1. Exchange Connection:
   * In the func\_connections.py file, the connect\_exchanges() function is responsible for connecting to multiple exchanges using the CCXT library.
   * The function retrieves the API keys and secrets for each exchange from the .env file using the config function from the decouple library.
   * It creates instances of the exchange classes (e.g., ccxt.binance, ccxt.kucoin, etc.) and initializes them with the respective API credentials.
   * The exchange instances are stored in a dictionary called EXCHANGES in the constants.py file, allowing easy access to each exchange throughout the bot.
2. Monitoring Trading Pairs:
   * The constants.py file contains a list called TRADING\_PAIRS that specifies the trading pairs the bot will monitor.
   * These trading pairs include popular cryptocurrencies like BTC, ETH, XRP, etc., traded against USDT.
   * The bot focuses on these specific trading pairs rather than monitoring every possible coin, providing a targeted approach to arbitrage trading.
3. Fetching Market Prices:
   * In the func\_public.py file, the fetch\_market\_prices() function is responsible for retrieving the latest market prices for the specified trading pairs from each connected exchange.
   * It uses the fetch\_ticker() method provided by CCXT to fetch the ticker data for each trading pair on each exchange.
   * The fetched prices are stored in a dictionary where the keys are the exchange names and the values are the corresponding prices for each trading pair.
   * The construct\_market\_prices() function in the same file combines the prices from all exchanges into a single dictionary, which is then converted into a pandas DataFrame for easier manipulation and analysis.
4. Identifying Arbitrage Opportunities:
   * The func\_arbitrage.py file contains the find\_arbitrage\_opportunities() function, which is responsible for identifying profitable arbitrage opportunities.
   * It takes the market prices DataFrame and the minimum profit percentage threshold as input.
   * The function compares the prices of each trading pair across the connected exchanges and calculates the potential profit percentage for each arbitrage opportunity.
   * If the profit percentage exceeds the defined threshold (MIN\_PROFIT\_PERCENTAGE in constants.py), the bot considers it a viable arbitrage opportunity.
   * The identified opportunities are stored in a list, including details such as the trading pair, buy exchange, sell exchange, and potential profit percentage.
5. Executing Trades:
   * The func\_entry\_pairs.py file contains functions for executing trades based on the identified arbitrage opportunities.
   * The open\_arbitrage\_position() function is responsible for placing buy and sell orders on the respective exchanges.
   * It takes the necessary parameters, such as the exchanges, trading pair, order quantities, and prices, and uses CCXT's create\_limit\_buy\_order() and create\_limit\_sell\_order() methods to place the orders.
   * The open\_arbitrage\_positions() function iterates over the identified arbitrage opportunities, determines the optimal order quantities based on the available balances and the maximum allowed exposure, and calls open\_arbitrage\_position() to execute the trades.
6. Managing Positions:
   * The func\_exit\_pairs.py file contains functions for managing opened positions.
   * The check\_order\_status() function is used to retrieve the status of an order (e.g., open, closed, canceled) using CCXT's fetch\_order() method.
   * The close\_arbitrage\_position() function is responsible for closing an opened position when the desired profit threshold is reached or when market conditions change.
   * It cancels any open orders associated with the position using CCXT's cancel\_order() method and updates the position status accordingly.
   * The manage\_arbitrage\_exits() function periodically checks the status of opened positions, determines if they should be closed based on the defined criteria, and calls close\_arbitrage\_position() to execute the position closure.
7. Risk Management:
   * Risk management techniques are implemented throughout the bot to protect against potential losses.
   * The constants.py file defines variables such as ABORT\_ALL\_POSITIONS and CLOSE\_AT\_PROFIT\_THRESHOLD that control the bot's behavior in terms of position management and risk mitigation.
   * In the func\_entry\_pairs.py file, the open\_arbitrage\_positions() function incorporates risk management by limiting the maximum exposure per trade based on the available account balance and the defined MAX\_EXPOSURE variable.
   * The func\_exit\_pairs.py file contains the manage\_arbitrage\_exits() function, which monitors the status of opened positions and determines if they should be closed based on the defined profit threshold or other risk management criteria
   * These risk management functions would be integrated into the trade execution and position management processes to ensure the bot operates within predefined risk boundaries.
   * At some point, a risk\_management.py file will likely be created to consolidate this
8. Main Execution:
   * The main.py file serves as the entry point of the arbitrage bot.
   * It initializes the bot by calling the necessary functions from other files, such as connect\_exchanges() from func\_connections.py to establish connections to the exchanges.
   * The main execution loop runs continuously, fetching market prices, identifying arbitrage opportunities, executing trades, and managing positions.
   * It coordinates the different components of the bot and handles any exceptions or errors that may occur during execution.