WAGNER JUNIOR MORETTI TOME

RELATÓRIO DE ESTÁGIO
DESENVOLVIMENTO WEB NA EMPRESA SQUADRA DIGITAL

LAVRAS – MG

2021

WAGNER JUNIOR MORETTI TOME

RELATÓRIO DE ESTÁGIO
DESENVOLVIMENTO WEB NA EMPRESA SQUADRA DIGITAL

Relatório de estágio apresentado à Universidade Federal de Lavras, como parte das exigências do curso de Ciência da Computação, para obtenção do título de Bacharel

Prof. Dr. Rafael Serapilha Durelli
Orientador

LAVRAS – MG

2021

WAGNER JUNIOR MORETTI TOME

RELATÓRIO DE ESTÁGIO
DESENVOLVIMENTO WEB NA EMPRESA SQUADRA DIGITAL

Relatório de estágio apresentado à Universidade Federal de Lavras, como parte das exigências do curso de Ciência da Computação, para obtenção do título de Bacharel

APROVADA em 22 de Novembro de 2021.

Prof. Dr. Rafael Serapilha Durelli	UFLA
Prof. Dr. Maurício Ronny de Almeida Souza	UFLA
M.e Thiago Trindade Cardoso	UFJF

Prof. Dr. Rafael Serapilha Durelli
Orientador

LAVRAS – MG
2021

AGRADECIMENTOS

Agradeço a todos que contribuíram diretamente ou indiretamente em minha formação, em especial a minha família e amigos pelo suporte e incentivo em todos momentos que precisei. Ao professor Rafael Serapilha Durelli, pela orientação durante esse projeto. A todos professores que contribuíram para minha formação, pois sem eles esse caminho não seria possível. Por fim agradeço a instituição UFLA e a todos seus funcionários que estão sempre contribuindo para torná-la uma das melhores faculdades do país.

RESUMO

Buscando desenvolver um sistema digital que simplifique o processo burocrático na contratação de seguros, a Pottencial em parceria com a Squadra Tecnologia deram início ao desenvolvimento de um Portal totalmente novo voltado para o mercado de seguros. O portal do corretor foi pensado como um site voltado para corretoras e seus funcionários, integrando e facilitando a compra e desburocratizando processos.

Este relatório tem como objetivo demonstrar a experiência do estagiário em participar da elaboração deste portal.

O sistema denominado Portal do Corretor, foi desenvolvido com backend em linguagem C# utilizando framework .NET e o frontend utilizando React JavaScript, padrão de projeto REST e banco de dados SQL-Server.

Palavras-chave: Portal do corretor, Pottencial Seguradora, .Net developer, C#, Squadra Digital.

ABSTRACT

Seeking to develop a digital system that simplifies the bureaucratic process in contracting insurance, Pottencial, in partnership with Squadra Tecnologia, started the development of a completely new Portal focused on the insurance market. The Portal do Corretor was designed as a website aimed at brokers and their employees, integrating and facilitating purchases and reducing bureaucracy.

This report aims to demonstrate the intern's experience in participating in the development of this portal.

The system called Portal do Corretor, was developed with backend in C# language, using .NET framework and frontend using React JavaScript, REST design pattern and SQL-Server database.

Keywords: Broker Portal, Pottencial Seguradora, .Net developer, C#, Squadra Digital.

LISTA DE FIGURAS

Figura 3.1 – S.O.L.I.D - Princípio da inversão de dependências 16
Figura 3.2 – HTTP - Protocolo 17
Figura 4.1 – Boards Azure DevOps 27
Figura 4.2 – Sprints Azure DevOps 28
Figura 4.3 – Exemplo Pipelines Azure DevOps 29
Figura 4.4 – Exemplo Repos Azure DevOps 30
Figura 4.5 – Exemplo Repos-PullRequest Azure DevOps 30
Figura 4.6 – Exemplo tela inicial Portal do Corretor 31
Figura 4.7 – Exemplo Portal Corretor - Início 31
Figura 4.8 – Fluxo de *Branchs* 32
Figura 4.9 – Estrutura do Projeto 33
Figura 4.10 – Chamada EndPoint HTTP - Delete 34
Figura 4.11 – Estrutura do Projeto *Branchs* 35
Figura 4.12 – Postman 36
Figura 4.13 – Exemplo: Análise SonarCloud 37

SUMÁRIO

1	**INTRODUÇÃO**	8
2	**SOBRE A EMPRESA**	10
2.1	**Squadra Tecnologia**	10
2.1.1	**Equipe - Tribo Canais**	10
2.2	**Pottencial Seguradora**	11
3	**TECNOLOGIAS UTILIZADAS E CONCEITOS TÉCNICOS**	12
3.1	***Back-end***	12
3.2	***Front-end***	12
3.3	***Spotify Scaling Agile***	13
3.4	**S.O.L.I.D**	13
3.4.1	**O princípio da responsabilidade única (*Single responsability principle*)**	14
3.4.2	**Princípio do Aberto Fechado (*Open-Closed principle*)**	14
3.4.3	**Princípio da substituição de Liskov (*Liskov substitution principle*)**	14
3.4.4	**O princípio da segregação de interfaces (*Interface segregation principle*))**	15
3.4.5	**O princípio da inversão de dependências (*Dependency Inversion Principle*))**	15
3.4.6	**Considerações finais sobre os princípios S.O.L.I.D**	15
3.5	**O protocolo HTTP**	16
3.5.1	**HTTP Métodos**	17
3.6	**Padrão de projeto REST**	18
3.7	***Domain Driven Design* (DDD)**	19
3.8	**API - Application Programming Interface**	19
3.9	**Injeção de dependências**	20
3.10	**.NET**	20
3.11	**C#**	21
3.12	**JavaScript**	21
3.13	**React**	22
3.14	**Postman**	22
3.15	**Microsoft Azure DevOps Services**	23
3.16	**Swagger**	23

3.17 **SonarCloud** 23
3.18 **SQL-Server** 24
4 **ATIVIDADES DESENVOLVIDAS** 25
4.1 **Treinamento** 25
4.2 **Cursos** 25
4.3 **Portal do Corretor** 26
4.3.1 **Gerenciamento de tarefas, Automatização de processos e Controle de versão** . 26
4.3.1.1 **Gerenciamento de tarefas - Boards** 26
4.3.1.1.1 **Gerenciamento de tarefas - Sprints** 26
4.3.1.2 **Automatização de processos** 28
4.3.1.3 **Controle de versão** 29
4.4 **Desenvolvimento** 30
4.5 **Git - Branchs** 32
4.6 **Estrutura do Projeto** 32
4.7 **Testes de fluxo** 35
4.8 **Testes Unitários** 35
5 **CONCLUSÃO** 38
REFERÊNCIAS 39

1 INTRODUÇÃO

Este texto consiste em demonstrar as tecnologias e conceitos a respeito do desenvolvimento do back-end de uma aplicação Web.

A Squadra Tecnologia possui a missão de oferecer soluções inovadoras, feitas sob medida para que as necessidades de cada projeto sejam atendidas. Como uma fábrica de software, seu objetivo é combinar experiência e inovação, oferecendo suporte especializado em todas as etapas do projeto. Squadra Digital (2021) Em parceria com a Pottencial Seguradora, uma insurtechs — termo que mistura *insurance* (seguro) e *technology* (tecnologia) — que busca revolucionar o mercado conservador, desburocratizando a contratação de seguros se deu início o desenvolvimento de um site totalmente novo chamado Portal do Corretor. POTTENCIAL SEGURADORA (2021)

Visando empregar parte do conhecimento adquirido durante o curso de Bacharel em Ciência da Computação UNIVERSIDADE FEDERAL DE LAVRAS (2019) , este relatório descreve as atividades produzidas durante o período de estágio na empresa Squadra Tecnologia. As atividades executadas tem como princípio obter uma experiência prática totalmente voltada para o mercado de trabalho, focando no aprendizado de tecnologias, metodologias ágeis e conhecimento sobre o funcionamento de uma empresa internamente.

Já o Portal do Corretor teve como objetivo geral ser desenvolvido com tecnologias atuais, modelos e técnicas de engenharia de software confiáveis, seguras e de fácil manutenção. Utilizando o design orientado a domínio e arquitetura hexagonal, o projeto que pode ser considerado um serviço web utilizou também o padrão REST em todo seu fluxo *back-end*, visando uma maior desacoplamento e agilidade entre as APIs.POTTENCIAL SEGURADORA - PORTAL DO CORRETOR (2021)

Sendo assim, este documento descreve a rotina do autor, que atuou como desenvolvedor *back-end* para o cliente Pottencial Seguradora, participando diretamente no desenvolvimento do portal durante o período de junho de 2020 até junho de 2021.

Os capítulos posteriores estão organizados da seguinte maneira: O Capítulo 2 apresenta as empresas que fizeram parte do projeto, mostrando suas origens e seus principais valores. O Capítulo 3 busca explicar as principais tecnologias utilizadas no processo de desenvolvimento do portal e faz um embasamento teórico sobre alguns conceitos e arquiteturas. O Capítulo 4 visa apresentar as tarefas que o estágio participou e desenvolveu durante o período de estágio. Por fim o Capítulo 5 faz uma con-

clusão sobre o estágio, comparando os conhecimentos adquiridos com a base teórica proporcionada pelo curso de graduação, buscando levantar pontos fortes e fracos.

2 SOBRE A EMPRESA

Este capítulo tem o intuito de apresentar as duas empresas que juntas aceitaram o desafio de elaborar o Portal do Corretor. Buscando manter uma cultura ágil e gerenciando suas equipes, a fim de manter a máxima agilidade possível na solução de problemas.

2.1 Squadra Tecnologia

A Squadra Tecnologia foi fundada em 1992. Desde então atuam como sua atividade principal o desenvolvimento de softwares sob encomenda.

Atualmente a empresa tem em sua composição mais de 600 profissionais com conhecimento técnico para atender todas as etapas de construção e aplicação de projetos digitais. Apresenta projetos em diversas áreas, ao todo são mais de 100 organizações que já avançaram em suas jornadas de transformação digital.

Dentre seus principais valores está a garantia de melhores soluções tecnológicas para inovar, evoluir e transformar. Identificando a necessidade do cliente, a partir da sua realidade, propor as melhores soluções com confiança e verdade, tudo isso visando proporcionar um ambiente descontraído pautado pelo respeito. Squadra Digital (2021)

2.1.1 Equipe - Tribo Canais

A Tribo Canais atua tanto no *front-end* como no *back-end* e trabalha com o modelo ágil chamado de Spotify Scaling Agile - mencionado na seção 3.3 - a fim de manter a máxima agilidade possível na solução de problemas. A equipe contava com um time composto por funcionários da Squadra e da Pottencial, contendo funcionários *front-end*, *back-end*, analistas de teste e gerente de projeto, trabalhando juntos para conseguir o melhor resultado.

O projeto era de responsabilidade da sede em Belo Horizonte-MG, mesmo o estagiário residindo em Lavras-MG essa mudança de localização não foi um impedimento pois todo processo seria via Home-Office.

Apesar de todo o treinamento prévio ter sido focado em tecnologias de *front-end*, fui alocado para trabalhar com o *back-end*, utilizando C%, .NET, SQL-Server entre outras tecnologias.

A equipe mantinha uma rotina de reuniões diárias. Essas reuniões eram usadas para relatar o que havia sido feito e o que seria desenvolvido no dia, também era combinado que caso houvesse

algum impedimento no desenvolvimento o estagiário deveria avisar o time o mais rápido possível, para juntos buscarem uma solução e dar continuidade ao fluxo da tarefa.

2.2 Pottencial Seguradora

A Pottencial Seguradora foi fundada em 2010, a partir do conhecimento em garantia dos acionistas, diretores e colaboradores do então Banco Pottencial.

Desde então atuam no mercado de seguros, no qual foram ganhando espaço e em 2016 atingiram, pela primeira vez, a liderança de mercado no segmento de seguro garantia, durante os quatro primeiros meses do ano. E fecharam o ano de 2016 ocupando o 2º lugar, com aproximadamente 14% do mercado. POTTENCIAL SEGURADORA (2021)

Atualmente apresenta escritórios em Belo Horizonte, Campinas, Porto Alegre, Curitiba, Recife e Salvador e mantém a liderança em Seguro Garantia no mercado brasileiro pelo 3º ano consecutivo, com market share de 19,2%; a vice-liderança em Fiança Locatícia. POTTENCIAL SEGURADORA - GARANTIA (2021)

Hoje visa na digitalização do mercado de seguros, através do novo portal do corretor é possível que os corretores consigam incluir as demandas para cotação, solicitar cadastros, prorrogar boletos e consultar os CCGs. Existem também ferramentas gerenciais para a corretora, como a consulta de comissões pagas e à receber, além de fluxo sobre operações aprovadas. POTTENCIAL SEGURADORA - PORTAL DO CORRETOR (2021)

3 TECNOLOGIAS UTILIZADAS E CONCEITOS TÉCNICOS

Este capítulo tem como objetivo apresentar algumas tecnologias utilizadas durante o período de estágio, também apresentar alguns conceitos sobre arquitetura, padrões de projetos e metodologias ágeis.

3.1 *Back-end*

Para QUEIRÓS (2018) o *back-end* envolve toda a parte da programação voltada ao funcionamento interno de um software. Ou seja, back-end é tudo aquilo que está por trás da interface de uma aplicação: banco de dados, autenticação , segurança de dados, envio e recebimento de informações, etc. Ao contrário do *front-end*, que está mais voltado a tudo que diz respeito ao conteúdo e a parte visual de um site ou software, o *back-end* permite que tudo isso funcione de maneira eficiente ao construir sistemas que garantem estabilidade ao produto.

Também serve para dar suporte aos projetos de tecnologia, unindo uma equipe multidisciplinar com uma visão mais abrangente do que está por trás de todos os processos.

3.2 *Front-end*

Segundo EIS (2015) pode ser considerado toda parte da programação relativa à interface de uma aplicação, ou seja, toda parte dinâmica de um site com a qual um usuário pode interagir é desenvolvido por um desenvolvedor *front-end*. Em suma, como o próprio nome sugere, *front-end* é tudo aquilo que diz respeito à parte da frente de um site, aplicativo ou software; toda a aparência visível pelos seus visitantes.

Abrange todo o processo de construções de interface e utilização, seja de um site, aplicativo ou software. O desenvolvimento *front-end* é interpretado pelos navegadores ou sistemas operacionais que ajudam a transmitir os elementos para a tela, podendo ser utilizado e interpretado pelo usuário final. ROBBINS (2012)

3.3 *Spotify Scaling Agile*

Segundo MARK CRUTH (2019), a metodologia ágil conhecida como *Spotify Scaling Agile* surgiu quando houve a necessidade do Spotify[1] melhorar seu controle de processos em produção e manutenção da plataforma. A organização básica da metodologia é a matriz com times orientados vertical e horizontalmente. Verticalmente temos as *Squads* e as Tribos, que são agrupadas por produto ou grupos de features correlacionados. Os *Squads* são a unidade básica de organização dos times, usualmente em torno de uma feature, ou subsecção de uma funcionalidade. Contém cerca de 3 a 10 membros e devem ser autônomos a ponto de conterem expertise dentro do grupos os aspectos do produto para desenvolver tudo da concepção à prototipação, design, desenvolvimento, e deployment. A Tribo é o segundo nível de agrupamento, e pode conter uma série de *Squads* que tenham funcionalidades e objetivos similares. Os *Squads* de uma tribo ficam fisicamente próximos uns dos outros, para que haja comunicação fluida. Seu tamanho pode chegar até 100 pessoas e *Squads*-membro das tribos se encontram periodicamente para sincronizar seus esforços, apresentarem seu trabalho, e trocarem experiências. Já a horizontal tem os *Chapters e Guilds*, que são agrupadas por *skill* e interesse. *Chapter* é um grupo que congrega profissionais com responsabilidades e *skills* parecidos. As *Guilds* podem ser de *Squads* e tribos diferentes, que se unem para trocar experiências, aprendizados.

As vantagens dessa metodologia envolve agilidade nas tomadas de decisão e a colaboração e aprendizado entre os membros, pois o método não prende as pessoas em grupos fechados então sempre acontece a troca de conhecimentos. Mantendo a empresa sempre competitiva e moderna

3.4 S.O.L.I.D

S.O.L.I.D é um acrônimo para cinco postulados de design *"Single responsibility, open-closed, Liskov substitution, interface segregation and dependency inversion"*. Estes conceitos foram apresentados por Robert C. Martin DEURSEN (2019) como princípios da orientação a objetos. É importante notar que não são regras absolutas para o desenvolvimento. Em vez disso, são um guia para boas práticas que ajudam a evitar problemas durante e depois da elaboração do projeto, como mudanças é o caso de mudanças que desencadeiam grandes conflitos em várias camadas em projetos mal formulados. Nesta seção será feita uma breve explicação sobre cada um destes princípios.

[1] Disponível em <https://www.spotify.com/br/about-us/contact/>.

3.4.1 O princípio da responsabilidade única (*Single responsability principle*)

"Uma entidade do sistema deve ter um, e apenas um, motivo para mudar".

Então esse princípio sugere que pode existir apenas um requisito do sistema que, quando alterado, causará uma mudança em uma classe. Quando uma classe possui várias responsabilidades, qualquer pequena alteração no sistema pode acabar causando problemas inesperados em outras partes do sistema, dificultando muito sua manutenção.

Uma discussão importante seria sobre como definir uma responsabilidade, mas não existe uma regra universal para esta pergunta. O que o desenvolvedor deve ter em mente é que as responsabilidades devem ser definidas de acordo com a possibilidade de futuras mudanças no sistema. Por exemplo: as classes sofrem mudanças constantes? Podem surgir novos requisitos e mudanças de tecnologias? Duas responsabilidades devem ser separadas apenas quando uma mudança independente em cada uma delas é realmente um cenário possível em um horizonte futuro do sistema. Sendo assim, não existe uma regra pois separar responsabilidades apenas por separá-las, pode caracterizar uma complexidade desnecessária.

3.4.2 Princípio do Aberto Fechado (*Open-Closed principle*)

"Uma entidade do sistema deve ser aberta para extensões, mas fechada para modificações".

Isto significa que quando uma classe está pronta, não devemos mais alterá-la para adicionar novas funcionalidades. Então quaisquer alterações em termos de funcionalidade deve ser uma nova classe, que herda daquela que já existia. As linguagens de programação orientadas a objeto nos oferecem algumas ferramentas que auxiliam na implementação deste princípio, através da herança, classes abstratas e interfaces públicas. As alterações em uma classe existentes são permitidas caso ocorra bugs, mas nunca para estender suas funcionalidades.

3.4.3 Princípio da substituição de Liskov (*Liskov substitution principle*)

"Subtipos devem ser substituíveis pelos seus tipos base".

Esse princípio determina que qualquer objeto que herde de uma interface ou qualquer outra abstração deve implementar todas as funcionalidades de sua classe base de forma coerente, sem apresentar nenhum comportamento inesperado do ponto de vista do cliente.

3.4.4 O princípio da segregação de interfaces (*Interface segregation principle*))

"Os clientes não devem ser forçados a depender de métodos que não usem".

Na prática, o significado deste princípio diz que não devemos generalizar as interfaces criadas, buscando evitar que os clientes sejam obrigados a implementar métodos que não utilizarão. Esta discussão depende do projeto, dos requisitos e de uma análise das possibilidades para o futuro do sistema. O objetivo aqui é garantir que o cliente ou alguma classe que implemente uma interface não seja obrigado a escrever métodos que não utilizarão.

3.4.5 O princípio da inversão de dependências (*Dependency Inversion Principle*))

"Módulos de alto nível não devem depender de módulos de baixo nível. Ambos devem depender de abstrações".

"Abstrações não devem depender dos detalhes da implementação. Os detalhes devem depender das abstrações".

Os módulos de alto nível do sistema são aqueles que são responsáveis pelas regras de negócio. Por isso, é essencial que eles não sejam impactados por quaisquer alterações feitas nos chamados módulos de baixo nível - interfaces de usuários, detalhes de implementações, camadas de dados, etc.

Na figura 3.1, "Sem o princípio de inversão de dependência", o Objeto A no Pacote A refere-se diretamente ao Objeto B no Pacote B. Já o " Com princípio de Inversão de Dependência", uma Interface A é introduzida como uma abstração no Pacote A. O Objeto A agora se refere à Interface A e o Objeto B herda de Interface A. Ou seja, tanto o Objeto A quanto o Objeto B agora dependem de uma abstração, houve uma inversão de dependência que existia do Objeto A para o Objeto B, sendo que o Objeto B agora dependente exclusivamente da Interface A.

3.4.6 Considerações finais sobre os princípios S.O.L.I.D

É importante ter em mente que estes princípios não são regras rígidas que devem ser seguidas a todo tempo.

Mas no projeto desenvolvido durante o estágio, os princípios S.O.L.I.D foram amplamente utilizados. Pois existiu uma análise onde diversas variáveis foram levadas em consideração, uma delas foi o crescimento da aplicação ao longo do tempo e a busca por uma fácil manutenção de código, buscando um menor desacoplamento. Em um sistema que não tende a evoluir e crescer demais,

Figura 3.1 – S.O.L.I.D - Princípio da inversão de dependências

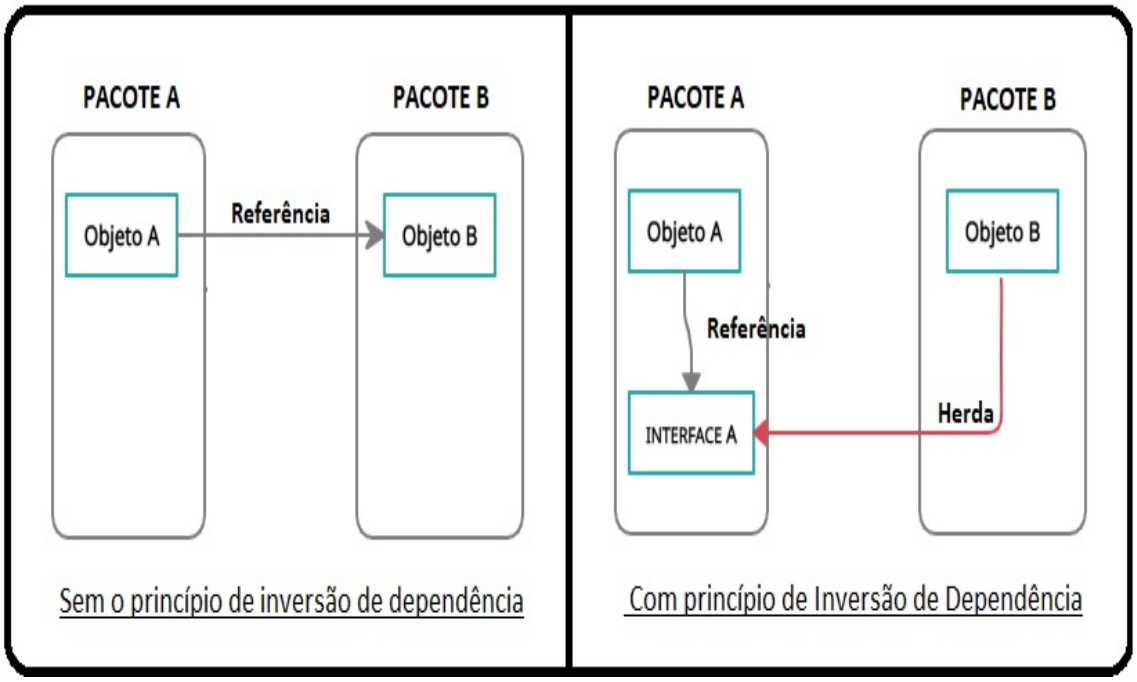


Diagrama demonstrando o princípio da Inversão de Dependências

os benefícios trazidos pelos princípios S.O.L.I.D podem muitas vezes não compensar o aumento no tempo de desenvolvimento e da complexidade do sistema. Assim como em outras áreas da Engenharia, deve-se sempre considerar estes trade-offs entre os benefícios e os custos do desenvolvimento.

3.5 O protocolo HTTP

De acordo com JOÃO ERIBERTO M.F. (2013) o protocolo HTTP é um protocolo da camada de aplicação de redes. Ele define regras e convenções utilizadas para a comunicação entre um cliente e um servidor. Para isso, se baseia no conceito de requisições (requests) e respostas (responses).

A Figura 3.2 demonstra a estrutura de uma mensagem HTTP. Sendo a primeira linha uma chamada de requisição, onde contém as informações do Verbo HTTP mais a URI para onde a requisição será enviada. Em seguida, temos as linhas de cabeçalho que são chamadas de Headers. Elas são uma coleção de chave/valor que podem conter informações sobre o navegador do cliente e outras informações. Em seguida, pode ser definido caso necessário o corpo da mensagem. Ele contém a entidade que se deseja transmitir.

Figura 3.2 – HTTP - Protocolo

Mensagem HTTP

3.5.1 HTTP Métodos

O protocolo HTTP define 8 métodos principais que são responsáveis por indicar a ação a ser executada para um dado recurso, são eles:

GET - Utilizado para obter quaisquer informações que são representadas pela URI informada na requisição.

POST - É utilizado para enviar informações para o servidor, usa o corpo da requisição para envio da entidade. Um exemplo de utilização é para fornecer os dados de um cadastro para o servidor.

PUT - Utilizado para fazer alterações. Mas caso não exista pode ser gerado um novo recurso, ele permite que este recurso seja criado e armazenado nesta URI.

DELETE - Utilizado para solicitar a exclusão do recurso.

HEAD - É idêntico ao método GET, porém a resposta não deve conter nenhuma informação no corpo da mensagem. Apenas os metadados contidos nos headers da mensagem serão enviados na resposta para o cliente.

TRACE - O servidor irá apenas refletir a mensagem de volta para o cliente. É utilizado para testes e diagnósticos, para saber o que está sendo recebido pelo servidor.

OPTIONS - É utilizado para obter informações sobre as opções de comunicações disponíveis na URI contida na requisição.

PATCH - Consiste em uma funcionalidade para modificação parcial.

MARK MASSE (2011)

3.6 Padrão de projeto REST

Representational State Transfer (REST) é descrito por MARK MASSE (2011) como uma aplicação de arquitetura de comunicação baseada no modelo web services, no qual um serviço/API é fornecido para consumo de forma independente. Em outras palavras, REST garante que a troca de dados entre cliente e servidor seja feita de forma atômica, sem acoplamento entre as partes. REST nos propõe uma interface muito mais simples do que o SOAP, porém com algumas limitações. Desenvolver utilizando REST é mais fácil e mais rápido e esta é a razão que as aplicações REST são frequentemente utilizadas para o desenvolvimento de serviços voltados a web ou dispositivos móveis. A aplicação REST utiliza HTTP como protocolo padrão para a comunicação de dados entre o Sistema de gerenciamento de banco de dados (SGBD) com aplicações de terceiros, retornando uma resposta em formato tipo texto, que pode ser tanto um formato de dados chamado JSON ou XML. A relação REST com HTTP também pode ser considerada um limitante, pois se estivermos projetando uma arquitetura orientada a serviços que precise interligar vários sistemas com operações sofisticadas e modeladas com BPMs, o padrão REST pode não ser a melhor opção.

Mas para padrões web, ela garante que possamos construir um sistema no qual nos preocupamos apenas com os métodos que manipulam os dados, e depois decidimos quem vai usar estes dados. Com isso, nosso "núcleo" que manipula dados está invariável em relação às constantes mudanças de tecnologia em relação ao cliente. O desenvolvimento de aplicações webservice com base em REST tem muitas vantagens:

- A alteração de serviços no aprovisionamento da web REST não requer qualquer alteração no código do lado do cliente.
- REST é definitivamente leve, uma vez que se destina à transferência de dados sobre um interface conhecida, Uniform Resource Identifier (URI).

- As APIs Restful podem ser consumidas utilizando simples pedidos GET, servidores proxy intermediários / reverseproxies podem armazenar a sua resposta muito facilmente.
- Suporta diretamente todos os tipos de dados.
- REST, adiciona um elemento de utilização de URIs padronizados, e também dá importância ao verbo HTTP/Métodos
- Assume um modelo de comunicação ponto-a-ponto, ao contrário de ambientes de computação distribuída onde a mensagem pode passar por um ou mais intermediários.
- Muito mais simples de desenvolver serviços web do que o SOAP.

3.7 *Domain Driven Design* (DDD)

O Projeto Orientado ao Domínio, ou *Domain Driven Design* (DDD) , é um conceito apresentado por Eric Evans em seu livro de mesmo nome, lançado em 2003 ERIC EVANS (2003). Essa modelagem de software que segue um conjunto de práticas com objetivo de facilitar a implementação de complexas regras e processos de negócios que tratamos como domínio.

O DDD coloca uma grande ênfase em manter a integridade conceitual do modelo da aplicação, o que é obtido pela combinação de diversos fatores, como um processo ágil o qual tem ênfase no feedback dos usuários e pessoas com maior experiência no determinado domínio, bem como a disponibilidade destas pessoas experientes e uma colaboração criativa com as mesmas.

A principal ideia do DDD é a de que o mais importante em um software não é o seu código, nem sua arquitetura, nem a tecnologia sobre a qual foi desenvolvido, mas sim o problema que o mesmo se propõe a resolver, ou em outras palavras, a regra de negócio. Como já diz o título do livro que deu origem ao padrão DDD de ERIC EVANS (2003), esse é o "coração", o ponto central de qualquer aplicação, portanto todo o resto deve ser trabalhado de forma que este "coração" seja entendido e concebido da melhor forma possível.

3.8 API - Application Programming Interface

De acordo com EDUARDO PIRES (2021) a interface de programação de aplicativos é o que expõe as funcionalidades de um sistema. Ela é uma interface que permite a troca de informações entre

os sistemas, independentemente de suas tecnologias ou detalhes das implementações. Uma aplicação envia uma requisição à API, informando o que ela deseja fazer. Pode ser uma consulta aos dados do sistema, um cadastro, atualização de dados, etc. Por exemplo, em uma API de cadastramento de pessoas, inicialmente pode ser solicitado uma chamada para a criação do usuário. A partir daí, a API irá retornar uma resposta à aplicação, indicando se o processamento da requisição foi feito com sucesso ou não, podendo retornar uma mensagem de erro, ou algum dado de identificação de usuário criado.

3.9 Injeção de dependências

É necessário que em um projeto real os objetos interajam entre si para que um sistema possa executar uma operação ou uma lógica de negócios. Porém, tradicionalmente cada objeto é responsável por definir e instanciar todas as suas dependências - os objetos do sistema com os quais ele interage. O problema desta lógica é que ela aumenta muito o acoplamento, dificultando testes e realização de uma classe de forma isolada de suas dependências.

O padrão de Injeção de Dependências foi criado exatamente para dar uma forma de isolar um objeto de todas as suas dependências, reduzindo o acoplamento do sistema. Basicamente, a ideia consiste em trazer a definição das dependências dos objetos para fora de suas classes e injetá-los nestes objetos para a utilização. Isto significa que os objetos não serão responsáveis por criar ou obter as suas dependências. Em vez disso, eles a receberão a partir da injeção.

Existem várias formas de se executar este padrão, mas uma forma simples de demonstrar o padrão é através da injeção feita pelos construtores. Neste caso, toda a parte da definição dos serviços seria externalizado e centralizado em um único local. Toda classe que tiver alguma dependência deverá possuir um construtor que espera as interfaces de todas estas dependências, por onde elas serão injetadas. ROBERT C. MARTIN (2019)

3.10 .NET

O .NET segundo a MICROSOFT (2020) é uma plataforma de desenvolvimento gratuita e de software livre, pode ser usado no desenvolvimento de aplicativos web, aplicativos móveis, aplicativos de desktop e muitos outros. Apresenta código aberto e alta participação da comunidade de desenvolvedores, utilizando licenças MIT e Apache 2 e é um projeto do grupo .NET Foundation. É atualizado

regularmente visando manter a segurança e qualidade e é suportado pela Microsoft Windows, macOS e Linux.

As distribuições binárias do .NET da Microsoft são criadas e testadas em servidores mantidos pela Microsoft no Azure e seguem as práticas de segurança e engenharia da Microsoft. O .NET dá suporte a três linguagens de programação, sendo elas o C#, que é uma linguagem de programação moderna, orientada a objeto; F# que é uma linguagem que suporta paradigma funcional, orientado a objetos e imperativo e Visual Basic. (MICROSOFT, 2020)

3.11 C#

Pode ser caracterizada de acordo com a MICROSOFT (2021) como uma linguagem de programação moderna, orientada a objeto e de tipo seguro. A linguagem permite ao desenvolvedor criar aplicações seguras e de grande escala que podem ser executadas no .NET. Pertence a família de linguagens C e também apresenta semelhanças com C++, Java e JavaScript.

Por apresentar bastante recursos, a utilização do C# vem sendo muito visada em aplicações robustas, que buscam uma maior durabilidade e confiabilidade. Seu garbage collector recupera automaticamente a memória ocupada por objetos não utilizados. A manipulação de exceção fornece uma abordagem estruturada e extensível, que visa uma melhor detecção e recuperação de erros. Apresentam tipos anuláveis que protegem contra variáveis que não se referem a objetos alocados. A sintaxe de linguagem de consulta integrada gera um padrão comum para trabalhar com dados de qualquer fonte e seu suporte a idiomas para operações assíncronas fornece a sintaxe para a criação de sistemas distribuídos. Valores de qualquer tipo podem ser armazenados, transportados e operados de maneira consistente. A linguagem dá suporte a tipos de referência definidos pelo usuário e tipos de valor, além de permitir a alocação dinâmica de objetos e o armazenamento em linha de estruturas leves, oferecendo suporte a tipos e métodos genéricos, que fornecem aumento na segurança e no desempenho do tipo.

3.12 JavaScript

É uma linguagem de programação utilizada principalmente em páginas web. Atualmente é uma das mais importantes tecnologias voltadas para o *front-end* e, junto ao trio HTML, CSS e PHP, formam um grupo de linguagens que suprem todas as exigências do desenvolvimento de uma página

completa, dinâmica e com boa performance. JavaScript permite implementar itens complexos em páginas web e com seus scripts é possível incluir, em uma página estática, elementos dinâmicos como mapas, formulários, operações numéricas, animações, infográficos interativos e muito mais.

Segundo a Netscape Communications Corporations, empresa responsável pela criação do JS, "JavaScript é uma linguagem de programação, leve, interpretada, orientada a objetos, baseada em protótipos e em first-class functions (funções de primeira classe), mais conhecida como a linguagem de script da Internet.

STOYAN STEFANOV (2010)

3.13 React

React.js é uma biblioteca JavaScript de código aberto que é usada para construir interfaces de usuário especificamente para aplicativos single-page. É utilizado no *front-end* para exibição em aplicativos web e móveis. O React também nos permite criar componentes de Interface do Usuário reutilizáveis. React foi criado pela primeira vez por Jordan Walke, um engenheiro de software que trabalha para o Facebook.

O React permite que os desenvolvedores criem grandes aplicativos web que podem alterar dados, sem a necessidade de recarregar a página. O objetivo principal do React é ser rápido, escalável e simples. Também permite que você reuse componentes que tenham sido desenvolvidos em outras aplicações e que usem a mesma função. A função de reusabilidade é uma vantagem importante para desenvolvedores em geral.

Maurício S. S. (2021)

3.14 Postman

O Postman é um aplicativo que apresenta o objetivo de testar e desenvolver APIs em uma interface bastante simples e intuitiva. Ele nos permite simular requisições HTTP (get,put,post,delete...) de forma rápida, armazenando-as para que possamos usá-las posteriormente.

Além disso, com o postman é possível analisar cada requisição feita e as respostas enviadas pela API, além de exibir visualmente de forma muito agradável e fácil de entender, ele serve como um facilitador reduzindo consideravelmente o tempo necessário para o desenvolvimento e testes de sua aplicação. Também é possível compartilhar as configurações de testes de uma API com outros

membros da equipe, e até mesmo copiar requisições feitas dentro do próprio navegador para dentro da API.

3.15 Microsoft Azure DevOps Services

A plataforma de nuvem Azure consiste na execução de aplicativos e serviços, baseada nos conceitos da computação em nuvem da Microsoft. Ele concentra toda a infraestrutura de cloud computing da empresa, incluindo os seus produtos, serviços e aplicativos. Com o Azure é possível implantar e desenvolver apps diversos, além de conter várias ferramentas de auxílio para gerenciamento e manutenção com uma interface bem amigável. Também podem ser migrados aplicativos, servidores e banco de dados para a nuvem da Microsoft.

O Microsoft Azure tem fortes camadas de segurança que protegem os usuários das mais variadas ameaças. Apresentam proteção contra ataques DDoS, que não são classificados exatamente como uma invasão ao sistema, mas como uma forma de impedir que seus recursos sejam acessados. Todas as operações realizadas no Microsoft Azure também são altamente protegidas. Além de ser possível gerenciar identidades e controlar os acessos, adotando as melhores práticas do mercado. Dessa maneira, é possível garantir que somente as pessoas certas consigam acessar os dados da empresa.

3.16 Swagger

Swagger é um framework para descrição, consumo e visualização de serviços RESTful. Sua principal função é permitir que a documentação possa evoluir no mesmo ritmo da implementação, já que pode ser gerada automaticamente com base em anotações do código.

O Swagger apresenta um módulo de UI que permite aos desenvolvedores interagirem com as APIs em sandbox de forma muito intuitiva, sem exigir conhecimento da implementação ou mesmo dos parâmetros. Equipes também podem utilizá-lo para estabelecer contratos de request e response, esses contratos pré-estabelecidos ajudam na implementação e ajudam para que siga de forma mais limpa e não seja necessário muitas modificações durante a integração.

3.17 SonarCloud

É um serviço usado para detectar bugs e vulnerabilidades de segurança em suas solicitações pull e em todos os seus repositórios de código.

O SonarCloud combina implantação contínua com fluxos de trabalho CI/CD baseados na nuvem existente e fornece orientação de resolução para qualquer problema de qualidade ou segurança de código detectado. Voltado para capacitar equipes de desenvolvimento de todos os tamanhos para escrever códigos mais limpos e seguros.

3.18 SQL-Server

O SQL-Server Server é uma plataforma de amplo domínio em SGDB e fornece recursos de gerenciamento de dados de classe empresarial com ferramentas de BI (Business Intelligence) integradas. O mecanismo de banco de dados do SQL-Server oferece um armazenamento tanto para dados relacionais quanto estruturados, permitindo criar e gerenciar aplicativos de dados altamente disponíveis e eficientes para uso em seus negócios (MICROSOFT,2009).

Pode ser amplamente utilizado tanto em ambientes voltados a grandes empresas e sites Web comerciais, enviando milhões de transações por dia ou até mesmo em pequenas empresas e aplicativos individuais e especializados que necessitam de um armazenamento de dados resistente e um serviço persistente.

4 ATIVIDADES DESENVOLVIDAS

Este capítulo descreve as principais atividades desenvolvidas durante o estágio na empresa Squadra. As seções deste capítulo estão organizadas de acordo com as principais atividades: treinamento e projeto.

4.1 Treinamento

Após o processo de seleção para o programa de estágio New Tinkers os estagiários passaram por um processo de treinamento com duração de um mês. Este processo teve como objetivo mostrar tecnologias bases que seriam empregadas durante todo o período de estágio.

O primeiro dia foi voltado para o conhecimento estrutural da Squadra, onde o gerente da sede de Lavras buscou apresentar os projetos em desenvolvimento e os principais valores da empresa.

Em seguida se deu início a um treinamento com tecnologias bases que são de praxe da empresa, buscando já preparar o estagiário para as futuras atividades que iriam desenvolver. Fomos apresentados ao sistema de controle de versão GIT e foi disponibilizado uma conta no GitLab para armazenamento e manutenção dos códigos gerados durante o período de treinamento. O conceito de modelagem de software também foi um dos assuntos abordados, padrões como S.O.L.I.D e DDD foram apresentados visando que o estagiário pudesse gerar códigos com maior qualidade, foi dito sobre a importância dos testes unitários e que seriam indispensáveis durante os projetos, após cada explicação sempre era passado exemplos e exercícios para melhor absorção e entendimento. Por fim, os estagiários receberam um preparo referente a culturas ágeis e plataformas que auxiliam nesse papel, como é o caso do Azure.

4.2 Cursos

Após o Onboarding, o estagiário realizou cursos com foco em desenvolvimento web.

Durante o curso de React na plataforma Cod3r[1], com duração de 60 horas, foi abordado a base de componentes, módulos, diretivas, serviços, injeção de dependências, entre outros.

Já o curso de JavaScript também realizado na plataforma Cod3r, com duração de 40 horas, abordou desde variáveis e operações na linguagem, como também validação de formulários, filtragem de tabelas e boas práticas em JavaScript.

[1] Disponível em <https://www.cod3r.com.br/collections>.

Por fim, após o treinamento prévio, os estagiários foram designados a uma equipe/projeto.

4.3 Portal do Corretor

As próximas atividades relatadas foram executadas na equipe nomeada Tribo Canais, responsável pelo Portal do Corretor. As próximas sessões tem como objetivo demonstrar algumas atividades e rotinas exercidas pelo estagiário durante o período que atuou no desenvolvimento do projeto.

4.3.1 Gerenciamento de tarefas, Automatização de processos e Controle de versão

Durante o estágio uma das principais ferramentas utilizadas foi o Azure DevOps, que tinha como foco 3 principais objetivos:

4.3.1.1 Gerenciamento de tarefas - Boards

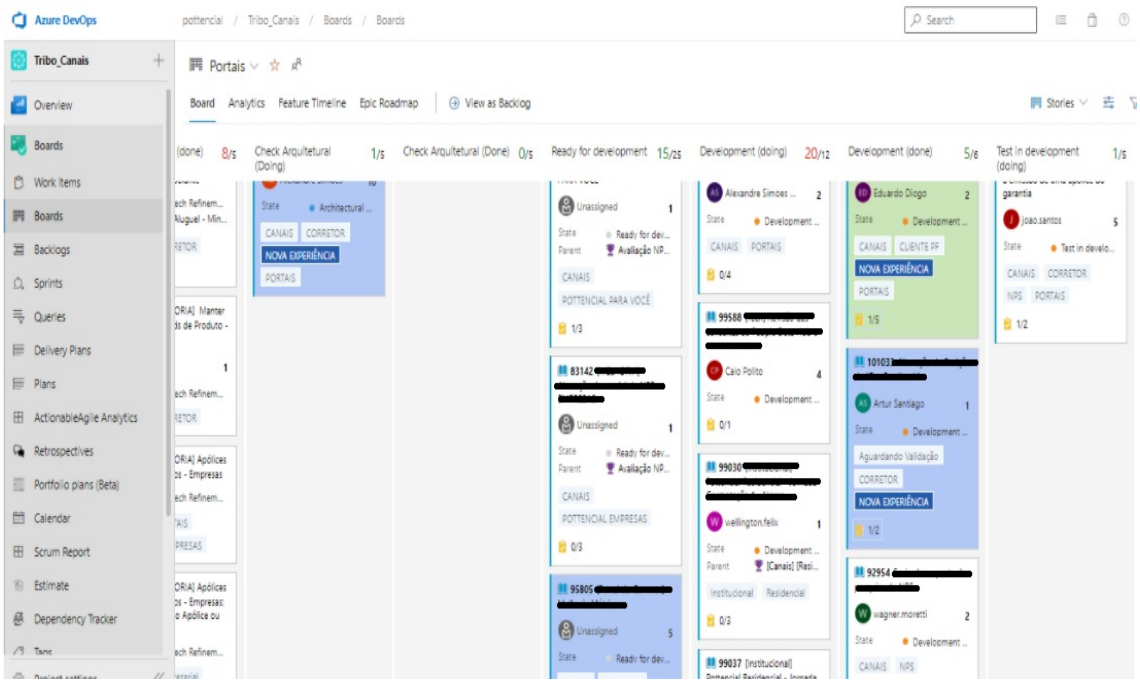
Gerenciamento de tarefas, utilizando o Azure Boards o estagiário consegue consultar as tarefas que estavam direcionadas a ele, além de também poder visualizar a história - template da página e regras de negócio - que contém os dados necessários para que a tarefa seja executada. Nele fica toda a cadeia de criação do projeto, desde a parte do desenvolvimento base da história, check arquitetural, desenvolvimento, testes, homologação e finalmente produção. Cada etapa precisa rigorosamente passar por todos esses processos para que ao fim seja dada como válida.

A Figura 4.1, mostra a tela do Azure DevOps - Boards, com ênfase nas 3 principais colunas de responsabilidade dos desenvolvedores, que são as *"Ready for development", "Development("Doing"), "Development(Done)"*. A primeira mantinha histórias que estavam disponíveis para começarem a ser desenvolvidas, a segunda contém as histórias que alguma tarefa havia sido puxada para elaboração e por fim a terceira coluna mantinha tarefas concluídas. Após a última etapa de responsabilidade dos desenvolvedores, vinha as colunas de testes, para que as histórias que haviam sido feitas fossem validadas.

4.3.1.1.1 Gerenciamento de tarefas - Sprints

Dentro do Boards também se faz o uso de um quadro Kanban na aba de Sprints, fornecendo um fluxo de trabalho visual e interativo, onde o funcionário movimenta a tarefa para mostrar com o que está trabalhando e se existe algum impedimento. O quadro continha quatro colunas: A fazer,

Figura 4.1 – Boards Azure DevOps



Tela responsável pelo fluxo de histórias, passando por todas etapas do desenvolvimento

Em desenvolvimento, Corrigido e Pronto. O gestor determinava a quantidade de horas destinada para cada tarefa, mas era livre qualquer alteração desse prazo, servia apenas como métrica para análise no andamento semanal. Era exigido que apenas uma tarefa estivesse na coluna ”Em desenvolvimento” por vez, se caso fosse necessário mudar de tarefa a atual deveria ser movida de volta à coluna “A fazer”. Já após conclusão a tarefa era movida para a coluna “Pronto”, caso necessário a coluna "Correção"era utilizada se fosse identificado algum erro na tarefa pronta, então a tarefa voltava para “A fazer” e após concluída ficava em “Corrigida” até que fosse testada novamente e liberada pelo analista de testes.

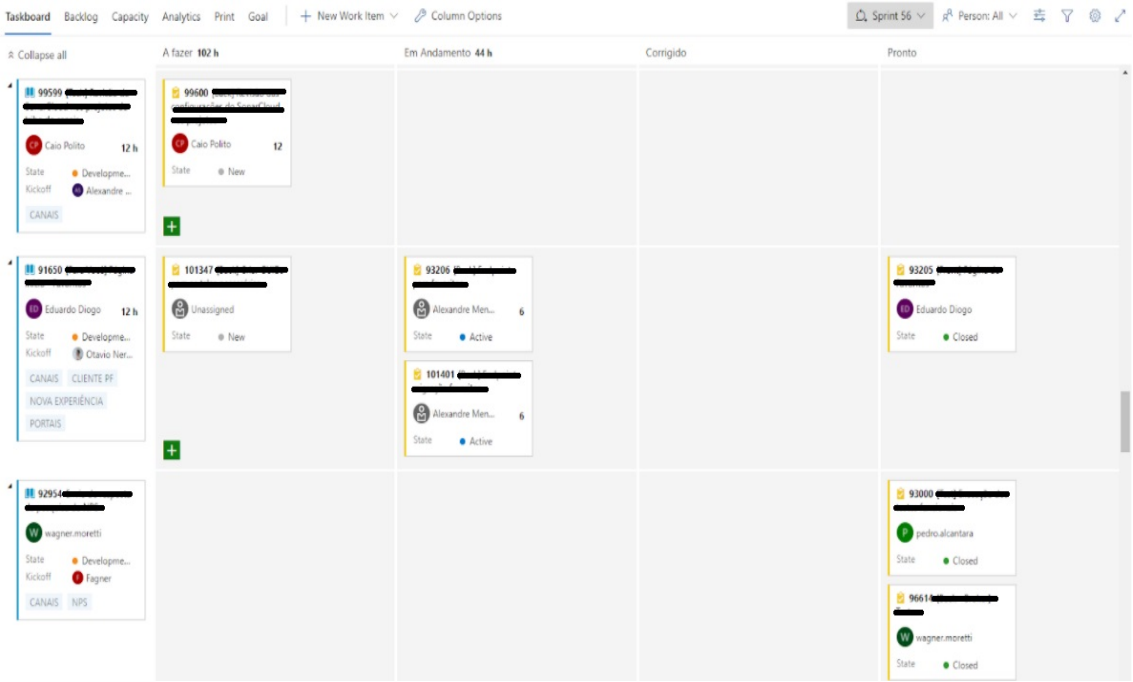
Impedimentos que são problemas que causam o travamento parcial ou total de uma tarefa, deveriam ser relatados o mais breve possível e eram tratados com prioridade, visando sempre resolvê-los o mais rápido possível para que o fluxo da tarefa pudesse continuar.

As sprints tinham duração de uma semana, sendo que toda sexta existia o planejamento semanal e uma revisão, essa revisão semanal tinha como objetivo mostrar para o cliente o que havia sido desenvolvido durante a semana, essa apresentação era mais focada no *front-end*, onde poderia ser avaliado os templates e se era necessário alguma mudança ou correção. O objetivo era buscar fazer essa apresentação com o maior nível de fidelidade possível, visando que o *front-end* estivesse integrado

com o *back-end* e não utilizando na tela dados falsos. Tarefas que por algum motivo não tivessem sido concluídas, eram redirecionadas a próxima sprint e recebiam uma maior atenção.

A Figura 4.2 mostra o quadro de projetos organizado por colunas, seguindo o modelo *Kamban*. Essa abordagem permite às equipes visualizar melhor a sua carga e fluxo de trabalho.

Figura 4.2 – Sprints Azure DevOps



Tela que contém as tarefas para aquela sprint, seguindo a metodologia kanban

4.3.1.2 Automatização de processos

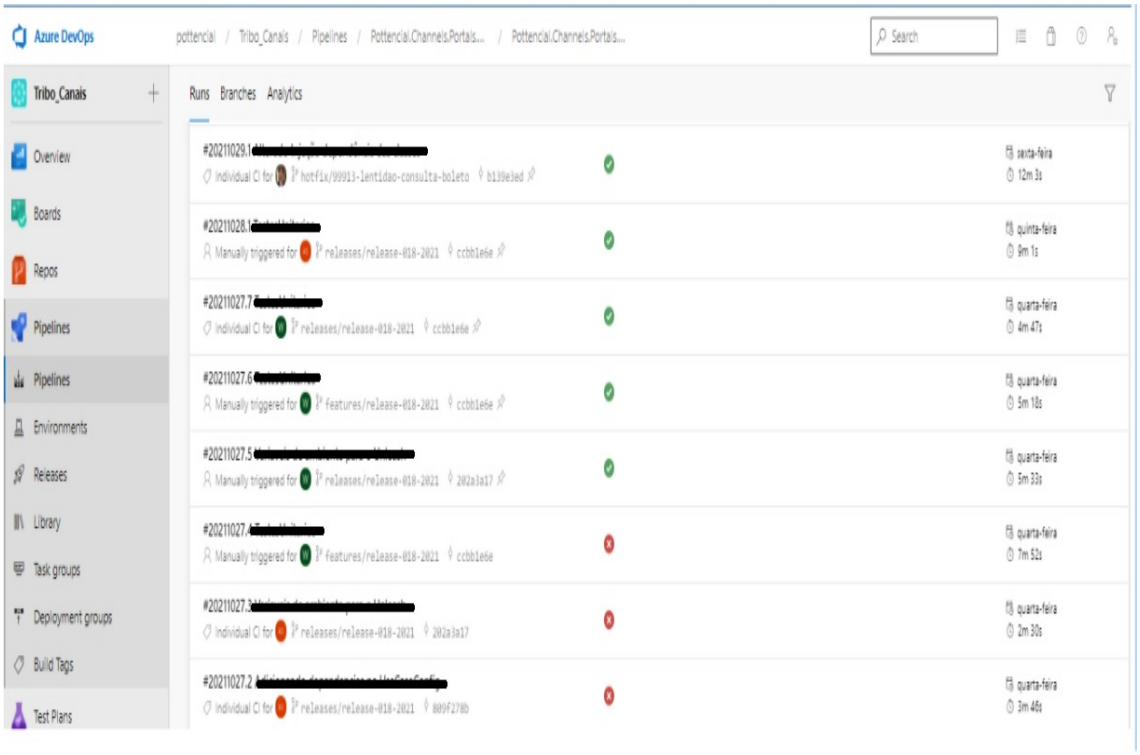
O Azure Pipelines cuidava da automatização de processos, ajudando a economizar tempo com detalhes técnicos e era utilizado para compilação, testes e implantação contínua com CI/CD.

O processo de CI bem-sucedido é quando novas mudanças no código de uma aplicação são desenvolvidas, testadas e consolidadas regularmente em um repositório compartilhado. No caso da nossa equipe existiam vários desenvolvedores ao mesmo tempo, então é a solução ideal para evitar conflitos entre ramificações.

Já o CD se refere à implantação contínua e trata da automação de fases avançadas do pipeline, mas são usados às vezes separadamente para ilustrar o nível de automação presente. Geralmente representa as mudanças feitas pelo desenvolvedor em uma aplicação, que são automaticamente testadas

contra erros e carregadas no repositório Azure. Nesse repositório, a equipe de operações implementa mudanças em um ambiente de produção ativo. Isso resolve o problema de baixa visibilidade e comunicação entre as equipes de negócios e desenvolvimento. Para isso, a finalidade da entrega contínua é garantir o mínimo de esforço na implantação de novos códigos. Tendo basicamente dois ambientes de teste, Development e HML, após aprovação nessas duas etapas e feita a devida homologação e as alterações são liberadas para publicação no ambiente de produção.

Figura 4.3 – Exemplo Pipelines Azure DevOps



Fonte: https://azure.microsoft.com/pt-br/services/devops/pipelines/

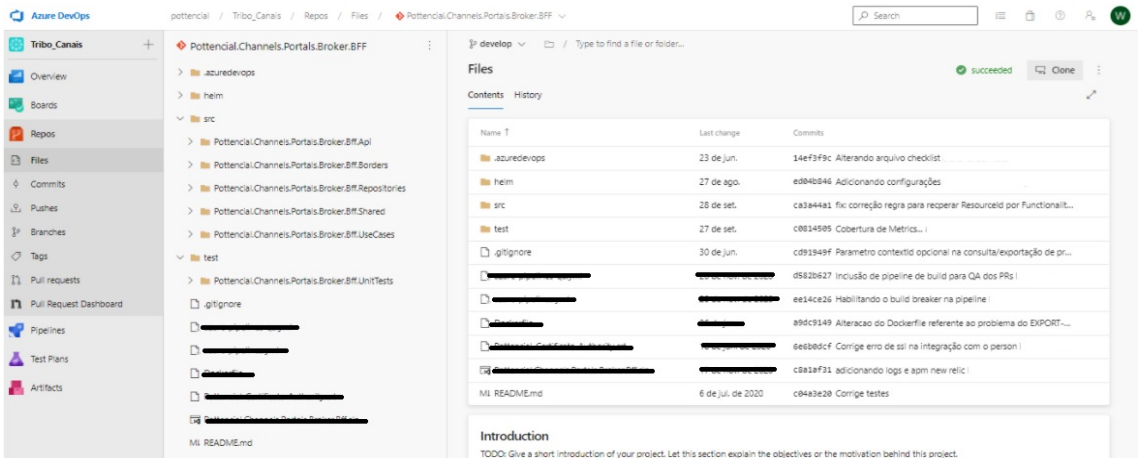
4.3.1.3 Controle de versão

Para uma manutenção avançada de arquivos era utilizado o Azure Repos, que oferecia uma melhor construção e união de códigos.

O processo de merge com a branch da sprint semanal - develop - também era através do Azure Repos, o funcionário poderia criar a solicitação de merge entre as branches, assim o pipeline era executado e verificava se existia algum problema, caso não houvesse ainda era necessário a aprovação de pelo menos outros dois desenvolvedores para que o pull request fosse mergiado para a develop.

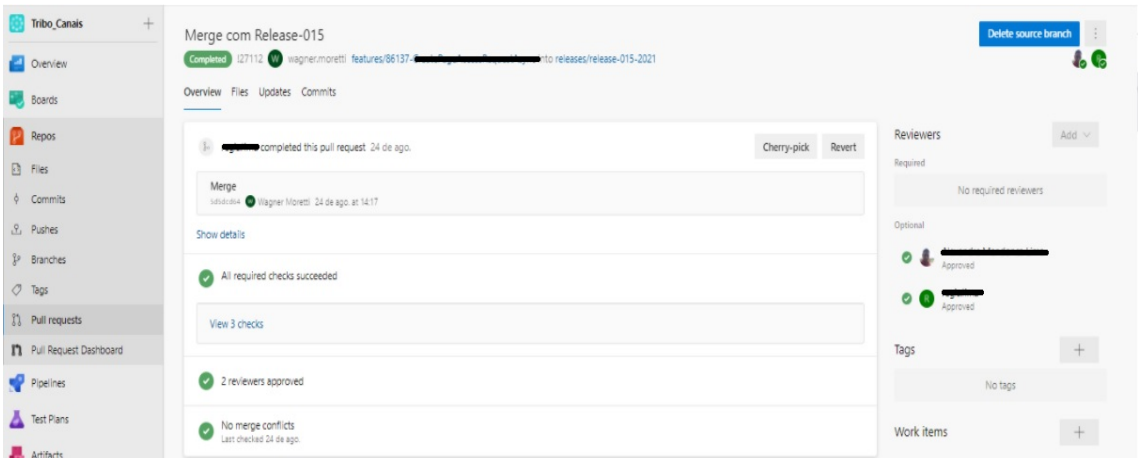
Além de ser responsável por oferecer hospedagem privada, ilimitada e em nuvem de repositórios Git e suporte para Controle de Versão.

Figura 4.4 – Exemplo Repos Azure DevOps



Fonte: https://azure.microsoft.com/en-us/services/devops/repos/

Figura 4.5 – Exemplo Repos-PullRequest Azure DevOps



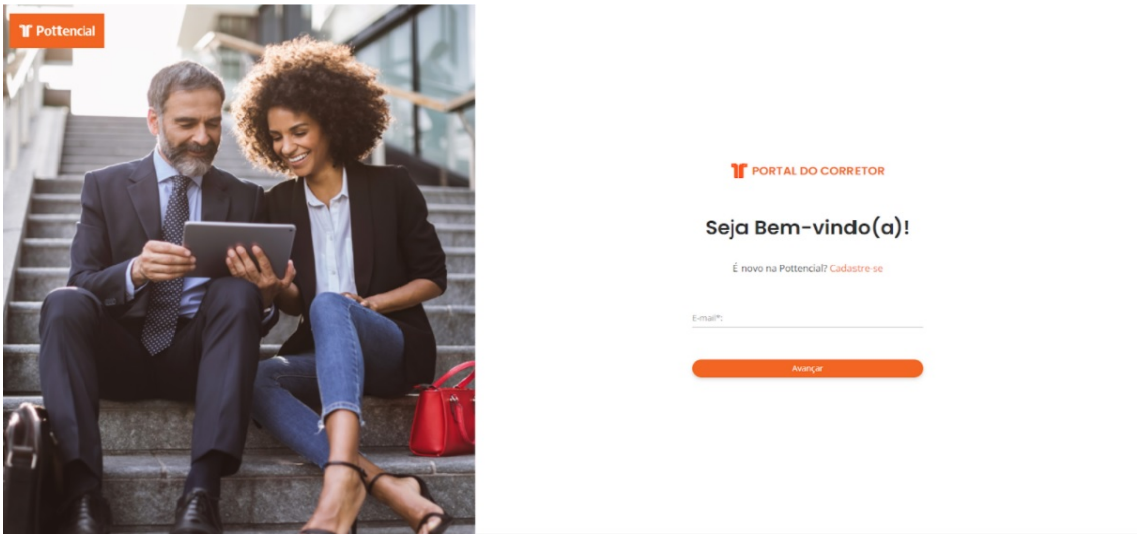
Fonte: https://azure.microsoft.com/en-us/services/devops/repos/

4.4 Desenvolvimento

Como desenvolvedor .NET, atuei principalmente na criação de fluxos conhecidos como endpoints, esses fluxos tinham como objetivo fornecer um meio padronizado e imutável de comunicação entre o cliente e o servidor, através de chamadas que utilizavam o protocolo HTTP com recursos URI e respostas em JSON. O ambiente principal de desenvolvimento era o Broker.bff, API criada na versão

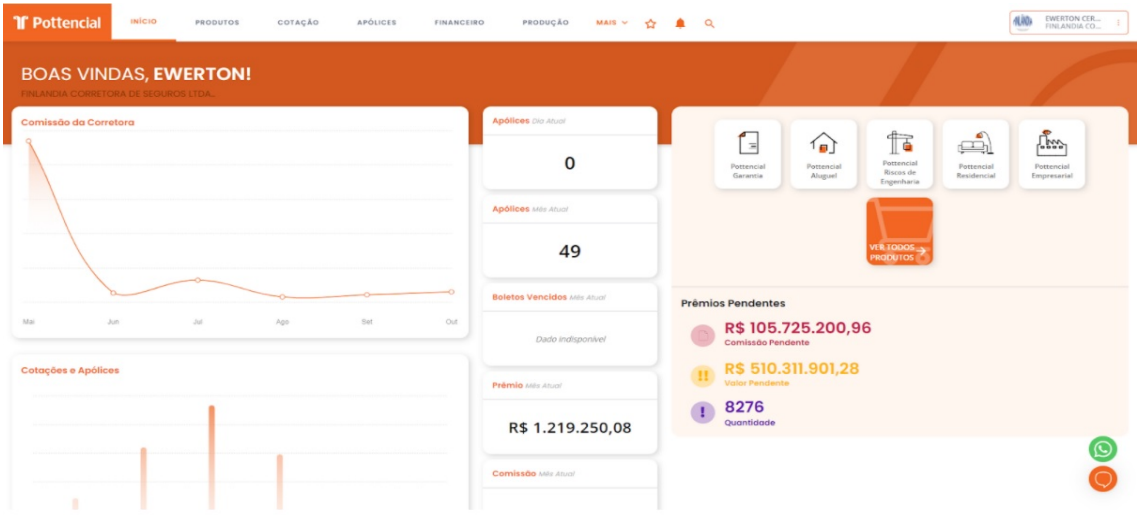
.net core 3.1, e era responsável por todos os endpoints que após integrados com a Api do *front-end* chamada de Broker.api, compunham o portal do corretor. As figuras 4.7 e 4.6 mostram o template produzido pela equipe de front-end e totalmente integrado com o Broker.API aplicação back-end. As figuras tem como objetivo mostrar um pouco do portal e das funcionalidades que ele apresenta.

Figura 4.6 – Exemplo tela inicial Portal do Corretor



Fonte: https://portalcorretor.pottencial.com.br/login/autenticacao

Figura 4.7 – Exemplo Portal Corretor - Início

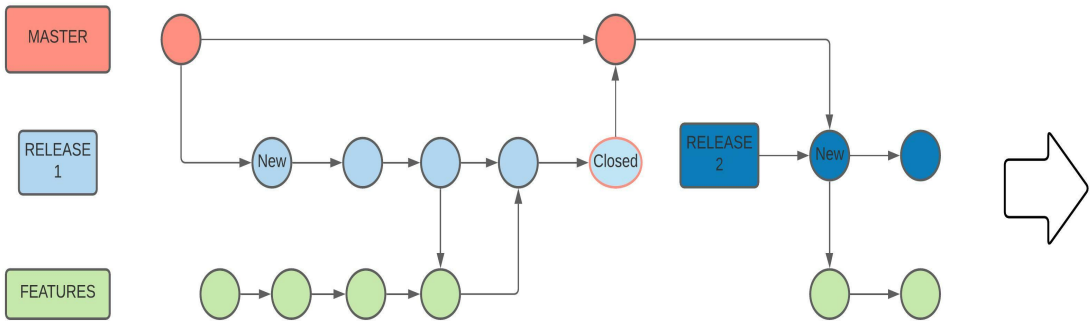


Fonte: https://portalcorretor.pottencial.com.br/login/autenticacao

4.5 Git - Branchs

Para manter um código organizado e seguro, além da branch Master tínhamos a Release, que mantinha o papel de permanecer ativa até o momento que algum código que estivesse nela entrasse no período de homologação. Para evitar conflitos e impedir que novos códigos ainda não testados fossem adicionados a branch que seria enviada a homologação, ela era fechada e aguardava os trâmites finais até o merge com a master e entrada em produção. Sendo assim o trabalho não era prejudicado, e novos códigos poderiam ser inseridos na nova release e publicados no ambiente de desenvolvimento para que fossem inicialmente testados. Esse processo também visava garantir que o código instável não fosse mesclado nos arquivos do projeto principal.

Figura 4.8 – Fluxo de *Branchs*



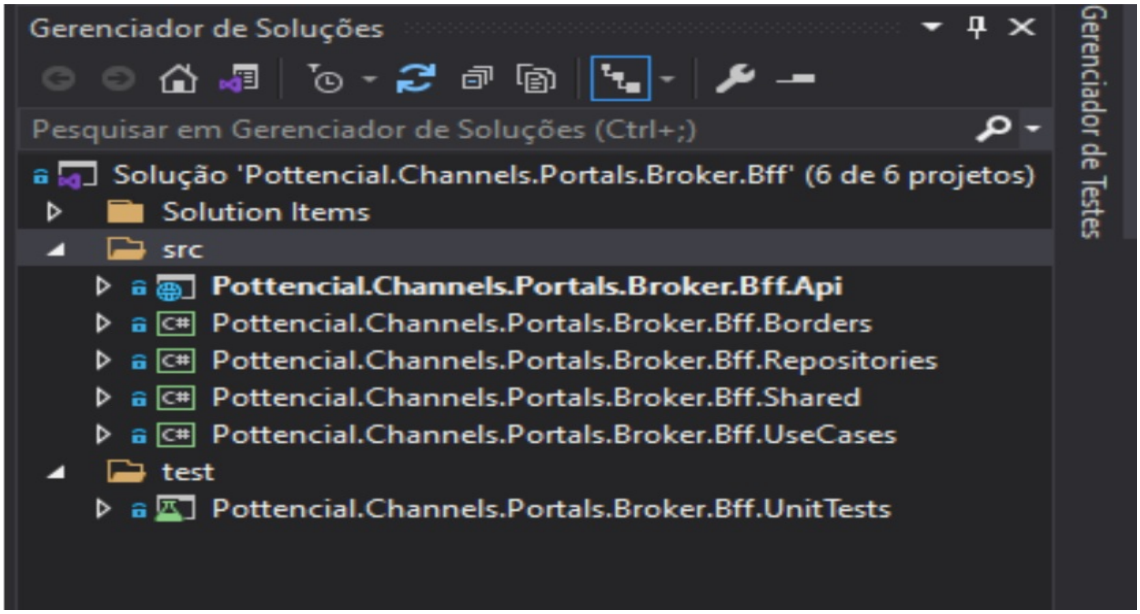
Fonte: Autoria Própria

Para a criação das features pessoais, era necessário adicionar uma chave referente a tarefa, esse número era gerado toda vez que uma tarefa era criada no Boards do Azure. Esse número mais o nome da tarefa normalmente era o padrão e tinha o objetivo de ajudar na identificação, onde a feature era facilmente encontrada caso houvesse necessidade de verificação de código.

4.6 Estrutura do Projeto

Utilizando princípios do S.O.L.I.D, arquitetura hexagonal e padrões DDD. O projeto tinha como objetivo o desempenho, por ser uma página web e conter muitas requisições, ao mesmo tempo visava ter uma fácil manutenção. O projeto contava com 5 camadas principais e 1 camada destinada aos testes unitários: Api, Borders, Repositories, Shared, UseCases e UnitTests. A figura 4.9 traz como exemplo as camadas padrão da API do projeto.

Figura 4.9 – Estrutura do Projeto



Fonte: Projeto Broker.BFF

Na camada de bff.API atuei na criação de controllers, que são como a porta de entrada da aplicação e é através dela que o *front-end* faz suas requisições, sempre mantendo o padrão de comunicação Json. A controller herdava da classe ApiController que fornece muitas propriedades e métodos úteis para lidar com solicitações HTTP. Já o namespace Microsoft.AspNetCore.Mvc era usado para fornecer atributos que podem ser usados para configurar o comportamento de controladores de API Web e dos métodos de ação. O exemplo a seguir, Figura 4.10 apresenta um método HTTP-Delete, que recebe um Guid como request e retorna um booleano como response , ambos em formato JSON. Também é usado atributos para especificar o verbo de ação HTTP com suporte e quaisquer códigos de status HTTP conhecidos que podem ser retornados.

O segundo nível bff.Bordes atuava como uma camada de domínio, responsável por armazenar as entidades, requisições, respostas, interfaces.UseCase, interfaces.Repository e interfaces-Adapters.

Já a camada bff.Repository era responsável pela integração com outras API's, além de também fazer consultas diretas ao banco de dados. Como o projeto consultava API's externas era necessário a integração para que o Broker.bff conseguisse receber e enviar informações externas. Um exemplo de API externa era a aplicação chamada Authorization.API, era através dela que a busca por dados sensíveis era feita. O Authorization.API era utilizado para obter id, guids, número de documentos pessoais, permissões, entre outros dados que requerem uma maior proteção. Já o banco de dados

Figura 4.10 – Chamada EndPoint HTTP - Delete

```
[Authorize]
[HttpDelete("email/{id}")]
[ProducesResponseType(StatusCodes.Status201Created, Type = typeof(bool))]
[ProducesResponseType(StatusCodes.Status404NotFound, Type = typeof(List<ErrorMessage>))]
[ProducesResponseType(StatusCodes.Status500InternalServerError, Type = typeof(List<ErrorMessage>))]
public async Task<IActionResult> DeleteEmailAsync(Guid id)
{
    return _actionResultConverter.Convert(await _deleteBrokerEmailUseCase.Execute(id));
}
```

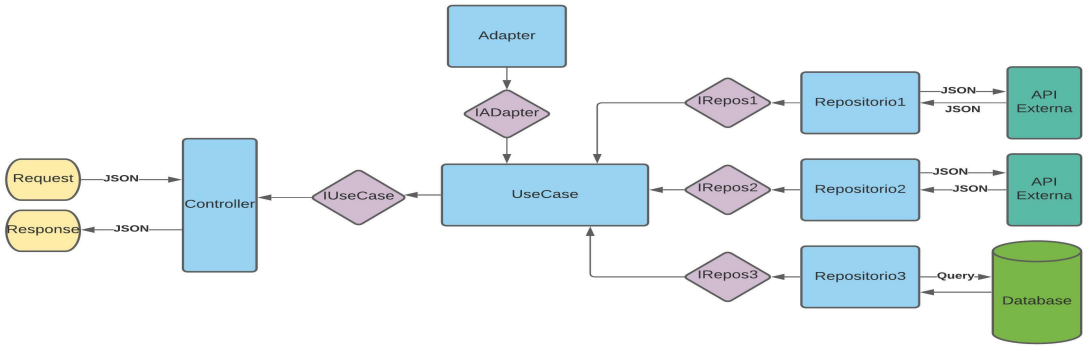
Fonte: Broker.BFF - Controller

utilizamos o SQL-server e o banco que o Broker.BFF acessava diretamente continha dados menos sensíveis e poderiam ser buscados com maior facilidade e rapidez.

O quarto nível ficava a camada bff.Share que cuidava das entidades e configurações necessárias no compartilhamento entre as API 's. Também cuidava dos dados recebidos do *front-end* no cabeçalho HTTP, os cabeçalhos permitem que o cliente e o servidor passem informações adicionais com a solicitação ou a resposta HTTP. Esses dados vindos do cabeçalho eram desserializados e reconstruídos em uma classe denominada UserInfo, para caso necessário o programador pudesse utilizá-los dentro da camada bff.UseCase onde é elaborada toda regras de negócio.

Por fim, no quinto nível temos a camada bff.UseCase, responsável por toda regra de negócio e manipulação de dados, sejam eles de entrada ou resposta. A camada bff.UseCase busca manter um baixo nível de acoplamento entre os diferentes módulos do sistema através da injeção de dependência, que é um design pattern que prega um controle externo visando remover dependências desnecessárias entre as classes e é através dessa classe que iremos fazer a ligação entre controller, repositório e adapter. Caso seja necessário o bff.UseCase também faz o uso de adapters, um adaptador é um intermediador que recebe solicitações do cliente e converte essas solicitações num formato que a requisição necessite. O Adaptador permite que classes com interfaces incompatíveis consigam trabalhar juntas. A Figura 4.11 busca busca trazer uma visão da estrutura do projeto back-end e como a API foi pensada e organizada para empregar conceitos S.O.L.I.D.

Figura 4.11 – Estrutura do Projeto *Branchs*



Fluxo demonstrando a estrutura back-end, com classes, interfaces, repositórios, entidades e adapters.

4.7 Testes de fluxo

Antes da criação dos testes unitários automatizados, primeiramente era necessário fazer os testes locais de fluxo, validando os dados, verificando a integração entre as aplicações e verificando se o endpoint chamado realmente estava executando seu propósito.

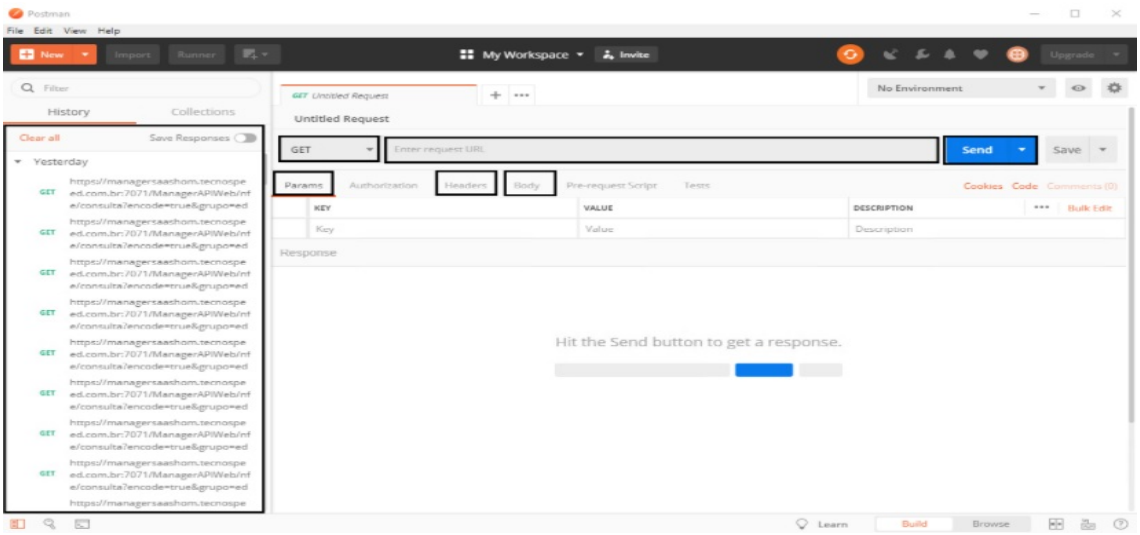
Para esses testes utilizamos o Postman, sendo usado como uma api cliente ele permite que as solicitações HTTP fossem testadas. Uma solicitação POST por exemplo, que fosse necessário passar dados como parâmetro, através dele é possível a criação de uma request no formato JSON e fazer a chamada. Além disso é possível adicionar tokens de autorização, ou qualquer outro parâmetro que seja necessário para que a requisição seja feita. Executando a API localmente e utilizando o Postman, passando a url local na chamada e a URI do controller, fica fácil identificar qualquer erro e verificar se está chegando uma resposta válida.

A figura 4.12 visa demonstrar as principais ferramentas da interface Postman. Na parte superior temos o cabeçalho HTTP (Get, Post, Put ...), onde podemos escolher qual o estilo da requisição, passando a URL e clicando no botão “Send” para fazer a consulta. Mais baixo temos campos onde podemos configurar os parâmetros necessários, e por fim no canto esquerdo temos um histórico demonstrando todas as últimas requisições feitas.

4.8 Testes Unitários

As características da qualidade formam o grupo de requisitos que avaliam a certificação das exigências dos clientes e descrevem suas funções e atribuições.

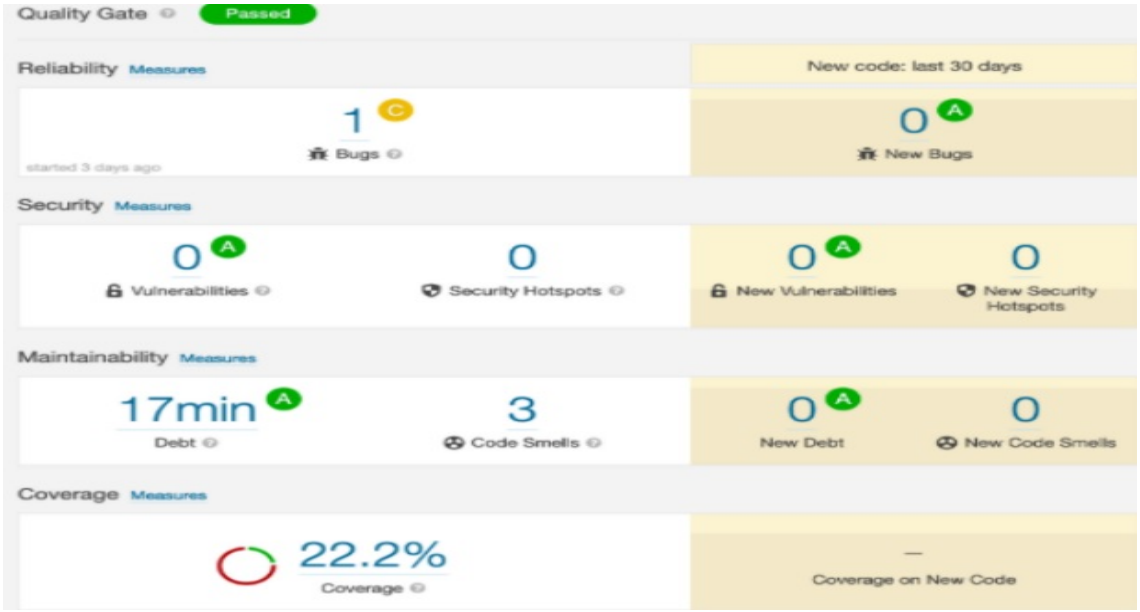
Figura 4.12 – Postman



Fonte: https://atendimento.tecnospeed.com.br/hc/pt-br/articles/360017143594-Como-instalar-e-utilizar-o-Postman-para-enviar-requisi%C3%A7%C3%B5es-HTTP

Era necessário que após a conclusão de cada tarefa fosse elaborado os testes unitários antes de conseguir aprovação para uma junção da branch atual com a branch em desenvolvimento. O SonarCloud era utilizado para o auxílio e detecção de problemas de qualidade, garantindo continuamente a manutenção, confiabilidade e segurança do código. A figura 4.13 traz um exemplo de um código submetido à análise pelo SonarCloud, o código em questão recebeu um alerta de bug, teve 3 code smells que são trechos de código que podem indicar um problema mais profundo. A figura 4.13 também apresenta uma indicação de apenas 22,2% de cobertura, essa porcentagem no exemplo é suficiente, mas no projeto desenvolvido durante o estágio essa cobertura não seria suficiente, pois o pipeline era configurado em 95%, o teste falharia e seria necessário a implementação de mais verificações.

Figura 4.13 – Exemplo: Análise SonarCloud



Fonte: https://medium.com/m4u-tech/qualidade-de-software-com-sonarcloud-bitbucket-5a5e05c2d2d7

5 CONCLUSÃO

O objetivo deste documento foi relatar as experiências obtidas durante o período de estágio supervisionado na empresa Squadra, uma empresa que atua no nicho de fabricação de softwares, nesse período atuei como desenvolvedor web backend na tribo Canais, que é responsável pelo portal do corretor em parceria com o cliente Pottencial Seguradora. Durante esses 12 meses de estágio, busquei ser capaz de aprimorar as competências de processos ágeis, desenvolvimento web e manipulação de banco de dados. Além de me aprofundar e colocar em prática tópicos vistos em sala de aula, também foi possível o contato com tecnologias totalmente novas como .netcore, azure, sonar cloud e swagger, assim como a estrutura de um projeto de grande escala e sua arquitetura base. Disciplinas como Estruturas de Dados, Sistemas Gerenciadores de Banco de Dados, Práticas de Programação Orientada a Objetos, Engenharia de Software e Testes de Software, foram importantes para uma melhor experiência durante o estágio, porque seus conceitos foram amplamente usados. Sendo um dos maiores projetos da empresa atualmente, tive a chance de trabalhar com uma equipe de profissionais excelentes que fizeram toda a diferença no meu aprendizado e crescimento, gerando grande valor profissional. Pode ser considerado um ponto negativo a falta de trabalho presencial, sendo todo o estágio de forma remota, gerando uma menor afinidade entre a equipe e uma maior dificuldade em assimilar alguns conteúdos no início do projeto. Depois dessa breve experiência no mercado de trabalho, fica evidente que apesar de todo o conhecimento teórico adquirido na faculdade, é fundamental que durante o curso o aluno tenha mais contato com tecnologias e metodologias que estão sendo usadas nas empresas, para que quando o aluno busque um estágio ou mesmo após formado, apresente uma melhor adaptação.

REFERÊNCIAS

DEURSEN, S. V. **Clean Code: A Handbook of Agile Software Craftsmanship**. 5. ed. [S.l.]: Pearson Education.Inc, 2019.

EDUARDO PIRES. **DDD, TDD BDD, Afinal, o que são essas siglas?** 2021. Disponível em: <http://eduardopires.net.br/2012/06/ddd-tdd-bdd/>. Acesso em: 07 out.

EIS, D. **Guia Front-End**. 1. ed. [S.l.]: Casa do Codigo, 2015.

ERIC EVANS. **Domain-Driven Design: Tackling Complexity in the Heart of Software**. 5. ed. [S.l.]: Pearson Education.Inc, 2003.

JOÃO ERIBERTO M.F. **Análise de Tráfego em Redes TCP/IP: Utilize Tcpdump na Análise de Tráfegos em Qualquer Sistema Operacional**. 1. ed. [S.l.]: NOVATEC, 2013.

MARK CRUTH. **ATLASSIAN Agile Coach - Discover the Spotify model**. 2019. Disponível em: <https://www.atlassian.com/agile/agile-at-scale/spotify>. Acesso em: 07 out. 2021.

MARK MASSE. **Rest API Design Rulebook: Designing Consistent Restful Web Service Interfaces**. 1. ed. [S.l.]: O'REILLY MEDIA, 2011.

Maurício S. S. **React - Aprenda Praticando: Desenvolva Aplicações web Reais com uso da Biblioteca React e de Seus Módulos Auxiliares**. [S.l.]: NOVATEC, 2021.

MICROSOFT. **Introdução ao .NET**. 2020. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/core/introduction>. Acesso em: 07 out. 2021.

MICROSOFT. **A TOUR OF THE C# LANGUAGE**. 2021. Disponível em: <https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>. Acesso em: 07 out. 2021.

POTTENCIAL SEGURADORA. **Sobre nós: crescemos acreditando em pessoas**. 2021. Disponível em: <https://pottencial.com.br/sobre-nos/>. Acesso em: 22 out. 2021.

POTTENCIAL SEGURADORA - GARANTIA. **Seguro Garantia**. 2021. Disponível em: <https://pottencial.com.br/seguro-garantia/>. Acesso em: 22 out. 2021.

POTTENCIAL SEGURADORA - PORTAL DO CORRETOR. **portal corretor**. 2021. Disponível em: <https://portalcorretor.pottencial.com.br/login/autenticacao>. Acesso em: 25 out. 2021.

QUEIRÓS, R. **Introdução ao Desenvolvimento Moderno Para a Web. Do Front-End ao Back-End. Uma Visão Global**. 1. ed. [S.l.]: FCA, 2018.

ROBBINS, J. N. **Learning web design: a beginners guide to HTML, CSS, Javascript, and webgraphics.** 5. ed. [S.l.]: Pearson Education.Inc, 2012.

ROBERT C. MARTIN. **Dependency Injection Principles, practices and patterns**. 2ª edição. ed. [S.l.: s.n.], 2019.

Squadra Digital. **Sobre nós: Squadra Digital**. 2021. Disponível em: <https://www.squadra.com.br/>. Acesso em: 07 out. 2021.

STOYAN STEFANOV. **Padrões JavaScript**. 2ª edição. ed. [S.l.]: O'REILLY'novatec, 2010.

UNIVERSIDADE FEDERAL DE LAVRAS. **O Curso de Ciência da Computação**. 2019. Disponível em: <http://www.bcc.ufla.br/>. Acesso em: 25 out. 2021.