CENTRO PAULA SOUZA

FATEC INDAIATUBA - FACULDADE DE TECNOLOGIA DE

INDAIATUBA DR. ARCHIMEDES LAMMOGLIA

ALICE VITÓRIA ALVES

FABRICIA GOMES PEREIRA

LUCAS MARQUES ROMERA

MATEUS MARQUES FRAHM

Irriga Vida

Projeto Interdisciplinar 3º Semestre

Indaiatuba

Dezembro/2021

CENTRO PAULA SOUZA

FATEC INDAIATUBA - FACULDADE DE TECNOLOGIA DE INDAIATUBA DR. ARCHIMEDES LAMMOGLIA

ALICE VITÓRIA ALVES
FABRICIA GOMES PEREIRA
LUCAS MARQUES ROMERA
MATEUS MARQUES FRAHM

Irriga Vida

Projeto Interdisciplinar 3º Semestre

Este trabalho visa como seu principal objetivo contribuir para a compilação de todos os recursos e ensinamentos vinculados pelo Projeto Integrador de disciplinas da FATEC Indaiatuba Dr. Archimedes Lammoglia.

Indaiatuba
Dezembro/2021

RESUMO NA LÍNGUA VERNÁCULA

A Irriga vida é uma empresa de automatização de irrigação com o foco no pequeno e médio produtor, com o objetivo de levar tecnologia e facilidade para esse publico alvo. Preservando dessa forma o recurso da água e a irrigação desnecessária do solo, com soluções práticas, econômicas e objetivas, permitindo a visualização geral de toda a produção e quebrando paradigmas de que tecnologia só é possível para grandes produtores.

O foco da empresa é a satisfação do cliente para inovar a maneira de plantar no país.

Palavras-Chave: Produtividade, inovação, tecnologia.

ABSTRACT

The Irriga Vida is a company of automation irrigation with the focus in small and medium producers, with the objective of take technology and facility for this type of audience. In this way preserving the water resource and the unnecessary irrigation of the soil, with easy solutions, economical and assertions, allowing the overview of all the production and breaking paradigms that the technology is only possible for large producers.

The focus of the company is the customer satisfaction to innovate the way of planting in country.

Key words: Productivity, innovation, technology.

SUMÁRIO

SUMÁRIO DE FIGURAS

SUMÁRIO DE TABELAS

1. INTRODUÇÃO

Após mais um semestre seguimos com a empresa de irrigação: A Irriga Vida é uma empresa que está focada na proposta de valor, que seria praticidade com segurança, gerando assim mais produtividade. Sempre procurando inovar e acompanhar as tendências de mercado, visando sempre a sustentabilidade. O diferencial da empresa é atender e superar as expectativas do cliente, pois sabe-se a dificuldade que se tem hoje na falta de acessibilidade para esses produtos para Micro e Pequenos Produtores. O Produto abrange toda a área da informática, sempre visando as novas tecnologias e os meios de acesso.

Os envolvidos nesse projeto são alunos do Curso de Análise e Desenvolvimento de Sistemas da Faculdade de Tecnologia de Indaiatuba Dr. Archimedes Lammoglia. Sendo eles Alice Vitória Alves, Fabricia Gomes Pereira, Lucas Marques Romera e Mateus Marques Frahm. Toda a equipe da Irriga Vida, tem funções ligadas para cada especialização, desde programadores até administradores.

A empresa foi desenvolvida com base no mercado atual, onde até 2050 seremos mais de 9 bilhões de pessoas no mundo, onde com os atuais métodos de agricultura não serão mais produtivos o suficiente para suprir toda essa futura demanda. Pensando nisso, a equipe da Irriga Vida realizou diversas buscas e encontrou uma pesquisa realizada pela Agência Nacional de Águas, que mostra que 67% do total de água utilizada atualmente vai para a agricultura para os métodos de irrigação. E a maioria destes utilizados atualmente desperdiça grande parte da água.

Uma das principais dificuldades, é a falta de interesse do agricultor em investir nos equipamentos, sem saber que a médio prazo, já teria retornado o valor investido. Outro ponto também é a questão de profissionais que estejam comprometidos nesse meio de trabalho.

2. CONTEÚDO

2.1 Banco de Dados

Planejamos e documentamos a estrutura do banco de dados por meio do diagrama de entidade e relacionamento, conforme instruído pela docente Maria das Graças Tomazela. Além disso criamos os scripts com a criação das tabelas e inserção de seus dados iniciais.

2.1.1 MER e DER – Conceitual

Para atender o projeto criamos 8 tabelas sendo elas:

TB_EQUIPAMENTO: Tem como função armazenar todos os equipamentos comprados da irriga vida pelo cliente, para ajudar no controle de versão de equipamentos e controle de manutenção;

TB_AGENDAMENTOS: Tabela responsável por controlar os agendamentos do cliente. É necessário ter pelo menos um equipamento cadastrado na tabela de equipamentos. Nela também existirá relação para visualizar qual usuário efetuou o agendamento.

TB_USUARIO: Tabela fundamental do sistema, responsável por controlar acessos dentro do sistema.

TB_TAREFAS: Tabela para o controle de tarefas a ser realizadas por usuário dentro do sistema, também tendo relacionamento com a tabela de usuários.

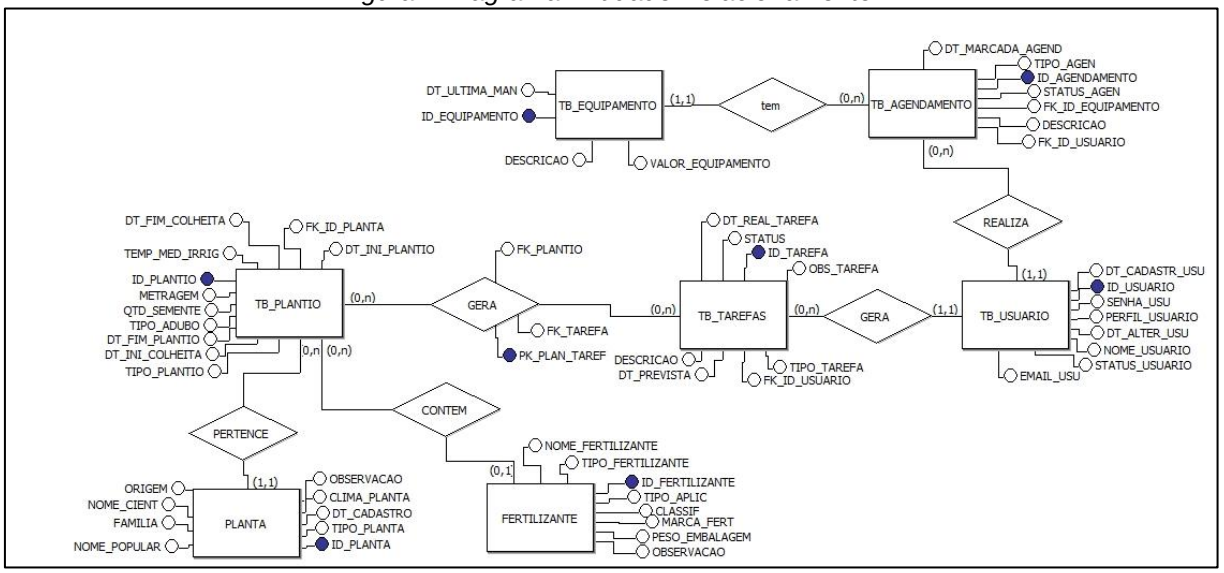
TB_TAREF_PLANTIO: Relacionamento entre as tabelas tarefas e plantio, tendo código único de identificação.

TB_PLANTIO: Tabela para controle do ciclo de vida de uma plantação, sendo possivel vincular com tabelas do tipo Plantio e colheita;

TB_PLANTA: Tabela responsável pelo cadastro de plantas para serem vinculadas ao plantio, com informações importantes no processo de plantio e manutenção da planta;

TB_FERTILIZANTES: Tabela responsável por cadastrar informações de fertilizantes para serem utilizados no plantio.

Figura 1 Diagrama Entidade Relacionamento

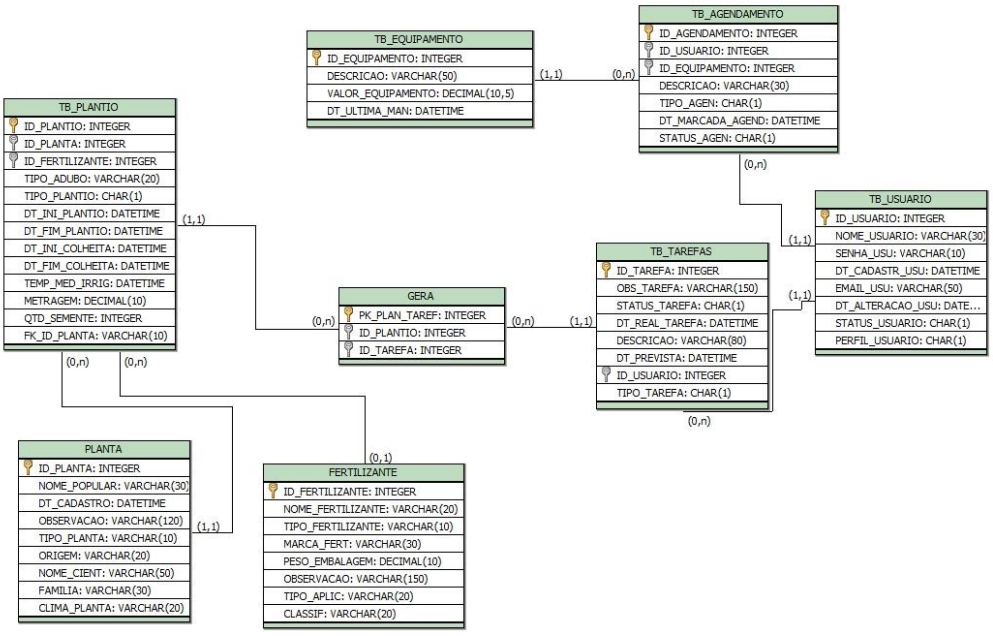


Fonte: Própria

2.1.2 DER modelo Lógico

O Modelo lógico é um diagrama de modelagem de banco de dados complementar ao conceitual, o modelo lógico serve para realizar a representação de uma forma mais gráfica, indicando os tipos e tamanhos dos campos, tornando mais similar a projeção do banco de dados.

Figura 2 Modelo lógico



Fonte: Própria

2.1.3 Script do Banco de dados

Para iniciar o processo de criação de banco, criamos a base DB_IRRIGA_VIDA:

```
CREATE DATABASE DB_IRRIGA_VIDA;
```

Criamos a tabela principal, TB_USUARIOS, tendo como chave o campo ID_USUARIO gerado automaticamente, usamos também a instrução CHECK para evitar cadastro errado de informações:

```
USE DB_IRRIGA_VIDA;

-- CRIAÇÃO DA TABELA PARA CONTROLE DE USUÁRIO
CREATE TABLE TB_USUARIO (
	ID_USUARIO INTEGER PRIMARY KEY NOT NULL IDENTITY (1,1),
	NOME_USUARIO VARCHAR (30) NOT NULL,
	SENHA_USU VARBINARY(MAX) NOT NULL,
	DT_CADASTRO_USU DATETIME NOT NULL,
	EMAIL_USU VARCHAR (50),
	DT_ALTERACAO_USU DATETIME,
	STATUS_USUARIO CHAR (1) NOT NULL CHECK (STATUS_USUARIO IN ('A','I')),
	PERFIL_USUARIO CHAR(1) NOT NULL  CHECK (PERFIL_USUARIO IN ('A','U')) default 'U'
);

);
```

Após isso fizemos a inserção dos dados iniciais dessa tabela, para evitar problemas ao adicionar as demais informações:

```
-- ADICIONANDO VALORES
INSERT INTO TB_USUARIO(NOME_USUARIO,
SENHA_USU,DT_CADASTRO_USU,EMAIL_USU,STATUS_USUARIO,PERFIL_USUARIO)
VALUES('ALICE','ADM123','02/10/2021','alice.alves@fatec.sp.gov.br','A','U');

INSERT INTO TB_USUARIO(NOME_USUARIO,
SENHA_USU,DT_CADASTRO_USU,EMAIL_USU,STATUS_USUARIO,PERFIL_USUARIO)
VALUES('MATEUS
MARQUES','ADM123','02/10/2021','mateus.marques@fatec.sp.gov.br','A','A');

INSERT INTO TB_USUARIO(NOME_USUARIO,
SENHA_USU,DT_CADASTRO_USU,EMAIL_USU,STATUS_USUARIO,PERFIL_USUARIO)
VALUES('FABRICIA','ADM123','02/10/2021','fabricia.gomes@fatec.sp.gov.br','A','U');

INSERT INTO TB_USUARIO(NOME_USUARIO,
SENHA_USU,DT_CADASTRO_USU,EMAIL_USU,STATUS_USUARIO,PERFIL_USUARIO)
VALUES('LUCAS ROMERA','ADM123','02/10/2021','lucas.romera@fatec.sp.gov.br','A','U');
```

Em seguida, criamos a tabela para controle de equipamentos:

```
-- TABELA DE CONTROLE DE EQUIPAMENTO
CREATE TABLE TB_EQUIPAMENTO(
	ID_EQUIPAMENTO INTEGER PRIMARY KEY NOT NULL IDENTITY(1,1),
	DESCRICAO VARCHAR(50) NOT NULL,
	VALOR_EQUIPAMENTO DECIMAL(10,5) NOT NULL,
```

```
	DT_ULTIMA_MAN DATETIME -- DATA DA ULTIMA MANUTENÇÃO
);
```

Em seguida, adicionamos suas informações:

```
INSERT INTO TB_EQUIPAMENTO(DESCRICAO, VALOR_EQUIPAMENTO, DT_ULTIMA_MAN)
VALUES('Irrigador Giratório', 123,'02/10/2020');

INSERT INTO TB_EQUIPAMENTO(DESCRICAO, VALOR_EQUIPAMENTO,DT_ULTIMA_MAN)
VALUES('Auto Temporizador',5000,'02/10/2019');

INSERT INTO TB_EQUIPAMENTO(DESCRICAO, VALOR_EQUIPAMENTO,DT_ULTIMA_MAN)
VALUES('Auto Temporizador Automatic',200,'02/10/2019');

INSERT INTO TB_EQUIPAMENTO(DESCRICAO, VALOR_EQUIPAMENTO,DT_ULTIMA_MAN)
VALUES('Irrigador hortaliças simples',10000,'02/10/2019');

INSERT INTO TB_EQUIPAMENTO(DESCRICAO, VALOR_EQUIPAMENTO,DT_ULTIMA_MAN)
VALUES('Mangueira para irrigação simples',50,'02/10/2019');
```

Na tabela de agendamentos, criamos o relacionamento através de chaves estrangeiras com as tabelas de usuários e equipamentos:

```
-- TABELA DE AGENDAMENTOS DE MANUTENÇÃO
CREATE TABLE TB_AGENDAMENTOS (
	ID_AGENDAMENTO INTEGER PRIMARY KEY NOT NULL IDENTITY(1,1),
	ID_USUARIO INTEGER NOT NULL,
	ID_EQUIPAMENTO INTEGER NOT NULL,
	DESCRICAO VARCHAR(30) NOT NULL,
	TIPO_AGEN CHAR(1) NOT NULL  CHECK (TIPO_AGEN IN ('P','U')), -- P -PREVENTIVA |
U - URGENCIA
	DT_MARCADA_AGEND DATETIME NOT NULL,
	STATUS_AGEN CHAR(1) NOT NULL  CHECK (TIPO_AGEN IN ('E','P','C')),  -- E -
EFETUADO | P - PENDENTE | C- CANCELADO
	FOREIGN KEY(ID_USUARIO) REFERENCES TB_USUARIO (ID_USUARIO),
	FOREIGN KEY(ID_EQUIPAMENTO) REFERENCES TB_EQUIPAMENTO (ID_EQUIPAMENTO)
);
```

Inserção respeitando os relacionamentos:

```
INSERT INTO TB_AGENDAMENTOS(ID_USUARIO, ID_EQUIPAMENTO,DESCRICAO, TIPO_AGEN,
DT_MARCADA_AGEND, STATUS_AGEN)
VALUES(1,1,'Preventida','P','10/10/2021','P');

INSERT INTO TB_AGENDAMENTOS(ID_USUARIO, ID_EQUIPAMENTO,DESCRICAO, TIPO_AGEN,
DT_MARCADA_AGEND, STATUS_AGEN)
VALUES(1,2,'Problema de vazamento','U','18/10/2021','C');

INSERT INTO TB_AGENDAMENTOS(ID_USUARIO, ID_EQUIPAMENTO,DESCRICAO, TIPO_AGEN,
DT_MARCADA_AGEND, STATUS_AGEN)
VALUES(3,4,'Problema de fabricação','U','01/01/2021','P');

INSERT INTO TB_AGENDAMENTOS(ID_USUARIO, ID_EQUIPAMENTO,DESCRICAO, TIPO_AGEN,
DT_MARCADA_AGEND, STATUS_AGEN)
VALUES(2,2,'Preventiva no prazo','P','01/01/2020','C');

INSERT INTO TB_AGENDAMENTOS(ID_USUARIO, ID_EQUIPAMENTO,DESCRICAO, TIPO_AGEN,
DT_MARCADA_AGEND, STATUS_AGEN)
```

```
VALUES(4,3,'Limpeza','U','18/05/2021','C');
```

Em seguida, iniciamos a criação da tabela de plantas para, logo em seguida criar as demais tabelas dependentes:

```
CREATE TABLE TB_PLANTA (
	ID_PLANTA INTEGER PRIMARY KEY NOT NULL IDENTITY(1,1),
	NOME_POPULAR VARCHAR(30) NOT NULL,
	DT_CADASTRO DATETIME,
	OBSERVACOES VARCHAR(120),
	TIPO_PLANTA VARCHAR(10),
	ORIGEM VARCHAR(20),
	NOME_CIENT VARCHAR(50),
	FAMILIA VARCHAR(30),
	CLIMA VARCHAR(20)
);
```

Inserção dos dados da tabela:

```
INSERT INTO TB_PLANTA(NOME_POPULAR, DT_CADASTRO,OBSERVACOES, TIPO_PLANTA, ORIGEM,
NOME_CIENT,FAMILIA,CLIMA)
VALUES('Alface Verde',SYSDATETIME(),'Manter de 20 a
25ºC','Hortence','Mediterraneo','Lactuca sativa','Hortalicas','temp amenas');

INSERT INTO TB_PLANTA(NOME_POPULAR, DT_CADASTRO, TIPO_PLANTA, ORIGEM,
NOME_CIENT,FAMILIA,CLIMA)
VALUES('Tomate',SYSDATETIME(),'Fruta','Mexico','Solanum
lycopersicum','Fruta','quente');

INSERT INTO TB_PLANTA(NOME_POPULAR, DT_CADASTRO, TIPO_PLANTA, FAMILIA,CLIMA)
VALUES('Morango',SYSDATETIME(),'Fruta','Fruta','frio');

INSERT INTO TB_PLANTA(NOME_POPULAR, DT_CADASTRO, TIPO_PLANTA, ORIGEM,
NOME_CIENT,FAMILIA,CLIMA)
VALUES('Acelora',SYSDATETIME(),'Fruta','Brasil','Malpighia
emarginata','Fruta','quente');

INSERT INTO TB_PLANTA(NOME_POPULAR, DT_CADASTRO, TIPO_PLANTA, ORIGEM,
NOME_CIENT,FAMILIA,CLIMA)
VALUES('Uva',SYSDATETIME(),'Fruta','Europa','Grapevines (Videiras)','Fruta','frio');
```

Assim como a tabela de plantas, criamos a tabela de fertilizantes em seguida para respeitar os relacionamentos:

```
CREATE TABLE TB_FERTILIZANTE (
	ID_FERTILIZANTE INTEGER PRIMARY KEY NOT NULL IDENTITY(1,1),
	NOME_FERTILIZANTE VARCHAR(20) NOT NULL,
	TIPO_FERTILIZANTE VARCHAR(10),
	MARCA_FERT VARCHAR(30),
	PESO_EMBALAGEM DECIMAL,
	OBSERVACAO VARCHAR(150),
	TIPO_APLIC VARCHAR(20),
	CLASSIF VARCHAR(20)
);
```

Script de inserção de fertilizantes:

```
INSERT INTO TB_FERTILIZANTE(NOME_FERTILIZANTE, MARCA_FERT, PESO_EMBALAGEM,
OBSERVACAO,TIPO_APLIC,CLASSIF)
VALUES('Fertilizante Liquído','Dimy',2,'Indicado para uso
doméstico.','liquido','domestico');
```

```
INSERT INTO TB_FERTILIZANTE(NOME_FERTILIZANTE, TIPO_FERTILIZANTE, MARCA_FERT,
PESO_EMBALAGEM, OBSERVACAO,TIPO_APLIC,CLASSIF)
VALUES('Fertilizante Forth','Tecnico','Forth',1,'complementar e repor os
nutrientes.','spray','domestico');

INSERT INTO TB_FERTILIZANTE(NOME_FERTILIZANTE, TIPO_FERTILIZANTE, MARCA_FERT,
PESO_EMBALAGEM, OBSERVACAO,TIPO_APLIC,CLASSIF)
VALUES('Fertilizante TEC','Tecnico','Forth',1,'complementar e repor os
nutrientes.','spray','domestico');

INSERT INTO TB_FERTILIZANTE(NOME_FERTILIZANTE, TIPO_FERTILIZANTE, MARCA_FERT,
PESO_EMBALAGEM, OBSERVACAO,TIPO_APLIC,CLASSIF)
VALUES('Fertilizante Forth industrial','Tecnico','Forth',1,'complementar e repor os
nutrientes.','liquido','industrial');

INSERT INTO TB_FERTILIZANTE(NOME_FERTILIZANTE, TIPO_FERTILIZANTE, MARCA_FERT,
PESO_EMBALAGEM, OBSERVACAO,TIPO_APLIC,CLASSIF)
VALUES('Fertilizante Forth 4kg','Tecnico','Forth',4,'complementar e repor os
nutrientes.','spray','domestico');
```

Criação da tabela de plantio, com as chaves estrangeiras das tabelas de Planta e Fertilizantes, com as informações necessárias para efetuar o controle do plantio:

```
CREATE TABLE TB_PLANTIO (
	ID_PLANTIO INTEGER PRIMARY KEY NOT NULL IDENTITY(1,1),
	ID_PLANTA INTEGER NOT NULL,
	ID_FERTILIZANTE INTEGER NOT NULL,
	TIPO_ADUBO VARCHAR(20),
	TIPO_PLANTIO CHAR(1),
	DT_INI_PLANTIO DATETIME NOT NULL,
	DT_FIM_PLANTIO DATETIME,
	DT_INI_COLHEITA DATETIME,
	DT_FIM_COLHEITA DATETIME,
	TEMPO_MED_IRRIGA DATETIME,
	METRAGEM DECIMAL,
	QTD_SEMENTE INTEGER,
	FOREIGN KEY(ID_PLANTA) REFERENCES TB_PLANTA (ID_PLANTA),
	FOREIGN KEY(ID_FERTILIZANTE) REFERENCES TB_FERTILIZANTE (ID_FERTILIZANTE)
);
```

Script de inserção de plantio:

```
INSERT INTO TB_PLANTIO(ID_PLANTA, ID_FERTILIZANTE, DT_INI_PLANTIO, QTD_SEMENTE)
VALUES(1,1,'02/10/2021',1000);

INSERT INTO TB_PLANTIO(ID_PLANTA, ID_FERTILIZANTE, DT_INI_PLANTIO, QTD_SEMENTE)
VALUES(2,1,'01/10/2021',1000);

INSERT INTO TB_PLANTIO(ID_PLANTA, ID_FERTILIZANTE, DT_INI_PLANTIO)
VALUES(1,2,'01/01/2021');

INSERT INTO TB_PLANTIO(ID_PLANTA, ID_FERTILIZANTE, DT_INI_PLANTIO, QTD_SEMENTE)
VALUES(3,2,'01/11/2021',200);

INSERT INTO TB_PLANTIO(ID_PLANTA, ID_FERTILIZANTE, DT_INI_PLANTIO, QTD_SEMENTE)
VALUES(4,4,'01/03/2021',5000);
```

Criação de tarefas com a chave estrangeira de usuários:

```
-- TABELA DE TAREFAS
CREATE TABLE TB_TAREFAS(
	ID_TAREFA INTEGER PRIMARY KEY NOT NULL IDENTITY(1,1),
	OBS_TAREFA VARCHAR(150),
	STATUS_TAREFA CHAR(1) NOT NULL  CHECK (STATUS_TAREFA IN ('P','F')),  -- P -
PENDENTE | F - FINALIZADA
	DT_REAL_TAREFA DATETIME, --DATA QUE A TAREFA FOI EXECUTADA
	DESCRICAO VARCHAR(80) NOT NULL,
	DT_PREVISTA DATETIME, -- DATA PREVISTA PARA REALIZAÇÃO DA TAREFA
	ID_USUARIO INTEGER NOT NULL,
	TIPO_TAREFA CHAR(1) NOT NULL, -- C - COLHEITA | A- AVALIAÇÃO
	FOREIGN KEY(ID_USUARIO) REFERENCES TB_USUARIO (ID_USUARIO)
);
```

Script de inserção de tarefas:

```
INSERT INTO TB_TAREFAS(DESCRICAO, ID_USUARIO, TIPO_TAREFA, STATUS_TAREFA)
VALUES('colheita alface',1,'C','P');

INSERT INTO TB_TAREFAS(DESCRICAO, ID_USUARIO, TIPO_TAREFA, STATUS_TAREFA)
VALUES('plantio',1,'P','F');

INSERT INTO TB_TAREFAS(DESCRICAO, ID_USUARIO, TIPO_TAREFA, STATUS_TAREFA)
VALUES('avaliação da produção',2,'A','P');

INSERT INTO TB_TAREFAS(DESCRICAO, ID_USUARIO, TIPO_TAREFA, STATUS_TAREFA)
VALUES('plantio',3,'P','F');

INSERT INTO TB_TAREFAS(DESCRICAO, ID_USUARIO, TIPO_TAREFA, STATUS_TAREFA)
VALUES('produção da produção',5,'A','P');
```

Tabela de vínculo entre plantio e tarefas respeitando as chaves estrangeiras:

```
-- VINCULO ENTRE PLANTIO E TAREFAS
CREATE TABLE TB_TAREF_PLANTIO(
	ID_TAREF_PLANTIO INTEGER PRIMARY KEY NOT NULL IDENTITY(1,1),
	ID_TAREFA INTEGER NOT NULL,
	ID_PLANTIO INTEGER NOT NULL,
	FOREIGN KEY(ID_TAREFA) REFERENCES TB_TAREFAS (ID_TAREFA),
	FOREIGN KEY(ID_PLANTIO) REFERENCES TB_PLANTIO (ID_PLANTIO)
);
```

Inserção dos dados do vínculo entre as duas tabelas (TAREFAS e PLANTIO):

```
INSERT INTO TB_TAREF_PLANTIO(ID_TAREFA, ID_PLANTIO)
VALUES(1,2);
INSERT INTO TB_TAREF_PLANTIO(ID_TAREFA, ID_PLANTIO)
VALUES(2,2);
INSERT INTO TB_TAREF_PLANTIO(ID_TAREFA, ID_PLANTIO)
VALUES(2,3);
INSERT INTO TB_TAREF_PLANTIO(ID_TAREFA, ID_PLANTIO)
VALUES(3,2);
INSERT INTO TB_TAREF_PLANTIO(ID_TAREFA, ID_PLANTIO)
VALUES(4,4);
```

2.2 Engenharia de Software II

Para a disciplina de engenharia de Software II realizamos a modelagem do software respeitando o modelo e a estrutura de código orientado a objetos, levantando os requisitos, documentando, criando regras de negócio e definindo as ações dos usuários(atores) dentro do software, através de diagramas UML.

2.2.1 Requisitos

Levantamos os requisitos através da técnica de *brainstorming* e criamos o escopo para a criação do projeto:

2.2.1.1 Contextualização

O sistema será utilizado por pequenos ou médios produtores agrícolas, que desejam ter as informações de plantio em um único lugar. A solução deverá ser simples, pois a ideia da empresa é justamente quebrar o paradigma de que só grandes agricultores podem ter acesso a tecnologia.

Será necessário realizar o cadastro das plantas cultivadas, seus fertilizantes, e em seguida as informações de plantio. Também será necessário emitir relatórios sobre as irrigações que ocorreram.

2.2.1.2 Objetivos e justificativa

Nosso objetivo é atender e superar as expectativas do cliente, pois sabemos a dificuldade que hoje enfrentamos na falta de acessibilidade para esses produtos para Micro e Pequenos Produtores. O Produto abrange toda a área da informática, sempre visando as novas tecnologias e os meios de acesso.

A finalidade é economizar recursos e tempo do produtor ajudando o mesmo a tomada de decisão, como por exemplo planejamento de recurso (irrigação), além de facilitar manutenções futuras na irrigação.

2.2.1.3 Evolução do sistema

Será criado uma ferramenta para agendar manutenções futuras diretamente de dentro do sistema. Essa agenda será compatível com os horários disponíveis pela nossa empresa de manutenção.

Também será criada uma tela para o planejamento do cultivo, com o intuito de gerar estimativas do processo de plantação. Podendo adicionar “Tarefas” previstas e seus status.

Teremos a opção de criação de tela para cadastrar histórico de insumos consumidos no plantio.

2.2.1.4 Macro funções

O sistema contará com dois módulos: Plantio (Cadastro de plantas, Tarefas, Plantio, Fertilizantes) e módulo Gerencial: Emissão de relatórios e Agendamento de manutenções.

2.2.1.5 Restrições

É necessário ter acesso ao um computador Windows 7 ou superior, licença SQL Server, é necessário também ter um irrigador integrado (para a geração dos relatórios) e ter a instalação do pacote JDK configurado.

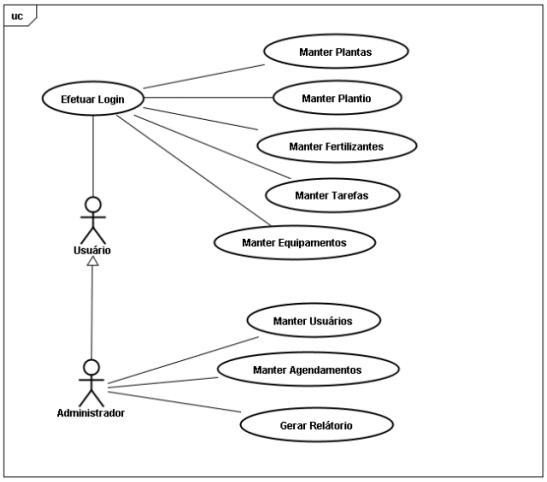
2.2.1.6 Benefícios

Os benefícios do sistema é ter um maior controle do que acontece durante o período do plantio, para aqueles que não tem grandes conhecimentos técnicos para utilizar ERP's complexos de agricultura. Além disso, ajuda a evitar desperdícios de insumos e auxilia na tomada de decisão.

2.2.2 Diagrama de Caso de Uso

O sistema terá dois usuários: usuário comum e usuário com acesso administrador. O usuário com controle adm, além das ações comuns, consegue manter os usuários, gerar relatórios e manter os agendamentos de manutenção do sistema.

Figura 3 Caso de Uso do sistema



Fonte: Própria

2.2.3 Narrativas

Para entendermos melhor o sistema, definimos os requisitos funcionais do sistema, definindo a prioridade, complexidade, risco, esforço e descrição, que estão listados na tabela abaixo:

Caso	Prioridade	Complexidade	Risco	Esforço (horas)	Observações
RF01	Crítico	Médio	Baixo	4	O usuário poderá cadastrar as plantas que posteriormente poderão ser vinculadas com os plantios;
RF02	Crítico	Alta	Médio	6	O usuário poderá realizar filtros e consultar as plantas cadastradas;
RF03	Crítico	Médio	Médio	5	O usuário poderá gerenciar os cadastros das plantas.
RF04	Importante	Médio	Baixo	4	O usuário poderá cadastrar fertilizantes que posteriormente poderão serem vinculadas com os plantios;
RF05	Importante	Alta	Médio	6	O usuário poderá realizar filtros e consultar dos fertilizantes
RF06	Importante	Médio	Médio	5	O usuário poderá gerenciar os cadastros dos fertilizantes
RF07	Crítico	Alta	Médio	6	O usuário poderá cadastrar plantios que posteriormente poderão serem vinculadas as tarefas;
RF08	Crítico	Alta	Médio	6	O usuário poderá realizar filtros e consultar dos plantios
RF09	Crítico	Alta	Médio	6	O usuário poderá gerenciar os cadastros dos plantios.
RF10	Útil	Médio	Baixo	3	O usuário poderá cadastrar tarefas;
RF11	Útil	Médio	Baixo	4	O usuário poderá realizar filtros e consultar as tarefas;
RF12	Útil	Médio	Médio	6	O usuário poderá gerenciar os cadastros das tarefas.
RF13	Crítico	Baixo	Baixo	3	Cadastros e gerenciamento de usuários;
RF14	Importante	Alta	Alta	8	Emissão de relatórios (relatório de tarefas e água);
RF15	Importante	Alta	Alta	8	Cadastro e gerenciamento de manutenções dos equipamentos e do sistema.

Tabela 1 Tabela do Planejamentos dos requisitos funcionais

Definimos também as regras de negócio usadas dentro do sistema, conforme a tabela abaixo:

Caso	Prioridade	Complexidade	Risco	Esforço (horas)	Observações
RN01	Crítico	Baixo	Baixo	2	Para Cadastrar plantio é necessário ter a planta já cadastrada.
RN02	Crítico	Baixo	Baixo	2	Não será permitido a exclusão de uma planta que está vinculada a um plantio ativo.
RN03	Crítico	Baixo	Baixo	2	Não será permitido a exclusão de fertilizante que está vinculado a um plantio ativo.
RN04	Importante	Baixo	Baixo	4	Não será permitido o cadastro de um item sem que os campos estejam preenchidos.
RN05	Útil	Baixo	Baixo	2	Não será permitido o cadastro de senha de usuário sem no mínimo 8 caracteres.

Tabela 2 Regras de Negócio

Para finalizar definimos os requisitos para a realização das ações dentro do software:

Caso	Prioridade	Complexidade	Risco	Esforço (horas)	Observações
RQ01	Importante	Médio	Baixo	4	Usabilidade - O sistema será intuitivo e simples para uso do usuário.
RQ02	Crítico	Alta	Alta	8	Confiabilidade – O sistema terá um controle de erro de logs, mostrando em que momento ocorreu a falha.
RQ03	Útil	Médio	Médio	4	Desempenho – A geração de relatórios será interrompida quando o tempo exceder 10 segundos.
RQ04	Útil	Alta	Alta	8	Design – O sistema terá um layout simples e intuitivo para auxiliar o usuário em sua navegação na interface.
RQ05	Útil	Baixo	Baixo	2	Padronização – O sistema é elaborado seguindo um padrão de nomenclatura das tabelas e variáveis no desenvolvimento.

Tabela 3 Requisitos do sistema

2.2.4 Diagrama de Classes

Para a criação de um software orientado a objetos criamos o diagrama abordando as classes descritas abaixo:

Classe Usuário: Classe responsável por conter as informações dos usuários do sistema.

Classe Agendamentos: Classe responsável por conter os registros relacionados aos agendamentos cadastrados.

Classe Equipamento: Classe responsável por conter os registros relacionados aos equipamentos cadastrados.

Classe Tarefa: Classe responsável por conter os registros relacionados as tarefas cadastradas.

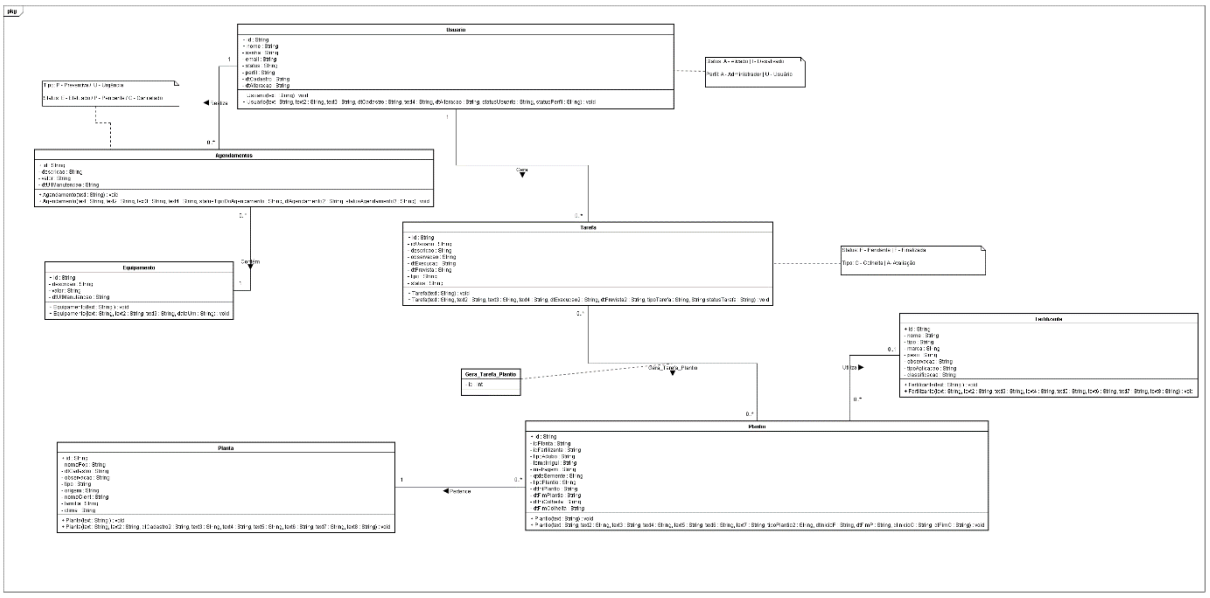
Classe Plantio: Classe responsável por conter os registros relacionados aos plantios cadastrados.

Classe Planta: Classe responsável por conter os registros relacionados as plantas cadastradas.

Classe Fertilizante: Classe responsável por conter os registros relacionados aos fertilizantes cadastrados.

Classe Associativa Gera_Tarefa_Plantio: Classe responsável em associar tarefa ao plantio.

Figura 4 Diagrama de Classes

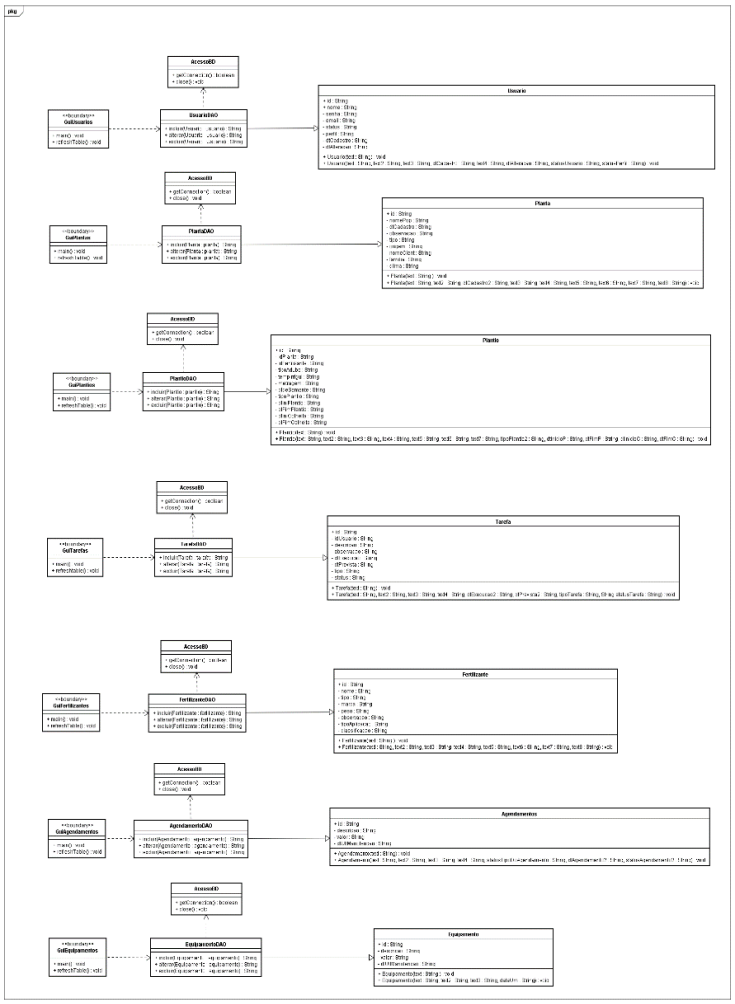


Fonte: Própria

No Diagrama representado abaixo podemos visualizar o diagrama de classes representando as classes DAO’s do sistema e seus CRUD’s.

Figura 5 Diagrama de Classes dos CRUD's

Figura 5 Diagrama de Classes dos CRUD's



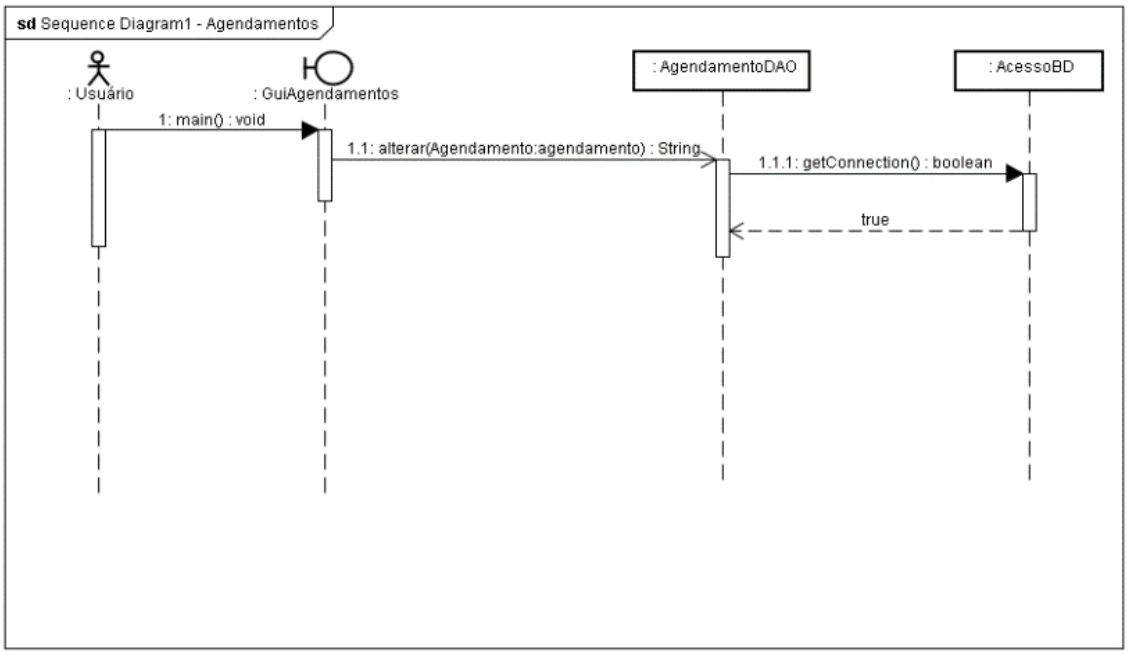
Fonte: Própria

2.2.5 Diagrama de Sequência

Para auxiliar o melhor entendimento das sequências do desenvolvimento do sistema, elaboramos os diagramas de sequência das classes de Agendamento e Plantas. No diagrama podemos ver as chamadas dentro do sistema e seus retornos.

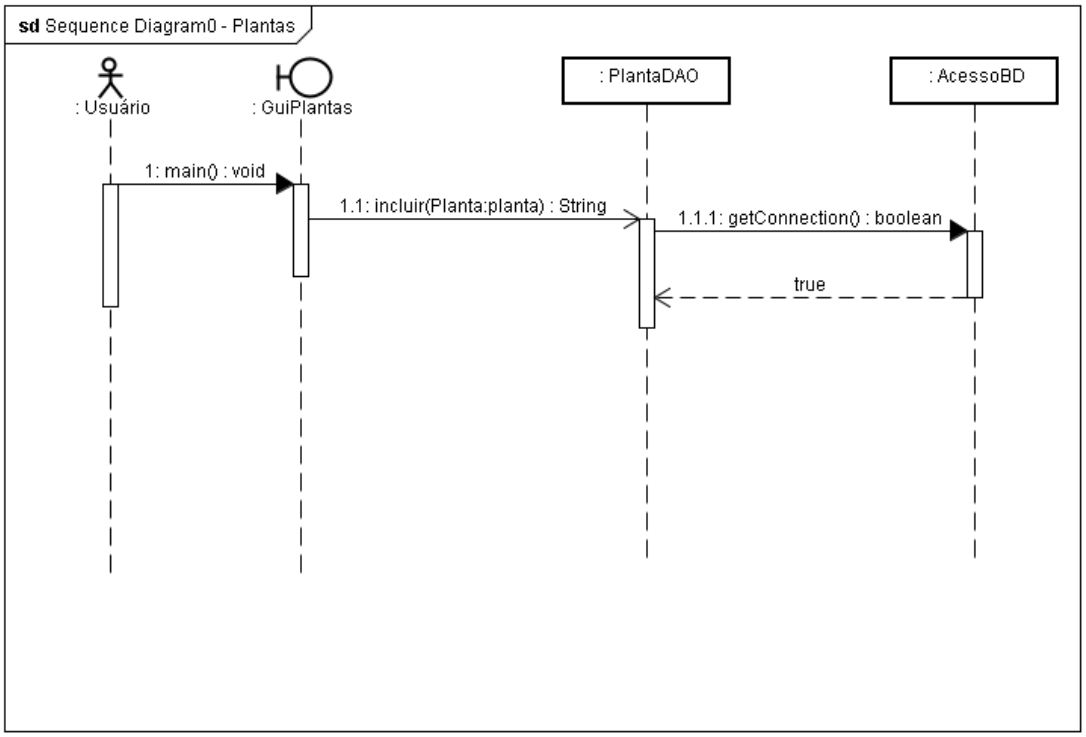
Figura 6 Diagrama de Sequência da Classe de Agendamento

Figura 6 Diagrama de Sequência da Classe de Agendamento



Fonte: Própria

Figura 7 Diagrama de Sequência da Classe Plantas



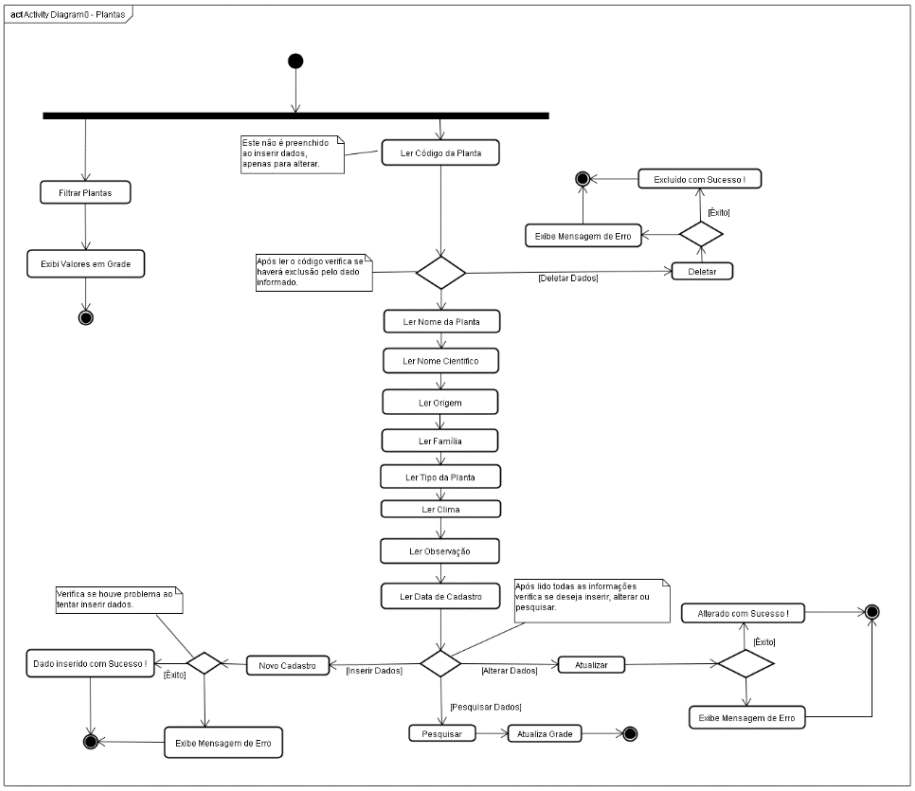
Fonte: Própria

2.2.6 Diagrama de Atividades

O diagrama de atividades tem o papel de ilustrar a dinâmica do software, modelando o fluxo de controle de atividade por atividade. Para representarmos,

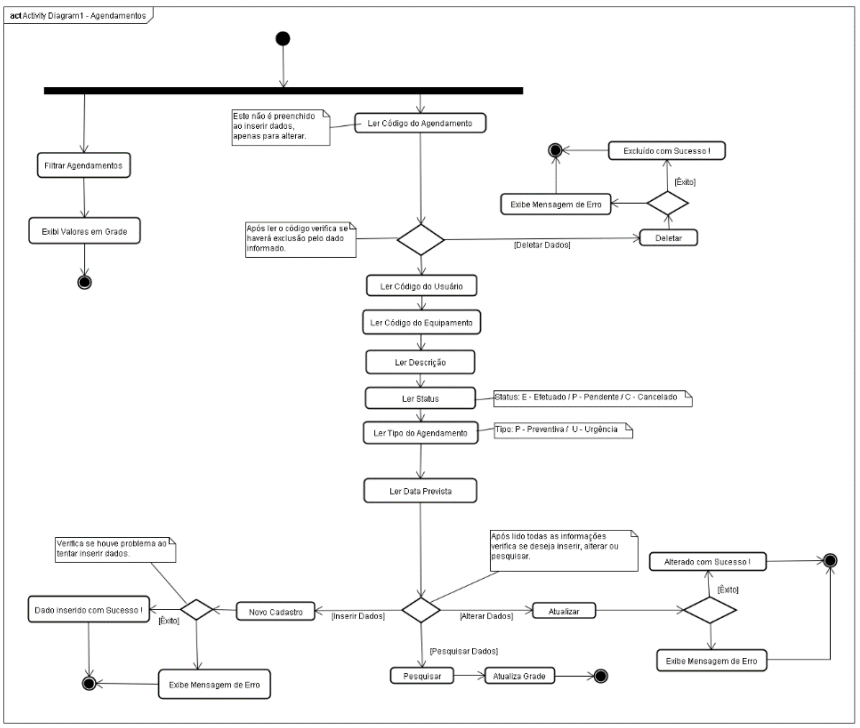
criamos os diagramas de atividades das classes de Agendamentos e Plantas conforme as imagens a seguir:

Figura 8 Diagrama de Atividades da Classe de agendamentos



Fonte: Própria

Figura 9 Diagrama de Atividades da Classe Planta



Fonte: Própria

2.3 Gestão e Governança de Tecnologia da Informação

Para a disciplina de Gestão e Governança de TI, realizamos o cálculo do Custo Total de Investimento (TCO) com o projeto, prisma do cliente e o vendedor.

Além do Cálculo do Retorno do Investimento (ROI) baseado com TCO feito anteriormente, considerando 3 cenários de comercialização: Ótimo, Médio, Ruim (sob mesmos prismas).

Para essa disciplina também realizamos o versionamento do software desenvolvido na matéria de programação Orientada a Objetos na ferramenta GIT HUB.

2.3.1 Custo Total de Investimento (TCO)

TCO (*Total Cost Ownership*) significa Custo Total de Propriedades. Com ele é possível analisar os custos de investimentos para aquisição de equipamentos ou produtos, passando assim maior confiança para os investidores.

Para calcula-lo devemos considerar todos os custos da empresa, como por exemplo folha de pagamento, conta de energia, água, ativos, depreciação, entre outros.

Além dos custos da empresa, há também o custo de produção, venda e cliente final, onde em cada processo é identificado o valor para cada etapa e lucro, podendo auxiliar o investidor se é viável ou não o investimento.

2.3.2 Retorno Sobre Investimento (ROI)

O ROI (*Return Over Investiment*) significa Retorno Sobre Investimento. Trata-se de um Indicador gerencial usado para saber qual foi o resultado financeiro de um investimento realizado. Com planos de amortização onde podem ser verificados em três cenários: pessimista, realista e otimista.

Usando o ROI, empresa e investidor conseguem ter controle exato sobre os meses onde terão recuo na quantidade de vendas ou aumento. Outro exemplo são os meses onde a empresa tem mais gastos ou até mesmo menos gastos.

Também é possível identificar a necessidade da redução de custos operacionais, podendo ser até a renovação de tecnologia com objetivos de diminuir o tempo ou o custo para execução.

2.3.3 Versionamento

Versionamento de software é um processo para o controle de versões que é

realizado para o acompanhamento das numerações que se modificam a cada novo lançamento. Estes números de versão geralmente são atribuídos em ordem crescente e correspondem a atualizações, inclusões de ferramentas e recursos ou exclusões para melhorar o seu funcionamento. Alguns softwares possuem números de versão internos (controle dos desenvolvedores) que divergem dos números de versão do produto (para conhecimento do usuário). O versionamento de código geralmente é feito com a utilização de ferramentas que permitem o controle de versão, de forma que diferentes usuários possam realizar alterações no código fonte de forma concorrente, liberando todas as alterações em um repositório de dados único. É uma metodologia importante para impedir a ocorrência de confusão sobre as versões dos softwares que estão em uso e evitar ataques cibernéticos.

É muito útil para os programadores que estão sempre atentos às modificações realizadas nos softwares com o objetivo de propiciar o uso de ferramentas otimizadas.

Com o versionamento, números únicos são atribuídos a cada nova versão dos programas, hardwares, drivers, firmwares e arquivos. Sempre que uma atualização é implementada, a numeração aumenta para tornar possível a identificação da versão mais moderna. Desta forma, os usuários conseguem comparar as versões instaladas em seus computadores ou celulares e realizar update quando desejarem.

É um método classificatório adotado por profissionais especializados que controlam e acompanham os históricos de atualização dos softwares para visualizar mudanças. Necessita ser monitorado por que as atualizações interferem na performance dos programas. As novidades podem ser percebidas pelos usuários finais que não exercem atividades relacionadas, mas querem fazer uso de ferramentas aprimoradas.

2.3.3.1 Vantagens

Um versionamento correto permite a otimização do processo de desenvolvimento e potencializa a segurança da informação dentro de uma empresa. Abaixo, mais algumas vantagens importantes.

- Análise de ameaças;
- Reparação de bugs;
- Alterações de arquitetura;
- Potencialização do trabalho colaborativo;
- Upgrades.

2.3.3.2 Versionamento Semântico

O versionamento semântico é um conjunto de regras e requerimentos para atribuição de versão de software e quando realizado de forma correta, ajuda os usuários a entenderem o estágio em que a aplicação está. Esses números são divididos em 3 grupos: *major version* (versão principal), *minor version* (versão secundária) e *patch version* (versão de correção) respectivamente. Onde cada um representa:

- ***Major* (Maior)**: indica a versão atual da interface pública do pacote. Isso deve ser incrementado toda vez que você fizer uma alteração que forçaria os usuários de seu pacote a atualizar seus próprios trabalhos.
- ***Minor* (Menor)**: este número descreve a versão funcional atual do seu software. Isso é incrementado cada vez que você adiciona uma nova funcionalidade, mas não altere de outra forma a interface do seu pacote. Ele comunica aos usuários que uma mudança significativa foi feita, mas o pacote permanece totalmente compatível com o problema menor anterior.
- **Número do *patch***: O número do patch é incrementado cada vez que você faz uma pequena alteração que não afete a interface pública ou a funcionalidade geral do seu pacote. Isso é mais usado para correções de bugs. Os consumidores devem sempre poder atualizar para a versão mais recente do *patch* sem hesitação.
- **Pré-*release*:** é uma versão candidata, com algumas instabilidades pois pode ter incompatibilidade no pacote.

Figura 10 Regras de versionamento

Fonte: imasters

2.3.3.3 Regras do versionamento

Todo padrão tem regras, e com o versionamento semântico não poderia ser diferente. Segue abaixo as 11 regras:

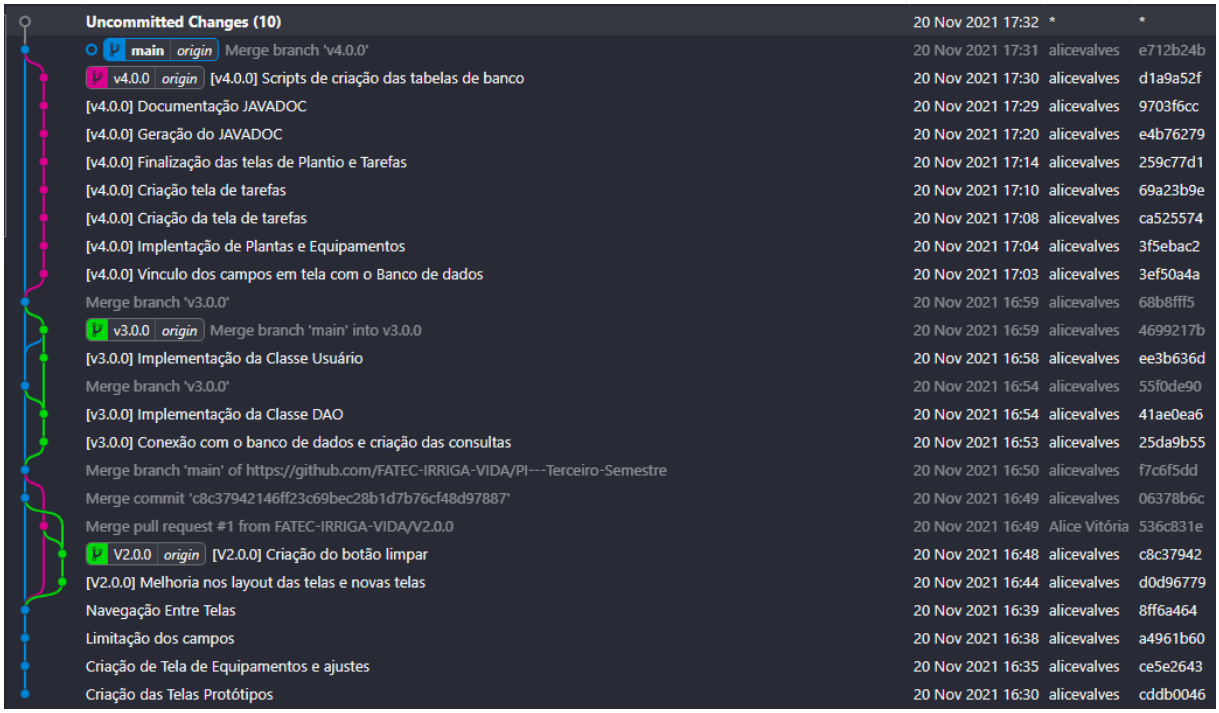
- **API pública:** Todo projeto que use versionamento semântico deve definir uma API pública, em pelo menos uma parte dela.
- **Formato do número de versão:** Todo projeto que use versionamento semântico deve conter versões no formato X.Y.Z. Sendo que X, Y e Z devem ser inteiros positivos e sem conter zeros à esquerda. (1.0.0 é correto, porém 01.0.0 é errado)
- **Versões imutáveis:** Uma vez modificada uma versão, ela não pode ser modificada de nenhuma maneira, qualquer alteração deve entrar como *major, minor* ou *path version.*
- **Desenvolvimento inicial:** Todo projeto que está em desenvolvimento inicial, deve iniciar o versionamento com a major version em 0, no caso (0.1.0), e ela também não será considerada uma versão estável, versões estáveis começarão em (1.0.0).
- **Versão 1.0.0:** A partir desta versão, a API deve seguir uma estrutura em cima da mesma, ou seja, toda versão futura deverá ser baseada ou dependente desta.
- **Versão de remendo (*patch version*):** Apenas deverá mudar caso novos bugs sejam corrigidos, e não poderá quebrar a aplicação de forma alguma, ou seja, a versão 1.0.1 deverá se comportar da mesma forma que a 1.0.0.
- **Versão secundária (*minor version*):** Deverá ser incrementada sob 3 condições: Quando funcionalidades novas e compatíveis com as versões atuais forem lançadas, quando uma funcionalidade é marcada como *deprecated* ou quando melhorias são inseridas de forma privada (sem acesso à partir da API pública).
- **Versão principal (*major version*):** Quando funcionalidades que quebrarem a aplicação forem inseridas, a *major version* deverá ser mudada, e a minor

e *path version* zeradas, no caso 1.9.9 passará a ser 2.0.0, e não 2.9.9 ou 2.0.9.

- **Versões de pré-lançamento (*pré-release versions*):** São versões futuras que ainda não estão estáveis o suficiente, podendo se tornar algo como 1.0.0-*alpha* ou 1.0.0-*beta*, entre outros.
- **Sobre dados de *build*:** Segundo essa regra, você pode, se quiser, adicionar informações extras de build junto à versão usando "+" seguido das informações adicionais separadas por pontos e que contem apenas caracteres alfanuméricos e hífen.
- **Sobre precedências de versão:** Esta regra define qual versão tem precedência sob outra, o critério usado para definir isto são:
  - Cada identificador (separado por ponto) é comparado da esquerda para a direita, até que uma diferença seja encontrada;
  - Identificadores compostos apenas por números são comparados numericamente;
  - Identificadores contendo letras ou hifens são comparados lexicalmente conforme a organização da tabela ASCII, ou seja "A < a < b".
  - Se os identificadores forem misturados entre numéricos e não-numéricos, então não-numéricos tem mais precedência que numéricos;
  - E por fim, uma versão com maior quantidade de identificadores tem mais precedência do que uma versão com menor quantidade de identificadores, no caso da versão com maior quantidade de identificadores ter todos os identificadores da versão com menor quantidade de identificadores na mesma posição (ou seja, 1.0.0-alpha < 1.0.0-alpha.1);

Para o projeto desse semestre realizamos o versionamento usando a *Branch* do GitHub, na imagem abaixo podemos visualizar os *commits*:

Figura 11 Gráfico dos commits gerados pelo GitGraph extensão do VSCode



Fonte: Própria

Para acessar os arquivos commitados basta acessar: https://github.com/FATEC-IRRIGA-VIDA/PI---Terceiro-Semestre

2.4 Programação Orientada a Objetos

Esse semestre desenvolvemos o softawer utilizando técnicas de programação orientada a objeto utilizando a linguagem de programação JAVA, abordada em aula pelo professor Sérgio Furgeri.

2.4.1 Protótipo da Interface do Usuário

Para nos auxiliar no desenvolvimento do sistema desenvolvemos protótipos da interface:

Tela 1 – LOGIN: Tela para efetuar o login no sistema, que ao informar login e senha correspondentes será redirecionado para a tela inicial.

Figura 12 Protótipo da Tela de Login

Fonte: Própria

Tela 2 - TELA INICIAL: Com objetivo de escolher a ação no caso dos usuários e administradores.

Figura 13 Protótipo da Tela Inicial

Fonte: Própria

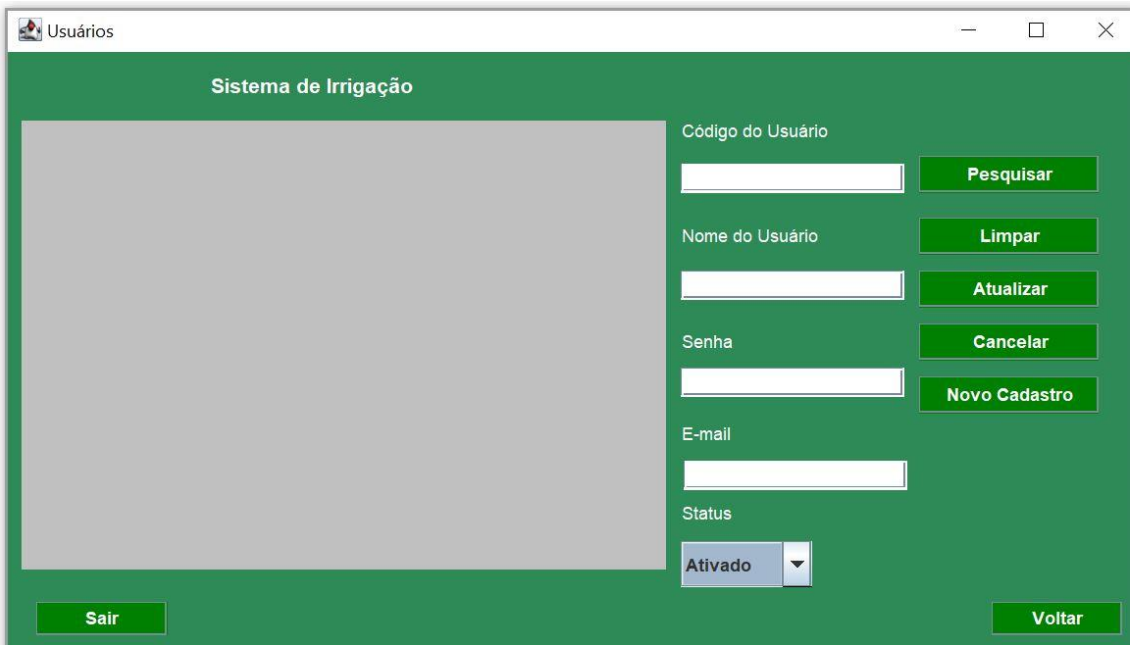
Tela 3 – Cadastros: Tela criada para selecionar qual opção é desejada, sendo eles: Usuários, Plantas, Plantios, Tarefas, Fertilizantes, Agendamentos, Equipamentos. Cada um desses botões redireciona para a tela correspondente.

Figura 14 Protótipo da Tela de Cadastros

Fonte: Própria

Tela 4 – Usuários: Tela para visualização, cadastro, alteração e exclusão de usuários. Tela só disponível para usuários do tipo administrador.

Figura 15 Protótipo Cadastro de Usuário



Fonte: Própria

Tela 5 – Plantas: Tela para visualização, cadastro, alteração e exclusão de plantas.

Figura 16 Protótipo Cadastro de Plantas

Fonte: Própria

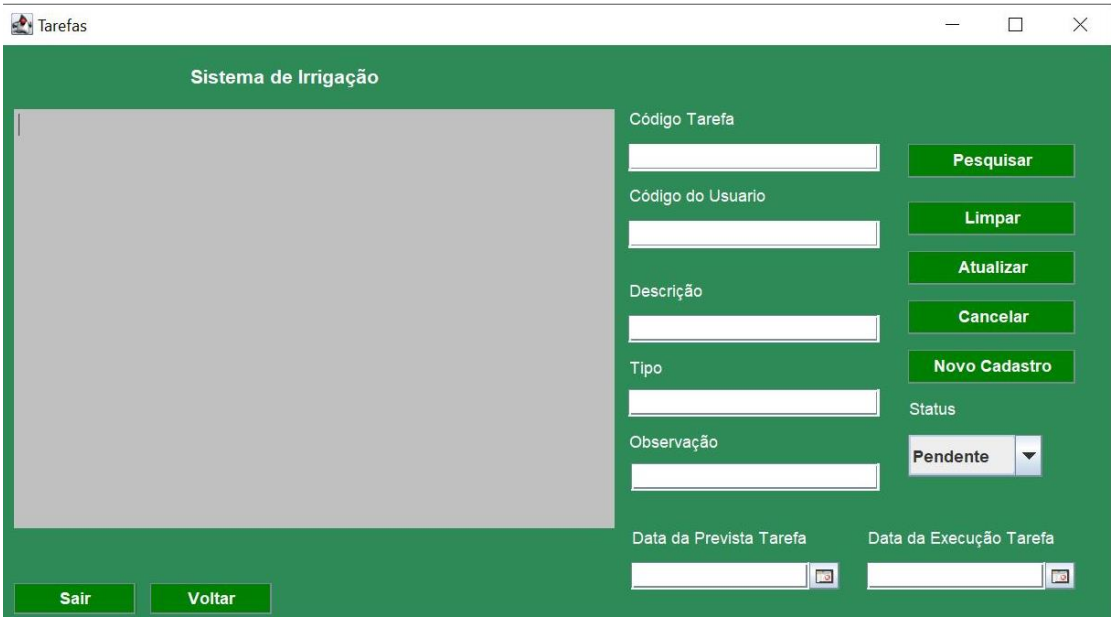
Tela 6 – Plantios: Tela para visualização, cadastro, alteração e exclusão de plantios.

Figura 17 Protótipo Cadastro de Plantio

Fonte: Própria

Tela 7 – Tarefas: Tela para visualização, cadastro, alteração e exclusão de tarefas (vinculadas a plantio).

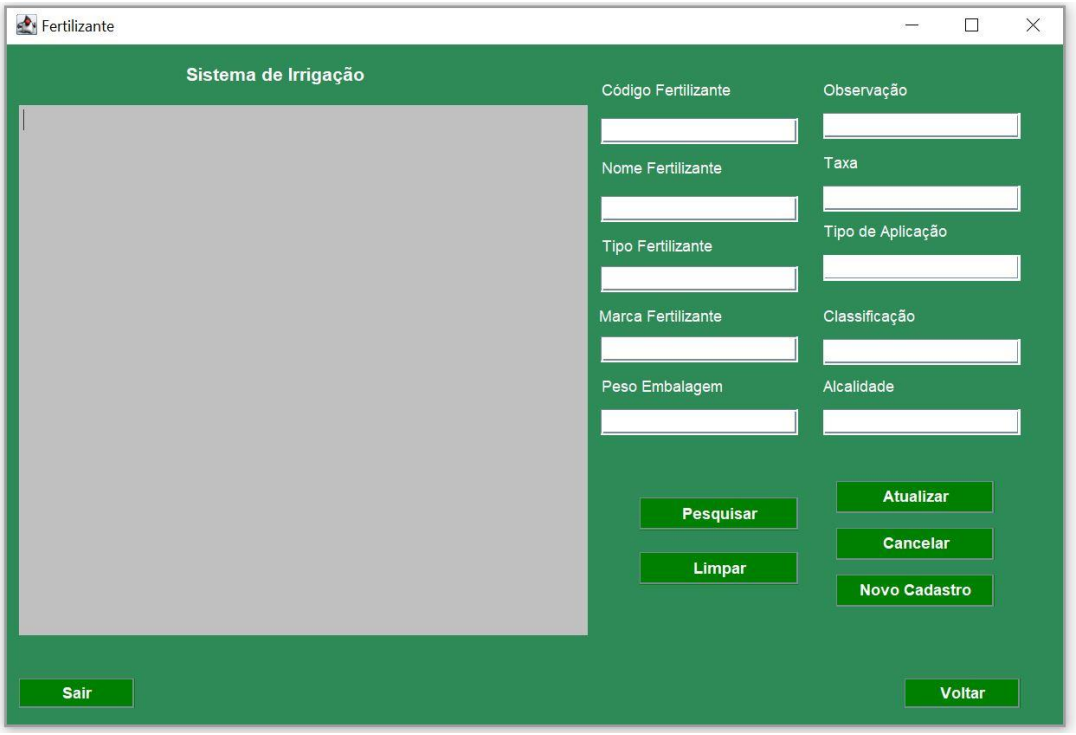
Figura 18 Protótipo Cadastro de Tarefas



Fonte: Própria

Tela 8 – Fertilizante: Tela para visualização, cadastro, alteração e exclusão de fertilizantes.

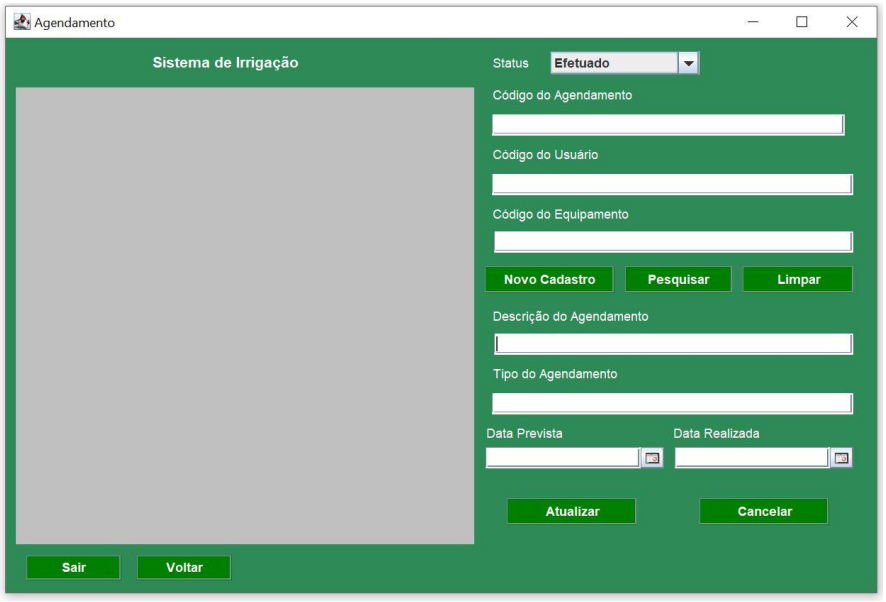
Figura 19 Protótipo Cadastro de Fertilizantes



Fonte: Própria

Tela 9 – Agendamento: Tela para visualização, cadastro, alteração e exclusão de agendamentos (Vinculado a equipamentos).

Figura 20 Protótipo Tela Controle Agendamento

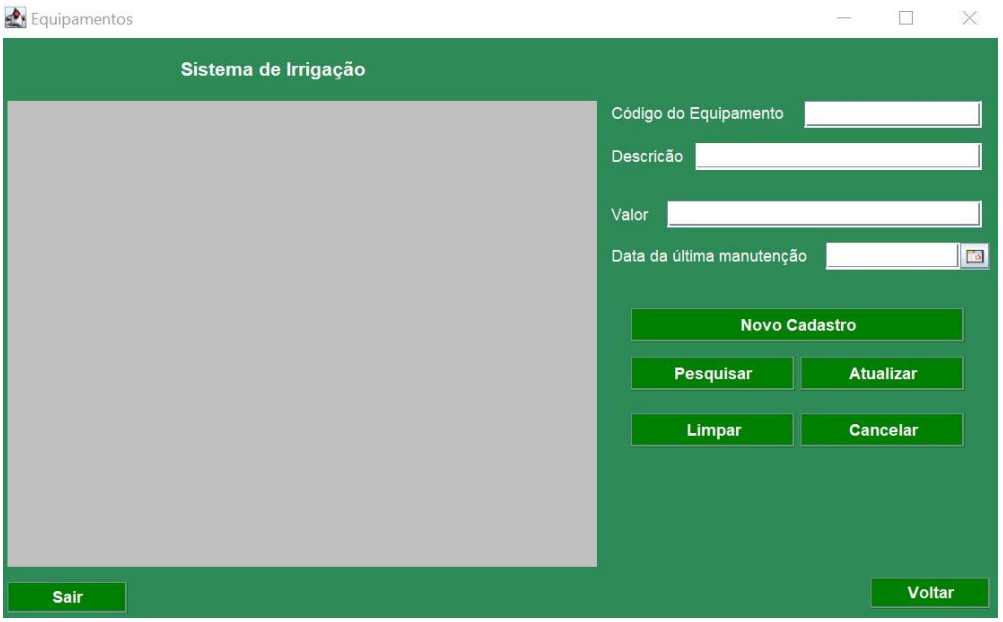


Fonte: Própria

.

Tela 10 – Equipamentos: Tela para visualização, cadastro, alteração e exclusão de equipamentos.

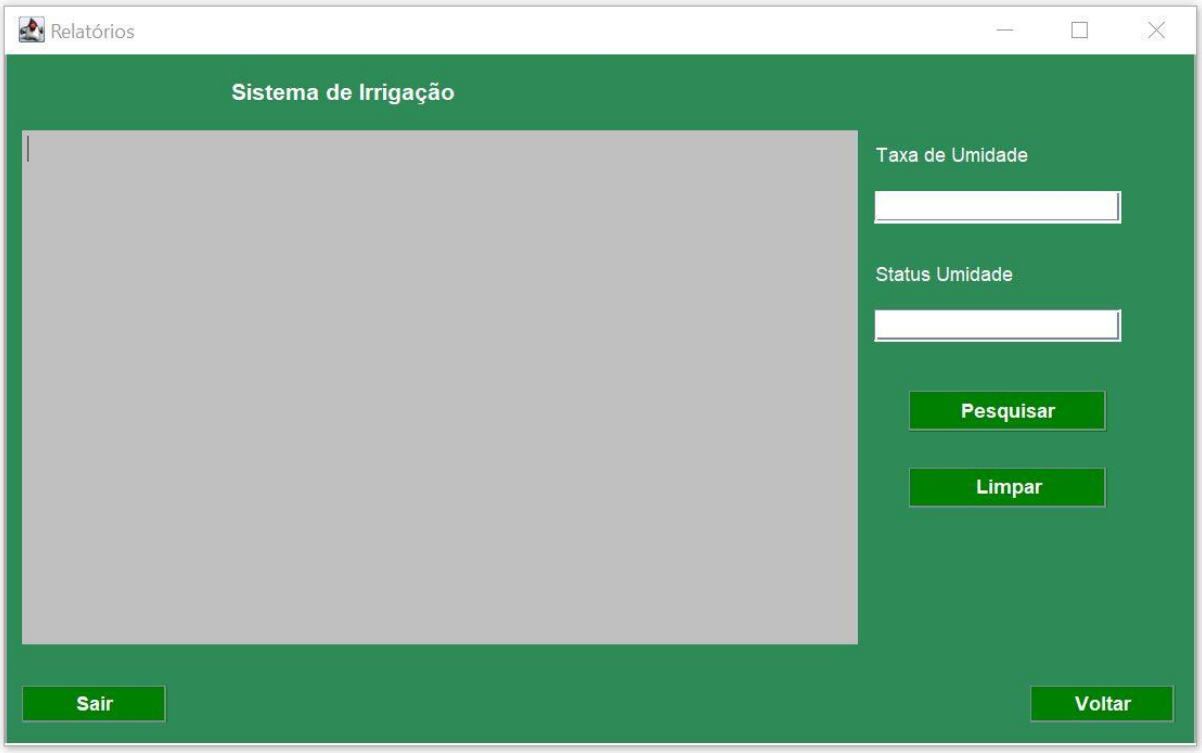
Figura 21 Protótipo Cadastro de Equipamentos



Fonte: Própria

Tela 11 – Relatórios: Tela para emissão de relatórios.

Figura 22 Protótipo Tela de Geração de Relatórios



Fonte: Própria

2.4.2 Codificação das classes de negócio

Usamos a linguagem JAVA para codificar as classes do nosso sistema usando a ferramenta *ECLIPSE.*

Iniciamos o desenvolvimento pela classe de usuário:

```
package metodos_projeto;

public class Usuario {

	public String id;
	public String nome;
	private String senha;
	private String email;
	private String status;
	private String perfil;
	private String dtCadastro;
	private String dtAlteracao;

	/**
	 * Método Construtor para receber o código do Usuário.
```

```
	 * @param text - variável que armazena o valor recebido do ID.
	 */
	public Usuario(String text) {
		this.id= text;
	}


	/**
	 * Método Construtor para receber todos os dados de um Usuário.
	 * @param text - variável que armazena o valor recebido do ID.
	 * @param text2 - variável que armazena o valor recebido do NOME.
	 * @param text3 - variável que armazena o valor recebido da SENHA.
	 * @param dtCadastro - variável que armazena o valor recebido da DATA
DE CADASTRO.
	 * @param text4 - variável que armazena o valor recebido dO EMAIL.
	 * @param dtAlteracao - variável que armazena o valor recebido da
DATA DE ALTERAÇÃO.
	 * @param statusUsuario - variável que armazena o valor recebido do
STATUS DO USUÁRIO.
	 * @param statusPerfil - variável que armazena o valor recebido do
PERFIL.
	 */
	public Usuario(String text, String text2, String text3, String
dtCadastro, String text4, String dtAlteracao,
			String statusUsuario, String statusPerfil) {
		this.id= text;
		this.nome=text2;
		this.dtCadastro=dtCadastro;
		this.senha=text3;
		this.email=text4;
		this.dtAlteracao=dtAlteracao;
		this.status=statusUsuario;
		this.perfil=statusPerfil;
	}


	/**
	 * Método que recebe a DATA DE CADASTRO.
	 * @return - retorna a DATA DE CADASTRO.
	 */
	public String getDtCadastro() {
		return dtCadastro;
	}

	/**
	 * Método que define a DATA DE CADASTRO.
	 * @param dtCadastro - variável que armazena a DATA DE CADASTRO.
	 */
	public void setDtCadastro(String dtCadastro) {
		this.dtCadastro = dtCadastro;
	}


	/**
	 * Método que recebe a DATA DE ALTERAÇÃO.
	 * @return - retorna a DATA DE ALTERAÇÃO.
	 */
	public String getDtAlteracao() {
		return dtAlteracao;
	}
```

```java
/**
 * Método que define a DATA DE ALTERAÇÃO.
 * @param dtAlteracao - variável que armazena a DATA DE ALTERAÇÃO.
 */
public void setDtAlteracao(String dtAlteracao) {
	this.dtAlteracao = dtAlteracao;
}


/**
 * Método que recebe o ID.
 * @return - retorna o ID.
 */
public String getId() {
	return id;
}


/**
 * Método que define o ID.
 * @param text - variável que armazena o ID.
 */
public void setId(String text) {
	this.id = text;
}


/**
 * Método que recebe o NOME.
 * @return - retorna o NOME.
 */
public String getNome() {
	return nome;
}

/**
 * Método que define o NOME.
 * @param text2 - variável que armazena o NOME.
 */
public void setNome(String text2) {
	this.nome = text2;
}

/**
 * Método que recebe a SENHA.
 * @return - retorna a SENHA.
 */
public String getSenha() {
	return senha;
}

/**
 * Método que define a SENHA.
 * @param text3 - variável que armazena a SENHA.
 */
public void setSenha(String text3) {
	this.senha = text3;
}
```

```
	/**
	 * Método que recebe o EMAIL.
	 * @return - retorna o EMAIL.
	 */
	public String getEmail() {
		return email;
	}

	/**
	 * Método que define o EMAIL.
	 * @param text4 - variável que armazena o EMAIL.
	 */
	public void setEmail(String text4) {
		this.email = text4;
	}

	/**
	 * Método que recebe o STATUS.
	 * @return - retorna o STATUS.
	 */
	public String getStatus() {
		return status;
	}

	/**
	 * Método que define o STATUS.
	 * @param statusUsuario - variável que armazena o STATUS.
	 */
	public void setStatus(String statusUsuario) {
		this.status = statusUsuario;
	}

	/**
	 * Método que recebe o PERFIL.
	 * @return - retorna o PERFIL.
	 */
	public String getPerfil() {
		return perfil;
	}

	/**
	 * Método que define o PERFIL.
	 * @param statusPerfil - variável que armazena o PERFIL.
	 */
	public void setPerfil(String statusPerfil) {
		this.perfil = statusPerfil;
	}
}
```

Em seguida realizamos o desenvolvimento da Classe de Planta:

```
package metodos_projeto;

public class Planta {


	public String id;
	private String nomePop;
	private String dtCadastro;
```

```
	private String observacao;
	private String tipo;
	private String origem;
	private String nomeCient;
	private String familia;
	private String clima;

	/**
	 *  Método Construtor para receber todos os dados de uma Planta.
	 * @param text - variável que armazena o valor recebido do ID.
	 * @param text2 - variável que armazena o valor recebido do NOME
POUPULAR.
	 * @param dtCadastro2 - variável que armazena o valor recebido da
DATA DE CADASTRO.
	 * @param text3 - variável que armazena o valor recebido da
OBSERVAÇÃO.
	 * @param text4 - variável que armazena o valor recebido do TIPO DA
PLANTA.
	 * @param text5 - variável que armazena o valor recebido da ORIGEM.
	 * @param text6 - variável que armazena o valor recebido do NOME
CIENTIFÍCO.
	 * @param text7 - variável que armazena o valor recebido da FAMÍLIA.
	 * @param text8 - variável que armazena o valor recebido do CLIMA.
	 */
	public Planta(String text, String text2, String dtCadastro2, String
text3, String text4, String text5, String text6,
			String text7, String text8) {
		this.id=text;
		this.nomePop=text2;
		this.dtCadastro=dtCadastro2;
		this.observacao=text3;
		this.tipo=text4;
		this.origem=text5;
		this.nomeCient=text6;
		this.familia=text7;
		this.clima=text8;
	}

	/**
	 * Método Construtor para receber o código de uma Planta.
	 * @param text - variável que armazena o valor recebido do ID.
	 */
	public Planta(String text) {
		this.id=text;
	}

	/**
	 * Método que recebe o ID.
	 * @return - retorna o ID.
	 */
	public String getId() {
		return id;
	}

	/**
	 * Método que define o ID.
	 * @param text - variável que armazena o ID.
	 */
	public void setId(String text) {
		this.id = text;
	}
```

```java
/**
 * Método que recebe o NOME POUPULAR.
 * @return - retorna o NOME POUPULAR.
 */
public String getNomePop() {
	return nomePop;
}

/**
 * Método que define o NOME POUPULAR.
 * @param text2 - variável que armazena o NOME POUPULAR.
 */
public void setNomePop(String text2) {
	this.nomePop = text2;
}

/**
 * Método que recebe a DATA DE CADASTRO.
 * @return - retorna a DATA DE CADASTRO.
 */
public String getDtCadastro() {
	return dtCadastro;
}

/**
 * Método que define a DATA DE CADASTRO.
 * @param dtCadastro2 - variável que armazena a DATA DE CADASTRO.
 */
public void setDtCadastro(String dtCadastro2) {
	this.dtCadastro = dtCadastro2;
}

/**
 * Método que recebe a OBSERVAÇÃO.
 * @return - retorna a  OBSERVAÇÃO.
 */
public String getObservacao() {
	return observacao;
}

/**
 * Método que define a OBSERVAÇÃO.
 * @param text3 - variável que armazena a OBSERVAÇÃO.
 */
public void setObservacao(String text3) {
	this.observacao = text3;
}

/**
 * Método que recebe o TIPO DA PLANTA.
 * @return - retorna o TIPO DA PLANTA.
 */
public String getTipo() {
	return tipo;
}

/**
 * Método que define o TIPO DA PLANTA.
 * @param text4 - variável que armazena o TIPO DA PLANTA.
 */
```

```
public void setTipo(String text4) {
	this.tipo = text4;
}

/**
 * Método que recebe a ORIGEM.
 * @return - retorna a ORIGEM.
 */
public String getOrigem() {
	return origem;
}

/**
 * Método que define a ORIGEM.
 * @param text5 - variável que armazena a ORIGEM.
 */
public void setOrigem(String text5) {
	this.origem = text5;
}

/**
 * Método que recebe o NOME CIENTIFÍCO.
 * @return - retorna o NOME CIENTIFÍCO.
 */
public String getNomeCient() {
	return nomeCient;
}

/**
 * Método que define o NOME CIENTIFÍCO.
 * @param text6 - variável que armazena o NOME CIENTIFÍCO.
 */
public void setNomeCient(String text6) {
	this.nomeCient = text6;
}

/**
 * Método que recebe a FAMÍLIA.
 * @return - retorna a FAMÍLIA.
 */
public String getFamilia() {
	return familia;
}

/**
 * Método que define a FAMÍLIA.
 * @param text7 - variável que armazena a FAMÍLIA.
 */
public void setFamilia(String text7) {
	this.familia = text7;
}

/**
 * Método que recebe o CLIMA.
 * @return - retorna o CLIMA.
 */
public String getClima() {
	return clima;
}

/**
```

```
	 * Método que define o CLIMA.
	 * @param text8 - variável que armazena o CLIMA.
	 */
	public void setClima(String text8) {
		this.clima = text8;
	}
}
```

A próxima Classe é referente a Plantio, que contem relacionamento com outras classes como Planta e Fertilizantes:

```
package metodos_projeto;

public class Plantio {

	public String id;
	private String idPlanta;
	private String idFertilizante;
	private String tipoAdubo;
	private String tempirrigui;
	private String metragem;
	private String qtdeSemente;
	private String tipoPlantio;
	private String dtIniPlantio;
	private String dtFimPlantio;
	private String dtIniColheita;
	private String dtFimColheita;

	// ID_PLANTIO	ID_PLANTA	ID_FERTILIZANTE	TIPO_ADUBO
	TEMPO_MED_IRRIGA	METRAGEM	QTD_SEMENTE	TIPO_PLANTIO
	//	DT_INI_PLANTIO	DT_FIM_PLANTIO	DT_INI_COLHEITA
	DT_FIM_COLHEITA

	/**
	 * Método Construtor para receber todos os dados de um Plantio.
	 * @param text - variável que armazena o valor recebido do ID.
	 * @param text2 - variável que armazena o valor recebido do ID DA
PLANTA.
	 * @param text3 - variável que armazena o valor recebido do ID DO
FERTILIZANTE.
	 * @param text4 - variável que armazena o valor recebido do TIPO DO
ADUBO.
	 * @param text5 - variável que armazena o valor recebido do TEMPO
MÉDIO DE IRRIGAÇÃO.
	 * @param text6 - variável que armazena o valor recebido da METRAGEM.
	 * @param text7 - variável que armazena o valor recebido da
QUANTIDADE DE SEMENTES.
	 * @param tipoPlantio2 - variável que armazena o valor recebido do
TIPO DO PLANTIO.
	 * @param dtInicioP - variável que armazena o valor recebido da DATA
INICIO PLANTIO.
	 * @param dtFimP - variável que armazena o valor recebido da DATA FIM
PLANTIO.
	 * @param dtInicioC - variável que armazena o valor recebido da DATA
INICIO COLHEITA.
	 * @param dtFimC - variável que armazena o valor recebido da DATA FIM
COLHEITA.
	 */
```

```
	public Plantio(String text, String text2, String text3, String text4,
String text5, String text6, String text7,
			String tipoPlantio2, String dtInicioP, String dtFimP,
String dtInicioC, String dtFimC) {
		this.id=text;
		this.idPlanta=text2;
		this.idFertilizante=text3;
		this.tipoAdubo=text4;
		this.tempirrigui=text5;
		this.metragem=text6;
		this.qtdeSemente=text7;
		this.tipoPlantio=tipoPlantio2;
		this.dtIniPlantio=dtInicioP;
		this.dtFimPlantio=dtFimP;
		this.dtIniColheita=dtInicioC;
		this.dtFimColheita=dtFimC;
	}

	/**
	 * Método Construtor para receber o código de um Plantio.
	 * @param text - variável que armazena o valor recebido do ID.
	 */
	public Plantio(String text) {
		this.id=text;
	}

	/**
	 * Método que recebe o ID.
	 * @return - retorna o ID.
	 */
	public String getId() {
		return id;
	}
	/**
	 * Método que define o ID.
	 * @param text - variável que armazena o valor recebido do ID.
	 */
	public void setId(String text) {
		this.id = text;
	}


	/**
	 * Método que recebe o ID DA PLANTA.
	 * @return - retorna o ID DA PLANTA.
	 */
	public String getIdPlanta() {
		return idPlanta;
	}
	/**
	 * Método que define o ID DA PLANTA.
	 * @param text2 - variável que armazena o valor recebido do ID DA
PLANTA.
	 */
	public void setIdPlanta(String text2) {
		this.idPlanta = text2;
	}


	/**
	 * Método que recebe o ID DO FERTILIZANTE.
```

```
	 * @return - retorna o ID DO FERTILIZANTE.
	 */
	public String getIdFertilizante() {
		return idFertilizante;
	}
	/**
	 * Método que define o ID DO FERTILIZANTE.
	 * @param text3 - variável que armazena o valor recebido do ID DO
FERTILIZANTE.
	 */
	public void setIdFertilizante(String text3) {
		this.idFertilizante = text3;
	}


	/**
	 * Método que recebe o TIPO DO ADUBO.
	 * @return - retorna o TIPO DO ADUBO.
	 */
	public String getTipoAdubo() {
		return tipoAdubo;
	}
	/**
	 * Método que define o TIPO DO ADUBO.
	 * @param text4 - variável que armazena o valor recebido do TIPO DO
ADUBO.
	 */
	public void setTipoAdubo(String text4) {
		this.tipoAdubo = text4;
	}


	/**
	 * Método que recebe o TEMPO MÉDIO DE IRRIGAÇÃO.
	 * @return - retorna o TEMPO MÉDIO DE IRRIGAÇÃO.
	 */
	public String getTempirrigui() {
		return tempirrigui;
	}
	/**
	 * Método que define o TEMPO MÉDIO DE IRRIGAÇÃO.
	 * @param text5 - variável que armazena o valor recebido do TEMPO
MÉDIO DE IRRIGAÇÃO.
	 */
	public void setTempirrigui(String text5) {
		this.tempirrigui = text5;
	}


	/**
	 * Método que recebe a METRAGEM.
	 * @return - retorna a METRAGEM.
	 */
	public String getMetragem() {
		return metragem;
	}
	/**
	 * Método que define a METRAGEM.
	 * @param text6 - variável que armazena o valor recebido da METRAGEM.
	 */
	public void setMetragem(String text6) {
```

```
		this.metragem = text6;
	}


	/**
	 * Método que recebe a QUANTIDADE DE SEMENTES.
	 * @return - retorna a QUANTIDADE DE SEMENTES.
	 */
	public String getQtdeSemente() {
		return qtdeSemente;
	}
	/**
	 * Método que define a QUANTIDADE DE SEMENTES.
	 * @param text7 - variável que armazena o valor recebido da
QUANTIDADE DE SEMENTES.
	 */
	public void setQtdeSemente(String text7) {
		this.qtdeSemente = text7;
	}


	/**
	 * Método que recebe o TIPO DO PLANTIO.
	 * @return - retorna o TIPO DO PLANTIO.
	 */
	public String getTipoPlantio() {
		return tipoPlantio;
	}
	/**
	 * Método que define o TIPO DO PLANTIO.
	 * @param tipoPlantio2 - variável que armazena o valor recebido do
TIPO DO PLANTIO.
	 */
	public void setTipoPlantio(String tipoPlantio2) {
		this.tipoPlantio = tipoPlantio2;
	}


	/**
	 * Método que recebe a DATA INICIO PLANTIO.
	 * @return - retorna a DATA INICIO PLANTIO.
	 */
	public String getDtIniPlantio() {
		return dtIniPlantio;
	}
	/**
	 * Método que define a DATA INICIO PLANTIO.
	 * @param dtInicioP - variável que armazena o valor recebido da DATA
INICIO PLANTIO.
	 */
	public void setDtIniPlantio(String dtInicioP) {
		this.dtIniPlantio = dtInicioP;
	}


	/**
	 * Método que recebe a DATA FIM PLANTIO.
	 * @return - retorna a DATA FIM PLANTIO.
	 */
	public String getDtFimPlantio() {
		return dtFimPlantio;
```

```
	}
	/**
	 * Método que define a DATA FIM PLANTIO.
	 * @param dtFimP - variável que armazena o valor recebido a DATA FIM
PLANTIO.
	 */
	public void setDtFimPlantio(String dtFimP) {
		this.dtFimPlantio = dtFimP;
	}


	/**
	 * Método que recebe a DATA INICIO COLHEITA.
	 * @return - retorna a DATA INICIO COLHEITA.
	 */
	public String getDtIniColheita() {
		return dtIniColheita;
	}
	/**
	 * Método que define a DATA INICIO COLHEITA.
	 * @param dtInicioC - variável que armazena o valor recebido da DATA
INICIO COLHEITA.
	 */
	public void setDtIniColheita(String dtInicioC) {
		this.dtIniColheita = dtInicioC;
	}


	/**
	 * Método que recebe a DATA FIM COLHEITA.
	 * @return - retorna a DATA FIM COLHEITA.
	 */
	public String getDtFimColheita() {
		return dtFimColheita;
	}
	/**
	 * Método que define a DATA FIM COLHEITA.
	 * @param dtFimC - variável que armazena o valor recebido da DATA FIM
COLHEITA.
	 */
	public void setDtFimColheita(String dtFimC) {
		this.dtFimColheita = dtFimC;
	}


}
```

Em seguida realizamos a classe de Tarefas, que também é uma classe com vários relacionamentos:

```
package metodos_projeto;

public class Tarefa {


	public String id;
	private String idUsuario;
	private String descricao;
	private String observacao;
	private String dtExecucao;
```

```
	private String dtPrevista;
	private String tipo;
	private String status;

	// ID_TAREFA	ID_USUARIO	DESCRICAO	OBS_TAREFA	DT_REAL_TAREFA
	DT_PREVISTA TIPO_TAREFA STATUS_TAREFA

	/**
	 * Método Construtor para receber todos os dados de uma Tarefa.
	 * @param text - variável que armazena o valor recebido do ID.
	 * @param text2 - variável que armazena o valor recebido do ID DO
USUÁRIO.
	 * @param text3 - variável que armazena o valor recebido da
DESCRIÇÃO.
	 * @param text4 - variável que armazena o valor recebido da
OBSERVAÇÃO DA TAREFA.
	 * @param dtExecucao2 - variável que armazena o valor recebido da
DATA REAL DA TAREFA.
	 * @param dtPrevista2 - variável que armazena o valor recebido da
DATA PREVISTA.
	 * @param tipoTarefa - variável que armazena o valor recebido do TIPO
DA TAREFA.
	 * @param statusTarefa - variável que armazena o valor recebido do
STATUS DA  TAREFA.
	 */
	public Tarefa(String text, String text2, String text3, String text4,
String dtExecucao2, String dtPrevista2,
			String tipoTarefa, String statusTarefa) {
		this.id=text;
		this.idUsuario=text2;
		this.descricao=text3;
		this.observacao=text4;
		this.dtExecucao=dtExecucao2;
		this.dtPrevista=dtPrevista2;
		this.tipo=tipoTarefa;
		this.status=statusTarefa;
	}

	/**
	 * Método Construtor para receber o código de uma Tarefa.
	 * @param text - variável que armazena o valor recebido do ID.
	 */
	public Tarefa(String text) {
		this.id=text;
	}


	/**
	 * Método que recebe o ID.
	 * @return - retorna o ID.
	 */
	public String getId() {
		return id;
	}

	/**
	 * Método que define o ID.
	 * @param text - variável que armazena o ID.
	 */
	public void setId(String text) {
		this.id = text;
```

```
}


/**
 * Método que recebe o ID DO USUÁRIO.
 * @return - retorna o ID DO USUÁRIO.
 */
public String getIdUsuario() {
	return idUsuario;
}

/**
 * Método que define o ID DO USUÁRIO.
 * @param text2 - variável que armazena o ID DO USUÁRIO.
 */
public void setIdUsuario(String text2) {
	this.idUsuario = text2;
}


/**
 * Método que recebe a DESCRIÇÃO.
 * @return - retorna a DESCRIÇÃO.
 */
public String getDescricao() {
	return descricao;
}

/**
 * Método que define a DESCRIÇÃO.
 * @param text3 - variável que armazena a DESCRIÇÃO.
 */
public void setDescricao(String text3) {
	this.descricao = text3;
}


/**
 * Método que recebe a OBSERVAÇÃO DA TAREFA.
 * @return - retorna a OBSERVAÇÃO DA TAREFA.
 */
public String getObservacao() {
	return observacao;
}

/**
 * Método que define a OBSERVAÇÃO DA TAREFA.
 * @param text4 - variável que armazena a OBSERVAÇÃO DA TAREFA.
 */
public void setObservacao(String text4) {
	this.observacao = text4;
}


/**
 * Método que recebe a DATA REAL DA TAREFA.
 * @return - retorna a DATA REAL DA TAREFA.
 */
public String getDtExecucao() {
	return dtExecucao;
}
```

```java
/**
 * Método que define a DATA REAL DA TAREFA.
 * @param dtExecucao2 - variável que armazena a DATA REAL DA TAREFA.
 */
public void setDtExecucao(String dtExecucao2) {
	this.dtExecucao = dtExecucao2;
}



/**
 * Método que recebe a DATA PREVISTA.
 * @return - retorna a DATA PREVISTA.
 */
public String getDtPrevista() {
	return dtPrevista;
}

/**
 * Método que define a DATA PREVISTA.
 * @param dtPrevista2 - variável que armazena a DATA PREVISTA.
 */
public void setDtPrevista(String dtPrevista2) {
	this.dtPrevista = dtPrevista2;
}


/**
 * Método que recebe o TIPO DA TAREFA.
 * @return - retorna o TIPO DA TAREFA.
 */
public String getTipo() {
	return tipo;
}

/**
 * Método que define o TIPO DA TAREFA.
 * @param tipoTarefa - variável que armazena o TIPO DA TAREFA.
 */
public void setTipo(String tipoTarefa) {
	this.tipo = tipoTarefa;
}


/**
 * Método que recebe o STATUS DA  TAREFA.
 * @return - retorna o STATUS DA  TAREFA.
 */
public String getStatus() {
	return status;
}

/**
 * Método que define o STATUS DA  TAREFA.
 * @param statusTarefa - variável que armazena o STATUS DA  TAREFA.
 */
public void setStatus(String statusTarefa) {
	this.status = statusTarefa;
}
```

```
}
```

Em seguida aplicamos a classe de Fertilizantes:

```
package metodos_projeto;

public class Fertilizante {

	public String id;
	private String nome;
	private String tipo;
	private String marca;
	private String peso;
	private String observacao;
	private String tipoAplicacao;
	private String classificacao;

	/**
	 * Método Construtor para receber todos os dados de um Fertilizante.
	 * @param text - variável que armazena o valor recebido do ID.
	 * @param text2 - variável que armazena o valor recebido do NOME.
	 * @param text3 - variável que armazena o valor recebido do TIPO.
	 * @param text4 - variável que armazena o valor recebido da MARCA.
	 * @param text5 - variável que armazena o valor recebido do PESO DA
EMBALAGEM.
	 * @param text6 - variável que armazena o valor recebido da
OBSERVAÇÃO.
	 * @param text7 - variável que armazena o valor recebido do TIPO DE
APLICAÇÃO.
	 * @param text8 - variável que armazena o valor recebido da
CLASSIICAÇÃO.
	 */
	public Fertilizante(String text, String text2, String text3, String
text4, String text5, String text6, String text7,
			String text8) {
		this.id = text;
		this.nome = text2;
		this.tipo=text3;
		this.marca=text4;
		this.peso=text5;
		this.observacao=text6;
		this.tipoAplicacao=text7;
		this.classificacao=text8;
	}

	/**
	 * Método Construtor para receber o código de um Fertilizante.
	 * @param text - variável que armazena o valor recebido do ID.
	 */
	public Fertilizante(String text) {
		this.id=text;
	}

	/**
	 * Método que recebe o ID.
	 * @return - retorna o ID.
	 */
	public String getId() {
		return id;
	}
```

```java
/**
 * Método que define o ID.
 * @param text - variável que armazena o ID.
 */
public void setId(String text) {
	this.id = text;
}



/**
 * Método que recebe o NOME.
 * @return - retorna o NOME.
 */
public String getNome() {
	return nome;
}

/**
 * Método que define o NOME.
 * @param text2 - variável que armazena o NOME.
 */
public void setNome(String text2) {
	this.nome = text2;
}


/**
 * Método que recebe o TIPO.
 * @return - retorna o TIPO.
 */
public String getTipo() {
	return tipo;
}

/**
 * Método que define o TIPO.
 * @param text3 - variável que armazena o TIPO.
 */
public void setTipo(String text3) {
	this.tipo = text3;
}


/**
 * Método que recebe a MARCA.
 * @return - retorna a MARCA.
 */
public String getMarca() {
	return marca;
}

/**
 * Método que define a MARCA.
 * @param text4 - variável que armazena a MARCA.
 */
public void setMarca(String text4) {
	this.marca = text4;
}
```

```
/**
 * Método que recebe o PESO DA EMBALAGEM.
 * @return - retorna o PESO DA EMBALAGEM.
 */
public String getPeso() {
	return peso;
}

/**
 * Método que define o PESO DA EMBALAGEM.
 * @param text5 - variável que armazena o PESO DA EMBALAGEM.
 */
public void setPeso(String text5) {
	this.peso = text5;
}


/**
 * Método que recebe a OBSERVAÇÃO.
 * @return - retorna a OBSERVAÇÃO.
 */
public String getObservacao() {
	return observacao;
}

/**
 * Método que define a OBSERVAÇÃO.
 * @param text6 - variável que armazena a OBSERVAÇÃO.
 */
public void setObservacao(String text6) {
	this.observacao = text6;
}


/**
 * Método que recebe o TIPO DE APLICAÇÃO.
 * @return - retorna o TIPO DE APLICAÇÃO.
 */
public String getTipoAplicacao() {
	return tipoAplicacao;
}

/**
 * Método que define o TIPO DE APLICAÇÃO.
 * @param text7 - variável que armazena o TIPO DE APLICAÇÃO.
 */
public void setTipoAplicacao(String text7) {
	this.tipoAplicacao = text7;
}


/**
 * Método que recebe a CLASSIICAÇÃO.
 * @return - retorna a CLASSIICAÇÃO.
 */
public String getClassificacao() {
	return classificacao;
}

/**
 * Método que define a CLASSIICAÇÃO.
```

```
	 * @param text8 - variável que armazena a CLASSIICAÇÃO.
	 */
	public void setClassificacao(String text8) {
		this.classificacao = text8;
	}

}
```

Em seguida podemos ver a classe de Agendamento:

```
package metodos_projeto;

public class Agendamento {

	public String id;
	public String idUsuario;
	public String idEquipamento;
	private String descricao;
	private String tipoAgendamento;
	private String dtAgendamento;
	private String statusAgendamento;

	// ID_AGENDAMENTO ID_USUARIO  ID_EQUIPAMENTO    DESCRICAO   TIPO_AGEN
	DT_MARCADA_AGEND  STATUS_AGEN

	/**
	 * Método Construtor para receber todos os dados de um Agendamento.
	 * @param text - variável que armazena o valor recebido do ID.
	 * @param text2 - variável que armazena o valor recebido do ID DO
USUÁRIO.
	 * @param text3 - variável que armazena o valor recebido do ID DO
EQUIPAMENTO.
	 * @param text4 - variável que armazena o valor recebido da
DESCRIÇÃO.
	 * @param statusTipoDoAgendamento - variável que armazena o valor
recebido do TIPO DO AGENDAMENTO.
	 * @param dtAgendamento2 - variável que armazena o valor recebido da
DATA MARCADA DO AGENDAMENTO.
	 * @param statusAgendamento2 - variável que armazena o valor recebido
do STATUS DO AGENDAMENTO.
	 */
	public Agendamento(String text, String text2, String text3, String
text4, String statusTipoDoAgendamento,
			String dtAgendamento2, String statusAgendamento2) {
		this.id=text;
		this.idUsuario=text2;
		this.idEquipamento=text3;
		this.descricao=text4;
		this.tipoAgendamento=statusTipoDoAgendamento;
		this.dtAgendamento=dtAgendamento2;
		this.statusAgendamento=statusAgendamento2;
	}

	/**
	 * Método Construtor para receber o código de um Agendamento.
	 * @param text - variável que armazena o valor recebido do ID.
	 */
	public Agendamento(String text) {
		this.id=text;
	}
```

```
/**
 * Método que recebe o ID.
 * @return - retorna ID.
 */
public String getId() {
	return id;
}

/**
 * Método que define o ID.
 * @param text - variável que armazena o ID.
 */
public void setId(String text) {
	this.id = text;
}


/**
 *  Método que recebe o ID DO USUÁRIO.
 * @return - retorna ID DO USUÁRIO.
 */
public String getIdUsuario() {
	return idUsuario;
}

/**
 * Método que define o ID DO USUÁRIO.
 * @param text2 - variável que armazena o ID DO USUÁRIO.
 */
public void setIdUsuario(String text2) {
	this.idUsuario = text2;
}


/**
 *  Método que recebe o ID DO EQUIPAMENTO.
 * @return - retorna ID DO EQUIPAMENTO.
 */
public String getIdEquipamento() {
	return idEquipamento;
}

/**
 * Método que define o ID DO EQUIPAMENTO.
 * @param text3 - variável que armazena o ID DO EQUIPAMENTO.
 */
public void setIdEquipamento(String text3) {
	this.idEquipamento = text3;
}


/**
 *  Método que recebe a DESCRIÇÃO.
 * @return - retorna a DESCRIÇÃO.
 */
public String getDescricao() {
	return descricao;
}

/**
```

```
	 * Método que define a DESCRIÇÃO.
	 * @param text4 - variável que armazena a DESCRIÇÃO.
	 */
	public void setDescricao(String text4) {
		this.descricao = text4;
	}


	/**
	 *  Método que recebe o TIPO DO AGENDAMENTO.
	 * @return - retorna TIPO DO AGENDAMENTO.
	 */
	public String getTipoAgendamento() {
		return tipoAgendamento;
	}

	/**
	 * Método que define o TIPO DO AGENDAMENTO.
	 * @param statusTipoDoAgendamento - variável que armazena o TIPO DO
AGENDAMENTO.
	 */
	public void setTipoAgendamento(String statusTipoDoAgendamento) {
		this.tipoAgendamento = statusTipoDoAgendamento;
	}


	/**
	 *  Método que recebe a DATA MARCADA DO AGENDAMENTO.
	 * @return - retorna a DATA MARCADA DO AGENDAMENTO.
	 */
	public String getDtAgendamento() {
		return dtAgendamento;
	}

	/**
	 * Método que define a DATA MARCADA DO AGENDAMENTO.
	 * @param dtAgendamento2 - variável que armazena a DATA MARCADA DO
AGENDAMENTO.
	 */
	public void setDtAgendamento(String dtAgendamento2) {
		this.dtAgendamento = dtAgendamento2;
	}


	/**
	 *  Método que recebe o STATUS DO AGENDAMENTO.
	 * @return - retorna STATUS DO AGENDAMENTO.
	 */
	public String getStatusAgendamento() {
		return statusAgendamento;
	}

	/**
	 * Método que define o STATUS DO AGENDAMENTO.
	 * @param statusAgendamento2 - variável que armazena o STATUS DO
AGENDAMENTO.
	 */
	public void setStatusAgendamento(String statusAgendamento2) {
		this.statusAgendamento = statusAgendamento2;
	}
}
```

Abaixo podemos ver a classe de Equipamentos:

```
package metodos_projeto;

public class Equipamento {

	public String id;
	private String descricao;
	private String valor;
	private String dtUltManutencao;

	/**
	 *  Método Construtor para receber todos os dados de um Equipamento.
	 * @param text - variável que armazena o valor recebido do ID.
	 * @param text2 - variável que armazena o valor recebido da
DESCRIÇÃO.
	 * @param text3 - variável que armazena o valor recebido do VALOR.
	 * @param dataUm - variável que armazena o valor recebido da DATA DA
ÚLTIMA MANUTENÇÃO.
	 */
	public Equipamento(String text, String text2, String text3, String
dataUm) {
		this.id=text;
		this.descricao=text2;
		this.valor=text3;
		this.dtUltManutencao=dataUm;
	}

	/**
	 * Método Construtor para receber o código de um Equipamento.
	 * @param text - variável que armazena o valor recebido do ID.
	 */
	public Equipamento(String text) {
		this.id=text;
	}

	/**
	 * Método que recebe o ID.
	 * @return - retorna o ID.
	 */
	public String getId() {
		return id;
	}

	/**
	 * Método que define o ID.
	 * @param text - variável que armazena o ID.
	 */
	public void setId(String text) {
		this.id = text;
	}

	/**
	 * Método que recebe a DESCRIÇÃO.
	 * @return - retorna a DESCRIÇÃO.
	 */
	public String getDescricao() {
		return descricao;
	}
```

```
	/**
	 * Método que define a DESCRIÇÃO.
	 * @param text2 - variável que armazena a DESCRIÇÃO.
	 */
	public void setDescricao(String text2) {
		this.descricao = text2;
	}

	/**
	 * Método que recebe o VALOR.
	 * @return - retorna o VALOR.
	 */
	public String getValor() {
		return valor;
	}

	/**
	 * Método que define o VALOR.
	 * @param text3 - variável que armazena o VALOR.
	 */
	public void setValor(String text3) {
		this.valor = text3;
	}

	/**
	 * Método que recebe a DATA DA ÚLTIMA MANUTENÇÃO.
	 * @return - retorna a DATA DA ÚLTIMA MANUTENÇÃO.
	 */
	public String getDtUltManutencao() {
		return dtUltManutencao;
	}

	/**
	 * Método que define a DATA DA ÚLTIMA MANUTENÇÃO.
	 * @param dataUm - variável que armazena a DATA DA ÚLTIMA MANUTENÇÃO.
	 */
	public void setDtUltManutencao(String dataUm) {
		this.dtUltManutencao = dataUm;
	}
}
```

2.4.3 Documentação JAVADOC

Ao enviar o arquivo dessa documentação, estamos enviando uma pasta com o nome “javadoc” com documentação JAVADOC de todas as classes em PDF. A documentação completa também pode ser encontrada no nosso github:

https://github.com/FATEC-IRRIGA-VIDA/PI---Terceiro-Semestre/tree/main/Irriga%20Vida%20-%20Sistema%20de%20Irrigacao/Sistema%20-%20Irriga%20Vida%20-%20PI%20-%20ADS%20-%203/doc

2.4.4 Arquitetura

Para acessar a arquitetura do software basta acessar o link a seguir: https://github.com/FATEC-IRRIGA-VIDA/PI---Terceiro-Semestre

2.5 Inglês

Com a disciplina de inglês foi possível fazer a introdução do trabalho, no dia da apresentação para a banca, em inglês. Foi desenvolvido um video propaganda para divulgação do projeto.

Para visualizar o video acessar o link:

https://www.youtube.com/watch?v=WNEfq4FC6TM

2.6 Resumo em Português

A disciplina nos auxiliou no desenvolvimento do projeto em vários momentos, como por exemplo na leitura das documentações e termos utilizados nas matérias mais técnicas do curso, que em sua maioria só são encontradas em Inglês, no momento da criação do vídeo propaganda para auxiliar na divulgação da empresa e no *abstratc* que foi abordado no início do documento.

2.6.1 Resumo em Inglês

The matter helps us in the development of the project in many moments, for

example reading the documentation and terms used in the moments more technical in our course, mostly they are only found in English, in the moment of the creation of the advertising video to assist in the dissemination of the company and in the abstract that was approached at the beginning of this document.

2.7 Lições Aprendidas e Principais Dificuldades

Após três semestres continuamos a aprender a lidar com um mundo acadêmico totalmente remoto, adquirindo bagagens ainda mais concretas sobre as etapas de criação de um sistema.

Durante todo o desenvolvimento aprendemos etapas fundamentais para a carreira de qualquer profissional de tecnologia de informação. Na análise com a matéria de Engenharia de Software II até desenvolvimento de bancos e sistemas totalmente direcionados à objetos.

Além claro de aprendermos como funciona a gestão dentro da àrea o que é fundamental para o dia a dia.

Durante a elaboração do projeto, foi possível com o aprendizado e integração de todas as partes, desde da programação do código, planejamento do sistema e integração com um banco de dados. Também desenvolvemos a interface gráfica do projeto, dando vida por trás da codificação elaborada.

O maior aprendizado foi a colaboração entre os professores e os membros da equipe que, apesar de não ter sido possível desenvolver o projeto em modo presencial, tornou possível a realização deste trabalho.

2.8 Dificuldades

Realizar um trabalho de PI no meio de uma Pandemia realmente dificultou a realização do nosso trabalho, aprender a trabalhar de forma totalmente remota foi uma experiência que gerou muita dificuldade no primeiro momento.

Conciliar a vida acadêmica, profissional e pessoal foi um grande desafio nesse semestre, além das incertezas que esse semestre nos proporcionou fora e dentro da sala de aula.

Apesar de todas as dificuldades, conseguimos auxílio para desenvolver soluções para os nossos problemas, permitindo assim a realização deste trabalho.

2.9 Oportunidades de Melhoria

Durante todo o planejamento do projeto, foi possível verificar que poderiam ser feitas como uma oportunidade futura de melhoria, algumas modificações e acréscimos de tarefas para melhor agregar e atender os clientes, pois pensamos em novas funcionalidades dentro do sistema, além da possibilidade de uma restruturação do layout e na possibilidade de criarmos um sistema totalmente online para semestres futuros.

Também durante a análise para o *brainstorming*, o qual foi a principal ferramenta para o levantamento de requisitos em nosso sistema, conseguimos identificar algumas funções que seriam de muita valia para aplicação em nosso projeto, porém que para esse momento não seria possível colocar em ação devido ao prazo para entrega, mas, com essa ferramenta conseguimos montar uma base para que futuramente, conforme avançamos durante o curso e nos conteúdos, serão feitas essas inclusões.

3. CONCLUSÕES

Após pesquisas, onde registram que o desperdício de água é grande, verificamos algo que podemos mudar, priorizando a chegada desse investimento para os Micro e Pequenos Agricultores, cujo a necessidade de uma maior produtividade com baixo consumo, fazem com que procurem novos meios, assim nos encontrando para que possamos pensar no melhor projeto para cada tipo de Plantação. Com esse intuito, fundamos a Irriga Vida, para trazer uma vida melhor para empreendedores desse ramo, com preços baixos para estimular sua chegada no mercado.

Durante esse semestre continuamos de forma remota o que é um grande desafio a todos, mas que de forma geral foi muito positiva e proveitosa. Lidamos com novos desafios e tivemos muitas situações onde foi necessário implantação de ferramentas em nosso projeto em um curto espaço de tempo.

Em Gestão e Governança aprendemos como é a gestão dentro de uma empresa de TI, aprendemos a fazer a comercialização do software, definindo os pré-requisitos, formas de licenciamentos, versionamento e o retorno sobre o investimento.

Em Engenharia de Software II foi possível levantar todos os requisitos necessários para o início do programa, além de documentarmos e planejarmos atráves dos diagramas de classe e caso de uso.

Na disciplina de Programação Orientada a Objetos tivemos o contato com a linguagem JAVA, abordando técnicas de programação orientadas a objetos, onde foi possível conectar com o banco de dados.

Em inglês foi possível agregar vocabulário, nos ajudando a fazer a criação do vídeo e ter noções de comunicação.

Em Banco de Dados aprendemos a modelar um banco de dados, utilizando ferramentas como o brModelo. Criamos o DER e o MER do sistema, além de elaborarmos os scripts dentro do *SQL Server*.

Nossa equipe entregou cada um, um pedacinho de cada, onde juntos, formamos a Irriga Vida, uma empresa que pensa de forma sustentável no amanhã. Continuaremos nessa luta, sempre buscando mais inovação para que alcancemos nossos objetivos, concretizando nossos planos.

Apesar de todas dificuldades que foram enfrentadas durante esse semestre, diante de todo esse período de distanciamento social, consolidar e por em prática muitos conhecimentos adquiridos nas disciplinas cursadas, e ainda assim fazer com

que todas essas disciplinas estivessem interligadas em nosso projeto de forma harmônica e complementar.

Agradecemos imensamente a todos os professores envolvidos no auxílio desse projeto e aos nossos companheiros da turma de Análise e Desenvolvimento de Sistemas do 3° Semestre da Faculdade de Tecnologia Dr. Archimedes Lammoglia de Indaiatuba, que de alguma forma contribuíram. Somos eternamente gratos, a todos os funcionários da FATEC que puderam nos proporcionar o espaço, ferramentas e o estudo para tornar tudo isso realidade. Deixamos aqui o nosso muito obrigado a todos.

4. REFERÊNCIAS BIBLIOGRÁFICAS

Baptista, Xavier. 2020. *Interessantissimo.* [Online] 19 de 11 de 2020. [Citado em: 19 de 04 de 2021.] https://interessantissimo.pt/curiosidades/vantagens-desvantagens-c-sharp/.

Borges, Thiago. 2014. *Thiago Borges.* [Online] 06 de 06 de 2014. [Citado em: 19 de 04 de 2021.] https://thiagoborges.net.br/o-que-e-o-c-sharp/.

Caelum. *Caelum.* [Online] [Citado em: 19 de 04 de 2021.] https://www.caelum.com.br/apostila/apostila-csharp-orientacao-objetos.pdf.

Jr, Edson Amaral. 2020. iMasters. *iMasters.* [Online] 8 de 7 de 2020. [Citado em: 14 de 11 de 2021.] https://imasters.com.br/codigo/versionamento-semantico-o-que-e-e-como-usar.

Matsumota, Leonardo. 2018. *Leonardo Matsumota.* [Online] 19 de 06 de 2018. [Citado em: 08 de 05 de 2021.] https://leonardo-matsumota.com/2018/06/19/testes-unitarios-com-intellitest-no-visual-studio/.

Microsoft. 2021. *Microsoft.* [Online] 28 de 01 de 2021. [Citado em: 19 de 04 de 2021.] https://docs.microsoft.com/pt-br/dotnet/csharp/tour-of-csharp/.

Microsoft . 2015. *Microsft.* [Online] 05 de 10 de 2015. [Citado em: 08 de 05 de 2021.] https://docs.microsoft.com/pt-br/visualstudio/test/generate-unit-tests-for-your-code-with-intellitest?view=vs-2019.

Pontes, Sérgio. 2018. *Jus Brasil.* [Online] 2018. https://sergiopontes.jusbrasil.com.br/artigos/614642198/o-que-fala-a-lei-geral-de-protecao-de-dados-pessoais.

Presidência da República. 2018. *Planalto.* [Online] 14 de Agosto de 2018. http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm.

RODRIGUES, JOEL. 2014. DEVMEDIA. *DEVMEDIA.* [Online] 01 de 01 de 2014. [Citado em: 14 de 11 de 2021.] https://www.devmedia.com.br/modelo-entidade-relacionamento-mer-e-diagrama-entidade-relacionamento-der/14332. 0.

Sanches, Carlos. 2020. *Duk.* [Online] 24 de Setembro de 2020. https://www.duk.com.br/sem-categoria/conheca-3-leis-que-sao-voltadas-para-o-area-de-tecnologia-da-informacao/.

Segura, Rafael de A. 2010. *Universidade Cruzeiro do Sul.* [Online] 2010. [Citado em: 02 de 05 de 2021.] https://arquivos.cruzeirodosulvirtual.com.br/materiais/disc_2010/2sem_2010/mat_grad_modeldados/unidade4/texto_teorico_IV.pdf.

Universidade Católica de Brasília. 2018 . *Universidade Católica de Brasília.* [Online] 2018 . [Citado em: 02 de 05 de 2021.] https://conteudo.catolica.edu.br/conteudos/nbt_cursos/engenharia_requisitos/tema_08/index.

html?access_token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwczpcL1wvY29udGV1ZG8uY2F0b2xpY2EuZWR1LmJyIiwiYXVkIjoiaHR0cHM6XC9cL2NvbnRldWRvLmNhdG9saWNhLmVkdS.