Enunciado do trabalho prático n.2

Integração de Sistemas de Informação

Ano letivo 2023/2024 – versão 1

Objetivo

O presente trabalho prático 2 (TP2) tem como objetivo desenvolver as capacidades dos estudantes na integração de diferentes serviços através da escrita de APIs, importação de dados e integração entre bases de dados e outros serviços. Leia atentamente todo o enunciado e contacte os docentes da disciplina em caso de dúvidas.

- Cada **grupo de trabalho** deverá ser **composto por os mesmos alunos que realizaram o TP1.** Não será permitida a formação de novos grupos e apenas poderão realizar o trabalho os grupos que submeteram o TP1.
- A submissão dos novos trabalhos será realizada via Moodle em formulário a indicar para o efeito.

Regras

- Para o desenvolvimento dos trabalhos propostos, serão utilizadas várias linguagens de programação, incluindo Python, TypeScript/JavaScript entre outras.
    - Alguns dos serviços a desenvolver não tem uma linguagem pré-definida. Nestes casos, os estudantes poderão optar por Golang ou Elixir: são disponibilizados exemplos de imagens Docker no repositório para ambas as linguagens.
- Os pormenores de implementação que se devem à interpretação dos enunciados por parte dos grupos de alunos deverão ser descritos no relatório com detalhe e justificação das opções tomadas.
- A implementação de funcionalidades extra não presentes no enunciado será valorizada, desde que estas funcionalidades não modifiquem os requisitos obrigatórios e não reduzam a dificuldade do trabalho. As funcionalidades extra implementadas deverão ser documentadas no relatório.
- A apresentação de trabalhos **não originais e que constituam plágio**, conduzem à imediata atribuição de **nota zero** no trabalho de grupo e a **eventuais processos disciplinares**.

Avaliação e Entrega

- O trabalho prático 2 faz parte da avaliação da Componente Prática da disciplina de **Integração de Sistemas** (correspondendo a 40-50% da nota final).
- A nota é atribuída individualmente aos elementos do grupo segundo a apresentação, visualização e discussão dos elementos entregues e as impressões obtidas pelos docentes acerca do aluno durante o decorrer das aulas de acompanhamento.
- Para aprovação à disciplina, a nota da Componente Prática deverá ter a classificação mínima obrigatória de **10.0 valores**.
- As discussões orais deste trabalho serão realizadas no dia indicado no calendário e no horário previamente definido para o efeito. Os grupos de trabalho devem reservar o dia e hora marcados, na sua agenda. Os trabalhadores-estudantes devem, ao abrigo do respetivo estatuto, solicitar a folha justificativa de exame para a empresa.
- O trabalho prático deverá ser submetido através do Moodle seguindo as instruções lá indicadas. A entrega deverá conter os seguintes elementos:
    - Código fonte;
    - Scripts de bases de dados, imagens Dockers e outros ficheiros de configuração que permitam correr o projeto;

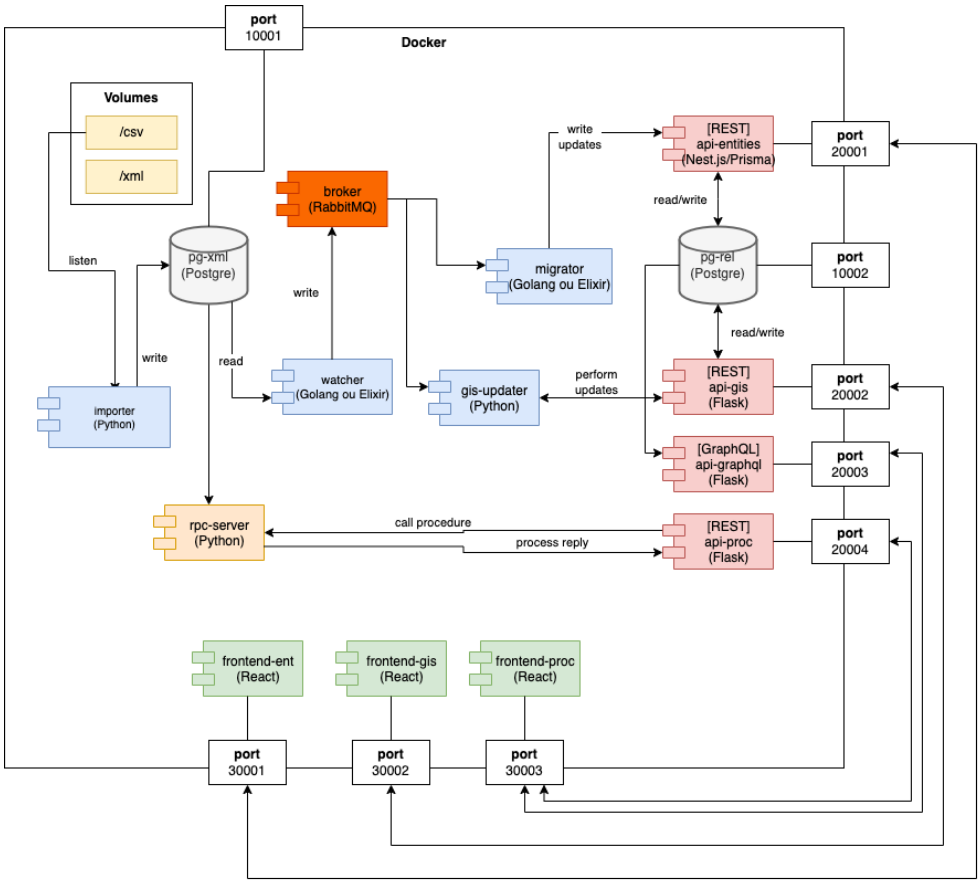
- Slides em PowerPoint / PDF (seguindo estrutura a indicar);
- Vídeo com duração máxima de 5 minutos com instruções de utilização e demonstração de funcionalidades;

Descrição do trabalho

- Para o trabalho é disponibilizada a base de código em um repositório git, que deverá ser utilizada nos trabalhos. O projeto contém uma configuração em Docker Compose com todas as dependências do projeto, assim como instruções de utilização. Alguns dos serviços a desenvolver não se encontram no Docker Compose, sendo que os alunos deverão adicionar estes serviços. Os scripts podem ser editados caso os alunos achem relevante adicionar alguma nova dependência.
- Deverá ser utilizado o mesmo dataset do TP1. Partes desenvolvidas durante o TP1 deverão ser integradas na nova base de código, seguido as indicações da secção **Arquitetura do trabalho** deste documento.
- Todos os módulos descritos na Arquitetura do trabalho deverão ser completados segundo as indicações

Arquitetura do trabalho

Considere o diagrama que demonstra a arquitetura global para o trabalho prático. Abaixo encontra-se igualmente a descrição de todos os módulos a serem implementados.



Módulos da arquitetura

Cada módulo da arquitetura deverá representar um serviço no Docker e, consequentemente, no ficheiro do Docker Compose.

Os módulos da aplicação são os seguintes:

pg-xml

Base de dados do TP1 onde são armazenados os ficheiros XML. Será adicionada uma nova tabela onde são guardadas as conversões de CSV para XML que foram realizadas, sendo que a estrutura da nova tabela é disponibilizada no código base.

Nota: deverá integrar as alterações que realizou no TP1 a fim de manter a funcionalidade de soft-delete já realizada.

pg-rel

Base de dados relacional onde vão ser guardados os dados das entidades importadas. A estrutura da base de dados deverá ter 2-3 tabelas que representam entidades do dataset do TP1, sendo que pelo menos 2 destas entidades deverão estar associadas, nomeadamente uma das entidades deverá ter uma relação 1-N ou N-N com uma outra entidade (exemplos: Equipa / Jogador – uma Equipa tem vários jogadores, um jogador está em apenas uma equipa; Filme / Ator – cada ator pode participar em vários Filmes e cada Filme tem vários atores).

Detalhes de implementação:

1. Os dados GIS, nomeadamente coordenadas geográficas, deverão ser guardados em colunas do tipo **geometry** – a estrutura típica será armazenar um POINT com 2 dimensões: latitude e longitude.
2. Deverão ser preferencialmente utilizados UUIDs (versão 5) em vez de Ids sequenciais em cada entidade (ver https://docs.python.org/3/library/uuid.html). Se alguma das entidades já tiver Ids no XML, estes Ids deverão ser usados para gerar o UUID, caso contrário poderá ser utilizado um UUID aleatório.

importer

Aplicação do tipo daemon, que corre em background. A aplicação deve constantemente procurar por novos ficheiros CSV no volume **csv** do Docker e iniciar a conversão para XML e posterior migração para a base de dados **pg-xml**. Nota importante: cada CSV deverá produzir vários ficheiros XML, cada um com uma parte dos dados originais. A variável de ambiente NUM_XML_PARTS indica o número de ficheiros XML que deverão ser gerados.

watcher

Aplicação que consulta a base de dados pg-xml, verificando se existem novos ficheiros XML adicionados. Cada vez que é detetado um novo ficheiro, a aplicação deverá gerar mensagens para o serviço **broker**. Os tipos de tarefa a gerar são:

- Importar uma entidade: por cada entidade detetada, gerar uma tarefa para importar essa entidade

- Atualizar dados geográficos: por cada nova entidade com dados geográficos incompletos, deverá ser gerada uma tarefa para atualizar os dados geográficos.

Nota importante: será necessário definir precedência de tarefas, isto é, não deverá ser possível realizar a tarefa de atualização dos dados geográficos antes da migração da entidade.

broker

Instância do RabbitMQ que gere as mensagens com as tarefas do sistema e as distribuí pelos serviços do tipo migrator ou gis-updater.

migrator

Aplicação do tipo daemon, que recebe tarefas de migração das entidades detetadas nos ficheiros XML. Poderão existir vários containers deste tipo a correr ao mesmo tempo. A aplicação recebe uma tarefa do broker e migra os dados para a base de dados **pg-rel**, utilizando a API **api-entities**.

update-gis

Aplicação do tipo daemon, que recebe tarefas de atualização de dados geográficos das entidades detetadas nos ficheiros XML. Poderão existir vários containers deste tipo a correr ao mesmo tempo. A aplicação recebe uma tarefa do broker e, à semelhança do TP1, atualiza as coordenadas com a API externa Search API do Nominatim. A escrita de atualização deverá ser feita utilizando a API **api-gis**.

api-entities

Aplicação do tipo Web API, em Nest.js e Prisma. A API deve permitir realizar CRUD de todas as entidades.

Deverão ser igualmente criados endpoints que permitam relacionar entidades umas com as outras (relação 1-N descrita em **pg-rel**).

A nomenclatura dos *endpoints* da API deverá seguir as normas REST. Deverão ser utilizados os estados e verbos HTTP mais adequados para as respostas e perguntas à API. Exemplos:

I. Obter um filme

```
GET {HOSTNAME}/api/movies/cc29b080-be7c-4e7f-b80c-76d47aee400c [200]
```

II. Obter os atores de um filme.

```
GET {HOSTNAME}/api/movies/a8248e6f-7f58-4734-9768-e4fd287ef97f/actors [200]
```

III. Adicionar um ator a um filme

```
POST {HOSTNAME}/api/movies/a8248e6f-7f58-4734-9768-e4fd287ef97f/actors [201]
```

IV. Atualizar um ator de um filme

```
PUT {HOSTNAME}/api/movies/a8248e6f-7f58-4734-9768-e4fd287ef97f/actors/ 8bb06980-
53ff-4c13-b3fb-4887b3a4e935 [200 ou 204]
```

V. As operações de PUT deverão ser idempotentes.

VI. Todas as operações deverão ser validadas através dos modelos e respeitar as restrições de integridade e relações (ex: não deverá ser possível apagar um Ator que já esteja incluído em algum filme).

VII. O formato de retorno de todos os dados é JSON.

VIII. **Pontos extra**: este serviço utilizar migrações para a criação da base de dados.

api-gis

API em Django para obter os dados geográficos por região.

A API terá dois endpoints:

- GET /api/tile – obtenção de todas as entidades em determinada área
- PATCH /api/entity/{ID} – atualização de uma entidade através do seu id único

Relativamente ao endpoint de obtenção de entidades:

```
GET {HOSTNAME}/api/tile?neLat=39.50&neLng=76.4&swLat=39.3&swLng=76.6 [200]
```

Os parâmetros compõe um retângulo e são os seguintes: neLat – latitude noroeste; neLng – longitude noroeste; swLat – latitude sudoeste; swLng – longitude sudoeste.

O endpoint deverá retornar todos as entidades que contenham coordenadas presentes na base de dados **pg-rel**, que sejam contidas nas coordenadas dos parâmetros. O formato de retorno dos dados é GeoJSON (ver https://geojson.org/). Os documentos em Geo JSON deverão retornar todas as propriedades da tabela, na região *"properties"* do formato, além da parte geométrica.

Dicas: poderá ser utilizada no SQL a função dos PostGIS denominada **ST_MakeEnvelope** para selecionar as entidades dentro da região. O código seguinte é um exemplo de como poderão construir um GeoJSON diretamente numa query SQL:

```
SELECT jsonb_build_object(
    'type',       'Feature',
    'id',         id,
    'geometry',   ST_AsGeoJSON(geom)::jsonb,
    'properties', to_jsonb( t.* ) - 'id' - 'geom'
    ) AS json
FROM (VALUES (1, 'one', 'POINT(1 1)'::geometry)) AS t(id, name, geom);
```

api-proc

API em Django que disponibiliza um endpoint por cada uma das funções criadas no **rpc-server** durante o TP1. A API deve comunicar diretamente com o **rpc-server** e passar os parâmetros à mesma, devolvendo o retorno do **rpc-server**.

api-graphql

Este módulo não é obrigatório, servindo somente como ponto de melhoria. As funções para consulta de dados desenvolvidas no TP1 devem ter uma implementação correspondente utilizando GraphQL. Os dados

deverão ser obtidos da base de dados **pg-rel** em vez da **pg-xml**, sendo por isso necessário refazer as queries em SQL em vez de XPath.

frontend-ent

Frontend simples que permite efetuar consultas à API **api-entities**. É fornecida uma estrutura base em React.

frontend-gis

Frontend simples que utiliza Leaflet.js. Este frontend permite efetuar consultas à API **api-gis**. É fornecida uma estrutura base em React.

frontend-proc

Frontend simples que permite efetuar consultas à API **api-proc** e **api-graphql**. Deverá ser criado, para cada função, um formulário simples que permite indicar os argumentos a utilizar. A interface deverá chamar a função através da **api-proc** e **api-graphql** (se disponível), indicando o resultado de ambas.

rpc-server

O servidor RPC desenvolvido durante o TP1.

Volumes

A aplicação contém também 2 volumes em Docker:

- **csv** – local onde devem ser depositados dos os ficheiros CSV que pretendemos importar para o projeto.
- **xml** – local onde devem ser depositados os ficheiros XML.

Estrutura de comunicação entre containers

- Portos 1****: bases de dados disponibilizadas fora do Docker
- Portos 2****: APIs acessíveis no exterior. De modo a simplificar o trabalho, não será utilizada autenticação para aceder a estes serviços.
- Portos 3****: interfaces web disponíveis que permitem aceder aos dados através das APIs.