International Journal of User-System Interaction 16 (4) 2023, 103-116

Efficient Sharing and Storing System

Bogdan Draghici, Cristian Mihaescu, Paul Stefan Popescu

University of Craiova, Romania
e-mail: draghici.bogdan.h6u@student.ucv.ro, {cristian.mihaescu, stefan.popescu}@edu.ucv.ro,

Abstract. An effective application for storing and sharing files must offer an intuitive and user-friendly interface that allows users to upload, organize and share their files without difficulty. This involves upload and storage functionality, allowing users to upload different types of files (documents, images, videos, etc.) and organize them into logical structures such as folders. At the same time, users should be able to share files with other people by generating access links or providing direct access to other application users, with control over permissions for each shared file or folder. Another concern in developing such an application is data security. Thus, the application must protect uploaded data adequately, including encryption and access control. There must also be effective search and organization functionality so that users can quickly and easily find the desired files. To ensure accessibility and ease of use for all users, the app should offer compatibility with various devices and operating systems, including the option for dark mode. It is also essential that the app interface highlights links and action buttons, making it easier for users to navigate and interact. In addition, to cater for an international audience, the app should offer the possibility to select the preferred language so that users can access and use the app in their native language. Regarding time management, information should be displayed to users in their local format.

Keywords: Software engineering, storing technologies, user interface.

DOI: 10.37789/ijusi.2023.16.4.1

1. Introduction

The aim of this project is to develop a robust and easy-to-use software application for efficient storage and distribution of digital files, addressing users' needs for organization, security, and collaboration of their data.

By implementing a complete system that allows uploading, managing, and sharing files and ensuring security and privacy standards, this project aims to provide a centralized and reliable solution for managing digital data, helping optimize work processes and improve users' efficiency in handling and sharing information.

The motivation for this project stems from the desire to provide a simple and efficient alternative to other complex file storage and distribution platforms. Based on a personal need to facilitate the sending of files and images between friends and family, this project aims to create an easy-to-use application that provides a centralized solution for managing and sharing digital files in a secure and convenient way.

With the increasing demand for online collaboration and concerns about data security, there is a clear need for a solution that integrates file upload, storage, sharing, and security functionality. Thus, this project aims to respond to this demand by providing an accessible and user-friendly platform that brings a simpler and more efficient alternative to digital file management.

2. Tools and technologies

MySQL referenced by (Christudas et. al. 2019) and Azure SQL were used to develop the solution, which is a database management solution.

MySQL, ideal for small and medium-sized projects, is stable and easy to use, making it perfect for web and business applications.

Azure SQL, a cloud service from Microsoft, is suitable for applications that require scalability and simplified management in the cloud.

Both use SQL for queries and offer data management and user access control functionality. The main difference is that MySQL is free and requires manual administration, while Azure SQL (Das et. al. 2019) is a paid service, eliminating the need for manual administration and offering built-in scalability and security.

2.1 Azure Platform

Azure (Verma et. al 2019) is a cloud platform offered by Microsoft that provides a wide range of services for developing, deploying and managing applications in the cloud. It includes computing, storage, database, analytics, and security services.

Azure also offers management and monitoring tools for performance optimization and support for multiple programming languages, platforms and technologies. The platform enables flexible deployments, including public, private and hybrid clouds.

2.2 Azure NuGet Packages

Several Azure packages have been used to integrate and deploy the development environment in the Azure cloud, ensuring efficient and secure management of email services, configuration secrets, and file storage. In this project, we used the following NuGet packages:

- Azure.Communication.Email for sending emails via Azure Communication Services. This package simplifies the integration of email functionality into .NET applications without the need to configure external SMTP servers (Hu et. al. 2024).
- Azure.Extensions.AspNetCore.Configuration.Secrets facilitate secure access to secrets stored in the Azure Key Vault, providing secure management of confidential data in ASP.NET Core (Whitesell et. al. 2022) applications.
- Azure.Storage.Blobs are used to interact with the Azure Blob Storage service to store and manage files. This package provides functionality for handling blobs, making it easy to integrate scalable and secure storage capabilities into .NET applications.

3. Proposed approach

3.1 Database structure

The database was created using the code-first programming technique. In this approach, classes representing entity types are created first, and tables in the database are generated or modified by adding migrations using the EF Core package and updating the database.

Figure 1: Database structure

Over the course of its development, the database has undergone many changes and now consists of 10 tables, with most of the links revolving around the User and StoredFile tables.

The user table stores information about users, such as user ID, role, tag, email, confirmation, password, last password change date and salt.

StoredFile stores information about uploaded files, including file ID, owner ID (the user who uploaded the file), ID of the virtual directory where the file is located, file name, access level, creation date, modification date, size in bytes, content type, and whether the file is deleted.

3.2 Backend application architecture

The backend solution for the application was built using the Visual Studio 2022 Strauss (2019) development environment and was divided into six projects: API, Caching, Core, Infrastructure, and Infrastructure.Migrations and SharedKernel. Each of these projects has a specific role.

3.2.1 SharedKernel Project

SharedKernel contains fundamental functionality and abstractions that are essential to the basic operation of the application. These are used in various

layers or modules of the application, providing a common foundation for the entire software structure. Among what this project provides are the following:

- Enumerations used within the application.
- Custom exception classes for error handling.
- Cryptography class that provides cryptographic utilities for encryption, decryption and hashing.
- Provider classes that facilitate access to configuration files and all services provided.
- A model for determining the memory used by different objects

Caching Project

Is designed to provide a reusable storage mechanism that can be used in different parts of the application to improve performance and reduce database load. By caching frequently accessed data, the application can serve requests faster and reduce the number of database queries. Contains:

- ICacheManager Interface: This interface defines the contract for caching operations, providing a consistent way of interacting with the cache regardless of the underlying cache implementation.
- CacheManager Implementation: This class provides the actual implementation of the ICacheManager interface. It uses an in-memory cache or any other cache provider to store and retrieve cached data. Configuration variables defined in the configuration file include options to disable the cache, set a maximum cache size limit, and define the default cache lifetime.
- CachingModule class: The configuration class that registers the application's distributed cache and cache storage services with the dependency injection container.

This modular approach provides centralization of caching logic, which makes it easier to manage and maintain.

3.2.1 Infrastructure Project

The Infrastructure project deals with data access and persistence logic. It contains:

- Entities: classes that represent tables in the database. Each entity class refers to a database table and contains properties corresponding to the table's columns.
- Configuration Classes: Classes that configure the mappings and relationships between entities using the Fluent API. These classes ensure the correct configuration of entities in the database.

- Repository classes: Classes that encapsulate the data access logic. They provide methods for interacting with the database, such as adding, updating, deleting and querying data. They can be divided into interfaces and implementations of interfaces.
- MyFileSpaceDbContext class: The base class in the Entity Framework that handles database connections, entity tracking, and CRUD operations. It represents a session with the database and provides methods for querying and saving data.
- SeedData Class: A class used to populate the database with key data. This is particularly useful during development and testing to ensure that the database has the data needed for the application to function correctly.
- InfrastructureModule class: A class responsible for registering repository interfaces and their implementations with the dependency injection (DI) container. This configuration ensures that repositories are injected as domain dependencies throughout the application.

3.2.3 The Infrastructure.Migrations project

This project manages database schema changes (Batini et. al. 1986) and versions using code-based migrations. It contains:

- Migration classes (Kearney et. al. 2004): automatically generated classes that define schema changes to be applied to the database. Each migration class represents a specific change or set of changes to the database schema.
- MyFileSpaceDbContextModelSnapshot: A file representing the current state of the model. It is used by the Entity Framework to detect changes in the model and generate the corresponding migrations.

The Core project is the core part of the application that contains the business logic and service implementations. It acts as a bridge between the API project and the Infrastructure project, ensuring that business rules are applied and that data is processed correctly before being passed to or from the data layer. It contains:

- Services: include the interfaces that define the contracts for the business logic as well as the implementations that contain the logic itself.
- CacheKeys class: contains constants that define the keys for caching various data entities. Centralizing these keys makes it easier to manage and avoid conflicts.
- JsonWebToken class deals with the creation and validation of JWTs

for authentication purposes and includes methods for generating tokens, validating tokens, and handling requests.

- DTOs (Data Transfer Objects): lightweight, serializable objects used to transfer data between different application layers, such as between Api and Core projects. They help to decouple layers and provide a clear contract for data exchange.
- MappingProfile class: defines how objects are mapped between different application levels, such as between entities and DTOs.
- Email templates: define the structure of the different emails sent by the application, such as account verification and password reset.
- Constants class: contains constant values used throughout the application, such as the root folder name, maximum storage memory for each user, file names for email templates, and other globally applicable constants.
- Specification classes encompass the criteria for querying data in the database, which help to abstract complex queries and reuse them in different services.
- IStorageManager interface and its implementations: provides an abstraction for file storage operations, allowing different storage providers (in our case, local file system and cloud storage in an Azure storage account).
- Session class: represents a user session and stores session-related information, such as login status, identifier, and user role.
- CoreModule class configures Core project services to be injected into other parts of the application. Ensures that service interfaces are mapped to their implementations. Depending on the runtime environment, provides different implementations for the IStorageManager interface.

3.3 Frontend application architecture

The structure of the frontend solution source folder is composed of the following components:

- The app folder: This contains all the components and modules that are part of the application framework. Within it is the application module with its configurations, the module routing with the paths to the application pages, the `app-component` directory containing the first Angular component to load into the application, and a directory for each module.
- The assets folder is the directory containing JS, audio, SCSS, icons, images, and JSON files for language internationalization.
- The environments folder: Here are files with classes configured for

the project environment.

- The theme folder contains the scss file where the project theme is configured.
- The global.scss file defines and applies styles that affect all HTML files of the components.
- The index.html file is responsible for the programmatic rendering of all application content.
- The main.ts file is responsible for creating the environment in which the application will run.

The structure of a module

Each module of the application is used for a specific functionality of the application. Each application module folder has a specific structure, with the following:

- The components folder contains components that can be added to a page. These must be declared within the module class file in order to be used in the application.
- The models' folder contains model classes enumerable or module-specific constants used for data transfer between components or for communication with the backend application.
- The pages folder contains module-specific pages. For some modules, this folder may be missing because they do not contain pages that can be displayed.
- Directives folder: This contains module-specific directives that can perform various functions in the background. This folder may be missing in some modules.
- The services folder contains one or more services used to retrieve information, usually from the backend application, by accessing access points.
- The *-routing.module.ts file: This file contains the paths to the pages that can be accessed within the module. In the case of a module that has no pages that can be displayed, this file will be missing.
- The *.module.ts file: This is where all components and pages implemented in the module are declared, declared components that can be used in other modules are exported, and modules from which certain components are to be used are imported.

3.4 Use cases

The use cases describe the functionalities and interactions between users and the file storage and distribution system.

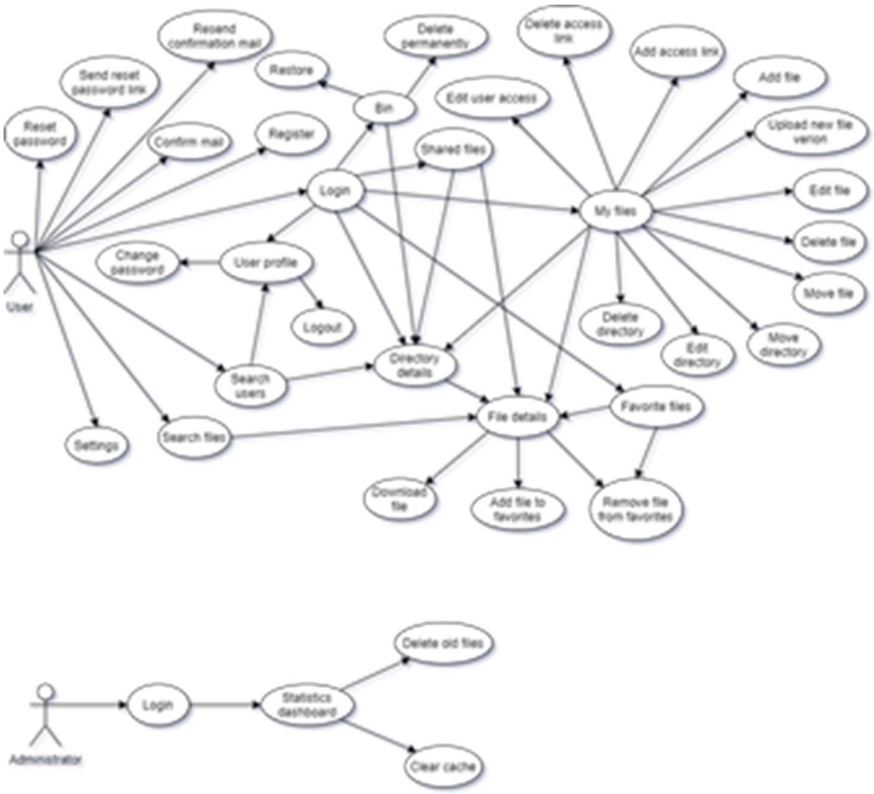

Figure 2: Use case diagram

In this diagram (Fauzan et. al. 2019), there are two types of actors: User and Administrator, each with distinct sets of use cases.

The diagram details the functionality available to users and administrators, showing how each role interacts with different aspects of the system to accomplish the associated tasks.

4. Experimental results

4.1 Displaying files and folders

Both files and folders will have specific icons, with more diversity for files depending on the type of content (photo, video, text document, archive, etc.). Also, each icon will have a color indicating the level of access:

- Private (purple): Only the owner has access.
- Restricted (blue): Accessible to the owner, those using the access link and authenticated users designated by the owner.
- Public (green): Available to anyone using the application without the need for authentication.

4.2 Hierarchical Display

In this mode, files are only visible within the currently accessed folder. Items corresponding to a folder or file will only display their specific icon and name.

Figure 3: Hierarchical view

A section available at the top of the interface will indicate to the user their current place in the folder structure, allowing them to navigate between its levels by selecting the desired folder names.

Figure 4: Navigation bar

Also, if the user clicks on the file name, they will be able to see the full name and size of the file, providing additional information about the contents of the file.

Figure 5: Tooltip file details

4.3 List Display

In this mode, all files are visible, and the user has the option to display all folders, depending on preference. There is also a function to filter files and folders by name.

Files will be listed by name and size, giving a clear picture of their contents.

Folders will be displayed with the name and date of the last modification, making it easy to identify and manage them as needed.

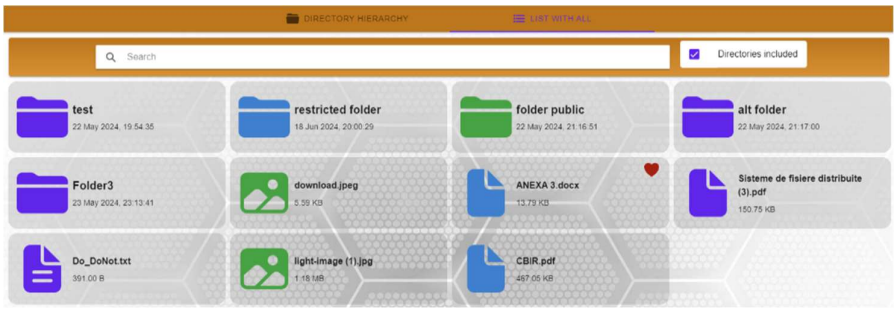

Figure 6: List view

4.4 Folder and file operations

Operations on folders and files can be performed using intuitive methods. On the desktop, users can right-click on the desired item to access the

available options, while on mobile phones, these can be accessed by holding down the item.

To improve accessibility, each option in the text menu will have a distinct font color, highlighting when the user hovers the mouse over the option.

It is important to note that all operations that involve modifying folders or files are only available to their owners. Other users can only view, mark as a favorite, or download.

4.5 User access management

When the owner wants to edit the user permissions, a modal will open where they can manage users. Here, the owner can search for users he wants to add to the list of users with permissions. Users who already have permissions will be displayed in orange in the "Allowed users" section, while those selected for addition will be highlighted in green. Those that the owner wishes to remove from the list of users with permissions will be added to the "Delete user access" list and will be highlighted in red. Once the owner has finished adding or removing users, they can save the changes to update the list of users with permissions.

Search functionality

On the search page, any visitor can search for files or users by name. Logged-in users have the option to include personal files as well as their own person in the search.

Figure 7: Search options

The files will appear as colored cards depending on the level of accessibility. Clicking on a card will open the details menu.

Figure 8: Browse files

Users will also appear as cards, which will have two specific buttons, one for viewing the user profile and another for viewing the user's collection.

Figure 9: Browse users

5. CONCLUSION

This paper tackled the problem of efficiently storing and sharing files. The system is built on the cloud using Azure technologies and provides a multitude of functionalities that enhance the user experience. The application was built not only to be user-friendly but also to target a large audience that is internationalized, enabling users to select their language and their timezone. The development pattern allows us to improve the system further in order to keep it up to date as new functionalities will have to be implemented

References

Christudas, B., & Christudas, B. (2019). MySQL (pp. 877-884). Apress.

Verma, A., Malla, D., Choudhary, A. K., & Arora, V. (2019, February). A detailed study of the Azure platform & its cognitive services. In *2019 International Conference on machine learning, Big Data, cloud and Parallel Computing (COMITCon)*, 129-134. IEEE.

Hu, L. (2024). A Design of an SMTP Email Server. *Journal of Electronic Research and Application*, 8(4), 41-47.

Whitesell, S., Richardson, R., Groves, M. D., Whitesell, S., Richardson, R., & Groves, M. D. (2022). ASP. NET Core Overview. Pro Microservices in. NET 6: With Examples Using ASP. *NET Core 6, MassTransit, and Kubernetes*, 29-49.

Das, S., Grbic, M., Ilic, I., Jovandic, I., Jovanovic, A., Narasayya, V. R., ... & Chaudhuri, S. (2019, June). Automatically indexing millions of databases in Microsoft Azure SQL database. In *Proceedings of the 2019 International Conference on Management of Data*, 666-679.

Fauzan, R., Siahaan, D., Rochimah, S., & Triandini, E. (2019, July). Use case diagram similarity measurement: A new approach. In *12th International Conference on Information & Communication Technology and Systems* (ICTS), 3-7. IEEE.

Batini, C., Lenzerini, M., & Navathe, S. B. (1986). A comparative analysis of methodologies for database schema integration. *ACM computing surveys (CSUR)*, 18(4), 323-364.

Kearney, M., & Beserra, B. (2004). Introduction: Migration and identities—a class-based approach. *Latin American Perspectives*, 31(5), 3-14.

Strauss, D. (2019). *Getting Started with Visual Studio 2019: Learning and Implementing New Feature*s. Apress.

.