**Project Documentation: Shopify Fulfillment Tool v5.1**

**1. Project Overview**

**Shopify Fulfillment Tool** is a desktop application designed to automate and optimize the order fulfillment process for Shopify stores. The program analyzes exported order and stock-level files, determines which orders can be fulfilled, and generates the necessary documents: packing lists for the warehouse and stock write-off files for courier services.

The application features a modern graphical user interface (GUI), making it accessible to users without technical skills.

**2. Key Functionality**

**2.1. Fulfillment Analysis**

This is the core of the application. The user uploads two CSV files: an order report from Shopify and a stock-level report.

* **Simulation:** The program simulates the process of writing off products, prioritizing orders with the highest number of items to maximize the number of fulfilled orders.
* **History:** The system maintains a fulfillment\_history.csv file to mark previously processed orders as "Repeat".
* **Output:** A detailed Excel report (fulfillment\_analysis.xlsx) is generated with four tabs:
  1. fulfillment\_analysis: A complete list of all line items from orders with a status of "Fulfillable" or "Not Fulfillable".
  2. Summary\_Present: A summary report of SKUs that are available for fulfillment.
  3. Summary\_Missing: A summary report of SKUs that are out of stock.
  4. Report Info: Metadata about the report generation time.

**2.2. Packing List Generation**

Based on a successful analysis, the user can create customized packing lists in .xlsx format.

* **Flexible Filtering:** config.json allows for the configuration of an unlimited number of packing list types with various filters (e.g., "single-item orders only" or "orders for DHL courier").
* **Professional Formatting:** The generated files have a professional appearance: items are grouped by order number, and borders, headers, and print settings (landscape orientation, A4 format) are applied.

**2.3. Stock Export Generation**

The application creates files for writing off stock, which are used for integration with courier or warehouse systems.

* **Templates:** The process is based on using existing .xls template files. The program opens a template, inserts SKU and quantity data, and saves it as a new file with the current date in the filename.
* **Configuration:** Like packing lists, export types are flexibly configured in config.json.

**2.4. Graphical User Interface (GUI)**

* **Interactive Tabs:**
  + **Execution Log:** A technical log for developers, showing the detailed progress of operations.
  + **Activity Log:** A user-facing log that records all successful operations (analysis, report creation) with a timestamp and description.
  + **Statistics:** An analytics dashboard that displays key metrics after an analysis (number of completed/uncompleted orders, courier statistics).
  + **Analysis Data:** An interactive table for viewing the analysis results directly within the application.
* **Modal Windows:** All additional windows (report selection, report builder) open on top of the main window, ensuring a logical and user-friendly workflow.
* **Report Builder:** Allows users to create their own custom Excel reports on the fly by selecting desired columns and applying simple filters.

**3. Technical Logic and Architecture**

The project has a three-tier architecture, which is a significant strength.

1. **Presentation Layer (gui\_main.py):** Solely responsible for displaying the interface and handling user interaction. It contains no business logic.
2. **Control Layer (shopify\_tool/core.py):** The central controller module. It acts as a bridge between the interface and the business logic. It is responsible for loading files, calling the appropriate analysis/generation functions, and saving the results.
3. **Business Logic Layer (shopify\_tool/analysis.py, packing\_lists.py, stock\_export.py):** "Pure" modules that perform specific tasks: analysis.py handles computations, packing\_lists.py formats Excel files, and stock\_export.py works with .xls templates.

**Data Flow:** GUI -> Core (loads data) -> Analysis (processes data) -> Core (saves result) -> GUI (displays result).

**4. Advantages and Positive Aspects**

* **Modularity and Scalability:** The clear separation of logic allows for easy addition of new features or modification of existing ones without breaking the entire application. For example, a new report type can be easily added by simply modifying core.py and config.json.
* **High Flexibility:** Using config.json to configure paths, filters, and report types makes the application extremely adaptable to changes in business processes without requiring code changes.
* **Robust GUI:** The use of threading for long-running tasks prevents the interface from freezing. The tab system and modal windows create a professional and intuitive user experience.
* **Foundation for Testing:** The presence of a tests/ directory and basic tests using pytest provides a foundation for building a reliable automated testing system, which is crucial for long-term maintenance.

**5. Assessment of Critical Shortcomings and Recommendations**

Despite its high quality, the project has several aspects that should be improved to enhance its reliability.

1. **Dependency on Outdated Libraries (xlrd, xlutils):**
   * **Problem:** These libraries are no longer actively maintained and only work with the old .xls format. This poses a security risk and limits functionality (e.g., new Excel features cannot be used).
   * **Criticality:** **Medium.** This is currently a necessary measure due to external system requirements, but it constitutes technical debt.
   * **Recommendation:** As soon as the external system is updated to support .xlsx, the stock\_export.py module should be urgently refactored to use the modern openpyxl library.
2. **File Handling Error Management:**
   * **Problem:** The current logic only checks for file existence (os.path.exists). It does not handle cases where a file might be corrupted, have an incorrect CSV format (e.g., a different delimiter), or be missing expected columns. This can lead to the program crashing with an error message that is unclear to the user.
   * **Criticality:** **High.** This directly affects the application's stability.
   * **Recommendation:** In the core.py module, add validation for the loaded DataFrame before passing it to analysis.py. Check for the presence of key columns ('Name', 'Lineitem sku', 'Артикул', etc.) and wrap file-reading operations in more specific try...except blocks (e.g., except pd.errors.ParserError or except KeyError).
3. **Lack of Error Logging:**
   * **Problem:** All exceptions (Exception as e) are simply printed to the console (print()). If the program crashes after being compiled into an .exe, this information will be lost, making it impossible to diagnose the problem.
   * **Criticality:** **High.** Without error logging, maintenance and bug fixing are extremely difficult.
   * **Recommendation:** Integrate the standard logging module. Configure it to write all errors (level ERROR) to a file (e.g., error.log) with the date, time, and full error traceback.