Sistema

O Sistema nasceu com o intuito inicial de efetuar um cadastro de um Cliente, liberando o valor no limite do crédito e aplicando a taxa de juros de acordo com o risco. Uma funcionalidade simples, porém, sabemos que os sistemas crescem drasticamente ao longo do tempo, e por isso, foi modelado e arquitetado para suportar possíveis mudanças e aumento de consumo.

A arquitetura implantada foi a Spring MVC(Model, View, Controller), onde visa separar as responsabilidades de cada parte do sistema, facilitando manutenção e compreensão do código.

A transmissão de dados é executada por JSON em protocolo HTTP baseado em RestFull contando com Serialização dos Objetos por utilizando a biblioteca do Jackson.

O servidor responsável por deixar a aplicação em pé é o TomCat, leve e atendendo as necessidades até o momento, o servidor fica em standAlone pois estamos utilizando SpringBoot. O SpringBoot visa executar o projeto de forma mais eficaz possível e evitar complicações por diversas anotações que a lib nos oferece.

Em uma maneira geral, o código fica muito mais limpo e facilita a manutenção da configuração do servidor.

O banco de dados é o MySql auxiliado ao Hibernate, responsável por aplicar as persistências na camada de acesso a dados e também efetuar toda a mapeação/validação das entidades diretamente no sistema, dessa forma, não é necessário scripts de tabelas e etc. Através de anotações, a primeira instância que o sistema for iniciado o Hibernate efetuará todos os Creates necessários (Tabelas, Sequences)

(\* Vale ressaltar que o hibernate.ddl-auto-create está como create-drop, portanto, o banco será reiniciado toda vez que a aplicação subir, para não perder os dados, basta mudar para update)

Para a gerenciamento de dependências foi utilizado o Maven, uma ferramenta que poupa muito o trabalho do desenvolvedor, basta anotar o que deseja no pom.xml.

Foi usando também Spring Test, pode-se entendê-lo como uma mapeação de todos os possíveis erros de cada camada da aplicação. Antes de efetuar o deploy, o sistema roda a classe de testes que irá efetuar todas as verificações listadas, evitando drasticamente erros em produção.

Na parte de FrontEnd, foi usado Angular 6/Typescript para fazer a intermediação do Cliente/Servidor e desenvolver a infraestrutura do módulo WEB.

A parte de documentação ficou responsável pela biblioteca springfox, utilizando a ferramenta Swagger2 com o auxílio da swagger-ui. Uma rica biblioteca que gera a documentação de todo o projeto através de anotações, fornecendo diversos tipos de informações com uma interface bem agradável a quem necessita consumir a API.

Para acessa -lá: Basta entrar no caminho:

http://localhost:8080/swagger-ui.html

Plano de Desenvolvimento

Para realizar o projeto, foi utilizado o modelo Desenvolvimento Cascata, mapeado as regras de negócios e as necessidades do “cliente”, análise das possíveis tecnologias que poderiam alcançar o que o sistema necessita, codificação das unidades e entrega do sistema.

Antes de iniciar o desenvolvimento foi pensado em todas as abstrações que poderiam ser reutilizadas futuramente poupando reescrever códigos.

API

Controller:

BaseController : Classe abstrata que contém os métodos padrões de ResponseEntity;

Converter:

Package de classes/interfaces utilitárias necessárias para realizar a conversão de Model -> Json ou Json -> Model

Error:

Package responsável por padronizar as possíveis exceptions do sistema.

Json:

Package com as classes pela entidade e configuração(Jackson/CORS/DefaultServlet) Json.

Persistence:

Package com abstração das entidades do sistema

Após a criação da infra com as abstrações, foram criadas as camadas entidade Client e inicialmente realizado os testes de GET/PUT/DELETE/POST através do postMan até que a view do cliente estivesse pronta.

Assim que foi concluído, colocou-se em prática o esboço já pré-definido do template que atenderia as necessidades e iniciou-se o desenvolvimento do FrontEnd, que também contou com a fase inicial de abstração dos recursos que poderiam ser reutilizados.

No package infra:

Componentes:

Components padrão da aplicação (Header/Footer/Sidebar)

Environment:

Arquivo de configuração com a URL da API.

Service:

Camada final da requisição para o servidor, possui os métodos padrões das entidades e a instancia do HttpClient para se comunicar com a API.

Url:

Classe onde será centralizado os endpoints da API.

Após fechar o fluxo de desenvolvimento, reforçou-se a todas as funcionalidades pretendidas e regras de negócio aplicadas para verificar se alcançamos o objetivo inicial esperado.