Universidade Federal do Espírito Santo
Centro Tecnológico
Departamento de Informática

Banco de Dados I (INF15979)
Prof. Rodrigo Laiola Guimarães

Especificação do Trabalho

O primeiro trabalho da disciplina de Banco de Dados I consiste no desenvolvimento de um microsserviço de *back-end* que permite a aplicações clientes (por ex., *front-end* ou *Application Programming Interface client*) gerenciar turmas e objetos de aprendizagem armazenados em um SGBD através de uma RESTful[1] API. Este trabalho oportuniza a prática dos conhecimentos obtidos na disciplina com tecnologias atuais e em um sistema real, o ambiente BOCA (*BOCA Online Contest Administrator*), o qual é usado para gerenciar competições da Maratona de Programação da SBC[2] e do Topcom[3] e, mais recentemente, como ferramenta de apoio em disciplinas de programação oferecidas pelo Departamento de Informática (DI) da Ufes.

Requisitos

O trabalho poder ser feito em grupos de até 2 alunos(as) e a implementação deve utilizar JavaScript e Node.js. Qualquer *framework*/biblioteca auxiliar (por ex., TypeScript) pode ser utilizado(a) desde que as consultas ao banco de dados sejam escritas explicitamente através de *statements* SQL. Deve ser utilizada como ponto de partida a versão conteinerizada e baseada em microsserviços do BOCA disponível em https://github.com/joaofazolo/boca-docker. A descrição sobre o seu funcionamento e a estrutura do seu banco de dados está presente no documento em anexo, enquanto as instruções para sua implantação estão disponíveis no referido repositório de código. Para favorecer a manutenibilidade e o desenvolvimento de novas funcionalidades, é imprescindível que o *back-end* seja encapsulado em um contêiner independente (vide Figura 1) e que alterações na estrutura do banco de dados (por ex., criação de novas tabelas) sejam dispostas em arquivos separados (ou *migrations*) e executadas na implantação do serviço (vide exemplos no repositório de código). Inclua um arquivo README.MD no repositório que, além das informações para execução do projeto, também possua: decisões, desafios/problemas com os quais vocês se depararam durante a execução do projeto; maneiras através das quais vocês podem melhorar a aplicação, seja em performance, estrutura ou padrões.

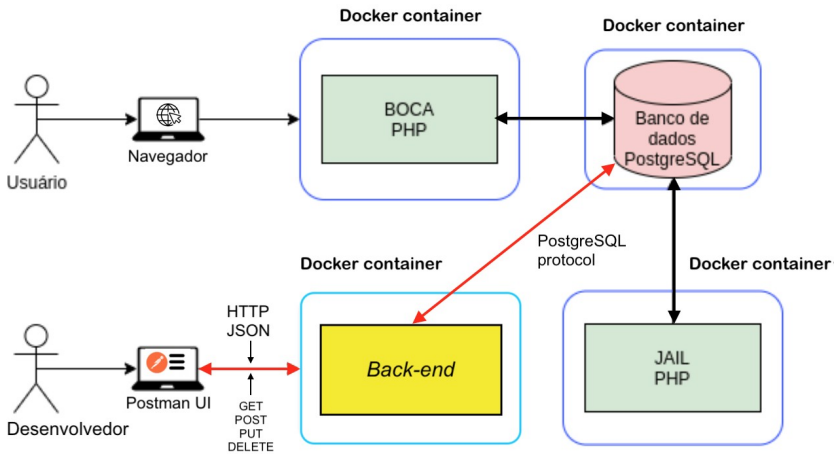

Figura 1 – Arquitetura conteinerizada do BOCA. Fonte: Modificado de Fazolo, 2021.

Dicas

- Tenham sempre um *mindset* de usabilidade, escalabilidade e colaboração;
- Documente o projeto e o código Markdown explicando a estrutura, processo de setup e requisitos;
- A organização das *branches* e os *commits* no repositório falam muito sobre a participação dos membros durante o desenvolvimento do trabalho;
- Usem boas práticas de programação;

1 Também conhecida como REST ou Representational State Transfer; um padrão de comunicação utilizado para interoperabilidade entre sistemas de computação.

2 http://maratona.ime.usp.br/manualBOCA.html

3 https://topcom.pet.inf.ufes.br

- Considerem que não sei nada sobre os seus conhecimentos, então quanto mais vocês mostrarem e o quão mais descritiva for a documentação e testes, melhor.

Podemos utilizar *frameworks*/bibliotecas?
R: Vocês podem usar bibliotecas, mas é importante que o impacto/relevância dessa decisão sobre a solução proposta seja devidamente justificada.

Posso usar um ORM (*Object Relational Mapper*)?
R: O uso de ORM é permitido apenas para estabelecer a conexão com o banco de dados. No entanto, para que seja possível avaliar os conhecimentos em SQL não usem ORM nas consultas ao banco. As bibliotecas normalmente possuem métodos que permitem consultas diretamente em SQL (*raw queries*) mesmo através do ORM.

Links úteis:
- https://nodejs.org/en/docs/guides/nodejs-docker-webapp/
- https://blog.logrocket.com/nodejs-expressjs-postgresql-crud-rest-api-example/

API

- Implementar as funções de CRUD (Create, Read, Update, Delete) para a tabela `contesttable`;

Endpoint	Método	Funcionalidade
api/contest	`GET`	Lista as competições de programação cadastradas
api/contest	`POST`	Cadastra uma nova competição de programação
api/contest/:id_c	`GET`	Mostra a competição de programação dada pelo id_c
api/contest/:id_c	`PUT`	Atualiza a competição de programação dada pelo id_c
api/contest/:id_c	`DELETE`	Remove a competição de programação dada pelo id_c

- Implementar as funções de CRUD para as tabelas dependentes (por exemplo, `contesttable` → `problemtable`, `contesttable` → `langtable` e `contesttable` → `sitestable` → `usertable`;

Exemplos para a tabela `problemtable`:

Endpoint	Método	Funcionalidade
api/contest/:id_c/problem	`GET`	Lista os problemas associados à competição dada pelo id_c
api/contest/:id_c/problem	`POST`	Cadastra um novo problema associado à competição dada pelo id_c
api/contest/:id_c/problem/:id_p	`GET`	Mostra o problema dado pelo id_p no contest id_c
api/contest/:id_c/problem/:id_p	`PUT`	Atualiza o problema dado pelo id_p no contest id_c
api/contest/:id_c/problem/:id_p	`DELETE`	Remove o problema dado pelo id_p no contest id_c

- Implementar as funções de CRUD para que seja possível indicar em quais linguagens um problema pode ser submetido (na modelagem atual, é permitido enviar a solução em qualquer uma das linguagens cadastradas mesmo essas não sendo suportadas para alguns exercícios);

Endpoint	Método	Funcionalidade
api/contest/:id_c/problem/:id_p/language	`GET`	Lista as linguagens de programação associadas ao problema dado pelo id_p
api/contest/:id_c/problem/:id_p/language	`POST`	Adiciona linguagens de programação ao problema dado pelo id_p
api/contest/:id_c/problem/:id_p/language	`DELETE`	Remove linguagens de programação associadas ao problema dado pelo id_p

- Criar a tabela `workingtable` para possibilitar a estruturação (listas) de problemas/exercícios dentro de uma competição (vide Figura 2). Implementar as funções de CRUD para as associações/relacionamentos pertinentes (por exemplo, `workingtable` x `usertable`, `workingtable` x `problemtable`;

Endpoint	Método	Funcionalidade
api/contest/:id_c/working/:id_w/user	`GET`	Lista os usuários associados à lista dada pelo id_w
api/contest/:id_c/working/:id_w/user	`POST`	Adiciona usuários à lista dada pelo id_w
api/contest/:id_c/working/:id_w/user	`DELETE`	Remove usuários da lista dada pelo id_w
api/contest/:id_c/user/:id_u/working	`GET`	Lista as listas associadas ao usuário dado pelo id_u
api/contest/:id_c/user/:id_u/working	`POST`	Adiciona listas ao usuário dado pelo id_u
api/contest/:id_c/user/:id_u/working	`DELETE`	Remove working do usuário dado pelo id_u

Submissão

Cada grupo deve criar um **repositório privado** no GitHub (sob a organização UFES20222BDCOMP) para hospedar a implementação do trabalho. Para isso, será preciso antes informar os *usernames* dos membros do grupo através deste link (somente um dos membros precisa fazê-lo até 14/11/2022, mas todos devem fazer parte do projeto posteriormente). Além do código fonte, o projeto deve conter toda a documentação necessária para que um usuário de um curso de graduação em computação na Ufes possa executá-lo (ou seja, desde informações sobre os pré-requisitos de hardware e software até mesmo comandos básicos para configurar o ambiente e implantar o serviço). Para finalizar o trabalho, deve ser criada uma *release* do projeto até a data limite de entrega.

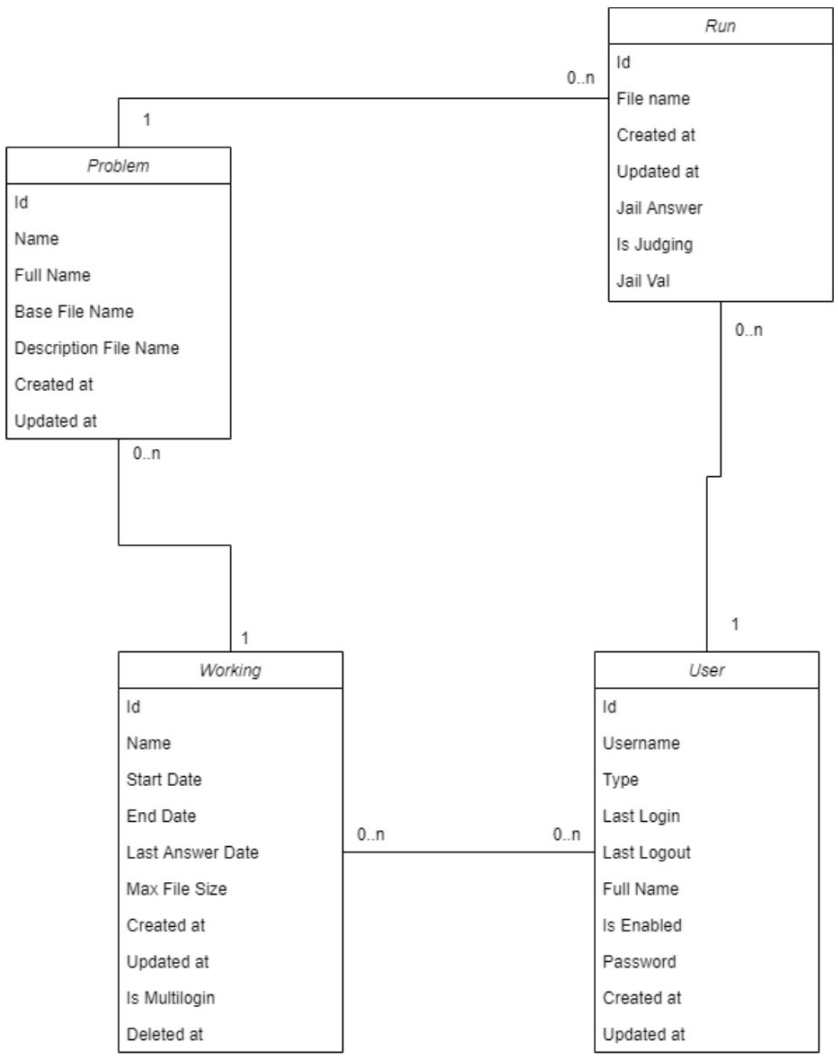

Figura 2 – Diagrama de classes do BOCA incluindo a classe *Working*. Fonte: Fazolo, 2021.

Prazos

Os prazos de entrega do trabalho podem ser consultados no calendário da página do curso no Google Classroom.

Avaliação

O trabalho será avaliado com base nos seguinte critérios:

- Funciona?
- Cumpre os requisitos?
- Qualidade da solução proposta (modelagem, desempenho, etc.)
- Boas práticas (estrutura e organização do código, documentação, etc.)
- Testes unitários automatizados

Observações finais

Caso haja algum erro neste documento, serão publicadas novas versões e divulgadas erratas nas aulas. É responsabilidade do(a) aluno(a) manter-se informado(a), frequentando as aulas ou acompanhando as novidades através das ferramentas utilizadas na disciplina.

Referências

- João Vitor Alves Fazolo, 2021. AlgoChallenger - Reengenharia do BOCA Online Contest Administrator utilizando uma arquitetura de microserviços para o ensino de programação. Trabalho de conclusão de curso de graduação (Ciência da Computação) - Universidade Federal do Espírito Santo.