UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

THAIS CRISTINA BERTOLDO

SISTEMA PARA AGENDAMENTOS DE HORÁRIOS EM BARBEARIAS

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO

2019

THAIS CRISTINA BERTOLDO

SISTEMA PARA AGENDAMENTOS DE HORÁRIOS EM BARBEARIAS

Monografia de Especialização, apresentada ao Curso de especialização em Tecnologia Java, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Especialista.

Orientadora: Profa. Andreia Scariot Beulke

PATO BRANCO
2019

MINISTÉRIO DA EDUCAÇÃO
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Especialização em Tecnologia Java

TERMO DE APROVAÇÃO

SISTEMA PARA AGENDAMENTOS DE HORÁRIOS EM BARBEARIAS

por

THAIS CRISTINA BERTOLDO

Este trabalho de conclusão de curso foi apresentado em 17 de dezembro de 2019, como requisito parcial para a obtenção do título de especialista em Tecnologia Java. Após a apresentação o candidato foi arguido pela banca examinadora composta pelos professores Andreia Scariot Beulke (orientadora), Mariza Miola Dosciatti e Vinicius Pegorini, membros de banca. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

_____
Andreia Scariot Beulke
Profa. Orientadora (UTFPR)

_____
Mariza Miola Dosciatti
(UTFPR)

_____
Vinicius Pegorini
(UTFPR)

_____
Vinicius Pegorini
Coordenador do curso

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

Os homens estão investindo mais em sua aparência e, com isso, o segmento de barbearias está se expandindo. Existem barbearias para os públicos de alto, médio e pequeno porte, sendo que a tendência é que esse mercado cresça cada vez mais no Brasil. Com a ascensão do mercado de beleza masculina, aumentou significativamente o número de barbearias e os clientes estão buscando ambientes tecnológicos, descontraídos, com qualidade no serviço prestado, locais com *games*, atendimento *on-line* e com boa reputação no meio digital. Este trabalho propõe a codificação de um sistema que permita o agendamento *on-line* pelo cliente e o gerenciamento de horários dos agendamentos pelos barbeiros. O sistema possibilita o cadastro de clientes, funcionários, serviços e horários de agendamentos. O desenvolvimento desse sistema permitiu a integração de diversas tecnologias. Dentre essas tecnologias no *back-end* destacam-se a linguagem de programação Java, juntamente com alguns módulos do *framework* Spring objetivando simplificar a programação e a implementação da *API REST*. Para o *front-end* foi utilizado o React que é uma biblioteca JavaScript de código aberto com foco em criar interfaces de usuário em páginas web. Na parte de segurança foi aplicado o *JSON Web Token (JWT)* que é um método RCT 7519 padrão da indústria para realizar autenticação entre duas partes por meio de um *token* assinado que autentica uma requisição web e é uma das formas mais utilizadas para autenticar usuários em APIs RESTful. Para possibilitar a atualização em *realtime* foi utilizado no *back-end* o módulo de *WebSocket* do *framework* Spring e no *front-end* foi utilizado o *stomp-websocket* com o *sockjs-client*.

Palavras-chave: Barbearia. ReactJS. ReactNative. Spring. Java. WebSocket. JWT. Hooks. ES6. REST.

ABSTRACT

Men are investing more in their appearance and, with this, the barbershop segment is expanding. There are barbershops for whole range of target market, and the trend is that this market will grow more and more in Brazil. With the rise of the male beauty market, the number of barbershops has significantly increased and customers are seeking for technological, relaxed environments, with quality in the service provided, places with games, online service and with good reputation in the digital scene. This work proposes the codification of a system that allows online scheduling by the client and the management of schedules by barbers. The system enables the registration of customers, employees, services and schedules. The development of this system allowed the integration of several technologies. Among these technologies in the back-end stand out the Java programming language, along with some modules of the Spring framework aiming to simplify the programming and implementation of the REST API. For the front end, react was used which is an open source JavaScript library focused on creating user interfaces on web pages. On the security part, the JSON Web Token (JWT) was applied which is an industry standard RCT 7519 method for performing two-part authentication through a signed token that authenticates a web request and is one of the most commonly used ways to authenticate users in RESTful APIs. To admit realtime updating, the Spring framework WebSocket module was used in the back-end and the stomp-websocket with sockjs-client was used in the front-end.

Keywords: Barber. ReactJS. ReactNative. Spring. Java. WebSocket. JWT. Hooks. ES6. REST

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

API	*Application Programming Interface*
DOM	*Document Object Model*
DTO	*Data transfer object*
JWT	*JSON Web Token*
REST	*Representational State Transfer*
UIs	*User Interface*
URL	*Uniform Resource Locator*

LISTAGEM DE CÓDIGOS

SUMÁRIO

1 INTRODUÇÃO

Este capítulo é composto pelas considerações iniciais que apresentam o contexto no qual se insere o sistema desenvolvido e pelos objetivos e a justificativa do trabalho, além de ser finalizado com a organização do texto que se refere à apresentação dos capítulos posteriores.

1.1 CONSIDERAÇÕES INICIAIS

Os homens estão mais vaidosos e investindo cada vez mais na sua aparência. Alguns anos atrás isso era visto com frequência no público feminino. Em 2016 eles foram responsáveis pelo investimento de quase R$ 20 bilhões no setor de beleza, sendo que até 2021, o Brasil deve liderar as vendas do mercado mundial do segmento de beleza masculina segundo o Sebrae (2018).

O comportamento do homem contemporâneo, com novos hábitos e cuidados pessoais no que se refere à sua aparência, fez com que as barbearias precisassem se adaptar para melhor atendê-los, pois eles prezam por serviços de qualidade, com técnicas modernas, profissionais experientes, bons produtos e, ainda, algumas barbearias oferecem serviços adicionais como, por exemplo, videogames e bebidas para atrair o cliente.

Com o avanço e acessibilidade da tecnologia, aliado ao perfil do homem moderno, é importante que as barbearias invistam, também, em sistemas que permitam proporcionar maior agilidade no atendimento aos seus clientes, como por exemplo, o agendamento *on-line* dos serviços. Devido a correria do mundo moderno este público busca praticidade e agilidade no dia-a-dia, podendo realizar o agendamento na hora que desejar e não apenas no horário em que o estabelecimento esteja em funcionamento.

Com o advento da lei nº 13.352/2016 (REPÚBLICA, 2016) que possibilitou a parceria entre os profissionais e os estabelecimentos, sem vínculos empregatícios, sendo definida em contrato a cota-parte percentual destinada ao estabelecimento parceiro, foi responsável por fomentar o mercado do micro-empreendedorismo individual, facilitando a abertura de novas barbearias e o compartilhamento de espaços.

Com o aquecimento deste mercado e a busca por melhoria nos serviços disponibilizados aos clientes visando a fidelização dos mesmos, surge a oportunidade para a comercialização de softwares que auxiliem neste processo.

O compartilhamento de espaços por vários barbeiros faz com que o controle de agendamentos no papel torne-se complexo para ser compartilhado. Em meio a este cenário, o desenvolvimento deste trabalho propõe-se o desenvolvimento de um sistema *web* que possibilite o controle de horários dos agendamentos de uma barbearia, visando auxiliar e automatizar os processos de agendamento dos clientes.

O sistema desenvolvido proporciona aos barbeiros e demais profissionais o compartilhamento de suas agendas. O agendamento *on-line* disponibilizado aos clientes traz comodidade e disponibilidade.

1.2 OBJETIVOS

A seguir são apresentados os objetivos deste trabalho, organizados em objetivo geral e objetivos específicos.

1.2.1 OBJETIVO GERAL

Implementar um sistema *web*, complementado por versão *mobile*, para o controle de agendamento de horários em barbearias.

1.2.2 OBJETIVOS ESPECÍFICOS

- Automatizar o agendamento de horários;
- Facilitar o acesso aos agendamentos para os barbeiros por meio do sistema *web*;
- Facilitar o agendamento de horários pelo cliente por meio do aplicativo móvel.

1.3 JUSTIFICATIVA

As barbearias estão passando por transformações, e além de oferecer serviços e produtos de qualidade, necessitam proporcionar aos seus clientes comodidade, visando a fidelização devido à alta concorrência. Para isso, é fundamental que as barbearias invistam em tecnologias que possam auxiliar em suas atividades e funções.

Tendo em vista este cenário, o trabalho em questão se justifica pela proposta de oferecer aos clientes a possibilidade de realizar agendamentos e consultas *on-line* por meio do aplicativo móvel, trazendo comodidade para o mesmo. Para os barbeiros o sistema *web* irá proporcionar agilidade no controle de agendamentos de horários da barbearia.

O uso de *WebSockets* para realizar a atualização em tempo real da agenda, se justifica pelo fato desta tecnologia permitir ao servidor a realização de envio dos dados ao cliente no mesmo momento em que ocorram atualizações, sem a necessidade de aguardar uma requisição do cliente. O React foi escolhido como tecnologia de desenvolvimento do *front-end*, pois está em constante evolução e facilita o desenvolvimento de interfaces de usuário interativas, além disso, é altamente performático. O React Native se apresenta como uma boa alternativa no desenvolvimento de aplicativos para múltiplas plataformas, visto que, consegue oferecer uma melhor integração entre as funções nativas do aparelho, em razão de ele não converter o código para para linguagens nativas, mas realizar a comunicação com o código nativo.

1.4 ESTRUTURA DO TRABALHO

Este texto está organizado em capítulos e este é o primeiro que apresenta a introdução, as considerações iniciais, os objetivos e a justificava de realização deste trabalho.

O Capítulo 2 apresenta os materiais e métodos utilizados para o desenvolvimento do trabalho.

No Capítulo 3 está o resultado da realização do trabalho que apresenta a modelagem, a apresentação e a implementação do sistema.

A conclusão é apresentada no Capítulo 4 e, por fim, estão as referências citadas no texto.

2 MATERIAIS E MÉTODO

Este capítulo apresenta as ferramentas e tecnologias utilizadas no desenvolvimento deste trabalho e os procedimentos técnicos necessários para utilizá-las.

2.1 FERRAMENTAS E TECNOLOGIAS

A Tabela 1 apresenta as ferramentas e as tecnologias utilizadas na modelagem e no desenvolvimento deste projeto.

Tabela 1 – Materiais utilizados no desenvolvimento do sistema

Ferramenta / Tecnologia	**Versão**	**Disponível em**	**Aplicação**
React	16.11.0	https:/reactjs.org	Biblioteca JavaScript
Material-UI	4.5.1	https:/material-ui.com/getting-started/installation	Coleção de componentes React para desenvolvimento *Web*
FullCalendar	4.3.0	https:/fullcalendar.io/docs/getting-started	Componente de calendário JavaScript para exibição de eventos
Axios	0.19.0	https:/github.com/axios/axios	Cliente HTTP baseado em *Promises* utilizado para realizar as chamadas HTTP à API's

React Async	0.27.0	https://docs.react-async.com/getting-started/usage	Biblioteca baseada em *promisse* que permite a busca de dados e falicita o manuseio de estados assíncronos da interface do usuário *hook*
Moment.js	2.24.0	https:/momentjs.com	Biblioteca para converter, validar, manipular e exibir datas e horas em JavaScript
Formik	1.5.8	https:/jaredpalmer.com/formik/docs/overview	Biblioteca que facilita o manuseio de formulários em React
Yup	0.27.0	https:/github.com/jquense/yup	Biblioteca para validação de formulários Javascript
Styled Components	4.4.0	https://www.styled-components.com/docs	Biblioteca React que permite a utilização de estilos a nível de componente na aplicação
Stomp Websocket	2.3.4	http:/jmesnil.net/stomp-websocket/doc/	Protocolo de mensagens orientado a texto

Sockjs Client	1.4.0	https:/github.com/sockjs/sockjs-client	Biblioteca JavaScript do navegador que fornece um objeto semelhante ao WebSocket
React Native	0.61.4	https:/facebook.github.io/react-native	*Framework* para desenvolvimento nativo de aplicativos
React Native Paper	3.2.1	https:/callstack.github.io/react-native-paper/getting-started.html	Coleção de componentes personalizáveis seguindo as orientações do *Material Design* para React Native
AsyncStorage	1.6.3	https:/github.com/react-native-community/async-storage	API nativa do React Native, utilizada para armazenar dados persistentes no dispositivo de forma assíncrona
Spring Boot	2.2.0	https:/spring.io/projects/spring-boot	Uso de convenções sobre configurações
Spring Websocket	2.2.0	https:/spring.io/guides/gs/messaging-stomp-websocket	Protocolo STOMP utilizado com o Spring para criação de aplicativo da web interativo

Spring Data	2.2.0	https:/spring.io/projects/spring-data	*Framework* de implementação de repositórios
Spring Security	5.2.0	https:/spring.io/projects/spring-security	*Framework* de autenticação e segurança da aplicação
Java *JWT*	0.9.0	https:/github.com/jwtk/jjwt	Biblioteca para criar e validar *JSON Web Tokens (JWTs)* em Java
Commons Email	1.2	https:/commons.apache.org/proper/commons-email/	API para envio de e-mail
PostgreSQL	10	https:/www.postgresql.org/	Banco de dados
PgAdmin	4	https:/www.pgadmin.org/	Administrador do banco de dadoss
JAVA	8	https:/www.java.com	Linguagem de programação
Visual Studio Code	1.40.2	https:/code.visualstudio.com/	Ambiente de desenvolvimento
Eclipse	4.12.0	https:/www.eclipse.org	Ambiente de desenvolvimento
Astah Community	8.2	http:/astah.net/download	Modelagem do sistema

PGModeler	0.9.1	https:/pgmodeler.io/download	Modelagem do DER do sistema

Fonte: Autoria própria (2019).

2.1.1 REACT

React é uma biblioteca JavaScript de código aberto utilizada para criar interfaces de usuário, que é mantida pelo Facebook. Seu objetivo é tornar fácil a tarefa de criação de interfaces de usuários interativas. Para Prata (2019) um dos fatores que torna isso possível é o fato de ele usar o paradigma de programação declarativa, fazendo com que seja necessário se preocupar mais com "o que" e menos com o "como". Além disso, a programação declarativa torna o código mais previsível e fácil de *debugar*.

O React é baseado em componentes e, com isso, permite que cada componente seja responsável por controlar e gerenciar seu próprio estado, possibilitando a combinação de vários componentes para criar *User Interface (UIs)* complexas.

O React possui desempenho satisfatório, uma das estratégias que ele possui para tornar as aplicações naturalmente performáticas é a utilização do conceito de Virtual Document Object Model (DOM) que permite a alteração no DOM real somente no que mudou de fato no DOM virtual, para isso, ele compara o DOM Virtual que está na memória com a nova versão.

O React é uma biblioteca JavaScript e não um *framework*, ou seja, não é uma solução completa, por isso, frequentemente há a necessidade de utilizar outras bibliotecas junto com o React para formar uma solução.

Ele vem se tornando popular, e o que reforça isso é o número de contribuidores no *github*, demostrando o motivo de estar presente em diversas aplicações.

2.1.2 REACT NATIVE

O React Native é um *framework* baseado no React criada pela equipe do Facebook. Ele proporciona o desenvolvimento de aplicações *mobile*, tanto para Android, como para iOS com interface nativa (não híbrida como no Ionic) utilizando apenas JavaScript .

Fernandes (2017) expõe que no React Native o código criado em JavaScript não é convertido para linguagens nativas, como Java e Objective-C, mas entendido

pelo dispositivo móvel por meio de uma ponte criada pelo React Native entre código nativo e o JavaScript. Com isso, consegue oferecer ao usuário uma experiência mais fluída, melhor integração entre funções do celular como câmera, além de obter ganho de performance.

O React Native foi lançado em 2015 e vem tendo um grande avanço na sua utilização. Ele está em constante evolução, e de acordo com Becker (2019) vem apresentando uma comunidade de contribuidores ativa e numerosa, além de diversas bibliotecas auxiliares que reduzem o tempo de desenvolvimento.

2.1.3 SPRING

O Spring é um *framework* de código aberto muito utilizado na plataforma Java e que foi criado por Rod Johnson. Ele é baseado nos padrões de projeto de inversão de controle e injeção de dependência.

O objetivo do Spring Boot é fornecer um conjunto de ferramentas para a construção rápida de aplicativos que sejam fáceis de configurar.

O Spring é um *framework* que tráz diversos benefícios às aplicações, aumentando a produtividade no desenvolvimento além de promover um grande aumento de performance em tempo de *runtime*, e facilitar o trabalho com testes unitários (SCHITINI, 2011 apud CARDOSO, 2018).

O Spring é composto por recursos organizados em cerca de 20 módulos, que podem ser implementados separadamente ou em conjunto com outros. Isto permite ao Spring ser aplicado nos mais variados tipos de aplicações.

3 RESULTADOS

Este capítulo apresenta o resultado da realização deste trabalho. O capítulo inicia com a descrição do escopo do sistema, seguido da modelagem, apresentação e implementação do sistema proposto.

3.1 ESCOPO DO SISTEMA

O sistema para gerenciamento de horários em barbearias possibilita o cadastro de serviços, clientes e profissionais e o gerenciamento em tempo real da agenda dos barbeiros.

No cadastro de serviços é possível informar o nome, a duração, o valor e a descrição dos serviços realizados pelos barbeiros.

No cadastro de funcionários são armazenados dados de identificação e o perfil de acesso ao sistema, além da possibilidade de informar que o profissional possui agenda e configurar os horários de trabalho e serviços realizados pelo profissional.

O cadastro do estabelecimento disponibilizará os campos de horário de funcionamento, nome do estabelecimento e informações exibidas no agendamento realizado pelo cliente.

O controle de agendamento permitirá a realização de agendamentos de horários para os clientes, possibilitando a escolha de um ou mais serviços, além da possibilidade de bloqueio de horários na agenda do profissional. Além disso, as atualizações são realizadas em tempo real.

O cliente pode ser cadastrado por meio do sistema *web* ou realizando seu próprio cadastro utilizando o aplicativo *mobile*. Para realizar o cadastro é necessário informar os dados de identificação e de acesso ao aplicativo.

Os agendamentos podem ser realizados pelo sistema *web* por meio do acesso dos funcionários, ou então, pelo cliente no aplicativo *mobile*. Para realizar o agendamento ele precisa escolher o serviço desejado, profissional e horário disponível.

3.2 MODELAGEM DO SISTEMA

A Tabela 2 apresenta a descrição dos requisitos funcionais e a Tabela 3 os requisitos não funcionais do sistema.

Tabela 2 – Requisitos funcionais

Identificação	Nome	Descrição
RF 01	Manter estabelecimento	Cadastro dos dados da barbearia.
RF 02	Manter serviços	Cadastro dos serviços prestados pela barbearia.
RF 03	Manter funcionários	Cadastro dos funcionários, contendo os dados necessários para identificação, informações de jornada de trabalho, serviços prestados, dados e perfil de acesso ao sistema que podem ser administrador ou funcionário.
RF 04	Manter clientes	Cadastro dos clientes, contendo os dados necessários para identificação e dados de acesso ao aplicativo.
RF 05	Manter agenda	Cadastro de agendamentos e de bloqueios.
RF 06	Realizar agendamento	Cadastro de agendamentos de serviços pelos funcionários e/ou pelos clientes.
RF 07	Atualizar perfil	Permite aos funcionários e clientes atualizar suas informações cadastrais e de acesso.

Fonte: Autoria própria (2019).

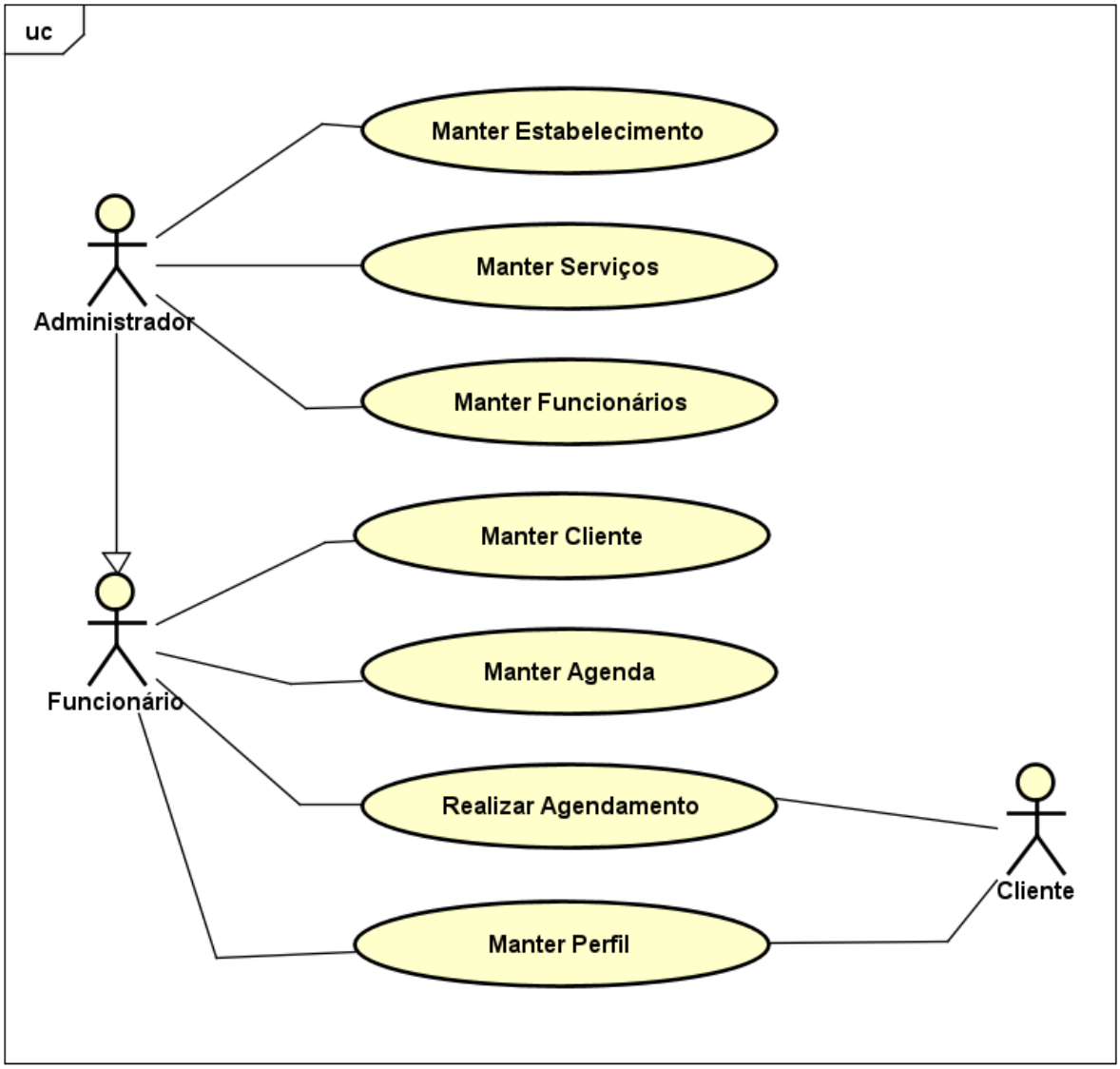
Tabela 3 – Requisitos não-funcionais

Identificação	Nome	Descrição
RNF 01	Acesso ao sistema	O acesso ao sistema será realizado por meio de login e senha. Apenas o acesso de funcionários e administradores deve ser permitido.
RNF 02	Acesso ao aplicativo	O acesso ao aplicativo será realizado por meio de login e senha. Apenas o acesso de clientes deve ser permitido.
RNF 03	Restrições de formato	Os campos de *e-mail* e senha devem ser válidos de acordo com as restrições de formato.
RNF 04	Campos de preenchimento obrigatórios	Os campos que são de preenchimento obrigatório deverão serão validados.
RNF 05	Exclusão	Não deve ser permitida a exclusão de cadastros com vínculos.
RNF 06	Agenda	A atualização da agenda dos profissionais deve ser em tempo real.

Fonte: Autoria própria (2019).

O diagrama de casos de uso apresentado na Figura 1 contém as funcionalidades essenciais do sistema realizadas pelos seus atores que são: administrador, funcionário e cliente. O administrador é responsável pelos cadastro dos dados do estabelecimento, serviços, usuários, clientes, agendamentos e atualizar seu perfil. O funcionário é quem realiza o processo de cadastro de clientes, agendamentos e atualiza seu perfil. O cliente poderá efetuar seu cadastro por meio do aplicativo e realizar agendamentos, bem como atualizar seu perfil.

Figura 1 – Diagrama de casos de uso

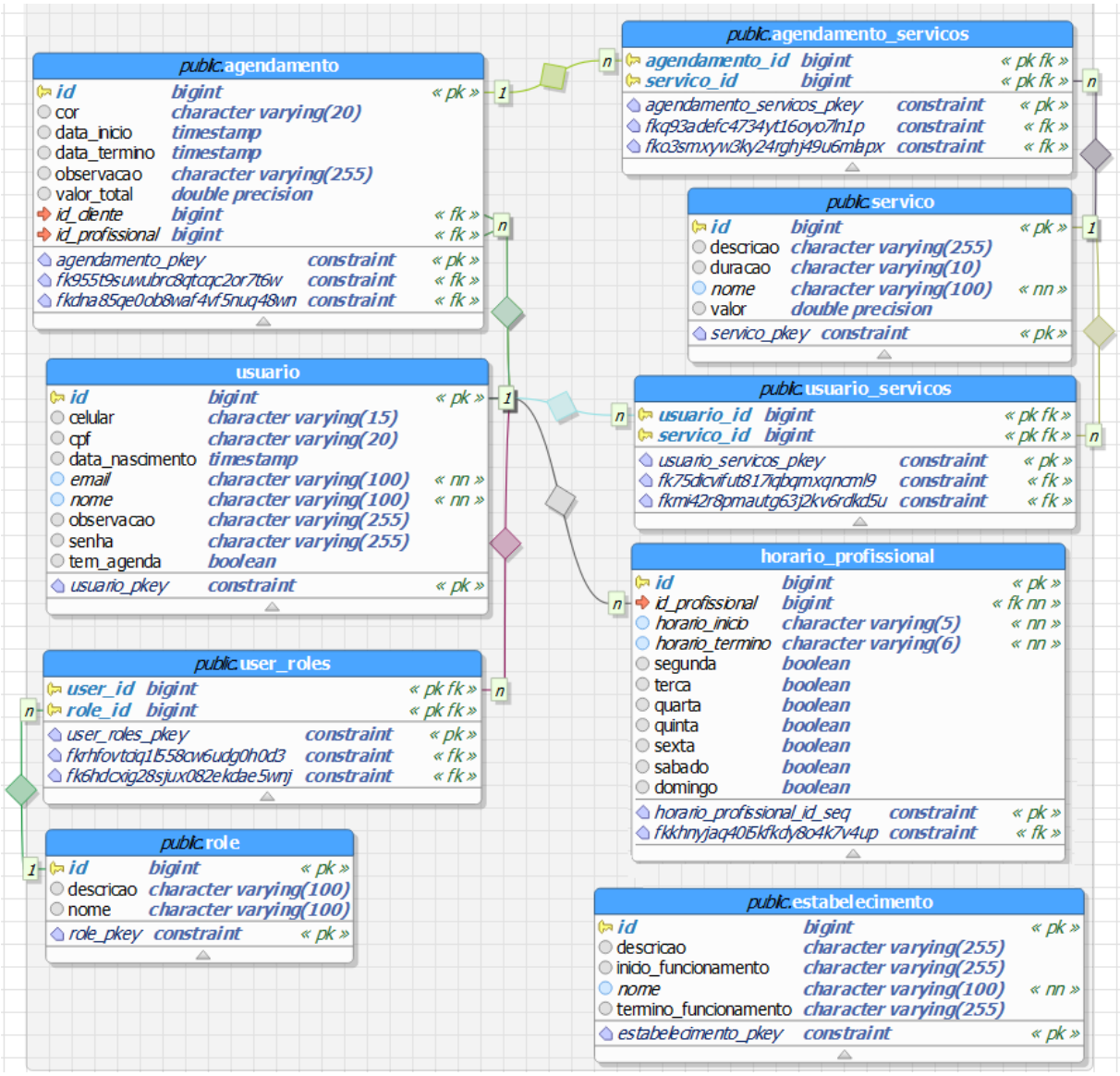


Fonte: Autoria própria (2019).

3.2.1 DIAGRAMA DE ENTIDADE E RELACIONAMENTO (DER)

Na Figura 2 são exibidos os relacionamentos entre as entidades do sistema, os tipos e os tamanhos de cada atributo.

Figura 2 – Diagrama de Entidade e Relacionamento (DER)



Fonte: Autoria própria (2019).

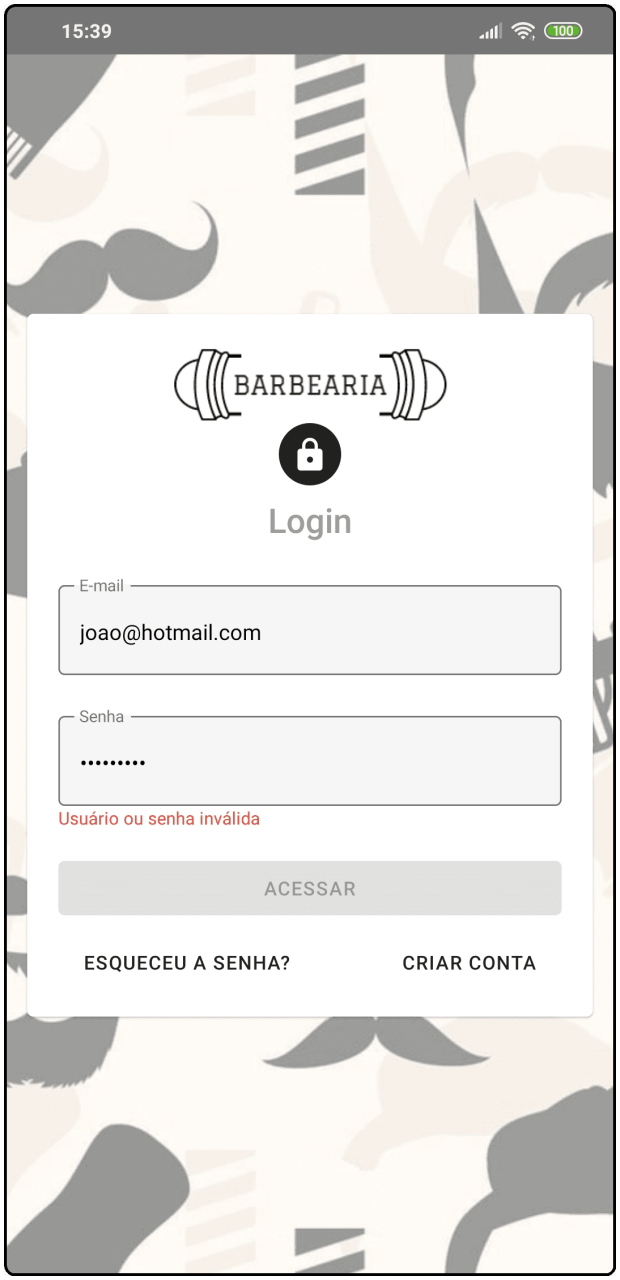
A tabela de usuário é responsável por armazenar os dados e permissões dos clientes e funcionários. A tabela de serviços armazena as informações dos serviços realizados pela barbearia. A tabela de agendamentos armazena os dados dos agendamentos realizados, além do cliente e profissional. Já a tabela de agendamentos serviços é responsável por armazenar os serviços vinculados aos agendamentos. A tabela de estabelecimentos armazena os dados básicos da barbearia, com por exemplo horário de funcionamento.

3.3 APRESENTAÇÃO DO SISTEMA

Essa seção apresenta a descrição das principais funcionalidades que poderão ser visualizadas por meio de *prints* das telas do sistema.

A Figura 3 exibe a tela de autenticação do aplicativo do cliente, na qual o acesso é realizado por meio dos campos de *e-mail* e senha. Também é exibido a mensagem de *feedback* de erro ao usuário. Este formato de mensagem é seguido como padrão para os demais formulários.

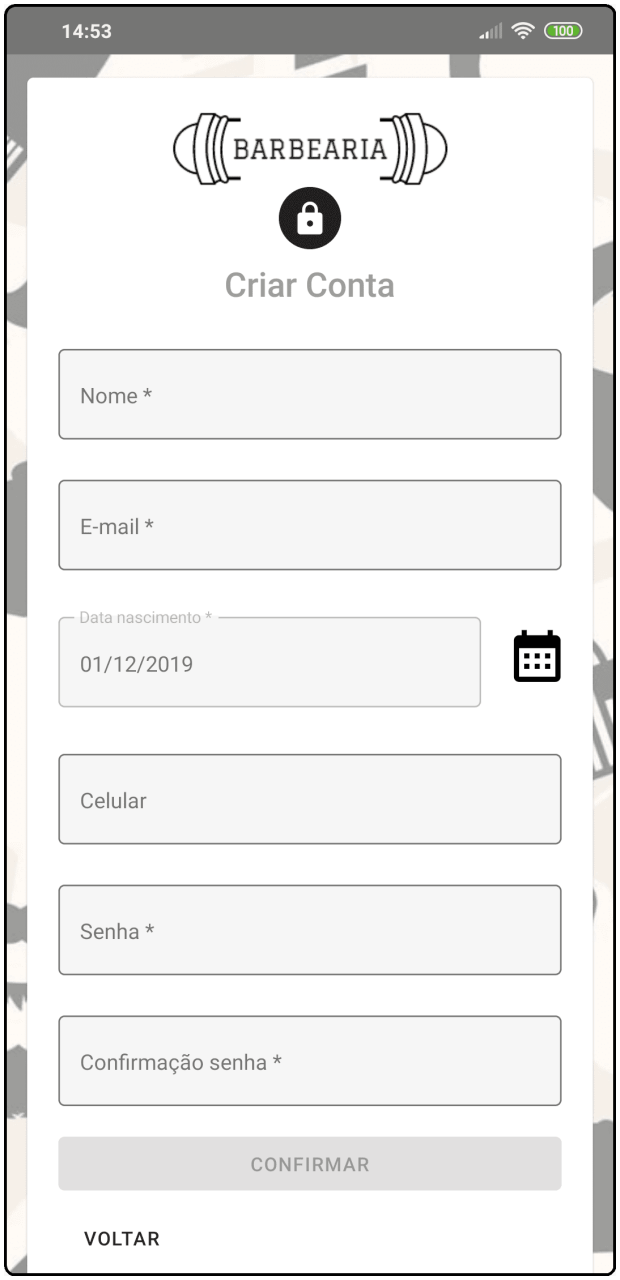
Figura 3 – Interface de autenticação do aplicativo



Fonte: Autoria própria (2019).

A Figura 4 exibe a tela de criação de conta, na qual o usuário deve informar seus dados de identificação e senha para acesso ao aplicativo. Os campos obrigatórios são sinalizados pelo caractere *asterisco (*)*.

Figura 4 – Interface de criação de conta do aplicativo



Fonte: Autoria própria (2019).

A Figura 5 exibe a tela de recuperação de senha, na qual o usuário deve informar seu endereço de *e-mail* para receber uma nova senha no *e-mail*.

Figura 5 – Interface de redefinição de senha do aplicativo

Fonte: Autoria própria (2019).

A Figura 6 exibe o *e-mail* de recuperação de senha enviado ao cliente.

Figura 6 – E-mail de recuperação de senha do aplicativo

Fonte: Autoria própria (2019).

A Figura 7 exibe a tela de agendamento do aplicativo quando não possui nenhum agendamento. A imagem de *feedback* é exibida como padrão para as demais telas do aplicativo quando não possuem nenhum registro a ser exibido.

Figura 7 – Interface de agendamento do aplicativo sem registros

Fonte: Autoria própria (2019).

A Figura 8 exibe a tela de agendamento do aplicativo quando possui agendamentos. Nesta tela é exibido o botão incluir (representando pelo ícone com sinal de +) que ao ser clicado direciona o cliente para o fluxo de cadastro de novo agendamento.

Figura 8 – Interface de agendamento do aplicativo com registros

Fonte: Autoria própria (2019).

As figuras 9, 10, 11 e 12 exibem o processo de inclusão de um agendamento do aplicativo. Este processo foi realizado utilizando *steps* que servem para guiar o cliente.

Na Figura 9 são exibidos os serviços disponíveis para realizar o agendamento.

Figura 9 – Passo 1 da interface de cadastro de agendamento do aplicativo

Fonte: Autoria própria (2019).

Na Figura 10 são exibidos os profissionais disponíveis para realizar o agendamento.

Figura 10 – Passo 2 da interface de cadastro de agendamento do aplicativo

Fonte: Autoria própria (2019).

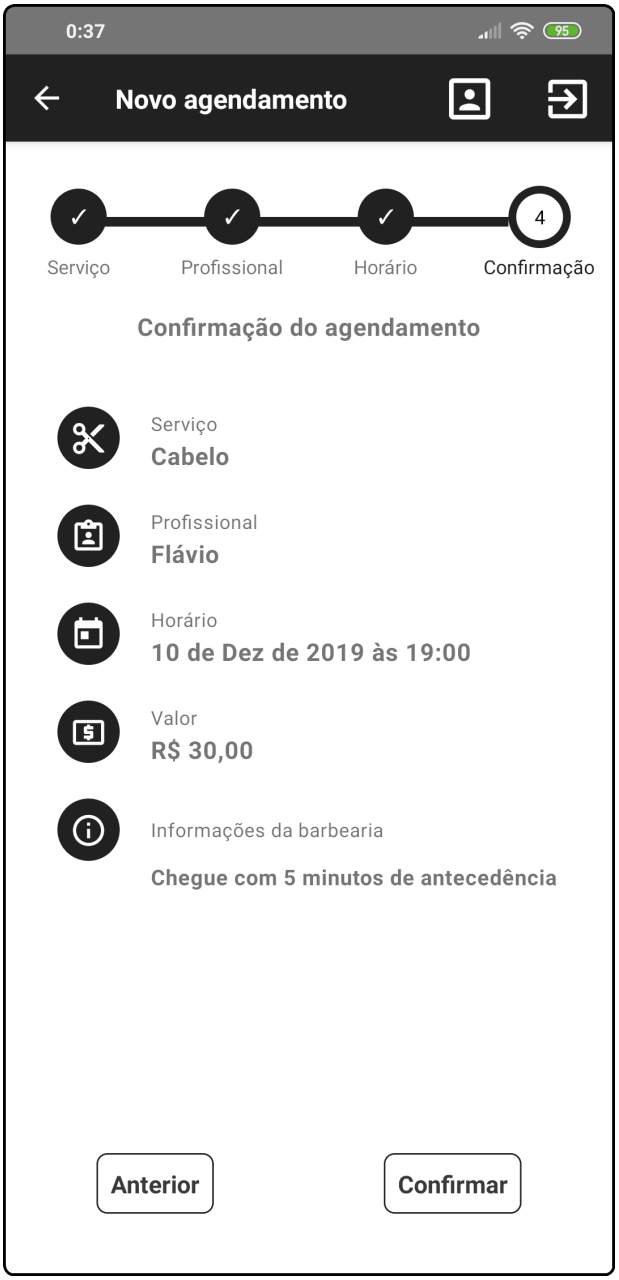
Na Figura 11 são exibidos os horários disponíveis para realizar o agendamento.

Figura 11 – Passo 3 da interface de cadastro de agendamento do aplicativo

Fonte: Autoria própria (2019).

Na Figura 12 é exibido os detalhes selecionados nos passos anteriores e solicitado a confirmação do agendamento.

Figura 12 – Passo 4 da interface de cadastro de agendamento do aplicativo



Fonte: Autoria própria (2019).

A Figura 13 apresenta a tela de perfil do cliente, que é exibida ao clicar no ícone de perfil da barra de navegação superior.

Figura 13 – Interface de perfil do cliente no aplicativo

Fonte: Autoria própria (2019).

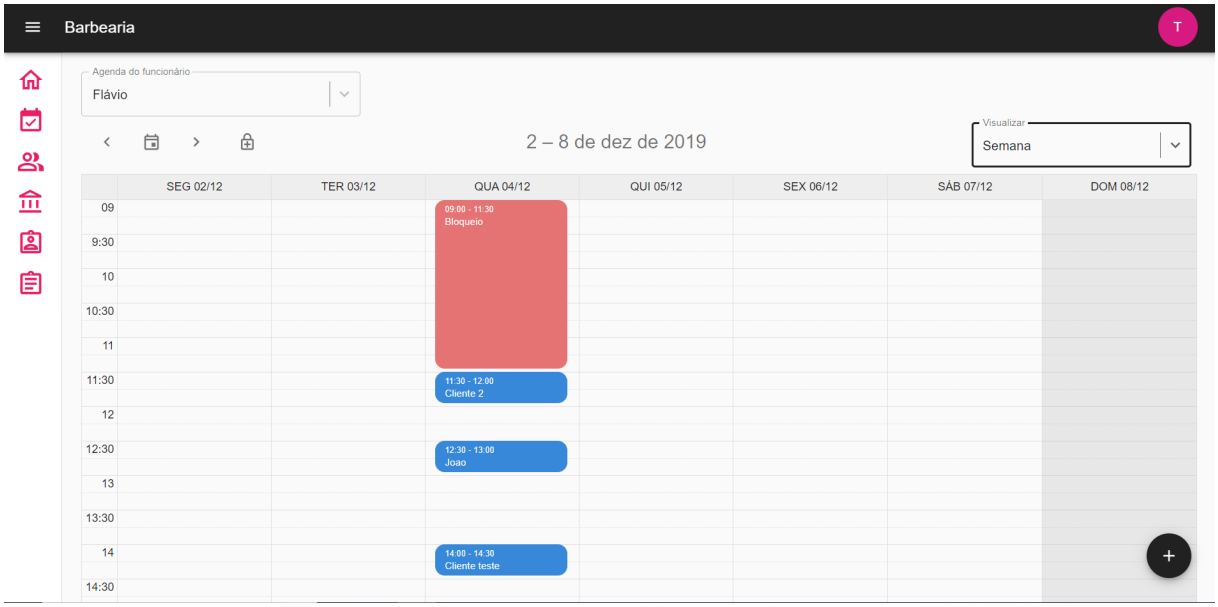
A Figura 14 exibe a tela de autenticação do sistema administrativo.

Figura 14 – Interface de autenticação no sistema administrativo

Fonte: Autoria própria (2019).

A Figura 15 exibe a tela de agenda no formato de semana do sistema administrativo. Nesta tela, caso o usuário autenticado possua perfil de administrador, será exibido o campo de funcionário para que ele possa visualizar a agenda de todos os funcionários, caso contrário, este campo não será exibido e ele poderá visualizar apenas a sua própria agenda. Nesta tela há também as opções de navegação da agenda para voltar, avançar e dia atual. O ícone de bloqueio representado pelo símbolo de cadeado com o sinal de + exibido na parte superior, ao ser clicado o usuário será direcionado para a tela de cadastro de bloqueio, que é exibido na cor vermelha na agenda. O botão, na parte inferior com o sinal de +, direciona o usuário para a tela de cadastro de agendamentos, que são exibidos na agenda com a cor azul.

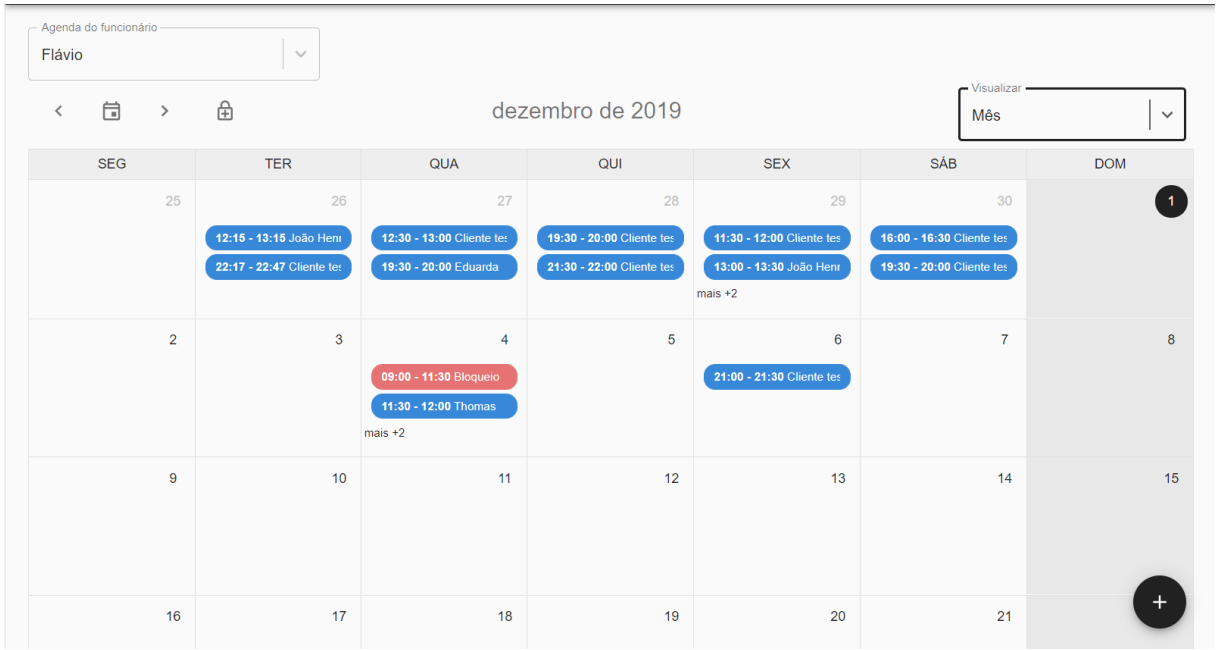
Figura 15 – Interface de agenda semanal sistema administrativo



Fonte: Autoria própria (2019).

A Figura 16 exibe a tela de agenda com a visualização no formato mês. São disponibilizados três formatos de visualização que são: dia, semana e mês que podem ser alternados utilizando o campo "Visualizar"disponibilizado na parte superior. Por padrão é exibido a agenda na visualização por dia.

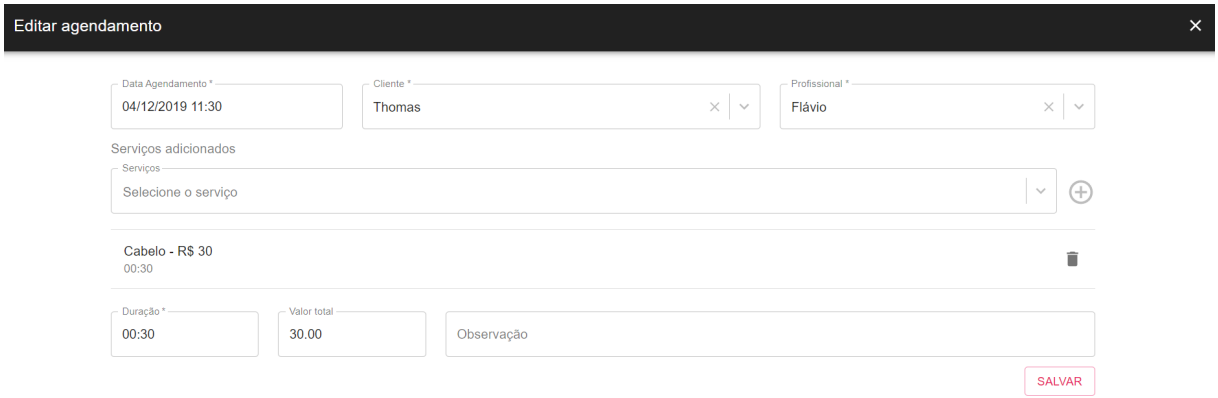
Figura 16 – Interface de agenda mensal do sistema administrativo



Fonte: Autoria própria (2019).

A Figura 17 exibe a tela de agendamento do sistema administrativo. Neste formulário é preciso informar a data e hora de início do agendamento, o cliente, profissional e um ou mais serviços que serão realizados, além da duração e valor total. Caso o usuário autenticado não possua o perfil de administrador o campo de funcionário ficará bloqueado.

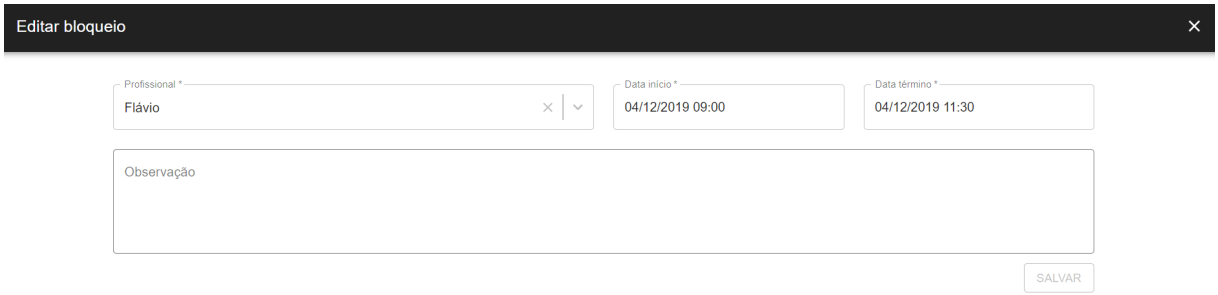
Figura 17 – Interface de cadastro de agendamento do sistema administrativo



Fonte: Autoria própria (2019).

A Figura 18 exibe a tela de bloqueio da agenda no sistema administrativo. Neste formulário é preciso informar a data e hora de início e término do bloqueio além do funcionário que será bloqueada a agenda. Caso o usuário autenticado não possua o perfil de administrador o campo de funcionário ficará bloqueado.

Figura 18 – Interface de bloqueio da agenda do sistema administrativo

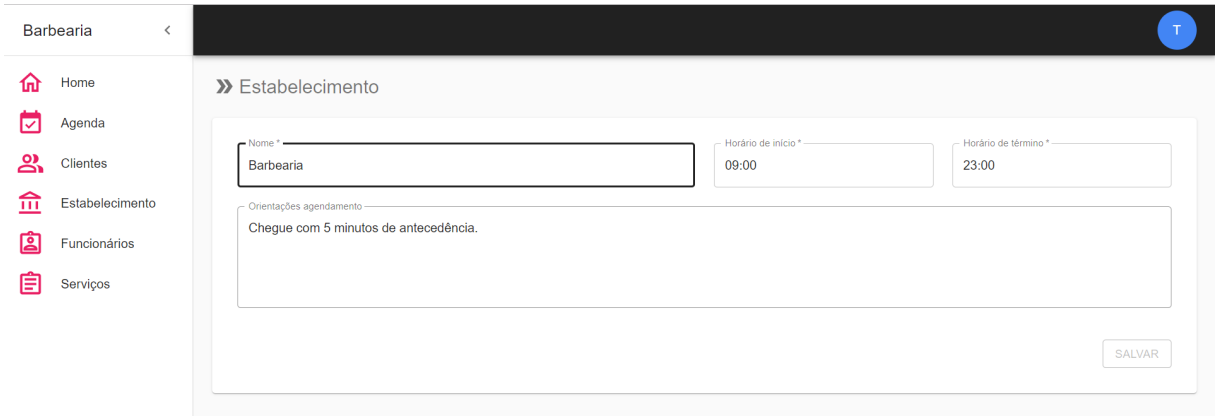


Fonte: Autoria própria (2019).

A Figura 19 exibe a tela de cadastro dos dados do estabelecimento contendo os campos de nome, horário de início e término de funcionamento e orientações a serem exibidas no agendamento *on-line* pelo aplicativo. O horário de funcionamento

é utilizado para definir o início e término da agenda. O nome do estabelecimento é exibido no menu de navegação. Este cadastro é exibido apenas para usuários com perfil de administrador.

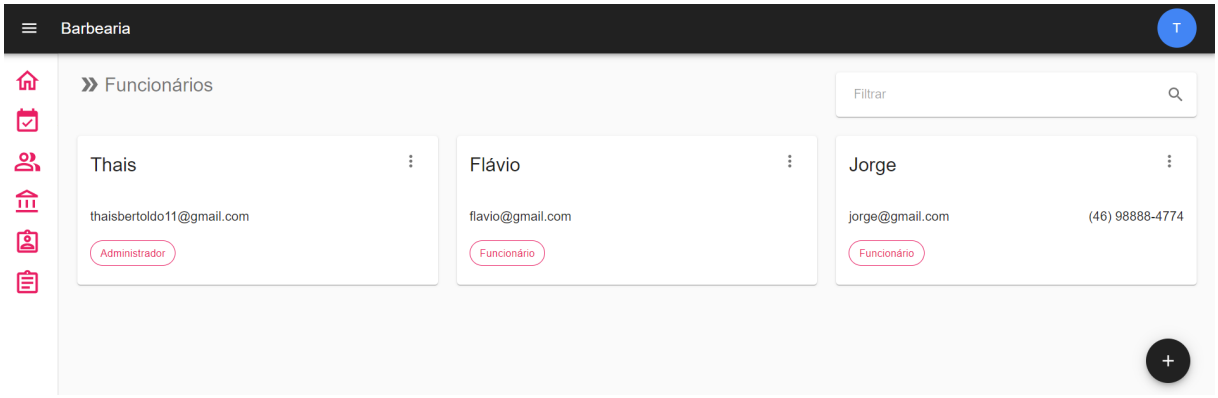
Figura 19 – Interface de dados do estabelecimento do sistema administrativo



Fonte: Autoria própria (2019).

A Figura 20 exibe a tela de listagem de funcionários do sistema administrativo. Este padrão de tela com os *cards*, componente de filtro e botão de inclusão, é utilizado também, para as telas de clientes e serviços. Este cadastro, assim como o de serviços é exibido apenas para usuários com perfil de administrador.

Figura 20 – Interface de listagem de funcionários do sistema administrativo

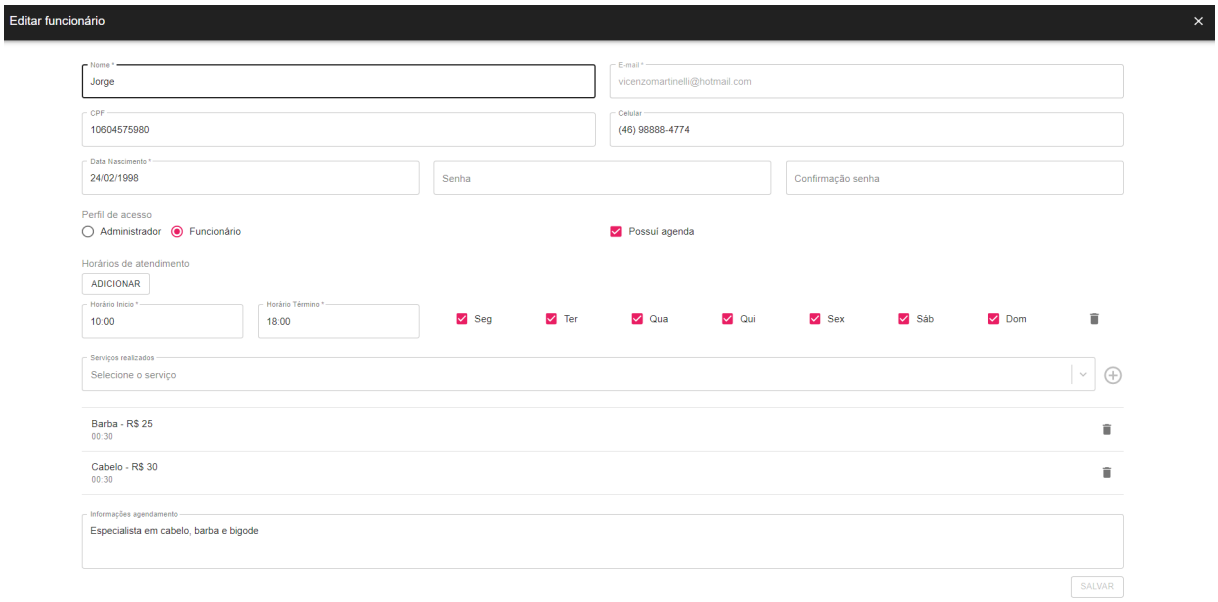


Fonte: Autoria própria (2019).

A Figura 21 exibe a tela de cadastro de funcionários do sistema administrativo. Neste formulário é possível informar os dados básicos de identificação e acesso ao sistema, definindo o perfil de acesso que pode ser administrativo ou funcionário, além

de poder definir se o usuário possui agenda. Caso o funcionário possua agenda são exibidos campos para definir o horário de trabalho e serviços realizados.

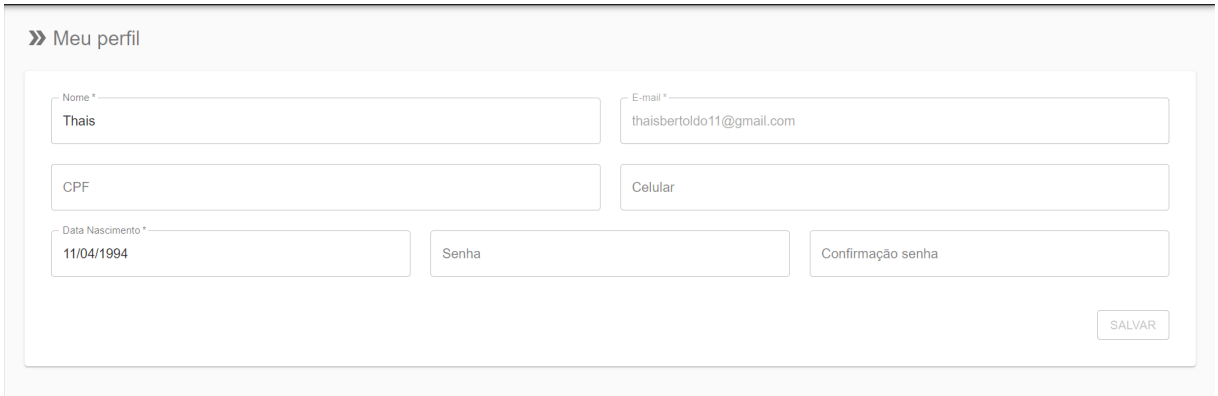
Figura 21 – Interface de cadastro de funcionários do sistema administrativo



Fonte: Autoria própria (2019).

A Figura 22 exibe a tela de edição do perfil do funcionário no sistema administrativo. Nesta tela ele pode atualizar suas informações pessoais e alterar sua senha.

Figura 22 – Interface de perfil do usuário do sistema administrativo



Fonte: Autoria própria (2019).

3.4 IMPLEMENTAÇÃO DO SISTEMA

Para a codificação do sistema, foi realizada a instalação e configuração do ambiente de desenvolvimento juntamente com os recursos utilizados (*frameworks*, *plugins*, etc.)

Para agilizar o desenvolvimento do *back-end* o projeto inicial foi criado utilizando a ferramenta disponibilizada no *site: https://start.spring.io* adicionando as dependências necessárias, que ficam disponíveis no arquivo *pom.xml*. Após isso foram adicionadas as configurações de conexão do banco de dados PostgreSQL no arquivo *application.properties*, conforme exibido na Listagem 1.

Listagem 1 – Configuração de conexão com banco de dados

```
1 spring.datasource.url=jdbc:postgresql://localhost:5432/tcc
2 spring.datasource.username=******
3 spring.datasource.password=******
4 spring.jpa.show-sql=true
5 spring.jpa.hibernate.ddl-auto=update
6 spring.datasource.driver-class-name=org.postgresql.Driver
7 spring.jpa.database-platform=org.hibernate.dialect.PostgreSQL95Dialect
```

Fonte: Autoria própria (2019).

Como foi utilizado o Spring Data ao estender o *CrudRepository* automaticamente é herdado vários métodos como, por exemplo, salvar, atualizar e deletar, para utilizá-los é preciso criar uma interface que estenda de CrudRepository, conforme Listagem 2.

Listagem 2 – CrudRepository

```
1 @Repository
2 public interface HorarioProfissionalRepository extends
3         CrudRepository<HorarioProfissional, Long> {
4
5   public void deleteByProfissional(Usuario profissional);
6
7   public List<HorarioProfissional> findByProfissional
8           (Usuario profissional);
```

```
9 }
```

Fonte: Autoria própria (2019).

Para garantir mais segurança foi aplicando o padrão *DTO* que, basicamente, é uma classe com atributos simples, utilizada para otimizar a comunicação entre o cliente e o servidor. Para facilitar a aplicação deste padrão, nas classes de *DTO* foi criado um construtor que recebe a classe de entidade e popula a classe *DTO* conforme exibido na Listagem 3.

Listagem 3 – Padrão DTO

```
1  public class ProfissionalDTO {
2    private long idProfissional;
3    private String nomeProfissional;
4    private String descricaoProfissional;
5
6    public ProfissionalDTO() {
7
8    }
9
10   public ProfissionalDTO(Usuario usuario) {
11     this.idProfissional = usuario.getId();
12     this.nomeProfissional = usuario.getNome();
13     this.descricaoProfissional = usuario.getObservacao() != null ?
14     usuario.getObservacao() : "";
15   }
16   ...
17 }
```

Fonte: Autoria própria (2019).

Os *constrollers* são anotados com *@RestController* que permite criar um controlador com características REST e que possa manipular as requisições vindas dos clientes. A anotação *@RequestMapping* permite definir a *Uniform Resource Locator (URL)* padrão para os métodos disponíveis dentro do *controller*. Como o Spring tem toda a automatização da aplicação do conceito de injeção de dependências basta anotar

com *@Autowired*, mas para que a instância possa ser injetada é preciso que ela seja um *bean Spring*. Para isso ela pode ser anotada com @Service.

O método *SecurityContextHolder.getContext().getAuthentication().getPrincipal()* é responsável por retornar o *UserDetails* do usuário conectado. Com ele é possível obter o *username*, e após isso, utilizando o *service* injetado é retornado o usuário conectado. Com este usuário utilizando novamente o *service* é buscado os agendamentos cadastrados, o retorno é iterado utilizando o *map* para converter o *agendamento* em *agendamentoDTO*, conforme exibido na Listagem 4

Listagem 4 – Controller

```
1  @RestController
2  @RequestMapping("/agendamento")
3  public class AgendamentoController {
4
5    @Autowired
6    private AgendamentoService agendaService;
7
8    @Autowired
9    private UsuarioService usuarioService;
10
11   @Autowired
12   private SimpMessagingTemplate webSocket;
13
14   @GetMapping
15   @PreAuthorize("hasRole('CLIENTE')")
16   @RequestMapping(value = "/cliente")
17   public List<AgendamentoDTO> listAllCliente() {
18     Object principal = SecurityContextHolder.getContext()
19         .getAuthentication().getPrincipal();
20
21     if (principal instanceof UserDetails) {
22       String email = ((UserDetails) principal).getUsername();
23       Usuario cliente = usuarioService.findOne(email);
24       return agendaService.findByCliente(cliente).stream().map(a ->
```

```
25                  new AgendamentoDTO(a)).collect(Collectors.toList());
26      } else {
27        return null;
28      }
29    }
30    ...
31  }
```

Fonte: Autoria própria (2019).

A Listagem 5 exibe a configuração do *WebSocket*. Nele é definido o registro dos clientes e o envio das mensagens.

Listagem 5 – WebSocket Spring

```
1  @Configuration
2  @EnableWebSocketMessageBroker
3  @EnableWebSocket
4  public class WebSocketConfiguration implements
5          WebSocketMessageBrokerConfigurer {
6    @Override
7    public void configureMessageBroker(MessageBrokerRegistry config) {
8      config.enableSimpleBroker("/agenda");
9      config.setApplicationDestinationPrefixes("/app");
10   }
11   @Override
12   public void registerStompEndpoints(StompEndpointRegistry registry) {
13     registry.addEndpoint("/agenda-websocket").setAllowedOrigins("*")
14         .withSockJS();
15   }
16 }
```

Fonte: Autoria própria (2019).

O método exibido na Listagem 6 é responsável por retornar os horários disponíveis do funcionário em um determinado dia. Para isso, primeiro é obtido o dia da semana da data desejada, após isso, é obtido os horários de trabalho do profissional

neste dia. É criado um *array* com os horários disponíveis baseado no horário do profissional e duração do serviço desejado. Após ser obtido os agendamentos criados para este dia, eles são percorridos e removidos dos horários disponíveis. Após isso, a lista de horários disponíveis é retornada ao cliente.

Listagem 6 – Horários disponíveis

```
1  @Override
2  public List<HorarioDisponivelDTO> findHorarioDisponivel(int dia,
3          int mes, int ano, String duracao, Usuario profissional) {
4
5    LocalDateTime dataInicio = LocalDateTime
6            .of(ano, mes, dia, 0, 0, 0);
7
8    LocalDateTime dataTermino = LocalDateTime
9            .of(ano, mes, dia, 23, 59, 59);
10
11   int diaSemana = dataInicio.getDayOfWeek().getValue();
12
13   Predicate<HorarioProfissional> byDay = h -> ((h.isSegunda() &&
14         diaSemana == 1) || (h.isTerca() && diaSemana == 2) ||
15         (h.isQuarta() && diaSemana == 3) || (h.isQuinta() &&
16         diaSemana == 4) || (h.isSexta() && diaSemana == 5) ||
17         (h.isSabado() && diaSemana == 6) || (h.isDomingo() &&
18         diaSemana == 7));
19
20   List<HorarioProfissional> horariosDia = profissional.
21       getHorariosProfissional().stream().filter(byDay)
22       .collect(Collectors.toList());
23
24   List<HorarioDisponivelDTO> horariosDisponiveis =
25       new ArrayList<HorarioDisponivelDTO>();
26
27   String[] duracaoSplit = duracao.split(":");
28   long hora = Long.parseLong(duracaoSplit[0]);
```

```
29    long minuto = Long.parseLong(duracaoSplit[1]);
30
31    for (HorarioProfissional h : horariosDia) {
32      String[] inicio = h.getHorarioInicio().split(":");
33      String[] termino = h.getHorarioTermino().split(":");
34
35      LocalDateTime dataInicioProfissional = LocalDateTime.of(
36          ano, mes, dia, Integer.parseInt(inicio[0]),
37          Integer.parseInt(inicio[1])
38        );
39
40      LocalDateTime dataTerminoProfissional = LocalDateTime.of(
41          ano, mes, dia, Integer.parseInt(termino[0]),
42          Integer.parseInt(termino[1])
43        );
44
45      if(dataInicioProfissional.isAfter(LocalDateTime.now())) {
46        horariosDisponiveis.add(
47            new HorarioDisponivelDTO(dataInicioProfissional)
48          );
49      }
50
51      while (dataInicioProfissional.isBefore(dataTerminoProfissional))
52      {
53        dataInicioProfissional = dataInicioProfissional
54        .plusHours(hora);
55
56        dataInicioProfissional = dataInicioProfissional
57            .plusMinutes(minuto);
58
59        if (dataInicioProfissional.isBefore(dataTerminoProfissional)
60            && dataInicioProfissional.isAfter(LocalDateTime.now())) {
61            horariosDisponiveis.add(new
62              HorarioDisponivelDTO(dataInicioProfissional)
```

```
63              );
64          }
65      }
66  }
67
68  horariosDisponiveis.sort(Comparator
69          .comparing(HorarioDisponivelDTO::getHorario));
70
71  List<Agendamento> agendamentos = agendaDao
72          .findByProfissionalAndDataInicioBetween(
73                  profissional, dataInicio, dataTermino
74          );
75
76      List<HorarioDisponivelDTO> newHorariosDisponiveis = new
77      ArrayList<>(horariosDisponiveis);
78
79      if (agendamentos.size() > 0) {
80          for (Agendamento agendamento : agendamentos) {
81              LocalDateTime inicioAgendamento =
82              agendamento.getDataInicio();
83              LocalDateTime terminoAgendamento =
84              agendamento.getDataTermino();
85
86              horariosDisponiveis = new
87              ArrayList<HorarioDisponivelDTO>(newHorariosDisponiveis);
88
89              for (int i = 0; i < horariosDisponiveis.size(); i++) {
90
91                  HorarioDisponivelDTO h = horariosDisponiveis.get(i);
92                  HorarioDisponivelDTO h2;
93                  if (i < horariosDisponiveis.size() - 1) {
94                      h2 = horariosDisponiveis.get(i + 1);
95                  } else {
96                      h2 = horariosDisponiveis.get(i);
```

```
97              }
98
99              if ((((inicioAgendamento.isAfter(h.getHorario()) ||
100             inicioAgendamento.isEqual(h.getHorario()))))
101                 && inicioAgendamento.isBefore(h2.getHorario()))
102                 || (terminoAgendamento.isAfter(h.getHorario())
103                 && !terminoAgendamento.equals(h.getHorario())
104                     && h2.getHorario().isAfter(inicioAgendamento))
105                 || (h.getHorario().equals(inicioAgendamento))) {
106
107               if (newHorariosDisponiveis.contains(h)) {
108                   newHorariosDisponiveis.remove(
109                       newHorariosDisponiveis.indexOf(h)
110                   );
111               }
112             }
113           }
114         }
115         return newHorariosDisponiveis;
116     } else {
117       return horariosDisponiveis;
118     }
119 }
```

Fonte: Autoria própria (2019).

Na Listagem 7 é exibido o arquivo *app.js* da aplicação em React, nele é definido o componente que deve ser renderizado. O método *useAuth* retorna o *eu* que é um objeto com os dados do usuário autenticado, caso ele exista significa que o usuário está autenticado e é carregado o componente *Admin*, caso contrário é carregado o componente *Auth*. Enquanto ele está sendo renderizado é exibido o componente de *progress* que é definido na função de *fallback*. O componente é carregado com *lazy*, com isso, as páginas só serão renderizadas quando forem acessadas, tornando a aplicação mais rápida. O *useMemo* retorna um valor *memoizado*, que é uma técnica de programação que procura aumentar a performance de uma função, cacheando seus

valores previamente computados, esta otimização ajuda a evitar cálculos caros em cada renderização.

Listagem 7 – App React

```
1  import React, { lazy, Suspense, useMemo } from 'react'
2
3  import CircularProgress from '@material-ui/core/CircularProgress'
4
5  import { useAuth } from './providers/auth'
6
7  import './App.css'
8  const Auth = lazy(() => import('./pages/auth'))
9  const Admin = lazy(() => import('./pages/admin'))
10
11 const App = () => {
12   const { eu } = useAuth()
13
14   const Component = useMemo(() => {
15     if (!eu) {
16       return Auth
17     }
18
19     return Admin
20   }, [eu])
21
22   return (
23     <Suspense fallback={<CircularProgress />}>
24       <Component />
25     </Suspense>
26   )
27 }
28
29 export default App
```

Fonte: Autoria própria (2019).

A Listagem 8 exibe o *provider* de autenticação da aplicação React. Nele é criado o *AuthContext* usando o *hook createContext da Context API*. A *Context API*, fornece uma maneira de passar os dados de componentes sem ter que passar manualmente em todos os níveis. Este contexto irá armazenar os dados do usuário (eu) e as funções de *login* e *logout*, e a partir do contexto, os componentes interessados conseguirão consumi-los de forma simples.

Listagem 8 – Auth Provider

```
1  import React, { useContext, createContext } from 'react'
2
3  import { useAsync } from 'react-async'
4
5  import CircularProgress from '@material-ui/core/CircularProgress'
6
7  const AuthContext = createContext(null)
8
9  export const useAuth = () => {
10   const context = useContext(AuthContext)
11
12   if (context === undefined) {
13     throw new Error('[useAuth]: Deve ser usado dentro
14         de um AuthProvider')
15   }
16
17   return context
18 }
19
20 export const AuthProvider = ({ bootstrap, onLogin, onLogout,
21     children }) => {
22
23   const { data, isPending, setData } = useAsync(bootstrap)
24
25   if (isPending) {
26     return <CircularProgress />
```

```
27    }
28
29    const login = data => onLogin(data).then(setData)
30    const logout = () => onLogout().then(setData)
31
32    const value = {
33      eu: data,
34      login,
35      logout
36    }
37
38    return(
39          <AuthContext.Provider value={value}>
40               {children}
41          </AuthContext.Provider>
42      )
43  }
```

Fonte: Autoria própria (2019).

Na Listagem 9 é exibido o componente *Providers* que é responsável por adicionar os *provider* do *Snackbar*, que é o componente utilizado para exibição de mensagens, o *AuthProvider* que foi exibido na Listagem 8, além do *BrowserRouter* que é responsável pelo controle das rotas da aplicação. Neste *provider* também é adicionado um *interceptor* responsável por obter o *token* do usuário autenticado e adicioná-lo no *header* da requisição. O *interceptor de response* é responsável por rejeitar a *promise* quando ocorrer erro.

Listagem 9 – Providers

```
1  import React, { useLayoutEffect } from 'react'
2
3  import { BrowserRouter } from 'react-router-dom'
4
5  import http from '../api/http'
6
7  import * as authService from '../services/auth'
8
9  import { AuthProvider } from './auth'
10 import { SnackbarProvider } from 'react-snackbar-alert'
11
12 http.interceptors.request.use(function(config) {
13   const token = authService.getToken()
14
15   if (token) {
16     config.headers['Authorization'] = `Bearer ${token}`
17   }
18
19   return config
20 })
21
22 http.interceptors.response.use(
23   function(response) {
24     return response
25   },
26
27   function(error) {
28     console.log('http error: ', error)
29     return Promise.reject(error)
30   }
31 )
32
33 const Providers = ({ children }) => {
```

```
34    useLayoutEffect(() => {
35      const params = new URLSearchParams(window.location.search)
36
37      const token = params.get('token')
38
39      if (token) {
40        authService.setToken(token)
41      }
42
43      window.history.replaceState({}, document.title,
44      `${window.location.origin}${window.location.pathname}`)
45    }, [])
46
47    return (
48      <SnackbarProvider position="bottom">
49        <AuthProvider bootstrap={authService.bootstrap}
50        onLogin={authService.login} onLogout={authService.logout}>
51          <BrowserRouter>{children}</BrowserRouter>
52        </AuthProvider>
53      </SnackbarProvider>
54    )
55  }
56
57  export default Providers
```

Fonte: Autoria própria (2019).

Na Listagem 10 é exibido o *service* de autenticação, nele são exportadas funções para *login* e *logout*, gravar, obter e remover o *token* do *Local Storage*, entre outros.

Listagem 10 – AuthService

```
1  import * as usuariosService from '../api/usuarios'
2  import * as loginService from '../api/login'
3
4  export function getToken() {
5    return window.localStorage.getItem(
```

```
6       process.env.REACT_APP_AUTH_TOKEN_KEY
7     )
8   }
9
10  export function setToken(token) {
11    return window.localStorage.setItem(
12      process.env.REACT_APP_AUTH_TOKEN_KEY, token
13    )
14  }
15
16  export function removeToken() {
17    return window.localStorage.removeItem(
18      process.env.REACT_APP_AUTH_TOKEN_KEY
19    )
20  }
21
22  export function bootstrap() {
23    const token = getToken()
24
25    if (!token) {
26      return Promise.resolve(null)
27    }
28
29    return usuariosService.eu().catch(() => Promise.resolve(null))
30  }
31
32  export async function login({ email, senha }) {
33    return loginService.login({ email, senha }).then(data => {
34      setToken(data.token)
35
36      return data.eu
37    })
38  }
39
```

```
40 export function logout() {
41   removeToken()
42
43   return Promise.resolve(null)
44 }
```

Fonte: Autoria própria (2019).

A Listagem 11 exibe o código do *index.js* do componente de autenticação, nele é definido o *provider* do tema que neste caso está sendo utilizado o *material-ui*, além do *provider* de autenticação e as rotas disponíveis neste componente. O controle de rotas é realizado utilizando o *React Router Dom*.

Listagem 11 – Index autenticação

```
1  import React, { lazy, Suspense } from 'react'
2
3  import { Redirect, Route, Switch } from 'react-router-dom'
4
5  import { MuiThemeProvider } from '@material-ui/core'
6  import { AuthLayout } from '../../components/AuthLayout'
7
8  import CircularProgress from '@material-ui/core/CircularProgress'
9
10 import theme from '../../theme'
11
12 const Login = lazy(() => import('./Login'))
13 const RecuperarSenha = lazy(() => import('./RecuperarSenha'))
14
15 const Auth = () => {
16   return (
17     <MuiThemeProvider theme={theme}>
18       <AuthLayout>
19         <Suspense fallback={<CircularProgress />}>
20           <Switch>
21             <Route path="/auth/login" exact component={Login} />
22             <Route path="/auth/recuperar-senha" exact
```

```
23                     component={RecuperarSenha} />
24                 <Route render={() => <Redirect to="/auth/login" />} />
25               </Switch>
26             </Suspense>
27           </AuthLayout>
28       </MuiThemeProvider>
29     )
30 }
31
32 export default Auth
```

Fonte: Autoria própria (2019).

A Listagem 12 exibe a função de conexão com o *WebSocket* no cliente. Caso não esteja conectado é realizada a conexão utilizando o *stompCliente* que recebe a função *onSubscribe* que é acionada quando houver algum retorno da *API*, neste caso, quando for incluso, editado ou excluído algum registro de agendamentos.

Listagem 12 – Socket cliente

```
1  import * as Stomp from 'stomp-websocket'
2  import * as SockJS from 'sockjs-client'
3
4  export const connectSocket = (onSubscribe, setSocketConnected,
5      socketConnected) => {
6    if (!socketConnected) {
7      const socket = new SockJS(`${process.env.REACT_APP_API_URL}/
8          agenda-websocket`)
9      const stompClient = Stomp.over(socket)
10
11     stompClient.connect(
12       {},
13       () => {
14         setSocketConnected(true)
15         stompClient.subscribe('/agenda', retorno => {
16           return onSubscribe(retorno.body)
17         })
```

```
18          },
19          e => {
20            console.log('error', e)
21          }
22        )
23        return () => {
24          if (stompClient !== null) {
25            stompClient.disconnect()
26            setSocketConnected(false)
27          }
28        }
29      }
30    }
```

Fonte: Autoria própria (2019).

A Listagem 13 exibe parte do código do componente da agenda. O primeiro *useEffect* é responsável pela conexão do cliente com o *WebSocket*, onde ele realiza a chamada da função de conexão exibida na Listagem 12 e quando recebe algum retorno da *API* grava o valor na constante *resSocket*. Este *useEffect* é chamado quando ocorre alteração nos valores de *setSocketConnected* e *socketConnected*. Já o segundo *useEffect* é executado quando ocorrer mudança nos valores de *onFetch*, *resSocket* ou *selectedFuncionario*. Por este motivo, quando houver alguma alteração nos agendamentos ele é acionado e com isso são buscados os agendamentos atualizados na *API*, fazendo com que a agenda seja atualizada automaticamente quando houver alterações.

Listagem 13 – onSubscribe socket agenda

```
1  ...
2  const [resSocket, setResSocket] = useState()
3  ...
4  useEffect(() => {
5      socket.connectSocket(
6        res => {
7          setResSocket(res)
```

```
8           setOnFetch(true)
9         },
10        setSocketConnected,
11        socketConnected
12      )
13    }, [setSocketConnected, socketConnected])
14  ...
15   useEffect(() => {
16      if (selectedFuncionario && onFetch) {
17        const fetchEventos = async () => {
18          setIsLoading(true)
19          const data = await agendasApi.obterAgendamentosFuncionario({
20              id: selectedFuncionario.value }
21          )
22
23          const dataHorarios = await usuariosApi.obterHorariosUsuario({
24              id: selectedFuncionario.value }
25          )
26
27          setHorarioComercial(dataHorarios)
28          setEventos(data)
29          setIsLoading(false)
30          setResSocket()
31        }
32        fetchEventos()
33        setOnFetch(false)
34      }
35    }, [onFetch, resSocket, selectedFuncionario])
```

Fonte: Autoria própria (2019).

A Listagem 14 exibe o componente utilizado para renderização da agenda. Este componente é altamente customizável e possui a grande maioria dos recursos exigidos para uma agenda. Ele tem opção de exibição por dia, semana, mês e agenda, além de possibilidade de definir a localidade permitindo assim a internacionalização.

Listagem 14 – Componente de agenda

```
<FullCalendar
      ref={calendarRef}
      allDaySlot={false}
      overlap={false}
      selectable={true}
      eventLimit={true}
      nowIndicator={true}
      editable={false}
      defaultDate={moment().toDate()}
      slotDuration="00:15"
      height="auto"
      displayEventEnd={true}
      locale={ptBrLocale}
      firstDay={1}
      plugins={[dayGridPlugin, timeGridPlugin,
           interactionPlugin, listPlugin]}
      {...props}
   />
</FullCalendarStyled>
```

Fonte: Autoria própria (2019).

A Listagem 15 exibe a função de validação de formulários utilizando o *Yup*. Os métodos de validação do *Yup* seguem o padrão onde o primeiro parâmetro é o valor e o segundo é a mensagem de erro. Ele disponibiliza diversos métodos de validações como por exemplo *required* que verifica se o campo está preenchido, caso não esteja retorna a mensagem de erro.

Listagem 15 – Validação de formulários

```
const validationSchema = useMemo(() => {
    return Yup.object({
      start: Yup.string().required('Horário é obrigatório'),
      cliente: Yup.object()
        .required('Cliente é obrigatório')
        .nullable(),
      profissional: Yup.object()
        .required('Profissional é obrigatório')
        .nullable(),
      duracao: Yup.string().required('Duração é obrigatório'),
      servicosAgendamento: Yup.array().required('Adicione ao menos
      um serviço')
    })
  }, [])
```

Fonte: Autoria própria (2019).

A Listagem 16 exibe um formulário criado utilizando o *Formik* que é uma biblioteca que facilita o manuseio de formulários em React. Ele espera uma função que deve ser chamada ao submeter o formulário (*onSubmit*), os valores iniciais do formulário (*initialValues*), *schema* de validação (*validationSchema*), além de vários outros. O mapeamento dos dados iniciais com os campos do formulário e feito utilizando o *name* dos *fields*.

Listagem 16 – Criação de formulários

```
<Formik onSubmit={onSubmit} initialValues={initialValues}
    validationSchema={validationSchema}>
  {({ isSubmitting, isValid }) => (
    <Form noValidate>
      <div className={classes.content}>
        <Grid container spacing={3}>
          <Grid item xs={matches ? 6 : 12}>
            <Field
              name="nomeServico"
```

```
10                autoFocus
11                variant="outlined"
12                fullWidth
13                label="Nome"
14                required
15                component={TextField}
16              />
17            </Grid>
18            <Grid item xs={matches ? 3 : 6}>
19              <Field
20                name="valorServico"
21                variant="outlined"
22                fullWidth
23                label="Valor"
24                required
25                component={TextField}
26              />
27            </Grid>
28            <Grid item xs={matches ? 3 : 6}>
29              <Field
30                name="duracaoServico"
31                variant="outlined"
32                fullWidth
33                label="Duração"
34                required
35                component={TextField}
36              />
37            </Grid>
38            <Grid item xs={12}>
39              <Field
40                name="descricaoServico"
41                variant="outlined"
42                fullWidth
43                multiline
```

```
44                rows="5"
45                label="Descrição"
46                component={TextField}
47              />
48            </Grid>
49          </Grid>
50          <Grid container alignItems="flex-start" justify="flex-end"
51          direction="row" spacing={3}>
52            <Grid item>
53              <Button variant="outlined" type="submit"
54                  color="secondary" disabled={isSubmitting ||
55                  !isValid}>
56                Salvar
57              </Button>
58            </Grid>
59
60            {isSubmitting && (
61              <Grid item>
62                <CircularProgress size="2rem" />
63              </Grid>
64            )}
65          </Grid>
66        </div>
67      </Form>
68    )}
69  </Formik>
```

Fonte: Autoria própria (2019).

A Listagem 17 exibe o código do *App.js* do React Native, nele é configurado o *provider* do *Paper* que é uma coleção de componentes do *Material Design* e o *provider* de tema, além do componente de rotas.

Listagem 17 – App React Native

```
1  import React from 'react';
2
3  import {Provider as PaperProvider, ThemeProvider}
4      from 'react-native-paper';
5
6  import {theme} from './theme/theme';
7
8  import Routes from './routes/Routes';
9
10 const App = () => {
11   return (
12     <PaperProvider theme={theme}>
13       <ThemeProvider theme={theme}>
14         <Routes />
15       </ThemeProvider>
16     </PaperProvider>
17   );
18 };
19
20 export default App;
```

Fonte: Autoria própria (2019).

A Listagem 18 exibe as configurações de rotas do aplicativo, para a configuração foi utilizado o *React Navigation*, ele fornece uma maneira de fazer a transição entre as telas e gerenciar o histórico de navegação do aplicativo. Com o *createStackNavigator* é possível realizar a navegação em pilhas, melhorando a usabilidade do usuário pois ele pode voltar utilizando o botão de voltar do aplicativo. Foram criadas duas rotas principais, uma com as rotas de autenticação e outra com as rotas do usuário autenticado, pois assim, é possível controlar as *stacks*, fazendo que quando o usuário estiver autenticado, ao clicar em voltar o *app* seja fechado ao invés de abrir a tela de autenticação.

Listagem 18 – Routes React Native

```
const Routes = createSwitchNavigator({
  AuthStack: createStackNavigator(
    {
      Login: {
        screen: Login,
        navigationOptions: {
          headerShown: false,
        },
      },
      RecuperarSenha: {
        screen: RecuperarSenha,
        navigationOptions: {
          headerShown: false,
        },
      },
      CadastroConta: {
        screen: CadastroConta,
        navigationOptions: {
          headerShown: false,
        },
      },
    },
    {
      initialRouteName: 'Login',
    },
  ),
  app: createStackNavigator(
    {
      AgendamentoList: {
        screen: AgendamentoList,
        navigationOptions: {
          headerTitle: 'Agendamentos',
        },
```

```
34        },
35        AgendamentoCreate: {
36          screen: AgendamentoCreate,
37          navigationOptions: {
38            headerTitle: 'Novo agendamento',
39          },
40        },
41        Perfil: {
42          screen: Perfil,
43          navigationOptions: {
44            headerTitle: 'Meu perfil',
45          },
46        },
47      },
48      {
49        defaultNavigationOptions: ({navigation}) => ({
50          headerStyle: {
51            backgroundColor: theme.colors.primary,
52          },
53          headerTintColor: '#fff',
54          headerTitleStyle: {
55            fontSize: 17,
56            fontWeight: 'bold',
57          },
58          headerRight: () => (
59            <>
60              <IconButton
61                color="#fff"
62                icon="account-box-outline"
63                size={35}
64                onPress={() => {
65                  navigation.navigate('Perfil');
66                }}
67              />
```

```
68              <IconButton
69                color="#fff"
70                icon="exit-to-app"
71                size={33}
72                onPress={() => {
73                  const logg = logout();
74                  if (logg) {
75                    navigation.navigate('Login');
76                  }
77                }}
78              />
79            </>
80          ),
81        }),
82      },
83      {
84        initialRouteParams: 'Login',
85      },
86    ),
87  });
```

Fonte: Autoria própria (2019).

4 CONCLUSÃO

O objetivo deste trabalho constituiu em projetar e implementar um sistema para automatizar o processo de agendamento em barbearias. Para a implementação foram utilizadas várias tecnologias, que possibilitaram a integração entre o *back-end*, *front-end* e o aplicativo móvel, proporcionando um grande avanço no conhecimento de processos e tecnologias de desenvolvimento de software.

Durante o processo de desenvolvimento foram encontradas dificuldades, devido à falta de experiência com algumas tecnologias, dentre elas o uso de *WebSocket* para tornar a agenda *realtime* e o React Native utilizado para o desenvolvimento de aplicativos para dispositivos móveis. Para superar estas dificuldades foram consultados tutoriais, documentação e vídeo-aulas.

Apesar das dificuldades iniciais encontradas, o uso das tecnologias escolhidas foi satisfatório, pois, o uso do *Websocket* foi eficaz para a atualização em tempo real, e o React Native agilizou o desenvolvimento do aplicativo para múltiplas plataformas, se mostrando performático e atendendo à todos os pré-requisitos desejados.

Pelo fato das tecnologias utilizadas serem *open-source* e possuírem uma comunidade ativa, há uma vasta documentação e diversos componentes disponíveis, o que agilizou e facilitou o desenvolvimento responsivo do layout do *front-end* e do aplicativo móvel.

O sistema foi desenvolvido visando proporcionar ao usuário uma forma intuitiva e clara para que os usuários possam interagir da melhor forma possível.

Como trabalho futuro, pretende-se aprimorar o controle de agendamentos do sistema *web*. No aplicativo pretende-se adicionar funcionalidade de cancelar agendamentos e enviar notificações próximo ao horário do agendamento. Também há o desejo de implementação de uma ferramenta de pesquisa de satisfação do atendimento da barbearia e do profissional.

REFERÊNCIAS

BECKER, L. O que é react native? 09 2019. Disponível em: <https://www.organicadigital.com/blog/o-que-e-react-native/>.

CARDOSO, B. E. E. Trabalho de conclusão de curso apresentado ao instituto federal minas gerais. 2018. Disponível em: <https://www.formiga.ifmg.edu.br/documents/2018/Biblioteca/TCCs_e_Artigos/-monografia_Bruno_Enes.Computacao.pdf>.

FERNANDES, D. 3 passos para aprender react native. 2017. Disponível em: <https://blog.rocketseat.com.br/3-passos-para-aprender-react-native/>.

PRATA, R. React: por que considerá-lo no seu projeto? 06 2019. Disponível em: <https://dtidigital.com.br/considere-react-no-seu-projeto/>.

REPÚBLICA, P. da. Lei nº 13.352, de 27 de outubro de 2016. 20 2016. Disponível em: <http://www.planalto.gov.br/ccivil_03/_Ato2015-2018/2016/Lei/L13352.htm>.

SCHITINI. Spring framework. 2011.

SEBRAE. Brasileiros irão liderar as vendas no mercado de beleza masculina. 07 2018. Disponível em: <https://blog.sebrae-sc.com.br/brasileiros-mercado-de-beleza-masculina/>.