**CENTRO UNIVERSITÁRIO CATÓLICA DE SANTA CATARINA**

**ENGENHARIA DE SOFTWARE**

**RICHARD SCHMITZ RIEDO**

**Gerador de código fonte de classes**

**Joinville**

**2025**

**Resumo**

Este documento apresenta os fundamentos técnicos e conceituais para o desenvolvimento de uma aplicação web inteligente que automatiza a geração de código-fonte Java a partir de diagramas UML de classes. A proposta tem como objetivo otimizar o processo de desenvolvimento orientado a objetos por meio da utilização de técnicas de reconhecimento de imagem, OCR (Tesseract), visão computacional (OpenCV) e inteligência artificial.

A ferramenta permitirá que usuários façam upload de diagramas UML nos formatos .png, .jpg ou .svg, e a partir da identificação de classes, atributos e métodos, o sistema será capaz de gerar automaticamente o código Java correspondente. O projeto é dividido em módulos frontend (Vue.js + Quasar), backend (Node.js + Express), e módulos de IA (em Python), com banco de dados PostgreSQL.

**Introdução**

O desenvolvimento de software orientado a objetos é amplamente utilizado para estruturação e organização de sistemas complexos, garantindo reutilização de código e modularidade. Dentro desse contexto, os **diagramas UML (Unified Modeling Language)** desempenham um papel fundamental na modelagem e documentação de sistemas, permitindo que desenvolvedores visualizem classes, atributos, métodos e relacionamentos antes da implementação do código. Porém, a criação manual desses diagramas para códigos pode ser considerado um trabalho extenso e propenso a falhas, impactando tanto a produtividade quanto a qualidade do software.

Para resolver esse problema, este projeto de portifólio propõe o desenvolvimento de uma ferramenta baseada em **Inteligência Artificial (IA)** capaz de interpretar diagramas UML e gerar automaticamente código em **Java**. Utilizando técnicas de **reconhecimento de imagem, processamento de texto e aprendizado de máquina**, o sistema analisará diagramas de classe fornecidos pelos usuários, identificará suas estruturas e produzirá o código correspondente. Dessa forma, busca-se **automatizar esse processo**, reduzindo o tempo de desenvolvimento e minimizando falhas no desenvolvimento manual.

A automação desse processo por meio de **Inteligência Artificial** representa um avanço significativo no **mundo da engenharia de software**, pois reduz o esforço manual, aumenta a produtividade e minimiza inconsistências na transcrição dos modelos. Além disso, uma ferramenta que gera código automaticamente a partir de diagramas UML pode beneficiar tanto **profissionais** quanto **estudantes**, facilitando o aprendizado e a aplicação de conceitos da programação orientada a objetos.

Dessa forma, este projeto contribui para a engenharia de software ao **automatizar uma etapa fundamental do desenvolvimento** com a otimização do fluxo de trabalho e promovendo maior precisão na implementação de sistemas. A adoção dessa tecnologia pode **agilizar e tornar mais confiável o desenvolvimento de software**, impactando diretamente a qualidade e a eficiência dos projetos.

**Descrição do Projeto**

O projeto propõe o desenvolvimento de uma ferramenta baseada em **Inteligência Artificial** para a conversão automatizada de **diagramas UML de classes** em código-fonte **Java**. A ferramenta permitirá que os usuários façam upload de diagramas UML e, por meio de **reconhecimento de imagem e processamento de texto**, o sistema identificará classes, atributos e métodos, gerando automaticamente código estruturado. O objetivo é otimizar o fluxo de trabalho no desenvolvimento de software, reduzindo o tempo gasto na transcrição manual de diagramas e minimizando erros de implementação.

**Problemas a Resolver**

Este projeto visa solucionar desafios comuns na conversão de diagramas UML para código-fonte, incluindo:

* **Redução do tempo de desenvolvimento**: A conversão manual de diagramas UML para código exige esforço e pode atrasar o desenvolvimento de software.
* **Minimização de erros de transcrição**: Durante a conversão manual, informações podem ser interpretadas incorretamente, levando a falhas no código.
* **Aprimoramento da produtividade**: Desenvolvedores podem focar mais na lógica do software e menos na transcrição de modelos UML.
* **Facilitação do aprendizado**: A ferramenta pode auxiliar estudantes e iniciantes na programação orientada a objetos ao fornecer código gerado automaticamente a partir dos diagramas UML.

**Limitações**

Algumas limitações serão estabelecidas para manter o escopo do projeto viável:

* **Suporte exclusivo para diagramas UML de classes**: O sistema não interpretará outros tipos de diagramas UML, como diagramas de sequência ou de casos de uso.
* **Geração de código apenas em Java**: O projeto será focado exclusivamente na conversão para a linguagem Java, sem suporte inicial para outras linguagens.
* **Reconhecimento limitado a diagramas bem estruturados**: O desempenho da IA pode ser comprometido caso os diagramas tenham baixa qualidade visual ou má organização.
* **Não inclui edição de diagramas**: A ferramenta apenas interpretará diagramas prontos, sem funcionalidade para desenhá-los ou editá-los dentro da plataforma.

**Especificação Técnica**

A especificação técnica é uma parte essencial no desenvolvimento de sistemas, pois fornece uma descrição detalhada dos aspectos operacionais e estruturais de um projeto. O principal objetivo é assegurar que todos os envolvidos no processo de desenvolvimento tenham uma compreensão clara dos requisitos, das tecnologias que serão utilizadas, dos fluxos de trabalho e das decisões de design que guiarão a implementação da solução proposta.

**Requisitos do Tema Proposto**

O sistema deverá contemplar os seguintes requisitos:

**Protocolos:**

* **HTTP/HTTPS**: Comunicação entre cliente e servidor.
* **RESTful API**: Estrutura de requisições entre frontend e backend.
* **JWT (JSON Web Token)**: Autenticação segura para acesso de usuários autenticados.

**Algoritmos e Procedimentos:**

* **OCR (Tesseract)**: Reconhecimento óptico de caracteres nos diagramas UML.
* **OpenCV**: Detecção e segmentação de caixas, setas e relações visuais entre classes.
* **IA treinada com exemplos de diagramas UML reais**: Para melhoria da acurácia e entendimento de estruturas complexas.
* **Conversão para código Java**: Mapeamento dos elementos identificados para estruturas da linguagem Java (classes, atributos, métodos, relacionamentos).

**Procedimentos:**

1. Usuário cria conta e realiza login ou efetua login caso já tenha uma conta.
2. Usuário cria um projeto.
3. Usuário envia uma imagem do diagrama UML.
4. O backend processa a imagem e extrai as informações usando OCR + IA.
5. O sistema converte os dados em código Java.
6. O código será exportado como ZIP.

**Formatos de Dados:**

* **Imagem de entrada**: .png, .jpg, .svg.
* **Código gerado**: arquivos .java.
* **Exportação final**: arquivo .zip com a estrutura de pastas e classes organizadas.
* **Dados de projeto e usuário**: JSON em comunicação entre frontend e backend.
* **Banco de dados relacional (PostgreSQL)**: para armazenar usuários, projetos e logs.

**Requisitos Funcionais (RF)**

|  |  |
| --- | --- |
| **Código** | **Requisito Funcional** |
| RF01 | O sistema deve permitir que o usuário se cadastre com e-mail e senha. |
| RF02 | O sistema deve permitir que o usuário faça login com suas credenciais. |
| RF03 | O sistema deve permitir que o usuário altere sua senha. |
| RF04 | O sistema deve permitir que o usuário exclua sua conta |
| RF05 | O sistema deve permitir que o usuário crie um projeto antes de enviar um diagrama UML. |
| RF06 | O sistema deve permitir o upload de diagramas UML de classes nos formatos PNG, JPG e SVG. |
| RF07 | O sistema deve processar o diagrama e identificar classes, atributos e métodos. |
| RF08 | O sistema deve gerar código-fonte em Java com base nas informações extraídas. |
| RF09 | O sistema deve permitir que o usuário visualize e edite o código gerado. |
| RF10 | O sistema deve permitir a exportação do código em um arquivo ZIP. |
| RF11 | O sistema deve listar os projetos anteriores criados pelo usuário. |
| RF12 | O sistema deve permitir a exclusão dos projetos criados pelo usuário. |
| RF13 | O sistema deve garantir a exclusão automática dos arquivos após o processamento. |

**Requisitos Não-Funcionais (RNF)**

|  |  |
| --- | --- |
| **Código** | **Requisito Não-Funcional** |
| RNF01 | O sistema deve ter uma interface web responsiva |
| RNF02 | O tempo de processamento dos diagramas deve ser inferior a 1 minuto. |
| RNF03 | As informações do usuário devem ser armazenadas de forma segura. |

**Discussão sobre as Escolhas de Design**

O projeto adota uma abordagem modular baseada em uma **arquitetura cliente-servidor**, visando escalabilidade, separação de responsabilidades e facilidade de manutenção. A decisão por dividir o sistema em **frontend, backend e módulo de processamento de IA** permite o desenvolvimento paralelo, além de facilitar a integração com futuras funcionalidades.

Durante o planejamento uma alternativa planejada foi o uso de uma arquitetura monolítica, mas optou-se por uma separação lógica de serviços para garantir maior flexibilidade, performance e independência entre os componentes. Além disso, essa estrutura permite que o módulo de IA seja reutilizado ou escalado separadamente, caso necessário.

O uso da linguagem **Java** para o código gerado foi escolhido por ser uma linguagem orientada a objetos amplamente utilizada, didática e com grande aceitação no mercado e no meio acadêmico.

**Visão Inicial da Arquitetura**

A arquitetura do sistema está dividida em cinco principais componentes:

* **Interface Web (Frontend)**: Responsável pela interação com o usuário. Permite login, cadastro, criação de projeto, envio de diagrama e visualização do código gerado.
* **API Backend**: Gerencia autenticação, controle de projetos, roteamento de requisições, e conexão com o módulo de IA e banco de dados.
* **Módulo de Processamento (IA + OCR)**: Realiza o reconhecimento das informações visuais do diagrama UML e extrai os elementos relevantes.
* **Módulo de Geração de Código**: Interpreta os dados extraídos e converte-os em código Java estruturado.
* **Banco de Dados (PostgreSQL)**: Armazena informações de usuários, projetos e logs de geração.

A comunicação entre os módulos ocorre via **requisições REST** em formato JSON, garantindo interoperabilidade entre diferentes tecnologias.

**Padrões de Arquitetura**

* **MVC (Model-View-Controller)**: Utilizado tanto no frontend (Vue.js) quanto no backend (Express.js) para organizar as responsabilidades entre dados, interface e controle.
* **Microserviços (lógico)**: Separação dos módulos de reconhecimento de imagem e geração de código como serviços desacoplados do backend principal, facilitando manutenção e escalabilidade.

**Modelos C4**

**1. Contexto**

O sistema se posiciona como uma ferramenta web acessível por navegadores. O usuário interage com o sistema por meio da interface web, realiza login, cria projetos e envia diagramas para geração de código. O sistema se comunica com serviços de autenticação, banco de dados e módulos de IA.

**2. Contêineres**

* **Frontend (Vue.js + Quasar)**: Navegador do usuário.
* **Backend (Node.js + Express)**: API RESTful responsável por gerenciar usuários, projetos e envio de diagramas.
* **Módulo IA/OCR (Python + OpenCV + Tesseract)**: Contêiner independente para análise dos diagramas.
* **Geração de Código (Python)**: Serviço responsável por traduzir os dados em código Java.
* **Banco de Dados (PostgreSQL)**: Contêiner de dados com autenticação, projetos e histórico.

**3. Componentes**

No backend, os principais componentes são:

* **AuthController** – Gerencia login, cadastro e redefinição de senha.
* **ProjectController** – Gerencia a criação, listagem e exclusão de projetos.
* **DiagramProcessorService** – Responsável por enviar os arquivos para o módulo de IA.
* **CodeGeneratorService** – Recebe os dados processados e gera o código em Java.
* **FileExportService** – Compacta e entrega os arquivos em formato ZIP.

**4. Código**

No nível de código, cada serviço possui organização por responsabilidade:

* **Módulo IA:** scripts separados para OCR, reconhecimento de padrões e parsing.
* **Módulo de Geração:** classes responsáveis por construir trechos de código com base em estruturas genéricas de classes UML.

**Stack Tecnológica**

**Linguagens de Programação**

* **JavaScript / TypeScript**: Utilizadas no frontend (Vue.js + Quasar) e backend (Node.js + Express) para manipulação da lógica da aplicação.
* **Python**: Utilizado nos módulos de inteligência artificial, OCR e geração de código Java.
* **SQL**: Usada para manipulação de dados no banco relacional PostgreSQL.
* **Java (gerado)** – Linguagem alvo da geração automática de código a partir dos diagramas UML.

**Frameworks e Bibliotecas**

* **Frontend:**
  + **Vue.js**: Framework progressivo para criação da interface de usuário.
  + **Quasar Framework**: Framework baseado em Vue.js para interfaces responsivas e com excelente performance.

**Backend:**

* **Node.js**: Ambiente de execução JavaScript para o lado servidor.
* **Express.js**: Framework web leve e flexível para construção da API RESTful.
* **jsonwebtoken (JWT**): Biblioteca para autenticação de usuários.
* **Bcryptjs:** Biblioteca para criptografia de senhas.

**IA / OCR / Geração de Código:**

* **Tesseract OCR:** Reconhecimento de caracteres em imagens.
* **OpenCV:** Biblioteca para visão computacional e processamento de imagem.
* **Scripts próprios** em Python para análise de diagrama e geração de classes Java.

**Ferramentas de Desenvolvimento e Gestão**

* **Git** e **GitHub**: Controle de versão, documentação do projeto e colaboração.
* **Postman**: Testes de endpoints da API RESTful.
* **Visual Studio Code**: IDE principal utilizada durante o desenvolvimento.
* **Prisma ORM**: Para abstração e manipulação do banco de dados relacional (PostgreSQL).

**Banco de Dados**

* **PostgreSQL**: Banco de dados relacional robusto e seguro, utilizado para armazenar informações de usuários, projetos, logs e histórico de geração de código.

**Estratégias e Ferramentas de Testes**

**Testes de Interface (End-to-End)**

* **Cypress**: Ferramenta utilizada para testes de ponta a ponta (E2E), garantindo que os fluxos críticos da aplicação web funcionem corretamente do ponto de vista do usuário.  
  **Exemplo de testes realizados:**
  + Fluxo de login e registro de usuário
  + Criação e exclusão de projetos
  + Upload de diagramas e geração de código

**Testes Unitários**

* **Jest** (para Node.js): Framework de testes utilizado no back-end para validação de funções e serviços isolados, como:
  + Verificação da geração de tokens JWT
  + Testes nos controladores de autenticação e projeto
  + Validação de serviços de processamento de arquivos e exportação
* **Vue Test Utils** (para componentes Vue.js): Utilizado para garantir que os componentes do front-end funcionem conforme esperado.
  + Testes em formulários de login, cadastro e upload
  + Validação de mensagens de erro e interações de interface

**Considerações de Segurança**

A segurança da informação é um aspecto fundamental em aplicações web, especialmente em sistemas que manipulam dados sensíveis de usuários, como credenciais de acesso e arquivos enviados. A seguir são apresentadas as principais medidas adotadas para mitigar riscos de segurança no projeto:

**Autenticação e Controle de Acesso**

* Uso de **JWT (JSON Web Token)** para autenticação segura e controle de sessão.
* Apenas usuários autenticados poderão acessar recursos como criação de projetos e envio de diagramas.
* Tokens possuem tempo de expiração e verificação de integridade via assinatura.

**Proteção de Credenciais**

* As senhas dos usuários serão armazenadas de forma criptografada utilizando **bcrypt**, impedindo a recuperação direta mesmo em caso de vazamento de dados.
* Validação de senhas fortes no momento do cadastro.

**Upload Seguro de Arquivos**

* Validação e sanitização dos nomes dos arquivos para evitar ataques por injeção ou path traversal.
* Armazenamento temporário dos arquivos com exclusão automática após o processamento.

**Comunicação Segura**

* Toda a comunicação entre cliente e servidor será feita através de **HTTPS**, garantindo a confidencialidade e integridade dos dados transmitidos.

**Prevenção contra vulnerabilidades comuns**

* **CORS configurado corretamente** no back-end.
* Proteções contra **XSS (Cross-Site Scripting)** e **CSRF (Cross-Site Request Forgery)** serão aplicadas nas rotas e formulários.
* Validação e sanitização de dados recebidos no back-end para evitar **Injeção de Código ou SQL Injection**.

**Próximos Passos**

Após a conclusão deste documento de RFC, o projeto seguirá para as fases de desenvolvimento e testes práticos, distribuídas entre os módulos de Portfólio I e II. A seguir, apresenta-se uma visão geral do cronograma:

**Portfólio I – Fase Inicial de Desenvolvimento**

* Configuração do ambiente de desenvolvimento (front-end, back-end e IA)
* Estruturação do repositório e definição da arquitetura de pastas
* Implementação do sistema de autenticação (cadastro, login, JWT)
* Criação do módulo de projetos e upload de diagramas UML
* Desenvolvimento da interface inicial com Quasar (Vue.js)
* Integração básica com o módulo de IA para testes iniciais de OCR
* Elaboração de testes automatizados com Jest e Cypress
* Testes manuais dos fluxos principais do sistema

**Portfólio II – Integração e Finalização**

* Aprimoramento do módulo de IA com foco em acurácia e performance
* Integração completa entre front-end, back-end e IA
* Geração automática do código Java a partir dos dados extraídos
* Implementação da visualização, edição e exportação do código em .zip
* Refino da interface e experiência do usuário
* Testes automatizados adicionais (front-end e back-end)
* Documentação final do sistema
* Publicação no GitHub com README atualizado

**Referências**

**Inteligência Artificial e OCR**

* **Tesseract OCR:** <https://github.com/tesseract-ocr/tesseract>
* **OpenCV:** <https://opencv.org>
* **Python:** <https://www.python.org>

**Desenvolvimento Web**

* **Node.js:** <https://nodejs.org>
* **Express.js:** <https://expressjs.com>
* **Vue.js:** <https://vuejs.org>
* **Quasar Framework:** <https://quasar.dev>
* **PostgreSQL:** <https://www.postgresql.org>
* **Prisma ORM:** <https://www.prisma.io>

**Segurança e Autenticação**

* **JSON Web Tokens (JWT):** <https://jwt.io>
* **bcrypt.js:** <https://github.com/dcodeIO/bcrypt.js>

**Testes Automatizados**

* **Cypress:** <https://www.cypress.io>
* **Jest**: <https://jestjs.io>
* **Vue Test Utils:** <https://test-utils.vuejs.org>

**Ferramentas de Desenvolvimento**

* **Visual Studio Code:** <https://code.visualstudio.com>
* **Git e GitHub**: <https://github.com>
* **Postman:** <https://www.postman.com>

Considerações Professor(a):

**Assinatura: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Considerações Professor(a):

**Assinatura: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Considerações Professor(a):

**Assinatura: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**