Table of Content

List of Figures

CHAPTER-1

I. Problem Background and its Context

Un-optimize space fill in our device like those document which we needed later for this we have to save our document in our device like mobile/laptop which require a space a lot in memory storage A web portal for storing the document in a cloud server ensure that the document will remain safe ,and generating the link for reference for user so they easily find their document . The Web application is easy to use and user friendly and secure

II. System Objective

The main objective of "File Store" is to facilitate a user friendly environment for all users and reduces the manual effort for reducing space . In past days User clear space manually but in further resolution of the technology we are able to resolve the Problem automatically. We develop File's Store web Application where a user upload their File's or Document in our server with Unique Code . Server Generate the Unique Link for User so ,they easily identify their document . Firstly User have to authenticate with Unique Code and then . When ever user click on the link they are able to download those document from server and it reduce the memory management Problem

III. Functionality (Core and advanced)

Interface Requirement

1. User Interface

The package must be user friendly and robust. It must prompt the user with proper message boxes to help them perform various actions and how to precede further the system must respond normally under any in out conditions and display proper message instead of turning up faults and errors.

2. Software and Hardware Requirements

In the following, we describe the software and hardware requirements for system developers and system users. During our system development, we have to design both static and dynamic website interfaces, create website functions and a database system, edit photos and pictures, and print out reports, so it has a set of software and hardware requirements.

Software Requirements

Operating System	:	Windows 7 and above
Front- End	:	HTML
	:	CSS
	:	Javascript
	:	Bootstrap
Database	:	MongoDB
Backend	:	Javascript
	:	NodeJS

Hardware Requirements

Processor	:	INTEL Pentium 4 or Higher
RAM	:	2GB or More
Hard Disk Drive	:	250GB or More
Video	:	1280X800, 256 colours
CD-ROM	:	Required
Key Board	:	Standard 101/102 or Digi Sync Family
Monitor	:	Display Panel (1280 X 800)
Display Adapter	:	Trident Super VGA
Network Adapter	:	SMC Ethernet Card Elite 16 Ultra
Mouse	:	Logitech Serial Mouse

TOOLS AND PLATFORM USED Visual Studio Code

Fig 1: VISUAL STUDIO CODE LOGO

Visual Studio Code is an amazing piece of software. To start, it is a clean, functional, and fast code editor. Not only does it have incredible built-in features (multi-edit and vim mode), but it has support for plugins, snippets, and many other things.

SOME TOOLS:

GOTO ANYTHING

Use *GOTO Anything* to open files with only a few keystrokes, and instantly jump to symbols, lines or words.

Triggered with CTRL+P, it is possible to:

- Type part of a file name to open it.
- Type @ to jump to symbols, # to search within the file, and : to go to a line number.

GOTO DEFINITION

Using information from syntax definitions, Visual Studio Code automatically generates a project-wide index of every class, method and function. This index powers *Goto Definition*, which is exposed in three different ways:

- A popup is displayed when hovering over a symbol
- Pressing F12 when the caret is on a symbol
- The *Goto Symbol in Project* functionalit.
-

CUSTOMIZE ANYTHING

Key bindings, menus, snippets, macros, completions and more - just about everything in Visual Studio Code is customizable with simple JSON files. This system gives you flexibility as settings can be specified on a per-file type and per-project basic.

PERFORMANCE

Visual Studio Code is built from custom components, providing for unmatched responsiveness. From a powerful, custom cross-platform UI toolkit, to an unmatched syntax highlighting engine, Visual Studio Code sets the bar for performance

MULTIPLE SELECTIONS

Make ten changes at the same time, not one change ten times. Multiple selections allow you to interactively change many lines at once, rename variables with ease, and manipulate files faster than ever.

Try pressing Ctrl+Shift+L to split the selection into lines and Ctrl+D to select the next occurrence of the selected word. To make multiple selections with the mouse, take a look at the Column Selection documentation.

POWERFUL API AND PACKAGE ECOSYSTEM

Visual Studio Code has a powerful, Python API that allows plugins to augment built-in functionality.

Package Control can be installed via the command palette, providing simple access to thousands of packages built by the community

SPLIT EDITING

Get the most out of your wide screen monitor with split editing support. Edit files side by side, or edit two locations in the one file. You can edit with as many rows and columns as you wish. Take advantage of multiple monitors by editing with multiple windows, and using multiple splits in each window.

Take a look at the View Layout menu for split editing options. To open multiple views into the one file, use the File New View into File menu item.

POWERFUL API AND PACKAGE ECOSYSTEM

Visual Studio Code has a powerful, Python API that allows plugins to augment built-in functionality.

Package Control can be installed via the command palette, providing simple access to thousands of packages built by the community

CUSTOMIZE ANYTHING

Key bindings, menus, snippets, macros, completions and more - just about everything in Visual Studio Code is customizable with simple JSON files. This system gives you flexibility as settings can be specified on a per-file type and per-project basis.

CROSS PLATFORM

Visual Studio Code is available for Mac, Windows and Linux. One license is all you need to use Visual Studio Code on every computer you own, no matter what operating system it uses.

Visual Studio Code uses a custom UI toolkit, optimized for speed and beauty, while taking advantage of native functionality on each platform.

PROGRAMMING IN VISUAL STUDIO CODE

Visual Studio Code is a proprietary cross-platform source code editor with a Python application programming interface (API). It natively supports many programming languages and mark-up languages, and functions can be added by users with plug-in, typically communitybuilt and maintained under free-software license **IDE.** The project is made using bootstrap and Foundation.

THEMES

- Visual Studio Code contains 23 different visual themes, with the option to download additional themes and configure custom themes via third-party plugins.
- The minimap feature shows a reduced overview of the entire file in the top-right corner of the screen. The portion of the file visible in the main editor pane is highlighted and clicking or dragging in this view scrolls the editor through the file.

PANELS,GROUPS AND SCREENMODE

- The program offers a number of screen modes including panels that can show up to four files at once as well as full screen and distraction free modes which only show one file without any of the additional menus around it.

FEATURES OF VISUAL STUDIO CODE

Column selection and multi-select editing:

This feature allows users to select entire columns at once or place more than one cursor in text, which allows for simultaneous editing All cursors then behave as if each of them was the only one in the text. Commands like move by character, move by line, text selection, move by words, move by subword, hyphen or underscore delimited), move to beginning/end of line, etc., affect all cursors independently, allowing one to edit slightly complex repetitive structures quickly without the need to use macros.

Auto completion

Visual Studio Code will offer to complete entries as the user is typing depending on the language being used. It also auto-completes variables created by the user.

Syntax highlight and high contrast display

The dark background on Visual Studio Code is intended to reduce eyestrain and increase the amount of contrast with the text. Syntax highlighting also makes syntaxes of the language easier to read.

In-editor code building:

This feature allows users to run code for certain languages from within the editor, which eliminates the need to switch out to the command line and back again. This function can also be set to build the code automatically every time the file is saved.

Snippets:

This feature allows users to save blocks of frequently used code and assign keywords to them. The user can then type the keyword and press tab to paste the block of code whenever they require it.

Other features:

Visual Studio Code has a number of features in addition to these including

- Auto-save, which attempts to prevent users from losing their work
- Customizable key assignments, a navigational tool which allows users to assign hotkeys to their choice of options in both the menus and the toolbar
- Find as you type, begins to look for the text being entered as the user types without requiring a separate dialog box

- Spell check function corrects as you type
- Macros
- Repeat the last action
- A wide selection of editing commands, including indenting and unindenting, paragraph reformatting and line joining.

INTRODUCTION OF REACT JS

React is a powerful JavaScript library for building user interfaces, primarily developed and maintained by Facebook. It's widely used for creating interactive web applications with dynamic and reusable components. React follows a component-based architecture, where the UI is broken down into small, self-contained components that manage their own state and can be composed together to build complex user interfaces.

Fig 2: REACT LOGO

Here's an introduction to some key concepts and features of React:

1. Component-Based Architecture:

- React applications are built using components, which are reusable and encapsulate both the UI and its behavior. Components can be composed together to form larger components, making it easy to build and maintain complex user interfaces.

2. Virtual DOM:

- React uses a virtual DOM (Document Object Model) to efficiently manage updates to the UI. Instead of directly manipulating the browser's DOM, React creates a lightweight representation of the DOM in memory. When the state of a component changes, React compares the virtual DOM with the actual DOM and only updates the parts that have changed, resulting in better performance.

3. JSX (JavaScript XML):

- JSX is a syntax extension for JavaScript that allows you to write HTML-like code within JavaScript. It makes it easier to define React components by combining UI markup and logic in a single file. JSX is then compiled to regular JavaScript by tools like Babel before being executed by the browser.

4. Unidirectional Data Flow:

- React follows a unidirectional data flow, where data flows downwards from parent components to child components via props (properties). This helps maintain a clear and predictable data flow within the application, making it easier to reason about and debug.

5. State Management:

- React components can have internal state, which is managed using the `useState` hook (for functional components) or the `setState` method (for class components). When the state of a component changes, React automatically re-renders the component and its children to reflect the updated state.

6. Lifecycle Methods:

- Class components in React have lifecycle methods that allow you to hook into various stages of a component's lifecycle, such as when it is mounted to the DOM, updated, or unmounted. These methods can be used to perform tasks like fetching data, subscribing to events, or cleaning up resources.

7. Hooks:

- React introduced Hooks in version 16.8 as a way to add state and other features to functional components. Hooks like `useState`, `useEffect`, `useContext`, etc., allow functional components to have state and lifecycle behavior similar to class components, making them more powerful and flexible.

TAILWIND CSS

Tailwind CSS is a utility-first CSS framework that allows developers to rapidly build custom user interfaces by composing utility classes directly in their HTML markup. Unlike traditional CSS frameworks like Bootstrap or Foundation, which provide pre-designed components and styles, Tailwind CSS provides a set of low-level utility classes that can be used to style elements.

Here are some key features and concepts of Tailwind CSS:

1. Utility-First Approach:

- Tailwind CSS embraces a utility-first approach, where styles are applied using utility classes directly in the HTML markup. For example, instead of defining a CSS class like `.btn-primary` to style a button, you would apply utility classes like `bg-blue-500`, `text-white`, and `px-4 py-2` directly to the `<button>` element.

2. Responsive Design:

- Tailwind CSS includes built-in support for responsive design using breakpoints. You can apply responsive utility classes like `sm:`, `md:`, `lg:`, and `xl:` to apply different styles based on the viewport size.

3. Customization:

- Tailwind CSS is highly customizable, allowing you to configure the framework to suit your project's specific needs. You can customize the default theme, add new utility classes, or even remove unused utility classes to optimize the final CSS bundle size.

4. Composability:

- Tailwind CSS encourages composability by allowing you to combine utility classes to create custom styles. Instead of relying on predefined components, you can compose your own components by applying utility classes as needed.

5. Optimized for Performance:

- Tailwind CSS is designed to generate minimal and optimized CSS output. By using utility classes only when needed, you can significantly reduce the size of the CSS bundle, resulting in faster load times and improved performance.

6. Dark Mode:

- Tailwind CSS includes built-in support for dark mode, making it easy to create dark-themed user interfaces by applying utility classes like `dark:` to elements.

7. Plugin Ecosystem:

- Tailwind CSS has a vibrant plugin ecosystem that extends its functionality. Plugins are available for adding features like additional utility classes, integration with third-party tools, and more.

JavaScript

Fig 3: JavaScript LOGO

JavaScript often abbreviated **JS**, is a programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS. Over 97% of websites use JavaScript on the client side for web page behaviour often incorporating third-party libraries. All major web browsers have a dedicated JavaScript engine to execute the code on users' devices. JavaScript is a high-level, often just-in-time compiled language that conforms to the ECMAScript standard. It has dynamic typing, prototype-based object-orientation, and first-class functions. It is multi-paradigm, supporting event-driven, functional, and imperative programming styles. It has application programming interfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM). The ECMAScript standard does not include any input/output (I/O), such as networking, storage, or graphics facilities. In practice, the web browser or other runtime system provides JavaScript APIs for I/O. JavaScript engines were originally used only in web browsers, but are now core components of some servers and a variety of applications. The most popular runtime system for this usage is Node.js. Although Java and JavaScript are similar in name, syntax, and respective standard libraries, the t **JavaScript** (**JS**) is a lightweight, interpreted, or just-in-time compiled programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as Node.js, Apache CouchDB and Adobe Acrobat. JavaScript is a prototype-based, multi-paradigm, single-threaded, dynamic language, supporting object-oriented, imperative, and declarative (e.g. functional programming) styles. Read more about JavaScript.

Web development (server-side),

Software development,

Mathematics,
System scripting.

What can JavaScript do?

- Python can be used on a server to create web applications.
- Adding interactive behavior to web pages
- Creating web and mobile apps
- Building web servers and developing server applications
- Game development

Why JavaScript?

- **Speed**: it's faster to execute code within a web browser in its native language than it is to execute code on the server.
- **Dynamic pages**: pages that users can interact with weren't possible before JavaScript's inception.
- **Reducing memory use**: executing the code in the browser helps free up space on servers, which helps cut costs.
- **Building responsive user interfaces**: almost all social media user interfaces rely on JavaScript.

JavaScript Syntax compared to other programming languages

JavaScript gives developers the versatility to work with the backend & frontend. With it, you can even test website applications, webpages, and more. So, JS frameworks are tools that help you develop advanced applications in a smoother manner. Before, web developers relied on basic JS and jQuery.

JavaScript Function Definitions

JavaScript functions are defined with the `function` keyword. You can use a function declaration or a function expression.

Function Declarations

```
function functionName(parameters){
  // code
}
```

Declared functions are not executed immediately. They are "saved for later use", and will be executed later, when they are invoked (called upon).

Example

```
function myFunction(a,b){

return a*b;
}
```

Function Expressions

A JavaScript function can also be defined using an **expression**.

A function expression can be stored in a variable:

Example

```
const x = function (a, b) {return a * b};
```

After a function expression has been stored in a variable, the variable can be used as a function:

Example

```
const x= function (a,b){return a*b};
let z = x(4, 3);
```

The function above is actually an **anonymous function** (a function without a name).

Functions stored in variables do not need function names. They are always invoked (called) using the variable name.

Understanding client-side JavaScript frameworks

JavaScript frameworks are an essential part of modern front-end web development, providing developers with tried and tested tools for building scalable, interactive web applications. Many modern companies use frameworks as a standard part of their tooling, so many front-end development jobs now require framework experience. In this set of articles, we are aiming to give you a comfortable starting point to help you begin learning frameworks.

As an aspiring front-end developer, it can be hard to work out where to begin when learning frameworks — there are so many different frameworks to choose from, new ones appear all the time, they mostly work in a similar way but do some things differently, and there are some specific things to be careful about when using frameworks.

JSON

JSON, or JavaScript Object Notation, is a general-purpose data interchange format that is defined as a subset of JavaScript's object literal syntax.

JavaScript is weakly typed, which means certain types are implicitly cast depending on the operation used.

- The binary `+` operator casts both operands to a string unless both operands are numbers. This is because the addition operator doubles as a concatenation operator
- The binary `-` operator always casts both operands to a number
- Both unary operators (`+`, `-`) always cast the operand to a number

Values are cast to strings like the following:

- Strings are left as-is
- Numbers are converted to their string representation
- Arrays have their elements cast to strings after which they are joined by commas (`,`)
- Other objects are converted to the string `[object Object]` where `Object` is the name of the constructor of the object

Values are cast to numbers by casting to strings and then casting the strings to numbers. These processes can be modified by defining `toString` and `valueOf` functions on the prototype for string and number casting respectively.

JavaScript has received criticism for the way it implements these conversions as the complexity of the rules can be mistaken for inconsistency. For example, when adding a number to a string, the number will be cast to a string before performing concatenation, but when subtracting a number from a string, the string is cast to a number before performing subtraction.

JavaScript type conversions

left operand	operator	right operand	result
[] (empty array)	+	[] (empty array)	"" (empty string)
[] (empty array)	+	{} (empty object)	"[object Object]" (string)
false (boolean)	+	[] (empty array)	"false" (string)
"123"(string)	+	1 (number)	"1231" (string)
"123" (string)	-	1 (number)	122 (number)
"123" (string)	-	"abc" (string)	NaN (number)

Often also mentioned is {} + [] resulting in 0 (number). This is misleading: the {} is interpreted as an empty code block instead of an empty object, and the empty array is cast to a number by the remaining unary + operator. If you wrap the expression in parentheses ({} + []) the curly brackets are interpreted as an empty object and the result of the expression is "[object Object]" as expected.

NodeJS-Express Documentation

User:

- Home Page
- Uploading Page
- File Storage
- Download Page
- Upload Section
- Add Files
- Download Page
- Show Page

Project File Structure First, let’s check the steps to build the Files Store Project NodeJS:

1. First it is important to start the project in Node.js by using npm install express
2. Then creating an dependencies inside our project by npm install <dependencies-name>
3. Create a folder for templates and add it to the server.js file.
4. Similarly, create a static file and media file as per requirements and add it to the server.js file.
5. Then you are good to go and start making the urls and views.

How To Run The Files Store App Project using Node.js – Express Framework with Source Code**?**

Step 1: Extract/unzip the file

Step 2: Go inside the project folder, open cmd and type the following commands to install Node Framework and run the webserver:

• Npm install <dependences-name.> or npm install

• Node server.js

Step 3: Finally, open the browser and go to localhost:3000 These are the step’s to run a File’s Store App Project in Node.JS

1.Npm install

First, You need to install the virtualenv, Open a command prompt by going to the project folder directory and typing CMD. After opening the CMD type ” npm install “.

```
# make sure that you are in the root directory of the project, use pwd or cd for windows
2cd RepoName
```

```
npm install
```

Fig:4

express env The above command saves the installation locally in the **node_modules** directory and creates a directory express inside node_modules. You should install the following important modules along with express

```
npm start
```

Fig:5

A node_modules directory can take up more than 200MB, so keeping all node_modules can cause space issues. If you really want to get rid of disk space issues and open to setup node_modules using npm install then the node_modules can be listed and deleted using the :

```
# list all node_modules directories in the current path
find . -name 'node_modules' -type d -prune
# remove all node_modules directories in the current path
find . -name 'node_modules' -type d -prune -exec rm -rf '{}' +
```

Fig:6

Firstly, install the Express framework globally using NPM so that it can be used to create a web application using node terminal.

```
$ npm install express –save
```

Fig:7

The above command saves the installation locally in the **node_modules** directory and creates a directory express inside node_modules. You should install the following important modules along with express –

- **body-parser** − This is a node.js middleware for handling JSON, Raw, Text and URL encoded form data.
- **cookie-parser** − Parse Cookie header and populate req.cookies with an object keyed by the cookie names.
- **multer** − This is a node.js middleware for handling multipart/form-data.

```
$ npm install body-parser --save
$ npm install cookie-parser --save
$ npm install multer –save
```

Fig:8

Request & Response

Express application uses a callback function whose parameters are **request** and **response** objects.

```
app.get('/', function (req, res) {
// --
})
```

Fig:9

- Request Object – The request object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on.
- Response Object – The response object represents the HTTP response that an Express app sends when it gets an HTTP request.

You can print **req** and **res** objects which provide a lot of information related to HTTP request and response including cookies, sessions, URL, etc.

Basic Routing

We have seen a basic application which serves HTTP request for the homepage. Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).

We will extend our Hello World program to handle more types of HTTP requests.

```
var express = require('express');
var app = express();

// This responds with "Hello World" on the homepage
app.get('/', function (req, res) {
   console.log("Got a GET request for the homepage");
   res.send('Hello GET');
})
```

```
// This responds a POST request for the homepage
app.post('/', function (req, res) {
   console.log("Got a POST request for the homepage");
   res.send('Hello POST');
})

// This responds a DELETE request for the /del_user page.
app.delete('/del_user', function (req, res) {
   console.log("Got a DELETE request for /del_user");
   res.send('Hello DELETE');
})

// This responds a GET request for the /list_user page.
app.get('/list_user', function (req, res) {
   console.log("Got a GET request for /list_user");
   res.send('Page Listing');
})

// This responds a GET request for abcd, abxcd, ab123cd, and so on
app.get('/ab*cd', function(req, res) {
   console.log("Got a GET request for /ab*cd");
   res.send('Page Pattern Match');
})

var server = app.listen(8081, function () {
   var host = server.address().address
   var port = server.address().port

   console.log("Example app listening at http://%s:%s", host, port)
})
```

Fig:10

Save the above code in a file named server.js and run it with the following command.

```
$ node server.js
You will see the following output − Example app listening at http://0.0.0.0:8081
```

Fig:11

Multer

Multer is a node.js middleware for handling multipart/form-data, which is primarily used for uploading files. It is written on top of busboy for maximum efficiency.

Installation

```
$ npm install --save multer
```

Fig:12

Usage

Multer adds a body object and a file or files object to the request object. The body object contains the values of the text fields of the form, the file or files object contains the files uploaded via the form.

Basic usage example:

Don't forget the enctype="multipart/form-data" in your form.

```
const express = require('express')

const multer  = require('multer')

const upload = multer({ dest: 'uploads/' })

const app = express();

app.post('/profile', upload.single('avatar'), function (req, res, next) {

  // req.file is the `avatar` file

  // req.body will hold the text fields, if there were any

})



app.post('/photos/upload', upload.array('photos', 12), function (req, res, next) {

  // req.files is array of `photos` files

  // req.body will contain the text fields, if there were any

})
```

```
const cpUpload = upload.fields([{ name: 'avatar', maxCount: 1 }, { name: 'gallery', maxCount: 8
}])

app.post('/cool-profile', cpUpload, function (req, res, next) {

  // req.files is an object (String -> Array) where fieldname is the key, and the value is array of
files



  // req.body will contain the text fields, if there were any

})
```

Fig:13

If your main Node.js application file is app.js, you can call it by typing:

```
node app.js
```

Fig:14

Above, you are explicitly telling the shell to run your script with node. You can also embed this information into your JavaScript file with a "shebang" line. The "shebang" is the first line in the file, and tells the OS which interpreter to use for running the script. Below is the first line of JavaScript.

While running the command, make sure you are in the same directory which contains the app.js file.

MongoDB

MongoDB is an open-source document database and leading NoSQL database. MongoDB is written in C++. This tutorial will give you great understanding on MongoDB concepts needed to create and deploy a highly scalable and performance-oriented database.

MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

Database and Collection

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

Document

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

The following table shows the relationship of RDBMS terminology with MongoDB.

RDBMS	**MongoDB**
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by MongoDB itself)
Database Server and Client	
mysqld/Oracle	mongod
mysql/sqlplus	mongo

_id is a 12 bytes hexadecimal number which assures the uniqueness of every document. You can provide _id while inserting the document. If you don’t provide then MongoDB provides a unique id for every document. These 12 bytes first 4 bytes for the current timestamp, next 3 bytes for machine id, next 2 bytes for process id of MongoDB server and remaining 3 bytes are simple incremental VALUE.

MongoDB - Advantages

Any relational database has a typical schema design that shows number of tables and the relationship between these tables. While in MongoDB, there is no concept of relationship.

Advantages of MongoDB over RDBMS

- **Schema less** − MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.
- Structure of a single object is clear.
- No complex joins.
- Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.
- Tuning.
- **Ease of scale-out** − MongoDB is easy to scale.
- Conversion/mapping of application objects to database objects not needed.
- Uses internal memory for storing the (windowed) working set, enabling faster access of data.

Use Of MongoDB

Why Use MongoDB?

- Document Oriented Storage − Data is stored in the form of JSON style documents.
- Index on any attribute
- Replication and high availability
- Auto-Sharding
- Rich queries

- Fast in-place updates
- Professional support by MongoDB

Where to Use MongoDB?

- Big Data
- Content Management and Delivery
- Mobile and Social Infrastructure

IV. Academic, Technical & Economic Feasibility

- Academic Feasibility

We haven't had any problems as we did have ITM subject in which basics of html, css and js were taught to us and we have study MySQL database in fifth semester.

- Technical Feasibility

We haven't had any problems utilising Javascript, NodeJS.

- Economic Feasibility

Till now we haven’t thought about making it commercial.

V. Risk Factors Identification &amp; their Mitigation

Creating a file upload website entails various risks, ranging from data breaches to legal liabilities. Here are some key risk factors and their potential mitigations:

1. Data Breaches:

- Risk: Unauthorized access to sensitive user data.

- Mitigation:

- Encrypt data both in transit and at rest.

- Implement strong access controls and authentication mechanisms.

- Regularly audit and update security protocols.

- Employ intrusion detection and prevention systems.

2. Malicious File Uploads:

- Risk: Uploading of malicious files (e.g., viruses, malware).

- Mitigation:
    - Implement file type verification to ensure only safe file formats are allowed.
    - Utilize antivirus scanning on uploaded files.
    - Restrict executable file uploads.
    - Employ content security policies to mitigate XSS and other injection attacks.

3. Legal Compliance:

- Risk: Violation of privacy laws (e.g., GDPR, CCPA) or copyright infringement.
- Mitigation:
    - Clearly define terms of service and acceptable use policies.
    - Obtain explicit consent from users for data processing and storage.
    - Implement mechanisms for users to report copyright infringement.
    - Regularly review and update compliance measures based on changing regulations.

4. Denial of Service (DoS) Attacks:

- Risk: Overloading servers with excessive file uploads or malicious requests.
- Mitigation:
- Implement rate limiting and request throttling.
- Use CAPTCHA or other challenge-response tests to differentiate between human and automated traffic.
    - Employ scalable infrastructure to handle sudden increases in traffic.

5. Data Loss:

- Risk: Accidental deletion or corruption of user-uploaded files.
- Mitigation:
    - Implement regular backups with redundancy.
    - Utilize version control to track changes and recover previous versions.

- Provide users with options to recover deleted files within a specified timeframe.

6. User Authentication and Authorization:

- Risk: Unauthorized access to sensitive functionality or data.

- Mitigation:

- Implement multi-factor authentication (MFA) for user accounts.

- Enforce strong password policies.

- Utilize role-based access control (RBAC) to limit user privileges.

- Regularly review and audit user access logs.

7. Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF):**

- Risk: Exploitation of web application vulnerabilities to steal user credentials or perform unauthorized actions.

- Mitigation:

- Sanitize user inputs to prevent XSS attacks.

- Utilize CSRF tokens to protect against CSRF attacks.

- Employ security headers such as Content Security Policy (CSP) to mitigate XSS risks.

8. Scalability and Performance:

- Risk: Inability to handle increasing user traffic or file uploads.

- Mitigation:

- Design the system with scalability in mind, utilizing scalable storage solutions and load balancing.

- Monitor system performance and scale resources accordingly.

- Implement caching mechanisms to improve performance.

Regular security audits and penetration testing can also help identify and address potential vulnerabilities in the system. Additionally, staying informed about emerging threats and security best practices is crucial for maintaining the security of a file upload website.

CHAPTER 2

I . Primary Research Techniques and Analysis

For a file store website specifically, here are some primary research techniques tailored to understand user needs and behaviors:

1. Usability Testing with Upload Process:

- Method: Invite users to perform tasks related to uploading files to the website, such as selecting files, naming them, and organizing them into folders.

- Analysis: Observe users as they interact with the upload interface. Note any usability issues, confusion points, or barriers encountered during the process. Gather feedback on the clarity of instructions, ease of use, and speed of uploads.

2. File Management Behavior Analysis:

- Method: Survey or interview users about their file management habits and preferences.

- Analysis: Explore how users categorize and organize files, their tagging and labeling practices, and their expectations regarding file search and retrieval. Identify pain points and areas where the current system may not align with user workflows.

3. Feature Prioritization Surveys:

- Method: Present users with a list of potential features related to file upload and storage and ask them to prioritize them based on importance.

- Analysis: Determine which features users value most and which ones they consider essential for their needs. Use this data to prioritize feature development and allocate resources effectively.

4. Security and Privacy Concerns Assessment:

- Method: Conduct interviews or focus groups specifically focused on security and privacy concerns related to file uploads and storage.

- Analysis: Explore user perceptions of data security, their expectations regarding privacy controls, and their trust in the platform to keep their files safe. Identify areas where additional security measures or transparency may be needed.

5. Performance Testing and Feedback:

- Method: Monitor the performance of file upload and storage processes in real-time and collect feedback from users.

- Analysis: Measure upload speeds, latency, and reliability of the storage infrastructure. Solicit user feedback on performance bottlenecks, such as slow upload times or file synchronization issues, and prioritize improvements accordingly.

6. Cross-Platform Compatibility Testing:

- Method: Test the file upload and storage functionality across different devices and operating systems.

- Analysis: Identify any compatibility issues or inconsistencies in user experience across platforms. Gather feedback from users on their experiences with uploading and accessing files from various devices and environments.

7. Data Usage Patterns Analysis:

- Method: Analyze user data usage patterns, including file types, sizes, and frequency of uploads and downloads.

- Analysis: Identify trends in data usage to inform storage capacity planning, optimize resource allocation, and anticipate future storage needs. Use insights to tailor storage plans and pricing structures to match user requirements.

8. Feedback Collection Mechanisms:

- Method: Implement feedback forms, suggestion boxes, or in-app feedback prompts to collect ongoing input from users.

- Analysis: Regularly review and categorize user feedback to identify recurring themes, feature requests, and areas for improvement. Use feedback to iteratively enhance the file upload and storage experience.

By leveraging these primary research techniques and analyzing the collected data, a file store website can better understand user needs, refine its features and functionality, and ultimately deliver a more satisfying user experience.

II. Secondary Research

Secondary research can provide valuable insights into existing systems, software development life cycles (SDLC), proposed system architectures, and comparisons of development software for a file uploading website. Here's how you can conduct secondary research in each of these areas:

1. Existing Systems Study:

- Search for academic papers, industry reports, and case studies on existing file uploading websites.

- Analyze competitor websites to understand their features, user experience, and technological choices.

- Review online forums, blogs, and social media discussions to gather user feedback and insights on existing platforms.

- Evaluate user reviews and ratings on software review platforms to gauge user satisfaction with different file uploading solutions.

2. Software Development Life Cycle (SDLC):

- Explore reputable sources such as textbooks, academic journals, and online resources on software engineering and SDLC methodologies.

- Study frameworks such as Agile, Scrum, Waterfall, and DevOps to understand their principles, advantages, and disadvantages.

- Examine case studies and real-world examples of companies implementing SDLC methodologies in developing file uploading websites.

- Consider industry best practices and standards for software development, including documentation, testing, and deployment processes.

3. Proposed System's Architecture:

- Research articles, whitepapers, and technical documentation on scalable and secure system architectures for web applications.

- Explore cloud computing platforms such as AWS, Google Cloud Platform, and Microsoft Azure for architecture design patterns and services relevant to file uploading.

- Consider microservices architecture, serverless computing, and containerization for building modular and scalable components of the file uploading system.

- Review case studies of similar web applications to understand their architectural choices, scalability challenges, and performance optimizations.

4. Development Software's Comparisons:

- Read reviews, comparisons, and evaluations of development software tools for web application development.

- Explore software development platforms such as GitHub, GitLab, and Bitbucket for version control and collaboration.

- Evaluate programming languages and frameworks commonly used for web development, such as JavaScript (Node.js), Python (Django), Ruby on Rails, and PHP (Laravel).

- Consider frontend frameworks/libraries (e.g., React.js, Angular, Vue.js) and backend technologies (e.g., Express.js, Flask) suitable for building interactive and responsive file uploading interfaces.

- Compare database systems (e.g., MySQL, PostgreSQL, MongoDB) based on scalability, performance, and data integrity requirements of the file storage component.

By conducting thorough secondary research in these areas, you can gain valuable insights and make informed decisions during the planning, design, and development stages of the file uploading website. Additionally, consulting with experts in relevant domains and attending industry conferences or workshops can provide additional perspectives and recommendations.

CHAPTER 3

I. Requirement Analysis

Requirement analysis is crucial for understanding the needs and expectations of stakeholders and users when developing a file upload website. Here's a structured approach to requirement analysis for such a project:

1. Identify Stakeholders:

- Determine who the primary stakeholders are, such as users, administrators, developers, and regulatory bodies.

- Understand their roles, responsibilities, and expectations regarding the file upload website.

2. Define Functional Requirements:

- Identify the core functionalities that the file upload website must support, such as:

- User registration and authentication

- Uploading files (single or multiple)

- File management (organizing, searching, and categorizing files)

- Access control (permissions and sharing)

- Version control (if applicable)

- Notifications (e.g., upload confirmation, file sharing)

- Reporting and analytics (usage statistics, storage capacity)

- Prioritize requirements based on their criticality and impact on user experience.

3. Specify Non-Functional Requirements:

- Define non-functional requirements that describe the qualities of the system, such as:

- Performance: Response time for uploading and downloading files, system availability

- Scalability: Ability to handle increasing numbers of users and files

- Security: Data encryption, user authentication, access controls

- Usability: Intuitive user interface, accessibility features

- Reliability: Error handling, data backup, disaster recovery

- Compliance: Adherence to legal and regulatory requirements (e.g., GDPR, HIPAA)

- Ensure non-functional requirements are measurable and testable.

4. Document Requirements:

- Document the gathered requirements in a clear, concise, and structured format, such as a requirements specification document.

- Use diagrams, mockups, and prototypes to illustrate user interfaces and workflows.

- Include acceptance criteria for each requirement to facilitate validation and testing.

5. Review and Validate Requirements:

- Conduct reviews and walkthroughs of the requirements documentation with stakeholders to ensure completeness and accuracy.

- Validate requirements against stakeholders' expectations and feedback to confirm alignment with project goals.

6. Manage Requirements Changes:

- Establish a process for managing changes to requirements throughout the project lifecycle.

- Document and track changes using a version control system or requirements management tool.

- Assess the impact of changes on project scope, schedule, and budget before implementation.

7. Iterate and Refine:

- Continuously iterate and refine requirements based on evolving stakeholder needs, technological advancements, and feedback from users and stakeholders.

- Ensure that requirements remain aligned with the overall project objectives and constraints.

By following these steps, you can conduct a comprehensive requirement analysis for a file upload website, ensuring that the resulting system meets the needs of stakeholders and users while adhering to quality standards and best practices.

II. System Design

Process Model

Process models are processes of the same nature that are classified together into a model. One possible use of a process model is to prescribe how things must/should/could be done in contrast to the process itself which is really what happens. A process model is roughly an anticipation of what the process will look like. What the process shall be will be determined during actual system development.

Context Diagram

The Context Diagram shows the system under consideration as a single high- level process and then shows the relationship that the system has with other external entities (systems, organizational groups, external data stores). The context diagram for cloud file store is shown as follows. The Online file store system is at the centre of the diagram. There are three entities (file, title name, security key) placed around the centre process. Data flows are involved in the interaction between the central process and the entities.

The user entity have two outgoing data flows one for uploading and another for file retrieval.

In every step there is security check into which user has to pass his security key to use the services of the website. The incoming data flow is report to user.

UML Diagram

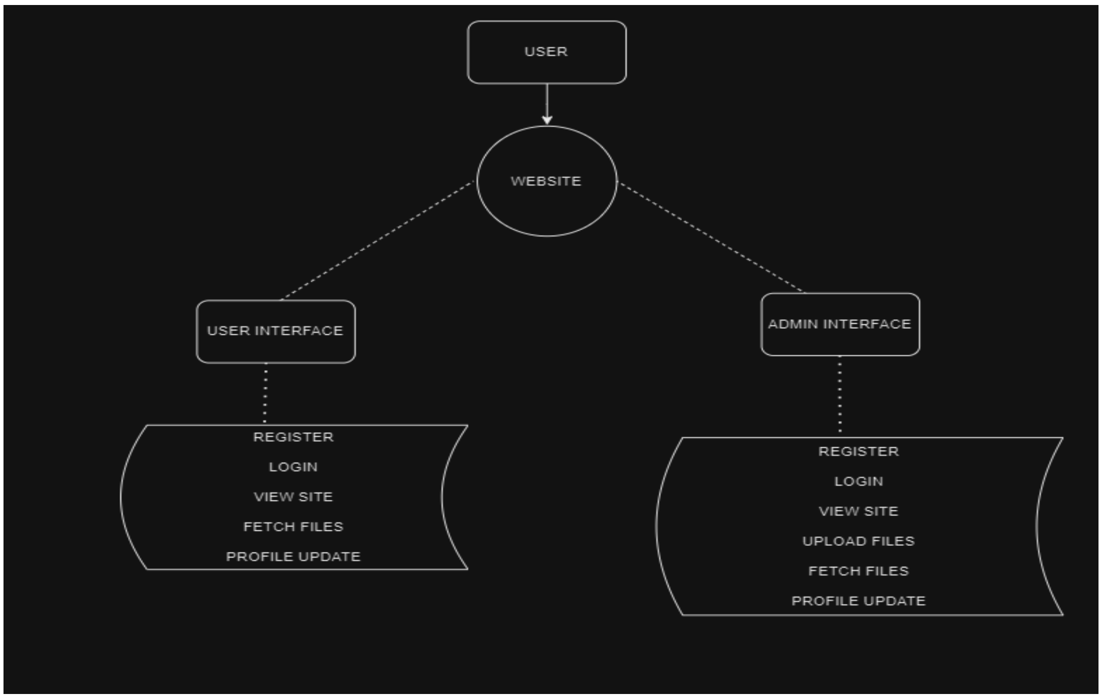


FIG 15 Data Flow Diagram

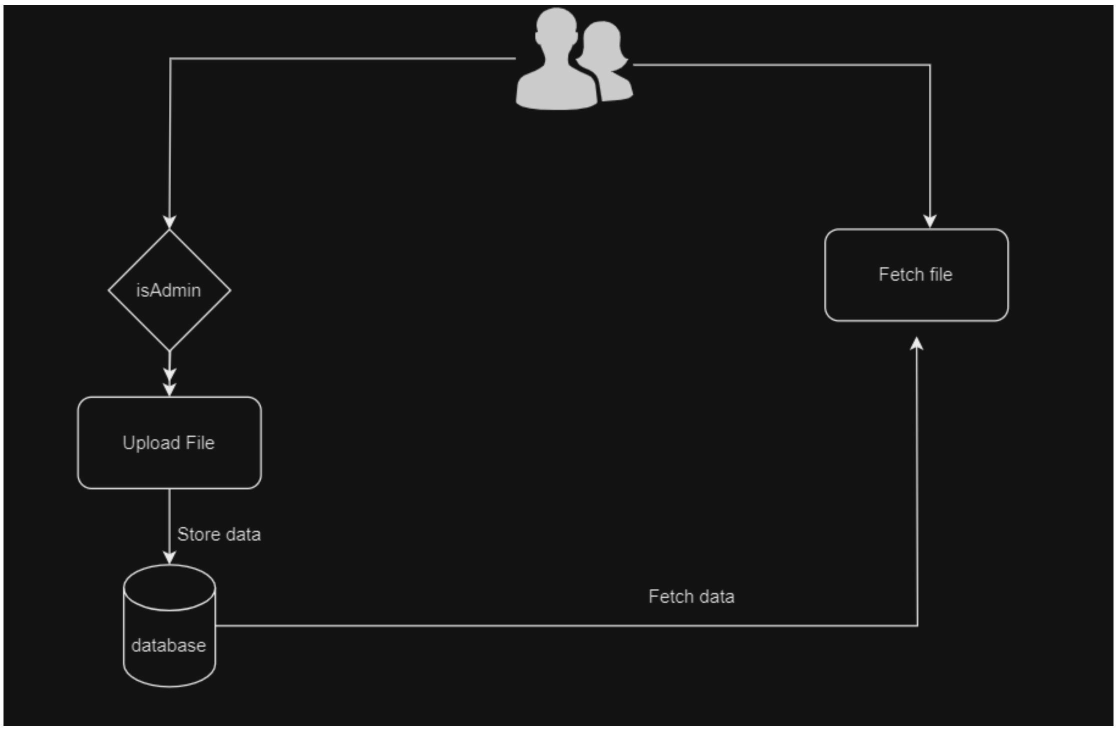


FIG 16 Tree Structure of File Store system

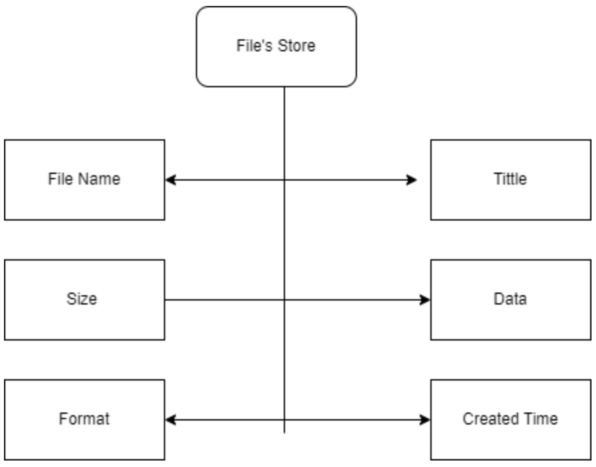


FIG 17 DataBase Requirement

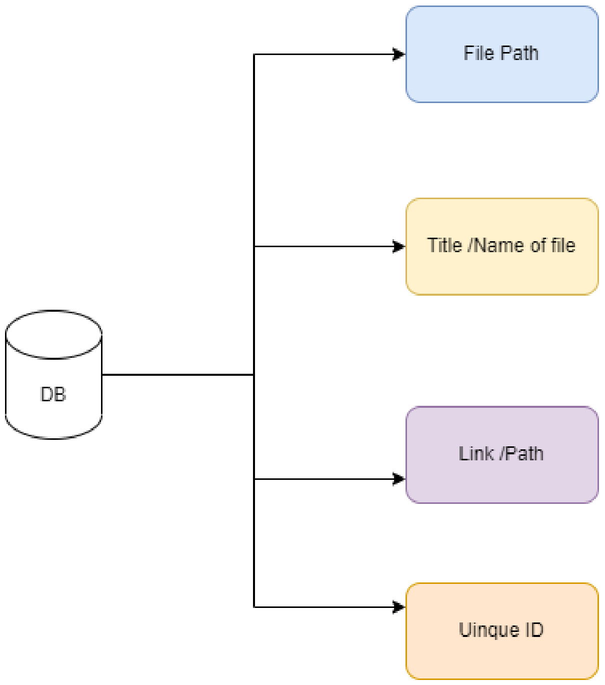
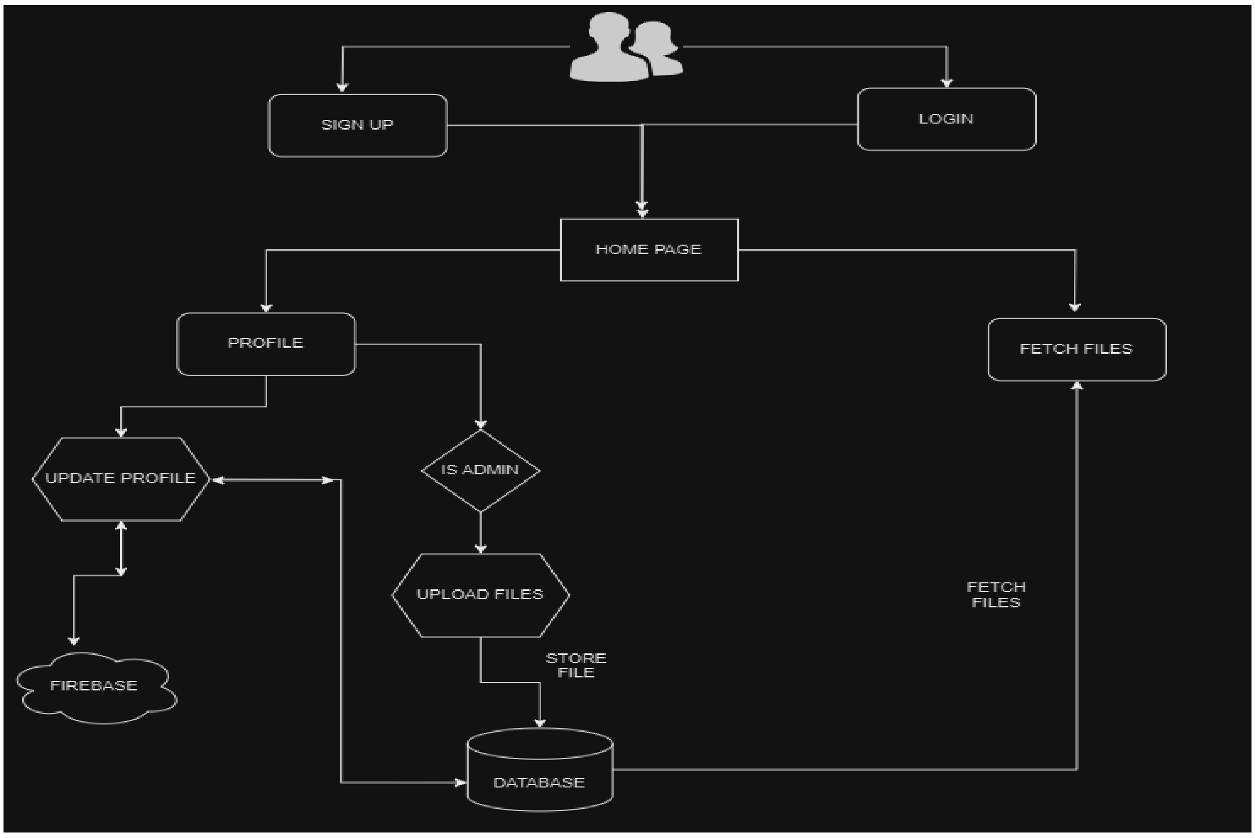


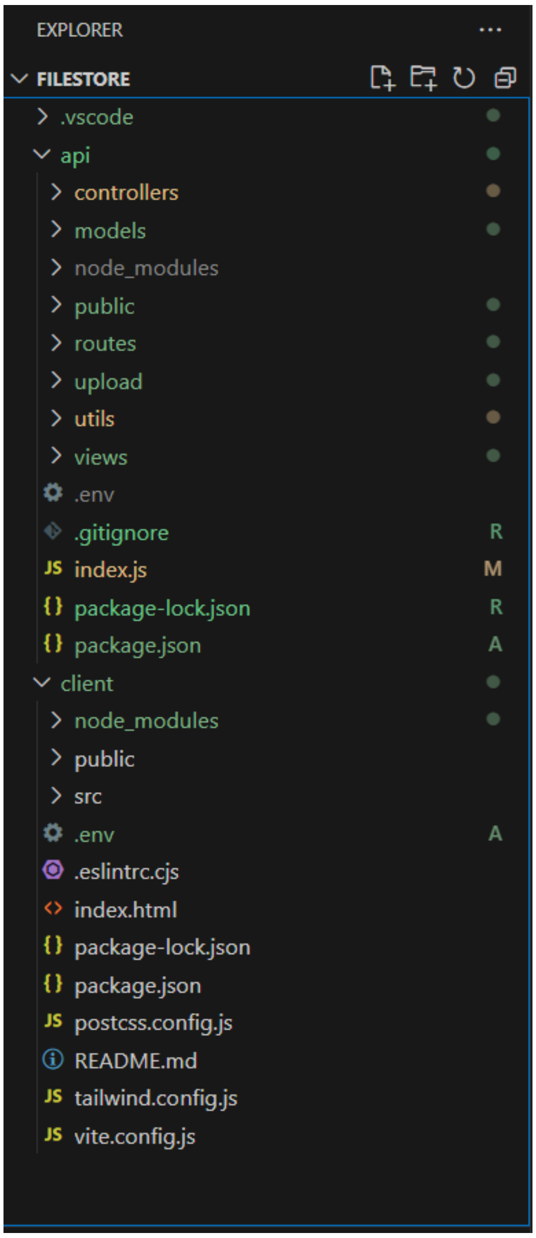
FIG 18 view of profile



CHAPTER 4

System Implementation

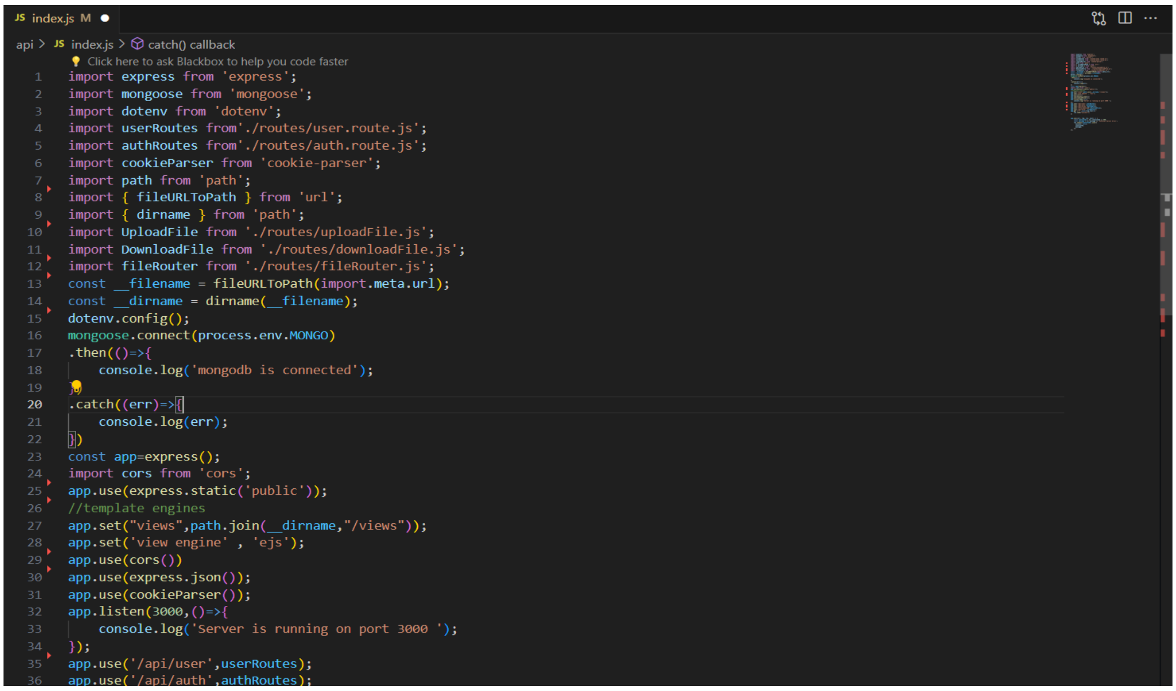
Directory Structure-It shows organization of files into hierarchy of folders used in our project.



```
EXPLORER                          ...
v FILESTORE
  > .vscode
  v api
    > controllers
    > models
    > node_modules
    > public
    > routes
    > upload
    > utils
    > views
      .env
      .gitignore                  R
      index.js                    M
      package-lock.json           R
      package.json                A
  v client
    > node_modules
    > public
    > src
      .env                        A
      .eslintrc.cjs
      index.html
      package-lock.json
      package.json
      postcss.config.js
      README.md
      tailwind.config.js
      vite.config.js
```

FIG 19 Directory structure

Index.js **-** It is the main JS file which is designing home page of our website.



index.js M

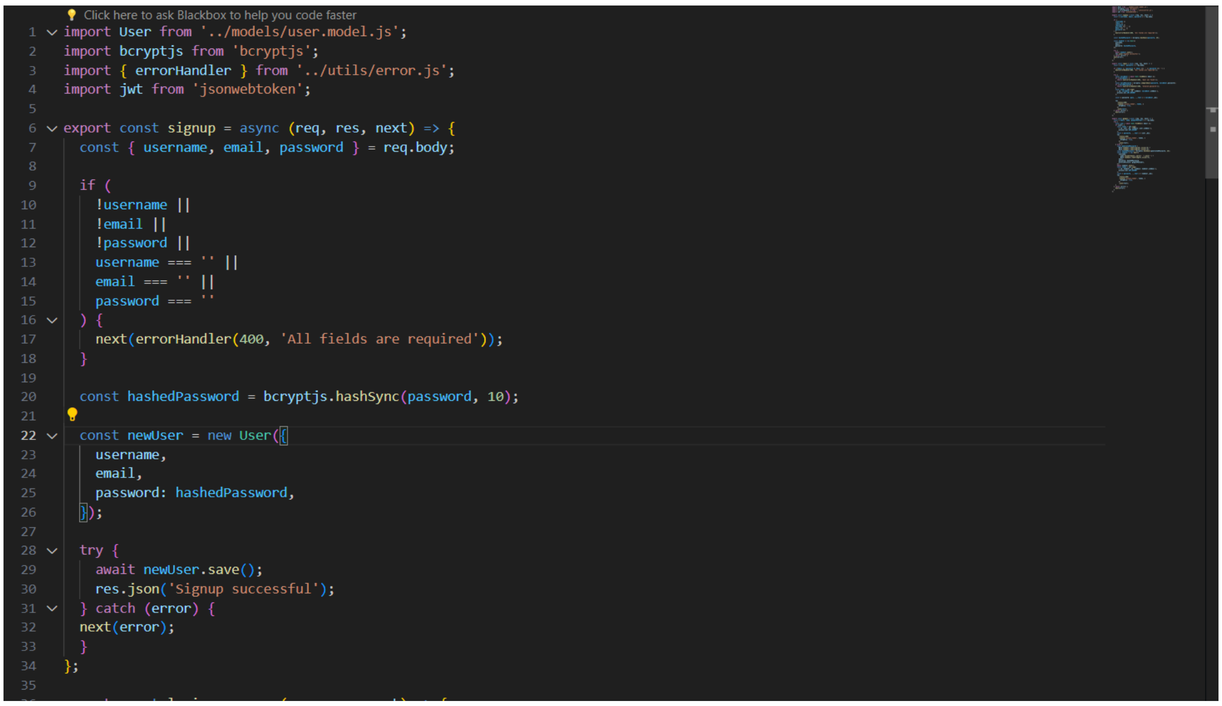
api > index.js > catch() callback

Click here to ask Blackbox to help you code faster

```javascript
import express from 'express';
import mongoose from 'mongoose';
import dotenv from 'dotenv';
import userRoutes from'./routes/user.route.js';
import authRoutes from'./routes/auth.route.js';
import cookieParser from 'cookie-parser';
import path from 'path';
import { fileURLToPath } from 'url';
import { dirname } from 'path';
import UploadFile from './routes/uploadFile.js';
import DownloadFile from './routes/downloadFile.js';
import fileRouter from './routes/fileRouter.js';
const __filename = fileURLToPath(import.meta.url);
const __dirname = dirname(__filename);
dotenv.config();
mongoose.connect(process.env.MONGO)
.then(()=>{
    console.log('mongodb is connected');
})
.catch((err)=>{
    console.log(err);
})
const app=express();
import cors from 'cors';
app.use(express.static('public'));
//template engines
app.set("views",path.join(__dirname,"/views"));
app.set('view engine' , 'ejs');
app.use(cors())
app.use(express.json());
app.use(cookieParser());
app.listen(3000,()=>{
    console.log('Server is running on port 3000 ');
});
app.use('/api/user',userRoutes);
app.use('/api/auth',authRoutes);
```

FIG 20 Index.js

Auth.controller.js **-** It is the JS file which is control the Authentication .

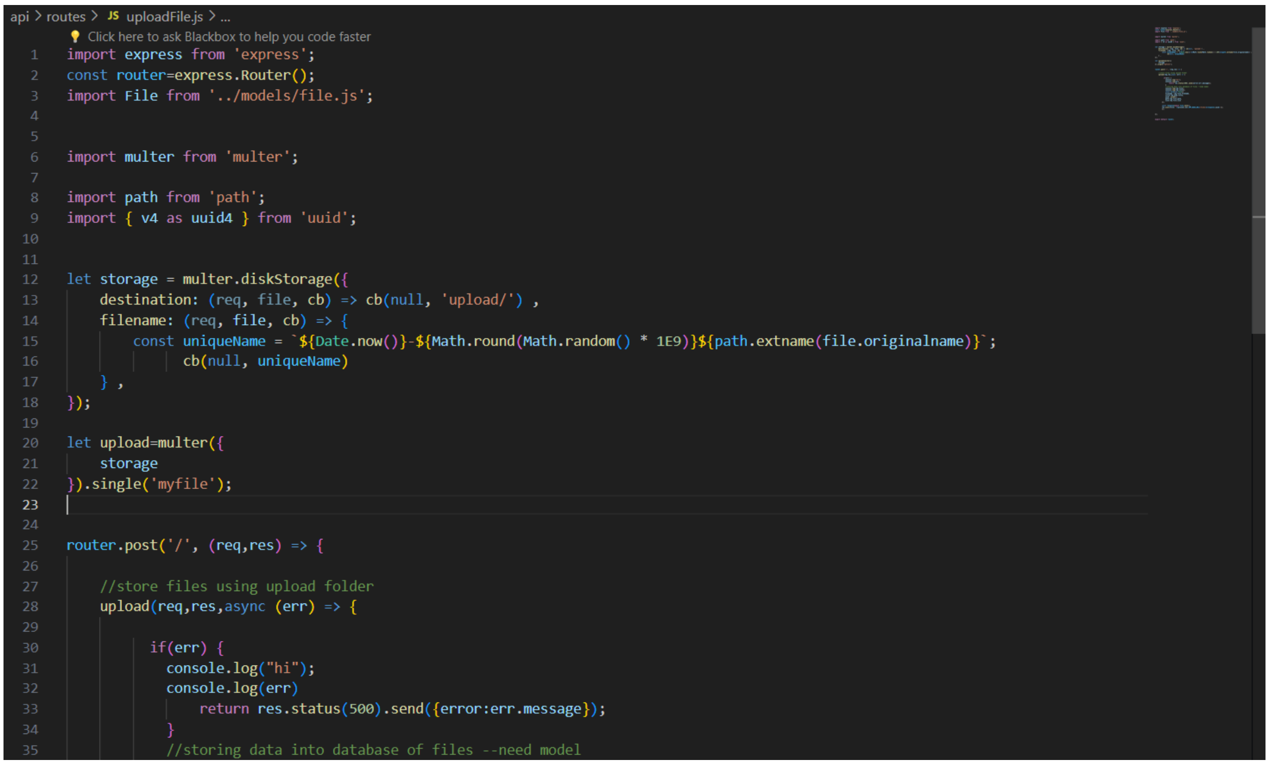


Click here to ask Blackbox to help you code faster

```javascript
import User from '../models/user.model.js';
import bcryptjs from 'bcryptjs';
import { errorHandler } from '../utils/error.js';
import jwt from 'jsonwebtoken';

export const signup = async (req, res, next) => {
  const { username, email, password } = req.body;

  if (
    !username ||
    !email ||
    !password ||
    username === '' ||
    email === '' ||
    password === ''
  ) {
    next(errorHandler(400, 'All fields are required'));
  }

  const hashedPassword = bcryptjs.hashSync(password, 10);

  const newUser = new User({
    username,
    email,
    password: hashedPassword,
  });

  try {
    await newUser.save();
    res.json('Signup successful');
  } catch (error) {
  next(error);
  }
};
```

FIG 21 Auth.controller.js

uploadFile.js **-** It is the JS file which is control uploading operation .



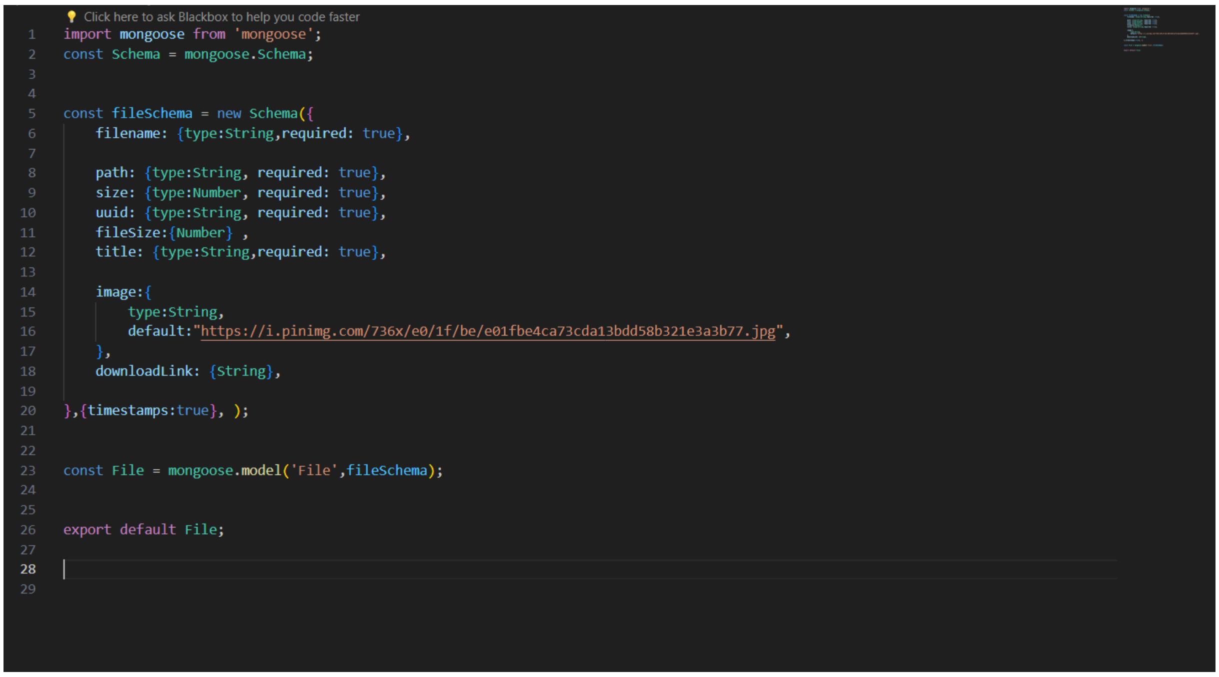
api > routes > JS uploadFile.js > ...

💡 Click here to ask Blackbox to help you code faster

```
import express from 'express';
const router=express.Router();
import File from '../models/file.js';


import multer from 'multer';

import path from 'path';
import { v4 as uuid4 } from 'uuid';


let storage = multer.diskStorage({
    destination: (req, file, cb) => cb(null, 'upload/') ,
    filename: (req, file, cb) => {
        const uniqueName = `${Date.now()}-${Math.round(Math.random() * 1E9)}${path.extname(file.originalname)}`;
            cb(null, uniqueName)
    } ,
});

let upload=multer({
    storage
}).single('myfile');


router.post('/', (req,res) => {

    //store files using upload folder
    upload(req,res,async (err) => {

        if(err) {
          console.log("hi");
          console.log(err)
              return res.status(500).send({error:err.message});
          }
          //storing data into database of files --need model
```

FIG 22 uploadFile.js

File.js- This JS file is used to create Scehma of Document for User .

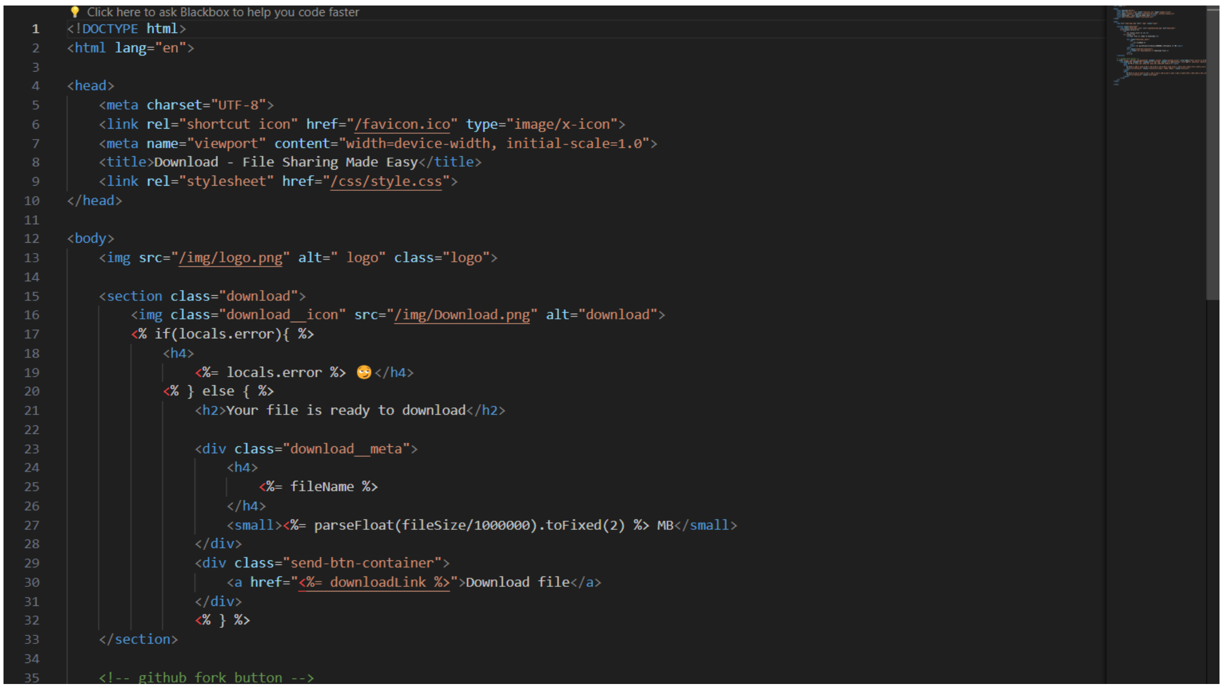


💡 Click here to ask Blackbox to help you code faster

```
import mongoose from 'mongoose';
const Schema = mongoose.Schema;


const fileSchema = new Schema({
    filename: {type:String,required: true},

    path: {type:String, required: true},
    size: {type:Number, required: true},
    uuid: {type:String, required: true},
    fileSize:{Number} ,
    title: {type:String,required: true},

    image:{
        type:String,
        default:"https://i.pinimg.com/736x/e0/1f/be/e01fbe4ca73cda13bdd58b321e3a3b77.jpg",
    },
    downloadLink: {String},

},{timestamps:true}, );


const File = mongoose.model('File',fileSchema);


export default File;

```

FIG 23. File.js

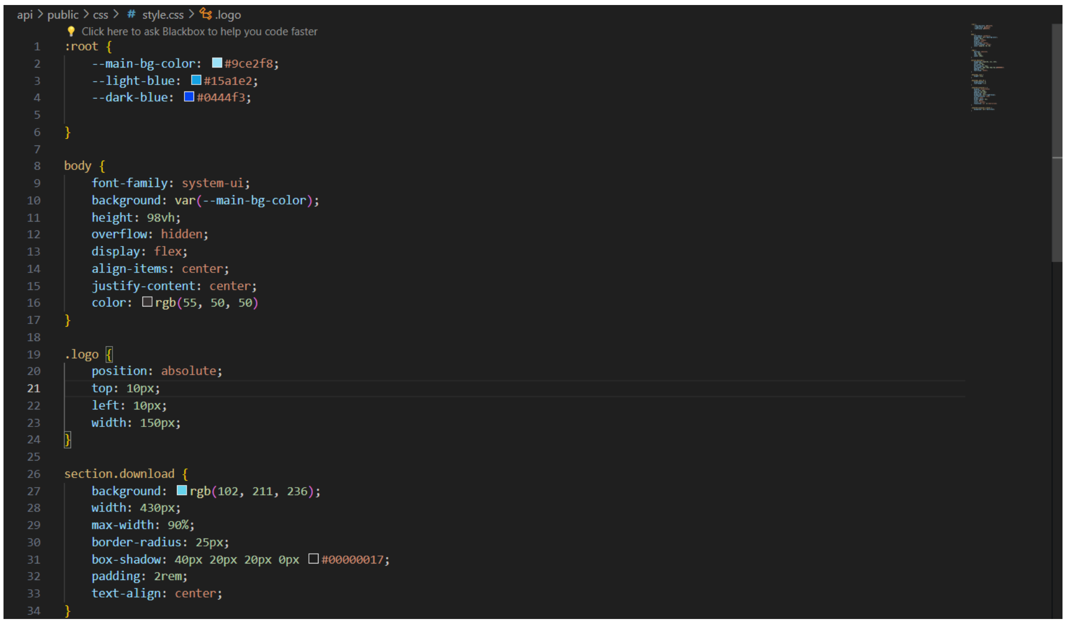
Download.ejs (Download.ejs)- These are JS functions that takes http requests and return http response like html documents.



```
Click here to ask Blackbox to help you code faster
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <link rel="shortcut icon" href="/favicon.ico" type="image/x-icon">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Download - File Sharing Made Easy</title>
    <link rel="stylesheet" href="/css/style.css">
</head>

<body>
    <img src="/img/logo.png" alt=" logo" class="logo">

    <section class="download">
        <img class="download__icon" src="/img/Download.png" alt="download">
        <% if(locals.error){ %>
            <h4>
                <%= locals.error %> 😏</h4>
            <% } else { %>
                <h2>Your file is ready to download</h2>

                <div class="download__meta">
                    <h4>
                        <%= fileName %>
                    </h4>
                    <small><%= parseFloat(fileSize/1000000).toFixed(2) %> MB</small>
                </div>
                <div class="send-btn-container">
                    <a href="<%= downloadLink %>">Download file</a>
                </div>
                <% } %>
    </section>

    <!-- github fork button -->
```

FIG 24. Download.ejs(download.ejs)

Style(Style.css)-This file is used to provide the CSS effect to webpage .

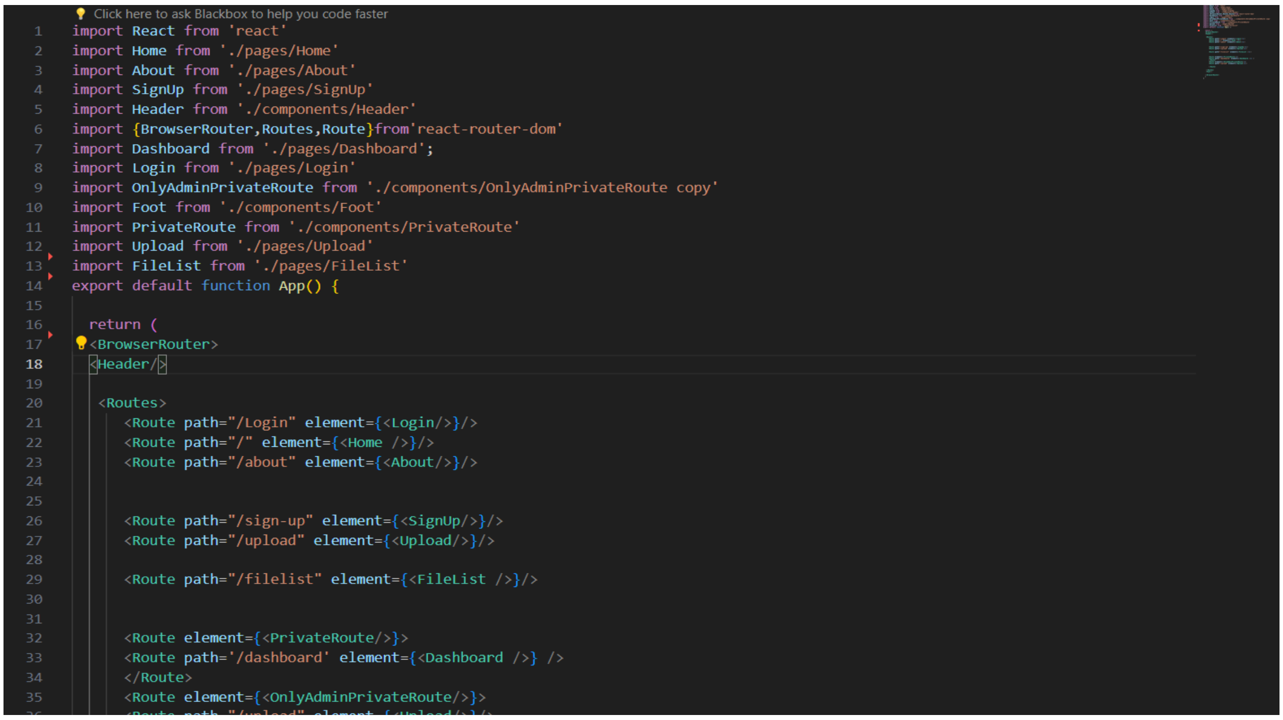


```
api > public > css > # style.css > .logo
Click here to ask Blackbox to help you code faster
:root {
    --main-bg-color: #9ce2f8;
    --light-blue: #15a1e2;
    --dark-blue: #0444f3;

}

body {
    font-family: system-ui;
    background: var(--main-bg-color);
    height: 98vh;
    overflow: hidden;
    display: flex;
    align-items: center;
    justify-content: center;
    color: rgb(55, 50, 50)
}

.logo {
    position: absolute;
    top: 10px;
    left: 10px;
    width: 150px;
}

section.download {
    background: rgb(102, 211, 236);
    width: 430px;
    max-width: 90%;
    border-radius: 25px;
    box-shadow: 40px 20px 20px 0px #00000017;
    padding: 2rem;
    text-align: center;
}
```

FIG 25 Style File (style.css)

App.jsx This file is contain all routes of the website

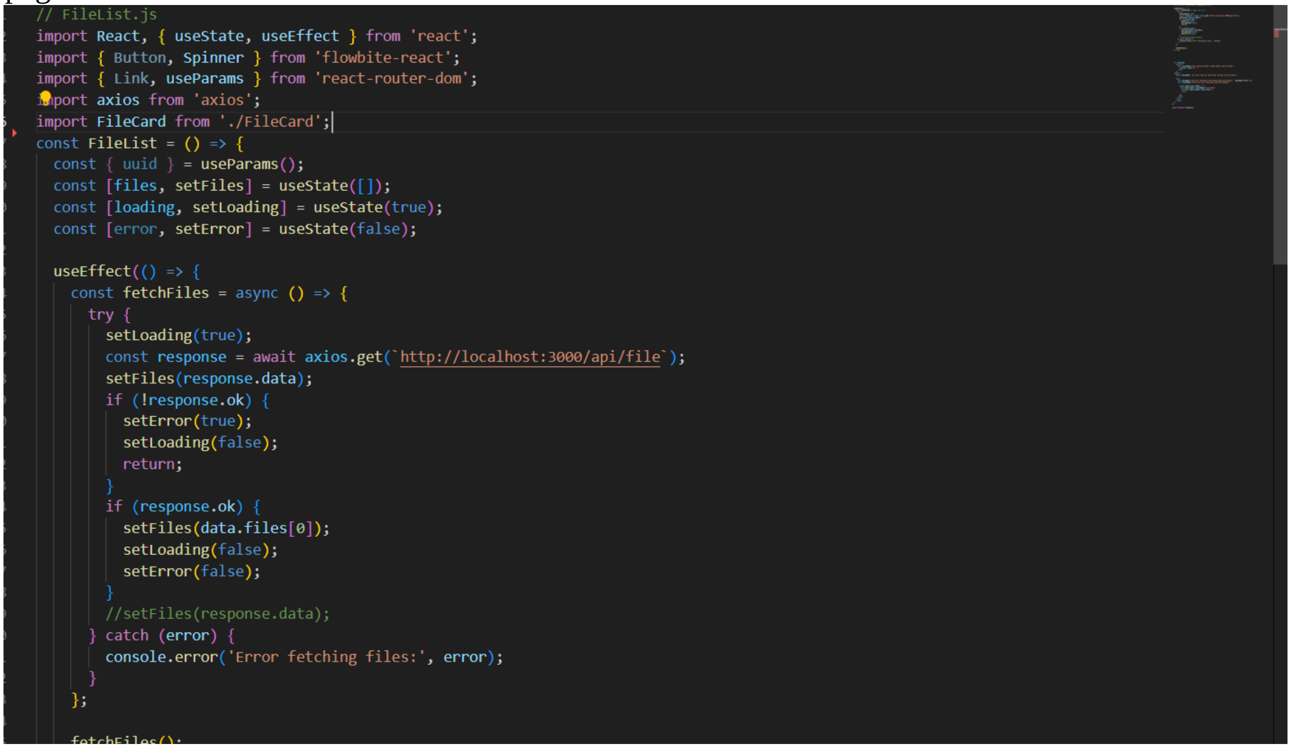
.



```
Click here to ask Blackbox to help you code faster
import React from 'react'
import Home from './pages/Home'
import About from './pages/About'
import SignUp from './pages/SignUp'
import Header from './components/Header'
import {BrowserRouter,Routes,Route}from'react-router-dom'
import Dashboard from './pages/Dashboard';
import Login from './pages/Login'
import OnlyAdminPrivateRoute from './components/OnlyAdminPrivateRoute copy'
import Foot from './components/Foot'
import PrivateRoute from './components/PrivateRoute'
import Upload from './pages/Upload'
import FileList from './pages/FileList'
export default function App() {

  return (
  <BrowserRouter>
  <Header/>

   <Routes>
      <Route path="/Login" element={<Login/>}/>
      <Route path="/" element={<Home />}/>
      <Route path="/about" element={<About/>}/>


      <Route path="/sign-up" element={<SignUp/>}/>
      <Route path="/upload" element={<Upload/>}/>

      <Route path="/filelist" element={<FileList />}/>


      <Route element={<PrivateRoute/>}>
      <Route path='/dashboard' element={<Dashboard />} />
      </Route>
      <Route element={<OnlyAdminPrivateRoute/>}>
```

FIG 26 App.jsx

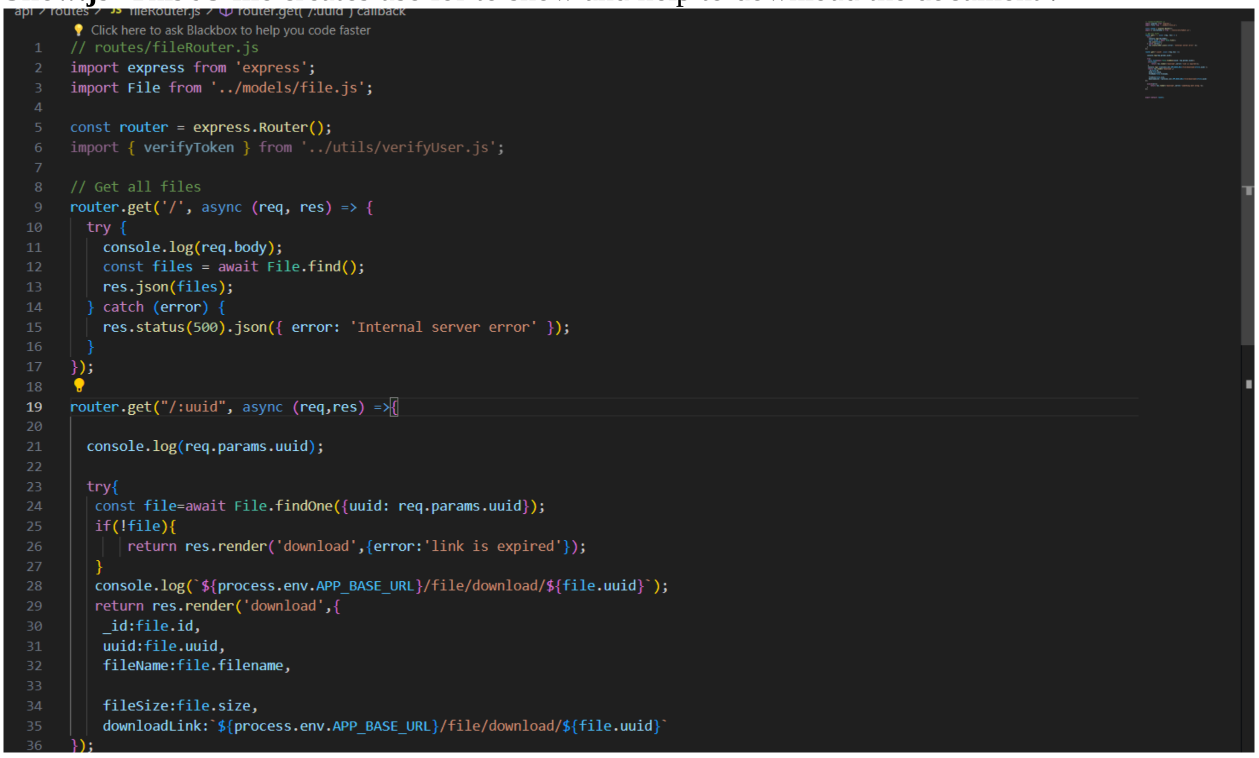
Data(filelist.jsx)-This React file involves formatting of data that how it will be shown on web page.



```
// FileList.js
import React, { useState, useEffect } from 'react';
import { Button, Spinner } from 'flowbite-react';
import { Link, useParams } from 'react-router-dom';
import axios from 'axios';
import FileCard from './FileCard';
const FileList = () => {
  const { uuid } = useParams();
  const [files, setFiles] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(false);

  useEffect(() => {
    const fetchFiles = async () => {
      try {
        setLoading(true);
        const response = await axios.get(`http://localhost:3000/api/file`);
        setFiles(response.data);
        if (!response.ok) {
          setError(true);
          setLoading(false);
          return;
        }
        if (response.ok) {
          setFiles(data.files[0]);
          setLoading(false);
          setError(false);
        }
        //setFiles(response.data);
      } catch (error) {
        console.error('Error fetching files:', error);
      }
    };

    fetchFiles();
```

FIG 27 FileList.jsx

Show.js -This JS file creates use for to show and help to download the document .



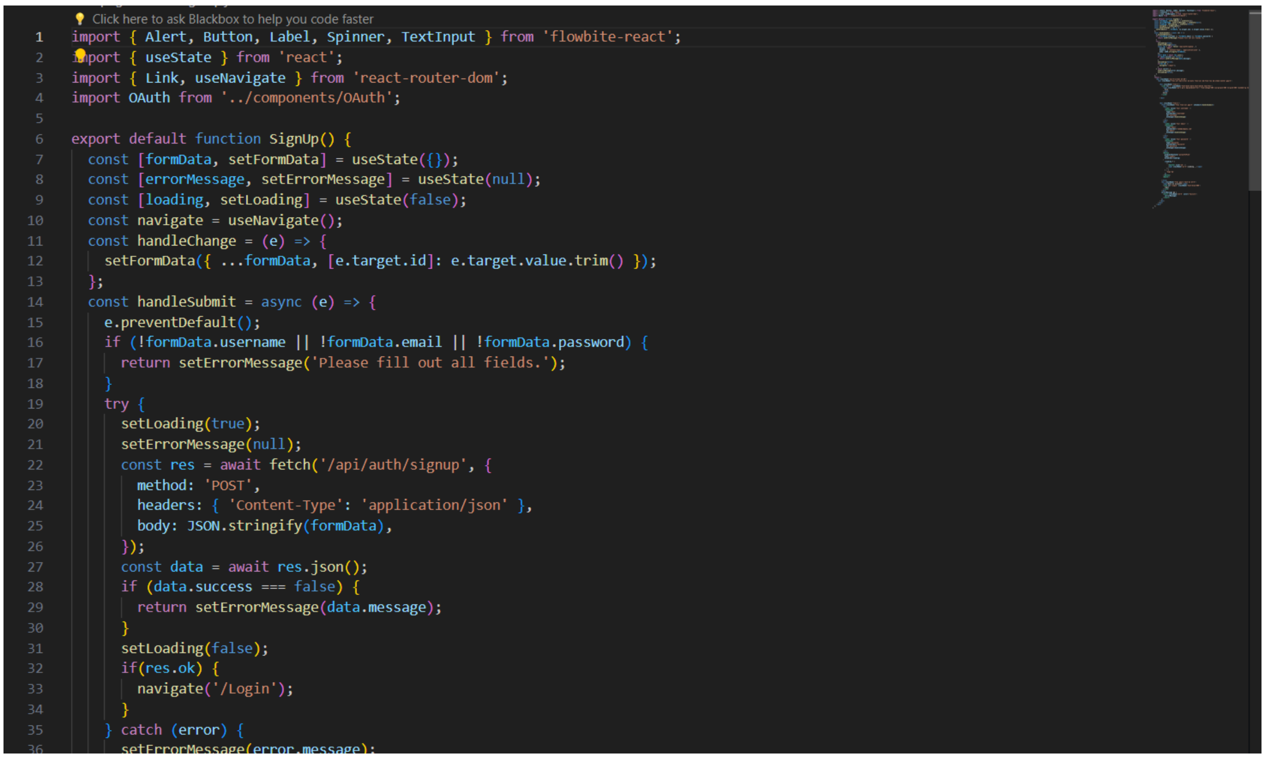
api > routes > fileRouter.js > router.get('/:uuid') callback

Click here to ask Blackbox to help you code faster

```
// routes/fileRouter.js
import express from 'express';
import File from '../models/file.js';

const router = express.Router();
import { verifyToken } from '../utils/verifyUser.js';

// Get all files
router.get('/', async (req, res) => {
  try {
    console.log(req.body);
    const files = await File.find();
    res.json(files);
  } catch (error) {
    res.status(500).json({ error: 'Internal server error' });
  }
});

router.get("/:uuid", async (req,res) =>{

  console.log(req.params.uuid);

  try{
   const file=await File.findOne({uuid: req.params.uuid});
   if(!file){
       return res.render('download',{error:'link is expired'});
   }
   console.log(`${process.env.APP_BASE_URL}/file/download/${file.uuid}`);
   return res.render('download',{
    _id:file.id,
    uuid:file.uuid,
    fileName:file.filename,

    fileSize:file.size,
    downloadLink:`${process.env.APP_BASE_URL}/file/download/${file.uuid}`
});
```

FIG 28 fileRouter.js

Signup.jsx: This React file involves formatting of data that how the user signup on web page.

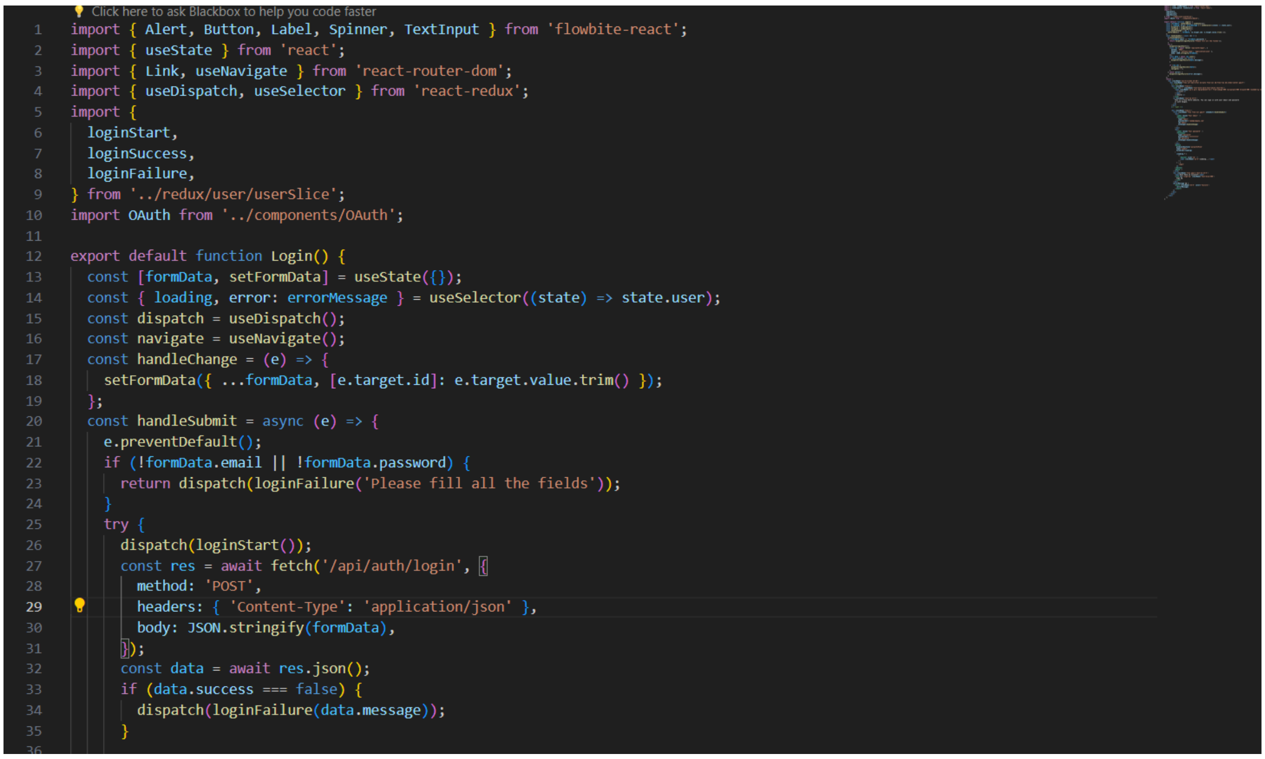


Click here to ask Blackbox to help you code faster

```
import { Alert, Button, Label, Spinner, TextInput } from 'flowbite-react';
import { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import OAuth from '../components/OAuth';

export default function SignUp() {
  const [formData, setFormData] = useState({});
  const [errorMessage, setErrorMessage] = useState(null);
  const [loading, setLoading] = useState(false);
  const navigate = useNavigate();
  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.id]: e.target.value.trim() });
  };
  const handleSubmit = async (e) => {
    e.preventDefault();
    if (!formData.username || !formData.email || !formData.password) {
      return setErrorMessage('Please fill out all fields.');
    }
    try {
      setLoading(true);
      setErrorMessage(null);
      const res = await fetch('/api/auth/signup', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(formData),
      });
      const data = await res.json();
      if (data.success === false) {
        return setErrorMessage(data.message);
      }
      setLoading(false);
      if(res.ok) {
        navigate('/Login');
      }
    } catch (error) {
      setErrorMessage(error.message);
```

FIG 29 Signup.jsx

Login.jsx: : This React file involves formatting of data that how the user Login on web page.



```
💡 Click here to ask Blackbox to help you code faster
import { Alert, Button, Label, Spinner, TextInput } from 'flowbite-react';
import { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import { useDispatch, useSelector } from 'react-redux';
import {
  loginStart,
  loginSuccess,
  loginFailure,
} from '../redux/user/userSlice';
import OAuth from '../components/OAuth';

export default function Login() {
  const [formData, setFormData] = useState({});
  const { loading, error: errorMessage } = useSelector((state) => state.user);
  const dispatch = useDispatch();
  const navigate = useNavigate();
  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.id]: e.target.value.trim() });
  };
  const handleSubmit = async (e) => {
    e.preventDefault();
    if (!formData.email || !formData.password) {
      return dispatch(loginFailure('Please fill all the fields'));
    }
    try {
      dispatch(loginStart());
      const res = await fetch('/api/auth/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(formData),
      });
      const data = await res.json();
      if (data.success === false) {
        dispatch(loginFailure(data.message));
      }
```

FIG 30 Login.jsx

CHAPTER 5

Testing

a) UNIT TESTING

Unit testing for a file upload and fetch website involves testing various components and functionalities to ensure they work correctly. Here's a structured approach to unit testing for such a website:

1. Identify Units to Test:

- Break down your codebase into smaller units that can be tested independently. For a file upload and fetch website, units might include:

  - File upload functionality
  - File retrieval functionality
  - Error handling
  - User authentication (if applicable)
  - File storage management
  - Frontend components (if unit testing frontend code)

2. Write Test Cases:

- Create test cases for each unit identified. Test cases should cover both positive and negative scenarios.

- Test cases for file upload might include:
- Uploading a file successfully
- Uploading a file with invalid file format
- Uploading a file that exceeds size limits
- Uploading multiple files
- Test cases for file retrieval might include:
- Retrieving a file by its ID
- Retrieving a non-existent file
- Retrieving files with proper authorization
- Attempting to retrieve files without proper authorization

3. Setup and Teardown:

- Ensure that each test case sets up any necessary preconditions (e.g., creating temporary files, setting up mock database) and cleans up afterward (e.g., deleting temporary files, resetting database state).

4. Mocking and Stubbing:

- Use mocking and stubbing techniques to isolate the unit being tested from its dependencies. For example, if your file upload functionality interacts with a database or a file storage service, mock those dependencies to simulate different scenarios without actually performing database operations or file operations.

5. Assertions:

- Write assertions to verify that the unit under test behaves as expected. For file upload and fetch website, this might involve checking that files are uploaded/downloaded correctly, errors are handled appropriately, proper responses are returned, etc.

6. Test Coverage:

- Aim for high test coverage to ensure that most of your code is exercised by your unit tests. Tools like code coverage analysis can help identify areas of your code that are not adequately covered by tests.

7. Parameterized Tests:

- If applicable, use parameterized tests to run the same test with different input data. For example, test file uploads with different file sizes, file formats, etc.

8. Continuous Integration:

- Integrate your unit tests into your continuous integration (CI) pipeline to automatically run them whenever code changes are made. This helps catch bugs early and ensures that new code doesn't break existing functionality.

9. Regression Testing:

- Periodically run your unit tests, especially after making changes to the codebase, to catch regressions introduced by new code.

10. Documentation:

- Document your unit tests to make it easier for other developers to understand what each test is testing and why it's important.

By following these steps, you can effectively test your file upload and fetch website to ensure its reliability and robustness.

b) Functional Testing

Functional testing for a file upload and fetch website involves testing the system as a whole to ensure that it meets the specified functional requirements. Here's a structured approach to functional testing for such a website:

1. Requirements Analysis:

- Review the functional requirements of the file upload and fetch website to understand what features it should offer, including supported file formats, file size limits, user authentication, etc.

2. Test Case Design:

- Based on the requirements, design test cases that cover all functional aspects of the website. Test cases should include both positive and negative scenarios.

- Test cases for file upload might include:

- Uploading a file with valid format and within size limits.

- Uploading a file with an invalid format or exceeding size limits.

- Testing file uploads under various network conditions (e.g., slow internet).

- Testing concurrent file uploads.

- Test cases for file fetch might include:

- Retrieving a file by its ID.

- Attempting to retrieve a non-existent file.

- Testing file retrieval under various network conditions.

- Testing file retrieval with proper and improper authorization.

3. Environment Setup:

- Set up test environments that closely resemble production environments, including databases, file storage systems, web servers, etc.

4. Execute Test Cases:

- Execute the designed test cases in the test environment.

- Document the results of each test case, including any deviations from expected behavior.

5. User Interface Testing:

- Test the user interface elements related to file upload and fetch, including buttons, forms, error messages, etc.

- Ensure that the user interface is intuitive and user-friendly.

6. Integration Testing:

- Test the integration of different components of the website, including frontend, backend, file storage systems, authentication systems, etc.

- Ensure that all components work together seamlessly.

7. Performance Testing:

- Test the performance of the website under different load conditions, including file upload/download concurrency, large file sizes, etc.

- Measure response times and ensure they meet acceptable thresholds.

8. Security Testing:

- Test the security of the website, including data encryption, prevention of unauthorized access to files, protection against common security vulnerabilities (e.g., XSS, CSRF), etc.

9. Accessibility Testing:

- Test the accessibility of the website to ensure that it can be used by people with disabilities. This includes testing with screen readers, keyboard navigation, color contrast, etc.

10. Documentation:

- Document the test results, including any issues found and their severity.

- Provide recommendations for improvements based on the test findings.

By following these steps, you can perform comprehensive functional testing for your file upload and fetch website, ensuring that it meets user requirements and performs reliably under various conditions.

CHAPTER 6

Future Scope & Limitations

As mentioned the project is on Node.js platform which is coded in Visual Studio Code with help React & Tailwind CSS and running as web page Reduce the Storage of Electronic devices such as Mobile ,Laptop. A user can upload their document and redirect on this at any time without any loss . Document upload on our server User can retrieve their document form any where to any time . Easy to share. No worry of losing a file.Exploring the future trajectory of "File Store" in the realm of PDF uploading and fetching unveils a landscape ripe with possibilities. One avenue for expansion lies in the integration of cutting-edge technologies such as artificial intelligence and machine learning. By harnessing these capabilities, "File Store" could potentially offer advanced features like automatic tagging, categorization, and even content extraction from PDF documents, thus empowering users with unprecedented efficiency and insight into their digital repositories. Moreover, the integration of sophisticated search algorithms, possibly incorporating natural language processing (NLP) and semantic search, could revolutionize the user experience by enabling more intuitive and accurate retrieval of PDF files based on context and content.Performance scalability is another aspect that warrants attention, especially as the user base and volume of uploaded PDF files grow over time. Ensuring optimal platform performance under increasing loads may require diligent optimization efforts and investments in scalable infrastructure solutions.Moreover, as "File Store" continues to evolve, addressing concerns surrounding data privacy and security will remain paramount. Users entrust the platform with sensitive documents, necessitating robust measures to safeguard data integrity and confidentiality. Regular security audits, adherence to industry best practices, and transparent communication regarding data handling practices are essential pillars in maintaining user trust and compliance with regulatory requirements.

CHAPTER 7

Conclusion

A website where you can store your documents and folder for reduce the space gathering in your Phone or Devices when ever you need just follow the link and download the document or Folder with Secret Id given by author of document. Un-optimize space fill in our device like those document which we needed later for this we have to save our document in our device like mobile/laptop which require a space a lot in memory storage A web portal for storing the document in a cloud server ensure that the document will remain safe ,and generating the link for reference for user so they easily find their document . The Web application is easy to use and user friendly and secure In past days User clear space manually but in further resolution of the technology we are able to resolve the Problem automatically. We develop File's Store web Application where a user upload their File's or Document in our server with Unique Code . Server Generate the Unique Link for User so ,they easily identify their document . Firstly User have to authenticate as a admin or a user and then . When ever user click on the link they are able to download those document from server and it reduce the memory management Problem

In the ever-expanding digital landscape where the volume of data we generate and manage continues to burgeon, "File Store" emerges as a beacon of innovation, a veritable oasis amidst the desert of digital clutter. Through its meticulous design and steadfast commitment to user-centric principles, "File Store" not only addresses the pressing challenges of document management but also redefines the very paradigm by which we interact with our digital assets.

Moreover, "File Store" is more than just a repository for documents—it is a bastion of security and trust. Through robust encryption protocols, stringent access controls, and regular security audits, the platform safeguards users' sensitive information with unwavering vigilance. Users can rest assured that their documents are stored safely in the cloud, protected from prying eyes and malicious threats.

CHAPTER 7

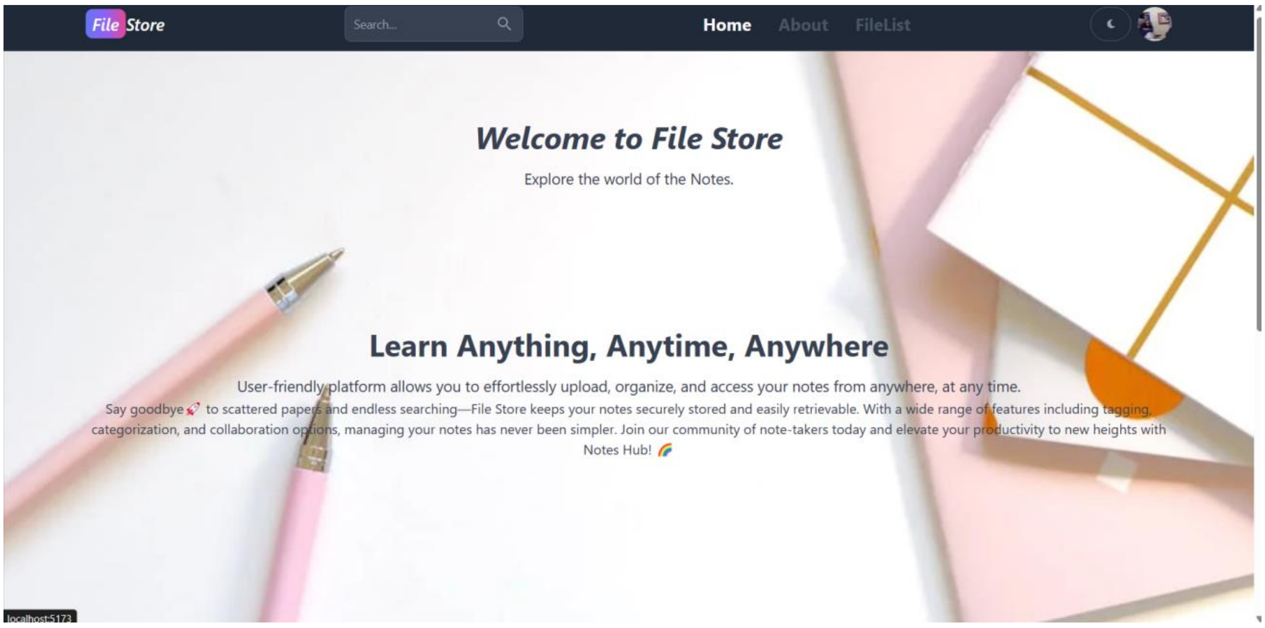
Bibliography

- Beginning MERN Stack: Build and Deploy a Full Stack MongoDB, Express, React, Node.js by Medium.com
- Notes Designed by Net Ninja
- JavaScript and JQuery: Interactive Front-End Web Development by W3School
- Visited
- https://medium.com/@rajatdhoot/learn-build-a-mern-stack-application-in-100-hours-part-1-ee56e3f61979
- https://namastedev.com/learn/namaste-react

CHAPTER 8

Appendix

HOME PAGE



Welcome to File Store

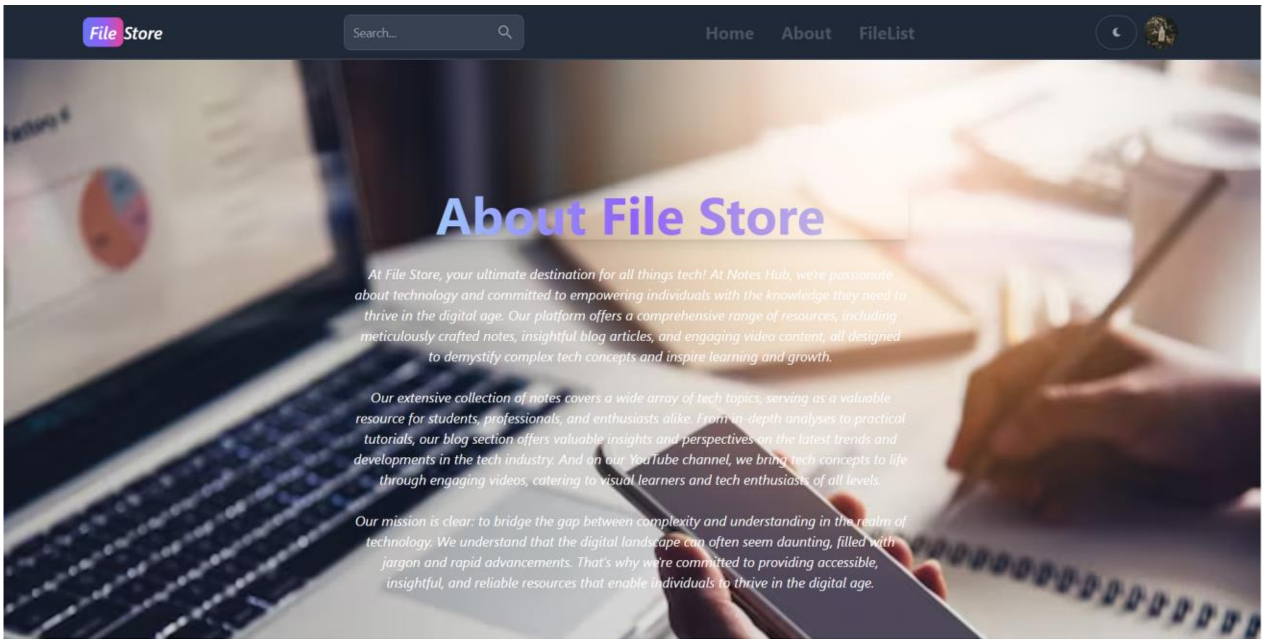
Explore the world of the Notes.

Learn Anything, Anytime, Anywhere

User-friendly platform allows you to effortlessly upload, organize, and access your notes from anywhere, at any time.
Say goodbye 🚀 to scattered papers and endless searching—File Store keeps your notes securely stored and easily retrievable. With a wide range of features including tagging, categorization, and collaboration options, managing your notes has never been simpler. Join our community of note-takers today and elevate your productivity to new heights with Notes Hub! 🌈

FIG 30 Home page

ABOUT PAGE



About File Store

At File Store, your ultimate destination for all things tech! At Notes Hub, we're passionate about technology and committed to empowering individuals with the knowledge they need to thrive in the digital age. Our platform offers a comprehensive range of resources, including meticulously crafted notes, insightful blog articles, and engaging video content, all designed to demystify complex tech concepts and inspire learning and growth.

Our extensive collection of notes covers a wide array of tech topics, serving as a valuable resource for students, professionals, and enthusiasts alike. From in-depth analyses to practical tutorials, our blog section offers valuable insights and perspectives on the latest trends and developments in the tech industry. And on our YouTube channel, we bring tech concepts to life through engaging videos, catering to visual learners and tech enthusiasts of all levels.

Our mission is clear: to bridge the gap between complexity and understanding in the realm of technology. We understand that the digital landscape can often seem daunting, filled with jargon and rapid advancements. That's why we're committed to providing accessible, insightful, and reliable resources that enable individuals to thrive in the digital age.

FIG 31 About page

Uploading Page

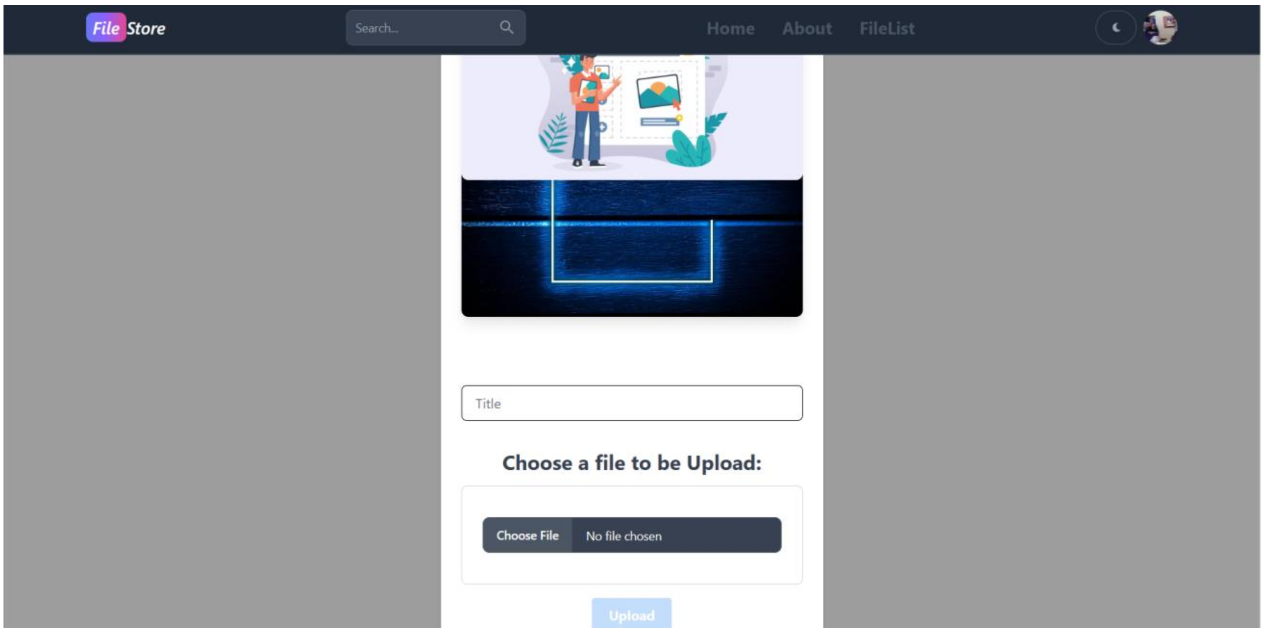
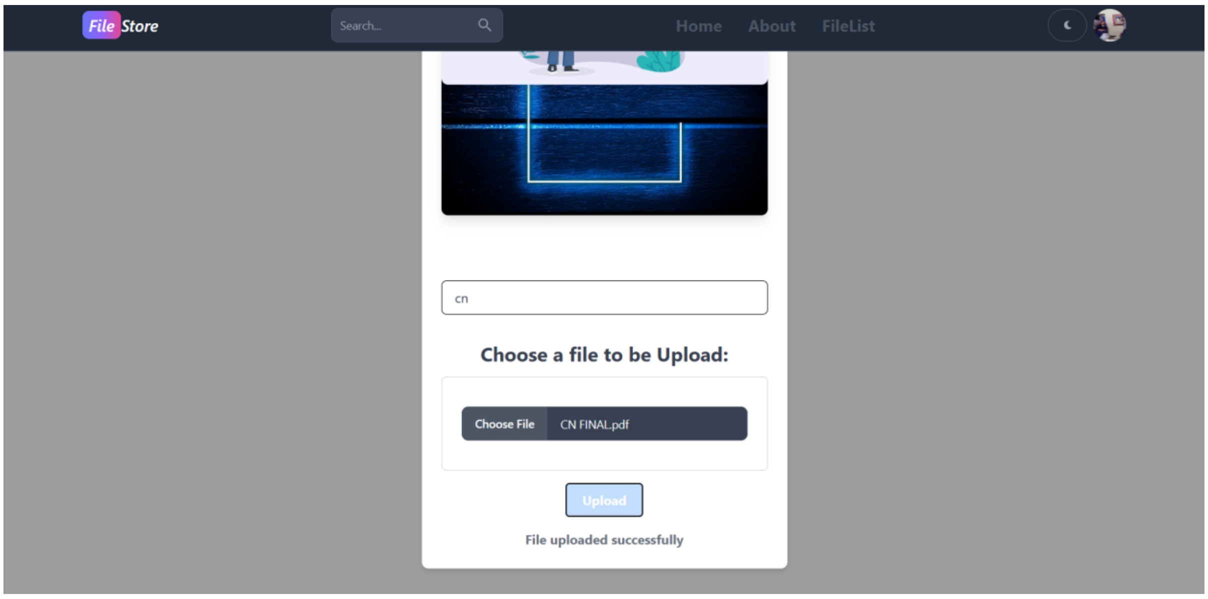


FIG 32(a) Uploading page



File uploaded successfully

FIG 33(b) Uploaded page

Fetch PDF Page

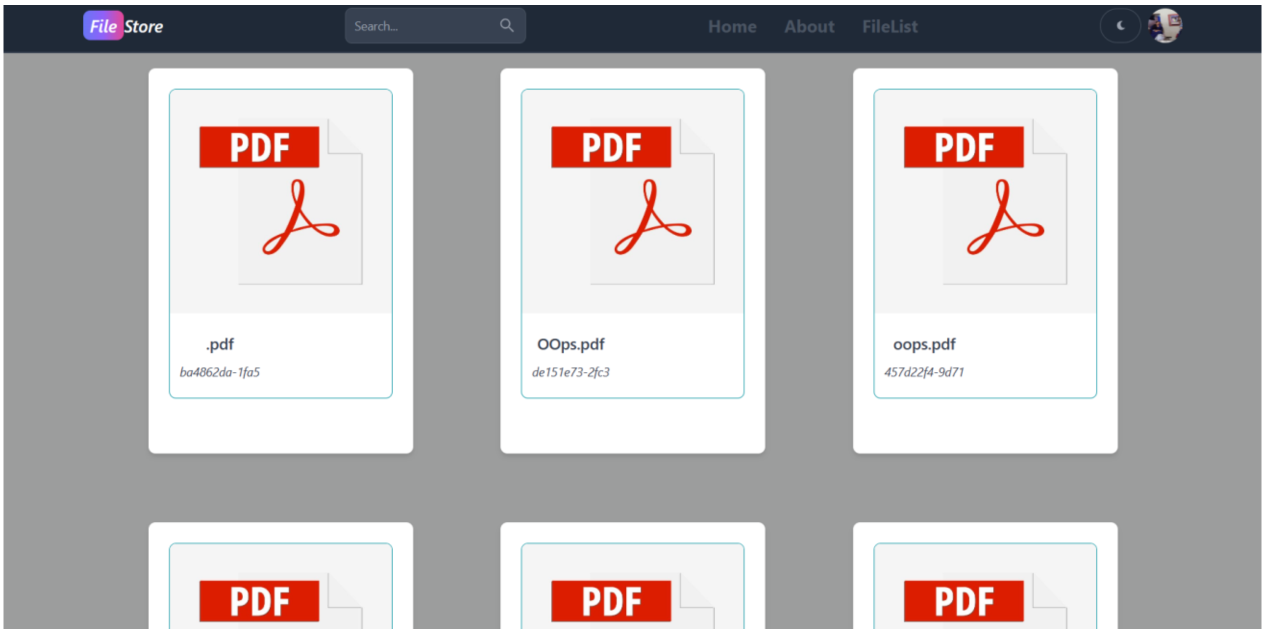
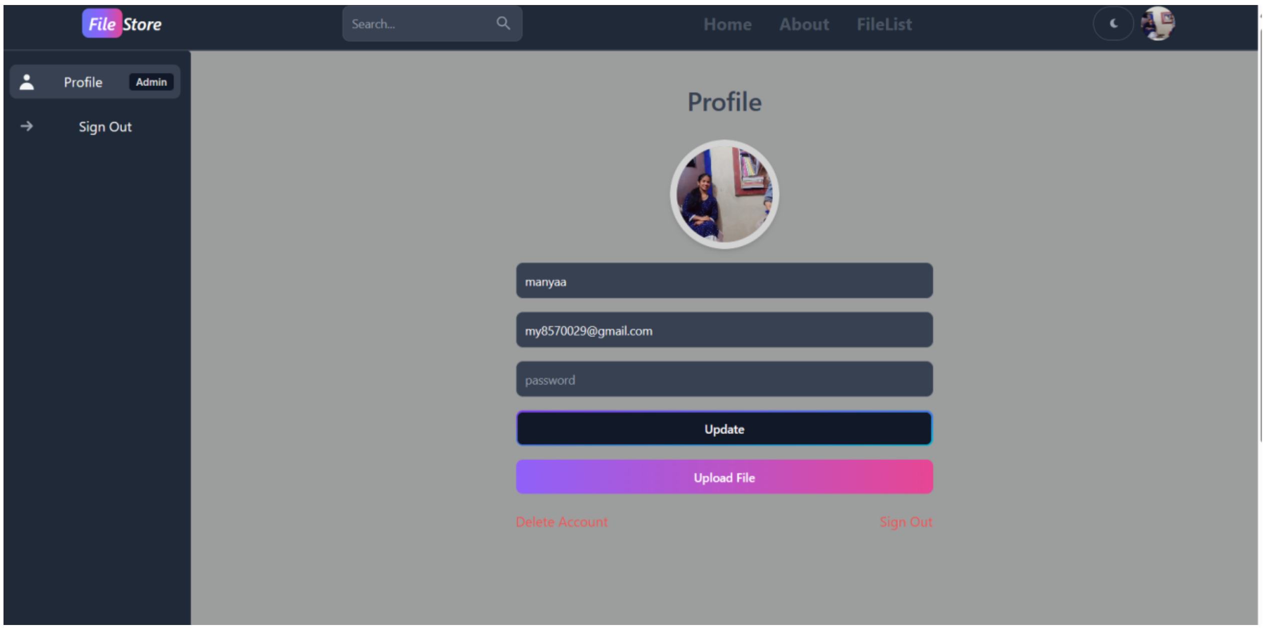


FIG 34 Fetch PDF

Profile Page



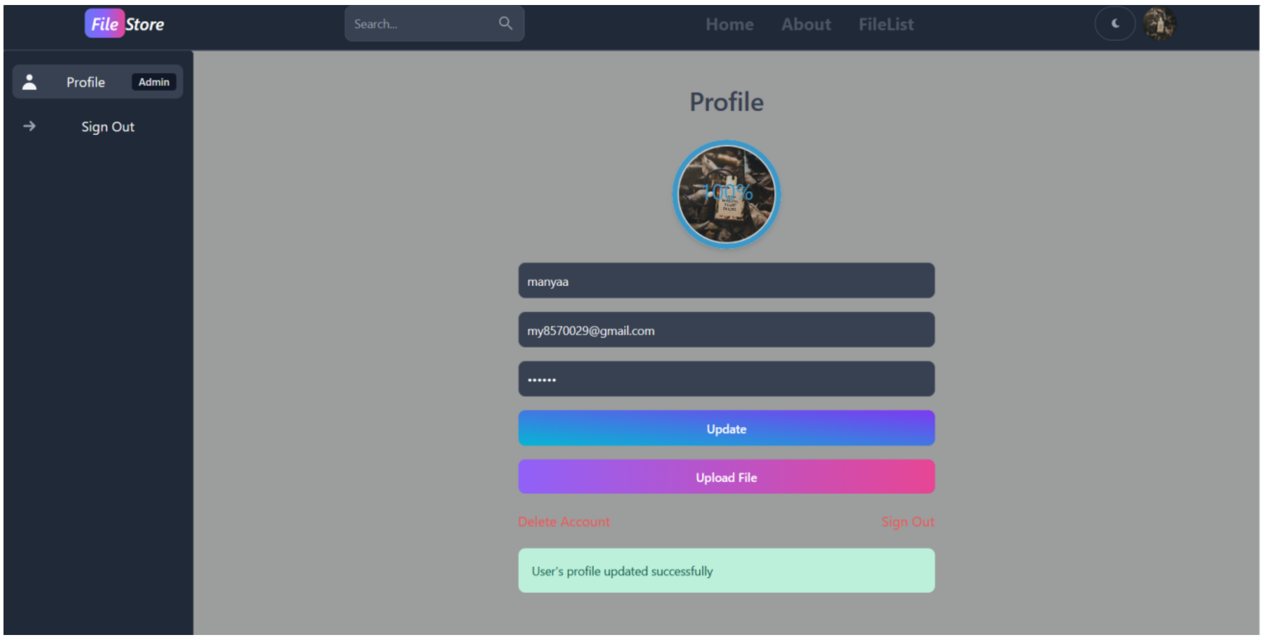
manyaa

my8570029@gmail.com

password

FIG 35 (a) Profile Page

Updated Profile Page



manyaa

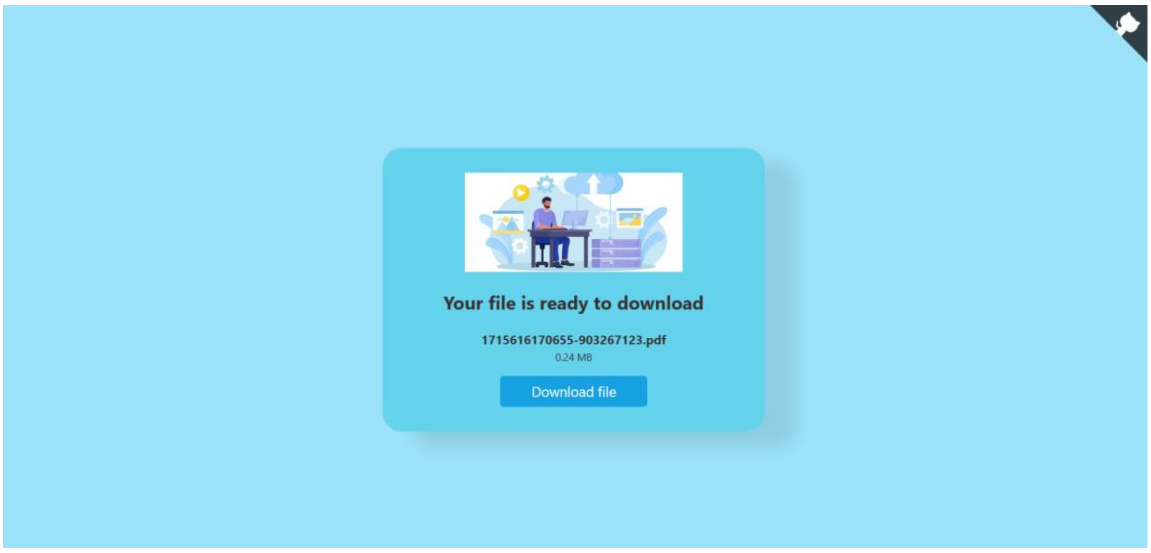
my8570029@gmail.com

••••••

User's profile updated successfully

FIG 36(b) Updated Profile Page

Download Page



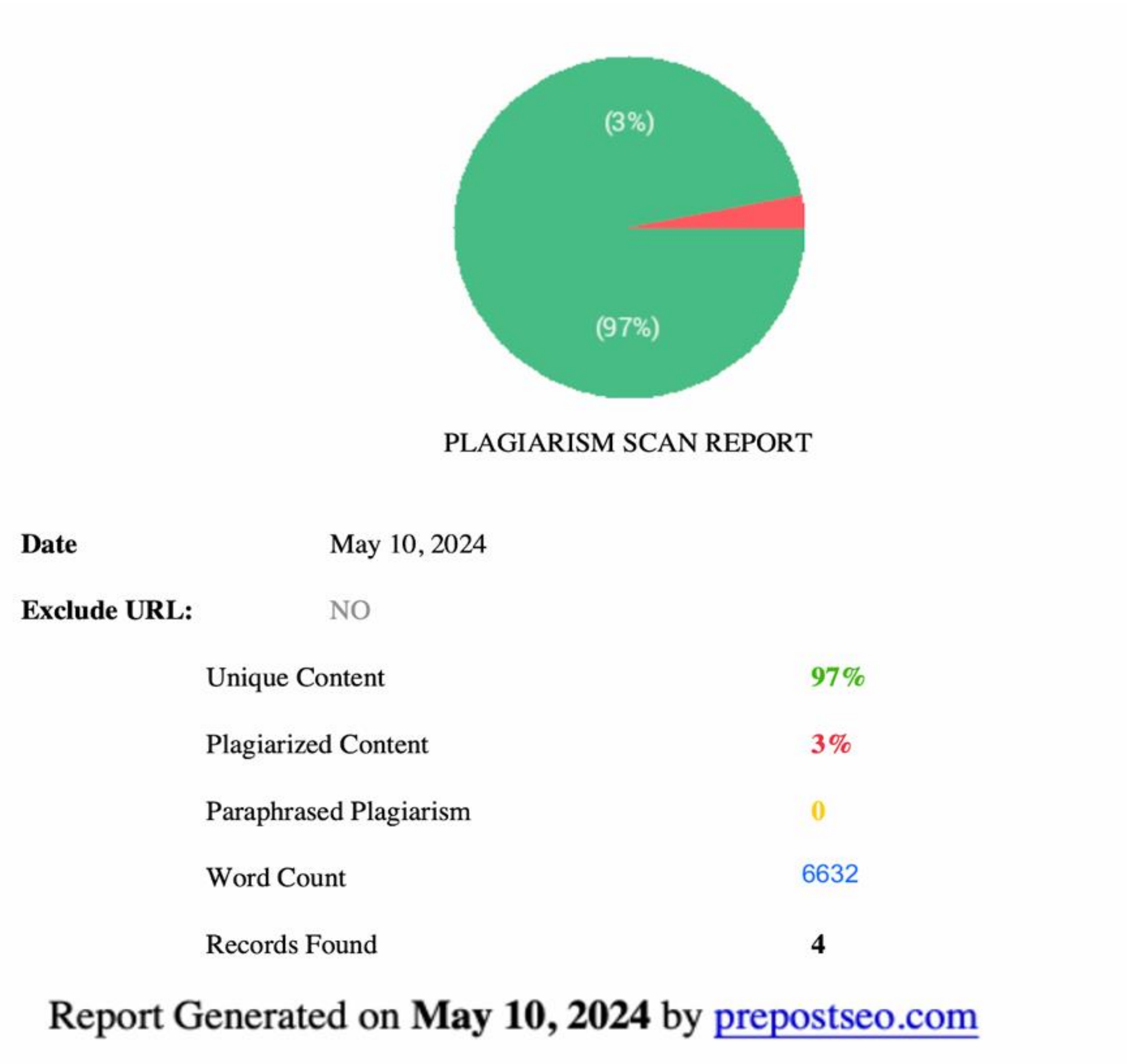
Your file is ready to download

1715616170655-903267123.pdf

0.24 MB

FIG 37 Download Page

Plagiarism Report



PLAGIARISM SCAN REPORT

Date	May 10, 2024
Exclude URL:	NO

Unique Content	**97%**
Plagiarized Content	**3%**
Paraphrased Plagiarism	**0**
Word Count	6632
Records Found	**4**

Report Generated on **May 10, 2024** by prepostseo.com