UNIVERSIDADE FEDERAL DE RORAIMA
CENTRO DE CIÊNCIAS E TECNOLOGIA - CCT
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO - DCC

KAIO GUILHERME FERRAZ DE SOUSA SILVA

RELATORIO DE BANCO DE DADOS SGCURSOS

Boa Vista
202

RESUMO

O presente relatório descreve o projeto SGCURSOS, desenvolvido como trabalho final da disciplina de Banco de Dados 2 na Universidade Federal de Roraima (UFRR). O projeto consiste em uma aplicação de gerenciamento de cursos online, com recursos de registro de usuários, criação e consulta de categorias e cursos, matrículas, emissão de certificados, entre outros. A arquitetura da aplicação é baseada em uma API desenvolvida em Node.js com o framework Express, utilizando o banco de dados PostgreSQL para armazenamento dos dados em tabelas como Users, Category, Course, Registration e Certificate. Além disso, foi implementado um frontend composto por diversas páginas HTML para interação com a API. Para facilitar a implantação, foi utilizado o Docker Compose, permitindo a criação de múltiplos containers para cada serviço, como banco de dados, backend e frontend. O projeto SGCURSOS representa a aplicação prática dos conhecimentos adquiridos na disciplina, abrangendo desde a modelagem do banco de dados até a implementação de uma aplicação funcional, contribuindo para a consolidação dos conceitos estudados.

SUMÁRIO

1. Banco

Para o projeto SGCursos, será utilizado o PostgreSQL como sistema de gerenciamento de banco de dados. A seguir, apresentamos a modelagem das tabelas que serão criadas no banco de dados:

1.1 Tabela Users

A tabela "Users" armazena informações sobre os usuários do sistema. Cada usuário é identificado por um ID único gerado automaticamente. Além disso, a tabela contém os campos "username" para o nome de usuário, "password" para a senha, "email" para o endereço de e-mail, "name" para o nome completo do usuário, "number" para o número de telefone, "image_path" para o caminho da imagem do perfil (opcional), "profile" para o perfil do usuário (admin, student, root) e "is_suspended" para indicar se o usuário está suspenso.

Nome	Tipo	Finalidade
id	INTEGER	Chave primária e autoincremento
username	STRING(50)	Nome de usuário
password	STRING(255)	Senha do usuário
email	STRING(255)	Endereço de email

name	STRING(100)	Nome completo do usuário
number	STRING(20)	Número de telefone
image_path	STRING(255)	Caminho da imagem do perfil (opcional)
profile	ENUM('admin', 'student', 'root')	Perfil do usuário (administrador, estudante ou root)
is_suspended	BOOLEAN	Indica se o usuário está suspenso

1.2 Tabela Category

A tabela "Category" armazena as categorias dos cursos. Cada categoria possui um ID único gerado automaticamente. Os campos "name" e "description" são utilizados para armazenar o nome e a descrição da categoria, respectivamente.

Nome	Tipo	Finalidade
id	INTEGER	Chave primária e autoincremento
name	STRING(100)	Nome da categoria
description	STRING(255)	Descrição da categoria

1.3 Tabela Course

A tabela "Course" armazena informações sobre os cursos oferecidos. Cada curso é identificado por um ID único gerado automaticamente. Os campos incluem "name" para o nome do curso, "description" para a descrição detalhada, "tags" para as tags relacionadas ao curso, "start_date" para a data de início, "duration_hours" para a duração em horas, "lessons" para a lista de lições, "status" para indicar se o curso está aberto ou fechado, "banner" para o caminho do banner do curso (opcional) e "category_id" como chave estrangeira para relacionar o curso com a categoria correspondente.

Nome	Tipo	Finalidade
id	INTEGER	Chave primária e autoincremento
name	STRING(100)	Nome do curso
description	TEXT	Descrição do curso
tags	ARRAY(DataTypes.STRING)	Tags relacionadas ao curso
start_date	DATE	Data de início do curso
duration_hours	INTEGER	Duração do curso em horas
lessons	ARRAY(DataTypes.STRING)	Lista de lições do curso
status	ENUM('open', 'closed')	Status do curso (aberto ou fechado)
banner	STRING	Caminho do banner do curso (opcional)
category_id	INTEGER	Chave estrangeira para a tabela de categorias

1.4 Tabela Registration

A tabela "Registration" registra a participação dos usuários nos cursos. Cada registro de participação é identificado por um ID único gerado automaticamente. Os campos incluem "progress_time" para o tempo de progresso no curso em porcentagem, "final_grade" para a nota final do registro no curso, "User_id" como chave estrangeira para relacionar o registro com o usuário correspondente e "Course_id" como chave estrangeira para relacionar o registro com o curso correspondente. Além disso, o campo "Certificate_id" é uma chave estrangeira opcional para relacionar o registro com o certificado correspondente.

Nome	Tipo	Finalidade
id	INTEGER	Chave primária e autoincremento
progress_time	INTEGER	Tempo de progresso no curso em porcentagem
final_grade	FLOAT	Nota final do registro no curso
User_id	INTEGER	Chave estrangeira para a tabela de usuários
Course_id	INTEGER	Chave estrangeira para a tabela de cursos
Certificate_id	INTEGER	Chave estrangeira para a tabela de certificados

1.5 Tabela Certificate

A tabela "Certificate" armazena informações sobre os certificados emitidos para os usuários. Cada certificado possui um ID único gerado automaticamente. Os campos incluem "name" para o nome do certificado, "course_name" para o nome do curso associado, "course_duration" para a duração do curso em horas, "registration_id" para o ID do registro associado ao certificado, "issued_at" para a data de emissão, "final_grade" para a nota final do curso associado e "validate_code" para o código de validação do certificado (opcional).

Nome	Tipo	Finalidade
id	INTEGER	Chave primária e autoincremento
name	STRING	Nome do certificado
course_name	STRING	Nome do curso associado ao certificado
course_duration	INTEGER	Duração do curso em horas
registration_id	STRING	ID do registro associado ao certificado
issued_at	DATE	Data de emissão do certificado
final_grade	FLOAT	Nota final do curso associado ao certificado
validate_code	STRING	Código de validação do certificado (opcional)

Essas tabelas estão interconectadas por meio de chaves estrangeiras para estabelecer relacionamentos entre os usuários, cursos, categorias, registros de participação e certificados. Essas relações permitem que o sistema registre a participação dos usuários nos cursos, atribua certificados e forneça informações relevantes sobre os cursos e usuários.

Essa estrutura de banco de dados permite o armazenamento organizado de informações relacionadas a usuários, cursos, categorias, registros de participação e certificados, facilitando a consulta e manipulação desses dados dentro do sistema.

1.6 populando o banco

Durante o processo de população do banco de dados do SGCURSOS, foram realizadas as seguintes etapas:

1. Inserção manual dos cursos:
    - Utilizando o arquivo **courses_data.sql**, foram inseridos manualmente 20 cursos no banco de dados.
    - As informações incluíam título, descrição, carga horária e categoria.
2. Geração de usuários fictícios:
    - Utilizando o script Python **generate_user.py**, foram gerados 200 usuários fictícios com informações aleatórias.
    - A biblioteca Faker foi utilizada para gerar dados fictícios como nomes, e-mails e senhas.
3. Geração de matrículas nos cursos:
    - Utilizando o script Python **generate_register.py**, foram geradas matrículas aleatórias para os cursos existentes.
    - O objetivo era garantir que cada curso tivesse pelo menos uma matrícula e que os usuários estivessem matriculados em diferentes cursos.
4. Execução dos arquivos SQL no pgAdmin:
    - Os dados gerados pelos scripts Python foram armazenados nos arquivos SQL: **insert_users.sql** e **insert_registrations.sql**.
    - Esses arquivos foram executados no pgAdmin, inserindo os dados fictícios nas tabelas correspondentes do banco de dados.
5. Liberação do acesso à porta pelo Docker Compose:
    - No arquivo **docker-compose.yml**, as portas utilizadas pelo PostgreSQL foram mapeadas para as portas do host.
    - Essa liberação permitiu o acesso adequado ao banco de dados durante o processo de população.

Dessa forma, foi possível popular o banco de dados do SGCURSOS com cursos, usuários e matrículas, utilizando inserção manual para os cursos e scripts Python para gerar usuários e matrículas aleatórias. O acesso ao banco de dados foi facilitado por meio do pgAdmin e a

liberação das portas no Docker Compose permitiu a conexão adequada entre o banco de dados e outras ferramentas.

2. API

a API SG Cursos permite a interação com outras aplicações por meio de requisições HTTP, proporcionando uma integração eficiente com sistemas existentes. Ela também oferece recursos avançados, como a geração de certificados para os alunos que concluíram com sucesso um curso específico.

A API SG Cursos foi construída utilizando as seguintes bibliotecas e dependências:

- **Express** (^4.18.2): O Express é um framework web rápido e minimalista para Node.js. Ele foi utilizado para criar as rotas e lidar com as requisições HTTP na API.
- **Express Validator** (^7.0.1): O Express Validator é uma biblioteca para validação de dados no Express. Ele foi utilizado para validar e sanitizar os dados enviados nas requisições.
- **pg** (^8.11.0): O pg é um cliente Node.js para PostgreSQL. Ele foi utilizado para realizar a conexão e interagir com o banco de dados PostgreSQL utilizado pela API.
- **pg-hstore** (^2.3.4): O pg-hstore é um pacote para manipulação de dados no formato hstore do PostgreSQL. Ele foi utilizado para trabalhar com colunas hstore no banco de dados.
- **Sequelize** (^6.31.1): O Sequelize é um ORM (Object-Relational Mapping) para Node.js. Ele foi utilizado para modelar e interagir com os dados do banco de dados PostgreSQL de forma simplificada.
- **Sequelize CLI** (^5.3.0): O Sequelize CLI é uma ferramenta de linha de comando para o Sequelize. Ele foi utilizado para criar e executar as migrações e seeders do banco de dados.
- **bcrypt** (^5.0.1): O bcrypt é uma biblioteca para criptografia de senhas. Ele foi utilizado para criptografar e comparar as senhas dos usuários armazenadas no banco de dados.
- **cors** (^2.8.5): O cors é um middleware para o Express que permite o controle de acesso a recursos da API através de políticas de compartilhamento de recursos de origens cruzadas (CORS).

- **jsonwebtoken** (^9.0.0): O jsonwebtoken é uma biblioteca para criação e verificação de tokens JWT (JSON Web Tokens). Ele foi utilizado para autenticação e autorização dos usuários na API.
- **morgan** (^1.10.0): O morgan é um middleware para o Express que registra os logs das requisições HTTP. Ele foi utilizado para registrar informações sobre as requisições feitas à API.
- **winston** (^3.9.0): O winston é uma biblioteca de registro de logs para Node.js. Ele foi utilizado para fornecer um sistema de registro de logs mais flexível e customizável na API.

Essas dependências foram escolhidas para proporcionar uma base sólida e eficiente para o desenvolvimento da API SG Cursos, permitindo uma integração com o banco de dados, validação de dados, autenticação de usuários, geração de tokens JWT, criptografia de senhas e registro de logs detalhados.

2.1 Usuários

O endpoint `/users` é o ponto de partida para todas as operações relacionadas a usuários na API. Através deste endpoint, é possível realizar as seguintes operações:

Método	Rota	Header	Função
POST	/register	-	Cria um novo usuário
POST	/login	-	Realiza o login do usuário
GET	/getAll	Authorization	Obtém todos os usuários
GET	/:id	Authorization	Obtém um usuário pelo ID
PUT	/:id	Authorization	Atualiza um usuário pelo ID
DELETE	/:id	Authorization	Exclui um usuário pelo ID

2.1.1 Rota: POST /register

Cria um novo usuário.

- Parâmetros do corpo da requisição:

    - `name` (obrigatório): Nome do usuário.
    - `email` (obrigatório): Endereço de e-mail do usuário.
    - `username` (obrigatório): Nome de usuário.
    - `password` (obrigatório): Senha do usuário.

2.1.2 Rota: POST /login

Realiza o login do usuário.

- Parâmetros do corpo da requisição:
    - `username`: Nome de usuário ou endereço de e-mail.
    - `password`: Senha do usuário.

2.1.3 Rota: GET /getAll

Obtém todos os usuários.

- Cabeçalho da requisição:
    - `Authorization`: Token de autenticação.

2.1.4 Rota: GET /:id

Obtém um usuário pelo ID.

- Parâmetros da URL:
    - `id`: ID do usuário a ser obtido.
- Cabeçalho da requisição:
    - `Authorization`: Token de autenticação.

2.1.5 Rota: PUT /:id

Atualiza um usuário pelo ID.

- Parâmetros da URL:
    - `id`: ID do usuário a ser atualizado.
- Parâmetros do corpo da requisição:
    - `name`: Novo nome do usuário.

    - `email`: Novo endereço de e-mail do usuário.
    - `username`: Novo nome de usuário.
    - `password`: Nova senha do usuário.
- Cabeçalho da requisição:
    - `Authorization`: Token de autenticação.

2.1.6 Rota: DELETE /:id

Exclui um usuário pelo ID.

- Parâmetros da URL:
    - `id`: ID do usuário a ser excluído.
- Cabeçalho da requisição:
    - `Authorization`: Token de autenticação.

2.2 Categorias

O endpoint `/category` é o ponto de partida para todas as operações relacionadas a categorias de cursos na API. Através deste endpoint, é possível realizar as seguintes operações:

Método	Rota	Header	Função
POST	/create	Authorization	Cria uma nova categoria
POST	/get	-	Obtém categorias
PUT	/:id	Authorization	Atualiza uma categoria pelo ID
DELETE	/:id	Authorization	Exclui uma categoria pelo ID

2.2.1 Rota: POST /create

Cria uma nova categoria.

- Parâmetros do corpo da requisição:
    - `name` (obrigatório): Nome da categoria.

    - `description` (obrigatório): Descrição da categoria.

2.2.2 Rota: POST /get

Obtém categorias.

- Parâmetros do corpo da requisição:
    - `all`: Valor booleano que indica se todas as categorias devem ser retornadas. Se for definido como `true`, os demais parâmetros serão ignorados.
    - `id`: ID da categoria a ser obtida.
    - `name`: Nome da categoria a ser obtida.

2.2.3 Rota: PUT /:id

Atualiza uma categoria pelo ID.

- Parâmetros da URL:
    - `id`: ID da categoria a ser atualizada.
- Parâmetros do corpo da requisição:
    - `name`: Novo nome da categoria.
    - `description`: Nova descrição da categoria.
- Cabeçalho da requisição:
    - `Authorization`: Token de autenticação.

2.2.4 Rota: DELETE /:id

Exclui uma categoria pelo ID.

- Parâmetros da URL:
    - `id`: ID da categoria a ser excluída.
- Cabeçalho da requisição:
    - `Authorization`: Token de autenticação.

2.3 Cursos

O endpoint `/courses` é o ponto de partida para todas as operações relacionadas a cursos na API. Através deste endpoint, é possível realizar as seguintes operações:

Tabela das Rotas:

Método	Rota	Header	Função
POST	/register	Authorization	Registra um novo curso
POST	/get	Authorization	Obtém cursos
PUT	/:id	Authorization	Atualiza um curso pelo ID
DELETE	/:id	Authorization	Exclui um curso pelo ID

2.3.1 Rota: POST /register

Registra um novo curso.

- Parâmetros do corpo da requisição:
  - `name` (obrigatório): Nome do curso.
  - `description` (obrigatório): Descrição do curso.
  - `category_id` (obrigatório): ID da categoria do curso.
  - `image_path` (obrigatório): Caminho da imagem do curso.
  - `start_date` (obrigatório): Data de início do curso.
  - `duration_hours` (obrigatório): Duração do curso em horas.
  - `lessons`: Lista de aulas do curso.

2.3.2 Rota: POST /get

Obtém cursos.

- Parâmetros do corpo da requisição:
    - `all`: Valor booleano que indica se todos os cursos devem ser retornados. Se for definido como `true`, os demais parâmetros serão ignorados.
    - `id`: ID do curso a ser obtido.
    - `name`: Nome do curso a ser obtido.
    - `tags`: Lista de tags do curso.
    - `category`: Nome da categoria do curso.
    - `participants`: Valor booleano que indica se as informações dos participantes devem ser retornadas (requer permissão de administrador).

2.3.3 Rota: PUT /:id

Atualiza um curso pelo ID.

- Parâmetros da URL:
    - `id`: ID do curso a ser atualizado.
- Parâmetros do corpo da requisição: (opcionais)
    - `name`: Novo nome do curso.
    - `description`: Nova descrição do curso.
    - `tags`: Nova lista de tags do curso.
    - `category_id`: Novo ID da categoria do curso.
    - `image_path`: Novo caminho da imagem do curso.
    - `start_date`: Nova data de início do curso.
    - `duration_hours`: Nova duração do curso em horas.
    - `lessons`: Nova lista de aulas do curso.
- Cabeçalho da requisição:
    - `Authorization`: Token de autenticação.

2.3.4 Rota: DELETE /:id

Exclui um curso pelo ID.

- Parâmetros da URL:
    - `id`: ID do curso a ser excluído.
- Cabeçalho da requisição:
    - `Authorization`: Token de autenticação
    -

2.4 Matrículas

O endpoint `/register` é o ponto de partida para todas as operações relacionadas à matrícula na API. Através deste endpoint, é possível realizar as seguintes operações:

Método	Rota	Header	Função
POST	/create	Authorization	Cria uma nova matrícula
POST	/get	-	Obtém matrículas
POST	/certificate	Authorization	Gera um certificado para uma matrícula
POST	/certificate/validate	-	Valida um certificado
PUT	/:id	Authorization	Atualiza uma matrícula pelo ID
DELETE	/:id	Authorization	Exclui uma matrícula pelo ID

2.4.1 Rota: POST /create

Cria uma nova matrícula.

- Parâmetros do corpo da requisição:
    - `user_id` (obrigatório): ID do usuário.
    - `course_id` (obrigatório): ID do curso.

2.4.2 Rota: POST /get

Obtém matrículas.

- Parâmetros do corpo da requisição:

    - `all`: Valor booleano que indica se todas as matrículas devem ser retornadas. Se for definido como `true`, os demais parâmetros serão ignorados.
    - `id`: ID da matrícula a ser obtida.
    - `course`: ID do curso da matrícula a ser obtida.
    - `user`: ID do usuário da matrícula a ser obtida.
    - `certificate`: Valor booleano que indica se o certificado deve ser incluído na resposta.

2.4.3 Rota: POST /certificate

Gera um certificado para uma matrícula.

- Parâmetros do corpo da requisição:
    - `id` (obrigatório): ID da matrícula para a qual o certificado deve ser gerado.

2.4.4 Rota: POST /certificate/validate

Valida um certificado.

- Parâmetros do corpo da requisição:
    - `id` (obrigatório): ID do certificado a ser validado.

2.4.5 Rota: PUT /:id

Atualiza uma matrícula pelo ID.

- Parâmetros da URL:
    - `id`: ID da matrícula a ser atualizada.
- Parâmetros do corpo da requisição:
    - `user_id`: Novo ID do usuário.
    - `course_id`: Novo ID do curso.
- Cabeçalho da requisição:
    - `Authorization`: Token de autenticação.

2.4.6 Rota: DELETE /:id

Exclui uma matrícula pelo ID.

- Parâmetros da URL:
    - `id`: ID da matrícula a ser excluída.
- Cabeçalho da requisição:
    - `Authorization`: Token de autenticação.

3. Frontend

O Frontend da API SG Cursos consiste em diversas páginas HTML que permitem aos usuários interagir com os recursos disponíveis. A seguir, é fornecida uma explicação breve do que cada página inclui:

3.1 index

A página "index" é a página inicial da aplicação. Ela apresenta uma visão geral dos cursos disponíveis, destacando as principais categorias e informações relevantes.

3.2 cursos

A página "cursos" exibe uma lista completa de todos os cursos disponíveis. Os usuários podem navegar pelos cursos, visualizar seus detalhes e realizar a matrícula em um curso específico.

3.4 curso

A página "curso" exibe informações detalhadas sobre um curso específico. Ela inclui a descrição do curso, a carga horária, os pré-requisitos e outros detalhes relevantes. Os usuários também têm a opção de se matricular no curso a partir desta página.

3.5 validar

A página "validar" permite aos usuários validar um certificado. Eles podem inserir o código do certificado e obter informações sobre sua autenticidade e validade.

3.6 usuários

A página "usuários" exibe uma lista de todos os usuários registrados na plataforma. Ela fornece informações básicas sobre cada usuário, como nome, e-mail.

3.7 cadastro

A página "cadastro" é destinada ao registro de novos usuários na plataforma. Ela solicita informações pessoais, como nome, e-mail e senha, para criar uma conta de usuário.

3.8 meus-cursos

A página "meus-cursos" exibe uma visão geral dos cursos aos quais um usuário está matriculado. Ela lista os cursos atuais do usuário, suas notas e progresso acadêmico.

3.9 editor

A página "editor" é uma ferramenta avançada que permite aos administradores criar e atualizar cursos. Os administradores podem adicionar detalhes do curso, como título, descrição, carga horária assim também como se molda para editar os usuários e os próprios cursos.

3.10 certificado

A página "certificado" é responsável por renderizar e disponibilizar o certificado para download. Os usuários podem visualizar e baixar o certificado de conclusão do curso que obtiveram.

Essas páginas fornecem uma interface intuitiva e amigável para que os usuários possam explorar os cursos, gerenciar suas matrículas, validar certificados e acessar informações relevantes sobre os cursos e usuários.

4. deploy

Nesta seção, abordaremos o processo de implantação do SGCURSOS, que é a etapa final para colocar nossa aplicação em funcionamento. A implantação refere-se ao processo de disponibilizar a aplicação em um ambiente de produção, onde ela estará pronta para ser acessada e utilizada pelos usuários.

Ao realizar o deploy do SGCURSOS, estamos garantindo que a aplicação esteja acessível de forma segura, confiável e escalável. Isso envolve a configuração de servidores, infraestrutura de rede, banco de dados e outros componentes necessários para que a aplicação funcione corretamente.

Durante este processo, exploraremos diferentes aspectos da implantação, como a utilização de contêineres Docker e o uso do Docker Compose para orquestrar os serviços. Também veremos como a AWS EC2 pode ser uma opção viável para hospedar nossa aplicação, oferecendo escalabilidade e disponibilidade.

4.1 microsserviços

Os microsserviços são uma abordagem arquitetural utilizada na aplicação SGCURSOS para dividir a funcionalidade em serviços independentes e altamente coesos. Cada micro serviço é responsável por uma parte específica da aplicação, o que permite uma maior escalabilidade e flexibilidade.

A aplicação SGCURSOS consiste em três microsserviços principais:

1. Backend: O microsserviço Backend é responsável pela lógica de negócio da aplicação. Ele é construído com base no framework Node.js e utiliza a biblioteca Express para fornecer uma API RESTful. Esse micro serviço se comunica com o banco de dados

PostgreSQL para armazenar e recuperar dados. A interação entre o Backend e o banco de dados ocorre por meio de requisições HTTP.

2. Postgres: O micro serviço Postgres é um banco de dados PostgreSQL, que armazena os dados da aplicação. Ele é executado em um contêiner separado e se integra ao micro serviço Backend por meio da rede definida no Docker Compose. Essa separação permite que o banco de dados seja escalado independentemente do Backend e oferece maior segurança e isolamento dos dados.
3. Nginx: O microsserviço Nginx é um servidor web utilizado para servir os arquivos estáticos do Frontend e atuar como um proxy reverso para o Backend. Ele gerencia as requisições HTTP recebidas, encaminhando-as para o micro serviço adequado. O Nginx também fornece suporte para SSL/TLS, permitindo conexões seguras com a aplicação.

Esses microsserviços interagem entre si por meio de requisições HTTP. O Frontend faz requisições para o Backend, que processa as solicitações e retorna as respostas apropriadas. O Backend, por sua vez, acessa o banco de dados Postgresql para armazenar e recuperar os dados necessários.

As vantagens dessa abordagem de microsserviços são várias:

- Escalabilidade: Como cada micro serviço é independente, é possível escalar apenas os serviços que exigem mais recursos, em vez de dimensionar a aplicação como um todo. Isso permite um melhor aproveitamento dos recursos disponíveis.
- Flexibilidade: Os microsserviços podem ser desenvolvidos, testados e implantados independentemente uns dos outros. Isso proporciona maior agilidade no desenvolvimento, permitindo que equipes trabalhem em diferentes partes da aplicação sem afetar o funcionamento global.
- Manutenção facilitada: Com microsserviços separados, é mais fácil fazer alterações ou corrigir problemas em um serviço específico sem afetar o funcionamento dos demais. Isso simplifica a manutenção e o gerenciamento do sistema como um todo.
- Segurança e isolamento: A separação dos microsserviços em contêineres individuais proporciona um nível de isolamento e segurança. Cada microsserviço tem sua própria instância e pode ter políticas de acesso e permissões específicas. Isso reduz o risco de vazamento de dados ou comprometimento do sistema.

Em resumo, os microsserviços na aplicação SGCURSOS permitem uma arquitetura flexível, escalável e modular. Cada micro serviço desempenha um papel específico na aplicação, interagindo por meio de requisições HTTP. Essa abordagem traz benefícios como escalabilidade seletiva, flexibilidade de desenvolvimento, manutenção facilitada e segurança aprimorada.

4.2 docker-compose

No arquivo Docker Compose (`docker-compose.yml`), são definidas as configurações para controlar o ambiente de execução de uma aplicação composta por vários serviços. Algumas das configurações mais comuns são:

Services: Define os diferentes serviços da aplicação, como bancos de dados, servidores web, APIs, etc. Cada serviço é listado com um nome único e pode ser configurado individualmente.

Image: Especifica a imagem Docker a ser usada para construir o contêiner do serviço. A imagem pode ser buscada de um registro público (como o Docker Hub) ou pode ser uma imagem personalizada criada localmente.

Build: Permite especificar o caminho do Dockerfile e quaisquer recursos adicionais necessários para construir uma imagem personalizada para o serviço.

Ports: Define as portas que serão expostas pelo contêiner e mapeadas para as portas do host. Isso permite o acesso aos serviços por meio de portas específicas no host.

Volumes: Permite especificar volumes para persistir dados ou compartilhar arquivos entre o host e o contêiner. Os volumes podem ser usados para armazenar bancos de dados, arquivos de configuração, arquivos de log, entre outros.

Depends_on: Define as dependências entre os serviços. Isso garante que os serviços dependentes sejam iniciados antes dos serviços que dependem deles.

Env_file: Permite especificar um arquivo de ambiente (como `.env`) que contém variáveis de ambiente a serem definidas no contêiner.

Networks: Define as redes que os serviços fazem parte. As redes permitem que os contêineres se comuniquem entre si por meio de nomes de serviço, em vez de terem que lidar diretamente com IPs.

Essas configurações são essenciais para definir as características de cada serviço, como a imagem a ser usada, as portas expostas, as dependências, as variáveis de ambiente e as redes. Com o Docker Compose, é possível simplificar a implantação e a execução de uma aplicação composta por vários contêineres Docker, fornecendo um ambiente controlado e consistente para os serviços da aplicação.

4.2.1 variáveis de ambiente

As variáveis mencionadas no arquivo `.env` têm o objetivo de fornecer informações de configuração necessárias para o funcionamento da aplicação. Veja abaixo uma explicação de cada uma delas:

- **API_PROTOCOL**: Especifica o protocolo a ser usado para acessar a API. Nesse caso, está definido como "http".
- **API_URL**: Define o endereço IP ou o nome do host onde a API está sendo executada. É necessário substituir "host_ip" pelo endereço real.
- **API_PORT**: Determina a porta em que a API está sendo executada. Neste caso, está definida como 3000.
- **DB_PORT**: Especifica a porta para a conexão com o banco de dados PostgreSQL. É definida como 5432, que é a porta padrão para o PostgreSQL.
- **DB_HOST**: Define o host do banco de dados. No exemplo, está configurado como "postgres", que é o nome do serviço definido no arquivo Docker Compose.
- **POSTGRES_DB**: Especifica o nome do banco de dados a ser usado pela aplicação. Neste caso, é definido como "sgcursos".
- **POSTGRES_USER**: Define o nome de usuário para autenticação no banco de dados PostgreSQL. É definido como "sgcursos".
- **POSTGRES_PASSWORD**: Especifica a senha para autenticação no banco de dados PostgreSQL. Neste caso, está definido como "sgcursos".
- **SECRET_TOKEN**: É uma chave secreta usada para assinar e verificar tokens JWT (JSON Web Tokens). No exemplo, foi gerado um valor aleatório com o comando `openssl rand -hex 32`.

- **EXPIRE_IN**: Define o tempo de expiração para os tokens JWT. Neste caso, está configurado como 6 horas.
- **SECRET_CERTIFICATE**: É uma chave secreta usada para fins de certificado. No exemplo, foi gerado um valor aleatório com o comando `openssl rand -hex 16`.

Para utilizar essas variáveis de configuração, você precisa definir os valores corretos para cada uma delas, de acordo com o ambiente de execução da sua aplicação.

4.2.2 Execução do compose

1. Certifique-se de ter o Docker e o Docker Compose instalados em seu sistema.
2. Abra o terminal ou prompt de comando e navegue até o diretório onde deseja clonar o repositório.
3. Execute o comando abaixo para clonar o repositório para o seu diretório local:

```
git clone https://github.com/Kaioguilherme1/SGCURSOS.git
```

4. Após o término do processo de clonagem, acesse o diretório do projeto:

```
cd SGCURSOS
```

5. Agora, você deve ter acesso aos arquivos do projeto. Certifique-se de ter o arquivo `.env` preenchido corretamente com as configurações necessárias, como as variáveis de ambiente mencionadas anteriormente.
6. Com o Docker Compose instalado e o diretório do projeto acessado, execute o comando abaixo para iniciar a construção e a execução dos contêineres definidos no Docker Compose:

```
docker-compose up -d
```

7. Aguarde até que os contêineres sejam criados e iniciados. Você poderá ver a saída do terminal indicando o progresso do processo.
8. Após a conclusão, você poderá acessar a aplicação em seu navegador usando o endereço IP da máquina ou o domínio configurado, dependendo da configuração da variável `API_URL`. No exemplo fornecido anteriormente, a API estaria disponível em `http://host_ip:80` ou `https://host_ip:443`.

Dessa forma, o Docker Compose será configurado e executado com base nos arquivos fornecidos no repositório, permitindo que você utilize a aplicação SGCURSOS localmente. Certifique-se de ter as dependências necessárias instaladas e as configurações corretas para garantir um funcionamento adequado.

4.3 aws EC2

A Amazon Elastic Compute Cloud (EC2) é um serviço de computação em nuvem que oferece capacidade de processamento redimensionável na nuvem da Amazon Web Services (AWS). Para fazer o deploy da aplicação SGCURSOS na AWS EC2, siga os passos abaixo:

1. **Crie uma instância EC2**: Acesse o painel da AWS e crie uma instância EC2 usando a AMI desejada.

2. **Configure as regras de segurança**: Na configuração da instância EC2, adicione as regras de segurança para permitir o tráfego nas portas necessárias. As portas e seus protocolos associados são:

Porta	Protocolo
80	TCP
443	TCP
3000	TCP

3. **Faça o download da chave SSH**: Ao criar a instância EC2, você terá a opção de criar ou utilizar uma chave SSH existente. Faça o download da chave e mantenha-a em um local seguro.

4. **Conecte-se à instância via SSH**: Abra o terminal e execute o seguinte comando para se conectar à instância EC2 usando a chave SSH:
   bash

```
ssh -i /caminho/para/chave-ssh.pem
usuário@endereço-ip-da-instância
```

1. Certifique-se de substituir `/caminho/para/chave-ssh.pem` pelo caminho correto para a chave SSH que você baixou e `usuário@endereço-ip-da-instância` pelo usuário e endereço IP da sua instância EC2.
2. **Configure o ambiente**: Dentro da instância EC2, navegue até a pasta da aplicação SGCURSOS e configure as variáveis de ambiente necessárias, como as informações de banco de dados, URL da API, segredos do JWT, etc, conforme no tópico anterior.
3. **Execute o Docker Compose**: Ainda na instância EC2, execute o comando `docker-compose up -d` para iniciar os contêineres da aplicação SGCURSOS.
4. **Verifique o acesso**: Após o tempo de inicialização dos contêineres, verifique se a aplicação está funcionando corretamente acessando a URL pública da sua instância EC2.

O uso do SSH para o deploy da aplicação SGCURSOS na AWS EC2 proporciona segurança e controle no acesso à instância, permitindo a configuração adequada das portas para o tráfego necessário. Além disso, o uso de contêineres Docker e o Docker Compose facilitam a implantação da aplicação, garantindo a consistência do ambiente de execução e a escalabilidade do sistema.

Ao seguir esses passos, você poderá fazer o deploy da aplicação SGCURSOS na AWS EC2 e disponibilizar a aplicação online de forma segura e confiável.