1.Trabalho Final – Projeto Fullstack com Spring + RabbitMQ

Entrega: 26 de novembro (até 23h59)

Vocês vão desenvolver um **sistema fullstack completo** que resolva algum tipo de problema ou necessidade.
"Problema" aqui não precisa ser algo ruim: pode ser algo criativo, mirabolante, divertido, desde que faça sentido e tenha uma proposta clara.

Formação dos grupos

- Pode ser:
    - **Individual**
    - **Dupla**
    - **Trio** (apenas com justificativa: o trio precisa explicar por que o escopo do projeto é maior e precisa de 3 pessoas)
- Independente do tamanho do grupo, **a complexidade do projeto precisa ser compatível**:
    - Individual → escopo menor, mas bem feito
    - Dupla/Trio → escopo maior, mais recursos, mais telas, mais regras de negócio

2. Requisitos Técnicos Obrigatórios

Back-end

- Desenvolvido em **Spring (Spring Boot)**.
- Deve expor uma **API REST** com, ***no mínimo***, as operações:
    - `GET / POST / PUT / DELETE`

- Banco de dados relacional (MySQL, Postgres, etc.).
- Camadas minimamente organizadas (controller, service, repository, entity/model).

Mensageria (RabbitMQ)

- Uso de **RabbitMQ** obrigatório.
- Pelo menos **2 serviços** separados (pode ser 2 projetos Spring Boot diferentes):
    - Por ***exemplo***:
        - `user-service` produz mensagem quando usuário se registra.
        - `email-service` ou `notificacao-service` consome a mensagem e faz algo (log, simulação de envio de e-mail, etc.).
- Deve haver pelo menos **1 fluxo de negócio real** passando pela fila.
    - Exemplo: “Pedido criado” → envia mensagem → serviço de estoque ou notificação consome.

Front-end

- **Tecnologia à escolha do aluno/grupo**:
    - Pode ser Angular, React, Vue, Thymeleaf, HTML+JS puro… (combinem com o professor se quiserem algo muito diferente).
- O front-end deve:
    - Consumir a API do back-end.
    - Permitir **criar, listar, atualizar e remover** pelo menos uma entidade principal (CRUD).
    - Ter uma interface minimamente organizada (não precisa ser um design perfeito, mas nada completamente cru).

Docker (obrigatório)

Cada grupo deve **containerizar o projeto** usando Docker.

Mínimo esperado:

1. **Back-end**
    - Cada serviço de back-end (ex.: `user-service`, `email-service`) deve ter um **Dockerfile**.
    - Deve ser possível subir o serviço via Docker (`docker build` / `docker run` ou `docker-compose`).
2. **Banco e RabbitMQ**
    - **Pode** ser via Docker também (recomendado), usando `docker-compose.yml` com:
        - Opcionalmente containerizar o RabbitMQ (positivo extra)
        - Banco de dados
        - Serviços de back-end
3. **Front-end**
    - Opcionalmente containerizado também (positivo extra).
    - Se não for containerizado, pelo menos documentar no README como rodar.
4. **Rodando tudo**
    - Ideal: um `docker-compose.yml` que suba, pelo menos:
        - Banco
        - Os 2 serviços principais
    - No README deve ter:
        - Como subir com Docker/Docker Compose.

        - Quais portas cada serviço usa.

3. Criatividade (vai contar nota!)

Criatividade **é critério de avaliação**. Vocês podem:

- Inventar sistemas diferentes do **“CRUDzinho padrão”**.
- Criar temas mirabolantes **(OPCIONAL)**:
    - Sistema de gerenciamento de missões de RPG.
    - Sistema de reservas de salas temáticas, estúdios, instrumentos, etc.
    - Plataforma de adoção de pets mágicos.
    - Organizador de campanhas de D&D / mesas online.
    - Sistema de controle de filas em eventos, baladas, workshops, etc.
    - Simular no front uma maquina de abastecimento de energia elétrica e o back controlar.
    - Usar o front para simular/imitar alguma máquina, processo ou robô.
- O importante: **a ideia precisa fazer sentido**, ter regras mínimas de negócio e não ser só uma tela jogada com 4 campos aleatórios.

4. Entregáveis

Até o dia **26/11**, cada grupo deve entregar:

1. **Código fonte**
    - Projeto(s) de back-end (Spring + Rabbit).
    - Projeto de front-end.
    - Preferencialmente em um repositório (ou mais) no **GitHub**.

2. **README.md** no repositório com:
    - Nome do projeto.
    - Integrantes do grupo.
    - Descrição do sistema (o que resolve, público alvo).
    - Tecnologias usadas.
    - Como rodar o back-end.
    - Como rodar o front-end.
    - Explicação simples de onde entra a mensageria (quem produz, quem consome).
3. **Pequena apresentação (defesa)**
    - Cada grupo terá alguns minutos para:
        - Explicar a ideia.
        - Mostrar o sistema rodando.
        - Mostrar onde está o fluxo de mensageria.
    - Todos do grupo devem falar pelo menos um pouco.

(Se você quiser, pode depois definir tempo certinho tipo 5–10 min, mas isso você ajusta em sala.)

5. Sugestões de temas

Alguns exemplos que encaixam bem com Spring + Rabbit + front:

- **Sistema de Pedidos**
    - Serviço de pedido → manda mensagem “PedidoCriado”
    - Serviço de notificação/estoque → consome e registra/atualiza algo.

- **Plataforma de Cursos/Workshops**
    - Quando um aluno se inscreve em um curso, um evento é disparado e outro serviço “simula” envio de e-mail / geração de certificado.
- **Organizador de Eventos / Festas / Shows**
    - Criação de evento → manda mensagem para um serviço de “divulgação” / “ingresso” / “fila de espera”.
- **Sistema de Feedback / Avaliações**
    - Ao enviar um feedback, é disparado um evento que outro serviço consome (ex.: gera estatísticas, log, etc.).
- **Qualquer coisa temática/criativa**
    - Exemplo: app de gerenciamento de heróis, vilões, squads, missões; quando missão é criada/concluída, manda evento para outro serviço.
- **Simulação**
    - Front é um painel que mostra um robozinho de entrega numa cidade (pode ser só ícones num mapa simples ou grid).
    - Simular uma mini fábrica: uma esteira com caixinhas e um braço robótico que pega e classifica.
    - Central de comando de drones com uma interface tipo “joguinho” onde você vê vários drones numa região.
    - Estação de recarga: Simular várias máquinas/robôs que precisam de recarga e manutenção.
    - Uma “Máquina de Realidade Alternativa” ou “Máquina de Misturar Poções”, “Máquina de Portais”, etc

6. Critérios de Avaliação

Você pode mostrar pros alunos também, se quiser já deixar transparente:

1. **Funcionalidade**
    - CRUD e ***OUTRAS FUNÇÕES*** funcionando.
    - Fluxos principais sem erro crítico.
2. **Back-end / Arquitetura**
    - Organização de camadas.

- Uso correto do Spring (controllers, services, repositories).

3. **Mensageria com RabbitMQ**
    - Pelo menos 2 serviços.
    - Produção e consumo de mensagens em um fluxo real do sistema.
4. **Front-end**
    - Integração real com a API.
    - Interface utilizável.
5. **Criatividade / Apresentação**
    - Ideia do projeto, clareza na explicação, defesa.