

UNIVERSIDADE DO VALE DO SAPUCAÍ
GUSTAVO HENRIQUE MARTINS

NATER MED
AGENDAMENTO ONLINE
PARA O POSTO DE SAÚDE DE NATÉRCIA-MG

POUSO ALEGRE, MG
2017

UNIVERSIDADE DO VALE DO SAPUCAÍ
GUSTAVO HENRIQUE MARTINS

NATER MED
AGENDAMENTO ONLINE
PARA O POSTO DE SAÚDE DE NATÉRCIA-MG

Trabalho de Conclusão de Curso apresentado ao curso de Sistemas de Informação, da Universidade do Vale do Sapucaí, como requisito parcial para a obtenção do título de bacharel em Sistemas de Informação.

Orientador: Prof. Ednardo David Segura

POUSO ALEGRE, MG

2017

UNIVERSIDADE DO VALE DO SAPUCAÍ
GUSTAVO HENRIQUE MARTINS

NATER MED
AGENDAMENTO ONLINE
PARA O POSTO DE SAÚDE DE NATÉRCIA-MG

Trabalho de conclusão de curso defendido e aprovado em: 23/11/2017 pela banca examinadora constituída pelos professores:

Prof. Ednardo David Segura

Orientador

Prof. Me. José Luiz da Silva

Avaliador

Prof. Ma. Valéria Santos Paduan Silva

Avaliadora

Martins, Gustavo Henrique.

NATER MED AGENDAMENTO ONLINE PARA O
POSTO DE SAÚDE DE NATÉRCIA-MG/Gustavo Henrique
Martins – Pouso Alegre, MG: Univás, 2017.

Trabalho de conclusão de curso (graduação) – Universidade
do Vale do Sapucaí, Univás, Sistemas de Informação.

Orientador: Ednardo David Segura

1 – Agendamento. 2 – Aplicação *on-line*. 3 – Saúde pública.

4 – Unidade Básica de Saúde.

AGRADECIMENTOS

Agradeço, primeiramente, a Deus, por ter me dado saúde, força, coragem e paciência para superar as dificuldades durante esta longa caminhada, e fazer com que tudo acontecesse da melhor forma, não somente nesses anos como universitário, mas em todos os momentos da minha vida.

Agradeço à minha família, em especial ao meu pai, Sebastião Raimundo e minha tia, Maria Sebastiana, que acreditaram e me ajudaram muito nessa conquista estando sempre presentes, me apoiando, me incentivando e investindo em meu sonho. Também, agradeço a minha mãe, Vitória Fernandes, que mesmo não estando presente no meio de nós sempre foi um exemplo de força, fé e coragem para mim e que lá do céu intercedeu por essa conquista. Aos meus irmãos, sobrinhos, tios que me apoiaram de alguma forma. E a minha namorada, Sânyara, pelo amor, paciência, compreensão, dedicação e incentivo, principalmente na etapa final do curso.

Agradeço à Universidade do Vale do Sapucaí, direção e administração pela oportunidade de fazer o curso e poder alcançar meu sonho. Aos professores e ao orientador Ednardo David Segura pela paciência, orientação, apoio, confiança, compreensão e conhecimentos passados. Aos colegas de curso pela amizade, apoio, companheirismo e momentos de diversão. Ao meu amigo e parceiro de trabalho Douglas Oliveira que iniciou este trabalho comigo e que, por motivos médicos familiares, teve que trancar o curso, todavia continuou me apoiando nessa jornada.

Aos meus patrões Mariângela, Ederson e Larissa pelo apoio e compreensão.

Agradeço à prefeitura de Natércia por disponibilizar o transporte e aos motoristas, Cesinaldo e Cleydson, pela paciência e profissionalismo. Aos profissionais da Unidade Básica de Saúde, Vivian e Rodrigo, que me forneceram informações importantes para desenvolver este trabalho.

Enfim, agradeço a todas as pessoas que direta ou indiretamente fizeram parte dessa etapa decisiva em minha vida.

DEDICATÓRIA

Dedico este trabalho, primeiramente, a Deus; à minha família, em especial ao meu pai Sebastião Raimundo e à minha tia Maria Sebastiana; à memória de minha mãe Vitória que me fez e faz muita falta; aos meus irmãos, sobrinhos e tios pelo incentivo; aos meus amigos, pelas alegrias, tristezas e dores compartilhadas; à minha namorada, Sânyara, pela compreensão e amor; aos professores, pelos conhecimentos passados e a todos aqueles que de alguma forma estiveram e estão próximos de mim nessa caminhada.

MARTINS, Gustavo Henrique; **NATER MED AGENDAMENTO ONLINE PARA O POSTO DE SAÚDE DE NATÉRCIA-MG.** 2017. Monografia – Curso de SISTEMAS DE INFORMAÇÃO, Universidade do Vale do Sapucaí, Pouso Alegre – MG, 2017.

RESUMO

Este trabalho centra-se no agendamento *on-line* de consultas por meio de uma aplicação *WEB* - Nater Med Agendamento Online, cujo objetivo geral é otimizar e gerenciar o serviço de agendamento de consultas. A aplicação disponibiliza os horários e a quantidade de fichas que os profissionais atendem diariamente, relatório da quantidade de agendamentos realizados, tabelas de medicamento disponíveis e indisponíveis, aba de avaliação e notícias da Unidade Básica de Saúde do município de Natércia – MG, proporcionando maior comodidade, agilidade e acessibilidade para os pacientes e os profissionais da área de saúde que o utilizam. Além de ser uma maneira segura e ágil de marcar as consultas, visa a minimização do tempo de espera e permite um atendimento mais rápido, organizado, informatizado e qualificado. A aplicação tem o propósito de solucionar um problema social que afeta muitos usuários da rede pública de saúde, de forma prática e ágil.

Palavras-chave: Agendamento. Aplicação *on-line*. Saúde pública. Unidade Básica de Saúde.

MARTINS, Gustavo Henrique; **NATER MED AGENDAMENTO ONLINE PARA O POSTO DE SAÚDE DE NATÉRCIA-MG.** 2017. Monografia – Curso de SISTEMAS DE INFORMAÇÃO, Universidade do Vale do Sapucaí, Pouso Alegre – MG, 2017.

ABSTRACT

This paper settle on the online scheduling of appointment through a WEB application - Nater Med Online Scheduling, whose main objective is improve and manage the appointment scheduling service. The application provides the schedules and quantity of ticktes that professionals attend daily, report the number of schedules made, available and unavailable drug tables, assessment tab and news from the Basic Health Unit (UBS) of the city of Natércia - MG, providing more convenience, agility and accessibility for patients and health professionals who use it. In addition, it is a safe and agile way of scheduling appointments, aiming at minimizing waiting time and allowing faster, organized, computerized and qualified service. The application has the purpose of solving a social problem that affects many users of the public health network in a practical and agile way.

Key words: *Scheduling. Web application. Public Health. Basic Health Unit.*

LISTA DE FIGURAS

Figura 1 - Caso de uso dos atores	28
Figura 2 - Modelagem utilizando DBDesigner.....	33
Figura 3 - Repositório do GitHub.....	34
Figura 4 - Terminal do Git Bash.....	34
Figura 5 - Site Oficial do Node.js.....	35
Figura 6 - Logotipo.....	37
Figura 7 - Organização dos arquivos da aplicação.....	37
Figura 8 - Página de cadastro acesso dos usuários.....	42
Figura 9 - Página de <i>login</i> dos usuários.....	45
Figura 10 - Página exclusiva ao usuário.....	51
Figura 11 - Página de cadastro de horários.....	53
Figura 12 - Página de agendamento de consulta no ambiente do usuário.....	54
Figura 13 - Página de agendamento de consulta no ambiente da secretaria.....	54
Figura 14- Profissionais para agendamento.....	55
Figura 15 - Kendo Calendar.....	56
Figura 16 - Tabela de fichas de atendimento do usuário.....	61
Figura 17 - Tabela de fichas de atendimento da secretaria.....	62
Figura 18 - Modal de confirmação do agendar consulta.....	65
Figura 19 - Modal de confirmação do desmarcar consulta.....	65
Figura 20 - Pesquisar por paciente.....	70
Figura 21 - Relatório de agendamentos.....	76
Figura 22 - Hospedagem na Digital Ocean.....	76
Figura 23 - Página de cadastro do usuário.....	79
Figura 24 - Página de horários.....	80
Figura 25 - Página de notícias.....	81
Figura 26 - Página de medicamentos.....	81
Figura 27 - Campo de pesquisa de medicamentos.....	82
Figura 28 - Aba de avaliação.....	82
Figura 29 - Requisitos para avaliação.....	83
Figura 30 - Resultados da avaliação.....	83

LISTA DE TABELAS

Tabela 1 - Horários da UBS.....	31
---------------------------------	----

LISTA DE CÓDIGOS

Código 1 - Estrutura básica do HTML.....	18
Código 2 - Exemplo de formatação com CSS.	20
Código 3 - Definir o jQuery.....	24
Código 4 - Definir o Hightcharts.	25
Código 5 - Arquivo <code>package.json</code>	36
Código 6 - Configuração do documento <code>.gitignore</code>	39
Código 7 - Conexão com o banco de dados.....	40
Código 8 - Configuração do documento <code>server.js</code>	41
Código 9 - Cadastro de acesso dos usuários contido no <code>routes.js</code>	42
Código 10 - Constante <code>path</code>	43
Código 11 - Cadastro de acesso dos usuários contido no <code>controller.js</code>	43
Código 12 - Cadastro de acesso dos usuários contido no <code>service.js</code>	44
Código 13 - Constante de <code>connection</code>	44
Código 14 - <i>Login</i> de acesso dos usuários contido no <code>routes.js</code>	45
Código 15 - <i>Login</i> de acesso dos usuários contido no <code>passport.js</code>	46
Código 16 - <i>Login</i> de acesso dos usuários contido no <code>controller.js</code>	47
Código 17 - <i>Login</i> de acesso dos usuários contido no <code>service.js</code>	48
Código 18 - Rota de acesso dos usuários contido no <code>routes.js</code>	49
Código 19 - Rotas dos ambientes contido no <code>routes.js</code>	50
Código 20 - Autenticação se o usuário está logado e autorizado contido no <code>routes.js</code>	50
Código 21 - <i>Logout</i> da aplicação contido no <code>routes.js</code>	51
Código 22 - Atualizar sessão com usuário escolhido contido no <code>routes.js</code>	52
Código 23 - Cadastrar horário dos profissionais contido no <code>service.js</code>	53
Código 24 - Profissionais para agendamento.....	56
Código 25 - Criando o calendário.	57
Código 26 - Retornar agendamentos ambiente do usuário.	59
Código 27 - Retornar agendamentos ambiente da secretaria.	59
Código 28 - Rota com parâmetros usuário contido no <code>routes.js</code>	59

Código 29 - Rota com parâmetros secretaria contido no <code>routes.js</code> .	60
Código 30 - Retornar agendamento do usuário contido no <code>controller.js</code> .	60
Código 31 - Retornar agendamento da secretaria contido no <code>controller.js</code> .	60
Código 32 - Retornar agendamento do usuário contido no <code>service.js</code> .	60
Código 33 - Retornar agendamento da secretaria contido no <code>service.js</code> .	61
Código 34 - Criar tabela de agendamento do usuário.	63
Código 35 - Criar tabela de agendamento da secretaria.	64
Código 36 - Criando o <code>modal</code> de confirmação do agendar consulta no ambiente do usuário.	66
Código 37 - Realizar agendamento no ambiente do usuário contido no <code>service.js</code> .	68
Código 38 - Desmarcar agendamento no ambiente do usuário contido no <code>service.js</code> .	69
Código 39 - Pesquisar por paciente contido no <code>service.js</code> .	70
Código 40 - Criando o <code>modal</code> de confirmação do agendar consulta no ambiente da secretaria.	71
Código 41 - Realização do agendamento no ambiente da secretaria contido no <code>service.js</code> .	72
Código 42 - Desmarcar agendamento no ambiente da secretaria contido no <code>service.js</code> .	73
Código 43 - Criação do relatório.	74
Código 44 - Criação do relatório no ambiente da secretaria contido no <code>service.js</code> .	75

LISTA DE ABREVIATURAS E SIGLAS

AJAX	<i>Asynchronous JavaScript And XML</i>
API	<i>Application Programming Interface</i>
CNES	Cadastro Nacional de Estabelecimentos de Saúde
CSS	<i>Cascading Style Sheet</i>
DOM	<i>Document Object Model</i>
HTML	<i>HyperText Markup Language</i>
HTTP	<i>HyperText Transfer Protocol</i>
IDE	<i>Integrated Development Environment</i>
MDN	<i>Mozilla Developer Network</i>
NPM	<i>Node Package Manager</i>
SGBD	Sistema Gerenciador de Banco de Dados
SGML	<i>Standard Generalized Markup Language</i>
SUS	Sistema Único de Saúde
SQL	<i>Structured Query Language</i>
SVG	<i>Scale Vector Graphics</i>
UBS	Unidade Básica de Saúde
UML	<i>Unified Modeling Language</i>
VML	<i>Vector Markup Language</i>
XML	<i>eXtensible Markup Language</i>
WWW	<i>World Wide Web</i>
W3C	<i>World Wide Web Consortium</i>

SUMÁRIO

1. INTRODUÇÃO	15
2. QUADRO TEÓRICO	18
2.1. HTML.....	18
2.2. CSS.....	19
2.3. JAVASCRIPT	20
2.4. NODE.JS	21
2.5. EXPRESS JS.....	22
2.6. MySQL	22
2.7. BOOTSTRAP	23
2.8. JQUERY.....	24
2.9. HIGHCHARTS	25
3. QUADRO METODOLÓGICO	27
3.1. Tipo de pesquisa	27
3.2. Contexto da pesquisa.....	27
3.3. Diagrama de Casos de Uso	28
3.4. Instrumentos	29
3.5. Procedimentos.....	29
3.6. Desenvolvimento.....	30
4. DISCUSSÃO DOS RESULTADOS	77
5. CONCLUSÃO	85
REFERÊNCIAS.....	87

1. INTRODUÇÃO

Nas últimas décadas, a saúde brasileira vem passando por diversos problemas, dentre eles: a falta de médicos, a precariedade nas instalações e equipamentos, a dificuldade para a marcação de consulta e/ou exame devido a longa fila de espera, entre outros.

Segundo Oliveira (2007, p. 15),

Um dos ramos da medicina que mais vem se destacando, nem sempre pelo lado positivo, é a saúde pública, que possui como interesse fundamental, a preocupação com a saúde numa perspectiva coletiva. É neste ramo que a medicina enfrenta uma de suas maiores dificuldades. A começar que os recursos são escassos e, há muita demanda por serviços médicos, e quem depende da saúde pública não tem alternativa, a não ser esperar o atendimento nas filas.

As Unidades Básicas de Saúde (UBS) são a porta de entrada preferencial do Sistema Único de Saúde (SUS). O objetivo desses postos é “atender até 80% dos problemas de saúde da população, sem que haja a necessidade de encaminhamento para outros serviços, como emergências e hospitais”, segundo os médicos sanitaristas Chioro e Scaff (1999, p. 24).

Para a população que depende dos postos de saúde, que não tem condições financeiras de arcar com os custos de uma consulta particular e também com a compra de remédios, só resta a alternativa de chegar à Unidade Básica de Saúde (UBS) um dia antes da entrega das senhas e ficar horas na fila, esperando o horário do atendimento.

De acordo com o setor de atendimento da UBS do município de Natércia-MG, o processo de marcação de consultas é realizado com o uso de fichas de atendimento limitadas, ou seja, o atendimento é por ordem de chegada e com número restrito de vagas, respeitando a prioridade de atendimento a idosos, deficientes, gestantes e emergências. Na UBS, independentemente de classe social, é possível receber atendimento em pediatria, psiquiatria, psicologia, ginecologia, clínica geral, nutricionista e odontologia.

A organização de consultas na UBS de Natércia-MG funciona, atualmente, de acordo com as demandas programadas e espontâneas. A demanda programada é aquela que requer um agendamento prévio na UBS. Já a demanda espontânea é quando o

paciente comparece à UBS inesperadamente, por motivos que ele julgou como necessidade de saúde.

No município de Natércia-MG, o usuário tem o desafio de acordar cedo e deslocar-se à UBS, que para muitos é de difícil acesso, pois está localizada em um dos pontos mais altos da cidade. Ao chegar à Unidade, o tempo de espera para a obtenção de senhas limitadas de atendimento é grande.

Por tanto, considerando o cenário atual da situação das UBS no Brasil, principalmente no município de Natércia-MG, identificou-se a necessidade de desenvolvimento de uma aplicação *WEB*¹ com objetivo geral de otimizar e gerenciar o serviço de agendamento *on-line*² de consultas, disponibilizar os horários e a quantidade de fichas que os profissionais atendem diariamente e, ainda contar com outras funcionalidades de rápido e fácil acesso, proporcionando assim, maior comodidade, agilidade e acessibilidade para os pacientes e os profissionais da área de saúde que o utilizam. Diante dessa necessidade, criou-se o Nater Med Agendamento Online.

De acordo com Leismann (2008, p. 15),

Em meio ao stress do dia-a-dia, o atendimento médico necessita ser rápido e prático. Para isto, a informática tem se tornado cada vez mais eficaz, com sistemas de informação que auxiliam neste processo, sendo assim, uma aliada fundamental para um melhor desempenho no atendimento médico público. Apesar do crescimento dos recursos, muitos ambientes médicos ainda dispensam a informatização de seus processos, pela cultura e a visão de alguns médicos e o próprio consultório que utiliza de arquivos manuais.

Para alcançar este objetivo, foram traçados os seguintes objetivos específicos: a) desenvolver o cadastro de acesso do paciente; b) desenvolver o *login* dos usuários; c) desenvolver o ambiente do usuário; d) desenvolver o ambiente da secretaria da UBS.

Visto que o difícil acesso – principalmente no deslocamento de idosos, gestantes e portadores de necessidades especiais - faz com que informações sobre consultas, horários e serviços fiquem inacessíveis aos usuários da UBS, o atual trabalho justifica-se em âmbito social, já que no município de Natércia-MG não há um sistema de apoio mútuo entre os usuários e a UBS. Para os usuários sem conhecimento de informática e sem os recursos para acessar a aplicação na unidade de atendimento, estão disponíveis

¹ Rede que conecta computadores por todo o mundo.

² Remete a um estado estar disponível.

computadores com acesso à internet e colaboradores que os auxiliarão a realizarem seus cadastros e agendamentos.

Logo, a aplicação Nater Med Agendamento Online ajudará tanto a comunidade quanto a UBS, a realizar melhorias no processo de agendamento e de gerenciamento de consultas, permitindo um atendimento mais rápido, organizado, informatizado e qualificado. Desse modo, o sistema resolve o problema da ausência de um sistema eficiente que facilite a marcação de consultas e/ou forneça informações ao usuário sobre a UBS e suas disponibilidades. O sistema também possibilitou ao autor aplicar os conhecimentos adquiridos no curso em benefício da população, bem como, ter um contato direto com novas tecnologias que não fazem parte do currículo do curso.

2. QUADRO TEÓRICO

Neste capítulo são abordadas as tecnologias utilizadas para o desenvolvimento da aplicação. Para tal, foram utilizadas as seguintes tecnologias: HTML, CSS, JavaScript, Node.js, Express.js, MySQL, Bootstrap, jQuery e Highcharts.

2.1. HTML

Em 1980, foi proposto um projeto baseado no contexto de hipertexto por Tim Berners-Lee que, inicialmente, tinha como ideia a linguagem Pascal. Em 1989, Tim contou com a ajuda do estudante Robert Cailliau do CERN e implementou com sucesso uma comunicação entre um cliente *HyperText Transfer Protocol* (HTTP) e um servidor pela *internet*. O HTML teve seu surgimento em 1990 e sofreu várias alterações em sua especificação. Após o ano de 1995, o padrão passou a ser controlado sobre a *World Wide Web Consortium* (W3C), que é conhecido como um consórcio internacional no qual se agrupam órgãos governamentais, organizações independentes e empresas que visam desenvolver padrões para interpretar e criar conteúdo para a *WEB*.

HTML tem o significado de linguagem de marcação de texto, e sua primeira versão foi baseada em uma linguagem chamada *Standard Generalized Markup Language* (SGML) que era utilizada para estruturar os documentos; através dela o HTML herdou diversas *tags*³: por exemplo `<h1>`, para título e `<p>` para parágrafo.

A estrutura básica do HTML é observada no Código 1.

Código 1 - Estrutura básica do HTML.

```
01. <!DOCTYPE html>
02. <html lang="pt-br">
03. <head>
04.   <meta charset="UTF-8">
05.   <title>Nater Med</title>
06. </head>
07. <body>
08.   <h1>AGENDAMENTO ONLINE DE CONSULTAS</h1>
09. </body>
10. </html>
```

Fonte: Elaborado pelo autor (2017).

³ Marcações para organizar informações.

O HTML foi atualizado para o HTML 2.0, que é uma versão de correção da linguagem anterior. Após algum tempo, foi lançado o HTML 3.0 por Dave Raggett, porém essa versão não foi implementada e foi superada pela versão 3.2 que também era uma versão de correção da versão anterior. A versão 3.2 contou com a implementação de várias características como tabelas, texto flutuante, dentre outras. Após alguns anos, surgiu a versão 4.0 que contou, novamente, com a participação de Raggett e, em dezembro de 1999 o HTML 4.1 teve sua publicação contendo compatibilidades das versões anteriores.

Iniciada no ano de 2008, o HTML 5 é a versão mais recente da linguagem HTML. Foram feitas algumas alterações das demais como novas *Application Programming Interface* (API), uso da aplicação de modo *off-line* e melhoria na refinação de erros.

Segundo Preston Prescott, essa nova linguagem apresenta as seguintes características:

- Novos atributos;
- Edição *off-line*;
- Melhoramento de suporte para armazenamento de rede local.

Por ser a versão mais atualizada e conter essas e outras características, utilizou-se neste trabalho o HTML5 para estruturar e apresentar o conteúdo da aplicação *WEB*.

2.2. CSS

CSS (*Cascading Style Sheets*) em português significa folhas de estilo em cascata. Foi proposto em outubro de 1994, por Hakon Lie que tinha como propósito criar uma ferramenta que facilitasse a programação de sites. Por isso, pode ser definida como um mecanismo que auxilia o programador a adicionar formatações e estilos (cores, fontes) aos documentos *WEB*.

Um exemplo de código CSS é mostrado no Código 2, em que se usa `color` e `font-size` para mudar a cor e tamanho da fonte do título.

Código 2 - Exemplo de formatação com CSS.

```
01. h1 {  
02.   color: #f0f0f0;  
03.   font-size: 40px;  
04. }
```

Fonte: Elaborado pelo autor (2017).

Vale ressaltar que a CSS pode ser tanto implementada dentro do HTML quanto em um documento separado, possuindo, então, uma grande vantagem: criar suas definições de estilo de página separadas e tornar possível várias páginas compartilhando a mesma aparência.

Utilizou-se nesse trabalho o CSS3, que é a linguagem mais recente encontrada. Foi desenvolvida na aplicação para tornar o *design* mais agradável, com boa aparência e usabilidade, e também por conter as novidades de sombras, transições, animações, gradientes, entre outros.

2.3. JAVASCRIPT

O criador do JavaScript foi Brendan Eich, que nasceu em 1961 nos Estados Unidos e ficou conhecido por seus trabalhos na NetScape e Mozilla; em 1995, deu início aos trabalhos na linguagem JavaScript. Desenvolvido inicialmente com o nome de Mocha, posteriormente modificado para LiveScript, e por fim JavaScript. LiveScript era o nome oficial da linguagem, quando teve o seu lançamento na primeira vez na versão beta do Netscape 2.0 em setembro de 1995; no entanto, teve seu nome alterado junto a Sun Microsystems em dezembro do mesmo ano, quando teve sua implementação no NetScape v2.0B3.

JavaScript é uma linguagem de *script* orientada a objetos, também conhecida por ser uma linguagem pequena e leve e atua dentro de um ambiente *host*⁴ (navegador *WEB*).

Segundo a MDN – *Mozilla Developer Network*: “JavaScript tem uma biblioteca padrão de objetos, como: *Array*, *Date*, e *Math*⁵, e um conjunto de elementos

⁴ Qualquer computador ou máquina conectado a uma rede, que conta com número de IP e nome definidos. Essas máquinas são responsáveis por oferecer recursos, informações e serviços aos usuários ou clientes.

que formam o núcleo da linguagem, tais como: operadores, estruturas de controle e declarações.”

Hoje, com o JavaScript é possível desenvolver aplicações *WEB* que possuem interatividade e capacidade de processamento equivalente às aplicações de um ambiente *desktop*.

O JavaScript foi escolhido para este trabalho por oferecer a vantagem de programação *cliente-side* e *back-side*, entre outras.

2.4. NODE.JS

O Node.js foi desenvolvido por Ryan Dahl e lançado em maio de 2009. Atualmente, seu desenvolvimento é mantido pela empresa da Linux Foundation, onde Dahl trabalha. Sua plataforma *open source*⁶ é construída sobre o *engine*⁷ de JavaScript V8 do Google e tem como objetivo de “fornecer uma maneira fácil de criar aplicativos de rede escaláveis” (NODE).

O Node.js é uma plataforma para criação de servidores rápidos e escaláveis, buscando manter um consumo reduzido de recursos computacionais, principalmente no que diz respeito à memória (TILKOV e VINOSKI, 2010, p. 80-83).

O *download* do Node pode ser facilmente feito em página oficial (<http://nodejs.org>).

Utilizou-se a tecnologia do Node.js, devido às suas vantagens:

- utiliza modelo I/O (*input/output*⁸) não-bloqueantes (*non-blocking*⁹) que o torna eficiente em aplicações em tempo real com intensa troca de dados;
- envio de diversas requisições em paralelo ao servidor que serão executadas em funções assíncronas (*async*¹⁰), gerando grande ganho em performance;
- gerenciador de pacotes NPM (*Node Package Manager*), o maior ecossistema de bibliotecas de código aberto no mundo;

⁵ Tradução: [Vetor, Data e Matemática].

⁶ Código fonte aberto.

⁷ Motor.

⁸ Tradução: [Entrada/Saída].

⁹ Sem bloqueio.

¹⁰ Tradução: [Assíncrono].

- contém uma documentação com guias abrangentes de seus usos;

2.5. EXPRESS JS

O Express js foi criado por TJ Holowaychuk em 2009, inspirado pelo *framework* Sinatra da linguagem Ruby. Segundo Almeida (2015), ele é um *framework WEB light-weight* que ajuda na organização de sua aplicação *WEB* na arquitetura MVC (Model View Controller) no lado do servidor.

Segundo Powers (2012, s.p.):

Um framework fornece suporte de infraestrutura que nos permite criar sites e aplicações mais rapidamente, fornecendo ao desenvolvedor um esqueleto sobre o qual construir e manusear muitos aspectos mundanos e ubíquos do processo de desenvolvimento de software e focar na criação de funcionalidades da nossa aplicação ou site. Também fornece coesão ao código, o que pode tornar o código mais fácil de gerenciar e manter (Powers 2012).

Por possuir a vantagem em acelerar e melhorar o desenvolvimento *WEB* o Node.js foi utilizado junto com o *framework* Express js para o desenvolvimento da API (*Application Programming Interface*) da aplicação.

2.6. MySQL

O MySQL teve seu início na década de 1990, pelos desenvolvedores David Axmark, Allan Larsson e Michael “Monty” Widenius, e é conhecido como o maior banco de dados de código aberto no mundo. É um sistema gerenciador de banco de dados (SGBD) que utiliza a linguagem SQL como interface, tem a capacidade de suportar grandes volumes de dados e contém uma interface simples e compatível com grande parte de sistemas operacionais que se tem nos dias de hoje.

Segundo Milani (2007), algumas de suas características principais são:

- **SGBD:** além de ser um armazém de dados contém outras funcionalidades como integridade dos dados, gerenciamento de acesso, dentre outros;

- formas de armazenamento: possui uma variedade de tabelas para armazenamento dos dados e por conter esse tipo de vantagem, é possível ter uma tabela para cada situação diferente;
- velocidade: contém alta velocidade pois usa o seu desenvolvimento com tabelas ISAM (substituído pelo MyISAM, na versão 5 do MySQL), utilizando também o *cache*¹¹ em consultas, dentre outras funcionalidades;
- segurança: sistema gerenciador de conexões que trabalha com criptografia no tráfego de senhas;
- SQL: bastante veloz, pois foi desenvolvido através de funções e códigos customizados pelos seus desenvolvedores;
- capacidade: tem um poder de armazenamento e execução altos; potente para executar milhões de consultas.

Então, dessa forma, a utilização do MySQL foi para manipular os dados e as informações; uma vez que é um banco que possui consistência, alta performance, é confiável e também de fácil utilização.

2.7. BOOTSTRAP

O Bootstrap foi criado em 2010 por dois desenvolvedores, um *designer* e outro desenvolvedor do Twitter. Hoje em dia, se tornou um dos *frameworks*¹² *front-end*¹³ mais populares do mundo, veio para facilitar a vida dos desenvolvedores *WEB* ao criar seus sites, e, além disso, também possui diversidades de componentes (*plug-ins*¹⁴) em JavaScript (jQuery).

Maurício Samy Silva define Bootstrap “como um poderoso, elegante e intuitivo *framework front-end* que possibilita um desenvolvimento *WEB* de modo ágil e fácil”.

A MDN (*Mozilla Developer Network*) o define como feito para pessoas de diversos níveis de habilidade, dispositivos de todas as formas e projetos de todos os tamanhos.

¹¹ Área de memória onde é mantida uma cópia temporária de dados armazenados.

¹² Conjunto de códigos abstratos e/ou genéricos.

¹³ Responsável por coletar a entrada do usuário.

¹⁴ Programa instalado no navegador permitindo o uso de recursos não presentes na linguagem HTML.

O download do bootstrap pode ser facilmente feito na sua página oficial (<http://getbootstrap.com>).

O Bootstrap foi usado, neste trabalho, para tornar o desenvolvimento mais rápido e acessível.

2.8. JQUERY

O jQuery foi desenvolvido por John Resig e lançado em dezembro de 2006. Segundo a MDN – *Mozilla Developer Network*, “jQuery é uma biblioteca que se concentra em simplificar o uso do JavaScript”. De acordo com a página oficial do jQuery (2017), ele é definido da seguinte maneira:

jQuery é uma biblioteca de JavaScript rápida, pequena e rica em recursos. Ele faz coisas como manipulação de elementos HTML do documento, manipulação de eventos, animação e solicitações Ajax, de maneira muito mais simples. Com uma API fácil de usar, ele ainda funciona em uma multiplicidade de navegadores. Em uma combinação de versatilidade e capacidade de extensão, jQuery mudou a maneira de milhões de pessoas escreverem JavaScript.

Algumas de suas funcionalidades são:

- Manipulação HTML / DOM;
- Manipulação CSS;
- Métodos de eventos HTML;
- Efeitos de animações;
- AJAX.

Antes de iniciar a utilização, foi necessário carregar por meio *tags*¹⁵ de *scripts*¹⁶ as bibliotecas do jQuery, conforme mostrado no Código 3.

Código 3 - Definir o jQuery.

```
01. <script type="text/javascript" src="jquery-3.1.1.min.js"></script>
```

Fonte: Elaborado pelo autor (2017).

¹⁵ Etiquetas.

¹⁶ Conjunto de instruções para que uma função seja executada em determinado aplicativo.

Algumas das vantagens que essa biblioteca trouxe foi a simplificação e o suporte a múltiplos navegadores *WEB* (*cross-browser*¹⁷). E no trabalho, usufruiu-se de suas facilidades para a criação de aplicações do lado do cliente (*client-side*¹⁸).

2.9. HIGHCHARTS

Segundo o IPARDES (2000a, p. 1), gráfico “[...] é a representação de dados e informação por meio de diagramas, desenhos, figuras ou imagens, de modo a possibilitar a interpretação da informação de forma rápida e objetiva”.

Alguns exemplos de gráficos que podem ser implementados usando o Highcharts são: gráficos de barras, gráficos de linhas, gráficos de pizza, entre outros que poderão ser utilizados de acordo com as necessidades.

O Highcharts utiliza-se do HTML, SVG (*Scale Vector Graphics*¹⁹) que fornecem compatibilidade para os navegadores e VML (*Vector Markup Language*²⁰) para *smartphones*²¹, com a finalidade de renderizar e visualizar os gráficos. (HIGHCHARTS, 2017).

Antes de iniciar a utilização, foi necessário carregar por meio de *tags* de *scripts* as bibliotecas do Highcharts, conforme mostrado no Código 4.

Código 4 - Definir o Hightcharts.

```
01. <script src="https://code.highcharts.com/highcharts.js"></script>
```

Fonte: Elaborado pelo autor (2017).

Para a criação dos gráficos, o usuário necessita informar dados que serão utilizados nas suas montagens, nas quais os dados poderão ser informados, por exemplo, enviando uma requisição para o servidor e retornando os dados de um banco de dados ou informando os dados de modo estático.

¹⁷ Múltiplos navegadores – suporte.

¹⁸ Lado do cliente.

¹⁹ Gráficos Vetoriais Escaláveis.

²⁰ Linguagem de Marcação Vetorial.

²¹ Telefone inteligente.

Para o desenvolvimento dos gráficos, utilizou-se nas gerações de relatórios da aplicação, a biblioteca Highcharts, que concedeu a formação dos mais simples aos mais complexos tipos de gráficos.

3. QUADRO METODOLÓGICO

Neste capítulo são apresentadas as metodologias para o desenvolvimento deste trabalho de pesquisa.

3.1. Tipo de pesquisa

Por meio de pesquisa, pretende-se adquirir conhecimento, práticas com o uso de novas ferramentas e, por fim, solucionar o problema de agendamento de consultas e outros serviços relacionados que se encontram na atual realidade.

Para Appolinário (2004, p.150), a definição de pesquisa é dada como:

Processo através do qual a ciência busca dar respostas aos problemas que se lhe apresentam. Investigação sistemática de determinado assunto que visa obter novas informações e/ou reorganizar as informações já existentes sobre um problema específico e bem definido.

Segundo Gil (2007, p. 17), pesquisa é definida como o “procedimento racional e sistemático que tem como objetivo proporcionar respostas aos problemas que são propostos. A pesquisa desenvolve-se por um processo constituído de várias fases, desde a formulação do problema até a apresentação e discussão dos resultados”.

Para a efetuação desse projeto foi utilizada a pesquisa aplicada cujo objetivo principal é obter conhecimento através de finalidades práticas.

De acordo com Barros e Lehfeld (2000, p.78),

A pesquisa aplicada tem como motivação a necessidade de produzir conhecimento para a aplicação de seus resultados, com o objetivo de contribuir para fins práticos, visando à solução mais ou menos imediata do problema encontrado na realidade.

Esse tipo de pesquisa foi essencial para o desenvolvimento da aplicação, já que visou solucionar um problema social que afeta muitos usuários da rede pública de atendimento do município de Natércia-MG, de forma prática e rápida.

3.2. Contexto da pesquisa

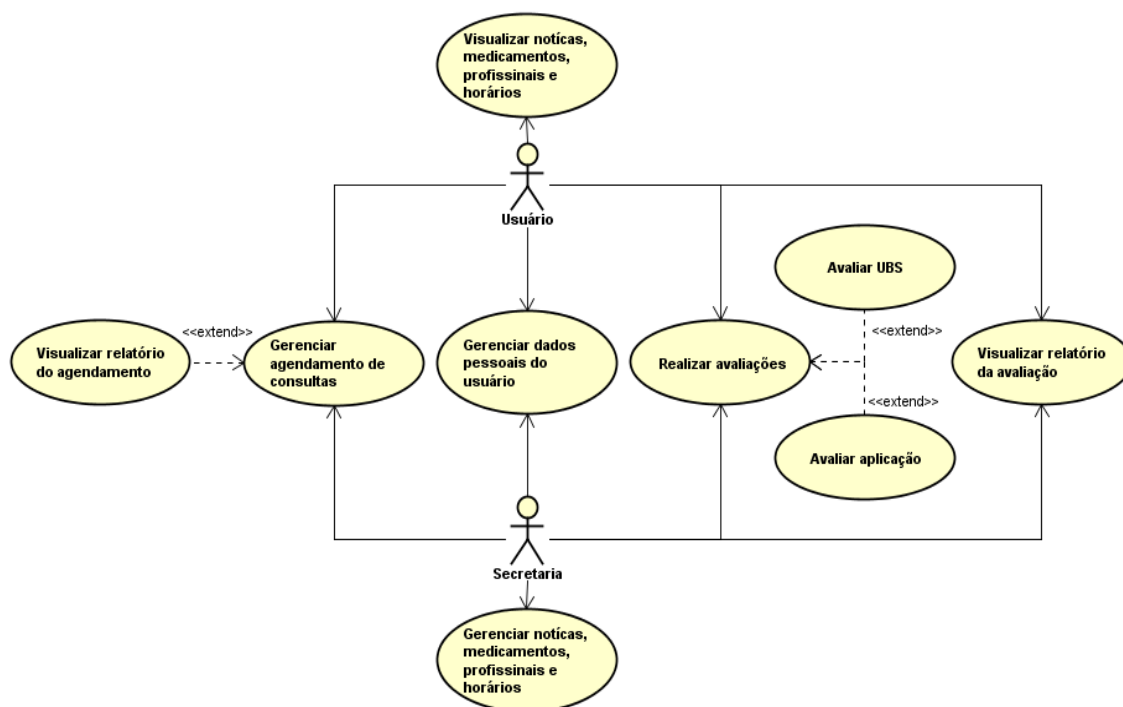
Este trabalho foi desenvolvido de acordo com uma necessidade identificada no município de Natércia-MG, localizada no sul de Minas Gerais com população estimada em cinco mil habitantes, referente ao agendamento de consultas, visando criar uma ferramenta *on-line* de gerenciamento de consultas que permite aos usuários uma maneira segura e ágil de marcações de consultas e exames objetivando a minimização do tempo de espera.

Para solucionar esse problema, a aplicação conta com soluções que trazem facilidade e comodidade para os colaboradores e usuários. A aplicação consiste em uma plataforma *on-line*, na qual é possível cadastrar os usuários e profissionais, agendar consultas e gerenciar horários e relatórios.

3.3. Diagrama de Casos de Uso

O diagrama de casos de uso tem a proposta de apresentar uma visão macro do funcionamento da aplicação. A Figura 1, em questão, mostra os casos de uso do usuário e da secretária.

Figura 1 - Caso de uso dos atores



Fonte: Elaborado pelo autor (2017).

3.4. Instrumentos

Os instrumentos utilizados no desenvolvimento dessa aplicação foram reuniões entre o desenvolvedor e o orientador, bem como o levantamento de dados na UBS de Natércia-MG para verificar a necessidade da criação da solução proposta. Foi realizada também uma pesquisa semiestruturada, que de acordo com Manzini (1990/1991, p. 154).

Está focalizada em um assunto sobre o qual confeccionamos um roteiro com perguntas principais, complementadas por outras questões inerentes às circunstâncias momentâneas à entrevista. Para o autor, esse tipo de entrevista pode fazer emergir informações de forma mais livre e as respostas não estão condicionadas a uma padronização de alternativas.

Essa pesquisa foi realizada por meio de entrevistas informais, sem a aplicação de questionários, em que foi possível verificar, com os colaboradores da UBS, quais as reais necessidades referentes ao agendamento e gerenciamento de consultas, coletando, também, algumas dicas de possíveis funcionalidades para a aplicação proposta.

3.5. Procedimentos

A seguir são apresentados os procedimentos para o desenvolvimento da aplicação.

- pesquisar informações pertinentes ao funcionamento da UBS em Natércia-MG;
- desenvolver a arquitetura do banco de dados MySQL;
- configurar o ambiente de desenvolvimento;
- desenvolver o logotipo;
- desenvolver as API's e códigos;
- elaborar páginas de usuário;
- desenvolver o relatório;
- hospedar aplicação em um servidor.

Em seguida, são apresentados de forma detalhada os procedimentos utilizados no desenvolvimento deste trabalho.

3.6. Desenvolvimento

O desenvolvimento da aplicação teve início com a realização de uma pesquisa que foi de suma importância e serviu de base para o decorrer da criação da aplicação proposta. Primeiramente, realizou-se entrevistas e pesquisas com os colaboradores e responsáveis pela área da saúde do município de Natércia-MG, para obter informações sobre o funcionamento da gestão administrativa da UBS em relação a agendamento de consultas, entrega de senhas, distribuição de medicamentos, horário dos profissionais, dentre outros.

Em relação ao tipo de atendimento, verificou-se que o ginecologista, psicólogo, psiquiatra, nutricionista e dentista atendem por demanda programada, ou seja, é necessário agendamento prévio com a secretaria. Somente após o término dos tratamentos já iniciados com esses profissionais, é que novas vagas serão abertas; esses tratamentos têm prioridade de agendamento. Já o atendimento do pediatra e clínico geral são por demanda espontânea, ou seja, por ordem de chegada. Vale ressaltar que, todos os profissionais respeitam a prioridade de atendimento a idosos, deficientes, gestantes e emergências.

Os agendamentos variam de profissional para profissional adequando-se às suas particularidades de horários e serviços. Os horários e a quantidade de fichas de atendimento podem ser vistos na Tabela 1.

Tabela 1 - Horários da UBS.

Profissional	Segunda	Terça	Quarta	Quinta	Sexta
Dentista	07h00min, Ficha indeterminada AGENDADO	07h00min, Ficha indeterminada AGENDADO	07h00min, Ficha indeterminada AGENDADO	07h00min, Ficha indeterminada AGENDADO	07h00min, Ficha indeterminada AGENDADO
Dentista	---	07h00min, Ficha indeterminada AGENDADO	---	07h00min, Ficha indeterminada AGENDADO	---
Dentista	07h00min, Ficha indeterminada AGENDADO	---	07h00min, Ficha indeterminada AGENDADO	---	07h00min, Ficha indeterminada AGENDADO
Clínico Geral	07h00min, 20 Fichas	---	---	07h00min, 20 Fichas	---
Clínico Geral	---	08h00min, 20 Fichas	---	---	08h00min, 20 Fichas
Clínico Geral	---	---	08h00min, 20 Fichas	---	---
Ginecologista	07h00min, 20 Fichas AGENDADO	---	---	07h00min, 20 Fichas AGENDADO	---
Nutricionista	---	12h00min, 25 fichas AGENDADO	---	---	---
Pediatra	12h00min, 20 Fichas	07h00min, 20 Fichas	07h00min, 20 Fichas	09h00min, 20 Fichas	---
Psicólogo	---	---	07h30min, 10 Fichas AGENDADO	---	07h30min, 10 Fichas AGENDADO
Psiquiatra	---	12h30min, 10 Fichas AGENDADO	12h30min, Ficha indeterminada Palestra/Campanha	12h30min, 10 Fichas AGENDADO	12h30min, Ficha indeterminada Palestra/Campanha

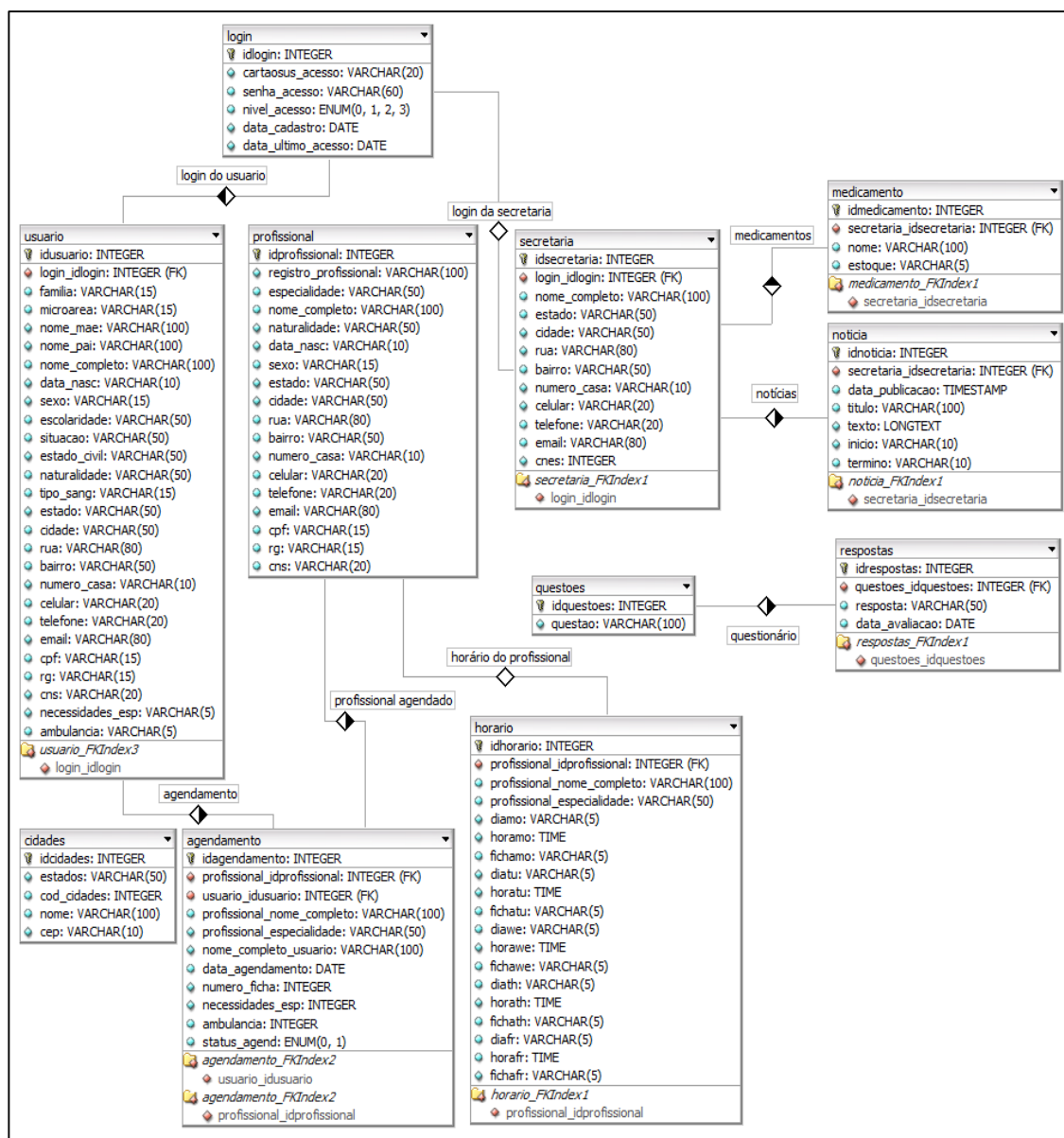
Fonte: UBS de Natércia (2017).

Os medicamentos seguem a Renam (Relação Nacional de Medicamentos Essenciais), lista que define os medicamentos que devem atender às necessidades de saúde da população brasileira do Sistema Único de Saúde (SUS).

No arquivo pessoal do usuário, há a identificação da família e da microárea, nome completo do paciente, da mãe, e opcional o do pai, data de nascimento, sexo, escolaridade, situação no mercado de trabalho, estado civil, naturalidade, tipo sanguíneo, endereço completo (estado, cidade, rua, bairro e número), celular/telefone, *email*, CPF, RG, CNS (Cartão Nacional do SUS); identifica-se ainda se portador de necessidade especial e a necessidade de ambulância para deslocamento.

Mediante a pesquisa realizada e com as informações mais relevantes em mãos, o próximo passo foi começar o desenvolvimento da aplicação cujo primeiro passo foi a definição da arquitetura do banco de dados, no qual a modelagem e implementação foi realizada utilizando o DBDesigner, como é apresentado na Figura 2.

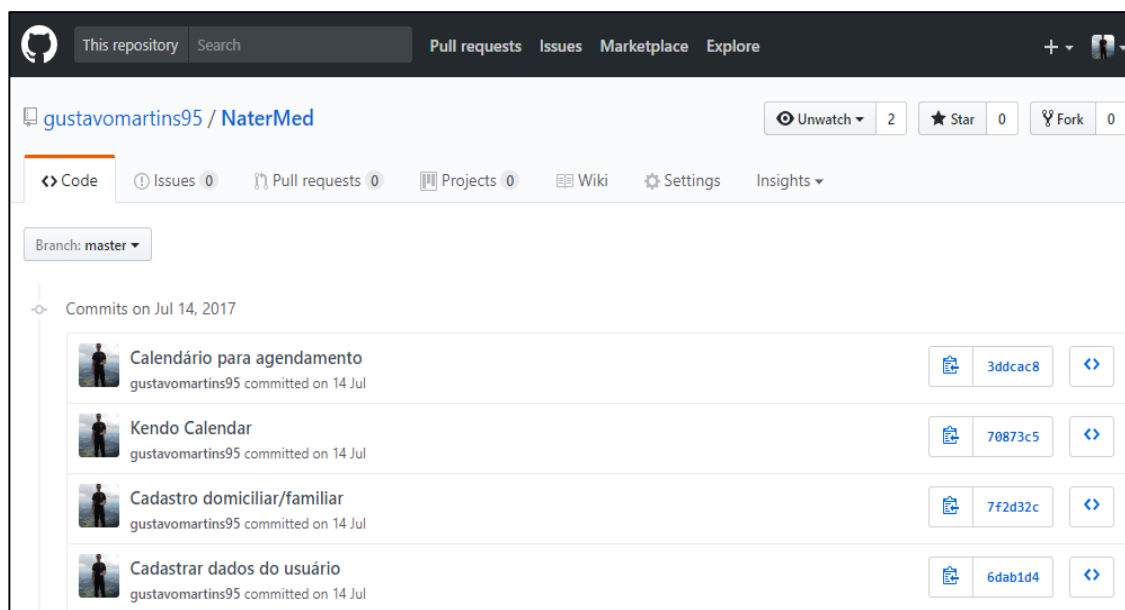
Figura 2 - Modelagem utilizando DBDesigner.



Fonte: Elaborado pelo autor (2017).

Após isso, configurou-se a plataforma gratuita do GitHub para controle de versão e hospedagem dos códigos desenvolvidos da aplicação. Nela, após o cadastro da conta, foi criado um repositório, de domínio público, cujo nome é NaterMed para hospedagem reservada da aplicação. A Figura 3 apresenta o repositório do GitHub utilizado.

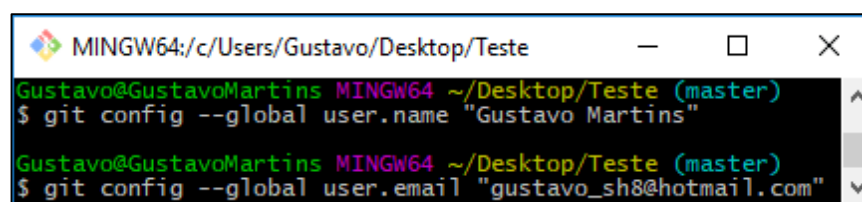
Figura 3 - Repositório do GitHub.



Fonte: Elaborado pelo autor (2017).

E, então, instalou-se o terminal próprio do GitHub (o Git Bash) para utilizar os comandos do Git. No Git Bash, o primeiro passo foi configurar o nome e *email* como mostrado na Figura 4. Caso queira verificar as configurações, utiliza-se o comando `git config -list` para listar todas as configurações encontradas naquele momento.

Figura 4 - Terminal do Git Bash.



Fonte: Elaborado pelo autor (2017).

Também foi instalado o Node.js cujo *download* pode ser feito em seu site oficial, como mostrado na Figura 5 e a instalação em sua máquina se dá normalmente. No terminal, após a instalação, foi possível verificar a versão usando `node -v`.

Figura 5 - Site Oficial do Node.js.



Fonte: Site Oficial do Node.js (2017).

Para gerenciar as dependências da aplicação foi usado o NPM. Por meio do comando `npm init`, iniciou-se um questionário de linha de comando que se concluiu com a criação de um `package.json` no diretório que iniciou o comando. Para adicionar as dependências, foi utilizado `npm install <package_name> --save`.

- `async`: é um módulo que fornece funções diretas e avançadas para trabalhar com JavaScript assíncrono;
- `bcryptjs`: usado para manter a senha dos usuários em sigilo e protegida. Ele criptografa a senha antes de salvar no banco de dados;
- `body-parser`: usado para facilitar a recuperação os dados submetidos nos formulários HTML. A recuperação foi feita por meio do `req.body`;
- `express`: usado para o desenvolvimento da API da aplicação;

- `express-session`: usado para criar dados de sessão, que foi usado na autenticação do `passport`;

- `http-status`: usado para facilitar o retorno dos códigos HTTP;

- `mysql`: usado para ter total conexão com o banco de dados criado;

- `passport`: usado na autenticação de *login*;

- `passport-local`: a autenticação foi feita usando os dados do próprio

banco de dados e não por credencias, como por exemplo, Facebook, Gmail e Twitter.

No Código 5 é mostrado o `package.json` do Nater Med, já com as dependências utilizadas.

Código 5 - Arquivo `package.json`.

```

01. {
02.   "name": "natermed",
03.   "version": "1.0.0",
04.   "description": "TCC S.I.",
05.   "main": "server.js",
06.   "scripts": {
07.     "test": "echo \"Error: no test specified\" && exit 1",
08.     "start": "node server.js"
09.   },
10.   "author": "Gustavo Martins",
11.   "license": "ISC",
12.   "dependencies": {
13.     "async": "^2.2.0",
14.     "bcryptjs": "^2.4.3",
15.     "body-parser": "^1.17.1",
16.     "express": "^4.15.2",
17.     "express-session": "^1.15.2",
18.     "http-status": "^1.0.1",
19.     "mysql": "^2.13.0",
20.     "passport": "^0.3.2",
21.     "passport-local": "^1.0.0"
22.   }
23. }
```

Fonte: Elaborado pelo autor (2017).

A partir disso, desenvolveu-se o logotipo, API, códigos e as interfaces da aplicação.

A ideia do nome para a aplicação surgiu pensando-se na facilidade da escrita, comunicação e entendimento pelos usuários. Chegou-se ao nome Nater Med Agendamento Online, e então se fez o *design* do logotipo com o auxílio da ferramenta *on-line freelogoservices.com*.

Para a representação gráfica escolheu-se um cardiograma - fazendo referência à saúde, e como plano de fundo uma placa mãe - referindo-se ao mundo tecnológico. As cores escolhidas foram o azul e o vermelho que, segundo as técnicas de marketing, fazem menção à saúde.

A definição do nome Nater Med Agendamento Online e o seu logotipo foi elaborado pelo autor do trabalho, como pode ser visto na Figura 6.

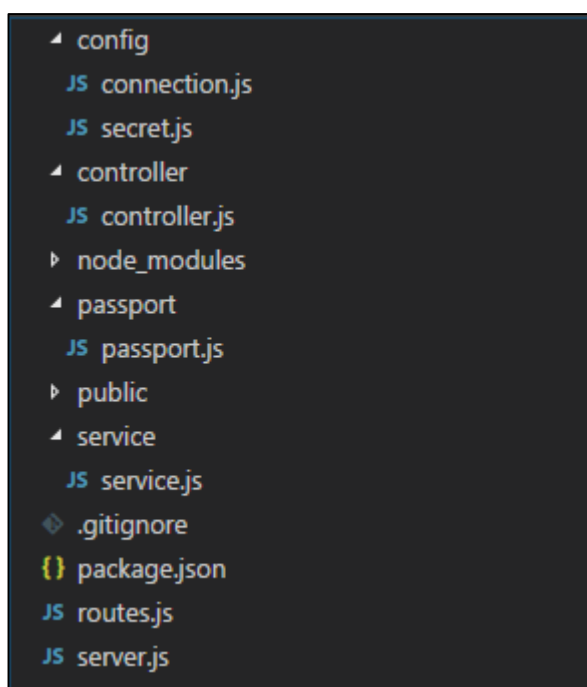
Figura 6 - Logotipo.



Fonte: Elaborado pelo autor (2017).

A organização da aplicação foi feita de forma que tudo tenha seu lugar e cada documento tem a sua responsabilidade, permitindo um trabalho mais centrado e modularizado. Na Figura 7 é mostrada a estrutura da aplicação.

Figura 7 - Organização dos arquivos da aplicação.



Fonte: Elaborado pelo autor (2017).

- `connection.js`: utilizado para estabelecer conexão com o banco de dados, provendo acesso às informações persistidas.
- `secret.js`: senha secreta usada na sessão criado pelo Express js.
- `server.js`: principal documento da aplicação, no qual ficou responsável criar o servidor, importar e configurar a maioria dos módulos usados na aplicação.
- `routes.js`: implementou a API da aplicação utilizando o *framework* Express js.
- `controller.js`: responsável por lidar com os dados e enviá-los ao `service.js` e/ou para o usuário.
- `service.js`: responsável pela leitura e escrita dos dados no banco de dados.
- `passport.js`: usado na autenticação do *login*.
- `package.json`: usado para gerenciar as dependências da aplicação.
- `.gitignore`: documento com regras utilizadas para ignorar o envio de arquivos temporários para o repositório do GitHub.
- `node_modules`: pasta em que todos os módulos instalados foram guardados.
- `public`: pasta em que os arquivos estáticos da aplicação foram guardados.

A configuração do documento `.gitignore` e a implementação usada é mostrada no Código 6.

Código 6 - Configuração do documento .gitignore.

```

01. # Created by https://www.gitignore.io/api/node
02. ### Node ###
03. # Logs
04. logs
05. *.log
06. npm-debug.log*
07. yarn-debug.log*
08. yarn-error.log*
09. # Runtime data
10. pids
11. *.pid
12. *.seed
13. *.pid.lock
14. # Directory for instrumented libs generated by jscoverage/JSCover
15. lib-cov
16. # Coverage directory used by tools like Istanbul
17. coverage
18. # nyc test coverage
19. .nyc_output
20. # Grunt intermediate storage
21. (http://gruntjs.com/creatingplugins#storing-task-files)
22. .grunt
23. # Bower dependency directory (https://bower.io/)
24. bower_components
25. # node-waf configuration
26. .lock-wscript
27. # Compiled binary addons (http://nodejs.org/api/addons.html)
28. build/Release
29. # Dependency directories
30. node_modules/
31. jspm_packages/
32. # Typescript v1 declaration files
33. typings/
34. # Optional npm cache directory
35. .npm
36. # Optional eslint cache
37. .eslintcache
38. # Optional REPL history
39. .node_repl_history
40. # Output of 'npm pack'
41. *.tgz
42. # Yarn Integrity file
43. .yarn-integrity
44. # dotenv environment variables file
45. .env

```

Fonte: Elaborado pelo autor (2017).

A conexão com o banco de dados MySQL e a implementação usada no documento é mostrada no Código 7.

Código 7 - Conexão com o banco de dados.

```
01. 'use strict'
02.
03. const mysql = require('mysql'),
04.     connection = mysql.createConnection({
05.       database: 'dbnatermed',
06.       host: 'localhost',
07.       user: 'root',
08.       password: ''
09.     })
10.
11. connection.connect(function (err) {
12.   if (err) {
13.     console.error('Error Connecting: ' + err)
14.     return
15.   }
16. })
17.
18. module.exports = connection
```

Fonte: Elaborado pelo autor (2017).

Para estabelecer a conexão foram definidas as seguintes opções:

- database: Nome do banco de dados a ser usado para esta conexão.
- host: O nome do host do banco de dados no qual está se conectando.
- user: O usuário MySQL para autenticar.
- password: A senha desse usuário MySQL.

Os documentos criados no *back-end* seguem a CommonJS, uma especificação de ecossistemas para o JavaScript, e a função embutida `require`, a maneira mais fácil de incluir módulos existentes em arquivos separados. O funcionamento básico do `require` é que ele lê o arquivo JavaScript, interpreta o *script* e, em seguida, retorna ao conteúdo do objeto `exports`. No Código 8, é mostrado a implementação usada na configuração do documento `server.js`.

Código 8 - Configuração do documento `server.js`.

```

01. 'use strict'
02.
03. const bodyParser = require('body-parser'),
04.     express = require('express'),
05.     passport = require('passport'),
06.     session = require('express-session'),
07.     keySecret = require('./config/secret'),
08.     app = express()
09.
10. app.use(bodyParser.urlencoded({
11.     extended: true
12. }))
13. app.use(bodyParser.json())
14. app.use(express.static("public"))
15. app.use(session({
16.     name: 'natermed',
17.     secret: keySecret,
18.     cookie: {
19.         maxAge: 3600000
20.     },
21.     resave: false,
22.     saveUninitialized: false
23. }))
24. app.use(passport.initialize())
25. app.use(passport.session())
26.
27. const port = 8080,
28.     hostname = "localhost"
29. app.listen(port, onStart())
30.
31. require('./passport/passport')(passport)
32. require('./routes')(app, passport)
33.
34. function onStart() {
35.     console.log(`Server started at http://${hostname}:${port}`)
36. }

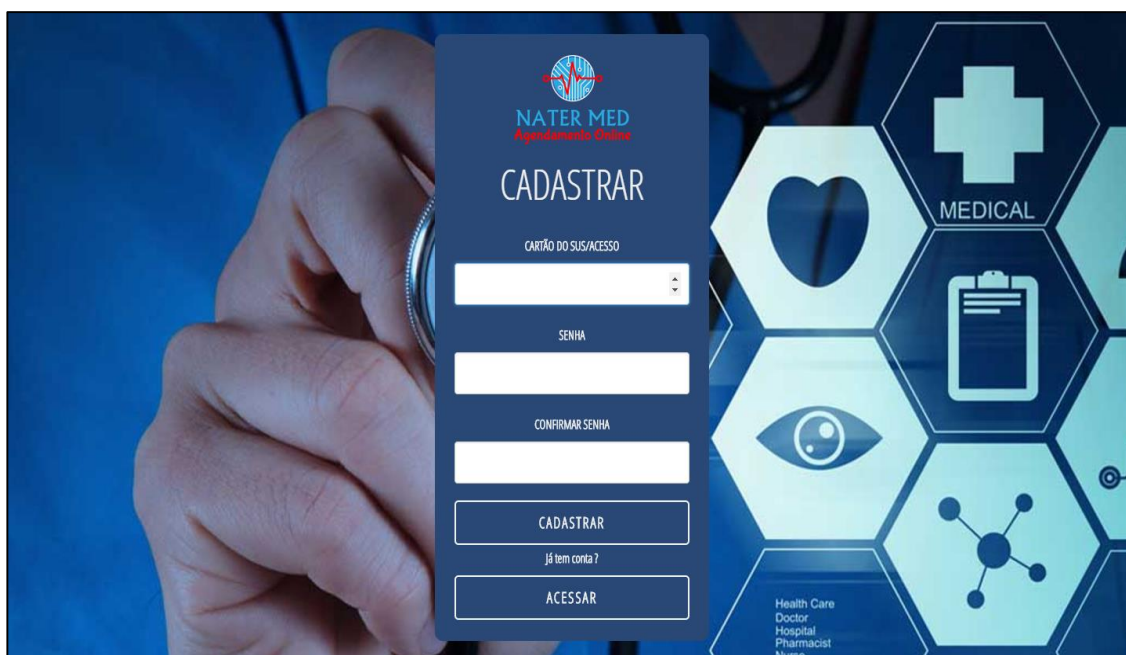
```

Fonte: Elaborado pelo autor (2017).

O cadastro de acesso dos usuários permite registrar o cartão de acesso (numérico) e a senha escolhida pelo usuário. Para utilizar a aplicação, primeiramente, é necessário que o usuário faça o cadastro de acesso no site. Para isso, é preciso que ele informe um cartão de acesso e uma senha, ambos a sua escolha. Já para a secretaria foi disponibilizado, pelo desenvolvedor, o cartão de acesso CNES (Cadastro Nacional de Estabelecimentos de Saúde) e uma senha que pode ser alterada pela secretaria, se necessário.

Para o cadastro do usuário, foi criado o documento `register.html` e dentro dele foi usado o formulário (form) com campos de `inputs` para obtenção e envio dos dados. Na Figura 8, é mostrado o resultado da página criada.

Figura 8 - Página de cadastro acesso dos usuários.



Fonte: Elaborado pelo autor (2017).

Os dados submetidos no formulário são enviados para a API da aplicação, como mostrado no Código 9, e contém o caminho (path) da rota `/register` com os métodos de roteamento `get`, responsável por entregar o arquivo para o HTML, e `post`, responsável por chamar o `controller.register`, derivados a partir dos métodos do HTTP. Dentro da função (function) de retorno, chamada de *callback*²², estão os objetos `req` (*request*²³), que representa a solicitação HTTP, e `res` (*response*²⁴), que representa a resposta HTTP.

Código 9 - Cadastro de acesso dos usuários contido no routes.js.

```

01. app.route('/register')
02.   .get(function (req, res) {
03.     res.sendFile(path + 'register.html')
04.   })
05.   .post(function (req, res) {
06.     controller.register(req, res)
07.   })

```

Fonte: Elaborado pelo autor (2017).

²² Função de retorno.

²³ Informação chegando no servidor através do navegador.

²⁴ Informação chegando no navegador através do servidor.

Para entregar o arquivo para o HTML, foi necessário indicar o diretório que o arquivo está. Como mostrado no Código 10, foi criada a constante `path` com `__dirname`, que retorna o diretório em que o *script* é executado, e concatenado com o nome da pasta `public`, estão os arquivos estáticos da aplicação.

Código 10 - Constante `path`.

```
01. | const path = __dirname + '/public/'
```

Fonte: Elaborado pelo autor (2017).

Os dados que chegam ao documento `controller.js`, como mostrado no Código 11, utilizam-se da função `body-parser` para facilitar a recuperação dos dados submetidos no formulário do HTML e são enviados ao `service.js`. A função `body-parser` monta objetos JSON, a partir dos campos `name` e `value` dos `inputs` criados no `form`.

Código 11 - Cadastro de acesso dos usuários contido no `controller.js`.

```
01. | // Registrar novo login/cartão
02. | register: function (req, res) {
03. |   service.register(req.body, function (error, status, message) {
04. |     res.status(status).json({ message: message })
05. |   })
06. | }
```

Fonte: Elaborado pelo autor (2017).

No documento `service.js`, mostrado no Código 12 nas linhas de 3 a 5, é o local em que a senha do usuário é criptografada. O campo do cartão de acesso no banco de dados está anotado como `UNIQUE`, e ao se tentar salvar um dado igual é retornado o erro `ER_DUP_ENTRY`.

Código 12 - Cadastro de acesso dos usuários contido no `service.js`.

```

01. // Registrar novo login/cartão
02. register: function (data, callback) {
03.     let hashedPassword = bcrypt.hashSync(
04.         data.txtSenha_Acesso, 10
05.     ),
06.     dataAtual = new Date(),
07.     sql = 'INSERT INTO login ' +
08.         '(cartaosus_acesso, senha_acesso, data_cadastro) ' +
09.         'VALUES (?, ?, ?)'
10.
11.     // Query no Banco de Dados
12.     connection.query(sql, [data.txtCartaosus_Acesso,
13.         hashedPassword, dataAtual], function (error, result) {
14.         if (error) {
15.             if (error.code == 'ER_DUP_ENTRY')
16.                 callback(error, httpStatus.CONFLICT, 'Cartão ' +
17.                     data.txtCartaosus_Acesso + ' já está em uso.')
18.             else
19.                 callback(error, httpStatus.INTERNAL_SERVER_ERROR,
20.                     'Desculpe-nos :( Tente novamente.')
21.         } else {
22.             callback(null, httpStatus.CREATED, 'Cartão ' +
23.                 data.txtCartaosus_Acesso + ' cadastrado com sucesso.')
24.         }
25.     })
26. }

```

Fonte: Elaborado pelo autor (2017).

Para se conectar com o banco de dados, foi preciso incluir o documento criado anteriormente no Código 7. No Código 13, é mostrado como criou-se a constante que foi adicionada conforme a necessidade da aplicação.

Código 13 - Constante de `connection`.

```

01. | const connection = require('../config/connection')

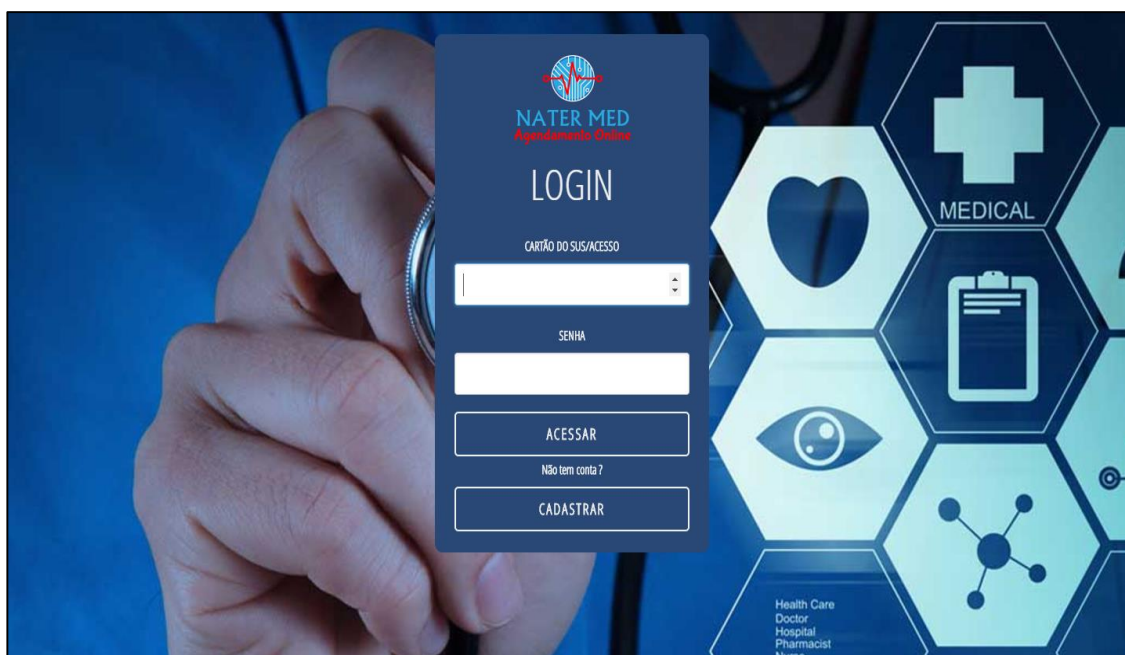
```

Fonte: Elaborado pelo autor (2017).

Por fim, como resultado, teve-se um cadastro dos usuários. E, para que o usuário possa acessar a aplicação e usufruir de todas suas vantagens, é necessário fazer o *login* na aplicação. O *login* dos usuários é seguro e permite acessar o ambiente do usuário ou da secretaria;

Foi criado o documento `login.html` e dentro dele foi usado formulário (form) com campos de `inputs` para obtenção e envio dos dados. Na Figura 9, é mostrado o resultado do formulário criado.

Figura 9 - Página de *login* dos usuários.



Fonte: Elaborado pelo autor (2017).

Os dados submetidos no formulário foram enviados para a rota `/login` da API, como mostrado no Código 14.

Código 14 - *Login* de acesso dos usuários contido no `routes.js`.

```

01.  /* Login */
02.  app.route('/login')
03.    .get(function (req, res) {
04.      res.sendFile(path + 'login.html')
05.    })
06.    .post(function (req, res, next) {
07.      passport.authenticate('local-login',
08.        function (error, user, info) {
09.          if (error) return res.status(500).json({ info:
10.            'Desculpe-nos :( Tente novamente.' })
11.          if (!user) return res.status(403).json({ info })
12.          req.login(user, function (error) {
13.            if (error) return res.status(500).json({ info:
14.              'Desculpe-nos :( Tente novamente.' })
15.            return res.status(200).json({ info })
16.          })
17.        })
18.      )(req, res, next)
19.    })

```

Fonte: Elaborado pelo autor (2017).

Para que o *login* fosse seguro, teve-se o cuidado de seguir os passos encontrados na documentação do Passport. Após o usuário submeter o formulário de *login*, um

POST²⁵ é pedido para rota `/login` e é feita a execução do `passport.authenticate`. Esse *middleware*²⁶ de autenticação está configurado para levar os dados para função de autenticação local, mostrada nas linhas de 12 a 27 do Código 15.

Código 15 - Login de acesso dos usuários contido no `passport.js`.

```

01. passport.serializeUser(function (user, done) {
02.   let sessionUser = {
03.     idlogin: user.idlogin,
04.     nivel_acesso: user.nivel_acesso,
05.     idusuario: user.idusuario
06.   }
07.   done(null, sessionUser)
08. })
09. passport.deserializeUser(function (sessionUser, done) {
10.   done(null, sessionUser)
11. })
12. passport.use('local-login',
13.   new LocalStrategy({
14.     usernameField: 'txtCartaosus_Acesso',
15.     passwordField: 'txtSenha_Acesso'
16.   }, function (txtCartaosus_Acesso, txtSenha_Acesso, done) {
17.     controller.login(txtCartaosus_Acesso, txtSenha_Acesso,
18.       function (error, user, message) {
19.         // exceção
20.         if (error) return done(error)
21.         // falha de autenticação
22.         if (!user) return done(null, false, message)
23.         // autenticado
24.         return done(null, user)
25.       })
26.   })
27. )

```

Fonte: Elaborado pelo autor (2017).

Na linha 17 do Código 15, os dados foram enviados para o documento `controller.js`, mostrados no Código 16 e enviados ao `service.js`, mostrado no Código 17, para verificar se os dados correspondem aos que estão no banco de dados.

²⁵ Dados anexados ao corpo da mensagem de requisição.

²⁶ É utilizado para transportar informações entre programas.

Código 16 - *Login* de acesso dos usuários contido no `controller.js`.

```
01. // Login usuário
02. login: function(txtCartaosus_Acesso, txtSenha_Acesso, callback){
03.     service.access(txtCartaosus_Acesso, txtSenha_Acesso,
04.         function(error, status, message, user) {
05.             if (status == httpStatus.OK
06.                 callback(null, user)
07.             else if (status == httpStatus.UNAUTHORIZED)
08.                 callback(null, false, message)
09.             else
10.                 callback(error)
11.         })
12. }
```

Fonte: Elaborado pelo autor (2017).

No Código 17, teve-se que usar o módulo `async` para verificar os dados do banco de dados passo a passo. Primeiro, foi verificado se o cartão informado é encontrado, depois foi criptografado a senha digitada e comparada com a que está no banco de dados e, por fim, foi feita a atualização da última data de acesso do usuário.

Código 17 - Login de acesso dos usuários contido no service.js.

```

01. // Login
02. access: function(txtCartaosus_Acesso, txtSenha_Acesso, callback){
03.     async.waterfall([
04.         dbLogin,
05.         dbPass,
06.         dbUpdateLastLogin
07.     ], function (error, status, message, user) {
08.         if (error) callback(error, status, message)
09.         else callback(error, status, message, user)
10.     })
11.     function dbLogin(cb) {
12.         let sql = 'SELECT * FROM login ' +
13.             'WHERE cartaosus_acesso = ? LIMIT 1'
14.         connection.query(sql, txtCartaosus_Acesso,
15.             function (error, result) {
16.                 if (error) {
17.                     cb(error, httpStatus.INTERNAL_SERVER_ERROR,
18.                         'Desculpe-nos :( Tente novamente.')
19.                 } else {
20.                     if (result == null || result.length == 0) {
21.                         cb(new Error(), httpStatus.UNAUTHORIZED,
22.                             'Cartão não encontrado.')
23.                     } else {
24.                         cb(null, result[0])
25.                     }
26.                 }
27.             })
28.     }
29.     function dbPass(dbResult, cb) {
30.         if (bcrypt.compareSync(txtSenha_Acesso,
31.             dbResult.senha_acesso)) {
32.             cb(null, dbResult)
33.         } else {
34.             cb(new Error(), httpStatus.UNAUTHORIZED, 'Senha inválida.')
35.         }
36.     }
37.     function dbUpdateLastLogin(dbResult, cb) {
38.         let dataAtual = new Date(),
39.         sql = 'UPDATE login SET data_ultimo_acesso = ? ' +
40.             'WHERE idlogin = ?'
41.         connection.query(sql, [dataAtual, dbResult.idlogin],
42.             function (error, result) {
43.                 if (error) {
44.                     cb(error, httpStatus.INTERNAL_SERVER_ERROR,
45.                         'Desculpe-nos :( Tente novamente.')
46.                 } else {
47.                     cb(null, httpStatus.OK, 'Login efetuado com sucesso.',
48.                         dbResult)
49.                 }
50.             })
51.     }
52. }

```

Fonte: Elaborado pelo autor (2017).

No Código 16, por meio da call-back, os dados foram retornados para autenticação local no Código 15. Caso der uma exceção interagindo com o banco de

dados, é preciso invocar `done (err)`. Quando não é possível encontrar o usuário ou as senhas, invoca-se `done (null, false)` ou um argumento opcional será passado; no caso, foi passada a mensagem `done (null, false, message)`. Se tudo correu bem e quer-se que o usuário faça o *login*, invoca-se `done (null, user)`. Chamado, o `done` fará o fluxo de volta ao `passport.authenticate` e, se o usuário for passado, o *middleware* irá chamar o `req.login`. Isso chamará o método `passport.serializeUser`, mostrado no Código 15, para acessar o objeto usuário que passa e determinar quais dados vão ser salvos na sessão. O resultado do método `passport.serializeUser` é anexar à sessão como `req.session.passport.user`. Para recuperar os dados que estão salvos na sessão, é só solicitar o `req.session.passport.user` na rota da API desejada.

Com os dados salvos na sessão, no *front-end*, foi feito redirecionamento para a rota `/access`, por meio do `window.location = "/access"`, que verifica qual o nível de acesso do usuário e o redireciona para a rota do seu ambiente. No Código 18, é mostrado o redirecionamento no documento `routes.js`, para os ambientes.

Código 18 - Rota de acesso dos usuários contido no `routes.js`.

```

01.  /* Rotas de acesso */
02.  app.get('/access', isLoggedIn, function (req, res) {
03.    if (req.session.passport.user.nivel_acesso == '1')
04.      res.redirect('/secretaria')
05.    else if (req.session.passport.user.nivel_acesso == '3')
06.      res.redirect('/usuario')
07.    else
08.      res.redirect('/logout')
09.  })

```

Fonte: Elaborado pelo autor (2017).

No Código 19, são mostradas as rotas que foram redirecionadas no Código 18.

Código 19 - Rotas dos ambientes contido no routes.js.

```

01.  /* Ambiente do usuário */
02.  app.get('/usuario', isLoggedIn, isAuthorized(['3']),
03.    function (req, res) {
04.      res.sendFile(path+'users/usuario/escolherUsuario.html')
05.    })
06.  /* Ambiente da secretaria */
07.  app.get('/secretaria', isLoggedIn, isAuthorized(['1']),
08.    function (req, res) {
09.      res.sendFile(path+'users/secretaria/escolherSecretaria.html')
10.    })

```

Fonte: Elaborado pelo autor (2017).

Antes de acessar o respectivo ambiente, foram colocadas duas funções de *middleware* para autenticação, são elas:

1. `isLoggedIn()`: verifica se o usuário está logado na aplicação. A validação foi, facilmente, feita com a função `req.isAuthenticated` que retorna um booleano verdadeiro ou falso.

2. `isAuthorized()`: é verificado se tem autorização para acessar a esta rota. Ao chamar a função na rota, foi passado como parâmetro quais os níveis de acesso que podem acessá-la e comparado com o nível de acesso que está salvo na sessão. A comparação foi feita com o método `indexOf()`, que retorna -1 caso não esteja presente.

Código 20 - Autenticação se o usuário está logado e autorizado contido no routes.js.

```

01.  function isLoggedIn(req, res, next) {
02.    if (req.isAuthenticated())
03.      return next()
04.    else
05.      res.redirect('/logout')
06.  }
07.
08.  function isAuthorized(access) {
09.    return function (req, res, next) {
10.      if(access.indexOf(
11.        req.session.passport.user.nivel_acesso)!= -1)
12.        return next()
13.      else
14.        res.redirect('/logout')
15.    }
16.  }

```

Fonte: Elaborado pelo autor (2017).

Nas validações do Código 20, caso não atendidos os requisitos, utilizou-se o `res.redirect` para redirecionar o usuário para a rota `/logout`. Na rota, para

encerrar a sessão poderia ser usada somente a função expressada na documentação do Passport, o `req.logout`; porém, na aplicação, usou-se também as funções do Express.js para efetuar essa operação, como mostrado no Código 21.

Código 21 - Logout da aplicação contido no `routes.js`.

```

01.  /* Logout */
02.  app.get('/logout', function (req, res) {
03.    req.session.destroy(function (error) {
04.      if (error) { return next(error) }
05.      res.clearCookie('natermed')
06.      req.logout()
07.      res.redirect('/login')
08.    })
09.  })

```

Fonte: Elaborado pelo autor (2017).

Por fim, ao passar pelas validações de segurança do *login*, será mostrada a página exclusiva do usuário que está logado. As páginas da secretaria e do usuário são parecidas visualmente; a primeira mostra o usuário previamente cadastrado e a segunda mostra a possibilidade de criar um cadastro domiciliar pelo usuário, ou seja, uma conta de usuário para cada integrante da família por registro de acesso. A Figura 10 apresenta a página do usuário já com os usuários cadastrados.

Figura 10 - Página exclusiva ao usuário.



Fonte: Elaborado pelo autor (2017).

As informações do *card* são retornadas ao banco de dados, no qual é feito um `select` na tabela usuário e retornado os dados que tem o mesmo `idlogin` salvo na sessão. O botão `acessar` tem o evento de `onclick` com o valor do `idusuario`

retornado na consulta acima, e ao clicar no botão do *card* escolhido é enviada uma requisição para a rota `/escolherusuario`. Na rota é feita a atualização dos campos salvos na sessão, como mostrado Código 21.

Código 22 - Atualizar sessão com usuário escolhido contido no `routes.js`.

```

01.  /* Salvar usuário escolhido na session */
02.  app.post('/escolherusuario', isLoggedIn, isAuthorized(['3']),
03.    function (req, res) {
04.      let newDataSession = []
05.      newDataSession.idlogin = req.session.passport.user.idlogin
06.      newDataSession.nivel_acesso =
07.        req.session.passport.user.nivel_acesso
08.      newDataSession.idusuario = req.body.idusuario
09.      // Atualizar session
10.      req.login(newDataSession, function (error) {
11.        return res.status(200).json({ message: "Logado" })
12.      })
13.    })

```

Fonte: Elaborado pelo autor (2017).

Um *array* foi criado e foram salvos os dados que já estavam alocados na sessão, e, por meio do `req.body.idusuario`, recuperou-se o `idusuario` escolhido. O *array* é passado para o `req.login`, que aciona o método `passport.serializeUser` e salva os novos dados.

Após a criação do acesso, feito o *login* e escolhido o usuário, desenvolveu-se a parte principal deste trabalho: o agendamento de consultas, que exigiu maior esforço e dedicação, pois, foi necessário que se adequasse a cada ambiente. E por fim, o agendamento no ambiente do usuário deu-se apenas para os profissionais que atendem por demanda espontânea, clínico geral e pediatra; no ambiente da secretaria, o agendamento deu-se para todos os profissionais, tanto para os que atendem por demanda espontânea quanto para os de demanda programada. A forma como foram feitas as páginas de agendamento nos dois ambientes são semelhantes.

Mas, antes de expor como foi feito o agendamento, é necessário salientar como foram elaborados os horários dos profissionais. A Figura 11, apresenta a página de cadastro de horários, e no Código 23, é mostrado como é foi feito o cadastro do horário dos profissionais no banco de dados.

Figura 11 - Página de cadastro de horários.

Escolha o profissional
Só é possível cadastrar um horário por profissional.

Dr. Alexandre - Clínico Geral ▼

Segunda-Feira Terça-Feira Quarta-Feira Quinta-Feira Sexta-Feira

Atende na segunda-feira? ☒

Início do atendimento 07:30

Quantidade de ficha 15

Cadastrar Limpar campos Voltar

Fonte: Elaborado pelo autor (2017).

Código 23 - Cadastrar horário dos profissionais contido no `service.js`.

```

01.  /* Operações do horário */
02.  cadastrarhorario: function (data, callback) {
03.      let sql = 'INSERT INTO horario ' +
04.          '(profissional_idprofissional, profissional_nome_completo, ' +
05.          'profissional_especialidade, diamo, horamo, ' +
06.          'fichamo, diatu, horatu, fichatu, diawe, horawe, fichawe, ' +
07.          'diath, horath, fichath, diafr, horafr, fichafr) ' +
08.          'VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)'
09.      // Query no Banco de Dados
10.      connection.query(sql,
11.          [data.txt_idProfissional, data.txt_Nome_Completo,
12.            data.txt_Especialidade,
13.            data.txt_diamo, data.txt_horamo, data.txt_fichamo,
14.            data.txt_diatu, data.txt_horatu, data.txt_fichatu,
15.            data.txt_diawe, data.txt_horawe, data.txt_fichawe,
16.            data.txt_diath, data.txt_horath, data.txt_fichath,
17.            data.txt_diafr, data.txt_horafr, data.txt_fichafr],
18.          function (error, result) {
19.              if (error) {
20.                  if (error.code == 'ER_DUP_ENTRY')
21.                      cb(error, httpStatus.CONFLICT, 'Só é possível
22.                        Cadastrar um horário por profissional.')
23.                  else
24.                      cb(error, httpStatus.INTERNAL_SERVER_ERROR,
25.                        'Desculpe-nos :( Tente novamente.')
26.              } else {
27.                  cb(null, httpStatus.OK, 'Cadastrado com sucesso.')
28.              }
29.          })
30.      }

```

Fonte: Elaborado pelo autor (2017).

Com o horário cadastrado, foi possível chegar ao resultado final do agendamento, como mostrado na Figura 12, que apresenta o ambiente do usuário, e a Figura 13, que apresenta o da secretaria.

Figura 12 - Página de agendamento de consulta no ambiente do usuário.

Selecione o profissional

Dr. Renan - Clínico Geral

* Agendamentos de consultas continuaram sendo feitas presencialmente.
* Não é possível agendar consultas aos sábados e domingos.

Dezembro 2017

Dom	Seg	Ter	Qua	Qui	Sex	Sab
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

Sábado, 11 Novembro, 2017

Ficha de Atendimento	Necessidades Especiais	Ações
1	✓	Marcada
2	✗	Desmarcar
3	✗	Marcada
4		Marcar
5		Marcar
6		Marcar
7		Marcar
8		Marcar

Fonte: Elaborado pelo autor (2017).

Figura 13 - Página de agendamento de consulta no ambiente da secretaria.

Procurar pelo nome do paciente

Selecione o paciente

Selecione o profissional

Dr. Renan - Clínico Geral

* Agendamentos de consultas continuaram sendo feitas presencialmente.
* Não é possível agendar consultas aos sábados e domingos.

Dezembro 2017

Dom	Seg	Ter	Qua	Qui	Sex	Sab
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

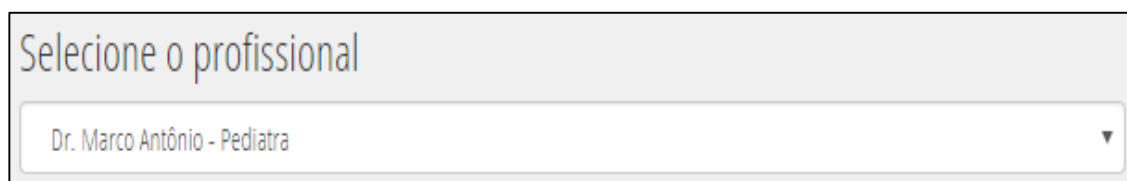
Sábado, 11 Novembro, 2017

Ficha de Atendimento	Família	Microárea	CNS	Nome do Paciente	Necessidades Especiais	Necessidade de Ambulância	Ações
1			705006293431451	Maria Sebastiana dos Reis	✓	✓	Desmarcar
2			707100388613220	Gustavo Henrique Martins	✗	✗	Desmarcar
3			708000885272528	Sebastião Raimundo Martins	✗	✗	Desmarcar
4							Marcar

Fonte: Elaborado pelo autor (2017).

O primeiro passo foi retornar do banco de dados todas as informações de horários dos profissionais. No entanto, no ambiente do usuário foi retornado somente o horário do clínico geral e do pediatra e, no da secretaria foram retornados os horários de todos os profissionais. As informações retornadas foram usadas para criar o `select`, como apresentado na Figura 14.

Figura 14- Profissionais para agendamento.

A imagem mostra uma interface de usuário com um cabeçalho cinza contendo o texto "Selecione o profissional" em uma fonte azul. Abaixo, há um campo de seleção (select) com o texto "Dr. Marco Antônio - Pediatra" e uma seta para baixo no canto inferior direito.

Fonte: Elaborado pelo autor (2017).

Para que o `select` contasse com todas as informações de horário, utilizou-se o novo atributo do HTML5 - `data`, usado para armazenar dados personalizados no HTML. Ele deve começar com o prefixo `data-` seguido do nome do atributo, o qual não deve conter letras maiúsculas. No Código 24 é apresentado como foi desenvolvido o `select`.

Código 24 - Profissionais para agendamento.

```

01. function criarSelect(hour) {
02.     for (index = 0; index < hour.length; index++) {
03.         newOptionItem=$( "<option value="
04.             +hour[index].profissional_idprofissional+" " +
05.             "data-nome_completo="
06.             +hour[index].profissional_nome_completo+" " +
07.             "data-especialidade="
08.             +hour[index].profissional_especialidade+" " +
09.             // Segunda - monday
10.             "data-diamo="+hour[index].diamo+" " +
11.             "data-horamo="+hour[index].horamo+" " +
12.             "data-fichamo="+hour[index].fichamo+" " +
13.             // Terça - tuesday
14.             "data-diatu="+hour[index].diatu+" " +
15.             "data-horatu="+hour[index].horatu+" " +
16.             "data-fichatu="+hour[index].fichatu+" " +
17.             // Quarta - wednesday
18.             "data-diawe="+hour[index].diawe+" " +
19.             "data-horawe="+hour[index].horawe+" " +
20.             "data-fichawe="+hour[index].fichawe+" " +
21.             // Quinta - thursday
22.             "data-diath="+hour[index].diath+" " +
23.             "data-horath="+hour[index].horath+" " +
24.             "data-fichath="+hour[index].fichath+" " +
25.             // Sexta - friday
26.             "data-diafr="+hour[index].diafr+" " +
27.             "data-horafr="+hour[index].horafr+" " +
28.             "data-fichafr="+hour[index].fichafr+">" +
29.             hour[index].profissional_nome_completo+" - " +
30.             hour[index].profissional_especialidade+"</option>")
31.         appendSelect(newOptionItem)
32.     }
33. }
34. function appendSelect(newOptionItem) {
35.     $("#txtEsp").append(newOptionItem)
36. }

```

Fonte: Elaborado pelo autor (2017).

O segundo passo foi criar o calendário e o escolhido foi o Kendo Calendar, como apresentado na Figura 15.

Figura 15 - Kendo Calendar.

Dezembro 2017						
Dom	Seg	Ter	Qua	Qui	Sex	Sab
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

Sábado, 11 Novembro, 2017

Fonte: Elaborado pelo autor (2017).

Para mostrar o calendário no HTML, de acordo com a documentação do Kendo Calendar, primeiro foi preciso criar a) `<div class="demo-section k-content">`. A partir disso, por meio do JavaScript foi anexada uma b) `<div id="calendar">`. Para chegar ao resultado apresentado na Figura 14, foi criado o calendário de acordo com o que está mostrado no Código 25.

Código 25 - Criando o calendário.

```

01. function criarCalendario() {
02.     newDivCalendar = '<div id="calendar"></div>'
03.     $('#demo-section').append(newDivCalendar)
04.     semana = ["mo", "tu", "we", "th", "fr", "sa", "su"]
05.     date = new Date()
06.     diaAtual = date.getDate()<10?'0'+date.getDate():date.getDate()
07.     mesAtual = (date.getMonth()+1)<10?'0'+(date.getMonth()+1):
08.         (date.getMonth()+1)
09.     anoAtual = date.getFullYear()
10.     $("#calendar").kendoCalendar({
11.         change: function () {
12.             diaSelecioneado = new Date(kendo.toString(this.value(), 'u'))
13.             // Retornar dados da data escolhida
14.             retonaragendamento(kendo.toString(this.value(), 'u'),
15.                 $('#txtEsp :selected').data('ficha' +
16.                     semana[diaSelecioneado.getDay()])))
17.         },
18.         min: new Date(anoAtual, (mesAtual - 1), diaAtual),
19.         max: new Date(anoAtual, 11, 31),
20.         disableDates: ["sa", "su",
21.             $('#txtEsp :selected').data('diamo') ? "" : "mo",
22.             $('#txtEsp :selected').data('diatu') ? "" : "tu",
23.             $('#txtEsp :selected').data('diawe') ? "" : "we",
24.             $('#txtEsp :selected').data('diath') ? "" : "th",
25.             $('#txtEsp :selected').data('diafr') ? "" : "fr"
26.         ],
27.         footer: "#: kendo.toString(data, 'D') #"
28.     })
29. }

```

Fonte: Elaborado pelo autor (2017).

Para a adaptação do calendário aos dias de atendimento de cada profissional, no desenvolvimento foram definidas as seguintes opções:

- linha 10: o calendário é instanciado;
- linha 11: ao clicar na data desejada ocorrer o evento (change);
- linha 18: data mínima que o calendário apresenta;
- linha 19: data máxima que o calendário apresenta;
- linhas 20 a 26: são os dias da semana que são desabilitados antes de apresentar o calendário;
- linha 27: mostra o dia atual.

Para desabilitar os dias da semana, nas linhas 20 a 26, foi necessário informar as duas primeiras letras do nome dos dias da semana em inglês. Aos sábados e domingos não há atendimento na UBS, por isso, as siglas “sa” de *saturday* e “su” de *sunday* já estão fixadas. Os demais dias da semana dependem do horário do profissional escolhido. Essas informações foram guardadas dentro do `select`, por meio do atributo `data`. Para validar o campo, usou-se o operador ternário, no qual, sua sintaxe é: condição ? expressão 1 : expressão 2, no qual o caracter “?” representa o condicional IF, e o caracter “.” representa o condicional ELSE. Caso a condição seja verdadeira, o operador retornará o valor da expressão 1; se não, ele retorna o valor da expressão 2.

No evento `change`, para recuperar a data escolhida foi usado o `kendo.toString(this.value(), 'u')`. Para recuperar a quantidade de fichas, instanciou-se o `Date` com a data escolhida como parâmetro e a guardou em `diaSelecioneado`. Por ter guardado a instância de `Date` em `diaSelecioneado` com o uso do método `getDay`, foi possível retornar ao dia da semana selecionado. O retorno do `getDay` corresponde a um número inteiro referente ao dia da semana: 0 para segunda-feira, 1 terça-feira, e assim por diante. Em seguida, é chamada a função para retornar os agendamentos do profissional selecionado que estão persistidos no banco de dados. Na função, o primeiro atributo é a data escolhida e o segundo é a quantidade de fichas que o profissional atende no dia escolhido.

O terceiro passo foi retornar os agendamentos do profissional selecionado.

No ambiente do usuário, o retorno se dá pelo Código 26 e da secretaria ele se dá pelo Código 27.

Código 26 - Retornar agendamentos ambiente do usuário.

```

01. function retonaragendamento(dataAgenda, qtd_ficha) {
02.     // Limpa table
03.     $("#tbodyDados tr").remove()
04.     $.ajax({
05.         url: "/retonaragendamento?id=" +
06.             $('#txtEsp :selected').val() + "&date=" + dataAgenda,
07.         type: "get",
08.         dataType: "json",
09.         async: true,
10.     }).done(function (callback) {
11.         criarTable(callback.data, callback.userid, callback.date,
12.             qtd_ficha)
13.     })
14. }

```

Fonte: Elaborado pelo autor (2017).

Código 27 - Retornar agendamentos ambiente da secretaria.

```

01. function retonargeralagendamento(dataAgenda, qtd_ficha) {
02.     // Limpa table
03.     $("#tbodyDados tr").remove()
04.     $.ajax({
05.         url: "/retonargeralagendamento?id=" +
06.             $('#txtEsp :selected').val() + "&date=" + dataAgenda,
07.         type: "get",
08.         dataType: "json",
09.         async: true,
10.     }).done(function (callback) {
11.         criarTable(callback.data, callback.date, qtd_ficha)
12.     })
13. }

```

Fonte: Elaborado pelo autor (2017).

Foi feita uma requisição GET para a rota com os parâmetros `id` e `date`. Para as rotas que contêm parâmetros, a montagem e a recuperação dos dados são diferentes, como mostrado nos Códigos 28 e 29. A recuperação foi usando o `req.query` seguido com nome do parâmetro. No Código 28, foi recuperado os dados da sessão.

Código 28 - Rota com parâmetros usuário contido no `routes.js`.

```

01. app.get('/retonaragendamento/:id?:date?', isLoggedIn,
02.     isAuthorized(['3']), function (req, res) {
03.         controller.retonaragendamento(res, req.query.id,
04.             req.query.date, req.session.passport.user)
05.     })

```

Fonte: Elaborado pelo autor (2017).

Código 29 - Rota com parâmetros secretaria contido no routes.js.

```

01. app.get('/retonargeralagendamento/:id?:date?', isLoggedIn,
02.   isAuthorized(['1']), function (req, res) {
03.     controller.retonargeralagendamento(res, req.query.id,
04.     req.query.date)
05.   })

```

Fonte: Elaborado pelo autor (2017).

No controller.js, como mostrado no Código 30 e 31, é chamado o service.js, mostrado no Código 32 e 33, que faz a seleção no banco de dados e depois retorna para controller.js que, por sua vez, retorna para o HTML. No Código 30, foi retornado o id do usuário salvo na sessão.

Código 30 - Retornar agendamento do usuário contido no controller.js.

```

01. retonaragendamento: function (res, id, date, dataSession) {
02.   service.retonaragendamento(id, date, function (error,
03.   status, data) {
04.     res.status(status).json({ data: data,
05.     userid: dataSession.idusuario, date: date })
06.   })
07. }

```

Fonte: Elaborado pelo autor (2017).

Código 31 - Retornar agendamento da secretaria contido no controller.js.

```

01. retonargeralagendamento: function (res, id, date) {
02.   service.retonargeralagendamento(id, date, function (error,
03.   status, data) {
04.     res.status(status).json({ data: data, date: date })
05.   })
06. }

```

Fonte: Elaborado pelo autor (2017).

Código 32 - Retornar agendamento do usuário contido no service.js.

```

01. retonaragendamento: function (id, date, callback) {
02.   let sql = 'SELECT * FROM agendamento WHERE ' +
03.   'profissional_idprofissional = ? && ' +
04.   'data_agendamento = ? ORDER BY numero_ficha'
05.   // Query no Banco de Dados
06.   connection.query(sql, [id, date], function (error, result) {
07.     if (error) {
08.       callback(error, httpStatus.INTERNAL_SERVER_ERROR,
09.       'Desculpe-nos :( Tente novamente.')
10.     } else {
11.       callback(null, httpStatus.OK, result)
12.     }
13.   })
14. }

```

Fonte: Elaborado pelo autor (2017).

Código 33 - Retornar agendamento da secretaria contido no `service.js`.

```



01. retonargeralagendamento: function (id, date, callback) {
02.   let sql = 'SELECT a.usuario_idusuario, ' +
03.     'a.nome_completo_usuario, a.numero_ficha, ' +
04.     'a.necessidades_esp, u.familia, u.microarea, u.cns, ' +
05.     'u.ambulancia FROM agendamento a JOIN usuario u ' +
06.     'ON a.usuario_idusuario = u.idusuario ' +
07.     'WHERE a.profissional_idprofissional = ? && ' +
08.     'a.data_agendamento = ? ORDER BY a.numero_ficha'
09.   // Query no Banco de Dados
10.   connection.query(sql, [id, date], function (error, result) {
11.     if (error) {
12.       callback(error, httpStatus.INTERNAL_SERVER_ERROR,
13.         'Desculpe-nos :( Tente novamente.')
14.     } else {
15.       callback(null, httpStatus.OK, result)
16.     }
17.   })
18. }

```

Fonte: Elaborado pelo autor (2017).

Os dados retornados do Código 30 e 31 foram usados para criar a tabela da ficha de atendimento, como mostrado nas Figuras 16 e 17, que é diferente nos ambientes do usuário e secretaria.

Figura 16 - Tabela de fichas de atendimento do usuário.

Ficha de Atendimento	Necessidades Especiais	Ações
1		<button>Marcada</button>
2		<button>Desmarcar</button>
3		<button>Marcar</button>
4		<button>Marcar</button>
5		<button>Marcar</button>

Fonte: Elaborado pelo autor (2017).

O `id` do usuário retornado no Código 30 foi usado para verificar se o agendamento retornado é do mesmo usuário logado. Se sim, foi colocado o botão desmarcar; se não, foi colocado o botão marcada no ambiente do usuário. Já na da secretaria, diferentemente do usuário, todos os agendamentos podem ser gerenciados.

Figura 17 - Tabela de fichas de atendimento da secretaria.

Ficha de Atendimento	Família	Microárea	CNS	Nome do Paciente	Necessidades Especiais	Necessidade de Ambulância	Ações
1			705006293431451	Maria Sebastiana dos Reis	✓	✓	Desmarcar
2			707100388613220	Gustavo Henrique Martins	✗	✗	Desmarcar
3			708000885272528	Sebastião Raimundo Martins	✗	✗	Desmarcar
4							Marcar

Fonte: Elaborado pelo autor (2017).

Para criar a tabela da ficha de atendimento, como mostrado nas Figuras 16 e 17, foram usados os Códigos 34 e 35.

Código 34 - Criar tabela de agendamento do usuário.

```

01. function criarTable(data, user, date, qtd_ficha) {
02.   $("#tbodyDados tr").remove()
03.   index = 0
04.   for (ficha = 1; ficha <= qtd_ficha; ficha++) {
05.     agendamento = {
06.       ficha: ficha, date: date,
07.       id_profissional: $('#txtEsp :selected').val(),
08.       nome_profissional:
09.         $('#txtEsp :selected').data('nome_completo'),
10.       especialidade: $('#txtEsp :selected').data('especialidade')
11.     }
12.     if (typeof data[index] == 'undefined') {
13.       newTrItem = $("<tr class='success'>" +
14.         "<td>" + ficha + "</td>" +
15.         "<td></td>" +
16.         "<td><button type='button' class='btn btn-success'
17.           onclick='realizaragendamento("+
18.             JSON.stringify(agendamento) +")>Marcar</button></td>" +
19.         "</tr>")
20.       appendTable(newTrItem)
21.     } else {
22.       if (data[index].numero_ficha == ficha) {
23.         newTrItem = $(
24.           (data[index].usuario_idusuario == user ?
25.             "<tr class='danger'>" :
26.             "<tr class='warning'>") +
27.           "<td>" + ficha + "</td>" +
28.           (data[index].necessidades_esp == 1 ?
29.             "<td><span class='glyphicon glyphicon-ok
30.               necessidadeOkay' aria-hidden='true'></span></td>" :
31.             "<td><span class='glyphicon glyphicon-remove
32.               necessidadeNo' aria-hidden='true'></span></td>") +
33.           (data[index].usuario_idusuario == user ?
34.             "<td><button type='button' class='btn btn-danger'
35.               onclick='desmarcaragendamento("+JSON.stringify
36.                 (agendamento)+ ")>Desmarcar</button></td>" :
37.             "<td><button type='button' class='btn btn-warning'
38.               disabled='disabled'>Marcada</button></td>") +
39.           "</tr>")
40.         appendTable(newTrItem)
41.         // Incrementa index
42.         index++
43.       } else {
44.         newTrItem = $("<tr class='success'>" +
45.           "<td>" + ficha + "</td>" +
46.           "<td></td>" +
47.           "<td><button type='button' class='btn btn-success'
48.             onclick='realizaragendamento("+JSON.stringify
49.               (agendamento)+")>Marcar</button></td>" +
50.           "</tr>")
51.         appendTable(newTrItem)
52.       }
53.     }
54.   }
55. }

```

Fonte: Elaborado pelo autor (2017).

Código 35 - Criar tabela de agendamento da secretaria.

```

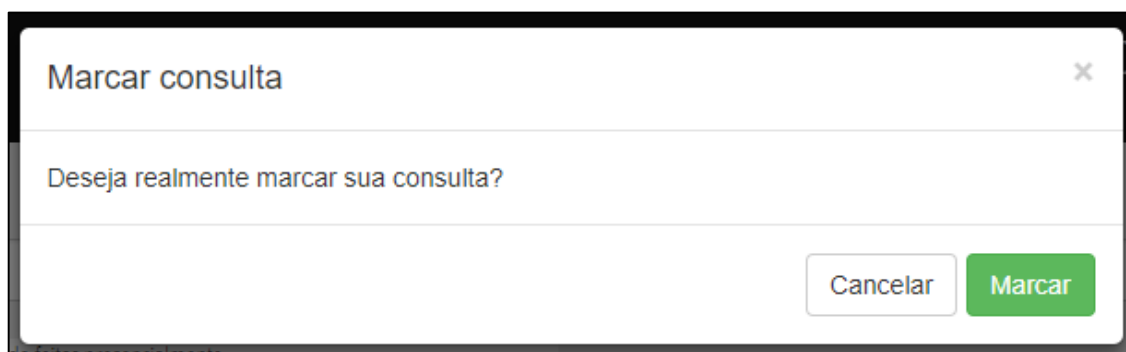
01. function criarTable(data, date, qtd_ficha) {
02.     $("#tbodyDados tr").remove()
03.     index = 0
04.     for (ficha = 1; ficha <= qtd_ficha; ficha++) {
05.         agendamento = {
06.             ficha: ficha, date: date,
07.             id_profissional: $('#txtEsp :selected').val(),
08.             nome_profissional:
09.                 $('#txtEsp :selected').data('nome_completo'),
10.             especialidade: $('#txtEsp :selected').data('especialidade'),
11.             id_usuario: $('#txtPaciente :selected').val()
12.         }
13.         if (typeof data[index] == 'undefined') {
14.             newTrItem = $("<tr class='success'>" +
15.                 "<td>" + ficha + "</td>" +
16.                 "<td></td><td></td><td></td><td></td><td></td><td></td>" +
17.                 "<td><button type='button' class='btn btn-success' " +
18.                     "onclick='realizargeralagendamento(" +
19.                         JSON.stringify(agendamento)+")>Marcar</button></td>" +
20.                 "</tr>")
21.             appendTable(newTrItem)
22.         } else {
23.             if (data[index].numero_ficha == ficha) {
24.                 desmarcarAgendamento = {
25.                     ficha: ficha, date: date,
26.                     id_usuario: data[index].usuario_id_usuario
27.                 }
28.                 newTrItem = $("<tr class='danger'>" +
29.                     "<td>" + ficha + "</td>" +
30.                     "<td>" + data[index].familia + "</td>" +
31.                     "<td>" + data[index].microarea + "</td>" +
32.                     "<td>" + data[index].cns + "</td>" +
33.                     "<td>" + data[index].nome_completo_usuario + "</td>" +
34.                     (data[index].necessidades_esp == 1 ?
35.                         "<td><span class='glyphicon glyphicon-ok " +
36.                             "necessidadeOkay' aria-hidden='true'></span></td>" :
37.                         "<td><span class='glyphicon glyphicon-remove " +
38.                             "necessidadeNo' aria-hidden='true'></span></td>") +
39.                     (data[index].ambulancia == 1 ?
40.                         "<td><span class='glyphicon glyphicon-ok " +
41.                             "necessidadeOkay' aria-hidden='true'></span></td>" :
42.                         "<td><span class='glyphicon glyphicon-remove " +
43.                             "necessidadeNo' aria-hidden='true'></span></td>") +
44.                     "<td><button type='button' class='btn btn-danger' " +
45.                         "onclick='desmarcargeralagendamento(" +
46.                             JSON.stringify(desmarcarAgendamento)+")>Desmarcar</button></td>" +
47.                     "</tr>")
48.                 appendTable(newTrItem)
49.                 index++
50.             } else {
51.                 newTrItem = $("<tr class='success'>" +
52.                     "<td>" + ficha + "</td>" +
53.                     "<td></td><td></td><td></td><td></td><td></td><td></td>" +
54.                     "<td><button type='button' class='btn btn-success' " +
55.                         "onclick='realizargeralagendamento(" +
56.                             JSON.stringify(agendamento)+")>Marcar</button></td>" +
57.                     "</tr>")
58.                 appendTable(newTrItem)
59.             }
60.         }
61.     }
62. }
63. }

```

Fonte: Elaborado pelo autor (2017).

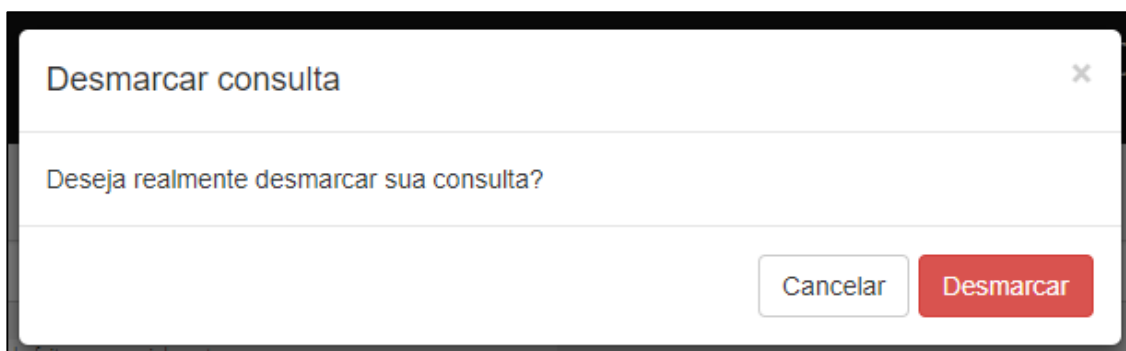
A partir dos passos configurados, foi utilizada a biblioteca Bootbox para criar caixas de diálogos usando os modals do Bootstrap. Nas Figuras 18 e 19, são mostrados os resultados do modal criado com a biblioteca Bootbox.

Figura 18 - Modal de confirmação do agendar consulta.



Fonte: Elaborado pelo autor (2017).

Figura 19 - Modal de confirmação do desmarcar consulta.



Fonte: Elaborado pelo autor (2017).

A forma em que os modals foram criados, primeiramente no ambiente do usuário, são levementes diferentes; portanto, será mostrado somente como foi criado o modals do agendar consulta. As únicas coisas que mudam entre o agendar e desmarcar consulta são:

- o nome da função;
- os atributos `title`, `message`, `label` e `className`;
- a rota da API.

Código 36 - Criando o modal de confirmação do agendar consulta no ambiente do usuário.

```

01. // Agendar consulta
02. function realizaragendamento(agendamento) {
03.     bootbox.confirm({
04.         title: "Marcar consulta",
05.         message: "Deseja realmente marcar sua consulta?",
06.         buttons: {
07.             cancel: {
08.                 label: 'Cancelar',
09.                 className: 'btn-default'
10.             },
11.             confirm: {
12.                 label: 'Marcar',
13.                 className: 'btn-success'
14.             }
15.         },
16.         callback: function (result) {
17.             // Semana
18.             semana = ["mo", "tu", "we", "th", "fr", "sa", "su"]
19.             if (result) {
20.                 $.ajax({
21.                     url: "/realizaragendamento",
22.                     type: "post",
23.                     dataType: "json",
24.                     async: true,
25.                     data: agendamento
26.                 }).done(function (callback) {
27.                     bootbox.alert({
28.                         message: callback.message,
29.                         callback: function () {
30.                             diaSelecioneado = new Date(agendamento.date)
31.                             retonaragendamento(agendamento.date,
32.                                 $('#txtEsp :selected').data('ficha' +
33.                                     semana[diaSelecioneado.getDay()])))
34.                         }
35.                     })
36.                 }).fail(function (callback) {
37.                     jsonCb = JSON.parse(callback.responseText)
38.                     bootbox.alert({
39.                         message: jsonCb.message,
40.                         callback: function () {
41.                             diaSelecioneado = new Date(agendamento.date)
42.                             retonaragendamento(agendamento.date,
43.                                 $('#txtEsp :selected').data('ficha' +
44.                                     semana[diaSelecioneado.getDay()])))
45.                         }
46.                     })
47.                 })
48.             }
49.         })
50.     })
51. }

```

Fonte: Elaborado pelo autor (2017).

Ao clicar em marcar ou desmarcar na tabela de fichas de agendamento, o modal é mostrado na tela. Ao clicar em marcar ou desmarcar, no modal, é feita uma requisição *POST* para a rota `/realizaragendamento` ou `/desmarcaragendamento` com os dados que foram criados nas linhas de 5 a 11 do

Código 34. Os dados recebidos na rota são enviados ao `controller.js` que os envia ao `service.js`. Nos Códigos 37 e 38, mostra-se como foi feito o agendar consulta com o salvamento dos dados no banco de dados e o desmarcar em que os dados são deletados.

Antes de marcar a consulta, retornou-se a informação do usuário ter alguma necessidade especial e/ ou necessitar de ambulância para deslocamento; foi verificado também se o usuário já tem uma consulta marcada no mesmo dia e, se a ficha escolhida está vaga.

Código 37 - Realizar agendamento no ambiente do usuário contido no service.js.

```

01. realizaragendamento: function (data, dataSession, callback) {
02.     async.waterfall([
03.         dbNecessAmbu,
04.         dbCheckIdUser,
05.         dbCheckDate,
06.         dbMark
07.     ], function (error, status, message) {
08.         callback(error, status, message)
09.     })
10.     function dbNecessAmbu(cb) {
11.         let sql = 'SELECT necessidades_esp, ambulancia, ' +
12.             'nome_completo FROM usuario WHERE idusuario = ?'
13.         connection.query(sql,
14.             [dataSession.idusuario],
15.             function (error, result) {
16.                 if (error) {
17.                     cb(error, httpStatus.INTERNAL_SERVER_ERROR,
18.                         'Desculpe-nos :( Tente novamente.')
19.                 } else {
20.                     cb(null, result[0])
21.                 }
22.             })
23.     }
24.     function dbCheckIdUser(dbResult, cb) {
25.         let sql = 'SELECT idagendamento FROM agendamento ' +
26.             'WHERE data_agendamento = ? ' +
27.             '&& usuario_idusuario = ? && profissional_idprofissional = ?'
28.         connection.query(sql,
29.             [data.date, dataSession.idusuario, data.id_profissional],
30.             function (error, result) {
31.                 if (error) {
32.                     cb(error, httpStatus.INTERNAL_SERVER_ERROR,
33.                         'Desculpe-nos :( Tente novamente.')
34.                 } else {
35.                     if (result == null || result.length == 0)
36.                         cb(null, dbResult)
37.                     else
38.                         cb(new Error(), httpStatus.UNAUTHORIZED,
39.                             'Não é possível marcar mais de uma ' +
40.                             'consulta por usuário no dia.')
41.                 }
42.             })
43.     }
44.     function dbCheckDate(dbResult, cb) {
45.         let sql = 'SELECT idagendamento FROM agendamento WHERE ' +
46.             'data_agendamento = ? && numero_ficha = ? && ' +
47.             'profissional_idprofissional = ?'
48.         connection.query(sql,
49.             [data.date, data.ficha, data.id_profissional],
50.             function (error, result) {
51.                 if (error) {
52.                     cb(error, httpStatus.INTERNAL_SERVER_ERROR,
53.                         'Desculpe-nos :( Tente novamente.')
54.                 } else {
55.                     if (result == null || result.length == 0)
56.                         cb(null, dbResult)
57.                     else
58.                         cb(new Error(), httpStatus.CONFLICT,
59.                             'Ficha já está em uso.')
60.                 }
61.             })
62.     }
63.     function dbMark(dbResult, cb) {

```

```

64.         let sql = 'INSERT INTO agendamento ' +
65.             '(profissional_idprofissional, usuario_idusuario, ' +
66.             'profissional_nome_completo, profissional_especialidade, ' +
67.             'nome_completo_usuario, data_agendamento, numero_ficha, ' +
68.             'necessidades_esp, ambulancia) ' +
69.             'VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)'
70.         connection.query(sql,
71.             [data.id_profissional, dataSession.idusuario,
72.             data.nome_profissional, data.especialidade,
73.             dbResult.nome_completo, data.date, data.ficha,
74.             dbResult.necessidades_esp, dbResult.ambulancia],
75.             function (error, result) {
76.                 if (error) {
77.                     cb(error, httpStatus.INTERNAL_SERVER_ERROR,
78.                         'Desculpe-nos :( Tente novamente.')
79.                 } else {
80.                     cb(null, httpStatus.OK,
81.                         'Agendamento realizado com sucesso.')
82.                 }
83.             })
84.     }
85. }

```

Fonte: Elaborado pelo autor (2017).

Código 38 - Desmarcar agendamento no ambiente do usuário contido no `service.js`.

```

01. desmarcaragendamento: function (data, dataSession, callback) {
02.     let sql = 'DELETE FROM agendamento WHERE data_agendamento = ? && ' +
03.         'numero_ficha = ? && usuario_idusuario = ?'
04.     // Query no Banco de Dados
05.     connection.query(sql, [data.date, data.ficha, dataSession.idusuario],
06.         function (error, result) {
07.             if (error) {
08.                 callback(error, httpStatus.INTERNAL_SERVER_ERROR,
09.                     'Desculpe-nos :( Tente novamente.')
10.             } else {
11.                 callback(null, httpStatus.OK,
12.                     'Consulta desmarcada com sucesso.')
13.             }
14.         })
15.     }

```

Fonte: Elaborado pelo autor (2017).

Diferentemente do ambiente do usuário, em que o usuário escolhido está salvo na sessão, no ambiente da secretaria é preciso buscar pelo paciente. Na Figura 20, mostra como é a opção de pesquisar por paciente.

Figura 20 - Pesquisar por paciente.

Procurar pelo nome do paciente

Digite no mínimo três caracteres ... Pesquisar

Selecione o paciente

Nenhum resultado encontrado.

Fonte: Elaborado pelo autor (2017).

Os resultados dos dados pesquisados foram usados para preencher o `select`. O Código 39 mostra como é feita a pesquisa no banco de dados.

Código 39 - Pesquisar por paciente contido no `service.js`.

```

01. retonargeralusuario: function (search, callback) {
02.   let sql = 'SELECT idusuario, familia, microarea, nome_completo, '+
03.     'nome_mae FROM usuario WHERE nome_completo LIKE "%'+search.txtPesquisar+ '%"
04.   // Query no Banco de Dados
05.   connection.query(sql, function (error, result) {
06.     if (error) {
07.       callback(error, httpStatus.INTERNAL_SERVER_ERROR,
08.         'Desculpe-nos :( Tente novamente.')
09.     } else {
10.       if (result == null || result.length == 0)
11.         callback(null, httpStatus.UNAUTHORIZED,
12.           'Nenhum resultado encontrado.')
13.       else
14.         callback(null, httpStatus.OK, result.length +
15.           ' usuário(os) encontrado(os).', result)
16.     }
17.   })
18. }

```

Fonte: Elaborado pelo autor (2017).

Após pesquisar e selecionar o paciente desejado, é possível agendar a consulta. Nas linhas de 3 a 8 do Código 40 é feita a verificação se o usuário foi selecionado e, caso não tenha selecionado, o `modal` da Figura 18 não aparecerá.

O Código 40 apresenta, apenas, o que o difere do Código 36.

Código 40 - Criando o modal de confirmação do agendar consulta no ambiente da secretaria.

```
01. // Agendar consulta
02. function realizargeralagendamento(agendamento) {
03.     if ($('#txtPaciente :selected').val()) {
04.         sendMsg("Escolha o usuário!", 0)
05.         bootbox.alert("Escolha o usuário!")
06.         // Retornar ao topo
07.         $('html, body').animate({ scrollTop: 0 }, 1000)
08.     } else {
09.         bootbox.confirm({
```

Fonte: Elaborado pelo autor (2017).

O Código 40 envia os dados, criados nas linhas de 5 a 12 do Código 35, para o *back-end* e são salvos no banco de dados. Nos Códigos 41 e 42, mostra-se como é realizado o agendar consulta com o salvamento dos dados no banco de dados e o desmarcar em que os dados são deletados.

Código 41 - Realização do agendamento no ambiente da secretaria contido no service.js.

```

01. realizargeralagendamento: function (data, callback) {
02.     async.waterfall([
03.         dbNecessAmbu,
04.         dbCheckIdUser,
05.         dbCheckDate,
06.         dbMark
07.     ], function (error, status, message) {
08.         callback(error, status, message)
09.     })
10.     function dbNecessAmbu(cb) {
11.         let sql = 'SELECT necessidades_esp, ambulancia, nome_completo FROM ' +
12.             'usuario WHERE idusuario = ?'
13.         connection.query(sql, [data.idusuario], function (error, result) {
14.             if (error) {
15.                 cb(error, httpStatus.INTERNAL_SERVER_ERROR,
16.                     'Desculpe-nos :( Tente novamente.')
17.             } else {
18.                 cb(null, result[0])
19.             }
20.         })
21.     }
22.     function dbCheckIdUser(dbResult, cb) {
23.         let sql = 'SELECT idagendamento FROM agendamento WHERE ' +
24.             'data_agendamento = ? && usuario_idusuario = ? && ' +
25.             'profissional_idprofissional = ?'
26.         connection.query(sql,
27.             [data.date, data.idusuario, data.id_profissional],
28.             function (error, result) {
29.                 if (error) {
30.                     cb(error, httpStatus.INTERNAL_SERVER_ERROR,
31.                         'Desculpe-nos :( Tente novamente.')
32.                 } else {
33.                     if (result == null || result.length == 0)
34.                         cb(null, dbResult)
35.                     else
36.                         cb(new Error(), httpStatus.UNAUTHORIZED,
37.                             'Não é possível marcar mais de uma ' +
38.                             'consulta por usuário no dia.')
39.                 }
40.             })
41.     }
42.     function dbCheckDate(dbResult, cb) {
43.         let sql = 'SELECT idagendamento FROM agendamento WHERE ' +
44.             'data_agendamento = ? && numero_ficha = ? && ' +
45.             'profissional_idprofissional = ?'
46.         connection.query(sql,
47.             [data.date, data.ficha, data.id_profissional],
48.             function (error, result) {
49.                 if (error) {
50.                     cb(error, httpStatus.INTERNAL_SERVER_ERROR,
51.                         'Desculpe-nos :( Tente novamente.')
52.                 } else {
53.                     if (result == null || result.length == 0)
54.                         cb(null, dbResult)
55.                     else
56.                         cb(new Error(), httpStatus.CONFLICT,
57.                             'Ficha já está em uso.')
58.                 }
59.             })
60.     }
61.     function dbMark(dbResult, cb) {
62.         let sql = 'INSERT INTO agendamento ' +
63.             '(profissional_idprofissional, usuario_idusuario, ' +
64.             'profissional_nome_completo, profissional_especialidade, '

```



```

65.         'nome_completo_usuario, data_agendamento, numero_ficha, ' +
66.         'necessidades_esp, ambulancia)' +
67.         ' VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)'
68.         connection.query(sql,
69.         [data.id_profissional, data.id_usuario, data.nome_profissional,
70.         data.especialidade, dbResult.nome_completo, data.date,
71.         data.ficha, dbResult.necessidades_esp, dbResult.ambulancia],
72.         function (error, result) {
73.             if (error) {
74.                 cb(error, httpStatus.INTERNAL_SERVER_ERROR,
75.                 'Desculpe-nos :( Tente novamente.')
76.             } else {
77.                 cb(null, httpStatus.OK,
78.                 'Agendamento realizado com sucesso.')
79.             }
80.         })
81.     }
82. }

```

Fonte: Elaborado pelo autor (2017).

Código 42 - Desmarcar agendamento no ambiente da secretaria contido no service.js.

```

01. desmarcargeralagendamento: function (data, callback) {
02.     let sql = 'DELETE FROM agendamento WHERE data_agendamento = ? ' +
03.     '&& numero_ficha = ? && usuario_idusuario = ?'
04.     // Query no Banco de Dados
05.     connection.query(sql, [data.date, data.ficha, data.id_usuario],
06.     function (error, result) {
07.         if (error) {
08.             callback(error, httpStatus.INTERNAL_SERVER_ERROR,
09.             'Desculpe-nos :( Tente novamente.')
10.         } else {
11.             callback(null, httpStatus.OK,
12.             'Consulta desmarcada com sucesso.')
13.         }
14.     })
15. }

```

Fonte: Elaborado pelo autor (2017).

Mediante aos dados salvos no banco de dados foi gerado o relatório com a quantidade de consultas realizadas na UBS. O desenvolvimento do relatório é mostrado no Código 43.

Código 43 - Criação do relatório.

```

01. jQuery(document).ready(function () {
02.     $.ajax({
03.         url: "/retornarsecretariarelatorio",
04.         type: "post",
05.         dataType: "json",
06.         async: true,
07.         data: $("form").serialize()
08.     }).done(function (callback) {
09.         criarRelatorio(callback.dbSeries)
10.     }).fail(function (callback) {
11.         jsonCb = JSON.parse(callback.responseText)
12.         sendMsg(jsonCb.message)
13.     })
14. })
15.
16. function criarRelatorio(dbSeries) {
17.     // Opções do gráfico
18.     var myChartOptions = {
19.         chart: {
20.             renderTo: 'relatorio',
21.             type: 'column'
22.         },
23.         title: {
24.             text: 'Total de agendamentos realizados no ' +
25.                 'ano de ' + $('#txtAno').val()
26.         },
27.         xAxis: {
28.             categories: ['Janeiro', 'Fevereiro', 'Março',
29.                         'Abril', 'Maio', 'Junho', 'Julho', 'Agosto',
30.                         'Setembro', 'Outubro', 'Novembro', 'Dezembro']
31.         },
32.         yAxis: {
33.             min: 0,
34.             title: {
35.                 text: 'Total de agendamentos realizados ' +
36.                     'com cada profissional no ano ' +
37.                     'de ' + $('#txtAno').val()
38.             },
39.             stackLabels: {
40.                 enabled: true,
41.                 style: {
42.                     fontWeight: 'bold',
43.                     color: (Highcharts.theme &&
44.                         Highcharts.theme.textColor) || 'gray'
45.                 }
46.             }
47.         },
48.         legend: {
49.             align: 'center',
50.             x: 0,
51.             verticalAlign: 'top',
52.             y: 25,
53.             floating: false,
54.             backgroundColor: (Highcharts.theme &&
55.                 Highcharts.theme.background2) || 'white',
56.             borderColor: '#CCC',
57.             borderWidth: 1,
58.             shadow: false
59.         },
60.         tooltip: {
61.             headerFormat: '<b>{point.x}</b><br/>',
62.             pointFormat: '{series.name}: {point.y}<br/>Total: {point.stackTotal}'
63.         },
64.         plotOptions: {

```

```

65.         column: {
66.             stacking: 'normal',
67.             dataLabels: {
68.                 enabled: true,
69.                 color: (Highcharts.theme &&
70.                     Highcharts.theme.dataLabelsColor) || 'white'
71.             }
72.         },
73.     },
74.     series: []
75. }
76.
77. // Criar a series do gráfico
78. for (let i = 0; i < dbSeries.length; i++) {
79.     // Variável que identifica se algum Dr. foi encontrado na series
80.     var encontrado = 0
81.     // Verificar se já tem o Dr. na series
82.     for (let j = 0; j < myChartOptions.series.length; j++) {
83.         // Se já tem, modifica o data []
84.         if (myChartOptions.series[j].name ==
85.             (dbSeries[i].profissional_nome_completo + ' - ' +
86.              dbSeries[i].profissional_especialidade)) {
87.             myChartOptions.series[j].data[dbSeries[i].mes - 1] = dbSeries[i].qtd_agen
88.             // Modifica a variável e break
89.             encontrado = 1
90.             break
91.         }
92.     }
93.     // Se não tem, criar uma nova series
94.     if (encontrado == 0) {
95.         var createSeries = {
96.             data: []
97.         }
98.         createSeries.name = dbSeries[i].profissional_nome_completo +
99.             ' - ' + dbSeries[i].profissional_especialidade
100.         for (let j = 0; j < 12; j++) {
101.             if (dbSeries[i].mes == (j + 1))
102.                 createSeries.data.push(dbSeries[i].qtd_agen)
103.             else createSeries.data.push("")
104.         }
105.         myChartOptions.series.push(createSeries)
106.     }
107. }
108.
109. // Criar o gráfico
110. var myChart = new Highcharts.Chart(myChartOptions)
111. }

```

Fonte: Elaborado pelo autor (2017).

Código 44 - Criação do relatório no ambiente da secretaria contido no service.js.

```

01. retornarsecretariarelatorio: function (data, callback) {
02.     let sql = 'SELECT profissional_nome_completo, profissional_especialidade, ' +
03.         'EXTRACT(MONTH FROM data_agendamento) AS mes, COUNT(*) as qtd_agen ' +
04.         'FROM agendamento WHERE EXTRACT(YEAR FROM data_agendamento) = ? ' +
05.         'GROUP BY profissional_nome_completo, profissional_especialidade, ' +
06.         'EXTRACT(MONTH FROM data_agendamento) ORDER BY EXTRACT(MONTH FROM data_agendamento)'
07.     // Query no Banco de Dados
08.     connection.query(sql, [data.txtAno], function (error, result) {
09.         if (error) {
10.             callback(error, httpStatus.INTERNAL_SERVER_ERROR, 'Desculpe-nos :( Tente novamente.')
11.         } else {
12.             if (result == null || result.length == 0)
13.                 callback(new Error(), httpStatus.UNAUTHORIZED, 'Nenhum resultado encontrado.')
14.             else
15.                 callback(null, httpStatus.OK, 'Resultados encontrados.', result)
16.         }
17.     })
18. }

```

Fonte: Elaborado pelo autor (2017).

O resultado do gráfico é mostrado na Figura 21.

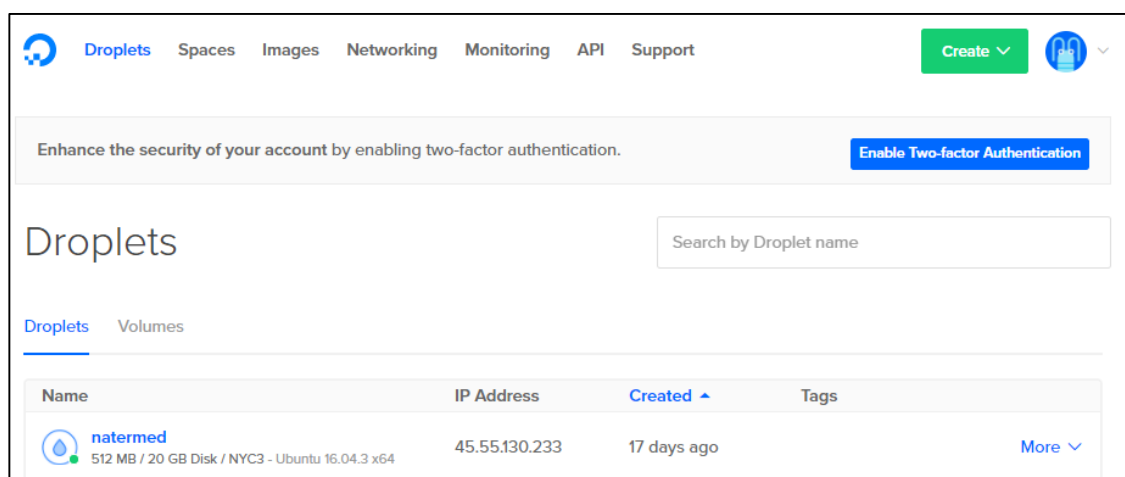
Figura 21 - Relatório de agendamentos.



Fonte: Elaborado pelo autor (2017).

A aplicação foi hospedada no Digital Ocean, por ele oferecer uma API simples, bom desempenho, implantação simples e em segundos e também um ótimo custo. Na Figura 22 é mostrada o ambiente da hospedagem.

Figura 22 - Hospedagem na Digital Ocean.



Fonte: Elaborado pelo autor (2017).

E por fim, no domínio foi utilizado a ferramenta *online . tk*, por ser gratuito. Sua obtenção foi no site da dot.tk. Para configurar o domínio, primeiramente, foi preciso apontar os servidores de nome da dot.tk para os servidores da Digital Ocean e nela configurar o domínio.

4. DISCUSSÃO DOS RESULTADOS

Neste capítulo são discutidos os resultados obtidos no desenvolvimento da aplicação, cujo objetivo geral é o desenvolvimento de uma aplicação para agilizar e facilitar o processo de marcação de consultas na cidade de Natércia-MG, visto que isso era uma necessidade do município.

De início, realizou-se entrevistas com os colaboradores da UBS, e, com isso, foi possível reunir os recursos necessários para começar o desenvolvimento da aplicação. Desse modo, iniciou-se o estudo das tecnologias e implementação das funcionalidades essenciais para esse trabalho. Foram escolhidas as linguagens relacionadas a *WEB* utilizadas no desenvolvimento de *sites*, uma vez que, podem ser acessados de diferentes dispositivos, facilitando o acesso dos usuários na aplicação.

Dentre as linguagens escolhidas, aplicou-se o HTML para estruturar a aplicação, uma vez que ele permite criar *tags* de diferentes tipos de elementos, como: títulos, parágrafos, quebras de linhas, tabelas, imagens, *links*, entre outros.

Segundo Dias (2003), a usabilidade pode ser considerada uma qualidade de uso, isto é, qualidade de interação entre usuários e sistema, que depende das características tanto do sistema quanto do usuário.

Para tornar a aplicação mais agradável, com boa aparência e usabilidade, utilizou-se o CSS. A partir dele, adicionaram-se formatações e estilos na estrutura HTML que proporcionaram visibilidade e o *design* da aplicação. Seu uso foi limitado por requerer muito tempo de digitação e entendimento da linguagem.

Para agilizar o processo de formatação do HTML, utilizou-se a ferramenta Bootstrap, por ela possuir uma diversidade de formatação já criadas pelos seus desenvolvedores e por contar com a facilidade de tornar a página responsiva com o uso do sistema de *grids* (grades que se dimensionam adequadamente à medida que o tamanho do *display* dos dispositivos aumenta/diminui), auxiliando a visibilidade em diferentes aparelhos.

O JavaScript foi utilizado no *front-end* e *back-end*²⁷; seu uso no *front-end*, em grande parte, se deu com jQuery (biblioteca que se concentra em simplificar o uso do

²⁷ Responsável por processar a entrada de dados

JavaScript) que possui as mesmas funções e métodos, bem como o AJAX cuja característica é a vantagem de efetuar requisições e resposta sem executar o *reload*²⁸ da página. Optou-se por sua utilização, uma vez que se ganha tempo escrevendo menos e obtendo resultados mais rápidos.

Utilizou-se no *back-end* o Node.js que utiliza apenas um único *thread* de execução para tratar requisições, ao invés de criar um *thread* específico para cada conexão. Isso trouxe como principal vantagem possibilitar que o servidor trate de milhares de conexões, sem que sejam necessários recursos computacionais exagerados. A plataforma implementa as suas funcionalidades e recursos através de módulos, instalados com a ferramenta chamada Node Package Manager (NPM), através de simples linhas de comando.

Para o desenvolvimento da API (*Application Programming Interface*) utilizou-se o Node.js com o *framework* Express.js por ser flexível e conter um conjunto robusto de recursos.

Verificou-se a necessidade de guardar os dados que estavam chegando do HTML; então fez-se a conexão com o banco de dados MySQL em que os dados enviados pelo AJAX são salvos. Segundo Alves (2009), o MySQL é um sistema de banco de dados relacional em que os dados são organizados em tabelas (relações) formadas por linhas e colunas, nas quais se relacionam as informações referentes a um mesmo assunto de modo organizado.

Com os dados persistidos foi utilizado o Highcharts para gerar relatórios da aplicação por meio de gráficos. Como resultado, a secretaria tem maior facilidade de identificar quantos agendamentos ocorreram com cada médico. Como foi mostrado na Figura 21 do quadro metodológico.

A *site* conta com os benefícios das linguagens já citadas anteriormente, como também, com as vantagens que serão alcançadas através da aplicação, a fim de facilitar o acesso e, conseqüentemente, atrair novos usuários.

Alcançaram-se os objetivos de cadastrar o acesso dos usuários, no qual é possível registrar o cartão de acesso (numérico) e escolher uma senha e, a partir disso,

²⁸ Recarregar página

realizar um *login* seguro no ambiente do usuário. Para a realização do *login*, utilizou-se o módulo Passport que é um *middleware* de autenticação para Node.js.

No ambiente do usuário é possível criar um cadastro domiciliar, ou seja, permite-se criar uma conta de usuário para cada integrante da família por registro de acesso, possibilitando assim, concentrar as informações e interligá-las.

No cadastro do usuário, teve-se o cuidado de recuperar os dados importantes, segundo a pesquisa feita na UBS, com cada um deles. Na página foi utilizado o Bootstrap, a fim de organizar os campos em que o usuário vai adicionar suas informações. Na Figura 23 é mostrada a página de cadastro do usuário.

Figura 23 - Página de cadastro do usuário.

Dados Pessoais

* Nome Completo * Nome da Mãe Nome do Pai * Naturalidade

* Sexo * Tipo Sanguíneo * Escolaridade * Situação * Estado Civil

* Data de Nascimento * CPF ☐ Não Declarado * RG ☐ Não Declarado * Cartão SUS ☐ Não Declarado

Contatos

Email Telefone Celular

Endereço

* Estado * Cidade Rua Bairro Número

Adicionais

* Tem alguma necessidade especial? ☒ Sim ☐ Não * Precisa de ambulância para o deslocamento? ☒ Sim ☐ Não

Fonte: Elaborado pelo autor (2017).

Após o cadastro e a escolha do usuário, pode-se visualizar o horário dos profissionais (Figura 24), fazer o agendamento de consulta, gerar o relatório de quantas consultas foram agendadas, consultar a lista de medicamentos disponíveis e indisponíveis na UBS e informar-se através das notícias divulgadas pela UBS.

Figura 24 - Página de horários.

Profissionais	Especialidades	Segunda	Terça	Quarta	Quinta	Sexta	Ações
Dr. Lidinei	Clínico Geral	Início às 07:00H, 20 Fichas	-	-	Início às 07:00H, 20 Fichas	-	Editar Excluir
Dr. Renan	Clínico Geral	-	Início às 08:00H, 20 Fichas	-	-	Início às 08:00H, 20 Fichas	Editar Excluir
Dr. Alexandre	Clínico Geral	-	-	Início às 08:00H, 20 Fichas	-	-	Editar Excluir
Dr. Lidinei	Ginecologista	Início às 07:00H, 20 Fichas	-	-	Início às 07:00H, 20 Fichas	-	Editar Excluir
Dra. Juliana	Nutricionista	-	Início às 12:00H, 25 Fichas	-	-	-	Editar Excluir
Dr. Marco Antônio	Pediatra	Início às 12:00H, 20 Fichas	Início às 07:00H, 20 Fichas	Início às 07:00H, 20 Fichas	Início às 09:00H, 20 Fichas	-	Editar Excluir
Dra. Janiely	Psicólogo	-	-	Início às 07:30H, 10 Fichas	-	Início às 07:30H, 10 Fichas	Editar Excluir
Dra. Bruna	Psiquiatra	-	Início às 12:30H, 10 Fichas	-	Início às 12:30H, 10 Fichas	-	Editar Excluir

Fonte: Elaborado pelo autor (2017).

Para criar a página de horários, foram utilizadas as informações da Tabela 1. Ela também mostra o horário dos dentistas, mas como eles utilizam agenda fixa e o agendamento é feito pela secretária dos profissionais e não utiliza de fichas de atendimento, o emprego da aplicação para agendamento de suas consultas ficou impossibilitado. Para incluir esse profissional na aplicação, seria necessário a criação de ambiente exclusivo no qual o agendamento seria guardado em uma fila de espera gerenciada pela secretária do profissional.

A página de notícias foi desenvolvida para que somente a secretaria possa alterá-la. Para criar uma notícia basta preencher os campos: título, início, término e o texto e clicar em cadastrar. Com isso, ela aparecerá na primeira página quando o usuário acessar informando-o sobre os eventos da UBS, como mostra na Figura 25.

Figura 25 - Página de notícias.



Fonte: Elaborado pelo autor (2017).

Assim como a página de notícias, a de medicamentos também foi desenvolvida para que somente a secretaria possa alterá-la. Ela contém a lista de medicamentos disponíveis e indisponíveis, permitindo que o usuário não precise se deslocar até a Unidade para receber essa informação. A página dos medicamentos é mostrada na Figura 26.

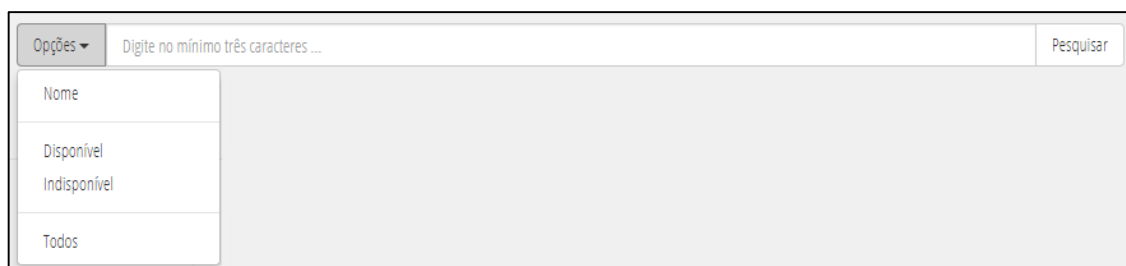
Figura 26 - Página de medicamentos.

Nome	Estoque	Ações
AAS 100mg	Indisponível	Editar Excluir
Dipirona 500mg	Disponível	Editar Excluir
Insulina NPH 100UL/ml	Disponível	Editar Excluir
Paracetamol 500mg	Disponível	Editar Excluir
Puran T4 50mg	Indisponível	Editar Excluir

Fonte: Elaborado pelo autor (2017).

Como são distribuídos diversos medicamentos, para facilitar a pesquisa colocou-se filtros com as opções mostradas na Figura 27.

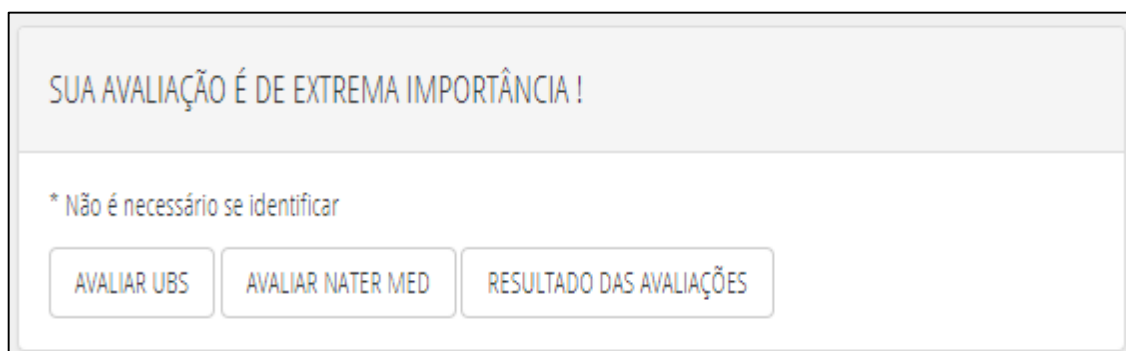
Figura 27 - Campo de pesquisa de medicamentos.

A interface de pesquisa de medicamentos apresenta um campo de entrada com o placeholder "Digite no mínimo três caracteres ...". À esquerda do campo, há um menu suspenso rotulado "Opções" com as opções "Nome", "Disponível", "Indisponível" e "Todos". À direita do campo, há um botão "Pesquisar".

Fonte: Elaborado pelo autor (2017).

A aplicação também conta com uma ala de avaliação, que é diferente nos ambientes do usuário e da secretaria. No primeiro ambiente há duas avaliações: uma para UBS e a outra para a aplicação. Já no segundo ambiente a avaliação é somente para a aplicação. Como mostrado nas Figura 28.

Figura 28 - Aba de avaliação.

A interface de avaliação exibe o título "SUA AVALIAÇÃO É DE EXTREMA IMPORTÂNCIA !". Abaixo, há uma mensagem: "* Não é necessário se identificar". Na base da interface, há três botões: "AVALIAR UBS", "AVALIAR NATER MED" e "RESULTADO DAS AVALIAÇÕES".

Fonte: Elaborado pelo autor (2017).

Como critério de avaliação, foram escolhidas quatros perguntas essenciais para o bom funcionamento da UBS e da aplicação. Como mostrado na Figura 29.

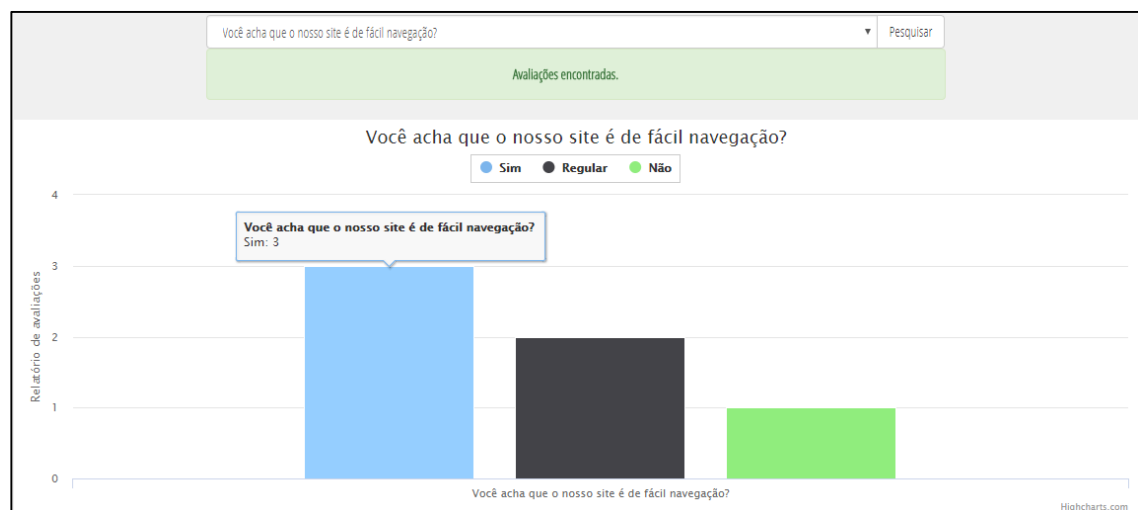
Figura 29 - Requisitos para avaliação.

1. Limpeza e organização do local. <input type="radio"/> Excelente <input type="radio"/> Bom <input type="radio"/> Fraco	1. Você acha que o nosso site é de fácil navegação? <input type="radio"/> Sim <input type="radio"/> Regular <input type="radio"/> Não
2. Atendimento da recepção. <input type="radio"/> Excelente <input type="radio"/> Bom <input type="radio"/> Fraco	2. O conteúdo do nosso website é claro e compreensível? <input type="radio"/> Sim <input type="radio"/> Um pouco <input type="radio"/> Não
3. O tempo de espera para atendimento. <input type="radio"/> Excelente <input type="radio"/> Bom <input type="radio"/> Fraco	3. Em geral, você ficou satisfeito com o nosso site? <input type="radio"/> Satisfeito <input type="radio"/> Nem satisfeito, nem insatisfeito <input type="radio"/> Insatisfeito
4. Qualidade do atendimento prestado. <input type="radio"/> Excelente <input type="radio"/> Bom <input type="radio"/> Fraco	4. Qual é a probabilidade de você recomendar o nosso site para outros? <input type="radio"/> Alta <input type="radio"/> Nem alta, nem baixa <input type="radio"/> Baixa

Fonte: Elaborado pelo autor (2017).

Após as avaliações pelos usuários, é possível ver os resultados como mostra na Figura 30.

Figura 30 - Resultados da avaliação.



Fonte: Elaborado pelo autor (2017).

Com a aplicação concluída, foi necessária sua hospedagem no Digital Ocean que oferece uma API simples, bom desempenho, implantação simples e rápida com um ótimo custo. No domínio foi utilizado a ferramenta *WEB tk* por ser gratuito.

Por fim, caso a aplicação WEB for implantada na UBS, cumpre seu propósito de solucionar um problema social que afeta muitos usuários da rede pública de saúde, de forma prática e ágil.

5. CONCLUSÃO

Desde o princípio, vários temas foram cogitados para o desenvolvimento do trabalho de conclusão de curso. Sendo assim, foi decidido desenvolver uma Aplicação *WEB* com o objetivo de otimizar e gerenciar o serviço de agendamento *on-line* de consultas da Unidade Básica de Saúde (UBS) do município de Natércia -MG. Diante da dificuldade da marcação de consultas se dar somente de forma presencial, criou-se o Nater Med Agendamento Online.

Para o desenvolvimento da aplicação, foi necessário obter informações sobre o funcionamento da gestão administrativa da UBS com relação a agendamento de consultas, entrega de fichas de atendimento, distribuição de medicamentos, horário dos profissionais, dentre outros. Para a obtenção dessas informações, de extrema importância, realizou-se entrevistas e pesquisas com os colaboradores e responsáveis da área da saúde do município de Natércia-MG.

Para possibilitar o acesso e abranger a maior parte dos usuários optou-se pelo uso de uma aplicação *WEB*. Essa escolha se deu, também, por não ter a necessidade de instalação da aplicação para que a utilizem.

As linguagens utilizadas no desenvolvimento da aplicação foram abordadas no curso de Sistemas de Informação da Univás pelo Professor Ednardo David Segura como também outras linguagens não presentes na grade do curso, adicionadas conforme as necessidades encontradas.

Todas as funcionalidades e objetivos que foram idealizados no início e no decorrer da aplicação foram alcançados. Exceto, a possibilidade de criar um prontuário eletrônico no qual o paciente e o profissional teriam a facilidade de visualizar facilmente as informações do prontuário *on-line* de forma legível. A partir de uma página implementada pelo desenvolvedor, o profissional poderia informar sobre a anamnese (sintomas narrados pelo paciente, a queixa principal da procura do paciente pelo atendimento médico), a Classificação Internacional de Doenças e Problemas Relacionados à Saúde (CID) (catálogo que contém uma codificação padrão para as doenças), opções de tratamento, e as opções da triagem como pressão arterial, temperatura, peso entre outros, quando necessário.

Com os dados seria possível gerar dois relatórios de suma importância como a incidência de determinado problema de saúde, facilitada pela identificação de doenças do CID e os medicamentos mais receitados e menos receitados. Porém, a funcionalidade do prontuário eletrônico foi retirada, por serem necessárias adoções de padrões de comunicação, leis e regras para que fosse regulamentado o processo de utilizá-lo.

Outras funcionalidades foram pensadas, mas serão implementadas futuramente, como criar um ambiente separado para o dentista liberar as fichas para agendamento em dias específicos do mês ou a possibilidade de o usuário solicitar um agendamento com qualquer profissional cadastrado no site no qual a solicitação seria guardada em um fila gerenciada pela secretaria. Para as solicitações que forem efetivamente agendadas, seriam enviadas SMSs de confirmação para o paciente, cujo envio foi estudado a partir do uso da plataforma do Twilio, que é uma API para troca de mensagens de texto, imagens e chamadas.

Sendo assim, concluiu-se que os resultados obtidos com a utilização das tecnologias foram totalmente satisfatórios, todas as expectativas com o desenvolvimento do trabalho de conclusão de curso e da aplicação foram alcançadas e, por fim, foi possível criar a aplicação Nater Med Agendamento Online.

REFERÊNCIAS

ALVES, William Pereira. **Banco de dados: teoria e desenvolvimento**. São Paulo: Érica, 2009.

APPOLINÁRIO, F. **Dicionário de metodologia científica: um guia para a produção do conhecimento científico**. São Paulo: Atlas, 2004.

BARROS, A. J. S. e LEHFELD, N. A. S. **Fundamentos de Metodologia: Um Guia para a Iniciação Científica**. 2 Ed. São Paulo: Makron Books, 2000.

BOOTSTRAP. **Framework para desenvolvimento de aplicações Web**. Disponível em: <<http://getbootstrap.com/>>. Acesso em: 10 mar. 2017.

CHIORO, A.; SCAFF, A. **Saúde e cidadania: a implantação do Sistema Único de Saúde**. Disponível em: <<http://www.consaude.com.br/sus.htm>>. Acesso em: 12 mar. 2017.

CSS. **Cascading Style Sheet**. Disponível em: <<http://www.infoescola.com/informatica/cascading-style-sheets-css/>>. Acesso em: 06 mai. 2017.

DIAS, Cláudia. **Usabilidade na Web**. Rio de Janeiro: Alta Books, 2003.

GIL, A.C. **Métodos e técnicas de pesquisa social**. 6.ed. [S.1]: Atlas, 2007.

HIGHCHARTS. Disponível em: <<https://www.highcharts.com/products/highcharts/>>. Acesso em: 18 out. 2017.

JQUERY. **Biblioteca de funções e métodos JavaScript**. Disponível em: <<https://jquery.com/>>. Acesso em: 13 mar. 2017.

LEISMANN, Roberta Raquel. **Sistema de agendamento do atendimento médico na unidade básica de saúde de Arroio Trinta**. Disponível em: <<http://campeche.inf.furb.br/tccs/2008-II/TCC2008-2-15-VF-RobertaRLeismann.pdf>>. Acesso em: 18 mar. 2017.

MANZINI, E. J. **A entrevista na pesquisa social**. Didática, São Paulo, 1990/1991.

MAZZA, Lucas. **HTML5 e CSS3**. Domine a web do futuro. 1ª ed. [S.1]: Casa do Código, 2014.

MDN. **Linguagens utilizadas na aplicação Web**. Disponível em: <<https://developer.mozilla.org/pt-BR/>>. Acesso em: 06 fev. 2017.

MILANI, André. **MySQL**, Guia do programador. 1 ed. Novatec, Janeiro de 2007.

NODE.JS. **Linguagem *back-end* para desenvolvimento de aplicações Web**. Disponível em <<http://nodejs.org>>. Acesso em: 13 mar. 2017.

OLIVEIRA, S. C. de. Tecnologia e a medicina. Revista Hospitais Brasil, São Paulo, nov. 2007. Disponível em: <http://www.revistahospitaisbrasil.com.br/Default.asp?Artigos_ListPage=2&Art_Chv=95&Rev_Pag=Artigos%5FShow>. Acesso em: 23 ago. 2017.

POWERS, Shelley. **JavaScript**. Aprendendo JavaScript. 2 ed. Novatec, Dezembro de 2008.

SILVA, Maurício Samy. **Bootstrap 3.3.5**. Aprenda a usar o framework Bootstrap para criar layouts CSS complexos e responsivos. 1ª ed. Novatec, Outubro de 2015.

TCC CONSULTAS. **Agendamento de consultas**. Disponível em: <https://projetos.inf.ufsc.br/arquivos_projetos/projeto_291/monografiaComAnexo.pdf>. Acesso em: 10 fev. 2017.

TCC SAÚDE. **Problemas na Saúde**. Disponível em:

<<http://www.lume.ufrgs.br/bitstream/handle/10183/1862/000310803.pdf> >. Acesso em: 11 fev. 2017.

TILKOV, Stefan; VINOSKI, Steve. **Node.js: Using JavaScript to Build High-Performance Network Programs**. IEEE Internet Computing, 2010.