

UNIVERSIDADE VEIGA DE ALMEIDA – UVA
BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

**ANÁLISE DO PROCESSO DE DESENVOLVIMENTO DE UMA
APLICAÇÃO MOBILE, IDEALIZADA ATRAVÉS DE UMA LEAN
INCEPTION, UTILIZANDO METODOLOGIAS LOW-CODE E
PROGRAMAÇÃO TRADICIONAL**

CAROLINE OLIVEIRA CAVALCANTI DE ALBUQUERQUE

RIO DE JANEIRO

2020

UNIVERSIDADE VEIGA DE ALMEIDA - UVA

CAROLINE OLIVEIRA CAVALCANTI DE ALBUQUERQUE

Monografia apresentada ao Curso de Engenharia da Computação da Universidade Veiga de Almeida, como requisito parcial para obtenção do título de Bacharel em Engenharia da Computação.

Orientador(a): André Lucio de Oliveira.

**ANÁLISE DO PROCESSO DE DESENVOLVIMENTO DE UMA
APLICAÇÃO MOBILE, IDEALIZADA ATRAVÉS DE UMA LEAN
INCEPTION, UTILIZANDO METODOLOGIAS LOW-CODE E
PROGRAMAÇÃO TRADICIONAL**

RIO DE JANEIRO

2020

UNIVERSIDADE VEIGA DE ALMEIDA - UVA
BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

CAROLINE OLIVEIRA CAVALCANTI DE ALBUQUERQUE

**ANÁLISE DO PROCESSO DE DESENVOLVIMENTO DE UMA
APLICAÇÃO MOBILE, IDEALIZADA ATRAVÉS DE UMA LEAN
INCEPTION, UTILIZANDO METODOLOGIAS LOW-CODE E
PROGRAMAÇÃO TRADICIONAL**

Monografia apresentada como
requisito parcial à conclusão do Curso de
Bacharel em Engenharia da Computação.

APROVADA EM:

CONCEITO: _____

BANCA EXAMINADORA:

PROF. MSc ANDRÉ LUCIO DE OLIVEIRA
ORIENTADOR

PROF. DSc MATHEUS BOUSQUET BANDINI

PROF. DSc ADRIANA APARICIO SICSU AYRES DO NASCIMENTO

Coordenação de Engenharia da Computação

Rio de Janeiro

*Dedico este trabalho a vida, aos meus pais,
minha família, aos meus amigos, a mim, a
todos que duvidam de sua capacidade e a
todos os que tem medo de programação.*

AGRADECIMENTOS

Antes de tudo, devo minha gratidão a inteligência suprema do universo, causa primária de todas as coisas, por me permitir estar na Terra, nesta era, desenvolvendo-me enquanto Ser Humano intelecto-moral. Aos espíritos amigos e protetores, minha gratidão por todo amor, cuidado e apoio que emanam em minha direção.

Gostaria também de agradecer aos meus pais, Ana Maria de Oliveira Cavalcanti e Angelo Cavalcanti de Albuquerque, por continuarem me recebendo, amando, cuidando e ensinando tão bem enquanto vivemos. Se sou uma pessoa melhor na atualidade, é também, porque vocês são vitoriosos na criação que me ofereceram. Também, desejo expressar meu reconhecimento por vocês terem sempre valorizado e priorizado meu bem-estar e evolução, assim como fizeram de tudo para me ajudar a concluir meus estudos.

Meus sinceros e honestos agradecimentos a minha família, avós e avôs, tias e tios, e primas e primos que sempre contribuíram moralmente, intelectualmente e financeiramente para que eu evoluísse em meus estudos, trabalhos e enquanto ser humano.

Quero agradecer aos meus sogros Claudia Defatima Bastos Albuquerque e Paulo Henrique da Silva por terem me apoiado com todo carinho e suporte que puderam dar para que eu conseguisse me dedicar a concluir meus estudos.

Aos meus amigos, gostaria de expressar minha gratidão por serem, estarem e permanecerem em minha vida, pois com certeza, vocês são força motriz nela. Em especial, pelo contexto deste trabalho gostaria de citar Pedro Roberto Barbosa Rocha por ter tido a paciência de me explicar Cálculo I e de ser uma grande inspiração acadêmica para mim, Ana Luiza Jorge Félix de Conceição por ter me proporcionado uma jornada cheia de risadas, leveza e aprendizados na Universidade, e Eduardo Albuquerque da Silva por ter tido a paciência de me ajudar a entender alguns aspectos da programação, pela bondade em doar seu tempo e se voluntariar para participar deste trabalho, e principalmente, por me lembrar que por mais que a desistência pareça ser o caminho mais fácil, é preciso e necessário continuar.

Gostaria de agradecer imensamente aos colegas de faculdade Camila Delaroli Gaspar, Bruno Gomide Ribeiro, Edgar Pinheiro da Câmara Júnior, Marcos Pedro Guerra Alves Miranda, Pedro Kats de Barros e Yan Barreto de Santana, que participaram diretamente deste trabalho doando tempo e disposição para ajudar no processo da *Lean Inception*.

Aos professores da minha vida, quero deixar minha gratidão por ajudarem a me construir e a me moldar, não só academicamente, mas como profissional e pessoalmente também. Em especial, e mesmo pelo cenário presente, desejo expressar minha gratidão e

estima pelo professor Edwillian Maia (in memoriam) por ter me incentivado o estudo pela conexão da informática com a sociedade, e por ter me inspirado a continuar acreditando e buscando provas a união da ciência com religião, ao professor André Lucio de Oliveira por ter me acolhido como orientanda num período tão conturbado, sempre buscando dar seu melhor para ajudar, por ser um exemplo pedagógico, e uma inspiração de ser humano. Também, quero agradecer aos professores Matheus Bousquet Bandini e Adriana Aparicio Sicsu Ayres do Nascimento por terem aceitado o convite de serem avaliadores da apresentação deste trabalho de conclusão de curso.

Meus sinceros agradecimentos ao meu gestor Guilherme Vaz Nunes e a equipe da liderança, que desde o momento em que fui contratada, sempre apoiaram o tempo que dedico aos meus estudos e sempre foram muito compreensivos com meus horários. Isso me possibilitou uma flexibilidade a qual foi extremamente necessária para a conclusão, não só do presente trabalho, mas como da faculdade também.

Finalmente, a todos que já citei anteriormente, gostaria que cada um sentisse minha gratidão, apreço e reciprocidade em seus corações. Vocês foram e continuam sendo fundamentais para minha formação pessoal, acadêmica e profissional. Parafraseando Shakespeare: “O meu amor eu guardo para os mais especiais. Não sigo todas as regras da sociedade e às vezes ajo por impulso. Erro, admito. Aprendo, ensino. Todos erram um dia: por descuido, inocência ou maldade. Conservar algo que faça eu recordar de ti seria o mesmo que admitir que eu pudesse esquecer-te” – William Shakespeare.

“As coisas não são ultrapassadas tão facilmente, são transformadas.

Nise da Silveira

RESUMO

Tendo em vista que as tecnologias de desenvolvimento de *software* têm evoluído rapidamente, novos desafios surgem e consequentemente novas soluções. Com esta evolução, muitas formas de programação foram criadas e outras otimizadas. No presente trabalho duas metodologias são apresentadas e aplicadas no desenvolvimento de uma funcionalidade de uma aplicação *mobile*: *low-code* e programação tradicional. Apesar de alguns conceitos do *low-code* estarem presente há décadas na área de tecnologia da informação, atualmente tem se expandido muito a partir de plataformas *web* e *mobile* que vêm sendo inseridas cada vez mais no mercado. Contudo, a metodologia de programação tradicional se mantém firme em relação ao seu posicionamento no mundo empresarial. O presente trabalho tem o objetivo de utilizar ambas as metodologias para o desenvolvimento de uma aplicação *mobile*. A partir do *framework* ágil *Lean Inception*, é possível idealizar um mínimo produto viável, e a partir daí iniciar o estudo de caso do desenvolvimento de uma das funcionalidades selecionada do mesmo, utilizando as metodologias citadas. Tal estudo de caso é analisado de acordo com o esforço e o tempo dedicados à programação, e os insumos são comparados a fim de valorizar a análise.

Palavras-Chave: Mobile, Low-Code, Programação Tradicional, Lean Inception, Mínimo Produto Viável.

ABSTRACT

Bearing in mind that software development technologies have been evolving rapidly, new challenges arise and consequently new solutions. With this evolution, many ways of programming have been created and others have been optimized. In the current thesis, two methodologies are presented and applied in the development of a mobile application functionality, low-code and traditional programming. Although the concept of low-code has been present for decades in information technology, it has currently expanded a lot from web and mobile platforms that are being increasingly inserted in the market. However, the traditional programming methodology remains firm in relation to its position in the business world. This thesis aims to use both methodologies for the development of a mobile application. From the agile Lean Inception framework, it is possible to idealize a minimum viable product, and from there start the case study of the development of one of the selected functionalities using the aforementioned methodologies. Such a case study is analyzed according to the effort and time dedicated to programming, and the inputs are compared in order to enhance the analysis.

Keywords: Mobile, Low-Code, Traditional Programming, Lean Inception, Minimum Viable Product.

LISTA DE ILUSTRAÇÕES

Figura 1 Custos de alterações como uma função do tempo em desenvolvimento	19
Figura 2 Cronograma Lean Inception Piloto	32
Figura 3 Visão do Produto - Lean Inception Piloto.....	33
Figura 4 É-Não é-Faz-Não faz - Lean Inception Piloto	34
Figura 5 Descrição Joana Estudante - Lean Inception Piloto.....	35
Figura 6 Descrição Yuri Feliz - Lean Inception Piloto	35
Figura 7 Descrição Margarida Sapucaí - Lean Inception Piloto	36
Figura 8 Mapa de Empatia Joana Estudante - Lean Inception Piloto.....	37
Figura 9 Mapa de Empatia Yuri Feliz - Lean Inception Piloto	38
Figura 10 Mapa de Empatia Margarida Sapucaí - Lean Inception Piloto	39
Figura 11 Esforço, Valor Para o Negócio e Valor Para UX 1 - Lean Inception Piloto.....	40
Figura 12 Esforço, Valor Para o Negócio e Valor Para UX 2 - Lean Inception Piloto.....	40
Figura 13 Gráfico do Semáforo - Lean Inception Piloto.....	41
Figura 14 Jornada 1 Joana Estudante - Lean Inception Piloto	42
Figura 15 Jornada 1 Yuri Feliz - Lean Inception Piloto.....	43
Figura 16 Jornada 1 Margarida Sapucaí - Lean Inception Piloto.....	44
Figura 17 Sequenciador de Funcionalidades - Lean Inception Piloto.....	45
Figura 18 Detalhando Amostra de Funcionalidades - Lean Inception Piloto	46
Figura 19 Dimensionamento de tarefas Outsystems - Lean Inception Piloto	47
Figura 20 Dimensionamento de tarefas Python, Ionic e Angular - Lean Inception Piloto.....	48
Figura 21 Entendendo custo e tempo e tirando a média Outsystems - Lean Inception Piloto	49
Figura 22 Entendendo custo e tempo e tirando a média Python, Ionic e Angular - Lean Inception Piloto	49
Figura 23 Canvas Outsystems - Lean Inception Piloto	50
Figura 24 Canvas Python, Ionic e Angular - Lean Inception Piloto	51
Figura 25 Aplicação UVApp e seus dois módulos – Outsystems.....	52
Figura 26 Módulo UVApp_CORE da aplicação UVApp – Outsystems	53
Figura 27 Campo Public setado como ‘Yes’ - Outsystems.....	53
Figura 28 Gerenciador de dependências – Outsystems.....	54
Figura 29 Criação do ícone plus - Outsystems.....	55
Figura 30 Criação da tela NotificationDetail - Outsystems	56
Figura 31 Criação do botão salvar - Outsystems.....	56
Figura 32 Client Action SalvarOnClick - Outsystems	57
Figura 33 Criação do Retrieve - Outsystems.....	58
Figura 34 Associando Client Action ao item da lista - Outsystems	59

Figura 35 Atualizando um item da lista de notificações - Outsystems	60
Figura 36 Notificação agendada - Outsystems	61
Figura 37 Client Action DeletarOnClick - Outsystems.....	61
Figura 38 Tela de notificação - Outsystems	62
Figura 39 Criar notificação - Outsystems.....	63
Figura 40 Tela de notificação - Outsystems	64
Figura 41 Atualização de notificação - Outsystems	65
Figura 42 Tela de notificação - Outsystems	66
Figura 43 Deleção de notificação - Outsystems	67
Figura 44 Tela de notificação - Outsystems	68
Figura 45 Diagrama de entidade de relacionamento <i>Notifications</i>	69
Figura 46 Tela de criação do modelo banco de dados e suas duas tabelas - MySQL	70
Figura 47 Tabela notifications - MySQL	71
Figura 48 Criando uma instância local - MySQL	72
Figura 49 Banco de dados mydb - MySQL.....	72
Figura 50 Página inicial de um projeto do tipo Tabs - Ionic	78
Figura 51 Página de notificações - Ionic	83
Figura 52 Página de criação de notificação - Ionic	84
Figura 53 Página de notificações - Ionic	85
Figura 54 Página de atualização da notificação - Ionic	86
Figura 55 Página de notificações com data alterada - Ionic.....	87
Figura 56 Página de detalhes da notificação - Ionic.....	88
Figura 57 Página de notificações - Ionic	89

LISTA DE TABELAS

Tabela 1- Associação entre os métodos das partes envolvidas no CRUD	76
Tabela 2 - Análise de horas trabalhadas.....	89

LISTA DE ABREVIATURAS E SIGLAS

- AJAX – *Asynchronous JavaScript And XML* (JavaScript e XML Assíncronos)
- ALGOL – *Algorithmic Language* (Linguagem Algorítmica)
- AM – *Agile Modeling* (Modelagem Ágil)
- API – *Application Programming Interface* (Interface de Programação de Aplicações)
- APPS – *Applications* (Aplicações)
- ASD – *Adaptative Software Development* (Desenvolvimento de Software Adaptativo)
- COBOL – *Common Business Oriented Language* (Linguagem Comum Orientada Para os Negócios)
- CRUD – *Create, Retrieve, Update, Delete* (Criar, Buscar, Atualizar, Deletar)
- CSS – *Cascading Style Sheet* (Folha de Estilo em Cascatas)
- DSDM – *Dynamic Systems development Method* (Método de Desenvolvimento de Sistemas Dinâmicos)
- FDD – *Feature Driven Development* (Desenvolvimento Dirigido a Funcionalidades)
- HTML – *Hipertext Markup Language* (Linguagem de Marcação de Hipertexto)
- HTTP – *Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto)
- ID – *Identifier* (Identificador)
- IDE – *Integrated Development Environment* (Ambiente de Desenvolvimento Integrado)
- JSON – *JavaScript Object Notation* (Notação de Objeto JavaScript)
- LSD – *Lean Software Development* (Desenvolvimento de Software Enxuto)
- MVP – *Minimum Viable Product* (Mínimo Produto Viável)
- NPM – *Node Package Manager* (Gerenciador de Pacotes do Node)
- PO – *Product Owner* (Dono do Produto)
- REST – *Representational State Transfer* (Transferência Representacional de Estado)
- SQL – *Structured Query Language* (Linguagem de Consulta Estruturada)
- TCC – Trabalho de Conclusão de Curso
- TI – Tecnologia da Informação
- UAP – *Unified Agile Process* (Processo Unificado Ágil)
- URL – *Unniform Resource Locator* (Localizador Uniforme de Recursos)
- UVA – Universidade Veiga de Almeida
- UX – *User Experience* (Experiência do Usuário)
- XML – *Extensible Markup Language* (Linguagem de Marcação Extensiva)
- XP – *Extreme Programming* (Programação Extrema)

SUMÁRIO

1	INTRODUÇÃO	15
2	METODOLOGIA ÁGIL.....	18
2.1	METODOLOGIA ÁGIL.....	18
2.2	PROCESSO ÁGIL	20
2.3	MODELOS DE PROCESSO ÁGIL	21
2.3.1	<i>Desenvolvimento Enxuto de Software.....</i>	<i>22</i>
2.4	LEAN STARTUP	23
3	MÉTODOS DE DESENVOLVIMENTO DE SOFTWARE.....	25
3.1	PROGRAMAÇÃO TRADICIONAL.....	26
3.2	LOW-CODE	28
4	APLICATIVO MÓVEL UVAPP	30
4.1	LEAN INCEPTION	30
4.2	DESENVOLVIMENTO COM METODOLOGIA LOW-CODE	51
4.3	DESENVOLVIMENTO COM METODOLOGIA TRADICIONAL	68
4.3.1	<i>Banco de dados.....</i>	<i>69</i>
4.3.2	<i>Back-end.....</i>	<i>73</i>
4.3.3	<i>Front-end.....</i>	<i>77</i>
4.4	ANÁLISE E COMPARAÇÃO DAS METODOLOGIAS	89
5	CONCLUSÃO.....	93
	REFERÊNCIAS	95

1 INTRODUÇÃO

As linguagens de programação são o elo de comunicação entre seres humanos e computadores. Elas permitem que se crie e desenvolva sistemas capazes de otimizar o trabalho manual, ou até mesmo o raciocínio. Na história da tecnologia, muitas linguagens foram desenvolvidas com este objetivo, desde as mais complexas para os humanos até as mais simples. Esse processo de evolução começou com linguagens de baixo nível, utilizadas na comunicação direta com o processador, e seguiu evoluindo através de uma estrutura que se aproxima mais da linguagem humana, a qual denomina-se linguagens de alto nível. Essas últimas são compiladas ou interpretadas para a linguagem de máquina, onde são processadas e executadas.

Durante as ondas da evolução das linguagens de programação, achou-se necessário classificá-las baseando-se em suas funcionalidades. Essa categorização recebeu o nome de paradigma de programação e possui algumas divisões como paradigma procedural, funcional, orientado a objeto, entre outros. O primeiro arquétipo é baseado em instruções geralmente executadas sequencialmente, ainda que apresentem várias maneiras de paralelizar os processos, e seus tipos são primitivos como inteiros, caracteres, números flutuantes, matrizes, entre outros. O paradigma funcional se pauta no ponto chave de que os comandos não são necessários e tudo pode ser feito com expressões, ou seja, essa programação trata as funções de modo que as mesmas possam ser passadas como parâmetros e valores para outras funções e seus resultados possam ser gravados em uma constante. O terceiro arquétipo, orientado a objeto, é dominante atualmente. Segundo o livro *The Evolution of Programming Languages* pode-se definir este padrão como sendo:

Uma característica fundamental de nossa compreensão do mundo é que organizamos nossa experiência como um número de objetos distintos (mesas e cadeiras, empréstimos bancários e expressões algébricas, ...) Quando desejamos resolver um problema em um computador geralmente precisa construir dentro do computador um modelo desse aspecto do real ou conceitual mundo ao qual a solução do problema será aplicada. (traduzido de HOARE, 1968 apud GROGONO, 2002, p.36).

A partir das elucidações sobre a história e características da programação, pode-se introduzir o assunto da pesquisa deste trabalho que tem como objetivo utilizar duas metodologias de programação de aplicações, *low-code* e programação tradicional, onde ambas, apesar de distintas, possuem a mesma arquitetura. As metodologias serão apresentadas perante a análise do processo de desenvolvimento, através de conhecimentos explícitos e

tácitos da construção de uma aplicação mobile.

O conceito dessa forma de programar – *low-code* – vem de há mais de 20 anos, com aplicações como o *Delphi* – cuja primeira *release* foi em 1995 –, que era dedicado ao desenvolvimento de sistemas desktop e possuía uma interface gráfica que permitia ao programador utilizar ferramentas como botões, caixas de textos, e outros, através das ações de clicar e arrastar. Contudo, ultimamente, cada vez mais esse conceito vem sendo ampliado e introduzido no mercado – principalmente voltado para desenvolvimento *web*.

Essas metodologias são formas de desenvolvimento de sistemas que utilizam abordagens diferentes apesar de necessitarem da mesma infraestrutura. A tradicional retrata, como o próprio nome sugere, uma forma convencional de programação, onde a codificação é necessária durante toda a construção. A outra, é uma forma mais simples de programar, onde é necessário escrever poucas linhas de código, ou muitas quando os desenvolvimentos se tornam mais complexos.

Todo esse movimento influencia na integração da tecnologia e o ordinário da vida cotidiana, pois permite que muitas pessoas, inclusive quem não tem técnica ou experiência em programação, experimente e se aproxime dessa tendência mundial crescente em exponencial que é a computação. Contudo, a qualidade das aplicações continua dependendo dos conhecimentos técnicos aplicados aos desenvolvimentos, e também, quando comparada à metodologia da programação tradicional, deve-se buscar um ponto de atenção na qualidade da lógica e do código escrito para avaliar a equivalência.

Também, o crescimento do *low-code* não significa que a metodologia tradicional de codificação esteja perdendo espaço, pois ainda são muito necessárias para algumas aplicações feitas através do paradigma estruturado, funcional e do orientado a objetos – dependendo do intuito.

O trabalho busca avaliar, após o desenvolvimento realizado utilizando as duas tecnologias referenciadas, o processo de aprendizagem obtido em cada uma das frentes. Esta avaliação é essencial para compará-las expondo os níveis de dificuldade e esforço de cada uma, e explicar a importância de cada uma. Todavia, antes de realizar a construção do sistema, será proposto o desenvolvimento de um mínimo produto viável para direcionar esta construção. Para chegar à conclusão de qual produto deve ser criado, fez-se uso de um *framework* ágil denominado *Lean Inception* que é baseado nos conceitos da *Lean Startup*.

Este trabalho está organizado em 5 capítulos. Após esta introdução, o Capítulo 2 introduz a metodologia ágil, seus princípios e aprofunda no modelo do desenvolvimento enxuto de software, o qual serve de base para a *Lean Startup*, que é um livro cujos

ensinamentos são sobre a utilização do método *lean* aplicado ao mundo organizacional. O Capítulo 3 apresenta as metodologias de desenvolvimento de aplicações escolhidas para a dissertação deste trabalho, explicando a importância de ambas e dando exemplos de ferramentas apropriadas para cada uma. No Capítulo 4, o *framework Lean Inception* foi utilizado para construir MVPs, onde no primeiro, uma funcionalidade foi escolhida para ser desenvolvida através da plataforma *low-code* Outsystems e da metodologia tradicional de programação, utilizando Python e Ionic com Angular para codificar *back-end* e *front-end*. Também, ainda neste capítulo, é feita uma análise do esforço necessário no desenvolvimento de cada uma das metodologias. Finalmente, no Capítulo 5, serão apresentadas a importância do trabalho para a autora e para a sociedade, como as adversidades que surgiram durante o processo de construção foram contornadas, suas contribuições para o porvir e sugestões de trabalhos futuros.

2 METODOLOGIA ÁGIL

Neste capítulo são apresentados a metodologia ágil e alguns modelos que facilitam o gerenciamento de projetos desde o planejamento ao desenvolvimento de produtos. Aqui será dissertado sobre o que é a metodologia ágil e a sua importância, o processo envolvido e os modelos de processo ágil.

Apesar de citar a maior parte dos modelos ágeis, o trabalho possui o foco em um específico que é o Desenvolvimento de *Software* Enxuto - DSE (*Lean Software Development*). O DSE é um insumo para o *framework* utilizado no desenvolvimento do projeto como um todo. O *framework* designa-se por *Lean Inception*.

2.1 METODOLOGIA ÁGIL

A “Aliança dos Ágeis” (“Agile Alliance”) é formada por um grupo de dezessete renomados desenvolvedores, autores e consultores da área de software - dentre eles Kent Beck -, que em 2001 assinaram o “Manifesto para o Desenvolvimento Ágil de Software” (“Manifesto for Agile Software Development”) que se constitui do seguinte modo:

Desenvolvendo e ajudando outros a desenvolver software, estamos desvendando formas melhores de desenvolvimento. Por meio desse trabalho passamos a valorizar:
 Indivíduos e interações acima de processos e ferramentas
 Software operacional acima de documentação completa
 Colaboração dos clientes acima de negociação contratual
 Respostas a mudanças acima de seguir um plano
 Ou seja, embora haja valor nos itens à direita, valorizaremos os da esquerda mais ainda. (PRESSMAN, 2011, p.81).

O desenvolvimento ágil possui benefícios interessantes, contudo, não é indicado para todos os projetos, pessoas, produtos e situações, nem faz oposição à prática tradicional de engenharia de *software* consistente. Em suma, métodos ágeis originaram-se de esforços para aprimorar limitações reais e perceptíveis da engenharia de *software* convencional; e assim, podem ser aplicados como uma filosofia geral para todos os trabalhos de software.

Com a imprevisibilidade da evolução de um sistema computacional, onde as condições de mercado e as necessidades do usuário final mudam com uma alta velocidade, torna-se difícil definir todos os requisitos antes que o projeto se inicie; então, em muitas situações é necessário ter agilidade suficiente para prover respostas a um ambiente fluido e de negócios.

Fluidez provoca mudanças, e se estas forem mal gerenciadas podem ser caras. Uma das qualidades da abordagem ágil é a capacidade de reduzir custos da mudança ao longo de

todo processo de *software*.

Conforme expõe Roger S. Pressman (2011, p.83), o conceito de agilidade é maior que uma resposta à mudança: abrange toda a filosofia do manifesto ágil supracitado, busca equilibrar o nível de interação com o cliente integrando-o no time ágil, facilita a comunicação interna entre o time e do time com as equipes envolvidas, reconhece que apesar de ser necessário um roteiro para guiar o projeto, o mesmo deve ser flexível e não descarta a documentação, contudo evidencia que entregas funcionais rápidas são mais importantes.

Para aplicar a metodologia, é necessário que a mesma seja projetada com a equipe a fim de fazer com que se adaptem ao novo modelo e alinhem-se com a proposta do manifesto o quanto antes, sem descartar os artefatos essenciais dos legados. É fundamental que se enfatize o processo de entrega incremental, cujo valor preza entregar o mínimo produto viável; ou seja, entregar o mínimo que o produto tem a oferecer, porém fazê-las constantes e incrementais.

O gráfico da Figura 1 mostra a importância da metodologia e exemplifica o ganho que ela traz para o projeto em relação ao custo de alteração:

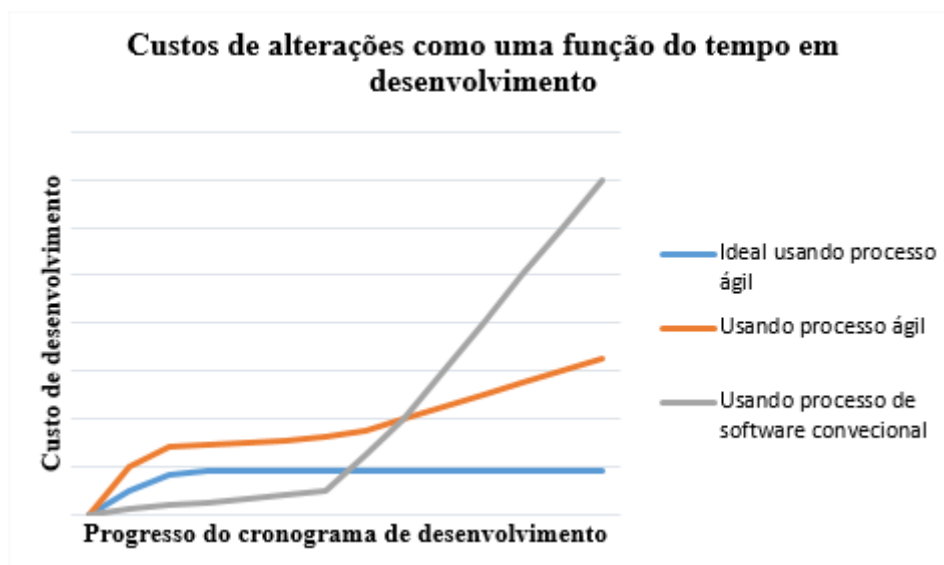


Figura 1 Custos de alterações como uma função do tempo em desenvolvimento

Fonte: Adaptado de PRESSMAN (2011).

Quando as entregas são incrementais – como acontece no processo ágil – e o processo é bem elaborado, o custo das mudanças é atenuado e sua curva é “achatada”, pois permite que a equipe compreenda melhor as alterações, trabalhe com partes de desenvolvimento – evitando erros e destacando cada teste –, e reduza ao máximo o retrabalho.

2.2 PROCESSO ÁGIL

O processo ágil é embasado na adaptabilidade, que permite administrar mais facilmente a imprevisibilidade que é inevitável em qualquer projeto. Existem no mínimo três preceitos fundamentais que caracterizam a maioria dos projetos de *software*:

1. É difícil afirmar antecipadamente quais requisitos de *software* irão persistir e quais sofrerão alterações. É igualmente difícil prever de que maneira as prioridades do cliente sofrerão alterações conforme o projeto avança.
2. Para muitos tipos de *software*, o projeto e a construção são “interconduzidos”. Ou seja, ambas as atividades devem ser realizadas em sequência, para que os modelos de projeto sejam provados conforme sejam criados. É difícil prever quanto trabalho será necessário antes que seu desenvolvimento seja implementado para avaliar o projeto.
3. Análise, projeto, construção e testes não são tão previsíveis quanto gostaríamos que fossem.

As entregas dos incrementos de *software* devem ser feitas num curto espaço de tempo e as adaptações devem acompanhar o ritmo das mudanças. Dessa forma institui-se uma estratégia de desenvolvimento incremental que capacita o cliente a avaliar as entregas regularmente, fornecer o *feedback* necessário para a equipe de *software* e sugestionar nas adaptações de processo feitas para incluir o *feedback* adequadamente.

A Aliança Ágil estabelece 12 princípios de agilidade (ROGER, 2011):

1. A maior prioridade é satisfazer o cliente por meio de entrega adiantada e contínua de *software* valioso.
2. Acolha bem os pedidos de alterações, mesmo atrasados no desenvolvimento. Os processos ágeis se aproveitam das mudanças como uma vantagem competitiva na relação com o cliente.
3. Entregue *software* em funcionamento frequentemente, de algumas semanas para alguns meses, dando preferência a intervalos mais curtos.
4. O pessoal comercial e os desenvolvedores devem trabalhar em conjunto diariamente ao longo de todo o projeto.
5. Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e apoio necessários e confie neles para ter o trabalho feito.
6. O método mais eficiente e efetivo de transmitir informações para e dentro de uma equipe de desenvolvimento é uma conversa aberta, de forma presencial.
7. Software em funcionamento é a principal medida do progresso.

8. Os processos ágeis promovem desenvolvimento sustentável. Os proponentes, desenvolvedores e usuários devem estar capacitados para manter um ritmo constante indefinidamente.

9. Atenção contínua para com a excelência técnica e para com bons projetos aumenta a agilidade.

10. Simplicidade – a arte de maximizar o volume de trabalho não efetuado – é essencial.

11. As melhores arquiteturas, requisitos e projetos emergem de equipes que se auto organizam.

12. A intervalos regulares, a equipe se avalia para ver como tornar-se mais eficiente, então sintoniza e ajusta seu comportamento de acordo.

Fica sob o critério de cada projeto ágil atribuir os “pesos” e “medidas” na hora de implementar esses 12 princípios, e ainda, alguns modelos ignoram um ou outro; entretanto, essa é a essência do ágil que é extremamente relevante para fazer um projeto harmônico em todas as frentes.

2.3 MODELOS DE PROCESSO ÁGIL

Existem alguns modelos de processo ágil que são implantados em muitas organizações, com:

1. *Extreme Programming* (XP)
2. *Adaptative Software Development* (ASD)
3. *Scrum*
4. *Dynamic Software Development Method* (DSDM)
5. *Crystal*
6. Feature Driven Development (FDD)
7. *Lean Software Development* (LSD)
8. *Agile Modeling* (AM)
9. *Unified Agile Process* (UAP)

Para ilustrar o processo ágil de forma mais detalhada, será apresentado de forma mais profunda o modelo base de um método – método este, inclinado a aplicações empresariais - que deu origem ao *framework* utilizado para o desenvolvimento deste trabalho que é o *Lean Software Development* ou Desenvolvimento de Software Enxuto (LSD).

2.3.1 Desenvolvimento Enxuto de Software

Princípios são verdades latentes imutáveis ao longo do tempo e do espaço, já as práticas são a aplicação desses princípios em situações específicas. As práticas devem mudar de acordo com a necessidade do ambiente.

A parte que se designa ao enxuto dessa metodologia possui sete princípios: eliminar desperdício, desenvolvimento de qualidade, criação de conhecimento, tomar decisões no momento certo, entregas rápidas, respeitar pessoas e otimizar o todo.

No conceito enxuto, o desperdício é um grande obstáculo, pois é algo fácil de acontecer devido a imprevisibilidade de um projeto, e que não agrega valor – valor como inferido pelo cliente - a um produto. Se os desenvolvedores programam mais funcionalidades do que o projeto precisa de imediato é um desperdício; o ideal é descobrir o que um cliente deseja e, em seguida, desenvolvê-lo e entregar exatamente o que ele quer, sem tirar nem pôr. Tudo que atrapalha a necessidade de um cliente é desperdício.

O segundo princípio preza, como o próprio nome sugere, um desenvolvimento de qualidade. Ele enfatiza que durante o processo haja muita concentração para evitar o máximo de erros e não se perder com testes quebrados. Evitar criar defeitos sempre.

A criação de conhecimento implica em participar direta ou indiretamente do negócio/mercado. O desenvolvimento de software é um processo de criação de conhecimento, pois a codificação sempre valida a arquitetura esboçada anteriormente, ou seja, na prática, o *design* detalhado sempre ocorre durante o desenvolvimento. Para melhor compreender:

Um dos aspectos intrigantes do desenvolvimento em "cascata" é a ideia de que o conhecimento, na forma de "requisitos", existe antes e separado da codificação. Desenvolvimento de software é um processo de criação de conhecimento. Embora um conceito arquitetônico geral esboçado antes da codificação, a validação dessa arquitetura ocorre como o código está sendo escrito (traduzido de POPPENDIECK M.; POPPENDIECK T., 2006, p.57-58).

Tomar decisões no momento certo é o quarto princípio. Este é essencial para que a equipe aprenda a aguardar o final do *timebox* para tomar decisões – principalmente se forem críticas. Decisões que são irreversíveis ou vão causar grande impacto devem ser reagendadas para último lugar na lista, a fim de que dê tempo para entender melhor todas as variáveis do problema e refletir sobre o maior número de soluções possíveis.

O quinto princípio de entregas rápidas é essencial, pois entregas funcionais e contínuas fazem o cliente valorizar o trabalho do fornecedor. Ao invés de esperar muito tempo para obter algum sistema enorme e que demandará muitos testes, ocorrem entregas de partes desse

sistema que são o produto minimamente viável; este último é incrementado a cada entrega.

Respeitar as pessoas é essencial, tanto na vida profissional quanto pessoal. Esse sexto princípio retrata como os olhares para as pessoas devem ser modificados, por exemplo, algumas vezes pode-se observar pessoas subestimando a capacidade de outras, e muitas vezes agem para imobilizá-la. Entretanto, é importante ter em mente que as próprias pessoas têm de se mostrar capazes de resolver suas questões e respeitar as decisões alheias, mesmo que não esteja de concordância.

Por último, o sétimo princípio visa otimizar o fluxo de valor como um todo. Sabe-se que um subfluxo influencia no fluxo inteiro, então se houver necessidade de mudanças em alguma parte, é necessário avaliar o impacto no todo para não sofrer consequências desagradáveis.

2.4 LEAN STARTUP

Lean startup ou *startup enxuta* é um livro que possui uma perspectiva voltada para o empreendedorismo. Seu nome originou-se da revolução que Taiichi Ohno e Shiego Shingo promoveram na *Toyota*, a produção enxuta. Conforme Ries (2011, p.21) salienta, a *startup* enxuta é uma nova maneira de considerar o desenvolvimento de produtos novos e inovadores, que enfatiza interação rápida e percepção do consumidor, uma grande visão e grande ambição, tudo ao mesmo tempo.

Em suma, o livro sugere que a opinião de cliente e consumidores sejam sempre incluídas no processo de construção do modelo de negócios – incluindo estratégias econômicas, características do produto, canais de distribuição -, assim é possível ser mais assertivo no desenvolvimento, evitando desperdícios, e também criar um produto mínimo viável ou MVP que atenda às expectativas dos usuários, clientes e consumidores. Com a entrega do MVP e o feedback recebido, é possível fazer melhorias e seguir criando novos de forma mais eficiente e eficaz.

Uma das formas de descrever a metodologia da *startup* enxuta, segundo o próprio livro é:

A metodologia da *startup* enxuta reconhece os esforços da *startup* como experimentos que testam sua estratégia, para ver quais partes são brilhantes e quais são absurdas. Um experimento verdadeiro segue o método científico. Começa com uma hipótese clara, que prognostica o que pode acontecer. Em seguida, testam-se tais prognósticos de forma empírica. Da mesma forma que a experimentação científica é permeada pela teoria, a experimentação da *startup* é orientada pela visão da startup. O objetivo de todo experimento associado à startup é descobrir como desenvolver um

negócio sustentável em torno daquela visão (RIES E., 2011, p.45).

A *startup* enxuta gira em torno de cinco princípios. O primeiro afirma que empreendedores estão por toda parte e que uma *startup* não se resume ao local onde se trabalha. Segundo Ries (2011, p.) o conceito de empreendedorismo se resume a qualquer pessoa que trabalhe dentro do que ele define como sendo uma *startup*: uma instituição humana projetada para criar novos produtos e serviços sob condições de extrema incerteza. Ou seja, a abordagem desse tipo de instituição pode funcionar em empresas de qualquer tamanho, mesmo numa de grande porte, em qualquer setor ou atividade.

O segundo princípio diz que empreender é administrar. Uma *startup* precisa de um tipo de gestão especificamente constituída para seu contexto de extrema incerteza. O autor crê que “empreendedor” deveria ser considerado um cargo em todas as empresas modernas que dependem da inovação para seu crescimento futuro.

No princípio que sucede o anterior, reforça-se a importância do aprendizado validado. Em outras palavras, o movimento das *startups* deve transcender o ordinário da fabricação, da relação com o cliente e da geração de lucros, as mesmas devem servir para aprender a desenvolver um negócio sustentável. Tal aprendizagem pode, cientificamente, ser validada através de experimentos frequentes que possibilitam que os empreendedores testem cada elemento de sua visão.

Construir-medir-aprender é o que se enfatiza no quarto princípio. Neste, revela-se que transformar ideias em produtos, obter métricas das reações dos clientes e aprender a definir se é o momento de mudar ou preservar, são as propostas fundamentais de uma *startup*. Todos os processos de *startup* bem-sucedidos devem-se voltar a acelerar esse ciclo de *feedback*.

Por fim, o último princípio refere-se à contabilidade para inovação. Métricas de progresso, definição de marcos e priorização de trabalhos são assuntos que merecem atenção especial para galgar melhorias nos resultados empreendedores e para poder atribuir responsabilidades aos inovadores.

Apresentados alguns conceitos sobre *lean startup*, tais como sua definição e princípios, além de, anteriormente, alguns modelos, processos, e definições sobre metodologias ágeis, é viável seguir explicando os próximos conceitos que compõem o presente trabalho. No próximo capítulo serão expostos alguns conceitos sobre metodologias de desenvolvimento de software, as quais serão divididas em duas partes: *low-code* e programação tradicional.

3 MÉTODOS DE DESENVOLVIMENTO DE SOFTWARE

As linguagens de programação modernas surgiram entre 1950 e 1960 com o *FORTRAN*, *LISP*, *COBOL* e *ALGOL*. Essas linguagens permitiam certa aproximação com a linguagem humana, até que foram evoluindo – e até hoje continuam –, sempre buscando otimizar suas estruturas, semânticas e complexidades, a fim de se relacionarem mais eficazmente com a máquina e eficientemente com o humano. Na década entre 1960 e 1970, “houve um amadurecimento do campo da linguagem de programação. Durante esse período, a batalha pelo uso de idiomas de nível superior foi o que claramente ganhou o sentido de que a codificação por máquina se tornara a exceção em vez da regra” (SAMMET, 1972, p.607). Do início de 1960 e final de 1970 as linguagens de programação floresceram com a criação do *Simula*, *C*, *Prolog*, *SQL*, *Pascal*, entre outros. A partir daí vieram novos conceitos como orientação a objetos e a aspectos, surgiram novas linguagens como *Java*, *Python*, inclusive para a *web* como *HTML*, *Javascript*, entre outras.

No decorrer desses anos que foram marcos para a história das linguagens de programação, foram criados paradigmas os quais essas linguagens se encaixam, em um ou mais. Paradigmas de linguagens de programação podem ser definidos como “cada paradigma é definido por um conjunto de conceitos de programação, organizados em uma linguagem básica simples chamada linguagem de núcleo do paradigma.” (ROY, 2012, p.12). Alguns dos paradigmas criados e usados até hoje são o imperativo ou procedural, programação por restrição, programação lógica, orientado a objetos, orientado a aspectos, programação relacional, entre outros.

O capítulo 3, apresenta dois métodos de desenvolvimento de software. Apesar de ambos utilizarem os mesmos paradigmas como o da programação relacional e a orientada a objetos, ambos utilizam diferentes linguagens de programação. Um é executado através de plataformas de programação *low-code*, ou seja, plataformas que permitem a construção de aplicações de forma mais lúdica, apresentado o conceito ilustrado de *workflow*, *drag and drop*, entre outros. O outro, da maneira mais convencional de codificação tecnológica; através de uma IDE, o usuário deve escrever linhas de código para realizar cada etapa das funcionalidade e visualizações possíveis para a aplicação.

Na atualidade, o número de plataformas *low-code* vem crescendo e prometendo se tornar uma tendência. Contudo, é praticamente impossível substituir o método normalizado, pois existem programas que são programados em linguagens denominadas de baixo nível – maior proximidade do entendimento de máquina –, e para estas, ainda é necessário a utilização

das linhas de código. A forma de programação que substitui essas linhas, é usada para aplicações construídas por linguagens designadas para alto nível – maior proximidade do entendimento humano.

Seguindo esta linha de raciocínio, serão apresentadas algumas linguagens, IDEs, plataformas e programas utilizados em ambas metodologias.

3.1 PROGRAMAÇÃO TRADICIONAL

A programação possui vários níveis em seu processo de evolução. No princípio, as codificações tinham um formato estruturado, onde o foco era desenvolver algoritmos sequenciais ou paralelos com ênfase em decisão e com auxílio de métodos estruturados para fazer interações. Essas interações permitem a criação de estrutura de dados nos formatos de vetores, matrizes, registros, listas, pilhas, filas, árvores, tabela *Hash*, grafos e outros.

O conceito de objetos veio sendo implementado com a evolução das linguagens de programação e seus paradigmas, facilitando criações de programas mais complexos. A orientação a objeto enfatiza o conceito de trazer objetos do mundo real para o mundo do software através do uso de classes, conceitos de herança, encapsulamento, acoplamento, interface, abstração, polimorfismo, entre outros.

O ambiente de desenvolvimento integrado ou IDE é um software que facilita o desenvolvimento de aplicação por unificar ferramentas comuns ao desenvolvimento em uma única interface gráfica. Sua estrutura geralmente consiste em editor de código-fonte, automação da compilação local e *debugger*.

O desenvolvimento de aplicações é dividido em duas partes, o *front-end* e o *back-end*.

O *front-end* é a frente do desenvolvimento que atua nas *interfaces*, gerenciando as telas que serão apresentadas e interagindo com o usuário, de forma a receber solicitações e mostrar respostas à essas solicitações. Há mais ou menos 10 anos atrás, o conceito que tem-se atualmente de *front-end* não existia, os profissionais que trabalhavam com aplicações web eram os chamados *webmaster* e *web designer*, onde o primeiro era responsável pela conexão com a base de dados e interações mais avançadas com o usuário e o segundo pelo design e partes não muito complexas. Essas aplicações eram geralmente usadas no *Internet Explorer 6* e *Netscape* (atual *Mozilla*) e codificadas em arquivos textos com extensão *.txt* que posteriormente era mudada para *.html* através de editores de texto como o *FrontPage* ou *Dreamweaver*. Depois de algum tempo, os *web designers* puderam aproveitar dos *plug-ins* que surgiram para facilitar a vida que quem trabalhava com essa frente da programação. Eles permitiam criar vídeos, imagens, áudios, etc, e tudo com a facilidade de criar e arrastar alguns

elementos, devendo apenas incluir alguns código como *play()* ou *stop()*. Em 2006, com o lançamento da biblioteca JQuery e a evolução do AJAX, XML e JSON, os *webmasters* – assim como os *web designers* – tiveram maior facilidade para manipular códigos em *Javascript* (linguagem utilizada para a programação *web* na época, junto com HTML 4 e CSS 1 e/ou CSS 2), aumentando a velocidade de carregamento das páginas e facilitando as entregas de dados ao *back-end*.

Em 2007 – início da acessibilidade à *internet* via *mobile* – a *Apple* decidiu que não deveria haver nenhum tipo de *plug-in* no *iPhone*, o que fez com que o *flash* fosse afetado diretamente. Em pouquíssimo tempo o novo dispositivo da *Apple* dominou o mercado e desde esse evento praticamente todo site desenvolvido tinha sua versão *mobile* e *web*. Muitas bibliotecas e *frameworks* começaram a surgir e aproveitar o espaço vazio para se destacar como *JQuerie Mobile*, por exemplo. Com tudo isso, as profissões de *web designer* e *webmaster* se mesclaram e tornaram-se o que chamamos hoje de desenvolvedor *front-end*, que tinham não somente as mesmas responsabilidades dos antigos, mas também trabalha com programação *Javascript*, se preocupa com diversas resoluções para adequar o *website* e se importa com a semântica e qualidade do código separando uma página pro HTML, uma para o CSS e outra para *Javascript*.

Em 2010 houve uma revolução por causa do lançamento do iPad que quebrou paradigmas sobre a interação do usuário com o meio digital. Como é um dispositivo *mobile* com IOS, o *Flash* perdeu ainda mais espaço e muitos desenvolvedores migraram para HTML5 e *Javascript*, causando aumento da comunidade e a qualidade. Assim o foco do desenvolvimento *front-end* se voltou para *Javascript*, HTML5 e CSS3; e com esse foco, logo começaram a ser criados diversos *frameworks* que facilitaram o desenvolvimento *web* e popularizaram o *front-end*.

Atualmente, as linguagens *front-end* são bem avançadas e o cenário é positivo e aberto para inovações. Algumas das linguagens e frameworks utilizados para o desenvolvimento *front-end* são *Python*, *Go*, *Javascript*, HTML5, CSS3, *React*, *Angular*, *Vue*, *Kotlin*, *Dart*, entre outros.

Em relação à história do *back-end*, como já fora citado no início desta seção, esta frente do desenvolvimento teve sua evolução vinda da programação estruturada e posteriormente o uso do conceito de objetos. Assim, o desenvolvimento *back-end* cuida é o cérebro das aplicações *web*, pois é nele que funções são criadas para executar as solicitações vindas do *front-end*. A interface com o usuário não tem acesso direto aos dados, seu acesso é através de requisições. Quando o usuário faz alguma requisição através do *front*, o *back* (lado

do servidor) cria processos e ações complexas para atender à demanda do lado do cliente tanto para buscar, criar, atualizar ou deletar dados que estão na base de dados.

O fluxo completo baseia-se em uma aplicação *web* ou *mobile*, onde o cliente faz alguma requisição – comumente chamada de *request* – através da *interface* do programa (*front-end*), o servidor (*back-end*) processa essa requisição operando em cima dela e envia solicitações para o banco de dados, este último executa buscas e envia uma resposta – comumente chamada de *response* – através do servidor, para o *front-end*, que por sua vez a apresenta ao cliente.

Algumas das linguagens e frameworks utilizados para o desenvolvimento *back-end* são *Javascript*, *Java*, *Python*, *C*, *C++*, *C#*, *Swift*, *NodeJS*, *Django*, *Ruby on Rails*, *Go*, *Flask*.

Na próxima subseção será apresentado como funciona este processo feito por meio de plataformas *low-code*.

3.2 LOW-CODE

As ferramentas *low-code* vêm crescendo rapidamente e tomando seu espaço no mundo como uma forma, não só de integração das pessoas não técnicas com a tecnologia de desenvolvimento de *software*, mas também de agilizar muitos processos no decorrer do desenvolvimento de aplicações *web*. Contudo, deve-se salientar que o conceito de *drag and drop* – utilizado em *low-code* – é antigo, pois ferramentas com o *Delphi Borland* – além de linguagem de programação era compilador e IDE – e o *Visual Basic* – linguagem de programação produzida pela *Microsoft* –, já o usavam no desenvolvimento de aplicações *desktop*.

Plataformas desse tipo de programação dão suporte ao desenvolvimento *web* e *mobile* e oferecem recursos de fácil compreensão através de uma camada de modelo de visualização gráfica, onde o programador arrasta componentes e solta no lugar que deseja, concebe de forma explícita estruturas de banco de dados e fluxo de processos através da visualização da modelagem de banco de dados e *workflows* que são construídos automaticamente enquanto se conecta componentes da ferramenta, e utiliza linhas de código apenas para otimizações e personalizações mais complexas – se necessário.

Existe também uma metodologia bem parecida com esta que é *no-code*, ou seja, como o próprio nome sugere, não utiliza nenhuma programação manual. A codificação é completamente feita via *drag and drop*, e geralmente é utilizada para construir aplicações fáceis. Um exemplo de ferramenta *no-code* é o *Scratch*.

O intuito do *low-code* é reduzir ao máximo a necessidade de codificar manualmente,

contudo em alguns casos, quando a demanda é mais complexa e necessita-se de artifícios mais específicos do que as oferecidas pela plataforma, é concedida a oportunidade de *hands-on* – ou seja, escrever o código seguindo a metodologia da programação tradicional.

Antes de adquirir estas ferramentas, deve-se pensar e planejar muito bem, pois assim como tudo, ela possui seus prós e contras.

As desvantagens podem ser em relação a (1) segurança das aplicações, pois como estas são hospedadas nos servidores do próprio fornecedor, se algo acontecer com a plataforma há um enorme risco de perder a aplicação, também (2) deve-se ter o cuidado de não fazer aquisição de uma plataforma que vá necessitar de profissionais cujo o custo de mercado esteja muito caro, outro ponto é (3) que não se sabe ao certo o que está por trás do aplicativo, então a depurar pode ser dificultada, (4) algumas vezes será necessário trocar os processos de negócio - para adaptar o programa às necessidades do cliente - pelo fato de os componentes não serem profundamente acessíveis, (5) restrição do código do fornecedor, pois apesar de algumas plataformas disponibilizarem o código após o encerramento da conta, o código possui complexidade alta para o entendimento humano e não possui uma estrutura tão legível e de fácil entendimento, e por fim, (6) a falta de integrações com sistemas legados.

Já as vantagens podem ser muitas como a (1) facilidade e agilidade na prototipagem, (2) rápida curva de aprendizado, (3) redução de custos ao contratar pessoas, pois os desenvolvedores não precisam ser especializados, (4) rápida criação de aplicações *web* e *mobile*, permitindo que a organização seja mais ágil, (5) permite a reutilização de componentes usando as interfaces da interface do usuário, o que facilita alterações, (6) as pessoas podem fazer mais aplicações e mais funcionalidades sem habilidades em programação tradicional, (7) atualização rápida do aplicativo e (8) facilidade em um ambiente colaborativo de programação.

Alguns exemplos de plataformas *low-code* são *Appian*, *Visual LANSA*, *KISSFLOW*, *Mendix*, *Outsystems*, *Salesforce Lightning*.

Dentre as opções supracitadas, o Outsystems foi escolhido para o desenvolvimento deste projeto, devido à gratuidade e familiaridade da desenvolvedora.

4 APLICATIVO MÓVEL UVAPP

Esse capítulo retrata o percurso traçado para realizar o MVP, o qual serviu como insumo para definir o que seria desenvolvido através dos dois métodos de programação de aplicativos. Ademais, aqui também serão apresentados o engajamento e a jornada de desenvolvimento das funcionalidades escolhidas para atingir o objetivo final desse trabalho que é analisar o desenvolvimento através de ambas as abordagens de codificação.

Em um primeiro momento, será mostrado o passo a passo da *Lean Inception* realizada com um grupo totalizando oito estudantes da Universidade Veiga de Almeida, sendo a maioria do curso de Computação (divididos entre ciência e engenharia) e apenas um do curso de Administração. A graduanda Camila Delaroli Gaspar – aluna de Ciência da Computação - fez o papel de PO ou dona do produto, pois seu trabalho de conclusão de curso tem o objetivo de aplicar metodologias ágeis no desenvolvimento de um software e fazer análises sobre a experiência do usuário durante o processo, e como ela tinha a ideia de utilizar como produto de sua pesquisa o aplicativo UVA *Mobile* (atual aplicativo móvel da Universidade Veiga de Almeida), o papel lhe coube perfeitamente. Os outros estudantes: Bruno Gomide Ribeiro, Edgar Pinheiro da Câmara Júnior, Eduardo Albuquerque da Silva, Marcos Pedro Guerra Alves Miranda, Pedro Kats de Barros e Yan Barreto de Santana, foram colegas voluntários que tiveram o papel de desenvolvedores nas atividades da *Lean Inception*, e a autora deste TCC foi a facilitadora.

Depois de finalizado o primeiro MVP, uma de suas funcionalidades foi selecionada e separada em tarefas menores - como propõe Paulo Caroli em o livro *Lean Inception: Como Alinhar Pessoas e Construir o Produto Certo* -. Essas tarefas foram desenvolvidas na plataforma de *low-code Outsystems*, e também através da programação tradicional utilizando a IDE *Pycharm* com a linguagem de programação *Python*, do editor de texto *Visual Studio Code* fazendo uso do framework *Ionic* com o *Angular* – que usa a linguagem *Typescript*.

Durante o desenvolvimento realizado por meio de ambos os métodos, houve uma análise do processo de aprendizagem das ferramentas, a fim de que posteriormente as análises fossem comparadas uma à outra.

4.1 LEAN INCEPTION

A *Lean Inception*, ou Criação Enxuta é uma ferramenta inspirada na *Lean Startup*. Quando Paulo Caroli – idealizador e autor da *Lean Inception* – se encantou com o modelo ágil da *Lean Startup* de desenvolver produtos, teve um *insight*. Ele obteve boas respostas sobre os

pontos de aprender e medir, e em relação ao construir deparou-se com a seguinte questão: “como construir? ”. A partir disso, Caroli – com o apoio das matérias de Design Thinking – criou uma sequência de atividades que ajudam uma equipe a construir um MVP de forma simples, intuitiva, lúdica e ágil.

Depois de compartilhar com centenas de pessoas e trabalhar com os retornos recebidos, surgiu – e evoluiu – o workshop Lean Inception: uma sequência de atividades para a criação de produtos de forma enxuta fortemente influenciadas por Design Thinking e Lean Startup. (CAROLI, 2018, p.23).

A criação enxuta deve ser executada em um *workshop* com uma equipe de desenvolvimento e um facilitador para definirem funcionalidades e construírem MVPs em um cronograma de uma semana.

O MVP está na interseção entre valioso, usável e factível. Para obter sucesso em sua entrega e na conquista do mercado é necessário que tenha o fator “UAU”, ou seja, o fator diferencial e inovador, que fará o usuário literalmente ter essa expressão de “UAU”, realizado com o resultado final.

O ambiente do *workshop* deve ser preparado dentro de uma sala única, onde os participantes se encontrarão durante toda a semana, e os cartões, *post-its*, papel A4, canetas para todos, cartazes e *flip chart*, - materiais que serão utilizados durante todas as atividades - devem estar disponíveis.

As atividades do *workshop* são divididas em: visão do produto, o produto é-não é-faz-não faz, objetivo do produto, entendendo os *trade-offs*, definido personas, *brainstorming* de funcionalidades, revisão técnica de esforço, experiência do usuário e de negócio, jornadas do usuário, construção do sequenciador, calculando tempo, esforço e custo, e o canvas MVP. Para executar as atividades o facilitador deve dividir a equipe proporcionalmente em grupos de no máximo cinco pessoas.

Contudo, apesar da *Lean Inception* ser originalmente e usualmente aplicada presencialmente em uma sala de guerra (*war room*), dedicada durante uma semana apenas para estas atividades, a mesma sofreu mudanças e foi aplicada virtualmente utilizando o *Hangouts* como ferramenta de vídeoconferência e o *Google Drive* para disponibilizar e compartilhar os *templates* com todos os envolvidos. Estas alterações foram necessárias por causa da quarentena a qual o mundo foi submetido no presente ano. Não só as ferramentas de apoio foram adaptadas, mas também o cronograma, o qual é apresentado na Figura 2.

Horário	Sábado 09/05/2020	Domingo 10/05/2020	Segunda 11/05/2020	Terça 12/05/2020	Quarta 13/05/2020	Quinta 14/05/2020	Sexta 15/05/2020	Sábado 16/05/2020	Domingo 17/05/2020
14:00	EU SOU								
14:30	KICK OFF								
15:00	VISÃO DO PRODUTO								
15:30	É-NÃO É-FAZ-NÃO FAZ								
16:00	OBJETIVO								
16:30	INTERVALO								
17:00	PERSONAS								
18:00									
18:30									
19:00	MAPA DE EMPATIA								
19:30									
20:00	CONSOLIDAR								
20:30									
21:00									
21:30									
22:00									
22:30									
23:00									

Figura 2 Cronograma Lean Inception Piloto

Fonte: autor.

Para começar, houve uma apresentação – feita pela facilitadora - pessoal e do objetivo do trabalho que se seguia, e uma apresentação do produto proposto pela Camila (dona do produto). Depois, uma atividade quebra-gelo foi iniciada onde todos se apresentaram e ficaram mais à vontade para continuar.

No primeiro dia foi feito mais atividades que o normal, pois no domingo (10/05/2020) era dia das mães, então as atividades foram unidas às do dia anterior. Notou-se que ao final os participantes estavam muito cansados e bem menos produtivos, entretanto todas as tarefas foram concluídas; e mais, ao final escreveram em um documento em branco todas as ideias de funcionalidades que foram surgindo ao longo das horas de trabalho.

Assim, iniciou-se efetivamente a primeira atividade que foi a visão do produto. Esta atividade é a essência do produto, e vai ajudar a guiar – de certa forma – as dinâmicas até o final, pois ela que direciona o objetivo. Aqui se define qual a estratégia de posicionamento, qual caminho inicial será trilhado e as características do produto. Todos participaram e alcançaram juntos os resultados apresentados na Figura 3.

TEMPLATE VISÃO DO PRODUTO
Para: Alunos e professores,
cujo: gerenciamento das informações e a comunicação entre as partes é falha.
O: UVApp
é um: aplicativo mobile
que: tem as funcionalidades do site - essenciais para o aluno e para o professor - da UVA e outras, e é de fácil usabilidade,
diferente do: UVA Mobile.
O nosso produto: É de fácil usabilidade e possui novas funcionalidades.

Figura 3 Visão do Produto - Lean Inception Piloto

Fonte: autor.

Cada atividade foi explicada previamente pela facilitadora antes de iniciar. Somente após todos terem entendido o intuito da tarefa é que a mesma começou a acontecer de fato. Algumas atividades tinham um tempo de cinco minutos para serem executadas individualmente por cada integrante e depois as respostas eram discutidas e condensadas em uma resposta única.

Seguindo adiante, a segunda atividade consentia em o time definir o que o produto é, o que ele não é, o que ele faz e o que ele não faz. Para isto, foi preciso esclarecer o que quer dizer É e FAZ. Na parte do É, deveriam descrever o produto como substantivo ou adjetivo, na do FAZ deveriam descrever com um verbo, indicando uma ação dele. Assim, depois do tempo acordado para executarem em particular, tudo foi compilado e transformado em uma coisa só. Muitas respostas eram semelhantes, então, essas viravam uma só, outras pareciam estar em desacordo com a coluna sugerida, e aí, foram redirecionadas para a certa. A Figura 4 ilustra o resultado final.

É	NÃO É	FAZ	NÃO FAZ
É um app mobile	Não é uma rede social	Faz notificações de eventos	Não faz pagamento on-line
É uma interface de comunicação entre alunos e professores	Não é um sistema completo	Faz lançamento de notas	Não faz pré-cadastro de novos alunos
É interface de gestão para alunos e professores	Não é um aplicativo Web	Faz lançamento de faltas e presença	Não faz financiamento
É plataforma de comunicação entre a administração do site da UVA e o usuário final	Não é uma plataforma EAD	Faz notificações de atualizações no sistema	
É voltado para o ensino presencial	Não é uma ferramenta de	Faz leitura de QR code para lançamento de presenças (aulas e eventos/Hora AC)	
	Não é um app voltado para anúncios	Faz exibição de notas	
	Não é um substituto do atendimento presencial	Faz exibição do calendário do semestre	
		Faz exibição de boletos	
		Faz compartilhamento de pdf	
		Faz exibição do código de barras	
		Faz atendimento via chat	
		Faz acompanhamento de protocolo de atendimento	
		Faz abertura de protocolo	
		Faz download de arquivos	

Figura 4 É-Não é-Faz-Não faz - Lean Inception Piloto

Fonte: autor.

Para concluir as atividades designadas ao primeiro dia, o objetivo do produto foi construído no mesmo formato que os afazeres anteriores; com cinco minutos para realização individual e aproximadamente 10 minutos para consolidar todas as respostas. A frase final que representou o objetivo idealizado em conjunto foi: “Trazer a portabilidade de um sistema Web para o celular, visando diminuir a carga de acessos ao site, e ter uma alta usabilidade para os usuários proporcionando novas funcionalidades com foco na comunicação eficiente. ”.

O “segundo” dia iniciou-se depois de uma pausa de aproximadamente 30 minutos após as atividades do primeiro dia, com uma das atividades mais complexas que é a descrição de personas. Personas são a “alma” do produto, pois são os usuários finais, ou seja, quem devemos impressionar de verdade. Sem elas, não é possível definir funcionalidades, pois estariam avulsas. Para criar uma persona, deve-se idealizar basicamente quem ela é, seu papel no produto/serviço final, o que ela pensa e suas dores

Uma boa quantidade de tempo foi investida nesta atividade. E um ponto interessante sobre esta é que todos fizeram juntos, todos ao mesmo tempo foram descrevendo as personas que achavam que seriam usuários do produto. A Figura 5 mostra a descrição da primeira persona criada que foi a Joana Estudante.


<p><u>Apelido e desenho</u></p> <ul style="list-style-type: none"> Joana estudante 	<p><u>Perfil</u></p> <ul style="list-style-type: none"> 23 anos Estudante de administração(6º período) Estagiária de administração Não mora com os pais Tem um namorado Não tem filhos Independente
<p><u>Comportamento</u></p> <ul style="list-style-type: none"> Estressada com a família Tem uma vida agitada Focada em objetivos profissionais Organizada Disciplinada Passa o tempo lendo livros Comparece à maioria das aulas 	<p><u>Necessidades</u></p> <ul style="list-style-type: none"> Ter acesso a diversos livros Precisa receber notificações para ajudar na organização Praticidade no acesso às informações do seu perfil na universidade Precisa emitir/compartilhar o boleto Ela precisa saber se terá aula ou não Verificar as suas notas

Figura 5 Descrição Joana Estudante - Lean Inception Piloto

Fonte: autor.

A Figura 6 demonstra as características do Yuri Feliz.


<p><u>Apelido e desenho</u></p> <ul style="list-style-type: none"> Yuri feliz 	<p><u>Perfil</u></p> <ul style="list-style-type: none"> 34 anos Formado em engenharia de produção Cursando moda Desempregado Mora com a mãe Solteiro Não tem filhos Dependente
<p><u>Comportamento</u></p> <ul style="list-style-type: none"> Ele é criativo Indisciplinado Utiliza muita redes sociais Sociável Desorganizado Ele é interessado pela área acadêmica 	<p><u>Necessidades</u></p> <ul style="list-style-type: none"> Precisa de notificações; Compartilhar notas; Ver o calendário de eventos da área de moda;

Figura 6 Descrição Yuri Feliz - Lean Inception Piloto

Fonte: autor.

E enfim a terceira persona é representada na Figura 7 como sendo a Margarida Sapucaí.


<p><u>Apelido e desenho</u></p> <ul style="list-style-type: none"> • Margarida Sapucaí 	<p><u>Perfil</u></p> <ul style="list-style-type: none"> • 58 anos • Professora de psicologia • Tem um consultório • Divorciada • Tem filhos que estudam na universidade
<p><u>Comportamento</u></p> <ul style="list-style-type: none"> • Calma/Zen • Ela descolada • Tem dificuldade com tecnologia • Gosta de ajudar os alunos • Flexível • Usa somente o necessário de tecnologia no trabalho 	<p><u>Necessidades</u></p> <ul style="list-style-type: none"> • Aplicativo que seja fácil de usar • Quer interagir com os alunos • Exibir que está por dentro na tecnologia

Figura 7 Descrição Margarida Sapucaí - Lean Inception Piloto

Fonte: autor.

Finalizando o tema criação de personas e o “segundo dia” – que foi feito no primeiro -, os envolvidos adentraram no momento de realizar a atividade do mapa de empatia, buscando sempre entender as necessidades de cada usuário. No mapa devem ser preenchidas as quatro áreas principais que são: o que eu _____ (penso, vejo, ouço, falo)? - estas áreas podem ser ajustadas conforme a necessidade do momento -, e as dores e ganhos. É uma ótima ferramenta para identificar, visualizar, classificar, explorar e entender os diferentes tipos de personas.

A mesma foi discutida e transcrita no *template* apropriado, vide Figura 8 para a primeira persona, a Figura 9 ilustrando o mapa da persona dois e a Figura 10 apresentando o mapa da Margarida.


 Joana Estudante	
<u>Penso</u> <ul style="list-style-type: none"> • Se o UVA Mobile funcionasse iria me ajudar muito • Podia ter um jeito mais fácil de acessar os livros • A comunicação da UVA é ruim • Que saco, eu preciso entrar nesse aplicativo bugado para ver minhas turmas/salas/notas 	<u>Ouço</u> <ul style="list-style-type: none"> • Eu nunca utilizei esse aplicativo • O aplicativo é bugado • O aplicativo é incompleto • O chat da Veiga não é bom • A comunicação é muito em cima da hora
<u>Faço</u> <ul style="list-style-type: none"> • Utiliza a carteirinha para ir ao cinema • Visualizar as notas • Compartilha o número do boleto para os pais 	<u>Falo</u> <ul style="list-style-type: none"> • Esse app poderia facilitar mais a minha vida • Eu podia compartilhar o boleto • Eu queria receber mensagens da universidade como: evento, aulas etc... • Não queria logar toda vez que abro o app
Dores <ul style="list-style-type: none"> • Não encontra todos os recursos que necessita no UVA Mobile; 	Ganhos <ul style="list-style-type: none"> • Otimização do tempo • Melhor organização • Facilidade de encontrar algumas as informações necessárias

Figura 8 Mapa de Empatia Joana Estudante - Lean Inception Piloto

Fonte: autor.

 Yuri Feliz	
<u>Penso</u> <ul style="list-style-type: none"> • O design desse app é horrível! • Queria poder compartilhar o boleto com a minha mãe • O app UVA Mobile poderia avisar os alunos sobre eventos e lançamentos de notas; • O app UVA Mobile poderia ter mais funções; • Não enxergo muita utilidade no app pois utilizo muito pouco e só uso para coisas específicas 	<u>Ouço</u> <ul style="list-style-type: none"> • O app UVA Mobile sobrecarrega celulares mais antigos • O app UVA Mobile é incompleto em relação ao site • Você viu o UVA Mobile? É cheio de bugs! • Eu já até desinstalei aquele aplicativo, prefiro usar o site • Nunca sei quando tem evento na veiga de moda • É muito ruim de achar o calendário de eventos no site da Veiga
<u>Faço</u> <ul style="list-style-type: none"> • Tiro print da tela que mostra minhas notas pra compartilhar com outras pessoas • Visualiza as faltas • Utilizo a carteirinha para ir ao cinema • Utilizo o material de apoio • Vejo minhas notas 	<u>Falo</u> <ul style="list-style-type: none"> • Esse app deveria ter opções de configurar e criar alertas para novas notas e eventos • Esse app é muito ruim e feio, acho que vou desinstalar • Só tenho esse app por causa da carteirinha • Corri tanto pra chegar a tempo na aula e não vai ter
<u>Dores</u> <ul style="list-style-type: none"> • Não tem calendário atualizado • Celular travando • Design e UX muito ruim 	<u>Ganhos</u> <ul style="list-style-type: none"> • Esse app não é nada intuitivo para alguém da minha idade. Carteirinha • Calendário no site da UVA

Figura 9 Mapa de Empatia Yuri Feliz - Lean Inception Piloto

Fonte: autor.


 Margarida Sapucaí	
<u>Penso</u> <ul style="list-style-type: none"> Esse app UVA Mobile não é nada intuitivo, achei difícil de utilizar A forma de comunicação com meus alunos é bem ruim Vejo muita dificuldade mas entendo a necessidade do app devido o trabalho. 	<u>Ouçô</u> <ul style="list-style-type: none"> Os alunos reclamando do app quanto a falta de recursos Meu computador quebrou ontem e eu to correndo contra o tempo para lançar a nota dos meus alunos com o computador da universidade Quase ninguém faz perguntas no Fórum "Tire suas dúvidas" nessa quarentena Fiz a chamada pelo celular ontem e não salvou, recebi uma chamada da coordenação. Nem conheço o aplicativo
<u>Faço</u> <ul style="list-style-type: none"> Lançamento das notas e faltas dos alunos no site Visualizar os horários das aulas e as turmas Uso o site para realizar minhas obrigações 	<u>Falo</u> <ul style="list-style-type: none"> Esse app poderia ter um fórum para cada turma, para que o prof e os alunos pudessem interagir fora de sala com mais facilidade. Esse UVA Mobile poderia ser aproveitado para os professores Podia ter um chat ao vivo com meus alunos caso eles queiram falar comigo Gostaria de poder mandar mais recados para os meus aluno Queria poder lembrá-los dos prazos pelo app
<u>Dores</u> <ul style="list-style-type: none"> Usabilidade Distanciamento com os alunos 	<u>Ganhos</u> <ul style="list-style-type: none"> Lançamento das notas e faltas Gerenciamento dos horários

Figura 10 Mapa de Empatia Margarida Sapucaí - Lean Inception Piloto

Fonte: autor.

Após um sábado exaustivo, cheio de ideias e execuções, as tarefas foram finalizadas para os desenvolvedores. Contudo, houve uma reunião apenas entre a PO e a facilitadora para discutirem sobre os *feedbacks* de cada uma em relação aos momentos do dia e para alinharem as expectativas em relação às atividades do próximo dia. Também, a facilitadora tirou um momento para consolidar tudo que havia sido feito no dia.

Na quarta, dia 13 de maio, iniciou-se a atividade do *brainstorming* de funcionalidades que já havia sido refletida no dia de atividade anterior. Cada funcionalidade é uma ação ou interação do usuário com o produto/serviço. Para ajudar a idealizar o que seria uma funcionalidade, basta pensar qual ação o produto deve fazer para que usuário consiga fazer o que pretende. Partindo destes princípios, os participantes elencaram as funcionalidades, avaliaram o esforço, o valor para o negócio e o valor para UX de cada uma, conforme ilustram as Figuras 11 e 12.

<ul style="list-style-type: none"> • Melhorar exibição da semana de aula <ul style="list-style-type: none"> ◦ E ◦ \$ ◦ ♥♥♥ 	<ul style="list-style-type: none"> • Permitir ocultar notas <ul style="list-style-type: none"> ◦ E ◦ \$ ◦ ♥♥ 	<ul style="list-style-type: none"> • Padronizar o layout das páginas <ul style="list-style-type: none"> ◦ EEE ◦ \$\$ ◦ ♥♥♥
<ul style="list-style-type: none"> • Melhorar a interface do módulo financeiro <ul style="list-style-type: none"> ◦ E ◦ \$\$\$ ◦ ♥♥♥ 	<ul style="list-style-type: none"> • Remover notificações inúteis <ul style="list-style-type: none"> ◦ E ◦ \$\$ ◦ ♥♥ 	<ul style="list-style-type: none"> • Otimização do desempenho nos dispositivos <ul style="list-style-type: none"> ◦ EEE ◦ \$\$\$ ◦ ♥♥♥
<ul style="list-style-type: none"> • Incluir fórum das matérias presenciais <ul style="list-style-type: none"> ◦ EEE ◦ \$\$ ◦ ♥♥ 	<ul style="list-style-type: none"> • Ter acesso a biblioteca virtual <ul style="list-style-type: none"> ◦ E ◦ \$\$ ◦ ♥♥ 	<ul style="list-style-type: none"> • Visualizar/Compartilhar boleto <ul style="list-style-type: none"> ◦ EE ◦ \$\$ ◦ ♥♥

Figura 11 Esforço, Valor Para o Negócio e Valor Para UX 1 - Lean Inception Piloto

Fonte: autor.

<ul style="list-style-type: none"> • Criar chat entre alunos e professores <ul style="list-style-type: none"> ◦ EE ◦ \$\$\$ ◦ ♥♥♥ 	<ul style="list-style-type: none"> • Visualizar calendário acadêmico <ul style="list-style-type: none"> ◦ EEE ◦ \$\$ ◦ ♥♥♥ 	<ul style="list-style-type: none"> • Automatizar notificações acadêmicas <ul style="list-style-type: none"> ◦ EE ◦ \$\$\$ ◦ ♥♥♥
<ul style="list-style-type: none"> • Visualizar calendário de eventos <ul style="list-style-type: none"> ◦ EEE ◦ \$ ◦ ♥♥♥ 	<ul style="list-style-type: none"> • Compartilhar notas <ul style="list-style-type: none"> ◦ E ◦ \$ ◦ ♥♥ 	<ul style="list-style-type: none"> • Incluir ícone de ajuda nas opções <ul style="list-style-type: none"> ◦ E ◦ \$ ◦ ♥♥♥
<ul style="list-style-type: none"> • Enviar notificações dos docentes para os discentes <ul style="list-style-type: none"> ◦ EE ◦ \$\$\$ ◦ ♥♥♥ 	<ul style="list-style-type: none"> • Lançar faltas <ul style="list-style-type: none"> ◦ EEE ◦ \$\$\$ ◦ ♥♥♥ 	<ul style="list-style-type: none"> • Lançar notas <ul style="list-style-type: none"> ◦ EEE ◦ \$\$\$ ◦ ♥♥♥
<ul style="list-style-type: none"> • Implementar layout diferente para o perfil de professor <ul style="list-style-type: none"> ◦ EEE ◦ \$\$\$ ◦ ♥♥♥ 	<ul style="list-style-type: none"> • Melhorar a exibição dos ícones e ferramentas das páginas <ul style="list-style-type: none"> ◦ E ◦ \$\$ ◦ ♥♥♥ 	

Figura 12 Esforço, Valor Para o Negócio e Valor Para UX 2 - Lean Inception Piloto

Fonte: autor.

A oitava atividade a ser feita foi gráfico do semáforo. Dois eixos que representam níveis de confiança, os quais o x mede o quão confiante os desenvolvedores estão de fazer aquela funcionalidade e o y mede o quão confiante estão da relevância da funcionalidade para o usuário final, foram desenhados; e depois marcações de níveis baixo, médio e alto foram atribuídas ao longo de cada um. Esta tarefa foi feita apenas pela facilitadora, pois não iria dar tempo para encaixá-la no terceiro dia. A mesma, foi validada pela equipe no quarto dia antes de realizarem a tarefa de descrever as jornadas dos usuários, como são apresentadas nas Figuras 14, 15 e 16.

O gráfico do semáforo – representado na Figura 13 – não sofreu muitas alterações do

time, apenas algumas dúvidas foram levantadas e as modificações sugeridas foram executadas.

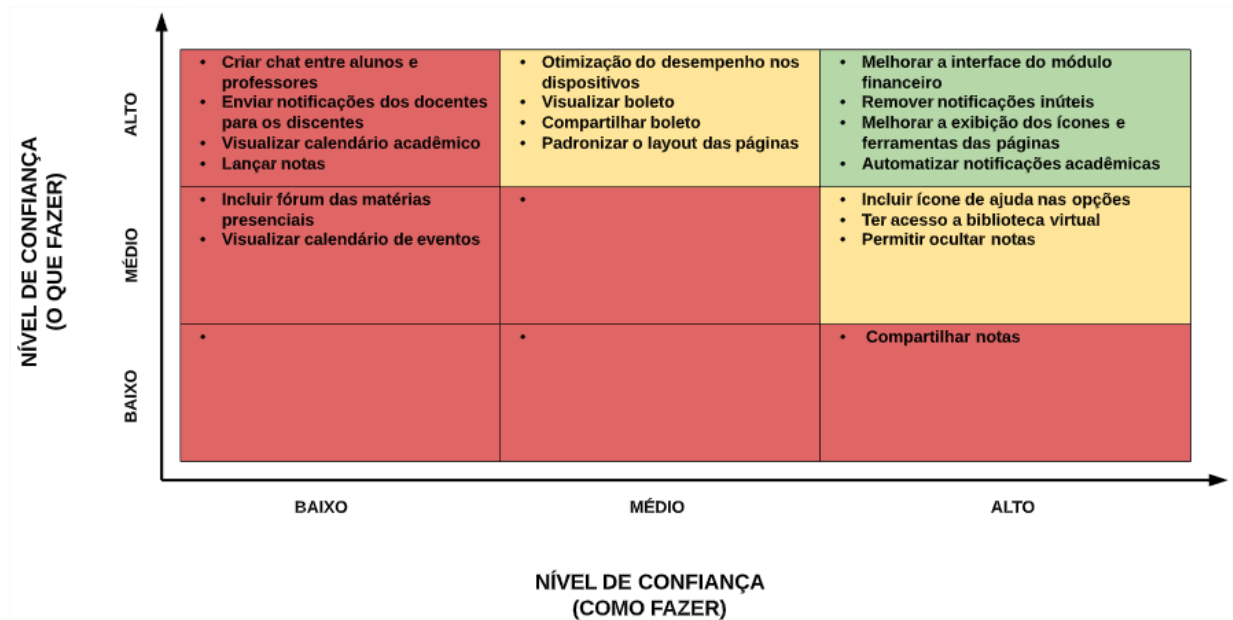


Figura 13 Gráfico do Semáforo - Lean Inception Piloto

Fonte: autor.

A jornada do usuário é um momento onde a equipe irá percorrer uma sequência de passos que os usuários darão para alcançar um determinado objetivo. Alguns desses passos explicitam a necessidade do usuário de interagir com algumas das funcionalidades criadas.

O primeiro passo desta tarefa é escolher uma das personas e definir um objetivo para ela. Para auxiliar, o facilitador apresentou a persona em um quadro, e ao lado superior esquerdo escrever seu objetivo. Após esse momento inicial, os participantes começaram a decidir como a pessoa começa seu dia, em que momento da jornada ela utiliza alguma funcionalidade e qual o desfecho do seu dia. Então, deve-se repetir este processo, passo a passo até todas as personas alcançarem seus objetivos. O nível de detalhamento das jornadas não deve ser muito alto e nem muito baixo.




Joana Estudante - Jornada 1

Passo	Funcionalidade
Acorda 6:30	
Toma café 7:00	
Abre o UVApp	Compartilhar boleto com os pais
Pega o ônibus 8:00	
Abre o UVApp	Abrir a biblioteca virtual
Guarda o celular	
Chega no trabalho 9:00	
Trabalha	
Sai do trabalho 16:00	Verificar notificação sobre aula
Abre o UVApp	
Verifica a aula do dia	Abrir exibição de calendário da semana de aula
Pega o ônibus 16:15	Verificar notificação de atividade acadêmica
Abre o UVApp	Abrir a biblioteca virtual
Chega na faculdade 17:30	

Figura 14 Jornada 1 Joana Estudante - Lean Inception Piloto

Fonte: autor.



Yuri Feliz - Jornada 1

Passo	Funcionalidade
Acorda cedo: 7:20h	
Pega o celular	
Abre o UVA app	Verificar eventos da semana de moda
Abre redes sociais	Divulgar os eventos da semana de moda
Abre o UVAapp	Verificar faltas
Pega o ônibus: 08:10	
Chega na faculdade: 08:30	
Sai da faculdade: 12:40	
Pega Onibus: 12:50	
Chega no shopping: 13:00	
Almoça (praça alimentação): 13:20	
Compra ingresso no cinema: 13:30	Visualizar carteirinha virtual da veiga
Vai para casa: 16:00	Visualizar notificação do sistema do aluno

Figura 15 Jornada 1 Yuri Feliz - Lean Inception Piloto

Fonte: autor.



Margarida Sapucaí - Jornada 1

Passo	Funcionalidade
Acorda cedo 06:00	
Toma café 06:30	
Pega metrô 06:50	
Abre o UVAapp 06:55	Envia notificação de atividade acadêmica
Guarda o celular 06:57	
Chega na faculdade 07:15	
Abre o UVAapp 07:16	Verifica a agenda semanal para visualizar qual sala é a aula
Dá aula 07:30	
Durante a aula 07:30~12:40	Lança presenças / faltas
Sai do trabalho para o almoço 12:40	
Vai para o consultório 14:30	
Chega no consultório 14:45	Fica disponível no sistema para os alunos
Atende um paciente 15:00	
Atende um paciente 16:00	
Vai para a faculdade 17:00	
Chega na faculdade 17:30	
Abre o UVAapp 17:45	Lançar notas
Navega pelo UVAapp 18:00	Verifica a agenda semanal para visualizar qual sala é a aula
Vai para casa 20:40	

Figura 16 Jornada 1 Margarida Sapucaí - Lean Inception Piloto

Fonte: autor.

A última atividade feita com a equipe inteira foi o sequenciador de atividades, apresentado na Figura 17. O objetivo desta atividade é de priorizar as funcionalidades tomando como base todas as atividades anteriores, em especial a das jornadas e de revisão técnica, de negócio e UX. Aqui, os integrantes do grupo refletiram sobre a prioridade das funcionalidades a serem criadas para oferecer um produto mínimo, porém com valor e viável. Para formar um MVP, os envolvidos escolheram as funcionalidades – em sequência – mais importantes das ondas definidas, e as agruparam. Mas, para formarem ondas – cada raia do sequenciador – eles tinham que seguir seis regras:

1. Uma onda pode conter, no máximo, três cartões.
2. Uma onda não pode conter mais de um cartão vermelho.
3. Uma onda não pode conter três cartões somente amarelos ou vermelhos.
4. A soma de esforço dos cartões não pode ultrapassar cinco “E”.

5. A soma de valor dos cartões não pode ser menos de quatro “\$” e quatro corações.

6. Se um cartão depende de outro, esse outro deve estar em alguma onda anterior.

A equipe gastou as duas horas inteiras para conseguir cumprir todas as regras e sequenciar as funcionalidades mais importantes. Por fim, o sequenciador foi finalizado.

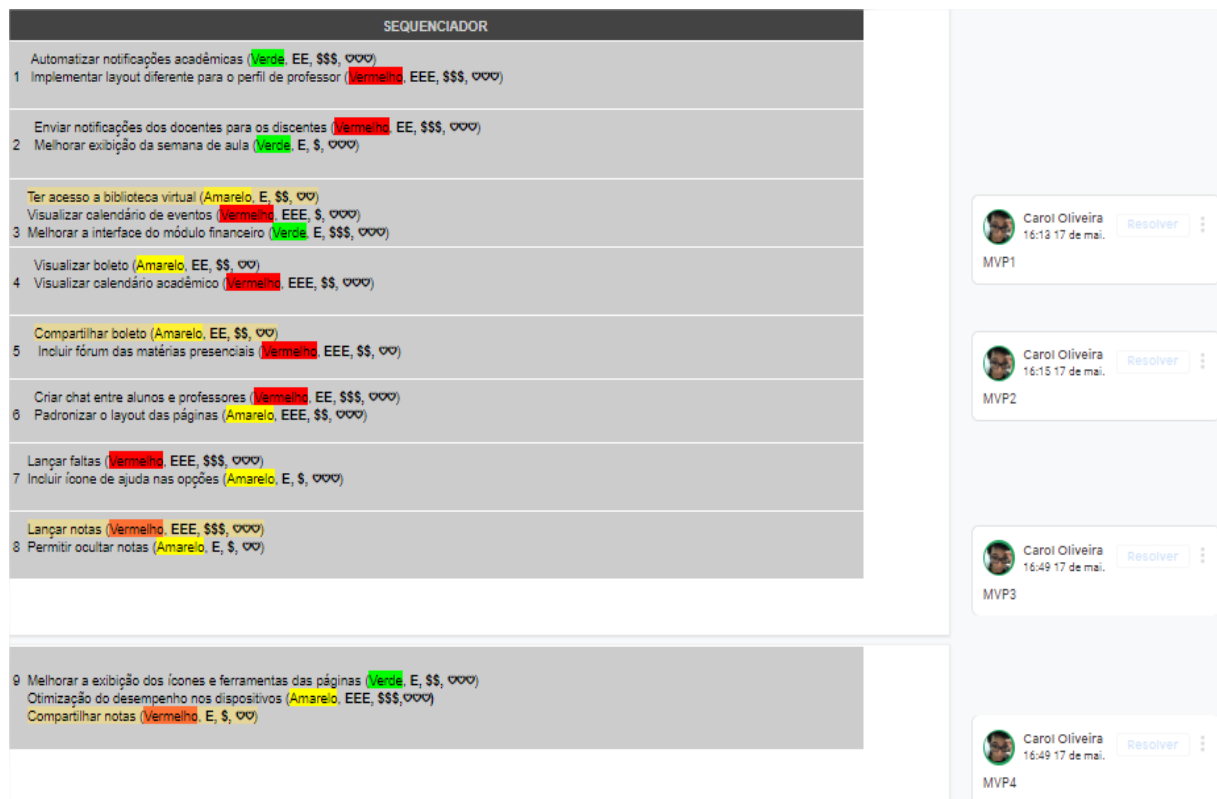


Figura 17 Sequenciador de Funcionalidades - Lean Inception Piloto

Fonte: autor.

A partir do dia 20 de maio, o restante das atividades foram feitas apenas pela facilitadora com auxílio do participante Eduardo. Como as próximas atividades são mais técnicas e específicas para quem realmente vai desenvolver, e o tempo para finalizar o presente trabalho era escasso, foi priorizado que fosse finalizado apenas pela autora do trabalho de conclusão de curso e facilitadora da *Lean Inception Piloto*.

A atividade que sequenciou as demais foi a de detalhar as funcionalidades escolhidas para agrupar o primeiro MVP, em tarefas menores. A Figura 18, representa o modelo que foi aplicado para a divisão de subtarefas de todas as funcionalidades, através daquela selecionada para desenvolver, analisar e comparar no escopo deste trabalho - F3 - Enviar notificações dos docentes para os discentes. A necessidade de dividir é porque facilita a

organização e priorização na hora de programar, e a estimativa do tempo de desenvolvimento.

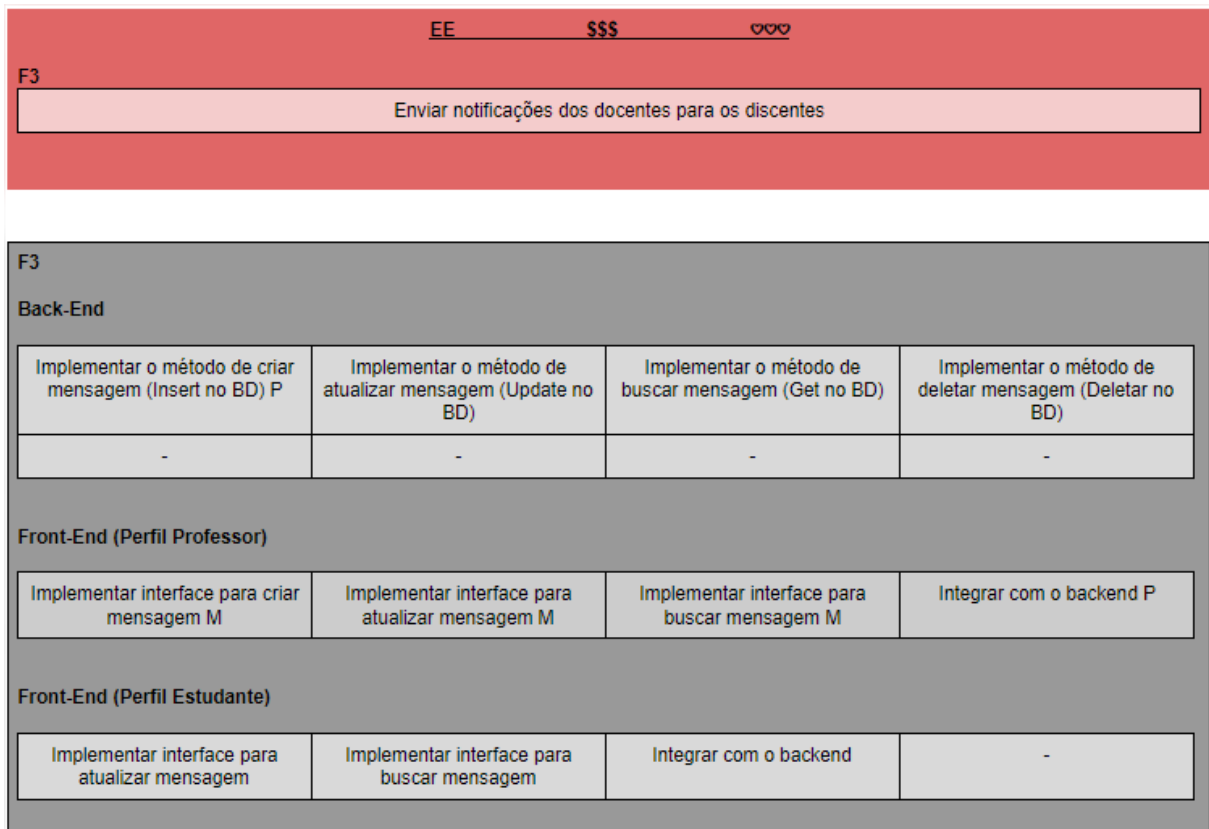


Figura 18 Detalhando Amostra de Funcionalidades - Lean Inception Piloto

Fonte: autor.

Conforme o que foi mencionado a pouco sobre a importância de dividir a funcionalidade em outras menores, a próxima atividade é ilustrada através das Figuras 19 e 20 como sendo o dimensionamento de cada tarefa criada. Este dimensionamento foi dividido nos tamanhos P para tarefas pequenas, M para as médias e G para as grandes. Também, foi feito com foco nas duas metodologias de desenvolvimento que serão utilizadas para programar a funcionalidade; o *low-code* (Outsystems) e programação tradicional (*Python*, *Angular* e *Ionic*).

F3 - Enviar notificações dos docentes para os discentes				
P	Implementar o método de criar mensagem (Insert no BD)	Implementar o método de atualizar mensagem (Update no BD)	Implementar o método de deletar mensagem (Deletar no BD)	Integrar frontend com o backend (perfil professor)
	Integrar frontend com o backend (perfil estudante)	-	-	-
M	Implementar interface para criar mensagem (perfil professor)	Implementar interface para atualizar mensagem (perfil professor)	Implementar interface para buscar mensagem (perfil professor)	Implementar interface para criar mensagem (perfil estudante)
	Implementar interface para criar mensagem (perfil estudante)	Implementar o método de buscar mensagem (Get no BD)	-	-
G	-			

Figura 19 Dimensionamento de tarefas Outsystems - Lean Inception Piloto

Fonte: autor.

F3 - Enviar notificações dos docentes para os discentes				
P	Implementar o método de criar mensagem (Insert no BD)	Implementar o método de atualizar mensagem (Update no BD)	Implementar o método de deletar mensagem (Deletar no BD)	Integrar frontend com o backend (perfil professor)
	Integrar frontend com o backend (perfil estudante)	-	-	-
M	Implementar interface para criar mensagem (perfil professor)	Implementar interface para atualizar mensagem (perfil professor)	Implementar interface para buscar mensagem (perfil professor)	Implementar interface para criar mensagem (perfil estudante)
	Implementar interface para criar mensagem (perfil estudante)	Implementar o método de buscar mensagem (Get no BD)	-	-
G	-			

Figura 20 Dimensionamento de tarefas Python, Ionic e Angular - Lean Inception Piloto

Fonte: autor.

Tirar a média do custo e do tempo é o próximo passo antes de montar o Canvas. Conforme ilustrado na Figura 21 e 22, foi definido uma estimativa de tempo para tarefas de cada funcionalidade da primeira onda. Essa estimativa é baseada no dimensionamento de tarefas executado na tarefa anterior. Depois, foram contabilizadas a quantidade de tarefas associadas cada uma das medidas e multiplicadas pela estimativa de tempo referente à mesma.

>> Tarefas pequenas: 10 minutos
>> Tarefas médias: 20 minutos
>> Tarefas grandes: 30 minutos

Funcionalidade	Tempo	Total/Funcionalidade	Total
Automatizar notificações acadêmicas	P: 00 x (10 minutos) = 00 minutos M: 01 x (20 minutos) = 20 minutos G: 01 x (30 minutos) = 30 minutos	50 minutos	450 minutos/ 7 horas e 30 minutos
Implementar layout diferente para o perfil de professor	P: 01 x (10 minutos) = 10 minutos M: 01 x (20 minutos) = 20 minutos G: 01 x (30 minutos) = 30 minutos	60 minutos	
Enviar notificações dos docentes para discentes	P: 05 x (10 minutos) = 50 minutos M: 06 x (20 minutos) = 120 minutos G: 00 x (30 minutos) = 00 minutos	170 minutos	
Melhorar exibição da semana de aula	P: 01 x (10 minutos) = 10 minutos M: 02 x (20 minutos) = 40 minutos G: 00 x (30 minutos) = 00 minutos	50 minutos	
Ter acesso a biblioteca virtual	P: 02 x (10 minutos) = 20 minutos M: 00 x (20 minutos) = 00 minutos G: 00 x (30 minutos) = 00 minutos	20 minutos	
Visualizar calendário de eventos	P: 01 x (10 minutos) = 10 minutos M: 02 x (20 minutos) = 40 minutos G: 00 x (30 minutos) = 00 minutos	50 minutos	
Melhorar interface do módulo financeiro	P: 01 x (10 minutos) = 10 minutos M: 02 x (20 minutos) = 40 minutos G: 00 x (30 minutos) = 00 minutos	50 minutos	

Figura 21 Entendendo custo e tempo e tirando a média Outsystems - Lean Inception Piloto

Fonte: autor.

>> Tarefas pequenas: 03 horas
>> Tarefas médias: 05 horas
>> Tarefas grandes: 24 horas

Funcionalidade	Tempo	Total/Funcionalidade	Total
Automatizar notificações acadêmicas	P: 00 x (03 horas) = 00 horas M: 01 x (05 horas) = 05 horas G: 01 x (24 horas) = 24 horas	29 horas	151 horas/ 6 dias e 7 horas
Implementar layout diferente para o perfil de professor	P: 01 x (03 horas) = 03 horas M: 01 x (05 horas) = 05 horas G: 01 x (24 horas) = 24 horas	32 horas	
Enviar notificações dos docentes para discentes	P: 05 x (03 horas) = 15 horas M: 06 x (05 horas) = 30 horas G: 00 x (24 horas) = 00 horas	45 horas	
Melhorar exibição da semana de aula	P: 01 x (03 horas) = 03 horas M: 02 x (05 horas) = 10 horas G: 00 x (24 horas) = 00 horas	13 horas	
Ter acesso a biblioteca virtual	P: 02 x (03 horas) = 06 horas M: 00 x (05 horas) = 00 horas G: 00 x (24 horas) = 00 horas	06 horas	
Visualizar calendário de eventos	P: 01 x (03 horas) = 03 horas M: 02 x (05 horas) = 10 horas G: 00 x (24 horas) = 00 horas	13 horas	
Melhorar interface do módulo financeiro	P: 01 x (03 horas) = 03 horas M: 02 x (05 horas) = 10 horas G: 00 x (24 horas) = 00 horas	13 horas	

Figura 22 Entendendo custo e tempo e tirando a média Python, Ionic e Angular - Lean Inception Piloto

Fonte: autor.

Finalmente, depois de todas as etapas é possível montar o Canvas em um intervalo de tempo estimado entre 20 a 60 minutos. Todas as atividades até agora foram executadas com a

finalidade de facilitar o preenchimento do Canvas, que por sua vez, é um compilado de tudo que foi visto anteriormente neste capítulo – vide Figuras 23 e 24 - e o direcionamento final antes de começar a etapa de desenvolvimento na prática.

Personas Segmentadas	Proposta do MVP	Resultado esperado
Joana Estudante Yuri Feliz Margarida Sapucaí	Validar se os alunos e professores usariam o UVApp	De 1 a 5 mil downloads Avaliação média de 4 a 5 estrelas De 100 a 500 cliques em notificações
Jornadas	Funcionalidades	Métricas para validar as hipóteses de negócio
Joana Estudante compartilha boleto, estuda, verifica notificação e calendário Yuri Feliz verifica calendário de eventos e compartilha, verifica faltas, visualiza carteirinha e notificações Margarida Sapucaí envia notificações acadêmica, verifica agenda semanal, fica disponível no chat, lança faltas e notas	Automatizar notificações acadêmicas Implementar layout diferente para o perfil de professor Enviar notificações dos docentes para discentes Melhorar exibição da semana de aula Ter acesso a biblioteca virtual Visualizar calendário de eventos Melhorar interface do módulo financeiro	Número de download do aplicativo Número de notificações enviadas pelo professor Número de notificações visualizadas pelos estudantes
	Custo & Cronograma	
	7:30 horas considerando 1 desenvolvedor R\$ 0,00	

Figura 23 Canvas Outsystems - Lean Inception Piloto

Fonte: autor.

Personas Segmentadas	Proposta do MVP	Resultado esperado
Joana Estudante Yuri Feliz Margarida Sapucaí	Validar se os alunos e professores usariam o UVApp	De 1 a 5 mil downloads Avaliação média de 4 a 5 estrelas De 100 a 500 cliques em notificações
Jornadas	Funcionalidades	Métricas para validar as hipóteses de negócio
Joana Estudante compartilha boleto, estuda, verifica notificação e calendário Yuri Feliz verifica calendário de eventos e compartilha, verifica faltas, visualiza carteirinha e notificações Margarida Sapucaí envia notificações acadêmica, verifica agenda semanal, fica disponível no chat, lança faltas e notas	Automatizar notificações acadêmicas Implementar layout diferente para o perfil de professor Enviar notificações dos docentes para discentes Melhorar exibição da semana de aula Ter acesso a biblioteca virtual Visualizar calendário de eventos Melhorar interface do módulo financeiro	Número de download do aplicativo Número de notificações enviadas pelo professor Número de notificações visualizadas pelos estudantes
	Custo & Cronograma	
	6 dias e 7 horas considerando 1 desenvolvedor R\$ 0,00	

Figura 24 Canvas Python, Ionic e Angular - Lean Inception Piloto

Fonte: autor.

Na próxima seção, um CRUD (*Create, Retrieve, Update, Delete*) da funcionalidade “Enviar notificações dos docentes para os discentes” será implementado através da plataforma *low-code Outsystems*.

4.2 DESENVOLVIMENTO COM METODOLOGIA LOW-CODE

Nesta seção, o desenvolvimento do código de implementação da aplicação se iniciará, porém, para validar a hipótese das metodologias e acrescentar outros itens como trabalhos futuros, será limitado apenas a um CRUD (*Create, Retrieve, Update, Delete*) da funcionalidade “Enviar notificações dos docentes para os discentes”. Devido ao custo benefício, a plataforma escolhida para a implementação foi o *Outsystems*. É uma ferramenta que possui uma versão gratuita para desenvolvimento e testes, e a autora do trabalho de conclusão de curso já havia feito um curso básico de um dos vários módulos do *Outsystems* – o módulo *Developing Web Apps (Outsystems 11)*.

Primeiramente, uma aplicação – representada através da Figura 25 - foi criada com dois módulos. O UVApp_CORE para representar o núcleo do programa, contendo a modelagem e a base de dados. E o UVApp, que conectado ao módulo citado anteriormente

configura o layout e ações do cliente.

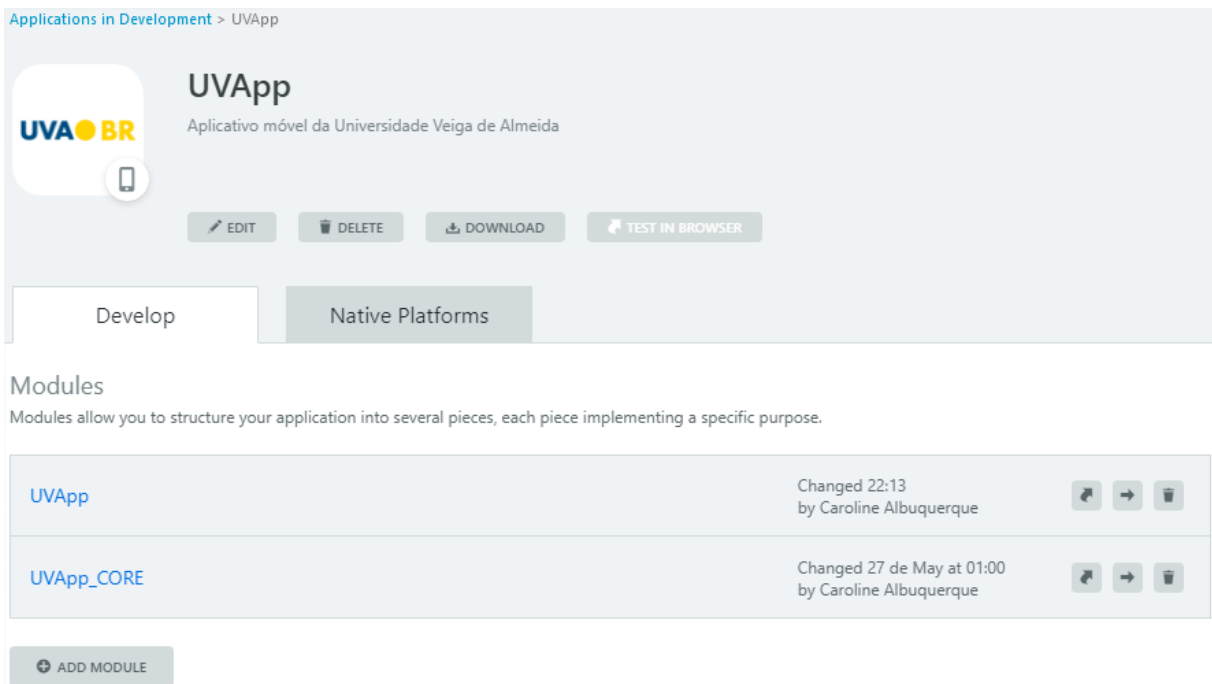


Figura 25 Aplicação UVApp e seus dois módulos – Outsystems

Fonte: autor.

Depois, no módulo UVApp_CORE, foram criadas duas tabelas – vide Figura 25:

- *Class*

A tabela *Class* contém as colunas *Id* e *Turma*. A segunda coluna se alimenta de uma estrutura de Excel vinculada a ela. Na estrutura existe uma única coluna cujo cabeçalho 'Turma' precede todas as turmas cadastradas.

- *Notification*

Nesta tabela – que é a principal desse trabalho – as colunas *Id*, *Title*, *Text*, *DateETime*, *ClassId* e *UserId* foram criadas. As colunas *ClassId* e *UserId* são chaves estrangeiras, e são referenciadas através do campo *Data Type* – cujo define o tipo de dados - como sendo respectivamente *Class Identifier* e *User Identifier*.

É conveniente ressaltar que *Outsystems*, por padrão, já vem com a tabela *UserId* criada, e que para criar as tabelas basta clicar com o botão direito do mouse em cima de *Database* – que se encontra dentro do diretório *Entities* – e escolher a opção *Add Entity* (adicionar entidade). E automaticamente, ao criá-las, a coluna *Id* é criada também junto a componentes de ação em relação à base de dados. Também, é preciso que o campo *Public* (da

configuração das tabelas) esteja *setado* como ‘Yes’, para que se mostrem visíveis a outros módulos, como mostram as Figuras 26 e 27.

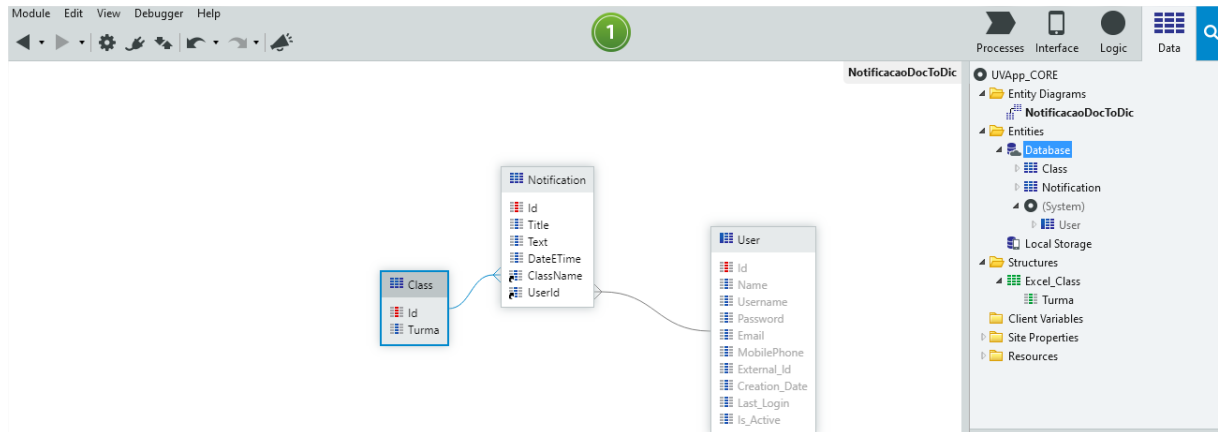


Figura 26 Módulo UVApp_CORE da aplicação UVApp – Outsystems

Fonte: autor.

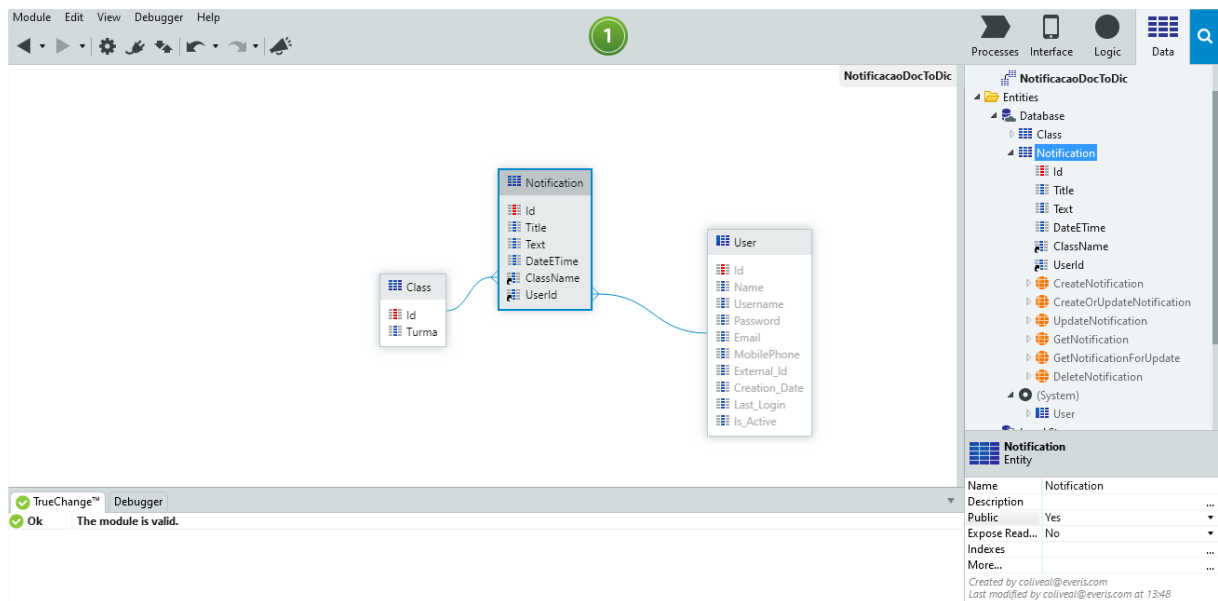


Figura 27 Campo Public setado como ‘Yes’ - Outsystems

Fonte: autor.

Seguindo adiante, o módulo UVApp_CORE foi conectado ao módulo UVApp. Para isso, foi necessário acessar o módulo UVApp, e clicar no ícone de gerenciador de dependências na parte superior da janela. Então, conforme ilustrado através da Figura 28, o módulo UVApp_CORE foi encontrado e toda sua estrutura foi selecionada antes de clicar no botão *Apply* para aplicar a conexão entre os módulos.

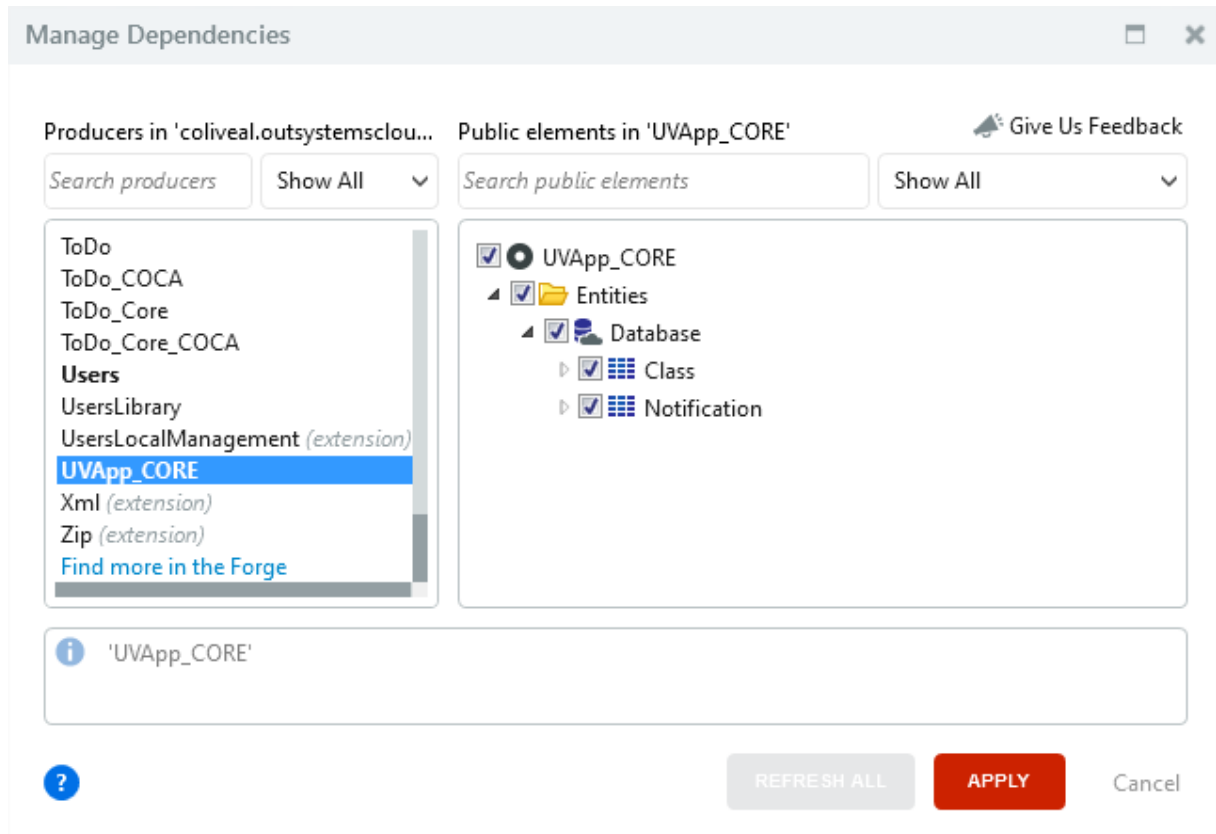


Figura 28 Gerenciador de dependências – Outsystems

Fonte: autor.

O próximo passo, de criar a tela de notificações foi bem simples pois esta ferramenta já nos oferece um *layout* pré-definido. Ainda no módulo UVApp, na aba *Interface* já existia uma tela principal *Main* que teve seu nome alterado para *Notification*.

Para desenvolver a parte do *Create* do CRUD, foi necessário arrastar o *widget Icon* para o lado superior direito do *layout* da tela e escolher o ícone *plus*, depois com o botão direito, clicar em cima do ícone e escolher a opção *Link To* e, no caso deste trabalho *linkar* a uma nova tela chamada *NotificationDetail* que irá conter todos os campos necessários para o preenchimento e criação da notificação. Assim, na próxima tela, bastou arrastar o *widget Form* para o centro do *layout* e depois arrastar a estrutura (encontrada na aba *Interface*) *GetNotificationById* para dentro do formulário. O resultado é que automaticamente o formulário de preenchimento das informações referentes à mensagem da notificação aparecerá formatado. Para finalizar esta parte, um botão, de nome Salvar, foi criado em baixo do formulário da mesma forma que os outros componentes, arrastando e soltando. Este botão, nas suas propriedades de Eventos, teve a configuração para que a qualquer clique ele invocasse uma *Client Action* que recebe o *Id* do usuário cria a notificação no banco. As ilustrações das Figuras de 29 a 32 representam de forma mais clara esse desenvolvimento.

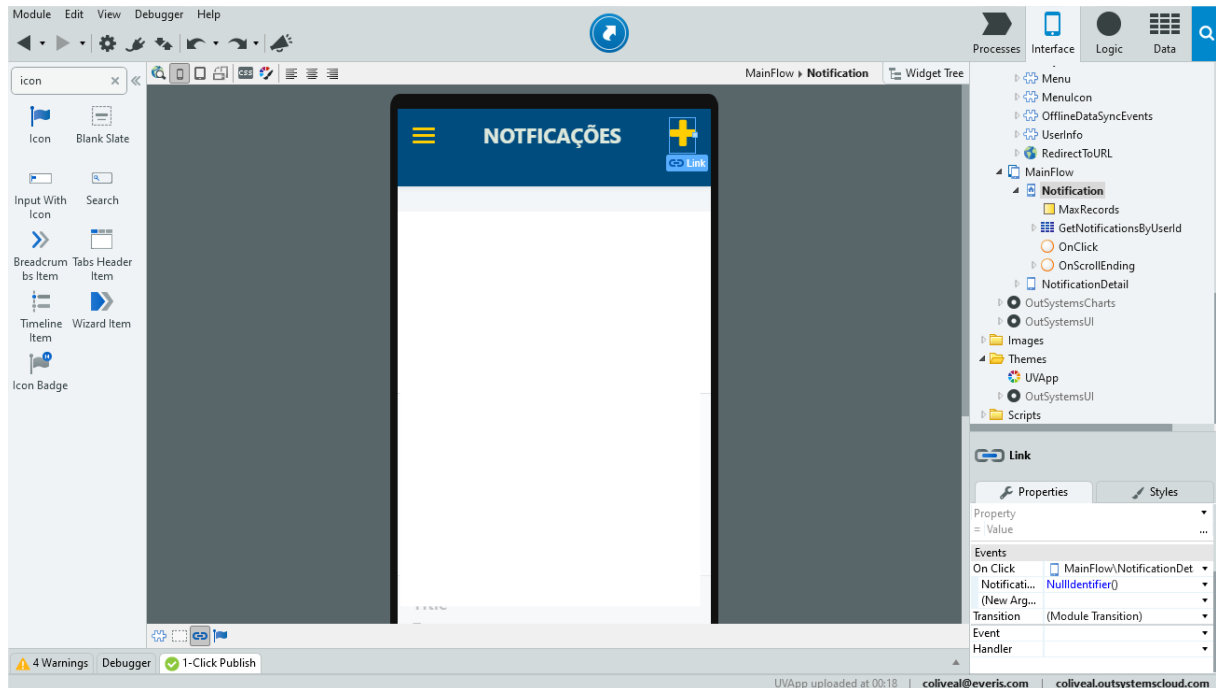


Figura 29 Criação do ícone plus - Outsystems

Fonte: autor.

Ao criar o *link*, foi passado como parâmetro para o evento *On Click* um identificador nulo, pois como trata-se da criação de uma notificação, esta não deve ter um identificador prévio. Também, é necessário para que a *action* (*CreateOrUpdateNotification*) criadora da notificação no banco identifique que é uma criação e não uma atualização, visto que a mesma é usada para as duas finalidades.

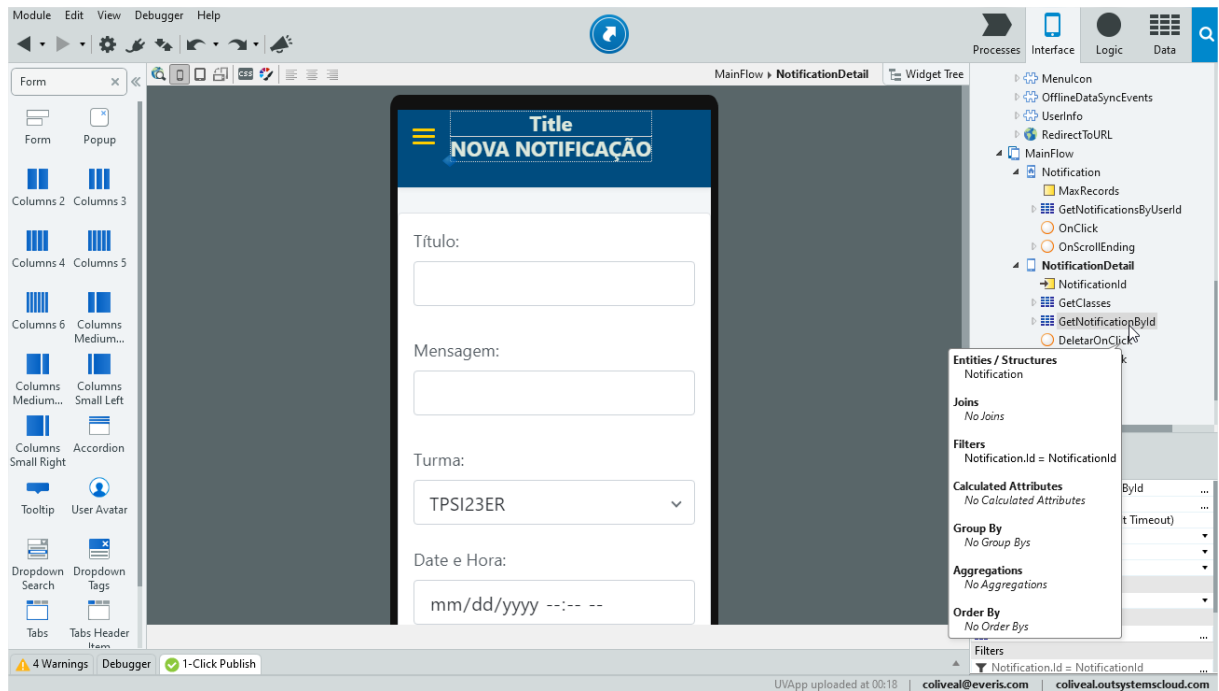


Figura 30 Criação da tela NotificationDetail - Outsystems

Fonte: autor.

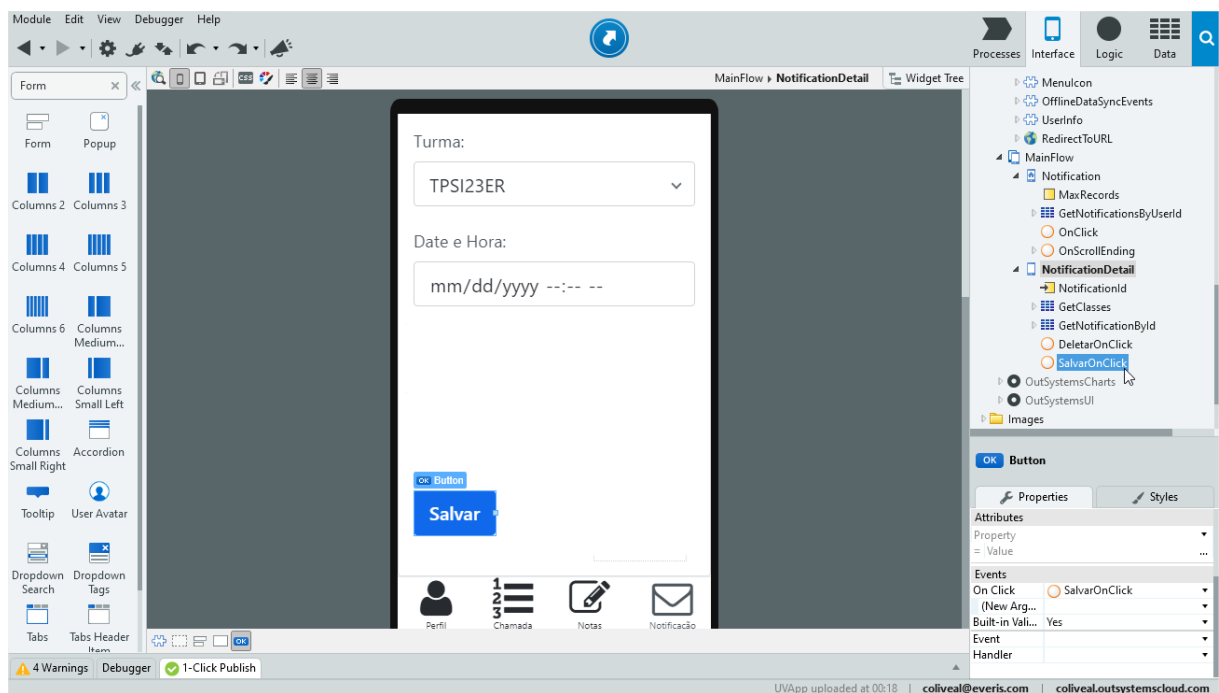


Figura 31 Criação do botão salvar - Outsystems

Fonte: autor.

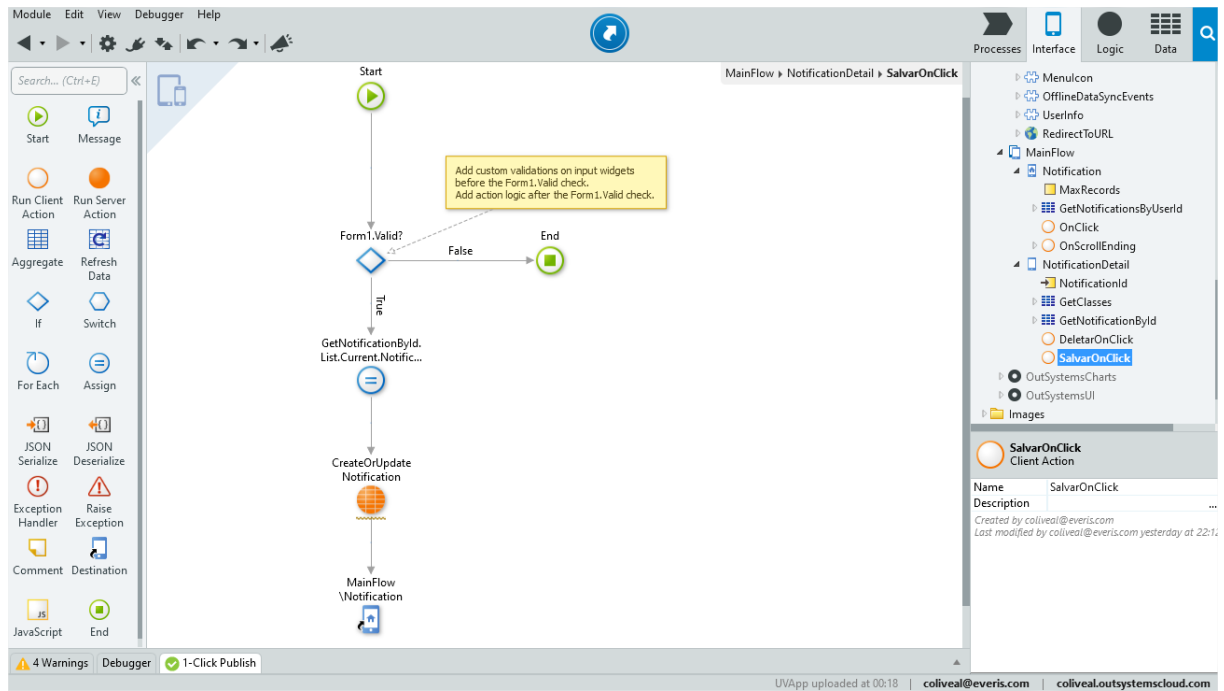


Figura 32 Client Action SalvarOnClick - Outsystems

Fonte: autor.

Depois de criar notificação, foi desenvolvido a parte *Retrieve* do CRUD, a qual foi bem simples também. Neste momento, bastou apenas voltar para a tela principal *Notifications* e arrastar o componente *Agregate GetNotificationsByUserId* (que agrega as informações da base de dados) para o centro da tela, e automaticamente as informações foram adicionadas à tela em formato de itens de uma lista. A Figura 33 ilustra com clareza como se consolidou esse desenvolvimento.

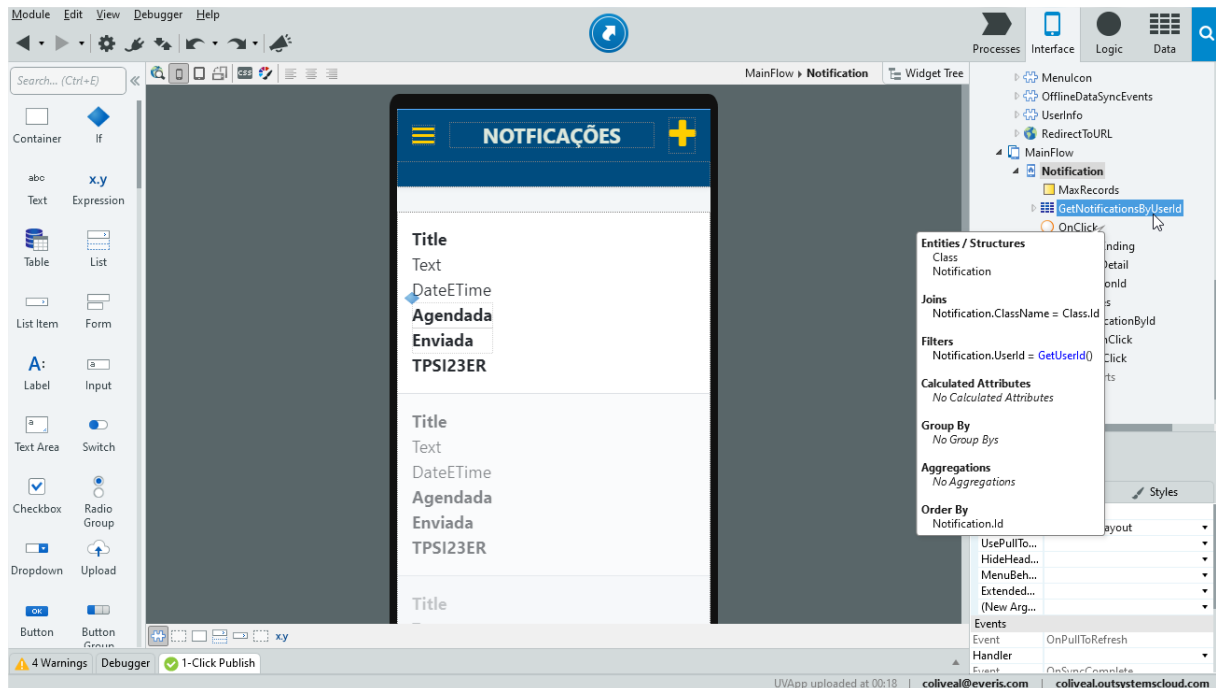


Figura 33 Criação do Retrieve - Outsystems

Fonte: autor.

Nota-se que abaixo da informação *DateETime* existem dois *status*: *Agendada* e *Enviada*. Esses *status* estão enclausurados em um *If* cuja condição é “Se a data e a hora forem maior que a atual...”; se a condição for verdadeira, a mensagem será marcada como *agendada*, do contrário, *enviada*.

Após o desenvolvimento da criação e recuperação de dados, a terceira parte do CRUD – o *Update* – foi construído facilmente, visto que a *Action* para criar a notificação, serve também para atualizar; a diferença é que ao invés de passar como parâmetro um identificador nulo, será passado o identificador da notificação escolhida pelo usuário. Sendo assim, ainda na tela de *Notification*, o *ListItem1* (primeiro item da lista) foi selecionado e em sua propriedade de eventos foi associado o *Client Action Onclick* ao evento que possui o mesmo nome. Dentro dessa *Client Action* foi trocado o componente de finalização *End* por um *Destination*, o qual destina (após o clique) o usuário para a tela de *NotificationDetail*. Na página decorrente, o processo é similar ao de criação, só que ao invés de preencher os campos, os mesmos já estarão preenchidos com as informações da notificação e preparados para serem atualizados. Ao final, basta clicar no botão salvar que o *Id* da notificação será enviado para a *Action* de *CreateOrUpdateNotification*, e a mesma identificará que é uma atualização de dados. As figuras 34 e 35 apresentam o processo ocorrido nesta etapa.

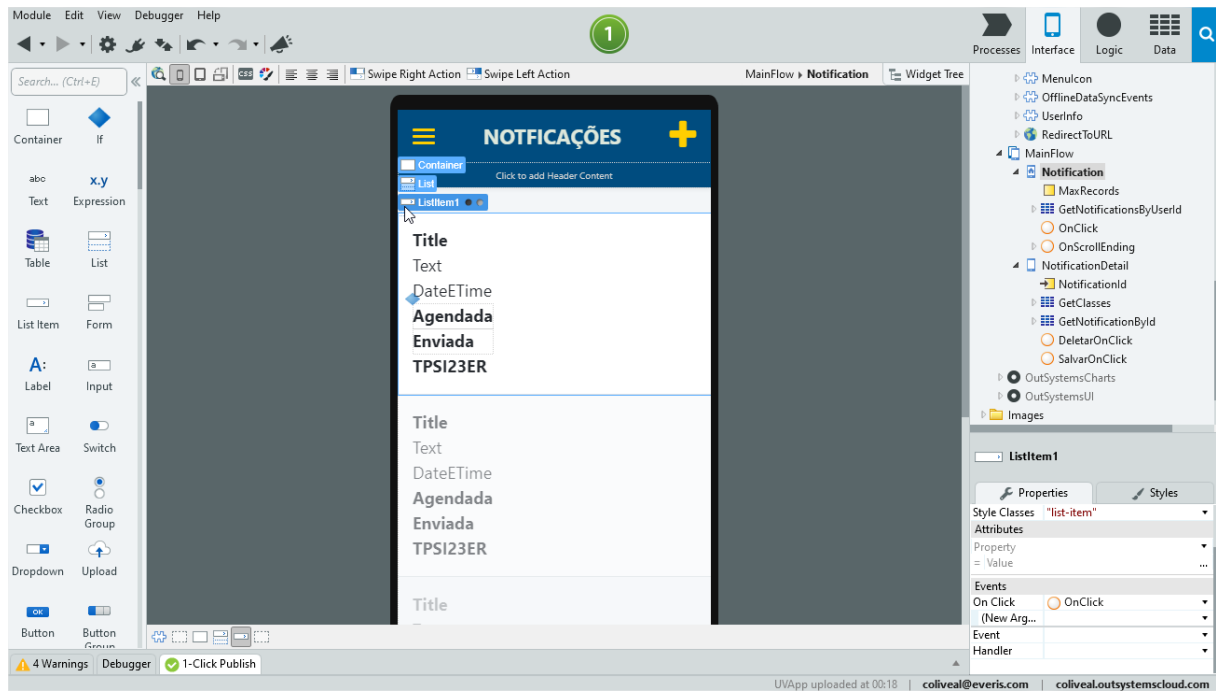


Figura 34 Associando Client Action ao item da lista - Outsystems

Fonte: autor.

Esta associação, apesar de ser dada ao primeiro item da lista, é replicada a todos eles. Pois, ao escolher o ListItem1, sugere-se (no Outsystems) que se o usuário clicar em qualquer espaço demarcado por um item, a ação ocorrerá. Neste caso a ação é ir para a próxima página.

Figura 35 Atualizando um item da lista de notificações - Outsystems

Fonte: autor.

A Figura 35 retrata um cenário testado no browser, pois na ferramenta não é possível ter cenários se não os de desenvolvimento.

Por fim, a última parte do CRUD - o *Delete* – foi desenvolvida. Para isso, na tela *NotificationDetail*, foi necessária a criação de uma condição *If* que valida se o *Id* da notificação é nulo e se a data e hora da notificação é igual a atual (esta última é para garantir apenas deleção de notificações agendadas), em caso positivo o botão fica invisível, do contrário torna-se visível. Dentro da parte verdadeira do *If* nada foi colocado, por outro lado na parte falsa um *button widget* (ou ferramenta botão) foi arrastado e configurado. Essa configuração se estendeu de forma que quando o usuário clica no componente o *DeletarOnAction* é acionado através do evento de *OnClick*. A *Client Action* de deleção de dados, é configurada para verificar o usuário que está executando a ação e através da *Action DeleteNotification*, deletar a notificação em questão. As Figuras 36 e 37 representam o processo de deleção.

Figura 36 Notificação agendada - Outsystems

Fonte: autor.

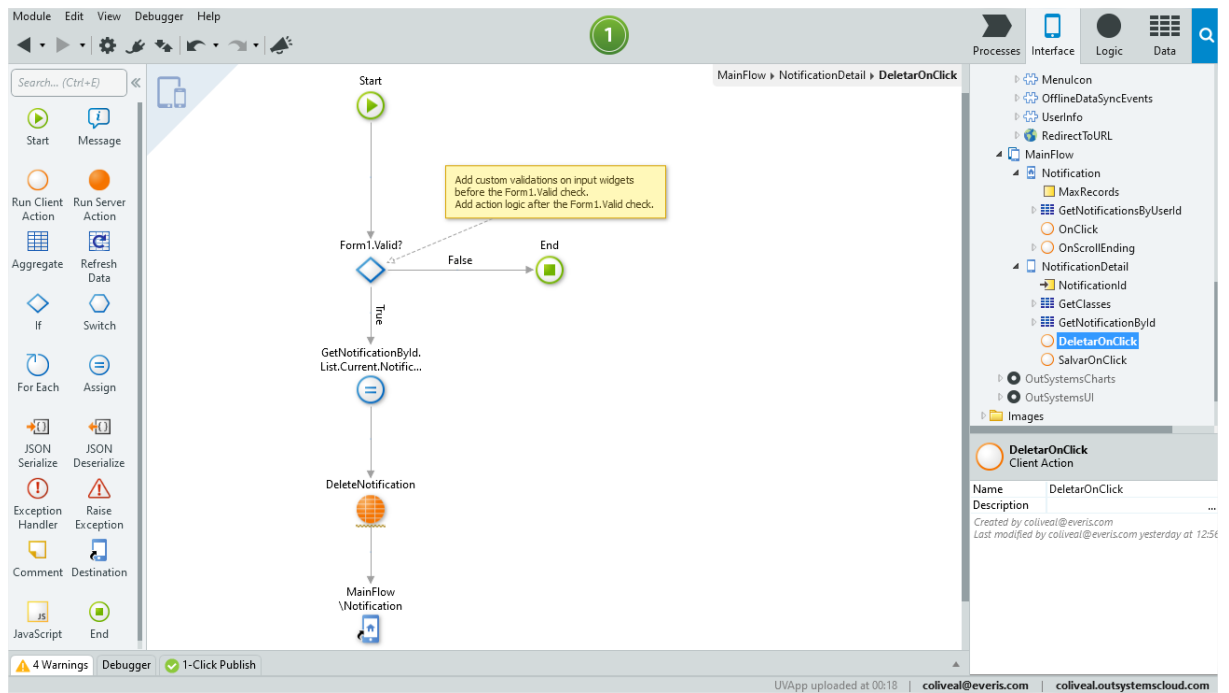


Figura 37 Client Action DeletarOnClick - Outsystems

Fonte: autor.

Com isso, foi possível finalizar o CRUD sugerido para avaliação da curva de aprendizado através do Outsystems.

A partir da Figura 38 até a 44 é possível visualizar um teste para cada parte do CRUD através da compilação e atualização no browser.

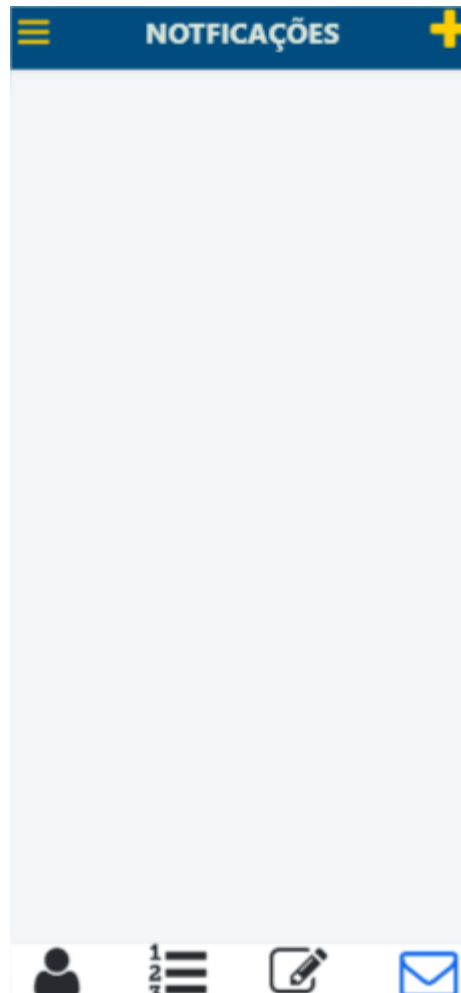


Figura 38 Tela de notificação - Outsystems

Fonte: autor.

Na Figura 38 é possível ver a tela inicial sem nenhuma alteração.

Aula cancelada

Título: *

Aula cancelada

Mensagem: *

Pessoal, tá com trânsito na ponte! |

Turma:

TPSID0F4

Date e Hora: *

25/05/2020 18:41

✓
Enviada

Salvar

1 2 3

Figura 39 Criar notificação - Outsystems
Fonte: autor.

O objetivo da Figura 39 é apresentar a criação de uma nova funcionalidade após o clique no ícone “mais”.

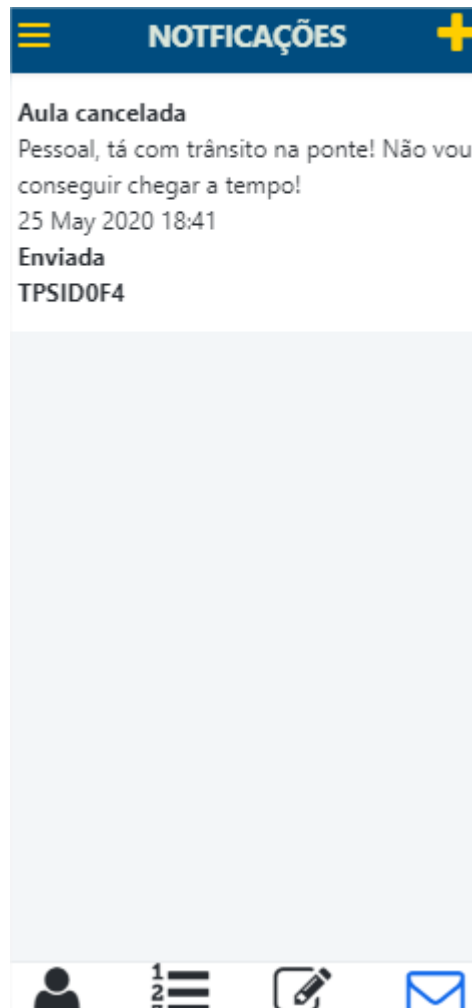


Figura 40 Tela de notificação - Outsystems

Fonte: autor.

A Figura 40 mostra que ao clicar no botão “salvar da tela anterior”, a notificação é salva automaticamente e atualizada na página principal de notificações.

The screenshot shows a mobile application interface for updating a notification. At the top, a blue header bar contains a hamburger menu icon and the text 'Aula cancelada'. Below this, the form has four input fields: 'Título: *' with the value 'Aula cancelada', 'Mensagem: *' with the value 'Pessoal, tá com trânsito na ponte! |', 'Turma:' with a dropdown menu showing 'TPSID0F4', and 'Date e Hora: *' with the value '25/06/2020 18:41' and a calendar icon. Below the date field is a calendar icon with a checkmark and the text 'Agendada'. At the bottom of the form are two buttons: 'Salvar' (blue) and 'Deletar' (blue outline). The bottom of the screen features a navigation bar with four icons: a person, a list with numbers 1, 2, 3, a pencil, and an envelope.

Figura 41 Atualização de notificação - Outsystems

Fonte: autor.

Caso seja preciso editar uma notificação é possível clicar em cima da mesma – na página inicial – que o *Outsystems* automaticamente redireciona para a página de atualização. Após alterar, basta clicar em salvar.

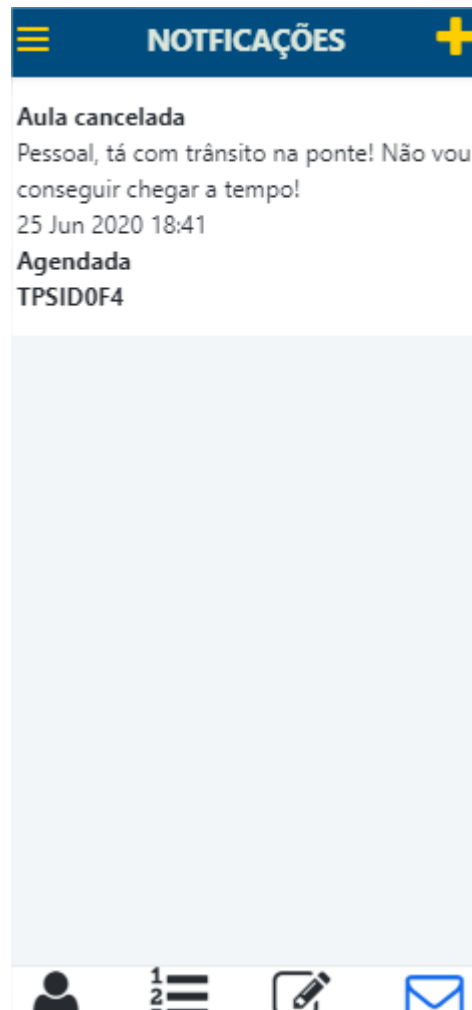
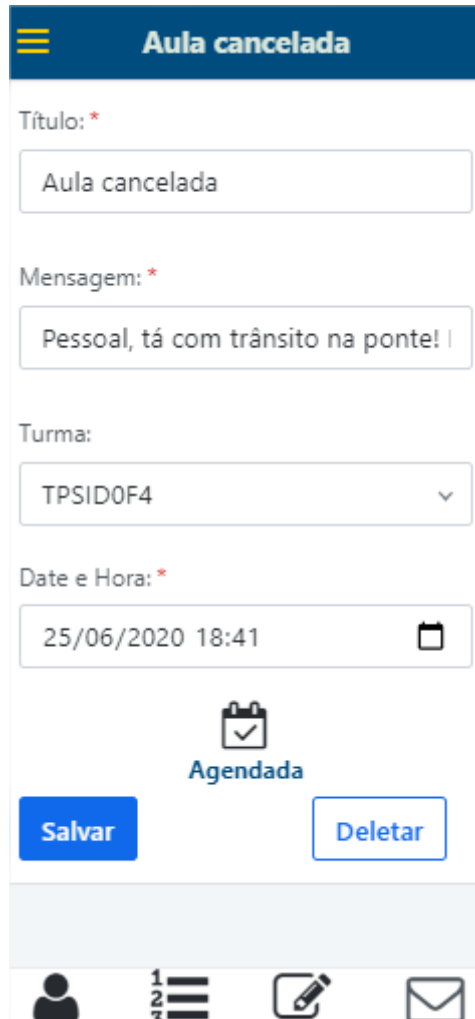


Figura 42 Tela de notificação - Outsystems

Fonte: autor.

A Figura 42 permite a visualização da modificação da data feita anteriormente.



Aula cancelada

Título: *

Aula cancelada

Mensagem: *

Pessoal, tá com trânsito na ponte! |

Turma:

TPSID0F4

Date e Hora: *

25/06/2020 18:41

Agendada

Salvar Deletar

Figura 43 Deleção de notificação - Outsystems
Fonte: autor.

Caso deseje-se deletar a notificação basta clicar em cima da mesma, e ao ser redirecionado para a página de atualização, clicar no botão “deletar”.

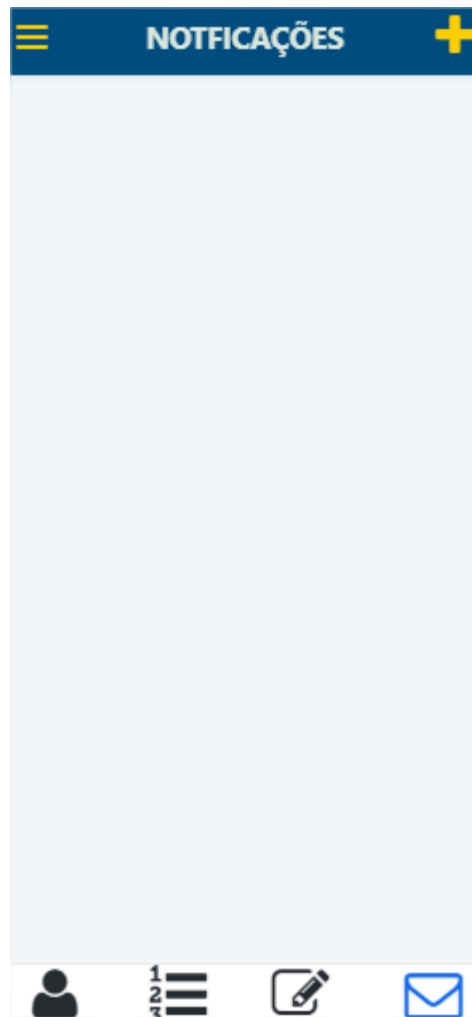


Figura 44 Tela de notificação - Outsystems

Fonte: autor.

Ao deletar, a base de dados e o *front-end* são atualizados automaticamente, e o registro não é mais visualizado.

Na próxima seção, esta mesma funcionalidade será desenvolvida e apresentada na metodologia tradicional utilizando o *framework Flask* com linguagem *Python* para o *back-end* e o *framework Ionic* com *Angular*, que é uma linguagem baseada em *Typescript* para o *front-end*.

4.3 DESENVOLVIMENTO COM METODOLOGIA TRADICIONAL

Diferentemente da plataforma utilizada na subseção anterior, onde a plataforma *low-code* compreende os módulos do banco de dados, do *back-end* e do *front-end* em uma única aplicação, na programação tradicional, é necessário codificar cada um desses módulos separadamente. Fazendo referência ao desenvolvimento da Funcionalidade 3 – proposta neste

trabalho -, foi preciso criar uma tabela em um banco de dados, depois desenvolver a parte do *back-end* (cujo objetivo é ser um servidor de dados para disponibilizá-los para o *front-end*), e por fim, o codificar o *front-end* para criar uma *interface* com o usuário e designar suas solicitações para o servidor.

Assim sendo, esta subseção será dividida para explicar cada módulo, e ao final compilar todos em um resultado final.

4.3.1 Banco de dados

Para a criação de uma tabela responsável por armazenar as notificações e seus respectivos atributos agregados, foi escolhido o *MySQL Workbench* para administrá-la. O *MySQL Workbench* é um sistema que possui um único ambiente de desenvolvimento para o sistema de banco de dados *MySQL*, e também integra desenvolvimento, administração, design, criação e manutenção de SQL através de uma interface de design de banco de dados. Ou seja, esta ferramenta permite a criação de banco de dados através de um desenvolvimento com ou sem código. A Figura 45 permite a visualização do diagrama de entidade de relacionamento utilizado na criação da base de dados.

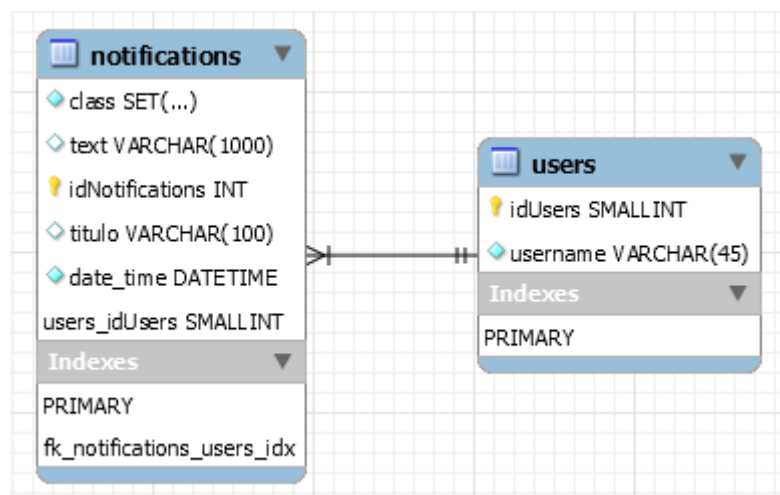


Figura 45 Diagrama de entidade de relacionamento *Notifications*

Fonte: autor

No diagrama estão dispostas, na tabela *notifications*, as colunas *class* do tipo *SET* porque serão inseridas turmas pré-definidas no banco, *text* do tipo *VARCHAR* onde será armazenado o texto da notificação, *idNotifications* do tipo *INT* que será o identificador da notificação – auto incrementado -, o título do tipo *VARCHAR* onde será armazenado o título da notificação, *date_time* do tipo *DATETIME* onde será armazenada a data de envio ou de agendamento, e *users_idUsers* do tipo *SMALLINT* que é chave estrangeira e liga as tabelas *notifications* e *users*. Na tabela *users* foram criadas as colunas *idUsers* do tipo *SMALLINT*

que será o identificador do usuário – auto incrementado -, e *username* do tipo *VARCHAR* onde será armazenado o nome do usuário. A cardinalidade é de grau 1:n, ou seja, um para muitos, onde cada usuários poderá ter muitas notificação associadas ao seu identificador.

Da Figura 46 até a 49 pode-se observar o processo de criação da tabela *notifications*.

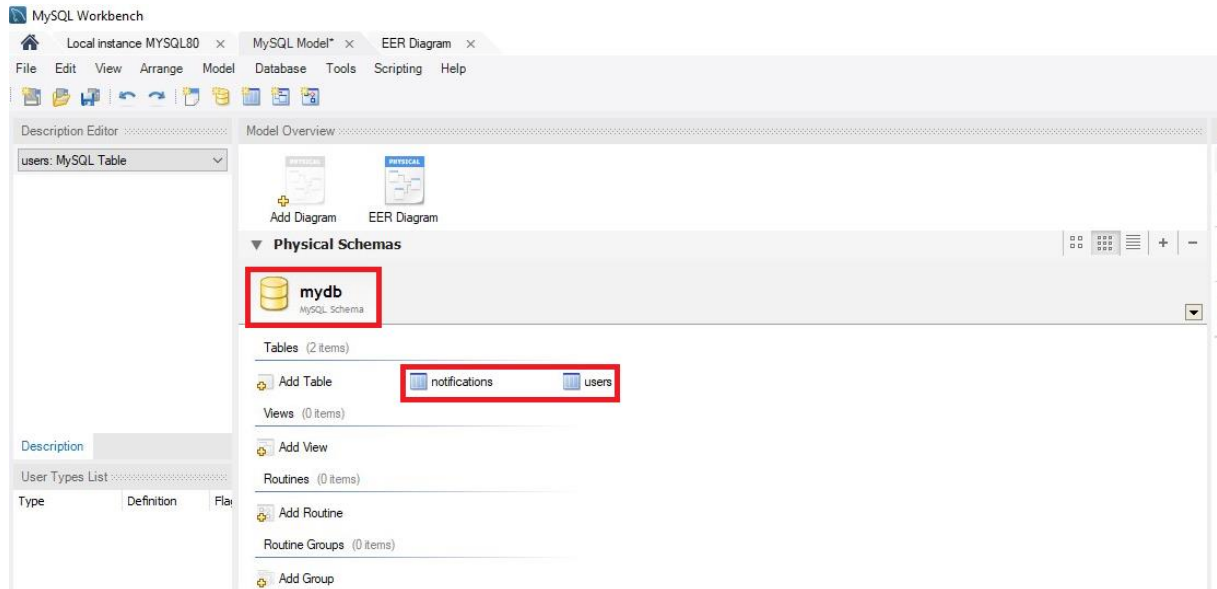


Figura 46 Tela de criação do modelo banco de dados e suas duas tabelas - MySQL

Fonte: autor.

O modelo da tabela *users* foi criado para ser chave estrangeira da tabela *notifications* e assemelhar a arquitetura com a do modelo UVApp_CORE do *Outsystems*.

notifications - Table x

Table Name: Schema: **mydb**

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
class	SET('TPSI23ER', 'TPS...)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
text	VARCHAR(1000)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
idNotifications	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
titulo	VARCHAR(100)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
date_time	DATETIME	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
idUser	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Column Name: Data Type:

Charset/Collation:

Comments:

Storage: ☐ Virtual ☐ Stored

☒ Primary Key ☒ Not Null ☐ Unique

☐ Binary ☐ Unsigned ☐ Zero Fill

☒ Auto Increment ☐ Generated

Columns Indexes Foreign Keys Triggers Partitioning Options Inserts Privileges

Figura 47 Tabela notifications - MySQL

Fonte: autor.

Para criar um novo registro, basta clicar na linha abaixo da última linha criada, escolher o tipo de dado que vai receber e selecionar o tipo de registros através das *flags* PK (*primary key*), NN (*not null*), UQ (*unique*), B (*binary*), UN (*unsigned*), ZF (*zero filled*), AI (*auto increment*) e G (*generate column*).

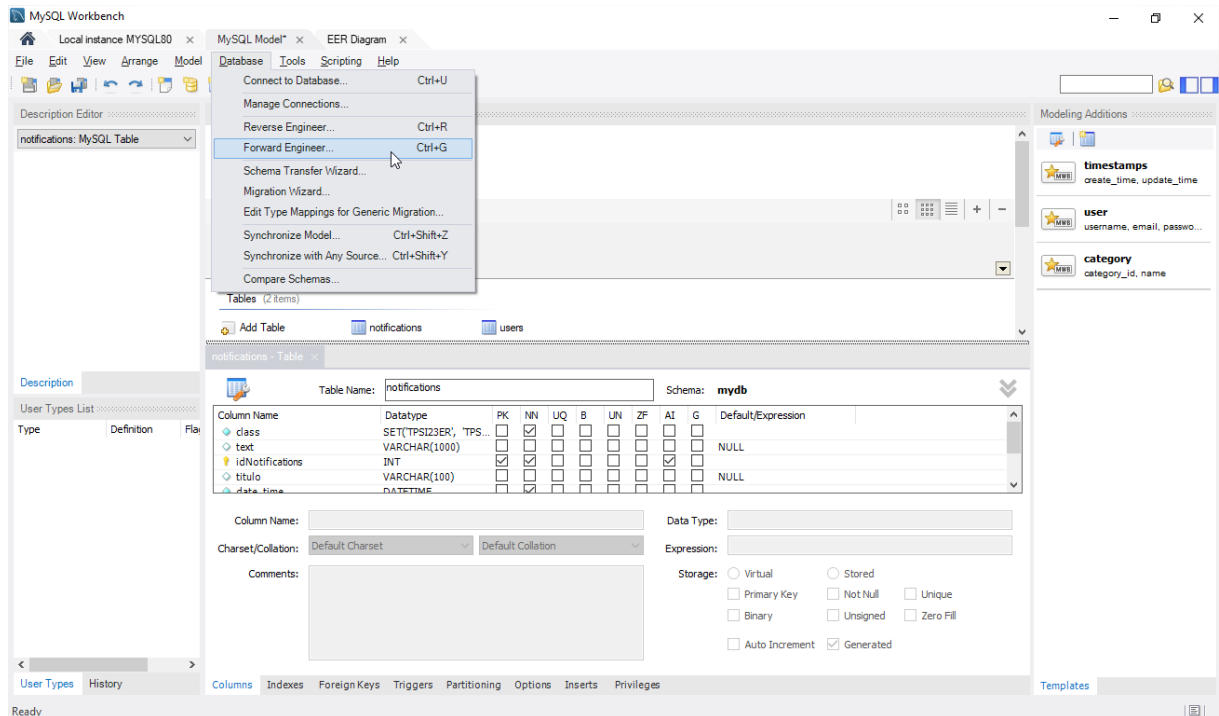


Figura 48 Criando uma instância local - MySQL

Fonte: autor.

Com apenas um clique pode-se criar uma instância local onde o modelo de dados é transpassado para um banco de dados através de *queries* geradas automaticamente.

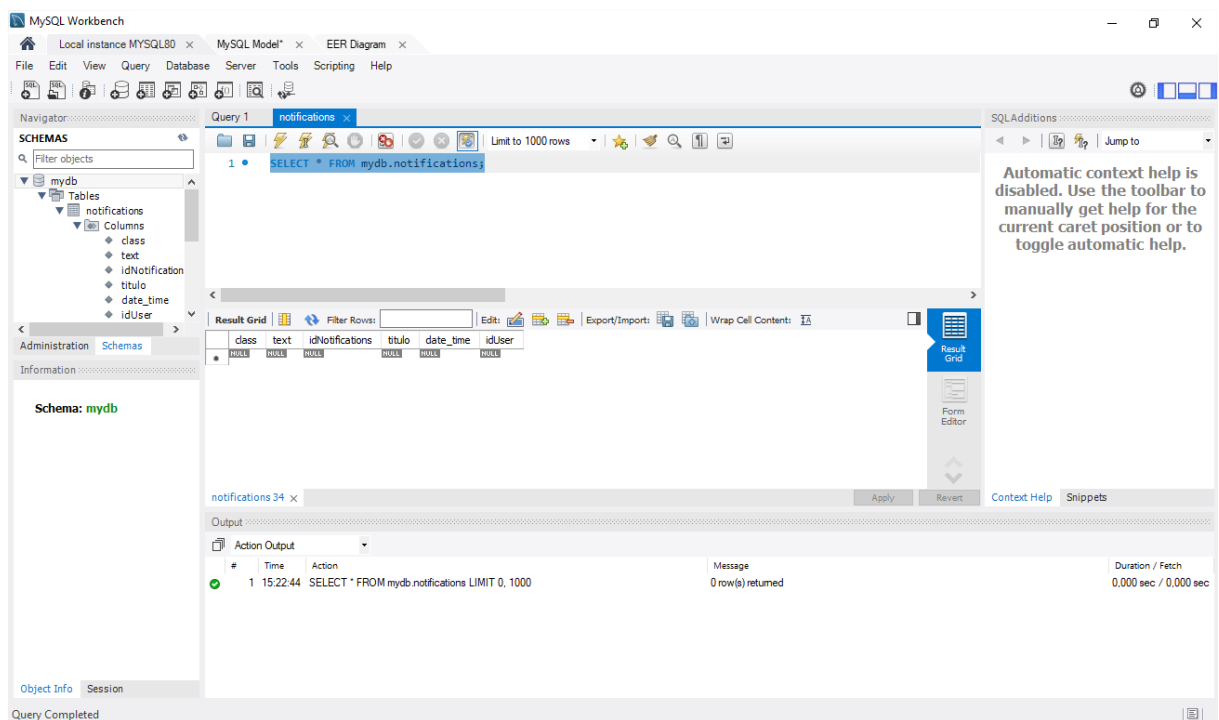


Figura 49 Banco de dados mydb - MySQL

Fonte: autor.

É possível montar *queries* através de um clique do botão direito em cima da tabela e a escolha da ação que se deseja realizar (criar, selecionar, alterar, deletar), ou através da codificação na área de trabalho do *MySQL Workbench*.

Na próxima subseção secundária será apresentado o processo de desenvolvimento do *back-end* que se comunica com o banco de dados.

4.3.2 Back-end

Para o desenvolvimento do *back-end* foi escolhida a linguagem *Python* utilizando a IDE *PyCharm* devido a afinidade da autora com a linguagem, pois a mesma já desenvolveu alguns pequenos programas de linguagem estruturada. Contudo, houve um grande desafio na criação do servidor, pois as bibliotecas e formas de implementação eram desconhecidas até então.

Com isso iniciou-se a codificação que se dividiu em duas etapas, a primeira é a conexão com o banco de dados, e a segunda é a de criação de uma API REST para receber requisições HTTP.

A API REST é basicamente uma interface de comunicação com aplicativos de softwares ou plataforma, que utiliza as requisições HTTP, utilizando a formatação de dados em JSON. A API é um conjunto de instruções que formam um protocolo de comunicação que serve para fornecer informações relevantes de uma aplicação. As operações básicas da API são POST, GET, PUT e DELETE. O REST é um conjunto de restrições utilizadas para que as requisições HTTP atendam os procedimentos definidos na arquitetura. As restrições basicamente são que a aplicação deve estar separada em uma estrutura de cliente e outra de servidor, as requisições devem ser feitas de forma independente, a API deve evitar chamadas recorrentes ao servidor utilizando cache, os recursos devem ser identificados e suas manipulações devem ser através de representações com mensagens descritivas, e deve-se utilizar links para navegar pelo aplicativo. Também, o tipo de API utilizada no presente trabalho é a privada – utilizada de forma local -, e retorno dos dados, por padrão, são definidos no formato JSON.

A primeira fase foi iniciada com uma larga pesquisa, buscando bibliotecas da linguagem *Python* que realizam conexão com a base de dados. Após esse processo de

pesquisa, a biblioteca *pymysql*¹ foi escolhida pela confiabilidade de estar disposta no site oficial do *Python* e pela fácil usabilidade. Sendo assim, a mesma foi importada na IDE e a conexão foi estabelecida através dos parâmetros passados conforme apresentado no código se segue:

```
import pymysql.cursors

SERVER_URL = "localhost"
DB = "mydb"
USER_NAME = "root"
PASSWORD = "a1b2c3d4e5"

SQL_CONNECTION = pymysql.connect(host=SERVER_URL,
                                  user=USER_NAME,
                                  passwd=PASSWORD,
                                  db=DB,
                                  charset='utf8mb4',
                                  cursorclass=pymysql.cursors.DictCursor,
                                  autocommit=True)
```

Feito isso, o próximo passo foi criar funções que executam *queries* no banco para criar, selecionar, atualizar e deletar os dados. O código abaixo representa como essas funções foram criadas para realizar o CRUD:

```
def create(turma, text, titulo, date_time, idUser):
    SQLcreate = """INSERT INTO mydb.notifications(class, text,
    titulo, date_time, idUser) VALUES (%s, %s, %s, %s, %s)"""

    with SQL_CONNECTION.cursor() as cursor:
        try:
            input_records = (turma, text, titulo, date_time, idUser)
            sql_exec = cursor.execute(SQLcreate, input_records)
            if sql_exec:
                print(sql_exec)
                print("Record Added")
            else:
                print(sql_exec)
                print("Not Added")
        except (pymysql.Error, pymysql.Warning) as e:
            print(f'error! {e}')
        cursor.close()
```

¹ Site por onde a biblioteca *pymysql* foi baixada <https://pypi.org/project/PyMySQL/>.

Assim como a função anterior foi criada, outras semelhantes também foram, porém cada uma com sua respectiva finalidade.

A segunda fase começou com a busca por bibliotecas que permitem criações de rotas para receber requisições HTTP em *Python*. Após encontrar a *http.client*, foram feitos alguns testes com o auxílio da ferramenta *Postman*, entretanto, depois de um longo tempo tentando sem conseguir os resultados esperados, foi preciso mudar de abordagem. A abordagem escolhida para essa mudança foi a utilização do framework *Flask*. Ele possibilitou a utilização da biblioteca *Flask* (para criar a aplicação UVApp respectiva ao *back-end*) e *request* para a conectar-se às requisições HTTP.

No caso deste trabalho, o *Flask* recebe de um *request* (requisição) um arquivo em formato JSON e o transforma em um objeto 'notificacao'. Após isso, envia os parâmetros dispostos no corpo do arquivo para as funções do CRUD. O próximo código a ser apresentado demonstra a rota da aplicação criada para receber o método *POST* através de uma requisição, e assim, enviar para o banco as solicitações recebidas.

```
@app.route('/api/createnotification/', methods=['POST'])
def createnotification():
    body = request.get_json()

    if "turma" not in body:
        return geraResponse(400, "O parâmetro turma é obrigatório.")
    if "text" not in body:
        return geraResponse(400, "O parâmetro text é obrigatório.")
    if "titulo" not in body:
        return geraResponse(400, "O parâmetro titulo é obrigatório.")
    if "date_time" not in body:
        return geraResponse(400, "O parâmetro date_time é obrigatório.")
    if "idUser" not in body:
        return geraResponse(400, "O parâmetro idUser é obrigatório.")

    notificacao = create(body["turma"], body["text"],
body["titulo"], body["date_time"], body["idUser"])
    return geraResponse(200, "Notificação Criada", "notificationId",
notificacao)
```

Vale ressaltar que ao desenvolver o método *GET*, houve a necessidade de expandir

esse método em dois. Um para buscar todas as notificações existentes no banco e outro para buscar as informações de uma notificação específica.

É possível notar alguns tratamentos de erro caso algum dos parâmetros não sejam recebidos no corpo do arquivo JSON.

Através da função ‘geraResponse’, o *back-end* gera uma resposta sinalizando se a operação foi concluída com sucesso ou não, e se foi mal sucedida, ele apresenta uma mensagem de erro personalizada. O código seguinte mostra como a função foi implementada:

```
def geraResponse(status, mensagem, nome_conteudo=False,
conteudo=False):
    response = {}
    response["status"] = status
    response["mensagem"] = mensagem

    if (nome_conteudo and conteudo):
        response[nome_conteudo] = conteudo

    return response
```

Tendo em vista todo o processo sendo apresentado separadamente, agora, este terá seu ciclo de vida descrito nas próximas linhas.

O processo completo do *back-end* nesta aplicação consiste em o *Flask* receber uma requisição (cujo corpo é formatado em JSON) HTTP através das rotas criadas, transformá-lo em um objeto, e enviar os parâmetros do corpo (*body*) desta solicitação – seja através do método *POST*, *GET*, *PUT*, ou *DELETE* - para o banco de dados; o banco executa a query respectiva ao método da requisição, retorna os dados solicitados - caso esteja executando a função *retrieve* que seleciona os dados solicitados para apresentação - e uma mensagem de sucesso ou insucesso. Após isso, o servidor recebe a resposta da base de dados, passa essa resposta para a função ‘geraResponse’, que retorna essas informações organizadas em um dicionário e, por fim, a função do servidor retorna uma resposta para a requisição.

A Tabela 1 apresenta uma associação entre os comandos utilizados nos métodos das requisições, no CRUD do servidor e nas *queries* do banco de dados.

Tabela 1- Associação entre os métodos das partes envolvidas no CRUD

Requisições	Servidor	Banco de Dados
POST	CREATE	INSERT

GET	RETRIEVE	SELECT
PUT	UPDATE	UPDATE
DELETE	DELETE	DELETE

4.3.3 Front-end

Este módulo de desenvolvimento representa a comunicação direta da aplicação com usuário e vice-versa.

Para o desenvolvimento do *front-end*, inicialmente, foi proposta a biblioteca *Kivy* do *Python*, pois a mesma foi desenvolvida voltada para aplicações mobile. Contudo, houve muita dificuldade para implementar a parte gráfica. Então, o próximo passo foi encontrar um *framework* – assim como houvera sido com o *Flask* – que facilitasse o desenvolvimento. Sendo assim, após algumas pesquisas, foi sugerido o *React Native* que é uma biblioteca *Javascript* para desenvolvimento de aplicativos de forma nativa para os sistemas *Android* e *IOS*. Em meio ao processo de instalação muitos erros aconteceram, como por exemplo o programa rodar no cmd *NodeJS*, ou o programa não reconhecer o emulador do *Android Studio*, entre outros menores que demandaram a reinstalação de tudo umas duas vezes. Depois de todas essas intempéries, decidiu-se mudar a estratégia novamente e o *framework Ionic* foi escolhido. A instalação do *Ionic* foi simples e o *framework* possui como componentes o *Cordova* (faz integração com recursos nativos) e o *Angular* (usado para a criação da parte *Web* da aplicação).

Ao instalar o *Ionic* foi necessário criar um novo projeto através do comando *ionic start <nome_do_projeto> <tipo_do_projeto>*. No caso deste projeto o comando dado foi *ionic start FUNCIONALIDADE3_FRONTEND tabs*. Com isso, toda uma estrutura básica do tipo *tabs* foi criada - Figura 50.

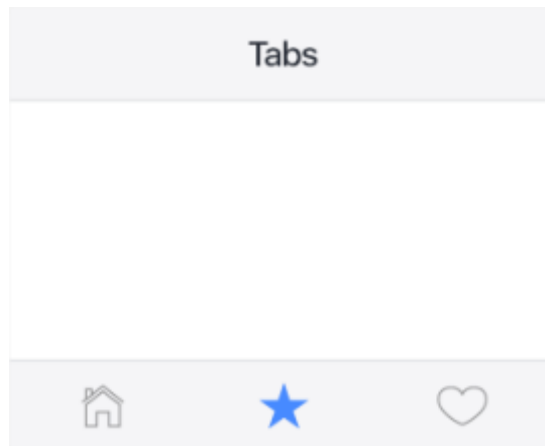


Figura 50 Página inicial de um projeto do tipo Tabs - Ionic

Fonte: autor.

Depois de todo esse pré-processo, a codificação pôde ser iniciada. O primeiro desafio foi criar uma interface que apresentasse todas as notificações, tal qual foi feito na página inicial do *Outsystems*. Para isso, o cabeçalho da página foi reutilizado para a inclusão de um botão de ícone com o símbolo do sinal positivo incitando a adição de uma nova notificação. O método *plusClick()* foi associado ao botão através do evento (*click*). Também, o *component List* do *Ionic* foi adicionado no arquivo *tabs1.page.html* – vide o código a seguir.

```
<ion-content [fullscreen]="true">
  <ion-list>
    <ion-item *ngFor="let notification of notifications" button
      (click)="itemClick(notification)" detail>
      <ion-label>
        <h2>{{notification.titulo}}</h2>
        <h3>{{notification.text}}</h3>
        <h3>{{notification.date_time}}</h3>
        <h3>{{notification.class}}</h3>
      </ion-label>
    </ion-item>
  </ion-list>
```

A diretiva **ngFor* foi utilizada para realizar um *loop* no objeto *notifications* e apresentar na tela cada notificação vinda do método *GET*. Os atributos do objeto são apresentados através das variáveis dispostas entre chaves.

Feito isso, o próximo objetivo foi criar um objeto no arquivo *tab1.page.ts* – que é o arquivo *Typescript* que se comunica com o HTML promovendo o modelo lógico da página – para enviar para a página as informações da base de dados.

É importante salientar que o *Typescript* é a linguagem base do *Angular* e um conjunto mais elaborado do *Javascript*.

Na classe principal da *tab1.page.ts* a variável *notifications* do tipo *array* de objetos foi criada para receber as notificações vindas do servidor. Depois, no método construtor foram importadas as classes *Router* (nativa do *Angular*), *NotificationProviderService* (criada no projeto) e *NgZone* (nativa do *Angular*). A classe *NotificationProviderService* foi declarada também na diretiva *providers* dentro da área de *Component* da página. Depois dessas declarações, três métodos foram criados dentro da classe principal: *ionViewDidEnter()*, *plusClick()* e *itemClick()*.

O primeiro método faz parte do ciclo de vida de páginas do Ionic e significa que o mesmo será disparado quando o roteamento do *Component* finalizar a animação da tela. Dentro dele a propriedade da página *ngzone* foi incluída para tornar os processos assíncronos, depois a função *getNotifications()* do serviço *notificationProvider* é disposta para atribuir ao *array* *notifications* as informações trazidas das requisições. O código abaixo representa o funcionamento deste processo.

```
public ionViewDidEnter() {
  this.ngzone.run(() => {
    this.notificationProvider.getNotifications().subscribe(
      data => {
        this.notifications = (data as any).notificacao;
      }, error => {
        console.log(error);
      }
    )
  })
}
```

A próxima função (*plusClick()*), apenas utiliza-se da propriedade *router* para navegar para a próxima página passando parâmetro nulo. A próxima página é a de detalhes da notificação e, neste caso, será apropriada para criar uma nova.

A terceira e última função desta página é a *itemClick()* – chamada no item da lista na página *tab1.page.html*. Ela, assim como a anterior, utiliza-se da propriedade *router* para navegar para a próxima página, porém, passando como parâmetro os objetos armazenados no *array* *notifications*.

Depois desse primeiro momento na primeira página, uma segunda pasta

notificationdetail foi criada para armazenar os arquivos utilizados para a criação da segunda página. Nesta, foi construída no arquivo *notificationdetail.page.html* uma estrutura de formulário para receber e apresentar as notificações enviadas do *notificationdetail.page.ts*, e dentro do formulário componentes de *inputs*, seleção de dados, calendário e ícones foram adicionados. Em cada componente dentro do formulário foram associados os parâmetros *formControlName* (exclusivo do formulário) e *value* (utilizado para passar valores para as variáveis da *notificationdetail.page.ts*); no botão Salvar, cuja função é submeter os valores atribuídos pelo usuário ao *formGroup* (parâmetro do componente que agrupa os valores do formulário), foi associado o tipo *submit* ao parâmetro *type*, e no início do formulário foi atribuído o valor *ionicform* à diretiva *formGroup* e o método *submitForm()* à diretiva *ngSubmit*. Também, dois botões foram criados fora do formulário: Deletar e Voltar, os quais chamavam as funções *deleteNotification()* e *voltar()* respectivamente em seus eventos (*click*).

Seguindo adiante, o arquivo *notificationdetail.page.ts* começou a ser codificado com algumas importações e declarações que serão mencionadas no decorrer da dissertação. Entretanto, o cerne da página começou com construção de um objeto *ionicform* (do tipo *FormGroup*) que recebe o grupo de informações de um formulário seguido da função *ngOnInit()* – função tal que compõe o ciclo de vida de páginas *Ionic* -, cuja é disparada durante a inicialização dos componentes, e possui finalidade de inicializar membros locais e fazer chamadas para serviços que precisam ser feitos apenas uma vez. Depois de inicializada a página, o *id* da notificação passado pelo roteamento da página anterior é recebido através do método *GET* da propriedade *route* do tipo componente *ActivatedRoute* (este componente é um observador que possui a função de mapear quaisquer informações que chegam) e alocado em uma variável. Após essa alocação, propriedade da página *ngzone* foi incluída para tornar os processos assíncronos, e o primeiro processo é a chamada da função *getNotification* do serviço *notificationProvider* com o *id* recebido anteriormente passado como parâmetro para obter os dados de uma notificação específica. Essas informações são passadas para uma variável, e depois há uma validação para saber se o *id* é diferente de nulo, pois se for nulo, as variáveis *titulo*, *text*, *turma* e *datetime* (variáveis que são passadas para a página HTML no parâmetro *value*), são preenchidas com os valores de cada item da notificação; do contrário, se o *id* for nulo, essas variáveis são preenchidas com valores nulos. Não só as variáveis criadas para popularem o parâmetro *value* do HTML foram criadas e inicializadas, mas também os itens do formulário *ionicform* tiveram seus valores setados com as respectivas informações providas da notificação.

Com tudo configurado, após a função *ngOnInit()*, a função *voltar* – acionada pelo

botão voltar na página HTML - foi implementada apenas com a definição de roteamento para a página anterior. Outra função que também foi implementada, foi a *submitForm()*, que é acionada pelo botão voltar enviando consigo o formulário. Assim que esta função é chamada, ocorre uma validação de *id*; se for igual a nulo, uma nova notificação é criada a partir da criação de um novo objeto do tipo *CriaNotificacao*, onde seus atributos, são os valores imputados no formulário pelo usuário – *this.ionicform.value.turma* é uma representação de um dos atributos do novo objeto. Assim, depois de criado, o objeto é passado como parâmetro pelo método *postNotification()* para o serviço *notificationProvider*. Por outro lado, se o *id* for diferente de nulo, um novo objeto do tipo *Notificacao* é criado e seus atributos, são os valores atualizados (ou não) no formulário pelo usuário – *this.ionicform.value.turma* é uma representação de um dos atributos do novo objeto. Criado o objeto, o mesmo é passado como parâmetro pelo método *postNotification()* para o serviço *notificationProvider*. Logo em seguida, após a validação, o usuário é direcionado para a página inicial de notificações. Por fim, a função *deleteNotification()* foi criada para ser acionada pelo evento (*click*) através do botão Deletar exposto no HTML, e enviar o id da notificação como parâmetro na requisição *DELETE* por meio do serviço *notificationProvider*. Após a chamada do serviço, a propriedade *router*, direciona o usuário para a página anterior.

A classe *NotificationProviderService*, a qual foi utilizada pelo parâmetro *notificationProvider* para apontar para os serviços que a mesma oferece, foi implementada de forma que contemplasse todas as requisições executadas nas páginas anteriores. Nesta etapa de codificação, o serviço *HttpClient* foi importado e utilizado para fazer as requisições *POST*, *GET*, *PUT* e *DELETE*. A seguir o código contendo a estrutura do método *PUT* e *DELETE* serão apresentados:

```
import { HttpClient } from '@angular/common/http';
import { Notificacoes } from 'src/contents/notificacoes.content';
import { CriaNotificacao } from
'src/contents/CriaNotificacao.content';

@Injectable({
  providedIn: 'root'
})

export class NotificationProviderService {

  constructor(public http: HttpClient) { }
```

```

private baseUrl: string = "/api/"

public updateNotification(notificacaoupdate: Notificacoes){
    return this.http.put(this.baseUrl + "updatenotification",
    notificacaoupdate);
}

public deleteNotification(id: number){
    return this.http.delete(this.baseUrl + "deletenotification/" +
    id);
}
}

```

No decorrer do processo, alguns erros de conexão estavam acontecendo pois o *front-end* não conseguia se comunicar com o servidor. Então, após algumas pesquisas, foi entendido que era necessário configurar uma comunicação *proxy*. Para isto, foi criado o arquivo *proxy.conf.json* com o seguinte código:

```

{
  "/api/*": {
    "target": "http://localhost:5000",
    "secure": false,
    "logLevel": "debug"
  }
}

```

Dessa forma, é possível definir que */api/* é equivalente à URL do servidor.

Com todas essas informações apresentadas separadamente sobre a estrutura do *front-end*, um momento de consolidação será utilizado para explicar como o processo acontece na realidade. Primeiro a página inicial (*tab1*) é inicializada, com isso, todas as notificações do servidor são carregadas pelo método *GET*. Neste momento o usuário pode seguir por dois caminhos, criar uma nova notificação ou atualizar alguma das existentes. Se a escolha for a de criar uma nova, o usuário será direcionado para a página que contém os detalhes de notificação, o mesmo preencherá todos os campos, e ao clicar em Salvar, uma requisição de método *POST* será enviada ao servidor e este adicionará uma notificação na base de dados. Contudo, se o usuário preferir atualizar uma notificação, ele deverá clicar em cima de alguma para ser direcionado para a página que contém os detalhes da notificação, atualizar as informações desejadas e clicar em Salvar. Após isso, o *front-end* enviará uma requisição de método *PUT* para o servidor que por sua vez atualizará a base de dados. Finalmente, se o


usuário quiser deletar uma notificação, é necessário que siga o mesmo processo anteriormente citado de clicar na notificação desejada, porém ao invés de clicar em Salvar, deve clicar em Deletar. Assim sendo, uma requisição será enviada (com o método *DELETE*) para o *back-end* solicitando a deleção do *id* em questão (passado como parâmetro).

As Figuras 51 até a 53 irão ilustrar o processo de criação de uma notificação.



Figura 51 Página de notificações - Ionic

Fonte: autor.

 Nova Notificação

Título *

Semana de Psicologia

Texto *

Atenção pessoALL!! 30AC para quem participar da seman

Turma

TPSIK9P2 ▾

Data e Hora

16-06-2020 10:10

Enviada

✓

SALVAR

DELETAR

VOLTAR

Figura 52 Página de criação de notificação - Ionic
Fonte: autor.

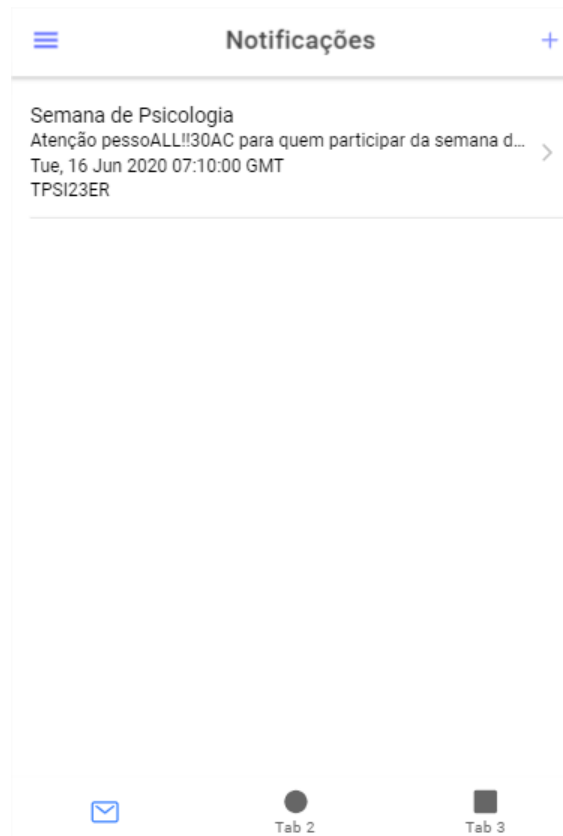



Figura 53 Página de notificações - Ionic
Fonte: autor.

Agora, após a inserção dessa notificação na base de dados será apresentada a atualização dos mesmos através das Figuras 54 e 55.


 **Semana de Psicologia**

Título *
ATENÇÃO!!! Semana de psicologia vindo aí

Texto *
Atenção pessoal!!!30AC para quem participar da semana

Turma TPSID0F4 ▾

Data e Hora 21-06-2020 07:10

Agendada


SALVAR

DELETAR


VOLTAR

Figura 54 Página de atualização da notificação - Ionic
Fonte: autor.



Figura 55 Página de notificações com data alterada - Ionic
Fonte: autor.

Por fim, após a data da notificação ter sido atualizada, o processo de exclusão será apresentado através das Figuras 56 e 57.


 **ATENÇÃO!!! Semana de psicologia vi...**

Título *
ATENÇÃO!!! Semana de psicologia vindo aí

Texto *
Atenção pessoal!!! 30AC para quem participar da semana

Turma TPSID0F4 ▾

Data e Hora 21-06-2020 04:10

Agendada


SALVAR

DELETAR

VOLTAR

Figura 56 Página de detalhes da notificação - Ionic
Fonte: autor.

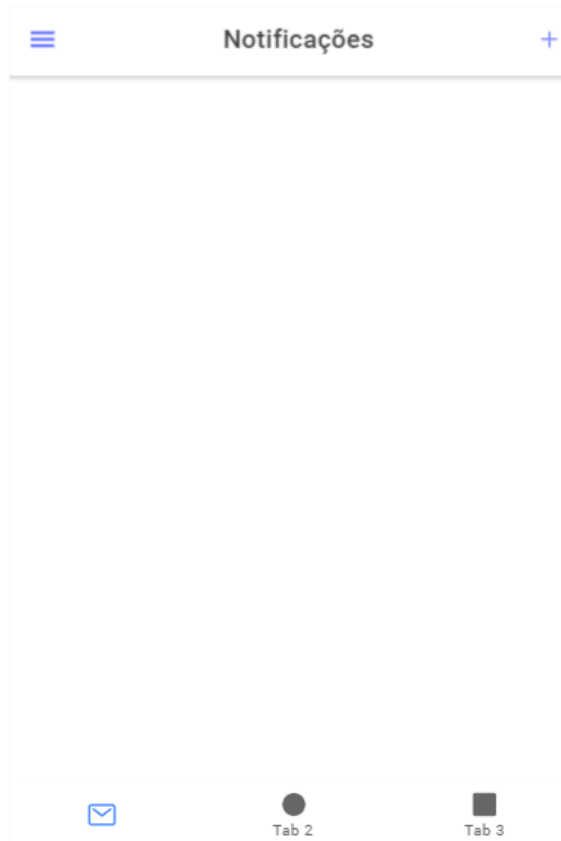


Figura 57 Página de notificações - Ionic
Fonte: autor.

Depois de todas as apresentações de como uma funcionalidade foi escolhida e desenvolvida por meio de duas metodologias diferentes, será dissertado na próxima subseção sobre o cerne deste trabalho² que é a curva de aprendizado.

4.4 ANÁLISE E COMPARAÇÃO DAS METODOLOGIAS

Tendo em vista o processo empírico de desenvolvimento da Funcionalidade 3 – Enviar notificações dos docentes para os discentes, foi possível agregar conhecimentos tácitos e explícitos, não só sobre a arte da programação, mas também sobre a curva de aprendizado propriamente dita. A Tabela 2 apresenta a comparação entre as métricas de tempo de desenvolvimento estimadas e reais para cada uma das metodologias.

Tabela 2 - Análise de horas trabalhadas

Horas trabalhadas

² O código completo pode ser encontrado em <https://github.com/Coliverfelt/TCC-Funcionalidade3>.

<i>Low-Code</i>		Programação Tradicional	
Estimado	Real	Estimado	Real
2 horas e 50 minutos	2 dias	1 dia e 21 horas	8 dias

Com essa tabela é possível observar que na programação *low-code*, houve muito menos esforço para aprender a desenvolver do que na programação tradicional.

No processo de aprendizagem com a primeira metodologia houve maior facilidade vide a interface da plataforma ser intuitiva e a linguagem utilizada para o desenvolvimento ser mais próxima da linguagem humana que a de máquina. Também, o curso *Developing Mobile Apps (OutSystems 11)* disponibilizado no site do *Outsystems*, foi essencial para dar impulso ao desenvolvimento, pois foi uma fonte confiável e uma oportunidade de ser introduzida à ferramenta através de outro módulo e de aproveitar grande parte do desenvolvimento sugerido no curso para a construção da funcionalidade proposta. O maior esforço feito para aprender foi feito em relação ao evento *onClick* que aciona a *Client Action* – ação do módulo do cliente, ou seja, que executa as requisições para o servidor -, onde é configurada o *Server Action* do CRUD. Contudo, o esforço não foi nada comparado ao processo de aprendizagem da segunda forma de desenvolvimento.

Logo no início do desenvolvimento utilizando programação tradicional, várias questões começaram a surgir como “Como serão estruturados os módulos necessários?”, “Que linguagem utilizar para o *front*?”, “Como conectar o *back-end* com o *front-end*?”, entre outras. Mas, para tentar amenizar a curva de aprendizagem, esse desenvolvimento foi estruturado em três fases.

Na primeira, a busca pelo *mysql* foi imediata, pois ao ler sobre o *MySQL Workbench* a ideia parecia certa. Assim, o processo de instalação e de construção da tabela foi bem arrojado; bastou seguir a intuição e as instruções da própria ferramenta. Contudo, algumas pesquisas para saber como instanciar e executar algumas *queries* no banco de dados foram um tanto necessárias e de grande valia.

Com a segunda fase propondo a conexão entre a base de dados e um programa na linguagem *Python*, as pesquisas encetaram e seguiram fluídas. Havia muitas bibliotecas que forneciam este aparato e muitas referências de como utilizá-las. Depois de conectar ao banco, houve um momento delicado onde seria necessário preparar o servidor codificado em *Python* para, além de realizar o CRUD, prover uma rota onde o *front-end* teria (posteriormente) acesso. Após muita resistência em deixar de utilizar a linguagem pura, foi resolvido que o melhor seria utilizar um *framework* para facilitar a conexão com requisições HTTP, pois

através da biblioteca que a linguagem disponibiliza para este tipo de processo, estava ficando mais complexo do que deveria ser. Então, para descomplicar, o *framework Flask* foi o escolhido para se trabalhar. O suporte dele a este tipo de estruturação facilitou a escrita do código – diminuindo inclusive o número de linhas – e apresentou mais simplicidade na hora de realizar buscas e encontrar respostas. O que foi interpretado pela autora foi que a utilização de frameworks agiliza o desenvolvimento de aplicações e às vezes facilita o entendimento sobre a proposta do código.

Por fim, na terceira, última e mais trabalhosa parte, o *front-end* começou a ser planejado e desenvolvido. Inicialmente insistiu-se em desenvolvê-lo com a biblioteca *Kivy* nativa da linguagem *Python*, todavia, houve muita dificuldade em desenvolver apenas a interface com o usuário, daí presumiu-se que daria ainda mais trabalho para criar as requisições ao servidor.

Então, depois desse insucesso, houve uma pesquisa sobre quais linguagens são próprias para desenvolvimento mobile e surgiram duas opções: *React Native* e *Ionic*. De pronto, a primeira alternativa foi logo apontada – sem nenhum motivo específico para isso – e as instalações começaram. Para rodar a aplicação desenvolvida em *React Native* em um *browser* era apenas necessário a instalação do *React Native* junto ao *npm* e *NodeJS*, pois a linguagem *React Native* é uma biblioteca *Javascript*, o *NodeJS* é um interpretador de *JavaScript* assíncrono focado em migrar a programação do cliente (*front-end*) para o servidor, e o *npm* é um gerenciador de pacotes *Node*. E para emular a aplicação, o *Android Studio* deveria ser instalado e configurado. Na tentativa de rodar a aplicação no emulador Android, ocorreram vários problemas como o *React Native* não conseguir se conectar ao mesmo, ou problemas de acesso no arquivo *avd*. Depois desses problemas, houve a tentativa de rodar a aplicação pelo *browser*, mas por algum motivo desconhecido até então, ao invés de abrir o navegador *web*, abria um *cmd Node*. Por causa do tempo demasiadamente investido e não produtivo para o projeto, a escolha de trocar para o desenvolvimento em *Ionic* ocorreu.

Finalmente, depois dessa jornada na tentativa de utilizar uma linguagem mais acessível para a codificação do *front-end*, o *Ionic* foi instalado e configurado no editor de texto *Visual Studio Code*. O *framework* escolhido é um *kit* de desenvolvimento de software *open source* construído como um conjunto de *web components* permitindo que o usuário escolha entre os *frameworks Angular*, *React* ou *Vue.js* para interface com o usuário. No caso desse trabalho, o *Angular* (baseado em *Typescript*) foi escolhido para o desenvolvimento. Com o ambiente preparado, a parte da codificação seguiu tranquila por parte do *Ionic* (montando a página HTML), pois há muito material no site oficial. Contudo, a finalidade da

estrutura de pastas baixadas do *Ionic* dispostas no editor de texto se tornou complicada de entender, mas com pesquisas foi possível contornar e entender a finalidade dos arquivos necessários para o desenvolvimento da funcionalidade proposta. Nos arquivos de extensão .ts seriam disponibilizados os códigos *Typescript* e nos .html, a arquitetura do *Ionic*. Também, no decorrer do desenvolvimento foi entendido a necessidade de criar mais três tipos de arquivos:

1. *Proxy* – Foi configurado um arquivo de extensão .json para fazer o papel de servidor, agindo como intermediário das requisições do cliente (*front-end*) para o servidor (*back-end*).
2. *Provider* – Esse arquivo .ts é o serviço responsável por enviar as requisições através do *proxy* associado.
3. *Contents* – Os *contents* (extensão .ts) foram objetos criados cujo objetivo era receber os dados do formulário preenchido ou atualizado pelo usuário e ser enviado para o servidor através das requisições.

O ponto mais complexo no desenvolvimento com programação tradicional foi ter de descobrir quais ferramentas deveriam ser utilizadas para atender o objetivo do CRUD.

Por fim, ao comparar ambas as metodologias, foi possível observar – através dos conhecimentos tácitos adquiridos – que a curva de aprendizagem é, indubitavelmente maior, na programação tradicional. Todo o trabalho de analisar módulo a módulo, componente a componente, faz com que a complexidade aumente de forma significativa em relação ao *low-code*, que já possui tudo em uma plataforma única e apresenta uma forma de codificar mais recíproca à linguagem humana.

5 CONCLUSÃO

Naturalmente, apesar do curto prazo para desenvolver a funcionalidade proposta no presente trabalho, o objetivo essencial – de aprofundar os conhecimentos em programação – foi concluído com sucesso, pois o volume de informações coletadas, junto ao empirismo, proporcionou a absorção concreta de muitos conceitos essenciais para desenvolver muitos tipos de aplicações. Contudo, é preciso que se pratique mais a fim de aprender novos conceitos e ferramentas, reforçar o que já foi compreendido para não deixar que seja olvidado, e permitir que o aprendizado anterior, nas próximas criações de sistemas, se torne explicitamente mais fluido, eficiente e eficaz.

Ao longo de todas as tarefas, quatro fases sucederam em ordens diversas. Uma das etapas decorrentes era a de compreender o conteúdo, ou seja, encarar o conhecimento preciso para continuar; outra etapa foi a de retenção desse conteúdo, fixando-o para aplicá-lo ou para compreender outro assunto que exigia o conhecimento prévio deste; a terceira etapa foi a prática, que permitiu a consolidação e solidificação de ambas as fases anteriores; a quarta e última parte foi a de disseminação do aprendizado adquirido, pois assim, no momento de escrever sobre todo o ocorrido, obteve-se a oportunidade de reter ainda mais o aprendizado e ainda dispor o mesmo ao alcance de muitas pessoas. A ordem de algumas das fases não dependeu do conteúdo, mas sim do estado mental da autora. Por exemplo, por vezes houve a tentativa de assumir a fase da prática antes das duas anteriores, o que, algumas vezes, deu certo fazendo com que a compreensão se desse através da tentativa e erro, porém, a forma mais eficaz foi seguir a ordem descrita.

Também, é viável sugerir que pessoas não técnicas ou que desejam aprender programação, tenham como primeiro contato, ferramentas *low-code* para facilitar seu processo e amenizar as curvas de aprendizagem se alguma vez forem utilizar a outra metodologia. Ou, caso encetem pela programação tradicional, comecem utilizando linguagens ou *frameworks* que não sejam muito verbosos, sejam de fácil compreensão e que possuam uma comunidade grande e disposta a ajudar disponibilizando materiais na internet – como dúvidas, soluções, exemplos, etc. Um exemplo de linguagem de programação é o *Python*, cujo proporcionou uma curva de aprendizado pouco inclinada em vista do que, por exemplo, o *Java* poderia proporcionar; e o *Ionic* também é um bom *framework* para iniciar o aprendizado, pois possui uma documentação muito clara e vasta. É importante ressaltar também que nem todos os sistemas possuem a viabilidade de serem desenvolvidas em plataformas *low-code*, pois esse tipo de desenvolvimento - com baixa codificação - é mais

propício para aplicações com interações mais alto nível.

A importância deste trabalho não contempla apenas a parte tecnológica, mas também a mental, visto que para aprender algo novo é preciso estar receptivo para a compreensão. Do contrário, ao passo que qualquer pensamento que interfira o fluxo do aprendizado retarda o mesmo, a concentração e o foco no objetivo dos estudos são de essencial importância para a fluidez do entendimento das informações adquiridas.

Ao fazer uma breve comparação entre os aprendizados adquiridos, pode-se perceber que são metodologias distintas, mas que possuem a mesma arquitetura e o mesmo objetivo. Também, foi possível visualizar a importância de ambas para o desenvolvimento social, onde atualmente, o consumo da internet e aplicações mobile tem crescido cada vez mais. Também, na área profissional, onde as formas de construir sistemas se ampliam, fazendo com que os profissionais tenham a oportunidade de aumentar suas habilidades e o mercado se abra para a contratação de mais pessoas. Não só pessoas com um nível alto de conhecimentos técnicos, mas também, pessoas de diversos níveis de conhecimentos em relação a programação, visto que o *low-code* é bastante acessível.

Como propostas de trabalhos futuros sugere-se a aplicação desse estudo com pessoas totalmente leigas, ou com níveis de conhecimentos variados, para conferir qual abordagem é melhor aceita, ou trabalhos que proponham técnicas e métricas que ajudem a diminuir a acentuação das curvas de aprendizado, ou a aplicação de um ou mais tipos de refatoração no código, ou ainda trabalhos que analisem o desempenho da aplicação.

REFERÊNCIAS

- GORGONO, P. *The Evolution of Programming Languages*. 1 ed. Canadá: *Department of Computer Science Concordia University*, 2002.125p.
- PRESSMAN, R S. *Engenharia de Software: Uma Abordagem Profissional*. 7 ed. Porto Alegre: AMGH, 2011.780p.
- POPPENDIECK, M; POPPENDIECK, T. *Implementing Lean Software Development: From Concept to Cash*. 1 ed. Estados Unidos: Addison-Wesley Professional , 2006.304p.
- RIES, E. *A Startup Enxuta: Como os Empreendedores Atuais Utilizam Inovação Contínua para Criar Empresas Extremamente Bem-sucedidas*. 1 ed. São Paulo: Texto Editores Ltda, 2012.336p.
- CAROLI, P. *Lean Inception: Como Alinhar Pessoas e Construir o Produto Certo*. 1 ed. São Paulo: Caroli, 2018.177p.
- SAMMET, E. J. *Programming Languages: History and Future*. In: *Association for Computing Machinery*, 1972, Estados Unidos. *Association for Computing Machinery* 1972. pp. 601-610.
- Software Testing Help. 10 Best Low-Code Development Platforms In 2020*. In: *Software Testing Help*, 1 de maio de 2020. Estados Unidos. Disponível em: <<https://www.softwaretestinghelp.com/low-code-development-platforms/>>. Acessado em: 15 de maio de 2020.
- STEPNOV E. *A Review Of 12+ Low-Code and No-Code Development Platforms*. In: *FlatLogic*, 27 de março de 2020. República da Bielorrússia. Disponível em: <<https://flatlogic.com/blog/a-review-of-12-low-code-and-no-code-development-platforms/>>. Acessado em: 15 de maio de 2020.
- Red Hat. Middleware. What is na IDE?*. In: *Red Hat*, 1 de maio de 2020. Disponível em: <<https://www.redhat.com/en/topics/middleware/what-is-ide>>. Acessado em: 15 de maio de 2020.
- GALANTE, V. *Melhores Frameworks Para O Desenvolvimento de Aplicativos*. In: *Usemobile*, 8 de janeiro de 2019. Minas Gerais. Disponível em: <<https://usemobile.com.br/framework-desenvolvimento-aplicativos-2019/>>. Acessado em: 15 de maio de 2020.
- Outsystems. Developing Mobile Apps (OutSystems 11)*. In: *Outsystems*. Disponível em: <<https://www.outsystems.com/learn/courses/115/developing-mobile-apps-outsystems-11/>>. Acessado em: maio de 2020.
- SILVA, S. (R) *Evolução do Front-End Nos Últimos 10 Anos*. In: *Softerize Magazine*, 8 de fevereiro de 2014. Porto Alegre. Disponível em: <<https://magazine.softerize.com.br/artigos/outros-artigos/revolucao-front-end-nos-ultimos-10-anos>>. Acessado em: 06 de junho de 2020.

GOEL, A. *10 Best Web Development Frameworks*. In: *Hackr.io*, 25 de maio de 2020. Índia. Disponível em: <<https://hackr.io/blog/top-10-web-development-frameworks-in-2020>>. Acessado em: 06 de junho de 2020.

Digital House. *Back-End: O Que É, Para Que Serve E Como Aprender?*. In: *Digital House*, 24 de outubro. São Paulo. Disponível em: <<https://www.digitalhouse.com/br/blog/back-end-o-que-e-para-que-serve-e-como-aprender>>. Acessado em: 06 de junho de 2020.

MySQL. *MySQL Workbench*. In: *MySQL*. Disponível em: <<https://dev.mysql.com/doc/workbench/en/>>. Acessado em: 8 de junho de 2020.

Python. *Python 3.7.7 documentation*. In: *Python*. Estados Unidos. Disponível em: <<https://docs.python.org/3.7/>>. Acessado em: 12 de junho de 2020.

CARNEY B.; MA M.; WIGERT C. *UI Components*. In: *Ionic Framework*, 9 de dezembro de 2019. Estados Unidos. Disponível em: <<https://ionicframework.com/docs/components>>. Acessado em: 16 de junho de 2020.

Angular. *Introduction to the Angular Docs*. In: *Angular*. Estados Unidos. Disponível em: <<https://angular.io/docs>>. Acessado em: 16 de junho de 2020.

BROWN A. *The Evolution of Computer Languages Over 136 Years*, 11 de agosto de 2019. Estados Unidos. Disponível em: <<https://interestingengineering.com/the-evolution-of-computer-languages-over-136-years>>. Acessado em: 21 de junho de 2020.

ROY, V. P. *Programming Paradigms for Dummies: What Every Programmer Should Know*. In: *Research Gate*, 02 de abril de 2012. Disponível em: <https://www.researchgate.net/publication/241111987_Programming_Paradigms_for_Dummies_What_Every_Programmer_Should_Know>. Acessado em: 21 de junho de 2020.

GAJIC, Z. *Delphi History: From Pascal to Embarcadero Delphi XE 2*. In: *ThoughtCo.*, 17 de junho de 2017. Disponível em: <<https://www.thoughtco.com/history-of-delphi-1056847>>. Acessado em: 27 de junho de 2020.

INTERSIMONE, D. *Visual Component Library First Draft (Incomplete)*. In: *Embarcadero*, 24 de maio de 1993. Disponível em: <<http://edn.embarcadero.com/article/32975>>. Acessado em: 27 de junho de 2020.

SOUZA, I. Entenda o que é Rest API e a importância dele para o site da sua empresa. In: *RockContent*, 16 de abril de 2020. Disponível em: <<https://rockcontent.com/blog/rest-api/>>. Acessado em: 04 de julho de 2020.