UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES

ACH2077 - Soluções Web Baseadas em Software Livre
Prof. Dr. Marcio Moretto Ribeiro

Integrated Medical Records - Unificando Prontuários Médicos

Augusto Calado Bueno (9779134)
Fernando de Padua Kikuti (9276901)
Regiany Nunes Almeida (8921496)

São Paulo
2019

ÍNDICE

RESUMO

Tendo em vista a inexistência nos dias atuais de um sistema que integre informações médicas, o presente estudo teve como objetivo o desenvolvimento de um sistema Web capaz de unificar e centralizar informações de saúde.

INTRODUÇÃO

A **Base médica integrada (BMI)** é uma plataforma online cujo objetivo é a centralização das informações de saúde, armazenando dados e registros de seus usuários como: histórico de imunização, histórico de exames, prontuários de consultas e diagnósticos passados provenientes de hospitais, clínicas e consultórios frequentados pelo paciente.

Tais informações serão disponibilizadas para seus usuários pacientes e, também, para profissionais da saúde com as devidas permissões de acesso, que poderão dispor-se dos dados contidos na plataforma para auxiliar no atendimento e diagnóstico do paciente, garantindo agilidade, qualidade e a precisão dos serviços prestados.

O principal segmento de atuação da BMI (Base Médica Integrada) será o de saúde, tendo como principais usuários as clínicas, consultórios, convênios e hospitais.

Atualmente existem diversas instituições médicas diferentes, algumas com o mesmo objetivo, entretanto estas organizações não conversam entre si e cria-se um problema grave de comunicação e inconsistência de dados, muitas vezes exigindo um retrabalho quando um cliente migra de uma instituição para outra, também há outro problema, o de diagnósticos imprecisos, que ocorre em função do fato de que para que o paciente seja diagnosticado com maior assertividade, é necessário dispor de um histórico médico de outros diagnósticos e exames. A BMI quer suprir essa necessidade, diminuindo a granularização de dados médicos em geral, seja em rede pública ou privada, permitindo um serviço de saúde de maior qualidade para a população e simplificar a vida do profissional de saúde na sua rotina diária.

FUNCIONALIDADES

Carteira de Vacinação

Com esta funcionalidade objetivamos virtualizar o registro das imunizações para os usuários cadastrados no sistema. No sistema realizamos a digitalização das imunizações já aplicadas em um paciente e mantemos esses registros atualizados inserindo novos registros à medida que o paciente realiza novas imunizações ou toma doses de reforço.

Carteira de Emergência

A carteira de emergência é uma funcionalidade pensada para ser um repositório de informações vitais para o atendimento do paciente. Nela existe a possibilidade de se cadastrar qualquer informação relevante para ser considerada no momento da prestação de serviços médicos como alergias, tipo sanguíneo, doenças crônicas etc.

Dividimos os cartões em três grandes grupos, que são:

- Alergias:
    - Caso o usuário possui alergia severa à alguma medicação, é possível cadastrar um cartão contendo as seguintes informações: O que causa alergia (e.g. nome agente causador) e informações adicionais (e.g. O que fazer caso uma crise alérgica aconteça);
- Doenças:
    - Uma informação vital a ser considerada na hora de se prestar serviços médicos em uma situação de emergência são as doenças contraídas pelo usuário. Por esse motivo é pode-se incluir no cartão de emergência tais informações;
- Informações gerais:
    - Demais tipos de informações podem ser adicionadas no grupo de informações gerais, como por exemplo se é doador de orgãos, tipo sanguíneo e etc.

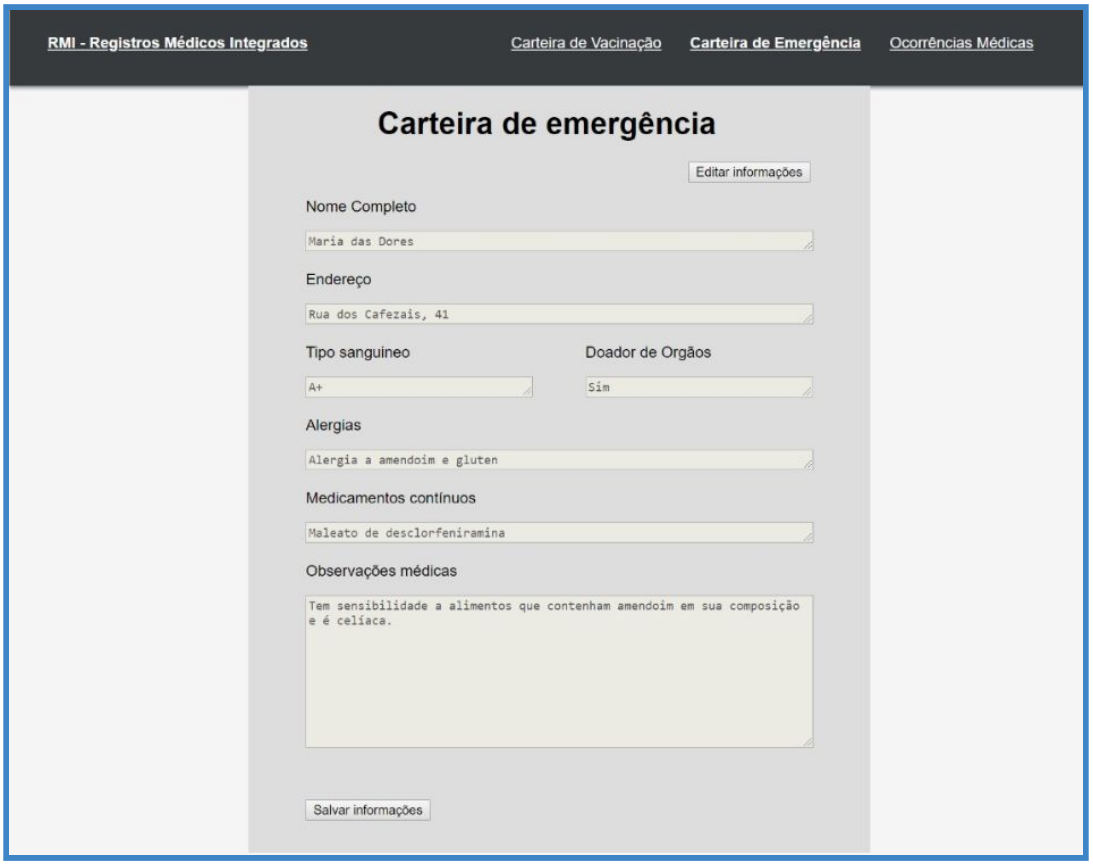

Figura - *Carteira de Emergência*

Ocorrências Médicas

O propósito da funcionalidade de ocorrências médicas é guardar e disponibilizar para os usuários os dados produzidos em cada sessão de atendimento médico, como diagnósticos, exames solicitados, condições da saúde no momento da prestação do atendimento (e.g. pressão ou se possuía febre). Para tal, virtualizados as etapas do procedimento de anamnese.

Quando o usuário logar no sistema, conseguirá acessar as informações das consultas passadas, podendo fornecer essas para ajudar profissionais da saúde no atendimento e na prestação de seus serviços.

Template do prontuário médico serializado pelo sistema

```
{
  "crm": "string",
  "currentDiseaseInfo": "string",
  "diagnostic": "string",
  "idMedicalRecord": 0,
  "mainComplain": "string",
  "medicalExam": [
    {
      "examName": "string",
      "examObservation": "string",
      "idMedicalExam": 0,
```

```
      "prescriptionDate": "string"
    }
  ],
  "medicalRecordDate": "string",
  "medicalRecordPlace": "string",
  "physicalExam": [
    {
      "idPhysicalExam": 0,
      "namePhysicalExam": "string",
      "value": "string"
    }
  ],
  "prescription": "string"
}
```


MÉTODOS

Desenvolvimento Backend

Ferramentas, Linguagens e Frameworks Utilizados

- Kotlin
- Spring Framework e Spring Boot
- Swagger
- JPA/Hibernate

Kotlin

Kotlin é a linguagem utilizada no backend do projeto. Kotlin é compilado para a Máquina virtual Java. Foi desenvolvida pela empresa JetBrains, tornou-se a linguagem oficial da plataforma mobile Android em 2017.

Optamos por utilizá-la, pois trata-se de uma linguagem bem versátil com suporte à paradigmas de programação como programação orientação a objetos e programação funcional.

Spring Framework e Spring Boot

O Spring Framework é um framework para desenvolvimento de aplicações java. Já o SpringBoot é um projeto da Spring que visa facilitar o processo de configuração do Spring Framework.

O Spring Framework conta com o paradigma de **injeção de dependências**. O objetivo desse padrão é diminuir a dependência entre as classes. E isso é alcançado da seguinte forma: As instâncias que uma determinada **classe A** precisa para ser instanciada e executada são geradas externamente à **classe A** e, posteriormente, são passadas para a classe que necessita dessas instâncias para executar a regra de negócio definida dentro dela (no caso, a classe A).

A injeção de dependência é um dos tipos do paradigma chamado inversão de controle, pois o controle de instanciar as propriedades que a classe A precisa fica por conta de quem usará a classe A.

Para exemplificar os conceitos anterior, abaixo existem duas classes uma chamada UrlRepository que é responsável por fazer operações no banco de dados e a segunda é a WebCrawler que é uma classe convencional que executa alguma lógica. Será possível notar duas diferentes implementações da classe WebCrawler que é com ou sem injeção de dependência.

Classe que realiza função de busca no BD - Exemplo

```
public interface UrlRepository extends MongoRepository<Url, ObjectId> {

    Url findByUrl(String url){

        #Code
    }
}
```

Sem injeção de dependência - Exemplo

```
public class WebCrawler {

    private final int TIMEOUT = 30000;
    private HashSet<String> urls;
    UrlRepository urlRepository

    public WebCrawler() {
        urls = new HashSet<String>();
        urlRepository = new UrlRepository();
    }
    #Code
}
```

Com injeção de dependência gerenciada pelo Spring Framework - Exemplo

```
public class WebCrawler {
    private final int TIMEOUT = 30000;
    private HashSet<String> urls;

    public WebCrawler() {
        urls = new HashSet<String>();
    }

//Injeção de Dependência

    @Autowired
    UrlRepository urlRepository
}
```

Swagger

Utilizamos o Framework Swagger para documentar os endpoints dos controllers desenvolvidos no backend. Com o Swagger conseguimos exemplificar modelos de request e response.

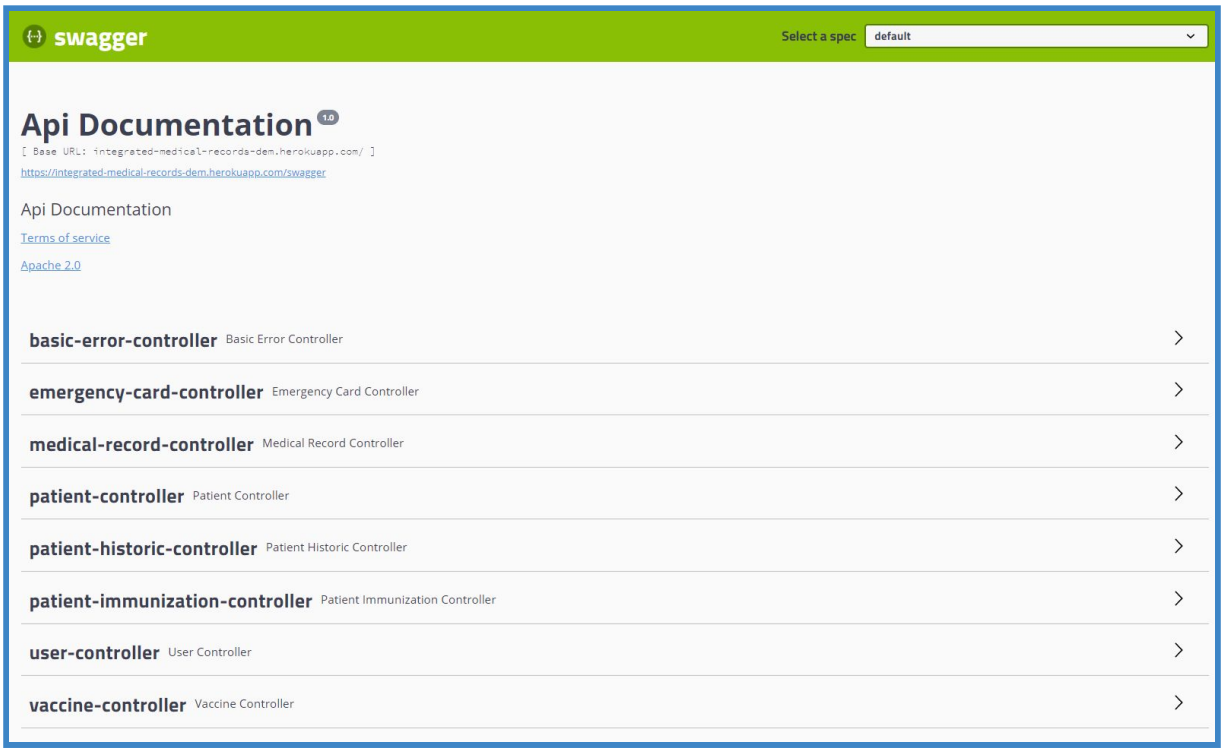

Figura - Swagger do Projeto

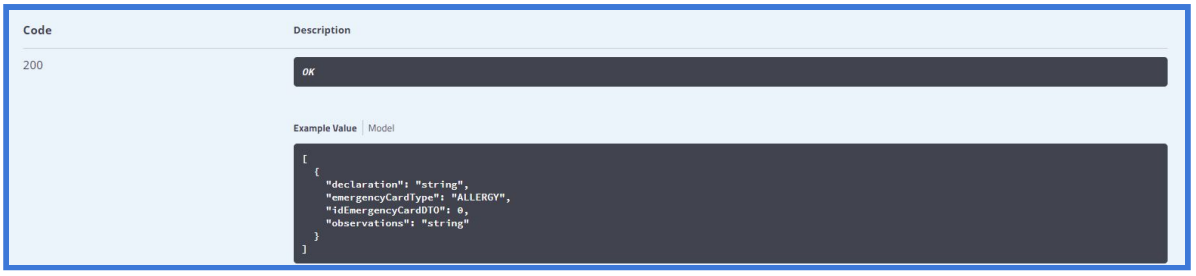

Figura - Response Exemplo Configurado no Swagger

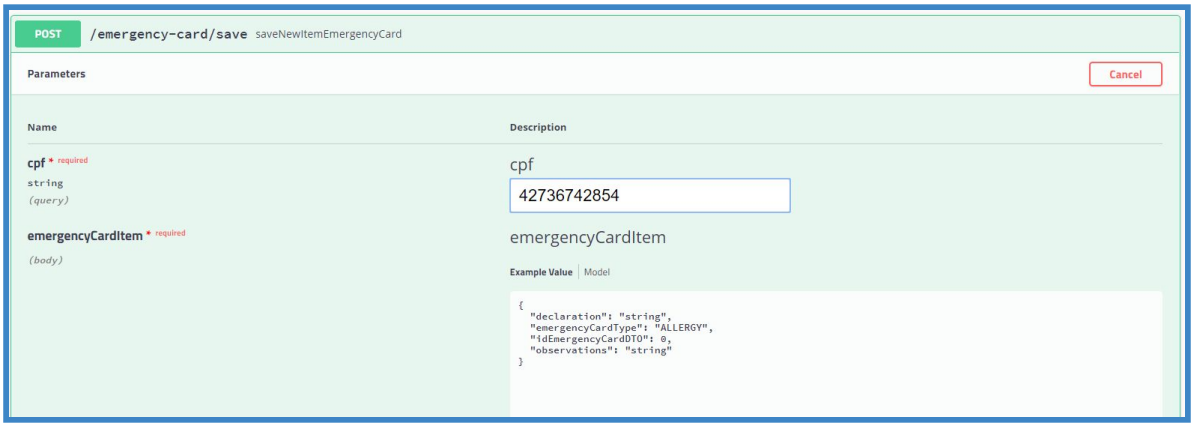

Figura - Post Request Exemplo Configurado no Swagger

JPA/Hibernate

O projeto contou com a utilização do Hibernate/JPA para realizar as operações de CRUD (create, retrieve, update, delete). Hibernate é uma ferramentas de mapeamento objeto-relacional (ORM).

Java Persistence API (ou simplesmente JPA) é uma API padrão da linguagem Java que descreve uma interface comum para frameworks de persistência de dados. A JPA define um meio de mapeamento objeto-relacional para objetos Java simples e comuns (POJOs), denominados beans de entidade (WIKIPEDIA, 2016).

No projeto, as classes que representam os objetos relacionais então dentro do pacote **domain**. Para a manipulação das informações contidas dos objetos do **domain** aplicou-se o padrão de projeto *Data Transfer Object* (DTO), e as classes que representam esses objetos estão localizadas no pacote **domain.dto.** Utilizamos os DTOs nas camadas de **serviço** e **controle** da aplicação e as classes domain na camada de **repositórios** onde operações com o banco de dados são realizadas.

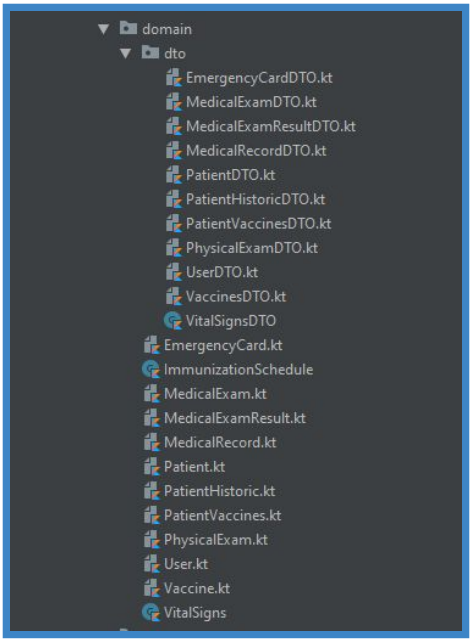

Figura - *Domain and DTO Layer*

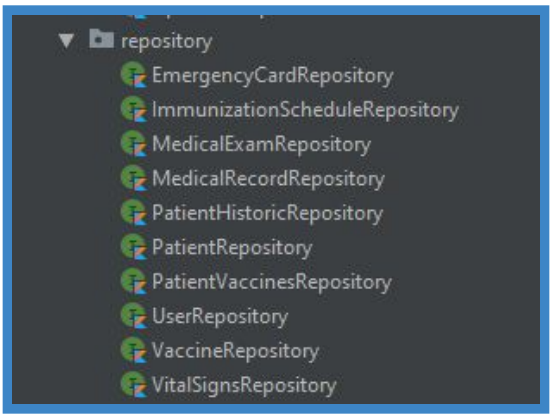

Figura - *Repository Layer*

Desenvolvimento Frontend

Escolhemos o Angular 6 para essa função. Ele é um framework em Typescript (considerado um superset do Javascript) que gerencia de forma modular o CSS, HTML e Javascript da página.

Dentre outros elementos, os que mais utilizamos foram:

- componentes: podem ser entendidos como uma versão simplificada de uma estrutura que compõe o DOM. Uma das grandes facilidades que seu uso nos trás, é a possibilidade de estruturar o código de maneira reaproveitável.
- roteamento: elemento básico que nos permite a transição entre uma página e outra.
- módulos: diretiva que nos ajuda a organizar melhor o código, assim podemos, por exemplo, agrupar módulos de componentes e serviços que tenham um objetivo em comum.
- serviços: podem servir a mais de um propósito mas, idealmente, eles são criados para encapsular trechos de regra de negócio específicas como por exemplo, cada componentes que deve realizar sua interação com o backend tem seu próprio serviço para gerenciá-las.

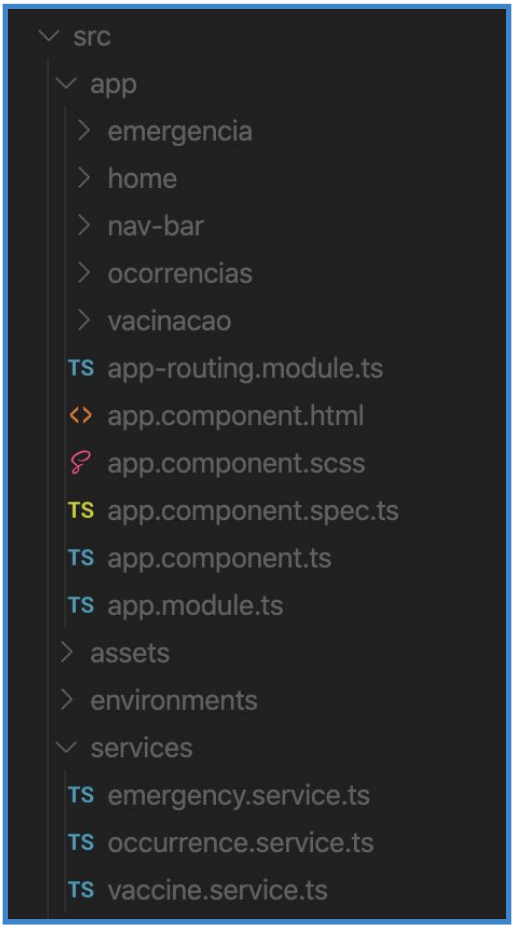

Figura - *Estrutura de pastas do do código*

Tivemos a criação de 4 componentes principais e dois secundários, além do base (a partir do qual é iniciado o roteamento). São eles:

- nav-bar: responsável pelo roteamento entre as três telas e, foi reaproveitado em todos os outros componentes principais.
- vacinação: componente principal que chama o secundário de vacinação/card. Implementa as regras de negócio para a funcionalidade de "Carteira de Vacinação"

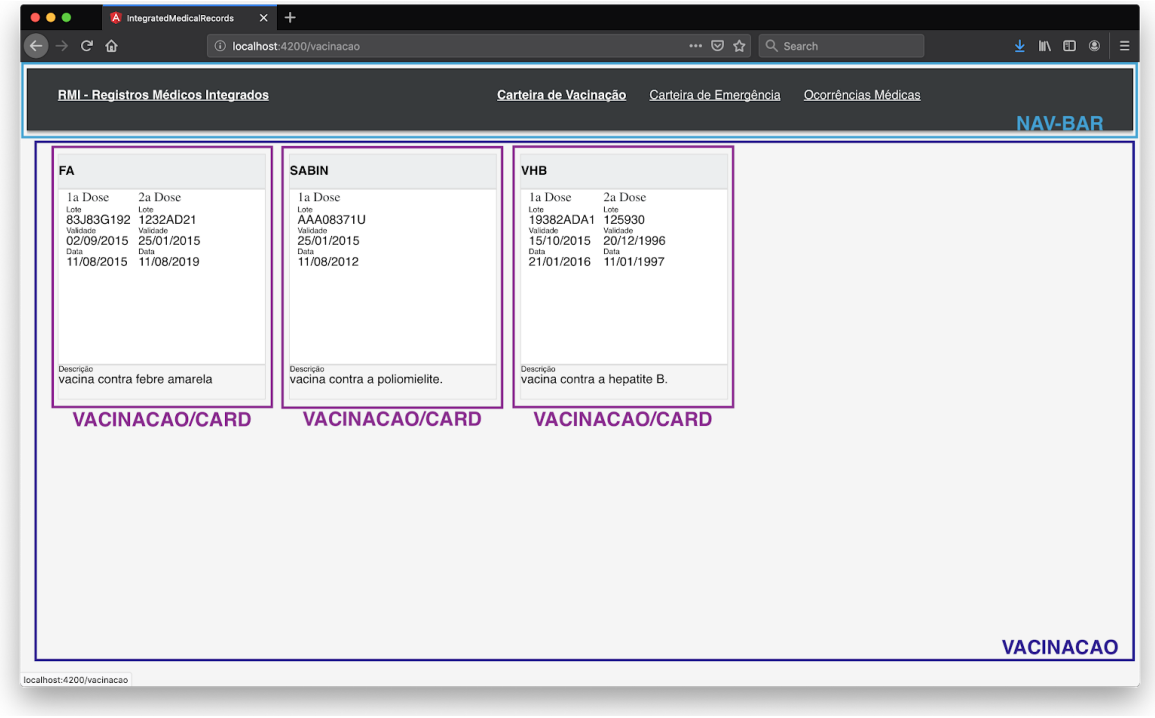

Figura - *Tela de Carteira de Vacinação e seus respectivos componentes*

- ocorrências: componente principal que chama o secundário de ocorrencias/card. Implementa as regras de negócio para a funcionalidade de "Ocorrências Médicas"

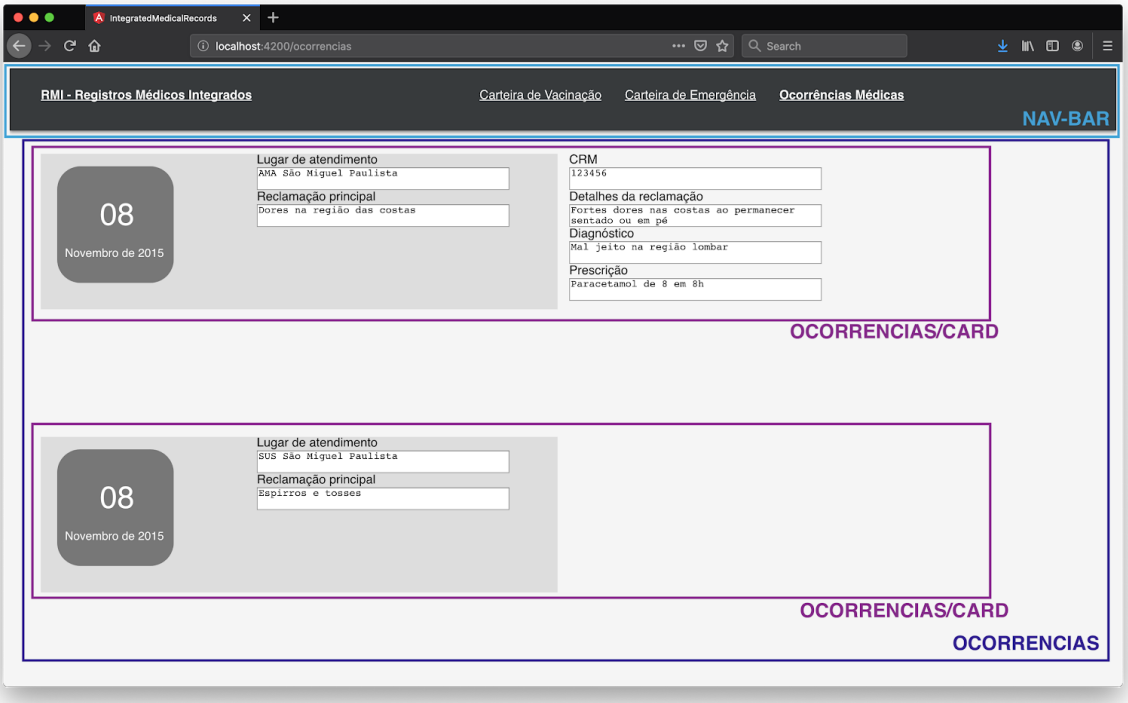

Figura - *Tela de Ocorrências Médicas e seus respectivos componentes*

- emergência: componente principal que chamada o secundário de vacinacao/card. Implementa as regras de negócio para a funcionalidade de "Carteira de Emergência"

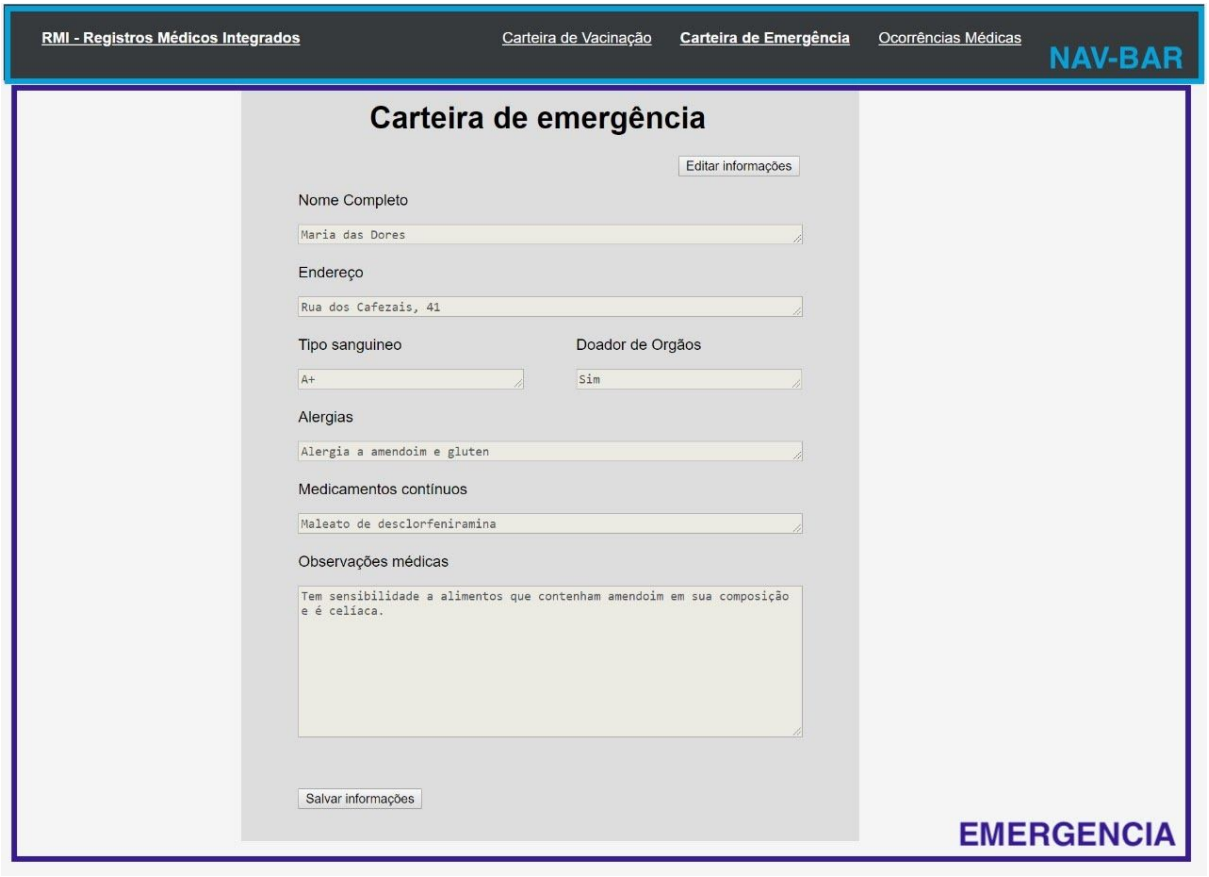

Figura - *Tela de Carteira de Emergência e seus respectivos componentes*

Já para os serviços, criamos 3 que se referem ao componentes *vacinação*, *ocorrências* e *emergência.* Cada um deles possui sua respectiva integração com os endpoints do backend para recuperação (via GET) e envio (via POST - caso necessário).

Banco de Dados

O projeto utilizou dois bancos de dados, um voltado para o ambiente de produção e outro voltado para o ambiente de desenvolvimento. O banco de dados utilizado em ambiente de produção é o MySQL (banco relacional e open source); já para o ambiente de desenvolvimento, utilizou-se o banco de dados em memória H2 (também é classificado como SQL).

Hibernate e JPA foram utilizados para a criação das estruturas do banco de dados (schemas e tabelas) de maneira programática. Com o mapeamento das tabelas em classes, conseguimos, utilizando o JPA, criar toda a estrutura do banco de dados sem a necessidade da utilização scripts.

O schema final do projeto desenvolvido para a disciplina de soluções web contém em sua totalidade 11 tabelas.

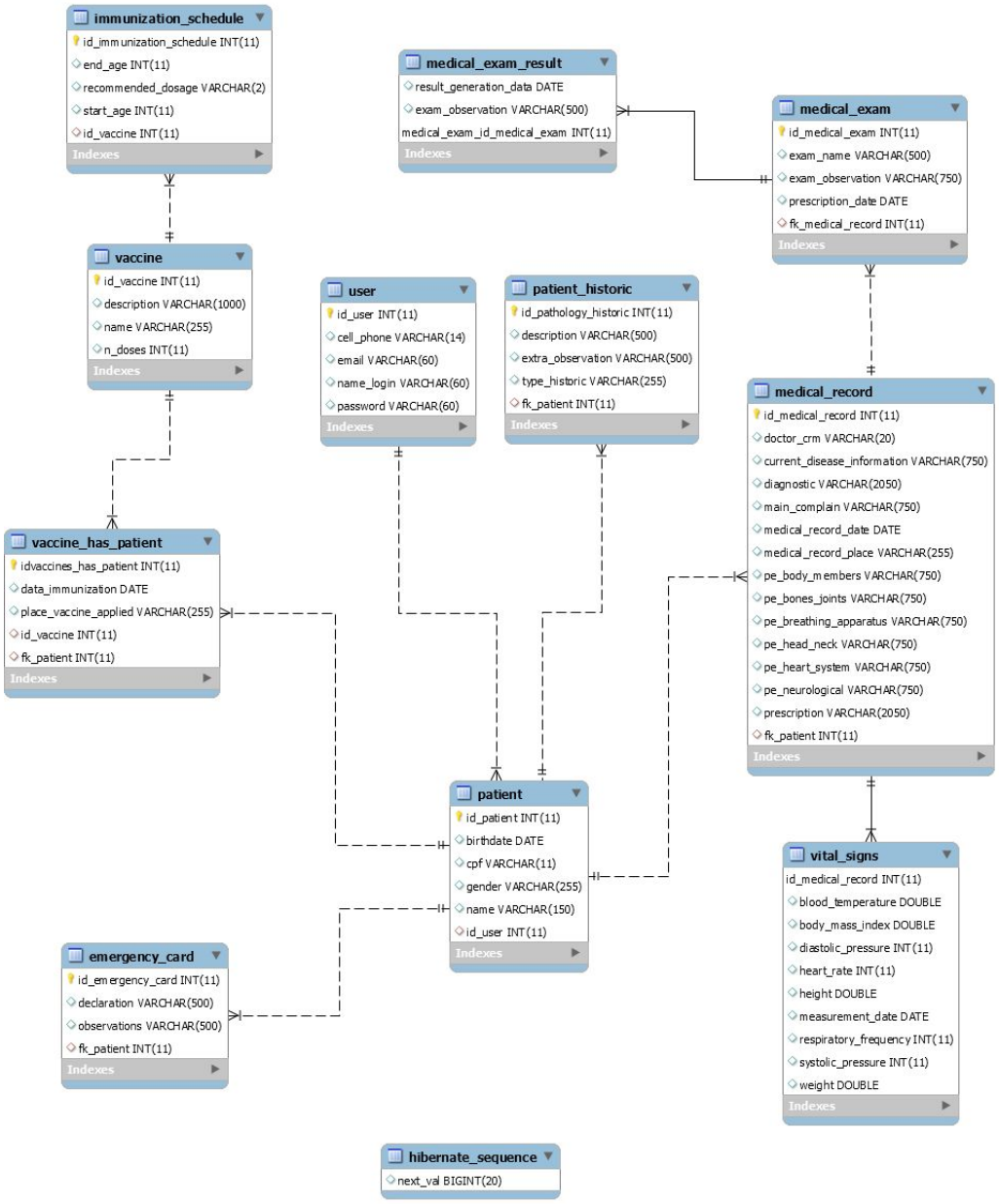

Figura - *Schema* do Projeto

Versionamento de código

Utilizamos o Git para o controle de versionamento do projeto. Além disso disponibilizamos o código fonte, que foi dividido em dois repositórios (frontend e backend) na plataforma GitHub. O código fonte do frontend encontra-se no branch **master** do repositório https://github.com/AugustoCalado/Integrated-Medical-Records-Front-End; já a implementação do código do backend, encontra-se no brach **no-hateoas-impl** do repositório https://github.com/AugustoCalado/Integrated-Medical-Records.

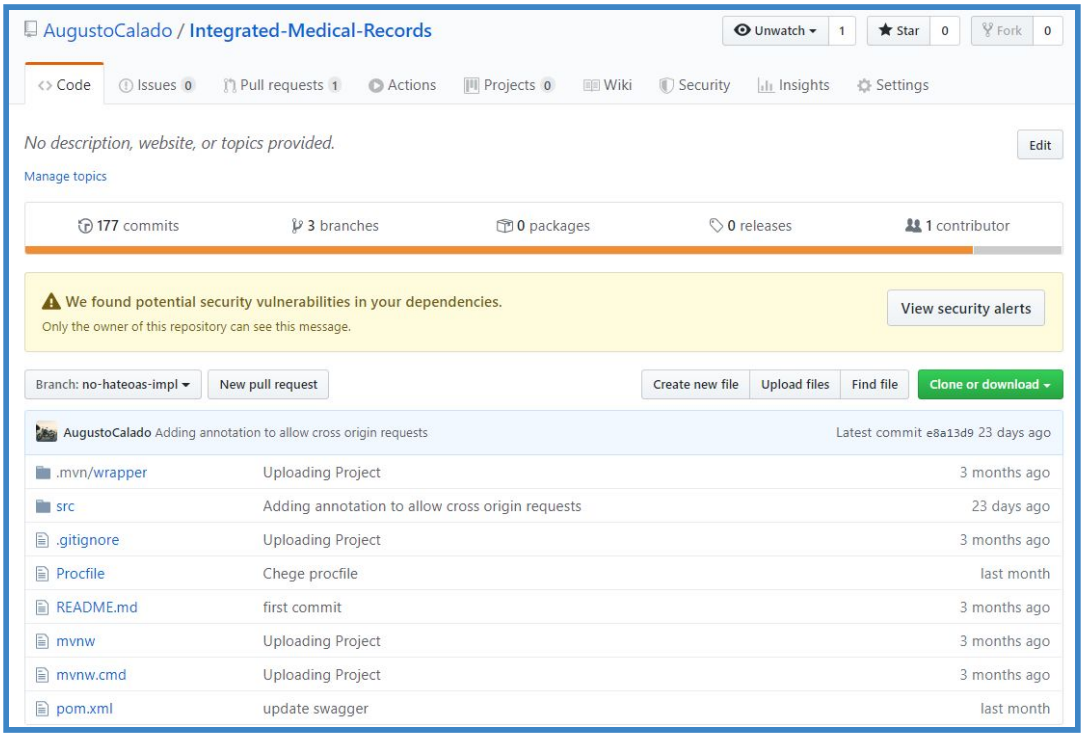

Figura - GitHub Page Backend

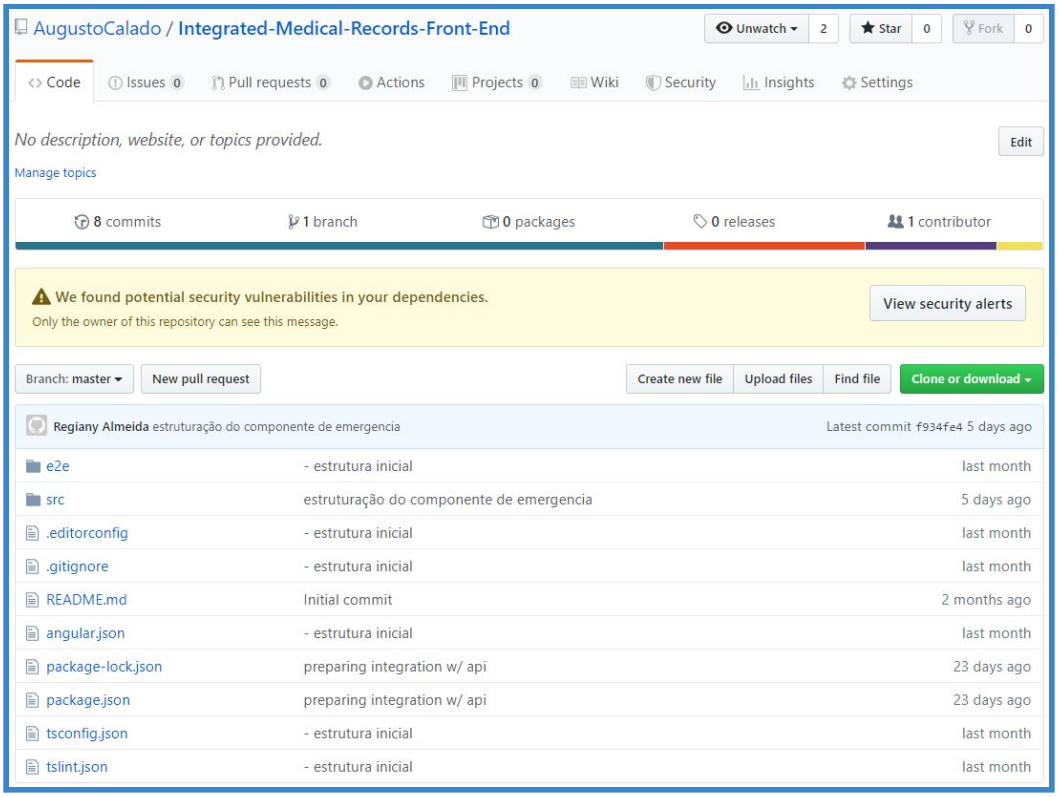

Figura - GitHub Page Frontend

Hospedagem

Para hospedar o back end do projeto, escolheu-se o Heroku, um nuvem PaaS (plataform as a service) que além de prover os serviços de hospedagem em nuvem, também nos proporcionou a inclusão do banco de dados relacional MySQL.

Através do Heroku CLI e do versionamento do código fonte no Github, conseguimos abstrair a realização do deploy da aplicação que, localmente, exigia conexão com banco de dados, configurações de variáveis de ambiente, JVM e Maven (sistema de automação de compilação), para um processo simples e rápido, bastando-se executar um comando para inicializar a aplicação no serviço de nuvem do Heroku.

git push heroku git_branch:master - Realiza o push de uma branch do github para o Heroku

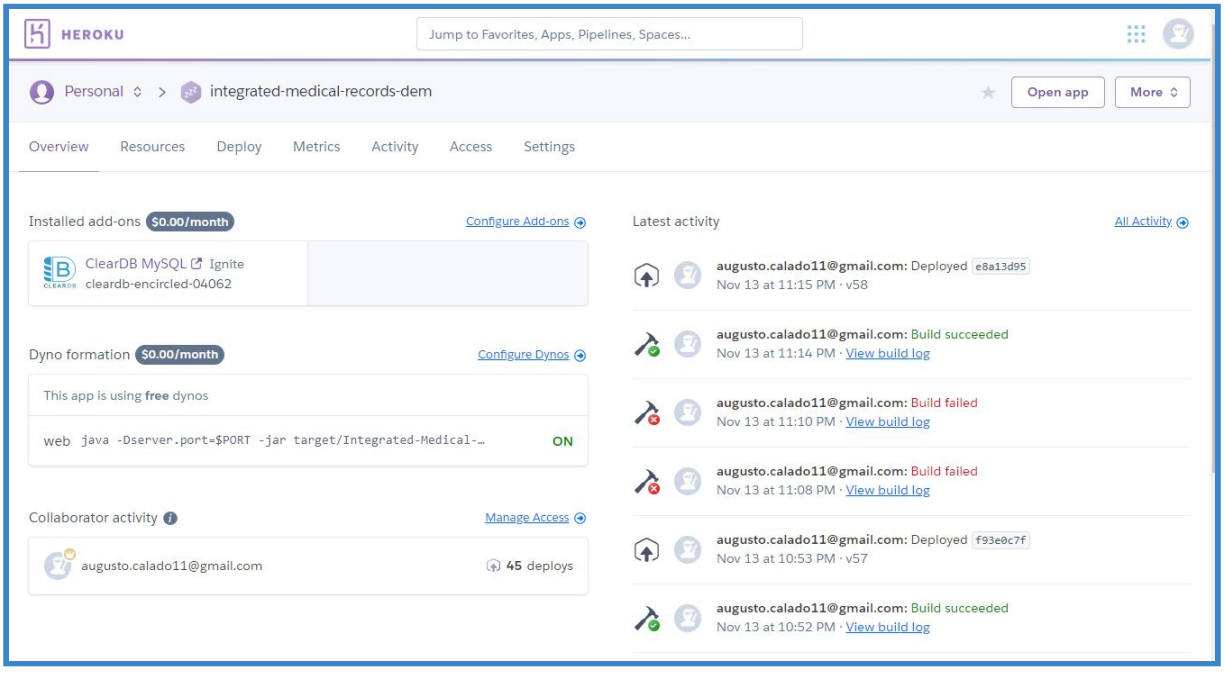

Figura - Heroku Interface

RESULTADOS

Os requisitos definidos no plano do projeto foram:

- Acesso ao registro de imunização (Carteirinha de Vacina Virtual);
- Acesso aos diagnósticos e prontuários produzidos em consultas médicas passadas;
- Carteira de Emergência do Paciente;

Para um nível de protótipo, o projeto atendeu os requisitos de acesso ao registro de imunização e prontuários médicos. Conseguimos simular uma comunicação entre um sistema externo (aquele usado pelo profissional de saúde no momento do atendimento), com o sistema desenvolvido pelo grupo. Nesta simulação, o sistema externo utiliza a nossa API de cadastro de informação e o Front-end acessa os endpoints de consulta do backend para exibir ao usuário as informações providas pelo sistema externo.

Com relação ao requisito de Carteira de Emergência do Paciente, conseguimos atender parte dos requisitos estabelecidos. No backend foram criados endpoints para consulta dos cartões de emergência para um determinado usuário, atualização das informações dos cartões de emergência e criação de novos cartões. Todas as operações funcionam como esperado, porém no momento da integração com o frontend, não conseguimos utilizar os endpoints para atualização e criação de cartões devido ao mecanismo CORS (Cross-origin resource sharing), que são bloqueados por padrão tanto nos frameworks do angular quanto no Spring. Por esse motivo, conseguimos apenas realizar as requisições de consultas e edição de um cartão, porém a edição não é persistida em nossa base de dados.

Outra tentativa de implementação realizada pelo grupo foi a utilização da arquitetura HATEOAS (Hypermedia as the Engine of Application State), no backend. Uma API HATEOAS provê informações que permite navegar entre seus endpoints de forma dinâmica visto que inclui links junto às respostas (GONÇALVES, 2016). Devido a dificuldades de implementação da arquitetura HATEOAS utilizando o SpringFramework e também pela escassez de tempo para se estudar alternativas para introdução dessa arquitetura, optamos por não seguir com essa solução.

CONCLUSÃO

Ao desenvolver o protótipo da Base Médica Integrada, tomamos conhecimento e colocamos em prática grandes modelos de arquitetura (MVC) e boas práticas de programação. Além disso pudemos trabalhar e explorar nossas ferramentas que estão se popularizando no mercado de trabalho, tais como Kotlin, SpringBoot e Angular.

Entendemos que o projeto de software proposto pelo grupo é desafiador e possui diversas barreiras a serem transpostas, mas apesar das dificuldades seu impacto social positivo tem um grande potencial de ajudar diversas pessoas e instituições.

Para trabalhos futuros, uma possível abordagem para o problema de gerenciamento das informações médicas, seria o uso de ontologias e *linked data*, que poderiam ser usados para relacionar os dados pulverizados de informações médicas que estão entre as instituições de saúde.

BIBLIOGRAFIA

WIKIPÉDIA. Java Persistence API. Disponível em: <https://pt.wikipedia.org/wiki/Java_Persistence_API>. Acesso em: 08 Dez 2019

GONÇALVES, Leandro da Costa Gonçalves. Entendendo HATEOAS. Disponível em: <http://www.semeru.com.br/blog/en/>. Acesso em: 08 Dez 2019