AI Integration Peelback MVP

Quick Overview

We'll use OpenAI's API to simplify medical papers - it's just HTTP requests, no ML knowledge needed.

Recommended Approach

1. Which AI Service?

OpenAI GPT-4o-mini

- Cost: ~$0.003 per paper (essentially free)
- Free $5 credit when signing up covers ~1,600 papers
- Best documentation for beginners
- Reliable structured outputs

2. How It Works

User uploads PDF → Extract text → Send to OpenAI API → Get simplified version → Display

3. Technical Implementation

Backend (Node.js):

- Keep API key secure on server
- Make API calls from backend endpoints
- Handle text extraction (pdf-parse library)

Frontend:

- Send requests to our backend
- Display structured results
- Handle complexity slider/audience selection

4. API Integration

javascript

```
const response = await fetch('https://api.openai.com/v1/chat/completions', {
  method: 'POST',
  headers: {
    'Authorization': 'Bearer API_KEY',
    'Content-Type': 'application/json'
```

```
  },
  body: JSON.stringify({
    model: "gpt-4o-mini",
    messages: [{role: "user", content: "Simplify: " + text}]
  })
});
```

5. Getting Structured Outputs

Instead of free-form text, we request JSON:

javascript

```
// Our prompt asks for specific sections:
{
  "title": "simplified title",
  "mainFindings": "summary",
  "sampleInfo": {
    "sampleSize": "450 participants",
    "demographics": "ages 25-65"
  },
  "keyStatistics": [...],
  "implications": "what this means"
}
```

This makes it easy to display data in the right sections of our page.

6. Different Audience Levels

We'll adjust prompts based on complexity slider:

- **Patient**: "Explain like I'm 12, use analogies, avoid jargon"
- **Policymaker**: "Executive summary, focus on policy implications"
- **Business**: "Focus on commercial opportunities and applications"

7. Frontend/Backend Architecture

Why We Need Both:

- **Backend (Node.js)**: Keeps API key secret, handles OpenAI calls, processes files
- **Frontend (HTML/CSS/JS)**: What users see and interact with

How They Communicate:

- Frontend sends requests to our backend server
- Backend processes and calls OpenAI
- Backend returns results to frontend
- Frontend displays the results

What Each Part Does:

Frontend (Client-Side):

- `index.html` - Upload form, complexity slider, results display
- `style.css` - Make it look good
- `app.js` - Handle file uploads, send data to backend, display results
- Runs in user's browser

Backend (Server-Side):

- `server.js` - Express server that receives requests
- `aiService.js` - Makes OpenAI API calls
- `pdfExtractor.js` - Extracts text from uploaded PDFs
- Runs on a server

Example Flow:

1. User uploads PDF in browser (frontend)
2. Frontend sends PDF to http://localhost:3000/api/process (backend)
3. Backend extracts text from PDF
4. Backend sends text to OpenAI API
5. Backend receives simplified JSON from OpenAI
6. Backend sends JSON back to frontend
7. Frontend displays it in nice HTML sections

8. Tech Stack Details

Frontend:

- Vanilla HTML/CSS/JavaScript (no React needed)
- Fetch API for HTTP requests
- File upload handling

Backend:

- Node.js with Express framework
- Libraries needed:
  - `express` - web server
  - `pdf-parse` - extract text from PDFs
  - `mammoth` - extract text from .docx
  - `dotenv` - manage API keys securely
  - `cors` - allow frontend to talk to backend

Setup:

bash

```
npm init -y
npm install express pdf-parse mammoth dotenv cors
```

9. Code Structure Example

Frontend (app.js) - What runs in browser:

```
javascript
// When user uploads file and clicks submit
async function handleSubmit(file, audienceLevel) {
  const formData = new FormData();
  formData.append('file', file);
  formData.append('audience', audienceLevel);

  // Send to OUR backend (not directly to OpenAI)
  const response = await fetch('http://localhost:3000/api/process', {
    method: 'POST',
    body: formData
  });

  const result = await response.json();
  displayResults(result); // Show simplified text
}
```

Backend (server.js) - What runs on our server:

```
javascript
const express = require('express');
const app = express();

// This endpoint receives requests from frontend
app.post('/api/process', async (req, res) => {
  // 1. Extract text from uploaded PDF
  const text = await extractPDFText(req.file);

  // 2. Call OpenAI (API key is secret here)
  const simplified = await callOpenAI(text, req.body.audience);

  // 3. Send back to frontend
  res.json(simplified);
});

app.listen(3000); // Server runs on port 3000
```

Why This Split?

- API keys are **never** exposed to users
- Users can't see our OpenAI calls
- We control costs (can add rate limiting)
- More secure

10. Key Technical Notes

- **Use environment variables** for API key
- **Add error handling** for failed API calls
- **Validate JSON** responses before displaying
- **Cache results** to avoid re-processing same paper
- **Set temperature to 0.3** for consistency

11. What Files We'll Create

Project Structure:

```
peelback/
├── frontend/
│   ├── index.html          # Upload form & results display
│   ├── style.css           # Styling
│   └── app.js              # Handles user interactions
├── backend/
│   ├── server.js           # Main Express server
│   ├── aiService.js        # OpenAI API calls
│   ├── pdfExtractor.js     # Text extraction
│   └── .env                # API key (never commit this!)
├── package.json
└── README.md
```

Frontend Files:

- HTML page with file upload button
- Complexity slider or audience selector
- Sections to display results (title, findings, sample info, etc.)
- JavaScript to send files to backend

Backend Files:

- Express server to receive uploads
- PDF text extraction logic
- OpenAI API integration
- Return formatted JSON to frontend

12. Development Workflow

During Development:

1. Run backend: `node backend/server.js` (localhost:3000)
2. Open frontend: `frontend/index.html` in browser
3. Upload PDF → Backend processes → Results display

Setting Up Project

Once Node.js is installed, here's how you'd set up Peelback:

bash

```
# 1. Create project folder

mkdir peelback

cd peelback


# 2. Initialize Node.js project (creates package.json)

npm init -y


# 3. Install the libraries we need

npm install express pdf-parse mammoth dotenv cors


# 4. Create your files

mkdir frontend backend

touch backend/server.js

touch frontend/index.html
```

What Gets Installed

When you run `npm install express`, it downloads:

- Express and its dependencies
- Saves them in a folder called `node_modules/`
- Updates `package.json` to track what you installed

Team only needs to share the code, not node_modules!

Each person runs `npm install` and gets the same libraries.

Running Backend

bash

 # Navigate to your project

cd peelback

 # Run the server

node backend/server.js

```

Output will be something like:

```

Server running on http://localhost:3000

Then open `frontend/index.html` in your browser

Team Setup

First person:

1. Creates project structure
2. Runs `npm init` and `npm install express pdf-parse ...`
3. Pushes to GitHub (without node_modules!)

Everyone else:

1. Clones the repository
2. Runs `npm install` (downloads all dependencies)

3. Creates their own `.env` file with API key
4. Runs `node backend/server.js`