

National College of Ireland

BSc (Honours) in Computing

Cyber Security

2023/2024

Michael Greed

20142269

X20142269@student.ncirl.ie

Passwordless Post-Quantum Resistant File Storage and Cloud Backup Application Technical Report

Contents

Executive Summary	5
1.0 Introduction	5
1.1. Background	5
1.2. Aims	6
1.3. Technology	7
1.4. Structure	8
2.0 System	8
2.1. Requirements	8
2.1.1. Functional Requirements	9
2.1.1.1. Use Case Diagram	9
2.1.1.2. Requirement 1: Register account	9
2.1.1.3. Description & Priority	9
2.1.1.4. Use Case	9
2.1.1.5. Requirement 2: Login	11
2.1.1.6. Description & Priority	11
2.1.1.7. Use Case	11
2.1.1.8. Requirement 3: Encrypt files at rest	12
2.1.1.9. Description & Priority	12
2.1.1.10. Use Case	12
2.1.1.11. Requirement 4: Decrypt files at rest	13
2.1.1.12. Description & Priority	13
2.1.1.13. Use Case	13
2.1.1.14. Requirement 5: Back up files	15
2.1.1.15. Description & Priority	15
2.1.1.16. Use Case	15
2.1.1.17. Requirement 6: Restore files from backup	16
2.1.1.18. Description & Priority	16
2.1.1.19. Use Case	16
2.1.1.20. Requirement 7: Manage user data	17
2.1.1.21. Description & Priority	17
2.1.1.22. Use Case	17
2.1.1.23. Requirement 8: Recover user account	18

2.1.1.24.	Description & Priority	18
2.1.1.25.	Use Case.....	18
2.1.1.26.	Requirement 9: Logout	19
2.1.1.27.	Description & Priority	19
2.1.1.28.	Use Case.....	19
2.1.2.	Data Requirements	20
2.1.3.	User Requirements.....	21
2.1.4.	Environmental Requirements	22
2.1.5.	Usability Requirements	23
2.2.	Design & Architecture	24
2.3.	Implementation	25
2.3.1.	Main algorithms	25
2.3.2.	Classes	26
2.3.3.	Functions	28
2.4.	Graphical User Interface (GUI)	42
2.5.	Testing	45
2.5.1.	Objectives.....	45
2.5.2.	Scope	45
2.5.3.	Test Strategies	45
2.5.4.	Test Environment	45
2.5.5.	Test Cases	46
2.5.6.	Automated testing	59
2.6.	Evaluation	59
2.6.1.	Functional evaluation.....	59
2.6.2.	Performance evaluation.....	60
2.6.3.	Scalability	61
2.6.4.	Challenges faced	61
2.6.5.	User Experience	61
2.6.6.	Results and outcomes	61
2.6.7.	Limitations.....	61
3.0	Conclusions	62
4.0	Further Development or Research	62
5.0	References	62
6.0	Appendices.....	64
6.1.	Project Proposal	64
6.1.1.	Objectives.....	64

6.1.2.	Background	64
6.1.3.	State of the Art	65
6.1.4.	Technical Approach	65
6.1.5.	Technical Details	66
6.1.6.	Special Resources Required	67
6.1.7.	Project Plan	67
6.1.8.	Testing	69
6.2.	Reflective Journals	70
6.2.1.	October 2023	70
6.2.1.1.	What?	70
6.2.1.2.	So what?	70
6.2.1.3.	Now what?	71
6.2.2.	November 2023	71
6.2.2.1.	What?	71
6.2.2.2.	So What?	72
6.2.2.3.	Now What?	72
6.2.3.	December 2023	73
6.2.3.1.	What?	73
6.2.3.2.	So What?	73
6.2.3.3.	Now What?	74
6.2.4.	January 2024	74
6.2.4.1.	What?	74
6.2.4.2.	So What?	75
6.2.4.3.	Now What?	75
6.2.5.	February 2024	75
6.2.5.1.	What?	75
6.2.5.2.	So What?	75
6.2.5.3.	Now What?	76
6.2.6.	March 2024	76
6.2.6.1.	What?	76
6.2.6.2.	So What?	77
6.2.6.3.	Now What?	77
6.2.7.	April 2024	77
6.2.7.1.	What?	77
6.2.7.2.	So What?	78
6.2.7.3.	Now What?	78

6.2.8.	May 2024.....	78
6.2.8.1.	What?	78
6.2.8.2.	So What?	79
6.2.8.3.	Now What?.....	79
6.2.9.	June 2024	79
6.2.9.1.	What?	79
6.2.9.2.	So What?	79
6.2.9.3.	Now What?.....	79
6.2.10.	June 2024	80
6.2.10.1.	What?.....	80
6.2.10.2.	So What?	80
6.2.10.3.	Now What?	80

Executive Summary

The main goal of this project is to create easy-to-use passwordless file storage and cloud backup application designed primarily for Windows 11 users. With cyber threats arising from post-quantum computing as the core driving motivator when planning on this application, it features OWASP recommended novel cryptographic technologies to protect from these risks in the rapidly evolving digital environment.

Due to the growing cyber risks, also the demand for protective solutions has increased in the recent years. Such solutions are particularly necessary because of the data privacy and protection requirements set by General Data Protection Regulation (GDPR) and the normalized remote working business environments. For data in transit the algorithms are still not mature enough in terms of years of experience using them so they may have still undiscovered flaws that could get exploited by the malicious parties. However, with the help of post-quantum resistant algorithms such as AES-256 for the data at rest confidentiality and integrity can be achieved even in the future. Furthermore, terminating password-based authentication from the login process enhances security posture of personal or business application users.

The design and implementation of the application especially promotes usability and user experience. Graphical interfaces of the application are easy-to-use and approachable even for the users without technical background.

The application aims to be relatively cost-efficient, low entry, and scalable cyber security solution. Additionally, the project provides good foundation for further research and development when the post-quantum algorithms have matured more.

1.0 Introduction

1.1. Background

Having strong background in IT, HR, and marketing, I have always been interested in the latest technology, information security, and finding ways to help people by providing them with user-friendly solutions. Big companies have had the same goal but sometimes there are malicious parties who would like to ruin those ambitions or get personal gains. Data leaks and cyber threats have become too common and even criminals who do not have technical skills can purchase cyber attacks as service from the black markets. Producing ethically viable and easily downloadable and installable solution to protect individual users, remote workers or businesses from cyber threats was the main motivational factor behind this application project.

This project produces passwordless post-quantum secure file storage and cloud backup solution for Windows 11 users. Implementing sustainable cryptographical solution using the latest framework which enables cross-platform development through this novel product is remarkable step for the adoption of post-quantum resistant cryptography in the larger scale. This project seemed like a perfect opportunity to learn and demonstrate my skills learned through the four years in the college.

Above all, this project let me research, design and implement the cutting-edge technologies and explore cryptography deeper under the surface. This project may even have commercial potential if developed further with investments and commercialized.

1.2. Aims

One of the major goals of this project is to create a passwordless post-quantum resistant file storage and cloud backup application for Windows users. More profoundly, there were four key objectives to be achieved which are protecting against quantum computing threats for data at rest, avoiding password-related risks, creating user-friendly and visually appealing application, and ultimately providing the users with a secure cloud backup solution for increased cyber safety guaranteeing availability of their data.

The application would leverage National Institute of Standards and Technology (2023) recommended performant AES-256 with GCM (Galois/Counter Mode) as recommended by OWASP (2024) as a best security practice in the process of encryption. To ensure security for files at rest envelope encryption is implemented in a similar manner as described by Amazon Web Services, Inc. (2023a) and by utilizing envelope encryption through AWS Key Management Service (2024a). This aims to guarantee the CIA triad data confidentiality, integrity, and authenticity. This proactive approach aimed to protect user data against hackers who may get access to the computer of a remote worker and then steal their data. This solution would leave the attackers empty handed as the data has been strongly encrypted.

Recognizing the inherent vulnerabilities associated with password-based authentication, the project also focused on integrating novel Supabase (Google Firebase competitor) (2024a), a secure and convenient, partly passwordless authentication system, through one time password code of 6 digits which is sent to the registered and authenticated user by email (Supabase, 2024b). This method helps to significantly reduce the risk of phishing attacks and streamline the user experience by eliminating the need for remembering and managing complex passwords. Resend (2023) is used as a SMTP service to send HTTPS secured email containing the code and Amazon Lightsail (2024b) virtual private server with Nginx and autorenewing Let's encrypt TLS certificate to act as redirection point for new account verifications.

Understanding the importance of accessibility, a core principle of the project was to develop an intuitive and user-friendly interface. This meant prioritizing clear instructions, simple interactions, and a design that catered to both technical and non-technical users. Empowering users to take control of their data security without technical hurdles was one of the cornerstones of the design philosophy of the project.

Integrating secure cloud backup with Amazon S3 aimed to offer an additional layer of protection against data loss. This redundancy ensured that even in case of device failure or accidental deletion, user data remained safe and accessible. This feature catered to the increasing reliance on cloud-based storage and the need for robust disaster recovery capabilities.

By achieving these objectives, the project aspired to create a user-friendly and future-proof solution that empowers Windows users to prioritize their data security. It aimed to contribute to a digital landscape where technology serves as a tool for user empowerment rather than a security barrier.

1.3. Technology

At the heart of robust security lies post-quantum resistant cryptography. Initially, I had opted for the standardized and meticulously analysed CRYSTALS-KYBER, a lattice-based key encapsulation mechanism, to safeguard data communication and encryption but due to the limitations of my present competence and inactive libraries, such as OpenSSH, in adapting this novel technology, I could not straightforwardly use it. Therefore, I decided to protect users by the means of effective local encryption and backup solution. Even if the data would be harvested by possible, even a quantum computer possessing malicious party, the data would be encrypted with the strongly recommended NIST algorithm. This future-proof choice ensures resistance against potential attacks from quantum computers, a looming threat to traditional cryptography.

Eliminating old password-based authentication is crucial for enhanced security because of the growing calculation power of quantum computers. Supabase OTP (One Time Password) is a passwordless alternative replacing traditional passwords after initial mandatory registration with email and password. This reduces the possible cyber attack surface, makes user experience simpler, and removes the necessity of having to remember passwords for the service.

The design of this application has been planned prioritizing user experience. C# and the new open-source .NET MAUI Framework provide a familiar and robust development environment for building the core functionalities and graphical user interfaces of the application. Their extensive selection of compatible NuGet libraries and strong community support ensure efficient development and long-term maintainability. Moreover, user-centered design principles will guide the interface development, emphasizing clarity, intuitiveness, and accessibility for users of all technical backgrounds. Prioritizing ease of use and clear instructions will be essential.

For secure cloud backup and enhanced reliability, Amazon (2024c) Simple Storage Service S3 will be utilized. This cloud storage service boasts robust security infrastructure and server-side encryption with also AES-256, ensuring data confidentiality at rest even in cloud. Additionally, its scalability caters to the growing data storage needs of users.

To ensure efficient management and delivery, I will adopt the agile Kanban methodology. This approach emphasizes continuous improvement, flexibility, and iterative development. Kanban boards help keeping on track with the features to be implemented and other scheduled tasks. The development time is cut remarkably because of this while still guaranteeing the agile updates and deployment of the application.

1.4. Structure

This technical report outlines the development of a passwordless post-quantum resistant file storage and cloud backup application for Windows users, aiming to address the growing need for robust cybersecurity and user-friendliness in data management.

The system section delves deeper into the specifications of the system. It starts by detailing various types of requirements, ranging from functionalities described through use cases to data structures, user needs, environmental constraints, and usability requirements. The design and architecture are then explained, showcasing the components of the system, algorithms, and demonstrating with a high-level architecture diagram. Implementation details, including key algorithms, classes, and code snippets, are presented in this section as well. The user interface is explored through screenshots and explanations of their functionalities. Finally, the testing process and its results, covering unit, integration, and end-user testing, are explained. The last evaluation part of the section summarizes how the system was assessed and presents performance metrics, scalability, correctness, and other relevant findings.

The report continues by discussing the strengths, limitations, advantages, and disadvantages of the project in the conclusion section. The conclusion offers a balanced perspective on the achievements and potential areas for improvement of the project outcome.

Further development and research section explores potential future directions for the project, considering additional resources and time. It proposes enhancements, expansions, or new research avenues that could build upon the current work.

The report concludes with a list of references used throughout the document and relevant appendices which are the project proposal and monthly reflective journals during this project.

2.0 System

2.1. Requirements

System requirements are preliminary and will become more precise when implementing the application functionalities. All the data in transit connections will use currently safe TLS-based HTTPS connections. Even if the data may be susceptible to harvest now, decrypt later attacks, the sent files would be encrypted and the malicious party does not have access to the original plaintext data key.

2.1.1. Functional Requirements

2.1.1.1. Use Case Diagram

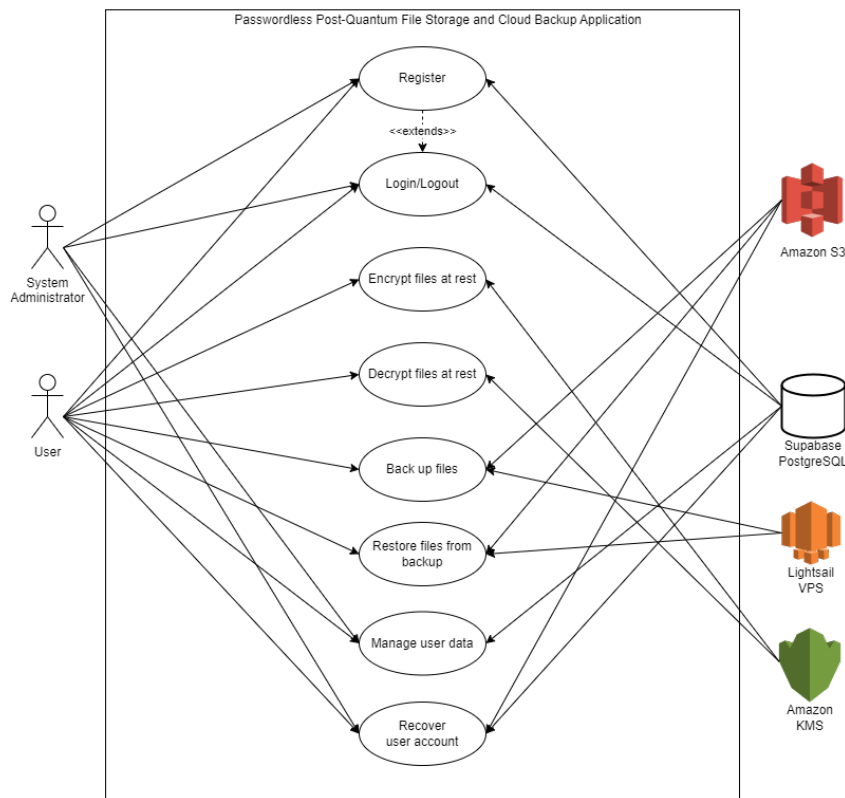


Figure 1 Use Case Diagram

2.1.1.2. Requirement 1: Register account

2.1.1.3. Description & Priority

A user can only use the application with a registered account. Registration is mandatory.

Priority: High. Without a user account the user will not have access to any of the application features.

2.1.1.4. Use Case

UC-1

Scope

The scope of this use case is to define a new user account to authenticate and authorize access to the user-specific resources and for account recovery.

Description

This use case describes the process of a user registering a new account into the application.

Flow Description

Precondition

The user is not logged into the system and does not have previously registered account.

Activation

This use case starts when a user clicks on the application icon or executable and opens the application.

Main flow

1. The user opens the application.
2. The system displays the login view.
3. The user clicks Register here link to navigate to the registration view (See A1).
4. The user enters their email, password and confirms their password in a separate field.
5. The system validates the email, password and confirm password fields.
6. The user clicks Register button (See E1).
7. The system creates a new entry into PostgreSQL database in Supabase.
8. The system sends a verification email to the email they filled into the system when registering.
9. The system redirects the user to the login view.
10. The user clicks the link in the verification email.

Alternate flow

A1 : User uses incorrect email or password.

1. The user enters an incorrect email, password, not matching password in the confirm password field or empty credentials.
2. The system validates the email, password and confirm password fields.
3. The system displays an error message, keeps the information entered before in the fields but prompts the user to try again and does not let the user register into the application.

Exceptional flow

E1 : System gets into an error state.

1. Due to database connection error, the system is unable to create an account.
2. The system catches the error and displays an error message to the user with OK button.
3. The user clicks OK button or closes the window.
4. The system displays registration view to the user.

Termination

The system presents the login view of the application for the registered user.

Post condition

The system goes into a wait state.

2.1.1.5. Requirement 2: Login

2.1.1.6. Description & Priority

A user must authenticate through Supabase REST API with an existing user account to access and use the application.

Priority: High. Without a successful authentication login event, a user cannot view any of their files or use any of the application features.

2.1.1.7. Use Case

UC-2

Scope

The scope of this use case is to define a single login method using a user account and Supabase REST API, restricting unauthorized access to the data and functionalities of the application.

Description

This use case describes the process of a user logging into the application.

Flow Description

Precondition

The user is not logged into the system. The user has created a user account into Supabase and verified their registration email. The user has been registered into Amazon S3 and Amazon KMS.

Activation

This use case starts when the user clicks on the application icon or executable and opens the application or when a user has already opened the application and registered a new account.

Main flow

1. The user opens the application.
2. The system displays the login window.
3. The user enters their email and clicks Login (See A1).
4. The system sends a passwordless OTP to the user email for login.
5. The system displays the OTP view in which the user can enter the code to start a session.
6. The user enters the code (See A2).
7. The success message is displayed.
8. The system starts the user session utilizing JWT token with reasonable expiry time (10 hours).
9. The user is now logged in and the file storage folder is displayed.

Alternate flow

A1 : Incorrect or empty email address

1. The user enters an incorrect or empty email.
2. The system validates the email field.
3. The system makes an API call to Supabase and does not find the email address.
4. The system displays an error message, prompts the user to try again and does not let the user log in to the application.

A2 : Incorrect passwordless credentials

1. The user enters wrong or expired OTP.
2. The system shows a display alert requiring the user to try again.
3. The user returns to the login screen and may start the login process again.

Termination

The system presents the personal user folder or “user space” of the application for the authenticated, logged in user.

Post condition

The system goes into a wait state.

[2.1.1.8. Requirement 3: Encrypt files at rest](#)

[2.1.1.9. Description & Priority](#)

When the user decides that he has finished working with the files, they click encrypt button of the application and the application encrypts the file utilizing envelope encryption through AWS KMS and AES-256 algorithm. Root encryption and decryption key is created and saved into Amazon KMS.

Unique data key generated using Amazon KMS is saved at the local storage in its encrypted form. AES-256 GCM algorithm is used to encrypt the file using plaintext decrypted key and the initialization vector is saved together with the file and the encrypted data key.

Priority: High. Without encryption, the personal files of the user would be susceptible to a possible theft or cyber attack.

[2.1.1.10. Use Case](#)

UC-3

Scope

The scope of this use case is to ensure the confidentiality of the personal user files.

Description

This use case describes the process of the system encrypting user files.

Flow Description

Precondition

The user has created an account in the system, opened the application and logged in using Supabase OTP. The user has active AWS KMS account. The user has an active session.

Activation

This use case starts when the user selects the file they would like to encrypt (and later back up to cloud by uploading the file to AWS S3).

Main flow

1. The user selects a file that they would like to encrypt for security.
2. The system detects that the user has clicked Encrypt button.
3. The system starts encrypting the file with AES-256 encryption (See E1).
4. The system requests AWS KMS to generate data key.
5. The system receives both the plain text and encrypted data keys but only the encrypted is saved locally.
6. The system starts encrypting the file using AES-256 using the plaintext data key.
7. The system creates initialization vector and saves it locally with the file and the encrypted data key.

Exceptional flow

E1 : System gets into an error state during encryption.

1. The system encounters an unexpected error during the encryption process.
2. The system displays an error message and instructions to the user with OK button.
3. The user clicks the OK button.
4. The system returns to the user space.

Termination

The system presents the window indicating successful encryption.

Post condition

The system has finished the encryption and is in wait state.

[2.1.1.11. Requirement 4: Decrypt files at rest](#)

[2.1.1.12. Description & Priority](#)

When logging in to the application, the application shows the current encrypted and unencrypted files in the local UserFiles folder after the user selects the main folder. The user needs to decrypt the files to be able to use them.

Priority: High. Without decryption, the user would not be able to access their encrypted files. The data key decryption root key would be saved in Amazon KMS.

[2.1.1.13. Use Case](#)

UC-4

Scope

The scope of this use case is to ensure authorized access to the authenticated user to their confidentially stored personal files in the local system.

Description

This use case describes the process of a previously created user who is logged in to the system getting access to their files for editing or other relevant purposes.

Flow Description

Precondition

The user has created an account in the system, opened the application and logged in using Supabase OTP. The user has active AWS KMS accounts. The user has an active session.

Activation

This use case starts when the user selects an encrypted file in the local storage.

Main flow

1. The user clicks decrypt button in the application toolbar.
2. The system requests decrypt function from AWS KMS.
3. The system starts the decryption process using KMS root decryption key for the corresponding selected encrypted file data key (See E1).
4. The system decrypts data key corresponding to the respectively named file and uses plaintext data key for AES-256 decryption.
5. The system gets the initialization vector and decrypts the file.
6. The encrypted file is deleted
7. The system has finished decryption and displays the file to the user.

Exceptional flow

E1 : The system gets into error state during decryption.

1. The system encounters an unexpected error during the decryption process.
2. The system displays an error message and instructions to the user with OK button.
3. The user clicks OK button.
4. The system returns user to the personal folder view.
5. The user tries decryption again by clicking the Decrypt button.
6. If the decryption is still unsuccessful, the system will instruct the user to contact the application creator for support.

Termination

The system presents the files of a user in their personal folder view, "user space".

Post condition

The system goes into a wait state.

2.1.1.14. Requirement 5: Back up files

2.1.1.15. Description & Priority

Personal files of a user should be backed up to the Amazon S3 server to reduce the risk of possible device malfunction, malware or possibly stolen device. Personal files of a user must be encrypted when sent to the Amazon S3 backup server using secure HTTPS connection utilizing TLS protocol.

Priority: High. Without backup risks related to the loss of access to the data may become realized.

2.1.1.16. Use Case

UC-5

Scope

The scope of this use case is to ensure the file availability in case of unexpected events.

Description

This use case describes the process of user backing up their data to Amazon S3 using back up button from the application toolbar.

Flow Description

Precondition

The user has created an account in the system, opened the application and logged in using Supabase OTP. The user has active Amazon S3 account configured properly with relevant permissions through groups. The user has an active session.

Activation

This use case starts when the user has selected the desired file and clicks the back up button in the graphical user interface of the application.

Main flow

1. The system will send a request to connect to S3 bucket to put a new object there.
2. The AWS S3 accepts the connection and starts transferring the file there.

Exceptional flow

E1: Connection issue during backup

1. The system displays an alert indicating the connection cannot be formed or is interrupted and asks user to try again.

Termination

The system confirms to the user through display alert that the files have been backed up and synced to AWS S3 successfully.

Post condition

The system goes into a wait state.

[2.1.1.17. Requirement 6: Restore files from backup](#)

[2.1.1.18. Description & Priority](#)

The user should be able to retrieve backup files from the Amazon S3 server if there is a device malfunction, malware infection or the device gets stolen. The personal files of a user are encrypted using server-side encryption at S3.

Priority: High. Without the ability to restore files from Amazon S3 backups, risks related to the availability of files become realized.

[2.1.1.19. Use Case](#)

UC-6

Scope

The scope of this use case is to ensure the availability of user files and integrity through the option to retrieve backups.

Description

This use case describes the process of user retrieving their files by downloading them from Amazon S3 with a new device if the previous one was stolen, broken, had a general malfunction, or was infected with malware.

Flow Description

Precondition

The user has downloaded the application to the new computer and has performed the initial installation. The user has logged in to the system and is now in the user space view.

Activation

The user has clicked recover files from cloud button in the application toolbar.

Main flow

1. The system will request to make a connection to the AWS S3 bucket.
2. The system will download the files in the S3 bucket to the local storage of the user.

Exceptional flow

E1: Error 1 to be defined

1. The process will become more precise during implementation.

Termination

The system confirms to the user with a new window that the files have been synced from Amazon S3 with a certain date and time.

Post condition

The system goes into a wait state.

[2.1.1.20. Requirement 7: Manage user data](#)

[2.1.1.21. Description & Priority](#)

The user needs to be able to create files within the secure application environment as easily as in a basic Windows environment.

Priority: Medium. The user needs to be able to update their email address.

[2.1.1.22. Use Case](#)

UC-7

Scope

The scope of this use case is to ensure a good user experience and provide control for the user to manage their data.

Description

This use case describes the process of user changing their data saved into the Supabase PostgreSQL BaaS database.

Flow Description

Precondition

The user has logged in to the system and the system is not performing other tasks. The system is in a wait state.

Activation

This use case starts when the user in user space view and selects Change email address from the File dropdown in the upper left-hand corner of the application window.

Main flow

1. The system identifies the option "Change email address" was clicked.
2. The system displays prompt to enter the new email address.
3. The user enters their email address
4. The user clicks ok (See E1).
5. The system sends user the verification link to their current and new email address. The both must be clicked for the change to take place.
6. The user clicks the link to verify the change.

7. The system updates data in the Supabase database.
8. The system redirects user to the application to login.
9. The user can now log in with the new email address

Exceptional flow

E1: System gets into an error state.

1. Due to a connection error to Supabase, the system is unable to update and save user details.
2. The system catches the error and displays an error message to the user with OK button and prompts the user to try again.
3. The system goes into a wait state.

Termination

The system returns to the personal folder view of the user.

Post condition

The system goes into a wait state.

[2.1.1.23. Requirement 8: Recover user account](#)

[2.1.1.24. Description & Priority](#)

A user must be able to recover their account for Supabase, Amazon S3, and Amazon KMS.

Priority: High.

[2.1.1.25. Use Case](#)

UC-8

Scope

The scope of this use case is to ensure that the user still has access to their files through required accounts even in the case of unexpected events to mitigate the risks involved.

Description

This use case describes the process of user recovering access to their account for Supabase, Amazon S3, and Amazon KMS.

Flow Description

Precondition

The user does not have access to their Supabase, Amazon S3, or Amazon KMS account.

Activation

This use case starts when the user tries to recover their account in the required services.

Main flow

1. The user goes to the relevant system website to recover their account.
2. The relevant system provides user with the necessary steps to recover their account.

Termination

The user manages to recover their account to access their files.

Post condition

The system lets the user log in and view their files normally.

[2.1.1.26. Requirement 9: Logout](#)

[2.1.1.27. Description & Priority](#)

A user must be able to terminate their session and log out of the system.

Priority: High.

[2.1.1.28. Use Case](#)

UC-9

Scope

The scope of this use case is to ensure that the proper session handling is in place and the user can log out of the system successfully.

Description

This use case describes the process of user logging out of the system and terminating their session.

Flow Description

Precondition

The user is logged in to the system and is not performing any tasks. The system is in wait state.

Activation

This use case starts when the user clicks the Log out button from the graphical user interface of the application, selects “Log out” from file menu or clicks cross icon in the right corner of the application.

Main flow

1. The system identifies that the user has clicked the UI element.

2. The system confirms from the user whether they would like to log out of the system.
3. The user clicks OK (See E1).

Exceptional flow

E1: System gets into an error state during session termination.

1. The system encounters an unexpected error during the session termination process.
2. The system displays an error message and instructions to the user with OK button.
3. The user clicks the OK button.
4. The system returns to the login window.

Termination

The system presents the window informing user that the log out was successful.

Post condition

The application has been closed.

2.1.2. Data Requirements

This section outlines the data requirements for the application. It specifies the types of data the application will collect, store, process, and transmit, along with the associated security considerations.

File data that the user provides is stored both locally on the device in the application storage but also in the Amazon S3 cloud as a backup version of the data. File meta data about the files, such as name and type are saved into the application. The application supports the following file types: Microsoft Word files (*.doc, *.docx, *.rtf), Microsoft Excel files (*.xls, *.xlsx), Microsoft PowerPoint files (*.ppt, *.pptx), PDF files (*.pdf), and text files (*.txt). Files are encrypted using AES-256 GCM if the user performs the encryption function in user space.

Data related to user authentication and access control consist of Supabase user credentials (email and initial password) which are stored at their secure server in PostgreSQL database. Additionally, the user data (encryption root key) will be saved in AWS KMS and files in the backup to AWS S3.

For data at rest, the unique data key will be generated locally for the logged in user and the root key used for data key decryption is saved in AWS KMS. Encrypted file specific data key is saved locally in the user space together with initialization vector used with AES-256 encryption.

For data in transit for the backups, TLS protocol will be used to prevent traditional man in the middle computational threats and the data should be encrypted before that by the user to protect against “harvest now, decrypt later” attacks.

Amazon S3 provides relevant IAM user access and other logs because of the robust security features of the cloud environment. The logs are immutable and cannot be tampered with. The logging elements include date and time of actions such as file uploads and downloads. Additionally, user actions are logged as well.

2.1.3. User Requirements

Users should be able to securely authenticate themselves using passwordless login. Authentication should be intuitive and easy to the user when giving access to the personal application views and user data securely. The acceptance criterion is that users should be able to log in using their email through OTP and relevant OTP view securely through Supabase Auth.

Furthermore, users should be able to encrypt, decrypt, backup and manage files securely. The application must allow users to move files to a secure folder, manipulate them (edit, rename and delete), and backup them encrypted to Amazon S3 cloud when necessary. The acceptance criterion is that users should be able to perform file operations efficiently within the local environment of the application. Additionally, the application could support file preview and versioning features, for example in the next version.

Files stored within the application should be encrypted by the user to achieve confidentiality. The application should employ strong encryption algorithm to protect the content of files stored locally. The acceptance criterion is that the user should be able to successfully use envelope encryption through AWS KMS and OWASP recommended AES-256 GCM encryption. Decryption should only be possible for authorized users with the appropriate AWS KMS credentials.

Availability of the user files should be always guaranteed. The acceptance criterion is that the users should be able to initiate backup and recovery operations for their data easily from within the application using AWS S3. The connections should use secure HTTPS and TLS protocols.

The confidentiality of the information provided by the users and their files should be a priority. Cyber security incidents such as information leaks, unauthorized access and other issues should be prevented using effective protective actions. The acceptance criterion is that the application should feature passwordless authentication policy and ensure data privacy through cryptographic measures such as strong encryption at rest and using computationally traditional strong HTTPS and TLS protocols. Maintenance through regular updates and security audits should be carried out when the application is in production.

Graphical user interface should be kept simple and easy to use. Navigation should be clear and intuitive. The acceptance criterion is that general tasks should be easily understood even by the users without technical background. This can be seen demonstrated through completed tasks without help or confusion. The application should look modern and pleasant from the visual perspective.

Laws and regulations should be considered in the functionalities of the application. Adhering to the legal requirements generates trust and transparency. The acceptance criterion is that the application design and functionalities should consider the privacy laws such as (General Data Protection Regulation) GDPR and other similar legal frameworks and rules.

2.1.4. Environmental Requirements

For client devices, the user must have access to device such as a computer or laptop with internet connectivity to access the features and functionalities of the application. From the infrastructure perspective, the application requires AWS S3 bucket and AWS KMS services. Minimum hardware specifications include CPU 1 gigahertz (GHz) or faster with two or more cores on a compatible 64-bit processor or system on a chip (SoC) or comparable, 4 GB of RAM, and enough SSD storage capacity for the files but at least 64 GB which is the minimum requirement for Windows 11.

The operating system must be Microsoft Windows 11 to be compatible with the application architecture at this phase, but .NET MAUI framework allows cross-platform development for android and iOS as well. A web server software, Apache but preferably NginX with Let's encrypt certificate, must be installed and configured to act as a redirection point when the user has completed the registration process. Amazon S3 (Simple Storage Service) is a web service that stores backup files from the local client application and the backups can be retrieved using AWS API. The application is developed using specific programming languages (C# and .NET) and framework (.NET MAUI). Therefore, it requires corresponding runtime environments.

The application requires stable internet connectivity to support backup functionality while ensuring uninterrupted access for users as well. Network security measures consist of firewall and encryption protocols which must be in place to protect against cyber threats.

TLS certificates are necessary to secure data in transit between the client device and the Amazon cloud server and Supabase database. Data encryption provides data confidentiality and integrity. Encryption algorithm AES-256 must be used to encrypt sensitive data at rest at first before transferring data. Respectively, data at rest must be secured using at least AES-128 which according to NIST (2024) will be post-quantum secure for tens of years based on present knowledge.

Regular backup procedures should be established to create copies of application data and configurations, ensuring data integrity and availability in the event of hardware failure or data loss. A complete disaster recovery plan should be written to prevent from unlikely events such AWS server outages, broken devices, criminal activity such as theft or cyber attacks, and natural disasters. The plan should be considered when designing and implementing the application functionality. Best cyber security practices, standards and recommendations should be followed.

2.1.5. Usability Requirements

The application interface must be easy to navigate and logical. Finding features and functionalities within the application should be straightforward. The acceptance criterion is that the menu items within the personal folder view and the action buttons should be clearly available and easy to notice. Secondly, menu items and navigation links should be logically grouped and labelled clearly to reflect the structure and functionality of the application.

The application interface must adhere to consistent design patterns and UI conventions. Users should experience a coherent look and feel across different sections and screens of the application. The acceptance criterion is that the application should feature consistent visual resources such as colors, typography, and buttons. Adherence to platform-specific design guidelines for Windows 11 by Microsoft (2021) native applications is also required.

The application could be extended to be used responsively with different screens in the further development phase in the future, however it is mainly targeted to Windows 11 users on desktops or laptops. Thus, the users should be able to use the application on the most generally available desktop computers and laptops. Secondly, general accessibility guidelines should be considered to accommodate users with disabilities.

The application must provide informative error messages and feedback to users. Users should receive clear guidance and instructions when errors occur, or actions fail through the user interface. The acceptance criteria include descriptive error messages that clearly explain the problem and suggest possible solutions. Furthermore, the use of visual techniques such as color changes and icons are utilized to highlight errors and indicate successful actions.

The application should provide accessible help resources and documentation. Users should have access to comprehensive help content and instructions to assist them in using the application effectively. The acceptance criterion would be to provide the user with manuals as documentation for more detailed guidance.

The application must deliver optimal performance and minimal loading times. Users should experience smooth and responsive interactions with the application, even under heavy load conditions. The acceptance criterion is the optimization of code and resources to minimize load times and enhance overall responsiveness. Monitoring and optimization of server-side performance is required to ensure fast data retrieval and processing.

The application should have channels for user feedback and support iterative improvements. Users should have an opportunity to provide feedback on their experience with the application. The development team should utilize this feedback to enhance the application based on this user experience data. The acceptance criterion is the implementation of a feedback channel which could be as simple as

email and possibly additional feedback functionality within the application at the later phase.

2.2. Design & Architecture

The user interface is based on Windows 11 design principles of Microsoft (2021) providing an intuitive interface for users to interact with the application. The main interface components are feature based, AWS KMS utility and AES encryption utility for envelope encryption, Supabase service for user creation and authentication and AWS S3 utility for backup and recover functionalities. The client application backend handles user requests, file operations, encryption, and communication with Amazon and Supabase.

The application uses Supabase for user registration and authentication. For this Supabase offers scalable and secure PostgreSQL cloud database and robust bcrypt algorithm secures the data.

Amazon Web Services S3 and AWS KMS both use strong server side encryption to secure the data at rest.

The application design makes use of .NET MAUI framework which is implemented with Model-View-ViewModel (MVVM) design pattern. Moreover, MVVM pattern address separation of concerns by splitting application logic, presentation logic, and user interface. The idea is that model and Viewmodel should not know about each other. This pattern helps with teaming, for example designer can work with UI and software developer with the view background logic. Furthermore, the pattern decreases code redundancy and helps with more effective unit testing.

Amazon S3 is used as the backend storage solution for storing encrypted files. It consists of buckets which are containers for storing files and objects which are individual files stored within buckets.

AES-256 is used for encrypting and decrypting files locally before storing them on Amazon S3 for backup (if the user wants so). The mathematical notation of the algorithm can be written as follows:

encryption:

$$E_k(m) = \text{AES-256-Encrypt}(k,m)$$

decryption:

$$D_k(c) = \text{AES-256-Decrypt}(k,c)$$

Where k is the plaintext encryption key which is produced by first decrypting the unique data key and m is the plaintext message. C is the encrypted message.

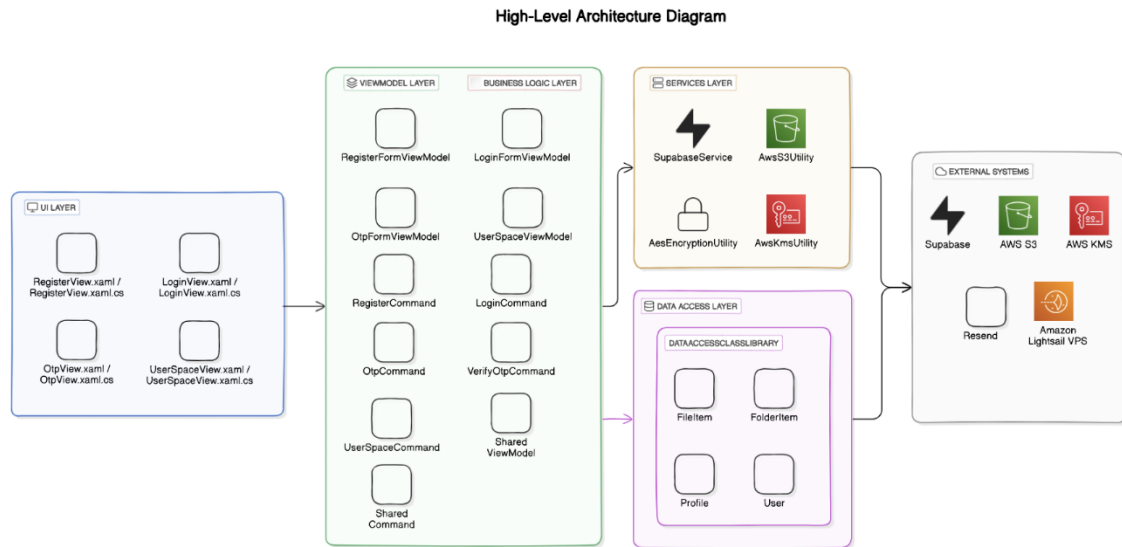


Figure 2. High-level architecture diagram.

The architecture of the application (Figure 2) consists of UI layer with XAML views, viewmodels and command classes control the business logic and data access library is separately keeping models for data interactions without knowing anything about view or UI layer. Services layer consists of the main services and utilities and the external cloud services are in the external systems section in the diagram.

A user interacts with the application through the client application user interface, which sends requests to third party services through the application backend utilities and functions. This architecture ensures a secure and efficient system for storing and managing files with encryption. It employs proven and recommended enterprise standard algorithms like AES-256 GCM for data security through envelope encryption and decryption.

2.3. Implementation

This section covers different code snippets corresponding to the use cases. The implementation relied heavily on NIST recommended algorithms for data at rest.

2.3.1. Main algorithms

AES-256 GCM and envelope encryption and decryption utilizing AWS Key Management Service and AesGcm Class were the main algorithms implemented in this project. Additionally, data key management was executed by managing the encryption and decryption of data keys through AWS API. Session persistence and retrieval through Supabase were necessary in terms of security and functionality as changing user email address in user space view would not have been possible without it. I also implemented validation rules for different forms.

From design patterns, SupabaseService uses singleton. Command pattern was used for example with LoginCommand or VerifyOtpCommand. Overall structure of the application follows MVVM (Model-View-Viewmodel) for program architecture and

this is demonstrated by implementing different features and commands for user actions.

2.3.2. Classes

Classes represent models, views and viewmodels as the implementation is based on MVVM pattern which is also one of the best ways to create a .NET MAUI application. This section contains class diagrams of the important classes.

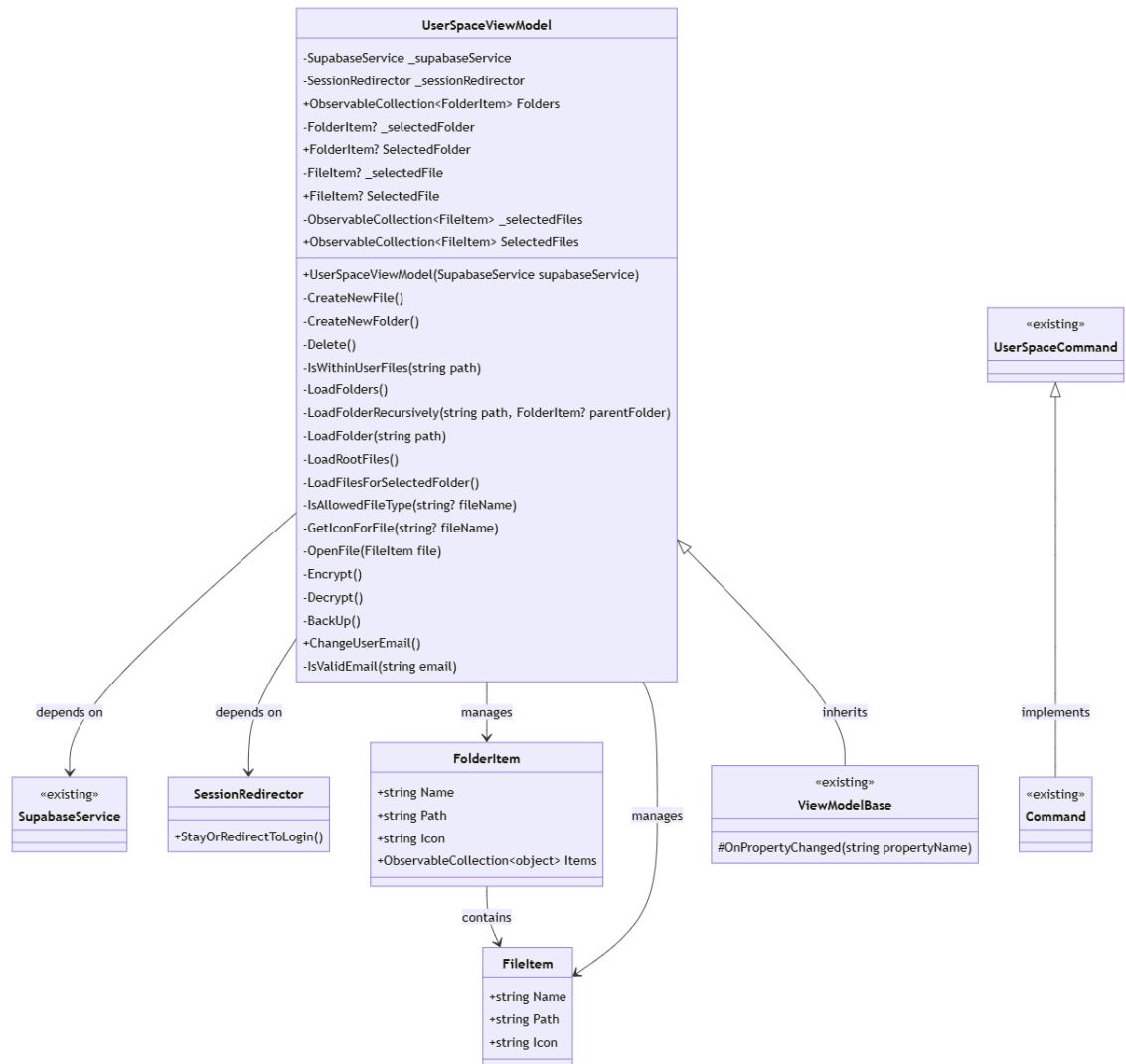


Figure 3. UserSpaceViewModel.cs class

`UserSpaceViewModel.cs` is responsible for main user actions within the `UserFiles` folder in the application, for example calling encrypt, decrypt and backup methods from the utility classes.

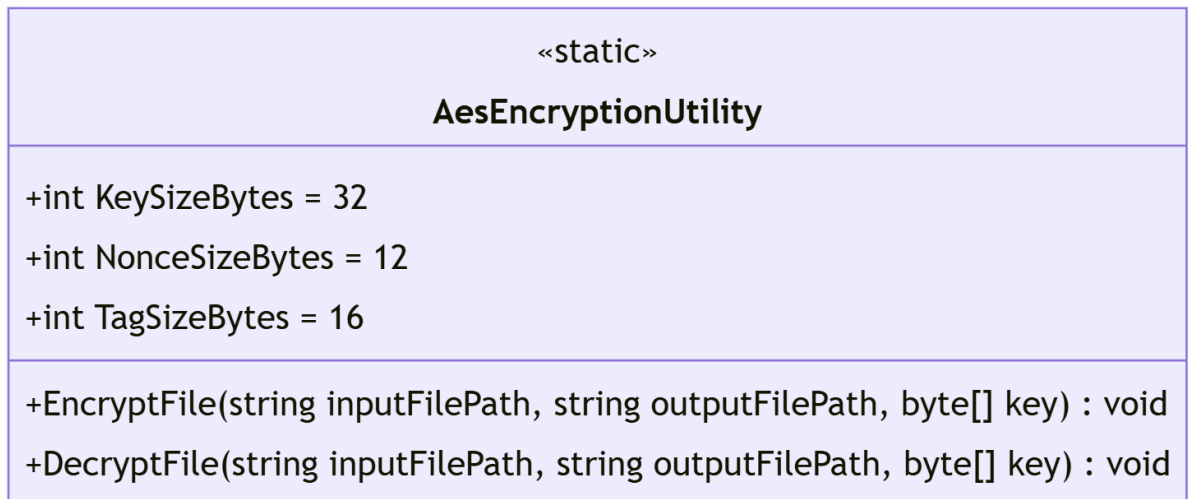


Figure 4. AesEncryptionUtility.cs class

AesEncryptionUtility class is responsible for encrypt file and decrypt file operations using AES-256-GCM with keysize of 32 bytes, nonce (or initialization vector) of 12 bytes and authentication tag of 16 bytes.

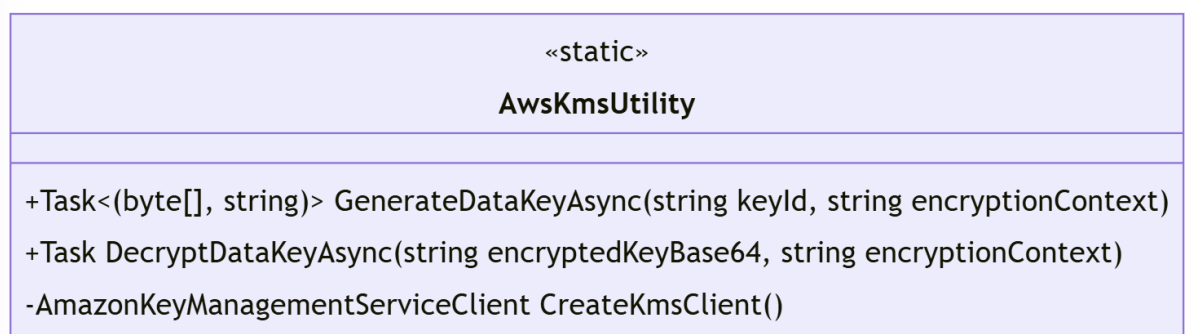


Figure 5. AwsKmsUtility.cs class

AwsKmsUtility class is handling data key generation and along with it its encryption and decryption using root key located in AWS Key Management Service.

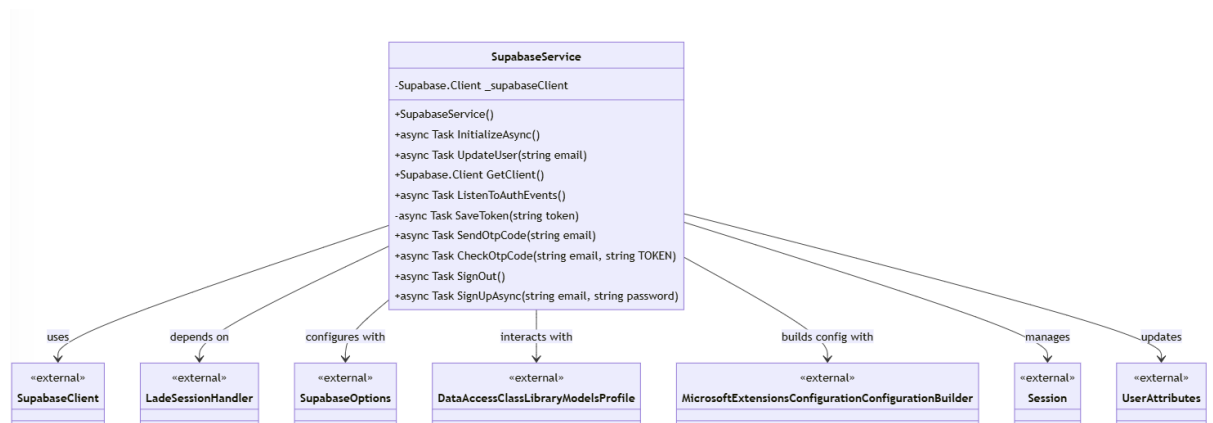


Figure 6. SupabaseService.cs class

SupabaseService.cs class takes care of interactions to Supabase Authentication and cloud database PostgreSQL operations.

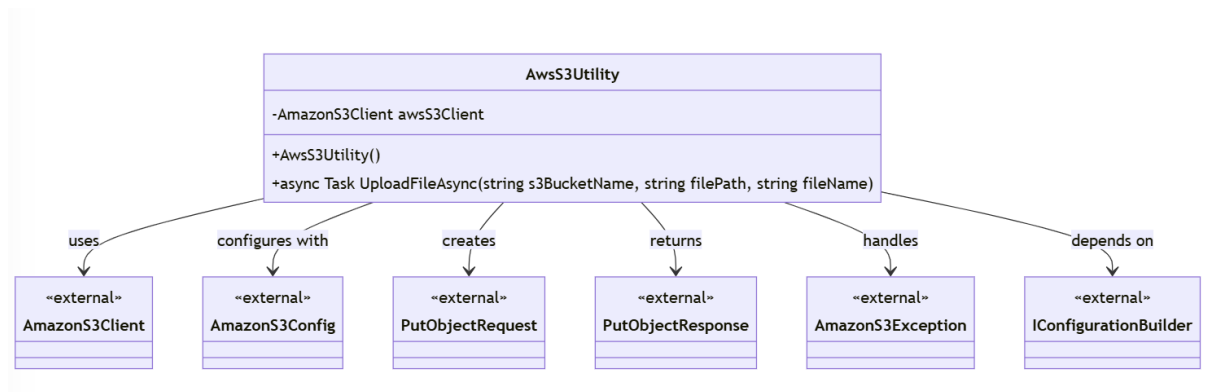


Figure 7. AwsS3Utility class

AwsS3Utility class connects to AWS S3 using set bucket policy and API keys in the client and manages file uploads in terms of backup operations.

2.3.3. Functions

This section contains graphical commented code snippets which are shortly described underneath each snippet. They represent the highest priority use cases.

```

147
148 // asynchronous public function to sign up a new user to Supabase PostgreSQL database
149 1 reference
150 public async Task SignUpAsync(string email, string password)
151 {
152     try
153     {
154         // attempt to call Supabase Auth.SignUp method to create a new user
155         await _supabaseClient.Auth.SignUp(email, password);
156     }
157     // catch exception if the user already exists in the system
158     catch (Exception ex)
159     {
160         // if the exception message contains the string "23505", the user already exists in the system
161         if(ex.Message.Contains("23505")){
162             throw new Exception($"The user exists in the system already - try again.");
163         }
164         else
165         {
166             // in any other error scenario, throw a generic exception to the end user
167             throw new Exception($"Failed to create a new user.");
168         }
169     }
170 }

```

Code snippet 1. SignUpAsync() function for user registration

The first code snippet (Code snippet 1) is from SupabaseService.cs class under Services folder and it describes how asynchronous SignUpAsync function with email and password strings (from RegisterFormViewModel.cs) as parameters using await keyword pauses execution without blocking the thread and calls to Supabase API to create a new user into the PostgreSQL database. The password is created only once and not needed later for the user activities. Supabase uses effective bcrypt to encrypt sent data securely.

```

29 // protected function to execute async task taking optional object parameter
30 // which can pass data from UI (view) to viewmodel, for example registration data
31 // 2 references
32 protected override async Task ExecuteAsync(object? parameter)
33 {
34     // if the main page exists, get it
35     var mainPage = Application.Current?.MainPage;
36     // Check if passwords match in the password and confirm password fields
37     if (viewModel.Password != viewModel.ConfirmPassword)
38     {
39         // if the main page exists, display an alert
40         if (mainPage != null)
41         {
42             // display an alert when the passwords do not match
43             await mainPage.DisplayAlert("Error", "Passwords do not match", "OK");
44         }
45         // if the passwords do not match, exit the method
46         return;
47     }
48
49     // validate email format of the user entered email address
50     var emailValidationResult = ValidateEmail(viewModel.Email);
51     // check if the email is not valid
52     if (!emailValidationResult.IsValid)
53     {
54         // if the main page exists, display an alert
55         if (mainPage != null)
56         {
57             // display error message
58             await mainPage.DisplayAlert("Error", emailValidationResult.errorMessage, "OK");
59         }
60         return;
61     }

```

Code snippet 2. First part of ExecuteAsync() function

ExecuteAsync function (Code snippet 2) is part of RegisterCommand.cs class which inherits from the shared base class AsyncCommandBase. It performs validation for matching user entered password and confirm password fields and then performs additional validation for the user entered email address during the registration process.

```

62 try
63 {
64     // attempt to initialize Supabase client
65     await _supabaseService.InitializeAsync();
66
67     // register user
68     await _supabaseService.SignUpAsync(viewModel.Email, viewModel.Password);
69
70     // Handle success
71     if (mainPage != null)
72     {
73         // display an alert when the registration is successful
74         await mainPage.DisplayAlert("Success", "Registered successfully! Check your email for confirmation link and confirm your user account.", "OK");
75     }
76
77     // Clear fields after successful registration
78     viewModel.Email = string.Empty;
79     viewModel.Password = string.Empty;
80     viewModel.ConfirmPassword = string.Empty;
81 }
82 catch (Exception ex)
83 {
84     // Handle exception
85     if (mainPage != null)
86     {
87         // display an alert when the registration fails
88         await mainPage.DisplayAlert("Error", $"Failed to sign up. Please try again. {ex.Message}", "OK");
89     }
90 }
91

```

Code snippet 3. The rest of ExecuteAsync() function.

After validating password and email, the function tries to initialize Supabase client (Code snippet 3) and to register the user by calling SignUpAsync() function. If the application main page exists, it displays registration success message and cleans the

user entered data from the fields. If there is an exception during the process, display alert is displayed to the user indicating that the registration failed and prompts to try again.

```

18 // LoginCommand class is responsible for handling the login logic
19 public class LoginCommand(LoginFormViewModel viewModel, SupabaseService supabaseService) : AsyncCommandBase
20 {
21     // Initialize private readonly view model and Supabase service
22     private readonly LoginFormViewModel viewModel = viewModel;
23     private readonly SupabaseService _supabaseService = supabaseService;
24
25     // overriding protected asynchronous ExecuteAsync method to execute LoginCommand inherited from AsyncCommandBase
26     protected override async Task ExecuteAsync(object? parameter)
27     {
28         // initialize Supabase client
29         await _supabaseService.InitializeAsync();
30
31         // checking if the email is empty or not valid or whether the email is not valid
32         if (string.IsNullOrEmpty(viewModel.Email) || !IsValidEmail(viewModel.Email))
33         {
34             // if the main page is not null, display an alert
35             var mainPage = Application.Current?.MainPage;
36             if (mainPage != null)
37             {
38                 // display an alert with the message "Email is required and must be valid"
39                 await mainPage.DisplayAlert("Error", "Email is required and must be valid", "OK");
40             }
41             return;
42         }
43         else
44         {
45             // if the email is valid, send OTP code through the Supabase service
46             bool OtpIsSent = await _supabaseService.SendOtpCode(viewModel.Email);
47             // if OTP is sent, create a new OtpFormViewModel with user email and call OnSentOtpSuccess method
48             if (OtpIsSent)
49             {
50                 _ = new OtpFormViewModel(_supabaseService)
51                 {
52                     Email = viewModel.Email
53                 };
54                 await viewModel.OnSentOtpSuccess();
55             }
56         }
57     }
58 }

```

Code snippet 4. SendOtpCode() method executed through ExecuteAsync() method in class LoginCommand.cs inherited from shared AsyncCommandBase base class.

The function ExecuteAsync() initializes Supabase client (Code snippet 4) and then validates email after which if the validation fails, display alert is displayed to the user. If the entered email is valid, OTP code is sent through the Subabase service to the user by calling _supabaseService.SendOtpCode() method if they have registered themselves to the system. If the OTP is sent to the user they will be redirected to the OTP view to enter the received code.

```

    }
    else
    {
        // if OTP is not sent, display an alert with the message "Failed to send OTP code. Please try again."
        var mainPage = Application.Current?.MainPage;
        if (mainPage != null)
        {
            await mainPage.DisplayAlert("Error", "Failed to send OTP code. Please try again.", "OK");
        }
    }
}

// IsValidEmail static helper method is responsible for validating the email
private static bool IsValidEmail(string? email)
{
    // return true if the email is not empty and the email address is in valid format
    // System.Net.Mail is a namespace that contains classes that enable constructing and sending email messages,
    // for example, checks presence of @ symbol in the email address and not empty part before and after the @ symbol
    // and allowed symbols in the email address and the length of the email address
    return !string.IsNullOrEmpty(email) && new System.Net.Mail.MailAddress(email).Address == email;
}

```

Code snippet 5. The rest of the ExecuteAsync() method of LoginCommand.cs including helper method to validate the user entered email

Continuing from the Code snippet 4, if OTP code is not sent to the user, the application will display an error message to the user. The latter helper method isValidEmail() checks if the entered email is null or empty and that it fulfils several criteria of a valid email address (Code snippet 5). If the method returns false, display alert indicating of error is displayed to the user.

```
41 // public asynchronous method to navigate to OTP view
42 1 reference
43 public async Task OnSentOtpSuccess()
44 {
45     // Navigate to OTP view
46     await Shell.Current.GoToAsync(nameof(OtpView));
}
```

Code snippet 6. Simple OnSentOtpSuccess() located in LoginFormViewModel.cs class under “Features” and “Login” folders to demonstrate clear project structure and navigation logic.

OnSentOtpSuccess() (Code snippet 6) performs the navigation logic in a case that Otp was sent successfully through SendOtpCode() function (Code snippet 4).

```
// private asynchronous Encrypt method is responsible for encrypting the selected file
1 reference
private async Task Encrypt()
{
    try
    {
        // Check if the selected file is null
        if (SelectedFile == null)
        {
            // if mainPage is not null, display an alert with the error message
            var mainPage = Application.Current?.MainPage;
            if (mainPage != null)
            {
                // Display an alert with the error message
                await mainPage.DisplayAlert("Error", "No file selected.", "OK");
            }
            return;
        }

        // Create a new instance of Aes to be used for encryption utilizing AesEncryptionUtility
        using (Aes aes = Aes.Create())
        {
            // Ensure AES-256 key size
            aes.KeySize = 256;
            // Generate a new IV for the AES encryption
            aes.GenerateIV();
            // save the IV to a byte array
            byte[] iv = aes.IV;

            // Load the configuration settings from the appsettings.json file
            var configuration = new ConfigurationBuilder()
                .SetBasePath(AppContext.BaseDirectory)
                .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)
                .Build();

            // Get the KMS key ID from the configuration settings
            var kmsKeyId = configuration["AWS:Kms-key-id"];

            // Get the unencrypted file path
            string unencryptedFilePath = SelectedFile.Path;
            // Get the selected file name without the extension
            string fileNameWithoutExtension = Path.GetFileNameWithoutExtension(unencryptedFilePath);

            // Generate the data key corresponding to the file using AWS KMS (filename as encryption context)
            var (plaintextKey, encryptedKeyBase64) = await AwsKmsUtility.GenerateDataKeyAsync(kmsKeyId, fileNameWithoutExtension);
        }
    }
}
```

Code snippet 7. Encrypt() method from UserSpaceViewModel.cs viewmodel, the first part

For file encryption, a file must be selected first or the error will be displayed through display alert to the user if the main page exists (Code snippet 7). Next, new instance of AES-256 is created and initialization vector which is saved to a byte array. Next AWS KMS key is got from from the configuration file for data key generation which is performed by AwsKmsUtility class and GenerateDataKeyAsync() method with kmsKeyId and filenameWithoutExtension (derived from the selected file path) parameters.

```
// store AWS KMS generated plaintext data key in byte array
byte[] plaintextKeyBytes = plaintextKeyBase64;
// store AWS KMS generated encrypted data key in encryptedDataKey variable after converting it from base64 to byte array
var encryptedDataKey = Convert.FromBase64String(encryptedKeyBase64);

// Check if the plaintext key is null or the encrypted key is null or empty
if (plaintextKeyBase64 == null || string.IsNullOrEmpty(encryptedKeyBase64))
{
    // if mainPage is not null, display an alert with the error message
    var mainPage = Application.Current?.MainPage;
    if (mainPage != null)
    {
        // Display an alert with the error message indicating that the data key generation failed
        await mainPage.DisplayAlert("Error", "Failed to generate or store data key.", "OK");
    }
    // exit the method execution
    return;
}

// save selected file path to a string variable inputFilePath
string inputFilePath = SelectedFile.Path;
// check if the inputFilePath is null or empty
if (string.IsNullOrEmpty(inputFilePath))
{
    // if mainPage is not null, display an alert with the error message
    var mainPage = Application.Current?.MainPage;
    if (mainPage != null)
    {
        // Display an alert with the error message indicating that the file path is null or empty to the user
        await mainPage.DisplayAlert("Error", $"File path is null or empty for the file {SelectedFile.Name}.", "OK");
    }
    // return, exit the method execution
    return;
}

// save the encrypted file combining the input file path and .encrypted extension to a string variable outputFilePath
string outputFilePath = inputFilePath + ".encrypted";

try
{
    // attempt to encrypt the file using AES-GCM with the plaintext data key using AesEncryptionUtility class
    AesEncryptionUtility.EncryptFile(inputFilePath, outputFilePath, plaintextKeyBase64);

    // check if the encrypted file does not exist at the outputFilePath
    if (!File.Exists(outputFilePath))
    {
        // if mainPage is not null, display an alert with the error message
        var mainPageInner = Application.Current?.MainPage;
        if (mainPageInner != null)
        {
            // Display an alert with the error message indicating that the encrypted file was not created
            await mainPageInner.DisplayAlert("Error", $"Encrypted file not created: {outputFilePath}", "OK");
        }
    }
}
```

Code snippet 8. Encrypt() method from UserSpaceViewModel.cs viewmodel, the second part

Plaintext key which was generated using AWS KMS (Code snippet 7) is saved in byte array (to be used later with EncryptFile() and DecryptFile() methods of AesEncryptionUtility.cs class) and validated so that it is not null and that encrypted is not null or empty. If either of those conditions is true, error during data key generation will be displayed to the user. Furthermore, encrypted data key is saved into variable encryptedDataKey and converted from base64. Next selected file path will be saved in the inputFilePath variable and checked for null or empty condition. If the path is

null or empty, error will be displayed to the user. Next, encrypted file outputFilePath and its extension is combined using inputFilePath variable and .encrypted extension. After this, AesEncryptionUtility class with EncryptFile() function is called with relevant parameters and executed. If encrypted file does not exist in the output file path after writing the encrypted file, and the main page exists, error is displayed to the user.

```

    }
    // return, exit the method execution
    return;
}

// save the encrypted data key reasonably
string encryptedDataKeyFilePath = unencryptedFilePath + "-key.base64";
// save the encrypted data key to a string variable encryptedKeyBase64
await File.WriteAllTextAsync(encryptedDataKeyFilePath, encryptedKeyBase64);

// Delete the encrypted file
File.Delete(unencryptedFilePath);

// if mainPage is not null, display an alert with the success message
var mainPage = Application.Current?.MainPage;
if (mainPage != null)
{
    // Display an alert with the success message indicating that the file was encrypted successfully
    await mainPage.DisplayAlert("Success", "File encrypted successfully.", "OK");
}

// check if the selected folder is not null and the selected folder items are not null
if (SelectedFolder != null && SelectedFolder.Items != null)
{
    // cast the items to fileItem and find the matching file by path
    var fileItem = SelectedFolder.Items.OfType<FileItem>().FirstOrDefault(f => f.Path == unencryptedFilePath);
    // if (the selected) fileItem is not null
    if (fileItem != null)
    {
        // remove the unencrypted file from the selected folder items
        SelectedFolder.Items.Remove(fileItem);

        // refresh the items in the selected folder by setting the selected folder to null and then back to the selected folder
        var previousFolder = SelectedFolder;
        SelectedFolder = null;
        SelectedFolder = previousFolder;
    }
}

// catch possible exception during encryption process
catch (Exception ex)
{
    // if mainPage is not null, display an alert with the error message
    var mainPage = Application.Current?.MainPage;
    if (mainPage != null)
    {
        // Display an alert with the error message indicating that an error occurred during encryption
        await mainPage.DisplayAlert("Error", $"Error encrypting file {SelectedFile.Name}: {ex.Message}", "OK");
    }
}
}
}

```

Code snippet 9. Encrypt() method from UserSpaceViewModel.cs viewmodel, the third part

Encrypted key is saved in a single file with .base64 extension in the format file name and -key.base4 ending (Code snippet 9). If everything went well and there was no exception during execution, Encrypt() method continues to inform the user about successful file encryption or if something went wrong during the encryption process or the execution of Encrypt() (Code snippet 10) method (Code Snippet 7, Code snippet 8 and Code snippet 9), possible errors would be displayed to the user. Before catching the encryption process exception, if selected folder and folder items are present, the selected unencrypted file is removed and the view refreshed for user to see the changes.

```

    // catch possible exception during the Encrypt() method execution
    catch (Exception ex)
    {
        // if mainPage is not null, display an alert with the error message
        var mainPage = Application.Current?.MainPage;
        if (mainPage != null)
        {
            // Display an alert with the error message indicating that an error occurred during encryption
            await mainPage.DisplayAlert("Error", $"Error during encryption: {ex.Message}", "OK");
        }
    }
}

```

Code snippet 10. Encrypt() method from UserSpaceViewModel.cs viewmodel, the third part the fourth part, rest of the exception handling

```

// public static AesEncryptionUtility class is responsible for handling the AES-GCM encryption and decryption logic
// as part of envelope encryption
2 references
public static class AesEncryptionUtility
{
    // Initialize private static readonly KeySizeBytes, NonceSizeBytes, and TagSizeBytes
    // referring to: Brady, S. (2021). Securely encrypting and decrypting files in .NET Core with AES-GCM
    // from https://www.thomaslevesque.com/2021/02/21/securely-encrypting-and-decrypting-files-in-net-core-with-aes-gcm/
    // Accessed 1 August 2024
    // AES-256 key size is 32 bytes: 32 bytes * 8 bits/byte = 256 bits
    public static readonly int KeySizeBytes = 32;
    // nonce size 12 bytes * 8 bits/byte = 96 bits, recommended for AES-GCM
    public static readonly int NonceSizeBytes = 12;
    // tag size 16 bytes * 8 bits/byte = 128 bits, recommended for AES-GCM
    public static readonly int TagSizeBytes = 16;

    // EncryptFile method is responsible for encrypting the file
    1 reference
    public static void EncryptFile(string inputFilePath, string outputFilePath, byte[] key)
    {
        // if the inputFilePath is null or empty, throw an ArgumentNullException
        if (string.IsNullOrEmpty(inputFilePath))
            throw new ArgumentNullException(nameof(inputFilePath));
        // if the outputFilePath is null or empty, throw an ArgumentNullException
        if (string.IsNullOrEmpty(outputFilePath))
            throw new ArgumentNullException(nameof(outputFilePath));
        // if the key is null or the key length is not equal to KeySizeBytes, throw an ArgumentNullException
        if (key == null || key.Length != KeySizeBytes)
            throw new ArgumentNullException(nameof(key));

        // initialize nonce (or initialization vector) and tag arrays with NonceSizeBytes and TagSizeBytes
        byte[] nonce = new byte[NonceSizeBytes];
        byte[] tag = new byte[TagSizeBytes];

        // using RNGCryptoServiceProvider to generate a random nonce
        /*
        using (var rng = new RNGCryptoServiceProvider())
        {
            rng.GetBytes(nonce);
        }
        */
        // RNGCryptoServiceProvider() was apparently obsolete:
        // source Microsoft Docs: https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography.rngcryptoserviceprovider?view=net-8.0
        // Accessed 1 August 2024
        // so had to switch to RandomNumberGenerator
        RandomNumberGenerator.Fill(nonce);

        // read the plaintext file content into a byte array
        byte[] plaintext = File.ReadAllBytes(inputFilePath);
        // initialize ciphertext array with the same length as the plaintext
        byte[] ciphertext = new byte[plaintext.Length];
    }
}

```

Code snippet 11. EncryptFile() function from AesEncryptionUtility class under Features and its subfolder Encryption, part 1

EncryptFile() method (Code snippet 11) which was called in Encrypt() (Code snippet 8) checks for possible null or empty values and ArgumentNullExceptions in its parameters, input file path, output file path and (encryption) key byte array. After that nonce (or initialization vector) and authentication tag are initialized into byte arrays. After that RandomNumberGenerator() method is used to fill the nonce with random values and ciphertext array is initialized.


```

// using AesGcm to encrypt the plaintext
// source Microsoft docs: https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography.aesgcm.encrypt?view=net-8.0
// Accessed 1 August 2024
using (AesGcm aesGcm = new AesGcm(key))
{
    // encrypt the plaintext using the nonce, plaintext, ciphertext, and tag
    aesGcm.Encrypt(nonce, plaintext, ciphertext, tag);
}

// write the nonce, ciphertext, and tag to the output file
using (FileStream fsOutput = new FileStream(outputFilePath, FileMode.Create, FileAccess.Write))
{
    // writing nonce or initialization vector for uniqueness
    // same plaintext will be different for every encryption
    fsOutput.Write(nonce, 0, nonce.Length);
    // writing ciphertext to the file
    fsOutput.Write(ciphertext, 0, ciphertext.Length);
    // writing authentication tag for integrity check
    fsOutput.Write(tag, 0, tag.Length);
}
}

```

Code snippet 12. EncryptFile() function from AesEncryptionUtility class under Features and its subfolder Encryption, part 2

Continuing the EncryptFile() method (Code snippet 11), AesGcm class is used to encrypt the plaintext using the defined nonce, plaintext, ciphertext and tag as parameters. Finally, FileStream class instance is used to write the nonce, ciphertext and (authentication) tag to the storage (Code snippet 12).

```

// private asynchronous Decrypt method is responsible for decrypting the selected file
// reference
private async Task Decrypt()
{
    try
    {
        // Check if the selected file is null
        if (SelectedFile == null)
        {
            // if mainPage is not null, display an alert with the error message
            var mainPage = Application.Current?.MainPage;
            if (mainPage != null)
            {
                // Display an alert with the error message indicating that no file was selected
                await mainPage.DisplayAlert("Error", "No file selected.", "OK");
            }
            return;
        }

        // save the selected file path to a string variable encryptedFilePath
        string encryptedFilePath = SelectedFile.Path;
        // save the decrypted file path to a string variable decryptedFilePath
        string decryptedFilePath = encryptedFilePath.Replace(".encrypted", "");

        // check if the encryptedFilePath is null or empty
        if (string.IsNullOrEmpty(encryptedFilePath))
        {
            // if mainPage is not null, display an alert with the error message
            var mainPage = Application.Current?.MainPage;
            if (mainPage != null)
            {
                // Display an alert with the error message indicating that the file path is null or empty
                await mainPage.DisplayAlert("Error", $"File path is null or empty for the file {SelectedFile.Name}.", "OK");
            }
            return;
        }

        try
        {
            // use Path.GetDirectory() method to get the directory of the encrypted AWS generated data key file path
            string directory = Path.GetDirectory(encryptedFilePath);
            // save the encrypted key file path to a string variable fileNameWithoutExtension indicating that the .encrypted was trimmed from the file name
            string fileNameWithoutExtension = Path.GetFileNameWithoutExtension(encryptedFilePath);
            // save the original key file name without extension to a string variable originalFileNameWithoutExtension indicating that now we have just the file name
            // this will be used by AwsKmsUtility for encryption context
            string originalFileNameWithoutExtension = Path.GetFileNameWithoutExtension(fileNameWithoutExtension);
            // save the encrypted key file path to a string variable encryptedDataKeyFilePath
            string encryptedDataKeyFilePath = Path.Combine(directory, fileNameWithoutExtension + "-key.base64");

```

Code snippet 13. Decrypt() function of UserSpaceViewModel.cs which will call AesEncryptionUtility.cs for DecryptFile() method in the next Code Snippet 14, part 1

Decrypt() function (Code snippet 13) performs the reverse logic when compared to the Encrypt() function (Code snippet 7). At first the method checks if the user selected file is null and displays corresponding error through alert if the file is null. Assuming the user has selected encrypted file with .encrypted extension, the path of the file is

save into encryptedFilePath variable. Decrypted file path is set accordingly by replacing the .encrypted file extension with an empty string for further processing it later in the method. Encrypted file path is checked for possible null or empty values and the relevant display alert is shown if the condition applies. Next, the directory where the encrypted data key is saved with the file will be got and saved into string called directory after which encrypted file name is stored in fileNameWithoutExtension and similarly to get the original file name without extension for AWS KMS decryption parameter for encryption context, is processed using Path class and GetFileNameWithoutExtension() method. Then the paths are combined with the identification string part "-key.base64" to retrieve the key file as the first parameter in AWS KMS decryption process.

```
// check if the encryptedDataKeyFilePath does not exist
if (!File.Exists(encryptedDataKeyFilePath))
{
    // if mainPage is not null, display an alert with the error message
    var mainTestingPage = Application.Current?.MainPage;
    if (mainTestingPage != null)
    {
        // Display an alert with the error message indicating that the encrypted key file was not found
        await mainTestingPage.DisplayAlert("Error", $"Encrypted key file not found: {encryptedDataKeyFilePath}", "OK");
    }
    return;
}

// save the original file to encryptedDataKeyBase64 variable
string encryptedDataKeyBase64 = await File.ReadAllTextAsync(encryptedDataKeyFilePath);

// AwsKmsUtility is called to decrypt the data key using the encrypted key and the original file name without extension as encryption context
byte[] decryptedKey = await AwsKmsUtility.DecryptDataKeyAsync(encryptedDataKeyBase64, originalFileNameWithoutExtension);

// check if the decryptedKey is null or the length of the decryptedKey is not equal to 32, corresponding to the AES key size
if (decryptedKey == null || decryptedKey.Length != 32)
{
    // if mainPage is not null, display an alert with the error message
    var mainTestPage = Application.Current?.MainPage;
    if (mainTestPage != null)
    {
        // Display an alert with the error message indicating that the decrypted key is invalid
        await mainTestPage.DisplayAlert("Error", "Invalid decrypted key.", "OK");
    }
    return;
}

// Decrypt the file using the decrypted AWS generated data key and the DecryptFile() method of AesEncryptionUtility class
AesEncryptionUtility.DecryptFile(encryptedFilePath, decryptedFilePath, decryptedKey);

// Check if the decrypted file was not created
if (!File.Exists(decryptedFilePath))
{
    // if mainPage is not null, display an alert with the error message
    var mainTesterPage = Application.Current?.MainPage;
    if (mainTesterPage != null)
    {
        // Display an alert with the error message indicating that the decrypted file was not created
        await mainTesterPage.DisplayAlert("Error", $"Decrypted file not created: {decryptedFilePath}", "OK");
    }
    return;
}

// Delete the encrypted file
File.Delete(encryptedFilePath);

// Delete the encrypted data key file
File.Delete(encryptedDataKeyFilePath);
```

Code snippet 14. Decrypt() function of UserSpaceViewModel.cs which calls AesEncryptionUtility.cs for DecryptFile() method, part 2

Continuing the Decrypt() function (Code snippet 13) if the encrypted data key file is not found, relevant display alert is shown to the user. Next, encrypted key file is read using await keyword and used by AWS KMS to get the plaintext data key which is saved into byte array to be used with AesEncryptionUtility DecryptFile() method to decrypt the encrypted file. If the decrypted key length is not corresponding to AES-256 bits, invalid decrypted key error will be displayed to the user through display alert. Now,

DecryptFile() method is called utilizing encryptedFilePath, decryptedFilePath and decryptedKey parameters and if the decrypted file does not exist after assumedly successful decryption, relevant display alert indicating error during decryption will be shown to the user. Finally, the encrypted file and encrypted data key file are deleted as the user does not need them anymore.

```
// if mainPage is not null, display an alert with the success message
var successPage = Application.Current?.MainPage;
if (successPage != null)
{
    // Display an alert with the success message indicating that the file was decrypted successfully
    await successPage.DisplayAlert("Success", "File decrypted successfully.", "OK");
}

// check if the selected folder is not null and the selected folder items are not null
if (SelectedFolder != null && SelectedFolder.Items != null)
{
    // Cast the items to FileItem and find the matching file by path
    var fileItem = SelectedFolder.Items.OfType<FileItem>().FirstOrDefault(f => f.Path == encryptedFilePath);
    if (fileItem != null)
    {
        SelectedFolder.Items.Remove(fileItem);

        // refresh the items in the selected folder by setting the selected folder to null and then back to the selected folder
        var previousFolder = SelectedFolder;
        SelectedFolder = null;
        SelectedFolder = previousFolder;
    }
}

// catch possible exception during decryption process
catch (Exception ex)
{
    // if mainPage is not null, display an alert with the error message
    var mainPage = Application.Current?.MainPage;
    if (mainPage != null)
    {
        // Display an alert with the error message indicating that an error occurred during decryption
        await mainPage.DisplayAlert("Error", $"Error decrypting file {SelectedFile.Name}: {ex.Message}", "OK");
    }
}

// catch possible exception during the Decrypt() method execution
catch (Exception ex)
{
    // if mainPage is not null, display an alert with the error message
    var errorPage = Application.Current?.MainPage;
    if (errorPage != null)
    {
        // Display an alert with the error message indicating that an error occurred during decryption
        await errorPage.DisplayAlert("Error", $"Error during decryption: {ex.Message}", "OK");
    }
}
}
```

Code snippet 15. Decrypt() function of UserSpaceViewModel.cs, part 3

Continuing the Decrypt() method (Code Snippet 13), if the application main page is not empty, file decrypted successfully display alert will be displayed to the user (Code snippet 15). The view will be refreshed for the user and possible exceptions caught during decryption process or execution of decrypt() method will be displayed to the user through display alerts.


```

// public static void DecryptFile method is responsible for decrypting the file
1 reference
public static void DecryptFile(string inputFilePath, string outputFilePath, byte[] key)
{
    // if the inputFilePath is null or empty, throw an ArgumentNullException
    if (string.IsNullOrEmpty(inputFilePath))
        throw new ArgumentNullException(nameof(inputFilePath));
    // if the outputFilePath is null or empty, throw an ArgumentNullException
    if (string.IsNullOrEmpty(outputFilePath))
        throw new ArgumentNullException(nameof(outputFilePath));
    // if the key is null or the key length is not equal to KeySizeBytes, throw an ArgumentNullException
    if (key == null || key.Length != KeySizeBytes)
        throw new ArgumentNullException(nameof(key));

    // read the file content into a byte array
    byte[] fileContent = File.ReadAllBytes(inputFilePath);

    // initialize nonce, tag, and ciphertext arrays
    byte[] nonce = new byte[NonceSizeBytes];
    byte[] tag = new byte[TagSizeBytes];
    // ciphertext length is the file content length minus the nonce and tag size
    byte[] ciphertext = new byte[fileContent.Length - NonceSizeBytes - TagSizeBytes];

    // copy the nonce, tag, and ciphertext from the file content
    Buffer.BlockCopy(fileContent, 0, nonce, 0, NonceSizeBytes);
    Buffer.BlockCopy(fileContent, fileContent.Length - TagSizeBytes, tag, 0, TagSizeBytes);
    Buffer.BlockCopy(fileContent, NonceSizeBytes, ciphertext, 0, ciphertext.Length);

    // initialize plaintext array with the same length as the ciphertext
    byte[] plaintext = new byte[ciphertext.Length];

    // using AesGcm to decrypt the ciphertext
    using (AesGcm aesGcm = new AesGcm(key))
    {
        // decrypt the ciphertext using the nonce, ciphertext, tag, and plaintext
        // which had been written to the encrypted file
        aesGcm.Decrypt(nonce, ciphertext, tag, plaintext);
    }

    // write the plaintext to the output file
    File.WriteAllBytes(outputFilePath, plaintext);
}

```

Code snippet 16. DecryptFile() function of AesEncryptionUtility.cs.

DecryptFile() method (Code snippet 16) will get its parameters string inputFilePath, string, outputFilePath and byte array key from Decrypt() method of UserSpaceViewModel.cs as described in the previous relevant snippet (Code snippet 13). DecryptFile() method checks for null or empty and possible argument null exceptions to validate the parameters of the method. Then the encrypted file is read into byte array variable fileContent. Now, required variables for decryption (nonce, authentication tag and ciphertext) are initialized. Next, the relevant parameters are read by the BlockCopy() method of Buffer class. The plaintext byte array needed for the decrypted file is initialized next. Now, new AesGcm class object with the decryption key will be created to perform decryption using the Decrypt() method of the class which takes nonce, ciphertext, tag and plaintext as parameters. Next, the plaintext decrypted file is written to the output file path.

```

// public virtual async method UploadFileAsync is responsible for uploading the file to AWS S3 and called in UserSpaceViewModel
1 reference
public virtual async Task UploadFileAsync(string s3BucketName, string filePath, string fileName)
{
    // initialize configuration with appsettings.json
    var configuration = new ConfigurationBuilder()
        .SetBasePath(AppContext.BaseDirectory)
        .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)
        .Build();

    // Retrieve the file location environment variable for Key parameter to be used with AWS S3 PutObjectRequest from appsettings.json
    string fileLocationInS3 = configuration["AWS:Key"];
    // combine the file path with the file name to set the file location in S3 with file name
    string fileLocationInS3WithFileName = fileLocationInS3 + fileName;

    // attempt to upload the file to S3
    try
    {
        // initialize PutObjectRequest with the S3 bucket Name, File path, and key in S3
        PutObjectRequest putObjectRequest = new PutObjectRequest
        {
            BucketName = s3BucketName,
            FilePath = filePath,
            Key = fileLocationInS3WithFileName
        };

        // await the PutObjectAsync method to upload the file to S3
        PutObjectResponse response = await awsS3Client.PutObjectAsync(putObjectRequest);
    }
    catch (AmazonS3Exception e)
    {
        throw new Exception($"Error uploading file to S3: {e.Message}");
    }
    // catch any other exception and throw an exception with the unknown error message
    catch (Exception e)
    {
        throw new Exception($"Unknown error: {e.Message}");
    }
}

```

Code snippet 17. UploadFileAsync() function of AwsS3Utility.cs class for backing up user files under Features and its Backup subfolder.

UploadFileAsync() function takes the AWS S3 bucket name, filePath (to upload the selected file set in UserSpaceViewModel.cs) and filename to set the Key parameter for AWS S3 PutObjectRequest (Code snippet 17). The start of the file location is set as environment variable and that is why the config file appsettings.json is initialized at first to retrieve it. Then this path (referring to S3 bucket path) is combined with the file name to form the full object path for the Key parameter. If there are errors during the upload process, they will be thrown as exceptions with relevant message.


```

// private asynchronous BackUp method is responsible for backing up the selected file to the predefined S3 bucket
1 reference
private async Task BackUp()
{
    // save the mainPage to a variable mainPage
    var mainPage = Application.Current?.MainPage;

    try
    {
        // Check if the selected file is null and display error through display alert if so
        if (SelectedFile == null)
        {
            if (mainPage != null)
            {
                // Display an alert with the error message indicating that no file was selected
                await mainPage.DisplayAlert("Error", "No file selected.", "OK");
            }
            return;
        }

        // Load the configuration settings from the appsettings.json file
        var configuration = new ConfigurationBuilder()
            .SetBasePath(AppContext.BaseDirectory)
            .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)
            .Build();

        // Get the S3 bucket name from the configuration settings
        var S3bucketName = configuration["AWS:Bucket-name"];

        // Check if the S3 bucket name is null or empty
        if (string.IsNullOrEmpty(S3bucketName))
        {
            if (mainPage != null)
            {
                // Display an alert with the error message indicating that the S3 bucket name is not configured properly
                await mainPage.DisplayAlert("Error", "S3 bucket name is not configured properly.", "OK");
            }
            return;
        }

        // Create a new instance of AwsS3Utility
        var awsS3Utility = new AwsS3Utility();

        // initialize the variables filePath and fileName to empty strings
        string? filePath = string.Empty;
        string? fileName = string.Empty;

        try
        {
            // save the selected file path to the filePath variable
            filePath = SelectedFile.Path;
            // save the selected file name to the fileName variable
            fileName = SelectedFile.Name;

            // attempt to upload the file to the S3 bucket using the UploadFileAsync() method of AwsS3Utility class
            await awsS3Utility.UploadFileAsync(S3bucketName, filePath, fileName);
        }
    }
}

```

Code snippet 18. BackUp() function of UserSpaceViewmodel.cs class which is responsible for implementing the back up logic and calling AwsS3Utility.cs class when necessary, part 1

At first, there is check for possible null selection if no files have been selected for backup and the back up toolbar option is clicked in the user interface by the user. Next, configuration settings are loaded from the config file to get the set S3 bucket name needed for the UploadFileAsync() method. If the bucket name is null or empty, error will be displayed to the user. Next, new AwsS3Utility instance is created and needed variables file path (filePath) and (filename) are initialized to empty string and then based on the selected file by the user, the corresponding values are saved in them. Consequently, UploadFileAsync() method of AwsS3Utility is called to attempt to execute the file upload.

```

        // if mainPage is not null, display an alert with the success message
        if (mainPage != null)
        {
            // Display an alert with the success message indicating that the file was backed up successfully
            await mainPage.DisplayAlert("Success", "File backed up successfully.", "OK");
        }
    }

    // catch possible exception during the backup upload process
    catch (Exception ex)
    {
        if (mainPage != null)
        {
            // Display an alert with the error message indicating that an error occurred during the backup upload process
            await mainPage.DisplayAlert("Error", $"Error uploading file {fileName}: {ex.Message}", "OK");
        }
    }
}

// catch possible exception during the BackUp() method execution
catch (Exception ex)
{
    if (mainPage != null)
    {
        // Display an alert with the error message indicating that an error occurred during the backup method execution
        await mainPage.DisplayAlert("Error", $"Error during backup: {ex.Message}", "OK");
    }
}
}

```

Code snippet 19. BackUp() function of UserSpaceViewmodel.cs class, part 2

The possible exceptions are caught during the upload process and the backup method execution.

2.4. Graphical User Interface (GUI)

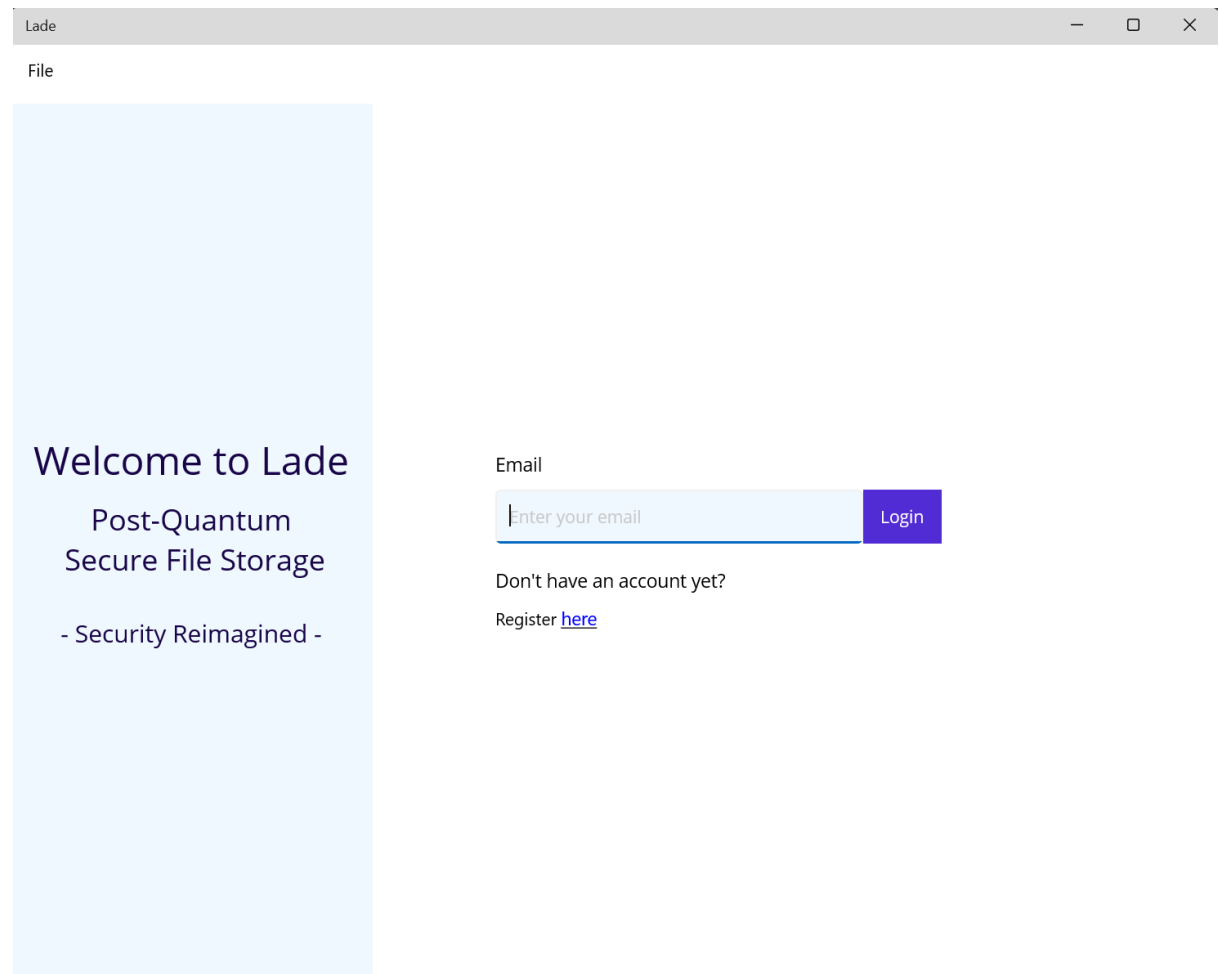


Figure 2. Login view of the Windows client application

The login view (Figure 2) is presented first to support frequent usage instead of always asking the user to register first. The view consists of the welcome splash area on the left and on the right there is a modern login entry to log in to the application with the relevant blueish button with "Login" text. If the user enters their email address and clicks login, they are being redirected to the new OTP view in which they can enter their OTP to authenticate themselves through Supabase Authentication service.

On the left-hand upper corner, there is a grouping File option like in many native Windows applications. It offers Exit functionality to exit from the application. The user is also able to quit the application clicking the upper right-hand corner x icon like in traditional Windows applications.

If the user has not registered for the application yet, they have an intuitive text label suggesting that and providing option to "Register here". By clicking on the underlined text, the user is redirected to register view.

Technically, the view consists of grids, flex layouts and stack layouts. The application has a minimum width but also supports full screen and expanding the application window.

← Lade

File

Welcome to Lade

Post-Quantum
Secure File Storage

- Security Reimagined -

Email

Enter your email

Password

Enter your password

Confirm Password

Confirm your password

Register

Already have an account?

Login [here](#)

Figure 3. Registration view

The welcome section on the left is still visible to the user after navigating to the registration view (Figure 3). The similar “brand” feeling is maintained and on the right, user is able to enter their email, password and password confirmation in the respective fields. When the user clicks the register button, the application posts the user data to Supabase API which will send a response to the user.

If the user accidentally clicked register in the previous screen, they could easily navigate back to the login page by clicking underlined here next to Login text and under “Already have an account?” label.

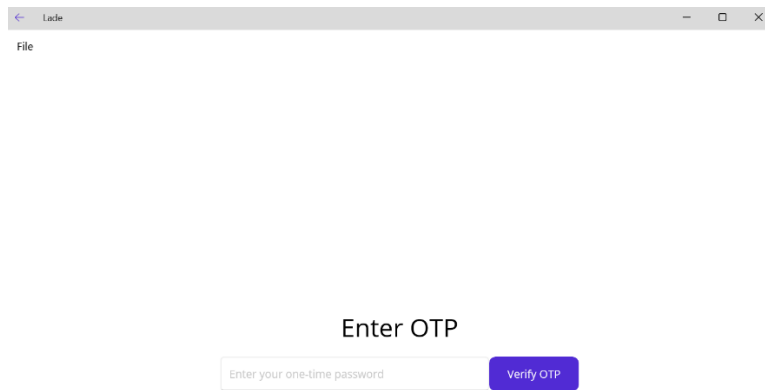


Figure 4. OTP view

The OTP view (Figure 4) is very intuitive. Heading text label together with entry placeholder tell user to enter their OTP and after entering the data, they should click the button on the right to verify the OTP they received after entering their email in the previous login view. After the successful authentication, session is established, and the user is redirected to user space view.

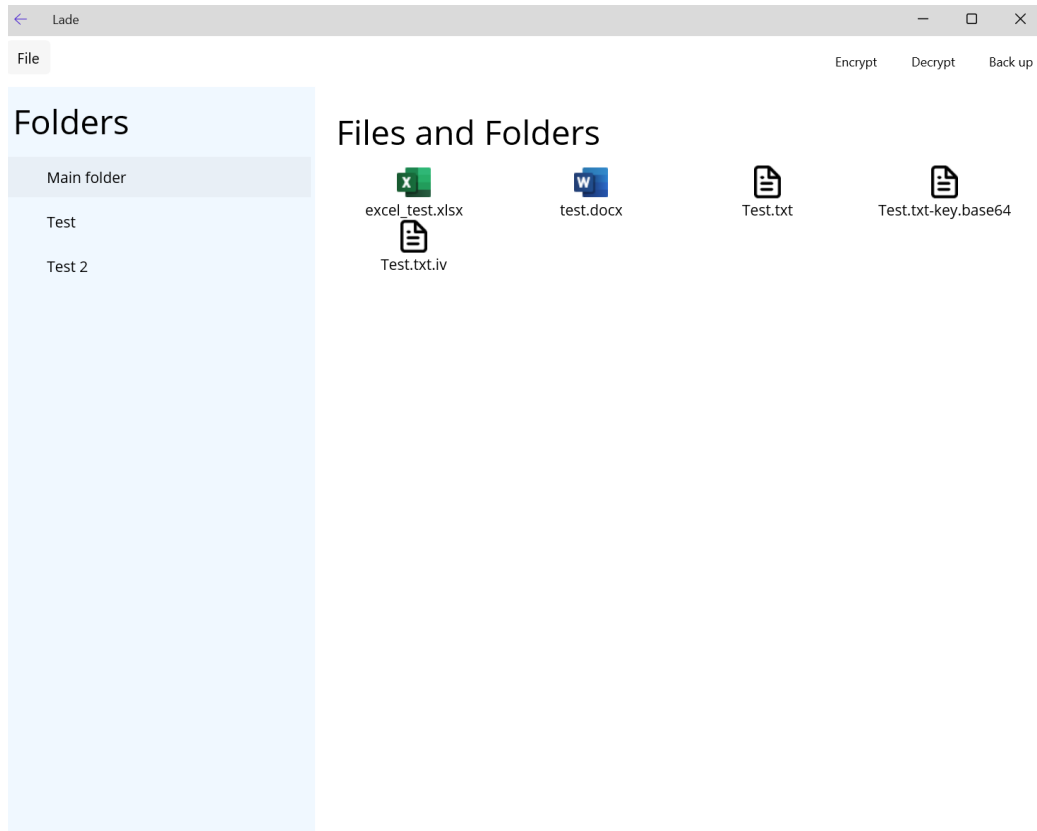


Figure 5. User space view.

The application has one main folder called Main folder but behind the scenes it is called UserFiles. On the left there are user folders and on the right files and folders indicated with relevant file icons. In the upper right-hand corner encrypt, decrypt and back up functionalities are represented clearly in the application toolbar.

2.5. Testing

Testing application functionality is essential to guarantee the usability of the application and to verify that the use cases and implemented logic works. The testing approach applied is traditional Test-Last or Test Later Development methodology because the fast prototyping is one of the goals of this project due to limited time resources (Functional Software Inc, 2022). The benefits of this approach also include writing less infeasible test cases when starting from the blank slate.

2.5.1. Objectives

This test plan achieves to test functionality to ensure that the application is production ready. The main objective is the verification of functional requirements so that business requirements and use cases expectations are met. Secondly, the goal is to check system performance through timed scenarios measuring application functionality response time based on use cases. Thirdly, the goal is to validate different implemented security features. Verifying system reliability and data integrity through the execution of functional testing plan is also essential.

2.5.2. Scope

There are 9 cases of which 8 can be tested programmatically. There are 2 test cases per use case to verify relevant functionalities. UC-8 is a use case based on third party systems because my application is based on distributed architecture, and that use case would require an instructions manual to be created for account recovery and general management. That is why it is out of scope in this testing plan

2.5.3. Test Strategies

Each use case was tested with at least two test cases. Unit testing for models was performed using popular open-source xUnit.net automated testing framework (.NET Foundation, 2024). Automated unit tests were completed using Visual Studio 2022 IDE. The rest of the testing is done mostly manually through functional and performance testing in the evaluation section of this report.

2.5.4. Test Environment

Testing operating system is Windows 11 Home version 23H2 OS Build 22631.3958. From hardware perspective, CPU is 13th Gen Intel(R) Core(TM) i9-13900H with the base clock speed of 2.60 GHz. There is 32GB of installed RAM and the system is 64-bit operating system with x64-based processor. There is 928 GB of SSD storage in the machine. The exact device model is ROG Zephyrus M16 GU604VI from ASUSTek COMPUTER INC.

Windows 11 has been updated to the latest version and there is Bitdefender free antivirus installed and actively monitoring the system.

2.5.5. Test Cases

Functional test cases are identified by abbreviation TC- and then followed by the number of the test case. There are two test cases corresponding to the associated use cases marked in the tables below.

TC-1: Successful registration	
Associated use case: Register account (UC-1)	
Objective	Expected Outcome
To verify that when valid data is provided during the registration process, the system successfully registers the user and displays a success message to the user.	This test case covers the registration functionality, ensuring that the system correctly handles user data and communicates with backend services to create a new user account. It includes the validation of data, interaction with external services, and user feedback.
Scope	
This test case covers the registration functionality, ensuring that the system correctly handles user data and communicates with backend services to create a new user account. It includes the validation of data, interaction with external services, and user feedback.	
Preconditions	
<ol style="list-style-type: none"> 1. The system is accessible and running. 2. The registration page or form is available and operational. 3. Supabase account has been created for the system with API credentials. 4. Necessary backend services (Supabase) is available and functional. 	
Test data	
Email: real email address, for example "x20142269@student.ncirl.ie" Password: "NCILsCool" Confirm Password: "NCILsCool "	
Test steps	
<ol style="list-style-type: none"> 1. Start the application. 2. Navigate to the registration page by clicking here link next to blue Register below login form. 3. Enter the credentials. 4. Submit the registration form. 5. Get feedback from the system through display alert 	
Pass Criteria	Fail Criteria
The test case passes if the success message is displayed to the user. Additionally, the data of the user must be correctly stored, and the user should be able to log in with the registered credentials.	The test case fails if an error message is displayed indicating unsuccessful user creation into the system.
Post conditions	
The user account is successfully registered in the system.	
Test results (Pass/Fail)	Pass

TC-2: Password Mismatch

Associated use case: Register account (UC-1)	
Objective	Expected Outcome
Verify that when the password and confirm password do not match, an error is shown.	An error display alert message is shown to the user indicating that passwords do not match.
Scope	
This test case focuses on validating the client-side logic that checks for matching passwords during the registration process. It ensures that the user receives clear feedback when the passwords do not match and the user cannot continue registration without entering matching passwords.	
Preconditions	
<ol style="list-style-type: none"> 1. The system is accessible and running. 2. The registration page or form is available and operational. 	
Test data	
Email: real not yet registered email address for testing, for example "x20142269@student.ncirl.ie" Password: "NCIIsCool" Confirm Password: "NCIIsNotCool"	
Test steps	
<ol style="list-style-type: none"> 1. Start the application. 2. Navigate to the registration page by clicking here link next to blue Register below login form. 3. Enter the credentials. 4. Submit the registration form. 5. Get feedback from the system through display alert 	
Pass criteria	Fail criteria
The test case passes if the error message is shown, clearly indicating that the passwords do not match, and the user is prevented from proceeding.	The test case fails if no error message is shown, if the error message is unclear or incorrect, or if the system allows the registration attempt to proceed.
Post conditions	
The tester clicks ok and returns to the registration page.	
Test results (Pass/Fail)	Pass

TC-3: Successful Passwordless Login to OTP Request view	
Associated use case: Login (UC-2)	
Objective	Expected Outcome
Verify that when a valid email is provided, the OTP request is sent successfully, and the user is navigated to the OTP view.	The SendOtpCode method of SupabaseService is called and executed successfully. The user is navigated to the OTP view.
Scope	
This test case covers the login functionality, specifically the transition from the login screen to the OTP request view upon providing a valid email. It involves verifying the correct invocation of the OTP sending mechanism and ensuring proper navigation within the application.	

Preconditions	
1. The system is accessible and running. 2. The registration page or form is available and operational. 3. The user has already registered to the system.	
Test data	
Valid user email: for example, x20142269@student.ncirl.ie	
Test steps	
1. Start the application. 2. Input valid email to the login form 3. Click login 4. Verify that you received OTP code in the specified email address 5. The application should redirect to the OTP view (in which the passwordless OTP code can be entered)	
Pass criteria	Fail criteria
The test case passes if the OTP code is successfully sent to the valid provided user email address and the user is navigated to the OTP view.	The test case fails if the OTP code is not sent, an error occurs, or the user is not navigated to the OTP view.
Post conditions	
OTP view is open for user to input their OTP code to authenticate them to the system.	
Test results (Pass/Fail)	Pass

TC-4: Invalid Email Format during login	
Associated use case: Login (UC-2)	
Objective	Expected Outcome
Verify that when an invalid email format is provided, an error message is displayed.	The IsValidEmail method returns false. The DisplayAlert method is called with a message indicating that the email is required and must be valid.
Scope	
This test case focuses on the validation of the email format during the login process. It ensures that the system can identify invalid email formats and provides clear feedback to the user.	
Preconditions	
1. The system is accessible and running. 2. The login page or form is available and operational. 3. The email validation logic at the client-side is implemented and functional.	
Test data	
Invalid email input: "x20142269@" without the end	
Test steps	
1. Start the application. 2. Input invalid email to the login form 3. Click login	
Pass criteria	Fail criteria

The test case passes if the system identifies the email as invalid an error message is shown to the user.	The test case fails if the system incorrectly identifies the email as valid or no error message is displayed.
Post conditions	
The user is not redirected to the OTP view but after clicking ok from the display alert must input the correct email address.	
Test results (Pass/Fail)	Pass

TC-5: Successful File Encryption	
Associated use case: Encrypt files at rest (UC-3)	
Objective	Expected Outcome
Verify that a file is successfully encrypted and saved when the "Encrypt" command is executed with valid file selection.	The file is encrypted and is named accordingly (filename.encrypted). The encrypted file is saved at the current folder together with the respective AWS KMS generated data key.
Scope	
This test case covers the functionality of file encryption within the system. It ensures that the selected file is properly encrypted, saved with an appropriate filename, and stored in the current location with the necessary encrypted data key.	
Preconditions	
<ol style="list-style-type: none"> 1. The system is accessible and running. 2. The user is logged in, authenticated with OTP, redirected to user space view and ready to perform file encryption. 3. The AWS Key Management Service (KMS) is accessible and correctly configured. 4. A valid file to be encrypted is available for encryption that the user has created in the system 	
Test data	
Original file: test.txt (user created) Expected encrypted filename: "test.txt.encrypted"	
Test steps	
<ol style="list-style-type: none"> 1. Select the file "test.txt" for encryption. 2. Click Encrypt button from the upper right-hand corner of the application to start the file encryption. 3. Get a success display alert "File encrypted successfully". 4. Check that encrypted file is visible in the current folder and named "test.txt.encrypted". 5. Check that the file is accompanied with "test.txt-key.base64 indicating that it is the AWS KMS generated data key for file decryption. 	
Pass criteria	Fail criteria
The test case passes if the file is correctly encrypted, named appropriately, and saved along with the AWS KMS-generated data key.	The test case fails if the file is not encrypted, incorrectly named, or if the encrypted file and data key are not saved as expected.
Post conditions	
The user stays in the user space view ready to perform another function such as back up, for example.	

Test results (Pass/Fail)	Pass
--------------------------	------

TC-6: Handle Encryption Failure Due to Missing Encryption Key	
Associated use case: Encrypt files at rest (UC-3)	
Objective	Expected Outcome
Verify that an appropriate error message is displayed when the encryption process fails due to a missing or invalid encryption key.	The application should display an error indicating that the data key needed for encryption is missing.
Scope	
This test case covers the error handling functionality during the file encryption process. It ensures that the system correctly identifies and handles situations where the encryption key is missing or invalid, providing clear feedback to the user.	
Preconditions	
<ol style="list-style-type: none"> 1. The system is accessible and running. 2. The user is logged in, authenticated and ready to perform file encryption in user space view. 3. The AWS Key Management Service (KMS) is configured, but the encryption key which should be generated by the service is missing. 	
Test data	
Original file: "test-encryption.doc" (user generated) From configuration file (appsettings.json) in the root folder of the program, leave empty the parameter Kms-key-id (which you normally get from AWS KMS)	
Test steps	
<ol style="list-style-type: none"> 1. Create the test file by right clicking on the right column where the files and folders are located. 2. Select New from the fly out menu and then File from its submenu. 3. Write the test file name and click ok. 4. Select the test file for encryption 5. Click Encrypt button in the toolbar from the upper right-hand corner 6. Get the error message indicating of missing encryption key: "Error during encryption: Error generating data key: 1 validation error detected: Value null at "KeyId" failed to satisfy constraint: Member must be not null". 	
Pass criteria	Fail criteria
The test case passes if the encryption process fails as expected and the user sees an appropriate display error message.	The test case fails if no error message is displayed.
Post conditions	
The user clicks ok from the error display alert and sees the user space view, ready to perform some other function.	
Test results (Pass/Fail)	Pass

TC-7: Successful File Decryption	
Associated use case: Decrypt files at rest (UC-4)	
Objective	Expected Outcome

Verify that an encrypted file is successfully decrypted and saved when the "Decrypt" command is executed with valid inputs.	The file is decrypted and saved at the output location (where the encrypted file together with its base64 AWS KMS data key was), restoring the original content. The application should inform the user that the file has been successfully decrypted.
Scope	
This test case covers the functionality of file decryption within the system. It ensures that the selected encrypted file is correctly decrypted using the associated AWS KMS data key, restoring the original content, and that the user is notified of the successful decryption.	
Preconditions	
<ol style="list-style-type: none"> 1. The system is accessible and running. 2. The user is logged in, authenticated and ready to perform file decryption in user space view. 3. The user has created the test file and encrypted it. 4. Generated encrypted data key and encrypted file are present in the current folder in user space view. 	
Test data	
Encrypted file "test.txt.encrypted" file and the associated data key "test.txt-key.base64" in the currently open folder	
Test steps	
<ol style="list-style-type: none"> 1. Select the encrypted test file (not the data key file) for decryption. 2. Click Decrypt button from the toolbar from the upper right-hand corner. 3. Success message "File decrypted successfully" should be displayed. 4. Click ok from the display alert. 5. Double click the decrypted file to verify it has the same contents as before and you can read its contents. 	
Pass criteria	Fail criteria
The test case passes if the file is correctly decrypted, restored to its original content, saved in the current folder, and the user receives a success notification.	The test case fails if the file is not decrypted correctly, the original content is not restored, the decrypted file is not saved, or the user does not receive a display alert.
Post conditions	
The user can see the user space and is ready to perform another function.	
Test results (Pass/Fail)	Pass

TC-8: Handle Decryption Failure Due to Incorrect Decryption Key	
Associated use case: Decrypt files at rest (UC-4)	
Objective	Expected Outcome
Verify that an appropriate error message is displayed when the decryption process fails due to an incorrect or corrupted decryption key.	An error message should be displayed to the user, indicating the failure to decrypt the file.
Scope	

This test case focuses on the error handling mechanisms during the file decryption process. It ensures that the system can correctly identify and respond to situations where the provided decryption key is incorrect, providing clear feedback to the user.	
Preconditions	
<ol style="list-style-type: none"> 1. The system is accessible and running. 2. The user is logged in, authenticated and ready to perform file decryption in user space view. 3. The user has created the test file and encrypted it. 4. Generated encrypted data key and encrypted file are present in the current folder in user space view. 	
Test data	
Previously encrypted file "test.txt.encrypted" file and the associated data key "test.txt-key.base64" which has been altered to "test2.txt-key.base64" in the currently open folder	
Test steps	
<ol style="list-style-type: none"> 1. Select the encrypted file (not modified data key) 2. Click Decrypt button from the toolbar from the upper right-hand corner. 3. Error message starting with "Encrypted key file not found" should be displayed. 4. Or if you edited the data key file content and removed a character at the end in Notepad, you get error message including "The input is not valid Base-64 string" text 	
Pass criteria	Fail criteria
The test case passes if the system correctly identifies the incorrect data key, the decryption process fails, and the user receives a clear and appropriate error message.	The test case fails if the system does not detect the incorrect key, the decryption process completes incorrectly, or no error message is shown.
Post conditions	
After clicking ok from the display alert, user can see the user space view and continue to perform other function.	
Test results (Pass/Fail)	Pass

TC-9: Successful File Backup	
Associated use case: Back up files (UC-5)	
Objective	Expected Outcome
Verify that a file is successfully backed up to the specified S3 bucket.	The file is successfully uploaded to the specified S3 bucket. The user is notified with a success message, such as "File backed up successfully."
Scope	
This test case covers the functionality of backing up files to an Amazon S3 bucket. It ensures that the file is correctly uploaded to the specified location in the cloud and that the user receives appropriate confirmation of the successful backup.	
Preconditions	
<ol style="list-style-type: none"> 1. The system is accessible and running. 2. The user is logged in, authenticated and ready to perform back up operation. 3. There is a test file created by the user in the current folder. 	

4. The AWS S3 bucket is accessible and configured with the appropriate permissions for file uploads.	
Test data	
User created "test.ppt" file	
Test steps	
<ol style="list-style-type: none"> 1. Select the test file 2. Click "Back up" button from the toolbar from the upper right-hand corner. 3. Success message "File backed up successfully" will be displayed. 4. (Optional) check AWS S3 and verify that the object can be found in the S3 bucket 	
Pass criteria	Fail criteria
The test case passes if the file is successfully uploaded to the specified S3 bucket, and the user is notified with a success message.	The test case fails if the file upload fails, the file is not found in the specified S3 bucket, or the user does not receive a success notification.
Post conditions	
After clicking ok from the display alert, user can see the user space view and continue to perform other function.	
Test results (Pass/Fail)	Pass

TC-10: Failure Due to Network Connectivity Issues	
Associated use case: Back up files (UC-5)	
Objective	Expected Outcome
Verify the behavior of the application when there is a network connectivity issue during the file backup process.	The application displays an error message indicating the failure, such as "Error uploading file". The file is not uploaded to the S3 bucket.
Scope	
This test case focuses on the ability of the application to handle upload error related network connectivity issues during the file backup process to an S3 bucket. It ensures that the application provides appropriate error feedback to the user and does not incorrectly indicate a successful upload.	
Preconditions	
<ol style="list-style-type: none"> 1. The system is accessible and running. 2. The user is logged in, authenticated and ready to perform back up operation. 3. There is a test file created by the user in the current folder. 4. The AWS S3 bucket is accessible and configured with the appropriate permissions for file uploads. 	
Test data	
User created large "failed-test.ppt" file	
Test steps	
<ol style="list-style-type: none"> 1. Select the test file 2. Click "Back up" button from the toolbar from the upper right-hand corner. 3. Disable network connection temporarily (usually Wi-Fi settings) when uploading the file. 4. Error message starting with "Error uploading file (file name here): Unknown error: No such host is known" should be displayed to the user. 	

5. (Optional) Check AWS S3 bucket and verify that the file is not there.	
Pass criteria	Fail criteria
The test case passes if the application correctly identifies the network issue, displays an appropriate error message, and the file is not uploaded to the S3 bucket.	The test case fails if the application does not display an error message, incorrectly indicates a successful upload, or the file is uploaded despite the network issue.
Post conditions	
After clicking ok from the display alert, user can see the user space view and continue to perform other function.	
Test results (Pass/Fail)	Pass

TC-11: Successful File Restore From The S3 Backup	
Associated use case: Restore files from backup (UC-6)	
Objective	Expected Outcome
To verify that a file can be successfully restored from an S3 backup to the local system.	The file "document.txt" is successfully restored from the S3 bucket to the local path. The user is notified of the successful restoration with a message.
Scope	
This test case covers the functionality of restoring files from an S3 bucket. It ensures that the file is correctly downloaded from the specified location in the cloud and saved to the local system.	
Preconditions	
<ol style="list-style-type: none"> 1. The app must be deleted and installed again to simulate broken device or lost access (for example theft). 2. The system is accessible and running. 3. The user is logged in, authenticated and ready to perform back up operation. 4. There is a test file created by the user in the current folder. 5. The AWS S3 bucket is accessible and configured with the appropriate permissions for file uploads. 	
Test data	
Previously backed up "test.ppt" file found in AWS S3 bucket	
Test steps	
<ol style="list-style-type: none"> 1. Select the test file 2. Click "Recover" button from the toolbar from the upper right-hand corner. 3. Prompt should be displayed to the user so that they are able to copy and paste the desired recovery path (if not the current folder). 4. Click ok after writing or pasting the file path. 5. The application starts the recovery process. 6. The application displays success message. 7. The recovered file can be found in the selected output location on the device. 	
Pass criteria	Fail criteria
The test case passes if the file is successfully restored and the user is notified with a success message.	The test case fails if the file is not restored, saved incorrectly, or the user does not receive a success notification.

Post conditions	
After clicking ok from the display alert, user can see the user space view, the recovered file and continue to perform other function.	
Test results (Pass/Fail)	Fail (not implemented due to time limitations)

TC-12: Handle Restoration Failure When Restoring File From S3	
Associated use case: Restore files from backup (UC-6)	
Objective	Expected Outcome
To verify the behavior of the application when attempting to restore a file but error occurs during the process.	An error message is displayed to the user indicating that there was an error during restoration process.
Scope	
This test case focuses on error handling when the specified file is not found in the S3 bucket. It ensures that the application provides appropriate feedback to the user.	
Preconditions	
<ol style="list-style-type: none"> 1. The system is accessible and running. 2. The user is logged in, authenticated and ready to perform back up operation. 3. There is a test file created by the user in the current folder. 4. The AWS S3 bucket is accessible and configured with the appropriate permissions for file uploads. 	
Test data	
Previously backed up "test.ppt" file found in AWS S3 bucket	
Test steps	
<ol style="list-style-type: none"> 1. Select the test file 2. Click "Recover" button from the toolbar from the upper right-hand corner. 3. Prompt should be displayed to the user so that they are able to copy and paste the desired recovery path (if not the current folder). 4. Click ok after writing or pasting the file path. 5. The application tries to start the recovery process but cannot for some reason (for example connection error due to network connectivity or S3 service down). 6. Error message is displayed to the user. 	
Pass criteria	Fail criteria
The test case passes if the application correctly displays an appropriate error message.	The test case fails if the application does not display any feedback to the user indicating of error during restoration process.
Post conditions	
After clicking ok from the display alert, user can see the user space view and continue to perform other function.	
Test results (Pass/Fail)	Fail (not implemented due to time limitations)

TC-13: Successful Email Address Change	
Associated use case: Manage user data (UC-7)	

Objective	Expected Outcome
To verify that the user can successfully change their email address and complete the verification process from the new email.	A verification email is sent to user defined new email address. The user clicks the verification link and the email address is successfully updated in the system.
Scope	
This test case covers the functionality of updating the email address of the user in the system (Supabase). It ensures that the user can request an email change, receive a verification link, and complete the email address update process.	
Preconditions	
<ol style="list-style-type: none"> 1. The system is accessible and running. 2. The user is logged in, authenticated and ready to perform email change operation in user space view. 3. The user has another email set up with which they can verify the new email address. 	
Test data	
New real email address, for example "x20142269@student.ncirl.ie"	
Test steps	
<ol style="list-style-type: none"> 1. Select File drop down menu from the upper left-hand corner of the application. 2. From the submenu select "Change email address" 3. Prompt opens to enter your email address. 4. Enter the test email address. 5. Click ok. 5. You get a success message from the system indicating of successful sending of the link. 6. Open the new email. 7. Click the registration link to verify your new email. 8. Log in to the system with a new email address. 	
Pass criteria	Fail criteria
The test case passes if the new email address is successfully updated and verified, and the user can log in with the new email address.	The test case fails if the verification email is not sent, the email address is not updated, or the user cannot verify the new email.
Post conditions	
The user should be sent OTP code to login with their new email address.	
Test results (Pass/Fail)	Pass

TC-14: Failure to Change Email Address Due to Invalid Email	
Associated use case: Manage user data (UC-7)	
Objective	Expected Outcome
To verify that the system correctly handles the input of an invalid email address format during the email change process.	An error message is displayed, indicating that the email address is invalid.
Scope	
This test case ensures that the system validates the format of the new email address and prevents the update if the email format is invalid, providing appropriate feedback to the user.	

Preconditions	
1. The system is accessible and running. 2. The user is logged in, authenticated and ready to perform email change operation in user space view.	
Test data	
Invalid email address: x20142269@	
Test steps	
1. Select File drop down menu from the upper left-hand corner of the application. 2. From the submenu select "Change email address" 3. Prompt opens to enter your email address. 4. Enter the test email address. 5. You get an error display alert indicating of invalid email address.	
Pass criteria	Fail criteria
The test case passes if the system correctly identifies the invalid email format, prevents the email change, and displays an appropriate error message.	The test case fails if the system allows the update with an invalid email or does not display an error message.
Post conditions	
After clicking ok from the display alert user can see the user space view and continue with other function.	
Test results (Pass/Fail)	Pass

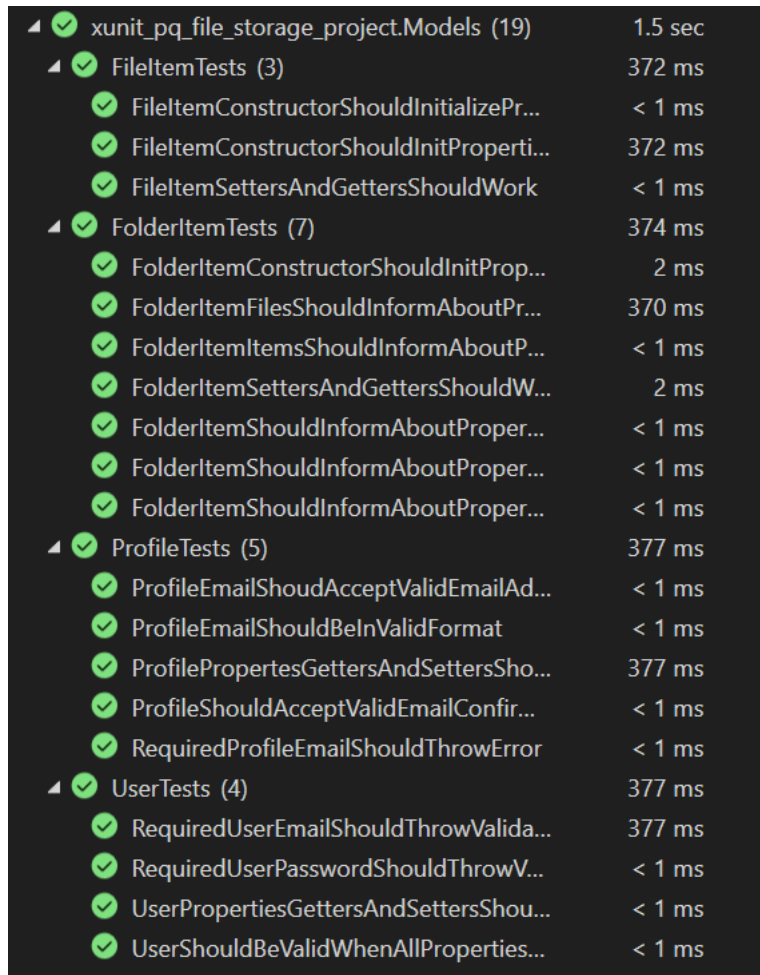
TC-15: Successful Logout	
Associated use case: Logout (UC-9)	
Objective	Expected Outcome
To verify that the user can successfully disconnect from the system through Supabase and close the application securely.	The user session is terminated, and the application has closed.
Scope	
This test case covers the functionality of logging out from the system. It ensures that the session of the user is properly terminated, and any sensitive data is securely handled during the logout process.	
Preconditions	
1. The system is accessible and running. 2. The user is logged in, authenticated and ready to perform operation in user space view.	
Test data	
Valid credentials of a registered user: already registered email address	
Test steps	
1. Select File drop down menu from the upper left-hand corner of the application. 2. From the submenu select "Log out and exit" 3. Application should display prompt asking if the user is sure and wants to exit. 4. User clicks ok. 5. User session is terminated, and the application closed.	

Pass criteria	Fail criteria
The test case passes if the user is successfully logged out, the session is securely terminated, and the user is either prompted to log out and close the application.	The test case fails if the user remains logged in, the session is not properly terminated, or the user is not notified of a successful logout.
Post conditions	
The application is closed and can be opened again if the user wants so.	
Test results (Pass/Fail)	Pass

TC-16: Logout Cancellation	
Associated use case: Logout (UC-9)	
Objective	Expected Outcome
To verify that the user can cancel the logout process, ensuring that the session remains active, and the application stays open.	The user remains logged in, and their session is not terminated. The application remains open and functional.
Scope	
This test case covers the functionality allowing users to cancel an initiated logout request. It ensures that the user can abort the logout process and continue their session without interruption.	
Preconditions	
<ol style="list-style-type: none"> 1. The system is accessible and running. 2. The user is logged in, authenticated and ready to perform operation in user space view. 	
Test data	
Valid credentials of a registered user: already registered email address	
Test steps	
<ol style="list-style-type: none"> 1. Select File drop down menu from the upper left-hand corner of the application. 2. From the submenu select "Log out and exit" 3. Application should display prompt asking if the user is sure and wants to exit. 4. User clicks cancel. 5. User returns to the current view as if nothing ever happened. 	
Pass criteria	Fail criteria
The test case passes if the user can cancel the logout process, remain logged in, and continue using the application without any interruption to their session.	The test case fails if the application logs the user out despite selecting the "Cancel" option, if session is interrupted, or if the application closes.
Post conditions	
The user can continue to perform other function in the current view.	
Test results (Pass/Fail)	Pass

2.5.6. Automated testing

Models from different DataAccessClassLibrary were tested with xUnit with the following passing results. The test mostly evaluated the setters and getters of the classes and InotifyPropertyChanged related to the property changes of the models.



▲ ✓ xunit_pq_file_storage_project.Models (19)	1.5 sec
▲ ✓ FileItemTests (3)	372 ms
✓ FileItemConstructorShouldInitializePr...	< 1 ms
✓ FileItemConstructorShouldInitProperti...	372 ms
✓ FileItemSettersAndGettersShouldWork	< 1 ms
▲ ✓ FolderItemTests (7)	374 ms
✓ FolderItemConstructorShouldInitProp...	2 ms
✓ FolderItemFilesShouldInformAboutPr...	370 ms
✓ FolderItemItemsShouldInformAboutP...	< 1 ms
✓ FolderItemSettersAndGettersShouldW...	2 ms
✓ FolderItemShouldInformAboutProper...	< 1 ms
✓ FolderItemShouldInformAboutProper...	< 1 ms
✓ FolderItemShouldInformAboutProper...	< 1 ms
▲ ✓ ProfileTests (5)	377 ms
✓ ProfileEmailShoudAcceptValidEmailAd...	< 1 ms
✓ ProfileEmailShouldBeInValidFormat	< 1 ms
✓ ProfilePropertesGettersAndSettersSho...	377 ms
✓ ProfileShouldAcceptValidEmailConfir...	< 1 ms
✓ RequiredProfileEmailShouldThrowError	< 1 ms
▲ ✓ UserTests (4)	377 ms
✓ RequiredUserEmailShouldThrowValida...	377 ms
✓ RequiredUserPasswordShouldThrowV...	< 1 ms
✓ UserPropertiesGettersAndSettersShou...	< 1 ms
✓ UserShouldBeValidWhenAllProperties...	< 1 ms

I tried to automate integration and systems testing as well using testing frameworks NSubstitute (2024) and Moq (2018) but always when I attempted to test, for example RegisterCommand.cs class I got “The process has no package identity.”

2.6. Evaluation

The system is evaluated in terms of performance using quantitative metrics and its scalability is evaluated based on overall architecture.

2.6.1. Functional evaluation

The application performs the use cases from UC-1, UC-2, UC-3, UC-4, UC-5 and UC-8 and UC-9 as expected based on functional tests in the previous section. However, there is still room for improvement to mimic native Windows applications and their smooth functionality better.

2.6.2. Performance evaluation

The system was evaluated based on response times for signing up, logging in, encryption and decryption with three tests per use case and got the following average results using Stopwatch class:

Test scenario: Register account (UC-1)	
Test 1	3630ms
Test 2	3692ms
Test 3	3014ms
Average	About 3445.34 ms (rounded up to 2 decimals)

Test scenario: Login (UC-2)	
Test 1	1724 ms
Test 2	2151 ms
Test 3	1758 ms
Average	About 1877.67 ms (rounded up to 2 decimals)

Test scenario: Encrypt files at rest (UC-3)	
Test 1	1722 ms
Test 2	932 ms
Test 3	1232 ms
Average	About 1295.34 ms (rounded up to 2 decimals)

Test scenario: Decrypt files at rest (UC-4)	
Test 1	1479 ms
Test 2	1018 ms
Test 3	923 ms
Average	About 1140 ms (rounded up to 2 decimals)

Test scenario: Back up files (UC-5)	
Test 1	1592 ms
Test 2	1216 ms
Test 3	1560 ms
Average	About 1456 ms (rounded up to 2 decimals)

Test scenario	Average response time
Register account (UC-1)	3445.34 ms
Login (UC-2)	1877.67 ms

Encrypt files at rest (UC-3)	1295.34 ms
Decrypt files at rest (UC-4)	1140 ms
Back up files (UC-5)	1456 ms

Based on the results, it seems Supabase API is slower than AWS services altogether. Amazon probably has better resources to handle the requests when compared to smaller actor Supabase. When registering account, it may take time for Supabase to handle insertion to the database through their Supabase Auth service and to send the verification link to the users who register to the application for the first time.

The application was completed using asynchronous operations where possible to make it more responsive to the user.

2.6.3. Scalability

The system uses scalable background infrastructure and technologies such as cloud based PostgreSQL, AWS KMS and AWS S3. Thus, the scalability of the application would not be an issue but the locally application is intended to be used on a single machine even though there could be multiple users in the AWS or Supabase if configured that way.

2.6.4. Challenges faced

Multiple times documentation of Supabase was lacking and .NET MAUI features felt like they were made more for mobile applications than native Windows desktop applications. Supabase user registration was overcomplicated as, for example, registration functionality in the API would let you think as a user that you could sign up to the application multiple times and that is why I had to implement separate table for checking that using plsql. Due to technological limitations and the premature nature of post-quantum algorithms themselves, project focused on session handling, effective encryption and performant operations in the perspective of ease of usability.

2.6.5. User Experience

The application attempts to mimic native Windows applications with smooth usability in terms of user interface looking modern, features working simply and intuitively. The application could be extended to other platforms as well so that it could be used with mobile devices as well but that would need more synchronization.

2.6.6. Results and outcomes

The project was successful as 8 out of 9 use cases were completed successfully and even some extra features like recursive folder loading for example. Authentication and authorization for the user files works well. The security using different AWS IAM roles, policies and permissions were implemented. Access control to AWS S3 bucket was enabled through bucket policy.

2.6.7. Limitations

Use case Restore files from backup (UC-6) was not completed due to time constraints of this project but the goal of backing up user files was nevertheless completed because the files could be downloaded straight from AWS S3 bucket to the local computer of the user through AWS console as well. Github actions for .NET MAUI project was too complicated to set up, so CI/CD was in use only through using Git and Github.

3.0 Conclusions

The project tries to address growing concerns related to the risk of common cyber attacks targeting home or work environment computer users and from “harvest now, decrypt later” post-quantum computational attacks. Secondly, the aim is also to mitigate data storage issues arising from GDPR requirements by applying post-quantum resistant encryption algorithms for data at rest (and in transit in the future research or implementation). Overall, the project was a success but could be developed further and be more modular so that it could be connected to multiple different service providers. This would be great project for the masters degree as well.

The official standardization process of post-quantum algorithms will take time and even if they would get official standardisation in the next upcoming years, the industry would still not have the same experience timewise as with traditional algorithms. This means that there may be vulnerabilities in the technological approach of these algorithms which have not been found yet, and which then again would render the usage of these algorithms dangerous from data protection perspective. However, for example Google has already adopted Kyber768 for their web browser and other systems but has had trouble with post-quantum TLS protocol this year (Future US Inc, 2024).

4.0 Further Development or Research

With additional time and investment, this project could evolve into an enterprise-grade product which could be sold at a reasonable price tag as a packaged comprehensive maintained data security solution in terms of price-quality ratio to individuals working remotely (for example, freelancers as well) or with other needs for storing sensitive data securely. There could be also grounds for additional research when post-quantum algorithms have been standardized and been in use for longer so that possible pitfalls have been discovered from the security perspective already.

5.0 References

.NET Foundation (2024) *About xUnit.net*. Available at: <https://xunit.net/> [Accessed 1 August 2024].

Amazon Web Services, Inc. (2024a) *AWS Key Management Service*. Available at: <https://docs.aws.amazon.com/kms/latest/developerguide/overview.html> [Accessed 1 August 2024].

Amazon Web Services, Inc. (2024b) *Amazon's Free Virtual Cloud Server*. Available at: <https://aws.amazon.com/free/compute/lightsail/> [Accessed 1 August 2024].

Amazon Web Services, Inc. (2024c) *Amazon S3 security*. Available at: <https://docs.aws.amazon.com/AmazonS3/latest/userguide/security.html> [Accessed 1 August 2024].

Amazon Web Services, Inc. (2023a) *AWS KMS concepts*. Available at: <https://docs.aws.amazon.com/kms/latest/developerguide/concepts.html> [Accessed 1 August 2024]

Amazon Web Services, Inc. (2023b) *Using server-side encryption with Amazon S3 managed keys (SSE-S3)*. Available at: <https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingServerSideEncryption.html> [Accessed 1 August 2024].

Amazon Web Services, Inc. (2023c) *Post-quantum hybrid SFTP file transfers using AWS Transfer Family*. Available at: <https://aws.amazon.com/blogs/security/post-quantum-hybrid-sftp-file-transfers-using-aws-transfer-family/> [Accessed 1 August 2024].

Arqit (2023) QuantumCloud™. Available at: <https://arqit.uk/quantumcloud> [Accessed 1 August 2024].

Cloudflare (2023a) *Cloudflare now uses post-quantum cryptography to talk to your origin server*. Available at: <https://blog.cloudflare.com/post-quantum-to-origins/> [Accessed 1 August 2024].

Cloudflare (2023b) *Do the ChaCha: better mobile performance with cryptography*. Available at: <https://blog.cloudflare.com/do-the-chacha-better-mobile-performance-with-cryptography> [Accessed 1 August 2024].

Functional Software Inc (2022) *Test-driven vs. test-later development: when should you write your tests?* [Accessed 1 August 2024].

Future US Inc. (2024) *Google Chrome's new post-quantum cryptography is causing some issues* [Accessed 1 August 2024].

Legion of the Bouncy Castle Inc. (2023) *The Legion of the Bouncy Castle C# Cryptography APIs*. Available at: <https://www.bouncycastle.org/csharp/> [Accessed 1 August 2024].

Microsoft (2021) *Windows 11 design principles*. Available at: <https://learn.microsoft.com/en-us/windows/apps/design/signature-experiences/design-principles> [Accessed 1 August 2024].

Microsoft (2018) *Unit Testing: Moq Framework*. Available at: <https://learn.microsoft.com/en-us/shows/visual-studio-toolbox/unit-testing-moq-framework> [Accessed 1 August 2024].

National Institute of Standards and Technology (2023) *Post-Quantum Cryptography | CSRC*. Available at: <https://csrc.nist.gov/projects/post-quantum-cryptography/faqs> [Accessed 1 August 2024].

NSubstitute (2024) *NSubstitute: A friendly substitute for .NET mocking libraries*. Available at: <https://nsubstitute.github.io/> [Accessed 1 August 2024].

OWASP (2024) *Cryptographic Storage Cheat Sheet*. Available at: [https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic Storage Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html) [Accessed 1 August 2024].

Resend (2023) *How to configure Supabase to send emails from your domain*. Available at: [How to configure Supabase to send emails from your domain · Resend](#) [Accessed 1 August 2024].

Statista (2023) *Global market share held by operating systems for desktop PCs, from January 2013 to July 2023*. Available at: <https://www.statista.com/statistics/218089/global-market-share-of-windows-7/#:~:text=Microsoft%27s%20Windows%20was%20the%20dominant,a%20fifth%20of%20the%20market> [Accessed 1 August 2024].

Supabase (2024a) *Supabase vs Firebase*. Available at: <https://supabase.com/alternatives/supabase-vs-firebase> [Accessed 1 August 2024].

Supabase (2024b) *Passwordless email logins*. Available at: <https://supabase.com/docs/guides/auth/auth-email-passwordless?queryGroups=language&language=js> [Accessed 1 August 2024].

Tutanota (2023) *The Race Is On: Tutanota Launches Development of Post-Quantum Secure Cloud*. Available at: <https://tuta.com/blog/pqdrive-project> [Accessed 1 August 2024].

6.0 Appendices

6.1. Project Proposal

6.1.1. Objectives

The primary objective of this project is to develop a secure and passwordless file storage and backup solution for Windows, incorporating post-quantum cryptographic techniques. The key goals include enhancing data security, simplifying user authentication, and ensuring robust protection against emerging threats posed by quantum computing while adhering to regulations such as European General Data Protection Regulation (GDPR).

One of the main objectives is also to create a working prototype which could at least encrypt the most common file types that office workers work with: Microsoft Word files (*.doc, *.docx, *.rtf), Microsoft Excel files (*.xls, *.xlsx), Microsoft PowerPoint files (*.ppt, *.pptx), PDF files (*.pdf), and Text files (*.txt).

The target user group of this application are Windows 11 desktop users because the largest portion of desktop systems in the world, 70 per cent of all desktop systems, still use the Windows operating system in the year 2023 according to Statista (2023). The application aims to appeal to the user group by offering a simple and cost-efficient, yet modern and easily marketable solution for remote or hybrid work environments.

6.1.2. Background

This project stems from the increasing need for advanced security measures in file storage and backup solutions in the future. This will be the reality also in light of the General Data Protection Regulation (GDPR) data handling requirements due to the

increase in remote after the coronavirus pandemic, especially considering the potential vulnerabilities posed by quantum computers.

The objective is to design, develop and implement a user-friendly yet highly secure system that eliminates the reliance on algorithms that are not ready for the post-quantum era for the data in transit, and at least AES-128 encryption for data at rest according to National Institute of Standards and Technology (NIST) (2023). To prevent external security issues arising from the use of traditional password-based authentication passwordless Windows Hello based authentication is used instead. The project will leverage cutting-edge standardized cryptographic methods such as the CRYSTALS (Cryptographic Suite for Algebraic Lattices)-KYBER ND-CCA2-secure Key Encapsulation Mechanism (KEM) algorithm and CRYSTALS-DILITHIUM Digital Signature Algorithm (DSA) to ensure data confidentiality through encryption, and both authenticity and integrity through digital signature.

6.1.3. State of the Art

Currently, some cloud storage and backup applications exist in the market and they are using quantum secure symmetric key encryption algorithms for data at rest. For example, Amazon Web Services with their S3 storage (2023b) encrypts data buckets using server-side encryption AES-256. From the perspective of post-quantum safe data in-transit solutions, both Amazon Web Services (2023c) and Cloudflare (2023) are forerunners of the new technology. However, there are some ready network protection solutions which utilize post-quantum ready algorithms such as Quantumcloud by Arqit (2023) but they are designed for enterprises rather than basic consumers and their pricing is not transparent at the moment.

There are no easy to use all-in-one desktop Windows 11 applications which would combine both data at rest encryption using passwordless authentication and a post-quantum secure backup solution. Additionally, since NIST has not yet officially standardized post-quantum secure algorithms, it may be difficult to launch new secure products yet to the market. There is financial value in the post-quantum market as demonstrated by the post-quantum cloud development project of Tutanota which was backed by German government with 1.5 million euro this year (Tutanota, 2023).

This project stands out by integrating post-quantum-ready advanced cryptographic methods seamlessly into a Windows environment using a beautiful user interface providing both a secure and convenient solution. The differentiation lies in the elimination of passwords, reducing the attack surface and enhancing overall security in a cost-efficient way because your security should not cost a fortune.

6.1.4. Technical Approach

The project will adopt an agile development methodology, ensuring adaptability to emerging challenges in more complicated projects and particularly Kanban to make project management simple and concentrate on one feature at a time. Kanban board will be used to manage the tasks visually.

This project will be divided into sprints of two weeks during which a specific set of tasks are completed and each sprint has one or more deliverables as milestones. For the work breakdown structure, the Gantt chart attached to this project plan can be found in Appendix 1.

Different requirements for the project are identified by understanding the purpose of the application and its target end-users. Secondly, researching present solutions or classical solutions will help in understanding their strengths, weaknesses, and common features to get a better perspective on the requirements and expectations for this type of application. User personas could be created to represent different types of users who might use this application. The personas have their needs, goals, and pain points which can be listed. Additionally, different user scenarios could be described and would include user actions, such as data encryption, cloud backup, and passwordless authentication. Essential functional and non-functional requirements could be listed based on these and then broken down further into sub-requirements with different priorities.

Continuous integration and continuous delivery (CI/CD) are utilized during the project to automate the building, testing, and deployment of the application. This contributes to guaranteeing the application maintains a deployable condition consistently, allowing for the swift and secure release of new features.

6.1.5. Technical Details

The implementation of the local desktop application will be in C#, leveraging the .NET version 6 for Windows compatibility. Principal libraries will include those supporting post-quantum cryptographic algorithms such as The Bouncy Castle Cryptography Library For .NET (2023).

Key Management Service (KMS) virtual private server will run on Ubuntu Linux, possibly running Docker container for Openxpki to act as a trusted authenticated agent between the local environment and Amazon S3 cloud backup storage. VPS implementation will use Amazon Lightsail.

Amazon S3 is a cost-effective object storage to be used for cloud backups in this project. Amazon S3 automatically encrypts all new objects by default with server-side encryption using Amazon S3 managed keys (SSE-S3) (Amazon Web Services, 2023b). Therefore objects stored in S3 are encrypted using 256-bit Advanced Encryption Standard (AES-256) before they are written to disk, and they are decrypted only when they are read back by authorized users. This ensures that the user data is protected from unauthorized access even if the S3 bucket is compromised.

Key algorithms under consideration include lattice-based cryptography, chosen for their resistance to quantum attacks, respectively CRYSTALS-KYBER together with classical Elliptic-Curve Diffie–Hellman (ECDH) forming Post-quantum (PQ) Hybrid Key Exchange in SSH (SFTP). Furthermore, for digital signature authentication, a standardized CRYSTALS-DILITHIUM algorithm will be used. The data will be transferred by taking advantage of the Transport Layer Security (TLS 1.3) protocol because it uses

the ChaCha20-Poly1305 encryption cipher suite, which is more secure than the older AES-GCM cipher suite (Cloudflare, 2023b).

6.1.6. Special Resources Required

Access to a testing environment that simulates quantum computing scenarios may be necessary. Moreover, possible consultation with a post-quantum cryptography expert and Microsoft Windows system architect regarding the reliability of the data in transit algorithm and long-term competence and experience with the Windows operating system environment could be of help with the project.

6.1.7. Project Plan

The project plan outlines a detailed timeline for development, with milestones such as requirements gathering, algorithm selection, prototype development, testing, and deployment.

Project start date: January 22nd, 2024

Project end date: August 5th, 2024

Week 1-2 (January 22nd - February 3rd)

Task: Develop a high-level system architecture and UI mock-ups for the application.

Deliverable: System architecture diagram with accompanying documentation and UI mockups.

Week 2-4 (February 4th - February 17th)

Task: Develop a detailed requirements specification for the application.

Deliverable: Preliminary requirements specification document for submission, may become more precise later.

Week 5-6 (February 18th - March 3rd)

Task: Implement the core functionality of the application, including post-quantum encryption and passwordless authentication. Conduct unit testing. **Preliminary requirements specification submission by March 3rd 2024.**

Deliverable: Working prototype of the application with core functionality implemented.

Week 11-12 (March 25th - April 7th)

Task: Implement additional features, such as cloud backup, Openxpki public key infrastructure key management service with a virtual private server, and digital signature algorithm CRYSTALS-DILITHIUM. Conduct unit testing.

Deliverable: Working prototype of the application with additional features implemented. Test reports documenting the results of unit testing.

Week 13-14 (April 8th - April 21st)

Task: Conduct integration testing.

Deliverable: Test reports documenting the results of integration testing.

Week 15-16 (April 22nd - May 5th)

Task: Conduct system testing.

Deliverable: System test report documenting the results of system testing. **Midpoint implementation, documentation and video presentation by 28th of April.**

Week 17-18 (May 6th - May 19th)

Task: Deploy the application to a staging environment.

Deliverable: Deployment report documenting the deployment process and results.

Week 19-20 (May 20th - June 2nd)

Task: Conduct user acceptance testing.

Deliverable: Performance test reports, security test reports, and user acceptance test reports.

Week 21-22 (June 3rd - June 16th)

Task: Fix any bugs found during user acceptance testing.

Deliverable: Bug fix report documenting the bugs that were fixed and the steps taken to fix them.

Week 23-24 (June 17th - June 30th)

Task: Retesting critical functionality, resolving any outstanding issues, and preparing the application for final deployment. Prepare the final deployment of the application.

Deliverable: Final test reports summarizing the results of all testing phases.
Deployment plan documenting the steps that will be taken to deploy the application to production.

Week 25-26 (July 1st - July 14th)

Task: Deploy the application to production.

Deliverable: Production deployment report documenting the deployment process and results.

Week 27-28 (July 15th - July 28th)

Task: Prepare the final documentation and video presentation.

Deliverable: Final documentation and video presentation.

Week 29-30 (July 29th - August 5th)

Task: Submit the final documentation and video presentation to Moodle.

Deliverable: Final documentation and video presentation submitted to Moodle by August 5th 2024.

6.1.8. Testing

Testing will include functional, unit and integration tests, and system tests to evaluate the resilience of the system. Furthermore, non-functional testing in terms of usability, performance and security will be executed.

Unit testing will concentrate on testing individual classes and functions with assertions to identify and isolate possible errors early in development to reduce the risk of bugs at that time and later on.

Non-functional testing utilizes performance testing to evaluate the application under different loads and usage conditions. It also includes security testing which identifies potential vulnerabilities and thus helps in avoiding incidents such as unauthorized access and sensitive data leaks.

Integration testing will test that the local Windows 11 desktop application integrates correctly with the Amazon S3 backup storage system using CRYSTALS-KYBER PQ-hybrid key encapsulation mechanism (KEM) implementation. Related to this process, testing that CRYSTALS-DILITHIUM digital signature algorithm (DSA) integrates with and that it integrates correctly with the local file storage and cloud backup system.

System tests will include testing that the system can successfully generate and exchange CRYSTALS-KYBER keys between two environments, local and cloud systems (Amazon S3). Testing that the system can successfully encrypt and decrypt data using CRYSTALS-KYBER keys is also necessary. Furthermore, for the cloud backups testing that the system can successfully store and retrieve encrypted data from a remote will be performed. Passwordless authentication is tested with practical authentication scenarios in which the application can successfully authenticate users using Windows Hello. Finally, testing that the system can successfully perform file storage and backup operations is required as well.

User acceptance testing ensures that the system is easy to use for end users, is performant and responsive and that it is also secure and reliable. Testing user-friendliness can be implemented with user personas, scenario-based testing, heuristic evaluation, and cognitive walkthroughs.

Performance testing and responsiveness of the application from the perspective of an end-user can be carried out by measuring the time it takes for the system to execute different tasks, such as generating and exchanging keys, encrypting and decrypting data, storing and retrieving data, and authenticating users. The tests can be repeated with a variety of different data sizes and system resource configurations using virtual machines.

Testing that the system is secure can be carried out by running known answer tests (KAT) for both CRYSTALS-KYBER and CRYSTALS-DILITHIUM. SAST (Static Application Security Testing) and DAST (Dynamic Application Security Testing) will be used to find possible vulnerabilities in the application code and mitigate them.

To test the system with real technical data, a variety of methods such as generating set of test files, could be made use of. Another testing method could be using multiple different file types and sizes to test the capabilities of the system. Furthermore, testing the system with high latency and low bandwidth networks will help define performance with different network conditions.

6.2. Reflective Journals

6.2.1. October 2023

6.2.1.1. What?

I pitched my final year project, a passwordless post-quantum file storage and cloud backup application for Microsoft Windows environment. Coming up with an application idea was not easy and I was able to select this one after critically evaluating my top five project ideas within the framework of usefulness to the end user, market needs for such an application, my competence, and available time resources. I also completed extensive research which included reading NIST publications, mapping already implemented applications, and studying possible existing post-quantum security implementations by industry leaders.

The passwordless authentication was a relatively new technical concept for me but I was able to select Windows Hello API for the task because the application will be implemented to work in Windows environment after all. At first, I was also considering Auth0 by Okta before I read that they had a recent security breach through their employees (social engineering) and decided that it is better to be safe than sorry in case that happens again in the future. For post-quantum encryption and decryption, I discovered that Amazon already offers a Key Management System (KMS) that should be post-quantum secure due to the algorithm it uses with in-transit data utilizing TLS encryption. For cloud backups, I envisioned using Amazon Glacier S3 since they should be able to make cloud backups post-quantum resistant to quantum computers soon enough, given that they have already implemented secure methods for Amazon KMS.

I am proud of the research I have done so far because I have already learned a lot about post-quantum cryptography and the current standards within that field. This research has been invaluable in shaping my project.

6.2.1.2. So what?

While doing the research before recording the pitch I felt a bit overwhelmed by the topic itself because it is so new to me, and the technologies themselves in addition to that. However, I am quite sure other researchers and students probably feel the similar way. However, after careful consideration I was able to critically evaluate the data available and select the most relevant information to determine which algorithms to utilize for my application. It was also helpful to learn about what industry leaders were doing to prepare for the post-quantum era.

The idea of giving a presentation on something that I do not fully understand yet felt intimidating initially. On the other hand, as I have got used to making presentations to my clients working as chief operations officer in a marketing agency, I knew I just had to prepare well and be mindful on what my audience would need to know about my project and this topic, and then present everything in coherent way. The pitch went well overall in terms of content in although the time to present all the technical details of the project idea was too short in my opinion. Nevertheless, I realized that the main idea was to present the idea in a lucrative way and briefly in this “elevator pitch”, and I just had to adapt my delivery of the presentation to that. As a sidenote, I think it never feels super natural when speaking only to a camera, especially without additional props but on the other hand, there is no pressure of the physical audience. It also felt challenging to talk about post-quantum project idea with confidence not having experience within that area in computing. Despite of these matters I am quite sure that I was able to include all of the necessary information though I was surprised at how quickly three minutes went by.

Pitching my project was an exceptional experience in the sense that I am sure it will provide me with valuable feedback for the future (from my upcoming supervisor), and furthermore, the whole research process preceding it gave me at some level clearer perception of the whole current state of post-quantum cryptographic era within the computing industry. Before starting this project I had no idea how much work there had been already done by individuals and similarly by the big players such as Amazon, Microsoft, Google, and IBM who clearly understood the importance of preparing for the age of quantum computing. This exposure has been instrumental in shaping the direction and focus of my project.

6.2.1.3. Now what?

I need to continue working on my project and ensure that I will be able implement all of the features I have proposed and make choices for the coding language and architectural choices within the project deadlines. The vision that I have for the user interface and coding language is not final yet but I will specify the specifics after having done a bit more research for my project proposal. I must also keep up with the latest developments in post-quantum security and make any necessary changes to my project as needed. The urgency of my work to be completed is highlighted by the very recent piece of news in which a known researcher claims to have cracked the RSA-2048 algorithm. If his claim is true, the industry should already be using the post-quantum algorithms now.

Moving forward, my focus is on continuing the development of my project, ensuring the implementation of all the proposed features, and keeping pace with the latest developments in post-quantum security. I also will report my work to my upcoming supervisor in the future reflective journals which also promotes my learning throughout this whole project process. Adaptations and modifications to the application itself may be necessary as the project progresses, but I am confident that I can successfully complete this undertaking and contribute substantially to the domain of post-quantum security.

6.2.2. November 2023

6.2.2.1. What?

My supervisor was finally assigned to me and he is Rohit Verma. Immediately after I heard the supervisors had been assigned, I promptly sent him an email regarding my project idea and suggested a meeting in Teams to discuss project-related issues and the common plan.

At the start of the meeting, he asked me to explain my idea to him so that we had a common understanding of the project idea. He understood quickly how this solution would benefit the target users and that we should concentrate on the implementation technologies and strategy.

We had a one-hour-long meeting during which we discussed different approaches to my project idea depending on the current technologies available in the market right now. We decided that we could use the CRYSTALS-KYBER algorithm for data in transit meaning transferring data to Amazon Glacier or another suitable, possibly cost-free cloud environment which I could investigate a bit further. For development and prototyping purposes saving costs would be better than investing capital in the idea at this stage as this is an academic project after all. Similarly, instead of using Amazon KMS which is a paid service, I would be utilizing OpenXPki for the key management system and X.509v3 certificates. Their system is an enterprise-grade solution and provides support for the latest post-quantum technologies where possible.

Programming language and project architecture were discussed in the context that I wanted to develop a desktop application for general computer users or workers who would be mostly using the Windows operating system. We were also thinking of possibilities in terms of post-quantum application testing which is at the moment challenging as there are no quantum computers available to use for testing the implementation of Kyber and Dilithium.

6.2.2.2. So What?

After having a debate with Rohit regarding my current skillset and the difficulty of implementing the local client application in a Windows environment, I concluded that C# would be a better option for this project because Microsoft owns its rights and for that sake, it could be used extremely well for this software development project. After my research on suitable technologies, I had previously been wondering whether to implement this application with C++ or C# with .NET framework of which C++ would have more performance than C# but only if configured right. Furthermore, for a beginner with the coding language, either C++ or C#, the latter would be faster to develop and it would probably turn out to be less challenging for me due to manual nature of C++ configurations such as memory optimization.

The main challenges at this stage after my initial research were how to secure the data in transit and at rest. In transit data would be secured using CRYSTALS Kyber-768 and for Dilithium for digital signatures. I did not know at our meeting how to make data at rest post-quantum secure as well but later after researching more on the topic, I found out that symmetric-key algorithm AES-256 could be utilized for that purpose. In the meeting with Rohit we had already come up with backup plan in case there were no post-quantum solutions for data at rest, I would only transfer the data securely using Kyber and Dilithium with intermediary KMS server, and then, for example, Amazon S3 would probably be quantum secure at the timeframe of application deployment in the year 2024.

6.2.2.3. Now What?

The most challenging issues at the moment are the perception on how these architectures and technologies fit together and how to form a reasonable project plan in the project proposal by the deadline. I also still have to research more on free tier cloud server

providers to implement KMS and file storage for the application and the testing methods to form a high-level test plan.

For client authentication TLS1.3 protocol with Kyber should be implemented for the traffic between the client and the KMS server and the storage server, however, TLS1.3 with Kyber has not been standardized yet and there may be challenges in respect of packet fragmentation due to the size of the algorithm key share. I may have to have a meeting with my supervisor regarding this and how to take this technically into account.

Overall, the project architecture and implementation do not seem obscure to me as a whole anymore and I have learned more about different approaches on how to tackle post-quantum computational threats. For example, I was surprised to find out that current AES-256 algorithm will probably be sufficient for a long time for encrypting the data at rest. To break the algorithm, it can be mathematically calculated that at least a million qubit quantum computer would be required and the current technological level is at a bit above thousand qubits machines. However, nobody knows how fast the technology may advance in the future but such a jump in qubit numbers would be carefully estimating unlikely in the next five years at least.

6.2.3. December 2023

6.2.3.1. What?

I made progress with the project by doing additional research for the required proposal that I submitted on the 20th of December 2023 just before Christmas. From C# libraries for Windows that I had envisioned for the project at my earlier coding language considerations, Bouncycastle seemed suitable for my CRYSTALS-KYBER and CRYSTALS-DILITHIUM implementation because of its ready NuGet package available for .NET framework. I was also able to choose Amazon S3 as my final service for cloud backups and Amazon Lightsail for Openxpki trusted intermediary Key Management Service (KMS) server. Amazon has a free tier that I can make use of which led me to choose it as the cloud service provider.

The software development approach was still unclear to me at the beginning of writing the proposal. Nevertheless, after looking into suitable methodologies, I decided to go with agile and more specifically Kanban as this is a one-man project and requirements specification is still to be defined at the next phase. I created a Gantt chart with the free trial of Monday.com work management cloud application. The chart consists of two-week sprints and relevant milestones and is attached to the proposal of the project.

6.2.3.2. So What?

After being able to move forward with the selected development software development methodology based on the nature and technical aspects of the project and creating the project roadmap in the form of Gantt chart, it feels like I have reached one step closer to the actual implementation from abstract to concrete level. Although I have not previously used Amazon or Openxpki for development projects, they have such a large user base and are both enterprise-grade solutions that I am positive they will prove to be great tools to work with despite their somewhat complex nature.

Similarly, C# is a rather old language by today's metrics for the age of coding language and .NET has had time to mature. Requirements gathering and design phase of the project are critical for the later success of the project and my goal is to accommodate SOLID principles

with relevant design patterns for this project to ensure scalability and maintainability of the application in the future as well. For example, algorithms may still change because they are not yet fully standardized by the National Institute of Standards and Technology or they may need to be altered.

While preparing the proposal for submission, I also had to address concerns regarding testing, particularly unit, integration and end user testing. Thus, the required functionalities and components should be designed in a way that relevant testing can be performed straightforwardly on them. I follow this ideology during the whole software development lifecycle.

6.2.3.3. Now What?

The next logical step of the project is to start the creation of a high-level system architecture diagram to provide a better understanding of the interrelations of different components for me as the project manager and team member and for the project supervisor who can be seen as a stakeholder. The creation process of the diagram may help me spot flaws in my software design and to reflect on the selected connections between the components in terms of project requirements. Therefore I may need to refine the diagram during and after the process of writing the preliminary software requirements specification. The diagram and related textual documentation also help in visualization of the overall software structure which helps to prevent functional or integrational issues further down the software development lifecycle during this project.

Furthermore, I will create wireframes and mid-fidelity user interface mockups to guide me in the software development process in a cohesive way and on the other hand, they help meet the expectations of the stakeholders. The appearance of the final product must be appealing to the target end-users because otherwise the whole project may end up not meeting with its original purpose.

6.2.4. January 2024

6.2.4.1. What?

This month, I dove headfirst into the creation of the high-level system architecture diagram and UI mockups as specified in the project plan that I created in December 2023. It was not just about ticking a box in the project plan; it was a small journey of discovery. Starting with just abstract ideas, I felt a thrill as they transformed into concrete visuals. It was like taking a blurry photograph and sharpening it into focus. Suddenly, the connections between different parts of the application became clear, and I could almost see the potential problems lurking in the shadows. For example, I had to look into OpenXPki documentation again to ensure it worked as intended in the framework of the project and to find out more about Amazon S3 as a multi-cloud environment.

The early design was not all smooth sailing. Despite all the research I had already done before, there were moments of frustration, staring at a blank screen trying to translate my thoughts into a clear design between components and their roles in the system. At that moment, I decided to take a deep breath and face the problem head-on. Eventually, the pieces clicked into place and a sense of accomplishment.

Prioritizing user experience on an ideological level while translating abstract ideas into visual representations brought remarkable clarity to my software design. The system architecture diagram exposed interconnections between components, allowing me to

identify and rectify potential flaws early on. This saved valuable time and effort compared to encountering issues during actual development.

6.2.4.2. So What?

The proactive approach produced significant benefits in shaping the project towards user-centered functionality and design. By constructing UI mockups, I was not only planning the internals of the system; I was visualizing and iterating on the user interface itself. Investing time in these visual aids was not just about efficiency. By crafting the UI mockups, I also stepped into the shoes of the user. It made me think about how they would interact with the application, what would be intuitive, and what might cause confusion. It felt like a shift in perspective, a paradigm change that forced me to design with the user in mind.

6.2.4.3. Now What?

As I prepare for the requirements gathering phase, I cannot help but feel excited and a little apprehensive. The research I have done on environments, technologies, and libraries has given me a solid foundation, but there is still so much to learn. We have reserved a time slot together with my supervisor for the next week on the 8th of February to discuss and review my project proposal. This upcoming meeting feels crucial. It is an opportunity to get valuable feedback, to make sure I am on the right track before diving deeper into the specifics. I hope to walk away with even clearer roadmap that I have right now, but I am also prepared to adapt and adjust as needed. This project is an ongoing exploration and learning experience, and I am ready to embrace the challenges and opportunities that lie ahead.

6.2.5. February 2024

6.2.5.1. What?

We had a brief meeting with my supervisor on the 8th of February to discuss my project proposal and the progress so far. I explained the initial analysis and design of my progress through the demonstration of a high-level architecture diagram of the distributed system and showcased the user interface of the application through the first version of a mockup that I had created. The application idea began to take its shape even at a more detailed level this month because I concentrated on the delivery of preliminary requirements specification as stated in my project plan.

Being struck down by illness for a week threw my schedule off track. This unexpected hurdle forced me to re-evaluate my time management strategies. While I managed to catch up, I realized the importance of building buffer periods into my plans, acknowledging that unforeseen circumstances can always arise.

6.2.5.2. So What?

When defining the requirements for the application, I did further research on how to store the minimal user data for the user accounts. I had an eureka moment and realized that I would have to use some generally used light database options for user data to support the creation of user accounts, providing sessions and simple user data management for name email. While crafting a use case diagram, I noticed that a recovery use case would be essential in the worst-case scenario of the user losing access to their computer, for example, due to device malfunction or theft.

Focusing on the requirements specification of the project proved to be a rewarding experience. While researching user data storage, I experienced an "aha moment" that led me to explore lightweight database options. This discovery significantly impacted the direction of the project, highlighting the critical role of continuous research and learning in software development.

Furthermore, I realized that Windows Hello does not really by itself perform the authentication but helps to communicate with underlying layers of the system. Having understood that, I figured out I could use Windows 11 Trusted Platform Module (TPM) version 2.0 for authentication and for storing encryption and decryption keys for local data encrypted with AES-128. In its current state, TPM 2.0 would appear to be not quantum resistant but new quantum-resistant TPM is on its way and the industry seems to rely on TPM as technology in the post-quantum future as well.

I also had to do some calculations for the file storage from the perspective of possibly encrypting tens of gigabytes of user data in the application. While selecting and analysing encryption algorithms, I encountered a fascinating trade-off. While AES-256 offered superior security compared to AES-128, its computational demands were impractical for the target hardware commonly available to the end users of the application. This realization underscored the importance of considering resource constraints when making technical decisions.

6.2.5.3. Now What?

Though the initial requirements are outlined, I acknowledge the need for further refinement, particularly regarding file backup and account recovery functional requirements. The iterative nature of software development requires constant evaluation and adaptation. Similarly, non-functional requirements may need some fine tuning.

Looking forward, I am excited to establish a continuous development environment for my Windows 11 application. Developing core functionalities like user registration, login, and data encryption and decryption, alongside thorough unit testing, will be crucial steps in solidifying the foundation of the project.

Overall, this month has been a period of significant growth and learning. The unexpected challenges, coupled with the valuable guidance from my supervisor and my own research, have equipped me with invaluable experience and knowledge. As I move forward, I am determined to leverage these learnings to develop a secure and robust application prototype. I feel especially motivated by the opportunity of transforming my initial idea into reality.

6.2.6. March 2024

6.2.6.1. What?

I started the implementation of the application utilizing C# and .NET MAUI according to the preliminary requirement specification document that I had created at the beginning of March 2024. Development for Windows platform is all new to me since the only experience I have is coding applications for Linux environment. Having created Github repository for my project, I decided to use Github Actions for CI/CD as it is familiar to me from the Cloud Application Development module. The first core functionalities of my project are registration and login which I am currently working on using Visual Studio 2022.

6.2.6.2. So What?

During the preparations of my Windows development toolset, I found out that when developing with .NET Multi-Platform App UI (MAUI) it is beneficial to utilize Model-View-ViewModel (MVVM) design pattern to decouple the core functionality of the application from its interface layer. After researching deeper into the .NET architecture, I felt that it would be beneficial to choose a more modern cross-platform framework for the development because older technology, Windows Presentation Foundation (WPF) will be replaced by the MAUI in the future and this application project would benefit from the expandability for future research or development. However, some of the basic functionality such as cursor pointer hover events on buttons seem surprisingly challenging to implement with MAUI as they have to be implemented using separate view models. Despite the initial difficulties, I have learned to use Extensible Application Markup Language (XAML) layouts to create relevant view presentation logic which is bound to related background functionality.

Continuous assessment tasks of other modules have taken more time than expected which has delayed the schedule of this project. Nevertheless, I have done my best with the limited time resources I have as a part-time student to progress the software development and to avoid burnout.

6.2.6.3. Now What?

To prepare for the midpoint presentation at the end of this month, I will complete at least the login core functionality of the application and depending on the steepness of my learning curve with the new technology to me, possibly user registration as well. Getting accustomed to the development workflow for Windows environment is vital to the success of this project.

The last month has been extremely busy in terms of work and school work. Yet I feel determined to push my limits and learn more to get everything ready by the end of this month to be able to showcase my development work so far and to answer the Q&A questions during the video demonstration. If I could turn back time, I wish I had more time to implement this project but I will adapt to the tight time constraints as well as I can.

6.2.7. April 2024

6.2.7.1. What?

I was excited that I was able to set up .NET MAUI development environment and work with this new cross-platform framework using the latest Visual Studio 2022. Unfortunately, database migrations using any other database than SQLite did not have straightforward tutorials. Furthermore, SQLite would by itself not necessarily be very scalable unlike other options such as PostgreSQL which Microsoft claimed .NET MAUI would support. Nevertheless, I was able to create a dummy project and perform the migration following one somewhat similar YouTube tutorial and referencing to this data access project in my .NET MAUI main application. So, I could use that database for my application if I wanted but I noticed that free plan of Firebase authentication by Google would enable me to achieve user registration and login more securely using their external API. This would also make sense because Google recently started using quantum resistant X25519Kyber768 encapsulation in their browser in production. The only problem now is that while Google has provided simple enough steps to implement passwordless email

link for signing in for Android and iOS, they have not done the same for desktop environment.

6.2.7.2. So What?

Having done some additional research, I realized that I could not use Windows Hello API as itself for authenticating users as combining Microsoft user account with a generic application account would cause security risks. On the other hand, Windows Hello for Business would be able to perform such functionality and the passwordless biometric authentication if the device has necessary hardware for the authentication, although this would require device management at the corporate level which would then again introduce vendor lock possibly also involving high costs due to Microsoft 365 licensing required. Because one of my core principles when starting this project was that anybody could easily download and start using this program with rather simple configuration steps and low costs, this would not help achieve this target. Despite of this, I also evaluated the option to implement the authentication using Windows Hello for Business and the

justification based on the cost structure. Would it perhaps be more reasonable to use corporate tools already available for this project even if an individual or organization must pay a little more?

During my work with the database migration, I also reconsidered my approach in terms of security and ease-of-use. I noticed Google Firebase would provide free features for thousands of users to authenticate through their Firebase API and I was able to implement registration after learning from several YouTube videos how to connect to the API. Firebase would be easy for the end-users to set up and it offers great scalability as well. However, the current problem is that I probably must create separate API with ASP.NET Core or similar technology to support passwordless sign in through their API as Google does not seem to be support Windows desktop environment in their documentation.

6.2.7.3. Now What?

Unit testing for the registration and login functionalities need to be implemented next. This is important for the project in the longer run when the application gets more complex and loosely coupled parts will need to work fluently. Additionally, I am getting ready to present and record the mid-point presentation to show my progress so far. After that I will concentrate on implementing the next core functionality which is the encryption for the files at rest. I have initially planned to have that completed during May and June as these features take more time to develop than I thought originally.

6.2.8. May 2024

6.2.8.1. What?

I am currently a bit stuck with the passwordless login utilizing Firebase. There does not seem to be any similar implementations or resources available that would especially target .NET MAUI windows application development with C#. However, I found out about Firebase open-source alternative which had just finished their beta phase and would be production-ready cost-efficient competitor with dedicated PostgreSQL capabilities. Thus, I started it would be wiser to implement at least the first two use cases using Supabase service as Firebase replacement. Additionally, quantum safe OpenSSH library that I had planned for establishing remote connection for file transfers had become inactive.

6.2.8.2. So What?

Having faced major technological challenges in respect of the application development, I felt lost. Nevertheless, I did not give up but kept trying to find alternative solutions to help me solve the current functional problems as specified in my requirement specification document. While browsing through documentation of Firebase and different other players such as Auth0, I also realized that the development of mobile application would have been easier than creating Windows application with the latest .NET MAUI framework. Even the framework itself has technical problems in user authentication according to Microsoft online documentation as for instance, web authenticator does not work at all. This really surprised me as I thought the latest framework would at least fully support the basic functionality such as authentication.

6.2.8.3. Now What?

Due to technical limitations and development time constraints, it is reasonable at this phase to implement the functional requirements without quantum algorithms planned and move on with AES-256 encryption at rest with decryption keys stored in Amazon KMS. I will replace the current Firebase registration and login functionality with the respective Supabase Auth and their PostgreSQL database for other user data.

6.2.9. June 2024

6.2.9.1. What?

I have managed to implement the passwordless one-time password login after initially registering the user for Supabase Auth. Furthermore, I managed to connect to Supabase PostgreSQL and to create separate public users table for authentication using Argon2id for password hashing and later, user verification and authorization. However, I quickly realized that Supabase auth by itself provides auth.users table in which it enables the user registration by itself and my solution would be redundant. Therefore, I decided to delete the code and just move forward utilizing Supabase Auth for basic user management and login. I managed to set up Amazon Lightsail VPS with automatically renewing SSL certificates by Let's encrypt to redirect email verified users to my ladeapp.live domain to inform them about the successful user registration.

6.2.9.2. So What?

Supabase has provided documentation on their website for C# library, but the extent is rather limited compared to other language libraries such as JavaScript. This makes it difficult to implement some basic features such as listening to authentication events as the documented pieces of code need certain name spaces and other definitions to work as described. Simple things such reading email_created_at from users.auth table of the service is restricted for security reasons and there is not really documented ways around the problem. This makes me feel frustrated as I know that I can implement simple user authentication according to the documentation very well but the technology itself does not have sufficient methods of executing the login flow properly. Setting up the VPS at Amazon with secure HTTPS was more straightforward process and it felt simple compared to the development work related to Supabase user login and one-time password system.

6.2.9.3. Now What?

I will finish the passwordless user login and move on to the encryption and decryption part of my technical specification document so that the end-users will be able to secure

their work-related assets. After that I will concentrate on backup functionality and logging from security perspective. At the end of the month, I have allocated time for documentation and preparation for the final submission of the whole project.

6.2.10. June 2024

6.2.10.1. What?

After having completed user registration and login functionalities use cases I created user space view for the user file management in a local folder. Additionally, I have completed and managed to get envelope encryption working using Amazon Web Services Key Management Service (AWS KMS) for data keys together with AES-256 GCM for the file encryption itself. To prepare for backup use case, I configured Amazon S3 with suitable permission policies and attached them to the user group.

6.2.10.2. So What?

I had no previous experience working with AWS systems so hands on. Therefore, I was excited to learn how to manage different permission policies through groups and create my own policies based on required AWS permissions of the application features. The system seemed rather overwhelming at first but then I realized how the role management works effectively.

Additionally, I had major issues with data key decryption process until I finally recognized that my encryption context had been different the whole time even though I thought they were matching. After checking that they matched, I was able to successfully continue to the next process with AES encryption.

Envelope encryption was a new practical concept for me to implement and there was a lot of trial and error, for example, with encryption padding while trying to implement AES-256 after being able to successfully connect to AWS KMS and utilize the decrypt function to get the plaintext data key for encryption. At first, I thought I had to use encrypt function through AWS KMS but then after I had spent multiple hours reading Amazon documentation carefully and researched online, I internalized the whole process and completed the code.

6.2.10.3. Now What?

I am currently working on the backup and recovery use cases and finalizing both my application and technical documentation for the final review and showcase. It has been an extremely busy summer schedulewise, and I feel accomplished to have learned numerous new things such as a completely new framework (.NET MAUI), C# as programming language, using an external database applying novel Supabase API and adopting multiple AWS services for the project. This final project is an important big step for me to become an experienced software developer and cybersecurity expert.