

UNIVERSIDADE VIRTUAL DO ESTADO DE SÃO PAULO
ENGENHARIA DA COMPUTAÇÃO

DANIEL RONCAGLIA CORREIA DOS SANTOS

Desenvolvimento de aplicativo mobile para prestadores de serviços locais,
utilizando metodologia ágil. Exemplo: mecânica de bicicleta

SÃO PAULO – SP
2019

DANIEL RONCAGLIA CORREIA DOS SANTOS

Desenvolvimento de aplicativo mobile para prestadores de serviços locais,
utilizando metodologia ágil. Exemplo: mecânica de bicicleta

Trabalho de Conclusão de Curso
apresentado ao curso de Engenharia da
Computação da Universidade Virtual do
Estado de São Paulo como requisito
parcial para obtenção de título de
bacharel em Engenharia da Computação

SÃO PAULO – SP

2019

DANIEL RONCAGLIA CORREIA DOS SANTOS

Desenvolvimento de aplicativo mobile para prestadores de serviços locais,
utilizando metodologia ágil. Exemplo: mecânica de bicicleta

Trabalho de Conclusão de Curso
apresentado ao curso de Engenharia da
Computação da Universidade Virtual do
Estado de São Paulo como requisito
parcial para obtenção de título de
bacharel em Engenharia da Computação

Aprovado em: _____

Conceito: _____

BANCA EXAMINADORA

Prof.

Presidente

Prof.

Membro

Prof.

Membro

RESUMO

O projeto descreve o desenvolvimento ágil de um aplicativo mobile destinando a ser modelo aos prestadores de serviços locais. O exemplo hipotético escolhido para o aplicativo chamado Fixbi é o de uma mecânica de bicicleta. Pelo celular, após logar sistema, os ciclistas podem chamar atendimentos de um mecânico e consultar seus atendimentos. O documento apresenta um histórico da computação com foco no mobile explicando as principais ferramentas. Para o sistema, é montado um banco de dados com as tabelas tipo usuário, usuários, ciclistas, mecânicos e atendimentos. Para a aplicação back-end foi codificado uma aplicação em linguagem C#. Os testes de funcionamento da aplicação fazem parte do projeto. As páginas do aplicativo tiveram seu desenho feitos através de vetores. A implantação do front-end mobile é realizada com as ferramentas mais recentes e testada por emulador virtual. A conclusão traça projeções para a continuidade.

Palavras-chaves: Desenvolvimento de sistemas; Aplicativos; Mobile; Engenharia da Computação, Metodologia ágil.

ABSTRACT

This project describes the agile development of a mobile application to serve as a model for local service providers. The hypothetical example chosen for the application a bicycle mechanic called Fixbi. By cell phone, after logging in, cyclists can call a mechanic and consult their calls. This document presents a history of computing focused on mobile explaining the main tools for that. A database is set up with the tables: user type, user, cyclist, mechanic and precedent tables. For the back-end application was encoded a C # language application. The application run tests are part of the project. The pages of the application had their drawing done through vectors. The deployment of the mobile front-end is accomplished with the latest tools and tested by virtual emulator. The conclusion draws projections for continuity.

Keywords: Systems development; Applications; Mobile; Computer Engineering, Agile Methodology.

SUMÁRIO

1 INTRODUÇÃO.....	5
1.1 Definição.....	5
1.2 Motivação.....	7
1.3 Justificativa.....	9
 2 REVISÃO METODOLÓGICA.....	 12
2.1 Lógica de programação.....	12
2.2 Metodologia ágil (Scrum).....	12
2.3 Controle de versões (Git).....	15
2.4 Processos de criação.....	16
 3 OBJETIVOS.....	 18
3.1 Objetivo geral.....	18
3.2 Objetivos específicos.....	18
 4 APLICATIVO FIXBI.....	 19
4.1 Full Stack.....	19
4.2 Banco de dados (SQL).....	19
4.3 Back-end (API – C#).....	27
4.4 Teste (Postman).....	42
4.5 Design (UI/UX).....	46
4.6 Mobile (React Native).....	49
 5 CONSIDERAÇÕES FINAIS.....	 67
5.1 Análise de resultados.....	67
5.2 Projeções para o projeto.....	68
 6 REFERÊNCIAS.....	 69

1 INTRODUÇÃO

O mercado de aplicativos para plataformas mobile tem crescido de forma exponencial desde o início dos 2000 por qualquer indicador disponível publicamente e deve manter esta tendência pelos próximos anos. Hoje sendo o principal dispositivo de mídia digital em número de pessoas que os utilizam, os smartphones (telefones inteligentes) estão criando uma alta demanda por novos sistemas.

Este projeto apresenta o desenvolvimento, utilizando metodologia ágil, de um aplicativo mobile que pode ser usado como modelo para prestadores de serviços. O estudo de caso é o de uma mecânica de bicicleta, chamada de FixBi (Figura 1), no qual os ciclistas poderão solicitar, através do celular, o serviço de um profissional especializado no conserto e manutenção de bicicletas.

Este documento é um relatório que apresenta as ferramentas e o desenvolvimento passo a passo do banco de dados do serviço, do back-end da aplicação e do aplicativo mobile.

Figura 1 – Logotipo do aplicativo Fixbi



Fonte: autor

1.1 Definição

O termo “app” foi escolhido em 2010 como a “palavra do ano” pela American Dialect Society, concurso que destaca as expressões que ganham presença nas comunicações nos Estados Unidos. O “app” é uma forma abreviada para se referir aos aplicativos, que são programas de computador que funcionam em

um sistema operacional, em especial nas plataformas mobiles. Naquele ano, a palavra esteve presente em campanha publicitária da Apple sobre o iPhone (AMERICAN DIALECT SOCIETY, 2011).

A definição de programa de computador é de que se trata de um conjunto de instruções em formato de algoritmos que descrevem funções a serem realizadas por um computador. É considerado o primeiro programa de computador um trabalho publicado em 1842 pela matemática inglesa Ada Lovelace, no qual sugere forma para calcular os números do princípio de Bernoulli através da máquina analítica de Charles Babbage (ADA, 1842).

Na década de 1980, já com desenvolvimento de um mercado de computação eletrônica, o cientista Mark Weiser cunha o conceito “computação ubíqua”, apresentando como tese a ideia de que as mais profundas tecnologias são aquelas que “desaparecem”. Comparando ao caso da escrita, Weiser entende que essas tecnologias se tornam indistinguíveis da nossa vida comum. Para o cientista, a computação caminha para sentido ao que chama de ubíqua (WEISER, 1991).

Em pouco menos de duas décadas, qualquer observador pode comprovar em lugares públicos que o mobile, através do celular, se tornou tão popular que “desapareceu” como tecnologia se tornando plataforma preferencial da maioria da população para comunicação cotidiana.

A literatura sobre o mobile é escassa em português, sendo a principal produção em inglês na academia, no mundo editorial e na comunidade de especialistas em tecnologia da informação. Em 2009, o desenvolvedor Brian Fling publica manual ensinando como desenvolver um aplicativo mobile. Em seu histórico, faz paralelo entre computador eletrônico, telefone fixo e celular, que surge por volta dos anos de 1970. O autor identifica o início da era dos smartphones por volta do ano 2002 quando empresas como a Nokia e Blackberry lançaram aparelhos com teclado e tela, dispondo de funções além da ligação telefônica. “O moderno telefone mobile é capaz de fazer quase tudo que o computador desktop pode fazer, mas com um potencial para maior relevância nas nossas atividades diárias”, analisa o especialista (FLING, 2009).

Um dos momentos pilares desta tecnologia é o lançamento do iPhone. Na apresentação do novo modelo em conferência em 9 de janeiro de 2007, o criador da Apple, Steve Jobs, mostra ao mundo o alcance do produto que revelava, que inclui

entre as suas novidades uma tela de controle multitoque (touch screen). “A Apple reinventa o telefone”, afirma o executivo da empresa (JOBS, 2007).

No mesmo ano do lançamento do iPhone, com o seu sistema iOS, a Google disponibiliza o sistema operacional Android, que funciona em diversos modelos de smartphones.

Uma das tendências do mercado de desenvolvimento de sistemas digitais é projetar a arquitetura do produto desde o início tendo como foco a ideia de “mobile first” (móvel primeiro). Um dos divulgadores do conceito é o designer Luke Wroblewski. Desde 2011, o designer aborda a questão explicando porque sites e aplicativos devem ser projetados para os dispositivos móveis. “Projetar para dispositivos móveis primeiro não apenas prepara você para o crescimento explosivo e as oportunidades neste espaço, mas também obriga a focar e permite inovar”, argumenta o designer. Wroblewski apresenta três pontos em favor da ideia: o mobile cresce como uma nova oportunidade, permite uma aplicação com foco e é capaz de trazer inovações para os sistemas. Ele lembra que é preciso considerar fatores como múltiplos tamanhos de telas com a tecnologia de desempenho otimizado com acesso touch e sensores de movimento e localização (WROBLEWSKI, 2011).

1.2 Motivação

Os números do mercado mobile chegam às cifras de bilhões. Os dados mais aprofundados sobre a presença dos dispositivos mobiles são da Global System for Mobile Communications (GSMA), entidade baseada em Londres que congrega mais de 1.200 empresas operadoras de redes mobile.

Relatório divulgada pela GSMA, em fevereiro de 2019, contabiliza 5,1 bilhões de pessoas em todo o mundo assinando serviços móveis, o que representa 67% da população global. Até 2025, a estimativa é de que mais 710 milhões de pessoas assinarão pela primeira vez serviços móveis (5,8 bilhões – 71% da população). Dessas conexões hoje, 60% são acessadas por smartphones, sendo 79% na estimativa para 2025. Segundo a pesquisa, atualmente 3,6 bilhões de pessoas acessam a internet pelo mobile (47% da população), chegando a 5 bilhões em sete anos (61% da população). O GSMA informa que atualmente 43% das conexões são do tipo 4G, ultrapassando o 2G, com a estimativa de chegar a 59% em 2025. Em sete anos, o G5 deve ter 1,4 bilhão de conexões. A indústria mobile já representa 4,6% do PIB mundial (US\$ 3,9 trilhões), gerando 14 milhões de empregos

diretos e 17 milhões de empregos indiretos. Em paralelo, a Internet das Coisas passará das atuais 9,1 bilhões de conexões para 25,2 bilhões. “Enquanto a ubiquidade dos smartphones significa que continuam a ser o ponto focal da economia da internet do consumidor, a variedade de dispositivos conectados é agora maior do que nunca. Os consumidores digitais de hoje (que usam PCs e smartphones) devem se tornar os clientes de tecnologias emergentes, como Inteligência Artificial”, analisa o relatório publicado pelo instituto (GSMA, 2019).

Nos três primeiros meses de 2019, 30 bilhões de aplicativos foram baixados das lojas Google Play e iTunes, um crescimento de 10% em relação ao ano anterior, segundo relatório divulgado pela consultoria App Annie. A Google, dona do sistema Android, e a Apple, dona do sistema iOS, anunciaram um lucro de US\$ 22 bilhões com o consumo de aplicativos em suas lojas virtuais, Google Play (US\$ 8 bilhões) e iTunes (US\$ 14 bilhões) no período. Foi o trimestre mais lucrativo da história, sendo 20% acima do ano anterior. Segundo a consultoria, o crescimento foi impulsionado pelos mercados emergentes com o crescimento dos aplicativos de entretenimento e de serviços como os de transporte e de entrega de comida. (APP ANNIE, 2019)

No mercado brasileiro também é possível encontrar números que motivam o desenvolvimento, dentro do âmbito da Universidade, de projetos relacionados com aplicações mobile.

Em relatório divulgado em maio de 2019, a Brasscom (Associação Brasileira das Empresas de Tecnologia da Informação e Comunicação), que representa 70 empresas e 25 instituições do setor, prevê que serão necessários 70 mil profissionais na área de software e serviços por ano para que seja atingida a meta de dobrar o tamanho do setor até 2024. A entidade, que inclui a participação de instituições de ensino, contabiliza a formação de 46 mil profissionais com perfil tecnológico por ano no Brasil. Atualmente, o setor de Tecnologia da Informação e Comunicação (TIC) emprega 845 mil pessoas, um número 43 mil maior do que em relação a 2017. Enquanto a média de salário nacional é de R\$ 1.836, no setor de TIC a média do salário é de R\$ 4.444, subindo para R\$ 5.066 em software e serviços. O setor de TIC produziu no País R\$ 197,4 bilhões em 2018, um crescimento 2,9% em relação ao ano anterior (BRASSCOM, 2019).

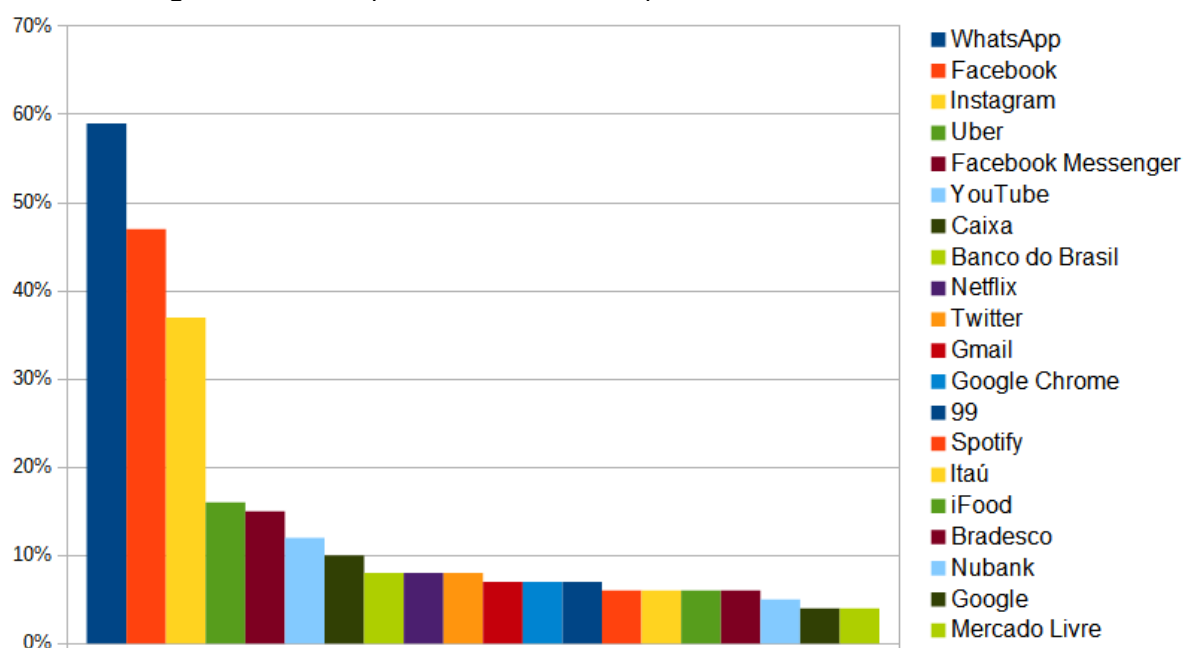
1.3 Justificativa

Com a tecnologia definida para este projeto (aplicativo mobile), é preciso justificar a escolha do tema que o sistema aborda, no caso em estudo uma mecânica de bicicleta com atuação local.

Por estarem presente em um dispositivo digital que tem como principal função a troca de informações, as aplicações mobile são eminentemente canais de comunicação entre duas partes. No campo dos negócios, recursos do smartphone (geolocalização, fotografia, captação e projeção de áudio e vídeo, identificação digital, sensor de movimento) permitem interações diferentes para a troca de produtos e serviços. O alcance das mudanças ainda será verificado, mas o mercado como um todo já foi impactado por essa tecnologia.

Divulgada em junho de 2019, pesquisa que monitora o uso de aplicativos no Brasil identifica um crescimento de sistemas de compartilhamento de transporte e de entrega de produtos. Segundo o levantamento “Panorama Mobile Time – Opinion Box”, feita em abril com 1.763 brasileiros que acessam internet e possuem smartphone, 99 e o iFood ocupam os 13º e 16º lugares entre os 20 mais populares (Figura 2). O Uber está na quinta posição, superando pela primeira vez o Facebook Messenger (MOBILE TIME; OPINION BOX, 2019).

Figura 2 – Os 20 aplicativos mais usados pelos brasileiros em abril de 2019



Fonte: Mobile Time - Opinion Box

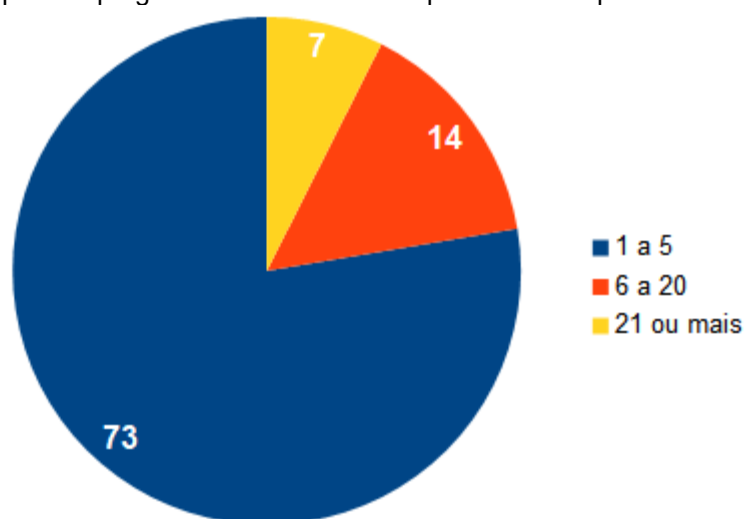
A pesquisa feita pelo site de notícias Mobile Time e pela empresa de consultoria Opinion Box apresenta outras conclusões sobre o mercado de aplicativos brasileiro. De acordo com o levantamento, subiu de 46% em abril de 2017 para 58% em abril de 2019 a proporção de internautas que já fizeram compras com “apps”. Entre os entrevistados, 91% usam Android, 7% iOS e 2% outros sistemas. Já baixaram aplicativos para o smartphone 96% dos entrevistados. Usa smartphone há mais de três anos 62% dos entrevistados, segunda a pesquisa. A compra de um aplicativo foi realizada por 19% das pessoas. (MOBILE TIME; OPINION BOX, 2019)

Para este projeto, é definido que deveria tratar de um serviço offline solicitado através de um meio online, tipo de aplicativo que tem demonstrado crescimento na utilização. A mecânica de bicicleta FixBi é escolhida como exemplo por ser um negócio de baixa complexidade, por haver um aumento no seu uso como transporte na cidade de São Paulo, por ser a bicicleta ecologicamente correta, sem a emissão de poluentes, e saudável como exercício físico para os usuários.

Para elaborar um perfil de serviço que pode ser oferecido pelo aplicativo FixBi é organizado dentro deste projeto uma pesquisa¹ divulgada pela internet entre ciclistas que moram na cidade de São Paulo. O levantamento obteve 94 respostas que trazem algumas percepções para o desenvolvimento do aplicativo.

O uso da bicicleta é eventual, com a maioria dos ciclistas andando até cinco vezes por mês (Figura 3).

Figura 3 – Resposta à pergunta: utiliza a bicicleta quantas vezes por mês?

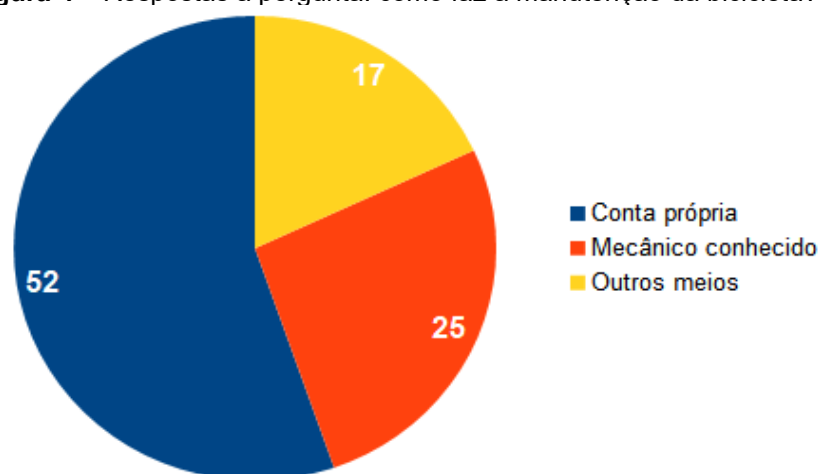


Fonte: autor

1 A pesquisa permanece em aberto podendo ser consultada no site Formulários Google: <https://forms.gle/kFi9nttAsb79ANecA>.

Dos 94 entrevistados, 52 dizem fazer eles mesmos a manutenção da bicicleta (Figura 4), mostrando tendência de espaço para serviços especializados – 25 entrevistados disseram não haver uma mecânica de bicicleta próximo da sua casa. Os problemas mais comuns citados foram pneu furado, regulagem de roda, de freio e de câmbio, quebra de corrente, troca de peças e limpeza. Na pesquisa, 37 afirmam que deixaram de passear com a bicicleta por não saber como arrumá-la.

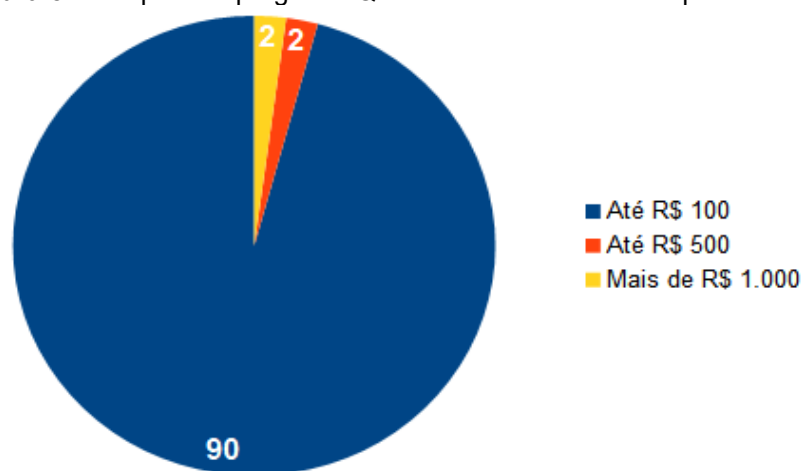
Figura 4 – Respostas à pergunta: como faz a manutenção da bicicleta?



Fonte: autor

A ampla maioria dos entrevistados afirmam que gastam até R\$ 100 por mês com a manutenção de suas bicicletas, apontando para baixa complexidade dos serviços e seu viés predominante local (Figura 5).

Figura 5 – Resposta à pergunta: Quanto investe na bicicleta por mês?



Fonte: autor

2 REVISÃO METODOLÓGICA

2.1 Lógica de programação

Escrever um programa de computador é como contar uma história com lógica. A habilidade de ler e escrever textos está presente na ampla maioria da população, enquanto a capacidade de ler e escrever códigos é algo raro, o que tende a mudar nas próximas décadas.

A lógica, que tem a origem em Aristóteles (384 a.C – 322 a.C), é a análise de métodos de raciocínio, tendo como interesse primordialmente a forma e não o conteúdo dos argumentos. Conforme definição feita por João Nunes de Souza, professor de lógica para ciência da computação na Universidade Federal de Uberlândia, ao se incrementar a utilização da lógica, ela se torna o estudo do raciocínio, do pensamento correto e verdadeiro, da demonstração científica, de pensamentos não científicos, ou de regras para verificação da verdade ou falsidade de um pensamento (SOUZA, 2015).

No livro “A history of modern computing”, o especialista Paul Ceruzzi define que os computadores foram inventados para “computar”, o seja resolver problemas matemáticos complexos. Mas hoje os computadores armazenam e recuperam dados, gerenciam redes de comunicações, processam multimídia, entre outras funções. Ceruzzi identifica que apenas nos anos 1940 a palavra “computador”, que era usado para pessoas que resolviam equações, foi pela primeira vez transferida para uma máquina. “Não é que o computador acabou não sendo usado para cálculos. Isso, pelo menos, seus inventores previram. Mas as pessoas encontram maneiras de fazer para que fizesse muito mais”, comenta. Na obra, Ceruzzi apresenta um resumo da área da computação entre a década de 1940 e 1990 apresentando as linguagens de programação, a evolução dos dispositivos eletrônicos e os principais atores (CERUZZI, 1998).

2.2 Metodologia ágil (Scrum)

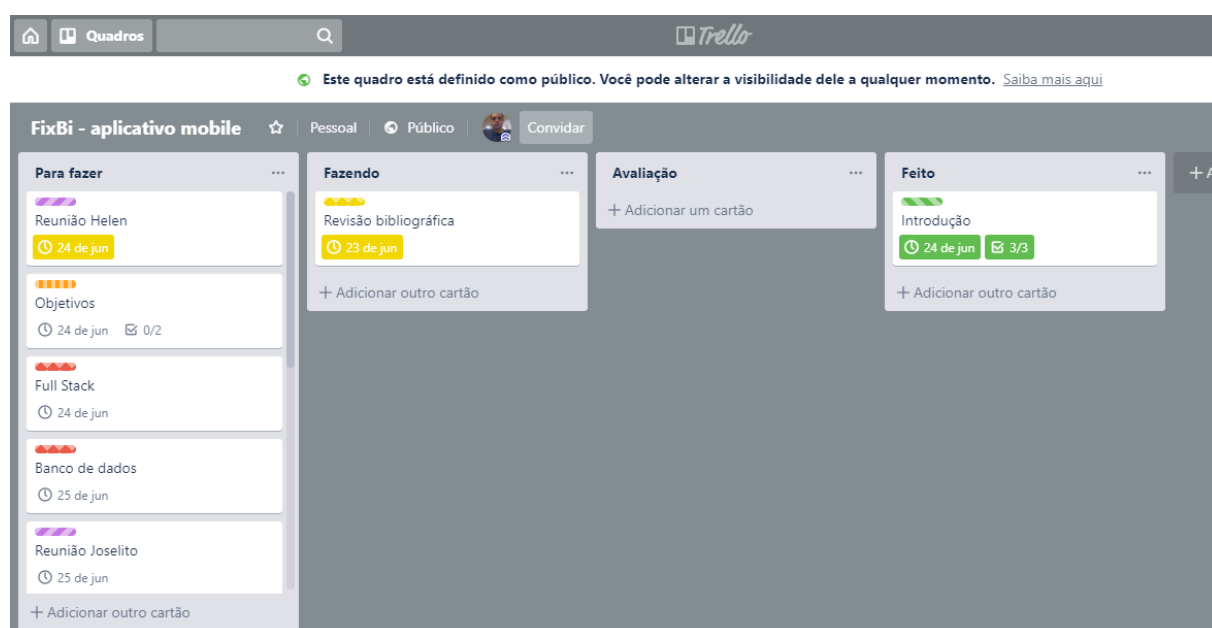
A popularização do computador eletrônico gerou uma demanda por sistemas computacionais, em especial em organizações como empresas e instituições públicas. Um mercado profissional de desenvolvimento de software tem crescido continuamente. O método cascata predominou na gestão desses projetos durante décadas. A metodologia ágil, porém, tem ganhado espaço nas organizações de tecnologia. Uma das ferramentas mais conhecidas o chamado Scrum.

A prova de que esse método tem interesse é a presença constante na lista dos mais vendidos do livro “Scrum: a arte de fazer o dobro do trabalho na metade do tempo”, de Jeff Sutherland. Ex-piloto de caça da Guerra do Vietnã, o consultor em tecnologia afirma ter criado, com Ken Schwaber, o método Scrum em 1993 como uma forma mais “rápida, confiável e eficiente” de desenvolver softwares da indústria de tecnologia. Segundo o especialista, até 2005 a maior parte do desenvolvimento era executado pelo método cascata, que considera “lento, imprevisível, e muitas vezes não resultava em um produto que as pessoas quisessem ou pelo qual se dispusessem a pagar”. Menciona que muitos projetos utilizavam os diagramas de Gantt, ferramenta inventada na década de 1910. De acordo com Sutherland, o Scrum é uma mudança racional se assemelhando a sistemas “evolucionários, adaptativos e autocorretivos”. A origem da palavra é uma jogada de rúgbi em que um time se une para avançar a bola. O Scrum é feito para equipes pequenas que dividem o trabalho em períodos curtos de tempo, chamadas de sprints. Nas sprints são entregues partes funcionais do projeto. Entre os princípios do conceito estão o reconhecimento de que os requisitos de projetos são alterados e que surgem desafios imprevistos. O foco deve ser assim em entregar partes mais funcionais dos projetos com adaptações no rumo do produto (SUTHERLAND, 2014).

Sutherland é um dos 17 desenvolvedores que assinaram em 2001 o Manifesto Ágil. O documento elabora uma série de valores e princípios que os desenvolvedores precisariam ter. Os valores são (1) indivíduos e interações são mais importantes que processos e ferramentas; (2) software em funcionamento mais importante que documentação abrangente; (3) colaboração com o cliente mais importante que negociação de contratos; (4) responder a mudanças mais importante que seguir um plano. “Estamos descobrindo maneiras melhores de desenvolver software, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo”, afirmam os cientistas no documento. O documento elenca ainda 12 princípios: (1) nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de software de valor; (2) aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas; (3) entregar software funcionando com frequência, na escala de semanas até meses, com preferência aos períodos mais curtos; (4) pessoas relacionadas aos negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto; (5) construir projetos ao redor de

indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho; (6) o método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa cara a cara; (7) software funcional é a medida primária de progresso; (8) processos ágeis promovem um ambiente sustentável. (9) Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter indefinidamente, passos constantes; contínua atenção à excelência técnica e bom design, aumenta a agilidade; (10) simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito; (11) as melhores arquiteturas, requisitos e designs emergem de times auto-organizáveis; e (12) em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo (AGILE MANIFESTO, 2001).

Figura 6 – Quadro scrum do projeto Fixbi no site Trello



Fonte: autor

Uma das ferramentas mais interessantes do Scrum é o seu quadro, no qual não são colocadas “tarefas”, mas “histórias” que estão divididas em “para fazer” (ou backlog), “fazendo”, “em avaliação” e “concluído”. A cada sprint um número de “histórias” devem ser concluídas. As que não foram realizadas, voltam para a avaliação da equipe.

Uma imagem comum nas mídias, em especial nos anúncios publicitários, é de equipes trabalhando em quadros com os famosos post-its (pequenos pedaços

papéis que podem ser colados em superfícies). Existem hoje diversas ferramentas que permitem a gestão do quadro de maneira online e colaborativa. Para este projeto foi escolhido o aplicativo Trello (Figura 2)².

2.3 Controle de versões (Git)

O versionamento dos códigos do projeto usa a ferramenta Git, sistema de controle de versões distribuído usado para desenvolvimento de software e para o registro de edições de qualquer tipo de arquivo. A tecnologia foi inicialmente desenvolvida em 2005 pelo cientista Linus Torvalds, que também criou o sistema operacional Linux (TORVALDS, 2007).

Na programação de computadores, é comum que sistemas apresentem novos problemas durante o desenvolvimento sendo necessário retornar para versões anteriores que não apresentavam defeitos. No Git, cada diretório é um repositório com um histórico completo que pode ter as revisões acompanhadas. Em projetos em equipe, permite o trabalho colaborativo com a identificação das mudanças. É possível utilizar o Git para qualquer projeto científico. “O Git é um sistema de controle de versão distribuído desenvolvido por Junio Hamano e Linus Torvalds. O Git não usa um servidor centralizado. O Git é executado em sistemas operacionais Linux, BSD, Solaris, Darwin, Windows, Android e outros sistemas operacionais” (GIT WIKI, 2010).

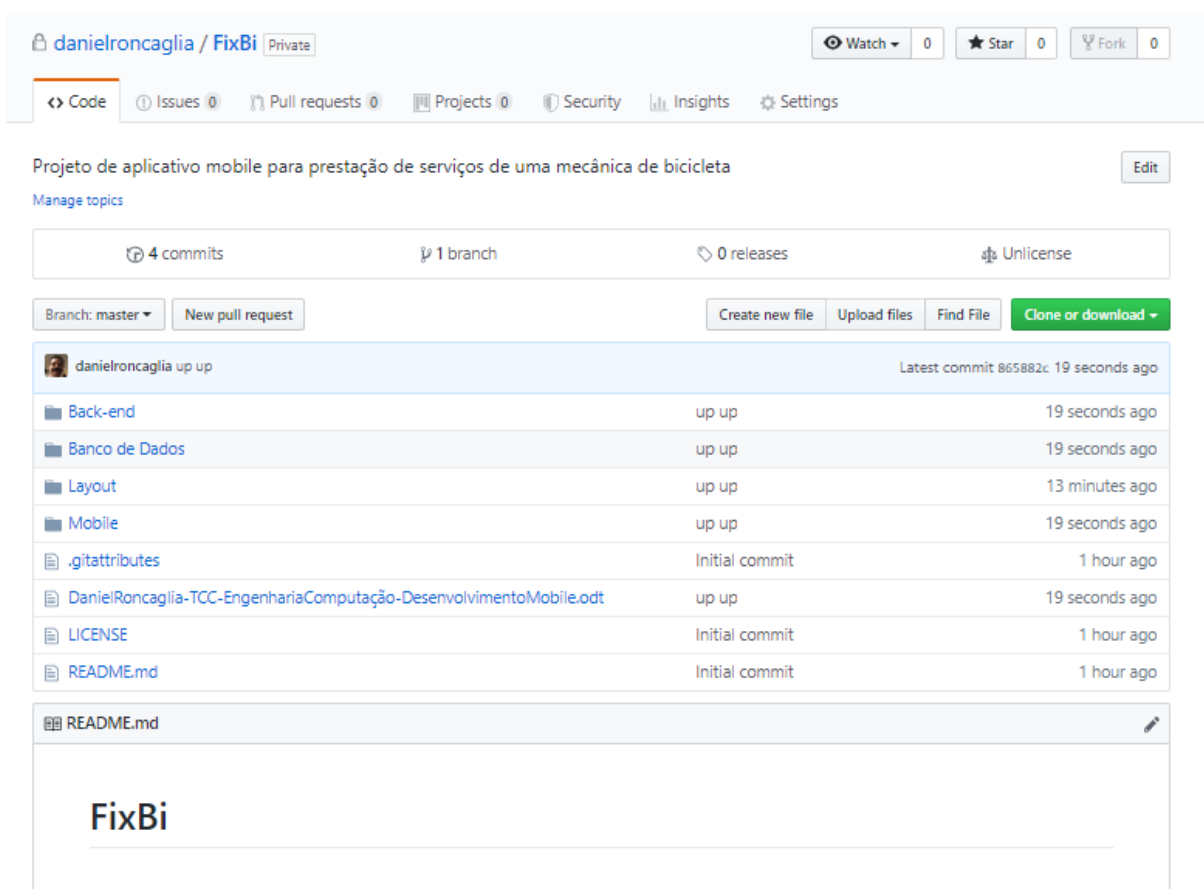
Em 2008, a tecnologia vira uma empresa chamada GitHub, serviço online de armazenamento de controle de versões de desenvolvimento de software. A Microsoft compra a empresa em 2018 por US\$ 7,5 bilhões. Em abril de 2019, a GitHub informa abrigar 100 milhões de repositórios, tendo 36 milhões de usuários (GITHUB, 2019).

Em síntese, o GitHub é a rede social favorita dos desenvolvedores. Para este projeto, foi criado um repositório no perfil do autor (Figura 7)³ no GitHub. Foi utilizado a versão desktop do GitHub que é de utilização intuitiva. É preciso porém certo conhecimento sobre utilização de sistemas de computador.

2 O quadro no Trello está disponível em: <https://trello.com/b/BjDNU3Zv/fixbi-aplicativo-mobile>

3 O repositório no GitHub está disponível em: <https://github.com/danielroncaglia/FixBi>

Figura 7 – Imagem do repositório do projeto FixBi no sistema Git



Fonte: autor

2.4 Processos de criação

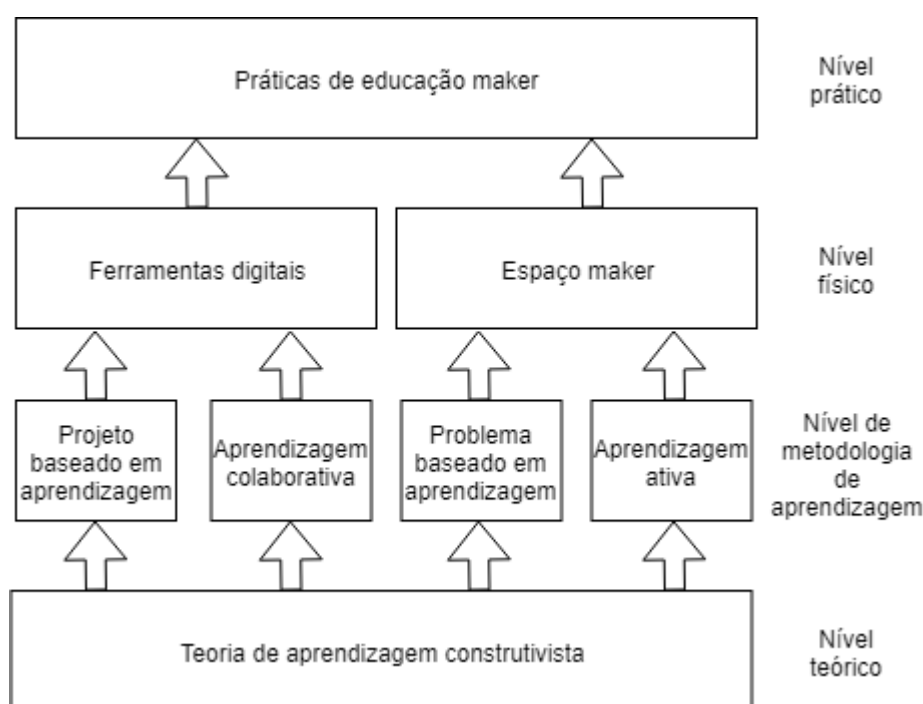
Outras metodologias modernas de elaboração de projetos, em especial as que envolvem tecnologia, também serviram como referencial para o projeto.

Uma delas é o design thinking, que é uma metodologia para resolução de problemas que tem as pessoas como centro do desenvolvimento do projeto. Criado e desenvolvido a partir dos anos 1960, o método é usado nos meios acadêmicos e empresariais para diversos tipos de projetos. Tem a característica de ser cíclico com as etapas ouvir, criar e implementar. Ouvir é o momento de aproximação dos interessados para compreender suas necessidades. Pode ser aplicado de diversas formas como visitas, entrevistas, pesquisas. Para este projeto, a etapa inicial foi realizada com questionário online sobre o uso de bicicleta. Criar é a etapa na qual os dados coletados são analisados de forma a se criar padrões e definir oportunidades. Podem ser usadas ferramentas como mapa conceitual, cartões insights, brainstorming (tempestade de ideias), entre outros. Neste projeto, esta fase ocorre

com a projeção para se desenvolver um aplicativo mobile para resolução do problema levantado na pesquisa. Implementar refere-se às ideias abstratas que precisam ser validadas e, para isso, necessitam ganhar forma material. Nesta fase, essas são transformadas em aplicações concretas. A implementação tem o objetivo de identificar quais são as capacidades necessárias para a realização do projeto. Realiza-se projeto-piloto para que consequentemente se garanta soluções eficazes e permita um processo contínuo de aprendizado e desenvolvimento. Neste projeto, a implementação é o repositório com os códigos do aplicativo (IDEO, 2009).

Outra referência é o movimento maker, que é uma evolução do faça-você-mesmo (do-it-yourself). Os dois possuem a característica de imaginar que qualquer pessoa pode desenvolver novas soluções para problemas cotidianos. O surgimento de equipamentos de fabricação digital em código aberto aponta uma nova tendência para o movimento maker destaca artigo “The influence of the maker movement on engineering and technology education”, assinado por equipe de especialistas chineses da Sichuan Open University. O natural relacionamento do movimento com a engenharia e a tecnologia chamaram a atenção de educadores (Figura 8). O artigo apresenta uma série de casos de aplicação do pensamento maker em cursos de engenharia (TAN, YANG, YU, 2016).

Figura 8 – Diagrama com os quatro níveis da aprendizagem maker



Fonte: TAN, YANG, YU

3 OBJETIVOS

3.1 Objetivo geral

O projeto tem o objetivo de desenvolver de maneira ágil um aplicativo mobile que server como exemplo para operações que necessitam de comunicação online para prestações de serviços offline com usuários do tipo profissional e cliente e com as funções cadastrar e listar atendimentos.

3.2 Objetivos específicos

- Documentar a realização do desenvolvimento de uma aplicação mobile;
- Pesquisar sobre a economia mobile e traçar tendências sobre a questão;
- Apresentar de maneira sucinta as ferramentas mais atuais do desenvolvimento web no modo full stack;
- Sistemar plano de desenvolvimento com ferramentas recentes;
- Trabalhar com a criação e a manipulação de banco de dados do tipo SQL;
- Codificar uma aplicação back-end em linguagem C#;
- Desenhar layouts de páginas do tipo login, cadastrar e listar para celulares;
- Executar uma interface emulando dispositivo Android através do React Native;

4 APLICATIVO FIXBI

4.1 Full Stack

Um dos profissionais mais valorizados no mercado de desenvolvimento de softwares é o full stack web developer, que é a pessoa capaz de desenvolver o cliente e o servidor do software. Além de saber linguagens como HTML e CSS, típicas do front-end, o full stack programa o browser (JavaScript ou React Native), o servidor (C# ou Node) e o banco de dados (SQL e MongoDB). Principal referência educacional de tecnologias web, o site W3Schools elenca uma série de vantagens e desvantagens sobre essa abordagem. Um bom desenvolvedor full stack pode dominar as técnicas envolvidas no projeto, fazer um protótipo rapidamente, ajudar outros membros da equipe, reduzir os custos e tempo usado em comunicação e entender melhor os aspectos das tecnologias. As desvantagens são a possibilidade de ser uma escolha errada para o projeto, do programa depender apenas de uma pessoa chave e da complexidade das ferramentas (W3SCHOOLS, 2019).

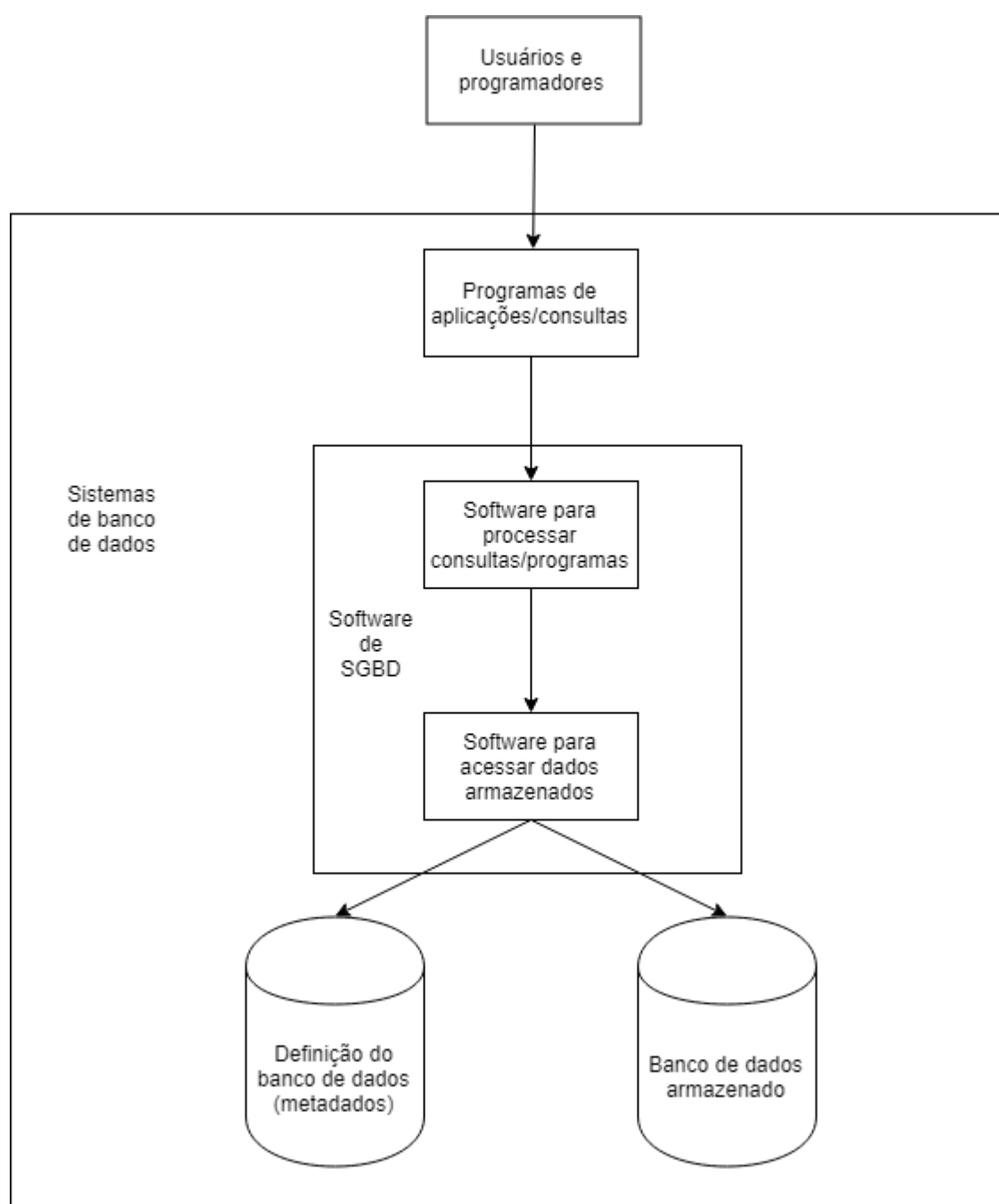
Programas de treinamento para desenvolvimento de aplicativos têm sido aberto por diversas instituições diante da demanda do mercado por profissionais. Um dos centros de excelência é a escola Senai Informática, localizada no centro de São Paulo, que oferece, entre outros cursos, um treinamento de nível técnico em desenvolvimento web. O projeto utiliza o roteiro de curso ministrado pelos instrutores Helena Strada e Fernando Henrique Guerra sobre desenvolvimento de aplicações web (STRADA; GUERRA, 2019).

4.2 Banco de dados (SQL)

O fundamento de um aplicativo mobile é seu banco de dados. Os sistemas de banco de dados são componentes essenciais na sociedade contemporânea, com seu impacto se tornando maior com o uso de computadores. Um banco de dados é uma coleção de dados relacionados, que são fatos como nome ou número de telefones. “Um banco de dados é uma coleção logicamente coerente de dados com algum significado inerente. Uma variedade aleatório de dados não pode ser corretamente chamada de banco de dados”, explicam os professores Ramez Elmasri e Shamkant Navathe, no livro “Sistemas de banco de dados”. Bancos de dados podem ter qualquer tamanho e complexidade, chegando ao bilhão como no caso do Facebook. Com os computadores surgiram nos anos 1970 os sistemas de banco de

dados (SGDB). Esses sistemas (Figura 9) são um software de uso geral para facilitar o processo de definição, construção, manipulação e compartilhamento de bancos entre usuários e aplicações. Um banco de dados pode ter um ciclo de vida de muitos anos, fazendo que o SGDB precise ser capaz de manter o sistema permitindo a sua evolução à medida que os requisitos mudam (ELMASRI; NAVATHE, 2007).

Figura 9 – Diagrama simplificado de um ambiente de banco de dados



Fonte: ELMASRI; NAVATHE

A modelagem de dados de um software apresenta três níveis: lógico, conceitual e físico. Para este projeto, no nível lógico, foram elaborados cinco tabelas: TIPO_USUARIOS (Tabela 1), USUARIOS (Tabela 2), CICLISTAS (Tabela 3), MECANICOS (Tabela 4) e ATENDIMENTOS (Tabela 5).

Tabela 1 – TIPO_USUARIOS

ID_TIPO_USUARIO	TIPO_USUARIO
1	Mecanico
2	Ciclista

Fonte: autor

Tabela 2 – USUARIOS

ID_USUARIO	ID_TIPO_USUARIO	EMAIL_USUARIO	SENHA_USUARIO
1	2	mariana@fixbi.com	12345
2	2	suely@fixbi.com	12345
3	2	correia@fixbi.com	12345

Fonte: autor

Tabela 3 – CICLISTAS

ID_CICLISTA	ID_USUARIO	NOME_CICLISTA	TELEFONE	INFORMACOES
1	2	Mariana Santos	(11)9999-1002	MTB
2	3	Suely Santos	(11)9999-1003	Iniciante
3	4	José Correia	(11)9999-1004	Cargueira

Fonte: autor

Tabela 4 – MECANICOS

ID_MECANICO	ID_USUARIO	NOME_MECANI	TELEFONE	INFORMACOES
1	1	Daniel Santos	(11)9999-1001	Câmbio

Fonte: autor

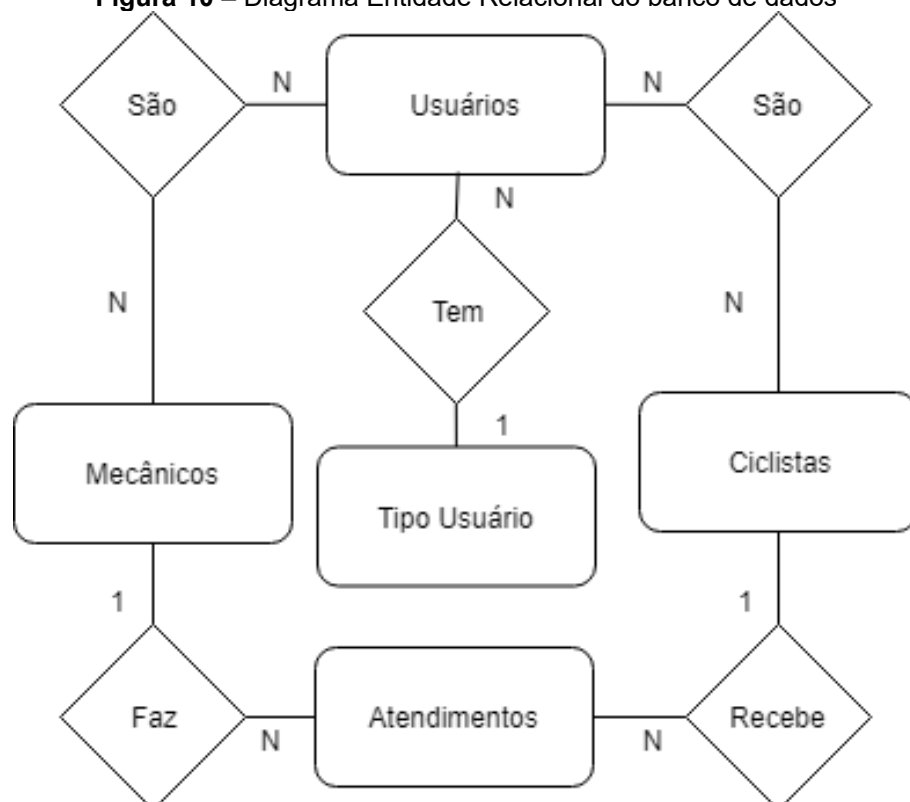
Tabela 5 – ATENDIMENTOS

ID_ATENDIM.	ID_CICLISTA	ID_MECANI	DATA/HORA	DESCRICAO	SITUACAO
1	1	4	26/6/19 -15h	Pneu furado	Realizada
2	2	4	27/7/19 – 14h	Freio	Agendada
3	3	4	1/7/19 – 16h	Câmbio	Cancelada

Fonte: autor

Uma das fases mais importantes de um projeto é a modelagem conceitual do banco de dados, como explicam Ramez Elmasri e Shamkant Navathe. O modelo Entidade-Relacionamento (ER) é um dos métodos mais populares, sendo apresentados através dos diagramas ER (Figura 10). Durante o levantamento e análise de requisitos da aplicação, são documentados os requisitos de dados e funcionais que o banco de dados precisa ter para as aplicações. O conceito básico do modelo ER é a entidade, que representa alguma coisa do mundo real, como um grupo de pessoas, ou um objeto conceitual, como, por exemplo, o cargo de mecânico. A entidade possui atributos, como nome, endereço, telefone. Os atributos podem ser divisíveis: o endereço tem rua, cidade, CEP. As entidades são preenchidas por valores, como o nome do mecânico. No banco de dados, as entidades se relacionam havendo diferentes instâncias: mecânico atende um ciclista. Os relacionamentos entre as instâncias contêm restrições entre as entidades que são chamadas de razão de cardinalidade (1:1, 1:N, N:N). Exemplo: um mecânico (1) pode realizar diversos atendimentos (n) (ELMASRI; NAVATHE, 2007).

Figura 10 – Diagrama Entidade Relacional do banco de dados



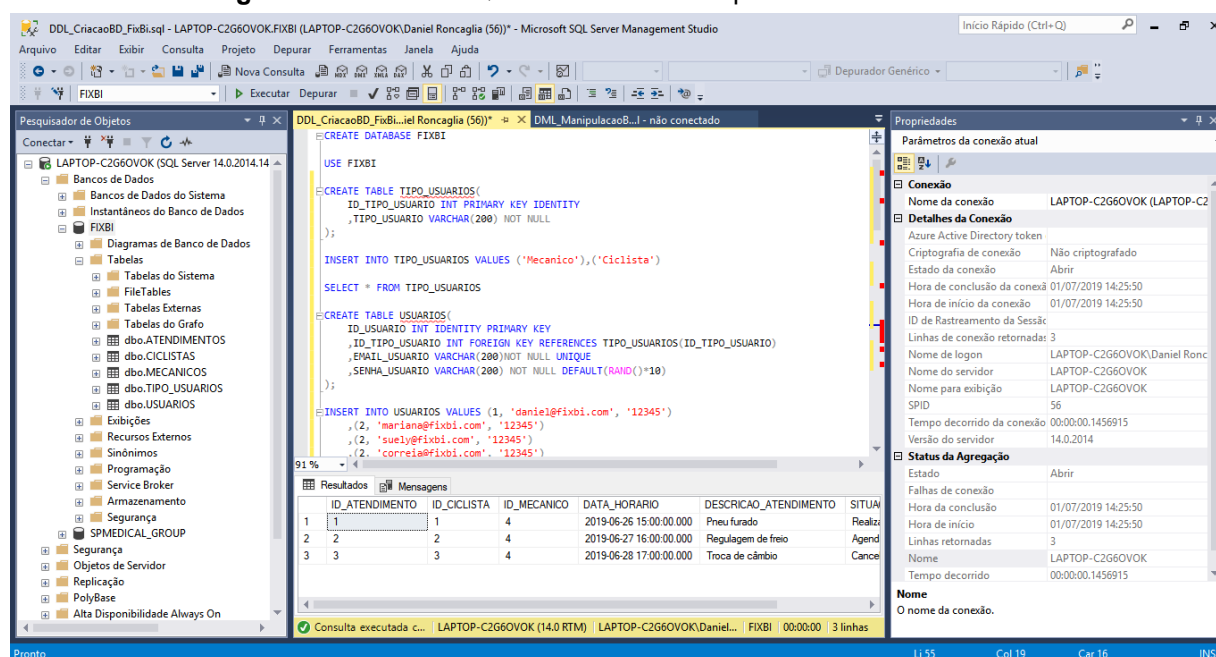
Fonte: autor

No nível físico, a implementação do banco de dados ocorre por script SQL utilizando como programa o SQL Server Management Studio, da Microsoft. Segundo a empresa, um projeto de script de banco de dados é um conjunto organizado de scripts, de informações de conexão e de modelos que estão associados a um banco de dados ou a uma parte de um banco de dados (MICROSOFT, 2017).

O SQL Server Management Studio da Microsoft (Figura 11) possui versões pagas, mas a versão Developer é gratuita para uso e download⁴. O site da empresa fundada pelo empresário Bill Gates também dispõe de documentação sobre o funcionamento do programa.

O Management Studio é um conjunto de ferramentas administrativas para gerenciar os componentes pertencentes ao SQL Server. Esse ambiente permite que os usuários executem tarefas como fazer backup de dados, editar consultas e automatizar funções.

Figura 21 – Tela do SQL Server com os scripts do banco de dados



Fonte: autor

4 O download está disponível em: <https://www.microsoft.com/pt-br/sql-server/sql-server-downloads>

Para a criação do banco de dados (Figura 12), são utilizados conjuntos de comandos SQL chamados de DDL (Data Definition Language), que em português significam Linguagem de Definição de Dados, responsáveis pela definição dos dados, ou seja, pela criação de bancos, esquemas, tabelas, campos, tipos de dados, etc. O comando mais evidente é CREATE TABLE.

Para a inserção de dados são utilizados scripts do tipo DML (Data Manipulation Language), que em português significa Linguagem de Manipulação de Dados. No caso, o comando INSERT TABLE. Para a validação do banco de dados, foram inseridos as informações presentes nas tabelas feitas no nível lógico com a simulação de cadastros de mecânicos e ciclistas.

Para a visualização dos dados inseridos nos bancos, são usados scripts do tipo Data Query Language), que em português significa Linguagem de Consulta de Dados. O comando de consulta é SELECT.

Figura 12 – Scripts DDL, DML e DGL do banco de dados Fixbi

```
CREATE DATABASE FIXBI
USE FIXBI

CREATE TABLE TIPO_USUARIOS(
    ID_TIPO_USUARIO INT PRIMARY KEY IDENTITY
    ,TIPO_USUARIO VARCHAR(200) NOT NULL
);

INSERT INTO TIPO_USUARIOS VALUES ('Mecanico'),('Ciclista')
SELECT * FROM TIPO_USUARIOS

CREATE TABLE USUARIOS(
    ID_USUARIO INT IDENTITY PRIMARY KEY
    ,ID_TIPO_USUARIO INT FOREIGN KEY REFERENCES TIPO_USUARIOS(ID_TIPO_USUARIO)
    ,EMAIL_USUARIO VARCHAR(200)NOT NULL UNIQUE
    ,SENHA_USUARIO VARCHAR(200) NOT NULL DEFAULT(RAND()*10));

INSERT INTO USUARIOS VALUES (1, 'daniel@fixbi.com', '12345')
    ,(2, 'mariana@fixbi.com', '12345')
    ,(2, 'suely@fixbi.com', '12345')
    ,(2, 'correia@fixbi.com', '12345')
SELECT * FROM USUARIOS
```

```

CREATE TABLE CICLISTAS(
    ID_CICLISTA INT PRIMARY KEY IDENTITY
    , ID_USUARIO INT FOREIGN KEY REFERENCES USUARIOS(ID_USUARIO)
    , NOME_CICLISTA VARCHAR(200) NOT NULL
    , TELEFONE_CICLISTA CHAR(14) NOT NULL UNIQUE
    , INFORMACOES_CICLISTA VARCHAR(200)
);

INSERT INTO CICLISTAS VALUES (2, 'Mariana Santos', '(11)9999-1001', 'Tem uma MTB')
    , (2, 'Sueley Santos', '(11)9999-1002', 'Vai comprar nova bike')
    , (3, 'José Santos', '(11)9999-1003', 'Tem uma cargueira')
SELECT * FROM CICLISTAS

CREATE TABLE MECANICOS(
    ID_MECANICO INT PRIMARY KEY IDENTITY
    , ID_USUARIO INT FOREIGN KEY REFERENCES USUARIOS(ID_USUARIO)
    , NOME_MECANICO VARCHAR(250) NOT NULL
    , TELEFONE_MECANICO CHAR(14) NOT NULL UNIQUE
    , INFORMACOES_MECANICO VARCHAR(200)
);

INSERT INTO MECANICOS VALUES (1, 'Daniel Santos', '(11)9999-1000', 'Câmbios')
SELECT * FROM MECANICOS

CREATE ATENDIMENTOS(
    ID_ATENDIMENTO INT PRIMARY KEY IDENTITY
    , ID_CICLISTA INT FOREIGN KEY REFERENCES CICLISTAS(ID_CICLISTA)
    , ID_MECANICO INT FOREIGN KEY REFERENCES MECANICOS(ID_MECANICO)
    , DATA_HORARIO DATETIME NOT NULL
    , DESCRICAO_ATENDIMENTO VARCHAR(200)
    , SITUACAO_ATENDIMENTO VARCHAR(200) NOT NULL
);

INSERT INTO ATENDIMENTOS VALUES
(1, 4, '2019-06-26T15:00:00', 'Pneu furado', 'Realizada')
, (2, 4, '2019-06-27T16:00:00', 'Regulagem de freio', 'Agendada')
, (3, 4, '2019-06-28T17:00:00', 'Troca de câmbio', 'Cancelada')

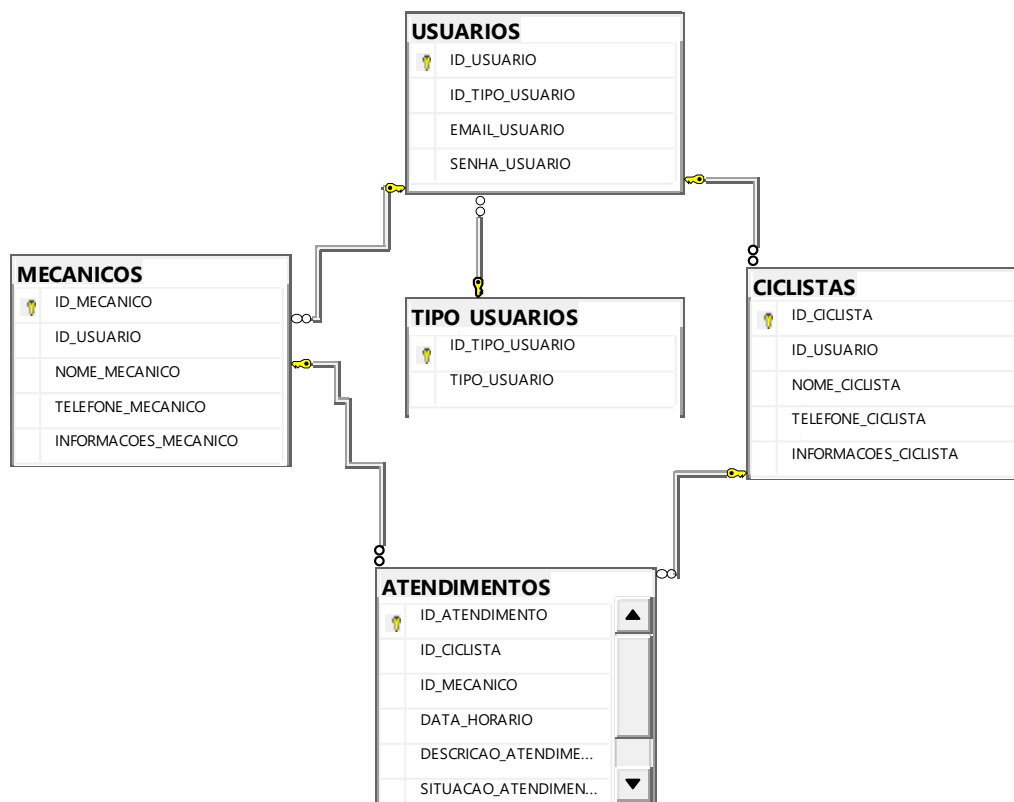
SELECT * FROM ATENDIMENTOS

```

Fonte: autor

Após a criação do banco de dados, é possível visualizar o diagrama das tabelas (Figura 13) criado pelo SQL Server.

Figura 13 – Diagrama do banco de dados gerado pelo SQL Server



Fonte: autor

4.3 Back-end (API – C#)

Na indústria de desenvolvimento web, dois termos muito utilizados são back-end e front-end. Em linhas gerais, a definição de front-end é aquilo que o usuário enxerga em especial o design das publicações, ou seja, a interface com o usuário. Um desenvolvedor de front-end trabalha em geral com linguagens HTML, CSS e JavaScript, conforme aponta Pluralsight, site de educação de tecnologia web. Já o back-end é a camada que faz o acesso ao banco de dados, sendo o “lado do servidor”. Basicamente, é o que permite o site funcionar com atualizações e mudanças. Seu funcionamento não aparece na página (PLURALSIGHT, 2015).

Para este projeto, o back-end é implantado através de uma API (Application Programming Interface), que em português é Interface de Programação de Aplicação. A API é uma espécie de “caminho” que permite a comunicação entre aplicações. Conforme a especialista Perry Eising, em artigo publicado no site Medium, em geral o termo é usado para se referir às APIs web, que retornam dados do tipo JSON ou XML. “A API não é o banco de dados nem o servidor, é o código que controla os pontos de acesso do servidor”, explica Eising. As APIs permitem que os sites alterem dados em outros aplicativos. Exemplos comuns de API são os botões de compartilhar conteúdos de sites no Facebook e Twitter. Grandes empresas de tecnologia, especialmente empresas de mídia social, frequentemente disponibilizam seus dados agregados para o público, mas APIs também são mantidas por organizações governamentais e outros tipos de empresas. A API difere-se de um site web suportado por um banco de dados estático. Na API, solicitações para recuperar ou gravar dados geralmente são feitas sem um front-end, enviando uma solicitação HTTP (para um servidor. É muito utilizado o JSON (JavaScript Object Notation), que em português significa Notação de Objetos JavaScript. O JSON é basicamente uma maneira de representar dados que se parecem com objetos JavaScript. Por ser legível, leve e funcional, é muito utilizado nas aplicações mais modernas. Um objeto JSON em geral tem este formato: “id_usuario”: 1; “tipo_usuario”: “mecânico”; “email”: “daniel@fixbi.com”; “senha”: “12345” (EISING, 2017).

O REST (Representational State Transfer), em português Transferência de Estado Representacional, é a arquitetura de sistema distribuído usado em serviços web. Segundo o World Wide Web Consortium, entidade internacional que organiza os padrões da World Wide Web, a arquitetura foi proposta por Roy Fielding em 2000 em tese de doutorado “Architectural Styles and the Design of Network-based

Software Architectures” (University Of California Irvine). Os serviços da web em conformidade com o estilo arquitetural REST, chamado RESTful Web Services (RWS), fornecem interoperabilidade entre sistemas de computador na internet. Esses serviços permitem que os sistemas solicitantes acessem e manipulem representações textuais de recursos da web usando um conjunto uniforme e predefinido de operações sem estado (WORLD WIDE WEB CONSORTIUM, 2004).

As APIs RESTful baseadas em HTTP (Hypertext Transfer Protocol) são definidas com os seguintes aspectos: (1) um URI de base, como `http://api.example.com/collection/`; (2) métodos HTTP padrão (por exemplo, GET, POST, PUT, PATCH e DELETE); (3) um tipo de mídia que define elementos de dados de transição de estado. A representação atual informa ao cliente como compor solicitações de transições para todos os próximos estados de aplicativo disponíveis. Isso pode ser tão simples quanto um URI (Uniform Resource Identifiers) ou complexo como um applet Java. Os métodos HTTP significam GET (recuperar); POST (Criar); PUT (Substituir); PATCH (Atualizar); DELETE (Apagar) (FIELDING, 2008).

Para o projeto, é utilizado POST para cadastrar novo atendimento e GET para listar os atendimentos agendados, realizados e cancelados.

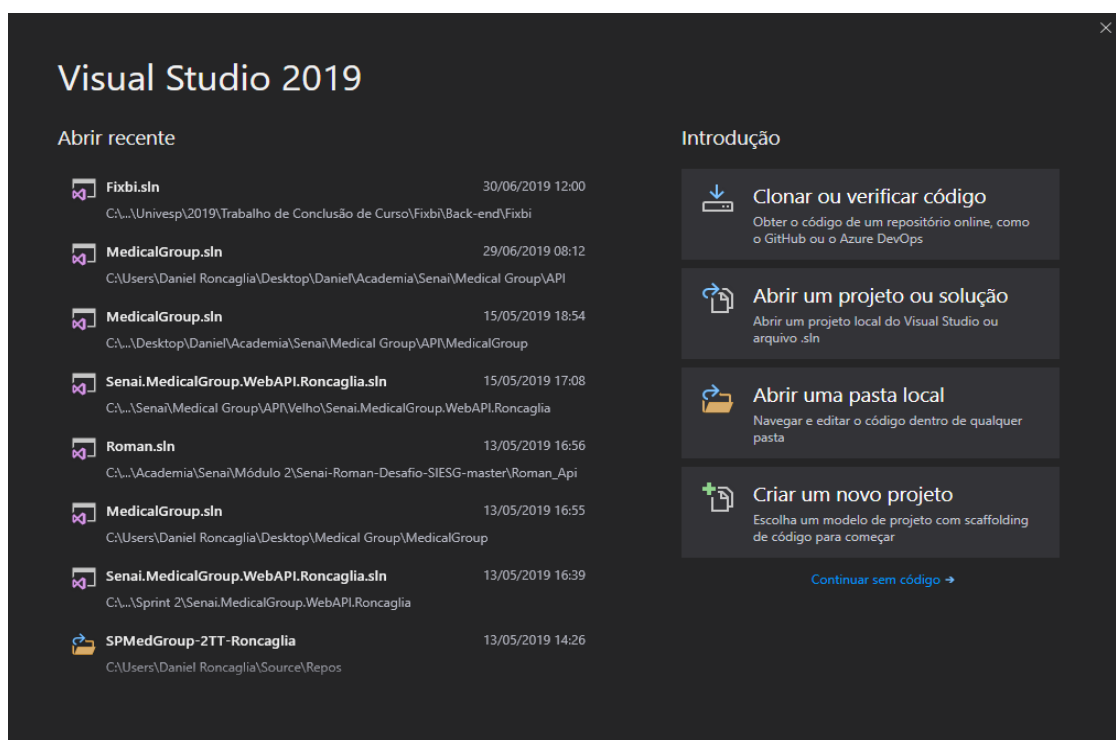
Como padrão de arquitetura, a API do Fixbi (Figura 19) utiliza o chamado MVC (Model – View – Controller), que separa a programação orientada a objetos nessas três partes. Com os componentes principais separados é possível reutilizar o código e o desenvolvimento paralelo, tornando a arquitetura popular para aplicações web. Linguagens de programação como JavaScript, Python, Ruby, PHP, Java e C # têm frameworks MVC. O Modelo é o componente central do padrão: estrutura de dados dinâmica da aplicação, independente da interface do usuário. A Visão é a representação da informações, como um gráfico, diagrama ou tabela. O controlador aceita a entrada do usuário e converte em comandos para o modelo ou visualização. O padrão foi proposto na década 1970 pelo cientista da computação Trygve Mikkjel Heyerdahl Reenskaug, professor emérito da Universidade de Oslo (REENSKAUG; COPLIEN, 2009).

A linguagem de programação utilizada para este projeto é C#, que faz parte da iniciativa ASP.NET. A pronúncia de C# é “See Sharp” e, de acordo com documentação da Microsoft, é uma simples, moderna, orientada a objetos e auto segura linguagem de programação. A raiz da linguagem é o C, sendo imediatamente familiar aos programadores C, C ++, Java e JavaScript. Segundo a Microsoft, o C # é

uma linguagem orientada a objetos, mas tem suporte para programação orientada a componentes. O design contemporâneo de software depende cada vez mais de componentes de software na forma de pacotes de funcionalidade autocontidos e auto-descritivos. A chave é que eles apresentam um modelo de programação com propriedades, métodos e eventos. Ainda segundo a Microsoft, os componentes possuem atributos que fornecem informações declarativas sobre o componente; e eles incorporam sua própria documentação. O C # fornece construções de linguagem para suportar diretamente esses conceitos, tornando o C # uma linguagem muito natural para criar e usar componentes de software (MICROSOFT, 2019).

Let's code (Figuras 14)⁵.

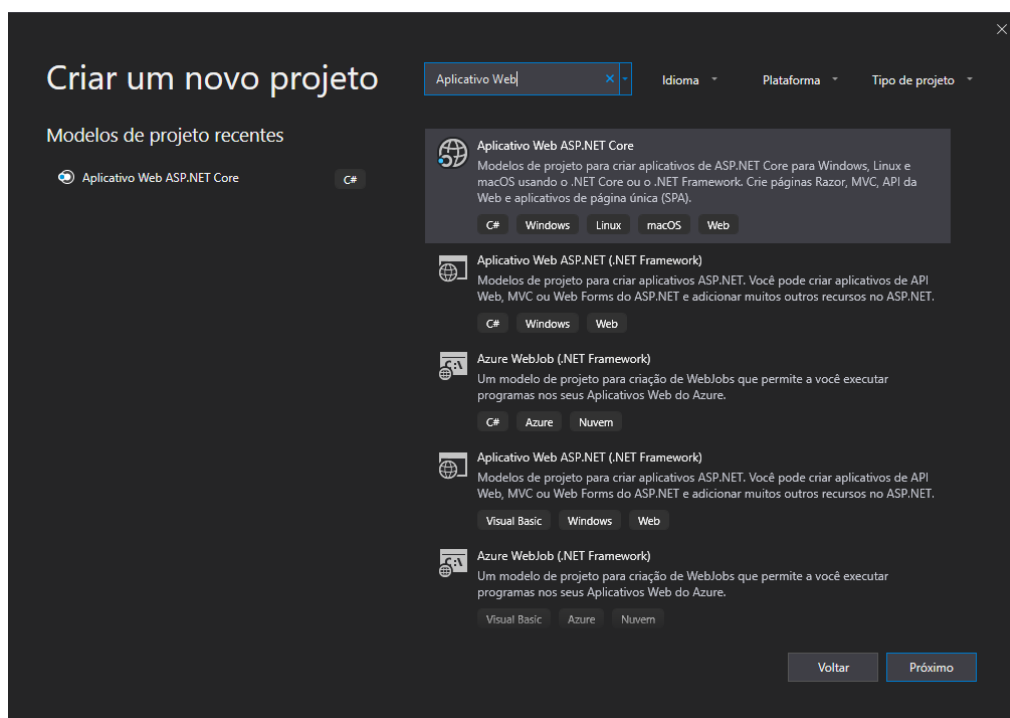
Figura 14 – Criação de novo projeto no Visual Code



Fonte: autor

5 O Visual Studio Code 2019 está disponível em: <https://visualstudio.microsoft.com/pt-br/vs/>

Figura 15 – Criação de Aplicativo Web ASP.NET Core



Fonte: autor

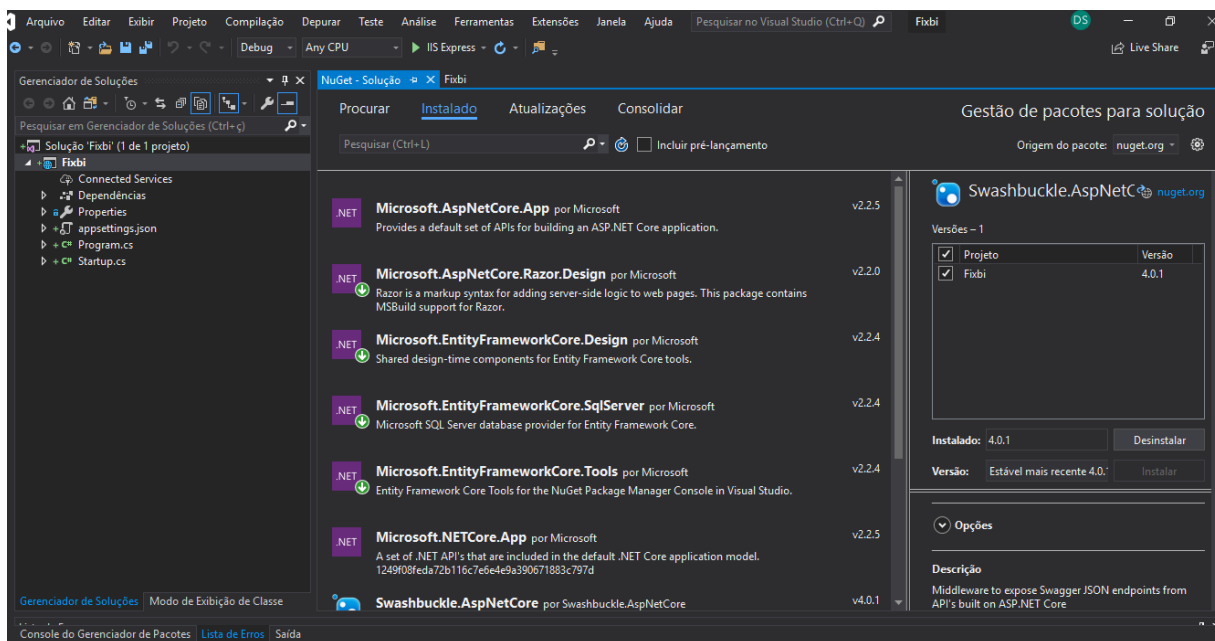
Após criar um Aplicativo Web ASP.NET Core do tipo vazio (Figura 15), é preciso escolher o nome do projeto e o local onde o aplicativo será gravado. A API do Fixbi está localizada na pasta Back-end do repositório GitHub.

No Visual Studio, é preciso baixar alguns pacotes NuGet (gerenciador de bibliotecas para a plataforma .NET) para que a API funcione corretamente `Microsoft.EntityFrameworkCore.Design`; `Microsoft.EntityFrameworkCore.SqlServer`; `Microsoft.EntityFrameworkCore.Tools`; `Swashbuckle.AspNetCore`; `Swashbuckle.AspNetCore.Swagger` (Figura 16).

As ferramentas do Visual Code Studio podem ser acessadas pela barra de busca com a IDE também disponível em português. Como pequenos “tijolos”, os pacotes trazem rotinas de códigos que serão usados para o projeto, dispensando do desenvolvedor back-end a necessidade de codificar linha por linha todas as funções.

O NuGet é um gerenciador de pacotes gratuito e de código aberto projetado para a plataforma de desenvolvimento Microsoft (Não confundir com o lanche chamado nugget). Desde a sua introdução em 2010, o NuGet evoluiu para um ecossistema de ferramentas e serviços distribuídos como extensão do Visual Studio (MICROSOFT, 2018).

Figura 16 – Pacotes NuGut instalados no projeto Fixbi no Visual Code



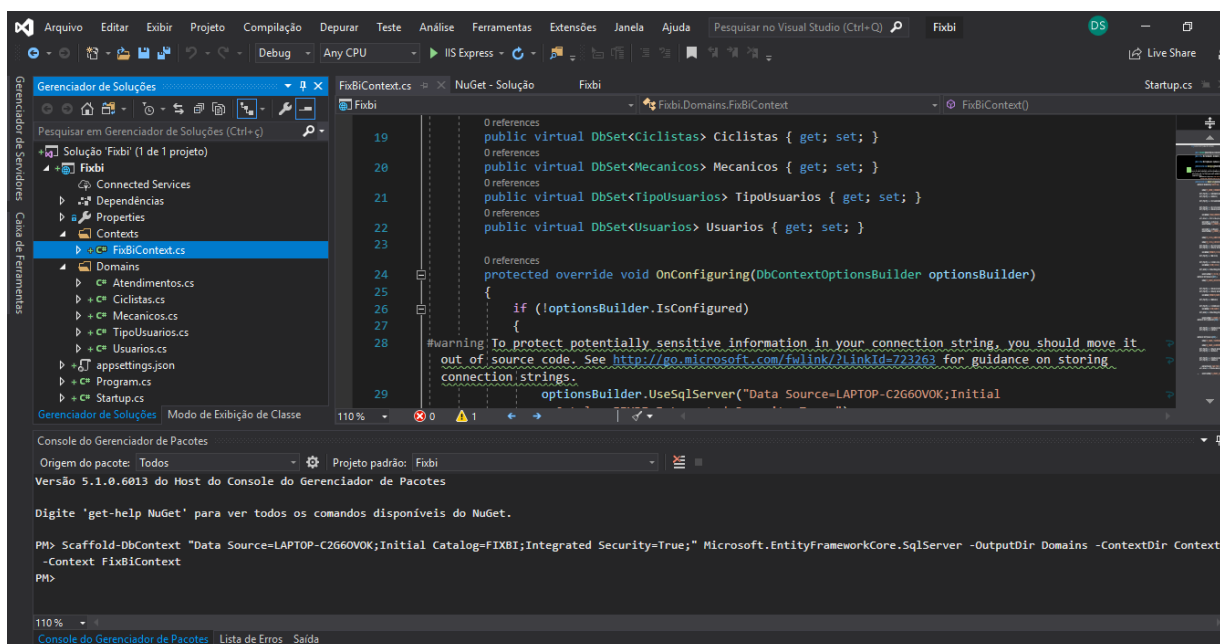
Fonte: autor

No Console de Gerenciador de Pacotes, o passo seguinte é aplicar o comando no terminal: `Scaffold-DbContext "Data Source=LAPTOP-C2G6OVOK;Initial Catalog=FIXBI;Integrated Security=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Domains -ContextDir Contexts -Context FixBiContext`.

A instrução tem o comando `Scaffold-DbContext` (string de conexão do banco de dados), `"Data Source=LAPTOP-C2G6OVOK;Initial Catalog=FIXBI;Integrated Security=True;"` (nome do provedor usado entre aspas), `-OutputDir` (nome da pasta que fica as classes), `-ContextDir` - (nome da pasta que fica o contexto), `-Context` - (nome do arquivo de contexto) (Figura 17).

Segundo a documentação da Microsoft, o EF (Entity Framework) Core é uma versão leve, extensível, de software livre e multiplataforma da tecnologia de acesso a dados do Entity Framework. Serve como um ORM (Object-Relational Mapping), em português Mapeador de Objeto Relacional, permitindo acesso a um banco de dados eliminando a necessidade de parte do código de acesso aos dados. Com o EF Core, o acesso a dados é executado usando um modelo. “Um modelo é composto por classes de entidade e um objeto de contexto que representa uma sessão com o banco de dados, o que permite consultar e salvar dados”. A abordagem DataBase First parte de um banco de dados existente em contraposição a abordagem Code First (MICROSOFT, 2016).

Figura 17 – Pastas Domains, com classes das tabelas, e Contexts, com FixbiContext



Fonte: autor

Na arquitetura MVC, o projeto tem agora o Model, sendo preciso codificar o View e o Controller. O primeiro passo é iniciar a configuração da classe Startup para utilizar ferramentas como Tokens, Json, Swagger, Hosting (Figura 18).

Figura 18 – Código em C# da classe Startup

```
using System;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.IdentityModel.Tokens;
using Newtonsoft.Json;
using Swashbuckle.AspNetCore.Swagger;

namespace Fixbi
{
    public class Startup
    {
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddMvc()
                .AddJsonOptions(options =>
                {

```

```

        options.SerializerSettings.NullValueHandling =
NullValueHandling.Ignore;
    })
    .SetCompatibilityVersion(Microsoft.AspNetCore.Mvc.CompatibilityVersion.
Version_2_1);
    services.AddSwaggerGen(c =>
    {
        c.SwaggerDoc("v1", new Info { Title = "Fixbi", Version = "v1" });
    });
    services.AddAuthentication(options =>
    {
        options.DefaultAuthenticateScheme = "JwtBearer";
        options.DefaultChallengeScheme = "JwtBearer";
    }).AddJwtBearer("JwtBearer", options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            IssuerSigningKey = new
SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes("Fixbi.Api")),
            ClockSkew = TimeSpan.FromMinutes(30),
            ValidIssuer = "Fixbi",
            ValidAudience = "Fixbi"
        };
    });
    services.AddCors(options =>
    {
        options.AddPolicy("CorsPolicy",
            builder => builder.AllowAnyOrigin()
                .AllowAnyMethod()
                .AllowAnyHeader()
                .AllowCredentials());
    });
}
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
}

```

```

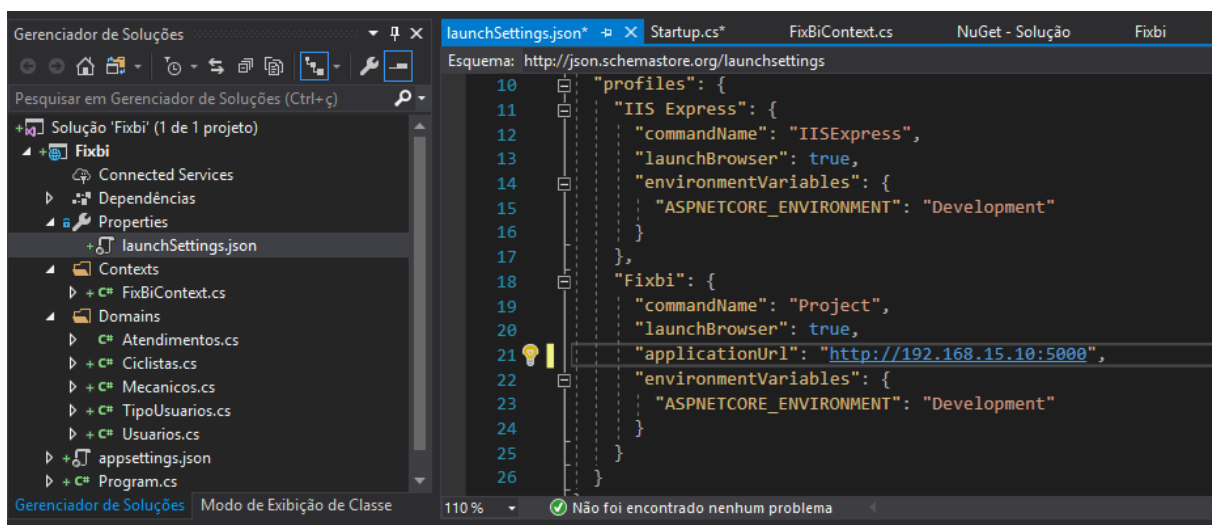
app.UseAuthentication();
app.UseSwagger();
app.UseSwaggerUI(c =>
{
    c.SwaggerEndpoint("/swagger/v1/swagger.json", "Fixbi");
});
app.UseCors("CorsPolicy");
app.UseMvc();
}
}
}

```

Fonte: autor

No arquivo `launchSettings.json`, dentro da pasta `Properties`, trocar o `applicationUrl` para a IP do computador (Figura 19).

Figura 19 – Troca de IP no `Properties` para ser URL do projeto durante o desenvolvimento



Fonte: autor

Com as configurações feitas, o projeto inicia pela montagem da página de login tendo como roteiro a escrita de classes dos tipos `ViewModel`, `Interfaces`, `Repositories` e `Controllers`. Dentro da pasta, `ViewModel`, criar a classe `LoginViewModel` para a entrada do e-mail e da senha do usuário (Figura 20).

Figura 20 – Código da classe LoginViewModel

```

using System.ComponentModel.DataAnnotations;

namespace Fixbi.ViewModels
{
    public class LoginViewModel
    {
        [Required(ErrorMessage = "Informe seu e-mail")]
        public string Email { get; set; }
        [Required(ErrorMessage = "Informe seu senha")]
        [StringLength(10, MinimumLength = 3, ErrorMessage = "Até 10 caracteres")]
        public string Senha { get; set; }
    }
}

```

Fonte: autor

Na pasta Interfaces, é preciso iniciar pela classe IUserarioRepository.cs. (Figura 21).

Figura 21 – Código da classe IUserarioRepository para busca por e-mail e senha

```

using Fixbi.Domains;
using Fixbi.ViewModels;

namespace Fixbi.Interfaces
{
    interface IUserarioRepository
    {
        Usuarios BuscarPorEmailESenha(LoginViewModel login);
    }
}

```

Fonte: autor

Na pasta Repositories, é preciso codificar o repositório dentro da classe UsuarioRepositories.cs (Figura 22).

Figura 22 – Código da classe UsuarioRepositories para acesso ao BD

```

using Fixbi.Interfaces;
using Fixbi.ViewModels;
using Fixbi.Domains;
using System.Linq;

```

```

using Microsoft.EntityFrameworkCore;
namespace Fixbi.Repositories
{
    public class UsuarioRepository : IUsuarioRepository
    {
        public Usuarios BuscarPorEmailESenha(LoginViewModel login)
        {
            using (FixBiContext ctx = new FixBiContext())
            {
                return ctx.Usuarios.Include(x =>
x.IdTipoUsuarioNavigation).FirstOrDefault(x => x.EmailUsuario == login.Email &&
x.SenhaUsuario == login.Senha);
            }
        }
    }
}

```

Fonte: autor

Com o Model e o View do login prontos, é preciso codificar o controlador na pasta Controller na classe LoginController.cs (Figura 23).

Figura 23 – Código da classe LoginController.cs para acessar o login

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.IdentityModel.Tokens;
using Fixbi.Domains;
using Fixbi.Interfaces;
using Fixbi.Repositories;
using System;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using Fixbi.ViewModels;
using System.Text;

namespace Fixbi.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    [Produces("application/json")]
    public class LoginController : ControllerBase
    {
        private IUsuarioRepository UsuarioRepository { get; set; }

        public LoginController()

```

```

{
    UsuarioRepository = new UsuarioRepository();
}

[HttpPost]
public IActionResult Logar(LoginViewModel login)
{
    try
    {
        Usuarios usuarioProcurado =
UsuarioRepository.BuscarPorEmailESenha(login);

        if (usuarioProcurado == null)
        {
            return NotFound(new
            {
                mensagem = "Usuário ou senha encontrados"
            });
        }
        var claims = new[]
        {
            new Claim(JwtRegisteredClaimNames.Email,
usuarioProcurado.EmailUsuario),
            new Claim(JwtRegisteredClaimNames.Jti,
usuarioProcurado.IdUsuario.ToString()),
            new Claim(ClaimTypes.Role,
usuarioProcurado.IdTipoUsuarioNavigation.TipoUsuario),
            new Claim("Role",
usuarioProcurado.IdTipoUsuarioNavigation.TipoUsuario)
        };

        var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes("Aplicativo.Fixbi"));
        var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
        var token = new JwtSecurityToken(
            issuer: "Fixbi",
            audience: "Fixbi",
            claims: claims,
            expires: DateTime.Now.AddMinutes(30),
            signingCredentials: creds
        );
    }
}

```

```

        return Ok(new
        {
            token = new JwtSecurityTokenHandler().WriteToken(token)
        });
    }
    catch (Exception ex)
    {
        return BadRequest(new
        {
            mensagem = "Erro: " + ex
        });
    }
}
}
}

```

Fonte: autor

Os mesmos passos são realizados para as funções Cadastrar Atendimentos e Listar Atendimentos. Na pasta Interfaces, codificar a classe `IAtenimentoRepository.cs` (Figura 24).

Figura 24 – Código da classe `IAtenimentoRepository`

```

using Fixbi.Domains;
using System.Collections.Generic;

namespace Fixbi.Interfaces
{
    public interface IAtenimentoRepository
    {
        void cadastrarAtendimento(Atendimentos atendimento);

        List<Atendimentos> listarAtendimentos();
    }
}

```

Fonte: autor

Na pasta Repositories, codificar a classe `AtendimentoRepository.cs` (Figura 25).

Figura 25 – Código da classe `AtendimentoRepositories` para acesso ao BD

```

using Fixbi.Domains;

```

```

using Fixbi.Interfaces;
using System.Collections.Generic;
using System.Linq;

namespace Fixbi.Repositories
{
    public class AtendimentoRepository : IAtendimentoRepository
    {
        public void cadastrarAtendimento(Atendimentos atendimento)
        {
            using (FixBiContext ctx = new FixBiContext())
            {
                ctx.Atendimentos.Add(atendimento);
                ctx.SaveChanges();
            }
        }

        public List<Atendimentos> todosAtendimentos()
        {
            using (FixBiContext ctx = new FixBiContext())
            {
                return ctx.Atendimentos.ToList();
            }
        }
    }
}

```

Fonte: autor

Com o Model e o View do login prontos, é preciso codificar o controlador na pasta Controller na classe AtendimentosController.cs (Figura 26).

Figura 26 – Código da classe AtendimentosController

```

using System;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Fixbi.Domains;
using Fixbi.Interfaces;
using Fixbi.Repositories;
namespace Fixbi.Controllers
{
    [Produces("application/json")]
    [Route("api/[controller]")]

```



```

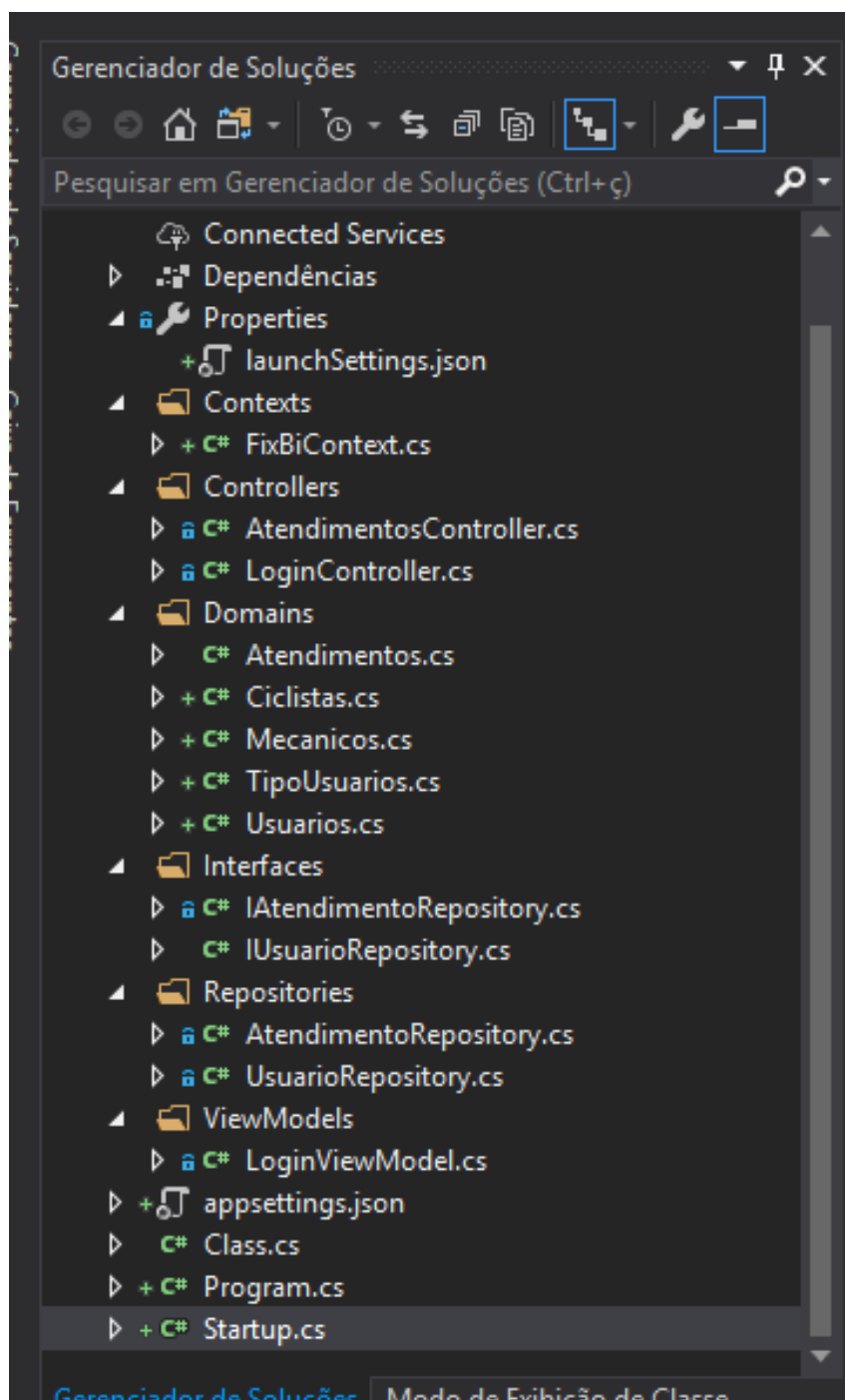
[ApiController]
public class AtendimentosController : ControllerBase
{
    private IAtendimentoRepository AtendimentoRepository { get; set; }
    public AtendimentosController()
    {
        AtendimentoRepository = new AtendimentoRepository();
    }
    [HttpPost("cadastrar")]
    [Authorize(Roles = "Ciclista, Mecanico")]
    public IActionResult cadastrarAtendimento(Atendimentos atendimento)
    {
        try
        {
            AtendimentoRepository.cadastrarAtendimento(atendimento);
            return Ok();
        }
        catch (Exception ex)
        {
            return BadRequest(new { mensagem = ex.Message });
        }
    }
    [HttpGet("listar")]
    [Authorize(Roles = "Ciclista, Mecanico")]
    public IActionResult listarAtendimentos()
    {
        try
        {
            return Ok(AtendimentoRepository.todosAtendimentos());
        }
        catch (Exception ex)
        {
            return BadRequest(new { mensagem = ex.Message });
        }
    }
} } }

```

Fonte: autor

Com isso, o projeto tem o conjunto de classes prontos (Figura 27).

Figura 27 – Conjunto de classes da API



Fonte: autor

4.4 Teste (Postman)

O teste de API é um tipo de software que envolve testar interfaces diretamente como parte da integração para determinar se atendem às expectativas de funcionalidade, confiabilidade, desempenho e segurança. Como as APIs não têm uma GUI (Graphical User Interface), o teste da API é realizado na camada de mensagens. O teste da API é considerado crítico para a automação de testes porque os ciclos de testes são difíceis de manter com os curtos ciclos do desenvolvimento dentro da metodologia ágil. Como a API é um conjunto de regras que contêm métodos de comunicação, o teste de verificação de sua funcionalidade envolve a coleta de informações e a verificação se atende às expectativas com retorno da resposta correta (REICHERT, 2015).

O aplicativo de teste escolhido para este projeto é o Postman⁶ (em português significa “carteiro”), que permite ao desenvolvedor que configure cabeçalhos e cookies que a API espera e verifica uma resposta. A startup indiana, que foi lançada em 2012 pelo desenvolvedor Abhinav Asthana, conta hoje com 7 milhões de usuários e 300 mil empresas como clientes. Em julho de 2019, a empresa levantou R\$ 50 milhões para investimento na expansão de seus negócios. Um teste no Postman é fundamentalmente um código JavaScript, que é executado depois que uma solicitação é enviada e uma resposta é recebida do servidor. O Postman tem versões pagas, mas pode ser utilizado de forma gratuita tanto na página web quanto em aplicativo nativo instalado no computador (SINGH, 2019).

O Postman tem utilização intuitiva. O teste é feito através de uma solicitação HTTP, que contém o método, a URL, os cabeçalhos, o corpo da solicitação, o script de pré-solicitação e os testes. São quatro os métodos de solicitação mais frequentes POST (criar ou atualizar dados), PUT (para atualização de dados), GET (recuperar e buscar dados) e DELETE (apagar dados). Além da URL definida para a API, os cabeçalhos de solicitação contêm dois valores chaves do aplicativo: tipo de conteúdo, em geral, JSON (JavaScript Object Notation), e o Token de autorização do tipo JWT, que é usado para identificar o solicitante. O corpo da solicitação contém os dados relacionados com o banco de dados, além dos scripts de pré-solicitação, que são uma parte do código que é executada antes do envio da solicitação. A resposta HTTP contém corpo, cookies, cabeçalhos, testes, código de status e tempo de resposta da API. Existem uma lista padrão do código de status da

6 O Postman está disponível em: <https://www.getpostman.com/>

aos usuários que acessem rotas, serviços e recursos das páginas web com esse token. O JWT também pode ser usado para a transmissão de informações.

Compacto e leve, o JWT é composto por três partes separadas: cabeçalho, carga útil e assinatura. A saída é três strings Base64-URL que podem ser facilmente transmitidos em ambientes HTML e HTTP, sendo mais compactos quando comparados a padrões baseados em XML, como o SAML (JSON WEB TOKENS, 2018).

Neste projeto, o token captado da API em comunicação com o banco de dados e trazido pelo Postman é testado no site JSON Web Tokens⁷. Na resposta é possível enxergar o cabeçalho com o tipo de algoritmo (HS256) e o tipo (JWT), o corpo com e-mail do usuário, o id de cadastro, o tipo de usuário e a regra do tipo de usuário (Figura 29).

Figura 29 –Tela do JWT Web Tokens com teste de login da API do Fixbi

The screenshot shows the JWT Web Tokens website interface. At the top, there's a navigation bar with links like 'Debugger', 'Libraries', 'Introduction', 'Ask', and 'Get a T-shirt!'. The main content area is divided into two panels: 'Encoded' and 'Decoded'.

Encoded Panel: It has a text area containing a long string of Base64-URL encoded characters. Below the text area, there's a blue checkmark icon and the text 'Signature Verified'.

Decoded Panel: It shows the decoded payload in a structured format. The 'HEADER' section displays the algorithm 'HS256' and the type 'JWT'. The 'PAYLOAD: DATA' section shows a JSON object with fields like 'email', 'jti', 'role', 'Role', 'exp', 'iss', and 'aud'. The 'VERIFY SIGNATURE' section shows the HMACSHA256 function being used to verify the token's signature.

At the bottom right of the 'Decoded' panel, there's a blue button labeled 'SHARE JWT'.

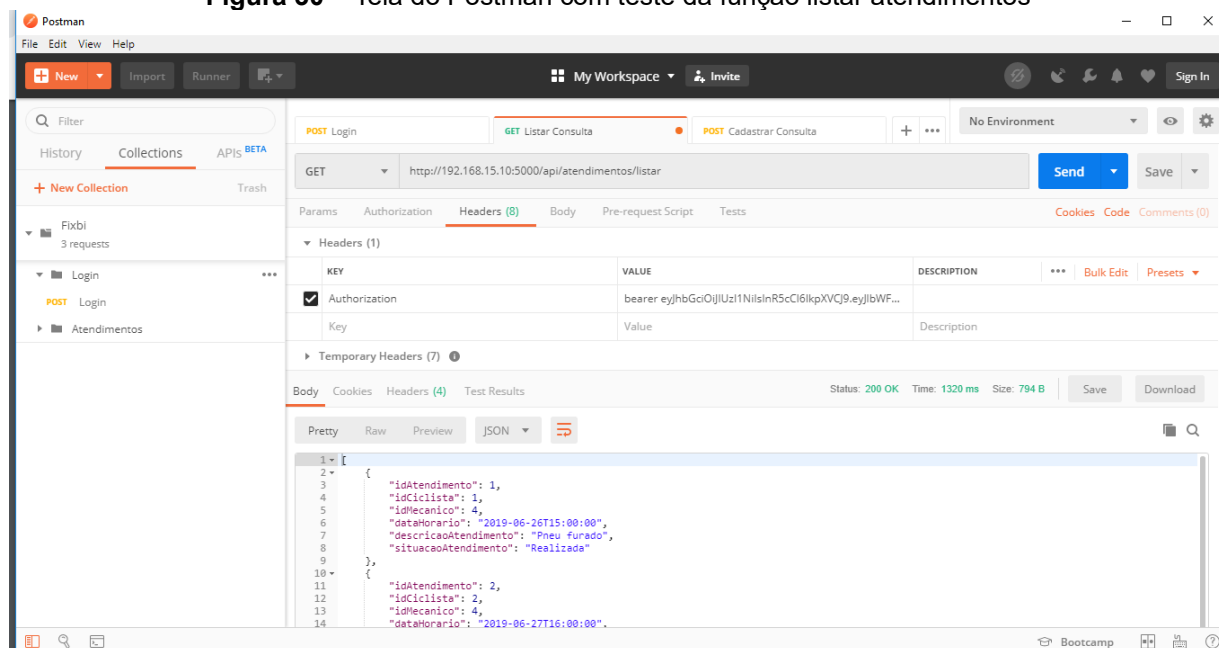
Fonte: autor

Com a autorização do usuário implementada pela API, é possível testar no Postman as funções listar e cadastrar atendimentos. É preciso colocar na aba Headers, no campo Key a opção "Authorization" e no campo Value a expressão "Bearer" seguida do token gerado no login. Os testes feitos para este projeto podem

⁷ O site está disponível em: <https://jwt.io/>

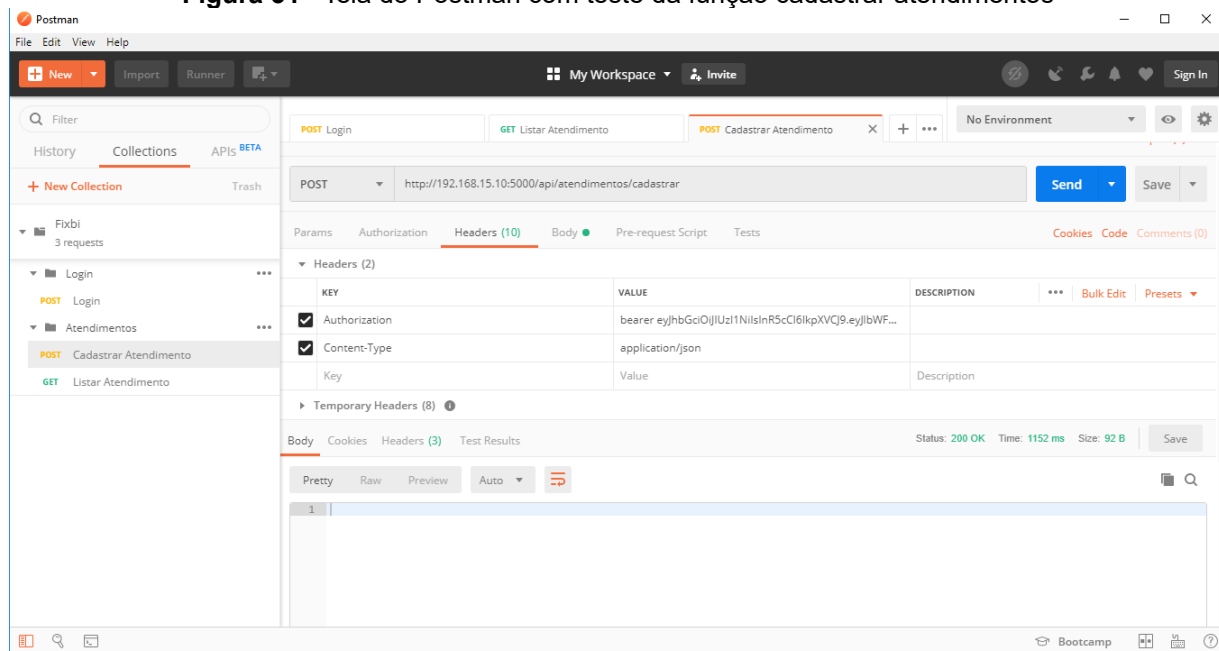
ser acessados no arquivo `Fixbi.postman_collection`, dentro da pasta Back-end do repositório GitHub. (Figuras 30 e 31).

Figura 30 – Tela do Postman com teste da função listar atendimentos



Fonte: autor

Figura 31 – Tela do Postman com teste da função cadastrar atendimentos



Fonte: autor

Com o back-end do aplicativo completo, o projeto necessita do front-end que será dividido em duas partes: o desenvolvimento do layout e da aplicação mobile.

4.5 Design (UI/UX)

Doutor em interação humano-computador pela Universidade Técnica da Dinamarca, o consultor em usabilidade web Jakob Nielsen deu seu nome em 1998 à Lei de Nielsen, na qual previa que a largura de banda da internet aumenta em 50% ao ano (10% a menos que a Lei de Moore para velocidade do computador). A teoria foi provada entre 1983 e 2018, conforme mostra no site da sua consultoria, mostrando as conexões que obteve começando em 300 bps (bytes por segundo) em 1984 a linha de 300 Mbps (megabytes por segundo) em 2018. A lei tem implicação no design por apontar como o usuário mais comum costuma se conectar à web. O especialista entende que a largura de banda é um dos dois elementos mais importantes da computação atualmente, junto com a qualidade da tela. Para ele, o web design precisa atender às massas fazendo que apenas raramente um site pode ser bem-sucedido se for destinado aos 10% mais avançados de usuários. Nielsen também definiu os cinco componentes de qualidade de usabilidade: aprendizagem, eficiência, memorabilidade e erros (NIELSEN, 1998).

Em 2012, Jakob Nielsen e Raluca Budiu publicam livro sobre o design mobile. A pergunta dos especialistas é como criamos uma experiência de usuário satisfatória quando limitada a um pequeno dispositivo. A publicação relata como fazer testes de usabilidade para celulares (como o uso de uma câmera atrás do usuário), como a usabilidade varia de acordo com a categoria dos dispositivos móveis e porque sites completos não funcionam para uso móvel, entre outros pontos (NIELSEN; BUDI, 2012).

O design embarca amplo fluxo de assuntos não se limitando ao gráfico. Para a web, existem dois tipos de designs UX (experiência do usuário) e UI (interface do usuário). Apesar de serem integrantes uns dos outros, os papéis dos tipos são bem diferentes com processos distintos. Segundo a consultoria Tripin Studio, saber quem são os clientes-alvo e como tornar a experiência deles com o produto mais gratificante possível é responsabilidade da equipe de design da UX. Em relação ao UI, criar uma interface do usuário é um desafio, especialmente porque precisa ser intuitiva. “Algo que parece ótimo, mas é difícil de usar, é um exemplo de ótima interface do usuário e baixa experiência de usuário. Enquanto algo muito usável que parece terrível é exemplar de grande UX e pobre IU”, resume a consultoria (TRIPIN STUDIO, 2018).

Como ferramenta de UX e UI, o projeto elabora uma projeção vetorial das telas login do usuário, cadastrar e listar atendimentos para o mobile. Essa projeção traz a estrutura das páginas.

Por décadas programas como Adobe Photoshop e Illustrator dominaram o mercado de editoração eletrônica, sendo imediatamente utilizados para a produção de páginas web. A questão é que esses aplicativos foram projetados para publicações impressa. Não são ferramentas ideais para projetos que envolvem hipertexto com a necessidade de prototipagem dessas relações. Em 2015, a Adobe lança em 2015 o XD⁸, primeiro projeto "começo-ao-fim" design UX para apps, segundo reportagem da revista Wired (RHODES, 2015).

O XD (Figura 32) é uma ferramenta de edição gratuita baseada em vetores projetada para criar protótipos de experiência do usuário para aplicativos da Web e mobile. De fácil manipulação, a ferramenta permite o design de vetores e wireframing de sites e a criação de protótipos de cliques interativos simples. O aplicativo ainda dispõe de bibliotecas referências das quais se podem copiar os objetos. O resultado desse projeto é baseado em uma prototipação em busca de um design mais simples possível (Figura 33).

Figura 32 – Tela do programa XD com wireframe das telas do aplicativo

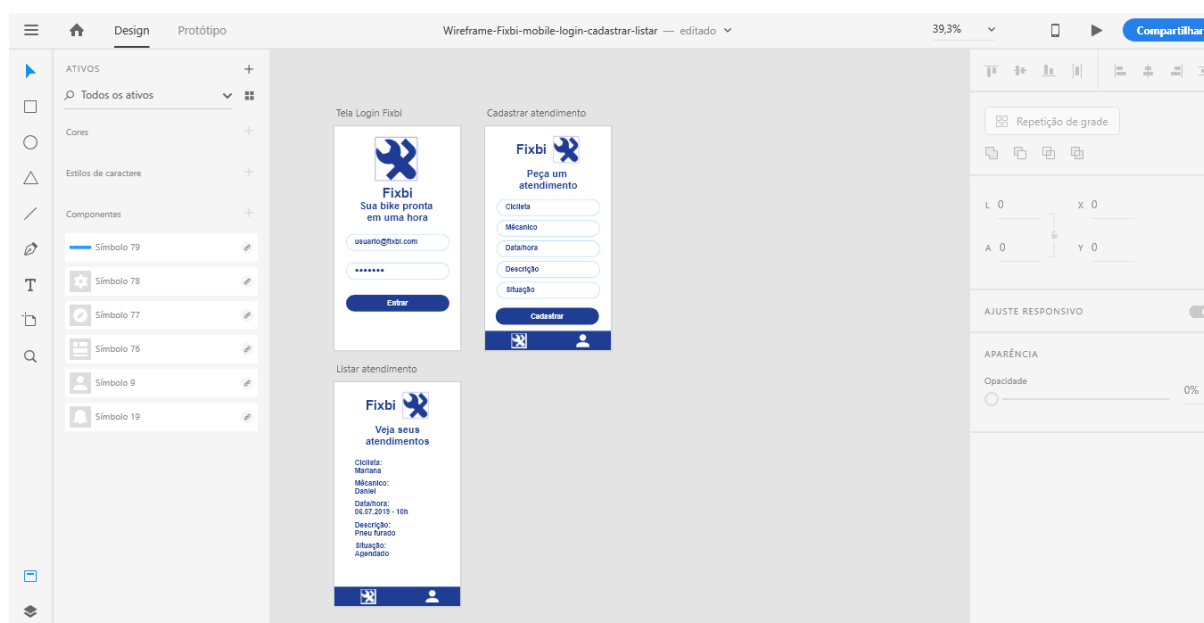


Figura: autor

8 O Adobe XD está disponível em: <https://creativecloud.adobe.com/apps/download/xd>

Figura 33 – Projeção das telas Login, Cadastrar e Listar do aplicativo Fixbi



A imagem apresenta três telas de um aplicativo chamado Fixbi, projetadas em um layout de wireframe. A primeira tela (Login) contém o logo da Fixbi, o slogan 'Sua bike pronta em uma hora', campos de entrada para e-mail e senha, e um botão 'Entrar'. A segunda tela (Cadastro) também possui o logo e o slogan 'Peça um atendimento', seguido por campos de entrada para 'Ciclista', 'Mecânico', 'Data/hora', 'Descrição' e 'Situação', um botão 'Cadastrar', e uma barra de navegação inferior com ícones de bicicleta e usuário. A terceira tela (Lista) mostra o logo e o slogan 'Veja seus atendimentos', uma lista de detalhes de um atendimento (Ciclista: Mariana, Mecânico: Daniel, Data/hora: 06.07.2019 - 10h, Descrição: Pneu furado, Situação: Agendado), e a mesma barra de navegação inferior.

Fixbi
Sua bike pronta em uma hora

usuario@fixbi.com

.....

Entrar

Fixbi
Peça um atendimento

Ciclista

Mecânico

Data/hora

Descrição

Situação

Cadastrar

Fixbi
Veja seus atendimentos

Ciclista:
Mariana

Mecânico:
Daniel

Data/hora:
06.07.2019 - 10h

Descrição:
Pneu furado

Situação:
Agendado

4.6 Mobile (React Native)

“Aprenda uma vez, escreva em qualquer lugar” é o lema do Facebook para o React Native, framework (arcabouço conceitual) Javascript para escrita real renderizada de aplicativos mobile que rodam nativamente em iOS e Android. “Queremos desenvolver um conjunto consistente de ferramentas e tecnologias que nos permita construir aplicações usando o mesmo conjunto de princípios através de qualquer plataforma”, afirma engenheiro Tom Occhino, na apresentação do React Native na React.js Conf 2015. (OCCHINO, 2015).

O lançamento desse produto pelo Facebook ocorre depois de o CEO desta rede social, Mark Zuckerberg, ter afirmado em conferência para desenvolvedores em 2012 que o seu maior erro foi ter apostado demais no HTML5 (HyperText Markup Language). O React já havia começado a ser implantado na feed do Facebook desde 2011. A frase de Zuckerberg resume as dificuldades para o desenvolvimento de aplicações mobile. Muito dos problemas se devem à demora para se fazer testes pois as ferramentas de desenvolvimento eram focadas para páginas web. A ideia inicial do React Native é usar a melhor capacidade dos conjuntos de desenvolvimento de software (SDKs) nativos com a velocidade do Javascript. O JS é uma linguagem de programação leve, interpretada ou just-in-time, com funções de primeira classe criada em 1995. Uma execução externa interpreta o código escrito em JS e publica na execução principal com atualização da interface do usuário chamando o código nativo. O React Javascript revoluciona a forma como os frameworks JS são construídos, introduzindo o conceito de DOM virtual permitindo que a programação seja declarativa (JOBNINJA, 2018).

Como apresenta documentação oficial do React, o DOM (Document Object Model) virtual é um conceito de programação em que uma representação ideal, ou “virtual”, de uma interface do usuário é mantida na memória e sincronizada com o DOM “real” por uma biblioteca como o ReactDOM. Esse processo é chamado de reconciliação. Uma das principais características é o DOM virtual (“modelo de objeto de documento virtual”). Com ele, o React cria um cache de estrutura de dados na memória, calcula as diferenças resultantes e atualiza o DOM exibido do navegador com eficiência (FACEBOOK, 2109).

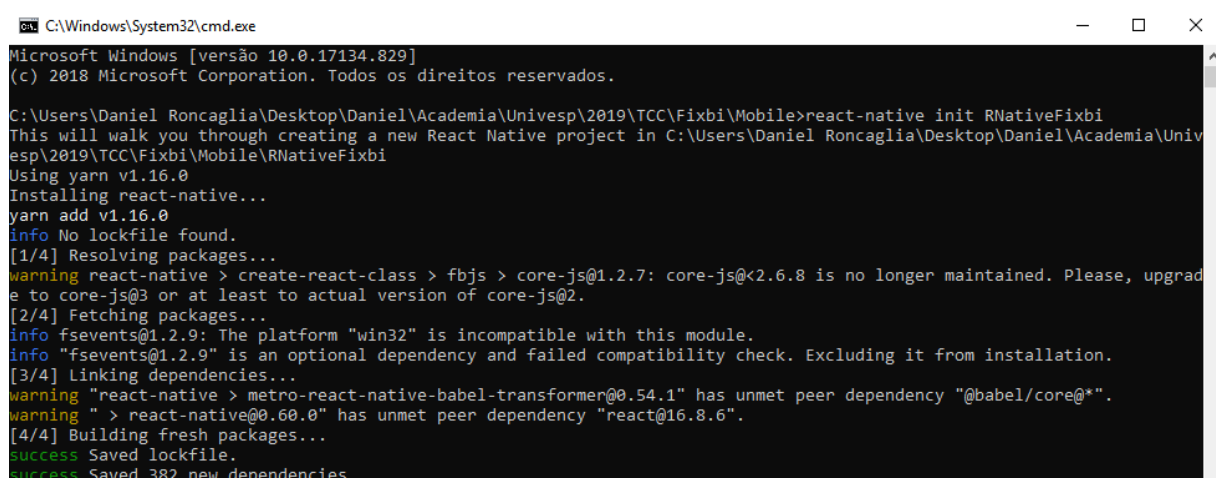
Uma das decisões no momento de desenvolver um aplicativo mobile é escolher entre nativo, híbrido e web. Como explica texto da empresa de desenvolvimento Gist, a resposta depende das prioridades do cliente como prazo,

objetivo, orçamento. Os três tipos têm as suas vantagens e desvantagens. Os aplicativos nativos e híbridos são aqueles que são baixados nas lojas dos sistemas (Apple Store e Google Play). A diferença é que os nativos são feitos diretamente para os sistemas: aplicativos Android são feitos em Java e aplicativos iPhone são feitos em Objective-C. As vantagens são serem mais rápidos e permitirem uma renderização melhor da imagem, além de poderem acessar ferramentas do aparelho como câmera, microfone, lista de contatos, geolocalização. O contra é o fato de ter um custo mais elevado para o desenvolvimento, além de ser preciso elaborar um aplicativo para cada plataforma. O híbrido também tem acesso às ferramentas do dispositivo, o que amplia o leque de recursos que o aplicativo poderá utilizar. A vantagem rodas nas duas plataformas. Os híbridos, como este projeto, são um pouco mais lentos para rodar e a performance visual é mais limitada. Os aplicativos web são como páginas web com interatividade similar ao do mobile rodando nos navegadores como Safari e Chrome sendo escritos em HTML, CSS ou JS (GIGVY, 2019).

O primeiro passo agora é a criação do projeto do tipo React Native. É preciso configurar o sistema operacional do desktop através de uma série de procedimentos do React Native⁹.

Com o sistema configurado, deve-se escolher uma pasta para alocar o novo projeto. No repositório, está na pasta Mobile. Ao se digitar “cmd”, o Windows abre o “Prompt de Comando”. Então, para criar o projeto executar o comando “react-native init [Nome do projeto]” de iniciação (Figuras 34 e 35).

Figura 34 – Aplicação do comando “react-native init RNativeFixbi” para a criação do projeto



```

C:\Windows\System32\cmd.exe
Microsoft Windows [versão 10.0.17134.829]
(c) 2018 Microsoft Corporation. Todos os direitos reservados.

C:\Users\Daniel Roncaglia\Desktop\Daniel\Academia\Univesp\2019\TCC\Fixbi\Mobile>react-native init RNativeFixbi
This will walk you through creating a new React Native project in C:\Users\Daniel Roncaglia\Desktop\Daniel\Academia\Univesp\2019\TCC\Fixbi\Mobile\RNativeFixbi
Using yarn v1.16.0
Installing react-native...
yarn add v1.16.0
info No lockfile found.
[1/4] Resolving packages...
warning react-native > create-react-class > fbjs > core-js@1.2.7: core-js@<2.6.8 is no longer maintained. Please, upgrade to core-js@3 or at least to actual version of core-js@2.
[2/4] Fetching packages...
info fsevents@1.2.9: The platform "win32" is incompatible with this module.
info "fsevents@1.2.9" is an optional dependency and failed compatibility check. Excluding it from installation.
[3/4] Linking dependencies...
warning "react-native > metro-react-native-babel-transformer@0.54.1" has unmet peer dependency "@babel/core@*".
warning " > react-native@0.60.0" has unmet peer dependency "react@16.8.6".
[4/4] Building fresh packages...
success Saved lockfile.
success Saved 382 new dependencies.
  
```

Fonte: autor

9 O React Native está disponível em: <https://facebook.github.io/react-native/docs/getting-started.html>

Figura 35 – Resultado da aplicação do comando de criação de projeto React Native

```

C:\Windows\System32\cmd.exe
esquery@1.0.1
external-editor@3.0.3
file-entry-cache@5.0.1
flat-cache@2.0.1
flatted@2.0.1
glob-parent@3.1.0
ignore@4.0.6
import-fresh@3.1.0
inquirer@6.4.1
is-extglob@2.1.1
is-glob@4.0.1
jest@24.8.0
json-stable-stringify-without-jsonify@1.0.1
levn@0.3.0
lodash.unescape@4.0.1
metro-react-native-babel-preset@0.55.0
optionator@0.8.2
parent-module@1.0.1
path-dirname@1.0.2
progress@2.0.3
react-refresh@0.2.0
rxjs@6.5.2
table@5.4.1
text-table@0.2.0
tslib@1.10.0
tsutils@3.14.0
write@1.0.3
Done in 31.71s.

Run instructions for iOS:
  • cd C:\Users\Daniel Roncaglia\Desktop\Daniel\Academia\Univesp\2019\TCC\Fixbi\Mobile\RNativeFixbi && react-native run-ios
  - or -
  • Open ios\RNativeFixbi.xcodeproj in Xcode
  • Hit the Run button

Run instructions for Android:
  • Have an Android emulator running (quickest way to get started), or a device connected.
  • cd C:\Users\Daniel Roncaglia\Desktop\Daniel\Academia\Univesp\2019\TCC\Fixbi\Mobile\RNativeFixbi && react-native run-android

C:\Users\Daniel Roncaglia\Desktop\Daniel\Academia\Univesp\2019\TCC\Fixbi\Mobile>

```

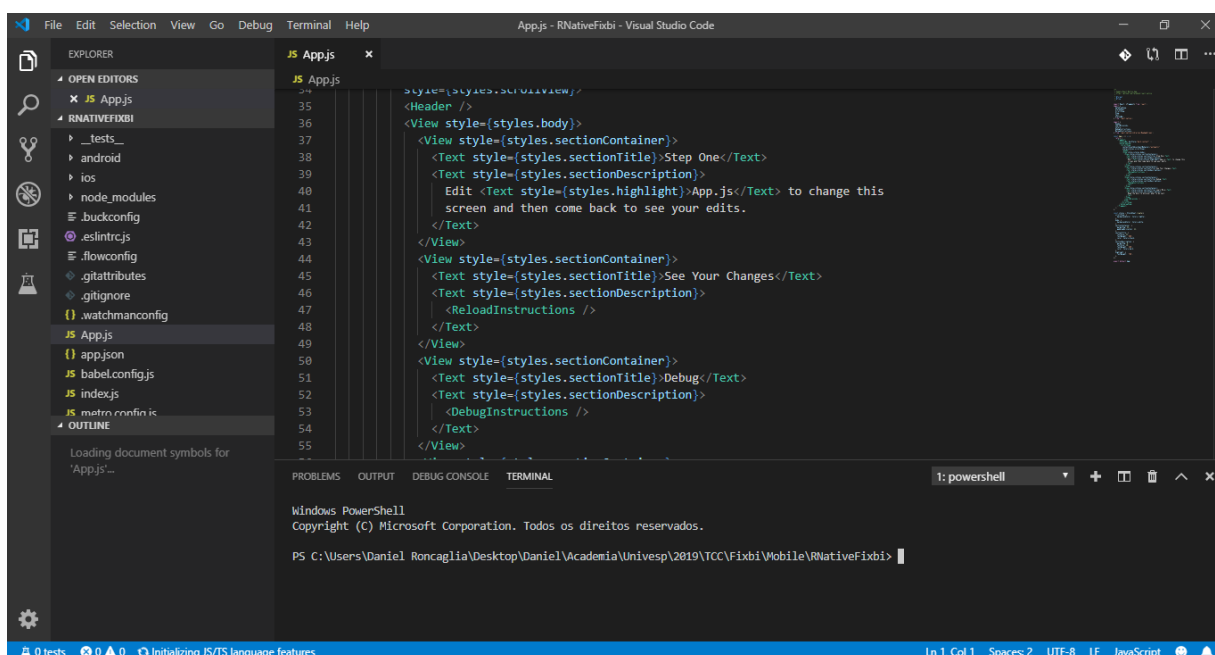
Fonte: autor

Para a edição do código React Native, será utilizado o Visual Studio Code¹⁰. Conforme a Microsoft, trata-se de editor de código-fonte leve, mas poderoso, que é executado na área de trabalho do desktop e está disponível para Windows, macOS e Linux. O Visual Studio Code tem com suporte embutido para JavaScript, TypeScript e Node.js e possui um ecossistema de extensões para outras linguagens (como C ++, C#, Java, Python, PHP, Go) e tempos de execução (como .NET e Unity) (MICROSOFT, 2019).

Com o Visual Studio Code instalado, pode-se abrir a pasta do projeto criado pelo comando React Native (Figura 36).

10 O Visual Studio Code está disponível em: <https://code.visualstudio.com/>

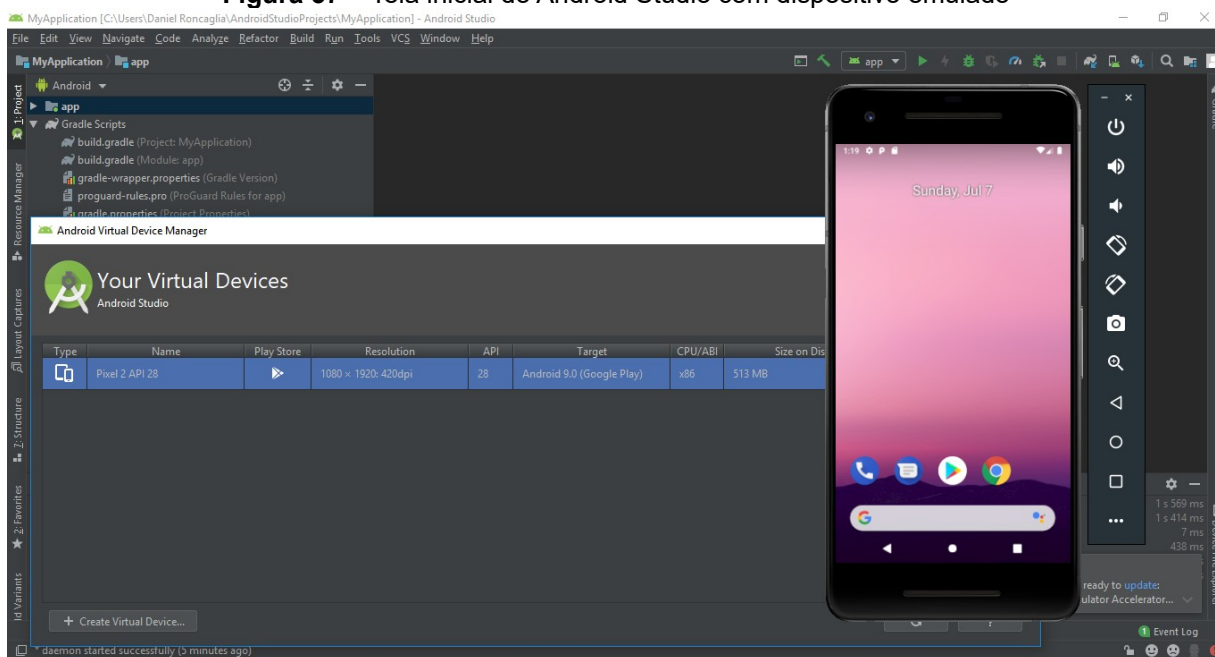
Figura 36 – Tela do projeto React Native no Visual Studio Code



Fonte: autor

Para emular o funcionamento do aplicativo será utilizado o Android Studio¹¹, é o ambiente de desenvolvimento integrado (IDE) oficial para o sistema operacional Android do Google lançado em 2013. Abrir um dispositivo (Figura 37).

Figura 37 – Tela inicial do Android Studio com dispositivo emulado

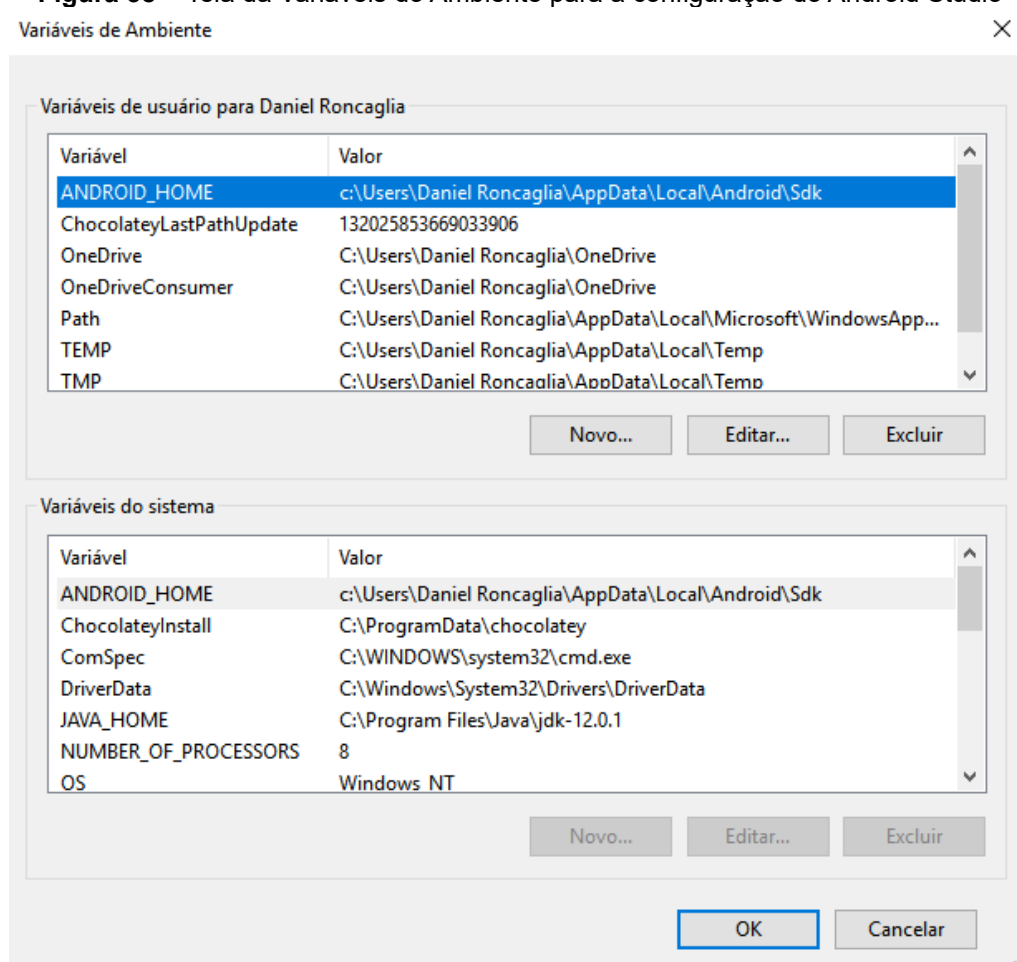


Fonte: autor

11 O Android Studio está disponível em: <https://developer.android.com/studio>

Na conferência anual com desenvolvedores que a Google organiza em Mountain View, na Califórnia, desde 2008 entre os meses de maio e junho, a Google I/O, o Android Studio foi lançado com a “necessidade dos desenvolvedores do Android em mente”, conforme afirma a equipe de engenheiros do produto. “Sabemos que você precisa de um sistema de construção que se adapte aos requisitos do projeto, mas que se estenda ao seu ambiente de desenvolvimento maior. O Android Studio usa um novo sistema de criação baseado em Gradle que fornece flexibilidade, compilações personalizadas, resolução de dependência e muito mais”, afirmam os especialistas (DUCROHET; NORBYE; CHOU, 2013).

Figura 38 – Tela da Variáveis do Ambiente para a configuração do Android Studio

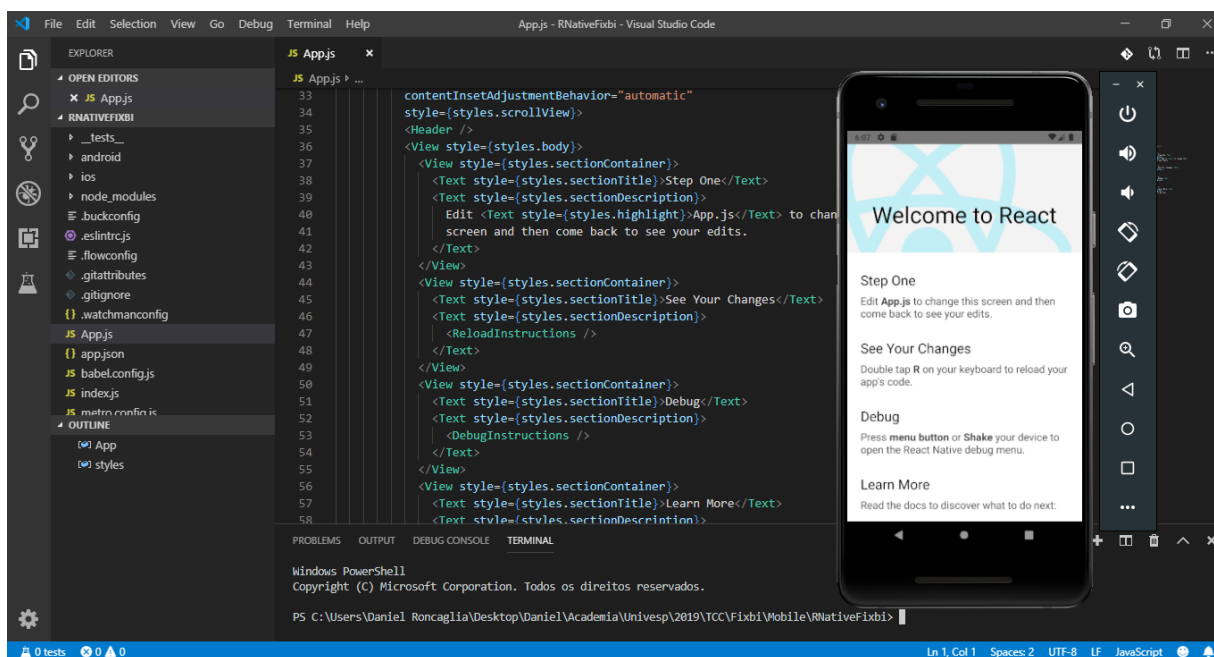


Fonte: autor

A instalação do Android Studio exige configurar as variáveis do ambiente do sistema operacional para o emulador e para a interpretação java (Figura 38).

Com o dispositivo do emulador ligado, aplicar o comando “react-native run-android” no terminal do Visual Studio Code. Após o processo de execução, o aplicativo é emulado pelo dispositivo do Android Studio (Figura 39).

Figura 39 – Tela com código do React Native no Visual Code e com dispositivo do emulador Android



Fonte: autor

O passo seguinte é a configuração do aplicativo. Antes é preciso instalar componente do React Native que será usado através do comando “npm install --save react-navigation”. A montagem do aplicativo começa com a criação de uma pasta “src”, nela são criadas três pastas “assets”, “pages” e “services” e uma classe “index.js”. Na pasta “pages”, criar as classes “signin.js”, “main.js” e “profile.js”.

Na pasta “Index.js” original do projeto, alterar a rota de navegação para a pasta “src” (Figura 40).

Figura 40 – Código na classe “Index.js” para direcionar navegação para pastar “src”

```
import {AppRegistry} from 'react-native';
import Navigator from './src';
import {name as appName} from './app.json';
AppRegistry.registerComponent(appName, () => Navigator);
```

Fonte: autor

Na pasta “services” dentro da pasta “src”, é colocada a classe api.js, que liga a aplicação do React Native com a API projetada através da IP configurada como URL do projeto (Figura 41).

Figura 41 – Código na classe “api.js” para fazer a ligação com o API

```
import axios from "axios";

const api = axios.create({
  baseURL: "http://192.168.15.10:5000/api"
});

export default api;
```

Fonte: autor

As três páginas do projeto signin.js (Figura 42), main.js e profile.js ficam no arquivo “pages” e as imagens na pasta “assets”.

Figura 42 – Código na classe “signin.js” para a tela login

```
import React, { Component } from "react";

import {
  StyleSheet,
  View,
  Text,
  Image,
  TextInput,
  TouchableOpacity,
} from "react-native";

import { AsyncStorage } from 'react-native';
import api from "../services/api";

class SignIn extends Component {
  static navigationOptions = {
    header: null };

  constructor(props) {
    super(props);
    this.state = { email: "", senha: "" }; }

  _realizarLogin = async () => {

    const resposta = await api.post("/login", {
      email: this.state.email,
      senha: this.state.senha
    });
```



```

const token = resposta.data.token;
await AsyncStorage.setItem("userToken", token);
this.props.navigation.navigate("MainNavigator");
};

render() {
  return (

    <View style={styles.telaLogin}>

      <Image
        source={require('../assets/Logo-FixBi.png')}
        style={styles.imagemLogin}
      />
      <Text style={styles.textoLogin}>{"Fixbi"}</Text>
      <Text style={styles.textoLogin2}>{"Sua bike pronta em uma hora"}</Text>

      <TextInput
        style={styles.inputLogin}
        placeholder="usuario@fixbi.com"
        onChangeText={email => this.setState({ email })}
      />

      <TextInput
        style={styles.inputLogin}
        placeholder="*****"
        secureTextEntry={true}
        onChangeText={senha => this.setState({ senha })}
      />

      <TouchableOpacity style={styles.botaoLogin}
        onPress={this._realizarLogin}>
        <Text style={styles.botaoLoginTexto}>Entrar</Text>
      </TouchableOpacity>

    </View>
  );
}
}

const styles = StyleSheet.create({

  telaLogin: {
    flex: 1,
    flexDirection: "column",
    alignItems: "center",
    justifyContent: "space-around",
  },

```

```

textoLogin: {
  fontSize: 45,
  fontFamily: "Arial",
  color: "#1E3D93",
  fontWeight: 'bold',
},

textoLogin2: {
  fontSize: 28,
  fontFamily: "Arial",
  color: "#1E3D93",
  fontWeight: 'bold',
},

imagemLogin: {
  height: 170,
  width: 170,
},

botaoLogin: {
  height: 50,
  elevation: 3,
  width: 300,
  borderWidth: 1,
  borderColor: "#1E3D93",
  backgroundColor: "#1E3D93",
  justifyContent: "center",
  alignItems: "center",
  borderRadius: 30
},

botaoLoginTexto: {
  fontSize: 28,
  fontFamily: "Arial",
  color: "white" },

inputLogin: {
  alignItems: "center",
  borderWidth: 2,
  borderColor: "#1E3D93",
  color: "#1E3D93",
  width: 300,
  height: 50,
  fontSize: 28,
  fontFamily: "Arial",
  borderRadius: 30 } }));

export default SignIn;

```

Fonte: autor

No emulador a página então entra em funcionamento (Figura 43).

Figura 43 – Tela da página Login no emulador Android Studio com o código da classe "signin.js"

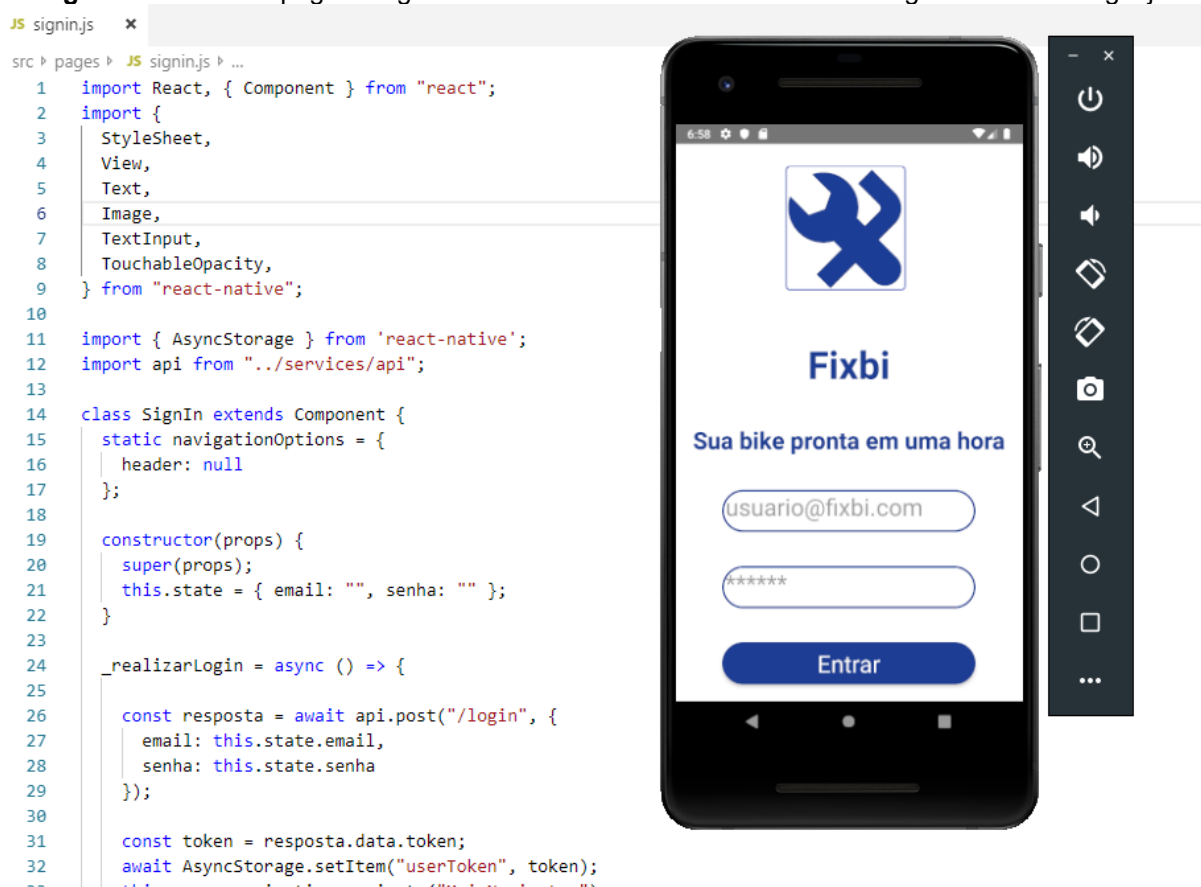


Figura: autor

Na classe "main.js", dentro da pasta "pages", o projeto colocará a listagem de atendimentos (Figura 44).

Figura 44 – Código na classe "main.js" para a tela listar atendimentos

```

import React, { Component } from "react";
import { Text, View, Image, StyleSheet, FlatList } from "react-native";
import axios from "axios";
import api from "../services/api";
import { AsyncStorage } from 'react-native';

class Main extends Component {

  static navigationOptions = {
    tabBarIcon: ({ tintColor }) => (
      <Image
        source={require("../assets/Logo-FixBi.png")}
        style={styles.iconeNavegacaoAtendimentos}

```

```

/>
)
};

constructor(props) {
  super(props);
  this.state = {
    Atendimentos: []
  };
}

componentDidMount() {
  this.carregarAtendimentos();
}

carregarAtendimentos = async () => {
  let token = await AsyncStorage.getItem('userToken');

  const resposta = await api.get("/atendimentos/listar", {
    headers: {
      "Content-Type": "application/json",
      'Authorization': "Bearer " + token
    }
  });
  const dadosDaApi = resposta.data;
  this.setState({ Atendimentos: dadosDaApi });
};

render() {
  return (

    <View style={styles.telaAtendimentos}>

      <Image
        source={require('../assets/Logo-FixBi.png')}
        style={styles.imagemAtendimentos}
      />

      <Text style={styles.textoAtendimentos}>{"Veja seus atendimentos"}</Text>

      <FlatList
        data={this.state.Atendimentos}
        keyExtractor={item => item.idAtendimento}
        renderItem={this.renderizaItem}
      />

    </View>
  );
}

```

```

renderizaItem = ({ item }) => (
  <View>

    <Text style={styles.textoListaAtendimentos}>Ciclista:</Text>
    <Text style={styles.textoListaAtendimentos}>{item.idCiclista}</Text>

    <Text style={styles.textoListaAtendimentos}>Data do atendimento:</Text>
    <Text style={styles.textoListaAtendimentos}>{item.dataHorario}</Text>

    <Text style={styles.textoListaAtendimentos}>Descrição:</Text>
    <Text style={styles.textoListaAtendimentos}>{item.descricaoAtendimento}</Text>

    <Text style={styles.textoListaAtendimentos}>Situação:</Text>
    <Text style={styles.textoListaAtendimentos}>{item.situacaoAtendimento}</Text>

    <Text style={styles.textoListaAtendimentos}>Nome do Mecânico:</Text>
    <Text style={styles.textoListaAtendimentos}>{item.idMecanico}</Text>

  </View>
);
}

const styles = StyleSheet.create({

  telaAtendimentos: {
    flex: 1,
    flexDirection: "column",
    alignItems: "center",
    justifyContent: "space-between",
  },

  imagemAtendimentos: {
    height: 120,
    width: 120,
  },

  textoAtendimentos: {
    fontSize: 40,
    fontFamily: "Arial",
    alignContent: "center",
    color: "#1E3D93",
  },

  textoListaAtendimentos: {
    fontSize: 22,
    fontFamily: "Arial",
    color: "#1E3D93",
    alignContent: "center",
  },

```

```
},
```

```
iconeNavegacaoAtendimentos: {
  width: 25,
  height: 25,
},
```

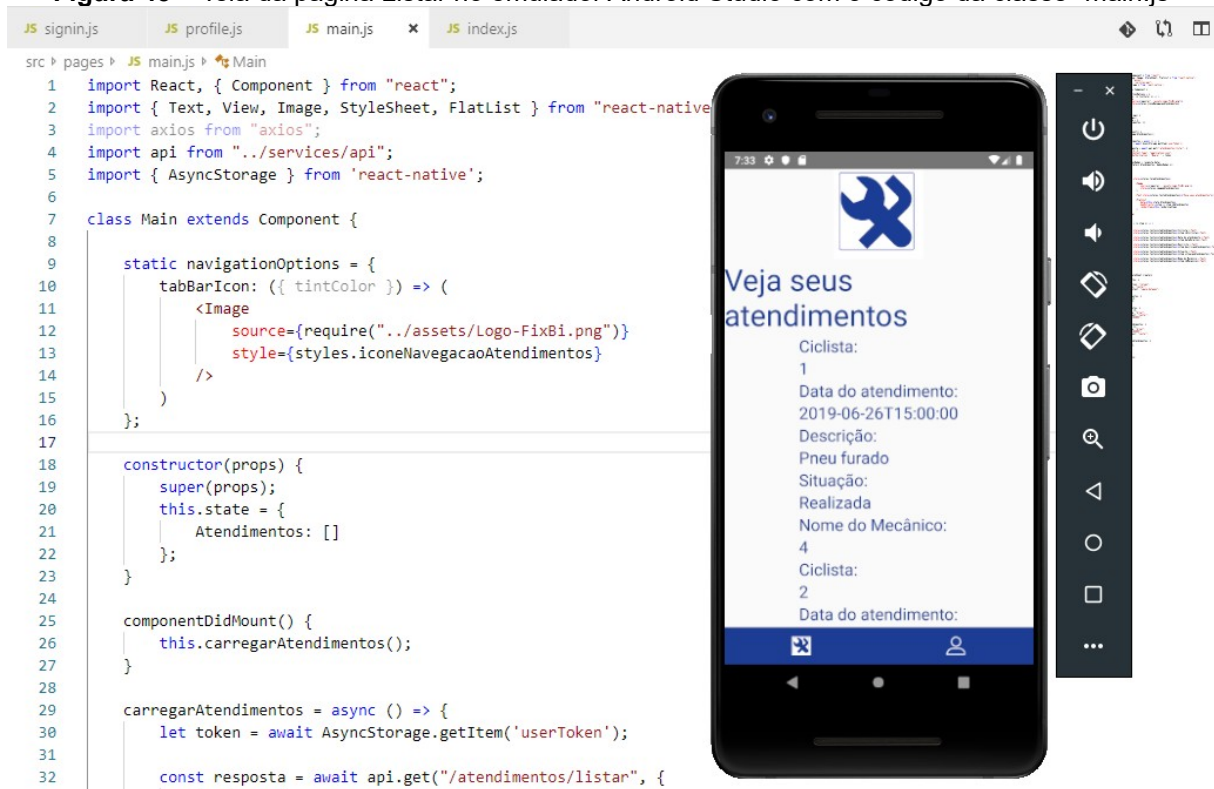
```
});
```

```
export default Main;
```

Figura: autor

No emulador a página Listar Atendimentos então entra em funcionamento (Figura 45).

Figura 45 – Tela da página Listar no emulador Android Studio com o código da classe "main.js"



Fonte: autor

O passo seguinte é fazer a página para cadastramento dos atendimentos, com o uso da classe "profile.js" (Figura 46).

Figura 46 – Código na classe "profile.js" para a tela cadastrar atendimentos

```
import React, { Component } from 'react';
```

```

import { Text, Image, StyleSheet, View, TouchableOpacity, AsyncStorage,
TextInput } from "react-native";
import api from '../services/api'
import jwt from 'jwt-decode';

class Profile extends Component {

  static navigationOptions = {
    tabBarIcon: ({ tintColor }) => (
      <Image
        source={require("../assets/profile.png")}
        style={styles.iconeNavegacaoPerfil}
      />
    )
  };

  constructor(props) {
    super(props);
    this.state = {
      idCiclista: ""
    , idMecanico: ""
    , dataHorario: ""
    , descricaoAtendimento: ""
    , situacaoAtendimento: ""
    }
  }

  cadastrarAtendimento = async () => {

    const resposta = await api.post("/atendimentos/cadastrar", {
      idCiclista: this.state.idCiclista,
      idMecanico: this.state.idMecanico,
      dataHorario: this.state.dataHorario,
      descricaoAtendimento: this.state.descricaoAtendimento,
      situacaoAtendimento: this.state.situacaoAtendimento,
    });
    console.warn(resposta);

  }

  buscardados = async () => {
    try {
      const value = await AsyncStorage.getItem("Aplicativo.Fixbi");
      if (value !== null) {
        this.setState({ IdUsuario: jwt(value).Id })
        this.setState({ token: value })
        console.warn(value);
      }
    } catch (error) {

```

```

}

}
componentDidMount() {
  this.buscardados();
}

render() {
  return (

    <View style={styles.telaCadastrar}>

      <Image
        source={require('../assets/Logo-FixBi.png')}
        style={styles.imagemCadastrar}
      />

      <Text style={styles.textoCadastrar}>{"Peça um atendimento"}</Text>

      <TextInput
        style={styles.inputCadastrar}
        placeholder="ID Ciclista"
        onChangeText={idCiclista => this.setState({ idCiclista })}
      />

      <TextInput
        style={styles.inputCadastrar}
        placeholder="ID Mecanico"
        onChangeText={idMecanico => this.setState({ idMecanico })}
      />

      <TextInput
        style={styles.inputCadastrar}
        placeholder="Data Horário"
        onChangeText={dataHorario => this.setState({ dataHorario })}
      />

      <TextInput
        style={styles.inputCadastrar}
        placeholder="Descrição"
        onChangeText={descricaoAtendimento => this.setState({ descricaoAtendimento })}
      />

      <TextInput
        style={styles.inputCadastrar}
        placeholder="Situação"
        onChangeText={situacaoAtendimento => this.setState({ situacaoAtendimento })}
      />
    </View>
  );
}

```



```

<TouchableOpacity
style={styles.botaoCadastrar}
onPress={this.cadastrarAtendimento}
>
<Text style={styles.botaoTextoCadastrar}>Cadastrar</Text>

</TouchableOpacity>

</View>
);
}
};

const styles = StyleSheet.create({

telaCadastrar: {
flex: 1,
flexDirection: "column",
alignItems: "center",
justifyContent: "space-around",
},

textoCadastrar: {
fontSize: 40,
fontFamily: "Arial",
color: "#1E3D93",
},

imagemCadastrar: {
height: 120,
width: 120,
},

botaoCadastrar: {
height: 40,
elevation: 3,
width: 300,
borderWidth: 1,
borderColor: "#1E3D93",
backgroundColor: "#1E3D93",
justifyContent: "center",
alignItems: "center",
borderRadius: 30
},

botaoTextoCadastrar: {
fontSize: 25,
fontFamily: "Arial",

```

```

color: "white"
},

inputCadastrar: {
  alignItems: "center",
  borderWidth: 2,
  borderColor: "#1E3D93",
  color: "#1E3D93",
  width: 300,
  height: 50,
  fontSize: 28,
  fontFamily: "Arial",
  borderRadius: 30
}

});

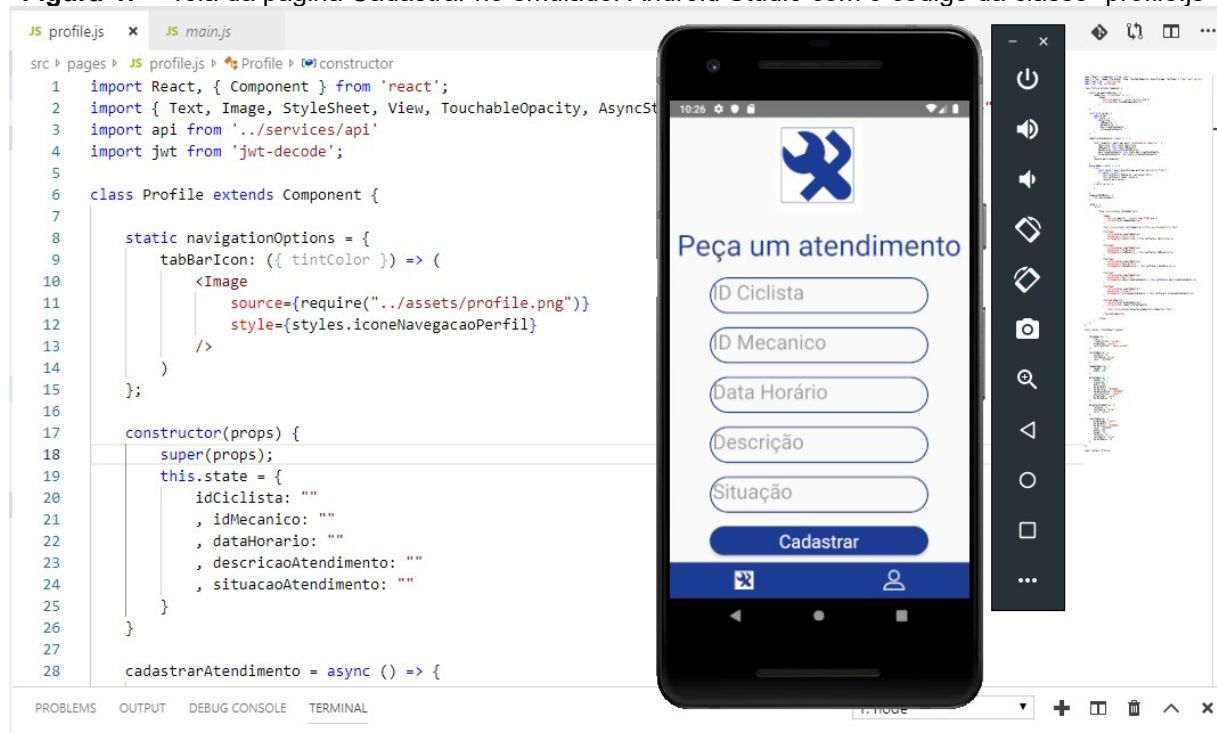
export default Profile;

```

Fonte: autor

No emulador a página de Cadastrar Atendimentos então entra em funcionamento (Figura 47). O conjunto de arquivos está no projeto RnativeFixbi na pasta Mobile do repositório do GitHub.

Figura 47 – Tela da página Cadastrar no emulador Android Studio com o código da classe "profile.js"



Fonte: autor

Ainda no projeto React Native, a classe “index.js” dentro da pasta “src” também está completa (Figura 48).

Figura 48 – Código na classe “index.js” para formar o caminha das páginas

```
import {createStackNavigator,
createBottomTabNavigator,
createAppContainer,
createSwitchNavigator}
from "react-navigation";

import Main from "../pages/main";
import Profile from "../pages/profile";
import SignIn from "../pages/signin";

const AuthStack = createStackNavigator({SignIn});

const MainNavigator = createBottomTabNavigator({
Main,
Profile
},
{
swipeEnabled: true,
tabBarOptions: {
showLabel: false,
showIcon: true,
inactiveBackgroundColor: "#1E3D93",
activeBackgroundColor: "#1E3D93",
activeTintColor: "#ffffff",
inactiveTintColor: "#ffffff",
style:{
height: 50
}
}
});

export default createAppContainer(
createSwitchNavigator(
{
MainNavigator,
AuthStack
},
{initialRouteName: "AuthStack"}
)
);
```

Figura: autor

5 CONSIDERAÇÕES FINAIS

5.1 Análise de resultados

O projeto para desenvolver o aplicativo mobile para prestadores de serviços, chamado de Fixbi, alcançou seus objetivos iniciais ao conseguir apresentar um aplicativo funcional e uma documentação completa sobre o assunto. O foco foi na simplicidade para que fosse entregue algo que realmente funciona.

Na parte de banco de dados, as três funções (logar usuários e cadastrar e listar atendimentos) e as tabelas não apresentaram problemas para o desenvolvimento. O interessante que é as diretrizes e os códigos usados poderão ser base para projetos futuros.

No mesmo sentido ocorreu com a API, no desenvolvimento back-end, com o funcionamento perfeito das funções logar usuários e cadastrar e listar atendimentos. Ao utilizar a arquitetura MVC, o código é compartimentado permitindo a sua reutilização em novos projetos.

Na aplicação mobile, houve um porém para cadastrar os atendimentos, erro que pode ser facilmente corrigido. Também é possível melhorar aspectos visuais que não puderam ser trabalhos neste projeto. Não foi implantada publicação do aplicativo em uma loja dos sistemas operacionais.

O texto deste documento também pode auxiliar pessoas que buscam apreender a desenvolver aplicativos mobile. Como exposto na motivação, o mercado brasileiro apresenta uma falta de profissionais qualificados para a área.

Mesmo sendo um trabalho relacionado com a universidade, o projeto como um todo foi feito praticamente de forma individual. Durante cinco anos, foram enfrentados diversos tipos de problemas dentro da Univesp. No período, ocorreram diversas mudanças na direção da universidade, no cronograma dos bimestres e das regras do curso, fatos que acrescentaram um alto grau de imprevisibilidade. Um dos indicadores desse problema é a alta taxa de evasão do curso de engenharia. Para os próximos estudantes, seria importante mudanças na organização e na metodologia do curso, do qual esta é a primeira turma.

Neste período do curso, não ocorreu qualquer contado direto dos membros da direção da universidade com o autor do projeto. Também há três anos não ocorre uma única orientação no desenvolvimento dos projetos integradores. Dias antes da entrega deste projeto, quando a pesquisa já estava adiantada, foram

indicados dois orientadores, que nada puderam mais do que apontar algumas incorreções referentes à formatação do relatório. Aspectos básicos também foram negligenciados. Em nenhuma vez a rede de internet do polo Pêra Marmelo, na zona norte da cidade de São Paulo, funcionou nas tentativas que o autor fez para se conectar. Os equipamentos e as ferramentas usados para o acompanhamento do curso e para a produção deste projeto foram custeados totalmente pelo próprio autor.

5.2 Projeções para o projeto

O projeto do aplicativo mobile deve continuar a ser desenvolvido com acréscimo de novas funções como cadastrar novos usuários ou geolocalização e melhorias no visual do produto. Nos próximos passos seria importante a publicação do aplicativo para que seja testado por usuários.

O aplicativo pode ter uso comercial para serviços do tipo offline chamados através de um dispositivo online. Diante do alto interesse do assunto, ainda é possível continuar a linha de pesquisa em programas de pós-graduação da área de Engenharia da Computação.

6 REFERÊNCIAS

ADA, Augusta. **Sketch of The Analytical Engine Invented by Charles Babbage**. Universidade de Genova: 1842. Disponível em: <http://www.fourmilab.ch/babbage/sketch.html> (Acesso em 20 jun.2019)

AGILE MANIFESTO. **Manifesto para desenvolvimento ágil de software**. Utah: 2001. Disponível em: <https://agilemanifesto.org/iso/ptbr/manifesto.html> (Acesso em: 20 mar. 2019)

AMERICAN DIALECT SOCIETY. **“App” 2010 Word of the Year, as voted by American Dialect Society**. Illinois: 2011. Disponível em: <https://www.americandialect.org/American-Dialect-Society-2010-Word-of-the-Year-PRESS-RELEASE.pdf> (Acesso em: 15 mai. 2019)

APP ANNIE. **Mobile hit new milestones in Q1 2019**. San Francisco: 2019. Disponível em: <https://www.appannie.com/en/insights/market-data/mobile-hit-new-milestones-in-q1-2019> (Acesso em: 20 abr. 2019)

BRASSCOM. **Relatório Setorial de Tecnologia da Informação e Comunicação**. Associação Brasileira das Empresas de Tecnologia da Informação e Comunicação. Brasília: 2019. Disponível em: <https://brasscom.org.br/relatorio-setorial-de-tic-2019> (Acesso em: 21 jun. 2019)

CERUZZI, Paul E. **A history of modern computing**. MIT Press. Cambridge: 1998. Disponível em: https://doc.lagout.org/science/0_Computer%20Science/0_Computer%20History/A%20History%20of%20Modern%20Computing.%202nd.pdf (Acesso em: 23 jun. 2019)

DUCROHET, Xavier; NORBYE, Tor; CHOU, Katherine. **Android Studio: An IDE built for Android**. Google. Mountain View: 2013. Disponível em: <https://android-developers.googleblog.com/2013/05/android-studio-ide-built-for-android.html> (Acesso em: 05 jul. 2019)

EISING, Perry. **What exactly IS an API?** Medium EUA: 2017. Disponível em: <https://medium.com/@perrysetgo/what-exactly-is-an-api-69f36968a41f> (Acesso em: 5 jun. 2019)

ELMASRI, Ramez; NAVATHE, Shamkant. **Fundamentals of database systems.** Person. Texas: 2007. Disponível em: https://books.google.com.br/books/about/Fundamentals_of_Database_Systems.html?id=owUZAQAAIAAJ&source=kp_book_description&redir_esc=y (Acesso em: 26 jun. 2019)

FACEBOOK. **Virtual Dom and Internals.** Palo Alto: 2109. Disponível em: <https://reactjs.org/docs/faq-internals.html> (Acesso em: 06 jul. 2019)

FIELDING, Roy T. **REST APIs must be hypertext driven.** Universidade da Califórnia. Irvine: 2008. Disponível em: <https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven> (Acesso em: 29 jun. 2019)

FLING, Brian. **Mobile Design and Development.** O'Reilly. Califórnia: 2009. Disponível em: <https://books.google.com.br/books?hl=en&lr=&id=LyMeulBTkH0C&oi=fnd&pg=PR5&dq=mobile+app+development+services&ots=fVZQv-rMwc&sig=Lb-fZ79vIbNPRZQr4hLU2KCyAQU#v=onepage&q&f=false> (Acesso em 18 jun. 2019)

GIGVY. **What's the Difference between Native vs. Web vs. Hybrid Apps?** EUA: 2019. Disponível em: <https://getgist.com/difference-between-native-vs-web-vs-hybrid-apps> (Acesso em: 06 jul. 2019)

GIT WIKI. **FAQ - General questions: what is Git?** EUA: 2010. Disponível em: https://git.wiki.kernel.org/index.php/Git_FAQ (Acesso em: 20 abr. 2019)

GITHUB. **GitHub is how people build software.** San Francisco: 2019. Disponível em: <https://github.com/about> (Acesso em: 24 jun. 2019)

GSMA Intelligence. **The Mobile Economy 2019**. Londres: 2019. Disponível em: <https://www.gsma.com/r/mobileeconomy> (Acesso em: 15 jun. 2019)

IDEO. **The Field Guide to Human-Centered Design**. EUA: 2009. Disponível em: <http://www.designkit.org> (Acesso em: 23 abr. 2018)

JOBNNINJA. **A short Story about React Native**. EUA: 2018. Disponível em: <https://jobnninja.com/blog/short-story-react-native> (Acesso em: 06 jul. 2019)

JOBS, Steve. **Steve Jobs Introducing The iPhone At MacWorld 2007**. Youtube. San Francisco: 2007. Disponível em: <https://www.youtube.com/watch?v=x7qPAY9JqE4> (Acesso em 15 abr. 2019)

JSON WEB TOKENS. **JWT.IO - JSON Web Tokens Introduction**. Bellevue: 2018. Disponível em: <https://jwt.io/introduction> (Acesso em: 20 abr. 2019)

KETecha, Khushboo. **API Testing using Postman**. Medium. EUA: 2018. Disponível em: <https://medium.com/aubergine-solutions/api-testing-using-postman-323670c89f6d> (Acesso em: 03 jul. 2019)

MICROSOFT. **Build Database Projects by Using SQL Server Management Studio**. Redmond: 2017. Disponível em: <https://docs.microsoft.com/en-us/sql/ssms/build-database-projects-by-using-sql-server-management-studio?view=sql-server-2017> (Acesso em: 12 mai. 2019)

MICROSOFT. **Entity Framework Core**. Redmond: 2016. Disponível em: <https://docs.microsoft.com/pt-br/ef/core> (Acesso em: 29 jun. 2019)

MICROSOFT. **NuGet 4.6 Release Notes**. Redmond: 2018. Disponível em: <https://docs.microsoft.com/en-us/nuget/release-notes/nuget-4.7-rtm> (Acesso em: 02 jul. 2019)

MICROSOFT. **Tour of CSharp.** Redmond: 2019. Disponível em: <https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp> (Acesso em: 28 jun. 2019)

MOBILE TIME; OPINION BOX. **Uso de Apps no Brasil em 2019.** São Paulo: 2019. Disponível em: <https://panoramamobiletime.com.br> (Acesso em: 22 jun. 2019)

NIELSEN, Jacob; BUDIU, Raluca. **Mobile Usability.** Nielsen Norman Group. Fremont: 2012. Disponível em: <https://www.nngroup.com/books/mobile-usability> (Acesso em: 04 jul. 2019)

NIELSEN, Jakob. **Nielsen's Law of Internet Bandwidth.** Nielsen Norman Group. Fremont: 1998. Disponível em: <https://www.nngroup.com/articles/law-of-bandwidth> (Acesso em: 04 jul. 2019)

OCCHINO, Tom. **React.js Conf 2015 Keynote - Introducing React Native.** Facebook. San Francisco: 2015. Disponível em: <https://www.youtube.com/watch?v=KVZ-P-ZI6W4> (Acesso em: 06 jul. 2019)

PLURALSIGHT. **What's the Difference Between the Front-End and Back-End?** EUA: 2015. Disponível em: <https://www.pluralsight.com/blog/film-games/whats-difference-front-end-back-end> (Acesso em: 20 jun. 2019)

REENSKAUG, Trygve; COPLIEN, James. **The DCI Architecture: A New Vision of Object-Oriented Programming.** Walnut Creek: 2009. Disponível em: https://www.artima.com/articles/dci_vision.html (Acesso em: 24 jun. 2019)

REICHERT, Amy. **Testing APIs protects applications and reputations.** Newton: 2015. Disponível em: <http://searchsoftwarequality.techtarget.com/tip/Testing-APIs-protects-applications-and-reputations> (Acesso em: 03 jul. 2019)

RHODES, Margaret. **Adobe's project comet is a start-to-finish UX design app.** Wired. Nova York: 2015. Disponível em: <https://www.wired.com/2015/10/adobes-new-project-comet-start-finish-website-design-app> (Acesso em: 18 mar. 2019)

SINGH, Manish. **Postman raises \$50 million to grow its API development platform.** TechCrunch. Sunnyvale: 2019. Disponível em: <https://techcrunch.com/2019/06/19/postman-series-b> (Acesso em: 03 jul. 2019)

SOUZA, João Nunes de. **Lógica para ciência da computação e áreas afins.** Elsevier. Rio de Janeiro: 2015. Disponível em: <https://www.amazon.com.br/L%C3%B3gica-para-Ci%C3%Aancia-Computa%C3%A7%C3%A3o-Souza-ebook/dp/B00YEW8J6> (Acesso em: 26 jun. 2019)

STRADA, Helena; GUERRA, Fernando Henrique. **Curso de Desenvolvimento de Sistemas.** Senai Informática. São Paulo: 2019. Disponível em: <https://github.com/senai-desenvolvimento> (Acesso em: 10 mai. 2019)

SUTHERLAND, Jeff. **Scrum: The Art of Doing Twice the Work in Half the Time.** Penguin Random House. Nova York: 2014. Disponível em: https://books.google.com.br/books/about/Scrum.html?id=ya7PBQAAQBAJ&source=kp_book_description&redir_esc=y (Acesso em: 10 fev. 2016)

TAN, Mingjie; YANG, Yongqi; YU, Ping Yu. **The influence of the maker movement on engineering and technology education.** Sichuan Open University. China: 2016. Disponível em: [http://www.wiete.com.au/journals/WTE&TE/Pages/Vol.14,%20No.1%20\(2016\)/14-Tan-M.pdf](http://www.wiete.com.au/journals/WTE&TE/Pages/Vol.14,%20No.1%20(2016)/14-Tan-M.pdf) (Acesso em: 20 abr. 2018)

TORVALDS, Linus. **Linus Torvalds visits Google to share his thoughts on git, the source control management system he created two years ago.** YouTube. EUA: 2007. Disponível em: <https://www.youtube.com/watch?v=4XpnKHJAok8&t=1m30s> (Acesso em: 05 abr. 2019)

TRIPIN STUDIO. **Go to the profile of Tripin Studio.** Hackernoon. EUA: 2018. Disponível em: <https://hackernoon.com/what-is-ui-ux-design-1f01e9dbbf02> (Acesso em: 04 jul. 2019)

W3SCHOOLS. **What is Full Stack?** EUA: 2019. Disponível em: https://www.w3schools.com/whatis/whatis_fullstack.asp (Acesso em: 25 jun. 2019)

WEISER, Mark. **The Computer for the 21st Century**. Cambridge: 1991. Disponível em: <https://www.ics.uci.edu/~corps/phaseii/Weiser-Computer21stCentury-SciAm.pdf> (Acesso em: 21 jun. 2019)

WORLD WIDE WEB CONSORTIUM. **Relationship to the World Wide Web and REST Architectures**. Cambridge: 2004. Disponível em: <https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest> (Acesso em: 29 jun. 2019)

WROBLEWSKI, Luke. **Mobile First**. A Book Apart. Estados Unidos: 2011. Disponível em: https://www.lukew.com/resources/mobile_first.asp (Acesso em: 20 fev. 2019)