UNIVERSIDADE ESTADUAL DA PARAÍBA
PRÓ-REITORIA DE PÓS-GRADUAÇÃO E PESQUISA
PPROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA E TECNOLOGIA EM SAÚDE

JOSÉ GEORGE DIAS DE SOUZA

UMA ARQUITETURA DE MICROSSERVIÇOS PARA UMA APLICAÇÃO DE SUPORTE AO ACOMPANHAMENTO PRÉ-NATAL

CAMPINA GRANDE
2022

JOSÉ GEORGE DIAS DE SOUZA

UMA ARQUITETURA DE MICROSSERVIÇOS PARA UMA APLICAÇÃO DE SUPORTE AO ACOMPANHAMENTO PRÉ-NATAL

Dissertação apresentada ao Programa de Pós-Graduação em Ciência e Tecnologia em Saúde da Universidade Estadual da Paraíba, como requisito parcial para a obtenção do título de Mestre em Ciência e Tecnologia em Saúde.

Área de concentração: Engenharia de Software, Usabilidade e Fatores Humanos.

Orientador: Prof. Dr. Daniel Scherer

CAMPINA GRANDE
2022

S729a Souza, Jose George Dias de.

Uma arquitetura de microsserviços para uma aplicação de suporte ao acompanhamento pré-natal [manuscrito] / Jose George Dias de Souza. - 2022.

115 p. : il. colorido.

Digitado.

Dissertação (Mestrado em Profissional em Ciência e Tecnologia em Saúde) - Universidade Estadual da Paraíba, Pró-Reitoria de Pós-Graduação e Pesquisa , 2022.

"Orientação : Prof. Dr. Daniel Scherer , Coordenação do Curso de Computação - CCT."

1. Gestação. 2. Pré-Natal. 3. Educação em saúde. 4. Arquitetura da informação. I. Título

21. ed. CDD 005.1

Elaborada por Camile de A. Gomes - CRB - 15/559 BC/UEPB

JOSÉ GEORGE DIAS DE SOUZA

UMA ARQUITETURA DE MICROSSERVIÇOS PARA UMA APLICAÇÃO DE ACOMPANHAMENTO PRÉ-NATAL

Dissertação apresentada ao Programa de Pós-Graduação em Ciência e Tecnologia em Saúde da Universidade Estadual da Paraíba como requisito para obtenção do título de Mestre em Ciência e Tecnologia em Saúde.

Dissertação aprovada em: 03/06/2022

BANCA EXAMINADORA:

Prof. Dr. Daniel Scherer
Universidade Estadual da Paraíba (UEPB)

Documento assinado digitalmente
gov.br VIVIAN CARDOSO DE MORAIS OLIVEIRA
Data: 26/07/2022 08:19:14-0300
Verifique em https://verificador.iti.br

Profa. Dra. Vivian Cardoso de Morais Oliveira
Universidade Federal de Campina Grande (UFCG)

Documento assinado digitalmente
gov.br ADEMAR VIRGOLINO DA SILVA NETTO
Data: 13/06/2022 16:05:45-0300
Verifique em https://verificador.iti.br

Prof. Dr. Ademar Virgolino da Silva Netto
Universidade Federal da Paraíba (UFPB)

Dedico este trabalho aos meus pais Genésio e Cleonice, a minha esposa Maria pela dedicação, companheirismo e amizade.

AGRADECIMENTOS

Agradeço a Deus por tantas graças na minha vida, pelas oportunidades e pela força, sobretudo em momentos nos quais eu estava sem ânimo. Agradeço à virgem Maria, pela grande intercessão junto ao Pai durante toda a minha caminhada acadêmica.

À minha família, especialmente à minha esposa Maria, por toda a energia e incentivo para que eu chegasse ao final deste mestrado. Ainda de forma especial agradeço aos meus pais, que sempre me apoiaram e incentivaram para que eu chegasse até aqui. Agradeço aos meus irmãos que torceram para que eu conseguisse finalizar essa importante etapa.

Ao meu orientador Daniel Scherer, agradeço imensamente pela colaboração, paciência e apoio durante esse período. Também agradeço aos membros da banca examinadora, pela contribuição e pela disponibilidade.

Aos meus professores do mestrado e também aos meus colegas, especialmente Ewerthon e Gustavo por todo o apoio e incentivo.

“É preciso trabalhar como se a gente não fosse morrer nunca, e viver como se a gente devesse morrer todos os dias.” (São João Bosco).

RESUMO

A assistência pré-natal é indispensável visto que tem como finalidade promover o desenvolvimento seguro e saudável da gestação. O propósito do pré-natal é assegurar um acompanhamento saudável da gestação, reduzindo os riscos relacionados à gestação para a mãe e o bebê. O acompanhamento pré-natal tem um papel importante na detecção, prevenção e tratamento de doenças. Contudo, existem alguns fatores que levam a uma parte da população de gestantes a não realizar adequadamente o acompanhamento. Com a constante evolução da ciência da computação, das mídias sociais e sobretudo dos smartphones as pessoas estão cada vez mais utilizando a internet como fonte de informação. Dessa forma os aplicativos desenvolvidos para o acompanhamento da gestação têm ganhando bastante notoriedade por pesquisadores que desenvolvem tecnologias para a saúde. Diante da necessidade de desenvolver aplicações escaláveis, com responsabilidades bem definidas, heterogeneidade de linguagens e adaptativos, vem crescendo o uso de aplicações com uma arquitetura de microsserviços na indústria de desenvolvimento. Um modelo arquitetural de microsserviços mostra-se ser mais eficiente do que o modelo monolítico, principalmente no contexto de uma aplicação que necessita atender inúmeros usuários em tempo real, além disso uma arquitetura de microsserviços é mais tolerante a falhas.

Palavras-chave: Gestação; Pré-Natal; Educação; Microsserviços.

ABSTRACT

Prenatal care is essential as it aims to promote the safe and healthy development of pregnancy. The purpose of prenatal care is to ensure a healthy monitoring of pregnancy, reducing the risks related to pregnancy for the mother and baby. Prenatal care plays an important role in the detection, prevention and treatment of diseases. However, there are some factors that lead to a part of the population of pregnant women not performing the follow-up properly. With the constant evolution of computer science, social media and especially smartphones, people are increasingly using the internet as a source of information. In this way, applications developed for monitoring pregnancy have gained much notoriety by researchers who develop technologies for health. Faced with the need to develop scalable applications, with well-defined responsibilities, heterogeneity of languages and adaptives, the use of applications with a microservices architecture in the development industry has been growing. A microservices architectural model proves to be more efficient than the monolithic model, especially in the context of an application that needs to serve numerous users in real time, in addition a microservices architecture is more fault tolerant.

Keywords: Pregnancy; Prenatal Care; Prenatal Education; Microservices.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

AMQP	*Advanced Message Queuing Protocol*
API	*Application Programming Interface*
BFF	*Back-end* para *Front-end*
CQRS	*Command Query Responsability Segregation*
DDD	*Domain-Driven Design*
ERP	*Enterprise Resource Planning*
GoF	*Gang of Four*
HTTP	*Hypertext Transfer Protocol*
IoT	*Internet of Things*
Jakpros	*Jakarta Reproduksi Sehat*
JPA	*Java Persistence API*
JSON	*JavaScript Object Notation*
LGPD	Lei Geral de Proteção de Dados
LTS	*Long-term support*
QA	*Quality Assurance*
SaaS	Software como serviço
SGBD	Sistema de gerenciamento de banco de dados
SO	Sistema Operacional
SPA	*Single Page Applications*
SQL	*Structured Query Language*
SUS	Sistema Único de Saúde
TDD	*Test Driven Development*
VM	Máquina Virtual
WEB	*World Wide Web*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

A assistência pré-natal é indispensável visto que tem como finalidade promover o desenvolvimento seguro e saudável da gestação. De acordo com Carroli, Rooney e Villar (2001), o desenvolvimento de práticas acolhedoras durante a gestação está correlacionado com um desfecho positivo do parto. O propósito do pré-natal é assegurar um acompanhamento saudável da gestação, reduzindo os riscos relacionados à gestação para a mãe e o bebê. Dessa maneira, o Ministério da Saúde (2006) propôs uma série de ações desejáveis para serem realizadas durante o pré-natal, entre essas: ações educativas com o intuito de reduzir os medos vinculados à gestação e condutas acolhedoras.

O pré-natal tem um papel importante na detecção, prevenção e tratamento de doenças. Para Lansky *et al.* (2014), os cuidados do pré-natal são importantes para redução da mortalidade materno-infantil, porém, de acordo com Tomasi *et al.* (2017), uma parte considerável da população não realiza adequadamente o pré-natal. Há inúmeros fatores que contribuem para as mulheres não realizarem devidamente o acompanhamento pré-natal. De acordo com Coutinho *et al.* (2010), a dificuldade de acesso por parte da gestante a uma unidade de saúde especializada é um deles.

Com o propósito de oferecer uma melhor assistência às gestantes e um pré-natal acolhedor, foi instituído pelo Ministério da Saúde (2011) a Rede Cegonha. Contudo, ainda existem obstáculos no Sistema Único de Saúde (SUS) a serem superados, por exemplo, as tecnologias disponibilizadas atualmente nas unidades de saúde não são eficazes para atender as gestantes que vivem geograficamente distantes. Além disso, o acesso a determinadas regiões é difícil, dessa maneira, as gestantes acabam por não receber o acompanhamento dos profissionais de saúde *in loco* como sugerido pelo (MINISTÉRIO DA SAÚDE, 2006).

Com a constante evolução da ciência da computação, das mídias sociais e, sobretudo, dos *smartphones*, as pessoas estão cada vez mais utilizando a internet como fonte de informação. Segundo McCann e McCulloch (2012), as mulheres, sobretudo de países industrializados, estão usando as redes sociais em busca de informações sobre gravidez, parto e amamentação. Em um estudo realizado por Lima-Pereira, Bermúdez-Tamayo e Jasienska (2012), mostrou-se que as gestantes confiavam nas informações disponibilizadas na internet sobre gestação, contudo, a maior parte das participantes não conhecia as fontes dessas informações.

Para Wicahyono *et al.* (2019) a internet e o *smartphone* exercem grande influência na interação humana tal como nos sistemas sociais: saúde, educação e setor comercial. Atualmente a computação *ubíqua* tem ganhado bastante notoriedade por aprimorar o uso do computador, tornando os computadores disponíveis em todo o ambiente conforme definição de (WEISER, 1993). De acordo com Kim *et al.* (2018) é importante adicionar a computação *ubíqua* à saúde, sobretudo para fornecer uma análise em tempo real do paciente e oferecer um tratamento mais assertivo.

De acordo com Drey e Consel (2010) a computação *ubíqua* propõe-se a atender às demandas de usuários pertencentes a diversas áreas tais como: saúde, automação residencial e gerenciamento de energia. Para Dewi *et al.* (2009) um aplicativo WEB *ubíquo* trata-se de um aplicativo que funcione a todo momento, em qualquer lugar e nos mais diferentes aparelhos.

As tecnologias adotadas pelo Ministério da Saúde (2006) já não atendem em completude às necessidades das mulheres, para Tripp *et al.* (2014) o modelo tradicional de assistência pré-natal precisa incorporar as novas tecnologias, o que inclui os aplicativos de apoio à gestação. Na área da tecnologia em saúde existem diversas aplicações de *software* com o propósito de monitorar, informar e capturar dados em tempo real da gestante. No trabalho desenvolvido por Wiweko *et al.* (2018) foi proposto um aplicativo com a finalidade de prover informações confiáveis para a grávida e com isso colaborar com o aprendizado da gestante.

Os estudos mais recentes que apresentam como solução tecnológica o desenvolvi- mento de um *software*, normalmente focam a abordagem em construir funcionalidades focadas na mulher, permitindo dessa forma, orientar a gestante sobre a gestação por meio de conteúdo educativo, monitoramento da gestante e comunicação direta com a unidade de saúde (EFREM *et al.*, 2019), (WICAHYONO *et al.*, 2019) e (SMITH *et al.*, 2017). Contudo as pesquisas não se preocuparam em projetar uma arquitetura de *software escalável*, sendo sempre feita a opção por uma arquitetura tradicional monolítica.

1.1 Problemática

Em uma arquitetura de *software* monolítica todos os serviços estão dentro de uma única base de código e esses serviços se comunicam com serviços externos através de interfaces diferentes, tais como: Interface de programação de aplicações

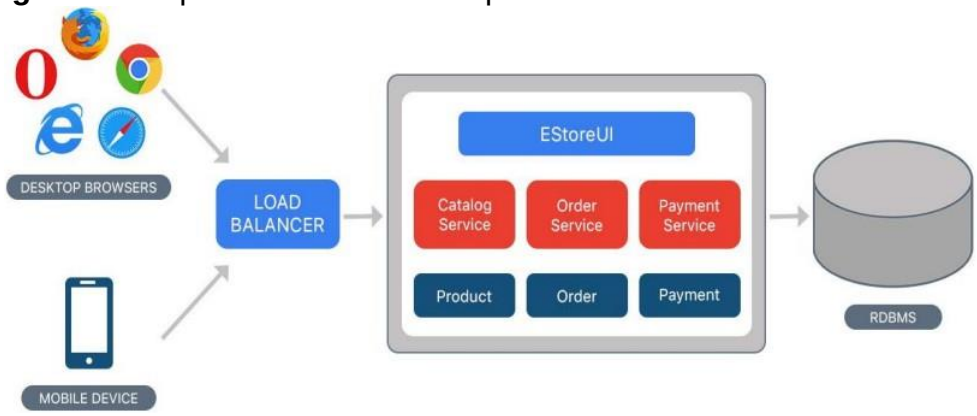
(API) e páginas *HyperText Markup Language* (HTML) (AL-DEBAGY; MARTINEK, 2018). Normalmente a comunicação se dá via API quando a aplicação possui um *front-end* desacoplado do *back-end*.

De acordo com Gos e Zabierowski (2020) uma arquitetura monolítica engloba todos os componentes de uma aplicação como a autorização, regra de negócios, apresentação, camada de banco de dados, integrações e todos os demais módulos da aplicação, combinados em único *software a* partir de uma única plataforma.

Na Figura 1 na qual contém uma representação de uma arquitetura monolítica para um sistema de comércio eletrónico é possível verificar que tanto o módulo de catálogo de serviço, como de ordem de serviço e o pagamento estão dentro de um único *software*.

À medida que novos usuários passam a utilizar a aplicação, o custo com manutenção, evolução e com servidores aumentam de forma significativa, pois existe um grande desafio em escalar uma aplicação monolítica. Na arquitetura monolítica não há a possibilidade de se utilizar diferentes linguagens de programação além disso, quando uma falha ocorre, toda a aplicação é afetada (AL-DEBAGY; MARTINEK, 2018).

Figura 1 - Arquitetura monolítica representando um comércio eletrônico



Fonte: Gos e Zabierowski (2020)

Diante da necessidade de desenvolver aplicações escaláveis, com responsabilidades bem definidas, heterogeneidade de linguagens e adaptativas, vem crescendo o uso de aplicações com uma arquitetura de microsserviços na indústria de desenvolvimento. De acordo com Bandara e Perera (2020) a indústria consente com a importância de migrar os sistemas monolíticos para uma arquitetura baseada em microsserviços.

Para Jagadeesan e Mendiratta (2020) o desenvolvimento e implantação de uma arquitetura de microsserviços ocorre de forma mais rápida, tendo em vista que os serviços podem ser dimensionados e atualizados independentemente. Atualmente esse modelo arquitetural vem sendo amplamente utilizado nas mais diversas áreas sobretudo em aplicações *Internet of Things* (IoT) pois possui atributos de qualidade como interoperabilidade, resiliência, escalabilidade e manutenibilidade que são essenciais para um *software* (CABRERA *et al.*, 2020).

Portanto é preciso projetar e desenvolver sistemas, considerando muitos fatores, sobretudo quem são os *stakeholders*. Dessa forma, o modelo arquitetural de microsserviços mostrou ser mais eficiente do que o modelo tradicional monolítico, no contexto de *software* que precisa ter escalabilidade para atender inúmeros usuários.

Com a difusão da internet das coisas, nos últimos anos, as tecnologias têm se tornado cada vez mais presentes na saúde. Conforme Dinh-Le *et al.* (2019), as instituições de saúde procuram cada vez mais realizar integrações de tecnologias para dessa forma aprimorar os cuidados com os pacientes. Para Sharma *et al.* (2018) as tecnologias digitais fornecem uma oportunidade significativa para prestação de cuidados e investigação de doenças.

De acordo com Butler Tobah *et al.* (2019), o acompanhamento virtual no período pré-natal é preponderante para reduzir o estresse da gestante além de propiciar mais satisfação e qualidade de vida. Em vista disso, é necessário considerar alguns fatores no desenvolvimento de qualquer aplicação para a saúde, como por exemplo: a segurança dos dados do paciente. Dessa maneira é preciso propor uma arquitetura escalável, onde seja possível aumentar a capacidade de novos usuários sob medida e segura, fazendo utilização de protocolos de segurança apropriados no intuito de atender gestantes, médicos e demais profissionais envolvidos com a gestação. Além disso, é preciso centralizar os dados em nuvem, tornando-os acessíveis a qualquer momento para sistemas integrados ou usuários com as devidas autorizações.

1.2 Objetivos

Neste trabalho, tem-se como objetivo o desenvolvimento de uma arquitetura de microsserviços centralizada a qual irá receber requisições de um sistema WEB e um aplicativo *mobile*. O objetivo dessa arquitetura é tornar mais rápida a comunicação

entre os profissionais de saúde e a gestante, além disso contribuir para reduzir os receios vivenciados pelas as gestantes através de um pré-natal mais humanizado e acolhedor.

Em um segundo momento, os dados gerados por esta aplicação poderão ajudar na tomada de decisão por parte das autoridades, de forma que possam direcionar investimentos e atenção para regiões com o maior índice de mulheres gestantes.

1.2.1 Objetivos Específicos

- Desenvolver uma arquitetura de microsserviços centralizada e capaz de se comunicar com outras aplicações;
- Desenvolvimento de uma aplicação WEB, a qual irá ser utilizada por profissionais de saúde responsáveis por prestar o atendimento pré-natal a gestante;
- Implementar um aplicativo *mobile*, capaz de funcionar em *smartphone* com limitação no *hardware.*

1.3 Estrutura do Documento

Este trabalho está organizado em 8 capítulos, no capítulo 2 é apresentado o referencial teórico no qual é introduzido o conceito de tecnologia em saúde, além disso são abordados temas relevantes para a pesquisa, como arquitetura de microsserviços, observabilidade e usabilidade.

No capítulo 3 são apresentados os trabalhos relacionados com o tema do projeto, inicialmente são discutidos os critérios utilizados para a seleção dos estudos e em seguida são abordados 5 estudos relevantes. No capítulo 4 é exposta a metodologia utilizada, no início deste capítulo é apresentado o m método ´ágil Scrum seguido do processo de versionamento de código, por fim são descritas as tecnologias utilizadas neste trabalho.

No capítulo 5 é apresentado o desenvolvimento da arquitetura proposta, bem como a modelagem da mesma, seguido por uma apresentação sobre o *front-end* e o modelo de dados utilizado. No capítulo 6 é debatido sobre a segurança da informação, são abordados tópicos relevantes para garantir a segurança de qualquer aplicação e

apresentados diferentes fluxos do protocolo OAuth 2. O capítulo 7 aborda a validação da arquitetura de microsserviços proposta, são abordados pontos importantes da qualidade de *software* e em seguida são realizados teste de desempenho em um ambiente *on-premises*. Por fim, são expostas as conclusões.

Além disso, no apêndice A são descritos os artefatos da aplicação e no apêndice B é apresentado o artigo publicado que foi gerado por esta pesquisa.

2 REFERENCIAL TEÓRICO

Neste capítulo será apresentado a fundamentação teórica acerca do pré-natal no Sistema Único de Saúde (SUS), tecnologias em saúde e tecnologias empregadas na realização do pré-natal. Em seguida, será exposto o modelo arquitetural proposto nesta pesquisa: Arquitetura de Microsserviços para o desenvolvimento de um sistema WEB, voltado para atender as gestantes. Além disso, serão apresentados conceitos importantes sobre usabilidade e uma descrição dos requisitos da plataforma proposta.

2.1 Assistência Pré-natal no Sistema Único de Saúde

O pré-natal é uma das mais importantes etapas vinculadas à gravidez, pois tem como objetivo propiciar o desenvolvimento seguro e saudável da gestação, garantindo mais qualidade de vida durante e após a gestação para a mulher e o recém-nascido. Segundo Carroli, Rooney e Villar (2001), o pré-natal quando, devidamente realizado, reduz os riscos relacionados a gravidez. Para Barros *et al.* (2010), o pré-natal é diretamente responsável por desfechos positivos na gestação, com isso é evidente a importância de um acompanhamento efetivo e humanizado com a gestante.

No decorrer do período gestacional, as mulheres acompanhadas recebem um suporte humanizado, buscando assim trabalhar a questão psicológica, procurando com isso reduzir os receios vivenciados na gestação. Outras atividades importantes também são realizadas tais como: práticas educativas que fornecem informações sobre a gestação, consultas médicas e análises preventivas no intuito de reduzir incidências inesperadas durante a fase gestacional. A orientação do Ministério da Saúde (2006) decorre sobre incorporar condutas acolhedoras, tais como ações educativas e preventivas, para com isso evitar intervenções médicas indesejadas.

O pré-natal é fundamental para, de forma precoce, rastrear e identificar doenças ou qualquer tipo de enfermidade que possa ser nociva para a gestação. Conforme o Ministério da Saúde (2006) aponta, é direito da mulher ter um pré-natal no SUS, assegurado o atendimento básico, bem como o direito de usufruir de atendimento mais especializado e específico para casos de alto risco. Além do mais, é dever do SUS garantir que a gestante possa conhecer de forma antecipada o estabelecimento onde será realizado o parto.

Apesar do atendimento pré-natal oferecido pelo SUS cobrir um percentual de 98,7% de gestantes da população brasileira de acordo com o Viellas *et al.* (2014), ainda existem muitas lacunas que precisam ser sanadas. De acordo com Coimbra *et al.* (2003), atualmente muitas gestantes iniciam tardiamente o pr é-natal, ou seja, o SUS falha no processo de identificar as gestantes de forma adequada. Em conformidade com Coutinho *et al.* (2010) existem em alguns casos dificuldade de acesso para a gestante chegar a unidade de saúde, consequência disso são as gestantes que residem em regiões remotas e na zona rural. A pesquisa de Tomasi *et al.* (2017) aponta que uma parte considerável da população não realiza o número adequado de seis consultas definidas pela OMS (Organização Mundial da Saúde) (OMS, 2005).

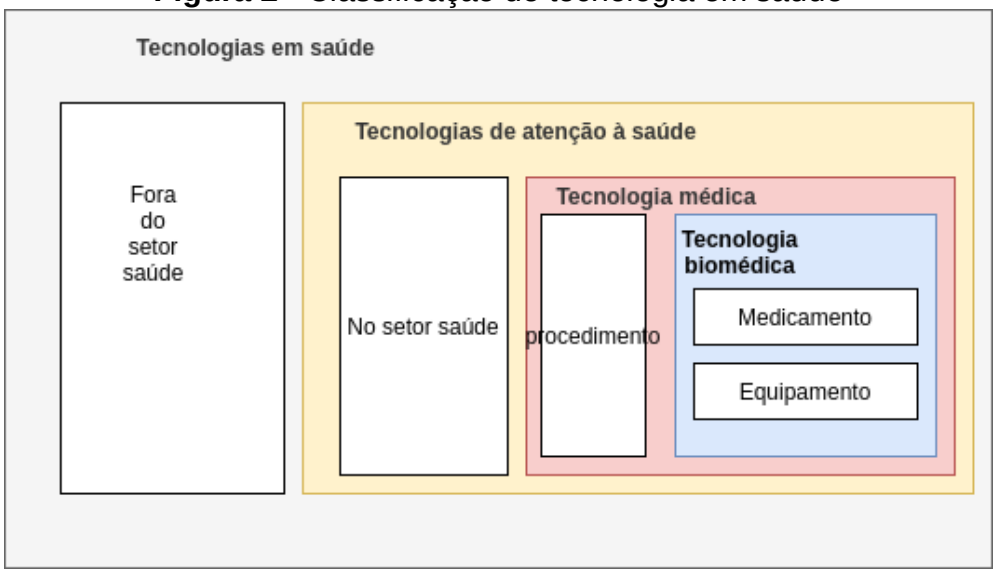
Segundo Lansky *et al.* (2014) os devidos cuidados do pré-natal estão diretamente ligados a uma gestação saudável reduzindo a mortalidade materno-infantil. Muito embora quase todas as mulheres gestantes sejam acompanhadas durante a gestação pelo SUS, existe muito ainda para ser feito. De acordo com Tomasi *et al.* (2017) a percentagem de mulheres que não realizam devidamente o pré-natal pode chegar a um quinto da população de gestante. As características populacionais do grupo de gestantes mais afetadas e que não realizam devidamente o pré-natal são mulheres em situação de vulnerabilidade econômica, jovens, com baixa escolarização e que vivem na região norte e nordeste. A partir desta pesquisa, visa-se propor e desenvolver uma plataforma, contemplando um sistema web, para utilização dos médicos e demais profissionais de saúde. Um aplicativo móvel, para ser usado pelas gestantes e um conjunto de API's de integração, onde será feita em tempo real toda a comunicação e sincronização dos dados em nuvem.

2.2 Tecnologia em Saúde

O termo tecnologia é vasto, Ferreira (2008) definiu o conceito de tecnologia como um agrupamento de conhecimento e princípios científicos. Para Liaropoulos (1997) pode ser considerada uma tecnologia em saúde: equipamentos, procedimentos, suporte médico e cuidados. De acordo com Ministério da Saúde (2004) as tecnologias em saúde não se limitam a máquinas e *softwares*, mas qualquer cuidado oferecido, apoio e comunicação bem estabelecida com o paciente pode ser

considerado uma tecnologia em saúde. A Figura 2 contém a hierarquia de tecnologia em saúde proposta por Liaropoulos (1997).

Figura 2 - Classificação de tecnologia em saúde



Fonte: Adaptação Liaropoulos (1997)

Observando a Figura 2 é possível entender que dentro da hierarquia proposta, as tecnologias médicas englobam procedimentos e tecnologia biomédica, nos quais compreende medicamentos e equipamentos. Ainda existe uma segunda classificação importante para tecnologia em saúde, de acordo com Goodman (1998) as tecnologias podem ser classificadas por: natureza, propósito e estágio de difusão. A natureza engloba medicamentos e procedimentos cirúrgicos, entre outros. O propósito compreende o tratamento e a prevenção e por fim o estágio de difusão que abarca experimentos e investigações científicas.

Para Rocha *et al.* (2008) qualquer cuidado no qual traga um bem-estar para o paciente pode ser considerado como uma tecnologia em saúde, sendo assim as principais tecnologias utilizadas no decorrer do pré-natal são: tecnologias comunicacionais no qual busca compreender e reduzir os receios da gestante e tecnologias educacionais no intuito de fornecer um entendimento melhor sobre as mudanças no corpo e a gravidez em linhas gerais. Além da caderneta de gestação, calendário de vacinação, consultas e visita a domicílio, tudo isso é fornecido normalmente pelo SUS.

Uma outra importante classificação de tecnologia em saúde foi dada por Merhy (2002) que classificou tecnologias em saúde em: leve, leve-dura e dura. As tecnologias leves descrevem relações de cuidado e vínculo com o paciente.

Tecnologias leve-dura trata-se das tecnologias direcionadas a saberes, sendo assim engloba: protocolos e normas, além de conhecimentos relativos à administração hospitalar. Tecnologia classificada como dura engloba máquinas e equipamentos.

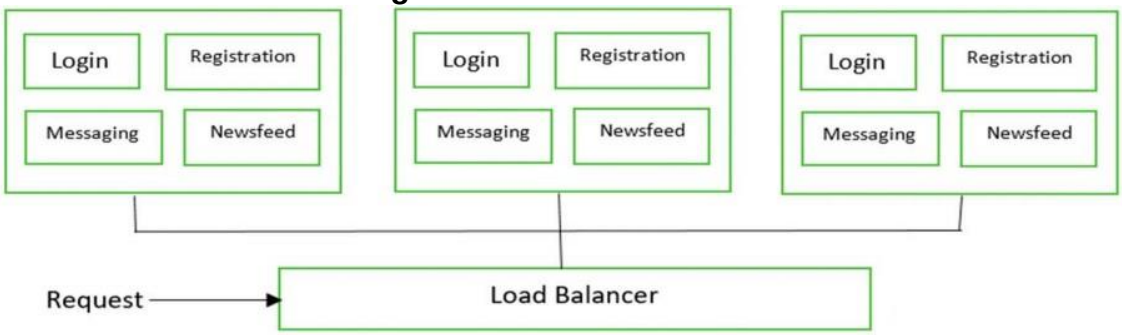
É comum encontrar na literatura médica, pesquisas relacionadas ao pré-natal que abordam a concepção e aplicação de tecnologias leves aplicadas ao pré-natal. No trabalho de Sousa (2014) foi desenvolvido um jogo educacional, com o intuito de minimizar receios e angústias vivenciadas pelas gestantes durante a gestação. O jogo trabalha com quatro temáticas, na qual aborda: trabalho de parto, parto, puerpério e cuidados com a mama. Em uma pesquisa importante, Facundo (2016) pesquisou sobre tecnologias comunicacionais com a gestante, com isso conclui que a comunicação adequada melhora a qualidade de vida das mulheres gestantes.

2.3 Arquitetura de Microsserviços

Tanto as aplicações WEB como móveis se tornaram essenciais para a sociedade, essas aplicações oferecem vários serviços imprescindíveis para as pessoas, isso inclui: *internet banking, marketplace*, redes sociais entre outros. Sendo assim essas aplicações ganharam uma grande complexidade computacional. A escalabilidade passou a ser fortemente considerada na arquitetura de qualquer aplicação moderna. De acordo com Bhatti, Bouch e Kuchinsky (2000) milissegundos que um site demore para ser carregado, gera insatisfação e faz com que usuários abandonem de forma precoce uma página WEB.

Na arquitetura de *software* tradicional, conhecida como arquitetura monolítica, toda a aplicação está dentro de um único e grande módulo. Esse modelo é ideal para aplicações que recebem um número controlado de acessos e não precisam escalar com rapidez. Em um estudo desenvolvido por Singh e Peddoju (2017) foi verificado que uma arquitetura de microsserviços apresenta melhor escalabilidade para atender um grande número de requisições, ou seja, o modelo de arquitetura de microsserviços é capaz de escalar melhor. Analisando a Figura 3 é possível identificar a complexidade de uma aplicação monolítica composta de diversos módulos que precisa escalar, a replicação de todos os módulos em diferentes servidores gera um custo elevado e possíveis problemas de *performance*.

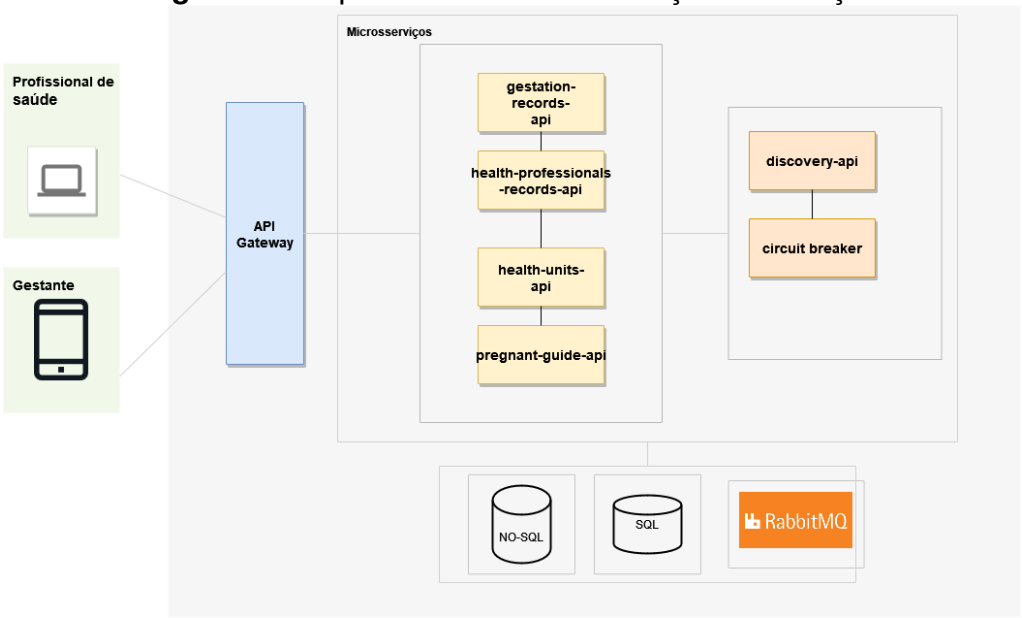
Figura 3 - Escalando o monolítico



Fonte: Singh e Peddoju (2017)

O modelo arquitetural de microsserviços segue uma abordagem bastante intrínseca, diferentemente do modelo padrão monolítico, uma arquitetura de microsserviços permite que um *software* seja escrito em pequenas partes independentes chamadas de serviço, nos quais podem ser utilizadas diferentes tecnologias e com melhor definição de responsabilidade. Nesse modelo os pequenos serviços trabalham em conjunto para entregar uma funcionalidade completa ao usuário final da aplicação. De acordo com Fowler (2014) esse modelo arquitetural facilita a reutilização de funcionalidades além de simplificar o processo de manutenção do *software*.

Figura 4 - Arquitetura de Microsserviços - Gestação



Fonte: Própria do autor

Embora esse padrão de arquitetura de microsserviços seja consideravelmente novo, a cada dia se torna mais popular e vem sendo amplamente utilizado por grandes

empresas, de acordo com Wan, Wu e Zhang (2020) as *big techs* como Facebook, Netflix, Amazon e Twitter já aderiram a esse modelo arquitetural de microsserviços. A causa disso é o fato das aplicações estarem sendo cada dia mais gerenciáveis através da nuvem. Diferente dos modelos de arquitetura monolítica onde se replicava o mesmo código em diferentes servidores para escalar a aplicação, um sistema de microsserviços permite que cada serviço da aplicação esteja implantado em um servidor diferente, isso reduz o tempo para processar uma requisição e a medida que o negócio cresce mais economia é gerada.

Na Figura 4 contém o modelo de arquitetura de microsserviços proposto nesta dissertação. Nesse modelo é possível visualizar quatro microsserviços diretamente relacionados à regra de negócio. Em outro momento será explicado a funcionalidade dos serviços relacionados ao negócio, mas também está descrito no diagrama apresentado na Figura 4, serviços comuns a maior parte das arquiteturas de microsserviços, tais como API Gateway, Discovery e Circuit Breaker.

- **API Gateway**: Em uma arquitetura de microsserviços, existem diversos serviços específicos para atender a determinadas demandas. Na plataforma gestação existem 7 serviços, a exemplo: Serviço de cadastro de gestante e serviço de cadastro de unidades de saúde. Nesse sentido é preciso centralizar todas as entradas de requisições (*Requests*) através de um único serviço e delegar ao mesmo a responsabilidade de fazer o roteamento da requisição para o serviço específico. Nesse sentido a *API Gateway* funciona como um roteador de requisições em uma arquitetura de microsserviços.
- **Service Discovery**: É um serviço de configuração com a responsabilidade de mapear de forma dinâmica todas as portas e IPs de todos os serviços de uma aplicação de microsserviços. Além disso, é responsável por gerir algumas métricas da aplicação. Caso um serviço disponibilizado pela aplicação fique indisponível será possível rastrear essa indisponibilidade através do serviço de *discovery.*
- **Circuit Breaker**: Um dos grandes gargalos dos microsserviços é a administração e manutenção de serviços espalhados pela nuvem. Normalmente na maior parte das arquiteturas que fazem a opção por esse modelo é necessário a implementação de um serviço específico para garantir a estabilidade dos demais serviços. Considerando que um pico de acesso em determinada hora do dia faça com que um serviço qualquer da aplicação fique indisponível. Nesse cenário caótico, o *circuit breaker* age de forma

a bloquear as requisições que seriam direcionadas para aquela instância que está falhando, apontando essas requisições para outros serviços até que aquele determinado serviço específico seja recuperado.

É comum que em aplicações com uma arquitetura de microsserviços se torne necessário a comunicação entre dois ou mais serviços distintos, exemplo: O serviço de cadastro de usuário precisa se comunicar com o serviço de envio de e-mail. Em vista disso é possível prover a comunicação entre os microsserviços de pelo menos duas formas distintas, em fluxos com processamento síncrono é comum que se utilize do protocolo HTTP, fazendo chamadas a *web services*, um exemplo desse modelo de comunicação é quando estamos navegando em um site de *e-commerce* e precisamos consultar o prazo de entrega da compra pelo CEP.

Em fluxos assíncronos a melhor forma de abrir comunicação entre os microsserviços é através de *broker* de mensageria. Utilizando o contexto de um *e-commerce* é comum que após o cliente finalizar uma compra demore alguns minutos para receber um e-mail de confirmação da compra, isso acorre pelo fato da compra realizada pelo cliente ter sido postada em uma fila de mensageria na qual N serviços distintos consumiram para processar a compra e só então depois de alguns minutos a depender do serviço que envia e-mail dispara para confirmar a compra realizada com sucesso ou para notificar o cliente de um possível problema.

Uma outra grande vantagem da utilização do modelo arquitetural de microsserviços é possibilidade da criação de serviços independentes, tornando assim as regras de negócios mais fáceis de implementar, haja vista que neste modelo arquitetural pode existir uma definição bem estrutura de camadas e funcionalidades, além disso permitindo que tecnologias distintas possam ser livremente usadas, sem grandes impactos. No presente trabalho, foi decidido a utilização de um modelo de microsserviços, justamente pelos motivos expressos por Singh e Peddoju (2017), a capacidade de atender um grande número de usuários e escalar. Tendo em vista a quantidade de unidades de saúde espalhadas pelo Brasil, o número de profissionais de saúde que passam de milhares e mais de milhão de gestantes grávidas anualmente é necessário um projeto complexo e capaz de atender um número considerável de requisições.

2.3.1 Mensageria

Atualmente, é comum que a indústria de desenvolvimento de *software* adote mensagerias, sobretudo no meio de soluções robustas, como é a arquitetura de microsserviços, isso ocorre pelo fato das mensageiras resolverem bem, ao menos dois grandes problemas: processamento em massa e processamento assíncrono. Em um estudo realizado por Hong, Yang e Kim (2018) foi analisado qual seria a melhor maneira de prover comunicação entre serviços diferentes, com isso foi explorado uma REST API e uma mensageria, no estudo em questão foi utilizado a mensageria RabbitMQ. Ao final do estudo foi verificado o ganho de *performance* ao utilizar o RabbitMQ para prover a comunicação entre os serviços dentro de uma arquitetura de microsserviços.

O servidor de mensageria RabbitMQ é *open source* e suporta mensagem em um protocolo chamado *Advanced Message Queuing Protocol* (AMQP), além disso é compatível com outras linguagens de programa¸c˜ao. A mensageria funciona da seguinte forma, um servidor é colocado no meio de um fluxo, no qual existe a necessidade de prover a comunicação entre duas aplicações ou dois serviços distintos. De um lado há um serviço consumidor e do outro o serviço que posta a mensagem na mensageria, sendo assim, a cada mensagem que é enviada a mensageria o consumidor é informado, esse cenário funciona como uma fila. O nome dado a esse conceito é de *Message Broker*.

Contudo existe um segundo cenário conhecido como tópico. Quando uma mensagem é enviada para um tópico na mensageria, existem vários consumidores, isso significa que há um ou muitos consumidores que precisam ser notificados pela fila.

Na arquitetura proposta neste trabalho, foi verificado a necessidade de implementar a utilização de mensagerias em fluxos que recebem um considerável número de informações e nenhuma informação pode ser perdida na rede, é o caso do fluxo de cadastro. Sendo assim, não era possível deixar em *cache* esses dados, portanto a opção foi utilizar a mensageria RabbitMQ por ser compatível com as tecnologias utilizadas para o desenvolver da solução proposta.

Outro *broker* de mensageria comumente utilizado na indústria é o Apache Kafka, durante a fase de implementação foi testado a possibilidade de sua utilização, contudo houve alguns problemas de integração com as ferramentas utilizadas e dessa forma foi feita

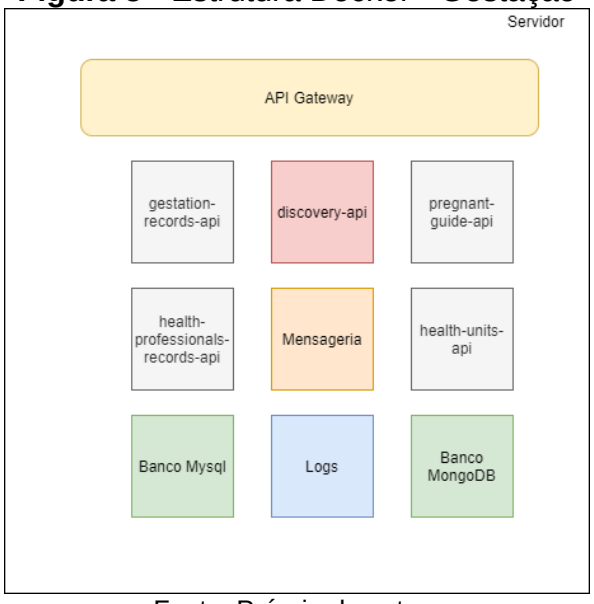
a opção pelo RabbitMQ. Maiores detalhes sobre o *broker* será apresentado na secção 5.4.

2.3.2 Docker

Atualmente a plataforma *Docker* vem sendo cada vez mais usada na indústria, de acordo com Gerber (2015), é crescente o número de novos usuários que optaram por usar essa tecnologia nos últimos tempos. O *Docker* permite criar, executar e publicar *containers* de maneira virtualizada, a grande vantagem do docker está na sua arquitetura, que permite a criação de Máquinas Virtuais (VM) leves, diferente do modelo de VM tradicional. Isso possibilita mais flexibilidade para escalar e consequentemente fazer o *deploy* de aplicações. Além do mais, o docker vem sendo amplamente utilizado para ajudar equipes de desenvolvimento a criar e compartilhar *containers*, de forma que o desenvolvedor possa conseguir facilmente executar o código do outro desenvolvedor em seu ambiente local (ANDERSON, 2015).

De acordo com Cito, Ferme e Gall (2016) os contêineres permitem configurar todo um ambiente computacional, isso inclui dependências necessárias tal como: Bibliotecas ou *frameworks* com isso o docker gera uma imagem. É essa imagem que contém todas as configurações de um ambiente que pode ser replicada facilmente em outros servidores. É possível instalar qualquer coisa dentro de um container, pois cada um tem sua própria interface de rede.

Figura 5 - Estrutura Docker - Gestação



Fonte: Própria do autor

Neste trabalho foi feita a opção por se trabalhar com *docker*, justamente pela sua flexibilidade e rapidez para realização de *deploys*. Todos os serviços estão dentro de *containers* específicos, totalmente isolados, conforme é possível observar na Figura 5. Para tornar o processo mais simplificado, toda a aplicação desenvolvida foi mapeada em um arquivo *docker compose* conforme algoritmo 1:

Algoritmo 1 - *Docker compose* para subir o serviço *gestation-records-api*

```
1:  procedure GESTATION-RECORDS-API:
2:  image : gestation − records − api
3:  hostname : gestation − records − api
4:  ports :
5:  −8082 : 8082
6:  links :
7:  −discovery − api
8:  −db
9:  depends − on :
10: −discovery − api
11: −db
```

Fonte: Própria do autor

No Algoritmo acima é possível verificar a criação de um container chamado *gestation- records-api*, da imagem desse container, ou seja, a aplicação. Nesse *script* também foi definido o *hostname* para esse container e a porta que ele irá subir, tanto a porta interna como a externa, nesse caso, são iguais: 8082. Além disso, é preciso informar ao *docker* que o *container* em questão só poderá subir após o *container* de *discovery-api*. Isso porque a aplicação de *discovery* que está dentro de um outro *container*, tem o objetivo de mapear os *Internet Protocol* (IP's) em tempo de execução, conforme visto anteriormente. Sendo assim basta executar um simples comando (*Docker run*) que toda a aplicação estará disponível.

2.3.3 Banco de Dados

No modelo arquitetural de microsserviços existe a possibilidade de adotar um banco de dados por serviço ou um único banco de dados para atender todos os serviços. Não existe certo ou errado, isso varia de acordo com cada negócio. Diferentemente da arquitetura monolítica, em uma arquitetura de microsserviços ainda é possível adicionar a uma aplicação, diferentes modelos de banco de dados, ou seja, é possível usar um banco relacional como também banco não-relacional. Para

Munonye e Martinek (2020) existem ao menos três formas distintas para armazenamento de dados no contexto de microsserviços: banco por serviço, banco de dados compartilhados entre muitos serviços e CQRS *(Command Query Responsibility Segregation)*, no qual utiliza uma abordagem de salvar dados baseado em eventos e comandos.

2.3.4 Observabilidade e Monitoramento

Com as aplicações WEB e *mobile* cada vez mais complexas e com a necessidade de ser escalável, surgiu a necessidade de adotar cada vez mais a utilização de arquiteturas de microsserviços na indústria, corroborando com Marie-Magdelaine, Ahmed e Astruc-Amato (2019). Sendo assim é essencial o monitoramento de um sistema em *cloud*, isso por diversos motivos, tais como: evitar custos desnecessários, verificar e compreender como se comportam as diversas camadas da aplicação para com isso evitar indisponibilidade. Contudo existe uma grande complexidade envolvendo os mecanismo de observalidade e monitoramento, de acordo com Géhberger, Mátray e Németh (2016) uma aplicação em *cloud* normalmente é dividida em inúmeros módulos, é o caso da aplicação proposta por esta pesquisa, essa divisão torna complexo os cálculos independente do algoritmo utilizado, tendo em vista a vasta divisão de módulos.

Até pouco tempo, as aplicações WEB possuíam um nível de monitoramento notadamente simples quando comparado as aplicações em *cloud*, envolvendo apenas sinais de alertas caso a aplicação fosse indisponibilizada por algum motivo. Hoje com o advento da computação em nuvem, o conceito de observalidade e monitoramento passou a incluir: alarmes; recuperação de dados; avaliação de *datasources* e métricas de escalonamento. Um *software* comumente utilizado para esse intuito é o Grafana, uma plataforma aberta para análise e monitoramento de aplicações. Para Picoreti *et al.* (2018) hospedar uma aplicação em nuvem permite o provisionamento da infraestrutura sob demanda além da economia a longo prazo.

2.4 Usabilidade

Uma das principais características de qualidade em um *software* é a usabilidade, de acordo com Yusop (2015) a usabilidade está diretamente relacionada

a aceitação de um *software* no mercado competitivo da atualidade. Para garantir a usabilidade de um *software* é necessário se atentar a vários fatores, isso vai além de projetar uma interface gráfica interativa. O *software* precisa ser transparente, consistente e sobretudo eficaz. Para Nielsen (1994b) um *software* precisa ser fácil de aprender, simples de usar, eficiente e permitir a rápida recuperação de erros, além disso o usuário deve ter facilidade de lembrar das funcionalidades.

Conforme Jerome e Kazman (2005) na indústria de desenvolvimento de *software* muitas empresas ainda desenvolvem sistemas sem envolver especialista em Interação Humano Máquina, isso é um tanto problemático, pois os engenheiros de *software* acabam por não considerar muitos aspectos durante o projeto e concepção do *software*. Muitas vezes a correção de um problema de usabilidade custa caro, pois a solução envolve, a depender do cenário, alteração na arquitetura de uma aplicação, sendo assim é importante efetuar análise e aplicar técnicas de usabilidade e desenvolver uma aplicação baseada no usuário desde a concepção até a entrega final.

Nas tecnologias em saúde a usabilidade é essencial para garantir a qualidade de vida e a segurança dos pacientes. Para Broekhuis, Velsen e Hermens (2019) projetar software para saúde é um tanto complexo do ponto de vista do projeto de interface:

> A usabilidade na eHealth difere de outros domínios em vários aspectos. Em primeiro lugar, a satisfação do usuário com um sistema de eHealth é difícil de estabelecer. Enquanto o e-commerce seduz os clientes com mensagens pessoais que se ajustam perfeitamente às suas necessidades e, assim, tentam aumentar a satisfação do usuário com o sistema, para a eHealth os usuários precisam ser informados sobre os efeitos positivos e negativos de seu comportamento de saúde. Isso significa que os usuários às vezes precisam ouvir conselhos que não querem ouvir, o que pode influenciar a satisfação do sistema. (BROEKHUIS; VELSEN; HERMENS, 2019).

Na maior parte das vezes o público leigo (não inclui profissionais de saúde) que utiliza uma aplicação voltada para a saúde, tem um grau de alfabetização distinto, conforme Chew, Bradley e Boyko (2004), isso traz um outro grande desafio, projetar uma interface capaz de permitir que usuários com grau escolar diferente consigam se adaptar a um *software* e executar as atividades corretamente. O estudo da usabilidade é essencial para aplicações de maneira geral, pois previne erros e muitas vezes é essencial para engajar um usuário a um *software*, com isso é extremamente importante, projetar uma interface em cima de boas práticas e sobretudo realizar testes de usabilidade contínuo ao desenvolvimento de um produto, buscando com isso

atingir a satisfação dos usuários.

2.5 Requisitos

Ao desenvolver qualquer projeto de *software* é muito importante focar na elicitação dos requisitos, tendo em vista a importância de tal para o correto funcionamento de um sistema. Atualmente está muito presente em grandes projetos, a subárea da engenharia de *software* denominada de Engenharia de Requisitos, responsável por documentar e gerenciar requisitos durante toda a vida útil de um *software*. Sendo assim foi necessário mapear todos os requisitos necessários para o projeto presente obter êxito, tendo em vistas as condições determinadas para o pleno funcionamento.

Para Sommerville (2011) os processos de Engenharia de Requisitos são separados em quatro grandes áreas: Viabilidade, Elicitação, Especificação e Validação. Antes de iniciar um novo projeto, é extremamente importante fazer uma análise, buscando identificar se realmente é viável aquela solução e se é possível de ser implementada. O processo de elicitação busca compreender as necessidades dos usuários. Na especificação é realizada de fato a descrição dos requisitos e por fim, tudo precisa ser validado e testado, garantindo assim uma entrega de qualidade.

No presente projeto, a elicitação de requisitos e especificação dos requisitos foi elaborada com base no Referencial Bibliográfico. Inúmeros documentos e pesquisas focadas na gestação foram considerados para compreender o contexto da gestação e as principais dificuldades enfrentadas pelas gestantes. Posteriormente esses requisitos já especificados e refinados foram transformados em histórias de desenvolvimento e então desenvolvidos.

2.6 Principais Requisitos

Nesta seção serão apresentados os principais requisitos funcionais e não funcionais. Os requisitos foram extraídos com base nas pesquisas citadas anteriormente neste referencial bibliográfico, sobretudo na caderneta da gestante do Ministério da Saúde (2006), além de uma entrevista com uma profissional de saúde com experiência em pré-natal.

2.6.1 Requisitos Funcionais

- **Cadastro de Unidade de Saúde**: Usuários com as devidas autorizações deverão conseguir realizar o cadastro de unidade de saúde, além de alterar dados de unidades já existentes. Na tabela 1 pode ser verificado os dados necessários para o cadastro de uma unidade de saúde tal como o seu respectivo tipo.

Tabela 1 - Dados da Unidade de Saúde Solicitados

	Nome do Atributo	**Tipo**
1	Nome	String
2	Endereço	Address
3	Latitude	String
4	Longitude	String
5	Nº de Registro	Float

Fonte: Própria do autor

- **Cadastro de Profissionais de Saúde**: Usuários com perfis master e administrador deverão conseguir cadastrar profissionais de saúde para determinada unidade de saúde, além disso, poderá editar dados de usuários existentes para a unidade de saúde na qual o mesmo seja administrador a nível de sistema. Na tabela 2 pode ser verificado os dados necessários para o cadastro de um profissional de saúde.

Tabela 2 - Cadastro de um profissional de saúde

	Nome do Atributo	**Tipo**
1	Nome	String
2	Nº de Registro	Float
3	Endereço	Address
4	CPF	Float
5	Data de nascimento	Date
6	Senha de login	String

Fonte: Própria do autor

- **Cadastro de Gestante**: O sistema deve estar aberto a receber novos cadastros de mulheres. Na tabela 3 pode ser verificado os dados necessários para o cadastro de uma gestante.

Tabela 3 - Dados da Gestante

	Nome do Atributo	Tipo
1	Nome	String
2	Data de Nascimento	Date
3	CPF	Float
4	Ocupação	String
5	Estado civil	String
6	Religião	String
7	Escolaridade	String
8	Endereço	Address
9	E-mail	String
10	Senha de login	String

Fonte: Própria do autor

- **Guia da gestante**: As gestantes devem enviar dúvidas relacionada à gestação via aplicativo móvel a qualquer momento.
- **Dúvidas comuns**: Deverá existir uma seção dentro do aplicativo móvel na qual as usuárias possam encontrar respostas a questionamentos realizados por elas ou por outras gestantes.
- **Tarefas da gestação**: Cada gestante deverá conseguir consultar no aplicativo as tarefas prescritas por um profissional de saúde habilitado.
- **Solicitar visita *in loco***: Por determinação do Ministério da Saúde (2006) a gestante tem o direito a receber um atendimento domiciliar. Dentro do aplicativo deverá ter um botão de ação com a solicitação da visita.

2.6.2 Requisitos não Funcionais

A tecnologia a ser desenvolvida deve funcionar em aparelhos com um *hardware* limitado, independentemente do Sistema Operacional (SO) isto é: aparelho com pelo menos 1.2 GHz de processamento, com um processador mínimo *dual core*, 1 GB de memória RAM e com conectividade através das redes, 3G WCDMA e 4G, além disso atender gestantes que possuem um pacote de internet mínimo, conexão 3G. Todas as informações inseridas no aplicativo pelas gestantes é sincronizado em tempo real com um sistema WEB, utilizado por profissionais de saúde, sendo assim é possível prevenir eventos adversos relacionados à gestação, trazendo maior qualidade de vida para a gestante e o bebê.

2.7 Considerações do Capítulo

Neste capítulo, foi explanado questões importantes sobre a gestação e as tecnologias adotadas pelo SUS. Além disso, foi apresentado pontos importantes sobre a arquitetura de microsserviços, tais como: banco de dados, escalabilidade e observabilidade, por fim foi tratado o conceito de usabilidade, ponto chave para o desenvolvimento de qualquer interface e apresentado os requisitos da plataforma proposta.

3 TRABALHOS RELACIONADOS

Neste capítulo serão apresentados os trabalhos relacionados a esta pesquisa, foram selecionados trabalhos relevantes sobre a problemática que envolve o desenvolvimento de aplicativos voltados para atender a mulher gestante. Inicialmente serão apresentados os critérios utilizados para a seleção dos trabalhos, em seguida será feita uma discussão sobre cada um deles.

3.1 Critérios utilizados

Atualmente a maior parte das aplicações *mobile* para a saúde são focadas na gestação (SMITH *et al.*, 2017), sendo assim é importante observar a usabilidade, *interface* e a arquitetura dessas aplicações, sobretudo para entender como é o comportamento dessas aplicações e como os usuários interagem. Sendo assim, o principal objetivo desse capítulo é analisar as tecnologias utilizadas para apoiar as mulheres na fase gestacional e os seus respectivos impactos. Foram empregados alguns critérios para a seleção nas bases ACM *digital library* e IEEE *xplore*:

- O estudo foi localizado através da combinação das seguintes palavras-chave: *performance, design, architecture, software, patient well-being, pregnancy and prenatal*.
- O trabalho aborda uma solução, técnica ou metodológica que relaciona arquitetura, *design* ou usabilidade de uma aplicação.
- O estudo está direcionado para uma solução de problema relacionado a um *software* médico para a saúde.
- O estudo está disponível para acesso.

3.2 Estudos selecionados

Nesta subseção serão apresentados os estudos encontrados na literatura que apresentem uma solução tecnológica para acompanhamento da gestação. Diante dos estudos apresentados será avaliado a agregação de novas funcionalidades para dentro da aplicação proposta pelo autor desta dissertação.

Quadro 1 - Quadro de trabalhos relacionados

Trabalho	Abordagem /Escopo	Alvo	Plataforma	Arquitetura	Escalabilidade	Usabilidade
Wicahyono *et al.* (2019)	Suporte pré-natal	Gestante /Profissionais de Saúde	Android	Monolítica	Não mencionado	Sim
Wiweko *et al.* (2018)	Educacional	Gestante	Web	Não mencionado	Não mencionado	Sim
Efrem *et al.* (2019)	Suporte pré-natal	Gestante /Profissionais de Saúde	Android	Não mencionado	Não mencionado	Sim
Smith *et al.* (2017)	Educacional	Gestante	Web	Não mencionado	Não mencionado	Sim
Souza (2022) Presente Trabalho	Suporte pré-natal	Gestante /Profissionais de Saúde	Android/ Web	Microsserviços	Sim	Sim

Fonte: Própria do autor

Todas as pesquisas conforme exposto no quadro 1 utilizaram conceitos de usabilidade com o intuito de melhorar a experiência da usuária. É importante sobretudo em aplicações de saúde que seja dada a devida atenção à usabilidade, como forma de prevenir erros. A diferença do presente trabalho para os demais, é que houve uma preocupação em modularizar o sistema de tal forma que as requisições solicitadas pelas as usuárias possam ser respondidas de forma rápida, evitando latência. Nenhuma pesquisa listada no quadro 1 propôs uma arquitetura escalável, nem mesmo foi abordado quão escaláveis as aplicações poderiam ser. Desta forma o presente trabalho também se diferencia, tendo em vista que a definição de arquitetura utilizada foi tomada visando a escalabilidade da aplicação.

3.2.1 Trabalho 1: Pregnancy Monitoring Mobile Application User Experience Assessment

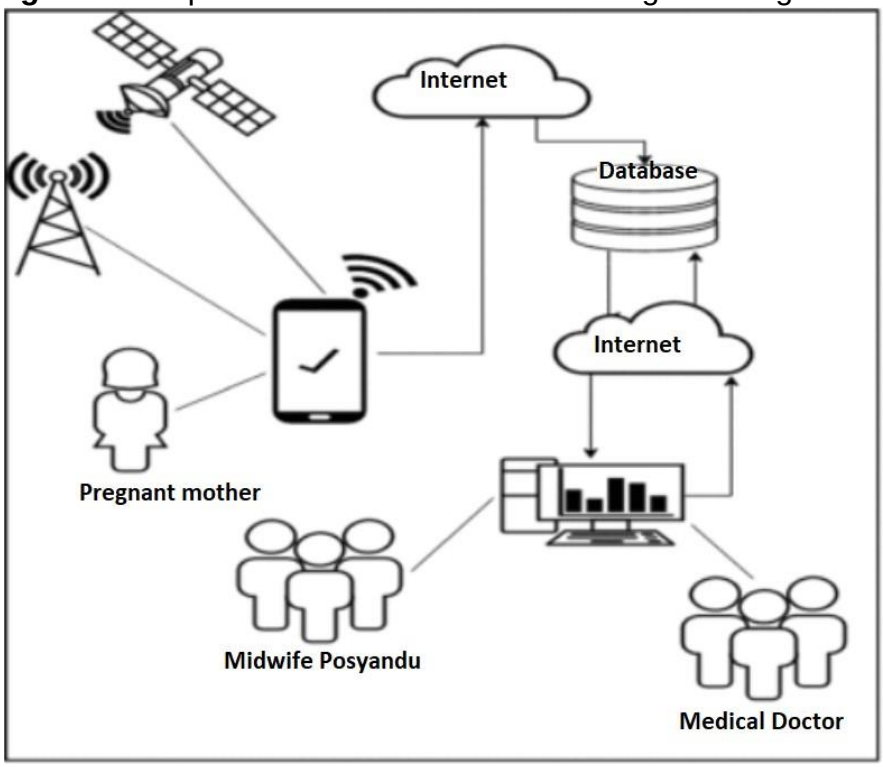
A pesquisa desenvolvida por Wicahyono *et al.* (2019) propôs um aplicativo móvel para ser utilizado pelas mulheres durante o pré-natal com o intuito de capturar alguns parâmetros das gestantes e enviar para uma unidade de saúde, entre os parâmetros nos quais o aplicativo proposto buscava extrair, estavam a mobilidade da gestante e o peso. Em um segundo momento, os pesquisadores investigaram a experiência da usuária na utilização da tecnologia.

A pesquisa foi desenvolvida na Indonésia e foi dividida em quatro etapas:

- Desenvolvimento do aplicativo;
- Implantação da tecnologia;
- Avaliação;
- Análise de dados gerados pelo aplicativo.

O *front-end* e o *back-end* do aplicativo foram validados através de testes caixa branca e caixa preta[1]. Foi realizado um estudo piloto com 30 gestantes, após o estudo os pesquisadores aplicaram um questionário para verificar a experiência de uso da aplicação. A figura 6 contém a arquitetura da aplicação de monitoramento inteligente da gestante desenvolvido por Wicahyono *et al.* (2019). Os *stakeholders* envolvidos são as gestantes, centros de saúde e profissionais de saúde. Os dados são centralizados em um banco de dados na nuvem.

Figura 6 - Arquitetura de monitoramento inteligente da gestante



Fonte: Wicahyono *et al.* (2019)

O aplicativo foi desenvolvido na plataforma Android e utiliza API *Mobile* Google *Vision* que captura dados em tempo real de localização da gestante e sincroniza esses dados em um banco de dados centralizado. O questionário aplicado após o período de validação foi respondido por 30 gestantes, com isso foi verificado que inúmeras melhorias precisam ser desenvolvidas dentro do aplicativo, entre as quais: a usabilidade e a eficiência que foram os pontos mais negativos na percepção das usuárias.

[1] No teste caixa-preta o testador não se preocupa com a estrutura interna do sistema, apenas na sua funcionalidade. No teste caixa-branca os casos de testes são gerados de acordo com a estrutura na qual o *software* foi modelado.

3.2.2 Trabalho 2: Jakpros: Reproductive Health Education Application for Pregnant Women

No estudo desenvolvido por Wiweko *et al.* (2018) foi proposto um aplicativo móvel chamado de Jakpros (*Jakarta Reproduksi Sehat*) com a finalidade de fornecer informações confiáveis sobre gestação para as mulheres grávidas. O estudo aconteceu em Jacarta, na Indonésia. Foram selecionadas 126 mulheres grávidas, foi verificado inicialmente o conhecimento delas relacionado ao tema: saúde reprodutiva e posteriormente foram feitas orientações às mulheres sobre como utilizar o aplicativo e foi solicitado que as mesmas utilizassem durante 2 semanas.

O aplicativo Jakpros foi desenvolvido em uma plataforma híbrida utilizando CSS, HTML5 e *Angular JS* de acordo com (WIWEKO *et al.*, 2018). Atualmente o aplicativo está disponível apenas para aparelhos com sistema operacional *Android*, porém pelo fato dos pesquisadores utilizarem uma tecnologia de desenvolvimento híbrida o aplicativo pode ser extensível para a WEB ou outras plataformas móveis. Ao final do estudo foi observado que o aplicativo Jakpros teve um proveito significativo em colaborar com a aprendizagem das gestantes e concluíram que a utilização do aplicativo teve papel importante em conscientizar as mulheres sobre a gestação e prevenir sobre problemas indesejados.

3.2.3 Trabalho 3: An emerging model of maternity care: smartphone, midwife, doctor?

No trabalho desenvolvido por Tripp *et al.* (2014) foi averiguado que os *smartphones* são amplamente utilizados na gravidez como fonte de informação. Com isso, o trabalhou procurou descrever a natureza dos aplicativos relacionados a gravidez e assim identificar se estes poderiam afetar de alguma forma os cuidados com a maternidade. Os pesquisadores analisaram aplicativos de saúde disponibilizados nas plataformas *Android* e *Apple* e selecionaram os que abordasse o tema da gestação e gravidez, além disso tivesse uma boa popularidade.

Foram analisados 497 aplicativos voltados para a gestação, desses 49% eram informativos, 13% interativos e 19% tinham alguma ferramenta médica e os demais eram aplicativos de mídia social. Os aplicativos com melhor avaliação foram os

interativos.

Dessa forma os pesquisadores concluíram que aplicativos relacionados a gravidez podem indicar uma mudança no comportamento e sobretudo empoderamento do paciente na prestação de cuidados e o modelo tradicional de assistência à maternidade precisa englobar os dispositivos eletrônicos, portanto é necessário que os profissionais médicos e autoridades políticas estejam cientes da necessidade de incorporar aplicativos eletrônicos no processo de pré-natal como auxílio às gestantes.

3.2.4 Trabalho 4: Designing an app for pregnancy care for a culturally and linguistically diverse community

O estudo conduzido por Smith *et al.* (2017) se preocupou em desenvolver um aplicativo que tivesse uma boa usabilidade, sobretudo para mulheres sem instrução. O trabalho foi dividido três partes:

- Oficina de *design* com profissionais de saúde e gestantes. Foi abordado temas importantes da gestação e proposto uma série de requisitos funcionais.
- Desenho e desenvolvimento de protótipos.
- Entrevistas estruturadas com profissionais de saúde e gestantes. Sobre *design* do aplicativo.

Dentro do aplicativo há orientações e desafios sobre amamentação e exercícios para o assoalho pélvico. Além disso, existem combinações de crenças e questões culturais. Ao final do estudo os pesquisadores verificaram que o aplicativo apresentou uma boa aceitabilidade com a interface e o sucesso nesse aspecto aconteceu pelo fato de ter havido antes da prototipação da interface uma oficina com gestantes e profissionais de saúde. O aplicativo fornece um meio seguro e eficaz de melhorar o entendimento das mulheres sobre saúde, e foi positivo no desfecho da gestação. As usuárias que fizeram o uso durante o estudo tiveram um desfecho positivo no parto.

3.2.5 Trabalho 5: Mobile Application Helps Planning Activities during Pregnancy

A pesquisa de Efrem *et al.* (2019) descreve o aplicativo ELISA que está em

fase de desenvolvimento. O propósito do aplicativo é de trazer informações aos pais, desde o momento que eles identificarem a gravidez até o parto. O aplicativo também auxilia no planejamento das tarefas críticas relativas à gravidez. O protótipo do aplicativo foi desenvolvido após entrevistas semiestruturadas com os profissionais da saúde e mulheres gestantes. Usando o aplicativo o usuário pode listar as principais atividades relativas à gestação tais como: reunião com grupo de gestante e consulta médica agendada. Uma outra seção do aplicativo permite que as gestantes possam fornecer dados clinicamente relevantes, como peso e altura. A aplicação ainda permite que as gestantes enviem perguntas a médicos relacionadas à gestação e a marcação de consultas. Os pesquisadores acreditam que ao final do desenvolvimento deste trabalho haverá grande impacto na saúde das mulheres, auxiliando e facilitando a vida das gestantes.

3.3 Considerações do Capítulo

Neste capítulo foram apresentados alguns trabalhos relacionados a este projeto. Foi discutido soluções tecnológicas aplicadas ao pré-natal e como essas soluções impactaram de forma positiva a gestação. Dessa maneira foi possível verificar que algumas soluções propostas tiveram um significativo impacto para conscientizar as gestantes, apresentando uma solução focada na gestação, buscando assim, fornecer apoio educativo para elucidar dúvidas e receios vinculados a gravidez (WICAHYONO *et al.*, 2019) e (WIWEKO *et al.*, 2018).

Outro aspecto relevante considerado nas pesquisas analisadas foi a usabilidade. No trabalho de Smith *et al.* (2017) houve um cuidado no projeto de interface visando dessa forma construir uma interface que pudesse ser fácil de usar e atendesse diferentes perfis de usuárias, sobretudo aquelas usuárias com pouca familiaridade com tecnologia e sem instrução. Na pesquisa de Wicahyono *et al.* (2019) houve uma avaliação da interface, contudo na conclusão do trabalho os pesquisadores concluíram que ainda existiam muitas lacunas que precisavam ser melhoradas no aspecto da usabilidade.

A maior parte dos trabalhos analisados não adentraram em detalhes técnicos relacionados ao desenvolvimento das aplicações. Wicahyono *et al.* (2019) ainda citou que foram realizados testes caixa branca e caixa preta e expôs em linhas gerais um diagrama representando a arquitetura desenvolvida, conforme é possível visualizar na

Figura 6. No trabalho de Wiweko *et al.* (2018) foi utilizado uma plataforma híbrida de desenvolvimento, no qual os autores citam a utilização do *angular js* para o desenvolvimento, contudo atualmente já existe formas mais eficazes de desenvolvimento de aplicação híbridas, que convertem o código fonte para nativo, fazendo com que as aplicações ganhem em *performance*, a exemplo: *react native e flutter*.

Embora todas as pesquisas analisadas possuam impactos significativos e funcionalidades relevantes sobretudo a pesquisa desenvolvida por Efrem *et al.* (2019), nenhuma pesquisa apresentou uma solução realmente escalável para ser adotada em uma cidade ou um estado, todas as soluções foram testadas com poucas usuárias e não foi em nenhum mo- mento discutido estratégias de escalonamento. A proposta dessa dissertação é justamente apresentar uma arquitetura de microsserviços, com capacidade de escalar para atender um grande número de usuárias, além disso neste projeto foi bem delineado as técnicas de *deploy*, conteinerização de serviços, testes automatizados, testes de integração, mensagerias e banco de dados relacional e não-relacional.

Dessa maneira o projeto desta dissertação possui similaridade de funcionalidades com as demais pesquisadas relacionadas, contudo a tecnologia utilizada é distribuída, ideal para funcionar em *cloud*, sendo que não foram encontrados até esta data na literatura nenhuma pesquisa que aborda o desenvolvimento de uma aplicação de microsserviços para atender gestantes. Desse modo, este trabalho apresenta um grande diferencial na escalabilidade e uma limitação: o custo. Tendo em vista uma aplicação de microsserviços necessita de mais processamento computacional para funcionar corretamente.

4 MATERIAIS E MÉTODOS

Neste capítulo serão apresentados os principais passos realizados para delinear a metodologia utilizada. Sendo assim foi dividido da seguinte forma: métodos utilizados, *scrum*, controle de versionamento de código e requisitos.

4.1 Metodologia utilizada

O desenvolvimento de um *software* precisa seguir um processo bem definido de engenharia de *software* para assim garantir a eficácia, eficiência, prazos, custos e outros aspectos de qualidade. De acordo Jain, Sharma e Ahuja (2018) nos últimos anos, muitas entregas de *software* falharam no processo de engenharia de qualidade e com isso nem conseguiram chegar à fase final de desenvolvimento, consequentemente nunca foram disponibilizados para os usuários finais.

As metodologias de desenvolvimento de *softwares* tradicionais não remedeiam as exigências atuais do mercado para desenvolver *softwares* com alta qualidade. Conforme Sommerville (2007) as metodologias ágeis surgem da necessidade de desenvolver *softwares* mais adaptativos e que permita mudanças de escopo no projeto de desenvolvimento, gerando um impacto mínimo tal como acelerando as entregas de produtos funcionais aos *stakeholders*.

As metodologias ágeis propõem simplificar todo o processo de desenvolvimento de *software*, tornando essas etapas de desenvolvimento mais dinâmicas, sólidas e interativas. Durante as etapas de construção do *software* as melhorias são contínuas, as equipes de forma geral, incluindo o cliente, se comunicam de forma clara, corrobora (MANUJA; MANISHA, 2014). Além disso, as entregas são rápidas e o projeto se torna flexível a eventuais mudanças de escopo.

De acordo com Beck e Bennekum (2001) o manifesto ágil é guiado com base em 12 princípios: Entrega contínua, mudanças são sempre aceitas, satisfação do cliente, o time desenvolve junto a solução, pessoas motivadas projetos bem sucedidos, agilidade e integração rápida, entregas de *software* funcional, desenvolvimento constante, procurar sempre melhorar a qualidade técnica, simplicidade, times auto-gerenciáveis e afetividade.

Nesta pesquisa foi feita a utilização do método ágil Scrum, no qual o processo é incremental e iterativo. De acordo com Schwaber e Sutherland (2013) o Scrum é um

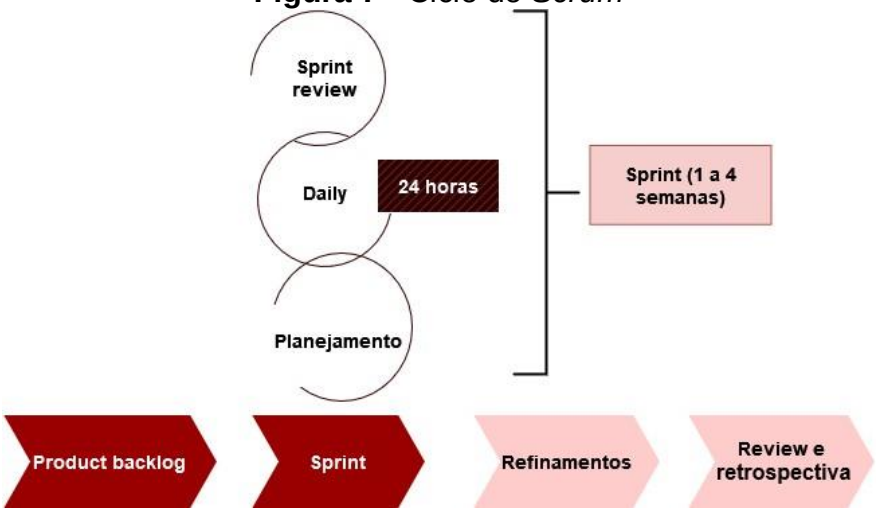
framework desenvolvido com a finalidade de solucionar problemas complexos, desenvolvendo e entregando um produto ao cliente final com um alto valor e uma grande qualidade. Segundo Srivastava, Bhardwaj e Saraswat (2017) um importante benefício do Scrum é a produtividade por meio da comunicação e do planejamento que proporcionam liberdade às equipes para descobrir maneiras de desenvolver soluções.

4.1.1 Scrum

A transparência dos processos, como também a inspeção e adaptação são vertentes muito fortes dentro do *framework* Scrum. Na figura 7 abaixo é possível entender de forma visual o fluxo de execução do Scrum. Dentro do *Scrum* existe a definição de três papéis principais:

- ***Product Owner***: Responsável por definir o produto, esse papel representa o cliente, ou seja, conhece com profundidade sobre o negócio e o produto no qual será desenvolvido.
- ***Scrum Master***: É o facilitador, apoia outros papéis em busca de facilitar a comunicação e remover possíveis impedimentos para se chegar a solução esperada pelo cliente.
- **Time de desenvolvimento**: A equipe responsável pelo desenvolvimento do produto, dentro desse time está incluído os arquitetos, desenvolvedores e testes QA (*Quality Assurance*).

Figura 7 - Ciclo do *Scrum*



Fonte: Adaptação Srivastava, Bhardwaj e Saraswat (2017)

No *framework Scrum* existe a definição de quatro eventos principais:

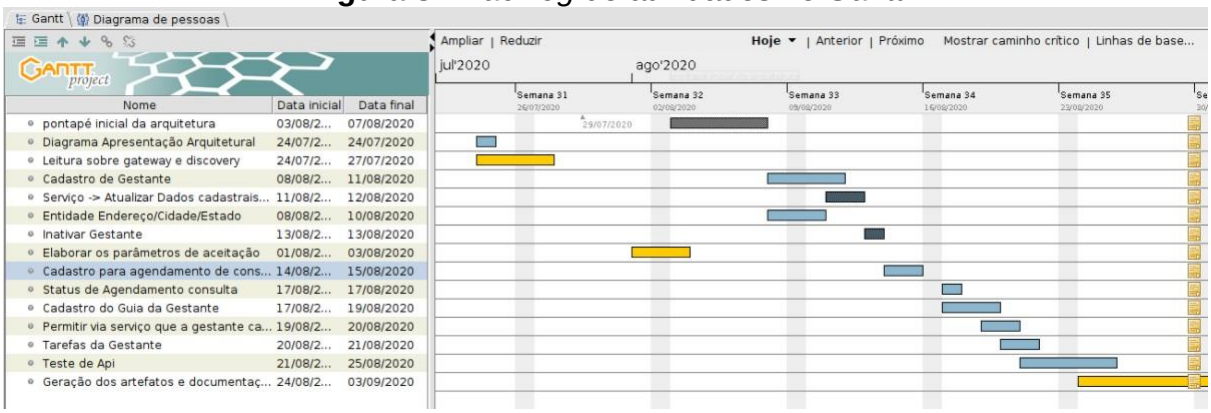
- ***Sprint planning***: No início de cada etapa de desenvolvimento, também chamada de *Sprint*, que normalmente tem duração de duas a quatro semanas, o time planeja quais as atividades precisarão ser entregues para após a finalização da *Sprint* o cliente ter um produto funcional e com alto valor.
- ***Sprint review***: No final de cada etapa de desenvolvimento (*Sprint*) é feita uma reunião chamada de *review* onde são apresentados a todos os papéis do *Scrum* os resultados do produto desenvolvido, com isso é aberta uma rodada de discussão para procurar melhorar ainda mais o produto.
- ***Sprint Retrospectiva***: É o momento onde os papéis debatem abertamente sobre o que deu certo e o que poderia ser melhorado.
- ***Daily***: Reunião diária com poucos minutos de duração, onde o time responde quais as atividades realizou ontem e quais irão realizar hoje e relatam sobre possíveis empecilhos para entregar as atividades.

O *Scrum* ainda define três artefatos e cinco valores: coragem, foco, comprometimento, respeito e abertura a mudanças. Na figura 7 é possível ver os trˆes artefatos definidos no *framework*. O *Product Backlog* reflete a lista de atividades que precisam ser desenvolvidas para a entrega do produto final, essas atividades precisam ser bem definidas e refinadas para que todos compreendam, sobretudo o time de desenvolvimento e consiga entregar ao final da *Sprint*. A *Sprint Backlog* representa uma etapa, um ciclo de desenvolvimento que pode durar até quatro semanas para ser concluído e conforme Schwaber e Sutherland (2013) o terceiro artefato do *Scrum* é o incremento:

> O incremento é a soma de todos os itens do Backlog do Produto completados durante a Sprint e o valor dos incrementos de todas as Sprints anteriores. Ao final da *Sprint* um novo incremento deve estar “Pronto”, o que significa que deve estar na condição utilizável e atender a definição de “Pronto” do Time Scrum. Este deve estar na condição utilizável independente do Product Owner decidir por libera-lo realmente ou não. (SCHWABER; SUTHERLAND, 2013, p.15).

Dessa maneira todo o desenvolvimento deste trabalho foi gerenciador com base no *framework Scrum*, o desenvolvimento foi dividido em *sprints* com duração de 30 dias cada uma delas. Normalmente, as reuniões de acompanhamento aconteciam uma vez por semana e nesses momentos eram definidos os próximos passos para o desenvolver da plataforma, como também era alinhando novos planejamento e estratégias transparentes e objetivas para alcançar o resultado final.

Figura 8 - *Backlog* de atividades no *Gantt*



Fonte: Própria do autor

Os principais eventos do Scrum foram devidamente seguidos, ao iniciar uma nova *Sprint* era então realizado uma *planning* onde era analisado o *backlog* de atividades e então era feita a priorização. Para ajudar com a organização do *backlog* e manter as entregas foi utilizado uma ferramenta *open source* chamado *Gantt*, na figura 8 é mostrado um exemplo de sua utilidade.

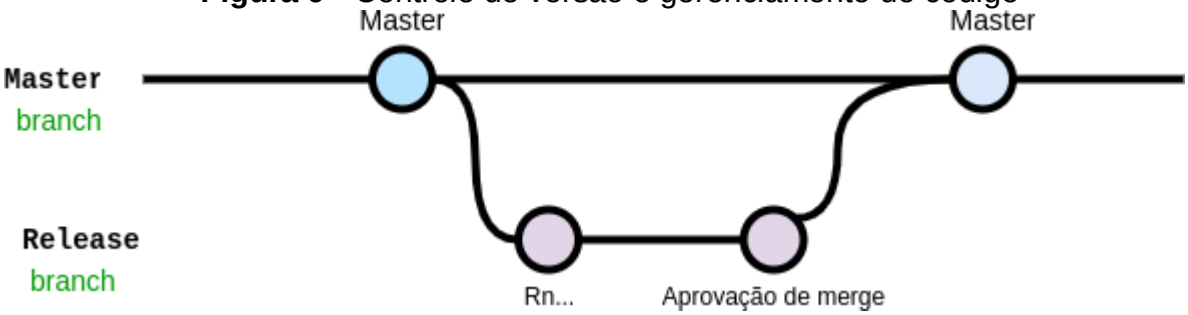
4.1.2 Versionamento de software

As metodologias ágeis trouxeram inúmeros benefícios para a engenharia de *software*, atualmente o processo de desenvolvimento e evolução de um *software* acontece de forma muito acelerada, com isso surgiu a necessidade de criar boas práticas para disponibilizar esse *software* para os usuários finais. Ou seja, quando uma alteração é feita em um *software* que já foi disponibilizado anteriormente em um servidor para os usuários finais, isso quer dizer, um software que já está no ambiente de produção. É necessário que essa atualização possa ser tão logo disponibilizada para os usuários, com isso é importante que seja efetuado um (*Deploy*) com segurança, para conter a modificação realizada. Esse processo por muito tempo foi custoso e arriscado. Por isso surgiu um conceito na engenharia de software, comumente utilizado pela indústria conhecido como *DevOps*.

De acordo com Macarthy e Bass (2020) o termo *DevOps* se refere a uma cultura e filosofia da engenharia de software, que utiliza equipes multifuncionais para criar *software* e implantar em servidores para usuários finais de forma muito mais rápida e segura. Para Kim e Willis (2016) as boas práticas de *DevOps* trazem inúmeros benefícios como: Entrega mais rápida, mais qualidade e segurança. No

presente trabalho foi utilizado um conceito do universo de *DevOps* conhecido como *GitFlow* que consiste em organizar as *branches* de desenvolvimento no *git*, de forma a manter o código sempre estável e seguro.

Figura 9 - Controle de versão e gerenciamento do código



Fonte: Própria do autor

Durante o processo de desenvolvimento da plataforma gestação, foi realizado um controle de *branches* no *Gitlab*, repositório utilizado para versionar o código deste projeto. O código versionado na *branch Master* sempre foi mantido estável e funcional, a cada nova *Sprint* iniciada era criada uma nova *branch Rn* na qual representava uma *release* (*1, 2, 3 . . . n*) essa *branch* era criada com base na *master* e ao final da *Sprint* era aberto um *merge request* para passar todo esse código da *sprint* em vigor para a master. Todo *commit* realizado no *Gitlab* seguia uma estrutura de *IDs*, isso para facilitar a rastreabilidade do desenvolvimento.

Ao iniciar o desenvolvimento do presente trabalho foi realizado um levantamento de requisitos, no qual foi identificado e mapeado, as principais funcionalidades dos sistemas, além disso as definições de tecnologias que seriam utilizadas e como o sistema deveria agir diante de determinada situação. Mais adiante será evidenciado o levantamento de requisitos e também será exposto o público alvo para o qual essa pesquisa se propõe ajudar, com isso uma das preocupações inerentes ao desenvolvimento foi garantir um sistema funcional em *smartphones* com baixo processamento. Durante todo o desenvolvimento do *software* proposto por esta pesquisa foram realizados testes de forma constante, sendo a última semana de cada *Sprint* dedicada a realização de testes: caixa preta e integração.

4.2 Principais Tecnologias Utilizadas

Para implementação da arquitetura proposta se faz necessário a utilização de um conjunto de tecnologias independentes que juntas são capazes de compor um *software* escalável e sobretudo capaz de receber evoluções para se adequar a novos contextos. Neste tópico serão apresentados algumas dessas tecnologias, no capítulo seguinte que trata do desenvolvimento da arquitetura será apresentado com mais detalhes as tecnologias utilizadas.

- **Spring Boot**: É um *framework* Java, muito utilizado pela indústria de desenvolvimento. A principal vantagem quando comparada a outros *frameworks* está na redução da complexidade em configurar um projeto, além da simplicidade. (KLY-MASH *et al.*, 2020). Além disso, o *Spring Boot* traz uma série de bibliotecas que podem ser utilizadas para resolver os mais diversos tipos de problemas como: controle, persistência e recuperação de dados. Pelo fato de funcionar de forma *stand alone* tem sido bastante utilizado em arquiteturas de microsserviços.
- **RabbitMQ**: Um servidor de mensageria que implementa o protocolo AMQP e possibilita publicar mensagens em filas, tem compatibilidade com inúmeras linguagens de programação, tais como: Java, C# e Python. Foi desenvolvido em Erlang e é *open source*, o que permite sua utilização em projetos comerciais gratuitamente. A escolha se deu pela comunidade ativa e a interoperabilidade, pois o RabbitMQ tem compatibilidade com as principais linguagens de programação do mercado tais como: java e python.
- **Flyway**: É um *framework* que tem como finalidade permitir a migração de banco de dados com mais facilidade. É comum que em um grande projeto, inúmeros desenvolvedores efetuem evoluções e melhorias diariamente, dessa forma, algumas vezes pode existir a necessidade de construir ou deletar um campo ou uma tabela existente no banco de dados. Nesse sentido, sem uma ferramenta de migração, é complexo fornecer em tempo hábil uma atualização segura e confiável para todos os membros do time responsáveis por manter o *software*. A cada nova alteração que precise ser realizada na base de dados o responsável pela alteração ao invés de ir diretamente no banco de dados realizar a alteração, escreve o código no flyway que gerencia essa alteração, dessa maneira, o flyway executa o comando e se tudo ocorrer com sucesso cria uma versão daquela alteração. Isso permite que o banco de dados possa ser gerenciável e versionado.

- **MySQL**: O banco de dados relacional utilizado foi MySQL. Relacional significa que os dados armazenados dentro do banco de dados possuem relação, nesse caso os dados são salvos em tabelas que possuem um ID e é esse ID que se relaciona com outras tabelas fazendo a união de dados e recuperando uma informação de forma completa. Nesse sentido, o MySQL é um dos principais SGBD (Sistema de gerenciamento de banco de dados) utilizado na indústria de desenvolvimento pois, apresenta integração com as principais linguagens de programação, além de fornecer suporte a recursos avançados tais como: *trigger*, cursores, *procedures* e funções.
- **MongoDB**: Diferentemente do modelo relacional, no modelo não-relacional os dados não se relacionam através de IDs que tinha o papel de chave. O banco de dados não-relacional apresenta uma estrutura diferente, onde os dados são persistidos em um grafo. Dessa forma um banco de dados não-relacional apresenta vantagens para escalar a depender do cenário de uso. Existe uma outra forma de persistência de dados chamada de *Key-Value Pairs* onde a estrutura de dados é semelhante a um *HashMap* e é ideal para salvar emissão de *tokens* de autorização. No presente trabalho foi feita a utilização do banco MongoDB, um banco muito flexível, fácil de aprender e que é ideal para persistência de documentos. Ao invés de um registro estar contido em várias tabelas, como seria no modelo relacional, no MongoDB é possível salvar um registro em uma única *collection*, no formato JSON.
- **Docker**: O Docker é um *software* no qual é útil para criar implantações de aplicativos de forma rápida. Através da criação de contêineres que são criados e gerenciados pelo docker é possível subir aplicações completas, além de ser mais fácil escalar as aplicações em diferentes ambientes, tendo em vista que um container é criado através de uma imagem e uma imagem é fácil de transportar entre diferentes ambientes.

4.3 Considerações do Capítulo

Neste capítulo, foi explanado a metodologia utilizada para o desenvolvimento da aplicação proposta. Além disso foi apresentado pontos importantes sobre a utilização da metodologia ágil *scrum* e conceitos importantes da engenharia de qualidade de *software* para o desenvolvimento de aplicações modernas.

5 DESENVOLVIMENTO DA APLICAÇÃO

De acordo com Al-Debagy e Martinek (2018) uma aplicação monolítica responde melhor, quando comparada com uma arquitetura de microsserviços, para uma carga restrita a 100 usuários. Porém a medida que uma aplicação cresce é importante dimensionar sua arquitetura para garantir escalabilidade e dessa forma garantir a satisfação dos usuários. Dessa forma a arquitetura de microsserviços vem crescendo e ganhando cada vez mais espaço na indústria de desenvolvimento.

Neste capítulo, serão apresentados os passos que foram empregados para o desenvolvimento da arquitetura microsserviços. Será apresentado uma visão geral das tecnologias utilizadas no *back-end, front-end* e banco de dados desenvolvidas neste trabalho de dissertação.

5.1 Arquitetura Distribuída

Uma arquitetura centralizada é suficiente para atender aplicações que não costumam receber picos de acessos em determinados períodos, além disso é normalmente mais fácil manter, contudo apresentam desvantagens quando comparadas ao modelo descentralizado. Uma arquitetura distribuída é mais eficiente quando existe a necessidade de alocar recursos para atender picos de acessos, além disso o custo é variável, isto significa que em determinados períodos do ano quando existir a necessidade de alocação de mais recursos o custo aumenta, no entanto passado o período o custo é reduzido. Além disso, no modelo distribuído é possível garantir a alta disponibilidade. Outra vantagem é a possibilidade de implantação com maior segurança, se ocorrer uma falha na implantação de um módulo, apenas aquele módulo ficará indisponível e não a aplicação como um todo.

Segundo Wang, Chen e Yang (2010) a confiabilidade é um dos elementos cruciais para determinar a qualidade de um sistema de computação distribuído. O modelo arquitetural utilizado neste trabalho foi o distribuído. Isso se deu pelo fato de que uma arquitetura descentralizada garante uma alta disponibilidade de recursos, redundância, extensibilidade e escalabilidade.

5.2 Microsserviços Reativos

A utilização de uma arquitetura de microsserviços reativos traz inúmeras vantagens a um projeto de engenharia de *software*, de acordo com Bonér (2017) os microsserviços reativos devem fornecer escalabilidade e resiliência, além de possuir seus próprios dados e serem autônomos. Dessa maneira o seu funcionamento não deve depender de outros microsserviços para funcionar corretamente. Com uma arquitetura de microsserviços reativos é possível definir responsabilidades, escalar cada serviço de forma individual, realizar testes e monitorar isoladamente cada serviço da aplicação, além de realizar *deployment* de serviço sem afetar outros serviços existentes na aplicação.

É comum que em uma arquitetura microsserviços um serviço necessite de uma informação pertencente a outro serviço. Muitas vezes a recuperação e persistência de dados envolvem um fluxo grande e complexo. No modelo arquitetural de microsserviços reativos é difícil garantir consistência, para minimizar esse problema foi então apresentando por Miles (2016) o padrão de projetos *Events-First Domain Driven Design*, que deriva do padrão de projeto *Domain-Driven Design* (DDD), que é útil para modelar sistemas em torno de regras de negócio.

Entre as boas práticas da engenharia de *software* está a importância de usar os padrões de projetos. De acordo com Washizaki *et al.* (2009) um padrão de projeto é uma solução abstrata e repetível, útil para resolver problemas em projetos de *software* que ocorrem em determinados contextos.

No presente trabalho foi utilizado o padrão *Events-First Domain Driven Design* em alguns contextos. Na realização de um novo cadastro, a arquitetura precisa enviar um e-mail de confirmação para a gestante, informando o sucesso ou não do cadastro e ao mesmo tempo postar a informação de cadastro em um *broker*, para ser consumida pelo serviço responsável por calcular a paridade entre a gestante e a unidade de saúde mais próxima.

Figura 10 - Classe Modelo *Pregnant* com *AggregateRoot*

```
1  package br.edu.uepb.nutes.core.domain.model;
2  @Data
3  @EqualsAndHashCode(onlyExplicitlyIncluded = true, callSuper =
       false)
4  @Entity
5  public class Pregnant extends AbstractAggregateRoot<Pregnant>{
6
7    public void publishEvent() {
8      registerEvent(new PregnantEvent(this));
9    }
10
11 }
```

Fonte: Própria do autor

A figura 10 demonstra a classe modelo de *Pregnant* (Gestante), essa classe herda de uma outra classe chamada *AbstractAggregateRoot* que originalmente é implementada pelo próprio *Spring Framework*, tendo o papel de disparar eventos. É preciso dentro da classe modelo sobrescrever o método *registerEvent* da classe mãe e lançar o novo evento personalizado.

A classe *PregnantEvent* contém apenas um atributo do tipo *Pregnant*. Nesse momento o evento já foi disparado, bastando apenas capturar o evento e realizar as ações necessárias. Para realizar a captura o Spring já dispõe de uma anotação chamada *@TransactionalEventListener*, dessa forma basta apenas anotar o método responsável pela captura passando como parâmetro o tipo de evento que se deseja capturar. Na figura 11 é possível observar a implementação.

Figura 11 - Captura de evento para envio de e-mail

```java
package br.edu.uepb.nutes.gestation.records.domain.listener;

@Component
public class SendEmailAddressListener {

  @Autowired
  private SendEmailService sendEmail;

  @TransactionalEventListener
  public void sendEmail(PregnantEvent event) {

    User user = event.getPregnant().getUser();

    var message = Message.builder()
        .subject("Cadastro efetuado com sucesso - "
            + user.getName())
        .body("confirmed-registration.html")
        .variable("user", user)
        .recipient(user.getEmail())
        .build();

    sendEmail.send(message);

  }
}
```

Fonte: Própria do autor

Figura 12 - Captura de evento para postar informações em *broker*

```java
package br.edu.uepb.nutes.gestation.records.domain.listener;

@Component
public class NotifyQueueUserListener {

  @Autowired
  private IQueueSend<User> userProducer;

  @TransactionalEventListener(phase = TransactionPhase.
     AFTER_COMMIT)
  public void notifyQueue(PregnantEvent event) {
    try {
      User user = event.getPregnant().getUser();
      userProducer.send(user);
    }catch (AmqpRejectAndDontRequeueException e) {
      throw new AmqpException("Erro ao enviar dados para a fila -
          QUEUE_ADDRESS_PARITY");
    }
  }
}
```

Fonte: Própria do autor

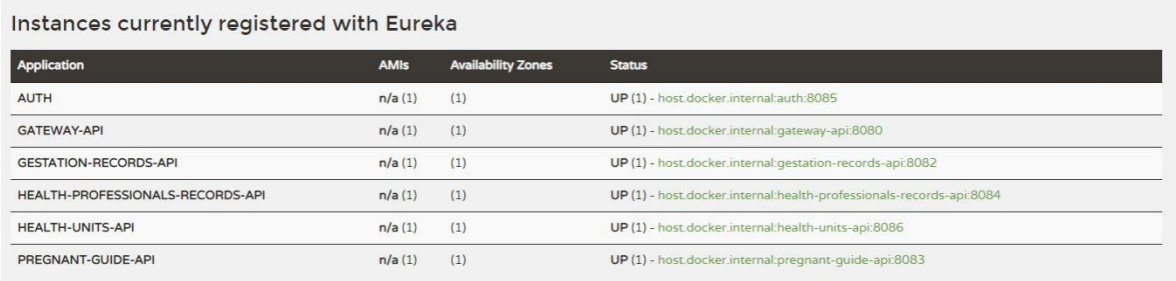
Na figura 12 é retratado o cenário de captura de evento e inserção no *broker*. O JPA (Java Persistence API), realiza uma série de operações de banco em memória para só então realizar um *commit* no banco, isso é um comportamento padrão do JPA, para evitar que a aplicação fique a todo instante descarregando operações no banco, sobrecarregando a aplicação. Por esse motivo foi utilizado a anotação: *TransactionalEventListener* informando que a fase da transação deveria ser *AFTER COMMIT* na qual a finalidade é realizar a inserção de um registro no *broker* somente após o JPA descarregar todas as operações em memória no banco de dados, isso é útil para evitar possíveis inconsistências de dados.

5.3 Monitoramento de Microsserviços

Os processos de monitoramento de microsserviços são indispensáveis para identificar métricas e assegurar a disponibilidade da aplicação. Segundo Brandão (2020) os processos de monitoramento são essenciais para reportar erros e identificar comportamentos errôneos. Mediante o retorno das ferramentas de monitoramento é possível realizar previsões, realizar correções de forma preventiva e escalar a aplicação.

Normalmente em sistemas críticos, são utilizadas diversas técnicas combinadas com ferramentas e processos para garantir a observabilidade e monitoramento de uma aplicação tais como: Grafana, ferramenta utilizada para geração de gráficos e *Dynatrace*, muito utilizado para realizar o monitoramento de ponta a ponta de um *software*. No presente trabalho não foram utilizadas as ferramentas citadas anteriormente, pois deixaria o escopo da pesquisa muito extenso.

Figura 13 - Eureka Server registro de serviços



Fonte: Própria do autor

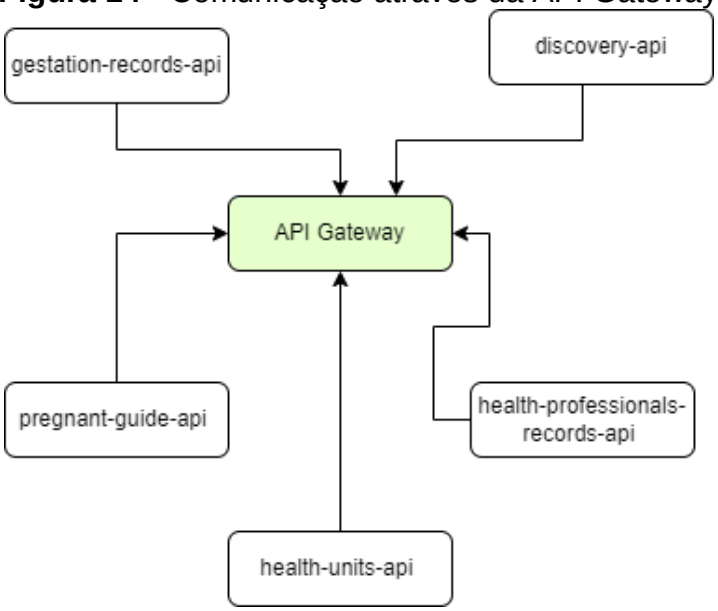
Dessa forma foi utilizado o registro de logs na aplicação e o painel do eureka

server, que está exposto na figura 13 disponibilizado no projeto *open source* da netflix, OSS, no qual permite ver de forma gráfica os serviços que estão em execução.

5.4 Comunicação entre Microsserviços

Como já foi discorrido no capítulo de referencial teórico, existem alguns modelos de comunicação entre microsserviços, entre os quais: o protocolo HTTP que é largamente utilizado atualmente para a comunicação síncrona. Dessa forma é amplamente utilizado o padrão BFF (*Back-end* para *Front-end*) que também é conhecido pelo nome padrão API *Gateway*, no qual possui semelhança ao padrão fachada, um padrão de projetos orientado a objetos. O BFF funciona da seguinte maneira, no contexto de uma aplicação de microsserviços, existe uma API *Gateway* responsável por orquestrar toda a comunicação entre as demais API's, dessa maneira o lado cliente da aplicação sempre fará a comunicação com as demais API's através da API de orquestração *Gateway*, esse comportamento pode ser melhor entendido na figura 14.

Figura 14 - Comunicação através da API Gateway



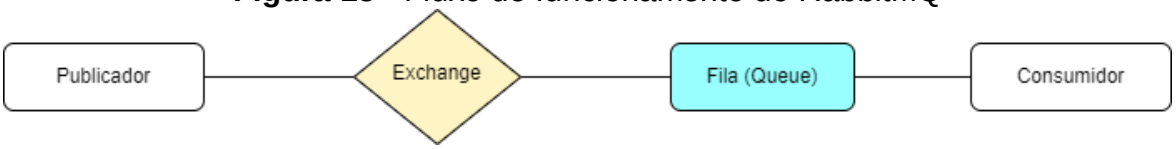
Fonte: Própria do autor

Contudo é comum que em determinados *softwares* se utilizem de um modelo de comunicação assíncrona. Neste contexto o protocolo *Advanced Message Queuing Protocol* (AMQP) é bastante útil, pois tem o papel de estabelecer comunicações de forma assíncrona entre aplicações.

Para fazer a utilização do protocolo AMQP é necessário escolher uma tecnologia compatível com o protocolo, nesse caso um *broker*. O *broker* é a tecnologia intermediária da comunicação. A atribuição do *broker* é receber e enviar mensagens. Para a implementação da arquitetura proposta foi preferido a utilização do *broker* RabbitMQ.

O RabbitMQ implementa além do protocolo AMQP, outros três protocolos: MQTT, STOMP e HTTP. Foi desenvolvido na linguagem de programação Erlang e tem se tornado muito utilizado para desacoplar serviços. As mensagens postadas no RabbitMQ são armazenadas em memória, isso traz um ganho de performance para as aplicações.

Figura 15 - Fluxo de funcionamento do RabbitMQ



Fonte: Própria do autor

Para permitir a troca de mensagem entre serviços o RabbitMQ implementa uma fila, contudo o *producer* (produtor da mensagem) não envia a mensagem direto para fila, a mensagem é enviada para um domínio chamado de *exchange*, conforme a figura 15. Dessa forma a função do *exchange* é postar a mensagem na fila. Nesse sentido o *exchange* pode enviar uma mesma mensagem para inúmeras filas e no final da cadeia existe o papel do *consumer* (consumidor), o *consumer* após ler a mensagem postada na fila executa alguma ação.

5.5 Conceitos de Engenharia de Software

No desenvolvimento de um *software* é necessário delinear sobre vários aspectos. Um projeto de *software* mal planejado terá sérios problemas com o passar do tempo, além disso, o custo de manutenção será cada vez mais alto. Sendo assim é importante analisar requisitos e o modelo de negócio para propor uma solução capaz de evoluir sem maiores impactos. Krüger (2019) defende que para evoluir um *software* é necessário que o desenvolvedor compreenda bem o funcionamento do sistema e consequentemente conheça sobre o negócio.

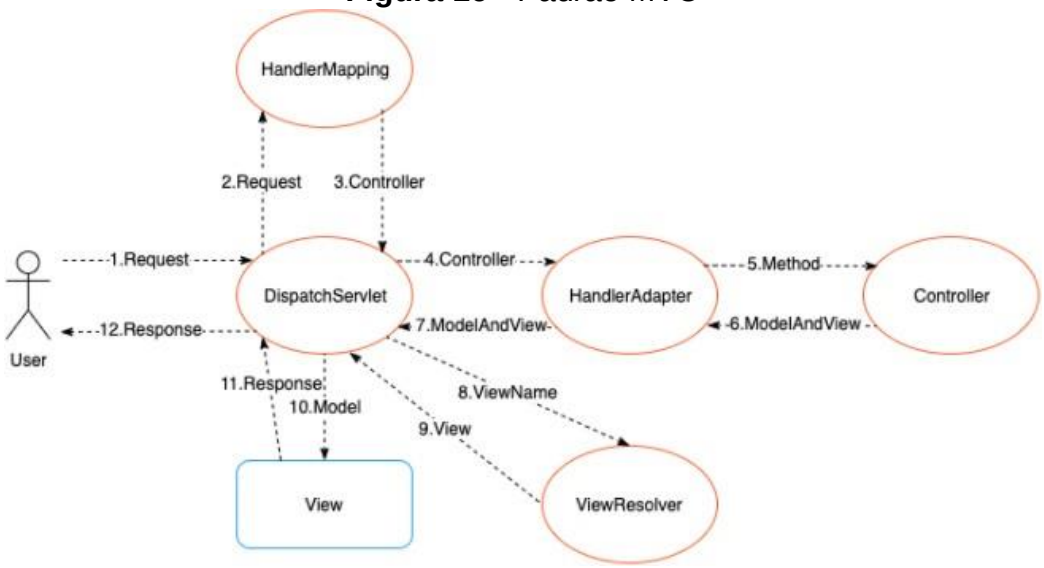
Com a pretensão de propor uma modelagem de software eficaz a indústria de

desenvolvimento tem fomentado o interesse pela utilização de boas práticas e padrões de projetos. No desenvolvimento deste trabalho foram utilizados alguns padrões de projeto importantes.

5.5.1 Padrões de Projeto

O *pattern Model-View-Controller* (MVC) é bastante utilizado pelo mercado, isso se dá devido sua flexibilidade. O MVC tem uma divisão bem definida de camadas, sendo que é possível reaproveitar o modelo de dados e os controles para utilizar nas mais diversas *views*, nesse contexto as *views* são interfaces chamadas pelos controladores que podem ser móvel, WEB ou até *desktop*.

Figura 16 - Padrão MVC



Fonte: Omidh (2019)

De acordo com Ramírez-Noriega *et al.* (2020) não é fácil implementar o padrão MVC em um projeto grande, mas quando utilizado torna a aplicação mais fácil de personalizar e a maior parte das aplicações WEB da atualidade implementam o padrão MVC. Observando a figura 16 é possível averiguar o comportamento do padrão MVC. A base do *back-end* desenvolvida por este trabalho foi construída no *framework* Java *Spring*, sendo assim o *Spring* guia nativamente o desenvolvimento baseado em camadas, aplicando com isso o MVC ao contexto de utilização.

Existe uma classificação para os padrões de projeto denominada pelo *Gang of Four* (GoF) conforme Hussain, Keung e Khan (2017), para se referenciar a padrões

de projetos de criação, entre esses padrões de criação está o *builder*. O *pattern Builder* é importante pois permite a criação simplificada de objetos complexos, isso traz mais legibilidade para o código e reduz a complexidade, tornando assim mais fácil futuras manutenções.

Figura 17 - Captura de evento para postar informações em broker

```
1  private Problem.ProblemBuilder createProblemBuilder(HttpStatus
       status, ProblemType problemType, String detail) {
2
3    return Problem.builder().timestamp(OffsetDateTime.now()).
       status(status.value()).type(problemType.getUri())
4      .title(problemType.getTitle()).detail(detail);
5  }
```

Fonte: Própria do autor

Na figura 17 é exposto um exemplo de criação de um objeto através do padrão *builder*. Em uma única linha de código é declarado e criado um objeto do tipo *Problem*. O *framework* java lombok[2] já trata da implementação do padrão em seu *background*, dessa maneira é preciso apenas anotar a classe na qual se deseja utilizar o padrão *builder* com a anotação *@Builder*.

O Padrão DAO (Data Access Object) nasceu da necessidade de padronizar a forma como o *software* se comunicava com o banco de dados, com isso é preciso que exista uma separação bem definida de regras de negócios e modelo de dados. Neste trabalho foi utilizado o *Spring Data* JPA para fazer toda a manipulação de dados, o JPA implementa o padrão DAO, com isso consegue transformar entidade em objetos e objetos em entidades.

5.5.2 S.O.L.I.D

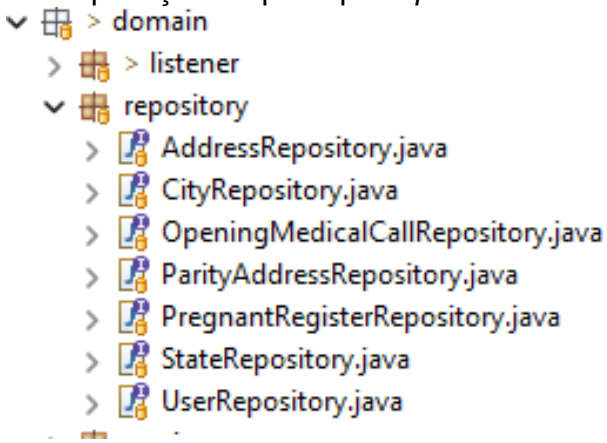
A sigla S.O.L.I.D foi definida por Michael Feathers, com base nos conceitos apresentados anteriormente por Fowler (2000) com a finalidade de guiar os desenvolvedores a desenvolver código mais simples, reutilizável e escalável. É uma boa prática de engenharia de *software* aplicar o S.O.L.I.D a projetos. O S.O.L.I.D trata de descrever cinco princípios de programação orientado a objetos, quando devidamente aplicados torna o código fonte de uma aplicação mais limpo, objetivo e

[2] O lombok é um *framework* que possibilita a escrita de código java sem verbosidade.

claro. Cada letra da sigla, faz referência a um princípio.

- **Single Responsibility Principle**: Esse princípio defende que uma função, uma classe ou um método possua apenas uma única responsabilidade. Delegando assim a cada coisa uma responsabilidade única. Nesta pesquisa utilizamos um padrão de projetos chamado de *Events-First Domain Driven Design*, conforme descrito no tópico de microsserviços reativos para informar a outras rotinas do sistema determinadas informações e evitar assim quebrar o primeiro princípio do S.O.L.I.D da responsabilidade única.
- **Open-Closed Principle**: O princípio aberto-fechado define que objetos e entidades devem sempre está aberta para serem estendidas, mas fechados para serem modificadas. Isso garante que uma nova funcionalidade implementada não quebre uma funcionalidade já existente. Pois o princípio defende a ideia de nunca alterar código já existente.

Figura 18 - Aplicação do princípio *Open-Closed Principle*



Fonte: Própria do autor

Figura 19 - Aplicação do princípio *Open-Closed Principle* interface

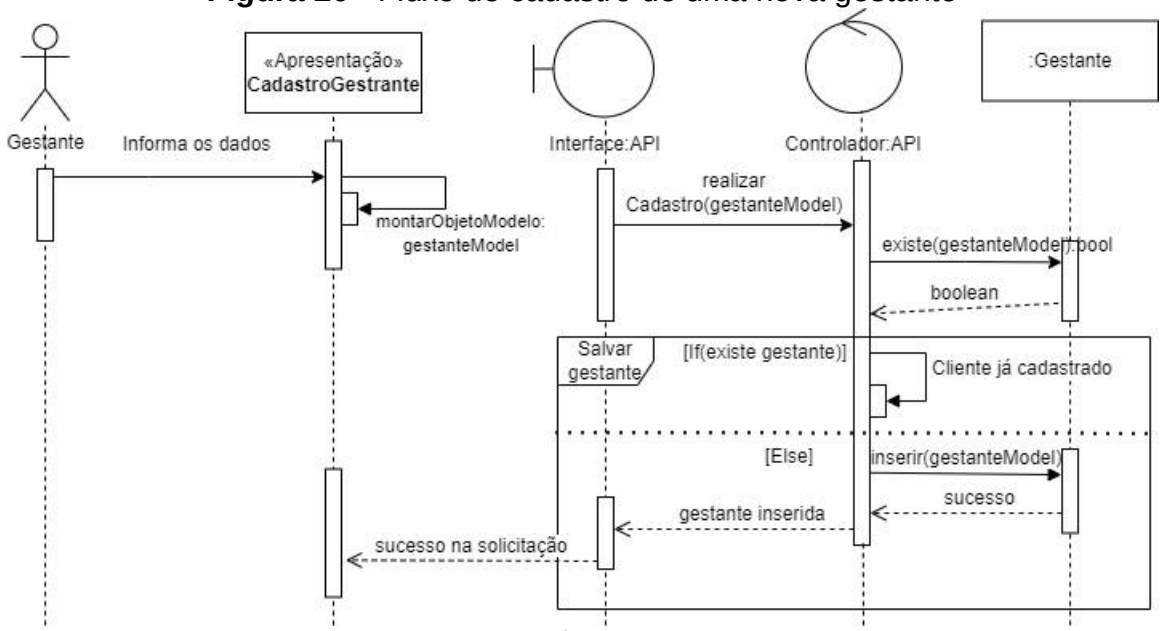
```
1  @Repository
2  public interface AddressRepository extends JpaRepository<Address, Long>{
3      //todas as funcionalidades de quem herda sao colocadas aqui dentro
4  }
```

Fonte: Própria do autor

Na figura 18 é exposto um conjunto de interfaces, as interfaces são importantes para evitar problemas com acoplamento. Seguindo o princípio vigente, novas funcionalidades apenas herdam características já definidas, conforme o exemplo exposto na figura 19.

- **Liskov Substitution Principle**: Esse princípio elucida que uma classe derivada deve ser mutável por sua classe base.
- **Interface Segregation Principle**: O princípio da Segregação da Interface esclarece que uma classe não deve ser obrigada a implementar métodos de uma interface nas quais ela não irá utilizar.
- **Dependency Inversion Principle**: Esse princípio trata da inversão de dependências no qual defende que um *software* deve depender de abstrações e não de implementações. Isso significa que um módulo A de um *software* não deve depender do módulo B ou C, mas sim da abstração do módulo B ou C.

Figura 20 - Fluxo de cadastro de uma nova gestante



Fonte: Própria do autor

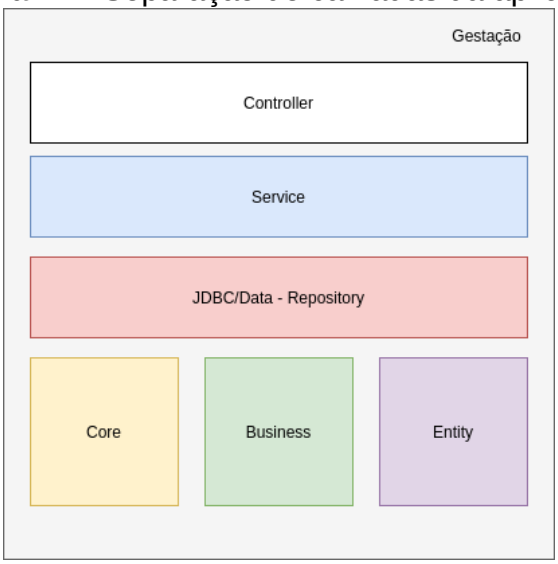
O diagrama exposto na figura 20 representa um novo cadastro de uma gestante na aplicação. Nesse fluxo não existe dependência entre as camadas, ou seja, a camada de apresentação não depende da camada de controle, apenas da abstração da camada de controle. Dessa forma satisfaz o princípio *Dependency Inversion Principle*.

5.6 Definição dos Microsserviços

Em cada uma das API's desenvolvidas neste trabalho, foi utilizada uma separação bem definida de camadas delimitando as responsabilidades, reduzindo o

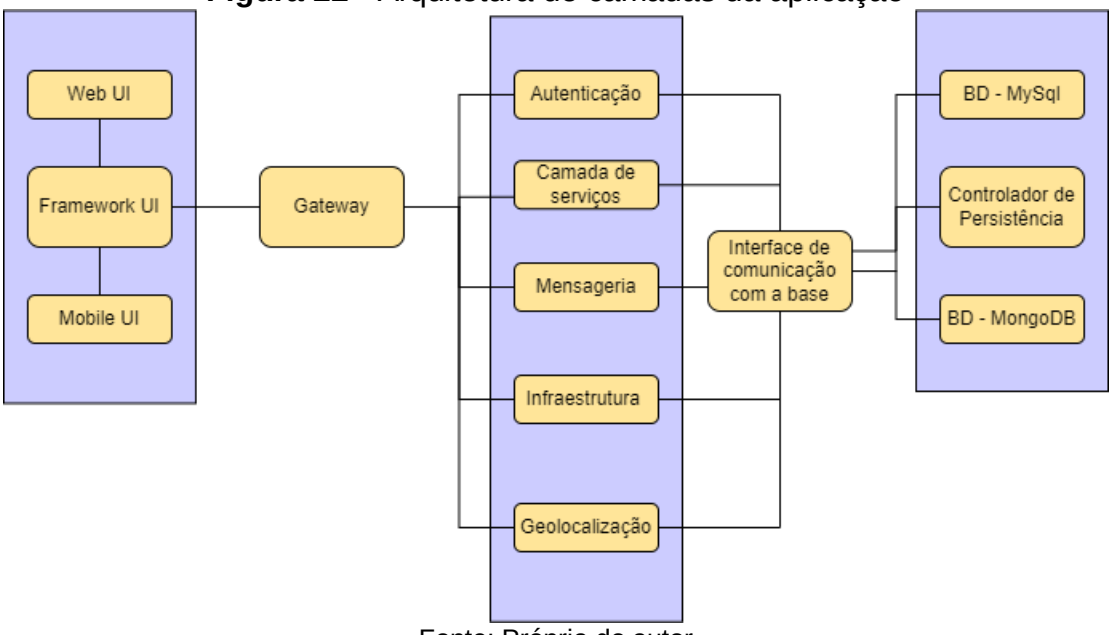
acoplamento e aumentando a coesão.

Figura 21 - Separação de camadas da aplicação



Fonte: Própria do autor

Figura 22 - Arquitetura de camadas da aplicação



Fonte: Própria do autor

O controle da aplicação se comunica com o banco de dados através da camada de serviço. A camada de repositório é responsável por se comunicar e gerenciar as sessões com o banco de dados. A camada de negócio está totalmente desacoplada das demais camadas e todas as rotinas reutilizáveis dentro do *software* estão implementadas dentro do core da aplicação.

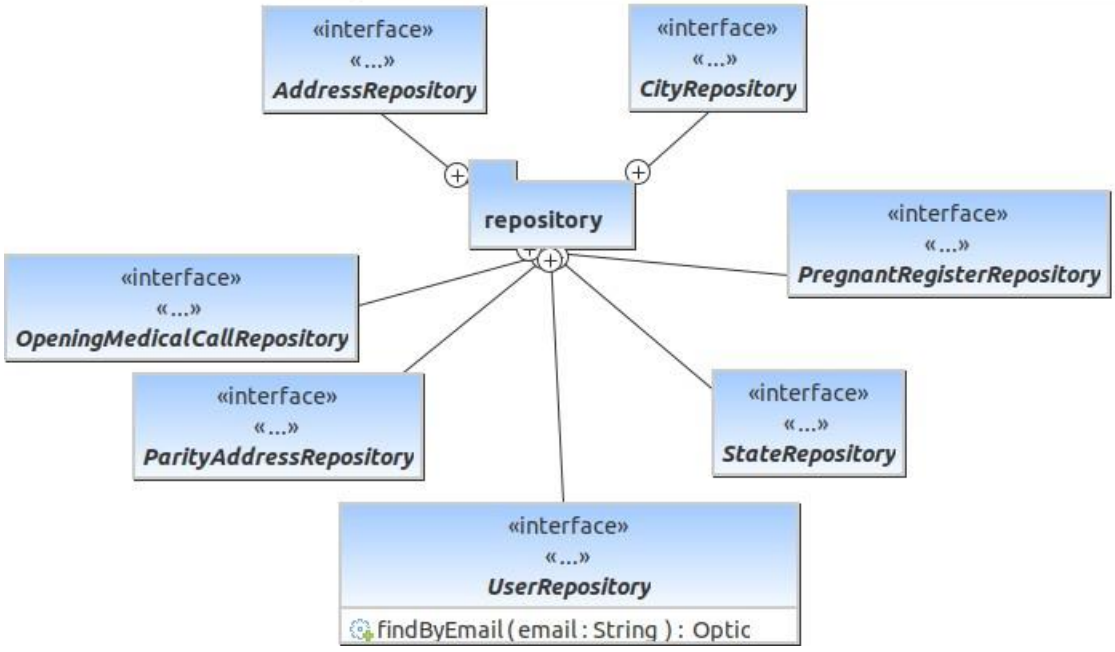
Normalmente em uma arquitetura de microsserviços existem duas estratégias

para estruturar o banco de dados, um único banco de dados para atender toda a aplicação e um banco de dados por serviço. Para Fan e Ma (2017) o preferível seria utilizar um banco de dados por serviço, pois reduziria o acoplamento entre os serviços, no entanto dividir o banco de dados por serviço pode gerar inconsistência de dados.

Neste trabalho todos os serviços compartilham os mesmos bancos de dados, isso gera mais economia e evita problemas de inconsistência de dados, contudo cada serviço da arquitetura é responsável por gerenciar a estrutura e as regras de negócios relacionadas ao banco de dados nos quais estes se utilizam. Isso reduz o acoplamento entre os serviços, tendo em vista que um serviço não conhece a estrutura de dados do outro.

A principal API da arquitetura é a *gestation-records-api*, justamente a responsável por conter regras de negócios relativas à gestante. Essa API tem outras responsabilidades, como enviar e-mail com informações importantes e realizar a postagem e escuta de Mensageria com dados providos da API responsável por manter as unidades de saúde.

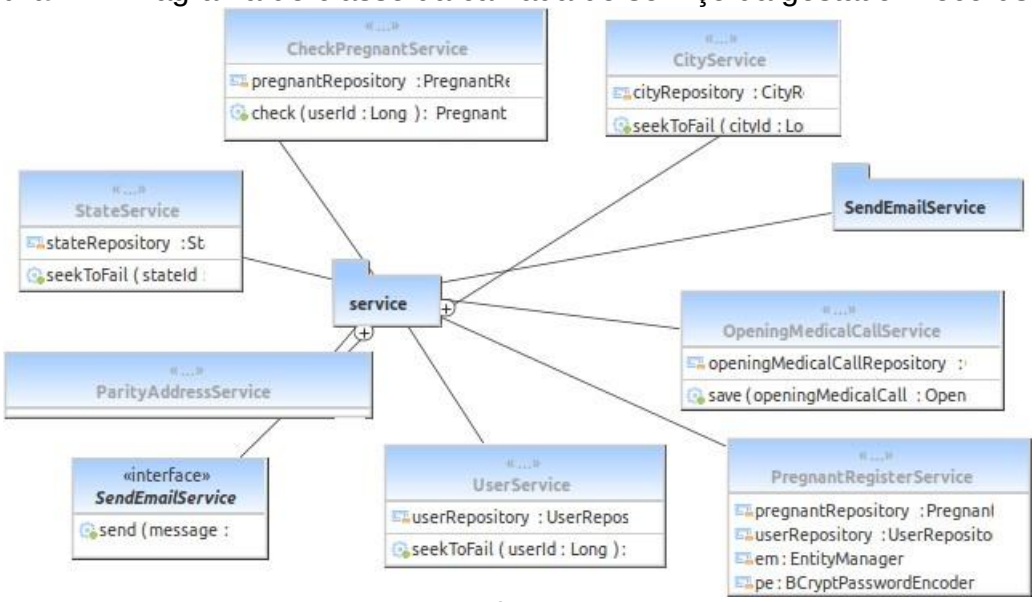
Figura 23 - Diagrama de classe da camada de repositório da gestation-records-api



Fonte: Própria do autor

A camada de *repository* faz uso massivo das implementações do *Spring Data JPA*. Cada classe de repositório implementa uma interface do *Spring* que traz um conjunto de consultas pré-configuradas.

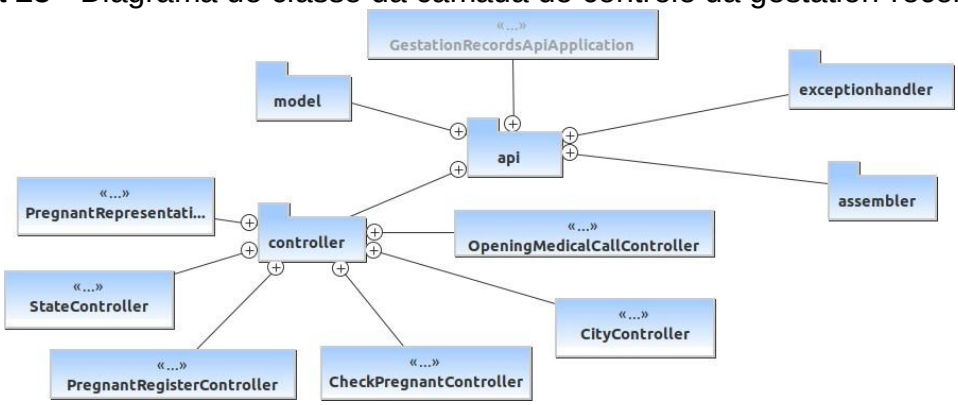
Figura 24 - Diagrama de classe da camada de serviço da gestation-records-api



Fonte: Própria do autor

A camada de serviço se comunica com a camada de repositório e aplica as regras de negócios associadas a cada rotina.

Figura 25 - Diagrama de classe da camada de controle da gestation-records-api



Fonte: Própria do autor

Como uma boa prática de desenvolvimento, os dados provenientes do banco de dados são convertidos em uma sub-camada da camada de controle, no exemplo da figura 25 esta sub-camada é a *assembler*. Isso traz uma granularidade melhor a aplicação, evitando assim que um serviço responda a uma requisição com mais informações do que realmente era preciso.

Para compor a arquitetura de microsserviços desenvolvida nesta dissertação foram construídos 7 serviços bem definidos.

Quadro 2 - Serviços da arquitetura gestação

Nome do serviço	Linguagem	Descrição	Banco de dados
gateway-api	Java 11	Serviço responsável por orquestrar todas as requisições.	MYSQL
discovery-api	Java 11	Registra estados dos demais serviço	Não faz uso.
gestation-records -api	Java 11	Responsável por gerenciar regras de negócios relacionadas à gestante.	MYSQL
health-professionals -records-api	Java 11	Responsável por gerenciar regras de negócios relacionadas aos profissionais de saúde.	MYSQL e Mongo
health-units-api	Java 11	Responsável por gerenciar regras de negócios relacionadas as unidades de saúde.	MYSQL
pregnant-guide-api	Java 11	Responsável por filtrar questões e dúvidas relacionadas à gestação.	MYSQL e Mongo
auth-api	Java 11	Responsável por gerenciar a segurança da aplicação.	MYSQL

Fonte: Própria do autor

No quadro 2 foi apresentado uma descrição geral para cada serviço desenvolvido. No início do desenvolvimento a última versão estável LTS (*long-term support*) do Java era a 11. Todos os serviços compartilham de um banco de dados MYSQL, exceto o serviço de *discovery-api* que monitora os dados em memória.

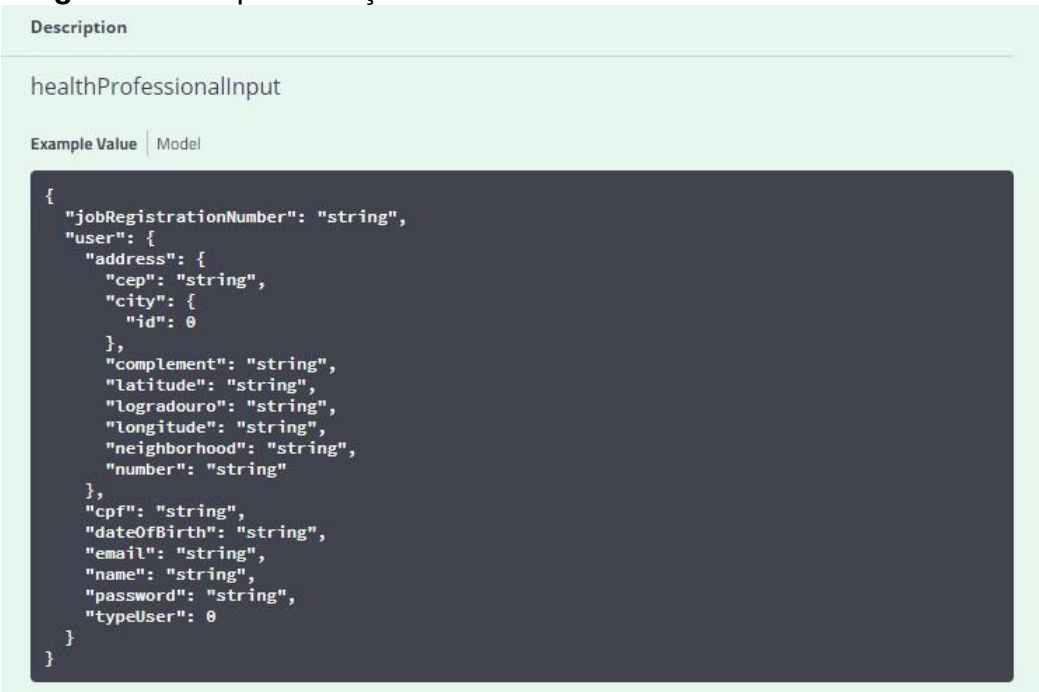
5.6.1 Swagger

Todas as API's foram documentadas em um *framework* chamado *Swagger*, que facilita a descrição e sobretudo a visualização dos serviços de uma API. O *Swagger* é ótimo para prover a integração entre sistemas, pois mesmo que não seja conhecida as minúcias de um serviço, é possível integrar com outros serviços e até outros sistemas com base no *input*, pois todo serviço precisa de *input* para retornar um *output*. À vista disso o *Swagger* rapidamente se popularizou na indústria de desenvolvimento.

Figura 26 - Serviços da API health-professionals-records

Fonte: Própria do autor

Figura 27 - Representação de Cadastro de um Profissional de Saúde



Fonte: Própria do autor

As figuras 26 e 27 apresentam visualmente uma documentação gerada para API responsável pelas tratativas dos dados dos profissionais de saúde.

5.6.2 Tratativa de erros em API's

Ao fazer uma requisição HTTP para uma API podem ocorrer diversos erros tais

como: recurso não encontrado (erro 404), erro no servidor (erro 500), usuário não autorizado (erro 403) entre outros. Sendo assim, é necessário que o retorno a uma requisição contendo uma mensagem de erro seja clara e objetiva. Com o propósito de padronizar essas mensagens de erro foi proposta por Nottingham e Wilde (2016) a RFC 7807 no qual o projeto desenvolvido nesta dissertação implementou em todas API's desenvolvidas. O quadro 2 contém a lista de API's.

Figura 28 - Estrutura de mensagem de erro

```
{
    "status": 404,
    "type": "https://gestation-records.com.br/
             recurso-nao-encontrado",
    "title": "Recurso não encontrado",
    "detail": "Não existe um cadastro de gestante com código 156",
    "userMessage": "Não existe um cadastro de gestante com
                    código 156",
    "timestamp": "2022-02-01T07:29:59.868231-03:00"
}
```

Fonte: Própria do autor

A figura 28 expõe a reprodução de um cenário de erro no qual foi realizada uma busca na API de gestantes por um código inexistente, nesse contexto a API retorna uma mensagem no formato JSON, contendo o status do erro, o tipo do erro, o momento da execução que o erro ocorreu além de uma mensagem contendo mais alguns detalhes relacionados ao problema. Esse padrão segue o padrão definido na RFC 7807 e foi utilizado de forma global em todas as API implementadas por este trabalho.

5.6.3 Cache

A utilização de *cache*[3] em aplicações é seriamente importante para garantir que páginas, serviços e recursos sejam carregados com mais rapidez. Isso traz um ganho considerável de *performance* nas aplicações tendo em vista que quando uma informação está no *cache* é rapidamente carregada de algum *browser* local ou um

[3] *Cache* é um mecanismo de acesso rápido que tem o intuito de apresentar informações e dados de forma quase instantânea.

servidor intermediário que gerou uma cópia da informação original. É comum utilizar *cache* para alguns serviços específicos de uma aplicação, principalmente informações que não mudam com constância, tais como: serviço que traz a informação de cidades e estados por CEP. Além disso, é preciso ter cautela para evitar manter na *cache* informações sensíveis do usuário.

Há dois tipos distintos de *cache* HTTP: *cache* privado e *cache* público ou compartilhado. No *cache* privado o dado que for mantido na *cache* ficará disponível apenas no navegador local do usuário, nesse sentido ao tentar abrir uma informação em um outro *browser* o navegador fara´ uma nova chamada ao *web server* para recuperar a informação. No *cache* público uma mesma informação ficará disponível para vários usuários, essa fica cacheada normalmente em um servidor intermediário, isso já torna mais rápida a solicitação pelo recurso, tendo em vista que não haverá chamada HTTP ao *web server*. Normalmente o navegador identifica que deve deixar uma informação em *cache* pelo cabeçalho da requisição provinda do *web server*.

A utilização de *cache* é importante quando não se compartilha dados sensíveis do usuário. É necessário ter bastante cautela quando se trata de uma aplicação em saúde.

Em nenhum momento um dado clínico de um usuário/paciente pode ficar exposto. E´ importante salientar que muitas informações em *cache* e *cookies* são utilizadas pelos *browser* para recomendação de conteúdo e anúncio personalizado. No presente trabalho foi implementado a estratégia de *cache* local, contudo inicialmente apenas o serviço que retorna a lista de cidades e estados está sendo mantida em *cache*.

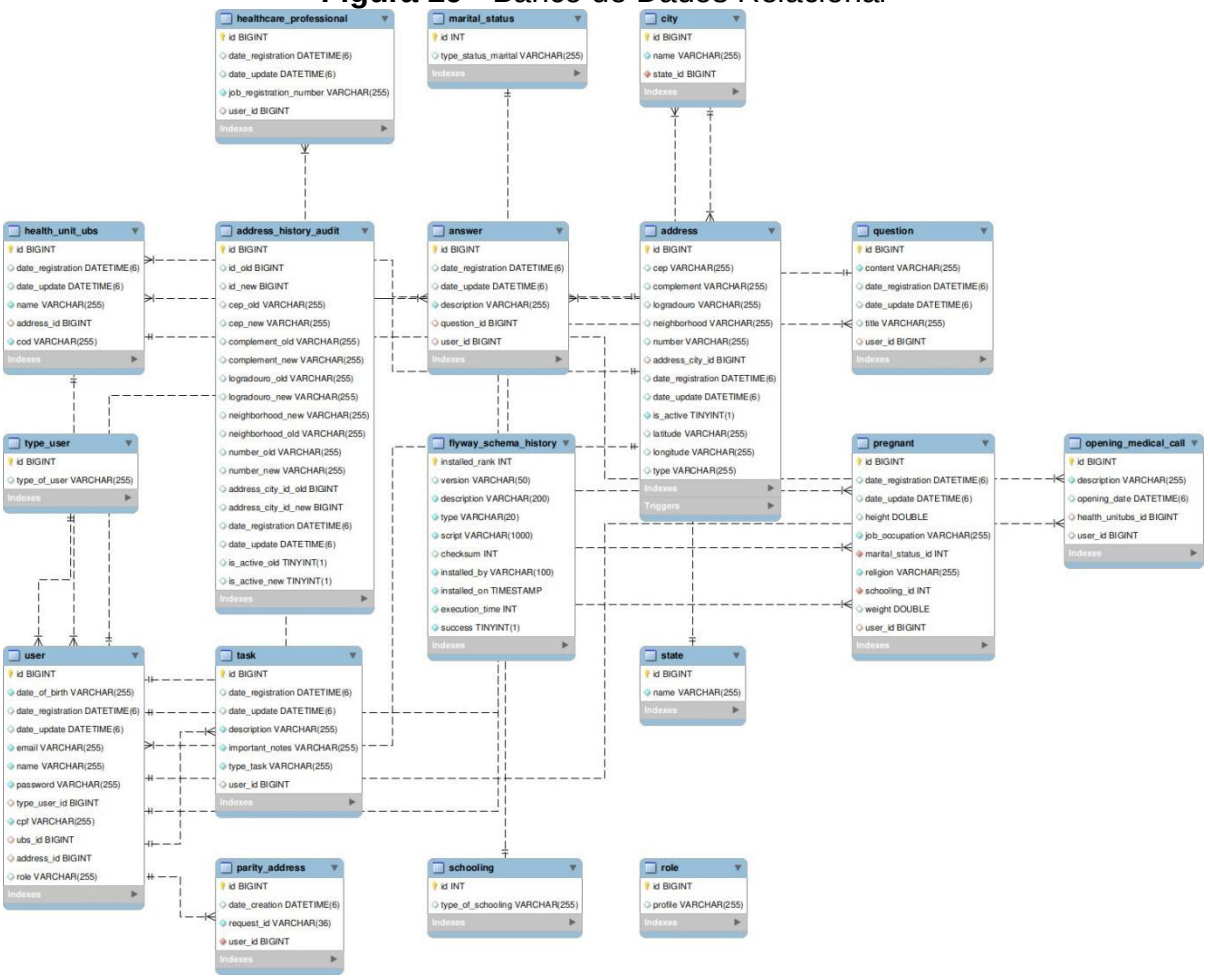
5.7 Banco de Dados

Com a atual exigência de *performance* das atuais aplicações que funcionam na WEB é comum que a indústria de desenvolvimento faça combinações entre diferentes modelos de banco de dados, buscando assim encontrar a forma mais rápida e barata de responder às solicitações dos usuários. Há dois modelos popularmente usados pela indústria no mundo inteiro: Banco de dados não-relacional e relacional.

O modelo relacional é o mais tradicional e apresenta uma abordagem baseada em armazenar as informações no esquema de tabelas com linhas e colunas, onde normalmente as colunas representam o tipo da informação e as linhas os dados a

serem armazenados. No modelo não-relacional é utilizado um modelo de dados otimizado, onde as informações são representadas normalmente de três maneiras distintas: Através de um conjunto de chave e valor, por meio de um documento JSON e através de arestas e vértices para persistir gráficos e imagens.

Figura 29 - Banco de Dados Relacional



Fonte: Própria do autor

Figura 30 - Banco de Dados Não-Relacional

Fonte: Própria do autor

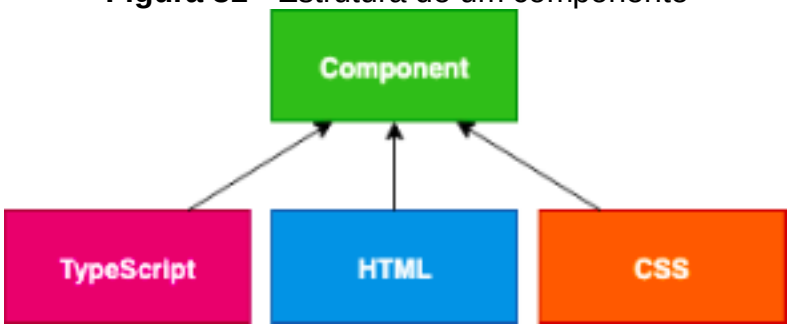
Neste projeto foi utilizado um banco de dados relacional, conforme o diagrama exposto na figura 29, para fazer a persistência de dados relacionados a maior parte às regras de negócio da aplicação, foi verificado que a melhor forma de manipular esses dados seriam utilizando um modelo relacional, pelo fato das entidades estarem ligadas. Em segundo momento foi verificada a necessidade de utilização de um banco não-relacional, pois este possui muita rapidez e flexibilidade (PATIL *et al.*, 2017), para isso foi feita a opção de salvar os dados no modelo de documento, persistindo os mesmo em um formato JSON, o contexto dessa utilização se deu pelo fato de ser mais fácil manipular os dados no fluxo de agendamento de consultas nas unidades de saúde.

5.8 Interface

5.8.1 Front-End da Plataforma WEB

O angular é um *framework front-end* bastante utilizado pela indústria de desenvolvi- mento de *software*. Foi desenvolvido e é atualmente mantido pelo o Google. Possui licença MIT *License* e atualmente está na versão 12.1.4 versão estável. O Angular foi desenvolvido em cima da linguagem JavaScript e entre os vários frameworks de *front-end* desenvolvidos em JavaScript, o Angular foi pioneiro na utilização do *typescript*, responsável por tornar o JavaScript uma linguagem desenvolvimento tipada.

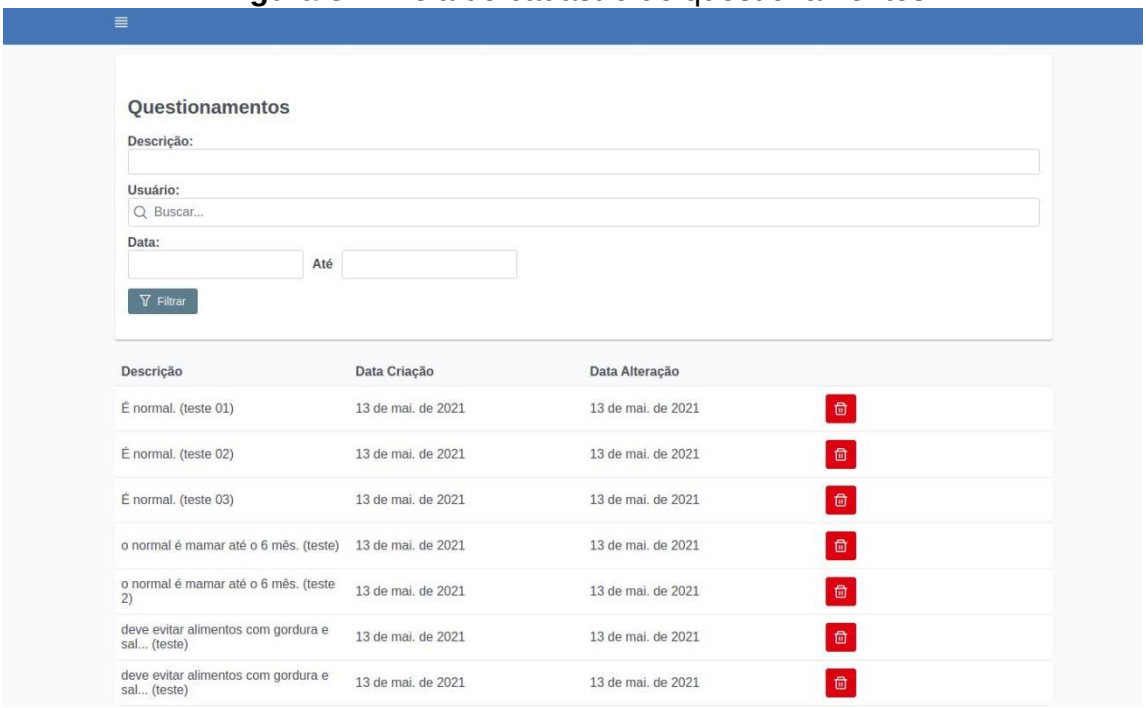
Figura 31 - Estrutura de um componente



Fonte: Escott e Noble (2019)

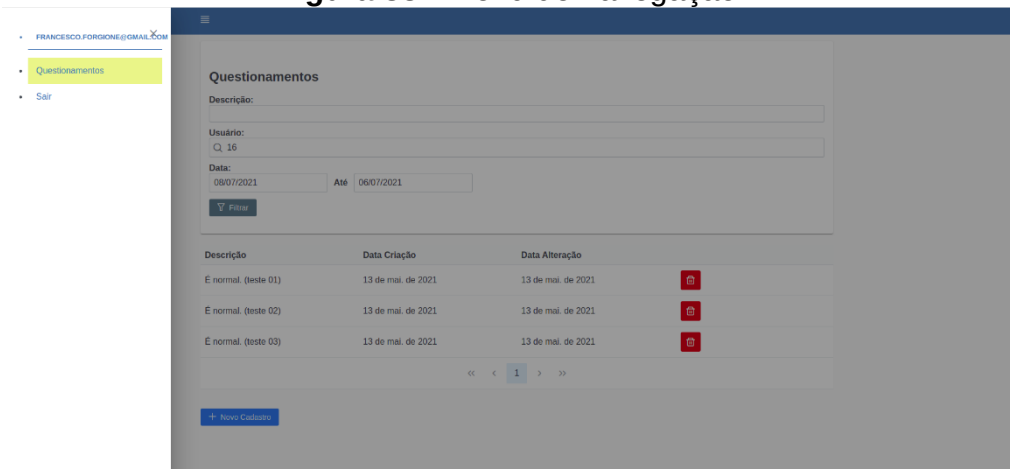
O angular foi desenvolvido com base no conceito de *Single Page Applications* (SPA), no qual uma página web é dividida em vários componentes, a figura 31 contém uma apresentação de forma visual um componente, que basicamente é composto por um HTML, um CSS para estilização e um arquivo de manipulação, escrito em *typescript*. Nesse sentido, quando um usuário navega pela página, os componentes são recarregados de forma *assíncrona*. Evitando com isso recarregar a página por inteiro. Além disso, quem renderiza as informações da página é o próprio *browser* do usuário e não o servidor, isso reduz o tempo que as páginas demoram para carregar uma informação solicitada pelo usuário. Abaixo estão algumas telas desenvolvidas por este projeto utilizando o angular para web.

Figura 32 - Tela de cadastro de questionamentos



Fonte: Própria do autor

Figura 33 - Menu de navegação



Fonte: Própria do autor

Uma grande vantagem de se utilizar um *framework* moderno como o angular é fato de que muitas configurações serem setadas por pré-configuradas *"default"*, reduzindo com isso o trabalho para desenvolvedores, fazendo com que os engenheiros de *software* só se preocupem em desenvolver as regras de negócio da aplicação. No angular uma série de rotas, módulos, *pipes* e diretivas são gerenciadas pelo próprio *framework* reduzindo a complexidade de desenvolvimento e impulsionando que novas páginas e sistemas sejam mais rapidamente desenvolvidos.

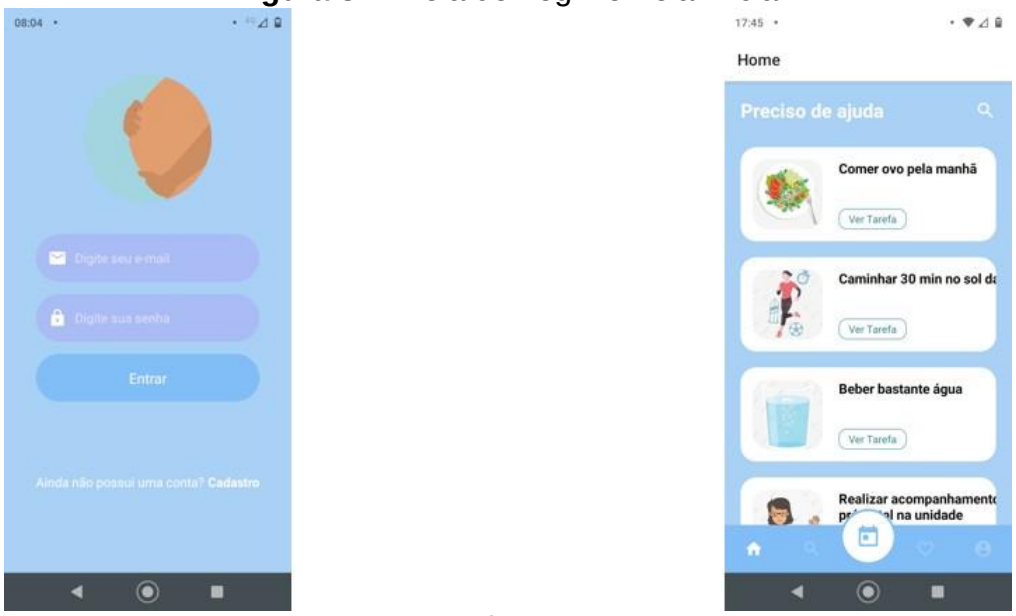
5.8.2 Front-End da Plataforma Mobile

De acordo com Brito *et al.* (2019), o desenvolvimento de aplicativos móveis é composto por três grupos: nativos, híbridos e web. O *react native* é um *framework* desenvolvido pela a empresa Meta com a licença de uso MIT e que se encaixa no grupo dos aplicativos híbridos. Dessa maneira é possível criar com o *react* aplicações multiplataforma utilizando apenas javascript, no processo de *build* da aplicação o *react* transforma o código javascript em código nativo, tornando a utilização mais fluida por parte do usuário.

Para Fentaw (2020) o *react native* foi rapidamente adotado em grandes aplicações na indústria de *software* pelo fato de ser rápido e permitir que um mesmo código funcione em duas plataformas diferentes (IOS e Android), dessa maneira reduzindo custos de desenvolvimento. O *react native* é similar ao *React* (*Framework* para a web), contudo diferente do *React* para a web o *Native* não manipula o Dom Virtual, apenas executa processos em segundo planos no qual se comunicam com

plataforma nativa e converte o código javascript em nativo (AZIZAH *et al.*, 2021).

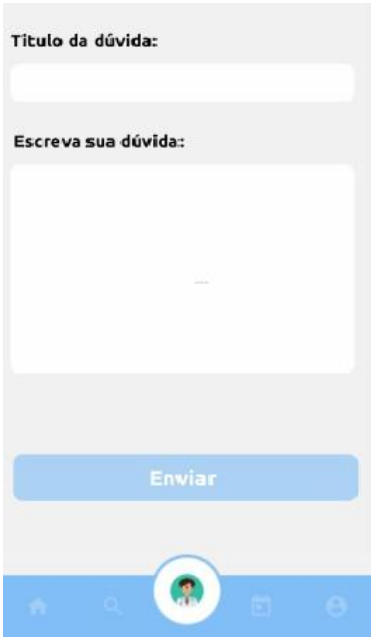
Figura 34 - Tela de Login e Tela Inicial



Fonte: Própria do autor

A primeira imagem da figura 34 representa a tela de login que será utilizada para a usuária se autenticar no aplicativo enquanto que a segunda imagem exibe a tela inicial do aplicativo com a lista de tarefas que a gestante deverá realizar e que foi prescrita anteriormente por um profissional de saúde.

Figura 35 - Tela de Cadastro de Dúvidas



Fonte: Própria do autor

É importante que a gestante que utiliza o aplicativo possa entrar em contato com um profissional especializado através de perguntas, dessa maneira a figura 35 apresenta a tela na qual a gestante poderá enviar dúvidas relativas à gestação.

Figura 36 - Tela de Busca por Dúvidas Frequentes

Fonte: Própria do autor

Dentro do banco de dados do aplicativo já existe uma lista de dúvidas, classificadas de acordo com o tema, que serve para a gestante entender melhor o processo da gestação e dessa forma minimizar os receios vivenciados pelo período gestacional. A figura 36 apresenta a tela na qual é possível consultar as dúvidas.

Figura 37 - Tela de Perfil

Fonte: Própria do autor

A qualquer momento a gestante poderá consultar no perfil do aplicativo alguns dados sobre a gestação, tais como: o número de semanas, data prevista do parto, quantidade de consultas e data prevista para acontecer o parto, conforme a figura 37.

5.9 Considerações do Capítulo

Neste capítulo foram abordadas as principais tecnologias utilizadas para o desenvolvimento do *back-end, front-end* e banco de dados proposta por este projeto. Foi abordado de maneira geral o funcionamento de algumas tecnologias e as principais vantagens de suas utilizações, também foi apresentando os serviços desenvolvidos, respectivamente suas linguagens e uma breve descrição de suas funcionalidades.

6 SEGURANÇA DA INFORMAÇÃO

Neste capítulo serão apresentados os conceitos relacionados à segurança, tal como as principais formas de implementar segurança em *softwares* definidas pelo protocolo *OAuth 2*. Também será apresentada uma breve discussão sobre a Lei Geral de Proteção de Dados. Por fim será abordado as diferenças entre autenticação *Stateful* e *Stateless* e quais benefícios que cada modelo propõe.

6.1 Lei Geral de Proteção de Dados

De acordo com Brasil (2018), a lei geral de proteção de dados (LGPD) que passou a vigorar em agosto de 2020, define uma série de medidas que precisam ser tomadas para que os dados sensíveis dos cidadãos possam ser utilizados de maneira segura, desta forma deve existir o consentimento do usuário de maneira prévia para o armazenamento e demais tratativas de dados pessoais por parte de empresas e organizações. Portanto o cidadão é o dono dos seus dados, podendo revogar o consentimento de utilização do mesmo a qualquer momento e até solicitar a deleção de seus dados de uma determinada plataforma.

A LGPD gerou um impacto significativo na indústria de desenvolvimento de *software*, com isso os novos sistemas computacionais precisam garantir mais segurança e transparência. Neste trabalho, foi utilizado o conceito de *Privacy by Design*, que define uma série de etapas a serem tomadas desde a concepção do *software* até o final do desenvolvimento. De acordo com Cavoukian (2020) o conceito de *Privacy by Design* foi bem aceito por empresas e consumidores e se apoia em sete pilares: Preventivo não corretivo, privacidade como padrão, privacidade incorporada ao projeto, funcionalidade completa, segurança ponta a ponta, visibilidade e transparência e respeito pela privacidade do usuário.

A recomendação da LGPD é pela utilização de técnicas de anonimização de dados, sendo assim, nesta pesquisa, foi feita a opção de que parte dos dados sensíveis como a senha de *login* dos usuários passassem por um processo de criptografia antes de serem salvos na base de dados. Isso aumenta significativamente a segurança, tendo em vista que mesmo que haja vazamento de dados, essas informações criptografadas não estão legíveis.

6.2 Segurança

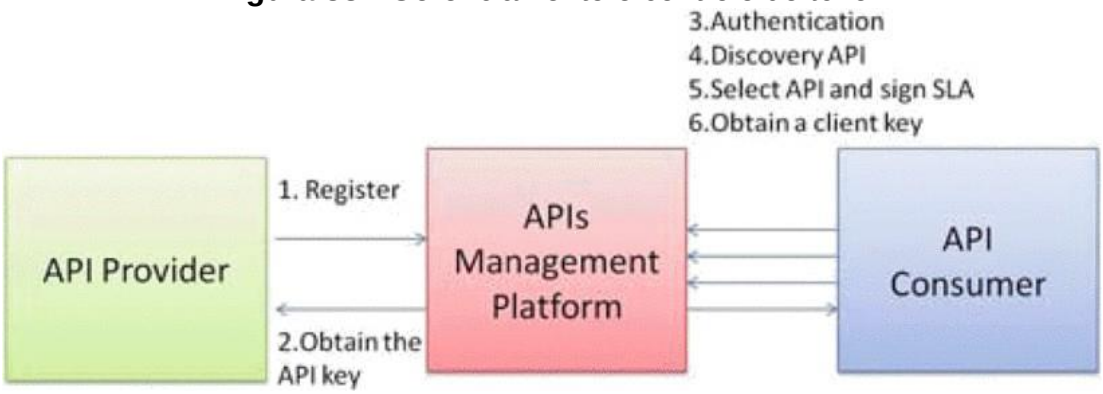
Para qualquer *software* a segurança é algo imprescindível, os arquitetos de sistemas precisam traçar a melhor técnica de garantir a total segurança dos dados trafegados bem como desenvolver maneiras eficientes de comunicação entre clientes e servidores, APIs internas e externas. Implementar segurança em uma API é muito mais complexo do que implementar em aplicações “tradicionais” onde o *back-end* e *front-end* obrigatoriamente estavam dentro de um mesmo servidor, isso se dá pelo fato de que uma API pode ter inúmeros *Client* e esses se encontrem em servidores espalhados pela nuvem.

Os sistemas *web* desenvolvidos no modelo de APIs utilizam o protocolo HTTP para transferência, solicitação e alterações de informações. Com isso são definidas previamente um formato de mensagem para assim abrir a comunicação com uma API. Normalmente é utilizado os formatos XML ou JSON como formato para comunicação e troca de mensagens. Com base nisso Liu e Xu (2012) salientam que a autenticação e autorização não corretamente implementada em uma API pode causar vulnerabilidades críticas tais como: XML malicioso, ataque de entidade externa, injeção de SQL, evocação de serviço indevido e negação de serviço para usuários devidamente autenticados.

Existem algumas formas e conceitos diferentes de implementar autenticação e autorização em REST APIs, contudo o protocolo *OAuth* definido por Hardt (2012) é amplamente utilizado nas mais diversas pesquisas. Atualmente é cada vez mais comum a necessidade de integrar sistemas. Essa integração consiste da necessidade de chamar um serviço externo para executar alguma rotina específica, exemplificando: normalmente em um *software* ERP (Sistema integrado de gestão empresarial) existe a necessidade de chamar uma API externa de um banco para realização de pagamentos, um serviço de e-mail de uma empresa especialista em envio de e-mail *marketing* para consolidar novos clientes entre muitos outros serviços.

Nesse contexto, o usuário que utiliza esse sistema ERP e precisa de uma resposta de um serviço externo, poderia está expondo seus dados pessoais, como senha e *login* de acesso. Ou seja, cada vez que houvesse a necessidade de chamar a API de um banco, ele passava a senha e *login*, isso seria um tanto grave, levando em consideração que essas informações poderiam ser “facilmente” interceptadas na rede e posteriormente usadas para outros fins.

Figura 38 - Gerenciamento e controle de *token*



Fonte: Wu e Lee (2013)

Nesse cenário de vulnerabilidade, o protocolo *OAuth* resolve grande parte dos problemas com a utilização de um *token*. Sendo assim, ao fazer uma requisição para um serviço externo não é mais necessário informar as credenciais que são dados sensíveis como e-mail e senha. O sistema informa apenas um *token* de autorização para o determinado recurso que deseja obter, conforme é exibido na figura 38. Normalmente no *token* é criptografado informações do usuário que deseja obter dados de um determinado recurso ou serviço, por algum algoritmo de criptografia, seja ele de chave simétrica (chave única para cifrar e decifrar) ou assimétrica (chave separada para cifrar e decifrar). Mesmo que esse *token* seja interceptado na rede por algum *software* malicioso, o *token* tem um “prazo de validade” e outras seguranças adicionais tal como uma verificação de dois fatores apresentada na proposta de Sakimura, Bradley e Agarwal (2015) trazendo assim mais segurança às aplicações que compartilham dados sensíveis.

Figura 39 - *Token* descriptografado

Fonte: Própria do autor

Na figura 39 é possível verificar algumas das informações que estão dentro do *token* trafegado entre as requisições, tais como o nome do usuário ao qual o *token* pertence, o escopo, o limite de tempo para expiração, as permissões que o usuário possui dentro da aplicação e uma outra informação extremamente importante o *client* responsável por enviar a solicitação, se esse *client* não for válido a requisição será cancelada e nenhuma informação será trafegada. No cabeçalho do *token* está o algoritmo utilizado para criptografar, justamente com o tipo JWT (JSON Web Token) em referência a Jones, Bradley e Sakimura (2015).

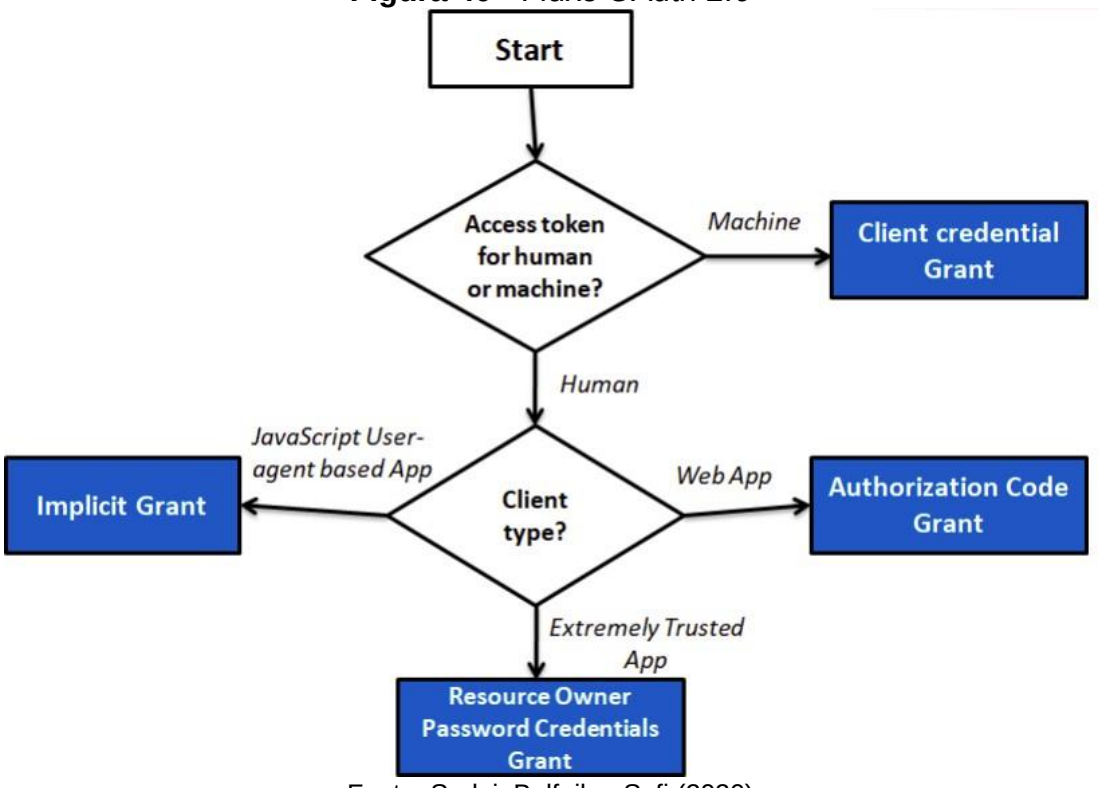
O protocolo *OAuth* 2 define quatro papéis principais de acordo com Hardt (2012):

- **Resource Owner**: O usuário final e dono do recurso. Esse papel fornece os dados e autoriza a aplicação a acessar suas informações.
- **Client**: É o papel responsável por interagir com o *Resource Owner* e *Authorization Server*, o *Client* pode ser um *browser* caso a aplicação for WEB como também um dispositivo móvel caso seja uma aplicação *mobile*.

- **Authorization Server**: O responsável por autenticar e autorizar todas as requisições que queiram acesso ao *Resource Server*. Logo é o *Authorization Server* o responsável pela emissão de *token* de acesso.
- **Resource Server**: É a API de recursos, no caso da plataforma gestação são todos os microsserviços que disponibilizam recursos para o usuário final, durante o processo de autorização o *Resource Server* quando recebe do *client* o *token* de solicitação o mesmo valida o *token* no *Authorization Server*.

O protocolo OAuth 2.0 define quatro fluxos diferentes para autenticação e autorização de acessos.

Figura 40 - Fluxo *OAuth 2.0*



Fonte: Sadqi, Belfaik e Safi (2020)

Nas próximas seções serão apresentados maiores detalhes de cada um dos fluxos expostos na figura 40.

6.2.1 Resource Owner Password Credentials

Uma das grandes vantagens do *OAuth* 2 é a possibilidade de API's acessar

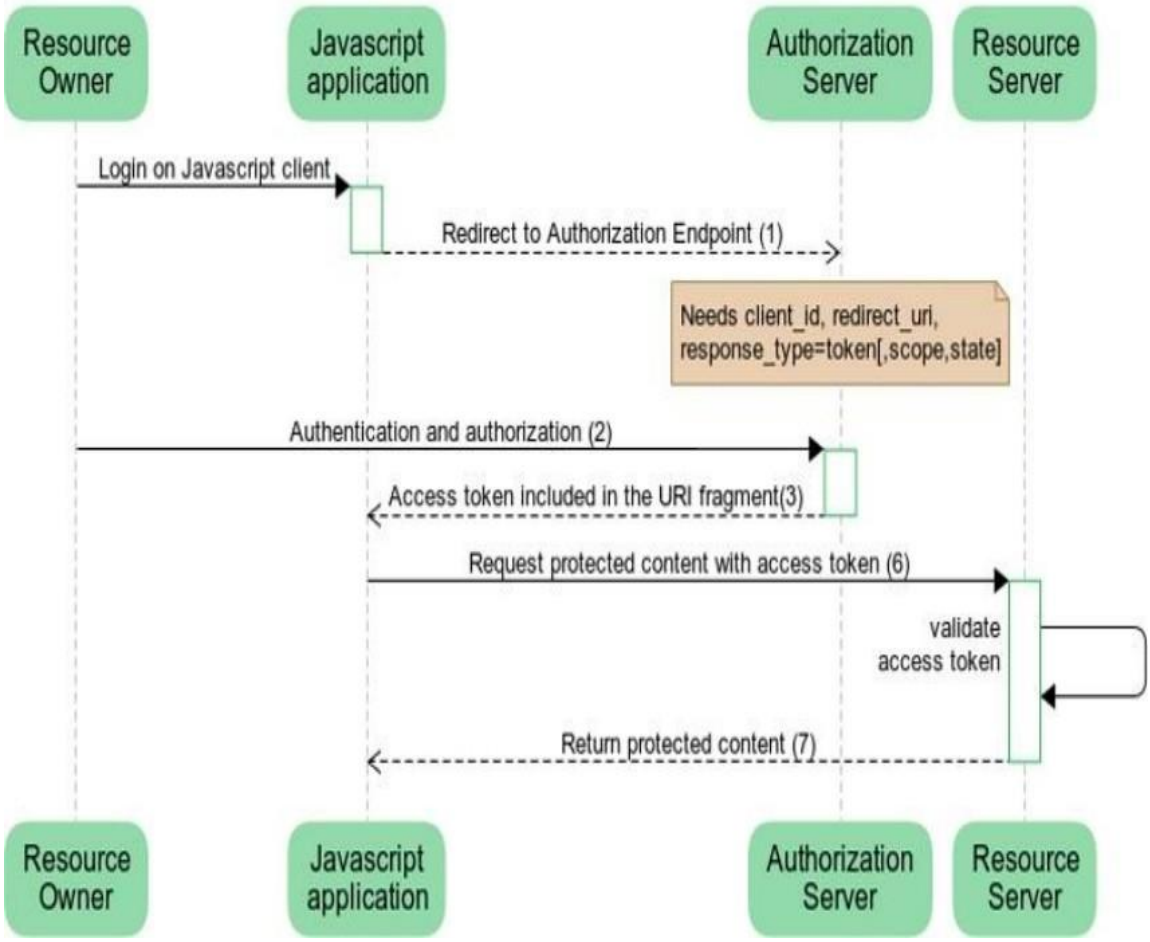
aplicações de terceiros sem conhecer as credenciais do usuário, para isso é necessário um *token* de acesso, nesse *token* é previamente configurado um tempo de expiração. Contudo na RFC 6749 existe a possibilidade de não usar um *token* mas sim as próprias credenciais dos usuários, e-mail e senha por exemplo. Isso é problemático e não é a forma mais segura, exemplificando: quando precisamos fazer um cadastro em algum site e o mesmo pergunta se desejamos logar com a nossa conta do Google por exemplo, em hipótese alguma queremos que o Google forneça nossas credenciais a esse terceiro. Logo no fluxo em questão não é isso que acontece. Sem embargo a RFC 6749 deixa claro que esse fluxo só deve ser utilizado em casos específicos e especiais, ou seja, quando o *client* é um cliente conhecido. Nesse caso, quando uma empresa possui um sistema interno que não é acessado por nenhum cliente externo, e que na maior parte das vezes para se comunicar a esse sistema é necessário um controle de acesso controlado na rede usando VPN (Rede privada virtual).

6.2.2 Client Credentials

O protocolo *OAuth* 2 específica também na RFC 6749 o fluxo *Client Credentials*, esse fluxo garante a autenticação entre a comunicação de servidores, nesse cenário não existe um cliente final. É muito comum que em grandes empresas existam diversos servidores se comunicando para recuperar uma informação necessária solicitada por um *client*, com essa informação recuperada entre os N servidores é necessário montar uma resposta e enviar para o usuário final. Nesse sentido é preciso garantir que as informações trafegadas entre os diferentes servidores estejam seguras, então por isso foi definido o fluxo de autenticação *Client Credentials*, que deve ser usados somente entre transição e comunicação de servidores, nesse fluxo o servidor se autentica usando um usuário próprio e não um usuário de um cliente final. Ou seja, cada servidor se comunica com seus pares utilizando seu próprio e único usuário de acesso.

6.2.3 Implicit grant

Figura 41 - Fluxo *OAuth 2.0 Implicit grant*



Fonte: Sadqi, Belfaik e Safi (2020)

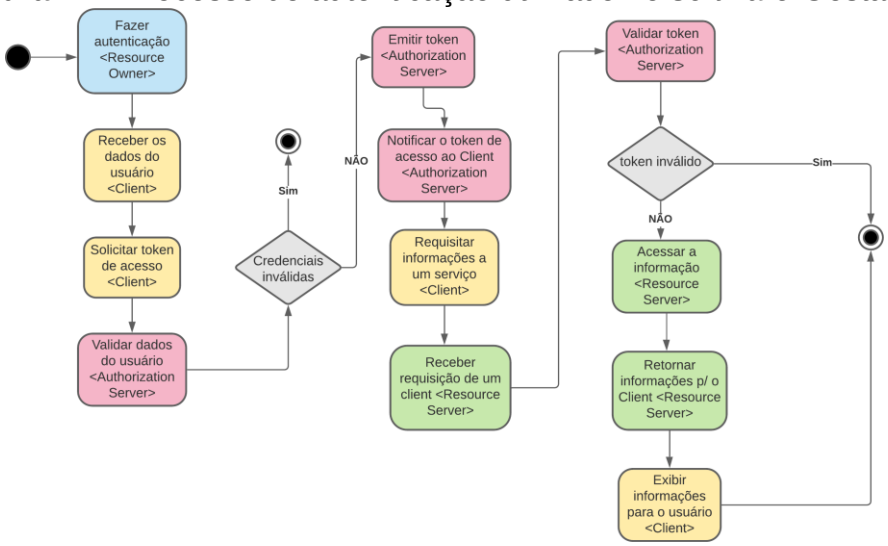
É um fluxo simplificado de autenticação que garante ao usuário acessar recursos através de um *access token* de autorização, nesse fluxo o usuário final é redirecionado ao *Authorization Server*, para autorizar o acesso ao recurso para só então conseguir acessar aquele recurso, ou seja, o *Client* não se comunica com o *Authorization Server* ele apenas redireciona essa comunicação entre o *Resource Owner* e o *Authorization Server*.

6.2.4 Authorization Code

No fluxo do *Authorization Code* toda as requisições, incluindo as chamadas a serviços de terceiro são realizadas usando um *token* de autorização, a única diferença entre o fluxo *Authorization Code* e *Implicit grant* é que nesse fluxo o *Client* se comunica

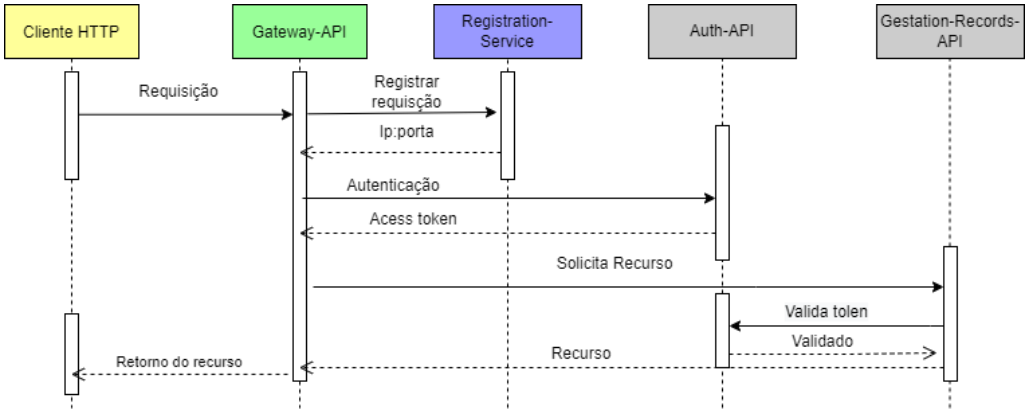
com o *Authorization Server* e recebe um código de autorização, posteriormente o código de autorização é novamente utilizado pelo *Client* para obter o *Access Token* e em seguida executar a solicitação pelo recurso desejado.

Figura 42 - Processo de autenticação utilizado no *software* Gestação



Fonte: Própria do autor

Figura 43 - Processo de cadastro de uma nova Gestante no software Gestação



Fonte: Própria do autor

Neste trabalho foi utilizado o fluxo *Authorization Code* por se tratar de uma aplicação que está centralizada e intermedeia a comunicação de diferentes usuários através de plataformas WEB e *mobile*. No futuro, decorrendo a necessidade de comunicação com API's externas, a arquitetura estará pronta para receber a atualização.

6.2.5 Autenticação Stateful e Stateless

Atualmente é bem comum que empresas utilizem em suas API's um banco de dados no meio da autenticação e autorização para salvar *tokens* emitidos pelo *Authorization Server*. Isso traz benefícios, pois o servidor de autorização terá total controle de quantos e para quais clientes determinados *tokens* de acesso foram emitidos, além de que em um cenário de interceptação de *token* indevida, o servidor poderá rapidamente inativar o *token* interceptado tendo em vista que o mesmo esta´ em banco de dados. Por esse motivo surgiram algumas opções de banco de dados não-relacional, um deles é o Redis, uma estrutura de armazenamento de dados não-relacional, que utiliza um conceito de chave e valor para salvar e emitir informações, que podem ser *tokens* JWT.

Contudo, Fielding e Taylor (2000) definem que uma API precisa ser *Stateless*, ou seja, ela não pode guardar estado nenhum da aplicação. Existe uma motivação importante por trás disso. Considerando um cenário de um ambiente de produção, onde todo o servidor está funcionando corretamente, mas de repente o servidor do banco de dados responsável por registrar e emitir os *tokens* de acesso fica indisponível, nenhum cliente daquela API's poderá usar os recursos por essa disponibilizados, pois nesse contexto de segurança o *token* é repassado em todas as requisições e o *Authorization Server* precisa de um estado de autenticação que é o banco de dados que registra e faz a emissão dos *tokens*.

Na plataforma gestação, proposta neste trabalho, ficou decidido não romper com a restrição proposta por Fielding e Taylor (2000), a plataforma não tem como saber quantos *tokens* foram emitidos e nem para quais usuários foram emitidos. Contudo a cada nova requisição o *Authorization Server* desenvolvido para a aplicação valida todas as informações contidas no *token* transparente JWT emitido para o usuário no momento da autorização conforme delineado pela RFC 7519. Dentro do *Token* Transparente todas as informações emitidas para o usuário são criptografadas e estão seguras. A figura 39 contém um exemplo desse *token*.

6.3 Considerações do Capítulo

Neste capítulo foram abordados os conceitos relacionados à segurança. Foi apresentado as principais formas de implementar segurança em *softwares* definidas

pelo protocolo *OAuth 2*. Por fim, foi abordado as diferenças entre autenticação *Stateful* e *Stateless* e quais benefícios que cada modelo propõe.

7 AVALIAÇÃO DA ARQUITETURA

Este capítulo apresenta as etapas de avaliação de desempenho da Arquitetura Desenvolvida. Inicialmente é discutido sobre a qualidade da arquitetura no que concerne a confiabilidade, eficiência, usabilidade e mantenabilidade. Por fim, o capítulo apresenta os resultados e uma análise experimental de desempenho.

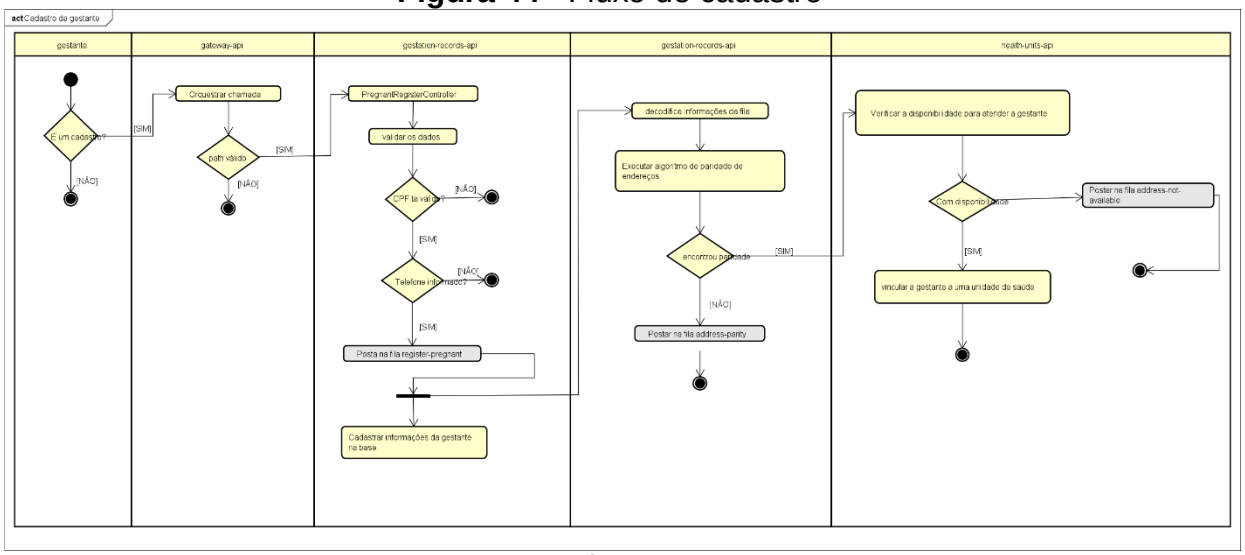
7.1 Quanto à Qualidade da Arquitetura

De acordo com Suwawi, Darwiyanto e Rochmani (2015), para um *software* ser considerado de boa qualidade deve garantir a satisfação dos usuários, dessa forma atendendo o propósito para o qual foi desenvolvido. A norma ISO/IEC 9126 classifica a qualidade de um *software* em seis pilares: Funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade. Embora tenha sido realizada uma extensa revisão bibliográfica acerca do pré-natal no SUS e entrevista com profissional de saúde, para desta forma propor as funcionalidades existentes na aplicação desenvolvida neste trabalho, não será considerado o parâmetro de funcionalidade na avaliação de qualidade.

7.2 Confiabilidade e Eficiência

A arquitetura desenvolvida foi pensada e projetada para ser tolerante a falhas, para tal é preciso garantir que mesmo diante de uma falha a arquitetura continuará provendo os serviços corretamente. Para esta finalidade foi preciso utilizar diversas técnicas computacionais. As regras de negócios foram adaptadas para tratar falhas, na figura 44 é possível observar que no fluxo de cálculo de paridade de endereço, o algoritmo pode não ser capaz de realizar corretamente o cálculo, com isso é enviado uma notificação a uma *broker* de mensageria, para posteriormente ser tratado por um humano. Também são realizados *rollback* em caso de falhas, isso evita a inconsistência de dados. Além da utilização de temporizador, para identificar casos onde o sistema está onerando.

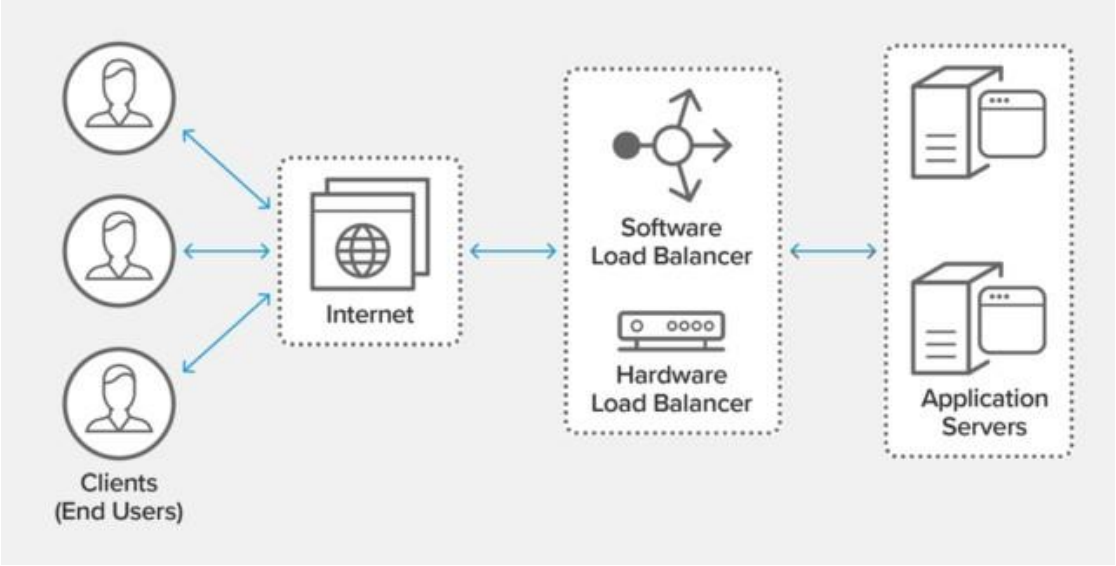
Figura 44 - Fluxo de cadastro



Fonte: Própria do autor

Outro recurso importante utilizado e que é fornecido pelo Docker, plataforma utilizado para criar contêineres é o *load balance*, neste projeto foi utilizado a técnica de *load balance Slave* na qual criar servidores intermediários à medida que o servidor principal já não consegue responder a todas as requisições. A técnica é eficiente e evita gastar recursos computacionais sem a devida necessidade.

Figura 45 - *Load Balancing*



Fonte: NGINX (2020)

Outra tentativa realizada para garantir a confiabilidade da arquitetura, foi a utilização de *fallbacks*, que são responsáveis por direcionar uma requisição para uma segunda opção, caso um serviço para o qual a requisição havia sido direcionada não

esteja disponível. Para isso, foi utilizado em conjunto o padrão de projeto *circuit breaker*, que é responsável por fechar o circuito responsável por receber requisições em caso de um serviço ou um API esteja com falha.

7.2.1 Usabilidade

Uma das principais características de qualidade de um *software* é a usabilidade. Pois a usabilidade está diretamente relacionada ao êxito da execução das atividades por parte dos usuários. Dessa maneira, é preciso garantir que um *software* responda corretamente aos *inputs* dos usuários, tornando a experiência de uso agradável. De acordo com Nielsen (1994a), a usabilidade engloba todos os envolvidos com o desenvolvimento de um *software*, isso implica em escrever código, conhecer o usuário e o contexto de uso.

Além das técnicas utilizadas para melhorar o desempenho da aplicação, neste projeto foram aplicadas as Regras de Ouro de Shneiderman *et al.* (2016), isso compreende: Manter a consistência da aplicação, diferenciar os diferentes tipos de usuários e projetar sistemas que atendam às diferentes necessidades, informar sobre erros, sequências que indiquem que determina ação foi concluída, evitar e tratar erros, reverter ações, permitir que os usuários tenham controle da interface e evitar que os usuários precisem memorizar ao invés de reconhecer.

7.2.2 Manutenibilidade

Um fator importante a ser considerado em qualquer projeto de *software* é a mantenabilidade, é necessário projetar uma arquitetura que garanta a qualidade do *software*, dessa forma evitando erros que podem custar caro a sua correção. A

decisão de qual arquitetura utilizar precisa ser tomada considerando vários pontos, entre esses: requisitos funcionais e não-funcionais. Outra medida importante para assegurar a qualidade é o desenvolvimento de testes, que devem acontecer desde do início do desenvolvimento até a fase final do projeto. O *Test Driven Development* (TDD) que é o desenvolvimento orientado a testes vem crescendo na indústria de desenvolvimento, garantindo dessa forma uma ampla cobertura de testes.

De acordo com Deng, Dehlinger e Chakraborty (2020), testar um *software* custa caro e em certos casos

o gasto com testes é responsável por 50% do orçamento de um projeto. Contudo, é extremamente importante a realização de testes ao longo e posteriormente a entrega de um *software*, para evitar que problemas maiores ocorram depois que os usuários já estejam usando a solução.

Durante o desenvolvimento deste trabalho foram realizados diversos testes, sobre- tudo testes automatizados. Para a realização de testes unitários, foi utilizado o tradicional *framework* jUnit. Os testes unitários são importantes para validar determinadas rotinas de um sistema, validando com isso, entradas e saídas. É importante que em projetos que envolvam a complexidade de uma arquitetura de microsserviços se realizem testes de serviços e integração, sendo assim neste projeto foi utilizado a biblioteca *Rest Assured* que tem como finalidade realizar testes de chamadas a métodos HTTP. Também foi realizado testes baseados em comportamentos (BDD), para Bennett (2021) o BDD é uma evolução do TDD sendo assim, foram realizados utilizando a biblioteca mockito, responsável por criar *mocks* de determinadas chamadas e comportamentos.

7.2.3 Portabilidade

O módulo WEB que será utilizado pelos profissionais de saúde não precisa ser instalado e funciona em qualquer navegador (*Browser)* com suporte ao O ECMAScript 6. O aplicativo *mobile* que será utilizado pela gestante foi desenvolvido com react *native*, contudo inicialmente apenas existe uma versão para ser instalado em *smartphones* android, mas no futuro pode ser extensível para IOS.

7.3 Quanto ao Desempenho

Para analisar o desempenho de uma aplicação é preciso aferir diversos parâmetros, entre os quais a latência e o *throughput*. Dessa maneira foram realizados alguns testes para averiguar algumas métricas de alguns serviços importantes desenvolvidos nas API’s que compõem a arquitetura desenvolvida neste trabalho. Os serviços avaliados foram: Login, acesso a lista de tarefas, acesso a lista de perguntas e acesso ao perfil. Os testes foram realizados utilizando o Apache JMeter que simula requisições HTTP de usuários simultâneos acessando uma aplicação.

Tendo em vista o custo para provisionar serviços na nuvem, foi feita a opção

por realizar os testes utilizando a máquina com menor potência disponível na *cloud* Amazon Web Services. Sendo assim, é importante enfatizar que o desempenho da aplicação será impactado, considerando que os recursos computacionais utilizados foram reduzidos a configurações mínimas, ainda assim, a capacidade de adicionar recursos a uma instância na nuvem faz com que a aplicação seja altamente escalável.

O quadro 3 apresenta as configurações da instância *Elastic Compute Cloud* utilizada para a realização dos testes. É importante enfatizar que um teste de desempenho realizado em uma instância pode apresentar valores completamente distintos quando realizado em uma outra instância que contenha um sistema operacional ou uma tecnologia de memória diferente, entre outros aspectos.

Quadro 3 - Configurações da Máquina Utilizada

Tipo de Instância	vCPU	Memória Ram (GB)	Preço Médio por Hora
t4g.micro	2	1	0,07 USD

Inicialmente foi definido um valor aceitável de erros, foram encontrados na literatura alguns trabalhos que realizaram validações semelhantes à esta. No estudo desenvolvido por Falcão (2014) foi considerado aceitável um valor médio de 10% de falhas, o experimento também foi validado na *Cloud* da Amazon e foi utilizada uma instância com melhor desempenho e processamento quando comparada a instância utilizada por este trabalho. Já no estudo conduzido por Lopes (2021), foi aceitável uma média de falhas de apenas 0.1% do número de requisições. No experimento não foram mapeadas as configurações do servidor de teste utilizado, o autor apenas escalava os microsserviços a cada vez que a aplicação apresentava falhas superiores ao valor desejado e este escalonamento foi aplicado até que a aplicação pudesse receber 5 mil usuários simultaneamente sem apresentar falhas. Dado que somente estava a disposição para a utilização a instância utilizada no quadro 3, foi considerado um valor médio de erros de 0.1%. Dessa maneira, nesta validação foi verificado quantos usuários simultaneamente poderiam utilizar a aplicação. Assim sendo, foram realizados 3 cenários de teste.

No primeiro cenário de teste foi verificado como a aplicação se comportaria recebendo um total de 8000 requisições, sendo disparadas por um total de 2000 usuários simultâneos. O quadro 4 apresenta os dados do teste.

Quadro 4 - Cenário #1

Cenário #1	Tempo(60s)/instância	Amostra	Latência	Qtd. Erros
/login	2	2000	3033ms	27%
/tarefas/listar	2	2000	2733ms	17,5%
/duvidas/listar	2	2000	2013ms	14,3%
/perfil/id	1	2000	178ms	0%
Total	7	8000	7957ms	58,8
Média	3,5	2000	1989,25ms	14,7

No primeiro teste realizado a quantidade média de erros foi de 14,7%, a maior causa dos erros foi devido aos *status code* 502 bad *gateway* e 504 *gateway timeout*. O serviço que apresentou maior lentidão, tal como erro, foi o serviço responsável por autenticar o usuário na plataforma. Na autenticação do usuário, seja um profissional de saúde ou uma gestante, são realizados uma série de validações de cadastro, com isso o fluxo de autenticação, gera maior “gargalo” ao sistema, contudo uma vez autenticado na aplicação o *token* gerado tem uma validade de expiração, com isso não há a necessidade de o usuário precisar sempre realizar o login. Possivelmente se fosse uma arquitetura monolítica a aplicação já ficaria indisponível com a tamanha quantidade de requisições recebidas.

No segundo cenário simulado, foi verificado que a quantidade média de erros foi reduzida de 14,7% para 3,18%. Neste cenário, a quantidade total de requisições foi reduzida para 6000, sendo disparadas por 1500 usuários simultâneos.

Quadro 5 - Cenário #2

Cenário #2	Tempo(60s)/instância	Amostra	Latência	Qtd. Erros
/login	2	1500	189ms	9%
/tarefas/listar	2	1500	97ms	3,5%
/duvidas/listar	2	1500	23ms	0,23%
/perfil/id	1	1500	17ms	0%
Total	7	6000	326ms	12,73
Média	3,5	1500	81,5ms	3,1825

A taxa de erro nesse cenário continua acima do valor esperado, contudo houve considerável melhora quando comparada com o cenário 1. Neste momento, os serviços que precisaram ser escalonados para segunda máquina, fizeram pouco uso de processamento e, consequentemente, conseguiram ser concluídos com uma melhor *performance* quando comparados com o primeiro cenário. Contudo, a quantidade de instâncias permaneceram iguais aos testes realizados no cenário 1.

No terceiro cenário, foi simulado que 1100 usuários simultâneos estavam

utilizando a aplicação, onde 4000 requisições no total estavam sendo feitas. Dessa forma, a quantidade média de erros, ficou dentro da margem esperada 0,1% e o número de instâncias (máquinas) foi reduzido para 6 ao invés de 7 utilizadas nos outros cenários de teste.

Quadro 6 - Cenário #3

Cenário #3	**Tempo(60s)/instância**	**Amostra**	**Latência**	**Qtd. Erros**
/login	2	1100	141ms	0,3%
/tarefas/listar	2	1100	58ms	0,1%
/duvidas/listar	1	1100	33ms	0%
/perfil/id	1	1100	23ms	0%
Total	6	4400	255ms	0,4%
M édia	1,5	1100	63,75ms	0,1%

O serviço de autenticação (*Login*) e Listar dúvidas e tarefas continuaram utilizando duas instâncias, contudo a segunda instância utilizada, quase não estava com carga. O *Load Balancer* foi configurado para que uma nova instância fosse criada a cada vez que qualquer uma das instâncias utilizadas chegasse a 70% de processamento.

7.4 Considerações do Capítulo

Neste capítulo apresentamos a avaliação de desempenho da Arquitetura de Microsserviços Desenvolvida. Foram selecionados alguns parâmetros para o teste de desempenho, onde foram identificados gargalos e melhorias na arquitetura, apesar disso foi verificado que o comportamento da arquitetura se manteve estável e com um desempenho aceitável no ambiente simulado.

8 CONCLUSÃO

Foi abordado a concepção e implementação de uma arquitetura de microsserviços voltada para uma aplicação no âmbito da saúde. Apesar de se tratar de um tema recente na indústria e na academia, o modelo arquitetura de microsserviços desempenham um papel importante no desenvolvimento de aplicações robustas, que precisam escalar com velocidade e atender a uma demanda substancial de usuários. Esta pesquisa propôs o desenvolvimento de uma arquitetura de microsserviços baseado no modelo de orquestração e o principal objetivo foi o desenvolvimento de um *software* consistente e escalável.

A maioria dos estudos que propõem uma aplicação voltada para atender o público de gestantes, atende aos requisitos funcionais, contudo não há demonstrações de *design* arquitetural, tão pouco de confiabilidade e escalabilidade. Dado o volumoso número de possíveis usuárias para tal aplicação torna-se praticamente inviável a utilização de uma aplicação com uma arquitetura monolítica. O desenvolvimento desta pesquisa constatou o potencial da utilização de uma arquitetura descentralizada, dado os resultados apresentados no capítulo anterior. Além disso, é importante considerar que uma arquitetura de microsserviços torna mais fácil a evolução tal como a manutenção de um *software*.

Os resultados foram parcialmente alcançados, o desenvolvimento da arquitetura proposta foi concluída e devidamente registrada sob o número BR512022000797-5. As devidas integrações com as tecnologias utilizadas foram realizadas, contudo as interfaces WEB e *Mobile* por limitação do tempo não foram totalmente finalizadas.

As principais contribuições deste trabalho são descritas a seguir:

- Uma arquitetura de microsserviços para suporte ao pré-natal;
- Permitir um acompanhamento mais humanizado da gestante;
- Identificar com mais rapidez eventuais problemas que possam ocorrer na gestação;
- Reduzir a distância entre as gestantes e os profissionais de saúde;
- Reduzir medos e receios provindos da gestação;
- Levantamento de dados que podem contribuir para políticas públicas de saúde.

Como trabalho futuro, planeja-se:

- **Algoritmo de Paridade**: Adicionar inteligência artificial ao algoritmo de paridade, o responsável por vincular a gestante a unidade de saúde mais próxima.

- **Adicionar Recomendação de Conteúdo**: Na lista de dúvidas, onde a gestante procura sanar inseguranças resultante da gestação é pretendido adicionar um algoritmo de recomendação, que com base no que a gestante pesquisa, vai identificando o perfil e recomenda conteúdo com base nele.
- **Utilização de Kubernetes**: Neste trabalho foi utilizado Docker no modo Swarm para orquestrar os *containers*, contudo no futuro é interessante testar a solução com o gerenciamento de *containers* realizado pelo Kubernetes.

REFERÊNCIAS

AL-DEBAGY, O.; MARTINEK, P. A comparative review of microservices and monolithic architectures. In: **2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)**. [S.l.: s.n.], 2018. p. 149-154.

ANDERSON, C. Docker [software engineering]. **IEEE Software, IEEE Computer Society**, Los Alamitos, CA, USA, v. 32, n. 03, p. 102-c3, may 2015. ISSN 1937-4194.

AZIZAH, A. H.; FAIDAH, S. Z.; ULUM, M. B.; HANDAYANI, P. Exploration of react native framework in designing a rule-based application for healthy lifestyle education. In: **2021 1st International Conference on Computer Science and Artificial Intelligence (ICCSAI)**. [S.l.: s.n.], 2021. v. 1, p. 391-394.

BANDARA, C.; PERERA, I. Transforming monolithic systems to microservices - an analysis toolkit for legacy code evaluation. In: **2020 20th International Conference on Advances in ICT for Emerging Regions (ICTer)**. [S.l.: s.n.], 2020. p. 95-100.

BARROS, F. C.; QAR, Z.; BHUTTA, A.; BATRA, M.; HANSEN, T. N.; VICTORA, C. G.; RUBENS, C. E. **Global report on preterm birth and stillbirth (3 of 7):** evidence for eff ectiveness of interventions BMC Pregnancy and Childbirth. [S.l.], 2010. v. 10, n. 1, 3 p. Disponível em: <http://www.biomedcentral.com/1471-2393/10/S1/S3>.

BECK, B.; BENNEKUM, M. Manifesto para desenvolvimento Ágil de software. In: **Manifesto para desenvolvimento Ágil de software**. 2001.

BENNETT, B. E. A practical method for api testing in the context of continuous delivery and behavior driven development. In: **2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)**. [S.l.: s.n.], 2021. p. 44-47.

BHATTI, N.; BOUCH, A.; KUCHINSKY, A. Integrating user-perceived quality into web server design. In: **Proceedings of the 9th International World Wide Web Conference on Computer Networks:** The International Journal of Computer and Telecommunications Netowrking. NLD: North-Holland Publishing Co., 2000. p. 1-16.

BONE´R, J. **Reactive Microsystems:** The Evolution of Microservices at Scale. O'Reilly Media, Incorporated, 2017. ISBN 9781491994368. Disponível em: <https://books.google.com.br/books?id=sL0LtAEACAAJ>.

BRANDÃO, D. **Log e Monitoramento de Microsserviços**. 2020. Disponível em: <https://medium.com/@dcruzb/log-e-monitoramento-de-microsseviC3A7os-11c465349a77> . Acesso em: 8 fev. 2022.

BRASIL. **Lei Nº 13.709, de 14 de agosto de 2018**. 2018. Disponível em: <http://www.planalto.gov.br/ccivil 03/ ato2015-2018/2018/lei/l13709.htm>. Acesso em: 30 dez. 2021.

BRITO, H.; SANTOS, A.; BERNARDINO, J.; GOMES, A. Mobile development in swift, java and react native: an experimental evaluation in audioguides. In: **2019 14th Iberian Conference on Information Systems and Technologies (CISTI)**. [S.l.: s.n.], 2019. p. 1-6.

BROEKHUIS, M.; VELSEN, L. van; HERMENS, H. Assessing usability of eHealth technology: A comparison of usability benchmarking instruments. **International Journal of Medical Informatics,** v. 128, p. 24-31, 2019. ISSN 1386-5056. Disponível em: <https://www.sciencedirect.com/science/article/pii/S138650561930067X>.

BUTLER TOBAH, Y. S.; LEBLANC, A.; BRANDA, M. E.; INSELMAN, J. W.; MORRIS, M. A.; RIDGEWAY, J. L.; FINNIE, D. M.; THEILER, R.; TORBENSON, V. E.; BRODRICK, E. M.; Meylor de Mooij, M.; GOSTOUT, B.; FAMUYIDE, A. Randomized comparison of a reduced-visit prenatal care model enhanced with remote monitoring. **American Journal of Obstetrics and Gynecology**, v. 221, n. 6, p. 638.e1-638.e8, 2019. ISSN 0002-9378. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0002937819308105>.

CABRERA, E.; C´aRDENAS, P.; CEDILLO, P.; PES´aNTEZ-CABRERA, P. Towards a methodology for creating internet of things (iot) applications based on microservices. In: **2020 IEEE International Conference on Services Computing (SCC).** [S.l.: s.n.], 2020. p. 472-474.

CARROLI, G.; ROONEY, C.; VILLAR, J. A. **How effective is antenatal care in preventing maternal mortality and serious morbidity? An overview of the evidence**. 2001.

CAVOUKIAN, A. Understanding how to implement privacy by design, one step at a time. **IEEE Consumer Electronics Magazine**, v. 9, n. 2, p. 78-82, 2020.

CHEW, L.; BRADLEY, K.; BOYKO, E. Brief questions to identify patients with inadequate health literacy. **Family medicine**, v. 36, p. 588-94, 10 2004.

CITO, J.; FERME, V.; GALL, H. **Using docker containers to improve reproducibility in software and web engineering research**. [S.l.: s.n.], 2016. p. 609-612. ISBN 978-3-319-38790-1.

COIMBRA, L. C.; SILVA, A. A. M.; MOCHEL, E. G.; ALVES, M. T. S. S. B.; RIBEIRO, V. S.; ARAGA˜O, V. M. F.; BETTIOL, H. Fatores associados à inadequação do uso da assistência pré-natal. **Revista de Saúde Pública**, Faculdade de Saúde Pública da Universidade de São Paulo, v. 37, n. 4, p. 456-462, 2003. ISSN 0034-8910. Disponível em: <http://www.scielo.br/j/rsp/a/Jwpw8dGyCS3cGnL6JLsmYJg/?lang=pt>.

COUTINHO, T.; MONTEIRO, M. F. G.; SAYD, J. D.; TEIXEIRA, M. T. B.; COUTINHO, C. M.; COUTINHO, L. M. Monitoramento do processo de assistência pré-natal entre as usuárias do Sistema Único de Saúde em município do Sudeste brasileiro. **Revista Brasileira de Ginecologia e Obstetrícia**, Federação Brasileira das Sociedades de Ginecologia e Obstetrícia, v. 32, n. 11, p. 563-569, nov. 2010.

ISSN 0100-7203. Disponível em: <http://www.scielo.br/j/rbgo/a/dt3P7j79Hqr7MfWtLNMZ8FJ/?lang=pt>.

DENG, L.; DEHLINGER, J.; CHAKRABORTY, S. Teaching software testing with free and open source software. In: **2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)**. [S.l.: s.n.], 2020. p. 412-418.

DEWI, D. A.; KUSUMA, T. M.; JUNATAS, B.; TRIHASTA, D. Case study: The condition of ubiquitous computing application in indonesia. In: **2009 WRI World Congress on Software Engineering.** [S.l.: s.n.], 2009. v. 1, p. 260-263.

DINH-LE, C.; CHUANG, R.; CHOKSHI, S.; MANN, D. Wearable health technology and electronic health record integration: Scoping review and future directions. **JMIR Mhealth Uhealth**, v. 7, n. 9, p. e12861, Sep 2019. ISSN 2291-5222. Disponível em: <https://mhealth.jmir.org/2019/9/e12861/>.

DREY, Z.; CONSEL, C. A visual, open-ended approach to prototyping ubiquitous computing applications. In: **2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops).** [S.l.: s.n.], 2010. p. 817-819.

EFREM, I.; HØYVIK, M.; HELDAL, I.; HELGESEN, C.; KOVARI, A.; KATONA, J.; COSTESCU, C.; ROSAN, A.; HATHAZI, A.; DEMETER, R.; THILL, S. Mobile application helps planning activities during pregnancy. In: **2019 10th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)**. [S.l.: s.n.], 2019. p. 339-342.

ESCOTT, K.R.; NOBLE, J. Design patterns for angular hotdraw. In: **Proceedings of the 24th European Conference on Pattern Languages of Programs**. New York, NY, USA: Association for Computing Machinery, 2019. (EuroPLop '19). ISBN 9781450362061. Disponível em: <https://doi-org.ez121.periodicos.capes.gov.br/10.1145/3361149.3361185>.

FACUNDO, S. H. B. C. **Tecnologias comunicacionais aplicadas no pré-natal**. 2016. Mestrado Em Saúde Coletiva, UNIFOR (Universidade de Fortaleza), Fortaleza, Brasil.

FALCÃO, E. d. L. **Deaf Accessibility as a Service**: uma arquitetura escalável e tolerante a falhas para o sistema de tradução VLIBRAS. Tese (Doutorado) — Universidade Federal da Paraíba, 2014.

FAN, C.Y.; MA, S.P. Migrating monolithic mobile application to microservice architecture: An experiment report. In: **2017 IEEE International Conference on AI Mobile Services (AIMS)**. [S.l.: s.n.], 2017. p. 109-112.

FENTAW, A. E. **Cross platform mobile application development:** a comparison study of React Native Vs Flutter. 2020. UNIVERSITY OF JYVA SKYLA. FACULTY OF INFORMATION TECHNOLOGY.

FERREIRA, A. B. de H. **Miniaurélio:** o dicionário da língua portuguesa. 8 ed. [S.l.]: Positivo, 2008.

FIELDING, R. T.; TAYLOR, R. N. **Architectural Styles and the Design of Network-Based Software Architectures.** Tese (Doutorado), 2000. AAI9980887.

FOWLER, M. **Design Principles and Design Patterns**. 2000. Disponível em: <https://fi.ort.edu.uy/innovaportal/file/2032/1/design principles.pdf>. Acesso em: 26 set. 2021.

FOWLER, M. **Microservices Guide**. 2014. Disponível em: <https://martinfowler.com/microservices/>. Acesso em: 16 ago. 2021.

GERBER, A. **The state of containers and the docker ecosystem:** 2015. 1 ed. [S.l.]: O'Reilly Media, Inc., 2015.

GOODMAN. **Introduction to health care technology assessment:** ten basic steps. *1998*. 1998. Disponível em: <http://www.nlm.nih.gov/nichsr/ta101>. Acesso em: 11 ago. 2021.

GOS, K.; ZABIEROWSKI, W. The comparison of microservice and monolithic architecture. In: **2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)**. [S.l.: s.n.], 2020. p. 150-153.

G éHBERGER, D.; M´aTRAY, P.; N éMETH, G. Data-driven monitoring for cloud compute systems. In: **2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)**. [S.l.: s.n.], 2016. p. 128-137.

HARDT, D. **The OAuth 2.0 Authorization Framework**. RFC Editor, 2012. RFC 6749. (Request for Comments, 6749). Disponível em: <https://rfc-editor.org/rfc/rfc6749.txt>.

HONG, X. J.; YANG, H. S.; KIM, Y. H. Performance analysis of restful api and rabbitmq for microservice web application. In: **2018 International Conference on Information and Communication Technology Convergence (ICTC)**. [S.l.: s.n.], 2018. p. 257-259.

HUSSAIN, S.; KEUNG, J.; KHAN, A. A. The effect of gang-of-four design patterns usage on design quality attributes. In: **2017 IEEE International Conference on Software Quality, Reliability and Security (QRS).** [S.l.: s.n.], 2017. p. 263-273.

JAGADEESAN, L. J.; MENDIRATTA, V. B. When failure is (not) an option: Reliability models for microservices architectures. In: **2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW).** [S.l.: s.n.], 2020. p. 19-24.

JAIN, P.; SHARMA, A.; AHUJA, L. **The impact of agile software development process on the quality of software product.** p. 812-815, 2018.

JEROME, B.; KAZMAN, R. **Surveying the solitudes**: An investigation into the relationships between human computer interaction and software engineering in practice. [S.l.: s.n.], 2005. p. 59-70. ISBN 978-1-4020-4027-6.

JONES, M.; BRADLEY, J.; SAKIMURA, N. **JSON Web Token (JWT)**. RFC Editor, 2015. RFC 7519. (Request for Comments, 7519). Disponível em: <https://rfc-editor.org/rfc/rfc7519.txt>.

KIM, D. G.; WILLIS, J. **The devops handbook**: How to create world-class agility reliability and security in technology organizations. 2016.

KIM, S.; YEOM, S.; KWON, O.J.; SHIN, D.; SHIN, D. Ubiquitous healthcare system for analysis of chronic patients' biological and lifelog data. **IEEE Access**, v. 6, p. 8909-8915, 2018.

KLYMASH, M.; TCHAIKOVSKYI, I.; HORDIICHUK-BUBLIVSKA, O.; PYRIH, Y. Research of microservices features in information systems using spring boot. In: **2020 IEEE International Conference on Problems of Infocommunications. Science and Technology (PICST)**. [S.l.: s.n.], 2020. p. 507-510.

KRÜGER, J. Tackling knowledge needs during software evolution. In: **Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2019. (ESEC/FSE 2019), p. 1244-1246. ISBN 9781450355728. Disponível em: <https://doi-org.ez121.periodicos.capes.gov.br/10.1145/3338906.3342505>.

LANSKY, S.; FRICHE, A. A. d. L.; SILVA, A. A. M.; CAMPOS, D.; BITTENCOURT, S. D. d. A.; CARVALHO, M. L. de; FRIAS, P. G. de; CAVALCANTE, R. S.; CUNHA, A. J. L. A. da. Pesquisa Nascer no Brasil: perfil da mortalidade neonatal e avaliação da assistência à gestante e ao recém-nascido. **Cadernos de Saúde Pública**, Escola Nacional de Saúde Pública Sergio Arouca, Fundação Oswaldo Cruz, v. 30, n. SUPPL1, p. S192-S207, 2014. ISSN 0102-311X. Disponível em: <http://www.scielo.br/j/csp/a/Ss5zQXrmrGrGJvcVMKmJdqR/?lang=pt>.

LIAROPOULOS, L. **Letter to the editor do we need 'care' in technology assessment in health care?** 1997. Disponível em: <https://doi.org/10.1017/S0266462300010291>.

LIMA-PEREIRA, P.; BERMU´DEZ-TAMAYO, C.; JASIENSKA, G. Use of the Internet as a source of health information amongst participants of antenatal classes. **Journal of Clinical Nursing**, v. 21, n. 3-4, p. 322-330, 2012. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-2702.2011.03910.x>.

LIU, K.; XU, K. Oauth based authentication and authorization in open telco api. **2012 International Conference on Computer Science and Electronics Engineering**, v. 1, p. 176-179, 2012.

LOPES, T. R. **Método de migração de sistemas monolíticos legados para a arquitetura de microsserviços**. Tese (Doutorado) – Universidade de Brasília, 2021.

MACARTHY, R. W.; BASS, J. M. **An empirical taxonomy of devops in practice**. p. 221-228, 2020.

MANUJA, M.; MANISHA. Moving agile based projects on cloud. In: **2014 IEEE International Advance Computing Conference (IACC).** [S.l.: s.n.], 2014. p. 1392-1397.

MARIE-MAGDELAINE, N.; AHMED, T.; ASTRUC-AMATO, G. Demonstration of an observability framework for cloud native microservices. In: **2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM).** [S.l.: s.n.], 2019. p. 722-724.

MCCANN, A. D.; MCCULLOCH, J. E. Establishing an Online and Social Media Presence for Your IBCLC Practice. **Journal of Human Lactation**, v. 28, n. 4, p. 450-454, 2012. Disponível em: <https://doi.org/10.1177/0890334412461304> .

MERHY, E. **Saúde**: a cartografia do trabalho vivo. 3 ed. ed. [S.l.]: Editora Hucitec, 2002.

MILES, R. **Antifragile Software Building Adaptable Software with Microservices**. [s.n.], 2016. Disponível em: <https://leanpub.com/antifragilesoftware>.

MINISTÉRIO DA SAÚDE, B. Secretaria de Atenção à Saúde. Departamento de Ações Programáticas Estratégicas. **Política Nacional de Atenção Integral à Saúde da Mulher**: plano de ação 2004-2007. 2004.

MINISTÉRIO DA SAÚDE, B. Pré-natal e Puerpério: atenção qualificada e humanizada – manual técnico. 2006. Disponível em: <https://bvsms.saude.gov.br/bvs/publicacoes/manualprenatalpuerperio3ed.pdf>.

MINISTÉRIO DA SAÚDE, B. **Portaria Nº 1.459**, de 24 de junho de 2011. Institui, no âmbito do Sistema Único de Saúde - SUS - a Rede Cegonha. 2011. Disponível em: <https://bvsms.saude.gov.br/bvs/saudelegis/gm/2011/prt1459 24 06 2011.html>.

MUNONYE, K.; MARTINEK, P. Evaluation of data storage patterns in microservices archicture. In: **2020 IEEE 15th International Conference of System of Systems Engineering (SoSE)**. [S.l.: s.n.], 2020. p. 373-380.

NGINX. **What Is Load Balancing?** 2020. Disponível em: <https://www.nginx.com/resources/glossary/ load-balancing/>. Acesso em: 01 mar. de 2022.

NIELSEN, J. Enhancing the explanatory power of usability heuristics. In: **Proceedings of the SIGCHI conference on Human Factors in Computing Systems.** [S.l.: s.n.], 1994. p. 152-158.

NIELSEN, J. **Usability Engineering.** San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994. ISBN 9780080520292.

NOTTINGHAM, M.; WILDE, E. **Problem Details for HTTP APIs**. RFC Editor, 2016. RFC 7807. (Request for Comments, 7807). Disponível em: <https://rfc-editor.org/rfc/rfc7807.txt>.

OMIDH. **Spring Boot 2.X:** Developing web Applications Using Spring MVC + MyBatis + Thymeleaf. 2019. Disponível em: <https://programming.vip/docs/spring-boot-2x-developing-web-applications-using-spring-mvc-mybatis-thymeleaf.html>. Acesso em: 8 fev. 2022.

OMS. **Orientações para o tratamento de infecções sexualmente transmissíveis**. 2005. Disponível em: <http://whqlibdoc.who.int/publications/portuguese/9248546269por.pdf>. Acesso em: 04 set. 2020.

PATIL, M. M.; HANNI, A.; TEJESHWAR, C. H.; PATIL, P. A qualitative analysis of the performance of mongodb vs mysql database based on insertion and retriewal operations using a web/android application to explore load balancing - sharding in mongodb and its advantages. In: **2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC).** [S.l.: s.n.], 2017. p. 325-330.

PICORETI, R.; CARMO, A. Pereira do; QUEIROZ, F. Mendonça de; GARCIA, A. S.; VASSALLO, R. F.; SIMEONIDOU, D. Multilevel observability in cloud orchestration. In: **2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech).** [S.l.: s.n.], 2018. p. 776-784.

RAMÍREZ-NORIEGA, A.; MARTÍNEZ-RAMÍREZ, Y.; LIZÁRRAGA, J. C.; NIEBLA, K. V.; SOTO, J. A software tool to generate a model-view-controller architecture based on the entity-relationship model. In: **2020 8th International Conference in Software Engineering Research and Innovation (CONISOFT)**. [S.l.: s.n.], 2020. p. 57-63.

ROCHA, P. K.; PRADO, M. L. do; WAL, M. L.; CARRARO, T. E. Cuidado e tecnologia: aproximações através do Modelo de Cuidado. **Revista Brasileira de Enfermagem**, Associação Brasileira de Enfermagem, v. 61, n. 1, p. 113-116, 2008. ISSN 0034-7167. Disponível em: <http://www.scielo.br/j/reben/a/kmVnsg8zYHPf4CRgjgPx4bj/?lang=pt>.

SADQI, Y.; BELFAIK, Y.; SAFI, S. Web oauth-based sso systems security. In: **Proceedings of the 3rd International Conference on Networking, Information Systems & Security.** New York, NY, USA: Association for Computing Machinery, 2020. ISBN 9781450376341. Disponível em: <https://doi-org.ez121.periodicos.capes.gov.br/10.1145/3386723.3387888>.

SAKIMURA, N.; BRADLEY, J.; AGARWAL, N. **Proof Key for Code Exchange by OAuth Public Clients**. RFC Editor, 2015. RFC 7636. (Request for Comments, 7636). Disponível em: <https://rfc-editor.org/rfc/rfc7636.txt>.

SCHWABER; SUTHERLAND. **Guia do Scrum.** 2013. Disponível em: <https://scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf>. Acesso em: 16 jun. 2021.

SHARMA, A.; HARRINGTON, R. A.; MCCLELLAN, M. B.; TURAKHIA, M. P.; EAPEN, Z. J.; STEINHUBL, S.; MAULT, J. R.; MAJMUDAR, M. D.; ROESSIG, L.; CHANDROSS, K. J.; GREEN, E. M.; PATEL, B.; HAMER, A.; OLGIN, J.; RUMSFELD, J. S.; ROE, M. T.; PETERSON, E. D. Using digital health technology to better generate evidence and deliver evidence-based care. **Journal of the American College of Cardiology**, v. 71, n. 23, p. 2680-2690, 2018. ISSN 0735-1097. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0735109718344139>.

SHNEIDERMAN, B.; PLAISANT, C.; COHEN, M.; JACOBS, S.; ELMQVIST, N.; DIAKOPOULOS, N. **Designing the User Interface:** Strategies for Effective Human-Computer Interaction. 6th. ed. [S.l.]: Pearson, 2016. ISBN 013438038X.

SINGH, V.; PEDDOJU, S. K. Container-based microservice architecture for cloud applications. In: **2017 International Conference on Computing, Communication and Automation (ICCCA)**. [S.l.: s.n.], 2017. p. 847-852.

SMITH, W.; WADLEY, G.; DALY, O.; WEBB, M.; HUGHSON, J.; HAJEK, J.; PARKER, A.; WOODWARD-KRON, R.; STORY, D. Designing an app for pregnancy care for a culturally and linguistically diverse community. In: **Proceedings of the 29th Australian Conference on Computer-Human Interaction.** New York, NY, USA: Association for Computing Machinery, 2017. (OZCHI '17), p. 337-346. ISBN 9781450353793. Disponível em: <https://doi.org/10.1145/3152771.3152807>.

SOMMERVILLE. **Engenharia de software**. 8 ed. [S.l.]: Pearson Addison-Wesley, 2007.

SOMMERVILLE. **Engenharia de software.** 9 ed. [S.l.]: Pearson Addison-Wesley, 2011.

SOUSA, A. C. A. *et al*. Aplicação de tecnologia leve no pré-natal: um enfoque na percepção das gestantes [application of prenatal care light technology: focus on pregnant women's perception]. **Revista Enfermagem UERJ**, v. 21, n. 5, p. 648-653, 2014. ISSN 0104-3552. Disponível em: <https://www.e-publicacoes.uerj.br/index.php/enfermagemuerj/article/view/10043>.

SRIVASTAVA, A.; BHARDWAJ, S.; SARASWAT, S. **Scrum model for agile methodology**. p. 864-869, 2017.

SUWAWI, D. D. J.; DARWIYANTO, E.; ROCHMANI, M. Evaluation of academic website using iso/iec 9126. In: **2015 3rd International Conference on Information and Communication Technology (ICoICT)**. [S.l.: s.n.], 2015. p. 222-227.

TOMASI, E.; FERNANDES, P. A. A.; FISCHER, T.; SIQUEIRA, F. C. V.; SILVEIRA, D. S. da; THUME´, E.; DURO, S. M. S.; de Oliveira Saes, M.; NUNES, B. P.; FASSA, A. G.; FACCHINI, L. A. Qualidade da atenção pré-natal na rede básica de saúde do Brasil: Indicadores e desigualdades sociais. **Cadernos de Saúde Publica**, v. 33, n.

3, p. 1-11, 2017. ISSN 16784464.

TRIPP, N.; HAINEY, K.; LIU, A.; POULTON, A.; PEEK, M.; KIM, J.; NANAN, R. An emerging model of maternity care: Smartphone, midwife, doctor? **Women and Birth**, Elsevier, v. 27, n. 1, p. 64-67, mar 2014. ISSN 1871-5192.

VIELLAS, E. F.; DOMINGUES, R. M. S. M.; DIAS, M. A. B.; GAMA, S. G. N. da; THEME, M. M.; COSTA, J. V. da; BASTOS, M. H.; LEAL, M. d. C. Assistência pré-natal no Brasil. **Cadernos de Saúde Pública**, Escola Nacional de Saúde Pública Sergio Arouca, Fundação Oswaldo Cruz, v. 30, n. SUPPL1, p. S85-S100, 2014. ISSN 0102-311X. Disponível em: <http://www.scielo.br/j/csp/a/CGMbDPr4FL5qYQCpPKSVQpC/?lang=pt>.

WAN, F.; WU, X.; ZHANG, Q. Chain-oriented load balancing in microservice system. In: **2020 World Conference on Computing and Communication Technologies (WCCCT).** [S.l.: s.n.], 2020. p. 10-14.

WANG, J.; CHEN, W.; YANG, H. Architecture description language based on software reliability evaluation for distributed computing system. In: **2010 International Conference on Computer Application and System Modeling (ICCASM 2010).** [S.l.: s.n.], 2010. v. 7, p. V7–370-V7–374.

WASHIZAKI, H.; FUKAYA, K.; KUBO, A.; FUKAZAWA, Y. Detecting design patterns using source code of before applying design patterns. In: **2009 Eighth IEEE/ACIS International Conference on Computer and Information Science.** [S.l.: s.n.], 2009. p. 933-938.

WEISER, M. Some computer science issues in ubiquitous computing. **Commun. ACM,** Association for Computing Machinery, New York, NY, USA, v. 36, n. 7, p. 75-84, jul. 1993. ISSN 0001-0782. Disponível em: <https://doi-org.ez121.periodicos.capes.gov.br/10. 1145/159544.159617>.

WICAHYONO, G.; SETYANTO, A.; RAHARJO, S.; MUNANDAR, A. Pregnancy monitoring mobile application user experience assessment. In: **2019 International Conference on Information and Communications Technology (ICOIACT)**. [S.l.: s.n.], 2019. p. 872-877.

WIWEKO, B.; RIYANTI, A.; OLIVIA, S.; PRIANGGA, M.; SILVANA, V.; PERTIWI, I. P.; WIBAWA, Y. S.; FEBRI, R. R.; PUTRO, A. L. R.; AGUNG, P. G.; HESTIANTORO, A.; MUHARRAM, R.; HARZIF, A. K.; PRATAMA, G. Jakpros: Reproductive health education application for pregnant women. In: **2018 International Conference on Advanced Computer Science and Information Systems (ICACSIS)**. [S.l.: s.n.], 2018. p. 225-229.

WU, M.-Y.; LEE, T.-H. Design and implementation of cloud api access control based on oauth. In: **IEEE 2013 Tencon - Spring**. [S.l.: s.n.], 2013. p. 485-489.

YUSOP, N. S. M. Understanding usability defect reporting in software defect repositories. In: **Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference**. New York, NY, USA: Association for Computing

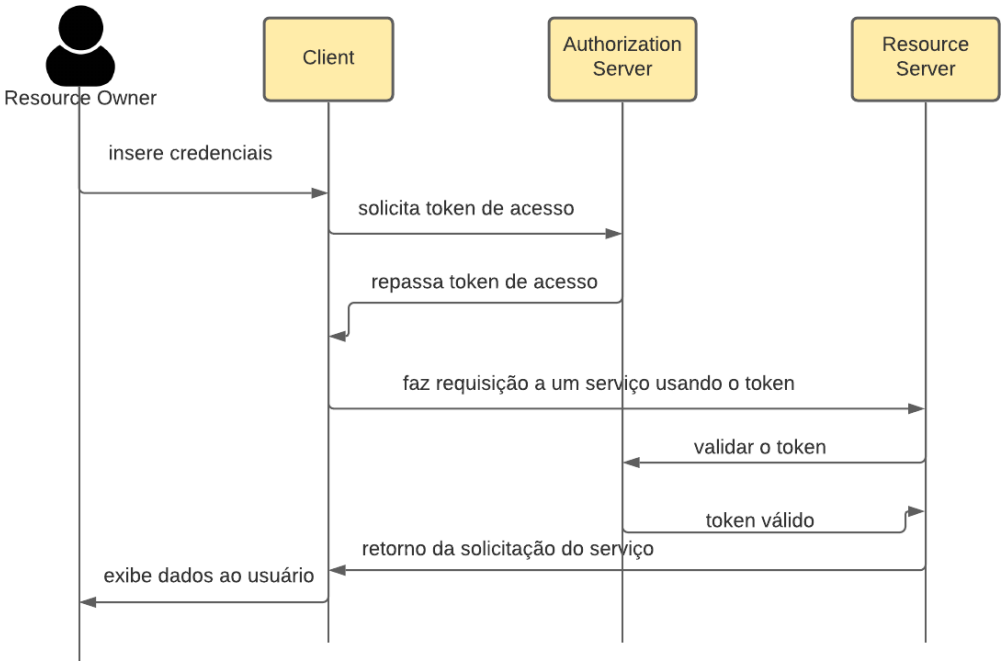
Machinery, 2015. (ASWEC' 15 Vol. II), p. 134-137. ISBN 9781450337960. Disponível em: <https://doi-org.ez121.periodicos.capes.gov.br/10.1145/2811681.2817757>.

Apêndice A – Artefatos

Neste capítulo são apresentados os artefatos de software gerados por este trabalho.

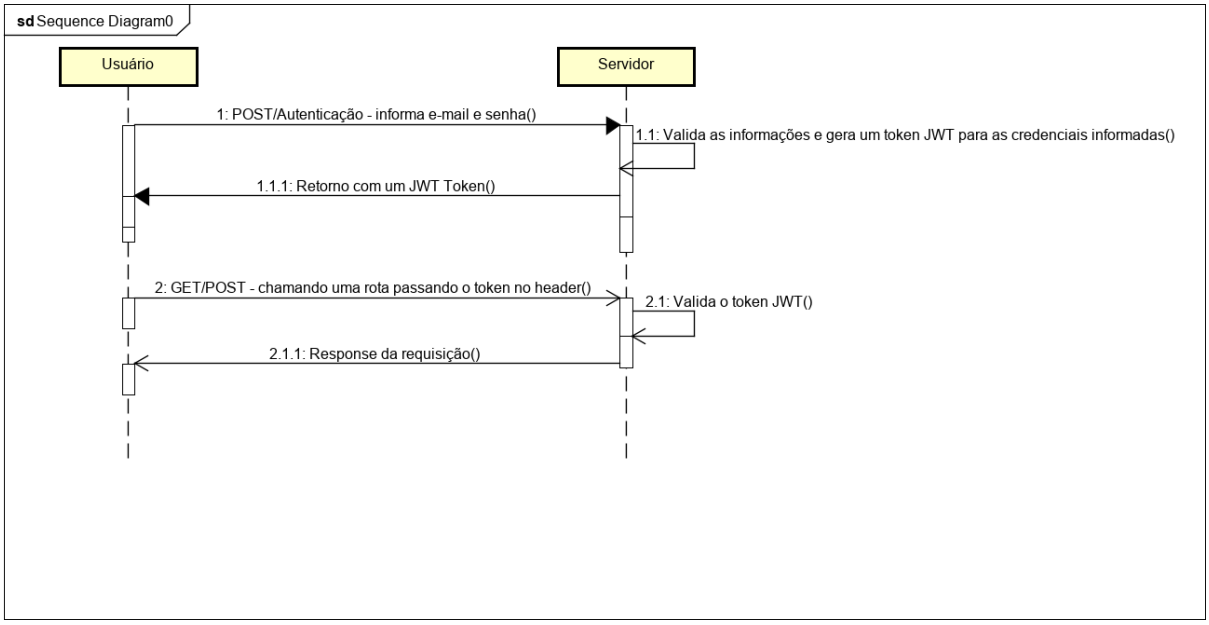
1 Diagramas de atividade para o fluxo de autorização e autenticação

Figura A.1 - Etapas de autenticação e autorização



A Figura A.1 representação Protocolo Auth 0.2 durante o fluxo de autenticação e autorização.

Figura A.2 - Fluxo de login



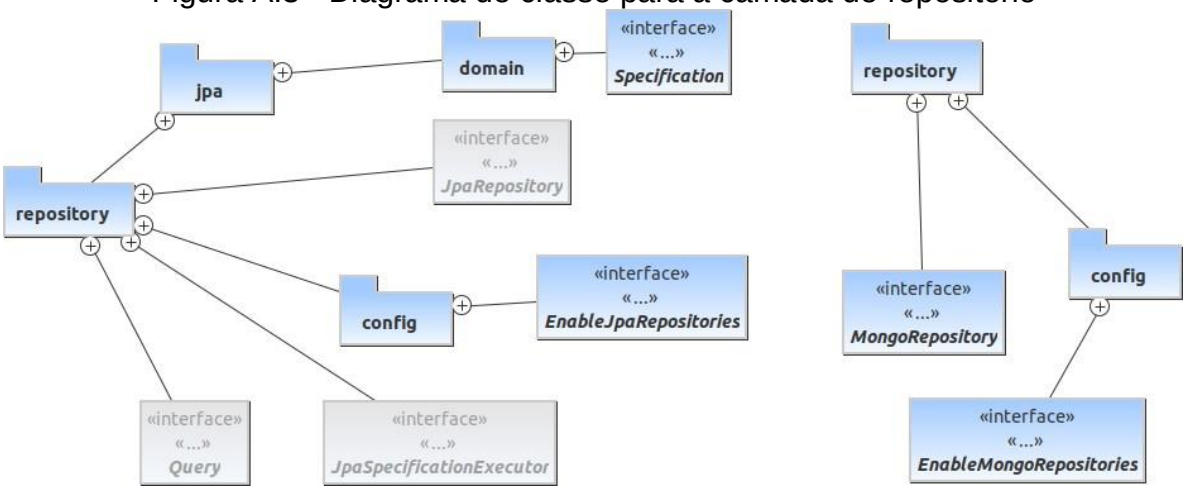
A Figura A.2 representa uma abstração do fluxo completo do OAuth 2.0, onde a API utilizada pelo um usuário envia uma requisição HTPP, com o verbo POST informando as credenciais e o servidor faz a validação e retorna um JWT token.

2 Diagrama de classes

A seguir serão apresentados alguns diagramas de classes de partes distintas do sistema.

2.1 Health-professionals-api

Figura A.3 - Diagrama de classe para a camada de repositório

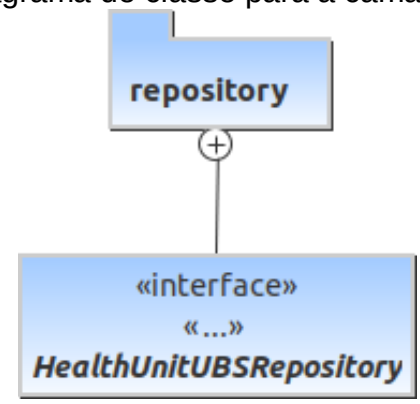


O Spring data JPA introduziu o conceito de *Specification* com o intuito de definir *predicates* reutilizáveis e reduzir a dificuldade de implementar consultas complexas, isso traz grande facilidade para realizar implementações com o *criteria-api*.

Para a API de profissionais de saúde existe um repositório dedicado a trabalhar com o banco mongoDB, um banco de dados não-relacional.

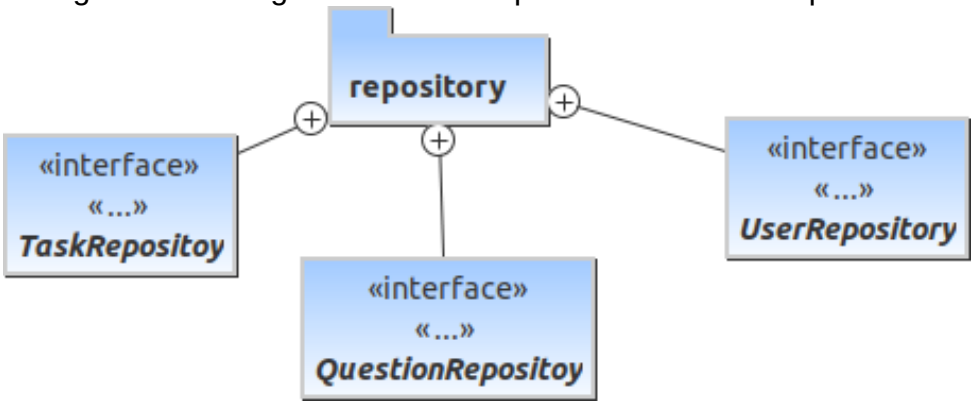
2.2 Health-units-api

Figura A.4 - Diagrama de classe para a camada de repositório



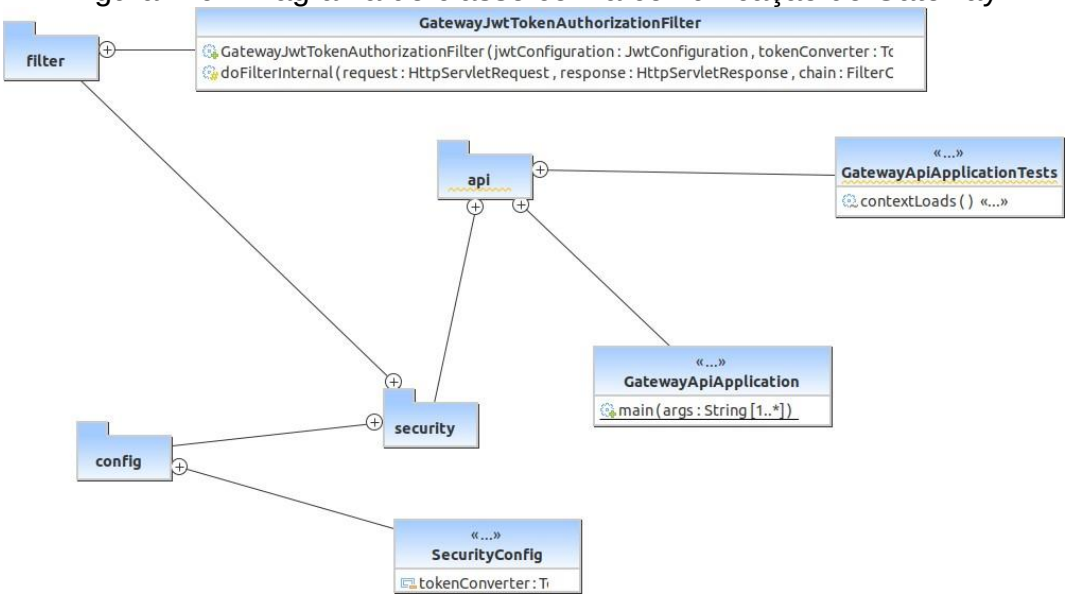
2.3 Pregnant-guide-api

Figura A.5 - Diagrama de classe para a camada de repositório



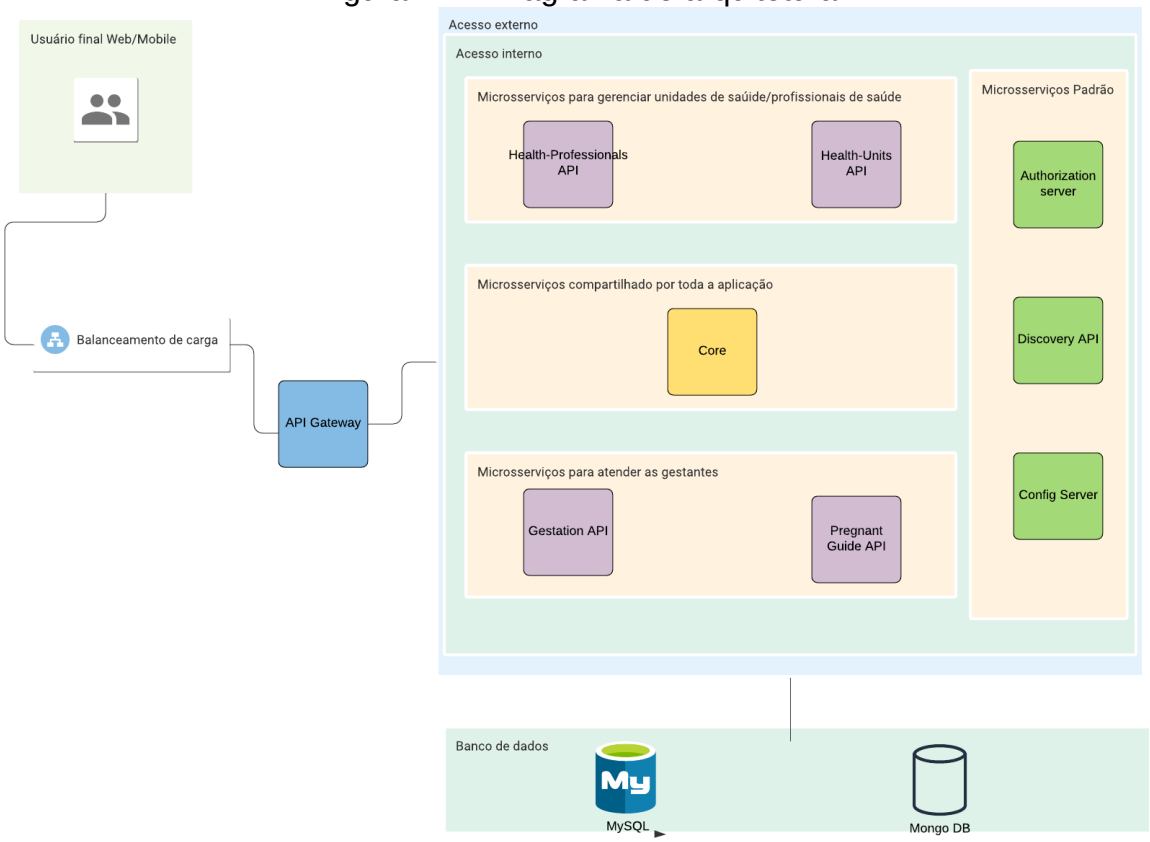
2.4 Gateway-api

Figura A.6 - Diagrama de classe com a comunicação do Gateway



3 Diagrama de arquitetura

Figura A.7 - Diagrama de arquitetura



A comunicação entre o *client* e o *gateway* é feita através do protocolo HTTP, o *gateway* se comunica com os demais serviços através de uma rede interna, utilizando o protocolo HTTP e *broker* de mensageria.

Figura A.8 - Camada de Persistência

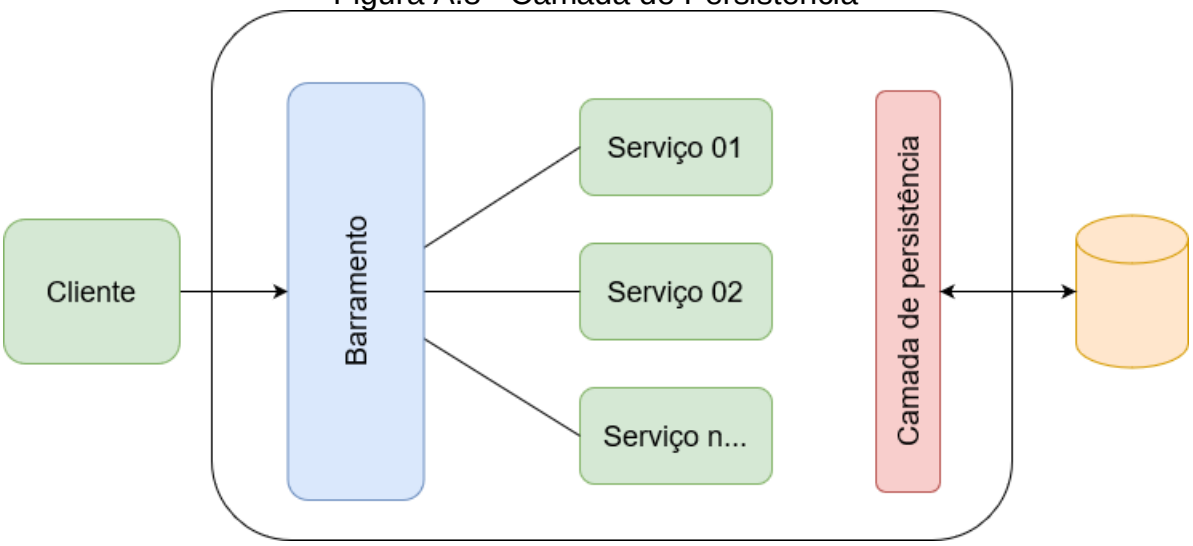
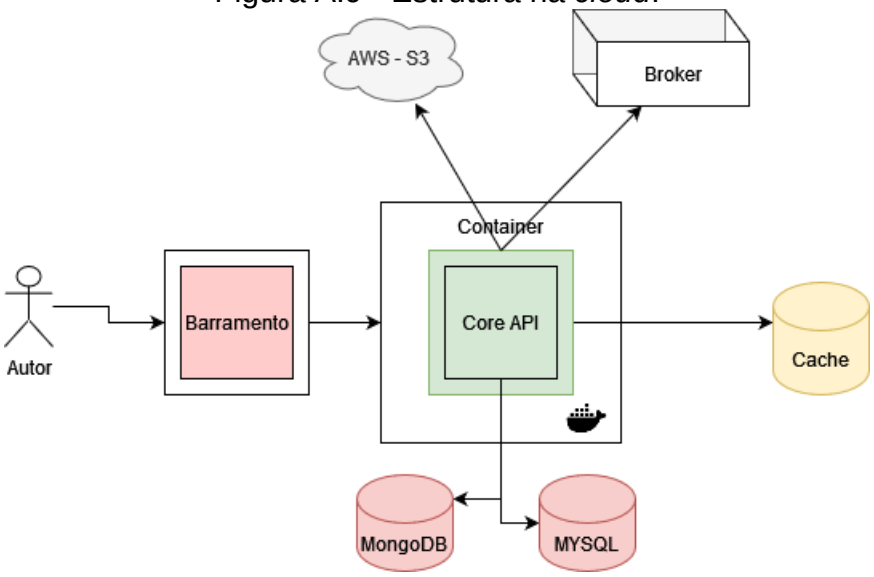


Figura A.9 - Estrutura na *cloud*.



4 Swagger

O Swagger é importante para facilitar a visualização dos serviços pertencentes a uma REST API. Em seguida será apresentando alguns serviços desenvolvidos neste trabalho.

4.1 Cadastro de Profissional de Saúde

Figura A.10 - Cadastro de um Profissional de Saúde

Description

healthProfessionalInput

Example Value | Model


```
{
  "jobRegistrationNumber": "string",
  "user": {
    "address": {
      "cep": "string",
      "city": {
        "id": 0
      },
      "complement": "string",
      "latitude": "string",
      "logradouro": "string",
      "longitude": "string",
      "neighborhood": "string",
      "number": "string"
    },
    "cpf": "string",
    "dateOfBirth": "string",
    "email": "string",
    "name": "string",
    "password": "string",
    "typeUser": 0
  }
}
```

Apêndice B – Artigos aceitos

2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)

Gestation: A Microservice architecture for a prenatal care application

Jose George Dias de Souza
Center for Strategy Technologies in Health (NUTES)
Paraiba State University
Campina Grande, Brasil
jose.george@aluno.uepb.edu.br

Daniel Scherer
Computer Department/Center for Strategic Technologies in Health (NUTES)
Paraiba State University
Campina Grande, Brasil
https://orcid.org/0000-0001-9800-4453

2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC) | 978-1-6654-2463-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/COMPSAC51774.2021.00099

***Abstract*— Pregnancy is an important phase in women's lives, prenatal care is important to prevent diseases and ensure that women and babies can have a better quality of life. However, in Brazil, many pregnant women still do not perform due prenatal care, there are several factors that contribute to the failure to perform prenatal care among them: geographical factors. Pregnant women who live far from health units are unable to start pregnancy care in time. This work proposes a viable and low-cost alternative for the government or private sector companies. A Microservices architecture is presented, capable of meeting requests from pregnant women via mobile application and allowing health professionals to monitor the pregnancy evolution of a health unit. The architecture was designed to offer satisfactory performance, scalability and simplicity.**

Keywords*— *Pregnancy, prenatal, architecture, performance, scalability.

I. Introduction

An important phase in the life of any woman is the gestational period. The Pregnancy is, however, a phase of discovery and is often accompanied by fears and fears, especially in pregnant women who are going through this period for the first time. According to [1] the woman faces several physiological as well as psychological changes [2].

Prenatal care is necessary to prevent diseases resulting from pregnancy, in addition to reducing risks for women and enabling healthy development for the baby. Currently, there are numerous researches that use technologies applied to health, such as patient monitoring. In the most specific area of obstetric and gynecological medicine, researchers focus on developing mobile and web applications, as [3] and [4] developed to attend, monitor and guide pregnant women during the prenatal period.

For the Ministry of Health of Brazil [5], the main purpose of prenatal care is to provide a healthy evolution of pregnancy without critical impacts. However, to achieve the expected results, the Ministry of Health of Brazil develops a series of educational activities in basic health units (UBS) and seeks to promote a welcoming and humanized care for pregnant women. However, these measures are not fully effective, there are many women who live geographically distant from the health units and end up failing to start prenatal care in a timely manner.

With the emergence of smartphones and constant advances in the field of information technology, the emergence of ubiquitous health systems is increasingly constant. According to [6] during the gestational phase, women often search the internet for information about the gestational process, many believe in the veracity of the data found, this can be dangerous because a treatment, medication or procedure is not applicable in all cases, and there are a lot of false information on the internet.

The purpose of this research is to present a modern software architecture, effective and capable of guaranteeing the data exchange between pregnant women and health professionals in a synchronous and asynchronous way, without data loss, consistent and fast. With this aim to ensure a more humanized and welcoming prenatal care, bringing benefits to the health of women and babies and reducing risks. For this, an architecture based on the microservices model with well-defined division of layers will be used, with the exchange of information between internal services performed in two ways: via Rest API and Message Queue. In addition, a relational bank will be used at times and at other times a non-relational bank for persistence and data recovery.

This research is divided as follows, in section 2 it will be discussed through a literature review some models of software architectures applied to the health context; then in section 3 an approach to the methodology will be made, with this, some important points will be presented to understand the development of the study; in section 4 we will discuss how it was designed and why we decided to opt for a microservice architecture; finally, in section 5 the final considerations will be presented.

II. Literature Review

In this topic, research that involves architectural software solutions to meet specific demands in the health area will be addressed.

A. Software architecture of mobile applications for health

In the work [7] the researchers presented a robust software architecture, distributed and applied to analyze data for health. The architecture design was composed of four design patterns: Model-view-viewModel pattern (MVVM), The Inversion of Control (IOC) pattern, Proxy pattern and repository pattern. During the research, the authors argued that using a non-relational database brings countless benefits when compared to relational banks, this for the dynamics of data analysis, due to the fact that non-relational banks allow the use of massive processing with MapReduce, ideal for analyzing large volumes of data. Also, according to the researchers, at present, most of the patient data is managed by traditional databases, that is, relational databases.

Currently, some research has been carried out in the field of Ambient Assisted Living (AAL), this is mainly due to the fact that the population is aging [9]. In the research proposed by [8], a reference architecture for home health support systems was addressed, the researchers highlight that a

978-1-6654-2463-9/21/$31.00 ©2021 IEEE
DOI 10.1109/COMPSAC51774.2021.00099

reference architecture covers knowledge of how to approach the business rules of the application up to architectural quality definitions. used in the research the domain-oriented quality models development (DUMOND) quality model, this model seeks to define quality models for specific domains based on standards and the software engineering literature. Many works currently developed present a proposal of difficult abstraction in the context of software architecture, that is, there are not enough guidelines to provide guidance on: scalability, development and evolution of the tool.

Software with monolithic architecture no longer meets current scalability needs, as well as the rapid development cycle [16], in addition monolithic architectures cost more because they need more computing resources to stay on cloud when compared to microservices [17]. Based on this [10] software with a service oriented architecture (SOA) proposed, the authors argue that an SOA architecture offers a low coupling and a distributed infrastructure, thus guaranteeing more sustainability to the application. The researchers concluded that the SOA architecture model is suitable for complex systems like health and can be highly flexible and interoperable.

Modeling an application's architecture is not an easy task, each software has a well-defined objective and the decision for which architecture model should be unique based on the application's proposal. Between SOA and Microservices this research opted for a Microservices architecture, as it presents a better service granularity, that means that each service is unique, solid, easy to understand and solves a specific problem very well. While in SOA architecture, services have different sizes and sometimes more than one specialty.

B. *Scalability and performance*

Using a Microservices architecture, it is possible to bring numerous gains to an application, especially with regard to performance, maintenance, scalability and interoperability [11]. The great advantage of this model is the possibility of an application having one or more databases, even one database per service and each service can easily be deployed on a different server. This brings countless benefits, considering that the engineers responsible for the application, can choose to use a server with more computational resources only to serve those services that receive more requests, while the other services can be deployed on simpler servers with less processing capacity, this is not possible in a monolithic architecture.

In software engineering design, it is necessary to consider several factors for creating and deploying software. Wrong decisions can generate a high cost at the end of the project. Software project managers always require the most reliable cost estimation model [12]. The planning and development of software based on Microservices architecture is not an easy task, if there is no planning, the project will not have real benefits. The choice of a microservice solution was due precisely to the fact that it connects pregnant women to health facilities, with speed, safety and quality. Bearing in mind that in some moments it will be necessary to use part of the application based on a non-relational database model and in another moment a relational base will be more effective, besides thinking about the scalability of the application, there was one more reason for this work to follow a Microservice architecture instead of a SOA or Monolithic architecture.

III. Methods

This research is an exploratory study, so it was necessary to approach the theme to be resolved: The decrease in the number of pregnant women who do not perform prenatal care. This approach took place through an extensive bibliographic analysis, where data were collected from other research and documents from the Ministry of Health of Brazil. Furthermore, it was important to study scalable technologies, frameworks, programming paradigms, in order to propose an effective architectural solution that could meet the proposed solution.

A. *Agile methodology*

In this work the Scrum agile work methodology is being used, the choice was made for the following reasons: Scrum is easy to use, flexible and modern. Scrum is easily adaptable for distributed environments [13]. Each week a meeting is held with the team involved, where the improvements made, the impasses and the future steps are discussed. Each month a document is presented with the details of the finalization of the release, that is, each sprint has a one-month plan. Scrum's values: courage, focus, commitment, respect and openness are important for the team involved with the research and development of the Gestation platform. All code versioning is managed by Gitlab, each sprint is then created a new branch, at the end of the month, a pull request is opened for approval and later approval is made the merge to the master branch, so the project always has a functional code, ready to go into production.

The prioritization of the Product Backlog is carried out after the completion of each sprint, the team gets together, raises positive and negative points, verifies the biggest challenges from the point of view of implementation, finds out if any activities need to be re-planned and prioritizes tasks for the next sprint. Up to January 2021, four sprints were performed.

B. *Functional requirements*

Functional requirements are directly linked to resources that need to be developed to meet the needs of users. To develop all of these requirements, an extensive review of the bibliography was necessary in order to raise the main challenges of prenatal care. Requirements are important elements in software design, their understanding for the project is predominant for success or failure [14]. Below are the main functional requirements that must be included in the pregnancy platform that this research is about.

- Registration of personal data of the pregnant woman with the address information.
- Registration of health professionals.
- Functionality to allow a health professional to send weekly tasks to the pregnant woman.
- Reminder of the consultation schedule for the pregnant woman.
- The pregnant woman's application should contain a screen with frequently asked questions.
- The pregnant woman must send questions to the doctors through her personal cell phone.
- The application should recommend educational content based on the profile of each pregnant woman thus minimizing the fears and fears of pregnancy.

C. Technologies

The basis of the architecture was developed using the Spring ecosystem of the Java language. More specifically, the Spring Cloud was used, as it allows and brings more facilities to build microservice applications. In addition, the Netflix OSS frameworks were used to register microservices and facilitate communication between them. The Java version was 11, because it is stable and well tested and the server used was the tomcat built by Spring Boot itself.

To facilitate server deployments, this research uses Docker, which plays an important role in creating application containers. The messaging service makes use of RabbitMQ, as this messaging service has good communication and flexibility with the Spring universe and is easy to use. The application currently makes use of a relational base: MySQL and another non-relational base: MongoDB. As the application grows, new structures are added. In the future, Angular will be used together with Flutter to create web and mobile interfaces.

IV. Proposed Architecture

The application proposed by this work has some different usage scenarios. That is why it is so important to model an architecture to serve users well, ensuring safety and reliability. The architecture proposed by this research corroborates to allow pregnant women to connect through an application on their smartphones, often with limited connections to the internet and limited devices from the hardware point of view. Therefore, it is necessary to ensure that information between health professionals and pregnant women is not lost on the network. The architecture still needs to return requests made via mobile application quickly even in situations like this. Another usage scenario focuses on health professionals, who assist pregnant women in basic health units. These professionals will use the platform through a web system, from within the health unit, to prescribe activities related to pregnancy, such as accompanying women during prenatal care.

Currently, the gestation platform has four responsible APIs to meet requests from pregnant women, health units and medical professionals. There is also a responsible for the API to manage doubts and enable interaction between pregnant women and health professionals. This API also will play an important role in the future, be responsible for recommending educational content for pregnant women.

In addition to these four, there are 3 more APIs on the platform, the gateway that is responsible for orchestrating calls to the platform services, the discovery responsible for managing the application instances and the API responsible for user authentication and control.

A. Comparison between monolithic architecture and microservices

Monolithic architecture has advantages and disadvantages in relation to a microservice architecture, the decision to choose is directly related to the business. When the intention is to build an application that will serve a smaller number of customers, where the long-term business will not need scalability, for sure, the best choice is for the simple one. Using a microservice architecture is too complex for a small application. In a monolithic application all layers of the system, services, security, communication with the database are contained in a single “file”, that is, the entire application is deployed on a single server.

As an application grows, scaling up a monolithic architecture becomes a rather complex task. Not to mention that spending increases exponentially. Therefore, it is necessary to allocate more and more computational resources to meet the application. Therefore, it is necessary to have a powerful server to attend service requests that are less used by customers as well as for more used services, the reason for this is the fact that in a monolithic application, the entire application is implanted in a single “server”. Another worrying factor in monolithic systems is the complexity and cost of maintenance, given that system analysts need to analyze a huge set of code to make a correction. Often there is still a high dependency within monolithic code, the same code snippet is "reused" by other parts of the system, so a correction on one side can cause a problem on the other.

In Brazil there are millions of women who get pregnant annually, in 2019 alone there were almost 3 million births [15], to meet all these pregnant women, thousands of health professionals are needed. Opening communication between pregnant women and health professionals is quite a challenge. In this sense, it is important to build a scalable, flexible, easy to use and low-cost maintenance application. Therefore, the option for a microservice architecture was made, considering that in this architectural model, each service is built independently, thus reducing the coupling, thus enabling cheaper maintenance, considering that it will no longer be necessary a long code analysis for a fix, the analyst just needs to check that specific service that needs to change and make the changes.

Unlike monolithic architectures, an architectural microservice model allows the generation of one or many "files" to be deployed on one or many different servers. In other words, the user registration service can be on a server, using a database, while the service to consult the date of the next prenatal consultation can be installed on another server, they are totally independent. This means that if one service becomes unavailable, the other will continue to function. Finally, microservices offer the possibility of using different technologies. In this research all the back-end of all services are being developed in the Java programming language, version 11, using the Spring Cloud framework. The application is served by two databases, one relational, using MySql and the other non-relational, using MongoDB, but this does not prevent that in the future new services can be developed using other languages, with other databases.

B. Application security

In Brazil, the General Law on the Protection of Personal Data (LGPD), law number 13,709, of August 2018, came into force in 2020. It deals with how the handling of personal data should be, especially in digital media. Brazilian law was based on the GDPR (General Data Protection Regulation) which deals with data security in the European Union. Therefore, patients need to have a complete understanding of what is done with their personal data and also agree with any type of use that may be made with their sensitive data. As this work presents a solution for health, it will work with sensitive data from patients and health professionals, so it was necessary to have a security readjustment to be in compliance with the LGPD. No personal data of a pregnant woman or health

professional is shared with any other external service, nor is this data used without the user's authorization.

A very common and widely used practice is to use cache for some specific API endpoints. From a performance point of view this is very good, considering that when a given resource is cached, the browser does not need to make a new call to the server to retrieve that resource again, only the cache is checked and the information is instantly displayed for the user. Some systems use the private cache, which is when that data is only available on the user's personal device, but there is the possibility of placing information in a public cache, that is, the same information is shared by users who use a particular application or service, this information is available on a shared intermediate server, so it is faster for the browser to retrieve this from a shared intermediate server than to retrieve it from the final server.

The use of cache is useful, very useful when you do not share sensitive user data, however, you must be very careful when using this type of technique in a health application. The disclosure of a patient's personal data in the health context can never be allowed. And it is important to note that a lot of cached data and cookies are used by browsers to get to know a user's profile and thereby personalize content and ad recommendations. Based on this, in this research there are only two services with cached information, the city service and the state service. When using the application at times, calls will be made to services that respond with a list of cities belonging to a given state, at no time will a user ID be passed to consult this cached data which makes the application secure.

The platform that is under development, in which this research addresses, makes a call to an external application, to consult health units closest to the pregnant woman who has just registered in the application. However, only user address data is passed in this call, such as: street, neighborhood and state as shown in figure 1.


```
{
   "nameApi":"gestation-records-api",
   "codeRequest":"6893991b-08b5-45c4-9bf9-c37283d80350",
   "address":{
      "cep":"58900-142",
      "number":"80",
      "complement":"Em frente a fafic",
      "neighborhood":"Centro",
      "latitude":"-12341839123",
      "longitude":"-123413412",
      "city":"Amparo",
      "state":"Paraiba"
   }
}
```


Fig 1- example of an external service call.

The application's registration flow, which can be viewed in figure 1, generates a generic ID and links it to a user to which a given address belongs. Then a call is made to an external service, which returns in the reply a list of units close to the location of the pregnant woman and exactly in the reply the same ID generated in the sending is sent. In other words, at no time did the exchange of sensitive information occur, where some user sensitive data was exposed, putting their privacy at risk.

In the application architecture proposed by this research, all external requests, which are made by a mobile or web device, are processed by an api-gateway, it is precisely this api that orchestrates all service requests. It is not possible for an external call to be able to call an internal api directly. In this sense, taking as an example the API responsible for registering a new pregnant woman is gestation-records-api, it is not possible to be called via rest directly without first going through the gateway-api. Within this architectural model it is only possible to make an internal call, internally in the API, through another internal api. As shown in figure 2 below, where in the questioning flow, a validation is performed to check if a particular professional has the privilege to answer such questioning opened by a pregnant woman. This is a function provided by the application that will be used by the pregnant woman, to ask questions to health professionals.

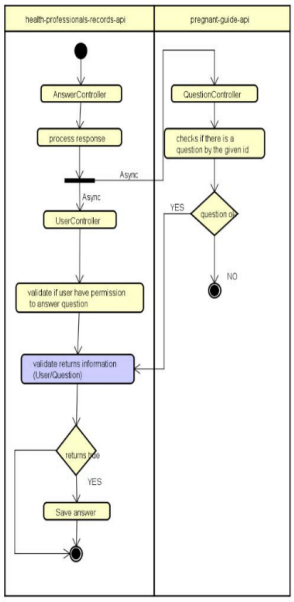


Fig 2 - response flow to questioning of the pregnant woman.

All requests that are made by users enter this architectural model, through an api-gateway as already mentioned, the gateway also orchestrates the call to the most different services, as well as being responsible for making an internal call in the authentication API to validate the credentials. of users with profile validations and access control.

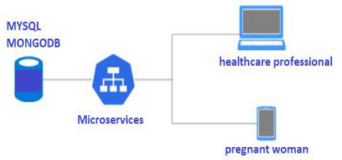


Fig 3 - architecture example.

Both requests for the pregnant woman's mobile application and requests for web systems that health professionals will use will be met by the microservice architecture proposed by this research. The architecture will connect to the relational and non-relational basis and will return the information to users.

V. CONSIDERATIONS

The development of any application for health is naturally not simple, it requires great dedication and effort to choose the best: models, standards and frameworks. Monitoring the pregnancy is relevant to ensure a safe delivery, however in Brazil there are differences and barriers that need to be overcome.

This work exposes the development of a microservice architecture, capable of opening a connection between women, especially those who live in remote regions, who do not have easy access to a health unit and health professionals, wishing to increase the interaction between women and ensure a more humane and welcoming delivery. This technology can be very useful, a good part of the population makes use of smartphones and is connected to the internet.

In addition to a broad, scalable and flexible architecture, this research needs to continue, as it will be necessary to develop a web application for health professionals and another mobile to assist pregnant women. The purpose of the mobile application is to allow the pregnant woman to communicate with the nearest health unit, ask questions, answer questions, receive consultation alerts and comply with activities prescribed by health professionals. During the prenatal period, already the application web will be used by health professionals for the proper monitoring of pregnant women.

ACKNOWLEDGMENT

The authors thank the State University of Paraiba and NUTES (Center for Strategic Technologies in Health) for the opportunity to develop the works.

REFERENCES

[1] Kalogeropoulos, D., Sung, V. C. T., Paschopoulos, M., Moschos, M. M., Panidis, P., & Kalogeropoulos, C. (2019). The physiologic and pathologic effects of pregnancy on the human visual system. In Journal of Obstetrics and Gynaecology (Vol. 39, Issue 8, pp. 1037–1048). Taylor and Francis Ltd. https://doi.org/10.1080/01443615.2019.1584 891.

[2] Bjelica, A., Cetkovic, N., Trninic-Pjevic, A., & Mladenovic-Segedi, L. (2018). The phenomenon of pregnancy - A psychological view. In Ginekologia Polska (Vol. 89, Issue 2, pp. 102–106). Via Medica. https://doi.org/10.5603/GP.a2018.0017.

[3] Wicahyono, G., Setyanto, A., Raharjo, S; Munandar, A. (2019). Pregnancy monitoring mobile application user experience assessment. 2019 International Conference on Information and Communications Technology, ICOIACT 2019, 872–877.

[4] Munandar, A., Setyanto, A., Raharjo, S., & Wicahyono, G. (2019). Pregnancy mapping and monitoring web based geographic's information system. 2019 4th International Conference on Information Technology, Information Systems and Electrical Engineering, ICITISEE 2019, 490–494.

[5] Ministério da Saúde. Secretaria de Atenção à Saúde. Departamento de Atenção Básica. Política Nacional de Atenção Básica. Ministério da Saúde. Secretaria de Atenção à Saúde. Departamento de Atenção Básica. Brasília: Ministério da Saúde, 2012.

[6] M Goetz, M Müller, L.M Matthies, J Hansen, a Doster, a Szabo, J Pauluschke-fröhlich, H Abele, C Sohn, M Wallwiener, S Wallwiener.Perceptions of Patient Engagement Applications During Pregnancy: A Qualitative Assessment of the Patient's Perspective. JMIR Mhealth Uhealth. Mai de 2017.

[7] H. Salavati, T. J. Gandomani and R. Sadeghi, "A robust software architecture based on distributed systems in big data healthcare," 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, 2017, pp. 1701-1705, doi: 10.1109/ICACCI.2017.8126088.

[8] L. M. Garces Rodriguez, A. Ampatzoglou, P. Avgeriou and E. Y. Nakagawa, "A Reference Architecture for Healthcare Supportive Home Systems," 2015 IEEE 28th International Symposium on Computer-Based Medical Systems, Sao Carlos, 2015, pp. 358-359, doi: 10.1109/CBMS.2015.39.

[9] E. Y. Nakagawa, P.O. Antonino, M. Becker, J. C. Maldonado, H. Storf, K. B. Villela, et al., "Relevance and perspectives of AAL in Brazil", Journal of Systems and Software New York United States: Elsevier Inc., pp. 985-996, 2013.

[10] S. C. Chu, "From component-based to service oriented software architecture for healthcare," Proceedings of 7th International Workshop on Enterprise networking and Computing in Healthcare Industry, 2005. HEALTHCOM 2005., Busan, South Korea, 2005, pp. 96-100, doi: 10.1109/HEALTH.2005.1500402.

[11] J. A. Valdivia, X. Limón and K. Cortes-Verdin, "Quality attributes in patterns related to microservice architecture: a Systematic Literature Review," 2019 7th International Conference in Software Engineering Research and Innovation (CONISOFT), Mexico City, Mexico, 2019, pp. 181-190, doi: 10.1109/CONISOFT.2019.00034.

[12] J. Keung, "Software Development Cost Estimation Using Analogy: A Review," 2009 Australian Software Engineering Conference, Gold Coast, QLD, 2009, pp. 327-336, doi: 10.1109/ASWEC.2009.32.

[13] M. Paasivaara, S. Durasiewicz and C. Lassenius, "Using scrum in a globally distributed project: a case study", Software Process: Improvement and Practice, vol. 13, pp. 527-544, 2008.

[14] A. Aurum and C. Wohlin, "Requirements Engineering: Setting the Context" in Engineering and Managing Software Requirements, Berlin, Heidelberg:Springer, pp. 1-15, 2005.

[15] Ministério da Saúde. Datasus: numbers born in 2019. Homepage. Available at: <http://tabnet.datasus.gov.br/cgi/tabcgi.exe?sinasc/cnv/pnvuf.def>. Accessed on: December 8th. 2020.

[16] S. Newman, Building microservices: design fine-grained systems, O'Reilly Media, Inc., 2015.

[17] M. Villamizar et al., "Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures," 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Cartagena, Colombia, 2016, pp. 179-182, doi: 10.1109/CCGrid.2016.37.