Developer Notes for IDAK Application

This document is intended to serve as a knowledge base for developers working on the IDAK application. It covers various technical details, design decisions, and development practices that contribute to the application's maintenance and enhancement.

Application Overview

The IDAK application is built in Python and is designed to automate the sending of bulk personalized emails. It uses data extracted from an Excel file to dispatch emails with optional attachments.

Codebase Structure

1. **Main Modules**:
    - `base.py`: The entry point of the application which orchestrates the email sending process.
    - `config_secrets.py`: Contains configuration settings such as SMTP details and file paths.
2. **Utility Functions**:
    - `convertRTFtoHTML()`: Converts RTF content to HTML format using LibreOffice.
    - `send_email()`: Handles the construction and sending of emails.
    - `get_excel_filename()`, `getTargetFolder()`: Functions to interact with the file system.
3. **External Dependencies**:
    - Python's `pandas` library for handling Excel files.
    - `smtplib` and `email.mime` modules for email operations.

Development Environment

1. **Python Version**:
    - The application is compatible with Python 3.6 and above.
2. **Required Tools**:
    - Git for version control.
    - A Python virtual environment for dependency management.
3. **Testing Tools**:
    - `unittest` framework for unit testing.
    - `mock` for mocking external dependencies during testing.

Coding Conventions

1. **Style Guide**:
    - Follow PEP 8 style guide for Python code.
    - Use descriptive variable names and maintain modularity in functions.
2. **Comments and Docstrings**:

    - Code should be well-commented, and each function should include a docstring explaining its purpose and usage.
3. **Error Handling**:
    - Use try-except blocks to catch potential errors and log them for debugging.

Source Control Practices

1. **Branching Strategy**:
    - Use feature branches for development, merging into `main` after code review and testing.
2. **Commit Messages**:
    - Write clear, concise commit messages that describe the changes and the reason for them.

Testing and Quality Assurance

1. **Writing Tests**:
    - Aim for high test coverage to catch bugs and prevent regressions.
    - Write tests for both positive scenarios and edge cases.
2. **Running Tests**:
    - Run the full test suite before committing changes to ensure functionality is intact.
3. **Code Reviews**:
    - Use pull requests for code reviews, ensuring at least one other developer reviews the changes.

Build and Deployment

1. **Continuous Integration (CI)**:
    - Set up CI to automate testing and ensure all merges into `main` pass the test suite.
2. **Deployment Procedures**:
    - Document the deployment steps and automate as much as possible.

Security

1. **Credential Management**:
    - Never hard-code credentials; use environment variables or secure vaults.
2. **Code Security Scans**:
    - Regularly perform static analysis to detect and fix security vulnerabilities.

Contribution Guidelines

1. **Issue Tracking**:
    - Use the issue tracker to assign and manage tasks.
2. **Feature Requests**:
    - Discuss new features extensively and document the specifications before implementation.

Additional Resources

- **Architecture Diagram**:
  - Refer to the architecture diagram for an overview of the application's structure.
- **API Documentation**:
  - If the application exposes any APIs, refer to the `API_DOCUMENTATION.txt` for details.

By adhering to the guidelines and practices outlined in this document, developers will contribute to the creation of a robust, maintainable, and secure application. For any clarifications or discussions, please reach out to the team lead or project manager.