UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

CAMILA EMANUELE DE SOUZA

ATUALIZAÇÃO DO BACKEND DO SISTEMA OPEN SOCIAL CARE: MIGRANDO DA ARQUITETURA SERVERLESS PARA UMA API EM LARAVEL

GUARAPUAVA

2024

CAMILA EMANUELE DE SOUZA

ATUALIZAÇÃO DO BACKEND DO SISTEMA OPEN SOCIAL CARE: MIGRANDO DA ARQUITETURA SERVERLESS PARA UMA API EM LARAVEL

OPEN SOCIAL CARE SYSTEM BACKEND UPDATE: MIGRATING FROM SERVERLESS ARCHITECTURE TO LARAVEL API

Projeto de Trabalho de Conclusão de Curso de Graduação apresentado à disciplina de Trabalho de Conclusão de Curso, do Curso Superior de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná, Campus Guarapuava, como requisito parcial para obtenção do título de Tecnólogo em Tecnologia em Sistemas para Internet.

Orientador: Prof. Dr. Andres Jessé Porfirio

Coorientador: Gustavo Vicari Duarte

GUARAPUAVA

2024

4.0 Internacional

Esta licença permite compartilhamento, remixe, adaptação e criação a partir do trabalho, mesmo para fins comerciais, desde que sejam atribuídos créditos ao(s) autor(es). Conteúdos elaborados por terceiros, citados e referenciados nesta obra não são cobertos pela licença.

CAMILA EMANUELE DE SOUZA

ATUALIZAÇÃO DO BACKEND DO SISTEMA OPEN SOCIAL CARE: MIGRANDO DA ARQUITETURA SERVERLESS PARA UMA API EM LARAVEL

Projeto de Trabalho de Conclusão de Curso de Graduação apresentado à disciplina de Trabalho de Conclusão de Curso, do Curso Superior de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná, Campus Guarapuava, como requisito parcial para obtenção do título de Tecnólogo em Tecnologia em Sistemas para Internet.

Data de aprovação: 03/julho/2024

Prof. Andres Jessé Porfirio
Doutor
Universidade Tecnológica Federal do Paraná - Campus Guarapuava

---

Prof. Sediane Carmem Lunardi Hernandes
Doutora
Universidade Tecnológica Federal do Paraná - Campus Guarapuava

---

Prof. Dênis Lucas Silva
Mestre
Universidade Tecnológica Federal do Paraná - Campus Guarapuava

GUARAPUAVA

2024

Dedico este trabalho ao meu amor, minha
família e professores, obrigado pelo apoio.

AGRADECIMENTOS

Agradeço a todos que contribuíram para a realização desse trabalho. Aos amigos e colegas pelas falas de carinho. Aos professores por toda assistência e dedicação. Meus sinceros agradecimentos por toda ajuda e incentivo.

As assistentes sociais do Conselho da comunidade de Guarapuava, que vieram em busca de uma resolução para os seus problemas, que acolheram a ideia de um sistema, como também ajudaram para que o desenvolvimento do projeto Open social care fosse possível.

Expresso minha gratidão ao meu orientador Professor Andres Jessé Porfirio, que dedicou do seu tempo para dar apoio a realização do trabalho e ao projeto Open Social Care. Obrigado por ter me ajudado nesse caminho com seus ensinamento, incentivos e paciência. Tudo isso foi fundamental para a realização desse trabalho.

Ao meu coorientador Gustavo Vicari Duarte, por ter compartilhados seus conhecimentos e ajudado a realização desse trabalho.

À minha família a qual me encorajou e prestou imenso apoio, em especial minha mãe que em momentos de incertezas, prestou carinho e palavras de conforto, a todo o sacrifico que precisou fazer para que meus objetivos pudessem ser alcançados. Graças ao suporte, carinho e paciência este trabalho pode ser concluído, assim como também, a conclusão da minha jornada acadêmica.

RESUMO

O Open Social Care é um sistema para cadastro de atendimentos sociais idealizado para ajudar assistentes sociais a terem uma melhor gestão do seu trabalho e ser um sistema de livre usabilidade e gratuito. O seu protótipo inicial foi desenvolvido usando uma infraestrutura em nuvem, inicialmente interessante para ser usado pelas assistentes sociais, porém limitada com relação ao uso real, além de ter sido idealizado para ser utilizado somente em instituições da execução penal. Então, este projeto propõe a substituição desta infraestrutura, visando melhorar a eficiência e escalabilidade do sistema, como também propor uma forma de generalização para que a aplicação possa atender outras instituições além da Execução Penal. A nova infraestrutura é desenvolvida a partir de uma *Application Programming Interface* (API) para ser robusta e flexível, permitindo que diferentes partes interessadas, seja instituições de Execuções Penais, organizações sem fins lucrativos ou outros prestadores de serviços sociais, acessem e gerenciem os registros de atendimento de forma mais eficaz. Visando atender cenários mais abrangentes a API é pensada para facilitar a realização de manutenções e personalizações. Assim, o desenvolvimento da API representa um passo significativo para a melhoria do sistema Open Social Care, visando atender diferentes instituições de forma mais eficiente e versátil, contribuído para a realização de atendimentos sociais.

Palavras-chave: sistema para atendimento social; assistentes sociais; substituicao de infraestrutura em nuvem; desenvolvimento de api.

ABSTRACT

Open Social Care is a system for registering social services, designed to help social workers better manage their work and to be free and open to use. Its initial prototype was developed using a cloud infrastructure, which was initially interesting for use by social workers, but limited in terms of actual use, as well as being designed to be used only in penal institutions. Therefore, this project proposes replacing this infrastructure in order to improve the efficiency and scalability of the system, as well as proposing a form of generalization so that the application can serve institutions other than the Penal Service.The new infrastructure will be developed on the basis of an API to be robust and flexible, allowing different stakeholders, be they penal institutions, non-profit organizations or other social service providers, to access and manage care records more effectively. In order to address broader scenarios, the API is designed to facilitate maintenance and customization. Thus, the development of the API will be a significant step towards improving the *Open Social Care* system, aiming to serve different institutions in a more efficient and versatile way, contributing to the provision of social care.

Keywords: social service system; social workers; cloud infrastructure replacement; development api.

LISTA DE FIGURAS

Figura 1 – Documento Prontuário SUAS. **16**

Figura 2 – Tela inicial IDS Social **17**

Figura 3 – Tela de monitoramento em tempo real GESUAS **18**

Figura 4 – Tela de gestão financeira do sistema SociÁgil (Dashboard) **19**

Figura 5 – Tela de gerenciamento de banco de dados do Open Social Care no Firebase. **21**

Figura 6 – Diagrama de casos de uso para os perfis do sistema **25**

Figura 7 – Quadro kanban das atividades no github **26**

Figura 8 – GitHub branchs **27**

Figura 9 – Comentário de uma revisão **28**

Figura 10 – *Build log do deploy* CapRover **28**

Figura 11 – Tela para cadastro de um formulário *Google* **30**

Figura 12 – Fragmento do diagrama de banco de dados destacando as organizações e usuários **34**

Figura 13 – Fragmento do diagrama de banco de dados destacando os sujeitos . . **34**

Figura 14 – Fragmento do diagrama de banco de dados destacando os atendimentos e questionários personalizados **35**

Figura 15 – Teste de *feature* criação de usuário **43**

Figura 16 – Teste unitário para a classe UserCreateAction **44**

Figura 17 – *pipeline CI* dos testes **45**

Figura 18 – Fragmento da documentação *Swagger* de autenticação **46**

Figura 19 – Diagrama do banco de dados **53**

LISTA DE TABELAS

Tabela 1 – Listagem de requisitos básicos do sistema 29

Tabela 2 – Tabela de tarefas 41

LISTAGEM DE CÓDIGOS FONTE

Listagem 1 – *Migration* criada para os *subjects* **38**
Listagem 2 – *Model* criada para o *subject* **39**
Listagem 3 – Teste unitário criado para o *model* de *subject* **40**
Listagem 4 – *Factory* criada para o *model* de *subject* **40**

LISTA DE ABREVIATURAS E SIGLAS

Siglas

ACID	Atomicidade, Consistência, Isolamento, Durabilidade
API	*Application Programming Interface*
CCG	Conselho da Comunidade da Comarca de Guarapuava
CRAS	Centro de Referência de Assistência Social
CREAS	Centro de Referência Especializado de Assistência Social
ER	Entidade relacionamento
HTML	*HyperText Markup Language*
HTTP	Hypertext Transfer Protocol
IU	Interface do usuário
NoSQL	*Not Only SQL*
PHP	*Hypertext Preprocessor*
PPLs	Pessoas Privadas de Liberdade
SDK	*Software Development Kit*
SMAS	Secretaria Municipal de Assistência Social
SQL	*Structured Query Language*
SUAS	Sistema Único de Assistência Social
UTFPR	Universidade Tecnológica Federal do Paraná
WEB	World Wide Web

SUMÁRIO

1 INTRODUÇÃO 12
1.1 Considerações iniciais 12
1.2 Objetivos 13
1.2.1 Objetivo geral 13
1.2.2 Objetivos específicos 13
1.3 Justificativa 14
1.4 Estrutura do trabalho 15
2 TRABALHOS RELACIONADOS 16
2.1 Prontuário SUAS 16
2.2 IDS Social 17
2.3 GESUAS 17
2.4 SociÁgil 18
2.5 Comparativo 19
3 O OPEN SOCIAL CARE 20
4 MATERIAIS E MÉTODOS 22
4.1 Materiais 22
4.1.1 Git e Github 22
4.1.2 PHP e Laravel 22
4.1.3 Docker 23
4.1.4 PostgreSQL 23
4.1.5 CapRover 24
4.2 Métodos 24
4.2.1 Análise e coleta de requisitos 24
4.2.2 Organização de tarefas para o desenvolvimento 25
4.2.3 Planejamento do banco de dados 26
4.2.4 Fluxo de desenvolvimento 27
5 RESULTADOS 29
5.1 Requisitos 29
5.2 Generalização do sistema 29
5.3 Segurança 31

5.4 Escopo do sistema 32
5.5 Modelagem do sistema 33
5.6 Implementação 36
5.7 Testes automatizados 42
5.8 Documentação da API 45
5.9 Trabalhos futuros 47
6 CONSIDERAÇÕES FINAIS 48
REFERÊNCIAS 49

1 INTRODUÇÃO

Esta sessão explana uma breve descrição do atual sistema, quais seus problemas e suas soluções.

1.1 Considerações iniciais

O Conselho da Comunidade é um dos órgãos da Execução Penal, regulado pela Lei 7.210, de 11/07/1984, - Lei de Execução Penal - Estas instituições representam a real possibilidade de intervir nas relações sociais dentro e fora da prisão [...] (OLIVEIRA, 2012). O Conselho da Comunidade existe para auxiliar juízes durante a execução penal, gerenciando documentação e garantindo os direitos às Pessoas Privadas de Liberdade (PPLs), como o recebimento de roupas e sapatos, como também que os direitos humanos sejam respeitados. Os conselhos também prestam atendimentos para a emissão de antecedentes criminais, obtenção de documentação civil, busca de materiais e recursos solicitados por/para PPLs ou familiares, além de elaborar projetos para remição de pena voltado para atividades de caráter humanizado. Na cidade de Guarapuava - PR, uma das atribuições do Conselho da Comunidade é o serviço social, onde as assistentes sociais realizam atendimentos às PPLs, sendo de grande importância o registro e o armazenamento dessas informações de forma segura, eficiente, disponível e gerenciável.

Atualmente a gestão dos atendimentos realizados pelas assistentes sociais é conduzida com registros em papel ou em arquivos digitais de texto, procedimentos estes suscetíveis a falhas. Estes procedimentos podem ser falhos tanto do ponto de vista humano, pois exigem a conferência manual dos dados, como também em relação ao material ou meio de armazenamento, gerando dificuldades na busca por informações, como também podendo gerar a perda de documentos. O Open Social Care é um sistema que visa auxiliar nas atividades desenvolvidas pelas assistentes sociais, provendo um melhor gerenciamento das informações coletadas em seus atendimentos.

O sistema teve sua primeira fase de desenvolvimento na disciplina de Sistemas Distribuídos da Universidade Tecnológica Federal do Paraná (UTFPR), câmpus Guarapuava, onde foi proposta uma plataforma básica (um protótipo) apenas para a verificação dos requisitos e compreensão do problema, com isso, estudando-se o potencial de contribuição de um sistema deste tipo na rotina das assistentes sociais. Com a primeira fase da aplicação finalizada e apresentada às assistentes sociais, foi possível identificar que, de fato, um sistema informatizado traria benefícios, todavia, foram identificadas limitações e alterações necessárias para a real utilização do sistema proposto.

Além disso, durante a fase inicial do desenvolvimento, observou-se que o sistema também apresenta potencial para ser usado em outras instituições e, inclusive, por assistentes sociais que não necessariamente pertencem a um Conselho da Comunidade, visto que o pro-

blema de registrar atendimentos é uma demanda comum a outros cenários onde os assistentes sociais atuam. Então, como o desenvolvimento foi feito voltado somente ao Conselho da Comunidade da Comarca de Guarapuava (CCG), é proposto tornar a aplicação mais robusta e flexível para que outros perfis de usuário ou instituições possam também fazer uso.

O protótipo anterior, seguia um modelo de arquitetura *Serverless*[1], fortemente atrelado à plataforma onde ele foi construído, o que causa dificuldades de generalização[2] e restrições em relação à normas de segurança e privacidade. Diante disso, considerou-se necessária uma refatoração da arquitetura do sistema, optando então por tecnologias que permitissem uma gestão transparante do sistema e dos dados gerados por ele.

Propôs-se então uma nova versão do sistema a qual os usuários realizam a personalização de formulários para cadastro de atendimentos, não ficando mais acoplado às demandas específicas dos Conselhos da Comunidade. Além disso, ao contrário da versão inicial, voltada a uma única instituição, a nova versão contém cadastros de organizações, podendo atender várias instituições, fazendo com que as assistentes sociais e demais usuários tenham um único acesso e possam participar dentro de outras entidades. Outro ponto a destacar é que a nova versão possui diferentes tipos de permissões para os usuários, onde cada perfil tem acessos a diferentes recursos da aplicação, tornando-a mais segura e menos suscetível ao erro. Nesta reformulação da arquitetura do sistema, a alternativa explorada foi a construção de uma API[3] que fornece os recursos disponibilizados pela arquitetura *Serverless*.

1.2 Objetivos

Esta seção apresenta os objetivos do trabalho.

1.2.1 Objetivo geral

Desenvolver uma API para o sistema Open Social Care e aplicá-la em substituição à infraestrutura proprietária usada no modelo *Serverless*.

1.2.2 Objetivos específicos

- Avaliação do *feedback* (parecer) obtido com o protótipo anterior;

[1] *Serverless*: Arquitetura sem um servidor próprio, em geral, utilizando infraestrutura em nuvem gerenciada e fornecida por empresas privadas.

[2] Generalização: Ampliar o uso do sistema para diferentes tipos de organizações, maiores detalhes podem ser observados no Capítulo 5 seção 5.2

[3] API (*Application Programming Interface*): conjunto de funções e procedimentos que permitem a integração de sistemas.

- Revisão dos requisitos do software dadas as sugestões apontadas pelas assistentes sociais;
- Planejamento, na forma de diagramas e *mockups* (representação gráfica), das alterações a serem realizadas na nova versão do *software*;
- Elaboração de um modelo para o banco de dados;
- Implementação de uma API *Restful*[4] que substitua os acessos realizados na plataforma proprietária do modelo *Serverless*;
- Implementação de testes para as funcionalidades desenvolvidas;
- Configuração de um servidor de *staging* (servidor de teste) para demonstração e coleta de *feedback* da nova versão do sistema;
- Escrita da documentação do *software*.

1.3 Justificativa

No CCG são realizados atendimentos semanalmente, gerando muitos arquivos e dados para a instituição. Esses atendimentos são realizados em fichas de papel ou ainda por meio digital, neste caso, arquivos de texto armazenados em nuvens como o *Google Drive* ou *One Drive*. Esse método de trabalho pode gerar vários problemas, como o levantamento de dados ou relatórios necessariamente realizado de forma manual, como também o método pode gerar a perda de documentos ou dificuldade em encontrar os arquivos, isso faz com que o trabalho se torne exaustivo, demorado e suscetível ao erro.

A estrutura do Open Social Care foi desenvolvida em um modelo em nuvem que não contemplava o gerenciamento de um servidor. A aplicação inicial foi construída utilizando o *Firebase* [5], considerado como um *Backend*, um modelo de serviço que oferece toda a infraestrutura voltada para o funcionamento interno do *software*, como sistemas, banco de dados, envio e recebimento de informações, armazenamento, entre outros. Isso quer dizer que o profissional não precisará desenvolver todo o sistema de forma manual, uma vez que o *Firebase* oferece esse serviço de forma mais automatizada (REMESSA, 2021).

O problema do protótipo ter sido feito dessa forma, é que fica obrigatório tudo estar centralizado na infraestrutura e nas tecnologias da *Google* (proprietária do *Firebase*), cada instituição que fosse utilizar a aplicação precisaria ter uma conta *Google* e realizar a configuração

[4] API *Restful*: É um padrão de arquitetura de software para requisções HTTP(*Hypertext Transfer Protocol*) e HTTPS(*Hyper Text Transfer Protocol Secure*)

[5] *Firebase*: É uma plataforma que possui uma infraestrutura de *backend* pronta para quem desenvolve aplicativos, possui banco de dados, autenticação e integração para aplicativos de celular. É um serviço fornecido pela empresa *Google*.

e manutenção do sistema por conta própria, gerando responsabilidade sobre o armazenamento e segurança das informações na conta pessoal de um único usuário (quem implementou e configurou o servidor do sistema). Ademais, a personalização das configurações na plataforma *Firebase* é mais limitada se comparada à executar a aplicação com servidor gerenciado por conta própria, onde a instituição pode ter amplos poderes para gerir a forma como os dados são armazenados. Além disso, o *Firebase* é focado em cobrança com base no uso de suas ferramentas, cada recurso tem um limite diário ou mensal de uso com base na quantidade de acessos ao sistema, sendo inviável aplicá-lo em instituições com orçamento mais limitado uma vez que é preciso estimar e prever os custos de infraestrutura antes do consumo dos recursos.

Outro ponto a ressaltar, é que o sistema inicial (protótipo) foi desenvolvido exclusivamente para atender à demanda do CCG, sendo limitado em relação ao uso em outras instituições que também realizam atendimentos sociais, dificultando a expansão do sistema para demais áreas que realizam atendimento social. Logo, uma das melhorias propostas é fazer um *software* que possa ser usado de forma genérica, isso é, a inclusão de cadastro de múltiplas instituições, onde também cada uma poderá realizar personalizações a fim de atender diferentes demandas em seus cadastros de atendimentos, possibilitando o uso do sistema para diferentes tipos de instituições. Afim de atender a demanda de personalizações para que o sistema atenda vários tipos de organizações, foi desenvolvido um mecanismo de modelos de formulários, os questionários personalizados para a realização de atendimentos. Assim, cada organização vai poder criar o seu próprio formulário de atendimento, com os campos de perguntas próprios para cada objetivo e demanda, permitindo diferentes cenários de uso para o sistema, não ficando mais acoplado à necessidade do CCG.

1.4 Estrutura do trabalho

O trabalho está estruturado em cinco seções principais. No Capítulo 1, foi apresentada uma descrição do projeto de forma geral, apresentando também seus problemas e propostas soluções, além de os objetivos do trabalho. Em sequência, no Capítulo 2 são apresentados os documentos e plataformas para gestão de atendimentos sociais, similares ao Open Social Care. No Capítulo 3, é abordado o protótipo Open Social Care, destacando as suas funcionalidades de infraestrutura. Em seguida, no Capítulo 4, são descritos os Materiais e Métodos que foram usados para o desenvolvimento das atividades. Finalmente, no Capítulo 6, são apresentadas as conclusões do trabalho, mostrando os pontos importantes de desafios e soluções.

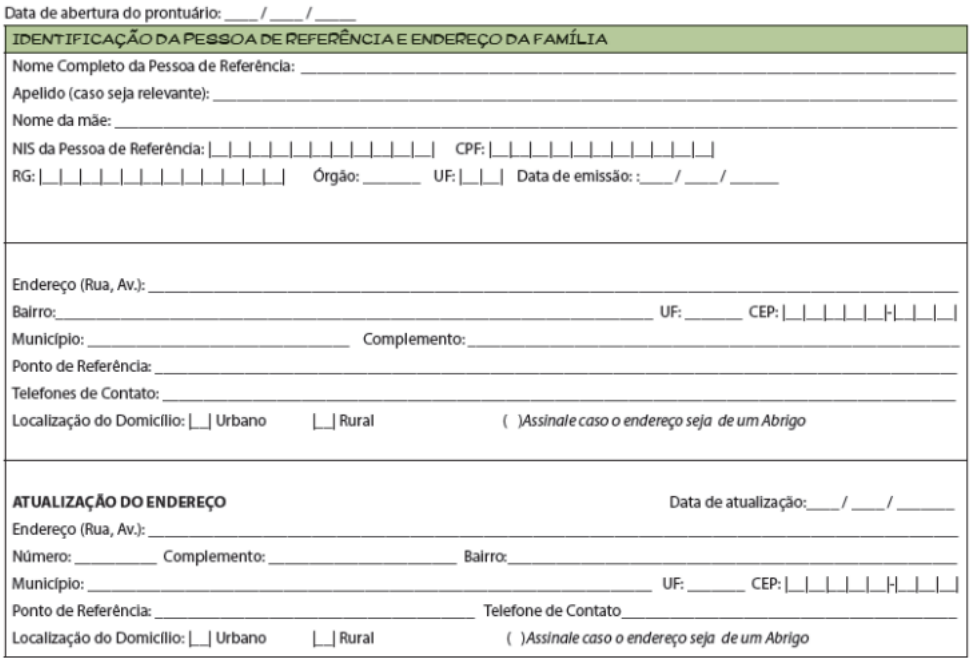
2 TRABALHOS RELACIONADOS

Nesta seção serão apresentados documentos e plataformas existentes para gerenciamento de atendimentos sociais ou que são utilizadas por assistentes sociais para realização de atendimentos.

2.1 Prontuário SUAS

O Prontuário Sistema Único de Assistência Social (SUAS), é um documento utilizado em todo o Brasil para armazenar informações relacionadas à assistência social prestada a indivíduos e famílias em situação de vulnerabilidade ou risco social. Na Figura 1 é possível visualizar uma versão do formulário. O Prontuário Eletrônico é uma ferramenta que auxilia o trabalho dos profissionais dos Centro de Referência de Assistência Social (CRAS), Centro de Referência Especializado de Assistência Social (CREAS) e Unidades de Acolhimento para Crianças e Adolescentes no registro dos atendimentos realizados às famílias e indivíduos, e que permite qualificar o atendimento social e analisar de forma sistematizada as informações sobre o território e a população atendida. Sua utilização permite manter um histórico dos atendimentos, agilizando assim o trabalho dos profissionais e facilitando a vida dos usuários do SUAS (SUAS, 2023).

Figura 1 – Documento Prontuário SUAS.



Data de abertura do prontuário: ____/ ____/ _____

IDENTIFICAÇÃO DA PESSOA DE REFERÊNCIA E ENDEREÇO DA FAMÍLIA

Nome Completo da Pessoa de Referência: _____

Apelido (caso seja relevante): _____

Nome da mãe: _____

NIS da Pessoa de Referência: |__|__|__|__|__|__|__|__|__|__|__| CPF: |__|__|__|__|__|__|__|__|__|__|__|

RG: |__|__|__|__|__|__|__|__|__|__|__| Órgão: _____ UF: |__|__| Data de emissão: :____/ ____/ _____

Endereço (Rua, Av.): _____

Bairro: _____ UF: _____ CEP: |__|__|__|__|-|__|__|__|

Município: _____ Complemento: _____

Ponto de Referência: _____

Telefones de Contato: _____

Localização do Domicílio: |__| Urbano |__| Rural ( )*Assinale caso o endereço seja de um Abrigo*

ATUALIZAÇÃO DO ENDEREÇO Data de atualização:____/ ____/ _____

Endereço (Rua, Av.): _____

Número: _____ Complemento: _____ Bairro: _____

Município: _____ UF: _____ CEP: |__|__|__|__|-|__|__|__|

Ponto de Referência: _____ Telefone de Contato _____

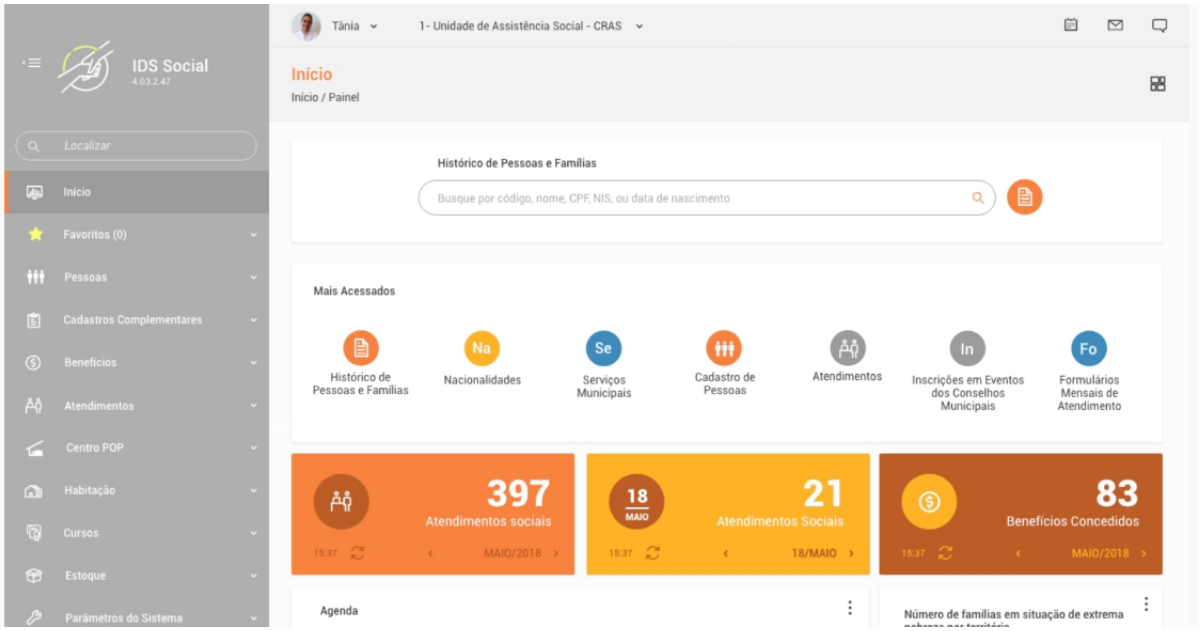
Localização do Domicílio: |__| Urbano |__| Rural ( )*Assinale caso o endereço seja de um Abrigo*

Fonte: (MINISTÉRIO DO DESENVOLVIMENTO SOCIAL E COMBATE À FOME, 2013).

2.2 IDS Social

O IDS Social é um *software* voltado a realização de atendimentos sociais. Ele oferece Cadastros unificados, Atendimento adequado ao Prontuário SUAS online, Informações de Atendimento e Acompanhamento Georreferenciadas, entre outras coisas. Com o IDS Social todas as necessidades e acompanhamentos da área social municipal são administradas com qualidade, segurança e centralidade de dados. O IDS Social garante integridade da base de cadastro entre setores, unidades de Assistência Social do município, CRAS e CREAS, disponibilizando um histórico único e detalhado para acompanhamento e efetividade dos planos de ações das áreas de Assistência Social municipais (IDS SOCIAL, 2023). Na Figura 2 é possível ver uma tela inicial do sistema, mostrando gráficos e algumas funcionalidades disponíveis no sistema.

Figura 2 – Tela inicial IDS Social



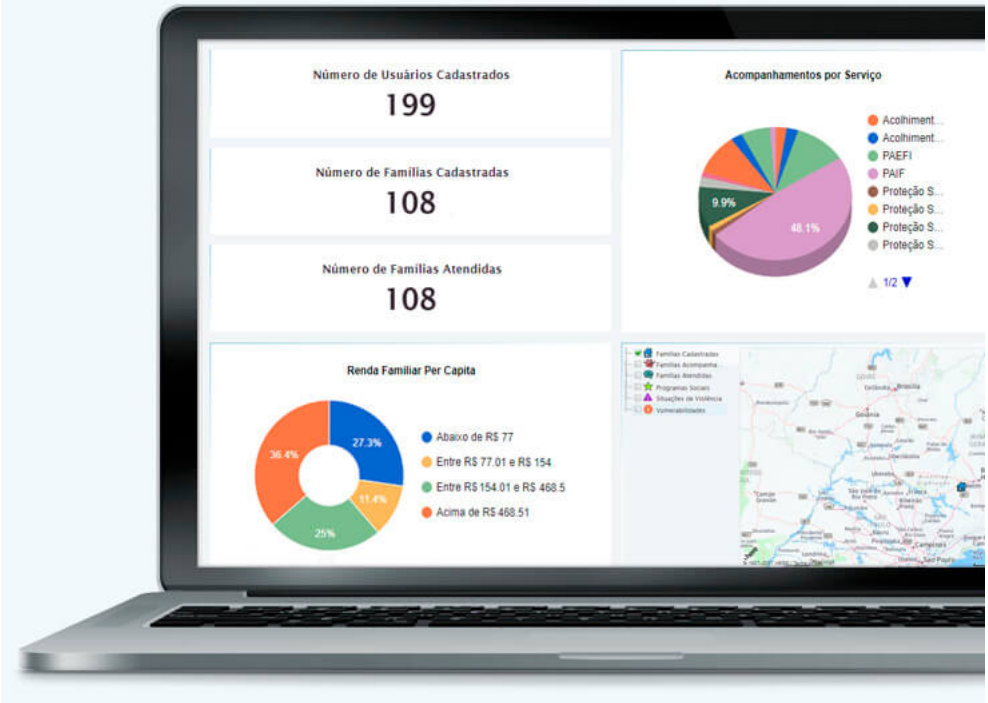
Fonte: (IDS SISTEMAS - BRASIL GESTÃO PÚBLICA, 2018).

2.3 GESUAS

O Gesuas ou Gestão do Sistema Único de Assistência Social, é um sistema para gestão e acompanhamento das ações e políticas de assistência social do SUAS. O Gesuas é a primeira versão online do prontuário SUAS. Com ele não há limite de prontuários e a impressão dos mesmos ficará a critério e necessidade dos técnicos. É o mesmo prontuário físico disponibilizado pelo Ministério do Desenvolvimento Social, só que online. O Gesuas auxilia os gestores na efetiva implementação da vigilância, gerando informações de forma territorializada (georreferenciada) e exibindo indicadores para gestão. O painel do gestor, mostrado na Figura 3, permite monitoramento em tempo real das atividades e ações realizadas nos equipamentos. Os relatórios para monitoramento, como o Registro Mensal de Atendimento (RMA) são gerados em

segundos. O Gesuas auxilia os gestores na efetiva implementação da vigilância, gerando informações de forma territorializada (georreferenciada) e exibindo indicadores para gestão (GESUAS, 2023).

Figura 3 – Tela de monitoramento em tempo real GESUAS



Fonte: (GESUAS, 2023).

2.4 SociÁgil

O SociÁgil é o sistema para Assistência Social usar para a gestão do SUAS, desenvolvido com base no cotidiano dos trabalhadores e de acordo com as normas e orientações técnicas do SUAS, integrando: Secretaria Municipal de Assistência Social (SMAS), CRAS, CREAS, Conselho Tutelar, Organizações da Sociedade Civil e demais políticas públicas (SOCIÁGIL, 2024). O sistema possui gestão financeira através de gráficos para controle e monitoramento, integração com sistema de prefeituras para licenciamentos e também gerenciamento de documentos, ele é uma plataforma voltada a gerencia e monitoramento do que cadastros de atendimentos.

Figura 4 – Tela de gestão financeira do sistema SociÁgil (Dashboard)



Fonte: (SOCIÁGIL, 2024).

2.5 Comparativo

Apresentados os documentos e as tecnologias similares ao Open Social Care, é destacado que algumas dessas ferramentas são de utilização paga, sendo inviável o uso por muitas instituições de serviços sociais ou até mesmo por assistentes sociais, pois a maioria dos trabalhos sociais realizados é voluntário, como no caso do CCG. Logo, o Open Social Care seria livre para uso, contribuindo para a área de atendimento social. Outro ponto é que as ferramentas citadas podem ter um uso muito específico no cadastro de atendimentos e como uma das propostas do sistema é a generalização, a aplicação proposta neste trabalho será mais abrangente. Além disso, vale destacar que o Prontuário SUAS, embora forneça recursos para o cadastro de pessoas, não atende as especificidades dos atendimentos das assistentes sociais com relação aos PPLs, característica necessária para o uso no CCG.

3 O OPEN SOCIAL CARE

O Open Social Care é um sistema para gerenciamento de atendimentos sociais, ele foi criado para auxiliar o dia a dia de trabalho de assistentes sociais do CCG. O sistema surgiu como um projeto de disciplina de Sistemas Distribuídos da UTFPR, câmpus Guarapuava, quando uma assistente social do CCG entrou em contato com o professor da disciplina e sugeriu a ideia de informatização do processo até então utilizado no CCG. Através de reuniões e discussões, foi feita a coleta dos requisitos e planejado como o sistema teria que funcionar. Por fim, foi desenvolvido um protótipo como um objeto de estudos na disciplina.

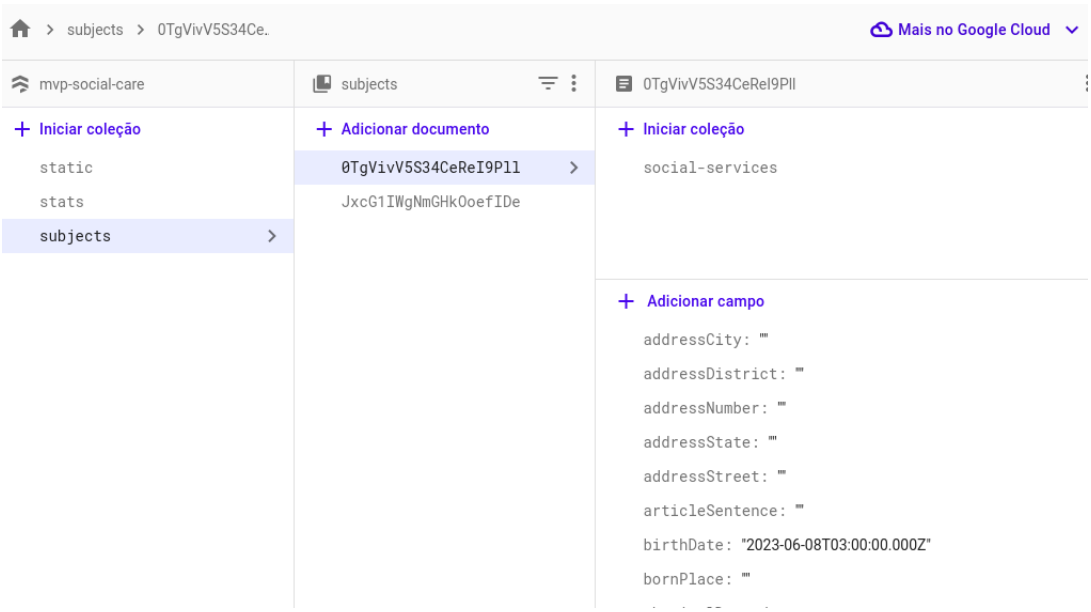
Dada a característica e os tópicos da disciplina, optou-se pelo estudo e experimentação da abordagem *Serverless* de modo que cada aluno pudesse configurar a infraestrutura e trabalhar na aplicação sem precisar de um servidor *Backend*. Na ocasião, as principais tecnologias utilizadas foram *Firebase* e *React.JS*. Com o *Firebase*, foi possível fazer uso de serviços prontos, como o banco de dados, autenticação de usuários e a hospedagem de arquivos e páginas. Conforme ilustrado na Figura 5, utilizou-se o *Firestore Database*, que é um banco de dados *NoSQL*. Nele é possível armazenar dados em documentos que contêm mapeamentos de campos para valores. Esses documentos são armazenados em coleções, que são contêineres de documentos que podem ser usados para organizar dados e criar consultas. Os documentos suportam muitos tipos de dados diferentes, desde *strings*[1] e números simples a objetos complexos e aninhados. Também é possível criar subcoleções dentro dos documentos e criar estruturas de dados hierárquicas que podem ser escalonadas à medida que o banco de dados cresce (FIREBASE, 2023a).

Para autenticação foi usado o serviço *Authentication* que fornece serviços de *Backend*, *Software Development Kit* (SDK) e bibliotecas de Interface do usuário (IU) prontas para autenticar usuários no seu aplicativo. Ele oferece suporte à autenticação usando senhas, números de telefone, provedores de identidade federados conhecidos, como *Google*, *Facebook* e *Twitter*, entre outros (FIREBASE, 2023c). Além disso, foi usado para salvar imagens e arquivos o *Firebase Storage*, um serviço de armazenamento de objetos criado para a escala do *Google*. Com os SDKs do *Firebase* para *Cloud Storage*, é possível usar a segurança do *Google* para fazer *upload* e *download* de arquivos nos aplicativos do *Firebase*, independentemente da qualidade da rede (FIREBASE, 2023b).

De modo geral, o experimento com a arquitetura *Serverless* foi bastante válido no contexto da disciplina. Entretanto, conforme já mencionado, o modelo sofre limitações em relação ao seu uso prático em cenário real. Ademais, a generalização do sistema, que irá possibilitar a utilização dele para demais instituições além do CCG, e a possibilidade de hospedá-lo em um servidor próprio, sem a dependência de tecnologias de terceiros, aumenta a contribuição e as possibilidades de aplicação do projeto.

[1] Sequências de caracteres alfanuméricos (letras, números e/ou símbolos)

Figura 5 – Tela de gerenciamento de banco de dados do Open Social Care no Firebase.



Fonte: Autoria própria (2024).

4 MATERIAIS E MÉTODOS

A ênfase deste capítulo está em descrever as principais ferramentas utilizadas no projeto e o processo de organização utilizado. Este capítulo está subdividido em duas seções, 4.1 Materiais e 4.2 Métodos.

4.1 Materiais

Nesta seção são apresentadas as principais ferramentas e tecnologias utilizadas para o desenvolvimento da API do sistema.

4.1.1 Git e Github

O *Git* é um sistema de controle de versionamento de código, possibilitando que possua diferentes versões e funcionalidades sendo desenvolvidas simultaneamente e independentes entre si (GIT, 2023) ao permitir criar várias instâncias do código, um sistema de controle de versionamento ajuda no trabalho em equipe, pois permite que cada pessoa possa trabalhar a sua parte individualmente.

Além disso, o *Github* permite rastrear e gerenciar alterações no código, como ao encontrar algum *bug* (erro), com o *Git* é possível achar o *commit* (envio no Git) que levou ao *bug*, analisar as mudanças e fazer a correção, sendo muito mais prático e rápido.

Já o *Github* é uma aplicação web que possibilita a hospedagem de repositórios *Git* (AQUILES, 2014). Para o desenvolvimento é bastante importante ter o código hospedado em um repositório, pois assim facilita o compartilhamento, o controle de versionamento, como mantém o projeto seguro e salvo. O *Git* e o *Github* trabalham em conjunto para auxiliar no desenvolvimento de um projeto, são duas ferramentas bastante utilizada por desenvolvedores.

4.1.2 PHP e Laravel

O *Hypertext Preprocessor* (PHP) é uma linguagem de código aberto de uso geral, muito utilizada, e especialmente adequada para o desenvolvimento web e que pode ser embutida dentro do *HyperText Markup Language* (HTML) (PHP, 2023). Devido ao fácil aprendizado e desenvolvimento, a linguagem PHP facilita o caminho do profissional que escolhe estruturar sites utilizando a plataforma. Dessa maneira, as edições e configurações são feitas de uma maneira muito mais simplificada [...] de forma prática, é uma linguagem sem custos e com melhorias constantes, pois qualquer programador pode melhorar seu código, retirando possíveis erros ou adicionando mais funções (CONTEIGE, 2023).

O *Laravel* é um *framework* feito para o PHP, [...] um *framework* compreende um conjunto de classes ou funções implementadas em uma linguagem de programação especifica usadas para auxiliar o desenvolvimento de um *software* (GABARDO, 2017). A utilização da linguagem de programação PHP para desenvolvimento de aplicações *Web* é facilitada com o uso do *Laravel*, uma vez que diversos recursos são fornecidos já implementados. Além disso, a adoção do *framework* contribui para a organização do projeto uma vez que padrões e nomenclaturas precisam ser adotados e padronizados. Outro benefício do uso do *framework* é a garantia da segurança da aplicação, visto que procedimentos de autenticação já são fornecidos e testados pela comunidade de desenvolvedores.

4.1.3 Docker

O Docker é um software de código aberto usado para implantar aplicativos dentro de *containers* (ambiente isolado) virtuais. A conteinerização permite que vários aplicativos funcionem em diferentes ambientes complexos (DANIELA C., 2023). Ao permitir criar e gerenciar contêineres é possível configurar todas as dependências da aplicação, o que ajuda a manter um padrão e facilita a implementação em diferentes ambientes, contribuindo para o trabalho em equipe e implementação do sistema em diversos servidores.

4.1.4 PostgreSQL

PostgreSQL é um sistema de banco de dados com código aberto, altamente estável que fornece suporte a diferentes funções de *Structured Query Language* (SQL), como chaves estrangeiras, sub consultas, *triggers* (eventos), e diferentes tipos e funções definidas pelo usuário. Ele aumenta ainda mais a linguagem SQL oferecendo várias características que escalam e reservam cargas de trabalho de dados. Esta tecnologia é usada principalmente para armazenar dados para muitos aplicativos móveis, web, geoespaciais e analíticas (KINSTA, 2023).

O PostgreSQL utiliza-se de transações Atomicidade, Consistência, Isolamento, Durabilidade (ACID), as transações ACID maximizam a confiabilidade e a integridade dos dados. Os dados permanecem consistentes mesmo se a operação for concluída apenas parcialmente. Por exemplo, se cair a energia inesperadamente durante a gravação em uma tabela em um banco de dados, sem transações ACID, pode ser que apenas parte dos dados seja salva, e o restante, não. O resultado é um banco de dados inconsistente, tornando a recuperação difícil e demorada (DATABRICKS, 2023). Com isso, a utilização do PostgreSQL contribui para a segurança dos dados, criando uma aplicação confiável e estável.

4.1.5 CapRover

O *CapRover* é um gerenciador de servidores gratuito, escalável e flexível, sendo aplicado como uma plataforma base para a gestão de servidores World Wide Web (WEB) e implantação (*Deploy*) de sistemas. Ele simplifica o processo de *deploy* e administração do sistema. O *CapRover* faz uso do *Docker* separando a aplicação em *containers* que podem ser gerenciados através da interface WEB da ferramenta. Ele facilita a configurações e monitoramento do sistema uma vez que possui integração com os repositórios GitHub e permite a automação de *deploys* em fluxos de entrega contínua (do inglês, *Continuous Delivery* - CD).

A ideia da ferramenta *CapRover* é que os desenvolvedores foquem na programação do *software* e não precisem se preocupar com a infraestrutura de servidor. Neste projeto o *CapRover* automatiza o processo de *deploy* da aplicação, realizando a atualização do servidor conforme as alterações submetidas no repositório *git* do projeto.

4.2 Métodos

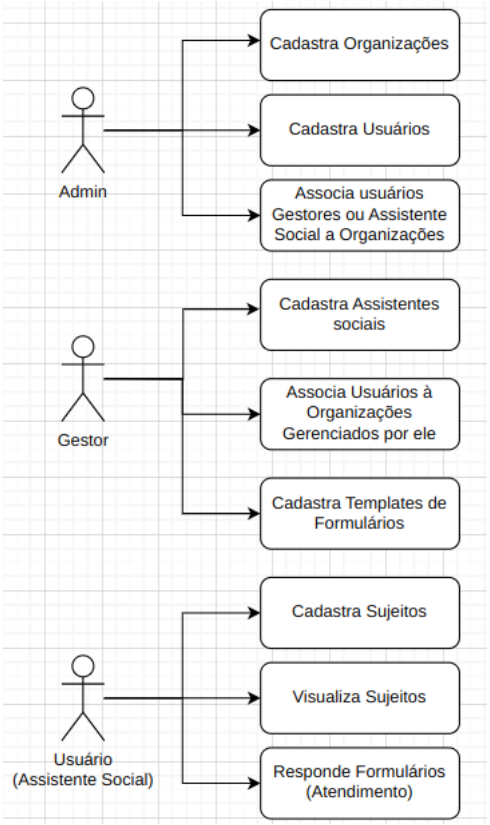
Nesta seção são apresentados os métodos de planejamento para o desenvolvimento do projeto.

4.2.1 Análise e coleta de requisitos

Para a coleta de requisitos do sistema, realizou-se uma apresentação do protótipo do sistema às assistentes sociais do CCG, por meio de reuniões, em que foram discutidas suas funcionalidades e melhorias. Os requisitos foram então separados com base em requisitos funcionais, técnico e operacional utilizando a metodologia *MoSCoW*(*Must-Have, Should-Have, Could-Have e Won't-Have*). O método *MoSCoW* é uma técnica de priorização usada na gestão de projetos e desenvolvimento de softwares com o intuito de encontrar um entendimento em comum entre as partes interessadas sobre a importância que elas atribuem a cada requisito (PIRES, 2019). Ele se baseia em etapas, o que o sistema precisa ter, deveria ter, poderia ter e não teria.

Além disso, fluxos de casos de uso foram pensados para melhor entender as funcionalidades que o sistema deveria ter a fim de validar a utilização e o comportamento da aplicação com base nos diferentes perfis de usuário. Um exemplo de diagrama de casos de uso é ilustrado na Figura 6. O caso de uso descreve o comportamento do sistema sob diversas condições conforme o sistema responde a uma requisição de um chamado ator primário. O ator primário inicia uma interação com o sistema para atingir um objetivo. O sistema responde, protegendo o interesse de todos (COCKBURN, 2005).

Figura 6 – Diagrama de casos de uso para os perfis do sistema

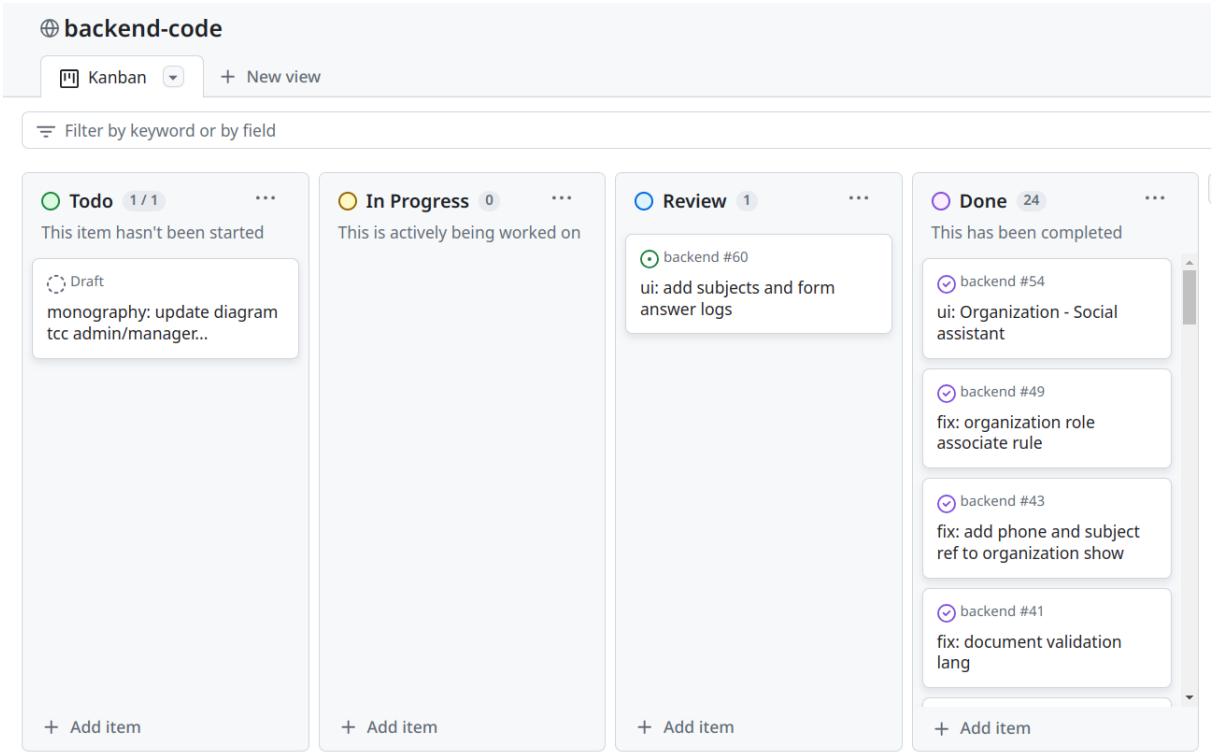


Fonte: Autoria própria (2024).

4.2.2 Organização de tarefas para o desenvolvimento

Para o desenvolvimento das atividades, é feita a utilização da metodologia Kanban. A metodologia Kanban pode ser resumida em uma maneira de organizar os processos que envolvem as equipes de uma organização, ressaltando a priorização das tarefas e tornando o foco bem definido para todos. Dessa maneira, é possível identificar e resolver problemas no fluxo de trabalho (AMORIM, 2023). Na separação das atividades, foi utilizado o recurso *projects* do *Github*, onde é possível separar as atividades em *cards* e organizar com base na sua etapa, *To do* (para fazer), *In Progress* (Em progresso), *Review* (Revisão) e *Done* (Feito), conforme expresso na Figura 7.

Figura 7 – Quadro kanban das atividades no github



Fonte: Autoria própria (2024).

4.2.3 Planejamento do banco de dados

Devido às características do problema, optou-se pelo uso de um modelo relacional de banco de dados. Um banco de dados armazena e fornece acesso a pontos de dados relacionados entre si. O modelo relacional é uma maneira intuitiva e direta de representar dados em tabelas. Em um banco de dados relacional, cada linha na tabela é um registro com uma ID exclusiva chamada chave. As colunas da tabela contêm atributos dos dados e cada registro geralmente tem um valor para cada atributo, facilitando o estabelecimento das relações entre os pontos de dados (ORACLE, 2023).

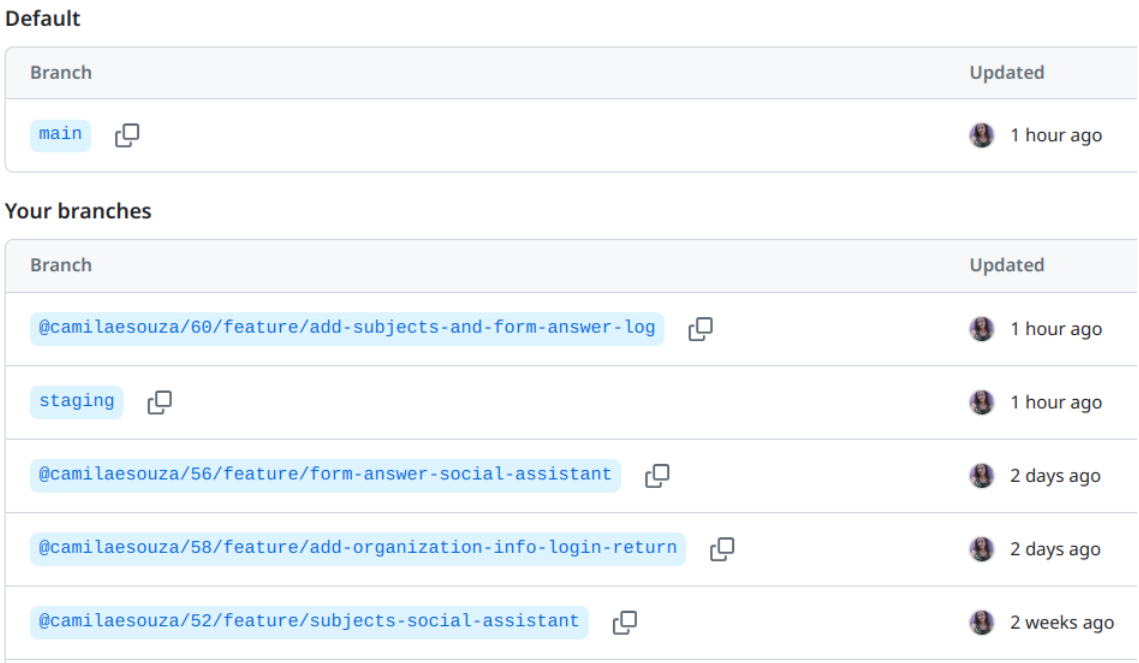
Com isso, foi realizada a modelagem do banco de dados utilizando a abordagem de modelo Entidade relacionamento (ER), diagrama ER é um tipo de fluxograma que ilustra “entidades” como pessoas, objetos ou conceitos que se relacionam entre si dentro de um sistema. Diagramas ER são mais utilizados para projetar ou depurar bancos de dados relacionais nas áreas de engenharia de software, sistemas de informações empresariais, educação e pesquisa. Também conhecidos como DERs, ou modelos ER, usam um conjunto definido de símbolos, tais como retângulos, diamantes, ovais e linhas de conexão para representar a interconectividade de entidades, relacionamentos e seus atributos. Eles espelham estruturas gramaticais, onde entidades são substantivos e relacionamentos são verbos (LUCID, 2023).

4.2.4 Fluxo de desenvolvimento

Para a implementação do sistema, foi seguido um fluxo de desenvolvimento para garantir um código estruturado e boa execução do projeto. O projeto foi estruturado usando as ferramentas do *GitHub*, o código fonte foi armazenado em um repositório, onde foi divido em duas *branchs* (Ramificação do código fonte), *main* e *staging*. A *branch main* armazena o código principal e funcional, já a *branch staging* é uma ramificação que contém uma versão estável do sistema, servindo especificamente para o servidor *staging* da aplicação, logo não são feitas alterações diretamente nela. Como primeira etapa, cita-se a elaboração de tarefas referente ao sistema, a organização dessas tarefas está exemplificado na subseção 4.2.2. Após a criação de tarefas, foi feita a priorização e escolha da ordem de tarefas para iniciar a implementação do sistema. Essa escolha de prioridade foi feita pensando na base das determinadas tarefas, sobre o que era necessário para a sua execução, por exemplo, para cadastrar uma organização no sistema, é necessário um usuário estar logado, logo a tarefa para implementar a autenticação no sistema deveria vir primeiro. A partir dessa organização, foram executados os seguintes passos.

1. Criação de *branchs*: Para cada tarefa era criado uma *branch* e nela desenvolvido parte do sistema relacionado a ela. Na Figura 8 é possível ver algumas das *branchs* criadas para o desenvolvimento, o nome foi dado seguindo um padrão, @ e então o nome do desenvolvedor, o número da tarefa, se a tarefa era uma nova funcionalidade (*feature*) ou uma correção (*fix*) e o nome da tarefa.

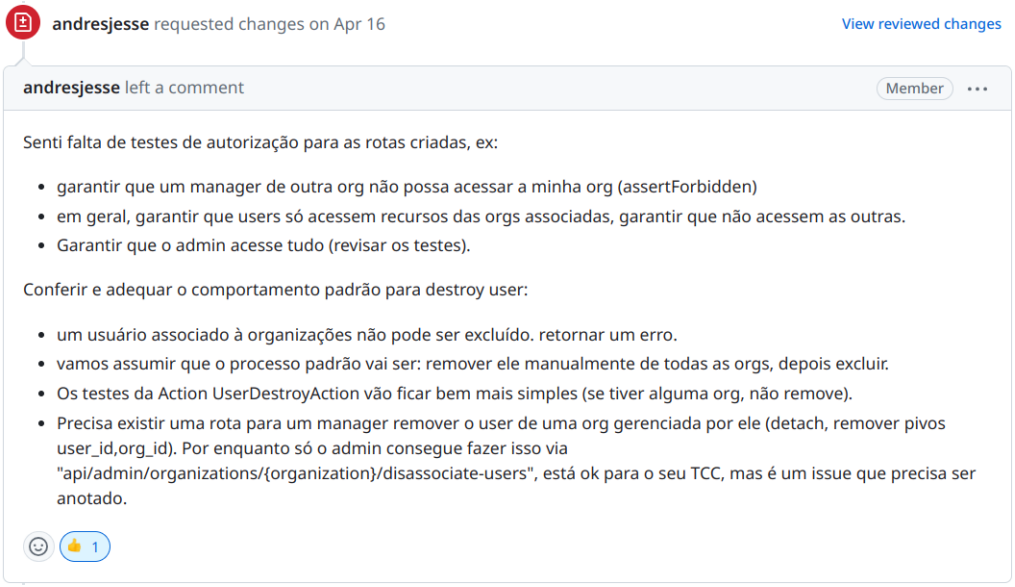
Figura 8 – GitHub branchs



Fonte: Autoria própria (2024).

2. *Pull requests* (Solicitação): Após desenvolver uma tarefa era criada uma *pull request*, uma solicitação para que essa implementação pudesse ser mesclada (*merge*) à *branch main*. Nessa etapa uma pessoa responsável por revisar, realizava avaliações sobre o código e funcionamento, em caso de inconsistência eram adicionadas observações e requisitadas mudanças para a correção. Na Figura 9 é exemplificada uma revisão e solicitação de mudanças.

Figura 9 – Comentário de uma revisão



Fonte: Autoria própria (2024).

3. *Merge e Deploy* (subir a tarefa/*branch* para a *main*): Com uma *Pull request* aprovada, o *merge* era realizado para a *branch main* e com isso realizado o *deploy* para o servidor *staging*. A Figura 10 apresenta o *log* ao realizar um *deploy* utilizando a plataforma CapRover.

Figura 10 – *Build log do deploy* CapRover



Fonte: Autoria própria (2024).

5 RESULTADOS

Os dados apresentados nessa seção demonstram o impacto do projeto e os avanços alcançados com ele.

5.1 Requisitos

O Open Social Care é um projeto que foi apresentado para as assistentes sociais do CCG, o protótipo inicial teve uma breve fase de testes, e no decorrer da utilização do sistema, foram reveladas dificuldades ao realizar alguns passos, além das preocupações técnicas, como permissão dos usuários, acessos ao sistema e a infraestrutura. Portanto, através disso foram feitas reuniões para validar os requisitos e coletar funcionalidades que o sistema poderia suprir. Além disso, foram feitas reuniões mais técnicas para validar a estrutura do sistema para chegar num melhor resultado.

Durante essa fase de validação do protótipo, como dito anteriormente, foi verificada a necessidade de mudar sua estrutura inicial, migrando de uma arquitetura sem servidor para uma API, logo com essa remodelagem foi preciso fazer uma nova análise com base nos requisitos já coletados para o protótipo e também melhorias para o sistema. Na Tabela 1, é possível ver os requisitos coletados nessa etapa inicial. Esses requisitos foram classificados pelo método MoSCoW, com as prioridades definidas a partir do protótipo anterior.

Requisito	Tipo (MoSCoW)	Descrição
01	M	Possuir tipos de perfis de acesso e permissões diferentes para cada (admin, gestor e assistente social)
02	M	Possuir cadastro, edição, listagem e visualização de sujeitos
03	M	Possuir cadastro, edição, listagem e visualização de atendimentos com base em um sujeito
04	S	Possuir cadastro, edição, listagem e visualização de organizações
05	M	Possuir cadastro, edição, visualização de questionários personalizados para atendimentos
06	M	Possuir cadastro, edição, listagem e visualização de usuários
07	M	Possuir login por usuário
08	C	Possuir servidor *staging*
09	C	Possuir upload de arquivos e imagens no cadastro de atendimentos
10	W	Possuir recursos para geração de relatórios

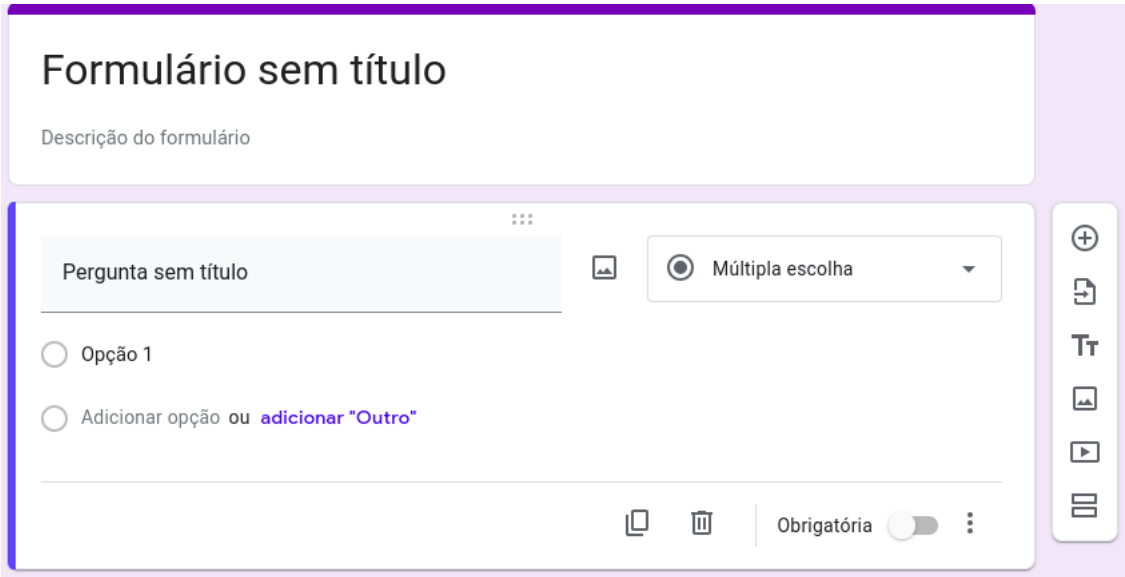
Tabela 1 – Listagem de requisitos básicos do sistema

5.2 Generalização do sistema

O protótipo do Open Social Care foi desenvolvido visando apenas as necessidades do CCG, tornando o sistema fortemente engessado às demandas e informações que são convenientes para eles. Durante a apresentação do sistema e reuniões para discutir o uso e validação, foi visto um potencial para que a aplicação pudesse ser utilizada por mais instituições além do CCG, com isso surgiu a ideia de ter um sistemas mais flexível e generalizado para uso de instituições de forma geral.

Para a construção de uma aplicação que possa ser utilizada por mais instituições, realizou-se o desenvolvimento de cadastro de múltiplas organizações, pois o protótipo inicial do Open Social Care só pode ser utilizado por uma única organização. Ao ter um cadastro para isso, o sistema pode ser utilizado por várias instituições de uma única vez, onde cada instituição tem os seus sujeitos e atendimentos separados, logo esse foi o primeiro passo para a generalização do sistema. O segundo passo foi a criação de questionários dinâmicos, sendo possível criar vários modelos de formulários para cadastros de atendimentos, não deixando mais o sistema com campos fixos que serviriam apenas uma instituição. Um exemplo de questionários dinâmicos seria o cadastro de formulário da empresa *Google*, onde as perguntas são personalizadas, desde opções e tipos de perguntas podendo ser descritiva ou de escolha, como mostrado na Figura 11.

Figura 11 – Tela para cadastro de um formulário *Google*



Fonte: Autoria própria (2024).

Para essa nova fase do sistema, modificações no banco de dados foram feitas, para que pudesse atender diversas necessidades. Durante o desenvolvimento surgiu o interesse de outra instituição (um Albergue da cidade de Guarapuava) em utilizar a plataforma. Com base nas informações dessa nova organização para realizar os atendimentos, pensou-se em melhorias nas tabelas de dados, afim de aprimorar a generalização, tornando o sistema mais flexível e adaptável a novas necessidades.

A generalização do sistema é uma solução tecnológica para atender à variedade de necessidades institucionais. Tal flexibilidade visa gerar economia de tempo, pois diferentes instituições podem fazer uso de um mesmo sistema para diferentes tipos de atendimentos, auxiliando assim na padronização de processos, visto que ao criar um questionário ele pode ser usado por mais instituições, o que permite a troca de informações e boas práticas entre elas. Ao construir um sistema que possa servir de forma genérica as organizações, é construído um sistema com possibilidade de crescimento e adaptação a mudanças de normas e regras, sendo

uma aplicação independente para atender vários objetivos. A generalização desempenha um papel fundamental para o crescimento e abrangência do Open Social Care. A seção 5.5 detalha aspectos técnicos de como é tratada essa generalização no banco de dados.

5.3 Segurança

Segurança é uma das características principais em qualquer sistema de informações, ela garante a proteção dos dados, a prevenção de ameaças e o gerenciamento de riscos. Garantir a segurança é fundamental para preservar a integridade, confidencialidade e disponibilidade das informações. No caso do Open Social Care, que pode possuir dados sigilosos, uma das maiores preocupações é sobre essa segurança.

O primeiro passo para garantir a segurança foi a implementação de *tokens* (senhas) para autenticação, sendo mais comum o nome de *login* (acesso ao sistema), isso valida a identidade de usuários. O *login* funciona como uma chave. Com ele, somente pessoas com as credenciais em mãos podem acessar computadores, desbloquear celulares, entrar em uma conta de rede social, entre outros casos. Dessa forma, o método confere mais segurança aos usuários (BLASI, 2022). Para a aplicação Open Social Care é usada a emissão de *tokens* do *Sanctum*, "*que é um pacote de autenticação API para o Laravel [...] Sanctum permite que você emita tokens de API/tokens de acesso pessoal que podem ser usados para autenticar solicitações de API para seu aplicativo*" (LARAVEL, 2023b). Para garantir a segurança dos dados, os *tokens* são criptografados usando *hash SHA-256* (função de resumo criptográfico que converte dados em uma sequência alfanumérica fixa de 256 *bits*). A cada requisição, os *tokens* são verificados por *Middlewares*, que funcionam como filtros para validação de acesso dos usuários, usados para verificar se uma solicitação recebida é autenticada com um *token* ao qual foi concedida uma determinada habilidade (LARAVEL, 2023b). Então se o usuário tiver um *token* inválido ou não possuir permissões para realizar ações dentro do sistema, ele é bloqueado.

Para evitar o vazamento de informações, foi implementada a geração de *logs* (histórico) de acesso. Assim, quando um usuário realizar determinada ação, como por exemplo, o acesso a uma área do sistema ou cadastros, essa informação será salva. Desta forma, é possível ter um controle do que está acontecendo no sistema e caso tenha algum vazamento de informação ser possível identificar o usuário responsável. Essa funcionalidade foi implementada criando uma tabela no banco de dados, contendo as informações do que foi acessado (model_type e model_id), qual usuário acessou (user_id), contexto do evento (event_context), podendo ser salvo a página acessada ou alguma informação extra para a ação, nome do evento (event_type) para o que foi realizado, por exemplo uma criação, uma coluna *json* com mais informações (*data*) e por último endereço *ip* (event_type) salvando a informação do local que foi realizado o evento. Quando um usuário realizar uma ação no sistema, como por exemplo, visualização de um sujeito, essa informação é salva na tabela de *logs*, contendo as informações, sendo possível então ter um controle de gestão do sistema. *Logs* são registros em forma de arquivos de texto puro em

linhas, onde essas linhas contém informações relativas a data e hora em que ações importantes ocorreram na aplicação. "*Ou seja, o que e quando ocorreu determinado evento[...] Com o uso de logs fica mais fácil obter a visibilidade da integridade da infraestrutura de TI (Tecnologia da Informação). As informações contidas nestes registros podem variar desde o desempenho do sistema, dados sobre auditoria, alertas de intrusão ou até transações e atividades de usuários*" (ROCHA, 2021).

Outro ponto a ser destacado sobre segurança, é a própria utilização dos recursos do *framework Laravel*, ele possui muitos recursos que tornam a aplicação segura e facilitam o desenvolvimento. Para autenticação ele utiliza *providers* (provedores) e *guards* (guardas) para facilitar o processo de autenticação. O objetivo dos *guards* é autenticar os usuários para cada solicitação que eles fazem, enquanto os *providers* facilitam a recuperação dos usuários do banco de dados (POPOVSKI, 2019). Um ataque comum também pode acontecer ao banco de dados e para isso o *Laravel* também tem proteções contra injeções no banco de dados, pois a ferramenta *Eloquent ORM (Object-Relational Mapping ou Mapeamento objeto-relacional)* usa ligação de parâmetros *PDO* (PHP Data Objects ou Objetos de dados PHP) para evitar injeção de *SQL*. A vinculação de parâmetros garante que usuários mal-intencionados não possam transmitir dados de consulta que possam modificar a intenção da consulta (POPOVSKI, 2019). O *Laravel* também possui recursos de validação e permissões para acesso às rotas e criação de dados, entre outros recursos que garantem uma aplicação segura.

A reformulação do sistema como uma API também contemplou criação de permissões e perfis de acesso, onde cada perfil pode acessar diferentes áreas do sistema enquanto outras ficam oculta para ele. Os perfis de acesso implementados foram: administrador(a), gestor(a) e assistente social. O perfil de acesso principal é o administrador(a), que tem acesso a principais partes do sistema, como cadastro de usuários, organizações e manteria um controle sobre a aplicação. O perfil gestor(a) tem acesso a cadastro de assistentes sociais, sujeitos e atendimentos. O perfil assistente social é o perfil com menos poder no sistema, tendo acesso apenas a cadastro de sujeitos e atendimentos. Essa diferença de acesso garante uma organização no sistema e segurança também, devido a ter regras específicas de acesso para cada perfil.

5.4 Escopo do sistema

O objetivo principal do sistema Open Social Care é fazer a gestão de atendimentos sociais, na aplicação é possível cadastrar sujeitos, com dados básicos, como por exemplo, nome e data de nascimento e para esses sujeitos cadastrar atendimentos, preenchendo dados abrangentes, como por exemplo, dados sobre familiares e pedidos de itens ou documentação.

A aplicação oferece níveis de acesso, administrador, gestor(a) e assistente social, onde cada perfil tem acessos e permissões diferentes no sistema. Com isso o sistema oferece o cadastro de organizações, e dentro dessas organizações é possível adicionar usuários que são responsáveis por cadastrar os sujeitos e os atendimentos específicos para a organização

que o usuário está alocado. Exemplo de uso: o administrador cria uma organização X, que fica responsável pela gestora Y, e então a gestora Y cria uma assistente social para sua organização, e essa assistente social fica encarregada de criar sujeitos e realizar atendimentos.

Outro detalhe importante, é que o sistema possui a criação de questionários personalizados para os formulários de atendimentos, para que diversas instituições possam fazer uso do sistema. Questionários personalizados podem ser criados pelos gerentes de uma determinada organização, onde a assistente social vai fazer uso ao cadastrar atendimentos.

5.5 Modelagem do sistema

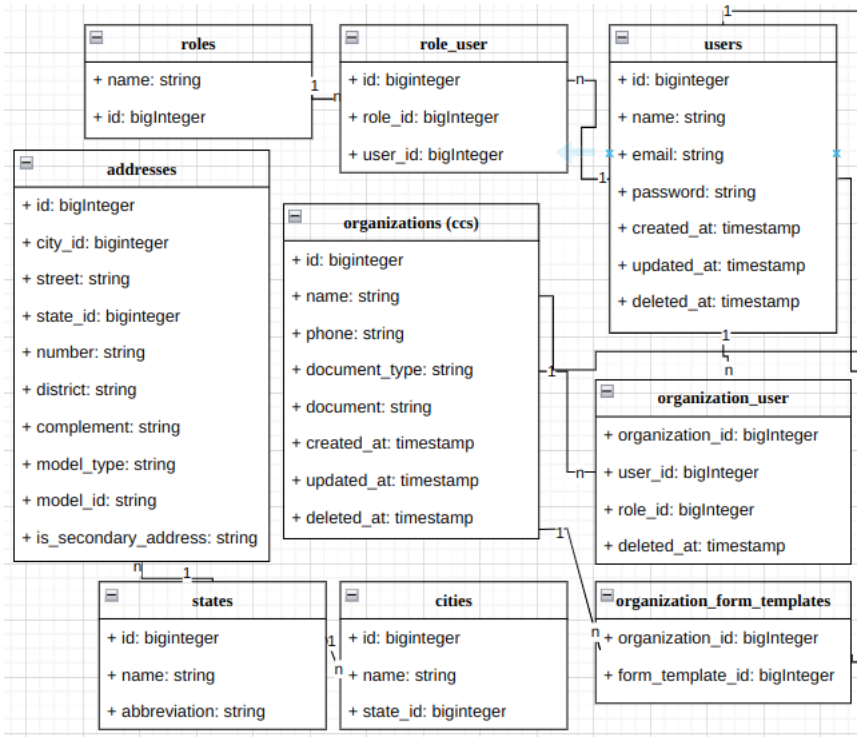
O banco de dados do protótipo inicial do Open Social Care foi feito usando o *Firestore Database* do *Firebase*, e como a refatoração da infraestrutura utiliza o banco de dados *PostgreSQL* junto com o *framework Laravel*, não foi possível reaproveitar o modelo de dados anterior. Por se tratarem de estruturas bem diferentes uma da outra, o *PostgreSQL* é um banco de dados objeto relacional, ele utiliza do SQL, enquanto o *Firestore Database* utiliza de dados não relacionais o *Not Only SQL* (NoSQL). O modelo de tabela relacional do SQL é altamente estruturado e requer que os dados sejam organizados em tabelas com esquemas pré-definidos. O modelo de documento do NoSQL, por outro lado, é mais flexível e permite que os dados sejam armazenados em documentos que não precisam seguir um esquema rígido (IMPACTA, 2023). Logo, nessa refatoração do sistema precisou-se construir um novo modelo de banco de dados, por meio de análises e reuniões, levando em conta os requisitos coletados e funcionalidades, foram identificadas as entidades necessárias, como por exemplo organizações e sujeitos, definido chaves primárias e chaves estrangeiras que estabelecem os relacionamentos entre as tabelas, com base nisso, um diagrama visual do banco de dados foi criado e está disponível no Apêndice A na Figura 19 [1].

Tratando-se dos elementos básicos do sistema, o banco de dados conta com tabelas para o cadastro de organizações, as quais podem possuir usuários. Usuários por sua vez podem possuir *roles* (perfis) para diferenciar seus diferentes papéis e permissões. A Figura 12 demonstra um fragmento da modelagem destes elementos.

Os sujeitos atendidos pelas assistentes sociais, neste trabalho denominados PPLs, são representados pela tabela *subjects* mostrado na Figura 13. Esta entidade é modelada separadamente dos usuários, visto que são indivíduos que não possuem login ou acesso direto ao sistema.

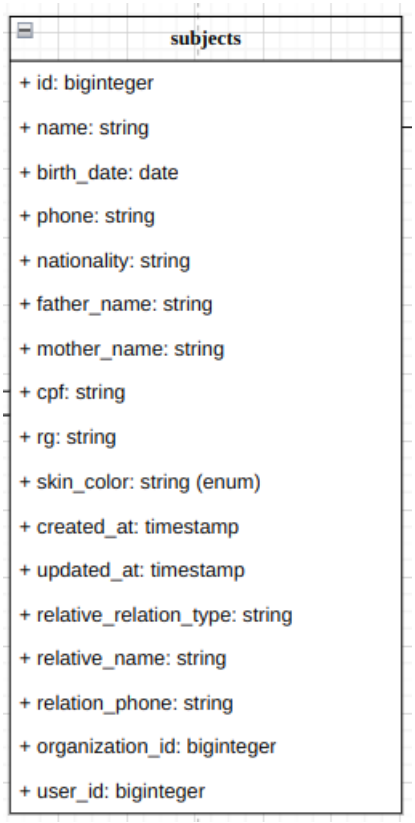
[1] Uma versão digital do diagrama também pode ser visualizada em: Link para o diagrama do banco de dados

Figura 12 – Fragmento do diagrama de banco de dados destacando as organizações e usuários



Fonte: Autoria própria (2024).

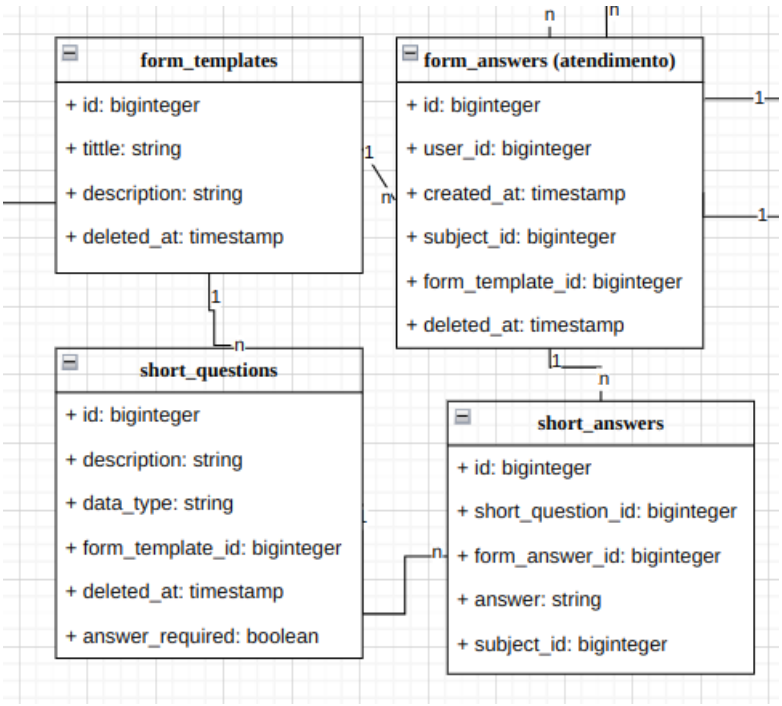
Figura 13 – Fragmento do diagrama de banco de dados destacando os sujeitos



Fonte: Autoria própria (2024).

A atividade realizada por assistentes sociais (usuários do sistema), é registrada no banco de dados por meio das tabelas form_answers que simboliza os atendimentos, short_answers a qual representam os preenchimentos das perguntas cadastradas nos questionários personalizados. Tais questionários são baseados em *templates* representados pelas tabelas form_templates, tabela onde são armazenadas informações principais referentes ao formulário e a tabela short_questions que seriam as perguntas para serem preenchidas pela assistente social. Esse fluxo de banco de dados pode ser observado na Figura 14.

Figura 14 – Fragmento do diagrama de banco de dados destacando os atendimentos e questionários personalizados



Fonte: Autoria própria (2024).

5.6 Implementação

A implementação do projeto começou com a configuração do repositório na plataforma *Github*, foi criada uma organização Open Social Care para que os projetos específicos do tema ficassem alocados, facilitando o armazenamento e o compartilhamento do projeto. Logo dentro dessa organização se encontra o repositório *backend* do sistema, feita a implementação da API. [2] Para o início do desenvolvimento da API, foi configurado e inciado o projeto com o *framework Laravel* utilizando o *Sail* para a configuração do ambiente de desenvolvimento. Para isso, foi utilizado o passo a passo da documentação oficial do *Laravel*, que pode ser encontrada nesse link https://laravel.com/docs/10.x/installation. O *Laravel Sail* é uma interface de linha de comando leve para interagir com o ambiente de desenvolvimento *Docker* padrão do *Laravel* (LARAVEL, 2023a), *Sail* é um facilitador para utilizar o *Docker* no ambiente de desenvolvimento. O projeto foi criado usando a configuração inicial do *Laravel* junto com o *PostgreSql*.

A configuração inicial usando o *Laravel Sail* vem com arquivos e funcionalidades padrões para começar a aplicação. Porém como o projeto se trata de uma API algumas funcionalidades e arquivos não são necessárias, por isso foi utilizado o *Starter Kit do breeze*, um inicializador para o projeto que vem com configurações especificas para o desenvolvimento. No caso, foi usado visando apenas o *backend*, logo remove os recursos destinados às *views* do projeto e adiciona funcionalidades básicas para gestão de usuários. Esses *kits* estruturam automaticamente seu aplicativo com as rotas, controladores e visualizações necessárias para registrar e autenticar os usuários do seu aplicativo (LARAVEL, 2023c).

Iniciando a implementação do sistema, seguindo o modelo de dados planejado, elaborou-se as *migrations* (migrações) [3] de todas as tabelas do banco de dados, também referenciando os relacionamentos entre elas. Na Listagem 1 é apresentado um exemplo de *migration*, que representa a criação da tabela *subjects* (sujeitos) com seus atributos. Seguindo a criação das *migrations*, foram construídos os *models* (modelos) [4] para cada tabela do banco de dados. Nos *models* é realizada a associação dos relacionamentos ao qual a tabela tem ou pertence. Na Listagem 2 é apresentada uma classe *model* do *subject* do sistema, contendo seus atributos e relacionamentos.

A cada etapa de construção da aplicação foi realizada a implementação de testes, que são importantes para garantir a qualidade e confiabilidade do sistema. Os testes permitem identificar falhas, economizando tempo e recursos. O projeto conta com testes unitários, que servem para testar partes individuais de um código, como por exemplo uma função ou método. No caso do Open Social Care, esta técnica é utilizada para testar funcionalidades dos *models*, como por

[2] Link do Github para a organização: https://github.com/open-social-care

[3] Migrations: São uma forma de gerenciar as mudanças na estrutura do banco de dados de uma aplicação. Eles permitem que você crie, altere ou exclua tabelas, colunas, índices e outros elementos do banco de dados de forma consistente e controlada (KRIGER, 2023).

[4] *Model*: é uma classe para representar uma entidade do banco de dados, com ela é possível inserir, editar e recuperar dados

exemplo seus relacionamentos, esses testes, garantem que os *models* foram implementados da forma correta e não haverá erros, como também que a aplicação está seguindo à modelagem prevista, um exemplo de teste pode ser visto na Listagem 3, onde foi testado o relacionamento do *model* de *subject*. Como suporte para os testes, foram criadas *factories*, elas são usadas para criação de dados falsos, muito utilizados nos testes, que ajudam a simular um *model* em um cenário real da aplicação, sem precisar que seja acessado ou criado via banco de dados, facilitando o desenvolvimento, na Listagem 4 temos uma *factory* criada para o *model* do *subject*, onde são atribuídos dados falsos que são gerados com a função *fake* do *Laravel* para os campos necessários do *subject* a ser usado em testes.

Listagem 1 – *Migration* criada para os *subjects*

```php
<?php

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('subjects', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->date('birth_date');
            $table->string('nationality')->nullable();
            $table->string('phone')->nullable();
            $table->string('father_name')->nullable();
            $table->string('mother_name')->nullable();
            $table->string('cpf')->nullable();
            $table->string('rg')->nullable();
            $table->enum('skin_color',
                array_column(SkinColorsEnum::cases(), 'value')
            )->nullable();
            $table->string('relative_relation_type')->nullable();
            $table->string('relative_name')->nullable();
            $table->string('relative_phone')->nullable();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('subjects');
    }
};
```

Fonte: Autoria própria (2024).

Listagem 2 – *Model* criada para o *subject*

```php
<?php
class Subject extends Model
{
    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    protected $fillable = [
        'name',
        'birth_date',
        'nationality',
        'phone',
        'father_name',
        'mother_name',
        'cpf',
        'rg',
        'skin_color',
        'relative_relation_type',
        'relative_name',
        'relative_phone',
        'created_at',
        'updated_at',
    ];

    /**
     * The attributes that should be cast.
     *
     * @var array<string, string>
     */
    protected $casts = [
        'birth_date' => 'datetime',
    ];

    public function formAnswers(): HasMany
    {
        return $this->hasMany(FormAnswer::class);
    }
}
```

Fonte: Autoria própria (2024).

Listagem 3 – Teste unitário criado para o *model* de *subject*

```php
<?php

class SubjectTest extends TestCase
{
    use RefreshDatabase;

    public function testShortQuestionHasManyFormAnswer()
    {
        $subject = Subject::factory()->createOneQuietly();
        $formAnswer1 = FormAnswer::factory()->for($subject)
            ->createOneQuietly();
        $formAnswer2 = FormAnswer::factory()->for($subject)
            ->createOneQuietly();

        $this->assertTrue($subject->formAnswers->contains($formAnswer1));
        $this->assertTrue($subject->formAnswers->contains($formAnswer2));
    }
}
```

Fonte: Autoria própria (2024).

Listagem 4 – *Factory* criada para o *model* de *subject*

```php
<?php

class SubjectFactory extends Factory
{
    /**
     * Define the model's default state.
     *
     * @return array<string, mixed>
     */
    public function definition(): array
    {
        $skinColors = array_column(SkinColorsEnum::cases(), 'value');

        return [
            'name' => fake()->name,
            'relative_name' => fake()->name,
            'relative_relation' => fake()->name,
            'birth_date' => fake()->dateTimeThisDecade,
            'contact_phone' => fake()->phoneNumber,
            'cpf' => fake()->unique()->numerify('###.###.###-##'),
            'rg' => fake()->unique()->numerify('##.###.###-#'),
            'skin_color' => $this->faker->randomElement($skinColors),
        ];
    }
}
```

Fonte: Autoria própria (2024).

Como mencionado anteriormente, no fluxo de desenvolvimento subseção 4.2.4, para cada tarefa implementada no sistema, foi criado um *pull request* para revisão da atividade implementada. Na Tabela 2 são apresentadas as tarefas realizadas durante o desenvolvimento da aplicação, em que é possível ver o título, descrição e o tipo, para o tipo temos *feature*, para novas funcionalidades e *fix*, para correções.

Titulo	Descrição	Tipo
Setup Project	Inicialização do código do projeto utilizando o *Laravel* com o *Sail*.	*Feature*
Add Migrations	Nessa tarefa foi criado as tabelas para cada entidade e funcionalidade do sistema, junto dessa tarefa também foi adicionado os *models* (Modelos)referentes a cada tabela criada, além de testes de *feature* para verificar se os relacionamentos estavam corretos.	*Feature*
Add address to subject	Tarefa de correção pois foi visto a falta do relacionamento entre sujeitos e endereços. Um sujeito tem endereços.	Fix
Create Seeders	Criação de *seeders* para o projeto, *seeders* são feitas para pré popular o banco de dados com informações foi criado 5 *seeders*, uma *seeder* para criação do usuário principal do sistema o *admin*, *seeders* para popular a tabela *states* (estados) e *cities* (cidades) referentes ao brasil.	*Feature*
Login	Adicionado a autenticação de usuários no sistema, nessa atividade foi feito o *login* de usuário, recuperação e atualização de senha e dados. Testes unitários e de *feature* para as funcionalidades desenvolvidas também foram feitos.	*Feature*
Config project	Em pesquisas foi visto que o *Laravel* possui algumas configurações que podem ajudar a deixar o sistema mais seguro e melhorar o funcionamento dele também, essa tarefa foi feita para adicionar algumas configurações que foram vistas como necessária e que poderiam agregar a melhoria do sistema	*Fix*
Admin Users	Nessa atividade foi implementada a área de usuários para o perfil de administrador do sistema. Foi desenvolvido criação, edição, visualização e remoção (CRUD), como testes para unitários e de *feature*.	*Feature*
Admin Organizations	Implementação do CRUD de organizações e testes, para o perfil administrador do sistema.	*Feature*
Default return in response	Essa tarefa foi uma correção para padronizar o tipo de retorno das response (respostas) *json* da *api*, esse desenvolvimento melhorou o formato de retorno dos dados para o *front-end* do sistema.	*Fix*
User info in return login	Adicionado informações do usuário no retorno do login, um pedido feito para facilitar os dados para o *front-end*	*Fix*
Add role to organizations	Atividade de correção para adicionar *role* (perfil) alocado a organização, pois foi visto a necessidade de que um usuário possa ter múltiplos perfis dentro do sistema, como também para cada organização ele possuir uma role diferente.	*Fix*
Audit	Adicionado tabela de *logs* para o sistema, testes unitários para a funcionalidade e adicionado o primeiro *log* para o login de usuários.	*Feature*
Manager Users	Desenvolvimento do CRUD de usuários e testes, para o perfil de gestor da aplicação.	*Feature*
Fix organization show	No método *show* da organização para o *admin* faltaram algumas informações para serem mostradas, essa tarefa foi uma correção para isso.	Fix
Lang validation	Em testes foi visto que faltaram algumas traduções para alguns atributos no retorno de erros de *requests* do *Laravel*, nessa atividade foi feito a adição.	Fix
List users organization	Atividade feita a pedido de melhoria para o *front-end*, foram adicionados dois métodos para organização, um método para listar usuários que não fazem parte da organização, e um método para usuários que fazem parte.	*Fix*
Extra info organization	Inseridas duas novas informações para as organizações, telefone e referência de sujeito *(subject_ref)*, a referência surgiu ao ver que cada organização tem um nome para referenciar sujeitos, assim cada uma poderia personalizar.	*Fix*
Manager Organizations	Implementação do CRUD de organizações e testes, para o perfil manager do sistema.	*Feature*
Manager Form templates	Implementação do CRUD e testes para os formulários personalizados, perfil gestor.	*Feature*
Social Assistant Subjects	Implementação do CRUD e testes para os sujeitos, perfil assistente social.	*Feature*
Social Assistant Organizations	Implementação do CRUD de organizações e testes, para o perfil assistente social.	*Feature*
Social Assistant Form answer	Implementação do CRUD de Atendimentos, para o perfil assistente social. Essa tarefa é referente a utilizar um *template* como base para responder as perguntas montadas nele.	*Feature*

Tabela 2 – Tabela de tarefas

5.7 Testes automatizados

Os testes automatizados são definidos como programas que realizam testes em *softwares* que estão em construção de modo padronizado, sem que seja necessária a intervenção humana. Isso porque eles contam com funcionalidades que são capazes de testar automaticamente todos os aspectos de uma plataforma, a fim de garantir um desempenho adequado. O procedimento traz muito mais precisão e agilidade para a etapa de testes, o que permite que você tenha facilidade para encontrar falhas de segurança, *bugs* e demais problemas que venham a prejudicar o uso da aplicação conforme projetado inicialmente (ENGINEERING, 2021). O desenvolvimento de testes facilita e ajuda na produção do sistema, visto que ao implementar algo ou fazer uma alteração no código, o teste pode apontar possíveis erros antes que esse ele esteja em um ambiente de produção e acabe sendo o usuário que encontre dificuldade de usar o sistema ou possíveis falhas.

No sistema Open Social Care foram implementados testes de *feature*, em que são realizadas verificações para garantir que uma função do sistema ocorra como o esperado. Os testes foram feitos para todas as funcionalidade do sistema, ajudando a validar possíveis falhas para que pudessem ser corrigidas com antecedência, garantindo um sistema mais seguro e funcional. Na Figura 15 tem-se como exemplo dois testes de *feature*, os dois testam o funcionamento da função *store* do *controller* de *users*, método de criação de usuários pelo sistema, na primeira função o método é testado passando as informações corretas e verificando se a resposta teve sucesso, no segundo é testado passando informações incorretas e verificando se o retorno é de erro.

Figura 15 – Teste de *feature* criação de usuário

```php
public function testStoreMethod()
{
    $userData = [
        'name' => 'Nome do Usuário',
        'email' => 'usuario@example.com',
        'password' => 'senha123',
        'password_confirmation' => 'senha123',
    ];

    $response = $this->postJson(route( name: 'admin.users.store'), $userData);

    $response->assertStatus( status: HttpResponse::HTTP_OK)
        ->assertJson(['message' => __( key: 'messages.common.success_create')]);

    $this->assertDatabaseHas( table: 'users', ['email' => 'usuario@example.com']);
}

public function testStoreMethodValidation()
{
    $response = $this->postJson(route( name: 'admin.users.store'), []);

    $response->assertStatus( status: HttpResponse::HTTP_UNPROCESSABLE_ENTITY)
        ->assertJsonStructure([
            'message',
            'errors' => [
                'name',
                'email',
                'password',
            ],
        ]);
}
```

Fonte: Autoria própria (2024).

Em blocos menores de código também foram desenvolvidos testes unitários que consistem em testar métodos e funções individuais de classes, componentes ou módulos usados pelo software (PITTET, 2024). Eles serviram para verificar de forma individual se as funções ou métodos funcionavam corretamente. Na Figura 16 é exemplificado um teste unitário de criação do usuário, o método *store* do *controller* de *users* faz uso de um arquivo separado o *UserCreateAction*, que possui a lógica para a criação do usuários, o teste valida essa função de forma unitária.

Figura 16 – Teste unitário para a classe UserCreateAction

```php
<?php

namespace Tests\Unit\Actions\Admin\User;

use ...

class UserCreateActionTest extends TestCase
{
    use RefreshDatabase;

    public function testExecuteAction()
    {
        $userDTO = new UserDTO([
            'name' => fake()->name,
            'email' => fake()->email,
            'password' => bcrypt( value: 'secret'),
        ]);

        UserCreateAction::execute($userDTO);

        $this->assertDatabaseHas( table: 'users', ['email' => $userDTO->email]);
    }
}
```

Fonte: Autoria própria (2024).

Para complementar e garantir a funcionalidade do código junto com os testes, foi construído *pipeline CI* [5] com o GitHub *actions*. O GitHub *Actions* é uma plataforma de integração contínua e entrega contínua (CI/CD) que permite automatizar a sua compilação, testar e pipeline de implantação. É possível criar fluxos de trabalho que criam e testam cada *pull request* no seu repositório, ou implantar *pull requests* mesclados em produção (GITHUB, 2024). A cada

[5] *pipeline CI*: O pipeline de CI/CD tem como objetivo promover a colaboração entre as equipes de desenvolvimento e operações. Ambos os conceitos têm como foco a automação dos processos de integração de código. Isso significa agilizar os processos para levar uma ideia (como uma nova funcionalidade, solicitação de melhoria ou correção de um *bug*) do desenvolvimento para a implantação no ambiente de produção, onde vai gerar valor para os usuários (HAT, 2023).

pull request criada é rodado uma *action* para validar que todos os testes estão rodando corretamente e que nenhum vai falhar. Na Figura 17 temos o *pipeline CI* dos testes rodados com sucesso, é possível observar que o total de testes implementados no sistema é de 288. Do total de testes desenvolvidos 133 foram testes de *feature* e 155 testes unitários.

Figura 17 – *pipeline CI* dos testes

Test with phpunit

```
1  ▶ Run vendor/bin/phpunit --coverage-text
19 PHPUnit 10.3.1 by Sebastian Bergmann and contributors.
20
21 Runtime:       PHP 8.2.19 with Xdebug 3.3.2
22 Configuration: /home/runner/work/backend/backend/phpunit.xml
23
24 ............................................................  63 / 288 ( 21%)
25 ............................................................ 126 / 288 ( 43%)
26 ............................................................ 189 / 288 ( 65%)
27 ............................................................ 252 / 288 ( 87%)
28 ....................................                         288 / 288 (100%)
29
30 Time: 00:19.781, Memory: 76.50 MB
31
32 OK (288 tests, 666 assertions)
33
34
35 Code Coverage Report:
36   2024-06-05 22:27:01
37
38  Summary:
39   Classes: 65.35% (83/127)
40   Methods: 69.97% (240/343)
41   Lines:   79.95% (1416/1771)
```

Fonte: Autoria própria (2024).

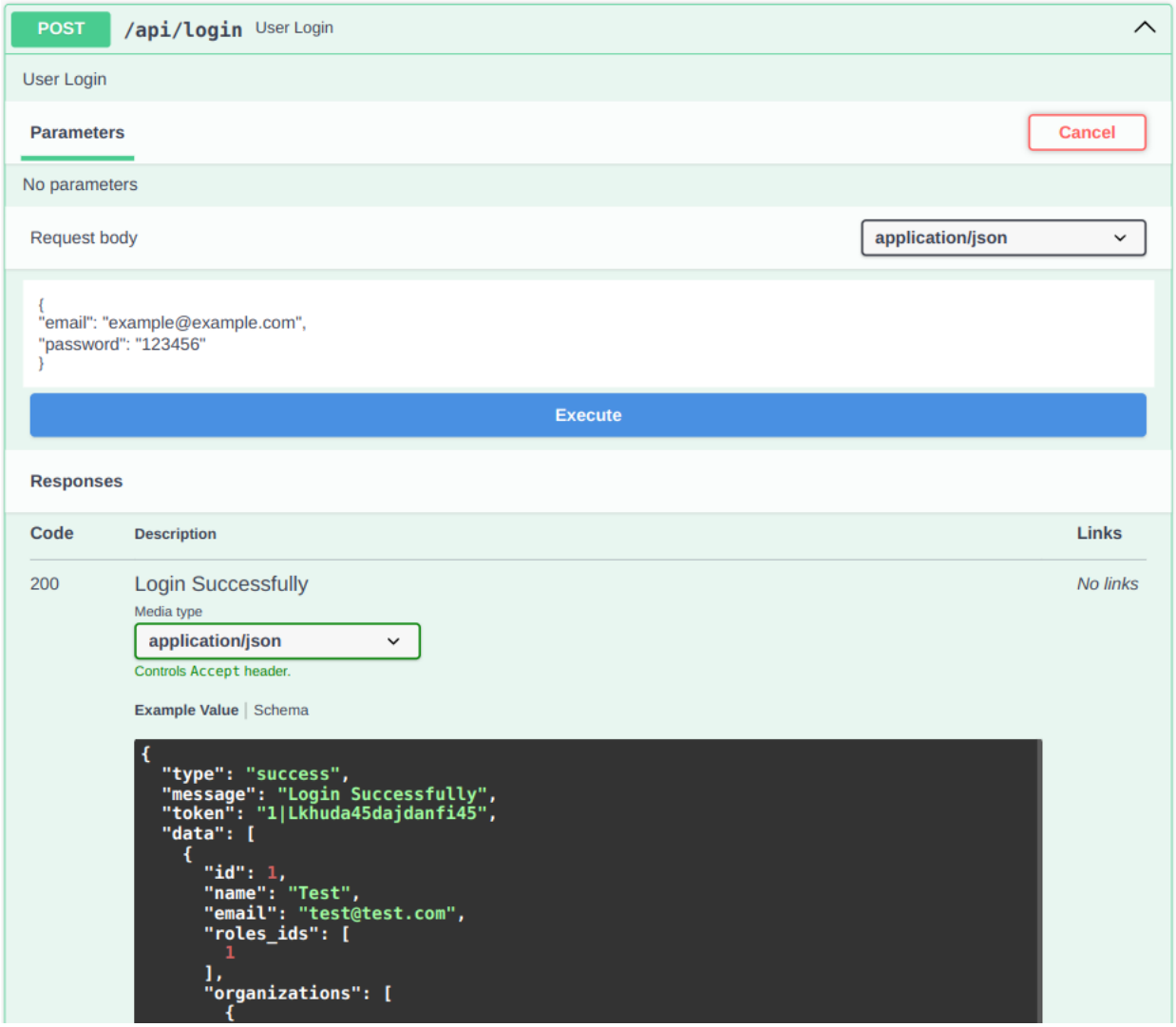
5.8 Documentação da API

Para o desenvolvimento de uma boa API nos dias atuais é necessário que ela possua uma documentação, pois facilita a integração entre aplicativos, clarifica suas funcionalidades, ajuda a manter um padrão no sistema e é essencial para garantir um desenvolvimento eficiente. Nesse trabalho foi crucial a sua implementação devido a implementação de um *front-end* do sistema, ela auxiliou para um trabalho rápido e eficaz.

Utilizou-se da ferramenta *Swagger*, trata-se de uma aplicação código aberto (*open source*) que auxilia desenvolvedores nos processos de definir, criar, documentar e consumir

APIs REST [6]. Em suma, o *Swagger* visa padronizar este tipo de integração, descrevendo os recursos que uma API deve possuir, como *endpoints* [7], dados recebidos, dados retornados, códigos Hypertext Transfer Protocol (HTTP) e métodos de autenticação, entre outros. Ele simplifica o processo de escrever API, especificando os padrões e fornecendo as ferramentas necessárias para escrever API segura, com alto desempenho e escalável (GR1D, 2022). A Figura 18 mostra a documentação para a autenticação no sistema, nela é possível ver a rota, os parâmetros necessários para a requisição e também o seu retorno de sucesso e alguns dos parâmetros retornados nesta resposta.

Figura 18 – Fragmento da documentação *Swagger* de autenticação



Fonte: Autoria própria (2024).

[6] *APIs REST*: padrão de arquitetura de software para desenvolvimento de API

[7] *Endpoints*: São endereços/rotas (*URL*) para acessar recursos de uma aplicação web.

5.9 Trabalhos futuros

O sistema foi desenvolvido pensando nas principais funcionalidades para o seu funcionamento inicial, algumas implementações que estavam no escopo não foram realizadas. Uma delas foi nos questionários personalizados possuir a opção de criar perguntas de múltiplas escolhas, no atual sistema apenas é possível criar perguntas de texto. Outras funcionalidades que ficaram de fora seria a possibilidade do envio de arquivos e fotos também nos questionários personalizados e o retorno sobre os atendimento realizados pelas assistentes sociais.

O motivo para essas funcionalidades não terem sido desenvolvidas foi para manter o desenvolvimento inicial simples, para que pudesse ser testado e validado melhor e quando o projeto evoluísse mais, desenvolver funcionalidades mais complexas.

Para o nosso escopo do sistema e do trabalho, não foi descrito a possível geração de relatórios, o que seria ideal para os usuários do sistema manter um gerenciamento melhor dos seus atendimentos ou sobre suas organizações, como também ampliaria o uso do sistema. Outro ponto, é que apesar dos *logs* de atividades do usuário ter sido desenvolvida, não foi implementada uma tela para que o usuário administrador possa visualizar essas informações, sendo possível somente ver direto no banco de dados. Seria útil para que o usuário pudesse visualizar esses detalhes de forma rápida e prática.

A expectativa é que trabalhos futuros abordem essas implementações mencionadas, visto que o sistema tem uma grande contribuição para a comunidade e para que ele possua um grande avanço. Essas melhorias visam a eficiência, aprimoramento e usabilidade do sistema, sendo uma colaboração ideal para a continuidade deste.

6 CONSIDERAÇÕES FINAIS

Neste trabalho foi abordado o desenvolvimento de uma API para substituir a infraestrutura *backend* do protótipo Open Social Care. Diante desse desafio, foram descritas as principais ferramentas e métodos a serem utilizadas para alcançar os objetivos propostos. O protótipo inicial era fortemente dependente de serviços de terceiros, diante disso, acredita-se que propor uma nova arquitetura com infraestrutura passível de gerenciamento favorecerá a implantação e a manutenção do sistema em cenários reais. Além disso, destaca-se que a infraestrutura inicial não era fácil de ser configurada, precisando ser feita individualmente para cada instituição que irá utilizar o sistema podendo tornar o protótipo inviável para algumas finalidades.

Também diante dos documentos e ferramentas apresentados, similares ao Open Social Care e que são utilizado por assistentes sociais, é destacado que algumas das ferramentas são de utilização paga, sendo inviável para ser utilizado por algumas instituições, como é o caso do CCG, logo a construção do sistema favorece esse tipo de instituição por se tratar de um sistema gratuito e livre para uso. Este sistema visa atender a diferentes contextos com o uso de formulários personalizáveis, tornando-o genérico para atendimentos de assistentes sociais. Além de trazer o benéfico da generalização, o sistema poderá ser utilizado por diversas instituições sendo abrangente para atender vários objetivos e demandas, fazendo-o ter um maior alcance para sua utilização e, consequentemente, ampliando a contribuição do presente trabalho.

Para a construção do sistema, como citado foi utilizado o método *MoSCoW*, com ele foi possível conduzir o gerenciamento das tarefas por forma de priorização com base nos requisitos, o que permitiu foco claro nas funcionalidades e foi essencial para o desenvolvimento. Como citado na seção 5.9, algumas funcionalidades ficaram de fora do escopo inicial, e a redução desse escopo auxiliou para que um sistema funcional e simples pudesse ser entregue em primeiro momento para validação e testes.

Pensando no aprimoramento do sistema, seria ideal uma refatoração no banco de dados, visando melhorar os relacionamentos entre permissões, usuários e organizações, visto que precisou-se fazer grandes modificações durante o desenvolvimento, podendo essa parte do sistema ser otimizada e tratada de uma forma melhor. Esta refatoração simplificaria o sistema e também aprimoraria sua performance. Mesmo necessitando dessa melhoria, o desenvolvimento da API foi essencial para o sistema, pois tornou o desenvolvimento mais flexível e fácil para o crescimento dele, como também foi uma grande contribuição para a utilização dele pelas assistentes sociais de forma geral.

Esse trabalho concluiu seus objetivos iniciais e abriu possibilidade de futuras pesquisas e desenvolvimentos. A expectativa é que ele possa ser explorado ampliando o seu uso e impacto na comunidade.

REFERÊNCIAS

AMORIM, S. **O que é Kanban e como usar essa metodologia**. 2023. Disponível em: https://enotas.com.br/blog/kanban/#:~:text=O%20que%20%C3%A9%20a%20metodologia%20Kanban%3F,problemas%20no%20fluxo%20de%20trabalho. Acesso em: 12 set. 2023.

AQUILES, A. **Controlando Versões com Git e GitHub**. São Paulo, SP - Brasil: Casa do Código, 2014. 199 p.

BLASI, B. G. D. **O que é login? Conheça os principais métodos de acesso no meio digital**. 2022. Disponível em: https://tecnoblog.net/responde/o-que-e-login/#:~:text=Por%20que%20o%20login%20%C3%A9,confere%20mais%20seguran%C3%A7a%20aos%20usu%C3%A1rios. Acesso em: 07 nov. 2023.

COCKBURN, A. **Escrevendo Casos de Usos Eficazes: Um guia prático para desenvolvedores de software**. Porto Alegre, RS - Brasil: Bookman Editora, 2005. 256 p.

CONTEIGE. **PHP - Vantagens e Desvantagens**. 2023. Disponível em: https://conteige.cloud/php-vantagens-e-desvantagens/#:~:text=Suporte%20de%20quantidade%20de%20dados,vantagem%20dessa%20linguagem%20de%20programa%C3%A7%C3%A3o. Acesso em: 08 nov. 2023.

DANIELA C. **O Que é Docker e Como Ele Funciona?** 2023. Disponível em: https://www.hostinger.com.br/tutoriais/o-que-e-docker. Acesso em: 14 nov. 2023.

DATABRICKS. **Transações ACID**. 2023. Disponível em: https://www.databricks.com/br/glossary/acid-transactions. Acesso em: 08 nov. 2023.

ENGINEERING. **O que são testes automatizados e como usá-los na empresa?** 2021. Disponível em: https://blog.engdb.com.br/testes-automatizados/. Acesso em: 10 jun. 2024.

FIREBASE. **Cloud Firestore**. 2023. Disponível em: https://firebase.google.com/docs/firestore?hl=pt-br#:~:text=O%20Cloud%20Firestore%20%C3%A9%20um%20banco%20de%20dados%20NoSQL%20hospedado,em%20APIs%20REST%20e%20RPC. Acesso em: 11 set. 2023.

FIREBASE. **Cloud Storage para Firebase**. 2023. Disponível em: https://firebase.google.com/docs/storage?hl=pt-br#:~:text=download%20de%20objetos.-,Quer%20armazenar%20outros%20tipos%20de%20dados%3F,Firebase%20e%20o%20Google%20Cloud. Acesso em: 11 set. 2023.

FIREBASE. **Firebase Authentication**. 2023. Disponível em: https://firebase.google.com/docs/auth?hl=pt-br#:~:text=O%20Firebase%20Authentication%20fornece%20servi%C3%A7os,Facebook%20e%20Twitter%2C%20entre%20outros. Acesso em: 11 set. 2023.

GABARDO, A. C. **Laravel para Ninjas**. São Paulo, SP - Brasil: Novatec Editora Ldta, 2017. 184 p.

GESUAS. **Pesquisa e redesign de sistema de apoio à assistência social**. 2023. Disponível em: https://www.gesuas.com.br/. Acesso em: 30 out. 2023.

GIT, I. **Git is a free and open source distributed version control system.** 2023. Disponível em: https://git-scm.com. Acesso em: 12 set. 2023.

GITHUB. **Entendendo o GitHub Actions**. 2024. Disponível em: https://docs.github.com/pt/actions/learn-github-actions/understanding-github-actions. Acesso em: 10 jun. 2024.

GR1D. **Swagger: entenda o que é e como usar**. 2022. Disponível em: https://gr1d.io/2022/04/15/swagger/. Acesso em: 12 jun. 2024.

HAT, R. **O que é CI/CD?** 2023. Disponível em: https://www.redhat.com/pt-br/topics/devops/what-is-ci-cd#:~:text=O%20pipeline%20de%20CI%2FCD%20%C3%A9%20uma%20parte%20essencial%20da,processos%20de%20integra%C3%A7%C3%A3o%20de%20c%C3%B3digo. Acesso em: 10 jun. 2024.

IDS SISTEMAS - BRASIL GESTÃO PÚBLICA. **Pesquisa e redesign de sistema de apoio à assistência social**. 2018. Disponível em: https://old.tog.design/portfolio/ids-social/. Acesso em: 30 out. 2023.

IDS SOCIAL. **Prontuário SUAS - Manual de instruções para o registro das informações especificadas**. 2023. Disponível em: https://ids.inf.br/ids-social/#:~:text=O%20IDS%20Social%20garante%20integridade,%C3%A1reas%20de%20Assist%C3%AAncia%20Social%20municipais. Acesso em: 30 out. 2023.

IMPACTA, R. **Diferenças entre SQL e NoSQL: comparando bancos de dados relacionais e não relacionais**. 2023. Disponível em: https://www.impacta.com.br/blog/diferencas-entre-sql-e-nosql-comparando-bancos-de-dados-relacionais-e-nao-relacionais/#:~:text=O%20modelo%20de%20tabela%20relacional,precisam%20seguir%20um%20esquema%20r%C3%ADgido. Acesso em: 08 nov. 2023.

KINSTA, I. **O que é PostgreSQL?** 2023. Disponível em: https://kinsta.com/pt/base-de-conhecimento/o-que-e-postgresql/. Acesso em: 13 jan. 2023.

KRIGER, B. **O que são Migrations e como utilizá-las em seu projeto?** 2023. Disponível em: https://kenzie.com.br/blog/migrations/#:~:text=Migrations%20s%C3%A3o%20uma%20forma%20de,de%20forma%20consistente%20e%20controlada. Acesso em: 10 nov. 2023.

LARAVEL. **Laravel Sail**. 2023. Disponível em: https://laravel.com/docs/10.x/sail. Acesso em: 10 nov. 2023.

LARAVEL. **Laravel Sanctum**. 2023. Disponível em: https://laravel.com/docs/10.x/sanctum. Acesso em: 08 nov. 2023.

LARAVEL. **Laravel Starter Kits**. 2023. Disponível em: https://laravel.com/docs/10.x/starter-kits. Acesso em: 10 nov. 2023.

LUCID, S. I. **O que é diagrama entidade relacionamento**. 2023. Disponível em: https://www.lucidchart.com/pages/pt/o-que-e-diagrama-entidade-relacionamento. Acesso em: 13 set. 2023.

MINISTÉRIO DO DESENVOLVIMENTO SOCIAL E COMBATE À FOME. **Prontuário SUAS - Manual de instruções para o registro das informações especificadas**. 2013. Disponível em: https://aplicacoes.mds.gov.br/sagi/snas/vigilancia/doc/ManualdePreenchimentoProntuario_versao_preliminar.pdf. Acesso em: 30 out. 2023.

OLIVEIRA, A. de F. G. **O conselho da comunidade e suas interfaces**. 2012. Disponível em: https://meuartigo.brasilescola.uol.com.br/atualidades/o-conselho-comunidade-suas-interfaces.htm. Acesso em: 22 ago. 2023.

ORACLE. **O que é um banco de dados relacional (RDBMS)?** 2023. Disponível em: https://www.oracle.com/br/database/what-is-a-relational-database/. Acesso em: 13 set. 2023.

PHP, T. G. **O que é o PHP?** 2023. Disponível em: https://www.php.net/manual/pt_BR/intro-whatis.php. Acesso em: 12 set. 2023.

PIRES, R. **Aprenda a usar a técnica MoSCoW nos projetos da sua agência**. 2019. Disponível em: https://rockcontent.com/br/blog/metodo-moscow/. Acesso em: 12 set. 2023.

PITTET, S. **Diferentes tipos de testes de software**. 2024. Disponível em: https://www.atlassian.com/br/continuous-delivery/software-testing/types-of-software-testing. Acesso em: 10 jun. 2024.

POPOVSKI, I. **Laravel Security Features**. 2019. Disponível em: https://iwconnect.com/laravel-security-features/. Acesso em: 07 nov. 2023.

REMESSA, O. **Firebase: descubra para que serve, como funciona e como usar**. 2021. Disponível em: https://www.remessaonline.com.br/blog/firebase-descubra-para-que-serve-como-funciona-e-como-usar/. Acesso em: 05 set. 2023.

ROCHA, A. **Gerenciamento de Logs: como funciona**. 2021. Disponível em: https://www.opservices.com.br/gerenciamento-de-logs/. Acesso em: 07 nov. 2023.

SOCIÁGIL. **SociÁgil**. 2024. Disponível em: https://www.sociagil.com.br. Acesso em: 03 jun. 2024.

SUAS, R. **Prontuário Eletrônico do SUAS**. 2023. Disponível em: https://aplicacoes.mds.gov.br/prontuario/. Acesso em: 30 out. 2023.

APÊNDICE A – Diagrama do banco de dados

Figura 19 – Diagrama do banco de dados

