Descrição Geral da arquitetura.

1. Visão Geral da Arquitetura

1.1 Aplicação Web (Front-end)

- **Views (telas de cadastro e consulta):** Nós desenvolvemos telas para cadastro de alunos, professores e disciplinas, além de interfaces para renovação de matrícula, visualização de boletim e login. Cada view foi construída usando componentes reutilizáveis, garantindo consistência de layout e fácil manutenção.
- **Services (cliente HTTP/REST):** Definimos módulos em JavaScript, Dart e TypeScript que realizavam chamadas HTTP em formato JSON ao back-end. Esses serviços tratavam detalhes de formatação de payload, tratamento de erros, inclusão de token de autenticação nos headers e reconexão automática em caso de falhas de rede.

1.2 Back-end

1.2.1 Controllers (camada de entrada)

- **Rota de professor:** Nós expusmos endpoints para criação, leitura, atualização e remoção de dados de professores, validando dados de entrada e centralizando o roteamento.
- **Rota de matrícula:** Implementamos endpoints para iniciar e renovar matrículas, garantindo que o status de cada aluno fosse atualizado corretamente e liberando disciplinas conforme pré-requisitos.
- **Rota de lançamento de nota:** Criamos endpoints que permitiam ao professor lançar notas de avaliações, convertendo valores numéricos em conceitos e armazenando histórico de lançamentos.
- **Rota de boletim:** Desenvolvemos endpoints para geração e consulta de boletins, agregando notas por disciplina e calculando médias e conceitos finais.
- **Rota de disciplina e de aluno:** Monopolizamos rotas de consulta e gerenciamento de disciplinas e informações gerais dos alunos, com filtros avançados e paginação.

1.2.2 Lógica de Negócio

- **Serviço de cadastro de professor, aluno e disciplinas:** Nosso serviço garantia regras de negócio, como verificação de duplicidade, formatos de documento e conexão com o repositório.
- **Geração de boletim:** Criamos processos que coletavam notas via mensageria, calculavam médias e formatavam relatórios em PDF prontos para download.
- **Serviço de login:** Definimos o fluxo de autenticação usando JWT, incluindo emissão de tokens de acesso e refresh, e controle de tentativas de login.
- **Serviço de e-mail:** Integramos com SMTP para envio de notificações, lembretes de matrícula e alertas de resultado, formatando templates HTML personalizados.

1.2.3 Model / Repositories

- **Modelos de dados:** Nós modelamos entidades de Notas, Professores, Disciplinas, Alunos e Logs, mapeando atributos às colunas do banco.
- **Repositórios:** Implementamos abstrações que isolavam o ORM/ODM (Sequelize, TypeORM, Mongoose, Hibernate ou Prisma), permitindo trocar a tecnologia de persistência sem alterar a lógica.

1.2.4 Tecnologias e Ferramentas

- **Ambiente de execução:** Adotamos Node.js para flexibilidade, Aqueduct para algumas APIs em Dart e PHP em sistemas legados.
- **ORM/ODM:** Utilizamos Sequelize e TypeORM para bancos relacionais, Mongoose para MongoDB, e avaliamos Hibernate e Prisma onde adequado.

1.3 Filas e Serviços Auxiliares

- **RabbitMQ:** Decidimos usar uma fila para desacoplar operações síncronas do front-end de tarefas mais lentas (envio de e-mail, geração de boletins), melhorando a resiliência e escalabilidade.
- **Serviço de Autenticação externo:** Integramos um provedor externo para centralizar login único e autenticação multifator.
- **Serviço de E-mail (SMTP):** Configuramos um servidor SMTP dedicado para envio confiável das mensagens.
- **Serviço de Log (centralizado):** Agrupamos logs de aplicação, auditoria e acessos em um serviço central, permitindo busca e monitoramento unificado.

1.4 Databases

- **App Database:** Escolhemos MySQL, MongoDB ou PostgreSQL conforme o caso de uso; mantivemos tabelas de alunos, professores, notas e disciplinas organizadas por esquemas.
- **Log Database:** Criamos um banco separado para armazenar registros de aplicação, envios de e-mail e autenticações, otimizando consultas analíticas.
- **E-Mail Database:** Mantivemos histórico completo de e-mails enviados, recebidos e respostas, permitindo rastreamento e retriagem.
- **Auth Database:** Registramos credenciais, tokens, atividades e tentativas de login em tabelas independentes.

1.5 Fluxo de Comunicação

- O front-end consumia APIs REST e enviava dados em JSON.
- O back-end gravava e lia dados diretamente do banco via drivers nativos ou via ORM.
- Para cada e-mail ou geração de boletim, postávamos mensagens no RabbitMQ, que eram consumidas e processadas por workers dedicados.

2. Justificativas Técnicas das Decisões

2.1 Cliente-Servidor

- Decidimos separar claramente responsabilidades de apresentação (cliente) e processamento (servidor), permitindo deploys independentes, isolamento de falhas e dimensionamento sob demanda.

2.2 Arquitetura em Camadas

- Organizamos o código em Controllers, Lógica de Negócio e Model/Repository para facilitar testes unitários, validação de contratos e reuso de componentes.

2.3 Microsserviços

- Identificamos que autenticação, envio de e-mails e geração de relatórios tinham SLAs e ciclos de vida distintos, então os extraímos como microsserviços para escalabilidade independente, atualizações isoladas e maior resiliência em caso de falhas pontuais.