METODOLOGIA SCRUM

DIAGRAMAÇÃO E PROTOTIPAÇÃO

TECNOLOGIAS UTILIZADAS NO DESENVOLVIMENTO DO SISTEMA MOBILE

TECNOLOGIAS E PADRÕES UTILIZADOS NO DESENVOLVIMENTO DA API

TESTES

Arquitetura cliente-servidor

BANCO DE DADOS

GIT

IMPLANTAÇÃO

2.2 BANCO DE DADOS...............................................................................................16

*2.2.1 Banco de dados relacional ........................................................................17*

SQL ....................................................................................................19

2.2.2 Postgresql.........................................................................................19

2.2.3 Aiven.........................................................................................19

2.3 TECNOLOGIAS E PADRÕES UTILIZADOS NO DESENVOLVIMENTO DA API....................22

intro

*2.3.2 Padrão REST ............................................................................................25*

Sem estado

*2.2.1 MVC e DAO........................................................................17*

2.3.1.1 Python.......................................................................................................24

2.3.1.2 Framework Flask................................................................................24

2.3.1.3 SQLAlchemy

2.3.2.2 JWT e Criptografia .................................................................................................27

*2.3.3 Docker.......................................................................................................33*

2.4 TECNOLOGIAS UTILIZADAS NO DESENVOLVIMENTO DO SISTEMA MOBILE.......................34

*2.4.1 Android ........................................................................................................35*

*2.4.3 JavaScript .................................................................................................35*

*2.4.1 React Native ........................................................................................................35*

2.5ARQUITETURA CLIENTE-SERVIDOR

2.5.1 NUVEM ..............................................................................................................40

*2.5.2 Azure..........................................................................................................41*

*2.5.1.1 Container Apps..........................................................................................41*

*2.5.1.2 Blob Storage.............................................................................................41*

2.6 METODOLOGIA SCRUM........................................................................................43

*2.6.1 Princípios e valores do Manifesto Ágil.......................................................43*

*2.6.2 Pilares do Scrum.......................................................................................44*

*2.6.3 Componentes do Time Scrum...................................................................45*

*2.6.4 Sprint.........................................................................................................45*

2.7 DIAGRAMAÇÃO E PROTOTIPAÇÃO.....................................................................................59

*2.7.1 Draw.io.........................................................................................................45*

*2.7.1 Figma.........................................................................................................45*

2.8 GIT....................................................................................................................59

2.9 INTEGRAÇÃO E IMPLANTAÇÃO ........................................................................................59

BREVE INTRODUÇÃO – CI/CD

*2.9.1 GitHub ..........................................................................................60*

*2.9.1.1 Github actions*

*2.9.2 Expo ........................................................................................................*

1. **TECNOLOGIAS, FERRAMENTAS E PADRÕES EMPREGADOS NO**

**DESENVOLVIMENTO DO SOFTWARE**

**2.2. Banco de Dados**

Em um projeto de software, a escolha do banco de dados é uma decisão crítica, pois afeta diretamente a estrutura e a eficiência do sistema. Segundo Date (2004, p. 3), o banco de dados é um repositório central onde uma coleção de dados computadorizados podem ser armazenados e organizados, esses dados podem ser acessados pelos usuários por meio de solicitações na aplicação.

No desenvolvimento deste projeto, foi escolhido o modelo de banco de dados relacional que pudesse garantir a integridade dos dados e a eficiência nas consultas.

* + 1. **Banco de Dados Relacional**

Existem diferentes tipos de bancos de dados, e um dos mais comuns, selecionado para este projeto, é o banco de dados relacional. Um banco de dados relacional funciona armazenando dados em tabelas, onde cada tabela representa uma entidade ou objeto específico. As colunas de uma tabela representam atributos ou características, como “Nome”, “Data”, “Quantidade”, etc. Cada coluna possui um tipo de dado específico, como “INTEGER” para valores numéricos inteiros ou “CHAR” para valores de texto. As linhas de uma tabela representam os registros únicos de uma entidade dentro desta tabela.

A principal qualidade que diferencia os bancos de dados relacionais dos demais são os “relacionamentos” entre tabelas. Esses relacionamentos permitem que diferentes tabelas “conversem” entre si. Por exemplo, em um banco de dados de uma loja virtual, existem as tabelas “Cliente”, que armazena os dados pessoais dos clientes, e “Pedido”, que armazena os dados dos pedidos específicos. Essas tabelas têm um relacionamento que permite identificar qual cliente realizou quais pedidos.

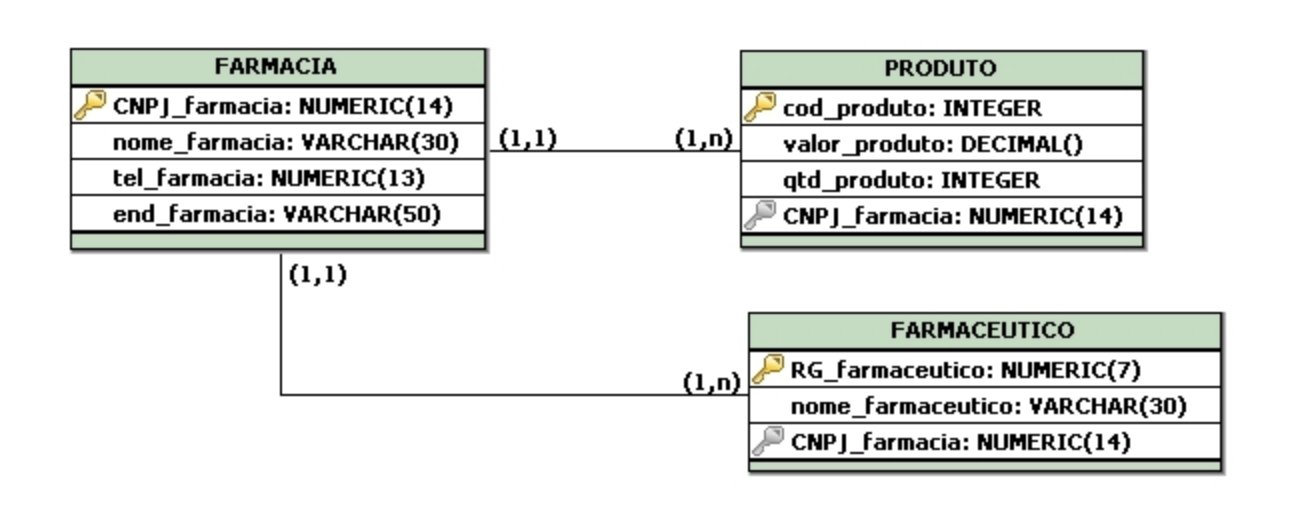
Para criar relacionamentos, toda tabela precisa de uma “chave primária”, que é uma coluna preenchida com um dado único, servindo como identidade para cada registro. Existem também as “chaves estrangeiras”, que são colunas que referenciam a chave primária de outra tabela, estabelecendo a conexão entre as duas tabelas.

Existem três tipos de relacionamentos:

* **Um-para-um (1:1):** Um registro de uma tabela está conectado a um registro de outra tabela.
* **Um-para-muitos (1:N):** Um registro de uma tabela está conectado a um ou mais registros de outra tabela.
* **Muitos-para-muitos (N:N):** Vários registros de uma tabela podem estar ligados a vários registros de outra tabela.

Esses conceitos podem ser exemplificados na seguinte imagem:

https://sae.unb.br/cae/conteudo/unbfga/sbd/new\_bancomodelagem.html



* + 1. **PostgreSQL**

PostgreSQL é um Sistema de Gerenciamento de Banco de Dados Objeto-Relacional (ORDBMS) de código aberto com mais de 35 anos de desenvolvimento ativo, o que lhe rendeu uma forte reputação de confiabilidade, robustez de recursos e desempenho (POSTGRESQL, 2024, Tradução Nossa).

Dentre suas diversas vantagens, o PostgreSQL conta com um sistema de Controle de Concorrência Multiversão (MVCC), em que cada requisição de leitura ou escrita acessa uma versão dos dados existente no momento em que foi iniciada. Isso isola as alterações feitas por outras requisições, permitindo que múltiplas operações de leitura e escrita ocorram simultaneamente, sem interferência mútua.

Essas características tornam o PostgreSQL altamente adequado para o desenvolvimento do projeto Nossa Via. No entanto, o fator determinante para sua escolha foi a extensão PostGIS, um módulo extremamente útil para a manipulação de dados geoespaciais. O PostGIS viabiliza operações avançadas em dados de localização, como cálculo de distâncias, áreas e interseções, facilitando processos fundamentais do projeto.

* + 1. **Aiven**

Aiven é uma plataforma que tem como objetivo facilitar a configuração, monitoramento e gerenciamento de infraestrutura de serviços de banco de dados em diversas nuvens, como a Microsoft Azure, AWS, Google Cloud e entre outras.

Como pode ser visto no próprio site oficial da plataforma, com Aiven é possível configurar e implantar um banco de dados do PostgreSQL em menos de 10 minutos (Aiven).

Considerando que os serviços da Aiven tornam o gerenciamento de infraestrutura mais rápido e simples e são gratuitos até um certo limite de capacidade, essa solução se mostrou ideal para ser aplicada durante o desenvolvimento do projeto Nossa Via.

2.3 TECNOLOGIAS E PADRÕES UTILIZADOS NO DESENVOLVIMENTO DA API...

Para o desenvolvimento do aplicativo, foram utilizadas APIs (*Application Programming Interface*), que funcionam como uma interface para a comunicação entre aplicações. Conforme definido pela Amazon (2023), "APIs são mecanismos que permitem que dois componentes de software se comuniquem usando um conjunto de definições e protocolos."

Essa tecnologia foi fundamental para o desenvolvimento da solução de software, atuando como/no o back-end da aplicação. O back-end é responsável pela interação com o banco de dados, servidores e pela execução de toda a lógica que está por trás da interface visível ao usuário final.

2.3.1 Padrão REST   
 O modelo de API adotado para a aplicação foi o padrão REST, amplamente utilizado devido à sua flexibilidade e popularidade. De acordo com a Amazon (2023), "Essas são as APIs mais populares e flexíveis encontradas na Web atualmente. O cliente envia solicitações ao servidor na forma de dados, o servidor processa essas solicitações e retorna os dados de saída ao cliente."

O termo REST vem do inglês *Representational State Transfer* e pode ser traduzido como "Transferência de Estado Representacional", e uma de suas principais características é a ausência de estado (*stateless*). Isso significa que o servidor não armazena dados de sessão do cliente entre requisições. A comunicação entre cliente e servidor segue o padrão HTTP (Hypertext Transfer Protocol), o que simplifica a troca de dados e permite uma integração ágil e eficiente com outras plataformas.

2.3.2 MVC e DAO   
 Para a arquitetura de desenvolvimento do sistema, optou-se pelo modelo MVC (Model View Controller) em conjunto com o padrão DAO (Data Access Object). Essa escolha visa garantir a segurança e a manutenibilidade do sistema. A utilização do padrão MVC proporciona um aumento significativo na reutilização e manutenção do projeto, uma vez que separa as camadas de apresentação, lógica de negócio e gerenciamento de fluxo, conforme discutido por Luciano e Alves (2011).

Existem diversas variações e maneiras de implementá-las, sendo a mais comum a estrutura de três camadas.

Nesse modelo, temos a camada de apresentação, a camada de negócios e a camada de dados. Cada uma dessas camadas possui suas responsabilidades específicas, o que ajuda a manter o sistema modular e facilita a manutenção e evolução da aplicação. (AWARI, 2023)

Segundo a IBM (2024), as três camadas da arquitetura cliente-servidor podem ser descritas da seguinte maneira:

A camada de apresentação é onde são exibidas as interfaces utilizadas pelos usuários, também é a partir dela que são feitas quaisquer solicitações que usuário possa precisar, podendo ser exibida em um navegador da Web, uma aplicação Desktop ou aplicativo mobile. Normalmente é desenvolvida utilizando linguagens como JavaScript, HTML e CSS, para permitir uma visualização mais amigável e de fácil uso para o usuário final.

A camada de negócios é responsável por receber e processar todas a solicitações feitas através da camada de apresentação, bem como ceder ou barrar acesso a determinados recursos, ela também utiliza as regras de negócio para validar as referidas requisições. Muitas vezes, ela acessa a camada de dados para inserir, atualizar, deletar ou consultar informações do sistema. Normalmente utiliza linguagens de programação como Java, C#, Python e outras.

A camada de dados, também conhecida como banco de dados, tem como função armazenar informações relevantes obtidas durante o processamento na camada de negócios, além disso, ela garante que esses dados estejam acessíveis as demais camadas da aplicação, enquanto os mantém protegidos de possíveis utilizações indevidas ou não autorizadas. Pode ser construída com os princípios de banco de dados relacionais ou não relacionais, utilizando linguagens como MySQL, PostgreSQL, Oracle, MongoDB, entre outros.

Após uma análise detalhada das características e benefícios de diversos produtos, optamos por utilizar o modelo de arquitetura cliente-servidor de três camadas no desenvolvimento do aplicativo Nossa Via. A camada de apresentação (front-end), foi desenvolvido e implantado através de um aplicativo móvel, baixado diretamente no dispositivo móvel do usuário final, isso se deve a familiaridade dos usuários com tecnologia móvel e a praticidade que ela oferece.

Já na camada de negócios (back-end), o da aplicação está sendo hospedados através do serviço de computação em nuvem da Microsoft Azure, especificamente, os Container Apps e Blob Storages, principalmente devido à flexibilidade, escalabilidade, baixo custo e eficiência oferecidas por essas tecnologias. Por último, mas não menos importante, na camada de dados, selecionamos a opção de infraestrutura/implantação do Aiven. Principalmente, escolhemos a Aiven porque oferece uma solução muito fácil de usar que pode ser dimensionada em qualquer direção e por um preço muito baixo. Esta é a camada sobre a operação principal do sistema que fica armazenado no banco de dados.

Complementarmente, o padrão DAO permite o isolamento dos dados, possibilitando alterações na estrutura do projeto de forma mais ágil e eficiente. Segundo Trindade e Fischer

Assim, o padrão de projeto DAO pode ser aplicado nas aplicações que façam acesso a qualquer tipo de banco de dados (relacional, orientado à objetos, arquivos de texto pleno, arquivos XML, servidores remotos, etc.), e que possa ter que acessar outros tipos quaisquer durante ou após o desenvolvimento da aplicação.(Trindade; Fischer)

Essa flexibilidade é essencial, pois permite a adaptação a diferentes tipos de dados, garantindo que o sistema possa evoluir conforme as necessidades.

A combinação dessas arquiteturas resulta em um software que apresenta flexibilidade tanto nas perspectivas de negócios quanto no sistema em si. Isso significa que alterações podem ser implementadas de maneira individualizada, sem impactar outras partes do sistema. Essa característica é especialmente importante quando se considera o ciclo de vida do software.

2.3.1.1 Python

2.3.1.2 Framework Flask

2.3.2.1 Cliente-servidor

2.3.2.2 Sem estado

2.3.2.2 JWT

2.3.3 Docker

**Tecnologias Utilizadas no Desenvolvimento do Sistema Mobile**

*2.3.1. React Native*

*2.3.2. JavaScript*

*2.3.3. Android*

* 1. **Arquitetura Cliente-Servidor**

A arquitetura cliente-servidor é uma modelo de organização de rede, onde os computadores de um sistema são subdivididos em dois grupos, o cliente, que realiza requisições em busca de recursos ou serviços, e os servidores, que são responsáveis por receber e responder a essas requisições (CONTROLE NET, 2024).

* + 1. **Nuvem**

No contexto de tecnologia, a “Nuvem” é o termo utilizado para descrever a ampla gama de servidores remotos conectados por todo o mundo, diversas empresas especializadas vendem o acesso desses servidores, assim, ao invés de armazenar dados ou softwares em suas próprias máquinas físicas, seus clientes podem acessá-los online.

É graças à computação em nuvem, que os componentes de um software podem funcionar sem a necessidade de possuir máquinas físicas. A arquitetura cliente-servidor em particular, permite que os usuários de um sistema possam realizar requisições a um servidor que está localizado até mesmo em outro país.

Essa tecnologia oferece diversos benefícios, como a terceirização de recursos, flexibilidade na implementação e facilidade de compartilhamento de dados. Nos serviços em nuvem, a responsabilidade pelo porte, manutenção e gerenciamento de hardware e software fica a cargo do provedor, dessa forma, o cliente é isento de preocupações relacionadas a esses aspectos. Na maioria dos casos, a implantação desses serviços é rápida e simples, permitindo ao usuário escolher a quantidade, o tipo e até a localização das máquinas necessárias.

Essa personalização é um dos principais motivos que tornam os serviços em nuvem mais econômicos. Além disso, devemos considerar o papel fundamental da nuvem no compartilhamento de dados. Em redes empresariais, os arquivos podem ser armazenados de forma segura e acessível, já em serviços de nuvem pública, é possível compartilhar arquivos globalmente, com algumas empresas oferecendo planos gratuitos até um certo limite de capacidade (Microsoft, 2024).

Segundo Bhardwaj et al. (2010), a Infraestrutura como Serviço (IaaS), é um serviço em nuvem que envolve a entrega de hardware e software como um serviço, permitindo que os usuários provisionem recursos sob demanda sem a necessidade de um compromisso a longo prazo. Diferente da Plataforma como Serviço (PaaS), o provedor de IaaS realiza pouca gestão além de manter o data center operacional, cabendo aos usuários a implantação e gestão dos serviços de software, similar ao que fariam em seus próprios data centers. Alguns exemplos de IaaS incluem Microsoft Azure, Amazon Web Services (AWS), Google Cloud etc. Entre esses, aquele escolhido para hospedar o back-end do aplicativo Nossa Via foi o Microsoft Azure.

* + 1. **Azure**

 O Azure é uma plataforma de computação em nuvem da Microsoft que disponibiliza uma ampla gama de produtos e serviços. O Azure destaca-se por sua flexibilidade, permitindo a utilização de diferentes ferramentas e frameworks, além de oferecer soluções escaláveis e seguras para setores variados, como saúde, finanças, governo e varejo (MICROSOFT, 2024).

Dentre seus serviços, dois se mostraram adequados para serem integrados no aplicativo Nossa Via, os Container Apps, que são responsáveis pela implantação e execução do back-end, e os Blob Storages que armazenam todas as imagens do aplicativo.

* + - 1. **Container Apps**

Segundo Susnjara e Smalley (2024), no desenvolvimento de software, containers são ambientes executáveis que incluem uma aplicação juntamente com todas as suas dependências, bibliotecas e arquivos de configuração essenciais para seu funcionamento.

Os Containers Apps da Azure são uma solução da Microsoft que permite a implantação de containers de maneira fácil e rápida, com isso, não há necessidade de configurar ou gerenciar servidores, além de ajudar a economizar custos durante a utilização dessa tecnologia (Microsoft, 2024).

* + - 1. **Blob Storage**

Binary Large Object, também conhecido como Blob, é uma grande massa de dados binários que pode ou não se enquadrar em um formato de arquivo.

O Blob Storage, é uma forma de armazenamento de objetos*,* onde os Blobs são guardados sem nenhuma hierarquia nos dados, dessa forma, podem ser escalados quase indefinidamente. Esse modelo é particularmente adequado para armazenar dados de mídias, como fotos, vídeos e outros (Could Flare, 2024).

1. **Framework Scrum**

A metodologia adotada para o desenvolvimento do aplicativo foi o framework Scrum, que é uma estrutura que segue os princípios do Manifesto Ágil e é amplamente utilizada para gerenciar projetos de maneira eficiente.

No Guia do Scrum, é definido que:

Scrum emprega uma abordagem iterativa e incremental para otimizar a previsibilidade e controlar o risco. Scrum envolve grupos de pessoas que, coletivamente, possuem todas as habilidades e conhecimentos necessários para fazer o trabalho e compartilhar ou adquirir essas habilidades conforme necessário (Schwaber; Sutherland, 2020, p. 4).

Em outras palavras, o Scrum facilita a entrega contínua e incremental de produtos, permitindo que as equipes se adaptem rapidamente às mudanças e aprimorem seus processos continuamente.

*2.4.1. Manifesto Ágil*

O Manifesto para Desenvolvimento Ágil de Software, foi criado em 2001 por um grupo de desenvolvedores de software e estabelece os valores e princípios fundamentais que norteiam as metodologias ágeis, incluindo o Scrum. O manifesto ágil valoriza:

Indivíduos e interações mais que processos e ferramentas. Software em funcionamento mais que documentação abrangente. Colaboração com o cliente mais que negociação de contratos. Responder a mudanças mais que seguir um plano (Manifesto Ágil, 2001).

Esses valores são complementados pelos seguintes princípios definidos pelo Manifesto Ágil (2001):

1. Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado.
2. Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.
3. Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.
4. Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.
5. Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.
6. O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face.
7. Software funcionando é a medida primária de progresso.
8. Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
9. Contínua atenção à excelência técnica e bom design aumenta a agilidade.
10. Simplicidade – a arte de maximizar a quantidade de trabalho não realizado – é essencial.
11. As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis.
12. Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

*2.4.2. Pilares do Scrum*

O Scrum é sustentado por três pilares, que são essenciais para o funcionamento eficaz do framework, esses pilares são citados por Schwaber e Sutherland (2020, p. 4), “os pilares empíricos do Scrum: transparência, inspeção e adaptação”.

* **Transparência:** Todos os aspectos do processo devem estar visíveis para os envolvidos, facilitando a compreensão e o acompanhamento dos progressos. Processos com baixa transparência podem levar a decisões que diminuem o valor e aumentam o risco. O uso de artefatos como o backlog e reuniões diárias garante a visibilidade do trabalho.
* **Inspeção:** O progresso em direção aos objetivos é inspecionado regularmente. Eventos como as revisões de sprint permitem que a equipe e o cliente avaliem os entregáveis e identifiquem possíveis problemas.
* **Adaptação:** Baseado nas inspeções, ajustes são feitos no processo ou no produto. Isso permite que o time se adapte rapidamente às mudanças nas necessidades do projeto.

Esses pilares trabalham juntos, a transparência permite a inspeção, a inspeção habilita a adaptação, e a adaptação garante que o projeto permaneça eficiente e flexível diante de mudanças.

*2.4.3. Componentes do Time Scrum*

O time Scrum é composto por três papéis principais:

* **Scrum Master:** Atua como facilitador, removendo obstáculos e garantindo que o time siga os princípios e as práticas do Scrum. Ele também assegura que todos entendam o processo e a sua importância para o sucesso do projeto.
* **Time de Desenvolvimento:** Consiste em profissionais que possuem as habilidades necessárias para entregar incrementos de produto em cada sprint. O time é autogerenciado e multidisciplinar, garantindo que todas as competências necessárias estejam presentes
* **Product Owner:** Representa os interesses do cliente e é responsável por maximizar o valor do produto desenvolvido. Ele também é responsável por gerenciar o Product Backlog, garantindo que o time tenha uma visão clara do que deve ser feito.

*2.4.4. Sprint*

O Sprint é o coração do Scrum. Trata-se de um ciclo de trabalho com duração fixa, segundo Pereira, Torreão e Marçal (2007, p. 4): “A recomendação é de que a duração da Sprint fique entre 2 a 4 semanas.”, nesse período, o time busca desenvolver incrementos de produto prontos para serem entregues. Cada Sprint começa com uma Sprint Planning, onde o time seleciona os itens do Product Backlog que serão trabalhados durante aquele ciclo.

No final da Sprint, uma revisão é realizada para inspecionar o trabalho concluído e ajustar as prioridades para as Sprints futuras, se necessário. As Sprints criam uma cadência previsível, oferecendo ao cliente e ao time a possibilidade de avaliar continuamente o progresso (Schwaber; Sutherland, 2020, p. 11).

Para o desenvolvimento deste trabalho, foi definido 5 Sprints com duração de 21 dias cada.

Tabela 1 – Cronograma Entrega Sprints

| Data de Entrega | Sprint |
| --- | --- |
| 28/08/2024 | Sprint 1 |
| 18/09/2024 | Sprint 2 |
| 09/10/2024 | Sprint 3 |
| 30/10/2024 | Sprint 4 |
| 20/11/2024 | Sprint 5 |

Fonte: Autores (2024).

*2.4.4.1 Product Backlog*

O Product Backlog é essencial para o Scrum, pois permite que a equipe de desenvolvimento saiba exatamente o que precisa ser feito e em ordem. De acordo com Sabbagh (2013, p. 111), “O Product Backlog é uma lista de tudo o que se acredita que será desenvolvido pelo Time de Desenvolvimento no decorrer do projeto”.

Segundo Sabbagh (2013, p. 112), “O Scrum não prescreve nenhum formato ou padrão para o Product Backlog. O importante é que ele tenha o formato de itens em uma sequência”. Cada item no Product Backlog deve estar claro e detalhado, de maneira que facilite o entendimento e a execução das tarefas, além disso o Product Backlog é projetado para ser dinâmico e flexível, ele pode e deve ser alterado conforme novas informações surgem e as prioridades mudam.

As figuras a seguir iram mostrar o Product Backlog do aplicativo.

Figura

* 1. **Diagramação e Prototipação**

Durante o processo de desenvolvimento de software, muitas vezes uma das maiores dificuldades dos desenvolvedores não é a programação ou implantação do projeto em si, mas sim compreender as características e escopo do projeto.

Segundo Sommerville (2012), projetos que não são bem documentados, ou que não têm seus requisitos adequadamente definidos, enfrentam grandes problemas com incertezas ou mudanças constantes.

Devido a esses problemas, muitas metodologias foram criadas a fim de ajudar a representar graficamente um software, tanto para os desenvolvedores, quanto para os clientes. Exemplos disso podem ser vistos nas técnicas de diagramação e prototipação.

A diagramação é uma forma de documentação que envolve especificar as características de um sistema de forma gráfica, ela ajuda a representar a arquitetura, componentes e relações do sistema, servindo como meio para comunicar ideias complexas para os membros da equipe e stakeholders.

Segundo Lima (2011, p. 27):

[...] na Tecnologia da Informação, a construção de modelos exige uma linguagem de modelagem que inclua elementos visuais para expressar conceitos e uma notação simples, mas poderosa para esses elementos.

Por isso, escolhemos utilizar a UML (Unified Modeling Language) para o desenvolvimento do projeto Nossa Via. Além de ter uma ótima clareza visual e versatilidade, seu padrão é reconhecido globalmente e tem suporte em diversas ferramentas. Utilizamos a ferramenta draw.io para a criação dos seguintes diagramas da UML: caso de uso, classe, sequência, atividade, implantação e estado.

Em relação a prototipação, Engholm (2010) destaca que, ao criar aplicações, a implementação de protótipos de interfaces é crucial para garantir bons resultados na usabilidade. Esses protótipos permitem uma visualização imediata do sistema pelos participantes do projeto, facilitando a validação e o refinamento do produto em desenvolvimento. Eles também garantem que tanto os desenvolvedores quanto os clientes possam visualizar como será o produto antes que esteja completamente desenvolvido, auxiliando na identificação de erros de usabilidade e design logo no início do desenvolvimento, assim, mudanças e correções podem ser feitas rapidamente sem um grande custo.

Após uma análise detalhada dos diferentes tipos de prototipos existentes, optamos por utilizar prototipos de alta fidelidade no desenvolvimento do projeto Nossa Via. A escolha se deve à necessidade de garantir uma representação precisa e detalhada da interface do usuário, proporcionando uma visualização clara do produto final. Para a criação desses protótipos, utilizaremos a ferramenta Figma, reconhecida por sua versatilidade e recursos colaborativos.

* + 1. **Draw.io**

Para a criação dos diagramas do projeto Nossa Via, utilizamos o Draw.io, uma ferramenta web de diagramação que permite criar uma grande variedade de diagramas, destacando-se tanto por sua facilidade de usabilidade quanto pelo grande detalhamento que oferece. A ferramenta também inclui opções de integração com diversas plataformas de armazenamento, uma função que utilizamos para guardar os diagramas no repositório do projeto no GitHub. (Draw.io).

* + 1. **Figma**

Figma é uma ferramenta web especializada na criação, compartilhamento e teste de designs de software, destacando-se pelo seu alto grau de detalhamento e personalização, isso fez com que se tornasse amplamente utilizado na criação de protótipos para telas dos mais variados softwares. Possui vantagens como visualização e criação com múltiplos usuários em tempo real, além da função de feedback integrado, que permite adicionar avaliações diretamente no design (Figma).

* 1. **Testes de Software**

No desenvolvimento de um software, é necessário garantir que o produto esteja atendendo todos os requisitos que foram especificados. Com a crescente complexidade dos sistemas, é inevitável que erros aconteçam, por isso é necessário validar todas as funcionalidades, infelizmente, nem sempre é fácil identificar o comportamento do sistema em todos os possíveis cenários, é comum que funções específicas sejam esquecidas, ou que o sistema reaja de maneira inesperada a diferentes situações. A fim de solucionar esse problema, foram criadas múltiplas metodologias para testar as funções e características de um software.

Testar um software traz muitos benefícios, desde melhorar a experiência do usuário até economizar tempo e dinheiro durante o processo de desenvolvimento. Porém, muitas vezes esses benefícios são intangíveis, sendo difícil prever e comparar como teria sido o desenvolvimento de um projeto com ou sem o uso dos testes. Isso é ainda mais agravado pelo fato de que realizar testes é um processo longo, complexo e caro, como pode ser confirmado pela fala de Sommerville (2011), “aproximadamente 60% dos custos de software são de desenvolvimento, 40% são custos de testes. Para o software customizado, os custos de evolução frequentemente superam os custos de desenvolvimento”.

Em muitos casos empresas possuem prazos e custos apertados, por isso escolhem não testar seus produtos para agilizar a entrega. Mas negligenciar os testes pode facilmente levar a perdas muito maiores, como falhas em ambiente de produção, experiência do usuário insatisfatória, custos de manutenção, danos à reputação e muitos outros.

Isso mostra como é fundamental utilizar estratégias de testes no desenvolvimento de um sistema, no projeto Nossa Via em particular, utilizamos dois métodos para testar o software, o teste de aceite no frontend, e testes unitários no backend.

Os testes de aceite são realizados simulando o comportamento do usuário final, nele verificamos se o software está executando corretamente as funções para quais foi construído. O roteiro com os testes realizados no projeto está localizado no apêndice X

Nos testes unitários, os componentes individuais do código, como funções, métodos, ou classes são avaliados individualmente, verificando se funcionam de forma isolada. No caso do projeto, são escritos pelos próprios desenvolvedores, e funcionam de maneira automatizada. O resultado dos testes está localizado no apêndice X

* 1. **Controle de versionamento**

Atualmente, os softwares têm se tornado cada vez mais complexos, Isso pode trazer certas dificuldades, como programar colaborativamente, perda de histórico do código, gerenciar versões de um sistema e complicações ao implementar ou revisar mudanças. Esses desafios geraram a inspiração para a criação dos sistemas de controle de versionamento.

Com esses sistemas, os desenvolvedores são capazes de manter um histórico de todas as mudanças feitas em um código. Após realizar um conjunto de alterações em um projeto, o programador pode aplicar essas alterações no código principal. Nesse momento, é criada uma nova “versão” do código, e as versões anteriores são armazenadas como backups que podem ser consultados ou até mesmo recuperados, assumindo o papel de versão principal.

Outra função que o controle de versionamento oferece são as chamadas “branches”, elas funcionam como cópia paralela do código principal, nela os desenvolvedores podem adicionar novas funcionalidades, corrigir bugs, ou fazer testes sem afetar o programa principal.

A utilização de branches também traz uma maneira eficaz de programadores trabalharem de maneira colaborativa, fazendo uso dessa funcionalidade, os desenvolvedores podem realizar mudanças no código paralelamente, cada um em suas próprias máquinas e branches, e em seguida, mesclar as alterações na versão principal. Tudo isso pode ser feito até mesmo offline, trazendo uma agilidade e estabilidade difícil de alcançar sem o uso dessas ferramentas.

Criada por Linus Torvalds em 2005, o Git é a ferramenta de controle de versionamento mais utilizada do mercado, seus sistemas de branches e mesclagem são mais rápidos, eficientes e menos propensos a erros. O Git é utilizado no projeto Nossa Via em conjunto da plataforma GitHub, a fim de garantir um armazenamento seguro, acessível e eficiente.

* 1. **Integração e implantação de Software**

A integração de software é o processo que envolve a conexão e comunicação entre componentes de um sistema ou entre diferentes sistemas, permitindo que as diversas partes compartilhem dados e funcionalidades, tornando o desenvolvimento mais dinâmico e modular. Esse procedimento é amplamente utilizado na maioria dos sistemas modernos, com poucas exceções, como sistemas autônomos ou de propósito único. O processo é ainda mais facilitado pelo CI (Continuous Integration), que automatiza a integração e os testes frequentes de código.

Essa técnica torna o desenvolvimento de softwares complexos mais eficiente, já que os desenvolvedores podem dividir o projeto em múltiplos "módulos" e integrá-los posteriormente, garantindo que dados de maneira automática e sincronizada. Além disso, a integração permite que um sistema utilize funcionalidades de outros softwares, como pode ser visto no aplicativo Nossa Via, que se integra ao Google Maps para navegação e ao Gmail para o envio de e-mails de recuperação de senha.

A implantação é uma etapa do ciclo de vida de um software onde o sistema é movido para o ambiente de produção, sendo disponibilizado para o usuário final. Esse processo envolve o desenvolvimento e teste do sistema, a configuração do ambiente, o deploy (implantação) e por fim, a manutenção e monitoramento. Pode-se realizar uma implantação de diversas maneiras, como manualmente, onde a equipe do projeto move o código para o ambiente de produção, ou no caso do Nossa Via, é realizada utilizando o CD (Continuous Delivery/Continuous Deployment), onde qualquer alteração no código é automaticamente após ser aprovada pelos testes automáticos.

Adotar um bom método de implantação pode trazer ótimos benefícios dependendo das necessidades do projeto, startups em particular tiram grande proveito de métodos automatizados de deploy, pois possuem a tendência de lançar atualizações mais frequentes, utilizando o CD a empresa pode se concentrar mais na criação de valor.

* 1. **GitHub**

O GitHub é uma plataforma baseada em nuvem que permite a hospedagem de código-fonte, oferecendo funcionalidades para armazenar, gerenciar e compartilhar sistemas de software utilizando o sistema de controle de versão Git. Além disso, integra-se a diversas ferramentas que auxiliam no desenvolvimento e na gestão de projetos, como Jira, Microsoft Teams e Microsoft Azure. A plataforma também inclui uma ferramenta nativa chamada GitHub Actions, que permite automatizar fluxos de trabalho, especialmente em processos de CI e CD, tornando o ciclo de desenvolvimento mais ágil e eficiente.

Sua popularidade tem crescido nos últimos anos, graças a vantagens como uma estrutura de repositórios que facilita a colaboração e o monitoramento de projetos. A interface amigável e intuitiva também simplifica o uso do Git, tornando-o mais acessível para desenvolvedores de todos os níveis.

No caso do projeto Nossa Via, o GitHub é utilizado para o armazenamento e desenvolvimento, sendo dividido em três repositórios: o Documentacao-TCC-NossaVia, que contém a monografia, diagramas, planos de testes e a documentação completa; o Back-end-TCC-NossaVia, que armazena o código do back-end, desenvolvido em Python com o framework Flask; e o Front-end-TCC-NossaVia, com o código do front-end, escrito em JavaScript utilizando a biblioteca React Native.

* 1. **GitHub Actions**

O GitHub Actions é uma ferramenta integrada ao GitHub, cujo propósito é automatizar e otimizar fluxos de trabalho. Ela permite que os desenvolvedores configurem pipelines para tarefas repetitivas, como testes, builds e deploys, simplificando os processos de CI e CD.

Graças à integração perfeita com os repositórios do GitHub, a ferramenta pode, por exemplo, ser configurada para rodar testes automaticamente sempre que um commit é realizado e, em seguida, implantar a nova versão do software após a aprovação dos testes. Isso oferece vantagens como a redução de erros manuais e maior agilidade no desenvolvimento.

No desenvolvimento do aplicativo Nossa Via, utilizamos o GitHub Actions para automatizar os processos de CI e CD. Após efetuar um commit em um dos repositórios do projeto no GitHub, o código passa automaticamente pelos testes unitários, e em seguida, é automaticamente implantado.

* 1. **Expo**

Expo é uma plataforma e conjunto de ferramentas que facilitam o desenvolvimento, teste e implantação de aplicativos móveis utilizando React Native. Entre suas funcionalidades, estão bibliotecas de APIs prontas para uso, permitindo que os desenvolvedores acessem recursos como câmera, GPS e armazenamento local. Outra utilidade é o aplicativo Expo Go, que, após instalado, permite que os desenvolvedores visualizem o projeto em dispositivos físicos em tempo real, analisando um QR code. A plataforma também oferece funções que facilitam a publicação na App Store e Google Play, além do "Expo Application Service" (EAS), que simplifica a geração de aplicativos binários (APK e IPA).

A utilização do Expo no aplicativo Nossa Via permitiu um desenvolvimento muito mais rápido e eficiente. O processo de configurar, implementar e testar o aplicativo foi extremamente simplificado. Também utilizamos suas APIs para obter imagens do armazenamento do dispositivo móvel e a localização do dispositivo.

FIGMA. O que é o Figma. Disponível em: <https://help.figma.com/hc/pt-br/articles/14563969806359-O-que-%C3%A9-o-Figma>. Acesso em: 06 out. 2024.

[DRAW.IO.](https://draw.io/?formCode=MG0AV3)Draw.io Documentation. Disponível em: <https://www.drawio.com/doc/?formCode=MG0AV3>. Acesso em: 06 out. 2024.

SOMMERVILLE, Ian. Engenharia de software. 9. ed. 2012.

ENGHOLM Júnior, Hélio. Engenharia de software na prática. São Paulo. 2010.

Couldflare. O que é armazenamento de blobs? **Couldflare**, 6 set. 2022. Disponível em: <https://www.cloudflare.com/pt-br/learning/cloud/what-is-blob-storage/>. Acesso em: 29 set. 2024.

Lima, Adilson da Silva. UML 2.3 do requisito à solução. 1° edição. São Paulo: Érica, 2011.

AZURE. O que é computação em nuvem? **Microsoft**, 2024. Disponível em: <https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-cloud-computing>. Acesso em: 29 set. 2024.

AZURE. Visão geral sobre os Aplicativos de Contêiner do Azure**. Microsoft**, 2024. Disponível em: <https://learn.microsoft.com/pt-br/azure/container-apps/overview>. Acesso em: 29 set. 2024.

SUSNJARA, Stephanie; SMALLEY, Ian. O que são contêineres? **IBM**, 9 maio 2024. Disponível em: <https://www.ibm.com/br-pt/topics/containers>. Acesso em: 2 out. 2024.

AWARI. Arquitetura de Software Cliente-Servidor: Estrutura e Interação entre os Componentes.  **Awari**. São Paulo, 29 jul. 2023. Disponível em: <https://awari.com.br/arquitetura-de-software-cliente-servidor-estrutura-e-interacao-entre-os-componentes/>. Acesso em: 29 set. 2024.

ANDRADE, Adrianne Paula Vieira *et al*. ADOÇÃO DE SISTEMAS DE ARMAZENAMENTO DE DADOS NA NUVEM: UM ESTUDO COM USUÁRIOS FINAIS. **RAI Revista de Administração e Inovação**, São Paulo: FEA USP, ano 2015, p. 4-25, 1 out. 2015. Bimestral. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1809203916301115>. Acesso em: 25 set. 2024**.**

BHARDWAJ, Sushil; JAIN, Leena; JAIN, Sandeep. CLOUD COMPUTING: A STUDY OF INFRASTRUCTURE AS A SERVICE (IAAS). **International Journal of Engineering and Information Technology**, Ambala, India: waves, ano 2010, p. 60-63.

AZURE. O que é computação em nuvem?**Microsoft**, 2024. Disponível em: <https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-cloud-computing#:~:text=Resumindo%2C%20a%20computa%C3%A7%C3%A3o%20em%20nuvem,flex%C3%ADveis%20e%20economias%20de%20escala>. Acesso em: 25 set. 2024.

Controle Net. Cliente-Servidor, uma estrutura lógica para a computação centralizada. **Controle Net**, 9 maio 2024. Disponível em: <https://www.controle.net/faq/cliente-servidor-uma-estrutura-para-a-computacao-centralizada#:~:text=Uma%20estrutura%20cliente%2Dservidor%20%C3%A9,rede%20em%20infraestruturas%20de%20TI>. Acesso em: 29 set. 2024.

PostgreSQL. New to PostgreSQL?. **PostgreSQL**, 26 set. 2024. Disponível em: <https://www.postgresql.org/>. Acesso em: 03 nov. 2024.

PostgreSQL is a powerful, open source object-relational database system with over 35 years of active development that has earned it a strong reputation for reliability, feature robustness, and performance.

Aiven. Aiven para PostgreSQL. **Aiven**. Disponível em: <https://aiven.io/postgresql>. Acesso em: 03 nov. 2024.

IBM. O que é arquitetura de três camadas? **IBM**, 9 maio 2024. Disponível em: <https://www.ibm.com/br-pt/topics/three-tier-architecture>. Acesso em: 2 out. 2024.

DATE, Christopher J. **Introdução a Sistemas de Bancos de Dados.**8. ed. Rio de Janeiro: Elsevier, 2004. 896 p. Disponível em: https://books.google.com.br/books?hl=pt-BR&lr=&id=xBeO9LSlK7UC&oi=fnd&pg=PP23&dq=banco+de+dados&ots=xdNEl4Bh7N&sig=OinvR--HlR1IXquZoSbke6SbKDM#v=onepage&q=banco%20de%20dados&f=false. Acesso em: 1 out. 2024.

SCHWABER, Ken; SUTHERLAND, Jeff. **O Guia do Scrum.**O Guia Definitivo para o Scrum: As Regras do Jogo. 2020. 16 p. Disponível em: <https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-PortugueseBR-3.0.pdf#page=4&zoom=100>. Acesso em: 25 set. 2024.

MANIFESTO ÁGIL. **Manifesto para Desenvolvimento Ágil de Software**. 2001. Disponível em: https://agilemanifesto.org/iso/ptbr/manifesto.html. Acesso em: 25 set. 2024.

PEREIRA, Paulo; TORREÃO, Paula; MARÇAL, Ana Sofia. **Entendendo Scrum para Gerenciar Projetos de Forma Ágil**. MundoPM - Project Management, Recife – PE. Editora Mundo, ano 2007, p. 1-11, Disponível em: https://faculdadeprojecao.nucleoead.net/pos/pluginfile.php/2123/mod\_resource/content/36/Entendendo\_Scrum\_para\_Gerenciar\_Projetos.pdf. Acesso em: 26 set. 2024.

SABBAGH, Rafael. **Scrum:**Gestão Ágil para Projetos de Sucesso. Casa do Código, 2013. 270 p. Disponível em: https://www.rafasabbagh.com/scrumbook. Acesso em: 26 set. 2024.

AWS**.**O que é uma API (interface de programação de aplicações)?. *[S.l.].***Amazon**, 2023. Disponível em: https://aws.amazon.com/pt/what-is/api/ . Acesso em: 24 set. 2024.

LUCIANO, JOSUÉ; ALVES, WALLISON JOEL BARBERÁ. Padrão de arquitetura MVC: Model-view-controller. **Revista EPeQ Fafibe**, São Paulo: UNIFAFIBE, ed. 3°, ano 2011, p. 102-107. Disponível em: http://unifafibe.com.br/revistasonline/arquivos/revistaepeqfafibe/sumario/20/16112011142249.pdf. Acesso em: 30 set. 2024.

TRINDADE, Joana M. F.; FISCHER, Leonardo G. Estudo e Aplicação de Padrões. Artigo - Curso de Informática, Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre – RS – Brasil. Disponível em: https://www.researchgate.net/publication/268434484\_Estudo\_e\_Aplicacao\_de\_Padroes. Acesso em: 1 out. 2024.

SGBDR

AWS

IaaS

CI

CD