RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
VOL. 1 NRO. 2 2020

APLICAÇÃO FACILITADORA DE CONTROLE DE PRESENÇA:

um estudo de tecnologias em nuvem para processamento digital de imagens

João Pedro Beck Land
Graduando em Engenharia de Software – Uni-FACEF
joaopedrobeckland@gmail.com

Ms. Leandro Borges
Docente – Uni-FACEF
leandro.borges@facef.br

Resumo
O presente artigo possui caráter bibliográfico exploratório, ou seja, visa oferecer uma visão ampla sobre o desenvolvimento de um sistema, no caso, da Aplicação Facilitadora de Controle de Presença. Explora-se ao longo do artigo quais e com as tecnologias e serviços externos que são utilizados na aplicação, bem como qual é a estrutura e funcionalidades do sistema. Apresenta a abordagem da engenharia de software voltada para a qualidade, com a metodologia de desenvolvimento orientada a testes. O objetivo com este artigo é ilustrar como daria-se, na teoria, o desenvolvimento de um *software* que utiliza-se de serviços externos em nuvem para proporcionar o reconhecimento facial.

Palavras-chave: API. Controle de Presença. Sistema.

Abstract
This article has an exploratory bibliographic character, that is, it aims to offer a broad view on the development of a system, in this case, the Presence Control Facilitator Application. It is explored throughout the article which and with the external technologies and services that are used in the application, as well as which is the structure and functionalities of the system. It presents the software engineering approach focused on quality, with a test-oriented development methodology. The purpose of this article is to illustrate how, in theory, the development of software that uses external cloud services to provide facial recognition would take place.

Keywords: *API. Presence Control. System.*

1 INTRODUÇÃO

O presente artigo visa apresentar uma aplicação voltada para a marcação e controle de presença de discentes do Centro Universitário Municipal de Franca, o Uni-FACEF. A aplicação será desenvolvida com foco em oferecer uma melhor solução e alternativa para o gerenciamento de frequência de cada estudante, de forma mais precisa, segura e rápida.

O problema de pesquisa a ser explorado pelo presente artigo pauta-se na seguinte questão: como realizar o controle de frequência dos estudantes do Uni-FACEF de uma maneira mais confiável e prática, com o apoio das técnicas e tecnologias da Engenharia de *Software*, para a instituição, docentes e discentes? A ideia por trás da problemática do artigo é desenvolver uma aplicação que torne mais ágil a marcação de presença dos alunos do Uni-FACEF. A solução desenvolvida apresenta-se como uma API (*Application Programming Interface*, em português, Interface de Programação de Aplicação) para a comunicação com outros serviços, como o do próprio Uni-FACEF.

Desta maneira, o artigo tem como objetivo apresentar um *software* que seja capaz de controlar a frequência dos estudantes, sem a necessidade do docente de destinar um tempo de aula para realizar este controle. O *software* atua com o auxílio de A.I (*artificial intelligence*) e *streaming* de vídeo, que através da análise do vídeo a A.I irá validar e verificar quem é o aluno em questão.

A ideia do projeto justifica-se ao perceber que o tempo utilizado para o controle de frequência (informalmente, a chamada) poderia ser investido em conteúdo prático de aula. Fortaleceu-se ainda mais a ideia ao considerar que o tempo gasto para realizar a chamada é proporcional a quantidade de estudantes presentes na sala, portanto, em salas com mais estudantes o tempo de aula reduz. Considera-se que este tempo de aula “gasto” pode ser ínfimo, porém ao considerarmos em visão global, no período de seis meses, por exemplo, o tempo depreendido torna-se significativo. Junto a isto, considera-se também que há docentes que não realizam a chamada em todas as aulas.

Para o gerenciamento do desenvolvimento da aplicação utilizou-se o método *Scrum*[1] e *Kanban*[2], auxiliado a isto, o *software Pipefy*[3] , foram usados para controlar as etapas do projeto. As ferramentas de desenvolvimento utilizadas são das mais variadas, desde ferramentas de prototipação de tela até testes de API, são eles: *Figma, Visual Studio Code, Insomnia* e *Amazon Web Services*. Usou-se também o TDD, para auxiliar nos testes. Para o banco da aplicação optou-se por utilizar o *MongoDB* e o *Postgres*. As linguagens de programação utilizadas foram *JavaScript (Node.js* e *React*) e *TypeScript*.

2 REFERENCIAL TEÓRICO DE ENGENHARIA DE SOFTWARE

Neste tópico é apresentado as tecnologias estudadas para o desenvolvimento da Aplicação Facilitadora de Controle de Presença. Esta seção (?) abrange desde a discussão sobre o banco de dados do sistema, até a exposição de quais são os serviços externos que auxiliam na execução da aplicação.

2.1 BANCO DE DADOS

A estruturação do banco de dados de uma aplicação é essencial para nortear o desenvolvimento da mesma. De acordo com Elmasri e Navathe (2005), “um banco de dados é uma coleção de dados relacionados. Os dados são fatos que podem ser gravados e que possuem um significado implícito”. Desta forma, a responsabilidade de um banco de dados é armazenar dados e registros que possuem sentido para determinada situação.

Quando Elmasri e Navathe (2005) afirmam a definição de um SGBD (sistema gerenciador de banco de dados), dizem que o SGBD “é uma coleção de programas que permite aos usuários criar e manter um banco de dados”. Em outras palavras, ainda de Elmasri e Navathe (2005), o SGBD é “um sistema de *software* de

[1]**“**Scrum é um framework para utilizado na gestão de projetos e desenvolvimento ágil de software”. Disponível em: <http://www.metodoagil.com/scrum/ >. Acesso em: 15 set. 2020.
[2]Kanban é “um sistema ágil e visual para controle de produção ou gestão de tarefas”. Disponível em: <https://artia.com/kanban/ >. Acesso em: 15 set. 2020.
[3] Pipefy é uma plataforma voltada para o gerenciamento de projetos e que possui recursos de automação dos fluxos de trabalho.

propósito geral que facilita os processos de definição, construção, manipulação e compartilhamento de bancos de dados entre vários usuários e aplicações". Portanto, o SGBD atua como um intermediador entre o próprio banco de dados e a aplicação e usuários que desejam acessar os registros armazenados.

Diante disso, "a construção de um banco de dados é o processo de armazenar os dados em alguma mídia apropriada controlada pelo SGBD", conforme diz Elmasri e Navathe (2005). Conclui-se que o banco de dados é parte crítica para o desenvolvimento de *softwares*, já que é através dele que se torna possível armazenar, recuperar e manipular registros que compõem a aplicação. Neste tópico apresenta-se os dois modelos de banco de dados utilizados pela Aplicação Facilitadora de Controle de Presença.

2.1.1 MongoDB

De com acordo Luiz Fernando Duarte Junior (2017), o *MongoDB* é um banco de dados *open source* de alta performance desenvolvido na linguagem *C++*, e diferente dos esquemas de outros tipos de bancos, não possui esquemas orientados à documentos. O *MongoDB* armazena as informações em JSON (*JavaScript Object Notation*), ou seja, de modo binário, o que permite aos dados serem aninhados de forma complexa, e ainda assim, serem fáceis de manipular.

2.1.2 *PostgreSQL*

De acordo com Edson José Dionísio (2015), o *SGBD PostgreSQL*, ou apenas *Postgres*, é um SGBD relacional, cujo foco é em extensibilidade e em padrões de conformidade. O *Postgres* armazena com segurança os dados e permite que sejam recuperados a partir de requisições no banco. Pode-se utilizar o *Postgres* simultaneamente para vários usuários.

2.2 DESENVOLVIMENTO

O desenvolvimento da aplicação é baseada na linguagem *Javascript*, presente tanto no *frontend* quanto no *backend*, com o *React* e o *Node*,

respectivamente. A Aplicação Facilitadora do Controle de Presença necessita de serviços externos que garantam e suportam o funcionamento do sistema. Neste tópico é apresentado quais são as tecnologias utilizadas no desenvolvimento da aplicação, ao abordar desde a construção do *backend* e *frontend*, e como se dá a relação dos serviços externos com a base do sistema.

2.2.1 *Node.js*

No vídeo Conceitos de *Node.js*, de Diego Fernandes (2020)[4], o *Node* permite utilizar o *JavaScript* no *backend*, ou seja, consiste em tudo o que usuário não enxerga. É a camada que contém as regras de negócio da aplicação e a interação com serviços externos, como métodos de pagamentos, por exemplo. O único jeito de conseguir ouvir eventos do usuário é utilizando rotas com métodos HTTP (*Hypertext Transfer Protocol*).

Apesar de usar *JavaScript*, o *Node* não é considerado uma linguagem e sim uma plataforma de desenvolvimento. Ele é utilizado no navegador *Chrome* e construído no *engine V8* para interpretar os códigos *JavaScript* e mostrar os resultados. E isso faz com que o *Node* seja muito rápido e possível de utilizar todos os recursos do *JavaScript*.

Além disso, o *Node* conta com o *NPM* (ou o *Yarn*), um gerenciador de pacotes que permite instalar bibliotecas de terceiros, como por exemplo uma biblioteca que integra meios de pagamentos. Isso facilita o desenvolvimento de cada aplicação. Ele é comparável ao *Composer* do *PHP* e o *PIP* do *Python*, por exemplo.

O *Node* segue a arquitetura *Event-Loop*, totalmente baseada em eventos e possui como ponto central a *Call Stack*, que consiste em uma pilha de eventos que foi disparado pelo código. Assim, o *Node* processa em um *loop* eterno, observando se existe uma nova função disparada na *Call Stack*, que são executadas em pilha.

Outra característica é que ele é *single-thread*, ou seja, ele executa em apenas um *thread* do processador por vez. Porém, o *Node* utiliza várias bibliotecas do *C++*, como a *libuv*, que permite utilizar *multi-thread* de forma oculta, permitindo que a *Call Stack* seja processada mais rápida.

[4] Bootcamp GoStack 2020

Além da arquitetura *Event-Loop*, o *Node* conta com a arquitetura *Non-Blocking I/O***,** que consiste em *input* e *output* não bloqueante, sendo assim, cada requisição para o *Node* não precisa necessariamente retornar todos os dados de uma vez só, sendo possível retornar esses dados em partes, pois dar uma resposta para o requisitante não significa que será bloqueado depois. Essa arquitetura é utilizada fortemente em *chats*, e ao usar o protocolo de *WebSocket* torna-se possível que a conexão com o *Node* nunca seja perdida, possibilitando o envio de novas atualizações.

2.2.2 *React*

De acordo com Diego Fernandes (2020), no vídeo Conceitos de *React.js*[5], o *React* é uma biblioteca voltada para a construção de interfaces, ou seja, tudo o que o usuário consegue visualizar. O *React* pode ser usado em aplicações *web*, *mobile* e até mesmo para aplicações de realidade virtual. Para seu desenvolvimento, o *React* utiliza-se do conceito de SPA (*single-page-application*), que surgiu em 2011, e se consolidou na época com o *Angular*. O SPA consiste em páginas que nunca mudam, e o que se modifica é o conteúdo exibido na tela para o usuário, isto proporciona um aumento de performance e usabilidade. Antes do SPA, cada rota acessada pelo *backend* retornava uma página com um conteúdo HTML. Atualmente, o *backend* retorna somente um JSON com as informações necessárias, e o *frontend* é o responsável pelas manipulações das rotas e exibições dos dados.

O *React* por si só é considerado apenas uma biblioteca, porém, quando analisado e considerando-se todo o ecossistema em volta, ele pode ser considerado um *framework*. Para a construção de interfaces para *web*, é necessário HTML (*Hypertext Markup Language*, estrutural), CSS (*Cascading Style Sheets*, estilização) e *Javascript* (lógica), porém com o *React*, tudo que é desenvolvido fica dentro do código *Javascript*, ou seja, todo conteúdo fica dentro de um único tipo de arquivo. Uma das características do *React* é a organização do código, que é nomeado de componentização. Um componente é um trecho de código que isolado não influencia

[5] i*dem nota nº 4*

nos demais códigos dentro da sua aplicação, ou seja, ele atua somente onde é chamado, não atuando como escopo global.

Outra vantagem, é a divisão de responsabilidades entre o *frontend* e o *backend*. As regras de negócio, acesso e manipulação do banco de dados e qualquer ação referente a isso, se tornam responsabilidades do *backend*. A responsabilidade de *frontend* é apenas carregar a interface para o usuário. A comunicação entre o *frontend* e o *backend* se dá através de APIs, que servem para garantir a troca de informações entre ambos, o que permite a alterar, apagar, salvar e listar recursos. O JSX é uma extensão de sintaxe *JavaScript*, que combina a sintaxe do JS com XML. Permite que seja possível escrever HTML no código *JavaScript*, e ele torna o código mais fácil de visualizar e dar manutenção.

As bibliotecas *Babel* e *Webpack* são, atualmente, essenciais para a utilização de aplicações *JavaScript*, cujo modelo mais recente faz uso dessas duas bibliotecas. O *Babel* faz com que a conversão do código *JavaScript* seja feita para uma versão que o navegador compreenda, e o *Webpack* auxilia o *JavaScript* a entender como importar imagens e CSS no código HTML.

O *Babel* é responsável por converter o código JS em um código que o navegador entenda. Já o Webpack possui três principais funções: o *bundle*, *loaders* e *live reload*. A partir do código compilado pelo *Babel* pego pelo *Webpack*, há a criação do *bundle*, um arquivo único que contém todo o código JS da aplicação. Os *loaders* ensinam o *JavaScript* a entender como importar arquivos de CSS e de imagem, por exemplo, uma vez que por padrão a biblioteca *Babel* compreende apenas arquivos JS, ou seja, os *loaders* são os responsáveis por fazer com que a aplicação entenda como importar arquivos diferentes de JS. A última função é o *live reload*, que é o responsável por carregar automaticamente as alterações.

2.2.3 Serviços externos

Para apoiar o funcionamento da Aplicação Facilitadora do Controle de Presença, é necessário que haja integração do sistema principal com três recursos externos, cada qual com sua funcionalidade e processos que são essenciais para o desempenho e performance da aplicação. Será utilizado um protocolo de comunicação, o *WebRTC* e dois serviços da *Amazon*: o *Kinesis* e o *Rekognition*.

2.2.3.1 *Amazon Kinesis*

De acordo com a própria definição da *Amazon*[6], o serviço *Kinesis* pode ser definido como um facilitador para "a coleta, o processamento e a análise de dados de streaming em tempo real, permitindo que você obtenha insights oportunos e reaja rapidamente às novas informações". Desta forma, o *Kinesis* possibilita que a Aplicação Facilitadora do Controle de Presença recolha o vídeo e envie para o *Rekognition*, responsável pela identificação dos rostos das pessoas, a fim de detectar a compatibilidade do rosto filmado com o rosto de algum aluno regularmente matriculado no Uni-FACEF.

Portanto, o *Amazon Kinesis* constitui-se basicamente em ser um serviço que irá intermediar o envio do vídeo para o processamento final no serviço *Amazon Rekognition*. O *Kinesis* não necessita esperar a finalização da coleta de dados para começar o processamento da informação, e esta característica permite que o serviço responda prontamente às solicitações que recebe.

2.2.3.2 *Amazon Rekognition*

Conforme a descrição do *Amazon Rekognition*[7], o serviço é definido da seguinte forma:

> O *Amazon Rekognition* facilita a adição de análises de imagem e vídeo aos seus aplicativos usando a tecnologia comprovada e altamente escalável de aprendizagem profunda, que não requer conhecimentos de *machine learning* para usar. Com o *Amazon Rekognition*, você pode identificar objetos, pessoas, texto, cenas e atividades em imagens e vídeos, além de detectar qualquer conteúdo inapropriado. O *Amazon Rekognition* também fornece recursos de análise facial e pesquisa facial altamente precisos que você pode usar para detectar, analisar e comparar rostos para uma ampla variedade de casos de uso de verificação de usuários, contagem de pessoas e segurança pública.

[6] Disponível em: <https://aws.amazon.com/pt/kinesis/>. Acesso em: 01 de set. 2020.
[7] Disponível em: <https://aws.amazon.com/pt/rekognition/?blog-cards.sort-byitem.additional Fields.createdDate&blog-cards.sort-order=desc>. Acesso em: 01 de set. 2020.

Desse modo, o *Amazon Rekognition* é o responsável por identificar quem é a pessoa que está presente no vídeo enviado pelo *Amazon Kinesis*, e retornar se há compatibilidade suficiente entre as pessoas cadastradas previamente e a pessoa em questão. O recurso de identificação com o *Rekognition* funciona através de uma das duas APIs do serviço: o *Amazon Rekognition Image* (para imagens) ou o *Amazon Rekognition Video* (para vídeos). O serviço em questão aprende a reconhecer os rostos com o suporte de uma AI.

A estrutura de funcionamento das APIs voltada para reconhecimento de faces, de acordo com a documentação oficial do serviço, afirma que "o *Amazon Rekognition* pode armazenar informações sobre faces detectadas em contêineres do lado do servidor conhecido como coleções"[8]. Desta forma, é possível utilizar "as informações faciais armazenadas em uma coleção para pesquisar faces conhecidas em imagens, vídeos armazenados e vídeos de *streaming*"[9]. Assim, o reconhecimento de faces de estudantes do Uni-FACEF seria executável através da combinação dos serviços *Amazon Kinesis* e *Rekognition*.

2.2.3.3 Protocolo de Comunicação *WebRTC*

O *WebRTC* é um protocolo de comunicação de código aberto apoiado e usado por diversas empresas como *Google*, *Apple* e *Microsoft*. Conforme o site oficial do *WebRTC*, mantido pelo *Google*[10], ele possibilita:

> Adicionar recursos de comunicação em tempo real ao seu aplicativo, que funciona sobre um padrão aberto. Ele suporta dados de vídeo, voz e genéricos a serem enviados entre pares, permitindo que os desenvolvedores criem soluções poderosas de comunicação de voz e vídeo.

Portanto, é uma forma de comunicação entre um o *frontend* e o *backend*, de maneira que possam interagir através do envio de informações de áudio e vídeo.

[8]Disponível em: <https://docs.aws.amazon.com/pt_br/rekognition/latest/dg/collections.html>. Acesso em: 01 de set. 2020.
[9] Idem nota nº 4
[10] Disponível em: <https://webrtc.org/>. Acesso em: 01 de set. 2020.

De acordo com a documentação do *Kinesis Video Streams*[11], a integração entre o serviço da *Amazon* e o *WebRTC* dá-se da seguinte maneira:

> O *Amazon Kinesis Video Streams* fornece uma implementação *WebRTC* em conformidade com os padrões como um recurso totalmente gerenciado. Você pode usar o *Amazon Kinesis Video Streams* com *WebRTC* para transmitir mídia ao vivo com segurança ou realizar interação bidirecional de áudio ou vídeo entre qualquer dispositivo de câmera *IoT* e reprodutores móveis ou *web* compatíveis com *WebRTC*. Como um recurso totalmente gerenciado, você não precisa construir, operar ou dimensionar qualquer infraestrutura em nuvem relacionada ao *WebRTC*, como servidores de sinalização ou retransmissão de mídia para transmitir mídia com segurança entre aplicativos e dispositivos.

No sistema da Aplicação Facilitadora de Controle de Presença o *WebRTC* é o responsável por interagir com o serviço *Amazon Kinesis*. E através deste protocolo de comunicação torna-se possível enviar o vídeo registrado pelo dispositivo de gravação (uma câmera ou webcam, por exemplo), para o *backend* da aplicação, e assim para o *Kinesis*. Portanto, o *WebRTC* age como um intermediário tradutor entre ambas pontas.

3 QUALIDADE

3.1 DEFINIÇÃO

De acordo com Raul Wazlawick (2019):

> A qualidade de *software* é uma área dentro da Engenharia de *Software* que visa garantir bons produtos a partir de bons processos. Pode-se falar, então, de dois aspectos da qualidade: a qualidade do produto em si e a qualidade do processo. Embora não exista uma garantia de que um bom processo vá produzir um bom produto, em geral, admite-se que uma equipe com um bom processo vá produzir produtos melhores do que se não tivesse processo algum”

Dessa forma, compreende-se que a qualidade de *software* é algo relativo, e que varia conforme a situação.

[11] Disponível em: https://docs.aws.amazon.com/kinesisvideostreams-webrtc-dg/latest/devguide/ what-is-kvswebrtc.html>. Acesso em: 01 de set. 2020. Tradução do autor, original em inglês, *Amazon.*

3.2 MÉTODOS PARA GARANTIR A QUALIDADE DE *SOFTWARE*

3.2.1 Verificação e Validação

De acordo com Ian Sommerville (2019), os processos de verificação e validação são relacionados com a aferição do desenvolvimento do *software*, no qual ele cumpre com seus requisitos e fornece os recursos previstos pelas pessoas que o pagaram. Sendo assim, Ian (2019, p. 205) afirma que:

> A verificação de *software* é o processo de conferir se o *software* cumpre seus requisitos funcionais e não funcionais declarados. A validação de *software* é um processo mais geral, cujo objetivo é assegurar que o *software* atenda às expectativas do cliente, e vai além da conferência da conformidade com a especificação, para demonstrar que o *software* faz o que se espera dele.

Ao definir-se validação e verificação, é importante ressaltar que estas áreas envolvem diversas outras relacionadas à garantia da qualidade de *software* [ou apenas SQA (*Software Quality Assurance*)], como as revisões técnicas, monitoramento de desempenho e uma série de testes.

3.2.2 Testes

Ao abordar sobre testes, Roger Pressman apud Miller (2016), o qual diz que “a motivação que está por trás do teste de programas é a confirmação da qualidade de *software* com métodos que podem ser economicamente e efetivamente aplicados a todos os sistemas, de grande e pequena escala”. Ainda segundo Pressman (2016):

> Teste é um conjunto de atividades que podem ser planejadas com antecedência e executadas sistematicamente. Por essa razão, deverá ser definido para o processo de software um modelo (*template*) para o teste — um conjunto de etapas no qual pode-se colocar técnicas específicas de projeto de caso de teste e métodos de teste.

Ian Sommervile (2019, p. 208) complementa a ideia do texto acima com a seguinte afirmação:

> Existem três estágios do teste de desenvolvimento:
> 1. Teste de unidade, em que são testadas unidades de programa ou classes individuais. Esse tipo de teste deve se concentrar em testar a funcionalidade dos objetos e seus métodos.
> 2. Teste de componentes, em que várias unidades são integradas, criando componentes compostos. Esse teste deve se concentrar em testar as interfaces dos componentes que promovem acesso às suas funções.
> 3. Teste de sistema, em que alguns ou todos os componentes em um sistema são integrados e o sistema é testado como um todo. O teste de sistema deve se concentrar em testar as interações dos componentes.

Portanto, de acordo com Ian Sommervile (2019), haveria três categorias de testes voltadas para o desenvolvimento de *software*, sendo elas: o teste de unidade, componentes e sistema.

3.2.3 *Test Driver Development* (TDD)

O *test driven development* (TDD), conhecido em português como desenvolvimento dirigido por testes, de acordo com Sommervile (2019), é uma das abordagens na qual o intuito é desenvolver o *software* por meio da intercalação entre testes e codificação. O TDD surgiu junto com a Programação Extrema (*Extreme Programming*, ou apenas, XP), porem ele é utilizado tanto no desenvolvimento ágil quanto nos tradicionais. Sommervile (2019, p. 219) nos diz que:

> As etapas no processo são:
> 1. Definir o incremento de funcionalidade necessário. Normalmente ele deve ser pequeno e implementável em poucas linhas de código.
> 2. Escrever um teste para a funcionalidade e implementá-lo como teste automatizado. Isso significa que o teste pode ser executado e informará se a funcionalidade for aprovada ou reprovada.
> 3. Executar o teste, junto a todos os outros testes que foram implementados. Inicialmente, a funcionalidade não terá sido implementada ainda, então o teste falhará. Isso é proposital, pois mostra que o teste acrescenta alguma coisa ao conjunto de testes.
> 4. Implementar a funcionalidade e executar novamente o teste. Isso pode envolver a refatoração do código existente para melhorá-lo e acrescentar novo código ao que já existe.
> 5. Depois que todos os testes são executados com sucesso, é possível passar para a implementação do próximo pedaço de funcionalidade.
> [...]
> O desenvolvimento dirigido por testes ajuda os programadores a esclarecerem suas ideias quanto ao que o segmento de código realmente deve fazer. Para escrever um teste, é preciso compreender o objetivo do que está sendo feito, já que essa compreensão facilita a escrita do código necessário.

O TDD, portanto, auxilia no desenvolvimento do *software* ao elucidar como deve ser a resposta diante de uma situação específica, no caso, o teste. Essa elucidação acontece devido a codificação acontecer após a criação do teste, ou seja, desenvolve-se a aplicação com enfoque no teste e em como a funcionalidade ao ser testada deve responder.

É uma abordagem de testes mais utilizada no *backend*, mas também pode ser utilizada para funcionalidades do *frontend*. O princípio do TDD é que o teste deve falhar inicialmente, já que a funcionalidade ainda não foi desenvolvida. No TDD, considera-se primeiramente qual é o teste em que será aplicado para a funcionalidade que se deseja testar. Após desenvolvido o teste, executa-se ele de forma a verificar o seu funcionamento e, neste momento, deve-se seguir a premissa inicial: ele deve fracassar. Com essa averiguação, pode-se desenvolver a funcionalidade que se visa testar e depois de concluída a codificação, o teste é rodado novamente. Nesta etapa, o teste deve ser executado adequadamente, e passa para a fase final, a de refatoração (melhoria do código inicial). Com a finalização do desenvolvimento da funcionalidade pretendida, repete-se o ciclo do TDD para desenvolver a próxima funcionalidade.

3.2 APLICAÇÃO DE QUALIDADE

Como citado anteriormente, uma das formas de garantir a qualidade de um *software* é aplicar técnicas de verificação, validação e testes. Durante o desenvolvimento do projeto faz-se uso de casos de testes previamente criados juntamente com a técnica de TDD. Ao longo da criação do *software*, para a aplicação desta técnica é utilizado um *framework* de testes direcionado para o ambiente que utiliza a linguagem *JavaScript/TypeScript* em sua codificação, o *Jest*.

De acordo com a equipe core do *Jest*[12], ele “é um *framework* de teste em *JavaScript* projetado para garantir a correção de qualquer código *JavaScript*. Ele permite que você escreva testes com uma API acessível, familiar e rica em recursos que lhe dá resultados rapidamente.” O *Jest* foi criado pelo *Facebook* e é focado na simplicidade, funciona com a maioria dos *frameworks JavaScript*, como *Node*.js,

[12]Disponível em: <https://jestjs.io/pt-BR/>. Acesso em: 14 jun. 2020.

React e *Angular*. Possui várias funcionalidades agregadas a ele, como o *coverage*, que consiste na porcentagem de código que foi coberto pelos testes.

4 METODOLOGIA DA PESQUISA

O presente artigo utilizou-se de livros, revistas e sites, a pesquisa é de classe exploratória uma vez que o tema a ser estudado é amplo e ainda pouco difundido ou experimentado, conforme visto no decorrer deste artigo. O objetivo principal do presente artigo foi explorar tecnologias que possibilitam a automatização de um sistema de presença de alunos em um centro universitário. Os objetivos secundários estão relacionados às tecnologias estudadas para conseguir entender a solução como um todo, nelas cita-se a utilização da linguagem *TypeScript* para o desenvolvimento da aplicação, tanto no *backend* quanto *frontend*, além do estudo voltado para o uso da abordagem do TDD ao longo da codificação.

5 RESULTADOS E DISCUSSÕES

5.1 PLANEJAMENTO DA APLICAÇÃO FACILITADORA DE CONTROLE DE PRESENÇA

Para o desenvolvimento do sistema realizou-se a documentação essencial para o início do projeto, de forma a estruturar e definir qual seria o escopo da Aplicação Facilitadora de Controle de Presença. Para isso, foi definido primeiramente o levantamento de requisitos do sistema, e a partir desta primeira etapa, possibilitou fazer o EAP, BPMN, documentação de requisitos e prototipação das telas do sistema, respectivamente.

Com a documentação finalizada pode-se compreender melhor quais são as delimitações, ações e recursos do *software* em questão. A Aplicação Facilitadora de Controle de Presença visa oferecer uma solução voltada para o gerenciamento do controle de presença no ambiente acadêmico do Uni-FACEF. Após a documentação da aplicação finalizada, as etapas seguintes são a de desenvolvimento e validação do sistema.

5.2 DESENVOLVIMENTO DA APLICAÇÃO FACILITADORA DE CONTROLE DE PRESENÇA

A aplicação é baseada na linguagem *TypeScript*, na qual o *frontend* é desenvolvido em *React* e o *backend* em *Node*. A responsabilidade principal do *frontend* é ser uma plataforma de visualização para que os docentes possam realizar o gerenciamento do controle da presença. O *backend* lida com as requisições que *frontend* necessita para exibir o conteúdo para o usuário, desta forma, é o responsável por coordenar os dados da aplicação. Além disso, é também no *backend* que ocorre a integração com a API externa da *Amazon*, e assim, torna-se possível a comunicação entre a Aplicação Facilitadora de Controle de Presença e o *Amazon Kinesis*.

A integração entre o *backend* da aplicação e a API da *Amazon* é um recurso que possui uma alta criticidade e influência no funcionamento geral da Aplicação Facilitadora de Presença, e isso deve-se por causa da sua responsabilidade, impacto e ação no sistema que a API tem. O processamento do vídeo e identificação de cada estudante será feito através da API da *Amazon*, que irá validar, ou não, a presença para o discente em questão. Por consequência, sem a completa integração com a API externa, o sistema não irá desempenhar suas funções de acordo com o esperado.

O reconhecimento do estudante é feito pelo serviço *Amazon Rekognition*, que recebe o vídeo do *Kinesis* para processar e identificar os rostos dos alunos, e retorna a compatibilidade para o *Kinesis*. O *backend* recebe em formato de JSON o retorno da API do *Amazon Kinesis*, que contém a informação de quem é o aluno. A partir do conhecimento desta informação, o *backend* decide o que faz com ela: registrar ou não a presença do discente.

No quesito qualidade da aplicação, pressupõe-se que a mesma passou pelas etapas de verificação, validação e testes, com uma performance conforme o previsto para os seus requisitos e funcionalidades. A fase de testes é crucial para garantir que a Aplicação Facilitadora de Controle de Presença responda a cada ação do usuário e do sistema da maneira como planejado, com enfoque particular para os testes que visam garantir a confiabilidade e segurança da integração da Aplicação

Facilitadora de Controle de Presença com a API *Amazon Kinesis*, e consequentemente, com o *Amazon Rekognition*.

Em relação ao banco de dados da Aplicação Facilitadora de Controle de Presença dois bancos serão necessários: o *MongoDB* e o *Postgres*. Optou-se pelo *MongoDB*, que faz uso de um esquema não relacional, para armazenar dados que precisam ser acessados com maior velocidade e não necessitam de uma estrutura rígida para registro. Quando se pensa no emprego do banco *Postgres*, um banco relacional, ele é utilizado para guardar informações que precisam de um esquema mais estruturado de armazenamento, e que sejam relacionados entre si.

Um exemplo do uso do *MongoDB* na Aplicação Facilitadora de Controle de Presença é o envio de notificações para o usuário, onde é necessário armazenar apenas o id do usuário, id da notificação e o texto da mensagem. Por outro lado, o *Postgres* é o responsável por proporcionar o armazenando de informações como o *log* de *login*, salvar registros de alterações, histórico de presença do aluno, relação das disciplinas cursadas e presença em cada uma delas e o estudante em questão.

O desenvolvimento da aplicação possui duas metodologias que permeiam sua codificação: o *Scrum* e o TDD. Planejar, definir, codificar e validar as tarefas do sistemas são etapas destrinchadas no método do *Scrum*, voltado para o desenvolvimento ágil; na qual defini-se um *backlog* inicial, e um tempo de *sprint*. Ao final de cada *sprint*, as tarefas do *backlog* devem estar prontas para que se possa iniciar a próxima *sprint*. O TDD é a metodologia focada em desenvolvimento de testes para a aplicação, majoritariamente utilizada no *backend* e permeia toda a codificação da aplicação. Esta prática de teste parte da premissa de que os testes são os guias para o desenvolvimento de cada funcionalidade do sistema, e eles são os primeiros a serem criados, antes mesmo de qualquer linha de código.

Portanto, o desenvolvimento da aplicação contempla desde a codificação do sistema, estruturação do banco de dados, integração com os serviços externos, gerenciamento da qualidade de *software* e até mesmo as metodologias que guiam o desenvolvimento.

5.3 IMPLEMENTAÇÃO DA APLICAÇÃO FACILITADORA DE CONTROLE DE PRESENÇA

Para que seja possível implementar a solução, é necessário que a mesma esteja completamente codificada e de acordo com os critérios de qualidade desejados, porém, devido ao objetivo do presente artigo, demonstrar como seria o desenvolvimento de um *software*, desde o início do levantamento de requisitos até a implementação propriamente dita, o que é abordado neste tópico, assim como no anterior, constitui-se puramente de uma estratégia e planejamento de codificação e implementação, que visa simular como seriam estas fases na prática.

Com toda a documentação completa e a estrutura de código devidamente desenvolvida e testada, faz-se necessário implementar a Aplicação Facilitadora de Controle de Presença, e a consequente disponibilização para uso por parte dos usuários. A implementação é a última fase na etapa de desenvolvimento da aplicação, e para isto, deve estar disponível para uso de forma interoperável e disponível de forma ininterrupta para acesso.

Desta forma, na última etapa, o *deploy* da Aplicação Facilitadora de Controle de Presença é realizado. A partir da implantação do sistema, torna-se possível aos usuários acessarem o mesmo, assim como o funcionamento com os serviços externos. Portanto, neste momento, a aplicação pode começar a executar suas ações e a retornar as informações necessárias.

5.4 FUNCIONALIDADES PENDENTES

Há três funcionalidades a serem desenvolvidas futuramente: assimilação do *hardware* (dispositivo de filmagem) com o *backend* da aplicação, integração com a *Amazon* de maneira completamente funcional, e o esquema de segurança de dados do sistema. Estes recursos não foram abordados diretamente e profundamente no exposto artigo devido a sua complexidade e abrangência.

Em relação ao acesso do *hardware* com a aplicação, a principal dificuldade consiste em como tornar possível a comunicação entre ambos, de forma que seja permitida o envio do vídeo para o *backend*, e consequentemente, para o serviço externo do *Amazon Kinesis.* É apenas com essa associação que o *Amazon Rekognition* pode receber o material para ser analisado e, desta forma, viabiliza o reconhecimento dos rostos dos estudantes através da AI do próprio serviço. A dificuldade nesta relação entre o *hardware* e o sistema ocorre devido a breve

documentação existente do protocolo de comunicação *WebRTC*, quando este é utilizado com a linguagem *JavaScript*.

A integração operacional do *Amazon Kinesis* com o *backend* da Aplicação Facilitadora de Controle de Presença é profunda e conta com falta de documentação detalhada por parte da Amazon. Quando se lida com linguagens nativas, como *Python* e *C*, encontram suporte na documentação do serviço *Kinesis*, porém, ao utilizar a linguagem *JavaScript*, a documentação não apresenta auxílio suficiente para facilitar e guiar as implementações e desenvolvimentos.

A segurança de dados refere-se a situações como a permissão de usuários, na qual é definido o tipo de acesso que os usuários podem ter. Um exemplo prático na Aplicação Facilitadora de Controle de Presença é a delimitação de permissão para usuários dos seguintes tipos: discente, docente, coordenador de graduação, administrador e usuário convidado. Com estas categorias de usuários torna-se possível limitar as ações e recursos de acesso para cada um. Para exemplificar, um usuário professor apenas poderá ver e editar os registros os quais está vinculado como docente da disciplina, do contrário, não é permitido o acesso. Da mesma forma, para um usuário aluno, que deseja ver sua frequência, ele consegue visualizar apenas as faltas e presenças nas disciplinas às quais está matriculado, e que ele mesmo relaciona-se com elas.

5.5 O FUTURO DA APLICAÇÃO FACILITADORA DE CONTROLE DE PRESENÇA

A próxima etapa da Aplicação Facilitadora de Controle de Presença é a criação de estratégias voltadas para colocar na prática o desenvolvimento do sistema. Apresentou-se no presente artigo um plano que traça a elaboração do projeto da Aplicação Facilitadora de Controle de Presença, desse modo, não foi pauta o desenvolvimento e execução propriamente dito do sistema, ao abordar apenas a apresentação teórica de como ocorreria a efetiva construção do *software*.

Com toda a base teórica e documentação anteriormente definidas, a implementação na prática torna-se mais ágil e direta, com riscos minimizados devido a toda a estrutura de suporte já existente em relação a aplicação. Portanto, o conteúdo apresentado no artigo atua como um orientador e facilitador no momento de execução do sistema.

O destino da Aplicação Facilitadora de Controle de Presença é ser implementado na prática, bem como aprimorar as funcionalidades pré-existentes, que são descritas ao longo do artigo; e também as apresentadas no tópico 5.4, que se refere a funcionalidade pendentes e que virão a ser desenvolvidas e instaladas.

6 CONCLUSÃO

Ao longo deste artigo apresentou-se um projeto de sistema voltado para realizar e gerenciar a frequência dos discentes no Uni-FACEF. A Aplicação Facilitadora de Controle de Presença caracteriza-se por ser um estudo e planejamento teórico de como seria organizada a abordagem de desenvolvimento e implementação da solução, caso esse sistema fosse de fato desenvolvido com o viés de produção e disponibilização para os usuários finais.

Conclui-se com o vigente artigo que a Aplicação Facilitadora de Controle de Presença é um projeto viável e hábil para implementação e desenvolvido na prática. Apresentou-se toda a documentação e estrutura base que proporciona o alicerce para o desenvolvimento, além de incluir as futuras funcionalidades já mapeadas.

REFERÊNCIAS

AMAZON Kinesis: colete, processe e analise facilmente streams de vídeo e dados em tempo real. Amazon Web Services. Disponível em: <https://aws.amazon.com/pt/kinesis/>. Acesso em: 01 set. 2020.

AMAZON Rekognition: automatize sua análise de imagem e vídeo com machine learning. Amazon Web Services. Disponível em: <https://aws.amazon.com/pt/rekognition/?blog-cards.sort-by=item.additionalFields.createdDate&blog-cards.sort-order=desc>. Acesso em: 31 ago. 2020.

DIONÍSIO, Edson José. **PostgreSQL Tutorial.** DevMedia. Disponível em: <https://www.devmedia.com.br/postgresql-tutorial/33025>. Acesso em: 12 abr. 2020.

ELMASRI, Ramez; NAVATHE, Shamkant B.. **Sistemas de Banco de Dados**. 4. ed. São Paulo: Pearson, 2005. Disponível em: <https://plataforma.bvirtual.com.br/Leitor/Publicacao/296/pdf/21?code=3NPEGXIDBcIoM2f5YbuKCoO0uvg4oNGfLZiBMPDZhy81sf7leZpjcxutGmRxp9xW3lL/1kY2Q0YODpSSUmVdAA==>. Acesso em: 14 set. 2020.

ESPINHA, Roberto Gil. **Kanban**: o que é e tudo sobre como gerenciar fluxos de trabalho. Artia. 2019. Disponível em: <https://artia.com/kanban/>. Acesso em: 14 set. 2020.

FERNANDES, Diego. **Conceitos React JS**. Rocketseat. Disponível em: <https://skylab.rocketseat.com.br/node/front-end-com-react-js/lesson/conceitos-react-js>. Acesso em: 12 abr. 2020.

FERNANDES, Diego. **Conceitos Node JS**. Rocketseat. Disponível em: <https://skylab.rocketseat.com.br/node/nivel-01/lesson/conceitos-node-js>. Acesso em: 12 abr. 2020.

JEST: testando JavaScript do jeito agradável. Jest. Disponível em: <https://jestjs.io/pt-BR/>. Acesso em: 14 jun. 2020.

JUNIOR, Luiz Fernando Duarte**. MongoDB para iniciantes em NoSQL.** iMartes. Disponível em: <https://imasters.com.br/banco-de-dados/mongodb-para-iniciantes-em-nosql>. Acesso em: 12 abr. 2020.

LAND, João Pedro Beck. **TCC**. GitHub. Disponível em: <https://github.com/JoaoPedro1999/tcc>. Acesso em: 17 set. 2020.

PESQUISAR faces em uma coleção. Amazon Web Services. Disponível em: <https://docs.aws.amazon.com/pt_br/rekognition/latest/dg/collections.html>. Acesso em: 01 set. 2020.

PRESSMAN, ROGER S. **Engenharia de Software**: uma abordagem profissional. PORTO ALEGRE: AMGH EDITORA, 2016.

SOMMERVILE, Ian. **Engenharia de Softwar**e. 10. ed. São Paulo: Pearson, 2019. Disponível em: <https://plataforma.bvirtual.com.br/Leitor/Publicacao/168127/pdf/172?code=5vcpK4ucWN6p1/oGuUhSALZJIGcRJrDp+x/pZX+eNMbhIVAQEN33o1EO67zFA2jKUHyCW4Z2nE5nsOKZpkdO/w==>. Acesso em: 14 set. 2020.

SCRUM: Framework Ágil para projetos complexos. Método Ágil. Disponível em: <http://www.metodoagil.com/scrum/>. Acesso em: 14 set. 2020.

WAZLAWICK, Raul Sidnei. **Engenharia de Software**: conceitos e práticas. 1. ed. Rio de Janeiro: Elsevier, 2013. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9788595156173/cfi/6/2!/4/2/2@0:0.00>. Acesso em: 15 set. 2020.

WebRTC. Disponível em: <https://webrtc.org/>. Acesso em: 31 ago. 2020.

What Is Amazon Kinesis Video Streams with WebRTC. Amazon Web Services. Disponível em: <https://docs.aws.amazon.com/kinesisvideostreams-webrtc-dg/latest/devguide/what-is-kvswebrtc.html>. Acesso em: 01 set. 2020.