bot.py

Este módulo define e executa a função principal do bot do Telegram, utilizando a biblioteca `python-telegram-bot`. O código realiza as seguintes operações:

1. Importações: São importadas classes essenciais do módulo `telegram.ext`, incluindo `ApplicationBuilder`, `CommandHandler`, `MessageHandler`, `CallbackQueryHandler`, `ConversationHandler` e `filters`, utilizadas para estruturar a lógica do bot. Além disso, variáveis de configuração (`TOKEN_TELEGRAM`, `STATUS`, `PEDIR_CPF`, `CONCLUIDO`) e funções auxiliares (`start`, `ajuda`, `button_handler`, `debug_add_saldo`, `drop`, `processar_cpf`, `cancel`) são importadas de módulos internos do projeto.

2. Inicialização da aplicação: A função `run_bot` cria a instância da aplicação com `ApplicationBuilder().token(TOKEN_TELEGRAM).build()`, utilizando o token definido no arquivo de configuração.

3. Definição do ConversationHandler: Um `ConversationHandler` é configurado para controlar um fluxo de conversação interativo com múltiplos estados. O ponto de entrada é o comando `/start`, que aciona a função `start`. O estado `PEDIR_CPF` responde a mensagens de texto (excluindo comandos) e chama a função `processar_cpf`. O estado `CONCLUIDO` responde a qualquer mensagem utilizando a função `cancel`. O comando `/cancel` é definido como fallback.

4. Adição de handlers:

    - O comando `/ajuda` é registrado com `ajuda`.

    - O `CallbackQueryHandler` é adicionado para lidar com botões inline via `button_handler`.

    - Se o modo de operação (`STATUS`) estiver configurado como `"DEV"`, são adicionados comandos administrativos como `/debugsaldo` e `/drop` para fins de desenvolvimento.

5. Execução do bot: Após a configuração dos handlers, o bot é iniciado com `app.run_polling()`, que mantém o bot escutando novas atualizações de forma contínua (polling).

AtivarSms.py

Descrição Técnica da Função `ativar_sms`

Este módulo define a função `ativar_sms`, responsável por acionar a API de ativação de SMS por meio de um `orderid` previamente gerado.

1. Importações:
    - São importados componentes da biblioteca `telegram` e `telegram.ext`, embora não utilizados diretamente neste trecho.
    - A função também importa a conexão e o cursor do banco de dados (`con`, `cursor`), além das variáveis de configuração `URL_SMS_ACTIVATE_API` e `API_SMS_ACTIVATE_KEY`.
    - A biblioteca `requests` é utilizada para realizar a requisição HTTP à API externa.
2. Definição da função `ativar_sms(orderid)`:
    - A função é assíncrona (`async`) e recebe como parâmetro o `orderid`, identificador único da solicitação de ativação de número virtual.
    - A URL da API é montada dinamicamente com base no `orderid`.
    - O cabeçalho da requisição HTTP inclui a chave da API (`apikey`) necessária para autenticação com a plataforma SMS.
3. Execução da requisição HTTP:
    - A função realiza uma requisição `GET` à API, com timeout de 10 segundos.

- Se a resposta for bem-sucedida (`status_code` 200), o conteúdo JSON é verificado.

- Caso contenha a chave `error`, uma exceção `ValueError` é lançada com a mensagem correspondente.

4. Retorno da função:

   - Em caso de sucesso, a função retorna um `tuple` com `True` e o código de ativação recebido (`resultado["code"]`).

   - Em caso de falha (exceção de rede, timeout, erro da API), retorna `False` e a descrição da exceção capturada.

Comprar_sms.py

Esta função assíncrona realiza a compra e ativação de um número virtual por meio da API da plataforma SMS-PVA. Ela centraliza o fluxo de requisição, resposta e validação do número adquirido.

1. Importações:

   - São utilizados objetos do Telegram (`Update`, `InlineKeyboardButton`, `InlineKeyboardMarkup`) para responder ao usuário.

   - `requests` é utilizado para comunicação HTTP com a API da SMS-PVA.

   - A função `ativar_sms` é importada para ativar o número após a compra.

   - Também são importadas variáveis de configuração como `API_SMS_ACTIVATE_KEY` e `URL_SMS_ACTIVATE_API`.

   - A função de logging `logger` é usada para registrar erros.

2. Definição da função `comprar_sms_sms_pva`:

   - Parâmetros: `service_id` (identificador do serviço desejado, como WhatsApp, Telegram etc.), `quantidade` (não utilizado no trecho atual),

`pais` (código ISO do país) e `update` (objeto de atualização do Telegram contendo o contexto da interação).

- Define o dicionário `params` com os parâmetros obrigatórios para a API: o ID do serviço e o código do país.
- Define o cabeçalho HTTP contendo a chave de autenticação.

3. Requisição de número à API:
   - A URL da requisição é construída com os dados do país e serviço.
   - Realiza a chamada `GET` à API da SMS-PVA.
   - Se a resposta contiver uma chave `error`, é lançada uma exceção com a descrição do erro.
4. Processamento da resposta:
   - Se bem-sucedido, o JSON da resposta é processado para extrair o número (`phoneNumber`) e o ID da ordem (`orderId`).
   - Uma mensagem de confirmação é enviada ao usuário via `update.message.reply_text`, indicando que o número está sendo validado.
5. Ativação do número:
   - A função `ativar_sms` é chamada passando o `orderid`. Ela retorna uma tupla indicando o sucesso e o código de ativação.
   - Caso a ativação seja bem-sucedida, é retornado um dicionário com `status=True`, o número e o código.
   - Se houver falha na ativação ou na requisição, é retornado um dicionário vazio.
6. Tratamento de erros:

- Exceções de rede ou falhas HTTP são capturadas e registradas com `logger.error`, garantindo rastreabilidade dos problemas.

comprar.py

Objetivo

A função assíncrona `comprar_sms_sms_pva` é responsável pela compra e ativação de números virtuais através da API da plataforma SMS-PVA. Ela gerencia o fluxo de requisição à API, a resposta obtida, e a validação do número adquirido, além de fornecer feedback ao usuário por meio de mensagens no Telegram.

Importações

- Telegram: Utiliza `Update`, `InlineKeyboardButton`, e `InlineKeyboardMarkup` para interação com o usuário no Telegram.
- requests: Biblioteca para comunicação HTTP com a API da SMS-PVA.
- Função `ativar_sms`: Importada para ativar o número adquirido após a compra.
- Variáveis de Configuração:
  - `API_SMS_ACTIVATE_KEY`: Chave da API para autenticação.
  - `URL_SMS_ACTIVATE_API`: URL base da API SMS-PVA.
- Logger: Utilizado para registrar erros e falhas no fluxo da operação.

Definição da Função `comprar_sms_sms_pva`

Parâmetros:

- `service_id` (str): Identificador do serviço desejado (ex: WhatsApp, Telegram, etc.).

- `quantidade` (int): Quantidade de números a serem adquiridos (não utilizado diretamente no trecho atual).
- `pais` (str): Código ISO 3166-1 do país no qual o número será adquirido (ex: 'BR' para Brasil).
- `update` (objeto `Update`): Contém as informações da interação do usuário no Telegram, permitindo que o bot responda com mensagens.

Fluxo da Função

1. Definição de Parâmetros:
    - A função começa definindo um dicionário `params`, que contém os parâmetros obrigatórios para a requisição à API da SMS-PVA:
        - `service_id`: O identificador do serviço desejado.
        - `pais`: O código ISO do país para a aquisição do número.
2. Cabeçalhos de Autenticação:
    - O cabeçalho HTTP é configurado com a chave de autenticação (`API_SMS_ACTIVATE_KEY`) para garantir que a requisição à API seja validada.
3. Requisição à API SMS-PVA:
    - A URL da requisição é construída utilizando os parâmetros `service_id` e `pais`.
    - A função então realiza uma chamada GET para a API da SMS-PVA.
    - Caso a resposta da API contenha um erro (chave `error`), uma exceção é lançada com a descrição do erro retornado pela API.

4. Processamento da Resposta:
    - Se a requisição for bem-sucedida, o JSON retornado pela API é processado.
    - O número virtual adquirido (`phoneNumber`) e o ID do pedido (`orderId`) são extraídos da resposta.
    - Uma mensagem de confirmação é enviada ao usuário, informando que o número está sendo validado.
5. Ativação do Número:
    - A função `ativar_sms` é chamada com o `orderId` do pedido. Esta função tenta ativar o número adquirido.
    - Se a ativação for bem-sucedida, a função retorna um dicionário com:
        - `status=True`: Indicando que a ativação foi concluída com sucesso.
        - `phoneNumber`: O número adquirido.
        - `activationCode`: O código de ativação gerado para o número.
    - Caso haja falha na ativação, a função retorna um dicionário vazio.
6. Tratamento de Erros:
    - Se ocorrerem exceções de rede ou falhas HTTP, essas exceções são capturadas.
    - O erro é registrado utilizando a função `logger.error` para garantir que o problema seja rastreável, permitindo uma análise posterior.

Exemplo de Fluxo de Execução

1. O usuário solicita a compra de um número para um serviço específico (por exemplo, WhatsApp).

2. A função realiza uma requisição à API SMS-PVA para obter um número virtual disponível para o país especificado.

3. Se a resposta da API for positiva, a função envia ao usuário uma confirmação de que o número foi adquirido e está sendo validado.

4. A função tenta ativar o número adquirido usando a função `ativar_sms`.

5. Se a ativação for bem-sucedida, o número e o código de ativação são retornados e o usuário é informado com sucesso.

6. Caso ocorra qualquer erro (falha de rede, serviço indisponível, etc.), uma mensagem de erro é registrada e pode ser enviada ao usuário.

cpfProc.py

Descrição Técnica da Função `processar_cpf`

Objetivo

A função `processar_cpf` é responsável por validar o CPF inserido pelo usuário, armazená-lo no banco de dados e enviar uma mensagem de boas-vindas com botões de navegação para o usuário, proporcionando um menu de opções para interações subsequentes. Caso o CPF seja inválido, ela solicita novamente o CPF.

Importações

- Telegram: Utiliza `InlineKeyboardButton`, `InlineKeyboardMarkup`, e `Update` para interação com o usuário no Telegram.

- ContextTypes e CallbackContext: Para manipulação de interações com o Telegram e execução de funções baseadas em contexto.

- Banco de Dados: Utiliza a variável `cursor` para realizar operações de banco de dados, como inserir e atualizar registros.

Definição da Função `processar_cpf`

Parâmetros:

- `update` (objeto `Update`): Contém informações sobre a interação do usuário com o bot (ex: mensagem recebida ou consulta de callback).

- `context` (objeto `CallbackContext`): Contém o contexto da execução, incluindo dados do usuário, callback query, etc.

Fluxo da Função

1. Recepção do CPF:

    - A função começa com a extração do CPF informado pelo usuário via `update.message.text.strip()`. O CPF é limpo de espaços extras, caso haja algum.

2. Validação do CPF:

    - A função verifica se o CPF inserido é válido, considerando as seguintes condições:

        - O CPF deve ter exatamente 11 caracteres.

        - O CPF deve ser composto apenas por dígitos numéricos (verificado com `.isdigit()`).

    - Caso o CPF não seja válido, uma mensagem de erro é enviada ao usuário, informando que o CPF precisa ser válido (apenas números e com 11 caracteres). O fluxo então retorna ao estado de pedido de CPF (`PEDIR_CPF`).

3. Armazenamento do CPF:

- Se o CPF for válido, o código atualiza o banco de dados, registrando o CPF do usuário na tabela `user`, associando-o ao `userid` do Telegram. Essa operação é realizada com a instrução SQL `UPDATE`.

4. Mensagem de Boas-vindas e Menu de Navegação:
    - Após o CPF ser registrado com sucesso, o bot envia uma mensagem de boas-vindas ao usuário, contendo uma explicação das opções disponíveis e os botões interativos para navegação.
    - A mensagem inclui um breve resumo das funções que o bot oferece:
        - Escolher Serviço: Seleção de serviço (ex: WhatsApp, Google, etc.).
        - Escolher País: Escolha do país de origem do número.
        - Fazer Recarga de Saldo: Adicionar créditos à conta.
        - Checar Números: Verificar números adquiridos.
        - Comprar Número: Realizar a compra de um número virtual.
    - Os botões são apresentados como `InlineKeyboardButton` e agrupados no `InlineKeyboardMarkup` para formar um menu interativo.
5. Reações Baseadas no Tipo de Mensagem:
    - Se a interação for uma mensagem normal (`update.message`), o bot responde com a mensagem de boas-vindas e os botões interativos diretamente.
    - Se a interação for um callback query (quando o usuário interage com um botão), o bot edita a mensagem existente com o texto de boas-vindas e os botões, substituindo a mensagem anterior.
6. Retorno do Estado:
    - A função retorna o valor `CONCLUIDO`, indicando que o processo de validação e atualização foi completado com sucesso, e o bot está pronto

para a próxima interação.

database.py

Objetivo

Este código realiza a inicialização de um banco de dados SQLite para armazenar informações de usuários e números virtuais. Ele cria duas tabelas: `user` e `numeros`. A tabela `user` armazena dados de identificação e preferências do usuário, enquanto a tabela `numeros` armazena os números virtuais adquiridos pelos usuários.

Importações

- sqlite3: Biblioteca padrão do Python para interagir com bancos de dados SQLite.
- os: Embora a biblioteca seja importada, não é utilizada no código fornecido. Ela pode ser útil para manipulação de caminhos de arquivos e verificações adicionais.

Fluxo do Código

1. Conexão com o Banco de Dados:
    - A conexão com o banco de dados SQLite é estabelecida com o comando `sqlite3.connect("main.db", check_same_thread=False)`.
    - O parâmetro `check_same_thread=False` permite que a conexão seja usada em múltiplos threads, o que pode ser útil no caso de um bot assíncrono, como o do Telegram.
    - A variável `con` contém a conexão, e `cursor` é o objeto que permite executar as operações SQL.
2. Criação da Tabela `user`:

- A tabela `user` é criada para armazenar informações relacionadas ao usuário, como:
    - userid (chave primária): Identificador único do usuário no Telegram.
    - service: Serviço escolhido pelo usuário (ex: WhatsApp, Telegram, etc.). O valor padrão é `'None'`.
    - pais: País associado ao número virtual. O valor padrão é `'None'`.
    - saldo: Saldo do usuário, inicialmente configurado como 0.0.
    - cpf: CPF do usuário, inicialmente configurado como `'None'`.
- O comando `CREATE TABLE IF NOT EXISTS` assegura que a tabela seja criada apenas se não existir, evitando duplicação de tabelas.

3. Criação da Tabela `numeros`:
    - A tabela `numeros` é criada para armazenar os números virtuais adquiridos pelos usuários. Ela contém:
        - userid: Referência ao `userid` da tabela `user`, indicando qual usuário comprou o número.
        - numero: O número virtual adquirido.
        - code: O código associado ao número, usado provavelmente para validação ou ativação.
    - Assim como a tabela `user`, a criação da tabela `numeros` também é feita com o comando `CREATE TABLE IF NOT EXISTS`.

4. Commit das Operações:
    - O comando `con.commit()` é executado para garantir que todas as mudanças realizadas (como a criação das tabelas) sejam salvas no banco de dados.

gerar_qr.py

A função `gerar_qrcode_pix` é responsável por gerar um código QR para pagamento via Pix, utilizando a API fictícia do PixUp. Ela recebe um identificador de pedido, acessa informações do usuário no banco de dados, e envia uma solicitação para criar o QR Code de pagamento. Caso a solicitação seja bem-sucedida, o código QR gerado é retornado. Caso contrário, um erro é informado ao usuário.

Importações

- telegram: Contém objetos necessários para interagir com o Telegram, como `Update`, `InlineKeyboardButton`, `InlineKeyboardMarkup`, que são usados para responder ao usuário.

- telegram.ext: Inclui classes como `ContextTypes`, `ConversationHandler`, e `CallbackContext`, que são utilizadas para gerenciar conversas no Telegram.

- requests: Usado para enviar solicitações HTTP para a API externa do PixUp.

- time: Importado, mas não utilizado no código fornecido. Poderia ser usado para pausas entre tentativas de conexão, se necessário.

- handlers.start: Importa a função `start`, embora não seja utilizada diretamente na função `gerar_qrcode_pix`.

- datetime: Importado, mas não utilizado diretamente no código. Pode ser útil se for necessário registrar ou verificar a data de criação do pedido.

Fluxo da Função

1. Acesso ao Banco de Dados:
   A função começa acessando o banco de dados para obter o CPF do usuário associado ao `userid` do Telegram. O comando SQL:
   con.execute(f"SELECT cpf FROM user WHERE userid = {update.effective_user.id}")
   recupera o CPF do usuário. O método `fetchone()` é utilizado para obter o primeiro resultado da consulta (caso o CPF esteja armazenado para o `userid`).

2. Montagem do Payload para a API:
   A função monta um payload JSON com os seguintes dados:

   - amount: O valor do pagamento (neste caso, fixo em 1).
   - payer: Um dicionário contendo o nome do usuário (`update.effective_user.name`) e seu CPF.
   - postbackUrl: URL de retorno que será chamada após o pagamento via Pix, para processar o resultado.

3. O payload é então configurado como:
   payload = {
   "amount": 1,
   "payer": {
   "name": update.effective_user.name,
   "document": cpf
   },
   "postbackUrl": "http://localhost:5000/webhook/pixup"
   }

4. Envio da Solicitação para a API:
   A solicitação POST é enviada à URL da API de PixUp:
   url = "https://api.pixupbr.com/v2/pix/qrcode"
   O `Authorization` é feito com um token obtido da variável de configuração `PIX_UP` (presumivelmente no arquivo `.env`).
   O cabeçalho de `Content-Type` é definido como `application/json`, pois estamos enviando dados JSON.

5. Verificação da Resposta da API:
   A resposta da API é analisada em formato JSON:
   data = response.json()
   Se o código de status for `401` (não autorizado), isso significa que houve um erro no processamento do pagamento. Nesse caso, a função edita a mensagem no Telegram para informar o erro e redireciona o usuário para a tela inicial após um tempo:
   if data["statusCode"] == 401:
   await update.callback_query.edit_message_text("⚠️ Ocorreu um erro ao processar o pagamento via Pix. Por favor, tente novamente mais tarde. Redirecionando para /start em 10 segundos")

return "None"

6. Retorno do QR Code:
   Se a solicitação for bem-sucedida, a função retorna o código QR gerado pela API:
   return data["qrcode"]

webhook.py

Objetivo
A função `pixup_webhook` é responsável por processar o webhook da API PixUp quando um pagamento via Pix é confirmado. Ela realiza as seguintes ações:

1. Recebe os dados do pagamento.
2. Atualiza o saldo do usuário no banco de dados.
3. Envia uma mensagem ao usuário via Telegram notificando-o sobre a atualização de saldo.

Além disso, o código contém funções auxiliares que gerenciam o saldo do usuário, recuperam o `chat_id` do Telegram e enviam mensagens para o Telegram.

Funções auxiliares

1. `atualizar_saldo(userid, valor)`
   - Objetivo: Atualiza o saldo do usuário no banco de dados.
   - Fluxo:
     - Recupera o saldo atual do usuário com base no `userid`.
     - Se o usuário existe, adiciona o valor recebido ao saldo existente.
     - Se o usuário não existe, insere um novo registro no banco com o saldo inicial.

- Entrada:
  - `userid`: ID do usuário.
  - `valor`: O valor a ser adicionado ao saldo.
- Processamento:
  - Atualiza o saldo ou cria um novo registro de saldo no banco de dados.
- Saída:
  - Atualiza o banco de dados com o novo saldo.

2. `get_chat_id(userid)`
   - Objetivo: Recupera o `chat_id` associado ao `userid` no banco de dados.
   - Fluxo:
     - Realiza uma consulta no banco de dados para obter o `chat_id` do usuário.
   - Entrada:
     - `userid`: ID do usuário.
   - Processamento:
     - Busca o `chat_id` do usuário no banco de dados.
   - Saída:
     - Retorna o `chat_id` do usuário ou `None` caso o usuário não seja encontrado.
3. `send_telegram_message(chat_id, text)`

- Objetivo: Envia uma mensagem para o Telegram utilizando o bot.
- Fluxo:
  - Utiliza a API do Telegram para enviar uma mensagem ao usuário.
  - A URL da API é construída com o token do bot, e a mensagem é enviada com o `chat_id` e o texto.
- Entrada:
  - `chat_id`: O ID do chat do usuário.
  - `text`: O texto da mensagem a ser enviada.
- Processamento:
  - Realiza uma requisição HTTP POST para a API do Telegram.
- Saída:
  - A mensagem é enviada ao Telegram, e a resposta da API não é processada neste caso.

Função principal

4. `pixup_webhook()`
   - Objetivo: Recebe e processa o webhook de pagamento do PixUp.
   - Fluxo:
     - A função é acionada por uma requisição POST ao endpoint `/webhook/pixup`.

        - A função recupera os dados do pagamento enviados pela API PixUp.
        - Verifica o status da transação (se o pagamento foi aprovado).
        - Se aprovado, chama a função `atualizar_saldo()` para adicionar o valor do pagamento ao saldo do usuário.
        - Recupera o `chat_id` do usuário com a função `get_chat_id()`.
        - Envia uma mensagem ao usuário notificando sobre a atualização do saldo com a função `send_telegram_message()`.
    - Entrada:
        - `request.json`: Dados do pagamento enviados pela API PixUp, como `status`, `amount`, e `custom_id`.
    - Processamento:
        - Processa a transação e, se aprovada, atualiza o saldo do usuário e envia uma notificação via Telegram.
    - Saída:
        - Retorna um status `200` ao PixUp, indicando que o webhook foi processado com sucesso.

Roteamento e execução do servidor

5. `run_webhook()`
    - Objetivo: Executa o servidor Flask que escuta os webhooks.
    - Fluxo:

        - A função inicia o servidor Flask na porta `5000` e escuta por requisições HTTP.
    - Entrada:
        - Nenhuma entrada diretamente.
    - Processamento:
        - Inicia o servidor Flask com as rotas definidas.
    - Saída:
        - O servidor Flask é iniciado e começa a escutar as requisições.

ajuda.py

Objetivo:
A função `ajuda` responde ao usuário com informações de contato, como Discord, Telegram e GitHub, além de fornecer uma opção para voltar ao menu principal. Se a interação for via mensagem, ela envia as informações diretamente, caso contrário, edita a mensagem existente com as informações.

Função: ajuda

- Parâmetros:
    - `update` (Update): O objeto de atualização do Telegram que contém a mensagem ou o callback da interação do usuário.
    - `context` (ContextTypes.DEFAULT_TYPE): O contexto padrão do Telegram, utilizado para acessar informações e métodos adicionais.
- Objetivo:
  Responder ao usuário com informações de contato, incluindo Discord, Telegram e GitHub. Além disso, a função oferece um botão para voltar ao menu inicial

(comando `/start`).

- Fluxo:
    - Criação do teclado de navegação:
      A função cria um teclado inline com um botão que permite ao usuário retornar ao menu inicial, utilizando o texto "👟 Voltar ao menu Start".
    - Definição do texto a ser enviado:
      A função define um texto contendo os dados de contato do desenvolvedor (Discord, Telegram, GitHub) e orientações sobre como entrar em contato.
    - Envio ou edição da mensagem:
        - Se a interação for via mensagem (`update.message`), a função envia a mensagem com as informações de contato e o teclado inline.
        - Se a interação for via botão (`update.callback_query`), a função edita a mensagem com as mesmas informações de contato e o teclado inline.
- Entrada:
    - `update.message`: Caso o comando tenha sido enviado por mensagem direta.
    - `update.callback_query`: Caso o comando tenha sido acionado por um botão de callback (como dentro de uma interação de menu).
- Saída:
    - A função envia uma mensagem ou edita uma mensagem existente com informações de contato e um botão para retornar ao menu principal.
- Observações:
    - A função utiliza o `ParseMode.MARKDOWN` para formatar o texto, permitindo que os links ou texto em negrito sejam apresentados

corretamente.

- O botão "Voltar ao menu Start" envia um callback `exit`, que pode ser tratado em outro lugar no código para retornar o usuário ao estado inicial ou menu principal.

Button_handler.py

A função `button_handler` é responsável por lidar com os diferentes botões de interação que os usuários clicam no bot. Ela lida com diversas ações, como escolher serviços, selecionar países, verificar saldo, realizar compras, entre outros.

Fluxo de Execução

1. Interação com "serv" ou "ser_":
    - Quando o usuário clica em um botão relacionado a um serviço, a função busca os serviços disponíveis da API (`smspva.com`) e os exibe de forma paginada.
    - O usuário pode navegar entre as páginas de serviços.
    - Quando um serviço é selecionado, ele é salvo no banco de dados associado ao usuário.
2. Interação com "pais" ou "pais_":
    - Quando o usuário clica em um botão de seleção de país, a função exibe uma lista de países, permitindo a navegação entre as páginas de países.
    - O país selecionado é salvo no banco de dados do usuário.
3. Verificação de Saldo:
    - Quando o usuário escolhe a opção de verificar saldo, o bot consulta o banco de dados e retorna o saldo do usuário.

- Se o saldo for encontrado, ele é exibido ao usuário.

4. Recarregar Saldo:
    - O usuário pode escolher a opção de "recarregar", que chama a função `comprar` (presumivelmente responsável por adicionar saldo à conta do usuário).
5. Comprar SMS:
    - Se o usuário seleciona a opção de comprar um número virtual, o bot verifica se o usuário tem saldo suficiente e, em caso positivo, chama a função `comprar_sms_sms_pva` para realizar a compra do número.
    - Após a compra, o número adquirido é salvo no banco de dados.
6. Ativar Número:
    - O usuário pode consultar os números comprados anteriormente. O bot busca no banco de dados os números associados ao usuário e os exibe.
7. Central de Dúvidas:
    - Quando o usuário escolhe "duvidas", o bot exibe uma lista de perguntas frequentes (FAQ) sobre o serviço, explicando o processo de compra, serviços disponíveis, pagamento, etc.
8. Exit:
    - O botão de "Voltar ao menu Start" retorna o usuário ao menu principal (`start`).
9. Ajuda:
    - Quando o usuário solicita ajuda, o bot chama a função `ajuda` que fornece informações de contato e suporte.

debug.py

Este código foi desenvolvido para adicionar saldo a um usuário no banco de dados via comandos do bot no Telegram. A seguir, detalho os principais componentes e funcionamento do código.

Objetivo Geral:
Adicionar uma funcionalidade de debug no bot do Telegram que permite adicionar um valor ao saldo de um usuário no banco de dados a partir de um comando de texto. Este comando é enviado para o bot com o formato `/debugsaldo <valor>`, onde `<valor>` é o valor monetário a ser adicionado ao saldo do usuário.

Componentes do Código

1. Importação de Módulos:
    - `telegram` e `telegram.ext`: Usados para interagir com a API do Telegram e lidar com atualizações de mensagens e interações.
    - `utils.database`: Contém a conexão com o banco de dados (`con`) e o cursor (`cursor`), usados para executar operações no banco de dados.
2. Função `debug_add_saldo`:
    - Parâmetros:
        - `update`: Objeto que contém informações sobre a atualização recebida, como os dados do usuário e a mensagem.
        - `context`: Objeto que contém o contexto da execução, como os argumentos passados ao comando.
    - Fluxo de Execução:
        - Verificação dos Argumentos: A função começa verificando se o usuário forneceu um argumento (valor) ao comando. Caso contrário, responde com uma mensagem de erro explicando como usar o comando corretamente.

        - Conversão do Valor: A seguir, tenta-se converter o argumento para um valor numérico (`float`). Se falhar, o bot responde indicando que o valor precisa ser numérico.

        - Verificação de Usuário: A função chama `get_user_by_telegram_id` para verificar se o usuário existe no banco de dados. Se o usuário não for encontrado, a execução é interrompida com uma mensagem de erro.

        - Adição de Saldo: Caso o usuário exista, o saldo é atualizado chamando a função `add_saldo`, que adiciona o valor ao saldo existente no banco de dados.

        - Resposta: Por fim, o bot responde ao usuário confirmando a adição do saldo, formatando a mensagem com o valor adicionado.

3. Função `add_saldo`:

    - Objetivo: Esta função é responsável por realizar a atualização do saldo do usuário no banco de dados.

    - Parâmetros:

        - `telegram_id`: ID do usuário no Telegram.

        - `valor`: Valor a ser adicionado ao saldo.

    - Fluxo de Execução:

        - A função executa uma query SQL `UPDATE` para incrementar o saldo do usuário no banco de dados. A query utiliza os parâmetros fornecidos (`valor` e `telegram_id`).

        - Após a execução da query, a transação é confirmada com `con.commit()`.

        - A conexão é fechada com `con.close()`. Nota: O fechamento da conexão após cada operação é algo que pode ser melhorado, pois em uma aplicação maior, isso pode gerar problemas de

desempenho ou de reuso da conexão.

4. Função `get_user_by_telegram_id`:
    - Objetivo: Verificar a existência de um usuário no banco de dados com base no `telegram_id`.
    - Parâmetros:
        - `telegram_id`: ID do usuário no Telegram.
    - Fluxo de Execução:
        - A função executa uma query SQL `SELECT` para buscar um usuário com o `telegram_id` fornecido.
        - Se o usuário for encontrado, a função retorna os dados do usuário; caso contrário, retorna `None`.
        - A conexão com o banco de dados é fechada após a execução da consulta.

drop.py

Este código implementa um comando no bot do Telegram, responsável por dropar e recriar as tabelas `usuarios` e `numeros` no banco de dados, além de garantir a criação de novas tabelas (`user` e `numeros`) caso não existam.

Objetivo Geral:

O objetivo desse código é proporcionar um comando de administração, que permite o reset completo de tabelas relacionadas a usuários e números, seguido pela recriação dessas tabelas com uma estrutura específica. Esse tipo de comando pode ser útil durante o desenvolvimento ou para redefinir o banco de dados de uma aplicação, mas deve ser utilizado com cautela, pois ele apaga dados existentes.

Componentes do Código

1. Importação de Módulos:

- `telegram` e `telegram.ext`: São usados para interagir com a API do Telegram e processar atualizações recebidas.

- `utils.database`: Contém a conexão (`con`) e o cursor (`cursor`) para interagir com o banco de dados SQLite. Esse módulo facilita a execução das operações SQL.

2. Função `drop`:

    - Parâmetros:

        1. `update`: Objeto contendo informações sobre a atualização recebida (geralmente relacionado à interação do usuário com o bot).

        2. `context`: Contexto da execução, contendo informações adicionais sobre o comando e seus parâmetros.

    - Fluxo de Execução:

        1. Dropar Tabelas:

A função começa executando dois comandos SQL para remover as tabelas `usuarios` e `numeros` do banco de dados. Isso é feito com os comandos:

python
CopiarEditar

```
cursor.execute("DROP TABLE usuarios")

cursor.execute("DROP TABLE numeros")
```

- Esses comandos deletam completamente as tabelas e todos os dados contidos nelas. Caso essas tabelas não existam, o código geraria um erro, mas não está tratado aqui, o que pode ser uma limitação.

2. Criar Tabelas `user` e `numeros`:

- Após dropar as tabelas antigas, o código cria novamente as tabelas `user` e `numeros`. O comando SQL utilizado para criar a tabela `user` define:
    - `userid`: Identificador único para o usuário, definido como chave primária.
    - `service`: Campo que armazena o serviço associado ao usuário, com valor padrão `'None'`.
    - `pais`: Campo que armazena o país do usuário, com valor padrão `'None'`.
    - `saldo`: Armazena o saldo do usuário, com valor inicial de `0.0`.
    - `cpf`: Armazena o CPF do usuário, com valor padrão `'None'`.

3. A tabela `numeros` é criada para armazenar os números virtuais atribuídos aos usuários. Ela possui:
    - `userid`: ID do usuário, associado à chave primária de `user`.
    - `numero`: Número virtual associado ao usuário.
    - `code`: Código de verificação do número.
4. Commit e Persistência:
    - Após executar as operações de `DROP` e `CREATE`, o código chama `con.commit()` para garantir que as alterações sejam persistidas no banco de dados.

start.py

Objetivo

Este código define o comportamento do comando `/start` de um bot do Telegram. O comando serve para inicializar a interação com o usuário, verificar seu cadastro no

banco de dados e fornecer uma mensagem de boas-vindas. Se o usuário não estiver registrado, o bot solicita o CPF para completar o cadastro.

Componentes do Código

1. Importação de Módulos:

    1. `telegram`: Utiliza o módulo `InlineKeyboardButton`, `InlineKeyboardMarkup`, `Update`, e `ParseMode` para criar botões interativos e formatar as mensagens enviadas para o usuário.

    2. `telegram.ext`: Importa `ContextTypes` e `ConversationHandler`, usados para estruturar e gerenciar a conversa com o usuário.

    3. `utils.database`: Conexão com o banco de dados SQLite (`con` e `cursor`) para interagir com a tabela de usuários e manipular dados.

    4. `config`: Importa as configurações (não fornecidas no código), possivelmente usadas para armazenar chaves ou constantes do sistema.

2. Função `start`:
   A função é assíncrona e executa a lógica quando o comando `/start` é invocado no bot.

   Fluxo de Execução:

    1. Criação de Tabelas:

        - O código executa uma instrução SQL para garantir que a tabela `user` exista no banco de dados. Caso não exista, ela é criada com os seguintes campos:

            - `userid`: Identificador único para o usuário, chave primária.

            - `service`: Serviço associado ao usuário, com valor padrão `'None'`.

            - `pais`: País do usuário, com valor padrão `'None'`.

        - `saldo`: Saldo do usuário, com valor padrão `0.0`.
        - `cpf`: CPF do usuário, com valor padrão `'None'`.

2. Verificação de Existência do Usuário:
    - O código realiza uma consulta para verificar se o usuário já existe na tabela `user` usando o `userid` (ID do Telegram do usuário). Caso o usuário não seja encontrado, um novo registro é inserido na tabela.

3. Primeiro Acesso do Usuário (Cadastro):
    - Se o usuário não existir, o bot solicita que o usuário forneça seu CPF, solicitando apenas os números (sem formatação). A mensagem é enviada utilizando o formato Markdown, destacando o pedido do CPF.

4. Usuário Existente (Mensagem de Boas-vindas):
    - Se o usuário já estiver registrado, o bot envia uma mensagem de boas-vindas, explicando as opções disponíveis:
        - Escolher Serviço: Selecionar o serviço desejado (ex: Google, WhatsApp).
        - Escolher País: Definir o país do número virtual.
        - Ver Saldo: Exibir o saldo atual do usuário.
        - Fazer Recarga de Saldo: Adicionar saldo à conta do usuário.
        - Checar Números: Verificar os números virtuais comprados e armazenados no banco de dados.
        - Comprar Número: Adquirir um novo número virtual.
        - Ajuda/FAQ: Obter informações de ajuda e dúvidas.

5. Uso de Botões Inline:
    - Para facilitar a navegação do usuário, a mensagem de boas-vindas inclui botões interativos (InlineKeyboardButtons), que permitem ao usuário interagir com o bot de forma rápida e simples.
    - Cada botão tem um `callback_data` que será usado posteriormente para identificar a ação a ser executada quando o usuário clicar.
6. Envio de Mensagem:
    - O bot envia a mensagem de boas-vindas e os botões de forma diferente dependendo do tipo de interação:
        - Se a interação for uma mensagem (usada quando o usuário inicia o comando `/start`), a mensagem é enviada com os botões.
        - Se a interação for um clique em um botão inline, a mensagem de boas-vindas é editada com os botões atualizados.
7. Retorno:
    - O código retorna `ConversationHandler.END`, o que indica o final da interação na fase de boas-vindas e a transição para a próxima fase do fluxo do bot (dependente do que o usuário selecionar).