.gitignore

```
node_modules
```

nodemon.json

```
{
  "watch": ["src"],
  "ext": "ts",
  "exec": "ts-node src/index.ts"
}
```

package.json

```
{
  "name": "reg_lesson",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "dev": "nodemon"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "packageManager":
        "yarn@1.22.22+sha512.a6b2f7906b721bba3d67d4aff083df04dad64c3997078
  "devDependencies": {
    "@types/express": "^5.0.0",
    "@types/multer": "^1.4.12",
    "@types/node": "^22.13.0",
    "nodemon": "^3.1.9",
    "ts-node": "^10.9.2",
    "typescript": "^5.7.3"
  },
  "dependencies": {
    "@langchain/community": "^0.3.28",
    "@langchain/core": "^0.3.37",
    "@langchain/ollama": "^0.1.5",
    "cors": "^2.8.5",
    "express": "^4.21.2",
    "langchain": "^0.3.15",
    "multer": "^1.4.5-lts.1",
    "pdf-parse": "^1.1.1"
  }
}
```

src/index.ts

```
import express, { json, Response, Request } from "express";
import cors from "cors";
import multer from "multer";
import * as fs from "fs";
import { generateOutput, generatePrompt, loadAndSplit, search }
        from "./rag";

const app = express();
app.use(json());
app.use(cors());

if (!fs.existsSync("uploads")) {
  fs.mkdirSync("uploads");
}

const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, "uploads");
  },
  filename: (_, file, cb) => {
    cb(null, file.originalname);
  },
});

const upload = multer({ storage });

interface Input extends Request {
  file: Express.Multer.File;
  body: {
    question: string;
  };
}

app.post(
  "/upload",
  upload.single("file"),
  async ({ file, body: { question } }: Input, res: Response) => {
    try {
      if (!file) {
        res.status(400).send("No file uploaded");
        return;
      }
      const filePath = `./uploads/${file.filename}`;
      let splits;
      if (!fs.existsSync(filePath)) {
        res.status(404).send("File not found");
        return;
      } else {
        // function for split file into chunks
        splits = await loadAndSplit(filePath);
        // upload file to vector database
      }

      // search
      const searches = await search(splits, question, filePath);
      // prompt
      const prompt = await generatePrompt(searches, question);
      // result
      const answer = await generateOutput(prompt);
      console.log("Answer generated", answer);
      res.status(200).send({ message: answer.content });
    } catch (error) {
      res.status(500).send(error.message);
    }
  }
);

app.listen(3001, () => {
  console.log("Server is running on port 3001");
});
```

src/rag.ts

```
import { PDFLoader } from "@langchain/community/document_loaders/
        fs/pdf";
import { ChatOllama, OllamaEmbeddings } from "@langchain/ollama";
import { RecursiveCharacterTextSplitter } from "@langchain/
        textsplitters";
import { MemoryVectorStore } from "langchain/vectorstores/
        memory";
```

```
import { PromptTemplate } from "@langchain/core/prompts";

export const loadAndSplit = async (path: string) => {
  const loader = new PDFLoader(path);
  const document = await loader.load();

  // Split the document into chunks
  const textSplitter = new RecursiveCharacterTextSplitter({
    chunkSize: 500,
    chunkOverlap: 0,
  });
  console.log("Splitting document into chunks");
  return textSplitter.splitDocuments(document);
};

const vectorSearchCache = {};

export const search = async (splits: any, query: string,
        filePath: string) => {
  if (!vectorSearchCache[filePath]) {
    const embeddings = new OllamaEmbeddings();
    vectorSearchCache[filePath] = await
        MemoryVectorStore.fromDocuments(
      splits,
      embeddings
    );
    console.log("Vectors created");
  } else {
    console.log("Using cached vectors");
  }

  const vectorStore = vectorSearchCache[filePath];
  console.log("Searching for similar vectors");
  return vectorStore.similaritySearch(query);
};

export const generatePrompt = (search, question) => {
  let context = "";
  search.forEach((result) => {
    context += result.pageContent + " ";
  });

  const prompt = PromptTemplate.fromTemplate(`
    Answer the question based only on the following context:
```

```
    {context}
    ---
    Answer the question based on the above context: {question}
  `);
  console.log("Prompt generated", prompt);
  return prompt.format({ context, question });
};

export const generateOutput = (prompt) => {
  const ollamaLlm = new ChatOllama({
    baseUrl: "http://localhost:11434/",
    model: "llama3.2",
  });
  console.log("Generating output");
  return ollamaLlm.invoke(prompt);
};
```

tsconfig.json

```
{
  "include": ["src/**/*.ts"],
  "compilerOptions": {
    "module": "commonjs",
    "declaration": true,
    "removeComments": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "esModuleInterop": true,
    "allowSyntheticDefaultImports": true,
    "target": "es2017",
    "sourceMap": true,
    "outDir": "./dist",
    "baseUrl": ".",
    "incremental": true,
    "skipLibCheck": true,
    "strictNullChecks": false,
    "noImplicitAny": false,
    "strictBindCallApply": false,
    "forceConsistentCasingInFileNames": false,
    "noFallthroughCasesInSwitch": false
  }
}
```

uploads/Harry Potter Prisoner of Azkaban.pdf

This is a binary file of the type: PDF

uploads/hp5-harry-potter-and-the-order-of-the-phoenix.pdf

This is a binary file of the type: PDF

uploads/typeorm-io-....pdf

This is a binary file of the type: PDF