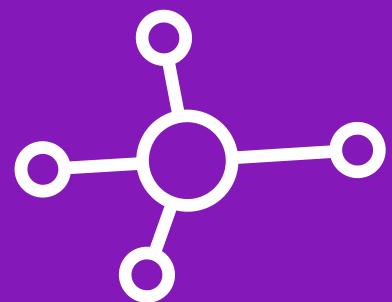


#Dev #Land



A rede social que conecta Devs
e Empresas Tech.



 **VEM SER**
DBC



César Augusto

Dev Java
São Paulo/ SP



Dayvidson Veiga

Dev Java
Campos/ RJ



Rafael Ferreira

Dev Java
Teresina/ PI



Até o final de 2022 o mercado de trabalho terá uma carência de aproximadamente

408 mil

devs

Há urgência em formar novos profissionais e qualificar carreiras já atuantes no mundo tech.



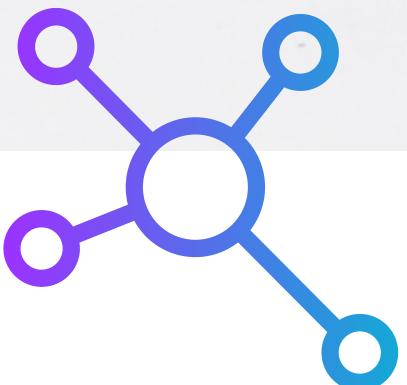
**A DevLand tem
como missão ser
a ponte entre
empresas e
profissionais
desenvolvedores**

Facilitando o networking e a
divulgação de programas de
formação e qualificação

Público Alvo

Empresas

A empresa pode divulgar oportunidades, além de se conectar com talentos através de filtros específicos



Devs

O desenvolvedor pode ficar atualizado com tecnologias e ter acesso a oportunidades de carreira



Principais Funcionalidades da Aplicação

**Criar um perfil
(Usuário DEV, ou
EMPRESA)**



Seguir e ser seguido



**Postagens e
Comentários**



Permite a criação de dois tipos de usuário com especificidades de permissionamentos, e segurança com login e senha.

Permite aos usuários seguirem outros, e serem seguidos

Permite cada usuário publicar postagens em 3 principais linhas: VAGAS, PROGRAMAS e PENSAMENTOS; além de também fazer comentários sobre cada postagem.





MODELO ENTIDADE RELACIONAMENTO (MER) DEVLAND - SOCIAL NETWORK </>

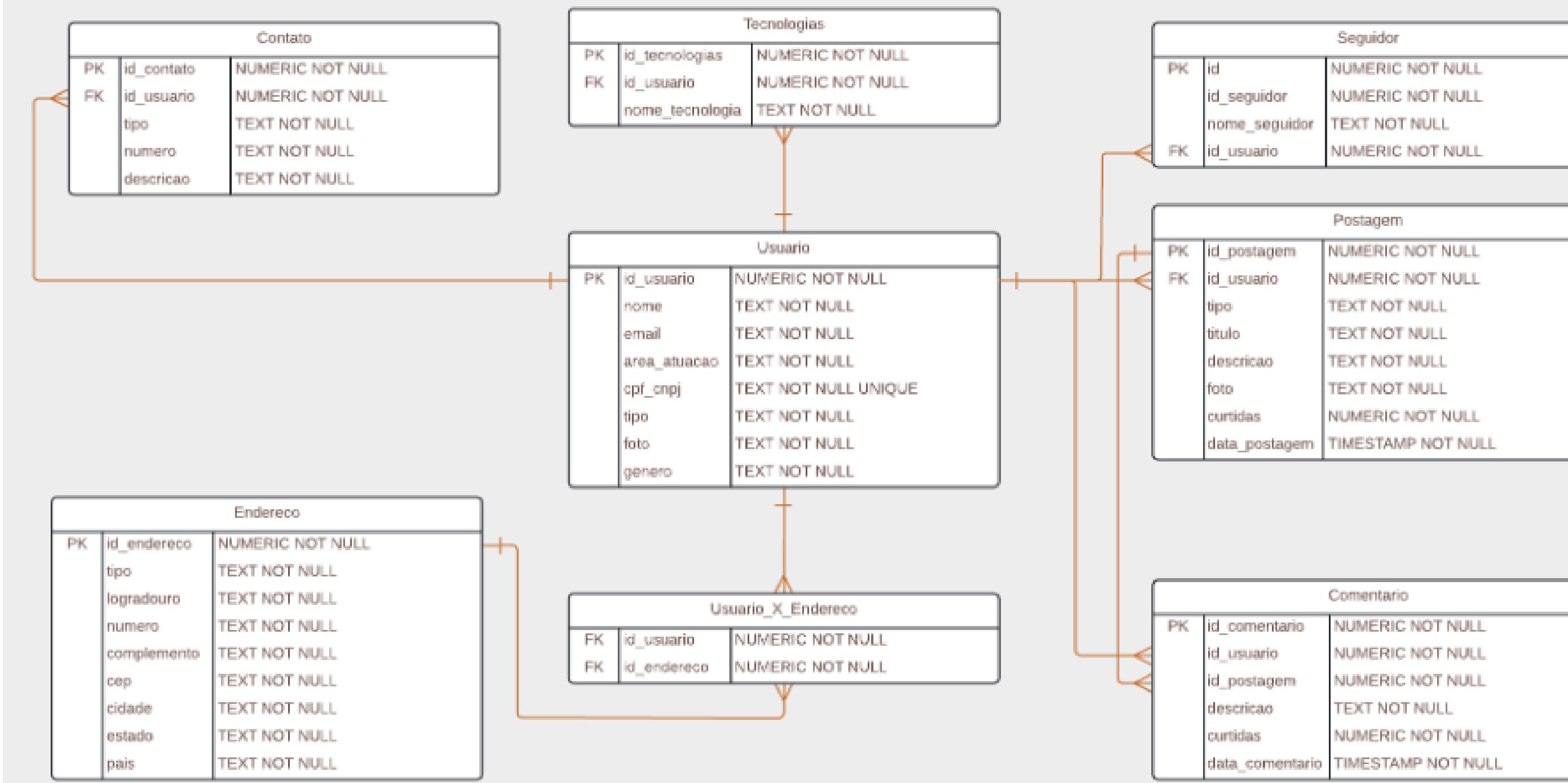


Diagrama de permissionamento

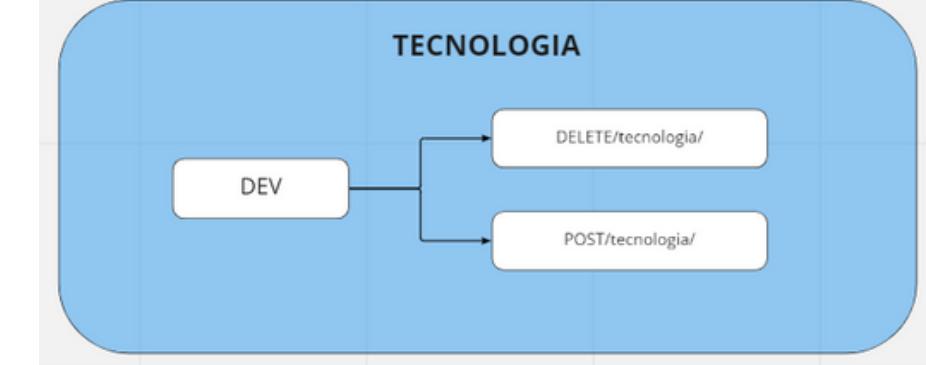
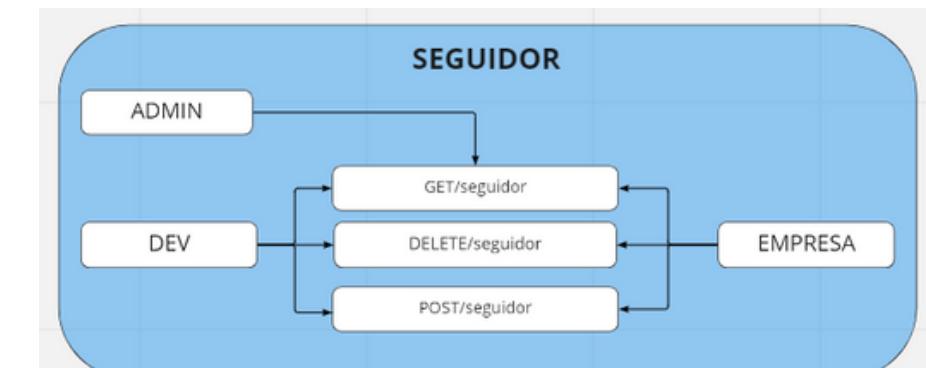
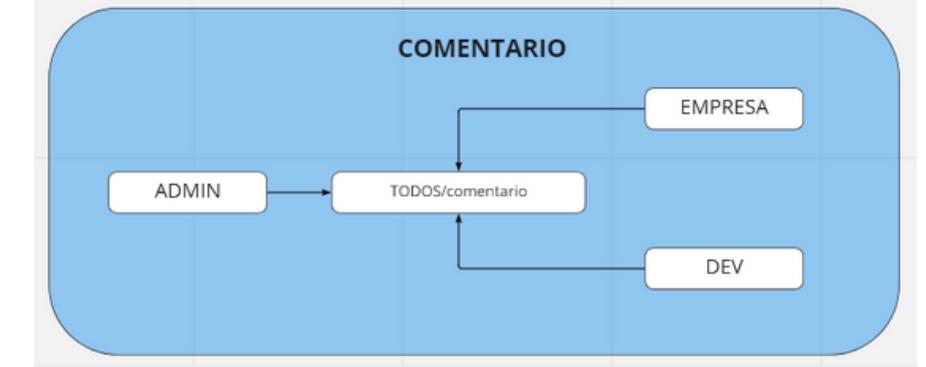
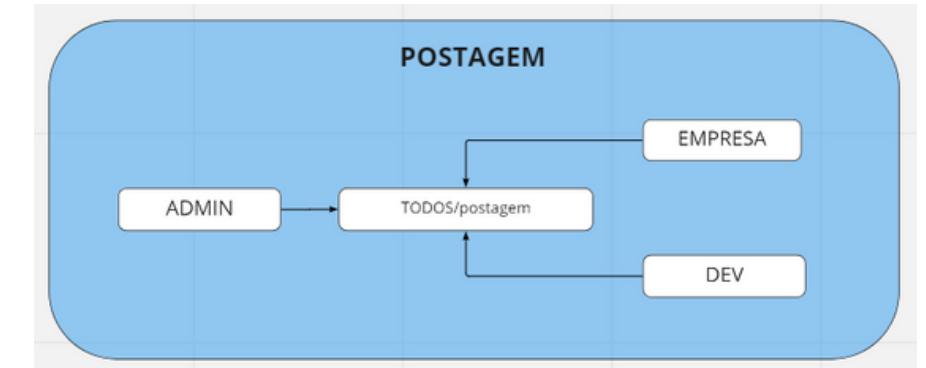
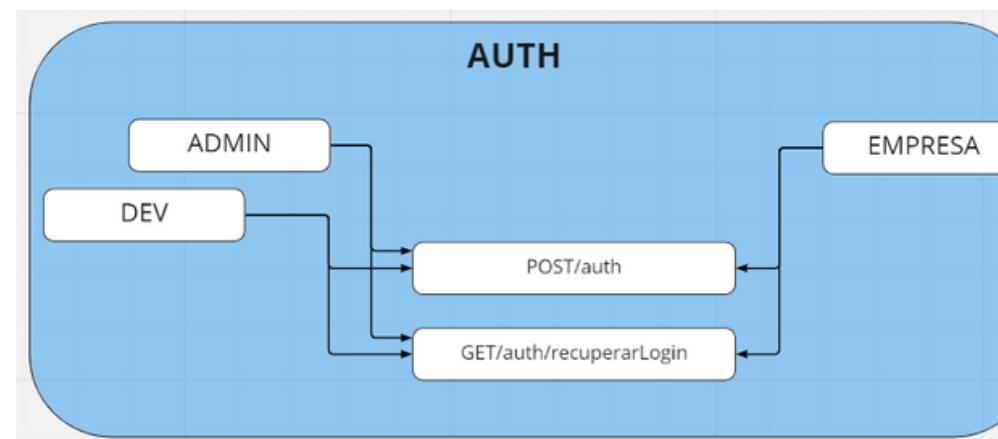
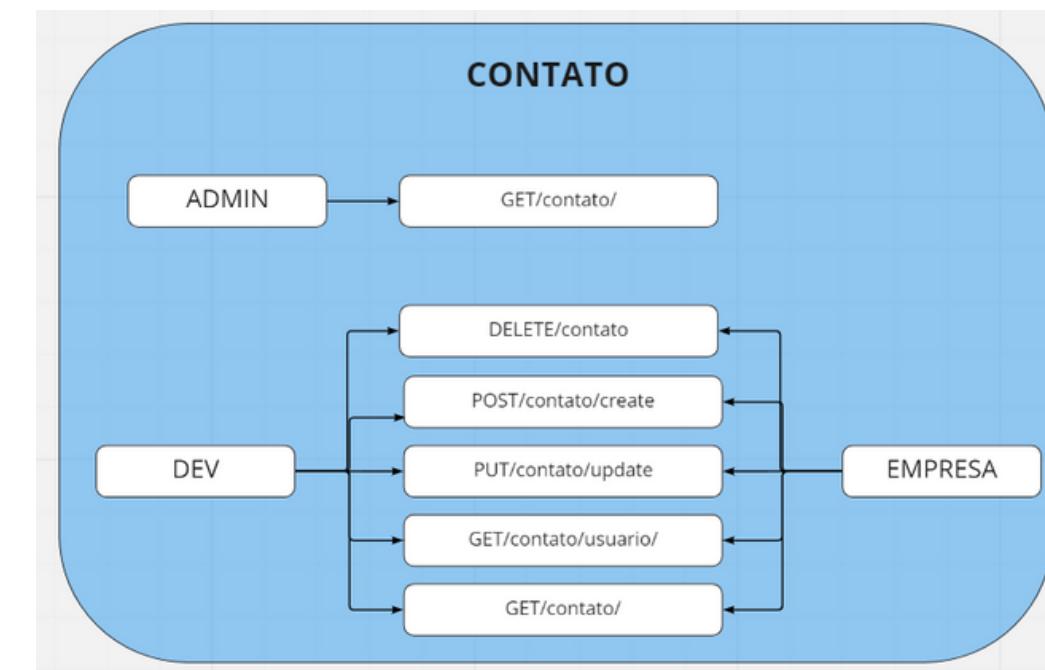
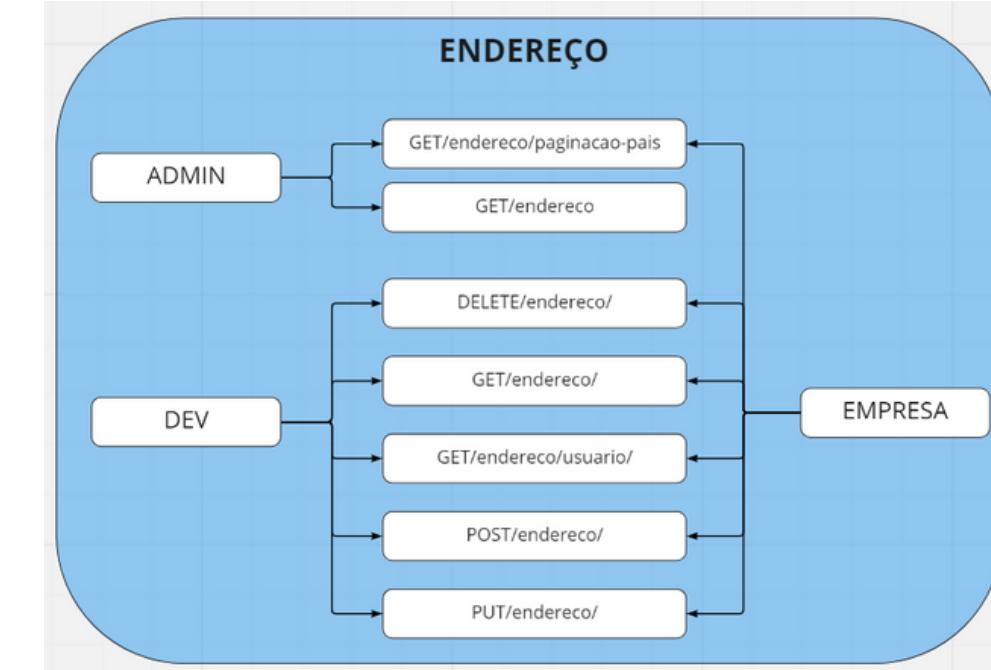
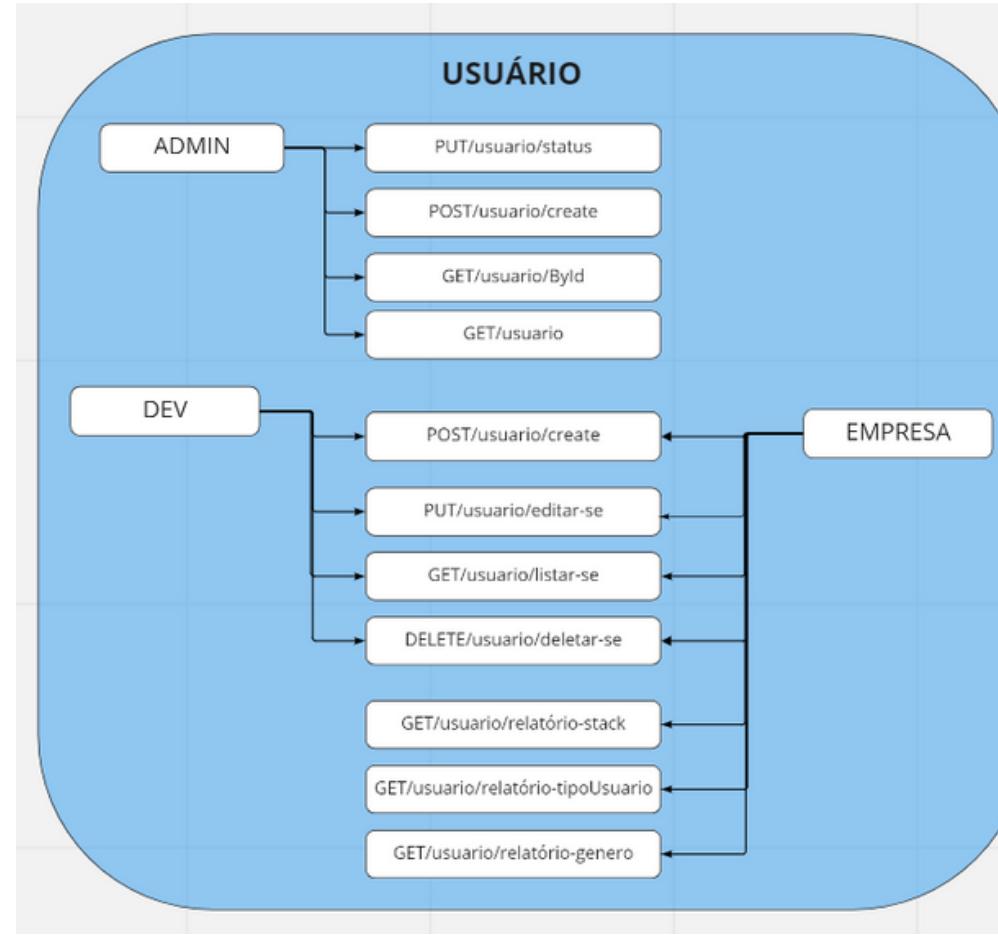
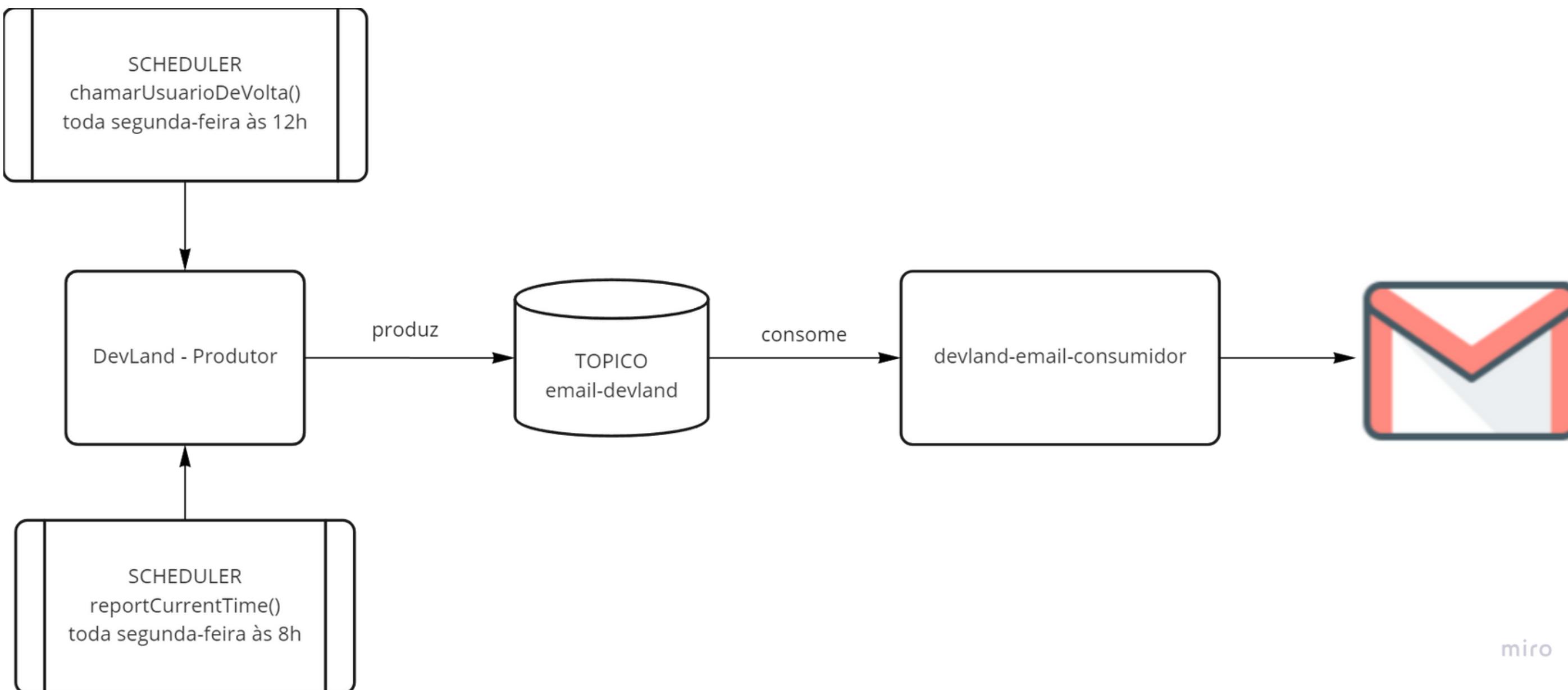


Diagrama de Arquitetura



miro



Apache Kafka



Usamos o serviço de mensageria Apache Kafka para intermediar a API principal DevLand ao serviço de envio de email (agora externo ao projeto principal)





Scheduler

Usamos a anotação
@Scheduled para tratar do
agendamento da execução
métodos específicos



```
@Scheduled(cron = "0 0 12 ? * MON")
public void chamarUsuarioDeVolta() throws JsonProcessingException {
```

```
@Scheduled(cron = "0 0 8 * * MON" )
public void reportCurrentTime() {
```



Método para capturar usuários com campos nulos e encaminhar os mesmos para o Kafka.

Utilização de uma Cron Expression para ocorrer capturas todas as Segundas-Feiras às 8h da manhã.

ForEach para envio a todos usuários com dados nulos.

```
public class VerificaPreenchimentoDeEnderecoContatoTodaSegunda {  
  
    1 usage  
    @Autowired  
    private UsuarioRepository usuarioRepository;  
  
    1 usage  
    @Autowired  
    private ProdutorService produtorService;  
  
    ▲ Cesar +1 *  
    @Scheduled(cron = "0 0 8 * * MON")  
    public void reportCurrentTime() {  
        List<DadosNulosDTO> usuarioDTO = usuarioRepository.verificadorUsuariosComDadosNulos();  
        usuarioDTO.forEach(usuario -> {  
            try {  
                produtorService.enviarMensagemEmail2(usuario, TipoMensagem.CADASTROINCOMPLETO.getTipo());  
                Thread.sleep( millis: 5000);  
            } catch (JsonProcessingException e) {  
                throw new RuntimeException(e);  
            } catch (InterruptedException e) {  
                throw new RuntimeException(e);  
            }  
        });  
    }  
}
```





Query para recuperar Usuários com Endereços ou Contatos Nulos



```
1 usage  ↗ Cesar
@Query(value = " select new br.com.vemser.devlandapi.dto.relatorios.DadosNulosDTO(" +
    " u.nome," +
    " u.email," +
    " c.tipo," +
    " c.numero," +
    " c.descricao," +
    " e.cidade," +
    " e.estado," +
    " e.pais" +
") " +
" from usuario u " +
" full join u.contatos c " +
" full join u.enderecos e " +
" where (c.tipo is null OR c.numero is null OR c.descricao is null OR e.cidade is null OR e.estado is null OR e.pais is null)")
List<DadosNulosDTO> verificadorUsuariosComDadosNulos();
```

Full join nas tabelas Contato e Endereço para localização de campos nulos e posterior produção de registro dentro do Kafka

Método para escutar registros presentes no Kafka e disparo de Emails



```
± Cesar +1
@Service
@RequiredArgsConstructor
@Slf4j
public class ConsumerEmailService {

    1 usage
    @Autowired
    private ObjectMapper objectMapper;

    4 usages
    private final freemarker.template.Configuration fmConfiguration;

    5 usages
    @Value("${spring.mail.username}")
    private String from;

    2 usages
    private final JavaMailSender emailSender;

    ± Cesar
    @KafkaListener(
        topics = "${kafka.email-devland}",
        groupId = "${kafka.user}",
        containerFactory = "listenerContainerFactory",
        clientIdPrefix = "email-devland"
    )
    public void consumir(@Payload String mensagem,
                         @Header(KafkaHeaders.RECEIVED_MESSAGE_KEY) String key,
                         @Header(KafkaHeaders.OFFSET) Long offset) throws JsonProcessingException {

        EmailDto emailDto = objectMapper.readValue(mensagem, EmailDto.class);

        sendEmailUsuario(emailDto);
    }
}
```

O object mapper lê o valor presente no Kafka e transforma este valor em um DTO para posterior disparo de e-mail



Método para disparo de e-mail com seus respectivos templates



```
1 usage  ━ Cesar +1 *
public void sendEmailUsuario(EmailDto emailDto) {
    try {
        MimeMessage mimeMessage = emailSender.createMimeMessage();
        MimeMessageHelper mimeMessageHelper = new MimeMessageHelper(mimeMessage, multipart: true);

        mimeMessageHelper.setFrom(from);
        mimeMessageHelper.setTo(emailDto.getEmail());
        if (emailDto.getTipoMensagem().getTipo().equals(TipoMensagem.CREATE.getTipo())) {
            mimeMessageHelper.setSubject("Olá, " + emailDto.getNome() + "! Seja bem-vindo(a) na DevLand!");
        } else if (emailDto.getTipoMensagem().getTipo().equals(TipoMensagem.UPDATE.getTipo())) {
            mimeMessageHelper.setSubject(emailDto.getNome() + ", seus dados foram atualizados!");
        } else if (emailDto.getTipoMensagem().getTipo().equals(TipoMensagem.DELETE.getTipo())) {
            mimeMessageHelper.setSubject(emailDto.getNome() + ", sentiremos sua falta na DevLand!");
        } else if (emailDto.getTipoMensagem().getTipo().equals(TipoMensagem.CADASTROINCOMPLETO.getTipo())) {
            mimeMessageHelper.setSubject(emailDto.getNome() + ", complete seu cadastro e seja encontrado na DevLand!");
        } else if (emailDto.getTipoMensagem().getTipo().equals(TipoMensagem.MISSYOU.getTipo())) {
            mimeMessageHelper.setSubject(emailDto.getNome() + ", percebemos sua ausencia, voce faz falta na DevLand!");
        } else {
            throw new RegraDeNegocioException("Falha no envio de e-mail");
        }
        mimeMessageHelper.setText(getContentFromTemplatePessoa(emailDto), html: true);
        emailSender.send(mimeMessageHelper.getMimeMessage());
    } catch (RegraDeNegocioException | MessagingException | IOException | TemplateException e) {
        log.info("Erro no envio de email");
    }
}
```

É utilizado um **if-else** para verificar o tipo de mensagem recebida e efetuar o disparo de email correto.

Método para construção de templates personalizados



```
1 usage  ↗ Cesar +1
public String getContentFromTemplatePessoa(EmailDto emailDto) throws IOException, TemplateException {
    Map<String, Object> dados = new HashMap<>();

    Template template;

    if (emailDto.getTipoMensagem().getTipo().equals(TipoMensagem.CREATE.getTipo())) {
        dados.put("nome", "Olá, " + emailDto.getNome() + "! Seja bem-vindo(a) na DevLand!");
        dados.put("mensagem", "Seu cadastro foi realizado com sucesso, seu código de identificação é " + emailDto.getIdUsuario());
        dados.put("email", "Qualquer dúvida, entre em contato com o suporte pelo e-mail " + from);
        template = fmConfiguration.getTemplate( name: "email-template.html");

    } else if (emailDto.getTipoMensagem().getTipo().equals(TipoMensagem.UPDATE.getTipo())) {
        dados.put("nome", "Olá, " + emailDto.getNome() + "! Seus dados foram atualizados!");
        dados.put("mensagem", "Seus dados foram atualizados com sucesso e já podem ser encontrados por empresas e talentos.");
        dados.put("email", "Qualquer dúvida, entre em contato com o suporte pelo e-mail " + from);
        template = fmConfiguration.getTemplate( name: "email-template.html");

    } else if (emailDto.getTipoMensagem().getTipo().equals(TipoMensagem.CADASTROINCOMPLETO.getTipo())) {
        dados.put("nome", "Olá, " + emailDto.getNome() + "!");
        dados.put("mensagem", "Notamos que faltam alguns dados do seu cadastro para ficar completo e assim poder ser encontrado por empresas com mais compatibilidade com seu perfil.");
        dados.put("email", "Qualquer dúvida, entre em contato com o suporte pelo e-mail " + from);
        template = fmConfiguration.getTemplate( name: "email-template.html");

    } else {
        dados.put("nome", "Olá, " + emailDto.getNome() + "! Sentiremos sua falta na DevLand");
        dados.put("mensagem", "Seu cadastro foi removido da nossa rede, mas você pode voltar quando quiser!");
        dados.put("email", "Qualquer dúvida, entre em contato com o suporte pelo e-mail " + from);
        template = fmConfiguration.getTemplate( name: "email-template.html");
    }
    String html = FreeMarkerTemplateUtils.processTemplateIntoString(template, dados);
    return html;
}
```

Na construção do template é recuperado os dados do usuário para elaboração de templates únicos e com aspecto clean. Desta forma, existindo apenas um arquivo .html na pasta templates.



Agendamento para verificação semanal dos usuários que não se logaram por mais de 8 dias e envio de email (usando o serviço de email) avisando-o da sua ausência.

```
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
```

```
    // ===== CONFERIR ACESSOS USUARIOS =====
    * Rafael Ferreira *
    @Scheduled(cron = "0 0 12 ? * MON")
    public void chamarUsuarioDeVolta() throws JsonProcessingException {
        List<LogAcessoRetornoDTO> logAcessoRetornoDTOS = logAcessoRepository.retornarAggregation();
        for (LogAcessoRetornoDTO log: logAcessoRetornoDTOS)
        {
            UsuarioEntity usuarioEntity = objectMapper.convertValue(log, UsuarioEntity.class);

            if(log.getData().isBefore(LocalDateTime.now().minusDays(8))) {
                produtorService.enviarMensagemEmail(usuarioEntity, TipoMensagem.MISSYOU.getTipo());
                System.out.println("mensagem MISSYOU enviada");
            }
        }
    }
```



Agendamento para verificação semanal dos usuários que não se logaram por mais de 8 dias e envio de email (usando o serviço de email) avisando-o da sua ausência.

```
4 usages  ↳ Rafael Ferreira *
public interface LogAcessoRepository extends MongoRepository<LogAcessoDTO, Integer> {

    1 usage  ↳ Rafael Ferreira *
    @Aggregation(pipeline = "{\n        \"$group: {\n            \"_id: '$email',\n            \"nome: {\n                \"$first: '$nome'\n            },\n            \"data: {\n                \"$max: '$data'\n            }\n        }\n    }")
    List<LogAcessoRetornoDTO> retornarAggregation();
}
```

Usamos um aggregate para retornar do MongoDB os dados dos ultimos acessos agrupados por email

Muito Obrigado!

#Dev
#Land

