PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

EquipeOn

Gerenciador de equipes externas

Caíque Molina Soares

PROJETO FINAL DE GRADUAÇÃO

CENTRO TÉCNICO CIENTÍFICO- CTC

DEPARTAMENTO DE INFORMÁTICA

Curso de Graduação de Ciência da Computação

Rio de Janeiro, Novembro de 2021

Relatório de Projeto Final, apresentado ao programa Ciência da Computação da PUC-Rio como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Ivan Mathias Filho

Rio de Janeiro, Novembro de 2021

Agradecimentos

Ao meu orientador Ivan Mathias Filho pela ajuda sempre que necessário e pelos conhecimentos passados.

Aos meus pais e irmão, por todo o amor, incentivo e por fazerem tudo isso possível.

Resumo

Este trabalho aborda um software para resolver problemas cotidianos de equipes de serviço em campo. Tendo como seu principal objetivo melhorar o controle dos funcionários e da qualidade dos serviços prestados, por meio de um sistema que foi desenvolvido para facilitar a gestão das empresas do setor. A aplicação permite a criação de ordens de serviço na web que são designadas para funcionários atenderem tarefas remotamente em seus smartphones e enviar sua resolução de volta para a empresa em tempo real, gerando uma maior rapidez, controle e diminuindo os custos de atendimentos.

Palavras Chaves

Ordens de serviço. Controle de tarefas. Criação de formulários. Atendimento em tempo real.

Abstract

This project is about a software to solve the daily problems of field service teams. Aiming to improve the control of employees and the quality of services provided through an auxiliary system developed to facilitate the management of companies in the sector. The application allows the creation of work orders on the web that are designated for company employees to remotely attend tasks on their smartphones and send their resolution back to the company in real time, generating greater speed, control and reducing service costs.

Keywords

Service orders. Task control. Creating forms. Real-time service.

Sumário

Tabela de figuras

1. Introdução

Em 2014, ao prestar assessoria para uma empresa especializada em aluguel e venda de hardwares (impressoras e computadores), foi relatado pelo diretor da empresa uma certa deficiência em controlar sua equipe de técnicos, que prestavam manutenção nos equipamentos da empresa em seus diversos clientes pelo Rio de Janeiro. A dificuldade era em organizar e gerenciar as ordens de serviço, além de controlar o rendimento dos funcionários e a qualidade dos serviços realizados. A partir desse relato, foi feita uma breve pesquisa nas empresas prestadoras de serviço, chegando a conclusão que era um problema comum de várias empresas no setor, surgindo então a oportunidade de desenvolver um software que ajudasse a resolver esse problema.

Mesmo que a atualidade seja movida pelo digital, e os smartphones estejam cada vez mais presentes em nosso dia-a-dia, ainda sim ainda é muito comum ver empresas usando processos em papel para resolver problemas. Os mesmos são, posteriormente, importados para algum sistema, o que dificulta a gestão da empresa e a agilidade nos processos, além de aumentar custos.

Após a análise dessa situação, foi elaborado um projeto pensando na solução desse problema, um sistema Web-App, onde a parte da Web seria a central da empresa, armazenando todos os dados relevantes para gestão dos serviços e um aplicativo para atender as tarefas externas, criadas pelo painel web.

Com a implementação desse sistema seria possível ter um maior controle sobre todas as tarefas realizadas pela empresa, enriquecendo a mesma com dados relevantes sobre os serviços executados, aumentando sua capacidade de atender problemas e diminuindo seus custos operacionais.

1.1 Foco do projeto

O projeto final irá apresentar uma solução criada para resolver problemas de gestão de serviços e equipes, uma deficiência comum encontrada entre os servidores de hardware de algumas empresas, sendo a aplicação composta por uma interface web administrativa e um aplicativo mobile para a execução de serviços.

1.2 Detalhamento dos tópicos

Este trabalho possui a seguinte estrutura, cujos os resumos dos capítulos são exibidos a seguir:

Situação atual (Capítulo 2) : Será abordado o contexto da origem da ideia do aplicativo, a motivação e uma breve análise de soluções existentes.

Objetivos e propostas do trabalho (Capítulo 3): Trata-se da explicação de como o projeto irá solucionar o problema encontrado e os principais pontos a serem alcançados.

Atividades realizadas (Capítulo 4): Aborda todos os estudos realizados sobre as tecnologias que foram utilizadas, uma breve comparação com as tecnologias antigas e vantagens de fazer a mudança.

Projeto e especificação do sistema (Capítulo 5): Trata-se de um capítulo mais técnico onde são especificados os requisitos do projeto, sua arquitetura e a metodologia de desenvolvimento utilizada.

Implementação e avaliação (Capítulo 6): Este capítulo mostra como foi planejada a implementação e execução do sistema, além dos testes feitos e avaliações gerais.

Considerações finais (Capítulo 7): Trata-se do capítulo final do trabalho com os aprendizados e comentários do aplicativo e o futuro do projeto.

Apêndices: Contém informações complementares deste sistema.

2. Situação Atual

2.1 Contexto do projeto

Em uma conversa com o dono da empresa, ele se propôs a colaborar na criação de um sistema complementar para a companhia, possibilitando a utilização da mesma como laboratório para desenvolvimento e testes da aplicação.

A empresa em questão atua como prestadora de serviços de impressoras e computadores, fazendo a manutenção dos hardwares quando necessário. Também fornece serviços de aluguel e venda dos equipamentos.

Ao se estudar mais a fundo os processos e estrutura da empresa em que foi feita a consultoria, nota-se que ela não possui um sistema para uso em campo para seus funcionários, isto é, um aplicativo que possibilitasse a comunicação com o sistema interno da empresa. Ou seja, no atendimento de uma ordem de serviço, desde o momento de sua criação (cadastro no sistema

interno) até o momento de sua execução (técnico ir até o cliente e resolver o problema), era feito manualmente por atendentes, o que eventualmente gerava erros e lentidão nos serviços.

No entanto, a empresa não tinha uma visualização fácil e clara das informações. Por exemplo, o sistema interno que a empresa utiliza coletava a hora em que uma ordem de serviço era criada e encerrada, mas não mostrava o tempo de execução do serviço em um relatório final, esses pontos estudados foram essenciais para o planejamento do software.

2.2 Pesquisa de soluções existentes

Também foram pesquisados sistemas similares que poderiam solucionar o problema do cliente. Ao ser questionado o motivo de nunca ter utilizado uma solução existente do mercado, o dono da empresa alegou não ter o conhecimento necessário, nem o tempo hábil para implementar a solução, mas que deixaria sua empresa a disposição para desenvolvimento e implementação do mesmo.

Ao analisar os possíveis concorrentes para o sistema, foram encontradas duas boas soluções, chamadas de *GO.ON[1]* e *Field Control[2]* que tinham basicamente a mesma estrutura: uma interface web gerencial com um aplicativo mobile para uso externo, ambas as plataformas utilizavam um design simples e objetivo. Entretanto, o grande problema delas, que foi visto nos comentários de seus clientes, é que são soluções pouco flexíveis para diferentes demandas, ou seja, empresas que fazem manutenção de elevadores precisam de opções diferentes das empresas que fazem manutenções em impressoras. Para resolver esse problema era preciso contactar os desenvolvedores para uma personalização de acordo com os pedidos do cliente. Observando esse problema, foi planejado um sistema que pudesse resolver as demandas de qualquer prestador de serviço, de forma simples.

Após toda a pesquisa de mercado e estudos sobre tais empresas, já era possível ter um esboço do projeto que viria a ser desenvolvido, uma solução que funcionasse como um sistema para servidores que tivessem integração com os sistemas internos já utilizados por eles, dessa forma, não causaria um impacto tão grande nos processos que utilizam e a barreira de entrada seria menor.

3. Objetivos e Proposta do trabalho

3.1 Objetivo do sistema

O objetivo principal do projeto era disponibilizar um sistema que atendesse às demandas externas de prestadores de serviço, fazendo uma comunicação em tempo real com o servidor interno das empresas, a fim de melhorar a gestão de serviços e funcionários das empresas, digitalizando processos, aumentando a produtividade e facilitando o controle geral.

3.2 Usuários e situação alvo

A principal funcionalidade do projeto a ser alcançada é a criação de ordens de serviço digitais na interface web e o encaminhamento das mesmas para o aplicativo mobile. Nesta aplicação, nosso público alvo são empresas que prestam qualquer tipo de serviço, e os usuários que utilizarão o sistema de fato são funcionários das empresas. Pelo o custo-benefício da aplicação, será desenvolvido o aplicativo apenas no formato para Android, por ser o sistema operacional mais utilizado no mundo e por seus smartphones terem um custo mais acessível.

3.3 O que será apresentado

Este projeto tem como proposta de trabalho a pesquisa de novas tecnologias e linguagens, a fim de reestruturar o sistema, buscando uma versão eficiente, estável e escalável da aplicação. Para demonstrar isso, serão apresentados os seguintes itens a seguir:

1. Reestruturação da arquitetura do software, utilizando padrões de desenvolvimentos atuais, isso inclui:

- Banco de dados
- API de integração

2. Reconstrução da interface web e do aplicativo mobile, focando nas principais funções para sua utilização, isso inclui:

- **Interface web:**
    - Criação/visualização de ordens de serviço
    - Painel analítico dos serviços
    - Criação/visualização de clientes

- Criação/visualização de Usuários (Técnicos, Atendentes e Administradores)
- Personalização/visualização de formulário de atendimento

- **Mobile Android:**
  - Atendimento/visualização de ordens de serviço
  - Perfil de usuário

3. Informações que demonstrem a eficiência e melhoria nas tecnologias escolhidas, casos de teste com cobertura dos principais problemas e layout moderno e intuitivo focado em uma melhor experiência do usuário.

A Figura 1 ilustra de forma simplificada cada uma das respectivas etapas do trabalho que será apresentado e que foram descritas acima.

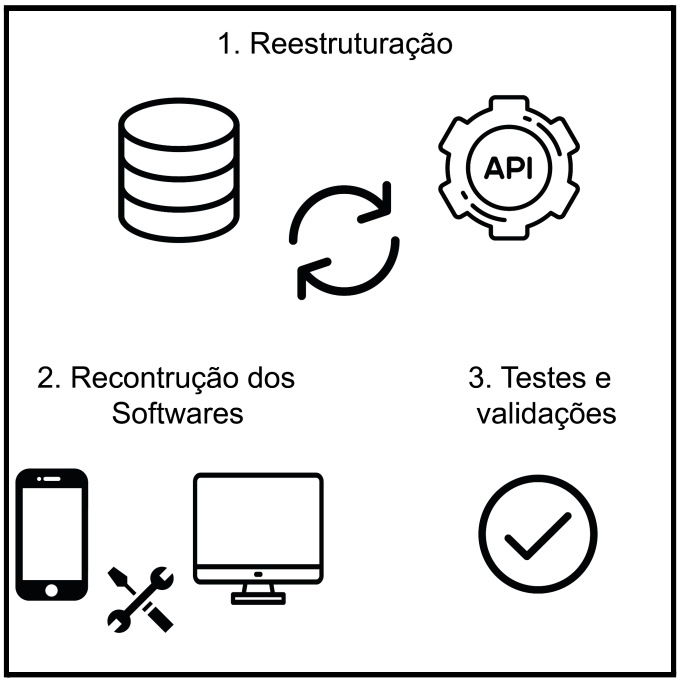

Figura 1. Representação visual das etapas do projeto

4. Atividades realizadas

4.1 Estudos preliminares

Para o desenvolvimento do projeto novas tecnologias foram estudadas para substituir as usadas anteriormente, buscando maior eficiência da aplicação,

melhora de estrutura, manutenção, simplicidade de código e escalabilidade. Após pesquisas para escolha das melhores opções no mercado, foram selecionadas as seguintes tecnologias em relação às tecnologias utilizadas anteriormente no sistema, que são mostradas na **Tabela 1**:

Primeira versão	**Nova versão**	**Descrição/Uso**
Laravel (PHP)	Django[3] (Python)[4]	Linguagem para API. Framework de fácil entendimento e alto nível.
JSON	JSON[5]	Formato de troca de dados simples e rápido, usado na API.
Java	Kotlin[6]	Linguagem de programação consistente e intuitiva, desenvolvida pela JetBrains e adotada pela Google ao Android.
Javascript/HTML/CSS	AngularJS/HTML/CSS [7]	Linguagem para desenvolvimento ágil para WEB.
Github	Github[8]	Plataforma de hospedagem de código fonte com controle de versão usando GIT.
Postman	Postman[9]	Ambiente de desenvolvimento para API, para facilitar e agilizar o trabalho.
Android Studio	Android Studio[10]	Ambiente de desenvolvimento

		Android.
PhpStorm	Visual Studio[11]	Ambiente de desenvolvimento versátil.
PostgresSQL	PostgreSQL[12]	Sistema gerenciador de banco de dados objeto relacional (SGBD).

Tabela 1. Tecnologias estudadas

4.2 Estudos conceituais e de tecnologia

- **Python utilizando framework Django:**

Para a substituição da linguagem PHP em Laravel, foi escolhido o Python com framework Django, que utiliza da arquitetura MVC (Model-View-Controller), sendo assim, possível desenvolver uma REST API estruturada e eficiente, formando um único Webservice para a aplicação.

O framework Django permite o rápido desenvolvimento e fácil manutenção, sendo o mais utilizado para linguagem Python, além de melhorar a segurança do sistema, ativando a proteção contra diversas vulnerabilidades padrões, prometendo uma melhor eficiência baseado na comparação aos tempos de requisição em relação ao Laravel, como é mostrado na Figura 2.

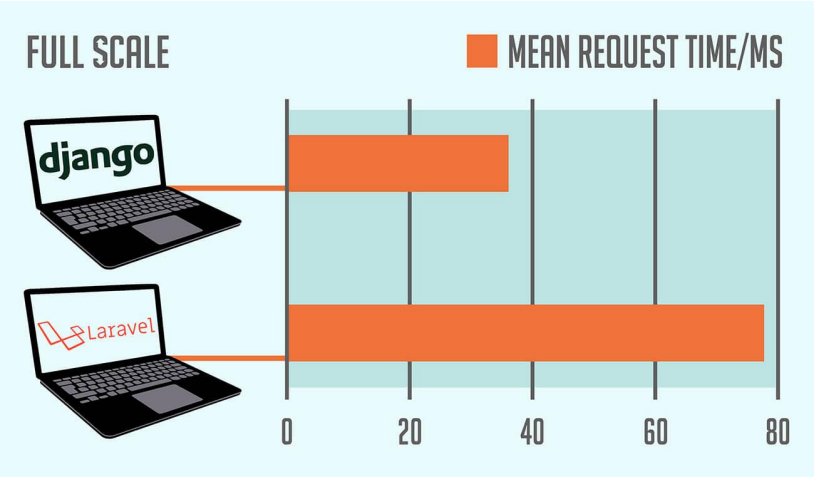

Figura 2. Performance Django Vs Laravel[13]

- **JSON:**

  Para troca de dados entre os módulos do sistema foi escolhido usar JSON, tanto para o aplicativo Mobile quanto para a interface web, com o objetivo de unificar todas as requisições em um único Web Service. Esse formato de troca de dados é ideal para aplicações REST, pois permite facilmente o envio e recebimento de dados.

- **Kotlin:**

  Para substituir a linguagem Java, foi escolhida uma nova linguagem para Android, chamada Kotlin, criada pela Jetbrains com forte contribuição da Google. Essa linguagem utiliza diversos recursos Java porém, com uma sintaxe bem mais simples e oferecendo mais recursos, sendo possível desenvolver mais com menos linhas de código.

  Por ser uma linguagem simples, concisa e pouco verbosa, não seria necessário muito tempo para a adaptação e melhoraria a manutenibilidade do sistema. Além disso, Kotlin é 100% interoperável com Java, o que facilitaria a migração.

- **AngularJS:**

  O AngularJS é um framework open-source de desenvolvimento front-end que possibilita o desenvolvimento de aplicações web, com foco em simplificar a codificação da interface web. A linguagem utiliza a arquitetura MVC[14], melhorando a estrutura e manutenção da interface, sendo um dos frameworks que mais cresceu para o desenvolvimento Web.

- **Postman:**

  O Postman é um ambiente de desenvolvimento que facilita a criação e testes da API, permitindo criar e salvar solicitações HTTP simples e também ler suas respostas. Tal ferramenta já era utilizada anteriormente, porém alguns recursos novos foram estudados para melhor utilização da ferramenta.

- **Ambiente de desenvolvimento e auxiliares**

  Para o desenvolvimento da aplicação Web, foi escolhido o Visual Studio, ambiente que suporta todas as linguagens utilizadas na

aplicação, além de possuir recursos que aceleram o desenvolvimento.

No desenvolvimento do aplicativo foi escolhido o Android Studio, ambiente de desenvolvimento do Android que possui completo suporte para Kotlin e muitos recursos para auxiliar no desenvolvimento.

Ambos os ambientes de desenvolvimentos são integrados com o Github, ferramenta de controle de versão, utilizada para armazenar todo o repositório do projeto.

- **PostgreSQL**

PostgreSQL é um servidor de banco de dados seguro para o armazenamento de informações tendo a capacidade de suportar grandes volumes de informações, sendo o ideal para o projeto.

Na implementação da nova versão do projeto, foi escolhido mantê-lo, porém fazer uma melhor utilização dos recursos oferecidos e retirar alguns de seus usos desnecessários no banco e deixar para o Webservice, tornando os módulos do sistema mais específicos e simples.

4.3 Testes e protótipos para aprendizado e demonstração

A princípio, como o objetivo do projeto é fazer uma reestruturação em um sistema previamente desenvolvido, não foi necessário a criação de protótipos da aplicação para validação da ideia, pelo fato de já ser uma aplicação validada. Porém, para comprovar se os estudos das novas linguagens e tecnologias teriam o efeito esperado, foram realizadas algumas pesquisas em performance entre Laravel e Django. O teste representado na Figura 3, representa a capacidade de requests processadas por segundo em cada framework, o resultado mostra uma performance 10 vezes maior com Django.

Figura 3. Processamento JSON Django Vs Laravel[15]

Como o pesquisa de performance, comprovou a viabilidade da linguagem para a reestruturação do sistema, foi elaborada uma nova arquitetura do sistema para centralizar todas as requisições em um único WebService via API de integração, diferentemente do projeto anterior, onde apenas o Mobile era integrado com o Webservice e a interface web era integrada diretamente com o gerenciador de banco de dados.

4.4 Método

O projeto foi dividido basicamente em duas etapas: a primeira etapa contava com a reestruturação dos módulos dos sistemas e o estudo das novas tecnologias, e a segunda etapa com o design e implementação do novo sistema.

A primeira etapa, a reestruturação dos módulos, foi basicamente voltada para os estudos sobre as tecnologias, técnicas de desenvolvimento e ferramentas novas.

Na segunda etapa, o desenvolvimento foi um processo bastante iterativo, começando por redesenhar todo o banco de dados do sistema, onde foram encontradas diversas falhas de design na estrutura, buscando uma melhor eficiência das tabelas e depois reestruturando todo o Webservice do sistema, que apresentava diversos bugs e problemas de performance, para criar uma API para a integração tanto do Mobile, quanto da interface web, ou seja, toda a parte de Backend foi feita para posteriormente finalizar com o Front-end.

Essa segunda etapa foi desenvolvida e baseada em um processo iterativo e incremental, onde encontros quinzenais eram realizados com o coordenador para que fossem discutidos os progressos nos projetos.

5. Projeto e especificação do sistema

5.1 Fase de requisitos

A fase de levantamento de requisitos dos sistema foi essencial para validar se o projeto atenderia às demandas do cliente. Nessa etapa, foram ajustados alguns aspectos do sistema para que fosse desenvolvido apenas o necessário para utilização inicial no cliente. As Figuras 4 e 5 mostram os principais usos do sistema. No Apêndice A, foi adicionado a listagem de requisitos completa das principais funções da aplicação.

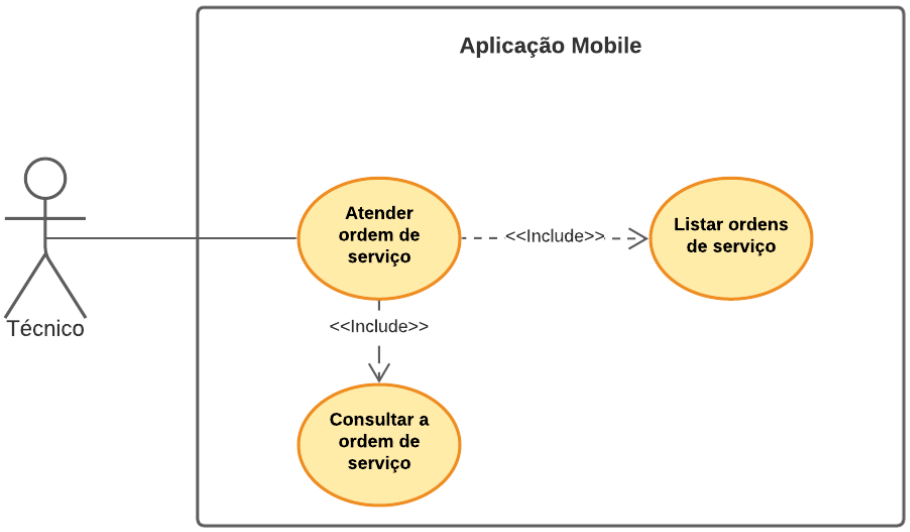

Figura 4. Diagrama de caso de uso da aplicação mobile

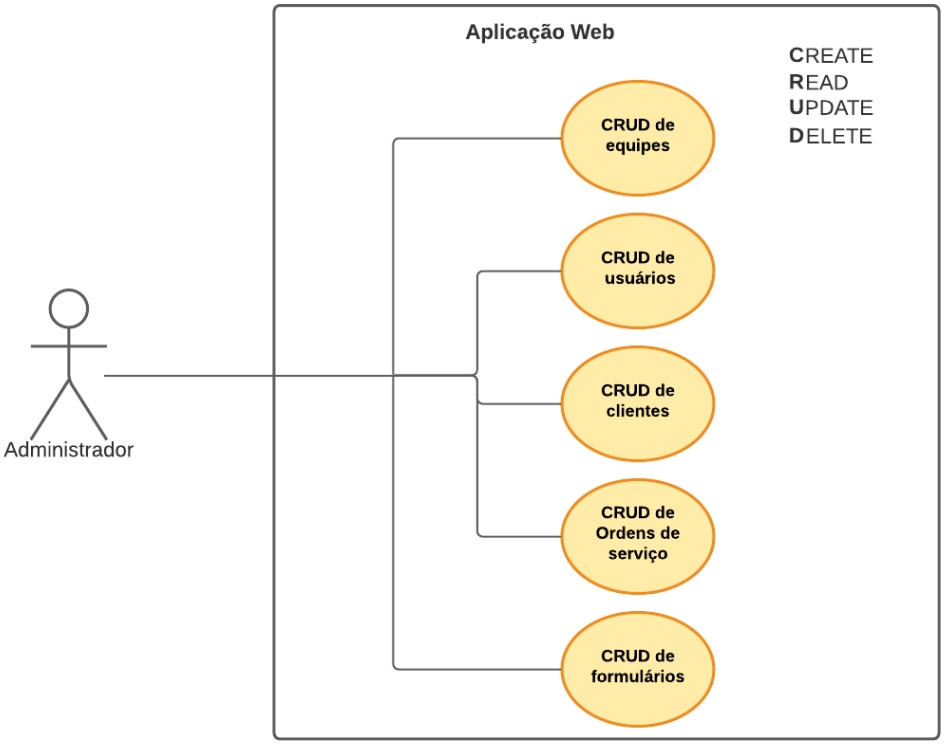

Figura 5. Diagrama de caso de uso da aplicação web

Os diagramas foram divididos em dois, pois a aplicação tem dois atores principais que exercem funções distintas, mas que se complementam. Em ambos os casos os usuários precisam estar autenticados para exercer as

principais funcionalidades da aplicação, a ação de Autenticação não foi representada no diagrama de casos de uso, pois não é uma função útil do sistema. De um lado o administrador coordena toda sua equipe no painel administrativo e do outro lado o técnico é responsável apenas pelos atendimentos. Na Figura 5, foi utilizada a simbologia CRUD**,** que representa um agrupamento das funções de criação, alteração, visualização e eliminação, presentes em todos os casos de uso da aplicação web.

A Figura 6 mostra a interação do técnico e os principais processos do aplicativo.

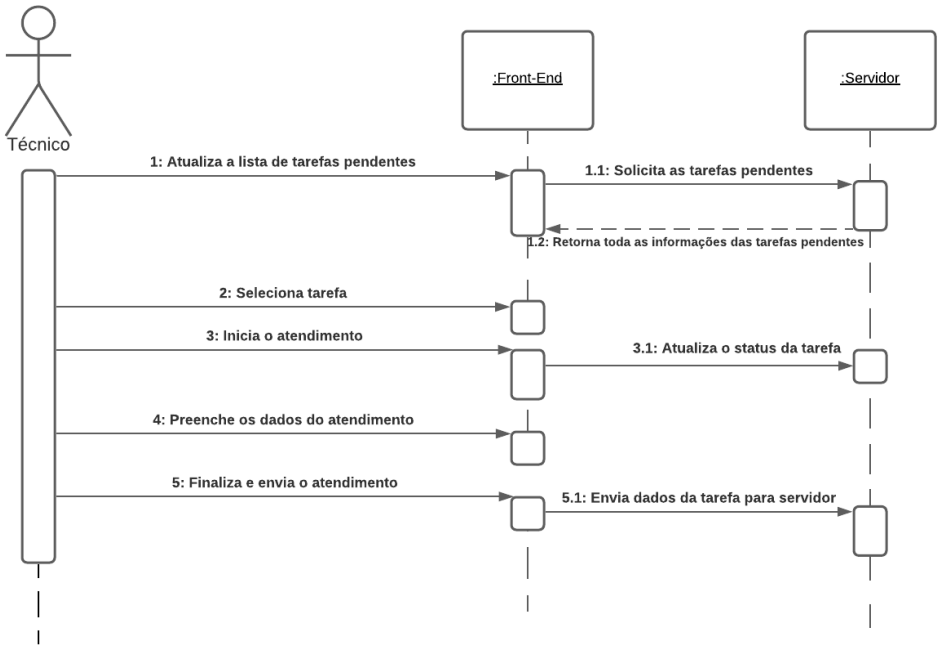

Figura 6. Diagrama de sequência alto nível do aplicativo

5.2 Arquitetura do sistema

Como representado na Figura 7, o software é composto por 2 módulos: Servidor (Banco de dados + Rest API) e Cliente Side (Aplicação Mobile + Aplicação Web). A interface do cliente faz conexão ao servidor, que tem a função de receber, processar e distribuir os dados. Cada sub-módulo da arquitetura será explicada detalhadamente nos capítulos seguintes.

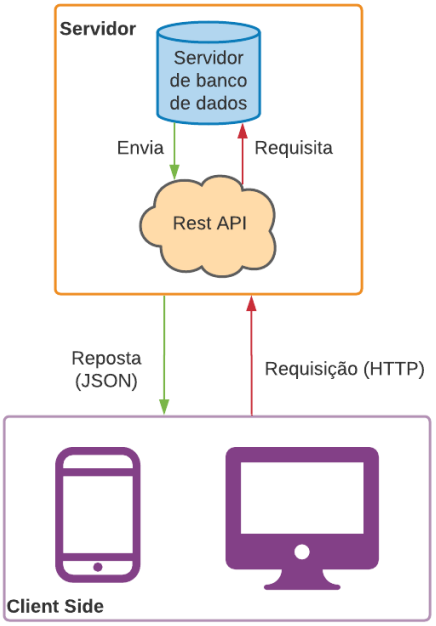

Figura 7. Arquitetura da aplicação

Para fins de esclarecimento, podemos tomar a requisição da Figura 8, como exemplo:

Figura 8. Requisição cadastro usuário

Está requisição POST recebe em seu corpo um JSON com todas as informações necessárias para cadastrar um usuário. Ela é enviada pela interface web para a API, o controller processa essa requisição, e envia para o gerenciador de banco de dados, que por fim registra as informações.

5. 3 Servidor de banco de dados

O gerenciador de banco de dados utilizado no sistema foi o Postgresql, o mesmo utilizado anteriormente. Porém, desta vez as tabelas foram otimizadas para uma maior eficiência e simplicidade do banco, a fim de atingir os objetivos propostos para o projeto. É importante destacar que o gerenciador de banco de banco de dados reside no mesmo servidor que a Rest API. Foi visto que anteriormente algumas tabelas e alguns campos não eram necessários, gerando um tempo de consulta maior e armazenando dados desnecessariamente, após uma análise e estudos, a estrutura do banco de dados foi refeita.

A modelagem completa do banco de dados se encontra no final do documento no Apêndice B junto com as descrições de cada tabela.

Para atingir melhores resultados com a mudança no banco de dados, muitas funções que eram tratadas em SQL foram passadas para o WebService, para evitar problemas e facilitar a manutenção.

Para fins de demonstração da criação de uma tabela no banco de dados, na Figura 9, é feita a criação da tabela de usuários. Esta tabela possui chaves primárias no campo **ID**, para identificar os dados inseridos no banco, e também possui chaves estrangeiras, denominadas de **tabela_id,** para referenciar tabelas. Além disso, todos os campos tiveram seus tipos escolhidos para ocupar menos espaço e também foram adicionados índices **B-tree** para acelerar as consultas.

```
CREATE TABLE users (
    id integer NOT NULL,
    email character varying(128) NOT NULL,
    password character varying(128) NOT NULL,
    phone character varying(32),
    name character varying(128),
    profile_picture character varying(256),
    status boolean DEFAULT true,
    type integer,
    created_at timestamp without time zone,
);

CREATE SEQUENCE users_id_seq
    AS integer
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;

ALTER TABLE ONLY public.users ALTER COLUMN id SET DEFAULT nextval('public.users_id_seq'::regclass);

ALTER TABLE ONLY public.users
    ADD CONSTRAINT users_pkey PRIMARY KEY (id);

CREATE INDEX user_email_index ON public.users USING btree (email);
```

Figura 9. SQL criação de usuários

5. 4 WebService (Rest API)

O grande diferencial do Webservice em relação ao projeto antigo, projetado para interligar toda a parte do usuário (Interface web e Mobile) ao banco de dados, responsável por toda comunicação de dados da aplicação.

Desenvolvida em Django, um framework em Python de alto nível que permite o rápido desenvolvimento de API's seguras e de fácil manutenção. Foi estruturado na arquitetura MVC (Model-View-Controller) que será explicada mais detalhadamente na seção 5.5, tornando o projeto mais organizado e com código mais limpo.

A API recebe requisições HTTP POST e GET, em geral, e monta objeto em JSON, representação de dados simples e legível, composta por chaves/valores. As chaves representam os nomes dos atributos e cada uma possui seus respectivos valores, esses dados são transmitidos para o lado do

cliente (Interface web ou mobile), lá eles são tratados para, ao final, serem mostrados para o usuário. Na Figura 10 é mostrado o JSON com as informações para cadastrar um usuário.

Figura 10. JSON criação de usuário

O desenvolvimento de uma REST API, foi pensado também para futuras integrações com sistemas de clientes, possibilitando a importação/exportação de dados.

5.5 Interface Web

A arquitetura da interface web sofreu uma grande mudança no novo projeto. Desenvolvida em AngularJS, este framework de código aberto facilita bastante o desenvolvimento de aplicações web. Ele segue os padrões de arquitetura MVC (Model-View-Controller) e usa o conceito de SPA (Single Page Application), onde a aplicação é desenvolvida em apenas uma página.

Como mencionado anteriormente, a comunicação entre a interface web e o gerenciador de banco de dados é feita pela API, que recebe uma requisição HTTP e retorna um JSON. O módulo que recebe e faz o tratamento do JSON, é controller. Sua função é decodificar o objeto recebido e enviar para o Model armazenar, onde será aplicada a lógica em questão e repassará os dados para a View, que irá apenas mostrar os dados para o cliente. A Figura 11 exemplifica com detalhes a estrutura da aplicação.

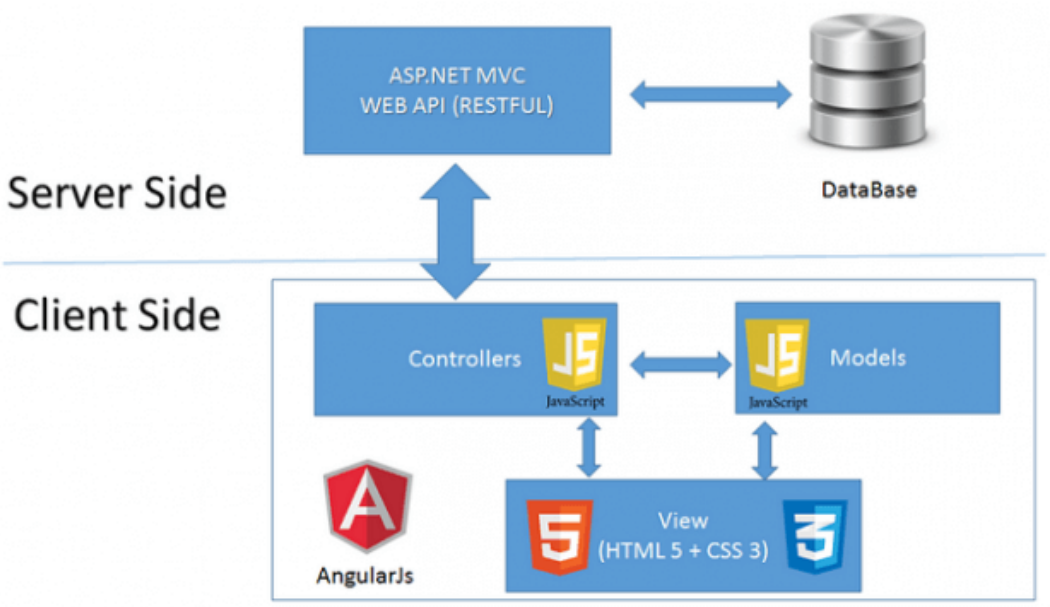

Figura 11. Arquitetura AngularJS com Rest API[16]

A estrutura do AngularJS é divida em três partes:

- **Model:** estrutura de dados que gerencia e armazena informações que são repassadas para o controller.
- **View:** É a parte com que o usuário vê e interage, ou seja, o front-end da aplicação, utilizando HTML e CSS.
- **Controller:** Toda a lógica por trás da visualização dos dados e faz comunicação com o Model.

5.6 Mobile Android

A grande mudança da aplicação Mobile foi sua linguagem, trocada de Java, para Kotlin, feita para Android, bastante semelhante à linguagem anterior. A arquitetura utilizada na interface android foi a MVP (Model-View-Presenter), que separa a interface do usuário, da lógica do sistema.

Para a comunicação do aplicativo com o webservice, foi utilizado o Retrofit e o GSON, as duas bibliotecas mais utilizadas para o consumo de API no Android. Enquanto o Retrofit tem a função de criar requisições HTTP para o servidor, o GSON tem a função de transformar a resposta do servidor em um objeto manipulável.

A Figura 12 abaixo ilustra como é feita a comunicação e o tratamento dos dados do aplicativo.

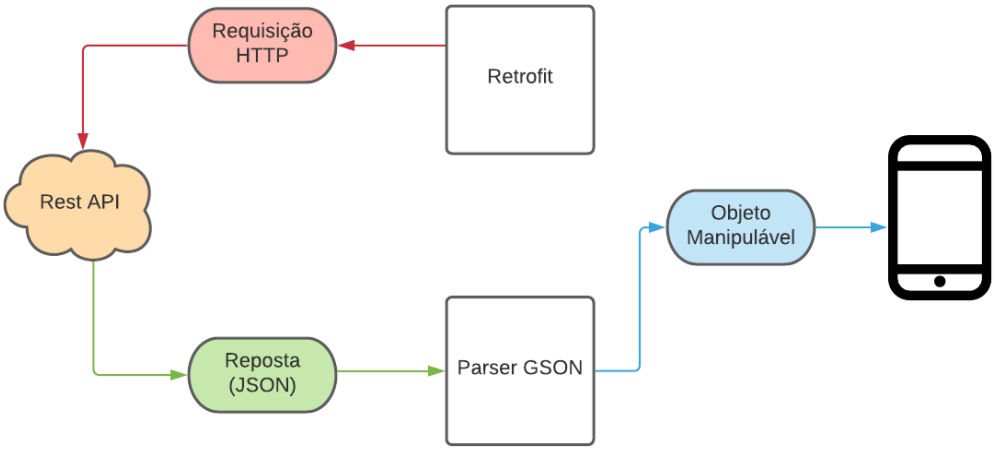

Figura 12. Integração Mobile com servidor

6. Implementação e avaliação

6. 1 Planejamento e execução de testes funcionais

Para testar e avaliar o sistema, visto que sua arquitetura conta com diversos módulos que possivelmente podem conter informações erradas ou bugs, foi escolhido pelo menos um teste funcional para cada módulo do sistema. Dessa forma, seria possível evitar boa parte dos possíveis problemas.

Para o banco de dados, foi feito um teste de integridade com o objetivo de injetar um alto volume de dados no banco e verificar se os módulos envolvidos iriam se manter íntegros, e também para verificar o comportamento do sistema em geral. Para isso foi usada a ferramenta JMeter, que auxiliou para popular a tabela de usuários com uma grande quantidade de dados. O sistema manteve respostas satisfatórias até cerca de 500 mil linhas inseridas na tabela de usuários.

Após o teste no Banco de Dados, foram realizados testes no servidor, a fim de verificar se as respostas esperadas estavam de acordo com as respostas recebidas. Para isso foram feitos testes unitários em cada controller da API, dessa forma seria possível checar eventuais falhas ou informações incorretas. É possível observar um exemplo de código na Figura 13 abaixo.

```python
from django.test import TestCase
from .models import User


class UserTestCase(TestCase):
    def setUp(self):
        User.objects.create(name="caique", email="caiquemolina@gmail.com",
                            password="123456", phone="(21)964666330",
                            type="0")

    def test_user_creation(self):
        """Verificar usuário criado e retornado correto"""
        user = User.objects.get(name="caique")
        self.assertEqual(user.name, 'caique')
        user = User.objects.get(email="caiquemolina@gmail.com")
        self.assertEqual(user.email, 'caiquemolina@gmail.com')
```

Figura 13. Teste unitário de criação de usuário

Ao longo do desenvolvimento dos módulos do cliente (Mobile e Web), visto que o servidor estava testado e para acelerar o projeto, foram feitos testes baseados em falha ou erro, que apareceram ao longo do processo de desenvolvimento. Para melhorar ainda mais a cobertura dos erros, foram semeados alguns erros comuns em sistemas, como: SQL INJECTION, inserção de valores nulos ou incondizentes, entre outros. Também foi usado teste unitário em alguns casos para facilitar a resolução dos erros.

Por fim, foram realizados testes de fumaça, que consistem em uma revisão das funcionalidades críticas do aplicativo, a fim de oferecer uma camada de proteção extra no projeto. Este teste geralmente é executado antes dos testes funcionais, porém usá-lo ao final dos testes, ofereceu uma segurança maior na aplicação. Ao final foram feitos alguns testes de usabilidade com algumas pessoas, para checar se o aplicativo estava simples e intuitivo.

Os testes descritos nessa seção foram executados na ordem da Figura 14:

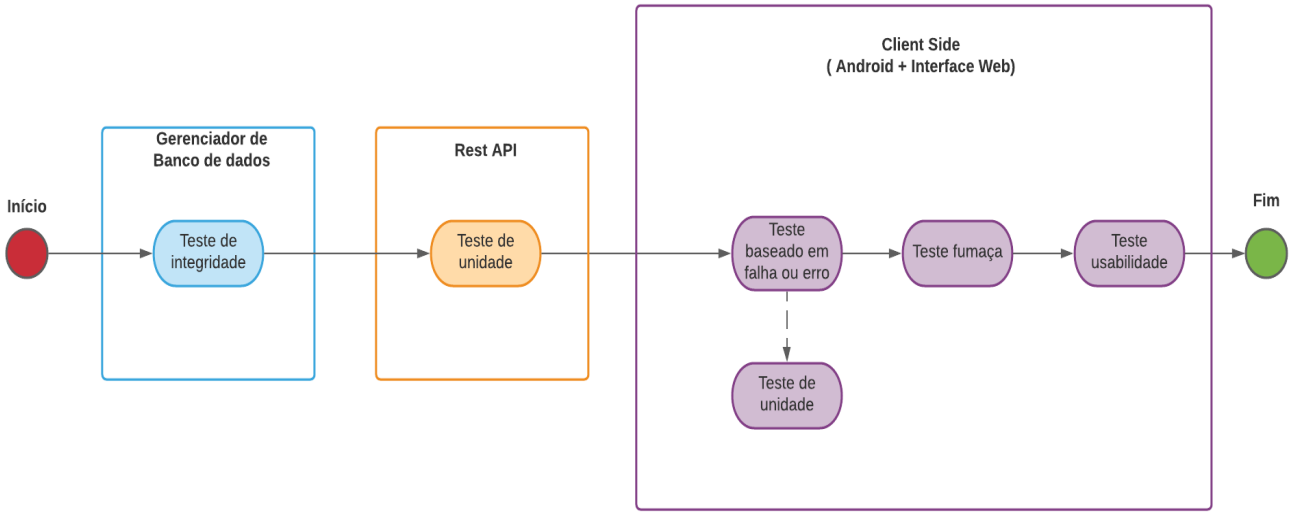

Figura 14. Planejamento dos testes funcionais

6.2 Planejamento e execução de testes de integração

Além dos testes funcionais do sistema descritos no tópico anterior, que são os mais importantes, por testarem as regras do negócio, é importante também realizar alguns testes de integração, até porque todos os módulos do sistema são integrados. Para isso, foi utilizado uma ferramenta para testar requisições bastante simples e prática, o **Postman.**

Para fins de demonstração, iremos analisar a requisição de criação de usuário, que recebe os parâmetros mostrados na Figura 15. Esta requisição deve retornar ao usuário criado do banco de dados junto com os parâmetros **created_at** e **id**.

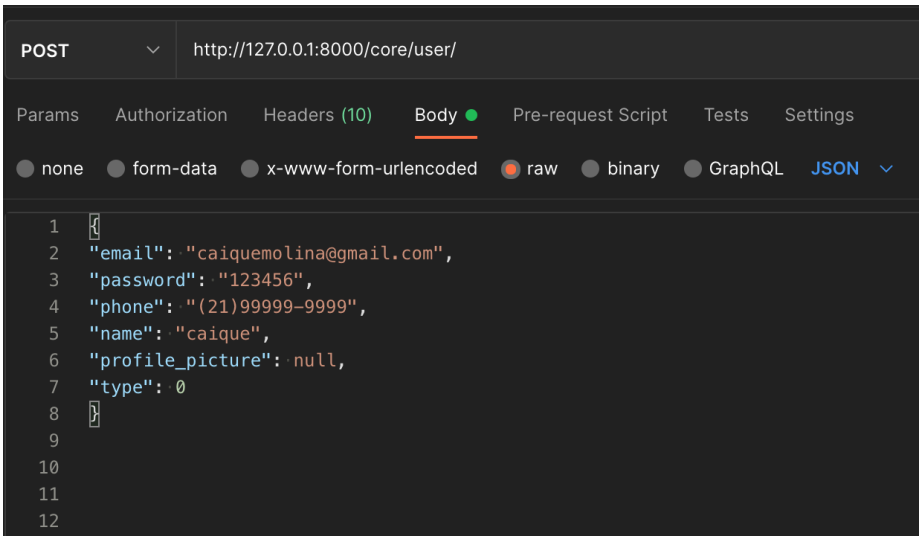

Figura 15. Envio requisição POST usuário

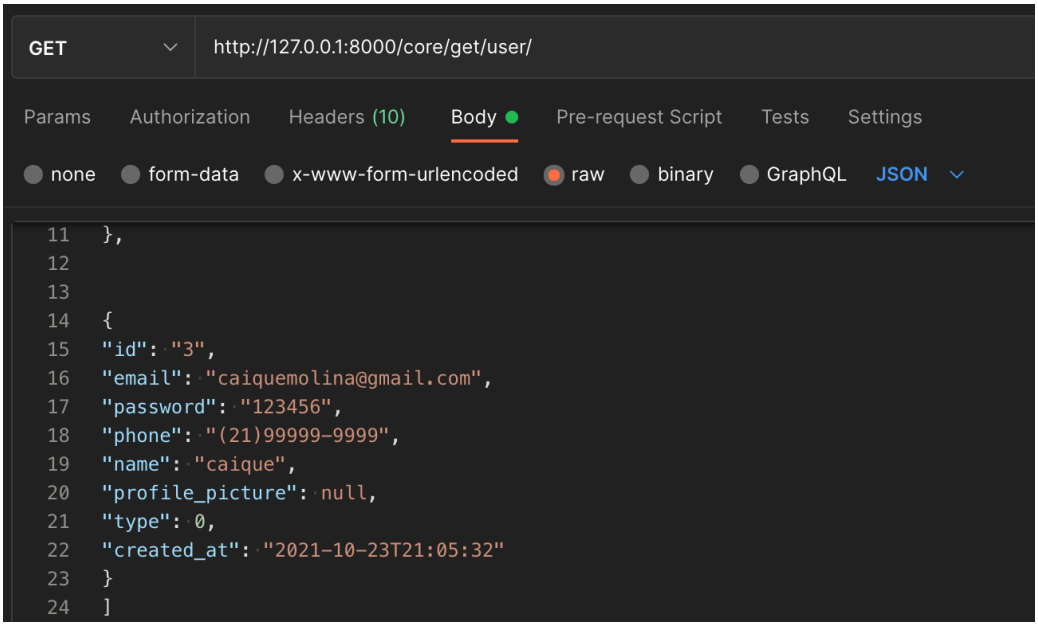

Figura 16. Retorno requisição GET usuário

Como pode ser observado na Figura 16, a requisição foi enviada com sucesso ao banco de dados e foi retornado o usuário com os parâmetros esperados.

6.3 Comentários sobre a implementação

Após o lançamento do sistema pela primeira vez, diversos problemas surgiram ao longo do tempo, demonstrando a importância de uma boa metodologia de desenvolvimento e de muitos testes. No entanto, a experiência obtida com os erros iniciais foi muito importante para a elaboração de um projeto mais confiável e funcional.

As pesquisas iniciais sobre as melhores tecnologias que se encaixavam no projeto, possibilitaram uma arquitetura organizada, de fácil manutenção e escalável para novas funcionalidades. Além disso, testes funcionais forneceram uma camada extra de segurança ao projeto e evitaram problemas que poderiam surgir no futuro.

6.4 Demonstrações e usos do sistema

Como proposto inicialmente, o aplicativo chamado *EquipeOn,* tem como foco ser um gerenciador de tarefas para empresas. Para atender esse objetivo foi preciso desenvolver duas interfaces para o cliente: uma interface web e o aplicativo Mobile, cada uma delas com suas características específicas que serão elencadas.

- **Aplicativo (suas funcionalidades podem ser observadas na** Figura 17**, na respectiva ordem)**

- **Listagem de tarefas:**

  Na tela de listagem de tarefas o usuário pode checar todas as ordens de serviço pendentes para serem atendidas.

- **Confirmação de tarefa:**

  Uma tela para confirmação com mais detalhes sobre a tarefa escolhida.

- **Atendimento da tarefa:**

  Tela de atendimento, onde o usuário irá preencher o formulário de atendimento previamente cadastrado na interface web.

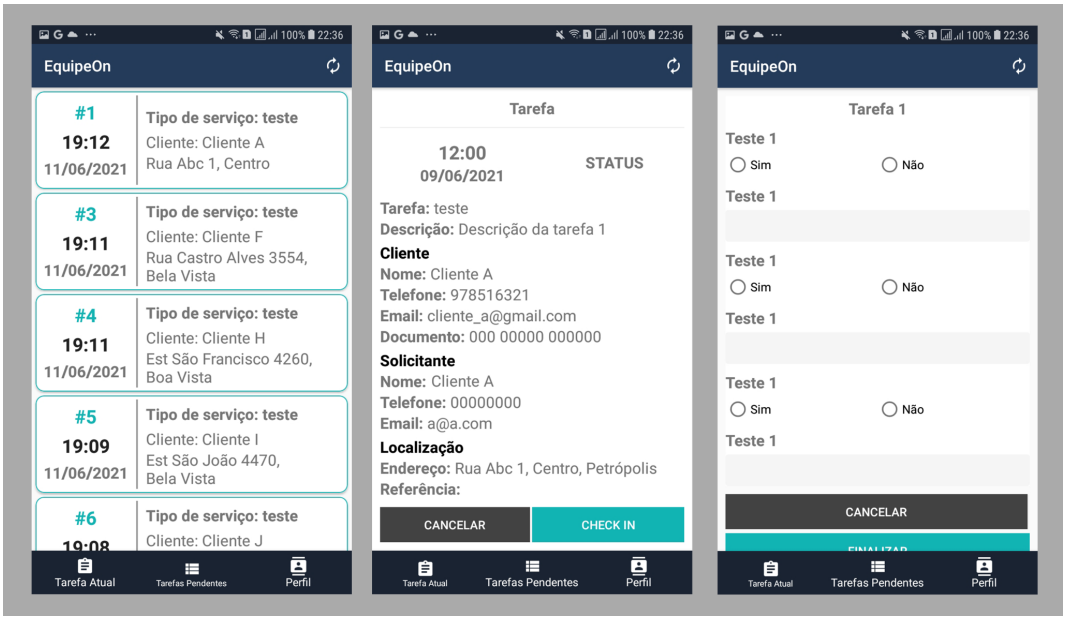

Figura 17. Telas principais do aplicativo

- **Interface Web**

- **Personalização de formulário:**

A livre criação de formulários por cada empresa é o diferencial neste aplicativo e o que faz ele ser útil para qualquer empresa que presta serviços. Ao designar uma ordem de serviço para um usuário de sua empresa, o atendente precisa escolher o formulário de atendimento que será respondido pelo técnico em seu smartphone.

Na tela de criação de formulários como é mostrado na Figura 18 abaixo, o usuário pode escolher 4 tipos perguntas que irão o smartphone do técnico:

1- Opção "Sim ou não": Esse campo é uma "checkbox" com apenas duas opções de resposta, "Sim ou Não".

2- Opção "Texto": Esse campo possibilita criar um textfield para qualquer tipo de resposta.

3- Opção "Foto": Esse campo permite que o técnico tire uma foto do atendimento com seu celular.

4- Opção "Avaliação": Esse campo cria um campo de avaliação de 1-5 para o técnico avaliar algo.

Figura 18. Personalização de formulário

- **Listagem de tarefas:**

A criação de ordens de serviço é a principal funcionalidade da aplicação, por isso foi o último recurso a ser desenvolvido, onde foi focado um layout simples e intuitivo para facilitar e agilizar a criação de tarefas por atendentes, além de ser a função mais utilizada no sistema, como pode ser observado na Figura 19.

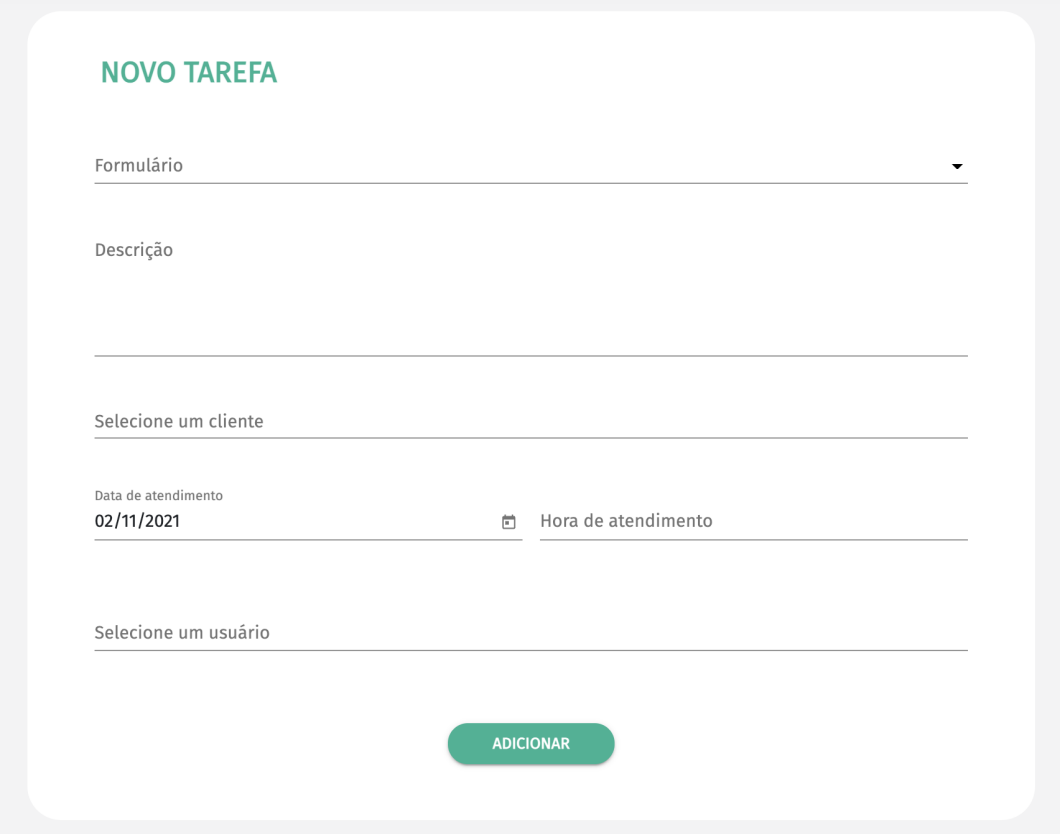

Figura 19. Criação de nova tarefa

- **Criação/Edição (Usuários, Clientes, Tarefas, Formulário e Times):**

Para o funcionamento da aplicação é preciso que a empresa cadastre os dados gerais de sua empresa, isso inclui: clientes, usuário, equipes e formulários de atendimento. Um exemplo de tela de criação de cliente é mostrado na Figura 20, todas as outras telas seguem o mesmo layout.

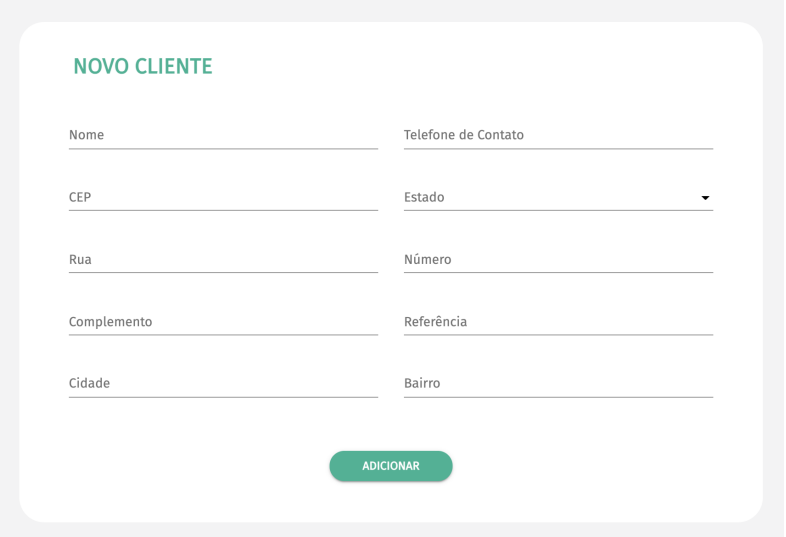

Figura 20. Criação de novo cliente

7. Considerações finais

7.1 Pontos negativos da arquitetura escolhida

Embora o desenvolvimento do projeto tenha sido um sucesso, com uma arquitetura que entrega o que foi proposto. É importante destacar os pontos negativos dela, até porque nenhuma arquitetura se mostra totalmente perfeita, com a alta demanda mundial por tecnologias e inovações, todos os meses é possível notar novidades que são introduzidas no mercado.

Linguagem não-híbrida: Os módulos do cliente (Web e Mobile) foram desenvolvidos cada um com sua linguagem, porém, a partir de 2021 linguagens híbridas ganharam força, como o Flutter e React, compilando os mesmo códigos tanto para Mobile, quanto para Web, o que facilitaria o desenvolvimento e manutenção.

Complexidade de requisições: Algumas requisições do sistema, devido a grande quantidade de tabelas do banco de dados (Figura 22), ficaram bastante complexas, retornando JSON's com muita informação.

Em conclusão, a arquitetura escolhida teve diversas vantagens, como sua modularidade, simplicidade de código, robustez para testagem, assim como algumas desvantagens, como a quantidade de linguagens de programação envolvidas e a complexidade do banco de dados. Alguns pontos da arquitetura foram otimizados no processo de desenvolvimento e outros ficaram pendentes para o futuro.

7.2 Possíveis expansões do projeto

O grande benefício da arquitetura escolhida é sua modularidade, sendo todos os componentes interligados pela Rest API, essa característica facilita sua expansão e aprimoramento para o futuro.

Como abordado no tópico anterior, o módulo cliente será refeito em breve, utilizando a linguagem Flutter, que vem ganhando bastante popularidade nos últimos tempos. Essa mudança possibilitará um aplicativo funcional para IOS e Android, além de uma interface web responsiva, tudo isso compartilhando do mesmo código fonte.

Além disso, a aplicação será levada para outras empresas parceiras do cliente inicial, de onde o sistema surgiu. Com o uso de novos clientes, o projeto ganhará mais feedbacks e consequentemente essa experiência irá evoluir o aplicativo.

Portanto, o aplicativo seguirá crescendo, visto que a demanda no mercado só aumenta, especialmente com o surgimento da pandemia do COVID-19, as empresas foram praticamente obrigadas a passar por uma transformação digital, então a procura por soluções que otimizem seus trabalhos e reduzam seus custos, aumentou, o que será uma ótima oportunidade para expandir o projeto.

8. Referências Bibliográficas

[1] GO.ON. **Website.** Disponível em: https://goon.mobi. Acesso em 20 de novembro de 2021

[2] Field Control. **Website.** Disponível em: https://fieldcontrol.com.br. Acesso em 20 de novembro de 2021

[3] Django. ROVEDA, UGO. **Kenzie.** Disponível em: https://kenzie.com.br/blog/django/. Acesso em 4 de outubro de 2020

[4] PYTHON. **Wikipédia.** Disponível em: https://pt.wikipedia.org/wiki/Python. Acesso em 5 de outubro de 2020

[5] JSON. **Wikipédia.** Disponível em: https://pt.wikipedia.org/wiki/JSON. Acesso em 1 de novembro de 2020

[6] Kotlin. **Wikipédia.** Disponível em: https://pt.wikipedia.org/wiki/Kotlin. Acesso em 2 de outubro de 2020

[7] AngularJS. **AngularJS ORG.** Disponível em: https://docs.angularjs.org/tutorial. Acesso em 15 de junho de 2021

[8] RATAMERO, LUCIANO. Boas práticas e organização Github. **Medium.** Disponível em: https://medium.com/@lucianoratamero/git-e-github-parte-3-boas-pr%C3%A1ticas-de-organiza%C3%A7%C3%A3o-de-branches-lucianoratamero-4d786c759bd2.

[9] Introduction to Postman. **Postman.** Disponível em: https://learning.postman.com/docs/getting-started/introduction/. Acesso em 3 de outubro de 2020

[10] Android Studio. **Wikipédia.** Disponível em: https://pt.wikipedia.org/wiki/Android_Studio. Acesso em 1 de novembro de 2020

[11] Guia de produtividade Visual Studio. **Microsoft.** Disponível em: https://docs.microsoft.com/pt-br/visualstudio/ide/productivity-features?view=vs-2019. Acesso em 5 de dezembro de 2020

[12] Guia completo PostgreSQL. **Devmedia.** Disponível em: https://www.devmedia.com.br/guia/guia-de-postgresql/34328. Acesso em 5 de outubro de 2020

Acesso em 2 de novembro de 2020

[13] ZATS, IGOR. Laravel Vs Django. **Keyua.** Disponível em: https://keyua.org/blog/laravel-vs-django-analysis-and-comparison/. Acesso em 30 de março de 2021

[14] O que é arquitetura MVC. **Wikipédia.** Disponível em: https://www.lewagon.com/pt-BR/blog/o-que-e-padrao-mvc. Acesso em 4 de novembro de 2020

[15] Performance Comparison Frameworks. **Fullscale.** Disponível em: https://fullscale.io/blog/django-vs-laravel-performance-comparison-of-web-application-frameworks/. Acesso em 30 de março de 2021

[16] Arquitetura AngularJS. **imasters**. Disponível em:https://imasters.com.br/dotnet/asp-net-mvc-5-criando-uma-aplicacao-com-angularjs-e-web-api-parte-01. Acesso em 30 de março de 2021

Apêndice A - Especificação de requisitos

Caso de Uso UC1 : Autenticação
Nível: Primário
Ator primário: Usuário
Interessados e interesses:

- Administrador: Deseja logar ao painel de controle (web) para ter acesso aos dados de sua empresa
- Técnico: Deseja logar no app para consultar suas ordens de serviço pendentes

Pré-condições: O usuário deve ter sido cadastrado
Pós-condições: O usuário é autenticado e tem acesso ao sistema, ilimitado às suas permissões
Requisitos necessários: O email e senha do usuário precisam coincidir
Frequência: Baixa
Fluxo Principal:

1. O usuário abre o aplicativo para se logar
2. O sistema requisita login e senha
3. O usuário informa login e senha
4. O usuário é logado

Extensões:

3a. O usuário fornece login ou senha errados
1 - O sistema sinaliza o erro e rejeita a entrada

Caso de Uso UC2 : Listar/Consultar ordens de serviço
Nível: Primário
Ator primário: Usuário
Interessados e interesses:

- Administrador: Deseja consultar as ordens de serviço abertas no sistema e seus respectivos status
- Técnico: Deseja consultar as ordens de serviço pendentes para serem executadas por ele

Pré-condições: É preciso ter ordens de serviço cadastradas no sistema, assim como clientes, usuários e formulários.
Pós-condições: Será mostrado todas as ordens de serviço no sistema
Frequência: Alta
Fluxo Principal:

1. O técnico abre o celular para consultar suas ordens de serviço
2. O sistema lhe retorna as tarefas pendentes

Fluxo Secundário:

1. O atendente abre a interface web para consultar o andamentos das tarefas
2. O sistema retorna uma lista com todas as tarefas

Caso de Uso UC3 : Atender ordem de serviço
Nível: Primário
Ator primário: Técnico

Interessados e interesses:

- Técnico: Deseja atender uma ordem de serviço, a fim de finalizar sua listagem de serviços

Pré-condições: É preciso ter ordens de serviço cadastrada para o usuário em questão.
Pós-condições: Será enviada a ordem de serviço finalizada para o painel de controle web.
Frequência: Alta
Questões em aberto:

- O técnico pode atender uma ordem de serviço sem estar no cliente?
- O que acontece se o problema não for resolvido?

Fluxo Principal:

1. O técnico escolhe uma ordem de serviço para atender
2. O sistema mostra os detalhes da ordem de serviço em questão
3. O técnico inicia o atendimento
4. O sistema requisita que os formulário sejam preenchidos
5. O usuário preenche o formulário e finaliza a tarefa.
6. O sistema encaminha o usuário para a lista de tarefas

Extensões:

2a. O usuário já tem uma tarefa iniciada
1 - O sistema sinaliza que já tem uma tarefa iniciada
5a. O usuário tenta finalizar o formulário sem todos os campos preenchidos
1 - O sistema sinaliza o erro e requisita todas as informações

Caso de Uso UC4 : Consultar equipes
Nível: Primário
Ator primário: Administrador
Interessados e interesses:

- Administrador: deseja consultar as equipes cadastradas no sistema

Pré-condições: É preciso estar com sua empresa cadastrada no sistema
Pós-condições: Será mostrado todas as equipes no sistema
Frequência: Baixa

Questões em aberto:

- Um usuário pode estar em mais de uma equipe?

Fluxo Principal:

1. O atendente deseja consultar as equipes cadastradas
2. O sistema retorna a lista de equipes cadastradas no sistema

Extensões:

1a. O usuário não vê nenhuma equipe
1 - O sistema segure o cadastro de uma nova equipe

Caso de Uso UC5 : Cadastrar equipes
Nível: Primário
Ator primário: Administrador
Interessados e interesses:

- Administrador: deseja cadastrar equipes no sistema

Pré-condições: É preciso estar com sua empresa cadastrada no sistema
Pós-condições: Será cadastrado a equipe no sistema
Frequência: Baixa
Fluxo Principal:

1. O atendente deseja consultar os usuários cadastrados
2. O sistema retorna a lista de usuários cadastrados no sistema

Caso de Uso UC6 : Consultar usuários
Nível: primário
Ator primário: Administrador
Interessados e interesses:

- Administrador: deseja consultar os usuários cadastrados no sistema

Pré-condições: É preciso estar com sua empresa cadastrada no sistema
Pós-condições: Será mostrado todas os usuários no sistema
Frequência: Baixa

Caso de Uso UC7 : Cadastrar usuários
Nível: Primário
Ator primário: Administrador
Interessados e interesses:

- Administrador: deseja cadastrar os usuários no sistema

Pré-condições: É preciso estar com sua empresa cadastrada no sistema
Pós-condições: O usuário será cadastrado no sistema
Frequência: Baixa

Caso de Uso UC8 : Consultar clientes
Nível: Primário
Ator primário: Administrador
Interessados e interesses:

- Administrador: deseja consultar os clientes no sistema

Pré-condições: É preciso estar com sua empresa cadastrada no sistema
Pós-condições: Será mostrado todos os clientes cadastrados no sistema
Frequência: Média

Caso de Uso UC9 : Cadastrar clientes
Nível: Primário
Ator primário: Administrador
Interessados e interesses:

- Administrador: deseja cadastrar clientes no sistema

Pré-condições: É preciso estar com sua empresa cadastrada no sistema
Pós-condições: Será cadastrado o cliente no sistema
Frequência: Média

Caso de Uso UC10 : Cadastrar ordem de serviço
Nível: Primário
Ator primário: Administrador
Interessados e interesses:

- Administrador: deseja cadastrar ordens de serviço no sistema

Pré-condições: É preciso que esteja cadastrado pelo menos um cliente, um usuário e um formulário de atendimento.
Pós-condições: Será cadastrado uma ordem de serviço no sistema
Frequência: Alta
Fluxo Principal:

1. O atendente deseja cadastrar uma ordem de serviço
2. O sistema a tela para cadastro de ordem de serviço
3. O atendente preenche os dados necessários para a tarefa
4. O sistema cadastra a ordem de serviço

Extensões:

3a. O atendente não preenche qualquer um dos dados
1 - O sistema sinaliza que todos os dados são necessários
3b. O atendente escolhe um horário já existente no sistema
1 - O sistema sinaliza e pede para escolher outro horário

Caso de Uso UC11 : Consultar formulários de atendimento
Nível: Primário
Ator primário: Administrador
Interessados e interesses:

- Administrador: deseja consultar os formulário de atendimento cadastrados no sistema

Pré-condições: É preciso estar com sua empresa cadastrada no sistema
Pós-condições: Será mostrado todas os formulários de atendimento no sistema
Frequência: Média

Caso de Uso UC12 : Cadastrar formulários de atendimento
Nível: Primário
Ator primário: Administrador
Interessados e interesses:

- Administrador: deseja cadastrar os formulários de atendimento no sistema

Pré-condições: É preciso estar com sua empresa cadastrada no sistema
Pós-condições: O formulário de atendimento será cadastrado no sistema
Frequência: Média

Apêndice B - Modelos

- **Modelo conceitual de dados**

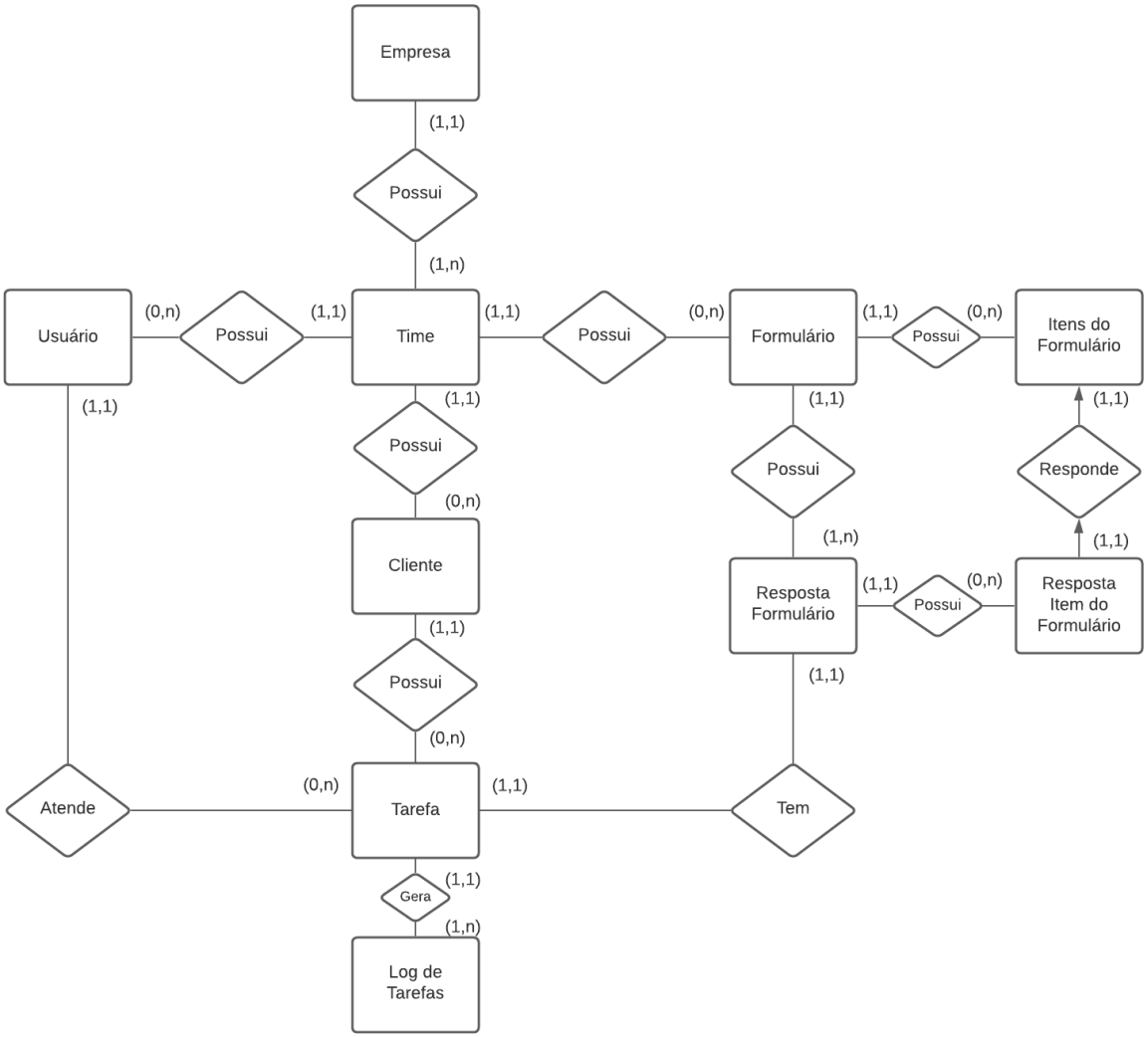

Figura 21. Modelo conceitual de dados

- **Representação do modelo de banco de dados**

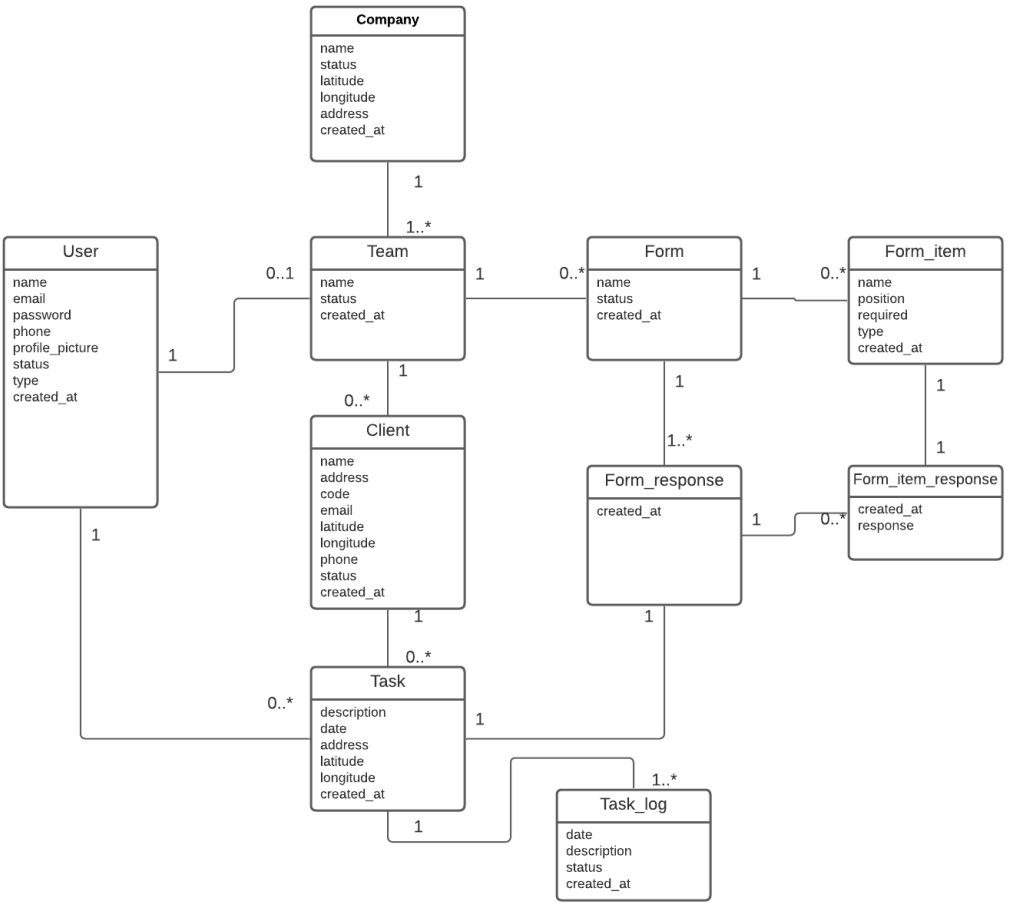

Figura 22. Representação do modelo de Banco de Dados

Foram utilizados índices para otimizar consultas, chaves primárias e chaves estrangeiras, cada tabela do banco tem a seguinte função:

- **Clients:** Armazena todos os dados relacionados aos clientes, que são usados para criação de ordem de serviço.
- **Users:** Armazena os dados de usuários, que são diferenciados pelo campo “Type”, onde é possível criar 3 tipos de usuários: Administrador, Atendente e Técnico.
- **Teams:** Tabela para criar times de atuação na empresa, para organizar os técnicos e atendentes.
- **Tasks:** Tabela principal da aplicação onde armazena as ordens de serviços, onde é preciso selecionar usuário, cliente e um formulário.
- **Task_logs:** Tabela que armazena os status das ordens de serviço, que são: Abertas, Em atendimento, Finalizadas e Canceladas.

- **Company:** Armazena os dados de cada empresa, sendo o nível mais alto.
- **Form:** Tabela que armazena os formulários de atendimento da empresa, essa estrutura é composta por outras tabelas auxiliares que permitem a personalização dos formulários.
- **Form_items:** Armazena as perguntas para um formulário específico, onde é possível criar opções de texto, foto, avaliações, entre outras
- **Form_response:** Responsável por relacionar a tabela de Tarefas com o formulário respondido.
- **Form_item_response:** Tabela para armazenar as respostas de cada item do formulário.