UNIVERSIDADE CATÓLICA DO SALVADOR
CURSO DE BACHARELADO EM ENGENHARIA DE SOFTWARE
ARQUITETURA DE SOFTWARE

ALISSON DE OLIVEIRA DOS SANTOS
DAVI DOS SANTOS RIBEIRO
LARISSA DA SILVA SANTANA
MARIA CLARA FERREIRA MENDES
MARIA FERNANDA PEREIRA BARCANTE COSTA
MURILO MORENO DOS SANTOS

SISTEMA DE CONTROLE DE ESPAÇOS ACADÊMICOS - COOLCODE

SALVADOR
2025

ALISSON DE OLIVEIRA DOS SANTOS
DAVI DOS SANTOS RIBEIRO
LARISSA DA SILVA SANTANA
MARIA CLARA FERREIRA MENDES
MARIA FERNANDA PEREIRA BARCANTE COSTA
MURILO MORENO DOS SANTOS

SISTEMA DE CONTROLE DE ESPAÇOS ACADÊMICOS - COOLCODE

Projeto a ser apresentado como
crédito parcial da disciplina
Arquitetura de software, com a
orientação do professor
Fernando César.

SALVADOR
2025

2a Atividade Avaliativa da 1a Unidade

Informe qual o cenário trabalhado

O sistema de controle de salas de aula e auditórios, denominado como Espaços Acadêmicos (EA), tem como objetivo gerenciar de forma eficiente as operações de reserva de espaços acadêmicos. A solução busca otimizar o uso dos espaços disponíveis, proporcionando uma experiência mais prática e organizada tanto para os administradores quanto para os professores.

O nosso projeto foi estruturado em uma arquitetura composta por duas camadas principais:

- **Aplicação Front-End**, responsável por proporcionar a interface de interação com os usuários;

- **Aplicação Back-End**, encarregada por todo o processamento das regras de negócio e persistência dos dados.

As duas aplicações se comunicam por meio de requisições e respostas HTTP. Ademais, toda a lógica de negócio está no back-end, garantindo segurança e integridade dos dados, enquanto o front-end busca e exibe as informações relevantes para o usuário final.

Diante disso, os usuários finais da aplicação são:

- **Professores**, os quais podem visualizar espaços disponíveis, realizar reservas simplificada de salas e auditórios, gerenciar as próprias reservas e confirmar a utilização dos espaços

- **Administradores**, os quais têm acesso ao gerenciamento completo dos espaços, cadastro de professores autorizados, visualização e aprovação de reservas, além de relatórios de uso do sistema.

Entre os recursos oferecidos pelo sistema, destacam-se:

O Gerenciamento de espaços: cadastro e gerenciamento facilitado de salas, auditórios e laboratórios; Reservas simplificadas: processo de reserva com visualização de disponibilidade; Confirmação de uso: sistema de validação que evita espaços reservados sem utilização.

Descreva, de forma resumida, o que cada componente da equipe fez no projeto

Alisson de Oliveira dos Santos

- Desenvolveu as funcionalidades da aplicação Front-End.
- Estruturou o escopo do banco de dados para definição do modelo relacional.

Davi dos Santos Ribeiro

- Definiu a arquitetura das aplicações Back-End e Front-End.
- Modelou o banco de dados relacional utilizado no projeto.
- Desenvolveu a interface do usuário (UI) no Front-End.
- Implementou a integração entre Front-End e Back-End via requisições HTTP.
- Estruturou o projeto com foco em escalabilidade e usabilidade.

Larissa da Silva Santana

- Produziu a documentação geral do projeto.
- Elaborou os diagramas C4 (Contexto, Container, Componente e Código)
- Elaborou os diagramas de classe.
- Commit da documentação

Maria Fernanda Pereira Barcante Costa

- Desenvolveu classes de repositório e pontos de entrada da API (endpoints REST).

Maria Clara Ferreira Mendes

- Criou instâncias EC2 na AWS para hospedar o projeto.
- Automatizou a infraestrutura com Terraform, aplicando o conceito de Infrastructure as Code (IaC).
- Gerou imagens Docker e publicou no Amazon ECR (Elastic Container Registry) para o projeto.

Murilo Moreno dos Santos

- Elaborou o esboço inicial do diagrama de classes para guiar a implementação das entidades.
- Implementou entidades do domínio conforme o modelo definido.
- Desenvolveu classes de repositório e pontos de entrada da API (endpoints REST).
- Elaborou a documentação do Back-end.

Link do repositório Git UCSAL do projeto Front-End
https://github.com/ucsal/des-front-end-coolcode_front

Descreva qual tecnologia usou na construção do projeto Front-End

O projeto foi desenvolvido utilizando React.js, o qual é uma biblioteca JavaScript declarativa para construção de interfaces de usuário. Em nosso projeto, o React permite a criação de componentes reutilizáveis, o que facilita a manutenção, a escalabilidade e a organização da aplicação.

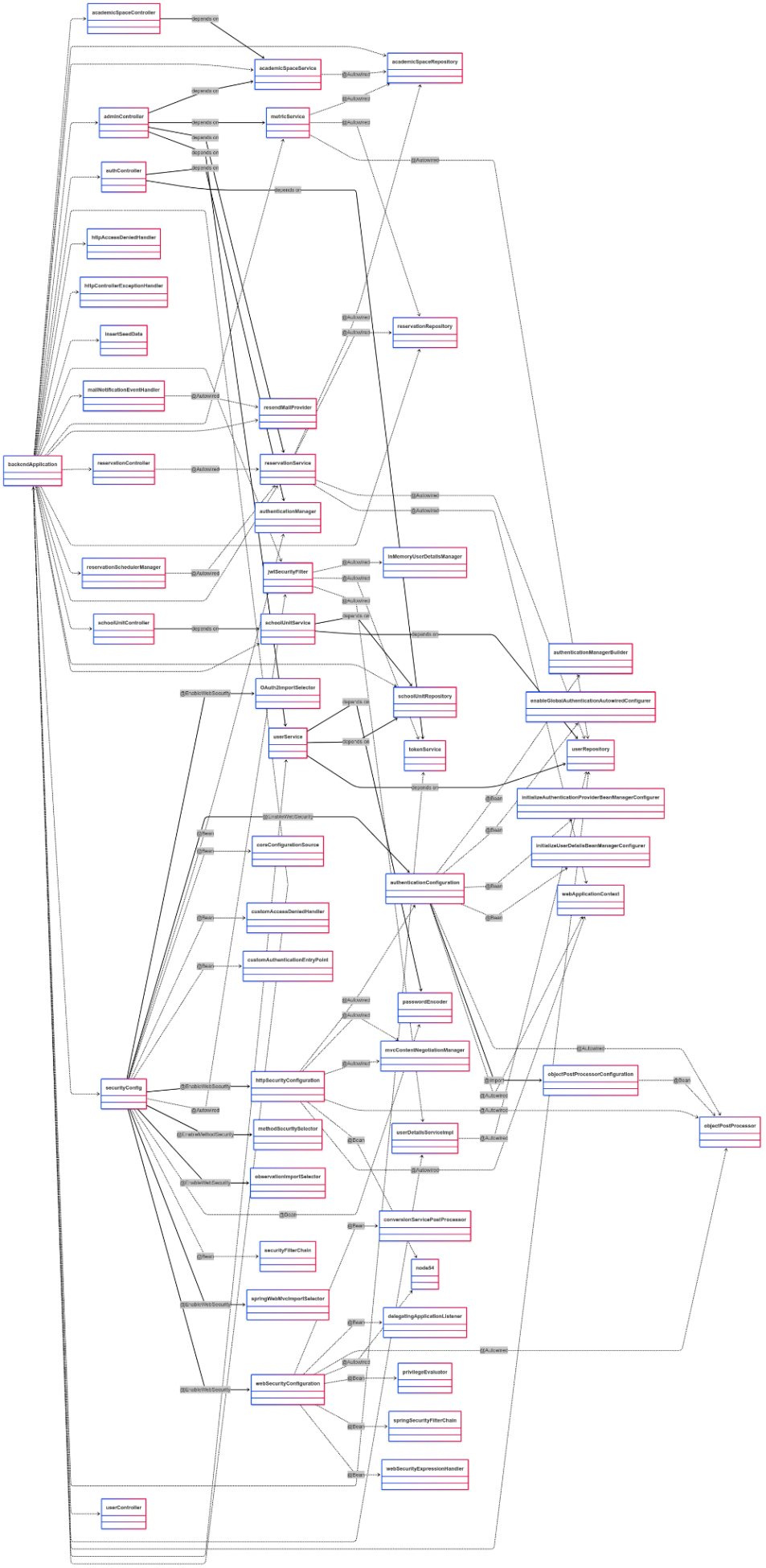
Entre as principais tecnologias e ferramentas utilizadas no Front-end do projeto:

React.js: Biblioteca principal para construção da interface.
React Router DOM: Gerenciamento de rotas (para realizar a navegação entre páginas).
Axios: Para fazer requisições HTTP ao Back-End.
Tailwind CSS: Utilizado para estilizar e tornar os componentes responsivos. Radix UI: Biblioteca de componentes de interface de usuário (UI).
JavaScript (ES6+): Foi a linguagem base utilizada com recursos como arrow functions, destructuring e async/await.
HTML e CSS (nos arquivos .jsx): aplicadas para estruturar e estilizar os elementos da interface.

Link do repositório Git UCSAL do projeto Back-End (JAVA + SPRING)

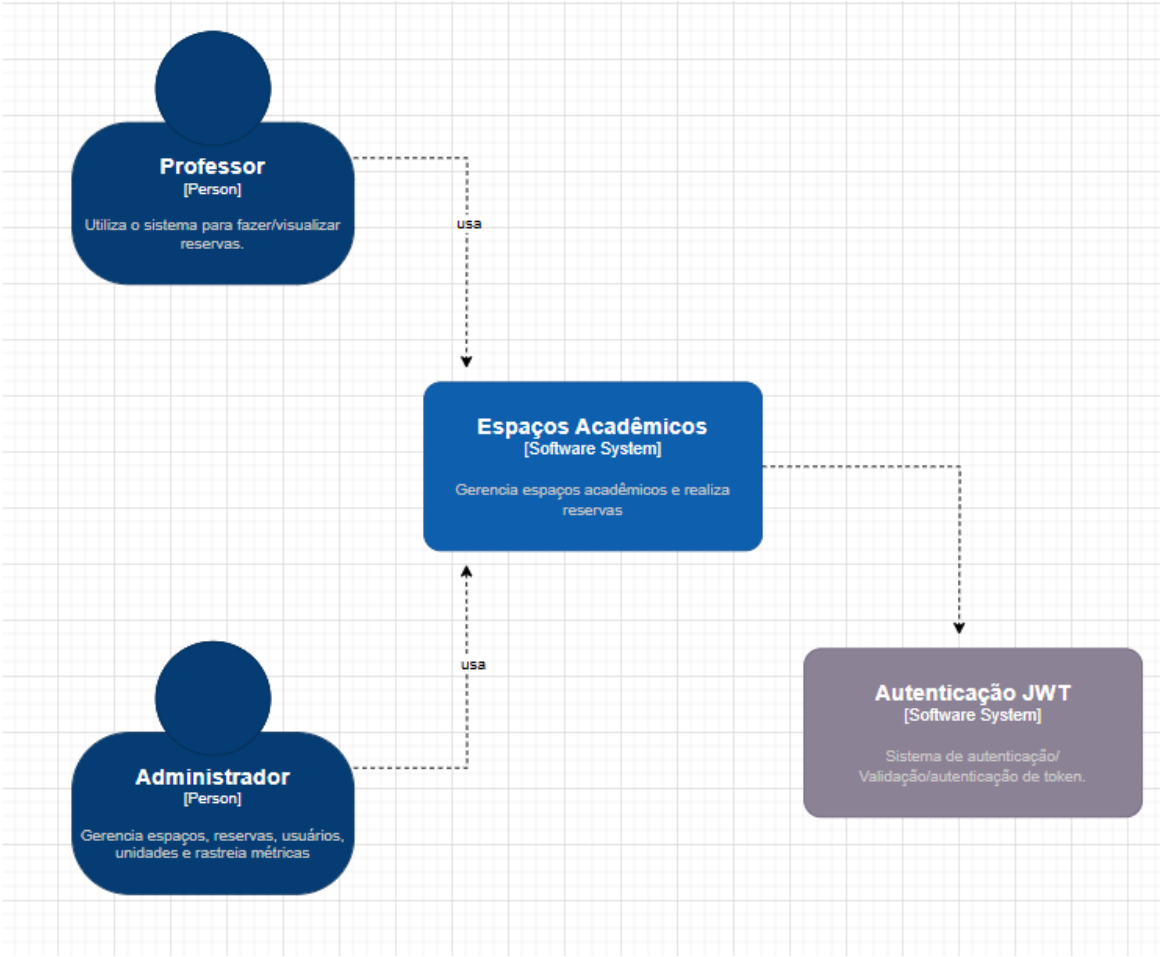
https://github.com/ucsal/d-back-end-coolcode_back

Entrega do Diagrama de Classe Atualizado (apenas se for o caso) - formato PNG
Faça upload de até 10 arquivos aceitos. O tamanho máximo é de 10 MB por item.

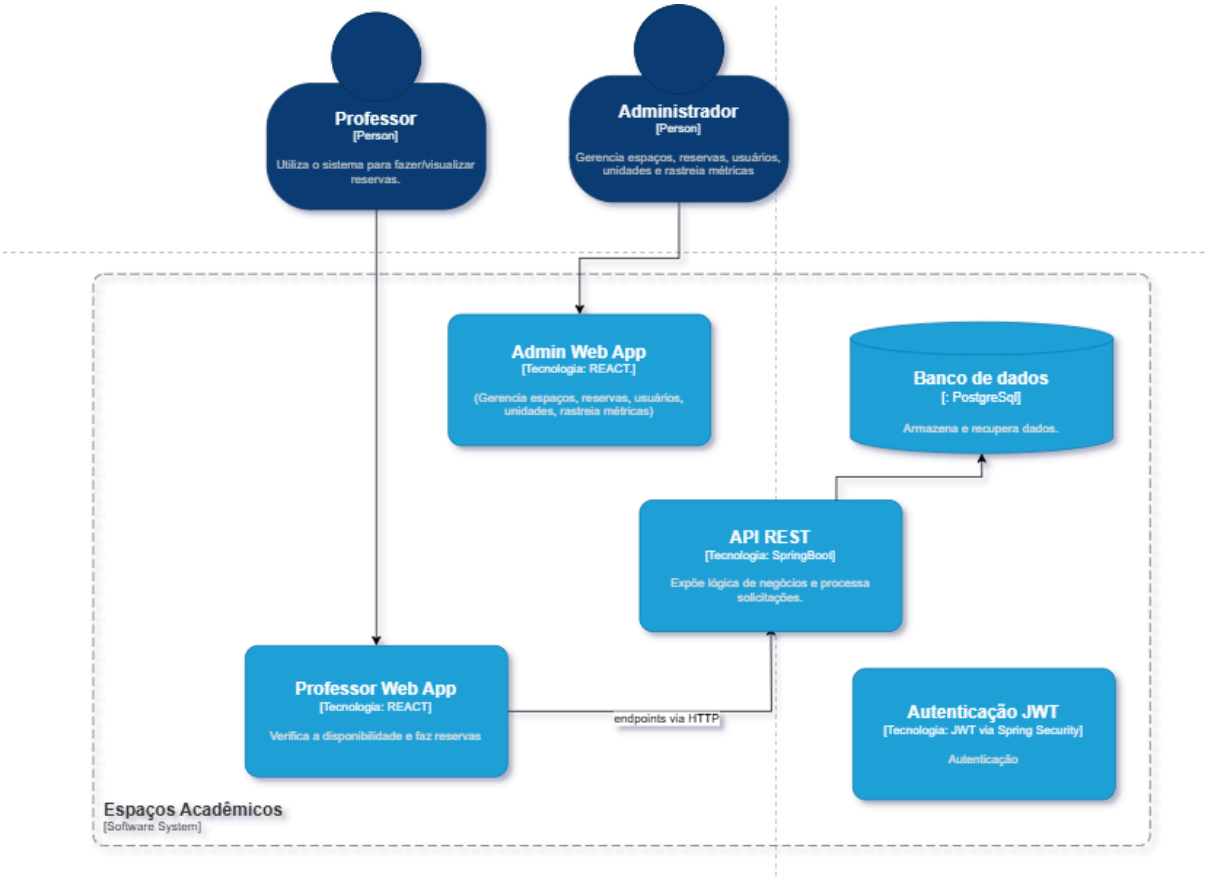

academicSpaceController
depends on
academicSpaceService
academicSpaceRepository
@Autowired
adminController
metricService
authController
httpAccessDeniedHandler
httpControllerExceptionHandler
insertSeedData
reservationRepository
mailNotificationEventHandler
@Autowired
resendMailProvider
backendApplication
reservationController
reservationService
authenticationManager
reservationSchedulerManager
jwtSecurityFilter
inMemoryUserDetailsManager
schoolController
schoolUnitService
OAuth2ImportSelector
@EnableWebSecurity
schoolUnitRepository
authenticationManagerBuilder
enableGlobalAuthenticationAutowiredConfigurer
userService
tokenService
userRepository
initializeAuthenticationProviderBeanManagerConfigurer
@EnableWebMvcSecurity
@Bean
corsConfigurationSource
initializeUserDetailsBeanManagerConfigurer
authenticationConfiguration
webApplicationContext
customAccessDeniedHandler
customAuthenticationEntryPoint
passwordEncoder
mvcContentNegotiationManager
@Autowired
securityConfig
httpSecurityConfiguration
objectPostProcessorConfiguration
@Order
methodSecuritySelector
@Configuration
@Import
userDetailsServiceImpl
objectPostProcessor
observationImportSelector
conversionServicePostProcessor
securityFilterChain
node54
springWebMvcImportSelector
delegatingApplicationListener
privilegeEvaluator
webSecurityConfiguration
springSecurityFilterChain
webSecurityExpressionHandler
userController

Entrega da documentação arquitetural modelo C4 - formato PDF

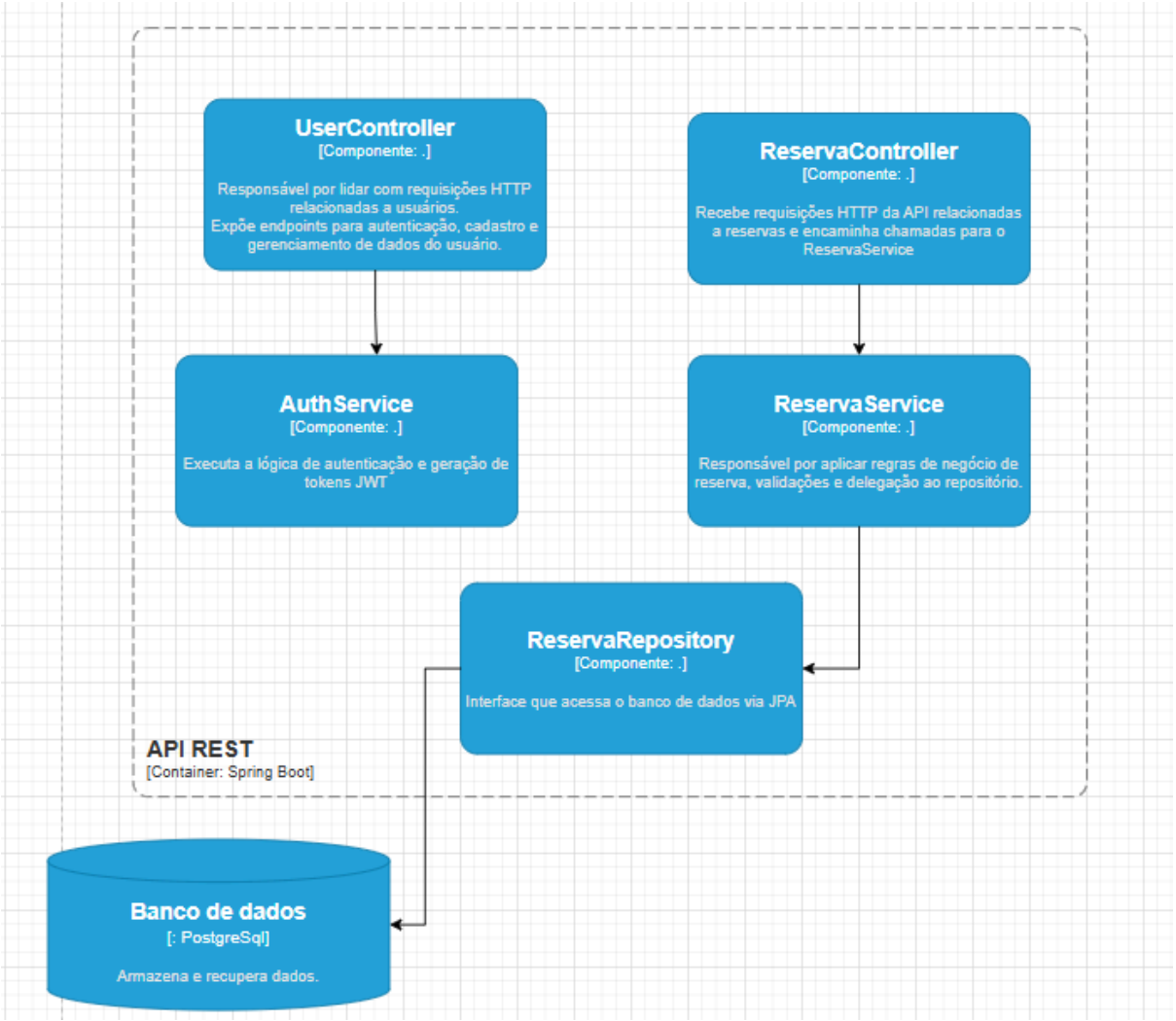
C4 model - nível 1



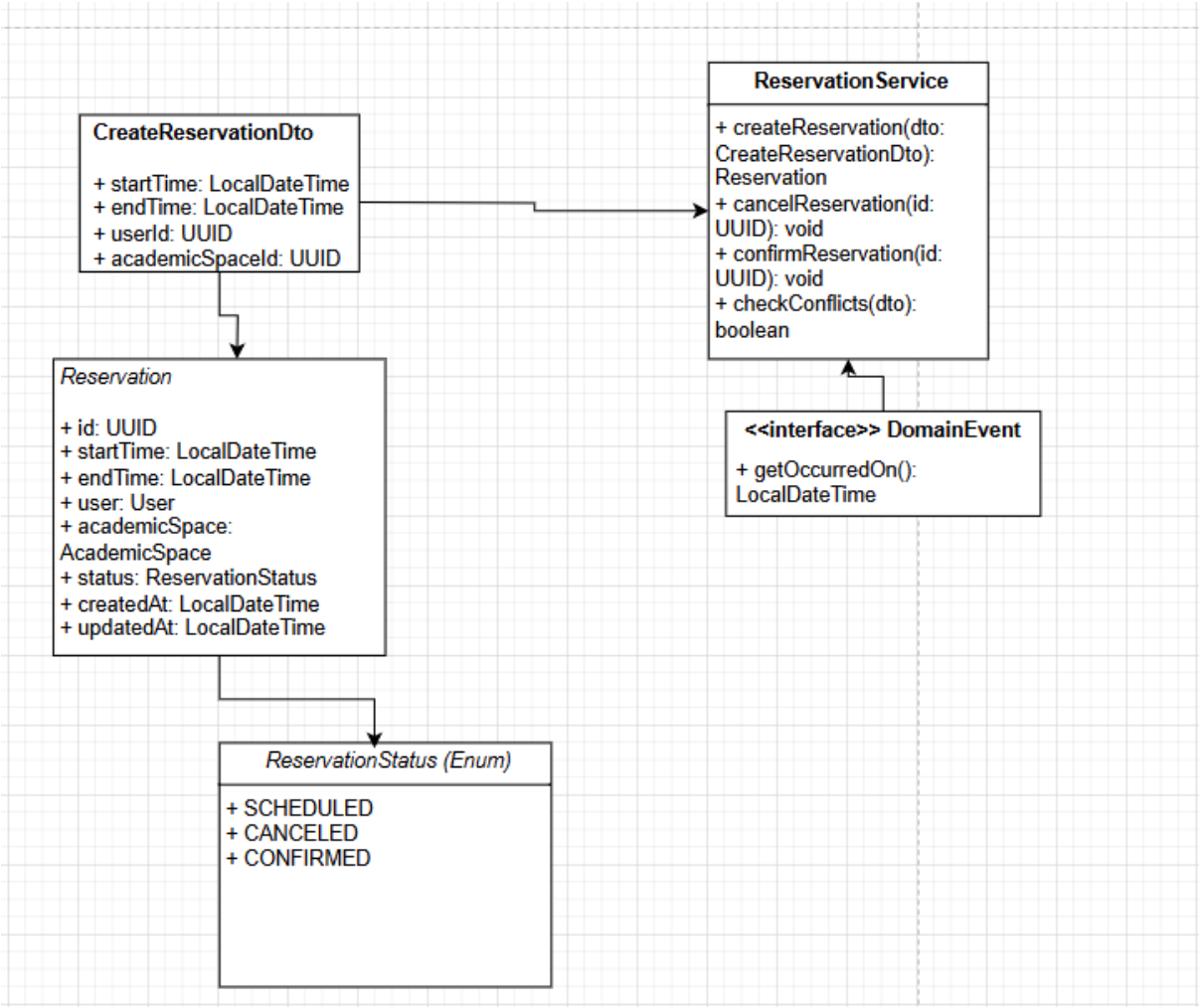
C4 Model - nível 2



C4 Model - nível 3



C4 Model - nível 4



1. Estrutura do Projeto:

O projeto foi estruturado com base em componentes reutilizáveis, organizando as partes da interface como blocos independentes. Também foi aplicado o conceito de separação de responsabilidades, mantendo a lógica, visual e serviços bem divididos.

O desenvolvimento do projeto seguiu a metodologia do Git Flow, que organiza o fluxo de trabalho com diferentes branches para diferentes propósitos. A branch main foi usada para manter a versão estável do sistema, enquanto a branch develop concentrou o desenvolvimento contínuo. Novas funcionalidades foram criadas a partir de branches feature/nome-da-funcionalidade, que eram testadas e depois integradas à develop. Isso garantiu um controle de versão eficiente e facilitou o trabalho em equipe, além de reduzir conflitos no código.

Na parte de arquitetura de software, adotamos uma arquitetura em camadas, separando claramente a lógica de apresentação (componentes visuais), serviços (regras de negócio e chamadas à API) e dados (respostas do back-end). Também aplicamos princípios de boas práticas, como o Princípio da Inversão de Dependência, mantendo o código desacoplado e mais fácil de testar e manter.

Para a construção do Front-End, utilizamos React.js, uma biblioteca moderna do JavaScript para criação de interfaces de usuário. O React possibilitou a criação de componentes reutilizáveis e uma navegação fluida utilizando o React Router DOM. As requisições ao back-end foram feitas com Axios, facilitando a comunicação com os serviços da API. O projeto também utilizou JavaScript (ES6+), garantindo um código limpo e atualizado. Além disso, a aplicação foi estruturada com uma boa separação entre lógica, visual e comunicação com a API, seguindo os princípios da manutenibilidade e escalabilidade.

1.2 Funcionalidades do Sistema

Gestão de Espaços Acadêmicos:

O sistema permite que administradores gerenciem os espaços acadêmicos disponíveis para reserva. As operações incluem a criação de novos espaços, atualização de informações sobre espaços existentes, e controle do status de disponibilidade desses espaços.

Gestão de Reservas:

Os usuários podem fazer reservas para utilizar os espaços acadêmicos. O sistema verifica conflitos de agendamento, e permite que os administradores cancelem ou modifiquem as reservas. A sobreposição de reservas é tratada para garantir que um mesmo espaço não seja reservado por múltiplos usuários ao mesmo tempo.

Gestão de Usuários:

Os usuários do sistema (professores, administradores) podem ser criados, atualizados, deletados e autenticados. O sistema garante que os administradores possam acessar todos os dados e que os professores tenham permissões limitadas a suas próprias reservas e espaços.

Gerenciamento de Unidades Escolares:

O sistema permite que os administradores vinculem usuários a unidades acadêmicas específicas, o que facilita a gestão de recursos.

Métricas:

O sistema oferece métricas que podem ser acessadas pelos administradores, incluindo contagem de reservas e distribuições de reservas ao longo da semana ou por espaço acadêmico.

2. Back-end

Este projeto é um sistema de controle de espaços acadêmicos desenvolvido com uma arquitetura de software baseada no padrão Domain-Driven Design (DDD), utilizando Spring Boot para a implementação do backend. A aplicação tem como objetivo a gestão de espaços acadêmicos, como salas de aula, laboratórios, e outros ambientes utilizados em uma instituição de ensino, permitindo que esses espaços sejam gerenciados e reservados por usuários como professores e administradores. A arquitetura do sistema é organizada em pacotes que seguem os princípios do DDD, onde cada pacote representa uma camada ou um domínio específico do sistema.

2.1. Arquitetura

A arquitetura do sistema é estruturada de maneira que os **domínios** e suas respectivas lógicas de negócio estão separados, utilizando os conceitos de **DDD**. Dessa forma, os pacotes são organizados de acordo com a responsabilidade de cada camada e com os tipos de funcionalidades que eles oferecem.

Camada de Domínio (Domain): Contém as entidades, serviços e lógica de negócio.

Camada de Infraestrutura (Infra): Implementa os serviços que interagem com o sistema externo (como por exemplo o banco de dados e APIs).

Camada Compartilhada (Shared): Contém as classes e utilitários comuns a diferentes partes do sistema, como exceções e entidades base.

2.2. Pacotes Principais e suas Funções

Pacote domain – Domínio do Sistema

O pacote domain contém os elementos que definem as regras de negócios. Ele é subdividido em pacotes que correspondem a diferentes aspectos do domínio.

Pacote space:

Função: Gerencia os espaços acadêmicos (como salas de aula, auditórios, laboratórios). Contém entidades e serviços que lidam com a criação, atualização, status e disponibilidade dos espaços.
Serviços principais: AcademicSpaceService, usado para manipular e gerenciar os espaços.

Pacote reservation:

Função: Gerencia as reservas dos espaços acadêmicos. Permite que usuários (professores, administradores) façam reservas para usar os espaços.

Serviços principais: ReservationService, responsável pela lógica de criação, atualização, cancelamento e verificação de sobreposição de reservas.

Pacote user:

Função: Gerencia os usuários do sistema, como professores e administradores. Inclui serviços para criação, atualização, remoção e autenticação de usuários.
Serviços principais: UserService, serviço para manipular dados de usuários, incluindo criação e atualização.

Pacote school:

Função: Gerencia as unidades escolares e permite que as unidades acadêmicas sejam associadas a usuários e espaços.
Serviços principais: SchoolUnitService, serviço que permite o gerenciamento das unidades acadêmicas.

Pacote infra – Infraestrutura

Este pacote contém as implementações técnicas necessárias para suportar o sistema, bem como as implementações de APIs REST e as interações com o banco de dados.

Controladores HTTP (controllers):

Os controladores são responsáveis por expor os serviços do sistema como APIs REST, manipulando as requisições HTTP. Exemplos incluem: AcademicSpaceController (para gerenciar espaços acadêmicos), ReservationController (para gerenciar reservas), AdminController (para funcionalidades administrativas), e UserController (para gerenciar informações do usuário).

Modelos de Resposta:

São utilizadas classes como ListResponse e PaginatedResponseBuilder para formatar respostas para a API de maneira consistente, incluindo paginação e formatação de listas.

Pacote shared

Este pacote contém classes utilitárias e abstrações que são reutilizadas no sistema.

Exceções (DomainException, DomainExceptionCode):

O nosso sistema utiliza exceções personalizadas para sinalizar erros específicos de domínio (por exemplo, ao tentar reservar um espaço que já está ocupado).

Entidades Base (DomainEntity):

O DomainEntity é uma classe base que fornece suporte para o gerenciamento de timestamps automáticos (data de criação e atualização) em entidades do sistema. Esta classe é herdada por outras entidades do domínio.

2.3 Arquitetura de Segurança

O sistema utiliza o **Spring Security** para controle de autenticação e autorização de usuários. A segurança é configurada de forma a garantir que apenas usuários autenticados com as permissões adequadas (como ROLE_ADMIN, ROLE_TEACHER) possam acessar determinados recursos.

Autenticação: Realizada através de tokens JWT, gerados quando o usuário faz login no sistema.

Autorização: As rotas do sistema são protegidas por anotações como @PreAuthorize, permitindo a definição de regras de acesso baseadas em roles.

2.4 Tecnologias utilizadas

Spring Boot: Framework principal utilizado para o desenvolvimento do backend.

Spring Security: Para autenticação e autorização de usuários.

JPA/Hibernate: Para mapeamento objeto-relacional e persistência de dados.

Swagger/OpenAPI: Para documentação automática das APIs REST.

JWT: Para a geração e validação de tokens de autenticação.

2.5 Pacote domain

Como supracitado, o pacote domain representa o núcleo da aplicação. É responsável por toda a lógica de negócios, incluindo entidades, serviços, regras e eventos. Tudo que rege o funcionamento interno do sistema está centralizado nele, com o objetivo de manter baixo acoplamento em relação às demais camadas da aplicação.

Subpacote domain.reservation

Responsável pela lógica relacionada à gestão de reservas de espaços acadêmicos.

Componentes:

DTOs (application.dto)

CreateReservationDto, UpdateReservationDto: Representam os dados necessários para criação ou atualização de uma reserva.
Possuem validações com anotações como @NotNull, @UUID.

Repositórios (application.repository)

ReservationRepository: Interface para manipulação de dados de reservas.
Contém consultas específicas, como: Verificação de sobreposição de reservas, filtros por nome do professor ou espaço, listagem de reservas expiradas e paginação com filtros.

Serviços (application.services)

ReservationService: Serviço central para o gerenciamento de reservas.

Esta, aplica regras de negócio como validações de conflito, cancelamento, verificação de usuários e emite eventos como ReservationCanceledEvent.

ReservationSchedulerManager: Executa tarefas programadas, como por exemplo, marca reservas expiradas como "confirmadas pela instituição" todos os dias às 3h.

Entidades (enterprise.entity)

Reservation: Entidade principal de reserva, relacionamentos com User e AcademicSpace.Métodos auxiliares como isScheduled(), isCanceled() e isConfirmed().

ReservationStatus: Enum que define os possíveis estados da reserva:
SCHEDULED, CANCELED, CONFIRMED_BY_THE_USER, CONFIRMED_BY_THE_ENTERPRISE.

Eventos (enterprise.events)

ReservationCanceledEvent: Evento emitido quando uma reserva é cancelada. Este é usado para acionar reações em outros subpacotes, como envio de notificações.

Subpacote domain.user

Responsável pela lógica de usuários do sistema, incluindo administradores e professores.

Componentes:

DTOs

CreateUserDto, UpdateUserDto, MakeLoginDto, RegisterTeacherDto: Representam os dados para criação, login, registro e edição de usuários.

Entidade

User: Entidade que representa o usuário, associado a uma unidade escolar.
UserRole: Enum que define os papéis possíveis (ADMIN, TEACHER).

Repositórios

UserRepository: Interface para manipulação de dados dos usuários, consultas por ID, e-mail e unidade escolar.
UserProjection: Interface para retornos otimizados e paginados com dados parciais dos usuários.

Serviços

UserService: Responsável pela criação, edição, busca e deleção de usuários.
MetricService: Expõe métricas agregadas do sistema, como: Total de usuários, reservas, espaços e gráficos de uso.

Subpacote domain.space

Responsável pela lógica de gestão dos espaços acadêmicos disponíveis para reserva.

Componentes:

DTOs

CreateAcademicSpaceDto: Dados necessários para criar um novo espaço.
ChangeSpaceStatusDto: Recebe o novo status (AVAILABLE, UNAVAILABLE) e o ID do espaço.

Entidade

AcademicSpace: Representa uma sala ou ambiente disponível para reservas.
Atributos como nome, sigla, descrição, capacidade e status.

Enum

SpaceStatus: Define os estados possíveis do espaço (disponível ou indisponível).

Repositório

AcademicSpaceRepository: Interface de consulta. Esta, permite filtragem por nome, sigla e status, além disso, inclui query nativa para recuperar as 10 salas mais reservadas nos últimos 7 dias.

Serviço

AcademicSpaceService: Responsável pela criação e alteração de status de espaços, busca paginada e listagem de espaços disponíveis.

Subpacote domain.school

Responsável por gerenciar unidades escolares.

Componentes:

DTO

CreateSchoolUnitDto: Representa os dados necessários para criar uma nova unidade escolar.

Repositório

SchoolUnitRepository: Interface de persistência de unidades escolares.

Serviço

SchoolUnitService: Responsável pela criação de novas unidades escolares.

Subpacote domain.notification

Responsável por reagir a eventos de domínio e iniciar ações externas como o envio de e-mails.

Componentes:

Interface de Contrato

MailProvider: Interface que define a operação de envio de e-mails (implementação concreta é externa ao domínio). Este subpacote permite que a lógica de domínio reaja a eventos sem conhecer os detalhes das ações executadas, garantindo desacoplamento e flexibilidade.

Pacote shared

O pacote shared contém classes e recursos reutilizáveis que oferecem suporte às demais funcionalidades do sistema. Ele fornece soluções genéricas e centralizadas para problemas, como gestão de tempo e tratamento de exceções.

Componentes do Pacote:

Classe DomainEntity

Classe base para entidades com controle automático de datas de criação e atualização (createdAt, updatedAt).

Classe DomainException

Exceção personalizada que encapsula erros de domínio com mensagens e códigos específicos.

Enum DomainExceptionCode

Enum que lista códigos padronizados para categorizar os erros lançados por DomainException.

Pacote infra

Camada de infraestrutura responsável por lidar com serviços externos, configurações de segurança, envio de e-mails e scripts auxiliares. Essa camada implementa contratos definidos na camada domain.

Subpacote http.exception

HttpControllerExceptionHandler

Esta classe responsável por capturar exceções lançadas pelos controllers e retornar respostas padronizadas com mensagens e códigos de erro. Utiliza anotações do Spring (@RestControllerAdvice, @ExceptionHandler).

MethodArgumentNotValidException: Retorna erros de validação de campos em requisições.
JWTVerificationException: Trata falhas na verificação de token JWT.
DomainException: Erros de domínio definidos pela aplicação.
MethodArgumentTypeMismatchException: Erros de tipo nos parâmetros de rota ou query.
Exception: Qualquer erro que não seja tratado.
IllegalArgumentException: Argumentos inválidos com detalhamento da causa.
TokenExpiredException: Token JWT expirado.

Subpacote http.model

ListResponse<ListContent>

Builder utilitário para retornar listas em ResponseEntity.

LoginResponseEntity

Builder utilitário que retorna uma resposta com o token de autenticação.

PaginatedResponseBuilder<Content>

Builder de resposta para dados paginados. Campos: page, pageSize, totalOfPages, content.

UserResponse

Record de resposta para dados do usuário autenticado, incluindo: id, email, name, role.

Controllers

AcademicSpaceController

Endpoint: /spaces
Anotação de segurança: @PreAuthorize("hasAnyRole('ROLE_TEACHER', 'ROLE_ADMIN')")
Responsabilidade: Retorna espaços acadêmicos disponíveis.

AdminController

Endpoint: /admin
Anotação de segurança: @PreAuthorize("hasRole('ROLE_ADMIN')")
Responsabilidade: Gestão de usuários, espaços, reservas e métricas administrativas.
Principais endpoints: Criar, atualizar, listar e deletar usuários; Criar e listar espaços acadêmicos; Alterar status de espaços; Buscar e cancelar reservas; Métricas de reservas (contagem por dia da semana, por espaço nos últimos 7 dias, etc.).

AuthController

Endpoint: /auth
Acesso livre: @PreAuthorize("permitAll()")
Responsabilidade: Autenticação de usuários (login).

ReservationController

Endpoint: /reservations
Anotação de segurança: @PreAuthorize("hasAnyRole('ROLE_TEACHER', 'ROLE_ADMIN')")
Responsabilidade: Criar, listar, atualizar e fazer checkout de reservas.

SchoolUnitController

Endpoint: /school-units
Anotação de segurança: @PreAuthorize("hasAnyRole('ROLE_ADMIN')")
Responsabilidade: Criar unidade escolar, listar todas. Listar professores vinculados a uma unidade específica.

UserController

Endpoint: /users
Anotação de segurança: @PreAuthorize("hasAnyRole('ROLE_TEACHER', 'ROLE_ADMIN')")
Responsabilidade: Retornar dados do usuário autenticado (/users/me).

infra.mail

ResendMailProvider

Responsabilidade: Implementa o contrato MailProvider para envio de e-mails via serviço Resend.
Anotações: @Component
Dependências: Resend (SDK externa) e @Value para injeção da API key

Método principal: sendEmail(String to, String subject, String body): Envia um e-mail com remetente fixo.

infra.scripts

InsertSeedData

Responsabilidade: Insere dados fictícios no banco de dados ao inicializar a aplicação.
Anotações: @Component, implementa CommandLineRunner
Configurações lidas: spring.datasource.url, spring.datasource.username, spring.datasource.password, application.config.database.run_seed

Ações principais: Insere um usuário ADMIN com dados fictícios, usando BCryptPasswordEncoder.

infra.security

SecurityConfig

Responsabilidade: Define a configuração global de segurança da aplicação.
Anotações: @Configuration, @EnableWebSecurity, @SecurityScheme
Configurações: CORS, CSRF desativado, JWT como mecanismo de autenticação, configura entrada de autenticação customizada e tratamento de acesso negado e registra o filtro JwtSecurityFilter.

JwtSecurityFilter

Responsabilidade: Intercepta requisições HTTP para validar e extrair dados do token JWT.
Anotações: @Component
Lógica: Recupera e valida o token JWT (exceto em /auth/sign-in) e injeta o usuário autenticado no contexto de segurança.

TokenService

Responsabilidade: Geração e validação de tokens JWT.
Anotações: @Service
Configurações lidas: api.security.jwt.secret
Lógica de geração: Cria token com expiração de 3 horas.

HttpAccessDeniedHandler

Responsabilidade: Trata requisições negadas com erro 401 (não autorizado).
Anotações: @Component

UserDetailsServiceImpl

Responsabilidade: Implementa UserDetailsService para buscar usuário por e-mail.
Anotações: @Service
Dependência: UserRepository (da camada domain)

UserAuthenticated

Responsabilidade: Implementa UserDetails para representar o usuário autenticado no Spring Security.
Permissões: Define authorities com base no role (ADMIN, TEACHER)
Nota: O método getUsername() está retornando a senha por engano – pode ser uma inconsistência.