UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS SOBRAL

CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

SÉRGIO SARAIVA DE SOUSA NETO

COORDENADOR API: API REST PARA CONTROLE E GERENCIAMENTO DE SISTEMA DE AVALIAÇÃO DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E COMPUTAÇÃO

SOBRAL - CE

2023

SÉRGIO SARAIVA DE SOUSA NETO

COORDENADOR API: API REST PARA CONTROLE E GERENCIAMENTO DE SISTEMA DE AVALIAÇÃO DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E COMPUTAÇÃO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Campus Sobral da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Iális Cavalcante de Paula Júnior

SOBRAL - CE

2023

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S1c SOUSA NETO, SÉRGIO SARAIVA DE.
Coordenador api : api rest para controle e gerenciamento de sistema de avaliação do programa de pós-graduação em engenharia elétrica e computação / SÉRGIO SARAIVA DE SOUSA NETO. – 2023.
45 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Sobral, Curso de Engenharia da Computação, Sobral, 2023.
Orientação: Prof. Dr. Iális Cavalcante de Paula Júnior.

1. PPGEEC. 2. Feedbacks. 3. API. 4. Arquitetura. 5. Padrões de Projeto. I. Título.
CDD 621.39

SÉRGIO SARAIVA DE SOUSA NETO

COORDENADOR API: API REST PARA CONTROLE E GERENCIAMENTO DE SISTEMA DE AVALIAÇÃO DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E COMPUTAÇÃO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Campus Sobral da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Computação.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Iális Cavalcante de Paula Júnior (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. José Claudio do Nascimento
Universidade Federal do Ceará (UFC)

Engenheira Stefane Adna dos Santos
Universidade Federal do Ceará (UFC)

Aos meus pais, Sérgio Saraiva de Sousa Júnior e Cláudia Carneiro Girão Saraiva, por nunca deixarem faltar nada e sempre dar condições para minha evolução e crescimento.

AGRADECIMENTOS

Aos meus pais, Cláudia Carneiro Girão Saraiva e Sérgio Saraiva de Sousa Júnior, por nunca deixarem faltar nada em nossa casa, serem referência de pessoas e sempre priorizarem a educação minha e de meu irmão.

Aos meu irmão, Igor Girão Saraiva por ser, além de irmão, amigo e ajudar a descontrair nos momentos necessários.

À minha namorada Samira Soares Vieira Gomes, por ser meu ponto de conforto e tranquilidade, nos momentos que mais precisei.

Ao meu amigo Breno Campos Gomes, por ter me acompanhado durante essa caminhada na universidade.

Ao meu Professor Orientador Iális Cavalcante de Paula Júnior, pela paciência e orientações que mantiveram o trabalho em curso.

“Uma lição sem dor não tem sentido. Porque não é possível obter nada sem um sacrifício. Mas, ao suportar e superar a dor ele conseguira um coração enorme e incomparável. Sim, um coração de aço.”

(Edward Elric)

RESUMO

A coleta de *feedbacks* e a autoavaliação são extremamente necessárias para o crescimento, melhoria e tomada de decisão, seja para o indivíduo, em sua carreira profissional, por exemplo, seja para uma instituição, como no acompanhamento da satisfação de seus clientes. Com isso, o Pós-Graduação em Engenharia Elétrica e Computação (PPGEEC) decidiu implantar um sistema que tornasse possível a coleta desses *feedbacks*, para acompanhar a situação de seus docentes, discentes e a estrutura da instituição. Logo, foi desenvolvido uma *Application Programming Interface* (API) e uma interface administrativa, onde era possível definir desde os formulários, com suas perguntas e opções de resposta, até quem poderia ter acesso ao sistema. Contudo, ficou pendente o desenvolvimento da interface responsável por coletar as respostas dos usuários, bem como, após uma breve análise do *back end*, algumas melhorias da API. Com isso em mente, foi proposto a reconstrução de uma API, mais robusta, utilizando novas tecnologias, arquiteturas e padrões de projeto, que supra os requerimentos do PPGEEC, além disso, a API deve ser funcional, manutenível e segura, para isso será feito análises de qualidade de código e testes de performance.

Palavras-chave: PPGEEC. *feedbacks*. API. Arquitetura. Padrões de Projeto

ABSTRACT

The collection of feedback and self-assessment are extremely necessary for growth, improvement, and decision-making, both for individuals in their professional careers and for institutions, such as monitoring customer satisfaction. As a result, the PPGEEC decided to implement a system enabling the collection of this feedback to monitor the situation of its educators, students, and institutional structure. Consequently, a API and an administrative interface were developed, allowing the definition of forms, including their questions and response options, as well as determining access privileges within the system. However, the development of the interface responsible for gathering user responses remained pending, along with some necessary improvements to the API after a brief analysis of the back end. In light of this, a proposal was made to reconstruct a more robust API utilizing new technologies, architectures, and design patterns that fulfill the PPGEEC's requirements. Furthermore, this API must be functional, maintainable, and secure. To achieve this, thorough analyses of code quality and performance tests will be conducted.

Keywords: PPGEEC. *feedbacks*. API. Architecture. Design Patterns

LISTA DE FIGURAS

Figura 1 – Exemplo de *Command Query Responsibility Segregation* (CQRS) 22
Figura 2 – Camadas da aplicação. 28
Figura 3 – Fluxo de requisições na API 28
Figura 4 – Diagrama do banco de dados 30
Figura 5 – CQRS na aplicação 31
Figura 6 – Diagrama da arquitetura da aplicação 32
Figura 7 – Rotas de usuários 33
Figura 8 – Rotas de disciplinas 34
Figura 9 – Rotas de formulários 35
Figura 10 – Rotas de avaliação 35
Figura 11 – Rotas de turmas 36
Figura 12 – Rotas de respostas 36
Figura 13 – Requisição capturada pelo Sentry 38
Figura 14 – Resultado inicial SonarQube 40
Figura 15 – Detalhes dos problemas encontrados pelo SonarQube 40
Figura 16 – Problema detalhe pelo SonarQube 41
Figura 17 – Novo resultado de problemas de manutenibilidade 42
Figura 18 – Resultado final do SonarQube 42

LISTA DE TABELAS

Tabela 1 – Resultados dos testes de carga 43
Tabela 2 – Segundo parte dos resultados de teste de carga 43

LISTA DE ABREVIATURAS E SIGLAS

AMQP	*Advanced Message Queuing Protocol*
API	*Application Programming Interface*
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
CQRS	*Command Query Responsibility Segregation*
DDD	*Domain-Drive Design*
ECDSA	*Elliptic Curve Digital Signature Algorithm*
HMAC	*Hash-based Message Authentication Code*
HTTP	*Hyper Text Transfer Protocol*
JWT	*JSON Web Token*
MQTT	*Message Queuing Telemetry Transport*
PPGEEC	Pós-Graduação em Engenharia Elétrica e Computação
REST	*Representational State Transfer*
RPC	*Remote Procedure Call*
RSA	*Rivest-Shamir-Adleman*
SOAP	*Simple Object Access Protocol*
STOMP	*Streaming Text Oriented Messaging Protocol*
UFC	Universidade Federal do Ceará
URI	*Uniform Resource Identifier*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO 14
1.1 Justificativa 15
1.2 Objetivos 15
1.2.1 Objetivo Geral 15
1.2.2 Objetivos espeíficos 16
2 FUNDAMENTAÇÃO TEÓRICA 17
2.1 Avaliação 360 17
2.2 Plano de autoavaliação do PPGEEC 17
2.3 Engenharia de requisitos 18
2.4 API *Representational State Transfer* (REST) 18
2.5 C# (C Sharp) 19
2.6 .Net Core 20
2.7 PostgreSQL 20
2.8 GitHub 20
2.9 GitHub Actions 21
2.10 CQRS - Command Query Responsibility Segregation 21
2.11 Mediator 21
2.12 DDD - *Domain Driven Design* 22
2.13 Docker 23
2.14 Azure 23
2.15 RabbitMQ 23
2.16 JWT (JSON Web Token) 24
2.17 Sentry 24
2.18 SonarQube 24
2.19 NBomber 25
2.20 Testes de Carga 25
3 METODOLOGIA 26
3.1 Requisitos 26
3.1.1 Requisitos funcionais 26
3.1.2 Requisitos não-funcionais 27

3.2 **Implementação** . . . 27
3.2.1 ***Aplicando o Domain-Drive Design (DDD)*** . . . 27
3.2.2 ***Entidades da Aplicação*** . . . 28
3.2.3 ***Banco de dados*** . . . 29
3.2.4 ***Padrões de Projeto*** . . . 30
3.2.4.1 *Aplicando o CQRS* . . . *30*
3.2.4.2 *Aplicando o Mediator* . . . *31*
3.2.5 ***Arquitetura*** . . . 31
3.2.6 ***Rotas*** . . . 32
3.2.6.1 *Usuários* . . . *33*
3.2.6.2 *Disciplinas* . . . *34*
3.2.6.3 *Formulários, Questões e Opções* . . . *34*
3.2.6.4 *Avaliações* . . . *34*
3.2.6.5 *Turmas* . . . *35*
3.2.6.6 *Respostas* . . . *36*
3.3 **Implantação** . . . 36
3.3.1 ***Pipeline no GitHub*** . . . 36
3.3.2 ***Serviços Azure*** . . . 37
3.4 **Monitoramento** . . . 38
4 **RESULTADOS** . . . 39
4.1 **Qualidade de Código** . . . 39
4.2 **Testes de Carga** . . . 43
5 **CONCLUSÕES E TRABALHOS FUTUROS** . . . 44
6 **TRABALHOS RELACIONADOS** . . . 45
6.1 **Sistema de Autoavaliação - PPGEEC/Universidade Federal do Ceará (UFC)** . . . 45
6.2 **Sistema de Autoavaliação Web - PPGEEC/UFC** . . . 45
REFERÊNCIAS . . . 46

1 INTRODUÇÃO

Para melhor compreensão desse trabalho devemos entender que desde a implantação da pós-graduação no Brasil nos moldes definidos pelo Parecer CFE 977/1965, que constava a instauração do sistema de cursos de pós-graduação e teria o objetivo de formar professorado competente, preparar pesquisadores e treinar técnicos e trabalhadores intelectuais, tendo em vista: (i) atender a expansão do ensino superior e elevar seus níveis de qualidade, (ii) desenvolver a pesquisa científica e (iii) atender ao desenvolvimento nacional em todos setores da economia e da sociedade (BRASIL, 1965), a pós-graduação stricto sensu avançou no sentido do seu crescimento numérico (CAPES, 2019).

Passados mais de 50 anos, a pós-graduação avançou, cresceu e se desenvolveu. Entre 1998 e 2017, o número de cursos de pós-graduação stricto sensu cresceu mais de 200% e o número de mestres e doutores no país aumentou significativamente (quase 23 mil doutores e mais de 64 mil mestres titulados apenas em 2018) (GEOCAPES, 2018). Com esse crescimento, ter uma forma efetiva de avaliação externa assegura padrões básicos, o que é importante em um país continental como o nosso, mas tem suas limitações. Uma delas é o fato de não ser formativa, em que os que estão no processo se envolvam também na solução dos problemas identificados. Neste sentido, a autoavaliação favorece a construção da identidade, heterogeneidade e envolvimento dos programas avaliados, para além dos padrões mínimos garantidos pela avaliação externa. (CAPES, 2019).

O PPGEEC é o programa de pós-graduação em engenharia elétrica e computação ofertado pela Universidade Federal do Ceara. Com isso, espera se que o desenvolvimento da autoavaliação no PPGEEC induza um processo de amadurecimento de pesquisadores e discentes no sentido de corresponsabilização, colaboração e engajamento na melhoria do stricto sensu, da qualidade da formação de pesquisadores brasileiros e, principalmente, da prática democrática (LEITE, 2020).

Com isso em mente, foi iniciado o desenvolvimento de uma aplicação capaz de suprir as necessidades do programa de pós graduação. Foi feito uma API e interface capaz de definir os formulários, as perguntas feitas em cada formulário, as opções de resposta de cada pergunta, além disso, definir quem pode acessar o sistema, com diferentes níveis de acesso, estes sendo: Discente, Docente, Técnico Administrativo e Admin.

Contudo, após uma análise do código da API e da interface desenvolvida até o momento, foi encontrado pontos de melhorias e falhas. Desenvolver uma API que forneça

aos clientes que a consumirem formas de listagem, criação, edição e deleção de formulários, professores, alunos e avaliações, bem como, formas de coletar respostas de dos formulários e gerar relatórios dessas respostas. Além disso deve ser implementado um sistema de login a nível de administrador, para que possa ser feito o gerenciamento de todas as entidades da aplicação.

1.1 Justificativa

Atualmente, para a coleta desses *feedbacks*, utiliza-se a ferramenta de formulários do Google, o *Google Forms*. Contudo é um proposta generalizada, visando cobrir muitos cenários, e que não supre certas necessidades específicas dos nossos usuários. Além disso, por ser uma ferramenta de terceiros, pode ser descontinuada a qualquer momento, deixando a PPGEEC sem seus dados e formulários.

Com isso, foi iniciado o desenvolvimento de uma ferramenta própria e autoral para realizar a coleta da avaliação dos membros do mestrado. Contudo, foi desenvolvido apenas uma API e uma interface administrativa, que após uma análise, é possível identificar que o *backend* necessita de melhorias e refatorações, e que fica faltando a implementação do *frontend* responsável por coletar esses *feedbacks*.

Dito isso, este trabalho é de suma importância para a criação de uma API funcional, segura, escalável, performática e de alta manutenibilidade. E, uma vez com essa API disponível, diversos outros clientes *frontend* podem ser desenvolvidos para consumi-la e finalizar a criação do sistema de avaliação.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral desse trabalho é desenvolver uma API REST peformática, confiável e de alta disponibilidade que atenda os requisitos do PPGEEC, oferecendo uma forma de gerenciar turmas, alunos, professores e avaliações de forma conveniente e organizada. Que contemple as funcionalidades necessárias para a aplicação formulários: c) Ficha de avaliação dos discentes em relação aos docentes (disciplinas) e e) Ficha de avaliação dos docentes em relação ao desempenho dos discentes das disciplinas ministradas, descritos no Regulamento do Plano de Autoavaliação do Programa de Pós-Graduação em Engenharia Elétrica e Computação.

1.2.2 Objetivos espeíficos

Os objetivos específicos definidos visando alcançar o objetivo geral são:

- Analisar o documento que regulamenta as avaliações da pos-graduação a fim de identificar os requisitos;
- Analisar API implementada previamente para identificar defeitos e pontos de de melhoria;
- Codificar a API de gerenciamento de avaliações, turmas, alunos e professores;
- Validar a qualidade do código desenvolvido;
- Validar a disponibilidade e performance da API desenvolvida.

2 FUNDAMENTAÇÃO TEÓRICA

Essa seção apresenta conceitos e características utilizados no desenvolvimento da aplicação, assim como as ferramentas escolhidas que utilizam esses conceitos. As descrições das ferramentas são adaptações de suas documentações oficiais, disponibilizadas pelas empresas responsáveis por seu desenvolvimento.

2.1 Avaliação 360

Neste modelo de avaliação o avaliador não só avalia seus pares ou subordinados, avalia a si mesmo, bem como seus superiores na cadeia de comando da organização.

A avaliação 360 graus é conhecida também como *feedback* 360 graus, *feedback* com múltiplas fontes, avaliação em rede ou avaliação de múltipla visão, entre outros, porém o mais importante é como o modo avaliativo é aplicado, pois seus resultados serão implantados na organização (SANTOS, 2017).

2.2 Plano de autoavaliação do PPGEEC

De acordo com o artigo segundo do Regulamento do Plano de Autoavaliação do Programa de Pós-Graduação em Engenharia Elétrica e Computação o Plano de autoavaliação do PPGEEC tem como objetivos:

- Avaliar de maneira sistemática e periódica o funcionamento do PPGEEC e dotar o curso de mecanismos de autoajusts, sempre que se mostrarem necessários;
- Estimular a busca por padrões de excelência operacional, através da identificação de oportunidades e melhoramentos, dentro de um processo de busca por melhoramentos contínuos;
- Proporcionar mecanismos de sintonia com a políticas de avaliação intenra e externa da pós-graduação de instituição, definidas pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES);
- Assegurar canais de comunicação, abertos em permanência, destinado ao corpo discente, técnico-administrativo e docente do PPGEEC/UFC, para receber queixas, reclamações, sugestões, etc. (PPGEEC, 2021).

2.3 Engenharia de requisitos

Engenharia de Requisitos é o nome que se dá ao conjunto de atividades relacionadas com a descoberta, análise, especificação e manutenção dos requisitos de um sistema. O termo engenharia é usado para reforçar que essas atividades devem ser realizadas de modo sistemático, ao longo de todo o ciclo de vida de um sistema e, sempre que possível, valendo-se de técnicas bem definidas (VALENTE, 2020).

Diversas técnicas podem ser usadas para elicitação de requisitos, incluindo entrevistas com stakeholders, aplicação de questionários, leitura de documentos e formulários da organização que está contratando o sistema, realização de workshops com os usuários, implementação de protótipos e análise de cenários de uso(VALENTE, 2020).

Os requisitos de um software podem ser funcionais ou não funcionais. Os requisitos funcionais definem o que o sistema deve fazer, de maneira objetiva, enquanto os requisitos não funcionais descrevem RESTrições ou exigências que o sistema deve cumprir (SOMMERVILE, 2019).

2.4 API REST

Uma API, ou interface de programação aplicativos, é um conjunto de regras que definem como aplicativos ou dispositivos podem se conectar e se comunicar uns com os outros. Uma API REST é uma API que se adéqua aos princípios de design do REST ou o estilo de arquitetura do *Representational State Transfer*. Por esta razão, as APIs REST são muitas vezes chamadas de APIs RESTful (IBM, 2023).

Definido pela primeira em 2000 pelo cientista de computação Dr. Roy Fielding em sua dissertação de doutorado, o REST proporciona um nível relativamente alto de flexibilidade e liberdade para desenvolvedores (IBM, 2023).

Algumas APIs, como *Simple Object Access Protocol* (SOAP) ou *Extensible Markup Language* (XML)-*Remote Procedure Call* (RPC), impõe um framework RESTrito para os desenvolvedores. Porém, as APIs REST podem ser desenvolvidas usando praticamente qualquer linguagem de programação e oferecem suporte a uma variedade de formatos de dados (IBM, 2023). O único requisito é que eles devem alinhar aos seis princípios de design de REST a seguir, conhecidos como restrições de arquitetura:

- **Interface uniforme**. Todas as solicitações da API para o mesmo recurso devem ser iguais,

não importa a origem da solicitação.

- **Desacoplamento do cliente-servidor**. No projeto da API REST, os aplicativos cliente e servidor devem ser completamente independentes um do outro. A única informação que o aplicativo cliente deve receber é o *Uniform Resource Identifier* (URI) do recurso solicitado. Ele não pode interagir com o aplicativo do servidor de qualquer outra forma. Da mesma forma, um aplicativo do servidor não deve modificar o aplicativo cliente, exceto para transferi-los aos dados solicitados via *Hyper Text Transfer Protocol* (HTTP).
- **Sem estado definido**. As APIs REST não possuem estado definido, o que significa que cada solicitação precisa incluir todas as informações necessárias para processá-lo. Em outras palavras, as APIs REST não requerem nenhuma sessão do lado do servidor.
- **Capacidade de armazenamento em cache**. Quando possível e necessário, os recursos devem ser armazenados em cache pelo cliente ou servidor.
- **Arquitetura de sistema em camadas**. Em APIs REST, as chamadas e respostas passam por diferentes camadas. Pode haver uma série de intermediários diferentes no *loop* de comunicação.
- **Código sob demanda**. Opcionalmente, as APIs REST geralmente enviam recursos estáticos, mas em certos casos, as respostas também podem conter código executável (como *applets* Java). Nestes casos, o código deve ser executado somente sob demanda.

2.5 C# (C Sharp)

O C# é uma linguagem de programação moderna, orientada a objeto e fortemente tipada. O C# permite que os desenvolvedores criem muitos tipos de aplicativos seguros e robustos que são executados no .NET (BILLWAGNER, 2023).

Vários recursos do C# ajudam a criar aplicativos robustos e duráveis. A coleta de lixo recupera automaticamente a memória ocupada por objetos não utilizados inacessíveis. Tipos anuláveis são protegidos contra variáveis que não se referem a objetos alocados. O tratamento de exceções fornece uma abordagem estruturada e extensível para detecção e recuperação de erros. As expressões Lambda dão suporte a técnicas de programação funcional. Consulta Integrada à Linguagem (LINQ) a sintaxe cria um padrão comum para trabalhar com dados de qualquer fonte. O suporte à linguagem para operações assíncronas fornece sintaxe para a criação de sistemas distribuídos (BILLWAGNER, 2023).

Esta foi a linguagem escolhida para o desenvolvimento da API por conta da familia-

ridade do autor com a ferramenta, além de ser uma linguagem robusta e fornecer funcionalidades os suficientes para o desenvolvimento da API.

2.6 .Net Core

O ASP.NET Core é a versão de código aberto do ASP.NET, que executa no macOS, Linux e Windows. O ASP.NET Core foi lançado pela primeira vez em 2016 e é um novo design de versões anteriores somente para Windows do ASP.NET (MICROSOFT, 2023a).

Além da familiaridade com o *framework*, o .NET Core foi escolhido como plataforma de desenvolvimento, pois pode ser executado em diversos sistemas operacionais, dando, assim, maior flexibilidade de escolha no momento que for fazer o *deploy* da aplicação, por exemplo.

2.7 PostgreSQL

O PostgreSQL é um banco de dados relacional de software livre com suporte de 30 anos de desenvolvimento, sendo um dos bancos de dados relacionais mais estabelecidos disponíveis. Como o PostgreSQL é robusto, seguro e extensível — e porque possui um rico ecossistema de ferramentas disponíveis — os desenvolvedores usam o PostgreSQL para uma variedade de casos de uso. O software foi projetado para ser compatível com todos os principais sistemas operacionais incluindo Linux, Windows e Macintosh, além de oferecer suporte a texto, imagens, sons e vídeos, sendo assim um banco de dados popular para pessoas e empresas com necessidades diversas (AZURE, 2023).

Por ser um banco de dados de software livre e possuir suporte a todas as operações necessárias, esse foi o sistema de banco de dados escolhido para o desenvolvimento da API.

2.8 GitHub

GitHub é uma plataforma de hospedagem de código para controle de versão e colaboração. Permite que você e outras pessoas trabalhem em conjunto em projetos de qualquer lugar (GITHUB, 2023c).

2.9 GitHub Actions

GitHub Actions é uma plataforma de integração contínua e entrega contínua (CI/CD) que permite automatizar a sua compilação, testar e pipeline de implantação. É possível criar fluxos de trabalho que criam e testam cada pull request no seu repositório, ou implantar pull requests mesclados em produção.

Ademais, utilizamos o GitHub Student Pack, para poder ter acesso a ferramentas que, sem isso, teriam que ser pagas para poder utilizar (GITHUB, 2023a).

O GitHub Student Developer Pack é um conjunto de ferramentas e recursos gratuitos oferecidos pelo GitHub para estudantes universitários que inclui acesso a serviços de desenvolvimento de software, como hospedagem em nuvem, ferramentas de desenvolvimento, serviços de gerenciamento de projetos e muito mais (GITHUB, 2023b).

2.10 CQRS - Command Query Responsibility Segregation

O CQRS significa Segregação de Responsabilidade de Comando e Consulta, um padrão que separa as operações de leitura e atualização de um armazenamento de dados. A implementação do CQRS em seu aplicativo pode maximizar o desempenho, a escalabilidade e a segurança. A flexibilidade criada pela migração para CQRS permite ao sistema evoluir melhor ao longo do tempo e impede que os comandos de atualização causem conflitos de mesclagem no nível de domínio (MICROSOFT, 2023b).

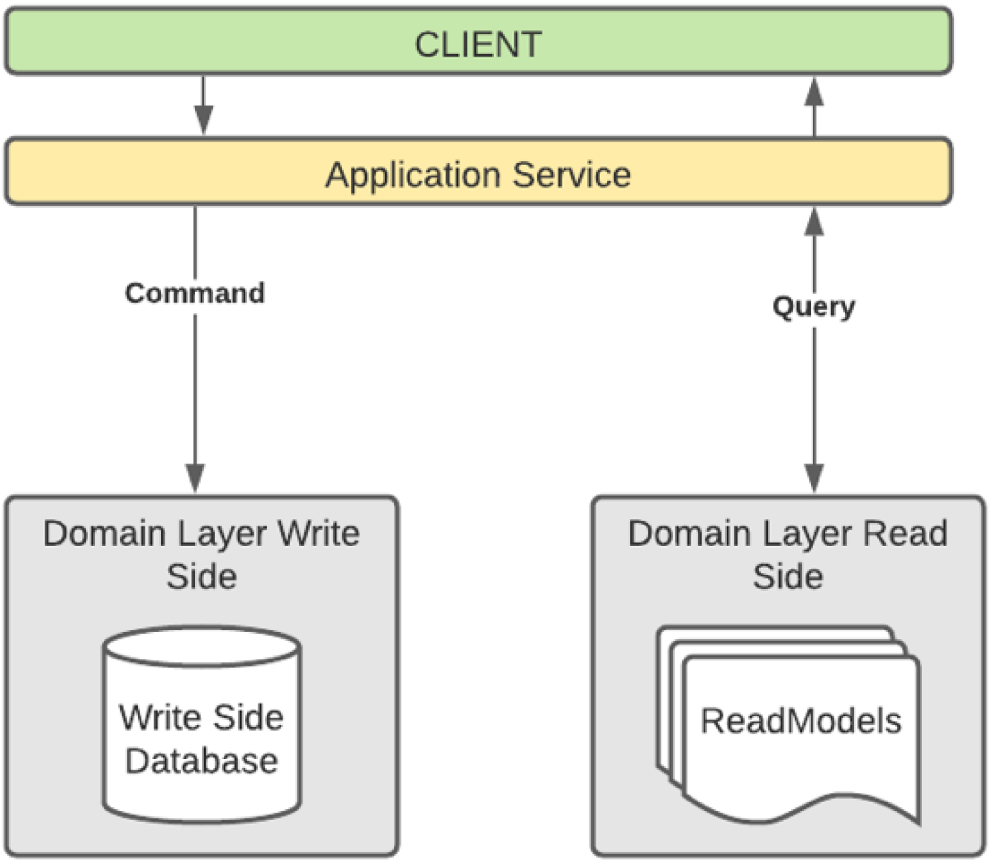
Para maior isolamento, você pode separar fisicamente os dados de leitura de dados de gravação. Nesse caso, o banco de dados de leitura pode usar seu próprio esquema de dados, otimizado para consultas (MICROSOFT, 2023b). Como podemos ver na Figura 1, que exemplifica um fluxo CQRS.

2.11 Mediator

O Mediator é um padrão de projeto comportamental que permite que você reduza as dependências caóticas entre objetos. O padrão restringe comunicações diretas entre objetos e os força a colaborar apenas através do objeto mediador (SHVETS, 2021).

O padrão Mediator sugere que você deveria cessar toda comunicação direta entre componentes que você quer tornar independentes um do outro. Ao invés disso, esses componentes devem colaborar indiretamente, chamando um objeto mediador especial que redireciona as

Figura 1 – Exemplo de CQRS



Fonte: Bolic (2021).

chamadas para os componentes apropriados. Como resultado, os componentes dependem apenas de uma única classe mediadora ao invés de serem acoplados a dúzias de outros colegas (SHVETS, 2021).

Então para cada comando ou consulta da aplicação existe um *handler* que vai executar de fato as regras de negócio, chamadas a banco de dados e serviços externos.

2.12 DDD - *Domain Driven Design*

Para modelarmos e abstrairmos o nosso sistema utilizamos o DDD, um conjunto de princípios para projeto de software, organizados e sistematizados em 2003, por Eric Evans, em um livro com o mesmo nome. Os princípios defendidos por DDD têm, no seu conjunto, um objetivo central: permitir o desenvolvimento de sistemas cujo design é centrado em conceitos próximos e alinhados com um domínio de negócio. Portanto, o design do sistema deve ser norteado para atender ao seu domínio. E não, por exemplo, para se moldar a uma determinada

tecnologia de programação (EVANS, 2016).

DDD defende que a separação entre domínio e tecnologia deve ser promovida e expressa na arquitetura do sistema. Para tanto, padrões como Arquitetura em Camadas, Arquitetura Limpa ou Arquitetura Hexagonal podem ser usados (VALENTE, 2020).

2.13 Docker

O Docker é um projeto de software livre para automatizar a implantação de aplicativos como contêineres autossuficientes portáteis que podem ser executados na nuvem ou localmente. Contêineres do Docker podem executar em qualquer lugar, localmente no *datacenter* do cliente, em um provedor de serviços externo ou na nuvem, no Azure. Os contêineres de imagem do Docker podem ser executados nativamente no Linux e no Windows (MONTEMAGNO, 2023).

2.14 Azure

O Azure é a plataforma de nuvem pública da Microsoft. O Azure oferece uma ampla coleção de serviços, incluindo plataforma como serviço (PaaS), infraestrutura como serviço (IaaS), banco de dados como serviço (DBaaS) e funcionalidades de serviços de bancos de dados gerenciados (EKUAN, 2023). O Azure, como outras plataformas de nuvem, se baseia em uma tecnologia conhecida como virtualização.

2.15 RabbitMQ

RabbitMQ é uma aplicação open source para troca de mensagens desenvolvida em linguagem Erlang, funcionando como uma camada intermediária entre aplicações, um tradutor de uma mensagem em uma linguagem específica para padrões globais de comunicação que podem ser transmitidos e recebidos em aplicações em outra linguagem. Construído no topo do protocolo *Advanced Message Queuing Protocol* (AMQP), mas também compatível com outros protocolos como HTTP,*Message Queuing Telemetry Transport* (MQTT) e *Streaming Text Oriented Messaging Protocol* (STOMP), o serviço provê resiliência, escalabilidade, roteamento flexível, capacidade de clusterização de nós fisicamente distantes de forma lógica, modelo de federação de servidores, sistema de tracing, suporte multi-clientes e interface amigável para gerenciamento das aplicações (NEVES, 2014).

2.16 JWT (JSON Web Token)

JSON Web Token (JWT) é um padrão aberto (RFC 7519) que define uma maneira compacta e independente de transmitir informações com segurança entre as partes como um objeto JSON. Essas informações podem ser verificadas e confiáveis porque são assinadas digitalmente. Os JWTs podem ser assinados usando um segredo (com o algoritmo *Hash-based Message Authentication Code* (HMAC)) ou um par de chaves pública/privada usando *Rivest-Shamir-Adleman* (RSA) ou *Elliptic Curve Digital Signature Algorithm* (ECDSA) (JWT.IO, 2023).

2.17 Sentry

Sentry é uma ferramenta de monitoramento de software que ajuda os desenvolvedores a identificar e corrigir problemas relacionados ao código. Do rastreamento de erros ao monitoramento de desempenho, o Sentry fornece observabilidade em nível de código que facilita o diagnóstico de problemas e o aprendizado contínuo sobre a integridade do código do seu aplicativo. Utilizaremos essa ferramenta pra auxiliar na observabilidade da aplicação (SENTRY, 2023).

2.18 SonarQube

SonarQube é uma ferramenta de revisão de código automática e autogerenciada que ajuda sistematicamente a fornecer código limpo. O SonarQube integra-se ao fluxo de trabalho existente e detecta problemas no código ajudanto a realizar inspeções contínuas de código nos nossos projetos. O produto analisa mais de 30 linguagens de programação diferentes e integra-se à pipeline de integração contínua (CI) de plataformas DevOps para garantir que seu código atenda aos padrões de alta qualidade (DOCS, 2023b).

Utilizaremos o SonarQube como ferramenta de analise de código da nossa API, esta ferramenta é capaz de fornecer diversas métricas, como vulnerabilidades, linhas duplicadas e falhas de segurança.

2.19 NBomber

NBomber é um *framework* de teste de carga moderna e flexível, projetada para testar qualquer sistema independentemente de um protocolo (HTTP/WebSockets/AMQP, etc). Usando o NBomber, podemos testar a confiabilidade e o desempenho de nossos sistemas e detectar regressões e problemas de desempenho mais cedo (DOCS, 2023a).

2.20 Testes de Carga

O teste de carga de API é o processo de testar o desempenho e a escalabilidade de uma API sob uma carga pesada simulada (LOADVIEW, 2023). Isso é feito para garantir que a API possa lidar com o tráfego esperado e fornecer desempenho consistente e confiável para seus usuários. Para uma preparação abrangente, as equipes devem testar o sistema em diferentes tipos de testes (LABS, 2023). Os principais tipos de testes são os seguintes:

- ***Smoke tests***. Ou testes de fumaça, validam se o código funcionam de forma esperda sobre carga mínima.
- ***Average-load test***. Ou testes de carga média, verificam como o sistema se comporta sobre condições normais de carga.
- ***Stress tests***. Teste de estresse, medem como um sistema performa no limite, quando a carga excede o esperado.
- ***Soak tests***. Testes de absorção, aferem a confiabilidade e performance do sistema sobre longo periodos de tempo.
- ***Spike tests***. Também chamados de teste de pico, checa o comportamento e a sobrenvivencia do sistema em casos de cargas massivas, repentinas e de curta duração.
- ***Breakpoint tests***. Conhecidos também como testes de interrupção, identificam a capacidade limite do sistema, aumentando a carga gradualmente.

3 METODOLOGIA

Nesta seção descreveremos como utilizamos as tecnologias escolhidas para a construção da API, descrevendo, passo a passo, desde a especificação de requisitos, a arquitetura e design da solução, a sua implementação e, por fim, a seu processo de *deploy*. Facilitando, assim, a futura manutenção, alteração ou melhoria da API por outros desenvolvedores.

3.1 Requisitos

Analisando o Regulamento do Plano de Autoavaliação do Programa de Pós-Graduação em Engenharia Elétrica e Computação traçamos, principalmente, os requisitos funcionais que a API deve atender. Os requisitos não-funcionais foram elencados para dar à API a capacidade de monitoramento, segurança, qualidade, avaliado por ferramentas e métricas amplamente utilizadas no meio empresarial.

3.1.1 Requisitos funcionais

O objetivo principal do trabalho é criar uma API que permita à coordenação da pós-graduação o gerenciamento e criação de alunos, professores, disciplinas, turmas, formulários e avaliações. E, para alunos e professores, acessar avaliações e responde-las.

Assim, a API deve permitir, para a coordenação:

- Login a nível administrativo;
- Gerenciamento de turmas;
- Gerenciamento de alunos;
- Gerenciamento de professores;
- Gerenciamento de disciplinas;
- Gerenciamento de formulários;
- Gerenciamento de avaliações;
- Envio de e-mail com link de acesso da avaliação para os usuários;
- Gerar relatório de respostas por avaliação.

Enquanto para os estudantes e professores, que responderão avaliações, a API deve permitir:

- Acesso à avaliação por meio do link, sem necessidade de senha;
- Listar formulário da avaliação;

- Responder formulário da avaliação.

3.1.2 Requisitos não-funcionais

Para os requisitos não funcionais, foram levados em consideração aspectos de segurança, qualidade e monitoramento.

Para segurança, devemos garantir que o gerenciamento das entidades do sistema só possa ser feito por administradores e, para isso, utilizamos login com email e senha, retornando, se bem sucedido um token JWT, que será utilizado em todas as outras rotas para poder ser feita a autenticação e autorização. No lado de professores e alunos, também mantemos a segurança enviando um email com um token único de acesso, ao informar esse token à API, também retornamos um token JWT, que também deve ser utilizado para autenticação e autorização nas outras rotas.

Para garantir a qualidade utilizaremos uma ferramenta chamada SonarQube, que irá analisar o código da API e aferir diversas métricas importantes para manter a qualidade do código desenvolvido, como vulnerabilidades, linhas duplicadas e etc.

Sobre monitoramento iremos utilizar o Sentry, para gerar *logs* (registros) de todas as requisições que são feitas para a API e capturar todos os erros que a aplicação encontrar, facilitando, assim, uma futura depuração.

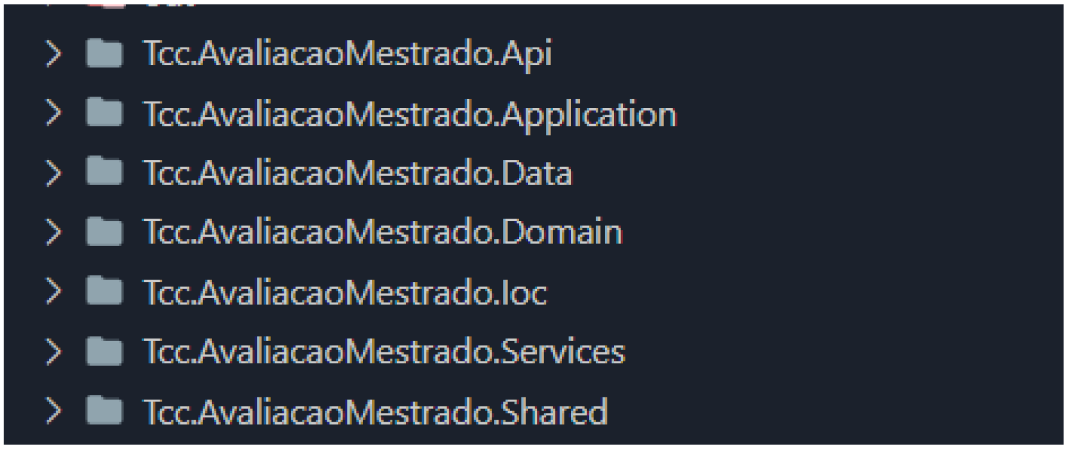
3.2 Implementação

3.2.1 Aplicando o DDD

Como baseamos o desenvolvimento da aplicação no DDD, utilizaremos a Arquitetura em Camadas para poder estruturar o nosso projeto. Na Figura 2 vemos que as pastas do projeto estão organizadas de acordo com as camadas e, na Figura 3, podemos observar o fluxo das requisições que chegam a API obedecem.

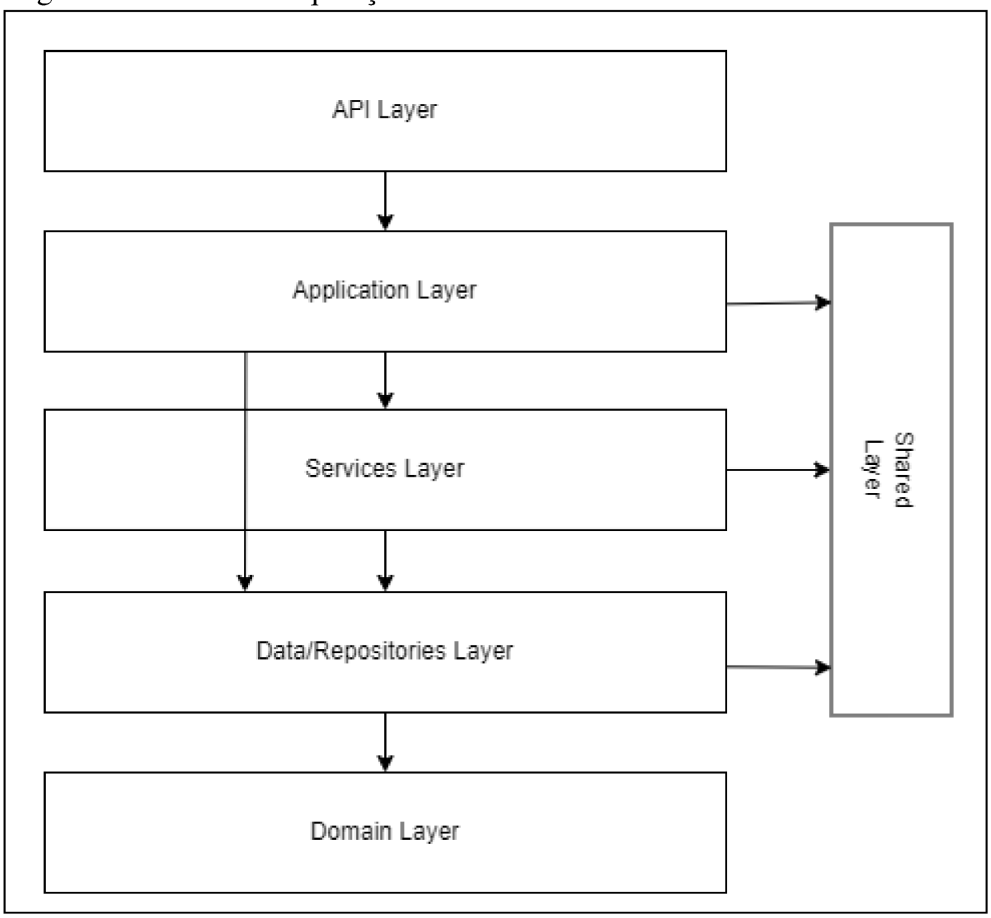
Cada camada possui uma única responsabilidade e forma uma abstração em volta dela, além disso as barreiras entre elas faz com que uma camada não se preocupe com a outra, ou seja, a camada de API, ou apresentação, não se preocupa em como a *Application Layer* se comporta.

Figura 2 – Camadas da aplicação.



Fonte: elaborado pelo autor (2023).

Figura 3 – Fluxo de requisições na API



Fonte: elaborado pelo autor (2023).

3.2.2 Entidades da Aplicação

No DDD, uma entidade é um objeto que possui uma identidade única, que o distingue dos demais objetos da mesma classe. Por exemplo, cada Usuário da nossa API é uma entidade, cujo identificador é o seu id do banco de dados, por exemplo (VALENTE, 2020).

Analisando o Regulamento do Plano de Autoavaliação do Programa de Pós-Graduação

em Engenharia Elétrica e Computação pudemos encontrar as seguintes entidades:

- Usuário;
- Docente;
- Discente;
- Admin;
- Turma;
- Disciplina;
- Avaliação;
- Formulário.

Cada entidade dessa será abstraída no banco de dados, que mostraremos a seguir, bem como no domínio da aplicação.

3.2.3 Banco de dados

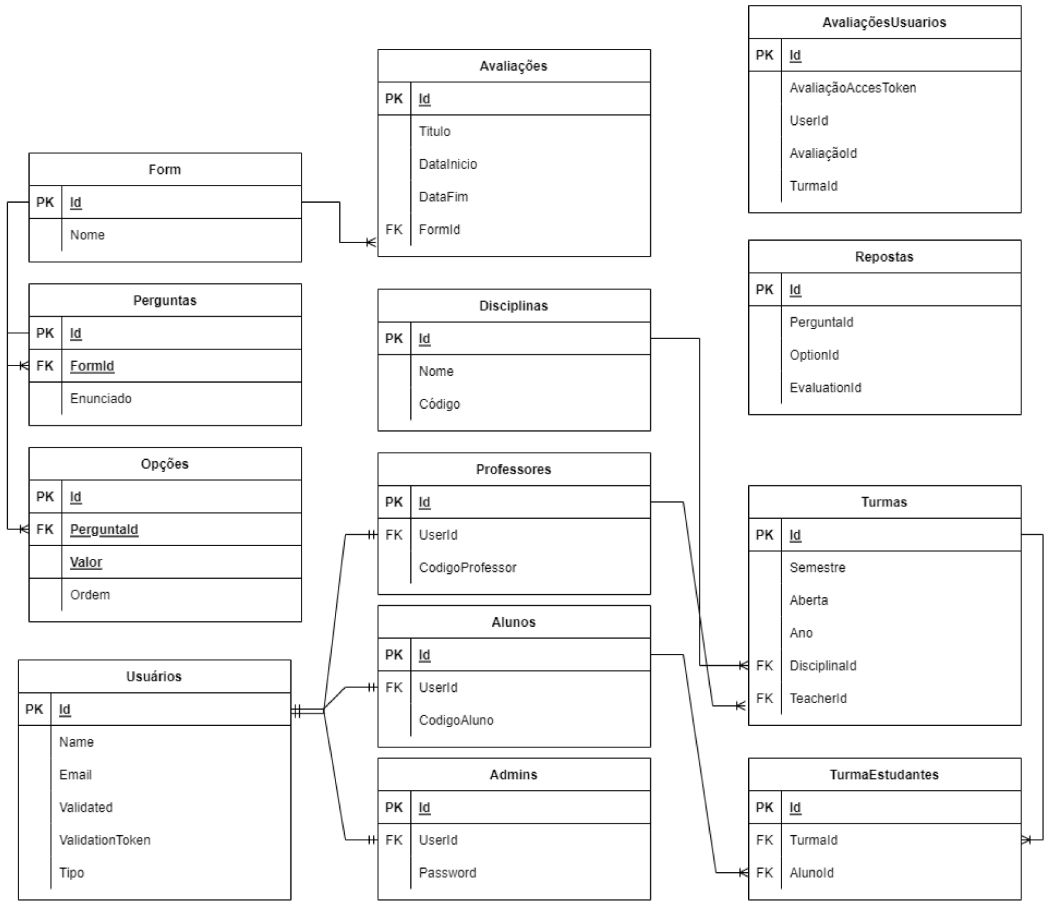
Para o banco de dados, foi necessário abstrair cada entidade do domínio da aplicação. Com isso em mente, criamos tabela para Usuários, temos três tipos de usuários, Docente, Discente e Admin. Foram criados mais três tabelas para cada tipo de usuário, Docente e Discente, possuem códigos de registro e Admin possui uma senha. Criamos uma tabela para Disciplinas e a tabela Turma relaciona Disciplina, Professor por meio de chave estrangeira e se relaciona com alunos usando uma tabela pivô.

Criamos uma tabela para Formulário, e mais duas tabelas para Perguntas e Opções de uma Pergunta. Na tabela de Avaliação existe a chave de estrangeira de formulário, e para relacionar Turma, Usuários e Avaliações, criamos uma nova Tabela que recebe Id de Turma, Usuário e Avaliação.

Por fim, para termos registro de Respostas, criamos outra tabela que recebe o id da Avaliação, da Opção escolhida e da Pergunta. Dessa forma conseguimos gerenciar todas as entidades necessárias e gerar os relatório que queremos.

Na Figura 4 temos o diagrama ER do banco de dados:

Figura 4 – Diagrama do banco de dados



Fonte: elaborado pelo autor (2023).

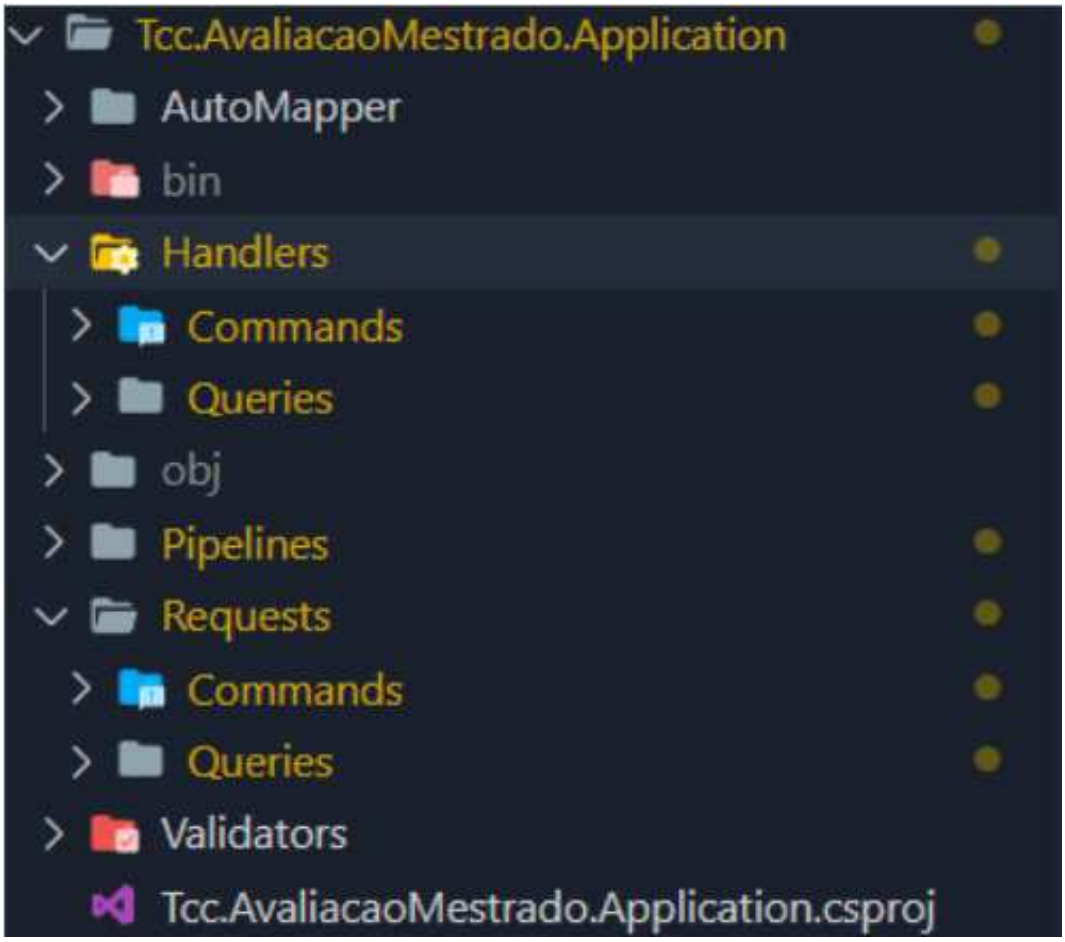
3.2.4 Padrões de Projeto

3.2.4.1 Aplicando o CQRS

Criamos, seguindo os conceitos do CQRS, para cada requisição que altera o banco de dados um comando, e para cada requisição que lê do banco de dados sua respectiva consulta. Como podemos ver na Figura 5, os comandos são os *commands* e consultas são as *queries*, separamos as consultas dos comandos.

Não chegamos a utilizar banco de dados separados para leitura e escrita de dados, visto que nosso sistema não tem perspectiva de tantos usuários assim, contudo, a nível de código, as requisições são separadas em comandos e consultas, tornando assim o código mais organizado e de fácil manutenibilidade. Não separamos o banco de dados, também, para não aumentar tanto a complexidade do sistema.

Figura 5 – CQRS na aplicação



Fonte: elaborado pelo autor (2023).

3.2.4.2 Aplicando o Mediator

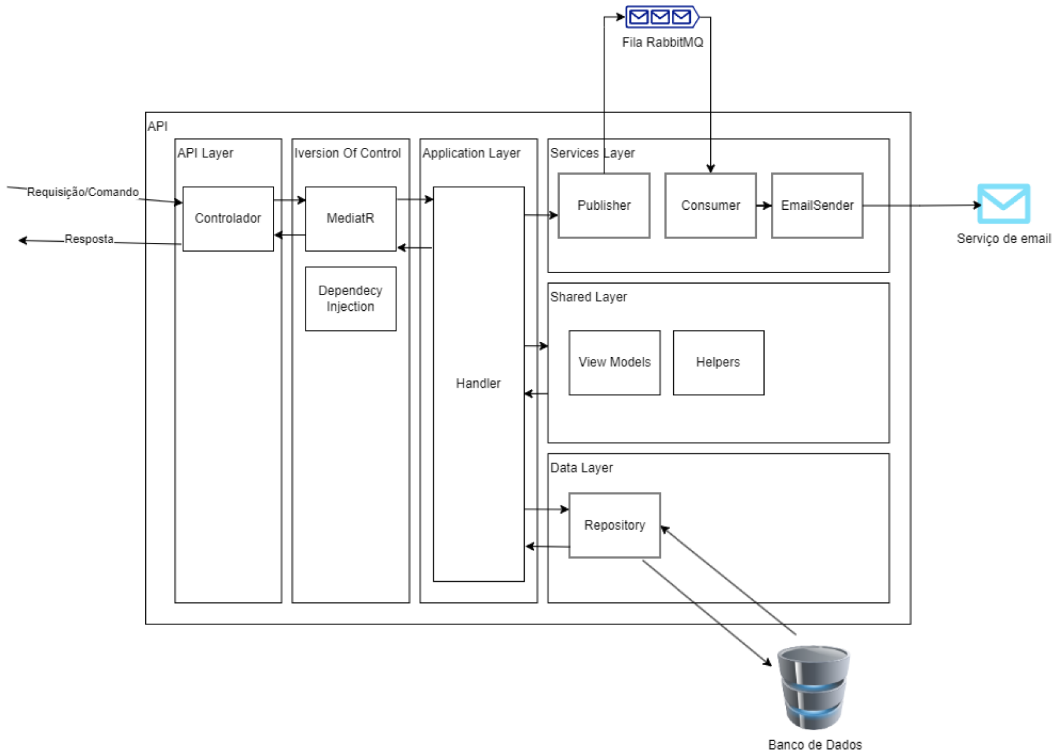
Como utilizamos o padrão Mediator devemos criar para cada comando ou consulta da aplicação, seu respectivo *handler*. No *handler* que contém toda a lígica e regras de negócio, chamadas para banco de dados e serviços externos. Podemos ver isso na Figura 5, também.

3.2.5 Arquitetura

Como foi dito anteriormente, utilizamos a Arquitetura de Camadas para estruturar internamente a aplicação, agora iremos mostrar como a solução é estruturada como um todo, qual é o ciclo de vida de uma requisição dentro da API. Utilizaremos o diagrama da Figura 6 para explicar melhor.

A requisição primeiro passa pelo controlador da aplicação, responsável por receber e retornar os dados da API, após receber a requsição o controlador transforma essa requisição em um comando ou em uma consulta. Utilizando o padrão *Mediator* com o auxílio da biblioteca

Figura 6 – Diagrama da arquitetura da aplicação



Fonte: elaborado pelo autor (2023).

MediatR enviamos o comando para o seu respectivo *handler*.

O handler então irá executar as regras de negócio necessárias para validar os dados recebidos do usuário, a fim de manter a integridade e segurança dos dados. Após isso, com o repositório da camada de dados irá escrever ou ler dados no banco de dados. Se necessário for, irá enfileirar um evento de e-mail na fila do *RabbitMQ*, fazemos isso pois envio de e-mail é um processo demorado e, se ficarmos aguardando o envio do mesmo, pode deixar a API mais lenta aos olhos do usuário. Ou seja, inserimos na fila e tratamos isso em segundo plano.

Por fim, retornamos ao usuário um *ViewModel*, se tudo tiver corrido bem, que representa os dados do banco de dados. Entretanto se tiver ocorrido alguma falha retornamos códigos HTTP, 400 para dados inválidos, 401 para não autenticado, 403 para não autorizado e 500 para erro interno.

3.2.6 Rotas

Para podermos consumir os recursos da API precisamos fazer requisições HTTP para cada recurso de aplicação. O usuário pode criar recursos, edita-los, lista-los e deleta-los, além de outras funcionalidades específicas. A documentação de cada rota e recurso pode ser acessado através do url 'api.avaliandoppgeec.live/swagger'. Utilizamos as Figuras 7, 8, 9, 10, 11

e 12 para ilustrar o *Swagger* da API.

3.2.6.1 Usuários

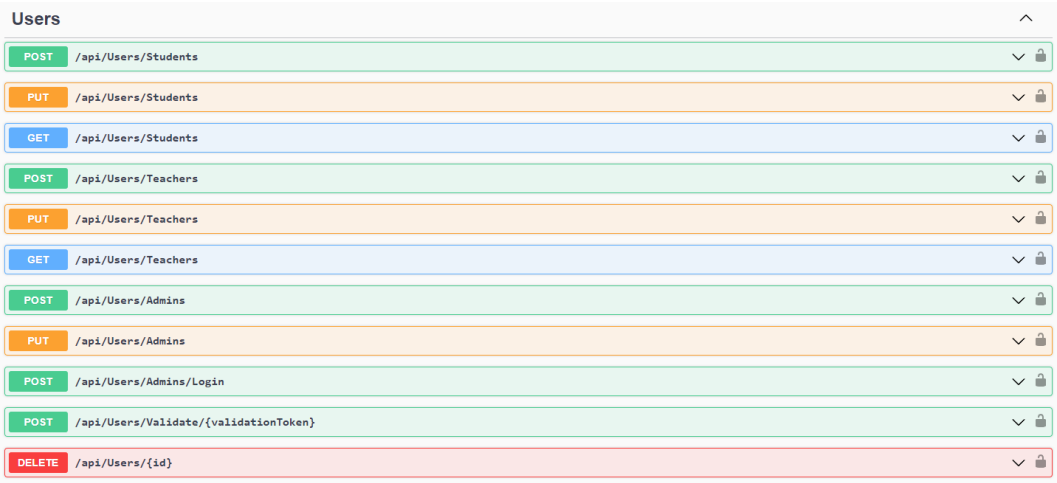
Nas rotas de usuários, nós podemos criar usuários de cada tipo, Estudante, Professor e Administrador, além de possuir rotas para login de Administrador, validação de e-mail e deleção de usuário.

Sempre que um usuário é criado, um e-mail de validação é enviado, apenas usuários validados podem ser gerenciados no sistema, por exemplo, um administrador só consegue fazer login se tiver validado e, professores e estudantes só podem ser vinculados a uma turma se estiverem validados, também.

Só pode existir um usuário com um e-email específico e só pode existir estudantes com um só código e professores também. Ao atualizar um usuário, verifica-se se o e-mail foi atualizado, se sim, envia-se novo e-email de validação para aquele usuário. Verificamos, também, se novo e-email e código informados estão sendo utilizados por outros usuários, para manter a integridade dos dados.

Todas as rotas são protegidas por autenticação e autorização, as quais só podem ser consumidas por administradores, exceto a rota de login e rota de validação de token, que são públicas.

Figura 7 – Rotas de usuários

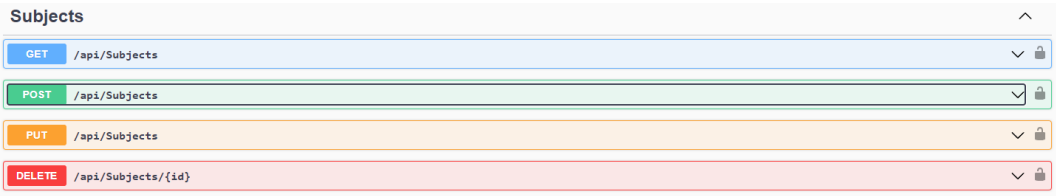


Fonte: elaborado pelo autor (2023).

3.2.6.2 Disciplinas

Para as rotas de disciplinas o usuário pode lista-las, cria-las, edita-las e deleta-las. Para criação e edição fazemos apenas a verificação do código da disciplina, para que não haja códigos duplicados. Estas rotas só podem ser acessadas por administradores.

Figura 8 – Rotas de disciplinas



Fonte: elaborado pelo autor (2023).

3.2.6.3 Formulários, Questões e Opções

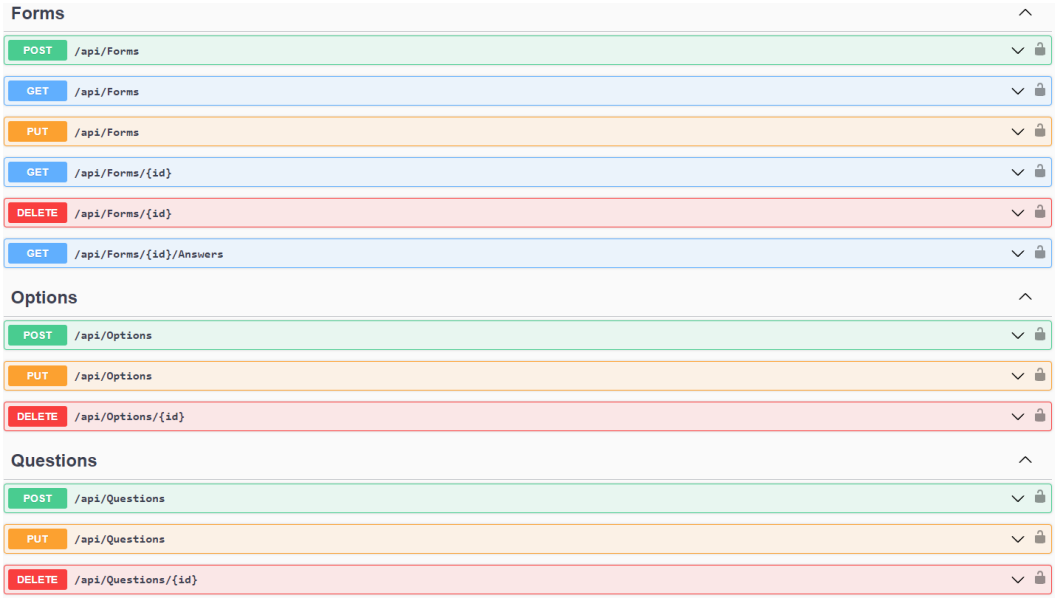
Na criação de Formulário é requisitado apenas o nome do formulário, contudo, no banco de dados temos um propriedade que só considera o formulário criado quando vinculamos alguma Pergunta a ele. Para vincularmos uma Pergunta a um Formulário precisamos enviar no momento da criação da pergunta o id do formulário que queremos que aquela pergunta faça parte. Devemos fazer o mesmo para as Opções de uma pergunta, informamos no momento de sua criação o id da pergunta que queremos vincula-la.

Todas as operações de leitura, escrita, edição e deleção estão disponíveis para formulários, contudo, não há possibilidade de listar Opções e Pergunta diretamente, pois não faz sentido realizar essa consulta avulsa. Para recuperar um formulário por completo, com opções e perguntas, é necessário utilizar a rota que retorna o formulário por id. Todas as rotas de formulário são protegidas por autenticação e autorização, apenas administradores podem realizar estas operações, exceto a rota de retornar formulário por id, nesta rota é permitido que Alunos e Professores façam requisições também.

3.2.6.4 Avaliações

Para avaliações existe validação apenas para as datas de inicio e fim de avaliação, a data de inicio e de fim não pode ser menor que a data atual e, a data de inicio não pode ser maior ou igual que a data de fim. Além disso, validamos, também, se o formulário vinculado àquela avaliação existe. Podemos fazer requisições de leitura, criação, edição e deleção. Apenas

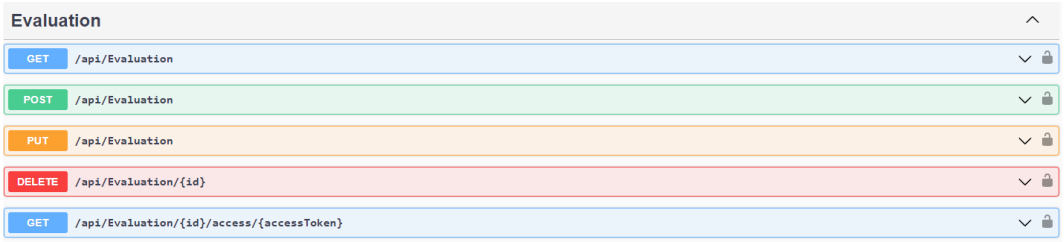
Figura 9 – Rotas de formulários



Fonte: elaborado pelo autor (2023).

administradores tem acesso a essas rotas.

Figura 10 – Rotas de avaliação



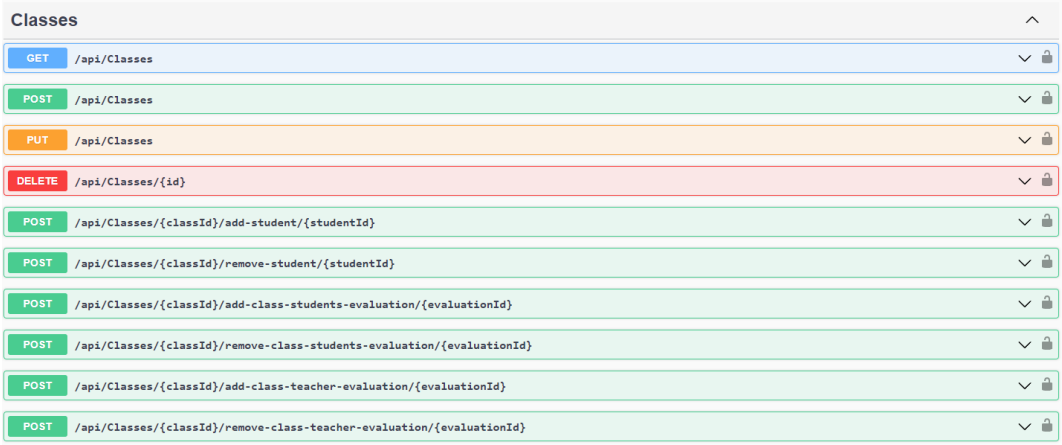
Fonte: elaborado pelo autor (2023).

3.2.6.5 Turmas

Nas turmas validamos se o id da disciplina e do professor existem e, se o código é único. Aqui, além de ser possível deletar, listar, atualizar e criar turmas, é possível, também, associar estudantes com uma turma e, associar uma avaliação aos estudantes daquela turma ou ao professor, bastando, somente, informar o id da avaliação e o id da turma que se deseja avaliar. A partir daí, a API vai buscar os estudantes daquela turma e associar à avaliação.

Os estudantes e o professor de uma turma só podem possuir uma avaliação para responder. Estas rotas só são acessíveis pelo administrador do sistema.

Figura 11 – Rotas de turmas

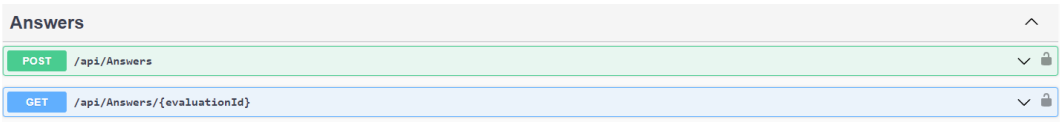


Fonte: elaborado pelo autor (2023).

3.2.6.6 Respostas

Para as respostas possuímos apenas duas rotas, uma para criar repostas, acessível apenas por estudantes e professores que possuem o link com o token de acesso recebido por e-mail, e outra para consultar respostas de uma avaliação, acessível apenas por administradores, informando apenas o id de uma avaliação. Só pode ser consultado as repostas de uma avaliação que tenha sido finalizada. As respostas são anônimas, portanto não capturamos o id do aluno.

Figura 12 – Rotas de respostas



Fonte: elaborado pelo autor (2023).

3.3 Implantação

Visto que se trata de uma API Web, precisamos preparar servidores para disponibilizar o serviço aos usuários. Para isso, utilizamos dois serviços, o GitHub Actions, para realizar o *deploy* automatizado e contínuo do código desenvolvido, e a Azure como *host* da API de fato.

3.3.1 Pipeline no GitHub

Além de ser utilizado para armazenar e versionar o código da aplicação, utilizamos o github para a construção da nossa *pipeline* de *deploy* automatizado, por meio da ferramenta GitHub Actions.

Nossa pipeline de *deploy* automatizado possui três estágios, cada um com uma funcionalidade especifica. Ela é executada a cada *push* que fazemos no repositório do GitHub.

No primeiro estágio realizamos a verificação da *build* da nossa aplicação, assim temos a segurança de que ela vai ser compilada e executada com sucesso e, que esse erro não vai chegar ao nosso ambiente de produção ou desenvolvimento.

Já no segundo estágio rodamos as migrações do banco de dados, para isso utilizamos o *Docker* para montar uma imagem com as migrações, publicamos essa imagem no nosso repositório de contêineres da Azure, baixamos a nova imagem no WebApp da Azure e executamos as migrações. Assim, as mudanças que realizamos localmente na estrutura do banco de dados é replicado para o banco de dados remoto.

Por fim, no terceiro estágio, ainda utilizando o Docker, montamos uma imagem com o código da nossa API, fazemos o mesmo processo de publicar a imagem e baixar a imagem na Azure e, dessa forma as alterações de código já estão publicadas.

3.3.2 Serviços Azure

Dentre os diversos serviços disponibilizados pela Azure, foram utilizados, para a hospedagem da API o serviço *WebApp* da Azure, no plano gratuito para a instancia de desenvolvimento, utilizamos uma máquina com 1 núcleo virtual de CPU, 1.75GBs de memória ram, 10GBS de armazenamento e sistema operacional Linux.

O banco de dados é hospedado usando o *Azure Database for PostgreSQL* com 32Gbs de armazenamento, não há necessidade de pagamento extra, visto que o Sistema Gerenciador de Banco de Dados é gratuito.

Já a fila *RabbitMQ* usamos uma máquina virtual para a hospedagem com uma imagem do *RabbitMQ*. A máquina possui 1 core de CPU e 500MB de memória ram.

Para armazenar e baixar as imagens Docker que criamos, utilizamos o serviço *Container registry*. O envio de e-mails é feito por meio do *Email Communication Service*.

Por fim, quanto ao endereço da aplicação, escolhemos o 'api.avaliandoppgeec.live'. O domínio foi adquirido por meio do *GitHub Student Pack* em colaboração com a UFC.

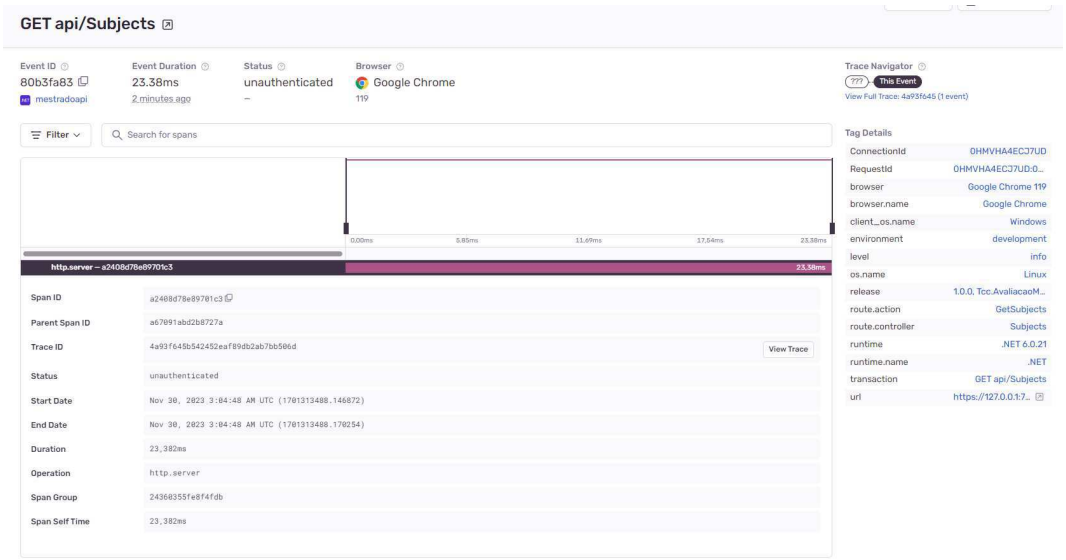
3.4 Monitoramento

Para dar a API observabilidade e maiores capacidades de monitoramento, o mercado possui diversas ferramentas que auxiliam nesse propósito, a escolhida foi o *Sentry* por familiaridade do autor e ser fornecida de forma gratuita por meio do *GitHub Student Pack*.

Utilizamos o Sentry para gerar registros de todas as requisições que são feitas para a nossa API, assim, se necessário for, podemos investigar de forma mais fácil se algum problema surgir no servidor.

Na Figura 13 temos toda a requisição descrita, quanto tempo durou, qual resposta gerou, qual navegador realizou a requisição, dentre outras informações.

Figura 13 – Requisição capturada pelo Sentry



Fonte: elaborado pelo autor (2023).

4 RESULTADOS

Nesta seção descreveremos os resultados obtidos por meio de duas formas. A primeira, por meio de uma análise feita por uma ferramenta chamada SonarQube, que analisa a qualidade geral de um código e fornece diversas métricas e sugestões de melhoria. E, para a segunda utilizaremos o *framework* de testes de carga chamado NBomber para medir a performance da nossa API.

4.1 Qualidade de Código

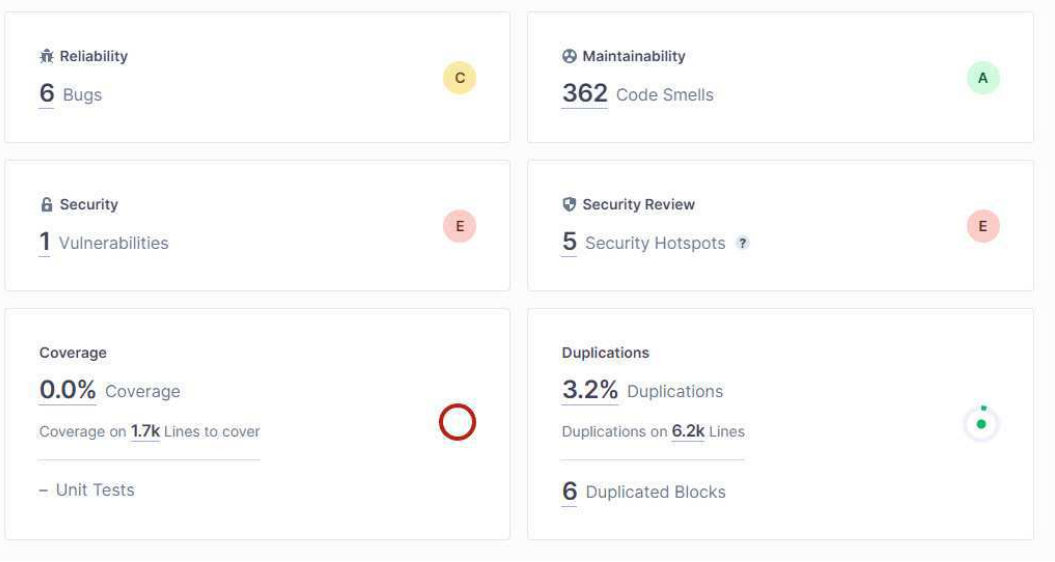
A qualidade do código é uma forma de falar sobre o quão eficiente, legível e utilizável é o código. A codificação é inerentemente aberta e você pode resolver o mesmo problema na mesma linguagem de programação de diversas formas. A qualidade do código mede a precisão e a confiabilidade do código, mas ser livre de erros e portátil não é a única medida da qualidade do código. Ela também inclui o quão amigável é o código para os desenvolvedores. A qualidade do código também descreve o quão fácil é entender, modificar e reutilizar o código, se necessário (AWS, 2023).

Dessa forma, utilizamos o SonarQube, uma ferramenta amplamente adotada no meio empresarial, inclusive por empresas de grande porte, que possui suporte para diversas linguagens de programação também.

Após analisarmos o código com o SonarQube, nos deparamos com o seguinte resultado:

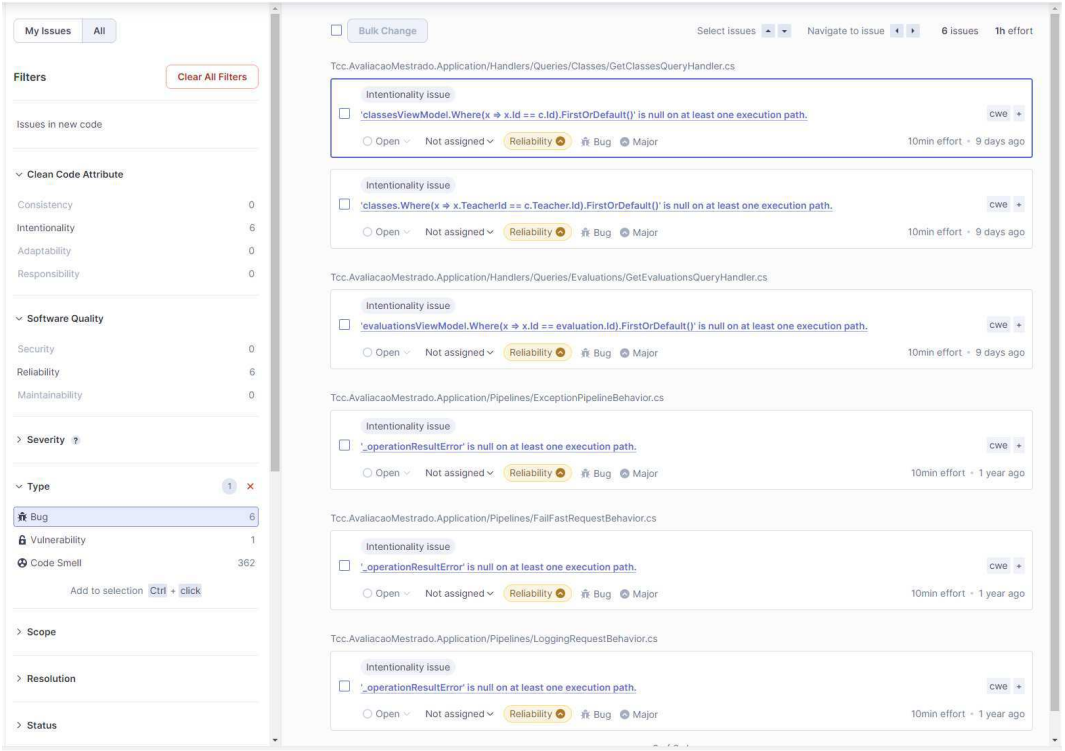
Podemos ver, nas Figuras 14 e 15, que existem 6 possíveis bugs, 1 vulnerabilidade, 362 code smells, que indicam problemas de padrões de código, 5 pontos de possíveis falha de segurança e 3.2% das linhas estão duplicadas. O SonarQube também fornece, em sua interface, mais detalhes desses problemas indicando, até, possíveis soluções.

Figura 14 – Resultado inicial SonarQube



Fonte: elaborado pelo autor (2023).

Figura 15 – Detalhes dos problemas encontrados pelo SonarQube



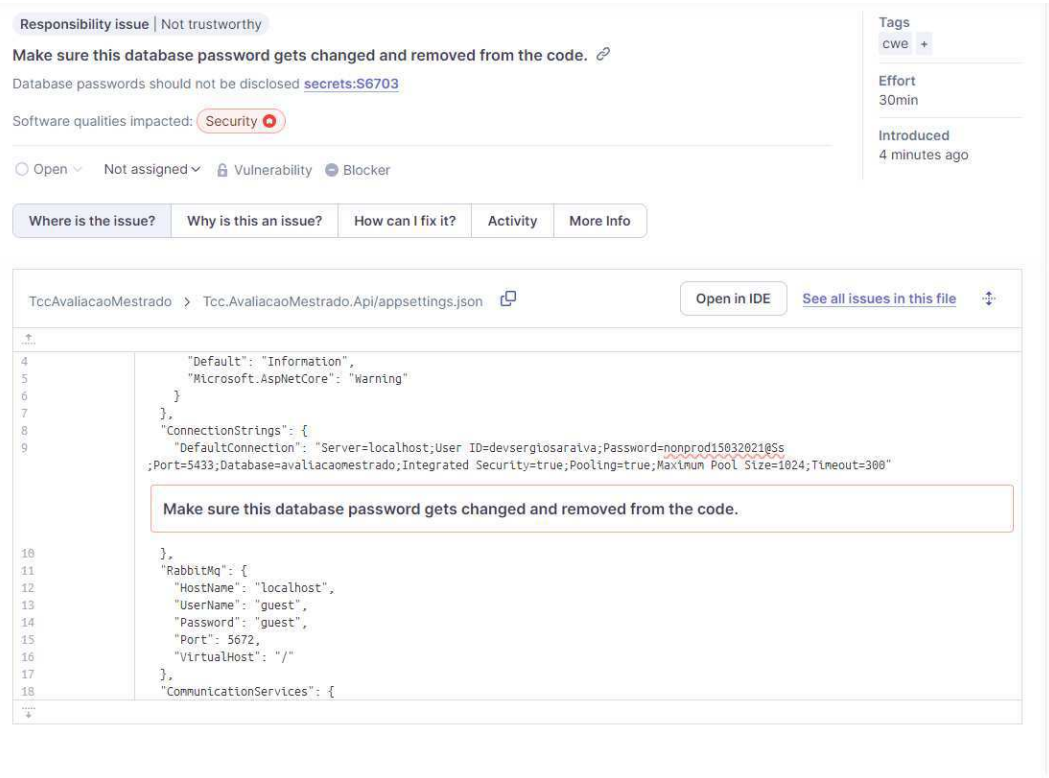
Fonte: elaborado pelo autor (2023).

Na figura 16 vemos que a ferramenta apontou que credenciais de acesso do banco de dados estavam expostas e recomendou que ou removêssemos ou mudássemos essa senha com frequência.

Com isso em mãos, atuamos para corrigir alguns dos problemas apontados, como podermos ver nas Figuras 17 e 18, após nova analise feita com as correções:

Reduzimos as riscos apontados como altos na manutenibilidade do código, ajudando

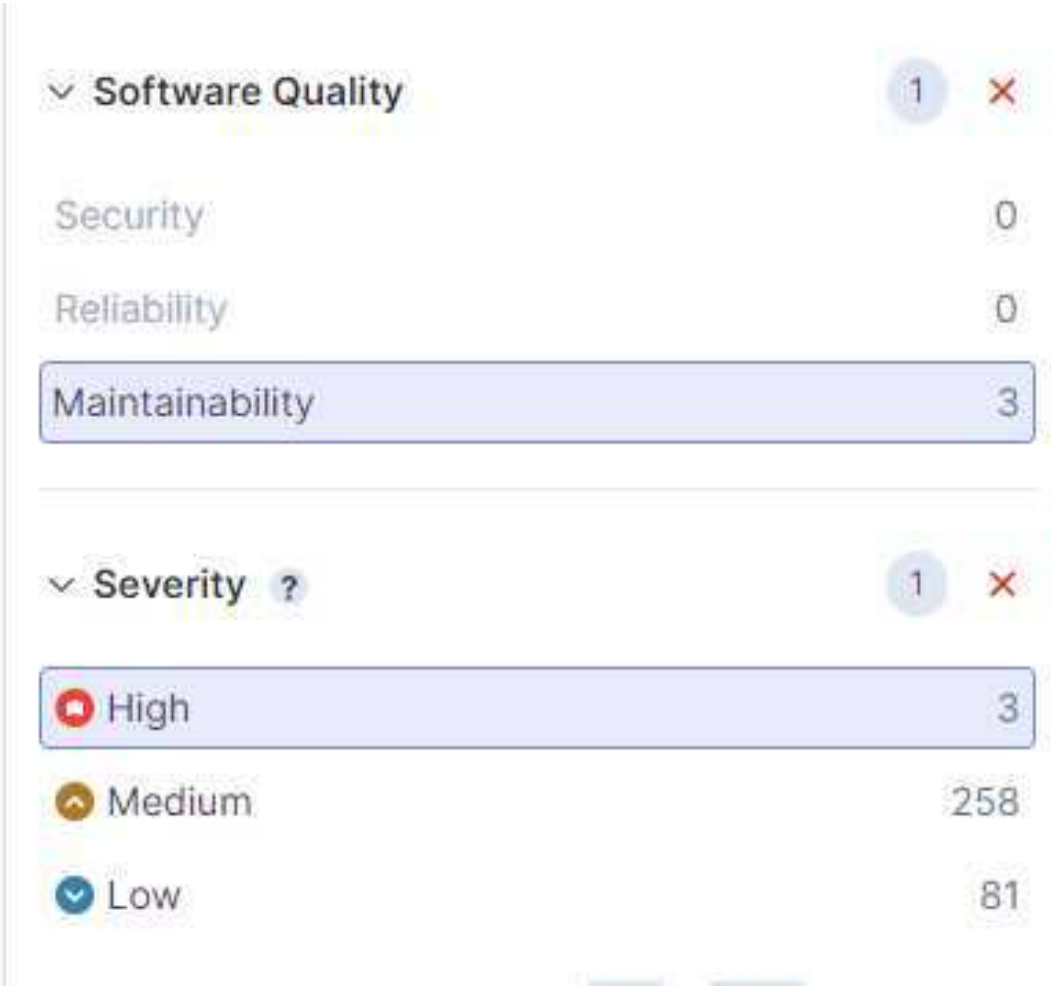
Figura 16 – Problema detalhe pelo SonarQube



Fonte: elaborado pelo autor (2023).

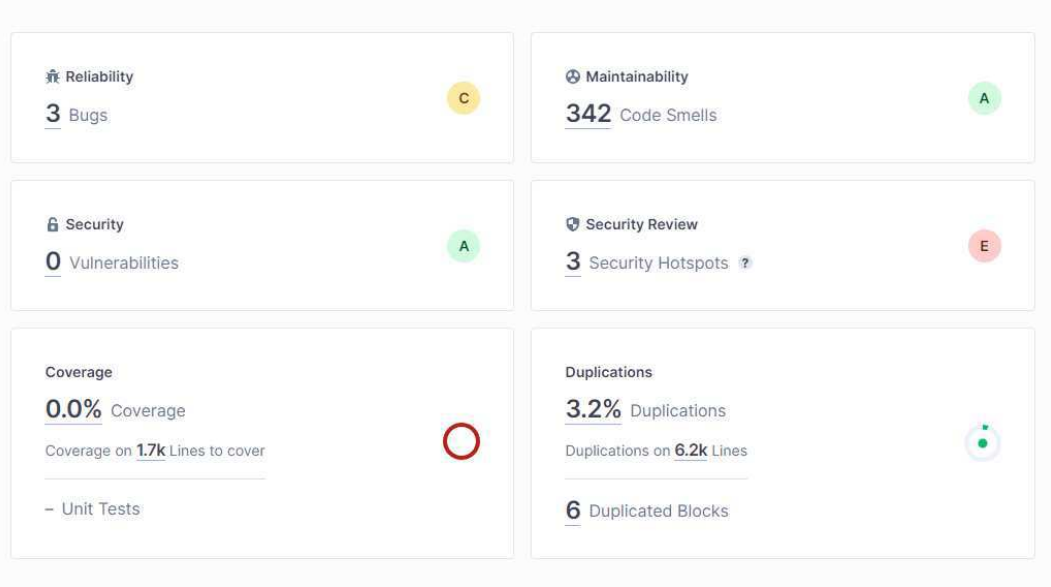
assim futuras alterações de outros desenvolvedores. Reduzimos a quantidade de *Code Smells* em 20, diminuímos 3 possíveis *bugs*, não há mais vulnerabilidades e reduzimos os riscos de segurança em 2. E, no geral, após nova análise, obtivemos esse resultado:

Figura 17 – Novo resultado de problemas de manutenibilidade



Fonte: elaborado pelo autor (2023).

Figura 18 – Resultado final do SonarQube



Fonte: elaborado pelo autor (2023).

4.2 Testes de Carga

Para realizar os testes da nossa API utilizaremos o ambiente de cloud que implantamos e a ferramenta NBomber para construirmos nossos testes.

Testaremos nossa API com *smoke test*, *spike test* e *stress test*. Para o *spike test* configuramos um cenário com um pico de 1000 (mil) requisições que deveriam ser executadas por segundo durante um intervalo de 30 segundos. Para o *smoke test* testamos um cenário que manteria executando 100 (cem) requisições, isto é, quando uma termina logo outra já é executada, dessa forma conseguimos simular 100 usuários conectados à API, mantemos isso durante 1 minuto. Já para o *stress test* executamos um cenário com que manteria 500 (quinhentas) requisições em execução, da mesma forma do teste anterior, durante 3 minutos. Abaixo temos os resultados obtidos.

Vale ressaltar que os testes foram feitos contra a rota de listar formulário por id, no ambiente de desenvolvimento.

Tabela 1 – Resultados dos testes de carga

Teste	Tempo(mm:ss)	Qtd. Req.	Sucesso	Falha	RPS
Spike	00:30	24874	740	24134	12.3
Smoke	01:00	588	588	0	9.8
Stress	03:00	1977	1977	0	11

Fonte: Elaborado pelo autor(2023).

Tabela 2 – Segundo parte dos resultados de teste de carga

Teste	Tempo(mm:ss)	Min(ms)	Média(ms)	Max(ms)
Spike	00:30	63213.16	73823.23	99938.18
Smoke	01:00	701.48	11065.19	14369.32
Stress	03:00	758.58	52978.13	66764.71

Fonte: Elaborado pelo autor(2023).

Os dados para o teste de pico condizem com o esperado, visto que a instancia que a API está hospedada é de pequeníssimo porte possui poucos recursos disponíveis, contudo é surpreendente que tenha suportado o teste de estresse, sem falhas, embora o tempo de resposta tenha sido muito alto. Era o esperado que suportasse o teste de fumaça, visto que eram poucas requisições e um tempo mais curto de teste.

5 CONCLUSÕES E TRABALHOS FUTUROS

Podemos concluir, por meio deste trabalho, que conseguimos construir uma API funcional que atende a todos os requisitos levantados durante a análise do Regulamento do Plano de Autoavaliação do Programa de Pós-Graduação em Engenharia Elétrica e Computação. Os requisitos não funcionais também foram cumpridos com sucesso através de pesquisa de mercado.

Após a análise de qualidade de código, vimos que construímos um sistema sólido, manutenível, de fácil legibilidade e seguro, tornando assim sua escalabilidade, melhoria e incrementação mais fácil e confortável para os desenvolvedores. Entretanto, fica como trabalho futuro a implementação de testes unitários ou, até mesmo, teste de integridade. Além de, até mesmo, o uso do próprio *SonarQube* para realizar melhorias graduais no código.

Dos testes de carga podemos traçar trabalhos futuros para melhorar a performance da API, como adicionar uma estratégia de *caching*, implementar o conceito de *load-balacing* ou até mesmo *auto-scaling*, onde a API consegue criar novas instâncias de si mesmo conforme aumenta a carga de usuários.

6 TRABALHOS RELACIONADOS

Nesta seção iremos citar o trabalho que serviu de ponto inicial para este trabalho, bem como o outro trabalho acadêmico que complementa este.

6.1 Sistema de Autoavaliação - PPGEEC/UFC

A partir deste trabalho de conclusão de curso, feito anteriormente, conseguimos analisar o código e o sistema feito e identificar os pontos de melhorias e falhas. Foi a partir deste trabalho, também, que podemos ter um norte do que a aplicação final deveria ser capaz de fazer.

6.2 Sistema de Autoavaliação Web - PPGEEC/UFC

Diferente do trabalho relacionado anterior, o trabalho desta seção complementa o presente trabalho, consumindo a API resultado deste trabalho para implementar uma interface de usuário. Dessa forma, os usuários finais conseguem interagir com as funcionalidades aqui implementadas com maior facilidade e conforto.

REFERÊNCIAS

AWS. **O que é qualidade de código?** 2023. Disponível em: <https://aws.amazon.com/pt/what-is/code-quality/>. Acesso em: 29 nov. 2023.

AZURE. **O que é PostgreSQL?** 2023. Disponível em: <https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-postgresql>. Acesso em: 29 nov. 2023.

BILLWAGNER. **Um tour pela linguagem C**. 2023. Disponível em: <https://learn.microsoft.com/pt-br/dotnet/csharp/tour-of-csharp/>. Acesso em: 28 nov. 2023.

BOLIC, D. **A Practical Guide to CQRS**. 2021. Disponível em: <https://itnext.io/a-practical-guide-to-cqrs-af4e2d797383>. Acesso em: 29 nov. 2023.

CAPES. Autoavaliação de programas de pós-graduação. CAPES, v. 1, n. 1, 2019.

DOCS, N. **Overview**. 2023. Disponível em: <https://nbomber.com/docs/getting-started/overview/>. Acesso em: 29 nov. 2023.

DOCS, S. **SonarQube 10.3 Documentation**. 2023. Disponível em: <https://docs.sonarsource.com/sonarqube/latest/>. Acesso em: 29 nov. 2023.

EKUAN, M. **Como funciona o Azure?** 2023. Disponível em: <https://learn.microsoft.com/pt-br/azure/cloud-adoption-framework/get-started/what-is-azure>. Acesso em: 29 nov. 2023.

EVANS, E. **Domain-driven design: atacando as complexidades no coração do software**. https://www.amazon.com.br/Domain-driven-design-atacando-complexidades-software/dp/8550800651: Alta Books, 2016. v. 3.

GITHUB. **Entendendo o GitHub Actions**. 2023. Disponível em: <https://docs.github.com/pt/actions/learn-github-actions/understanding-github-actions>. Acesso em: 29 nov. 2023.

GITHUB. **GitHub Student Developer Pack**. 2023. Disponível em: <https://education.github.com/pack>. Acesso em: 29 nov. 2023.

GITHUB. **Olá, Mundo**. 2023. Disponível em: <https://docs.github.com/pt/get-started/quickstart/hello-world>. Acesso em: 29 nov. 2023.

IBM. **O que é uma API de REST?** 2023. Disponível em: <https://www.ibm.com/br-pt/topics/rest-apis>. Acesso em: 28 nov. 2023.

JWT.IO. **Introduction to JSON Web Tokens**. 2023. Disponível em: <https://jwt.io/introduction>. Acesso em: 29 nov. 2023.

LABS, G. **Load test types**. 2023. Disponível em: <https://k6.io/docs/test-types/load-test-types/>. Acesso em: 29 nov. 2023.

LOADVIEW. **Teste de Carga de API: O Guia Definitivo para APIs de Teste de Carga Online**. 2023. Disponível em: <https://www.loadview-testing.com/pt-br/teste-de-carga-de-api/>. Acesso em: 29 nov. 2023.

MICROSOFT. **O que é ASP.NET Core?** 2023. Disponível em: <https://dotnet.microsoft.com/pt-br/learn/aspnet/what-is-aspnet-core>. Acesso em: 29 nov. 2023.

MICROSOFT. **Padrão CQRS**. 2023. Disponível em: <https://learn.microsoft.com/pt-br/azure/architecture/patterns/cqrs>. Acesso em: 29 nov. 2023.

MONTEMAGNO, J. **O que é o Docker?** 2023. Disponível em: <https://learn.microsoft.com/pt-br/dotnet/architecture/microservices/container-docker-introduction/docker-defined>. Acesso em: 29 nov. 2023.

NEVES, L. **O que é RabbitMQ?** 2014. Disponível em: <https://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2014_2/leonardo/intro.html>. Acesso em: 29 nov. 2023.

PPGEEC. **REGULAMENTO DO PLANO DE AUTOAVALIAÇÃO DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E COMPUTAÇÃO – PPGEEC**. 2021. Disponível em: <https://ppgeec.ufc.br/wp-content/uploads/2021/02/sei-ppgeec-resolucao-autoavaliacao.pdf>. Acesso em: 28 nov. 2023.

SANTOS, R. S. V. D. AvaliaÇÃo 360 graus como ferramenta da gestÃo estratÉgica de pessoas: Análise na empresa contec contabilidade, no município de itaituba/pa. 2017. Disponível em: <https://www.faculdadedeitaituba.com.br/pdf.php?id=45&f=TCC%20RADYLA%20FINAL.pdf>.

SENTRY. **What is Sentry?** 2023. Disponível em: <https://docs.sentry.io/product/>. Acesso em: 29 nov. 2023.

SHVETS, A. **Dive Into DESIGN PATTERNS**. https://refactoring.guru/design-patterns/book: Independente, 2021. v. 1.

SOMMERVILE, I. **Engenharia de Software**. https://www.amazon.com.br/Engenharia-Software-Ian-Sommerville/dp/8543024978: Independente, 2019. v. 10.

VALENTE, M. T. **Engenharia de Software Moderna**. https://engsoftmoderna.info/: Independente, 2020. v. 1.