CENTRO DE EDUCAÇÃO PROFISSIONAL DE TIMBÓ

CURSO TÉCNICO EM INFORMÁTICA

SOFTWARE PARA GERENCIAMENTO DE EMPRESAS DE TERRAPLENAGEM

THIAGO METZNER GONÇALVES

TIMBÓ
2022

SOFTWARE PARA GERENCIAMENTO DE EMPRESA DE TERRAPLENAGEM

Por

THIAGO METZNER GONÇALVES

Trabalho aprovado em sua forma final pelo Curso Técnico de Informática do Centro de Educação Profissional de Timbó - CEDUP

_____

Prof. Roberto Luiz Debarba – Orientador

_____

Prof. Edésio Marcos Slomp – Coorientador

_____

Prof. Rutineia Luciene Bell - Coordenadora

Timbó, 18 de julho de 2022

AGRADECIMENTOS

Agradeço a todos os professores que me ajudaram e apoiaram ao longo da trajetória, em especial ao professor Roberto pela dedicação que teve para me direcionar ao caminho correto até a entrega do projeto.

Agradeço também à coordenação do curso pelo esforço que teve para nos proporcionar o melhor ensinamento.

E quero fazer um agradecimento especial também à minha família, pelo apoio e incentivo que me deram desde o início do projeto.

RESUMO

Na atualidade que estamos vivendo, as tecnologias estão em uma constante e rápida evolução, costumes antigos ficaram extremamente ultrapassados e com isso, muitas vezes as empresas e comércios não conseguem suprir as demandas que recebem por conta da demora de suas operações. Ainda é muito comum encontrar pequenas empresas, que realizam seus controles de dados em planilhas e ou, anotações físicas, em cadernos, agendas. O que não é nada eficiente quando se tem oportunidade de possuir um software de gerenciamento que consegue guardar esses registros de forma quase que instantânea e mais rápido ainda, a consulta dos registros já armazenados. O protótipo apresentado vem com o intuito de agilizar o gerenciamento de uma empresa de terraplenagem a partir de um sistema que é responsável pelo controle de dados e registros.

Palavras-chave: Software para terraplenagem, software de gerenciamento, controle de dados.

SUMÁRIO

1 INTRODUÇÃO 6
1.1 OBJETIVOS DO TRABALHO 7
1.1.1 Objetivo Geral 7
1.2 ESTRUTURA DO TRABALHO 8
2 FUNDAMENTAÇÃO TEÓRICA 9
2.1 FRONTEND 9
2.2 BACKEND 9
2.3 LINGUAGEM DE PROGRAMAÇÃO 9
2.4 HTML 10
2.5 CSS 10
2.6 TAILWIND CSS 11
2.7 ECMA SCRIPT 11
2.8 REACT 12
2.9 NODE 12
2.10 BANCO DE DADOS 12
2.11 POSTGRES SQL 13
2.12EXPRESS 13
2.13PRISMA 13
2.14NEXT.JS 14
2.15TYPESCRIPT 14
2.16SASS CSS 15
3 DESENVOLVIMENTO 16
3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO 16
3.1.1 Requisitos Funcionais 16
3.1.2 Requisitos Não Funcionais 17
3.2 ESPECIFICAÇÃO 17
3.2.1 Diagrama de Casos de Uso 17
3.2.2 Notação de casos de uso 18
3.2.3 Diagrama de Atividades 20
3.2.4 Banco de dados – Modelo Conceitual 21
3.2.5 Banco de dados – Modelo Lógico 21

3.2.6 Diagrama de classes 22
3.3 IMPLEMENTAÇÃO 23
3.3.1 Técnicas e ferramentas utilizadas 24
3.3.1.1 Beekeeper Studio Ultimate 24
3.3.1.2 Insomnia 25
3.3.1.2 Git e GitHub 25
3.3.2 Operacionalidade da implementação 26
3.3.2.1 Autenticação de usuário no Frontend. 27
3.3.2.2 Criação das tabelas do banco de dados 28
3.3.2.3 Autenticação do usuário no Backend. 29
3.3.2.4 Tela de cadastro de usuário 30
3.4 RESULTADOS E DISCUSSÃO 31
4 CONCLUSÕES 33
4.1 EXTENSÕES 33
5 REFERENCIAS 35

1 INTRODUÇÃO

A ideia de um software utilizado para gerenciamento de uma terraplenagem surgiu pois a empresa Nico Terraplenagem possuía seu controle de dados interno baseado em planilhas, estas demoram certo tempo para serem preenchidas e após algum tempo inserindo dados o resultado final se dava por uma planilha confusa em que os dados demoravam muito tempo para serem consultados, gerando desconforto e impaciência do cliente e usuário, sendo que esses dados muitas vezes precisam ser consultados em instantes.

Uma resolução que já foi cogitada a parcialmente adotada pela empresa foi a criação de mais planilhas, que dividiam os gastos e lucros por mês, porém, mesmo assim, teriam de consultar as vezes mais de uma planilha para conseguir determinado resultado. Essas características acarretam em um consumo de tempo maior causando uma gestão financeira menos adequada.

> Devido à grande competição que as empresas dos mais diversos ramos e tamanhos travam na busca por desenvolvimento e entrega de valor a seus clientes, uma gestão financeira adequada se torna fundamental como ferramenta de auxílio para que as organizações obtenham uma colocação satisfatória nos mercados em que atuam, colaborando com o planejamento, resolução de problemas e melhoramento de resultados (BERTOLETTI, 2015).

De primeira mão o sistema tem como foco principal manter a gestão financeira da empresa de acordo com os vencimentos das parcelas à pagar e as contas que a empresa tem a receber de determinados clientes, porém, o software também tem a funcionalidade de realizar cadastros. Terá um sistema de cobrança que fará com que o usuário tenha um contato com seu cliente de forma mais rápida quando observar que entraram parcelas em atraso, por exemplo.

Para o desenvolvimento deste projeto foi utilizada como técnica de pesquisa a “entrevista dirigida”, onde, desta forma, a participação da empresa na qual a criação dele foi destinada será de grande importância, possibilitando vasta base de conhecimentos.

Serão utilizadas diversas ferramentas de desenvolvimento, tendo como linguagem principal o JavaScript juntamente com algumas bibliotecas como o React e o Bootstrap. Seu Backend foi escrito em Node que também foi o responsável por realizar a comunicação com o banco de dados.

1.1 OBJETIVOS DO TRABALHO

O objetivo principal do protótipo é conter um módulo de gerenciamento financeiro bem desenvolvido juntamente com alguns cadastros e subcadastros básicos. Deve ser utilizado ferramentas e linguagens novas. E ser leve para poder rodar em máquinas com uma capacidade de processamento reduzida.

1.1.1 Objetivo Geral

Desenvolver um software WEB para gerenciamento de terraplanagem, tendo como principal foco o controle financeiro e os cadastros.

1.1.2 Objetivos Específicos

- Esquematizar a gestão financeira com foco na facilidade deixando-a de forma intuitiva, utilizando elementos que facilitam a interação.
- Possuir níveis de acesso, dentre eles, usuário e supervisor.
- Integrar com o banco de dados;
- Hospedar software tendo como servidor o próprio computador do cliente ou o computador principal da empresa. Os terminais devem realizar a comunicação com o sistema utilizando o endereço de IP disponibilizado na rede de internet local;
- Empregar um sistema de envio de e-mail de cobrança automático para os clientes com duplicatas em atraso.
- Complementar o software com temas de cores, Dark-Mode e Light-Mode.

1.2 ESTRUTURA DO TRABALHO

O primeiro capítulo se trata da introdução do trabalho, onde serão apresentadas as intenções da aplicação e os objetivos que foram atendidos com o desenvolvimento do software.

O segundo capítulo tem a função de retratar a fundamentação do protótipo, descrevendo as linguagens e ferramentas que foram utilizadas no desenvolvimento e implementação do software

O terceiro capítulo descreve como foi realizado o desenvolvimento do software, apresentando a diagramação, especificações e análises realizadas para atender os requisitos anteriormente descritos. Também estão presentes detalhes do desenvolvimento e da implementação do protótipo.

Por fim, no quarto e último capítulo, são descritas as conclusões acerca do desenvolvimento do trabalho em modo geral, descrevendo a impressão que obtive sobre o trabalho como um todo, descrevo as dificuldades encontradas ao longo do desenvolvimento e também algumas ideias que tive ao longo da implementação.

No capítulo quatro também pode ser encontrada algumas ideias para projetos futuros que algumas extensões do protótipo que estão em desenvolvimento.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 FRONTEND

É muito conhecida por ser a parte visual de um sistema web, estando diretamente ligada com a experiência que o usuário terá ao utilizar as funcionalidades do site. É no Frontend que o desenvolvedor implementa a diversificação visual tornando a experiência do usuário algo único.

Kalbach (2009, p. 206) comenta que “a fundação de todos os sites web, HTML (e seu primo mais restrito XHTML) são as linguagens de programação básicas para criar os mecanismos de navegação e a estrutura de um site web. O W3C é o órgão de padronização por manter essas linguagens”. O Frontend, portanto, merece muita atenção do desenvolvedor web, pois é a parte visual que irá interagir com o usuário.

2.2 BACKEND

São as tecnologias que fazem parte do servidor, ou seja, mantém um site ou sistema funcionando, no Backend está localizado a maior parte de segurança de um sistema, criptografando senhas e dificultando o acesso de invasores, Backend é responsável pela comunicação com o banco de dados em geral, gerenciando as requisições do Frontend.

“Uma aplicação ou programa Backend serve indiretamente como apoio aos serviços de Frontend, geralmente por estarem mais próximos ao recurso exigido ou por terem a capacidade de se comunicar com o recurso necessário” (FERREIRA, PAIVA, 2012, p.14). Sendo assim, o Backend e o Frontend são as duas partes que formam um website, podendo ser desenvolvidos por profissionais especializados em cada uma das áreas.

2.3 LINGUAGEM DE PROGRAMAÇÃO

As linguagens de programação como o próprio nome se refere, são linguagens,

assim como o inglês, português, espanhol, foram criadas com o intuito de facilitar a comunicação com alguém ou algo, neste caso, foram desenvolvidas para um programador conseguir se comunicar com um computador.

Hoje em dia, existem linguagens das mais diversas, com sintaxes diferentes, cada uma desenvolvida para se comportar melhor em suas determinadas áreas. Essas linguagens têm seguranças e dificuldades de aprendizados diferentes, algumas são extremamente complexas, criadas para chegar mais próximo ao binário, que é a linguagem em que o computador realmente entende, essas linguagens são consideradas linguagens de “baixo nível”, Assembly é um exemplo destas.

Existem também as linguagens de “alto nível”, que são aquelas linguagens que são mais parecidas com a linguagem do ser humano, a maioria delas é inspirada no inglês, possuindo diversos elementos nativos desta, um exemplo é o Java Script.

2.4 HTML

O HTML (Hypertext Markup Language) surgiu em 1991 e desde então recebeu diversas atualizações. Sua responsabilidade principal é demarcar a estrutura de uma página da web. Essa estrutura do HTML é formada por um conjunto de elementos, ou seja, os hipertextos, que se conectam entre si formando a página.

Os elementos HTML ou também chamados de tags HTML, são utilizados para informar ao navegador que tipo de estrutura é essa que está sendo construída, podendo ser títulos, parágrafos, imagens, links, entre outros.

2.5 CSS

CSS (Cascading Style Sheets) são folhas de estilo em cascata. O CSS é uma linguagem que complementa e formata o HTML (Hypertext Markup Linguagem a Linguagem de Marcação de Hipertexto) organizando melhor as linhas e adicionando novas possibilidades ao código. Com ele, você pode modificar praticamente tudo dentro do seu layout (como as cores, background, características de fontes, margens, preenchimentos, posição, até a própria estrutura do site com a propriedade float).

O CSS ajuda a manter as informações de um documento separadas dos detalhes de como exibi-la. Esses detalhes de como exibir o documento são conhecidos como estilo. O desenvolvedor mantém o estilo separado do conteúdo e assim, pode evitar duplicação, tornar a manutenção mais fácil e utilizar o conteúdo com diferentes estilos para diferentes propósitos.

2.6 TAILWIND CSS

Tendo sua origem no CSS o TailWind surgiu para facilitar a criação dos layouts web, priorizando a otimização do tempo, veio para ser um diferencial quando se trata de uma interface de simples criação, mas que atrai o usuário com suas cores e elementos únicos.

Assim como o Bootstrap e o React-Bootstrap, o TailWind é uma biblioteca que basicamente contém componentes prontos, sendo necessária apenas a chamada destes componentes que já são estilizados. O CSS ainda pode ser utilizado para fazer ajustes a acrescentar alguns detalhes que o componente padrão do TailWind não recebe.

2.7 ECMA SCRIPT

Teve seu início nos anos 90 num tempo de revolução, neste período a internet começou a tomar proporções gigantescas e era necessário que os sites existentes recebessem adaptações. Na época, os poucos browsers que existiam eram estáticos e muito trabalhosos.

O Ecma Script tentou por diversas vezes mudar seu nome para Java Script, o que não foi possível pois a Sun Microsystems (hoje Oracle) era detentora desse nome e o mesmo não poderia ser utilizado, porém, até hoje, os desenvolvedores carinhosamente chamam o Ecma Script de Java Script.

Java Script hoje é a linguagem de programação mais popular no desenvolvimento Web. Suportada por todos os navegadores, a linguagem é responsável por praticamente qualquer tipo de dinamismo que queiramos em nossas páginas.

É uma linguagem fortemente tipada, e é considerada uma linguagem de alto nível.

2.8 REACT

React é uma biblioteca do Java Script que tem como intuito a valorização da construção de interfaces, aplica-se com um notável dinamismo nas interações com o usuário. Surgiu inicialmente como a ideia de algo inovador, mas ao mesmo tempo de fácil aprendizado e que utilizava resquícios da linguagem no qual foi inspirado(Java Script).

Pelo fato do desenvolvimento em React ser mais facilitado e pelo dinamismo que a biblioteca possui, novos programadores passaram a pesquisar e se aprofundar nela, unindo com sua versatilidade, foi o que a tornou uma das bibliotecas mais utilizadas quando se trata tanto de sites mais simples como também de sistemas web mais complexos.

2.9 NODE

O Node.js é um ambiente de execução Javascript, linguagem padrão de manipulação de páginas HTML, criada em 1995. Anteriormente, foi usada para o desenvolvimento Cliente-Side, mas com a evolução da Internet, também passou a ser aplicada como Server-Side.

O ambiente do Node.js é utilizado por gigantes do mercado de tecnologia, como Netflix e LinkedIn. Sua escolha é embasada em uma característica muito peculiar: sua alta escalabilidade**,** uma vez que a execução single-thread permite criar um Event Loop com requisições que não demandam output.

Essa arquitetura, somada a outras vantagens, como o baixo custo e a flexibilidade, permitem que o Node.js embase, principalmente, aplicações multidirecionais com comunicação e troca de dados em tempo real.

2.10 BANCO DE DADOS

Inicialmente o “banco de dados“ era encontrado em agendas, livros, cadernos, onde as pessoas faziam suas anotações para futuramente esses registros serem acessados e suas informações serem relembradas por quem precisava dessa consulta. O conceito de

banco de dados surgiu com o intuito de agilizar a consulta de dados e otimizar o tempo no qual os usuários faziam suas consultas.

2.11 POSTGRES SQL

É um dos bancos de dados mais utilizados da indústria. Banco de dados 100% comunitário teve sua origem no propósito de ser disponibilizado de forma gratuita.

Pelo fato de ser “open source”, suas alterações e funcionalidades de grande mão tem origem na comunidade, que pode reportar bugs ou realizar a tratativa destes para o benefício de todos.

O Postgres é recomendado para aplicações que detenham maior demanda de operações SQL, devido a sua capacidade de processamento.

2.12 EXPRESS

O Express.js é um Framework rápido e um dos mais utilizados em conjunto com o Node.js, facilitando no desenvolvimento de aplicações Backend e até, em conjunto com sistemas de templates, aplicações Full-Stack.

Escrito em JavaScript, o Express.js é utilizado por diversas empresas ao redor do mundo, dentre elas a Fox Sports, PayPal, IBM, Uber, entre outras.

Muito popular tanto em grandes empresas quanto na comunidade, o Express facilita a criação de aplicações utilizando o Node em conjunto com o JavaScript, tornando este ecossistema ainda mais poderoso.

2.13 PRISMA

Não há exatamente uma data de lançamento da ferramenta, mas os primeiros artigos que se tem registro no GitHub foram lançados em 2017. O prisma nasceu com o intuito de facilitar a comunicação e o controle das informações técnicas do banco de dados.

Na descrição oficial da ferramenta está descrito que o prisma é algum tipo de ORM fundamentalmente diferente dos modelos tradicionais que eram aplicados anteriormente. Uma alternativa para outras ORMs (Object-Relational Mapping, é uma técnica para aproximar o desenvolvimento de aplicações ao do banco de dados relacional), como TypeORM e Sequelize. Hoje o prisma se encontra em sua terceira etapa de desenvolvimento que é chamado de Prisma 3.

2.14 NEXT.JS

É um framework do React com foco em produção e eficiência criado e mantido pela equipe da Vercel, ele busca reunir diversas funcionalidades como renderização hibrida e estática de conteúdo, suporte a TypeScript, pre-fetching, sistema de rotas, pacotes de funcionalidades e diversos plugins e exemplos para acelerar seu desenvolvimento fornecendo uma estrutura completa para você iniciar seu projeto. Com todas essas facilidades pré-configuradas ele se assimila a um create-react-app onde você inicia o projeto muito rápido e sem preocupação com configurações de webpack, estruturas de pastas, configuração de rotas e etc. Algumas características do Next.js são:

- Hibrido SSG e SSR: Torna possível renderizar sua página durante a build (Static Site Generation) ou em cada request (Server-side Rendering) no mesmo projeto.
- Hot Code Reloading: Qualquer alteração feita em seu código durante o desenvolvimento será refletida na aplicação local em tempo real, atualizando de forma automática.
- Otimização de Imagem: Componente nativo do Next para otimização de suas imagens com redimensionamento, lazyload, imagens em formato moderno e de fácil implementação.

2.15 TYPESCRIPT

É um framework que busca trazer maneiras mais eficientes de escrever os códigos Java Script. Começou a ser desenvolvido em 2012 pela Microsoft e deste então sofreu diversas alterações. Por ser um framework mais “jovem” é encontrado com mais frequência em aplicações em React.

A principal vantagem do Type Script em relação ao JavaScript “tradicional” é adicionar recursos importantes e úteis para a construção de projetos em larga escala, como tipagem estática, forte e automática, orientação a objetos e a possibilidade de descobrir e corrigir erros em tempo real durante o desenvolvimento.

2.16 SASS CSS

O SASS é uma linguagem de extensão do CSS, a sigla significa “Syntactically Awesome Style Sheets” traduzindo ao pé da letra, folhas de estilo com uma sintaxe incrível. A sua ideia é adicionar recursos especiais como variáveis, mixins, funções e operações.

Foi criado para facilitar o desenvolvimento em CSS, permitindo criar funções e variáveis dentro do próprio arquivo que está sendo ditada. Atualmente não recebe muita fama, mas com certeza é uma boa opção para quem quer fugir do CSS padrão.

3 DESENVOLVIMENTO

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Nessa seção do trabalho serão apresentados os Requisitos do protótipo, sendo eles: Requisitos Funcionais e Não Funcionais. Em Especificação estão presentes dois diagramas: Diagrama de Caso de uso e Diagrama de Classes.

3.1.1 Requisitos Funcionais

Os Requisitos Funcionais são responsáveis por estabelecer as principais funcionalidades da aplicação. Descreve alguns comportamentos do protótipo e determina algumas situações. No Quadro 01 foram apresentados os requisitos funcionais do protótipo.

Quadro 01 – Requisitos Funcionais

RF		UC
RF01	Manter login com níveis de acesso	UC01
RF02	Manter contas a pagar	UC02
RF03	Manter contas a receber	UC03
RF04	Manter clientes	UC04
RF05	Manter usuários	UC05
RF06	Manter o sistema hospedado na máquina local do cliente	UC06
RF07	Visualizar cobranças	UC07
RF08	Manter envio de cobranças por e-mail.	UC08
RF09	Manter funcionários	UC09
RF10	Manter serviços	UC10

3.1.2 Requisitos Não Funcionais

Os requisitos não funcionais podem ser descritos como requisitos que estão presentes na aplicação, mas que não interferem diretamente na funcionalidade. No Quadro 02 podemos ver alguns exemplos

Quadro 02 – Requisitos Não Funcionais

RNF01	Utiliza navegador para ser acessado	UC01
RNF02	Integrar-se ao banco de dados	UC02
RNF03	Permitir transitar entre um “Light-Mode” ou “Dark-Mode”.	UC03
RNF04	Utilizar telas responsivas para ser utilizada em diferentes dispositivos	UC04

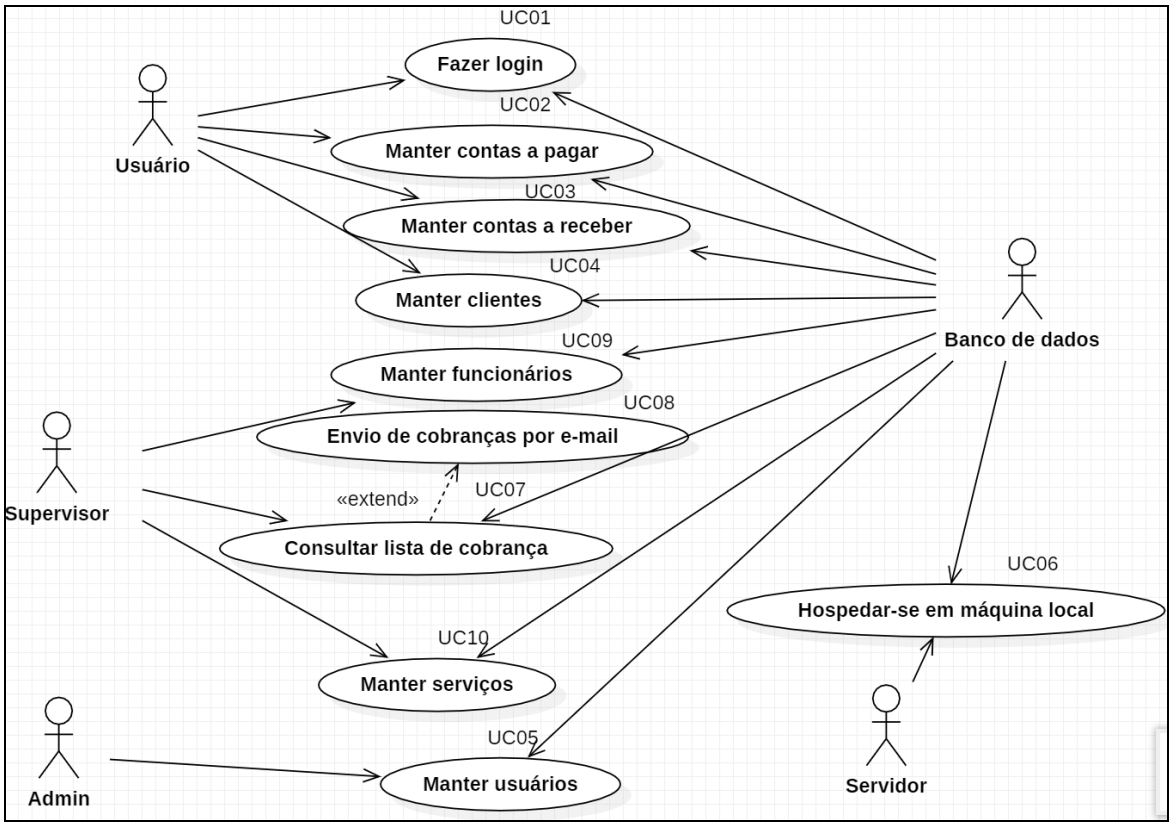
3.2 ESPECIFICAÇÃO

A especificação do protótipo será apresentada através de diagramas e notações, estas são apenas representações em modo geral dos comportamentos e operações que o protótipo oferece.

3.2.1 Diagrama de Casos de Uso

Segundo Pressman (2006, p. 137 e 138) [...] “um caso de uso conta uma história estilizada sobre como o usuário final (desempenhando um de uma série de papéis possíveis) interage com o sistema sob um conjunto de circunstâncias específicas.” Ou seja, um caso de uso representa o software do ponto de vista do usuário. Podemos observar na Figura 01 como os principais atores (usuários) irão interagir com o sistema, atendendo aos requisitos funcionais já especificados. O Servidor Firebase está na posição de ator, pois o protótipo irá interagir com ele, assim como o administrador.

Figura 01 – Diagrama de Casos de uso.



Fonte: Elaborado pelo autor

3.2.2 Notação de casos de uso

A notação ou a descrição mais detalhada de alguns casos de uso mais relevantes se faz necessária para um melhor entendimento das ações e está exposta no Quadro 03, onde especifica as ações de cada caso de uso, que atores estão envolvidos, os requisitos atendidos e tratamento de possíveis erros.

As notações ou especificações de casos de uso servem para passar ao leitor, um registro mais detalhado dos casos de uso mais relevantes em diferentes cenários. Nos quadros abaixo teremos informações valiosas sobre os atores envolvidos, requisitos atendidos e possíveis resoluções de erros

Quadro 03 – Notação de casos de uso.

UC01 - EFETUAR LOGIN COM NíVEL DE ACESSO	
REQUISITOS ATENDIDOS	RF01 – Manter login com níveis de acesso
ATORES ENVOLVIDOS	Usuário, Supervisor, Administrador

PRÉ-CONDIÇÃO	Possuir um cadastro já feito no sistema
CENÁRIO PRINCIPAL	Será exibida uma tela contendo um campo para o usuário digitar LOGIN e um campo SENHA
CENÁRIO ALTERNATIVO 1	Se o login ou senha estiverem incorretos irá aparecer uma mensagem de erro: “Login e Senha incorretos”
CENÁRIO ALTERNATIVO 2	Se o usuário não estiver cadastrado terá que solicitar cadastro.

UC3 - MANTER CADASTRO DE CONTAS A PAGAR	
REQUISITOS ATENDIDOS	RF03 - Administrador terá consciência das contas que a empresa tem pagamento em aberto
ATORES ENVOLVIDOS	
PRÉ-CONDIÇÃO	Possuir cadastro para acesso no sistema com nível de acesso
CENÁRIO ALTERNATIVO 1	Usuário acessa a tela, cadastra uma nova conta a pagar, verifica se possui alguma conta em aberto ou altera alguma conta já cadastrada se necessário.
CENÁRIO ALTERNATIVO 2	Ao tentar realizar o cadastro de uma conta a pagar, consulta o destinado ao pagamento, porém verifica que este não possui cadastro no sistema. O usuário terá que acessar o cadastro da entidade e realizar o cadastro da entidade primeiro.

UC5 - MANTER CADASTRO DE FUNCIONÁRIOS	
REQUISITOS ATENDIDOS	RF05 – Terá acesso
PRÉ-CONDIÇÃO	Possuir cadastro para acesso no sistema com nível de acesso, funcionário deverá ter o cadastro de entidade primeiro
CENÁRIO ALTERNATIVO 1	Usuário acessa o cadastro de funcionários, se o cadastro de entidade desse funcionário já foi realizado poderemos realizar o cadastro do funcionário
CENÁRIO ALTERNATIVO 2	Ao acessar o cadastro de funcionários e verificar que o mesmo não possui cadastro de entidade terá que realizar este primeiramente

Fonte: Elaborado pelo autor

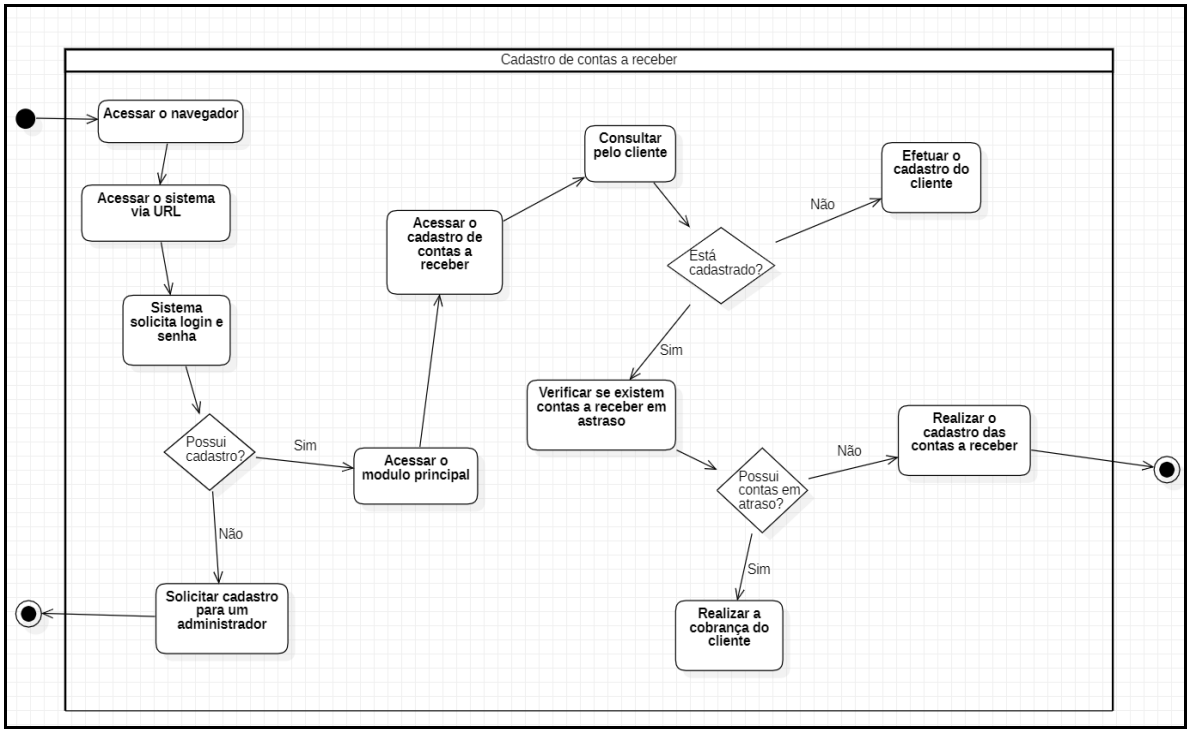
3.2.3 Diagrama de Atividades

O diagrama de atividades tem o papel de representar o fluxo de uma operação realizada dentro da própria aplicação, não tem propriedade de apresentar o código em si, mas apenas um direcionamento para o leitor poder se situar e acompanhar o processo descrito dentro da própria aplicação.

O diagrama de atividades não demonstra somente uma operação dentro da aplicação, mas serve também para o desenvolvedor poder se situar e traçar seu caminho na hora de desenvolver sem perder tempo pensando como seria efetuando o desenvolvimento com uma maior agilidade.

Na Figura 02 podemos verificar uma demonstração deste diagrama, pode ser visualizado que o diagrama segue um fluxo, deste o acesso do usuário até um processo por ele realizado.

Figura 02 – Diagrama de Atividades.



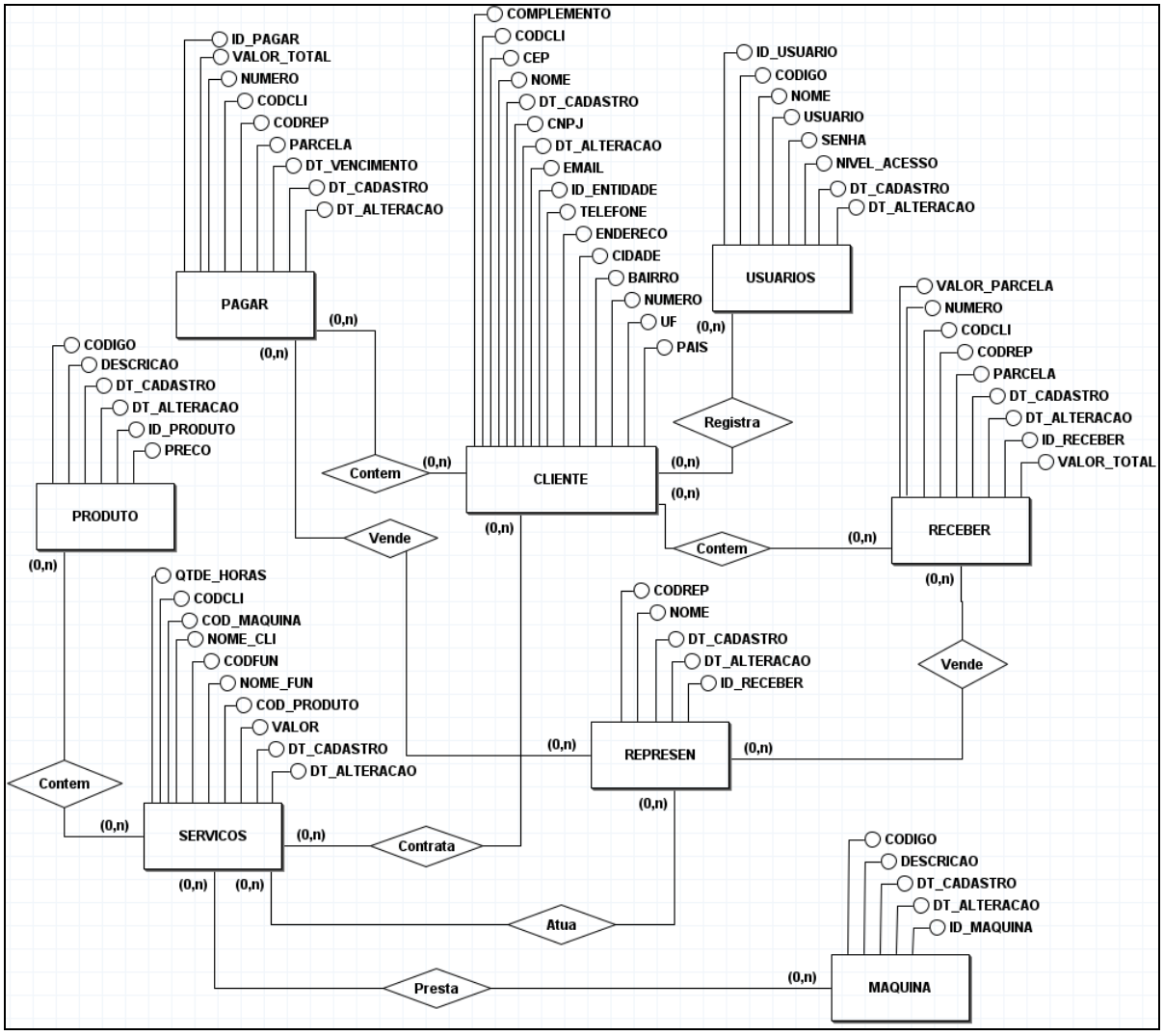
Fonte: Elaborado pelo autor

3.2.4 Banco de dados – Modelo Conceitual

Segundo Heuser (2009, p. 25), modelo conceitual é uma descrição do banco de dados de forma independente de implementação em um SGBD (Sistema Gerenciador de Banco de Dados), ele registra que dados há na estrutura, mas não como estes serão armazenados.

Na figura 03 podemos encontrar o diagrama deste modelo, apresentando o banco de dados de forma geral e relacional.

Figura 03 –Banco de Dados Conceitual



Fonte: Elaborado pelo autor

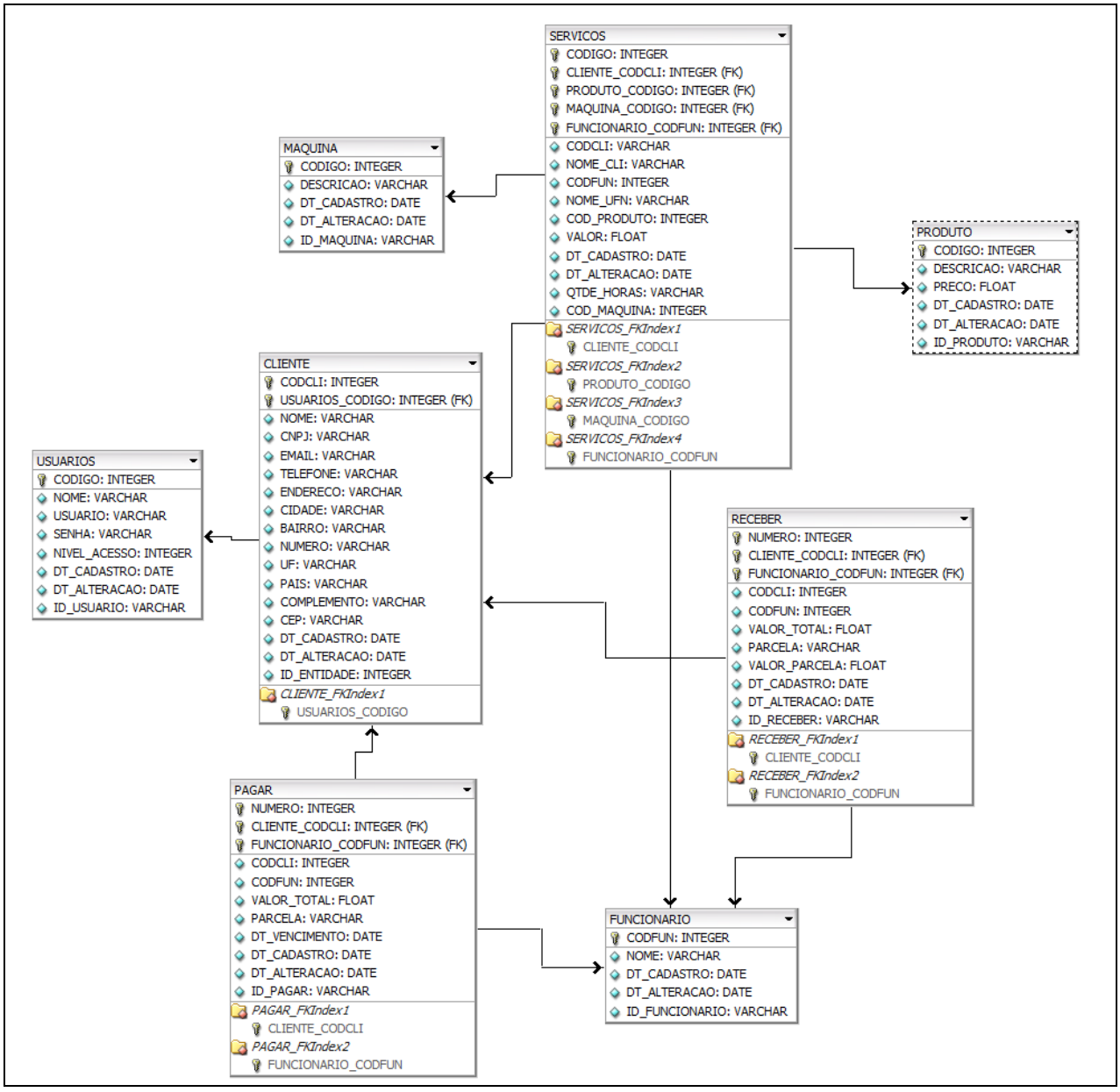
3.2.5 Banco de dados – Modelo Lógico

O modelo atua na representação da estrutura dos dados do banco de dados

conforme é visualizada pelo usuário, dependendo de um software gerenciador de banco de dados, incluindo também o tipo de dado armazenado em cada coluna e seus relacionamentos respectivamente.

Na figura 04 está representando o diagrama, este foi desenvolvido utilizando a ferramenta DBDesigner.

Figura 04 – Banco de dados Lógico.



Fonte: Elaborado pelo autor

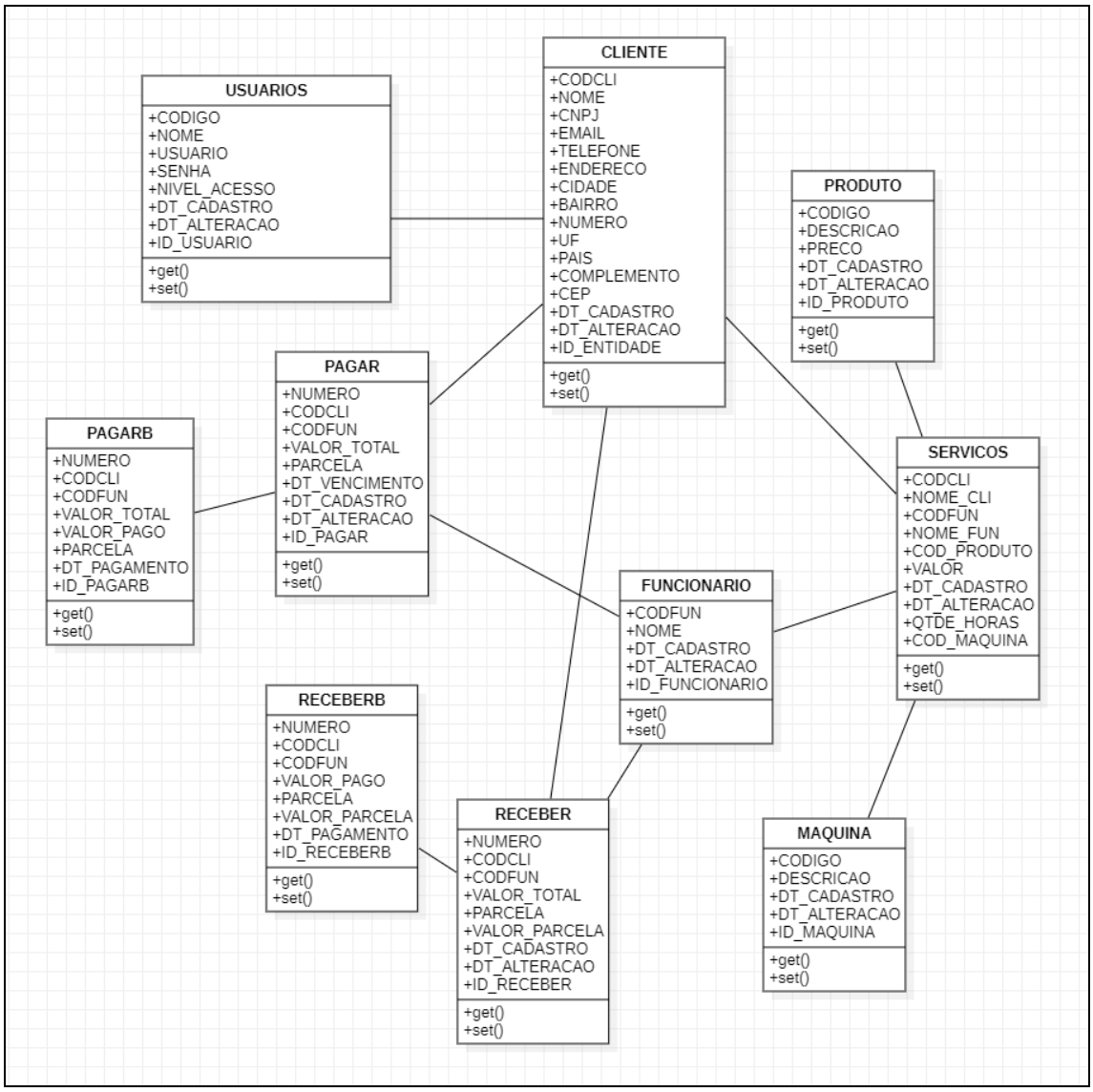
3.2.6 Diagrama de classes

O Diagrama de Classes atua na demonstração superficial de como está projetada a

disponibilidade dos métodos e atributos presentes na aplicação, contém informações importantes para o leitor adquirir uma noção do que irá encontrar ao verificar as classes presentes.

A Figura 5, representa o diagrama de classes do projeto, nele estão presentes as principais classes que foram utilizadas juntamente com as suas ligações entre métodos e atributos.

Figura 05– Diagrama de Classes.



Fonte: Elaborado pelo autor

3.3 IMPLEMENTAÇÃO

Neste tópico será apresentada uma visão geral do que foi implantado. Mostrando exemplos de como foi realizado o desenvolvimento, expressando as facilidades e

dificuldades ao desenvolver. Será mostrado também um pouco do código em si, a conexão com o banco de dados e alguns outros detalhes mais técnicos

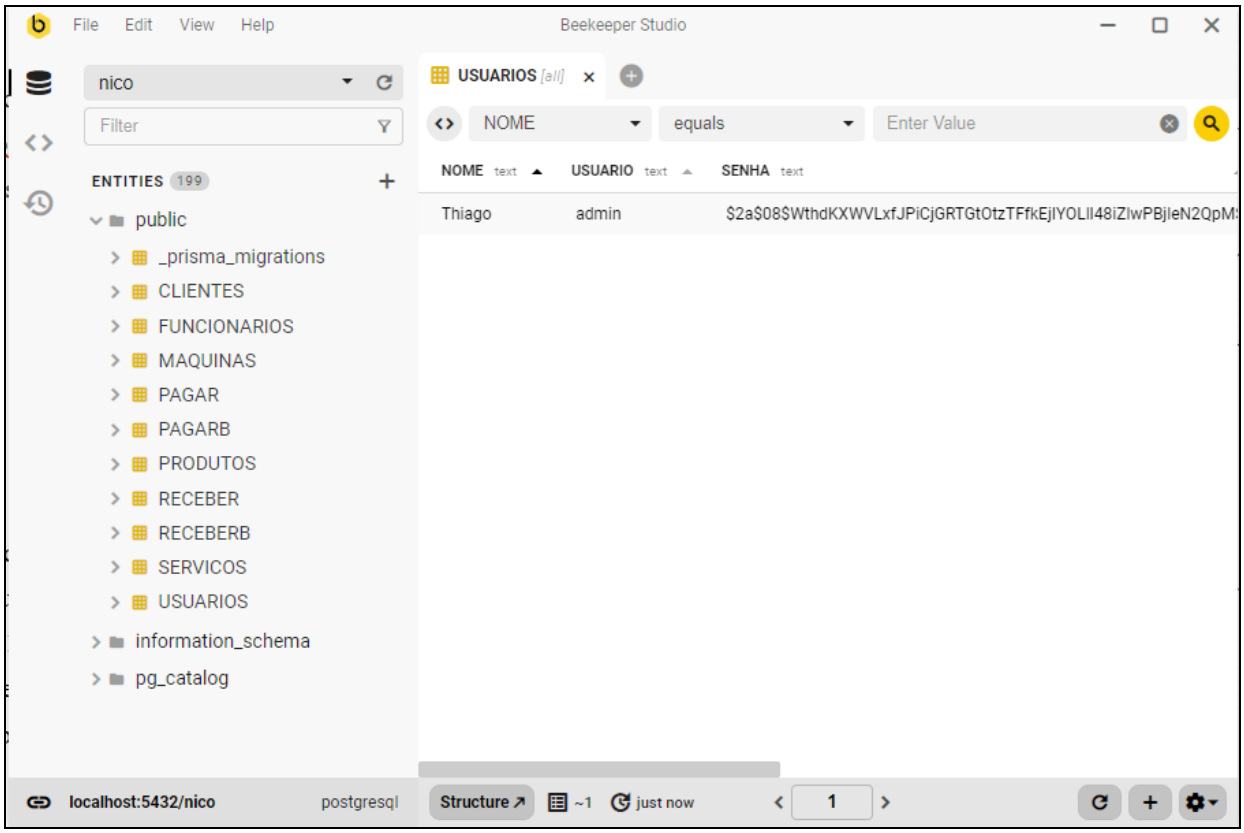
3.3.1 Técnicas e ferramentas utilizadas

Neste tópico será apresentado algumas das técnicas e ferramentas que foram utilizadas para realizar o desenvolvimento do protótipo.

3.3.1.1 Beekeeper Studio Ultimate

O Beekeeper Studio foi utilizado no processo de desenvolvimento para facilitar as operações no banco de dados. Beekeeper é um SGBD que otimiza o tempo de resposta e aumenta a praticidade para visualizar as tabelas e atualizações que estão sendo realizadas no banco de dados. A Figura 06 retrata a visualização da tabela ”USUARIOS”.

Figura 06– Beekeeper Studio.

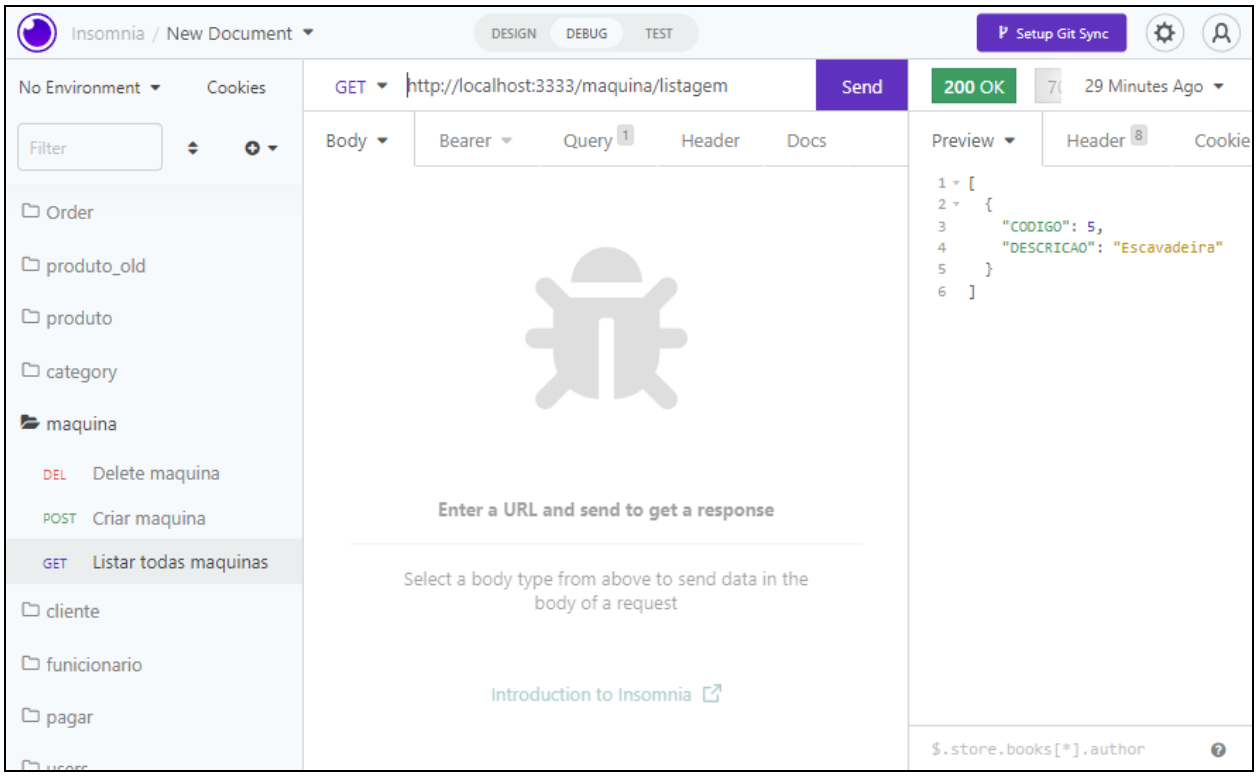


Fonte: Elaborado pelo autor

3.3.1.2 Insomnia

O Insomnia é um aplicativo que permite acessar as rotas do Backend e nelas efetuar suas funções, fazendo com que consigamos manipular os dados do banco de dados sem necessariamente termos um Frontend funcional. Foi muito utilizado quando as rotas estavam começando a serem desenvolvidas e também, muito utilizada na hora dos testes, possibilitando encontrar alguns bugs e erros de desenvolvimento do Backend. Na Figura 07 uma demonstração da requisição de listagem das máquinas .

Figura 07– Insomnia



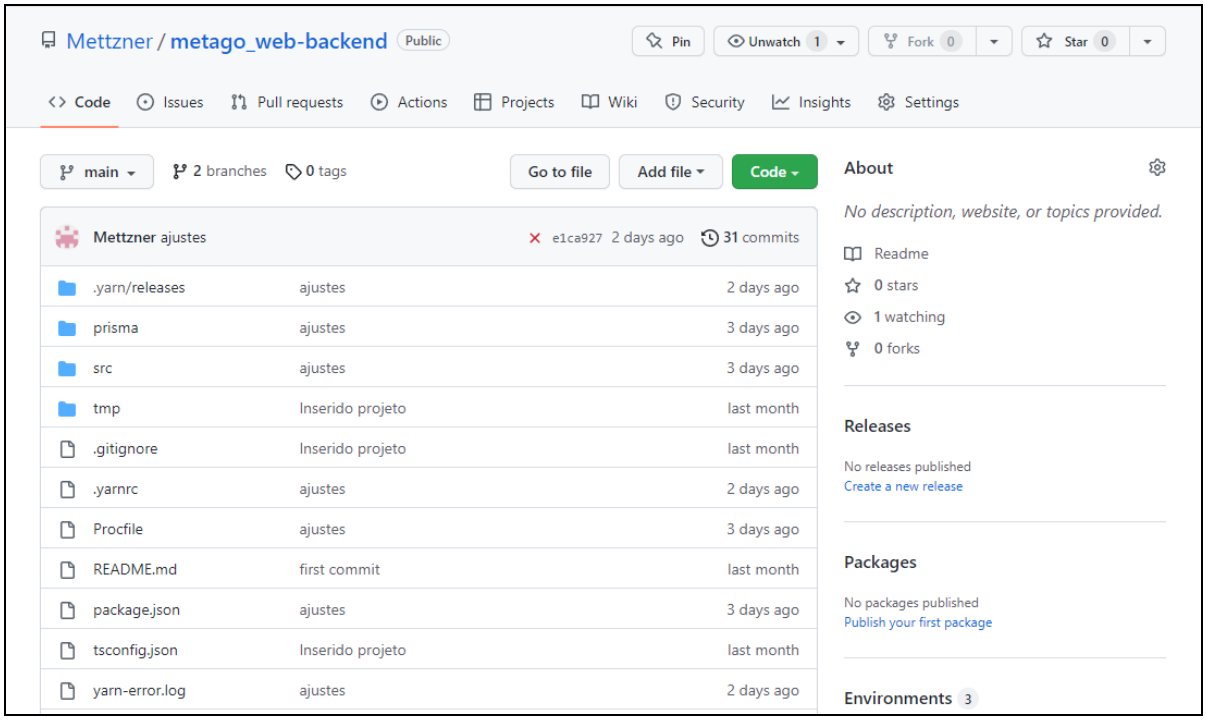
Fonte: Elaborado pelo autor

3.3.1.2 Git e GitHub

A ferramenta Git foi utilizada para o versionamento do protótipo, permitindo salvar suas versões em nuvem para evitar perda do código. O código foi enviado para o

GibHub utilizando a ferramenta GitHub Desktop que serve para facilitar o envio dos arquivos sem utilizar o código. Na Figura 08 um exemplo da interface da ferramenta GitHub.

Figura 08–Interface do GitHub.



Fonte: Elaborado pelo autor

3.3.2 Operacionalidade da implementação

Neste tema estão sendo apresentado algumas das principais telas e funcionalidades do sistema, telas ou operações que são comuns dentro do protótipo. Está também sendo apresentado um pouco do comportamento do banco de dados, criação das tabelas e algumas peculiaridades que o protótipo possui. No Frontend foram utilizadas duas principais bibliotecas, sendo elas o React e o Next.js. O uso dessas, permitiu trabalhar de forma mais dinâmica principalmente com a questão das rotas, sem utilizar o react-router-dom padrão do React.

Com o Next.js foi possível “amarrar” o sistema passando as rotas de acesso das telas direto pelo nome dos arquivos e juntamente com o Axios ajudou a tratar também as rotas que comunicam com o Backend, passando apenas o endereço e porta de acesso do banco de dados através do comando BaseURL.

Já no Backend foram utilizadas duas principais bibliotecas, o Node.js e o

Express.js, o Node com principalmente pelo diferencial de tornar possível o desenvolvimento do Backend em JavaScript, sendo assim mais simples de ser empregado. Já o Express.js veio com a funcionalidade de controles de requisições ser inteiramente compatível com o Next.js, utilizando as extensões Request, Response, NextFunction, foi possível também tornar o tempo de respostar mais otimizado, resultando para o Frontend os dados requisitados em instantes. A criação das tabelas e chamadas destas para as operações foram feitas através do Prisma. Ele também foi responsável por fazer a comunicação com o banco de dados.

Quando o Frontend chamava a rota para gravação dos dados, por exemplo, após passar pelas tratativas do Express, era encaminhado para a classe de Controller onde era passada uma função nativa do prisma com o método post, neste momento é inserido o SQL de insert para os campos determinados receberem esses dados.

3.3.2.1 Autenticação de usuário no Frontend.

É a tela de login, essa, é responsável identificar o acesso do usuário no sistema, restringindo-o e não permitindo que este tenha acesso ao sistema caso não possua seu cadastro realizado. Tem a função também de filtrar o nível de acesso desse usuário definindo o que vai ou não ter permissão para utilizar dentro do sistema.

A autenticação é realizada utilizando contexts, ou seja, armazena alguns arquivos temporários dentro dos cookies e do local storage que permitem controlar o que o usuário está fazendo.

A utilização de um token de autenticação temporário por usuário, por exemplo, permite que o usuário só consiga realizar o acesso ás telas após fazer o login, esse token fica presente nos cookies e é inserido automaticamente no momento que o usuário acessa e tem prazo de duração de 30 dias, após o prazo, o usuário terá que fazer o login novamente e dessa forma, adiquirir um novo token. Na Quadro 04 pode-se observar um exemplo do código desenvolvido em React.

Quadro 04 – Código de autenticação de usuário(React)

```
async function signIn({ USUARIO, SENHA }: SignInProps) {
  try {
    const response = await api.post('/sessao', {
      //requisitos para realizar o acesso
      USUARIO,
```

```
                SENHA
            })
            const { CODIGO, NOME, token } = response.data
            setCookie(undefined, '@nextauth.token', token, {
                maxAge: 60 * 60 * 24 * 30,
              path: "/"
            })
            setUser({
                CODIGO,
                NOME,
                USUARIO,
            })
            api.defaults.headers['Authorization'] = `Bearer ${token}`
            toast.success("Logado com sucesso")
            Router.push('/principal')
        } catch (err) {
            console.log(err)
            toast.error("Erro ao acessar")
        }
}
```

Fonte: Elaborado pelo autor

3.3.2.2 Criação das tabelas do banco de dados

Para criar as tabelas do banco de dados foi utilizado uma ferramenta chamada Prisma. Esta permite realizar a criação das tabelas por meio de um código na própria ferramenta. No código montado é possível também fazer as ligações das tabelas, definindo as propriedades dos campos e referenciando estes.

O uso do prisma permitiu uma facilidade maior na hora de realizar o acesso ao banco de dados e aumentou muito a produtividade ao longo do desenvolvimento, onde o maior impacto se deu quando foi preciso realizar alguns ajustes no banco de dados e apenas inserindo esse ajuste no código e migrando para o database já foi possível corrigir o problema que havia sido encontrado. Na Quadro 05 é possível observar um exemplo de algumas tabelas criadas utilizando o Prisma.

Quadro 05 – Criação de tabela utilizando Prisma.

```
model Maquina {
  CODIGO       Int     @default(autoincrement())
  DESCRICAO    String
  DT_CADASTRO  DateTime? @default(now())
  DT_ALTERACAO DateTime? @default(now())
```

```
  ID_MAQUINA   String    @id @default(uuid())
  Servicos Servicos[]
  @@map("MAQUINAS")
}
model Funcionario {
  CODFUN        Int       @default(autoincrement())
  NOME  String
  DT_CADASTRO    DateTime? @default(now())
  DT_ALTERACAO   DateTime? @default(now())
  ID_FUNCIONARIO String    @id @default(uuid())

  Servicos Servicos[]
  PagarB   PagarB[]
  Receber  Receber[]
  ReceberB ReceberB[]
  Pagar    Pagar[]
  @@map("FUNCIONARIOS")
}
```

Fonte: Elaborado pelo autor.

3.3.2.3 Autenticação do usuário no Backend.

Para realizar a autenticação no Backend foi utilizada a especificação JWT (Json Web Token). A JWT foi responsável por transportar os dados de cadastro ou login do usuário por um JSON criptografado até o método que manipula esses dados no Backend.

No momento que esses dados criptografados chegam no método, é utilizado a biblioteca Bcrypt.js que fica responsável por receber a senha presente no JSON e realizar a sua criptografia nessa senha, armazenando-a no banco de dados já criptografada.

Esse processo se repete quando o usuário realiza o login, porém, dessa vez, de forma inversa, no momento em que o usuário insere seus dados e estes chegam ao Backend o sistema realiza uma consulta nos dados desse usuário, nesses dados está presente a senha criptografada que é comparada com a senha que o usuário acabou de inserir, caso as senhas sejam idênticas o usuário recebe autorização para acessar, caso contrário não conseguirá realizar o acesso. Na Quadro 06 podemos observar uma parte do trecho de código responsável por realizar o cadastro do usuário com a criptografia da senha.

Quadro 06 – Autenticação do usuário no Backend.

```
Const    suário = await prismaClient.usuarios.create({
   data: {
      NOME: NOME,
      USUARIO: USUARIO,
      SENHA: passwordHash,
      NIVEL_ACESSO: NIVEL_ACESSO,
   },
   select: {
      CODIGO: true,
      NOME: true,
      USUARIO: true
   }
})
```

Fonte: Elaborado pelo autor

3.3.2.4 Tela de cadastro de usuário

Neste tópico, está sendo representada a de cadastro de usuário utilizada para determinar usuário que utilizarão o sistema e seus níveis de acesso, nela o usuário se depara com uma interface simples conseguindo visualizar únicos campos, estes solicitam nome do usuário o usuário e a senha para poder entrar no sistema e o nível de acesso. O usuário insere os dados solicitados, pressiona a tela “Enter” ou seleciona o botão “Cadastrar”, se possuir cadastro irá receber uma mensagem informando que o usuário já está cadastrado. A tela em questão está sendo retratada na Figura 09.

Figura 09 – Tela de cadastro de usuários.

Fonte: Elaborado pelo autor

3.4 RESULTADOS E DISCUSSÃO

Com o desenvolvimento e análise do trabalho, foi possível concluir que os objetivos inicialmente apresentados foram parcialmente atendidos. Ao longo da trajetória foi comprovado que as linguagens utilizadas tornavam o desenvolvimento mais simples e dinâmico, sendo esse o motivo por ter sido escolhido essas ferramentas.

A integração do protótipo com o banco de dados foi um dos objetivos apresentados. Este foi atendido a partir do momento que as primeiras requisições de teste no Insomnia passaram a gravar esses dados e retorná-los quando solicitados.

Mais um dos objetivos apresentados e que foram atendidos foi a validação de nível de acesso de acordo com o cadastro dos usuários, podendo ser notada no momento em que o usuário com nível Usuário ou Supervisor não possui permissão para gravar outros usuários.

Pode-se dizer que o objetivo para esquematizar a gestão financeira com envio de cobranças automático por e-mail foi parcialmente atendido, pois ao longo do desenvolvimento surgiram outras necessidades que passaram a ter prioridade para serem entregues. Estas sendo, os cadastros mais básicos do sistema, pois sem estes, a parte financeira em si não poderia ser finalizada.

Por fim, os objetivos de empregar mudança de tema entre Light-Mode e Dark-Mode e de hospedagem do sistema em máquina local foram atendidos. Tornando possível alterar as cores do sistema de modo que se tornasse mais confortável a visualização e o sistema passou a poder ser acessado utilizando o IP da máquina.

4 CONCLUSÕES

Um dos motivos por ter sido escolhido esse tema, além de suprir a necessidade da empresa de possuir um sistema de gerenciamento, foi o conhecimento adquirido utilizando as ferramentas escolhidas. Para chegar no resultado apresentado, foi buscado inspiração em alguns outros sistemas e protótipos existentes, estes, encontrados na internet, ou a partir de algumas sugestões da própria empresa.

Para atender os objetivos propostos pela empresa foi realizado o levantamento de requisitos através de uma entrevista dirigida com o proprietário. Neste momento foi possível retirar algumas dúvidas e levantar algumas necessidades do sistema. A partir deste momento foi possível destacar quais seriam requisitos principais do projeto.

Ao longo do desenvolvimento pode ser notado algumas dificuldades. Estas sendo, a falta de conhecimento nas linguagens escolhidas, sendo necessário buscar informações e realizar diversos estudos para poder dar continuidade no protótipo.

Uma das dificuldades encontradas também foi a pesquisa, mesmo tendo uma internet farta de informações referentes ao desenvolvimento nessas linguagens, em alguns momentos o protótipo teve breves paradas para que fosse possível dedicar um tempo maior procurando um erro ou bug que estava ocorrendo.

Como pontos positivos pode ser dito que o trabalho de estudo em cima das linguagens agregou para conhecimento pessoal e tornou o desenvolvimento do protótipo algo divertido.

Por fim, fazendo uma breve análise nos pontos levantados, podemos verificar que os pontos de hospedagem do sistema na nuvem e de possuir uma gestão financeira completa foram parcialmente atendidos, visto que a hospedagem foi realizada em máquina local e para realizar uma gestão financeira completa seria preciso ter vários cadastros e subcadastros criados. Mesmo assim, foram implementadas as telas de contas a pagar e receber, funcionais, mas com diversas limitações. Portanto foi decidido que estes pontos seriam implementados e melhorados em versões futuras.

4.1 EXTENSÕES

Algumas ferramentas utilizadas para a estilização do sistema como o SASS, apesar de serem ótimas, podem ser facilmente substituídas por ferramentas que já fornecem os componentes prontos, como o Material UI, que foi utilizado, e a partir disso

foi retirado esta conclusão. Ele inibe o tempo investido para estilizar os componentes e faz com que o sistema tenha as mesmas proporções, dessa forma fornecendo simetria entre os elementos.

Outro ponto que foi levantado com o desenvolvimento foi o de emissão de relatórios, que facilitariam muito a visualização e organização dos dados de forma física, tendo a possibilidade de ser impresso e permitindo com que o usuário pudesse trabalhar com esses relatórios de forma mais ampla.

Mais um ponto que será implementado em uma futura versão, será o complemento das telas financeiras, que receberão uma “amarração“ maior com o sistema.

5 REFERENCIAS

ALKASS, S., EL-MOSLMANI, K., e ALHUSSEIN, M. (2003) A computer model for selecting equipment for earthmoving operations using queuing theory. Annual Conference of the International Council for Research and Innovation in Building and Construction (CIB), Auckland, New Zealand, W78.

AZEVEDO, J.G.C, et al, Desenvolvimento de software para a otimização da produção horário de equipamentos em operações de terraplenagem. In UNIVERSIDADE FEDERAL DO PARANÁ, 2018. UFPR .

DAHL, Ryan - History of Node.js. Phx Tag Soup, 2011. Disponível em: . Acesso em: 10 mai. 2014 INTRODUCTION to node.js with Ryan Dahl. Disponível em: . Acesso em: 19 jun. 2014.

FALCÃO, Viviane A; SOUSA, Luiz M. **Planejamento de obras de terraplanagem e pavimentação:** um manual de referência, 2014.

FIORINI, S., Staa, A.V., Baptista, R.M., Engenharia de Software com CMM, Brasport Livros e Multimídia Ltda, Rio de Janeiro, 1998.

GRANSBERG, D. D.; POPESCU, C. M.; RYAN, R. Construction Equipment Management for Engineers, Estimators, and Owners. CRC Press, 2006.

KHUAT, T. Developing a frontend application using ReactJS and Redux. Dissertação (Degree Programme in Business Information Technology Bachelor's) — Laurea University of Applied Sciences, Leppävaara, 2018.

LIMA, M. O guia completo do React e o seu ecossistema. [S. l.]: - Imasters, 2017. Disponível em: https://imasters.com.br/desenvolvimento/o-guia-completo-do-react-e-o-seu-ecossistema.

MELO, D. O que é TypeScript? [Guia para iniciantes] | Aplicativos e Software | Tecnoblog.

MOSELHI, O, e ALSHIBANI, A. (2009) Optimization of earthmoving operations in heavy.

PEREIRA, Antônio Nunes - A Importância do Controle Interno para gestão de empresas, 2008.

PORTUGAL, R. L. Q.; LEITE, J. C. S. P. A Transparência do GitHub para uso de Artefatos como fontes de informação na Engenharia de Requisitos. [s.l: s.n.].

REDAÇÃO VINDI. O que é REST API? Entenda mais sobre integração de sistemas. Acesso em: 31 maio. 2021.

VENTURA, Shayder. B. **Estudo de caso:** empresa prestadora de serviço de terraplanagens em relação ao clima adverso na região de Criciúma/SC, 2016.

VIANA, THiago. M. *et al.* Análise e proposta de sistema de gerenciamento em obras de terraplanagem, 2013.
YIN, Robert K. **Estudo de caso:**planejamento e métodos. 2 ed. São Paulo: Atlas,2001.

ZAHRAN, S., Software Process Improvement – Practical Guidelines for Business Success, 1 ed., Addison-Wesley, 1998.