FACULDADE DE DESENVOLVIMENTO DO RIO GRANDE DO SUL

DOUGLAS CARDOSO UTZ

LEONARDO NOGUEIRA ROSA

LUCAS ANDRÉ MENDONÇA DO NASCIMENTO

LUIZ HENRIQUE MARQUES

NICOLE CRISTIANO SCHEFFER

PEDRO DOS SANTOS MEDRONHA

YASMIN DA SILVA LUZIA

**Projeto A3 de Modelagem de Software: Sistema de microcrédito**

2022

LISTA DE FIGURAS

Figura 1 - *Product Backlog* …………………………………………………………………. 2

Figura 2 - Modelo de diagrama de casos de uso ……………………………………………. 7

Figura 3 - Modelo de diagrama de classes ………………………………………………….. 8

Figura 4 - Modelo Entidade Relacionamento ………………………………………………. 10

**SUMÁRIO**

[**1 INTRODUÇÃO**](#_dp6yd8rz460s) **1**

[1.1 *USER STORIES*](#_gkn1aswi7ns2) 1

[1.2 *PRODUCT BACKLOG*](#_eu35997mpdyv) 2

[**2 DESENVOLVIMENTO**](#_m9aqzhezgocd) **3**

[2.1 REQUISITOS](#_39vdsp3nss) 3

[2.1.1 REQUISITOS FUNCIONAIS](#_kkgmd8ms4r68) 3

[2.1.2 REQUISITOS NÃO FUNCIONAIS](#_1vu8wn9428zx) 3

[2.1.3 ESPECIFICAÇÃO DE REQUISITOS](#_51ibir9zmbn0) 3

[2.2 DIAGRAMA DE CASOS DE USO](#_78ym50qmq3rq) 6

[2.3 DIAGRAMA DE CLASSES](#_g146fhuid8px) 8

[2.4 MODELO ENTIDADE RELACIONAMENTO (MER)](#_4ad3zto0k9kj) 9

[2.4.1 ENTIDADES](#_u4nl6s85z7ef) 9

[2.4.2 ATRIBUTOS](#_hwb6xgzeo6wy) 9

[2.4.3 RELACIONAMENTOS](#_ruw6wfz0rdm2) 9

[2.4.4 MODELO](#_mif6ieodp0zz) 10

[2.5 *SCRIPTS* DO BANCO DE DADOS](#_6evlmz7lv2dp) 11

[**3 CONCLUSÃO**](#_jtpt3qg2i81e) **15**

[**REFERÊNCIAS BIBLIOGRÁFICAS**](#_hyoat8evdbok) **16**

# **1 INTRODUÇÃO**

A engenharia de *software* tem em todas suas partes grande importância, mas uma das etapas chave é a modelagem. A modelagem de *software* é o primeiro passo dado pelos analistas, desenvolvedores e clientes ou, em muitos casos também, usuários. Nesta etapa são colocadas em foco as razões para criação do *software*, suas características, as necessidades do usuário e a criação do banco de dados.

Segundo BOOCH, RUMBAUGH e JACOBSON (2005), a criação desses modelos é motivada por 4 objetivos principais: eles ajudam a visualizar o sistema como ele é ou como desejamos que ele seja; permitem especificar a estrutura ou o comportamento de um sistema; proporcionam um guia para a construção do sistema; documentam as decisões tomadas no projeto.

Para que a modelagem seja feita corretamente algumas técnicas são implementadas em seu desenvolvimento, dois exemplos de técnicas são as *User Stories* e o *Product Backlog*, estas técnicas fazem parte do *framework* de gerenciamento de projetos chamado Scrum.

## 1.1 *USER STORIES*

*User story* é uma ferramenta criada para auxiliar a definição e organização dos requisitos, de escrever um requisito de projeto de forma enxuta.

Todas as *user stories* incluem frases escritas, uma série de conversas sobre as funcionalidades desejadas.

COHN (2004), propôs um formato a ser preenchido:

Como \_\_\_\_ : Tipo de usuário.

Eu quero \_\_\_\_ : Objetivo, necessidade do cliente.

Para que \_\_\_\_ : Benefícios que o usuário deve ter quando a história for entregue.

Somente as *user stories* não são suficientes para desenvolver uma funcionalidade, mas é uma entrada para melhorar a implementação.

As *user stories* feitas por nossa equipe, baseadas no formato descrito por COHN, acima citado, são as seguintes:

Como administrador, eu quero cadastrar e excluir funcionários, cadastrar e inativar clientes, solicitar extratos bancários, cadastrar transações e emitir boletos para administrar os funcionários da instituição e atender os clientes.

Como funcionário da instituição, eu quero cadastrar e inativar clientes, solicitar extratos bancários, cadastrar transações e emitir boletos para que eu possa administrar e atender os clientes.

O sistema, deve fazer o balanço financeiro mensal, fazer análises de crédito, inativar clientes após muito tempo sem utilizar dos nossos serviços, deletar clientes após um longo tempo de inatividade, possuir uma interface gráfica intuitiva e um backup dos dados para que ele seja acessível, de fácil entendimento, seguro e produtivo.

## 1.2 *PRODUCT BACKLOG*

De acordo com SCHWABER e SUTHERLAND (2020), o *Product Backlog* é uma lista ordenada, e emergente, do que é necessário para criar ou melhorar um produto.

A seguir, o *Product Backlog* produzido por nossa equipe, baseado nas *user stories* e no *framework* Scrum:

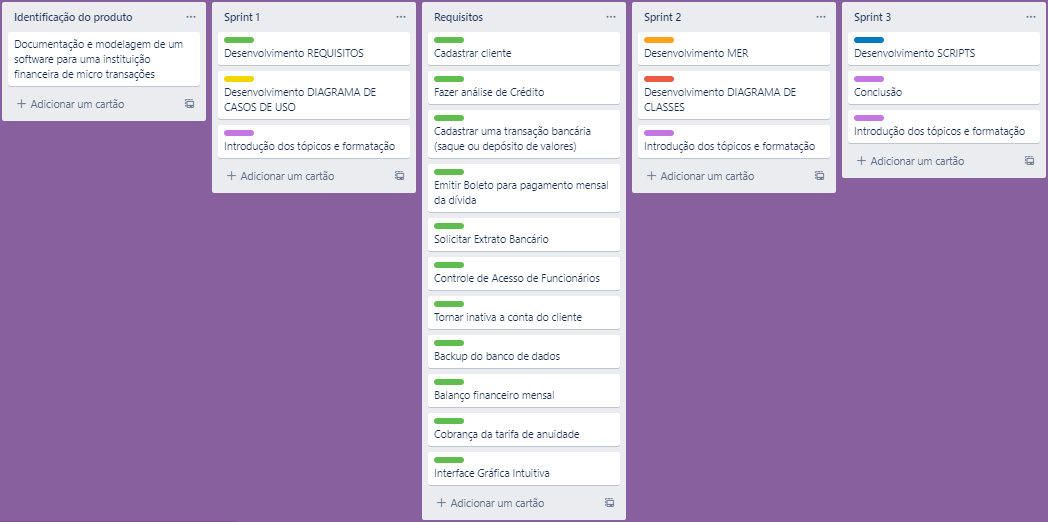


Figura 1 - *Product Backlog*

# 2 DESENVOLVIMENTO

## 2.1 REQUISITOS

Os requisitos de um *software* são importantes para documentá-lo, para mostrar aos clientes o trabalho que será feito e para guiar os responsáveis pelo processo de desenvolvimento. Serão abordados dois tipos de requisitos: os funcionais e não funcionais.

De acordo com a ISO/IEC/IEEC (2010), citadas e descritas por VASQUEZ e SIQUEIRA SIMÕES (2016, pg. 30), requisitos são, de forma resumida: "Uma condição ou capacidade que deve ser alcançada ou possuída por um sistema, produto, serviço, resultado ou componente para satisfazer um contrato, padrão, especificação ou outro documento formalmente imposto."

### 2.1.1 REQUISITOS FUNCIONAIS

Os requisitos funcionais tem como principal característica descrever o que o *software* fará, em um sentido objetivo.

Segundo SOMMERVILLE (2011, pg. 59) "São declarações de serviços que o sistema deve fornecer, de como o sistema deve reagir a entradas específicas e de como o sistema deve se comportar em determinadas situações."

### 2.1.2 REQUISITOS NÃO FUNCIONAIS

Os requisitos não funcionais tem como objetivo regrar um *software*, impor características nele, e definir processos do desenvolvimento.

De acordo com SOMMERVILLE (2011, pg. 59) "São restrições aos serviços ou funções oferecidos pelo sistema. Incluem restrições de *timing*, restrições no processo de desenvolvimento e restrições impostas pelas normas."

### 2.1.3 ESPECIFICAÇÃO DE REQUISITOS

Segundo VAZQUEZ e SIQUEIRA SIMÕES (2016) a especificação de requisitos é um importante instrumento de comunicação entre a equipe que vai desenvolver o software e o cliente, um documento que é instrumento de comunicação entre eles.

Este documento fornece as informações para que a equipe possa entender o que os clientes desejam. É uma composição de vários tipos de documentos.

Modelo de um dos documentos:

1. REQUISITOS FUNCIONAIS

[RF001] Cadastrar cliente.

[RF002] Cadastrar um usuário do sistema.

[RF003] Fazer análise de crédito.

No sistema, de acordo com os dados do cliente.

Se ela possuir dívida ativa na instituição de microcrédito, não será possível requerer crédito.

Baseado na renda mensal e no número de parcelas para o pagamento.

[RF004] Solicitar Extrato Bancário.

[RF005] Cadastrar uma transação bancária.

Saque ou depósito de valores.

Pagamento ou retirada do crédito.

[RF006] Emitir boleto para pagamento mensal da dívida.

[RF007] Inativar um cliente.

Desligá-lo temporariamente, mantendo seu cadastro por algum tempo.

Não poderá ficar inativo se estiver com divida.

[RF008] Após seis meses sem transações muda status para inativo.

[RF009] Após um ano de inatividade o cadastro do cliente é excluído.

[RF010] Cobrança mensal da tarifa de anuidade.

12 reais ao ano.

[RF011] Balanço financeiro mensal.

Para saber o grau de inadimplência.

Porcentagem de clientes que não pagaram e quanto foi emprestado.

2. REQUISITOS NÃO FUNCIONAIS

2.1 Confiabilidade

[RNF001] O sistema deve ser capaz de realizar backups.

2.2 Eficiência

[RNF002] O sistema não pode demorar mais de cinco segundos para retornar o resultado de uma busca ou operação.

2.3 Funcionalidade

[RNF003] Suporte ao cliente.

[RNF004] O sistema deve permitir que somente pessoas que tenham sido autenticadas por algum componente de controle de acesso (senha e registro de funcionário) possam visualizar informações.

[RNF005] O sistema deve ser capaz de interoperar com os sistemas de Interface do Usuário (Java) e banco de dados (MySQL).

2.4 Portabilidade

[RNF006] O sistema deve ser capaz de ser operacionalizado em sistema Unix e Windows.

2.5 Usabilidade

[RNF007] Não conter conteúdo em língua estrangeira, exceto palavras adaptadas à língua portuguesa.

[RNF008] Interface gráfica intuitiva.

## 2.2 DIAGRAMA DE CASOS DE USO

Os diagramas de casos de uso são utilizados para documentar e descrever as funcionalidades e a interação entre um sistema e o usuário. Eles são comumente os primeiros diagramas a serem modelados em uma aplicação, e são compostos por quatro partes:

* Cenário;
* Ator;
* Caso de uso;
* Comunicação.

O cenário descreve a sequência de eventos após a interação do usuário com o sistema, já o ator define o tipo de usuário, enquanto o *use case* descreve a tarefa ou função realizada pelo ator, e por fim, temos a comunicação, que é a parte que liga o ator com o seu caso de uso.

Após a relação entre o ator e suas ações for estabelecida, ela deve ser devidamente documentada.

Segundo JACOBSON (1992), idealizador do conceito de casos de uso, as definições informais são as definições de cada cenário de caso de uso documentado de forma textual. Porém, a visualização do diagrama se torna mais fácil quando este é modelado em forma de imagem.

Utilizando dos padrões da UML, os diagramas de casos de uso são formados por desenhos simples e autoexplicativos, que descrevem a interação do usuário com o sistema e suas funções.

Dessa forma, pode-se visualizar as relações entre os casos de uso e determinar novas funções para o sistema se assim for necessário.

Modelo de diagrama de casos de uso:

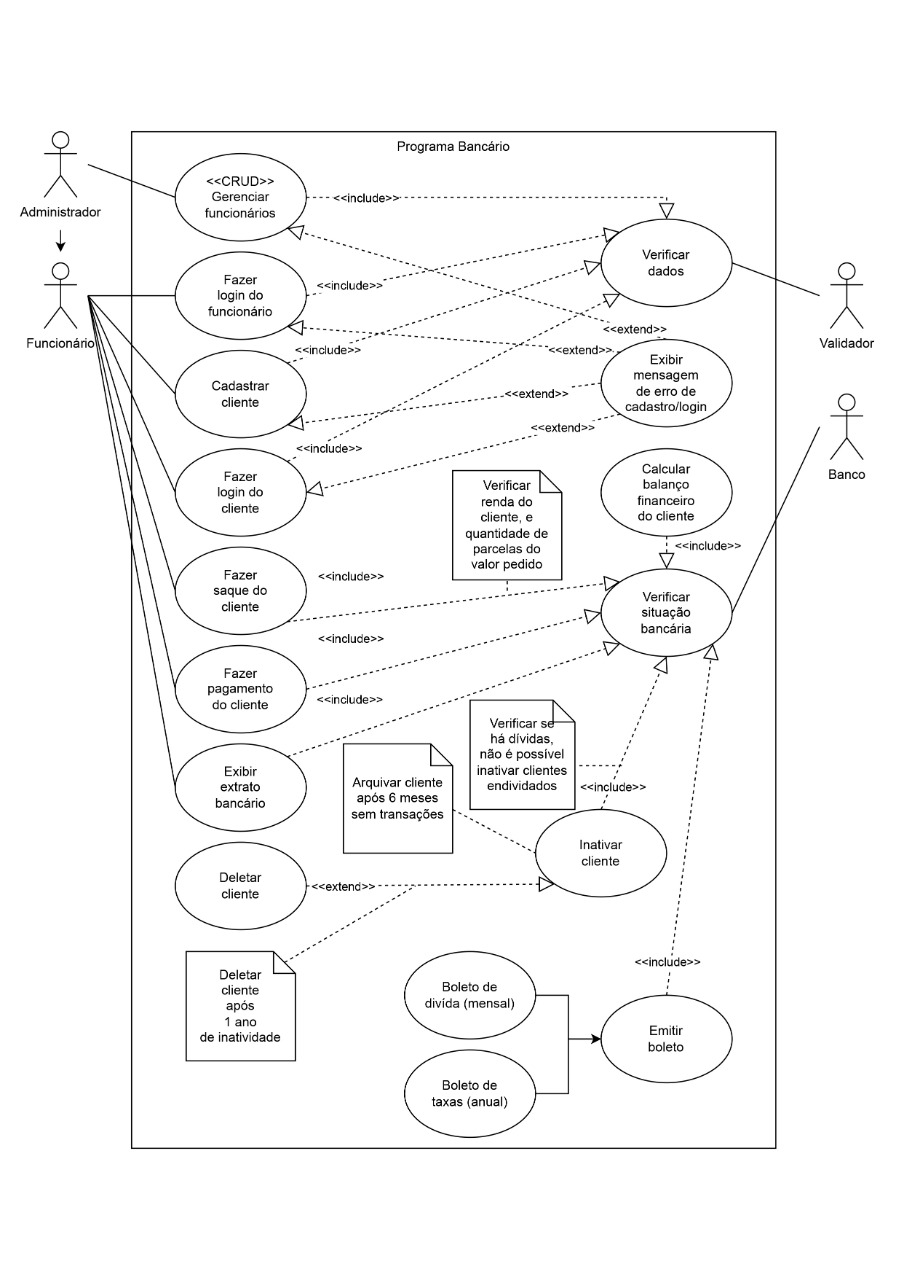


Figura 2 - Modelo de diagrama de casos de uso

## 2.3 DIAGRAMA DE CLASSES

O diagrama de classes descreve a estrutura de um sistema. Mostra partes específicas do sistema, o relacionamento entre os componentes, descreve exatamente como o sistema funciona.

Podemos demonstrar com a modelagem como os objetos se relacionam entre si, o que fazem e que funções eles fornecem.

Segundo BOOCH, RUMBAUGH e JACOBSON (2012) este tipo de modelagem padroniza o sistema facilitando a compreensão por ser uma linguagem muito expressiva e de fácil acesso a todas as visões necessárias ao desenvolvimento e implantação do sistema.

Modelo de diagrama de classes:

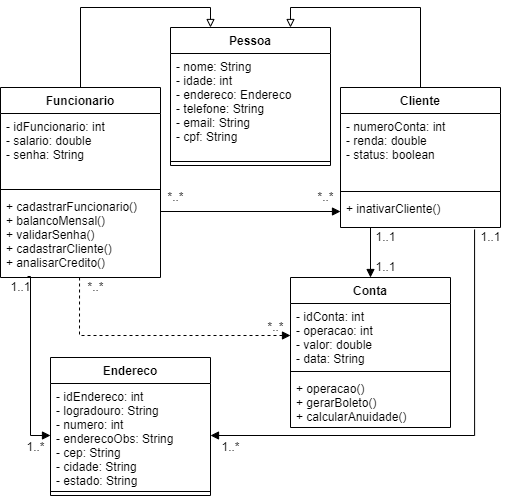


Figura 3 - Modelo de diagrama de classes

## 2.4 MODELO ENTIDADE RELACIONAMENTO (MER)

Segundo DevMedia (2014), o Modelo Entidade Relacionamento (MER) é um modelo conceitual utilizado para descrever os objetos (entidades) envolvidos em um domínio de negócios, com suas características (atributos) e como elas se relacionam entre si (relacionamentos).

### 2.4.1 ENTIDADES

Como descrito por PUGA, FRANÇA e GOYA (2013), uma entidade representa uma categoria de elementos relevantes para um negócio, podendo ser, por exemplo, clientes, fornecedores, vendas, contratos, etc.

Sendo assim, uma entidade representa um conjunto de dados que precisam ser armazenados e que serão utilizados em várias aplicações e/ou programas que descrevem o funcionamento do negócio.

### 2.4.2 ATRIBUTOS

Os atributos são as características que as entidades possuem, e, de acordo com PUGA, FRANÇA e GOYA (2013), podem ser definidas como:

* Informação associada a uma entidade;
* Característica ou propriedade de uma entidade ou relacionamento;
* Descrição, identificação, qualificação ou quantificação de uma entidade.

### 2.4.3 RELACIONAMENTOS

Em um MER, segundo PUGA, FRANÇA e GOYA (2013), o relacionamento estabelece uma relação entre diferentes entidades, sendo esses relacionamentos classificados de três formas distintas, sendo elas: um para um, um para muitos, e muitos para muitos.

No relacionamento um para um, cada uma das duas entidades envolvidas referenciam obrigatoriamente apenas uma unidade da outra.

Por outro lado, no relacionamento um para muitos, uma das entidades envolvidas pode referenciar várias unidades da outra, porém, do outro lado cada uma das várias unidades referenciadas só pode estar ligada uma unidade da outra entidade.

Por fim, a última classificação de relacionamento é a de muitos para muitos, determinando que cada entidade, de ambos os lados, pode referenciar múltiplas unidades da outra.

### 2.4.4 MODELO

Modelo Entidade Relacionamento:

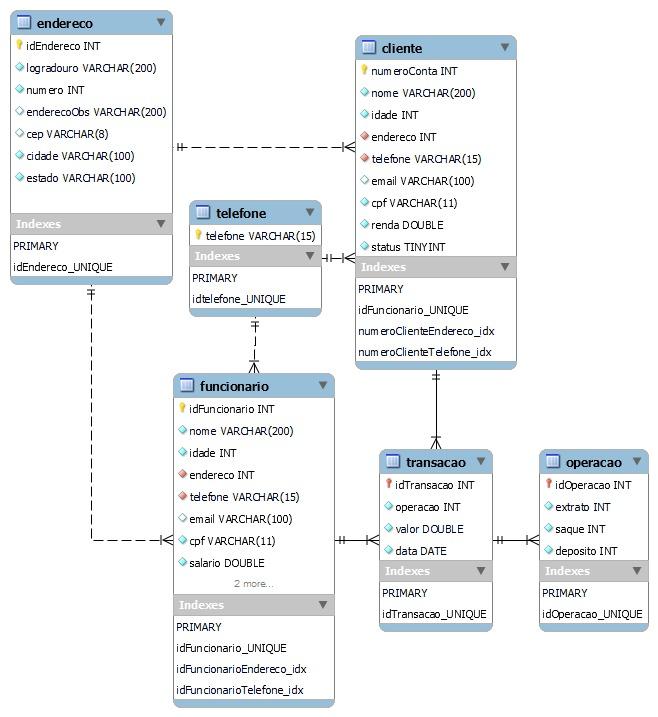


Figura 4 - Modelo Entidade Relacionamento

## 2.5 *SCRIPTS* DO BANCO DE DADOS

*Scripts* utilizados na criação do banco de dados:

CREATE SCHEMA IF NOT EXISTS `db\_sarancredi` DEFAULT CHARACTER SET utf8;

USE `db\_sarancredi` ;

DROP TABLE IF EXISTS `db\_sarancredi`.`telefone` ;

CREATE TABLE IF NOT EXISTS `db\_sarancredi`.`telefone` (

`telefone` VARCHAR(15) NOT NULL,

PRIMARY KEY (`telefone`),

UNIQUE INDEX `idtelefone\_UNIQUE` (`telefone` ASC) VISIBLE

);

DROP TABLE IF EXISTS `db\_sarancredi`.`endereco` ;

CREATE TABLE IF NOT EXISTS `db\_sarancredi`.`endereco` (

`idEndereco` INT NOT NULL AUTO\_INCREMENT,

`logradouro` VARCHAR(200) NOT NULL,

`numero` INT NOT NULL,

`enderecoObs` VARCHAR(200) NULL,

`cep` VARCHAR(8) NULL,

`cidade` VARCHAR(100) NOT NULL,

`estado` VARCHAR(100) NOT NULL,

PRIMARY KEY (`idEndereco`),

UNIQUE INDEX `idEndereco\_UNIQUE` (`idEndereco` ASC) VISIBLE

);

DROP TABLE IF EXISTS `db\_sarancredi`.`cliente` ;

CREATE TABLE IF NOT EXISTS `db\_sarancredi`.`cliente` (

`numeroConta` INT NOT NULL AUTO\_INCREMENT,

`nome` VARCHAR(200) NOT NULL,

`idade` INT NOT NULL,

`endereco` INT NOT NULL,

`telefone` VARCHAR(15) NOT NULL,

`email` VARCHAR(100) NULL,

`cpf` VARCHAR(11) NOT NULL,

`renda` DOUBLE NOT NULL,

`status` TINYINT NOT NULL,

PRIMARY KEY (`numeroConta`),

UNIQUE INDEX `idFuncionario\_UNIQUE` (`numeroConta` ASC) VISIBLE,

INDEX `numeroClienteEndereco\_idx` (`endereco` ASC) VISIBLE,

INDEX `numeroClienteTelefone\_idx` (`telefone` ASC) VISIBLE,

CONSTRAINT `numeroClienteEndereco` FOREIGN KEY (`endereco`) REFERENCES `db\_sarancredi`.`endereco` (`idEndereco`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `numeroClienteTelefone` FOREIGN KEY (`telefone`) REFERENCES `db\_sarancredi`.`telefone` (`telefone`) ON DELETE NO ACTION ON UPDATE NO ACTION

);

DROP TABLE IF EXISTS `db\_sarancredi`.`funcionario` ;

CREATE TABLE IF NOT EXISTS `db\_sarancredi`.`funcionario` (

`idFuncionario` INT NOT NULL AUTO\_INCREMENT,

`nome` VARCHAR(200) NOT NULL,

`idade` INT NOT NULL,

`endereco` INT NOT NULL,

`telefone` VARCHAR(15) NOT NULL,

`email` VARCHAR(100) NULL,

`cpf` VARCHAR(11) NOT NULL,

`salario` DOUBLE NOT NULL,

`senha` VARCHAR(100) NOT NULL,

PRIMARY KEY (`idFuncionario`),

UNIQUE INDEX `idFuncionario\_UNIQUE` (`idFuncionario` ASC) VISIBLE,

INDEX `idFuncionarioEndereco\_idx` (`endereco` ASC) VISIBLE,

INDEX `idFuncionarioTelefone\_idx` (`telefone` ASC) VISIBLE,

CONSTRAINT `idFuncionarioEndereco` FOREIGN KEY (`endereco`) REFERENCES `db\_sarancredi`.`endereco` (`idEndereco`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `idFuncionarioTelefone` FOREIGN KEY (`telefone`) REFERENCES `db\_sarancredi`.`telefone` (`telefone`) ON DELETE NO ACTION ON UPDATE NO ACTION

);

DROP TABLE IF EXISTS `db\_sarancredi`.`transacao` ;

CREATE TABLE IF NOT EXISTS `db\_sarancredi`.`transacao` (

`idTransacao` INT NOT NULL,

`operacao` INT NOT NULL,

`valor` DOUBLE NOT NULL,

`data` DATE NOT NULL,

PRIMARY KEY (`idTransacao`),

UNIQUE INDEX `idTransacao\_UNIQUE` (`idTransacao` ASC) VISIBLE,

CONSTRAINT `idFuncionarioTransacao` FOREIGN KEY (`idTransacao`) REFERENCES `db\_sarancredi`.`funcionario` (`idFuncionario`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `numeroContaTransacao` FOREIGN KEY (`idTransacao`) REFERENCES `db\_sarancredi`.`cliente` (`numeroConta`) ON DELETE NO ACTION ON UPDATE NO ACTION

);

DROP TABLE IF EXISTS `db\_sarancredi`.`operacao` ;

CREATE TABLE IF NOT EXISTS `db\_sarancredi`.`operacao` (

`idOperacao` INT NOT NULL AUTO\_INCREMENT,

`extrato` INT NOT NULL DEFAULT 1,

`saque` INT NOT NULL DEFAULT 2,

`deposito` INT NOT NULL DEFAULT 3,

PRIMARY KEY (`idOperacao`),

UNIQUE INDEX `idOperacao\_UNIQUE` (`idOperacao` ASC) VISIBLE,

CONSTRAINT `Operacao` FOREIGN KEY (`idOperacao`) REFERENCES `db\_sarancredi`.`transacao` (`idTransacao`) ON DELETE NO ACTION ON UPDATE NO ACTION

);

# 3 CONCLUSÃO

Foram trabalhadas várias etapas na documentação, como o levantamento e a especificação de requisitos, contendo uma breve explicação sobre cada componente. Também, alguns modelos de diagramas foram desenvolvidos, para uma documentação mais precisa, sendo eles sobre: casos de uso, classes, e o Modelo Entidade Relacionamento.

Além disso, foi desenvolvido um projeto de banco de dados para o sistema de microcrédito documentado. Os *scripts* do banco de dados citado estão presentes no documento, para maior compreensão técnica do projeto.

Em relação aos requisitos, sempre há certas dificuldades na hora de elaborá-los. Nosso projeto, por mais que tenha considerado o levantamento de requisitos, verificando e validando os serviços e restrições do sistema, não teve as possíveis dificuldades típicas da especificação de requisitos, como a falta de conhecimento sobre o domínio do negócio, a falta de comunicação entre os envolvidos, ou a gestão precária da mudança e evolução de requisitos.

Por fim, após a confecção do documento, foi possível compreender a importância de um documento oficial entre cliente e fornecedor, pois ele permite compreender a visão dos envolvidos sobre o serviço a ser desenvolvido, possibilitando uma maior compreensão dos diferentes pontos de vista sobre um mesmo sistema. Além disso, ele também serve como um norte para as equipes de desenvolvimento, pois assim é possível manter uma maneira organizada e contínua de gerência.

# REFERÊNCIAS BIBLIOGRÁFICAS

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML**: guia do usuário. 2ª ed. Rio de Janeiro: Campus, 2005.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML**: Guia do Usuário. Rio de Janeiro: Elsevier Editora Ltda, 2012.

CUNHA, Mike. **User Stories Applied**: For Agile Software Development. Boston: Addison-Wesley Professional, 2004.

JACOBSON, Ivar. **Object-Oriented Software Engineering**: A Use Case Driven Approach. Michigan: ACM Press, 1992.

MER e DER: Modelagem de Bancos de Dados. DevMedia, 2014. Disponível em: https://www.devmedia.com.br/mer-e-der-modelagem-de-bancos-de-dados/14332. Acesso em: 6 de junho de 2022.

PUGA, Sandra; FRANÇA, Edson; GOYA, Milton. **Banco de dados**: implementação em SQL, PL/SQL e Oracle 11G. São Paulo: Pearson, 2013.

SCHWABER, Ken.; SUTHERLAND, Jeff. The 2020 Scrum Guide TM. SCRUM Guides, 2020. Disponível em: https://scrumguides.org/scrum-guide.html#purpose-of-the-scrum-guide. Acesso em: 23 de maio de 2022.

SOMMERVILLE, Ian. **Engenharia de Software**. 9ª ed. São Paulo: Pearson Prentice Hall, 2011.

VAZQUEZ, Carlos Eduardo; SIQUEIRA SIMÕES, Guilherme. **Engenharia de Requisitos**: software orientado ao negócio. Rio de Janeiro: BRASPORT Livros e Multimídia Ltda., 2016.