==== app/admin/page.tsx ====

```
"use client";

import { useState, useRef, useEffect } from "react";
import { Button } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import { Label } from "@/components/ui/label";
import { Textarea } from "@/components/ui/textarea";
import {
  Table,
  TableBody,
  TableCell,
  TableHead,
  TableHeader,
  TableRow,
} from "@/components/ui/table";
import {
  Card,
  CardContent,
  CardDescription,
  CardFooter,
  CardHeader,
  CardTitle,
} from "@/components/ui/card";
import { Tabs, TabsContent, TabsList, TabsTrigger } from "@/components/ui/tabs";
import { Plus, Trash2, Upload, Download } from "lucide-react";
import { Sidebar } from "@/components/sidebar";
import { Navbar } from "@/components/navbar";
import { useToast } from "@/components/ui/use-toast";
import { useDropzone } from "react-dropzone";
import * as XLSX from "xlsx";

// Type for keyword
type Keyword = {
  id?: string;
  term: string;
  definition: string;
};

export default function AdminPage() {
  const { toast } = useToast();
  const [keywords, setKeywords] = useState<Keyword[]>([]);
  const [newKeyword, setNewKeyword] = useState("");
  const [newDefinition, setNewDefinition] = useState("");
  const [isLoading, setIsLoading] = useState(true);
  const fileInputRef = useRef<HTMLInputElement>(null);

  // Fetch keywords on component mount
  useEffect(() => {
    fetchKeywords();
  }, []);

  const fetchKeywords = async () => {
    try {
      setIsLoading(true);
      const response = await fetch("/api/keywords");
      if (!response.ok) throw new Error("Failed to fetch keywords");
      const data = await response.json();
      setKeywords(data);
    } catch (error) {
      console.error("Error fetching keywords:", error);
      toast({
        title: "Error",
```

```
        description: "Failed to fetch keywords",
        variant: "destructive",
      });
    } finally {
      setIsLoading(false);
    }
  };

  const handleAddKeyword = async () => {
    if (!newKeyword.trim() || !newDefinition.trim()) return;

    try {
      const response = await fetch("/api/keywords", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          term: newKeyword,
          definition: newDefinition,
        }),
      });

      if (!response.ok) throw new Error("Failed to add keyword");

      const data = await response.json();
      setKeywords([...keywords, data]);
      setNewKeyword("");
      setNewDefinition("");

      toast({
        title: "Success",
        description: "Keyword added successfully",
      });
    } catch (error) {
      console.error("Error adding keyword:", error);
      toast({
        title: "Error",
        description: "Failed to add keyword",
        variant: "destructive",
      });
    }
  };

  const handleRemoveKeyword = async (id: string) => {
    try {
      const response = await fetch(`/api/keywords/${id}`, {
        method: "DELETE",
      });

      if (!response.ok) throw new Error("Failed to delete keyword");

      setKeywords(keywords.filter((keyword) => keyword.id !== id));

      toast({
        title: "Success",
        description: "Keyword deleted successfully",
      });
    } catch (error) {
      console.error("Error deleting keyword:", error);
      toast({
        title: "Error",
        description: "Failed to delete keyword",
        variant: "destructive",
```

```
    });
  }
};

const processCSV = (text: string) => {
  const results: Keyword[] = [];
  const lines = text.split("\n");

  lines.forEach((line) => {
    if (!line.trim()) return;

    const [term, definition] = line.split(",").map((item) => item.trim());
    if (term && definition) {
      results.push({ term, definition });
    }
  });

  return results;
};

const processExcel = (arrayBuffer: ArrayBuffer) => {
  const workbook = XLSX.read(arrayBuffer);
  const worksheet = workbook.Sheets[workbook.SheetNames[0]];
  const data = XLSX.utils.sheet_to_json<{ term: string; definition: string }>(
    worksheet
  );

  return data.map((row) => ({
    term: row.term,
    definition: row.definition,
  }));
};

const handleFileUpload = async (files: File[]) => {
  if (files.length === 0) return;

  const file = files[0];
  try {
    let keywords: Keyword[] = [];

    if (file.name.endsWith(".csv")) {
      const text = await file.text();
      keywords = processCSV(text);
    } else if (file.name.endsWith(".xlsx") || file.name.endsWith(".xls")) {
      const arrayBuffer = await file.arrayBuffer();
      keywords = processExcel(arrayBuffer);
    } else {
      throw new Error("Unsupported file format");
    }

    if (keywords.length === 0) {
      throw new Error("No valid keywords found in file");
    }

    const response = await fetch("/api/keywords", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(keywords),
    });

    if (!response.ok) throw new Error("Failed to import keywords");
```

```
    await fetchKeywords();

    toast({
      title: "Success",
      description: `Imported ${keywords.length} keywords successfully`,
    });
  } catch (error) {
    console.error("Error importing keywords:", error);
    toast({
      title: "Error",
      description:
        error instanceof Error ? error.message : "Failed to import keywords",
      variant: "destructive",
    });
  }
};

const exportKeywords = () => {
  // Create worksheet
  const worksheet = XLSX.utils.json_to_sheet(
    keywords.map((k) => ({ term: k.term, definition: k.definition }))
  );

  // Create workbook
  const workbook = XLSX.utils.book_new();
  XLSX.utils.book_append_sheet(workbook, worksheet, "Keywords");

  // Generate Excel file
  XLSX.writeFile(workbook, "keywords.xlsx");
};

const { getRootProps, getInputProps } = useDropzone({
  onDrop: handleFileUpload,
  accept: {
    "text/csv": [".csv"],
    "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet": [
      ".xlsx",
    ],
    "application/vnd.ms-excel": [".xls"],
  },
  multiple: false,
});

return (
  <div className="flex h-screen bg-background">
    <Sidebar />
    <div className="flex flex-col flex-1 overflow-hidden">
      <Navbar />
      <main className="flex-1 overflow-auto p-4">
        <div className="container mx-auto">
          <h1 className="text-2xl font-bold mb-6">Admin Dashboard</h1>

          <Tabs defaultValue="keywords">
            <TabsList className="mb-4">
              <TabsTrigger value="keywords">Keyword Management</TabsTrigger>
              <TabsTrigger value="blog">Blog Management</TabsTrigger>
              <TabsTrigger value="settings">Settings</TabsTrigger>
            </TabsList>

            <TabsContent value="keywords" className="space-y-4">
              <Card>
                <CardHeader>
                  <CardTitle>Add New Keyword</CardTitle>
                  <CardDescription>
```

```
                Add a new keyword and its definition to the dictionary
              </CardDescription>
            </CardHeader>
            <CardContent className="space-y-4">
              <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
                <div className="space-y-2">
                  <Label htmlFor="keyword">Keyword</Label>
                  <Input
                    id="keyword"
                    value={newKeyword}
                    onChange={(e) => setNewKeyword(e.target.value)}
                    placeholder="Enter keyword"
                  />
                </div>
                <div className="space-y-2">
                  <Label htmlFor="definition">Definition</Label>
                  <Textarea
                    id="definition"
                    value={newDefinition}
                    onChange={(e) => setNewDefinition(e.target.value)}
                    placeholder="Enter definition"
                    rows={3}
                  />
                </div>
              </div>
            </CardContent>
            <CardFooter>
              <Button onClick={handleAddKeyword}>
                <Plus className="h-4 w-4 mr-2" />
                Add Keyword
              </Button>
            </CardFooter>
          </Card>

          <Card>
            <CardHeader>
              <CardTitle>Import/Export Keywords</CardTitle>
              <CardDescription>
                Upload a CSV or Excel file with keywords and definitions
                or export the current dictionary
              </CardDescription>
            </CardHeader>
            <CardContent>
              <div className="flex flex-col gap-4">
                <div
                  {...getRootProps()}
                  className="border-2 border-dashed rounded-md p-6 text-center cursor-pointer
hover:bg-muted/50 transition-colors"
                >
                  <input {...getInputProps()} />
                  <Upload className="h-8 w-8 mx-auto mb-2 text-muted-foreground" />
                  <p className="text-sm text-muted-foreground">
                    Drag & drop a CSV or Excel file here, or click to
                    select a file
                  </p>
                  <p className="text-xs text-muted-foreground mt-2">
                    File should have columns: term, definition
                  </p>
                </div>

                <div className="flex justify-end">
                  <Button
                    variant="outline"
                    onClick={exportKeywords}
```

```
          disabled={keywords.length === 0}
        >
          <Download className="h-4 w-4 mr-2" />
          Export Keywords
        </Button>
      </div>
    </div>
  </CardContent>
</Card>

<Card>
  <CardHeader className="flex flex-row items-center justify-between">
    <div>
      <CardTitle>Keyword Dictionary</CardTitle>
      <CardDescription>
        Manage your healthcare terminology dictionary
      </CardDescription>
    </div>
  </CardHeader>
  <CardContent>
    {isLoading ? (
      <div className="flex justify-center p-8">
        <div className="animate-spin rounded-full h-8 w-8 border-b-2 border-primary"></div>
      </div>
    ) : keywords.length === 0 ? (
      <div className="text-center p-8 text-muted-foreground">
        <p>
          No keywords found. Add some keywords or import from a
          file.
        </p>
      </div>
    ) : (
      <div className="border rounded-md">
        <Table>
          <TableHeader>
            <TableRow>
              <TableHead>Keyword</TableHead>
              <TableHead>Definition</TableHead>
              <TableHead className="w-[100px]">
                Actions
              </TableHead>
            </TableRow>
          </TableHeader>
          <TableBody>
            {keywords.map((keyword) => (
              <TableRow key={keyword.id}>
                <TableCell className="font-medium">
                  {keyword.term}
                </TableCell>
                <TableCell>{keyword.definition}</TableCell>
                <TableCell>
                  <Button
                    variant="ghost"
                    size="icon"
                    onClick={() =>
                      handleRemoveKeyword(keyword.id!)
                    }
                  >
                    <Trash2 className="h-4 w-4 text-destructive" />
                  </Button>
                </TableCell>
              </TableRow>
            ))}
          </TableBody>
```

```
                </Table>
              </div>
            )}
          </CardContent>
        </Card>
      </TabsContent>

      <TabsContent value="blog">
        <Card>
          <CardHeader>
            <CardTitle>Blog Management</CardTitle>
            <CardDescription>
              Manage blog posts for the healthcare reform portal
            </CardDescription>
          </CardHeader>
          <CardContent>
            <div className="flex justify-end mb-4">
              <Button asChild>
                <a href="/blog/new">Create New Blog Post</a>
              </Button>
            </div>
            <p>
              Blog management will be implemented in a future update.
            </p>
          </CardContent>
        </Card>
      </TabsContent>

      <TabsContent value="settings">
        <Card>
          <CardHeader>
            <CardTitle>Application Settings</CardTitle>
            <CardDescription>
              Configure application settings
            </CardDescription>
          </CardHeader>
          <CardContent>
            <p>Settings will be implemented in a future update.</p>
          </CardContent>
        </Card>
      </TabsContent>
    </Tabs>
  </div>
 </main>
 </div>
 </div>
 );
}



==== app/api/keywords/[id]/route.ts ====

import { NextResponse } from "next/server";
import { KeywordService } from "@/services/keyword-service";

export async function DELETE(
 request: Request,
 { params }: { params: { id: string } }
) {
 try {
  // Validate ID parameter
  const id = params.id;
  if (!id) {
```

```
      return NextResponse.json(
        { error: "Keyword ID is required" },
        { status: 400 }
      );
    }

    // Check if keyword exists before deleting
    const keyword = await KeywordService.getKeywordById(id);
    if (!keyword) {
      return NextResponse.json({ error: "Keyword not found" }, { status: 404 });
    }

    await KeywordService.removeKeyword(id);
    return NextResponse.json({ success: true });
  } catch (error) {
    console.error("Error deleting keyword:", error);
    return NextResponse.json(
      {
        error: "Failed to delete keyword",
        details: error instanceof Error ? error.message : String(error),
      },
      { status: 500 }
    );
  }
}


==== app/api/keywords/route.ts ====

import { NextResponse } from "next/server";
import { KeywordService } from "@/services/keyword-service";

export async function GET() {
  try {
    const keywords = await KeywordService.getKeywords();
    return NextResponse.json(keywords);
  } catch (error) {
    console.error("Error fetching keywords:", error);
    return NextResponse.json(
      {
        error: "Failed to fetch keywords",
        details: error instanceof Error ? error.message : String(error),
      },
      { status: 500 }
    );
  }
}

export async function POST(request: Request) {
  try {
    // Validate request content type
    const contentType = request.headers.get("content-type");
    if (!contentType || !contentType.includes("application/json")) {
      return NextResponse.json(
        { error: "Content-Type must be application/json" },
        { status: 400 }
      );
    }

    const data = await request.json();

    if (Array.isArray(data)) {
      // Bulk create/update
      // Validate array items
```

```
      for (const item of data) {
        if (
          !item.term ||
          typeof item.term !== "string" ||
          !item.definition ||
          typeof item.definition !== "string"
        ) {
          return NextResponse.json(
            {
              error: "Each keyword must have a term and definition as strings",
            },
            { status: 400 }
          );
        }
      }

      // Import keywords
      const keywords = await KeywordService.importKeywords(data);
      return NextResponse.json(keywords);
    } else {
      // Single create/update
      // Validate single item
      if (
        !data.term ||
        typeof data.term !== "string" ||
        !data.definition ||
        typeof data.definition !== "string"
      ) {
        return NextResponse.json(
          { error: "Keyword must have a term and definition as strings" },
          { status: 400 }
        );
      }

      const keyword = await KeywordService.addKeyword(
        data.term,
        data.definition
      );
      return NextResponse.json(keyword);
    }
  } catch (error) {
    console.error("Error creating/updating keywords:", error);
    return NextResponse.json(
      {
        error: "Failed to create/update keywords",
        details: error instanceof Error ? error.message : String(error),
      },
      { status: 500 }
    );
  }
} ???
}???


==== app/layout.tsx ====

import type React from "react";
import type { Metadata } from "next";
import { Inter } from "next/font/google";
import "./globals.css";
import { AuthProvider } from "@/contexts/auth-context";

const inter = Inter({ subsets: ["latin"] });
```

```
export const metadata: Metadata = {
  title: "Vermont Healthcare Reform Portal",
  description:
    "A platform for analyzing healthcare reform documents in Vermont",
  generator: "v0.dev",
};

export default function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode;
}>) {
  return (
    <html lang="en">
      <body className={inter.className}>
        <AuthProvider>{children}</AuthProvider>
      </body>
    </html>
  );
}

import "./globals.css";


==== app/page.tsx ====

"use client";

import { useState, useEffect } from "react";
import { Sidebar } from "@/components/sidebar";
import { Navbar } from "@/components/navbar";
import { DocumentViewer } from "@/components/document-viewer";
import { KeywordHighlighter } from "@/components/keyword-highlighter";
import { NewsHeadlines } from "@/components/news-headlines";
import { NewsArticle } from "@/components/news-article";
import { ResizableGrid } from "@/components/resizable-grid";

// Define storage key for document text
const DOCUMENT_TEXT_STORAGE_KEY = "vt-healthcare-document-text";

export default function Home() {
  const [documentText, setDocumentText] = useState<string>("");
  const [selectedArticle, setSelectedArticle] = useState<any>(null);

  // Load saved document text from localStorage on component mount
  useEffect(() => {
    if (typeof window !== "undefined") {
      const savedText = localStorage.getItem(DOCUMENT_TEXT_STORAGE_KEY);
      if (savedText) {
        setDocumentText(savedText);
      }
    }
  }, []);

  // Handle document text extraction
  const handleTextExtracted = (text: string) => {
    setDocumentText(text);
    // Save to localStorage
    if (typeof window !== "undefined") {
      localStorage.setItem(DOCUMENT_TEXT_STORAGE_KEY, text);
    }
  };

  return (
```

```
    <div className="flex h-screen bg-background">
      <Sidebar />
      <div className="flex flex-col flex-1 overflow-hidden">
        <Navbar />
        <main className="flex-1 overflow-hidden p-4">
          <ResizableGrid>
            <DocumentViewer onTextExtracted={handleTextExtracted} />
            <KeywordHighlighter text={documentText} />
            <NewsHeadlines onArticleSelected={setSelectedArticle} />
            <NewsArticle article={selectedArticle} />
          </ResizableGrid>
        </main>
      </div>
    </div>
  );
}


==== app/settings/page.tsx ====

"use client";

import { Sidebar } from "@/components/sidebar";
import { Navbar } from "@/components/navbar";
import {
  Card,
  CardContent,
  CardDescription,
  CardFooter,
  CardHeader,
  CardTitle,
} from "@/components/ui/card";
import { Tabs, TabsContent, TabsList, TabsTrigger } from "@/components/ui/tabs";
import { Button } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import { Label } from "@/components/ui/label";
import { Switch } from "@/components/ui/switch";
import { Separator } from "@/components/ui/separator";

export default function SettingsPage() {
  return (
    <div className="flex h-screen bg-background">
      <Sidebar />
      <div className="flex flex-col flex-1 overflow-hidden">
        <Navbar />
        <main className="flex-1 overflow-auto p-4">
          <div className="container mx-auto max-w-4xl">
            <h1 className="text-2xl font-bold mb-6">Settings</h1>

            <Tabs defaultValue="general">
              <TabsList className="mb-6">
                <TabsTrigger value="general">General</TabsTrigger>
                <TabsTrigger value="appearance">Appearance</TabsTrigger>
                <TabsTrigger value="notifications">Notifications</TabsTrigger>
                <TabsTrigger value="advanced">Advanced</TabsTrigger>
              </TabsList>

              <TabsContent value="general">
                <Card>
                  <CardHeader>
                    <CardTitle>General Settings</CardTitle>
                    <CardDescription>
                      Manage your general application settings
                    </CardDescription>
```

```
    </CardHeader>
    <CardContent className="space-y-6">
      <div className="space-y-2">
        <Label htmlFor="name">Display Name</Label>
        <Input
          id="name"
          defaultValue="Vermont Healthcare Reform Portal"
        />
      </div>

      <div className="space-y-2">
        <Label htmlFor="email">Contact Email</Label>
        <Input
          id="email"
          type="email"
          defaultValue="contact@vthealthcare.org"
        />
      </div>

      <Separator />

      <div className="flex items-center justify-between">
        <div className="space-y-0.5">
          <Label htmlFor="auto-save">Auto-save Documents</Label>
          <p className="text-sm text-muted-foreground">
            Automatically save documents when changes are made
          </p>
        </div>
        <Switch id="auto-save" defaultChecked />
      </div>

      <div className="flex items-center justify-between">
        <div className="space-y-0.5">
          <Label htmlFor="analytics">Usage Analytics</Label>
          <p className="text-sm text-muted-foreground">
            Collect anonymous usage data to improve the
            application
          </p>
        </div>
        <Switch id="analytics" defaultChecked />
      </div>
    </CardContent>
    <CardFooter>
      <Button>Save Changes</Button>
    </CardFooter>
  </Card>
</TabsContent>

<TabsContent value="appearance">
  <Card>
    <CardHeader>
      <CardTitle>Appearance Settings</CardTitle>
      <CardDescription>
        Customize the look and feel of the application
      </CardDescription>
    </CardHeader>
    <CardContent className="space-y-6">
      <div className="space-y-2">
        <Label>Theme</Label>
        <div className="flex gap-4">
          <Button variant="outline" className="flex-1">
            Light
          </Button>
          <Button variant="outline" className="flex-1">
```

```
            Dark
          </Button>
          <Button variant="outline" className="flex-1">
            System
          </Button>
        </div>
      </div>

      <Separator />

      <div className="flex items-center justify-between">
        <div className="space-y-0.5">
          <Label htmlFor="animations">Interface Animations</Label>
          <p className="text-sm text-muted-foreground">
            Enable animations throughout the interface
          </p>
        </div>
        <Switch id="animations" defaultChecked />
      </div>

      <div className="flex items-center justify-between">
        <div className="space-y-0.5">
          <Label htmlFor="compact">Compact Mode</Label>
          <p className="text-sm text-muted-foreground">
            Use a more compact layout to fit more content on
            screen
          </p>
        </div>
        <Switch id="compact" />
      </div>
    </CardContent>
    <CardFooter>
      <Button>Save Changes</Button>
    </CardFooter>
  </Card>
</TabsContent>

<TabsContent value="notifications">
  <Card>
    <CardHeader>
      <CardTitle>Notification Settings</CardTitle>
      <CardDescription>
        Manage your notification preferences
      </CardDescription>
    </CardHeader>
    <CardContent>
      <p>
        Notification settings will be implemented in a future
        update.
      </p>
    </CardContent>
  </Card>
</TabsContent>

<TabsContent value="advanced">
  <Card>
    <CardHeader>
      <CardTitle>Advanced Settings</CardTitle>
      <CardDescription>
        Configure advanced application settings
      </CardDescription>
    </CardHeader>
    <CardContent>
      <p>
```

```
                Advanced settings will be implemented in a future update.
              </p>
            </CardContent>
          </Card>
        </TabsContent>
      </Tabs>
    </div>
  </main>
 </div>
 </div>
 );
}


==== components/document-viewer copy.tsx ====

"use client";

import type React from "react";
import { useState, useRef, useEffect } from "react";
import {
  ChevronLeft,
  ChevronRight,
  Upload,
  FileText,
  ZoomIn,
  ZoomOut,
  MessageSquare,
} from "lucide-react";
import { Button } from "@/components/ui/button";
import Script from "next/script";
import Link from "next/link";

// Define a global type for the PDF.js library
declare global {
  interface Window {
    pdfjsLib: any;
  }
}

// Define storage keys
const DOCUMENT_URL_STORAGE_KEY = "vt-healthcare-document-url";
const DOCUMENT_NAME_STORAGE_KEY = "vt-healthcare-document-name";
const DOCUMENT_PAGE_STORAGE_KEY = "vt-healthcare-document-page";
const DOCUMENT_SCALE_STORAGE_KEY = "vt-healthcare-document-scale";

export function DocumentViewer({
  onTextExtracted,
}: {
  onTextExtracted: (text: string) => void;
}) {
  const [currentPage, setCurrentPage] = useState(1);
  const [totalPages, setTotalPages] = useState(0);
  const [documentUrl, setDocumentUrl] = useState<string | null>(null);
  const [documentName, setDocumentName] = useState<string>("");
  const [isLoading, setIsLoading] = useState(false);
  const [scale, setScale] = useState(1.0);
  const [pdfLoaded, setPdfLoaded] = useState(false);
  const [pdfDocument, setPdfDocument] = useState<any>(null);
  const fileInputRef = useRef<HTMLInputElement>(null);
  const canvasRef = useRef<HTMLCanvasElement>(null);
  const containerRef = useRef<HTMLDivElement>(null);

  // Load saved document state from localStorage
```

```
  useEffect(() => {
    if (typeof window !== "undefined") {
      const savedUrl = localStorage.getItem(DOCUMENT_URL_STORAGE_KEY);
      const savedName = localStorage.getItem(DOCUMENT_NAME_STORAGE_KEY);
      const savedPage = localStorage.getItem(DOCUMENT_PAGE_STORAGE_KEY);
      const savedScale = localStorage.getItem(DOCUMENT_SCALE_STORAGE_KEY);

      if (savedUrl) {
        setDocumentUrl(savedUrl);
      }
      if (savedName) {
        setDocumentName(savedName);
      }
      if (savedPage) {
        setCurrentPage(Number.parseInt(savedPage, 10));
      }
      if (savedScale) {
        setScale(Number.parseFloat(savedScale));
      }
    }
  }, []);

  // Save document state to localStorage when it changes
  useEffect(() => {
    if (typeof window !== "undefined") {
      if (documentUrl) {
        localStorage.setItem(DOCUMENT_URL_STORAGE_KEY, documentUrl);
      }
      if (documentName) {
        localStorage.setItem(DOCUMENT_NAME_STORAGE_KEY, documentName);
      }
      localStorage.setItem(DOCUMENT_PAGE_STORAGE_KEY, currentPage.toString());
      localStorage.setItem(DOCUMENT_SCALE_STORAGE_KEY, scale.toString());
    }
  }, [documentUrl, documentName, currentPage, scale]);

  // Function to render the current PDF page
  const renderPage = async (pageNum: number) => {
    if (!canvasRef.current || !pdfDocument) return;

    try {
      // Get the page
      const page = await pdfDocument.getPage(pageNum);
      const viewport = page.getViewport({ scale });

      // Set canvas dimensions
      const canvas = canvasRef.current;
      const context = canvas.getContext("2d");
      if (!context) return;

      canvas.height = viewport.height;
      canvas.width = viewport.width;

      // Clear canvas
      context.clearRect(0, 0, canvas.width, canvas.height);

      // Render the page
      await page.render({
        canvasContext: context,
        viewport: viewport,
      }).promise;

      // Extract text from the current page
      const textContent = await page.getTextContent();
```

```
    const pageText = textContent.items.map((item: any) => item.str).join(" ");
    onTextExtracted(pageText);
  } catch (error) {
    console.error("Error rendering PDF page:", error);
  }
};

// Extract text from PDF
const extractTextFromPDF = async () => {
  if (!pdfDocument) return;

  try {
    let fullText = "";

    for (let i = 1; i <= pdfDocument.numPages; i++) {
      const page = await pdfDocument.getPage(i);
      const textContent = await page.getTextContent();
      const pageText = textContent.items
        .map((item: any) => item.str)
        .join(" ");
      fullText += pageText + " ";
    }

    onTextExtracted(fullText);
  } catch (error) {
    console.error("Error extracting text from PDF:", error);
  }
};

// Load PDF when document URL changes
useEffect(() => {
  if (!documentUrl || !pdfLoaded) return;

  setIsLoading(true);

  const loadPDF = async () => {
    try {
      // Use window.pdfjsLib which is loaded from CDN
      const loadingTask = window.pdfjsLib.getDocument(documentUrl);
      const pdf = await loadingTask.promise;

      setPdfDocument(pdf);
      setTotalPages(pdf.numPages);

      // Extract text after loading
      await extractTextFromPDF();

      setIsLoading(false);
    } catch (error) {
      console.error("Error loading PDF:", error);
      setIsLoading(false);
    }
  };

  loadPDF();
}, [documentUrl, pdfLoaded]);

// Render page when current page or scale changes
useEffect(() => {
  if (pdfDocument) {
    renderPage(currentPage);
  }
}, [currentPage, scale, pdfDocument]);
```

```
const handleFileUpload = (e: React.ChangeEvent<HTMLInputElement>) => {
  const file = e.target.files?.[0];
  if (!file) return;

  // Revoke previous object URL to prevent memory leaks
  if (documentUrl && documentUrl.startsWith("blob:")) {
    URL.revokeObjectURL(documentUrl);
  }

  setDocumentName(file.name);
  const fileUrl = URL.createObjectURL(file);
  setDocumentUrl(fileUrl);
  setCurrentPage(1); // Reset to first page
};

const zoomIn = () => setScale((prev) => Math.min(prev + 0.2, 3.0));
const zoomOut = () => setScale((prev) => Math.max(prev - 0.2, 0.5));

return (
  <>
    {/* Load PDF.js from CDN */}
    <Script
      src="https://cdnjs.cloudflare.com/ajax/libs/pdf.js/3.4.120/pdf.min.js"
      onLoad={() => {
        // Set worker source after library loads
        window.pdfjsLib.GlobalWorkerOptions.workerSrc =
          "https://cdnjs.cloudflare.com/ajax/libs/pdf.js/3.4.120/pdf.worker.min.js";
        setPdfLoaded(true);
      }}
      strategy="afterInteractive"
    />

    <div className="flex flex-col h-full">
      <div className="flex items-center justify-between p-2 border-b">
        <h3 className="font-medium">Document Viewer</h3>
        <div className="flex items-center gap-2">
          {documentUrl && (
            <Button variant="outline" size="sm" asChild>
              <Link
                href={`/comments?document=${encodeURIComponent(
                  documentName
                )}&page=${currentPage}`}
              >
                <MessageSquare className="h-4 w-4 mr-2" />
                Add Comments
              </Link>
            </Button>
          )}
          <Button
            variant="outline"
            size="sm"
            onClick={() => fileInputRef.current?.click()}
          >
            <Upload className="h-4 w-4 mr-2" />
            Upload PDF
            <input
              ref={fileInputRef}
              type="file"
              accept=".pdf"
              className="hidden"
              onChange={handleFileUpload}
              aria-label="Upload PDF document"
            />
          </Button>
```

```
  </div>
</div>

<div
  ref={containerRef}
  className="flex-1 flex items-center justify-center bg-muted/30 overflow-auto relative"
>
  {isLoading ? (
    <div className="flex flex-col items-center gap-2">
      <div className="animate-spin rounded-full h-8 w-8 border-b-2 border-primary"></div>
      <p className="text-sm text-muted-foreground">
        Processing document...
      </p>
    </div>
  ) : documentUrl ? (
    <div className="w-full h-full flex flex-col items-center">
      <div className="flex-1 w-full flex items-center justify-center overflow-auto p-2">
        <canvas
          ref={canvasRef}
          className="shadow-lg"
          aria-label={`PDF page ${currentPage} of ${totalPages}`}
        />
      </div>
    </div>
  ) : (
    <div className="flex flex-col items-center gap-2 text-muted-foreground">
      <FileText className="h-12 w-12" />
      <p>Upload a PDF document to begin</p>
    </div>
  )}
</div>

{documentUrl && totalPages > 0 && (
  <div className="p-2 flex items-center justify-between gap-2 w-full border-t">
    <div className="flex items-center gap-2">
      <Button
        variant="outline"
        size="icon"
        onClick={zoomOut}
        disabled={scale <= 0.5}
        aria-label="Zoom out"
      >
        <ZoomOut className="h-4 w-4" />
      </Button>
      <span className="text-sm">{Math.round(scale * 100)}%</span>
      <Button
        variant="outline"
        size="icon"
        onClick={zoomIn}
        disabled={scale >= 3.0}
        aria-label="Zoom in"
      >
        <ZoomIn className="h-4 w-4" />
      </Button>
    </div>

    <div className="flex items-center gap-2">
      <Button
        variant="outline"
        size="icon"
        disabled={currentPage <= 1}
        onClick={() => setCurrentPage((p) => Math.max(1, p - 1))}
        aria-label="Previous page"
      >
```

```
              <ChevronLeft className="h-4 w-4" />
            </Button>
            <span className="text-sm">
              Page {currentPage} of {totalPages}
            </span>
            <Button
              variant="outline"
              size="icon"
              disabled={currentPage >= totalPages}
              onClick={() =>
                setCurrentPage((p) => Math.min(totalPages, p + 1))
              }
              aria-label="Next page"
            >
              <ChevronRight className="h-4 w-4" />
            </Button>
          </div>
        </div>
      )}
    </div>
  </>
 );
}


==== components/document-viewer.tsx ====

"use client";

import type React from "react";
import { useState, useRef, useEffect } from "react";
import {
  ChevronLeft,
  ChevronRight,
  Upload,
  FileText,
  ZoomIn,
  ZoomOut,
  MessageSquare,
  RefreshCw,
} from "lucide-react";
import { Button } from "@/components/ui/button";
import Script from "next/script";
import Link from "next/link";

// Define a global type for the PDF.js library
declare global {
  interface Window {
    pdfjsLib: any;
  }
}

// Define storage keys
const DOCUMENT_URL_STORAGE_KEY = "vt-healthcare-document-url";
const DOCUMENT_NAME_STORAGE_KEY = "vt-healthcare-document-name";
const DOCUMENT_PAGE_STORAGE_KEY = "vt-healthcare-document-page";
const DOCUMENT_SCALE_STORAGE_KEY = "vt-healthcare-document-scale";

export function DocumentViewer({
  onTextExtracted,
}: {
  onTextExtracted: (text: string) => void;
}) {
  const [currentPage, setCurrentPage] = useState(1);
```

```
  const [totalPages, setTotalPages] = useState(0);
  const [documentUrl, setDocumentUrl] = useState<string | null>(null);
  const [documentName, setDocumentName] = useState<string>("");
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState<string | null>(null);
  const [scale, setScale] = useState(1.0);
  const [pdfLoaded, setPdfLoaded] = useState(false);
  const [pdfDocument, setPdfDocument] = useState<any>(null);
  const [pdfBytes, setPdfBytes] = useState<Uint8Array | null>(null);
  const fileInputRef = useRef<HTMLInputElement>(null);
  const canvasRef = useRef<HTMLCanvasElement>(null);
  const containerRef = useRef<HTMLDivElement>(null);
  const blobUrlRef = useRef<string | null>(null);

  // Load saved document state from localStorage
  useEffect(() => {
    if (typeof window !== "undefined") {
      const savedUrl = localStorage.getItem(DOCUMENT_URL_STORAGE_KEY);
      const savedName = localStorage.getItem(DOCUMENT_NAME_STORAGE_KEY);
      const savedPage = localStorage.getItem(DOCUMENT_PAGE_STORAGE_KEY);
      const savedScale = localStorage.getItem(DOCUMENT_SCALE_STORAGE_KEY);

      if (savedName) {
        setDocumentName(savedName);
      }
      if (savedPage) {
        setCurrentPage(Number.parseInt(savedPage, 10));
      }
      if (savedScale) {
        setScale(Number.parseFloat(savedScale));
      }

      // We'll handle the URL separately since we need to fetch the file again
      if (
        savedUrl &&
        savedUrl.startsWith("http") &&
        !savedUrl.startsWith("blob:")
      ) {
        setDocumentUrl(savedUrl);
      }
    }
  }, []);

  // Save document state to localStorage when it changes
  useEffect(() => {
    if (typeof window !== "undefined") {
      if (documentUrl && !documentUrl.startsWith("blob:")) {
        localStorage.setItem(DOCUMENT_URL_STORAGE_KEY, documentUrl);
      }
      if (documentName) {
        localStorage.setItem(DOCUMENT_NAME_STORAGE_KEY, documentName);
      }
      localStorage.setItem(DOCUMENT_PAGE_STORAGE_KEY, currentPage.toString());
      localStorage.setItem(DOCUMENT_SCALE_STORAGE_KEY, scale.toString());
    }
  }, [documentUrl, documentName, currentPage, scale]);

  // Function to render the current PDF page
  const renderPage = async (pageNum: number) => {
    if (!canvasRef.current || !pdfDocument) return;
    setError(null);

    try {
      // Get the page
```

```
    const page = await pdfDocument.getPage(pageNum);
    const viewport = page.getViewport({ scale });

    // Set canvas dimensions
    const canvas = canvasRef.current;
    const context = canvas.getContext("2d");
    if (!context) return;

    canvas.height = viewport.height;
    canvas.width = viewport.width;

    // Clear canvas
    context.clearRect(0, 0, canvas.width, canvas.height);

    // Render the page
    await page.render({
      canvasContext: context,
      viewport: viewport,
    }).promise;

    // Extract text from the current page
    const textContent = await page.getTextContent();
    const pageText = textContent.items.map((item: any) => item.str).join(" ");
    onTextExtracted(pageText);
  } catch (error) {
    console.error("Error rendering PDF page:", error);
    setError("Failed to render PDF page. Please try again.");
  }
};

// Extract text from PDF
const extractTextFromPDF = async () => {
  if (!pdfDocument) return;

  try {
    let fullText = "";

    for (let i = 1; i <= pdfDocument.numPages; i++) {
      const page = await pdfDocument.getPage(i);
      const textContent = await page.getTextContent();
      const pageText = textContent.items
        .map((item: any) => item.str)
        .join(" ");
      fullText += pageText + " ";
    }

    onTextExtracted(fullText);
  } catch (error) {
    console.error("Error extracting text from PDF:", error);
    setError("Failed to extract text from PDF. Please try again.");
  }
};

// Fetch and load PDF data
const fetchAndLoadPDF = async (url: string) => {
  setIsLoading(true);
  setError(null);

  try {
    // Fetch the PDF file as an ArrayBuffer
    const response = await fetch(url);

    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
```

```
    }

    const arrayBuffer = await response.arrayBuffer();
    const bytes = new Uint8Array(arrayBuffer);

    // Store the PDF bytes
    setPdfBytes(bytes);

    // Load the PDF using PDF.js
    const loadingTask = window.pdfjsLib.getDocument({ data: bytes });
    const pdf = await loadingTask.promise;

    setPdfDocument(pdf);
    setTotalPages(pdf.numPages);

    // Extract text after loading
    await extractTextFromPDF();

    setIsLoading(false);
  } catch (error) {
    console.error("Error loading PDF:", error);
    setError(
      `Failed to load PDF: ${
        error instanceof Error ? error.message : String(error)
      }`
    );
    setIsLoading(false);
  }
};

// Load PDF when document URL changes
useEffect(() => {
  if (!documentUrl || !pdfLoaded) return;

  // Don't try to load from blob URLs directly
  if (!documentUrl.startsWith("blob:")) {
    fetchAndLoadPDF(documentUrl);
  }
}, [documentUrl, pdfLoaded]);

// Render page when current page or scale changes
useEffect(() => {
  if (pdfDocument) {
    renderPage(currentPage);
  }
}, [currentPage, scale, pdfDocument]);

// Clean up blob URLs when component unmounts or when a new file is loaded
useEffect(() => {
  return () => {
    if (blobUrlRef.current) {
      URL.revokeObjectURL(blobUrlRef.current);
    }
  };
}, []);

const handleFileUpload = async (e: React.ChangeEvent<HTMLInputElement>) => {
  const file = e.target.files?.[0];
  if (!file) return;

  setIsLoading(true);
  setError(null);

  try {
```

```
    // Revoke previous object URL to prevent memory leaks
    if (blobUrlRef.current) {
      URL.revokeObjectURL(blobUrlRef.current);
      blobUrlRef.current = null;
    }

    setDocumentName(file.name);

    // Read the file as an ArrayBuffer
    const arrayBuffer = await file.arrayBuffer();
    const bytes = new Uint8Array(arrayBuffer);

    // Store the PDF bytes
    setPdfBytes(bytes);

    // Create a blob URL for download/sharing purposes only
    const blob = new Blob([bytes], { type: "application/pdf" });
    const blobUrl = URL.createObjectURL(blob);
    blobUrlRef.current = blobUrl;

    // We don't set documentUrl to the blob URL anymore
    // Instead, we'll work directly with the bytes

    // Load the PDF using PDF.js
    const loadingTask = window.pdfjsLib.getDocument({ data: bytes });
    const pdf = await loadingTask.promise;

    setPdfDocument(pdf);
    setTotalPages(pdf.numPages);
    setCurrentPage(1); // Reset to first page

    // Extract text after loading
    await extractTextFromPDF();

    setIsLoading(false);
  } catch (error) {
    console.error("Error processing PDF file:", error);
    setError(
      `Failed to process PDF file: ${
        error instanceof Error ? error.message : String(error)
      }`
    );
    setIsLoading(false);
  }
};

const zoomIn = () => setScale((prev) => Math.min(prev + 0.2, 3.0));
const zoomOut = () => setScale((prev) => Math.max(prev - 0.2, 0.5));

const retryLoading = () => {
  if (documentUrl && !documentUrl.startsWith("blob:")) {
    fetchAndLoadPDF(documentUrl);
  } else if (pdfBytes) {
    // If we have the PDF bytes, try loading again
    const loadingTask = window.pdfjsLib.getDocument({ data: pdfBytes });
    setIsLoading(true);
    setError(null);

    loadingTask.promise
      .then((pdf: any) => {
        setPdfDocument(pdf);
        setTotalPages(pdf.numPages);
        return extractTextFromPDF();
      })
```

```
      .then(() => {
        setIsLoading(false);
      })
      .catch((err: any) => {
        console.error("Error reloading PDF:", err);
        setError(`Failed to reload PDF: ${err.message}`);
        setIsLoading(false);
      });
  }
};

return (
  <>
    {/* Load PDF.js from CDN */}
    <Script
      src="https://cdnjs.cloudflare.com/ajax/libs/pdf.js/3.4.120/pdf.min.js"
      onLoad={() => {
        // Set worker source after library loads
        window.pdfjsLib.GlobalWorkerOptions.workerSrc =
          "https://cdnjs.cloudflare.com/ajax/libs/pdf.js/3.4.120/pdf.worker.min.js";
        setPdfLoaded(true);
      }}
      strategy="afterInteractive"
    />

    <div className="flex flex-col h-full">
      <div className="flex items-center justify-between p-2 border-b">
        <h3 className="font-medium">Document Viewer</h3>
        <div className="flex items-center gap-2">
          {pdfDocument && (
            <Button variant="outline" size="sm" asChild>
              <Link
                href={`/comments?document=${encodeURIComponent(
                  documentName
                )}&page=${currentPage}`}
              >
                <MessageSquare className="h-4 w-4 mr-2" />
                Add Comments
              </Link>
            </Button>
          )}
          <Button
            variant="outline"
            size="sm"
            onClick={() => fileInputRef.current?.click()}
          >
            <Upload className="h-4 w-4 mr-2" />
            Upload PDF
            <input
              ref={fileInputRef}
              type="file"
              accept=".pdf"
              className="hidden"
              onChange={handleFileUpload}
              aria-label="Upload PDF document"
            />
          </Button>
        </div>
      </div>

      <div
        ref={containerRef}
        className="flex-1 flex items-center justify-center bg-muted/30 overflow-auto relative"
      >
```

```
  {isLoading ? (
    <div className="flex flex-col items-center gap-2">
      <div className="animate-spin rounded-full h-8 w-8 border-b-2 border-primary"></div>
      <p className="text-sm text-muted-foreground">
        Processing document...
      </p>
    </div>
  ) : error ? (
    <div className="flex flex-col items-center gap-4 p-6 max-w-md text-center">
      <div className="bg-red-50 text-red-700 p-4 rounded-lg">
        <p className="font-semibold mb-2">Error loading PDF</p>
        <p className="text-sm">{error}</p>
      </div>
      <Button variant="outline" onClick={retryLoading}>
        <RefreshCw className="h-4 w-4 mr-2" />
        Try Again
      </Button>
    </div>
  ) : pdfDocument ? (
    <div className="w-full h-full flex flex-col items-center">
      <div className="flex-1 w-full flex items-center justify-center overflow-auto p-2">
        <canvas
          ref={canvasRef}
          className="shadow-lg"
          aria-label={`PDF page ${currentPage} of ${totalPages}`}
        />
      </div>
    </div>
  ) : (
    <div className="flex flex-col items-center gap-2 text-muted-foreground">
      <FileText className="h-12 w-12" />
      <p>Upload a PDF document to begin</p>
    </div>
  )}
</div>

{pdfDocument && totalPages > 0 && (
  <div className="p-2 flex items-center justify-between gap-2 w-full border-t">
    <div className="flex items-center gap-2">
      <Button
        variant="outline"
        size="icon"
        onClick={zoomOut}
        disabled={scale <= 0.5}
        aria-label="Zoom out"
      >
        <ZoomOut className="h-4 w-4" />
      </Button>
      <span className="text-sm">{Math.round(scale * 100)}%</span>
      <Button
        variant="outline"
        size="icon"
        onClick={zoomIn}
        disabled={scale >= 3.0}
        aria-label="Zoom in"
      >
        <ZoomIn className="h-4 w-4" />
      </Button>
    </div>

    <div className="flex items-center gap-2">
      <Button
        variant="outline"
        size="icon"
```

```
              disabled={currentPage <= 1}
              onClick={() => setCurrentPage((p) => Math.max(1, p - 1))}
              aria-label="Previous page"
            >
              <ChevronLeft className="h-4 w-4" />
            </Button>
            <span className="text-sm">
              Page {currentPage} of {totalPages}
            </span>
            <Button
              variant="outline"
              size="icon"
              disabled={currentPage >= totalPages}
              onClick={() =>
                setCurrentPage((p) => Math.min(totalPages, p + 1))
              }
              aria-label="Next page"
            >
              <ChevronRight className="h-4 w-4" />
            </Button>
          </div>
        </div>
      )}
    </div>
  </>
 );
}


==== components/keyword-highlighter.tsx ====

"use client";

import type React from "react";
import { useState, useEffect, useCallback, useRef } from "react";
import { Search } from "lucide-react";
import { Input } from "@/components/ui/input";
import type { Keyword } from "@/services/keyword-service";

type KeywordDictionary = {
  [keyword: string]: string;
};

export function KeywordHighlighter({ text }: { text: string }) {
  const [highlightedKeywords, setHighlightedKeywords] = useState<
    React.ReactNode[]
  >([]);
  const [searchTerm, setSearchTerm] = useState("");
  const [keywords, setKeywords] = useState<KeywordDictionary>({});
  const [isLoading, setIsLoading] = useState(true);
  const fallbackUsed = useRef(false);
  const [useFallback, setUseFallback] = useState(false);

  // Function to use fallback keywords
  const useFallbackKeywords = useCallback(() => {
    if (fallbackUsed.current) return;

    console.log("Using fallback keywords");
    const fallbackData = {
      "green mountain care board":
        "An independent group created by the Vermont Legislature in 2011 to oversee the development of
health care policy in Vermont.",
      medicaid:
```

```
      "A joint federal and state program that helps with medical costs for some people with limited income and resources.",
    healthcare:
      "The organized provision of medical care to individuals or a community.",
    reform: "To make changes in something in order to improve it.",
    "payment models":
      "Methods of paying healthcare providers for services rendered.",
    "all-payer model":
      "A healthcare payment model that involves all payers (Medicare, Medicaid, commercial) using the same approach to pay providers.",
    "blueprint for health":
      "Vermont's state-led initiative that works to integrate care across the healthcare spectrum.",
    "rural healthcare":
      "Healthcare services provided in rural areas, often facing unique challenges of access and resources.",
    telehealth:
      "The delivery of health care, health education, and health information services via remote technologies.",
  };

  setKeywords(fallbackData);
  fallbackUsed.current = true;
 }, []);

 // Load keywords on component mount
 useEffect(() => {
  const fetchKeywords = async () => {
   try {
    setIsLoading(true);

    // Use a timeout to handle potential network issues
    const timeoutPromise = new Promise((_, reject) =>
     setTimeout(() => reject(new Error("Request timeout")), 5000)
    );

    // Race the fetch against the timeout
    const response = (await Promise.race([
     fetch("/api/keywords"),
     timeoutPromise,
    ])) as Response;

    // Handle non-OK responses without throwing
    if (!response.ok) {
     console.warn(`Keywords API returned status: ${response.status}`);
     setUseFallback(true);
     return;
    }

    const data = await response.json();

    const keywordDict: KeywordDictionary = {};
    data.forEach((item: Keyword) => {
     keywordDict[item.term.toLowerCase()] = item.definition;
    });

    setKeywords(keywordDict);
   } catch (error) {
    console.error("Error fetching keywords:", error);
    // Use fallback data on any error
    setUseFallback(true);
   } finally {
    setIsLoading(false);
   }
  };
```

```
  fetchKeywords();
}, []);

useEffect(() => {
  if (useFallback) {
    useFallbackKeywords();
  }
}, [useFallback, useFallbackKeywords]);

// Process text when it changes or when keywords/search term changes
useEffect(() => {
  if (!text || isLoading) {
    setHighlightedKeywords([]);
    return;
  }

  // Create a list of all keywords
  const keywordList = Object.keys(keywords);

  // Filter keywords based on search term
  const filteredKeywords = searchTerm
    ? keywordList.filter((k) =>
        k.toLowerCase().includes(searchTerm.toLowerCase())
      )
    : keywordList;

  if (filteredKeywords.length === 0) {
    setHighlightedKeywords([]);
    return;
  }

  // Find all keyword matches in the text
  const matches = new Set<string>();
  const textLower = text.toLowerCase();

  filteredKeywords.forEach((keyword) => {
    const keywordLower = keyword.toLowerCase();
    if (textLower.includes(keywordLower)) {
      matches.add(keyword);
    }
  });

  // Create formatted elements for each matched keyword
  const elements = Array.from(matches).map((keyword, index) => {
    const definition = keywords[keyword.toLowerCase()];
    return (
      <div key={index} className="mb-8">
        <div className="font-bold text-red-600 text-lg">{keyword}</div>
        <div className="mt-6 text-sm text-gray-800">{definition}</div>
      </div>
    );
  });

  setHighlightedKeywords(elements);
}, [text, searchTerm, keywords, isLoading]);

return (
  <div className="flex flex-col h-full">
    <div className="flex items-center justify-between p-2 border-b">
      <h3 className="font-medium">Keyword Highlights</h3>
      <div className="relative w-48">
        <Search className="absolute left-2 top-2.5 h-4 w-4 text-muted-foreground" />
        <Input
```

```
            type="search"
            placeholder="Filter keywords..."
            className="pl-8 h-9"
            value={searchTerm}
            onChange={(e) => setSearchTerm(e.target.value)}
            aria-label="Filter keywords"
          />
        </div>
      </div>

      <div className="flex-1 p-4 overflow-auto">
        {isLoading ? (
          <div className="flex items-center justify-center h-full">
            <div className="animate-spin rounded-full h-8 w-8 border-b-2 border-primary"></div>
          </div>
        ) : text ? (
          highlightedKeywords.length > 0 ? (
            <div className="space-y-6">{highlightedKeywords}</div>
          ) : (
            <div className="flex items-center justify-center h-full text-muted-foreground">
              <p>No matching keywords found in the document</p>
            </div>
          )
        ) : (
          <div className="flex items-center justify-center h-full text-muted-foreground">
            <p>No document text to analyze</p>
          </div>
        )}
      </div>
    </div>
  );
}
```

==== components/resizable-grid.tsx ====

```
"use client";

import type React from "react";

import { useState, useRef, useEffect } from "react";

export function ResizableGrid({ children }: { children: React.ReactNode[] }) {
  const [gridLayout, setGridLayout] = useState({
    topLeftHeight: 50, // percentage
    topRightHeight: 50, // percentage
    leftWidth: 50, // percentage
  });

  const containerRef = useRef<HTMLDivElement>(null);
  const isDraggingHorizontal = useRef(false);
  const isDraggingVertical = useRef(false);
  const [isMobile, setIsMobile] = useState(false);

  // Check if we're on a mobile device
  useEffect(() => {
    const checkMobile = () => {
      setIsMobile(window.innerWidth < 768);
    };
```

```
  checkMobile();
  window.addEventListener("resize", checkMobile);

  return () => window.removeEventListener("resize", checkMobile);
}, []);

const handleHorizontalResize = (e: MouseEvent | TouchEvent) => {
  if (!isDraggingHorizontal.current || !containerRef.current) return;

  const containerWidth = containerRef.current.clientWidth;
  let clientX: number;

  if ("touches" in e) {
    clientX = e.touches[0].clientX;
  } else {
    clientX = e.clientX;
  }

  const newLeftWidth = (clientX / containerWidth) * 100;

  // Limit the minimum size
  if (newLeftWidth > 20 && newLeftWidth < 80) {
    setGridLayout((prev) => ({
      ...prev,
      leftWidth: newLeftWidth,
    }));
  }
};

const handleVerticalResize = (e: MouseEvent | TouchEvent) => {
  if (!isDraggingVertical.current || !containerRef.current) return;

  const containerHeight = containerRef.current.clientHeight;
  let clientY: number;

  if ("touches" in e) {
    clientY = e.touches[0].clientY;
  } else {
    clientY = e.clientY;
  }

  const newTopHeight = (clientY / containerHeight) * 100;

  // Limit the minimum size
  if (newTopHeight > 20 && newTopHeight < 80) {
    setGridLayout((prev) => ({
      ...prev,
      topLeftHeight: newTopHeight,
      topRightHeight: newTopHeight,
    }));
  }
};

useEffect(() => {
  const handleMouseMove = (e: MouseEvent) => {
    handleHorizontalResize(e);
    handleVerticalResize(e);
  };

  const handleTouchMove = (e: TouchEvent) => {
    handleHorizontalResize(e);
    handleVerticalResize(e);
  };
```

```
  const handleEnd = () => {
    isDraggingHorizontal.current = false;
    isDraggingVertical.current = false;
  };

  document.addEventListener("mousemove", handleMouseMove);
  document.addEventListener("mouseup", handleEnd);
  document.addEventListener("touchmove", handleTouchMove);
  document.addEventListener("touchend", handleEnd);

  return () => {
    document.removeEventListener("mousemove", handleMouseMove);
    document.removeEventListener("mouseup", handleEnd);
    document.removeEventListener("touchmove", handleTouchMove);
    document.removeEventListener("touchend", handleEnd);
  };
}, []);

// For mobile devices, stack the panels vertically
if (isMobile) {
  return (
    <div ref={containerRef} className="w-full h-full flex flex-col gap-4">
      {children.map((child, index) => (
        <div
          key={index}
          className="flex-1 min-h-[300px] overflow-hidden rounded-lg border bg-background shadow"
        >
          {child}
        </div>
      ))}
    </div>
  );
}

return (
  <div ref={containerRef} className="w-full h-full relative">
    <div
      className="absolute top-0 left-0 overflow-hidden rounded-lg border bg-background shadow"
      style={{
        width: `${gridLayout.leftWidth}%`,
        height: `${gridLayout.topLeftHeight}%`,
        padding: "1px",
      }}
      aria-label="Document Viewer Panel"
    >
      {children[0]}
    </div>

    <div
      className="absolute top-0 overflow-hidden rounded-lg border bg-background shadow"
      style={{
        left: `${gridLayout.leftWidth}%`,
        width: `${100 - gridLayout.leftWidth}%`,
        height: `${gridLayout.topRightHeight}%`,
        padding: "1px",
      }}
      aria-label="Keyword Highlights Panel"
    >
      {children[1]}
    </div>

    <div
      className="absolute left-0 overflow-hidden rounded-lg border bg-background shadow"
```

```
        style={{
          top: `${gridLayout.topLeftHeight}%`,
          width: `${gridLayout.leftWidth}%`,
          height: `${100 - gridLayout.topLeftHeight}%`,
          padding: "1px",
        }}
        aria-label="News Headlines Panel"
      >
        {children[2]}
      </div>

      <div
        className="absolute overflow-hidden rounded-lg border bg-background shadow"
        style={{
          top: `${gridLayout.topRightHeight}%`,
          left: `${gridLayout.leftWidth}%`,
          width: `${100 - gridLayout.leftWidth}%`,
          height: `${100 - gridLayout.topRightHeight}%`,
          padding: "1px",
        }}
        aria-label="News Article Panel"
      >
        {children[3]}
      </div>

      {/* Horizontal resize handle */}
      <div
        className="absolute top-0 bottom-0 w-4 cursor-col-resize bg-transparent hover:bg-primary/10
z-10 transition-colors duration-200"
        style={{ left: `calc(${gridLayout.leftWidth}% - 8px)` }}
        onMouseDown={() => {
          isDraggingHorizontal.current = true;
        }}
        onTouchStart={() => {
          isDraggingHorizontal.current = true;
        }}
        role="separator"
        aria-orientation="vertical"
        aria-label="Resize panels horizontally"
        tabIndex={0}
        onKeyDown={(e) => {
          if (e.key === "ArrowLeft") {
            setGridLayout((prev) => ({
              ...prev,
              leftWidth: Math.max(20, prev.leftWidth - 1),
            }));
          } else if (e.key === "ArrowRight") {
            setGridLayout((prev) => ({
              ...prev,
              leftWidth: Math.min(80, prev.leftWidth + 1),
            }));
          }
        }}
      />

      {/* Vertical resize handle */}
      <div
        className="absolute left-0 right-0 h-4 cursor-row-resize bg-transparent hover:bg-primary/10 z-10
transition-colors duration-200"
        style={{ top: `calc(${gridLayout.topLeftHeight}% - 8px)` }}
        onMouseDown={() => {
          isDraggingVertical.current = true;
        }}
        onTouchStart={() => {
```

```
          isDraggingVertical.current = true;
        }}
        role="separator"
        aria-orientation="horizontal"
        aria-label="Resize panels vertically"
        tabIndex={0}
        onKeyDown={(e) => {
          if (e.key === "ArrowUp") {
            setGridLayout((prev) => ({
              ...prev,
              topLeftHeight: Math.max(20, prev.topLeftHeight - 1),
              topRightHeight: Math.max(20, prev.topRightHeight - 1),
            }));
          } else if (e.key === "ArrowDown") {
            setGridLayout((prev) => ({
              ...prev,
              topLeftHeight: Math.min(80, prev.topLeftHeight + 1),
              topRightHeight: Math.min(80, prev.topRightHeight + 1),
            }));
          }
        }}
      />

      {/* Intersection handle */}
      <div
        className="absolute w-8 h-8 cursor-move bg-transparent hover:bg-primary/20 z-20 rounded-full
transition-colors duration-200"
        style={{
          left: `calc(${gridLayout.leftWidth}% - 16px)`,
          top: `calc(${gridLayout.topLeftHeight}% - 16px)`,
        }}
        onMouseDown={() => {
          isDraggingHorizontal.current = true;
          isDraggingVertical.current = true;
        }}
        onTouchStart={() => {
          isDraggingHorizontal.current = true;
          isDraggingVertical.current = true;
        }}
        aria-label="Resize panels in both directions"
        tabIndex={0}
      />
    </div>
  );
}


==== lib/prisma.ts ====

import { PrismaClient } from "@prisma/client";

// PrismaClient is attached to the `global` object in development to prevent
// exhausting your database connection limit.
const globalForPrisma = global as unknown as { prisma: PrismaClient };

export const prisma = globalForPrisma.prisma || new PrismaClient();

if (process.env.NODE_ENV !== "production") globalForPrisma.prisma = prisma;

export default prisma;


==== lib/utils.ts ====
```

```
import { clsx, type ClassValue } from "clsx"
import { twMerge } from "tailwind-merge"

export function cn(...inputs: ClassValue[]) {
  return twMerge(clsx(inputs))
}


==== prisma/migrations/20250328040627_init/migration.sql ====

-- CreateTable
CREATE TABLE "Keyword" (
    "id" TEXT NOT NULL,
    "term" TEXT NOT NULL,
    "definition" TEXT NOT NULL,
    "createdAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updatedAt" TIMESTAMP(3) NOT NULL,

    CONSTRAINT "Keyword_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "Comment" (
    "id" TEXT NOT NULL,
    "content" TEXT NOT NULL,
    "documentName" TEXT NOT NULL,
    "documentPage" INTEGER NOT NULL,
    "author" TEXT NOT NULL,
    "createdAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updatedAt" TIMESTAMP(3) NOT NULL,

    CONSTRAINT "Comment_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "BlogPost" (
    "id" TEXT NOT NULL,
    "title" TEXT NOT NULL,
    "slug" TEXT NOT NULL,
    "content" TEXT NOT NULL,
    "excerpt" TEXT NOT NULL,
    "author" TEXT NOT NULL,
    "published" BOOLEAN NOT NULL DEFAULT false,
    "createdAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updatedAt" TIMESTAMP(3) NOT NULL,

    CONSTRAINT "BlogPost_pkey" PRIMARY KEY ("id")
);

-- CreateIndex
CREATE UNIQUE INDEX "Keyword_term_key" ON "Keyword"("term");

-- CreateIndex
CREATE INDEX "Comment_documentName_documentPage_idx" ON "Comment"("documentName",
"documentPage");

-- CreateIndex
CREATE UNIQUE INDEX "BlogPost_slug_key" ON "BlogPost"("slug");
```

Table of Contents

1. Project Purpose

The Vermont Healthcare Reform Portal is a web application designed to provide information, resources, and tools related to healthcare reform initiatives in Vermont. The application serves as a knowledge base and collaboration platform for stakeholders interested in Vermont's healthcare policies, reforms, and terminology.

2. Technology Stack

- **Frontend**: Next.js (App Router), React, TypeScript, Tailwind CSS, shadcn/ui components
- **Backend**: Next.js API Routes (serverless functions)
- **Database**: PostgreSQL (migrated from localStorage)
- **ORM**: Prisma
- **Authentication**: Not yet implemented (could be added with NextAuth.js)

3. Project Structure

```plaintext
vt-healthcare-reform/
├── app/                    # Next.js App Router directory
│   ├── admin/              # Admin section
│   │   ├── keywords/       # Keyword management
│   │   │   └── page.tsx    # Keywords admin page
│   │   └── page.tsx        # Main admin dashboard
│   ├── analytics/          # Analytics section
│   │   └── page.tsx        # Analytics dashboard
│   ├── api/                # API routes
│   │   ├── blog/           # Blog API endpoints
│   │   │   ├── [id]/       # Blog post by ID
│   │   │   │   └── route.ts  # GET, PUT, DELETE operations
│   │   │   └── route.ts    # GET, POST operations
│   │   ├── comments/       # Comments API endpoints
│   │   │   ├── [id]/       # Comment by ID
│   │   │   │   └── route.ts  # GET, DELETE operations
│   │   │   └── route.ts    # GET, POST operations
│   │   ├── export/         # Export functionality
│   │   │   └── route.ts    # Export data to Excel
│   │   ├── import/         # Import functionality
│   │   │   └── route.ts    # Import data from Excel
│   │   └── keywords/       # Keywords API endpoints
│   │       ├── [id]/       # Keyword by ID
│   │       │   └── route.ts  # DELETE operation
│   │       └── route.ts    # GET, POST operations
│   ├── blog/               # Blog section
│   │   ├── [slug]/         # Blog post by slug
│   │   │   └── page.tsx    # Blog post detail page
│   │   └── page.tsx        # Blog listing page
│   ├── comments/           # Comments section
│   │   └── page.tsx        # Comments page
```

```
│   ├── glossary/              # Glossary section
│   │   └── page.tsx           # Glossary page
│   ├── settings/              # Settings section
│   │   └── page.tsx           # Settings page
│   ├── globals.css            # Global CSS
│   ├── layout.tsx             # Root layout
│   └── page.tsx               # Home page
├── components/                # React components
│   ├── ui/                    # UI components (shadcn/ui)
│   │   ├── button.tsx         # Button component
│   │   ├── card.tsx           # Card component
│   │   └── ...                # Other UI components
│   ├── excel-export.tsx       # Excel export component
│   ├── excel-import.tsx       # Excel import component
│   ├── navbar.tsx             # Navigation bar
│   └── sidebar.tsx            # Sidebar navigation
├── lib/                       # Utility libraries
│   ├── excel-utils.ts         # Excel utilities
│   ├── prisma.ts              # Prisma client
│   └── utils.ts               # General utilities
├── prisma/                    # Prisma ORM
│   ├── migrations/            # Database migrations
│   └── schema.prisma          # Prisma schema
├── public/                    # Static assets
├── services/                  # Service layer
│   ├── blog-service.ts        # Blog service
│   ├── comment-service.ts     # Comment service
│   └── keyword-service.ts     # Keyword service
├── .env                       # Environment variables
├── .env.local                 # Local environment variables
├── next.config.js             # Next.js configuration
├── package.json               # Project dependencies
├── tailwind.config.js         # Tailwind CSS configuration
└── tsconfig.json              # TypeScript configuration
```

4. Key Components

4.1 Pages

The application is organized into several key sections:

- **Blog**: Articles about healthcare reform initiatives in Vermont
- **Glossary**: A dictionary of healthcare terminology
- **Comments**: A system for users to leave comments on documents
- **Admin**: Administrative interface for managing content
- **Analytics**: Data visualizations and metrics (placeholder)
- **Settings**: Application configuration (placeholder)

4.2 API Routes

The application uses Next.js API routes to handle data operations:

- **/api/keywords**: Manage healthcare terminology
- **/api/blog**: Manage blog posts
- **/api/comments**: Manage user comments
- **/api/import**: Import data from Excel files
- **/api/export**: Export data to Excel files

4.3 Services

The service layer abstracts database operations:

- **KeywordService**: Manages healthcare terminology
- **BlogService**: Manages blog posts
- **CommentService**: Manages user comments

4.4 Components

The UI is built with reusable components:

- **Sidebar**: Navigation sidebar
- **Navbar**: Top navigation bar
- **ExcelImport/ExcelExport**: Components for importing/exporting data
- **shadcn/ui components**: Button, Card, Input, etc.

5. Data Models

5.1 Keyword

The Keyword model represents healthcare terminology. Each keyword has a term and definition. The term field is unique to prevent duplicates.

Key fields:

- id: UUID primary key
- term: Unique string
- definition: Text field
- createdAt: Timestamp
- updatedAt: Timestamp

5.2 Comment

The Comment model represents user comments on documents. Comments are associated with a specific document and page. The database includes an index to optimize queries that filter by document name and page.

Key fields:

- id: UUID primary key
- content: Text field
- documentName: String
- documentPage: Integer
- author: String
- createdAt: Timestamp
- updatedAt: Timestamp

5.3 BlogPost

The BlogPost model represents articles about healthcare reform. Each post has a unique slug for SEO-friendly URLs. The published field allows for draft posts.

Key fields:

- id: UUID primary key
- title: String
- slug: Unique string
- content: Text field (Markdown)
- excerpt: Text field
- author: String
- published: Boolean
- createdAt: Timestamp

- updatedAt: Timestamp

6. Feature Highlights

6.1 Glossary System

The glossary system provides a searchable dictionary of healthcare terminology. Users can:

- Browse terms alphabetically
- Search for specific terms
- View detailed definitions

The admin interface allows administrators to:

- Add new terms
- Edit existing terms
- Delete terms
- Import/export terms via Excel

6.2 Blog System

The blog system provides articles about healthcare reform. Users can:

- Browse all published articles
- Search for specific topics
- Read full articles with markdown formatting

The admin interface (partially implemented) will allow administrators to:

- Create new articles
- Edit existing articles
- Publish/unpublish articles
- Delete articles

6.3 Comment System

The comment system allows users to provide feedback on documents. Users can:

- View comments for specific documents
- Add new comments
- Filter comments by document and page

6.4 Excel Import/Export

The application includes functionality to import and export data using Excel files:

- Import keywords from Excel files
- Export keywords to Excel files
- Handles both .xlsx and .csv formats

7. Database Integration with Prisma

The application has been migrated from localStorage to PostgreSQL using Prisma ORM. This provides:

1. **Data Persistence**: Data is stored in a robust database rather than browser storage
2. **Scalability**: Can handle larger datasets and concurrent users
3. **Data Integrity**: Enforces constraints like unique terms and relationships

4. **Query Efficiency**: Optimized database queries with indexes
5. **Type Safety**: Prisma generates TypeScript types for the data models

7.1 Prisma Client Usage

The application uses a singleton Prisma client to prevent connection exhaustion. This pattern ensures that only one Prisma Client instance is created in development, preventing database connection limits from being reached.

7.2 Data Seeding

Each service includes logic to seed initial data when the database is empty. This ensures that new installations have sample data to work with. The seeding process checks if the database is empty before adding initial records, preventing duplicate data.

8. Setup and Deployment

8.1 Local Development

1. **Clone the repository**:

```plaintext
git clone <repository-url>
cd vt-healthcare-reform
```

2. **Install dependencies**:

```plaintext
npm install
```

3. **Set up environment variables**:
Create a `.env` file with:

```plaintext
DATABASE_URL="postgresql://username:password@localhost:5432/vt_healthcare?schema=public"
```

4. **Generate Prisma client**:

```plaintext
npx prisma generate
```

5. **Run database migrations**:

```plaintext
npx prisma migrate dev --name init
```

6. **Start the development server**:

```plaintext
npm run dev
```

8.2 Database Management

- **View database**: `npx prisma studio`
- **Create migration**: `npx prisma migrate dev --name <migration-name>`
- **Apply migrations**: `npx prisma migrate deploy`

8.3 Production Deployment

For production deployment, you can use Vercel:

1. **Connect to GitHub repository**
2. **Set environment variables** in Vercel dashboard
3. **Deploy**

For the PostgreSQL database, you can use services like:

- Neon (recommended)
- Supabase
- Railway
- Heroku Postgres

9. Future Enhancements

1. **Authentication**: Add user authentication with NextAuth.js
2. **Document Repository**: Add a system for uploading and viewing PDF documents
3. **Advanced Analytics**: Implement real analytics with charts and visualizations
4. **User Roles**: Add role-based access control (admin, editor, viewer)
5. **Notifications**: Add a notification system for new comments or content
6. **Full-text Search**: Implement advanced search capabilities across all content

10. Conclusion

The Vermont Healthcare Reform Portal is a comprehensive web application for managing and sharing information about healthcare reform initiatives. The migration from localStorage to PostgreSQL with Prisma provides a robust foundation for future growth and feature development.

The application's modular architecture makes it easy to add new features and maintain existing ones. The service layer abstracts database operations, making it possible to change the underlying database technology without affecting the rest of the application.

By using Next.js App Router, the application benefits from server components, automatic code splitting, and optimized rendering. The combination of Prisma and PostgreSQL provides a type-safe, scalable database solution that can handle the application's data needs as it grows.

The Excel import/export functionality makes it easy to work with data from other systems, and the comment system enables collaboration among users. The blog and glossary systems provide valuable information about healthcare reform initiatives in Vermont.

Overall, the Vermont Healthcare Reform Portal is a well-structured, modern web application that provides a solid foundation for future development and expansion.