ChatGPT

Interactive PDF Viewer Options

Building an interactive PDF viewer/editor that supports text selection, highlighting, pop-up annotations, and seamless integration with the existing React/Next.js app is achievable using a mix of open-source libraries or commercial SDKs. A common approach is to base the viewer on Mozilla's **PDF.js** (an open-source PDF rendering engine) and then add annotation/highlight layers on top. For example, PDF.js is a powerful free library for rendering PDFs in the browser, but by itself it only **displays** PDFs (even rendering existing annotations) and doesn't provide built-in editing or highlighting [1] [2]. In practice, developers pair PDF.js with a React integration (or custom code) to overlay highlights or popups. For instance, one tutorial notes that with PDF.js "you'll implement basic rendering functionality… requiring custom development for additional features," whereas a commercial SDK offers many features out-of-the-box [3] [2].

Key open-source choices include: - **react-pdf-highlighter**: A React library built on PDF.js that provides annotation components. It explicitly supports text and rectangular highlights, even image highlights, and shows popover text for each highlight [4]. Its API lets users draw highlight boxes on selected text, and you can attach popovers (for notes or actions) to each highlight [4]. The library also supports programmatically scrolling to a saved highlight. It is MIT-licensed [5] and well-suited for adding interactive highlights in React.
- **react-pdf-viewer + Highlight Plugin**: The react-pdf-viewer is a robust React PDF component. It supports plugins, including a *highlight* plugin that lets users select text and automatically create highlights [6]. This plugin exposes callbacks (e.g. `renderHighlightTarget`) so you can render a custom UI (like a form or button) when the user selects text [7]. In other words, upon text selection it can display a pop-up element (for adding a note or triggering an AI rewrite), then overlay the highlight on the PDF. This approach handles text selection and highlight rendering without you writing low-level code. (Note: the react-pdf-viewer library itself is open source, but its license requires a commercial license for production use [8].)
- **react-pdf (Wojtek Maj's)**: A commonly used React wrapper for PDF.js. It makes displaying PDFs simple, but it has *no built-in annotation or highlight support* [1]. In fact, its documentation notes that libraries like `react-pdf` or the core PDF.js "only provide a viewer, not editing – we support rendering annotations for viewing, but we don't yet support adding annotations" [9]. Thus, using `react-pdf` alone would require manually adding highlight overlays or hooking into the PDF.js text layer yourself.

Other open solutions include lower-level libraries like `pdfjs-annotate.js` (which still doesn't do text highlights out of the box [1]), or building a completely custom overlay with Canvas or SVG. For example, you could render each PDF page on a HTML canvas (via PDF.js), then overlay a transparent HTML `<div>` layer containing `<span>`s for each text chunk, allowing CSS highlights and click events on the text. This approach is flexible but requires significant manual work (as one blog explains, PDF.js highlighting "requires manual rendering" and is best for simple needs [2]).

Commercial/Advanced SDKs

If budget and licensing allow, commercial SDKs offer far more functionality with less development effort. These SDKs wrap or extend PDF.js (or use their own renderers) to provide annotations, form filling, signatures, etc. For instance: - **PDFTron / Apryse WebViewer**: A mature, cross-platform PDF library. It advertises *"a comprehensive set of tools"* for reading, writing, editing, and annotating PDFs (and

even Office/CAD files) [10]. PDFTron supports 30+ annotation types including highlights, redactions, stamps, and more [11]. Its WebViewer can be embedded in React, and it includes APIs to programmatically search text, retrieve text positions (bounding "quads" for each character), and add highlight annotations in one step. For example, the WebViewer API lets you call `getTextPosition(page, startIndex, endIndex)` to return the exact on-page coordinates of each character of a search term [12] [13]. You could use this to highlight a clause's text by searching for it in the PDF. While powerful, PDFTron is commercial (with free trial), and integration would require adding its SDK instead of PDF.js.
- **PSPDFKit or Syncfusion PDF**: Similar to PDFTron, these commercial SDKs provide full editing/annotation toolkits. For example, Syncfusion's React PDF Viewer supports text highlights and "free text" annotations via methods like `addAnnotation()` [14] [15]. PSPDFKit (not shown here) likewise offers UI for highlighting and notes. These come with developer licenses.
- **PDF.js Express (by Apryse)**: A lighter option from the makers of PDFTron. It wraps PDF.js and provides annotations and forms. It has a "free viewer" tier (limited features) and a paid "Plus" tier for full editing [16] [17]. Notably, the documentation says the Express viewer "offers [a way] to quickly add annotation, e-signatures, and form filling to [React] PDF viewer" [17]. This could be a compromise if you only need a few users or limited scale.

Technical Considerations

Whichever approach you choose, consider the following:

- **DOM vs Canvas**: PDF.js renders pages on a `<canvas>`, with an invisible overlaid *text layer* for select/copy. Highlight libraries typically overlay HTML elements on top of this text layer. For example, a highlight plugin might insert absolutely-positioned `<div>`s (with a semi-transparent background) on the text spans. This ensures smooth highlighting and lets the browser handle text selection. React-pdf-highlighter and react-pdf-viewer already handle this detail internally [4] [6], whereas a custom solution would need to manually compute bounding boxes (from PDF.js's `getTextContent`) and position the highlight `<div>`. Note that if many highlights or very large PDFs are used, performance can be an issue; using the PDF.js worker thread properly (as recommended in `react-pdf-highlighter` docs) can mitigate slow rendering [18].

- **Click-to-Popup and Annotation UI**: To support "clicking on PDF to show a pop-up" or "clicking an analysis clause to highlight it in the PDF", you'll wire UI events. Most libraries fire events on highlight regions. In **react-pdf-highlighter**, each highlight can have an attached React popover for notes (it calls them "Popover text for highlights" [4]). In **react-pdf-viewer's** highlight plugin, you provide a `renderHighlightTarget` that appears after text selection – this can be a custom form or button to trigger AI actions [7]. On the other hand, if clicking on plain PDF text (not an existing highlight), you could listen for mouse events on the text layer and then display a floating overlay div at that position.

- **Linking Clauses to PDF Positions**: Your clause analysis backend already returns `position_start`/`position_end` for each clause in the extracted text [19]. To highlight the same text in the PDF viewer, you need to find where those offsets occur. One strategy is: after loading the PDF in the viewer, search for the clause's text string (via the text layer) and compute the page and bounding box. For example, PDFTron's WebViewer approach above finds "quads" for each letter of a search term [12] [13]. With PDF.js, you could use `pdfDocument.getPage(pageNum).getTextContent()` and scan for the clause substring, then use its glyph positions to draw a highlight. Some third-party viewers (like PDF.js Express) also provide APIs to find text or highlight annotations programmatically. Once located, you can scroll to that page and overlay a highlight. Conversely, clicking on a highlight in the PDF could call the analysis (you already have the clause summaries) and show a popup in your React UI.

- **React/Next.js Integration**: In Next.js, heavy client-side modules like PDF.js should be dynamically loaded or used only in client components (to avoid SSR issues). You can create a dedicated React component for the PDF viewer that is only rendered in the browser. This component can manage its own state (current page, highlights, etc.) and accept props like "clauses data" or callbacks. The rest of your app's layout (Tabs, summaries, etc.) can remain unchanged. According to your requirement, no major architecture changes are needed: you'd simply add a new viewer component (using one of the above libraries) into the review UI and hook up events between it and the clauses panel.

- **Editing vs. Viewing**: You mentioned wanting "light editing" (like text highlighting or annotation, possibly AI rewrite). All the highlighted libraries support *viewing* highlights and adding new ones on-the-fly, but none allow editing the PDF's actual content text inside the file (that would require PDF content streams editing, which is complex). However, since your rewrite feature seems to involve copying text out to AI and replacing it, the UI need only show a highlight and maybe replace the displayed text. For example, you could let the user select text and click "Rewrite" which sends that string to the AI and returns a suggestion. Then you could overlay the new text (e.g., in a tooltip or editable field), and if accepted, inject it back into the PDF viewer. (Note: storing those edits inside the actual PDF file would require rewriting the PDF, which is quite advanced – you might just handle it at the application layer or save as a new PDF.)

Recommendations

In summary, for a React-based app like ClauseIQ:

- **Open-Source Start:** A good first choice is to use **react-pdf-highlighter** or the **react-pdf-viewer with highlight plugin**, since both support text highlighting and popovers [4] [6]. They both run on top of PDF.js and fit naturally in a React UI. You can use `position_start`/`position_end` from your clauses to drive which text to highlight. For example, after processing, you could programmatically add highlights covering those text ranges (if you map them to coordinates).

- **Preview vs. Full PDF:** If you only need to highlight and annotate (not fully edit), you might continue using a canvas-based viewer. But if someday you need true PDF editing (forms or signatures), consider a commercial SDK. For now, an open-source approach minimizes new complexity in your stack.

- **Integration Work:** Whichever library you pick, you'll likely do the following: load the user's PDF file in the viewer component, overlay the clauses as highlights (either user-created or programmatically placed), and attach click handlers. When a user selects text in the PDF, show an annotation popup (using the library's highlight plugin UI or your own overlay). When a user clicks a clause in the side panel, find the text in the PDF and scroll/highlight it (this might use the PDF.js text layer to locate the span).

Overall, many teams successfully build contract-review viewers this way. For example, one comparison guide explicitly recommends using **PDF.js for basic viewing with custom code** or a **commercial PDF SDK (like Apryse/WebViewer)** for full annotation features [3] [2]. Given your desire to "keep everything nice and smooth" and avoid re-architecting, I'd start by prototyping with the React-PDF-Highlighter or React-PDF-Viewer plugin (both built on PDF.js) and hooking them into your clause data [4] [6]. They will let you implement clickable highlights and popups in the existing app without a major overhaul. If you later need more power or better performance, you could consider a commercial library (noting the license costs).

Sources: Official docs and examples of PDF libraries, and relevant expert write-ups on PDF.js and React integration [1] [2] [3] [4] [7] [17] [10] [12]. These describe how PDF.js-based viewers work, how annotation/highlighting plugins function, and what features commercial SDKs offer. They support the strategies outlined above.

---

[1] [5] [9] [18] react-pdf-highlighter - npm
https://www.npmjs.com/package/react-pdf-highlighter/v/5.0.0

[2] How to use PDF.js to highlight text programmatically
https://www.nutrient.io/blog/how-to-add-highlight-annotations-to-pdfs-in-javascript/

[3] How to build a React PDF viewer with PDF.js and Next.js | Nutrient
https://www.nutrient.io/blog/how-to-build-a-reactjs-viewer-with-pdfjs/

[4] react-pdf-highlighter/README.md at main · agentcooper/react-pdf-highlighter · GitHub
https://github.com/agentcooper/react-pdf-highlighter/blob/main/README.md

[6] [7] Highlight plugin - React PDF Viewer
https://react-pdf-viewer.dev/plugins/highlight/

[8] GitHub - react-pdf-viewer/react-pdf-viewer: A React component to view a PDF document
https://github.com/react-pdf-viewer/react-pdf-viewer

[10] [11] ComPDFKit VS PDFTron, Which PDF SDK Is Better? | by ComPDF | Medium
https://compdfkit.medium.com/compdfkit-vs-pdftron-which-pdf-sdk-is-better-69c91f67eb8d

[12] [13] Get Text Position in PDF using JavaScript | Apryse documentation
https://docs.apryse.com/web/guides/extraction/text-position

[14] [15] [16] [17] React PDF.js Viewer: Annotate, Form Fill, Sign | PDF.js Express
https://pdfjs.express/documentation/react

[19] page.tsx
https://github.com/vedantk1/legal-ai/blob/5fe5b62d6f30830fc1604f949ebecc957f87f4af/frontend/src/app/page.tsx