Regra de negócio:

Vamos montar um sistema de usuários de Clientes e Empresas. As empresas simularão bancos hospedados na nossa API JavaBanks, onde será possível depositar e sacar valores de suas contas.
Devemos criar uma validação para padrão de CPF e de CNPJ, onde não deixaremos CPFs ou CNPJs fugirem do padrão conhecido hoje.
Todas as notificações ocorrerão por e-mail.

Para Empresas:
O JavaBanks possui uma taxa de 0,01% para depósitos e para saques, taxa que o banco conseguirá ver quanto que lhe foi cobrado. Será possível realizar consulta do saldo da Empresa. O Saldo da empresa pode ser negativado.
A Empresa será notificada toda vez que uma transação acontecer.

Para Clientes:
Clientes estarão isentos de todas as taxas e só poderão sacar quando seu saldo for superior a 0 (e somente o valor equivalente ao de seu saldo, impossibilitando-o de negativar a própria conta).
Os clientes são notificados quando fazem transações.

Sobre a API:

Montaremos uma API com a estrutura MVC utilizando o Spring Boot e MySQL. Utilizaremos a codificação com Boas Práticas afim de deixar o código mais acessível e de melhor manutenibilidade à outros programadores.
As dependências que usaremos serão:

- Spring WEB
- Spring Data JPA
- Lombok
- JDBC API
- Java Mail Sender

Arquitetura da API:

Dividiremos a API em 5 pacotes:

Verificate é o pacote responsável por conter classes verificadoras das nossas entradas de *CPF* e *CNPJ;*

Service contém a classe de serviços do Java Mail Sender, onde definimos como enviar o email;

Model é o pacote que contém as entidades que trabalharemos na API, no nosso caso tempos Clientes e Empresas, essas classes contém os atributos usados nas tabelas do Banco de Dados;

Repository é o pacote responsável por ter interfaces dos repositórios das entidades, esses repositórios são responsáveis por métodos de comunicação com o Banco de Dados;

*Controlle*r é o pacote que contém a lógica de programação utilizada para entrar em contato com o Banco de Dados, os controladores definem quando e como entrar em contato com o Banco de Dados.

Configurações necessárias:

Algumas configurações são estritamente necessárias para o bom funcionamento do programa. Vamos começar configurando o application.properties:

```
#Configuração Banco de Dados
spring.datasource.url=jdbc:mysql://localhost:3306/JavaBanks
spring.datasource.username=Usuário do seu MySQL
spring.datasource.password=Senha do seu MySQL
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

#Configuração Adicional JPA
spring.jpa.show-sql=true

#Configuração do JavaMail
spring.mail.host=smtp.gmail.com
spring.mail.port=465
spring.mail.username=Login do Email que será utilizado para enviar as notificações
spring.mail.password=Senha Apps Google Gerada
spring.mail.properties.smtp.auth=true
spring.mail.properties.mail.smtp.ssl.enable=true
```

Configuração do Email no Google:

Após criar uma conta google para a finalidade de ser o remetente dos emails que servirão de notificação, devemos ir às configurações da conta google (imagem 1) e realizar a verificação por duas etapas, o Google não permite uma conexão insegura realizando envio de emails. Próximo passo é ir até a barra de pesquisa de configurações google e pesquisar “Senhas de App” (imagem 2). Faça login e defina um nome para a senha que será criada, após isso, copie e cole o código de 16 caracteres que apareceu e coloque no campo “***spring.datasource.password***”

Imagem 1:

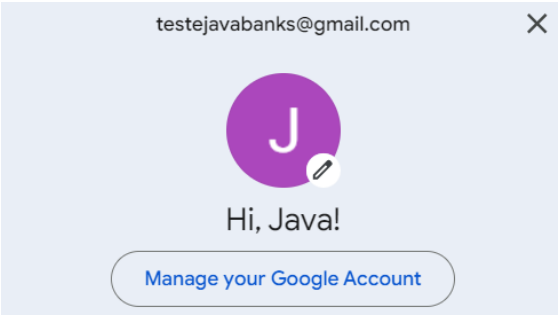
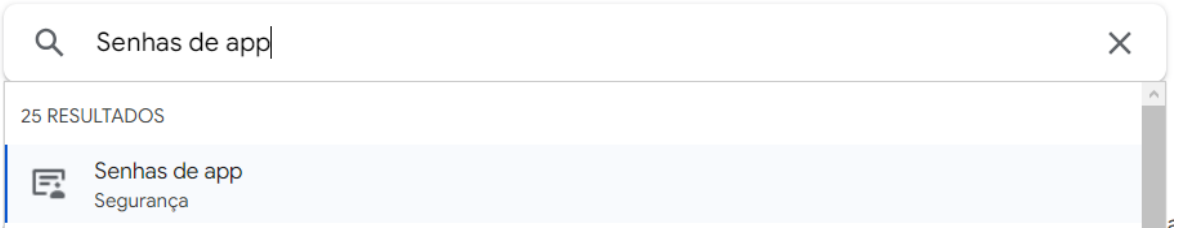


Imagem 2:



Nesse exemplo, estou usando o email testejavabanks@gmail.com para enviar emails

Testando o Programa:

Para testar o programa vamos utilizar o Postman e o Temp Mail (https://temp-mail.org/pt/) utilizando os seguintes endpoints:

1. Para ver os clientes cadastrados:
[GET] localhost:8080/clientes
Retorno esperado:


```
{
    "empresa": 1,
    "idcliente": 1,
    "nome": "Valter",
    "cpf": "00690670079",
    "email": "koxok41268@monutri.com",
    "senha": "1234",
    "saldo": 0.0
}
```


2. Para ver as empresas cadastradas:
[GET]localhost:8080/empresas
Retorno esperado:


```
{
    "idempresa": 1,
    "nome": "exemplo",
    "cnpj": "33542490000107",
    "email": "koxok41268@monutri.com",
    "senha": "exemplo",
    "saldo": 0.0,
    "taxas": 0.0
}
```


3. Para cadastrar clientes precisamos cadastrá-lo em uma empresa existente, para que seu saldo seja atrelado (Body -> Raw -> JSON):
[POST] localhost:8080/clientes


```
{
    "empresa": 1,
     "nome": "Exemplo",
     "cpf": "00690670079",
     "email": "exemplo@exemplo.com",
     "senha": "exemplo"
 }
```


Pode-se tentar cadastrar um saldo diferente de 0, mas há uma lógica para previnir isso, afim de respeitar a regra de negócio
Retorno esperado:

```
1   Cliente Cadastrado com sucesso!
```

4.Para cadastrar empresas (Body -> Raw -> JSON):
[POST] localhost:8080/clientes

```
{
    "nome": "Exemplo",
    "cnpj": "33542490000107",
    "email": "exemplo@exemplo.com",
    "senha": "exemplo"'
}
```


Pode-se tentar cadastrar um saldo ou taxa diferente de 0, mas há uma lógica para previnir isso, afim de respeitar a regra de negócio

Retorno Esperado:

```
Empresa cadastrada com sucesso!
```

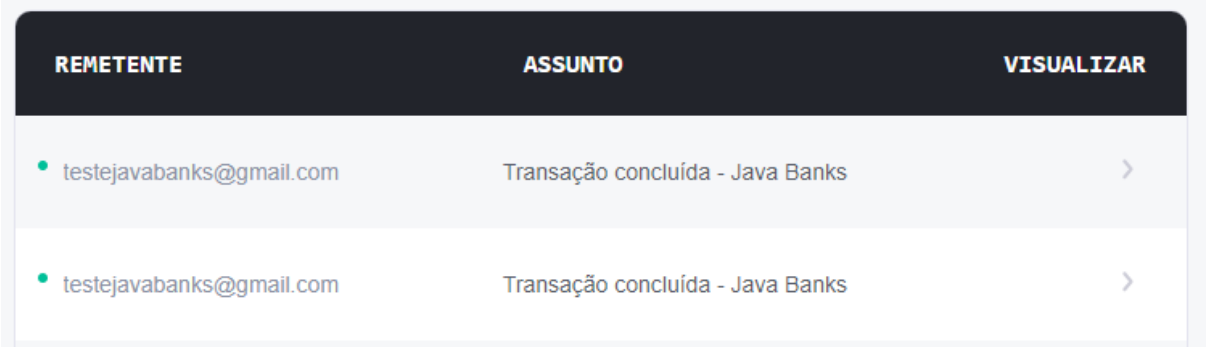
5. Para realizar um depósito, precisamos definir um cliente pelo ID dele e a quantidade pelos endpoints utilizando o PUT:

[PUT]localhost:8080/clientes/depositar/idcliente/quantidade

Usamos idcliente = 1 e quantidade = 10000

Retorno esperado:

```
Depósito de 10000.0. Saldo: 10000.0
```



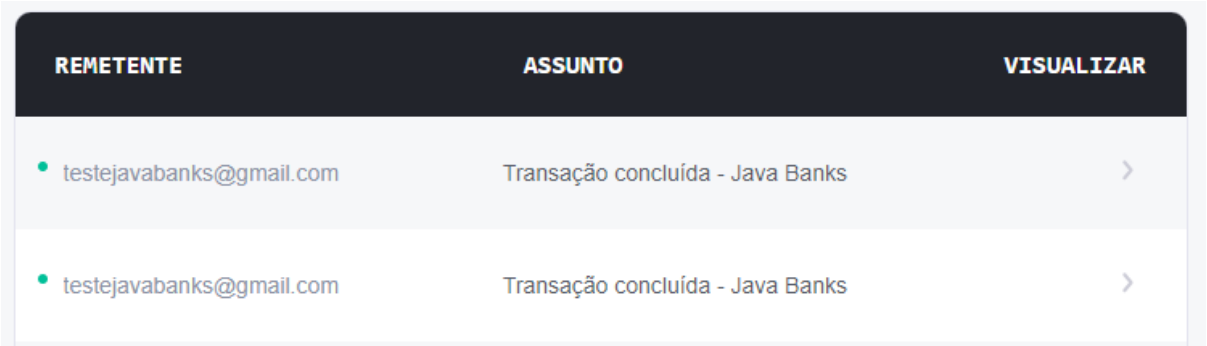
6. Para realizar um saque, precisamos definir um cliente pelo ID dele e a quantidade pelos endpoints utilizando o PUT:

[PUT]localhost:8080/clientes/sacar/idcliente/quantidade

Usamos idcliente = 1 e quantidade = 5000

Retorno esperado:

```
Saque de 5000.0. Novo Saldo: 5000.0
```



Possíveis Erros:

1 - Erro ao cadastrar CNPJ

Retorno obtido:

```
1   CNPJ inválido!
```

Motivo:

CNPJ digitado errado, com pontos, cnpj inválido ou não contém os 14 dígitos necessários

2- Erro nos dados do Body

Retorno obtido:

```
1   Erro nos Dados do Body
```

Motivo:

CPF digitado errado, com pontos, cpf inválido ou não contém os 11 dígitos necessários. Pode ser que tenha atribuído o cliente ao ID de uma empresa inexistente.

3 - Erro no envio de Emails

Retorno Obtido:

```
Operação Realizada. Erro ao enviar Email para o cliente!
```

Motivo:

Configuração do Java Mail Sender incorreta.

4- Saldo Insuficiente:

Retorno obtido:

```
1   Saldo Insulficiente!
```

Motivo:

Saldo do Cliente insuficiente para saque

5 - Erro ao encontrar cliente:

Retorno Obtido:

```
1   Cliente Não Encontrado
```

Motivo:

Id do cliente incorreto.