



FACULDADE DE TECNOLOGIA DE SÃO PAULO
DEPARTAMENTO DE TECNOLOGIA DA INFORMAÇÃO

Alisson Magalhães Silva
Mariana Nogueira de Deus
Matheus de Albuquerque Dimolitsas
Wesley Silva Pestana

**ANÁLISE DA IMPLEMENTAÇÃO DE CÓDIGO POR GERADORES DE
CÓDIGO A PARTIR DA DESCRIÇÃO DE CASOS USOS**

Relatório Final do Trabalho de Conclusão de Curso

São Paulo

Junho de 2024



FACULDADE DE TECNOLOGIA DE SÃO PAULO

DEPARTAMENTO DE TECNOLOGIA DA INFORMAÇÃO

Alisson Magalhães Silva

Mariana Nogueira de Deus

Matheus de Albuquerque Dimolitsas

Wesley Silva Pestana

ANÁLISE DA IMPLEMENTAÇÃO DE CÓDIGO POR GERADORES DE CÓDIGO A PARTIR DA DESCRIÇÃO DE CASOS USOS

Relatório Final do Trabalho de Conclusão de Curso

Trabalho submetido como exigência
parcial para a obtenção do Grau de
Tecnólogo em Análise e
Desenvolvimento de Sistemas

Orientadora: Prof^ª Dra. Márcia Ito

São Paulo

Junho de 2024

RESUMO

SILVA M. A. DEUS N. M. DIMOLITSAS A. M. PESTANA S. W. Relatório Técnico (Trabalho de Conclusão de Curso) – Faculdade de Tecnologia de São Paulo, São Paulo/SP, 37f, 2024.

O relatório desenvolvido para o projeto do trabalho de conclusão de curso aborda a criação de software utilizando geradores de códigos a partir de casos de uso. O objetivo da pesquisa é determinar a quantidade e o tipo de informações que um gerador de código precisa para produzir um software funcional e eficiente que atenda os requisitos do usuário. A pesquisa envolveu a realização de solicitações a diferentes geradores de código, utilizando diversas abordagens, para avaliar suas capacidades de atender às solicitações dos usuários. Constatou-se que, para obter sucesso na utilização dos geradores de código, é necessário possuir um conhecimento técnico básico. Isso permite que as solicitações sejam feitas de maneira adequada, resultando em software que atenda aos requisitos desejados. A pesquisa concluiu que, apesar da utilidade dos geradores de código, o conhecimento técnico básico é imprescindível para a criação de software funcional e eficiente, capaz de atender aos requisitos do usuário de maneira satisfatória.

Palavras-Chave: Gerador, Código, Desenvolver

ABSTRACT

SILVA M. A., DEUS N. M., DIMOLITSAS A. M., PESTANA S. W. Technical Report (Final Undergraduate Project) – – Faculdade de Tecnologia de São Paulo, São Paulo/SP, 37 pages, 2024.

The report developed for the final project addresses the creation of software using code generators based on use cases. The objective of the research is to determine the amount and type of information a code generator needs to produce functional and efficient software that meets user requirements. The research involved making requests to different code generators, using various approaches, to evaluate their ability to meet user requests. It was found that, to successfully use code generators, basic technical knowledge is necessary. This allows requests to be made appropriately, resulting in software that meets the desired requirements. The research concluded that, despite the usefulness of code generators, basic technical knowledge is essential for creating functional and efficient software that satisfactorily meets user requirements.

Keywords: Generator, Code, Develop

Sumário

1. INTRODUÇÃO.....	2
2. OBJETIVO.....	3
3. MÉTODO DE PESQUISA.....	3
4. ATIVIDADES REALIZADAS	3
5. RESULTADOS	4
5.1. ChatGPT	4
5.2. Tabnine	15
5.3. Codeium	27
6. CONCLUSÕES.....	36
7. TRABALHOS FUTUROS.....	36
8. AGRADECIMENTOS.....	36
9. REFERÊNCIAS BIBLIOGRÁFICAS.....	37

1. INTRODUÇÃO

“A inteligência artificial é um campo da ciência da computação que se dedica ao estudo e ao desenvolvimento de máquinas e programas computacionais capazes de reproduzir o comportamento humano na tomada de decisões e na realização de tarefas, desde as mais simples até as mais complexas” (Guitarra, 2024).

A IA funciona coletando e combinando um grande volume de dados e identificando padrões nesse conjunto de informações. Isso é feito por meio de métodos como *machine learning* e *deep learning*.

Ela utiliza algoritmos que simulam o raciocínio humano com base nos padrões aprendidos, permitindo que os programas executem tarefas complexas.

Desde 2020, temos visto a Inteligência Artificial se tornando acessível ao público em geral e, de certa forma, transformando-se em um produto tanto para pessoas quanto para empresas. Segundo o relatório "*Artificial Intelligence – In-depth Market Insights & Data Analysis*", publicado pelo site de inteligência de dados Statista, prevê-se que a receita do mercado de softwares de inteligência artificial aumente 35% ao ano até 2025, alcançando um valor de 126 bilhões de dólares.

Uma Inteligência Artificial (IA) geradora de código é um tipo de software que emprega algoritmos de aprendizado de máquina e inteligência artificial para automatizar o processo de escrita de código. Essas ferramentas são projetadas para analisar o código já existente, compreender sua estrutura e identificar padrões de programação. Com base nessa análise, a IA é capaz de gerar novos trechos de código que se encaixam nos padrões identificados. (McFarland, 2024)

Essa tecnologia pode ser extremamente útil para desenvolvedores de software, pois pode acelerar o processo de desenvolvimento, reduzir erros e automatizar tarefas repetitivas. Além disso, as IA geradoras de código podem ajudar a aumentar a produtividade dos desenvolvedores, permitindo que se concentrem em aspectos mais criativos e complexos do desenvolvimento de software.

2. OBJETIVO

O objetivo desta pesquisa consiste em investigar e utilizar geradores de códigos gratuitos disponíveis na nuvem, analisando o que pode ser gerado com essas ferramentas a partir de casos de uso e verificar se o que foi solicitado foi realizado.

3. MÉTODO DE PESQUISA

A Metodologia adotada consiste em:

1. Buscar e selecionar os geradores de código
2. Estudo qualitativo de geradores de código selecionados
3. Definição de um Caso de Uso para ser usado nos geradores de códigos
4. Usar as definições do caso de uso e gerar os códigos pelos geradores
5. Analisar o que foi gerado e verificar se o que foi solicitado foi realizado
6. Registrar as conclusões da análise.

4. ATIVIDADES REALIZADAS

Com o início do projeto, foram escolhidos três geradores para os testes: ChatGPT, Tabnine e Codeium. Com a definição dos geradores, o caso de uso escolhido foi "Avaliar Aluno". Para este caso de uso, foi proposta a criação de uma interface onde seja possível avaliar a frequência e o desempenho teórico do aluno, bem como recuperar e enviar as notas diretamente para o aluno.

O caso de uso "Avaliar Aluno" foi selecionado para tentativa de desenvolvimento. Neste contexto, propõe-se o desenvolvimento de uma interface dedicada a essa funcionalidade, projetada para avaliar tanto a frequência quanto o desempenho teórico dos alunos, além de permitir a recuperação e o envio das notas diretamente para o aluno.

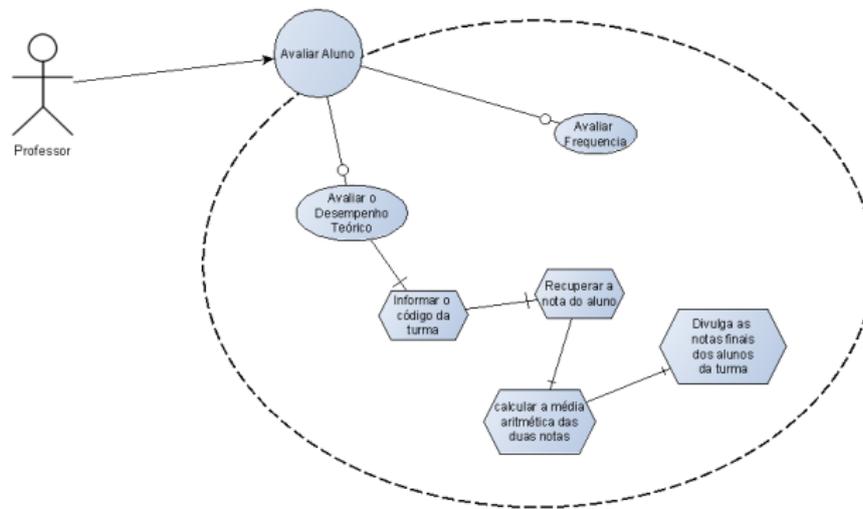


Figura 1: Caso de Uso Avaliar Aluno

5. RESULTADOS

5.1. ChatGPT

Com o Caso de Uso, deram-se início às primeiras tentativas de ter um programa feito pelo ChatGPT usando apenas os casos de uso.

Ao utilizar apenas o Caso de Uso, o gerador de código pode fornecer respostas genéricas, sendo necessário aprofundar mais o que está sendo solicitado para obter um resultado mais satisfatório. Se o prompt estiver muito aberto à interpretação, ele possivelmente solicitará mais contexto sobre a funcionalidade, pedindo mais detalhes, por exemplo:

Considerando um caso de uso "Avaliar Aluno", foi solicitado um programa que avalie o aluno, a resposta foi:

“Claro, posso te ajudar com isso! Para criar um programa que avalie um aluno, precisamos de mais detalhes sobre como essa avaliação será feita. Você gostaria de um programa que avalie o aluno com base em notas de provas, trabalhos, participação em sala de aula, ou de outra forma? E qual seria o critério de avaliação?”.

Depois de fornecer um contexto simples para o gerador, ele retorna um código para o usuário, porém um código muito simples:

Um programa que analise pelo menos 2 avaliações e suas faltas

Entendi! Vamos criar um programa simples em Python que permita avaliar um aluno com base em duas notas e sua quantidade de faltas. Vou mostrar um exemplo básico para você:

```
python Copiar código

def calcular_media(nota1, nota2):
    return (nota1 + nota2) / 2

def avaliar_aluno(nota1, nota2, faltas):
    media = calcular_media(nota1, nota2)
    if faltas > 10:
        return "Reprovado por faltas"
    elif media >= 7.0:
        return "Aprovado"
    elif 4.0 <= media < 7.0:
        return "Recuperação"
    else:
        return "Reprovado"

def main():
    nota1 = float(input("Digite a nota da primeira avaliação: "))
    nota2 = float(input("Digite a nota da segunda avaliação: "))
    faltas = int(input("Digite a quantidade de faltas do aluno: "))

    resultado = avaliar_aluno(nota1, nota2, faltas)
    print("Resultado da avaliação:", resultado)

if __name__ == "__main__":
    main()
```

Figura 2: Código gerado pelo chat GPT

Com essa resposta, é possível notar alguns outros detalhes: a decisão sobre a linguagem na qual o programa foi definido e não foi incluída uma interface gráfica, pois o código fornecido executa apenas no terminal. Para obter uma interface gráfica, é necessário especificar isso para o ChatGPT.

Conforme se usa o ChatGPT, ele vai criando um contexto para aquela conversa. Se o usuário pedir um programa em uma linguagem específica X, e em seguida pedir outro programa sem especificar nenhuma linguagem, o ChatGPT fará o programa na linguagem X.

Levando em conta as informações anteriores, foi decidido criar um programa com HTML, CSS e Javascript, para obter prompts mais precisos e avaliar o código gerado.

O que se pode observar do ChatGPT é que sua entrega sempre é aquilo que você pediu de forma simples. Em um projeto, ele não determina qual arquitetura ou design patterns o projeto usará, a menos que você solicite um projeto que utilize um framework que já venha com uma arquitetura estabelecida em sua base.

É possível analisar isso no front-end com Prompt: “crie uma interface HTML com Bootstrap com 3 input: Código do Aluno, Nota 1 Nota 2 “ e “ faça o código Javascript com as funções de validação dos campos e o cálculo da média”.

```

<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Formulário de Notas</title>
  <!-- Incluindo o Bootstrap CSS -->
  <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="
</head>
<body>

<div class="container mt-5">
  <h2>Formulário de Notas</h2>
  <form id="formNotas">
    <div class="form-group">
      <label for="codigoAluno">Código do Aluno:</label>
      <input type="text" class="form-control" id="codigoAluno" placeholder="Digite o código
    </div>
    <div class="form-group">
      <label for="nota1">Nota 1:</label>
      <input type="number" class="form-control" id="nota1" placeholder="Digite a nota 1" r
    </div>
    <div class="form-group">
      <label for="nota2">Nota 2:</label>
      <input type="number" class="form-control" id="nota2" placeholder="Digite a nota 2" r
    </div>
    <button type="submit" class="btn btn-primary">Calcular Média</button>
  </form>
</div>

<!-- Incluindo o Bootstrap JS (opcional) -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script

```

Figura 3: Código front-end gerado pelo Chat GPT 1

```
<script>
  // Função para calcular a média das notas
  function calcularMedia(nota1, nota2) {
    return (parseFloat(nota1) + parseFloat(nota2)) / 2;
  }

  $(document).ready(function(){
    $('#formNotas').submit(function(event){
      // Prevenir o envio do formulário padrão
      event.preventDefault();

      // Obter os valores dos campos
      var codigoAluno = $('#codigoAluno').val();
      var nota1 = $('#nota1').val();
      var nota2 = $('#nota2').val();

      // Validar se os campos estão preenchidos
      if(codigoAluno && nota1 && nota2) {
        // Calcular a média
        var media = calcularMedia(nota1, nota2);

        // Exibir a média
        alert('A média do aluno ' + codigoAluno + ' é: ' + media.toFixed(2));
      } else {
        // Exibir mensagem de erro se algum campo estiver vazio
        alert('Por favor, preencha todos os campos.');
      }
    });
  });
</script>

</body>
</html>
```

Figura 4: Código Javascript gerado para o front-end

O código entregue pelo ChatGPT é funcional, porém é perceptível que ele não seguiu nenhum padrão de desenvolvimento. O gerador sugeriu apenas um arquivo HTML com código Javascript. Caso haja a necessidade de escalar o software no futuro ou mesmo criar um software para uso profissional, um iniciante enfrentará grandes problemas. Mesmo ao utilizar uma ferramenta de geração de código, o usuário precisará entender o que está solicitando e compreender o que o gerador de código está retornando.

Com o objetivo de testar as habilidades de back-end do ChatGPT, decidiu-se criar uma API com Node.js que integra o front-end com um banco de dados. Foram criadas duas tabelas no banco de dados:

Tabela Aluno: nome, sobrenome, data_nascimento, turma_id, nota1, nota2, media.

Tabela Turma: nome, descricao

Os scripts de criação das tabelas no banco de dados foram gerados pelo próprio ChatGPT.

No back-end com Node.js, foi criada uma API para que o front-end pudesse acessar as operações no banco de dados. O ChatGPT sugeriu a utilização da arquitetura em módulos, que é uma abordagem comum em projetos Node.js, consistindo na separação do projeto em Controladores, Rotas e Modelos.

Antes de criar o código da API, foi estabelecido o código para a conexão com o banco de dados. Nesse sentido, o ChatGPT sugeriu a utilização da biblioteca Dotenv, disponível no pacote NPM, para gerenciar as variáveis de ambiente contendo as credenciais do banco de dados no arquivo .env.

O arquivo de conexão gerado pelo chat-gpt:

```
1  const { Pool } = require('pg');
2  require('dotenv').config();
3
4  const pool = new Pool({
5    user: process.env.DB_USER,
6    password: process.env.DB_PASSWORD,
7    host: process.env.DB_HOST,
8    port: process.env.DB_PORT,
9    database: process.env.DB_DATABASE
10 });
11
12 module.exports = pool;
```

Figura 5: Código front-end gerado pelo Chat GPT 1

Com o arquivo de conexão criado, começamos pelo desenvolvimento dos controladores. Como existem duas tabelas no banco de dados, foram gerados dois controladores:

Turma:

```
const pool = require('../connection/connection');

const criarTurma = async (nome, descricao) => {
  try {
    const result = await pool.query('INSERT INTO turma (nome, descricao) VALUES ($1, $2) RETURNING *',
    return result.rows[0];
  } catch (error) {
    throw new Error('Erro ao criar turma');
  }
};

const obterTodasTurmas = async () => {
  try {
    const result = await pool.query('SELECT * FROM turma');
    return result.rows;
  } catch (error) {
    throw new Error('Erro ao obter turmas');
  }
};

// Implemente outras operações CRUD para turma aqui...

module.exports = { criarTurma, obterTodasTurmas };
```

Figura 6: Código Back-end gerado pelo chat GPT

Aluno:

```
const pool = require('../connection/connection');

const criarAluno = async (nome, sobrenome, data_nascimento, turma_id, nota1, nota2) => {
  try {
    const media = (nota1 + nota2) / 2; // Calcula a média
    const result = await pool.query('INSERT INTO aluno (nome, sobrenome, data_nascimento, turma_id, nota1, nota2) VALUES (?, ?, ?, ?, ?, ?)');
    return result.rows[0];
  } catch (error) {
    throw new Error('Erro ao criar aluno');
  }
};

const atualizarNotasEMedia = async (id, nota1, nota2) => {
  try {
    // Recalcula a média
    const media = (nota1 + nota2) / 2;

    // Atualiza as notas e a média no banco de dados
    const result = await pool.query('UPDATE aluno SET nota1 = $1, nota2 = $2, media = $3 WHERE id = $4');

    return result.rows[0];
  } catch (error) {
    throw new Error('Erro ao atualizar notas e média do aluno');
  }
};
```

Figura 7: Código Javascript com Node JS que faz integração com banco de dados

```
const obterTodosAlunos = async () => {
  try {
    const result = await pool.query('SELECT * FROM aluno');
    return result.rows;
  } catch (error) {
    throw new Error('Erro ao obter alunos');
  }
};

// Implemente outras operações CRUD para aluno aqui...

module.exports = { criarAluno, obterTodosAlunos, atualizarNotasEMedia};
```

Figura 8: Código Javascript com Node JS que faz a busca de alunos no banco

Com os controladores criados, agora é necessário criar as rotas que irão utilizar esses controladores. Para que as rotas possam lidar com as requisições HTTP, o gerador de código sugeriu a instalação da biblioteca Express via NPM. Com a biblioteca instalada, o código sugerido para as requisições foi:

Turma:

```
const express = require('express');
const router = express.Router();
const turmaController = require('../controllers/turmaController');

router.post('/', async (req, res) => {
  const { nome, descricao } = req.body;
  try {
    const turma = await turmaController.criarTurma(nome, descricao);
    res.json(turma);
  } catch (error) {
    console.error('Erro ao criar turma:', error);
    res.status(500).json({ error: 'Erro ao criar turma' });
  }
});

router.get('/', async (req, res) => {
  try {
    const turmas = await turmaController.obterTodasTurmas();
    console.log(turmas);

    res.json(turmas);
  } catch (error) {
    console.error('Erro ao obter turmas:', error);
    res.status(500).json({ error: 'Erro ao obter turmas' });
  }
});

module.exports = router;
```

Figura 9: Código Javascript com Node JS que controla as rotas do software

Aluno:

```
const express = require('express');
const router = express.Router();
const alunoController = require('../controllers/alunosController');

router.post('/', async (req, res) => {
  const { nome, sobrenome, data_nascimento, turma_id, nota1, nota2 } = req.body;
  try {
    const aluno = await alunoController.criarAluno(nome, sobrenome, data_nascimento, turma_id, nota1, nota2);
    res.json(aluno);
  } catch (error) {
    console.error('Erro ao criar aluno:', error);
    res.status(500).json({ error: 'Erro ao criar aluno' });
  }
});

router.put('/:id', async (req, res) => {
  const { id } = req.params;
  const { nota1, nota2 } = req.body;
  try {
    const aluno = await alunoController.atualizarNotasEMedia(id, nota1, nota2);
    res.json(aluno);
  } catch (error) {
    console.error('Erro ao atualizar notas e média do aluno:', error);
    res.status(500).json({ error: 'Erro ao atualizar notas e média do aluno' });
  }
});
```

Figura 10: Código Javascript com Node JS que controla as rotas do software 2

```
router.put('/:id', async (req, res) => {
  const { id } = req.params;
  const { nota1, nota2 } = req.body;
  try {
    const aluno = await alunoController.atualizarNotasEMedia(id, nota1, nota2);
    res.json(aluno);
  } catch (error) {
    console.error('Erro ao atualizar notas e média do aluno:', error);
    res.status(500).json({ error: 'Erro ao atualizar notas e média do aluno' });
  }
});
```

Figura 11: Código Javascript com Node JS que controla as rotas do software 3

Para utilizar os recursos gerados anteriormente, é necessário criar um arquivo inicial onde o usuário configura o que será usado pela aplicação. Ao utilizar a API pela primeira vez, ocorreu um problema de permissão, resultando em um erro para o ChatGPT. Nesse caso, ele sugeriu a instalação da biblioteca CORS através do pacote NPM e configurá-la no arquivo app.js, que é o arquivo principal de configuração do Node.js:

```
const express = require('express');
const bodyParser = require('body-parser');
const turmaRoutes = require('./src/routes/turmaRoutes');
const alunoRoutes = require('./src/routes/alunoRoutes');
const cors = require('cors');

const app = express();
const port = 3000;

app.use(cors());

app.use(bodyParser.json());

app.use('/turmas', turmaRoutes);
app.use('/alunos', alunoRoutes);

app.listen(port, () => {
  console.log(`Servidor está rodando em http://localhost:${port}`);
});
```

Figura 12: Código Javascript com Node JS que configura o servidor

Com o back-end funcionando, basta adaptar o front-end para realizar as requisições na porta em que o servidor está executando. Para gerar um código mais objetivo do que precisava ser feito, foi passada a estrutura de JSON para o ChatGPT, solicitando que ele fizesse uma requisição HTTP com o método POST.

```

<script>
  // Função para calcular a média das notas
  function calcularMedia(nota1, nota2) {
    return (parseFloat(nota1) + parseFloat(nota2)) / 2;
  }

  // Função para limpar os valores dos campos
  function limparCampos() {
    $('#codigoAluno').val('');
    $('#nota1').val('');
    $('#nota2').val('');
  }

  $(document).ready(function(){
    $('#formNotas').submit(function(event){
      // Prevenir o envio do formulário padrão
      event.preventDefault();

      // Obter os valores dos campos
      var codigoAluno = $('#codigoAluno').val();
      var nota1 = $('#nota1').val();
      var nota2 = $('#nota2').val();

```

Figura 13: Código Javascript usado no front-end para integração com o back-end

```

// Validar se os campos estão preenchidos
if(codigoAluno && nota1 && nota2) {
  // Verificar se as notas estão dentro do intervalo permitido
  if(nota1 >= 0 && nota1 <= 10 && nota2 >= 0 && nota2 <= 10) {
    // Calcular a média
    var media = calcularMedia(nota1, nota2);

    // Exibir a média
    alert('A média do aluno ' + codigoAluno + ' é: ' + media.toFixed(2));

    // Limpar os valores dos campos
    limparCampos();

    // Enviar requisição POST para o servidor
    var data = {
      "nota1": parseFloat(nota1),
      "nota2": parseFloat(nota2)
    };

```

Figura 14: Código Javascript usado no front-end para integração com o back-end 2

```

fetch('http://localhost:3000/alunos/' + codigoAluno, {
  method: 'PUT',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify(data)
})
.then(response => {
  if (!response.ok) {
    throw new Error('Erro ao enviar os dados do aluno.');
```

Figura 15: Código Javascript usado no front-end para integração com o back-end 3

Os códigos, em sua maioria, funcionaram; no entanto, algumas funções às vezes são muito simples e o ChatGPT não utiliza padrões de desenvolvimento. Em alguns pontos, ele pode sugerir algo adicional para você utilizar ou solicitar mais contexto para que tenha uma resposta mais precisa.

5.2. Tabnine

Para o tabnine o primeiro prompt enviado para ele foi: “Me faça um software para avaliar um aluno”

O código sugerido por ele foi:

```

import tkinter as tk
from tkinter import messagebox

class StudentEvaluation:
    tabnine: test | explain | document | ask
    def __init__(self, root):
        self.root = root
        self.root.title("Avaliação de Aluno")

        # Crie os widgets aqui
        self.name_label = tk.Label(self.root, text="Nome do Aluno:")
        self.name_entry = tk.Entry(self.root)

        self.grade_label = tk.Label(self.root, text="Nota:")
        self.grade_entry = tk.Entry(self.root)

        self.submit_button = tk.Button(self.root, text="Enviar", command=self.submit_evaluation)

        # Posicione os widgets aqui
        self.name_label.grid(row=0, column=0)
        self.name_entry.grid(row=0, column=1)

        self.grade_label.grid(row=1, column=0)
        self.grade_entry.grid(row=1, column=1)

        self.submit_button.grid(row=2, columnspan=2)

```

Figura 16: Código Python gerado pelo Tabnine

```

tabnine: test | explain | document | ask
def submit_evaluation(self):
    name = self.name_entry.get()
    grade = self.grade_entry.get()

    if not name or not grade:
        messagebox.showerror("Erro", "Por favor, preencha todos os campos.")
        return

    try:
        grade = float(grade)
    except ValueError:
        messagebox.showerror("Erro", "Por favor, insira uma nota válida.")
        return

    if grade < 0 or grade > 10:
        messagebox.showerror("Erro", "A nota deve estar entre 0 e 10.")
        return

    # Faça algo com a avaliação aqui (salve em um banco de dados, por exemplo)
    messagebox.showinfo("Avaliação enviada", f"Avaliação de {name} com nota {grade} enviada com sucesso.")

    # Limpe os campos de entrada
    self.name_entry.delete(0, tk.END)
    self.grade_entry.delete(0, tk.END)

if __name__ == "__main__":
    root = tk.Tk()
    app = StudentEvaluation(root)
    root.mainloop()

```

Figura 17: Código Python gerado pelo Tabnine 2

Ao executar o código gerado pelo Tabnine, obtivemos uma interface bastante simples, com um design não muito funcional para o uso diário. Ao examinar o código, percebe-se que o gerador não utiliza nenhuma arquitetura ou padrão de design. Embora tenha sido sugerido o uso de um banco de dados, não houve integração implementada para isso.

Em uma segunda tentativa, explorando mais o caso de uso "Avaliar Aluno", o prompt utilizado no Tabnine foi: "Desenvolva um software para avaliar um aluno, seu desempenho teórico e sua frequência."

```
import tkinter as tk
from tkinter import messagebox

class StudentEvaluation:
    tabnine: test | explain | document | ask
    def __init__(self, root):
        self.root = root
        self.root.title("Avaliação de Aluno")

        # Crie os widgets aqui
        self.name_label = tk.Label(self.root, text="Nome do Aluno:")
        self.name_entry = tk.Entry(self.root)

        self.grade_label = tk.Label(self.root, text="Nota:")
        self.grade_entry = tk.Entry(self.root)

        self.theory_performance_label = tk.Label(self.root, text="Desempenho Teórico:")
        self.theory_performance_entry = tk.Entry(self.root)

        self.attendance_label = tk.Label(self.root, text="Frequência:")
        self.attendance_entry = tk.Entry(self.root)

        self.submit_button = tk.Button(self.root, text="Enviar", command=self.submit_evaluation)

        # Posicione os widgets aqui
        self.name_label.grid(row=0, column=0)
        self.name_entry.grid(row=0, column=1)

        self.grade_label.grid(row=1, column=0)
        self.grade_entry.grid(row=1, column=1)

        self.theory_performance_label.grid(row=2, column=0)
        self.theory_performance_entry.grid(row=2, column=1)

        self.attendance_label.grid(row=3, column=0)
        self.attendance_entry.grid(row=3, column=1)
```

Figura 18: Código Python gerado pelo Tabnine 3

```

self.submit_button.grid(row=4, columnspan=2)

tabnine test | explain | document | ask
def submit_evaluation(self):
    name = self.name_entry.get()
    grade = self.grade_entry.get()
    theory_performance = self.theory_performance_entry.get()
    attendance = self.attendance_entry.get()

    if not name or not grade or not theory_performance or not attendance:
        messagebox.showerror("Erro", "Por favor, preencha todos os campos.")
        return

    try:
        grade = float(grade)
        theory_performance = float(theory_performance)
        attendance = float(attendance)
    except ValueError:
        messagebox.showerror("Erro", "Por favor, insira valores válidos.")
        return

    if grade < 0 or grade > 10:
        messagebox.showerror("Erro", "A nota deve estar entre 0 e 10.")
        return

    if theory_performance < 0 or theory_performance > 100:
        messagebox.showerror("Erro", "O desempenho teórico deve estar entre 0 e 100.")
        return

    if attendance < 0 or attendance > 100:
        messagebox.showerror("Erro", "A frequência deve estar entre 0 e 100.")
        return

    # Faça algo com a avaliação aqui (salve em um banco de dados, por exemplo)
    messagebox.showinfo("Avaliação enviada", f"Avaliação de {name} com nota {grade}, desempenho teórico {theory_performance}% e frequência {attendance}% enviada com sucesso.")

    # Limpe os campos de entrada
    self.name_entry.delete(0, tk.END)
    self.grade_entry.delete(0, tk.END)
    self.theory_performance_entry.delete(0, tk.END)
    self.attendance_entry.delete(0, tk.END)

if __name__ == "__main__":
    root = tk.Tk()
    app = StudentEvaluation(root)
    root.mainloop()

```

Figura 19: Código Python gerado pelo Tabnine 4

The screenshot shows a window titled "Avaliação de Aluno" with a standard Windows-style title bar. Inside the window, there are four text input fields stacked vertically, each preceded by a label: "Nome do Aluno:", "Nota:", "Desempenho Teórico:", and "Frequência:". Below the "Frequência:" field is a button with the text "Enviar". The background of the window is a light gray color.

Figura 20: Primeiro software gerado pelo tabnine

Adicionando os dados de desempenho teórico e frequência, foram incluídos apenas dois campos sem qualquer lógica adicional. O programa gerado é bastante simples e não seria prático para uso diário.

Tentando descrever o caso de uso completo com o prompt: "Me faça um software para avaliar um aluno, seu desempenho teórico e a frequência onde o usuário possa informar o código da turma, recuperar a nota do aluno, calcular a média aritmética das duas notas e divulgar as notas finais dos alunos na turma."

Resultado:

```

import tkinter as tk
from tkinter import messagebox
import sqlite3

class StudentEvaluation:
    tabnine: test | explain | document | ask
    def __init__(self, root):
        self.root = root
        self.root.title("Avaliação de Aluno")

        # Crie a conexão com o banco de dados
        self.conn = sqlite3.connect("student_evaluation.db")
        self.cursor = self.conn.cursor()

        # Crie a tabela de alunos se ela ainda não existir
        self.cursor.execute("""
            CREATE TABLE IF NOT EXISTS students (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT NOT NULL,
                grade REAL,
                theory_performance REAL,
                attendance REAL,
                class_code TEXT NOT NULL
            )
        """)

        # Crie os widgets aqui
        self.class_code_label = tk.Label(self.root, text="Código da Turma:")
        self.class_code_entry = tk.Entry(self.root)

        self.name_label = tk.Label(self.root, text="Nome do Aluno:")
        self.name_entry = tk.Entry(self.root)

        self.grade_label = tk.Label(self.root, text="Nota:")
        self.grade_entry = tk.Entry(self.root)

        self.theory_performance_label = tk.Label(self.root, text="Desempenho Teórico:")
        self.theory_performance_entry = tk.Entry(self.root)

```

Figura 21: Código Python gerado pelo Tabnine que faz integração pelo SQLite

```

self.attendance_label = tk.Label(self.root, text="Frequência:")
self.attendance_entry = tk.Entry(self.root)

self.submit_button = tk.Button(self.root, text="Enviar", command=self.submit_evaluation)
self.retrieve_button = tk.Button(self.root, text="Recuperar Nota", command=self.retrieve_grade)
self.calculate_button = tk.Button(self.root, text="Calcular Média", command=self.calculate_average)
self.publish_button = tk.Button(self.root, text="Divulgar Notas", command=self.publish_grades)

# Posicione os widgets aqui
self.class_code_label.grid(row=0, column=0)
self.class_code_entry.grid(row=0, column=1)

self.name_label.grid(row=1, column=0)
self.name_entry.grid(row=1, column=1)

self.grade_label.grid(row=2, column=0)
self.grade_entry.grid(row=2, column=1)

self.theory_performance_label.grid(row=3, column=0)
self.theory_performance_entry.grid(row=3, column=1)

self.attendance_label.grid(row=4, column=0)
self.attendance_entry.grid(row=4, column=1)

self.submit_button.grid(row=5, columnspan=2)
self.retrieve_button.grid(row=6, columnspan=2)
self.calculate_button.grid(row=7, columnspan=2)
self.publish_button.grid(row=8, columnspan=2)

```

Figura 22: Código Python gerado pelo Tabnine que gera interface gráfica do sistemas

```

tabnine: test | explain | document | ask
def submit_evaluation(self):
    class_code = self.class_code_entry.get()
    name = self.name_entry.get()
    grade = self.grade_entry.get()
    theory_performance = self.theory_performance_entry.get()
    attendance = self.attendance_entry.get()

    if not class_code or not name or not grade or not theory_performance or not attendance:
        messagebox.showerror("Erro", "Por favor, preencha todos os campos.")
        return

    try:
        grade = float(grade)
        theory_performance = float(theory_performance)
        attendance = float(attendance)
    except ValueError:
        messagebox.showerror("Erro", "Por favor, insira valores válidos.")
        return

    if grade < 0 or grade > 10:
        messagebox.showerror("Erro", "A nota deve estar entre 0 e 10.")
        return

    if theory_performance < 0 or theory_performance > 100:
        messagebox.showerror("Erro", "O desempenho teórico deve estar entre 0 e 100.")
        return

    if attendance < 0 or attendance > 100:
        messagebox.showerror("Erro", "A frequência deve estar entre 0 e 100.")
        return

    # Insira os dados do aluno no banco de dados
    self.cursor.execute("""
        INSERT INTO students (name, grade, theory_performance, attendance, class_code)
        VALUES (?, ?, ?, ?, ?)
    """, (name, grade, theory_performance, attendance, class_code))
    self.conn.commit()

```

Figura 23: Código Python gerado pelo Tabnine que faz integração com o banco e gera messagebox

```

messagebox.showinfo("Avaliação enviada", f"Avaliação de {name} enviada com sucesso.")

# Limpe os campos de entrada
self.name_entry.delete(0, tk.END)
self.grade_entry.delete(0, tk.END)
self.theory_performance_entry.delete(0, tk.END)
self.attendance_entry.delete(0, tk.END)

tabnine: test | explain | document | ask
def retrieve_grade(self):
    class_code = self.class_code_entry.get()
    name = self.name_entry.get()

    if not class_code or not name:
        messagebox.showerror("Erro", "Por favor, informe o código da turma e o nome do aluno.")
        return

    # Recupera a nota do aluno do banco de dados
    self.cursor.execute("""
        SELECT grade FROM students
        WHERE name =? AND class_code =?
    """, (name, class_code))
    result = self.cursor.fetchone()

    if result is None:
        messagebox.showinfo("Nota", "Não foi encontrada nenhuma avaliação para este aluno.")
    else:
        grade = result[0]
        messagebox.showinfo("Nota", f"A nota do aluno {name} é {grade}.")

```

Figura 24: Código Python gerado pelo Tabnine que faz integração com o banco e gera messagebox

```

tabnine: test | explain | document | ask
def calculate_average(self):
    class_code = self.class_code_entry.get()

    if not class_code:
        messagebox.showerror("Erro", "Por favor, informe o código da turma.")
        return

    # Calcula a média aritmética das notas dos alunos da turma
    self.cursor.execute("""
        SELECT AVG((grade + theory_performance) / 2) FROM students
        WHERE class_code =?
    """, (class_code,))
    result = self.cursor.fetchone()

    if result is None:
        messagebox.showinfo("Média", "Não foi encontrada nenhuma avaliação para esta turma.")
    else:
        average = result[0]
        messagebox.showinfo("Média", f"A média aritmética das notas da turma {class_code} é {average:.2f}.")

```

Figura 25: Código gerado pelo tabnine que faz os cálculos das médias

```

tabnine: test | explain | document | ask
def publish_grades(self):
    class_code = self.class_code_entry.get()

    if not class_code:
        messagebox.showerror("Erro", "Por favor, informe o código da turma.")
        return

    # Recupera as notas e nomes dos alunos da turma
    self.cursor.execute("""
        SELECT name, grade, theory_performance, attendance FROM students
        WHERE class_code =?
        """, (class_code,))
    results = self.cursor.fetchall()

    if len(results) == 0:
        messagebox.showinfo("Notas", "Não foi encontrada nenhuma avaliação para esta turma.")
    else:
        grades_text = "Notas da Turma " + class_code + ":\n\n"
        for result in results:
            name, grade, theory_performance, attendance = result
            grades_text += f"{name}: Nota {grade}, Desempenho Teórico {theory_performance}, Frequência {attendance}\n"

        messagebox.showinfo("Notas", grades_text)

if __name__ == "__main__":
    root = tk.Tk()
    app = StudentEvaluation(root)
    root.mainloop()

```

Figura 26: Código gerado pelo tabnine que busca as notas dos alunos

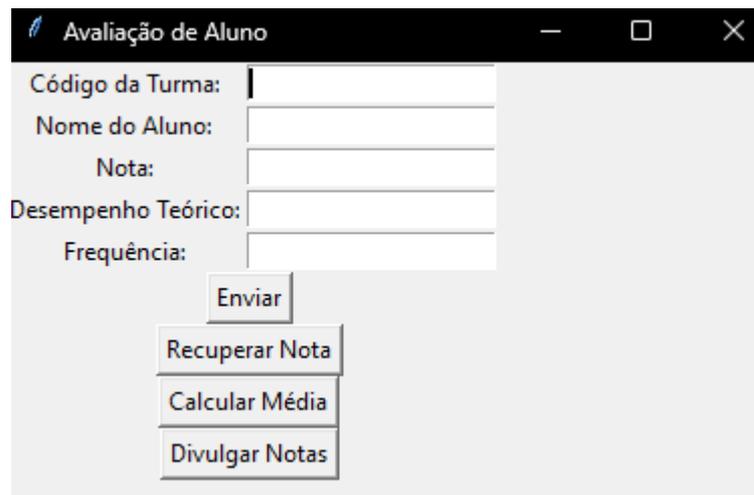


Figura 27: Interface gerada pelo tabine

Botão recuperar nota:

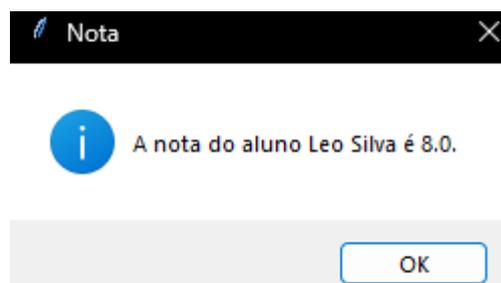


Figura 28: MessageBox gerada pelo tabnine

Botão Calcular Média:

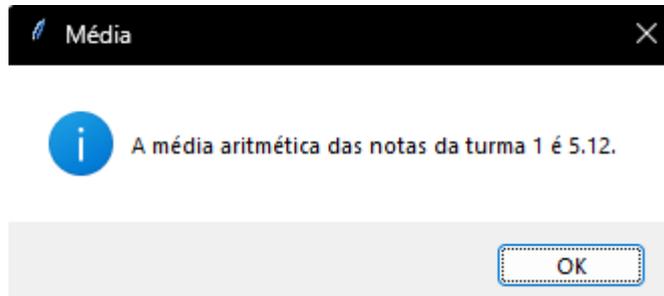


Figura 29: MessageBox gerada pelo tabnine 2

Botão Divulgar Notas:

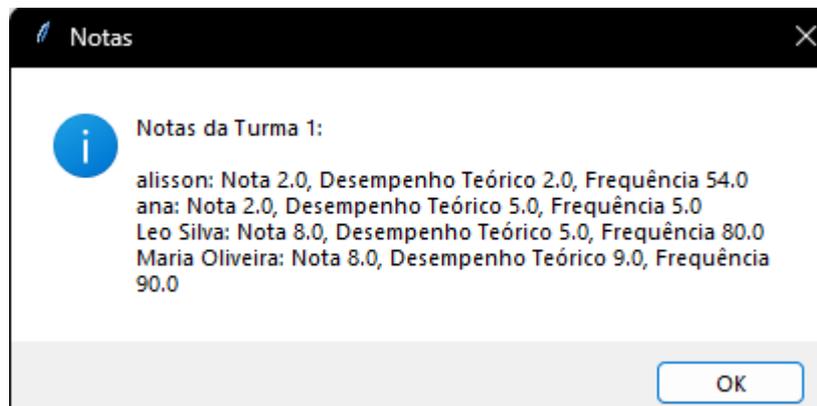


Figura 30: MessageBox gerada pelo tabnine 3

O programa ainda permanece bastante simples; no entanto, foi integrado com o SQLite e o código não apresentou problemas durante a execução. Contudo, o programa continua sendo inviável para uso diário pelos usuários. O Tabnine demonstra incapacidade de criar um software funcional apenas com base nas descrições dos casos de uso. No entanto, mostra-se eficiente na criação de pequenos blocos de código individualmente. Quando são solicitados processos específicos, pedindo para ele seguir uma determinada arquitetura ou usar algum framework, ele consegue atender. No entanto, ao lidar apenas com descrições que podem ser interpretadas de diversas maneiras, o resultado é insatisfatório.

Ao pedir individualmente para ele criar uma interface gráfica integrada com uma API, foi obtido um resultado simples, uma aplicação que não seria útil para o usuário final. Foi solicitado um front-end com HTML e Bootstrap e um back-end com Node.js:Front-end

Avaliação de Aluno

Cadastrar Aluno	Notas Finais da Turma
<p>Código da Turma</p> <input type="text"/>	<p>Turma: 1</p>
<p>Matrícula do Aluno</p> <input type="text"/>	<p>Nome: Alisson</p>
<p>Nome</p> <input type="text"/>	<p>Matrícula: 23</p>
<p>Nota Teórica</p> <input type="text"/>	<p>Nota Final: 6.5</p>
<p>Nota Prática</p> <input type="text"/>	
<p>Frequência</p> <input type="text"/>	
<p><input type="button" value="Cadastrar"/></p>	

Figura 31: Interface gerada pelo tabnine com HTML e BootStrap

Back-end:

```

<!DOCTYPE html>
<html>
<head>
  <title>Avaliação de Aluno</title>
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
  <div class="container">
    <h1 class="text-center my-4">Avaliação de Aluno</h1>

    <div class="row">
      <div class="col-md-6">
        <div class="card">
          <div class="card-header">
            Cadastrar Aluno
          </div>
          <div class="card-body">
            <form id="register-student-form">
              <div class="form-group">
                <label for="turma">Código da Turma</label>
                <input type="text" class="form-control" id="turma" required>
              </div>
              <div class="form-group">
                <label for="matricula">Matricula do Aluno</label>
                <input type="text" class="form-control" id="matricula" required>
              </div>
              <div class="form-group">
                <label for="nome">Nome</label>
                <input type="text" class="form-control" id="nome" required>
              </div>
              <div class="form-group">
                <label for="nota-teorica">Nota Teórica</label>
                <input type="number" class="form-control" id="nota-teorica" required>
              </div>
            </form>
          </div>
        </div>
      </div>
    </div>
  </div>

```

Figura 32: Código front-end Gerado pelo tabnine com HTML e Bootstrap

```

              <div class="form-group">
                <label for="nota-pratica">Nota Prática</label>
                <input type="number" class="form-control" id="nota-pratica" required>
              </div>
              <div class="form-group">
                <label for="frequencia">Frequência</label>
                <input type="number" class="form-control" id="frequencia" required>
              </div>
              <button type="submit" class="btn btn-primary">Cadastrar</button>
            </form>
          </div>
        </div>
      </div>
    </div>

    <div class="col-md-6">
      <div class="card">
        <div class="card-header">
          Notas Finais da Turma
        </div>
        <div class="card-body" id="class-grades">
          <!-- Notas finais da turma serão exibidas aqui -->
        </div>
      </div>
    </div>
  </div>
</div>

```

Figura 32: Código front-end Gerado pelo tabnine com HTML e Bootstrap 2

```

</script>
// Atualize a função JavaScript para cadastrar um novo aluno
document.getElementById('register-student-form').addEventListener('submit', function(event) {
  event.preventDefault();

  const turma = document.getElementById('turma').value;
  const matricula = document.getElementById('matricula').value;
  const nome = document.getElementById('nome').value;
  const nota_teorica = parseFloat(document.getElementById('nota-teorica').value);
  const nota_pratica = parseFloat(document.getElementById('nota-pratica').value);
  const frequencia = parseFloat(document.getElementById('frequencia').value);

  const data = { turma, matricula, nome, nota_teorica, nota_pratica, frequencia };

  fetch('http://localhost:3000/students', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(data)
  })
  .then(response => response.json())
  .then(data => {
    alert(data.message);
    document.getElementById('register-student-form').reset();
    fetchClassGrades(); // Atualize as notas finais da turma após cadastrar um novo aluno
  })
  .catch(error => {
    console.error('Error:', error);
    alert('Ocorreu um erro ao cadastrar o aluno.');
```

Figura 33: Código Javascript no front-end Gerado pelo tabnine

```

function fetchClassGrades() {
  fetch('http://localhost:3000/studentsopa')
  .then(response => response.json())
  .then(data => {
    const classGradesElement = document.getElementById('class-grades');
    let gradesHTML = '';

    data.forEach(student => {
      gradesHTML += `
        <div>
          <p><strong>Turma:</strong> ${student.turma}</p>
          <p><strong>Nome:</strong> ${student.nome}</p>
          <p><strong>Matrícula:</strong> ${student.matricula}</p>
          <p><strong>Nota Final:</strong> ${student.nota_final}</p>
        </div>
      `;
    });

    classGradesElement.innerHTML = gradesHTML;
  })
  .catch(error => {
    console.error('Error:', error);
    alert('Ocorreu um erro ao recuperar as notas finais da turma.');
```

Figura 34: Código Javascript no front-end Gerado pelo tabnine 2

Especificando alguns pontos dentro do gerador, obtivemos alguns resultados melhores, porém ocorreram alguns erros. Primeiramente, quando foi pedido para o gerador migrar a aplicação, ele esqueceu algumas informações como o nome do usuário e criou apenas uma nota única. Também esqueceu do cadastro do aluno, criando uma aplicação funcional inicialmente, mas sem métodos para alimentar a base de dados do sistema.

Posteriormente, ao solicitar individualmente, o gerador desenvolveu os métodos necessários. O mesmo aconteceu com o front-end, onde inicialmente faltavam alguns itens, mas após pedido específico, o gerador criou os campos necessários.

O Tabnine se mostra muito dependente do usuário que o utiliza. Se o usuário não especificar detalhadamente o que precisa, ele tende a gerar programas simples.

5.3. Codeium

Partindo do Caso de Uso no Codeium, o resultado inicial é algo bastante simples que não seria prático para uso diário. O prompt enviado para ele foi: "Desenvolva um software para avaliar um aluno."

Resposta:

```
Codeium: Refactor | Explain
class Aluno:
    Codeium: Refactor | Explain | Generate Docstring | X
    def __init__(self, nome, notas):
        self.nome = nome
        self.notas = notas

    Codeium: Refactor | Explain | Generate Docstring | X
    def calcular_media(self):
        return sum(self.notas) / len(self.notas)

    Codeium: Refactor | Explain | Generate Docstring | X
    def avaliar(self):
        media = self.calcular_media()
        if media >= 7:
            return f"{self.nome} foi aprovado com média {media}!"
        elif media >= 5:
            return f"{self.nome} está de recuperação com média {media}!"
        else:
            return f"{self.nome} reprovado com média {media}!"

# Exemplo de uso
aluno = Aluno("João", [8, 7, 6, 9])
print(aluno.avaliar())
```

Figura 35: Código Python gerado pelo Codeium

O código gerado funciona de forma estática e não há integração com banco de dados nem criação de interface.

Aprofundando-se mais no caso de uso para tentar obter um software melhor, utilizou-se o seguinte prompt: "Desenvolva um software para avaliar um aluno, seu desempenho teórico e a frequência."

Resultado:

```
Codeium: Refactor | Explain
class Aluno:
Codeium: Refactor | Explain | Generate Docstring | X
def __init__(self, nome, notas_teorica, desempenho_teorica, frequencia):
    self.nome = nome
    self.notas_teorica = notas_teorica
    self.desempenho_teorica = desempenho_teorica
    self.frequencia = frequencia

Codeium: Refactor | Explain | Generate Docstring | X
def calcular_media_teorica(self):
    return sum(self.notas_teorica) / len(self.notas_teorica)

Codeium: Refactor | Explain | Generate Docstring | X
def avaliar(self):
    media_teorica = self.calcular_media_teorica()

    if media_teorica >= 7 and self.desempenho_teorica >= 8 and self.frequencia >= 75:
        return f"{self.nome} foi aprovado com média teórica {media_teorica}, desempenho teórico {self.desempenho_teorica} e frequência {self.frequencia}%"
    else:
        return f"{self.nome} foi reprovado. Média teórica: {media_teorica}, Desempenho teórico: {self.desempenho_teorica}, Frequência: {self.frequencia}%"

# Exemplo de uso
aluno = Aluno("Maria", [8, 7, 6, 9], 8, 80)
print(aluno.avaliar())
```

Figura 36: Código Python gerado pelo Codeium 2

O código continua sendo estático, sem o uso de arquiteturas ou padrões de desenvolvimento, e para o dia a dia não seria eficiente. De certa forma, ele devolve o que foi pedido; porém, como é um software onde não podemos nem digitar as notas, ele não gerou uma interface gráfica.

Seguindo com o caso de uso, foi enviado o seguinte prompt: "Desenvolva um software para avaliar um aluno, seu desempenho teórico e a frequência, onde o usuário possa informar o código da turma, recuperar a nota do aluno, calcular a média aritmética das duas notas e divulgar as notas finais dos alunos na turma."Resultado:

```

class Aluno:
    Codeium: Refactor | Explain | Generate Docstring | X
    def __init__(self, nome, notas_teorica):
        self.nome = nome
        self.notas_teorica = notas_teorica
        self.nota_final = None

    Codeium: Refactor | Explain | Generate Docstring | X
    def calcular_media_teorica(self):
        return sum(self.notas_teorica) / len(self.notas_teorica)

    Codeium: Refactor | Explain | Generate Docstring | X
    def set_nota_final(self, nota_pratica):
        self.nota_final = nota_pratica

    Codeium: Refactor | Explain | Generate Docstring | X
    def get_nota_final(self):
        return self.nota_final

```

Figura 37: Código Python gerado pelo Codeium que salva as notas em memória

```

class Turma:
    Codeium: Refactor | Explain | Generate Docstring | X
    def __init__(self, codigo, alunos):
        self.codigo = codigo
        self.alunos = alunos

    Codeium: Refactor | Explain | Generate Docstring | X
    def recuperar_nota(self, nome):
        for aluno in self.alunos:
            if aluno.nome == nome:
                return aluno.notas_teorica

    Codeium: Refactor | Explain | Generate Docstring | X
    def calcular_media_turma(self):
        notas_turmas = [aluno.calcular_media_teorica() for aluno in self.alunos]
        return sum(notas_turmas) / len(notas_turmas)

    Codeium: Refactor | Explain | Generate Docstring | X
    def divulgar_notas_finais(self):
        for aluno in self.alunos:
            nota_teorica = aluno.calcular_media_teorica()
            nota_pratica = aluno.get_nota_final()
            nota_final = (nota_teorica + nota_pratica) / 2
            print(f"{aluno.nome}: Nota Teórica: {nota_teorica}, Nota Prática: {nota_pratica}, Nota Final: {nota_final}")

```

Figura 38: Código Python gerado pelo Codeium que calcula média e recupera as notas

```

# Exemplo de uso
turma = Turma("A1", [])

while True:
    opcao = input("Digite a opção desejada:\n"
                 "1 - Cadastrar aluno\n"
                 "2 - Recuperar nota\n"
                 "3 - Calcular média da turma\n"
                 "4 - Divulgar notas finais\n"
                 "5 - Sair\n")
    if opcao == "1":
        nome = input("Digite o nome do aluno: ")
        notas_teorica = [float(nota) for nota in input("Digite as notas teóricas separadas por vírgula: ").split(",")]
        aluno = Aluno(nome, notas_teorica)
        turma.alunos.append(aluno)
        print("Aluno cadastrado com sucesso!")
    elif opcao == "2":
        nome = input("Digite o nome do aluno: ")
        notas = turma.recuperar_notas(nome)
        if notas:
            print(f"Notas teóricas de {nome}: {notas}")
        else:
            print(f"Aluno {nome} não encontrado na turma.")
    elif opcao == "3":
        media_turma = turma.calcular_media_turma()
        print(f"Média da turma: {media_turma}")
    elif opcao == "4":
        turma.divulgar_notas_finais()
    elif opcao == "5":
        break
    else:
        print("Opção inválida. Tente novamente para continuar.")

```

Figura 39: Código Python gerado pelo Codeium

O Codeium, especificando o caso de uso inteiro, teve um resultado bem insatisfatório para o uso diário. O software gerado continua sem interface gráfica e sem integração com banco de dados, o que o torna inviável para o usuário comum.

Assim como os demais geradores de código, o usuário que for utilizar o Codeium precisa ter o software bem definido e saber o que deseja ter como sistema e suas tecnologias, pois ele precisa definir coisas como arquitetura e bibliotecas que o software vai usar. Se o prompt estiver muito aberto a interpretação, o gerador devolverá resultados mais básicos ou pedirá mais contexto sobre a aplicação.

Pedindo uma interface feita com HTML usando o framework Bootstrap, o resultado é este:

Avaliação de Alunos

Nome do Aluno:

Nota 1:

Nota 2:

Código da Turma:

Figura 40: Interface com HTML e bootstrap gerado pelo Codeium

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Avaliação de Alunos</title>
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
  <div class="container">
    <h1>Avaliação de Alunos</h1>
    <form>
      <div class="form-group">
        <label for="nome">Nome do Aluno:</label>
        <input type="text" class="form-control" id="nome">
      </div>
      <div class="form-group">
        <label for="nota1">Nota 1:</label>
        <input type="number" class="form-control" id="nota1">
      </div>
      <div class="form-group">
        <label for="nota2">Nota 2:</label>
        <input type="number" class="form-control" id="nota2">
      </div>
      <button type="button" class="btn btn-primary" onclick="salvarAluno()">Salvar Aluno</button>

      <!-- Botão para recuperar todos os alunos da turma -->
      <button type="button" class="btn btn-primary" onclick="recuperarAlunos()">Recuperar Alunos</button>

      <!-- Campo de input para inserir o código da turma -->
      <div class="form-group">
        <label for="codigoTurma">Código da Turma:</label>
        <input type="text" class="form-control" id="codigoTurma">
      </div>
      <button type="button" class="btn btn-primary" onclick="recuperarAluno()">Recuperar Aluno</button>
    </form>
  </div>

```

Figura 41: Código da interface gerado pelo Codeium com HTML e Bootstrap

```
</form>
<div class="mt-3" id="resultado"></div>
</div>
<script>
Codeium: Refactor | Explain | Generate Function Comment | X
function salvarAluno() {
  let codigoTurma = document.getElementById('codigoTurma').value;
  let nome = document.getElementById('nome').value;
  let nota1 = document.getElementById('nota1').value;
  let nota2 = document.getElementById('nota2').value;

  fetch('http://localhost:3000/salvar-aluno', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      codigoTurma: codigoTurma,
      nome: nome,
      nota1: nota1,
      nota2: nota2,
    })
  })
  .then(response => {
    if (!response.ok) {
      throw new Error('Erro ao salvar o aluno.');
```

Figura 42: Código da interface gerado pelo Codeium que faz integração com Javascript com o back-end

```

        .then(data => {
            console.log(data);
        })
        .catch(error => {
            console.error('Erro:', error);
        });
    }
}

Codeium: Refactor | Explain | Generate Function Comment | X
function recuperarAlunos() {
    let codigoTurma = document.getElementById('codigoTurma').value;

    fetch(`http://localhost:3000/recuperar-alunos/${codigoTurma}`)
        .then(response => {
            if (!response.ok) {
                throw new Error('Erro ao recuperar os alunos.');
            }
            return response.json();
        })
        .then(data => {
            console.log(data);
        })
        .catch(error => {
            console.error('Erro:', error);
        });
}

```

Figura 43: Código da interface gerado pelo Codeium que faz integração com Javascript com o back-end 2

```

Codeium: Refactor | Explain | Generate Function Comment | X
function recuperarAluno() {
    let id = document.getElementById('idAluno').value;

    fetch(`http://localhost:3000/recuperar-aluno/${id}`)
        .then(response => {
            if (!response.ok) {
                throw new Error('Erro ao recuperar o aluno.');
            }
            return response.json();
        })
        .then(data => {
            console.log(data);
        })
        .catch(error => {
            console.error('Erro:', error);
        });
}
</script>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.3/dist/umd/popper.min.js"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>

```

Figura 44: Código da interface gerado pelo Codeium que faz integração com Javascript com o back-end 3

Nota-se que ele faz o que foi pedido de forma simples. Ele não sugeriu nenhum framework como Angular ou React para desenvolvimento, mas conseguiu fazer a integração com a API que foi solicitada. Em um primeiro momento, ele fez toda a lógica no JavaScript do próprio HTML. Depois, foi solicitado que ele consumisse a API feita no projeto e, desta forma, ele fez a integração da API.

O back-end, que foi feito em Python, foi migrado para JavaScript para facilitar a integração das duas partes. Como foi solicitada uma API com base no front-end, o resultado foi este:

```
const express = require('express');
const cors = require('cors');
const sqlite3 = require('sqlite3').verbose();

const app = express();
const port = 3000;

app.use(express.json());
app.use(cors());

const db = new sqlite3.Database('alunos.db');

db.serialize(() => {
  db.run("DROP TABLE IF EXISTS alunos"); // Drop da tabela existente
  db.run("CREATE TABLE IF NOT EXISTS alunos (id INTEGER PRIMARY KEY AUTOINCREMENT, codigoTurma TEXT, nome TEXT, nota1 REAL, nota2 REAL)");
});

app.post('/salvar-aluno', (req, res) => {
  const { codigoTurma, nome, nota1, nota2 } = req.body;

  db.run("INSERT INTO alunos (codigoTurma, nome, nota1, nota2) VALUES (?, ?, ?, ?)", [codigoTurma, nome, nota1, nota2], function(err) {
    if (err) {
      return res.status(500).send('Erro ao salvar o aluno no banco de dados.');
    }

    res.send('Aluno salvo com sucesso.');
  });
});
```

Figura 45: Código Back-end gerado com Node JS pelo Codeium que a faz integração com banco de dados

```
app.get('/recuperar-alunos/:codigoTurma', (req, res) => {
  const codigoTurma = req.params.codigoTurma;

  db.all("SELECT * FROM alunos WHERE codigoTurma = ?", [codigoTurma], (err, rows) => {
    if (err) {
      return res.status(500).send('Erro ao recuperar os alunos da turma.');
```

```
    }

    res.json(rows);
  });
});

app.get('/recuperar-aluno/:id', (req, res) => {
  const id = req.params.id;

  db.get("SELECT * FROM alunos WHERE id = ?", [id], (err, row) => {
    if (err) {
      return res.status(500).send('Erro ao recuperar o aluno.');
```

```
    }

    if (!row) {
      return res.status(404).send('Aluno não encontrado.');
```

```
    }

    res.json(row);
  });
});

app.listen(port, () => {
  console.log(`Servidor rodando em http://localhost:${port}`);
});
```

Figura 46: Código Back-end gerado com Node Js que faz busca no banco de dados

Neste exemplo foi gerada uma API funcional; porém, ele gerou um código estruturado onde toda a configuração de servidor, rotas, controladores e persistência de dados foi feita em um único arquivo, o que não é uma boa prática de desenvolvimento de código.

6. CONCLUSÕES

Com os testes realizados, é possível concluir que os geradores de código não são capazes de partir apenas dos casos de uso para desenvolver um software completo e funcional para o dia a dia. Em todos os geradores testados, o resultado foi um software minimalista que não seria viável para uso prático.

No entanto, mesmo não entregando um software completo apenas a partir do caso de uso, os geradores de código são ótimas ferramentas para desenvolvimento de recursos isolados. Se o usuário que estiver desenvolvendo tiver um bom conhecimento de programação e uma definição clara do que deseja desenvolver, ele conseguirá ser mais produtivo. Ainda assim, mesmo especificando bem o que precisa, há chances de erros por parte do gerador de código.

Portanto, o gerador de código é mais útil como uma ferramenta de auxílio ao desenvolvimento do que como um gerador de código de ponta a ponta. No estado atual dos geradores, a sua utilização para desenvolvimento completo pode gerar muitos problemas e não se mostra eficiente para tais situações.

7. TRABALHOS FUTUROS

Como trabalho futuro, recomenda-se a realização de uma análise algorítmica mais detalhada para avaliar a consistência do código gerado pelas inteligências artificiais, além do estudo de tradução entre linguagens. Isso envolve solicitar que o gerador de código traduza um código em linguagem C para Java e avaliar a compatibilidade da tradução gerada. Com base nesses códigos e nas comparações para identificar qual foi o mais consistente, seria relevante investigar, se possível, as razões dessa diferença. Além disso, sugere-se estudos mais aprofundados sobre *deep learning*.

Para agregar mais valor ao relatório, recomenda-se a realização de testes com um caso de uso mais complexo, que exija uma interface mais elaborada, verificando se o design UX está sendo bem aplicado, assim como a criação de código de *back-end* e uma estrutura de banco de dados.

8. AGRADECIMENTOS

Agradeço à FATEC e ao Centro Paula Souza pela estrutura durante toda minha formação, à Prof.^a Dr.^a Marcia Ito pelo convite para a realização da pesquisa com sua orientação.

9. REFERÊNCIAS BIBLIOGRÁFICAS

INTELIGÊNCIA artificial. *In*: GUITARRA, Paloma. **Inteligência artificial**. 25 jul. 2023. Disponível em: <https://brasilecola.uol.com.br/informatica/inteligencia-artificial.htm>. Acesso em: 17 jun. 2024.

10 melhores geradores de código de IA (junho de 2024). 2 jun. 2024. Disponível em: <https://www.unite.ai/pt/best-ai-code-generators/>. Acesso em: 17 jun. 2024.

ARTIFICIAL Intelligence: in-depth market analysis. **Artificial Intelligence: in-depth market analysis**, [S. l.], p. 133, 3 abr. 2024. Disponível em: <https://www.statista.com/study/50485/in-depth-report-artificial-intelligence/>. Acesso em: 17 jun. 2024.