

**A PROJECT REPORT**

*On*

**Storeit**

*Submitted by*

**Mr. PRATHAM RAJESH PARMAR**

*in partial fulfilment for the award of the degree*

*of*

**BACHELOR OF SCIENCE**

*in*

**COMPUTER SCIENCE**

*Under the guidance of*

**Prof. Pallavi Awate**

**Department Of Computer Science**

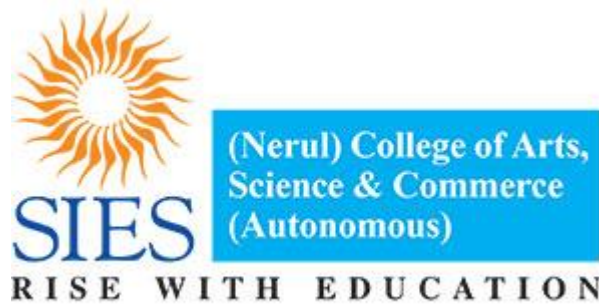


**SIES (Nerul) College of Arts, Science and Commerce (Autonomous)**

**NAAC Re-accreditation (3rd Cycle) Grade "A"**

**Sri Chandrasekarendra Saraswati Vidyapuram, Plot 1-C, Sector V, Nerul, Navi  
Mumbai – 400 706**

**(Sem VI) (2024 – 2025)**



**SIES (Nerul) College of Arts, Science and Commerce (Autonomous)**

**NAAC Re-accreditation (3rd Cycle) Grade "A"**

Sri Chandrasekarendra Saraswati Vidyapuram, Plot 1-C, Sector V, Nerul, Navi  
Mumbai – 400 706

## **Department of Computer Science**

### **CERTIFICATE**

This is to certify that **Mr. Pratham Rajesh Parmar** of **T.Y.B.Sc (Semester-VI)** class has satisfactorily completed the Project **Store-It**, to be submitted in the partial fulfillment for the award of **Bachelor of Science in Computer Science** during the academic year **2024 – 2025**.

**Seat No: S.22.89**

**Date of Submission:**

---

**Project Guide**

---

**Head of Department**

**(Prof. Dr. Sheeja K.)**

---

**College Seal**

---

**Signature of Examiner**

## **DECLARATION**

I, **Pratham Rajesh Parmar**, hereby declare that the project entitled “**StoreIt**” submitted in the partial fulfillment for the award of **Bachelor of Science in Computer Science** during the academic year **2024 – 2025** is my original work and the project has not formed the basis for the award of any degree, associateship, fellowship or any other similar titles.

**Signature of the Student:**

**Place:**

**Date:**

## **ACKNOWLEDGMENT**

I extend my sincere gratitude to my parents and my family for providing me with the necessary facilities, a conducive environment, and the opportunity for education.

A heartfelt thank you to my esteemed Project Guide, Professor Pallavi Awate, for her invaluable guidance and unwavering supervision, which played a pivotal role in the successful completion of the project.

Special thanks to all the students whose challenges and issues I observed, inspiring the conception and completion of this project.

# PREFACE

In today's digital world, efficient file management and seamless sharing are essential for both personal and professional use. **Store-It** is a modern storage management and sharing platform designed to provide users with a streamlined and intuitive experience. This project aims to simplify file organization, accessibility, and collaboration, ensuring users can store, manage, and share their data effortlessly.

Built with the latest technologies—**React 19, Next.js 15, Appwrite, TailwindCSS, ShadCN, and TypeScript**—Store-It delivers a responsive, scalable, and secure solution for handling digital assets. Users can **upload files of any format from their PC, rename, delete, download, and share them within the platform**. Additionally, an **automatic categorization system** enhances organization, allowing for a hassle-free experience

## Application Purpose

The purpose of **Store-It** is to provide users with a seamless and efficient platform for **storing, managing, and sharing files** securely. It enables users to **upload files of any format**, rename, delete, download, and share them with others on the platform while ensuring **automatic categorization** for better organization. Built with modern web technologies, Store-It enhances **collaboration and accessibility**, making file management effortless for individuals and teams alike.

## Main Features

**User Authentication with Appwrite:** Implement signup, login, and logout functionality using Appwrite's authentication system.

**File Uploads:** Effortlessly upload a variety of file types, including documents, images, videos, and audio, ensuring all your important data.

**View and Manage Files:** Users can browse through their uploaded files stored in Appwrite storage, view on a new tab, rename file or delete.

**Download Files:** Users can download their uploaded files giving them instant access to essential documents.

**File Sharing:** Users can easily share their uploaded files with others, enabling collaboration and easy access to important content.

# **TABLE OF CONTENTS**

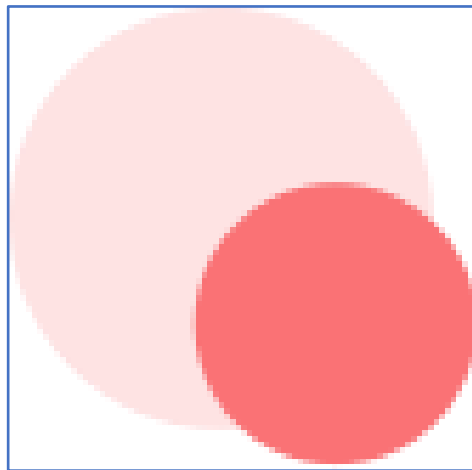
<b>Sr. No</b>	<b>Contents</b>	<b>Page no.</b>
1	Title	7
2	Introduction	8
3	Tools and Technologies	9
4	Plagiarism Report	14
5	Gantt Chart	15
6	System Requirements	16
	a) Client Side	
	b) Server Side	
7	Functional Requirements	17
8	Non-Functional Requirements	18
9	Class Diagram	19
10	E-R Diagram	20
11	Sequence Diagram	21
12	Flowchart	22
13	Use Case Diagram	23
14	Folder Structure	24-27
15	Code Snippets	27-40
16	Deployment	40-41
17	Database Overview	42
18	Results	43-52
19	References	53

# TITLE

It provides me a great opportunity to present this projecton the topic

## STORE-IT

A modern storage management and sharing platform designed to provide users with a streamlined and intuitive experience. Store-It is more than just a storage platform—it is a step toward smarter and more collaborative file management.



# INTRODUCTION

A storage management and file sharing platform that lets users effortlessly upload, organize, and share files. Built with the latest Next.js 15 and the Appwrite Node SDK, utilizing advanced features for seamless file management.

Deployed link: <https://store-it-eight-khaki.vercel.app/>

## Purpose:

The purpose of **Store-It** is to provide users with a seamless and efficient platform for **storing, managing, and sharing files** securely. It enables users to **upload files of any format**, rename, delete, download, and share them with others on the platform while ensuring **automatic categorization** for better organization.

### 1. Efficient File Storage & Management

- Allows users to **upload** and **store** files of any format.
- Provides options to **rename, delete, and organize** files for better accessibility.
- Supports **automatic categorization** to help users find their files easily.

### 2. Seamless File Sharing

- Users can **share files with others** on the same platform without needing external services.
- Ensures secure and controlled access to shared content.

### 3. Cross-Platform Accessibility

- Being a web-based platform, Store-It allows users to access their files from **any device** with an internet connection.
- Eliminates the need for external storage devices like USB drives.

### 4. Secure & Scalable Storage Solution

- Leverages **Appwrite** for secure authentication and file storage.
- Ensures **data privacy** and **protection** from unauthorized access.

### 5. User-Friendly Experience

- Built with **React 19, Next.js 15, TailwindCSS, and ShadCN** to provide a modern, responsive, and clean UI.
- Aims to **reduce complexity** in file management with an intuitive interface.



## **Key Features:**

1. **User Authentication with Appwrite:** Implement signup, login, and logout functionality using Appwrite's authentication system.
2. **File Uploads:** Effortlessly upload a variety of file types, including documents, images, videos, and audio, ensuring all your important data.
3. **View and Manage Files:** Users can browse through their uploaded files stored in Appwrite storage, view on a new tab, rename file or delete.
4. **Download Files:** Users can download their uploaded files giving them instant access to essential documents.
5. **File Sharing:** Users can easily share their uploaded files with others, enabling collaboration and easy access to important content.
6. **Dashboard:** Gain insights at a glance with a dynamic dashboard that showcases total and consumed storage, recent uploads, and a summary of files grouped by type.
7. **Global Search:** Users can quickly find files and shared content across the platform with a robust global search feature.
8. **Sorting Options:** Organize files efficiently by sorting them by date, name, or size, making file management a breeze.
9. **Modern Responsive Design:** A fresh and minimalist UI that emphasizes usability, ensuring a clean aesthetic across all devices.

## **TOOLS AND TECHNOLOGIES USED**

1. **Next.js:** A React-based framework for building web applications. It offers server-side rendering, static site generation, and API routes, making it ideal for creating fast, SEO-friendly web apps like this file sharing and management system.
2. **Appwrite:** A backend platform for web and mobile developers. It offers various services like databases, authentication, storage, and more. It can manage user authentication, file data, and securely store file-permission and auth data.
3. **TypeScript:** A typed superset of JavaScript that adds static types, improving code quality and reducing bugs. It enhances developer experience and makes your app more scalable by catching potential errors during development.

4. **TailwindCSS:** A utility-first CSS framework that allows you to design responsive and modern user interfaces with ease. It speeds up the styling process by providing ready-to-use utility classes directly in your HTML or JSX.

5. **ShadCN:** A collection of components and utilities built on top of TailwindCSS. It enhances the design and development experience by offering pre-built, accessible UI components that can be customized to fit your app's needs.

## **Target Audience:**

### **1. Individual Users (General Public - Broadest Audience):**

- **Description:** Everyday individuals who need a better way to manage and share their personal digital files. This could include students, freelancers, families, or anyone who accumulates digital content (photos, documents, videos, etc.).
- **Needs:**
  - **Simplified storage:** Easier than relying on scattered folders on their computers or multiple cloud services.
  - **Organization:** Automatic categorization to reduce digital clutter.
  - **Easy sharing with friends and family:** Sharing photos, videos, documents without email attachments or complex cloud sharing links.
  - **Accessibility from anywhere:** Accessing their files from different devices.
  - **User-friendly interface:** Simple and intuitive to use, even for non-technical users.
- **Example Use Cases:**
  - A student storing study materials and sharing notes with classmates.
  - A family sharing photos and videos with relatives.
  - A freelancer backing up important client documents and sharing deliverables.
- **Marketing Angle:** Focus on ease of use, organization, personal convenience, and "your digital life, simplified."

### **2. Small Teams and Collaborators (Small Business, Project Groups):**

- **Description:** Teams working together on projects, small businesses, or groups collaborating on shared tasks.
- **Needs (in addition to individual needs):**
  - **Collaborative file sharing:** Centralized location for team files.

- **Version control (desirable for later phases):** Managing different versions of documents.
- **Shared workspaces/folders:** Organized areas for team projects.
- **Permission control (desirable for later phases):** Controlling who can access and edit files.
- **Improved team communication (potentially integrated later):** Facilitating file sharing as part of workflows.
- **Example Use Cases:**
  - A marketing team sharing campaign assets.
  - A development team sharing project documentation and code snippets.
  - A group of researchers collaborating on a paper and sharing research data.
- **Marketing Angle:** Focus on team collaboration, improved workflow, centralized file access, boosting productivity for small teams.

### 3. Professionals and Freelancers (Specific Niches):

- **Description:** Individuals in specific professions who handle large volumes of digital files or need secure and organized storage. This could include:
  - **Photographers and Videographers:** Storing and sharing high-resolution images and video files.
  - **Designers and Creatives:** Managing design files, mockups, and client assets.
  - **Writers and Editors:** Storing drafts, articles, and collaborative documents.
  - **Architects and Engineers:** Sharing blueprints, CAD files, and project plans.
- **Needs (often more demanding in specific areas):**
  - **Large file support:** Handling large media files effectively.
  - **Preview capabilities:** Quickly previewing images, videos, and documents.
  - **Organization tailored to their workflow:** Potentially customizable categorization or tagging.
  - **Reliability and security:** Trusting the platform to store important professional assets.
  - **Client sharing capabilities:** Securely sharing files with clients.
- **Example Use Cases:**
  - A photographer sharing a portfolio with potential clients.
  - A designer sharing design mockups for client review.
  - A writer collaborating with an editor on a book manuscript.

- **Marketing Angle:** Focus on professional-grade storage, specialized features for creatives/professionals, reliability, security, and streamlining specific workflows.

#### 4. Educational Institutions (Students, Teachers, Researchers):

- **Description:** Students, teachers, researchers, and educational institutions needing a platform for sharing educational materials, assignments, research data, and collaborative projects.
- **Needs:**
  - **Learning Management System (LMS) integration (future integration potential):** Potentially integrating with existing educational platforms.
  - **Assignment submission and grading (future features):** Facilitating educational workflows.
  - **Group projects and collaboration:** Supporting student teamwork.
  - **Storage for educational resources:** Organizing lecture notes, presentations, and learning materials.
  - **Secure and private environment (especially for student data):** Privacy and security are paramount in education.
- **Example Use Cases:**
  - Students submitting assignments electronically.
  - Teachers sharing course materials with students.
  - Research groups collaborating on research papers and sharing data.
- **Marketing Angle:** Focus on educational applications, improved learning environment, collaboration in education, ease of sharing educational resources, and potentially integrations with educational tools.

## Scope:

### 1. User Authentication and Authorization:

- User registration and login (essential for security and personal storage).
- Basic user profiles (maybe just username and email initially).

### 2. File Upload:

- Uploading files of *any* format from a local computer.
- Progress indicators during upload.
- File size limits (initially, maybe a reasonable limit, and make it configurable later).

### 3. File Storage:

- Secure storage of uploaded files (leveraging Appwrite's storage capabilities).
- Managing storage quota per user (even if it's a very generous initial quota).

#### 4. **File Management:**

- **View:** Listing files in a user's storage.
- **Rename:** Changing the name of files.
- **Delete:** Removing files from storage.
- **Download:** Downloading files to a local computer.

#### 5. **Automatic Categorization:**

- Basic automatic categorization based on file type (e.g., Documents, Images, Videos, Audio, Other). This could be based on file extensions initially.
- Displaying files organized by category.

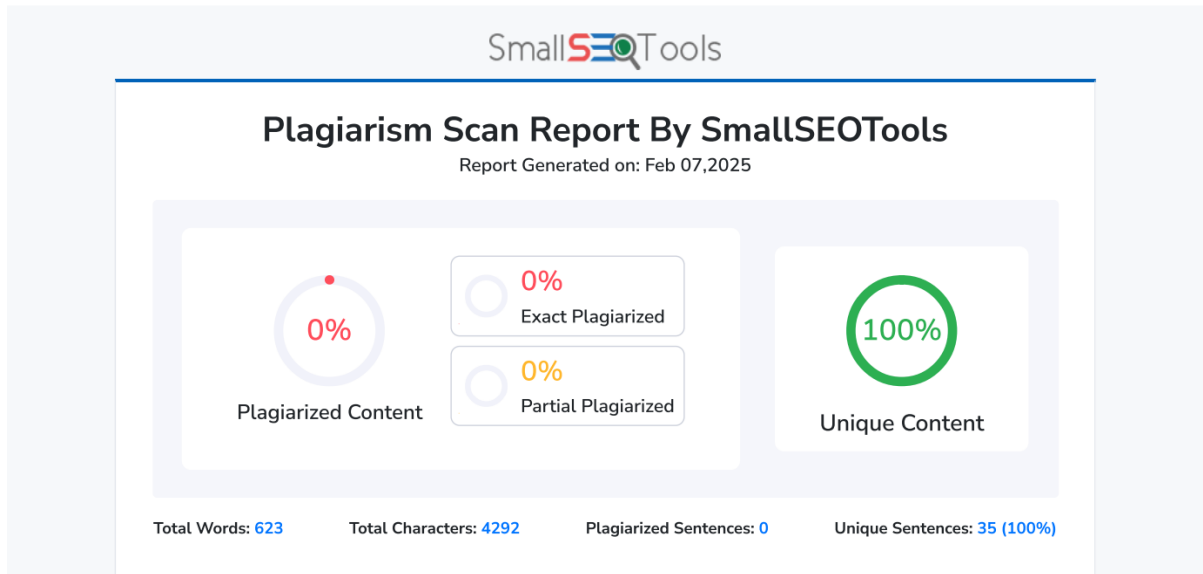
#### 6. **File Sharing:**

- Sharing files with other registered users on the platform.
- Basic sharing permissions (initially, maybe just "view" access).

#### 7. **User Interface (UI) and User Experience (UX):**

- A clean, intuitive, and responsive web interface built with React, Next.js, TailwindCSS, and ShadCN.
- Basic file browsing and management UI elements.

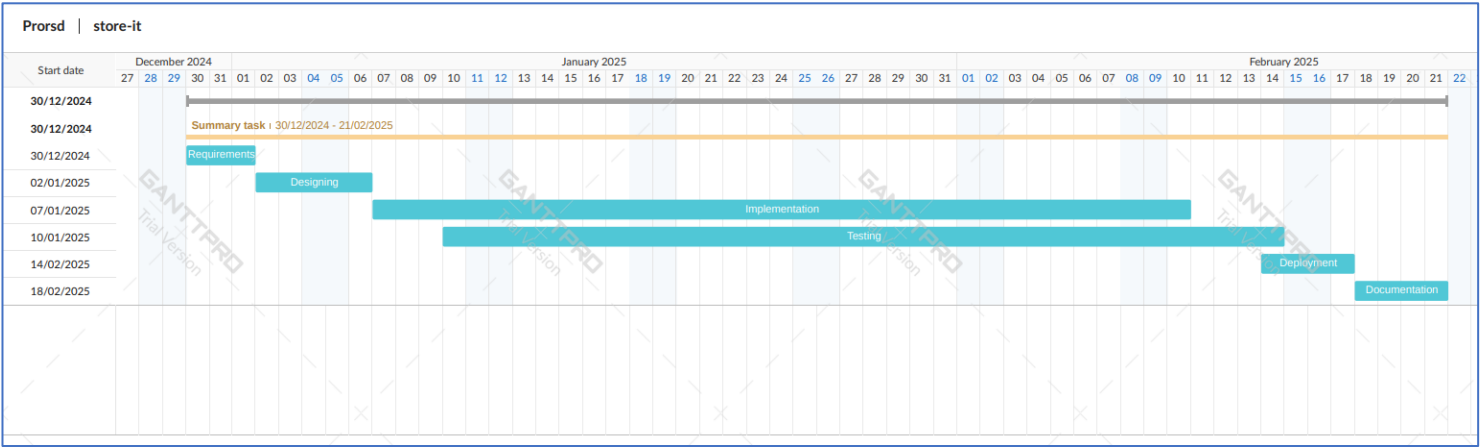
# PLAGIARISM REPORT



## Expected Outcomes:

1. **Efficient File Management** – Users can easily upload, rename, delete, and download files of any format.
2. **Seamless File Sharing** – Users can share files securely with others on the platform.
3. **Automatic Categorization** – Files will be organized intelligently, improving accessibility.
4. **User-Friendly Experience** – A modern, responsive, and intuitive UI with smooth navigation.
5. **Secure Storage & Access** – Authentication, role-based access, and encryption for data protection.

# GANTT CHART



# System Requirements:

These include the software and hardware needed for your app to run effectively.

## 1. Client-Side Requirements (For Users):

- **Browser:** Latest version of Chrome, Firefox, Edge, or Safari
- **Internet Connection:** Stable connection for seamless file upload/download
- **Device:**
  - PC/Laptop (Windows, macOS, Linux)
  - Mobile/Tablet (Responsive UI support)
- **Minimum RAM:** 2GB (4GB recommended for smooth performance)

## 2. Development Environment (For Developers)

- **Operating System:** Windows, macOS, or Linux
- **Node.js:** v18+ (for running Next.js and React)
- **Package Manager:** npm 9+ or pnpm/yarn
- **Appwrite Server (Optional for Local Development):** Docker installed for self-hosted backend
- **IDE:** VS Code, WebStorm, or any preferred code editor
- **Browser Developer Tools:** For debugging

## 3. Backend & Hosting Requirements (Server-Side)

- **Backend Service:** Appwrite (Self-hosted or Appwrite Cloud)
- **Storage:** Sufficient cloud storage for handling user files
- **Server:**
  - Minimum: 2 vCPUs, 4GB RAM (for self-hosted Appwrite)
  - Recommended: 4 vCPUs, 8GB RAM (for better performance)
- **Database:** Appwrite's built-in database (or external DB if needed)
- **Hosting:**
  - Vercel (Recommended for Next.js)
  - Alternative: AWS, DigitalOcean, or any cloud provider



# Functional Requirements:

These define what the app should do to meet user needs and expectations.

## 1. User Authentication & Authorization

- Users must be able to **register** and **log in** using email and OAuth.
- Users should be able to **reset passwords** and manage account settings.
- Role-based access should be implemented for **secure file sharing** (e.g., view-only, edit access).

## 2. File Management

- Users should be able to **upload files of any format** from their PC.
- Ability to **rename, delete, and download** uploaded files.
- **Automatic categorization** should classify files based on type (e.g., images, documents, videos).
- Display **file metadata** (size, format, upload date, etc.).

## 3. File Sharing & Collaboration

- Users should be able to **share files** with other registered users on the platform.

## 4. Search & Filter Functionality

- Users should be able to **search for files** by name, type, or date.
- Filters should allow sorting by **file size, format, upload date, and owner**.

## 5. User Dashboard & UI

- A responsive dashboard should display **uploaded files and sharing status**.
- Dark mode and **theme customization** options for user preferences.

## 6. Security & Data Protection

- Implement **secure file storage** using Appwrite's authentication and database.
- Encrypt files **during upload and storage** for data security.
- Only authorized users should access **protected files and personal data**.

## 7. Integration & Future Enhancements

- Support for **third-party storage services** (Google Drive, Dropbox, etc.).
- Implement **real-time collaboration** for shared files.
- Add **version control** to track changes in uploaded files.

# **Non-Functional Requirements:**

## **1. Performance:**

- The system should handle concurrent user requests efficiently.
- Web pages should load within an acceptable timeframe for a seamless user experience.

## **2. Security:**

- Implement secure authentication using JWT.
- Ensure data privacy and integrity during transmission and storage.
- Regularly update dependencies to patch security vulnerabilities.

## **3. Scalability:**

- Design the system to scale horizontally to accommodate an increasing number of users.
- Optimize database queries for efficient data retrieval as the user base grows.

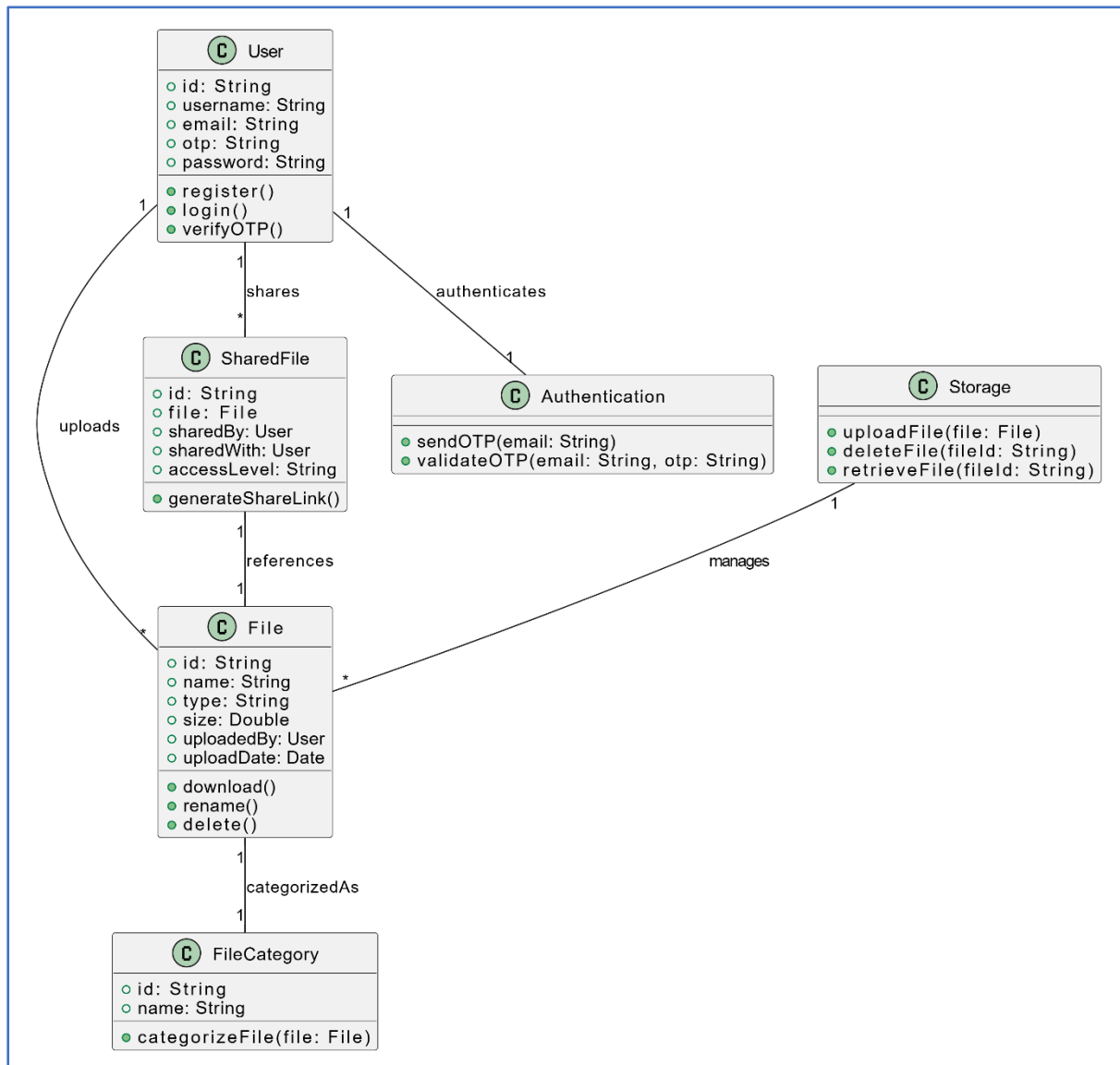
## **4. Usability:**

- The user interface should be intuitive and user-friendly for both teachers and students.
- Provide informative error messages to guide users in case of incorrect input or system errors.

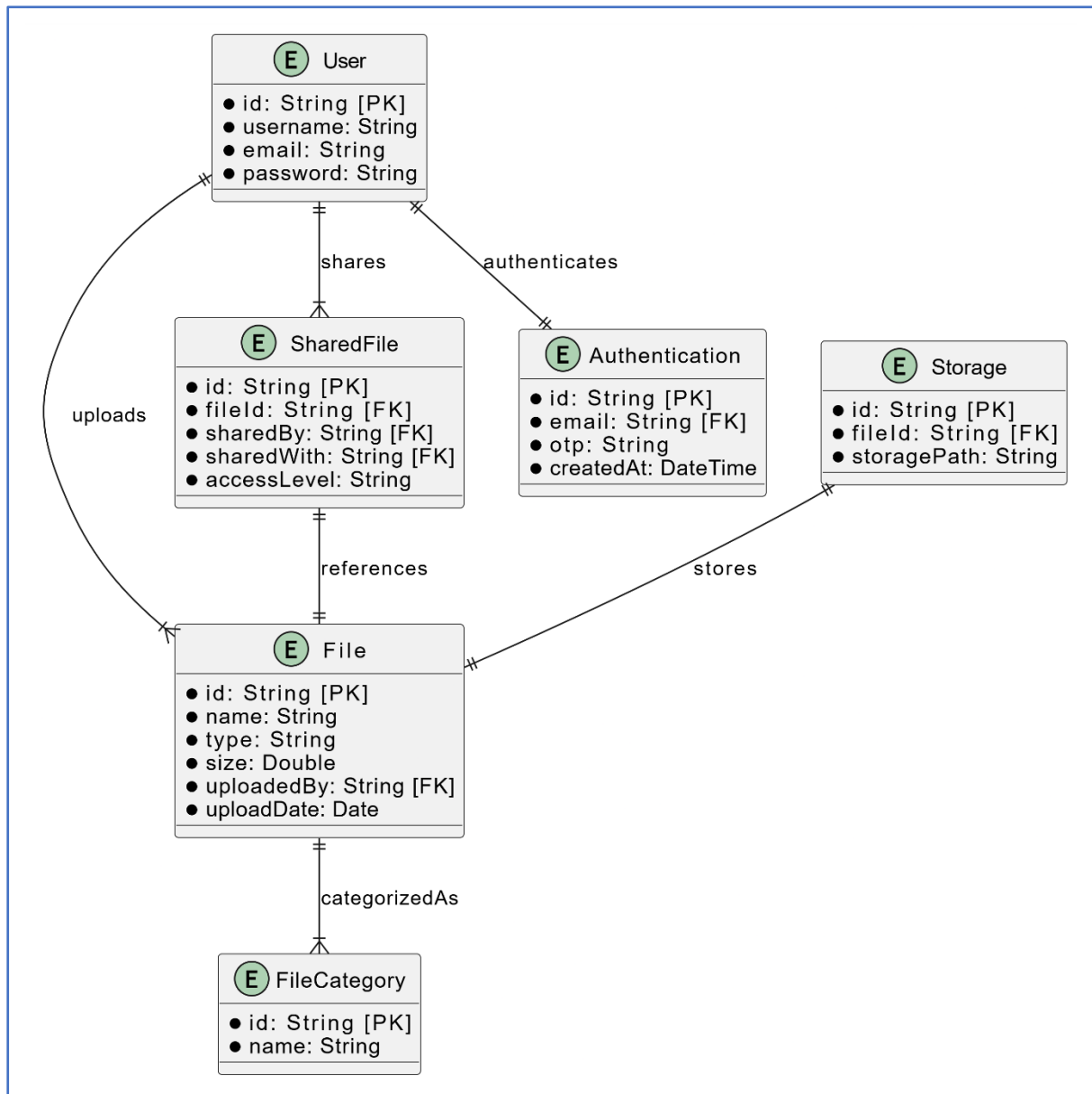
## **5. Reliability:**

- The system should be available and reliable, with minimal downtime for maintenance.

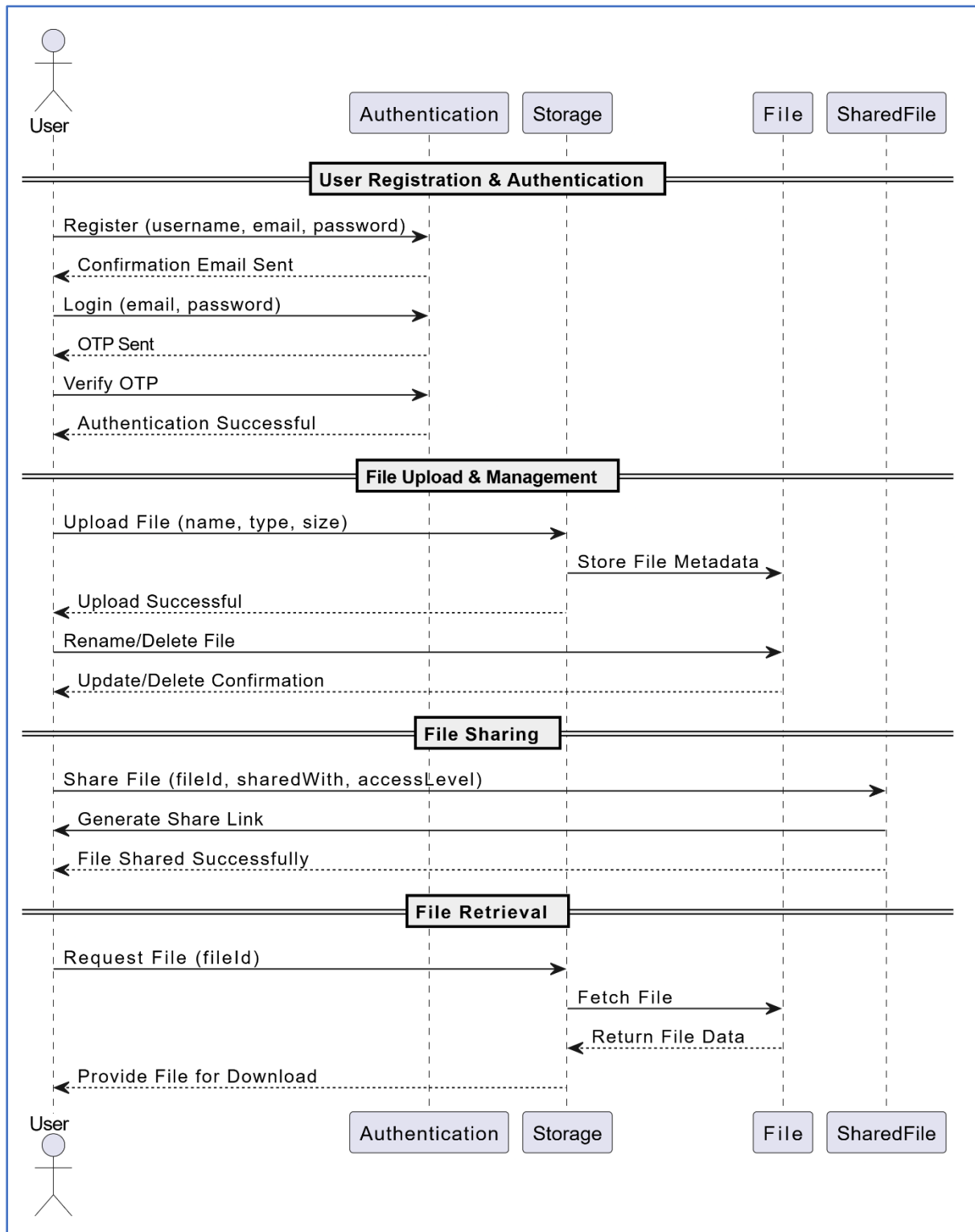
# CLASS DIAGRAM



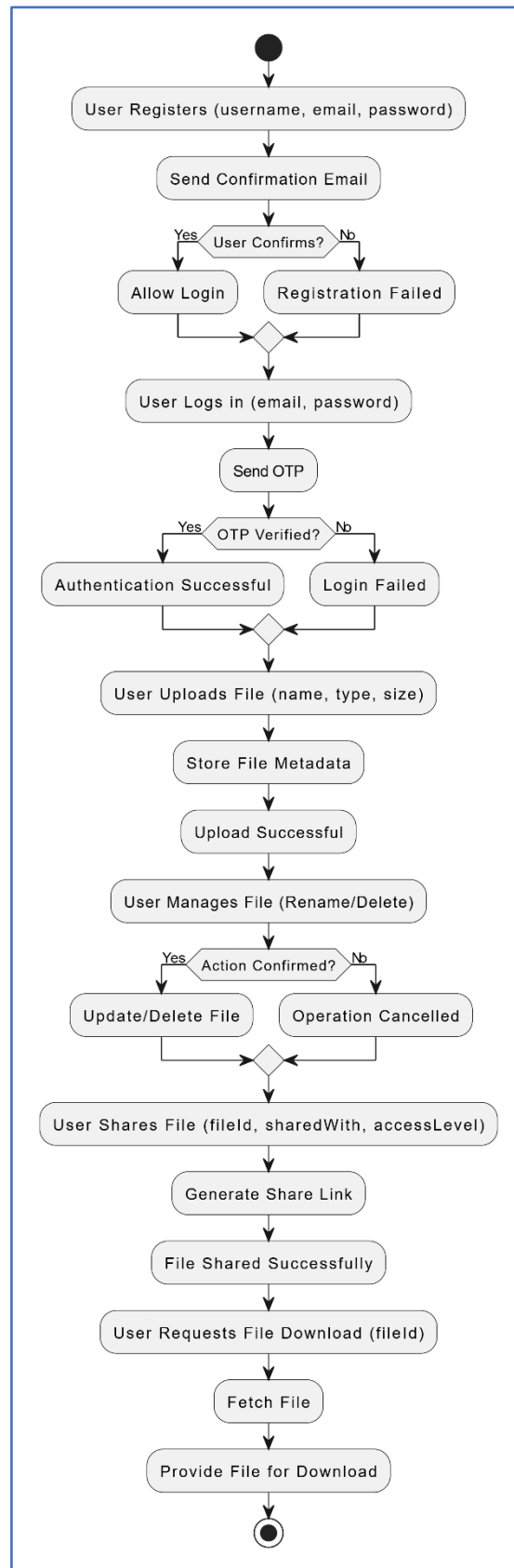
# ENTITY-RELATIONSHIP DIAGRAM



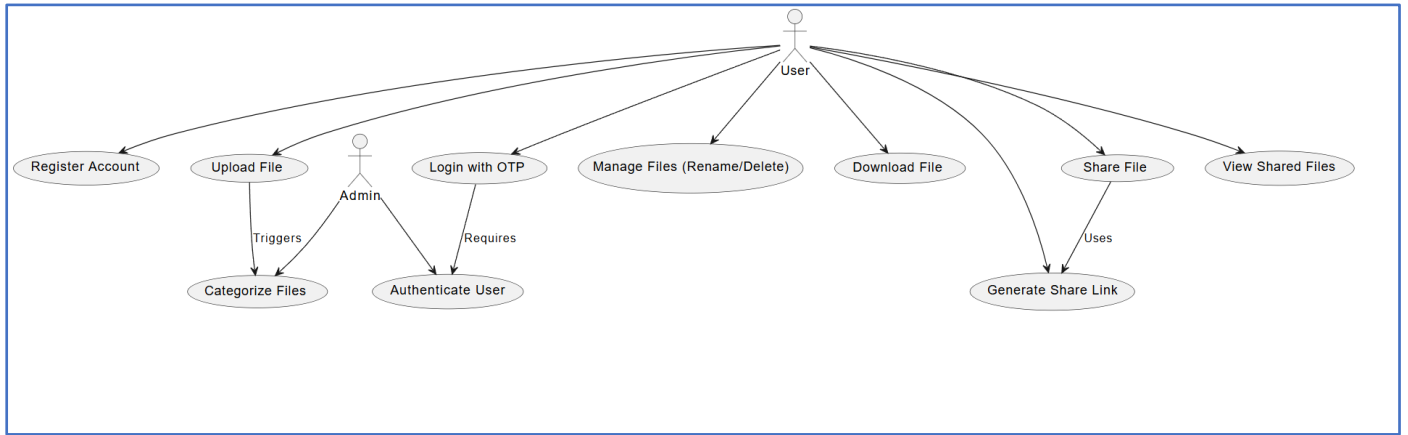
# SEQUENCE DIAGRAM



# FLOWCHART

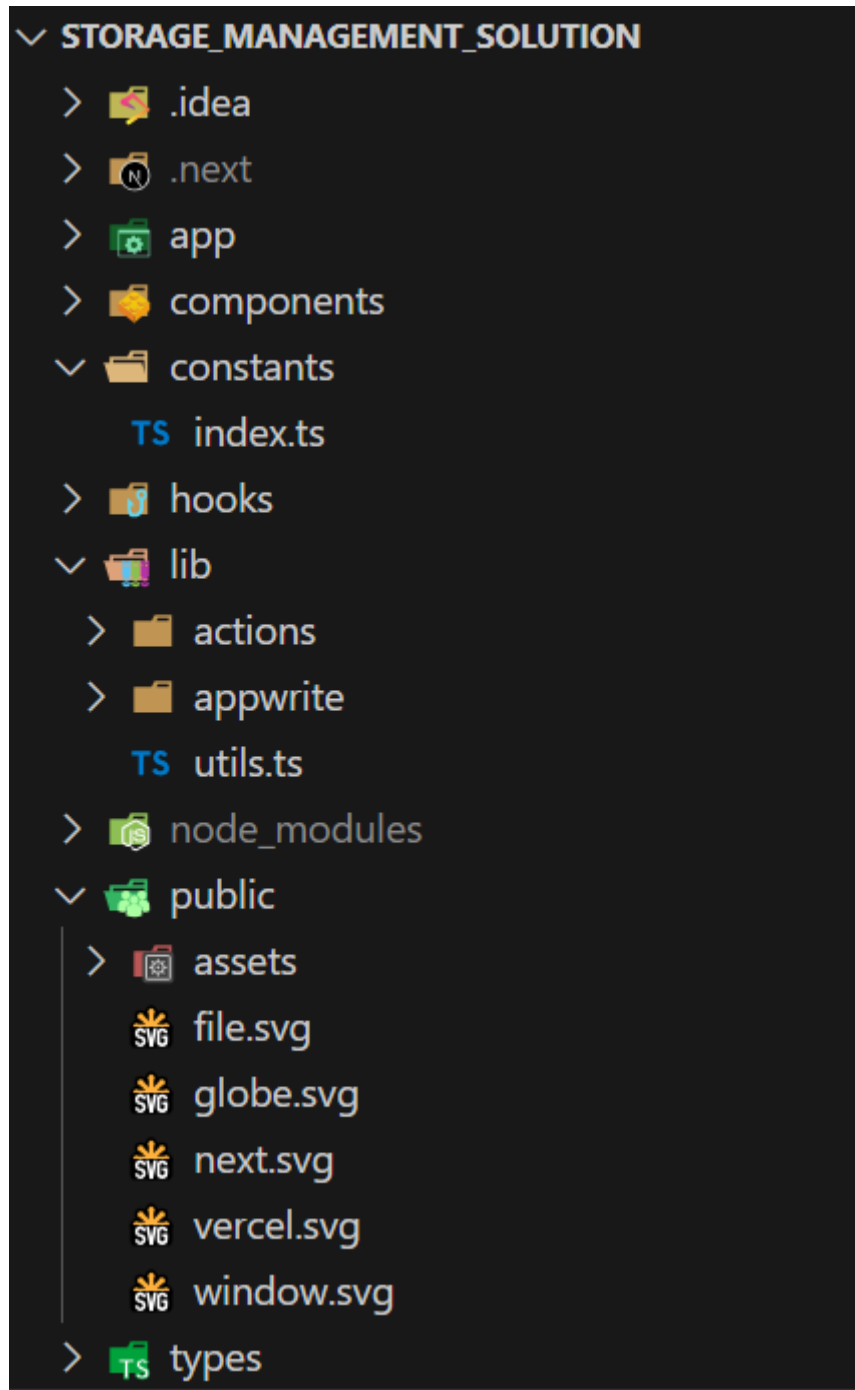


# USE CASE DIAGRAM



















# Folder structure

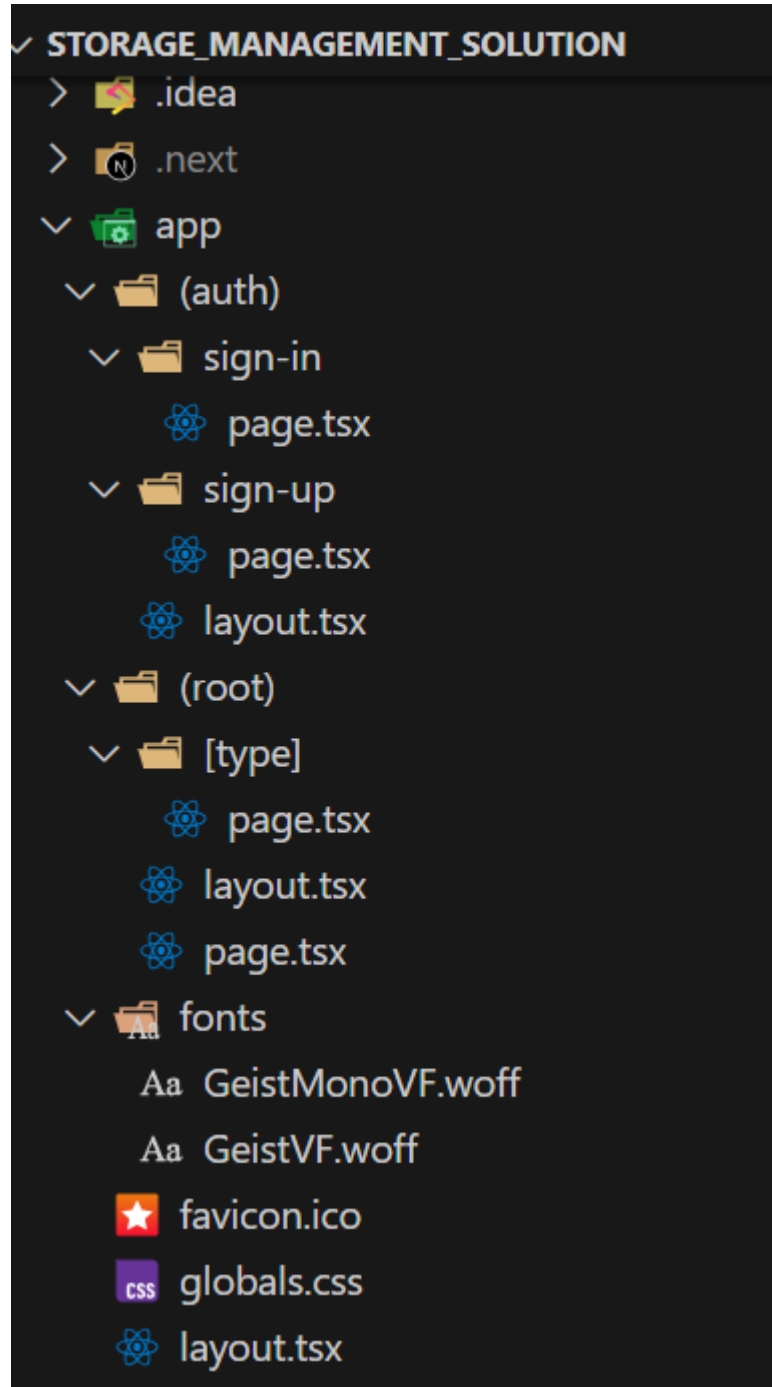
## 1.Backend

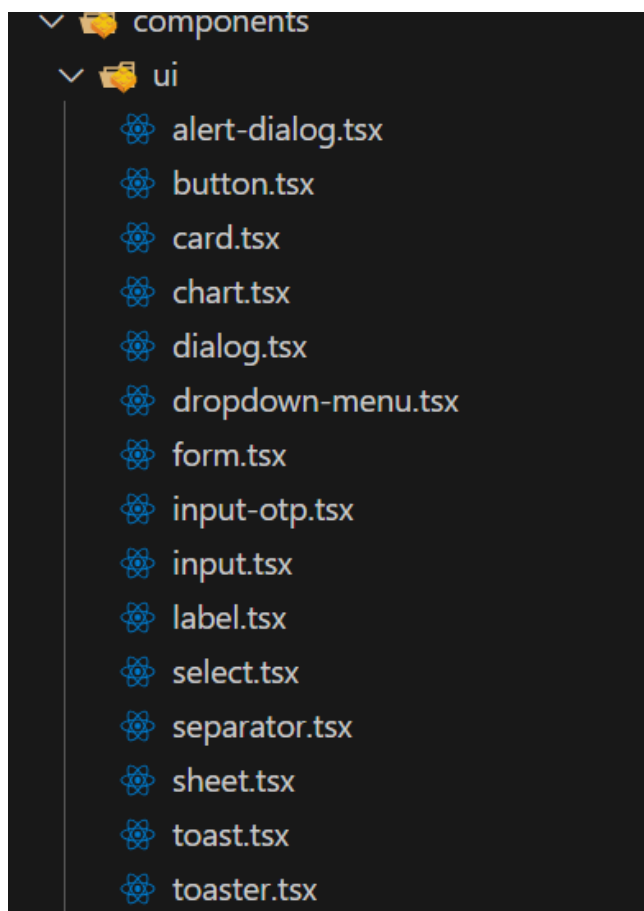
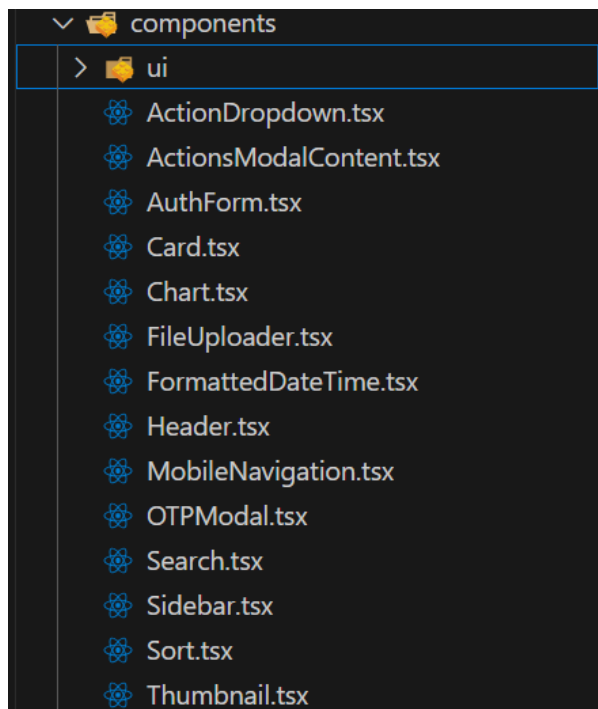




- >  app
- >  components
- ✓  constants
  - TS index.ts
- >  hooks
- ✓  lib
  - ✓  actions
    - TS file.actions.ts
    - TS user.actions.ts
  - >  appwrite
    - TS utils.ts
- >  node\_modules
- ✓  public
  - >  assets
    -  file.svg
    -  globe.svg
    -  next.svg
    -  vercel.svg
    -  window.svg
- ✓  types
  - TS index.d.ts

## 2. Frontend





## CODE SNIPPETS

### 1. Page.tsx

```
import Image from "next/image";
import Link from "next/link";
import { Models } from "node-appwrite";

import ActionDropdown from "@components/ActionDropdown";
import { Chart } from "@components/Chart";
import { FormattedDateTime } from "@components/FormattedDateTime";
import { Thumbnail } from "@components/Thumbnail";
import { Separator } from "@components/ui/separator";
import { getFiles, getTotalSpaceUsed } from "@lib/actions/file.actions";
import { convertFileSize, getUsageSummary } from "@lib/utlis";

const Dashboard = async () => {
  // Parallel requests
  const [files, totalSpace] = await Promise.all([
    getFiles({ types: [], limit: 10 }),
    getTotalSpaceUsed(),
  ]);

  // Get usage summary
  const usageSummary = getUsageSummary(totalSpace);

  return (
    <div className="dashboard-container">
      <section>
        <Chart used={totalSpace.used} />

        { /* Uploaded file type summaries */ }
```

```

<ul className="dashboard-summary-list">
  {usageSummary.map((summary) => (
    <Link
      href={summary.url}
      key={summary.title}
      className="dashboard-summary-card"
    >
      <div className="space-y-4">
        <div className="flex justify-between gap-3">
          <Image
            src={summary.icon}
            width={100}
            height={100}
            alt="uploaded image"
            className="summary-type-icon"
          />
          <h4 className="summary-type-size">
            {convertFileSize(summary.size) || 0}
          </h4>
        </div>

        <h5 className="summary-type-title">{summary.title}</h5>
        <Separator className="bg-light-400" />
        <FormattedDateTime
          date={summary.latestDate}
          className="text-center"
        />
      </div>
    </Link>
  )})
</ul>

```

```
</section>
```

```
{/* Recent files uploaded */}
```

```
<section className="dashboard-recent-files">
```

```
<h2 className="h3 xl:h2 text-light-100">Recent files uploaded</h2>
```

```
{files.documents.length > 0 ? (
```

```
<ul className="mt-5 flex flex-col gap-5">
```

```
{files.documents.map((file: Models.Document) => (
```

```
<Link
```

```
href={file.url}
```

```
target="_blank"
```

```
className="flex items-center gap-3"
```

```
key={file.$id}
```

```
>
```

```
<Thumbnail
```

```
type={file.type}
```

```
extension={file.extension}
```

```
url={file.url}
```

```
/>
```

```
<div className="recent-file-details">
```

```
<div className="flex flex-col gap-1">
```

```
<p className="recent-file-name">{file.name}</p>
```

```
<FormattedDateTime
```

```
date={file.$createdAt}
```

```
className="caption"
```

```
/>
```

```
</div>
```

```
<ActionDropdown file={file} />
```

```
</div>
```

```
</Link>
```

```

    )}
  </ul>
) : (
  <p className="empty-list">No files uploaded</p>
)
</section>
</div>
);
};

export default Dashboard;

```

## 2. Layout.tsx

```

import React from "react";
import Sidebar from "@components/Sidebar";
import MobileNavigation from "@components/MobileNavigation";
import Header from "@components/Header";
import { getCurrentUser } from "@lib/actions/user.actions";
import { redirect } from "next/navigation";
import { Toaster } from "@components/ui/toaster";

export const dynamic = "force-dynamic";

const Layout = async ({ children }: { children: React.ReactNode }) => {
  const currentUser = await getCurrentUser();

  if (!currentUser) return redirect("/sign-in");

  return (
    <main className="flex h-screen">

```

```

<Sidebar {...currentUser} />

<section className="flex h-full flex-1 flex-col">
  <MobileNavigation {...currentUser} />
  <Header userId={currentUser.$id} accountId={currentUser.accountId} />
  <div className="main-content">{children}</div>
</section>

<Toaster />
</main>

);
};

export default Layout;

```

### 3. AuthForm.tsx

```

"use client";

import { z } from "zod";
import { zodResolver } from "@hookform/resolvers/zod";
import { useForm } from "react-hook-form";

import { Button } from "@/components/ui/button";
import {
  Form,
  FormControl,
  FormField,
  FormItem,
  FormLabel,
  FormMessage,
} from "@/components/ui/form";

```



```

import { Input } from "@/components/ui/input";
import { useState } from "react";
import Image from "next/image";
import Link from "next/link";
import { createAccount, signInUser } from "@/lib/actions/user.actions";
import OtpModal from "@/components/OTPModal";

```

```

type FormType = "sign-in" | "sign-up";

```

```

const authFormSchema = (formType: FormType) => {
  return z.object({
    email: z.string().email(),
    fullName:
      formType === "sign-up"
        ? z.string().min(2).max(50)
        : z.string().optional(),
  });
};

```

```

const AuthForm = ({ type }: { type: FormType }) => {
  const [isLoading, setIsLoading] = useState(false);
  const [errorMessage, setErrorMessage] = useState("");
  const [accountId, setAccountId] = useState(null);

```

```

  const formSchema = authFormSchema(type);
  const form = useForm<z.infer<typeof formSchema>>({
    resolver: zodResolver(formSchema),
    defaultValues: {
      fullName: "",
      email: "",
    },
  },

```

```

});

const onSubmit = async (values: z.infer<typeof formSchema>) => {
  setIsLoading(true);
  setErrorMessage("");

  try {
    const user =
      type === "sign-up"
        ? await createAccount({
            fullName: values.fullName || "",
            email: values.email,
          })
        : await signInUser({ email: values.email });

    setAccountId(user.accountId);
  } catch {
    setErrorMessage("Failed to create account. Please try again.");
  } finally {
    setIsLoading(false);
  }
};

return (
  <>
    <Form {...form}>
      <form onSubmit={form.handleSubmit(onSubmit)} className="auth-form">
        <h1 className="form-title">
          {type === "sign-in" ? "Sign In" : "Sign Up"}
        </h1>
        {type === "sign-up" && (

```

```

<FormField
  control={form.control}
  name="fullName"
  render={({ field }) => (
    <FormItem>
      <div className="shad-form-item">
        <FormLabel className="shad-form-label">Full Name</FormLabel>

        <FormControl>
          <Input
            placeholder="Enter your full name"
            className="shad-input"
            {...field}
          />
        </FormControl>
      </div>

      <FormMessage className="shad-form-message" />
    </FormItem>
  )}
/>
)}

```

```

<FormField
  control={form.control}
  name="email"
  render={({ field }) => (
    <FormItem>
      <div className="shad-form-item">
        <FormLabel className="shad-form-label">Email</FormLabel>

```

```

    <FormControl>
      <Input
        placeholder="Enter your email"
        className="shad-input"
        {...field}
      />
    </FormControl>
  </div>

  <FormMessage className="shad-form-message" />
</FormItem>
)}
/>

<Button
  type="submit"
  className="form-submit-button"
  disabled={isLoading}
>
  {type === "sign-in" ? "Sign In" : "Sign Up"}

  {isLoading && (
    <Image
      src="/assets/icons/loader.svg"
      alt="loader"
      width={24}
      height={24}
      className="ml-2 animate-spin"
    />
  )}
</Button>

```

```

{errorMessage && <p className="error-message">{*errorMessage}</p>}

<div className="body-2 flex justify-center">
  <p className="text-light-100">
    {type === "sign-in"
      ? "Don't have an account?"
      : "Already have an account?"}
  </p>
  <Link
    href={type === "sign-in" ? "/sign-up" : "/sign-in"}
    className="ml-1 font-medium text-brand"
  >
    {" "}
    {type === "sign-in" ? "Sign Up" : "Sign In"}
  </Link>
</div>
</form>
</Form>

{accountId && (
  <OtpModal email={form.getValues("email")} accountId={accountId} />
)}
</>

);
};

export default AuthForm;

```

## User Defined Data types

```
declare type FileType = "document" | "image" | "video" | "audio" | "other";
```

```
declare interface ActionType {  
  label: string;  
  icon: string;  
  value: string;  
}
```

```
declare interface SearchParamProps {  
  params?: Promise<SegmentParams>;  
  searchParams?: Promise<{ [key: string]: string | string[] | undefined }>;  
}
```

```
declare interface UploadFileProps {  
  file: File;  
  ownerId: string;  
  accountId: string;  
  path: string;  
}
```

```
declare interface GetFilesProps {  
  types: FileType[];  
  searchText?: string;  
  sort?: string;  
  limit?: number;  
}
```

```
declare interface RenameFileProps {  
  fileId: string;  
  name: string;  
  extension: string;
```

```
    path: string;
}

declare interface UpdateFileUsersProps {
    fileId: string;
    emails: string[];
    path: string;
}

declare interface DeleteFileProps {
    fileId: string;
    bucketField: string;
    path: string;
}

declare interface FileUploaderProps {
    ownerId: string;
    accountId: string;
    className?: string;
}

declare interface MobileNavigationProps {
    ownerId: string;
    accountId: string;
    fullName: string;
    avatar: string;
    email: string;
}

declare interface SidebarProps {
    fullName: string;
    avatar: string;
    email: string;
}
```

```

declare interface ThumbnailProps {

  type: string;

  extension: string;

  url: string;

  className?: string;

  imageClassName?: string;

}

```

```

declare interface ShareInputProps {

  file: Models.Document;

  onChange: (e: React.ChangeEvent<HTMLInputElement>) => void;

  onRemove: (email: string) => void;

}

```

## DEPLOYMENT

### a) Git and Github Setup

#### 1. Installing Git and GitHub CLI:

- a. Download and installing Git from official website.
- b. Installing GitHub CLI

#### 2. Command Line Setup (CMD) - Git Commands:

- Open a command prompt and configuring Git identity: git config --global user.name "Pratham Parmar"
- git config --global user.email "prathamparamar@gmail.com"
- Initialize a new Git repository in your project folder: git init
- Adding files to the staging area: git add .
- Commit the changes: git commit -m "Initial commit"
- Creating a new repository on GitHub.
- adding the remote

repository and push the code:

git remote add origin branch -M



master

git push -u origin master

## **b) Hosting**

### **1. Environment Variables for Database and API URL:**

Add your database connection string :

NEXT\_PUBLIC\_APPWRITE\_ENDPOINT="https://cloud.appwrite.io/v1"

NEXT\_PUBLIC\_APPWRITE\_PROJECT="YOUR PROJECTID"

NEXT\_PUBLIC\_APPWRITE\_DATABASE="YOUR DATABASE ID"

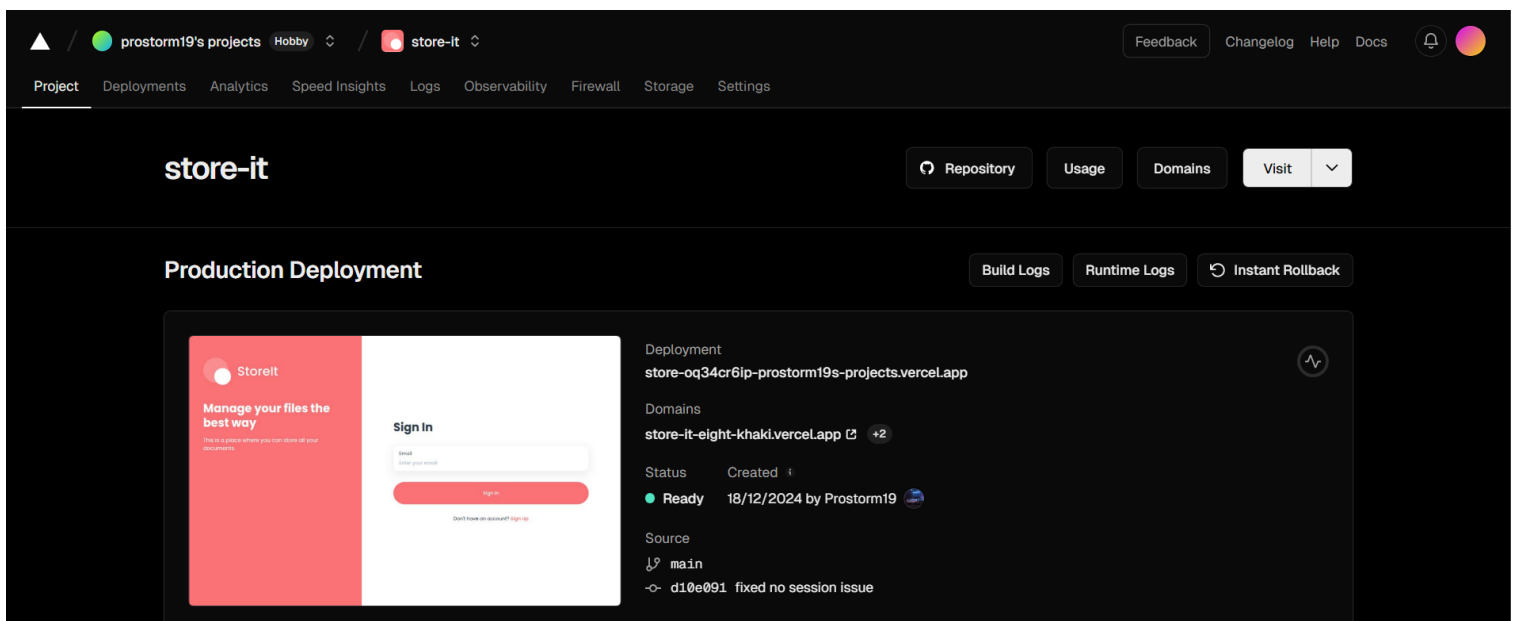
NEXT\_PUBLIC\_APPWRITE\_USERS\_COLLECTION="USER COLLECTION ID"

NEXT\_PUBLIC\_APPWRITE\_FILES\_COLLECTION="FILES COLELCTION ID"

NEXT\_PUBLIC\_APPWRITE\_BUCKET="STORAGE BUCKET ID"

NEXT\_APPWRITE\_KEY="YOUR SECRET API KEY"

Accessing Frontend: Open your browser and navigate to the specified localhost or IP where your frontend is hosted.



# DATABASE OVERVIEW

appwrite

BETA

/ Project / Jsm\_storeit / Databases

Upgrade

Feedback

Support

Q

PR Pratham Rajesh Parmar Project

Overview

Auth

Databases

Functions

Messaging

Storage

Databases

Usage

Databases 1/1 created

Columns 5

+ Create database

DATABASE ID	NAME	BACKUPS	CREATED	UPDATED
67664840902cc9f23ef6	general	No backup policies	Dec 16, 2024, 21:03	Dec 16, 2024, 21:03

6 Databases per page. Total results: 1

< Prev 1 Next >

appwrite

BETA

/ Project / Jsm\_storeit / Databases / General

Upgrade

Feedback

Support

Q

PR Pratham Rajesh Parmar Project

Overview

Auth

Databases

Functions

Messaging

Storage

< general 67664840902cc9f23ef6

Collections

Backups

Usage

Settings

Collections

Columns 4

+ Create collection

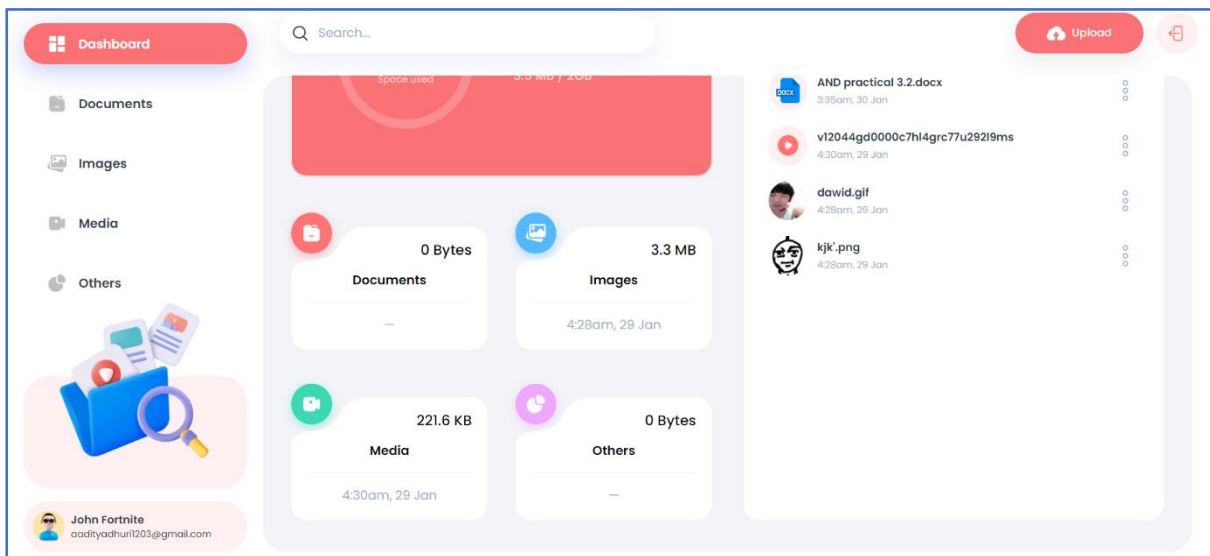
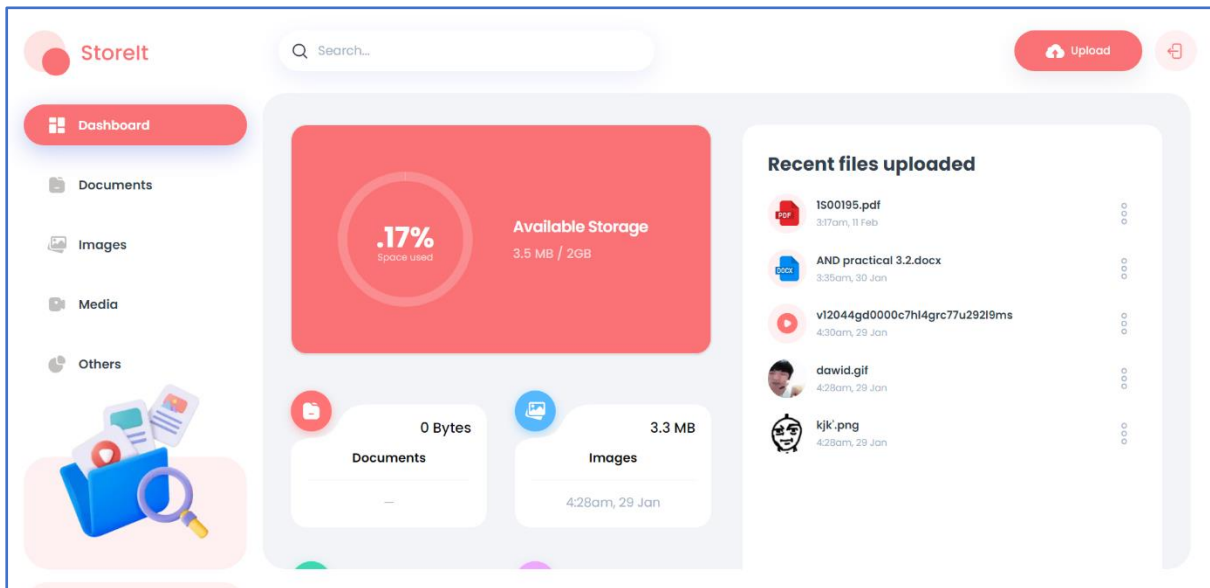
	COLLECTION ID	NAME	CREATED	UPDATED
<input type="checkbox"/>	67694e8906293adcbe82	files	Dec 16, 2024, 21:30	Dec 18, 2024, 19:05
<input type="checkbox"/>	6768485f062ac7381030	users	Dec 16, 2024, 21:03	Dec 16, 2024, 21:28

6 Collections per page. Total results: 2

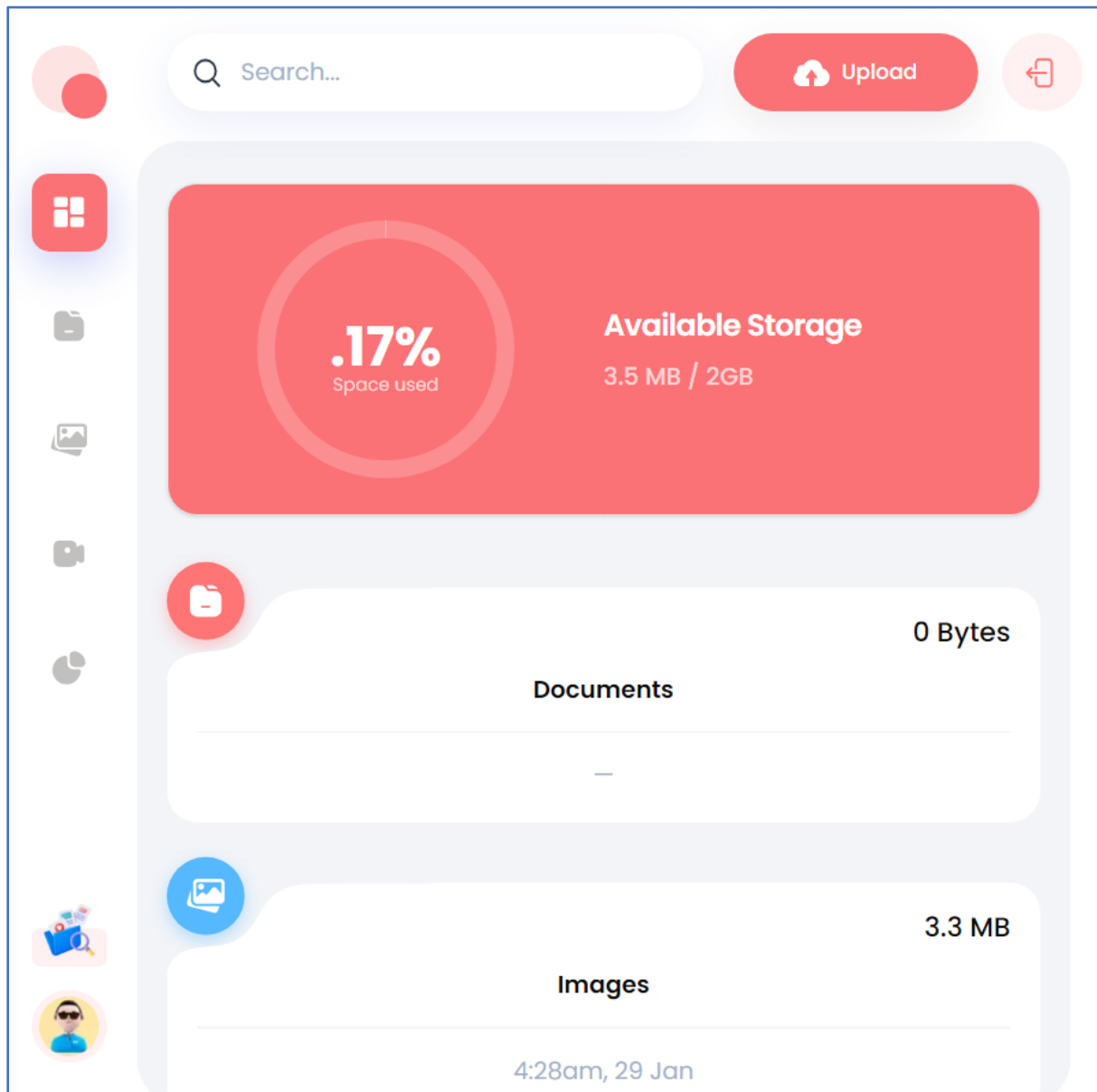
< Prev 1 Next >

# RESULTS

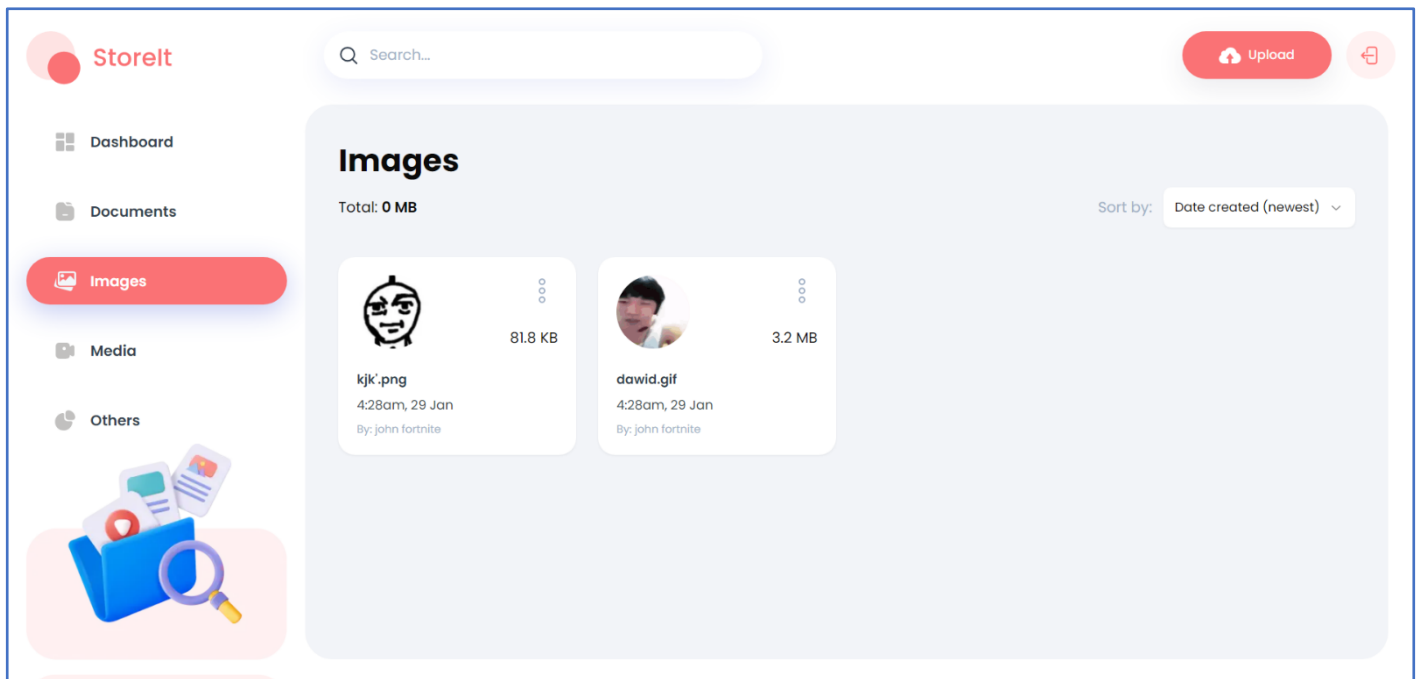
## 1. Home Page on Desktop



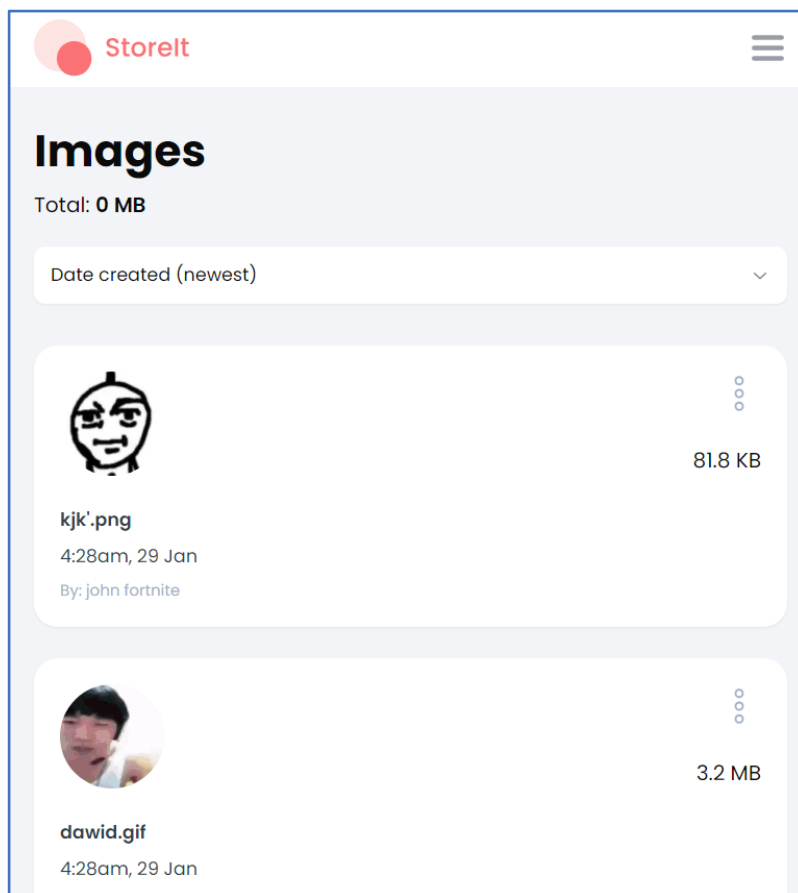
## 2. Home Page on Mobile



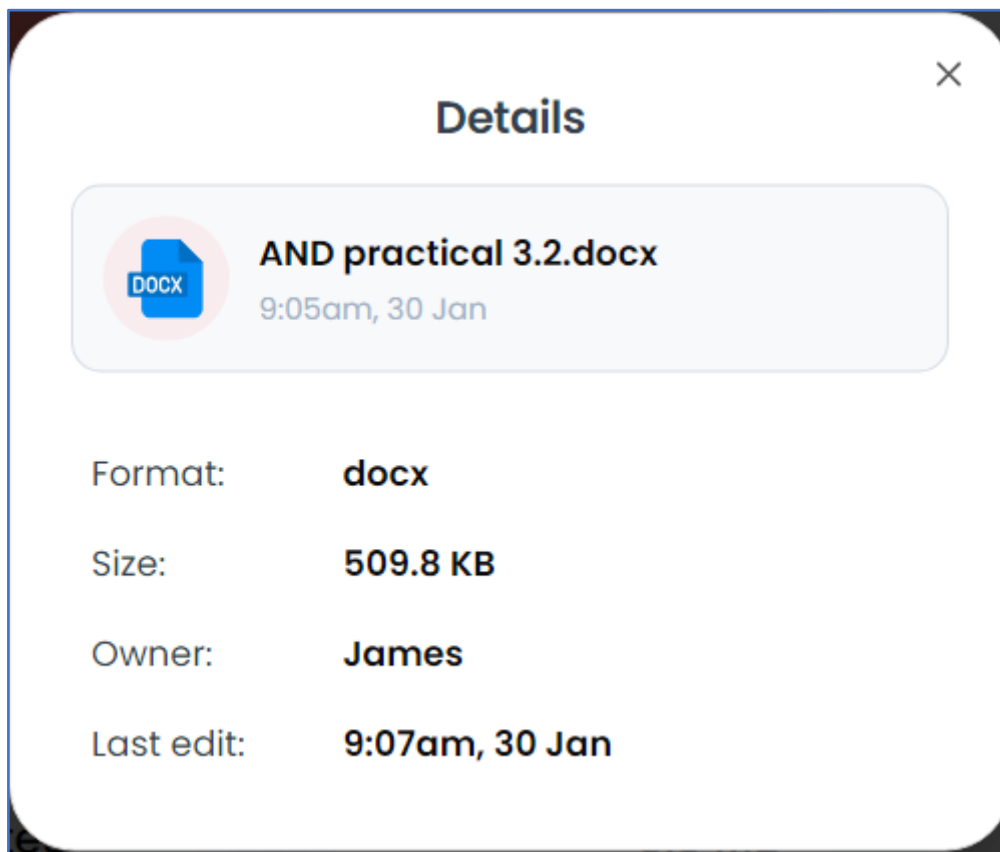
### 3. Category Page on Desktop



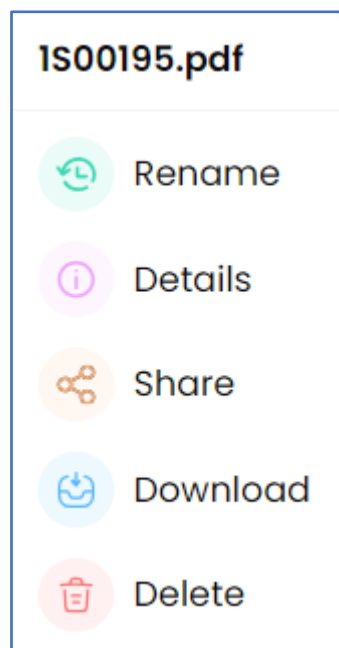
### 4. Category Page on Mobile




## 5. File Details View



## 6. File Edit and Properties View




## 7. Sign-up Page desktop



# StoreIt

## Manage your files the best way

This is a place where you can store all your documents.



### Sign Up

Full Name

Enter your full name

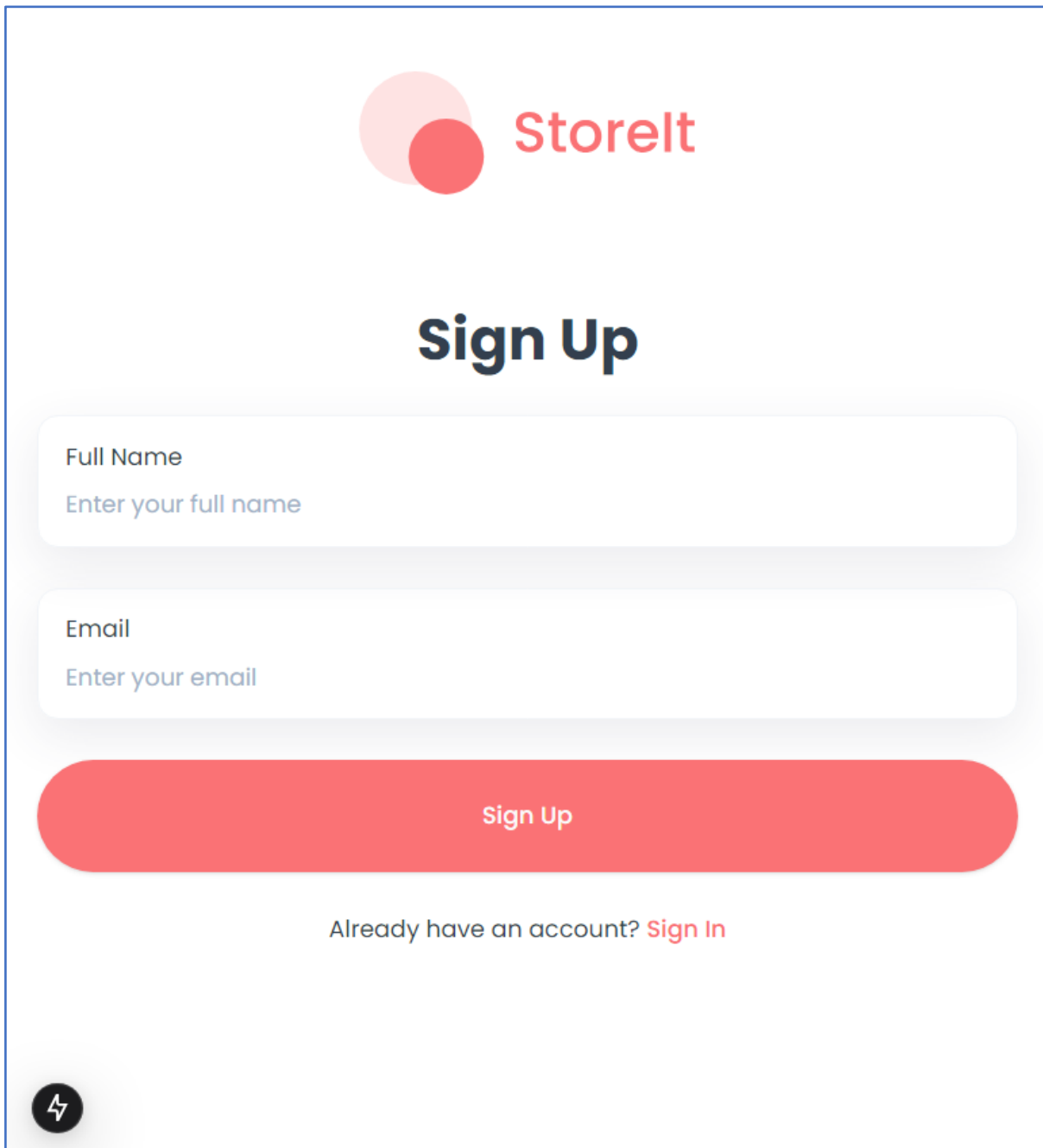
Email

Enter your email

Sign Up

Already have an account? [Sign In](#)

## 8. Sign-up Page on Mobile



The image shows a mobile app sign-up screen for 'Storelt'. At the top, there is a logo consisting of two overlapping red circles of different shades, followed by the text 'Storelt' in a red, sans-serif font. Below the logo, the title 'Sign Up' is displayed in a large, bold, dark blue font. There are two input fields: the first is labeled 'Full Name' with a light blue placeholder text 'Enter your full name'; the second is labeled 'Email' with a light blue placeholder text 'Enter your email'. Below these fields is a large, rounded red button with the text 'Sign Up' in white. Underneath the button, there is a link that says 'Already have an account? Sign In', where 'Sign In' is in red. In the bottom left corner, there is a small black circular icon containing a white lightning bolt symbol.

Storelt


# Sign Up

Full Name  
Enter your full name

Email  
Enter your email

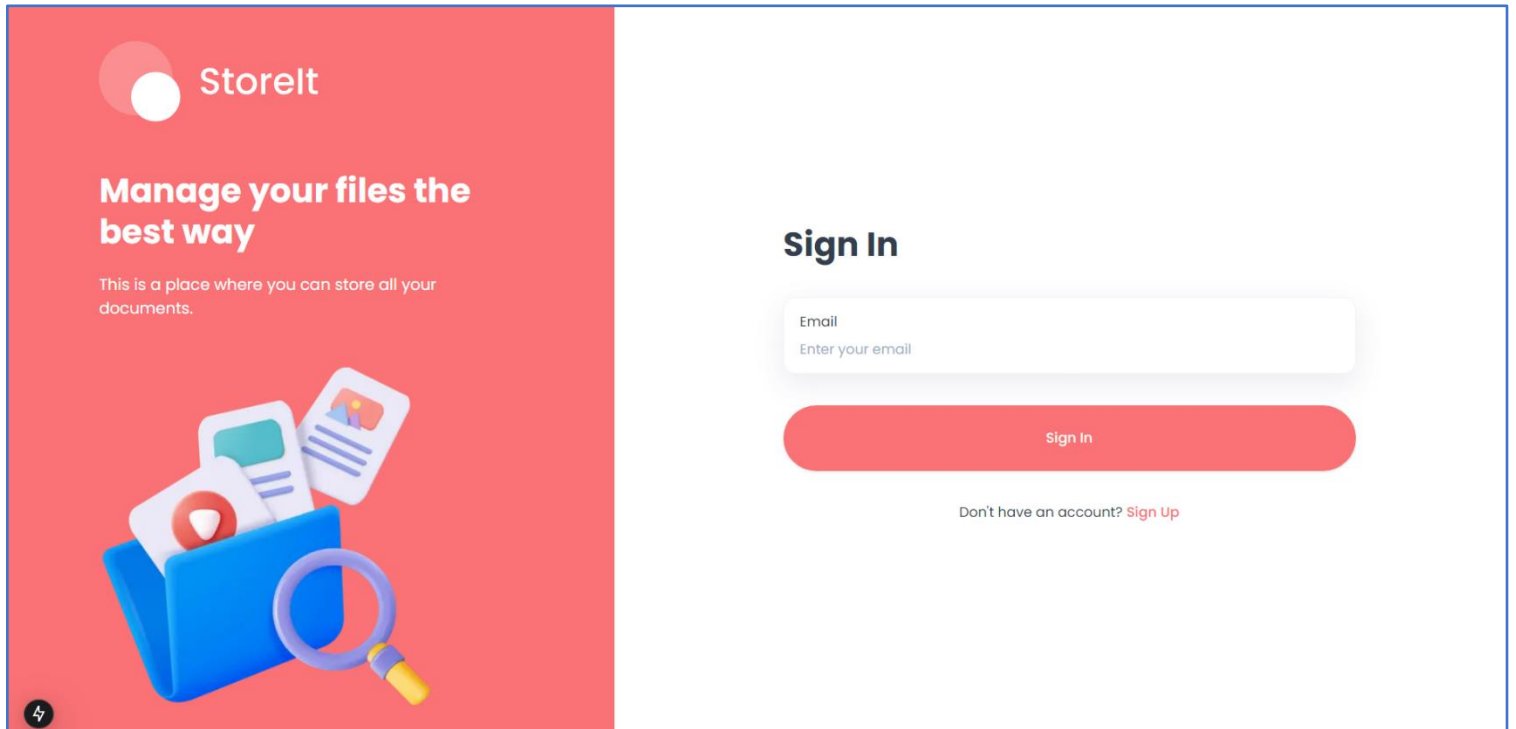
Sign Up

Already have an account? [Sign In](#)

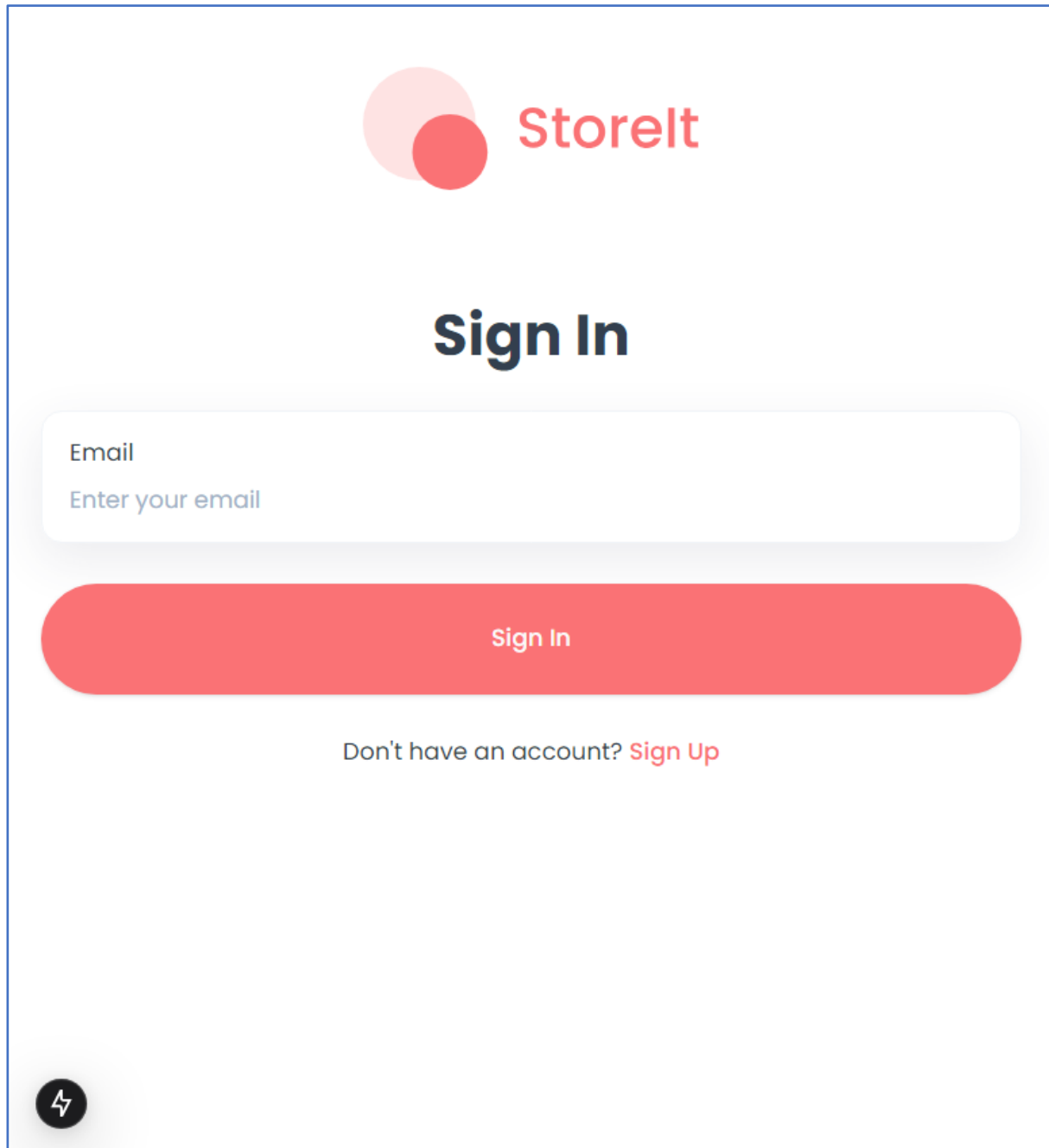




## 9. Sign-in Page on Desktop



## 10. Sign-In Page on Mobile



The image shows a mobile sign-in page for 'Storelt'. At the top, the logo consists of two overlapping circles (one light red, one dark red) followed by the text 'Storelt' in a red sans-serif font. Below the logo is the heading 'Sign In' in a large, bold, dark blue font. Underneath is a white rounded rectangular input field with the label 'Email' and the placeholder text 'Enter your email' in a light blue font. Below the input field is a large, rounded red button with the text 'Sign In' in white. Under the button is the text 'Don't have an account?' followed by a red link 'Sign Up'. In the bottom left corner, there is a small black circular icon containing a white lightning bolt symbol.


Storelt

# Sign In

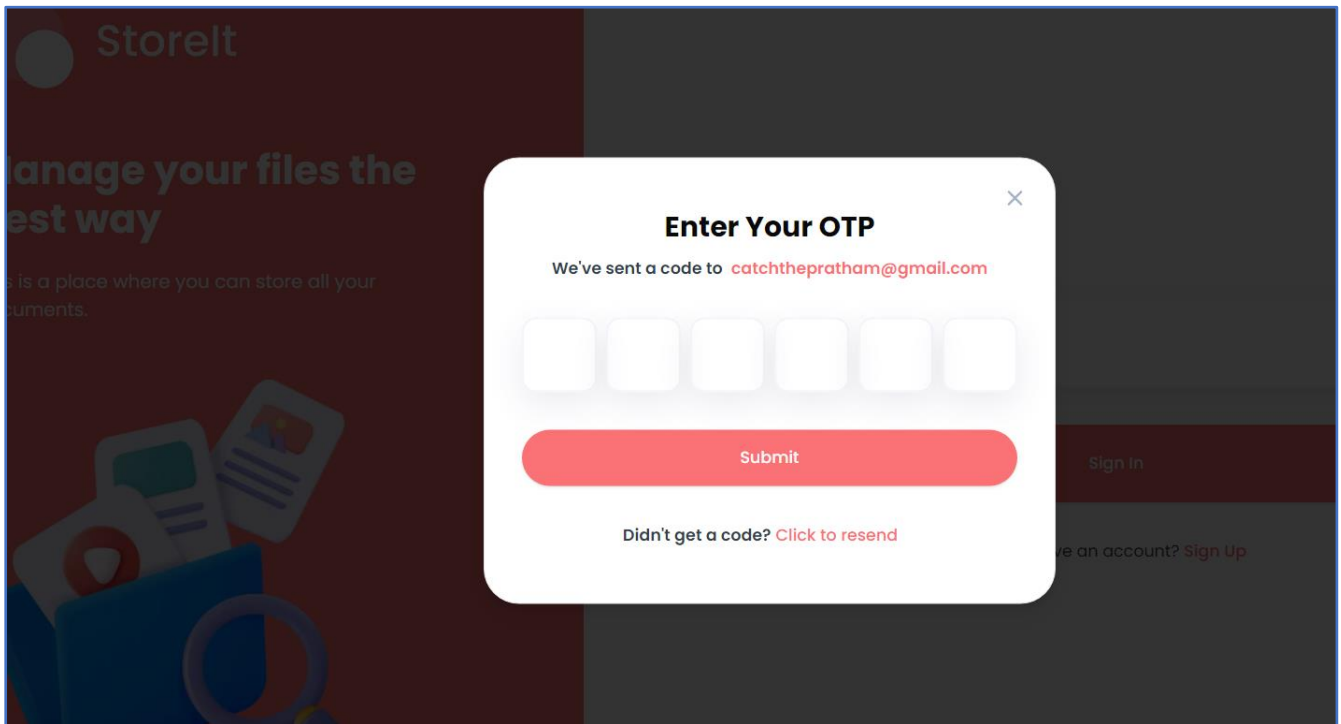
Email  
Enter your email

Sign In

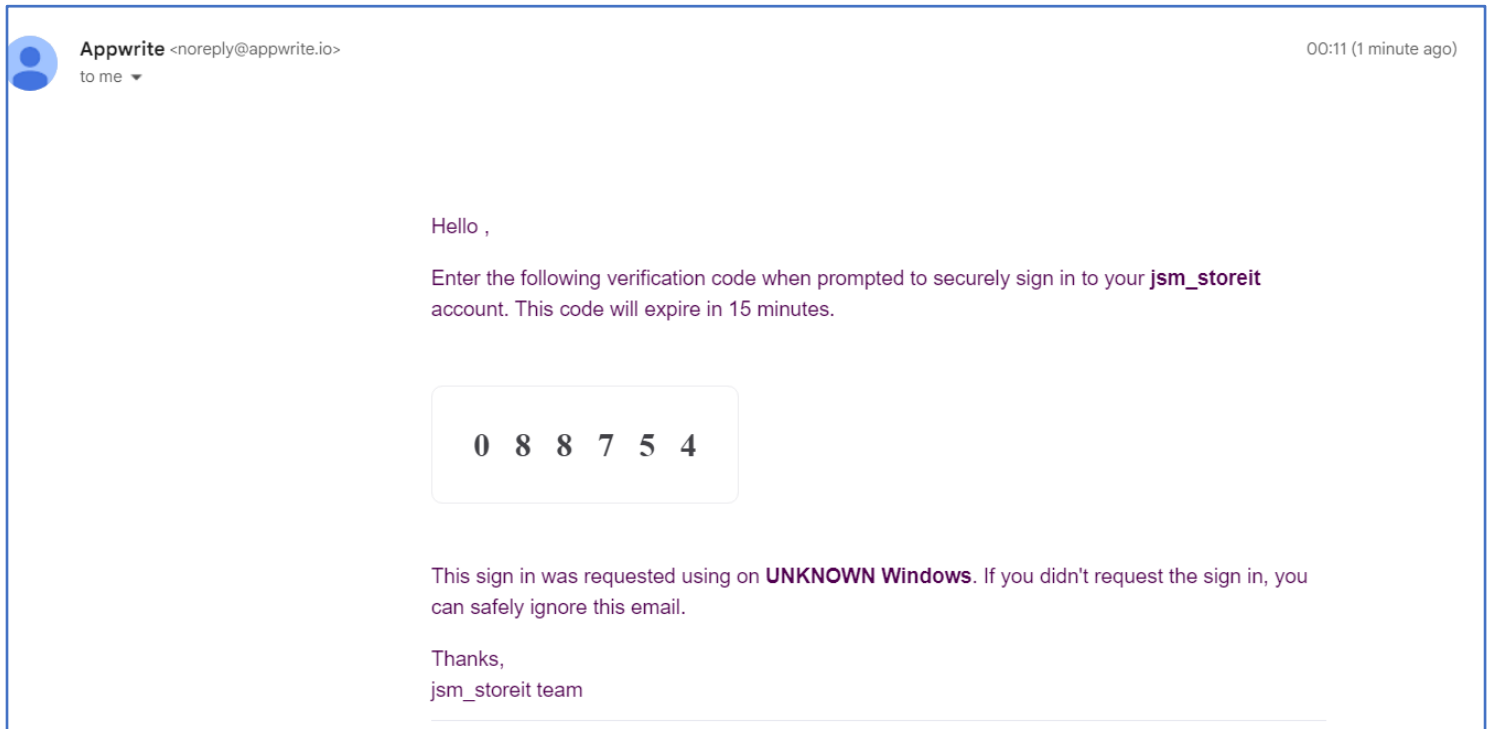
Don't have an account? [Sign Up](#)



## 11. OTP Pop-up Dialog Box after clicking on Sign-in/Sign-up



## 12. OTP in mail



## **REFERENCES**

- a) [GitHub - Prostorm19/store\\_it](#)
- b) <https://cloud.appwrite.io/>
- c) <https://figma.com/>
- d) <https://ui.shadcn.com/>
- e) <https://nextjs.org/docs/getting-started/installation>
- f) <https://nextjs.org/docs/app/building-your-application/routing>
- g) <https://tailwindcss.com/>