Below is a **step-by-step approach** for **implementing** this real estate investment report generator, combining the **modular section framework**, **PropertyData API** integration, and **OpenAI-based** document processing. The plan outlines the **technical architecture**, **development phases**, and **key considerations** so you can move toward coding with clarity.

---

1. Architectural Overview

1. **Front-End (User Interface)**
    1. **Asset Setup**: Let users select “Land vs. Building,” proposed uses, and relevant scenarios (e.g., HMO, short-let).
    2. **Document Upload**: Drag-and-drop, file manager with AI-driven classification.
    3. **Report Section Editor**: Ability to see proposed sections, add/remove modules, and review each section’s draft.
    4. **Review & Finalize**: A final consolidated preview before export.
2. **Back-End (Core Logic & Data Processing)**
    1. **Section Manager**: Determines which sections are relevant, holds templates for each.
    2. **OpenAI Processor**:
        - Analyzes uploaded documents for data extraction (floor area, rents, etc.).
        - Generates textual summaries (e.g., rewriting location analysis).
    3. **PropertyData API Integrations**:
        - Endpoint calls for valuations, comps, planning, etc.
        - Manages the credit usage and caching.
    4. **Database** (optional, but recommended):
        - Store user sessions, uploaded docs metadata, extracted data points.
        - Keep an audit trail of changes.
3. **Data Flow**
    1. **User sets options** → System determines relevant sections.
    2. **User uploads docs** → System uses OpenAI to extract relevant fields.
    3. **System calls** PropertyData APIs for local comps, yields, planning, etc.
    4. **Section templates** are populated with merged data from both user docs + PropertyData + user overrides.
    5. **User** iterates on each section, possibly re-uploading more docs or editing text/numbers.
    6. **Final** assembled PDF or web-based report is generated.

---

2. Development Phases

Phase 1: Foundations & Basic UI

1. **Set Up Project Structure**
    - Use a web framework (e.g., **React** or **Vue** for front-end, **Node.js** or **Python** for back-end).
    - Decide if a monolith or microservices approach (likely a single back-end is sufficient to start).
2. **User Authentication & Basic Flow**
    - If needed, implement login, sign-up, session management.
    - Optionally, start with no auth if it's an internal tool.
3. **Basic "Create Report" Wizard**
    - Page 1: "Land or Building?"
    - Page 2: Proposed usage choices → System sets recommended sections.
    - Page 3: Show an overview of selected sections, user can remove or add from a list.
4. **Document Upload UI**
    - Allow user to upload multiple files, store them in a temporary location.
    - **Stub** out the AI extraction for now (or do minimal text parsing) until Phase 2.
5. **Section Template Structure**
    - Create placeholders for each major section (Site Description, Location, etc.).
    - The user can see these sections in a simple read-only or text area format.

Goal: Have a minimal but functional skeleton where a user can pick an asset type, upload docs (not processed yet), and see blank (or dummy) sections in a final placeholder PDF.

Phase 2: OpenAI Integration & Document Parsing

1. **OpenAI API Setup**
    - Integrate with GPT or similar model.
    - Create routines for:
        - **Document classification**: e.g., pass a snippet/metadata to GPT with system instructions like "Categorize this doc type."
        - **Data extraction**: Prompt GPT with "Here's a doc. The user wants fields: [Rent, address, etc.]. Extract them if possible."
2. **Metadata & Labeling**
    - After initial classification, show the user a short summary: "This doc might be a Rent Schedule. Confirm?"
    - The user can override the label if GPT is wrong.
3. **Extraction Logic**
    - For each relevant field (e.g., "floor area," "proposed rent," "budget"), define a structured approach:
        - **Prompt**: "Given this text, find the floor area. If more than one, note the conflict."

        - The system stores the extracted data in a structured format (e.g., JSON).
4. **Conflict Resolution**
    - If GPT identifies multiple values (two different square footages?), store them as candidate values.
    - Present a short UI for the user: “Which is correct?” or “Input the correct number.”
    - This resolves into a final chosen data set.
5. **Update Section Templates**
    - For “Site/Building Description,” the system populates placeholders with the extracted data.
    - For “Financial Analysis,” it uses “Refurb cost” from a doc if found, or a user override if not found.
6. **Iteration**
    - Let the user re-upload or add new docs at any time.
    - Re-run the extraction on the new doc. If relevant data is found, it updates the relevant fields.

Phase 3: PropertyData API Integration

1. **PropertyData Client**
    - Build a service or module (e.g., `propertyDataService.js`) that:
        - Authenticates with your **PropertyData API key**.
        - Provides easy functions like `getRents(postcode, bedrooms)`, `getPlanningData(postcode)`, etc.
    - Implement robust **error handling** (throttling, credit usage checks).
2. **Mapping Endpoints to Report Sections**
    - E.g.,
        - *Location analysis* might call `/crime`, `/demographics`, `/population`.
        - *Market analysis* might call `/prices`, `/rents`, `/growth`.
        - *Development analysis* might call `/development-calculator`, `/valuation-sale`, etc.
3. **Caching / Minimizing Redundant Calls**
    - Store results in a short-lived cache or database row (e.g., “We fetched prices for W14 9JH at 2025-06-10 14:00”).
    - If the user re-renders the same report without big changes, you don’t call the same endpoints again unnecessarily.
4. **Data Merging**
    - Combine user doc data with **API** data. If there's a mismatch, prefer user input for specific property details.
    - If there's no user doc data, fallback to API assumptions or typical market defaults.
5. **Integrating in the UI**

    - Possibly show a small indicator: “This rent figure is from the PropertyData API (last updated: X).”
    - Let the user override if needed.

Phase 4: Section Editor & Real-Time Calculations

1. **Section Editing**
    - Each section has structured data + a narrative text portion.
    - When a user modifies a key field (e.g., “bedrooms = 4”), it triggers recalculations and updates all references.
2. **Real-Time or On-Demand Recalculation**
    - A small event system can notify dependent sections: “Yield changed,” “Stamp duty changed,” etc.
    - Possibly show a “Recalculate” button if you want to avoid continuous calls.
3. **User-Driven Deletion of Sections**
    - If “Short Let” is no longer relevant, the user can remove that entire section.
    - The system must remove references in the final output as well.
4. **Handling Complex Modules**
    - For HMO, you might have sub-forms like “Number of double en-suites,” “Number of single shared-bath.”
    - For a big development, you might have multiple building types (12 flats + 4 retail units). The system aggregates them for a total GDV.

Phase 5: Final Report Generation & PDF

1. **Template Engine**
    - Possibly use a library like **Handlebars** or **Mustache** (for Node) or **LaTeX** if you want a pro PDF look.
    - Each section is a partial template with placeholders for data & text.
2. **Assembly Process**
    - The system merges user data, extracted data, and PropertyData results into each section’s placeholders.
    - Summaries / paragraphs might come from GPT. Ensure you store them so you don’t re-generate inconsistent text on final output.
3. **PDF Rendering**
    - Tools like **Puppeteer** or **wkhtmltopdf** can render HTML → PDF.
    - Or a dedicated PDF library if you prefer direct PDF generation.
4. **Final Review Screen**
    - Let the user see the entire doc in a preview before they hit “Export PDF.”
    - They can do last-minute text tweaks.
5. **Versioning & Storage**

    - Optionally store final PDF for future retrieval, or let the user download only.
    - Could store “Report v1,” “Report v2” in your DB.

Phase 6: Polishing, Security, and Scalability

1. **API Credit Monitoring**
    - Keep track of how many calls you’ve made. Provide warnings if near monthly limit.
    - Possibly add usage logs for each user or each report.
2. **Security & Compliance**
    - Secure storage of user docs with encryption at rest if needed.
    - If you handle personal data, ensure GDPR compliance.
    - Validate user input carefully to avoid injection vulnerabilities.
3. **Performance Optimizations**
    - Caching repeated calls.
    - Efficient queries to the DB.
    - Possibly queue certain heavy tasks (like large doc parsing or batch comps retrieval).
4. **User Feedback & Iterations**
    - Gather feedback from testers on UI flow, clarity of the final report, extraction accuracy.
    - Tweak prompts for GPT if certain data extraction is inaccurate.
    - Possibly add a “Confidence Score” from GPT extractions so the user knows how reliable the result is.
5. **Commercial Data**
    - As you expand, consider hooking up EGI or other commercial data sources once available.
    - Follow a similar approach for data extraction from commercial APIs.

---

3. Future Enhancements

- **Multi-User Collaboration**: Let multiple stakeholders (e.g., real estate agents, investors, architects) edit the same report.
- **Notifications & Workflows**: E.g., “Waiting on client to upload floor plan,” “Structural engineer doc needed.”
- **Analytics Dashboard**: Summarize how many reports are generated, average ROI, popular sections used, etc.
- **Auto-Populate from Sourcing**: If you have a property list or feed, auto-fill address, size, etc. to speed up report creation.
- **Advanced AI Vision**: Potentially parse scanned PDFs or images for textual data.

- **Machine Learning**: Over time, learn typical cost assumptions, local yield patterns, or frequently used disclaimers to automate more of the process.

---

4. Putting It All Together

1. **MVP**: Focus on getting the core flow correct (Phase 1 + minimal doc extraction + a few key PropertyData calls).
2. **Refinements**: Introduce deeper GPT extraction and conflict resolution.
3. **Full Production**: Layer on all relevant endpoints, real-time editing, robust PDF templates, caching, and scaling.

Throughout, keep the **user experience** front-of-mind. A streamlined UI that **walks** users through a logical process—from choosing an asset type, uploading docs, reconciling data, to finalizing a **beautiful** PDF—will be the key to adoption and satisfaction.

Conclusion

This plan breaks down exactly **how** to tackle each component of your real estate investment report generator—from **UI flows** and **back-end services** to **AI-driven data extraction** and **PropertyData API integration**. By proceeding in **phases**, you can iteratively deliver value, confirm the approach with stakeholders, and steadily add sophistication to achieve a robust, scalable product.