Computação Distribuída
2019 / 2020

Licenciatura em Engenharia Informática

Trabalho Prático #4 – Aplicação web RESTful

Introdução

Pretende-se construir a aplicação *Tasklists*, um *software-as-a-service* online para a gestão de tarefas. Esta será constituído por uma aplicação no servidor, implementada usando a web framework *Flask* (http://flask.pocoo.org/), e um *frontend* do tipo SPA (Single-Page Application). A comunicação deverá ser feita maioritariamente através de *webservices RESTful*.

Funcionamento geral

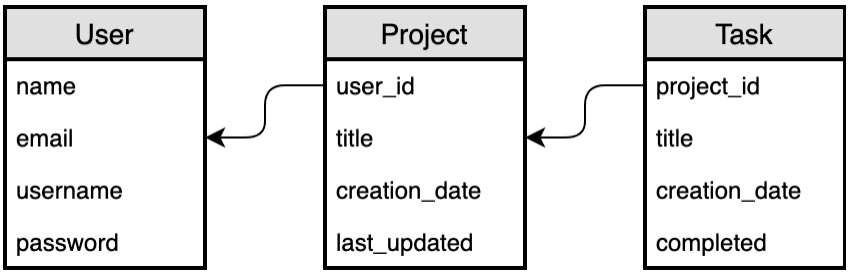
Quando se acede ao site, deverá ser possível aos utilizadores registarem-se ou inserirem as suas credenciais. Após inserir as suas credenciais o utilizador deverá ser redirecionado para a página principal da aplicação.

A página principal da aplicação será constituída por duas áreas: na esquerda deverá estar a lista com todos os projectos do utilizador, ordenados por data de última actualização, e à direita as tarefas pertencentes ao projecto selecionado. Deverá haver uma forma simples e *user-friendly* de criar, editar e remover projectos e tarefas. Deverá também ser possível marcar/desmarcar as tarefas como concluídas, e as tarefas já concluídas deverão estar claramente demarcadas das restantes.

Por fim, no topo da página principal deverá haver um botão para sair da aplicação e um link para algo que permita modificar os dados pessoais do utilizador.

Persistência de dados

A informação na aplicação deverá persistir numa base de dados SQLite3. Para tal sugere-se a seguinte estrutura:



API REST

Segue uma lista de *endpoints* necessários para a API.

1. **Utilizador**:
   - `/api/user/register/` - Registar um utilizador.
   - `/api/user/` - Obter e modificar informação de um utilizador.

2. **Projectos:**
   - `/api/projects/` - Obter lista de projectos ou adicionar um projecto a um utilizador.
   - `/api/projects/<id>/` - Obter, editar ou remover um projecto a um utilizador.

3. **Tarefas:**
   - `/api/projects/<id>/tasks/` - Obter lista de tarefas ou adicionar nova tarefa.
   - `/api/projects/<id>/tasks/<id>/` - Obter, editar ou remover uma tarefa.

De notar que todos os *endpoints* da API apenas recebem e/ou retornam JSON. O HTML deverá estar em ficheiros estáticos e a informação para preencher na aplicação deverá ser obtida via API.

Autenticação e autorização

Todos os endpoints (excepto o *register*) requerem autenticação para se poder identificar o utilizador e verificar se este tem permissões para aceder ao projecto e às respectivas tarefas. O esquema de autenticação a usar será o *basic auth* (https://en.wikipedia.org/wiki/Basic_access_authentication).

O código fornecido contém um exemplo de autenticação no *endpoint* dos *users*. Para usar *Basic Auth* no Postman: https://learning.postman.com/docs/postman/sending-api-requests/authorization/#basic-auth.

Unittests

O ficheiro *tests.py* contém *unittests* cujo objectivo é testar o funcionamento correcto da API. Dado que os testes estão bastante incompletos, a implementação correcta destes será cotado como um extra.

Caso utilizem ORMs (SQLAlchemy, etc.) poderão adaptar o ficheiro de testes de modo a utilizar correctamente o ORM.

Entrega e avaliação

Este trabalho deverá ser realizado em **grupos de 2 alunos** usando o *Github Classrooms* como repositório de código e tem como data limite o **dia 28/Junho/2020 às 23h55**. Deverá ser modificado o *readme* de forma a incluir a identificação dos alunos e do docente responsável, o *link* para o repositório, e os extras implementados.

Todos os ficheiros deverão ser colocados num **ficheiro zip** (com o número dos elementos do grupo) e submetido via *moodle*.

Irá considerar-se a seguinte grelha de avaliação:

Correcção da solução (testes, avaliação manual, etc.)	12 val.
Qualidade e modularização do código (pylint, etc.)	04 val.
Extras (~1 valor por extra)	04 val.

Alguns extras a considerar: (1) utilização de frameworks no *frontend* (*bootstrap*, *reactjs*, *vue.js*, etc.); (2) *deployment* da aplicação na *cloud* (ex: *pythonanywhere.com*); (3) mensagens privadas entre utilizadores; (4) implementação de *unittests* relevantes; (5) utilização de ORMs (*SQLAlchemy*, *peewee*, etc.); (6) utilização de *plugins* que resolvam problemas concretos (*flask-admin*, *flask-session*, *flask-mail*, *flask-restful*, http://flask.pocoo.org/extensions/, etc.); (7) etc.., sejam criativos!

Bom trabalho!