UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
UNIDADE ACADÊMICA DE SISTEMAS E COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

AGNALDO SOUTO XAVIER JUNIOR

AUTOMAÇÃO PARA O PROJETO LASERTERAPIA DA UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

CAMPINA GRANDE - PB

2020

AGNALDO SOUTO XAVIER JUNIOR

AUTOMAÇÃO PARA O PROJETO LASERTERAPIA DA UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Tiago Lima Massoni.

CAMPINA GRANDE - PB

2020

X3a Xavier Junior, Agnaldo Souto.
Automação para o projeto Laserterapia da Universidade Federal de Campina Grande. / Agnaldo Souto Xavier Junior. – 2020.

9 f.

Orientador: Prof. Dr. Tiago Lima Massoni.
Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. Coleta de dados. 2. Análise de dados. 3. Dados – coleta e análise. 4. Projeto Laserterapia - UFCG. 5. Tecnologia aplicada à saúde. 6. Automação em coleta de dados. I. Massoni, Tiago Lima. II. Título.

CDU:004.6(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

AGNALDO SOUTO XAVIER JUNIOR

AUTOMAÇÃO PARA O PROJETO LASERTERAPIA DA UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Tiago Lima Massoni
Orientador – UASC/CEEI/UFCG

Professor Dr. Adalberto Cajueiro de Farias
Examinador – UASC/CEEI/UFCG

Trabalho aprovado em: 2020.

CAMPINA GRANDE - PB

Automação para o projeto Laserterapia da Universidade Federal de Campina Grande

Agnaldo Souto Xavier Junior
Universidade Federal de Campina Grande
agnaldo.junior@ccc.ufcg.edu.br

Tiago Lima Massoni
Universidade Federal de Campina Grande
massoni@dsc.ufcg.edu.br

RESUMO

Coleta e análise de dados de forma fácil e simples está deixando de ser uma opção e se tornando uma necessidade para quem trabalha com isso. Atualmente, é notável a dificuldade de quem trabalha anotando coisas em cadernos ou planilhas para poder organizar ou visualizar esses dados depois. É esse o caso do projeto Laserterapia e Ações Educativas Transdisciplinares na Oncologia Pediátrica, que atualmente trabalha diariamente com coleta de dados dos pacientes, porém encontrando bastante dificuldade no processo. Tendo em vista essa necessidade, apresento neste trabalho um sistema web que permite uma fácil coleta e um armazenamento desses dados coletados, facilitando bastante o trabalho da equipe em questão e evitando perda de dados. Para avaliar a qualidade do sistema implementado para eles, 10 colaboradores do projeto responderam um formulário que consiste de 9 questões. Como resultado, o sistema apresentou um grau de satisfação de 4.9 de 5, sendo considerado pelo professor orientador do projeto e pelos colaboradores uma importante ferramenta que irá auxiliá-los no dia-a-dia.

PALAVRAS-CHAVE

Coleta, dados, aplicação.

Links Úteis:

https://github.com/AgnaldoCC/laserterapia_frontend
https://github.com/AgnaldoCC/laserterapia_backend
https://forms.gle/29iSTdqTvs26uCnn6

1. INTRODUÇÃO

Atualmente, é difícil conviver sem a tecnologia. Para quem trabalha com coleta de dados por exemplo, é inegável que fica difícil trabalhar sem o auxílio de uma ferramenta, tornando cada vez mais crucial que exista um meio de automatizar essa coleta, tornando mais fácil de consultar ou analisar esses dados futuramente.

O projeto Laserterapia da Universidade Federal de Campina Grande[1] encontrou essa exata dificuldade para coletar e persistir os dados dos pacientes, com o passar de 6 anos, já que a equipe precisa constantemente estar lidando com dados, tanto socioeconômicos quanto clínicos, de vários pacientes, especialmente após as aplicações do laser. Essa dificuldade para coletar e gerenciar os dados vem impedindo a equipe, formada atualmente por 29 integrantes, de fazerem contribuições significativas na área (que é tão pouco abordada no meio acadêmico) através de publicações em revistas, artigos científicos, entre outros.

O sistema desenvolvido busca ajudar esse Projeto, fornecendo uma ferramenta para auxiliar a coleta de dados clínicos e socioeconômicos dos pacientes, permitindo o cadastro de pacientes e aplicações. A aplicação utiliza a metodologia Progressive Web App (PWA) [2]. Este novo modelo de aplicação combina recursos oferecidos pelos mais modernos navegadores, com as vantagens de uso de um aparelho mobile, fazendo com que a aplicação possa ser acessada por um navegador no computador ou no celular.

A avaliação da aplicação foi feita através de um formulário, que consistiu de 9 questões, abordando a corretude das funcionalidades já existentes e sobre a satisfação geral do que já está implementado. O questionário foi respondido por 10 colaboradores do projeto, e apresentou resultado positivo sobre o quanto esse aplicativo vai ajudar no dia-a-dia do projeto, com uma média de 4.9 de 5 em grau de satisfação.

2. SOLUÇÃO

Este trabalho tem como principal objetivo criar uma ferramenta capaz de fazer uma coleta de dados, socioeconômicos e clínicos, referentes ao paciente e as aplicações feitas nele, eficiente e sem perda, para que seja feita a automação da coleta e análise de dados para o projeto de Laserterapia da Universidade Federal de Campina Grande. Ao final, foi entregue a equipe responsável pelo projeto uma aplicação web, com a finalidade de ajudá-los a ter uma coleta de dados mais eficiente, assim

otimizando o trabalho da equipe durante a aplicação do laser. Além disso, com esses dados já guardados em uma base, será possível fazer análises em cima desses dados, permitindo estudos mais complexos sobre os pacientes e sobre a Laserterapia em si.

2.1 - Funcionalidades

Logo de início, o usuário irá poderá fazer se cadastrar no sistema, informando *nome, email, curso, senha* e uma foto de perfil optativa. Porém, por se tratar de uma aplicação que irá conter dados sensíveis, a aplicação não permite qualquer pessoa fazer um cadastro. Primeiro, é necessário que um usuário administrador(já cadastrado no banco de dados) permita o cadastro de um certo usuário, informando seu email em uma página própria para isso. Após isso, o usuário com o email informado pelo administrador poderá fazer o cadastro e se logar na aplicação. Uma vez logado, o usuário poderá fazer o cadastro de um paciente, e por sua vez cada paciente contém dados socioeconômicos, como também uma lista de aplicações(que nada mais são do que as aplicações do laser feito naquele paciente). As aplicações são apresentadas dentro da tela de cada paciente ordenadas em ordem cronológica. Dentro da tela de cada aplicação, o usuário pode ver o local onde foi aplicado o laser, se houve melhora em relação a última aplicação e qual o colaborador do projeto que fez a última aplicação naquele paciente.

2.2 - Arquitetura

A arquitetura web utilizada foi a padrão para desenvolvimento de aplicações, que envolve um servidor e um cliente[2]. Essa abordagem é muito usada atualmente, pois diminui a complexidade da aplicação que irá rodar no navegador/dispositivo do cliente, se preocupando apenas em fazer requisições e consumir dados vindos do servidor, como pode-se observar na *Figura 1*.

O cliente seria o frontend, responsável por exibir as informações vindas do backend de uma forma visualmente agradável, e consumir os serviços do servidor.

O servidor seria o backend, responsável por toda a lógica por trás do sistema, como conexões com banco de dados, além de processar e dar um retorno para as chamadas feitas pelo cliente.

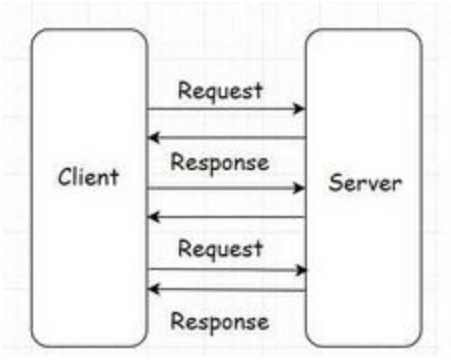

Figura 1: Modelo Cliente-Servidor exemplificado durante o ciclo de vida de uma aplicação. [3]

2.2.1 - Tecnologias utilizadas no servidor (Backend)

O servidor foi feito com as tecnologias NodeJS [4], junto com Express. Essas tecnologias foram escolhidas por diversos motivos, como por exemplo o fato de ser toda em Javascript, que acaba tendo uma boa sincronia com o frontend, que também é feito em javascript. Outros motivos são a fácil configuração e fácil manutenibilidade, sendo extremamente simples adicionar uma nova rota no futuro, caso precise. Para a persistência dos dados, foi utilizado MongoDB [5], um banco de dados orientado a documentos que tem uma ótima sincronia com o NodeJS, pois: fica muito fácil o armazenamento e recuperação de documentos não relacionais, podendo ser feito com poucas linhas de código; Tanto o NodeJs quanto o MongoDB trabalham nativamente com a mesma estrutura de dados: JSON por parte do Node, e Documentos(bastante similares ao JSON) por parte do Mongo[6]; MongoDB é bem mais simples de fazer alterações do que banco de dados relacionais. A desvantagem desta alternativa se dá ao fato de que é consideravelmente mais rápido recuperar dados em banco de dados relacionais.

2.2.2 - Estrutura do backend

A estrutura no backend foi feita baseada nos padrões já existentes [7], sendo dividido em 4 diretórios principais, apresentados na *Figura 2*: app, config, database e modules. A pasta app é a que contém todo o código da aplicação, sendo subdividida em controllers, middlewares e models. Os controllers são onde estão as rotas da aplicação, que fazem consultas no banco de dados e são consumidas pelo frontend para apresentar essas informações para o usuário. Os middlewares são trechos de código que são executados depois da chamada do cliente e antes dessa chamada chegar ao servidor. Na aplicação, foram utilizados 2 middlewares: um para checar se o usuário é o administrador do sistema e outro para chegar se o token enviado pelo usuário está correto. A pasta models são onde estão os modelos da aplicação, responsáveis por definir a estrutura dos objetos que serão salvos no Mongo. No projeto, existem 4 models: application, authorized, patient e user. Cada um desses modelos possuem atributos que podem ou não ser obrigatórios na criação de uma nova instância, como por exemplo o modelo user, apresentado na *Figura 3*.

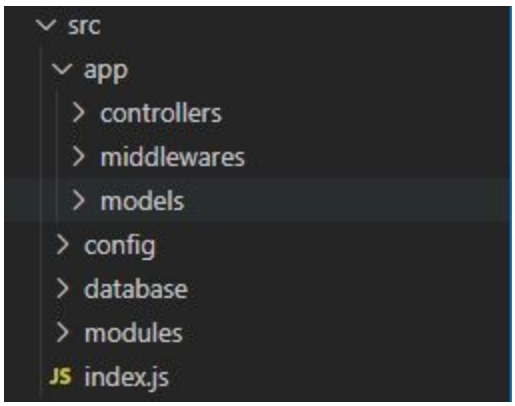

Figura 2 - Estrutura de pastas do backend

```
const UserSchema = new mongoose.Schema({
 name: { type: String, required: true },
 email: {
   type: String,
   unique: true,
   required: true,
   lowercase: true,
   unique: true
 },
 role: { type: String, enum: ["admin", "user"], default:
"user" },
 password: { type: String, required: true, select: false
},
 course: { type: String },
 profilePicture: String,
 passwordResetToken: { type: String, select: false },
 passwordResetExpires: { type: Date, select: false },
 createdAt: { type: Date, default: Date.now }
});
UserSchema.pre("save", async function(next) {
 const hash = await bcrypt.hash(this.password, 10);
 this.password = hash;
 next();
});
const User = mongoose.model("User", UserSchema);
```

Figura 3: Modelo de Usuário da aplicação

A pasta config é onde estão os arquivos de configuração da aplicação, apresentados na *Figura 4*, como configurações do servidor SMTP, responsável pelo envio dos emails da aplicação, as credenciais de acesso ao mongo, um arquivo com o token aleatório para gerar e autenticar o token JWT do usuário, e por último um arquivo com as credenciais do usuário de email da aplicação. A pasta database é onde está o arquivo de configuração e acesso ao Mongo.

Por último, está a pasta modules, que possui os módulos externos da aplicação. Nesse caso, está presente apenas o arquivo mailer, responsável por enviar o email de reset de senha do usuário.

Uma melhor exemplificação de como funciona a estrutura do NodeJS com o banco de dados Mongo pode ser visto na *Figura 4*:

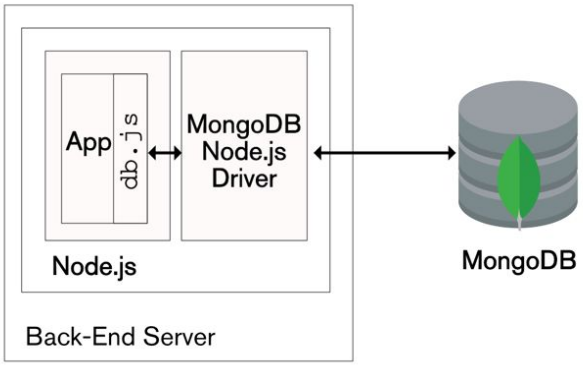

Figura 4: Exemplificação de como o NodeJS trabalha em conjunto com o Mongo. [8]

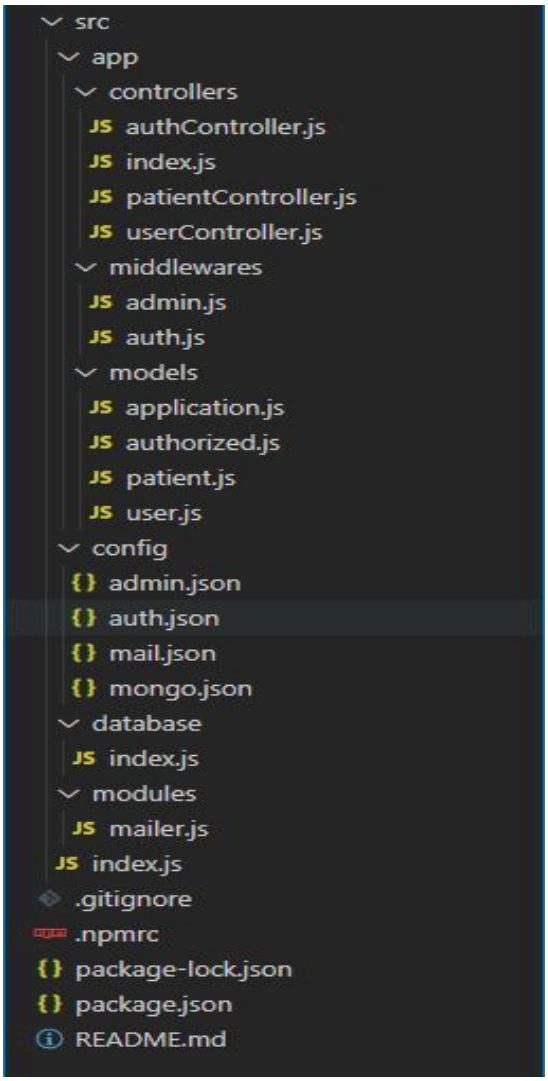

Figura 4 - Arquivos da aplicação backend

2.2.3 - Autenticação

Para a autenticação do usuário, foi utilizado Json Web Tokens (JWT). O token de usuário é gerado no backend, após receber e validar os dados do usuário, utilizando uma chave aleatória, no momento que o usuário faz o login na aplicação, como mostra a *Figura 5*. A autenticação da aplicação funciona da seguinte forma: primeiramente o usuário, através do cliente, insere seu nome de usuário e senha; em seguida o backend recebe e valida as informações e, caso estejam corretas, gera um token referente ao usuário com um prazo de validade,; por último, o token é enviado para o cliente que o armazena para utilizá-lo no campo *Authorization* do *Header* de futuras chamadas referentes a rotas protegidas. Um diagrama para mostrar melhor o processo de autenticação pode ser visto na *Figura 8.*

```
router.post("/authenticate", async(req, res) => {
    try {
        const { email, password } = req.body;
              const user = await User.findOne({ email
}).select("+password");
        if (!user) {
            return res.send({ error: "Usuário não existe"
});
        }
        if (!bcrypt.compareSync(password, user.password))
{
            return res.send({ error: "Senha inválida" });
        }
        user.password = undefined;
          return res.send({ user, token: generateToken({
id: user.id }) });
    } catch (error) {
        console.log(error);
         return res.status(400).send({ error: "Ocorreu um
erro na autenticação" });
    }
});
```

Figura 5: Método de Autenticação implementado no lado do servidor.

2.3 - Tecnologias utilizadas no frontend

O lado do cliente foi feito utilizando a biblioteca ReactJs [9], também feita em Javascript, criada e mantida pelo Facebook. Entre os motivos para a escolha dessa tecnologia para o frontend, estão: maior domínio devido a experiências passadas com a biblioteca; Arquitetura baseada em componentes reutilizáveis, que facilita bastante o desenvolvimento e manutenção da aplicação; Fácil gerenciamento de estado utilizando uma outra biblioteca, que funciona como um complemento e em total sintonia com o React, chamada Redux [11]. A arquitetura implementada pelo React/Redux pode ser vista na *Figura 6.*

Figura 6: Estrutura padrão utilizada por uma aplicação React/Redux. [10]

2.3.1 - Estrutura do frontend

O frontend da aplicação foi dividido em 6 pastas principais: assets, components, pages, redux, styles e templates, como mostrado na *Figura 7.*

A primeira pasta é a assets. Nela estão contidos os arquivos de fontes e todas as imagens utilizadas na aplicação. A segunda pasta é a components, na qual estão contidos todos os componentes da aplicação. Logo em seguida temos a pasta pages, que contém todas as páginas da aplicação, onde cada uma é listada e atribuída uma rota no arquivo de rotas do frontend. A página seguinte é a redux, que é subdividida em 2 pastas: actions e reducers. As actions são responsáveis por receber uma entrada ou fazer uma conexão com o servidor, e gerar um novo estado para a aplicação. Já os reducers são funções puras que recebem os resultados das actions e propagam essa mudança de estado para todos os componentes que utilizam aquela parte da aplicação. A próxima é a pasta styles, que contém todos os arquivos de estilização da aplicação, adotando o mesmo nome do componente que se está utilizando. Por exemplo, o componente CollaboratorCard é formado por 2 arquivos: CollaboratorCard.jsx, que é o componente em si, e o arquivo CollaboratorCard.css, que possui a estilização para esse componente. A última pasta é a templates, que possui todos os templates da aplicação, que podem ser utilizados para gerar páginas diferentes porém com muitos elementos em comum.

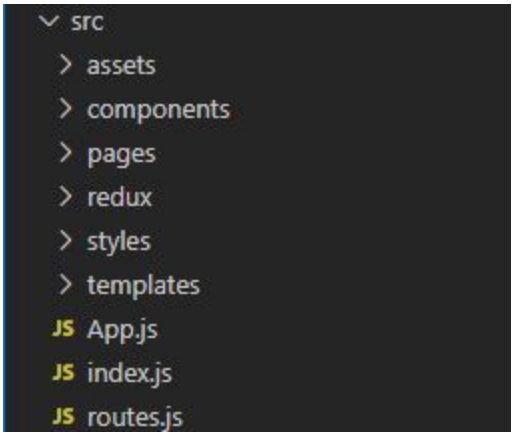

Figura 7 - Estrutura de pastas do frontend

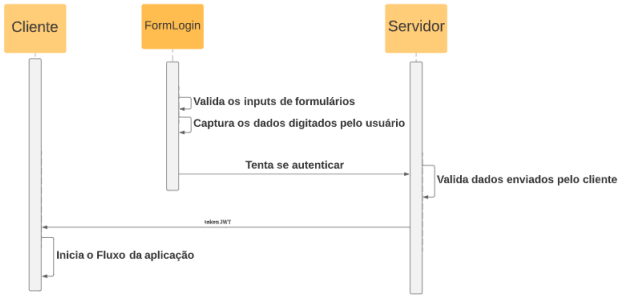

Figura 8: Diagrama de Sequência do Fluxo de Autenticação da aplicação

3. SISTEMA EM USO

Nesta seção serão descritas a metodologia e resultados do sistema em uso.

Foi disponibilizada uma versão inicial do aplicativo para os colaboradores do projeto Laserterapia, e por uma semana eles puderam utilizar a versão inicial do aplicativo.

Para coletar os dados referentes à satisfação com o aplicativo, foi utilizado um formulário do Google, com perguntas sobre as funcionalidades já existentes na aplicação. O formulário foi respondido por 10 colaboradores do projeto.

O questionário consiste em 9 questões bastante objetivas, que visam saber se cada funcionalidade do sistema está funcionando corretamente. Dentre essas 9, existe uma na qual foi solicitado para os colaboradores darem uma nota para o quanto eles acham que o aplicativo irá ajudar no dia-a-dia do projeto. O resultado pode ser visto na *Figura 8.*

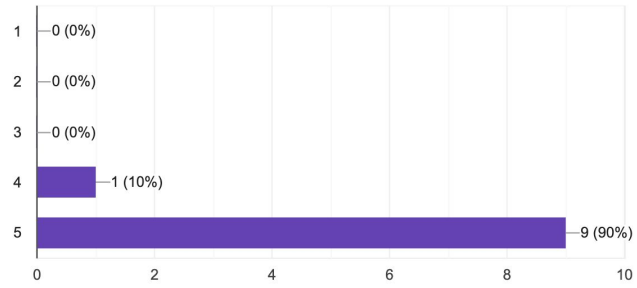

Figura 8: Grau de satisfação dos colaboradores com o aplicativo.

Sobre as funcionalidades existentes, as respostas foram unânimes de que estão todas funcionando corretamente. Como exemplo, podemos citar as respostas referentes a corretude da funcionalidade de login e cadastro de novos colaboradores pelo usuário administrador, onde ambas apresentaram uma corretude de 100% de acordo com os usuários.

Ao serem questionados se há alguma funcionalidade que iria ajudá-los mais no dia-a-dia que não está implementada atualmente, o professor coordenador do projeto sugeriu o seguinte:

- "Infográfico ou dados numéricos sobre informações gerais dos pacientes. Por exemplo: quantidade de pacientes dosexo masculino e feminino, quantidade de pacientes dos 8 aos 11 anos.".

Como pode-se ver, de acordo com as avaliações dos usuários, o sistema implementado para o projeto Laserterapia irá ajudar bastante os colaboradores, tendo ainda um bom espaço para evolução com novas funcionalidades.

4. EXPERIÊNCIA

Nesta seção apresentamos as experiências e lições aprendidas durante todo o processo deste trabalho, com ênfase no processo de desenvolvimento, desafios encontrados e trabalhos futuros.

4.1 Processo de desenvolvimento

Primeiramente, foi feita uma reunião com o coordenador do projeto para aprender mais sobre o projeto, e quais seriam as funcionalidades necessárias a serem implementadas. Após isso, foram decididas as tecnologias que seriam utilizadas e começou-se o desenvolvimento.

Em um primeiro momento, foi feita uma implementação completa do backend de todas as funcionalidades do backlog, com

os testes sendo executados via postman. Após isso, iniciou-se o desenvolvimento do frontend, que iria consumir os endpoints implementados anteriormente no backend. O processo não foi auxiliado por nenhuma ferramenta, seguindo apenas o backlog que estava organizado por funcionalidades.

Para o versionamento do código, foi utilizado a ferramenta Github, e para disponibiliza-la para os colaboradores, foi utilizado a plataforma de hospedagem de aplicações em nuvem, Heroku.

4.2 Desafios

Sobre o desenvolvimento, o principal desafio foi conseguir estruturar o processo de desenvolvimento em si. Após a estruturação do backlog, o mais viável teria sido adotar um processo de implementação incremental, uma funcionalidade por vez. Fazendo isso, talvez o trabalho tido com testes de regressão e principalmente mudanças no backend para atender as necessidades do frontend, tivesse sido consideravelmente menor.

Coletar os requisitos também foi um desafio grande, tendo em vista que é uma parte muito importante do desenvolvimento da aplicação, já que uma boa coleta irá ser uma boa fundação para o desenvolvimento, sendo verdade também o contrário. Outro desafio importante foi conseguir criar uma interface amigável, junto com toda uma identidade visual para o projeto.Nesse aspecto, acredito que a aplicação foi um sucesso, tendo em vista os comentários positivos do Coordenador e dos colaboradores do projeto. Em termos técnicos, o principal desafio foi de construir a aplicação em perfeita sincronia entre backend e frontend, evitando o máximo possível retrabalho em ambas as implementações.

4.3 Trabalhos futuros

Para trabalhos futuros, pretende-se implementar novas funcionalidades ao projeto, como uma nova tela para visualização de gráficos informativos sobre os pacientes, por exemplo um para mostrar uma curva de acordo com uma nota atribuída ao paciente pelo colaborador na hora da aplicação, de quão boa está a situação deste paciente. Além disso, há espaço para ajustes na interface visual do projeto, visando torná-lo mais atraente aos usuários.

5. REFERÊNCIAS

[1] Projeto Laserterapia UFCG:

https://laserterapiaufcg.wixsite.com/home

[2] Arquitetura cliente-servidor:

https://www.canalti.com.br/arquitetura-de-computadores/arquitetura-cliente-servidor/#:~:text=Defini%C3%A7%C3%A3o,dos%20dados%20(os%20clientes).

[3] Imagem exemplificando comunicação entre cliente-servidor:

https://www.quora.com/What-is-the-best-combination-in-building-a-new-web-application-React-js-Node-js-or-React-js-Go

[4] Por que usar Node.js? Uma justificativa passo a passo:

https://blog.geekhunter.com.br/por-que-diabos-eu-usaria-o-node-js-uma-justificativa-passo-a-passo/

[5] Introdução ao MongoDB:

https://www.devmedia.com.br/introducao-ao-mongodb/30792

[6] NodeJS - MongoDB vs MySQL:

https://pt.stackoverflow.com/questions/166150/node-js-mongodb-x-mysql

[7] The perfect architecture flow for your next Node.js project:

https://blog.logrocket.com/the-perfect-architecture-flow-for-your-next-node-js-project/

[8] Exemplo de comunicação entre NodeJS e Mongo:

https://www.mongodb.com/blog/post/the-modern-application-stack-part-2-using-mongodb-with-nodejs

[9] React JS: Uma biblioteca javascript para criar interfaces de usuário:

https://pt-br.reactjs.org/

[10] Arquitetura de uma aplicação que utiliza React/Redux:

https://www.esri.com/arcgis-blog/products/3d-gis/3d-gis/react-redux-building-modern-web-apps-with-the-arcgis-js-api/

[11] Medium, Hélio Kroger, Entendendo React e Redux de uma vez por todas:

https://medium.com/@hliojnior_34681/entenda-react-e-redux-de-uma-vez-por-todas-c761bc3194ca