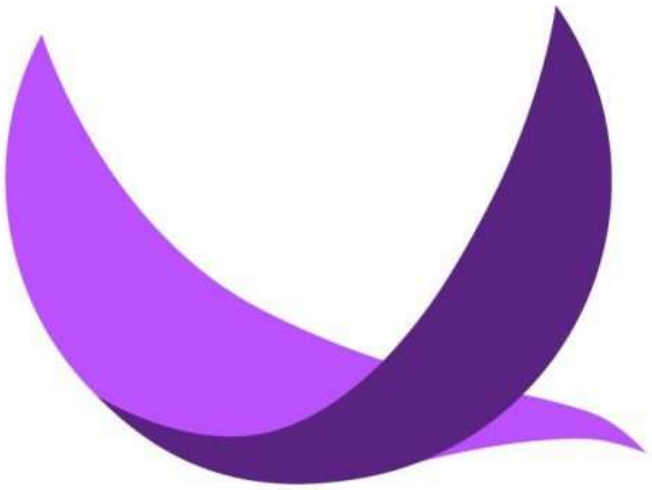


samplemed

Relatório de execução do exercício

(por: Luiz **Gustavo** Costa de **Menezes**)

Abaixo, se encontram as principais informações sobre a realização do "**Exercício Samplemed",** referente ao teste técnico.

Como solicitado, foi criado um MPV, referente à um blog, tendo como principais funções:

- Cadastro de Usuários;
- Cadastro de Artigos;
- Sistema de Login ;
- Autenticação em todas as chamadas da API;
- Conexão entre Palavras-chave e Artigos;
- Comunicação feita através de API;

Foram utilizadas as seguintes tecnologias:

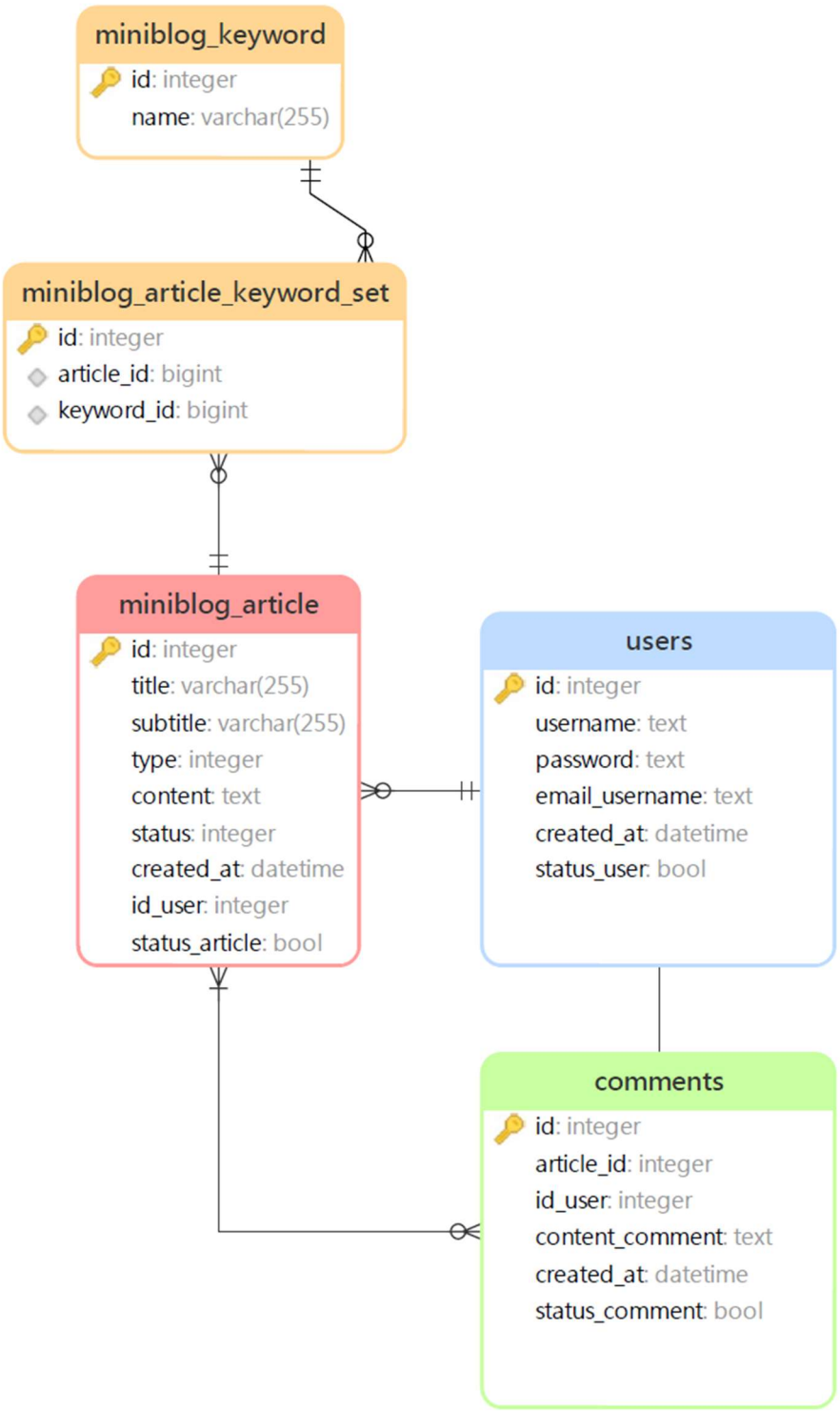
- BACKEND:
  - Python (3.11.3);
  - Django (4.2.5);
  - asgiref(3.7.2);
  - djangorestframework(3.14.0);
  - djangorestframework-simplejwt(5.3.0);
  - PyJWT(2.8.0);
  - pytz(2023.3.post1);
  - sqlparse(0.4.4);
  - tzdata(2023.3);
- FRONTEND:
  - HTML;
  - Javascript;
- DATABASE:
  - SQLite 3;

ARQUITETURA

Utilizando o banco de dados SQLite para o MVP, foram criadas algumas tabelas como exemplo para demostrar o funcionamento simples e intuitivo para armazenamento de todos os dados gerados no MVP.

PS.: Em um ambiente de produção, um banco de dados ideal para suportar um tráfego de dados maior, seria o PostgreSQL:

Abaixo segue o diagrama de exemplo do MVP:



O usuário após o cadastro, utilizando o próprio sistema do DJANGO, poderá ver e adicionar novos artigos e comentários.

Abaixo segue a estrutura do banco de dados:

comments

	Field	Type	Extra
P	id	integer	Auto Increment
	article_id	integer	
	id_user	integer	
	content_comment	text	
	created_at	datetime	
	status_comment	bool	

miniblog_article

	Field	Type	Extra
P	id	integer	Auto Increment
	title	varchar(255)	
	subtitle	varchar(255)	
	type	integer	
	content	text	
	status	integer	
	created_at	datetime	
	id_user	integer	
	status_article	bool	
	created_users	integer	

miniblog_article_keyword_set

	Field	Type	Extra
P	id	integer	Auto Increment
	article_id	bigint	
	keyword_id	bigint	

Index	Fields	Extra
miniblog_article_keyword_set_article_id_07126345	article_id	
miniblog_article_keyword_set_article_id_keyword_id_b08a48b8_uniq	article_id, keyword_id	Unique
miniblog_article_keyword_set_keyword_id_18509609	keyword_id	

miniblog_keyword

	Field	Type	Extra
P	id	integer	Auto Increment
	name	varchar(255)	

sqlite_sequence

	Field	Type	Extra
	name		Allow Null
	seq		Allow Null

PRÁTICAS PARA O BOM FUNCIONAMENTO DA APLICAÇÃO

Resiliência

- Utilizando o sistema de micro serviços, é possível manter a aplicação mesmo se algum vier a apresentar problemas;
- Tratamento de exceções utilizando blocos ‘try...except’, para orientar a aplicação de como tratar os possíveis cenários;
- Monitoramento de disponibilidade e desempenho da aplicação, para prever possíveis sobrecargas;
- Balanceamento de carga em caso em que a demanda esteja alta, afim de manter a aplicação disponível;

Performance

- Consultas SQL:
    - Criação de índices nas tabelas para otimizar as consultas;
    - Analisar cada consulta SQL afim de identificar gargalos e possíveis retornos não desejados;

    - Utilização de ferramentas Profiling, visando medir o tempo de execução de cada método.
- Cache:
    - Montar estratégias para armazenar em cache, dados que são frequentemente lidos, evitando assim, sobrecarga no banco de dados;
    - Podemos utilizar o “Django Cache Framework”, a fim de implementar o armazenamento em cache
- Paginação:
    - Definir limites nas consultas SQL para que sejam retornados apenas os dados necessários.
- Compressão de Recursos:
    - Comprimir todos os recursos estáticos, como templates, imagens e arquivos .js para reduzir tempo de carregamento e disponibilidade da aplicação:

Segurança

- Autenticação e Autorização:
    - Utilização do JWT para autenticação e acesso às APIs;
    - Utilização de ferramentas do DJANGO para controle de acesso e permissão dos usuários.
- Validação de entrada:
    - Validar todas as entradas do usuário, visando evitar SQL Injection e XSS (Cross-Site Scripting).

- Proteção contra CSRF:
    - Utilizando ferramentas do DJANGO, para proteger a aplicação de ataques CSRF (Cross-Site Request Forgery).
- Limite de Taxa:
    - Utilização uma limitação de requisições por IP em um determinado período de tempo, visando proteger contra ataques DDOS e negação de serviço.

Simultaneidade

- Thread Safety:
    - Tornar o código thread-safety em locais onde podem ocorrer múltiplas threads simultaneamente.
- Escalabilidade:
    - Ao montar a arquitetura da aplicação é necessário observar a escalabilidade horizontal, a fim de distribuir entre vários servidores, todas as requisições.

- Mensageria:
    - A depender do fluxo de informações transacionadas, é interessante utilizar uma fila de mensagens para reduzir a carga no servidor.

Conclusão:

É possível, através de boas práticas e uma arquitetura eficiente da aplicação, torná-la altamente disponível, resistente aos principais ataques e que torne a experiência do usuário, a melhor possível, economizando recursos e tornando-a eficaz, eficiente e lucrativa, com baixo nível de manutenção e facilidade para tal.