CENTRO UNIVERSITÁRIO PARA O DESENVOLVIMENTO DO ALTO VALE DO ITAJAÍ - UNIDAVI

PIERRE CAPISTRANO LOPES

PROTÓTIPO DE SISTEMA WEB PARA GERENCIAMENTO

DE PROPOSTAS EM ESCRITÓRIOS DE SOLUÇÕES AGRÍCOLAS

RIO DO SUL

2023

CENTRO UNIVERSITÁRIO PARA O DESENVOLVIMENTO DO ALTO VALE DO ITAJAÍ - UNIDAVI

PIERRE CAPISTRANO LOPES

PROTÓTIPO DE SISTEMA PARA GERENCIAMENTO EFICIENTE

DE ESCRITÓRIOS DE PROJETOS AGRÍCOLAS

Trabalho de Conclusão de Curso a ser apresentado ao curso de Sistemas da Informação, da Área das Ciências Naturais, da Computação e das Engenharias, do Centro Universitário para o Desenvolvimento do Alto Vale do Itajaí, como condição parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Prof. Orientador: Jullian Hermann Creutzberg

RIO DO SUL

2023

CENTRO UNIVERSITÁRIO PARA O DESENVOLVIMENTO DO ALTO VALE DO ITAJAÍ - UNIDAVI

PIERRE CAPISTRANO LOPES

PROTÓTIPO DE SISTEMA PARA GERENCIAMENTO EFICIENTE

DE ESCRITÓRIOS DE PROJETOS AGRÍCOLAS

Trabalho de Conclusão de Curso a ser apresentado ao curso de Sistemas da Informação, da Área das Ciências Naturais, da Computação e das Engenharias, do Centro Universitário para o Desenvolvimento do Alto Vale do Itajaí- UNIDAVI, a ser apreciado pela Banca Examinadora, formada por:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Professor Orientador: Jullian Hermann Creutzberg

Banca Examinadora:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof.

Rio do Sul, xx de mês de 2023.

Epígrafe, Epígrafe, Epígrafe, Epígrafe, Epígrafe, Epígrafe, Epígrafe, Epígrafe, Epígrafe, Epígrafe, Epígrafe, Epígrafe (Autor).

Dedicatória Dedicatória Dedicatória Dedicatória Dedicatória Dedicatória Dedicatória Dedicatória Dedicatória Dedicatória Dedicatória Dedicatória Dedicatória Dedicatória Dedicatória Dedicatória Dedicatória Dedicatória Dedicatória Dedicatória.

AGRADECIMENTOS

Agradecimentos agradecimentos agradecimentos agradecimentos agradecimentos agradecimentos agradecimentos agradecimentos agradecimentos agradecimentos agradecimentos agradecimentos.

RESUMO

Elemento obrigatório, que consiste na apresentação concisa dos pontos relevantes de um texto. O resumo deve dar uma visão rápida e clara do conteúdo e das conclusões do trabalho. Deve conter entre 150 a 500 palavras.

**Palavras-Chave:** palavra 1, palavra 2, palavra 3.

ABSTRACT

Versão do resumo em inglês (também pode ser em espanhol).

**Keywords:** keyword 1, keyword 2, keyword 3.

LISTA DE FIGURAS

Figura 1- Título Figura 21

Figura 2- Título Figura 21

Figura 3- Título Figura 21

Figura 4- Título Figura 21

Figura 5- Título Figura 21

LISTA DE QUADROS

Quadro 1 - Título Quadro 21

Quadro 2 - Título Quadro 21

Quadro 3 - Título Quadro 21

Quadro 4 - Título Quadro 21

Quadro 5 - Título Quadro 21

LISTA DE ABREVIATURAS E SIGLAS

MA Modelo de Abreviatura

**SUMÁRIO**

(a contagem inicia na folha de rosto, mas a paginação inicia apenas na Introdução)

1. INTRODUÇÃO 11

1.1 OBJETIVOS 12

1.2.1 Geral 12

1.2.2 Específicos 12

1.3 JUSTIFICATIVA 12

1. introdução

Nos últimos anos, o Alto Vale do Itajaí tem sido uma região de destaque no setor agrícola, com a agricultura familiar desempenhando um papel fundamental em seu desenvolvimento econômico. Consequentemente, os escritórios de projetos agrícolas têm ganhado cada vez mais importância nesse contexto, à medida que oferecem suporte e serviços essenciais aos agricultores locais. No entanto, o crescimento desses escritórios demanda um gerenciamento eficiente, assim como ocorre em qualquer empresa em expansão. É neste ponto que a tecnologia pode desempenhar um papel crucial, oferecendo soluções para aprimorar e otimizar esse processo de gerenciamento.

Os escritórios de projetos agrícolas desempenham um papel vital ao oferecer serviços diversos, como financiamento agrícola, investimentos, análises de terras, seguro de safra, entre outros, que são essenciais para os agricultores da região. No entanto, a prestação desses serviços requer o gerenciamento eficiente de uma ampla gama de informações. Caso essas informações não sejam devidamente administradas, podem ocorrer perdas de dados, ineficiência nos processos e, em casos extremos, problemas judiciais.

Diante dessa necessidade de gerenciamento mais eficiente, surge um problema evidente: a ausência de sistemas específicos no mercado para o gerenciamento de escritórios de projetos agrícolas. Atualmente, existem apenas sistemas ERP (*Enterprise Resource Planning*) genéricos, que atendem a diferentes tipos de empresas, além de sistemas voltados para o gerenciamento de projetos de forma mais geral. Contudo, um escritório de projetos agrícolas possui requisitos específicos que vão além do escopo dessas soluções existentes.

Logo, há uma demanda latente por um sistema de gerenciamento de escritórios de projetos agrícolas que possua as ferramentas adequadas para gerenciar projetos, suas documentações, atividades e progresso. Além disso, é essencial contar com funcionalidades para controle de clientes, gestão financeira, bem como a geração de gráficos e relatórios contendo indicadores relevantes para o acompanhamento do desempenho desses escritórios.

Diante desse contexto, este trabalho de conclusão de curso tem como objetivo principal o desenvolvimento de um protótipo de sistema para o gerenciamento de escritórios de projetos agrícolas. O sistema proposto visa suprir a lacuna existente no mercado, fornecendo uma solução personalizada e eficiente para atender às necessidades específicas desses escritórios. Por meio deste estudo, pretende-se explorar as funcionalidades essenciais e os benefícios potenciais que um sistema desse tipo pode oferecer, contribuindo assim para o aprimoramento e profissionalização dos escritórios de projetos agrícolas na região do Alto Vale do Itajaí.

**Tema: Protótipo de sistema para gerenciamento de escritórios de projetos agrícolas.**

1.1 PROBLEMA DE PESQUISA

Qual a melhor forma de gerenciar escritórios de projetos agrícolas?

1.2 Objetivos

1.2.1 Geral

* Desenvolver um protótipo de sistema web para gerenciamento de propostas em escritórios de soluções agrícolas.
  + 1. Específicos
* Descrever as tecnologias que serão utilizadas no desenvolvimento do protótipo;
* Detalhar a análise do protótipo por meio do levantamento de requisitos, elaboração de diagramas, e da modelagem das classes;
* Exemplificar o processo atual realizado por um escritório de soluções agrícolas;
* Construir o protótipo com utilização das tecnologias selecionadas para o projeto;

1.3 Justificativa

O Alto vale do Itajaí é uma região onde a agricultura representa grande parte do PIB, principalmente a agricultura familiar, e dessa forma os escritórios de projetos agrícolas crescem cada vez mais, necessitando de um gerenciamento maior como qualquer empresa em crescimento, assim a tecnologia poderá auxiliar esse processo.

Um escritório de projetos agrícolas presta serviços para agricultores que necessitam de financiamento agrícola, investimento, análises de terras, seguro de safra, etc. E todos esses serviços que são fornecidos necessitam de uma gama de informações que, caso não gerenciadas corretamente, podem ocasionar perda de dados, ineficiência dos processos e no pior dos casos problemas judiciais.

A partir dessa necessidade surge um problema, não existe no mercado sistemas específicos para o gerenciamento de escritórios desse tipo, existem somente ERPs (sistemas para gerenciar todos os tipos de empresas) e sistemas para gerenciamento de projetos.

Um sistema de para escritórios de projetos agrícolas necessita de ferramentas para gerenciar os projetos, bem como suas documentações, atividades, andamento, etc. Necessita de um controle de clientes, financeiro, além de gráficos e relatórios com os indicadores dos escritórios.

1.4 CONTEXTUALIZAÇÃO DA EMPRESA

Texto texto texto texto texto texto texto texto texto texto texto texto texto texto texto. texto texto texto texto texto texto. texto texto texto texto texto texto. texto texto texto texto texto texto. texto texto texto texto texto texto. texto texto texto texto texto texto. texto texto texto texto texto texto. texto texto texto texto texto texto. texto texto texto texto texto texto. texto texto texto texto texto texto. texto texto texto texto texto texto. texto texto texto texto texto texto. texto texto texto texto texto texto. texto texto texto texto texto texto.

Tamano da empresa (quantidade de filiais, funcionários)

Entrevista em profundidade não estruturada. Quando foi feita, data. Com quem foi feita;

Processo atual, o que é feito, quais as dificuldades

2. REFERENCIAL TEÓRICO

2.1 banco de dados

De acordo com Alves (2014), um banco de dados é um conjunto de valores e arquivos que se relacionam entre si e demonstram um cadastro de pessoas, vendas, produtos, agendas, etc. Por exemplo, uma planilha ou uma ficha cadastral podem ser exemplos de cadastros. O banco de dados guarda todos esses cadastros em formato virtual e os disponibiliza para as aplicações consultarem e emitirem relatórios, realizarem vendas, etc.

2.1.1 Banco de dados relacional

De acordo com Alves (2014, p. 20), Banco de dados relacional “se caracteriza pelo fato de organizar os dados em tabelas (ou relações), formadas por linhas e colunas. Assim, essas tabelas são similares a conjuntos de elementos ou objetos, uma vez que relacionam as informações referentes a um mesmo assunto de modo organizado.”

Conforme Silva (2021), bancos de dados relacionais são os mais utilizados no mercado atual, mesmo que tenha surgido a partir de modelos de 1970. Eles são conhecidos por serem sistemas de propósito geral, utilizarem a linguagem SQL (Structured Query Language; ou Linguagem de Consulta Estruturada, em português) para consultar e manipular os dados, além das propriedades ACID para transações, conforme apresentado no Quadro 1.

**Quadro 1 - Propriedades ACID**

|  |  |
| --- | --- |
| **Atomicidade** | As transações são atômicas, executam completamente ou não executam. |
| **Consistência** | As transações criam novos estados válidos, ou, em caso de falha, o banco de dados volta para o último estado válido. |
| **Isolamento** | Garante que uma transação em andamento não será interferida por outras transações. |
| **Durabilidade** | Garante que uma transação em andamento não será interferida por outras transações. |

Fonte: Elaborado a partir de Pritchett (2008)

Ainda de acordo com Silva (2021, p. 14):

Outra característica marcante dos bancos de dados relacionais é a integridade referencial, que garante a acurácia e a consistência dos dados dentro de um relacionamento entre tabelas. Isso é feito por meio de uma chave estrangeira, que faz referência a um valor de uma chave primária em outra tabela, e a integridade referencial garantirá que esse relacionamento é íntegro (o registro referenciado existe)

O Dr. Edgard F. Codd formulou os princípios básicos do sistema de banco de dados relacional em 1968, baseando-se na teoria dos conjuntos e da álgebra relacional. Segundo ele, certos conceitos da matemática podiam ser aplicados ao gerenciamento de bancos de dados, provavelmente por ter sido um brilhante matemático. Segundo CODD (1985), existem um conjunto de doze regras para que um banco de dados relacional fosse admitido como tal:

* Regra de informações;
* Regra de acesso garantido;
* Tratamento de valores nulos;
* Catálogo relacional ativo;
* Inserção, exclusão e alteração em bloco;
* Linguagem de manipulação de dados abrangente;
* Independência física dos dados;
* Independência logica dos dados;
* Regra de atualização de visões;
* Independência de integridade;
* Independência de distribuição;
* Regra não subversiva.

2.1.2 Banco de dados não relacional

Conforme Sadamoto et al. (2012), Bancos de dados não relacionais, também conhecidos como NoSQL, são sistemas de gerenciamento de bancos de dados que não seguem o modelo relacional tradicional. Ao contrário dos bancos de dados relacionais, que armazenam informações em tabelas com linhas e colunas, os bancos de dados não relacionais utilizam uma variedade de modelos de dados, como documentos, grafos, chave-valor e colunas amplas, para armazenar informações de maneira mais flexível e escalável. As principais características dos bancos de dados não relacionais incluem alta escalabilidade horizontal, flexibilidade no esquema de dados, suporte a dados semiestruturados e não estruturados, e alta disponibilidade. Além disso, muitos bancos de dados não relacionais são projetados para funcionar em ambientes distribuídos e podem fornecer recursos de replicação e particionamento para melhorar a capacidade de lidar com grandes volumes de dados.

Bancos de dados não relacionais são adequados para cenários em que a escalabilidade, a flexibilidade e a agilidade são mais importantes do que a consistência transacional ou a conformidade com um modelo de dados estrito, conforme afirmado por Sadamoto et al. (2012). Esses autores ainda destacam que os bancos de dados NoSQL são particularmente adequados para o armazenamento de grandes volumes de dados não estruturados ou semi-estruturados, como documentos, redes sociais, registros de log, entre outros. Além disso, eles podem fornecer recursos de replicação e particionamento para melhorar a capacidade de lidar com grandes volumes de dados e suportar ambientes distribuídos. Por fim, os bancos de dados não relacionais também são úteis em aplicações de análise de dados, processamento de big data, Internet das Coisas (IoT) e aplicações de busca, onde a flexibilidade do esquema de dados é mais importante do que a garantia de integridade referencial.

2.2 programação orientada a objetos

Programação Orientada a Objetos (POO) é um paradigma de programação que se concentra em representar entidades do mundo real como objetos, que possuem atributos e métodos que definem seu comportamento e funcionalidade. O objetivo da POO é facilitar a modelagem e a resolução de problemas complexos, dividindo-os em partes menores e mais gerenciáveis. Nesse paradigma, as entidades são representadas por classes, que são usadas para criar objetos. A POO é baseada em quatro conceitos principais: encapsulamento, herança, polimorfismo e abstração. O encapsulamento é a capacidade de esconder os detalhes internos de um objeto e protegê-lo contra acesso indevido. A herança permite que as classes herdem atributos e métodos de outras classes e estendam seu comportamento. O polimorfismo permite que os objetos sejam tratados de maneiras diferentes, dependendo do contexto em que são usados. A abstração é a capacidade de extrair as características essenciais de um objeto e criar uma classe abstrata que define a interface para essas características. Juntos, esses conceitos permitem que os desenvolvedores criem programas orientados a objetos que são modulares, flexíveis e fáceis de manter (BARNES; KÖLLING, 2012).

Conforme Farinelli (2007), A Programação Orientada a Objetos “consiste em conceber um sistema computacional como um todo orgânico formado por objetos que se relacionam entre si. Esse enfoque pode ser aplicado tanto à análise de sistemas quanto à programação, e essa é uma das principais vantagens da orientação a objetos”, portanto a mesma metodologia serve tanto para a definição lógica do sistema quanto para a sua implementação.

Farinelli (2007, p. 31) diz que:

dados e processos são apenas componentes, e o enfoque está em identificar quais os objetos que interagem entre si no sistema. Os dados são identificados procurando os atributos que definem os objetos, e os procedimentos pelas operações que estes objetos realizam. A interação entre os objetos é definida pelas estruturas e relacionamentos que são identificados. O resultado é que em um modelo orientado a objetos, existe total coerência entre os dados e os processos, mesmo quando há muitas pessoas trabalhando no mesmo sistema.

Portanto, a definição de dados e processos em análise de sistemas, é uma questão antiga, pois existem metodologias que definem as estruturas de dados primeiros, e após os processos que a utilizam, mas também existem metodologias que fazem o contrário, definindo primeiro os procedimentos que serão automatizados e posteriormente os dados que deverão ser utilizados. (FARINELLI, 2007).

2.2.1 Linguagens orientadas a objeto

Abaixo está listado as principais linguagens que são utilizadas para Programação Orientada a Objetos:

**Quadro 1 – Linguagens de Programação Orientadas a Objetos**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Linguagem | Herança | Polimorfismo | Encapsulamento | Abstração | Classe Abstrata | Interface | Construtores |
| **C#** | Sim | Sim | Sim | Sim | Sim | Sim | Sim |
| **Delphi** | Sim | Sim | Sim | Sim | Sim | Sim | Sim |
| **JavaScript** | Não | Sim | Parcial | Sim | Não | Sim | Sim |
| **PHP** | Sim | Sim | Sim | Sim | Sim | Sim | Sim |
| **Java** | Sim | Sim | Sim | Sim | Sim | Sim | Sim |

Fonte: Elaborado a partir da Documentação das Linguagens (2023)

C# (pronuncia-se "C sharp") é uma linguagem de programação moderna e orientada a objetos, desenvolvida pela Microsoft como parte da plataforma .NET. O diferencial do C# está na sua performance e eficiência, além da integração com outras tecnologias da Microsoft, como o .NET Framework e o Visual Studio. O C# apresenta uma sintaxe elegante e simples, além de recursos avançados de programação orientada a objetos, como encapsulamento, herança e polimorfismo. Um dos principais problemas enfrentados pelo C# atualmente é a sua limitação em relação à plataforma Windows. Embora a linguagem seja amplamente utilizada no desenvolvimento de aplicações para desktop e mobile na plataforma .NET, sua utilização em outros sistemas operacionais ainda é limitada. Além disso, algumas características do C# podem ser consideradas complexas e difíceis de entender para desenvolvedores iniciantes. (MICROSOFT, 2021).

Delphi é uma linguagem de programação orientada a objetos, originalmente criada pela Borland e atualmente mantida pela Embarcadero Technologies. O diferencial do Delphi está na sua produtividade e simplicidade, além da facilidade de uso e da grande comunidade de desenvolvedores que a utilizam. A linguagem Delphi oferece recursos avançados de programação orientada a objetos, incluindo herança, polimorfismo e interfaces, além de recursos adicionais, como componentes visuais e bibliotecas de terceiros. Um dos principais problemas enfrentados pelo Delphi é a sua limitação em relação à plataforma Windows. Embora a linguagem seja amplamente utilizada no desenvolvimento de aplicações para desktop e mobile, sua utilização em outros sistemas operacionais ainda é limitada. Além disso, a linguagem Delphi apresenta uma curva de aprendizado relativamente longa e pode ser considerada menos popular do que outras linguagens de programação. (EMBARCADERO, 2021).

JavaScript é uma linguagem de programação de script orientada a objetos, amplamente utilizada para desenvolvimento web. O diferencial do JavaScript está na sua capacidade de criar interatividade e dinamismo em páginas web, além da sua flexibilidade e compatibilidade com outros frameworks e bibliotecas. O JavaScript apresenta recursos avançados de programação orientada a objetos, como encapsulamento, herança prototipal e polimorfismo. Um dos principais problemas enfrentados pelo JavaScript atualmente é a segurança. Como a linguagem é executada no lado do cliente, é possível que hackers explorem vulnerabilidades na programação JavaScript para invadir sistemas ou roubar informações confidenciais. Além disso, a grande quantidade de bibliotecas e frameworks disponíveis para o JavaScript pode tornar o desenvolvimento complexo e difícil de gerenciar. (MOZILLA DEVELOPER NETWORK, 2021).

PHP é uma linguagem de programação orientada a objetos, amplamente utilizada para desenvolvimento web do lado do servidor. O diferencial do PHP está na sua facilidade de aprendizado e uso, além da grande quantidade de bibliotecas e extensões disponíveis para sua utilização. O PHP apresenta recursos avançados de programação orientada a objetos, como encapsulamento, herança, polimorfismo e interfaces, além de recursos adicionais, como bibliotecas de terceiros e extensões. Um dos principais problemas enfrentados pelo PHP é a sua segurança. Como a linguagem é amplamente utilizada no desenvolvimento web do lado do servidor, é possível que hackers explorem vulnerabilidades na programação PHP para invadir sistemas ou roubar informações confidenciais. Além disso, a linguagem PHP apresenta uma curva de aprendizado relativamente longa e pode ser considerada menos elegante e sofisticada do que outras linguagens de programação. (PHP, 2021).

Java é uma linguagem de programação orientada a objetos, desenvolvida pela Sun Microsystems e atualmente mantida pela Oracle. O diferencial do Java está na sua portabilidade, segurança e escalabilidade, além da grande quantidade de frameworks e bibliotecas disponíveis para sua utilização. O Java apresenta uma sintaxe simples e clara, além de recursos avançados de programação orientada a objetos, como encapsulamento, herança e polimorfismo. Além disso, a linguagem Java apresenta recursos adicionais, como segurança e portabilidade. Um dos principais problemas enfrentados pelo Java atualmente é a sua performance. Embora a linguagem seja amplamente utilizada em sistemas empresariais, sua performance pode ser considerada inferior a outras linguagens de programação, como C++ e Rust. Além disso, a utilização excessiva de frameworks e bibliotecas pode tornar o desenvolvimento complexo e difícil de gerenciar. (ORACLE, 2021).

2.3 Web services

Web Services são conjuntos de programas que podem ser publicados, buscados e chamados por meio da internet. Esses programas podem realizar um simples processo de troca de mensagens, como também complexas transações comerciais ou industriais, por exemplo, um processo de compra de produtos. Uma vez que um Web Service é publicado em um servidor web, vários programas e até mesmo outros Web Services podem acessá-lo e chamá-lo, tanto para obtenção de dados como para a interação com serviços que uma organização oferece (TAMAE, 2004).

2.3.1 API

Freitas, Birnfield e Saraiva (2021, p.43) definem API como, “[...] construções de aplicações que permitem que os desenvolvedores criem funcionalidades complexas mais facilmente. Tais construções abstraem o código mais complexo, proporcionando o uso de sintaxes de forma mais simples.”. Em relação à definição de API, Rodrigues e Neumann (2020) acrescentam que ela também pode ser descrita como um conjunto de procedimentos e diretrizes de um sistema, permitindo o acesso externo às funcionalidades sem a necessidade de acessar diretamente o código ou a implementação, mas de maneira abstrata e completamente independente da programação.

De acordo com MDN (2021), uma API é comumente constituída por um conjunto de métodos, propriedades, eventos e URLs padronizados, que podem ser utilizados no desenvolvimento de aplicações para facilitar a interação entre a aplicação, o servidor e serviços de terceiros.

2.3.2 Arquitetura REST

Segundo as afirmações de Rodrigues et al. (2020), o conceito de REST (Representational State Transfer) surgiu a partir de uma dissertação com o intuito de propor uma padronização para a integração entre aplicações baseadas no protocolo HTTP. Para complementar, Red Hat (2020) esclarece que REST não é um protocolo ou um padrão em si, mas sim um conjunto de restrições arquiteturais que oferece aos desenvolvedores a flexibilidade de utilizar a arquitetura de várias maneiras. Essas possibilidades vão desde comunicações simples para adquirir dados até uma aplicação que não armazena dados, mas obtém todas as informações diretamente do servidor através das comunicações estabelecidas pela arquitetura REST.

2.3.3 Métodos HTTP

Conforme apresentado por MDN (2021), o Protocolo HTTP define um conjunto de métodos de requisição que indicam o tipo de ação que se deseja processar. Cada método possui características próprias, mas podem compartilhar algumas de suas características com outros. Como podem ser observados os métodos e suas funções no Quadro 2.

**Quadro 1 – Métodos HTTP**

|  |  |
| --- | --- |
| **Método** | **Função** |
| GET | O método GET solicita a representação de um recurso específico. Requisições utilizando o método GET devem retornar apenas dados. |
| HEAD | O método HEAD solicita uma resposta de forma idêntica ao método GET, porém sem conter o corpo da resposta. |
| POST | O método POST é utilizado para submeter uma entidade a um recurso específico, frequentemente causando uma mudança no estado do recurso ou efeitos colaterais no servidor. |
| PUT | O método PUT substitui todas as atuais representações do recurso de destino pela carga de dados da requisição. |
| DELETE | O método DELETE remove um recurso específico. |
| CONNECT | O método CONNECT estabelece um túnel para o servidor identificado pelo recurso de destino. |
| OPTIONS | O método OPTIONS é usado para descrever as opções de comunicação com o recurso de destino. |
| TRACE | O método TRACE executa um teste de chamada loop-back junto com o caminho para o recurso de destino. |
| PATCH | O método PATCH é utilizado para aplicar modificações parciais em um recurso. |

Fonte: Elaborado a partir de MDN (2021).

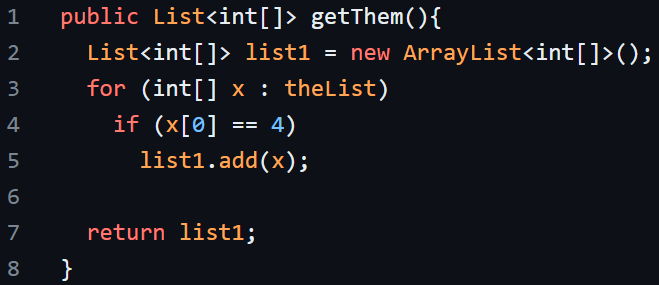
2.4 BOAS PRATICAS DE DESENVOLVIMENTO

Conforme Martin (2009, p. 8), “Um código limpo é simples e direto. Ele é tão bem legível quanto uma prosa bem escrita. Ele jamais torna confuso o objetivo do desenvolvedor, em vez disso, ele está repleto de abstrações claras e linhas de controle objetivas”.

Ainda conforme Martin, um código confuso diminui a produtividade da equipe, cada nova alteração cause problemas em outras duas ou três partes, exigindo remendos, amarrações para que o código não quebre e com o tempo, a bagunça cresce tanto que é impossível solucionar. Com a redução da produtividade, a gerência contrata novos funcionários para aumentar a mão de obra, mas os novos programadores não conhecem o código por completo, não conseguindo diferenciar a mudança que altera o propósito do projeto e a aquela que atrapalha. Criando mais e mais confusões, a produtividade chega cada vez mais perto de zero.

2.4.1 Nomeação

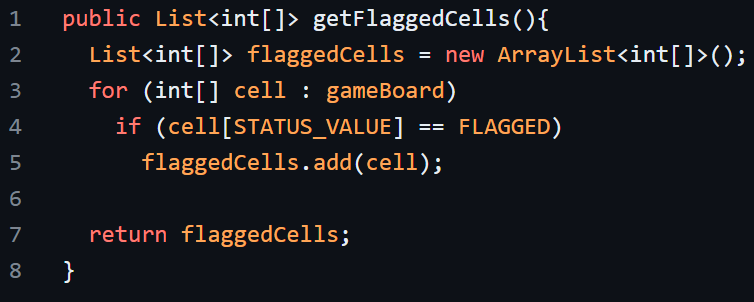
O início de boas práticas de desenvolvimento se dá pela nomeação de variáveis, métodos e classes, pois o nome deve dizer porque existe, o que faz e como é usado. (MARTIN, 2009). A Figura 1 mostra um código que não utiliza uma boa nomeação de suas variáveis, funções, etc.

**Figura 1 – Código sem boas práticas de nomeação**  


**Fonte: Martin (2009, p. 18)**

Esse código é simples, porém seu contexto não está implícito, dificultando o entendimento. A nomeação seguindo as boas práticas garante que em qualquer parte de um código, o seu contexto estará implícito, facilitando a leitura conforme a Figura 2.

**Figura 2 – Código sem boas práticas de nomeação**



**Fonte: Martin (2009, p. 19)**

Além de nomes que dão o contexto do código, é necessário utilizar nomes pronunciáveis. Como uma variável do tipo Date nomeada de “genymdhms” por exemplo, ela pode contextualizar a sua utilização, mas a mesma variável nomeada de “generationTimestamp” facilita muito mais o entendimento, além de melhorar a comunicação da equipe de desenvolvimento. (MARTIN, 2009)

Martin (2009) também salienta a utilização de nomes passíveis de busca, auxiliando na procura de determinadas variáveis, constantes e métodos por todo o projeto, pois utilizar o número 7 como uma constante é comum, mas buscar por todo o código pode demorar por ser um número que pode aparecer em diversos contextos, mas utilizar uma constante com nome MAX\_CLASSES\_PER\_STUDENT torna a pesquisa muito mais rápida.

2.4.2 Funções

A primeira regra para funções é que elas devem ser pequenas. Conforme Martin (2019). Ele afirma que não possui informações de estudos que indiquem que funções extremamente pequenas são superiores. Mas que pode afirmar que ao longo de aproximadamente quatro décadas tem desenvolvido funções de diferentes tamanhos, seja de 3000 linhas; funções de 100 a 300 linhas; e funções que apresentavam apenas 20 a 30 linhas. Essa experiência ensinou que, após inúmeras tentativas e erros, as funções devem ser muito compactas.

2.6. REACTJS

Conforme Roldán (2021), o React é uma biblioteca JavaScript de código aberto e flexível, projetada para a criação de interfaces de usuário complexas a partir de pequenos componentes separados. Com eficiência, o React aprimora a flexibilidade, a manutenção e o desempenho das aplicações, fornecendo um impulso significativo ao fluxo de trabalho, aumentando a velocidade sem comprometer a qualidade.

Na sua documentação, React (2022) enfatiza o objetivo do ReactJS de simplificar o desenvolvimento de interfaces de usuário interativas, permitindo que as interfaces se atualizem e sejam renderizadas automaticamente à medida que os dados são alterados. Eles também destacam a estrutura baseada em componentes encapsulados do ReactJS, que possibilita a fácil combinação desses componentes para criar interfaces mais complexas e reutilizáveis.

2.7. FRAMEWORK

Conforme afirmado por Freitas, Birnfield e Saraiva (2021, p.47) “Framework é uma abstração que une códigos comuns entre vários projetos de software, provendo uma funcionalidade genérica. Portanto, trata-se de uma forma mais simples de desenvolver aplicações pelo reuso de componentes.”.

Oliveira e Zanetti (2020, p.101) complementam a utilidade dos frameworks dizendo que “A adoção de frameworks facilita o desenvolvimento de aplicações, pois implementa as funções mais comumente utilizadas em determinada parte do projeto.”.

2.7.1. .NET Framework

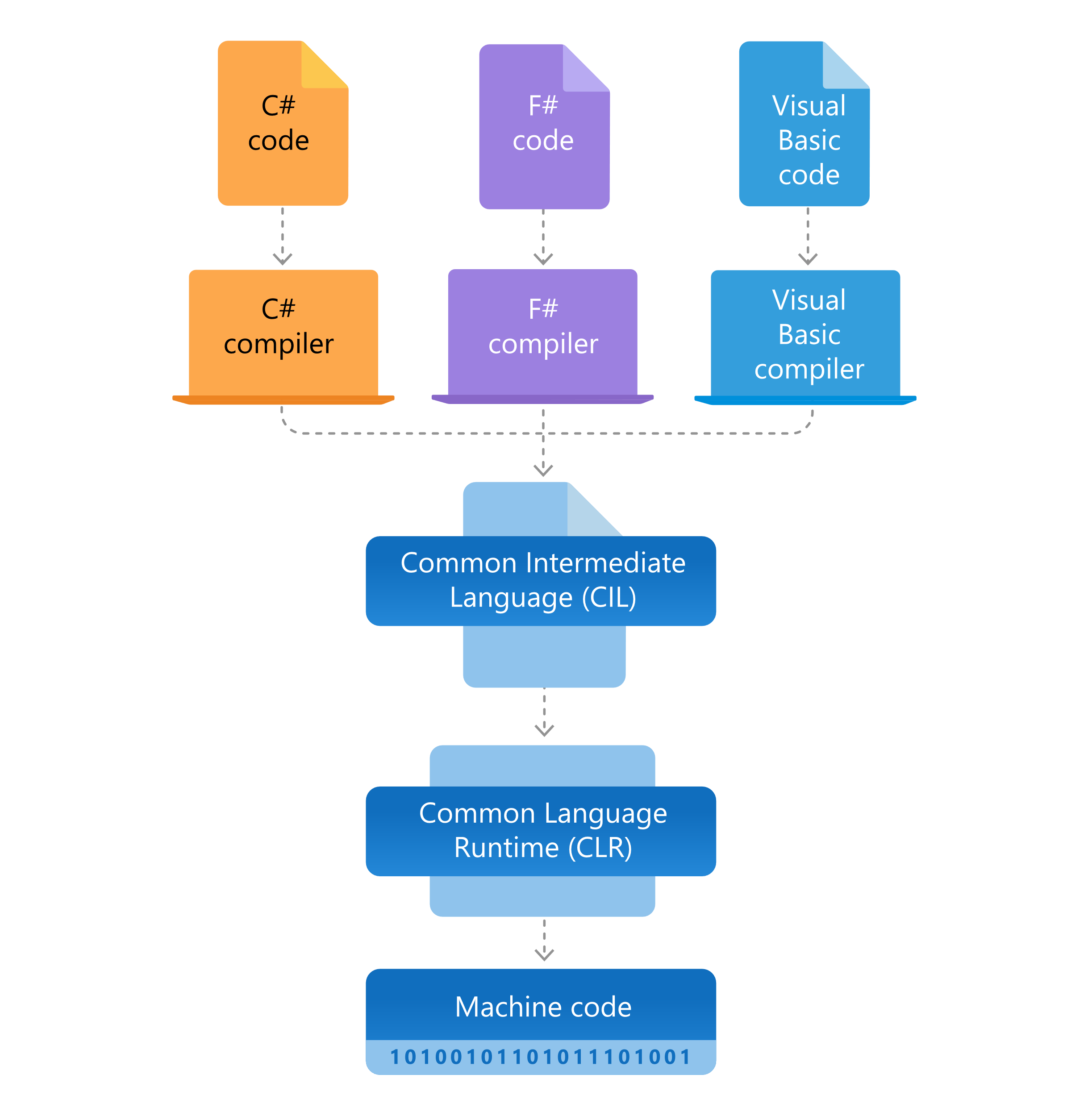
De acordo com a Microsoft (2023), o .NET Framework é uma plataforma de execução gerenciada para o Windows que disponibiliza uma variedade de recursos para os aplicativos em execução. Ele é composto por duas partes principais: o CLR (Common Language Runtime), responsável pela execução dos aplicativos, e a biblioteca de classes do .NET Framework, que oferece um conjunto de códigos testados e reutilizáveis que os desenvolvedores podem utilizar em seus próprios aplicativos.

Os dois componentes principais do .NET Framework são o Common Language Runtime e a Biblioteca de Classes do .NET Framework:

O Common Language Runtime (CLR) é o mecanismo de execução que identifica aplicativos em execução. Ele fornece serviços como gerenciamento de threads, coleta de lixo, fortemente tipado, identificação de exceção e muito mais. Já a biblioteca de classes oferece um conjunto de APIs e tipos para funcionalidades comuns. Ela oferece tipos de cadeias de caracteres, datas, números, etc. A biblioteca de classes inclui APIs para leitura e gravação de arquivos, conexão a bancos de dados, desenho e muito mais. (MICROSOFT, 2023, n.p.)

Cada linguagem de programação .NET tem um compilador que transforma seu código em Common Intermediate Language. Em runtime, o Common Language Runtime transforma o código compilado em código do computador e o executa, conforme a Figura 3. O código compilado é armazenado em conjuntos, seja arquivos com uma extensão de arquivo .dll ou .exe. (MICROSOFT, 2023).

**Figura 3 – Compilação no .NET Framework**



**Fonte: Microsoft (2023).**

2.7.2. Entity Framework

O Entity Framework é um framework de mapeamento objeto-relacional (ORM), “permitindo aos desenvolvedores trabalharem com dados relacionais como objetos específicos do domínio, eliminando a necessidade de grande parte do código de acesso aos dados que os desenvolvedores geralmente precisam escrever.”. (ENTITY FRAMEWORK, 2021, p.1).

Ainda conforme o Entity Framework (2021). Um ORM é uma ferramenta que possibilita a armazenagem automatizada de dados de objetos de domínio em um banco de dados relacional, como o MS SQL Server, sem a necessidade de muita programação. Ela é composta por três componentes principais: objetos de classe de domínio, objetos de banco de dados relacional e informações de mapeamento que definem como os objetos de domínio são relacionados aos objetos de banco de dados (tabelas, visualizações e procedimentos armazenados). A ORM auxilia na separação do design do banco de dados do design das classes de domínio, tornando a aplicação mais fácil de ser mantida e expandida. Além disso, ela automatiza as operações CRUD padrão (Create, Read, Update e Delete), evitando que o desenvolvedor tenha que escrevê-las manualmente.

HTML

CSS

JAVASCRIPT

Flanagan (2014) descreve o JavaScript como umas das três principais ferramentas para desenvolvimento web, juntamente com HTML e CSS, sendo uma tecnologia presente em praticamente todos os sites da atualidade. Oliveira (2020) complementa dizendo que é uma linguagem de programação orientada a objetos, interpretada e executada por navegadores web e seu objetivo principal é oferecer maio interatividade às páginas.

Na verdade, o nome “JavaScript” é um pouco enganoso. A não ser pela semelhança sintática superficial, JavaScript é completamente diferente da linguagem de programação Java. E JavaScript já́ deixou para trás suas raízes como linguagem de script há́muito tempo, tornando-se uma linguagem de uso geral robusta e eficiente. A versão mais recente da linguagem [...] define novos recursos para desenvolvimento de software em grande escala. (FLANAGAN, 2014, p.18).

MDN (2021) adverte sobre JavaScript de que ele não deve ser confundido com a linguagem de programação Java, mesmo tendo os nomes de suas marcas comerciais bastante semelhantes são duas linguagens de programação significativamente diferentes em sintaxes, semânticas e também bem em seus casos de uso.

Sobre a linguagem JavaScript, Oliveira (2020, p.46) certifica que “Uma característica importante da linguagem JavaScript é que ela não apresenta tipos de dados, isto é, qualquer variável definida é do tipo variante.”, e Oliveira e Zanetti (2021, p.8) complementam dizendo que “O tipo de dados de determinada variável pode ser modificado ao longo da execução da aplicação, conforme o seu conteúdo vai sendo alterado.”.

SWAGGER

A abordagem fornecida pelo Swagger, também conhecido como OpenAPI, desempenha um papel fundamental na documentação e desenvolvimento de APIs. Conforme ressaltado pela própria documentação do Swagger, essa especificação oferece um padrão linguagem-agnosticado para descrever APIs RESTful, permitindo, assim, a colaboração eficaz entre equipes multidisciplinares. Através da definição padronizada de endpoints, parâmetros, respostas e autenticação, o Swagger promove uma compreensão clara da API, facilitando a geração automatizada de código em diversas linguagens (Swagger, 2023).

Swagger (2023) afirma que a capacidade do Swagger de manter uma fonte única de documentações sobre a API e melhorar a clareza da comunicação entre as partes interessadas é o que o torna muito utilizado. Além disso, destaca que a ferramenta possibilita a realização de testes automatizados para garantir a qualidade da API. Ao adotar o Swagger, as equipes podem alcançar uma documentação consistente e uma comunicação fluida, contribuindo para a qualidade e sucesso do desenvolvimento de APIs.

JWT

O JSON Web Token (JWT) é um padrão aberto definido pelo RFC 7519, com o objetivo de estabelecer um método compacto e independente para transmitir com segurança informações do usuário entre cliente e servidor. Essas informações são encapsuladas em um objeto JSON e podem ser enviadas através de um cabeçalho HTTP. O token JWT resultante é armazenado no dispositivo do usuário, e suas informações são protegidas por meio de criptografia usando um segredo conforme a Figura 4. Isso é realizado por meio de algoritmos HMAC (Hash-based Message Authentication Code) ou por meio de pares de chaves pública e privada. A autenticidade do token é garantida, permitindo que suas informações sejam verificadas em cada solicitação (Jones et al., 2015).

**Figura 4 – Exemplo de Json Web Token**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**Fonte: JWT (2023).**

Diversas bibliotecas que implementam o padrão JWT estão disponíveis para várias linguagens de programação, incluindo .NET, Python, NodeJs, Java, JavaScript, Perl, Ruby e PHP. Essas bibliotecas são utilizadas tanto para autenticação de usuários, onde o token é enviado a cada solicitação para permitir ou negar o acesso a determinadas rotas, serviços ou recursos, quanto para garantir a transmissão segura de informações. Isso ocorre porque o conteúdo do token, devido à sua assinatura criptografada, pode ser verificado quanto a adulterações (JWT, 2023).

3. Metodologia DA PESQUISA

O presente trabalho caracteriza-se como pesquisa aplicada e descritiva, pois foi desenvolvido um protótipo da aplicação e também teve como um dos objetivos verificar se o protótipo desenvolvido tem as funcionalidades necessárias para possibilitar o uso comercial.

O trabalho buscou responder o seguinte problema: É possível desenvolver um sistema para otimizar os processos de um escritório de projetos agrícolas?

Após a finalização do levantamento bibliográfico, foi realizado um estudo mais aprofundado sobre o desenvolvimento de aplicações com a arquitetura cliente servidor, utilizando as ferramentas e padrões de desenvolvimentos mais modernos possíveis, visando deixar a aplicação relevante por mais tempo, evitando futuros retrabalhos, migrações para novos padrões e facilitando possíveis adições futuras ao trabalho.

Com base nessa pesquisa foi definido que a aplicação seria feita na IDE Visual Studio, com a linguagem C#, integrando com o banco de dados SQL Server por meio do Entity Framework e design baseado no conjunto de boas práticas definidos pela Microsoft na documentação mais atualizada das tecnologias. Além disso será utilizado o ReactJS para desenvolver a parte visual da aplicação. Essas escolhas foram feitas por serem tecnologias de grandes empresas que atualizar frequentemente as ferramentas com melhorias.4. PROTÓTIPO DE SISTEMA PARA GERENCIAMENTO EFICIENTE DE ESCRITÓRIOS DE PROJETOS AGRÍCOLAS

Neste capítulo são apresentados todos os aspectos técnicos do processo de análise de implementação do protótipo.

4.1 MAPEAMENTO DE CENÁRIO

4.2 ANÁLISE

No capítulo da revisão bibliográfica, foram explorados tópicos referentes à engenharia de software, os quais foram indispensáveis para serem aplicados nessa fase, a fim de adquirir o conhecimento essencial das necessidades do negócio do cliente e dos propósitos da aplicação. Nesse contexto, a análise desempenha um papel crucial ao esclarecer o que será elaborado. Além disso, a análise também desempenha um papel fundamental na eficiência da manutenção da aplicação, uma vez que todos os procedimentos empregados ao longo do desenvolvimento estão devidamente detalhados, simplificando a resolução de quaisquer problemas que possam surgir.

4.2.1 Visão Geral do Protótipo

4.2.2 Requisitos

Nesta seção estão elencados os requisitos para o funcionamento da aplicação. Esses requisitos deverão ser desenvolvidos para que seja atendido a necessidade do cliente, e também, cumprindo aquilo que foi proposto. O Quadro X, apresenta os requisitos funcionais necessários para o desenvolvimento do protótipo e as regras de negócio serão apresentadas no Quadro X.

**Quadro X – Requisitos Funcionais**

|  |  |  |  |
| --- | --- | --- | --- |
| **Número** | **Nome** | **Descrição** | **Regras de Negócio** |
| RF01 | Login | O protótipo deverá permitir que o usuário realize o login para acessar determinados recursos da aplicação. Para realizar o acesso será necessário informar o e-mail e senha. |  |
| RF02 | Logout | O protótipo deverá permitir o usuário finalizar seu acesso. |  |
| RF03 | Menu | O protótipo deverá possuir um menu que possibilite o acesso aos recursos da aplicação de a qualquer momento. |  |
| RF04 | Cadastro de Usuários | O protótipo deve permitir o cadastro de usuários. Sendo necessário informar o email, senha e confirmação da senha. |  |
| RF05 | Cadastro de Clientes | O protótipo deve permitir que o usuário cadastre os clientes. Os campos para o cadastro são Nome (obrigatório), Apelido (opcional), CPF/CNPJ (obrigatório), Observação (opcional)... |  |
| RF06 | Cadastro de Proposta | O protótipo deve permitir que o usuário cadastre suas propostas, que possuem tipos diferentes (Custeio, Investimento, Etc.). Os campos para o cadastro são Tipo (obrigatório), Cliente (obrigatório), Observação (opcional)... |  |
| RF07 | Cadastro de Documentos | O protótipo deve permitir o cadastro de documentos, que podem ter origens distintas (cliente, terreno, proposta), portanto os campos são Nome (obrigatório), Tipo (obrigatório), Arquivo (obrigatório), Observação (opcional), origem (obrigatório). Dependendo do tipo da proposta os campos irão possuir obrigatoriedade diferente, além de campos diferentes, onde o protótipo irá tratar. |  |

**Quadro X – Requisitos Não Funcionais**

|  |  |
| --- | --- |
| **Número** | **Descrição** |
| RNF01 | O protótipo deve ser desenvolvido com C# e ReactJS. |
| RNF02 | As telas de cadastros devem possuir tratamento para valores nulos no front-end. |
| RNF03 | Todos os processos devem possuir autenticação e autorização de usuário. |
| RNF04 | O protótipo deverá implementar JWT para autenticação e autorização, expirando em 2 horas. |
| RNF05 | Para o armazenamento de dados deverá ser utilizado SQL Server. |
| RNF06 | Para gerenciamento do banco de dados deverá ser utilizado o Entity Framework. |
| RNF07 | Todo registro deve ter controle de status. |
| RNF08 | O controle de auditoria deverá ser padronizado, indicando o usuário da ação e a data/hora. |

5. CONCLUSÃO

**REFERÊNCIAS**

ALVES, William P. **Banco de Dados**. São Paulo Editora Saraiva, 2014. E-book. ISBN 9788536518961. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788536518961/. Acesso em: 24 mar. 2023.

BARNES, D. J.; KÖLLING, M. **Programação Orientada a Objetos com Java**. 3. ed. São Paulo: Pearson, 2012.

CODD, Edgar Frank. (1985). **Is Your DBMS Really Relational?** Computerworld, 19(30), 29-33. Disponível em: https://www.analytictech.com/borgatti/whatEver/readingsem/Codd.pdf. Acesso em: 24 mar. 2023.

DIAS, Ariel da S. **Administração de bancos de dados**. São Paulo Editora Saraiva, 2021. E-book. ISBN 9786589965688. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9786589965688/. Acesso em: 24 mar. 2023.

DIAS, Ariel da S. **Administração de bancos de dados**. São Paulo Editora Saraiva, 2021. E-book. ISBN 9786589965688. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9786589965688/. Acesso em: 24 mar. 2023.

Embarcadero. **RAD Studio Documentation**. Disponível em: https://docwiki.embarcadero.com/RADStudio/Sydney/en/Main\_Page. Acesso em: 12 maio 2023.

FARINELLI, Fernanda. **Conceitos básicos de programação orientada a objetos**. Instituto Federal Sudeste de Minas Gerais, 2007.

FERREIRA, Arthur Gonçalves**. Interface de programação de aplicações (API) e web services.** São Paulo Platos Soluções Educacionais 2021 1 recurso online ISBN 9786553560338.

FRAMEWORK, Entity. **Entity framework**. Architecture [edit], v. 21, p. 22, 2021.

FREITAS, Pedro Henrique C. et al. **PROGRAMAÇÃO back end 3**. Porto Alegre SAGAH 2021 1 recurso online ISBN 9786581492274.

MARTIN, Robert C. **Código limpo: habilidades práticas do Agile software**. Rio de Janeiro Alta Books, 2009.

MDN. **Glossário**. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Glossary>. Acesso em: 29 jun. 2022.

MDN. **Métodos de requisição HTTP**. Disponível em: <https://developer.mozilla.org/ptBR/docs/Web/HTTP/Methods>. Acesso em: 29 jun. 2022.

Microsoft. **C# language reference**. Disponível em: https://docs.microsoft.com/pt-br/dotnet/csharp/. Acesso em: 12 mai. 2023.

MICROSOFT. **O que é .NET Framework?**.Disponível em: <https://dotnet.microsoft.com/pt-br/learn/dotnet/what-is-dotnet-framework>. Acesso em: 29 jun. 2023.

Mozilla. **Javascript**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript. Acesso em: 12 mai. 2023.

OLIVEIRA, Cláudio Luís Vieira. **JavaScript descomplicado : programação para a Web, IoT e dispositivos móveis**. São Paulo Erica 2020 1 recurso online ISBN 9788536533100.

Oracle. **Java SE Documentation**. Disponível em: https://docs.oracle.com/en/java/javase/14/. Acesso em: 12 mai. 2023.

PHP. **PHP: Hypertext Preprocessor**. Disponível em: https://www.php.net/manual/pt\_BR/. Acesso em: 12 mai. 2023.

Pritchett, D. (2008). **BASE: An Acid Alternative**. ACM Queue, 6(3), 48-55. Disponível em: https://dl.acm.org/doi/pdf/10.1145/1394127.1394128. Acesso em: 24 mar. 2023.

REACT. **Uma biblioteca Javascript para criar interfaces de usuário.** Disponível em: <https://pt-br.react.dev/learn>. Acesso em 22 junho. 2023.

RED HAT. **API REST**. 2020. Disponível em: <https://www.redhat.com/pt-br/topics/api/whatis-a-rest-api>. Acesso em: 29 jun. 2023.

ROLDÁN, Carlos Santana. **React 17:** design patterns and best practices. 3. ed. Birmingham: Packt, 2021. Ebook.

SADAMOTO, A. M. A. et al. **NoSQL databases: an overview**. In: Proceedings of the 2012 Brazilian Symposium on Computing Systems Engineering (SBESC). IEEE, 2012.

SILVA, Luiz F. Calaça et al. **Banco de dados não relacional**. Porto Alegre SAGAH 2021 1 recurso online ISBN 9786556901534.

TAMAE, André. **Desenvolvimento de Aplicações Web com JSP, Servlets, JavaServer Faces, Hibernate, Ajax e Web Services**. São Paulo: Érica, 2004.

FLANAGAN; DAVID. Javascript. Porto Alegre: Grupo A, 2014. Ebook.

SWAGGER. **About Swagger**. Disponível em: <https://swagger.io/about/>. Acesso em: 23 ago. 2023.

JONES, M., Bradley, J., Sakimura, N. (2015) “**JSON Web Token (JWT) – RFC 7519**”,

Internet Engineering Task Force, IEFT. Disponível em: <https://tools.ietf.org/pdf/rfc7519.pdf>. Acesso em: 23 ago. 2023.

**JWT.IO** (2023). Introduction to JSON Web Tokens. Disponível em: <https://jwt.io/introduction>. Acesso em: 23 ago. 2023.

**ANEXOS**