UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

FERNANDO LISBOA COSTA

VIRTUALIZAÇÃO, REFATORAÇÃO E REVITALIZAÇÃO DO SERVIÇO CURSOS UFCG: UM RELATO DE EXPERIÊNCIA

CAMPINA GRANDE - PB

2023

FERNANDO LISBOA COSTA

VIRTUALIZAÇÃO, REFATORAÇÃO E REVITALIZAÇÃO DO SERVIÇO CURSOS UFCG: UM RELATO DE EXPERIÊNCIA

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Fábio Jorge Almeida Morais

CAMPINA GRANDE - PB

2023

FERNANDO LISBOA COSTA

VIRTUALIZAÇÃO, REFATORAÇÃO E REVITALIZAÇÃO DO SERVIÇO CURSOS UFCG: UM RELATO DE EXPERIÊNCIA

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA

Fábio Jorge Almeida Morais

Orientador – UASC/CEEI/UFCG

Hyggo Oliveira de Almeida

Examinador – UASC/CEEI/UFCG

Francisco Vilar Brasileiro

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em 28 de junho de 2023.

CAMPINA GRANDE - PB

RESUMO

À medida que um *software* envelhece, muitas das tecnologias que o compõem se tornam obsoletas e difíceis de evoluir. Além disso, novas abordagens para construir serviços surgem, como a virtualização de ambientes de execução, que proporciona maior facilidade na integração de novos desenvolvedores e no gerenciamento de dependências. Neste trabalho, o objetivo é aplicar processos de virtualização, refatoração e revitalização ao sistema web Cursos UFCG, utilizado pelos alunos da Universidade Federal de Campina Grande para visualizar grades de cursos, simular composições curriculares e analisar matrículas. O sistema é composto por código legado e enfrenta desafios relacionados à evolução e gerenciamento de dependências. Para lidar com essas questões, foi utilizada a plataforma Docker para virtualização, combinada com análise da estrutura do projeto e suas dependências. O objetivo principal é modernizar o ambiente de execução e aplicar refatorações para se adequar a um novo conjunto de ferramentas e linguagem de programação. Espera-se que esse trabalho melhore o ciclo de vida do desenvolvimento de *software* e atualize as tecnologias empregadas no projeto.

VIRTUALIZATION, REFACTORING, AND REVITALIZATION OF THE COURSES UFCG SERVICE: AN EXPERIENCE REPORT

ABSTRACT

As a software ages, many of the technologies it relies on become outdated and difficult to evolve. In addition to that, new ways of building services emerge, such as virtualization of execution environments, which provide easier integration of new developers and dependency management. This work aims to apply virtualization and refactoring processes to the server of the web system Cursos UFCG, used by students at the Federal University of Campina Grande to view course schedules, simulate curriculum compositions, and analyze enrollments. The system consists of legacy code and faces challenges related to evolution and dependency management. To address these issues, the Docker platform was used for virtualization, along with an analysis of the project's structure and its dependencies. The goal is to modernize the execution environment and apply refactoring to adapt to a new set of tools and programming language. It is expected that this work will improve the software development lifecycle and update the technologies employed in the project.

Virtualização, Refatoração e Revitalização do Serviço Cursos UFCG: Um Relato de Experiência

Fernando Lisboa Costa
Unidade Acadêmica de Sistemas e Computação
Universidade Federal de Campina Grande
Campina Grande - PB, Brasil
fernando.costa@ccc.ufcg.edu.br

Fábio Jorge Almeida Morais
Unidade Acadêmica de Sistemas e Computação
Universidade Federal de Campina Grande
Campina Grande - PB, Brasil
fabio@computacao.ufcg.edu.br

RESUMO

À medida que um *software* envelhece, muitas das tecnologias que o compõem se tornam obsoletas e difíceis de evoluir. Além disso, novas abordagens para construir serviços surgem, como a virtualização de ambientes de execução, que proporciona maior facilidade na integração de novos desenvolvedores e no gerenciamento de dependências. Neste trabalho, o objetivo é aplicar processos de virtualização, refatoração e revitalização ao sistema web Cursos UFCG, utilizado pelos alunos da Universidade Federal de Campina Grande para visualizar grades de cursos, simular composições curriculares e analisar matrículas. O sistema é composto por código legado e enfrenta desafios relacionados à evolução e gerenciamento de dependências. Para lidar com essas questões, foi utilizada a plataforma Docker para virtualização, combinada com análise da estrutura do projeto e suas dependências. O objetivo principal é modernizar o ambiente de execução e aplicar refatorações para se adequar a um novo conjunto de ferramentas e linguagem de programação. Espera-se que esse trabalho melhore o ciclo de vida do desenvolvimento de *software* e atualize as tecnologias empregadas no projeto.

PALAVRAS-CHAVE

Virtualização, refatoração, revitalização, dockerização, Cursos UFCG.

REPOSITÓRIO

https://github.com/fernandollisboa/cursos-ufcg-node-backend

1. INTRODUÇÃO

A fim de aproveitar-se das melhores técnicas disponíveis no mercado, desenvolver projetos em ambientes virtualizados tornou-se prática comum em equipes que visam garantir que seus serviços sejam distribuídos da melhor forma. Até o início dos anos 2010, grandes empresas de tecnologia costumavam difundir seus produtos de *software* por meio de arquiteturas monolíticas, compiladas em um único arquivo executável. No entanto, com o avanço dos estudos na área da computação, esse tipo de arquitetura acabou se provando problemática em alguns contextos, o que impulsionou o surgimento de novas alternativas de desenvolvimento.

Uma dessas soluções é a virtualização, que permite a divisão do *software* em módulos distintos, chamados de serviços, proporcionando maior flexibilidade na integração entre sistemas e facilitando a distribuição de funcionalidades. Essa abordagem descentralizada possibilita tratar cada módulo como uma entidade isolada.

A adoção de processos de virtualização traz diversos benefícios em termos de tolerância a falhas e disponibilidade [1]. A abordagem modular da virtualização permite o desenvolvimento e a implementação separada de serviços independentes. Isso significa que, caso ocorra uma falha em um serviço específico independente, apenas ele será afetado, enquanto os demais continuarão funcionando normalmente. Não obstante, a virtualização oferece a flexibilidade de escalar individualmente cada serviço de acordo com sua demanda específica, possibilitando o provisionamento vertical ou horizontal para lidar com um maior volume de requisições, sem impactar os outros serviços.

Utilizando ferramentas modernas, como o gerenciador de contêineres Docker [2], a virtualização de aplicações tornou-se mais acessível e eficaz. O Docker possibilita preparar ambientes de execução que gerenciam módulos independentes, mesmo em sistemas originalmente monolíticos, evitando conflitos de versão e configuração. Além disso, a atualização e a refatoração do código são essenciais para manter um *software* relevante e funcional ao longo do tempo. A verificação de versões instaladas, a substituição de funções e métodos obsoletos e a melhoria na qualidade do código são práticas fundamentais para garantir a vida útil de uma aplicação.

No contexto de serviços legados e não virtualizados, podemos destacar a aplicação Cursos UFCG [3], que se encaixa nessa classificação. O serviço criado pelo Laboratório de Sistemas Distribuídos da Universidade Federal de Campina Grande tem como objetivo auxiliar os estudantes de graduação no planejamento de suas disciplinas durante o curso, oferecendo informações sobre matrículas, taxas de reprovação e sugestões de disciplinas. No entanto, desde sua criação em 2016 pela equipe Analytics, o sistema não recebeu atualizações constantes e enfrenta problemas decorrentes do envelhecimento do *software*, como código legado, API lenta devido ao uso de Python e referência a um banco de dados desatualizado.

Diante disso, o objetivo deste estudo é realizar processos de virtualização, refatoração e revitalização do serviço Cursos UFCG. O foco principal é na melhoria de indicadores de desempenho, cobertura de testes e robustez da arquitetura.

2. REFERENCIAL TEÓRICO

Nesta seção serão tecidas considerações sobre conceitos utilizados durante o processo de virtualização do sistema Cursos.

2.1 Virtualização de Aplicações

O conceito de virtualização, conforme definido em [4], descreve a criação de uma camada de abstração entre o *hardware* e o *software*. Essa camada tem o propósito de proteger o acesso direto do *software* aos recursos físicos do *hardware*. A virtualização possibilita que o *software* opere em um ambiente virtualizado isolado, sem a necessidade de conhecimento ou dependência direta dos detalhes do *hardware* subjacente.

A camada de abstração proporcionada pela virtualização permite que o *software* seja replicado e executado em diferentes infraestruturas, o que se torna especialmente útil em cenários onde há a necessidade de migrar ou executar o mesmo *software* em diferentes sistemas ou ambientes, sem a necessidade de reescrevê-lo ou adaptá-lo significativamente.

Os benefícios da virtualização são significativos. Um deles é a capacidade de criar ambientes independentes e executar várias máquinas virtuais em um único servidor, resultando em economia de espaço físico e recursos computacionais, além de permitir o compartilhamento de um único servidor entre múltiplos ambientes de execução. A virtualização também simplifica a execução em diferentes ambientes, como servidores locais ou em nuvem, permitindo uma implantação flexível do sistema em provedores de serviços em nuvem ou servidores próprios, sem a necessidade de realizar grandes adaptações na aplicação. Isso proporciona maior flexibilidade e escalabilidade para as aplicações e sistemas.

2.1.1 Máquinas Virtuais

Segundo a descrição fornecida em [5], máquinas virtuais (*Virtual Machines - VM's)* são ambientes de execução de *software* que operam de forma isolada e são capazes de emular sistemas operacionais, replicando as características do *hardware* de um servidor físico. As VMs abstraem virtualmente todos os componentes da máquina base (*host*) em sua contraparte virtualizada (*guest*), incluindo o *kernel* do sistema operacional e suas funcionalidades.

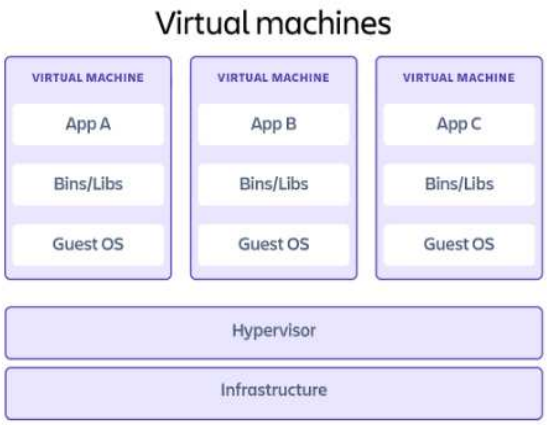


Figura 1: Arquitetura da Virtualização com VM's

Na arquitetura da virtualização com máquinas virtuais, como ilustrado na Figura 1, cada VM é executada em um ambiente isolado, com seu próprio sistema operacional e recursos virtuais atribuídos. Essa abordagem permite que cada VM funcione como uma entidade independente, capaz de executar seus próprios aplicativos e serviços. Além disso, várias VMs podem compartilhar a mesma infraestrutura física, aproveitando ao máximo os recursos disponíveis, proporcionando a utilização mais eficiente dos servidores e permitindo a consolidação de várias VMs em um único *hardware*. Além disso, o compartilhamento da infraestrutura física possibilita o gerenciamento flexível e dinâmico dos recursos, permitindo a provisão conforme a demanda. A virtualização também permite remontar o mesmo ambiente em diferentes máquinas ou plataformas, facilitando a migração de uma VM de um servidor para outro sem interrupção do serviço, garantindo disponibilidade contínua do sistema.

2.1.2 Conteinêres

Já os contêineres, conforme descrito em [5], são uma abstração que empacota o código e suas dependências. Diferentemente das VMs, os contêineres compartilham o *kernel* do sistema operacional com outros contêineres, sendo executados como processos isolados. Essa abordagem oferece uma alternativa mais leve em comparação com as VMs, ocupando menos espaço e permitindo que mais aplicativos sejam executados em paralelo.

Uma das principais ferramentas para criação e gerenciamento de contêineres é o Docker. O Docker é conhecido por sua eficiência e facilidade de uso, permitindo a criação de ambientes independentes que encapsulam os componentes necessários para a execução de uma aplicação. Isso inclui bibliotecas, *frameworks* e configurações. Com o Docker, é possível empacotar e distribuir aplicativos de maneira consistente, garantindo sua portabilidade e facilitando o processo de implantação em diferentes ambientes de desenvolvimento e produção.

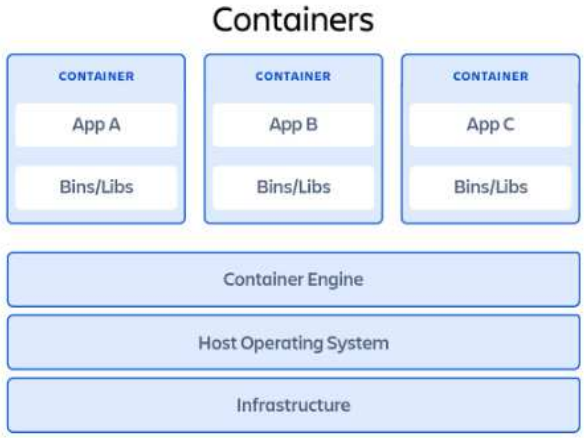


Figura 2: Arquitetura da Virtualização com Contêineres

3. METODOLOGIA

Nesta seção serão detalhados os aspectos metodológicos utilizados neste estudo.

3.1 Análise do Projeto Original

Nesta subseção, será realizada uma análise detalhada do projeto Cursos UFCG na sua forma original, com o objetivo de compreender sua estrutura e funcionalidades. Serão examinados o repositório original do servidor [6], o repositório do *frontend* [7] e seu deploy público [3], além de documentos contidos em seus subdiretórios e outros materiais relevantes.

3.1.1 Descrição Geral

Nesta seção, serão descritas as funcionalidades do serviço como um todo, tendo como referência o deploy do *frontend* para fins ilustrativos. O Cursos UFCG é composto por dois componentes principais: o *backend* e o *frontend*, que trabalham em conjunto para formar um sistema *web* responsável por fornecer o serviço em questão. O objetivo principal é auxiliar os estudantes de graduação da Universidade Federal de Campina Grande (UFCG) na visualização dos fluxogramas de composição curricular dos seus cursos. Isso é possível através da ferramenta de busca, na qual o usuário pode encontrar o curso desejado ao digitar o seu nome na barra de pesquisa, exemplificada na Figura 3.

Figura 3: Página inicial do Cursos

Após realizar a busca e selecionar o curso desejado, o usuário é redirecionado para a página do fluxograma correspondente. Nessa página, o usuário tem acesso a uma representação visual das disciplinas que compõem a grade curricular do curso, organizadas em períodos letivos, o que pode ser observado na Figura 4.

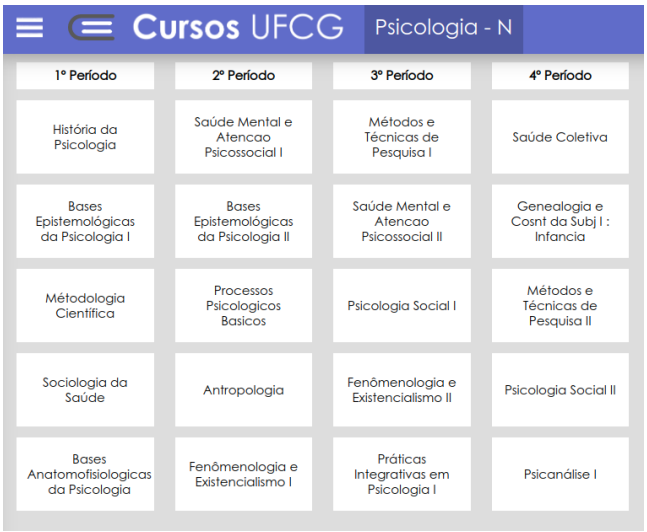


Figura 4: Página de fluxograma do curso de Psicologia (Campina Grande)

Dentro da página do fluxograma, o usuário tem acesso a um menu lateral que oferece diversas opções de navegação para complementar a visualização das disciplinas do curso, como representado na Figura 5.

Figura 5: Menu lateral da página de fluxograma

A página "Minha Grade" permite que o usuário construa interativamente a sua própria grade curricular e visualize quais disciplinas estão disponíveis para matrícula. Para facilitar a compreensão da disponibilidade, as disciplinas são destacadas com cores diferentes. Nessa página, as disciplinas já selecionadas ou cursadas são mostradas em cores preenchidas e aquelas disponíveis para matrícula são exibidas em branco e as disciplinas indisponíveis são apresentadas de forma desbotada, em cinza, conforme na Figura 6. Isso ajuda o usuário a identificar rapidamente as disciplinas que não estão disponíveis no momento por restrição de pré-requisitos.

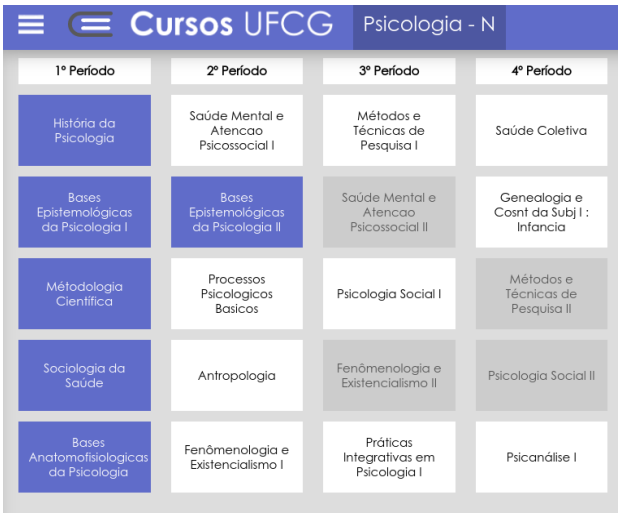


Figura 6: Página Minha Grade

A página "Taxa de Aprovação" apresenta as taxas de reprovações de cada disciplina do fluxograma, assim como um *slider* que permite filtrar os períodos considerados no levantamento dos dados de reprovação. Cada disciplina é representada por um retângulo, onde o preenchimento da área é proporcional à taxa de reprovação. Quanto maior a taxa de reprovação, maior será a área colorida dentro do retângulo correspondente à disciplina, exemplificada na Figura 7.

Essa representação visual facilita a identificação das disciplinas com maiores índices de reprovação, possibilitando uma análise comparativa entre elas. Os usuários podem avaliar e identificar as disciplinas que requerem maior atenção e dedicação.

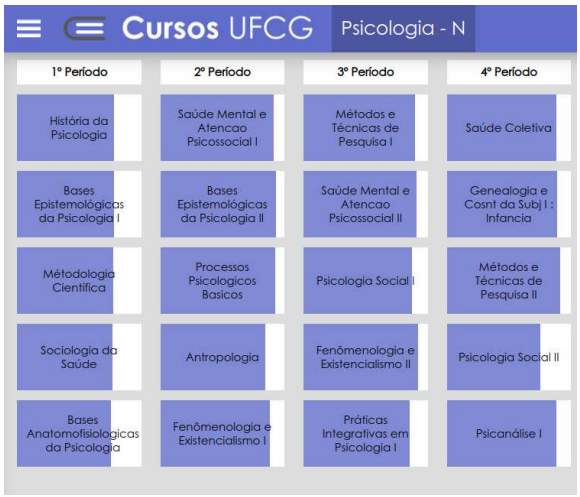


Figura 7: Página Taxa de Reprovação

A página Correlação permite ao usuário visualizar as conexões entre as disciplinas por meio de um grafo. Essa representação é obtida a partir da análise do banco de dados contendo o histórico dos estudantes e suas matrículas. Ao explorar o grafo na página Correlação, o usuário pode navegar pelas conexões entre as disciplinas, obtendo uma visão mais abrangente do contexto acadêmico, conforme a Figura 8. Essa visualização auxilia na tomada de decisões, como a escolha de disciplinas similares e a identificação de possíveis caminhos alternativos no percurso acadêmico.

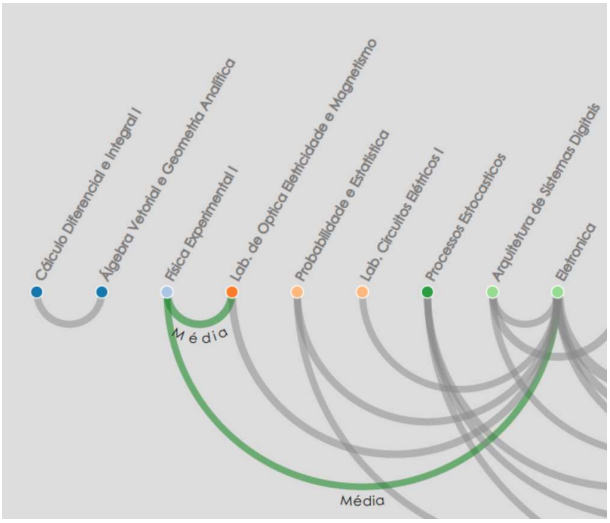


Figura 8: Página Correlação

A página Raio-X exibe informações gerais do curso para o usuário, como o *ranking* das disciplinas com maior taxa de reprovação e a quantidade de alunos que já se formaram, divididos por ano, como pode ser visto na Figura 9.

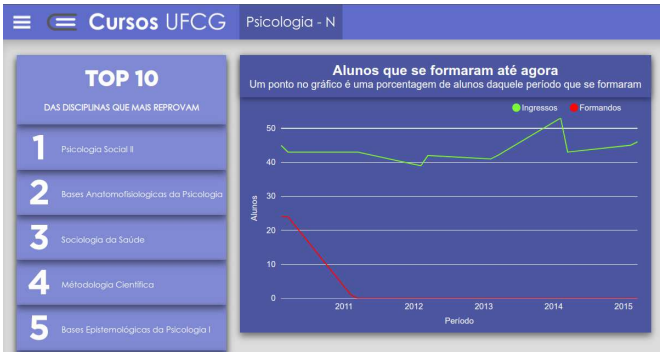


Figura 9: Página de Raio-X

Além das funcionalidades básicas, o sistema também oferece a funcionalidade de Análise de Matrícula. Para acessá-la, o usuário precisa estar na página Minha Grade e preencher as disciplinas desejadas para se matricular em um determinado período, juntamente com as disciplinas previamente cursadas. Após aguardar o cálculo realizado pelos scripts em R, o sistema gera um relatório que apresenta um indicador de risco de reprovação da matrícula como um todo, exemplificada na Figura 10.

Adicionalmente, existe a função "Recomendação de Matrícula", que está disponível apenas para os cursos de Engenharia Elétrica (Campina Grande) e Ciência da Computação - D (Campina Grande). Essa funcionalidade utiliza algoritmos específicos para analisar o histórico acadêmico do aluno e recomendar disciplinas para matrícula, levando em consideração as disciplinas previamente cursadas e os pré-requisitos.

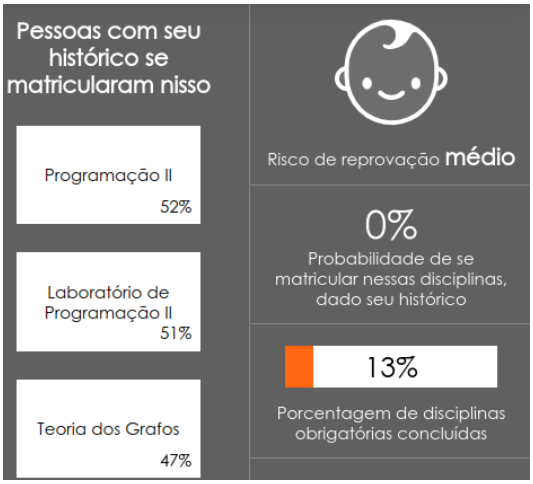


Figura 10: Função Recomendação e Análise de Matrícula

3.1.2 Funcionalidades do Servidor

O servidor do serviço Cursos UFCG oferece uma variedade de funcionalidades por meio dos seus endpoints. Essas funcionalidades incluem:

- **Informações gerais sobre um curso específico (/:curso):** Esse endpoint fornece informações gerais sobre um curso em particular, como seu nome, *campus*, quantidade de vagas, horas, turnos, etc.

- **Disciplinas ofertadas por um curso específico (/:curso/disciplinas):** Com esse endpoint, é possível obter a lista de disciplinas disponíveis em um curso, incluindo informações sobre cada disciplina, como código, nome e carga horária.
- **Quantidade de formandos de um curso específico (/:curso/formandos):** Com esse endpoint, é possível obter a quantidade de alunos que se formaram em um curso específico, divididos por semestre.
- **Taxa de sucesso de um curso específico (/:curso/taxa-sucesso):** Esse endpoint calcula a taxa de sucesso de um curso, ou seja, a porcentagem de alunos que foram aprovados em relação ao total de alunos matriculados.
- **Taxa de sucesso de um curso específico, divido por período (/:curso/taxa-sucesso/periodos):** Com esse endpoint, é possível obter a taxa de sucesso do curso dividida por períodos específicos. Isso permite uma análise mais granular do desempenho dos alunos ao longo do tempo.
- **Geração de estatísticas de um curso específico (/:curso/estatisticas):** Esse endpoint gera estatísticas relevantes sobre um curso, fornecendo informações como o número de alunos regulares matriculados no curso, o número de alunos que se formaram e a taxa de reprovação nas disciplinas do curso.
- **Cálculo da correlação entre disciplinas de um curso específico (/:curso/correlacao):** Esse endpoint realiza o cálculo da correlação entre as disciplinas de um curso específico. Isso ajuda a identificar quais disciplinas possuem uma relação mais forte entre si, permitindo uma melhor visualização das interconexões no currículo do curso.
- **Análise de matrícula com base no histórico do aluno (/:curso/analise):** Esse endpoint realiza uma análise detalhada da matrícula em um curso específico, levando em consideração o histórico do aluno e as disciplinas selecionadas. Ele fornece informações sobre a matrícula em um curso específico, incluindo taxa de conclusão do curso, frequência de matrícula e risco de reprovação.
- **Recomendação de matrícula com base no histórico do aluno (/:curso/recomendacao):** Com esse endpoint, é possível gerar recomendações de matrícula personalizadas para um aluno, com base em seu histórico acadêmico e nos requisitos do curso. Isso auxilia os alunos na escolha de disciplinas adequadas para o próximo período letivo.

3.1.3 Arquitetura do Servidor

O servidor do Cursos UFCG original consiste em uma API desenvolvida na linguagem Python [8] compatível com a versão 2.7 e utilizando o *framework* Flask [9]. A fração principal da aplicação principal consiste em somente um arquivo monolítico responsável por tratar das requisições do cliente, aplicar regras de negócio e realizar chamadas para um banco de dados remoto através do protocolo ODBC[10] (Open Database Connectivity). Há também um arquivo contendo código que não é mais utilizado (apiRestOld.py).

Além disso, há um diretório contendo *scripts* na linguagem R [11], responsável por computar funcionalidades complementares da aplicação, como criar redes bayesianas para analisar correlações entre disciplinas e analisar sugestões de matrículas. A estrutura geral de diretórios do projeto está conforme ilustrada na Figura 11.

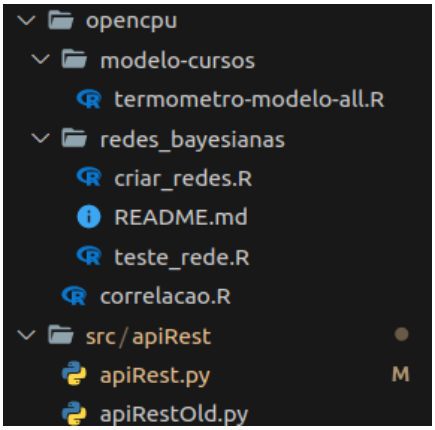


Figura 11: Estrutura do projeto original

No âmbito deste estudo, o objetivo é recriar e virtualizar a execução da API que está definida no arquivo apiRest.py. Para isso, será adotada uma abordagem baseada em camadas, com o intuito de facilitar a manutenção do código e preservar as funcionalidades e *endpoints* originais, garantindo assim a retrocompatibilidade. Vale ressaltar que o trecho em R não será utilizado neste momento e será deixado para um estudo futuro. Além disso, a antiga API (apiRestOld.py) não será revitalizada, uma vez que suas funcionalidades não são relevantes para o estado atual da aplicação.

3.1.4 Pontos de Melhoria

Durante a análise exploratória do código original, foram identificados os seguintes pontos de melhoria:

- **API definida em um só arquivo**: A lógica de gerenciamento da aplicação está concentrada em um único arquivo, tornando-o difícil de entender, manter e reutilizar o código. Recomenda-se modularizar a estrutura da API em diferentes arquivos e camadas, seguindo boas práticas de arquitetura de *software*. Isso facilitará a organização do código e permitirá um trabalho colaborativo mais eficiente [12].
- **Falta de Clean Code**: O código original não segue as práticas de Clean Code, apresentando nomenclatura inconsistente e pouco semântica em diversos trechos. Isso dificulta a compreensão da lógica, torna as modificações mais complexas e pode levar a erros.
- **Sem definição de versão das dependências**: As bibliotecas e *frameworks* utilizados no código original não possuem versões especificadas. Isso pode resultar em problemas de compatibilidade ou conflitos entre dependências, especialmente em diferentes ambientes de execução e desenvolvimento.
- **Código em mais de um idioma diferente**: O código original contém trechos escritos em inglês e português. É uma prática recomendada manter o código em um

único idioma, a menos que exista uma justificativa específica para utilizar múltiplos idiomas.

- **Falta de testes**: A aplicação não possui cobertura de testes, o que aumenta o risco de surgimento de bugs inesperados. É fundamental implementar testes unitários para cada função da camada de serviço da aplicação. Isso permitirá validar o comportamento esperado do código e garantir sua qualidade.
- **Alto tempo de respostas para requisições utilizando o OpenCPU**: As requisições dependentes do trecho em R da aplicação apresentam um tempo de resposta elevado, geralmente superior a 4000ms. Isso pode resultar em funcionalidades que não são responsivas o suficiente para atender às expectativas dos usuários e até mesmo inviabilizar algumas partes da aplicação.

Figura 12: Resultado da suíte de testes de integração. Na terceira coluna, os tempos de resposta das requisições

3.2 Ferramentas Utilizadas nos Processos de Virtualização e Revitalização

O servidor foi reimplementado em Node.js [13] utilizando o *framework* Express [14] e a linguagem de programação JavaScript [15]. Essas escolhas foram baseadas no desempenho oferecido por essas tecnologias em comparação à implementação anterior em Python/Flask. Além disso, a compatibilidade nativa do Node.js com o formato JSON foi um fator relevante, considerando a necessidade de melhorar o desempenho da aplicação ao manipular e compartilhar objetos do *database* original.

Para a virtualização do sistema, foi adotada a plataforma Docker, que viabiliza a criação e orquestração de ambientes virtuais por meio de contêineres. A escolha dessa tecnologia se dá pelo fato da sua flexibilidade e eficiência no empacotamento de aplicações e suas dependências.

Foi implementada uma instância do banco de dados NoSQL Redis [16] para gerenciar o armazenamento em *cache* das requisições. O Redis é um sistema de armazenamento de dados de alta performance em memória, amplamente utilizado para armazenamento em cache de dados devido à sua rápida recuperação e baixa latência. Sua estrutura de dados em chave-valor permite um acesso eficiente aos dados, tornando-o uma escolha adequada para o gerenciamento de *cache*.

Para verificação das melhorias práticas, foi criada uma coleção de testes com a extensão para VSCode [17] ThunderClient [18], além da criação de testes de unidade utilizando o *framework* de testes Jest [19]. Essas abordagens foram adotadas com o objetivo de garantir a qualidade do sistema, permitindo a validação das suas funcionalidades e desempenho.

4. RESULTADOS E DISCUSSÃO

Nesta seção, serão apresentados os principais resultados alcançados durante a virtualização, refatoração e revitalização do serviço Cursos UFCG**.**

4.1 Virtualização

Nesta subseção, serão descritos os processos envolvidos na criação do ambiente de execução virtualizado do Cursos UFCG. O uso do Docker possibilitou a criação de uma imagem personalizada que contém todos os componentes necessários para a execução do serviço de forma isolada. A imagem base utilizada, disponível no Docker Hub [20], já conta com a instalação do Node.js na versão 18 e do sistema operacional Linux Alpine [21]. Além disso, foi realizada a configuração da instalação do Driver ODBC para suportar a utilização do protocolo de banco de dados. A construção da imagem Docker foi simplificada através do comando "npm run docker:build", tornando mais fácil a criação e atualização do ambiente de execução. Um trecho do Dockerfile utilizado pode ser visualizado na Figura 13.

```
FROM node:18

WORKDIR /app

COPY package.json .

ENV ACCEPT_EULA=Y

RUN apt-get update -y && apt-get update \
  && apt-get install -y --no-install-recommends curl gcc g++ gnupg unixodbc-dev

# Install MySQL ODBC driver
ADD https://dev.mysql.com/get/Downloads/Connector-ODBC/8.0/mysql-connector-odbc-8
RUN tar -C . -xzvf mysql-connector-odbc-8.0.26-linux-glibc2.12-x86-64bit.tar.gz
RUN cd mysql-connector-odbc-8.0.26-linux-glibc2.12-x86-64bit \
  && cp -r lib/* /usr/local/lib \
  && cp -r bin/* /usr/local/bin \
  && myodbc-installer -a -d -n "MySQL ODBC 8.0 Driver" -t "Driver=/usr/local/lib/
  && myodbc-installer -a -d -n "MySQL ODBC 8.0" -t "Driver=/usr/local/lib/libmyod
```

Figura 13: Trecho do Dockerfile

O Docker Compose [22] foi utilizado para gerenciar a execução dos contêineres do serviço Cursos UFCG e suas dependências. Ele permite definir e coordenar a infraestrutura necessária para executar o serviço em conjunto com outros componentes, como a conexão com o banco de dados de *cache* Redis. O comando "npm run docker-compose:up" permite iniciar os contêineres de forma conveniente e garantir a correta interconexão entre eles. O documento Docker Compose está ilustrado na Figura 14.

```yaml
version: '3.9'
services:
  redis:
    container_name: cursos_api_redis
    image: redis:alpine
    networks:
      - cursos-network
    ports:
      - '6379:6379'
  app:
    container_name: cursos_api_node
    build:
      context: ..
      dockerfile: ./docker/Dockerfile
    environment:
      REDIS_HOST: redis
    ports:
      - '3000:3000'
    depends_on:
      - redis
    networks:
      - cursos-network
networks:
  cursos-network:
    driver: bridge
```

Figura 14: Arquivo Docker Compose

Com essas definições, é possível executar a aplicação em diferentes ambientes de forma mais consistente. A virtualização resultante oferece uma camada de abstração que define e isola o serviço Cursos UFCG e suas dependências, garantindo que ele funcione de maneira previsível, independentemente do ambiente de hospedagem. Além disso, a utilização do Docker Compose simplifica o gerenciamento das interconexões entre os contêineres do sistema, permitindo uma configuração facilitada do ambiente de execução.

4.2 Refatoração e Revitalização

Nesta subseção, serão tecidas considerações sobre os processos de refatoração e revitalização do serviço Cursos UFCG.

4.2.1 Nova Arquitetura

O serviço Cursos, em sua forma original, não possuía divisão entre as camadas da aplicação comumente aplicadas em *software*, como *Services, Controllers* e *Repositories*. Em seu lugar, todas as aplicações de regras de negócio e validação de dados eram realizadas em um só arquivo, o que resultava em um código denso e difícil de se manter, testar e evoluir.

Para contornar esse problema, a nova versão do projeto foi projetada utilizando o modelo Controller-Service-Repository. Esse modelo arquitetural visa aprimorar a organização e modularidade do código através da divisão clara de responsabilidades entre os diferentes componentes do sistema.

O modelo Controller-Service-Repository divide a lógica da aplicação em três camadas distintas:

- **Controller**: Responsável por receber as requisições HTTP e lidar com as interações entre a API e o cliente. O *Controller* valida os dados de entrada e direciona as chamadas para os *Services* adequados. Ele atua como uma ponte entre o cliente e as demais camadas da aplicação.
- **Service**: Concentra a lógica de negócio da aplicação. O *Service* é responsável por realizar operações mais complexas, coordenar a interação entre componentes do sistema e implementar as regras de negócio. Ele utiliza os *Repositories* para acessar e manipular os dados necessários para executar as funcionalidades da aplicação.
- **Repository**: Responsável pela interação com a camada de banco de dados. O *Repository* fornece uma abstração para acessar e manipular os dados do sistema. Ele abstrai os detalhes específicos do banco de dados, permitindo que a lógica de negócio se concentre nas camadas relevantes.

A estrutura do projeto resultante é ilustrada conforme a Figura 15.

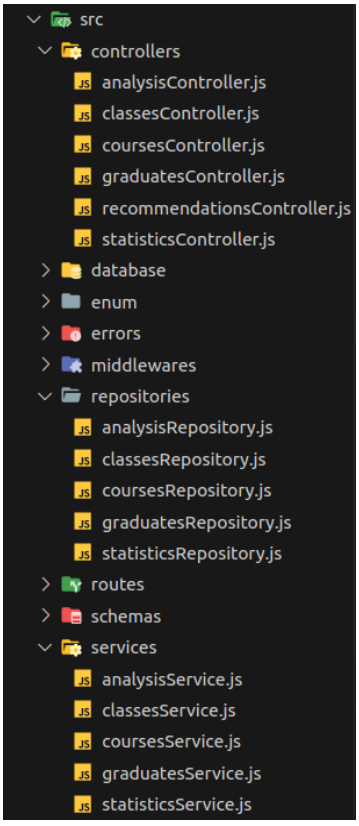


Figura 15: Estrutura do novo projeto Cursos UFCG

Além das camadas supracitadas, também foi criado o diretório *routes*, contendo arquivos que definem os endpoints da aplicação e manipula o comportamento das rotas, mapeando-os conforme requisição do cliente.

O diretório *middlewares* contém trechos de código reutilizáveis e comuns a várias rotas diferentes, modularizando funções como validação de parâmetros de rota via biblioteca Joi [23] e tratamento de erros em todo o sistema.

Em *databases*, foram definidas as formas de acesso aos dados persistidos no sistema, seja via conexão ODBC ou utilizando o armazenamento em *cache* Redis.

O uso desse modelo de arquitetura promove a separação de responsabilidades, facilitando a manutenção, teste e a evolução da aplicação. Cada camada possui um papel bem definido, o que melhora a organização do código e a reutilização de componentes.

4.2.2 Definições de Dependências

As definições de dependências foram possíveis graças à utilização do npm (Node Package Manager) [24], ferramenta utilizada no desenvolvimento de aplicações Node.js. O npm permite gerenciar as dependências do projeto através do arquivo package.json, onde é possível listar todas as dependências necessárias para o funcionamento da aplicação.

Nesse arquivo, são especificados os nomes das dependências, suas versões utilizadas e outras informações relevantes, conforme a Figura 16. Com base nesse arquivo, o npm é capaz de baixar e instalar as dependências de forma automatizada, garantindo que todas as bibliotecas necessárias estejam disponíveis para o projeto em conformidade com suas versões.

```
  "author": "@fernandollisboa",
  "license": "MIT",
  "dependencies": {
    "axios": "^1.4.0",
    "cors": "^2.8.5",
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "joi": "^17.9.1",
    "odbc": "^2.4.7",
    "redis": "^4.6.5"
  },
  "devDependencies": {
    "@sucrase/jest-plugin": "^3.0.0",
    "eslint": "^8.37.0",
    "eslint-config-prettier": "^8.8.0",
    "eslint-plugin-import": "^2.27.5",
    "eslint-plugin-prettier": "^4.2.1",
    "husky": "^8.0.3",
    "jest": "^29.5.0",
    "nodemon": "^2.0.22",
    "sucrase": "^3.31.0"
  }
```

Figura 16: Dependências listadas no arquivo package.json

Com o uso do npm e a definição explícita das dependências no arquivo package.json, é possível garantir a consistência e a estabilidade das versões bibliotecas utilizadas no projeto, assegurando a execução correta em diferentes ambientes de execução e até mesmo facilitando o trabalho em equipe

4.2.3 Implementação de Cache

No serviço Cursos UFCG, o Redis foi adotado como mecanismo de *cache* para armazenar consultas realizadas ao banco de dados ODBC, com o objetivo de reduzir a carga de requisições direcionadas ao banco. Foi estabelecido um tempo de expiração padrão para os dados armazenados em *cache*, garantindo que o conteúdo permaneça atualizado com certa frequência.

A implementação do Redis como *cache* foi realizada utilizando a biblioteca redis [25] para Node.js, que oferece uma interface para estabelecer a conexão com o Redis e executar operações de armazenamento e recuperação de dados. O Redis foi configurado como um serviço independente, sendo acessado pelo serviço Cursos UFCG por meio de uma conexão TCP.

Dessa forma, antes de realizar consultas ao banco de dados SQL, o serviço verifica se os dados necessários estão presentes em *cache*, potencialmente reduzindo a carga de requisições no banco de dados principal. Além disso, caso não seja possível estabelecer uma conexão com o Redis, o serviço continua funcionando sem problemas, utilizando apenas o banco de dados SQL através do ODBC.

4.2.4 Melhorias em Clean Code

No processo de refatoração, foram identificados diversos problemas no código do projeto antigo que afetavam sua legibilidade e manutenibilidade. Dentre eles, foi observada a presença de trechos de código repetidos, nomes de variáveis e funções pouco semânticas e, em alguns casos, escritos em português. Além disso, foram encontrados trechos de SQL misturados em meio ao código da aplicação.

Para solucionar esses problemas, foram adotadas medidas como a eliminação de código repetido, a padronização de nomes de variáveis e funções seguindo as melhores práticas de nomenclatura e a separação dos trechos de SQL em um módulo à parte *(Repository)*.

Essas mudanças contribuíram para um código mais limpo e de fácil manutenção. Também foram identificados trechos de código reutilizados em várias partes do sistema, e foram criadas abstrações adequadas para melhorar a reutilização do código.

4.2.5 Considerações sobre Refatoração

Durante o processo de refatoração, foi possível realizar melhorias significativas no código. No entanto, alguns trechos específicos precisaram ser mantidos como estavam para garantir a retrocompatibilidade com as integrações existentes. Um exemplo disso são os atributos provenientes das requisições HTTP, como ilustrados nas Figuras 17 e 18, que são usados nos retorno da rota "/:curso/analise" e na requisição de análise de matrícula enviada para a OpenCPU, respectivamente.

```
return {
  taxa_complecao: completionPercentage,
  frequencia_matricula: enrollmentFrequency,
  risco_reprovacao: failingRisk,
};
```

Figura 17: Atributos de retorno da rota /:curso/analise

```
const param = {
  historico_: `c(${transcript})`,
  disciplinas: `c(${courses})`,
  course_name: `"${courseName}"`,
  p: maxSemester,
};
```

Figura 18: Atributos de requisição da função "Recomendação de Matrícula", enviados para o OpenCPU

Esses detalhes foram preservados para evitar alterar o comportamento da API e garantir que as integrações existentes continuem funcionando corretamente. É importante ressaltar que, em futuras versões do programa, será necessário levar em conta esses trechos e considerar a modificação para seguir o padrão mais utilizado em API's RESTFul, o camelCase, além da tradução dos atributos para o idioma inglês.

4.2.6 Considerações sobre OpenCpu

Neste estudo, a revitalização do serviço Cursos UFCG teve foco nas melhorias gerais do sistema, sem abranger especificamente a tecnologia OpenCpu. Embora o OpenCpu já esteja integrado ao serviço, a decisão foi tomada de não incluir modificações relacionadas a ele no processo de revitalização. Essa escolha baseou-se no escopo do projeto, que se concentrou na otimização do código, na modernização da arquitetura, na virtualização do ambiente de execução e na melhoria do desempenho geral do sistema.

É importante ressaltar que o OpenCpu, sendo uma tecnologia adicional, pode requerer atenção específica em relação ao seu desempenho. Embora não tenham sido realizadas modificações diretas no OpenCpu durante a revitalização, é recomendável que trabalhos futuros considerem a possibilidade de analisar e otimizar o desempenho dessa tecnologia, a fim de melhorar a experiência dos usuários.

4.3 Testes

Com o objetivo de verificar a validade das modificações realizadas no estudo, foram desenvolvidas suítes de testes automatizados. Esses testes foram projetados para avaliar se as alterações implementadas atingiram os resultados esperados e se não introduziram novos problemas ou efeitos indesejados. Levando em consideração que a nova arquitetura implementada aumentou o nível de testabilidade da aplicação, o desenvolvimento dos testes ocorreu sem problemas.

Dessa forma, foram implementados testes de unidade utilizando a ferramenta Jest. Essas suítes de testes verificam se as modificações aplicadas cumprem suas respectivas funções, tomando como base seus antigos comportamentos.

Para a realização dos testes de integração, foi utilizado o ThunderClient, ferramenta que permite a execução de requisições HTTP e a verificação dos resultados. Esses testes tiveram o objetivo de avaliar o comportamento geral do sistema e seu desempenho após as modificações, analisando como principal fator o tempo de resposta.

4.3.1 Testes de Unidade

Os testes de unidade foram focados na camada *Service* da aplicação, que foi considerada a menor unidade a ser testada. O objetivo desses testes foi verificar o correto funcionamento das funções e lógicas de negócio implementadas nessa camada.

Para garantir uma boa qualidade de testes, foram estabelecidos critérios mínimos de cobertura: atingir uma cobertura de no mínimo 90% dos statements (instruções de código), 66,66% das branches (ramificações de código) e 80% das functions (funções) dentro da camada *Service*.

Os testes de unidade foram implementados utilizando a ferramenta Jest, que oferece uma estrutura completa para criação e execução de testes no Node.js. Os testes desenvolvidos podem ser encontrados no repositório da aplicação revitalizada [26]. Esses testes servem como uma base para validar as funcionalidades da camada *Service* estão corretamente implementadas.

File		Statements		Branches		Functions	
analysisService.js		100%	9/9	100%	0/0	100%	1/1
classesService.js		100%	17/17	66.66%	2/3	80%	4/5
coursesService.js		100%	11/11	66.66%	2/3	100%	6/6
graduatesService.js		100%	3/3	66.66%	2/3	100%	2/2
statisticsService.js		100%	3/3	66.66%	2/3	100%	2/2

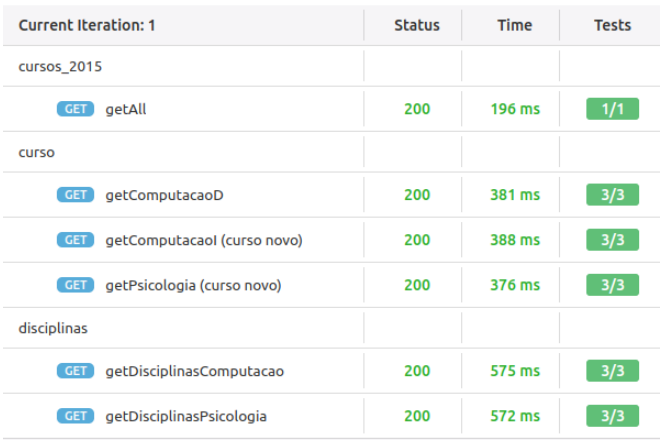
Figura 19: Resultado de testes de unidade com Jest

4.3.2 Testes de Integração

Os testes de integração foram desenvolvidos para avaliar o comportamento do sistema como um todo, verificando a interação entre seus componentes e a integração das funcionalidades implementadas. Além disso, uma análise quantitativa foi realizada para comparar os tempos de resposta da versão anterior da API com a nova implementação, a fim de comprovar as melhorias alcançadas.

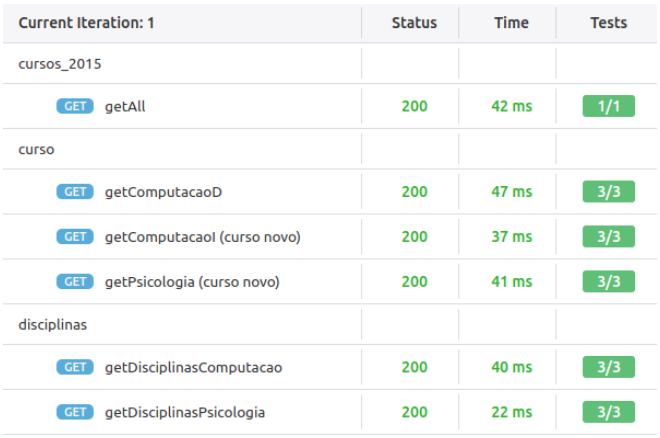
Os testes de integração incluíram medições do tempo de resposta das requisições em diferentes cenários de uso. Os resultados obtidos revelaram uma diminuição média de 84,03% no tempo de resposta das requisições em relação à versão anterior, conforme evidenciado nos trechos apresentados nas Figuras 20 e 21.

Essa redução significativa no tempo de resposta é um resultado positivo do processo de revitalização do serviço Cursos UFCG. As melhorias implementadas, como a otimização do código e a adoção de uma nova arquitetura, contribuíram para a melhoria do desempenho do sistema como um todo.



Current Iteration: 1	Status	Time	Tests
cursos_2015			
GET getAll	200	196 ms	1/1
curso			
GET getComputacaoD	200	381 ms	3/3
GET getComputacaoI (curso novo)	200	388 ms	3/3
GET getPsicologia (curso novo)	200	376 ms	3/3
disciplinas			
GET getDisciplinasComputacao	200	575 ms	3/3
GET getDisciplinasPsicologia	200	572 ms	3/3

Figura 20: Trecho de resultado de testes de com o servidor Cursos UFCG antigo (Python)



Current Iteration: 1	Status	Time	Tests
cursos_2015			
GET getAll	200	42 ms	1/1
curso			
GET getComputacaoD	200	47 ms	3/3
GET getComputacaoI (curso novo)	200	37 ms	3/3
GET getPsicologia (curso novo)	200	41 ms	3/3
disciplinas			
GET getDisciplinasComputacao	200	40 ms	3/3
GET getDisciplinasPsicologia	200	22 ms	3/3

Figura 21: Trecho de resultado de testes de com o servidor Cursos UFCG revitalizado (Node.js)

Além disso, ao considerar o tempo de execução das requisições com os dados armazenados em *cache* por meio do Redis, os resultados apresentam melhoras significativas, conforme ilustrado na Figura 22. O tempo de resposta é reduzido em uma escala de 10 a 20 vezes em comparação ao tempo sem o uso do Redis.

Current Iteration: 1	Status	Time	Tests
cursos_2015			
GET getAll	200	2 ms	1/1
curso			
GET getComputacaoD	200	5 ms	3/3
GET getComputacaoI (curso novo)	200	2 ms	3/3
GET getPsicologia (curso novo)	200	2 ms	3/3
disciplinas			
GET getDisciplinasComputacao	200	3 ms	3/3
GET getDisciplinasPsicologia	200	3 ms	3/3

Figura 22: Trecho de resultado de testes de com o servidor Cursos UFCG revitalizado (Redis)

Esses resultados validam a efetividade das mudanças realizadas durante o processo de revitalização. O desempenho aprimorado do serviço Cursos UFCG abre portas para uma experiência mais satisfatória aos usuários e fortalece a base tecnológica do projeto como um todo.

4.4 Tabela de Resolução de Problemas

Para fornecer uma visão detalhada dos problemas identificados e das soluções implementadas, foi criada a Tabela 1. Essa tabela apresenta uma listagem estruturada dos problemas encontrados no projeto antigo, indicando se eles foram solucionados ou não na nova versão, juntamente com os respectivos detalhes.

Problema no projeto antigo	Solucionado?	Detalhes
API definida em um só arquivo	Sim	Seção 4.2.1
Sem definição de versão das dependências	Sim	Seção 4.2.2
Falta de Clean Code	Sim	Seção 4.2.4
Código em mais de um idioma diferente	Parcialmente	Seção 4.2.5
Alto tempo de respostas para requisições utilizando o OpenCPU	Não	Seção 4.2.6
Falta de testes	Sim	Seção 4.3

Tabela 1: Tabela de Resolução de Problemas

5. CONCLUSÃO

O presente trabalho teve como objetivo revitalizar o serviço Cursos UFCG, buscando melhorar sua arquitetura, desempenho e manutenibilidade. Ao longo do processo, foram identificados e abordados diversos problemas, como código redundante, falta de modularidade, dificuldades na gestão do ambiente de execução e nenhuma cobertura de testes.

Através da virtualização utilizando Docker, foi possível criar um ambiente de execução isolado e replicável, facilitando a distribuição e execução do serviço em diferentes ambientes de desenvolvimento. Além disso, a implementação do Redis como *cache* permitiu reduzir a carga no banco de dados SQL, melhorando o desempenho do sistema.

A refatoração do código foi um passo importante para melhorar a legibilidade, manutenibilidade e reutilização do código. Foram aplicados princípios de Clean Code e boas práticas de programação, resultando em um código mais organizado e de fácil compreensão.

A introdução de testes automatizados foi fundamental para verificar a validade das modificações realizadas e garantir a integridade e funcionamento correto do serviço. Os testes de unidade e integração permitiram avaliar a qualidade das modificações e garantir que as funcionalidades do sistema estavam operando conforme o esperado.

Como resultado dessas melhorias, o serviço Cursos UFCG obteve uma arquitetura mais moderna, um código mais limpo e de fácil manutenção, um melhor desempenho e uma maior escalabilidade. Além disso, as boas práticas de desenvolvimento adotadas ao longo do processo contribuíram para aprimorar a qualidade do código e a reutilização de componentes.

Como trabalho futuro, sugere-se a revitalização do servidor em R para melhorar seu desempenho e continuidade da implementação de testes automatizados na aplicação principal. Com essas melhorias, será possível fortalecer ainda mais a qualidade do sistema.

Finalmente, os processos de virtualização, refatoração e revitalização do serviço Cursos UFCG trouxeram benefícios significativos, promovendo uma melhor experiência para os futuros desenvolvedores, facilitando a manutenção e a evolução do sistema, e preparando-o para enfrentar os desafios do futuro. O trabalho realizado demonstrou a importância de práticas sólidas de desenvolvimento e a utilização de tecnologias modernas na construção de aplicações robustas e eficientes.

6. AGRADECIMENTOS

A minha jornada acadêmica até aqui não foi traçada sozinha, mas não caberiam todos aqueles que gostaria de homenagear. Agradeço ao meu orientador, professor Fábio Morais, pelo seu apoio e direcionamento ao longo de todo o processo. Sua sugestão de tema e feedbacks atenciosos foram essenciais para a concepção deste trabalho.

Gostaria de agradecer à minha mãe, Drª Maria Helena Lisboa, pelo amor, juízo e confiança. Minha família e amigos de todos os cantos - os de Campina Grande em especial -, pela companhia e amizade. À minha namorada Júlia, pela paciência e carinho (principalmente em tempos de TCC). Essa vitória é nossa!

7. REFERÊNCIAS

[1] Matthew Portnoy. (2016). Virtualization Essentials. John Wiley & Sons.

[2] Docker. 2023. Docker: Accelerated, Containerized Application Development [Online]. Disponível em: https://www.docker.com/. Acesso em: 10 de maio de 2023

[3] Analytics UFCG, 2016. Cursos UFCG [Online]. Disponível em: http://analytics.ufcg.edu.br/cursosufcg/#/. Acesso em 01 de junho de 2023

[4] Manoel Veras. 2011. Virtualização: Componente central do Datacenter. Rio de Janeiro: Brasport.

[5] Docker. 2023. What is a Container? | Docker [Online]. Disponível em: https://www.docker.com/resources/what-container/. Acesso em: 10 de maio de 2023

[6] Analytics UFCG. 2023. Cursos UFCG Backend. GitHub [Online]. Disponível em: https://github.com/analytics-ufcg/cursos-ufcg-backend. Acesso em: 01 de junho de 2023.

[7] Analytics UFCG. 2023. Cursos UFCG Frontend. GitHub [Online]. Disponível em: https://github.com/analytics-ufcg/cursos-ufcg-frontend. Acesso em: 01 de junho de 2023.

[8] Python Software Foundation. 2023. Python.org [Online]. Disponível em: https://www.python.org/. Acesso em: 01 de junho de 2023.

[9] Pallets Projects. 2023. Flask - Documentação do Framework [Online] Disponível em: https://flask.palletsprojects.com/en/2.3.x/. Acesso em: 01 de junho de 2023.

[10] MySQL AB. 2023. Documentação do Connector/ODBC MySQL [Online]. Disponível em: https://dev.mysql.com/doc/connector-odbc/en/. Acesso em: 01 de junho de 2023.

[11] The R Project for Statistical Computing. 2023. Sobre o R [Online]. Disponível em: https://www.r-project.org/about.html. Acesso em: 01 de junho de 2023.

[12] Martin, R. C. 2008. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.

[13] Node.js Foundation. 2023. Node.js [Online]. Disponível em: https://nodejs.org/en/. Acesso em: 04 de junho de 2023.

[14] Express. 2023. Express.js [Online]. Disponível em: https://expressjs.com/pt-br/. Acesso em: 04 de junho de 2023.

[15] Mozilla Developer Network. 2023. JavaScript | MDN [Online]. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript. Acesso em: 04 de junho de 2023.

[16] Redis. 2023. Redis [Online]. Disponível em: https://redis.io/. Acesso em: 04 de junho de 2023.

[17] Microsoft. 2023. Visual Studio Code [Online]. Disponível em: https://code.visualstudio.com/. Acesso em: 04 de junho de 2023.

[18] ThunderClient. 2023. ThunderClient [Online]. Disponível em: https://www.thunderclient.com/. Acesso em: 04 de junho de 2023.

[19] Jest. 2023. Jest - Documentação [Online]. Disponível em: https://jestjs.io/pt-BR/. Acesso em: 04 de junho de 2023.

[20] Docker. 2023. Node - Docker Hub [Online]. Disponível em: https://hub.docker.com/_/node. Acesso em: 05 de junho de 2023.

[21] Alpine Linux. 2023. Alpine Linux [Online]. Disponível em: https://www.alpinelinux.org/. Acesso em: 05 de junho de 2023.

[22] Docker. 2023. Docker Compose - Documentação [Online]. Disponível em: https://docs.docker.com/compose/. Acesso em: 05 de junho de 2023.

[23] Joi. 2023. Joi - Biblioteca de validação de objetos em JavaScript [Online]. Disponível em: https://joi.dev/. Acesso em: 06 de junho de 2023.

[24] npm. 2023. npm [Online]. Disponível em: https://www.npmjs.com/. Acesso em: 06 de junho de 2023.

[25] Redis. 2023. Documentação do Redis para Node.js [Online]. Disponível em: https://redis.io/docs/clients/nodejs/. Acesso em: 06 de junho de 2023.

[26] Fernando Lisboa Costa. 2023. Cursos UFCG Node Backend [Online]. Disponível em: https://github.com/fernandollisboa/cursos-ufcg-node-backend. Acesso em: 06 de junho de 2023.

[27] Mauro Junior F Costa. (2022). Aplicações em Contêineres – Reduzindo a Complexidade de Configuração de Ambientes de Software. União Metropolitana de Educação e Cultura, Lauro de Freitas.