WEB ACADEMY

Programação Avançada Back-end

Daniel Augusto Nunes da Silva

Apresentação

Ementa

- Mecanismos de segurança e controle de acesso: Autenticação por senha e JWT. Utilização de Data Transfer Objects (DTO). Validação de dados. Paginação e ordenação. Documentação de REST APIs. Acesso a APIs externas. Gerenciamento de cache.

Objetivos

- **Geral:** Capacitar os alunos para o desenvolvimento avançado de aplicações back-end, por meio da exploração de tecnologias, práticas e padrões essenciais, preparando-os para os desafios do ambiente profissional.

- **Específicos:**
  - Aplicar técnicas de autenticação e segurança, para garantir a proteção adequada dos recursos das aplicações.
  - Explorar técnicas avançadas para melhorar a eficiência e a qualidade das soluções back-end.
  - Abordar métodos de integração de aplicações back-end e estratégias de otimização de desempenho.

Conteúdo programático

Autenticação e Segurança

- Introdução ao Spring Security;
- Autenticação de usuários e autorização de acesso;
- Autenticação JWT.
- Segurança na comunicação via HTTP.

Práticas Avançadas

- Padrão de projeto DTO;
- DTO com Java Records;
- Mapeamento de DTOs;
- Paginação e ordenação de resultados;
- Validações no back-end;
- Documentação de REST APIs com Swagger.

Integração e otimização

- Acesso a APIs externas: RestTemplate, RestClient e WebClient;
- Implementação de cache: conceitos e estratégias.

Sites de referência

- Spring Boot Reference Documentation
  - https://docs.spring.io/spring-boot/3.5/index.html
- Spring Getting Started Guides
  - https://spring.io/guides#getting-started-guides
- Swagger Documentation
  - https://swagger.io/docs/
- Baeldung
  - https://www.baeldung.com/
- Engenharia de Software Moderna
  - https://engsoftmoderna.info/

Contato

https://github.com/danielnsilva

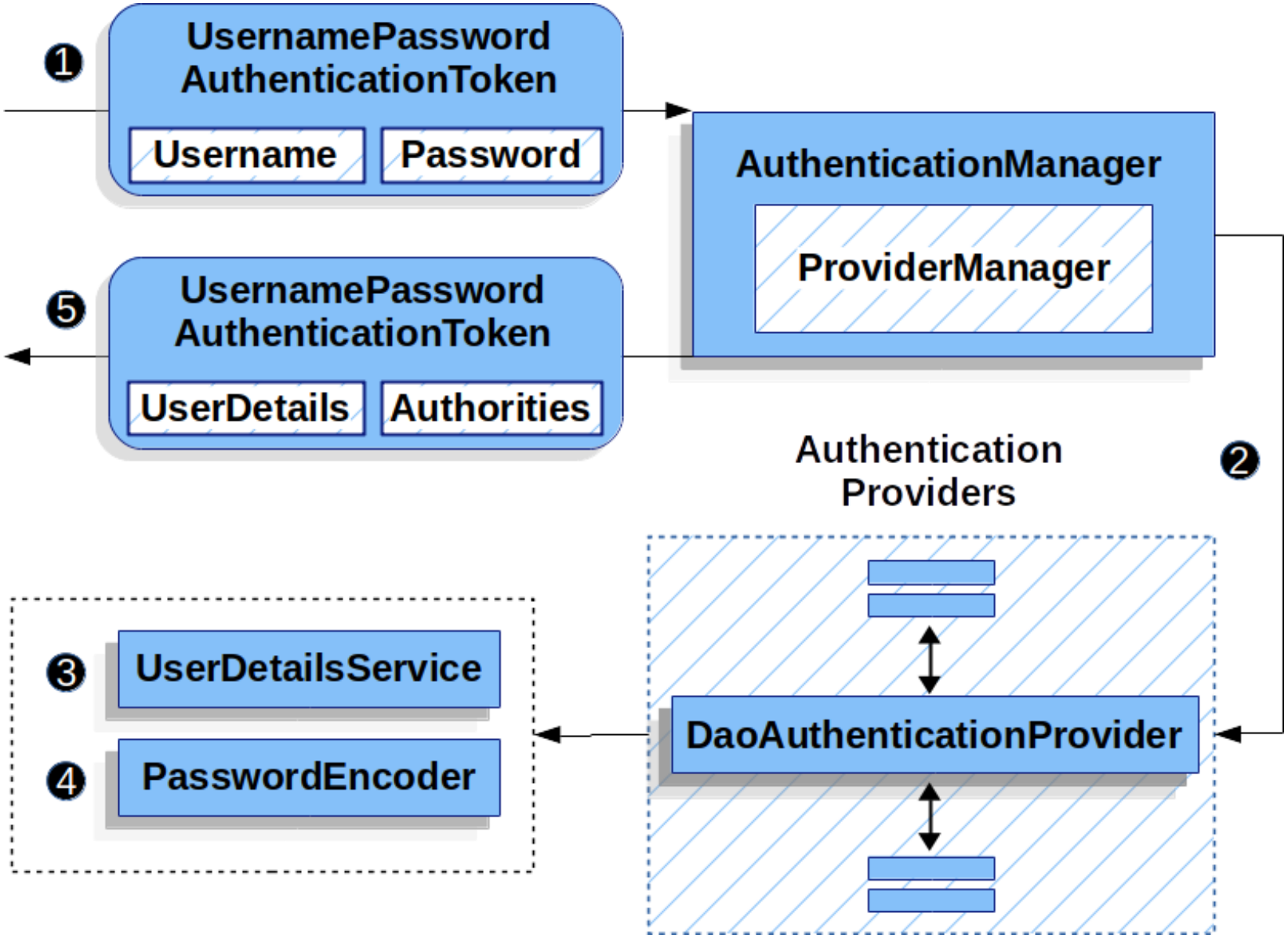
Autenticação e Segurança

Introdução ao Spring Security

- **Spring Security** fornece recursos para **autenticação**, **autorização de acesso** e proteção contra ataques comuns.
- **Autenticação**: verifica a identidade do usuário.
  - Exemplos: usuário/senha, certificado digital X.509, protocolo CAS para autenticação SSO (Single Sign-On).
- **Autorização**: verifica as permissões do usuário para realizar determinadas ações ou acessar recursos específicos.
- Possui suporte ao **OAuth2** (login via Google, Facebook, GitHub, etc.).

Autenticação de usuários

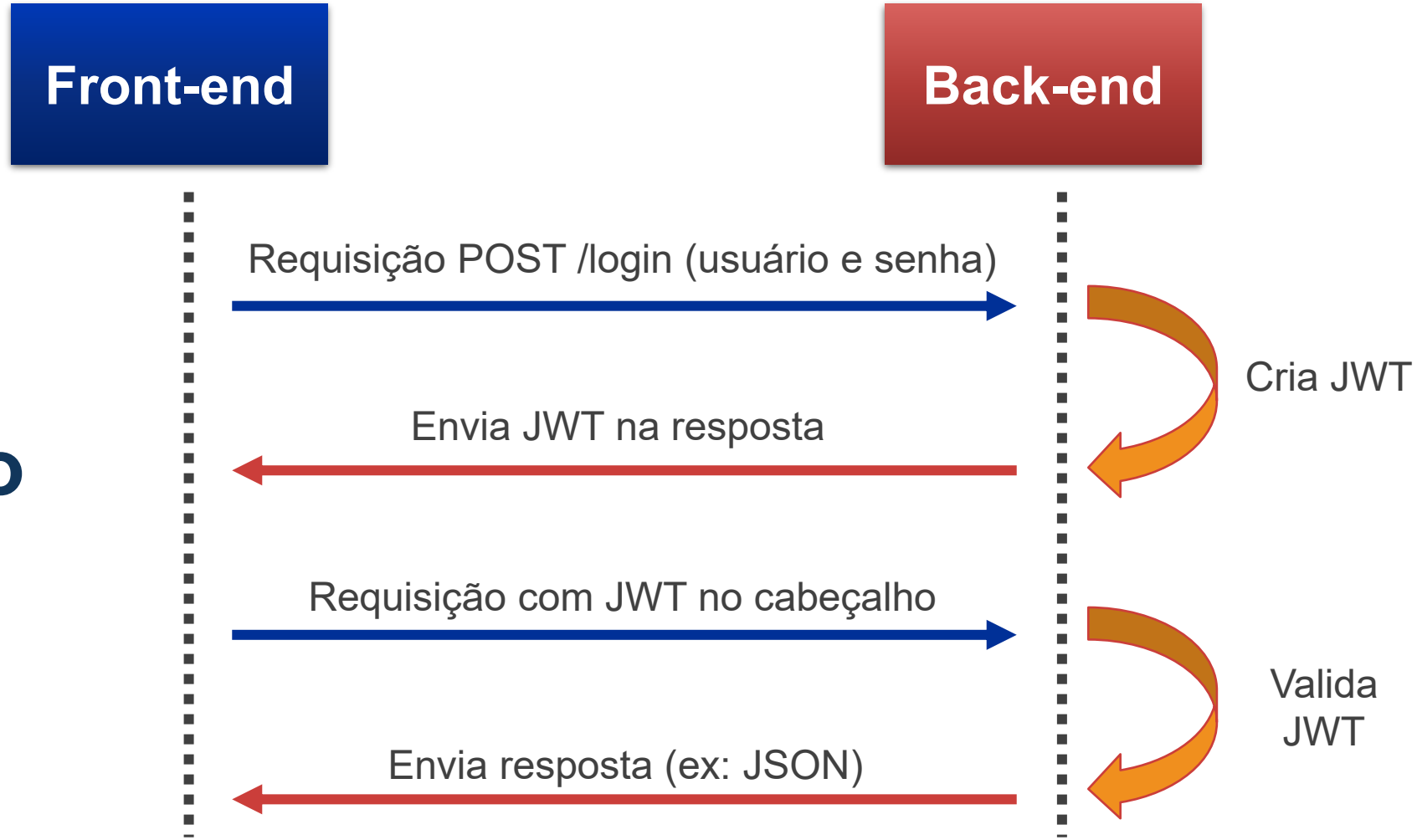
Provedor de autenticação que implementa um DAO para recuperar informações do usuário.



Fonte: https://docs.spring.io/spring-security/reference/6.5/servlet/authentication/passwords/dao-authentication-provider.html

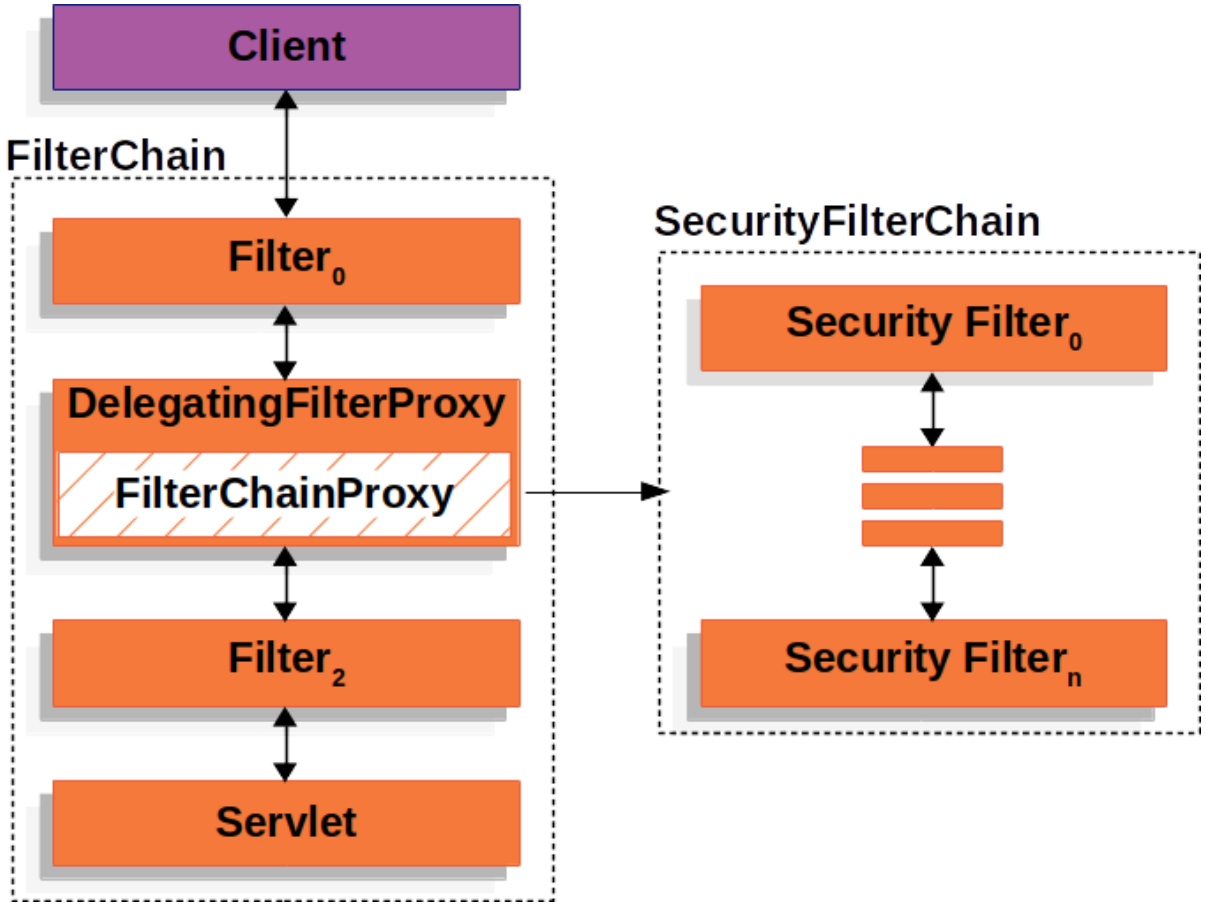
Autenticação JWT

JSON Web Token



Cadeia de Filtros de Segurança no Spring Security

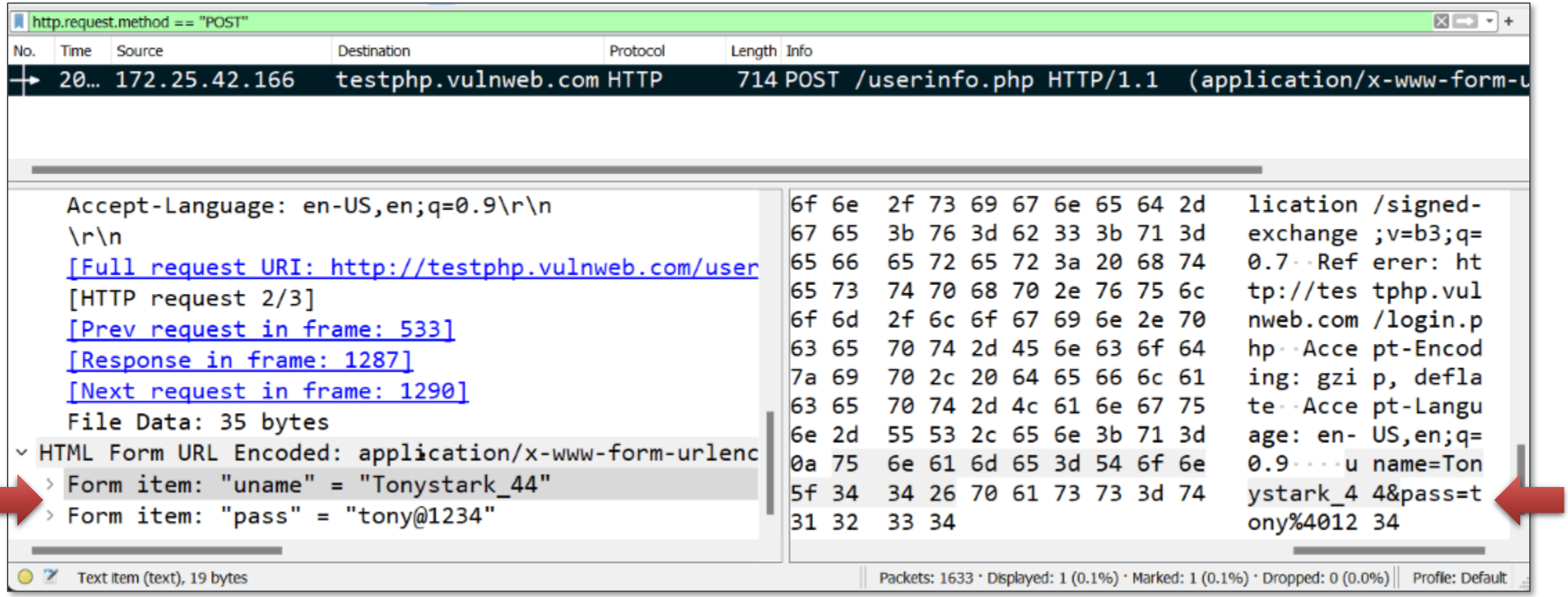
- O **Spring Security aplica uma série de filtros de segurança específicos** para verificar autenticação, autorização e outras políticas de segurança.
- Após passar por todos esses filtros a requisição chega ao **Servlet** para ser processada normalmente.



Fonte: https://docs.spring.io/spring-security/reference/6.5/servlet/authentication/passwords/dao-authentication-provider.html

Segurança na comunicação via HTTP

Usuário e senha sendo capturados no Wireshark:



Fonte: https://www.geeksforgeeks.org/sniffing-of-login-credential-or-password-capturing-in-wireshark/

SSL/TLS

- **SSL** (**S**ecure **S**ockets **L**ayer) permite o tráfego de dados pela rede de forma segura, estabelecendo um **canal de comunicação entre aplicações onde as informações são criptografadas**.
- **TLS** (**T**ransport **L**ayer **S**ecurity) é o **successor do SSL** e funciona de forma semelhante.
  - Apesar do termo SSL ser mais popular, na maioria das vezes o termo correto que deveria ser utilizado é TLS.
- O protocolo **HTTPS** é uma implementação do HTTP com uma camada adicional de segurança (HTTPS = HTTP + SSL/TLS).

Certificados autoassinados

- São **certificados gerados por entidade própria**, sem validação por uma Autoridade Certificadora (CA), e são normalmente usados em ambientes de desenvolvimento, testes internos e **sistemas locais onde a validação externa não é essencial**.
- Possuem a **vantagem da redução de custo**, mas sem validação de uma CA confiável, **aumentam as chances de problemas de segurança**.
- **Serviços de nuvem normalmente fornecem acesso seguro** (HTTPS), e a há ainda a possibilidade de utilizar certificado próprio.

Habilitar SSL no Spring Boot

- **Criar certificado**

```
keytool -genkeypair -alias SGCM -keyalg RSA -keysize 2048 -storetype PKCS12 -keystore certificado.p12
-validity 3650 -dname "CN=SGCM, OU=localhost, O=UFAC, L=Rio Branco, S=AC, C=BR" -ext
san=dns:localhost
```

```
keytool -export -keystore certificado.p12 -alias SGCM -file certificado.crt
```

  - Os arquivos **certificado.p12** e **certificado.crt** devem ser colocados no diretório **src/main/resources/**

- **application.properties**

```
server.ssl.key-store=classpath:certificado.p12
server.ssl.key-store-password=webacademy
server.ssl.key-store-type=PKCS12
```

Práticas Avançadas

Padrão de projeto DTO

- O padrão **DTO** (*Data Transfer Object*) representa um **formato utilizado na transferência de dados** entre sistemas diferentes (ou entre camadas de um mesmo sistema).
- Não é o mesmo que classes da camada de modelo: **DTO não representa um objeto que será persistido**.

```
public class UsuarioDto {
    // Mesmos atributos da camada
    // de modelo, mas sem a senha.
    private Long id;
    private String nomeCompleto;
    private String nomeUsuario;
    private String papel;
    private boolean ativo;
    // Getters e Setters
}
```

DTO com Java Records

```java
public class UsuarioDto {
    private Long id;
    private String nomeCompleto;
    private String nomeUsuario;
    private String papel;
    private boolean ativo;
    // Getters e Setters
}
```

```java
public record UsuarioDto(
    Long id,
    String nomeCompleto,
    String nomeUsuario,
    String papel,
    boolean ativo
) {}
```

```java
// Acessar um atributo
dto.nomeCompleto();
```

Java Records são imutáveis, pois uma vez criado, o estado de um objeto não podem ser alterado. Não há como definir valor para os atributos após criação do objeto (sem *setters).*

Validação de dados no back-end

- Validação é uma forma de **verificação de qualidade dos dados**, garantindo que as informações fornecidas seguem as regras definidas.
- A **validação no back-end ajuda a garantir a segurança e a integridade dos dados**, independentemente da verificação feita no front-end, que pode ser manipulada.
  - Validação no front-end tem objetivo diferente: melhorar a experiência do usuário.
- Princípio ***Fail Fast***: retornar o mais rápido possível uma falha, interrompendo a operação atual, e evitando uso de recursos desnecessários.

Validação de dados no back-end

- Spring Boot **Starter Validation** inclui no projeto recursos do Java Bean Validation, que permitem realizar validação por meio de anotações.
- Exemplos de validadores: @NotNull, @NotBlank e @Email.
- Lista de validadores:
  - https://jakarta.ee/specifications/bean-validation/3.1/jakarta-validation-spec-3.1#builtinconstraints
- Também é possível criar **validadores customizados**.

Paginação e ordenação de resultados

- **Paginação de resultados** é uma técnica comum em aplicações que acessam bancos de dados, com objetivo de melhorar a **performance**, **usabilidade**, além de **reduzir a carga sobre o servidor da aplicação**.

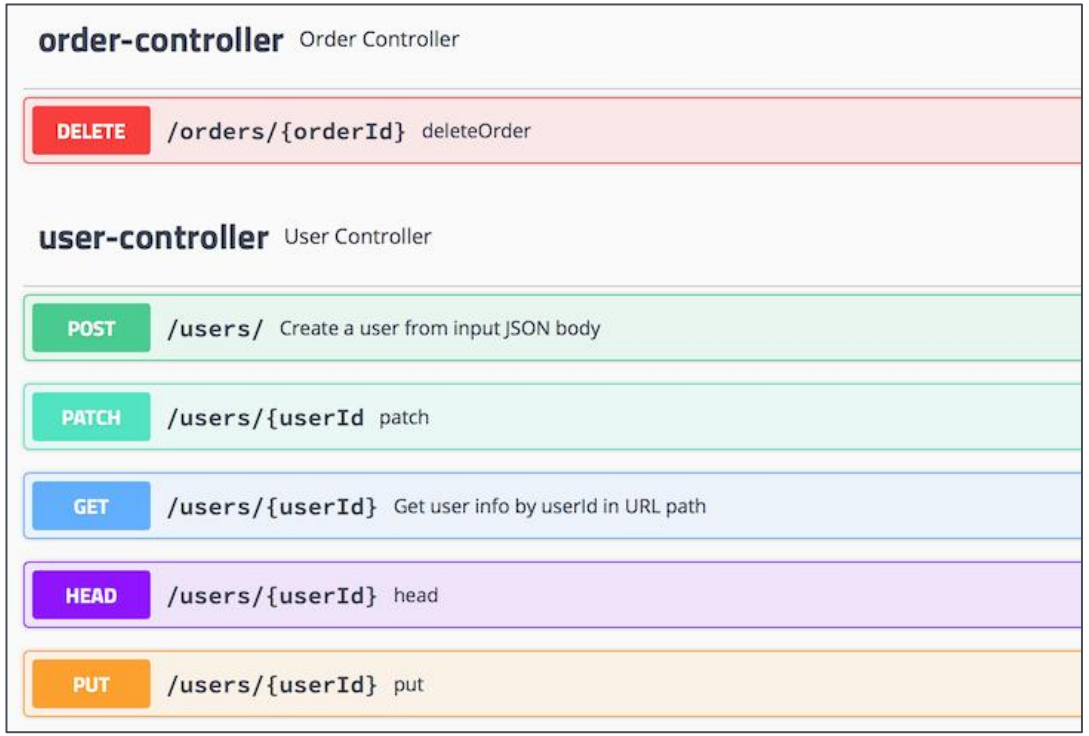
- O **Spring Data** fornece a interface **Pageable**, que facilita a implementação de **paginação e ordenação** em consultas a banco de dados.

```
public ResponseEntity<Page<Atendimento>> consultar(Pageable page) {
    Page<Atendimento> registros = servico.consultar(page);
    return ResponseEntity.ok(registros);
}
```

- Exemplo de requisição: `GET /atendimento/consultar?page=0&size=10&sort=data,desc&sort=hora,asc`

Documentação de APIs

- A documentação é uma **parte essencial na implementação de REST APIs**, seja para servir a um front-end ou mesmo para integrar com outras aplicações.
- É importante descrever a especificação de uma API para evitar **redundâncias**, garantir **comunicação clara**, facilitar a **integração**, assegurar **consistência**, e possibilitar **escalabilidade** e **reuso**.
- Exemplo: https://petstore.swagger.io/



Swagger

- Swagger é um **conjunto de ferramentas** de código aberto **para projetar, documentar e usar APIs**.
  - Exemplos: Swagger Editor, Swagger UI e Swagger Codegen.
- Utiliza a **especificação OpenAPI**, que padroniza a forma de descrever APIs.
- OpenAPI foi originalmente desenvolvido como ***Swagger Specification*** (até versão 2), e depois padronizado e mantido pela OpenAPI Initiative (versão 3).

Integração do Swagger com Spring Boot

- O trabalho de construir e manter atualizada a documentação é difícil de ser realizado de forma manual, mas a **integração com frameworks permite automatizar o processo**.

- No Spring, a integração pode ser feita com auxílio de uma ferramenta mantida pelo projeto springdoc-openapi.

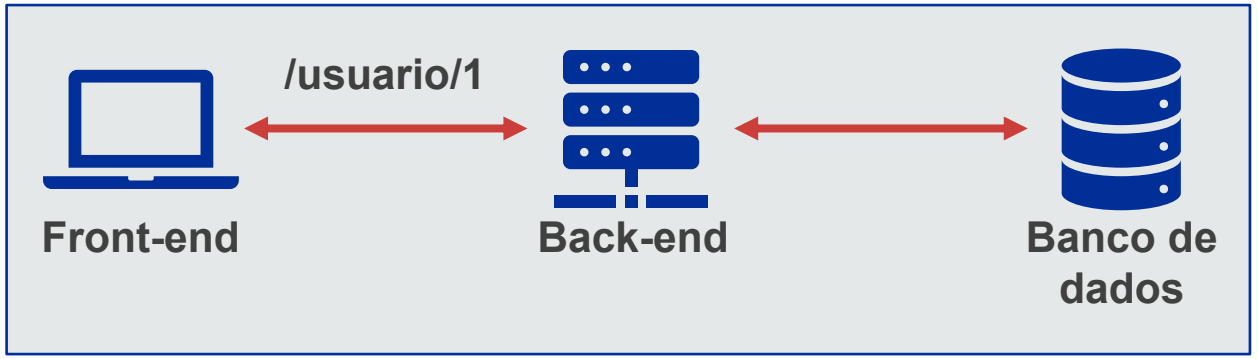
```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.8.6</version>
</dependency>
```

Integração e otimização

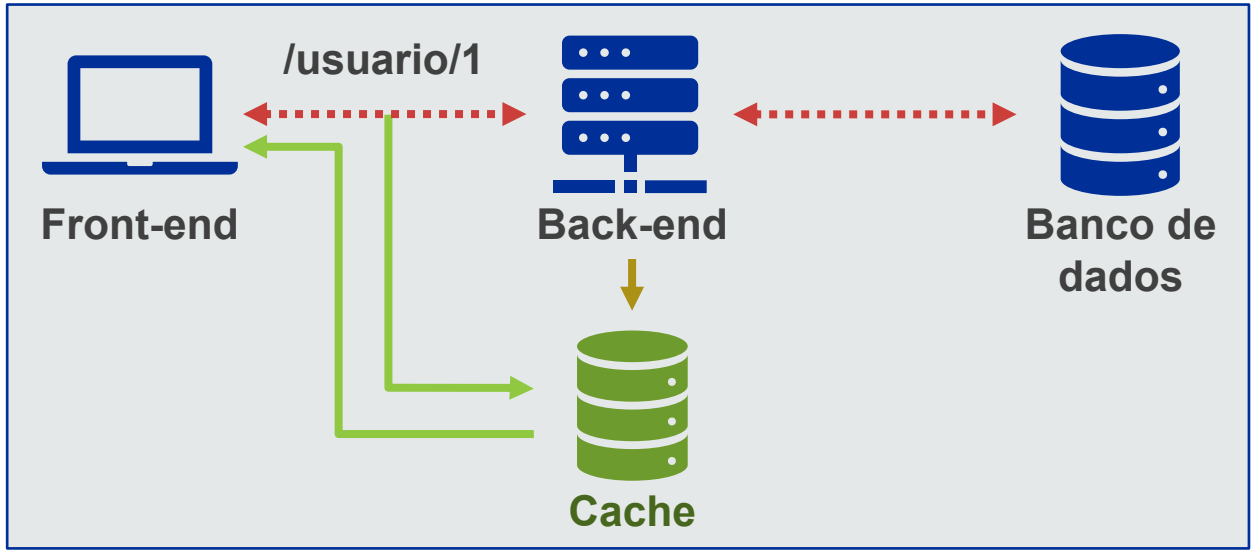
Armazenamento em cache

- O **cache** é um tipo de **armazenamento temporário** que permite **acesso rápido** a dados, melhorando o desempenho.
- **Armazena resultados de operações frequentes** que não são alterados em um determinado intervalo de tempo.
- **Cache** e **Buffer** são diferentes: buffer armazena dados temporariamente entre dispositivos que funcionam em diferentes velocidades ou taxas de transferência.

Sem cache



Com cache



Cache no Spring

- O Spring Framework possui um mecanismo para habilitar cache, mas o recurso de armazenamento depende de um provedor de cache.
- Se nenhum provedor for especificado, o Spring habilita o recurso de **armazenamento em cache baseado em um tipo de HashMap**.
- Também **possui suporte a bancos de dados em memória**, como o Redis.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-cache</artifactId>
</dependency>
```

Cache no Spring

Anotações

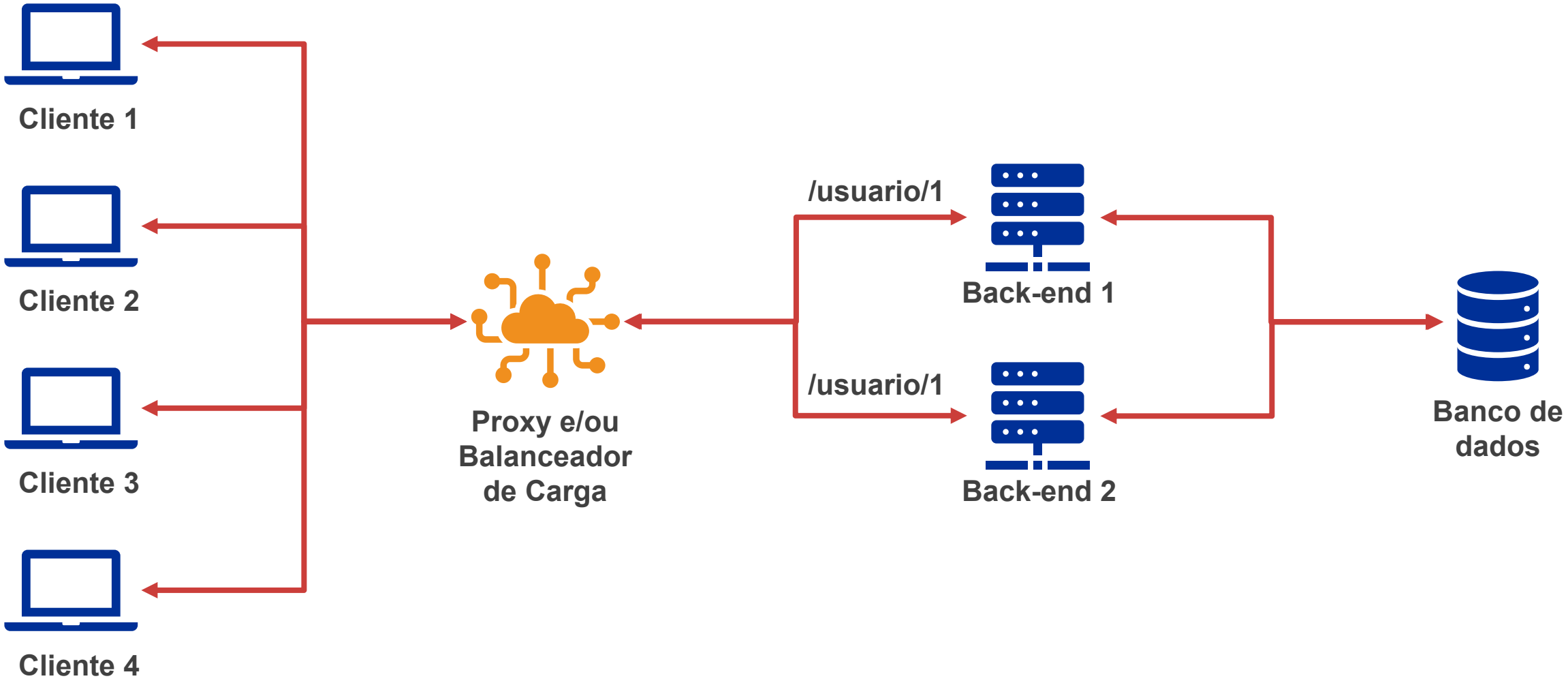
- **@Cacheable**: armazena dados no cache.
- **@CacheEvict**: remove dados do cache.
- **@CachePut**: atualiza o cache.
- **@Caching**: agrupa múltiplas operações de cache.

Cache condicional

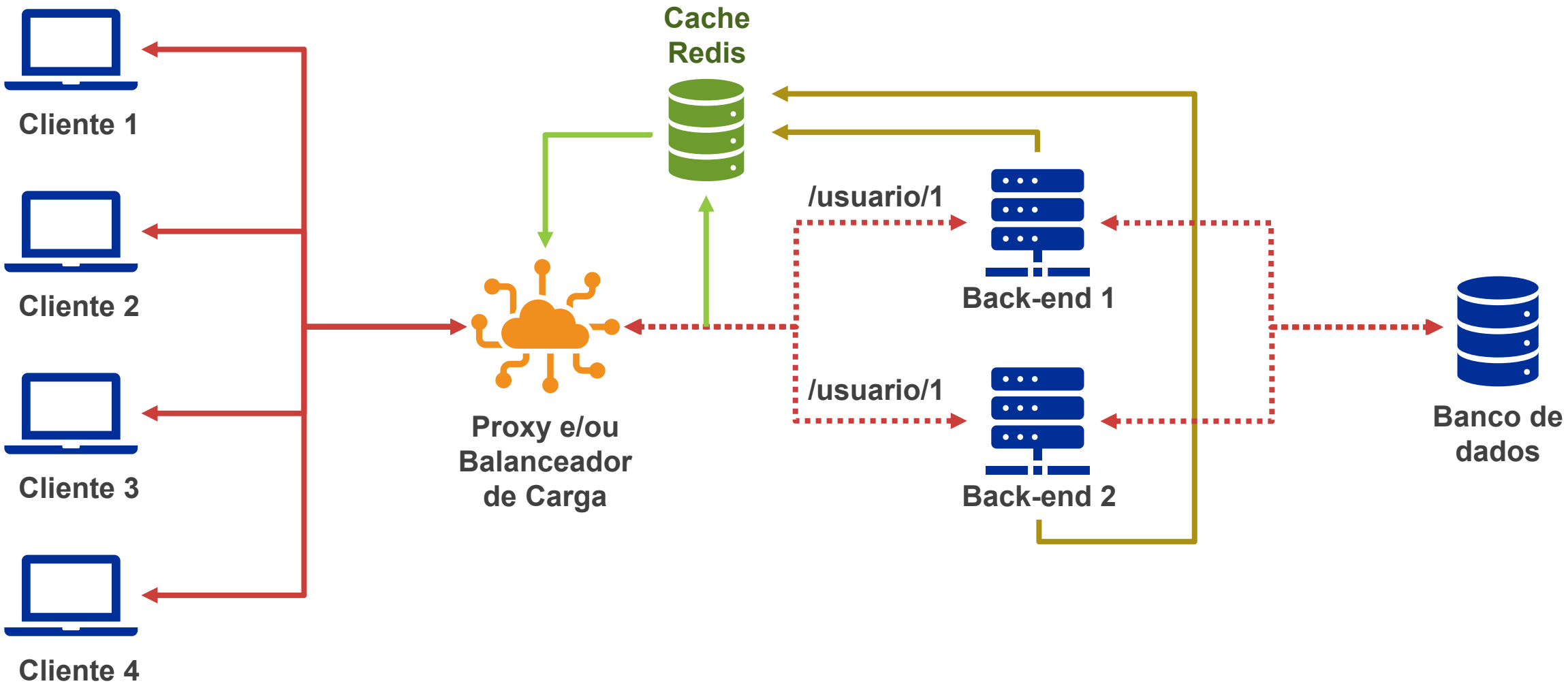
- **unless**: realiza a operação de cache **se a expressão for falsa** (após método ser executado).
- **condition**: realiza a operação de cache **se a expressão for verdadeira** (antes do método ser executado).

Referência: https://docs.spring.io/spring-framework/reference/integration/cache/annotations.html

Cache em ambientes escaláveis



Cache em ambientes escaláveis



Acesso a APIs externas

- A **integração com serviços externos** é uma prática comum, frequentemente realizada através de operações com **REST APIs**, permitindo acesso a dados e funcionalidades adicionais.
- Além de conectar-se a **serviços como pagamentos, dados meteorológicos e redes sociais**, também facilita a integração com sistemas internos da própria organização.
- É fundamental considerar aspectos relacionados à **segurança dos dados** e o impacto no **desempenho da aplicação** ao implementar essas integrações.

Acesso a APIs externas com Spring

- Opções do Spring para realizar operações em REST APIs:
  1. **RestTemplate**: método mais antigo, disponível desde a versão 3 do Spring Framework, que realiza chamadas utilizando comunicação síncrona (bloqueante).
  2. **WebClient**: introduzido na versão 5 do Spring (depende do Spring WebFlux), oferece comunicação assíncrona e sintaxe funcional semelhante à Stream API do Java.
  3. **RestClient**: opção mais recente, semelhante ao WebClient, mas com comunicação síncrona, funcionando como alternativa ao RestTemplate.

Referência: https://docs.spring.io/spring-framework/reference/integration/rest-clients.html

Fim!

Referências

- DEITEL, Paul; DEITEL, Harvey. **Java: Como Programar**. 10. ed. São Paulo: Pearson, 2016. 968 p.
- MARCO TULIO VALENTE. **Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade**, 2020. Disponível em: https://engsoftmoderna.info/
- MOZILLA (ed.). **MDN Web Docs: Aprendendo desenvolvimento web**. [S. l.], 2025. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn.
- SPRING (ed.). **Spring Boot Reference Documentation**. [S. l.], 2025. Disponível em: https://docs.spring.io/spring-boot/docs/current/reference/html/index.html.
- WALLS, Craig. **Spring in Action**. 6. ed. Shelter Island: Manning, 2021. 520 p.