**Proposal for Academic Document Review and Publishing Application**

**Project Overview**

The proposed system is a **web-based platform** designed to facilitate the review, annotation, and evaluation of academic documents. It provides tools for annotating documents, tagging sections for reference, generating AI-based inferences, checking for duplicate content, and assessing the overall quality and originality of research papers. The platform is intended to streamline the academic review process, making it easier to ensure research quality before publication.

**Target Users**

The application is designed for:

* **Students**: To annotate and review research papers for study or coursework.
* **Researchers**: To collaborate on document reviews, add insights, and prepare papers for submission.
* **Professors**: To review and evaluate student submissions or research papers.
* **Principal Investigators (PIs)**: To oversee research quality and manage document reviews.
* **Publishers**: To review document quality, check for originality, and evaluate readiness for publishing.

**System Features and Functionalities**

**1. Document Annotation**

* **Description**: This feature allows users to directly annotate PDF documents with highlights, notes, and comments. It provides tools to mark up documents and share insights.
* **Features**:
  + **Highlight Text**: Select specific text and highlight it using different colors for categorization.
  + **Add Comments**: Attach comments or notes to specific sections of the document for detailed feedback or observations.
  + **Markup Tools**: Use drawing tools to underline, circle, or box important sections or images within the document.
  + **Real-Time Collaboration**: Multiple users can annotate simultaneously, enabling collaborative reviews with instant updates.

**2. Tagging and Summarization**

* **Description**: Allows users to tag highlighted sections with keywords and add notes that can be compiled into a summary for future reference.
* **Features**:
  + **Add Tags to Highlights**: Assign tags to highlighted sections, enabling quick reference and easy navigation to key areas.
  + **Reference Notes**: Attach detailed notes or summaries to highlights, which can be used for later study or review.
  + **Download Summaries**: Generate a downloadable summary (PDF or Word) of all highlighted sections and notes, allowing users to compile insights into a single document.

**3. AI-Powered Inference and Q&A Generation**

* **Description**: AI tools analyze highlighted sections to generate insights, suggest inferences, and frame questions and answers based on the content. This feature aids users in understanding the content and provides context.
* **Features**:
  + **Inference Suggestions**: AI analyzes highlighted sections and suggests possible inferences or interpretations, helping users grasp the content more deeply.
  + **Question and Answer Generation**: The AI generates questions and potential answers based on the highlighted sections, which can help users engage critically with the material.
  + **Content Classification**: The system classifies highlighted sections into categories such as “Introduction,” “Methods,” “Results,” or “Discussion” based on context.

**4. Quality Assessment and Plagiarism Detection**

* **Description**: The application provides a net score to evaluate the quality of the research paper based on language, content coherence, and originality. It also checks for duplicate content and plagiarism.
* **Features**:
  + **Language Quality Score**: Evaluates grammar, vocabulary, and readability to provide a language proficiency score.
  + **Content Quality Score**: Analyzes the coherence, logical flow, and relevance of the content.
  + **Plagiarism Detection**: Checks the document for duplicate content and provides a plagiarism report, indicating any sections that may be copied or lack originality.
  + **Overall Paper Score**: Combines language, content, and originality scores to provide a comprehensive assessment of the paper’s quality.

**5. User-Specific Dashboards**

* **Description**: Each user type has access to a customized dashboard that provides access to tools, documents, and insights based on their role.
* **Features**:
  + **Student Dashboard**: Displays recently reviewed documents, highlights and notes, and summary download options.
  + **Researcher Dashboard**: Shows active research documents, collaboration tools, and AI-generated inferences and summaries.
  + **Professor Dashboard**: Allows access to student submissions, annotated papers, and quality assessment reports.
  + **PI Dashboard**: Provides an overview of research quality, document statistics, and overall scoring.
  + **Publisher Dashboard**: Displays plagiarism reports, quality assessments, and downloadable summaries for final review.

**6. Reporting and Export**

* **Description**: Users can generate reports that compile annotations, tags, AI inferences, and quality assessments into a document.
* **Features**:
  + **Report Generation**: Generates a summary report of annotations, tags, AI-generated content, and scoring.
  + **Export Options**: Export annotated documents and reports in multiple formats (PDF, Word) for easy sharing or offline use.

**System Design and Architecture**

**1. System Architecture**

* **Frontend**: Built with **React.js** or **Vue.js** for an interactive, responsive user interface.
* **Backend**: Implemented using **Node.js** or **Django** with a RESTful API for data exchange between the frontend and backend.
* **Database**: Uses a **relational database** (such as PostgreSQL or MySQL) to store user data, annotations, tags, scores, and reports.
* **AI/ML Components**: Leverages **NLP models** (such as GPT-4) to generate inferences, questions, and quality assessments. AI models are deployed on cloud platforms like **AWS SageMaker** or **Google AI Platform**.
* **File Storage**: PDF documents and annotated files are stored in **cloud storage** services like AWS S3 or Google Cloud Storage.

**2. User Roles and Permissions**

* **Students**: Can annotate, tag, generate summaries, and view AI inferences.
* **Researchers**: Access to full annotation capabilities, AI-powered insights, and plagiarism reports.
* **Professors**: Can review student papers, access quality assessment tools, and generate reports.
* **PIs**: Have access to all features, including viewing overall quality scores and plagiarism checks.
* **Publishers**: Can view quality assessments, plagiarism reports, and summaries, but do not engage in annotation.

**3. Workflow and User Journey**

* **Upload Document**: Users can upload documents in PDF format.
* **Annotate and Tag**: Highlight and annotate sections, add tags, and attach notes.
* **AI Inference and Q&A**: Leverage AI to analyze highlighted sections and generate insights.
* **Quality Assessment**: Review net score and plagiarism check results.
* **Dashboard Overview**: Access user-specific dashboards with relevant document summaries, scores, and reports.

**4. Security and Compliance**

* **User Authentication**: Secure role-based access control to manage permissions.
* **Data Encryption**: Encrypts all data during storage and transfer to protect user privacy.
* **Audit Logging**: Tracks all actions for accountability and compliance.
* **Compliance**: Ensures adherence to **GDPR** and **HIPAA** standards for data privacy.

**Technology Stack**

* **Frontend**: React.js, Vue.js, Tailwind CSS for design
* **Backend**: Node.js or Django with REST API
* **Database**: PostgreSQL or MySQL
* **AI/ML**: GPT-4 or custom NLP models deployed on AWS SageMaker or Google AI Platform
* **File Storage**: AWS S3 or Google Cloud Storage
* **Authentication**: Auth0 or custom OAuth for secure login

**Conclusion**

This platform provides a comprehensive toolset for academic document review and analysis. By combining annotation, tagging, AI-powered insights, and quality assessment, the system supports students, researchers, professors, PIs, and publishers in producing high-quality, original research ready for publication.

**System Design Document**

**1. System Architecture Overview**

* **Frontend**: Built using **React.js** for a responsive, interactive user interface.
* **Backend**: **Node.js** with **Express** to manage API requests, connect to the database, and perform server-side processing.
* **Database**: **PostgreSQL** to store user data, document metadata, annotations, and tags.
* **AI/ML Components**: **NLP models** like **GPT-4** deployed on **AWS SageMaker** for generating inferences, questions, and content quality scoring.
* **File Storage**: **AWS S3** for document storage, allowing for secure and scalable storage of PDF files.
* **Authentication**: **Auth0** for secure, role-based access control.

**2. Key Components and Technologies**

* **PDF Annotation Library**: **PDF.js** for rendering PDF files on the web. Additional libraries like **React-PDF** and **Fabric.js** for annotation features.
* **Drag and Drop**: **React DnD** for implementing drag-and-drop functionality to select and tag content.
* **Text Analysis**: **OpenAI GPT-4 API** for generating questions, inferences, and context from highlighted text.
* **Plagiarism and Duplicate Check**: **Turnitin API** or **Plagscan API** for duplicate content and plagiarism checking.
* **Frontend Styling**: **Tailwind CSS** for rapid, responsive styling.
* **RESTful API**: Using **Express.js** for building the backend API endpoints that connect the frontend and database.

**Workflow Breakdown**

**1. Document Upload and Rendering**

* **Workflow**:
  + **Step 1**: Users upload a PDF document via the frontend.
  + **Step 2**: The PDF is stored in **AWS S3** and a reference to the document is saved in **PostgreSQL**.
  + **Step 3**: The document is retrieved from S3 and rendered in the browser using **PDF.js**.
* **Components**:
  + **Frontend**: React components to handle file uploads and PDF rendering with PDF.js.
  + **Backend**: API endpoint (/uploadDocument) that receives the document, stores it in AWS S3, and saves the metadata in PostgreSQL.
* **Sample Code**:

javascript

Copy code

// Frontend: File Upload

const handleFileUpload = async (file) => {

const formData = new FormData();

formData.append("file", file);

const response = await fetch("/api/uploadDocument", {

method: "POST",

body: formData,

});

const data = await response.json();

// Handle response

};

**2. Document Annotation and Tagging**

* **Workflow**:
  + **Step 1**: Users select text or sections of the PDF and choose to annotate or highlight it.
  + **Step 2**: Users can add tags or notes to the highlighted sections.
  + **Step 3**: All annotations are saved to the database and can be retrieved for future reference.
* **Components**:
  + **Frontend**: Use **React-PDF** for PDF rendering and **Fabric.js** for annotation capabilities. Store annotation details like highlight color, tags, and comments.
  + **Backend**: API endpoint (/saveAnnotation) that receives the annotation details and saves them to the PostgreSQL database.
* **Sample Code**:

javascript

Copy code

// Frontend: Annotation

const saveAnnotation = async (annotation) => {

const response = await fetch("/api/saveAnnotation", {

method: "POST",

headers: { "Content-Type": "application/json" },

body: JSON.stringify(annotation),

});

const data = await response.json();

// Handle response

};

**3. Drag-and-Drop Content Selection for Tagging**

* **Workflow**:
  + **Step 1**: Users highlight a section, then drag it to the “Tag” panel to add specific tags.
  + **Step 2**: Tags are stored in the database and linked to the specific highlight.
* **Components**:
  + **Frontend**: Use **React DnD** for drag-and-drop tagging functionality.
  + **Backend**: API endpoint (/saveTags) to save the associated tags for each highlight.
* **Sample Code**:

javascript

Copy code

// Frontend: Tagging with Drag and Drop

const [{ isDragging }, dragRef] = useDrag({

type: "TEXT",

item: { id: highlightId },

collect: (monitor) => ({

isDragging: monitor.isDragging(),

}),

});

return (

<div ref={dragRef} style={{ opacity: isDragging ? 0.5 : 1 }}>

<button onClick={() => saveTags(highlightId, tags)}>Tag</button>

</div>

);

**4. AI-Generated Inferences and Q&A for Highlights**

* **Workflow**:
  + **Step 1**: After highlighting a section, the user clicks on “Generate Inference” or “Generate Q&A”.
  + **Step 2**: The highlighted text is sent to the backend, where the **GPT-4 API** processes it and returns suggestions.
* **Components**:
  + **Frontend**: Button to trigger AI-based inference generation.
  + **Backend**: API endpoint (/generateInference) that sends the highlighted text to GPT-4 and returns inferences or questions/answers.
* **Sample Code**:

javascript

Copy code

// Backend: Generate Inference with GPT-4

app.post("/api/generateInference", async (req, res) => {

const { text } = req.body;

const gptResponse = await openai.Completion.create({

model: "gpt-4",

prompt: `Analyze and provide insights or questions for the following text: ${text}`,

max\_tokens: 150,

});

res.json({ inference: gptResponse.choices[0].text });

});

**5. Quality Scoring and Plagiarism Check**

* **Workflow**:
  + **Step 1**: After the document is annotated, users click “Analyze Quality” to generate a quality score based on language and content.
  + **Step 2**: The document text is sent to a plagiarism detection API like **Turnitin** or **Plagscan**.
  + **Step 3**: Results are displayed on the user’s dashboard with a score and any detected duplicate content.
* **Components**:
  + **Frontend**: Button to trigger quality analysis and plagiarism check, with results displayed in a scoring panel.
  + **Backend**: API endpoint (/analyzeQuality) to process language quality via **GPT-4** and handle plagiarism check through **Turnitin API**.
* **Sample Code**:

javascript

Copy code

// Backend: Quality Analysis

app.post("/api/analyzeQuality", async (req, res) => {

const { text } = req.body;

const scoreResponse = await openai.Completion.create({

model: "gpt-4",

prompt: `Evaluate the following text for language quality: ${text}`,

max\_tokens: 50,

});

const plagiarismResponse = await turnitinApi.checkPlagiarism(text);

res.json({

qualityScore: scoreResponse.choices[0].text,

plagiarismReport: plagiarismResponse,

});

});

**6. User Dashboards**

* **Workflow**:
  + **Step 1**: Upon login, users are directed to a role-specific dashboard.
  + **Step 2**: The dashboard displays relevant documents, summaries, and analysis results.
* **Components**:
  + **Frontend**: Role-based React components that dynamically render the dashboard based on user type.
  + **Backend**: Authentication middleware ensures users only access their assigned dashboards and data.
* **Sample Code**:

javascript

Copy code

// Frontend: Conditional Rendering for Role-Based Dashboard

const Dashboard = ({ role }) => {

switch (role) {

case "Student":

return <StudentDashboard />;

case "Researcher":

return <ResearcherDashboard />;

case "Professor":

return <ProfessorDashboard />;

case "PI":

return <PIDashboard />;

case "Publisher":

return <PublisherDashboard />;

default:

return <GeneralDashboard />;

}

};

**7. Security and Compliance**

* **Authentication**: Use **Auth0** for secure login and role-based access control.
* **Data Encryption**: Encrypt all data transfers and store sensitive data securely.
* **Audit Logging**: Track all user actions to ensure compliance and data integrity.

**Conclusion**

This detailed system design document provides a comprehensive roadmap for building the Academic Document Review and Publishing Application. With an emphasis on features like annotation, tagging, AI-driven insights, plagiarism checking, and role-specific dashboards, this platform will offer a robust and efficient solution for academic document review and publishing.