REST

A **Representational State Transfer (REST)**, em português Transferência de Estado Representacional, é uma abstração da arquitetura da World Wide Web(Web), um estilo arquitetural que consiste de um conjunto coordenado de restrições arquiteturais aplicadas a componentes, conectores e elementos de dados dentro de um sistema de hipermídia distribuido. O REST ignora os detalhes da implementação de componente e a sintaxe de protocolo com o objetivo de focar nos papéis dos componentes, nas restrições sobre sua interação com outros componentes e na sua interpretação de elementos de dados significantes.

O termo transferência de estado representacional foi apresentado e definido no ano de 2000 por Roy Fielding, um dos principais autores da especificação do protocolo HTTP que é utilizado por sites da Internet, em uma tese de doutorado (PHD) na UC Irvine. A REST tem sido aplicada para descrever a arquitetura web desejada, identificar problemas existentes, comparar soluções alternativas e garantir que extensões de protocolo não violem as principais restrições que fazem da Web um sucesso. Fielding desenvolveu a REST em colaboração com seus colegas enquanto trabalhava no HTTP 1.1 e nos Identificadores de Recursos Uniformes.

O estilo arquitetural de REST também é aplicado no desenvolvimento de serviços web. Pode-se caracterizar os web services como "RESTful" se eles estiverem em conformidade com as restrições descritas na seção restrições arquiteturais.

O termo *REST* se referia, originalmente, a um conjunto de princípios de arquitetura (descritos mais abaixo), na atualidade se usa no sentido mais amplo para descrever qualquer interface web simples que utiliza XML (ou YAML, JSON, ou texto puro) e HTTP, sem as abstrações adicionais dos protocolos baseados em padrões de trocas de mensagem como o protocolo de serviços Web SOAP. É possível projetar sistemas de serviços Web de acordo com o estilo arquitetural REST descrito por Fielding, e também é possível projetar interfaces XMLHTTP de acordo com o estilo de RPC mas sem utilizar SOAP. Estes usos diferentes do termo *REST* causam certa confusão em discussões técnicas, onde RPC não é um exemplo de REST.

Os sistemas que seguem os princípios REST são freqüentemente chamados de *RESTful'.*

REST afirma que a Web já desfrutou de escalabilidade como resultado de uma série de conceitos de projeto fundamentais:

- Um *protocolo cliente/servidor sem estado*: cada mensagem HTTP

contém toda a informação necessária para compreender o pedido. Como resultado, nem o cliente e nem o servidor necessitam gravar nenhum estado das comunicações entre mensagens. Na prática, muitas aplicações baseadas em HTTP utilizam cookies e outros mecanismos para manter o estado da sessão (algumas destas práticas, como a reescrita de URLs, não são permitidas pela regra do REST).

- Um conjunto de *operações bem definidas* que se aplicam a todos os *recursos* de informação: HTTP em si define um pequeno conjunto de operações, as mais importantes são **POST**, **GET**, **PUT** e **DELETE**. Com frequência estas operações são combinadas com operações CRUD para a persistência de dados, onde POST não se encaixa exatamente neste esquema.
- Uma *sintaxe universal* para identificar os recursos. No sistema REST, cada recurso é unicamente direcionado através da sua URI.
- O *uso de hipermídia*, tanto para a informação da aplicação como para as transições de estado da aplicação: a representação deste estado em um sistema REST são tipicamente HTML ou XML. Como resultado disto, é possível navegar com um recurso REST a muitos outros, simplesmente seguindo ligações sem requerer o uso de registros ou outra infraestrutura adicional.

REST/RPC

Uma aplicação web REST requer um enfoque de desenho diferente a uma aplicação baseada em RPCs. No RPC, dá-se ênfase à diversidade de operações do protocolo, ou *verbos*; por exemplo uma aplicação RPC poderia definir operações como:

```
getUser()
addUser()
removeUser()
updateUser()
getLocation()
addLocation()
removeLocation()
updateLocation()
listUsers()
listLocations()
findLocation()
findUser()
```

Em REST, ao contrário, a ênfase está na diversidade de recursos, nos *nomes*; por exemplo, uma aplicação REST poderia definir os seguintes tipos de recursos:

```
Usuario {}
Localizacao {}
```

OAuth2

O **OAuth2** é um framework de autorização aberto, poderoso e flexível que pode ser usado para proteção de aplicações e APIs REST. Com OAuth2 é possível prover um mecanismo padronizado para Identity Management, em que todos os componentes do sistema possam interagir de um modo seguro, usualmente onde uma aplicação cliente necessita de acesso a um recurso protegido, agindo no lugar do usuário.

OAuth define quatro funções:

- Proprietário de recursos
- Cliente
- Servidor de recursos
- Servidor de Autorização

Proprietário de recursos: Usuário

O proprietário do recurso é o usuário que autoriza um aplicativo a acessar sua conta. O acesso do aplicativo à conta do usuário é limitado ao "escopo" da autorização concedida (por exemplo, acesso de leitura ou gravação).

Servidor de recursos / autorização: API

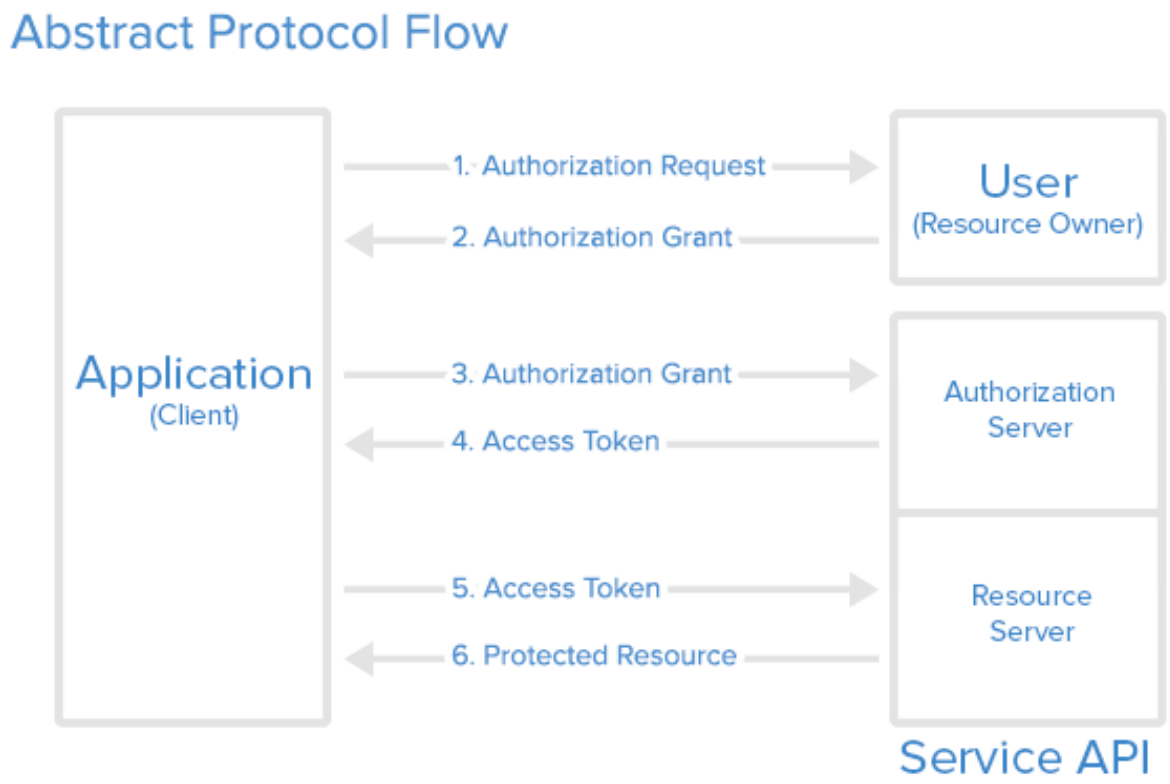
O servidor de recursos hospeda as contas de usuário protegidas e o servidor de autorização verifica a identidade do usuário e emite tokens de acesso ao aplicativo.

Do ponto de vista do desenvolvedor de aplicativos, a API de um serviço cumpre as funções de servidor de recursos e autorização. Nós nos referiremos a ambas as funções combinadas, como a função ou função da API.

Cliente: Aplicação

O cliente é o aplicativo que deseja acessar a conta do usuário. Antes de poder fazê-lo, ele deve ser autorizado pelo usuário e a autorização deve ser validada pela API.

Fluxo de protocolo abstrato



Swagger

O Swagger é um projeto composto por algumas ferramentas que auxiliam o desenvolvedor de APIs REST em algumas tarefas como:

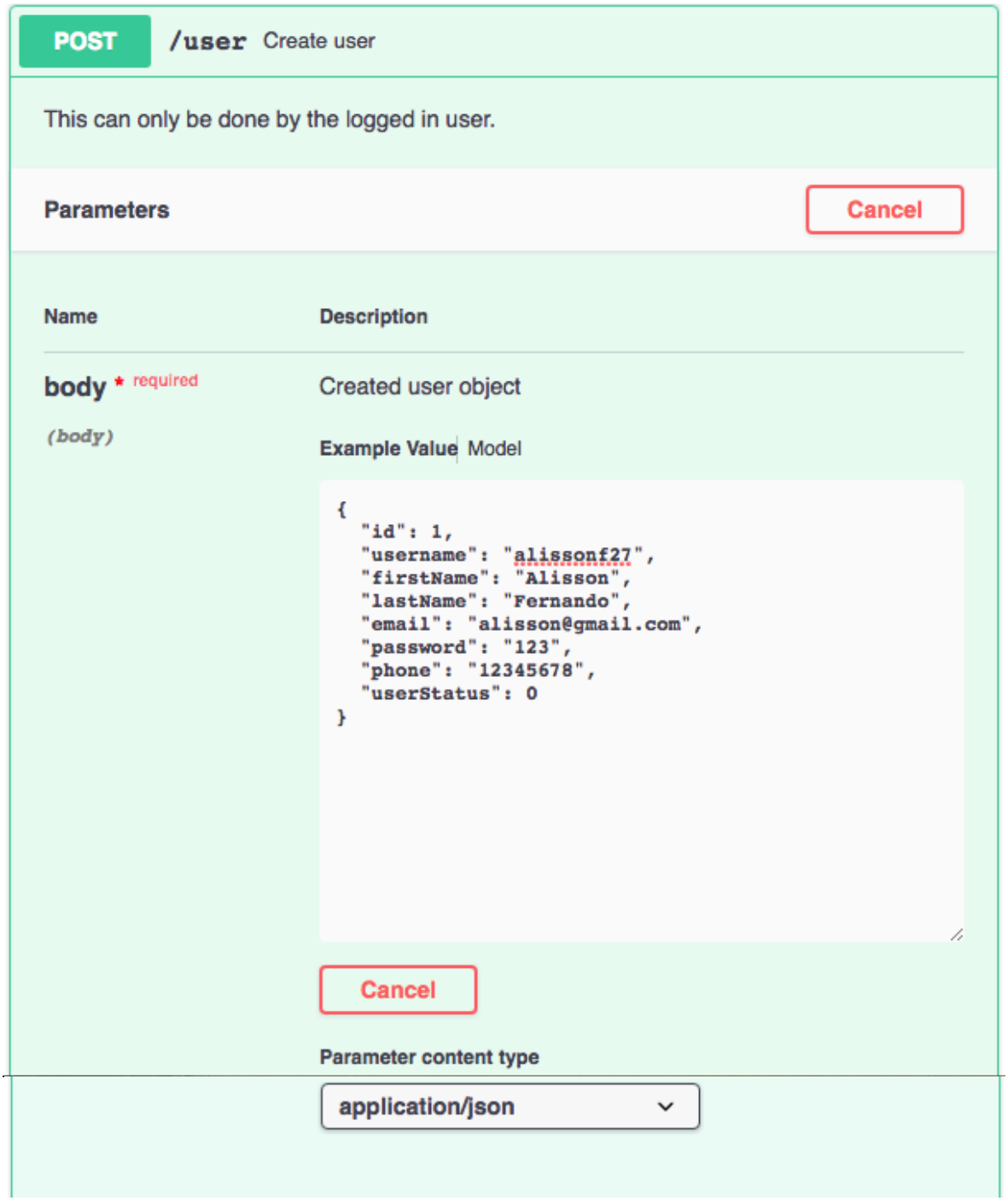
- Modelagem da API.
- Geração de documentação (legível) da API.
- Geração de códigos do Cliente e do Servidor, com suporte a várias linguagens de programação.

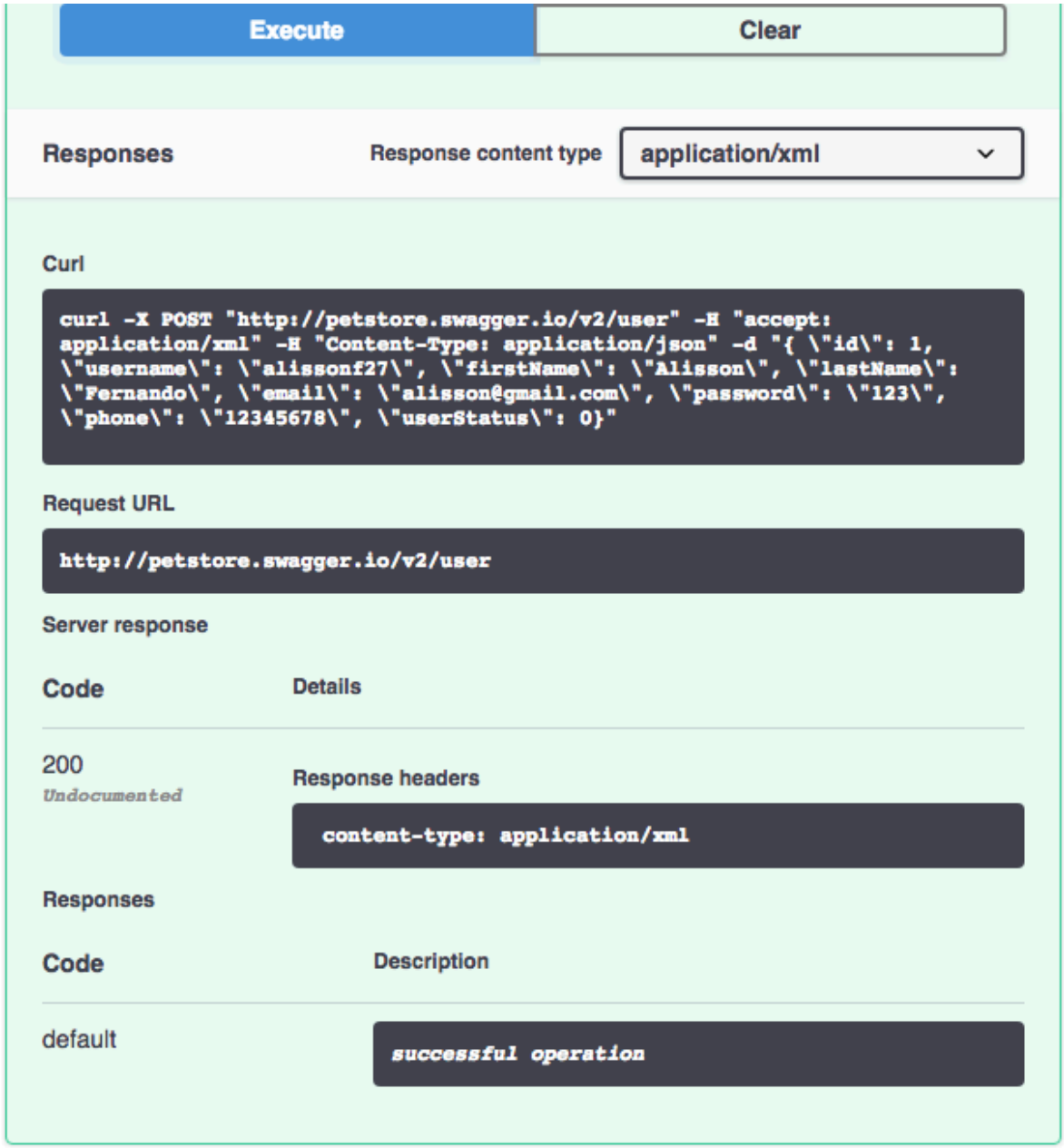
Para isso, o Swagger especifica a OpenAPI, uma linguagem para descrição de contratos de APIs REST. A OpenAPI define um formato JSON com campos padronizados (através de um JSON Schema) para que você descreva recursos, modelo de dados, URIs, Content-Types, métodos HTTP aceitos e códigos de resposta. Também pode ser utilizado o formato YAML, que é um pouco mais legível e será usado nesse post.

Além da OpenAPI, o Swagger provê um ecossistema de ferramentas. As principais são:

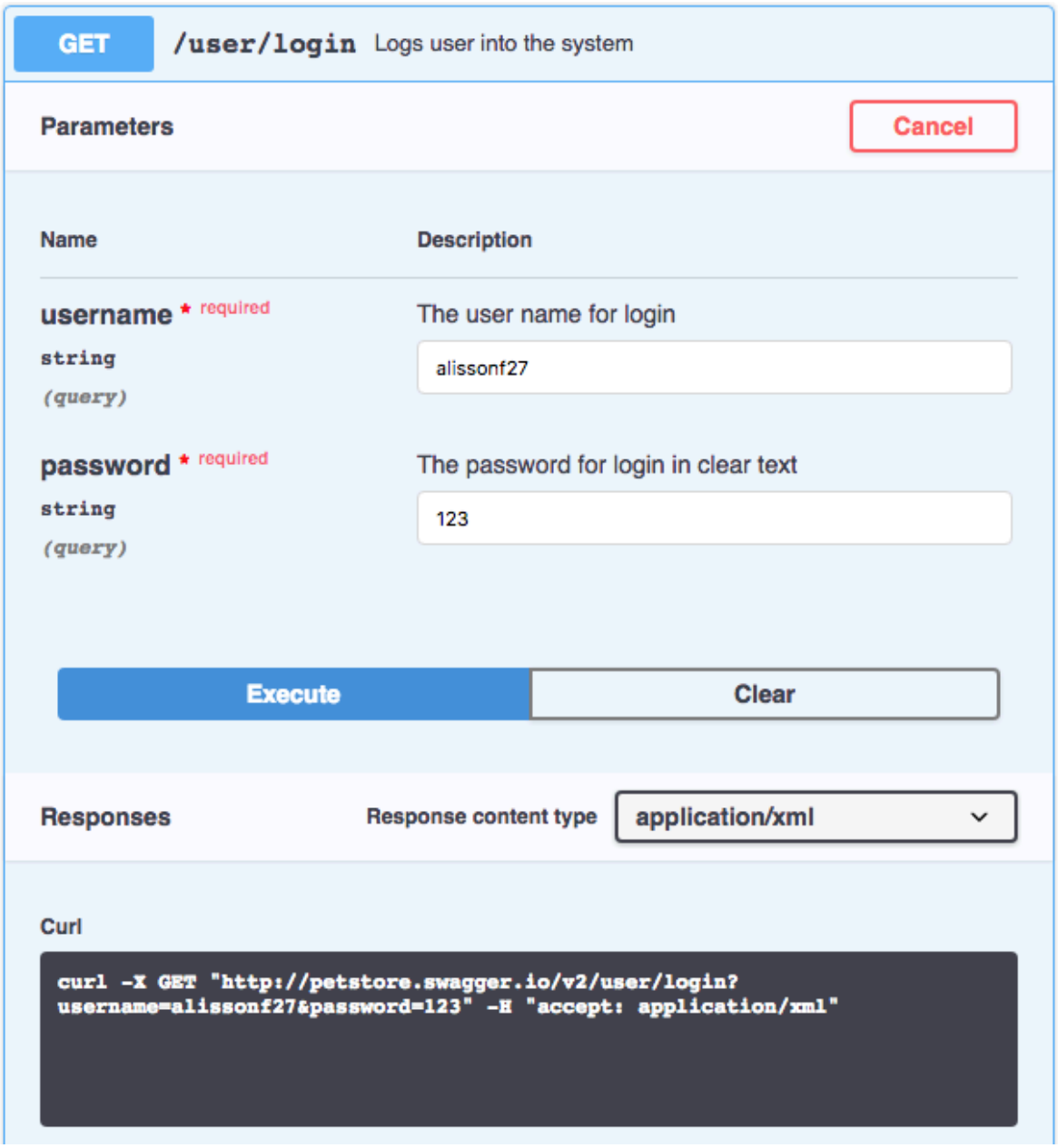
- Swagger Editor – para a criação do contrato.
- Swagger UI– para a publicação da documentação.
- Swagger Codegen – para geração de “esqueletos” de servidores em mais de 10 tecnologias e de clientes em mais de 25 tecnologias diferentes.

Exemplo de cadastramento de usuario:





Exemplo de login de usuario:


GET
/user/login
Logs user into the system
Parameters
Cancel
Name
Description
username * required
string
(query)
The user name for login
alissonf27
password * required
string
(query)
The password for login in clear text
123
Execute
Clear
Responses
Response content type
application/xml
Curl
curl -X GET "http://petstore.swagger.io/v2/user/login?
username=alissonf27&password=123" -H "accept: application/xml"

Relatório:

Começamos o trabalho pesquisando todos os tópicos solicitados em várias fontes diferentes e, conforme íamos encontrando boas fontes, fomos copiando o conteúdo encontrado para um arquivo texto.

Após juntar todas as informações que julgávamos necessárias, fomos lendo, resumindo e sintetizando em um texto de mais fácil compreensão, destacando os pontos mais importantes. À partir dessa

síntese, discorremos as explicações de cada tópico, conforme solicitado.

Fizemos uma simulação no editor do Swagger Online, implementando a função de cadastramento, bem como a de login de usuário, inclusive contando com casos de exceções.

Tivemos muita dificuldade com certos termos técnicos, mas fomos pesquisando mais a fundo cada um deles, para que tivéssemos total compreensão do que e como estávamos fazendo e, dessa forma, poder finalizar o trabalho com êxito.