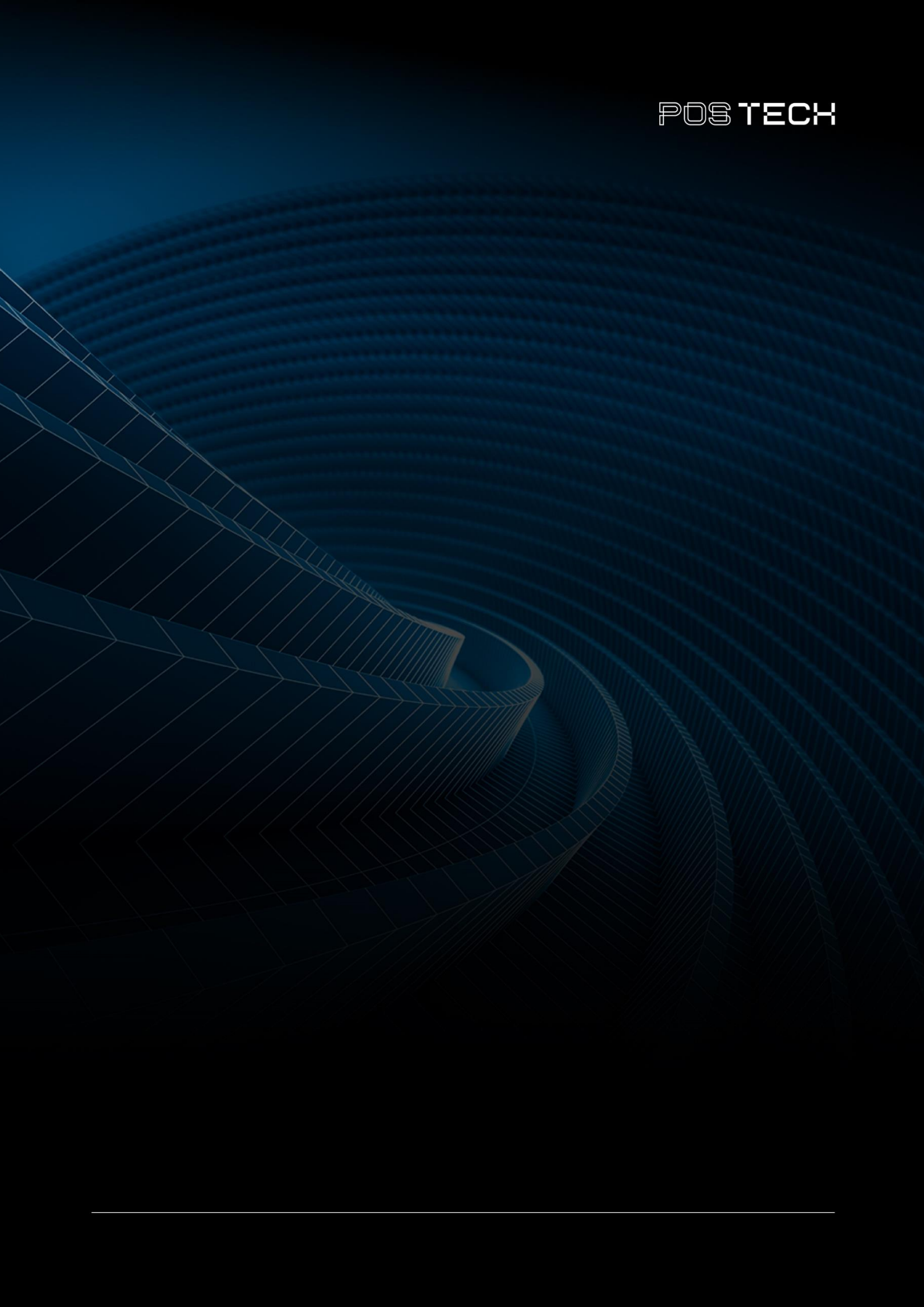
POS TECH



Tech Challenge

Tech Challenge é o projeto da fase que engloba os conhecimentos obtidos em todas as disciplinas dela. Esta é uma atividade que, a princípio, deve ser desenvolvida em grupo. É importante atentar-se ao prazo de entrega, uma vez que essa atividade é obrigatória e vale 90% da nota de todas as disciplinas da fase.

O problema

O MVP da FIAP Cloud Games (FCG) foi um sucesso! A plataforma já consegue cadastrar usuários e gerenciar jogos. Agora, com a visão de expandir o negócio, a FCG precisa implementar um fluxo de compra de jogos e um sistema de notificações para melhorar a experiência do usuário.

A arquitetura monolítica atual está se mostrando um gargalo. Adicionar funcionalidades complexas como processamento de pagamentos e envio de e-mails em um único serviço aumentaria o acoplamento, dificultaria a manutenção e impediria que cada parte do sistema escalasse de forma independente. Para suportar o crescimento, é essencial evoluir a arquitetura.

O Desafio

O objetivo desta fase é refatorar a aplicação monolítica em uma **arquitetura de microsserviços** orientada a eventos. Vocês irão decompor a aplicação original e criar novas funcionalidades, resultando em quatro serviços independentes: Usuários, Catálogo, Pagamentos e Notificações.

Cada microsserviço deverá residir em seu próprio repositório Git, garantindo total autonomia de código e ciclo de vida. Para garantir que esses serviços sejam resilientes e escaláveis, eles deverão ser containerizados com Docker e a implantação orquestrada em um ambiente Kubernetes, utilizando as melhores práticas para gerenciamento de configurações e segredos. A comunicação entre eles será feita de forma assíncrona, utilizando um sistema de Mensageria.

Este desafio foi estruturado para aplicar os conhecimentos adquiridos nas disciplinas de Microsserviços, Docker, Kubernetes e Mensageria.

Tech Challenge

Funcionalidades Obrigatórias

1. Arquitetura de Microsserviços e Repositórios:

- Desenvolva quatro microsserviços distintos.
- Cada microsserviço deve residir em seu próprio repositório Git, garantindo total autonomia e separação de código.
  - **Microsserviço de Usuários (UsersAPI):** responsável por cadastro, autenticação (geração de token JWT) e autorização de usuários.
  - **Microsserviço de Catálogo (CatalogAPI):** responsável pelo CRUD de jogos e por iniciar o fluxo de compra.
  - **Microsserviço de Pagamentos (PaymentsAPI):** responsável por processar (simular) o pagamento de uma compra de jogo.
  - **Microsserviço de Notificações (NotificationsAPI):** responsável por enviar (simular, logando no console) e-mails de boas-vindas e de confirmação de compra.

2. Fluxos Orientados a Eventos (Mensageria com RabbitMQ ou Kafka):

Fluxo de Cadastro de Usuário:

1) O **UsersAPI** cria um novo usuário e publica um evento UserCreatedEvent.
2) O **NotificationsAPI** consome este evento e "envia" um e-mail de boas-vindas.

Fluxo de Compra de Jogo:

1) O **CatalogAPI** recebe uma requisição para adicionar um jogo à biblioteca de um usuário e publica um evento OrderPlacedEvent, contendo UserId, GameId e Price.
2) O **PaymentsAPI** consome o OrderPlacedEvent, processa o pagamento e publica o resultado em um novo evento: PaymentProcessedEvent (com status Approved ou Rejected).

3) O **CatalogAPI** consome o PaymentProcessedEvent. Se o pagamento foi Approved, ele adiciona o jogo à biblioteca do usuário.
4) O **NotificationsAPI** também consome o PaymentProcessedEvent. Se foi Approved, "envia" um e-mail de confirmação da compra.

3. Orquestração Local e de Implantação:

- Cada um dos quatro repositórios de microsserviço deve conter seu próprio Dockerfile.
- **(Opcional)** Se achar melhor, você pode criar um quinto repositório específico para orquestração, colocando a responsabilidade de infraestrutura nele.

4. Orquestração com Kubernetes:

- Os manifestos devem estar em uma pasta **/k8s** na raiz do projeto de cada repositório.
- É **obrigatório** o uso de Deployments para gerenciar os Pods. A criação de Pods de forma isolada não será aceita.
- É **obrigatório** o uso de ConfigMaps para armazenar configurações não sensíveis (ex.: nome de filas/tópicos, URLs de serviços).
- É **obrigatório** o uso de Secrets para armazenar dados sensíveis (ex.: connection strings de banco de dados, chaves de API).

Requisitos Técnicos

Desenvolvimento:

- Utilizar **.NET 8 ou superior** para todos os microsserviços.
- Cada serviço deve ter sua própria solução (.sln) ou projeto (.csproj) dentro de seu respectivo repositório.

Containerização com Docker:

- Os Dockerfiles devem ser otimizados para produção, utilizando multi-stage builds.

- O docker-compose.yml no repositório de orquestração ou em um local de sua preferência deve permitir que a aplicação completa seja executável com docker-compose up.

Comunicação com Mensageria:

- Utilize uma biblioteca .NET robusta para interagir com o broker, como MassTransit ou a biblioteca oficial do RabbitMQ.Client/Confluent.Kafka.

Orquestração com Kubernetes:

- Os serviços devem se comunicar dentro do cluster utilizando seus nomes de Service do Kubernetes (ex: http://payments-api:80).
- O deploy deve ser testado em um cluster Kubernetes local (Kind, Minikube, k3d, Docker Desktop Kubernetes).

Entregáveis da Fase 2

- **Vídeo de até 20 minutos demonstrando todos os requisitos.**
  - Apresentar a estrutura dos diferentes repositórios.
  - Demonstrar a execução dos serviços com docker-compose a partir do repositório de orquestração.
  - Executar o fluxo de cadastro e o fluxo de compra completos.
  - Mostrar os arquivos de manifesto no repositório de orquestração (Deployment, Service, ConfigMap, Secret).
  - Demonstrar o deploy da aplicação no cluster Kubernetes local (kubectl apply -f .) e verificar se todos os Pods estão rodando (kubectl get pods).
- **Códigos-fonte nos repositórios, incluindo:**
  - Os links para os repositórios individuais de cada microsserviço.
  - O link para o repositório de orquestração (**só válido para quem criou o quinto repositório**).

  - Cada repositório de microsserviço deve ter um README.md explicando sua finalidade e variáveis de ambiente.
  - O repositório de orquestração, ou onde você colocar os arquivos, deve ter um README.md principal, com instruções claras de como executar o projeto com Docker e como fazer o deploy no Kubernetes.

- **Relatório de entrega (PDF ou TXT) – esse arquivo deve ser postado na data da entrega e conter:**
  - Nome do grupo.
  - Participantes e usernames no Discord.
  - Link da documentação.
  - Link do(s) repositório(s).
  - Link do vídeo salvo no Youtube ou lugar de sua preferência.

Lembramos que, caso você tenha qualquer dúvida, é só nos chamar no Discord!


PDS TECH