UNIVERSIDADE FEDERAL DE SANTA CATARINA

IMPLEMENTAÇÃO DE MICROSSERVIÇOS PARA UM SISTEMA DE GESTÃO DE APRENDIZAGEM

Mariany Ferreira da Silva

Florianópolis - SC

2024

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE BACHAREL EM SISTEMAS DA INFORMAÇÃO

IMPLEMENTAÇÃO DE MICROSSERVIÇOS PARA UM SISTEMA DE GESTÃO DE APRENDIZAGEM

Mariany Ferreira da Silva

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas da Informação

SUMÁRIO

LISTA DE FIGURAS

LISTA DE REDUÇÕES

ACID *Atomicidade, Consistência, Isolamento e Durabilidade*

API *Application Programming Interface*

DDD *Domain Driven Design*

LMS *Learning Management System*

JSON *JavaScript Object Notation*

MVP *Minimum Viable Product*

SGA *Sistema de Gestão de Aprendizagem*

RESUMO

Este Trabalho de Conclusão de Curso tem como objetivo desenvolver o Produto Mínimo Viável de um Sistema de gestão de Aprendizagem (SGA) usando uma abordagem dirigida por domínio (*Domain-Driven Design* - DDD), e arquitetura de microsserviços garantindo consistência de dados e comunicação síncrona e assíncrona entre eles. Para alcançar este objetivo, foi utilizada a metodologia de abordagem exploratória, com uma pesquisa bibliográfica para construir um referencial teórico sólido e a aplicação dos conhecimentos adquiridos no desenvolvimento do sistema.

Os resultados desta pesquisa indicam que uma abordagem dirigida por domínio ajuda a diminuir a complexidade de implementação de um SGA pois cada contexto delimitado facilita a separação de responsabilidades. Já a arquitetura de microsserviços oferece vantagens significativas em termos de modularidade, escalabilidade e independência de implantação.

Este estudo contribui para a área de sistemas de informação ao fornecer novas insights, aplicando de forma conjunta *DDD* e arquitetura de microsserviços na busca por responder a uma demanda contemporânea.

Palavras-chave: *domain-driven design*, microsserviços, sistema de gestão de aprendizagem

1. INTRODUÇÃO

A transformação digital acelerada e os avanços tecnológicos recentes impulsionaram uma mudança significativa no setor educacional. Instituições de ensino de todo o mundo têm buscado adaptar-se a este novo cenário, sistematizando e migrando cursos presenciais para modalidades a distância ou híbridas. Neste contexto, os Sistemas de Gestão de Aprendizagem (SGA) emergem como ferramentas fundamentais para facilitar e enriquecer o processo educacional, oferecendo suporte a atividades educacionais apoiadas pelas tecnologias de informação e comunicação.

Segundo um relatório da Relatórios Insights (2023), o mercado de LMS deve atingir mais de US$69 bilhões até 2030, com uma taxa de crescimento anual composta (CAGR) de 19,2% entre 2023 e 2030. Já o o Global Education Census Report (CAMBRIDGE ASSESSMENT INTERNATIONAL EDUCATION, 2018) apontou que a tecnologia nas salas de aula está cada vez mais presente, com 48% dos alunos usando computadores de mesa, 42% usando smartphones, 33% utilizando quadros interativos e 20% usando tablets.

A maioria dos SGAs implementa funcionalidades que incluem: criação, organização e gerenciamento de cursos e treinamentos online; hospedagem e distribuição de materiais em diversos formatos, como vídeos, textos, quizzes, podcasts, etc; ferramentas para discussões interativas, fóruns, questionários e tarefas colaborativas; recursos para avaliação do desempenho dos alunos e emissão de certificados; controle de acesso, gerenciamento de inscrições e perfis de usuários; monitoramento do progresso dos alunos, geração de relatórios de desempenho e análise de dados; entre outros (Moodle, 2024). No mercado atual, temos diversos SGAs disponíveis como:

Moodle: um dos mais populares e amplamente utilizados, especialmente em instituições educacionais, possui código aberto e sua interface é personalizável, mas requer conhecimento técnico sólido para personalização avançada. Foi desenvolvido em PHP, HTML, CSS e JavaScript. É estruturado como um sistema modular, com um núcleo de aplicação (core) que implementa diversas as funcionalidades básicas, como interface de usuário, gestão de cursos, autenticação, segurança, avaliação e feedback e é cercado por inúmeros plugins que fornecem funcionalidades específicas como módulos de atividades, integração com outros sistemas, conversão de arquivos etc.

Canvas: muito utilizado em universidades e escolas, conhecido por sua interface amigável. Apesar de oferecer uma versão gratuita, o acesso a recursos avançados é limitado e pode ter um custo significativo. Além disso, usuários citam falta de opções de personalização. De acordo com o documento "Canvas Architecture" (INSTRUCTURE, 2022), possui código fechado e foi desenvolvido utilizando o framework Ruby on Rails, HTML, CSS e JavaScript. É um sistema distribuído e disponibilizado utilizando o formato de software como serviço (SaaS), onde o fornecedor do software se responsabiliza por toda a estrutura necessária à disponibilização do sistema, e o cliente utiliza o software via internet, pagando um valor pelo serviço.

Blackboard Learn: oferece recursos que atendem a uma ampla gama de necessidades educacionais, mas o preço e a interface são menos intuitiva do que outras plataformas LMS, possui código fechado e foi desenvolvido utilizando Java, HTML, CSS e JavaScript. Assim como o Canvas é um sistema distribuído e disponibilizado utilizando o formato de software como serviço (SaaS) (BLACKBOARD, 2024).

A complexidade e a dinâmica do ambiente educacional nas universidades, escolas de ensino fundamental, médio ou cursos avulsos demandam soluções tecnológicas que sejam não apenas eficientes e escaláveis, mas também flexíveis e adaptáveis às constantes mudanças nas necessidades de instituições, educadores e alunos. Além disso, a implementação de tantas funcionalidades torna o desenvolvimento de um SGA completo complexo, trabalhoso e demorado.

Os SGAs descritos anteriormente utilizam abordagens de desenvolvimento que dificultam a manutenibilidade e expansão do sistema. O uso de abordagens mais modernas de desenvolvimento de SGAs pode reduzir esse problema. A abordagem Dirigida por Domínio (*Domain-Driven Design* - DDD) é uma opção interessante que pode ser empregada neste tipo de sistema. Esta abordagem também facilita o emprego de um modelo de comunicação adequado entre os microsserviços e consistência de dados entre eles, necessário para o funcionamento correto do SGA.

Neste TCC será proposto e desenvolvido um Produto Mínimo Viável (ou *Minimum Viable Product*, da sigla MVP) denominado Sumé, centrado nas funcionalidades básicas e essenciais de um SGA, que envolvem a criação e gestão de cursos, turmas e atividades, permitindo a integração de funcionalidades adicionais em etapas posteriores. Estas questões fundamentam a seguinte questão de pesquisa, a qual buscamos responder neste TCC:

É possível desenvolver o Produto Mínimo Viável de um Sistema de gestão de Aprendizagem usando uma abordagem dirigida por domínios, e arquitetura de microsserviços garantindo consistência de dados e comunicação síncrona e assíncrona entre eles?

Para responder esta pergunta, o projeto propõe definir requisitos e características-chave para criação e gestão de cursos, turmas e atividades, que envolvem: o design dos domínios *Courses*, *Classrooms* e *Activities*, utilizando os princípios e práticas do *Domain-Driven Design* (DDD), enfatizando a importância da modelagem do domínio de negócio e da colaboração entre desenvolvedores e especialistas do domínio. Com este objetivo delimitado, definimos os seguintes objetivos específicos:

1. Implementação desses domínios em microsserviços independentes e desacoplados;
2. Estruturação e modelagem das bases de dados de cada microsserviço, para manter a consistência e a acessibilidade das informações em um ambiente distribuído, reduzindo o acoplamento e aumentando a resiliência do sistema.
3. Implementação de comunicação síncrona e assíncrona entre os domínios, utilizando os *Domain Events* previamente elaborados no design do sistema, para facilitar a compreensão dos fluxos de trabalho e das interações entre os microsserviços, permitindo a definição de padrões de comunicação eficientes e flexíveis.

Ao abordar esses objetivos, este TCC busca contribuir para o avanço das práticas de desenvolvimento de software no campo da tecnologia educacional, oferecendo uma solução inovadora que combina eficiência, flexibilidade e adaptabilidade a diversos contextos educacionais.

1.1 JUSTIFICATIVA

O desenvolvimento de um MVP do Sumé está alinhado com as tendências contemporâneas em educação e tecnologia, visando suprir as demandas

emergentes de instituições educacionais, docentes e discentes. A justificativa para este projeto reside em sua capacidade potencial de enriquecer o desenvolvimento de sistemas de gestão da aprendizagem, oferecendo uma plataforma mais intuitiva, acessível e adaptável através de uma arquitetura modular e flexível projetada e implementada utilizando metodologias e ferramentas modernas que possibilitem manutenção eficiente e crescimento qualitativo.

Muitos dos SGAs existentes no mercado foram criados utilizando uma arquitetura monolítica, onde a maior parte dos componentes do sistema são fortemente acoplados, dificultando a escalabilidade e a flexibilidade do sistema. Isso dificulta que universidades, escolas de ensino fundamental, médio ou cursos avulsos utilizem o mesmo SGA e escolham quais funcionalidades são relevantes para cada caso de uso, uma vez que elas estão integradas em um único bloco (ZABIEROWSKI, 2020).

Outros SGAs foram implementados utilizando arquitetura modular, que divide a aplicação em módulos, cada um responsável por um conjunto de funcionalidades. Embora os módulos sejam projetados para serem independentes, a integração entre eles pode se tornar complexa, especialmente em sistemas grandes e com muitos módulos, além disso a comunicação entre módulos pode introduzir alguma sobrecarga de desempenho, especialmente se as interfaces de comunicação não foram projetadas corretamente (DESH DEVS, 2023).

Por não serem de código aberto, alguns SGAs não permitem ou facilitam a mudança de regras de negócio para que o sistema se adapte às necessidades específicas de cada instituição.

É nesse cenário que o Sumé se destaca, trazendo uma abordagem inovadora e diferenciada. O Sumé é projetado utilizando os princípios do

Domain-Driven Design (DDD), uma metodologia que coloca o foco no domínio do problema a ser resolvido, buscando obter um entendimento profundo e preciso das regras, processos e terminologia específicos desse domínio, garantindo que o sistema reflita fielmente as necessidades reais dos usuários (alunos, educadores e instituições de ensino). Outra característica é a adoção de uma arquitetura de microsserviços. Essa abordagem arquitetural divide o sistema em pequenos serviços independentes e autônomos, cada um responsável por uma funcionalidade específica. Os microsserviços podem ser desenvolvidos, implantados e atualizados de forma independente, facilitando a adição de novas funcionalidades e a evolução contínua do sistema. Além disso, é um sistema de código aberto, permitindo que as instituições adaptem funcionalidades e regras de negócio com facilidade.

A implementação do MVP possibilita uma avaliação concreta e eficaz da viabilidade e funcionalidade do sistema, assegurando que as funcionalidades e capacidades essenciais sejam testadas e refinadas com base no feedback direto dos usuários. Outro aspecto é o foco nos desafios técnicos e práticos da implementação. A análise desses desafios fornecerá observações para a comunidade acadêmica e para os profissionais da área, contribuindo para o desenvolvimento de melhores práticas e estratégias eficientes no campo da tecnologia educacional.

1.2 METODOLOGIA

Para alcançar os objetivos propostos, será adotada uma abordagem exploratória, iniciando com uma pesquisa bibliográfica aprofundada para construir um referencial teórico sólido. Isso incluirá a revisão de literatura acadêmica sobre arquitetura de microsserviços, práticas de gestão de dados em ambientes distribuídos e tecnologias emergentes no desenvolvimento de SGAs.

Tecnologias pertinentes serão selecionadas para realizar a implementação dos microsserviços do SGA servindo como um estudo de caso prático. Esta implementação facilitará a análise dos desafios, eficácia e viabilidade da arquitetura de microsserviços e das estratégias de modelagem de dados.

Durante o desenvolvimento do protótipo, todas as etapas, desafios e soluções serão documentadas de forma detalhada. As informações documentadas serão analisadas qualitativamente para identificar padrões, desafios e estratégias adotadas. Também traremos as limitações enfrentadas durante a pesquisa, como restrições de escopo do protótipo ou limitações de acesso a recursos.

1.3 ESTRUTURA DO TRABALHO

O trabalho será organizado da seguinte maneira: no capítulo 2, os referenciais teóricos estudados para realizar o desenvolvimento do projeto são apresentados; no capítulo 3 é apresentado o desenvolvimento do SGA Sumé, considerando desafios na implementação utilizando microsserviços e na estruturação e modelagem da base de dados; o capítulo 4 apresenta as considerações finais sobre o desenvolvimento, conceitos e tecnologias estudadas, dificuldades encontradas, e trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 SISTEMA DE GESTÃO DE APRENDIZAGEM

Sistema de Gestão de Aprendizagem, conhecidos em inglês como *Learning Management Systems* (LMS), são plataformas amplamente utilizadas para facilitar a educação a distância. Conforme descrito por Almeida (2003), esses sistemas têm como finalidade principal oferecer suporte a atividades educacionais apoiadas pelas tecnologias de informação e comunicação.

Além de seu papel na educação a distância, esses ambientes virtuais também desempenham um papel complementar no ensino presencial e híbrido. Eles facilitam a interação entre alunos e educadores além da turma, permitindo, por exemplo, a criação de cursos, planejamento de aulas, realização de tarefas e a organização eficiente do material didático que pode ser acessado em qualquer lugar por meio de uma variedade de dispositivos de acesso, incluindo computadores, celulares, tablets, entre outros (DE MORAES, 2023).

2.2 DESIGN DE SOFTWARE

O *Domain-Driven Design* (DDD) é uma abordagem de design de software que oferece um conjunto de princípios e práticas destinados a auxiliar os desenvolvedores a compreender melhor o domínio de negócio específico ao qual o software se destina, os problemas, necessidades, funcionalidades e regras, enfatizando a importância da cooperação proativa e contínua entre desenvolvedores e especialistas do domínio para modelá-lo de forma eficaz e implementá-lo com eficiência, visando construir software que não apenas atenda às necessidades atuais do negócio mas também seja capaz de evoluir junto com ele (EVANS, 2003). Esta abordagem é ideal para a implementação deste trabalho por várias razões:

Complexidade do Domínio: O DDD é especialmente útil em contextos onde o domínio do negócio é complexo. Os microsserviços *Courses*, *Classroom* e *Activity*, possuem suas próprias regras de negócio e lógicas que precisam ser claramente entendidas e implementadas.

Modelagem e Estrutura de Dados: O DDD promove a criação de um modelo de domínio rico, que ajuda a definir claramente as entidades, valor objetos, agregados, e repositórios, facilitando a estruturação do código e das bases de dados de cada microsserviço de forma consistente e alinhada com as regras de negócio.

Comunicação: A implementação de comunicação síncrona e assíncrona é crucial para a eficiência e escalabilidade do sistema. O DDD, por meio de eventos de domínio, fornece uma base sólida para o design de comunicação eficaz.

Para aplicar DDD nesse projeto é necessário fundamentar os conceitos, que nos ajuda a compreender como organizar e modelar o domínio, garantindo que o desenvolvimento do software esteja alinhado com as complexidades e nuances do domínio de negócio.

Linguagem Ubíqua: Propõe o uso de uma linguagem comum entre desenvolvedores e especialistas do domínio. A ideia é que todos os envolvidos no projeto compartilhem o mesmo vocabulário e entendam os termos da mesma forma, o que facilita a comunicação e reduz mal-entendidos (VERNON, 2013).

Contextos Delimitados: Divisão do domínio em múltiplos contextos delimitados, cada um representando uma fronteira lógica dentro da qual um modelo de domínio específico é aplicado. Isso ajuda a manter a coerência do modelo e evita conflitos e ambiguidades entre diferentes partes do sistema. (EVANS, 2003).

Modelo de Domínio: Representação estruturada das entidades, seus relacionamentos e regras de negócio, que reflete as necessidades e complexidades do domínio em questão (EVANS, 2003).

Entidades e Objetos de Valor: As entidades são objetos que têm uma identidade contínua ao longo do tempo e são claramente identificáveis. Objetos de valor, por outro lado, são definidos apenas por seus atributos e não possuem uma identidade própria (VERNON, 2013).

Agregados: Cluster de objetos de domínio (entidades e objetos de valor) que podem ser tratados como uma unidade para fins de processamento de dados. Cada agregado tem uma raiz e uma fronteira clara, dentro da qual todas as regras de consistência são aplicadas (EVANS, 2003).

Repositórios: Usados para encapsular a lógica necessária para acessar dados de armazenamento, permitindo que o restante da aplicação se concentre na lógica de negócio. Eles agem como uma interface para a obtenção de agregados do domínio (VERNON, 2013).

Mapas de Contexto: Mostra como diferentes modelos de domínio interagem e se integram, e como as linguagens ubíquas são compartilhadas ou traduzidas entre esses contextos. Ele é usado para planejar e entender as dependências e interações entre diferentes partes do sistema (EVANS, 2003). Existem vários padrões que podem ser aplicados para descrever as relações entre contextos, como Parceria, Núcleo Compartilhado, Conformista, entre outros. Nesse projeto temos relações de:

- **Consumidor-Fornecedor (Customer-Supplier):** Descreve a interação entre dois contextos, onde um contexto upstream fornece serviços ou dados para um contexto downstream, que depende

desses serviços ou dados para funcionar. Nessa relação, o Fornecedor tem mais influência porque é ele quem decide o formato e o tempo de resposta (VERNON, 2013).

- **Camada Anticorrupção (Anti-Corruption Layer):** É uma forma de proteger a comunicação entre diferentes contextos, evitando que problemas ou mudanças em uma parte afetem negativamente a outra. Um contexto **downstream** usa uma camada anticorrupção para traduzir ou adaptar a comunicação com um contexto **upstream**, protegendo seu próprio modelo de influências indesejadas. Embora seja útil para manter a organização e a clareza, montar essa camada pode ser caro e complexo (VERNON, 2013).

Eventos de Domínio: São eventos significativos no domínio que são importantes para o negócio, usados para desencadear efeitos colaterais em diferentes partes do sistema, facilitando a comunicação entre contextos delimitados. Se operações realizadas pelo sistema são causalmente relacionadas, então uma operação pode causar outra e elas devem ocorrer em uma ordem específica. Para exemplificar, pode ser que um Agregado não possa ser criado ou modificado até que uma operação específica ocorra em outro Agregado. Esse tipo de arquitetura de sistema linearizado e causal pode ser implementado através da criação e publicação de Eventos de Domínio (VERNON, 2013).

2.3 MICROSSERVIÇOS

De acordo com Taibi (2017), microsserviços são pequenas aplicações que oferecem a vantagem de serem implantadas, escaladas e testadas de forma independente. Adicionando a essa perspectiva, Shadija e Rezai (2017) sugerem que cada microsserviço deve ser projetado para atender a uma única funcionalidade de

negócio. Dessa forma, a combinação desses microsserviços realiza funções de negócios mais amplas e integradas.

Newman (2020), aponta que essa abordagem permite a combinação de diversas tecnologias, linguagens de programação, estilos de codificação, plataformas de implantação e sistemas de banco de dados. Essa variedade de opções viabiliza a criação de uma solução altamente eficaz e adaptada aos problemas e oportunidades dos domínios em que estão inseridas, visando otimizar os resultados para o usuário final do sistema.

Consequentemente, muitas empresas têm favorecido a arquitetura de microsserviços, atraídas pelos seus numerosos benefícios. Estes incluem uma maior facilidade de testes, a possibilidade de realizar implantações rápidas, independentes e simplificadas, além de uma melhor escalabilidade, modularidade e agilidade. Adotando esta abordagem, pequenas equipes conseguem operar de forma dinâmica e autônoma, mesmo dentro de grandes organizações (RICHARDSON C., 2019). Embora a adoção de microsserviços ofereça vantagens operacionais, essa escolha também acarreta várias complexidades relacionadas à implementação de sistemas distribuídos e à modelagem de domínio.

2.4 DADOS DISTRIBUÍDOS

Várias fontes na literatura em publicações especializadas em microsserviços defendem o uso de bancos de dados separados para cada microsserviço. Richardson (2019) explora padrões de design de microsserviços e discute a importância de cada microsserviço ter seu próprio banco de dados para evitar dependências e garantir a resiliência do sistema. “Quando falamos sobre a construção de bancos de dados para microsserviços é importante reduzir ou eliminar o acoplamento no banco de dados - mas isso realmente significa

acoplamento no nível do esquema do banco de dados” (IBM, 2020). Também é importante lembrar que os microsserviços têm diferentes requisitos de armazenamento, em alguns é mais vantajoso usar uma base de dados relacional, noutros é mais simples usar uma base de dados não relacional, e outros até podem necessitar de bases de dados *key-value* ou orientadas a grafos.

Uma das preocupações na arquitetura de microsserviços é manter os dados separados para evitar problemas de desempenho e escalabilidade, ao mesmo tempo em que se evita a redundância e se mantém a consistência dos dados (RICHARDSON, 2019). O objetivo é garantir a autonomia dos bancos de dados de cada microsserviço, assegurando também a integração e a consistência entre eles.

Manter dados distribuídos requer um esforço adicional de gerenciamento, além disso garantir a consistência dos dados entre diferentes microsserviços pode ser desafiador, especialmente em operações que envolvem múltiplos contextos, pois a comunicação entre microsserviços pode introduzir latência e aumentar a complexidade do sistema.

2.4.1 ACID

As características ACID são um conjunto de propriedades fundamentais de transações em sistemas de banco de dados. ACID é um acrônimo que representa A - atomicidade, onde cada transação deve ser concluída em sua totalidade ou não deve ter efeito algum; C - consistência, que assegura que uma transação sempre transforma o banco de dados de um estado consistente para outro estado consistente, mantendo a integridade dos dados; I - isolamento, que é a garantia de que as transações sejam executadas de forma isolada umas das outras; e por fim D - durabilidade, uma vez que uma transação é concluída, as alterações feitas por ela no banco de dados são permanentes e persistem mesmo no caso de uma falha no

sistema. Estas propriedades garantem que as transações em bancos de dados sejam processadas de maneira confiável (SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S., 2010).

2.4.2 CONSISTÊNCIA EVENTUAL

Conforme Burckhardt (2014), a consistência eventual é um modelo de consistência que garante que, se nenhuma nova atualização for realizada em um determinado conjunto de dados, eventualmente todas as consultas à esses dados retornarão o último valor atualizado de forma consistente. Isso significa que, durante um período de tempo, consultas idênticas podem resultar em valores divergentes, mas devem convergir eventualmente. Nesses casos, os dados são persistidos de maneira assíncrona, sem que a aplicação necessite esperar a confirmação de sucesso em todos os microsserviços do sistema.

O problema da consistência dos dados distribuídos é abordado de diferentes maneiras, dependendo da necessidade da aplicação. Um dos principais padrões usados em microsserviços é o padrão SAGA. Segundo Richardson C. (2019), a SAGA é uma sequência de transações que são executadas em diferentes microsserviços. Se uma transação falhar, as transações anteriores são compensadas (revertidas) para manter a consistência eventual. Existem duas formas de implementar uma SAGA:

Sequência de Transações Locais: Uma SAGA é uma série de transações locais, cada uma atualizando dados dentro de um único banco de dados. Essas transações são executadas em uma ordem específica.

Coreografia: Cada microsserviço é responsável por publicar eventos e reagir a eventos publicados por outros microsserviços, coordenando a SAGA de forma descentralizada.

Orquestração: Um coordenador central é responsável por orquestrar a SAGA nos microsserviços e lidar com falhas e compensações.

Para Richardson C. (2019), devemos dar preferência à usar coreografia em SAGAs simples, já SAGAs complexas devem ser implementadas idealmente como máquinas de estado gerenciadas por um orquestrador. Ele ainda aponta que este padrão garante atomicidade, consistência e durabilidade (ACD), mas durante a execução de uma SAGA, leituras e escritas intermediárias podem ser visíveis para outras transações, violando a propriedade de isolamento. Esse comportamento pode resultar em dois problemas:

1. Outras SAGAs podem modificar os dados acessados pela SAGA enquanto ela ainda está em execução.
2. Outras SAGAs podem ler esses dados antes que a SAGA finalize suas atualizações, o que pode expô-las a dados inconsistentes.

Segundo Marigi G. e Vocale M., a SAGA deve implementar estratégias, conhecidas como contramedidas, para minimizar o impacto da falta de isolamento. Uma das estratégias mais adotadas é o uso de um estado, para cada operação, que sinaliza que a operação está em andamento, essa estratégia é conhecida como *semantic lock*. Se outra instância do SAGA iniciar, ela deverá verificar os estados e tomar uma ação determinada pela lógica da aplicação que pode envolver esperar até o conseguir tomar o *semantic lock*, tentar novamente ou cancelar a operação.

2.5 INTEGRAÇÃO E COMUNICAÇÃO ENTRE MICROSSERVIÇOS

Shadija e Rezai (2017) sugerem que uma característica fundamental que todos os microsserviços devem ter em comum é a adoção de um padrão de comunicação. Frequentemente, isso envolve o uso de protocolos como

Representational State Transfer (REST), *Remote Procedure Call* (RPC) e *Advanced Message Queuing Protocol* (AMQP). A necessidade desse padrão de comunicação se torna evidente, considerando que uma característica distintiva dos microsserviços é a independência de suas fontes de dados. Assim, sempre que um microsserviço precisa acessar dados de outro, essa interação deve ocorrer por meio do protocolo de comunicação estabelecido.

Mensagens síncronas são úteis quando a resposta imediata é crucial, mas podem causar acoplamento mais rígido entre os microsserviços, limitações em tarefas paralelas, risco de falhas em cascata e fragilidade em cadeias de dependência. Para reduzir o acoplamento entre os microsserviços ao utilizar comunicação síncrona, é possível utilizar um *API Gateway*. Este componente é responsável por mapear e realizar a comunicação síncrona, evitando o contato direto entre os microsserviços. A comunicação assíncrona, embora ofereça maior flexibilidade e menos acoplamento, requer um monitoramento cuidadoso, pois as interações entre os microsserviços são menos previsíveis (RICHARDSON C., 2019).

2.5.1 REST

O REST surgiu como uma tentativa de padronizar a arquitetura de aplicações Web. Baseado no protocolo HTTP (*Hypertext Transfer Protocol*), amplamente utilizado na internet e servindo como base para a comunicação entre clientes e servidores web, carrega o conceito *stateless,* onde cada solicitação HTTP contém todas as informações necessárias para ser processada pelo servidor, sem depender de qualquer contexto armazenado no servidor. O REST define um conjunto limitado de métodos (*GET, POST, PUT, DELETE*) onde todos os recursos são identificados por URIs (*Uniform Resource Identifiers*), e as representações desses recursos são

transferidas usando formatos como JSON (*JavaScript Object Notation*) (SAUDATE, 2013).

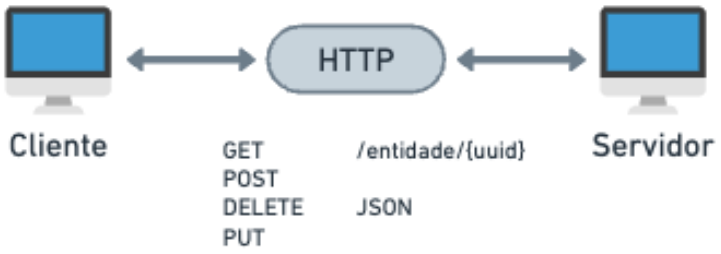

Figura 1. REST

Fonte: Produzido pela autora

2.5.2 RPC

De acordo com Karthik, Kumar e Smith (2023), o RPC define um conjunto de regras e convenções para permitir que uma aplicação web execute procedimentos ou funções em outra aplicação web, como se fossem uma única aplicação, abstraindo a complexidade da comunicação de rede e facilitando a interação entre sistemas distribuídos. Nesse protocolo, o desenvolvedor define as interfaces de microsserviço relevantes e os tipos de mensagem em um arquivo de esquema e um compilador irá traduzir o esquema em *stubs* de programa. Os stubs são responsáveis pelo processo de transmissão e tradução das mensagens, serializando e deserializando dados para os formatos entendidos pelo cliente e pelo servidor.

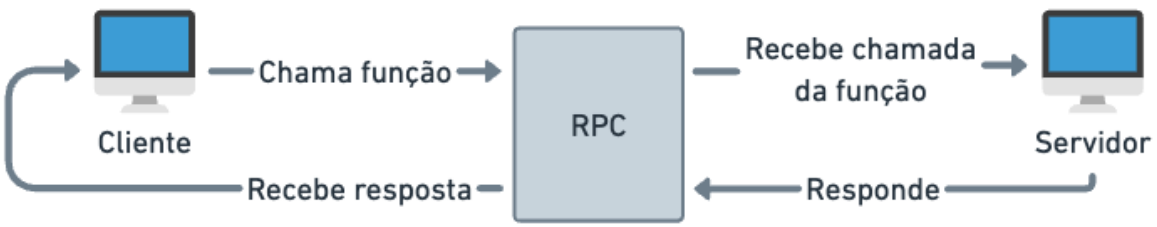

Figura 2. RPC

Fonte: Produzido pela autora

2.5.3 AMQP

O AMQP foi desenvolvido para facilitar a comunicação assíncrona entre sistemas distribuídos dando suporte à comunicação direta entre dois pontos ou por meio do modelo de publicação e subscrição, assegurando a entrega confiável de mensagens, incluindo confirmações de mensagens, transações e segurança na camada de transporte (TRIELOFF et al., 2006). Esse protocolo é estruturado em torno de vários componentes que facilitam a comunicação entre produtores (publishers) e consumidores (consumers) de mensagens.

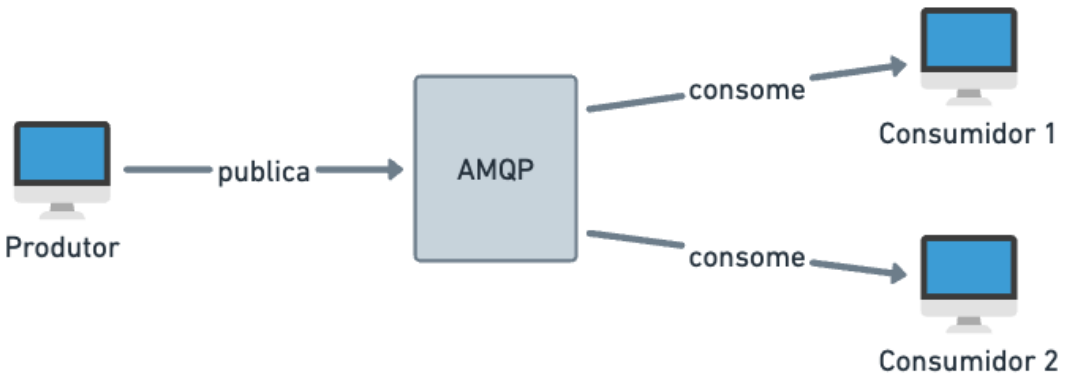

Figura 3. AMQP (pub/sub)

Fonte: Produzido pela autora

2.6 CRESCIMENTO QUALITATIVO CONTÍNUO

Para alcançar sucesso ao implementar sistemas modernos, é fundamental que eles sejam projetados para acomodar seu crescimento contínuo. Para isso, enfrentamos dois tipos distintos, porém interligados, de crescimento: qualitativo e quantitativo, ambos essenciais para a escalabilidade e evolução do sistema. O crescimento quantitativo é medido em termos numéricos, como um aumento no número de solicitações, usuários, transações ou no volume de dados processados

pelo sistema, no contexto técnico, isso geralmente envolve escalabilidade horizontal e vertical do sistema, dedicando atenção para as instâncias do microsserviço ou recursos de infraestrutura para distribuir a carga de trabalho. Por outro lado, o crescimento qualitativo diz respeito à manutenção, melhoria e ao enriquecimento das capacidades do sistema, abrangendo a introdução de novas funcionalidades, a otimização de processos existentes, ou o aumento da capacidade de lidar com tarefas mais complexas e avançadas (FOWLER, 2017).

Em uma arquitetura de microsserviços, cada um delespode ser escalado independentemente, permitindo que o sistema responda de maneira mais eficiente às variações de carga. Além disso, a falha em um microsserviço não necessariamente compromete todo o sistema, o que aumenta a resiliência e a disponibilidade do sistema (NEWMAN, 2015).

Essa arquitetura também torna o sistema mais flexível, permitindo que as equipes de desenvolvimento respondam rapidamente às mudanças nas necessidades e características do domínio. Além disso, podemos integrar diferentes sistemas, tecnologias e frameworks para diferentes microsserviços, otimizando o desempenho e facilitando a experimentação e a adoção de novas tecnologias (NEWMAN, 2020).

Aplicar *Domain-Driven Design* ajuda a criar uma arquitetura de microsserviços que é robusta, eficiente e capaz de se adaptar a mudanças no domínio. Ao definir *Bounded Contexts*, utilizar uma Linguagem Ubíqua, e modelar entidades e value objects de maneira precisa, DDD facilita a integração de novas responsabilidades e microsserviços, minimizando interrupções e garantindo a escalabilidade e a manutenção do sistema (JAISWAL e AGRAWAL, 2013).

3. PROPOSTA E DESENVOLVIMENTO DO MVP SUMÉ

Para obter uma compreensão e visualização detalhada dos variados microsserviços e potenciais oportunidades em um SGA, esta fase do projeto é dedicada descrever o papel, o objetivo principal e as responsabilidades específicas de cada microsserviço dentro do ecossistema do SGA. Além disso, esta etapa inclui uma análise da arquitetura, destacando suas vantagens e desvantagens. Essa análise busca oferecer uma visão sobre como essas decisões de arquitetura afetam o projeto.

3.1 REQUISITOS FUNCIONAIS

Os requisitos funcionais definem o que o sistema deve fazer. Eles descrevem os comportamentos ou funções específicas do sistema. Com base na observação dos SGAs existentes e também em conversas com desenvolvedores e usuários de SGAs, elencamos os principais requisitos funcionais do SGA Sumé. O sistema deve:

1. Identificar os papéis dos usuários (estudante, educador ou administrador).
2. Manter detalhes de um curso, como código, nome, descrição, objetivos e imagens associadas.
3. Manter detalhes de uma matéria, como código, nome, descrição, objetivo, carga horária, quantidade de créditos, data de publicação.
4. Manter detalhes de uma matriz curricular, como código, nome e descrição.
5. Permitir a inscrição de usuários em um cursos, indicando o papel do usuário nesse curso, se tem uma data de expiração, e se foi deletada por algum motivo específico.

6. Permitir o vínculo entre matérias e matrizes, indicando se uma matéria é obrigatória e se pertence a algum agrupamento (1º, 2º, 3º bimestre/semestre, etc).
7. Manter detalhes de turmas, como código, nome, descrição, data início e fim, se é possível auto inscrição, data em que a turma será aberta.
8. Manter detalhes do cronograma de uma turma, como local, dia da semana (recorrente), horário de início e de final.
9. Permitir a inscrição de usuários em turmas, se tem uma data de expiração.
10. Permitir o vínculo entre turmas e aulas, indicando as datas de início e fim para a conclusão.
11. Manter detalhes de aulas, como nome, tipo, módulo, objetivo e data de publicação.
12. Manter detalhes de atividades, como nome, descrição, recurso de aprendizado associado, tipo do recurso de aprendizado associado, objetivo atrelado à taxonomia de Bloom (criar, avaliar, aplicar, compreender, lembrar).
13. Manter *logs* das atividades, como status (pendente, atrasada, iniciada, finalizada, etc) e o conteúdo do log em um formato flexível (JSON) que considere o tipo do recurso de aprendizado associado e métricas, como tempo total para finalizar, dias de atraso (se houver), etapas concluídas (se houver), acertos (se houver), etc.
14. Permitir o vínculo entre aula e atividades.
15. Fornecer operações de criação, leitura, atualização e exclusão (CRUD) para todas as entidades.
16. Ter um Identificador Único Universal em todas as entidades e para os registros armazenados nas bases de dados.

17. Manter dados relevantes e que auxiliam no funcionamento do sistema, como nome e/ou código, descrição, data de criação, atualização, deleção, etc.

3.2 REQUISITOS NÃO-FUNCIONAIS

Os requisitos não funcionais descrevem as qualidades e restrições operacionais do sistema. Eles garantem a confiabilidade, eficiência e facilidade de manutenção do sistema. Na elaboração dos requisitos não-funcionais do SGA Sumé, definimos que o sistema deve:

1. Ser capaz de lidar com um número cada vez maior de cursos, inscrições, matrizes curriculares e matérias sem degradação de desempenho.
2. Fornecer tempos de resposta rápidos para interações do usuário e processamento de dados.
3. Executar consultas de banco de dados otimizadas para lidar com operações complexas com eficiência.
4. A API (*Application Programming Interface*) deve ser intuitiva e de fácil utilização, permitindo uma fácil navegação e gestão dos cursos e entidades relacionadas.
5. Ter alta disponibilidade, com tempo de inatividade mínimo.
6. Implementar procedimentos de tratamento e recuperação de erros para manter a estabilidade do sistema.
7. A arquitetura do sistema deve ser modular para facilitar atualizações e manutenção.
8. O código deve seguir padrões de codificação claros e consistentes para garantir a legibilidade e facilidade de melhorias futuras.

3.3 DESIGN DO SOFTWARE

Aplicando DDD como nossa abordagem de design de software e os conceitos estudados na revisão teórica, detalharemos o domínio de negócio específico ao qual o software se destina, os problemas, necessidades, funcionalidades e regras.

3.3.1 LINGUAGEM UBÍQUA

Para criar uma linguagem ubíqua dentro do contexto de *Domain-Driven Design* (DDD) para *Courses*, *Classroom*, e *Activities*, é essencial estabelecer uma terminologia clara e compartilhada que todos os envolvidos no projeto possam entender e usar consistentemente.

Usuário *(User)*: Indivíduo que interage com o sistema e pode assumir diferentes papéis, como estudante, educador, administrador ou coordenador.

Papel do usuário (*User Role*): Categoria que define o conjunto de permissões e responsabilidades atribuídas a um usuário dentro do sistema.

Estudante (*Student*): Usuário que está inscrito em cursos e participa de atividades de aprendizagem.

Educador (*Teacher*): Usuário responsável por ministrar cursos, criar e gerenciar atividades e avaliações.

Administrador (*Administrator*): Usuário com privilégios elevados para criar e gerenciar outros usuários e papéis de usuários, assim como outras entidades dentro do sistema.

Coordenador (*Coordinator*): Usuário com privilégios elevados para criar e gerenciar cursos, turmas, atividades, assim como outras entidades dentro do sistema.

Curso (*Course*): Uma sequência estruturada de conteúdo educacional destinada a transmitir conhecimento e habilidades em um determinado assunto.

Inscrição do Curso *(Course Subscription)*: Relacionamento que representa a inscrição de um usuário em um curso.

Matriz Curricular *(Matrix)*: Estrutura que define a organização e os requisitos de um curso.

Matérias da Matriz (*Matrix Subjects*): Associação entre matérias e a matriz curricular a que pertencem.

Matéria (*Subject*): Tópico específico ou unidade de estudo dentro de um curso.

Turma (*Classroom*): O ambiente virtual onde o ensino e a aprendizagem ocorrem.

Agenda da Turma (*Classroom Timetable*): Agenda de aulas para uma turma específica.

Inscrição da Turma (*Classroom Subscription*): Relacionamento que representa a alocação de alunos a uma turma.

Aula da Turma (*Classroom Lesson*): Sessão de ensino individual que ocorre em uma turma.

Aula (*Lesson*): Sessão de ensino individual que faz parte de um curso.

Atividade da Aula (*Activity Lesson*): Aula associada a uma atividade de aprendizagem.

Atividade (*Activity*): Conteúdo, tarefa ou exercício que faz parte do processo de aprendizagem. Uma aula é composta de uma ou mais atividades.

Registro de Atividade *(Activity Log)*: Registro da realização de uma atividade por um aluno contendo o progresso, data e hora de início, data e hora final.

Recurso de aprendizado (*Resource*): Material ou ferramenta disponível para suporte ao ensino e aprendizagem. Um recurso pode ser um vídeo, quiz, documento, jogo educativo, entre outros.

3.3.2 CONTEXTOS DELIMITADOS

Para definir os "Contextos Delimitados" (*Bounded Contexts*), precisamos pensar no encapsulamento de uma funcionalidade específica e no seu modelo de domínio correspondente, projetando-o para ser autônomo, com interfaces bem definidas para interagir com outros contextos quando necessário. Conforme mostra a Fig. 4, o SGA Sumé terá os seguintes contextos:

Contexto de Cursos *(Courses Context):* Gerenciamento de cursos, inscrições, matrizes curriculares e matérias.

Contexto de Turmas *(Classroom Context):* Alocação de turma, agendamento de aulas e gerenciamento de inscrições em turmas.

Contexto de Atividades *(Activities Context)*: Criação e rastreamento de atividades educacionais, gerenciamento de recursos didáticos e lições.

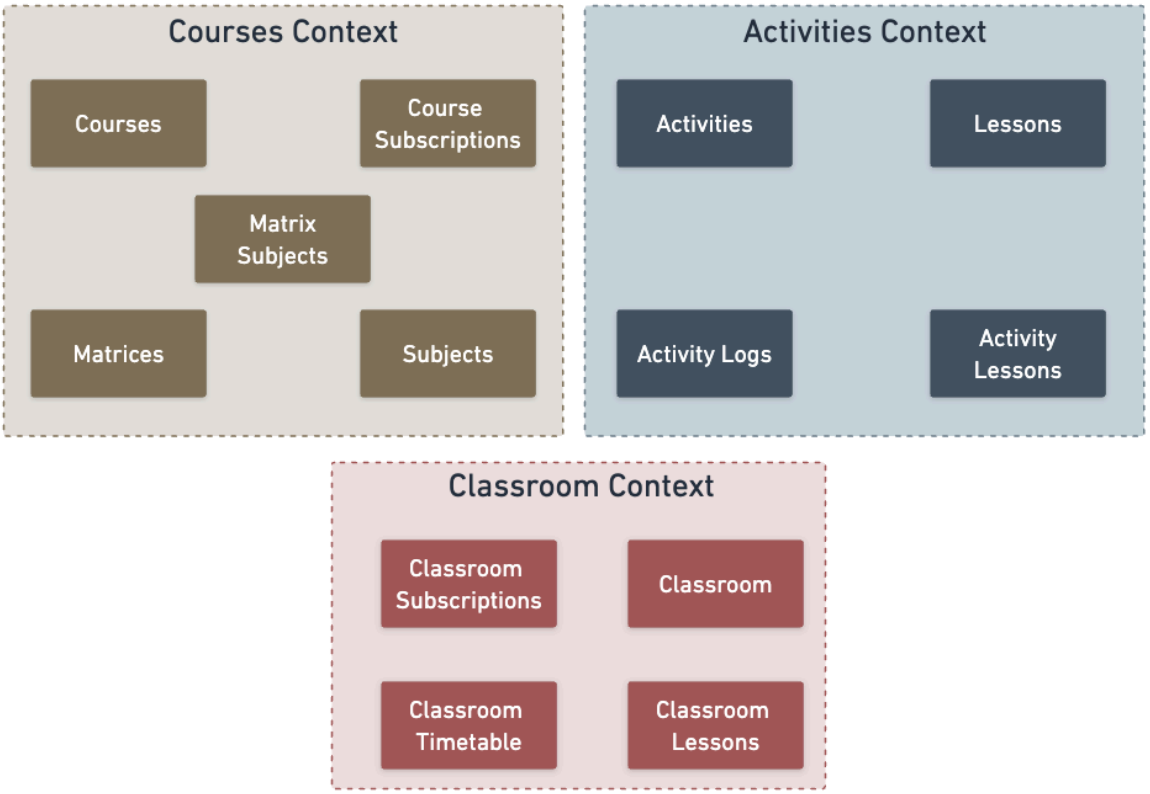

Figura 4. Contextos delimitados do sistema

Fonte: Produzido pela autora

A imagem acima ilustra a organização dos contextos delimitados e algumas entidades importantes de cada domínio. Esses contextos fazem parte do Domínio Principal do sistema e trabalham juntos para fornecer uma experiência de aprendizagem completa, permitindo que os alunos se inscrevam em cursos, participem de aulas e completem atividades relacionadas ao seu processo educacional.

3.3.4 MODELO DE DOMÍNIO

Para criar um modelo de domínio utilizamos os Contextos Delimitados como base e a partir deles identificamos as raízes agregadas, entidades de domínio e seus atributos, além de eventos e serviços de domínio.

Contexto de Cursos *(Courses Context)*

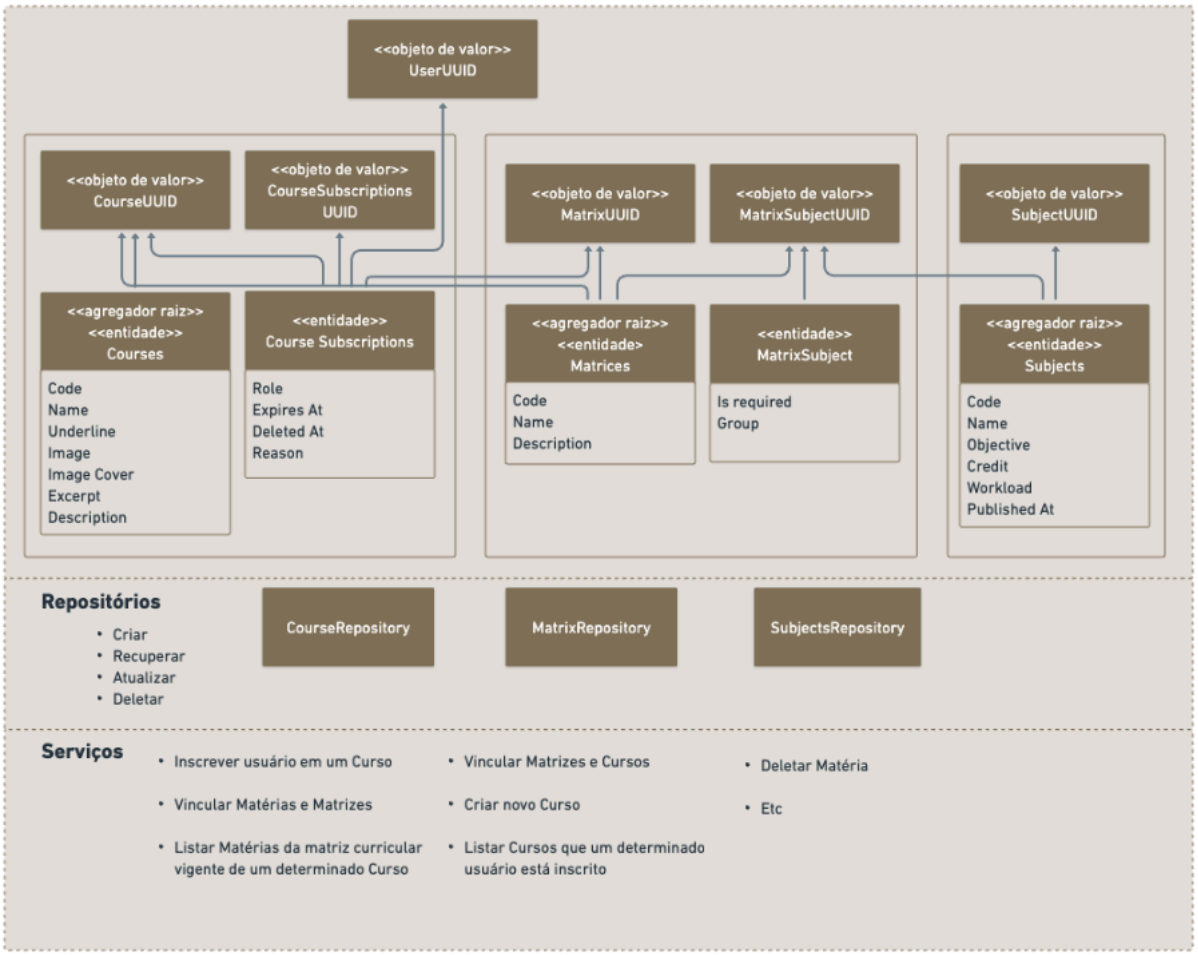

Figura 5. Courses: Modelos do domínio

Fonte: Produzido pela autora

Conforme mostra a Fig. 5, o agregador raiz *Courses*, agrega informações sobre o cursos e usuários inscritos, já o agregador raiz *Matrices*, agrega matrizes curriculares e matérias das matrizes curriculares e o agregador raiz *Subjects* agrega matérias. Sempre que um estudante deseja se inscrever em um curso, por exemplo, *Courses* interage com *Matrices* para validar e criar a inscrição. A entidade *Course*, possui uma regra de negócio que define que um curso pode ou não ter uma matriz associada, essa regra deve ser implementada através do objeto de valor *CourseUUID* dentro da entidade *Matrix*. Na entidade *MatrixSubject* o atributo *Group* identifica em qual agrupamento da matriz do curso essa matéria está (1º, 2º, 3º bimestre/semestre/módulo, etc).

Contexto de Turmas *(Classroom Context)*

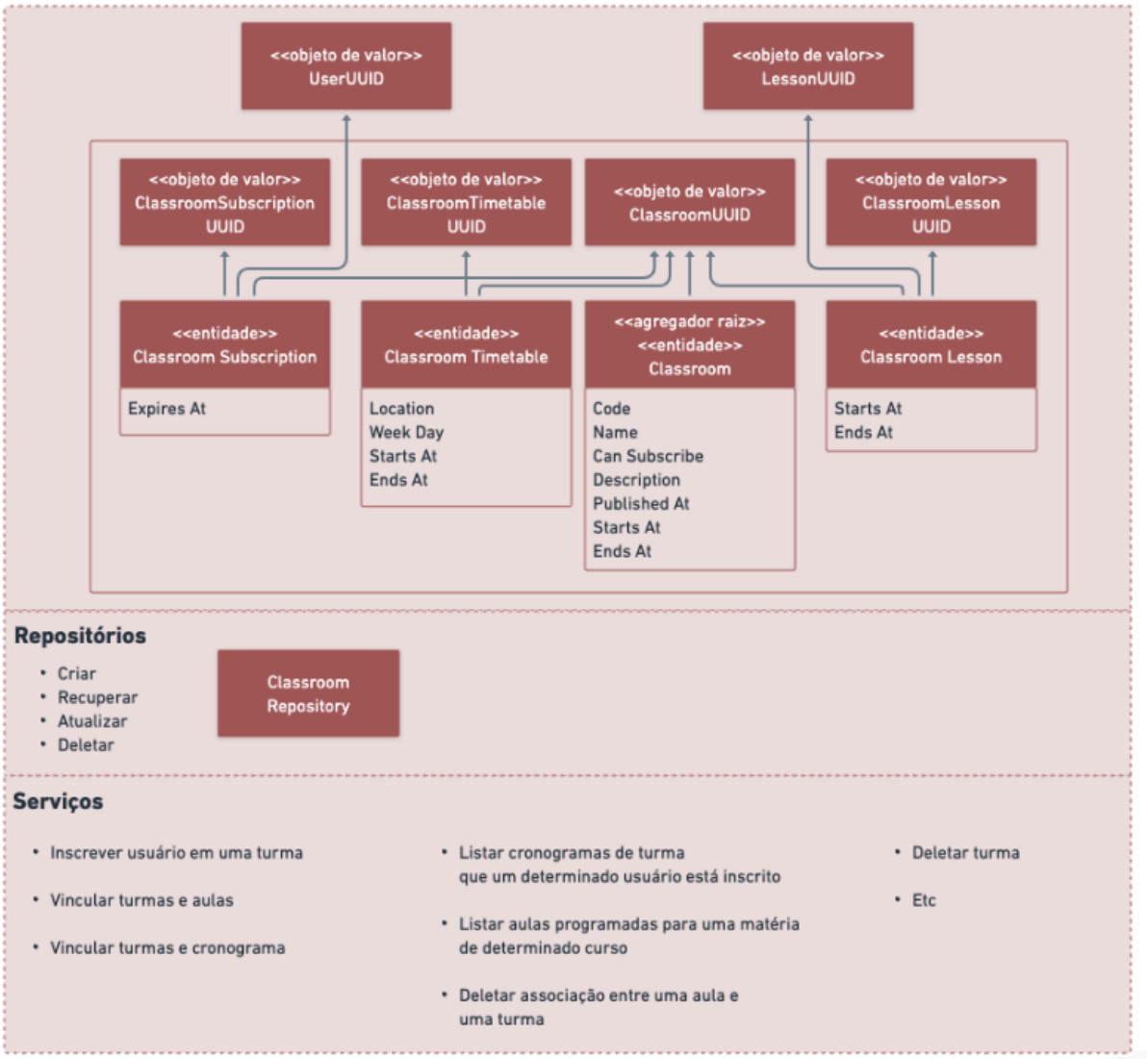

Figura 6. Classroom: Modelos do domínio

Fonte: Produzido pela autora

Na Fig. 6, o agregador raiz *Classroom*, agrega informações sobre turmas, suas agendas, aulas e usuários inscritos. Sempre que um estudante deseja se inscrever em uma turma, por exemplo, *Classroom* interage com *ClassroomSubscription* para validar e criar a inscrição. A entidade *ClassroomTimetable*, possui uma regra de negócio que exige que uma turma seja associada, essa regra deve ser implementada através do objeto de valor

ClassroomUUID, essa entidade também possui os atributos *Week Day* e *Starts At* que identificam em qual dia da semana e horário a turma é aberta.

Contexto de Atividades *(Activities Context)*:

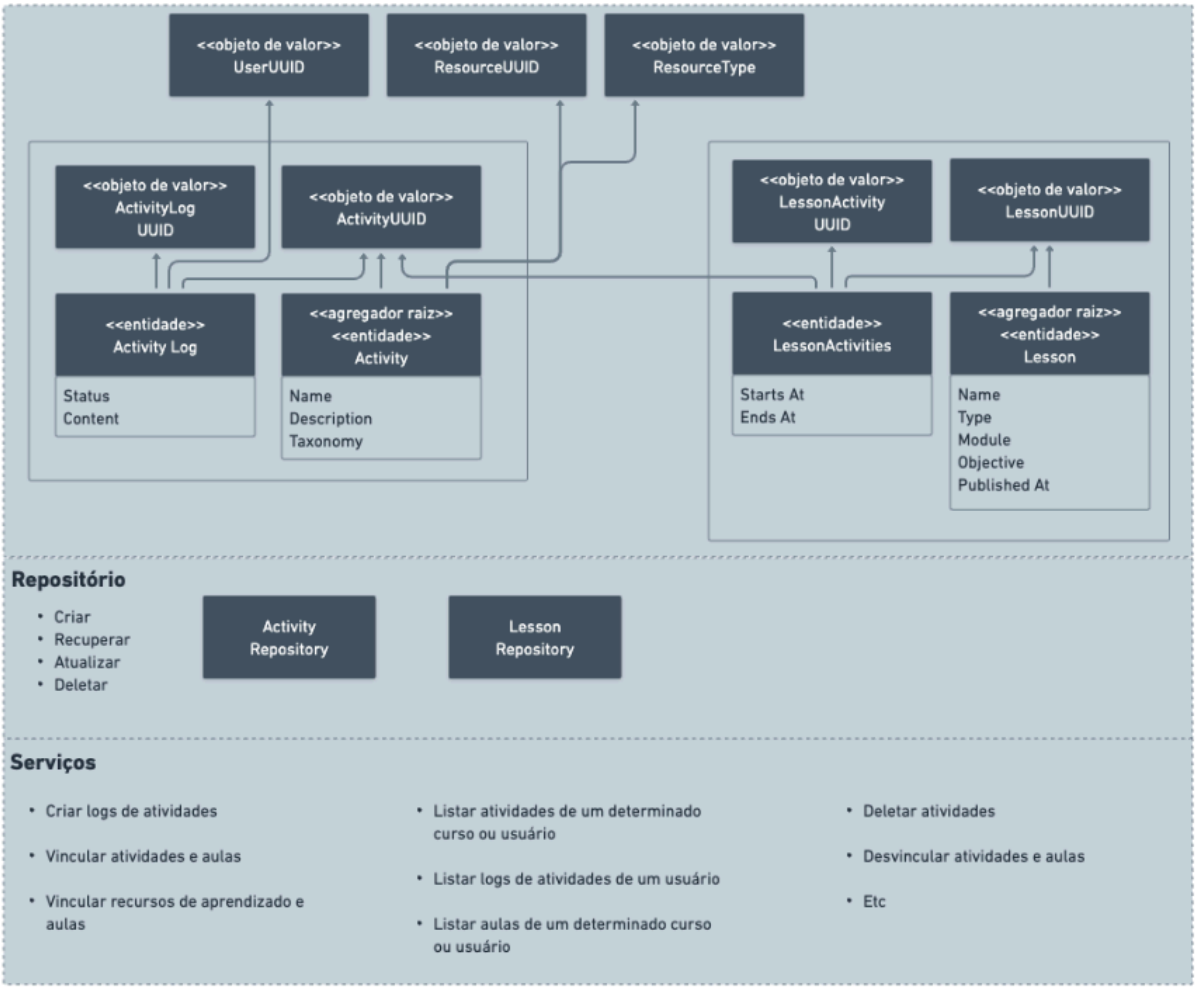

Figura 7. Activities: Modelos do domínio

Fonte: Produzido pela autora

Observa-se na Fig. 7 que o agregador raiz *Activitiy* reúne informações sobre atividades e *logs* das atividades realizadas, já o agregador raiz *Lessons* agrega aulas e suas atividades. A entidade *Activity* possui uma regra de negócio que exige a associação com um *resource* e seu tipo (vídeo, quiz, scorm, pdf, etc), cada tipo será um microsserviço desenvolvido em trabalhos futuros, essa regra deve ser implementada através dos objetos de valor *ResourceUUID* e *ResourceType*. Essa

entidade também possui o atributo *Taxonomy* que identifica se essa atividade pode ser associada a algum objetivo educacional da Taxonomia de Bloom (criar, avaliar, aplicar, compreender, lembrar). Para adicionar atividades à alguma aula, *Lesson* interage com *Activity* para validar e criar a associação

Para acessar todos os dados temos repositórios que encapsulam a lógica necessária, como conexão com a base de dados, execução de consultas e gerenciamento de erros.

3.3.5 MAPAS DE CONTEXTO

Para criar um modelo de domínio utilizamos os Contextos Delimitados como base e a partir deles planejar as dependências e interações entre diferentes partes do sistema.

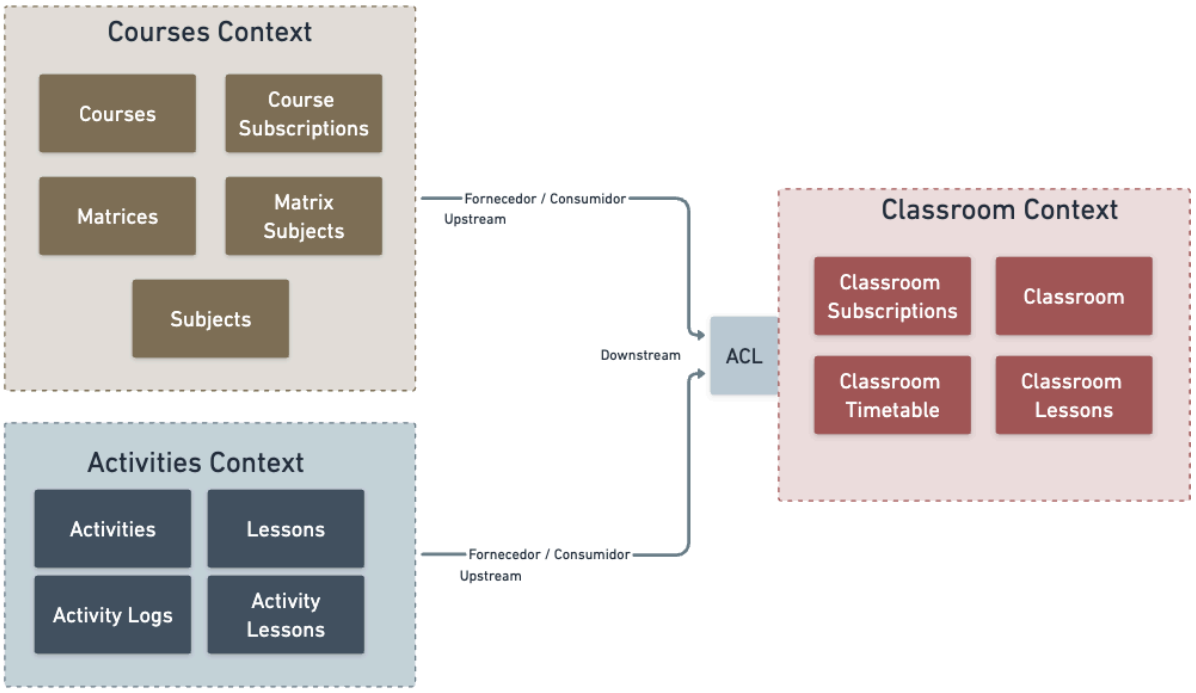

Figura 8. Mapa de Contexto

Fonte: Produzido pela autora

A Fig 8 apresenta o mapa de contexto onde o *Courses Context* é o domínio principal onde são gerenciados os cursos, inscrições de cursos, matrizes e

disciplinas. Este contexto é crucial para a definição e organização dos cursos oferecidos. O *Activities Context* é um subdomínio de suporte onde são gerenciadas as atividades de aprendizado, aulas e *logs* de atividades. Este contexto apoia o processo de ensino e aprendizado. Já *Classroom Context* é um subdomínio de suporte responsável pela gestão das turmas, horários, inscrições e aulas. Ele consome informações dos contextos de cursos e atividades para organizar e gerenciar as salas de aula de forma eficiente.

3.3.6 EVENTOS DE DOMÍNIO

Considerando a importância dos eventos de domínio no contexto do *Domain-Driven Design* (DDD), podemos elaborar uma série de eventos e seus encadeamentos. Esses eventos refletem mudanças de estado significativas e são usados para sincronizar dados e operações entre os microsserviços:

Inscrição no curso deletada (*CourseSubscriptionDeleted*): Emitido quando um usuário desfaz a inscrição em um curso.

- **SAGA:**
    - Deleção assíncrona de inscrições em turma no contexto de turmas.

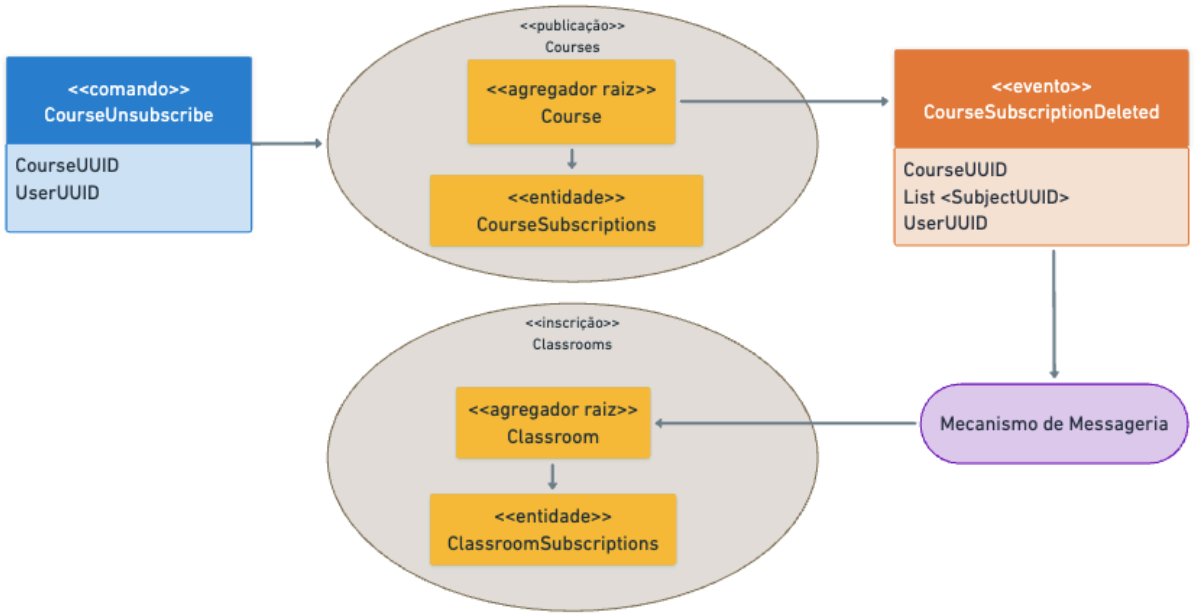

Figura 9. *CourseSubscriptionDeleted*

Fonte: Produzido pela autora

Ao receber um comando, para desfazer a inscrição do usuário em um curso, onde os identificadores são informados respectivamente nos atributos *UserUUID* e *CourseUUID* do comando, abstraindo a cama de segurança, o sistema deve:

No contexto de *Courses*:

1. Deletar todas as entidades *CourseSubscriptions* que tem *CourseUUID* e *UserUUID* correspondente.
2. Recuperar todas as matérias associadas à matriz do curso, quando houver.
3. Disparar um evento CourseSubscriptionDeleted que carrega em suas propriedades o CourseUUID, UserUUID e a lista de *Subjects* associadas à matriz do curso, quando houver.

No contexto de Classroom:

1. Receber o evento CourseSubscriptionDeleted
2. Se houver matérias associadas ao evento na propriedade List<Subject>:

    a. Deletar todas as entidades ClassroomSubscriptions onde CourseUUID, UserUUID e SubjectUUID correspondem aos valores armazenados nos atributos do evento
3. Se não houver matérias associadas ao evento na propriedade List<Subject>:
    a. Deletar todas as entidades ClassroomSubscriptions onde CourseUUID, UserUUID correspondem aos valores armazenados nos atributos do evento.

Matéria deletada (SubjectDeleted): Emitido quando uma matéria é removida.

- **SAGA:**
    - Deleção assíncrona de todas as associações com matrizes.
    - Deleção assíncrona de turmas.
    - Deleção assíncrona de inscrições em turmas.
    - Deleção assíncrona de cronogramas de turmas.
    - Deleção assíncrona de aulas de turmas.

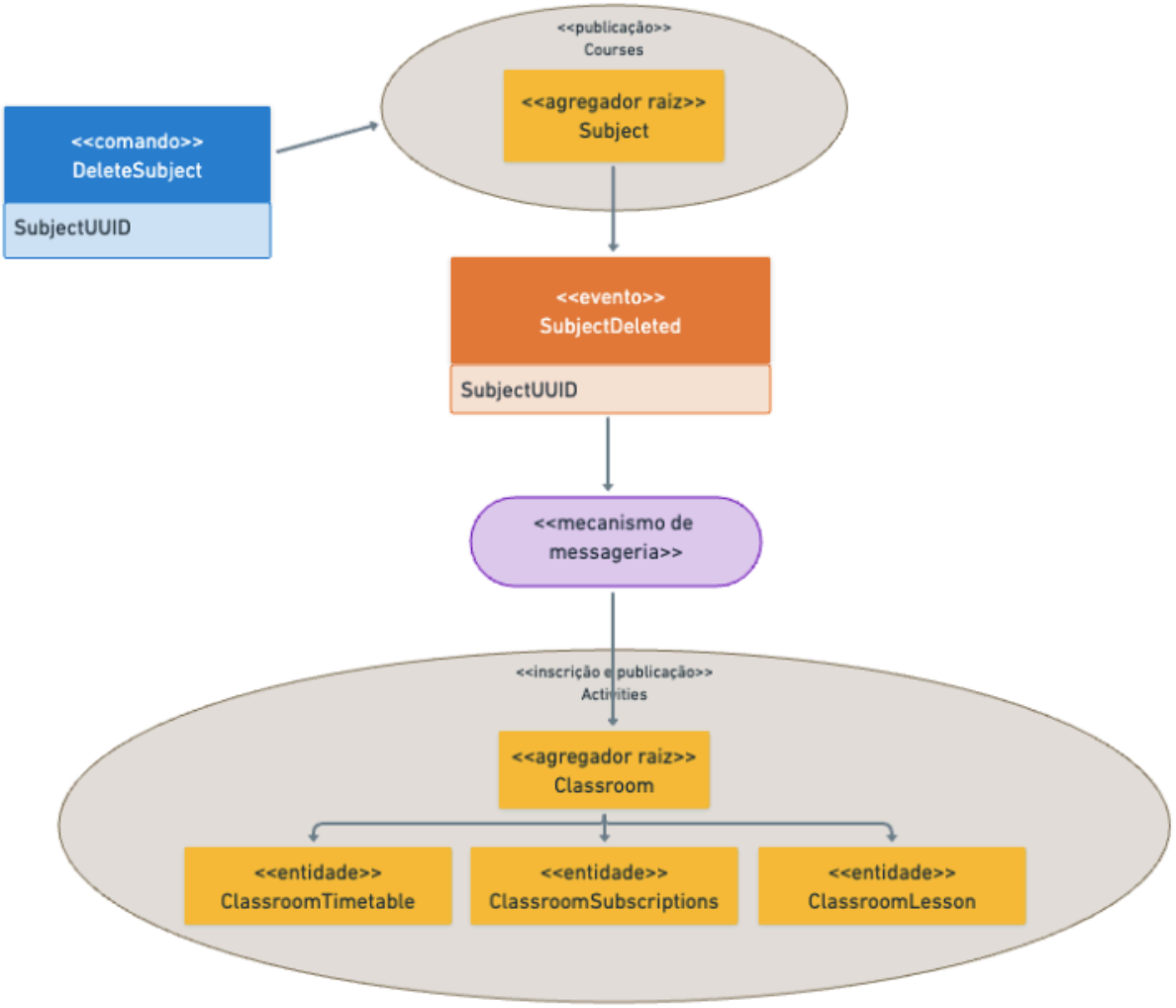

Figura 10. *SubjectDeleted*

Fonte: Produzido pela autora

Ao receber um comando, para deletar uma matéria, onde o identificador é informado no atributo *SubjectUUID* do comando, abstraindo a cama de segurança, o sistema deve:

No contexto de *Courses*:

4. Deletar a entidade *Subject* que tem o *SubjectUUID* correspondente.
5. Disparar um evento *SubjectDeleted* que carrega em suas propriedades o *SubjectUUID*.
6. Receber o evento *SubjectDelete*

7. Deletar todas as entidades *MatrixSubjects* que tem o *SubjectUUID* correspondente associado.

No contexto de *Classroom*:

1. Receber o evento *SubjectDeleted*
2. Para todas as entidades Classroom que tem o *SubjectUUID* correspondente associado:
    a. Deletar a entidade *Classroom*
    b. Deletar todas as entidades *ClassroomSubscriptions* que tem o *ClassroomUUID* correspondente associado.
    c. Deletar todas as entidades *ClassroomTimetable* que tem o *ClassroomUUID* correspondente associado.
    d. Deletar todas as entidades *ClassroomLessons* que tem o *ClassroomUUID* correspondente associado.

Aula deletada (*LessonDeleted*): Emitido quando uma aula é removida.

- **SAGA:**
    - Deleção assíncrona de aulas de turmas no contexto de turmas.
    - Deleção assíncrona de todas as associações com atividades.

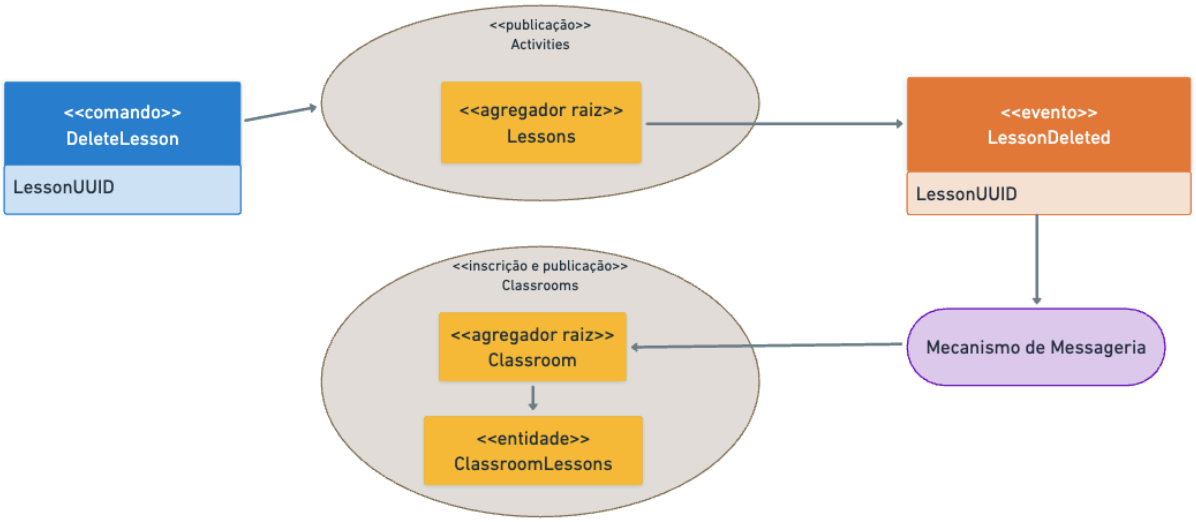

Figura 11. *LessonDeleted*

Fonte: Produzido pela autora

Ao receber um comando, para deletar uma aula, onde o identificador é informado no atributo *LessonUUID* do comando, abstraindo a cama de segurança, o sistema deve:

No contexto de *Activites*:

1. Deletar a entidade *Lesson* que tem o *LessonUUID* correspondente.
2. Disparar um evento *LessonDeleted* que carrega em suas propriedades o *LessonUUID*.

No contexto de *Classroom*:

1. Receber o evento *LessonDeleted*
2. Deletar todas as entidades *ClassroomLessons* que tem o *LessonUUID* correspondente associado.

Estes eventos de domínio, dentre outros, são fundamentais para a comunicação entre os microsserviços e para garantir a consistência e a integridade dos dados em operações que são causalmente relacionadas. A implementação desses eventos pode ser realizada como SAGAs coreografadas através de mecanismos de mensageria ou sistemas de eventos, como filas de mensagens ou tópicos de eventos, que permitem a publicação e assinatura de eventos entre microsserviços de forma assíncrona e desacoplada.

A escolha por SAGAs coreografadas é justificada por várias razões, dentre elas a autonomia dos microsserviços. Na coreografia, cada serviço é responsável por publicar eventos e reagir a eventos de outros serviços, permitindo que os microsserviços mantenham um alto grau de independência. Além disso, sem um orquestrador central, evita-se a criação de um ponto único de falha e reduz-se a

complexidade de manutenção de um componente centralizado. A coreografia também facilita a adição de novos serviços ou a modificação de fluxos existentes sem a necessidade de alterar um componente central de orquestração.

3.4 MICROSSERVIÇOS

A arquitetura de microsserviços dividiu o Sistema de Gestão de Aprendizagem em uma coleção de serviços pequenos e autônomos, cada um responsável por uma funcionalidade específica. Cada contexto delimitado definido (Courses, Activities e Classrooms) foi desenvolvido como um microsserviço que opera de forma independente e implementa serviços de criação, recuperação, atualização e remoção de dados. No desenvolvimento, foram utilizados repositórios de códigos separados que estão disponíveis nos links https://github.com/marianysilva/microservice-course/tree/dev, https://github.com/marianysilva/microservice-classroom/tree/dev e https://github.com/marianysilva/microservice-activity/tree/dev. Esses repositórios foram gerenciados utilizando o GitHub, que é uma plataforma de hospedagem de código-fonte e arquivos com controle de versão usando o Git.

Essa divisão permitiu que cada serviço fosse desenvolvido, testado e mantido de forma independente, promovendo uma maior modularidade e facilitando a gestão do código. Além disso, cada microsserviço pode ser implantado de forma independente, o que significa que atualizações ou correções em um serviço não requerem a reimplantação de toda a aplicação. Por exemplo, se houver uma atualização necessária no serviço "Course Subscriptions" no contexto de "Courses", essa atualização pode ser feita sem interromper os serviços "Activities" ou "Classroom". Isso reduz o tempo de inatividade e permite uma entrega contínua de novas funcionalidades e correções de bugs.

Os microsserviços, também permitem que cada serviço seja escalado de forma independente, de acordo com a demanda. Se o serviço "Activities" no contexto de "Activities" estiver enfrentando alta demanda, ele pode ser escalado sem afetar outros serviços como "Courses" ou "Classroom". Isso é possível porque cada microsserviço pode ser implantado em contêineres ou máquinas virtuais separadas, permitindo uma escalabilidade horizontal eficiente.

3.5 DADOS DISTRIBUÍDOS

A base de dados para microsserviços é um componente vital na arquitetura de sistemas modernos, especialmente em ambientes distribuídos. A fase de *Design* envolve tomar decisões críticas sobre como os dados serão armazenados, acessados e gerenciados dentro do ecossistema de microsserviços. Nessa fase precisamos compreender os requisitos específicos de dados de cada microsserviço, incluindo o tipo de dados, volume, frequência de acesso e requisitos de desempenho.

3.5.1 BANCOS DE DADOS

A análise das entidades e objetos de valor necessários dos microsserviços *Courses*, *Classroom* e *Activities* revela uma variedade de tipos de dados que precisam ser armazenados e gerenciados:

Identificadores Únicos (UUID*)*: Utilizados para garantir a unicidade global de registros, facilitando a integração e referência entre diferentes microsserviços.

Inteiros (*integer*): Empregados para atributos de identificação numérica, como IDs de tabelas, ou número de créditos de uma matéria.

Strings (*varchar*): Usados para armazenar textos de tamanho variável, como nomes, descrições e códigos.

Textos (*text*): Para armazenar informações mais extensas, como descrições detalhadas de cursos ou objetivos de aulas.

Data e hora *(timestamp)*: Registram datas e horas de criação, atualização, exclusão e publicação de registros. As tabelas terão o atributo *deleted_at*, um carimbo de data e hora para indicar registros logicamente excluídos, permitindo a recuperação e auditoria de dados aplicando a técnica de *soft delete*.

Booleanos (*boolean*): Indicam valores verdadeiros ou falsos, como a obrigatoriedade de uma matéria na matriz curricular.

JSON: Armazenam estruturas de dados complexas e flexíveis, como *logs* de atividades de forma compacta.

Existem duas categorias principais de sistemas de gerenciamento de banco de dados (SGBDs) que podem ser consideradas para armazenar os tipos de dados mencionados: bancos de dados relacionais e NoSQL.

Bancos de Dados Relacionais: São baseados no modelo relacional de dados e oferecem recursos como integridade referencial, transações ACID e linguagem de consulta estruturada (SQL). Eles são adequados para dados estruturados e relações complexas entre entidades.

Bancos de Dados NoSQL: Incluem várias categorias, como chave-valor, orientados a documentos, colunares e grafos. São conhecidos por sua escalabilidade horizontal e flexibilidade de esquema, sendo úteis para grandes volumes de dados não estruturados ou semiestruturados.

A escolha de PostgreSQL como SGBD nos microsserviços *Courses*, *Classroom* e *Activities* é justificada pelo suporte robusto à integridade referencial, essencial para manter a consistência dos dados em um sistema com múltiplas entidades inter-relacionadas, como cursos, matrículas, aulas e *logs* de atividades.

Outro ponto crítico é a capacidade de realizar transações ACID, crucial para operações que envolvem múltiplas alterações nos dados, garantindo que todas as operações sejam concluídas com sucesso ou que nenhuma seja aplicada. Além disso, o PostgreSQL é um SGBD maduro e confiável, que oferece muitos recursos nativos e com um histórico comprovado de desempenho e segurança em aplicações críticas.

A imagem abaixo ilustra a arquitetura do sistema de microserviços. O sistema é composto por três microserviços principais, cada um se comunica com seu próprio banco de dados PostgreSQL para armazenar e recuperar dados específicos, utilizando mensagens TCP/IP.

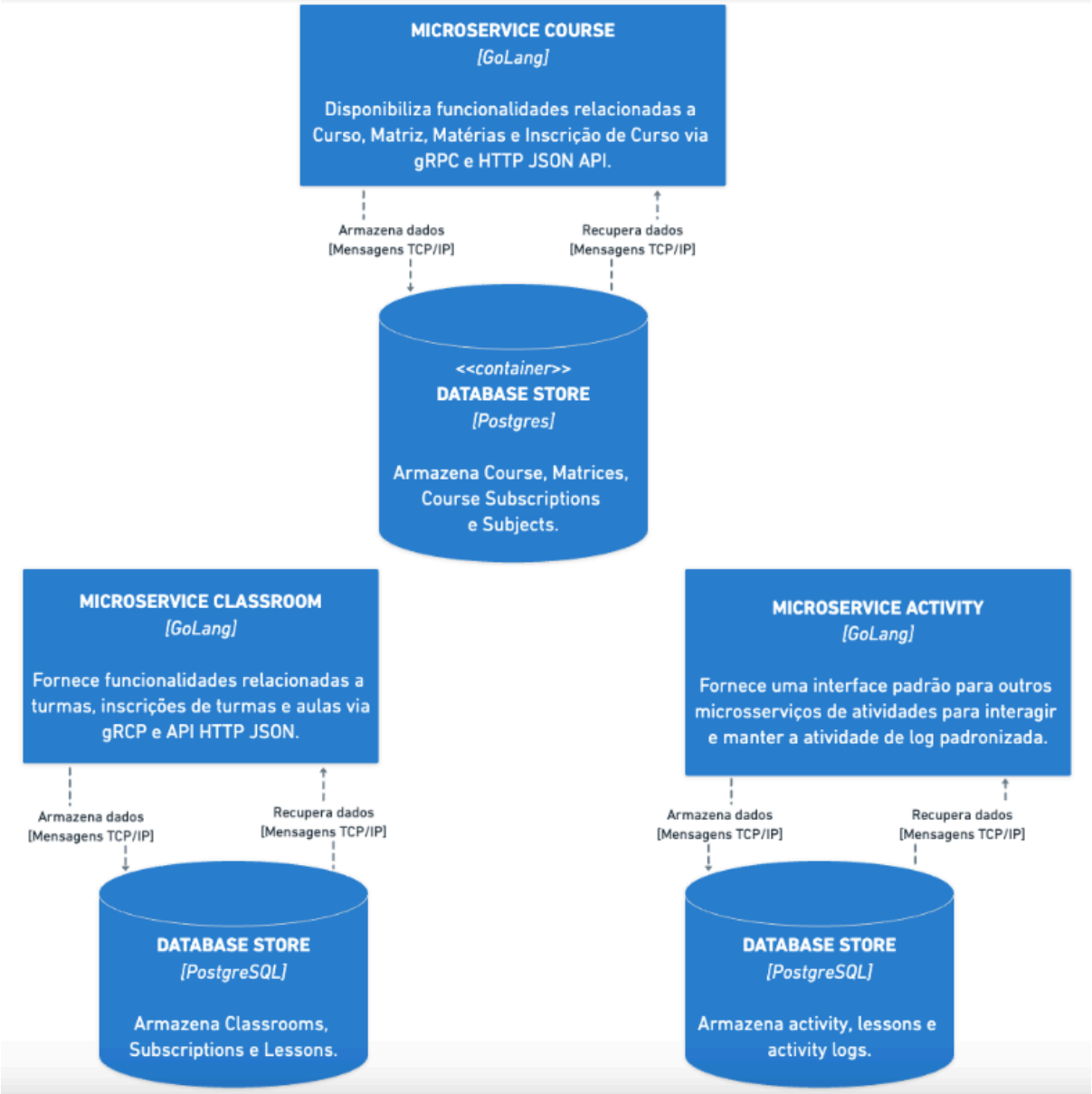

Figura 12. Microsserviços e Bancos de Dados

Fonte: Produzido pela autora

Para lidar com mudanças nos dados e na estrutura do banco de dados, adotamos a estratégia de migrações de dados e migrações de *schema*. A migração de dados envolve a transferência de dados de uma estrutura ou formato para outro, essencial quando temos mudanças significativas na forma como os dados são armazenados ou quando novos dados precisam ser integrados ao sistema. Já a migrações de *schema*, envolve alterações na estrutura do banco de dados, como a

adição de novas tabelas, colunas ou índices, essencial para suportar novas funcionalidades ou otimizar o desempenho do sistema. Essa abordagem permite que as alterações sejam aplicadas de forma controlada e segura, garantindo a integridade e a consistência dos dados ao longo do tempo.

3.5.2 MODELAGEM RELACIONAL

O diagrama Entidade-Relacionamento (ER) é uma ferramenta essencial na modelagem de dados conceituais. Ele descreve as entidades envolvidas em um domínio de negócios, seus atributos e as relações entre elas, representando de forma abstrata as estruturas que o banco de dados da aplicação armazenará, facilitando a compreensão e a comunicação entre desenvolvedores e especialistas do domínio.

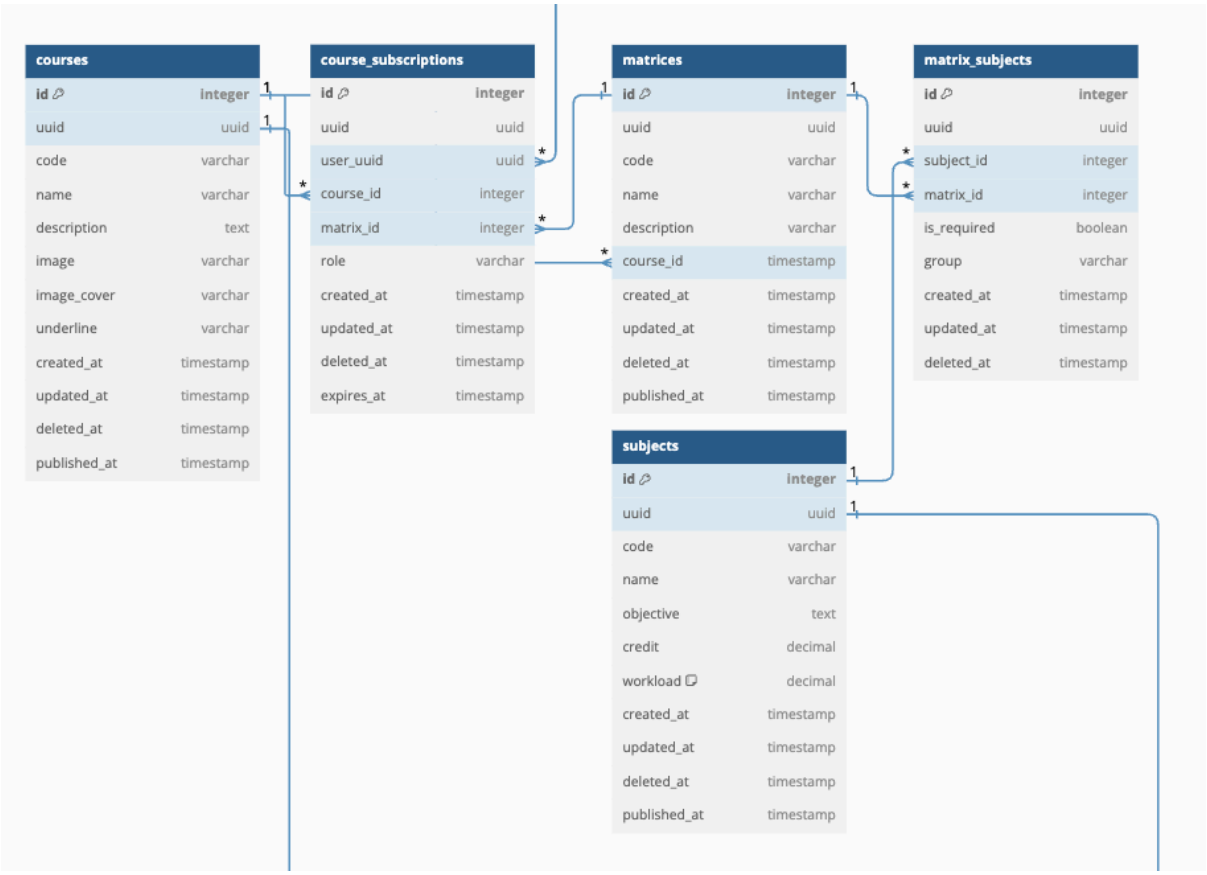

Figura 13. Courses Bancos de Dados

Fonte: Produzido pela autora

A fig. 13 ilustra o diagrama entidade-relacionamento dos dados para o microsserviço *Courses* que gerencia informações sobre cursos, matrizes curriculares, inscrições e matérias, representadas pelas tabelas *courses*, *matrices*, *course_subscriptions*, e *subjects* respectivamente. Cada tabela possui uma chave primária inteira, chamada *id*, que ajuda na performance do banco de dados e um identificador único universal, chamado *uuid*, além de atributos específicos para armazenar dados relevantes, como *name*, *description* e *code*. As relações entre as tabelas são estabelecidas por meio de chaves estrangeiras, utilizando identificador único universal (*uuid*) para relações externas ao microsserviço, como *user_uuid* na tabela *courses* que permite a associação de cursos e usuários e utilizando identificador único inteiro (*id*) para relações internas, como *course_id* na tabela matrices que permite a associação de cursos e suas matrizes curriculares.

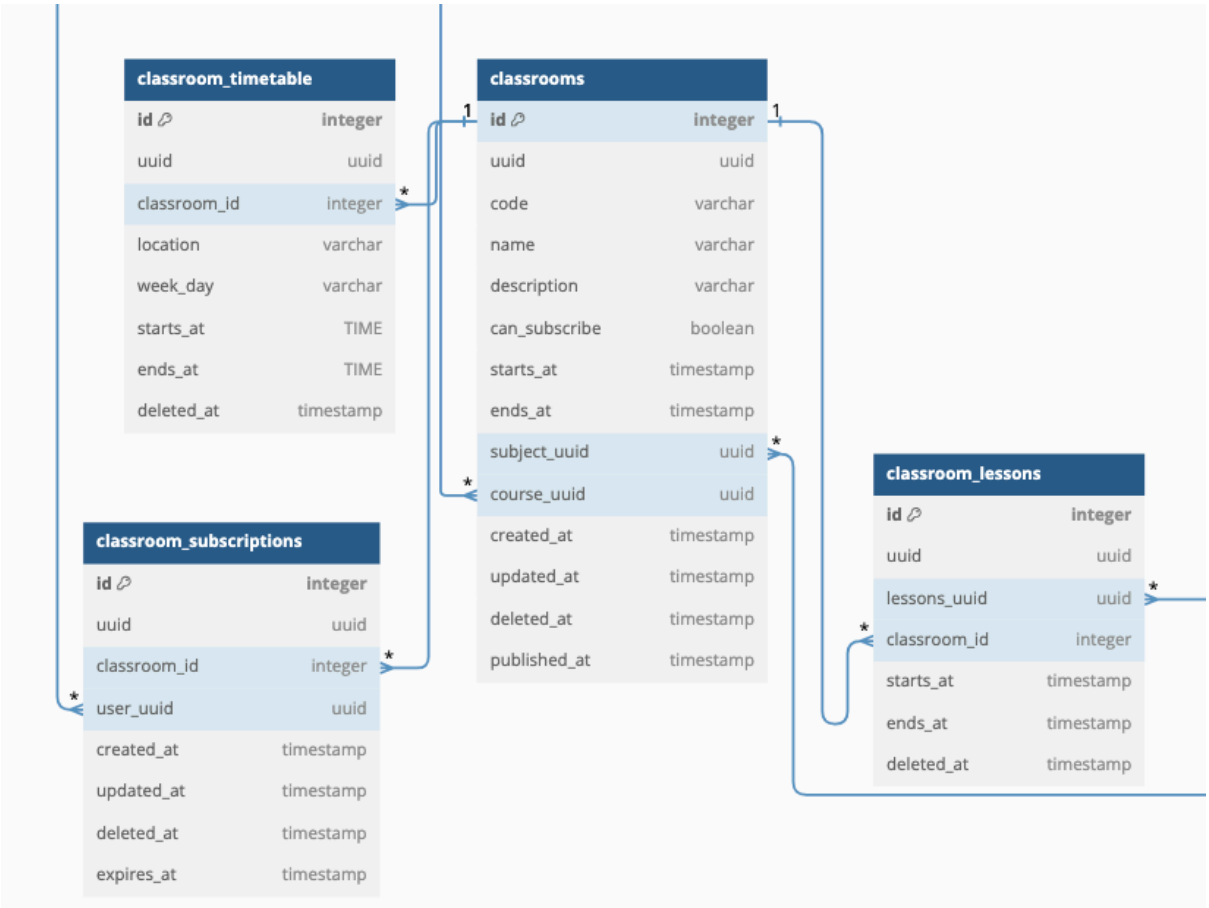

Figura 14. Classroom Bancos de Dados

Fonte: Produzido pela autora

A fig. 14 ilustra o diagrama entidade-relacionamento dos dados para o microsserviço Classroom que gerencia informações sobre turmas, seus cronogramas, aulas, e inscrições, representadas pelas tabelas *classrooms*, *classroom_timetable*, *classroom_lessons*, e *classroom_subscriptions* respectivamente. Cada tabela possui uma chave primária inteira, chamada *id*, que ajuda na performance do banco de dados e um identificador único universal, chamado *uuid*, além de atributos específicos para armazenar dados relevantes, como name, description e code. As relações entre as tabelas são estabelecidas por meio de chaves estrangeiras, utilizando identificador único universal (uuid) para relações externas ao microsserviço, como "user_uuid" na tabela "classroom_subscriptions" que permite a associação de turmas e usuários e utilizando identificador único inteiro (id) para relações internas, como "classroom_id" na tabela "classroom_lessons" que permite a associação de turmas e aulas.

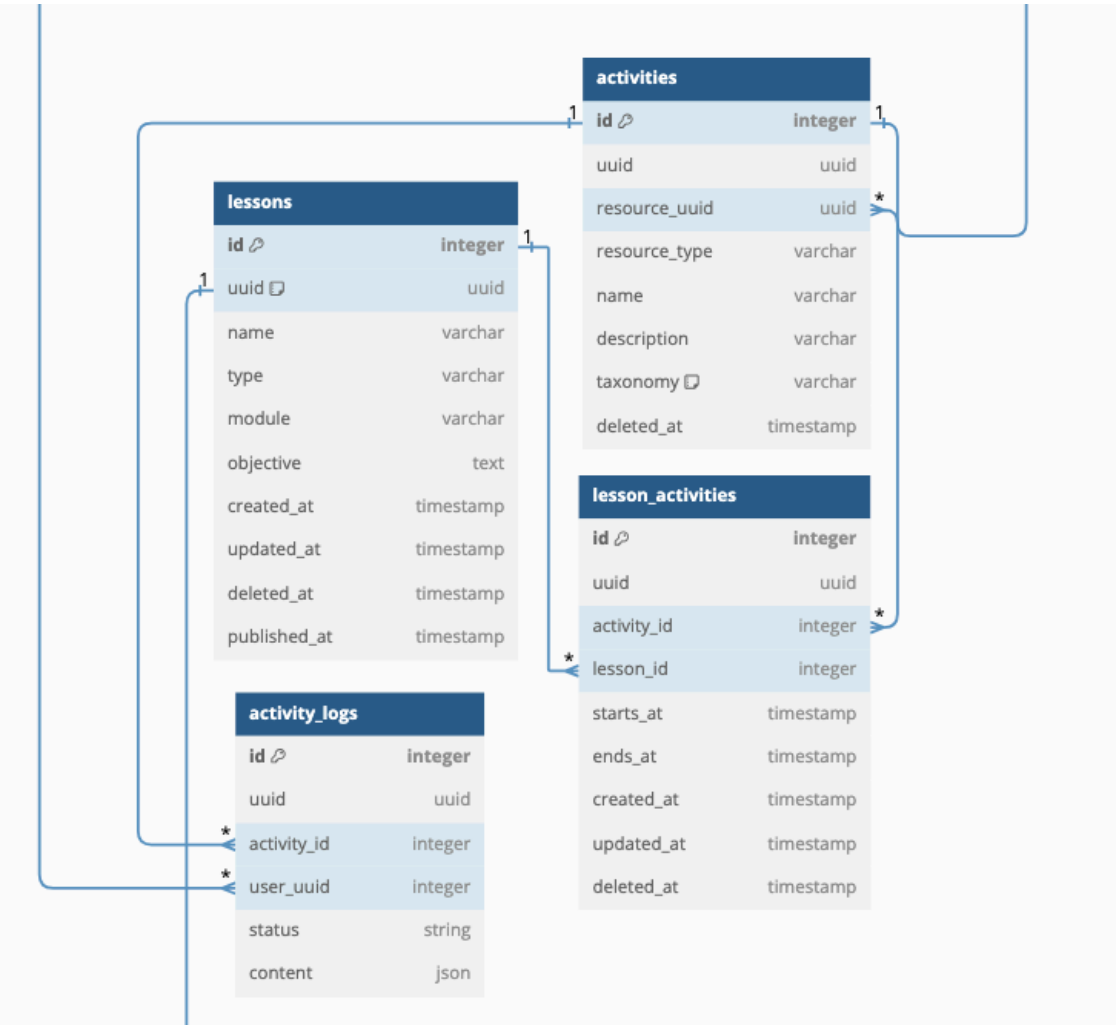

Figura 15. Activity Bancos de Dados

Fonte: Produzido pela autora

A fig. 14 ilustra o diagrama entidade-relacionamento dos dados para o microsserviço Activities que gerencia informações sobre as aulas, atividades e *logs*, representadas pelas tabelas "lessons", "activities", "lesson_activities", e "activity_logs". Cada tabela possui uma chave primária inteira, chamada "id", que ajuda na performance do banco de dados e um identificador único universal, chamado "uuid", além de atributos específicos para armazenar dados relevantes, como "name", "description" e "code". As relações entre as tabelas são estabelecidas por meio de chaves estrangeiras, utilizando identificador único universal (uuid) para relações externas ao microsserviço, como "user_uuid" na tabela

"classroom_subscriptions" que permite a associação de turmas e usuários e utilizando identificador único inteiro (id) para relações internas, como "classroom_id" na tabela "classroom_lessons" que permite a associação de turmas e aulas.

3.6 INTEGRAÇÃO E COMUNICAÇÃO ENTRE MICROSSERVIÇOS

Com o objetivo de implementar as funcionalidades básicas dos microsserviços de Courses, Classroom e Activities, previstas nos requisitos funcionais do projeto, como a criação, recuperação, atualização e deleção de dados, bem como os eventos de domínio detalhados no design do software, essa fase aprofunda a compreensão sobre a interação e integração dos diversos microsserviços através da exploração da forma como esses microsserviços se comunicam.

3.6.1 REST

Cada microsserviços desenvolvido neste trabalho possuem sua própria API REST, cuidadosamente documentada para garantir a clareza e a facilidade de uso. Essas APIs REST expõem todos os endpoints que implementam os requisitos funcionais do MVP. A documentação detalhada inclui descrições dos endpoints, métodos HTTP suportados, parâmetros de entrada, exemplos de requisições e respostas, além de códigos de status HTTP retornados. Essa abordagem não só facilita a integração e a comunicação entre os microsserviços, mas também permite que desenvolvedores externos compreendam e utilizem as APIs de maneira eficiente, promovendo a interoperabilidade e a manutenção do sistema. As fig. 16 à 19 mostram parte da documentação gerada para um dos microsserviços desenvolvidos.

Figura 16. Courses API REST: courses, matrices, matrix_subjects

Fonte: Produzido pela autora

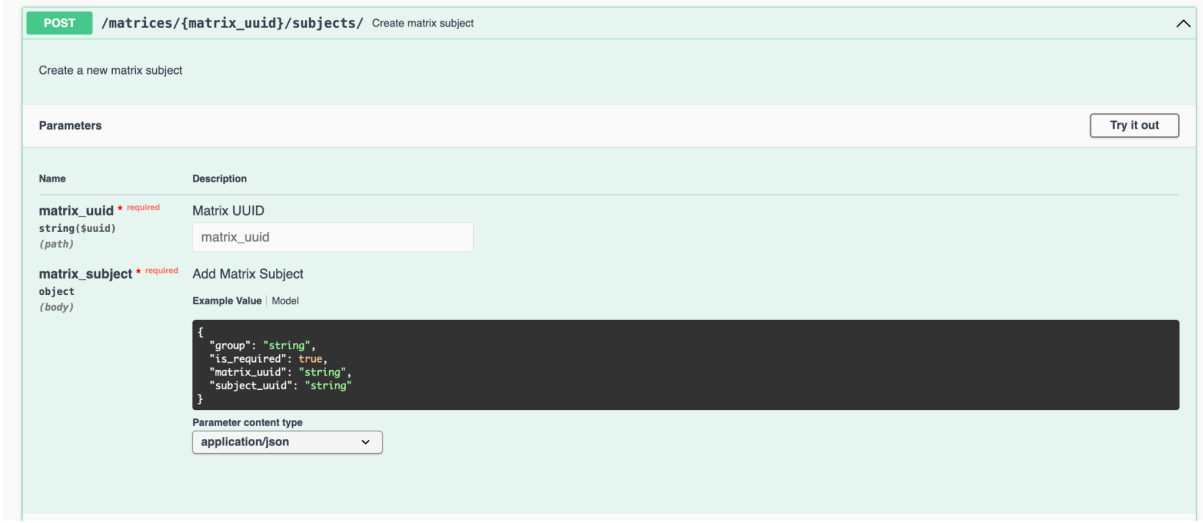

Figura 17. Courses API REST: REQUEST POST matrix_subjects

Fonte: Produzido pela autora

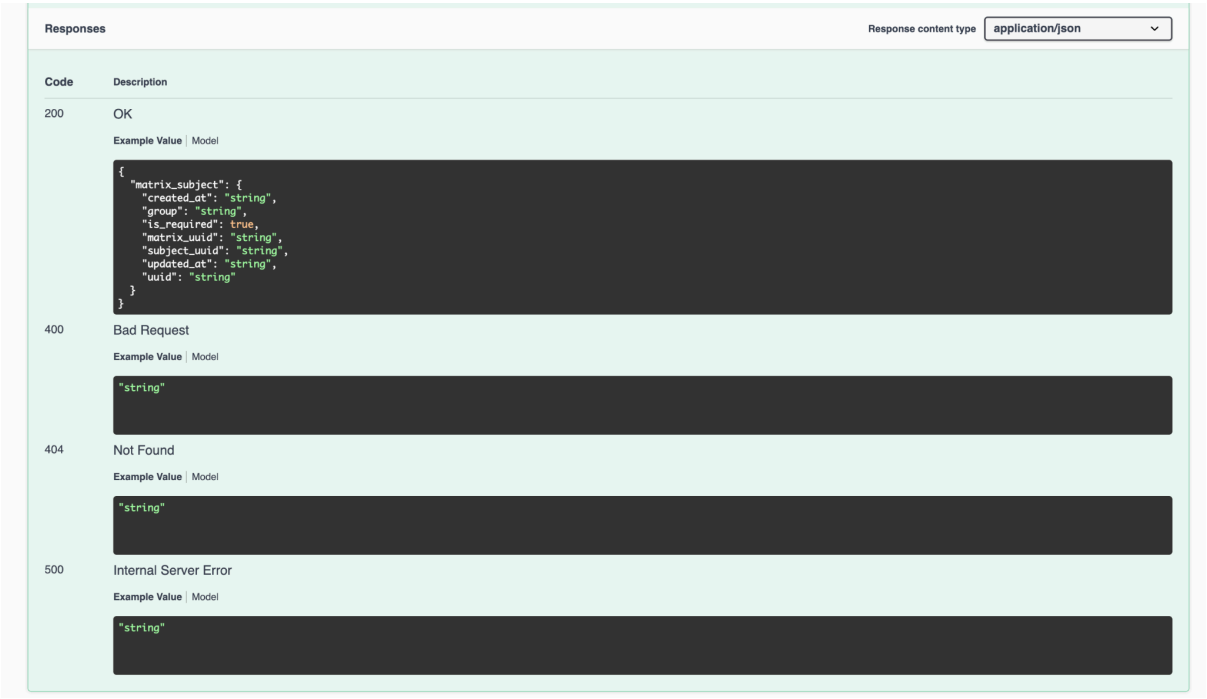

Figura 18. Courses API REST: RESPONSE POST matrix_subjects

Fonte: Produzido pela autora

Figura 19. Courses API REST: subjects, subscriptions

Fonte: Produzido pela autora

O microsserviço Courses, por exemplo, possui endpoints específicos para cada entidade, implementando os métodos HTTP adequados para as operações CRUD (Create, Read, Update, Delete). Para a criação de uma entidade, o endpoint utiliza o método POST, recebendo no corpo da requisição HTTP um JSON contendo todos os atributos necessários. Para a atualização de uma entidade existente, são utilizados endpoints com o método PUT, que recebem parâmetros para identificar a entidade a ser atualizada e um JSON no corpo da requisição com os atributos a serem modificados. A exclusão de entidades é realizada através de endpoints que

implementam o método DELETE, recebendo parâmetros que identificam a entidade a ser removida. Além disso, endpoints com o método GET foram criados para a recuperação e listagem das entidades. Os demais microsserviços seguem a mesma estrutura, implementando funcionalidades semelhantes, adaptadas às características específicas de cada entidade em seus respectivos contextos delimitados.

3.6.2 COMUNICAÇÃO SÍNCRONA COM GRPC

Google Remote Procedure Call (gRPC), é um framework de código aberto desenvolvido pelo Google, baseado em RPC, que utiliza o protocolo HTTP/2 para comunicação eficiente entre microsserviços. Ele utiliza o *Protocol Buffers* (Protobuf), um formato de serialização de dados binário que busca melhorar a transferência de dados, consumindo poucos recursos e transferindo uma maior quantidade de dados, oferecendo alta performance e tipagem forte (gRPC, 2024).

Uma das funcionalidades implementadas no sistema é a criação de turmas, no contexto de Classroom que recebe diversos atributos, incluindo os identificadores únicos CourseUUID (curso) e Subject UUID (matéria). Subject UUID é um atributo opcional. Para garantir a integridade dos dados e a consistência do sistema, é essencial validar que o curso e a matéria informados são válidos e se relacionam corretamente através de uma matriz curricular. A validação deve ser realizada de forma síncrona e eficiente para não comprometer a experiência do usuário. Para isso, optamos por utilizar o gRPC que fornece comunicação de baixa latência e alta performance, essencial para operações que exigem validação em tempo real.

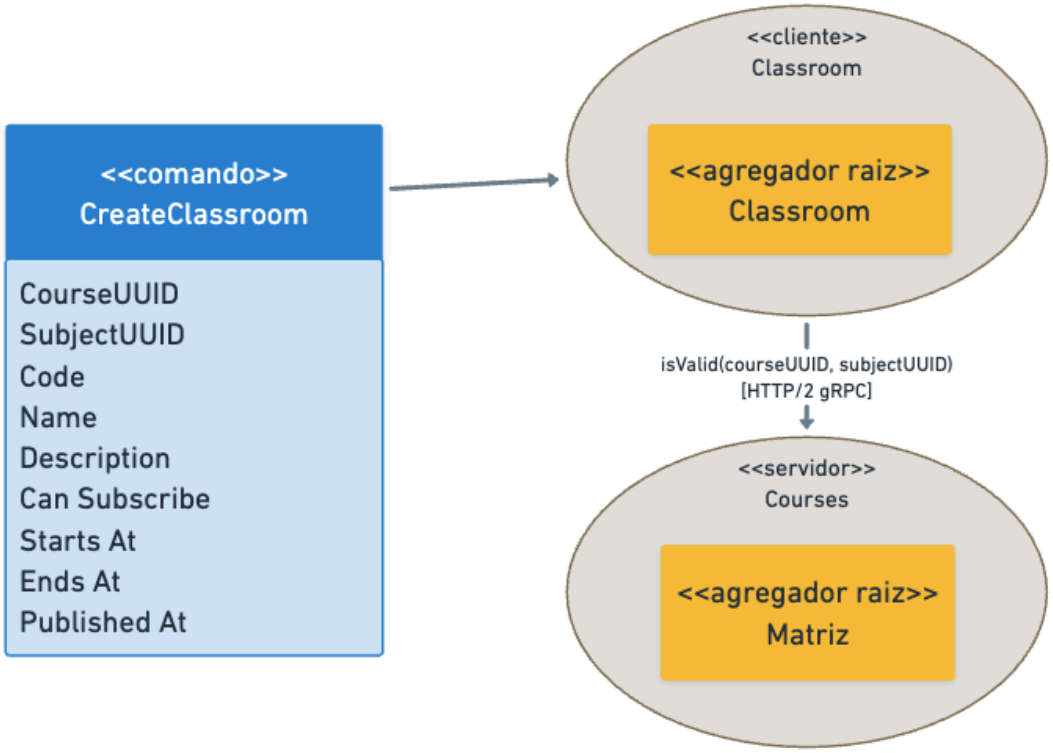

Figura 20. Create Classroom (sync - gRPC)

Fonte: Produzido pela autora

Para implementar a comunicação síncrona, conforme ilustrado na fig 20, primeiro se faz necessária a definição das interfaces gRPC para os diferentes microsserviços envolvidos. Isso é feito utilizando arquivos de esquema .proto que especificam os métodos que eles expõem. Depois é necessário utilizar o compilador *protoc* para gerar o código do cliente e servidor a partir do arquivo .proto. Isso cria stubs que serão usados para a comunicação entre os microsserviços. Com isso, é possível implementar a lógica do cliente que irá chamar o serviço gRPC para validar os dados, e por fim, a lógica do servidor que irá processar as solicitações vindas dos microsserviços clientes.

O gRPC oferece vantagens significativas em termos de latência de rede e suporte a comunicação bidirecional, mas também apresenta desafios, pois pode

exigir mais recursos de CPU e memória do que as abordagens baseadas em REST. Isso impacta a complexidade de manutenção devido a necessidade de monitorar e gerenciar recursos, para garantir que o sistema esteja sempre disponível e funcionando corretamente.

3.6.3 COMUNICAÇÃO ASSÍNCRONA COM RABBITMQ

RabbitMQ é um servidor de mensageria de código aberto que suporta vários protocolos de padrão aberto, incluindo AMQP, e oferece muitas opções para definir como as mensagens vão ser enviadas do produtor para um ou mais consumidores, além da capacidade de confirmar a entrega de mensagens e replicar mensagens (RABBITMQ, 2024).

Uma das funcionalidades implementadas no sistema é a deleção de matérias, no contexto de Courses. O objeto de valor *SubjectUUID*, que faz referência ao agregado *Subjects,* está presente no contexto de Classroom, pois para criar uma turma, é necessário identificar a matéria e o curso correspondentes. Para garantir a integridade dos dados e a consistência do sistema, sempre que uma matéria é deletada, é necessário remover, de forma assíncrona, todas as entidades que fazem referência a ela. Para isso, optamos por utilizar a estratégia de publicação e consumo de mensagens por meio do protocolo AMQP, implementado no RabbitMQ.

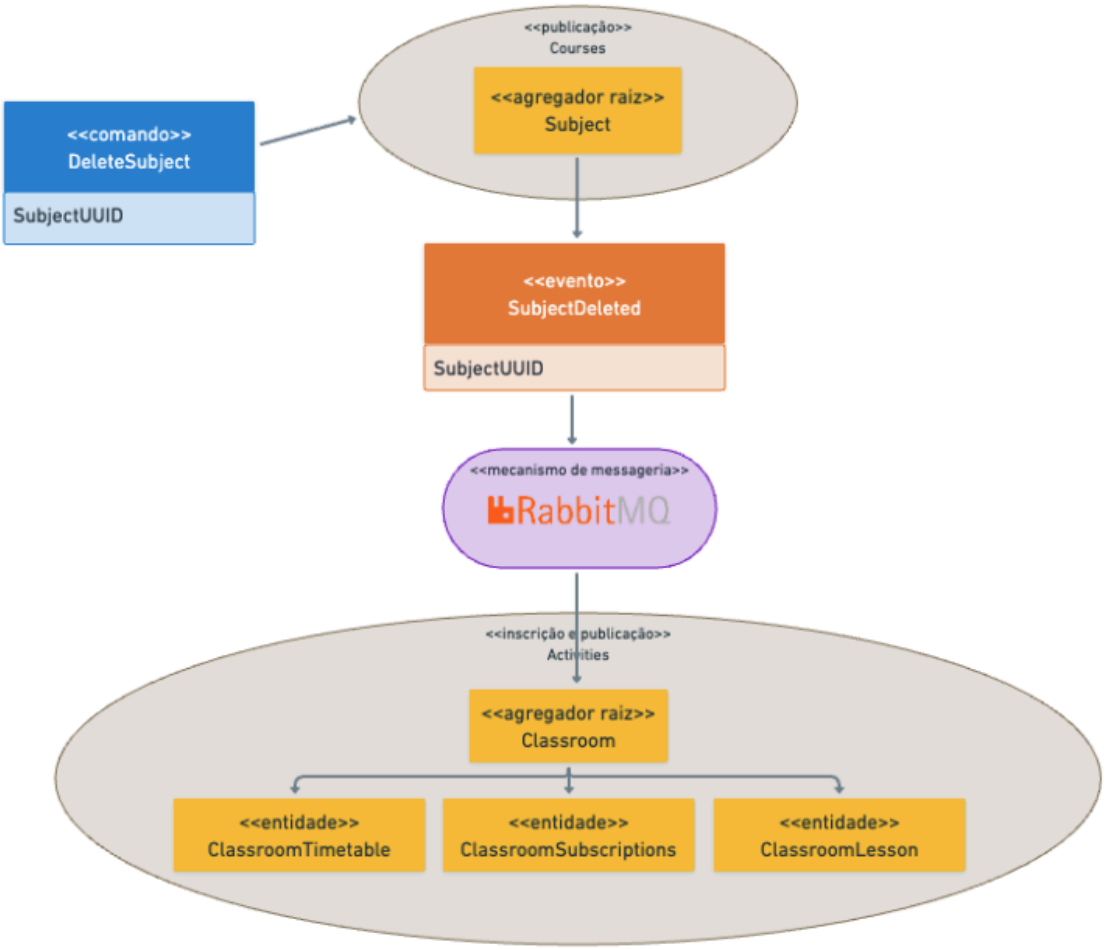

Figura 21. Delete Subject (async RabbitMQ)

Fonte: Produzido pela autora

Para implementar a comunicação assíncrona, conforme ilustrado na fig 21, primeiro se faz necessária a definição de uma exchange, responsável por receber mensagens dos produtores e enviar para os consumidores interessados conforme a configuração necessária. Depois, podemos adicionar filas que receberão as mensagens, implementar o envio das mensagens no produtor, a assinatura das mensagens nos consumidores e o comportamento desejado sempre que uma mensagem é consumida.

Apesar de oferecer vantagens como flexibilidade de roteamento, controle de tempo das mensagens, manuseio de falhas e simplicidade na implementação dos consumidores, manter um sistema de mensageria como RabbitMQ adiciona

complexidade de manutenção devido a necessidade de monitorar e gerenciar filas, para garantir que o sistema esteja sempre disponível e funcionando corretamente.

Este capítulo apresentou a proposta e desenvolvimento do MVP Sumé, suas principais características, definições e vantagens. No próximo capítulo, elencamos os principais pontos desta pesquisa, os desafios superados, a contribuição acadêmica alcançada com a aplicação dos conceitos e tecnologias apresentadas bem como sugestões de trabalhos futuros.

4. CONCLUSÃO

Ao longo deste trabalho, nossa atenção esteve voltada para a compreensão e implementação do domínio de aplicação no contexto educacional. Este processo incluiu a definição e o design dos domínios e microsserviços essenciais para a implementação de um MVP, bem como a identificação das estratégias de comunicação entre esses microsserviços. Também dedicamos esforços para criar um diagrama entidade-relacionamento dos dados para cada microsserviço, avaliando os tipos de dados que precisamos armazenar.

A complexidade de implementação e manutenção de um sistema de gestão de aprendizagem baseado em microsserviços é significativa, mas pode ser gerenciada com práticas adequadas de design, documentação e infraestrutura. Revisitando a pergunta de pesquisa elaborada na introdução deste TCC, constatamos que é possível desenvolver um SGA mínimo usando uma abordagem dirigida por domínio, garantindo comunicação entre os microsserviços e consistência de dados entre eles.

Os contextos delimitados Courses, Classroom e Activities são compostos de diversas entidades que possuem atributos e comportamentos específicos que precisam ser modelados, implementados e mantidos. A complexidade aumenta à medida que o número de entidades e suas interações cresce, exigindo um design cuidadoso para garantir a integridade e a consistência dos dados. O DDD ajuda a diminuir essa complexidade pois cada contexto delimitado facilita a separação de responsabilidades. Isso significa que mudanças em um contexto não afetam diretamente os outros.

A arquitetura de microsserviços oferece vantagens significativas em termos de modularidade, escalabilidade e independência de implantação. No entanto,

também introduz complexidades adicionais, como comunicação síncrona (via gRPC) e assíncrona (via RabbitMQ) entre os microsserviços, para garantir a entrega e o processamento correto das mensagens e a consistência dos dados entre diferentes microsserviços.

A adoção das estratégias apresentadas neste trabalho permite uma maior flexibilidade e adaptabilidade, essenciais para atender às necessidades dinâmicas do ambiente educacional.

4.1 RELAÇÃO DO TCC COM A GRADUAÇÃO

A elaboração deste TCC foi influenciada por diversas disciplinas cursadas ao longo da graduação. As disciplinas de Programação Orientada a Objetos e Programação Web, forneceram a base para a modelagem de entidades e a implementação de comportamentos específicos dentro dos microsserviços. A compreensão de conceitos como objetos, encapsulamento e acoplamento foi essencial para o design modular e reutilizável dos códigos do sistema. A Engenharia de Software abordou metodologias de desenvolvimento, práticas de design e padrões de arquitetura, que levaram à adoção de práticas adequadas de design e documentação. As disciplinas de Bancos de Dados forneceram fundamentos sobre modelagem de dados, SQL e sistemas de gerenciamento de banco de dados, essenciais para a criação do diagrama entidade-relacionamento (ER) dos dados e para a definição dos tipos de dados a serem armazenados. Já disciplinas de Redes de Computadores e Computação Distribuída proporcionaram conhecimentos sobre protocolos de comunicação, segurança, comunicação entre processos e consistência de dados, conceitos fundamentais para entender e implementar a comunicação síncrona (RPC) e assíncrona (AMQP) entre os microsserviços, garantindo a entrega e o processamento correto dos eventos de domínio. Por fim,

as disciplinas de Introdução ao TCC, TCC I e TCC II forneceram orientação metodológica e suporte para a elaboração do TCC, como suporte na estruturação do trabalho, definição da pergunta de pesquisa e organização das etapas de desenvolvimento, desde a definição do tema até a defesa final.

4.2 TRABALHOS FUTUROS

Para aprimorar ainda mais o Sistema de Gestão de Aprendizagem desenvolvido neste trabalho, propomos a criação de um middleware centralizado que gerencie as chamadas para os diferentes microsserviços. A utilização de middleware pode reduzir a complexidade da infraestrutura e melhorar a escalabilidade do sistema, permitindo que novos microsserviços sejam adicionados ou atualizados sem impactar diretamente os clientes. Além disso, o middleware pode incluir funcionalidades de balanceamento de carga e monitoramento, proporcionando uma visão centralizada do desempenho e da saúde dos microsserviços.

Para garantir a confiabilidade, desempenho e escalabilidade do sistema desenvolvido neste trabalho, propõe-se a realização de uma série de testes abrangentes como parte dos trabalhos futuros. Estes testes visam avaliar diversos aspectos do sistema, desde a funcionalidade de componentes individuais até o comportamento do sistema como um todo sob diferentes condições de carga.

Outro trabalho futuro importante é a criação de um microsserviço dedicado à autenticação e segurança. Um microsserviço de autenticação centralizado pode simplificar esse processo, gerenciando tokens de autenticação (como JWT) e implementando Single Sign-On (SSO) para facilitar a experiência do usuário. Este serviço garantiria que todas as requisições aos microsserviços fossem autenticadas e autorizadas de maneira consistente, melhorando a segurança geral do sistema.

Por fim, a criação de um frontend utilizando os microsserviços desenvolvidos neste trabalho para proporcionar uma interface para os usuários finais, como administradores, educadores, auxiliares e estudantes.

4.3 QUANDO OPTAR PELO SUMÉ

O desenvolvimento do SGA Sumé representa uma abordagem inovadora na criação de plataformas educacionais, combinando a flexibilidade dos microsserviços com os princípios do DDD e um foco especial na gestão dos dados. Esta arquitetura posiciona o Sumé como uma escolha superior em cenários que demandam alta escalabilidade e adaptabilidade.

Esse projeto visa facilitar também a implementação de análises preditivas do desempenho dos estudantes e educadores. Integrar sistemas de recomendação personalizados para conteúdos educacionais, facilitar a adoção de tecnologias de IA para tutoria automatizada e avaliação de aprendizado. A facilidade de integração de novas tecnologias permite que instituições mantenham-se na vanguarda das práticas educacionais, pois a plataforma é capaz de evoluir continuamente, incorporando as mais recentes inovações em tecnologia educacional e análise de dados.

5. REFERÊNCIAS

MOODLE. Moodle - Open-source learning platform. Disponível em: https://moodle.com/. Acesso em: 3 jun. 2024.

RELATÓRIOS INSIGHTS. Learning Management Systems (LMS) Market [2023-2030]. Disponível em: https://www.reportsinsights.com/industry-forecast/learning-management-systems-lms-market-statistical-analysis-673862. Acesso em: 3 jun. 2024.

gRPC. gRPC: um projeto de incubação da CNCF. 30 mar. 2024. Disponível em: https://grpc.io. Acesso em: 19 maio 2024.

RABBITMQ. RabbitMQ: One broker to queue them all. 2024. Disponível em: https://www.rabbitmq.com. Acesso em: 19 maio 2024.

KARTHIK, S.; KUMAR, A.; SMITH, J. Remote Procedure Call as a Managed System Service. arXiv, 14 abr. 2023. Disponível em: https://arxiv.org/pdf/2304.07349.pdf. Acesso em: 19 maio 2024.

DASHDEVS. Modular Architecture in Mobile Development. DashDevs, 2023. Disponível em: https://dashdevs.com/blog/modular-architecture-in-mobile-development/. Acesso em: 03 jun. 2024.

DE MORAES, Joelda Ferreira et al. Ambientes virtuais de aprendizagem: uma revisão integrativa acerca da sua relevância para o processo de ensino-aprendizagem. Contribuciones a las Ciencias Sociales, 2023. Disponível em: https://www.semanticscholar.org/paper/Ambientes-virtuais-de-aprendizagem%3A-uma-revis%C3%A3o-da-Moraes-J%C3%BAnior/26a37ece4f6418f6af502378e400c1ecb106e532 . Acesso em 08 Maio 2024.

INSTRUCTURE. Canvas Architecture. Salt Lake City, 2022. Disponível em: https://www.instructure.com/sites/default/files/file/2022-03/Canvas%20Architecture%20%282022%29.pdf. Acesso em: 3 jun. 2024.

ZABIEROWSKI, WOJCIECH. The Comparison of Microservice and Monolithic Architecture. ResearchGate, 2020. Disponível em: https://www.researchgate.net/publication/341956559_The_Comparison_of_Microservice_and_Monolithic_Architecture. Acesso em: 03 jun. 2024.

IBM. Choosing the Right Database for Microsservices, 2020. Disponível em: https://www.ibm.com/cloud/blog/choosing-the-right-databases-for-microservices. Acesso em 25 Nov. 2023.

NEWMAN, Sam; Migrando Sistemas Monolíticos para Microsserviços. 1. ed. Novatec, 2020.

RICHARDSON, C. Microservices Patterns. 1. ed. Manning Publications Co., 2019.

MARIGI, G.; VOCALE, M. Saga: The new era of transactions in a microservices architecture. Red Hat Summit. 2019. Disponível em https://pt.slideshare.net/slideshow/saga-transactions-msa-architecture/230585813. Acesso em: 17 maio 2024.

CAMBRIDGE ASSESSMENT INTERNATIONAL EDUCATION. Global Education Census Report. Cambridge, 2018. Disponível em: https://www.cambridgeinternational.org/Images/514611-global-education-census-survey-report.pdf. Acesso em: 3 jun. 2024.

TAIBI, Davide; LENARDUZZI, Valentina; PAHL, Claus. Processes, Motivations and Issues for Migrating to Microservices Architectures: An Empirical Investigation. IEEE Cloud Computing, 2017. Disponível em

https://www.researchgate.net/publication/319187656_Processes_Motivations_and_Issues_for_Migrating_to_Microservices_Architectures_An_Empirical_Investigation. Acesso em 25 Nov. 2023.

SHADIJA, Dharmendra; REZAI, Mo; HILL, Richard. Towards an Understanding of Microservices. 2017. Disponível em https://www.researchgate.net/publication/319952918_owards_an_Understanding_of_Microservices . Acesso em 25 Nov. 2023.

FOWLER, Susan. Microsserviços Prontos Para a Produção: Construindo Sistemas Padronizados em uma Organização de Engenharia de Software. 1. ed. Novatec Editora, 2017.

NEWMAN, Sam. Criando Microsserviços. Projetando sistemas com componentes menores e mais especializados. 2. ed. O'Reilly, 2015.

BURCKHARDT, S. Principles of eventual consistency. Microsoft Research, 2014.

VERNON, V. Implementing domain-driven design. Addison-Wesley, 2013.

JAISWAL, Vinay; AGRAWAL, Alka. Domain-Driven Design (DDD): Bridging the Gap between Object-Oriented Design and Ontology Engineering. International Journal of Information Technology and Computer Science, v. 5, n. 12, p. 41-49, 2013.

SAUDATE, Alexandre. REST: Construa API's inteligentes de maneira simples. Casa do código, 2013.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. Sistema de Banco de Dados. 3. ed. Editora Pearson, 2010.

TRIELOFF, C. et al. Advanced message queuing protocol specification. amq-spec. AMQP.org, v. 0.10, 2006.

EVANS, E. Domain-driven design: tackling complexity in the heart of software. Addison-Wesley Professional, 2003.

ALMEIDA, M. E. B. de. Educação à distância na internet: abordagens e contribuições dos ambientes digitais de aprendizagem. Educação e Pesquisa, São Paulo, v. 29, n. 2, p. 327-340, jul./dez. 2003.

APÊNDICE A – ARTIGO DO TRABALHO NO MODELO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO

Implementação de Microsserviços para um Sistema de Gestão de Aprendizagem

Mariany Ferreira da Silva, Patricia Della Mea Plentz

Departamento de Informática e Estatística, Universidade Federal de Santa Catarina,

Florianópolis, SC, Brasil

fsilvamariany@gmail.com, patricia.plentz@ufsc.br

Abstract. *This study presents the development of a Minimum Viable Product (MVP) for a Learning Management System (LMS) using Domain-Driven Design (DDD) and microservices architecture. The research employs an exploratory approach, combining a comprehensive literature review with practical application of acquired knowledge in system development. The primary objective is to create an efficient system with data consistency and effective communication between components, both synchronous and asynchronous. Results indicate that a domain-driven approach significantly reduces the complexity of LMS implementation by facilitating the separation of responsibilities through bounded contexts. Additionally, the microservices architecture offers substantial advantages in terms of modularity, scalability, and deployment independence. This research contributes to the field of information systems by providing new insights through the combined application of DDD and microservices architecture, addressing contemporary demands in software development. The study demonstrates how these methodologies can be effectively applied to create more robust and adaptable solutions in the realm of Learning Management Systems.*

Resumo. *Este estudo apresenta o desenvolvimento de um Produto Mínimo Viável (MVP) para um Sistema de Gestão de Aprendizagem (SGA) utilizando Design Orientado a Domínio (DDD) e arquitetura de microsserviços. A pesquisa emprega uma abordagem exploratória, combinando uma revisão abrangente da literatura com a aplicação prática dos conhecimentos adquiridos no desenvolvimento do sistema. O objetivo principal é criar um sistema eficiente com consistência de dados e comunicação eficaz entre os componentes, tanto síncrona quanto assíncrona. Os resultados indicam que uma abordagem orientada a domínio reduz significativamente a complexidade da implementação do SGA, facilitando a separação de responsabilidades por meio de contextos delimitados. Além disso, a arquitetura de microsserviços oferece vantagens substanciais em termos de modularidade, escalabilidade e independência na implantação. Esta pesquisa contribui para a área de sistemas de informação ao fornecer novos insights por meio da aplicação conjunta de DDD e arquitetura de microsserviços, abordando demandas contemporâneas no desenvolvimento de software. O estudo demonstra como essas metodologias podem ser aplicadas de forma eficaz para criar soluções mais robustas e adaptáveis no âmbito dos Sistemas de Gestão de Aprendizagem.*

1. Introdução

A transformação digital e os avanços tecnológicos têm impulsionado mudanças significativas no setor educacional, levando instituições de ensino a adaptar seus cursos

para modalidades a distância ou híbridas. Neste contexto, os Sistemas de Gestão de Aprendizagem (SGA) emergem como ferramentas essenciais para facilitar e enriquecer o processo educacional.

O mercado de SGA está em crescimento acelerado, com projeções indicando que atingirá mais de US$69 bilhões até 2030, apresentando uma taxa de crescimento anual composta (CAGR) de 19,2% entre 2023 e 2030. Este crescimento reflete a crescente adoção de tecnologias em salas de aula, com uma parcela significativa de alunos utilizando dispositivos como computadores, smartphones e tablets para aprendizagem.

Atualmente, existem diversos SGAs disponíveis no mercado, como Moodle, Canvas e Blackboard Learn, cada um com suas características e limitações. No entanto, a complexidade e a dinâmica do ambiente educacional demandam soluções tecnológicas que sejam não apenas eficientes e escaláveis, mas também flexíveis e adaptáveis às constantes mudanças nas necessidades de instituições, educadores e alunos.

Este trabalho propõe o desenvolvimento de um Produto Mínimo Viável (MVP) de um SGA denominado Sumé, utilizando uma abordagem dirigida por domínio (Domain-Driven Design - DDD) e arquitetura de microsserviços. O objetivo é criar um sistema que não apenas atenda às necessidades básicas de gestão de aprendizagem, mas também ofereça uma base sólida para futuras expansões e adaptações.

O projeto se concentra na definição de requisitos e características-chave para criação e gestão de cursos, turmas e atividades, implementando os domínios Courses, Classrooms e Activities como microsserviços independentes e desacoplados. Além disso, busca-se estruturar e modelar as bases de dados de cada microsserviço e implementar comunicação síncrona e assíncrona entre os domínios.

2. Metodologia

Neste estudo, propomos uma abordagem exploratória para investigar a arquitetura de microsserviços e suas implicações na gestão de dados em ambientes distribuídos, com foco em tecnologias emergentes aplicadas ao desenvolvimento de Sistemas de Gestão Acadêmica (SGAs). Inicialmente, será realizada uma pesquisa bibliográfica, visando construir um referencial teórico relevante sobre o tema. A seleção de tecnologias pertinentes permitirá a implementação prática dos microsserviços do SGA, servindo como um estudo de caso para análise dos desafios, eficácia e viabilidade dessa arquitetura e das estratégias de modelagem de dados. Durante o desenvolvimento do protótipo, todas as etapas, desafios e soluções são documentadas e analisadas, com o objetivo de identificar padrões e estratégias adotadas. Além disso, são discutidas as limitações enfrentadas ao longo da pesquisa, incluindo restrições de escopo e limitações de acesso a recursos.

3. Fundamentação Teórica

Sistemas de Gestão de Aprendizagem (SGAs), conhecidos como Learning Management Systems (LMS), exercem um papel importante de facilitação da educação a distância e híbrida. Esses sistemas são essenciais para suportar atividades educacionais, permitindo a interação entre alunos e educadores através de plataformas acessíveis em diversos

dispositivos. Além de seu papel na educação a distância, os SGAs complementam o ensino presencial, oferecendo funcionalidades como criação de cursos, planejamento de aulas e organização de materiais didáticos (De Moraes, 2023).

O Domain-Driven Design (DDD) é uma abordagem de design de software central para este trabalho, devido à sua eficácia em lidar com domínios de negócios complexos. O DDD promove a modelagem de um domínio rico, facilitando a definição clara de entidades, objetos de valor e repositórios. Essa abordagem é particularmente útil para a implementação de microsserviços, cada um com suas próprias regras de negócio. A comunicação eficaz entre esses microsserviços é garantida por meio de eventos de domínio, assegurando que o desenvolvimento do software esteja alinhado com as complexidades do domínio (Evans, 2003).

A arquitetura de microsserviços é destacada por sua capacidade de implantações independentes e escaláveis, permitindo que cada serviço atenda a uma única funcionalidade de negócio. Essa modularidade oferece vantagens como testes simplificados e maior agilidade no desenvolvimento. No entanto, a implementação de sistemas distribuídos e a modelagem de domínio apresentam complexidades que exigem um gerenciamento cuidadoso para manter a consistência e integração dos dados (Richardson C., 2019).

A gestão de dados distribuídos é um desafio crítico, com a literatura sugerindo o uso de bancos de dados separados para cada microsserviço. Isso garante autonomia e evita acoplamentos indesejados. A consistência eventual é adotada para assegurar que todas as consultas retornem valores atualizados, mesmo que inicialmente possam divergir. O padrão SAGA é utilizado para gerenciar transações distribuídas, garantindo a consistência dos dados através de compensações em caso de falhas (Richardson C., 2019).

Shadija e Rezai (2017) sugerem que a comunicação entre microsserviços é facilitada por protocolos como REST, RPC e AMQP, que permitem a interação entre sistemas distribuídos. A escolha do protocolo depende das necessidades específicas de cada microsserviço, equilibrando a necessidade de respostas imediatas com a flexibilidade proporcionada pela comunicação assíncrona.

O crescimento qualitativo contínuo é essencial para a escalabilidade e evolução do sistema. Isso envolve a introdução de novas funcionalidades, otimização de processos existentes e aumento da capacidade de lidar com tarefas complexas. A arquitetura de microsserviços permite que cada serviço seja escalado independentemente, aumentando a resiliência e disponibilidade do sistema. Além disso, a integração de diferentes tecnologias e frameworks otimiza o desempenho e facilita a experimentação e adoção de novas tecnologias (Fowler, 2017).

4. Implementação do MVP Sumé

O desenvolvimento do MVP Sumé, um Sistema de Gestão de Aprendizagem (SGA) baseado em microsserviços, foi projetado para oferecer uma solução modular e escalável. A arquitetura de microsserviços permite que cada componente do sistema opere de forma independente, facilitando a manutenção e a escalabilidade. A

implementação seguiu uma abordagem de Domain-Driven Design (DDD), que ajudou a definir contextos delimitados e modelos de domínio claros para cada funcionalidade do sistema.

4.1. Requisitos Funcionais e Não-Funcionais

Os requisitos funcionais do Sumé incluem a identificação de papéis de usuários, gerenciamento de cursos, matérias, matrizes curriculares, turmas e atividades. O sistema deve permitir a inscrição de usuários em cursos e turmas, além de manter logs detalhados das atividades. Os requisitos não-funcionais garantem que o sistema seja capaz de lidar com uma carga crescente de dados sem perda de desempenho, fornecendo respostas rápidas e alta disponibilidade. A API deve ser intuitiva e de fácil utilização, permitindo navegação eficiente.

4.2. Design e Arquitetura

O design do Sumé utiliza DDD para criar uma linguagem ubíqua e definir contextos delimitados, como os contextos de Cursos, Turmas e Atividades. Cada contexto foi implementado como um microsserviço independente, com operações CRUD (Create, Read, Update, Delete). A arquitetura modular permite atualizações e manutenção sem interrupção do sistema. Isso facilita a integração de novos serviços ou a modificação dos existentes sem impacto significativo nos demais. Os microsserviços foram desenvolvidos e gerenciados em repositórios separados no GitHub, permitindo um desenvolvimento e implantação independentes.

A modelagem de dados foi realizada utilizando diagramas Entidade-Relacionamento (ER) para representar as entidades e suas relações. A migração de dados e de schema foi adotada para lidar com mudanças nos dados e na estrutura do banco de dados, garantindo a integridade e a consistência dos dados ao longo do tempo. Essa abordagem permite uma evolução controlada do sistema, suportando novas funcionalidades de forma segura.

O PostgreSQL foi escolhido como sistema de gerenciamento de banco de dados, devido ao seu suporte robusto à integridade referencial e transações ACID, essenciais para a consistência dos dados. Cada microsserviço possui seu próprio banco de dados, permitindo uma gestão independente e eficiente dos dados.

Os eventos de domínio são fundamentais para a comunicação entre os microsserviços e para garantir a consistência e a integridade dos dados em operações que são causalmente relacionadas. Exemplos incluem eventos para deleção de inscrições e matérias, que são tratados de forma assíncrona para manter a integridade do sistema.

Para realizar comunicação síncrona foi utilizado o protocolo RPC, através da chamada de funções remotas. Para comunicação assíncrona foi utilizado AMQP através de eventos de domínio, utilizados para sincronizar operações entre microsserviços, garantindo a consistência dos dados. A escolha por SAGAs coreografadas permite que os microsserviços mantenham um alto grau de independência, evitando pontos únicos de falha e facilitando a manutenção do sistema.

A implementação de comunicação síncrona foi feita utilizando gRPC, que oferece alta performance e baixa latência, essencial para operações que exigem validação em tempo real. Para comunicação assíncrona, o RabbitMQ foi utilizado, proporcionando flexibilidade e controle no roteamento de mensagens. Essa abordagem permite que o sistema seja escalável e resiliente, mesmo em cenários de alta carga.

Cada microsserviço possui sua própria API REST, cuidadosamente documentada para garantir a clareza e a facilidade de uso. A documentação detalhada inclui descrições dos endpoints, métodos HTTP suportados, parâmetros de entrada, exemplos de requisições e respostas, além de códigos de status HTTP retornados. Essa abordagem facilita a integração e a comunicação entre os microsserviços, promovendo a interoperabilidade e a manutenção do sistema.

5. Conclusão

O presente trabalho abordou a implementação de um Sistema de Gestão de Aprendizagem (SGA) utilizando uma arquitetura de microsserviços e Domain-Driven Design (DDD). Foram definidos e projetados os domínios e microsserviços essenciais para um MVP, incluindo estratégias de comunicação entre eles e diagramas entidade-relacionamento para cada microsserviço.

A pesquisa demonstrou a viabilidade de desenvolver um SGA mínimo usando uma abordagem dirigida por domínio, garantindo comunicação eficiente entre os microsserviços e consistência de dados. Os contextos delimitados Courses, Classroom e Activities foram modelados, destacando a complexidade inerente ao sistema e a importância do DDD na separação de responsabilidades.

A arquitetura de microsserviços adotada ofereceu vantagens em termos de modularidade, escalabilidade e independência de implantação, embora tenha introduzido desafios na comunicação síncrona (via gRPC) e assíncrona (via RabbitMQ) entre os serviços. As estratégias apresentadas proporcionaram maior flexibilidade e adaptabilidade, essenciais no ambiente educacional dinâmico.

O trabalho foi influenciado por diversas disciplinas da graduação, como Programação Orientada a Objetos, Engenharia de Software, Bancos de Dados e Computação Distribuída, que forneceram a base teórica e prática necessária para o desenvolvimento do projeto.

References

Evans, E. (2003) "Domain-driven design: tackling complexity in the heart of software.", Addison-Wesley Professional.

Richardson, C. Microservices Patterns. 1. ed. Manning Publications Co., 2019.

Shadija, Dharmendra; Rezai, Mo; HILL, Richard. (2017) "Towards an Understanding of Microservices", https://www.researchgate.net/publication/319952918_owards_an_Understanding_of_Microservices . Novembro 2023.

De Moraes, Joelda Ferreira et al. (2023) "Ambientes virtuais de aprendizagem: uma revisão integrativa acerca da sua relevância para o processo de ensino-aprendizagem." Contribuciones a las Ciencias Sociales, https://www.semanticscholar.org/paper/Ambientes-virtuais-de-aprendizagem%3A-uma-revis%C3%A3o-da-Moraes-J%C3%BAnior/26a37ece4f6418f6af502378e400c1ecb106e532, Maio 2024.

Fowler, Susan (2017) "Microsserviços Prontos Para a Produção: Construindo Sistemas Padronizados em uma Organização de Engenharia de Software". 1. ed. Novatec Editora.

APÊNDICE B – CÓDIGO DESENVOLVIDO NO TRABALHO

No desenvolvimento deste trabalho de conclusão de curso, foram utilizados repositórios de códigos separados para cada microsserviço desenvolvido. Os códigos estão disponíveis nos links https://github.com/marianysilva/microservice-course/tree/dev , https://github.com/marianysilva/microservice-classroom/tree/dev e https://github.com/marianysilva/microservice-activity/tree/dev. Esses repositórios são gerenciados utilizando o GitHub, que é uma plataforma de hospedagem de código-fonte e arquivos com controle de versão usando o Git.