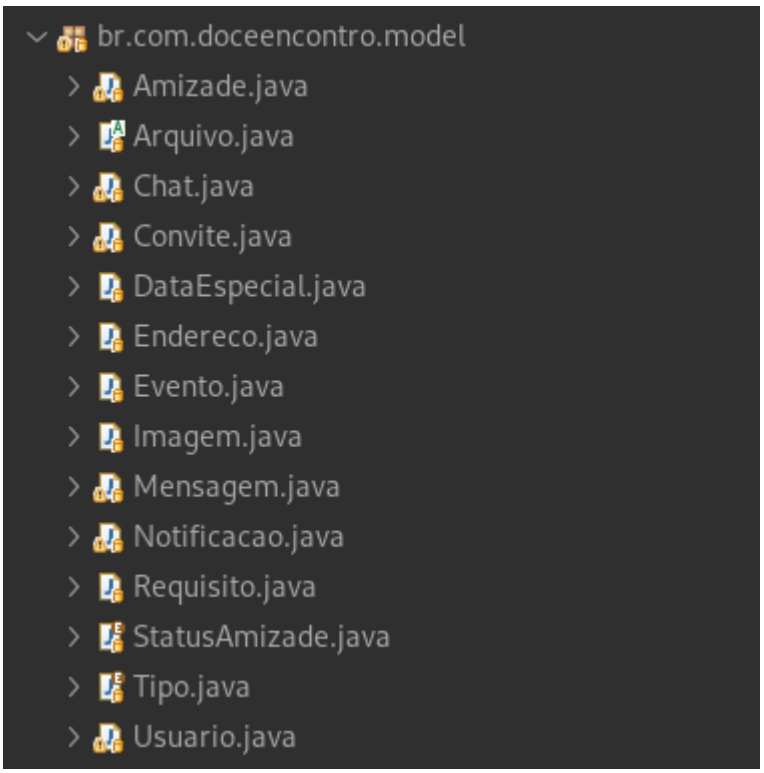
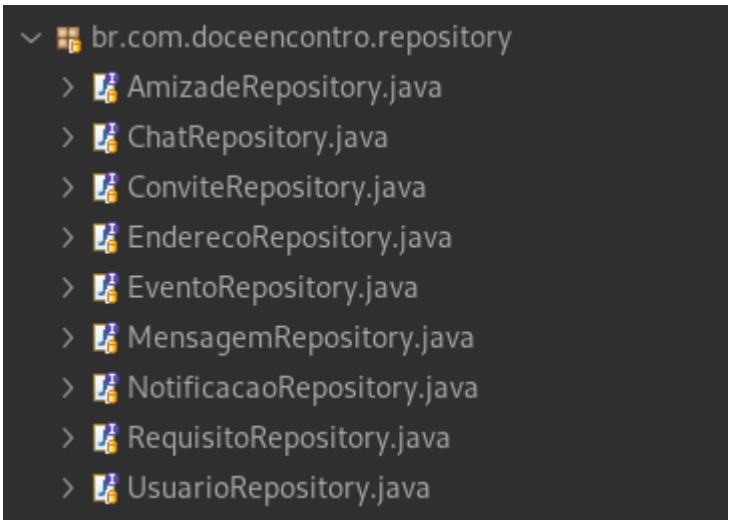
Startup Project Final: Doce Encontro

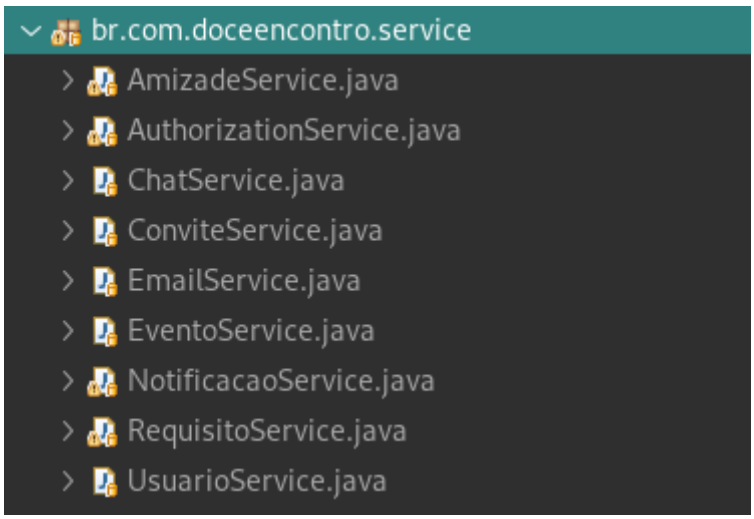
1. Estrutura de camadas do backend

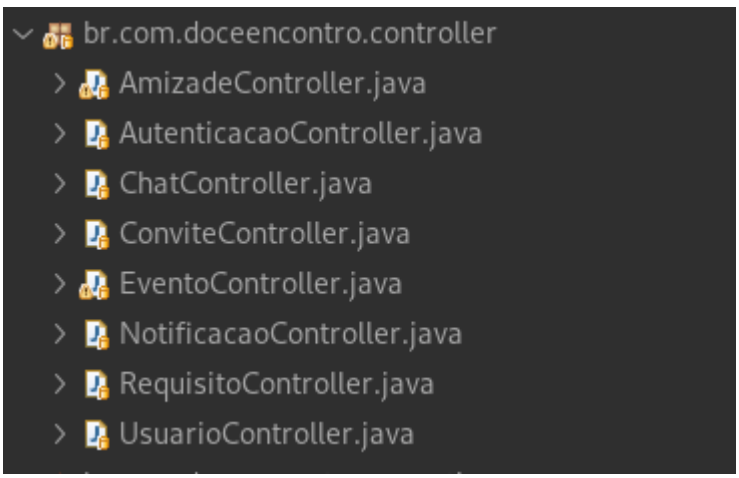
As camadas do projeto são separadas são separadas em:

- Model: Com as entidades;
- Repository: Responsável pelo mapeamendo objeto relacional;
- Service: camada onde são implementadas as regras de negócio;
- Controller: Para receber as requisições HTTP.



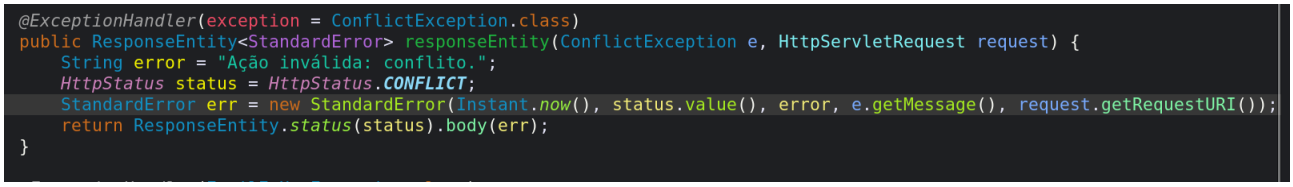




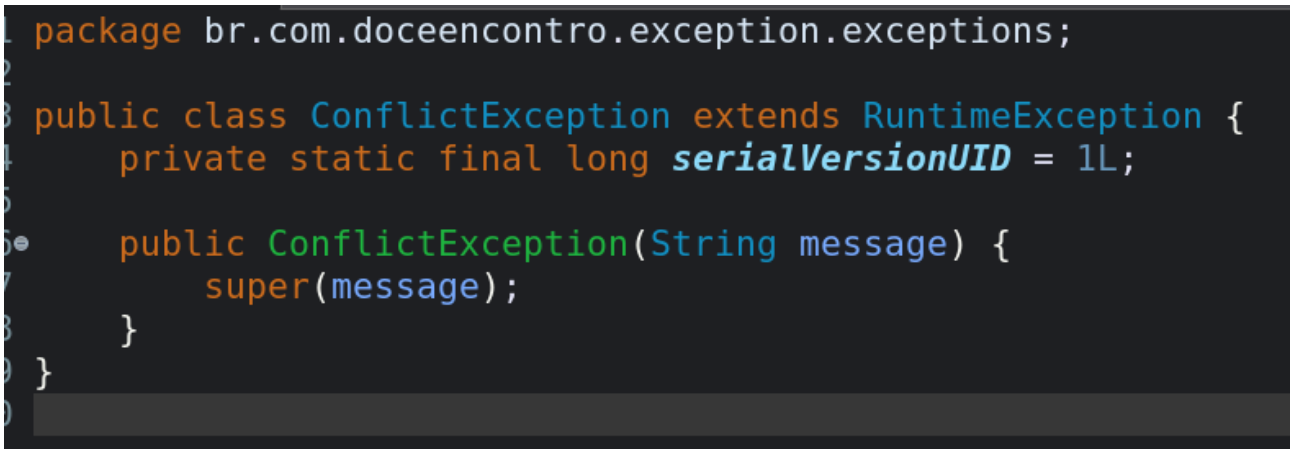


2. Exceções

As exceções são baseadas nos códigos de erro HTTP. As exceções utilizam conceitos de polimorfismo para as exceções HTTP genéricas poderem tomar forma de outras exceções, tornando o expansivo e evitando a repetição de código.



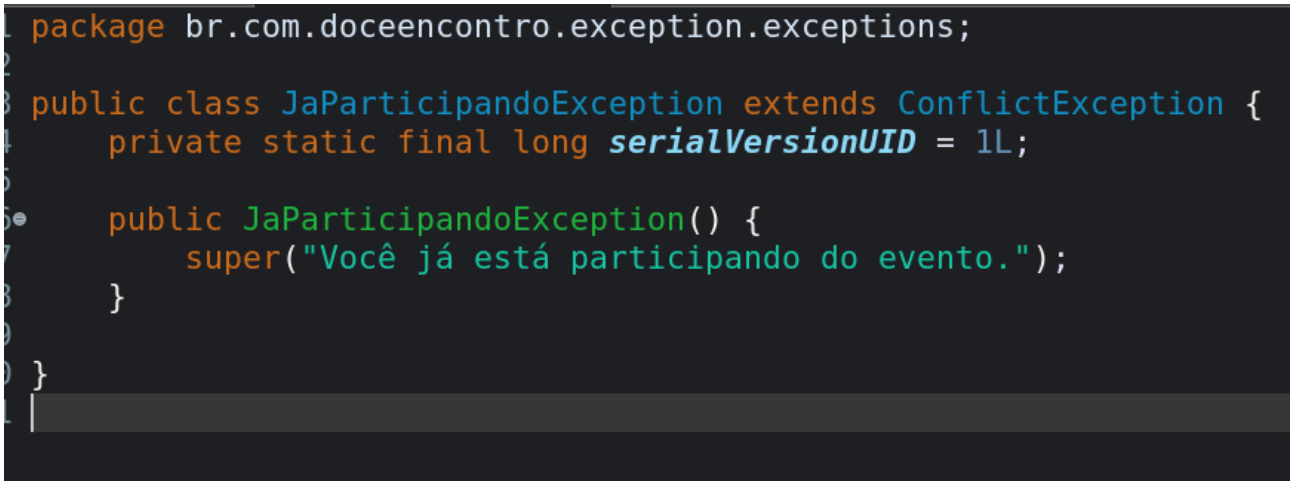
```java
@ExceptionHandler(exception = ConflictException.class)
public ResponseEntity<StandardError> responseEntity(ConflictException e, HttpServletRequest request) {
    String error = "Ação inválida: conflito.";
    HttpStatus status = HttpStatus.CONFLICT;
    StandardError err = new StandardError(Instant.now(), status.value(), error, e.getMessage(), request.getRequestURI());
    return ResponseEntity.status(status).body(err);
}
```



```java
package br.com.doceencontro.exception.exceptions;

public class ConflictException extends RuntimeException {
    private static final long serialVersionUID = 1L;

    public ConflictException(String message) {
        super(message);
    }
}
```



```java
package br.com.doceencontro.exception.exceptions;

public class JaParticipandoException extends ConflictException {
    private static final long serialVersionUID = 1L;

    public JaParticipandoException() {
        super("Você já está participando do evento.");
    }

}
```

.

3. DTOs

O projeto utiliza DTOs para transferir para o frontend, apenas os dados necessários para o contexto da aplicação. Também é utilizado para evitar loops de JSON e não expor dados sensíveis do usuário no frontend.

```java
public class UsuarioDetailsDTO {

    private String id;

    private String nome;

    private String email;

    private List<EventoResponseDTO> eventosCriados;

    private List<EventoResponseDTO> eventosParticipados;

    public UsuarioDetailsDTO(Usuario usuario) {
        this.id = usuario.getId();
        this.nome = usuario.getNome();
        this.email = usuario.getEmail();

        this.eventosCriados = converterEventoDTO(usuario.getEventosCriados());
        this.eventosParticipados = converterEventoDTO(usuario.getEventosParticipados());

    }

    private List<EventoResponseDTO> converterEventoDTO(List<Evento> eventos) {
        return eventos.stream()
                .map(e -> new EventoResponseDTO(e))
                .collect(Collectors.toList());

    }
}
```

.

```java
package br.com.doceencontro.model.dtos;

import br.com.doceencontro.model.Usuario;

@Data
public class UsuarioResponseDTO {

    private String id;

    private String nome;

    public UsuarioResponseDTO(Usuario usuario) {
        this.id = usuario.getId();
        this.nome = usuario.getNome();
    }

}
```

4. Consistência no registro de dados

Para evitar dados incorretos sejam registrados na plataforma, a aplicação utiliza de Annotations, algumas personalizadas, para garantir uma consistência maior nos registros. Já as mensagens de erro declaradas em uma classe utilitária do pacote utils, em atributos estáticos, para que possam ser reutilizados em outros contextos da aplicação.

```
public record EventoRequestDTO(
        @NotBlank(message = MensagemErro.OBRIGATORIO) @Length(min = 10, message = MensagemErro.MIN)
        @Length(max = 100, message = MensagemErro.MAX)
        String titulo,

        @NotBlank(message = MensagemErro.OBRIGATORIO) @Length(min = 10, message = MensagemErro.MIN)
        String descricao,

        @NotBlank(message = MensagemErro.OBRIGATORIO)
        String tipo,

        @NotNull(message = MensagemErro.OBRIGATORIO) @Future(message = MensagemErro.FUTURO)
        LocalDateTime data,

        String local,

        @NotBlank(message = MensagemErro.OBRIGATORIO)
        String estado,

        @NotBlank(message = MensagemErro.OBRIGATORIO)
        String cidade,

        @NotBlank(message = MensagemErro.OBRIGATORIO)
        String rua,

        Integer numero
        ) {

}
```

```
package br.com.doceencontro.utils;

public class MensagemErro {

    public static final String OBRIGATORIO = "Campo obrigatório.";

    public static final String INVALIDO = "Campo inválido.";

    public static final String MIN = "O campo deve conter no mínimo {min} caracteres.";

    public static final String MAX = "O campo deve conter no máximo {max} caracteres.";

    public static final String FUTURO = "A data deve estar no futuro.";

    private MensagemErro() {}
}
```

```
package br.com.doceencontro.annotation;

import jakarta.validation.ConstraintValidator;

public class SenhaForteValidator implements ConstraintValidator<SenhaForte, String> {

    private static final String REGEX = "^(?=.*[A-Z])(?=.*[a-z])(?=.*\\d)(?=.*[@$!%*?&])[A-Za-z\\d@$!%*?&]{8,}$";

    @Override
    public boolean isValid(String senha, ConstraintValidatorContext context) {
        return senha != null && senha.matches(REGEX);
    }
}
```

5. Aspects

A aplicação utiliza Aspects para executar códigos apenas se uma ação anterior tenha sido um sucosso, assim garantindo a consistência da aplicação.

```
@AfterReturning(
        pointcut = "execution(* br.com.doceencontro.service.EventoService.editarEvento(..))",
        returning = "result")
public void notificarEdicao(EventoResponseDTO result) {
    String eventoTitulo = result.getTitulo();
    String eventoId = result.getId();

    Evento buscarEvento = eventoService.findById(eventoId);

    String notificacaoTitulo = "Alterações em " + eventoTitulo + ".";
    String notificacaoCorpo = "As informações do evento " + eventoTitulo + " foram alteradas.";

    notificacaoService.notificarParticipantes(buscarEvento, notificacaoTitulo, notificacaoCorpo);

    emailService.enviarEmailParticipantes(buscarEvento, notificacaoTitulo, notificacaoCorpo);

}
```

6. Reutilização de código

Um dos métodos para garantir um código limpo e reutilizável, é a utilização de funções estáticas no pacote utils:

```
public class EventoUtils {

    public static boolean isParticipando(Evento evento, String usuarioId) {
        Optional<Usuario> buscarUsuario = evento.getParticipantes().stream()
                .filter(p -> p.getId().equals(usuarioId))
                .findFirst();

        if (buscarUsuario.isPresent()) {
            return true;
        }
        return false;
    }

    public static void garantirParticipacao(Evento evento, String usuarioId) {
        Optional<Usuario> buscarUsuario = evento.getParticipantes().stream()
                .filter(p -> p.getId().equals(usuarioId))
                .findFirst();

        if (buscarUsuario.isEmpty()) {
            throw new NotParticipandoException();
        }
    }
```

Outra forma, é disponibilizar findByIds com tratamento de exceção em todos os Services, que tem o propósito de ser reutilizado em outros contextos da aplicação:

```
public List<Usuario> findAll() {
    return usuarioRepository.findAll();
}

public Usuario findById(String id) {
    Optional<Usuario> buscarUsuario = usuarioRepository.findById(id);

    if (buscarUsuario.isEmpty()) {
        throw new UsuarioNotFoundException();
    }
```

```
public AmizadeResponseDTO adicionarAmigo(String usuarioId, String amigoEmail) {
    Usuario amigo = usuarioService.buscarPorEmailExcetoId(amigoEmail, usuarioId);

    Usuario usuario = usuarioService.findById(usuarioId);

    Optional<Amizade> buscarAmizade = amizadeRepository.buscarAmizade(usuarioId, amigo.getId());

    if (buscarAmizade.isPresent()) {
        Amizade amizadeExistente = buscarAmizade.get();

        if (amizadeExistente.getStatus().equals(StatusAmizade.ACEITO)) {
            throw new AmizadeExistenteException();
        }
        throw new PedidoExistenteException();
    }

    Amizade novaAmizade = new Amizade(null, usuario, amigo, StatusAmizade.PENDENTE);

    novaAmizade.addAmigo(usuario, amigo);

    return converterDto(amizadeRepository.save(novaAmizade));

}
```

A forma mais utilizada de reutilizar código na aplicação, é a de retornar o token por meio de uma função estática, que é utilizada por todos os controllers do projeto.

```
public class IdToken {
    public static String get() {
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        Usuario usuario = (Usuario) authentication.getPrincipal();

        return usuario.getId();
    }
}
```

```
@PostMapping("/{amigoEmail}")
public AmizadeResponseDTO adicionarAmigo(@PathVariable String amigoEmail) {
    return amizadeService.adicionarAmigo(IdToken.get() ,amigoEmail);
}
```

7. Performance

Uma das formas de melhorar a performance, é carregando todas as entidades necessárias e fazendo os tratamentos necessários em memória, ao invés de fazer vários selects no banco de dados.

```
public String participar(String eventoId, String usuarioId) {
    Evento buscarEvento = findById(eventoId);
    Usuario buscarUsuario = usuarioService.findById(usuarioId);

    // Verificando se o usuário foi convidado
    Optional<Usuario> buscarConvidado = buscarEvento.getConvite().getDestinatarios().stream()
    .filter(c -> c.getId().equals(usuarioId))
    .findFirst();

    if (buscarConvidado.isEmpty()) {
        throw new NotConvidadoException();
    }

    buscarEvento.addParticipante(buscarUsuario);

    buscarEvento.getChat().adicionarParticipante(buscarUsuario);

    buscarEvento.getConvite().getDestinatarios().remove(buscarUsuario);

    eventoRepository.save(buscarEvento);

    return "Usuário adicionado com sucesso";
}
```

Outra forma, é o uso de funções assíncronas, dessa forma, o usuário não precisa esperar todos os processos terminarem para receber uma resposta, como por exemplo, o envio de emails de forma assíncrona.

```
@Async
public void enviarEmailParticipantes(Evento evento, String assunto, String corpo) {
    SimpleMailMessage message = new SimpleMailMessage();

    message.setFrom(dcenEmail);

    message.setSubject(assunto);

    for (Usuario participante : evento.getParticipantes()) {
        String saudacao = "Olá " + participante.getNome() + ", ";
        message.setText(saudacao + Character.toLowerCase(corpo.charAt(0)) + corpo.substring(1));

        message.setTo(participante.getEmail());
        try {
            javaMailSender.send(message);
        } catch (MailException e) {
            System.out.println("Erro ao enviar email: " + e.getMessage());
        }
    }
```

8. Segurança

Além do uso de DTOs para evitar a exposição de dados sensíveis dos usuários, a aplicação utiliza o JWT para o controle de sessão.

```java
@Service
public class TokenService {

    @Value("${dcenPass}")
    private String tokenPass;

    public String generateToken(Usuario userModel){
        try {
            Algorithm algorithm = Algorithm.HMAC256(tokenPass);

            String token = JWT.create()
                .withIssuer("auth")
                .withSubject(userModel.getId())
                .withExpiresAt(getExpirationDate())
                .sign(algorithm);
            return token;


        } catch (JWTCreationException exception) {
            throw new RuntimeException("ERROR WHILE GENERATING TOKEN", exception);
        }
    }
```

Utiliza também criptografia nas senhas.

```java
public ResponseEntity<Object> register (RegisterDTO registerDto){
    if (verificarEmailExistente(registerDto.email())) {
        throw new EmailEmUsoException();
    }

    // Criptografando as senhas
    String encryptedPassword = new BCryptPasswordEncoder().encode(registerDto.senha());

    Usuario newUser = new Usuario(registerDto.nome(), registerDto.email(), encryptedPassword);
    this.userRepository.save(newUser);
    return ResponseEntity.ok().build();
}
```

Também faz uso de variáveis de ambiente para não expor informações sensíveis.

```
dcenPass=${DCEN_PASS}
dcenEmail=${DCEN_EMAIL}
host=${HOST}

spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=${DCEN_EMAIL}
spring.mail.password=${DCEN_EMAIL_PASS}
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

Por fim, exige autenticação na maioria das rotas.

```
@Bean
public SecurityFilterChain securityFilterChain (HttpSecurity httpSecurity) throws Exception{
    return httpSecurity
        .csrf(csrf -> csrf.disable())
        .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
        .authorizeHttpRequests(authorize -> authorize
                .requestMatchers(HttpMethod.POST,"/usuarios/login").permitAll()
                .requestMatchers(HttpMethod.POST, "/usuarios/registrar").permitAll()
                .requestMatchers("/usuarios/**").authenticated()
                .requestMatchers("/eventos/**").authenticated()
                .anyRequest().permitAll()
            )
    .addFilterBefore(securityFilter, UsernamePasswordAuthenticationFilter.class)
    .build();
}
```