ChatGPT

AI-Powered PDF Study Guide Generator Web App

This project creates a web application where a user uploads an academic PDF and receives a downloadable PDF containing concise study notes, diagrams/flowcharts, and a generated question bank. The app will parse the PDF text, use AI models to summarize and generate quiz questions, and compile everything into a neatly formatted PDF. The backend runs on Node.js with Express, and the frontend is built in React using Material-UI (MUI) for the interface. All AI processing will use free, hosted Hugging Face models or open-source libraries so no paid APIs are needed [1] [2]. In fact, Hugging Face's Inference API offers free access to tasks like summarization and question generation [1], and you can obtain a free access token for it [3].

Key Features and Requirements

- **PDF Upload & Processing:** The user uploads an academic PDF via the React frontend. The backend accepts the file (using middleware like Multer) and extracts its full text (e.g. with a library such as *pdf-parse* [4]). Text extraction libraries like `pdf-parse` are open-source and can pull out page text cleanly for further AI processing [4].
- **Summarized Notes:** The app generates concise "short notes" by summarizing the extracted text. This uses a Hugging Face summarization model (for example, a BART or T5 model via a pipeline) to condense content into key points [5]. The output should be formatted as bullet-point notes in the final PDF for quick revision.
- **Diagrams and Flowcharts:** For visual aids, the app will create diagrams or flowcharts that capture the main processes or hierarchies from the text. This can be done by converting key summary points into a diagram definition. For instance, one can use the **Mermaid** library (JavaScript-based) to define flowcharts via text syntax [6]. Mermaid is free and can render complex diagrams from simple markup, making it ideal for dynamic diagram generation.
- **Question Bank Generation:** The app generates a variety of practice questions from the PDF content. The user selects which question types to include (e.g. theoretical, application-based, numerical, multiple-choice, fill-in-the-blank, true/false). We will use Hugging Face language models to create questions and answers. For example, a T5-based pipeline (`pipeline("multitask-qa-qg")`) can produce question-answer pairs from text [7]. Specialized prompts or fine-tuned models can generate multiple-choice questions (MCQs) with distractors [8], and simple transformations of statements can yield True/False or fill-in-the-blank items. All questions should be based solely on the PDF content.
- **PDF Output:** Finally, the app compiles the notes, diagrams, and questions into an HTML template and converts it into a PDF. Libraries like **Puppeteer** or **jsPDF/pdfMake** can take HTML content and generate a PDF on the server [9]. This PDF is then sent back to the frontend for the user to download. The final PDF thus includes sections of summarized notes, embedded images of flowcharts, and a list of generated questions formatted by type.

Technology Stack

- **Frontend:** Use **React** (with Create React App or Vite) and **Material-UI (MUI)** for styling. MUI provides ready-made components (buttons, checkboxes, progress spinners, etc.) to build a clean UI. The

upload form will use a file input (or a drag-and-drop dropzone component) to let the user select a PDF. Options can be presented as MUI checkboxes or toggles for question types. On submit, the form sends the file and preferences to the backend via an HTTP POST (using `fetch` or Axios).

- **Backend:** Use **Node.js** with **Express** to handle requests. Set up an `/upload` or `/generate` endpoint that accepts multipart file uploads (with a package like Multer). After receiving the PDF, use a PDF parsing library such as `pdf-parse` to extract the text [4]. Then process the text with AI models as described below. Finally, the server responds with the generated PDF file (or a download link).
- **AI/NLP Integration:** Leverage **Hugging Face Transformers** models via their free Inference API. You can use the `@huggingface/inference` NPM client or send HTTP requests to Hugging Face endpoints. For example, instantiate an `InferenceClient` with your free access token [3] to call summarization and question-generation models. Hugging Face has pipelines for summarization (`pipeline("summarization")`) [5] and multi-task QA/QG (`pipeline("multitask-qa-qg")`) [7]. Since only free APIs are allowed, ensure you only call Hugging Face's community-hosted endpoints. All recommended libraries and models here have free usage tiers.
- **PDF Generation:** To build the final PDF, convert an HTML document to PDF on the server. Options include **Puppeteer** (headless Chrome) or **pdfMake/pdfKit**. For instance, Puppeteer can load a styled HTML template (which includes the notes, images, and questions) and then print it to PDF via `page.pdf()` [9]. This is a common pattern: generate HTML on the backend, then render to PDF with such a library.

Frontend Implementation

- Use React with MUI to create a clean upload page. Include components for: file upload (e.g. `<Button>` triggering `<input type="file">` or a drag-drop zone), and form controls to select question types (MUI `<Checkbox>` or `<ToggleButton>` for each type like MCQ, True/False, etc.).
- On form submission, use `fetch` or Axios to POST the PDF file and options to the backend endpoint. Example: create a `FormData` object, append the file and preference fields, and send it to `/generate`.
- Show a loading indicator (MUI `<CircularProgress>` or similar) while waiting for the server's response.
- Once the backend returns, provide a download link or automatically trigger the PDF download. For example, receive the PDF bytes or a URL, then create a link with `href=URL.createObjectURL()` so the user can save it.

Backend Implementation

- **Express Setup:** Initialize an Express app. Use middleware like `multer()` to handle incoming file uploads. For example, `app.post('/generate', upload.single('pdfFile'), async (req, res) => { ... });` where `req.file` is the uploaded PDF.
- **PDF Text Extraction:** Inside the endpoint, use a Node PDF parsing library (e.g. `pdf-parse`) to extract text:

```
const pdf = require('pdf-parse');
const dataBuffer = req.file.buffer;
```

```
const pdfData = await pdf(dataBuffer);
const text = pdfData.text;
```

This pulls all the text from the PDF into a string for processing [4].

- **Summarization:** With the text extracted, call a summarization model. For example, use the Hugging Face inference client or pipeline:

```
const hf = new InferenceClient(HF_TOKEN);
const summary = await hf.textGeneration({
    model: "facebook/bart-large-cnn", // or another summarization model
    inputs: text,
    parameters: { max_new_tokens: 150 }
});
```

Or use a summarization pipeline for brevity [5]. This returns a short summary (or list of summaries). Use this to create bullet-point notes. Optionally split the PDF text into sections and summarize each for structured notes.

- **Diagram/Flowchart Generation:** From the summary or key sections, construct diagrams. For instance, identify processes or hierarchical relationships in the content and write a Mermaid definition. Example:

```
const mermaidDefinition = `
  graph LR
    A[Start] --> B{Decision}
    B --> C[Option 1]
    B --> D[Option 2]
    C --> E[End]
    D --> E
`;
```

Then render this to an image (Mermaid CLI or a JS renderer can output SVG/PNG). Insert the image into the PDF. Mermaid's documentation confirms it can generate such diagrams from text [6].

- **Question Bank Generation:** Generate each requested question type as follows:
- *Theoretical/Application Questions:* Use the HF question-generation pipeline. For each key paragraph or sentence, call `pipeline("multitask-qa-qg")` to get questions and answers [7].
- *Multiple-Choice Questions:* Use a specialized model or prompt. For example, one can use a pretrained model like "llama2-pdf-to-quizz-13b" which is trained to output MCQs from a document [8], or simply prompt GPT-style via Hugging Face's text-generation: "Generate a detailed MCQ about [subject] with 4 choices."
- *Fill-in-the-Blank:* Take important statements from the summary and replace a keyword with a blank. For instance, from "___ is the powerhouse of the cell," leaving a blank and the answer "mitochondria."
- *True/False:* Turn factual statements from the text into questions like "True or False: [statement]." You can generate a factual statement (or slight false alternative) using the model and format it accordingly.

Collate the generated questions, answer options, and correct answers. Label and format them by type (e.g. list all MCQs together, all T/F together, etc.).

- **Assembling Content:** Create an HTML or a template string that includes sections: a title, short notes (as bullet list), embedded diagram image(s), and the question bank (grouped by selected types). For diagrams, embed the rendered image (or SVG) into the HTML. You might use a server-side templating library or just string concatenation to build this.
- **PDF Generation:** Use an HTML-to-PDF converter. For example, launch Puppeteer in headless mode:

```
const browser = await puppeteer.launch();
const page = await browser.newPage();
await page.setContent(yourHTMLContent, {waitUntil: 'networkidle0'});
await page.pdf({ path: 'study_guide.pdf', format: 'A4' });
await browser.close();
```

This will create a PDF file that includes all notes, images, and questions. Alternatively, libraries like jsPDF or pdfMake can be used to programmatically build PDFs from scratch [9], but Puppeteer often makes formatting with HTML/CSS easier.

- **Response:** Finally, send the generated PDF back to the client. For example, set the response headers for a file download and pipe the PDF data, or provide a download URL that the frontend can fetch.

AI Model Usage (Hugging Face)

We will rely on free Hugging Face models and APIs for NLP tasks. Hugging Face's Inference API supports summarization and question generation [1]. The code can use the `@huggingface/inference` client: just initialize it with your access token (free to generate) [2]. For summarization, call a summarization model's endpoint (e.g., `"facebook/bart-large-cnn"` or `"t5-small"`). As shown in the Hugging Face docs, `pipeline("summarization")` can produce a concise summary from text [5]. For question generation, `pipeline("multitask-qa-qg")` will output QA pairs when given input text [7]. To produce multiple-choice questions, you might use a text-generation model with a prompt like "Generate one MCQ with 4 options based on the following passage." A Hugging Face discussion even shows a LLaMA-2 model trained to turn a document into MCQs [8] – this proves the concept (you could replicate it by a clever prompt or find a similar public model). All these Hugging Face services work under a free tier.

Diagram and Flowchart Generation

To create diagrams, we can programmatically use **Mermaid.js**. Mermaid allows writing flowcharts or diagrams in a simple markdown-like syntax and renders them in-browser or via a CLI [6]. In our backend, we would generate a Mermaid script from the summarized content (e.g. nodes and arrows for process flow) and then render it to an image. This image can be included in the output HTML before PDF conversion. Since Mermaid is open-source, it satisfies the "free" requirement. If more complex diagrams are needed, other free JS libraries (like D3.js or GoJS with a free license) could also be used, but Mermaid is straightforward for common flowcharts.

PDF Compilation and Download

After generating notes, diagrams, and questions, assemble them into a final document. Using an HTML template makes it easy: style sections with CSS (Material-UI theming or simple custom styles) and layout the content for readability. Then use a PDF library to convert that HTML to PDF. For example, **Puppeteer** can load the HTML content (as shown above) and output a clean PDF, handling page breaks as needed. The PDF should have a cover/title page, then sections like “Summary Notes”, “Diagrams”, and “Practice Questions”. Once the PDF is ready, send it back in the Express response (e.g. `res.download('study_guide.pdf')`), so the user can save or open it.

Free APIs and Licensing

All used services will be free. Hugging Face models and APIs are free for community use (with rate limits) and require only a free token to authenticate [3]. We will explicitly avoid any paid AI services. All libraries mentioned (React, MUI, Express, pdf-parse, Puppeteer, Mermaid, etc.) are open-source or have free tiers. The final product will rely exclusively on these free tools and hosted models, as specified.

Implementation Steps (Summary)

1. **Setup Frontend:** Create a React app and install Material-UI. Build components for file upload and question-type selection, and a submit button.
2. **Setup Backend:** Initialize an Express server with an endpoint (e.g. `/generate`). Use Multer to handle PDF uploads.
3. **Extract PDF Text:** In the endpoint handler, use `pdf-parse` (or similar) to convert the uploaded PDF into plain text [4].
4. **Summarize Content:** Call a Hugging Face summarization model (e.g. via `pipeline("summarization")`) on the extracted text to get bullet-point notes [5].
5. **Generate Diagrams:** Identify key processes in the notes and translate them into a Mermaid flowchart definition. Render the Mermaid graph to an image (using a headless renderer).
6. **Generate Questions:** For each requested type, use appropriate logic: a QG pipeline for open questions [7], text-generation with prompts for MCQs [8], etc. Collect the questions and answers.
7. **Build HTML Content:** Create an HTML template (or use a templating engine) that includes the summary notes (as lists), embeds the diagram image, and lists the questions grouped by type.
8. **Convert to PDF:** Use a PDF library (e.g. Puppeteer) to turn the HTML into a PDF file [9]. Ensure styles and page breaks are handled.
9. **Return PDF:** Send the generated PDF back to the frontend for download (e.g. using `res.sendFile` or `res.download`).
10. **Frontend Download:** On receiving the response, the React app should prompt the user to download the PDF.

Each of these steps will involve coding in React (JSX, components, API calls) and Node.js (Express routes, using NPM packages). By following this structured plan and using the cited tools and APIs, the Replit environment can be fed a clear specification to generate the complete application code.

Sources: The approach leverages known open-source libraries and Hugging Face models. For example, Node.js can use `pdf-parse` to extract text [4], Hugging Face provides summarization and QG pipelines

[5] [7], and Mermaid can render diagrams from text [6]. These references confirm the feasibility of each component of the solution.

---

[1] Exploring the Hugging Face Inference API in JavaScript - Free AI-Powered Course
https://www.educative.io/courses/exploring-the-hugging-face-inference-api-in-javascript

[2] [3] @huggingface/inference - npm
https://www.npmjs.com/package/@huggingface/inference

[4] Parsing PDFs in Node.js - LogRocket Blog
https://blog.logrocket.com/parsing-pdfs-node-js/

[5] Summarization
https://huggingface.co/docs/transformers/en/tasks/summarization

[6] mermaid-js/mermaid: Generation of diagrams like flowcharts or ...
https://github.com/mermaid-js/mermaid

[7] valhalla/t5-small-qa-qg-hl · Hugging Face
https://huggingface.co/valhalla/t5-small-qa-qg-hl

[8] fbellame/llama2-pdf-to-quizz-13b · Hugging Face
https://huggingface.co/fbellame/llama2-pdf-to-quizz-13b

[9] Best HTML to PDF libraries for Node.js - LogRocket Blog
https://blog.logrocket.com/best-html-pdf-libraries-node-js/