Escolhendo uma estrutura moderna de banco de dados e web em Python

Se você deseja criar um aplicativo web em Python, escolher uma estrutura será a primeira decisão técnica que você precisará tomar. Com tantas opções, qual você deve escolher?

A maioria dos aplicativos da web atualmente são aplicativos REST que executam operações CRUD no banco de dados selecionado. Isso significa que muito do código que você deseja escrever é bastante genérico e pode ser bastante simplificado com código genérico de bibliotecas.

Especificamente, procuramos dois tipos de bibliotecas:

1. Uma estrutura da Web
2. Uma estrutura de banco de dados

Como estamos criando um novo aplicativo da web, temos a liberdade de olhar para qualquer coisa, desde o antigo e confiável até o novo e sofisticado. Avaliaremos com base nos seguintes critérios:

- Deve ser simples e extensível
- Deve ter desempenho
- Deve minimizar a quantidade de trabalho que temos que fazer
- Deve usar Python moderno
- Se possível, a biblioteca da Web e do banco de dados deve funcionar bem juntas

Por último, há sempre o fator X, a biblioteca deve funcionar da maneira que gostamos. No final, a maioria dos frameworks e bibliotecas darão conta do recado, então também gostaríamos de usar o framework.

Para o leitor com pouco tempo, há uma tabela comparativa no final do artigo. E o último parágrafo de cada seção também é um resumo de quando usar cada biblioteca.

Os competidores

Para frameworks Web, consideraremos as seguintes bibliotecas e frameworks:

- Django
- Flask
- Sanic
- FastAPI

Para a estrutura de banco de dados, consideraremos as seguintes bibliotecas:

- Django ORM
- SQLAlchemy
- Tortoise ORM

Web Frameworks

DJANGO

https://www.djangoproject.com

Django é um dos maiores e mais antigos frameworks web e tem “baterias incluídas”. Isso significa que ele fornece quase tudo que você precisa para uma estrutura web.

Possui visualizações baseadas em classes e funções, suporta muito bem a criação de páginas de modelo, possui um excelente site de administração no qual você pode conectar-se facilmente ao seu banco de dados sem mexer no SQL.

Além disso, por ser antigo e bem estabelecido, possui uma série de bibliotecas construídas sobre ele. Desde coisas pequenas, como cabeçalhos CORS, até coisas maiores, como integração e agendamento de aipo. Porém, o melhor é Django Rest Framework, que é uma biblioteca que possui uma abordagem REST-first para Django e fornece atalhos para validação, classes genéricas e modelo automático GET (todos e um item), PUT, PATCH, DELETE e até HEAD integração. Isso faz com que seja a maneira correta de usar o Django atualmente, na minha opinião.

No entanto, nem tudo favorece Django. Uma das críticas mais comuns é que o Django é muito opinativo e ensina como gerenciar seu código. Embora eu pessoalmente não siga essa linha de pensamento, para pequenas aplicações da web isso pode ser um pouco exagerado. Em segundo lugar, o Django é lento. E por último, um ponto negativo pessoal para mim é que Django parece velho. Ele não usa muitos recursos modernos do Python. Falta suporte para digitação (olhando para seus QuerySets), async ainda não está incluído e o Django tem uma tendência a duplicar coisas normais do Python (para ser justo, elas não foram incluídas quando o Django as criou).

Para mim, Django é uma escolha muito sólida para qualquer grande projeto com foco em confiabilidade e para projetos CRUD muito pequenos onde você não quer se preocupar com uma grande configuração de banco de dados.

FLASK

https://flask.palletsprojects.com

Flask é o micro framework web mais popular que existe. É leve, fácil de começar e não ensina como codificar do seu jeito. De certa forma, é o oposto do Django.

O Flask geralmente é usado com visualizações baseadas em funções, embora também suporte visualizações baseadas em classes. Suporta Jinja nativamente. Não fornece um ORM de banco de dados, deixando você livre para escolher o que quiser.

Como um dos frameworks mais populares, também possui uma tonelada de bibliotecas, incluindo integrações com SQLAlchemy, validações automáticas, autenticação e middleware de segurança, etc. Por ser um framework web puro, torna mais fácil seguir um bom design REST.

Isso não significa que não haja desvantagens. Como uma estrutura de micro web, ele fornece apenas manipulação HTTP básica pronta para uso. Embora isso possa ser um benefício, para criar endpoints REST significa que você precisa adicionar muitas coisas manualmente, como validação e documentação. Em segundo lugar, ele sofre dos mesmos problemas do Django. É bastante lento e, novamente na minha opinião, parece velho. Ele quase não usa nenhum dos recursos modernos do Python. Embora suas bibliotecas compensem algumas de suas falhas, até mesmo a maioria das bibliotecas são antigas (mas testadas e validadas).

Flask é uma escolha sólida para aplicativos focados na Web de pequeno e médio porte, para os quais bibliotecas úteis e fáceis de usar são as principais preocupações.

SANIC
https://github.com/huge-success/sanic

Sanic é uma estrutura web assíncrona mais recente que se concentra tanto na velocidade técnica quanto na velocidade de desenvolvimento. Embora seu logotipo antigo possa não ter dado muita confiança, é uma das melhores estruturas assíncronas que existem.

Do ponto de vista do desenvolvimento, ele se parece muito com o Flask. Ele promove visualizações baseadas em funções, mas também oferece suporte a visualizações baseadas em classes. Parâmetros de solicitação, etc., são encontrados nas solicitações de função.

Em termos de velocidade, seu homônimo não mente e oferece a maior velocidade dos frameworks aqui considerados. Junto com uma forma assíncrona de trabalhar, vai te ajudar muito a escalar facilmente. Como uma estrutura emergente, possui algumas bibliotecas que cobrem os casos de uso mais comuns, embora não tanto quanto Django ou Flask.

Do outro lado da moeda, ele sofre de desvantagens semelhantes às do Flask. Seu foco em micro requer sobrecarga extra para bons endpoints REST com validação, etc. E como uma comunidade menor, não terá o mesmo suporte que o Flask teria.

Escolha Sanic se precisar de um aplicativo web de alto desempenho ou se quiser uma versão assíncrona do Flask.

FAST API
https://fastapi.tiangolo.com

Por último, para os frameworks web, temos FastAPI. FastAPI é uma estrutura web assíncrona mais recente que se concentra na integração da digitação na base de código REST. É uma combinação de Starlette e Pydantic em uma estrutura web.

O estilo de código é semelhante ao Flask com visualizações baseadas em funções. Os modelos Pydantic validam automaticamente as solicitações e fornecem as respostas de erro corretas ao usuário, se necessário. Tudo isso é criado em Python digitado que permite detectar erros em seu IDE e com mypy.

Como o próprio nome sugere, também é rápido, apenas um pouco mais lento que o Sanic. Funciona tanto de forma assíncrona quanto sincronizada, dependendo do estilo do seu código. Com os modelos Pydanctic, ele também gera automaticamente uma documentação Swagger e Redoc. Por ser um novo framework, sua lista de integrações não é enorme, mas possui algumas integrações muito úteis, incluindo SQLAlchemy e Tortoise ORM.

FastAPI brilha quando você está procurando uma estrutura web onde a validação de entrada será uma parte importante e deseja melhorar a qualidade do código com a digitação.

Database frameworks

DJANGO ORM

Django também possui um Mapeador de Relacionamento de Objetos (ORM, Object-Relationship Mapper), que mapeia seu modelo de código para um modelo de banco de dados. Como parte da estrutura da web, é claro, ele se integra perfeitamente ao restante da estrutura.

Trabalhar com o Django ORM é muito fácil. Definir classes é simples. Basta definir algo como modelo de banco de dados e dizer quais campos ele precisa ter. Ele suporta campos diretos e também chaves estrangeiras, relacionamentos muitos para muitos e até mesmo um para um. Usar modelos também é muito fácil. As teclas diretas reversas são importadas para o outro modelo, portanto podem ser acessadas diretamente e tudo funciona muito bem.

Além disso, uma das minhas partes favoritas é a estrutura de migração que cria automaticamente a lógica para criar e ajustar suas tabelas SQL. Embora a solução pronta para uso nem sempre seja perfeita, suas suposições estão corretas na maioria das vezes e você sempre pode ajustar manualmente as atualizações.

Por outro lado, o Django ORM traz consigo todo o Django. Usar apenas a parte ORM parece um pouco exagerado, pois você ainda precisará integrar o Django como um todo. Existem também algumas peculiaridades para evitar consultar seus dados com muita frequência. E assim como o Django, está começando a parecer bastante antigo e não suporta digitação ou assíncrono.

Use o Django ORM quando estiver usando o Django como sua estrutura web. Para outros casos, eu não recomendaria de imediato, embora tenha gostado muito de trabalhar com ele.

SQLAlchemy
https://www.sqlalchemy.org

SQLAlchemy é o maior sistema ORM em Python, e por um bom motivo. É extenso, cobrirá praticamente qualquer caso de uso e oferece suporte a todos os bancos de dados existentes.

Ele é dividido em uma parte principal e uma parte ORM, que ajuda na conexão com qualquer banco de dados que você deseja suportar e na configuração das tabelas. O ORM é onde a mágica acontece, ajudando você a evitar a escrita de consultas SQL delicadas. Na minha opinião, a forma de pensar lembra a lógica SQL, juntando tabelas manualmente. Isso permite que você otimize muito bem suas consultas.

Embora as migrações não sejam suportadas nativamente, a biblioteca Alembic baseia-se nisso no SQLAlchemy. Ele também pode gerar automaticamente as migrações para você e, na maioria das vezes, adivinhará a maneira correta de mover-se entre os estados do banco de dados. Por último, como a maior biblioteca ORM, há inúmeras bibliotecas construídas sobre ela, ajudando você em muitos casos de uso.

Do lado negativo, é complicado trabalhar com isso. Seu estilo SQL não combina tão bem com o estilo Python. Por causa disso, a curva de aprendizado é maior para SQLAlchemy do que, por exemplo, para Django. A digitação não está presente nativamente (embora tenha um stub mypy) e o assíncrono é atualmente um recurso beta.

Se você está procurando um ORM sólido com o qual muitas pessoas estão familiarizadas e que será capaz de lidar com qualquer caso extremo que você apresentar, opte pelo SQLAlchemy.

Tortoise ORM
https://github.com/tortoise/tortoise-orm

Tortoise ORM é uma nova biblioteca ORM assíncrona inspirada no Django. Embora ainda esteja em estado beta (0,16 no momento em que este artigo foi escrito), ele já possui os principais recursos que você espera de um ORM.

Inspirado no Django, possui uma curva de aprendizado baixa, principalmente se você tiver alguma experiência com Django. Ele também suporta relações de chaves estrangeiras, muitos para muitos e um para um, e as chaves reversas também são importadas para o outro modelo.

Como um dos únicos ORMs assíncronos, ele se encaixa perfeitamente em uma pilha assíncrona moderna. Ele também suporta digitação nativa, incluindo seus QuerySets genéricos. Por enquanto, os relacionamentos reversos ainda não são digitados nativamente, mas a adição de digitações manuais é suportada sem prejudicar a lógica.

Por outro lado, por ser uma estrutura relativamente nova, ela não oferece suporte a todos os casos de uso existentes. Ainda não possui um quadro de migração (embora pretenda criar um). Isso exige que você faça um esforço manual extra em SQL para manter suas tabelas atualizadas. Além disso, leva algum tempo para se acostumar com a sintaxe assíncrona para garantir que você obtenha

o que deseja. Por outro lado, deixa muito claro quais dados estão presentes e quando você precisa consultar o banco de dados, que é ofuscado, por exemplo, no Django.

Selecione Tortoise ORM se você estiver criando uma pilha assíncrona. Como ainda está em versão beta, ainda não posso recomendá-lo totalmente para uso em produção pesada, mas é uma ótima ferramenta para projetos pessoais menores.

Conclusão

Todas essas estruturas são escolhas sólidas. Frameworks mais antigos podem não tirar proveito da mais nova sintaxe ou novidades do Python, mas são confiáveis, bem testados e conhecidos por toda a comunidade. Você encontrará suporte com muito mais facilidade. As estruturas mais recentes usam os recursos mais recentes, ajudando você com qualidade ou velocidade do código, mas têm menos integrações e o suporte será mais difícil de encontrar. Tudo se resume a preferência pessoal.

Comparação

Web Frameworks

Framework	Django	Flask	Sanic	Fast API
Simples e extensível	Não	Sim	Sim	Sim
Performático	Não	Não	Sim	Sim
Minimiza o trabalho	Sim	Não	Não	Sim
Python moderno	Não	Não	Não	Sim
Integra-se com	Django	SQL Alchemy	SQL Alchemy	-
Vantagem	Otimas Ferramentas	Otimas Ferramentas	O mais rápido	Dá menos trabalho
Desvantagem	Volumoso	Old school	Apenas um Flask mais rápido	Novo e não testado

Bancos de dados

Framework	Django ORM	SQL Alchemy	Tortoise
Simples e extensível	Sim	Não	Sim
Minimiza o trabalho	Sim	Não	Sim
Python moderno	Não	Não	Sim
Integra-se com	Django	Tudo	-
Vantagem	Parte do Django	Suporta tudo	Async moderno
Desvantagem	Volumoso	Complicado	Novo e não testado

Vocabulário

CRUD é a sigla para CREATE, READ, UPDATE e DELETE. Estes termos descrevem as quatro operações essenciais para criar e gerenciar elementos de dados persistentes, principalmente em bancos de dados relacionais e NoSQL.

Referência

MEESTER, Anton De. Escolhendo uma estrutura moderna de banco de dados e web em Python. Publicado em: 5 de novembro de 2020. Disponível em: https://totalwealth.app/choosing-a-modern-python-web-and-database-framework-bc02dfc29b4a