PYTHON ACADEMY

DESENVOLVIMENTO WEB COM PYTHON E DJANGO

O Django é uma framework de desenvolvimento Web Python completa e robusta. Nesse ebook introdutório, vamos abordar os a arquitetura do framework e primeiros passos para o desenvlvimento web com Python.

PYTHONACADEMY.COM.BR

Salve salve Pythonista! 👋

Django é um *framework* de alto nível, escrito em Python que encoraja o desenvolvimento limpo de aplicações *web*.

Desenvolvido por experientes desenvolvedores, Django toma conta da parte pesada do desenvolvimento *web*, como tratamento de requisições, mapeamento objeto-relacional, preparação de respostas HTTP, para que, dessa forma, você gaste seu esforço com aquilo que **realmente interessa**: suas **regras de negócio**!

Foi desenvolvido com uma preocupação extra em **segurança**, evitando os mais comuns ataques, como [Cross site scripting (XSS)][xss-reference], [Cross site request forgery (CSRF)][csrf-reference], [SQL injection][sql-injection-reference], entre outros (veja mais [aqui][django-security]).

É bastante **escalável**: Django foi desenvolvido para tirar vantagem da maior quantidade de hardware possível (desde que você queira). Django usa uma arquitetura "zero-compartilhamento", o que significa que você pode adicionar mais recursos em qualquer nível: servidores de banco de dados, cache e/ou servidores de aplicação.

Para termos uma boa noção do Django como um **todo**, vou fazer a seguinte sequência posts, utilizando uma abordagem *bottom-up* (de baixo pra cima): começaremos com esse *post* mais conceitual, depois passaremos para a **camada de Modelo**, depois veremos a **camada de *Views*** e, por fim, a **camada de *Templates***.

Portando, ajustem a , façam um e **#vamosnessa** 💻!

Introdução

Como disse anteriormente, o Django é um *framework* para construção de aplicações web em Python.

E, como todo *framework* web, ele é um *framework* MVC (***M**odel **V**iew **C**ontroller*), certo? **Não exatamente!**

De acordo com sua documentação, eles o declaram como um **MTV** - isto é: ***M**odel-**T**emplate-**V**iew*. Mas por que a diferença?

No entedimento dos desenvolvedores do Django sobre MVC, a *View* representa **qual** informação você vê, não **como** você vê. Há uma sutil diferença.

Nesse caso, uma *View* é uma forma de processar os dados de uma URL específica, por que esse método descreve qual informação é apresentada.

Além disso, é imprescindível separar **conteúdo** de **apresentação** – que é onde os templates residem. No Django, uma *View* descreve **qual** informação é apresentada, mas uma *View* normalmalmente delega para um template, que descreve **como** a informação é apresentada.

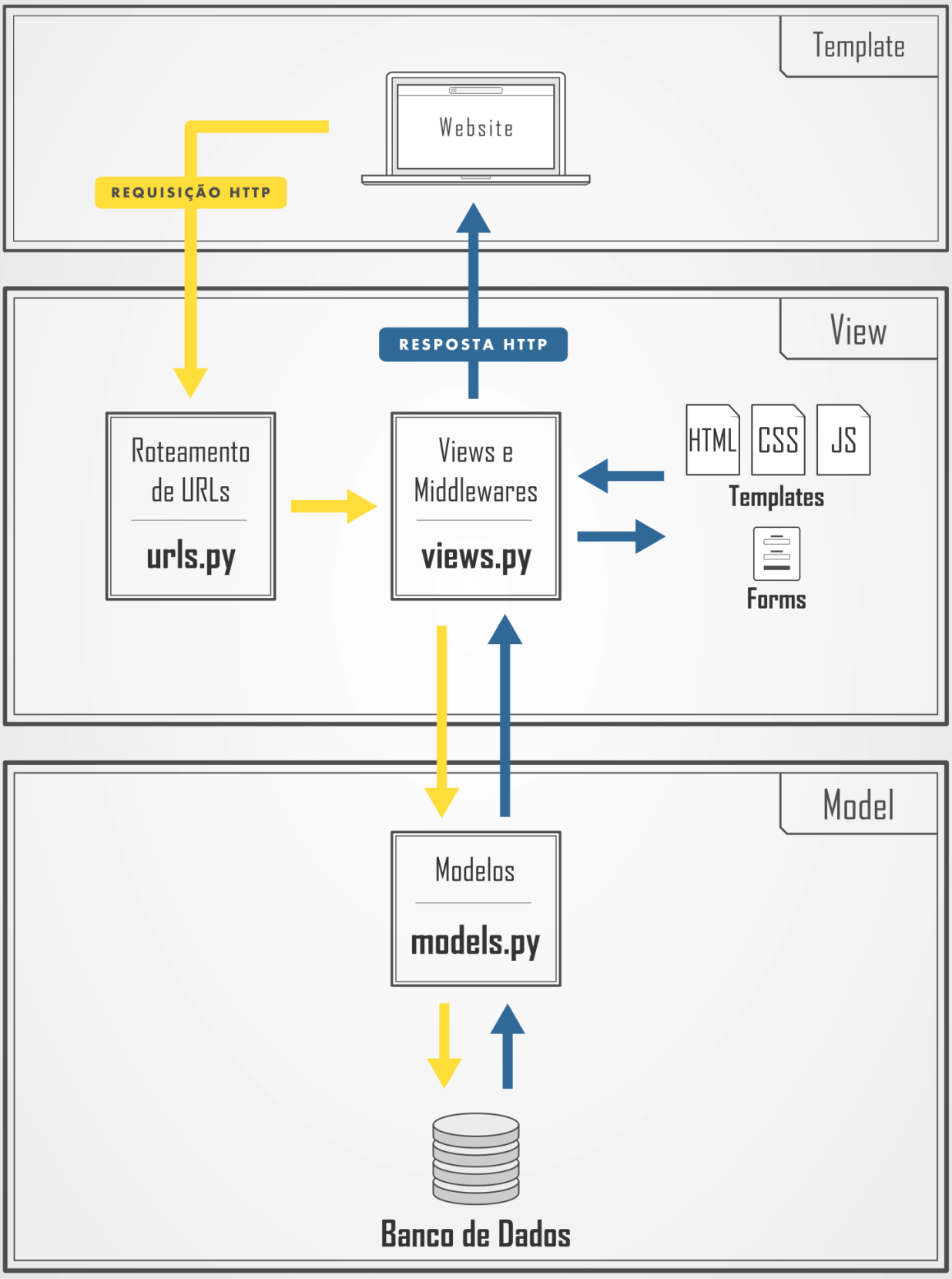
Assim, onde o ***Controller*** se encaixa nessa arquitetura? No caso do Django, é o próprio framework que faz o trabalho pesado de processar e rotear uma requisição para a *View* apropriada de acordo com a configuração de URL descrita pelo desenvolvedor.

Para ajudar a entender um pouco melhor, vamos analisar o fluxo de uma requisição saindo do *browser* do usuário, passando para o servidor onde o Django está sendo executado e retornando ao *browser* do usuário.

Fluxo de uma requisição no Django

A fim de ilustrar o funcionamento interno do Django e o fluxo de uma requisição “em suas entranhas”, eu fiz a seguinte ilustração:

ARQUITETURA DO django



Fluxo da Requisição
Fluxo da Resposta

O Django é dividida em três camadas: camada de modelos, camada de visualização e camada de *templates*.

Conforme disse ali em cima, vamos estudar cada camada em *posts* separados.

Vamos agora, dar nossos primeiros passos com o Django, começando pela sua **instalação**!

Instalação

Vamos agora ao que interessa!

Primeiro, precisamos nos certificar que o Python e o pip (gerenciador de pacotes do Python) estão instalados corretamente.

Vá no seu terminal ou *prompt* de comando e digite `python`. Deve ser aberto o terminal interativo do Python (se algo como `bash: command not found` aparecer, é por que sua instalação não está correta).

Agora, digite `pip --version`. A saída desse comando deve ser a versão instalada do pip. Se ele não estiver disponível, faça o *download do instalador nesse link* e execute o código.

Vamos executar esse projeto em um ambiente virtual utilizando o *virtualenv* para que as dependências não atrapalhem as que já estão instaladas no seu computador.

(*Para saber mais sobre o virtualenv, leia esse post aqui sobre desenvolvimento em ambientes virtuais*)

Após criarmos nosso ambiente virtual, instalamos o Django com:

```
pip install Django
```

Para saber se a instalação está correta, podemos testar abrindo o terminal interativo do Python (digitando `python` no seu terminal/*prompt* de comando) e executando os comandos:

```
>>> import django
>>> print(django.get_version())
```

A saída deve ser a versão do Django instalada.

No momento em que escreve esse post, a versão é:

```
3.2.1
```

Se durante esse processo algum erro ocorrer, não tenha medo de postar no **box de comentários** aqui embaixo sua dúvida! Estamos aqui pra aprendermos juntos!

Hello world, Django!

Com tudo instalado corretamente, vamos agora fazer um mini-projeto para que você veja o Django em ação!

Nosso projeto é fazer um CRUD simples de Funcionários. Ou seja, vamos fazer uma aplicação onde será possível adicionar, listar, atualizar e deletar Funcionários.

Vamos começar criando a estrutura de diretórios e arquivos principais para o funcionamento do Django. Para isso, o pessoal do Django fez um comando muito bacana pra nós: o `django-admin.py`.

Se sua instalação estiver correta, esse comando foi adicionado ao seu PATH! Tente digitar `django-admin --version` (ou `django-admin.py --version`) no seu terminal.

Digitando apenas `django-admin`, é esperado que aparece a lista de comandos disponíveis, similar a:

```
Available subcommands:

[django]
    check
    compilemessages
    createcachetable
    dbshell
    diffsettings
    dumpdata
    flush
    inspectdb
    loaddata
    makemessages
    makemigrations
    migrate
    runserver
    sendtestemail
    shell
    showmigrations
    sqlflush
    sqlmigrate
    sqlsequencereset
    squashmigrations
    startapp
    startproject
    test
    testserver
```

Por ora, estamos interessados no comando `startproject` que cria um novo projeto com a estrutura de diretórios certinha para começarmos a desenvolver!

Executamos esse comando da seguinte forma:

```
django-admin.py startproject helloworld
```

Criando a seguinte estrutura de diretórios:

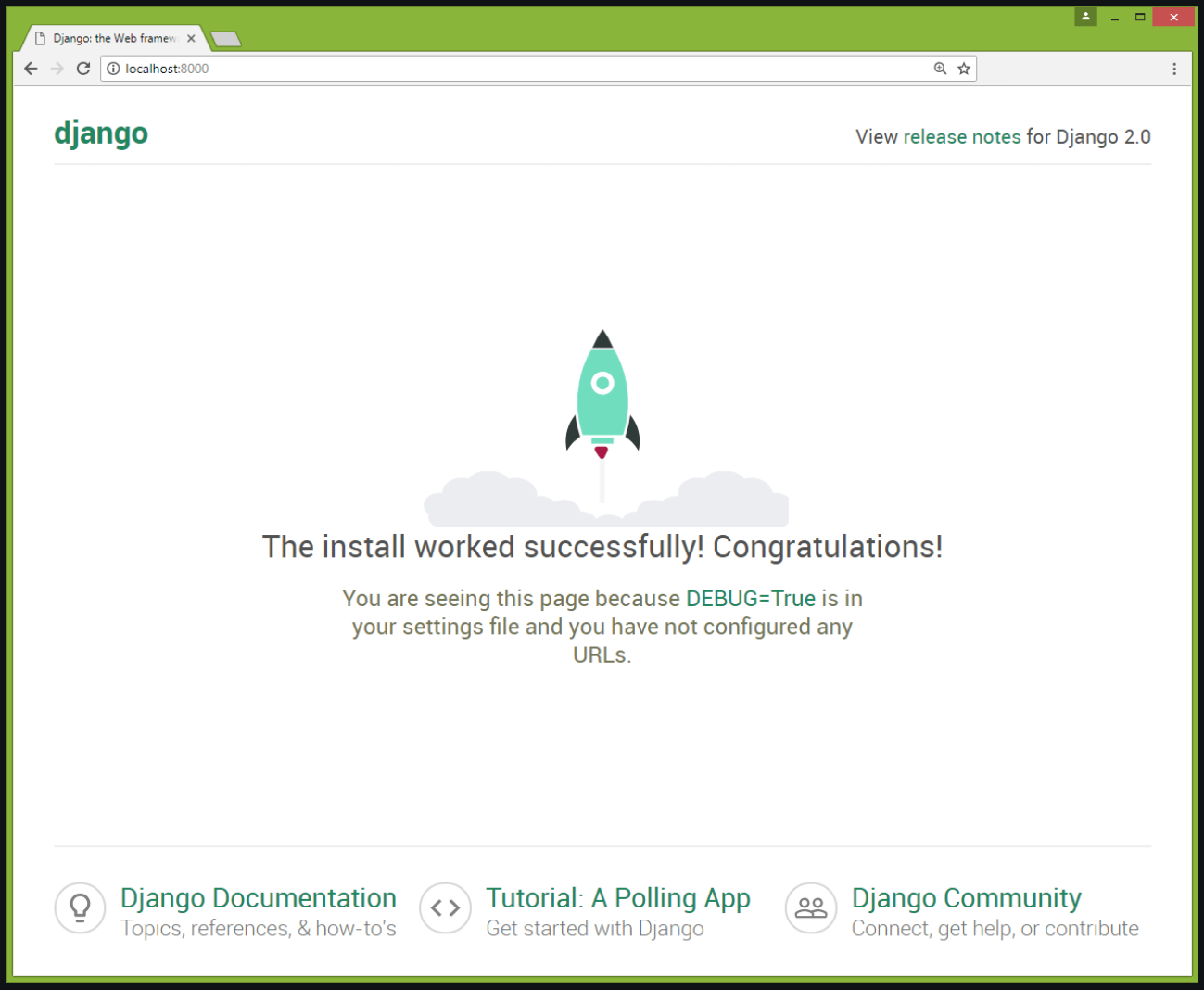
```
- helloworld
    - __init__.py
    - settings.py
    - urls.py
    - wsgi.py
- manage.py
```

Explicando cada arquivo:

- `helloworld/settings.py`: Arquivo muito importante com as configurações do nosso projeto, como configurações do banco de dados, aplicativos instalados, configuração de arquivos estáticos e muito mais.
- `helloworld/urls.py`: Nossa `URLConf` - aqui vamos dizer ao Django **quem** responde a **qual** URL.
- `helloworld/wsgi.py`: Aqui configuramos a interface entre o servidor de aplicação e nossa aplicação Django.
- `manage.py`: Arquivo gerado automaticamente pelo Django que expõe comandos importantes para manutenção da nossa aplicação.

Por exemplo, estando na pasta raíz do projeto, execute o comando `python manage.py runserver`. Depois, acesse seu *browser* no endereço `http://localhost:8000`.

A seguinte tela deve ser mostrada:



Se ela aparecer, significa que nossa configuração está **correta** e o Django está pronto para começarmos a desenvolver!

> 💡 *Falando em Django, estou construindo o **DevBook** exatamente com Django! O DevBook é uma plataforma que usa IA para gerar ebooks técnicos profissionais. Não deixe de conferir!*

Agora, vamos criar um `app` chamado `website` para separarmos os arquivos de configuração da nossa aplicação, que vão ficar na pasta `/helloworld`, dos arquivos relacionados ao *website*.

De acordo com a documentação, um `app` no Django é:

> *"Uma aplicação Web que faz **alguma coisa**, por exemplo - um blog, um banco de dados de registros públicos ou um aplicativo de pesquisa. Já um projeto é uma coleção de configurações e apps para um website em particular."*

Um projeto pode ter vários `apps` e um `app` pode estar presente em diversos projetos.

A fim de criar um novo `app`, o Django provê outro comando, chamado `django-admin.py startapp` que nos ajuda a criar os arquivos e diretórios necessários para tal objetivo.

Na raíz do projeto, execute:

```
django-admin.py startapp website
```

Assim, nossa estrutura de diretórios deve estar similar a:

```
- helloworld
    - __init__.py
    - settings.py
    - urls.py
    - wsgi.py
- website
    - migrations/
    - __init__.py
    - admin.py
    - apps.py
    - migrations.py
    - models.py
    - tests.py
    - views.py
- manage.py
```

Para que o Django gerencie esse novo app, é necessário adicioná-lo a **lista de `apps` instalados**. Fazemos isso no arquivo `helloworld/settings.py`.

Conforme disse ali em cima, esse arquivo contém diversas configurações do seu projeto Django. Uma delas (e muito importante) é a `INSTALLED_APPS`.

Ela é uma lista e diz ao Django o conjunto de `apps` que devem ser gerenciados pelo Django para cada projeto.

Como acabamos de criar o `app` website e já havia o `app` principal `helloworld`, é necessário adicioná-los a essa lista. Portanto, procure por:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

e adicione `website` e `helloworld`:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'helloworld',
    'website'
]
```

Agora, vamos fazer algumas alterações na estrutura do projeto.

Primeiro, vamos passar o arquivo de modelos `models.py` de `/website` para `/helloworld`, pois os arquivos comuns ao projeto vão ficar centralizados no app `helloworld` (geralmente temos apenas um arquivo `models.py` por projeto).

Podemos, então, excluir a pasta `/migrations` e o arquivo `migrations.py` da pasta `/website`, pois estes serão gerados e gerenciados pelo app `helloworld`.

Por fim, devemos estar com a estrutura de diretórios da seguinte forma:

```
- helloworld
    - __init__.py
    - settings.py
    - urls.py
    - wsgi.py
    - models.py
- website
    - __init__.py
    - admin.py
    - apps.py
    - tests.py
    - views.py
- manage.py
```

Quer comparar com a minha versão, [clique aqui para fazer o download do projeto][download-project]!

Com isso, terminamos nossa primeira lição! Mas calma que temos muito ainda para ver!

Espero que tenha dado tudo certo! Se não, por favor, poste no box de comentários aqui embaixo o que aconteceu que nós damos um jeito!

Conclusão

Nesse artigo, vimos um pouco sobre o Django, suas principais características e como começar a desenvolver utilizando-o!

Espero que tenham gostado e que eu tenha despertado a vontade de aprender mais sobre esse excelente *framework*!

O próximo *post* trata da Camada *Model* do Django (acesse-o AGORA aqui!), que é onde residem as entidades do nosso sistema e toda a lógica de acesso a dados!

Quer levar esse conteúdo para onde for com nosso **ebook GRÁTIS**?

Então aproveita essa chance 👇👇👇

Fiquem ligados e até a próxima!

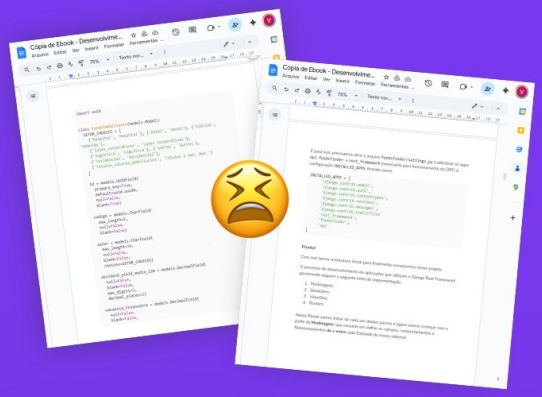
[xss-reference]//www.owasp.org/index.php/Cross-site_Scripting_(XSS) [csrf-reference]//www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF) [sql-injection-reference]//www.owasp.org/index.php/SQL_Injection [django-security]//docs.djangoproject.com/en/4.2/topics/security/

Não se esqueça de conferir!

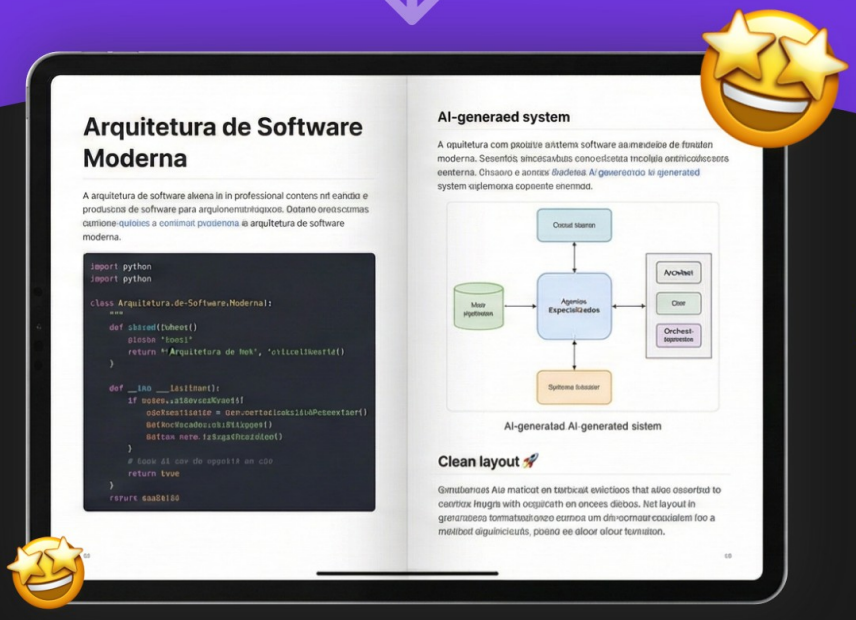
DevBook

Crie Ebooks técnicos em minutos com IA

Conheça a 1ª IA Especializada na criação de Ebooks **com código**!



Chega de formatar código no Google Docs



Syntax Highlight

Infográficos feitos por IA

Adicione Banners Promocionais

Deixe que nossa IA faça o trabalho pesado

Edite em Markdown em Tempo Real

TESTE AGORA

PRIMEIRO CAPÍTULO 100% GRÁTIS