UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA
BACHARELADO EM ENGENHARIA DE SOFTWARE

IAN JERÔNIMO NOBRE BARRETO

DESENVOLVIMENTO DO MÓDULO DE GESTÃO DE VISITAS A MUSEUS DO SOFTWARE PUBLICUS

NATAL, RN
2025

IAN JERÔNIMO NOBRE BARRETO

DESENVOLVIMENTO DO MÓDULO DE GESTÃO DE VISITAS A MUSEUS DO SOFTWARE PUBLICUS

Monografia apresentada ao curso de graduação em Engenharia de Software, da Universidade Federal do Rio Grande do Norte, como requisito parcial à obtenção do título de Bacharel em Engenharia de Software.

Orientador: Prof. Bruno Santana da Silva.

NATAL, RN

2025

Ian Jerônimo Nobre Barreto

DESENVOLVIMENTO DO MÓDULO DE GESTÃO DE VISITAS A MUSEUS DO SOFTWARE PUBLICUS

Trabalho de Conclusão de Curso na modalidade Monografia, submetido como parte dos requisitos necessários para conclusão do curso de Engenharia de Software da Universidade Federal do Rio Grande do Norte

Banca Examinadora

Bruno Santana da Silva – UFRN
Orientador

Adja Ferreira de Andrade - UFRN

Jair Cavalcanti Leite - UFRN

Natal, RN
2025

Universidade Federal do Rio Grande do Norte - UFRN

Sistema de Bibliotecas - SISBI

Catalogação de Publicação na Fonte. UFRN - Biblioteca Central Zila Mamede

Barreto, Ian Jerônimo Nobre.
Desenvolvimento do módulo de gestão de visitas a museus do software publicus / Ian Jeronimo Nobre Barreto. - 2025.
86f.: il.

Monografia (graduação) - Universidade Federal do Rio Grande do Norte, Centro de Ciências Exatas e da Terra, Engenharia de Software.
Orientação: Prof. Dr. Bruno Santana da Silva.

1. Gestão museal - Monografia. 2. Sistema de informação - Monografia. 3. Flutter - Monografia. 4. Desenvolvimento multiplataforma - Monografia. I. Silva, Bruno Santana da. II. Título.

RN/UF/BCZM CDU 069.6:004

Elaborado por Jackeline dos Santos Pinheiro da Silva Maia Cavalcanti - CRB-15/317

RESUMO

A fim de ajudar os museus a gerenciar seus fluxos de visitações, em particular compreendendo seus papéis sociais como agentes da educação não formal, tornou-se evidente a necessidade de oferecer ferramentas tecnológicas que os auxiliem a cumprir sua missão cultural e educativa. Então, esse trabalho contribuiu com a evolução do desenvolvimento incremental da segunda versão do software Publicus, um software para gestão de agendamentos e visitas a museus, por meio do desenvolvimento do módulo para a gestão de visitas a museus. O processo de desenvolvimento deste módulo se iniciou com a produção do diagrama de casos de uso, suas descrições textuais e do diagrama de classes, respeitando a arquitetura original do software. Em seguida, realizou-se a programação da parte cliente deste módulo com a linguagem Dart e o framework Flutter e da parte servidor com a linguagem PHP. Por fim, sete casos de testes foram planejados para este módulo, sendo 4 deles implementados com testes automatizados utilizando a biblioteca Flutter Test. A implementação deste módulo de gestão de visitas do Publicus oferece aos museus uma ferramenta que auxilia a gestão de visitas avulsas e em grupo de diferentes tipos, incluindo visitas escolares agendadas ou não agendadas. Uma boa gestão das informações sobre visitação contribui para que os museus tenham condições de oferecer bons serviços aos seus visitantes. Para a educação, o Publicus facilita a gestão de agendamentos de visitas escolares a museus, com o potencial de fortalecer a interação entre instituições de ensino e museus e promover a aprendizagem articulada entre a educação formal e não formal.

Palavras-chave: gestão museal; sistema de informação; flutter; desenvolvimento multiplataforma.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Através da interação com o meio físico e social no decorrer da vida, as pessoas experienciam cotidianamente diversas oportunidades de aprender e desenvolverem seus entendimentos de mundo (Rego, 2013). O processo de aprendizagem estimulado por instituições de ensino, como escolas e universidades, com ações educativas planejadas e conteúdos previamente determinados pode ser chamado de educação formal. Entretanto, o aprendizado humano não é promovido exclusivamente por instituições de ensino. Alguns ambientes sociais também podem ser responsáveis por estimular e proporcionar a educação não formal (Cazelli; Vergara, 2007; Pereira; Silva; Reis, 2021; Santos, 2016; Trilla, 2003). Dentre as instituições responsáveis por colaborar com o processo de educação não formal estão os museus. O Conselho Internacional de Museus (ICOM) define museu como:

> "uma instituição de carácter permanente, sem fins lucrativos, ao serviço da comunidade e do seu desenvolvimento, aberto ao público e que adquire, conserva, divulga e expõe, com objetivos científicos, educativos e lúdicos, testemunhos tangíveis e intangíveis do homem e do seu meio ambiente." (Boylan, 2015, p. 241).

As diversas atividades fornecidas pelos museus possibilitam para o seu público a criação de oportunidades de interação, comunicação e reflexão associadas com o lazer. Desta forma, os museus podem ser considerados protagonistas na educação não formal devido às experiências proporcionadas em seus espaços (Marandino, 2001; Costa et al., 2007; Silva; Diniz, 2011; Köptcke, 2014).

Além disso, essas contribuições dos museus com a educação não formal podem ser otimizadas quando associadas com a educação formal proporcionada por instituições de ensino (Marandino; Contier, 2015; Rodrigues; Miguel; Aldabalde, 2022; Segatto; Oliveira, 2022). Contudo, devido ao volume de visitantes envolvidos, os museus necessitam gerenciar o fluxo de visitações escolares que recebem. Assim, eles podem melhor organizar as suas atividades e conseguirem atender a demanda das instituições de ensino (Almeida et al., 2024).

A contagem de público é fundamental como ferramenta de gestão para que o museu possa oferecer bons serviços à sociedade (Boylan, 2015; Cândido, 2014; Cazelli *et al.*, 2022). Por exemplo, a gestão do fluxo de visitantes pode indicar exposições de

maior público, necessidade de adequação dos serviços oferecidos, ou a necessidade de ampliação das ações educativas.

Ademais, em 13 de abril de 2021 o Instituto Brasileiro de Museus (Ibram) publicou a Portaria Ibram nº 291. Essa Portaria estabeleceu que todos os museus brasileiros deveriam informar o número total de visitantes no ano para o Ibram, com o intuito de subsidiar a Política Nacional de Museus (Brasil, 2003) e a Política Nacional de Educação Museal (Ibram, 2017).

Logo, a contagem de público em museus deve ser encarada como uma atividade crucial e obrigatória, que aumenta a qualidade não só do funcionamento da instituição, mas também do atendimento às necessidades dos visitantes.

Com o intuito de auxiliar os museus a gerir e analisar informações sobre os públicos que os visitam, tem sido desenvolvido o sistema Publicus, de propriedade da Universidade Federal do Rio Grande do Norte (UFRN) sob o registro de número BR512020000320-6 no INPI. Em sua primeira versão, ele possui dois módulos: um para contagem de público e outro para analisar os perfis de públicos visitantes. Essa versão vem sendo utilizada pelo Museu Câmara Cascudo (MCC) desde setembro de 2017.

No entanto, a versão inicial do Publicus não contemplava o agendamento de visitas escolares a museus. Então, uma nova versão desse sistema tem sido desenvolvida para auxiliar a gestão do processo de agendamento dessas visitas. Essa evolução do sistema começou com um trabalho de conclusão de curso de Design do Victor Paiva (2023). Ele investigou e caracterizou o problema de design da gestão de visitas escolares a museus para projetar um aplicativo móvel que os professores poderão utilizar nesses agendamentos.

Essa nova versão foi planejada para os seguintes papéis de usuário: responsável, recepcionista, pedagogo e administrador. O responsável é um professor encarregado por marcar visitas ao museu de estudantes da sua instituição de ensino. Os três últimos papéis são desempenhados por funcionários do museu. Cada papel de usuário pode executar um conjunto de funcionalidades específicas dentro de um módulo, ainda que exista uma boa sobreposição de funcionalidades entre eles.

No momento, o projeto da nova versão do Publicus foi continuado por Antônio Medeiros e Herus Costa para contemplar os módulos de outros perfis e a plataforma desktop. O Publicus está sendo desenvolvido com uma arquitetura cliente-servidor, prevista para funcionar em smartphone, tablet, desktop e web. O desenvolvimento da

nova versão já foi concluído para o módulo de agendamentos e visitas do perfil de responsável no smartphone, com colaboração deste autor. Antes da execução deste trabalho, Daniel Fernando e Bruno Santana iniciavam o desenvolvimento do módulo de agendamentos no desktop para o perfil de pedagogo. O módulo de visitas para o perfil de pedagogo ainda não havia sido contemplado em iniciativas anteriores.

1.1 OBJETIVO

Portanto, este trabalho teve como objetivo de desenvolver o módulo para a gestão de visitas a museus da segunda versão do Software Publicus. Este módulo atende principalmente a usuários no papel de recepcionista, mas também poderá ser utilizado por outros funcionários do museu (como os pedagogos) e pelos professores responsáveis pelas visitas realizadas.

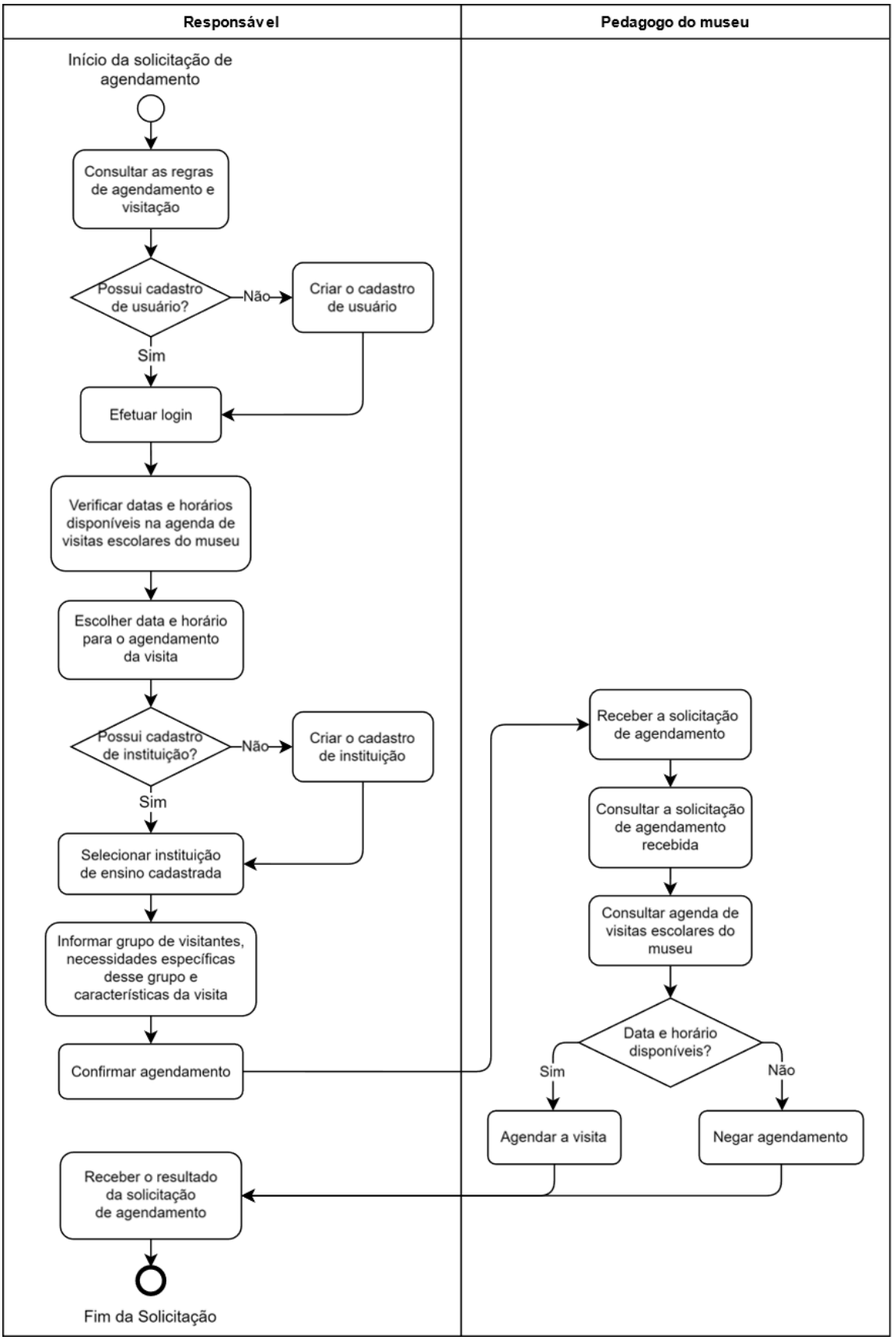
2 VISITAS ESCOLARES EM MUSEUS

Devido ao público escolar ser uma parte importante não só do MCC, mas também de outros museus, torna-se extremamente importante a gestão das visitas escolares aos museus. Através de três diferentes estudos realizados por Paiva e Silva (Paiva; Silva, 2024; Silva; Paiva, 2022, 2023), foram identificados três fluxogramas para tarefas básicas do processo de agendamento de visitas escolares a museus. São eles: solicitação de agendamento, solicitação de reagendamento e recepção de visitas.

Como podemos observar na Figura 1, o início da solicitação de agendamento se dá através do (professor) responsável com a consulta a regras de agendamento e visitação. Se o responsável não possuir um cadastro, ele deverá criar um usuário. Caso contrário, ele prossegue para a efetuação do login. Após a efetuação do login, o responsável verifica as datas e horários disponíveis para o agendamento e escolhe uma das datas para o agendamento da visita. Com uma data escolhida, o responsável terá agora que selecionar a instituição que realizará a visita. Uma nova instituição deve ser criada, caso ainda não tenha sido cadastrada. Então, o responsável informa o grupo de visitantes, as necessidades específicas do grupo e as características da visita e por último confirma o agendamento.

Após a confirmação do responsável, o pedagogo, que trabalha no museu, recebe e consulta a solicitação do agendamento. Então, ele verifica a agenda de visitas escolares do museu e, com base na disponibilidade, deve agendar a visita ou negar a solicitação. Assim, o responsável recebe o resultado da solicitação e o processo de agendamento de visita escolar é concluído.

Figura 1: Fluxograma de agendamento de visitas escolares ao museu

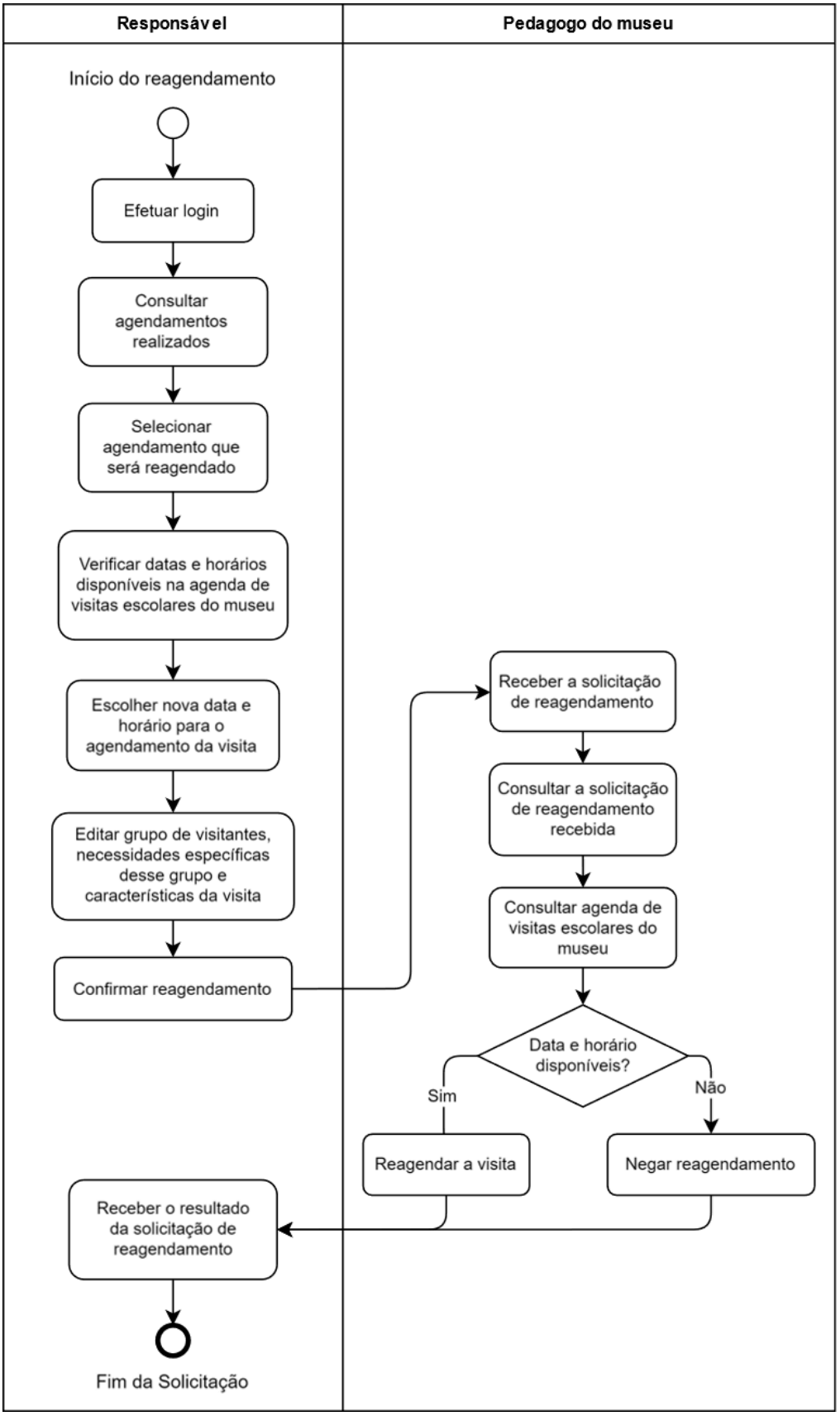


Fonte: Retirado de (Paiva, 2023).

A seguir temos o fluxograma de reagendamento de visita escolar ao museu (Figura 2), esse processo tem início com a efetuação do login pelo responsável. Ele consulta os seus agendamentos já realizados e seleciona o agendamento que deseja reagendar. Em seguida, o responsável verifica as datas e horários que estão disponíveis e seleciona a nova data para o agendamento da visita. Depois, ele edita o grupo de visitantes, assim como as necessidades específicas do grupo e as características da visita, por fim confirma o reagendamento.

Posteriormente, o pedagogo do museu recebe e consulta a solicitação de reagendamento recebida e confere a agenda de visitas do museu. Negando ou aceitando a solicitação com base na disponibilidade do museu, finalizando o processo de reagendamento.

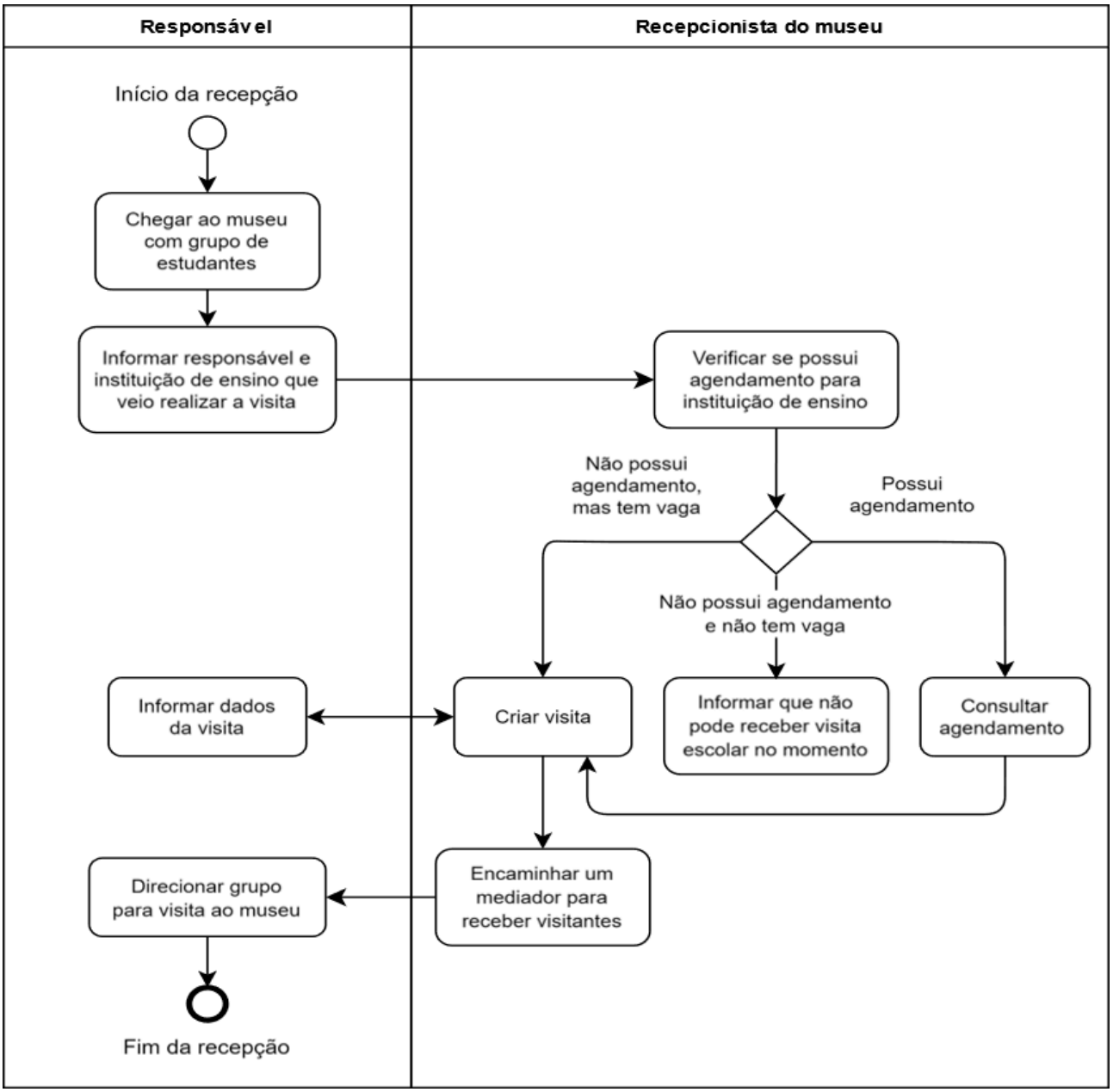
Figura 2: Fluxograma de reagendamento de visita escolar ao museu



Fonte: Retirado de (Paiva, 2023).

O processo de recepção de visitas escolares, representado na Figura 3, começa com a chegada do grupo de estudantes e responsáveis ao museu. Além disso, deve ser informado o responsável e a instituição de ensino que veio realizar a visitar. Logo em sequência, o recepcionista deve verificar se existe o agendamento para essa instituição.

Figura 3: Fluxograma de recepção de visitas escolares ao museu



Fonte: Retirado de (Paiva, 2023).

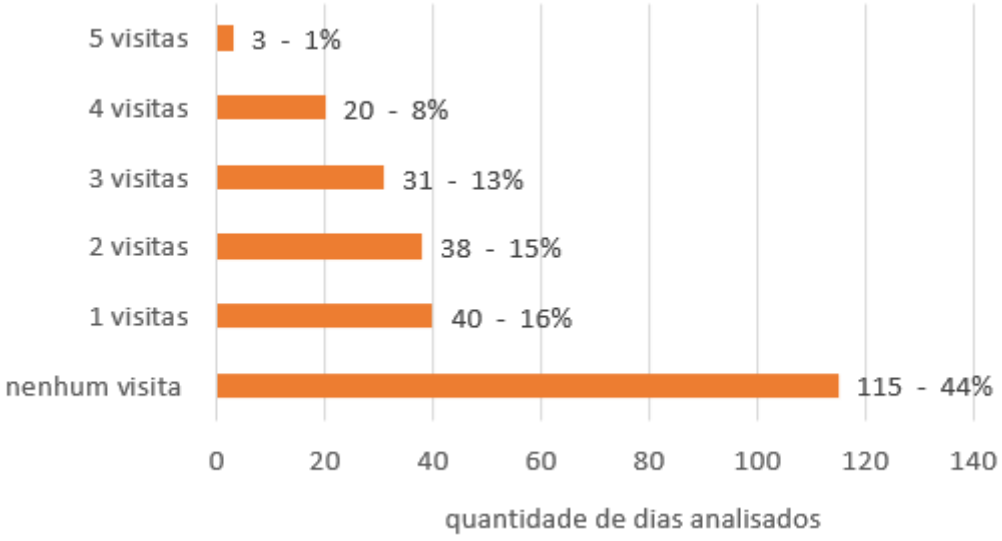
Se a instituição não possuir agendamento e o museu não tiver disponibilidade para receber a visita, o processo de recepção se encerra. Se a instituição não possuir

agendamento mas o museu tiver disponibilidade de receber a visita, o processo segue para o cadastro da visita. Se a instituição possuir agendamento, o processo continua com a seleção do agendamento e depois com o cadastro da visita.

No cadastro da visita, o responsável deverá informar os dados necessários para o recepcionista criar a visita. Por fim, o recepcionista encaminha um mediador para receber a visita e o responsável direciona o grupo de estudantes para a visitação, concluindo o processo de recepção de visitas.

Entre abril e dezembro de 2019, foi realizada uma pesquisa sobre agendamentos de visitas escolares do Museu Câmara Cascudo que indicou a diversidade e a recorrência anual das visitas escolares nesta instituição (Silva; Medeiros, 2021). Durante o período analisado, constatou-se que na maioria (56%) dos dias o museu recebeu pelo menos uma visita. O museu recebeu até cinco visitas escolares num único dia (Figura 4).

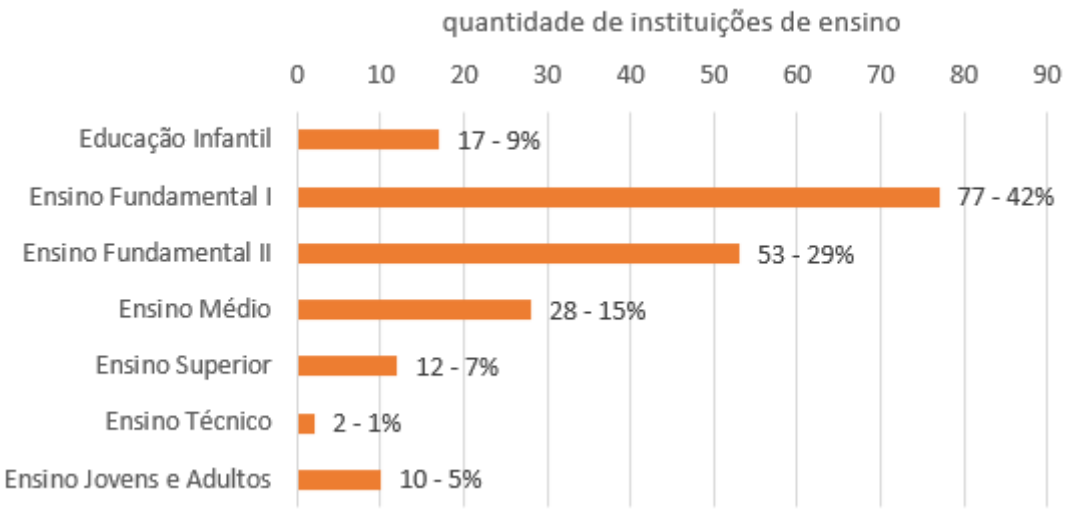
Figura 4: Quantidade de agendamento de visitas de instituições de ensino em um dia



Fonte: Retirado de (Silva; Medeiros, 2021).

Além disso, esta pesquisa observou que apesar de 42% das visitas serem realizada por alunos do Ensino Fundamental I, o museu chegou a receber visitas dos mais diversos níveis de ensino, desde da Educação Infantil até o Ensino Superior. Como podemos visualizar na Figura 5.

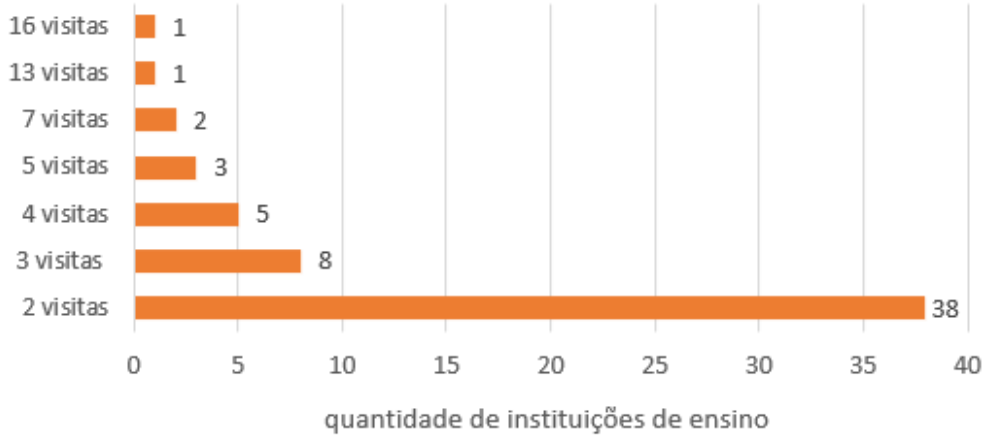
Figura 5: Quantidade de visitas por nível de ensino



Fonte: Retirado de (Silva; Medeiros, 2021).

Ademais, é válido ressaltar a recorrência em que as instituições visitam o museu. A pesquisa notou que um total de 58 instituições de ensino retornaram para visitar o museu pelo menos uma vez no mesmo ano (Figura 6). Com uma dessas instituições realizando o número de 16 visitas nesse período, sendo esse o número máximo de visitas que uma instituição realizou.

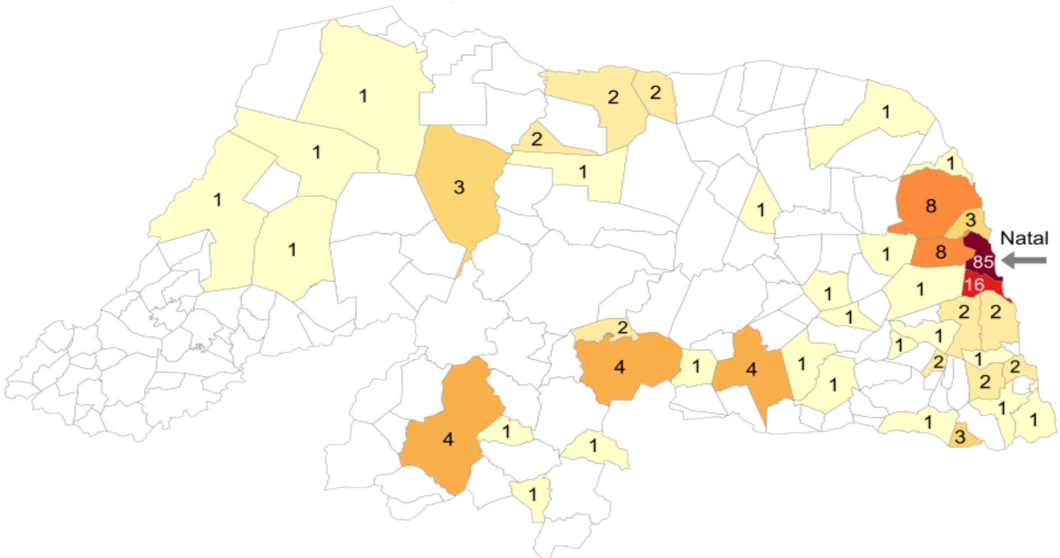
Figura 6: Quantidade de instituições de ensino por número de visitas ao MCC



Fonte: Retirado de (Silva; Medeiros, 2021).

Por último, é justo enfatizar a diversidade de origem das instituições de ensino que visitaram o Museu Câmara Cascudo. O estudo constatou que o museu recebeu visitas de 3 estados. Sendo 179 instituições (97%) do estado do Rio Grande do Norte, 3 instituições (2%) do estado de Pernambuco e 2 instituições (1%) da Paraíba. O museu também foi visitado por instituições de 43 cidades no Rio Grande do Norte, 26% das cidades no estado (Figura 7).

Figura 7: Quantidade de instituições de ensino que agendaram visitas ao MCC por cidades do RN



Fonte: Retirado de (Silva; Medeiros, 2021).

3 O SISTEMA PUBLICUS

Até o momento, conhecemos poucos softwares com objetivos semelhantes ao Publicus. O Iandé[1] é um plugin para o WordPress que permite o visitante realizar *check-in* nas visitas agendadas. Porém, não há possibilidade de registrar visitas avulsas e não há participação de funcionários do museu para conferência destes dados cadastrados. Já o Paytour[2] é um sistema de gestão de venda de ingressos, que não atende às necessidades de museus com entrada gratuíta. Como este trabalho contribui para a evolução do sistema Publicus, as versões existentes serão apresentadas a seguir.

3.1 PRIMEIRA VERSÃO DO SISTEMA PUBLICUS

O software Publicus, registrado pelo Instituto Nacional de Propriedade Industrial (INPI) com número BR512020000320-6, foi desenvolvido com o objetivo de auxiliar na gestão de informações sobre públicos que visitam museus ou outras instituições similares. O Publicus deveria não só acompanhar o fluxo de visitação de um museu, mas também deveria possibilitar às pessoas que trabalham na gestão realizarem a análise dos dados de visitação coletados. Com o intuito de cumprir esses requisitos, o sistema foi dividido em dois módulos: um para registro de dados da contagem de público e outro para realizar a análise dos dados coletados.

Originalmente, o software foi projetado pensando nos seguintes papéis de usuário: recepcionista e gestores do museu. Os funcionários que trabalham na recepção teriam acesso apenas ao módulo de registro de dados. Já os gestores têm acesso tanto ao módulo de registro quanto ao módulo de análise dos dados do Publicus.

Para ilustrar melhor as funcionalidades e o escopo do sistema, elaboramos neste trabalho por engenharia reversa o diagrama de casos de uso (Figura 8) e o modelo de dados (Figura 9) da primeira versão do Publicus.

[1] https://wordpress.com/pt-br/plugins/iande
[2] https://www.paytour.com.br/segmentos/sistema-de-reservas-para-museus/

Figura 8: Diagrama de casos de uso da primeira versão do sistema Publicus

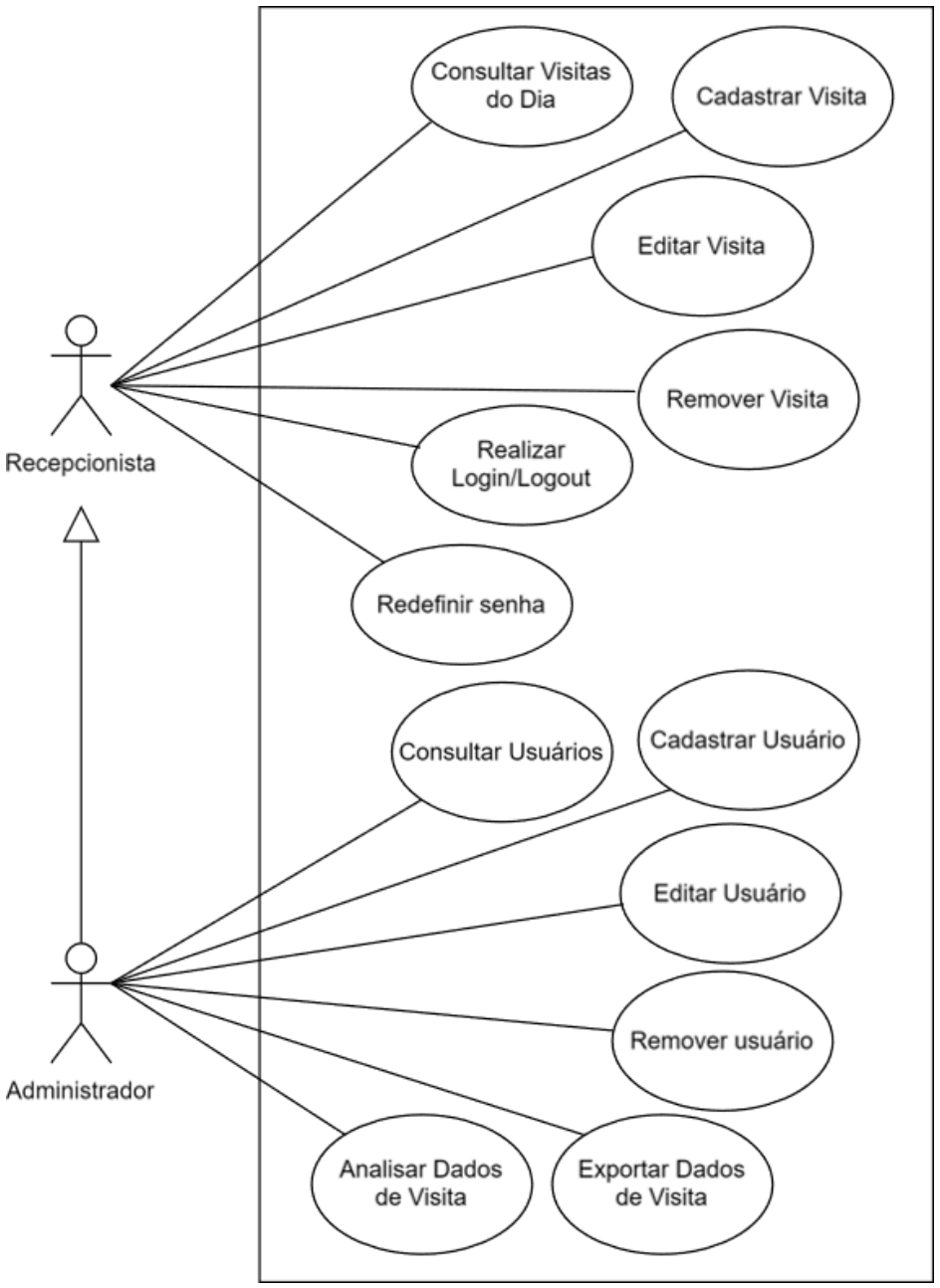
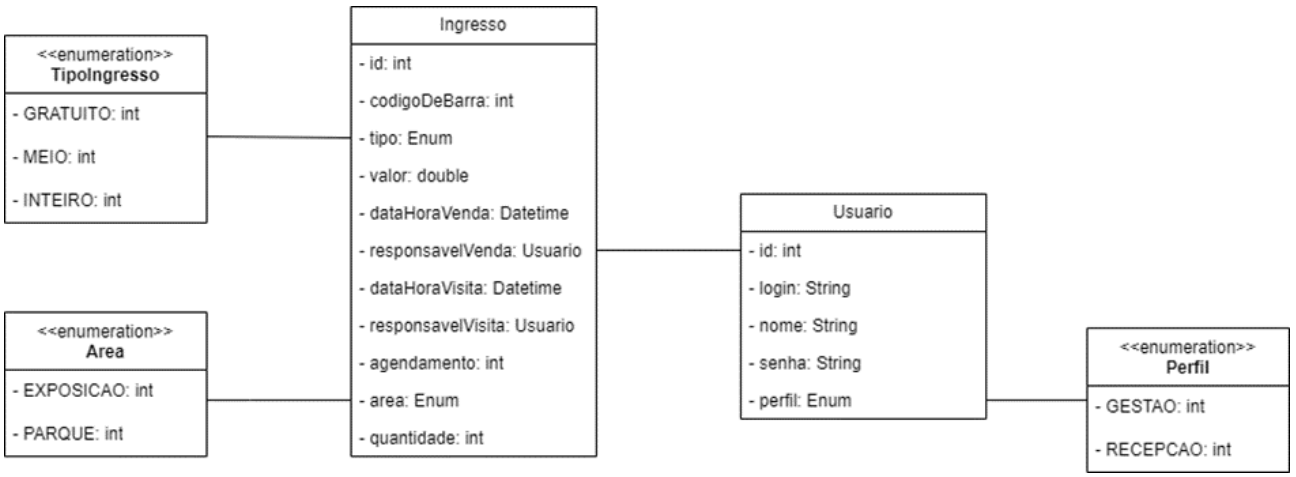
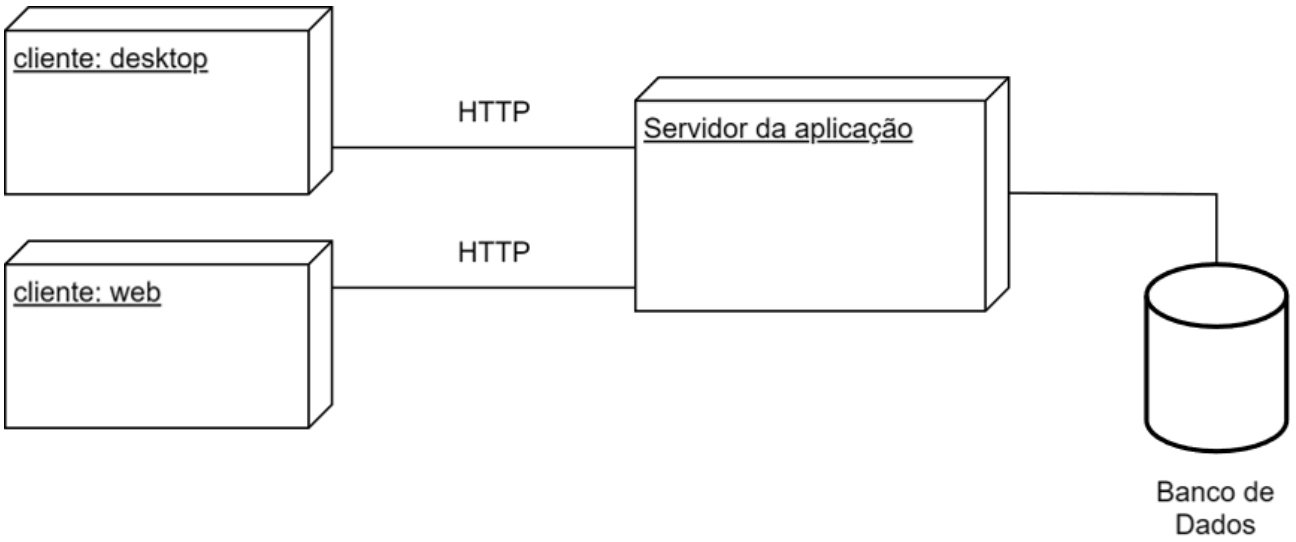


Figura 9: Diagrama de casos de uso da primeira versão do sistema Publicus



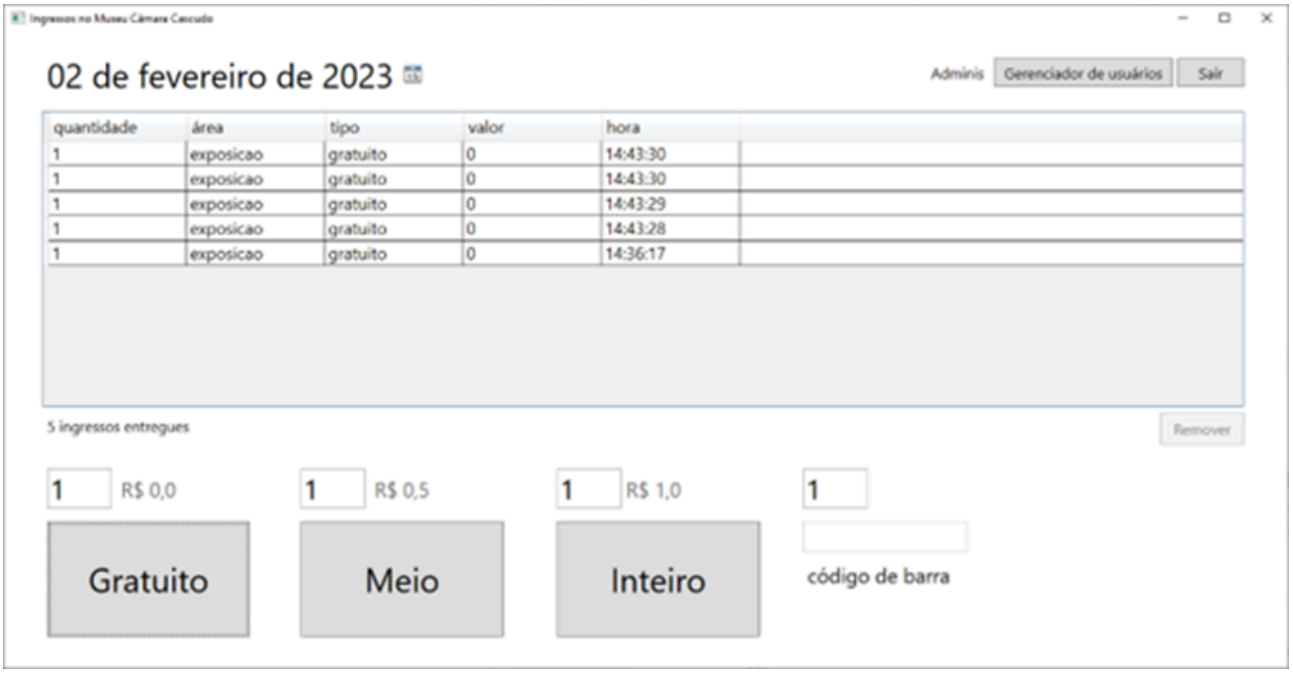
Com relação a arquitetura do Publicus original, o sistema foi estruturado no modelo cliente-servidor, com uma parte cliente específica para plataforma web e outra parte específica para o desktop. Para o desktop, foi desenvolvida uma aplicação WPF (*Windows Presentation Foudation*) com a linguagem de programação C#. Para a web, foi implementada uma aplicação em PHP. Ambas aplicações se comunicam diretamente com a parte servidor que faz acesso a um único banco de dados MariaDB. O servidor web também foi desenvolvido com a linguagem de programação PHP. O diagrama de implantação do sistema Publicus encontra-se na Figura 10.

Figura 10: Diagrama de implantação da primeira versão do software Publicus



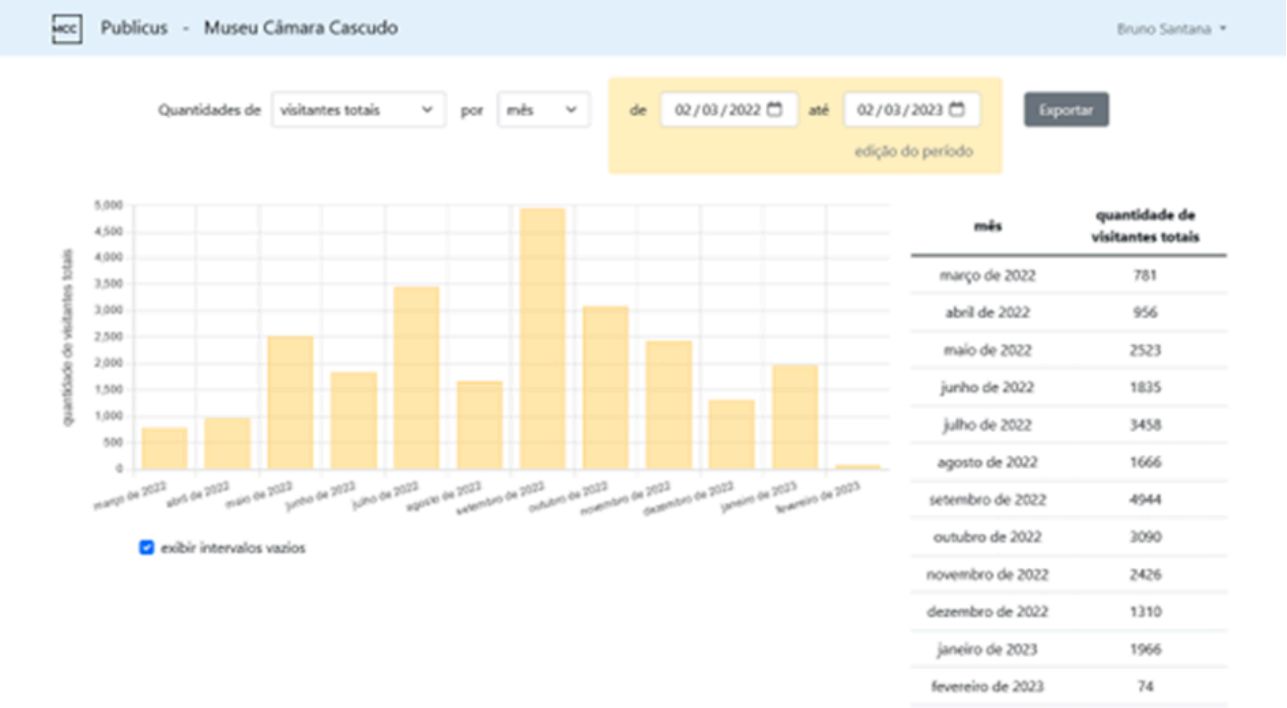
O módulo para coleta de dados da contagem de público possui as seguintes funcionalidades: (1) registrar visitantes de um dia, (2) remover registro de visitantes de um dia, (3) alterar data para registro de visitantes, além das funcionalidades para (4) gerir usuários. A Figura 11 ilustra a tela principal deste módulo.

Figura 11: Tela de cadastro de visitas no módulo de contagem de público do software Publicus



Além dessa funcionalidade, nesse segundo módulo é possível fazer a exportação dos dados de visitação para um arquivo CSV, assim como a gestão de usuários.

Figura 12: Tela principal do módulo de análise de dados de visitação no software Publicus



3.2 SEGUNDA VERSÃO DO SISTEMA PUBLICUS

Como mencionado anteriormente, uma nova versão do sistema Publicus está sendo desenvolvida para auxiliar na gestão de agendamentos de visitas escolares. Quando estiver pronta, a segunda versão substituirá por completo a versão original. Essa nova versão começou a ser projetada por Victor Paiva (Paiva, 2023; Silva; Paiva, 2022; 2023; Paiva; Silva, 2024). Ele investigou as necessidades dos usuários e se concentrou no projeto de interface da versão para smartphone do responsável.

Ele elaborou três personas (Cooper et al., 2007), uma para cada papel de usuário desta nova versão do Publicus: responsável, pedagogo e recepcionista. A primeira persona é o responsável Marcelo Silva (Figura 13). Ele representa um professor que é responsável por marcar atividades extraclasse com as suas turmas e utilizará o Publicus para solicitar agendamentos de visitas escolares para a instituição que ele trabalha.

Figura 13: Persona responsável, que agenda visitas escolares em museus

Fonte: Retirado de (Paiva, 2023).

A segunda persona, Camila Dutra (Figura 15), representa uma recepcionista do museu. Ela é encarregada de atender as visitas do museu em que trabalha e utilizará o Publicus principalmente para realizar o registro das visitas do museu.

Figura 14: Persona recepcionista, que recepciona as visitas escolares em museus

Fonte: Retirado de (Paiva, 2023).

Já a última persona elaborada foi Lais Alves (Figura 14). Laís é a pedagoga do museu que é responsável por gerir o setor educacional e promover atividades educativas no museu.

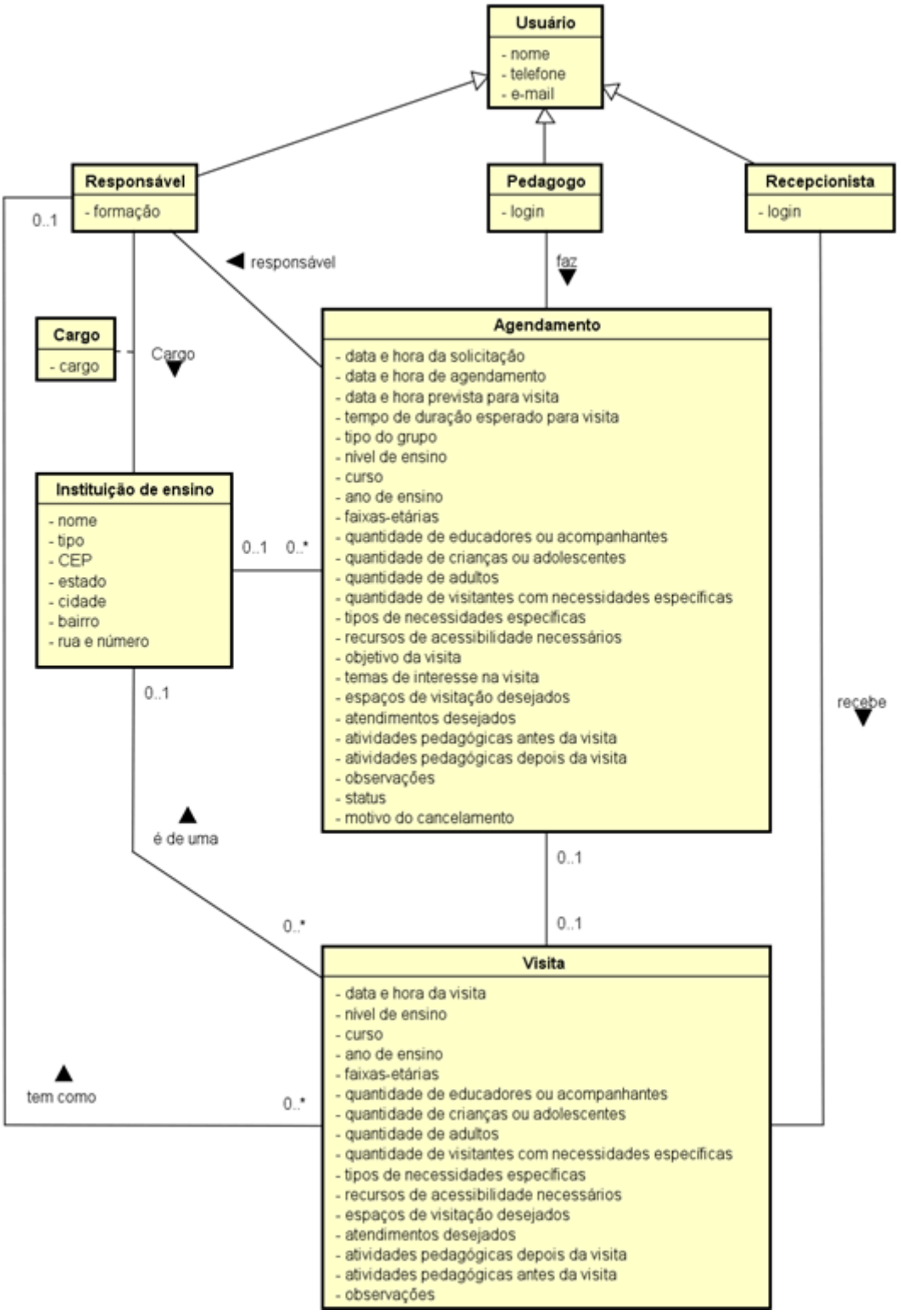
Figura 15: Persona pedagoga, que realiza a gestão de visitas escolares em museus

Fonte: Retirado de (Paiva, 2023).

Além da elaboração das personas, o estudo de Paiva possibilitou uma visão mais abrangente e diversificada sobre agendamento de visitas escolares em museus. Ele também elaborou o diagrama de classes (Wazlawick, 2015) ilustrado na Figura 16.

Foram definidas oito entidades para o sistema: usuário, responsável, pedagogo, recepcionista, agendamento, visita, cargo e instituição de ensino. Um usuário possui três atributos: nome, telefone e e-mail. No caso dele ser pedagogo ou recepcionista, ele provavelmente terá um contato muito frequente com o sistema. Por isso, ele pode possuir também um login para facilitar o acesso.

Figura 16: Diagrama de classes da segunda versão do software Publicus



Fonte: Retirado de (Paiva, 2023).

Já para o caso dele ser um responsável, ele possuiria uma formação acadêmica e cargos nas instituições de ensino em que trabalha. Ademais, esse usuário pode ser

responsável por diversos agendamentos e visitas. Uma instituição de ensino foi modelada com sete atributos: nome, tipo (público ou privada), CEP, estado, cidade, bairro, rua e número. Ela pode possuir diversos agendamentos e visitas.

Um agendamento possui um total de 24 atributos: data e horário da solicitação do agendamento, data e horário que o agendamento foi realizado, data e horário previsto para a visita, duração estimada da visita, tipo do grupo (escolar, lazer, etc.), nível de ensino, curso, ano de ensino, faixas etárias, quantidade de acompanhantes, quantidades de crianças ou adolescentes, quantidade de adultos, quantidade de visitantes com necessidades específicas, recursos de acessibilidade necessários, objetivo da visita, temas de interesse da visita, espaços de visitação desejados, atendimentos desejados, atividades pedagógicas antes da visita, atividades pedagógicas após a visita, observações, status (solicitado, agendado, negado, reagendamento solicitado, reagendamento negado, reagendado, cancelado pelo pedagogo, cancelado pelo responsável) e o motivo do status.

Agendamentos e visitas podem estar ou não relacionados. Um agendamento pode estar associado a no máximo uma visita e vice-versa. Assim, é possível haver um agendamento sem visita posterior, bem como uma visita que não tenha sido previamente agendada.

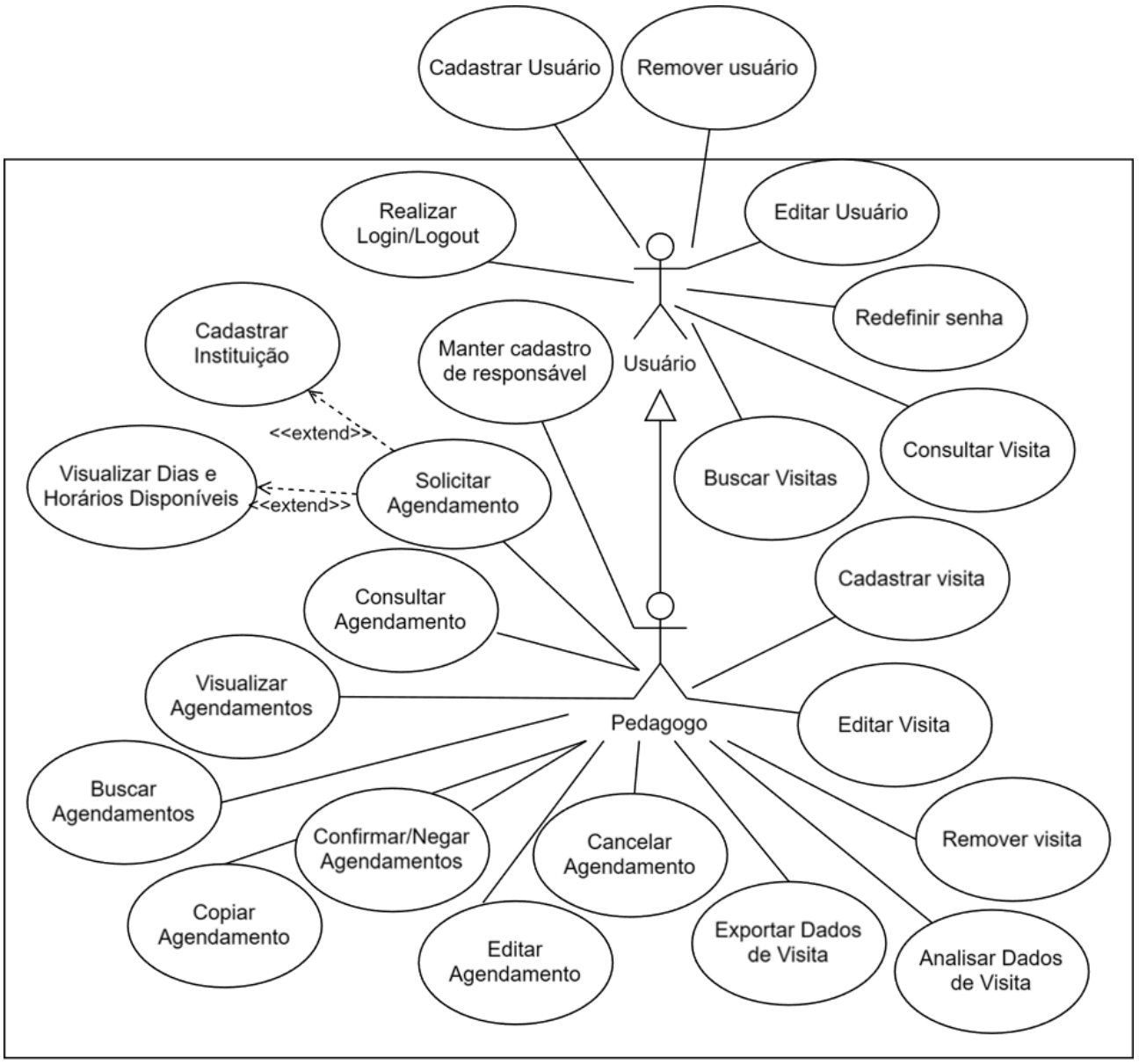
Uma visita foi modelada com 16 atributos: data e horário da visita, nível de ensino, curso, ano de ensino, faixas etárias, quantidade de acompanhantes, quantidades de crianças ou adolescentes, quantidade de adultos, quantidade de visitantes com necessidades específicas, recursos de acessibilidade necessários, objetivo da visita, temas de interesse da visita, espaços de visitação desejados, atendimentos desejados, atividades pedagógicas antes da visita, atividades pedagógicas após a visita e observações.

A partir do projeto da segunda versão do Publicus realizado pelo Paiva (2023), o professor Bruno Santana e os estudantes Danrley Lima e Ian Barreto, autor deste trabalho, elaboraram o diagrama de casos de uso da segunda versão do sistema (Figura 17). Foi definido que todos os usuários do Publicus podem: cadastrar uma nova conta (apenas para o papel de responsável), realizar login e logout, editar os seus dados, redefinir sua senha e remover sua conta.

Para o papel de responsável pela visita, foi estabelecido que esse papel pode: cadastrar uma nova instituição, consultar dias e horários disponíveis para visita, solicitar

um agendamento, visualizar resposta do agendamento, solicitar reagendamento, cancelar um agendamento e visualizar seus agendamentos.

Figura 17: Diagrama de Casos de Uso da segunda versão do software Publicus



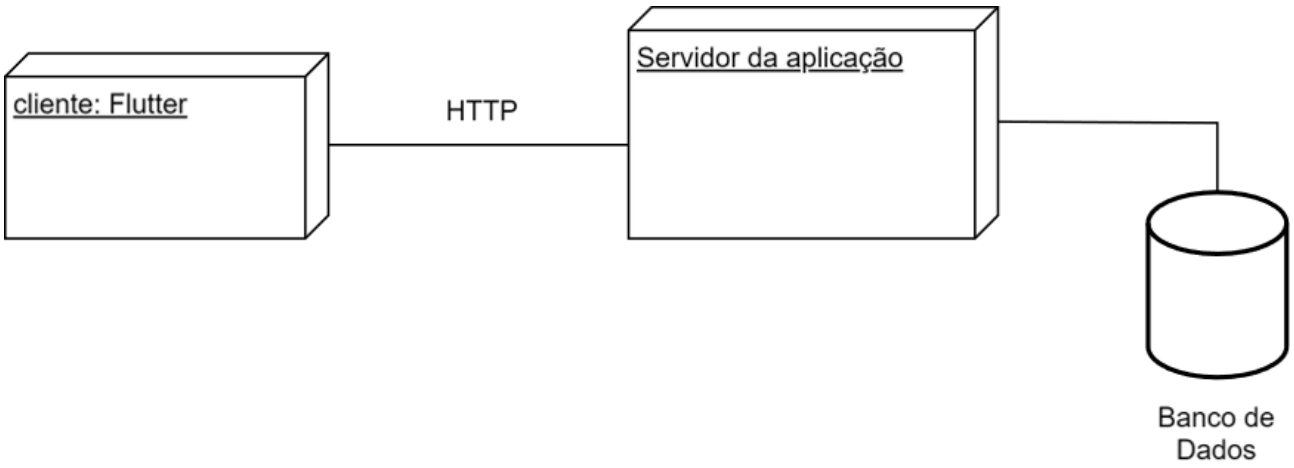
Já para o papel de pedagogo, foram determinados os seguintes casos de uso: manter disponibilidade na agenda, visualizar e responder solicitações de agendamento, consultar as necessidades específicas de um grupo, criar, editar e cancelar um agendamento e visualizar todos agendamento do museu.

Inicialmente, o papel de recepcionista foi definido com os casos de uso: visualizar agendamentos, solicitar agendamento, cadastrar instituição e consultar horários disponíveis. Por último, foi definido um papel de administrador para manter o cadastro de pedagogo e recepcionista.

Para a arquitetura da segunda versão do Publicus foi escolhido o estilo arquitetural cliente-servidor. Este servidor da aplicação (Figura 18) é diferente do que foi desenvolvido na primeira versão (Figura 10), por ter sido construído como uma API para favorecer a interoperabilidade e a modularidade do sistema. Este novo servidor também foi implementado na linguagem de programação PHP, utilizando MariaDB como sistema de gerenciamento de banco de dados. Visto a necessidade do Publicus ser utilizado em diversas plataformas, foi determinado o desenvolvimento da aplicação utilizando o framework Flutter com a linguagem Dart. A escolha dessa ferramenta ocorreu devido a possibilidade da construção de sistemas multiplataformas (dispositivos móveis, desktop e web) a partir de uma única base de código. Deste modo, espera-se economizar esforços no desenvolvimento e na manutenção do sistema nas diferentes plataformas. Na Figura 18 temos o diagrama de implantação da segunda versão do software Publicus

.

Figura 18: Diagrama de Implantação da segunda versão do Publicus

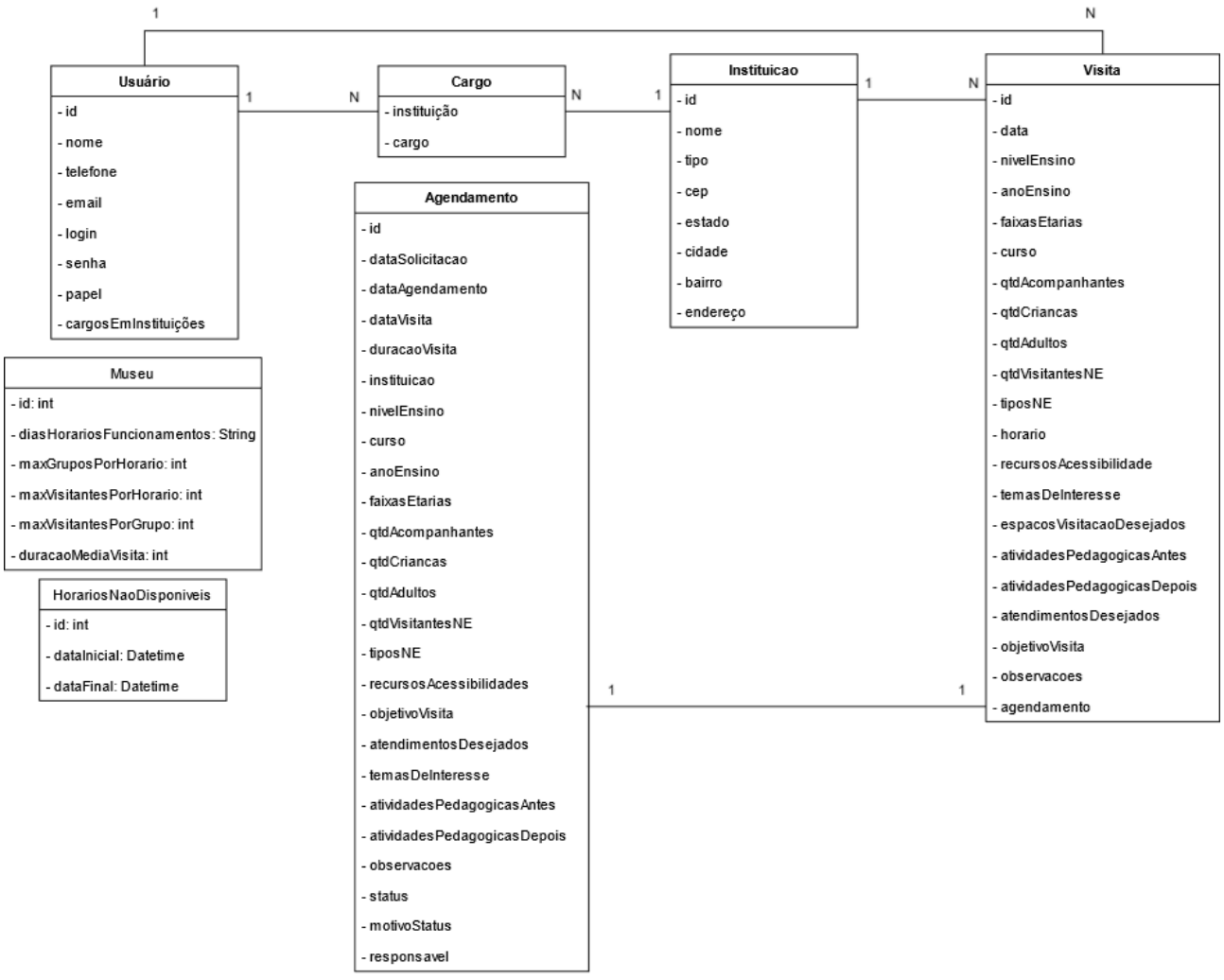


Durante o desenvolvimento da aplicação, percebeu-se a necessidade de mudanças no modelo de dados do Publicus proposto originalmente pelo Victor Paiva (2023). Portanto, foi definida uma nova entidade Museu para registrar as configurações específicas de um museu, a partir dos seguintes atributos: dias de funcionamento, horário

de funcionamento, quantidade máxima de grupos por visita e duração média de visita. Além disso, decidiu-se implementar os diferentes papéis de usuário por meio de um atributo na classe Usuário, que pode assumir os valores: responsável, pedagogo, recepcionista e administrador.

Como mencionado previamente, durante a execução deste trabalho a segunda versão do Publicus ainda estava em desenvolvimento. Apenas o módulo do perfil responsável para smartphone estava concluído, com a colaboração do autor deste trabalho. Já o módulo para agendamentos estava sendo implementado e o módulo para a gestão de visitas ainda não tinha sido contemplado. A Figura 19 apresenta o diagrama de classes da segunda versão do Publicus antes do início deste trabalho.

Figura 19: Diagrama de dados atualizado da segunda versão do Publicus



4 DESENVOLVIMENTO DO MÓDULO DE VISITAS DO SOFTWARE PUBLICUS

Nesta seção, inicia-se a contribuição deste trabalho com a evolução do Software Publicus, desenvolvendo o módulo para a gestão de visitas a museus da segunda versão. Este novo módulo propõe uma solução única que visa unir a gestão de visitas avulsas presente na primeira versão do Publicus com a gestão de agendamentos e visitas escolares, proposta apenas na nova versão. Deste modo, a segunda versão poderá substituir a original por completo e oferecer novas funcionalidades.

A gestão de visitas recebidas pelo museu geralmente é de responsabilidade dos recepcionistas. Entretanto, eventualmente os pedagogos podem atuar na recepção do museu em situações excepcionais, como um evento específico ou substituindo um funcionário que faltou. Deste modo, tanto usuário de perfil recepcionista quanto de perfil de pedagogo deveriam ter autorização para utilizar o módulo de gestão de visitas.

Para o ciclo de vida do software, foi adotado o processo de desenvolvimento em Cascata. Primeiramente, foi definido o escopo do módulo visitas no sistema, atualizado o modelo de dados e levantados os casos de uso a serem implementados. Para tanto, considerou-se o projeto de interface elaborado por Antônio Medeiros, Herus Costa e Bruno Santana. Depois, foi iniciada a etapa de implementação do sistema, mantendo a arquitetura da segunda versão do Publicus (Figura 18) com a parte cliente direcionada a smartphone e desktop. Por último, foi realizado o planejamento e a implementação dos testes automatizados. A seguir, serão apresentados os casos de usos desenvolvidos, o modelo de dados do sistema e a programação da parte cliente e da parte servidor do módulo de visitas. É importante notar que este trabalho precisou seguir as escolhas tecnológicas (linguagens, frameworks, banco de dados, etc.) de implementação realizadas por terceiros para dar continuidade ao desenvolvimento da segunda edição do Publicus, por isso as justificativas dessas decisões estão fora do escopo deste trabalho.

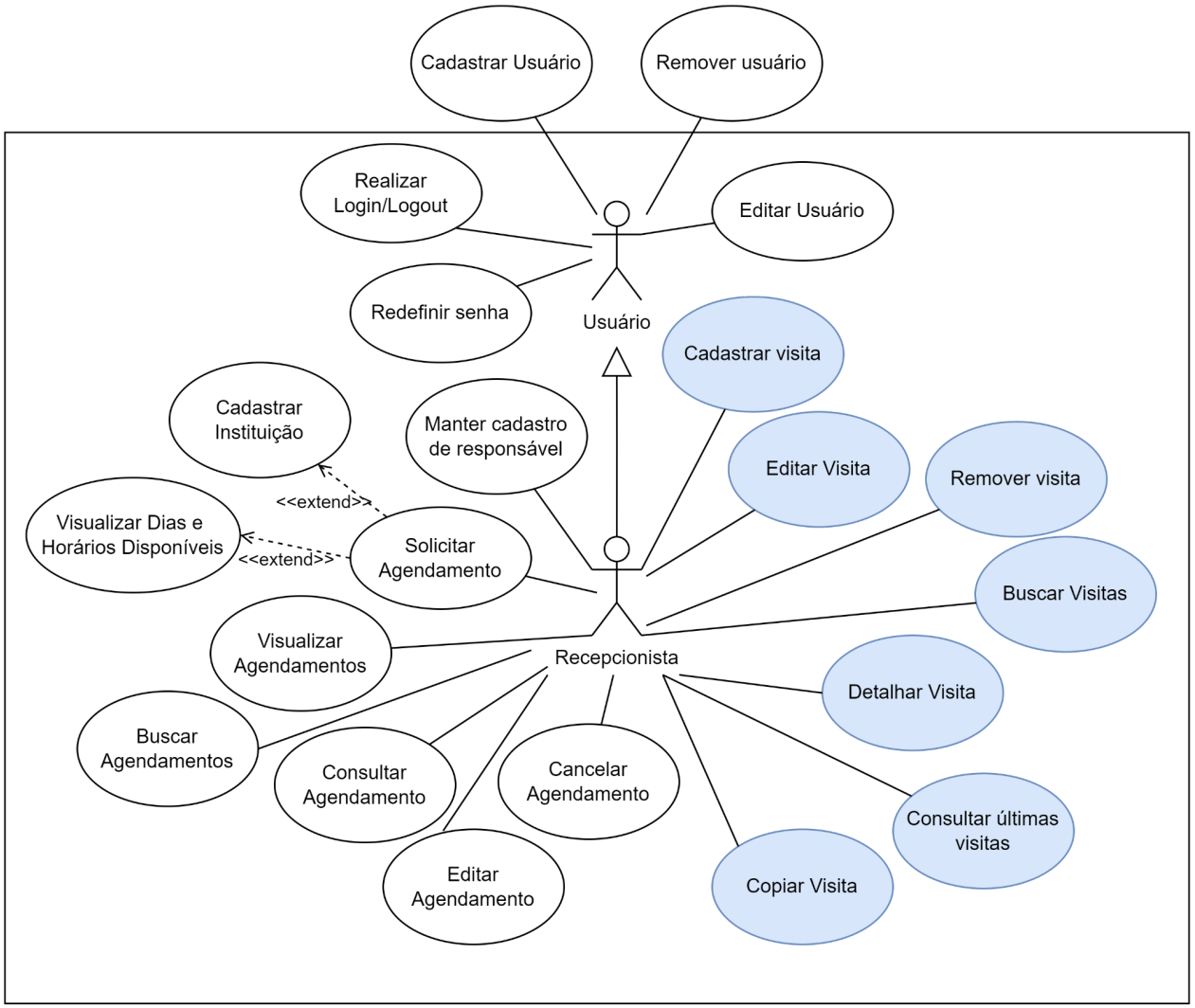
4.1 CASOS DE USO

Para isso, tomou-se como base o diagrama de casos de uso da Figura 17, elaborado para a segunda versão do Publicus. Ele indica que o recepcionista apenas solicita agendamento, consulta horários disponíveis, cadastra instituição e visualiza agendamentos. Neste trabalho, foi elaborada uma evolução deste diagrama, que agora

também contempla os casos de uso “Detalhar Visita” e “Copiar Visita” conforme apresentado na Figura 20.

Os casos de uso representados em branco na Figura 20 foram aproveitados da implementação do módulo de agendamentos do sistema Publicus realizado por terceiros. Esta implementação anterior contempla tanto o *frontend* quanto o *backend.* Já os casos de uso representados em azul na Figura 20 foram elicitados e desenvolvidos neste trabalho. Assim, o perfil de recepcionista também será responsável por cadastrar, editar, remover, buscar, detalhar, copiar e consultar as últimas visitas realizadas.

Figura 20: Diagrama de casos de uso do software Publicus



A partir da elaboração deste diagrama, a descrição textual de cada caso de uso do módulo de visitas foi realizada neste trabalho, pois o projeto do Publicus ainda não

contava com esta documentação. Este detalhamento dos casos de uso considerou o projeto da interface com usuário elaborado por Antônio, Herus e Bruno. Este projeto também orientou o desenvolvimento do *frontend* e *backend* do software Publicus.

No Quadro 1, podemos observar a descrição textual para o caso de uso “buscar visitas”. Um usuário pode realizar a busca de visitas a partir do tipo de grupo, instituição, data inicial, data final e se há ou não visitantes com necessidades específicas. A Figura 21 ilustra a tela de busca para desktop e a Figura 22 a tela equivalente para smartphone.

Quadro 1: Descrição textual do caso de uso buscar visita

CSU01 Buscar Visitas

Descrição: Caso de uso responsável por recuperar visitas cadastradas no Publicus, de acordo com o nome da instituição, status, data inicial e data final da visita e necessidade específica dos visitantes fornecidos pelo usuário.

Pré Condições: Estar autenticado no sistema

Ator: Recepcionista e responsável

Cenário Principal de Sucesso:

1. O usuário acessa a aba de “visitas” do sistema;
2. O sistema exibe todas as visitas no Publicus realizadas no período de 365 dias antes da data atual, independente do nome da instituição e de necessidades específicas dos visitantes, até o limite da quantidade de visitas indicadas na configuração (definição da constante de quantidade de itens buscados);
3. O usuário escolhe o formato em que ele deseja visualizar as visitas (detalhe, tabela ou calendário) e informa os parâmetros de busca desejado (nome da instituição, data inicial, data final e necessidade específica dos visitantes);
4. O sistema exibe todas as visitas encontradas de acordo com os parâmetros de busca definidos pelo usuário, no formato escolhido pelo usuário.

Fluxo Alternativo (2) Usuário logado é responsável:

a) O sistema exibe as visitas cadastradas no Publicus para o responsável logado, realizadas no período de 365 dias antes da data atual, independente do nome da instituição e de necessidades específicas dos visitantes, até o limite da quantidade de visitas indicadas na configuração (definição da constante de quantidade de itens buscados);

Fluxo Alternativo (4) Usuário logado é responsável:

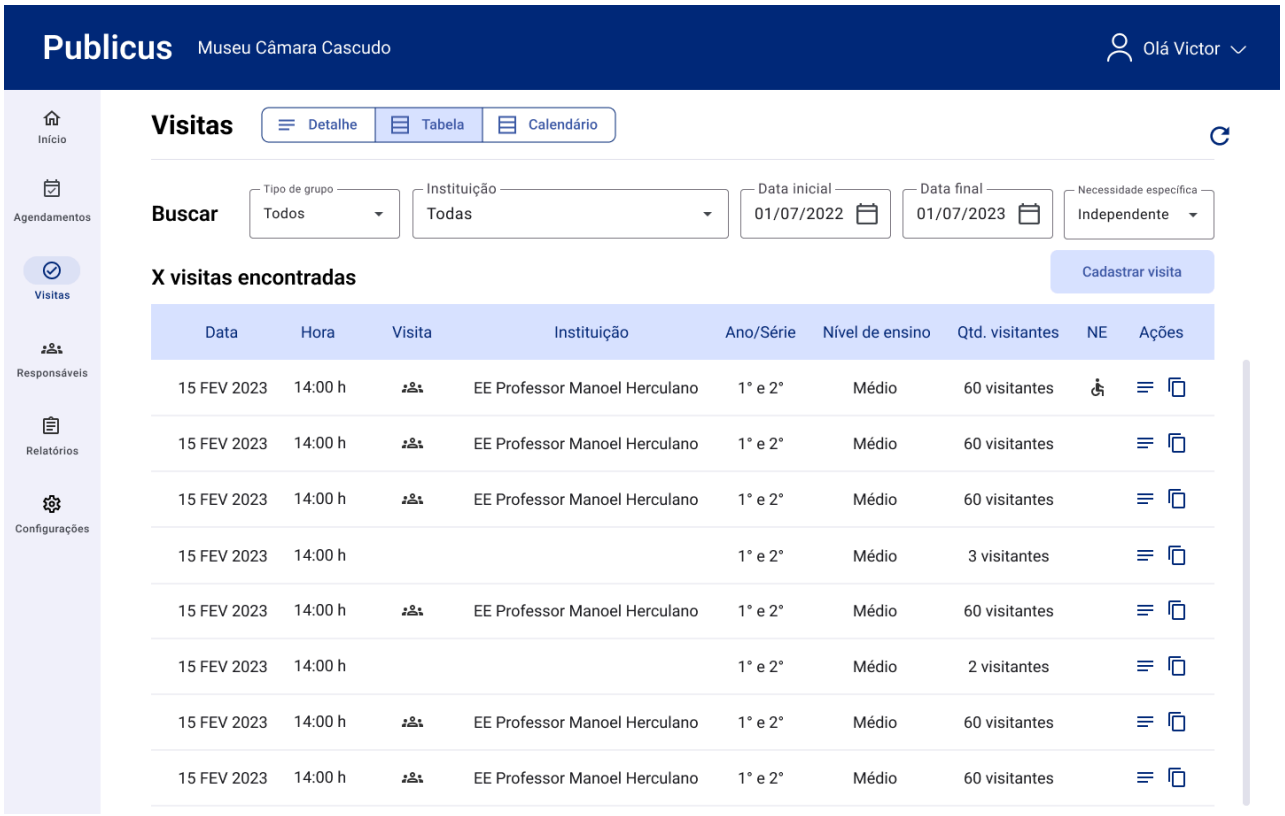
a) O sistema exibe as visitas cadastradas para o usuário logado, encontradas de acordo com os parâmetros de busca definidos pelo usuário, no formato escolhido pelo usuário.

Extensão (2 e 4) Recuperação de visitas sob demanda:

a) O usuário passa do limite de visualização das visitas em memória no resultado da busca;

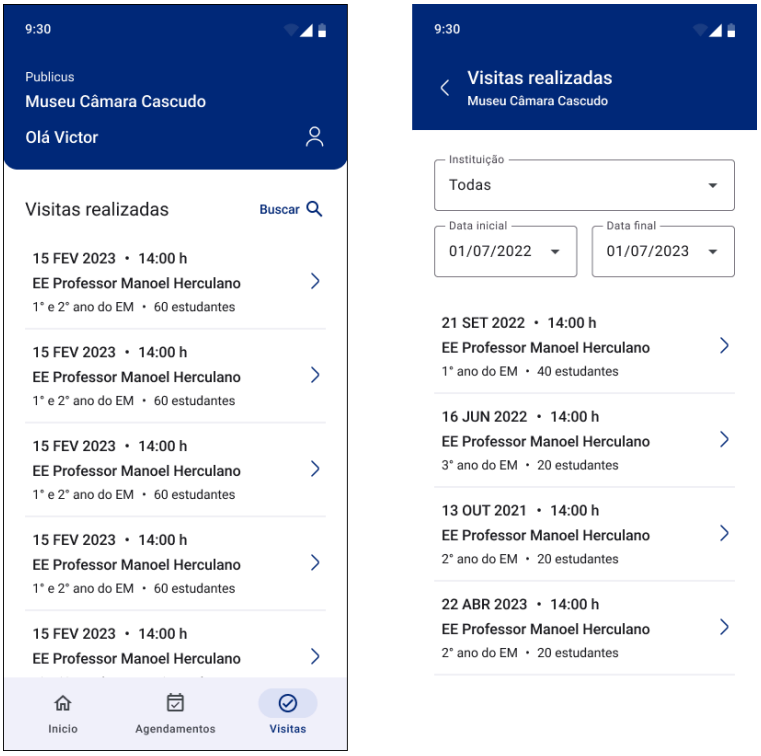
b) O sistema carrega para memória e exibe as próximas visitas presentes no resultado de busca, incrementando o offset com a quantidade de visitas indicadas na configuração (definição da constante de quantidade de itens buscados).

Figura 21: Protótipo da tela para o caso de uso buscar visitas para desktop



Fonte: Projeto de interface do Antônio, Herus e Bruno.

Figura 22: Protótipo da tela para o caso de uso buscar visitas para smartphone



Fonte: Retirado de (Paiva, 2023).

No Quadro 2 encontramos a descrição textual do caso de uso para detalhar uma visita. Já na Figura 23 podemos observar o protótipo da interface de usuário deste caso de uso tanto para smartphone (esquerda), quanto para desktop (direita).

Quadro 2: Descrição textual do caso de uso detalhar visita

CSU02 Detalhar Visita

Descrição: Caso de uso responsável por exibir os detalhes de uma visita cadastrada no Publicus.

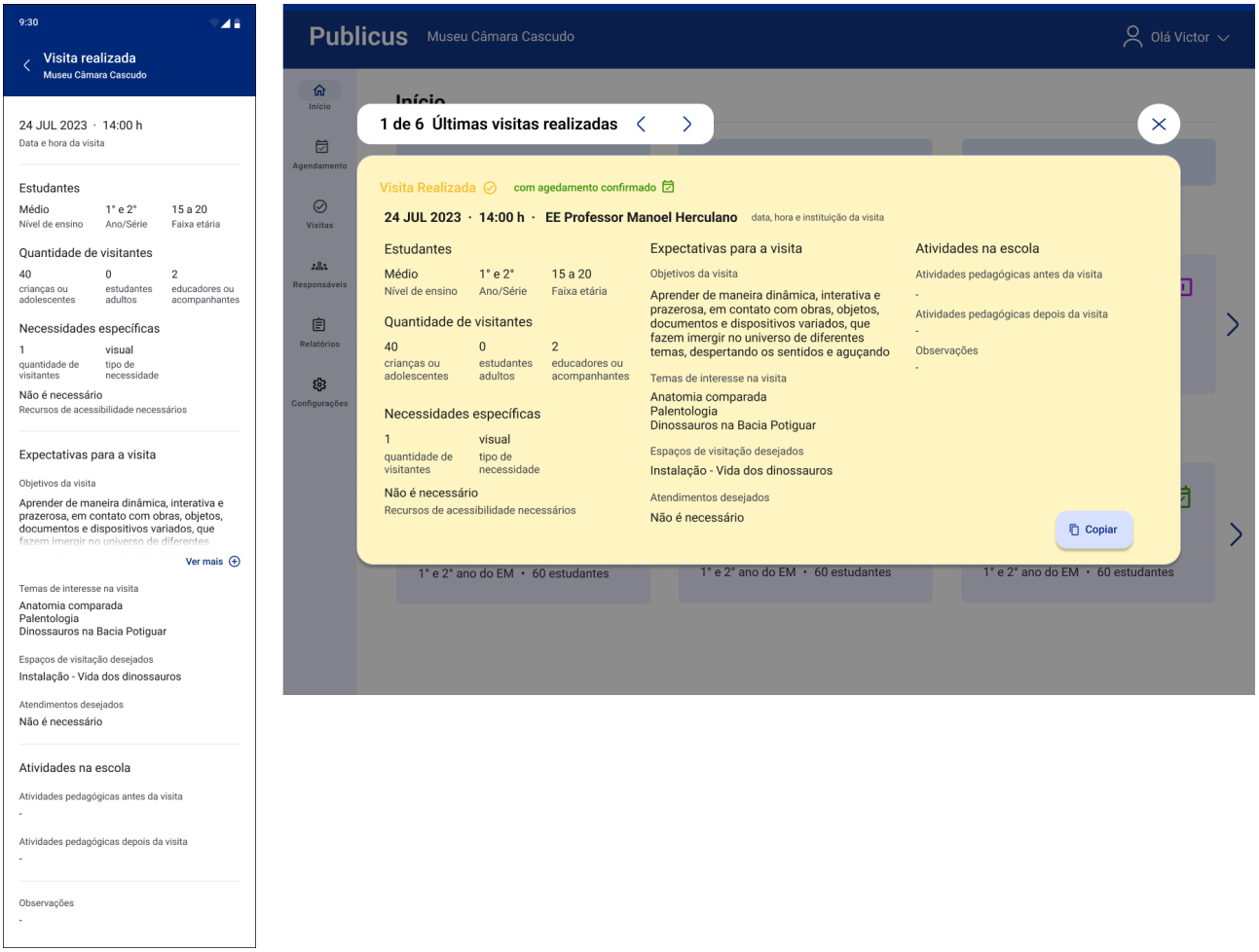
Pré Condições: Estar autenticado no sistema

Ator: Recepcionista e responsável

Cenário Principal de Sucesso:

1. O usuário seleciona uma visita em um conjunto de visitas para ver mais detalhes;
2. O sistema exibe todos os dados da visita selecionada: data, nível de ensino, ano de ensino, faixas etárias, quantidade de acompanhantes, quantidade de crianças e adolescentes, quantidade de adultos, quantidade de visitantes com necessidades específicas, às necessidades específicas, horário, recursos de acessibilidade, temas de interesse, espaços de visitação desejado, atividades pedagógicas, atendimentos desejados, observações e a instituição responsável pela visita;
3. Se o usuário avança ou retrocede, a consulta de detalhes de outra visita no conjunto em questão continua no passo 2.

Figura 23: Protótipo da tela para o caso de uso detalhar visitas para smartphone (esquerda) e destop (direita)



Fonte: Projeto de interface do Victor (esquerda), Antônio, Herus e Bruno (direita).

Podemos encontrar a descrição textual do caso de uso remover visita no Quadro 3. Um dos caminhos para o usuário conseguir remover uma visita é clicando no ícone da lixeira na coluna “Ações” da tabela de visitas, que pode ser observada na Figura 21.

Quadro 3: Descrição textual do caso de uso remover visita

CSU03 Remover Visita
Descrição: Caso de uso responsável por remover uma visita cadastrada no Publicus.
Pré Condições: Estar autenticado no sistema
Ator: Recepcionista

Cenário Principal de Sucesso:

1. O sistema exibe a tela de consulta de uma visita;
2. O usuário clica no botão “Remover visita”;
3. O sistema solicita confirmação do usuário;
4. O usuário confirma a remoção;
5. O sistema remove a visita indicada e exibe os detalhes da próxima visita do conjunto de visitas em questão

Fluxo alternativo (4) O usuário cancela a remoção:

a) O sistema fecha o modal de confirmação e encerra o fluxo principal.

Fluxo alternativo (5) O conjunto de visitas ficou vazio sem modal aberto:

a) O sistema exibe o conjunto de visitas vazio.

Fluxo alternativo (5) O conjunto de visitas ficou vazio com modal aberto:

a) O sistema fecha o modal e exibe o conjunto de visitas vazio.

O caso de uso consultar últimas visitas foi descrito no Quadro 4, já na Figura 24 podemos observar o protótipo para a tela inicial contendo o carrossel, mencionado na descrição textual, que exibe as últimas visitas do museu.

Quadro 4: Descrição textual do caso de uso consultar últimas visitas

CSU04 Consultar últimas visitas

Descrição: Caso de uso para exibir as últimas visitas do usuário logado, caso seja do perfil responsável, ou de todos os usuários, caso o usuário logado seja um funcionário do museu.

Pré Condições: Estar autenticado no sistema

Ator: Recepcionista e responsável

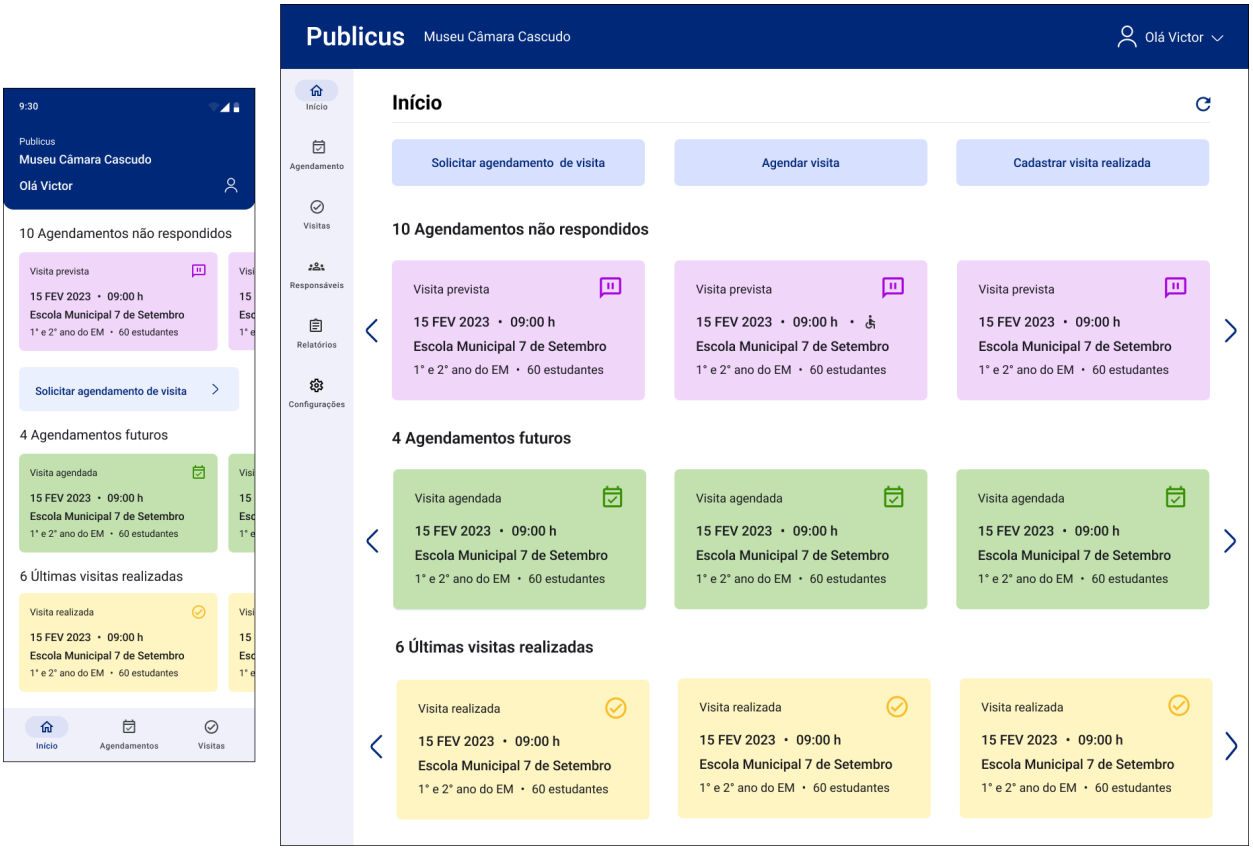
Cenário Principal de Sucesso:

1. O usuário acessa a aba “Início” do Publicus;
2. O sistema exibe as últimas visitas recebidas pelo museu em um carrossel de cartões, independente de quem realizou a visita.

Fluxo Alternativo (2) O usuário logado é um responsável:

a) O sistema exibe as últimas visitas realizadas pelo usuário logado no museu, em um carrossel de cartões.

Figura 24: Protótipo da tela para o caso de uso consultar últimas visitas para smartphone (esquerda) e destop (direita)



Fonte: Projeto de interface do Vitor (esquerda), Antônio, Herus e Bruno (direita).

No sistema Publicus é possível realizar o cadastro de visita de cinco formas diferentes: cadastrar uma visita avulsa (Figura 25), cadastrar uma visita em grupo com agendamento associado (Figura 26), cadastrar uma visita em grupo sem agendamento associado (Figura 27), cadastrar uma visita em grupo a partir de um responsável e cadastrar uma visita em grupo a partir de um agendamento. Essas diferentes maneiras de inserir uma nova visita foram descritas no Quadro 5.

Quadro 5: Descrição textual do caso de uso cadastrar visita

CSU05 Cadastrar visita
Descrição: Caso de uso para cadastrar as visitas avulsas e em grupo no software Publicus.
Pré Condições: Estar autenticado no sistema
Ator: Recepcionista

Cenário Principal de Sucesso:

1. O usuário acessa a aba de visitas;
2. O sistema exibe uma tabela contendo as visitas cadastradas nos últimos 365 dias;
3. O usuário clica no botão “Cadastrar visita”;
4. O sistema exibe o formulário para cadastrar uma visita avulsa com a data e a hora preenchida automaticamente. Além disso, exibe um carrossel contendo as últimas visitas cadastradas pelo usuário logado;
5. O usuário fornece o número de visitantes e caso necessário informa o número de crianças e adolescentes, número de visitantes com necessidade específica, tipos de necessidade específica, recursos de acessibilidade necessários e observações;
6. O usuário clica no botão “Cadastrar visita”;
7. O sistema informa ao usuário que uma visita foi cadastrada com sucesso, esvazia os campos do formulário e adiciona a visita cadastrada ao carrossel.

Fluxo Alternativo (5) Cadastrar visita em grupo com agendamento confirmado:

a) O usuário clica no botão segmentado “Visita em grupo”;
b) O sistema exibe o formulário para cadastrar uma visita em grupo e um carrossel contendo os agendamentos confirmados do dia atual;
c) O usuário seleciona o agendamento da visita;
d) O sistema preenche o formulário automaticamente com os dados do agendamento (tipo de grupo, instituição, responsável, níveis de ensino, anos de ensino, faixas etárias, número de visitantes, número de crianças e adolescentes, número de visitantes com necessidade específica, tipos de necessidade específica, recurso de acessibilidade necessários, observações);
e) O usuário faz as alterações necessárias e clica no botão “Cadastrar visita”;
f) O sistema informa que a visita foi cadastrada com sucesso e adiciona a nova visita no carrossel de últimas visitas;

Fluxo Alternativo (5) Cadastrar visita em grupo sem agendamento confirmado:

a) O usuário clica no botão segmentado “Visita em grupo”;
b) O sistema exibe o formulário para cadastrar uma visita em grupo e um carrossel contendo os agendamentos confirmados do dia atual;
c) O usuário preenche o formulário informando o tipo de grupo, a instituição, o responsável, os níveis de ensino, os anos de ensino, as faixas etárias, o número de visitantes, o número de crianças e adolescentes, o número de visitantes com necessidade específica, os tipos de necessidade específica, os recursos de acessibilidade necessários e as observações;
d) O usuário clica no botão “Cadastrar visita”;
e) O sistema informa que a visita foi cadastrada com sucesso e adiciona a nova visita no carrossel de últimas visitas;

Fluxo Alternativo (1) Cadastrar visita em grupo a partir de um responsável:

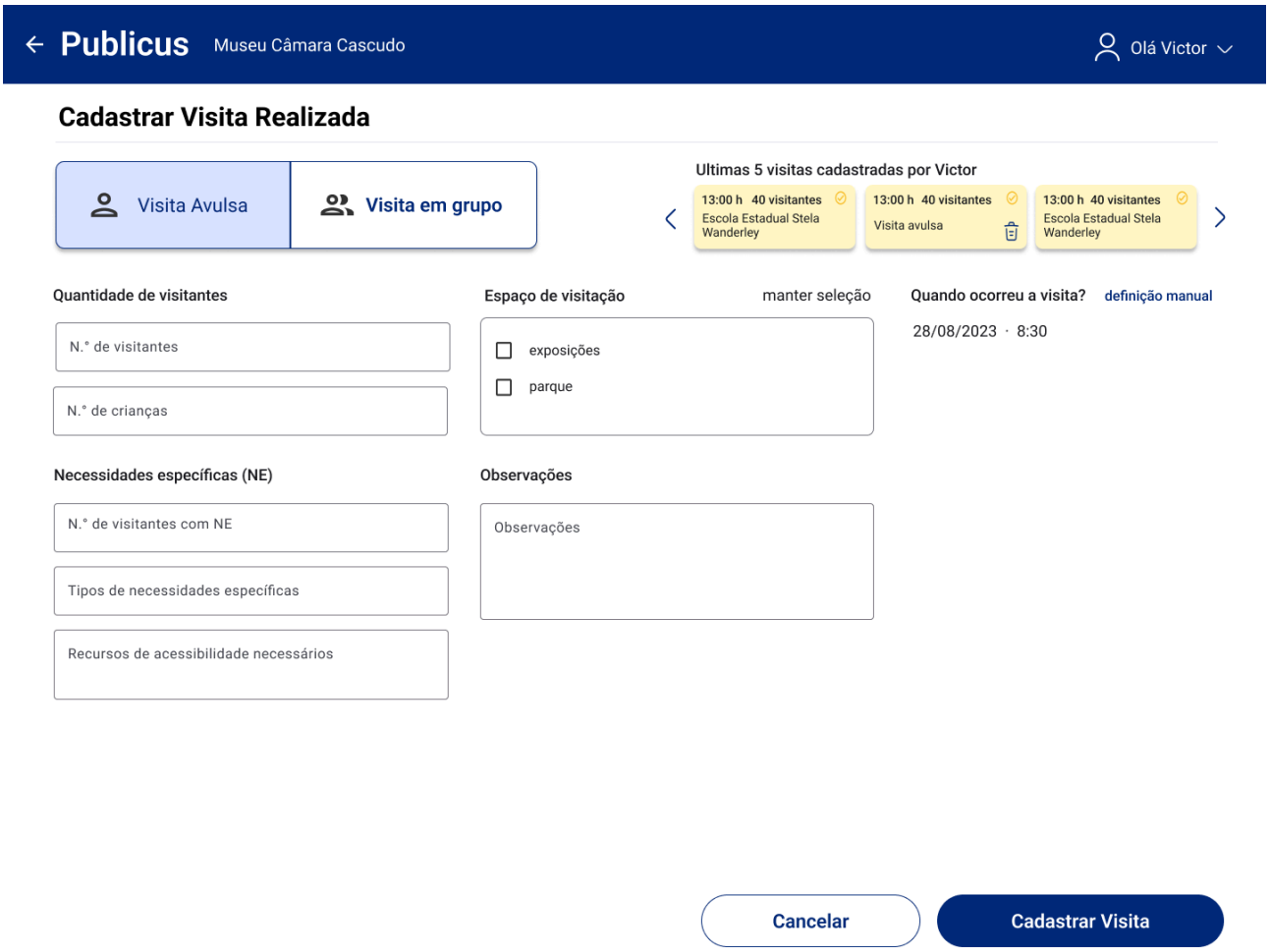
a) O usuário acessa a aba de responsáveis;
b) O sistema exibe uma tabela com todos os responsáveis cadastrados no sistema;

c) O usuário clica no ícone “Cadastrar visita para este responsável” no responsável desejado;
d) O sistema exibe o formulário para o cadastro de visita em grupo, preenchendo automaticamente o campo responsável;
e) O usuário preenche os demais campos (tipo de grupo, instituição, níveis de ensino, anos de ensino, faixas etárias, número de visitantes, número de crianças ou adolescentes, número de visitantes com necessidade específica, tipos de necessidade específica, recursos de acessibilidade necessários e observações);
f) O usuário clica no botão “Cadastrar visita”;
g) O sistema redireciona o usuário para a página de responsáveis e informa que a visita foi cadastrada com sucesso;

Fluxo Alternativo (1) Cadastrar visita em grupo a partir de um agendamento confirmado:

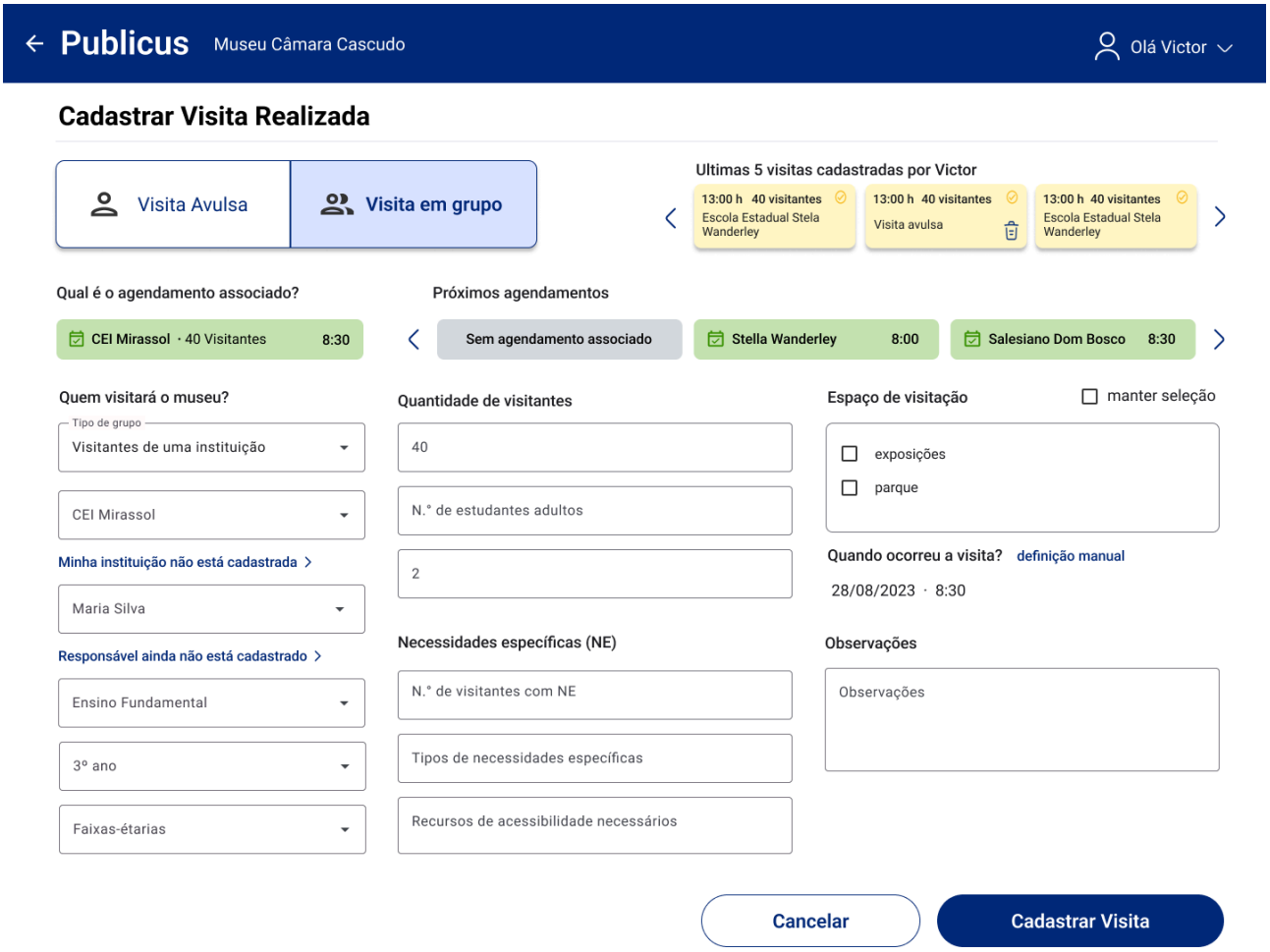
a) O usuário acessa a aba de agendamentos;
b) O sistema exibe uma tabela com os agendamentos realizados no período de 365 dias;
c) O usuário clica duas vezes no agendamento confirmado desejado;
d) O sistema exibe os detalhes do agendamento selecionado;
e) O usuário clica no botão “Cadastrar visita”;
f) O sistema exibe o formulário de cadastro de visita em grupo preenchido automaticamente com os campos do agendamento selecionado;
g) O usuário faz as alterações necessárias e clica no botão “Cadastrar visita”;
h) O sistema redireciona o usuário para a página de agendamentos e informa que uma nova visita foi cadastrada com sucesso.

Figura 25: Protótipo da interface de usuário para cadastrar uma visita avulsa



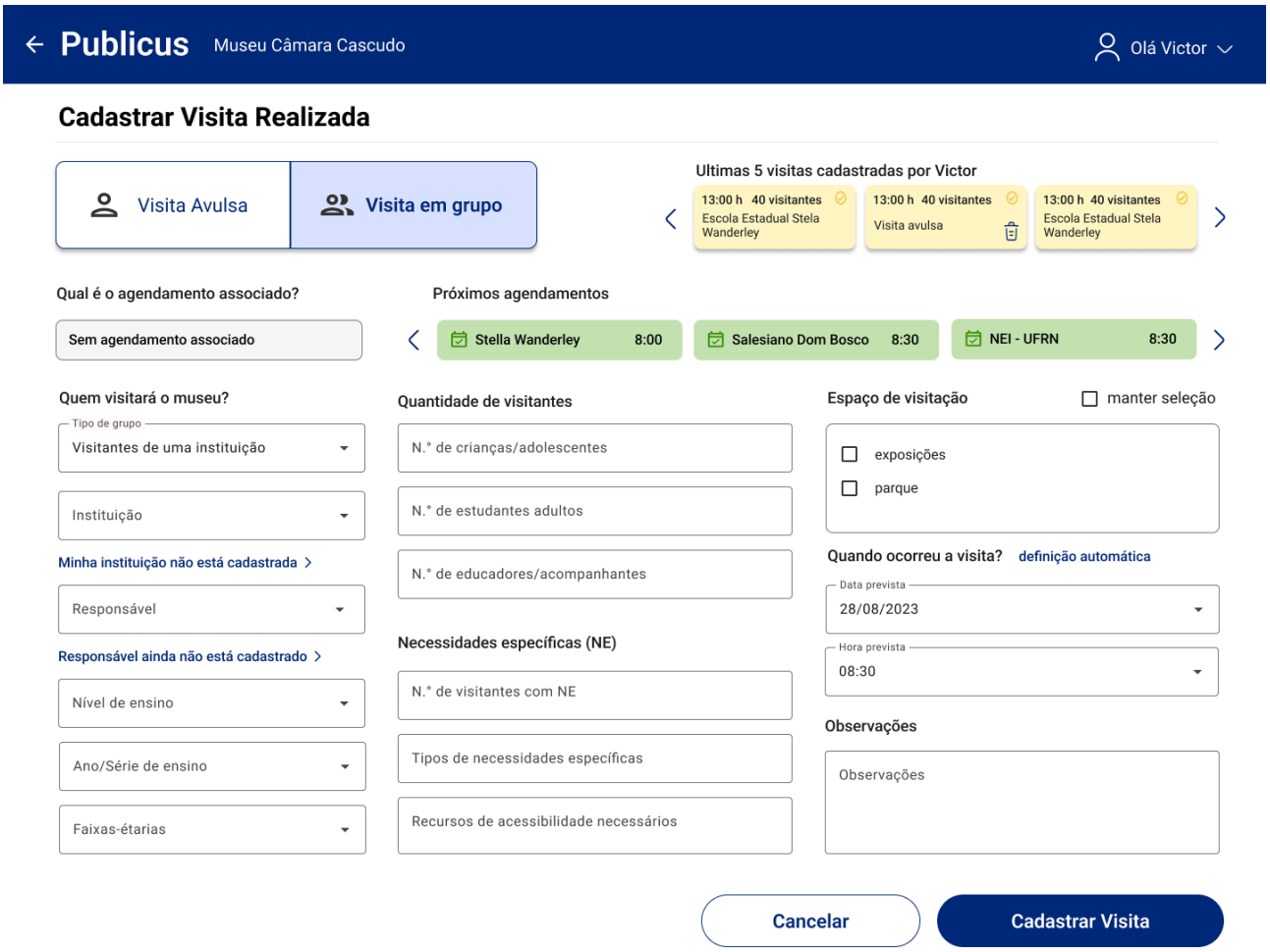
Fonte: Projeto de interface do Herus e Bruno.

Figura 26: Protótipo da interface de usuário para cadastrar visita em grupo com agendamento confirmado



Fonte: Projeto de interface do Herus e Bruno.

Figura 27: Protótipo da interface de usuário para cadastrar visita em grupo sem agendamento confirmado



Fonte: Projeto de interface do Herus e Bruno.

No Quadro 6 e Quadro 7 encontram-se as descrições textuais dos casos de uso editar visita e copiar visita, respectivamente.

Quadro 6: Descrição textual do caso de uso editar visita

CSU06 Editar Visita
Descrição: Caso de uso responsável por editar uma visita cadastrada no Publicus.
Pré Condições: Estar autenticado no sistema
Ator: Recepcionista
Cenário Principal de Sucesso: 1. O usuário acessa a aba de visitas;

2. O sistema exibe uma tabela contendo as visitas cadastradas no sistema no período de 365 dias;
3. O usuário clica no ícone de “Editar visita” na visita desejada;
4. O sistema exibe um formulário para a edição de visita preenchido automaticamente com os dados da visita selecionada;
5. O usuário faz as alterações necessárias e clica em “Editar visita”;
6. O sistema redireciona o usuário para a página de visitas e informa que uma visita foi editada com sucesso.

Quadro 7: Descrição textual do caso de uso copiar visita

CSU07 Copiar Visita

Descrição: Caso de uso responsável por copiar uma visita cadastrada no Publicus.

Pré Condições: Estar autenticado no sistema

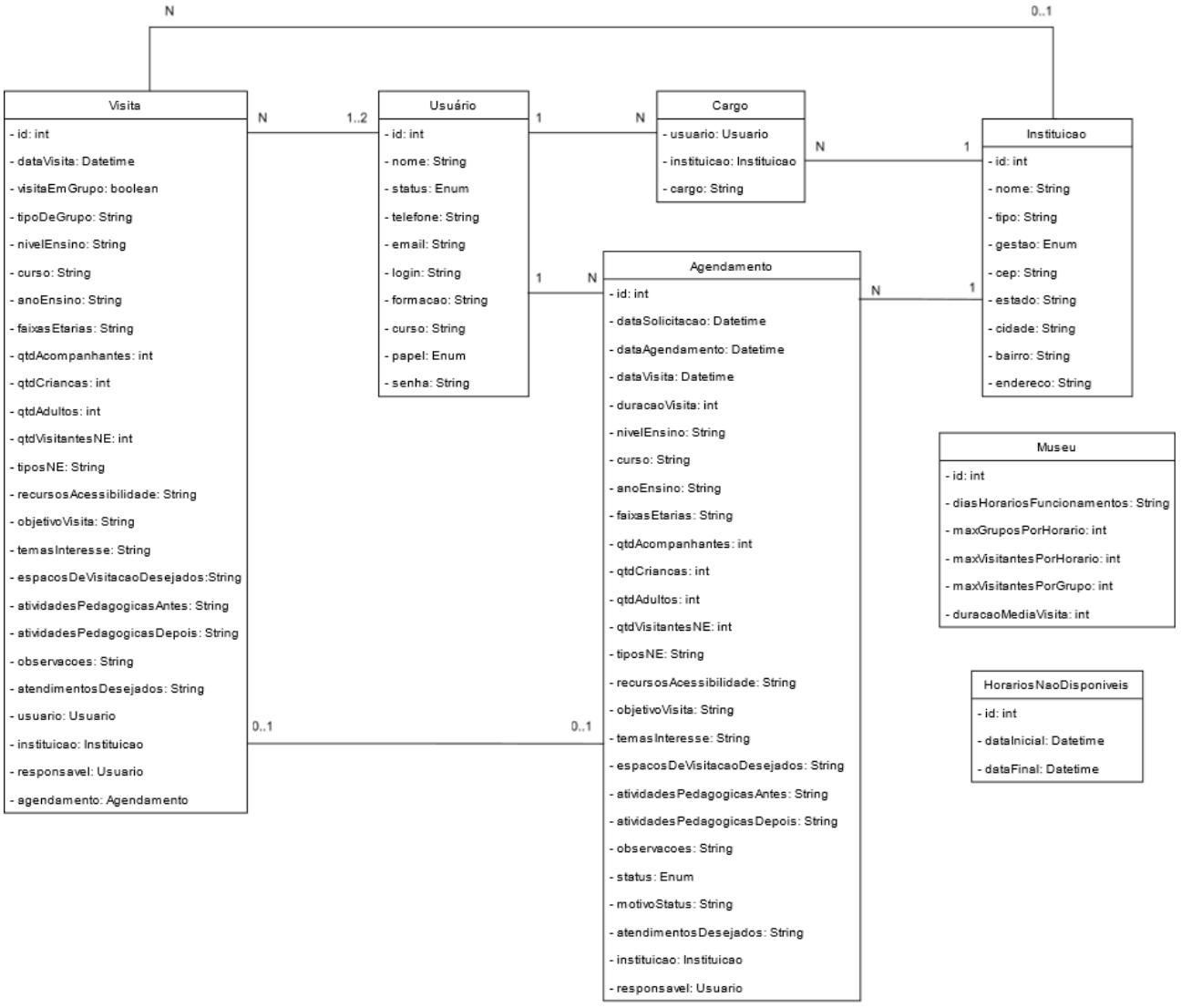
Ator: Recepcionista

Cenário Principal de Sucesso:
1. O usuário acessa a aba de visitas;
2. O sistema exibe uma tabela contendo as visitas cadastradas no sistema no período de 365 dias;
3. O usuário clica no ícone de “Copiar visita” na visita desejada;
4. O sistema exibe um formulário para o cadastro de visita preenchido automaticamente com os dados da visita selecionada;
5. O usuário faz as alterações necessárias e clica em “Cadastrar visita”;
6. O sistema redireciona o usuário para a página de visitas e informa que uma nova visita foi cadastrada com sucesso.

4.2 MODELO DE DADOS

Durante a etapa de desenvolvimento, percebeu-se a necessidade de realizar modificações no modelo de dados mostrado na Figura 19. Agora a entidade Visita possui também os campos “visitaEmGrupo” e “tipoDeGrupo”. O primeiro é responsável por identificar se uma visita é em grupo ou avulsa. O segundo armazena o tipo de grupo da visita, que pode ter os seguintes valores: educação formal, educação informal, instituição não educacional e sem instituição. Na Figura 28 podemos observar a nova versão do modelo de dados do software Publicus.

Figura 28: Modelo de dados da segunda versão do software Publicus



4.3 PROGRAMAÇÃO DA PARTE CLIENTE DO MÓDULO DE VISITAS

A partir das descrições textuais dos casos de uso elaboradas e da atualização do modelo de dados, deu-se início à etapa de implementação do módulo de visitas do Publicus. Durante o desenvolvimento deste módulo foi adotado um processo de desenvolvimento iterativo e incremental, contemplando todos os casos de uso descritos textualmente na Seção 4.1. Cada iteração durou cerca de uma semana.

Além disso, é justo ressaltar que, devido ao sistema já estar em desenvolvimento, foi necessário seguir o padrão já estabelecido, a fim de manter a consistência com o que havia sido implementado anteriormente.

Para o controle de versões do código-fonte, foi utilizado a ferramenta git, com o código fonte do projeto sendo armazenado em um repositório privado no Gitlab de projetos do Instituto Metrópole Digital (IMD), como observado na Figura 29. No período de março até agosto de 2024, o autor deste trabalho contribuiu com um total de vinte commits na branch principal do repositório da parte cliente do sistema Publicus (Figura 30).

Figura 29: Repositório privado do projeto da parte cliente do software Publicus

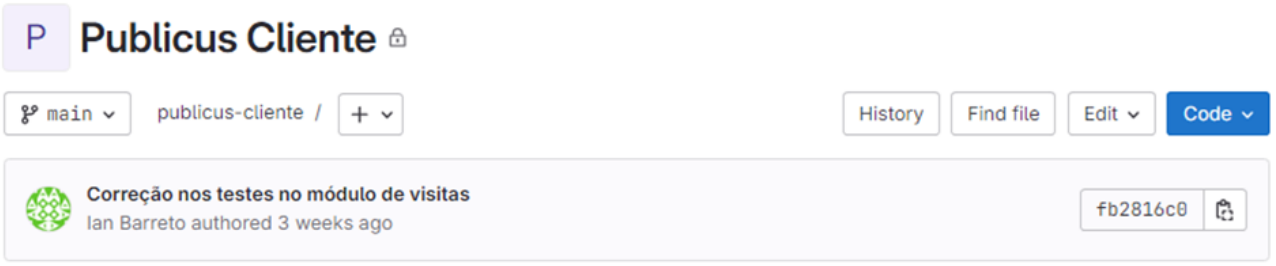
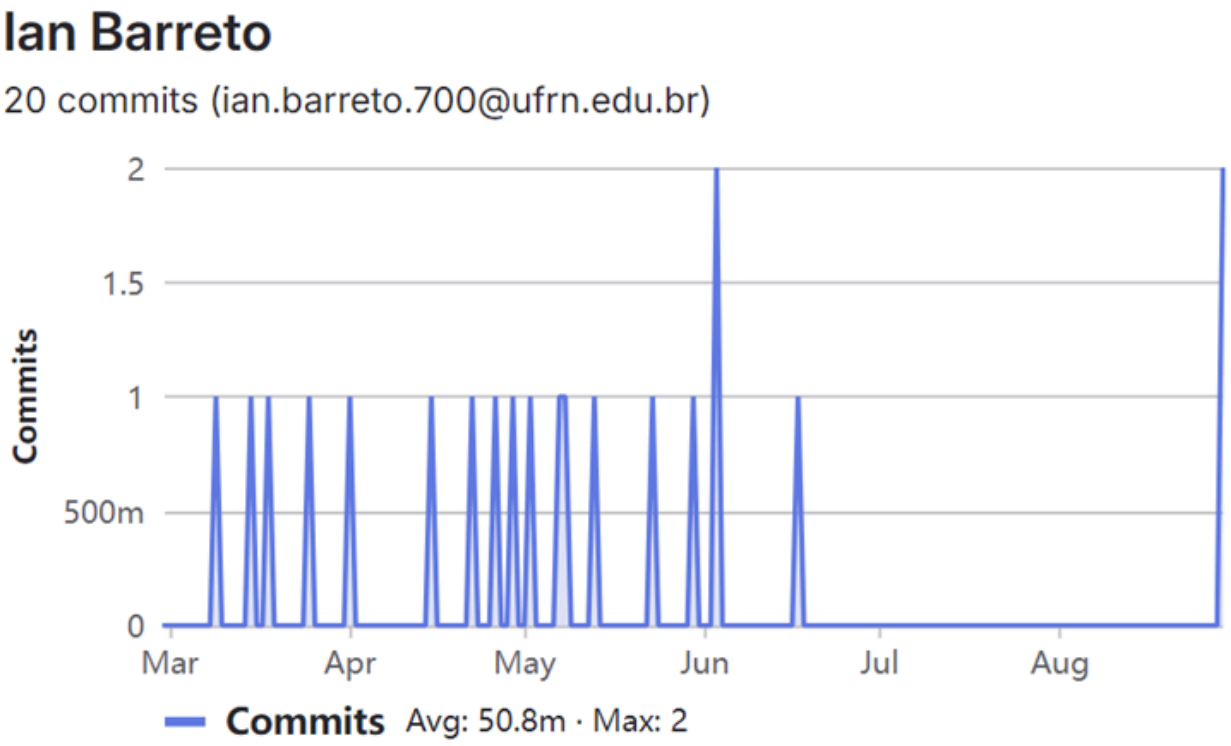


Figura 30: Gráfico de commits do autor deste trabalho neste projeto no git



O desenvolvimento da parte cliente do Publicus utilizou o framework Flutter na versão 3.19.0, com a linguagem de programação Dart na 3.3.0. Como o Publicus é um

software multiplataforma, o Flutter facilita a sua implementação por reutilizar código compartilhado entre as plataformas e administrar bem os códigos diferentes que existem entre elas. O framework Flutter também é capaz de gerar automaticamente muitas adaptações necessárias para o desenvolvimento multiplataforma. Suas extensões para os ambientes de desenvolvimento Visual Studio Code e Android Studio auxiliam bastante o desenvolvimento, inclusive para executar a aplicação sendo desenvolvida, seja em plataformas reais (como o Windows onde se programa) ou simuladas onde o Publicus será utilizado no futuro (como o Android). Deste modo, o Flutter permite ter um projeto único, com o mesmo código-fonte, para desenvolver um sistema para dispositivos móveis e desktop do Publicus.

Na criação de um novo projeto utilizando o framework Flutter, uma estrutura geral é gerada automaticamente. Há diretórios específicos para cada plataforma (Android, iOS, Linux, macOS, Windows e web). Cada diretório contém o código fonte necessário para executar a aplicação com as particularidades de cada plataforma. Além disso, o Flutter também gera um diretório `lib` que contém o código fonte compartilhado por todas as plataformas. É neste diretório que fica o arquivo `main.dart`, responsável por inicializar a execução do software, bem como gerenciar as rotas (“navegação entre telas”) durante a interação do usuário. Também são gerados o diretório `assets`, que contém as imagens utilizadas no sistema, e o arquivo de configuração `pubspec.yaml`, responsável por especificar as dependências que serão utilizadas no projeto e os arquivos de imagens contidos no diretório `assets`.

Em um projeto Flutter, os widgets são os elementos básicos que constroem uma aplicação. Pode ser considerado um widget qualquer declaração de uma parte da interface de usuário. Os widgets são desde elementos estruturais, como botões e menus, até elementos que definem o estilo e layout da aplicação, por exemplo, um esquema de fonte ou cores e o padding também podem ser considerados como widgets. Também é válido ressaltar a possibilidade de um widget ser formado a partir da combinação de outros widgets.

Desse modo, em Flutter podemos ter widgets mais simples e imutáveis, chamados de Stateless Widgets, elementos constantes desenhados apenas uma única vez, assim como ter widgets mais complexos e com múltiplos estados, chamados de Stateful Widgets. Um Stateful Widget é responsável por fornecer uma informação de configuração

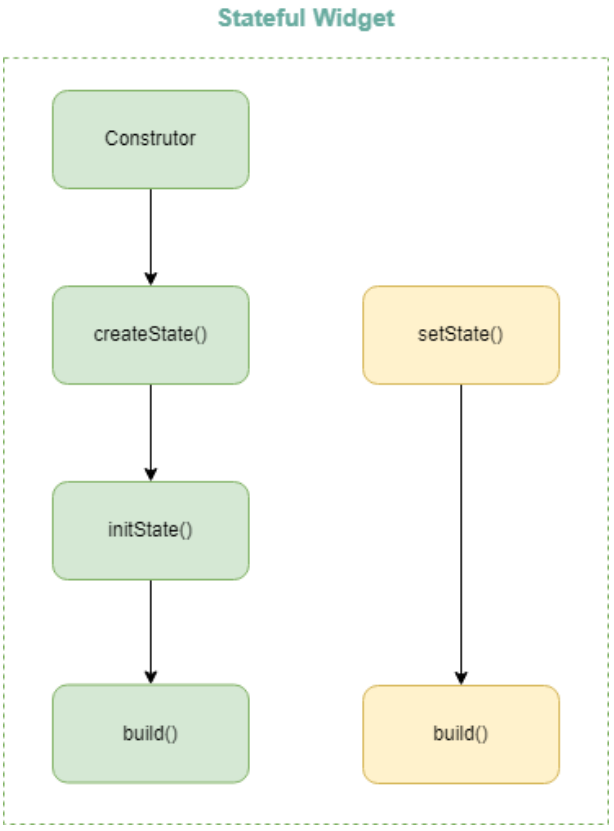
imutável e um objeto State que pode ser alterado com o tempo e que aciona uma reconstrução da interface.

O início do ciclo de vida de um Stateful Widget, ilustrado à esquerda da Figura 31, se dá a partir do seu construtor, que realiza uma chamada ao método `createState` responsável por criar um objeto State. Este objeto contém todas as variáveis que podem alterar o estado do widget e, então, acionar uma reconstrução da interface.

Após a criação do objeto State, o método `initState` é invocado uma única vez, ele é encarregado pela inicialização do estado de um Stateful Widget. Já o método `build`, diferentemente do `initState`, pode ser invocado mais de uma vez em um Stateful Widget, sendo o responsável pela (re)construção da interface.

Quando um estado de um Stateful Widget for alterado, deverá ser realizada uma chamada ao método `setState` (direita da Figura 31). Esta função é encarregada de alterar o estado do widget e invocar o método `build` para que ele possa desenhar a interface de acordo com o novo estado do widget.

Figura 31: Ciclo de vida de um Stateful Widget



O ambiente de desenvolvimento integrado (IDE) utilizado foi o Android Studio, que possibilita a execução de um projeto Flutter em múltiplos dispositivos. Para a execução do sistema em um smartphone foi utilizado um emulador Android 8.0 Oreo, que emula o dispositivo Pixel 3.

Os seguintes pacotes foram utilizados para o desenvolvimento deste projeto: Cupertino Icons (1.0.2), Go Router (14.0.1), Flutter SVG (2.0.2), Step Progress Indicator (1.0.2), Http (1.1.0), Crypto (3.0.2), Flutter Localizations, Flutter Secure Storage (9.0.0), Intl (0.18.1), Async (2.11.0), Path Provider (2.1.3), File Picker (8.0.3), Connectivity Plus (6.0.3).

No desenvolvimento da interface deste trabalho, foi necessária a utilização de ícones com o intuito de melhorar a experiência do usuário. Apesar do framework Flutter oferecer uma boa variedade de ícones, houve a necessidade de usar ícones propostos por Victor Paiva (Paiva, 2023), que podem ser observados na Figura 32.

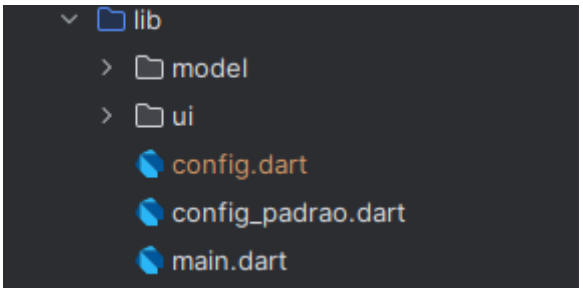
Figura 32: Ícones personalizados do Publicus

Fonte: (Paiva, 2023).

Sobre a estruturação dos diretórios do projeto, o código-fonte em Dart está localizado na pasta `lib` (Figura 33). Ela contém duas pastas: `model` e `ui`. Na pasta `model`, encontram-se as classes de domínio da aplicação: agendamento, cargo, instituição, lista, museu, requisição, usuário e visita. Já na pasta `ui`, temos as interfaces e componentes utilizados no software. Além disso, é válido mencionar que na raiz da pasta `lib`, temos três arquivos: `config.dart`, `config_padrao.dart` e `main.dart`. Os dois

primeiros são arquivos de configuração do projeto, enquanto o último é o arquivo de inicialização do projeto.

Figura 33: Estrutura da pasta lib



Todos os arquivos da pasta `model` existiam antes deste trabalho, inclusive foram desenvolvidos com a colaboração deste autor. Durante o desenvolvimento do módulo de visitas do Publicus neste trabalho, houve a necessidade de atualizar os arquivos `lista.dart` e `visita.dart` (Figura 34) para se adequarem a implementação dos casos de uso realizada neste trabalho.

Já na pasta `ui` (Figura 35), para este trabalho foram criados os arquivos `home_page_visitas.dart`, `detalhar_visita.dart`, `detalhar_visitas.dart`, `inserir_visita.dart`, `lista_visitas.dart` e `visualizar_visitas.dart`. Assim como na pasta `model`, houve também a necessidade de editar arquivos já existentes do diretório `ui`. No arquivo `home_page_inicio.dart` foi desenvolvido um carrossel que exibe as últimas visitas recebidas pelo Museu. Ademais, na implementação do caso de uso "Cadastrar Visita", os arquivos `detalhar_agendamento.dart`, `detalhar_agendamentos.dart`, `detalhar_responsavel.dart` e `detalhar_responsaveis.dart` tiveram que ser modificados para ser possível cadastrar uma visita a partir de um responsável ou de um agendamento confirmado.

Destacados na cor azul nas Figuras 34 e 35, podemos observar os arquivos que o autor contribuiu durante a etapa de desenvolvimento deste trabalho.

Figura 34: Arquivos da pasta model

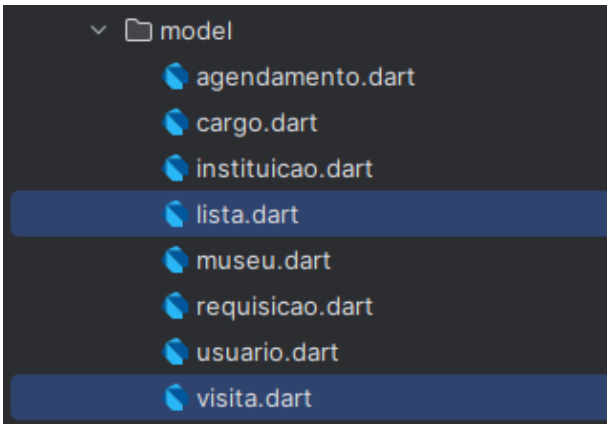
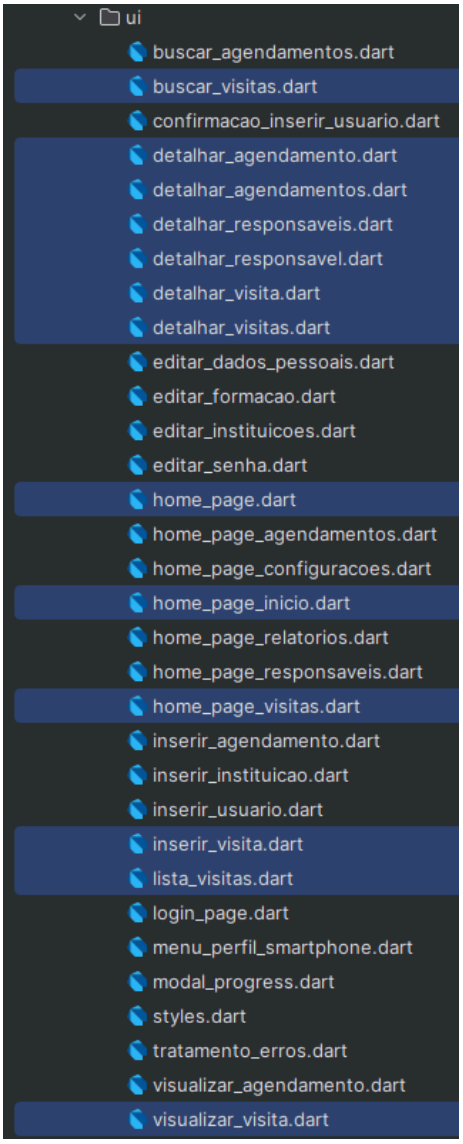


Figura 35: Arquivos da pasta UI



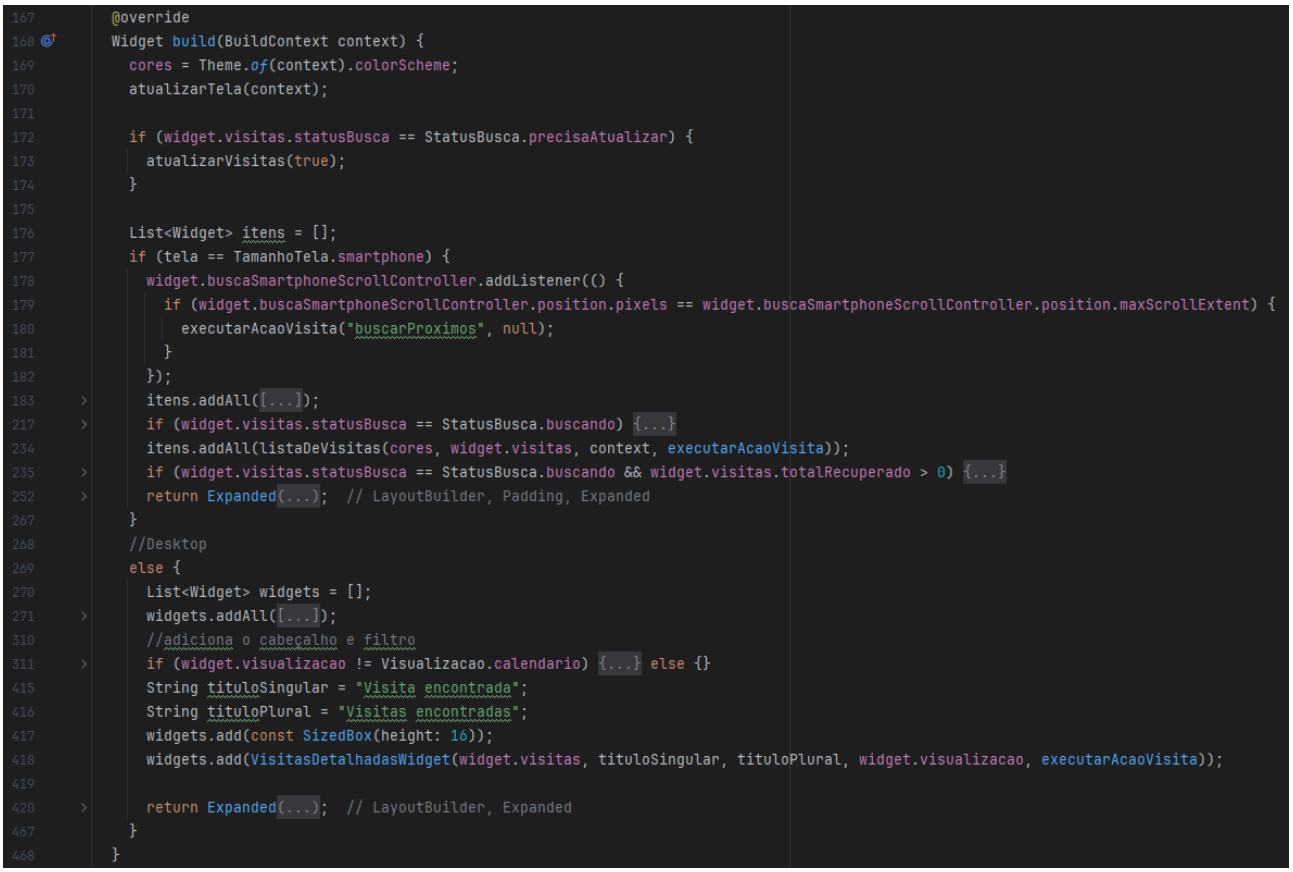
O desenvolvimento da adaptação automática da interface do Publicus para diferentes dispositivos buscou integrar as soluções de cada plataforma num único projeto, promovendo o reúso e a facilidade de manutenção do código-fonte. Para tanto, foram utilizadas duas estratégias básicas. A primeira estratégia identifica o tamanho atual da tela com o método `atualizarTela` (Figura 36), para determinar qual dispositivo deve ser considerado no momento de construção da interface.

Figura 36: Implementação do método “atualizarTela”

```
222  void atualizarTela(BuildContext context) {
223    double tamanho = MediaQuery.of(context).size.width;
224
225    if (tamanho <= 600) {
226      tela = TamanhoTela.smartphone;
227    } else if (tamanho > 600 && tamanho <= 900) {
228      tela = TamanhoTela.tablet;
229    } else {
230      tela = TamanhoTela.desktop;
231    }
232  }
```

A segunda estratégia adapta o layout da interface sendo construída em função do tamanho de tela identificado. Esta adaptação foi implementada no método `build` dos widgets (Figura 31) para que eles construam a interface adequada para o tamanho atual da tela. Por exemplo, no arquivo `home_page_visitas.dart` (Figura 37), encontramos o código-fonte da página inicial do módulo de visitas do sistema. O método `build` faz o cálculo do tamanho da tela atual utilizando a função `atualizarTela`, na linha 170. A partir do resultado obtido, ele irá realizar a construção da interface adequada para aquele dimensionamento, usando estruturas de controle, como o `if` da linha 177 para smartphone e o `else` da linha 269 para desktop. Como explicado anteriormente, no ciclo de vida de um Stateful Widget, o método `build` sempre será invocado para realizar a construção da interface, dessa forma, ao redimensionar a tela, será realizada uma nova chamada ao `build`, que irá realizar o cálculo novamente e desenhar a interface adequada às novas proporções da tela.

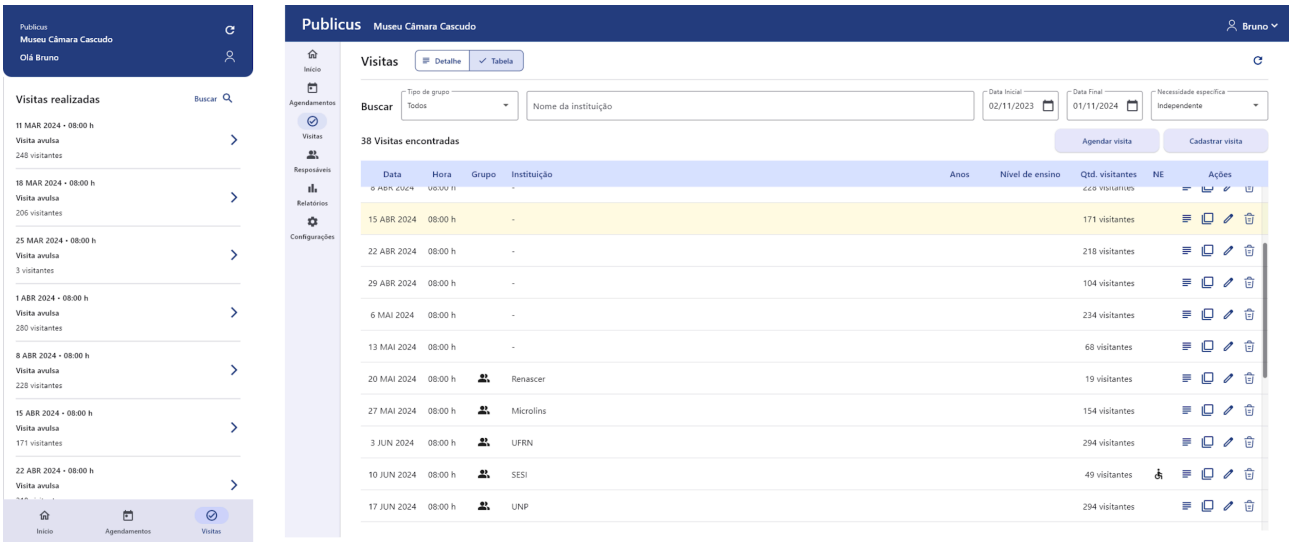
Figura 37: Trecho de código do arquivo home_page_visitas.dart



```dart
@override
Widget build(BuildContext context) {
  cores = Theme.of(context).colorScheme;
  atualizarTela(context);

  if (widget.visitas.statusBusca == StatusBusca.precisaAtualizar) {
    atualizarVisitas(true);
  }

  List<Widget> itens = [];
  if (tela == TamanhoTela.smartphone) {
    widget.buscaSmartphoneScrollController.addListener(() {
      if (widget.buscaSmartphoneScrollController.position.pixels == widget.buscaSmartphoneScrollController.position.maxScrollExtent) {
        executarAcaoVisita("buscarProximos", null);
      }
    });
    itens.addAll([...]);
    if (widget.visitas.statusBusca == StatusBusca.buscando) {...}
    itens.addAll(listaDeVisitas(cores, widget.visitas, context, executarAcaoVisita));
    if (widget.visitas.statusBusca == StatusBusca.buscando && widget.visitas.totalRecuperado > 0) {...}
    return Expanded(...);  // LayoutBuilder, Padding, Expanded
  }
  //Desktop
  else {
    List<Widget> widgets = [];
    widgets.addAll([...]);
    //adiciona o cabeçalho e filtro
    if (widget.visualizacao != Visualizacao.calendario) {...} else {}
    String tituloSingular = "Visita encontrada";
    String tituloPlural = "Visitas encontradas";
    widgets.add(const SizedBox(height: 16));
    widgets.add(VisitasDetalhadasWidget(widget.visitas, tituloSingular, tituloPlural, widget.visualizacao, executarAcaoVisita));

    return Expanded(...);  // LayoutBuilder, Expanded
  }
}
```

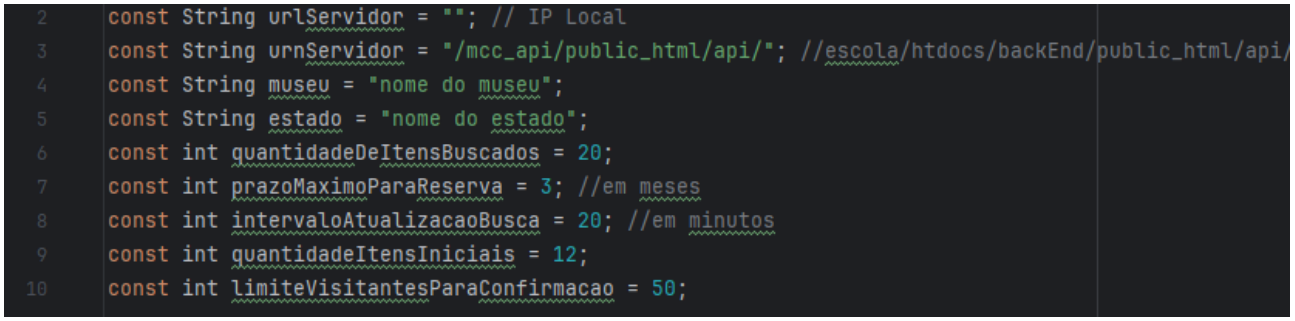
A Figura 38 ilustra as duas diferentes interfaces que podem ser construídas no `build` dependendo do tamanho da tela definida no método `atualizarTela`. Na esquerda da figura, encontramos a interface ao executar o sistema Publicus em um emulador Android, já na direita temos a mesma tela ao executar o sistema Publicus em um dispositivo desktop no sistema operacional Windows.

Figura 38: Página inicial de visitas, versão mobile (esquerda) e versão desktop (direita)



Com o intuito de aumentar a manutenibilidade do sistema, foram criados dois arquivos para definir valores padrões para certas variáveis de configuração. Estes arquivos foram chamados de `config.dart` e `config_padrao.dart`. Desse modo, caso seja necessário alterar os valores destas configurações, seria preciso alterar esses valores em apenas um lugar do código-fonte, facilitando assim a manutenção do Publicus. Além disso, é importante salientar a necessidade da criação de dois arquivos de configuração devido ao Publicus ser um projeto colaborativo com gestão de código-fonte via git. Criou-se o `config_padrao.dart` com os valores iniciais dessas variáveis e um arquivo `config.dart` que será ignorado pelo sistema de controle de versões. Dessa forma, o arquivo `config.dart` pode ter sua configuração alterada por cada desenvolvedor para acomodar particularidades do seu computador local, sem interferir nos valores padrões definidos no outro arquivo. A Figura 39 apresenta as variáveis armazenadas no arquivo `config_padrao.dart` e no Quadro 8 temos as suas respectivas utilidades.

Figura 39: Arquivo config_padrao.dart



```dart
const String urlServidor = ""; // IP Local
const String urnServidor = "/mcc_api/public_html/api/"; //escola/htdocs/backEnd/public_html/api/
const String museu = "nome do museu";
const String estado = "nome do estado";
const int quantidadeDeItensBuscados = 20;
const int prazoMaximoParaReserva = 3; //em meses
const int intervaloAtualizacaoBusca = 20; //em minutos
const int quantidadeItensIniciais = 12;
const int limiteVisitantesParaConfirmacao = 50;
```

Quadro 8: Variáveis do config_padrao.dart

Variável	Utilidade
urlServidor	Endereço da API que será consultada pelo sistema Publicus no servidor web
urnServidor	URN que será consultado pelo sistema Publicus no servidor web
museu	Nome do museu a ser exibido na interface
estado	Estado em que o museu está localizado
prazoMaximoParaReserva	Prazo máximo para agendar uma visita, a partir do dia atual
intervaloAtualizacaoBusca	Intervalo de tempo em minutos para que o Publicus realize uma atualização automática dos dados, ou seja, faça uma nova consulta aos dados no servidor
quantidadeDeItensBuscados	Quantidade máxima de itens a serem retornados em uma consulta a API
quantidadeItensIniciais	Quantidade máxima de itens a serem exibidos na página inicial
limiteVisitantesParaConfirmacao	A partir desse número de visitantes, será exigido uma confirmação por modal ao cadastrar uma visita

O Material Design[3] é um sistema de design construído e mantido pela Google, que inclui uma orientação aprofundada sobre características, componentes (*widgets*) e comportamentos básicos da interface com usuário para Android, Flutter e sistemas web.

[3] https://m3.material.io/

Durante o desenvolvimento do Publicus, foi criado o arquivo `styles.dart` que define a baseline da aplicação. No Material Design, baseline é definido como o esquema de cores estático e padrão para a interface. Na Figura 40 e 41, podemos visualizar a baseline para o tema claro e escuro, respectivamente, definidos por Victor Paiva (Paiva, 2023) durante a elaboração do projeto de interface da segunda versão do Publicus. Já na Figura 42, encontra-se um exemplo de como o esquema de cores foi definido em nível de código no arquivo `styles.dart`.

Figura 40: Esquema de cores do Publicus para tema claro

Figura 41: Esquema de cores do Publicus para tema escuro

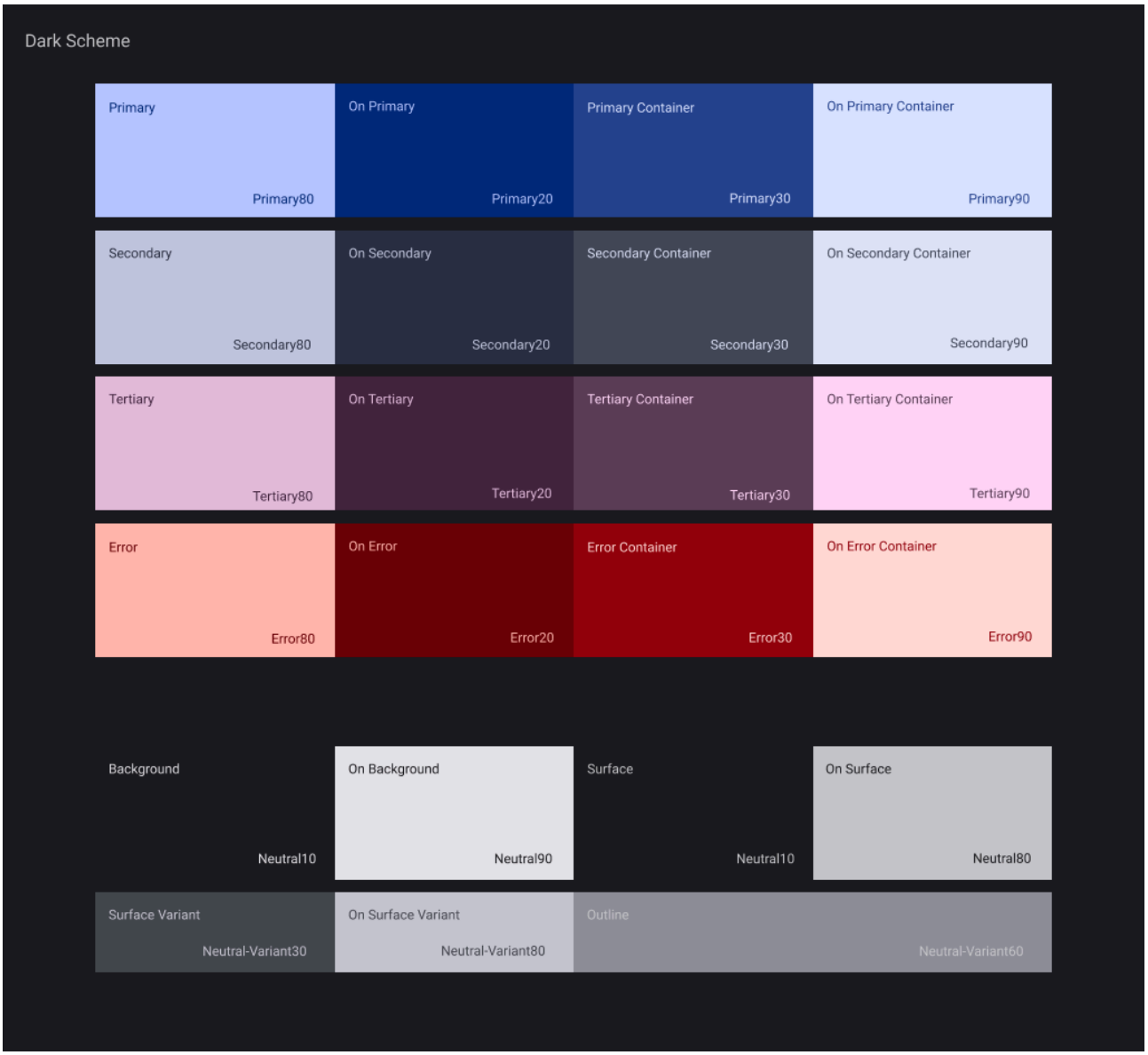
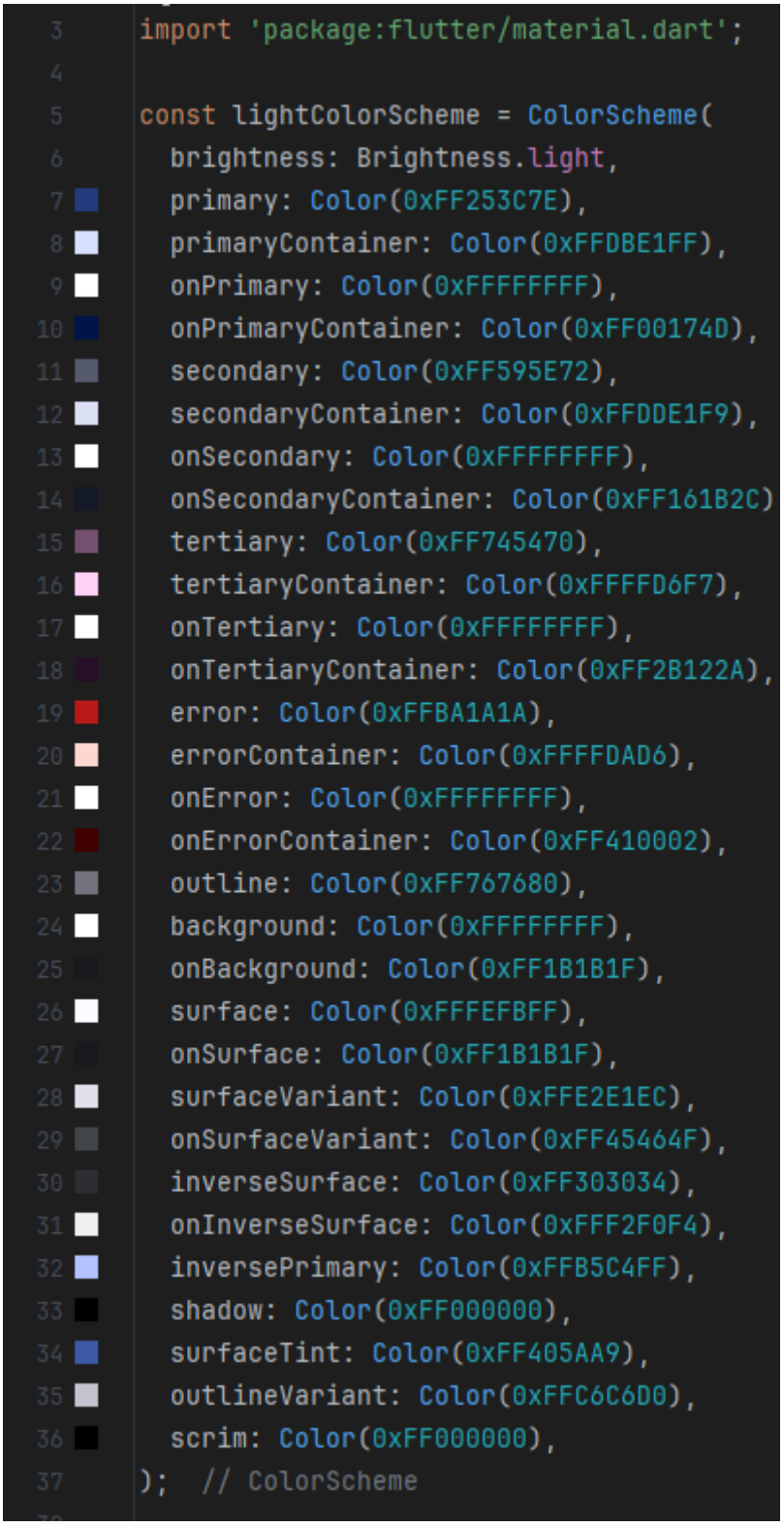


Figura 42: Definição do esquema de cores no arquivo `styles.dart`



```dart
import 'package:flutter/material.dart';

const lightColorScheme = ColorScheme(
  brightness: Brightness.light,
  primary: Color(0xFF253C7E),
  primaryContainer: Color(0xFFDBE1FF),
  onPrimary: Color(0xFFFFFFFF),
  onPrimaryContainer: Color(0xFF00174D),
  secondary: Color(0xFF595E72),
  secondaryContainer: Color(0xFFDDE1F9),
  onSecondary: Color(0xFFFFFFFF),
  onSecondaryContainer: Color(0xFF161B2C),
  tertiary: Color(0xFF745470),
  tertiaryContainer: Color(0xFFFFD6F7),
  onTertiary: Color(0xFFFFFFFF),
  onTertiaryContainer: Color(0xFF2B122A),
  error: Color(0xFFBA1A1A),
  errorContainer: Color(0xFFFFDAD6),
  onError: Color(0xFFFFFFFF),
  onErrorContainer: Color(0xFF410002),
  outline: Color(0xFF767680),
  background: Color(0xFFFFFFFF),
  onBackground: Color(0xFF1B1B1F),
  surface: Color(0xFFFEFBFF),
  onSurface: Color(0xFF1B1B1F),
  surfaceVariant: Color(0xFFE2E1EC),
  onSurfaceVariant: Color(0xFF45464F),
  inverseSurface: Color(0xFF303034),
  onInverseSurface: Color(0xFFF2F0F4),
  inversePrimary: Color(0xFFB5C4FF),
  shadow: Color(0xFF000000),
  surfaceTint: Color(0xFF405AA9),
  outlineVariant: Color(0xFFC6C6D0),
  scrim: Color(0xFF000000),
);  // ColorScheme
```

Além do esquema de cores, o `styles.dart` também foi utilizado para definir um padrão para a tipografia a partir da hierarquia de informação projetada pelo Victor (Paiva, 2023), conforme mostrado na Figura 43.

Figura 43: Definição dos estilos de texto no arquivo `styles.dart`

```dart
const estiloTitulo1 = TextStyle(fontWeight: FontWeight.w800, fontSize: 60);
const estiloTitulo2 = TextStyle(fontWeight: FontWeight.w700, fontSize: 48);
const estiloTitulo3 = TextStyle(fontWeight: FontWeight.w600, fontSize: 38);
const estiloTitulo4 = TextStyle(fontWeight: FontWeight.w600, fontSize: 30);

const estiloSubTitulo1 = TextStyle(fontWeight: FontWeight.w700, fontSize: 30);
const estiloSubTitulo2 = TextStyle(fontWeight: FontWeight.w600, fontSize: 24);
const estiloSubTitulo3 = TextStyle(fontWeight: FontWeight.w500, fontSize: 20);
const estiloSubTitulo4 = TextStyle(fontWeight: FontWeight.w600, fontSize: 18);
const estiloSubTitulo5 = TextStyle(fontWeight: FontWeight.w500, fontSize: 16);
const estiloSubTitulo6 = TextStyle(fontWeight: FontWeight.w500, fontSize: 14);

const estiloCorpoTexto1 = TextStyle(fontWeight: FontWeight.w400, fontSize: 18);
const estiloCorpoTexto2 = TextStyle(fontWeight: FontWeight.w400, fontSize: 16);
const estiloCorpoTexto3 = TextStyle(fontWeight: FontWeight.w400, fontSize: 14);
const estiloCorpoTexto4 = TextStyle(fontWeight: FontWeight.w400, fontSize: 12);

const estiloBarraNavegacao = TextStyle(fontWeight: FontWeight.w500, fontSize: 12);
```

Em relação à navegação entre as telas, foi utilizado o pacote GoRouter para auxiliar no fluxo entre páginas da aplicação. No arquivo `main.dart`, foi declarada uma variável `_router`, que armazena uma lista de rotas da aplicação. Essa variável será passada como o valor do atributo `routerConfig` do widget `MaterialApp`, que é responsável por realizar a configuração das rotas do sistema. `MaterialApp` será o *widget* retornado pelo método `build` no arquivo `main.dart` (Figura 44). A escolha do pacote GoRouter se deve pelo aumento de coesão e redução do acoplamento, favorecendo assim o reuso e a manutenibilidade do software.

Figura 44: Trecho de código da função `build` no arquivo `main.dart`

```
return MaterialApp.router(
    debugShowCheckedModeBanner: false,
    routerConfig: _router,
    title: 'Publicus',
    theme: ThemeData(colorScheme: lightColorScheme, useMaterial3: true),
    scaffoldMessengerKey: scaffoldKey,
    localizationsDelegates: GlobalMaterialLocalizations.delegates,
    supportedLocales: const [Locale("pt", "BR")],
    scrollBehavior: const MaterialScrollBehavior().copyWith(
      dragDevices: {PointerDeviceKind.mouse, PointerDeviceKind.touch, PointerDeviceKind.stylus, PointerDeviceKind.trackpad},
    ));  // MaterialApp.router
```

Uma rota é composta por um `path` e um `builder`. O `path` é o caminho responsável por identificar a rota que se pretende acessar, enquanto o `builder` é uma função que deverá retornar o Widget da página que a rota irá redirecionar. Na Figura 45, temos um exemplo simples de como as rotas são configuradas. Nesta figura, encontramos a rota inicial do Publicus indicada pelo `path` "/". No `builder` desta rota, podemos ver que é possível uma rota retornar diferentes widgets. Quando o usuário não estiver autenticado na aplicação, ele será direcionado para a página de login. Caso contrário, o usuário será direcionado para a página inicial (`HomePage`).

Figura 45: Trecho de código de uma rota do GoRouter no arquivo `main.dart`

```
43      GoRoute(
44        path: "/",
45        builder: (BuildContext context, GoRouterState state) {
46          if (Usuario.usuarioLogado == null) {
47            return LoginPage(true);
48          }
49          return const HomePage();
50        },
51      ),  // GoRoute
```

Além disso, o GoRouter possibilita que o usuário seja direcionado para uma mesma rota a partir de diferentes telas. Por exemplo, as Figuras 46 e 47 ilustram a página inicial do Publicus e a página inicial do módulo de visitas, respectivamente. Ambas possuem o botão "Cadastrar visita". Quando o usuário clicar neste botão, ele será

redirecionado para a mesma rota (`context.push('/cadastrarVisita',…)`) e a tela para Cadastrar visita será exibida (Figura 48). Isso facilita o reúso e a manutenção desta chamada em diferentes locais da aplicação.

Figura 46: Página inicial do software Publicus (desktop)

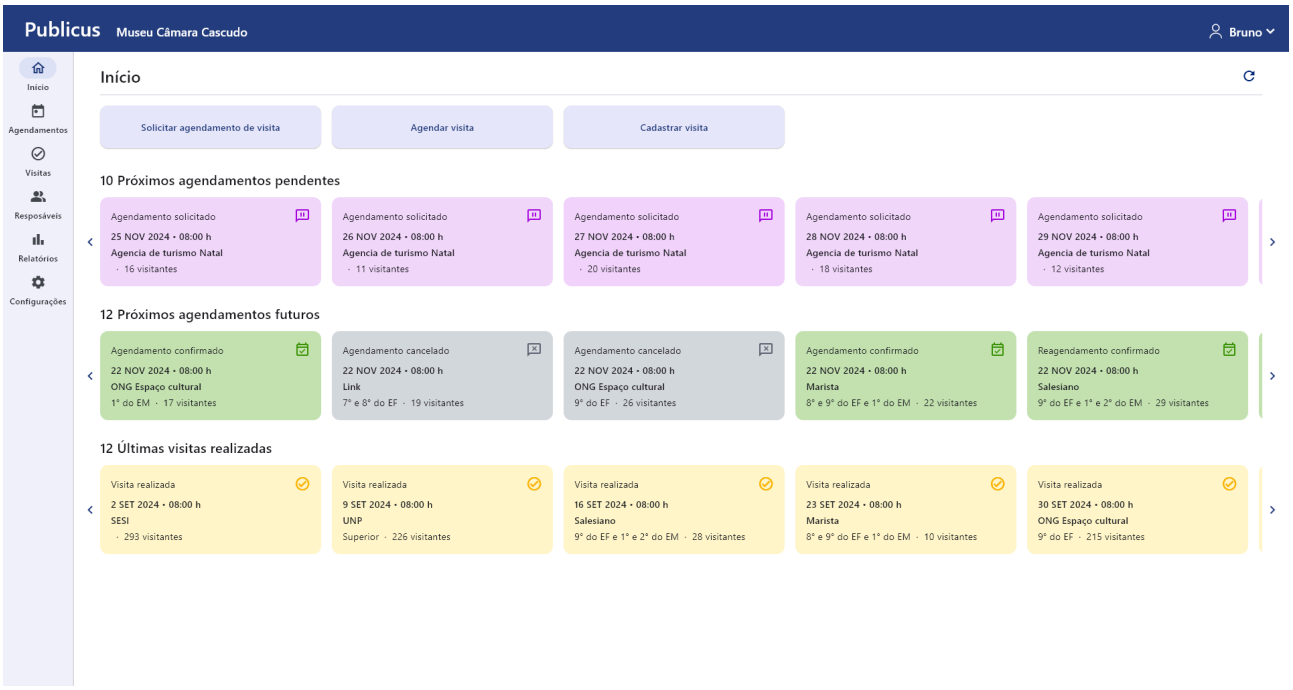


Figura 47: Página inicial do módulo de visitas do software Publicus (desktop)

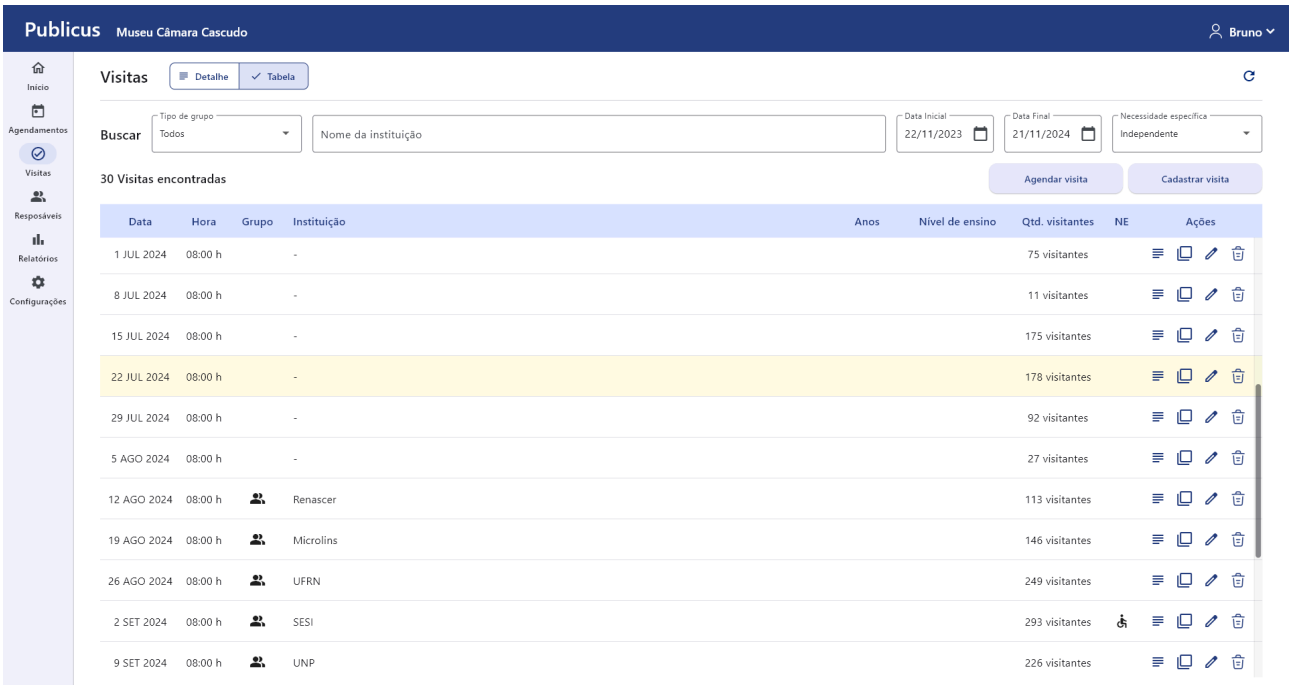
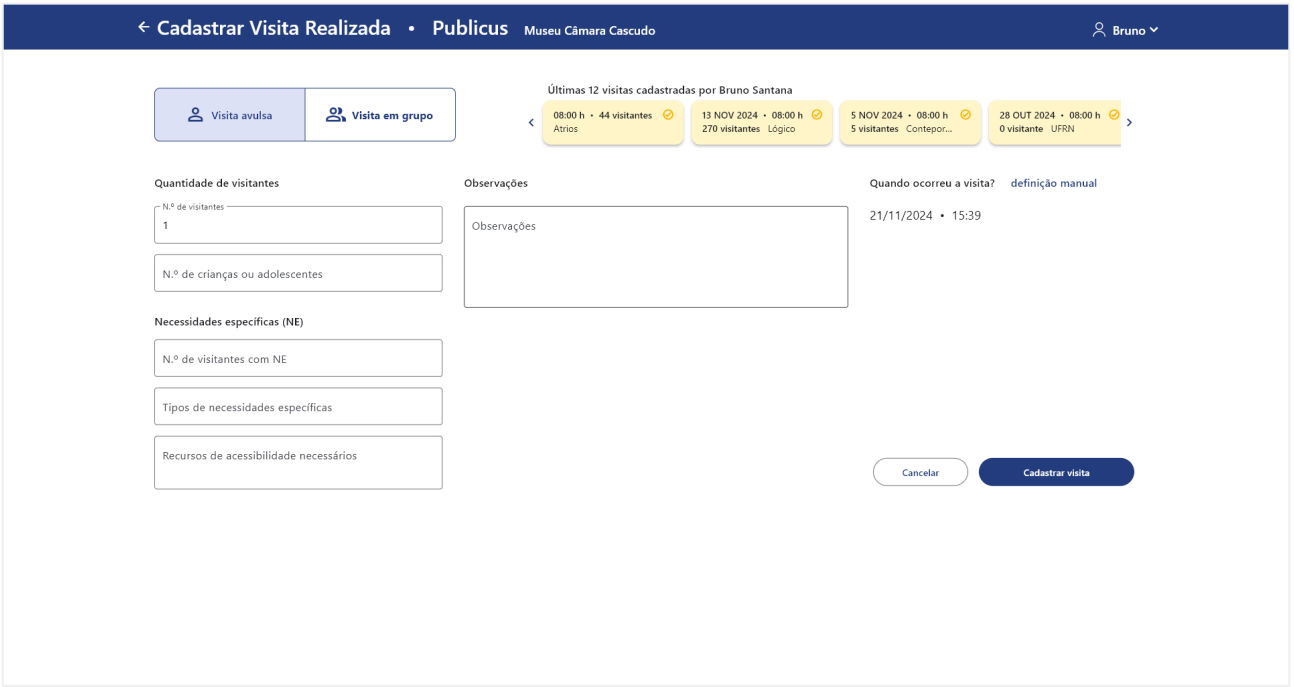


Figura 48: Página Cadastrar visita avulsa

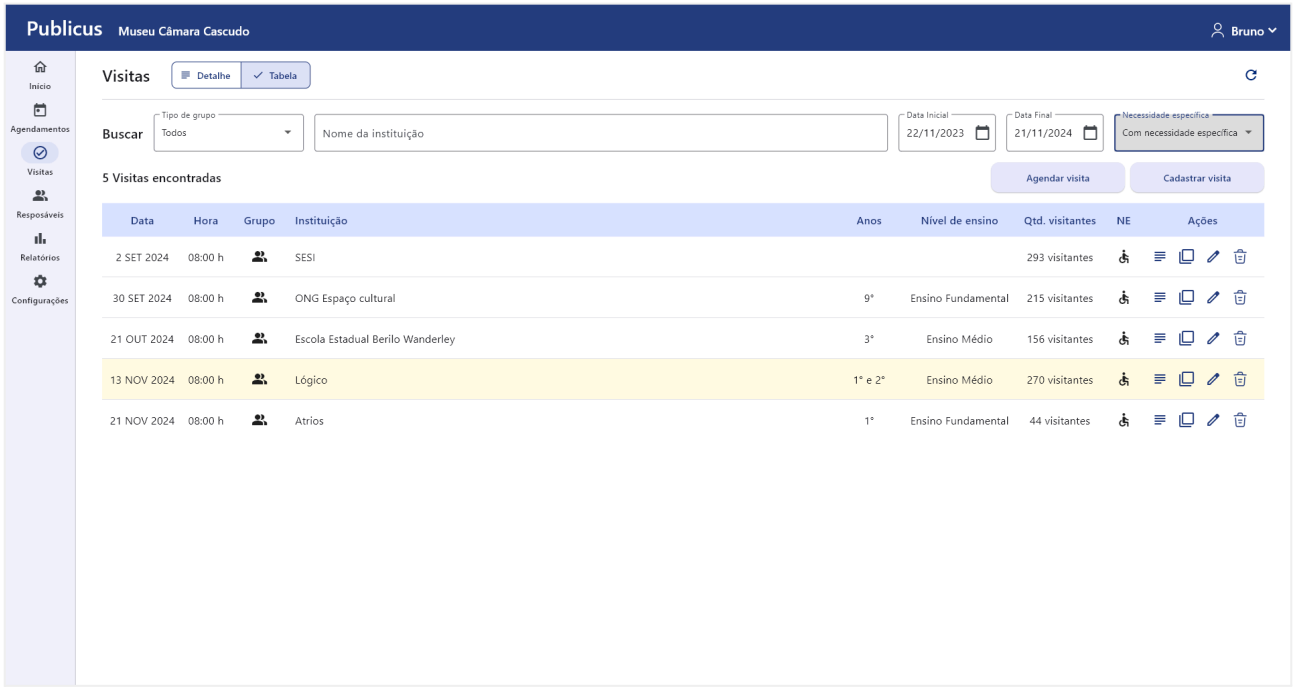


Também é válido mencionar que o GoRouter utiliza o conceito de pilhas para empilhar as telas que foram abertas pelo usuário. Isso permite que o usuário retorne facilmente para a tela anterior, removendo a tela que está no topo da pilha (`context.pop()`), ou seja, a última tela aberta pelo usuário.

A seguir, serão apresentadas as telas desenvolvidas para cada um dos 7 casos de uso no escopo deste trabalho (Figura 20). Desta forma, é possível observar em execução a interface do software programado neste trabalho. Quase todas as telas são direcionadas apenas para desktop, exceto a página inicial que também tem seu equivalente em smartphone. As outras telas sobre visitas no smartphone não foram apresentadas, pois elas já existiam antes da elaboração deste trabalho e não sofreram alterações significativas desenvolvidas nele.

A Figura 49 ilustra a tela desenvolvida para o caso de uso “Buscar Visitas”.

Figura 49: Página desenvolvida para o caso de uso Buscar Visita



As Figuras 50 e 51 ilustram as telas desenvolvidas para o caso de uso “Detalhar Visita” neste trabalho.

Figura 50: Página desenvolvida para o caso de uso Detalhar Visita (Página Inicial do módulo de visitas)

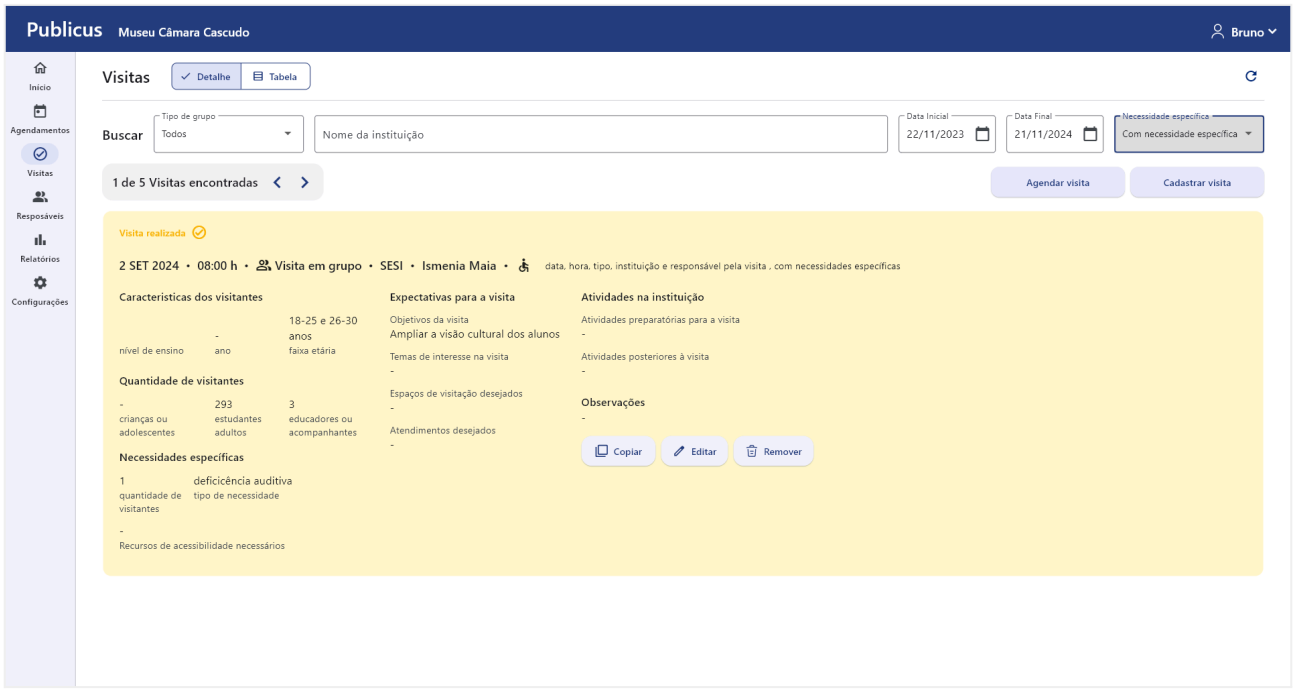
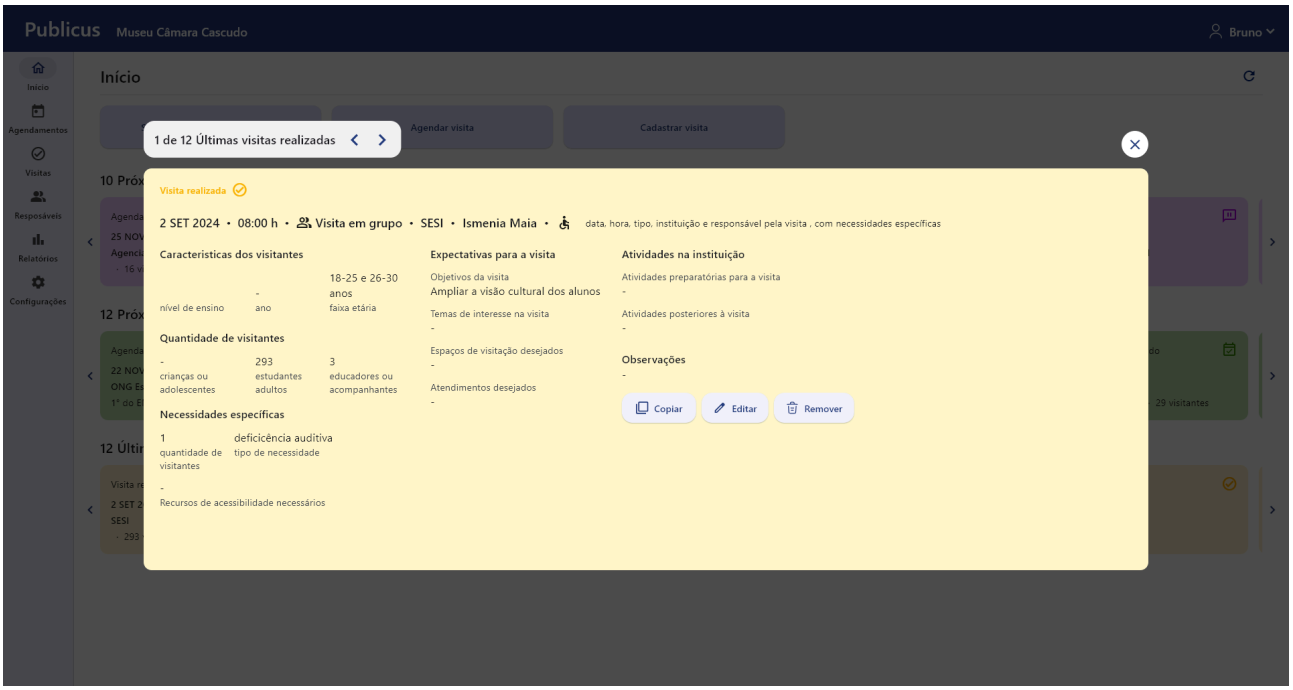
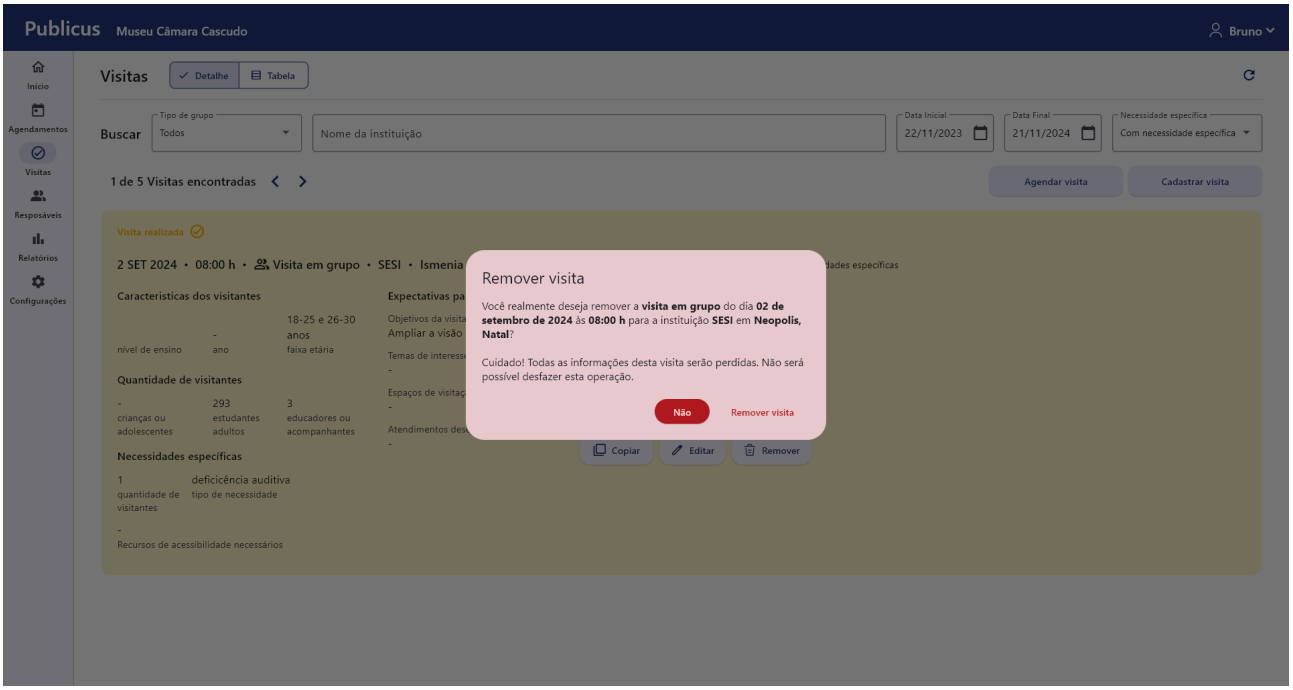


Figura 51: Página desenvolvida para o caso de uso Detalhar Visita (Página Inicial do Publicus)



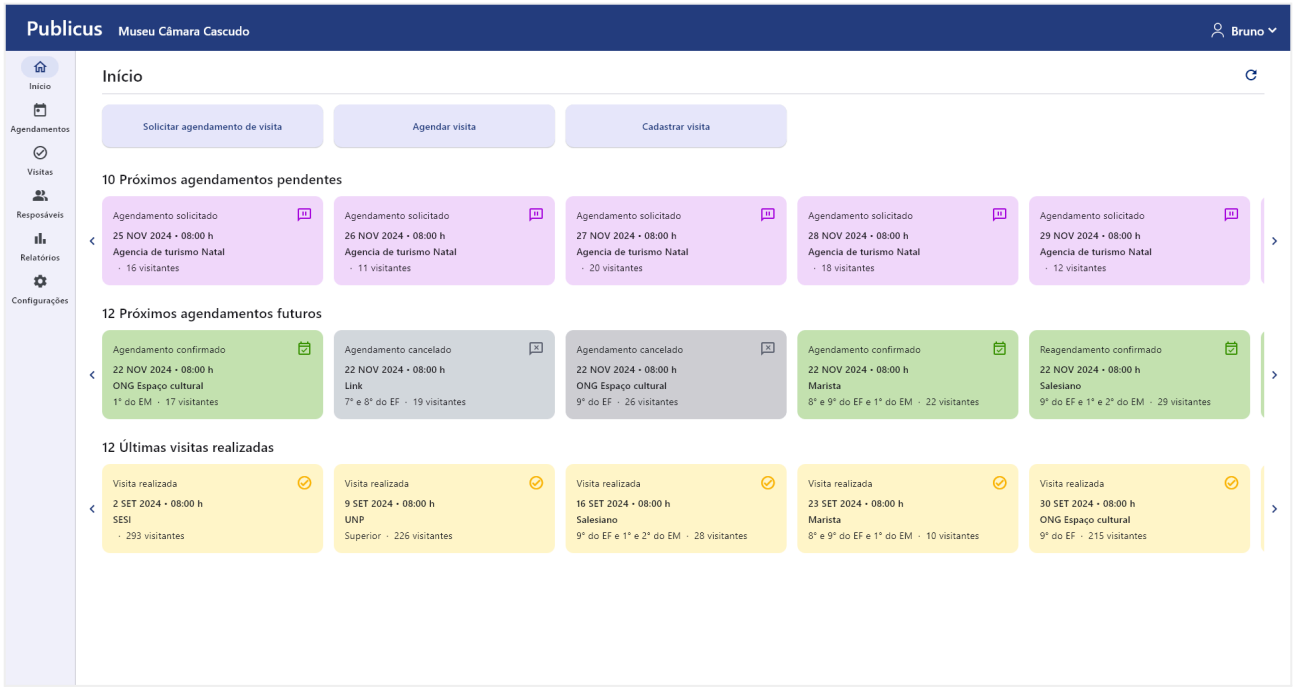
A Figura 52 ilustra a tela desenvolvida para o caso de uso “Remover Visita” neste trabalho.

Figura 52: Página desenvolvida para o caso de uso Remover Visita



A Figura 53 ilustra a tela de desktop adaptada para o caso de uso “Consultar Últimas Visitas” neste trabalho. As principais modificações foram incluir o botão “Cadastrar visita” e exibir um novo carrossel no final em amarelo, com as últimas visitas cadastradas.

Figura 53: Página desenvolvida para o caso de uso Consultar Últimas Visitas



As Figuras 54 e 55 ilustram as telas desenvolvidas para os casos de uso “Cadastrar Visita”, “Editar Visita” e “Copiar Visita” neste trabalho. A primeira registra visitas avulsas, enquanto que a segunda registra visitas em grupo.

Figura 54: Página desenvolvida para o caso de uso Cadastrar, Editar e Copiar Visita Avulsa

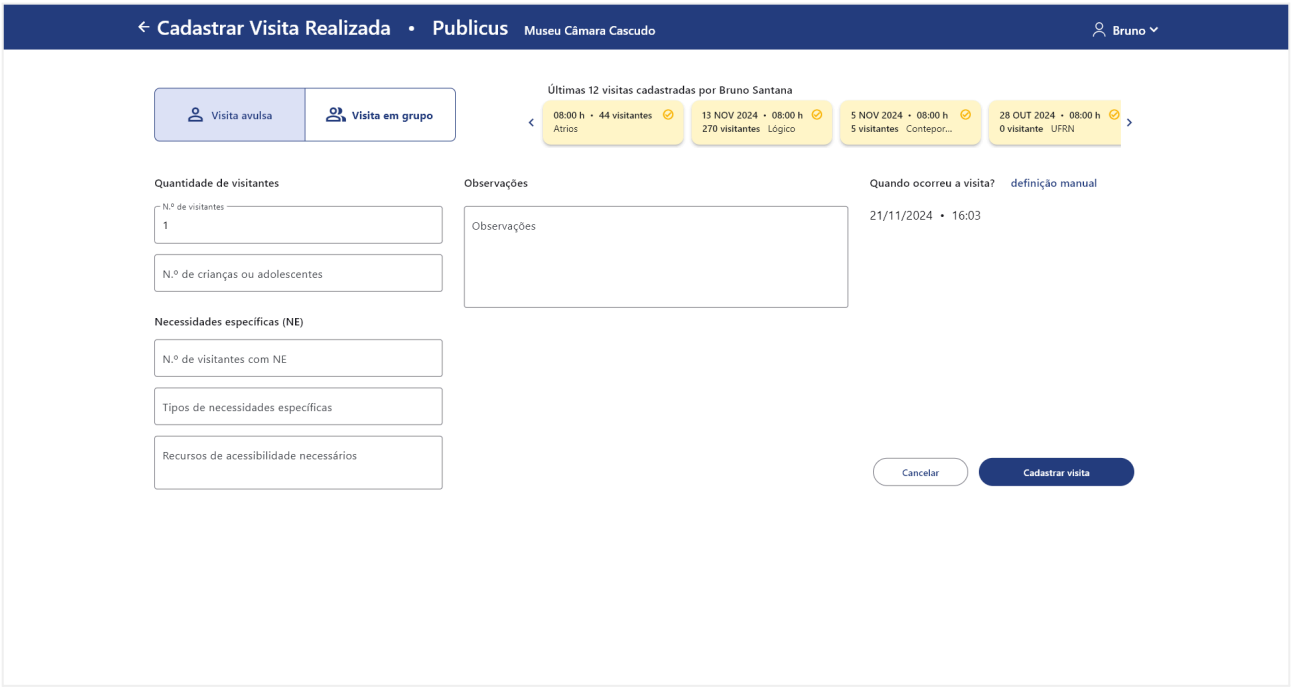
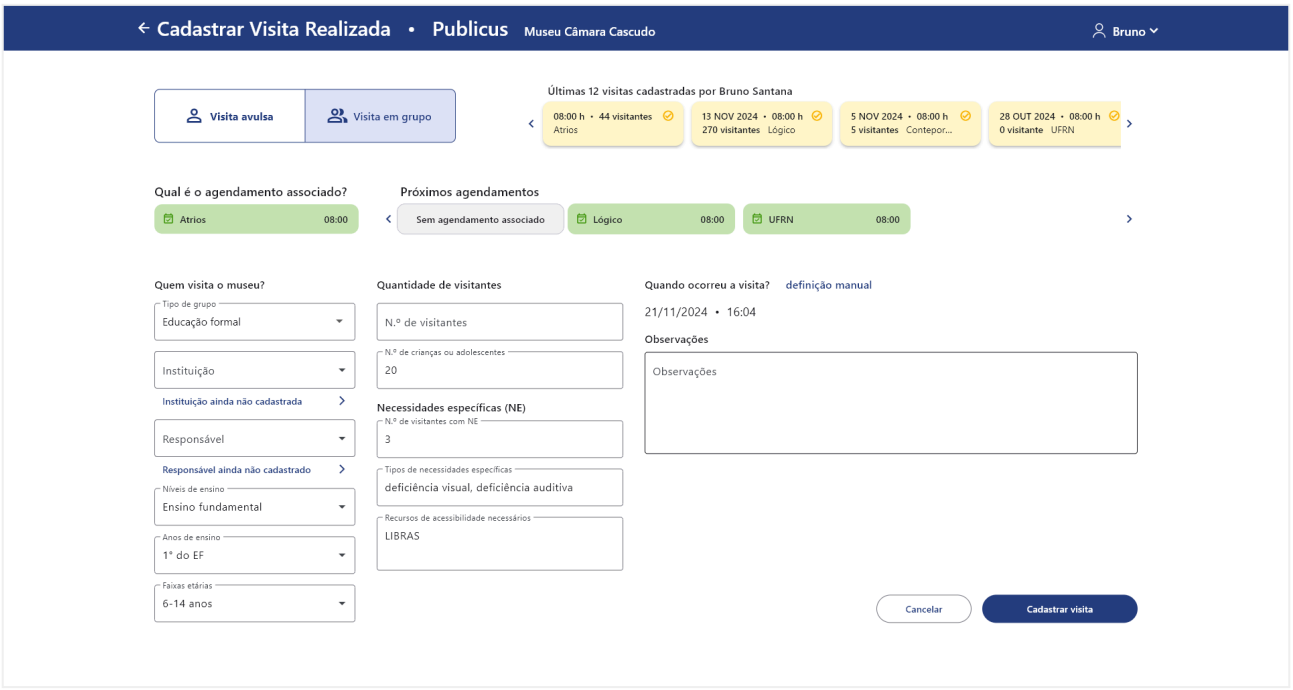


Figura 55: Página desenvolvida para o caso de uso Cadastrar, Editar e Copiar Visita em Grupo



4.4 PROGRAMAÇÃO DA PARTE SERVIDOR DO MÓDULO DE VISITAS

A implementação da parte servidor do módulo de visitas no software Publicus também utilizou a ferramenta git para controlar as versões do código-fonte. Novamente, o projeto da parte servidor foi armazenado em um repositório privado no Gitlab de projetos do Instituto Metrópole Digital (IMD). Conforme ilustrado na Figura 56, no período de março até agosto de 2024, o autor deste trabalho contribuiu com um total de sete *commits* na *branch* principal do *backend* do software Publicus (Figura 57).

Figura 56: Repositório da parte servidor do software Publicus

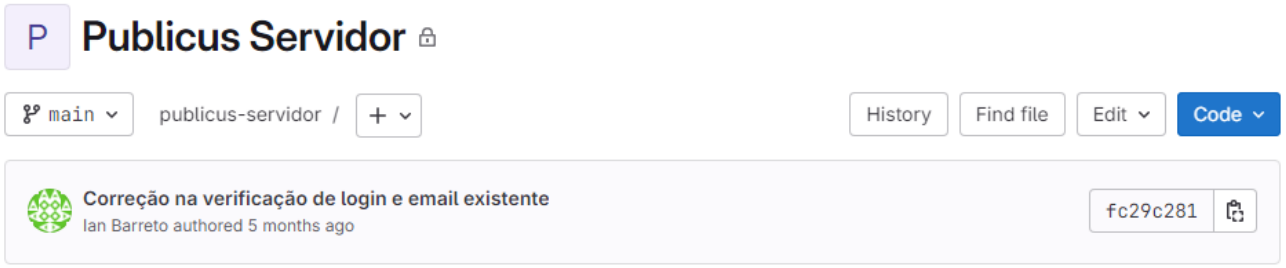
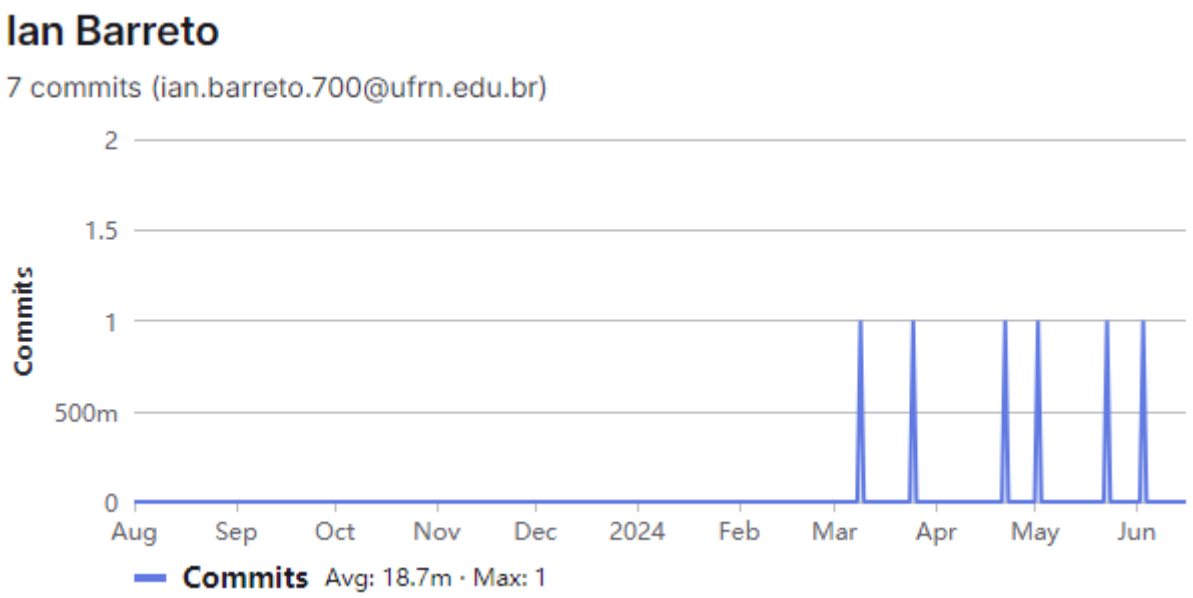


Figura 57: Gráfico de commits do autor na parte servidor do software Publicus



A implementação da parte servidor do software Publicus foi realizada com a linguagem de programação PHP na versão 8.1.10 e com o sistema de gerenciamento de banco de dados (SGDB) MySQL na versão 8.0.31. Além disso, o XAMPP, uma distribuição do Apache que contém o MariaDB e dá suporte a linguagem de programação PHP, foi utilizado para executar o projeto durante o período de desenvolvimento. A depuração do código-fonte foi realizada com a ferramenta Xdebug 3.1.5, uma extensão PHP que oferece recursos de depuração.

Também é válido ressaltar que durante o desenvolvimento do sistema utilizamos a ferramenta Insomnia, com o propósito de testar a API através do envio de requisições HTTP. Isso permitiu simular o comportamento de um usuário sem a necessidade de uma interface gráfica. Então, foi possível validar as rotas do sistema e seus resultados durante a fase de desenvolvimento da parte servidor sem depender da implementação da parte cliente, garantindo que os serviços funcionassem de acordo com o esperado.

Como ambiente de desenvolvimento integrado (IDE), foi utilizado o Visual Studio Code. Para administrar o banco de dados foi usado o MySQL Workbench, uma ferramenta que possibilita a administração, design, criação e manutenção do banco de dados em um único ambiente de desenvolvimento.

Durante o desenvolvimento do *backend*, foi necessário utilizar duas dependências no projeto: PHP-JWT (versão 6.3) e PHPMailer (versão 6.6). O PHP-JWT é uma biblioteca em PHP que possibilita a codificação e decodificação de JSON Web Tokens (JWT). O JWT é utilizado pelo Publicus para realizar a autenticação dos usuários e validar se um usuário tem acesso a determinado serviço que está sendo utilizado. Já o PHPMailer é uma biblioteca para realizar o envio de e-mails no software Publicus. É importante ressaltar que para o gerenciamento dessas dependências foi necessário utilizar o software Composer.

Apesar da possibilidade de um sistema desenvolvido em PHP poder ser desenvolvido sem distinção entre front e backend, foi definido que para o sistema Publicus apenas o backend seria implementado nessa linguagem de programação. Optar por um backend em PHP exclusivamente para construir uma API padronizada possibilitou o uso de práticas modernas de desenvolvimento, favorecendo assim a interoperabilidade e a modularidade do sistema.

Além disso, ao evitar o acoplamento entre a lógica de negócio e a apresentação, garantiu-se que o backend pudesse ser modificado com menor impacto direto no

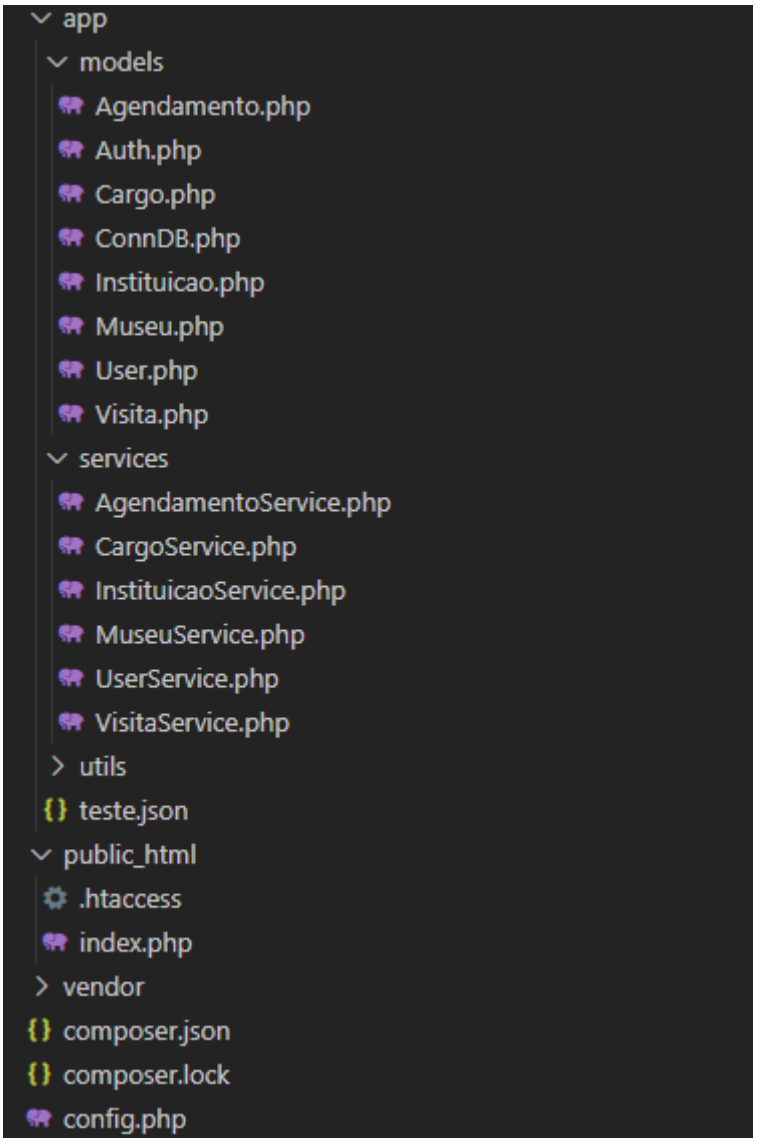
frontend. Isso tende a reduzir o custo de manutenção e o tempo de desenvolvimento do software.

Sobre a estruturação dos diretórios da parte servidor do Publicus, o código-fonte em PHP está localizado dentro do diretório `app` (Figura 58). Ele contém o arquivo `index.php` e os subdiretórios `models` e `services`. O arquivo index.php recebe as requisições da API provenientes da parte cliente e as encaminha para o *service* adequado. Por sua vez, um arquivo *service* no diretório `services` encaminha a requisição para o arquivo de modelo adequado, após fazer as devidas verificações, tais como garantir que o usuário esteja logado e que ele possua o perfil autorizado para realizar tal requisição. Cabe aos arquivos do modelo, a implementação das classes de domínio do software Publicus com a maior parte da lógica de negócio.

Todos os arquivos da API do Publicus já existiam antes do desenvolvimento deste trabalho. Entretanto, com a implementação do módulo de visitas do sistema, foi necessário refatorar e produzir novas funcionalidades nos seguintes arquivos: `Agendamento.php`, `Visita.php` e `VisitaService.php`. Apesar de `Agendamento.php` não fazer parte diretamente do módulo de visitas, foi necessário realizar uma correção para a funcionalidade de buscar agendamento, para que fosse possível identificar agendamentos confirmados para serem associados às visitas.

Já nos arquivos `Visita.php` e `VisitaService.php` foram realizadas modificações nas funcionalidades buscar e remover visitas, além do desenvolvimento de consultar visita e buscar agendamentos que não possuem uma visita associada.

Figura 58: Estrutura de arquivos da parte servidor do software Publicus



Nas Figuras 59 e 60, destacados na cor azul, estão os arquivos em que houve contribuição deste autor durante o desenvolvimento do módulo de visitas do Publicus.

Figura 59: Arquivos da pasta models

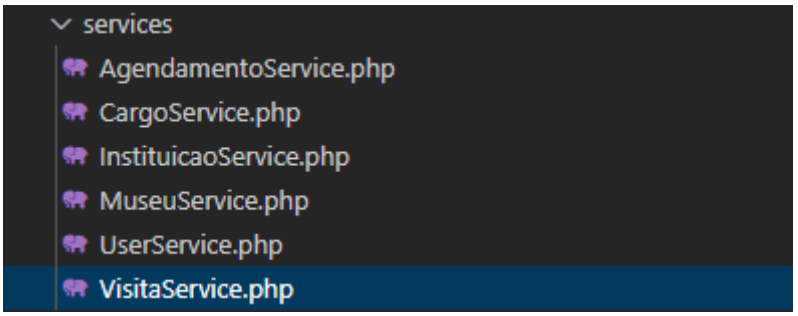
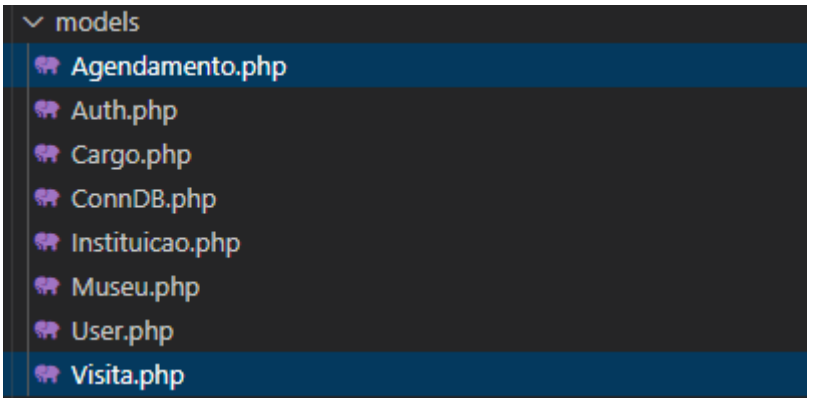


Figura 60: Arquivos da pasta services



5 TESTES AUTOMATIZADOS DO MÓDULO DE VISITAS DO SOFTWARE PUBLICUS

Esta seção apresenta como foram planejados, desenvolvidos e executados os testes no módulo de visitas do software Publicus realizados neste trabalho. Todos os testes executados foram testes de sistema, que verificam o funcionamento adequado das funcionalidades do sistema integrado, desde a interface com usuário até o banco de dados. Os testes de sistema podem ser realizados de forma manual ou automatizada. A fim de garantir eficiência na execução, optou-se pela estratégia de realizar a implementação de testes automatizados utilizando a biblioteca Flutter Test (equivalente às ferramentas de teste Selenium e Cypress), com a linguagem de programação Dart. Deste modo, é fácil e rápido repetir a execução dos testes automatizados de sistema quantas vezes forem necessárias durante a manutenção do Publicus.

Foram planejados testes automatizados para os seguintes casos de uso: Cadastrar Visita (CSU05), Buscar Visita (CSU01), Editar Visita (CSU06), Remover Visita (CSU03) e Detalhar Visita (CSU02). Assim como nas descrições textuais dos casos de uso, o projeto Publicus não contava com nenhuma documentação dos casos de testes. Logo, todos os casos de testes apresentados a seguir foram planejados e documentados durante a realização deste trabalho.

Os Quadros 9, 10 e 11 apresentam os planos de teste elaborados para o caso de uso Cadastrar Visita e seus fluxos alternativos.

Quadro 9: Caso de Teste 1 para cadastrar visitas avulsas

ID	**1**	
Caso de Uso	Cadastrar Visita (CSU05)	
Objetivo	Verificar se a função de cadastrar uma nova visita avulsa está executando corretamente	
Pré-Condições	● O usuário deve estar logado no sistema; ● O usuário logado deve ser pedagogo/recepcionista;	
Passo	**Ação**	**Resultado**
1	Selecionar a opção “Visita” no menu lateral	Direcionar o usuário para a página de visitas

2	Clicar no botão “Cadastrar visita”	Direcionar o usuário a página de cadastro de visita avulsa
3	Forneça os valores de visita (número de visitantes = 3, número de crianças = 1, número de visitantes com NE = 1, tipo de necessidade = deficiência auditiva, recurso de acessibilidade = intérprete de libras)	O sistema retorna uma mensagem que a visita foi cadastrada com sucesso
4	Verifique se a visita cadastrada está aparecendo no carrossel de Últimas Visitas Cadastradas	A visita foi adicionada com sucesso e aparece no carrossel

Quadro 10: Caso de Teste 2 para cadastrar visitas em grupo

ID	**2**	
Caso de Uso	Cadastrar Visita (CSU05)	
Objetivo	Verificar se a função de cadastrar uma nova visita em grupo sem agendamento está executando corretamente	
Pré-Condições	● O usuário deve estar logado no sistema; ● O usuário logado deve ser pedagogo/recepcionista;	
Passo	**Ação**	**Resultado**
1	Selecionar a opção “Visita” no menu lateral	Direcionar o usuário para a página de visitas
2	Clicar no botão “Cadastrar visita”	Direcionar o usuário a página de cadastro de visita avulsa
3	Clicar no botão segmentado “Visita em grupo”	Direcionar o usuário a página de cadastro de visita em grupo
4	Clicar no botão “instituição ainda não cadastrada”	Abrir o modal de cadastro de instituição
5	Forneça os valores de instituição(nome da instituição = Lógico, tipo = educação formal, gestão = privada, cidade = Natal, bairro = Tirol , cep = 59020-330 , rua e	O sistema retorna uma mensagem que a instituição foi cadastrada com sucesso

	número = R. Alberto Maranhão, 942)	
6	Clicar no botão “responsável ainda não cadastrado”	Abrir o modal de cadastro de responsável
7	Forneça os valores de responsável(instituição = Lógico, cargo = Coordenador, nome = Bernardo Lucca, telefone = (84) 99999-9999, email = bernardolucca@gmail.com, login = bernardolucca, nível de escolaridade = Pós-graduação, curso = Pedagogia, senha = 12345678 , confirmação de senha = 12345678)	O sistema retorna uma mensagem que o responsável foi cadastrado com sucesso
8	Forneça os valores de visita (tipo de grupo = educação formal, instituição = Lógico, responsável = Bernardo Lucca, nível de ensino = educação jovens e adultos, faixa etário = 15-17 e 18-25, número de visitantes = 20, número de crianças = 5, número de visitantes com NE = 1, tipo de necessidade = deficiência auditiva, recurso de acessibilidade = intérprete de libras)	O sistema retorna uma mensagem que a visita foi cadastrada com sucesso
9	Verifique se a visita cadastrada está aparecendo no carrossel de Últimas Visitas Cadastradas	A visita foi adicionada com sucesso e aparece no carrossel

Quadro 11: Caso de Teste 3 para cadastrar visitas em grupo com agendamento confirmado

ID	**3**
Caso de Uso	Cadastrar Visita (CSU05)
Objetivo	Verificar se a função de cadastrar uma nova visita em grupo com agendamento está executando corretamente

Pré-Condições	● O usuário deve estar logado no sistema; ● O usuário logado deve ser pedagogo/recepcionista; ● A instituição utilizada deve estar previamente cadastrada; ● O responsável da visita deve estar previamente cadastrado; ● O agendamento da visita deve estar previamente cadastrado; ● O agendamento da visita deve estar cadastrado para o dia atual, com a instituição Lógico marcado às 14:30;	
Passo	**Ação**	**Resultado**
1	Selecionar a opção “Visita” no menu lateral	Direcionar o usuário para a página de visitas
2	Clicar no botão “Cadastrar visita”	Direcionar o usuário a página de cadastro de visita avulsa
3	Clicar no botão segmentado “Visita em grupo”	Direcionar o usuário a página de cadastro de visita em grupo
4	Clicar no agendamento da instituição Lógico marcado para 14:30	Preencher os campos do formulário de acordo com os dados do agendamento
5	Forneça os valores de visita (número de visitantes = 18)	O sistema retorna uma mensagem que a visita foi cadastrada com sucesso
6	Verifique se a visita cadastrada está aparecendo no carrossel de Últimas Visitas Cadastradas	A visita foi adicionada com sucesso e aparece no carrossel

O Quadro 12 mostra o plano de teste planejado para o caso de uso Buscar Visita.

Quadro 12: Caso de Teste 4 para buscar visitas

ID	**4**
Caso de Uso	Buscar Visita (CSU01)
Objetivo	Verificar se a função de buscar uma visita está executando corretamente
Pré-Condições	● O usuário deve estar logado no sistema;

	● O usuário logado deve ser pedagogo/recepcionista; ● Pelo menos uma visita deve estar cadastrada com a data de visitação entre 03/06/2024 e 10/06/2024;	
Passo	**Ação**	**Resultado**
1	Selecionar a opção “Visita” no menu lateral	Direcionar o usuário para a página de visitas
2	Forneça os valores de busca (data inicial = 03/06/2024, data final = 10/06/2024)	Listar as visitas cadastradas dentro do período
3	Verifique se todas as visitas listadas estão entre 03/06/2024 e 10/06/2024	Apenas visitas com data de visitação entre 03/06/2024 e 10/06/2024 foram exibidas

O Quadro 13 demonstra o plano de teste desenvolvido para o caso de uso Remover Visita.

Quadro 13: Caso de Teste 5 para remover visitas

ID	**5**	
Caso de Uso	Remover Visita (CSU03)	
Objetivo	Verificar se a função de remover uma visita está executando corretamente	
Pré-Condições	● O usuário deve estar logado no sistema; ● O usuário logado deve ser pedagogo/recepcionista; ● Pelo menos uma visita deve estar cadastrada;	
Passo	**Ação**	**Resultado**
1	Selecionar a opção “Visita” no menu lateral	Direcionar o usuário para a página de visitas
2	Clique no ícone “Remover visita” da primeira visita da tabela	O sistema retorna uma mensagem informando que a visita foi removida com sucesso
3	Verifique se o número de visitas encontradas diminui em uma unidade e a primeira visita não aparece mais na tabela	A visita não aparece mais na tabela e o número de visitas encontradas diminuiu

O Quadro 14 apresenta o plano de teste pensado para o caso de uso Editar Visita.

Quadro 14: Caso de Teste 6 para editar visitas

ID	**6**	
Caso de Uso	Editar Visita (CSU06)	
Objetivo	Verificar se a função de editar uma visita está executando corretamente	
Pré-Condições	● O usuário deve estar logado no sistema; ● O usuário logado deve ser pedagogo/recepcionista; ● Pelo menos uma visita deve estar cadastrada;	
Passo	**Ação**	**Resultado**
1	Selecionar a opção “Visita” no menu lateral	Direcionar o usuário para a página de visitas
2	Clique no ícone “Editar visita” da primeira visita da tabela	O sistema direciona o usuário para a página de editar visita
3	Forneça os valores de visita (quantidade de visitantes = 5)	O sistema exibe uma mensagem dizendo que a visita foi editada com sucesso e redireciona para a página anterior
4	Verifique se a quantidade de visitantes da visita editada é igual a 5	A visita foi editada com sucesso e o número de visitantes é igual a 5

O Quadro 15 mostra o plano de teste elaborado para o caso de uso Detalhar Visita.

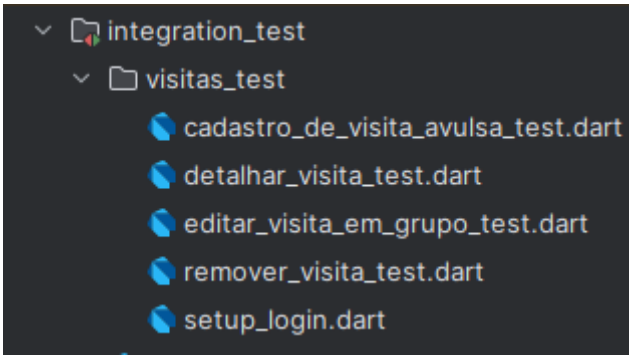
Quadro 15: Caso de Teste 7 para detalhar visitas

ID	**7**
Caso de Uso	Detalhar Visita (CSU06)
Objetivo	Verificar se a função de detalhar uma visita está executando corretamente
Pré-Condições	● O usuário deve estar logado no sistema;

	• O usuário logado deve ser pedagogo/recepcionista; • Pelo menos uma visita deve estar cadastrada;	
Passo	**Ação**	**Resultado**
1	Clique no primeiro cartão do carrossel de últimas visitas realizadas	Direcionar para o modal exibindo os detalhes da visita
2	Verifique se os dados do cartão coincidem com os dados apresentados no modal	O modal foi exibido e as informações coincidem com a do cartão clicado

Para manter uma separação adequada do código fonte do sistema, os testes automatizados estão organizados dentro do diretório `integration_test`. Além disso, os testes referentes ao módulo de visitas encontram-se no subdiretório `visitas_test`, como podemos visualizar na Figura 61.

Figura 61: Diretório `integration_test`



A fim de evitar retrabalho, foi implementado dentro do arquivo `setup_login.dart`, um método `login` que deverá ser invocado antes da execução de qualquer caso de teste. Esta função automatiza o preenchimento e envio do formulário de login, garantindo que os cenários de testes comecem com o usuário autenticado. A implementação desse método evita duplicação de código e promove consistência no ambiente de testes.

Além disso, a função de login também é responsável por inicializar o banco de dados do ambiente de testes. O método `inicializarBancoDeTeste` realiza uma requisição web a um script PHP responsável por limpar os dados antigos e popular o

banco com dados iniciais para os testes automatizados. Originalmente este script foi desenvolvido por Luiz Maia para atualizar os dados sobre agendamento de visitas. Porém, ele ainda não contemplava a atualização dos dados de visita. Então, neste trabalho foi realizada uma extensão deste script do Luiz para atualizar os dados necessários aos testes do módulo de visitas. No Publicus, estas atualizações são ainda mais importantes porque a maior parte das funcionalidades depende do dia atual e de datas próximas que foram cadastradas no banco. A utilização deste script assegura o isolamento entre os testes, fazendo com que cada um comece com o banco em um estado previsível. Dessa forma os dados inseridos em um teste não interferem nos demais, garantindo consistência e confiabilidade na execução. Na Figura 62, podemos observar a implementação da função `login`.

Figura 62: Método `login` no arquivo `setup_login.dart`.

```dart
Future<void> login(WidgetTester tester) async {
  await Test.inicializarBancoDeTeste();
  await tester.pumpAndSettle(const Duration(seconds: 2));
  var login = find.byKey(const Key('login'));
  if(login.hasFound){

    await tester.tap(find.byKey(const Key('login')));

    await tester.enterText(find.byKey(const Key('login')), 'c');

    await tester.tap(find.byKey(const Key('senha')));

    await tester.enterText(find.byKey(const Key('senha')), 'c');

    await tester.ensureVisible(find.byKey(const Key('botaoLogin')));

    await tester.tap(find.byKey(const Key('botaoLogin')));

  }

  await tester.pumpAndSettle();
}
```

No arquivo `cadastro_de_visita_avulsa_test.dart` (Figura 63) foi implementado o Caso de Teste 1, ilustrado no Quadro 9. Esse cenário de teste simula o processo de cadastro de uma visita avulsa dentro da aplicação Publicus.

Figura 63: Arquivo `cadastro_de_visita_avulsa_test.dart`.

```dart
void main() {
  IntegrationTestWidgetsFlutterBinding.ensureInitialized();
  group('cadastro de visita avulsa', () {
    testWidgets('cadastrar visita avulsa com sucesso', (WidgetTester tester) async {
      app.main();

      await setup.login(tester);

      await tester.tap(find.byKey(const Key('keyBotaoCadastrarVisita')));
      await tester.pumpAndSettle();

      await tester.enterText(find.byKey(const Key("keyCadastrarVisitaNumeroDeVisitantes")), '20');

      await tester.tap(find.byKey(const Key('keyBotaoConfirmar')));

      await tester.pumpAndSettle(const Duration(seconds: 2));

      var cards = find.byKey(const Key('keyCardVisita'));
      var dados = ((find.descendant(of: cards.first, matching: find.byType(RichText)).evaluate().first.widget as RichText).text as
      var numVisitantes = (dados?.first as TextSpan).text?.split(' • ').last;
      var visitaAvulsa = (dados?.last as TextSpan).text;

      expect(numVisitantes?.trim(), '20 visitantes');
      expect(visitaAvulsa?.trim(), 'Visita avulsa');

      await tester.tap(find.byKey(const Key('keyBotaoCancelar')));
      await tester.pumpAndSettle();

      cards = find.byKey(const Key('keyCardVisita'));
      await tester.dragUntilVisible(cards.first, find.byKey(const Key('keyScrollVertical')), const Offset(0, 500));
      await tester.pump();

      cards = find.byKey(const Key('keyCardVisita'));
      var instituicao = (find.descendant(of: cards.last, matching: find.byKey(const Key('keyCardInstituicao'))).evaluate().first.wi
      var anoENumVisitantes = (find.descendant(of: cards.last, matching: find.byKey(const Key('keyCardAnoENumVisitantes'))).evaluat

      expect(instituicao, "Visita avulsa");
      expect(anoENumVisitantes?.last.trim(), '20 visitantes');
    });
  });
```

A execução deste teste se inicia com a chamada ao método `app.main()`, que é responsável por inicializar a aplicação. Em seguida, é feita a invocação da função `login`, explicada anteriormente. Daqui em diante, o teste automatizado precisa simular as ações do usuário na interface até a conclusão do caso de teste. Para isso, utilizou-se a classe `WidgetTester` fornecida pelo framework de testes do Flutter. Ela nos permite automatizar a interação com a interface do sistema de forma programada, simulando ações de um usuário real.

Para simular o clique do usuário, utilizamos a função `tap` e passamos como parâmetro o widget a ser clicado. Também conseguimos inserir um texto, utilizando o `enterText` com a indicação de qual texto será “digitado” em qual widget. Já a função `pumpAndSettle` permite que a execução do teste espere a interface terminar de atualizar (terminar uma animação, trocar de tela, etc.) antes de prosseguir com o teste.

A fim de assegurar que o teste foi bem sucedido, utilizamos a função `expect`. Ela é encarregada por confirmar se o valor obtido após a execução corresponde ao valor esperado. Assim, um caso de teste é considerado bem sucedido se todos os valores verificados coincidem com os resultados esperados.

A execução dos testes automatizados foi realizada em um ambiente controlado, no qual as especificações podem ser encontradas no Quadro 17.

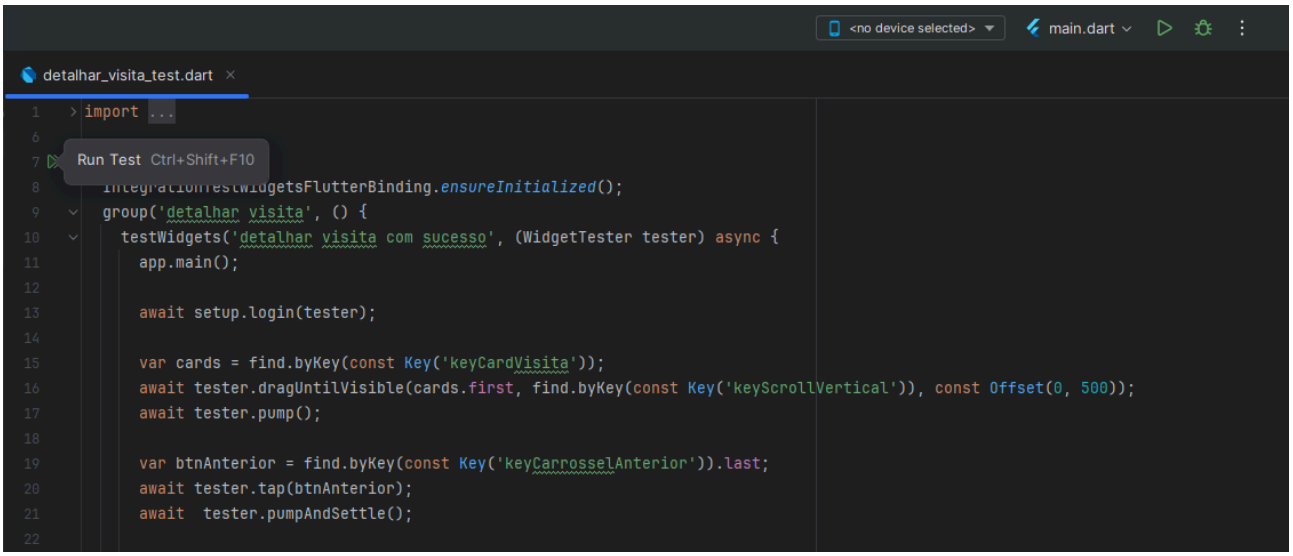
Quadro 17: Ambiente de Execução

Processador	Intel(R) Core(TM) i5-10400F CPU @ 2.90GHz 2.90GHz
Memória RAM	16GB
Sistema Operacional	Windows 10 64 bits

Os testes foram desenvolvidos e executados utilizando o ambiente de desenvolvimento Android Studio, que oferece suporte para a execução dos testes desenvolvidos utilizando a biblioteca Flutter Test.

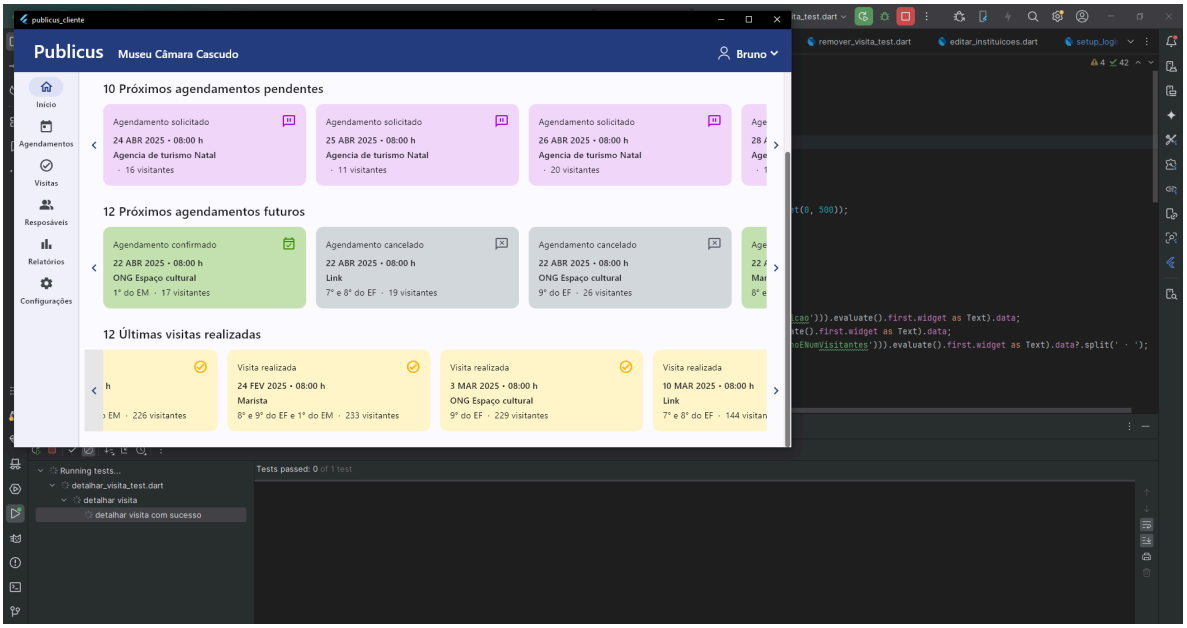
A execução de um caso de teste pode ser descrita em três etapas. A primeira etapa corresponde ao início da execução, ilustrado na Figura 64. A execução do teste é iniciada pelo comando “Run Test”, que carrega a aplicação e simula o ambiente real de uso. Além disso, o Android Studio também permite escolher em qual dispositivo o teste será executado, permitindo que os testes sejam realizados em diversos ambientes, como emuladores de smartphones, tablets e dispositivos físicos conectados.

Figura 64: Início da execução de um caso de teste.



Após a aplicação ser carregada, a segunda etapa, ilustrada na Figura 65, é iniciada com a navegação e o preenchimento dos campos de modo automático na interface, simulando o comportamento de um usuário real. É nesta etapa que as verificações do teste serão executadas, seguindo todas as instruções do caso de teste.

Figura 65: Execução de um caso de teste.



Por último, a terceira etapa exibe o resultado da execução do teste automatizado e o tempo consumido na sua execução, conforme mostrado na Figura 66.

Figura 66: Resultado da execução de um caso de teste.

O Quadro 17 mostra os resultados obtidos após a execução dos casos de teste: Cadastrar Visita Avulsa, Detalhar Visita, Editar Visita em Grupo e Remover Visita. Todos os testes foram executados para o sistema operacional Windows, porque quase todas essas funcionalidades se aplicam a apenas a versão desktop do software Publicus.

Quadro 18: Resultados dos testes automatizados

Caso de Teste	Resultado	Tempo de Execução
Cadastrar Visita Avulsa	Passou	18,072s
Detalhar Visita	Passou	17,794s
Editar Visita em Grupo	Passou	41,456s
Remover Visita	Passou	32,059s

6 CONSIDERAÇÕES FINAIS

A gestão de museus precisa acompanhar o fluxo de visitações para poder cumprir sua missão cultural e educativa na sociedade. Nesse contexto, o sistema Publicus, de propriedade da Universidade Federal do Rio Grande do Norte (UFRN), surgiu inicialmente com o intuito de auxiliar na contagem de público. A partir de sua segunda versão, o Publicus expandiu seu escopo para incluir o agendamento de visitas escolares e de outros grupos. Contudo, a gestão de visitas ainda não havia sido contemplada. Então, esse trabalho teve como objetivo contribuir com o desenvolvimento da segunda versão do software Publicus por meio da implementação do módulo para a gestão de visitas a museus.

O processo de desenvolvimento deste módulo se iniciou com a produção do diagrama de casos de uso, suas descrições textuais e do diagrama de classes, respeitando a arquitetura original do software. Em seguida, realizou-se a programação da parte cliente deste módulo com a linguagem Dart e o framework Flutter e da parte servidor com a linguagem PHP. Por fim, sete casos de testes foram planejados para este módulo, sendo 4 deles implementados com testes automatizados utilizando a biblioteca Flutter Test.

A implementação deste módulo de gestão de visitas do Publicus oferece aos museus uma ferramenta que auxilia a gestão de visitas avulsas e em grupo de diferentes tipos, incluindo visitas escolares agendadas ou não agendadas. Uma boa gestão das informações sobre visitação contribui para que os museus tenham condições de oferecer bons serviços aos seus visitantes. Para a educação, o Publicus facilita a gestão de agendamentos de visitas escolares a museus, com o potencial de fortalecer a interação entre instituições de ensino e museus e promover a aprendizagem articulada entre a educação formal e não formal.

Durante a realização deste trabalho foi possível aplicar conhecimentos aprendidos em diversas disciplinas do curso de Engenharia de Software, em especial é válido ressaltar disciplinas como Análise e Projeto Orientado a Objetos, Desenvolvimento para Dispositivos Móveis, Banco de Dados, Desenvolvimento de Sistemas Web I e II, Teste de Software I e II.

Por último, é esperado que trabalhos futuros continuem com a evolução do software Publicus. A mais curto prazo é possível dar continuidade na implementação dos

testes automatizados que foram planejados mas ainda não foram contemplados neste trabalho. Também é necessário realizar a implantação do sistema e planejar e executar uma avaliação de usabilidade com professores e funcionários do museu. Em relação a possíveis novos requisitos para o sistema, seria interessante a implementação de funcionalidades relacionadas a elaboração de relatórios de visitação e gráficos que possibilitam a análise de dados, como quantidade de escolas, quantidade de visitantes, entre outros.

REFERÊNCIAS

ALMEIDA, A. *et al.* Públicos das visitas agendadas ao Museu de História Natural e Jardim Botânico da Universidade Federal de Minas Gerais: contribuições dos registros para a gestão de museus. **Museologia e Patrimônio**, p. 239–255, 2024.

BOYLAN, P. (org.). **Como gerir um museu: manual prático**. Brodowski, São Paulo: Associação Cultural de Apoio ao Museu Casa de Portinari, Secretaria da Cultura do Estado de São Paulo, 2015.

BRASIL. Ministério da Cultura. **Bases para a Política Nacional de Museus: memória e cidadania**. Brasília, DF, 2003.

CÂNDIDO, M. M. D. **Orientações para gestão e planejamento em Museus**. Florianópolis: FCC, 2014.

CAZELLI, S. et al. Conhecer para contar: o público de museus de ciência do Rio de Janeiro. **Museologia e Patrimônio**, p. 379–408, 2022.

CAZELLI, S.; VERGARA, M. O passado e o presente das práticas de educação não formal na cidade do Rio de Janeiro. In: **Encontro de História da Educação do Estado do Rio de Janeiro**, Niterói – Rio de Janeiro. CD-ROM do I EHEd-RJ, 2007.

COOPER, A.; REIMANN, R.; CRONIN, D. **About Face 3: The Essentials of Interaction Design.** New York, NY: John Wiley & Sons, 2007.

COSTA, A. F.; NASCIMENTO, C. M. P.; MAHOMED, C.; REQUEIJO, F.; CAZELLI, S. Pensando a relação museu-escola: o MAST e os professores. In: **Encontro Nacional de Pesquisa Em Educação Em Ciências**, 2007.

IBRAM, INSTITUTO BRASILEIRO DE MUSEUS. **Regulamenta dispositivos do Decreto no 8.124/2013 quanto à obrigatoriedade do envio ao Instituto Brasileiro de Museus do quantitativo anual de visitação dos museus brasileiros.** 2021. Disponível em: https://www.gov.br/museus/pt-br/assuntos/legislacao-e-normas/portarias/portaria-ibram-no-291-de-13-de-abril-de-2021. Acesso em: 23 set. 2024.

INSTITUTO BRASILEIRO DE MUSEUS (IBRAM). **Política Nacional de Educação Museal (PNEM)**. Brasília, DF, 2017.

KÖPTCKE, L. S. Revisitando a parceria museu-escola: currículo e formação profissional. **Museologia e Patrimônio**, vol.7, n. 2, p. 15-35, 2014.

MARANDINO, M. Interfaces na relação museu-escola. **Caderno Brasileiro de Ensino de Física**, v. 18, n. 1, p. 85–100, 2001.

MARANDINO, M.; CONTIER, D. (org.). **Educação Não Formal e Divulgação em Ciência: da produção do conhecimento a ações de formação**. São Paulo: Faculdade de Educação da USP, 2015.

PAIVA, V. H. F. **Projeto de artefato digital para agendamento de visitas escolares a museus**. 2023. 320 f. Trabalho de Conclusão de Curso (Graduação em Design) - Departamento de Design, Universidade Federal do Rio Grande do Norte, Natal, 2023.

PAIVA, V. H. F.; SILVA, B. S. Que informações são importantes no agendamento de visitas escolares a museus?. **Revista ACB**, v. 28, n. 1, p. 1–20, 2024.

PEREIRA, W. A.; SILVA, J. B.; REIS, D. A. Investigando as relações entre as práticas em espaços de educação não formal e formal. **Revista Cocar**, v. 15, n. 32, 2021.

REGO, T. C. **Vygotsky: uma perspectiva histórico-cultural da educação**. Petrópolis: Vozes, 2013.

RODRIGUES, L. T.; MIGUEL, M. C.; ALDABALDE, T. V. Mediação em Museus: mapeando o tema e identificando lacunas. **Revista Conhecimento em Ação**, p. 209–233, 2022.

SANTOS, S. S. Espaços Educativos Científicos: Formal, Não Formal e Informal. **Revista Areté | Revista Amazônica de Ensino de Ciências**, v. 9, n. 20, p. 98–107, 2016.

SEGATTO, F. Z.; OLIVEIRA, R. C. Relato de experiência docente em espaços não formais e o planejamento do professor. **Revista de Ensino de Biologia da SBEnBio**, p. 188–209, 2022.

SILVA, B. S.; MEDEIROS, C. M. L. A diversidade do público escolar que visita o Museu Câmara Cascudo. **Museologia & Interdisciplinaridade**, v. 10, n. 20, p. 191–208, 2021.

SILVA, B. S.; PAIVA, V. H. F. **Informações solicitadas no agendamento de visitas escolares em websites de museus brasileiros**. Informação em Pauta, Fortaleza, v. 8, p. 1-16, 2023.

SILVA, B. S.; PAIVA, V. H. F. **Oportunidades de melhoria no processo de agendamento de visitas escolares no museu câmara cascudo**. Pontodeacesso, v. 16, n. 1, p. 156-174, 20 jun. 2022.

SILVA, C. S.; DINIZ, R. E. S. Perfil e prática pedagógica dos professores visitantes de um centro de ciências: indicativos sobre a relação museu-escola. In: **XIII Encontro Nacional de Pesquisa em Educação em Ciências**, 2011.

TRILLA, J. **La educación fuera de la escuela: ámbitos no formales y educación social**. Grupo Planeta (GBS), 2003.

WAZLAWICK, R. S. **Análise e design orientados a objetos para sistemas de informação: modelagem com UML, OCL e IFML.** 3. ed. Rio de Janeiro: Elsevier, 2015.