Relatório de ativade avaliativa
Prova 01

Lohan T. Tosta[1]

[1]Faculdade de Computação - Universidade Federal de Mato Grosso do Sul (UFMS)
Campo Grande – MS – Brazil

lohan.toledo@ufms.br

Resumo. *Este relatório trata da execução da atividade avaliativa "Prova 01", o desenvolvimento de um sistema distribuído em containers à partir de um código fonte inicial. Por meio de uma explicação detalhada, este relatório ilustra as etapas da atividade e as mudanças ocorridas no código, bem como apresenta as devidas justificativas.*

1. Das etapas

O ponto de partida do projeto consistia em um diretório básico, contendo: um arquivo de código Python, um arquivo de *logging*, um arquivo de banco de dados, um arquivo de dependências e um arquivo para variáveis de ambiente. O objetivo final, conforme indicado pela atividade, é organizar um repositório estruturado, contendo um diretório de código-fonte em Python, um arquivo de dependências, um *Dockerfile* para compilar a imagem do *container* da aplicação e um arquivo *docker-compose* para compor o sistema em múltiplos *containers*, incluindo um *container* específico para o banco de dados MySQL.

Para alcançar esses objetivos, as etapas iniciais envolveram: leitura e entendimento dos artefatos, execução e análise de funcionalidades, segmentação do código, *containerização* da aplicação e desenvolvimento de novos requisitos. Cada etapa foi essencial para o processo de transposição da aplicação para um sistema distribuído e na implementação de novas funcionalidades.

2. Leitura e entendimento dos artefatos

A primeira etapa consistiu em compreender as dependências da aplicação. Esse processo iniciou-se pela análise das importações em *app.py* e do arquivo *requirements.txt*. A consulta à documentação oficial disponível foi fundamental para identificar as bibliotecas e ferramentas necessárias para a execução da aplicação.

Em seguida, a leitura detalhada do código, comando por comando, foi essencial para entender as diferentes partes da aplicação, incluindo: a inicialização de componentes e configurações, a implementação de classes ligadas às entidades de domínio ou *views*, as funções de serviço e controle de rotas, e a sequência de inicialização da aplicação como um todo.

3. Execução e análise de funcionalidades

Após revisar as instruções de execução da aplicação fornecidas no documento da atividade e entender o código estático, iniciou-se o processo de execução da aplicação localmente. A aplicação foi executada sem problemas, e a inclusão de *loggings* temporários no código permitiu observar o fluxo de dados, validar as funcionalidades implementadas e identificar as funcionalidades faltantes segundo o documento da atividade.

4. Segmentação do código

Feitas a verificação e validação do código em tempo de desenvolvimento e execução, iniciou-se o processo de segmentação do código. O código foi separado em diversos arquivos dentro de um diretório nomeado *src*.

Em *src*, os arquivos: *app.py*, *admin.py*, *auth.py*, *db.py* e *log.py* - responsáveis pela inicialização de componentes de mesmo nome; *config.py* - responsável pela obtenção e disponibilização de variáveis de ambiente; *main.py* - responsável pela inicialização da aplicação como um todo, bem como pela exposição em porta, conforme disponibilizado por *config*.

Em *controllers*, diretório de *src*, o arquivo *profile_controller.py*, responsável pelo controle da rota principal da aplicação.

Em *models*, diretório de *src*, o arquivo *profile.py*, responsável por definição do modelo conforme o componente de banco de dados demanda. Facilitador para *CRUD* desta entidade.

Em *services*, diretório de *src*, os arquivos: *auth_service.py* e *profile_service.py*, ambos responsáveis por expor funções (serviços) relacionados a autorização e à entidade *profile*, respectivamente.

Em *views*, diretório de *src*, o arquivo *my_model_view.py*, responsável pela implementação de uma *view* personalizada para o componente admin. Também o arquivo *profile_view*, que implementa uma *view* que herda o comportamento de *MyModelView* e fornece uma interface de usuário para *CRUD* de registros do tipo *Profile*.

5. *Containerização* da aplicação

Após a segmentação da aplicação. Iniciou-se a containerização da aplicação. Além da elaboração de um *dockerfile* que fizesse uso apenas dos arquivos necessários e compilasse a imagem da aplicação, duas implementações importantes foram realizadas: a mudança na forma de obtenção de variáveis de ambiente e a declaração de volumes do docker por onde o container acessaria o banco de dados (*sqlite*) e registraria *logs*.

A mudança na forma de obtenção de variáveis de ambiente foi implementada nesta fase, visando facilitar o desenvolvimento de novos requisitos e agilizar o processo de compilação e execução dos containers durante os testes. Ela consistia em não declará-las em arquivo separado, mas na inicialização dos *containeres*.

A declaração de volumes foi feita para garantir a persistência dos dados, independentemente do ciclo de vida do container. Além disso, é a forma pela qual múltiplos containers dessa aplicação podem acessar os mesmos dados.

6. Desenvolvimento de novos requisitos

Após a *containerização* da aplicação, restou implementar os requisitos faltantes: a compatibilidade com o banco de dados *MySql* e a inicialização dos containeres da aplicação com um usuário administrador criado automaticamente.

Em primeiro lugar, para tornar compativel com o banco *MySql* duas dependências novas foram incluídas em *requirements.txt*: *pymysql* e *cryptography*. Com as de-

pendências instaladas, bastou definir e implementar a forma como a aplicação se comportaria. Ficou definido que, desejando usar o banco sqlite, nenhuma variável de ambiente a respeito de banco de dados deve ser incluída. Em favor de uso do banco MySql, além da containerização adequada, a variável de ambiente DATABASE deve ser declarada contendo a URI de conexão com o banco.

Para a inicialização com criação de usuário administrador automática, duas variáveis de ambiente foram utilizadas: *ADMIN_USER* e *ADMIN_PASSWORD* definindo nome e senha do usuário a ser criado. Quando não forem declaradas, um usuário padrão de nome ”admin” e senha ”123” será criado. Quando inicializada, a aplicação verifica a existência deste usuário e, caso não exista, o cria no banco.

Além dos requisitos faltantes, uma última alteração foi feita: o fator determinante para a qualidade de administrador para algum usuário deixa de ser a variável de ambiente *ADMINISTRATORS* e passa a ser um campo relacionado à entidade *Profile* .

7. Conclusão

Ao final das estapas, após criar o arquivo *docker-compose* para verificar e validar o deploy do sistema, e após compilação da imagem da aplicação e submissão para o serviço *Docker Hub*, bem como sumbissão do repositório para o serviço *Git Hub*, o deploy deste sistema, que foi analisado, executado, distribuído e aperfeiçoado, é possível à partir da execução do *Docker-compose* com arquivo homônimo em apenas um comando.