योगः कर्मसु कौशलम्
IN PURSUIT OF PERFECTION

Vivekananda Institute of Professional Studies - Technical Campus

Minor Project

DVAULT

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Computer Application

Submitted To: -
Ms. Mitanshi Rustagi

Assistant Professor

Submitted By :-
Divanshu Soni
03129802022

ACKNOWLEDGEMENT

I am deeply grateful to the management and faculty of Vivekananda Institute of Professional Studies for providing me with an exceptional learning experience. The institute's state-of-the-art facilities and dedicated faculty have created an environment that fosters intellectual growth and innovation.

I am especially grateful to Ms. Mitanshi Rustagi for her invaluable guidance and support. Her expertise and encouragement were crucial to the project's success.

I would also like to thank the numerous authors whose insightful writings have enriched my understanding of different technologies I had to work through.

CERTIFICATE FROM THE GUIDE

This is to certify that the project entitled "**DVAULT**" submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Applications to Vivekananda Institute of Professional Studies – Technical Campus through VSIT, is an authentic work carried out by Mr. Divanshu Soni, Roll No. 03129802022, under my guidance.

The matter embodied in this project has not been submitted earlier for the award of any degree, to the best of my knowledge and belief.

Signature of the Guide

Signature of the Student

Table of Contents

Synopsis of the Project

Introduction

-based storage solution for all your images, files, and documents. Easily keep everything in one place and share them with friends using expirable links. Enjoy effortless data management Web with Cloud Box!

Problem Statement:

In today's digital landscape, the need for seamless file sharing is undeniable. However, traditional methods often pose challenges, leaving users grappling with storage limitations and security concerns. Current sharing practices can be cumbersome, lacking an efficient system to manage shared content, resulting in confusion and data clutter. Addressing these issues, our project, DVAULT, offers a web-based storage solution designed to alleviate storage constraints, enhance security, and streamline the management of shared files, providing users with a user-friendly and efficient platform for their data management needs

Key Functionalities

- **SmartShare with Expirable URLs:**

Effortlessly share data using SmartShare, a feature that enables users to generate expirable URLs. This ensures secure and temporary access, allowing for controlled sharing with added peace of mind.

- **API Support:**

Enjoy flexibility and integration possibilities with DVAULT's API support. Connect seamlessly with other applications and services, extending the functionality of DVAULT to suit your unique workflow and collaboration needs.

- **Robust Security and Privacy Measures:**

Prioritize the safety of your data with DVAULT's robust security and privacy features. Benefit from end-to-end encryption, secure access controls, and regular security updates to ensure your files remain confidential and protected.

- **Intuitive User-Friendly Interface:**

Experience a smooth and intuitive user interface designed for simplicity and efficiency. Navigate through DVAULT effortlessly, with clear and user-friendly features that empower you to manage your data seamlessly, making your storage and sharing tasks a breeze.

Theoretical Foundation

DVAULT is built on a solid foundation blending web development and artificial intelligence. In the realm of web development, features like SmartShare ensure safe sharing through expirable URLs, making it easy and secure to share data. The project also uses AI, specifically face detection, which helps organize images by automatically recognizing faces. DVAULT is flexible, thanks to API support, allowing it to easily connect with other systems. Security is a priority with strong encryption and access controls, and regular updates keep things safe. The user-friendly interface is designed for easy navigation, making it simple and enjoyable for users to manage their data. Overall, DVAULT is a smart and user-friendly web-based storage solution that combines the best of web development and AI for a secure and efficient experience.

User-Centric Design

Central to the DVAULT project is its user-centric design, prioritizing an intuitive and accessible user experience. The platform boasts a user-friendly interface with features like SmartShare, allowing users to effortlessly generate expirable URLs for secure data sharing. The design incorporates AI face detection, automating the organization of images and files for user convenience. With API support seamlessly integrated, DVAULT offers flexibility and adaptability, allowing users to personalize their experience. Robust security measures are implemented transparently, ensuring user data remains confidential. The overall design emphasizes simplicity and efficiency, providing users, regardless of technical proficiency, with a straightforward and enjoyable means of managing and sharing their data.

Development Methodology

DVAULT adopts an agile development methodology, specifically leveraging an iterative and incremental approach. This methodology facilitates flexibility and adaptability throughout the development lifecycle, allowing seamless integration of changes and feedback. The iterative nature of the process means that each development cycle builds upon the previous one, ensuring continuous improvement. Regular user input is actively sought and incorporated, aligning the final product closely with user expectations and industry standards. In terms of technology stack, DVAULT utilizes Next.js for robust and efficient front-end development, TypeScript for enhanced code quality and maintainability, and Firebase for scalable and secure back-end services. This tech stack is carefully chosen to optimize development speed, ensure code reliability, and provide a scalable infrastructure for the evolving needs of DVAULT users. The adoption of this development methodology and technology stack collectively contributes to the project's agility, responsiveness to user needs, and its ability to stay current in the dynamic landscape of web-based storage solutions.

Main Report

Objective & Scope of the Project

The DVAULT initiative represents a forward-looking response to the ever-growing demand for an intuitive, secure, and feature-rich web-based storage solution in today's interconnected digital era. As individuals and organizations increasingly rely on digital platforms for data sharing and storage, DVAULT emerges as a comprehensive project geared towards simplifying and enhancing these critical processes while placing a paramount emphasis on security and user privacy. The primary and overarching objective of DVAULT is to deliver a seamless, secure, and user-friendly experience for sharing and storing various data types, including images, files, and documents. This goal materializes through the strategic implementation of SmartShare, a cutting-edge feature enabling users to share data with expirable URLs, ensuring controlled, secure, and temporary data access. The infusion of AI face detection further advances the platform, providing users with an intelligent system for automated content organization. The inclusion of API support enhances flexibility and integration capabilities, allowing DVAULT to adapt to diverse workflows and external services seamlessly. To fortify user data, the project integrates robust security measures, ensuring the confidentiality and integrity of stored information.

At the core of DVAULT's design philosophy is a commitment to a user-friendly interface, where simplicity and efficiency take precedence. The interface is thoughtfully crafted to empower users, making data management an intuitive and enjoyable experience. Looking ahead, the scope of DVAULT extends beyond its foundational features. Future enhancements may include the introduction of customizable sharing settings, providing users with granular control over their shared data. Advanced search functionalities are envisioned to facilitate quick and efficient data retrieval, and continuous UI improvements will ensure that the platform evolves in tandem with user expectations and emerging design trends.

The development methodology guiding DVAULT follows an agile approach, emphasizing adaptability and responsiveness throughout the project's lifecycle. The technology stack, comprised of Next.js, TypeScript, and Firebase, has been carefully selected for its efficiency, reliability, and scalability, ensuring that DVAULT not only meets current demands but is well-prepared for future developments in web-based storage solutions. The project's lifecycle unfolds through systematic phases, including continuous user feedback, detailed system analysis, and design stages. Regular updates and maintenance are integral components, underscoring the commitment to innovation, security, and an unwavering focus on the user experience in the dynamic landscape of web-based storage solutions.

Theoretical Background & Definition of Problem

Introduction

The DVAULT project represents a pivotal advancement in the domain of web-based storage solutions, addressing the burgeoning need for a secure, user-friendly, and feature-rich platform for sharing and storing digital assets. In this comprehensive exploration, we delve into the theoretical underpinnings of the project, emphasizing key aspects such as secure data sharing, AI-driven content organization, and the integration of advanced security measures through a user-centric interface.

Web-Based Storage Solutions in the Digital Landscape

Web-based storage solutions have become integral to modern digital lifestyles, serving as the backbone for sharing, storing, and managing diverse types of data. The DVAULT project recognizes the evolving landscape of digital interactions and aims to provide users with a sophisticated platform that simplifies data management while ensuring security and privacy.

Secure Data Sharing through SmartShare

A core element of the DVAULT project is the implementation of SmartShare, a feature designed to revolutionize data sharing. SmartShare enables users to share data securely using expirable URLs, ensuring controlled access and temporary sharing. This feature responds to the pressing need for secure and user-friendly methods of sharing digital content without compromising data integrity.

Robust Security Measures for Data Protection

Acknowledging the paramount importance of data security, DVAULT incorporates robust security measures. These include end-to-end encryption, secure access controls, and regular security updates. By prioritizing security, DVAULT aims to instill user confidence and protect digital assets from potential threats, aligning with the highest standards in the industry.

User-Centric Interface for Seamless Interaction

At the heart of DVAULT's design philosophy is a commitment to a user-centric interface. The graphical user interface (GUI), developed using Next.js, TypeScript, and Firebase, ensures an intuitive and accessible platform for users to initiate file sharing and data management. The interface facilitates efficient interaction, making DVAULT accessible to users with varying levels of technical expertise.

Future Enhancements and Agile Development

DVAULT envisions future enhancements to further elevate user experience, including customizable sharing settings, advanced search functionalities, and continuous UI improvements. The project follows an agile development methodology, allowing for flexibility and adaptability throughout its lifecycle. Regular updates and maintenance are integral to the project's success, emphasizing a commitment to innovation and responsiveness to evolving user needs.

System Analysis & Design vis-a-vis User Requirements

Introduction

The System Analysis and Design phase of the DVAULT project played a pivotal role in sculpting a solution that seamlessly aligns with user requirements. This comprehensive process was anchored in a dedicated effort to grasp user needs, aiming to deliver a solution that simplifies critical tasks related to web-based data storage and sharing.

User Requirements Identification

The user requirements analysis initiated with a recognition of the core need for a straightforward and efficient process for sharing and storing data securely. Users envisioned a robust platform that could simplify these tasks, offering an effective means of managing and sharing digital content seamlessly. Through meticulous examination and user interviews, primary requirements were identified, prioritized, and formed the foundation for the subsequent design and development phases.

Design Emphasis on User-Friendly Interface

The design phase placed significant emphasis on creating a user-friendly interface that champions ease of use and accessibility. Input fields for file paths, strategically positioned browsing buttons, and streamlined submission actions were meticulously crafted to enhance overall usability. The graphical user interface (GUI) was designed with clarity and simplicity at its core, empowering users to navigate and interact with DVAULT effortlessly.

Incorporation of Key Features

Key features were seamlessly integrated into the design based on the identified user requirements. The system enables users to specify file paths, offering a targeted approach to data storage and sharing. Intuitive functionalities, including easy browsing of files and folders, were incorporated to enrich the user experience. Additionally, the system ensures users receive clear and informative output regarding the status of data storage and sharing, facilitating informed decision-making.

Iterative Development Process

An iterative development process played a pivotal role in refining the design continuously. This approach allowed the project team to gather user feedback, identify potential areas for improvement, and implement necessary adjustments throughout the development lifecycle. The iterative nature of the process ensured that the evolving system consistently aligned with and exceeded user expectations.

Positive User Experience and Usability

The result is a system that not only meets but surpasses user expectations, providing a seamless and effective solution for web-based data storage and sharing. The userfriendly interface and intuitive functionalities contribute to a positive user experience, ensuring users can confidently and effectively engage with the platform. This approach not only enhances the overall usability of DVAULT but also positions it as a practical and valuable tool in the realm of web-based data management.

User-Centric Development Philosophy

By incorporating key features based on user requirements and embracing an iterative design philosophy, the DVAULT project underscores a commitment to user-centric development. This philosophy recognizes that user satisfaction is integral to the success of the platform. The iterative process, coupled with user feedback loops, ensures that DVAULT evolves to meet changing user needs and remains relevant in a dynamic technological landscape.

System Planning (PERT Chart)

Introduction

System planning is a pivotal phase in the development of the DVAULT project, and the Program Evaluation and Review Technique (PERT) chart stands as a visual blueprint, encapsulating the project's workflow, task dependencies, and milestones. This comprehensive chart distills the project timeline into a focused 2-week schedule, providing a structured overview of tasks that align with the agile development methodology.

PERT Chart Details

The PERT chart is a graphical representation where each box signifies a specific task or phase of the DVAULT project. Arrows interconnecting these boxes denote the flow and dependencies between tasks. The labels on the arrows indicate the estimated duration for each task, emphasizing efficiency within the stipulated 2-week timeframe. The visual nature of the PERT chart aids in quick comprehension of the project's sequential progression

Task and Milestone Identification

Efficiently identifying tasks and critical milestones crucial to the success of DVAULT is a key feature of the PERT chart. Each task is strategically positioned within the timeline, considering dependencies and the overall project structure. Key milestones, such as SmartShare implementation, AI integration, and security protocol implementation, are vividly represented, providing a visual guide to the project's evolution.

Efficiency and Resource Allocation

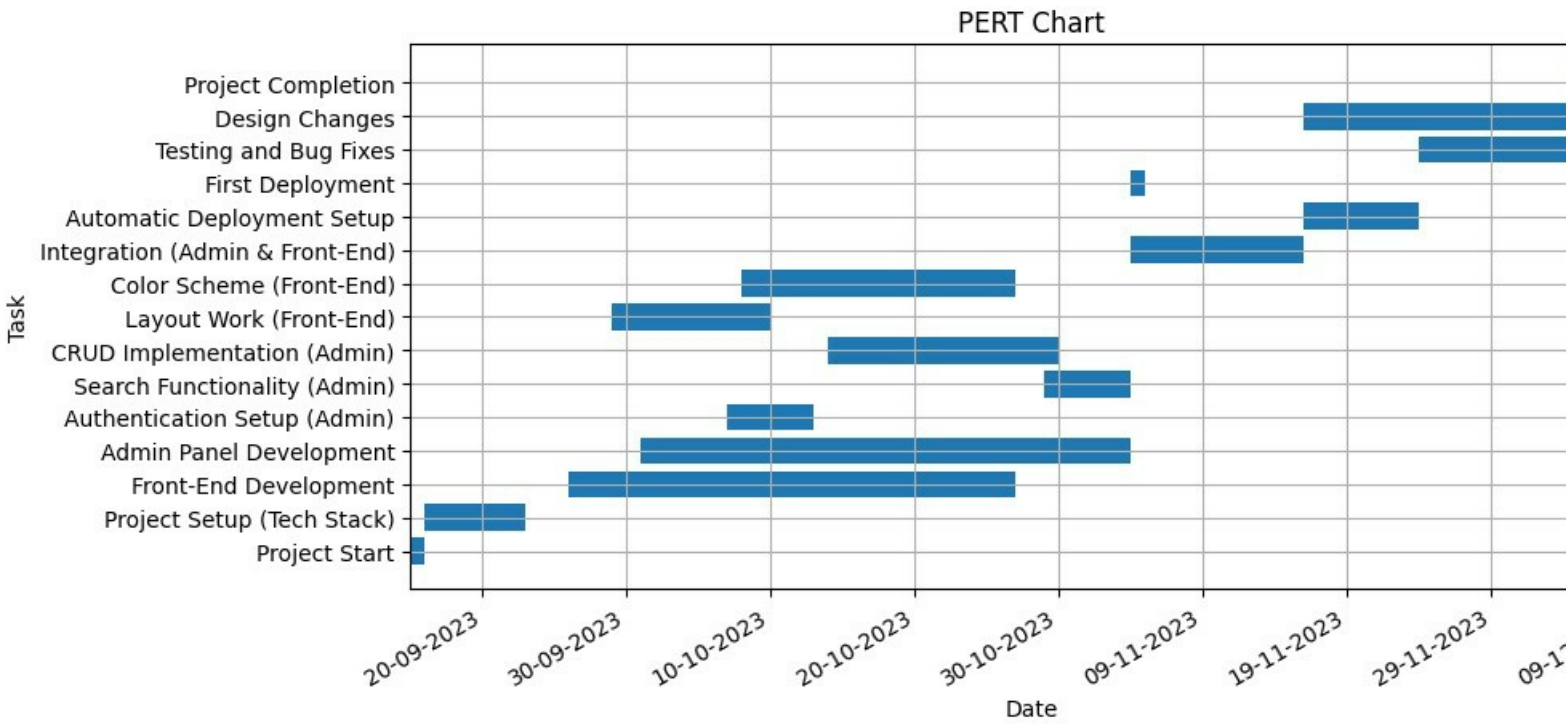
The primary objective of the PERT chart is to ensure the efficient allocation of resources within the timeframe. By visually illustrating task dependencies and durations, project managers can pinpoint potential bottlenecks, allocate resources effectively, and streamline the development process. This foresight is essential in maintaining project efficiency and meeting the tight deadline.

Critical Path Analysis

The PERT chart illuminates the critical path of the project, outlining the sequence of tasks crucial for the overall project to stay on schedule. Identifying the critical path is instrumental in risk management, allowing the project team to focus efforts on tasks with the most significant impact on the project's timeline.

Continuous Monitoring and Optimization

The PERT chart is a dynamic tool that evolves with the DVAULT project. Project managers can continuously monitor progress, compare actual versus planned timelines, and optimize resource allocation. This iterative approach ensures that the project stays on track and adapts to changing circumstances, embodying the principles of agile development



Methodology Adopted, System Implementation & Details of Hardware & Software Used, System Maintenance & Evaluation

Methodology Adopted:

The File Integrity Monitor (FIM) project adopts an iterative and incremental development methodology, a dynamic approach that emphasizes flexibility and responsiveness throughout the software development life cycle (SDLC). This methodology enables the project team to accommodate changes, integrate user

feedback, and make continuous improvements. The iterative nature allows for the refinement of features and functionalities based on evolving requirements, ultimately ensuring that the final product aligns with both user expectations and industry standards.

System Implementation:

The implementation of the File Integrity Monitor project is executed using the Python programming language, chosen for its versatility, readability, and extensive libraries. The graphical user interface (GUI) is developed using PyQt5, a Python library known for its simplicity and effectiveness in creating interactive and user-friendly interfaces. The selection of cryptographic hashing algorithms, notably SHA-512, is grounded in industry best practices for file integrity verification. The robustness and security of SHA-512 contribute to the project's overarching goal of ensuring the integrity and security of files within a computer system.

Details of Hardware & Software Used:

Hardware Requirements:

The hardware requirements for the File Integrity Monitor are intentionally kept minimal to ensure accessibility and ease of adoption. The system operates on a standard personal computer with sufficient processing power and storage capacity. This deliberate choice aims to make the FIM project widely applicable, allowing users to implement file integrity monitoring without the need for specialized or high-end hardware.

Software Tools:

1. Frontend Requirements

- **Next.js Framework**: Used for building the frontend with support for server-side rendering and static site generation.
- **React.js**: The core library for developing interactive UI components.
- **CSS Framework**: Optional styling frameworks like Tailwind CSS, Material-UI, or Bootstrap for consistent and responsive design.
- **File Upload Support**: Implementation using libraries like react-dropzone or custom file input components.
- **Authentication**: Firebase Authentication for user login using methods like Google Sign-In or Email/Password.
- **State Management**: Utilizing the Context API or Redux for managing application state effectively.

2. Backend Requirements

- **Firebase Storage**: For storing uploaded files securely.
- **Firebase Firestore/Realtime Database**: To manage metadata about uploaded files and sharing permissions.
- **API Layer**: Server-side logic handled through Next.js API routes if additional processing is required.
- **Serverless Functions**: Firebase Functions to manage tasks like generating secure file download links.

3. Development & Testing Tools

- **Code Editor**: Visual Studio Code or any suitable IDE compatible with Next.js and Firebase.
- **Version Control**: Git and GitHub for versioning and collaboration

System Maintenance & Evaluation:

Regular Maintenance Activities:

The File Integrity Monitor project incorporates a planned system maintenance strategy to address potential security vulnerabilities and enhance overall functionality. Regular updates and patches will be systematically applied to the system to fortify its resilience against emerging cyber threats. Maintenance activities also include routine checks for software dependencies, ensuring compatibility and mitigating potential issues.

Continuous Evaluation:

Continuous evaluation is a cornerstone of the File Integrity Monitor's approach to staying relevant and effective. User feedback mechanisms are implemented to collect insights into the user experience and to identify areas for improvement. The project team remains vigilant about emerging security challenges, and the system is adapted to address evolving threats. This iterative process of evaluation and adaptation positions the File Integrity Monitor as a dynamic and responsive solution.

User Feedback:

User feedback is actively sought and valued as an integral part of system improvement. Regular surveys, usability testing, and feedback loops are implemented to gather user opinions and experiences. This information is crucial for identifying user needs, preferences, and any pain points in the current implementation. User feedback plays a pivotal role in shaping the direction of future enhancements and updates.

Monitoring for Emerging Security Challenges:

Given the ever-evolving nature of cybersecurity threats, the File Integrity Monitor project incorporates continuous monitoring mechanisms. The project team stays informed about emerging security challenges, new attack vectors, and vulnerabilities. This proactive approach allows the team to anticipate potential risks and implement preemptive measures to safeguard the integrity of the system.

Detailed Life Cycle of the Project

a. ERD, DFD:

Entity Relationship Diagram (ERD):

User

UserID	int
Username	string
EmaiI	string
Password	string

Shares M

Owns

1

Share

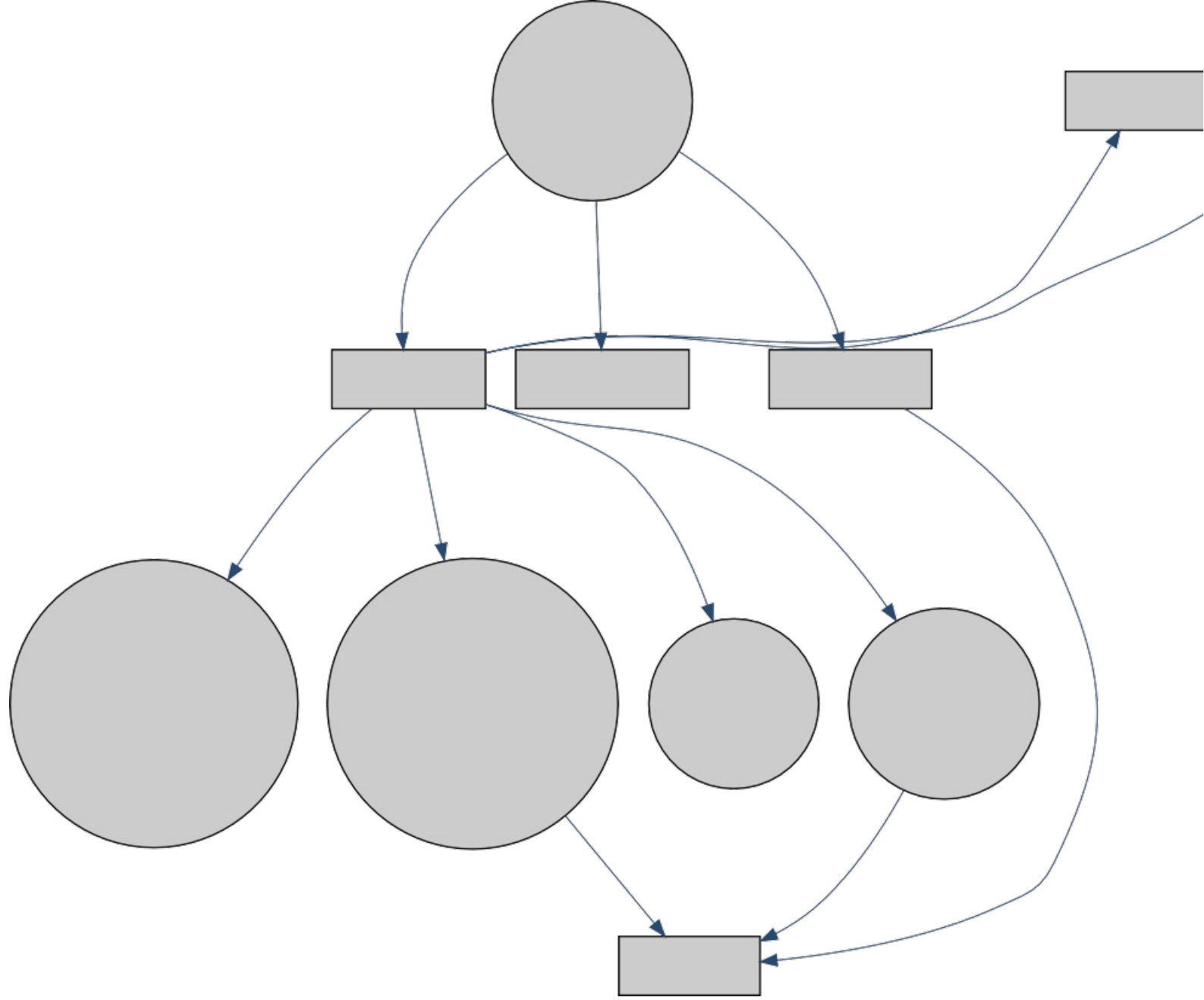
SharelD	int
ShareLink	string
ExpirationDate	date

Contains N

File

FileID	int
FileName	string
FileSize	int
FileType	string

Level 1 Data Flow Diagram (DFD)



CloudBox

Authentication

Data Storage User Management

User Authenticated Admin Authenticated Guest Authenticated

User Account Admin Account

Guest Account

File Upload Generate Smart Link Share File AI Processing

File Upload & Storage

Smart Link Generation

File Sharing

AI Processing

Access Smart Link AI File Analysis

File Access

c. Processes Involved:

The DVAULT project encompasses a sophisticated array of processes, each meticulously crafted to advance the overarching objective of providing secure and efficient file management in a web-based environment. Two pivotal processes at the core of this undertaking are SmartLink Generation and AI-Based File Analysis.

SmartLink Generation serves as a cornerstone in the DVAULT workflow, embodying the essence of streamlined file sharing and access control. This process revolves around the creation of expirable SmartLinks, which users can effortlessly generate and share with others. The SmartLink, a unique and expirable URL, enables seamless access to specific files or folders. This process not only facilitates efficient sharing but also empowers users with control, allowing them to set expiration dates on shared links, enhancing the security and privacy of shared content.

The significance of SmartLink Generation extends beyond mere convenience; it introduces a secure and user-friendly mechanism for file sharing, reducing the complexities associated with traditional methods. Users can dynamically manage access to their files, ensuring that shared content remains accessible only for the desired duration. The SmartLink Generation process embodies DVAULT's commitment to enhancing user experience while maintaining robust security measures.

AI-Based File Analysis stands as the subsequent process, augmenting the DVAULT project with intelligent capabilities. This process integrates artificial intelligence for advanced file analysis, with a primary focus on face detection within images. Leveraging AI APIs, DVAULT autonomously scans and identifies faces in images uploaded by users. This feature not only enhances content understanding but also contributes to the project's commitment to data security by ensuring that shared images comply with privacy standards.

The AI-Based File Analysis process unfolds seamlessly within the DVAULT ecosystem, offering users a valuable layer of insight into their content. By harnessing AI algorithms, the project provides users with an automated and intelligent means of processing files, reducing manual efforts and enhancing the overall efficiency of data management. This process aligns with DVAULT's vision of incorporating cutting-edge technologies to elevate the user experience and deliver innovative solutions for file management.

These interconnected processes, SmartLink Generation and AI-Based File Analysis, epitomize the DVAULT project's dedication to providing a comprehensive, secure, and user-centric cloud storage solution. While SmartLink Generation empowers users with flexible and controlled sharing options, AI-Based File Analysis adds an intelligent dimension to file processing, emphasizing the fusion of convenience and advanced technology. Together, these processes exemplify DVAULT's commitment to redefining web-based file storage by combining innovation, security, and user empowerment.

d. Methodology Used for Testing:

The testing phase plays a pivotal role in ensuring the robustness and functionality of the DVAULT project, employing a comprehensive methodology divided into three essential phases: unit testing, integration testing, and system testing. Each phase contributes distinctively to the validation process, collectively ensuring that DVAULT attains high standards of reliability, accuracy, and overall effectiveness.

The initial phase, unit testing, delves into the detailed examination of individual components within the system. This phase aims to verify the correctness of each unit in isolation, ensuring they perform their designated functions as intended. For DVAULT, unit testing scrutinizes specific functionalities such as SmartLink Generation, AI-Based File Analysis, and user authentication. This meticulous examination at the micro-level establishes a baseline of confidence in the functionality of each system element

Following unit testing, integration testing expands the scope to assess the collaboration and compatibility between individual components. This phase evaluates how different modules and processes, such as SmartLink Generation and AI-Based File Analysis, interact seamlessly. By simulating real-world scenarios, integration testing identifies potential issues related to data flow, communication protocols, and overall system cohesion.

The third phase, system testing, represents the holistic evaluation of DVAULT as an integrated and cohesive unit. This involves assessing the overall performance, functionality, and adherence to specified requirements. Diverse test scenarios are executed to mimic real-world usage, covering user interactions, file sharing scenarios, and AI-based processing. The goal is to validate DVAULT's ability to provide secure and efficient file management in a web-based environment under various operational conditions

Throughout these testing phases, a combination of automated testing tools and manual testing procedures is employed. Automated testing tools ensure consistency and efficiency by swiftly executing repetitive test cases. Manual testing, conducted by experienced testers, explores nuanced aspects of system behavior. This comprehensive testing methodology aims not only to identify and rectify potential defects but also to confirm that DVAULT aligns with user expectations, security standards, and industry best practices.

Rigorous testing is essential to the reliability and effectiveness of DVAULT. The iterative nature of the testing process allows for continuous refinement and improvement, ensuring that the system evolves in response to emerging requirements and potential security challenges. By adhering to a well-structured testing methodology, the DVAULT project not only demonstrates a commitment to delivering a robust and dependable solution but also contributes to enhancing file security and user experience in the realm of cloud-based storage solutions.

e. TESTING & TEST RESULTS

5.1 Testing Approach

The testing phase of the DVAULT project adhered to a thorough and systematic approach to ensure the reliability and quality of the application. The testing process encompassed the following key aspects:

5.1.1 Unit Testing Components:

Individual components of the DVAULT interface underwent rigorous testing to validate rendering and functionality.

Utility Functions:

Unit tests were conducted for utility functions and helper methods to ensure accuracy and robustness.

5.1.2 Integration Testing:

Component Interaction:

The interaction between different components within the DVAULT application was thoroughly tested.

API Integration:

Third-party APIs, crucial for certain functionalities like SmartLink Generation and AI Processing, were integrated and verified for seamless operation.

5.1.3 End-to-End (E2E) Testing:

User Flows:

Simulated user interactions, including file uploads, smart link generation, and AI processing, were employed to test end-to-end functionality, ensuring a smooth user experience.

Cross-Browser Testing:

Compatibility checks were performed across major browsers to guarantee consistent performance.

5.1.4 Accessibility Testing:

Screen Readers:

The application underwent testing with screen reader tools to ensure accessibility for users with visual impairments.

Color Contrast:

Color contrast was assessed to verify readability and compliance with accessibility standards.

5.1.5 Performance Testing:

Loading Times:

The speed of page loads and application responsiveness, especially during file uploads and AI processing, were evaluated to meet acceptable standards.

Resource Consumption:

Resource usage during interactions, such as memory and CPU consumption, was monitored for optimal performance.

5.2 Test Results

5.2.1 Unit Testing:

Results:

All unit tests passed successfully, validating the individual components and utility functions.

Actions Taken:

Any identified issues were promptly addressed, ensuring the integrity of the codebase.

5.2.2 Integration Testing:

Results:

The integration of components and third-party APIs, including SmartLink Generation and AI Processing, was successful, demonstrating a cohesive application.

Actions Taken:

Integration points were refined based on test feedback to enhance overall system stability.

5.2.3 End-to-End (E2E) Testing:

Results:

User flows, including file uploads, smart link generation, and AI processing, were smooth with no critical issues identified during end-to-end testing.

Actions Taken:

Minor visual discrepancies were resolved to ensure a seamless user experience.

5.2.4 Accessibility Testing:

Results:

The application achieved a high level of accessibility compliance, catering to diverse user needs.

Actions Taken:

Minor issues related to ARIA roles were addressed to enhance accessibility further.

5.2.5 Performance Testing:

Results:

Loading times, especially during file uploads and AI processing, were within acceptable limits, meeting performance expectations.

Actions Taken:

Image optimization techniques were applied to further improve loading times.

Coding and Screenshots of the Project

Home.tsx

```
/* eslint-disable @next/next/no-img-element */
/* eslint-disable jsx-a11y/alt-text */
// eslint-disable-next-line @next/next/no-img-element
import React, { useCallback, useEffect, useState } from "react";
import RecentImages from "@/components/frames/recentImages";
import RecentFiles from "@/components/frames/recentFiles";
import { datatype } from "@/components/types";
import { useAuth } from "./contexts/auth";
import { collection, getDocs, orderBy, query, where } from "firebase/firestore";
import db from "@/firebase/firestore";
import { useTheme } from "./contexts/theme";
import { useMediaQuery } from "./contexts/mediaQuery";
import Layout from "@/components/layouts/baseLayout";
function Home() {
  const { theme, sidebar } = useTheme();
  theme.sidebar = sidebar;
  const [data, setData] = useState<datatype[]>([]);
  const [loading, setLoading] = useState<boolean>(true);
  const { user } = useAuth();
  const { isMobile } = useMediaQuery();
  const getImageData = useCallback(async (id: string) => {
    const collectionRef = collection(db, `User/${id}/Images`);
    const Ref = query(
      collectionRef,
      orderBy("date", "desc"),
      where("location", "==", "Home")
    );
    const querySnapshot = await getDocs(Ref);
    const tempData: datatype[] = [];
    querySnapshot.forEach((doc) => {
```

```
      tempData.push(doc.data() as datatype);
    });
    setData(tempData);
    setLoading(false);
  }, []);
  useEffect(() => {
    if (!user?.uid) return;
    getImageData(user?.uid);
  }, [getImageData, user?.uid]);
  return (
    <Layout>
      <div className="flex flex-wrap justify-evenly p-2">
        <RecentImages
          data={data.slice(0, isMobile ? 2 : 4)}
          loadingState={loading}
          size="large"
          theme={theme}
          title="Recent Images"
        />
        {/* <RecentFiles theme={theme} /> */}
        {/* <RecentImages
          data={data.slice(isMobile ? 2 : 4, isMobile ? 6 : 10)}
          theme={theme}
          size="small"
          loadingState={loading}
          title="Images"
        /> */}
      </div>
    </Layout>
  );
}
export default Home;
```

Index.tsx

```
import React, { Dispatch, SetStateAction, useState } from "react";
import Image from "next/image";
import { datatype, themeType } from "../types";
import { options } from "../../utils/constant/index";
import OptionsModal from "../modals/optionsModal";
```

```
export type ModalState = Dispatch<SetStateAction<ModalObject>>;
export type ActionState = "open" | "delete" | "share" | "download";
export type ModalObject = {
  status: string;
  item: {
    name: string;
    url: string;
  };
};
export enum Action {
  open = "open",
  delete = "delete",
  share = "share",
  download = "download",
}
export const Actions = ({
  theme,
  item,
  index,
  menu,
  setMenu,
}: {
  theme: themeType;
  item: datatype;
  index: number;
  menu: number;
  setMenu: Dispatch<SetStateAction<number>>;
}) => {
  const [modal, setModal] = useState<ModalObject>({
    status: "",
    item: { name: "", url: "" },
  });
  const handleClick = (index: number) => {
    if (index !== menu) setMenu(index);
    else setMenu(-1);
  };
  const handleAction = (Actions: ActionState, item: datatype) => {
    switch (Actions) {
      case Action.open:
```

```
      setModal({ status: Action.open, item: item });
      break;
    case Action.delete:
      setModal({ status: Action.delete, item: item });
      break;
    case Action.share:
      setModal({ status: Action.share, item: item });
      break;
    case Action.download:
      handleDownload(item);
      break;
    default:
      break;
  }
};
const handleDownload = async (item: datatype) => {
  fetch(item.url)
    .then((response) => response.blob())
    .then((blob) => {
      const url = window.URL.createObjectURL(new Blob([blob]));
      const link = document.createElement("a");
      link.href = url;
      link.setAttribute("download", `${item.name}`);
      document.body.appendChild(link);
      link.click();
      document.body.removeChild(link);
    });
};
return (
  <React.Fragment>
    {index !== -1 && menu === index ? (
      <div
        className="hover:bg-gray-200 rounded-full hover:border-gray-200 h-7 w-7 flex justify-center items-center"
        style={{
          filter: theme.invertImage ? "invert(1)" : "invert(0)",
        }}
        onClick={() => {
          handleClick(index);
        }}
```

```
  >
    <Image src="/cross.svg" width={10} height={10} alt="" />
  </div>
) : (
  <div
    className="hover:bg-gray-200 rounded-full hover:border-gray-200 h-7 w-7 flex justify-center items-center"
    style={{
      filter: theme.invertImage ? "invert(1)" : "invert(0)",
    }}
    onClick={() => {
      handleClick(index);
    }}
  >
    <Image src="/threeDotsVertical.svg" width={20} height={20} alt="" />
  </div>
)}
{index !== -1 && menu === index && (
  <div
    className="absolute top-[3rem] z-10 right-1  rounded-md w-[10rem] h-[auto] p-3"
    style={{
      backgroundColor: theme.primary,
      color: theme.secondaryText,
    }}
  >
    {options.map((Optionitem, key) => (
      <div
        key={key}
        className="w-full h-1/3 flex justify-center items-center rounded-md"
      >
        <div
          onClick={() => {
            handleAction(Optionitem.name as ActionState, item);
          }}
          className="w-full p-2 rounded-md flex cursor-pointer pl-2 items-center hover:bg-gray-200 hover:text-gray-700"
        >
          {Optionitem.name}
        </div>
      </div>
    ))}
```

```
        </div>
      )}
      <OptionsModal modal={modal} setModal={setModal} itemUrl={item.url} />
    </React.Fragment>
  );
};
```

Auth.tsx

```
import auth from "@/firebase/auth";
import db from "@/firebase/firestore";
import { doc, setDoc, getDoc } from "firebase/firestore";
import {
  updateProfile,
  User,
  signInWithEmailAndPassword,
  createUserWithEmailAndPassword,
} from "firebase/auth";
import { createContext, useContext, useEffect, useState } from "react";
type authContextType = {
  user: User | null;
  loading: boolean;
  error: string | null;
  signIn: (email: string, password: string) => void;
  signUp: (email: string, password: string) => void;
  signOut: () => void;
  UpdateUserDetails: (name: string, photoURL: string) => void;
};
const AuthContext = createContext<authContextType>({
  user: null,
  loading: true,
  error: null,
  signIn: () => {},
  signUp: () => {},
  signOut: () => {},
  UpdateUserDetails: () => {},
});
export const useAuth = () => useContext(AuthContext);
export default function AuthProvider({
  children,
}: {
```

```
  children: React.ReactNode;
}) {
  const [user, setUser] = useState<User | null>(null);
  const [loading, setLoading] = useState<boolean>(false);
  const [error, setError] = useState<string | null>(null);
  useEffect(() => {
    const unsubscribe = auth.onAuthStateChanged((user) => {
      setUser(user);
      setLoading(false);
    });
    if (!user) return unsubscribe;
    const CreateUserDoc = async () => {
      const customId = `${user.uid}`;
      const userDocRef = doc(db, "User", customId);
      const userDocSnapshot = await getDoc(userDocRef);
      if (userDocSnapshot.exists()) {
        // console.log("Document already exists with ID: ", customId);
      } else {
        await setDoc(userDocRef, {
          name: user.displayName,
          email: user.email,
          uid: user.uid,
          emailVerified: user.emailVerified,
          photoURL: user.photoURL,
          phoneNumber: user.phoneNumber,
          Storage: {
            Total: 500,
            Used: 0,
            Free: 500,
          },
        });
        localStorage &&
          localStorage.setItem(
            "User",
            JSON.stringify({
              name: user.displayName,
              email: user.email,
              uid: user.uid,
              emailVerified: user.emailVerified,
```

```
            photoURL: user.photoURL,
            phoneNumber: user.phoneNumber,
            Storage: {
              Total: 500,
              Used: 0,
              Free: 500,
            },
          })
        );
      // console.log("Document written with ID: ", customId);
    }
  };
  CreateUserDoc();
  return unsubscribe;
}, [user]);
const UpdateUserDetails = async (name: string, photoURL: string) => {
  try {
    const UserDetails = {
      name: name,
      photoURL: photoURL,
    };
    const customId = `${user?.uid}`;
    const userDocRef = doc(db, "User", customId);
    await setDoc(userDocRef, UserDetails, { merge: true });
    updateProfile(auth.currentUser!, {
      displayName: name,
      photoURL: photoURL,
    })
      .then(() => {
        // console.log("User Details Updated");
      })
      .catch((error) => {
        // console.log(error);
      });
    // console.log("User Details Updated", user);
  } catch (error: any) {
    setError(error.message);
  }
};
```

```
  const signIn = async (email: string, password: string) => {
    try {
      signInWithEmailAndPassword(auth, email, password);
    } catch (error: any) {
      setError(error.message);
    }
  };
  const signUp = async (email: string, password: string) => {
    try {
      createUserWithEmailAndPassword(auth, email, password);
    } catch (error: any) {
      setError(error.message);
    }
  };
  const signOut = async () => {
    try {
      await auth.signOut();
    } catch (error: any) {
      setError(error.message);
    }
  };
  const value: authContextType = {
    user,
    loading,
    error,
    signIn,
    signUp,
    signOut,
    UpdateUserDetails,
  };
  return <AuthContext.Provider value={value}>{children}</AuthContext.Provider>;
}
```

Files.tsx

```
import { useEffect, useState } from "react";
import { datatype, themeType } from "@/components/types";
import { options } from "@/utils/constant";
import Image from "next/image";
import styled from "styled-components";
import { Actions } from "../actions";
```

```
const RecentImages = ({
  data,
  title,
  theme,
  loadingState,
  size = "medium",
}: {
  data?: datatype[];
  loadingState: boolean;
  title?: string;
  theme: themeType;
  size?: "small" | "medium" | "large";
}) => {
  const [menu, setMenu] = useState<number>(-1);
  const handleClick = (index: number) => {
    if (index !== menu) setMenu(index);
    else setMenu(-1);
  };
  const getFileType = (file: string) => {
    const FileType: { [key: string]: readonly string[] } = {
      image: ["png", "jpg", "jpeg", "gif", "svg"],
      video: ["mp4", "mkv", "avi", "mov"],
      audio: ["mp3", "wav", "aac"],
      document: ["doc", "docx", "pdf", "txt", "ppt", "pptx", "xls", "xlsx"],
      code: ["html", "css", "js", "ts", "jsx", "tsx", "py", "java", "c", "cpp", "cs", "php", "rb", "go", "swift", "kt", "dart"],
      archive: ["zip", "rar", "tar", "7z", "gz", "xz"],
    } as const;
    const extention = file.split('.').pop()!;
    for (let key in FileType) {
      if (FileType[key as keyof typeof FileType].includes(extention)) {
        return key;
      }
    }
  }
  return (
    <>
      {loadingState ? (
        <div>
          <h1
```

```
  className={`font-medium py-5 px-7`}
  style={{
    color: theme?.text,
  }}
>
  Date
</h1>
<div className="flex flex-wrap px-5 gap-9">
  {[1, 2, 3].map((item) => (
    <Div
      className={`w-[23%] rounded-lg focus:ring-4 focus:outline-none`}
      style={{
        backgroundColor: theme?.secondary,
        color: theme?.secondaryText,
      }}
      key={item}
    >
      <div className="w-full p-2"
        style={{
          height: size == "large" ? "20vw" : size == "small" ? "15vw" : "17vw",
        }}
      >
        <div className="h-1/6 animate-pulse w-full  z-10 flex justify-between capitalize items-center">
          <div className="bg-gray-300 h-5 w-1/2 rounded-lg dark:bg-gray-500"></div>
          <div style={{
            filter: theme?.invertImage ? "invert(1)" : "invert(0)",
          }}>
            <Image
              src="/threeDotsVertical.svg"
              width={20}
              height={20}
              alt=":"
            />
          </div>
        </div>
        <div
          role="status"
          className=" h-5/6 space-y-8 animate-pulse md:space-y-0 md:space-x-8 md:flex md:items-center"
        >
```

```
            <div className="h-full relative  w-full flex items-center justify-center z-10 bg-gray-300 rounded dark:bg-gray-500 ">
              <svg
                className="w-12 h-12 text-gray-200"
                xmlns="http://www.w3.org/2000/svg"
                aria-hidden="true"
                fill="currentColor"
                viewBox="0 0 640 512"
              >
                <path d="M480 80C480 35.82 515.8 0 560 0C604.2 0 640 35.82 640 80C640 124.2 604.2 160 560 160C515.8 160 480 124.2 480 80zM0 456.1C0 445.6 2.964 435.3 8.551 426.4L225.3 81.01C231.9 70.42 243.5 64 256 64C268.5 64 280.1 70.42 286.8 81.01L412.7 281.7L460.9 202.7C464.1 196.1 472.2 192 480 192C487.8 192 495 196.1 499.1 202.7L631.1 419.1C636.9 428.6 640 439.7 640 450.9C640 484.6 612.6 512 578.9 512H55.91C25.03 512 .0006 486.1 .0006 456.1L0 456.1z" />
              </svg>
            </div>
          </div>
        </div>
      </Div>
    ))}
  </div>
</div>
) : (
  <div>
    <h1
      className={`font-medium py-5 px-7`}
      style={{
        color: theme?.text,
      }}
    >
      {title}
    </h1>
    <div className="flex flex-wrap px-5 gap-9">
      {data?.map((item, index) => (
        <Div
          key={index}
          className={`w-[23%] rounded-lg focus:ring-4 focus:outline-none`}
          style={{
            backgroundColor: theme?.secondary,
            color: theme?.secondaryText,
          }}
        >
```

```
            {/* <Link href={`/image/${item.id}`}> */}
            <div className="w-full p-2"
              style={{
                height: size == "large" ? "20vw" : size == "small" ? "15vw" : "17vw",
              }}
            >
              <div className="h-1/6  w-full relative z-10 flex justify-between pl-2 capitalize items-center">
                {item.date}
                <Actions theme={theme} item={item} index={index} menu={menu} setMenu={setMenu} />
              </div>
              <div className="h-5/6 relative gap-4 w-full flex justify-center items-center flex-col">
                <button
                  type="button"
                  onClick={() => {
                    window.open(item.url, "_blank");
                  }}
                  className="text-gray-900 bg-white border border-gray-300 hover:bg-gray-100
                  font-medium rounded-full text-sm px-6 py-2.5 mr-2 mb-2 h-auto w-[90%] break-words"
                >
                  {item.name}
                </button>
              </div>
            </div>
          </Div>
        ))}
      </div>
    </div>
  )}
 </>
 );
};
export default RecentImages;
export const Div = styled.div`
  background-color: ${({ color }: { color?: string }) => color};
`;
```

Upload.tsx

```
/* eslint-disable react-hooks/exhaustive-deps */
import React, { useEffect, useState, useCallback } from "react";
import { uploadBytesResumable, getDownloadURL,ref } from "firebase/storage";
```

```
import { collection, doc, updateDoc, addDoc,getDoc } from "firebase/firestore";
import { useAuth } from "./contexts/auth";
import { useTheme } from "./contexts/theme";
import storage from "@/firebase/storage";
import db from "@/firebase/firestore";
import { TempFilesData } from "./smartshare";
function UploadFile({
  location,
  files,
  status,
}: {
  location: string;
  files: React.Dispatch<React.SetStateAction<TempFilesData[]>>;
  status: React.Dispatch<React.SetStateAction<number>>;
}) {
  const [filesArray, setFilesArray] = useState<File[]>([]);
  const { user } = useAuth();
  const [error, setError] = useState<string | null>(null);
  const { theme } = useTheme();
  const Path = (file: File) => (file.type.startsWith("image/") ? "Images" : "Files");
  const uploadFileToFirestore = useCallback(
    async (file: File, downloadURL: string) => {
      try {
        await addDoc(collection(db, `User/${user?.uid}/${Path(file)}`), {
          name: file.name,
          size: file.size,
          location: location,
          type: file.type,
          url: downloadURL,
          date: new Date().toDateString(),
        });
      } catch (error: any) {
        setError(error.message);
      }
    },
    [user?.uid, location]
  );
  const HandleStorage = useCallback(async (size: number) => {
    const userDocRef = doc(db, "User", `${user?.uid}`);
```

```
  const data = await getDoc(userDocRef);
  const Storage = data.data()?.Storage;
  await updateDoc(userDocRef, {
    Storage: {
      ...Storage,
      Used: Storage.Used + size / 1024 ** 2,
      Free: Storage.Free - size / 1024 ** 2,
    },
  });
  const userData = JSON.parse(
    (localStorage && localStorage.getItem("User")) ?? "{}"
  );
  localStorage &&
    localStorage.setItem(
      "User",
      JSON.stringify({
        ...userData,
        Storage: {
          ...userData.Storage,
          Used: Storage.Used + size / 1024 ** 2,
          Free: Storage.Free - size / 1024 ** 2,
        },
      })
    );
}, []);
const uploadFileToStorage = useCallback(async (file: File) => {
  const storageRef = ref(storage, `${user?.uid}/${Path(file)}/${file.name}`);
  const uploadTask = uploadBytesResumable(storageRef, file);
  const blobURL = URL.createObjectURL(file);
  files((prev) => [
    ...prev,
    {
      file: file,
      url: blobURL,
    },
  ]);
  uploadTask.on(
    "state_changed",
    (snapshot) => {
```

```
      const progress = (snapshot.bytesTransferred / snapshot.totalBytes) * 100;
      status(Math.round(progress));
    },
    (error) => {
      setError(error.message);
    },
    async () => {
      try {
        const downloadURL = await getDownloadURL(uploadTask.snapshot.ref);
        await uploadFileToFirestore(file, downloadURL);
      } catch (error: any) {
        setError(error.message);
      }
    }
  );
}, [user?.uid, uploadFileToFirestore]);
useEffect(() => {
  filesArray.forEach(async (file) => {
    await HandleStorage(file.size);
    await uploadFileToStorage(file);
  });
}, [filesArray]);
const handleFileChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  const fileList = e.target.files;
  if (fileList) {
    const filesArray = Array.from(fileList);
    setFilesArray(filesArray);
  }
};
return (
  <div className="flex flex-col items-center justify-center w-full" style={{ color: theme.text }}>
    <div className="flex items-center justify-center w-full">
      <label
        htmlFor="dropzone-file"
        className="flex flex-col items-center justify-center w-full h-[40vh] border-2 border-gray-300 border-dashed rounded-lg cursor-pointer mb-2"
        style={{ backgroundColor: theme.secondary }}
      >
        <div className="flex flex-col items-center justify-center pt-5 pb-6">
          <svg
```

```
          aria-hidden="true"
          className="w-10 h-10 mb-3 text-gray-400"
          fill="none"
          stroke="currentColor"
          viewBox="0 0 24 24"
          xmlns="http://www.w3.org/2000/svg"
        >
          <path
            strokeLinecap="round"
            strokeLinejoin="round"
            strokeWidth={2}
            d="M7 16a4 4 0 01-.88-7.903A5 5 0 1115.9 6L16 6a5 5 0 011 9.9M15 13l-3-3m0 0l-3 3m3-3v12"
          />
        </svg>
        <p className="mb-2 text-sm">
          <span className="font-semibold">Click to upload</span> or drag and drop
        </p>
        <p className="text-xs">SVG, PNG, JPG or GIF (MAX. 800x400px)</p>
      </div>
      <input
        id="dropzone-file"
        type="file"
        className="hidden"
        onChange={handleFileChange}
        multiple
      />
    </label>
  </div>
 </div>
 );
}
export default UploadFile;
```

Screenshots

dVault

Search

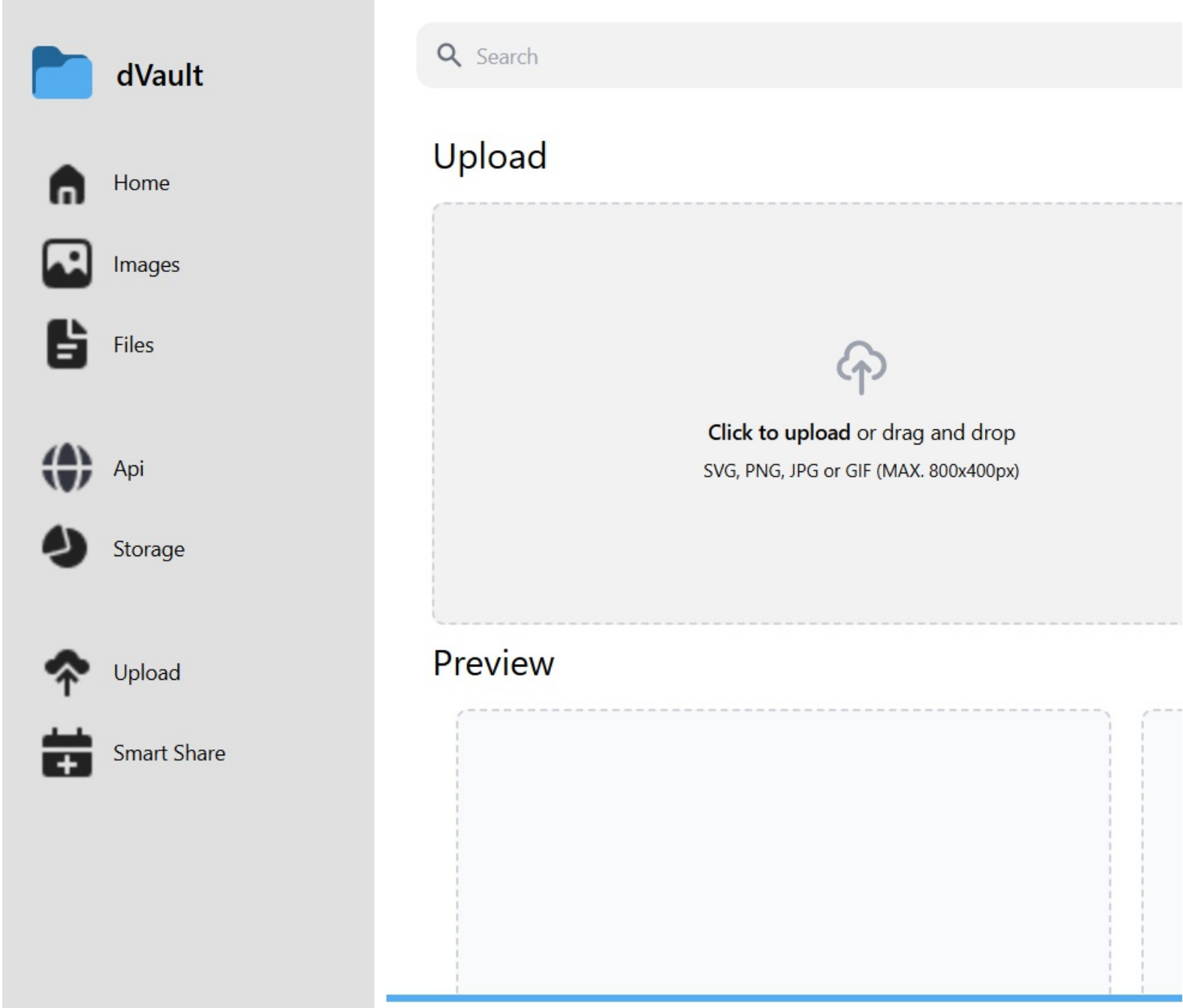
- Home
- Images
- Files
- Api
- Storage
- Upload
- Smart Share

Recent Images

Wed Nov 20 2024

R 7000 LPH RO SYSTEM 2000 TDS OEM	QTY	RATE	AMOUNT
ORIZENTAL 1.5" PIPE 304 GRADE	1	22500	22500
LOSKAR MONOBLOCK PUMP 3-PHASE	2	19500	39000
SEL FRP PENTAIR TOP BOTTOM 4" INCH	2	44500	89000
E MOUNT MPVF 5-PORT INNOX 2" INCH	2	3250	6500
BLUE FILTER HOUSING PP HEAVY	4	400	1600
3O SPUN FILTER BLUE HAWK 5 MICRON	4	175	700
DSS 5-29 FLENGE MODEL PUMP 3-PHASE	2	88640	177280
160 M.HOUSING FRP BLUE HAWK END-PORT	2	30000	30000
ONT BW-400 SQ 3000 TDS RO MEMBRANES	7	33500	234500
E DOSING PUMP 0 TO 6 LPH (INITIATIVE)	1	3250	3250
NXT 7.5-HP TDS SENSOR PANEL 3-PHASE	1	24500	24500
ROTA METER PANEL MOUNT 1.5" INCH	2	1950	3900
BACK/ONLINE CONNECTION P.GUAGE	4	200	800
SWITCH DANFOSS KP 35-36	2	880	1760
IPVC FITTING COMPLETE WITHOUT WIRE	1 LOT	19500	19500
ARBON WATER WASHABLE	1050/350	11550	11550
5-63 DISTRIBUTION KIT TOP BOTTOM	2	2850	5700
ASE VALVE TOP BOTTOM 1.5"	1	250	500
PUT LINE+WASTAGE LINE COMPLETE	1	22500	22500
WELDER CHARGES	1	5000	5000
3O CIP HOUSING WITH SPUN FILTER	1	850	850
	2	1750	3,500
S HEAD CRI FLUSHING PUMP SINGLE PHASE	1	7850	7,850
AND LOCAL FREIGHT EXTRA			7,12,240

Wed Nov 20 2024


dVault
Home
Images
Files
Api
Storage
Upload
Smart Share
Search
Upload
Click to upload or drag and drop
SVG, PNG, JPG or GIF (MAX. 800x400px)
Preview

dVault

Search

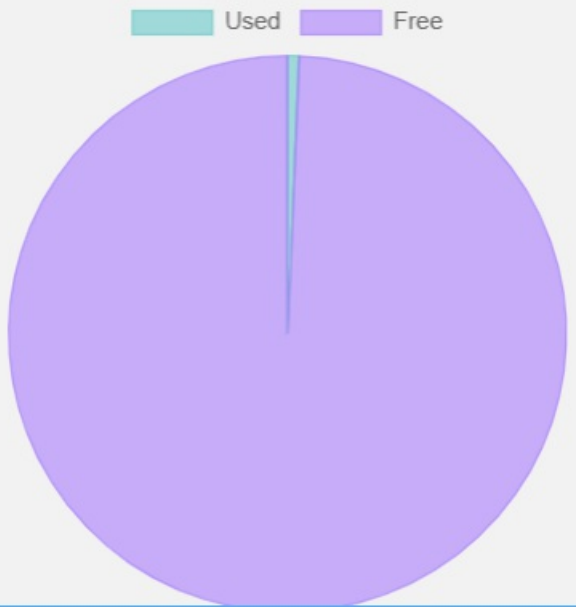
- Home
- Images
- Files
- Api
- Storage
- Upload
- Smart Share

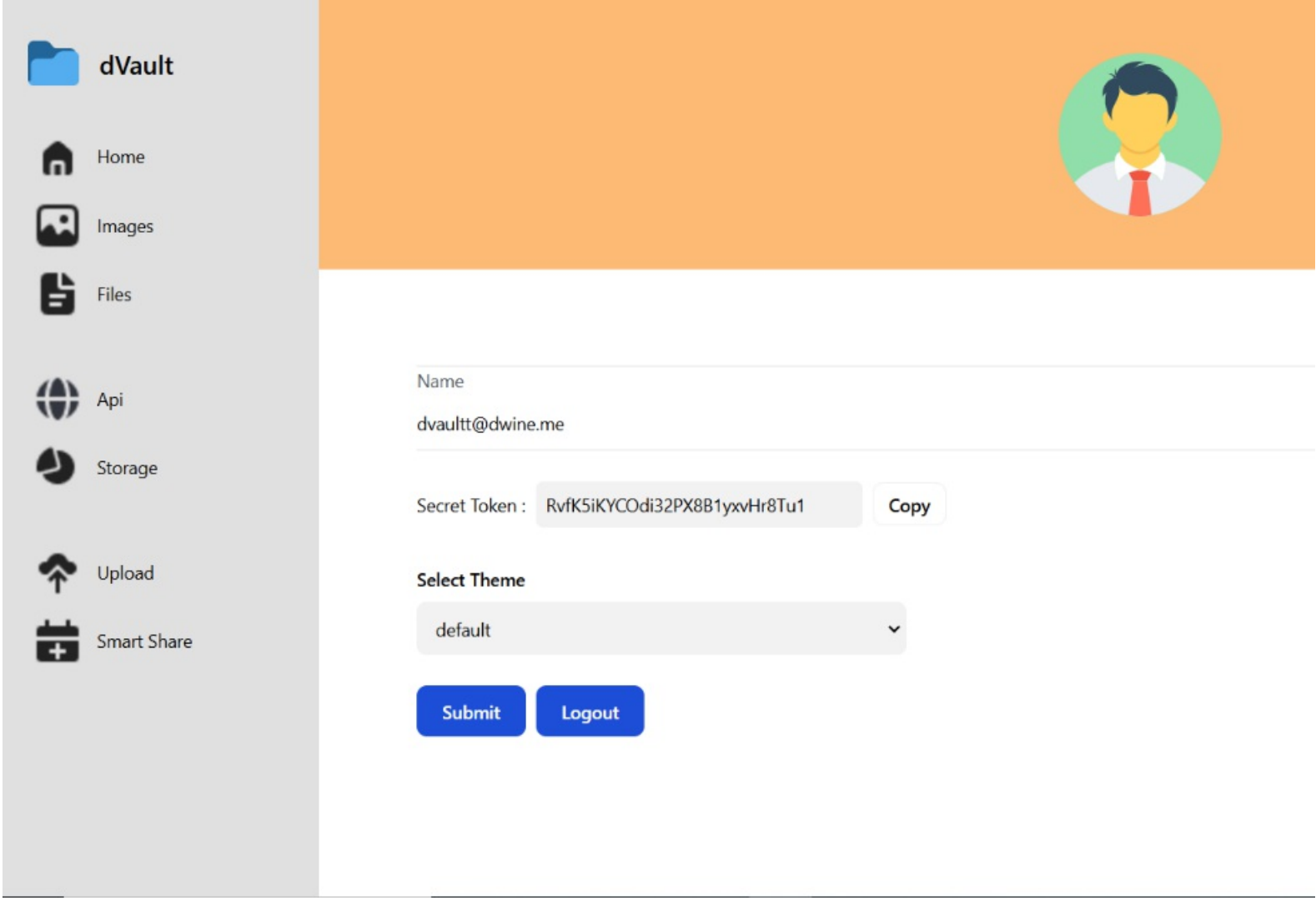
Storage

Storage Used

3.30 MB

49





Conclusion and Future Scope

The DVAULT project stands as a significant milestone in meeting the contemporary demand for a secure, user-friendly web-based storage solution. Through its successful implementation, DVAULT provides users with a streamlined platform for sharing and storing digital assets, prioritizing security, and promoting a positive user experience. The incorporation of features like SmartShare, AI-driven content organization, and robust security measures underscores the project's commitment to addressing user needs and industry best practices

Future Scope:

Exploration of Real-Time Monitoring:

One avenue for future development involves exploring real-time monitoring capabilities. Integrating mechanisms for immediate detection of file changes would enhance the platform's responsiveness to potential security threats, providing users with swift alerts and enabling proactive responses to suspicious activities.

Integration with External Security Systems:

The project's future scope extends to integration with external security systems. Collaborating with existing security infrastructure could offer a more comprehensive approach to threat detection. This integration would leverage the strengths of DVAULT and external systems, creating a synergistic defense mechanism against a diverse range of cyber threats.

Enhanced Reporting Features:

To offer users more granular insights into file integrity, enhancing reporting features is a logical next step. Future iterations could include advanced reporting capabilities with detailed analytics and visualizations, empowering users with a deeper understanding of their data security posture and facilitating informed decisionmaking.

Adaptation to Emerging Security Challenges:

As cybersecurity threats continually evolve, DVAULT is poised to adapt to emerging challenges. Future developments should consider proactive measures to stay ahead of potential threats, whether through the integration of cutting-edge security technologies, regular updates to counter new vulnerabilities, or collaboration with cybersecurity experts to address emerging trends.

Collaboration with Industry Standards:

Ensuring compatibility with evolving industry standards is paramount. Future iterations of DVAULT should align with emerging standards and regulations in the field of web-based storage and cybersecurity. This commitment to compliance ensures that the project remains relevant and applicable within the broader technological

landscape.

References

- Docs | Next.js (nextjs.org)
- Docs | Material UI React components based on Material Design (mui.com)
- Firebase Documentation (google.com)
- Cloud Functions for Firebase (google.com)
- Firestore | Firebase
- Firebase Authentication
- VIPS | Podcast
- Redux | Docs
- Tailwind CSS | Docs
- Vercel | Docs