Mural Unb de Socialização Artística - MUSA

Documento de Arquitetura

Versão 1.0

Histórico de Revisão

Data	Versão	Descrição	Autor(es)
31/10	1.0	Elaboração e preenchimento inicial do documento	Kelyton, Pedro Henrique, Natan

Autores:

Matrícula	Nome	Descrição do papel assumido na equipe	% de contribuição ao trabalho (*)[1]
241011582	Pedro Henrique Monteiro Nascimento	Developer	20
241012033	Kelyton De Lucas Moraes Santos	Developer	16
241011537	Natan José França	Lidér/Scrum Master	15
241011635	Thiago Alencar De Oliveira	Developer	7
241011116	Guilherme Lorenzi Ventura	Developer	7
232021795	Laura Ester Fernandes Silva Goulart	Developer	7

[1] (*) – para cada integrante da equipe, considere sua participação tanto no Documento de Arquitetura, quando nos demais documentos já entregues pela equipe (Visão do produto e do projeto; Declaração de escopo) e atribua um, percentual. A soma dos percentuais de todos os integrantes deve fechar em 100%)

241011967	Leticia Vitoria Gomes	Developer	7
241011546	Pedro Augusto Moretti Moreira	Developer	7
241012015	Bryan Martinez Gutierrez Leite	Developer	7
241011930	Gustavo Rodrigues De Noronha	Developer	7

Sumário

1 INTRODUÇÃO

1.1 Propósito

Este documento descreve a arquitetura do sistema sendo desenvolvido pelo grupo, na disciplina de MDS – Métodos de Desenvolvimento de Software – edição do segundo semestre de 2025, para o sistema MUSA (Mural Unb de Socialização Artística), a fim de fornecer uma visão abrangente do sistema para desenvolvedores, testadores e demais interessados em aspectos relacionados às tecnologias a serem usadas no desenvolvimento.

1.2 Escopo

O detalhamento do escopo se encontra no documento Lean inception juntamente com o documento de Visão do produto e do projeto. Porém, em linhas gerais o escopo do produto compreende o desenvolvimento de uma rede social acadêmica para centralizar a divulgação de atividades culturais, artísticas e empreendedoras da comunidade UnB, por meio de um feed de posts, uma central de eventos e um mural de serviços.

2 REPRESENTAÇÃO ARQUITETURAL

Dessa forma, aqui será apresentado a representação arquitetural do sistema.

2.1 Definições

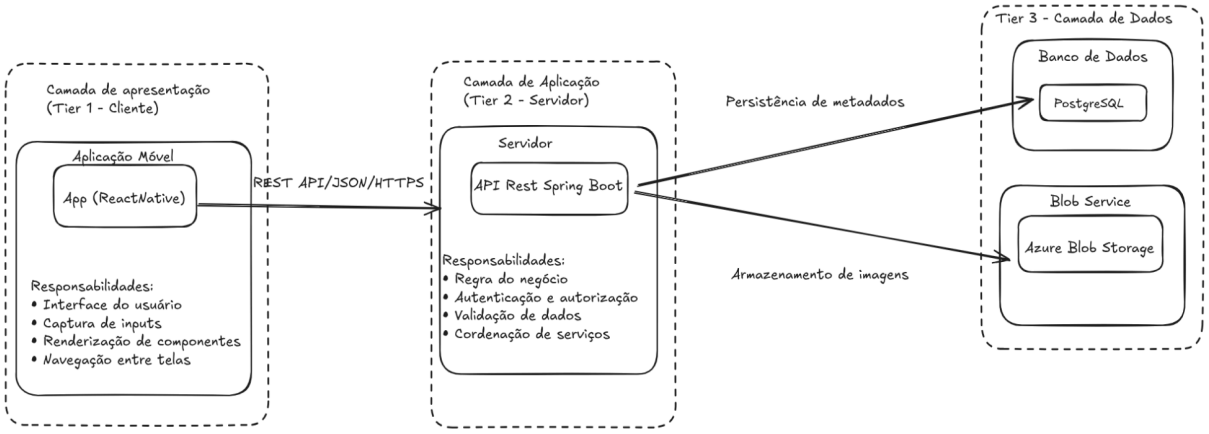
O sistema seguirá uma arquitetura Cliente Servidor em três camadas, com separação entre apresentação, lógica de negócio e persistência de dados.

2.2 Justificativa

Foi escolhida essa arquitetura por conta das necessidades específicas que a aplicação propõe como solução: Acesso universal por professores, alunos, organizações da UnB, técnicos em primeiro momento; Gestão de conteúdo artístico e acadêmico; Armazenamento de imagem.

2.3 Detalhamento

Figura 1 - Diagrama de arquitetura de alto nível do sistema



Fonte: Os autores (2025)

2.4 METAS E RESTRIÇÕES ARQUITETURAIS

Dessa forma, aqui será apresentado as metas e restrições arquiteturais.

2.4.1 O sistema deve utilizar cache local para reduzir o número de requisições ao servidor.

O uso do cache (no caso do produto, *AsyncStorage*) permite que informações acessadas e recebidas da camada de aplicação sejam exibidas sem conexão e acessadas sem chamadas adicionais.

2.4.2 Tratar estados de erros e ausência de conexão de forma visual e informativa

O devido tratamento de erros e a divisão entre erros do cliente (*ClientError*) e erros internos (*Error*) permite que o usuário saiba quando há falhas de redes ou indisponibilidade do servidor. Além disso, permite uma divisão clara de erros sensíveis à aplicação e erros para visualização do cliente.

2.4.3 Manter a segurança básica de autenticação e armazenamento de dados sensíveis

É essencial a proteção de dados dos usuários, mesmo utilizando práticas simples como criptografia de tokens e HTTPS. O sistema atual não possui camadas mais avançadas e abstratas de segurança.

2.4.4 Garantir que o tempo de resposta médio das requisições não ultrapasse 2 segundos em 90% dos casos

O sistema deve garantir que o tempo de resposta médio das requisições não ultrapasse dois segundos em noventa por cento dos casos. A restrição arquitetural do atual sistema consiste no limite do número de requisições simultâneas pela camada de serviço, e pelo uso de cache. Entretanto, a estabilidade depende da conexão do usuário e da carga do servidor.

2.4.5 O sistema deve seguir o padrão “Airbnb” para codificação e commits

A utilização do guia de commits do estilo Airbnb para projetos JavaScript e TypeScript, além do Prettier para formatação automática, assegura que o código mantenha um padrão uniforme e previsível durante todo o ciclo de desenvolvimento. No sistema atual, o padrão já foi integrado ao ambiente do projeto, sendo executado manualmente antes dos commits. A meta é a automação desse fluxo e treinamento dos desenvolvedores.

2.5 BACKLOG DO PRODUTO (ESCOPO DO PRODUTO)

O sistema MUSA (Mural Unb de Socialização Artística) está sendo desenvolvido para atuar como uma plataforma centralizada de socialização e exposição de trabalhos artísticos e acadêmicos. Nesse sentido, o funcionamento geral do sistema, refletido no backlog dinâmico mantido na ferramenta Zenhub, compreende a gestão de usuários, incluindo criação de contas, perfis e autenticação. Além disso, a plataforma permitirá interações sociais, como a criação e deleção de postagens, a capacidade de seguir outras contas, e também a gestão de eventos e a comercialização de produtos ou serviços.

A definição do estilo arquitetural de três camadas foi uma decisão técnica diretamente influenciada tanto por esses requisitos quanto pela experiência pregressa da equipe. Com efeito, a escolha do Spring Boot para a API Rest do servidor se justifica pela presença de dois desenvolvedores com experiência de mercado nesta tecnologia, enquanto a adoção do *React Native* para o aplicativo móvel se alinha à competência de outros dois integrantes com experiência em desenvolvimento mobile. Adicionalmente, a equipe optou pelo uso de *Docker*

com scripts de *entrypoint* dedicados para mitigar os desafios de configuração de ambiente. Tal abordagem padroniza o ambiente de desenvolvimento mobile para todos os membros, reduzindo os impedimentos técnicos para desenvolvedores sem experiência prévia em *React Native* e permitindo que a equipe foque na entrega de valor.

2.6 VISÃO LÓGICA

O sistema é subdividido logicamente seguindo a arquitetura de três camadas, conforme detalhado no diagrama de representação arquitetural. A primeira subdivisão é a Camada de Apresentação (Tier 1), que consiste na interface com o usuário. Este componente, um aplicativo móvel desenvolvido em React Native , tem seus pacotes de código focados em responsabilidades como a renderização de componentes , a captura de inputs e a navegação entre telas.

A segunda subdivisão é a Camada de Aplicação (Tier 2), que centraliza a lógica de negócios do sistema. Este servidor , uma API Rest em Spring Boot , organiza-se internamente em pacotes de código que executam as regras de negócio, a autenticação e autorização, a validação de dados e a coordenação de serviços. A comunicação entre a camada de apresentação e a de aplicação é realizada exclusivamente por meio da interface REST API, utilizando os protocolos JSON e HTTPS.

Por fim, a terceira subdivisão lógica aborda a Comunicação com o Banco de Dados (Tier 3). Esta responsabilidade é totalmente encapsulada pela Camada de Aplicação. O servidor Spring Boot é o único componente que se comunica com as fontes de dados, realizando a persistência de metadados no banco PostgreSQL e o armazenamento de imagens no Azure Blob Storage.

2.7 VISÃO DE DADOS (MER)

Abaixo, descreve-se as tabelas e seus atributos, relações e cardinalidades do modelo entidade relacionamento criada para o app MUSA.

2.7.1 Entidades e Atributos

Quadro 1 - Entidades e Atributos

Entidade	Descrição	Atributos principais
profiles	Representa usuários e organizações.	id, username, email, password, type, followers_count, following_count, created_at
posts	Publicações feitas por perfis.	id, profile_id, description, img_link, likes_count, created_at
events	Eventos criados por perfis organizacionais.	id, profile_id, title, event_time, description, location, created_at
follows	Representa o relacionamento de “seguir” entre dois perfis.	follower_profile_id, following_profile_id, created_at
likes	Representa curtidas feitas por perfis em posts.	user_id, post_id, created_at
shareds	Compartilhamentos de posts entre perfis.	id, profile_sharing_id, profile_shared_id, post_id, created_at
memberships	Relação entre perfis e organizações.	id, profile_id, organization_id, role, created_at
event_participants	Perfis que confirmaram presença em eventos.	profile_id, event_id, created_at
artist_products	Produtos artísticos vendidos por perfis.	id, profile_id, title, price, img_link, created_at

Fonte: Os autores (2025)

2.7.2 Relação entre tabelas

Quadro 2 - Relação entre tabelas

Tipo	Cardinalidade	Descrição
profiles → posts	1:N	Um perfil pode criar vários posts; cada post pertence a um único perfil.
profiles → events	1:N	Um perfil pode criar vários eventos; cada evento pertence a um perfil.
profiles → follows	N:M	Um perfil pode seguir vários outros perfis, e ser seguido por vários.
profiles → likes → posts	N:M	Um perfil pode curtir vários posts, e cada post pode ser curtido por vários perfis.
profiles → shareds → posts	N:M	Um perfil pode compartilhar vários posts; um post pode ser compartilhado por vários perfis.
profiles → memberships → profiles (organization)	N:M	Um perfil pode participar de várias organizações; uma organização pode ter vários membros.
profiles → event_participants → events	N:M	Um perfil pode participar de vários eventos; um evento pode ter vários participantes.
profiles → artist_products	1:N	Um perfil (artista) pode ter vários produtos; cada produto pertence a um artista.

Fonte: Os autores (2025)

2.8 VISÃO DE IMPLANTAÇÃO

O software será implantado em uma infraestrutura baseada em nuvem, utilizando serviços da plataforma RailWay. Essa opção foi escolhida por oferecer facilidade de configuração, integração com o repositório do GitHub e devido ao custo reduzido, que em primeiro momento é grátis.

A aplicação backend, desenvolvida em java com o framework Spring Boot, será hospedada no Railway. A ferramenta realiza a criação automática do ambiente de execução e fornece uma URL pública para consumo da API pelo frontend.

Para o armazenamento de dados, será utilizado o banco de dados relacional PostgreSQL hospedado no Railway e o Azure Blob Service para armazenamento em massa de imagens. O frontend, desenvolvido em React Native, será distribuído através de builds gerados para Android.

2.9 RESTRIÇÕES ADICIONAIS

Uma restrição negocial fundamental do projeto é que a aplicação não possui fins comerciais, o que exclui do escopo a necessidade de integração com quaisquer APIs de pagamento. Por outro lado, as funcionalidades de notificação e compartilhamento, que dependem de um sistema de mensageria e serviços de push, são identificadas como pontos de alta complexidade. A implementação dessas features exigirá um tempo de desenvolvimento dedicado, sendo um fator de risco a ser gerenciado no cronograma.

3 BIBLIOGRAFIA

AIRBNB. JavaScript Style Guide. Disponível em: http://github.com/airbnb/javascript. Acesso em: 31 out. 2025.

MOZILLA DEVELOPER NETWORK. Documentação oficial sobre REST e HTTP. Disponível em: https://developer.mozilla.org/. Acesso em: 31 out. 2025.