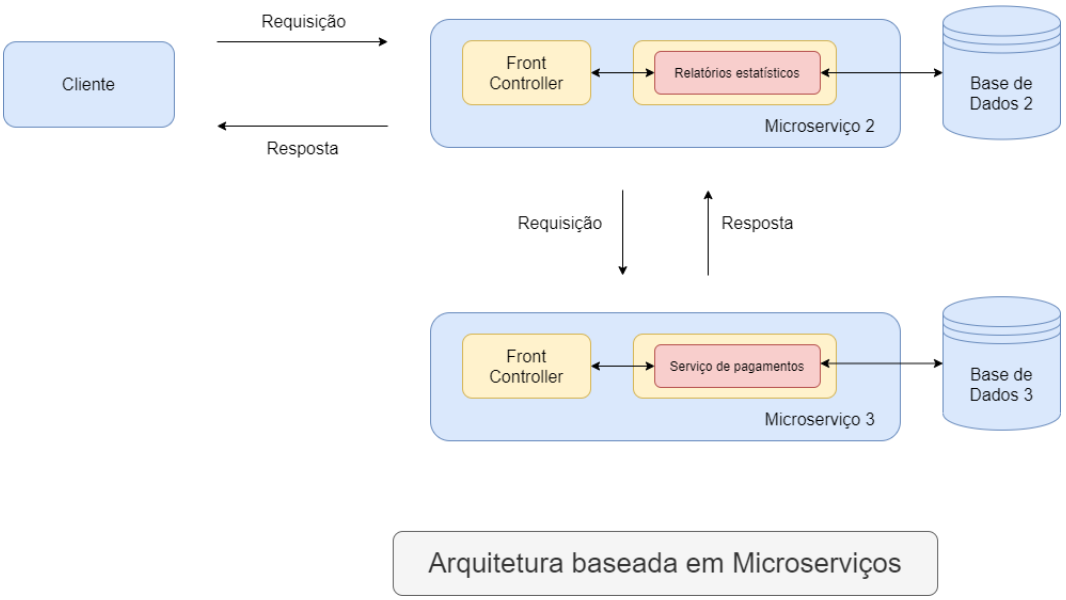
A sua funcionalidade **Relatórios Estatísticos** depende de dados existentes nas bases de dados pertencentes a outros microsserviços. Como essa funcionalidade poderia ser implementada sem o acesso direto a essas bases? Os microsserviços precisam se comunicar de alguma forma.

3.2.2 Comunicação entre microsserviços Para resolver esse tipo de problema, os microsserviços precisam se comunicar de alguma forma. Há dois tipos essenciais de comunicação entre microsserviços: o **síncrono** e o **assíncrono**.

3.2.3 Comunicação síncrona Na **comunicação síncrona**, um microsserviço realiza uma requisição direta a outro e fica no aguardo da resposta. Veja a Figura 3.2.3.

Professor Rodrigo Bossini
professorbossini@gmail.com

Figura 3.2.3

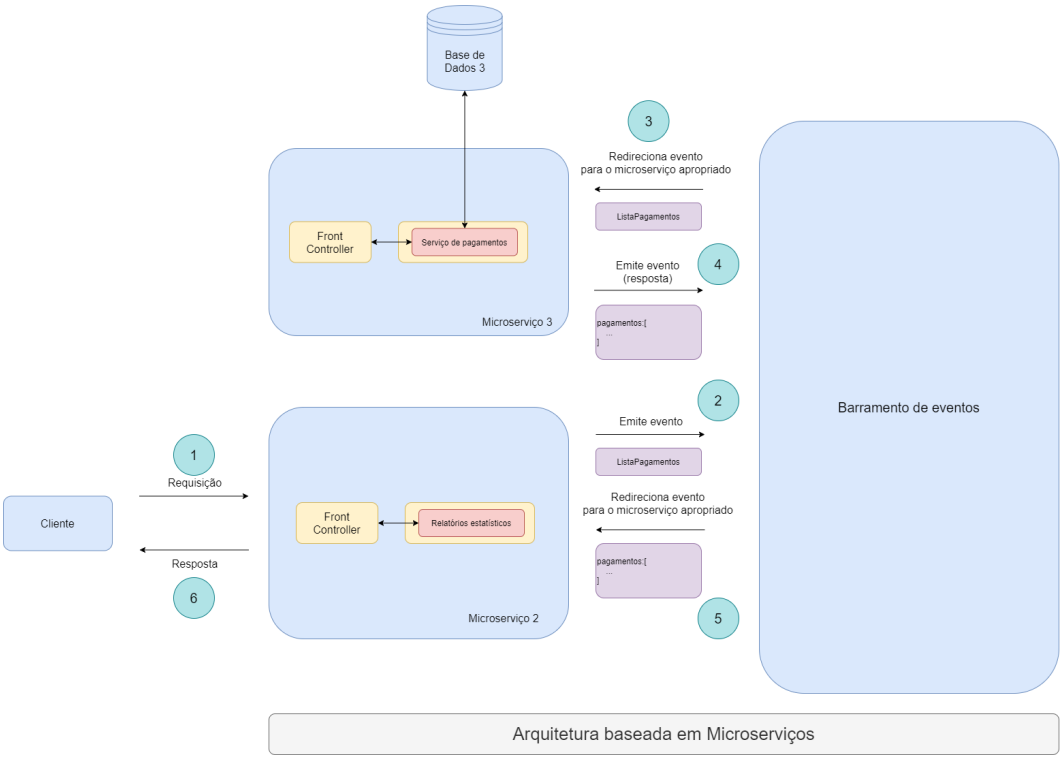


Arquitetura baseada em Microserviços

Em qualquer modelo de comunicação, há vantagens e desvantagens a serem consideradas. **A comunicação síncrona, por exemplo, traz como vantagem o acesso indireto a bases de dados**. Inclusive, pode ser o caso de o microsserviço em questão sequer precisar de uma. Por outro lado, é natural a **dependência entre os microsserviços**. Neste exemplo, caso o Serviço de Pagamentos fique indisponível, o microsserviço de Relatórios Estatísticos também deixará de funcionar.

3.2.4 Comunicação assíncrona - barramento de eventos A **comunicação assíncrona**, por sua vez, tem algumas formas de implementação. Uma delas é baseada em **eventos**. Seu funcionamento se dá em função de um **barramento de eventos**. Veja a Figura 3.2.4.

Figura 3.2.4



Neste modelo de comunicação assíncrona, cada microsserviço se conecta ao barramento de eventos[1]. Uma vez conectado ao barramento, um microsserviço pode

- emitir eventos, o que significa que eles são enviados ao barramento.
- receber eventos do barramento que são gerados por outros microsserviços.

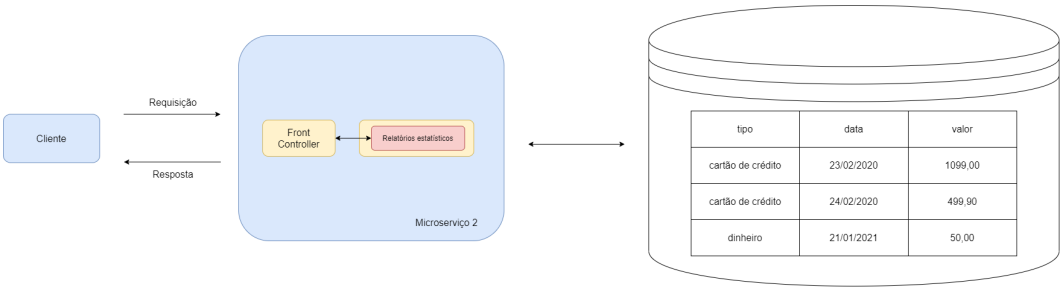
Este modelo perde em **simplicidade** quando comparado ao modelo de comunicação síncrona. Há também a **dependência entre serviços**.Pelo fato de a comunicação ser assíncrona, o microsserviço de origem pode se ocupar com **outras tarefas enquanto aguarda uma eventual resposta do barramento de eventos**.

3.2.5 Comunicação assíncrona - base de dados em função das demais

Uma outra forma de comunicação assíncrona se baseia na construção de uma base de dados em função de outras. Veja a Figura 3.2.5. Ela pode ser uma espécie de **view** contendo somente as partes de interesse.

[1]Note que o barramento de eventos poderia ser um ponto de falha centralizado do sistema. Ele é, idealmente, implantado em uma plataforma que lida com esse tipo de problema automaticamente.

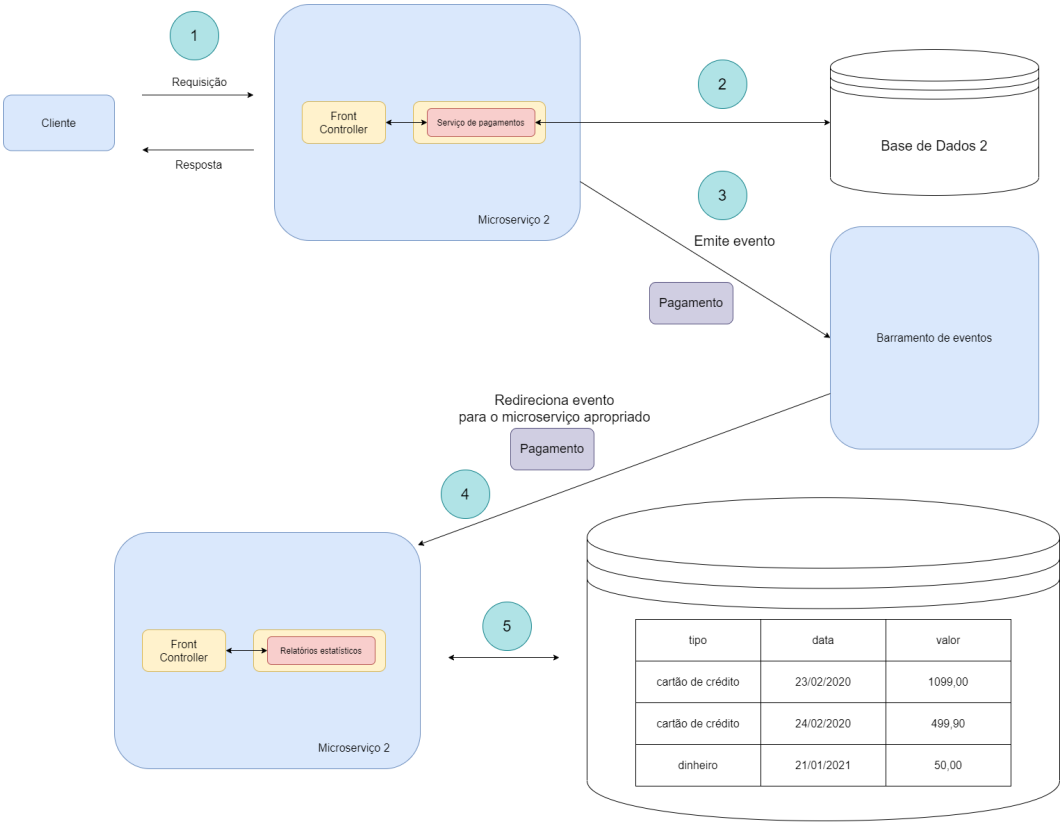
Figura 3.2.5



Arquitetura baseada em Microserviços

A questão óbvia a ser respondida é como essa base pode ser criada sem que um microsserviço acesse diretamente a base de dados de outro e sem estabelecer dependências diretas entre eles. A ideia geral é ilustrada na Figura 3.2.6

Figura 3.2.6



Arquitetura baseada em Microserviços

Quando o microsserviço de Pagamentos recebe uma nova requisição, ele armazena os dados de um novo pagamento em sua base, garantindo o seu funciona-

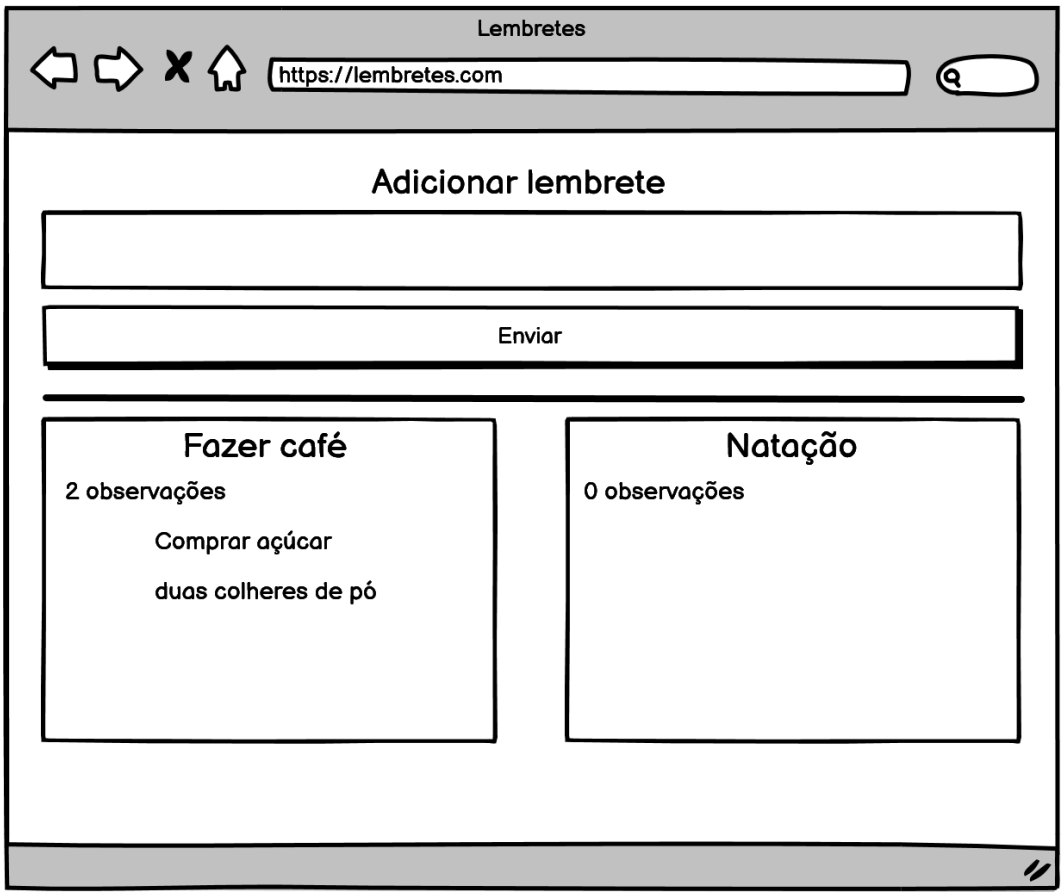
Professor Rodrigo Bossini
professorbossini@gmail.com

mento. Além disso, ele emite um evento que é direcionado a um barramento de eventos. O barramento de eventos se encarrega de fazer uma espécie de envio **broadcast** para que microsserviços interessados naquele evento sejam notificados. Neste exemplo, o microsserviço de Relatórios Estatísticos recebe o evento e atualiza a sua base. Note que a requisição é assíncrona e, **caso o microsserviço de Relatórios Estatísticos esteja temporariamente indisponível, o microsserviço de Pagamentos não deixa de funcionar**. E o oposto também é verdadeiro: **caso o microsserviço de Pagamentos esteja indisponível, o microsserviço de Relatórios Estatísticos pode operar utilizando a base de dados que possuir no momento**.

3.3 Construindo uma aplicação do zero com microsserviços Nesta seção, iremos implementar uma aplicação utilizando a Arquitetura baseada em microsserviços. A ideia é implementar os diversos recursos necessários um a um, ao invés de utilizar pacotes já prontos para isso. Nas próximas seções, lidaremos também com o seu uso.

3.3.1 Visão geral A Figura 3.3.1 mostra a tela principal da aplicação. Ela permite que o usuário armazene os seus **lembretes** e, eventualmente, adicione um número arbitrário de **observações** a eles.

Figura 3.3.1



3.3.2 Quais microsserviços implementar? Uma decisão a ser tomada diz respeito a quais e quantos microsserviços implementar. Para esta aplicação, iremos implementar **um microsserviço para cada tipo de objeto que ela manipula**. Assim, teremos um microsserviço para os lembretes e outro para as observações. Eles realizarão as seguintes tarefas.

- microsserviço de lembretes
    1. criar um lembrete
    2. listar os lembretes
- microsserviço de observações
    1. criar uma observação
    2. listar as observações de um lembrete

É importante observar que, mesmo para essas funcionalidades aparentemente simples, já existem questões relativamente complexas a serem resolvidas. As funcionalidades do microsserviço de observações dependem de dados de lembretes.

Assim, teremos de decidir qual forma de comunicação será empregada entre os microsserviços.

3.3.3 Implementando o microsserviço de lembretes Os microsserviços serão implementados utilizando-se o NodeJS[4]. Caso não tenha instalado, visite o Capítulo 4 para mais informações.

3.3.4 Workspace Comece criando um novo diretório para abrigar os arquivos dos projetos.

3.3.5 O microsserviço de lembretes Cada microsserviço será implementado como um projeto NodeJS independente. Crie uma pasta chamada **lembretes** em seu workspace e use

```
npm init -y
```

para criar um novo projeto. A opção **-y** indica que deseja-se utilizar valores padrão para cada item que caracteriza o projeto, como nome, versão etc.

3.3.6 Pacotes Os pacotes que utilizaremos, a princípio, são

- **express** - um framework web para o NodeJS que adiciona níveis de abstração para, entre outras coisas, a manipulação de requisições *HTTP*.
- **cors** - **CORS** significa **Cross-Origin Resource Sharing**. Trata-se de um mecanismo utilizado para especificar como cliente e servidor podem compartilhar recursos, em particular para o caso em que tiverem domínios diferentes. O pacote cors disponibiliza uma API que simplifica esse tipo de especificação.
- **axios** - um pacote que simplifica a realização de requisições HTTP assíncronas usando Ajax.
- **nodemon** - seu nome vem de **Node Monitor**. É natural a necessidade de reinicializar o servidor em tempo de desenvolvimento. Em geral, isso é necessário para que novas atualizações realizadas possam ser testadas. O nodemon é um pacote que monitora a execução de um servidor NodeJS e que o reinicializa automaticamente quando detecta que arquivos de extensões especificadas são alterados.

Eles podem ser instalados com

```
npm install express cors axios nodemon
```

Certifique-se de executar esse comando utilizando um terminal vinculado ao diretório em que se encontra o seu projeto NodeJS.

3.3.7 O microsserviço de observações Repita os passos para criar o projeto referente às observações. Crie uma pasta chamada **observacoes** em seu workspace - cuidado para não criá-la dentro do diretório do microsserviço de lembretes - e execute

npm init -y

e

npm install express cors axios nodemon

logo a seguir.

3.3.8 Requisições e métodos HTTP do microsserviço de lembretes Lembre-se que o protocolo HTTP, cuja RFC principal pode ser encontrada no Link 3.3.1, possui métodos com finalidades específicas.

Link 3.3.1
https://tools.ietf.org/html/rfc2616

Utilizaremos as seguintes especificações.

- Método HTTP: **PUT**. Padrão de acesso: **/lembretes**. Corpo: **{texto: string}**. Atividade: **Criar um novo lembrete**.
- Método HTTP: **GET**. Padrão de acesso: **/lembretes**. Corpo: **vazio**. Atividade: **Obter a lista de lembretes**.

3.3.9 Código inicial do microsserviço de lembretes Comece criando um arquivo chamado **index.js** na pasta **lembretes**. O Bloco de Código 3.3.1 mostra a implementação inicial do servidor com as duas rotas propostas.

Bloco de Código 3.3.1

```
1  const express = require ('express');
2  const app = express();
3  app.get ('/lembretes', (req, res) => {
4
5  });
6  app.put ('/lembretes', (req, res) => {
7
8  });
9
10 app.listen(4000, () => {
11     console.log('Lembretes. Porta 4000');
12 });
```


Professor Rodrigo Bossini
professorbossini@gmail.com

3.3.10 Base inicialmente volátil para o microsserviço de lembretes Inicialmente não nos preocuparemos com a implementação de uma base de dados propriamente dita. Os dados serão todas armazenados em uma coleção em memória volátil. Faça a sua definição como mostra o Bloco de Código 3.3.2.

Bloco de Código 3.3.2

```
1  const express = require ('express');
2  const app = express();
3  const lembretes = {};
4  app.get ('/lembretes', (req, res) => {
5
6  });
7  app.put ('/lembretes', (req, res) => {
8
9  });
10
11 app.listen(4000, () => {
12     console.log('Lembretes. Porta 4000');
13 });
```

3.3.11 Requisição GET: Devolvendo a coleção de lembretes A implementação do método GET é muito simples: basta devolver a coleção inteira de lembretes. Veja a sua implementação no Bloco de Código 3.3.3.

Bloco de Código 3.3.3

```
1  . . .
2  app.get('/lembretes', (req, res) => {
3      res.send(lembretes);
4  });
5  . . .
```

3.3.12 Requisição PUT: Geração de id e criação de lembrete Quando um lembrete for inserido, ele será associado a um número de identificação para que, no futuro, seja possível associá-lo a suas observações e diferenciá-lo dos demais. A princípio, nosso id será um simples contador. Quando a requisição é recebida, precisamos extrair o campo **texto** para construir o objeto a ser armazenado. Para tal, vamos utilizar o pacote **body-parser**. Ele devolverá um **middleware** que irá adicionar um campo chamado **body** à requisição, simplificando a extração do conteúdo enviado pelo cliente. Veja o Bloco de Código 3.3.4.

Bloco de Código 3.3.4

```
. . .
const bodyParser = require('body-parser');
const app = express();
app.use(bodyParser.json());
lembretes = {};
contador = 0;
. . .
app.put('/lembretes', (req, res) => {
    contador++;
    const { texto } = req.body;
    lembretes[contador] = {
        contador, texto
    }
    res.status(201).send(lembretes[contador]);
});
```

3.3.13 Executando o servidor com nodemon Abra o arquivo *package.json* do projeto e encontre a chave **scripts**. Ajuste-a como no Bloco de Código 3.3.5.

Bloco de Código 3.3.5

```
{
  "name": "lembretes",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "axios": "^0.21.1",
    "cors": "^2.8.5",
    "express": "^4.17.1",
    "nodemon": "^2.0.7"
  }
}
```

Em um terminal vinculado ao diretório em que se encontra o projeto, use

Professor Rodrigo Bossini
professorbossini@gmail.com

npm start

para colocar o servidor em execução.

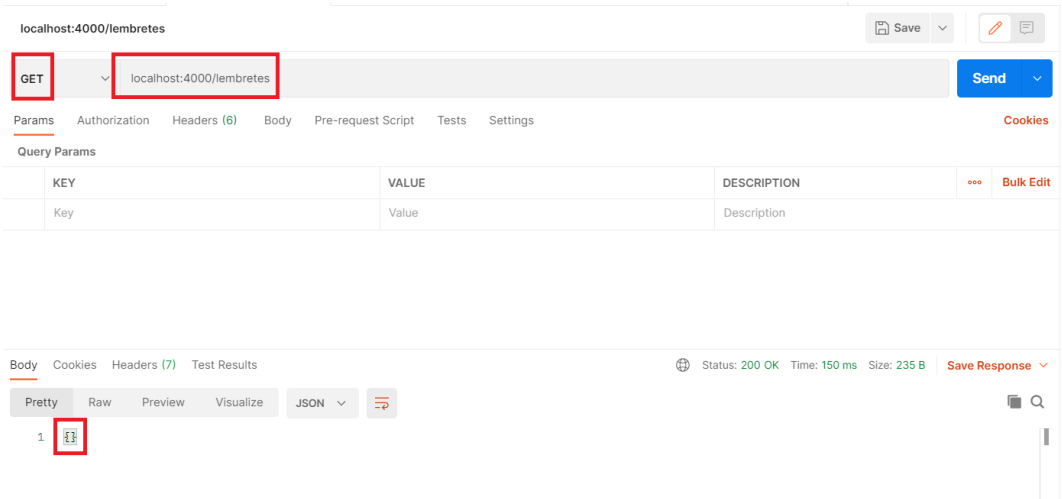
3.3.14 Testes com Postman O **Postman**[2] é um cliente HTTP. Passaremos a utilizá-lo para testar nossas APIs. Para testar o endpoint de obtenção da lista de lembretes, use os seguintes valores no Postman.

- Método: **GET**.
- Endereço: **localhost:4000/lembretes**

Nota. Ao abrir o Postman, pode ser necessário clicar em **Create New Request** a fim de visualizar a tela exibida pela Figura 3.3.2. Note que ele possui diversos recursos como a criação de coleções de requisições, ambientes e o uso do **Scratch Pad** e de **Workspaces**.

Veja a Figura 3.3.2. O resultado obtido deve ser um objeto JSON vazio, afinal, ainda não fizemos nenhuma inserção.

Figura 3.3.2



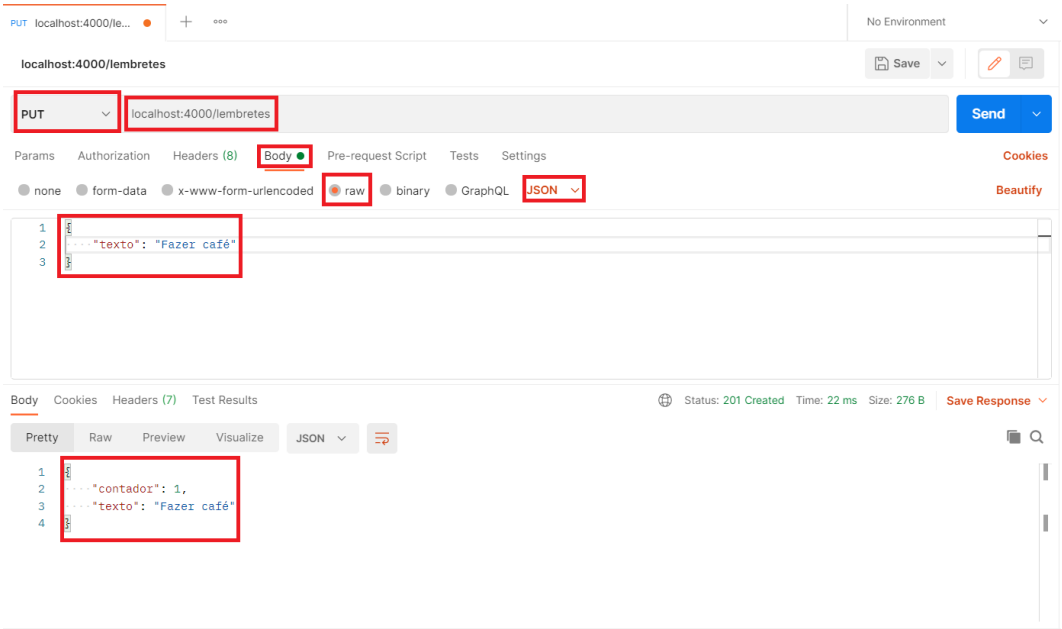
[2] https://www.postman.com/

Para testar o endpoint de inserção de novo lembrete, use os seguintes valores no Postman.

- Método: **PUT**.
- Endereço: **localhost:4000/lembretes**
- Body: **{"texto": "Fazer café"}**. Para especificar esse valor, clique *Body » raw » No menu de seleção (por padrão mostra "Text") escolha JSON*

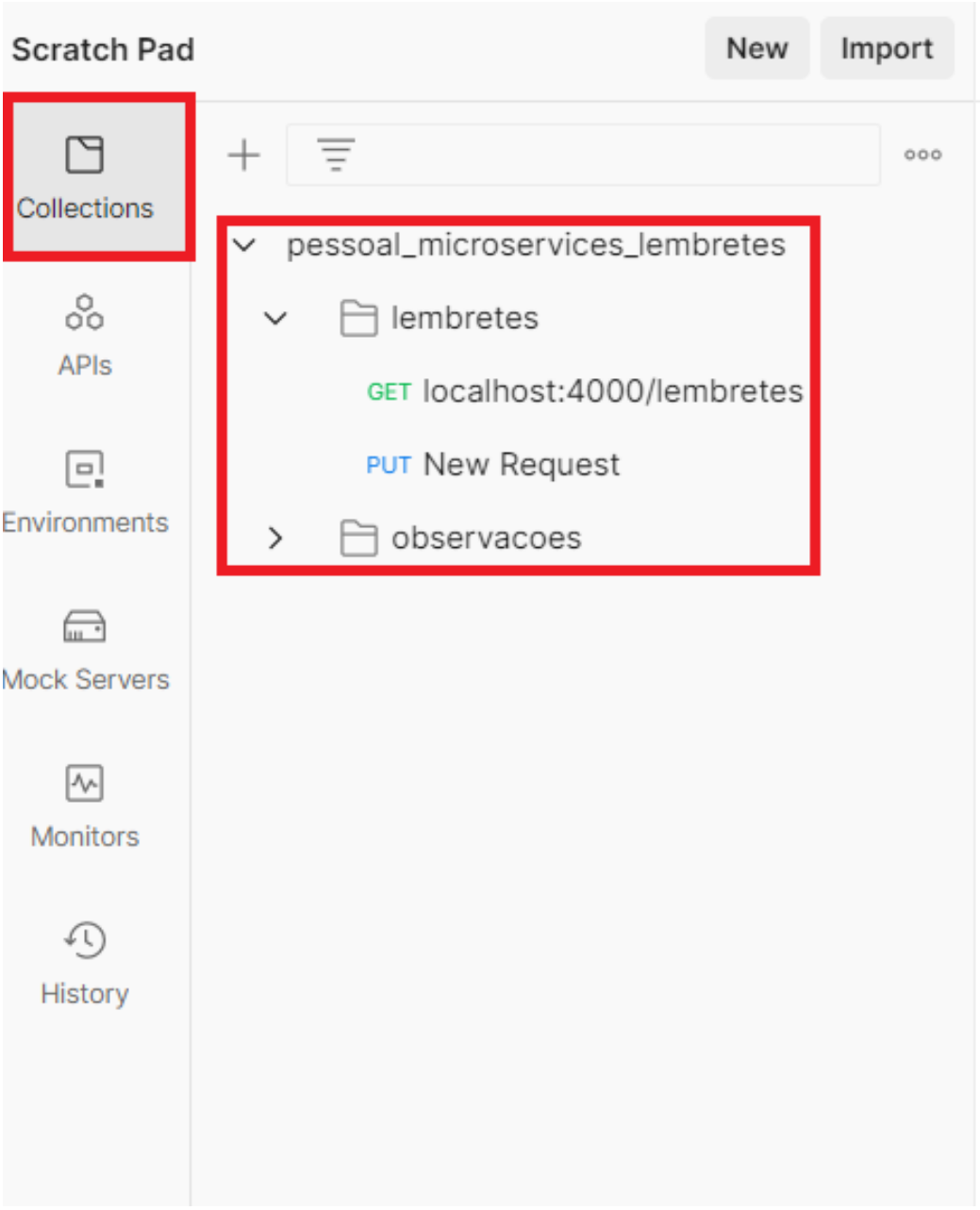
Veja a Figura 3.3.3.

Figura 3.3.3



3.3.15 Organizando as requisições em uma coleção Pode ser uma boa ideia fazer o uso do recurso de **Coleções de requisições** do Postman. Ele é interessante pois nos permite agrupar requisições relacionadas e mantê-las armazenadas para uso futuro. Para criar uma nova coleção de requisições no Postman, clique *Collections*. A seguir, clique *Create New Collection*. Escolha um nome para a sua coleção, idealmente algo que te ajude a lembrar a que estão associadas as requisições que serão agora agrupadas. A seguir, crie duas pastas: uma para as requisições referentes aos lembretes e outra para as requisições referentes às observações. Para criar as pastas, basta clicar com o direito sobre o nome da coleção. Crie, a seguir, as requisições relacionadas a lembretes em sua respectiva pasta. Veja a Figura 3.3.4.

Figura 3.3.4



3.3.16 Implementando o microsserviço de observações A implementação do microsserviço de observações é semelhante àquela feita para o microsserviço de lembretes. Entretanto, há um ponto importante a ser considerado: cada observação está associada a um lembrete. Note que isso poderá implicar na necessidade

da comunicação entre os microsserviços. Suas especificações são as seguintes.

- Método HTTP: **PUT**. Padrão de acesso: **/lembretes/:id/observacoes**. Corpo: **{conteudo: string}**. Atividade: **Criar uma nova observação associada ao lembrete cujo id se encontra especificado no padrão de acesso**.
- Método HTTP: **GET**. Padrão de acesso: **/lembretes/:id/observacoes**. Corpo: **vazio**. Atividade: **Obter a lista de observações associadas ao lembrete cujo id se encontra especificado no padrão de acesso**.

3.3.17 Código inicial do microsserviço de observações Comece criando um arquivo chamado **index.js** na pasta **observacoes**. O código inicial é semelhante àquele visto na criação do microsserviço de lembretes. Note, entretanto, que os padrões de acesso são diferentes. Também é necessário utilizar uma porta diferente. Assim os dois microsserviços poderão ser mantidos em execução simultaneamente. Veja o Bloco de Código 3.3.6.

Bloco de Código 3.3.6

```
1  const express = require ('express');
2  const bodyParser = require('body-parser');
3
4  const app = express();
5  app.use(bodyParser.json());
6
7  //:id é um placeholder
8  //exemplo: /lembretes/123456/observacoes
9  app.put('/lembretes/:id/observacoes', (req, res) => {
10
11 });
12
13 app.get('/lembretes/:id/observacoes', (req, res) => {
14
15 });
16
17 app.listen(5000, (() => {
18     console.log('Lembretes. Porta 5000');
19 }));
```

3.3.18 Base inicialmente volátil para o microsserviço de observações A definição da base de dados que armazena a coleção de observações é exibida no Bloco de Código 3.3.7. Ela é um objeto JSON em que **cada chave é o id de um lembrete e seu valor associado é a coleção de observações associadas àquele lembrete**.

Bloco de Código 3.3.7

```
1   . . .
2   const app = express();
3   app.use(bodyParser.json());
4
5   const observacoesPorLembreteId = {};
6
7   //:id é um placeholder
8   //exemplo: /lembretes/123456/observacoes
9   app.put('/lembretes/:id/observacoes', (req, res) => {
10
11  });
12  . . .
```

3.3.19 Gerando códigos UUID Na implementação da base de dados do serviço de lembretes, utilizamos um simples contador para representar um código único utilizado para diferenciar um lembrete dos demais. Podemos fazer uso de técnicas mais sofistificadas utilizando, por exemplo, o pacote **uuid**. Ele implementa a especificação UUID dada pela **RFC 4122** que pode ser visitada por meio do Link 3.3.2.

Link 3.3.2
https://aws.amazon.com/pt/microservices/

Para fazer a sua instalação, use

npm install uuid

Caso deseje, passe a utilizá-lo também no microsserviços de lembretes. Note que, por serem serviços independentes, nada impede que utilizem estratégias diferentes.

Professor Rodrigo Bossini
professorbossini@gmail.com

3.3.20 Requisição PUT: Inserindo uma nova observação O algoritmo para inserção de nova observação é o seguinte.

- Gerar um novo identificador para a observação a ser inserida.
- Extrair, do corpo da requisição, o texto da observação.
- Verificar se o id de lembrete existente na URL já existe na base e está associado a uma coleção. Em caso positivo, prosseguir utilizando a coleção existente. Caso contrário, criar uma nova coleção.
- Adicionar a nova observação à coleção de observações recém obtida/criada.
- Fazer com que o identificador do lembrete existente na URL esteja associado a essa nova coleção alterada, na base de observações por id de lembrete.
- Devolver uma resposta ao usuário envolvendo o código de status HTTP e algum objeto de interesse, possivelmente a observação inserida ou, ainda, a coleção inteira de observações.

Veja a sua implementação no Bloco de Código 3.3.8.

Bloco de Código 3.3.8

```
1   . . .
2   const { v4: uuidv4 } = require('uuid');
3   . . .
4   //:id é um placeholder
5   //exemplo: /lembretes/123456/observacoes
6   app.put('/lembretes/:id/observacoes', (req, res) => {
7     const idObs = uuidv4();
8     const { texto } = req.body;
9     //req.params dá acesso à lista de parâmetros da URL
10    const observacoesDoLembrete =
11      observacoesPorLembreteId[req.params.id] || [];
12    observacoesDoLembrete.push({ id: idObs, texto });
13    observacoesPorLembreteId[req.params.id] =
14      observacoesDoLembrete;
15    res.status(201).send(observacoesDoLembrete);
16  });
```

Para testar, abra o arquivo *package.json* do projeto observações e adicione um script que utiliza o `nodemon`, como feito para o projeto de lembretes. Veja o Bloco de Código 3.3.9.

Bloco de Código 3.3.9

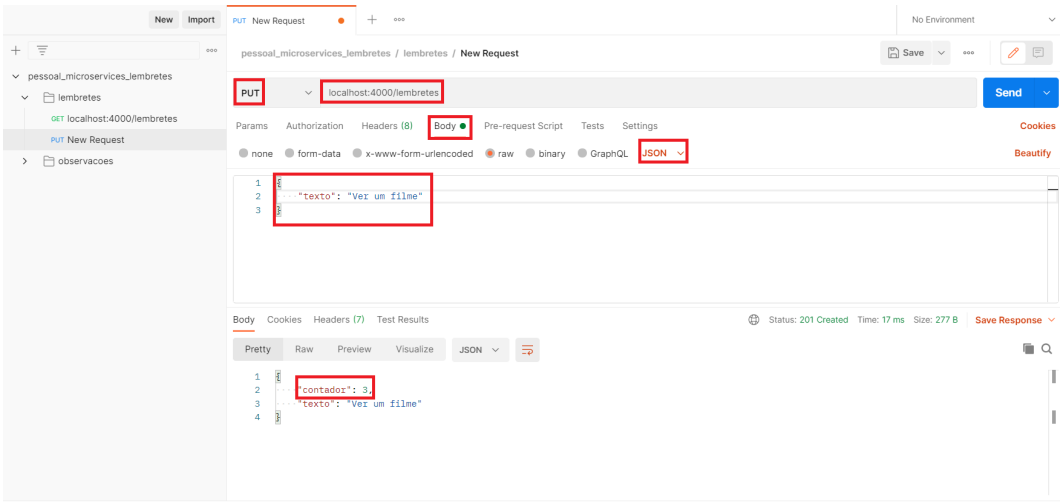
```
    {
  "name": "observacoes",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "axios": "^0.21.1",
    "cors": "^2.8.5",
    "express": "^4.17.1",
    "nodemon": "^2.0.7",
    "uuid": "^8.3.2"
  }
}

```

Utilize

npm start

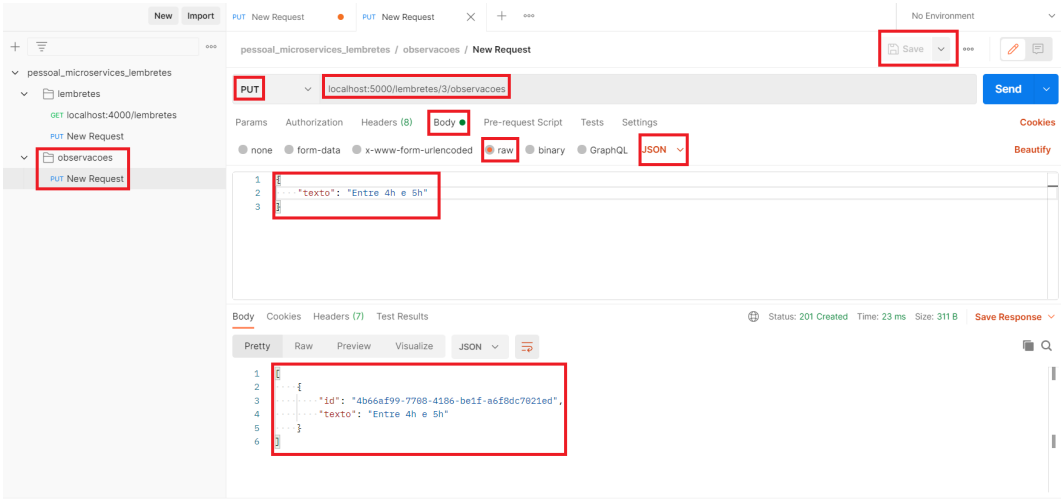
para colocar o microsserviço em execução. A seguir, abra o Postman e envie uma requisição ao microsserviço de lembretes para fazer a inserção de um novo e anotar o seu identificador gerado. Veja a Figura 3.3.5.

Figura 3.3.5



Anote o identificador gerado para o lembrete recém inserido e faça uma requisição de inserção ao microsserviço de observações, como mostra a Figura 3.3.6. Como feito até então, crie a requisição na sub-pasta anteriormente criada para agrupar requisições feitas ao microsserviço de observações. Clique `Save` para salvar os dados da requisição.

Figura 3.3.6



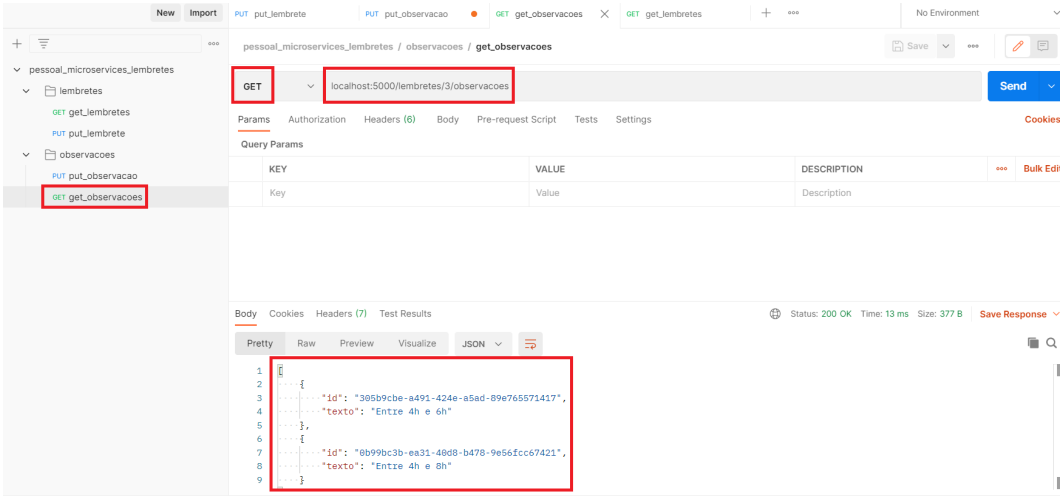
3.3.21 Requisição GET: Devolvendo a lista de observações de um lembrete A implementação do endpoint de obtenção de observações de um lembrete especificado é um tanto simples. Ela extrai o identificador de lembrete existente na URL e acessa a base de dados. Caso uma coleção seja encontrada na base, ela é devolvida. Caso contrário, o microsserviço devolve uma coleção vazia. Veja o Bloco de Código 3.3.10.

Bloco de Código 3.3.10

```
1  app.get('/lembretes/:id/observacoes', (req, res) => {
2    res.send(observacoesPorLembreteId[req.params.id] || []);
3  });
```

No Postman, crie uma nova requisição para testar a nova implementação, como na Figura 3.3.7.

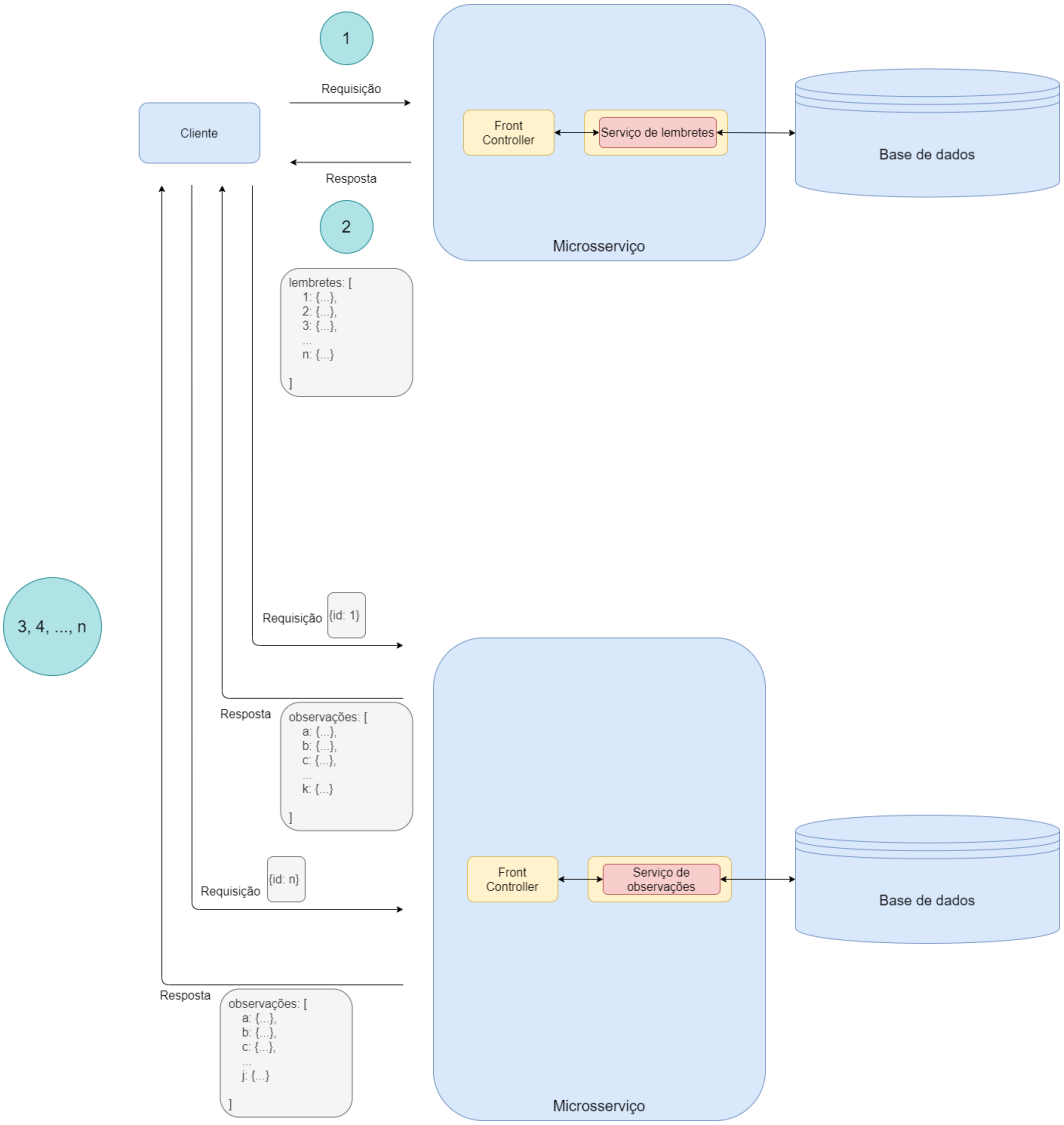
Figura 3.3.7



3.3.22 Busca da coleção de lembretes incluindo observações: $n+1$ requisições feitas pelo cliente Uma aplicação cliente, como aquela exibida pela Figura 3.3.1, pode estar interessada em obter a coleção de lembretes incluindo a coleção de observações referente a cada um deles. Devido à arquitetura que estamos empregando no Back End, podem ser necessárias muitas requisições para atender essa necessidade, como mostra a Figura 3.3.8. Uma primeira requisição é feita para a obtenção da coleção de lembretes. A seguir, n requisições são feitas para a obtenção de cada uma das coleções de observações.

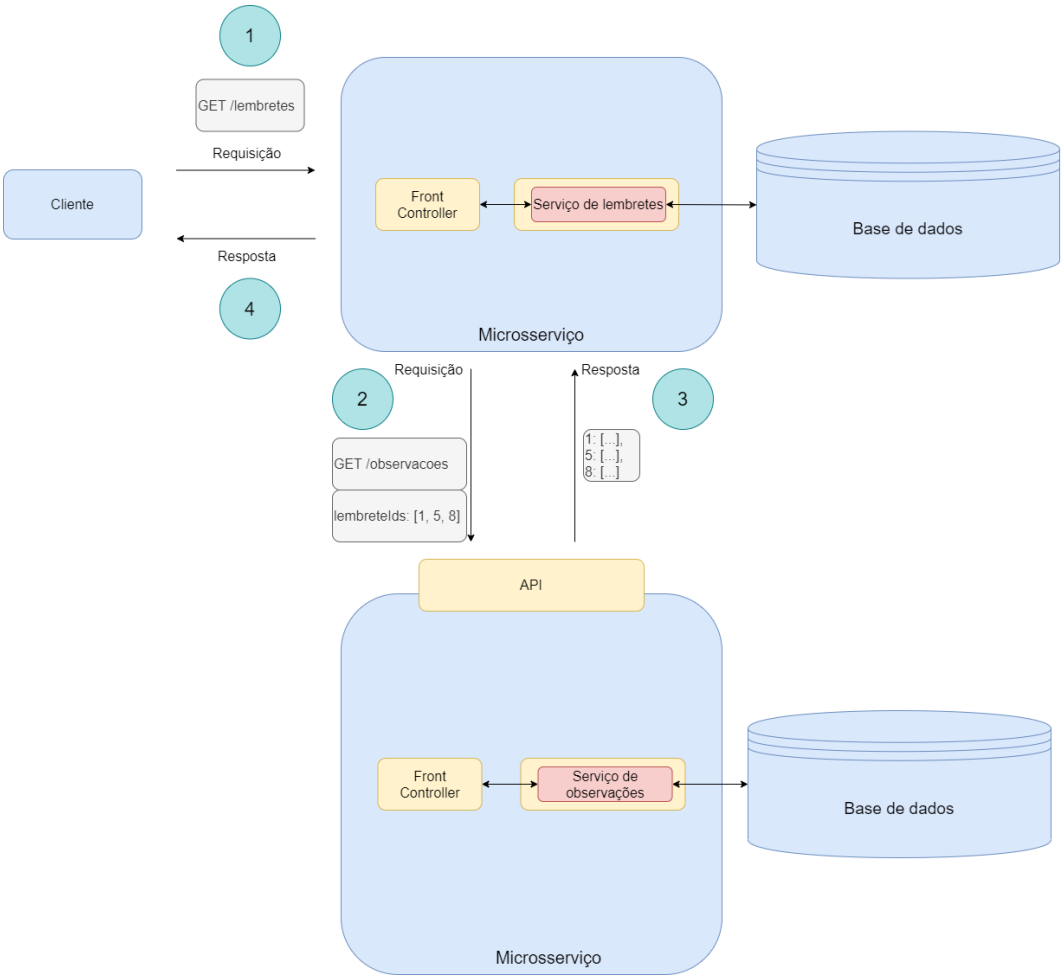
Professor Rodrigo Bossini
professorbossini@gmail.com

Figura 3.3.8



3.3.23 Busca da coleção de lembretes incluindo observações: comunicação síncrona Pode ser interessante tornar transparente para o cliente esse número de requisições, simplificando a sua implementação. A ideia é aplicar técnicas de comunicação entre microsserviços para fazê-lo. Uma dela é a **Comunicação Síncrona**, como ilustra a Figura 3.3.9

Figura 3.3.9

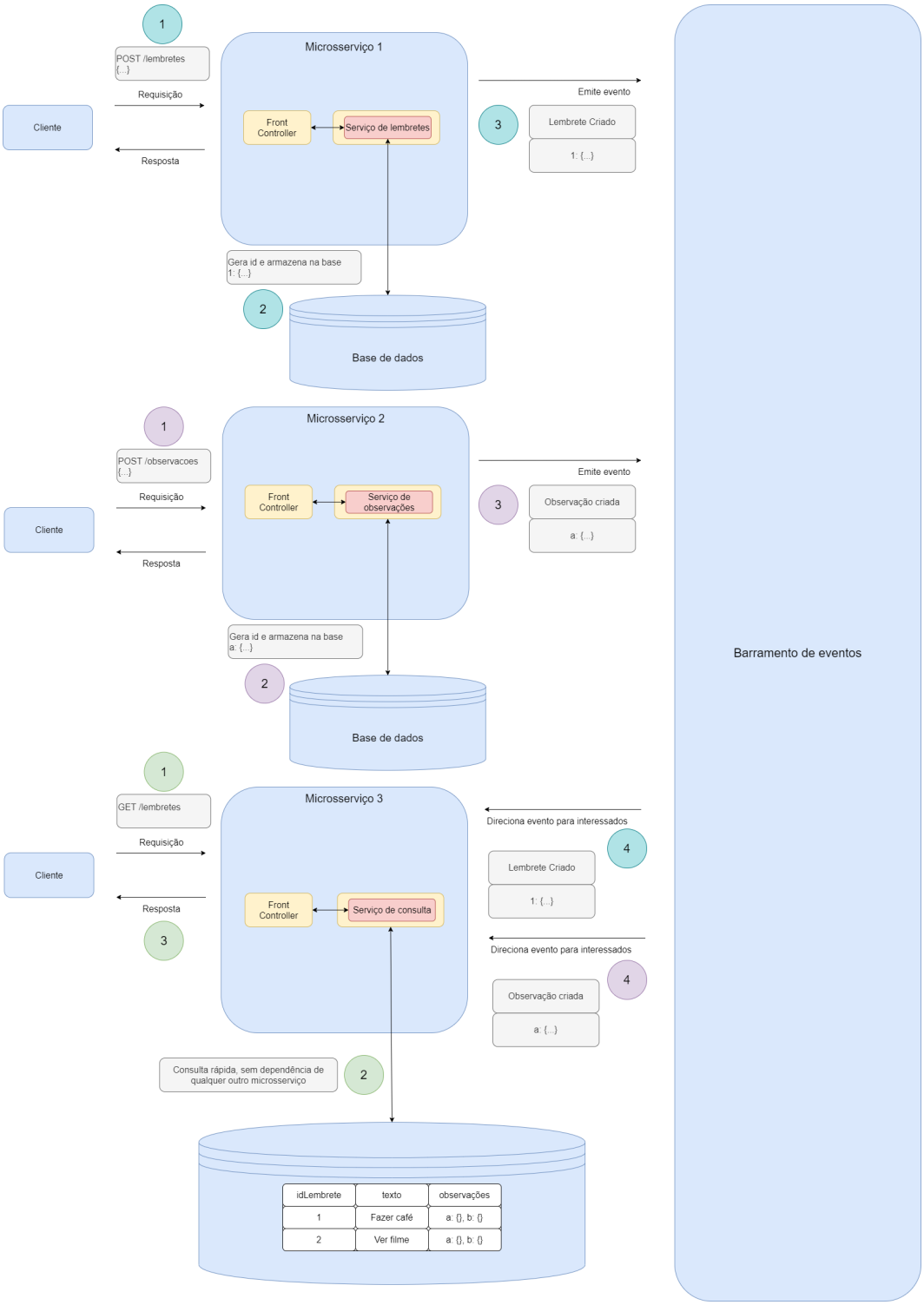


Como discutimos, esse é um modelo de simples entendimento. Entretanto, ele apresenta dependência entre microsserviços e pode dar origem a cadeias de requisições a diferentes microsserviços, o que pode comprometer o desempenho do microsserviço de origem.

3.3.24 Busca da coleção de lembretes incluindo observações: comunicação assíncrona Outra possibilidade envolve o uso da comunicação assíncrona entre os microsserviços. Ela traz vantagens como a independência entre microsserviços e a possibilidade de melhorias de desempenho. Há, por outro lado, a necessidade de se cuidar da redundância de dados. O modelo ilustrado na Figura 3.3.10 traz a proposta de uso de um microsserviço de consulta.

Professor Rodrigo Bossini
professorbossini@gmail.com

Figura 3.3.10



3.3.25 Uma implementação manual de um barramento de eventos Há diferentes implementações de barramentos de eventos - também chamadas de filas

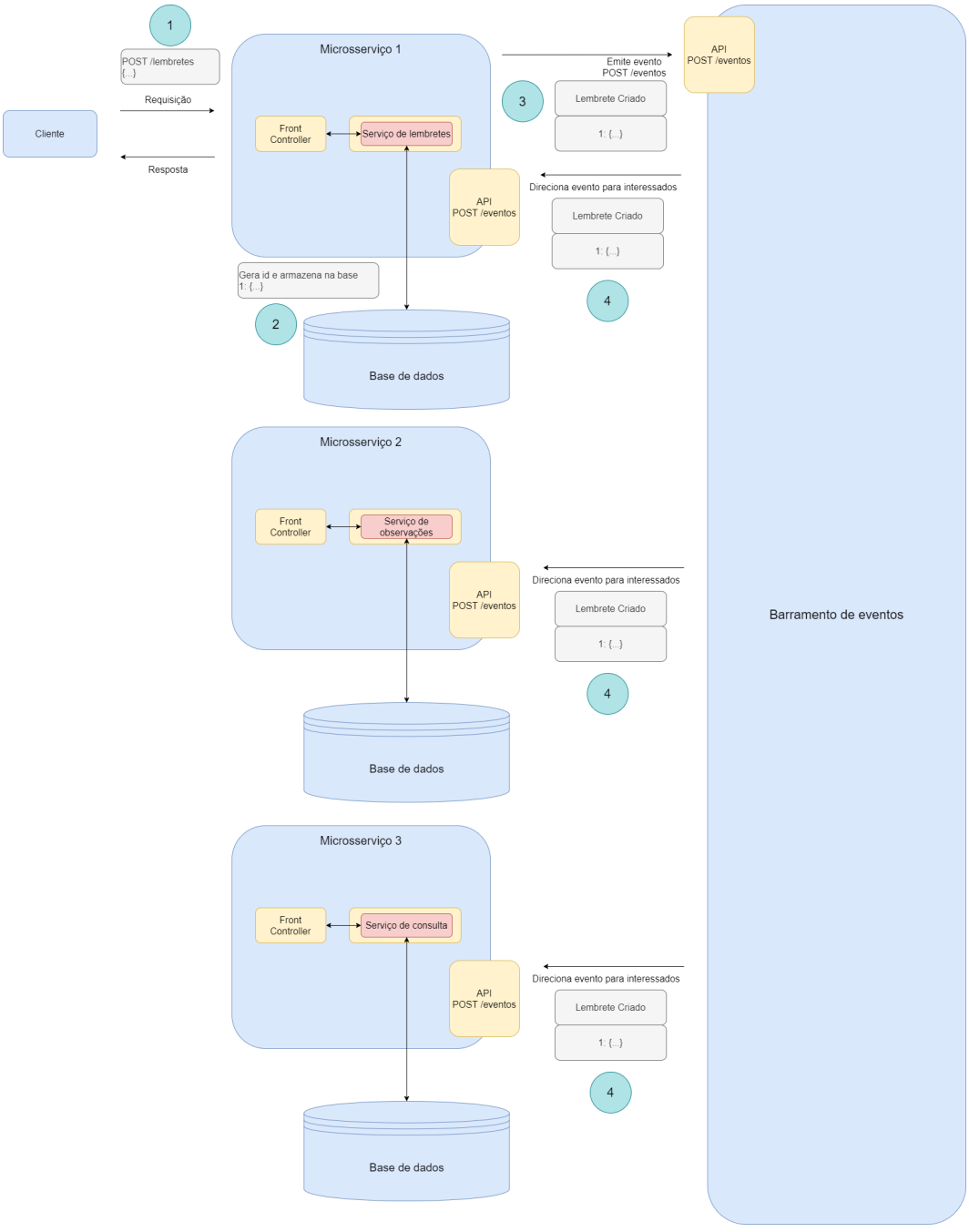
Professor Rodrigo Bossini
professorbossini@gmail.com

de mensagens e outros - disponíveis. Alguns exemplos são os seguintes.

- Apache ActiveMQ
- RabbitMQ
- Apache Kafka
- Amazon MQ
- Google Cloud Pub/Sub

Nesta seção, iremos fazer a nossa própria implementação para fins de aprendizado. Aplicações profissionais certamente empregam soluções como as citadas. Elas são amplamente testadas e utilizadas por milhões de usuários, além de, em geral, serem implantadas em serviços de computação em nuvem de alta disponibilidade, tolerância a falhas etc. A Figura 3.3.11 ilustra o funcionamento básico de nossa implementação. O barramento de eventos - que também é um microsserviço - possui um endpoint que viabiliza a entrega de eventos. Cada serviço interessado em algum tipo de evento também disponibiliza um endpoint assim. Quando o barramento de eventos recebe um evento, ele faz uma espécie de envio **broadcast**.

Figura 3.3.11



3.3.26 Criando o novo projeto O barramento de eventos será um novo microsserviço, independente dos demais. Por isso, crie uma nova pasta chamada *barramento-de-eventos* em seu workspace e use

```
npm init -y
```

para criar o novo projeto. Abra um terminal vinculado à nova pasta e instale as dependências com

npm install express nodemon axios cors

Crie um novo arquivo chamado *index.js* na nova pasta e, como feito com os demais projetos, abra o arquivo *package.json* e especifique um novo script de execução como mostra o Bloco de Código 3.3.11.

Bloco de Código 3.3.11

```
{
  "name": "barramento-de-eventos",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "axios": "^0.21.1",
    "cors": "^2.8.5",
    "express": "^4.17.1",
    "nodemon": "^2.0.7"
  }
}
```

O Bloco de Código 3.3.12 mostra a implementação inicial do barramento de eventos. Ele possui um endpoint que se encarrega de direcionar o evento recebido aos demais microsserviços.

Bloco de Código 3.3.12

```
const express = require('express');
const bodyParser = require('body-parser');
//para enviar eventos para os demais microsserviços
const axios = require('axios');

const app = express();
app.use(bodyParser.json());

app.post('/eventos', (req, res) => {
    const evento = req.body;
    //envia o evento para o microsserviço de lembretes
    axios.post('http://localhost:4000/eventos', evento);
    //envia o evento para o microsserviço de observações
    axios.post('http://localhost:5000/eventos', evento);
    res.status(200);
});

app.listen(10000, () => {
    console.log('Barramento de eventos. Porta 10000.')
})
```

Use

npm start

para colocar o microsserviço em execução.

Capítulo 4

Instalação do NodeJS

O NodeJS é um ambiente que viabiliza, entre outras coisas, a execução de código Javascript do lado do servidor. A sua instalação acompanha o **Node Package Manager (npm)**, um gerenciador de pacotes por meio do qual podemos fazer a instalação de pacotes necessários para cada projeto. Neste material veremos algumas formas para sua instalação.

4.1 Instalação com o instalador regular do NodeJS Uma maneira bastante simples para fazer a instalação do NodeJS é por meio do download do seu instalador, disponível no site oficial[4]. A instalação do NodeJS já implica na instalação do npm.

4.2 Instalação usando um gerenciador de versões Há ainda a possibilidade de fazer a instalação do NodeJS por meio de um **Node Version Manager**(NVM), ou seja, um gerenciador de versões do NodeJS. Ele permite que tenhamos diversas versões do NodeJS instaladas e que façamos a alternância entre elas conforme desejado. Além disso, o funcionamento do NodeJS ocorre no diretório do usuário do sistema operacional, o que quer dizer que seu uso tende a evitar problemas de permissão para acesso a determinados diretórios. O instalador de um NVM pode ser obtido nos links 4.2.1 (Linux e Mac) e 4.2.2(Windows).

Link 4.2.1
https://github.com/nvm-sh/nvm

Link 4.2.2
https://github.com/coreybutler/nvm-windows

Uma vez instalado o NVM, para instalar o node, basta usar

---

nvm install versao-desejada

É interessante instalar a última versão LTS disponível na maior parte dos casos. As versões disponíveis para instalação podem ser listadas com

nvm ls-remote

no Linux e no Mac e com

nvm list available

no Windows. A última versão LTS disponível no momento em que esse documento foi escrito, era a 14.15.5. Para fazer a sua instalação, o comando é

nvm install 14.15.5

A seguir, para colocá-la em uso, use

nvm use 14.15.5

Você pode verificar se o NodeJS foi corretamente instalado com

node –version

ou com

node -v

Referências

[1] L. Bass, P. Clements e R. Kazman. *Software Architecture in Practice*. 3ª ed. Addison-Wesley Professional, 2012.

[2] A. Fox e D. Patterson. *Engineering Software as a Service - An Agile Approach Using Cloud Computing*. 1ª ed. Strawberry Canyon LLC, 2018.

[3] Martin Fowler. *Who Needs an Architect?* Disponível em: https://martinfowler.com/ieeeSoftware/whoNeedsArchitect.pdf. Acesso em fevereiro de 2021.

[4] Ryan Dahl. *Node.js*. Disponível em: https://nodejs.org/. Acesso em fevereiro de 2020.

.