IS – 1º trabalho prático individual + Trabalho de Grupo

Trabalho Prático Individual: Desenvolvimento de Serviços Web Multitecnologia

Objetivo:

Desenvolver um sistema cliente-servidor demonstrando a implementação e integração de múltiplas tecnologias de serviços web, incluindo funcionalidades de exportação e importação de dados nos formatos XML e JSON.

Requisitos:

1. **Servidor (Ubuntu):**
    - Implementar um servidor com as seguintes tecnologias:
        - **SOAP** (com validação XSD).
        - **REST** (com validação JSON Schema e consultas JSONPath).
        - **GraphQL** (queries e mutations).
        - **gRPC** (serviços unários e streaming).
    - Linguagem: Python ou Node.js.
    - Implementação:
        - Único processo ou múltiplos containers Docker.
        - Caso utilize Docker:
            - Criar Dockerfiles para cada serviço.
            - Utilizar docker-compose.yml para orquestração.
            - Compartilhar dados entre serviços via volumes Docker (JSON ou XML).
    - Armazenar dados persistentemente em arquivos JSON ou XML (sem uso de SGBD).
    - Disponibilizar endpoints para operações CRUD num recurso à escolha (ex.: utilizadores, produtos).
    - Disponibilizar funcionalidades específicas para exportação e importação de dados em XML e JSON.
    - Garantir testabilidade de todos os endpoints através do Postman.
2. **Cliente (Python ou JavaScript):**
    - Desenvolver um cliente interagindo com o servidor através de todas as tecnologias mencionadas.
    - - Demonstrar claramente funcionalidades de exportação e importação de dados em XML e JSON.
    - Sugestões de implementação:

        - **Python (desktop):** Utilizar requests, zeep, grpc, etc.
        - **JavaScript (web):** Node.js + Express, com fetch, Apollo Client, etc.
        - Outra configuração, mediante aprovação prévia do professor.

3. **Armazenamento e Validação de Dados:**
    - SOAP: XML validado por XSD.
    - REST e GraphQL: JSON validado por JSON Schema e consultas via JSONPath.
    - gRPC: Validação diretamente no código servidor.
4. **Tema:**
    - Livre escolha (ex.: sistema de gestão de tarefas, catálogo de produtos, etc).
5. **Entrega:**
    - Criar um repositório GitHub com:
        - Código fonte do servidor e cliente (bem estruturado e documentado).
        - Dockerfiles e docker-compose.yml (quando aplicável).
        - Documentação:
            - Descrição detalhada dos endpoints/serviços.
            - README.md com instruções claras para execução, exemplos de chamadas (Postman) e esquemas de validação.
            - Vídeo de demonstração (até 8 minutos), para todos os alunos.
    - Estrutura sugerida do repositório:
        - /servidor
        - /cliente
        - /documentacao
        - docker-compose.yml (quando aplicável)
    - **Acesso ao Repositório GitHub:**
        - Adicionar o professor como **colaborador com permissões de leitura** no repositório GitHub assim que possível, para acompanhamento do progresso através dos commits.
        - Realizar commits frequentes com mensagens claras e informativas.
6. **Apresentação:**
    - Demonstração presencial (8 minutos) para 15 alunos selecionados aleatoriamente.

    - Restantes alunos deverão submeter o vídeo de demonstração no repositório.

7. **Avaliação:**
    - **Funcionalidade (60%)**: Implementação correta e completa das tecnologias.
    - **Organização (30%)**: Estrutura e documentação clara e eficiente do código.
    - **Apresentação (10%)**: Clareza, objetividade e qualidade da demonstração.

Observações Adicionais:

- WebSockets, armazenamento, segurança em APIs e outros temas da UC serão abordados no trabalho em grupo, a realizar.
- Escolher apenas um formato principal de dados persistentes (JSON ou XML), garantindo capacidade de conversão entre formatos (por exemplo, para fins de exportação e importação).
- Enviar o link do repositório GitHub por email o mais cedo possível.
- O trabalho deverá estar concluído e depositado até ao final do dia 17 de abril.
- As apresentações presenciais serão realizadas no dia 22 de abril.

IS – 1º trabalho prático individual + Trabalho de Grupo

Trabalho Prático de Grupo — Integração de Sistemas

Objetivo Geral

Desenvolver uma aplicação cliente-servidor distribuída, baseada num dos projetos individuais, incorporando múltiplas tecnologias de integração e comunicação entre **múltiplos servidores** (fornecidos pelo professor). A solução deve incluir **serviços web multitecnologia**, **autenticação avançada**, **bases de dados NoSQL**, **mensageria assíncrona**, e **WebSockets**.

1. **Melhoria do Projeto Base**

- Corrigir erros do trabalho individual original
- Atualizar documentação e testes

2. **Arquitetura Distribuída**

- A aplicação deve funcionar distribuída entre **2 ou 3 servidores Ubuntu** atribuídos
- Cada servidor pode executar serviços distintos:
    - Exemplo: SOAP num, REST noutro, gRPC noutro
- Comunicação entre serviços deve ocorrer por:
    - **HTTP (REST/SOAP)**
    - **gRPC**
    - **Mensageria (RabbitMQ ou Kafka)**

3. **Serviços Web obrigatórios**

- **SOAP** (com XSD)
- **REST** (com JSON Schema e JSONPath)
- **GraphQL**
- **gRPC**
- **WebSockets** (para notificações em tempo real ou status)

4. **Cliente funcional (hospedado em servidor)**

- O cliente deve ser obrigatoriamente executado a partir de **um dos servidores Ubuntu atribuídos ao grupo**.

  - Pode ser uma **aplicação web** servida por um dos servidores (ex: http://ip-do-servidor:3000)
  - Ou uma **aplicação desktop** que interaja com os outros servidores do grupo a partir desse ponto de execução
- O cliente deve:
  - Consumir serviços SOAP, REST, GraphQL, gRPC
  - Realizar autenticação e autorizações com JWT/OAuth2/OpenID
  - Comunicar via WebSockets com o servidor correspondente
  - Permitir a exportação/importação de dados em JSON/XML
  - Reagir a notificações em tempo real via WebSockets

5. Autenticação e Autorização

- Implementar **OAuth 2.0 / OpenID Connect**
- Uso de **JWTs** para autenticação entre serviços
- Controlo de acesso baseado em roles

6. Base de Dados NoSQL

- MongoDB ou Firebase (preferencialmente com Docker)
- Armazenamento persistente de dados transversais

7. Filas de Mensagens e Comunicação Assíncrona

- RabbitMQ, Apache Kafka ou MQTT
- Usar para ações como logs, notificações ou eventos da aplicação
- Mensagens devem conter contexto de quem gerou a ação (via JWT)

8. Comunicação em Tempo Real

- Implementar **WebSockets** entre o cliente e um dos servidores
- Ex: notificações de eventos, mudanças de estado, logs ao vivo

9. Tecnologias sugeridas

Componente	Opções
Backend	Node.js / Python
NoSQL	MongoDB / Firebase
Mensageria	RabbitMQ / Kafka
Autenticação	JWT, OAuth 2.0, OIDC
WebSockets	socket.io / ws / FastAPI WebSocket
Orquestração	Docker, docker-compose
Comunicação	HTTP, gRPC, AMQP, WS

10. Entrega

- GitHub com:
  - Código cliente e servidor(es)
  - Dockerfiles para cada componente
  - docker-compose.yml (por servidor ou global)
  - Documentação com:
    - Arquitetura distribuída
    - Endpoints
    - Autenticação
    - Integração entre servidores
    - Testes (ex: Postman, unitários)
  - Vídeo de demonstração (máx. 10 min)

11. Avaliação

Critério	Peso	
Funcionalidade e serviços web	30	Avalia se todos os serviços especificados (REST, SOAP, GraphQL, gRPC) estão corretamente implementados e funcionam de forma coerente, com validação

Critério	Peso	
		adequada (XSD/JSON Schema). Endpoints devem permitir operações CRUD e exportação/importação de dados.
Execução distribuída e integração	20	Verifica se a solução está corretamente distribuída por **2 ou 3 servidores**, com serviços a comunicar entre si (via HTTP, gRPC, RabbitMQ, etc). Espera-se que cada servidor tenha responsabilidade distinta e interaja com os outros.
Autenticação e segurança	10	Avalia a implementação de **OAuth 2.0 / OpenID Connect e JWT**. O sistema deve proteger rotas, validar tokens, e aplicar controlo de acessos baseado em roles (ex: utilizador comum vs. admin).
Mensageria assíncrona	10	Verifica o uso efetivo de **filas de mensagens** com RabbitMQ, Kafka ou MQTT. As mensagens devem ser enviadas e consumidas após ações específicas (ex: criação de dados, erros, logs). Deve existir pelo menos um **consumidor funcional**.
WebSockets	10	Avalia a presença de comunicação em tempo real, como **notificações ao cliente**, atualizações de estado ou logs ao vivo. Espera-se um canal WebSocket funcional com dados transmitidos a partir de eventos no servidor.
Cliente funcional	10	O cliente (desktop ou web) deve estar **hospedado num dos servidores**, interagir com todos os serviços, apresentar resultados ao utilizador, lidar com autenticação e receber mensagens em tempo real.
Documentação e organização	5	Avalia a clareza e completude da documentação: README.md, explicações dos serviços, instruções de execução, esquemas de autenticação, exemplos de chamadas (Postman). A organização do repositório também conta.
Apresentação (vídeo ou presencial)	5	Avalia se o grupo explica bem o projeto, mostra o sistema a funcionar, destaca as integrações entre servidores, e demonstra domínio do que foi feito. Clareza, objetividade e preparação são fundamentais.

Para cada critério, aplicar a seguinte escala:

Pontuação	Descrição
100%	Implementado e funcional, com todos os requisitos cumpridos
75%	Funciona, mas com pequenas falhas ou incompletudes
50%	Funcionalidade parcial ou incompleta
25%	Presença mínima ou incorreta

Pontuação	Descrição
0%	Ausente

12. Constituição dos Grupos

- O trabalho deve ser realizado em **grupos de 3 alunos**.
- Cada grupo deve **selecionar um dos trabalhos individuais já desenvolvidos** como ponto de partida.

13. Observações Adicionais:

- Integração em Cloud e Automação serão abordados em termos de avaliação no teste, a realizar.
- Enviar o link do repositório GitHub por email o mais cedo possível.
- O trabalho deverá estar concluído e depositado até ao final do dia 2 de junho.
- As apresentações presenciais serão realizadas nos dias 3 e 5 de junho.