Sistemas Web Seguros

UNIDADE 01

Fundamentos da arquitetura orientada a serviços

Nesta unidade, vamos conhecer os fundamentos da Arquitetura Orientada a Serviços (SOA) e duas abordagens de transmissão de dados *on-line*: *Simple Object Access Protocol* (SOAP) e *Representational State Transfer* (REST). Por meio desses conhecimentos, vamos projetar soluções seguras para a comunicação entre sistemas *web*, como no caso do SISFIN e SISRH.

ARQUITETURA ORIENTADA A SERVIÇOS

Sistemas complexos

Quando começamos nossos estudos em engenharia de *software*, é natural iniciar com sistemas **tradicionais**, ou seja, simples, homogêneos, bem delimitados, sobre uma arquitetura padrão e construídos de forma monolítica. Contudo, na maioria das vezes, as soluções de TI reais que utilizamos em nosso dia a dia são **complexas**, com tecnologias e abordagens heterogêneas, dinâmicas e compostas por diferentes entidades autônomas.

A modelagem de sistemas complexos requer um paradigma próprio que atenda a algumas necessidades:

- Estruturar nossa visão por serviços, ou seja, as funcionalidades que são entregues por cada sistema.
- Concentrar-se na interoperabilidade, ou seja, ter a capacidade de comunicação com diversos outros componentes.
- Como cada componente de um sistema complexo pode ser autônomo, requer uma abordagem flexível e distribuída. A isso chamaremos fracamente acoplada.
- Criar um alto nível de abstração, para sermos capazes de lidar com a complexidade desses sistemas.

Arquitetura orientada a serviços

Nas últimas décadas, um novo paradigma de arquitetura de sistemas emergiu dentro da TI, denominado SOA. Trata-se de uma resposta à necessidade de construir *softwares* adaptados a ambientes distribuídos e heterogêneos que a internet tornou mais habituais; portanto, é um paradigma de arquitetura que se ajusta bem aos sistemas complexos.

A SOA é uma abordagem para arquitetura de sistemas distribuídos que emprega serviços fracamente acoplados, com interfaces e protocolos de comunicação padrões para fornecer integração entre plataformas heterogêneas.

Um exemplo para ajudar

Imagine que vamos construir um aplicativo *web* que permita a qualquer pessoa alugar bicicletas que estão distribuídas em uma região metropolitana.

Uma alternativa seria passarmos anos desenvolvendo um subsistema de geolocalização que funcionaria exibindo mapas com a localização em tempo real das bicicletas e depois outro subsistema para lidar com os pagamentos, além de outro subsistema para *login*, autenticação de usuário e assim por diante.

Figura 1. Sistema composto pela construção de subsistemas

Outra possibilidade seria simplesmente aproveitar o serviço de mapas do Google para localização das bicicletas, um serviço de pagamento como o PayPal e usar um serviço de *login* do Facebook para autenticar os usuários.

Provavelmente, o único módulo a ser construído, de fato, são os serviços de locação para calcular as tarifas, multas, tempo de corrida, entre outros. Nosso trabalho seria integrar esse módulo a diversos serviços.

Portanto, em vez de construirmos um sistema baseado em todos os diferentes componentes internos, podemos desenvolver um aplicativo com uma SOA.

Figura 2: Sistema composto por integrações a outros serviços

No capítulo 4 do livro **SOA: princípios de design de serviços**, podemos ver a orientação a serviços como um paradigma de *design* que se baseia no princípio da **separação de preocupações**. Essa teoria afirma que um problema maior pode ser resolvido quando decomposto em um conjunto de subproblemas menores. De forma análoga, quando projetamos um sistema pela decomposição de subsistemas ou serviços, estamos aplicando a separação de preocupações. Para decomposição de um sistema em múltiplos serviços, podemos observar oito princípios (ERL, 2009):

1. **Contrato de serviço padronizado:** o serviço deve ter um propósito claro, com entradas e saídas definidas; por exemplo, o serviço de autenticação deve ser capaz de receber credenciais do usuário, validá-las e, quando autorizado, retornar as respectivas autorizações.
2. **Baixo acoplamento de serviço:** o acoplamento se refere ao relacionamento entre dois ou mais elementos. Quando construímos serviços com baixo acoplamento, permitimos que a lógica do serviço evolua independentemente da implementação de outros serviços.
3. **Abstração de serviço:** os serviços devem ocultar, aos seus consumidores, o maior número possível de detalhes subjacentes. Lógicas internas devem ser abstraídas, garantindo aos consumidores a simplicidade de uso.
4. **Capacidade de reúso:** sempre que possível, deve-se projetar serviços de tal forma que seu uso possa ser ampliado para outros cenários; um serviço de autenticação,

por exemplo, possui um alto grau de reúso, pois a maioria dos sistemas necessita dessa função.

5. **Autonomia de serviço:** para um serviço realizar seu propósito, é essencial que ele tenha um grau significativo de controle sobre o seu ambiente e recursos.
6. **Independência de estado:** este princípio recomenda que os serviços armazenem informações de estado apenas se for estritamente necessário. O excessivo armazenamento para controle de estado acaba minando a escalabilidade do serviço.
7. **Visibilidade do serviço:** os serviços devem ter mecanismos que facilitem ser encontrados. Para nos beneficiar do reúso, os serviços devem prover formas de ser identificados, além de ter orientações padronizadas para facilitar o entendimento do seu uso.
8. **Composição de serviços:** à medida que sofisticamos nossas soluções, os serviços podem ser compostos, sendo utilizados de modo orquestrado. A composição de serviços demanda certo esforço de adaptação, mas espera-se que eles atuem na resolução de funcionalidades mais complexas.

Vamos a um exemplo genérico!

EXEMPLO

Agora, vamos analisar um exemplo fora da TI para ilustrar os oito princípios da orientação a serviços.

Imagine que somos sócios de uma hamburgueria. Nosso propósito é oferecer aos clientes refeições e bebidas em um ambiente moderno e aconchegante. Para atingi-lo, precisamos lidar com diferentes cenários: aquisição de ingredientes de qualidade, funcionários bem treinados, decoração alinhada ao estilo do estabelecimento e assim por diante.

Podemos analisar nossa hamburgueria como um sistema; não pense em *software*, mas em um sistema organizacional, composto por outros subsistemas. Precisamos definir uma estrutura na qual os subsistemas possam interoperar para entregar o serviço esperado. Vamos pensar na estrutura desse sistema utilizando os oito princípios da SOA.

Em primeiro lugar, cada componente do sistema está prestando um serviço, seja o funcionário que serve uma refeição, seja a cozinha que prepara os alimentos ou, ainda, as cadeiras nas quais as pessoas se sentam, tudo que compõe a hamburgueria deve ser representado.

Vamos projetar esse sistema! Não estamos interessados no funcionamento interno de cada componente; por exemplo, não precisamos saber qual princípio da física está por trás de um *freezer*, como o *chef* de cozinha manipula os ingredientes ou como o sistema de ar-condicionado funciona internamente. A isso chamamos **abstração de serviço**, pois não precisamos dessa informação. Apenas o provedor do serviço precisa conhecer a lógica interna do componente. Para nós, são simplesmente serviços.

Portanto, quando um cliente paga a sua conta com seu cartão de crédito, ele simplesmente passa o cartão e insere sua senha ou outro PIN de autenticação. Ninguém na loja entende, ou precisa entender, como a transação é processada. Apenas o provedor do serviço de pagamento tem essas informações.

Também podemos observar que o provedor do serviço de pagamento tem controle quase total sobre a lógica interna. A isso chamamos **autonomia de serviço**. Ainda sobre o pagamento do

cartão de crédito, sabemos que, para o cliente e o atendente, é importante entender o que eles devem fazer para interagir com esse componente e quais respostas o componente enviará. Chamamos isso **contrato de serviço padronizado**.

Também podemos notar uma placa acima da lixeira com a frase “Deixei seu lixo aqui”, incentivando os clientes a descartar seus resíduos. Isso é chamado **visibilidade do serviço**. Os serviços são complementados por metadados comunicativos pelos quais podem ser efetivamente descobertos e interpretados.

Quando atendemos a um cliente, não nos preocupamos em armazenar muitos dados sobre ele, não temos interesse neles. Cada vez que um cliente entra em nossa hamburgueria, é como se fosse a primeira vez, apesar de os atendentes reconhecerem fisionomias ou nomes de alguns clientes (informação mínima); nosso negócio não é armazenar informações desnecessárias. Chamamos isso **independência de estado**.

Como parte do nosso negócio, temos uma rede de fornecedores e pessoal de manutenção. Se precisamos de mais pessoal, ligamos para a agência de recrutamento e teremos um novo funcionário a partir do próximo mês. Se a loja precisa de uma manutenção, podemos ligar para um engenheiro. Se precisamos de mais mesas, também podemos comprá-las. Todos esses diferentes serviços são chamados **baixo acoplamento de serviço**. Isso significa que esses diferentes módulos podem entrar ou sair, acoplar ou desacoplar do sistema conforme a necessidade, mantendo, assim, sua independência.

Finalmente, quando nosso negócio está funcionando, podemos ampliá-lo abrindo uma ou várias filiais. Não temos de começar do zero todas as vezes. Podemos simplesmente estender nosso contrato de seguro, pedir mais mesas, usar a mesma conta bancária e assim por diante. Isso é chamado **capacidade de reúso**, por meio da **composição de serviços**. Como os serviços são independentes de qualquer processo específico, eles podem ser reutilizados, compostos e recompostos em novos negócios.

De volta à TI

Passamos um tempo falando de SOA por meio da nossa hamburgueria, mas SOA oferece sua real contribuição quando aplicada à construção de sistemas complexos, projetados para grandes organizações ou negócios.

Web *services* SOAP e REST

É importante destacar que SOA não é uma tecnologia, mas, sim, um conceito arquitetural de modelagem para sistemas complexos. Quando vamos construir uma solução de TI projetada em SOA, precisamos de tecnologias e protocolos de comunicação.

Atualmente, encontramos duas abordagens para transmissão de dados. **SOAP** é um protocolo de acesso a serviços *web* baseado em padrões que existem há muito tempo, originalmente desenvolvido pela Microsoft. **REST** é outro padrão, feito em resposta às deficiências do SOAP, que procura corrigir os problemas e fornecer um método mais simples de acessar serviços *web*.

Embora tenham semelhanças, como o *Hypertext Transfer Protocol* (HTTP), o SOAP é um conjunto mais rígido de padrões de mensagens do que o REST, sendo as regras importantes para atingirmos um nível bom de padronização nas comunicações. Já o REST é tido como um estilo de arquitetura, com uma estrutura mais simples, portanto é naturalmente mais flexível. Tanto o SOAP quanto o REST dependem de regras bem estabelecidas que todos concordam em obedecer para o propósito de troca de informações.

Simple object access protocol

O SOAP depende exclusivamente de *Extensible Markup Language* (XML) para fornecer serviços de mensagens. A XML usada para fazer solicitações e receber respostas em SOAP pode se tornar extremamente complexa. Um dos recursos do SOAP mais importantes é o tratamento de erros embutido. Se houver um problema com sua solicitação, a resposta conterá informações de erro que você poderá usar para corrigir o problema.

Veja um exemplo de chamada para o *web service* em SOAP para calcular a soma de dois números.

Solicitação para somar dois números:

```
<soapenv:Envelope
```

```
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soap="http://soap/">
   <soapenv:Header/>
   <soapenv:Body>
      <soap:soma>
         <numero1>3</numero1>
         <numero2>5</numero2>
      </soap:soma>
   </soapenv:Body>
</soapenv:Envelope>
```

Resposta do serviço SOAP com o resultado da soma:

```
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:somaResponse xmlns:ns2="http://soap/">
         <return>8.0</return>
      </ns2:somaResponse>
   </S:Body>
</S:Envelope>
```

Representational state transfer

O REST oferece uma alternativa mais leve. Muitos desenvolvedores acharam o SOAP complicado e difícil de usar; por exemplo, trabalhar com SOAP em JavaScript obriga o desenvolvedor a escrever muito código para realizar tarefas simples, porque deve-se criar a estrutura XML para qualquer requisição.

Em vez de usar XML para fazer uma solicitação, o REST pode funcionar por meio de um simples *Uniform Resource Locator* (URL). Em algumas situações, você deve fornecer informações adicionais, mas a maioria dos serviços *web* que usam REST depende exclusivamente do uso da abordagem de URL.

EXEMPLO

Veja o mesmo exemplo da calculadora para um *web service* em REST.

Chamada:

http://servidor/sisrh/rest/calculadora/soma?numero1=**3**&numero2=**5**

Resposta:

8

Percebeu como a chamada REST é mais simples e como a chamada SOAP é mais padronizada?

Nas próximas unidades, vamos nos aprofundar nas duas abordagens.

Vantagens de cada abordagem

O SOAP oferece as seguintes vantagens, quando comparado ao REST (SMARTBEAR, 2020):

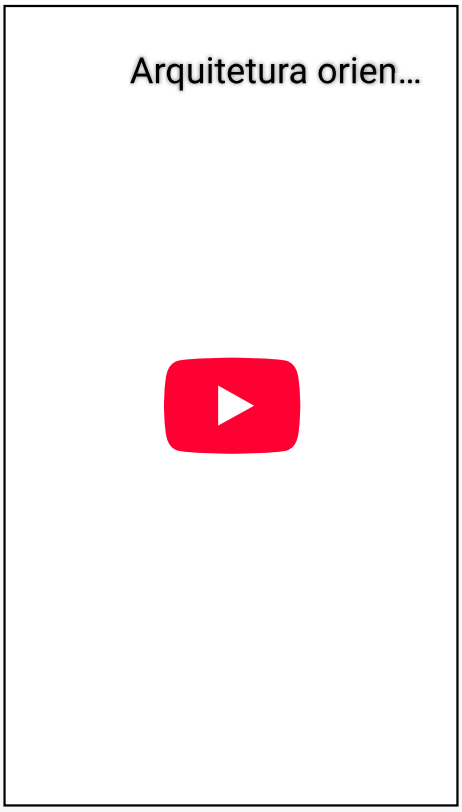
- Linguagem, plataforma e transporte independentes (REST requer o uso de HTTP).
- Funciona bem em ambientes corporativos distribuídos (REST assume comunicação direta ponto a ponto).
- Padronização.
- Tratamento de erros embutido.
- Automação quando usado com determinados produtos de linguagem.

O REST é mais fácil de usar e mais flexível, tendo as seguintes vantagens sobre o SOAP (SMARTBEAR, 2020):

- Nenhuma ferramenta sofisticada para interagir com o serviço *web*.
- Menor curva de aprendizado.
- Eficiente (SOAP usa XML para todas as mensagens, REST pode usar formatos de mensagem menores).
- Rápido, pois não requer processamento extensivo.
- Mais próximo de outras tecnologias da *web*, com o JavaScript.

Arquitetura orientada a serviços com REST e SOAP

Vamos nos aprofundar nas abordagens REST e SOAP e como cada uma pode nos auxiliar na comunicação de serviços *web* projetados em SOA.



Embora o SOAP seja a opção preferida por muitas empresas, para outras é muito complexo e não flexível o suficiente. Ambas as arquiteturas de informação têm seus nichos específicos, por isso vamos conhecê-las com mais detalhes nas próximas unidades.

CONCLUSÃO

- A SOA deve ser vista como um paradigma de integração entre sistemas complexos de engenharia.
- SOA é um conceito, não uma tecnologia. Para criar uma solução SOA, podemos utilizar os dois padrões mais aceitos pela indústria de *software*: SOAP e REST.
- SOAP é um protocolo mais padronizado, baseado em estrutura XML.
- REST oferece um mecanismo mais simples e leve de ser processado.

REFERÊNCIAS

ERL, T. **SOA**: princípios de design de serviços. São Paulo: Prentice Hall, 2009.

SMARTBEAR. **SOAP vs REST**: what's the difference? 2020. Disponível em: https://smartbear.com/blog/soap-vs-rest-whats-the-difference. Acesso em: 25 out. 2021.