04/03/2025

**Design Document**

Version 1.1

Ayodeji Olagoke

Rares Matei Husarescu

# **Version Control**

|  |  |  |
| --- | --- | --- |
| **Version number** | **Date** | **Changes** |
| 1.0 | 03.03.2024 | Initial version of the document |
| 1.1 | 04.03.2025 | Adding chapters and updated ERD diagram |
| 1.2 | 02.04.2025 | Added final version of the Class Diagram |
| 1.3 | 03.04.2025 | Added final version of the Use Case Diagram. |
| 1.4 | 04.04.2025 | Minor make up’s improvements |

Table of Contents

[Version Control 1](#_Toc194611450)

[Chapter 1: Introduction 2](#_Toc194611451)

[Description of New System 2](#_Toc194611452)

[Chapter 2: UML Diagram 4](#_Toc194611453)

[Chapter 3: Class Diagram 5](#_Toc194611454)

[Chapter 4: Data Model ERD 6](#_Toc194611455)

[Chapter 5: Testing 7](#_Toc194611456)

[6.1 – Menu Overview 7](#_Toc194611457)

[6.1 – Authorization Matrix 7](#_Toc194611458)

[Chapter 7: Organizational Consequences 8](#_Toc194611459)

[7.1 – Continuous Test Strategy 8](#_Toc194611460)

[7.2 – Testing Methodologies 8](#_Toc194611461)

[7.3 – Stakeholder Engagement and User Acceptance 8](#_Toc194611462)

[7.4 – Implementation and Training 8](#_Toc194611463)

[7.5 – Quality Assurance Criteria 9](#_Toc194611464)

[Backup Strategy 10](#_Toc194611465)

# **Chapter 1: Introduction**

The purpose of this document is to provide an overview of the design and architecture of a Netflix-like streaming platform. The system will allow users to register, create profiles, watch films and series, and manage their subscriptions. The platform will include different user roles such as Admin, Mediator, and Junior staff for content management.

# **Description of New System**

**Functional Requirements**

* User registration with email and password.
* Email activation for new accounts.
* Password recovery through email.
* User profile creation (maximum 4 per account) with name, age, and profile photo.
* Ability to set language preferences.
* Subscription plans: SD, HD, UHD with a free 7-day trial.
* Content watch history and personalized recommendations.
* Watch list management.
* Invite a user for a discount on subscription.
* Admin functionalities including content creation, system configuration, and financial reports.
* Mediator and Junior staff roles for limited content management.

**Non-Functional Requirements**

* Secure authentication and access control.
* Data persistence and reliability.
* Scalability to support high volumes of streaming.
* Compliance with content restrictions based on age.
* Optimized streaming quality based on user subscription.

# **Chapter 2: UML Diagram**

Immagine che contiene testo, design

Il contenuto generato dall'IA potrebbe non essere corretto.

# **Chapter 3: Class Diagram**

Immagine che contiene schermata, diagramma

Il contenuto generato dall'IA potrebbe non essere corretto.

# **Chapter 4: Data Model ERD**

Immagine che contiene diagramma, Piano, Disegno tecnico, schematico

Il contenuto generato dall'IA potrebbe non essere corretto.

# **Chapter 5: Testing**

Test Type Descriptions:

* **Unit Test -** Verifies the smallest functional part of the application in isolation — in this case, testing login logic with correct credentials and proper token generation.
* **Authorization Test** - Confirms that access to specific API endpoints is properly restricted based on authentication status or token validity. These tests ensure that only authorized users can retrieve protected data.
* **Functional Test –** Test the actual behavior of an endpoint or feature to ensure it works as expected when valid or invalid data is passed — such as requesting a user that doesn't exist.
* **Security Test** - Ensures the system follows secure practices, such as never exposing sensitive information like plain-text passwords in API responses.
* **Error Handling Test** - Validates how the system responds to unexpected internal failures (like database issues) and ensures it returns meaningful and safe error messages instead of crashing.

Here Unit Test:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | |  | | --- | | **Test Type** |  |  | | --- | |  | | **Description** | **Preconditions** | **Steps** | **Expected Result** |
| TC\_LOGIN\_001 | Unit Test | User login with valid credentials | |  | | --- | |  |  |  | | --- | |  |   User exists with valid email & password; account is active | 1. POST to /api/Login with correct credentials 2. Inspect response | 200 OK Token returned, 401 Unauthorized for invalid credentials |

Here Authorization Test:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | |  | | --- | | **Test Type** |  |  | | --- | |  | | **Description** | **Preconditions** | **Steps** | **Expected Result** |
| |  |  |  | | --- | --- | --- | | |  | | --- | | TC\_AUTH\_001 |  |  | | --- | |  | |  |  | | --- | |  | | |  | | --- | | Authorization Test |  |  | | --- | |  | | |  | | --- | | Access /api/User with valid JWT |  |  | | --- | |  | | User exists but account\_status = inactive | 1. Send GET /api/User with Authorization: Bearer <token> 2. Observe status and data | 200 OK List of users returned |
| TC\_AUTH\_002 | |  | | --- | | Authorization Test |  |  | | --- | |  | | |  | | --- | | Access /api/User without JWT |  |  | | --- | |  | | None | |  | | --- | |  |  |  | | --- | | 1. Send GET /api/User with no Authorization header | | 401 Unauthorized |
| TC\_AUTH\_003 | |  | | --- | | Authorization Test |  |  | | --- | |  | | |  | | --- | | Access /api/User/{id} with invalid token |  |  | | --- | |  | | |  | | --- | | Invalid or expired JWT token |  |  | | --- | |  | | |  | | --- | | 1. Send GET /api/User/1 with Authorization: Bearer <fake\_token> |  |  | | --- | |  | | |  | | --- | | 401 Unauthorized |  |  | | --- | |  | |

Here Functional Test:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | |  | | --- | | **Test Type** |  |  | | --- | |  | | **Description** | **Preconditions** | **Steps** | **Expected Result** |
| TC\_GET\_001 | |  | | --- | | Functional Test |  |  | | --- | |  | | |  | | --- | | Get user that does not exist |  |  | | --- | |  | | User with ID 9999 does not exist Valid JWT token | |  | | --- | | 1. Send GET /api/User/9999 with valid token |  |  | | --- | |  | | |  | | --- | | 404 Not Found |  |  | | --- | |  | |

Here Security Test:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | |  | | --- | | **Test Type** |  |  | | --- | |  | | **Description** | **Preconditions** | **Steps** | **Expected Result** |
| |  | | --- | | TC\_SECURITY\_001 |  |  | | --- | |  | | Security | |  | | --- | | Ensure password is never exposed in API |  |  | | --- | |  | | |  | | --- | | Users exist |  |  | | --- | |  | | |  | | --- | | 1. GET /api/User with token |  |  | | --- | |  | | |  | | --- | | Password field should be hashed or excluded entirely from response |  |  | | --- | |  | |

Here Error Handling Test:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | |  | | --- | | **Test Type** |  |  | | --- | |  | | **Description** | **Preconditions** | **Steps** | **Expected Result** |
| |  | | --- | | TC\_ERROR\_001 |  |  | | --- | |  | | |  | | --- | | Error Handling |  |  | | --- | |  | | |  | | --- | |  |  |  | | --- | | Simulate DB failure | | |  | | --- | | DB connection is down |  |  | | --- | |  | | |  | | --- | | 1. GET /api/User while DB is down |  |  | | --- | |  | | 500 Internal Server Error , Graceful error message |

**Chapter 6: Menu Structure & Authorization**

This chapter outlines the application's menu structure and the corresponding authorization levels assigned to different user roles. It details the navigational framework and access permissions, ensuring a secure and role-appropriate interaction with the system's functionalities.

## **6.1 – Menu Overview**

* **View Data:** Offers comprehensive insights into financial data.
* **Overall Expenses Per Card**: Displays the total expenditures made using each card.
* **Monthly Expenses**: Summarizes expenses incurred in a month.
* **Expenses Per Category**: Breaks down expenses according to predefined categories.
* **Current Balance Per Card**: Shows the real-time balance of each card.
* **Personal Account & Settings**: Allows users to view and modify their account settings.
* **Data Entry**: Dedicated to the input of financial data in the MT940 standard format.

|  |  |
| --- | --- |
| Menu | Overall view of all possible data, submenu |
| View Data | Access to the overall expenses per card |
| Access to monthly expenses |
| Access to categories trasnsactions |
| Current balance per card |
| Access to personal account and settings |
| Data entry | MT940 format |

## **6.1 – Authorization Matrix**

The table provided delineates the permission levels for both regular users and administrators. The permissions are defined as follows:

* **R (Read):** Ability to view data.
* **W (Write):** Privilege to create or add new data.
* **U (Update)**: Permission to modify existing data.
* **D (Delete):** Authority to remove data.
* **P (Prohibited Access):** No access rights.

|  |  |  |  |
| --- | --- | --- | --- |
| **Menu Structure** | | **Authorization** | |
| **Main Menu** | **Sub Menu** | **User** | **Administrator** |
| **MT940** |  |  |  |
|  | Uploading | P | RWUD |
| **Accounting Menu** |  |  |  |
|  | Link to cost center | R, U | RWUD |
|  | Create cost center | P | RWUD |

# **Chapter 7: Organizational Consequences**

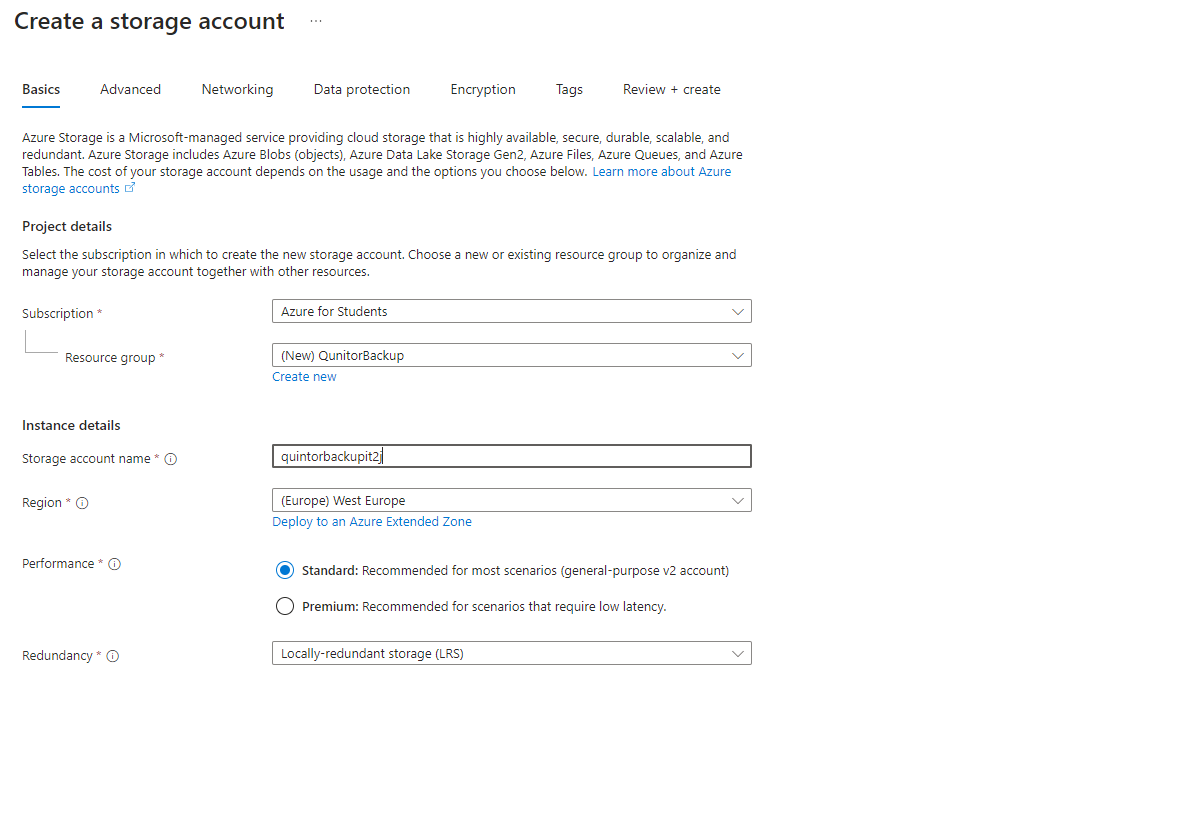
This chapter details the strategic approach to testing and quality assurance that the IT3J team will employ throughout the development lifecycle of the project. The methodologies adopted not only aim to ensure the delivery of robust and secure code but also facilitate the integration of the product into the organizational structure with minimal disruption.

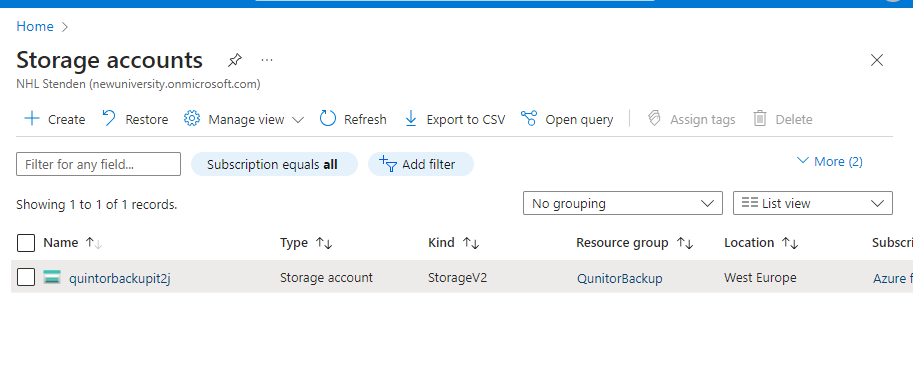
## **Backup Strategy**

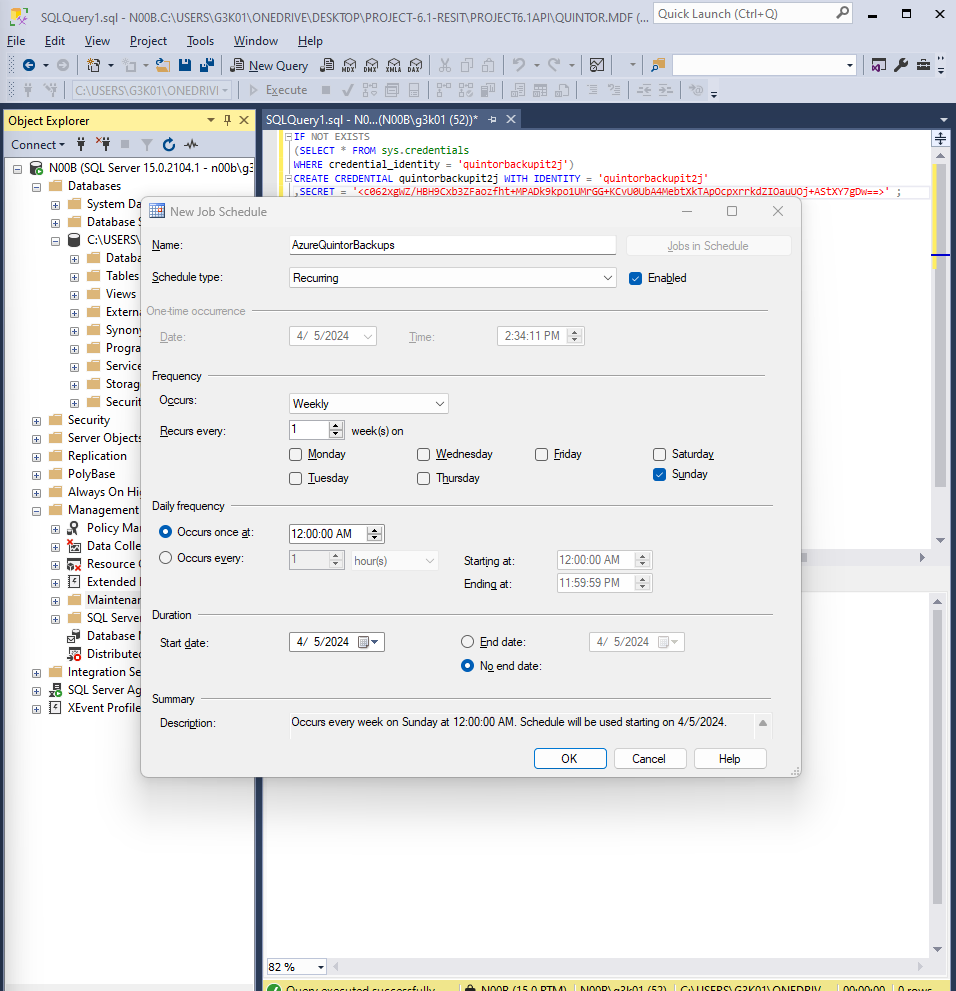
While our databases are hosted locally, the data storage process runs over a backup set up in Microsoft Azure. This choice has been made based on the 2 backup options of SSMS. The team had the choice to back up the database either on a local disk or online through URL.

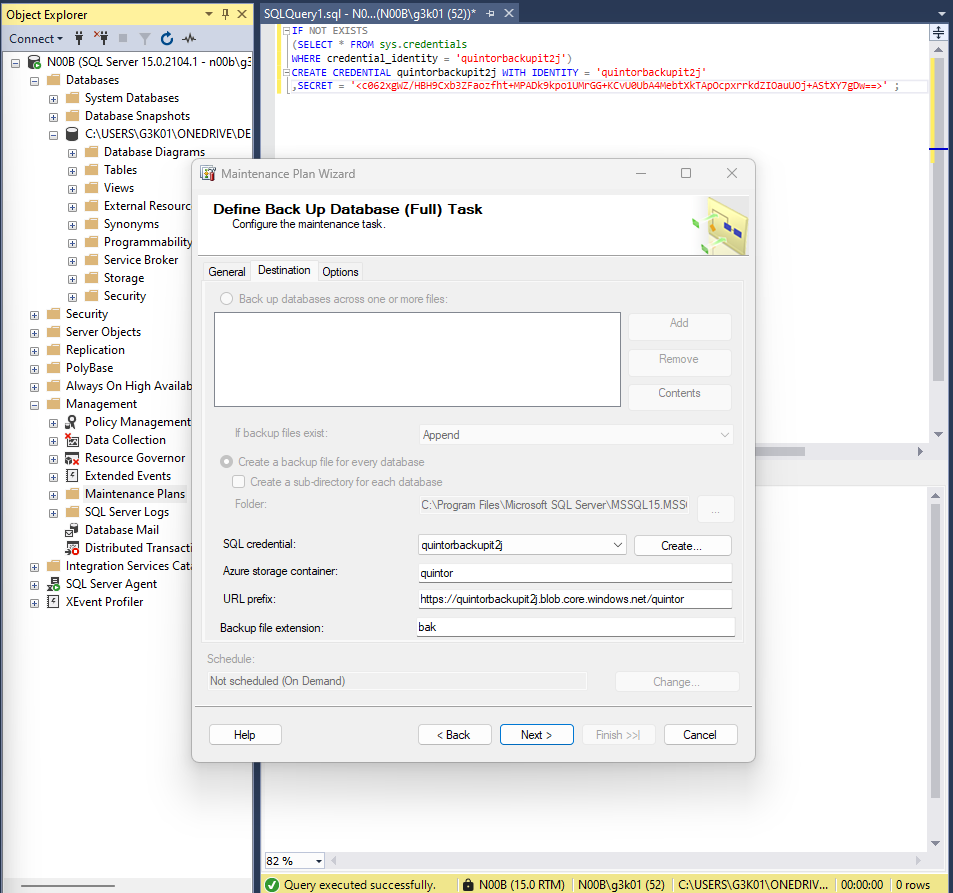
Since this is an external project (external in: client outside of the NHL Stenden university), the local storage option was not up for discussion. Therefore, the group set up and used an Azure container, where the database is saved iteratively, every 7 days.

In the following picture is the storage account presented with the basic information.









Proof of backup creation

