Este é o CS50

Introdução à Ciência da Computação (CS50)

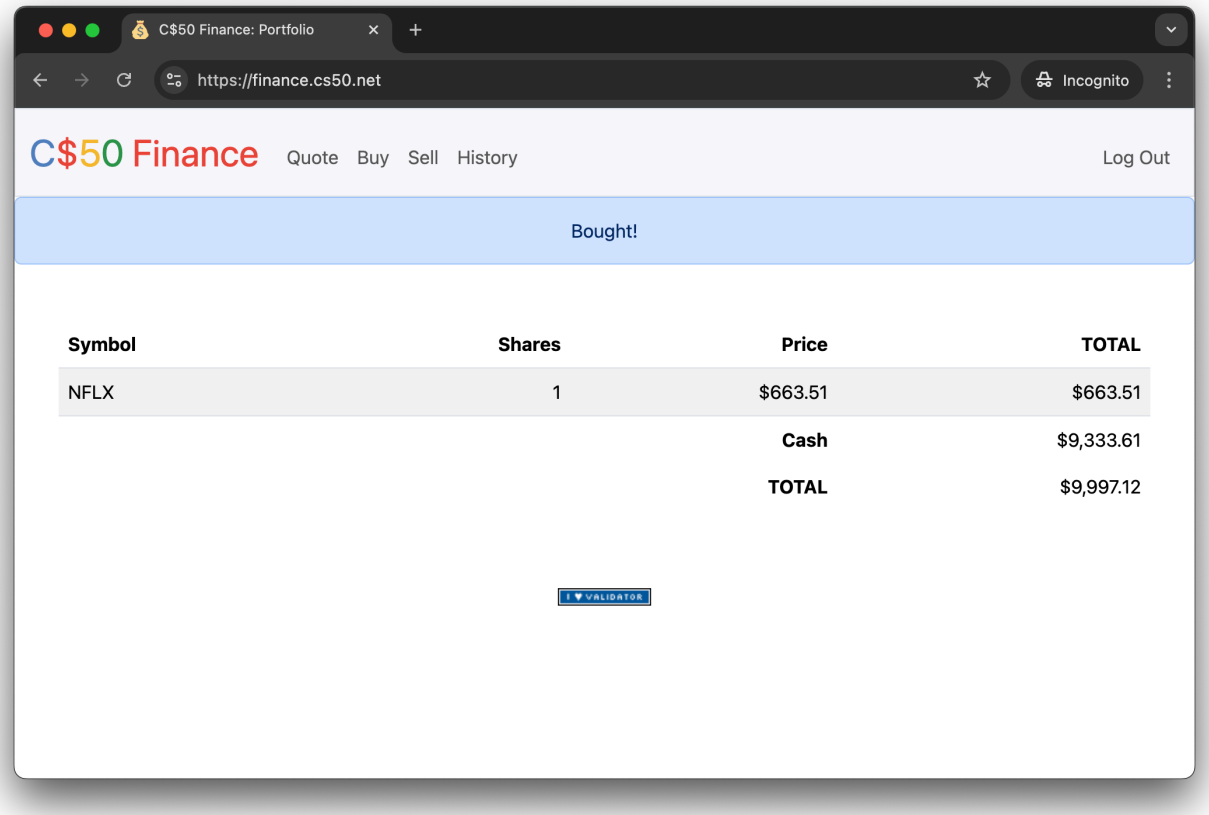
OpenCourseWare

Doar (https://cs50.harvard.edu/donate)

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan)
(https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/)
(https://www.reddit.com/user/davidjmalan)
(https://www.threads.net/@davidjmalan) (https://twitter.com/davidjmalan)

C$50 Finanças

Implemente um site através do qual os usuários possam "comprar" e "vender" ações, conforme o exemplo abaixo.



Fundo

Se você não tem muita certeza do que significa comprar e vender ações (ou seja, participações em uma empresa), acesse este link (https://www.investopedia.com/articles/basics/06/invest1000.asp) para um tutorial.

Você está prestes a implementar o C$50 Finance, um aplicativo web que permite gerenciar carteiras de ações. Essa ferramenta não só permite verificar os preços reais das ações e os valores das carteiras, como também possibilita comprar (bem, "comprar") e vender (bem, "vender") ações consultando seus preços.

De fato, existem ferramentas (uma delas é conhecida como IEX) que permitem baixar cotações de ações por meio de sua API (interface de programação de aplicativos) usando URLs como esta `https://api.iex.cloud/v1/data/core/quote/nflx?token=API_KEY`. Observe como o símbolo da Netflix (NFLX) está incorporado nesta URL; é assim que a IEX sabe de quem são os dados a serem retornados. Esse link não retornará nenhum dado, pois a IEX exige o uso de uma chave de API, mas, se retornasse, você veria uma resposta em formato JSON (JavaScript Object Notation) como esta:


```
{
  "avgTotalVolume":6787785,
  "calculationPrice":"tops",
  "change":1.46,
  "changePercent":0.00336,
  "close":null,
  "closeSource":"official",
  "closeTime":null,
  "companyName":"Netflix Inc.",
  "currency":"USD",
  "delayedPrice":null,
  "delayedPriceTime":null,
  "extendedChange":null,
  "extendedChangePercent":null,
  "extendedPrice":null,
  "extendedPriceTime":null,
  "high":null,
  "highSource":"IEX real time price",
  "highTime":1699626600947,
  "iexAskPrice":460.87,
  "iexAskSize":123,
  "iexBidPrice":435,
  "iexBidSize":100,
  "iexClose":436.61,
  "iexCloseTime":1699626704609,
  "iexLastUpdated":1699626704609,
  "iexMarketPercent":0.00864679844447232,
  "iexOpen":437.37,
  "iexOpenTime":1699626600859,
  "iexRealtimePrice":436.61,
  "iexRealtimeSize":5,
  "iexVolume":965,
  "lastTradeTime":1699626704609,
```

```
  "latestPrice":436.61,
  "latestSource":"IEX real time price",
  "latestTime":"9:31:44 AM",
  "latestUpdate":1699626704609,
  "latestVolume":null,
  "low":null,
  "lowSource":"IEX real time price",
  "lowTime":1699626634509,
  "marketCap":192892118443,
  "oddLotDelayedPrice":null,
  "oddLotDelayedPriceTime":null,
  "open":null,
  "openTime":null,
  "openSource":"official",
  "peRatio":43.57,
  "previousClose":435.15,
  "previousVolume":2735507,
  "primaryExchange":"NASDAQ",
  "symbol":"NFLX",
  "volume":null,
  "week52High":485,
  "week52Low":271.56,
  "ytdChange":0.4790450244167119,
  "isUSMarketOpen":true
}
```


Observe como, entre as chaves, há uma lista de pares chave-valor separados por vírgulas, com dois pontos separando cada chave do seu valor. Faremos algo muito semelhante com nossa própria API de banco de dados de ações.

Vamos agora nos concentrar em obter o código de distribuição para este problema!

Começando

Faça login em cs50.dev (https://cs50.dev/) , clique na janela do terminal e execute `cd` o comando. Você verá que o prompt do terminal será semelhante ao seguinte:

```
$
```

Em seguida, execute

```
wget https://cdn.cs50.net/2024/fall/psets/9/finance.zip
```

para baixar um arquivo ZIP chamado `finance.zip` para o seu espaço de código.

Em seguida, execute

```
unzip finance.zip
```

para criar uma pasta chamada `finance`. Você não precisa mais do arquivo ZIP, então pode executar

```
rm finance.zip
```

e responda com "y" seguido de Enter quando solicitado para remover o arquivo ZIP que você baixou.

Agora digite

```
cd finance
```

Em seguida, pressione Enter para entrar (ou seja, abrir) esse diretório. Seu prompt agora deve ser semelhante ao abaixo.

```
finance/ $
```

Execute o programa `ls` isoladamente e você deverá ver alguns arquivos e pastas:

```
app.py  finance.db  helpers.py  requirements.txt  static/  templates/
```

Se você encontrar algum problema, siga esses mesmos passos novamente e veja se consegue determinar onde errou!

Correndo

Inicie o servidor web integrado do Flask (dentro de `finance/`):

```
$ flask run
```

Acesse o URL exibido `flask` para ver o código de distribuição em ação. No entanto, você ainda não poderá fazer login ou se cadastrar!

Dentro do arquivo `finance/`, execute o comando `sqlite3 finance.db` para abrir `finance.db` com `sqlite3`. Se você executar `.schema` no prompt do SQLite, observe que `finance.db` vem com uma tabela chamada `users`. Observe sua estrutura (ou seja, o esquema). Note que, por padrão, novos usuários receberão US$ 10.000 em dinheiro. Mas se você executar `SELECT * FROM users;`, não há (ainda!) nenhum usuário (ou seja, nenhuma linha) para navegar.

Outra forma de visualizar `finance.db` é com um programa chamado phpLiteAdmin. Clique em `finance.db` no navegador de arquivos do seu espaço de código e, em seguida, clique no link exibido abaixo do texto "Please visit the following link to authorize GitHub Preview". Você deverá ver informações sobre o próprio banco de dados, bem como uma tabela, , `users` assim como você viu no `sqlite3` prompt com `.schema`.

Entendimento

`app.py`

Abra `app.py` o arquivo. No topo, você encontrará várias importações, incluindo o módulo SQL do CS50 e algumas funções auxiliares. Falaremos mais sobre elas em breve.

Após configurar o Flask (https://flask.palletsprojects.com/en/stable/) , observe como este arquivo desativa o cache de respostas (desde que você esteja no modo de depuração, que é o padrão no seu espaço de código Code50), para evitar que você faça uma alteração em algum arquivo sem que seu navegador perceba. Observe também como ele configura o Jinja (https://jinja.palletsprojects.com/en/3.1.x/) com um "filtro" personalizado `usd` , uma função (definida em ` `helpers.py` json.json`) que facilita a formatação de valores em dólares americanos (USD). Além disso, ele configura o Flask para armazenar sessões (https://flask.palletsprojects.com/en/1.1.x/quickstart/#sessions) no sistema de arquivos local (ou seja, no disco) em vez de armazená-las em cookies (assinados digitalmente), que é o padrão do Flask. O arquivo então configura o módulo SQL do CS50 para usar o SQL Server `finance.db` .

Em seguida, há uma série de rotas, das quais apenas duas estão totalmente implementadas: `/users` `login` e `/users/users` `logout` . Leia `login` primeiro a implementação de `/users/users`. Observe como ela usa `db.execute` `get_user_id` (da biblioteca CS50) para consultar ` /users/users` `finance.db` . E observe como ela usa ` `check_password_hash` get_user_id` para comparar os hashes das senhas dos usuários. Observe também como `/ `login` users/users` "lembra" que um usuário está logado, armazenando seu `id` `user_id` , um INTEGER, em `user_id` `session` . Dessa forma, qualquer uma das rotas deste arquivo pode verificar qual usuário, se houver, está logado. Finalmente, observe como, uma vez que o usuário tenha feito login com sucesso, `/users/ users` `login` redirecionará para `/ `"/"` users/users`, levando o usuário para sua página inicial. Enquanto isso, observe como `/users/users` `logout` simplesmente limpa `session` `user_id`, efetivamente desconectando o usuário.

Observe como a maioria das rotas são "decoradas" com `@login_required` (uma função definida `helpers.py` também em). Esse decorador garante que, se um usuário tentar acessar qualquer uma dessas rotas, ele será redirecionado primeiro para `login` para fazer login.

Observe também como a maioria das rotas suporta GET e POST. Mesmo assim, a maioria delas (por enquanto!) simplesmente retorna um pedido de desculpas, já que ainda não foram implementadas.

`helpers.py`

Em seguida, observe ` `helpers.py` . Ah, aqui está a implementação de `apology` `. Note como ela acaba renderizando um modelo, `apology.html` `. Ela também define internamente outra função, `escape` `, que usa simplesmente para substituir caracteres especiais em pedidos de desculpas.

Ao definir ` escape dentro de ` apology ,` restringimos o escopo da primeira apenas à segunda; nenhuma outra função poderá (ou precisará) chamá-la.

A próxima linha no arquivo é login_required . Não se preocupe se esta for um pouco enigmática, mas se você já se perguntou como uma função pode retornar outra função, aqui está um exemplo!

Em seguida, temos `shared_stock_quote` lookup , uma função que, dado um `shared_quote` symbol (por exemplo, NFLX), retorna uma cotação de ações para uma empresa na forma de um `shared_quote` dict com três chaves: ` name shared_quote` , cujo valor é um `shared_quote`; `shared_quote`, cujo valor é um `shared_quote` ; e `shared_quote`, cujo valor é um ` shared_quote`, uma versão canônica (em maiúsculas) do símbolo da ação, independentemente de como esse símbolo foi capitalizado ao ser passado para ` shared_stock_quote`. Observe que esses não são preços "em tempo real", mas mudam ao longo do tempo, assim como no mundo real! str price float symbol str lookup

Por último no arquivo está usd , uma função curta que simplesmente formata um float como USD (por exemplo, 1234.56 é formatado como $1,234.56).

requirements.txt

Em seguida, dê uma olhada rápida em requirements.txt . Esse arquivo simplesmente define os pacotes dos quais este aplicativo dependerá.

static/

Dê uma olhada também em static/ , dentro do qual está styles.css . É aí que reside o CSS inicial. Sinta-se à vontade para alterá-lo como achar melhor.

templates/

Agora observe o arquivo `in` templates/ . Ele login.html é, essencialmente, um formulário HTML estilizado com Bootstrap (https://getbootstrap.com/) . apology.html Já o arquivo `template` é um modelo para um pedido de desculpas. Lembre-se de que apology o arquivo `in` helpers.py recebia dois argumentos: message `name`, que era passado para `template` render_template como o valor de bottom `name`, e, opcionalmente, `name` code , que era passado para ` render_template template` como o valor de `name` top . Observe apology.html como esses valores são usados no final! E aqui está o porquê (https://github.com/jacebrowning/memegen) :-)

Por último, temos o layout.html . Ele é um pouco maior que o normal, principalmente porque vem com uma "barra de navegação" elegante e otimizada para dispositivos móveis, também baseada no Bootstrap. Observe como ele define um bloco, main , dentro do qual os templates (incluindo apology.html e login.html) serão inseridos. Ele também inclui suporte para o recurso de exibição de mensagens

(https://flask.palletsprojects.com/en/1.1.x/quickstart/#message-flashing) do Flask , permitindo que você transmita mensagens de uma rota para outra para o usuário visualizar.

Especificação

`register`

Conclua a implementação `register` de forma que permita ao usuário criar uma conta por meio de um formulário.

- Exija que o usuário insira um nome de usuário, implementado como um campo de texto cujo valor `name` é `username`. Exiba uma mensagem de desculpas se o campo inserido pelo usuário estiver em branco ou se o nome de usuário já existir.
  - Observe que `cs50.SQL.execute` (https://cs50.readthedocs.io/libraries/cs50/python/#cs50.SQL)será lançada uma `ValueError` exceção se você tentar usar `INSERT` um nome de usuário duplicado, pois criamos um `UNIQUE INDEX` em `users.username`. Portanto, certifique-se de usar `try` e `except` para determinar se o nome de usuário já existe.
- Exija que o usuário insira uma senha, implementada como um campo de texto cujo valor `name` seja `password`, e em seguida, essa mesma senha novamente, implementada como um campo de texto cujo valor `name` seja `confirmation`. Exiba uma mensagem de desculpas se qualquer uma das entradas estiver em branco ou se as senhas não coincidirem.
- Envie a entrada do usuário através `POST` de `/register`.
- `INSERT` O novo usuário é inserido em `users`, armazenando um hash da senha do usuário, não a senha em si. `generate_password_hash` (https://werkzeug.palletsprojects.com/en/2.3.x/utils/#werkzeug.security.generate_password_hash)É provável que você queira criar um novo modelo (por exemplo, `register.html`) que seja bastante semelhante a `login.html`.

Depois de implementar `register` corretamente, você poderá se cadastrar e fazer login (já que `login` ambos `logout` já funcionam)! E você poderá visualizar suas linhas pelo phpLiteAdmin ou pelo `sqlite3`.

`quote`

Complete a implementação `quote` de forma que permita ao usuário consultar o preço atual de uma ação.

- Exija que o usuário insira o símbolo de uma ação, implementado como um campo de texto cujo valor `name` é `symbol`.
- Envie a entrada do usuário através `POST` de `/quote`.

- Provavelmente você vai querer criar dois novos modelos (por exemplo, `<template>` `quote.html` e ` `quoted.html` <template>`). Quando um usuário acessar o site `/quote` via GET, renderize um desses modelos, dentro do qual deve haver um formulário HTML que será enviado para o `/quote` site via POST. Em resposta a um POST, `quote` o site pode renderizar o segundo modelo, incorporando nele um ou mais valores de `<template>` `lookup`.

buy

Conclua a implementação `buy` de forma que permita ao usuário comprar ações.

- Exija que o usuário insira o símbolo de uma ação, implementado como um campo de texto cujo valor `name` é `symbol`. Exiba uma mensagem de desculpas se a entrada estiver em branco ou se o símbolo não existir (conforme o valor de retorno de `lookup`).
- Exija que o usuário insira um número de ações, implementado como um campo de texto cujo valor `name` seja `shares`. Exiba uma mensagem de desculpas se a entrada não for um número inteiro positivo.
- Envie a entrada do usuário através `POST` de `/buy`.
- Ao concluir, redirecione o usuário para a página inicial.
- É bem provável que você queira ligar `lookup` para consultar o preço atual de uma ação.
- Provavelmente você vai querer saber `SELECT` quanto dinheiro o usuário tem atualmente `users`.
- Adicione uma ou mais tabelas novas para `finance.db` acompanhar as compras. Armazene informações suficientes para saber quem comprou o quê, a que preço e quando.
  - Utilize os tipos SQLite apropriados.
  - Defina `UNIQUE` índices em todos os campos que devem ser únicos.
  - Defina índices (não- `UNIQUE`) em quaisquer campos pelos quais você irá pesquisar (como via `SELECT` com `WHERE`).
- Apresente um pedido de desculpas, sem concluir a compra, caso o usuário não possa arcar com a quantidade de ações ao preço atual.
- Você não precisa se preocupar com condições de corrida (ou usar transações).

Depois de implementado `buy` corretamente, você poderá visualizar as compras dos usuários em suas novas tabelas através do phpLiteAdmin ou `sqlite3`.

index

Complete a implementação `index` de forma que exiba uma tabela HTML resumindo, para o usuário atualmente logado, quais ações ele possui, a quantidade de ações, o preço atual de cada ação e o valor total de cada posição (ou seja, ações vezes o preço). Exiba também o saldo em dinheiro atual do usuário, juntamente com o total geral (ou seja, o valor total das ações mais o saldo em dinheiro).

- É provável que você queira executar várias `SELECT` operações. Dependendo de como você implementar suas tabelas, poderá achar as operações GROUP BY (https://www.google.com/search?q=SQLite+GROUP+BY) , HAVING (https://www.google.com/search?q=SQLite+HAVING) , SUM (https://www.google.com/search?q=SQLite+SUM) e/ou WHERE (https://www.google.com/search?q=SQLite+WHERE) interessantes.
- Provavelmente você vai querer ligar `lookup` para cada ação.

`sell`

Complete a implementação `sell` de forma que permita ao usuário vender ações de uma empresa (da qual ele é proprietário).

- Exija que o usuário insira o símbolo de uma ação, implementado como um menu `select`. Exiba uma mensagem de desculpas caso o usuário não selecione uma ação ou se (de alguma forma, após o envio) o usuário não possuir ações daquela ação. `name` `symbol`
- Exija que o usuário insira um número de ações, implementado como um campo de texto cujo valor `name` seja `shares`. Exiba uma mensagem de desculpas se a entrada não for um número inteiro positivo ou se o usuário não possuir essa quantidade de ações.
- Envie a entrada do usuário através `POST` de `/sell`.
- Ao concluir, redirecione o usuário para a página inicial.
- Você não precisa se preocupar com condições de corrida (ou usar transações).

`history`

Complete a implementação `history` de forma que exiba uma tabela HTML resumindo todas as transações de um usuário, listando linha por linha cada compra e cada venda.

- Para cada linha, indique claramente se uma ação foi comprada ou vendida e inclua o símbolo da ação, o preço (de compra ou venda), o número de ações compradas ou vendidas e a data e hora em que a transação ocorreu.
- Você pode precisar alterar a tabela que criou `buy` ou complementá-la com uma tabela adicional. Tente minimizar as redundâncias.

toque pessoal

Inclua pelo menos um toque pessoal de sua escolha:

- Permitir que os usuários alterem suas senhas.
- Permitir que os usuários adicionem dinheiro extra à sua conta.
- Permitir que os usuários comprem mais ações ou vendam ações que já possuem por meio da `index` própria plataforma, sem precisar digitar manualmente os códigos das ações.

- Implemente alguma outra funcionalidade de escopo comparável.

Passo a passo

Testando

Certifique-se de testar seu aplicativo web manualmente, como por exemplo:

- Registrar um novo usuário e verificar se a página do portfólio dele carrega com as informações corretas.
- Solicitar uma cotação usando um símbolo de ação válido,
- comprar uma mesma ação várias vezes e verificar se os totais da carteira estão corretos.
- vender todas ou parte de uma ação, verificando novamente o portfólio, e
- Verificando se a sua página de histórico exibe todas as transações do usuário conectado.

Teste também alguns usos inesperados, como por exemplo:

- Inserir sequências alfabéticas em formulários quando apenas números são esperados.
- Inserir números zero ou negativos em formulários quando apenas números positivos são esperados.
- Inserir valores de ponto flutuante em formulários quando apenas números inteiros são esperados.
- Tentar gastar mais dinheiro do que o usuário possui,
- tentar vender mais ações do que um usuário possui,
- inserir um símbolo de ação inválido, e

- incluindo caracteres potencialmente perigosos como `'` e `;` em consultas SQL.

Você também pode verificar a validade do seu HTML clicando no botão **I ❤VALIDATOR** no rodapé de cada uma de suas páginas, o que enviará seu HTML para validator.w3.org (https://validator.w3.org/) .

Após estar satisfeito, para testar seu código `check50`, execute o comando abaixo.

```
check50 cs50/problems/2025/x/finance
```

> Tenha em mente que isso `check50` testará seu programa inteiro como um todo. Se você executá-lo **antes de** concluir todas as funções necessárias, ele poderá relatar erros em funções que, na verdade, estão corretas, mas dependem de outras funções.

Estilo

```
style50 app.py
```

Solução da equipe

Fique à vontade para estilizar seu aplicativo de forma diferente, mas aqui está a solução da equipe!

https://finance.cs50.net/ (https://finance.cs50.net/)

Fique à vontade para criar uma conta e explorar o site. **Não** use a mesma senha que você usa em outros sites.

É **razoável** analisar o HTML e o CSS da equipe.

Dicas

- Para formatar um valor como um valor em dólares americanos (com os centavos listados com duas casas decimais), você pode usar o `usd` filtro em seus modelos Jinja (imprimindo valores como `{{ value | usd }}` em vez de `{{ value }}`.
- Dentro de ` `cs50.SQL` is` existe um `execute` método cujo primeiro argumento deve ser uma consulta `str` SQL. Se essa `str` consulta contiver parâmetros com ponto de interrogação aos quais valores devem ser associados, esses valores podem ser fornecidos como parâmetros nomeados adicionais para ` is` `execute`. Veja a implementação de ` `login` is` para um exemplo. O valor de retorno de ` `execute` is` é o seguinte:

    - Se `str` `is` for um `table` `SELECT`, então ` `execute` is` retorna um conjunto `list` de zero ou mais `dict` objetos, dentro dos quais estão chaves e valores que representam os campos e células de uma tabela, respectivamente.
    - Se `str` `is` for um `int` `INSERT`, e a tabela na qual os dados foram inseridos contiver um `incrementable` `PRIMARY KEY`, então `execute` `is` retornará o valor da chave primária da linha recém-inserida.
    - Se `str` for um `DELETE` ou um `UPDATE`, então `execute` retorna o número de linhas excluídas ou atualizadas por `str`.
- Lembre-se de que isso `cs50.SQL` registrará na janela do terminal todas as consultas que você executar `execute` (para que você possa confirmar se elas estão corretas).
- Certifique-se de usar parâmetros delimitados por ponto de interrogação (ou seja, um estilo de parâmetro de `{ {paramstyle} (https://www.python.org/dev/peps/pep-0249/#paramstyle) } `named`) ao chamar `execute` o método do CS50, à la ` {{paramstyle}}` `WHERE ?`. **Não** use f-strings `format` (https://docs.python.org/3/library/functions.html#format)ou `+` concatenação (ou seja, concatenação), para evitar o risco de um ataque de injeção de SQL.
- Se (e somente se) você já estiver familiarizado com SQL, fique à vontade para usar o SQLAlchemy Core (https://docs.sqlalchemy.org/en/latest/index.html) ou o Flask-SQLAlchemy (https://flask-sqlalchemy.readthedocs.io/en/stable/) (ou seja, o SQLAlchemy ORM (https://docs.sqlalchemy.org/en/latest/index.html)) em vez de `cs50.SQL`.
- Lembre-se de que, embora seja possível validar os valores inseridos em um formulário HTML, usuários experientes podem contornar essa validação. Certifique-se de validar os valores também no servidor. Da mesma forma, se você optou por usar JavaScript em sua solução, verifique se o aplicativo continua funcionando mesmo que o usuário tenha desativado o JavaScript.
- Fique à vontade para adicionar arquivos estáticos adicionais ao `static/`.
- Ao adicionar funções extras `helpers.py`, certifique-se de não abrir uma nova conexão com o banco de dados. Se uma função auxiliar exigir o uso do banco de dados, passe-o como parâmetro em vez de recriar a conexão.
- É bem provável que você queira consultar a documentação do Jinja (https://jinja.palletsprojects.com/en/3.1.x/) ao implementar seus modelos.
- É **razoável** pedir a outras pessoas que experimentem (e tentem reproduzir erros em) seu site.
- Fique à vontade para alterar a estética dos sites, por exemplo, através de
    - bootswatch.com (https://bootswatch.com/) ,
    - getbootstrap.com/docs/5.1/content (https://getbootstrap.com/docs/5.1/content/) ,
    - getbootstrap.com/docs/5.1/components (https://getbootstrap.com/docs/5.1/components/) e/ou
    - memegen.link (https://memegen.link/) .
- A documentação do Flask (https://flask.palletsprojects.com/en/1.1.x/quickstart/) e a documentação do Jinja (https://jinja.palletsprojects.com/en/2.11.x/templates/) podem ser

úteis para você!

Perguntas frequentes

ImportError: Nenhum módulo chamado 'application'

Por padrão, `flask` o programa procura um arquivo chamado `file` `app.py` no seu diretório de trabalho atual (porque configuramos o valor de `PATH` `FLASK_APP`, uma variável de ambiente, para `PATH` `app.py`). Se você estiver vendo este erro, provavelmente executou o programa `flask` no diretório errado!

Erro do sistema operacional: [Errno 98] Endereço já em uso

Se, ao executar o comando `flask`, você vir este erro, é provável que ainda haja `flask` outro processo em execução em outra aba. Certifique-se de encerrar esse outro processo, por exemplo, com Ctrl+C, antes de iniciar `flask` novamente. Se não houver nenhuma outra aba aberta, execute o comando `fuser -k 8080/tcp` para encerrar quaisquer processos que ainda estejam utilizando a porta TCP 8080.

Como enviar

No seu terminal, execute o comando abaixo para submeter seu trabalho.

```
submit50 cs50/problems/2025/x/finance
```

Por que meu envio passa no check50, mas mostra "Nenhum resultado" no meu Livro de Notas após executar o submit50?

Em alguns casos, `submit50` o sistema pode não avaliar a tarefa devido a (1) formatação inconsistente no seu `app.py` arquivo e/ou (2) envio de arquivos adicionais e desnecessários junto com o conjunto de problemas. Para corrigir esses problemas, execute o comando `style50 app.py` na `finance` pasta. Resolva quaisquer problemas encontrados. Em seguida, examine o conteúdo da sua `finance` pasta. Exclua arquivos desnecessários, como sessões do Flask ou outros arquivos que não fazem parte da sua implementação do conjunto de problemas. Além disso, execute o comando `check50` novamente para garantir que sua submissão ainda funcione. Finalmente, execute o `submit50` comando acima mais uma vez. Seu resultado aparecerá no seu **Livro de Notas (https://cs50.me/cs50x)** em alguns minutos.

Observe que, se houver uma pontuação numérica ao lado da sua atividade de finanças na `submissions` área do seu **Boletim de Notas (https://cs50.me/cs50x)** , o procedimento

descrito acima não se aplica a você. Provavelmente, você não atendeu completamente aos requisitos da lista de problemas e deve usar essa `check50` pontuação como referência para saber o que ainda precisa ser feito.