SkyStack Cloud Vault: A Modern Cloud Storage Solution

Arnab Rai
22BDS005
DSAI
IIIT Dharwad

Md. Danish Ansari
22BDS039
DSAI
IIIT Dharwad

Project Repository: https://github.com/arnabrai/SkyStack-Cloud-Projec
Video Presentation: https://drive.google.com/drive/folders/16vlFxg9iEbSE_aH3KjY_1UQQ93Pc3cEr?usp=sharing

***Abstract*—SkyStack Cloud Vault is a modern web application providing secure cloud storage functionality. This paper presents the streamlined architecture, implementation details, and key features of the application. We discuss the technology stack, including React, TypeScript, Tailwind CSS, and Supabase, demonstrating how these modern web technologies can be leveraged to create a secure and user-friendly cloud storage solution with comprehensive file management capabilities. The paper highlights security considerations, performance optimizations, and future development directions for cloud storage applications, offering insights into effective cloud application development practices.**

***Index Terms*—Cloud Storage, React, TypeScript, Tailwind CSS, Supabase, Web Application, Security, File Management, User Experience**

I. Introduction

Cloud storage has evolved into a critical infrastructure component in our increasingly digital world. As organizations and individuals generate exponentially growing volumes of data, the need for secure, accessible, and efficient storage solutions has become paramount. According to recent industry reports, the global cloud storage market is expected to reach $297.5 billion by 2027, growing at a CAGR of 24.3% from 2020 to 2027 [1].

Traditional cloud storage solutions often face challenges related to security vulnerabilities, complex user interfaces, and limited integration capabilities. SkyStack Cloud Vault addresses these limitations by providing a modern, secure, and user-friendly cloud storage solution built using React, TypeScript, Tailwind CSS, and Supabase. The application focuses on security, usability, and performance, offering features such as advanced file management, granular sharing controls, and detailed storage analytics.

This paper presents a comprehensive analysis of the SkyStack Cloud Vault application, covering system architecture, implementation details, security considerations, and performance optimizations. We demonstrate how modern web development approaches and technologies can be effectively applied to create robust cloud storage solutions that meet contemporary needs for data security, accessibility, and management.

II. System Architecture

SkyStack Cloud Vault follows a layered architecture that promotes modularity, maintainability, and scalability. The application is divided into client-side and server-side components, with Supabase providing the backend infrastructure.

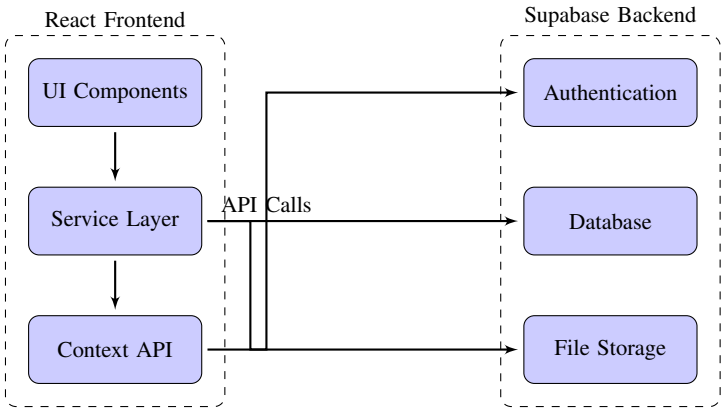

Fig. 1: System Architecture of SkyStack Cloud Vault

A. Frontend Architecture

The frontend is built using React and TypeScript with a component-based architecture that prioritizes code reusability, maintainability, and performance. The frontend architecture follows modern React best practices and leverages the latest features of React 18.

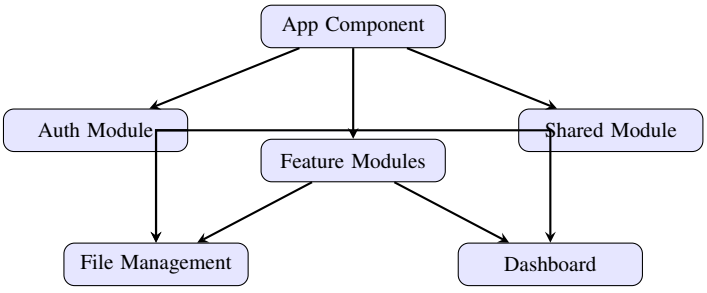

Fig. 2: Frontend Component Architecture

Key aspects of the frontend architecture include:

- **Component Hierarchy**: UI organized into atomic, reusable components following a composition-based approach

- **Context API**: Used for global state management with optimized re-rendering through selective context providers
- **TanStack Query**: Implements efficient data fetching, caching, and synchronization with automatic background refetching
- **TypeScript**: Comprehensive static typing for improved code quality, maintainability, and developer experience
- **Custom Hooks**: Encapsulates complex business logic and state management in reusable hooks
- **Lazy Loading**: Implements code splitting to optimize initial load performance

B. Backend Architecture

Supabase provides a comprehensive backend infrastructure that eliminates the need for a custom server implementation while maintaining high levels of security and scalability. The backend architecture is designed to provide real-time capabilities and automatic API generation.

- **PostgreSQL Database**: Stores user data and file metadata with advanced relational features
- **Authentication**: JWT-based user authentication and session management with support for multiple authentication providers
- **Storage**: Scalable file storage with configurable access controls and direct upload capabilities
- **Row Level Security (RLS)**: Fine-grained access control at the database level to enforce security policies
- **Real-time Subscriptions**: WebSocket-based real-time updates for collaborative features
- **Edge Functions**: Serverless functions for custom backend logic when required

This approach leverages the "Backend as a Service" (BaaS) model, reducing development time while maintaining flexibility and control over data access patterns.

III. Key Features and Implementation

A. Technology Stack Overview

SkyStack Cloud Vault is built using a modern technology stack that prioritizes performance, developer experience, and maintainability:

TABLE I: Technology Stack Components

Component	Technologies
Frontend Framework	React 18 with TypeScript
State Management	Context API, TanStack Query
UI Framework	Tailwind CSS, Shadcn UI
Backend	Supabase (PostgreSQL, Auth, Storage)
Build Tools	Vite, ESBuild
Testing	Vitest, React Testing Library, Playwright

This stack was selected for its balance of performance, development speed, and scalability. React provides a robust component model, TypeScript adds type safety, and Tailwind CSS enables rapid UI development. Supabase eliminates the need for a custom backend while providing enterprise-grade security and scalability.

B. Authentication and User Management

The authentication flow is implemented using Supabase Auth, providing secure user management with JWT-based authentication and multiple sign-in methods.

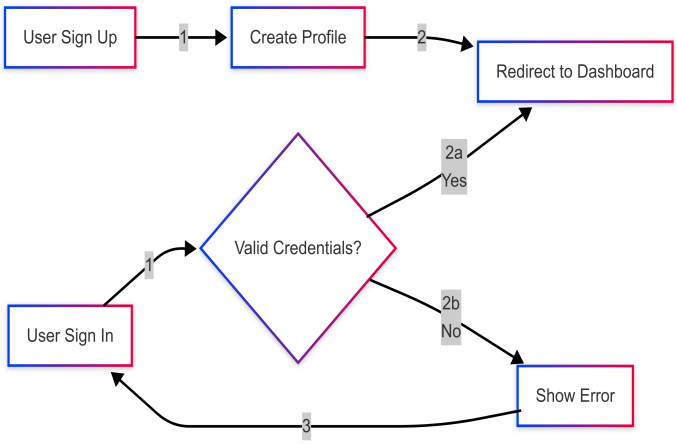

Fig. 3: Authentication and User Management

The authentication implementation uses the Supabase client:

```
// Simplified authentication implementation
import { createClient } from '@supabase/supabase-js';

const supabase = createClient(
  process.env.SUPABASE_URL,
  process.env.SUPABASE_ANON_KEY
);

// Sign up function
export const signUp = async (email, password) => {
  const { user, error } = await supabase.auth.signUp({
    email,
    password,
  });

  if (user) {
    await createUserProfile(user.id);
  }

  return { user, error };
};
```

C. File Management

The core functionality revolves around efficient file operations including upload, download, organization, and sharing.

D. Database Schema

The PostgreSQL database in Supabase stores user data, file metadata, and sharing information with the following key tables:

- **profiles**: User profile information
- **files**: File metadata (name, path, type, size)
- **folders**: Folder structure information
- **shared_files**: File sharing permissions and details

```
-- Files table with RLS policy
CREATE TABLE public.files (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  name TEXT NOT NULL,
  file_path TEXT NOT NULL,
  file_type TEXT,
  size BIGINT,
  user_id UUID NOT NULL REFERENCES auth.users(id),
  parent_folder_id UUID REFERENCES public.folders(id),
```

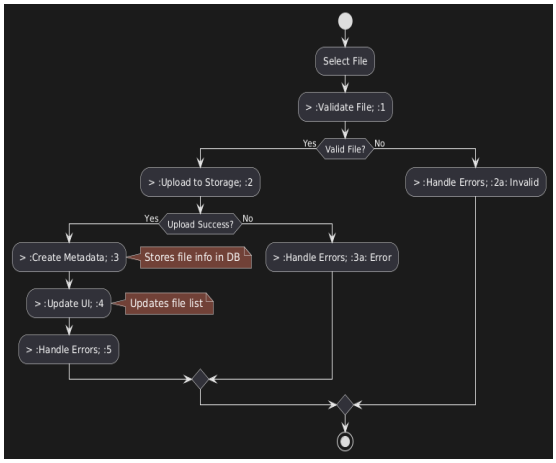

Fig. 4: File Management

```
  created_at TIMESTAMPTZ DEFAULT now()
);

-- RLS policy for files
CREATE POLICY "Users can access their own files"
ON public.files
USING (auth.uid() = user_id);
```

E. File Sharing Implementation

The file sharing functionality allows users to share files with customizable permissions:

```
// Simplified shareFile function
export const shareFile = async (fileId, email, accessLevel = 'view', expiryDate =
      null) => {
  try {
    // Create sharing record
    const { data, error } = await supabase
      .from('shared_files')
      .insert({
        file_id: fileId,
        shared_with: email,
        access_level: accessLevel,
        expires_at: expiryDate
      });

    if (error) throw error;

    // Update file's shared status
    await supabase.from('files').update({ shared: true }).eq('id', fileId);

    return { success: true };
  } catch (error) {
    return { success: false, error: error.message };
  }
};
```

F. Storage Analytics

The application provides visual analytics of storage usage with breakdowns by file type using Recharts:

```
// Storage stats visualization with Recharts
const StorageStats = ({ usedStorage, totalStorage, breakdown }) => (
  <div className="space-y-4">
    <div className="flex justify-between text-sm">
      <span>Storage Used</span>
      <span>{formatBytes(usedStorage)} of {formatBytes(totalStorage)}</span>
    </div>

    <Progress value={(usedStorage / totalStorage) * 100} className="h-2" />

    <PieChart width={200} height={200}>
      <Pie
        data={breakdown}
        dataKey="size"
        nameKey="label"
        cx="50%"
        cy="50%"
        outerRadius={60}
      >
        {breakdown.map((entry, index) => (
          <Cell key={index} fill={entry.color} />
        ))}
      </Pie>
      <Tooltip formatter={(value) => formatBytes(value)} />
    </PieChart>
  </div>
);
```

IV. Security Considerations

Security is a critical aspect of SkyStack Cloud Vault, with several measures implemented:

- **Authentication**: Secure JWT-based authentication
- **Authorization**: Row Level Security policies for data access control
- **Data Privacy**: Isolated file storage with user-specific paths
- **Input Validation**: Client and server-side validation
- **Secure File Sharing**: Controlled sharing with expiration dates

V. Performance Optimization

Performance optimization is a critical aspect of SkyStack Cloud Vault, ensuring a responsive and efficient user experience even when handling large files and numerous operations.

A. Frontend Performance Optimization

Several strategies are employed to optimize frontend performance:

- **Code Splitting**: Implements dynamic imports to reduce initial bundle size
- **Memoization**: Uses React.memo and useMemo to prevent unnecessary re-renders
- **Virtualized Lists**: Implements windowing for file lists with react-window
- **Lazy Loading**: Defers loading of non-critical components
- **Image Optimization**: Implements responsive images and WebP format

B. Backend Performance Optimization

On the backend, several optimizations ensure efficient data retrieval and storage:

- **Database Indexing**: Strategic indexes on frequently queried columns
- **Query Optimization**: Optimized SQL queries with efficient JOINs
- **Connection Pooling**: Efficient database connection management
- **Caching**: Implementation of cache layers for frequently accessed data
- **Chunked Uploads**: Breaking large file uploads into manageable chunks

VI. Acknowledgements

We would like to express our sincere gratitude to our Pro- fessor Animesh Chaturvedi for their invaluable guidance, constant support, and insightful feedback throughout the development of this project. Their expertise and encouragement have been instrumental in refining our ideas and approaches. We are also grateful to our college, IIIT Dharwad, for providing the resources and platform that enabled us to carry out our work.

VII. Conclusion

SkyStack Cloud Vault demonstrates the effective application of modern web technologies to create a secure, user-friendly cloud storage solution. The project showcases how React, TypeScript, Tailwind CSS, and Supabase can be combined to build a scalable and maintainable application with a focus on security, performance, and user experience.

The modular architecture with clear separation of concerns enables

References

[1] Fortune Business Insights, "Cloud Storage Market Size, Share & COVID-19 Impact Analysis," Market Research Report, 2021.

[2] Facebook Inc., "React: A JavaScript library for building user interfaces," [Online]. Available: https://reactjs.org/. [Accessed: 10-Apr-2025].

[3] Microsoft Corporation, "TypeScript: JavaScript with syntax for types," [Online]. Available: https://www.typescriptlang.org/. [Accessed: 10-Apr-2025].

[4] Tailwind Labs, "Tailwind CSS: A utility-first CSS framework," [Online]. Available: https://tailwindcss.com/. [Accessed: 10-Apr-2025].

[5] Supabase Inc., "Supabase: The open source Firebase alternative," [Online]. Available: https://supabase.io/. [Accessed: 10-Apr-2025].

[6] S. Drasner, "Shadcn UI: Re-usable components built using Radix UI and Tailwind CSS," [Online]. Available: https://ui.shadcn.com/. [Accessed: 10-Apr-2025].

[7] T. Tannerlinsley, "TanStack Query: Powerful asynchronous state management," [Online]. Available: https://tanstack.com/query/. [Accessed: 10-Apr-2025].

[8] PostgreSQL Global Development Group, "Row security policies," in PostgreSQL Documentation, 2022. [Online]. Available: https://www.postgresql.org/docs/current/ddl-rowsecurity.html.

[9] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC 7519, Internet Engineering Task Force (IETF), May 2015.

[10] C. Modi et al., "A survey on security issues and solutions at different layers of Cloud computing," The Journal of Supercomputing, vol. 63, no. 2, pp. 561-592, 2013.