UNIVERSIDADE DO VALE DO ITAJAÍ

TECNÓLOGO EM ANÁLISE E DESENVOLVIMENTO DE SISTEMA

HANDS ON WORK - V

CAMPUS – SÃO JOSÉ (SC)

PEDRO HENRIQUE MACHADO PORATH

APLICATIVO MÓVEL PARA GESTÃO DA MANUTENÇÃO VIÁRIA DO MUNICÍPIO DE SÃO JOSÉ - SC

SÃO JOSÉ – SC

2021

APLICATIVO MÓVEL PARA GESTÃO DA MANUTENÇÃO VIÁRIA DO MUNICÍPIO DE SÃO JOSÉ - SC

Relatório desenvolvido na disciplina de Hands on Work V do curso Técnólogo em Análise e Desenvolvimento de Sistema da Universidade do Vale do Itajaí (Univali).
Professor: Prof. Me. Lucas Debatin.

São José – SC
2021

SUMÁRIO

1 INTRODUÇÃO

Durante o primeiro ciclo de disciplinas da terceira fase do curso de Análise e Desenvolvimento de Sistemas da Universidade do Vale do Itajaí (Univali), através da integração dos conteúdos das disciplinas de Engenharia de Software e Database Design for Apps foi possível na disciplina de “Hands on Work V” seguir a proposta do plano de ensino que era o desenvolvimento de um projeto de aplicativo móvel.

O município de São José, assim como de diversos outros brasileiros possuem problemas com a gestão de suas vias públicas, como por exemplo: falta de sinalização vertical e horizontal, deformações no pavimento, problemas de drenagem, iluminação pública etc. Em muitos desses casos os problemas persistem devido ao desconhecimento do fato pelos órgãos responsáveis ou ainda por problema de gestão devido a falta de um mapeamento digital e de fácil atualização dos problemas do município.

Nesse sentido, a população josefense pode cobrar os órgãos responsáveis e atuar como fiscais do município. Dessa forma, para esta atividade foi planejado o desenvolvimento de um aplicativo móvel que faça registros de problemas no sistema viário do município de São José (SC) citados anteriormente.

Neste momento a aplicação é apenas um protótipo, logo será desenvolvido para caráter experimental apenas a parte de coletada de dados, assim o desenvolvimento da solução de gestão dos dados para um segundo momento. Caso a ideia venha a ser aderida pelo município, a solução teria como usuários a sua própria população a um nível de privilégio de coleta de dados, além dos gestores públicos que além de poderem coletar dados também são responsáveis pela gestão dos dados coletados.

Visando solucionar o problema de maneira simples e intuitiva, a operação do aplicativo será totalmente no idioma português. O usuário deverá em uma primeira seção realizar o preenchimento alfanumérico do problema em questão, em uma segunda etapa fazer um registro fotográfico e por fim coletar uma coordenada geográfica do local facilitando assim a identificação do local a ser ajustado.

2 BANCO DE DADOS

2.1 DIAGRAMA ENTIDADE RELACIONAMENTO (DER)

A utilização de modelos de entidade relacionamento (ER) é muito importante para a correta e eficiente construção de um banco de dados. Dessa forma, Peter Chen na década de 1970 desenvolveu a modelagem ER para design de Banco de Dados. Segundo Ramakrishnan (2011, p.22):

> O modelo de dados entidade-relacionamento (ER) nos permite descrever os dados envolvidos em uma empresa do mundo real em termos de objetos e seus relacionamentos e é amplamente utilizado para desenvolver um projeto inicial de banco de dados. Ele fornece conceitos úteis que nos possibilitam mover de uma descrição informal do que os usuários desejavam de seu banco de dados para uma descrição mais detalhada e precisa que pode ser implementada em um SGBD (RAMAKRISHNAN, 2011, p.22).

Os modelos ER se aportam em diagramas ER ou fluxogramas que ilustram entidades para poder mais facilmente modelar e criar bancos de dados relacionais, em termos de regras lógicas e de negócio (em um modelo lógico de dados) e em termos da tecnologia específica a ser implementada (em um modelo físico de dados).

Para esta atividade, primeiramente foi desenvolvido o diagrama ER pela visão de Chen, onde retângulos representam as entidades, elipses os atributos, losangos os relacionamentos e linhas ligam atributos a entidades e entidades a relacionamentos, como é possível observar na Figura 1.

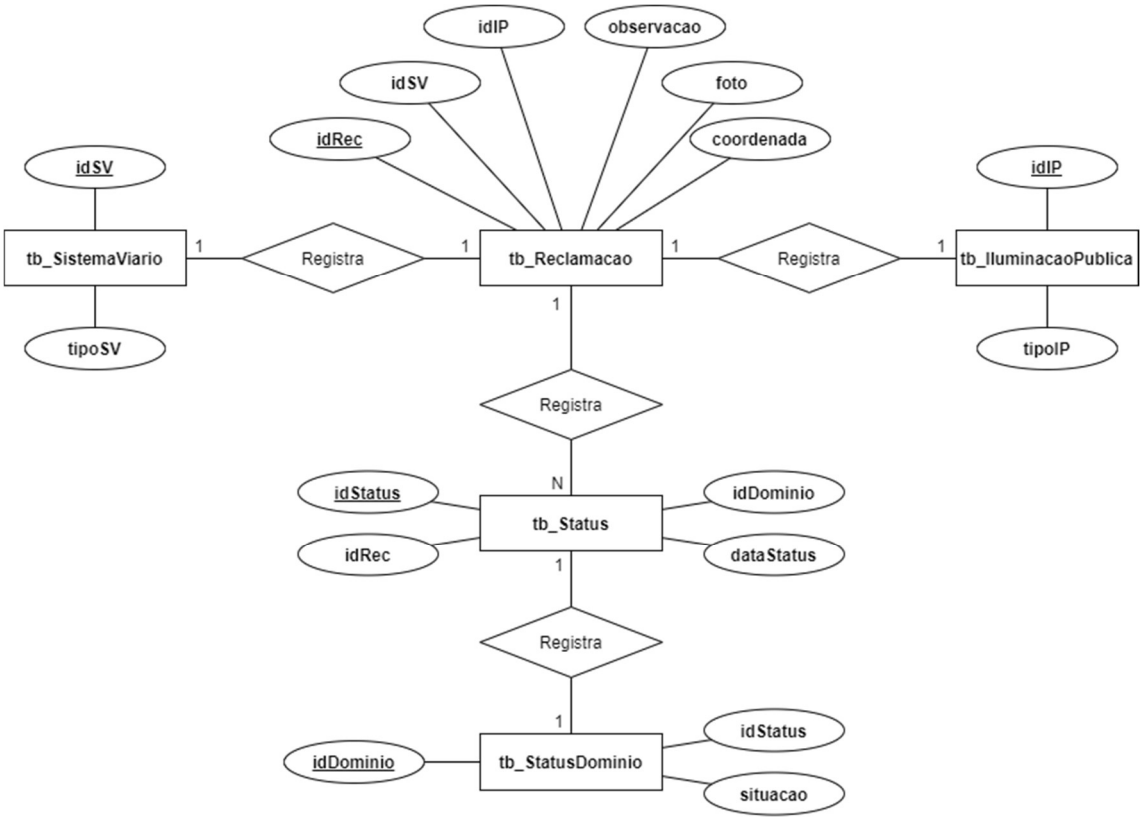

Figura 1: Diagrama ER por Peter Chen. Fonte: O autor.

2.2 MODELO DE ENTIDADE RELACIONAMENTO (MER)

Como é sabido o DER é a representação gráfica do Modelo de ER, dessa forma após a concepção do DER foi possível desenvolver o MER de forma normalizada. Sendo ela representada no Quadro 1.

tb_SistemaViario (idSV, tipoSV)
tb_Reclamacao (idRec, idSV, idIP, observação, foto, coordenada)
tb_iluminacaoPublica (idIP, tipoIP)
tb_Status (idStatus, idRec, idDominio, dataStatus)
tb_StatusDominio (idDominio, idStatus, situacao)

Quadro 1: Modelagem ER. Fonte: O autor.

A modelagem ER também pode ser visualizada a um maior nível de complexidade na Figura 2, onde além das entidades e atributos também é possível observar a cardinalidade do relacionamento.

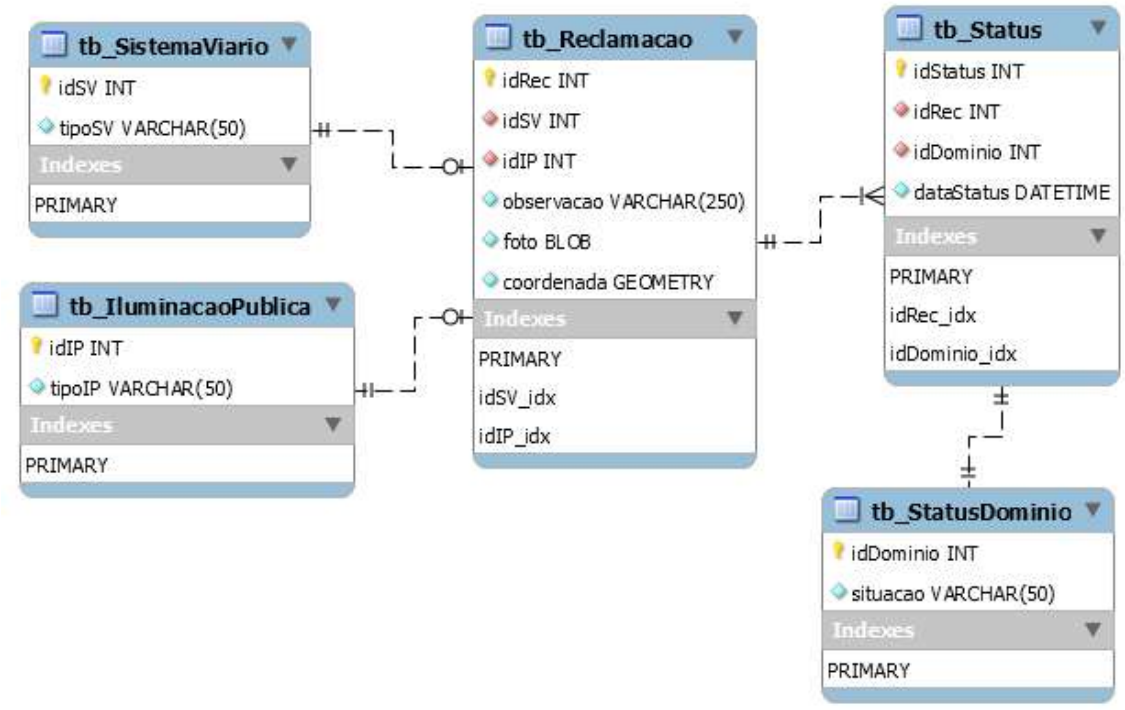

Figura 2: Modelagem ER. Fonte: O autor.

2.3 CÓDIGO SQL (BANCO E TABELAS)

Após o desenvolvimento do Modelo de Entidade Relacionamento pôde-se de fato iniciar o desenvolvimento do script em *Structured Query Language* (SQL) do Banco de Dados e tabelas do projeto a partir da perspectiva da *Data Definition Language* (DDL). O DDL uma linguagem de definição utilizada na composição do SQL e é utilizada na modelagem e no desenvolvimento de bancos de dados para a definição de estruturas de dados, como colunas, tabelas, linhas e índices.

Dessa forma então pode-se estruturar o Banco de Dados em um sistema relacional, com o total de cinco tabelas contendo chaves de relacionamentos entre elas, como é ilustrado no Quadro 2.

```
CREATE SCHEMA IF NOT EXISTS `bd_Reclamacao` DEFAULT CHARACTER SET
utf8 ;
USE `bd_Reclamacao` ;

CREATE TABLE IF NOT EXISTS `bd_Reclamacao`.`tb_SistemaViario` (
  `idSV` INT NOT NULL AUTO_INCREMENT,
  `tipoSV` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`idSV`))
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `bd_Reclamacao`.`tb_IluminacaoPublica` (
  `idIP` INT NOT NULL AUTO_INCREMENT,
  `tipoIP` VARCHAR(50) NOT NULL,
```

```
  PRIMARY KEY (`idIP`))
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `bd_Reclamacao`.`tb_Reclamacao` (
  `idRec` INT NOT NULL AUTO_INCREMENT,
  `idSV` INT NOT NULL,
  `idIP` INT NOT NULL,
  `observacao` VARCHAR(250) NOT NULL,
  `foto` BLOB NOT NULL,
  `coordenada` GEOMETRY NOT NULL,
  PRIMARY KEY (`idRec`),
  INDEX `idSV_idx` (`idSV` ASC) VISIBLE,
  INDEX `idIP_idx` (`idIP` ASC) VISIBLE,
  CONSTRAINT `fk_Reclamacao_SistemaViario`
    FOREIGN KEY (`idSV`)
    REFERENCES `bd_Reclamacao`.`tb_SistemaViario` (`idSV`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Reclamacao_IluminacaoPublica`
    FOREIGN KEY (`idIP`)
    REFERENCES `bd_Reclamacao`.`tb_IluminacaoPublica` (`idIP`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `bd_Reclamacao`.`tb_StatusDominio` (
  `idDominio` INT NOT NULL AUTO_INCREMENT,
  `situacao` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`idDominio`))
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `bd_Reclamacao`.`tb_Status` (
  `idStatus` INT NOT NULL AUTO_INCREMENT,
  `idRec` INT NOT NULL,
  `idDominio` INT NOT NULL,
  `dataStatus` DATETIME NOT NULL,
  PRIMARY KEY (`idStatus`),
  INDEX `idRec_idx` (`idRec` ASC) VISIBLE,
  INDEX `idDominio_idx` (`idDominio` ASC) VISIBLE,
  CONSTRAINT `fk_Status_Reclamacao`
    FOREIGN KEY (`idRec`)
    REFERENCES `bd_Reclamacao`.`tb_Reclamacao` (`idRec`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Status_StatusDominio`
    FOREIGN KEY (`idDominio`)
    REFERENCES `bd_Reclamacao`.`tb_StatusDominio` (`idDominio`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Quadro 2: Código SQL (DDL) com criação do BD e tabelas do projeto.

2.4 EXEMPLOS DE MANIPULAÇÃO E CONSULTA DE DADOS (SQL)

Neste tópico foram exemplificados o uso de instruções do *Data Manipulation Language* (DML) do SQL que são: INSERT (Quadro 3), UPDATE (Quadro 4), DELETE (Quadro 5). Além da instrução SELECT (Quadro 6) do *Data Query Language* (DQL).

```sql
INSERT INTO tb_SistemaViario (idSV, tipoSV) VALUES (1, 'Buraco no
pavimento');
INSERT INTO tb_SistemaViario (idSV, tipoSV) VALUES (1, 'Buraco no
pavimento');
INSERT INTO tb_SistemaViario (idSV, tipoSV) VALUES (2, 'Via não
pavimentada');
INSERT INTO tb_SistemaViario (idSV, tipoSV) VALUES (3, 'Problema com
a drenagem na via');
INSERT INTO tb_SistemaViario (idSV, tipoSV) VALUES (4, 'Semáforo
desligado');
INSERT INTO tb_SistemaViario (idSV, tipoSV) VALUES (5, 'Falta de
sinalização');

INSERT INTO tb_IluminacaoPublica (idIP, tipoIP) VALUES (1,
'Iluminação queimada');
INSERT INTO tb_IluminacaoPublica (idIP, tipoIP) VALUES (2, 'Falta de
Iluminação');

INSERT INTO tb_StatusDominio (idDominio, situacao) VALUES (1,
'Pendente');
INSERT INTO tb_StatusDominio (idDominio, situacao) VALUES (2, 'Em
análise');
INSERT INTO tb_StatusDominio (idDominio, situacao) VALUES (3,
'Concluída');
```

Quadro 3: Código SQL (DML) para inserir valores nas tabelas do BD.

```sql
UPDATE tb_SistemaViario SET tipoSV = 'Buraco no pavimento' WHERE
idSV = 1;
UPDATE tb_IluminacaoPublica SET tipoIP = 'Falta de Iluminação' WHERE
idIP = 2;
UPDATE tb_StatusDominio SET situacao = 'Concluída' WHERE idDominio =
3;
UPDATE tb_Reclamacao SET observacao = 'Falta de drenagem provoca
inundação' WHERE idSV = 3;
UPDATE tb_SistemaViario SET tipoSV = 'Falta de sinalização' WHERE
idSV = 5;
```

Quadro 4: Código SQL (DML) para atualizar valores nas tabelas do BD.

```sql
DELETE FROM tb_SistemaViario WHERE idSV = 1;
DELETE FROM tb_SistemaViario WHERE tipodSV LIKE '%pav%';
DELETE FROM tb_IluminacaoPublica WHERE idIP > 0;
DELETE FROM tb_Reclamacao WHERE idIP IS NULL;
DELETE FROM tb_StatusDominio WHERE situacao IN ('Pendente', 'Em
análise');
```

Quadro 5: Código SQL (DML) para excluir valores nas tabelas do BD.

```sql
SELECT COUNT (idSV) FROM tb_SistemaViario WHERE tipoSV = 'Via não
pavimentada';
SELECT idDominio FROM tb_StatusDominio WHERE situacao = 'Pendente';
SELECT * FROM tb_IluminacaoPublica WHERE idIP < 2;
SELECT idRec, observacao FROM tb_Reclamacao WHERE idIP IS NULL;
SELECT idRec, observacao FROM tb_Reclamacao WHERE idSV IS NOT NULL
AND idIP > 3;
SELECT * FROM tb_StatusDominio ORDER BY situacao ASC;
SELECT tipoIP FROM tb_IluminacaoPublica LIMIT 0,3;
SELECT sv.tipoSV rec.idSV FROM tb_SistemaViario sv INNER JOIN
tb_Reclamacao rec ON rec.idSV = sv.idSV);
SELECT ip.tipoIP rec.idIP FROM tb_IluminacaoPublica ip INNER JOIN
tb_Reclamacao rec ON rec.idIP = ip.idIP);
SELECT sd.situacao st.idDominio FROM tb_StatusDominio sd INNER JOIN
tb_Status st ON sd.idDominio = sv.idDominio);
SELECT st.dataStatus rec.idRec FROM tb_Status st INNER JOIN
tb_Reclamacao rec ON rec.idRec = st.idRec);
SELECT sv.idSV rec.idSV FROM tb_SistemaViario sv INNER JOIN
tb_Reclamacao rec ON rec.idSV = sv.idSV);
```

Quadro 6: Código SQL (DQL) para selecionar valores nas tabelas do BD.

2.5 PROCEDIMENTO ARMAZENADO E GATILHO NO BANDO DE DADOS

Neste tópico serão exemplificados o uso de Funções (Functions), Procedimento (Procedures) e Gatilhos (Triggers).

```sql
-- FUNCTION

CREATE OR REPLACE FUNCTION insere_tipoSV (tipoSV_p VARCHAR)
RETURNS SETOF tb_SistemaViario AS $$
    INSERT INTO tb_SistemaViario (tipoSV) VALUES (tipoSV_p);
    SELECT * FROM tb_SistemaViario;
$$ LANGUAGE SQL;
```

Quadro 7: Código SQL representando uma Função.

```sql
-- PROCEDURE

CREATE OR REPLACE PROCEDURE atualiza_Status (INTEGER, INTEGER)
LANGUAGE SQL AS $$
    UPDATE tb_Status SET idRec = $2 WHERE idStatus = $1;
$$;

CALL atualiza_Status(1, 3);

SELECT * FROM atualiza_Status ORDER BY idStatus;
```

Quadro 8: Código SQL representando um Procedimento.

```
-- TRIGGER

CREATE OR REPLACE FUNCTION desativa_IluminacaoPublica () RETURNS
TRIGGER AS $$
    DECLARE
        tipoIP VARCHAR;
    BEGIN
        IF NEW.ativo = false THEN
            SELECT COUNT(id) INTO contador_IluminacaoPublica FROM
tb_IluminacaoPublica WHERE idIP = NEW.id;
            IF contador_IluminacaoPublica > 0 THEN
                RAISE NOTICE;
                RETURN NULL;
            END IF;
        END IF;
        RETURN NEW;
    END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER desativa_IluminacaoPublica_trigger BEFORE UPDATE ON
tb_IluminacaoPublica
FOR EACH ROW EXECUTE PROCEDURE desativa_IluminacaoPublica();

SELECT * FROM tb_IluminacaoPublica;

UPDATE tb_IluminacaoPublica SET ativo = true WHERE idIP = 2;

SELECT * FROM tb_IluminacaoPublica;

UPDATE tb_IluminacaoPublica SET ativo = false WHERE idIP = 2;

UPDATE tb_IluminacaoPublica SET ativo = false WHERE idIP = 1;
```

Quadro 9: Código SQL representando um Gatilho.

3 ENGENHARIA DE SOFTWARE

3.1 DEFINIR OS RECURSOS NECESSÁRIOS DO PROJETO

- Pessoas:
    - 1 profissional que irá atuar nas funções de analista, programador e tester;
- Hardware:
    - 1 servidor de banco de dados para desenvolvimento;
    - 1 notebook com processador Intel(R) Core (TM) i7-10510U CPU @ 1.80GHz 2.30 GHz, memória RAM de 16,0 GB; e
    - 1 Monitor de 24 polegadas.
- Software:
    - Linguagens: Framework IONIC 6 (angular), 1 licença estudante da IDE WebStorm (https://jetbrains.com), 1 licença do Diagrams.net (https://www.diagrams.net/) e 1 licença PRO do Cypress (https://www.cypress.io/)

3.2 ELABORAR A ANÁLISE DE RISCOS DO PROJETO.

ID	Descrição	Probabilidade	Impacto	Gatilho	Plano de Contingência
1	Modificações de requisitos provenientes da prefeitura	Baixa	Alta	Modificações no fluxo de atividade da prefeitura	Gerar um contrato adicional para desenvolver as alterações.
2	Pressão para reduzir prazos de entrega	Baixo	Alto	Os stakeholders solicitarem o software para quanto antes.	Alocar mais programadores na equipe e gerar uma cobrança extra para a universidade.
3	Problema de gerenciamento do projeto	Médio	Médio	Dificuldade para compreender todas as funcionalidades do projeto	Agendar reuniões com maior frequência com os Stakeholders, para evitar possíveis erros que se propaguem no decorrer do projeto.
4	Atualização de tecnologia ao longo do desenvolvimento do projeto	Baixo	Baixo	Novas pesquisas e ideias surgirem ao longo do desenvolvimento visando facilitar o desenvolvimento do projeto.	Implementar novas ideias apenas se a nova estratégia se mostrar eficiente e necessária.

3.3 ELABORAR O CICLO DE VIDA DO DESENVOLVIMENTO DO APLICATIVO.

De acordo com IBM (2006), do ponto de vista do gerenciamento, o ciclo de vida de software do Rational Unified Process (RUP) é dividido em quatro fases sequenciais, cada uma concluída por um marco principal, ou seja, cada fase é basicamente um intervalo de tempo entre dois marcos principais. À cada final de fase, uma avaliação é executada para determinar se os objetivos da fase foram alcançados. Uma avaliação satisfatória permite que o projeto passe para a próxima fase.

Dessa forma, para o planejamento e acompanhamento das atividades se fez necessário a elaboração do Ciclo de Vida do desenvolvimento do Aplicativo representado por um fluxograma, como é possível observar na Figura 3.

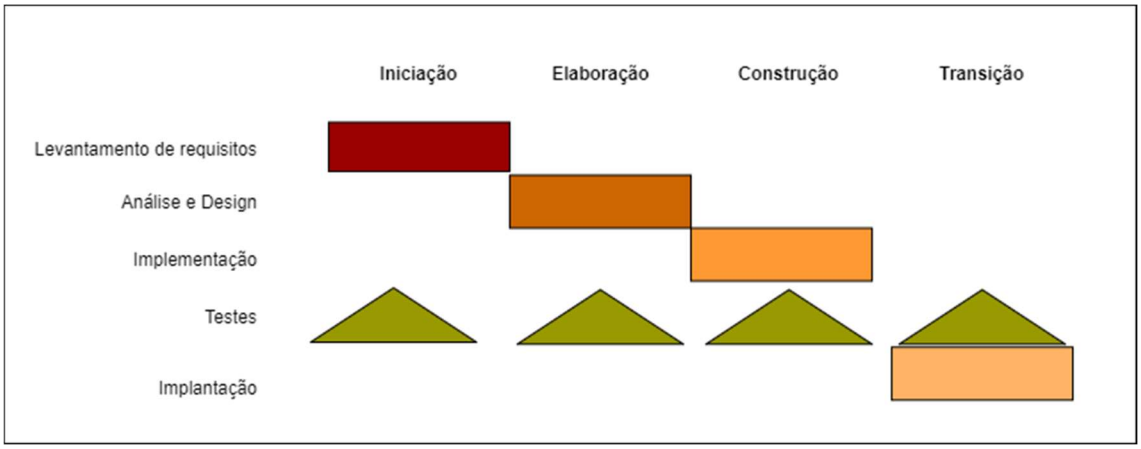

Figura 3: Ciclo de Vida do desenvolvimento do Aplicativo (RUP).

3.4 DIAGRAMA DE CASO DE USO E CÁLCULO DO USE CASE POINTS (UCP).

O Diagrama de Caso de Uso, tem origem na Linguagem de Modelagem Unificada (UML) e resume os detalhes dos usuários do seu sistema (atores) e as interações deles com o sistema. A partir da Figura 4 é possível verificar o Diagrama de Caso de Uso do sistema a ser desenvolvido.

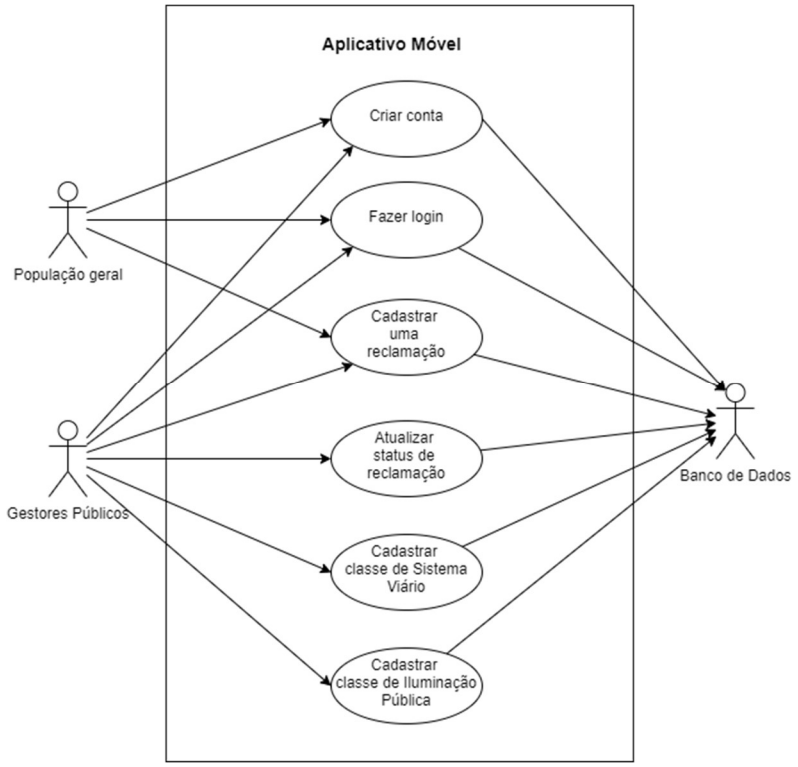

Figura 4: Representação de Diagrama de Caso de Uso.

Após a definição do Diagrama de Uso de Caso foi possível calcular os Pontos de Caso por Uso, do inglês *Use Case Points* (UCP). Esse, é uma técnica de estimativa de desenvolvimento de software. A seguir são então apresentadas as funções:

- Fazer login: validar senha (simples);
- Criar conta: Editar, excluir (simples);
- Cadastrar classe de Sistema Viário: criar, alterar, excluir e listar (médio);
- Cadastrar classe de Iluminação Pública: criar, alterar, excluir e listar (médio);
- Cadastrar uma reclamação: criar (simples); e
- Atualizar status de reclamação: alterar, excluir, listar e enviar e-mail (médio).

Tabela 1: Cálculo de User Case Points

Complexidade	**Quantidade**	**Peso**	**Total**
Simples	3	5	15
Médio	3	10	30
Complexo	0	15	0
		Total UCP	45

3.5 METODOLOGIA ADOTADO NO PROJETO

Para gestão e organização deste projeto foi definida a utilização de metodologia Ágil. Ao analisar as possibilidades, entendeu-se que o Scrum teria uma fácil adaptabilidade com a proposta, devido o controle de processos empíricos e a entrega iterativa. Dentro outros motivos, pôde-se destacar:

- Eliminação do desperdício;
- Transparência;
- Detecção rápida de erros e problemas;
- Controle da evolução do projeto; e
- Motivação do time.

3.6 ELABORAR O CRONOGRAMA DO PROJETO.

Atividade	Data de conclusão
Desenvolver Introdução	15/003/2021
Desenvolver DER e MER	17/03/2021
Códigos SQL do projeto	25/03/2021
Desenvolver Simulações SQL com DML e DQL	31/03/2021
Definir os recursos necessários do projeto:	03/04/2021
Elaborar a análise de riscos do projeto	05/04/2021
Elaborar o ciclo de vida do desenvolvimento do aplicativo	05/04/2021
Elaborar o diagrama de caso de uso de todo o aplicativo, e, com base nisso, calcular o Use Case Points (UCP).	10/04/2021
Definir a melhor metodologia ágil para o projeto. Justifique a sua resposta	15/04/2021
Finalizar relatório do projeto	22/04/2021

Quadro 10: Definição do cronograma de atividades.

4 CONCLUSÃO

Com o aprendizado de conceitos da linguagem SQL e Engenharia de Software, está sendo possível desenvolver a arquitetura de um sistema Mobile e aprender um pouco sobre o gerenciamento de Banco de Dados. Como o sistema ainda está incompleto, para essa primeira etapa os arquivos desenvolvidos para o sistema em questão ainda não se

encontram no repositório GitHub com acesso público. O que será feito na próxima entrega, uma vez que além de cumprir a função social de compartilhamento de conhecimento, fica disponível para que outros usuários possam indicar melhorias.

5 REFERÊNCIAS

DEVMEDIA (Brasil). **Introdução a Requisitos de Software**. [2020?]. Disponível em: https://www.devmedia.com.br/introducao-a-requisitos-de-software/29580. Acesso em: 04 abr. 2021.

IBM. **Ciclo de Vida do RUP**. 2006. Disponível em: https://www.cin.ufpe.br/~gta/rup-vc/core.base_rup/customcategories/rup_lifecycle_100BF298.html. Acesso em: 05 abr. 2021.