Projeto Dirigido por Domínio e Arquitetura de Microsserviços: Um estudo de caso para um Sistema de Gerência de Pavimentos

Pedro Henrique Costa Gadelha

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

João Pessoa, PB

Junho de 2023
Pedro Henrique Costa Gadelha

Projeto Dirigido por Domínio e Arquitetura de Microsserviços: Um estudo de caso para um Sistema de Gerência de Pavimentos

Monografia apresentada ao curso Ciência da Computação do Centro de Informática, da Universidade Federal da Paraíba, como requisito para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Raoni Kulesza

João Pessoa, PB
Junho de 2023

G124p Gadelha, Pedro.
Projeto Dirigido por Domínio e Arquitetura de Microsserviços: Um estudo de caso para um Sistema de Gerência de Pavimentos / Pedro Gadelha. - João Pessoa, 2023.
70 f. : il.

Orientação: Raoni Kulesza.
TCC (Graduação) - UFPB/CI.

1. Desenvolvimento Orientado ao Domínio. 2. Microsserviços. 3. Gerenciamento de Pavimentos. 4. Arquitetura de Software. 5. DDD. I. Kulesza, Raoni. II. Título.

UFPB/CI

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

Trabalho de Conclusão de Curso de Ciência da Computação intitulado **Projeto Dirigido por Domínio e Arquitetura de Microsserviços: Um estudo de caso para um Sistema de Gerência de Pavimentos** de autoria de Pedro Henrique Costa Gadelha, aprovada pela banca examinadora constituída pelos seguintes professores:

_____

Prof. Dr. Raoni Kulesza
Universidade Federal da Paraíba

_____

Prof. Dr. Marcelo Iury Sousa Oliveira
Universidade Federal da Paraíba

_____

Prof. Dr. Ricardo Almeida de Melo
Universidade Federal da Paraíba

João Pessoa, 21 de Junho de 2023

Centro de Informática, Universidade Federal da Paraíba
Rua dos Escoteiros, Mangabeira VII, João Pessoa, Paraíba, Brasil CEP: 58058-600
Fone: +55 (83) 3216 7093 / Fax: +55 (83) 3216 7117

RESUMO

Este trabalho aborda a aplicação de conceitos de Desenvolvimento Orientado ao Domínio e arquitetura de microsserviços na construção de um sistema de gerência de pavimentos com o intuito de criar um sistema escalável, flexível, robusto, adaptável e modular para um domínio muito complexo. Os resultados obtidos demonstram que a utilização de DDD facilita a compreensão do domínio por incentivar a comunicação entre desenvolvedores e os especialistas do domínio, criando assim, um produto final de maior qualidade. Já a utilização da arquitetura de microsserviços promove uma maior modularidade e independência ao sistema, já que a implementação de suas funcionalidades são independentes umas das outras. Ao final do trabalho, concluiu-se que a adoção dessas abordagens no desenvolvimento de software facilita o processo de desenvolvimento de soluções mais robustas e eficazes no domínio de gerenciamento de pavimentos.

Palavras-chave: <Desenvolvimento Orientado ao Domínio>, <Microsserviços>, <Gerenciamento de Pavimentos>, <Arquitetura de Software>, <DDD>.

ABSTRACT

This undergraduate thesis addresses the application of Domain-Driven Design concepts and microservices architecture in the development of a pavement management system with the aim of creating a scalable, flexible, robust, adaptable, and modular system for a highly complex domain. The results obtained demonstrate that the use of DDD facilitates domain understanding by promoting communication between developers and domain experts, thus resulting in a higher-quality end product. The utilization of microservices architecture promotes greater modularity and independence within the system, as the implementation of its functionalities is decoupled. At the end of the study, it was concluded that the adoption of these approaches in software development facilitates the process of creating more robust and effective solutions in the field of pavement management.

Key-words:<Domain-Driven Design>, <Microservices>, <Pavement Management>, <Software Architecture>, <DDD>.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

SGP	–	Sistema de Gerência de Pavimentos
DDD	–	Desenvolvimento Orientado ao Domínio
HDM	–	*Highway Development and Management*
MTL	–	*Monitoring, Tracing and Logging*
ANP	–	*Analytical Network Process*
DNIT	–	Departamento Nacional de Infraestrutura e Transportes
IGG	–	Índice de Gravidade Global
IGI	–	Índice de Gravidade Individual
IDE	–	*Integrated Development Environment*
SHRP	–	*Strategic Highway Research Program*

SUMÁRIO

1. Introdução

A área de desenvolvimento de software vem testemunhando uma rápida evolução, impulsionada pela demanda por sistemas eficientes, flexíveis e escaláveis. Nesse contexto, duas abordagens vem ganhando destaque: Desenvolvimento Orientado ao Domínio (DDD) (Evans, 2016) e arquitetura de microsserviços (Fowler, 2014)

Em seu livro Eric Evans (2016) apresenta a abordagem de criação de software Desenvolvimento Orientado ao Domínio, ela visa compreender e modelar as complexidades e o processo do domínio, desenvolvendo uma linguagem única que será utilizada por especialistas de domínio e os desenvolvedores, possibilitando assim, um melhor entendimento do domínio do negócio resultando em softwares sejam criados com maior agilidade e qualidade.

Para Martin Fowler (2014) a arquitetura de microsserviços é uma forma de desenvolver um sistema como um conjunto de pequenos serviços, oferecendo uma forma modular de construir sistemas distribuídos, dividindo a aplicação em serviços independentes, cada um responsável pela implementação de uma funcionalidade específica, permitindo assim, que o sistema desenvolvido tenha uma maior escalabilidade, resiliência, flexibilidade e manutenibilidade.

O gerenciamento de pavimentos é uma área crítica para garantir a segurança e a qualidade das estradas e vias urbanas, envolvendo coleta, análise e uso de informações relacionadas às condições e manutenção desses pavimentos. O Brasil possui uma malha rodoviária muito extensa, em sua totalidade são 1.720.700 quilômetros, tornando-a a 4ª maior malha rodoviária do mundo (COLOMBO, 2022). Além disso, o transporte rodoviário é o meio de deslocamento mais utilizado no país, sendo responsável por 61% do escoamento de cargas no país (ALVARENGA, 2020). Quando comparamos com outros países podemos ver que a utilização de transportes rodoviários no Brasil é muito superior, com a figura 1 podemos ver que, no Brasil, o transporte rodoviário é favorecido quando comparado a outros meios de transporte, isso se deve ao fato de que, no Brasil, a malha rodoviária dá acesso à mais áreas do país, além disso, ela é usada para complementar o transporte marítimo e aéreo (CARPO LOGISTICS, 2019).

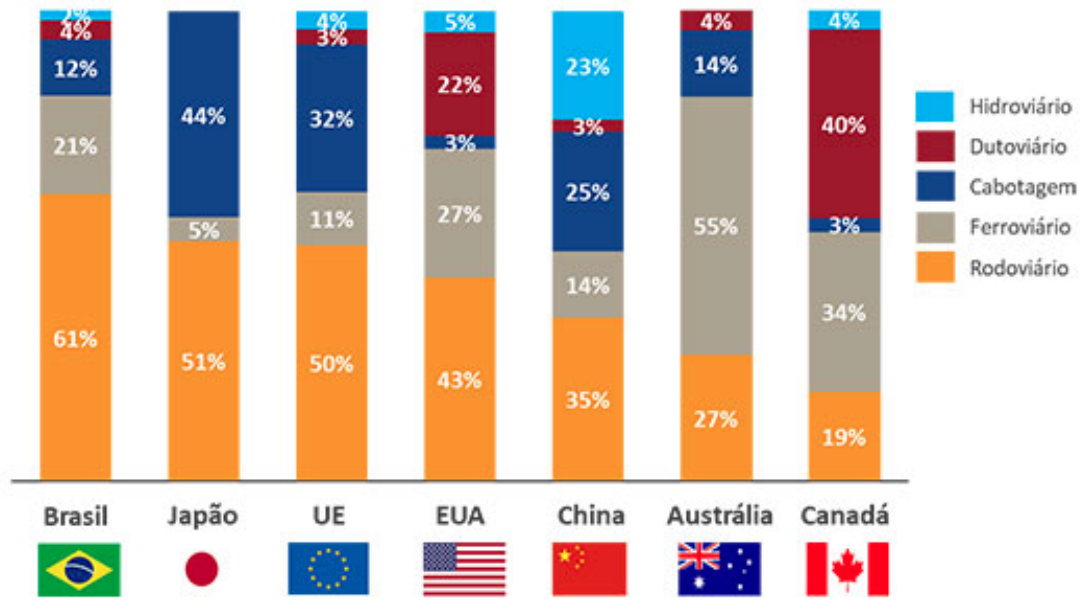

Figura 1 - Quadro comparativo do uso dos meios de transporte

Ao final deste trabalho, será apresentada a arquitetura de um Sistema de Gerência de Pavimentos que foi projetada utilizando os conceitos de DDD e microsserviços discutidos ao longo do mesmo. Além disso, é esperado que os benefícios da utilização dessas abordagens como melhorias na eficiência, qualidade e adaptabilidade do sistema fiquem claros para o leitor.

1.1 Tema

Este trabalho utiliza a abordagem DDD para propor uma arquitetura para uma aplicação orientada a microsserviços. Durante o trabalho serão introduzidos conceitos de arquitetura de software, DDD, microsserviços, gerenciamento de pavimentos, trabalhos que possuem um ou mais temas em comum com este trabalho, uma proposta do sistema com seus requisitos funcionais, não funcionais e domínios e, por fim, será apresentada a arquitetura do sistema, descrevendo seus componentes e seus relacionamentos.

1.2 Problema

Gerenciamento de pavimentos é um processo desafiador, por isso, existem Sistemas voltados para tal atividade. O *Highway Development Management 4* (HDM-4) serve como

ferramenta primária para a administração de pavimentos, porém o mesmo é um sistema datado tendo sua última versão, a 2.0, lançada em 2005. Por isso, devido às limitações da época em que foi desenvolvido, ele não possui diversas funcionalidades que podem ser de grande valor para o gerenciamento de pavimentos como a falta de interação com sistemas externos de georreferenciamento, sistemas de informações geográficas, sensores inteligentes, etc. Além disso, por ser um software desenvolvido para desktop seu potencial de escalabilidade, atualização e utilização são limitados pois cada um está limitado à máquina que foi instalado.

Diante desses desafios, surge a necessidade de explorar abordagens inovadoras que possam resolver os problemas enfrentados pelos SGPs. As abordagens DDD e arquitetura de microsserviços são alternativas promissoras para lidar com a complexidade do domínio, promover a flexibilidade e escalabilidade do sistema e facilitar a integração com sistemas externos.

1.2.1 Objetivo Geral

Este trabalho tem como objetivo principal propor uma arquitetura de software de um SGP aplicando as abordagens DDD e a arquitetura de microsserviços com o intuito de melhorar a eficiência, qualidade e adaptabilidade desses sistemas.

1.2.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

1. Estudar os conceitos de DDD;
2. Estudar os conceitos de Arquitetura de Microsserviços;
3. Desenvolver um estudo de caso aplicando os conceitos estudados;
4. Identificar os benefícios e desafios encontrados ao aplicar DDD e Arquitetura de Microsserviços;

1.3 Estrutura da Monografia

A monografia está estruturada em capítulos, além do capítulo atual que é a introdução temos mais cinco capítulos, que são:

- O capítulo dois apresenta a fundação teórica da monografia, nele são apresentados conceitos de arquitetura de software, microsserviços, DDD e de sistemas de gerenciamento de pavimentos;
- O capítulo três apresenta e discute trabalhos relacionados ao desenvolvido neste trabalho;
- O capítulo quatro apresenta o sistema, nele são elicitados os requisitos funcionais e não funcionais além de aplicar conceitos de DDD como: linguagem ubíqua, contextos delimitados e mapa de contexto;
- O capítulo cinco propõe a arquitetura do sistema, nele é apresentado o modelo C4 e o modelo do banco de dados;
- No capítulo seis chegamos à conclusão do trabalho, nele são dadas as considerações finais do autor e sugestões de trabalhos futuros;

2. Conceitos Gerais e Revisão da Literatura

2.1 Arquitetura de Software

De acordo com Guilherme Germoglio (2010) o processo de desenvolver software é algo complexo, além de lidar com essa complexidade também devemos lidar com como o software vai resolver o problema que motivou seu desenvolvimento, em outras palavras, não devemos apenas nos preocupar com desenvolver o software de forma que ele solucione um problema, é importante também que esse problema seja resolvido de forma eficiente e com qualidade.

2.1.1 Elementos Arquiteturais

Elementos arquiteturais são responsáveis por dar forma ao software, eles definem como o software é particionado, podendo ser divididos em dois tipos: os estáticos e dinâmicos.

Os elementos estáticos definem as partes do sistema e qual sua organização, eles refletem o sistema durante o design e são constituídos de elementos de software, elementos de dados e elementos de hardware. Eles são compostos das relações entre os elementos que compõem a arquitetura do sistema. Essas relações definem diversos aspectos fundamentais do funcionamento do sistema, especificando a comunicação e o controle de informação e comportamento.

Já os elementos dinâmicos definem como o sistema reage a estímulos internos e externos e como o sistema se comporta durante a execução, entre eles estão processos, módulos, protocolos e classes.

2.1.2 Decisões Arquiteturais

As escolhas entre as alternativas de design são chamadas de decisões arquiteturais, separando o sistema em elementos e relação. Tem como objetivo alcançar um ou mais atributos de qualidade do sistema. Decisões possuem três características principais: descrevem elementos

do sistema, explicam qual é o objetivo por trás de alguma decisão e possuem fundamentação em algum padrão ou conhecimento prévio do arquiteto.

Decisões arquiteturais também são responsáveis por descrever o rumo que um software deve seguir durante seu ciclo de vida, decisões podem ser tomadas para adicionar , remover ou modificar funcionalidades bem como sua modificação e manutenção. Além disso, também facilitam o rastreamento de requisitos por ser responsável por relacioná-los a elementos arquiteturais.

2.1.3 Interessados (do inglês, *Stakeholders*)

Os *stakeholders* podem ser pessoas, grupos ou entidades que possuem uma forte influência sobre a arquitetura de software e os atributos de qualidade. Eles possuem diversas responsabilidades durante o ciclo de vida do software, entre elas financiamento, projeto, desenvolvimento, testes, uso e manutenção do software. Além disso, eles também têm necessidades como bom desempenho, segurança e usabilidade do sistema. A arquitetura tem como objetivo facilitar o cumprimento das responsabilidades e atender às necessidades dos *stakeholders*.

Durante o desenvolvimento de software pode existir conflito de interesses entre os *stakeholders*, isso acontece pois em um sistema grande também teremos uma grande quantidade de *stakeholders* envolvidos. Alguns exemplos de *stakeholders* são: usuários, clientes, arquiteto, desenvolvedor, testador e o gerente de projeto.

Figura 2 - *Stakeholders* Possíveis em uma Aplicação

2.1.4 Requisitos

Requisitos são atributos de qualidade definidos ao longo do processo de desenvolvimento de software. Eles podem ser classificados como requisitos funcionais e não-funcionais. Eles são definidos pelos *stakeholders* e representam os seus interesses, por isso é possível que exista conflito entre eles. Cabe ao arquiteto priorizar os requisitos mais importantes para o funcionamento do sistema, já que atender a todos de forma ótima não é possível.

2.1.4.1 Requisitos Funcionais

Requisitos funcionais descrevem como o software deve se comportar, eles costumam ser definidos pelos clientes, por isso sua implementação é um grande desafio e, normalmente, requerem conhecimentos que vão além do escopo de desenvolvimento de software.

2.1.4.2 Requisitos Não-Funcionais

Requisitos não-funcionais descrevem propriedades, características ou restrições do software, eles podem ser definidos por vários *stakeholders* diferentes e estão normalmente ligados a interface com o usuário, consumo de recursos, tempo de resposta, etc. Eles podem ser divididos em três tipos: produto que estão relacionados à qualidade do software, processo que estão relacionados às restrições do software e externos que são relacionados às limitações impostas por fatores externos.

2.2 Arquitetura de Microsserviços

A arquitetura de microsserviços é uma forma de desenvolver uma aplicação integrando vários microsserviços, estes que exercem suas funções de forma independente e podem se comunicar entre si. Também temos uma quantidade mínima de gerenciamento centralizado dos serviços, possibilitando que ao criar um microsserviço o desenvolvedor tenha liberdade de utilizar a linguagem de programação e tecnologias que mais se adequem ao seu propósito.

A arquitetura de microsserviços surgiu a partir de frustrações com a arquitetura monolítica, nela o software é desenvolvido em uma única aplicação, ou seja, qualquer alteração feita na aplicação fará com que seja necessário que a build seja refeita junto com seu deploy. Já os microsserviços por serem independentes do restante da aplicação podem ser facilmente escaláveis e implementados de forma individual.

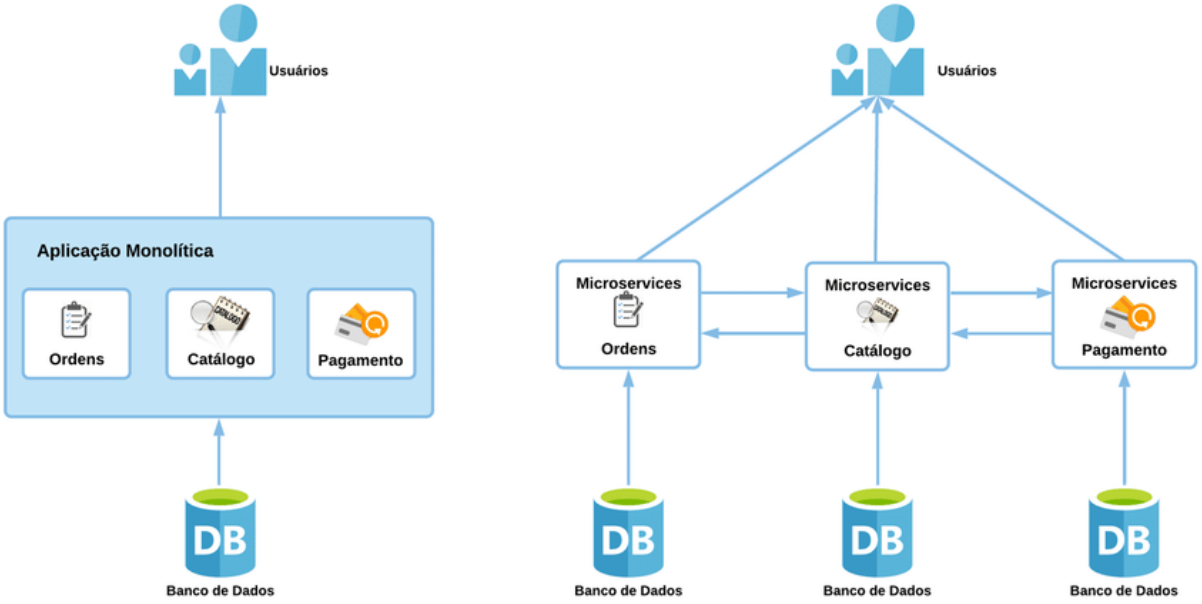

Figura 3 - Arquitetura Monolítica vs Arquitetura de Microsserviços

2.2.1 Microsserviços

Para Vlad Khononov (2021) o microsserviço é como uma micro porta da frente de uma aplicação, ou seja, ele possui a mesma função de um serviço convencional, porém com suas funcionalidades reduzidas. Reduzir as funcionalidades através de um microsserviço é algo vantajoso, pois assim teremos poucos motivos para mudança e torna o serviço mais independente para desenvolvimento e manutenção.

2.2.2 Características de uma Arquitetura de Microsserviços

Não existe uma definição formal do estilo arquitetural de microsserviços, mas para Martin Fowler (2014) é possível definir suas principais características que são:

- Componentização via microsserviços: microsserviços são utilizados como componentes pois através deles é possível encapsular uma série de funcionalidades e podemos facilmente substituí-lo ou melhorá-lo. Além disso, a componentização faz com que sua interface seja mais explícita;
- Organização baseada na capacidade do negócio: na arquitetura de microsserviços o desenvolvimento e manutenção dos microsserviços são divididos baseando-se nas capacidades de cada equipe, que encapsulam todas as habilidades necessárias;

- Produtos não projetos: na arquitetura de microsserviços a equipe responsável pelo desenvolvimento de um produto é responsável também por mantê-lo durante todo o seu ciclo de vida, fazendo com que os desenvolvedores tenham contato diário com como o seu software se comporta em produção, e consequentemente, aumentando o contato com seus usuários;
- Endpoints inteligentes: na arquitetura de microsserviços as aplicações tem como objetivo serem o máximo desassociadas e coesivas possível, por isso elas devem funcionar como um filtro em que recebem uma requisição, aplicam a lógica e retornam uma resposta utilizando protocolos *REST* simples;
- Governança descentralizada: garante ao desenvolvedor a liberdade de escolher as tecnologias que serão utilizadas para resolver determinado problema, isso é uma vantagem para os microsserviços já que será possível escolher as tecnologias que mais se adequam a cada microsserviço;
- Gerenciamento de dados descentralizado: microsserviços podem possuir bancos dedicados, podendo serem instâncias diferentes do mesmo banco de dados ou totalmente diferentes. Porém, também torna necessária a utilização de transações para manter a consistência entre os dados em microsserviços diferentes;
- Automação da infraestrutura: equipes que desenvolvem microsserviços fazem entregas contínuas, ou seja, o software desenvolvido tem que estar pronto para ser implementado em produção a qualquer momento, por isso o uso extensivo de técnicas de automação de infraestrutura são necessárias. Alguns exemplos são: testes automáticos, testes de performance, testes de integração, etc;
- Tolerância à falhas: aplicações que utilizam microsserviços são projetadas para serem capazes de tolerar falhas, por isso possuem monitoração em tempo real robusta. Através da monitoração é possível detectar falhas rapidamente e talvez até mesmo restaurá-las automaticamente para ter o mínimo possível de *downtime*. Além disso, uma monitoração robusta ajuda aos desenvolvedores a encontrar e corrigir problemas;
- Design evolutivo: microsserviços são projetados para serem facilmente modificados, ou até mesmo reescritos, sem causar nenhum tipo de impacto para qualquer outro micro serviço que o utilize;

2.2.3 Desafios de uma Arquitetura de Microsserviços

No artigo Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review (Söylemez, 2022) foi feito um estudo sobre os maiores desafios ao se implementar arquitetura de microsserviços, utilizando uma análise sistemática da literatura (SLR) e levando em consideração 85 artigos relacionados a microsserviços como objetos de estudo. Com isso, foi possível identificar os principais desafios de uma arquitetura de microsservicos, eles são:

- Descoberta de serviços: é uma tarefa desafiadora pois existem vários mecanismos de descoberta de serviços na fase de design como serviços do lado do cliente, do lado do servidor e híbridos. Além disso, a decisão deve ser feita levando em consideração os requisitos e atributos de qualidade do sistema;
- Gerenciamento e Consistência de Dados: o maior desafio está no gerenciamento de transações distribuídas, já que é normal que microsserviços diferentes implementem soluções de persistência de dados diferentes, tornando-se um desafio manter os dados sincronizados entre as múltiplas soluções;
- Testes: cada microsserviço está hospedado em um ambiente diferente e pode utilizar tecnologias, linguagens e infraestruturas diferentes aumentando bastante a complexidade dos testes;
- Prever, Medir e Otimizar de Performance: é ideal que a performance do sistema seja estimada antes da sua implementação pois pode ser muito difícil ou caro fazer isso após;
- Comunicação e Integração: é difícil de garantir que os microsserviços estão se comunicando de forma confiável e que o protocolo consegue suportar fluxos complexos, mesmo que eles se comuniquem utilizando protocolos leves;
- Orquestração de Serviços: orquestração de serviços é um desafio pois é necessário que o software consiga (1) rastrear os requisitos dos containers para fazer ajustes necessários de acordo com o uso de recursos; (2) fornecer soluções de armazenamento entre diferentes containers; (3) planejamento e configuração da implementação containers; (4) balancear as requisições entre os servidores do *backend*; (5) escalar serviços de forma automática; (6) entender o comportamento de falha e recuperação dos containers;

- Segurança: por ser uma arquitetura distribuída é difícil de aplicar um mecanismo de controle de acessos. Além disso, a dificuldade de criar um *framework* para comunicação segura entre os microsserviços e monitorar o tráfego de rede com regras de segurança para atender os requisitos contribuem para aumentar a complexidade relacionada à segurança na arquitetura de microsserviços;
- Monitorando, Rastreando e Registrando (MTL): monitorar, rastrear e registrar são de extrema importância para garantir que o sistema é capaz de satisfazer os requisitos de disponibilidade, performance e confiabilidade. Porém, essas atividades consistem de diversos pontos desafiadores, *logs* de microsserviços diferentes podem seguir padrões diferentes, se o problema não pode ser rastreado pelos *logs* a habilidade de monitorar o sistema é afetada diretamente. Se os problemas não forem detectados e contornados rapidamente a disponibilidade, confiabilidade e a performance do sistema vão ser afetadas;
- Decomposição: se um domínio não for decomposto corretamente a arquitetura de microsserviços não vai ser vantajosa e pode causar problemas ao sistema, principalmente à sua escalabilidade, performance, disponibilidade e confiabilidade.

2.3 Desenvolvimento Orientado ao Domínio (DDD)

Para Eric Evans (2016) o DDD é um conjunto de princípios para projeto de software, tendo como objetivo permitir o desenvolvimento de sistemas cujo design é centrado em conceitos próximos e alinhados com o domínio de negócio.

DDD defende que o design do sistema deve ser norteado para atender ao seu domínio de negócio e não frameworks, linguagens de programação, arquiteturas e outras tecnologias. Os desenvolvedores devem ter um profundo conhecimento do domínio da aplicação que está sendo desenvolvida, para se obter esse conhecimento são feitas diversas discussões com os especialistas de domínio.

Para fazer a separação entre o domínio e as tecnologias utilizadas no desenvolvimento são utilizados padrões arquiteturais como a Arquitetura de Camadas, Arquitetura Hexagonal, Arquitetura de Cebola, etc. Além disso, DDD possui três conceitos principais que são: linguagem ubíqua, contextos delimitados e mapas de contexto.

2.3.1 Domínio

Para Wesley Williams (2019) o domínio pode ser considerado como o coração do negócio, ele baseia-se em um conjunto de ideias, conhecimentos e processos de negócio, sem ele não existe razão para o desenvolvimento de software.

2.3.2 Linguagem Ubíqua

Linguagem ubíqua consiste em um conjunto de termos que devem ser plenamente entendidos tanto por especialistas de domínio quanto pelos desenvolvedores da aplicação, ambos devem conseguir entender os mesmos termos para facilitar a comunicação, em outras palavras, ela vai ter o papel de ser a linguagem falada no dia a dia da empresa. A linguagem ubíqua tem dois propósitos principais: possibilitar a comunicação fluida entre desenvolvedor e especialista de domínio e nomear entidades do sistema como classes, métodos, atributos, módulos, etc.

Além de se classificar os termos na linguagem ubíqua, também é importante definir os relacionamentos e associações existentes entre eles, para documentar esses relacionamentos pode ser utilizado um diagrama de classes UML.

Figura 4 - Linguagem Ubíqua

A figura demonstra que existem termos que só são conhecidos pelos especialistas de domínio e termos que só devem ser conhecidos pelos desenvolvedores, porém também mostra que existem termos que devem fazer parte do vocabulário de ambos para facilitar o entendimento.

2.3.3 Contextos Delimitados (do inglês, *Bounded Context)*

Muitos softwares são grandes demais para serem colocados em um único domínio, por isso é natural que eles sejam divididos em diversos domínios diferentes que atendem a usuários diferentes e possuem sua própria linguagem ubíqua. Essa divisão é feita utilizando os contextos delimitados.

2.3.4 Mapa de Contexto (do inglês, *Context Map*)

O mapa de contexto é o mapeamento dos contextos delimitados, com ele é possível entender como cada contexto se comunica com os demais e como eles funcionam de forma independente. De acordo com Vladik Khononov (2019) existem três grupos que dividem os padrões de colaboração entre os mapas de contexto, são eles:

- Parceiros: neste padrão os times donos dos contextos delimitados cooperam para resolver quaisquer problemas de integração que podem vir a aparecer;
- Kernel compartilhado: define áreas de contato entre múltiplos contextos delimitados, cada time é livre para modificar a área de interseção;
- Produtor-consumidor: neste padrão um dos contextos delimitados produz (*upstream*) algo que é consumido (*downstream*) por outro;
- Conformista: este padrão acontece quando o contexto consumidor (*downstream*) se adequa ao contexto produtor (*upstream*);
- Camada anti-corrupção: neste padrão o domínio *downstream* não quer se adequar ao domínio *upstream*, para resolver o problema uma camada anti-corrupção é criada para adequar os dados;
- Serviço open-host: neste padrão o modelo do consumidor é desacoplado em uma interface pública;
- Não colaboração: neste padrão não existe colaboração entre os contextos;

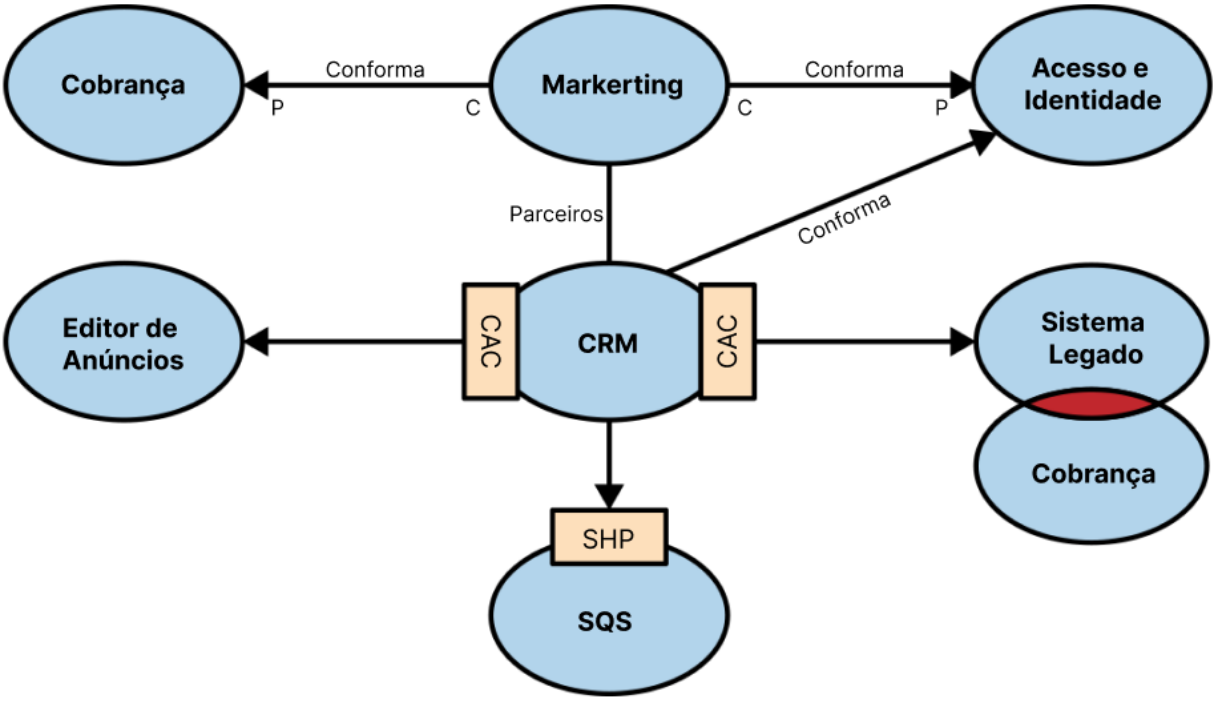

Figura 5 - Exemplo de Mapa de Contexto

2.3.5 Objetos de Domínio

Também conhecidos como *building blocks,* os objetos de domínio devem ser entendidos como ferramentas conceituais que um projetista deve utilizar para projetar com sucesso um determinado sistema. Eles podem ser entidades, objetos de valor, serviços, agregados e repositórios.

2.3.5.1 Entidades e Objetos de Valor

Uma entidade é um objeto que possui uma identidade única, ou um id, que o distingue dos demais da mesma classe. Já os objetos de valor não possuem um id, eles são caracterizados apenas pelos seus valores atribuídos, se dois objetos de valor possuírem os mesmos valores eles serão idênticos. Além disso, objetos de valor são imutáveis, ou seja, uma vez criados seus valores não podem ser alterados e caso seja necessário alterar é preciso excluir o antigo e criar um novo objeto.

2.3.5.2 Serviços

Serviços são objetos do domínio que implementam operações importantes para o domínio da aplicação que não são específicas para nenhuma entidade, eles também devem ser *stateless*, ou seja, não devem possuir atributos apenas métodos. Normalmente são implementados como singletons, possuindo uma única instância durante toda a execução do sistema.

2.3.5.3 Agregados

Segundo Marco Valente (2022) agregados são um conjunto de entidades ou objetos de valor que só devem ser acessíveis através da sua raiz, que obrigatoriamente será uma entidade. Ela deve ser capaz de referenciar todos os objetos que o formam, porém esses objetos não devem ser visíveis para o resto do sistema.

2.3.5.4 Repositórios

De acordo com Marco Valente (2022) , repositório é um objeto utilizado para recuperar ou persistir outros objetos no banco de dados da aplicação, eles são utilizados para prover uma abstração que blinde os desenvolvedores de preocupações relacionadas ao banco de dados.

2.3.6 Desafios ao Implementar DDD

Já para Vaughn Vernon (2016) os desafios mais comuns ao implementar o DDD são dedicar tempo e esforço o suficiente para a criação de uma linguagem ubíqua, envolver os especialistas de domínio de forma constante durante o desenvolvimento do projeto e mudar a forma que os desenvolvedores pensam sobre soluções no domínio.

2.3.6.1 Dedicar Tempo e Esforço o Suficiente para a Criação da Linguagem Ubíqua

Pesquisar conceitos e terminologias, conversar com especialistas de domínio para descobrir, capturar e melhorar a linguagem ubíqua é considerado como um dos desafios de se aplicar DDD pois é um processo que requer muito tempo e dedicação durante o

desenvolvimento, caso esse processo seja negligenciado a linguagem ubíqua não será robusta o suficiente para cumprir o seu papel no DDD.

Em *An Ontology-based Approach for Domain-Driven Design of Microservice Architectures* (Diepenbrock, 2017) é apontado um desafio sintático na aplicação do DDD que é times possuírem nomenclaturas diferentes para o mesmo conceito, isso é causado pela falta de suporte para manter e expressar a linguagem ubíqua no DDD.

2.3.6.2 Envolver os Especialistas de Domínio Durante o Desenvolvimento

Conseguir o envolvimento necessário dos especialistas de domínio também é um desafio ao implementar o DDD, porém para Vernon (2016) é de extrema importância que se consiga a cooperação de ao menos um especialista de domínio para que seja possível que os desenvolvedores sejam capazes de modelar um software que reflita o modelo mental dos especialistas.

2.3.6.3 Mudar a Forma que os Desenvolvedores Pensam Sobre Soluções

Segundo Vernon (2016) os desenvolvedores tendem a pensar de forma técnica, isso é um problema pois nem sempre pensar de forma técnica é o melhor caminho, por exemplo, durante o desenvolvimento da linguagem ubíqua.

2.3.6.4 Erros comuns ao se implementar DDD

Segundo Daniel Whittaker (2015) existem dez erros comuns ao adotar DDD, eles são:

- Permitir que a persistência de dados e bancos de dados tenham influência no modelo;
- Falta de comunicação entre os especialistas de domínio e os desenvolvedores;
- Ignorar a linguagem dos especialistas de domínio;
- Não identificar os contextos delimitados;
- Não atualizar os contextos delimitados para se adequarem aos novos entendimentos;
- Usar modelos de domínio sem uma função bem definida;
- Assumir que toda a lógica é lógica de domínio;

- Usar exageradamente testes de interação;
- Tratar segurança como parte do domínio quando ela não faz parte;
- Focar na infraestrutura.

2.4 Gerenciamento de Pavimentos

Um sistema de gerência de pavimentos (SGP) é obrigatório para a captação de investimentos de infraestrutura rodoviária devido ao programa *Highway Development and Management* (HDM) do *World Bank*. Atualmente o programa está na sua quarta edição (HDM-4) e possui ferramentas capazes de auxiliar a análise de rodovias, porém apesar disso países desenvolvidos preferem implementar suas próprias SGPs projetadas com suas malhas rodoviárias em mente.

2.4.1 *Highway Development and Management* (HDM)

O Highway Development and Management (HDM) é um software desenvolvido pelo Banco Mundial que permite analisar determinado trecho rodoviário e identificar a viabilidade das soluções propostas (BetaInfraestrutura, 2019). O seu desenvolvimento iniciou-se em 1968 com o nome de Highway Cost Model (HCM), apenas em 1976 o Banco Mundial expandiu o banco de dados do HCM abrangendo estudos complementares em países como Índia, Brasil e Caribe, e além disso, o software passou ser constantemente aprimorado e atualizado e adotando a nomenclatura que conhecemos hoje. Desde sua concepção o HDM teve várias versões, a mais atual é conhecida como HDM-4 tendo seu lançamento em 2000.

2.4.2 Sistema de Gerência de Pavimentos (SGP)

SGP é um conjunto de ferramentas ou métodos que auxiliam na escolha de uma estratégia ótima que vai fornecer, avaliar e manter pavimentos em condições de serviço adequadas ao longo do tempo.

Para Hudson et. al. (2015) existem ao todo cinco características essenciais que um SGP deve ter implementado, elas são:

- Capacidade de ser facilmente atualizado e/ou modificado;
- Capacidade de considerar estratégias alternativas;

- Capacidade de identificar a alternativa ou estratégia ótima;
- Capacidade de tomar decisões baseadas em procedimentos racionais;
- Capacidade de auto aprendizado baseando-se nas consequências das decisões tomadas anteriormente.

Na figura 6 temos ilustrado o fluxograma do funcionamento de um sistema de gerência de pavimentos:

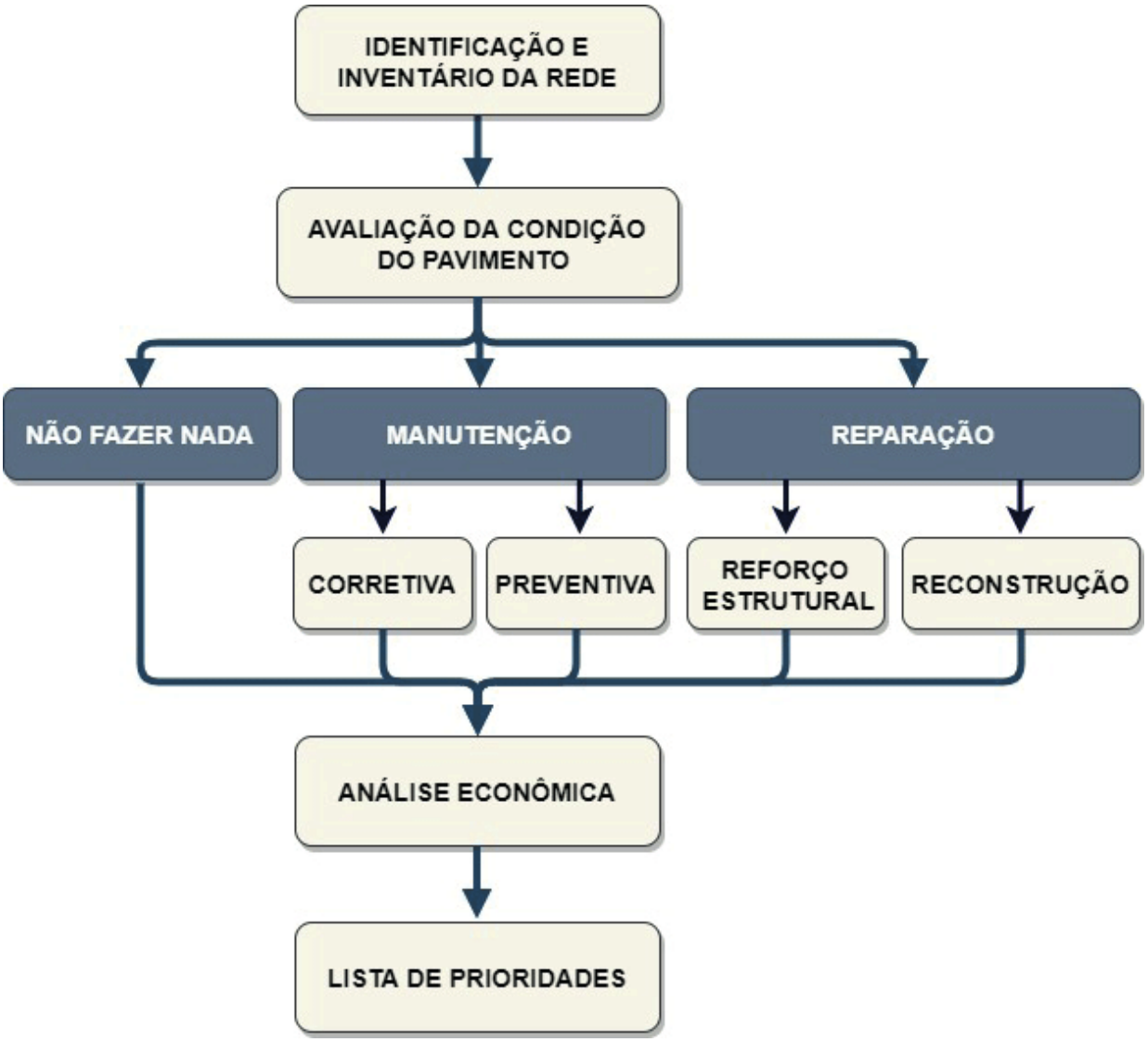

Figura 6 - Fluxograma de um SGP

2.4.3 Banco de Dados do Sistema

De acordo com o DNIT (2011) o banco de dados do sistema deve estar disponível para todos os usuários ligados à gerência de pavimentos a fim de homogeneizar o acesso aos dados coletados e às suas atualizações, ele também deve ser abrangente e confiável sendo fundamental para o bom funcionamento do SGP. Os dados armazenados nele serão a base para a tomada de decisões, sendo obrigatoriamente atuais, objetivos e confiáveis. Além disso, é necessário que esses dados sejam constantemente atualizados, aumentando assim a precisão da aplicação e facilitando a descoberta de estratégias alternativas.

Alguns exemplos de categorias de dados que podem ser coletados são:

- Desempenho como irregularidades, atrito, deflexão e desgaste da superfície;
- Histórico como manutenção, construção, acidentes e tráfego;
- Política como orçamentos, disponibilidades e alternativas;
- Geometria como curvatura vertical, dimensões das seções e espessura da camada;
- Custos como reabilitação, construção, manutenção e custos ao usuário;
- Meio ambiente como clima e drenagem;

A coleta destes dados permite que seja feito um diagnóstico mais preciso da malha atual, prever condições futuras dos pavimentos e um melhor planejamento no sistema de gerência.

2.4.4 Avaliação de Desempenho

A avaliação de desempenho de um pavimento é uma parte fundamental de um SGP, é a partir dela que se pode determinar o grau de deterioração de uma determinada área, quais atividades de manutenção e reabilitação são necessárias, permitir a verificação da qualidade de construção do pavimento e verificar se ele atende aos padrões do projeto. Além disso, a avaliação também serve como *feedback* ao banco de dados, já que eles podem ajudar na análise de novas alternativas.

As avaliações são divididas em dois tipos, funcionais e estruturais, a partir dos seus resultados é possível obter diagnósticos, determinar estratégias de intervenção e estimar a vida útil restante de um pavimento. Além disso, elas devem ser feitas periodicamente e possuem

quatro critérios principais: conforto de rodagem, capacidade de suportar a solicitação das cargas, segurança e confortos visuais ou de ruído gerados pelo tráfego.

- Avaliações funcionais: levam em consideração aspectos que chamam atenção de um usuário sem conhecimento técnico, sendo capaz de apontar as principais deficiências de determinado trecho do pavimento que resultam em desconforto para o usuário e maiores custos operacionais e de segurança. Em outras palavras as avaliações funcionais determinam a serventia de um pavimento, possibilitando assim, determinar o desempenho do pavimento;
- Avaliações estruturais: avalia a capacidade de um pavimento manter sua integridade, estimando a vida útil de um pavimento levando em consideração as condições de tráfego existentes;

2.4.5 Levantamento de Defeitos

O levantamento de defeitos tem como objetivo quantificar e qualificar os defeitos na superfície de um pavimento resultando numa diminuição na avaliação de condição da malha, baseando-se na gravidade, frequência e no tipo do defeito. Através dessa atividade é possível: (1) determinar os tipos, a severidade e extensão dos defeitos visíveis; (2) determinar os índices de condição do pavimento; (3) diagnosticar os problemas apresentados pelo pavimento; (4) determinar as necessidades atuais e futuras; (5) auxiliar no dimensionamento do pavimento a ser restaurado; (6) estabelecer prioridades orçamentárias; (7) prever a curva de deterioração; (8) estimar vida útil restante dos pavimentos (SILVA, 2018).

O manual de identificação de defeitos (SHRP, 2014) é responsável por descrever e classificar os principais defeitos que podem ser encontrados em um pavimento, entre eles: trincas por fadiga, trincas em blocos, trincas nas bordas, trincas longitudinais, trincas por reflexão, trincas transversais, remendos, panelas, deformação permanente, corrugação, exsudação, agregados polidos, desgaste, desnível entre pista e acostamento e bombeamento.

Após o levantamento de defeitos é necessário quantizar a condição do pavimento, para isso os pavimentos são separados em grupos homogêneos e identificados em seções, a quantização é feita utilizando técnicas como o índice de gravidade global (IGG).

2.4.5.1 Índice de Gravidade Global (IGG)

Segundo Antonia Almeida et. al. (2019) o índice de gravidade global é um método de avaliação objetiva de rodovias com fatores de ponderação para defeitos de revestimento em concreto asfáltico, mas comumente aplicado para qualquer tipo de revestimento flexível. O IGG é calculado pelo somatório de todos os Índices de Gravidade Individual (IGI) do pavimento, a equação é a seguinte:

$$IGG = \sum IGI$$

Equação 1 - Equação do IGG

Para calcular o IGI precisamos calcular o produto da frequência relativa do defeito no pavimento e do fator ponderação, logo a equação do IGI é:

$$IGI = f_a * f_p$$

Equação 2 - Equação do IGI

A frequência relativa é a relação entre a quantidade de vezes que defeitos são verificados em um pavimento e o número N de pavimentos inventariados multiplicado por 100. Já para se determinar o valor do fator ponderação do pavimento utilizamos os valores definidos pela norma DNIT 006/2003 PRO (2003) na figura abaixo:

Ocorrência Tipo	Codificação de ocorrências de acordo com a Norma DNIT 005/2002-TER "Defeitos nos pavimentos flexíveis e semi-rígidos – Terminologia" (ver item 6.4 e Anexo D)	Fator de Ponderação fp
1	Fissuras e Trincas Isoladas (FI, TTC, TTL, TLC, TLL e TRR)	0,2
2	FC-2 (J e TB)	0,5
3	FC-3 (JE e TBE) NOTA:Para efeito de ponderação quando em uma mesma estação forem constatadas ocorrências tipos 1, 2 e 3, só considerar as do tipo 3 para o cálculo da freqüência relativa em percentagem (fr) e Índice de Gravidade Individual (IGI); do mesmo modo, quando forem verificadas ocorrências tipos 1 e 2 em uma mesma estação, só considerar as do tipo 2.	0,8
4	ALP, ATP e ALC, ATC	0,9
5	O, P, E	1,0
6	EX	0,5
7	D	0,3
8	R	0,6

Figura 7 - Fator Ponderação do Pavimento

2.4.6 Métodos de Priorização

A priorização das intervenções nos pavimentos tem como objetivo otimizar a utilização de recursos ordenando os casos com base na sua relevância. A partir do IGG podemos determinar o conceito de degradação do pavimento e, com ele, determinar quais pavimentos devem ser priorizados. Isso é feito a partir do padrão definido pela norma DNIT 006/2003 PRO (2003) apresentado na figura abaixo:

Conceitos	Limites
Ótimo	$0 < IGG \leq 20$
Bom	$20 < IGG \leq 40$
Regular	$40 < IGG \leq 80$
Ruim	$80 < IGG \leq 160$
Péssimo	$IGG > 160$

Figura 8 - Conceitos de Degradação do Pavimento

3. Trabalhos Relacionados

3.1 DDD: um Estudo de Caso

O trabalho desenvolvido por Barbosa (2021) tem como objetivo analisar os padrões existentes para aplicação do DDD a fim de analisar suas principais vantagens quando comparamos com padrões já existentes. Durante o desenvolvimento do protótipo a arquitetura de camadas proposta pelo DDD foi aplicada em um único contexto delimitado, nele foram implementadas quatro camadas: (1) camada de API onde ficam as interfaces e os controllers; (2) camada da aplicação que é responsável por receber as requisições da camada de API e direcioná-las para os casos de uso corretos; (3) camada de domínio que é onde estão as entidades, objetos de valor e as interfaces implementadas pela camada de infraestrutura; (4) camada de infraestrutura é responsável por pela comunicação com os recursos externos da aplicação como bancos de dados e APIs externas.

3.2 Desenvolvimento de Aplicações Utilizando DDD e Cloud Computing

O trabalho desenvolvido por Matheus Vieira (2023) tem como objetivo utilizar os conceitos de DDD no desenvolvimento de software e criar a infraestrutura necessária para disponibilizá-lo na nuvem. O sistema foi desenvolvido utilizando a linguagem Kotlin para o back-end e o framework Angular para o front-end, além de terem sido criados objetos importantes do DDD como uma linguagem ubíqua, contextos delimitados e um mapa de contexto. Além disso, foram utilizados recursos de nuvem da AWS como: DynamoDB, RDS, *Elastic Beanstalk* e ECR.

3.3 Model-Driven Engineering e Desenvolvimento Orientado ao Domínio para Suporte ao Design de Microsserviços

No trabalho de Roger Schmidt (2018) é proposto um conjunto de elementos que oferecem suporte ao design de sistemas de microsserviços. A abordagem proposta estabelece uma definição clara sobre os níveis de abstração visando formalizar o design de software no ambiente de desenvolvimento, sendo composta por cinco etapas: produção do modelo orientado

a eventos do domínio, modelagem estrutural DDD, produção de representações do DDD, geração de estruturas e por fim a implementação dos microsserviços.

3.4 Projetando Aplicações Baseadas em Microsserviços Utilizando a Abordagem DDD

Roland H. Steinegger et. al. (2017) em seu trabalho introduz as atividades necessárias para desenvolver um sistema baseado em microsserviços de acordo com os requisitos do DDD, além de mostrar sua aplicação através de um sistema de gerenciamento de teses e as limitações associadas a aplicação de DDD em sistemas baseados em microsserviços.

3.5 Pode DDD nos Levar a Encontrar a Forma Ótima de Modularizar um Microsserviço?

Sistemas de informação estão cada dia mais migrando para cloud, por isso é necessário seguir os requisitos aplicados pelos padrões da nuvem como alta disponibilidade, escalabilidade e diminuição do tempo médio de recuperação. Em seu estudo Hulya Vural et. al. (2021) tem dois objetivos: investigar se DDD leva a melhor forma de modularizar um microsserviço e entender se coesão e acoplamento ajudam a identificar o melhor contexto delimitado. Nos casos de estudo utilizados no trabalho foi possível verificar que DDD levou à melhor forma de modularizar, quanto ao segundo objetivo foi possível constatar que ao medimos os valores de acoplamento e coesão podemos encontrar o melhor contexto delimitado dividindo-o no ponto de menor acoplamento e maior coesão.

3.6 Projetando um Sistema de Saúde Online Utilizando DDD em Arquitetura de Microsserviços

O trabalho desenvolvido por M Rizki et. al. (2021) coloca em prática a abordagem DDD em conjunto com a arquitetura de microsserviços para projetar um sistema de saúde online, partindo da análise de uma arquitetura monolítica que é usada pelo sistema legado e analisando os domínios do sistema para a criação dos microsserviços e resultando em vários modelos e designs de software.

3.7 Planejamento de Manutenção de Pavimentos de maneira Sustentável em Países em Desenvolvimento baseado em Sistema de Gerência de Pavimentos

Marco Montoya-Alcaraz et. al. (2019) desenvolve um SGP para a autoestrada Centinela-La Rumorosa, localizada no estado de Baja California no México, permitindo planejar e administrar a alocação de recursos de forma apropriada ao fazer uma manutenção e na tomada de decisões.

3.8 Implantação de Sistema de Gerência de Pavimentos

Em seu estudo, Jocerlan Pereira da Rocha et. al. (2019) implantou um SGP em um trecho da BR-135 no Maranhão. Durante o estudo foram levantados dados sobre a rodovia, esses dados foram aplicados no programa HDM-4 para ajudar nos processos de manutenção e restauração dos pavimentos, por fim, conclui-se que o SGP torna a tomada de decisões mais fácil no decorrer do tempo.

3.9 Plataforma Web para o Gerenciamento de Inspeção Rodoviária e Informação de Manutenção: Um Passo Preliminar para Sistemas de Gerência de Pavimentos Inteligentes

Em seu trabalho, Bosurgi et. al. (2022) propõe uma plataforma que permite a coleta, análise e visualização de dados em tempo real que são capturados por sensores colocados em trechos de rodovias, a plataforma tem como objetivo dar um diagnóstico das condições de um pavimento e propor estratégias de manutenção. Por fim, a plataforma foi utilizada em um caso de estudo realizado na *Autostrada A20*, localizada na Itália, através deste caso de estudo foi possível comprovar a eficiência e utilidade do sistema.

3.10 Sistema de Gerência de Pavimentos de Baixo Custo para Países em Desenvolvimento

O trabalho de Khahro et. al. (2021) tem como objetivo projetar modelo de SGP para ajudar na tomada de decisões sugerindo priorização de determinadas seções do pavimento que possuem um grau maior de prioridade, principalmente em países onde o orçamento é limitado.

Ao fim do trabalho, conclui-se que o *Analytical Network Process* (ANP) têm diferentes pesos para diferentes processos de manutenção e reparo.

3.11 Plataforma em Nuvem para Coletar e Processar Dados de Múltiplos Sensores em Pavimentos

O trabalho de Fabrizio de Vita et. al. (2021) desenvolve um Sistema de Informação Geográfica (SIG) capaz de coletar e analisar dados de forma dinâmica, o sistema é dividido em quatro camadas: sensores, computação, proposta de manutenção e visualização. (1) A camada de sensores é responsável pela coleta de dados; (2) a camada de computação é responsável por analisar os dados para prover um entendimento melhor do estado do pavimento, além disso, a camada também tem uma inteligência artificial capaz de fazer previsões relacionadas à análise de pavimentos; (3) a terceira camada faz a sugestão de intervenções baseando-se no estado atual do pavimento; (4) a camada de visualização permite que os dados coletados sejam visualizados em tempo real. Essas camadas são divididas em duas partes: a borda, que seria a parte física do sistema e engloba a camada de sensores, e a nuvem, que engloba as outras três camadas e são implementadas utilizando soluções de nuvem.

3.12 Discutindo os Trabalhos Relacionados

Os trabalhos citados nas seções anteriores contribuíram para o desenvolvimento do trabalho descrito neste documento, as contribuições de cada um são:

- Os trabalhos de Barbosa (2021). Hippchen et. al. (2017) e Rizki et. al. (2021) contribuíram para o melhor entendimento de como aplicar técnicas de DDD e microsserviços na prática,;
- O trabalho Vieira (2023) assim como os discutidos no tópico anterior ajudou na aplicação de técnicas DDD, mas além disso, ele também mostra como criar uma infraestrutura em cloud para suportar um software;
- A utilização da abordagem proposta em Schmidt (2018) facilita o design e implementação de softwares que utilizam técnicas de DDD e microsserviços;
- O trabalho Vural (2021) ajuda a fazer uma divisão de contextos delimitados de forma mais eficiente medindo acoplamento e coesão;

- O trabalho de Montoya-Alcaraz et. al. (2019) mostra o processo de criação de um modelo de SGP feito para uma rodovia específica do México;
- O trabalho Rocha et. al. (2019) aplica de forma prática o HDM-4 em uma rodovia brasileira, mostrando o processo de coleta de dados e tomada de decisões;
- Os trabalhos de Bosurgi et. al. (2022), Khahro et. al. (2021) e De Vita et. al. (2021) tratam do mesmo tipo de sistema desenvolvido neste trabalho, ajudando assim, na escolha de tecnologias;

4. Proposta do Sistema

4.1 Domínio do Sistema

A aplicação desenvolvida neste projeto é um sistema de gerência de pavimentos (SGP), tendo como principal objetivo ajudar na tomada de decisões quanto às ações a serem tomadas para repará-las ou renová-las de forma a otimizar a utilização de recursos.

4.2 Requisitos Funcionais do Sistema

Para a elicitação de requisitos funcionais do sistema foi utilizada a técnica de *brainstorm* tomando como base funcionalidades básicas de uma aplicação web e as principais características de um SGP segundo Hudson et. al. (2015). Os requisitos elicitados foram:

[RF-001] Cadastro de Usuários

A aplicação deve permitir o cadastro de novos usuários no sistema.

[RF-002] Autenticação de Usuários

A aplicação deve permitir que usuários façam login desde que os dados sejam válidos.

[RF-003] Recuperação de Senha

A aplicação deve permitir que o usuário altere sua senha caso ele tenha se esquecido.

[RF-004] Atualização de Dados Cadastrais

A aplicação deve permitir que o usuário atualize seus dados cadastrais.

[RF-005] Cadastrar Orçamentos

O usuário deve ser capaz de cadastrar o valor do orçamento disponível para a empresa/órgão governamental que ele faz parte.

[RF-006] Cadastro de Pavimentos

A aplicação deve permitir que novos pavimentos sejam cadastrados no banco de dados.

[RF-007] Atualização de Estado dos Pavimentos

A aplicação deve permitir a atualização de dados dos pavimentos

[RF-008] Consulta de Pavimentos

A aplicação deve permitir que usuários pesquisem pavimentos no banco de dados

[RF-009] Calcular Degradação

A aplicação deve utilizar os dados coletados por sensores para o cálculo do índice de condição do pavimento (IGG).

[RF-010] Calcular Criticidade

A aplicação deve calcular o grau de criticidade de um pavimento levando como base o IGG.

[RF-011] Gerar Relatório de Pavimentos

A aplicação deve ser capaz de gerar um relatório de estado dos pavimentos cadastrados no banco de dados, o relatório deve levar a criticidade e o orçamento disponível.

[RF-012] Sugerir Intervenções

A aplicação deve ser capaz de sugerir possíveis intervenções para solucionar os problemas de determinado pavimento.

[RF-013] Utilizar Mapas Temáticos

A aplicação deve utilizar mapas temáticos para uma melhor visualização de pavimentos e sua distribuição na rede.

[RF-014] Permitir a Importação de Dados

A aplicação deve permitir a importação de dados através de arquivos (xml, txt, etc) para facilitar sua utilização.

[RF-015] Auto-Aprendizado

A aplicação deve utilizar intervenções anteriores para aprender e dar sugestões mais precisas.

[RF-016] Atualização de Dados

Mensalmente a aplicação deve fazer uma atualização de dados relacionados aos pavimentos no banco de dados.

[RF-017] Atualização de Imagem de Perfil

O usuário deve ser capaz de atualizar sua imagem de perfil.

[RF-018] Upload de Imagens de Pavimentos

O sistema deve permitir o upload de imagens dos pavimentos.

4.3 Requisitos Não Funcionais

Assim como os requisitos funcionais a elicitação de requisitos não funcionais foi feita utilizando *brainstorm*, além disso, como base foi utilizada a taxonomia unificada para requisitos não funcionais de Benitti e Rhoden (2015).

[RNF-001] Tempo de Timeout

O sistema deve responder às requisições em até 30 segundos.

[RNF-002] Réplicas de Alta Disponibilidade

O banco de dados do sistema deve possuir réplicas de alta disponibilidade em *datacenters* diferentes para diminuir a possibilidade de falhas relacionadas a desastres naturais.

[RNF-003] Restrição de Formato

O sistema não deve ser *case-sensitive.*

[RNF-004] Capacidade de Armazenamento

O sistema deve ter uma capacidade de armazenamento de 1TB.

[RNF-005] Capacidade de Processamento

O sistema deve ter 8 processadores disponíveis.

[RNF-006] Quantidade Máxima de Erros Causados por Modificações

O sistema deve ter uma quantidade de erros causados por modificações inferiores a 5.

[RNF-007] Extensibilidade do Sistema

O sistema deve permitir adicionar ou remover novas funcionalidades sem a necessidade de ser reiniciado.

[RNF-008] Registro de Falhas

O sistema deve registrar os erros ocorridos durante sua execução.

[RNF-009] Controle de Mudanças

Mudanças no sistema devem ser facilmente identificadas.

[RNF-010] Utilização de Microsserviços

O sistema deve utilizar microsserviços para executar determinadas operações.

[RNF-011] Testes Unitários

Todos os microsserviços do sistema devem estar cobertos por testes unitários.

[RNF-012] Modulariade

Mudanças em microsserviços não devem afetar outros microsserviços.

[RNF-013] Validade de Autenticação

A autenticação do usuário deve ser válida por no máximo 48 horas.

[RNF-014] Backup de Dados

O sistema deve fazer backup diário dos dados armazenados no banco de dados.

[RNF-015] Criptografia de Senhas

Senhas de usuários do sistema devem ser criptografadas quando armazenadas.

[RNF-016] Armazenamento de Imagens

As imagens dos pavimentos devem ser armazenadas em um disco separado de 1TB.

4.4 Linguagem Ubíqua do Sistema

Para a utilização da linguagem ubíqua foi criado um glossário com os principais termos descritos nas funcionalidades do sistema, isso se faz necessário para garantir que termos não tenham significados diferentes para contextos diferentes, facilitando assim, a comunicação entre desenvolvedores e os especialistas de domínio. Com base nisso, os seguintes termos foram escolhidos para formar a linguagem ubíqua:

- Usuário: refere-se ao tomador de decisões que está utilizando o sistema;
- Pavimento: refere-se a uma parte do revestimento que está sendo analisada;
- Intervenção: refere-se às ações que podem ser tomadas para solucionar os defeitos no pavimento, podendo ser ações de renovação ou manutenção;
- Defeito: refere-se às imperfeições presentes no revestimento que aumentam seu grau de degradação;
- Grau de degradação: é um valor referente a qualidade do pavimento no momento da análise;
- Condição dos pavimentos: refere-se ao estado do pavimento no momento da coleta dos dados, é possível ter acesso à informações sobre os defeitos, intervenções e criticidade do pavimento;
- Análises: refere-se à análise feita por especialistas para determinar os defeitos de um pavimento;

4.5 Contextos Delimitados do Sistema

Para definir os contextos delimitados do sistema precisamos olhar para os requisitos descritos na seção 3.2, a partir disso, podemos agrupá-los baseando-se no núcleo de cada

requisito. Com isso, podemos perceber que a aplicação possui três domínios: usuário, pavimentos e intervenção.

O domínio de pavimentos é o domínio principal do sistema, pois sem ele a aplicação não funcionaria e grande parte das funcionalidades fazem parte dele. Nele, além do contexto delimitado de pavimentos, também temos dois sistemas externos: o sistema de mapas temáticos e o sistema de sensores inteligentes que se comunicam com o contexto de pavimentos para facilitar a visualização dos pavimentos e tornar possível a obtenção de dados em tempo real e de forma automática.

Já o domínio de intervenções funciona como um domínio secundário, ele implementa funcionalidades importantes para o sistema mas ele é de menor importância quando comparado ao de pavimentos.

Já o domínio de usuários é um domínio genérico que serve para permitir que pessoas autorizadas utilizem o sistema, nele também temos um sistema externo, o sistema de envio de e-mails, ele se comunica com o contexto de usuário para enviar os e-mails de recuperação de senha.

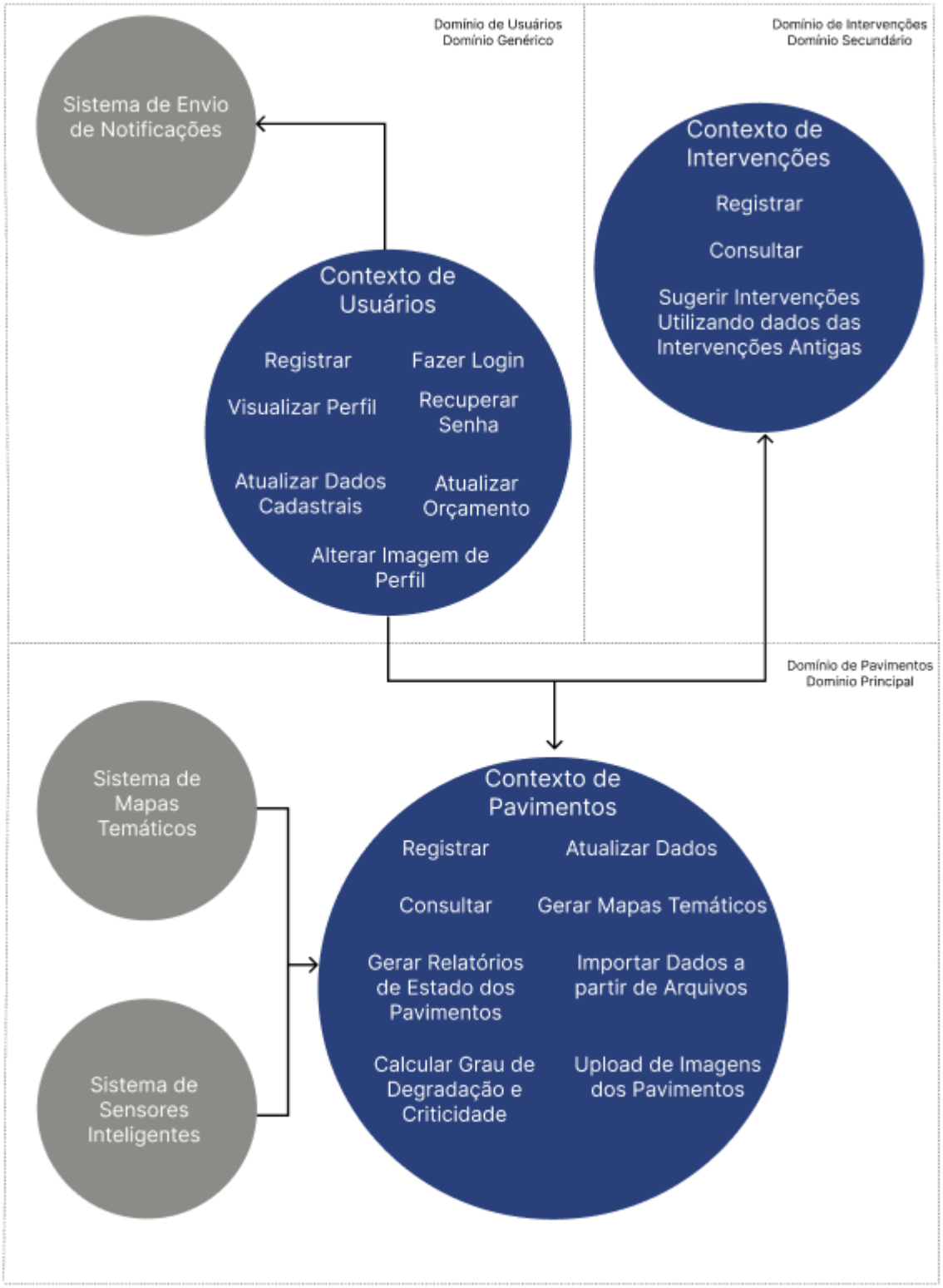

Figura 9 - Domínios do Sistema

4.6 Mapa de Contexto do Sistema

Utilizando o mapa de contexto podemos definir os relacionamentos entre os contextos delimitados do domínio, na imagem abaixo temos a representação do mapa de contexto do sistema descrito neste trabalho. O contexto delimitado de pavimentos tem um relacionamento produtor-consumidor com o de usuários, já que pavimentos produz os dados que serão utilizados pelos usuários. Já o relacionamento entre os contextos delimitados de pavimentos e intervenções funcionam de forma parecida, porém pavimentos é o consumidor enquanto intervenções é o produtor. Os dados retornados pelo domínio de intervenções serão consumidos pelo domínio de pavimentos para geração de relatórios, por isso, uma camada de anticorrupção é utilizada para adequar os dados.

Em cinza temos os contextos delimitados pertencentes aos sistemas externos, eles são o sistema de sensores inteligentes, sistema de mapas temáticos e o sistema de envio de notificações. Os sistemas de sensores e mapas possuem um relacionamento produtor-consumidor com o contexto de pavimentos, enquanto o contexto de usuários também possui um relacionamento produtor-consumidor, mas com o sistema de envio de e-mails.

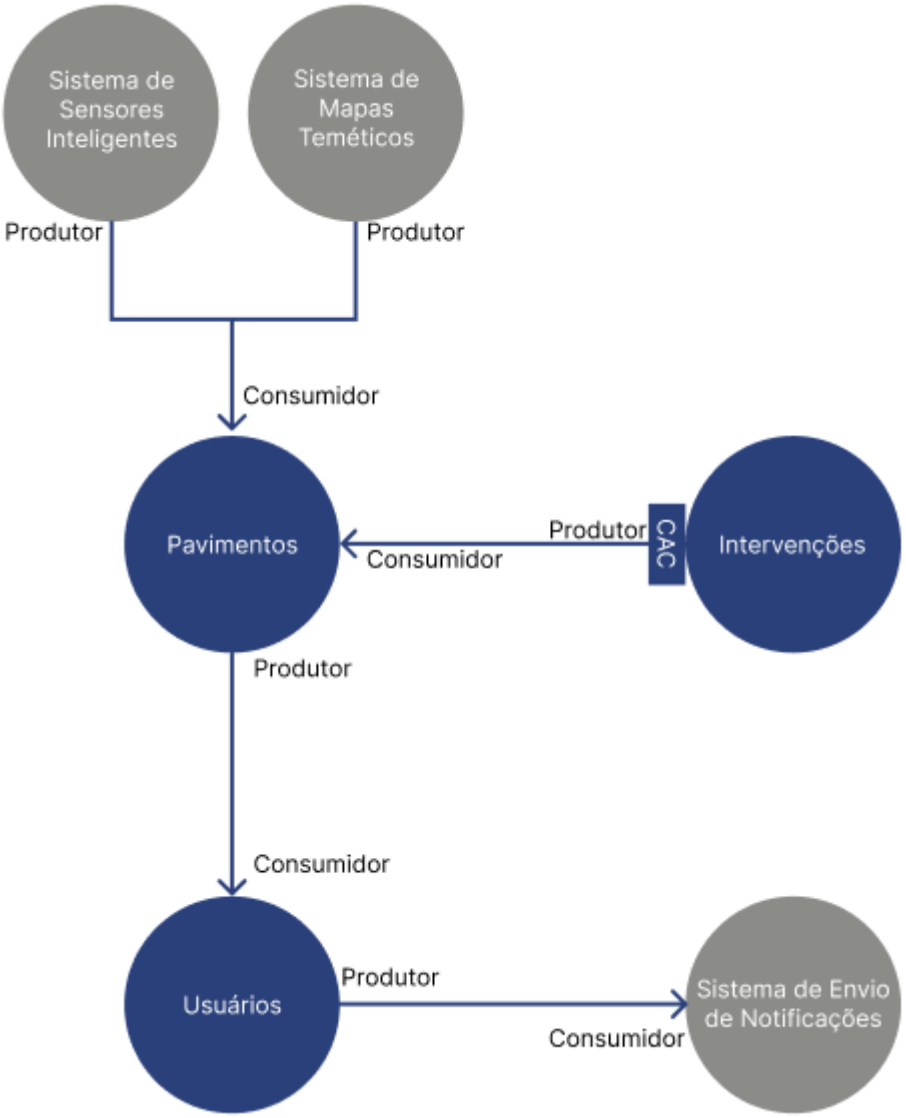

Figura 10 - Mapa de Contexto

5. Apresentação da Arquitetura do Sistema

5.1 Modelo C4

O modelo C4 é uma abordagem que prioriza a abstração durante a diagramação da arquitetura de software (Brown, 2019), ele é composto por 4 níveis diferentes: nível de contexto, nível de container, nível de componentes e nível de código.

Nas próximas seções serão apresentados diagramas que representam os níveis do sistema proposto neste trabalho, seguindo o modelo C4 três dos quatro níveis tiveram seus diagramas desenvolvidos: contexto, container e componentes, o diagrama de código, porém, não foi projetado por ser um nível opcional e que pode ser gerado automaticamente por ferramentas como IDEs. Além disso, os links de acesso para cada um dos diagramas podem ser encontrados no apêndice A deste trabalho.

5.1.1 Diagrama de Contexto

O diagrama de contexto está no nível de maior abstração do modelo C4, nele o sistema principal deve estar no centro e ao seu redor devem estar os usuários e sistemas externos que com ele interagem.

Abaixo temos o diagrama de contexto do sistema, na imagem temos três tipos de objetos diferentes: (1) o objeto que se encontra no centro da imagem representa o sistema principal, um Sistema de Gerência de Pavimentos (SGP); (2) o usuário, que se encontra no topo do diagrama, faz requisições ao sistema principal e recebe respostas; (3) e os sistemas externos, evidenciados na cor cinza, são sistemas que não fazem parte do sistema principal mas são usados por ele para atender a requisitos como: recuperação de senha, utilização de mapas temáticos e coleta de dados automática.

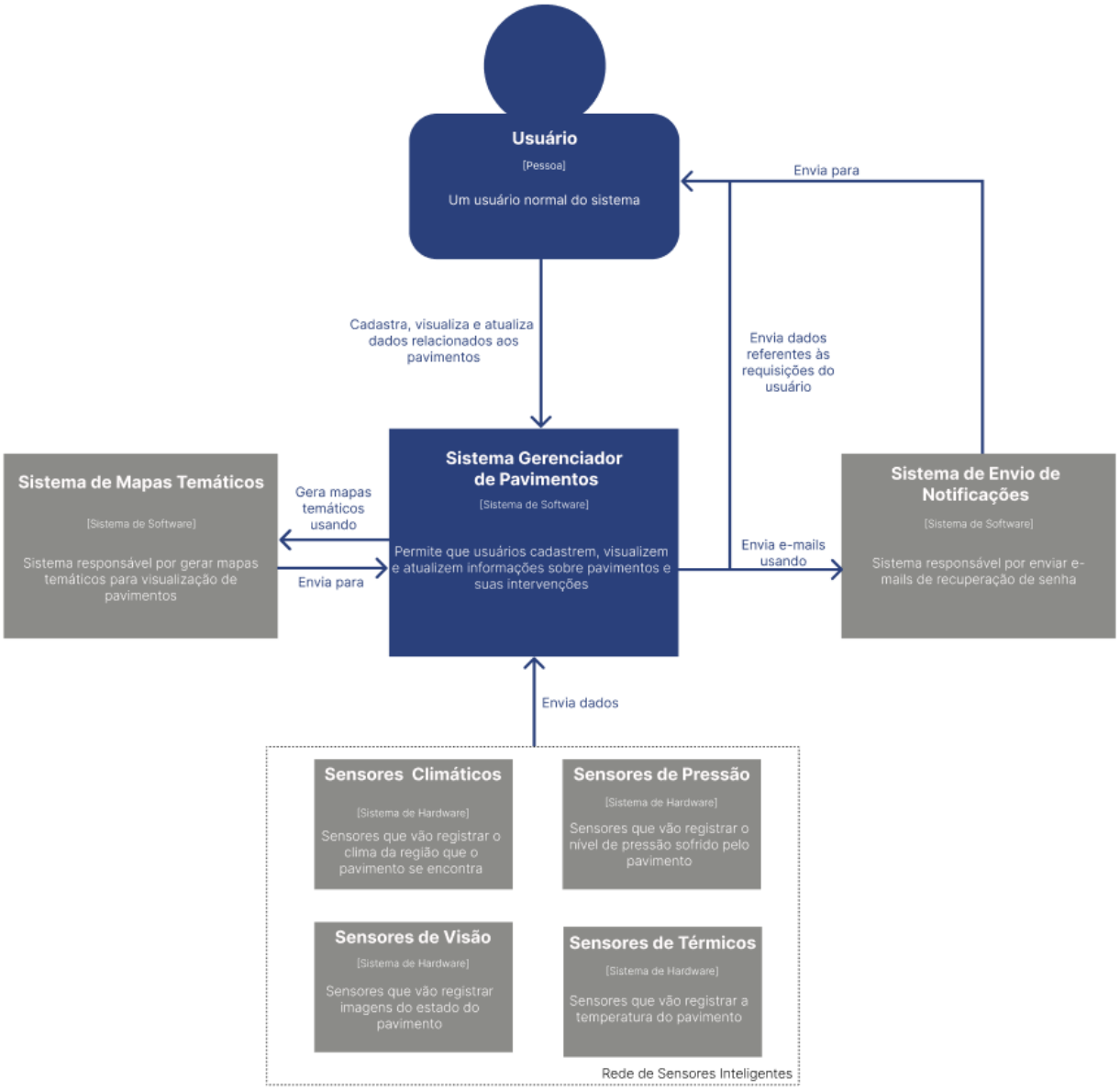

Figura 11 - Diagrama de Contexto do Sistema

5.1.2 Diagrama de Container

O diagrama de container é o segundo nível do modelo C4, nele fazemos um “zoom” no diagrama de contexto com o objetivo de definir os containers de um sistema, um container pode ser um aplicativo desktop, aplicativo mobile, sistema de página única, etc. Este diagrama é responsável por mostrar o formato da arquitetura de software, como as responsabilidades são distribuídas no sistema, as tecnologias utilizadas e como os containers se comunicam uns com os outros.

Abaixo temos o diagrama de container do sistema, aplicamos o zoom no sistema principal do diagrama apresentado na seção anterior. Na imagem podemos ver que o usuário utiliza o *frontend* do sistema através métodos HTTP e o frontend faz requisições para a API, também utilizando métodos HTTP, de acordo com as funcionalidades que o usuário deseja acessar. O container da API e o container do banco de dados formam o *backend* do sistema, sendo responsáveis por implementar as funcionalidades do sistema, fazer a integração dos sistemas externos com o restante da aplicação e armazenar os dados necessários para o funcionamento do sistema.

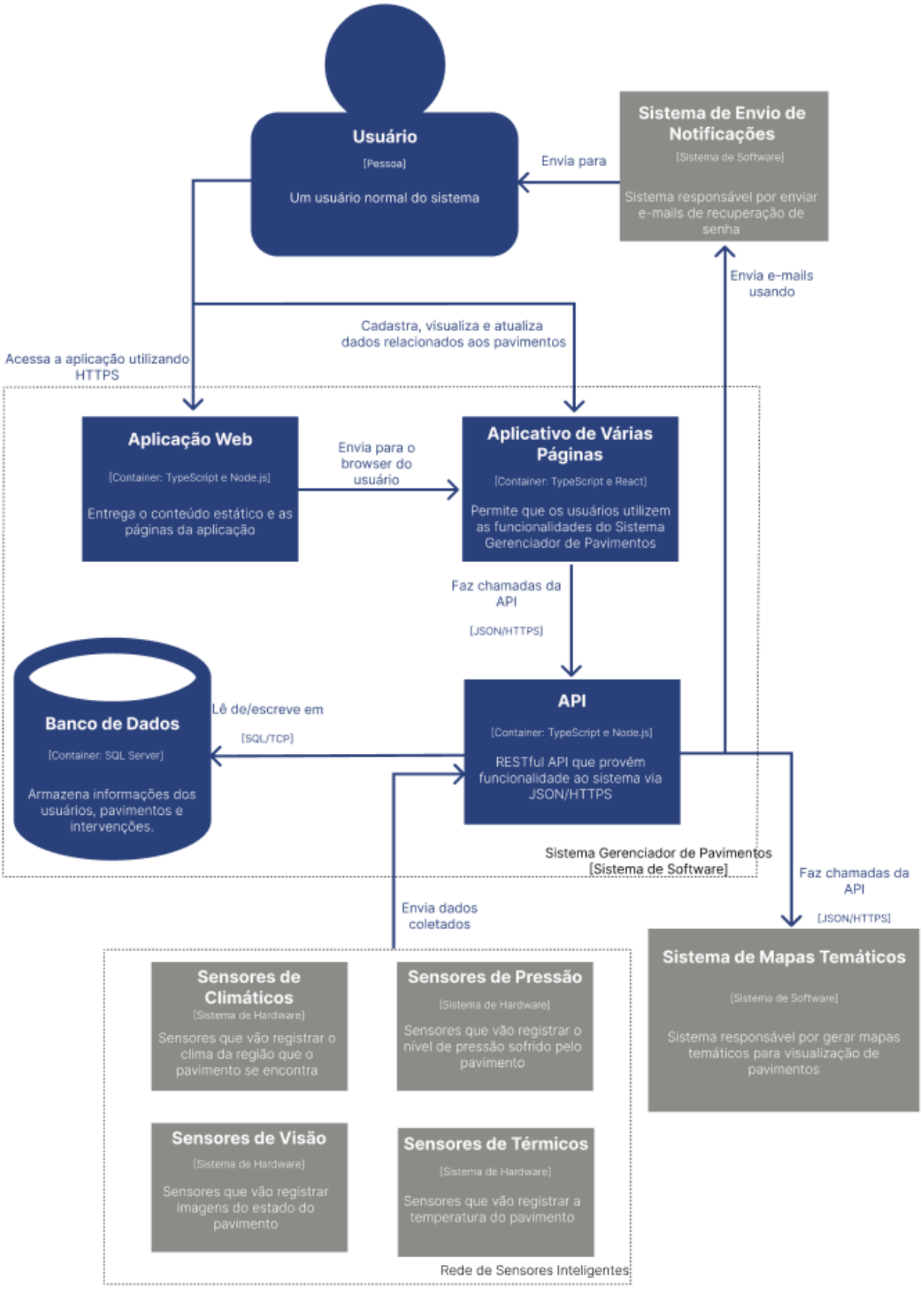

Figura 12 - Diagrama de Container

5.1.3 Diagrama de Componentes

O diagrama de componentes é o terceiro nível do modelo C4, nele nós decompomos cada container da aplicação em componentes e especificamos suas responsabilidades, suas tecnologias e implementações.

O diagrama de componentes do sistema foi dividido em três subsistemas para facilitar a visualização e entendimento, eles são: usuários, pavimentos e intervenções. Além disso, cada diagrama mostra o relacionamento entre os componentes de cada subsistema, bem como suas funções.

5.1.3.1 Diagrama do Subsistema de Usuários

Na imagem abaixo podemos ver o diagrama do subsistema de usuários, nele estão representados os controladores, microsserviços, repositório, banco de dados e sistemas externos do subsistema de usuários. Cada componente do sistema pode se relacionar com um ou mais componentes e a leitura ou escrita de dados no banco de dados é feita única e exclusivamente através do repositório do subsistema, nele estão implementados todos os métodos necessários para que os microsserviços consigam realizar seus papéis corretamente. Os relacionamentos dos componentes desse subsistema podem ser descritos como:

- Controlador de usuários: implementa três métodos HTTPs: POST, GET e PATCH.
    - o método POST utiliza o microsserviço de criação de usuários, nele está implementada a lógica para criação de usuários no sistema;
    - o método GET utiliza o microsserviço de visualização de perfil para buscar no banco de dados as informações de um usuário já existente no sistema;
    - o método PATCH utiliza o microsserviço de atualização de dados para buscar e atualizar os dados de um usuário existente;
- Controlador de login: implementa um método POST que autentica usuários no sistema através do microsserviço de autenticação;

- Controlador de atualização de imagem de perfil: implementa um método PATCH, ele utiliza um microsserviço que faz upload da imagem selecionada pelo usuário e atualiza o perfil do usuário com a nova imagem;
- Controlador de recuperação de senha: implementa um método POST que vai usar o microsserviço de recuperação de senha, este microsserviço faz uso de um sistema externo para envios de e-mail com as instruções para a recuperação de senha para o usuário;
- Controlador de atualização de orçamento: que implementa um método PATCH, ele usa um microsserviço para atualizar o saldo ligado ao usuário.

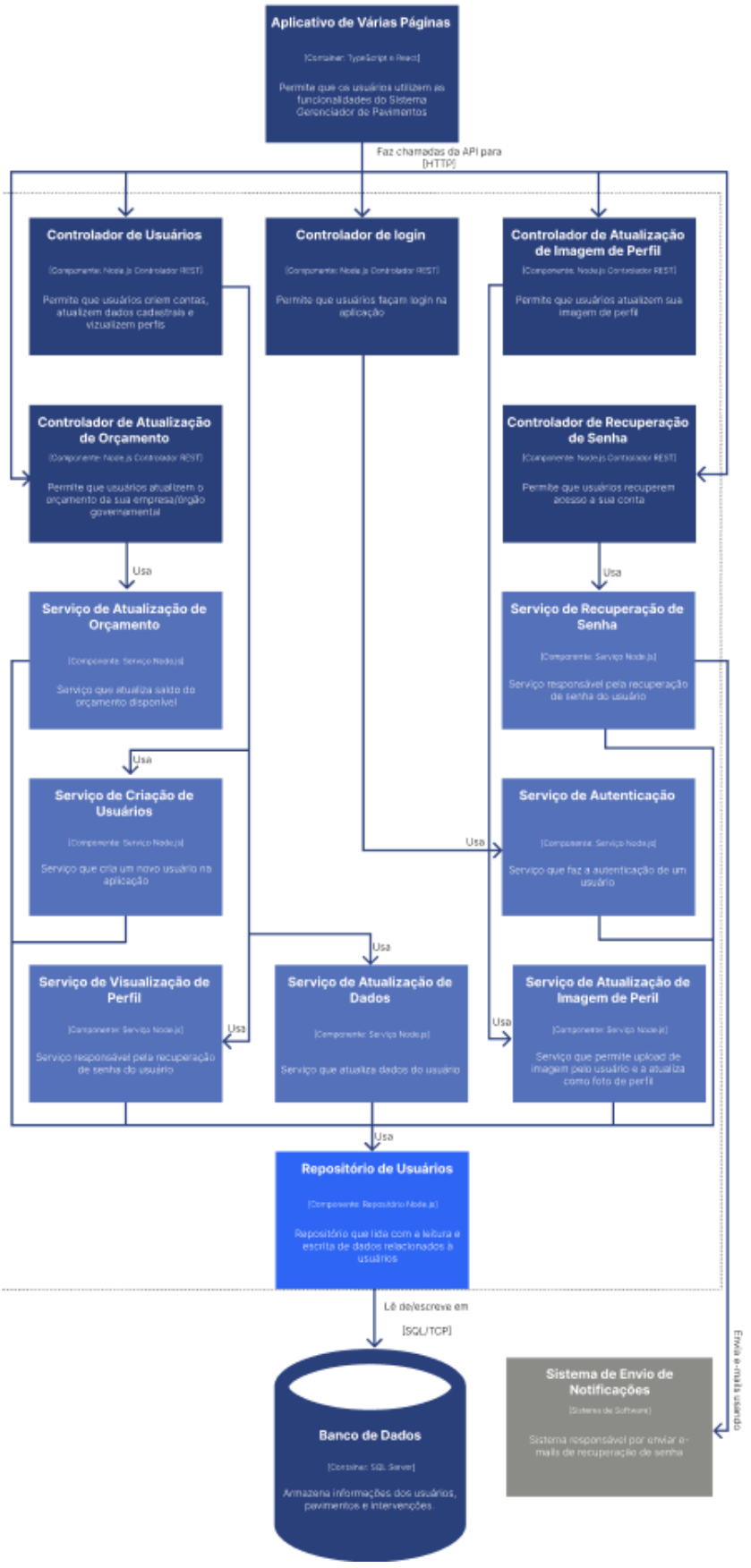

Figura 13 - Diagrama de Componentes do Subsistema de Usuários

5.1.3.2 Diagrama do Subsistema de Pavimentos

Na imagem abaixo podemos ver o diagrama do subsistema de pavimentos, ele segue os mesmos padrões do diagrama apresentado na seção anterior. Os relacionamentos dos componentes desse subsistema podem ser descritos como:

- Controlador de pavimentos: assim como o controlador de usuários ele implementa três métodos HTTPs: POST, GET e PATCH.
    - No método POST é feita a criação de novos pavimentos utilizando um microsserviço para a criação, por ele são utilizados outros microsserviços para calcular o IGG do pavimento, determinar o grau de criticidade do pavimento e fazer upload das imagens;
    - No método GET é feita a busca de pavimentos no banco de dados utilizando o microsserviço de pesquisa de pavimentos;
    - No método PATCH é utilizado um microsserviço de atualização de pavimentos para atualizar os dados relacionados a um determinado pavimento;
- Controlador de importação de dados: implementa um método POST onde é utilizado o microsserviço de importação, através dele é possível importar dados de um arquivo específico (.txt, .csv, etc) para o banco de dados;
- Controlador de relatórios: implementa um método GET que faz uso de um microsserviço de geração de relatórios, eles são gerados com base nos parâmetros passados pelo usuário;
- Controlador de mapas temáticos: implementa um método GET, ele faz uso de um microsserviço de mapas temáticos que vai buscar os dados de determinado pavimento no banco de dados e, com a ajuda de um sistema externo, gera mapas para facilitar a visualização dos pavimentos;
- Controlador de atualizações automáticas implementa um método POST, este controlador não deve ser acessado pelo usuário e deve ser utilizado mensalmente pelo sistema para atualizar os dados dos pavimentos já existentes no banco de dados utilizando o microsserviço para atualização de dados dos pavimentos. Além disso, caso os sensores sejam instalados em um pavimento novo o controlador deverá usar o microsserviço para a criação de novos pavimentos.

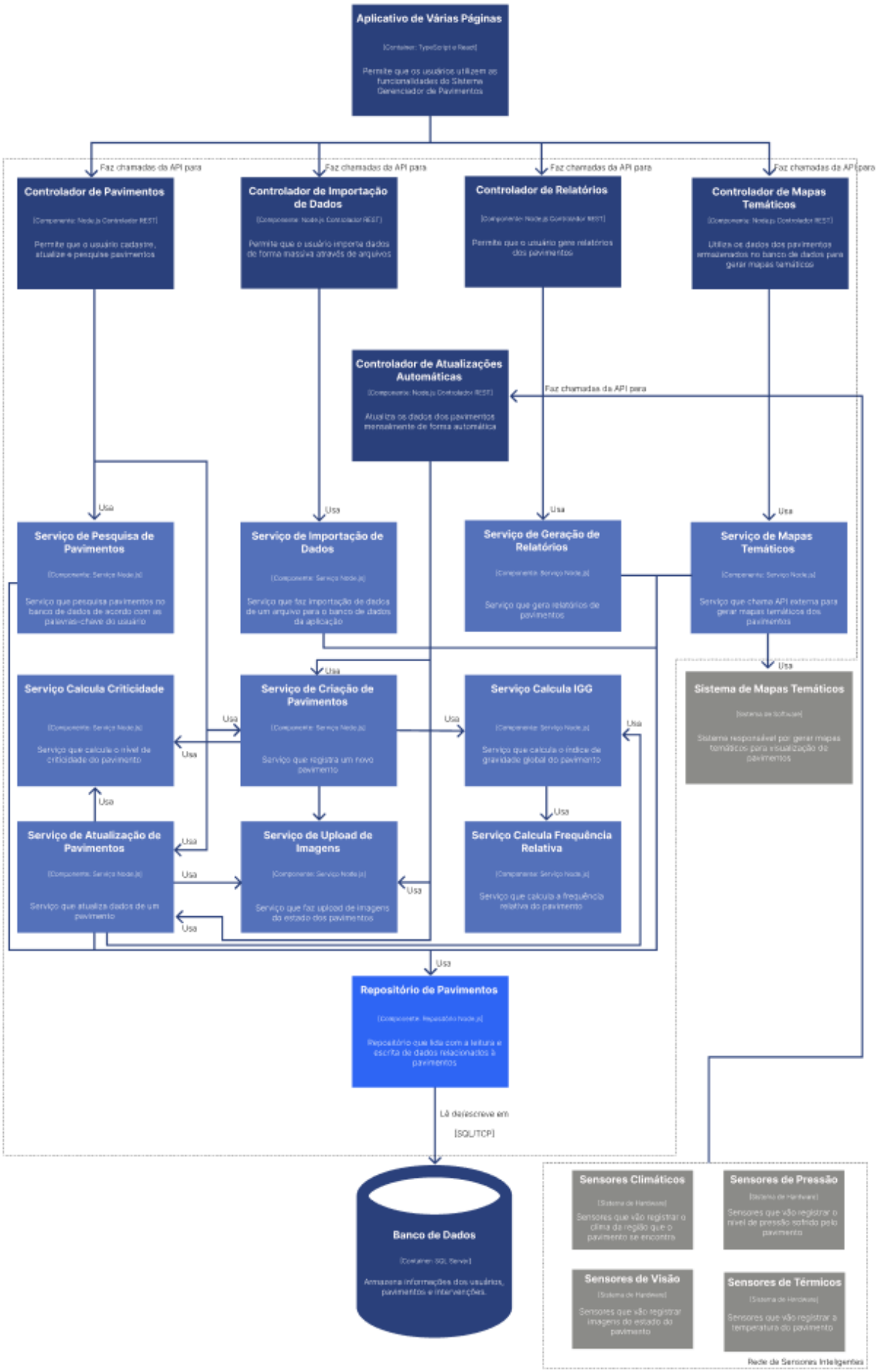
Aplicativo de Várias Páginas
[Container: TypeScript e React]
Permite que os usuários utilizem as funcionalidades do Sistema Gerenciador de Pavimentos
Faz chamadas da API para
Faz chamadas da API para
Faz chamadas da API para
Faz chamadas da API para
Controlador de Pavimentos
[Componente: Node.js Controlador REST]
Permite que o usuário cadastre, atualize e pesquise pavimentos
Controlador de Importação de Dados
[Componente: Node.js Controlador REST]
Permite que o usuário importe dados de forma massiva através de arquivos
Controlador de Relatórios
[Componente: Node.js Controlador REST]
Permite que o usuário gere relatórios dos pavimentos
Controlador de Mapas Temáticos
[Componente: Node.js Controlador REST]
Utiliza os dados dos pavimentos armazenados no banco de dados para gerar mapas temáticos
Controlador de Atualizações Automáticas
[Componente: Node.js Controlador REST]
Atualiza os dados dos pavimentos mensalmente de forma automática
Faz chamadas da API para
Usa
Usa
Usa
Usa
Serviço de Pesquisa de Pavimentos
[Componente: Serviço Node.js]
Serviço que pesquisa pavimentos no banco de dados de acordo com as palavras-chave do usuário
Serviço de Importação de Dados
[Componente: Serviço Node.js]
Serviço que faz importação de dados de um arquivo para o banco de dados da aplicação
Serviço de Geração de Relatórios
[Componente: Serviço Node.js]
Serviço que gera relatórios de pavimentos
Serviço de Mapas Temáticos
[Componente: Serviço Node.js]
Serviço que chama API externa para gerar mapas temáticos dos pavimentos
Usa
Usa
Usa
Usa
Usa
Usa
Usa
Usa
Usa
Usa
Usa
Usa
Usa
Usa
Usa
Serviço Calcula Criticidade
[Componente: Serviço Node.js]
Serviço que calcula o nível de criticidade do pavimento
Serviço de Criação de Pavimentos
[Componente: Serviço Node.js]
Serviço que registra um novo pavimento
Serviço Calcula IGG
[Componente: Serviço Node.js]
Serviço que calcula o índice de gravidade global do pavimento
Sistema de Mapas Temáticos
[Sistema de Software]
Sistema responsável por gerar mapas temáticos para visualização de pavimentos
Serviço de Atualização de Pavimentos
[Componente: Serviço Node.js]
Serviço que atualiza dados de um pavimento
Serviço de Upload de Imagens
[Componente: Serviço Node.js]
Serviço que faz upload de imagens do estado dos pavimentos
Serviço Calcula Frequência Relativa
[Componente: Serviço Node.js]
Serviço que calcula a frequência relativa do pavimento
Usa
Repositório de Pavimentos
[Componente: Repositório Node.js]
Repositório que lida com a leitura e escrita de dados relacionados à pavimentos
Lê de/escreve em
[SQL/TCP]
Banco de Dados
[Container: SQL Server]
Armazena informações dos usuários, pavimentos e intervenções.
Sensores Climáticos
[Sistema de Hardware]
Sensores que vão registrar o clima da região que o pavimento se encontra
Sensores de Pressão
[Sistema de Hardware]
Sensores que vão registrar o nível de pressão sofrida pelo pavimento
Sensores de Visão
[Sistema de Hardware]
Sensores que vão registrar imagens do estado do pavimento
Sensores de Térmicos
[Sistema de Hardware]
Sensores que vão registrar a temperatura do pavimento
Rede de Sensores Inteligentes

Figura 14 - Diagrama de Componentes do Subsistema dos Pavimentos

5.1.3.3 Diagrama do Subsistema de Intervenções

Na imagem abaixo podemos ver o diagrama do subsistema de intervenções, ele segue os mesmos padrões dos diagramas apresentados nas seções anteriores. Os relacionamentos dos componentes desse subsistema podem ser descritos como:

- Controlador de intervenções: implementa dois métodos HTTPs: POST e GET.
    - O método POST usa um microsserviço para a criação de novas intervenções no banco de dados;
    - O método GET usa um microsserviço que faz a busca de intervenções que já estão armazenadas no banco de dados utilizando parâmetros passados pelo usuário como filtro
- Controlador de intervenções: implementa um método POST que faz uso do microsserviço de sugestão de intervenções, nele são consideradas intervenções passadas, o grau de prioridade de um pavimento e o orçamento ligado ao usuário que fez a solicitação para sugerir a melhor forma de intervenção.

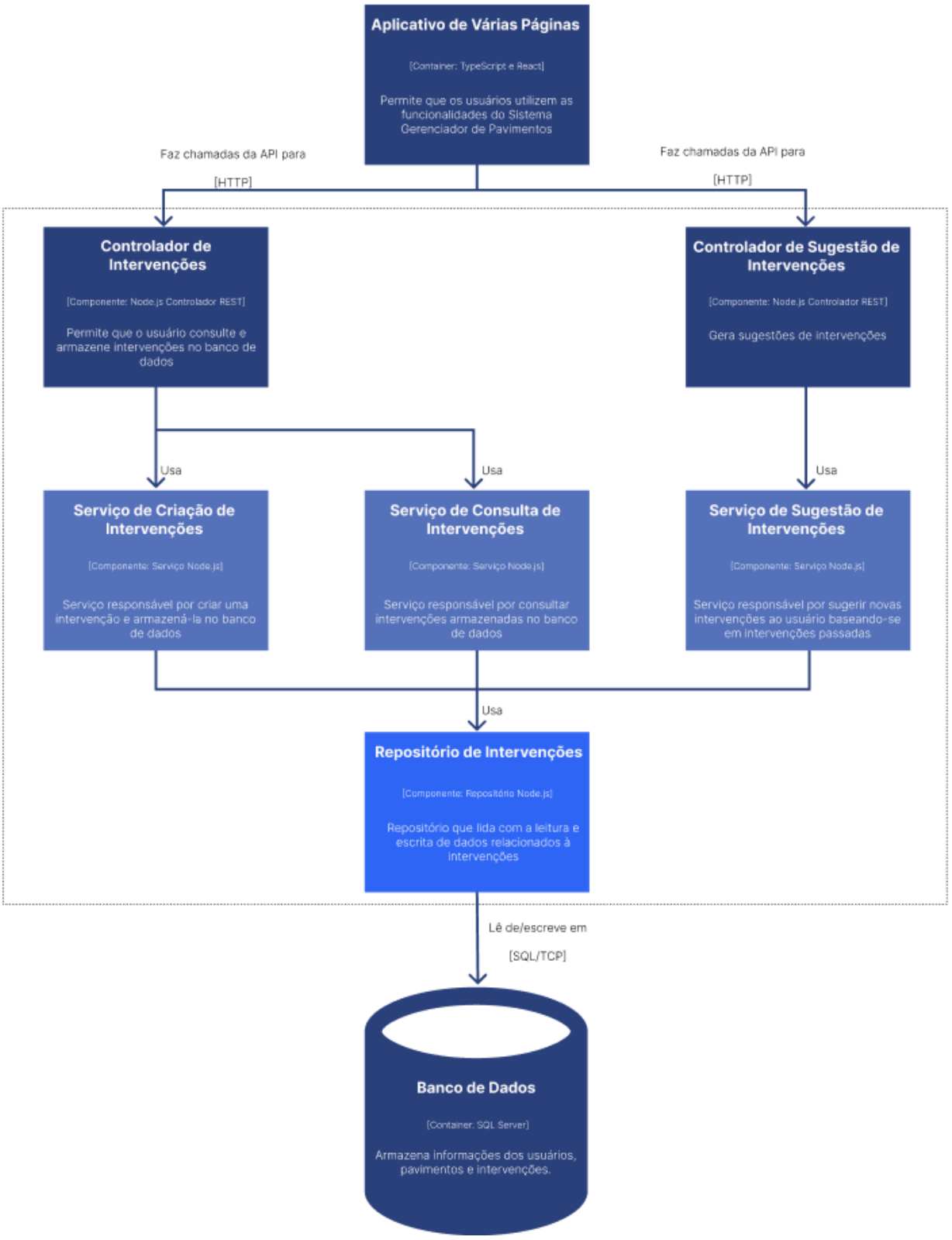

Figura 15 - Diagrama de Componentes do Subsistema de Intervenções

5.2 Modelo do Banco de Dados do Sistema

O banco de dados da aplicação possui ao todo seis tabelas: a de usuários, orçamentos, pavimentos, intervenções, defeitos e sugestões. Suas tabelas e relacionamentos estão descritos na figura abaixo:

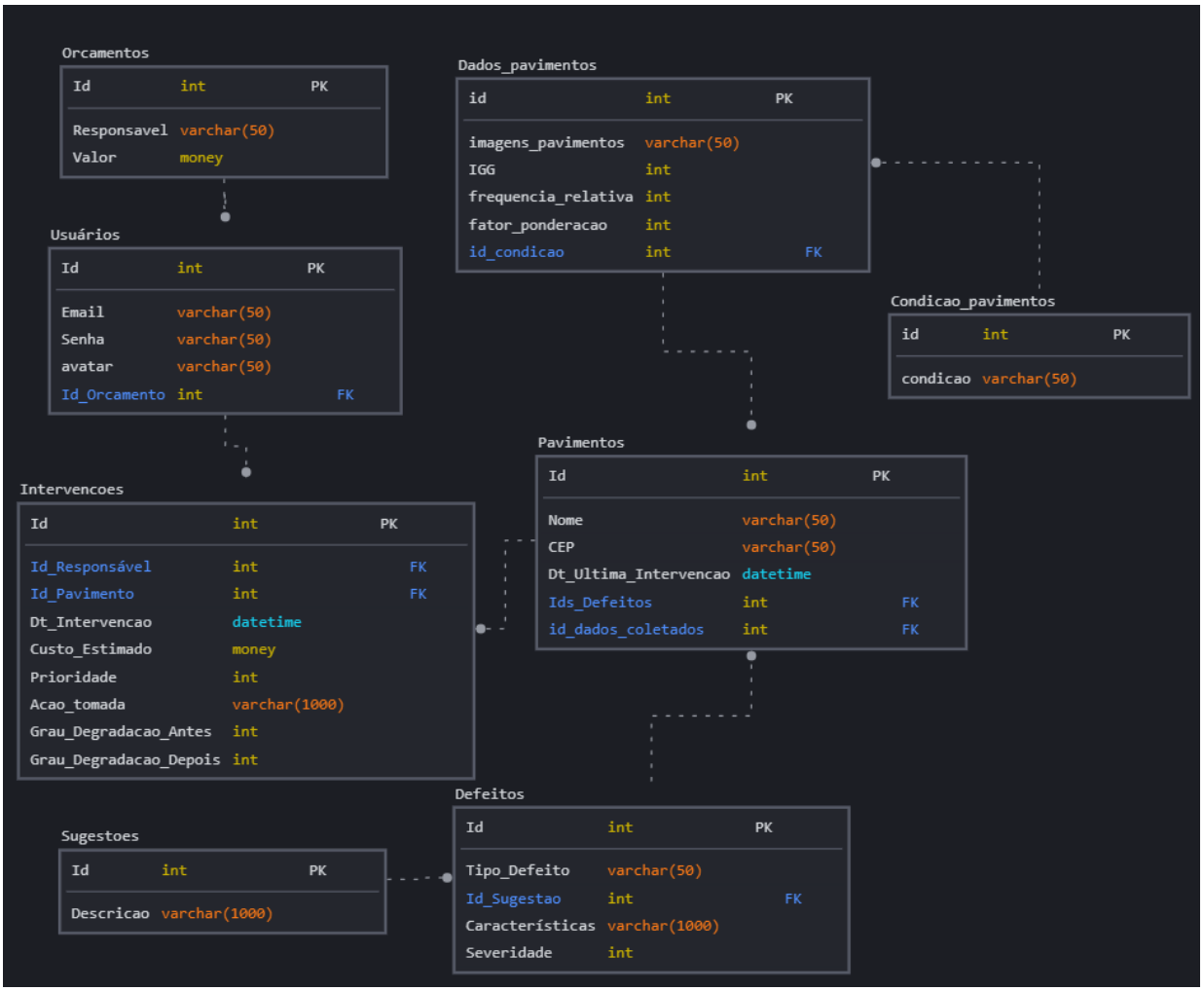

Figura 16 - Modelo do banco de dados

5.2.1 Tabela de Usuários

A tabela de usuários vai armazenar os dados dos usuários do sistema, apenas usuários cadastrados terão acesso às funcionalidades do SGP. A tabela possui cinco colunas: a coluna de id que funcionará como um identificador único para cada usuário, o campo email do tipo varchar que deve ser único, um campo senha do tipo varchar, cujos dados devem ser criptografados, um campo avatar do tipo varchar que faz referência ao caminho que a imagem

de perfil do usuário está armazenado e, por fim, uma chave estrangeira que faz referência à tabela de orçamentos para ligar o valor de orçamento disponível de determinada entidade ao usuário.

5.2.2 Tabela de Pavimentos

A tabela de pavimentos vai armazenar dados sobre os pavimentos que vão ajudar na tomada de decisões e priorização das intervenções. Nela vamos encontrar informações gerais sobre o pavimento como: nome, CEP e a data da última intervenção para o pavimento. Também temos duas chaves estrangeiras nesta tabela, elas são: Id_Defeitos que referência a tabela que armazena as definições dos defeitos dos pavimentos e Id_dados_coletados que referência a tabela que armazena os dados dos pavimentos coletados pelos sensores.

5.2.3 Tabela de Intervenções

Na tabela de intervenções vamos encontrar informações sobre cada intervenção de forma individual, nela vamos possuir informações sobre o pavimento que foi alvo da intervenção, seu responsável, a data da intervenção, o custo, prioridade, uma descrição da ação tomada, e o grau de degradação antes da ação e depois. Além disso, ela também ajudará no aprendizado do SGP, pois baseando-se nas decisões feitas no passado é possível sugerir ações com mais precisão no futuro.

5.2.4 Tabela de Defeitos

A tabela de defeitos funciona como um glossário dos defeitos descritos no manual de identificação de defeitos (SHRP, 2014), nela contém informações como o tipo do defeito, suas características, criticidade e um relacionamento com a tabela de sugestões de como resolvê-los.

5.2.5 Tabela de Sugestões

A tabela de sugestões, assim como a tabela de defeitos, funciona como um glossário das sugestões descritas no manual de identificação de defeitos (SHRP, 2014), nela contém informações sobre as ações sugeridas de como realizar a manutenção ou o reparo de um pavimento.

5.2.6 Tabela de Orçamentos

A tabela de orçamentos armazena os valores monetários associados ao órgão público ou empresa que determinado(s) usuários faz(em) parte. A partir desse valor podemos gerar relatórios de forma que as intervenções sugeridas não ultrapassem o valor disponível.

5.2.7 Tabela de Dados dos Pavimentos

A tabela de dados dos pavimentos armazena os dados coletados pelos sensores instalados nos pavimentos, nela é armazenado o caminho para a pasta que estão armazenadas as imagens dos pavimentos, o índice de gravidade global (IGG), a frequência relativa, o fator de ponderação. A tabela também faz referência a tabela de condições do pavimento que vai armazenar a classificação da condição do pavimento.

5.2.8 Tabela de Condição dos Pavimentos

A tabela de condição dos pavimentos armazena o conceito de degradação do pavimento de acordo com o padrão definido pela norma DNIT 006/2003 PRO (2003) descrito na seção 2.4.6

6. Conclusões e Trabalhos Futuros

Este trabalho teve como objetivo projetar uma arquitetura de software utilizando conceitos da abordagem DDD, tais conceitos são muito importantes no processo de desenvolvimento de software, tendo em vista que, ao se aplicar DDD é possível criar soluções de forma mais eficiente, clara, escalável e que estão alinhadas com o domínio do negócio. Além disso, também foi desenvolvido um estudo de caso no domínio de gerenciamento de pavimentos com o intuito de por em prática os conceitos de DDD e arquitetura de microsserviços estudados. Conclui-se que a abordagem DDD oferece uma abordagem estruturada para analisar e modelar domínios de sistemas, através de sua aplicação é possível criar um modelo rico e expressivo, refletindo de forma fiel a realidade do domínio analisado.

Outro objetivo foi utilizar microsserviços para criar um sistema autônomo, coeso, resiliente, observável e centrado nos domínios da aplicação que foram definidos aplicando os conceitos de DDD. Observando os diagramas de modelo apresentados na seção anterior podemos constatar que a arquitetura de microsserviços permite a construção de um sistema modular e flexível dividindo o sistema em microsserviços que funcionam de forma independente. Além disso, os microsserviços também facilitam a adoção de novas tecnologias, permitindo assim, uma evolução gradual do sistema.

Ao decorrer do desenvolvimento do trabalho também foram encontrados desafios relacionados à junção de DDD e arquitetura de microsserviços, entre eles, a falta de informações fornecidas pelo modelo de domínio, ele não especifica quais são os microsserviços necessários para o sistema nem quais tecnologias vão ser usadas na sua implementação, dificultando assim, a identificação dos limites e responsabilidades dos microsserviços. Além disso, também é importante ressaltar os desafios individuais de cada abordagem que foram discutidos nas seções 2.2.3 e 2.3.6.

O estudo de caso desenvolvido durante o trabalho teve como objetivo criar uma arquitetura de software no domínio da gerência de pavimentos, para isso, foram usados conceitos de DDD e arquitetura de microsserviços adquiridos ao longo do trabalho para realizar a análise e modelagem do sistema. Porém, é importante ressaltar que a arquitetura apresentada não foi validada, logo, faz-se necessária uma validação rigorosa feita em conjunto com os

desenvolvedores e especialistas do domínio para validar sua viabilidade, desempenho e se o sistema se adequa aos requisitos específicos do domínio.

Finalmente, para os próximos trabalhos sugere-se que seja feita a implementação da arquitetura descrita neste trabalho, além disso, também sugere-se que seja feita a arquitetura do frontend e posteriormente sua implementação. Com esta etapa concluída, será possível começar a utilizar soluções em nuvem para hospedagem, armazenamento e disponibilização do sistema para o público.

Referências

ALMEIDA, Antonia; GONÇALVES, Haikel; SILVA, Ataslina; OLIVEIRA, Francisco. **PROPOSTA DE ADAPTAÇÃO DO ÍNDICE DE GRAVIDADE GLOBAL PARA AVALIAÇÃO DE TRATAMENTOS SUPERFICIAIS POR PENETRAÇÃO**. 33 Congresso de Pesquisa de Ensino em Transporte da ANPET, Balneário Camboriú - SC, p. 1329-1339, 10 nov. 2019. Disponível em: https://www.anpet.org.br/anais/documentos/2019/Infraestrutura/Dimensionamento,%20Avalia%C3%A7%C3%A3o%20e%20Gest%C3%A3o%20de%20Pavimentos%20IV/2_187_AC.pdf. Acesso em: 20 maio 2023.

ALVARENGA, Henrique. **Brazilian Freight Transportation Matrix demands investments**. ILOS, 21 ago. 2020. Disponível em: http://eng.ilos.com.br/web/brazilian-freight-transportation-matrix-demands-investments/. Acesso em: 22 jun. 2023.

BARBOSA, Eduardo. **Desenvolvimento dirigido a domínio: um estudo de caso**. Orientador: Prof. Dr. Fabiano Azevedo Dorça. 2021. 45 p. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Federal de Uberlândia, Uberlândia/MG, 2021. Disponível em: https://repositorio.ufu.br/bitstream/123456789/32180/1/DesenvolvimentoDirigidoDom%c3%adnio.pdf. Acesso em: 11 mar. 2023.

Benitti, F.B.V.; Rhoden, J.S. **Uma taxonomia unificada para requisitos não funcionais**. Revista Eletrônica de Sistemas de Informação, v. 14, n. 3, set-dez 2015. doi:10.21529/RESI.2015.1403004 Disponível em: http://www.periodicosibepes.org.br/index.php/reinfo/article/view/1678/pdf. Acesso em: 18 abr. 2023.

BETAINFRAESTRUTURA. **Afinal, o que é HDM-4?**. [*S. l.*], 16 fev. 2019. Disponível em: https://www.betainfraestrutura.com.br/2019/02/16/hdm/. Acesso em: 13 abr. 2023.

Bosurgi, G, Bruneo, D, De Vita, F, Pellegrino, O, Sollazzo, G. **A web platform for the management of road survey and maintenance information: A preliminary step towards smart road management systems**. *Struct Control Health Monit*. 2022; 29(3):e2905. doi:10.1002/stc.2905 Disponível em: https://onlinelibrary.wiley.com/doi/epdf/10.1002/stc.2905 Acesso em: 15 abr. 2023.

Brown, Simon. **The C4 model for visualising software architecture: Context, Containers, Components, and Code**. [*S. l.*], 2019. Disponível em: https://c4model.com/. Acesso em: 22 abr. 2023.

CARPO LOGISTICS. **O transporte rodoviário de carga no Brasil e suas características**. Carpolog.com.br, 19 set. 2019. Disponível em: http://www.carpolog.com.br/blog/o-transporte-rodoviario-de-carga-no-brasil-e-suas-caracteristicas/#:~:text=No%20Brasil%2C%20o%20transporte%20rodovi%C3%A1rio,manuten%C3%A7%C3%A3o%20e%20troca%20de%20pe%C3%A7as. Acesso em: 22 jun. 2023.

COLOMBO, Rafael. **Qual o tamanho da malha viária no Brasil e no mundo?!**. proauto.org.br, 31 maio 2022. Disponível em: https://www.proauto.org.br/blog/qual-o-tamanho-da-malha-viaria-no-brasil-e-no-mundo/. Acesso em: 22 jun. 2023.

DE VITA, Fabrizio; SOLLAZZO, Giuseppe; BRUNEO, Dario; PELLEGRINO, Orazio; BOSURGI, Gaetano. A Cloud Platform for Collecting and Processing Road Pavement Multi Sensor Data. 2021 IEEE International Conference on Smart Computing (SMARTCOMP), IEEE, 8 ago. 2021. DOI 10.1109/SMARTCOMP52413.2021.00072. Disponível em: https://ieeexplore.ieee.org/abstract/document/9556296. Acesso em: 17 abr. 2023.

Departamento Nacional de Infraestrutura de Transportes (DNIT). Diretoria Executiva. Instituto de Pesquisas Rodoviárias. **Manual de Gerência de Pavimentos**. Rio de Janeiro, 2011.

DIEPENBROCK, Andreas; RADEMACHER, Florian; SACHWEH, Sabine. **An Ontology-based Approach for Domain-driven Design of Microservice Architectures**. INFORMATIK, openstax.org, 2017. DOI 10.18420/in2017_177. Disponível em: https://dl.gi.de/handle/20.500.12116/3944. Acesso em: 20 fev. 2023.

DNIT 006/2003 – PRO, 2003. **Avaliação objetiva da superfície de pavimentos flexíveis e semi-rígidos –Procedimento**. Departamento Nacional de Infraestrutura de Transportes, Instituto de Pesquisas Rodoviárias - Rio de Janeiro /RJ;

Evans, Eric. **Domain-Driven Design: Atacando as Complexidades no Coração do Software**, 3ª edição, Alta Books, 16 de dezembro de 2016.

FOWLER, Martin. **Microservices: a definition of this new architectural term**. Martinfowler.com, 25 mar. 2014. Disponível em: https://martinfowler.com/articles/microservices.html#footnote-beck-rate-of-change. Acesso em: 7 fev. 2023.

Germoglio, Guilherme. **Arquitetura de Software**, Houston, Creative Commons, 5 de janeiro de 2010.

GOVERNO FEDERAL. **Rodovias Federais - Informações Gerais - Sistema Federal de Viação**. [*S. l.*], 2 set. 2020. Disponível em: https://www.gov.br/infraestrutura/pt-br/assuntos/transporte-terrestre_antigo/rodovias-federais/rodovias-federais-informacoes-gerais-sistema-federal-de-viacao#:~:text=A%20extens%C3%A3o%20total%20da%20malha,correspondem%20a%20rodovias%20n%C3%A3o%20pavimentadas. Acesso em: 17 maio 2023.

HIPPCHEN, Benjamin; GIESSLER, Pascal; HEINZ STEINEGGER, Roland; SCHNEIDER, Michael; ABECK, Sebastian. **Designing Microservice-Based Applications by Using a Domain-Driven Design Approach**. International Journal on Advances in Software, Www.iariajournals.org/software, ano 2017, v. 10, n. 3 & 4, p. 432-445.

HUDSON, W. Ronald; HAAS, Ralph; FALLS, Lynne. **Pavement Asset Management**. 1. ed. [S. l.]: Wiley-Scrivener, 2015. 432 p. v. 1. ISBN 9781119038702.

KHONONOV, Vlad. **Learning Domain-Driven Design: Aligning Software Architecture and Business Strategy**. 1. ed. [*S. l.*]: O'Reilly Media, 2021. 337 p. v. 1. ISBN 1098100131.

KHONONOV, Vladik. **What Is Domain-Driven Design?**. 1. ed. [*S. l.*]: O'Reilly Media, 2019. ISBN 9781492057796.

KHAHRO, Shabir; MEMON, Zubair; GUNGAT, Lillian; YAZID, Muhamad; RAHIM, Abdur; MUBARAKI, Muhammad; YUSOFF, Nur. **Low-Cost Pavement Management System for Developing Countries**. Sustainability, Https://www.mdpi.com/journal/sustainability, v. 13, 25 maio 2021. DOI https://doi.org/10.3390/su13115941. Disponível em: https://www.mdpi.com/2071-1050/13/11/5941. Acesso em: 15 abr. 2023.

MONTOYA-ALCARAZ, Marco; MUNGARAY-MOCTEZUMA, Alejandro; GARCÍA, Leonel. **Sustainable Road Maintenance Planning in Developing Countries Based on Pavement Management Systems: Case Study in Baja California, México**. Sustainability, Mexicali, 2019. Disponível em: https://www.mdpi.com/2071-1050/12/1/36. Acesso em: 10 abr. 2023.

RIZKI, M; N FAJAR, A; RETNOWARDHANI, A. **Designing Online Healthcare Using DDD in Microservices Architecture**. 5 th International Conference on Computing and Applied Informatics (ICCAI 2020), [*s. l.*], 2021. DOI 10.1088/1742-6596/1898/1/012010. Disponível

em: https://iopscience.iop.org/article/10.1088/1742-6596/1898/1/012010. Acesso em: 27 mar. 2023.

ROCHA, Jorcelan; NUNES, Larysse. **IMPLANTAÇÃO DE SISTEMA DE GERÊNCIA DE PAVIMENTOS: TRECHO DA 3**. Enciclopédia Biosfera, [*s. l.*], v. 16, n. 29, ed. 1, 30 jun. 2019. DOI 10.18677/EnciBio_2019A180. Disponível em: https://www.conhecer.org.br/enciclop/2019a/eng/implantacao.pdf. Acesso em: 11 abr. 2023.

SCHMIDT, Roger. **MODEL-DRIVEN ENGINEERING E DOMAIN-DRIVEN DESIGN PARA SUPORTE AO DESIGN DE MICROSSERVIÇOS**. Orientador: Dr. Marcello Thiry. 2018. 260 f. Dissertação de Mestrado (Mestrado em Computação Aplicada) - Universidade do Vale do Itajaí, São José (SC), 2018. Disponível em: https://www.univali.br/Lists/TrabalhosMestrado/Attachments/2441/Roger%20A.%20Schmidt%20-%20Disserta%C3%A7%C3%A3o%20MCA%20Univali.pdf. Acesso em: 24 mar. 2023.

SILVA, Camila. **DESENVOLVIMENTO DE UM SISTEMA DE GERÊNCIA DE PAVIMENTOS APLICADO AO CAMPUS I DA UNIVERSIDADE FEDERAL DA PARAÍBA**. Orientador: Prof. Dr. Ricardo Almeida de Melo. 2018. Trabalho de Conclusão de Curso (Superior) - Universidade Federal da Paraíba, [*S. l.*], 2018.

SHRP. **Distress Identification Manual for the Long-Term Pavement Performance Studies**. The Strategic Highway Research Program. National Academy of Science. Washington, D. C., 2014.

SÖYLEMEZ, Mehmet; TEKINERDOGAN, Bedir; KOLUKISA TARHAN, Ayça. **Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review. Applied Sciences**, mdpi.com, ano 2022, v. 11, p. 1-40, 29 mar. 2022. Disponível em: mdpi.com/2076-3417/12/11/5507. Acesso em: 19 fev. 2023.

TULIO VALENTE, Marco. **Domain-Driven Design (DDD): Um Resumo**. Https://engsoftmoderna.info/, 1 jan 2022. Disponível em: https://engsoftmoderna.info/artigos/ddd.html. Acesso em: 14 fev. 2023.

VERNON, Vaughn. **Implementing Domain-Driven Design**. 1. ed. [*S. l.*]: Addison-Wesley, 2016. 672 p.

VIEIRA, Matheus. **Desenvolvimento de Aplicações Utilizando DDD e Cloud Computing**. Orientador: Ronaldo Castro de Oliveira. 2023. 56 p. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) - Universidade Federal de Uberlândia, Uberlândia/MG, 2023. Disponível em: https://repositorio.ufu.br/bitstream/123456789/37344/1/DesenvolvimentoAplica%c3%a7%c3%b5esUtilizando.pdf. Acesso em: 11 mar. 2023.

VURAL, Hulya; KOYUNCU, Murat. **Does Domain-Driven Design Lead to Finding the Optimal Modularity of a Microservice?**. IEEE, ieeexplore.ieee.org, v. 9, p. 32721-32733, 2 mar. 2021. DOI 10.1109/ACCESS.2021.3060895. Disponível em: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9359794. Acesso em: 27 mar. 2023.

WILLIANS, Wesley. **O que é DDD – Domain Driven Design**. Fullcycle.com.br, 19 ago. 2019. Disponível em: https://fullcycle.com.br/domain-driven-design/. Acesso em: 14 fev. 2023.

WHITTAKER, Daniel. **Are You Making These 10 DDD Mistakes?**. Https://danielwhittaker.me/, 2015. Disponível em: https://danielwhittaker.me/2020/01/22/are-you-making-these-10-ddd-mistakes/. Acesso em: 5 mar. 2023.

Apêndice A - Modelo C4

Os diagramas apresentados na seção 5 deste trabalho estão disponíveis no figma e podem ser acessados via os seguintes links:

- Diagrama de Contexto
- Diagrama de Container
- Diagramas de Componentes