**Step 1: Set Up the Development Environment and Repository**

**Objective**: Create a project repository and install necessary tools and dependencies.

* [x] Install Python 3.10 or later on your system.
* [x] Install Git for version control.
* [x] Create a new directory for the project (e.g., aegis).
* [x] Initialize a Git repository in the project directory.
* [x] Create a virtual environment to isolate dependencies.
* [x] Install required Python packages: pymupdf, customtkinter, playwright, openpyxl, pyinstaller.
* [x] Run playwright install to install browser binaries for web automation.
* [x] Create a .gitignore file to exclude unnecessary files (e.g., virtual environment, compiled files).
* [x] Commit initial setup to the repository and push to a remote (e.g., GitHub).

**Step 2: Implement PDF Data Extraction**

* **Objective**: Develop code to extract the ID field from a PDF using PyMuPDF.
* [x] Create a Python module (e.g., pdf\_parser.py) for PDF processing.
* [x] Use PyMuPDF (fitz) to open and read the PDF.
* [x] Implement logic to extract the ID based on the provided PDF structure (to be supplied later).
* [x] Add error handling for file not found or invalid PDF format.
* [x] Test the extraction with a sample PDF.
* [ ] Commit changes to the repository.

**Step 3: Build the CustomTkinter GUI**

* **Objective**: Create a user interface for file selection and process initiation.
* [ ] Create a Python module (e.g., [gui.py](http://gui.py)) for the GUI.
* [ ] Use CustomTkinter to design a window with:
  + [ ] A file selection button (browse) and drag-and-drop support.
  + [ ] A checkbox to toggle opening the Excel file after completion.
  + [ ] A "Start" button to trigger automation.
  + [ ] A text area or label for status messages ("Processing...", "Done!", errors).
* [ ] Implement file selection logic using tkinter.filedialog.
* [ ] Test the GUI for responsiveness and functionality.
* [ ] Commit changes to the repository.

**Step 4: Implement Playwright Web Automation**

* **Objective**: Automate navigation, form submission, and data scraping on the target website.
* [ ] Create a Python module (e.g., web\_automation.py) for web interactions.
* [ ] Use Playwright to:
  + [ ] Navigate to <https://www.thevirtualagent.co.za/>.
  + [ ] Handle sign-in (credentials to be provided or mocked for testing).
  + [ ] Navigate to the correct page for ID submission.
  + [ ] Fill in the ID field and submit the form.
  + [ ] Scrape up to three cell phone numbers from the results page.
* [ ] Add error handling for network issues, invalid credentials, or missing elements.
* [ ] Test automation with a sample ID.
* [ ] Commit changes to the repository.

**Step 5: Implement Excel File Writing**

* **Objective**: Generate an Excel file with extracted data using openpyxl.
* [ ] Create a Python module (e.g., excel\_writer.py) for Excel handling.
* [ ] Use openpyxl to create an .xlsx file with columns for ID and up to three cell phone numbers.
* [ ] Implement logic to write extracted data to the Excel file.
* [ ] Add an option to open the Excel file after saving (based on GUI checkbox).
* [ ] Add error handling for file write issues.
* [ ] Test Excel file generation and opening.
* [ ] Commit changes to the repository.

**Step 6: Implement Email Sending**

* **Objective**: Add functionality to send the Excel file as an email attachment.
* [ ] Create a Python module (e.g., email\_sender.py) for email handling.
* [ ] Use smtplib and email modules to send the Excel file via email.
* [ ] Configure email settings (e.g., SMTP server, credentials to be provided or mocked).
* [ ] Add error handling for SMTP connection or authentication issues.
* [ ] Test email sending with a sample Excel file.
* [ ] Commit changes to the repository.

**Step 7: Integrate Components and Add Feedback**

* **Objective**: Combine all components into a cohesive application with real-time feedback.
* [ ] Create a main Python script (e.g., [main.py](http://main.py)) to orchestrate the workflow:
  + [ ] Load the GUI.
  + [ ] Trigger PDF parsing, web automation, Excel writing, and email sending on "Start" button click.
* [ ] Update the GUI to display status messages during each step (e.g., "Processing PDF...", "Scraping website...", "Done!").
* [ ] Implement comprehensive error handling to display user-friendly error messages in the GUI.
* [ ] Test the integrated application end-to-end.
* [ ] Commit changes to the repository.

**Step 8: Package the Application**

* **Objective**: Bundle the application into a standalone Windows executable.
* [ ] Use PyInstaller to create a .exe file from [main.py](http://main.py).
* [ ] Configure PyInstaller to include all dependencies (e.g., PyMuPDF, Playwright browsers, CustomTkinter).
* [ ] Test the executable on a Windows machine to ensure it runs offline.
* [ ] Create a final release directory with the executable and necessary files.
* [ ] Commit the build script and final artifacts to the repository.

**Step 9: Final Testing and Documentation**

* **Objective**: Ensure the application is robust and well-documented.
* [ ] Perform end-to-end testing with multiple PDFs and scenarios.
* [ ] Create a [README.md](http://README.md) file with:
  + [ ] Project overview.
  + [ ] Setup instructions (for developers).
  + [ ] Usage instructions for the executable.
* [ ] Commit final changes and push to the remote repository.