Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти – другий (магістерський)

Дослідження методів і технологій створення розподілених систем електронної архівації даних
(тема)

Виконав: студент 2 курсу, групи ПЗСм-18-1
Фалатюк Г.О
(прізвище, ініціали)

спеціальності 121- Інженерія програмного забезпечення
(код і повна назва спеціальності)

Освітньо-професійної програми
(тип програми)

Програмне забезпечення систем
(повна назва освітньої програми)

Керівник проф. Дудар З.В.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри, проф. _____ З.В. Дудар

20__ р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
Кафедра Програмної інженерії
Рівень вищої освіти - другий (магістерський)

Спеціальність 121-Інженерія програмного забезпечення
(код і повна назва)
Тип програми освітньо-професійна програма

Освітня програма Програмне забезпечення систем

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Фалатюку Геннадію Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів і технологій створення розподілених систем електронної архівації даних
затверджена наказом університету від "___" _____20 ___ р № _____
заповнюється вручну після отримання наказу
2. Термін подання студентом роботи до екзаменаційної комісії
17 грудня 2019 р.
3. Вихідні дані до роботи *вимоги до системи архівації та довготривалого зберігання електронних даних, атрибути якості програмного забезпечення, пояснювальна записка, платформа розробки .NET, середовище розробки Microsoft Visual Studio 2019 Community Edition, система контролю версій GIT, система управління розробкою проектів Azure Dev Ops та хмарна платформа Azure.*
4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз проблемної галузі і постановка задачі, огляд методів та стандартів довготривалого збереження електронних даних, огляд атрибутів якості програмного забезпечення, огляд архітектурних стилів та шаблонів проектування, методи тестування програмного забезпечення на відповідність нефункціональним вимогам.*

Наступний аркуш завдання

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка *
1.	Аналіз предметної галузі та постановка задачі	17 вересня 2019р.	
2.	Аналіз та формування вимог до програмної системи	27 вересня 2019р.	
3.	Дослідження атрибутів якості програмного забезпечення	04 жовтня 2019р.	
4.	Дослідження впливу архітектурних стилів, шаблонів проектування та технологій на атрибути якості програмного забезпечення	18 жовтня 2019р.	
5.	Проектування архітектури системи	25 жовтня 2019р.	
6.	Розробка системи	13 листопада 2019р.	
7.	Розробка плану тестування програмного забезпечення на відповідність функціональним та нефункціональним вимогам	18 листопада 2019р.	
8.	Розгортання і тестування системи	25 листопада 2019р.	
9.	Обробка результатів тестування	28 листопада 2019р.	
10.	Підготовка пояснювальної записки	05 грудня 2019р.	
11.	Підготовка презентації та доповіді	07 грудня 2019р.	
12.	Попередній захист	09 грудня 2019р.	
13.	Нормоконтроль, рецензування	11 грудня 2019р.	
14.	Занесення диплома в електронний архів	13 грудня 2019р.	
15.	Допуск до захисту у зав. Кафедри	15 грудня 2019р.	
* заповнюється вручну після виконання чергового пункту			

Дата видачі завдання 02 вересня 2019 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Дудар З.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить 86 сторінок, 13 рисунки, 5 додатків, 5 таблиць, 20 джерел.

ЕЛЕКТРОННЕ АРХІВУВАННЯ, РОЗПОДІЛЕНІ СИСТЕМИ, ASP NET CORE, ELASTICSEARCH, WEB, TRANSACTIONS, EVENTUAL CONSISTENCY.

Об’єктом дослідження є сучасні методи та технології реалізації розподілених систем електронної архівації даних.

Метою роботи є розробка прототипу системи електронної архівації даних та дослідження впливу різних архітектурних стилів програмного забезпечення та технологій розробки і розгортання на продуктивність даної системи.

Методи розробки базуються на інструментах для створення веб-додатків на платформі .NET.

В результаті роботи було створено прототип систем електронної архівації даних та обрані архітектурний стиль, шаблони проектування та технологій, що забезпечують досягнення необхідних атрибутів якості системи.

E-ARCHIVING, DISTRIBUTED SYSTEMS, ASP NET CORE, ELASTICSEARCH, WEB, TRANSACTIONS, EVENTUAL CONSISTENCY

The object of study is modern methods and tools for construction of distributed e-archive systems.

The goal of the work is to develop the prototype of the e-archive system and investigate the impact of different software architecture styles and technologies for application development and deployment on the overall performance of the system.

The development methods are based on the tools for development of web-applications on the .NET platform.

The result of the work is the prototype of the e-archive system that conforms to the defined system quality attributes through the combination of certain architecture styles, design patterns and technologies.

ЗМІСТ

ВСТУП

Активне впровадження електронних документів в сферу громадського використання спричиняє за собою вирішення задач довготривалого зберігання інформації. Проблема неухильного зростання обсягів паперових документів актуальна для всіх підприємств, організацій та установ, незалежно від роду їх діяльності, форми власності та галузевої приналежності. Підкреслимо, що сучасний період розвитку інформаційної сфери суспільства характеризується стрімким зростанням обсягів електронних документів. Інтернет змінив звичні методи отримання інформації, ініціював появу нових засобів доступу людей до знань. Все більша частка припадає на інформацію, яка народжується, існує, циркулює, зберігається тільки в електронному вигляді. Особливо це стосується наукової інформації. Електронна форма уможливлює сьогодні більш компактне зберігання інформації, її оперативне та широке розповсюдження і, крім того, дозволяє зручно маніпулювати нею [1].

На різних етапах свого розвитку архіви комплектувалися спочатку паперовими документами, потім з’явилася можливість отримувати аудіо- та відео документи. В останній час стало можливим формувати фонд електронними документами, що значно підвищує значення архівів. Використання електронних документів дає позитивний ефект, за рахунок більш оперативного обслуговування, можливості багатократного копіювання документу без втрати експлуатаційних властивостей, можливості пошуку за ключовими словами, меншої трудомісткості редагування, тиражування перекладу.

На разі питанням зберігання електронної інформації приділяється багато уваги, особливо на міжнародних конференціях в яких беруть участь представники бібліотек, архівів, видавництв та компанії виробники баз даних та інформаційних продуктів. Однією з проблем при використанні електронних документів є те, що вони значно більше схильні небезпеці викривлення та руйнування ніж інформація на традиційних носіях. Вони можуть бути знищені не тільки в результаті необережного поводження з ними персоналу чи вірусної атаки але й в результаті технічної несправності обладнання. Друга не менш серйозна проблема зберігання

електронної інформації – це старіння комп’ютерів та програмного забезпечення, що зобов’язує власників електронних документів переводити їх в нові формати та перезаписувати на нові носії інформації, тобто проводити міграцію [2].

Створення електронного архіву спрямовано на вирішення наступних завдань:

– автоматизації й оптимізації системи надійного збереження всіх типів електронних документів;
– підтримки різних версій документів;
– автоматичного контролю термінів зберігання документів і перенесення їх на різні фізичні носії інформації;
– ефективної системи пошуку документів за реквізитами та змістом;
– розмежування рівнів доступу для різних категорій користувачів системи;
– внутрішнього кодування документів;
– відстеження інформаційних запитів користувачів;
– централізації адміністрування й управління всією системою загалом;

У цих умовах досить високим є ризик втрати даних, на створення чи придбання яких іноді витрачаються значні зусилля і кошти. Тому надзвичайної актуальності набуває питання довготривалого збереження цифрових ресурсів.

Створення програмних систем, особливо таких комплексних, як електронні архіви, потребує врахування великої кількості факторів починаючи з аналізу функціональних і нефункціональних вимог, розробки архітектурного стилю, вибору шаблонів проектування і технологій і закінчуючи стратегією розгортання системи.

Метою даної атестаційної роботи є проектування архітектури системи електронного архіву, яка задовольнятиме певні атрибути якості програмного забезпечення, та визначити вплив архітектурних стилів, шаблонів проектування, технологій і стратегії розробки і розгортання на ці атрибути.

Об’єктом дослідження є процес електронної архівації цифрових даних.

Предметом дослідження є архітектурні стилі, шаблони проектування, технології і стратегії розробки і розгортання програмного забезпечення, які можуть бути використані для створення системи електронного архіву.

Під час виконання атестаційної роботи були використані наступні методи дослідження:

- загально логічні – аналізі предметної галузі та виявлення проблем було здійснено за допомогою аналізу, синтезу, індукції, абстрагування, та узагальнення. Перед виконанням тестів з навантаження системи було проведено моделювання очікуваної поведінки;
- теоретичні – під час аналізу вимог до програмного забезпечення та проектування архітектури системи було застосовано метод сходження від абстрактного до конкретного;
- емпіричні – було проведено експерименти з навантаження системи для обробки різних об’ємів даних, виконані заміри характеристик системи при виконанні тестів та порівняні отримані результати;

У результаті виконання атестаційної роботи було обрано атрибути якості, якими повинна володіти система електронного архіву, обрано і встановлено кількісний вплив архітектурних стилів та шаблонів проектування на ці атрибути якості. Також визначено технологічний стек і стратегію розгортання, які можуть бути використані для створення системи електронного архіву, та розрахована її вартість.

Отримані результати можуть бути використані для проектування нових систем електронних архівів, а також для визначення напряму, релевантності і вартості зміни архітектури і технологій існуючих архівних систем.

Результати роботи опубліковані у статі «Investigation of Architecture and Technology Stack for e-Archive System» на міжнародній науковій IEEE конференції «Problems of Infocommunications. Science and Technology (PIC S&T′2019)».

1 АНАЛІЗ ПРОБЛЕМНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Забезпечення надійного сховища для великих обсягів цифрових даних є надзвичайно важливим завданням в сучасному світі, оскільки кількість цих даних постійно зростає. Світовий об'єм цифрових даних налічує понад 8 зеттабайт і буде подвоюватися щороку [3].

Довготривале зберігання профільних документів в умовах постійної зміни програмно-технічних засобів передбачає планову міграцію цих документів з існуючого донині покоління програмно-технічних засобів обчислювальної системи в наступне. Технологія довготривалого та постійного зберігання має задовольняти вимоги збереженості (цілісність, автентичність) та постійної визначеності електронних документів (доступ із візуалізацією та можливістю відтворення на папері).

З проблемою довгострокового зберігання даних наприкінці дев’яностих початку двохтисячних років зіткнувся Міжнародний Консультативний Комітет з космічних систем передачі даних (CCSDS – Consultative Committee for Space Data Systems), коли потрібно було заархівувати дані космічних досліджень за останні п’ятдесят років. Задля вирішення цієї проблеми було створено модель OAIS (Open Archival Information Model).

OAIS є міжнародним стандартом цифрового збереження. Він описує стандартну архітектуру і функціональність цифрового архіву. Концептуальна модель OAIS складається з наступних частин [4]:

а) словник для розмов про спільні операцій, послуги та інформаційні структури сховища;

б) модель даних для інформації, яку електронний архів приймає, управляє всередині і надає іншим. Ця модель представлена трьома типами пакетів:

1) SIP (Submission Information Package) – інформаційний пакет, який використовується для прийому даних у сховище,

2) AIP (Archival Information Package) інформаційний пакети, які використовується для внутрішнього зберігання і управління в сховищі;
3) DIP (Dissemination Information Package) – інформаційні пакети, які використовуються для експорту споживачам.

в) обов'язкові відповідальності, які сховище зобов'язується виконувати. Ці обов'язки мають на увазі узгодження з виробниками і споживачами інформаційних пакетів на предмет повноти і зрозумілості архівування інформації та дотримання чітко визначених і добре документованих процедур для отримання, збереження, аутентифікації і надання цієї інформації;

г) рекомендований набір функцій для виконання необхідних обов'язків сховища. Ці функції розбиті на шість функціональних модулів:
1) «Ingest»;
2) «Data Management»;
3) «Archival Storage»;
4) «Access»;
5) «Administration»;
6) «Preservation Planning».

На рисунку 1 зображені основні потоки інформації та набір функцій з яких складається архівна система. OAIS виділяє три ролі користувачів, які є зовнішніми до системи, і з якими електронний архів взаємодіє:

– Producer – люди або системи, які виробляють дані і передають дані для збереження;
– Consumer – люди або системи, які використовують послуги з пошуку і отриманню архівних даних, які надає архівна система. Згідно зі стандартом є конкретна група користувачів під назвою «Designated Community» – група користувачів, які повинні вміти розуміти архівні дані;
– Management – організація що виступає в ролі постачальника архівних послуг, вона визначає стратегію архіву, цілі і завдання.

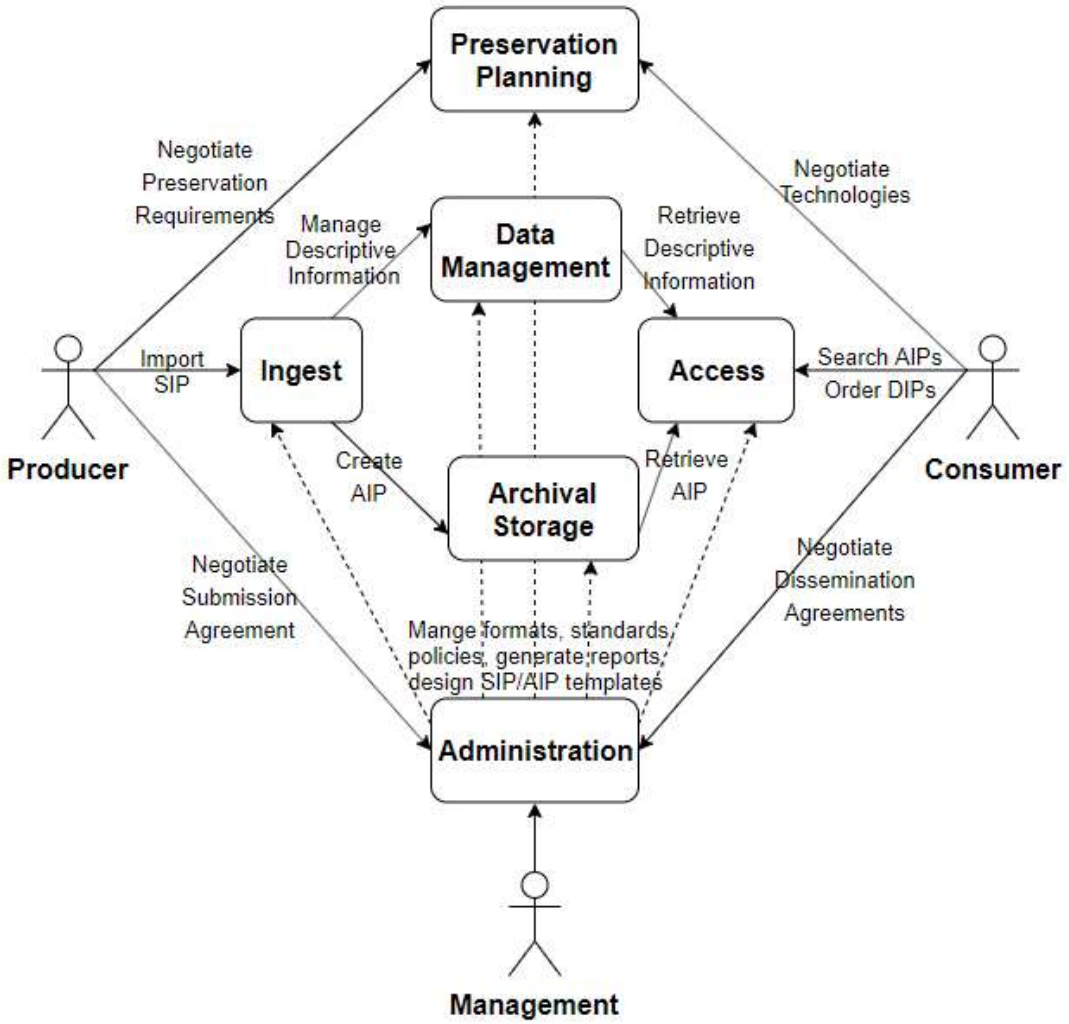


Рисунок 1 – Функціональна модель OAIS

Далі розглянемо більш детально кожну за функціональних сутностей архівної системи та визначимо яку роль вони відіграють у системі електронного архіву.

Функціональний модуль «Ingest» надає послуги та функції для прийому інформаційних пакетів (SIP) від Producer(s) (або з внутрішніх елементів з функціонального модулю «Administration») і підготувати вміст для зберігання та управління в архіві. До функції даного модулю входять:

– отримання інформаційних пакетів (SIP);
– перевірка якості даних, що знаходяться в пакетах;
– створення інформаційного пакету для внутрішнього зберігання і управління (AIP);

- вилучення або створення описової інформації пакету, використовуючи PDI (Preservation Descriptive Information) пакету, для подальшого її включення до бази даних архіву;
- координація оновлень сутностей з «Archival Storage» та «Data Management» модулів.

Процес передачі даних перетворює SIP(s), отримані в сесії імпорту даних, у набір AIP(s) та дескрипторів пакетів, які можуть бути збережені та прийняті функціональними об'єктами з модулів «Archival Storage» та «Data Management».

Коли для створення одного AIP потрібно багато SIP(s), функціональна область «Ingest» забезпечить тимчасове зберігання для SIP(s) до тих пір, поки не надійдуть усі SIP(s), необхідні для AIP.

Функціональний модуль «Data Management» надає послуги та функції для наповнення, підтримки та доступ до описової інформації, яка ідентифікує та описує дані, що зберігаються в архівних пакетах, та адміністративні дані, що використовуються для управління архівом. До функції даного модулю входять:

- адміністрування функцій бази даних архіву (підтримка схеми, конфігурацію відображень та цілісність посилань);
- виконання оновлень бази даних архіву (завантаження нових описових даних пакетів або адміністративних даних архіву);
- виконання запитів щодо даних управління архівом задля генерації відповідей на запити з «Access» модулю та створювати звіти стосовно цих запитів.

Функціональний модуль «Archival Storage» надає послуги та функції для зберігання, підтримки та вилучення архівних пакетів (AIP). До функції даного модулю входять:

- отримання AIP від «Ingest» модуля та додавання їх до постійного сховища;
- управління ієрархією сховища;
- оновлення засобів масової інформації, на яких безпосередньо зберігаються архівні пакети;
- проведення рутинної та спеціальної перевірки помилок;

- забезпечення можливостей відновлення після аварій;
- постачання архівних пакетів для «Ingest» модуля для виконання запитів на доступ до інформації що зберігається в архіві.

Функціональний модуль «Access» надає послуги та функції для що необхідні Consumers для встановлення існування, описової інформації, місцезнаходження та доступності інформації, що зберігається в архівній системі. Функції даного модуля дозволяють Consumers формувати пошукові запити та отримувати інформацію про архівні пакети. До функції даного модулю входять:

- спілкування з Consumers для отримання запитів;
- застосування елементів керування для обмеження доступу до спеціально захищеної інформації;
- координація виконання запитів до їх успішного завершення;
- генерація відповідей на запити у вигляді пакетів для розповсюдження (DIP);
- інформування Consumers щодо стану їх запитів.

Функціональний модуль «Administration» відповідає за підтримку злагодженої роботи усіх функціональних модулей архівної системи. До функції даного модулю входять:

- забезпечення інших функціональних модулів послугами та функціями необхідними для здійснення загальних операцій в системі.
- отримання та узгодження договорів прийняття інформаційних пакетів SIP(s) від Producers:
- аудит пакетів, що надходять до системи, на предмет відповідності встановленим стандартам архівування в системі;
- підтримка управління конфігурацією апаратного та програмного забезпечення системи;
- забезпечення контролю та вдосконалення операцій у архівному сховищі;
- інвентаризація, звітування про та переміщення / оновлення вмісту архівного сховища;

– встановлення та підтримка архівних стандартів та політик, надання підтримки клієнтам та активації збережених запитів;
– надання підтримки клієнтам та активації запитів;
– аудит системних операцій, продуктивності системи та використання системи;

Функціональний модуль «Preservation Planning» надає послуги та функції для моніторингу навколишнього середовища OAIS, забезпечує рекомендації та плани зберігання інформації для забезпечення її залишається доступності і можливості буди одностайно зрозумілою представниками «Designated Community» протягом тривалого часу, навіть якщо оригінальне обчислювальним середовище стає застарілим протягом цього інтервалу. До функції даного модулю входять:

– оцінювання вмісту архівного сховища та періодична рекомендація оновлення архівної інформації;
– рекомендація міграції поточних архівів;
– розробка рекомендацій щодо стандартів та політики архіву;
– надання періодичних звітів про аналіз ризиків;
– моніторинг змін у технологічному середовищі та вимогах щодо обслуговування та базі знань «Designated Community».

Даний модуль розробляє шаблони інформаційних пакетів та надає допомогу в їх розробці та ревізії, щоб спеціалізувати ці шаблони для конкретних постачань SIP(s) та AIP(s). Також цей модуль відповідає за розробку детальних планів міграції, прототипів програмного забезпечення та тестових планів для забезпечення реалізації міграційних цілей модулю «Administration».

2.2 Виявлення проблем та актуалізація рішень

Модель OAIS забезпечує загальні поняття, словниковий запас, моделі даних, обов'язки, які повинен виконувати електронний архів, та встановлює сукупність рекомендованих функції для покриття цих обов'язків. Проте електронний архів в OAIS – це не просто програмне застосування, а скоріше організація, яка складається з багатьох компонентів (включаючи різні компанії, реальних людей,

будівлі, обчислювальну техніку, програмні застосування, фінансові і юридичні операції, тощо) між якими існують чітко налагоджені зв'язки і процеси, які дозволяють їм функціонувати як єдиний електронний архів. Однак, програмне забезпечення в електронному архіві є одним з найважливіших компонентів без якого його існування не є можливим. Зважаючи на це, постає актуальним завдання проектування і створення програмної системи, яка задовольнятиме усі функціональні і нефункціональні вимоги і може бути використаним у якості провідного програмного забезпечення в електронному архіві.

При створенні будь-якого програмного забезпечення можна виділити два етапи, які мають максимальний вплив на якість результуючого продукту:

– аналіз предметної галузі, результатом якого є сформовані функціональні і нефункціональні вимоги до додатку;
– проектування архітектури системи.

Помилки допущені на цих етапах є найбільш дорогими для виправлення і можуть призвести до згортання продукту. Якщо дослідження предметної галузі та аналіз вимог до програмної системи електронного архіву загалом базується на моделі OAIS, то питання проектування архітектури системи (вибір технологій, форматів даних, протоколів зв'язку, проектування внутрішнього та зовнішнього API, розробка стратегії розгортання та обслуговування системи) є відкритим. Зважаючи на різноманіття технологій розробки і розгортання програмних систем, архітектурних стилів і шаблонів, постає актуальним завдання правильного вибору комбінації технологічного стеку, архітектури і шаблонів проектування, які задовольнятимуть функціональні і нефункціональні вимоги програмної системи електронного архіву.

Архітектура – це фундамент програми, і це перше (після аналізу вимог), з чого слід починати побудову системи. Враховуючи функціональні та нефункціональні вимоги, головна мета архітектури програмного забезпечення – визначити, з яких компонентів система має складатися, як ці компоненти збираються спілкуватися один з одним і як їх потрібно розгорнути для виконання

цих вимог. Це найважча і найважливіша частина побудови додатків, адже всі помилки, допущені на цьому етапі, найдорожче виправити в майбутньому.

Наразі, стрімкий розвиток хмарних технологій змінює спосіб розробки програм. Замість монолітів, додатки розкладаються на менші, децентралізовані сервіси. Ці сервіси спілкуються через API або за допомогою асинхронних повідомлень чи реагують на певні події. Програми масштабуються горизонтально, за рахунок додавання нових вузлів та створення кластерів по мірі попиту ресурсів.

Ці тенденції ведуть до нових викликів:

– стан програми стає розподіленим;
– операції виконуються паралельно і асинхронно;
– в цілому система повинна бути стійкою до збоїв;
– розгортання повинно бути автоматизованим і передбачуваним;
– моніторинг та телеметрія мають вирішальне значення для розуміння внутрішнього стану системи.

При розробці сучасних веб-орієнтованих систем, перше, що треба обрати – це архітектурний стиль, у якому буде побудована майбутня система. Архітектурний стиль визначає «форму» системи, він створює певні обмеження на організацію взаємодії та дизайн компонентів, які у кожному конкретному випадку мають свої переваги та недоліки.

Фахівці з розробки програмних продуктів Майкрософт виділяють шість основних архітектурних стилів [5]:

– N-tier – це традиційна архітектура для корпоративних програм. Залежності управляються діленням додатку на шари, які виконують логічні функції, такі як презентація, бізнес-логіка та доступ до даних. Кожен шар може викликати лише шари, які знаходяться нижче за ієрархією. Внесення змін в одну частину програми, не торкаючись решти є досить важким. Це обмежує швидкість оновлень і додавання нових функцій.
– Web-Queue-Worker – ця архітектура передбачає тонкий веб-шар, основною метою якого є перевірка запитів користувачів та видача команд для тривалих процесорних завдань, що входять до черги, і worker-шар,

який слухає цю чергу для вхідних завдань і обробляє їх асинхронно. Така архітектура підходить для доменів з довготривалими операціями, що потребують багато ресурсів.

- Microservices – у цьому випадку додаток складається з безлічі невеликих незалежних служб, кожен з яких реалізує можливості єдиного бізнесу. Мікросервіси слабко пов'язані і спілкуються один з одним за допомогою строго визначеного API. Ця архітектура підходить для складних доменів.
- CQRS – підхід, який передбачає різні моделі даних для читання та запису. Ця архітектура дозволяє незалежно масштабувати навантаження для читання та запису.
- Event-driven – архітектура, що реалізує модель зв’язку «Publish-Subscribe», де виробники публікують події, а споживачі підписуються на них. І виробники, і споживачі незалежні один від одного.
- Big Compute – стиль архітектури, який займається високоефективними обчисленнями на великих обсягах даних за допомогою поділу цих даних на групи та проведення їх паралельної обробки.

Кожен за перелічених архітектурних стилів призначений для забезпечення певних атрибутів якості системи. Щоб обрати необхідний архітектурний стиль, спершу треба визначитися з атрибутами якості, якими має володіти система.

Атрибути якості – це загальні фактори, що впливають на поведінку під час виконання, на проектування системи, і на досвід користувачів. Вони представляють проблемні сфери, які мають значний вплив на систему через усі її шари та вузли. Деякі з цих атрибутів пов'язані між собою до загального дизайну системи, тоді як інші специфічні для часу виконання, часу проектування або проблем, орієнтованих на користувачів. Відповідність програми бажаним атрибутам якості, вказує на успіх дизайну та загальну якість програмного забезпечення.

Усі атрибути якості системи поділяються на чотири групи [6]:

a) Design Qualities – атрибути якості дизайну системи:

1) Conceptual Integrity – визначає послідовність та узгодженість загальної конструкції. Сюди відносять дизайн компонентів та модулів системи, стиль кодування та найменування змінних;
2) Maintainability – це здатність вносити зміни до системи з певним ступенем легкості. Ці зміни можуть вплинути на компоненти, послуги, функції та інтерфейси при додаванні змін функціональності, виправлення помилок або відповідність новим вимогам бізнесу;
3) Reusability – визначає можливість компонентів та модулів бути придатними для повторного використання в інших програмах або в інших сценаріях системи. Повторність використання мінімізує дублювання компоненти, а також час реалізації;

б) Run-time Qualities – атрибути якості, що проявляються під час виконання системи:
1) Availability – визначає пропорцію часу в системі, коли вона функціонує та робить. Її можна виміряти у відсотках від загального часу, коли система не працювала, за попередньо визначений період. На доступність впливають системні помилки, інфраструктура, проблеми, шкідливі атаки та навантаження системи;
2) Interoperability – здатність системи або різних систем успішно працювати разом, спілкуючись та обмінюючись інформацією з іншими зовнішніми системами, написаними та керованими зовнішніми сторонами;
3) Manageability – визначає, наскільки легким є управління додатком для системних адміністраторі, як правило, через достатні і корисні прилади, що підлягають застосуванню для моніторингу систем та для налагодження і налаштування продуктивності;
4) Performance – є показником швидкості відповіді системи на запити користувача та виконання будь-якої дії протягом заданого інтервалу часу. Вона може бути виміряна за затримкою (час, необхідний для

відповіді на будь-яку подію) або пропускною здатністю (кількість подій, що відбуваються протягом певного інтервалу часу);

5) Reliability – це здатність системи залишатися функціонуючою протягом часу. Надійність вимірюється як ймовірність того, що система не буде виконувати призначені функції протягом визначеного проміжку часу;
6) Scalability – це здатність системи обробляти збільшення навантаження, не впливаючи на загальну продуктивність, або здатність легко збільшуватися;
7) Security – це здатність системи запобігати зловмисникам або випадковим діям за межами призначеного використання, а також запобігати розголошенню чи втраті інформації. Безпечна система спрямований на захист активів та запобігання несанкціонованому внесенню змін інформації;

в) System Qualities – загальні атрибути якості системи:
   1) Supportability – це здатність системи надавати корисну інформацію для виявлення та вирішення проблем, коли не вдалося правильно відпрацювати;
   2) Testability – це міра того, наскільки легко створити критерії випробувань для системи та її компонентів та виконати тести, щоб визначити відповідність цим критеріям. Висока здатність до тестування робить більш імовірним своєчасну та ефективну ізоляцію помилок в системі;

г) User Qualities – атрибути якості, що визначаються з точки зору користувачів:
   1) Usability – визначає, наскільки добре додаток відповідає вимогам користувача та споживача, будучи інтуїтивно зрозумілим, простим для локалізації та глобалізації, забезпечуючи зручний доступ для людей з обмеженими можливостями та спричиняючи позитивний загальний досвід роботи.

Розробляти системи, що відповідають усім атрибутам досить складно і дорого, тому необхідно проаналізувати компроміси між різними атрибутами якості, та обрати комбінацію, яка є найбільш цінною для розроблюваної системи.

У випадку будування архівної системи цільові атрибути можна відсортувати за пріоритетом та розбити на дві категорії:

а) глобальні атрибути, відповідність яким має бути врахована на ранніх етапах вискорівневого проектування системи:
   1) Reliability;
   2) Scalability;
   3) Performance;
   4) Security;
   5) Availability;
   6) Manageability;

б) локальні атрибути, відповідність яким досягається на етапі низькорівневого проектування і безпосередньої розробки системи:
   1) Maintainability;
   2) Supportability;
   3) Reusability;
   4) Testability;
   5) Usability;
   6) Interoperability;

Атрибути якості системи мають безпосередній вплив на таку важливу характеристику системи, як вартість. Вартість програмного продукту ділиться на дві категорії:

– вартість розробки системи – кількість часу помножена на вартість години роботи, яка необхідна для створення програмного продукту. Вона включає затрати на аналіз, проектування, розробку, тестування, документування і підтримку системи (усунення дефектів, додавання нової функціональності);

– вартість експлуатації системи – кількість грошей за певний інтервал часу, яку необхідно сплачувати за підтримку функціонування розгорнутої системи. Вона включає витрати на середовище хостингу та персонал підтримки;

Атрибути якості впливають на вартість наступним чином – їх досягнення збільшує вартість розробки системи, проте здебільшого спрямовано на зменшення вартості експлуатації.

Досягнення вищезазначених атрибутів якості забезпечується комбінацією кількох аспектів:

– обраним архітектурним стилем;
– шаблонами проектування, використаними під час дизайну та безпосередньої розробки програмного забезпечення;
– обраним стеком технології;
– обраною стратегією та засобами розгортання.

Наразі, на фоні тенденції проектів з відкритим похідним кодом в IT індустрії, з’являється безліч безкоштовних технологій для рішення одних й тих самих завдань. Так, існують досить велике різноманіття SQL баз даних, документ орієнтованих баз даних, графових баз даних, пошукових рушіїв, хмарних і розподілених сховищ, технологій для розгортання та моніторингу систем. Тому, коли справа доходить до проектування системи, постає досить важке питання вибору конкретних технологій, і для вирішення цього питання необхідне провести дослідження переваг та недоліків кожного з доступних рішень.

Результат цього аналізу свідчить про необхідність проектування архітектури системи, що задовольнятиме функціональні вимоги моделі OAIS та відповідатиме необхідним атрибутам якості.

2.3 Постановка задачі

В атестаційній роботі магістра потрібно спроектувати архітектуру та розробити прототип системи електронного архіву, який задовольнятиме функціональним вимогам OAIS та відповідатиме наступним атрибутам якості:

– Performance;
– Scalability;
– Reliability;
– Availability;
– Security.

Для досягнення вищезазначеної мети необхідно дослідити вплив наступних архітектурних стилів і шаблонів проектування на обрані атрибути якості системи електронного архіву:

а) архітектурні стилі:
    1) N-Tier;
    2) Web-Queue-Worker;
    3) Microservices;
    4) CQRS;
    5) Event-Driven;
б) шаблони проектування [7]:
    1) Compensating Transaction;
    2) Transaction Log Tailing;
    3) Idempotent Event Handlers;
    4) Saga;
    5) Circuit Breaker;
    6) Retry;
    7) Federated Identity;

Враховуючи вплив шаблонів проектування на вартість та складність розробки системи обрати їх відповідну комбінацію.

Дослідити переваги та недоліки наступних технологій:

а) бази даних:
    1) реляційні:
        – Azure SQL;
        – Managed Microsoft SQL Server Instances;
    2) нереляційні:

- Mongo DB;

б) сховища даних:

1) Azure Blob Storage;
2) Azure Files;

в) пошукові рушії:

1) Elasticsearch;

г) технології розгортання:

1) Virtual Machines;
2) Azure Container Instances.

Проаналізувати співвідношення продуктивності і вартості використання вищезазначених технологій та обрати найбільш доцільну комбінацію для системи електронного архіву.

2 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Задачею даної атестаційної роботи магістра є проектування архітектури архівної системи, що задовольнить функціональні вимоги OAIS та відповідатиме обраним атрибутам якості. Для проведення запланованих досліджень необхідно побудувати прототип архівної системи використовуючи різні архітектурні стилі, шаблони проектування і технології. Побудова будь-якої системи починається з аналізу і формалізацій функціональних і нефункціональних вимог до програмного продукту.

2.1 Функціональні вимоги

Архівна система має відповідати вимогам і забезпечувати набір функцій, визначений OAIS. Прототип архівної системи необхідний для проведення обраних дослідження повинен мати наступну функціональність:

а) наявність системи реєстрації, автентифікації і авторизації користувачів в системі у вигляді локальних акаунтів;

б) наявність гнучкої системи прав з підтримкою груп користувачів;

в) функціонал для «Producers» ролі:

1) завантаження пакетів у архів:

- можливість додавати декілька файлів у пакет;
- можливість вказати атрибути доступу необхідні для доступу до пакету;

2) узгоджувати правила завантаження пакетів у архів:

- вказувати дозволені формати файлів;
- вказувати алгоритм для розрахунку контрольної суми;

г) функціонал для «Consumers» ролі:

1) гнучкий пошук інформації:

- пошук за конкретними частинами опису пакета;
- повнотекстовий пошук за описом пакета;
- повнотекстовий пошук у контенті текстових форматів файлів;

2) конфігурація відображення результатів пошуку;

3) можливість скачати знайдений пакет;

4) можливість залишити запит на доступ до пакета, у разі недостатнього рівня доступу;

д) функціонал для «Management» ролі:

1) управляти правами доступу користувачів;

2) переглядати статистику по архівам:

- кількість пакетів в архіві;
- журнал операцій, що відбулися;

3) управляти налаштуваннями архіву:

- вказувати схему описових даних пакетів;
- встановлювати стандарти для описових даних зберігання пакетів;

4) виконувати модерацію запитів щодо доступу до пакетів;

5) встановлювати квоти на максимальний розмір даних в архіві для користувачів згідно з типу їх передплати;

6) отримувати рекомендації щодо модифікації системи:

- збільшення розміру архівного сховища;
- зміна конфігурації системи для отримання бажаної продуктивності певних функцій системи;
- попередження про необхідність міграції даних у інший формат;

е) наявність інтуїтивно-зрозумілого користувачам інтерфейсу.

2.2 Нефункціональні вимоги

Прототип архівної системи має відповідати наступним нефункціональним вимогам:

а) серверна частина повинна підтримувати виконання у середовищі під управлінням операційної системи Windows;

б) система повинна підтримувати RESTful API с можливістю написання клієнтів сторонніми особами;

в) система повинна бути захищена від поширених ризиків для безпеки веб-додатків [8]:

1) вставка інструкцій;
2) некоректна аутентифікація та управління сеансами;
3) міжсайтове виконання сценаріїв (XSS);
4) небезпечні прямі посилання на об’єкти;
5) витік критичних даних;
6) підробка міжсайтових запитів (CSRF);

г) система повинна мати наступні базові рівні продуктивності окремих функцій системи:

1) завантажувати до архівної системи 25 Гб даних, представленого 500 пакетів розміром по 50 Мб не довше ніж 10 хвилин;
2) підготовлювати до скачування 25 Гб даних, представленого 100 DIP пакетами, що містять 5 архівних пакетів розміром по 50 Мб кожен не довше ніж 15 хвилин;
3) виконувати пошук в архіві, що містить 1 000 000 пакетів не довше ніх 10 секунд;

д) система повинна бути легко і витрато ефективно масштабуєма для обробки тимчасового зростання навантаження;

е) система в жодному разі не повинна призводити до втрати даних користувачів: ані в разі виникнення помилки під час виконання операції, ані у разу часткового виходу системи з ладу;

ж) система повинна швидко відновлюватися до робочого стану у разі виникнення несправності якогось з компонентів;

з) система має бути відказостійкою: відмова однієї функціональності не повинна призводити до виходу з ладу усієї системи;

и) система повинна підтримувати часті оновлення;

к) система повинна буду легкою для модифікації.

3 ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ З УРАХУВАННЯМ АТРИБУТІВ ЯКОСТІ

Проектування архітектура програмного забезпечення зазвичай є складним та об’ємним, особливо якщо мова йде про веб-орієнтовані системи з насиченою предметною областю, через що неможливо розробити гарне рішення з нуля. Тому, перше, що слід вибрати при розробці програми, – це архітектурний стиль.

3.1 Вибір архітектурного стилю

Архітектурний стиль програмного забезпечення визначає високорівневу парадигму та принципи взаємодії компонентів, яких необхідно дотримуватися при розробці системи.

Враховуючи предметну область електронного архіву, у нас вже є вертикально розкладені функції (імпорт, управління даними, зберігання, адміністрування, пошук, експорт та планування збереженням) та моделі (SIP, AIP, DIP). Більше того, завантаженість кожної функціональності може різко відрізнятися, що призводить до необхідності незалежного масштабування кожної функції. Система електронного архіву повинна мати можливість обробляти великі обсяги даних за проектом, що також передбачає необхідність горизонтального масштабування. Як і будь-яке програмне забезпечення, система електронного архіву не може бути побудована одразу, тому вона буде будуватися поступово, що передбачає часті випуски оновлень.

Також завдяки низькій зв’язності функцій системи між собою, оновлення однієї функції можуть не впливати на інші, тому для мінімізації впливу на процесу оновлення слід мати можливість незалежно виконувати оновлення різних частин системи.

Беручи до уваги розглянуті раніше архітектурні стилі, створимо порівняльну таблицю відповідності цих архітектурних стилів до вимог системи електронного архіву. Результат порівняння наведено у Таблиці 1.

Таблиця 1 – Відповідність архітектурних стилів до вимог системи електронного архіву

E-Archive Requirement	Architecture styles					
	N-tier	Web-Queue-Worker	Microservices	CQRS	Event-Driven	Big Data
Independent functional scaling	-	+/-	+	-	+	+/-
Scaling out	+	+	+	+	+	+
Fault isolation	-	+/-	+	+/-	+	+/-
Gracefull degradation	+/-	+/-	+/-	+	-	-
Modular updates	-	-	+	-	+	+
Ease of modification	-	-	+	-	+	-
Loose coupling	-	+/-	+	-	+	+/-
Ease of Construction	+	+/-	-	+/-	-	-
Ease of Deployment	+	+/-	-	+/-	-	-

Кожен з вищезазначених стилів має свої переваги та недоліки, саме тому найбільш доречним є комбінація практик з усіх цих стилів за для досягнення бажаних характеристик системи. Проте домінуючий архітектурний стиль має бути один. Враховуючи результати порівняння архітектурних стилів, найбільш підходящим домінуючим стилем є Microservices, адже даний стиль дозволяє легко комбінувати різні практики на різних частинах системи незалежно. Оскільки переважна кількість операції в системі електронного архіву є довготривалими, доцільним є використання Event Driven архітектури, що передбачає асинхронне спілкування між мікропослугами через шину повідомлень замість прямих викликів.

Проте в обраній комбінації архітектурних стилів є один значний недолік – це велика початкова вартість і складність розробки системи. По-перше при проектуванні мікросервісної архітектури з нуля, існує досить висока ймовірність зробити це неправильно і замість переваг, що надає даний архітектурний стиль, отримати лише недоліки. Виділяють наступні ризики і найбільш популярні помилки, які допускають при розробці мікросервісної архітектури:

- надмірне залучення різних технологій, які насправді є непотрібними для вирішення поставленої задачі;
- неправильне виділення мікро сервісів і організація комунікації між ними;
- відсутність продуманої схеми розгортання і поставки системи кінцевому споживачу.

Насправді, вищезазначені ризики/помилки є спільними для розробки будь-якої архітектури, проте вартість цих помилок у мікро сервісній архітектури на порядок вища ніж у будь-якій іншій. Задля мінімізації ризику неправильного дизайну мікросервісної архітектури, доцільним є попереднє проектування і розробка багаторівневої (N-Tier) архітектури з подальшою її еволюцією у мікро-сервіси. Даний підхід має наступні переваги:

- проектування багаторівневої архітектури дозволяє побачити повну картину системи, побудувати усі зв’язки та взаємодії між компонентами, що є вкрай необхідним для її подальшого розбиття на мікросервіси;
- менша загальна складність системи, що зменшує початкову вартість розробки системи;
- менша кількість використаних технологій, що призводить до збільшення надійності системи, а також зменшує початкову вартість її розробки;
- можливість поступової еволюції відповідно до вимог бізнесу, що повність відповідає сучасній тенденції Agile розробки програмного забезпечення.

3.2 Проектування варіантів використання системи

Після вибору архітектурного стилю, наступним кроком є безпосередньо детальне проектування архітектури програмного забезпечення з розбиттям системи на компоненти, та організації потоків даних між ними.

Проте щоб зробити це правильно, спершу необхідно зробити проектування варіантів використання за допомогою уніфікованої мови моделювання UML [9]. На рисунку 2 зображена діаграма варіантів використання системи електронного архіву.

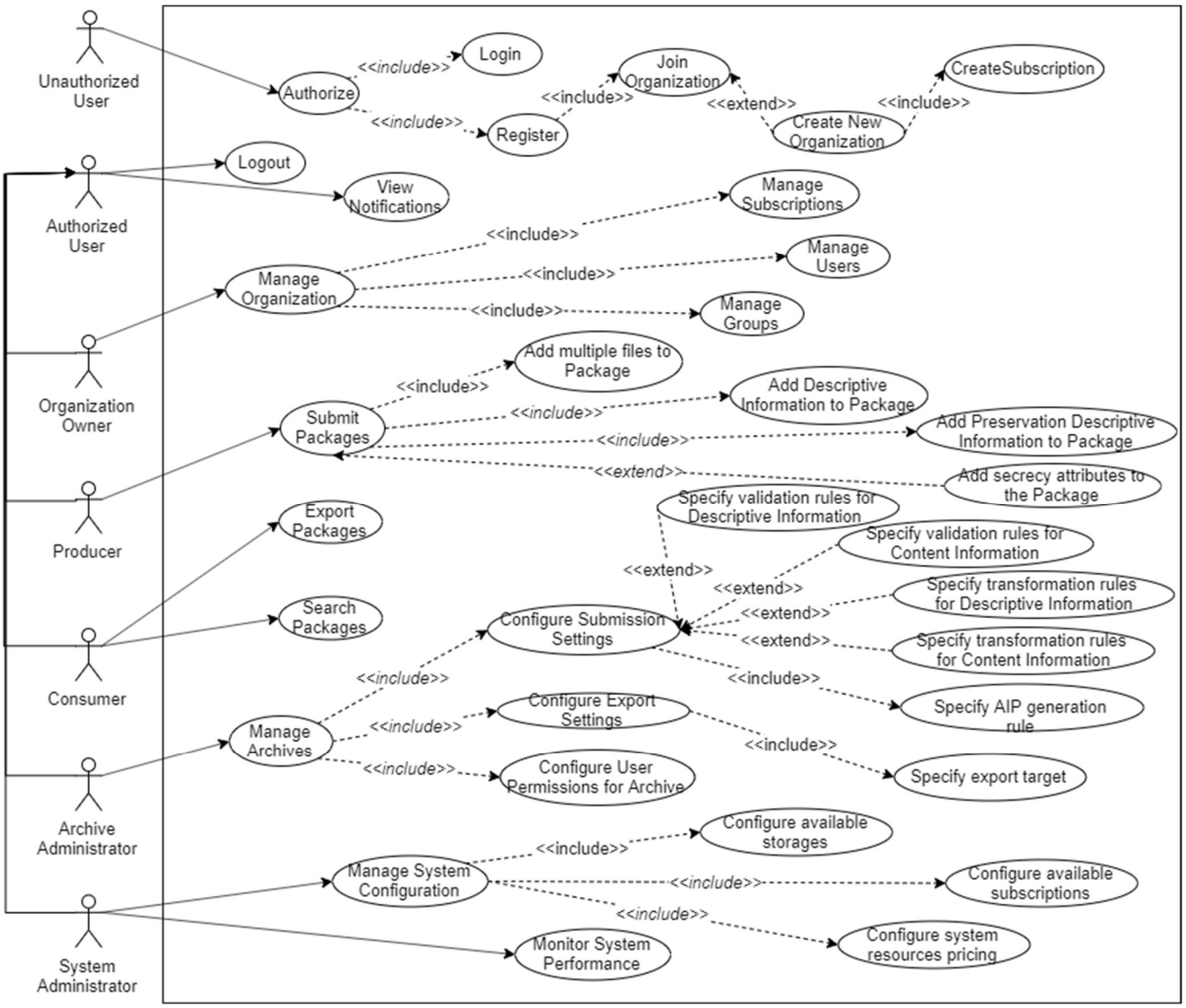


Рисунок 2 – Діаграма варіантів використання системи електронного архіву

Усі користувачі системи поділяються на дві групи:

– «Unauthorized User» – користувач, що має можливість лише переглядати інформацію про систему і не може виконувати ніякі інші функції окрім авторизації;
– «Authorized User» – користувач, що може виконувати функції в системі залежно від ролі і налаштувань доступу;

Як уже було зазначено, модель OAIS виділяє три зовнішні ролі, що взаємодіють з електронним архівом: «Producer», «Consumer» та «Management». Ці бізнес ролі, проектуються на п’ять типів користувачів усереднені системи:

- «Organization Owner» – користувач з правами власника організації може управляти налаштуваннями організації;
- «Producer» – користувач з правами на завантаження пакетів в архів;
- «Consumer» – користувач з правами на пошук та експорт пакетів з архіву;
- «Archive Administrator» – користувач з правами на створення та управління налаштуваннями архівів;
- «System Administrator» – користувач з правами на управління налаштуваннями усієї системи.

3.3 Предметно-орієнтоване проектування системи

Оскільки система електронного архіву має доволі складну предметну область, необхідно провести предметно-орієнтоване проектування (Domain-Driven Design) [10] для створення модульної архітектури, що якнайкраще відображає предметну область і, як наслідок, матиме легкою в модифікації, підтримці і розумінні кодову базу.

Предметно-орієнтоване проектування оперує наступними поняттями:

- домен – предметна область, середовище, галузь. Предметна область, яка використовується при створенні програмного забезпечення;
- модель – система абстракцій, яка описує окремі аспекти предметної області;
- загальна мова – мова побудована навколо моделі предметної області. Використовується як розробниками програмного забезпечення, так і експертами обраної галузі.
- контекст – середовище, в якому предмет або дія означує своє значення.

Зазвичай домен представляють у вигляді сполучення декількох взаємопов'язаних моделей. Дизайн програмного продукту являє собою сукупність принципів для підтримки цілісності моделі, постійний рефакторінг як засіб підтримки відповідності моделі предметній галузі, та поєднання декількох моделей в єдину схему.

Найвищим структурним компонентом предметно-орієнтованого проектування є обмежений контекст. Обмежений контекст представляє собою агрегацію логічно пов’язаних сутностей і взаємодії між. Загалом система складається з одно або декількох обмежених контекстів, які поєднуються у карту контекстів.

Кожен обмежений контекст має корінь агрегації – об’єкт, що містить у собі усі сутності агрегації, та логіку і правила взаємодії між ними.

Комунікація між різними обмеженими контекстами відбувається лише через корні агрегації цих контекстів.

Так, згідно з функціями зазначеними в моделі OAIS, а також враховуючи розроблені варіанти використання, можна виділити сім обмежених контекстів:

– «Identity» контекст – містить сутності і функції пов’язані з управлінням користувачами та групами;
– «Import» контекст – містить сутності і функції пов’язані з імпортом пакетів в систему;
– «Archive» контекст – містить сутності і функції пов’язані з довготривалим збереженням пакетів в системі;
– «Search» контекст – містить сутності і функції пов’язані з пошуком пакетів в системі;
– «Export» контекст – містить сутності і функції пов’язані з експортом пакетів в системі;
– «System» контекст – містить сутності і функції пов’язані з управлінням загальним станом системи;
– «Customer» контекст – містить сутності і функції пов’язані з управлінням рахунками користувачів;
– «Preservation Planning» – містить сутності і функції пов’язані з моніторингом функціонування архівів, стану даних і плануванням їх міграції.

Як було сказано раніше, існує три типи інформаційних пакетів: SIP, AIP, DIP. Кожен тип заснований на функції архівного процесу, який його використовує, а

також на перетвореннях між ними. На справді, будь-який інформаційний пакет складається з наступних частин:

– Content Information (CI) – сукупність інформації, яка є вихідною метою збереження електронним архівом.
– Preservation Descriptive Information (PDI) – описові дані збереження, це сукупність інформації, яка забезпечує довіру, доступ до та контекст CI протягом невизначеного періоду (тобто посилання, контекст, походження, фіксованість та інформація про права доступу).
– Descriptive Information (DI) – описова інформація, це сукупність інформації, яка описує CI і використовується для пошуку, аналізу, отримання або замовлення інформації з системи електронних архівів.

Різниця між цими видами пакунків полягає у повноті та форматі інформації. Таким чином, SIP зазвичай містить CI і частину PDI, AIP містить CI і повинен містити повний PDI і DI, DIP може містити частини CI, DI і PDI. Оскільки CI, DI та PDI – це абстрактні терміни, необхідно чітко визначити їх формати та представлення в поточній системі. Отже, в пакеті AIP CI буде представлений у вигляді колекції файлів будь-якого підтримуваного формату, DI – XML-файл, який відповідає схемі цільового архіву, а PDI – XML-файл із схемою XSD, визначеною системою [11].

3.4 Проектування багаторівневої архітектури

Багаторівнева (N-Tier) багатошарова (N-Layer) архітектура – це класична клієнт-серверна архітектура, в якій функції відображення, обробки і збереження даних поділяються на шари і розташовуються на різних вузлах [12]. Схема тришарової багаторівневої архітектури зображена на рисунку 3.

Ця архітектура передбачає наявність серверів, які надають інформацію або інші послуги програмам, які звертаються до них, та клієнтів, які використовують сервіси, що надаються серверами. Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів.

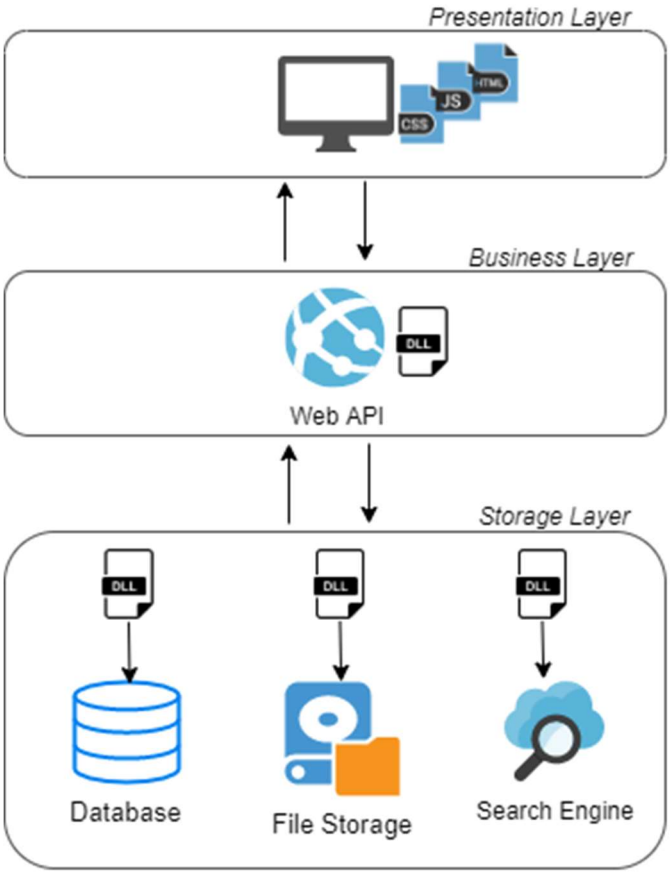


Рисунок 3 – Схема клієнт серверної тришарової архітектури системи електронного архіву

При даній архітектурі система розподіляється на різні рівні абстракції, що дозволяє вносити зміни на одних рівнях не втручаючись в роботу інших рівнів.

3.5 Проектування мікросервісної архітектури

Враховуючи типи пакетів та функції, які повинен надати електронний архів, набір компонентів, з яких система складається, зображені на рисунку 4:

а) Import Svc – сервіс, який відповідає за попередню обробку даних, що надходять для імпорту в архів (SIP). Це перший етап процесу створення AIP, який здійснює перетворення даних у формат, прийнятий цільовим архівом (наприклад, розділити один SIP на кілька AIP, розділити / перетворити DI, перетворити CI в інший формат файлу тощо).

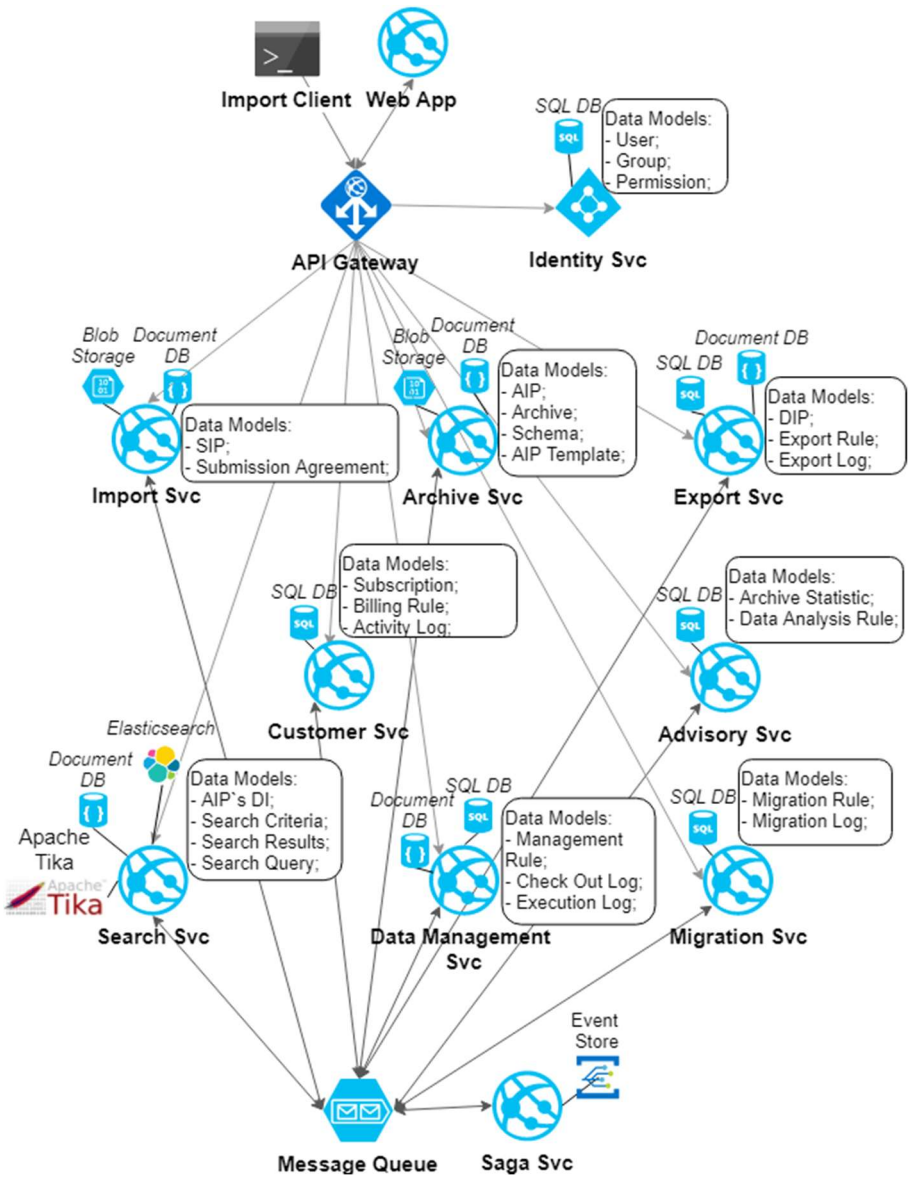


Рисунок 4 – Діаграма компонентів архітектури системи електронного архіву

б) Archive Svc – сервіс, який відповідає за фактичне зберігання архівних даних. Основними сутностями цієї служби є "Архів", вони є аналогами фактичного архіву з реального світу. Архів містить інформаційні пакети у вигляді AIPs. Кожен архів має свою схему у форматі XSD, яка визначає структуру DI AIP. Усі пакети, які імпортуються в систему, зберігаються в певному архіві. Даний сервіс надає набір API для додавання AIP, отримання інформації про AIP, пошуку та оновлення окремих частин AIP. Процес розміщення або оновлення AIP в архіві включає наступні етапи, які буде виконано цією службою:

1) перевірка даних – перевірка завантаженого DI AIPa відповідно до схеми архіву, в який цей AIP імпортується. Перевірка CI (тобто формати файлів, розміри файлів, вміст тощо);
2) розрахунок контрольної суми. Обчислення інформації зберігання для CI та DI;
3) технічне обслуговування PDI. Оновлення відповідних розділів у PDI при зміні CI та DI;
4) Технічне обслуговування DI. Використовуючи налаштований шаблон AIPa, якщо існує зв'язок між CI та DI оновити DI, коли CI змінюється;

в) Search Svc – сервіс, який відповідає за функціональність пошуку в системі електронного архіву. Він індексує DI кожного AIP, також, якщо він налаштований, виконує вилучення тексту з CI та індексує його. Крім того, ця послуга містить усю конфігурацію для пошукового запиту та появи результатів;

г) Data Management Svc – служба, відповідальна за ручне та автоматичне управління AIP. Ця послуга забезпечує таку функціональність, як конфігурація та запуск правил модифікації DI, PDI та CI AIP, перевірка AIP для редагування, візуалізація частин AIP для ручного редагування;

д) Export Svc – сервіс, який відповідає за експорт пакетів із системи електронного архіву. Ця служба перетворює AIP в DIP у форматі, прийнятному для цілей експорту;

е) Міграція Svc – служба, відповідальна за міграцію даних між архівами. Ця послуга виконує функцію планування збереження, яка передбачає безпечну міграцію даних з архіву з застарілим сховищем в архів із сучасним сховищем;

ж) Identity Svc – сервіс, який відповідає за управління користувачами. Ця послуга забезпечує аутентифікацію та авторизацію, налаштування користувачів та групи та управління дозволами. Найкращий спосіб зробити автентифікацію / авторизацію в мікро сервісній архітектурі – це

механізм аутентифікації на основі токенів [13]. Такий підхід передбачає окремий мікросервіс, який зберігає всю інформацію облікового запису, тому може аутентифікувати користувача. Коли користувач підтвердив свою особу (за допомогою пароля або іншим способом), ця служба видає маркер авторизації з усією інформацією про права користувача в системі. Потім цей маркер шифрується та надсилається клієнту. Коли користувач виконує будь-які захищені операції в системі, він надсилає цей маркер разом із запитом у відповідну службу. Кожен мікросервіс у системі здатен розшифрувати маркер і використовувати сховану в ньому інформацію для перевірки прав користувача на виконання запитуваних дій. Формат токена, який використовується для служби Identity, – веб-маркер JSON (JWT) з алгоритмом підписання HMACSHA256. Ця схема аутентифікації має наступні переваги [14]:

1) централізоване управління ідентичністю – користувачі, групи та дозволи контролюються одним мікро сервісом;
2) немає централізованої відмови в авторизації – коли користувач аутентифікован та отримав маркер, він може отримати доступ до будь-якого з мікросервісів у системі, доки не мине час активності маркера, навіть якщо служба Identity не працює;
3) немає затримок у мережі для авторизації – авторизаційний маркер містить всю інформацію про дозволи користувачів, тому запитуваний мікросервіс не повинен викликати Identity Svc для отримання додаткових відомостей;
4) підвищена безпека – токен зашифрований сильним криптографічним алгоритмом, що робить його неможливим для підробки.

з) API Gateway – це компонент розгортання, який забезпечує єдину точку входу API для всіх API внутрішніх мікросервісів;

и) Web App – це перша інтерактивна частина з інтерфейсом користувача в системі, яка забезпечує доступ до всіх функціональних можливостей, що постачаються за допомогою мікросервісів;

к) Import Client – це друга інтерактивна частина системи з інтерфейсом користувача, яка є настільною програмою, яка здійснює імпорт пакетів у систему;

л) Saga Svc – ця служба відповідає за підтримку цілісності даних системи під час виконання довготривалої операції;

м) Message Queue – це шина повідомлень, яка встановлює асинхронну модель зв'язку між мікросервісами.

3.6 Вибір стратегії впровадження системи

Впровадження програмного забезпечення включає у себе дії, що направлені на введення системи у експлуатацію. Процес впровадження залежить від способу розповсюдження програмного забезпечення. Дана програмна система підтримує два способи впровадження:

- SAAP (Software as a Product);
- SAAS (Software as a Service).

Впровадження першим способом складається з поставки кінцевим споживачам інсталяційного пакету і його подальше розгортання безпосередньо користувачем. Даний спосіб поставки спроектованої системи підійде організаціям, які мають власну IT інфраструктуру та/або мають обмеження на законодавчому рівні про місцезнаходження даних. Особливість даного методу впровадження є можливість обмежити використання програмного продукту для певної групи користувачів, шляхом розгортання системи у приватній мережі. Недоліком є необхідність самостійно підтримувати сервери, на яких розгорнуто систему електронного архіву.

Другий спосіб впровадження полягає у розгортанні архівної системи на серверах компанії-розробника цієї системи, в той час як кінцевий користувач регулярно вносить плату за користування цими послугами шляхом придбання

підписки або укладання прямого договору з компанією-постачальником. Постачальник, у свою чергу, надає користувачеві доступ до програми відповідно до узгоджених стандартів безпеки, доступності та продуктивності. Для доступу до програмного забезпечення все, що потрібно користувачеві, – це з'єднання з Інтернетом. Як результат, постачальник несе відповідальність за обслуговування апаратного забезпечення (яке, очевидно, повністю підпадає під юрисдикцію постачальника), оновлення, які розгортаються у всій системі, як тільки вони випускаються (і, як правило, є безкоштовними), та безпеку резервне копіювання даних. Даний спосіб розгортання є більш складним для компанії-постачальника, оскільки необхідно розробити і підтримувати певні Service Layer Agreement (SLA), і згідно з ними розробити цінову політику на доступ до сервісу електронної архівації даних.

4 РОЗРОБКА СИСТЕМИ З УРАХУВАННЯМ АТРИБУТІВ ЯКОСТІ

Враховуючи складність розробки спроектованої мікросервісної архітектури, доцільним є розробити багаторівневу тришарову архітектуру систему у якості першої версії продукту, та оцінити атрибути якості отриманої системи. Надалі проводити поступову еволюцію системи відповідно до вимог бізнесу. Дана стратегія, по-перше дозволить доволі швидко впровадити продукт на ринок, а по друге дозволить слідкувати за зміною показників якості програмного забезпечення при зміні архітектури на мікросервісну.

4.1 Організація структури проектів

Для безпосередньої внутрішньої реалізації програмних модулів та організації взаємодії між ними доцільним є використання «цибулевої» архітектури (Onion Architecture) [15], що зображена на рисунку 5. Її ключова особливість полягає в управлінні залежностями. Фундаментальне правило – будь-який модуль програми може залежати від ближчих до центру «цибулини» модулів, але не може залежати від більш далеких. Іншими словами, будь-яка зв'язаність повинна бути спрямована до центру «цибулини». «Цибулева» архітектура тяжіє до ООП і ставить об'єкти понад будь-яких інших аспектів програми. У самому центрі знаходиться доменна модель. Навколо доменної моделі знаходяться інші шари з додатковою бізнес логікою. Кількість шарів в додатку може змінюватися, але доменна модель повинна завжди залишатися в самому центрі і, оскільки будь-яка зв'язаність спрямована до центру, доменна модель пов'язана тільки сама з собою.

Перший шар навколо доменної моделі, це те місце де зазвичай розміщуються інтерфейси репозиторіїв, що забезпечують збереження і отримання об'єктів. Необхідно відзначити, що сам процес збереження об'єктів знаходиться поза ядра додатка, оскільки цей процес зазвичай включає в себе взаємодію з сервером баз даних, а в центрі ядра знаходяться тільки інтерфейси. На зовнішньому радіусі додатку знаходяться UI, інфраструктура і тести. Зовнішній шар зарезервований за елементами які змінюються часто. Ці елементи повинні бути ізольовані від ядра програми. На самому краю, знаходяться класи реалізують інтерфейси репозиторіїв.

Ці класи тісно пов'язані з конкретною технологією доступу до даних і саме тому повинні бути винесені за межі ядра програми.

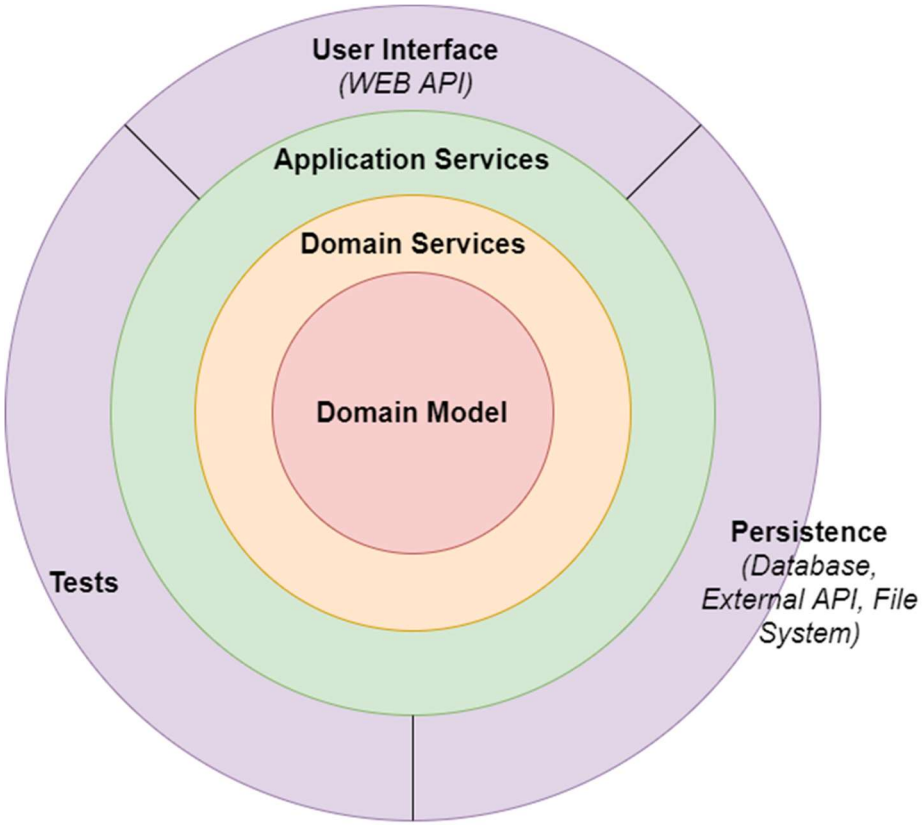


Рисунок 5 – Концептуальна схема «цибулевої» архітектури

«Цибулева» архітектура заснована на принципі інверсії залежності. Для роботи програмному додатку необхідні реалізації розташованих в проекті інтерфейсів, а оскільки конкретні реалізації знаходяться на зовнішньому радіусі додатку необхідний механізм для впровадження залежності. Дана архітектура забезпечує продуманий підхід до структуризації і візуалізації програмного забезпечення. Вона э простою в реалізації і забезпечує зрозумілий односторонній зв’язок між компонентами системи. Взаємозамінність та роздільність периферійних компонентів дозволяє змінювати шар збереження даних, шар представлення не змінюючи бізнес-логіку додатку.

Для написання коду програмного забезпечення використовується інтегрована середа розробки Visual Studio 2019. У даному середовищі розробки виділяють наступні об’єкти структурування файлів додатку:

- Solution – головний об’єкт, що містить повну інформацію про файли та проекти, які входять в склад додатку;
- Project – логічна одиниця, що представляє собою модуль додатку та містить файли з кодом та ресурси, які у результаті компілюються у виконуваний файл або бібліотеку.

Оскільки архітектура додатку побудована на основі «Цибулевої» моделі, то є доцільним організувати структуру файлів та проектів дотримуючись принципів обраної архітектурної моделі і згідно з обмеженими контекстами виділеними під час предметно-орієнтованого проектування. Отримана структура зображена на рисунку 6.

Рішення «EArchive» складається з наступних частин:

- директорія «Tests» містить проект інтеграційних тестів для Web API – «EA.Tests.Integration», а також проекти з тестами для основної бізнес-логіки додатку та інфраструктурної функціональності – «EA.Tests.Unit»;
- директорія «Presentation» містить проект, у якому зосереджено Web API та клієнтська логіка додатку – «EA.Web».
- проект «EA.Domain» містить файли з кодом, що реалізує частину бізнес логіки системи, яка безпосередньо пов’язана з доменними сутностями. Він являє собою проект бібліотеки класів та містить класи доменних моделей та сервісів для роботи з ними;
- проект «EA.Application» є проектом бібліотеки класів та містить код, що реалізує частину бізнес логіки, яка співвідноситься з варіантами використання системи і являє собою бізнес процеси.
- проект «EA.Persistence» є проектом бібліотеки класів та містить код, що реалізує шар доступу даних в системі. Тут знаходяться класи для роботи з базами даних, файловою системою та пошуковим рушієм.
- проект «EA.Common» є проектом бібліотеки класів та містить код, що є спільним для усіх шарів системи. Загалом тут знаходяться утилітарні класи та спільні моделі і інтерфейси.

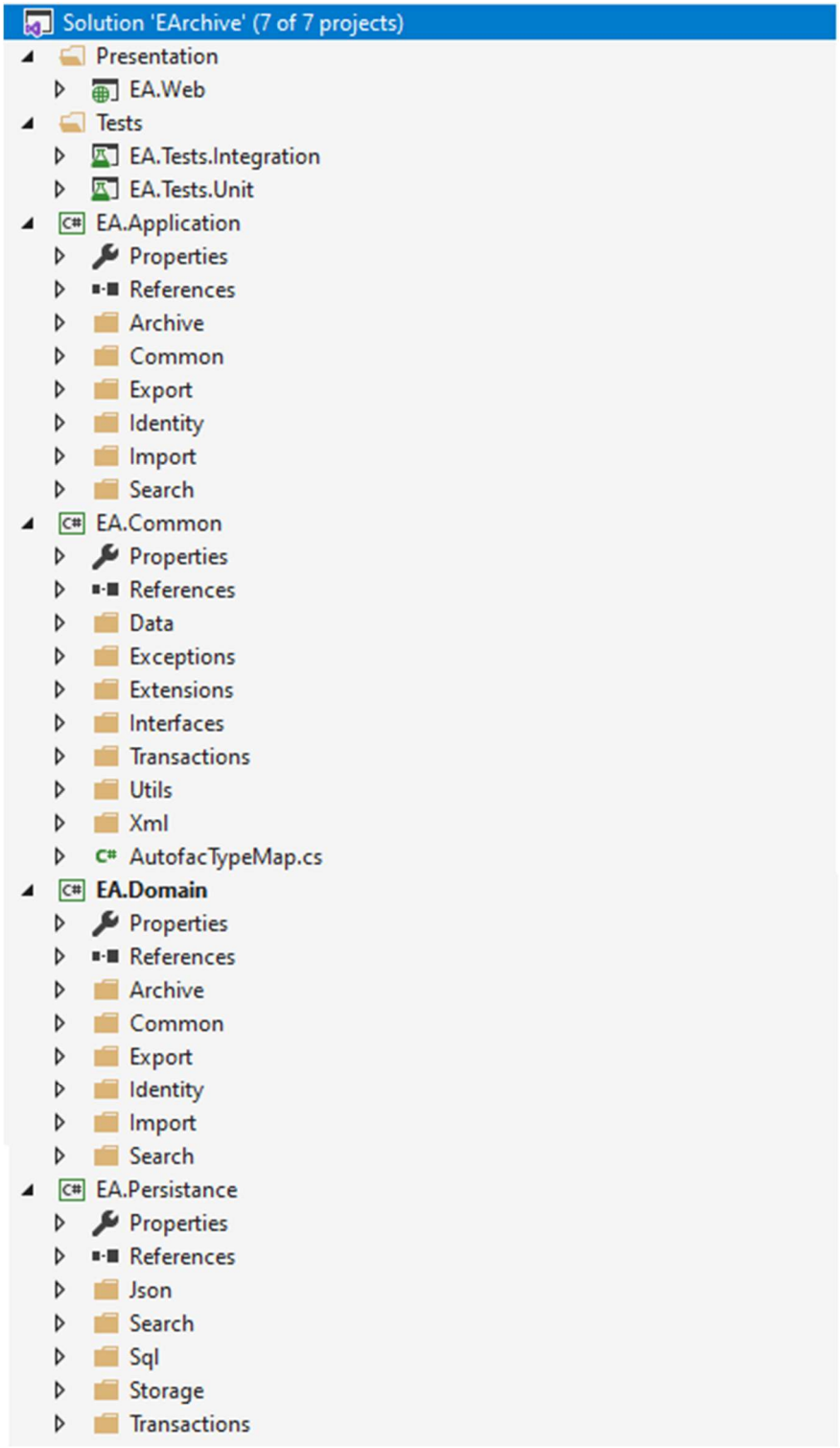


Рисунок 6 – Структура проектів і файлів системи електронного архіву

Загалом, структура кодової бази системи електронного архіву є поєднанням горизонтального розділення на рівні згідно з «Цибулиною» архітектурою і вертикального розділення на обмежені контексти згідно з предметно-орієнтованого проектуванням.

4.2 Вибір шаблонів проектування

Основна бізнес логіка проекту побудована з використанням підходу CQRS (Command Query Responsibility Separation) [16], схема якого зображена на рисунку 7. Основна ідея даного підходу полягає у розділенні моделі даних для читання від моделі даних для запису.

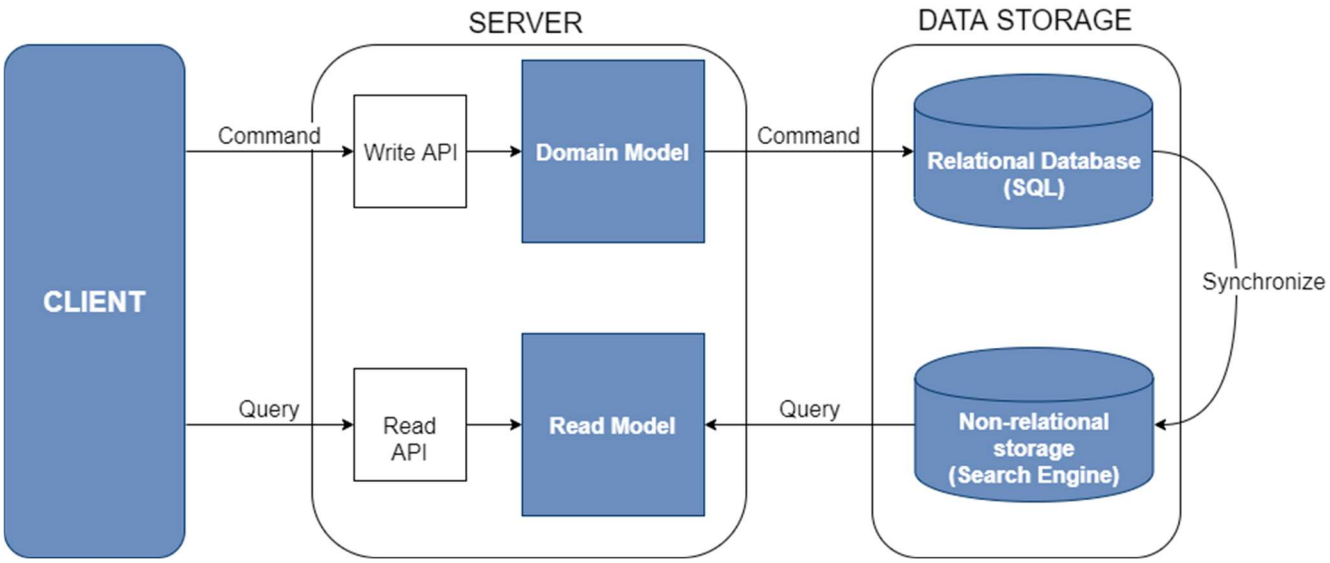


Рисунок 7 – Схема підходу CQRS

CQRS припускає наявність двох типів об’єктів: команди та запити. Команди використовуються з єдиною метою – виконати повну операцію щодо зміну стану сутності. Запити навпаки використовуються лише для читання і не повинні спричиняти будь-які побічні зміни стану системи.

Щоб відділити бізнес-логіку додатку від жорсткого зв’язування з конкретними технологіями збереження даних, було використано принцип «Dependency Inversion» і шаблон проектування «Repository» в сукупності з «Unit Of Work». Реалізація принципу «Dependency Inversion» зображена на рисунку 8.

Принцип «Dependency Inversion» полягає в тому, що залежності всередині системи будуються на основі абстракцій, що не повинні залежати від деталей, а навпаки, деталі мають залежати від абстракцій [17].

```
public interface IRepository<TEntity> where TEntity: class
{
    Task<TEntity> GetByIdAsync(int id);

    Task<List<TEntity>> ListAllAsync();

    Task<List<TEntity>> ListAsync(ISpecification<TEntity> specification);

    void Add(TEntity entity);

    void Update(TEntity entity);

    void Delete(TEntity entity);
}

public interface IUnitOfWork
{
    IRepository<TEntity> Repository<TEntity>() where TEntity : class;

    Task SaveChanges();
}
```

Рисунок 8 – Код реалізації принципу «Dependency Inversion» на прикладі інтерфейсів IRepository та IUnitOfWork

Патерни «Repository» та «Unit Of Work» призначені для створення абстракції між рівнями доступу до даних і бізнес-логіки додатка. Реалізація даних шаблонів дозволяє ізолювати додаток від змін в сховище даних і спрощує автоматичне модульне тестування або розробку на основі тестування. Фрагмент коду реалізації патерну «Repository» зображено на рисунку 9.

Безпосередньо репозиторій є об’єктом, що забезпечує роботу з проекцією даних, що зберігаються у сховищі, у пам’яті та синхронізацію цієї проекції зі сховищем. Даний об’єкт надає інтерфейс для здійснення операцій щодо зчитування та запису даних стосовно конкретної доменної сутності, приховуючи деталі і технології, що лежать в основі комунікації зі сховищем

Оскільки репозиторій створюється для однієї сутності, то для виконання складного бізнес процесу, який одночасно охоплює зміну стану декількох сутностей, необхідно мати стільки репозиторіїв, скільки сутностей. Кожен репозиторій здійснює синхронізацію із базою даних через контекст. Використовуючи підхід «Dependency Inversion», де контекст передається у

репозиторій у якості параметру, немає жодної гарантії, що усі ці репозиторії отримають один й той самий контекст, що є необхідною вимогою для успішного виконання подібної операції.

```
public class EfRepository<TEntity> : IRepository<TEntity> where TEntity : class
{
    private readonly ApplicationDbContext dbContext;
    private readonly DbSet<TEntity> dbSet;

    public EfRepository(ApplicationDbContext dbContext)
    {
        this.dbContext = dbContext;
        this.dbSet = this.dbContext.Set<TEntity>();
    }
    public Task<List<TEntity>> ListAsync(ISpecification<TEntity> specification)
    {
        // fetch a Queryable that includes all expression-based includes
        IQueryable<TEntity> queryableResultWithIncludes = specification.Includes.Aggregate(
                this.dbSet.AsQueryable(),
                (current, include) => current.Include(include));

        // modify the IQueryable to include any string-based include statements
        IQueryable<TEntity> secondaryResult = specification.IncludeStrings.Aggregate(
                queryableResultWithIncludes,
                (current, include) => current.Include(include));

        // filter according to specification`s criteria
        if (specification.Criteria != null)
        {
            secondaryResult = secondaryResult.Where(specification.Criteria);
        }

        // order by the IQueryable
        if (specification.OrderBy != null)
        {
            secondaryResult = specification.OrderBy(secondaryResult);
        }

        // return the result of the query
        return secondaryResult.ToListAsync();
    }
        ---- [part of code omited] ----
}
```

Рисунок 9 – Фрагмент коду реалізації патерну «Repository»

Одночасна зміна стану зв’язаних сутностей, що належать до різних контекстів приведе до помилки і не буде здіснена. Для вирішення цієї проблеми необхідно використовувати патерн «Unit Of Work». Він однозначно та явно забезпечує той факт, що всі репозиторії, які входять до конкретної одиниці роботу

матимуть єдиний контекст і збереження цієї одиниці роботи приведе до одночасної синхронізації даних з усіх цих репозиторіїв з базою даних.

Для того, щоб відділити бізнес-логіку додатку від цієї залежності необхідно використати принцип «Dependency Inversion». Для цього створимо інтерфейси IRepository та IUnitOfWork імплементацію цих інтерфейсів EfRepository та EfUnitOfWork відповідно. Інтерфейси є частиною проекту з бізнес-логікою додатку (EA.Common), в той час, коли їх реалізації знаходиться в проекті з інфраструктурною функціональність (EA.Persistance). Реалізація патерну «Unit Of Work» зображена на рисунку 10.

```
public class EfUnitOfWork : IUnitOfWork
{
    private readonly ApplicationDbContext dbContext;
    private readonly ConcurrentDictionary<string, object> repositories;

    public EfUnitOfWork(ApplicationDbContext dbContext)
    {
        this.dbContext = dbContext;
        this.repositories = new ConcurrentDictionary<string, object>();
    }

    public IRepository<TEntity> Repository<TEntity>() where TEntity : class
    {
        object repo = this.repositories.GetOrAdd(
            typeof(TEntity).FullName,
            key => new EfRepository<TEntity>(this.dbContext));

        return (IRepository<TEntity>)repo;
    }

    public Task SaveChanges()
    {
        return this.dbContext.SaveChangesAsync();
    }
}
```

Рисунок 10 – Реалізація патерну «Unit Of Work»

Завдяки цьому при зміні технології збереження даних, необхідно буде вносити зміни лише в проект EA.Persistance, в той час як бізнес логіка залишиться без змін.

Для зв’язку бізнес логіки та шару представлення, яким є Web API, було використано бібліотеку MediatR. Дана бібліотека містить реалізацію патерну «Медіатор», який визначає об'єкт, що приховує спосіб взаємодії множини об'єктів. Фрагмент коду з використанням даного підходу зображено на рисунку 11.

```
[Route("api/[controller]/[action]")]
public class AccountController : Controller
{
    private readonly IMediator mediator;

    public AccountController(IMediator mediator)
    {
        this.mediator = mediator;
    }

    [HttpPost(Name = "AccountRegister")]
    [AllowAnonymous]
    public async Task<IActionResult> Register([FromBody]RegisterInputModel model)
    {
        if (this.ModelState.IsValid)
        {
            try
            {
                AuthorizedUser authUser = await this.mediator.Send(new RegisterUserCommand(
                    model.UserName,
                    model.Email,
                    model.Password,
                    model.ConfirmPassword));

                return new OkObjectResult(authUser);
            }
            catch (UserValidationException ex)
            {
                foreach (var error in ex.Errors)
                {
                    ModelState.AddModelError(string.Empty, error.Description);
                }
            }
        }

        return new BadRequestObjectResult(this.ModelState);
    }

    ----[part of code omitted] ----
}
```

Рисунок 11 – Фрагмент коду з використанням патерну «Медіатор» для звязку логіки додатку з шаром представлення

«Медіатор» забезпечує слабку зв'язаність системи, звільняючи об'єкти від необхідності явно посилатися один на одного, і дозволяючи тим самим незалежно змінювати взаємодії між ними. Використання даного підходу дозволяє

підтримувати контролери «тонкими», за рахунок того, що основна логіка винесена у обробники запитів. Також це дозволяє зменшити залежність логіки додатку від конкретного веб-фреймворку.

Вищезазначені шаблони і практики проектування були спрямовані на збільшення модульності та розширюваності системи. Наступним завданням є підтримка цілісності системи. Цілісність є важливою частиною системи електронного архіву, оскільки основне завдання архіву – довготривале збереження інформації. Підтримання послідовності в розподіленій системі є складним завданням.

Для досягнення цілісності операцій, які включають декілька джерел збереження даних використана наступна комбінація шаблонів і алгоритмів:

- Distributed Transaction with Two-Phase Commit – підхід підтримки цілісності між кількома ресурс-менеджерами, які приймають участь в транзакції. Особливість цього підходу полягає в наявності менеджера транзакції, який координує процес виконання транзакції і упевнюється у тому, що кожен з ресурс менеджерів зберіг зміни;
- Transaction Log Tailing – підхід до оновлення сховища даних, який передбачає написання наміру дії оновлення в окремий стійкий файл журналу і після збереження цього файлу журналу на диску виконати оновлення джерела даних;
- Compensating Transaction – схема дизайну, яка передбачає, що команда / обробник, що оновлює джерело даних, має компенсуючу дію, яка може скасувати зміни до джерела даних, які були зроблені передбачуваною дією;
- Idempotent Event Handlers – підхід, що дозволяє виконувати один і той же обробник з однаковими аргументами кілька разів, і кожен раз результат буде однаковим. Таким чином, виклик обробника подій до доданого елемента з тим самим ідентифікатором до бази даних не повинен ні додавати декілька записів, ні викидати помилки, а створювати запис лише один раз.

Підтримка цілісності є надзвичайно важливим завданням для системи електронної архівації даних. Тож у якості основного підходу підтримки цілісності даних було обрано протокол розподіленої транзакції з двох фазним комітом (Distributed Transaction with Two-Phase Commit). Суть цього протоколу полягає в наявності двох типів учасників:

– Transaction Manager – актор, що керує транзакцією;
– Resource Manager – актор, що керує станом конкретного ресурсу, зміна стану якого виконується у транзакції.

У архівній системі є три типи постійних ресурсів:

– SQL Database – зберігає усі дані системи;
– File Storage – зберігає безпосередньо контент інформаційних пакетів та технічні файли системи (схеми метаданих, логи транзакцій, тощо);
– Search Engine – зберігає проекцію Discriptive Information інформаційного пакету, за якою виконується повнотекстовий пошук.

При додаванні інформаційного пакету в архів, частка даних цього пакету повинна бут записана в усі три ресурси збереження даних. Дана операція повинна бути атомарною, а саме, якщо один з ресурсів не зможе зберегти запис, усі інші ресурси, якщо вони вже створили записи для цього пакета, повинні відмінити їх.

Протокол розподіленої транзакції з двох фазним комітом разом з шаблоном компенсаційної транзакції та шаблоном Transaction Log Tailing дозволяють досягти вищезазначеної мети. Для цього були використано класи з простору імен System.Transactions з .Net Framework. Для реалізації було обрано підхід явної транзакції. Для цього будь-яка дія з ресурсом має бути реалізована у вигляді класу, що реалізує інтерфейс IEnlistmentNotification, який складається з наступних методів:

а) Prepare – цей метод буде викликано управляючим транзакції під час першої фази коміту. Код даного методу повинен зробити дві дії:
    1) записати Transaction Log – усю інформацію про операцію, що буде виконуватися, у надійне сховище даних, щоб у разі помилки можна було відмінити зміни;

2) безпосередньо виконати операцію над тим ресурсом, яким володіє даний управляючий. Якщо операція пройшла успішно, даний управляючий ресурсом повинен повідомити про це управляючого транзакцією, щоб той продовжив виконання транзакції. Якщо операція завершилася помилкою, даний управляючий ресурсом повинен повідомити управляючого транзакцією про помилку, щоб він ініціював процедуру відкату розподіленої транзакції;

б) Commit – цей метод буде викликано управляючим транзакції під час другої фази коміту, коли усі ресурс менеджери повідомили про успішне виконання операцій над ресурсами, якими вони управляють. Код даного методу повинен зробити результат операції, виконаної у методі Prepare постійним і видалити Transaction Log запис для цієї транзакції, що буде свідчити про її завершення;

в) Rollback – цей метод буде викликано управляючим транзакції під час другої фази коміту, коли хоча б один з ресурс менеджерів повідомив про помилку під час виконання операцій над ресурсами, якими вони управляють. Код даного методу повинен відмінити результат операції, виконаної у методі Prepare і видалити Transaction Log запис для цієї транзакції, що буде свідчити про її завершення.

В системі є три постійних ресурси даних, але оскільки .Net Framework вже має реалізацію управляючого ресурсами для роботи з базою даних SQL Server, тож необхідно реалізовувати управляючих ресурсами для операцій з файловою системою та пошуковим рушієм.

Щоб не реалізовувати управляючого транзакцією самостійно, Microsoft пропонує компонент, який є частиною операційної системи Windows – Microsoft Distributed Transaction Coordinator (MSDTC). Код для інтеграція з цим компонентом є частиною .Net Framework простору імен System.Transactions, що значно спрощую процес розробки.

4.3 Вибір технологічного стеку

Для розробки системи електронного архіву наступний технологічний стек є доцільним:

- Asp .Net Framework MVC and Web API 2.0 – фреймворк для розробки веб-орієнтованих додатків, що виконуються в середовищі під управлінням операційної системи Windows [18];
- MS SQL Server – реляційна база даних SQL, традиційна у світі .Net. Дана база даних використовується для зберігання даних більшості даних системи електронного архіву; Перевагою цієї технології є наявність клієнтської бібліотека для мови програмування C# з підтримкою розподілених транзакцій. Недоліком даної технології є відсутність з коробки горизонтального масштабування, тож його доведеться робити власноруч;
- Elasticsearch – пошуковий рушій, який використовується для індексації Descriptive Information і тексту, вилученого з Content Information. Він підтримує гнучкі запити, агрегації, повнотекстові запити, пропозиції та інші функції, важливі для системи електронного архіву. Elasticsearch підтримує горизонтальне масштабування і його можна легко розгорнути як кластер;
- Apache Tika – інструментарій вилучення тексту, який використовується для вилучення тексту з Content Information інформаційного пакету для подальшої індексації в пошуковій системі. Це відкрите рішення із зручним API HTTP. Оскільки це служб, яка не зберігає стан, її можна легко масштабувати горизонтально;
- Azure Blob Storage – хмарне сховище для файлів інформаційних пакетів. Цей тип сховища підтримує реплікацію, необхідну для системи електронного архіву. Перевагою цієї технології є низька вартість та наявність клієнтської бібліотека для мови програмування C#;
- SMB File Share – альтернативне сховище для файлів інформаційних пакетів; Перевагою цієї технології є можливість горизонтального

масштабування шляхом розгортання у Windows Failover Cluster. До недоліків відноситься складність розгортання такого кластеру;

– Microsoft Distributed Transaction Coordinator – компонент Microsoft Windows, призначений для координації зміни даних на двох або більше постійних носіях. Перевагою цієї технології є підтримка участі будь-яких ресурсів в розподіленій транзакції. Недоліком цієї технології є складність розробки клієнтського коду для нових ресурсів, які не підтримуються з коробки, а також жорстка прив’язка до операційної системи Windows, що робить неможливим її використання у хмарних сервісах.

Поєднання цих технологій дозволяє створити систему електронного архіву, яка повністю реалізує всі необхідні функції, може обробляти великі обсяги даних, відповідати вимогам щодо продуктивності, цілісності та відновлення.

Для розгортання системи використані віртуальні машини під управлінням операційної системи Windows у хмарному середовищі Azure.

5 ТЕСТУВАННЯ СИСТЕМИ

Тестування програмного забезпечення – це процес технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись.

5.1 Функціональне тестування

Метою функціонального тестування є виявлення невідповідностей між реальною поведінкою реалізованих функцій і очікуваною поведінкою відповідно до специфікації і вимог. Для кожного модулю системи функціональні тести охоплювали всі реалізовані функції з урахуванням найбільш ймовірних типів помилок. Під час розробки програмної системи було використано наступні методи тестування:

– шляхом «білої скриньки» – було проведено дослідження внутрішні елементи програми та зв’язки між ними. Даний метод тестування використовувався під час написання модульних тестів;
– шляхом «чорної скриньки» – було проведено дослідження роботи кожної з функцій системи використовуючи інтерфейс користувача; Даний метод тестування було використано під час написання модульних тестів.

Загалом тестування систему було проведено на трьох рівнях: системне тестування, інтеграційне тестування (Integration Testing) та модульне тестування (Unit Testing) [19];

Для тестування системи на системному рівні виконувалося було виділено окрему віртуальну машину, на якій було розгорнуто систему та проведено мануальне тестування.

Інтеграційне тестування системи полягало в написанні інтеграційних тестів до Web API. Для цього було створено окремий проект «EA.Tests.Integration».

Модульне тестування проводилося безпосередньо під час розробки системи шяхом написання модульних тестів. Для написання модульних тестів було використано «MS Test» фреймворк від Microsoft.

5.2 Нефункціональне тестування

Розроблена система електронного архіву була протестована на відповідність наступним нефункціональним вимогам:

– Performance;
– Scalability;
– Availability;
– Recoverability;
– Security.

Для проведення навантажувального тестування були підготовлені дані, наведені у таблиці 2.

Таблиця 2 – Дані для проведення навантажувального тестування

Storage, MB	CI Size, MB	CI Count In AIP	DI Size, MB	Count of AIPs with 1 CI & 1KB DI	Count of AIPs with 5 CI & 10KB DI	Count of AIPs with 100 CI & 100KB DI
1,000,000	1	1	0.001	999,000.999	199,600.798	9,990.010
1,000,000	10	5	0.010	99,990.001	19,996.001	999.900
1,000,000	100	100	0.100	9,999.900	1,999.960	99.999
1,000,000	1,000			999.999	200.000	10.000

Дані для тестів були підібрані з урахуванням наступних параметрів:

– Максимальний розмір доступного сховища в системі – Storage;
– Кількість файлів, що становлять Content Information, у пакеті – CI Count;
– Розмір кожного файлу, що становить Content Information, у пакеті – CI Size;
– Розмір файлу, що становить Descriptive Information, у пакеті– DI Size;

Щоб тестові дані були максимально наближені до реальних, було розглянуто три різних за об’ємом і кількістю файлів типи пакетів:

– Пакети з 1 CI файлом, розмір якого вирує від 1 до 1000MB;
– Пакети з 5 CI файлами, розмір яких вирує від 1 до 1000MB;

– Пакети зі 100 CI файлами, розмір якого вирує від 1 до 1000MB.

Для проведення тестування масштабованості системи було обрано 4 різних конфігурації системи:

– Одна віртуальна машина з 2-х ядерним процесором і 4 ГБ оперативної пам’яті;
– Дві віртуальних машини з 2-х ядерним процесором і 4 ГБ оперативної пам’яті кожна;
– Три віртуальних машини з 2-х ядерним процесором і 4 ГБ оперативної пам’яті кожна;
– Одна віртуальних машини з 4-х ядерним процесором і 16 ГБ оперативної пам’яті;

Детальні характеристики кожної з конфігурацій наведено у таблиці 3.

Таблиця 3 – Технічні характеристики різних конфігурацій розгортання системи для проведення тестування масштабованості

Configurations	1	2	3	4
VM Size	B2s	B2s	B2s	B4ms
VMs Count	1	2	3	1
RAM per VM, GB	4	4	4	16
vCPUs Count per VM	2	2	2	4
vCPU Frequency, Ghz	2.4	2.4	2.4	2.4
Data Disk Size per VM, GB	1024	1024	1024	1024
Data Disks Count per VM	2	1	1	2
Data Disk Type	S30 (Standard HDD)	S30 (Standard HDD)	S30 (Standard HDD)	S30 (Standard HDD)
Data Disk Cost, $/month	40.96	40.96	40.96	40.96
VM Cost, $/month	39.88	39.88	39.88	154.75
Total vCPU Count	2	4	6	4

Продовження таблиці 3

Configurations	1	2	3	4
Total vCPU Count	2	4	6	4
Total RAM, GB	4	8	12	16
Total Data Disk Size, GB	2048	2048	3072	2048
Total Cost, $/month	121.8	161.68	242.52	236.67

У таблиці 4 наведено тест кейси для «Load & Scalability» тестування, які були проведено для кожної з конфігурацій системи.

Таблиця 4 – Тест-кейси для «Load & Scalability» тестування системи

Test-Case	CI Size, MB	CI Count	DI Size, MB	AIPs Count	Total Size, MB	Simultaneous Users Count
1	10	5	0.010	500	25,005	1
2	10	5	0.010	500	25,005	4
3	10	5	0.010	500	25,005	8
4	10	5	0.010	500	25,005	16

У таблиці 5 наведено тест кейси для «Volume & Scalability» тестування, які були проведено для кожної з конфігурацій системи.

Таблиця 5 – Тест-кейси для «Volume & Scalability» тестування системи

Test-Case	CI Size, MB	CI Count	DI Size, MB	AIPs Count	Total Size, MB	Simultaneous Users Count
1	1	1	0.001	999,000.999	1,000,000	4
2	10	5	0.010	19,996.001	1,000,000	4
3	100	5	0.010	1,999.960	1,000,000	4
4	1,000	5	0.010	200.000	1,000,000	4
10	10	100	0.100	999.900	1,000,000	10

Для тестування безпеки системи на відповідність OWASP Top 10 2017 було використано OWASP ZAP. Це сканер для веб-орієнтованих рішень з відкритим вихідним кодом, що написаний на Java. Інтерфейс програми зображено на рисунку 12. Під час тестування сканером було виявлено наступні загрози, які були пропущені під час розробки:

– X-Frame-Options Header Not Set;
– Web Browser XSS Protection Not Enabled
– X-Content-Type-Options Header Missing.

Вищезазначені загрози було усунено під час фази стабілізації системи.

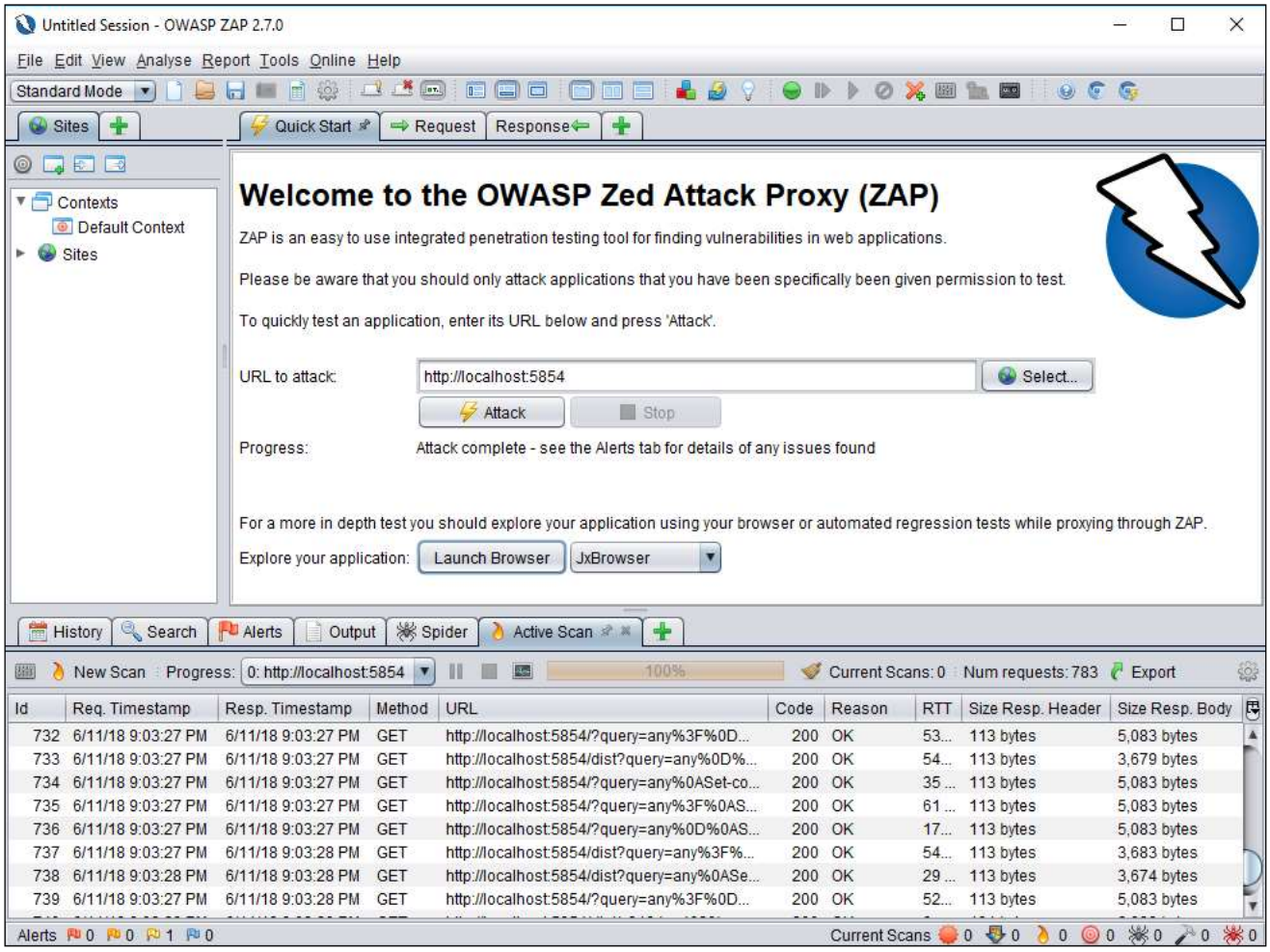
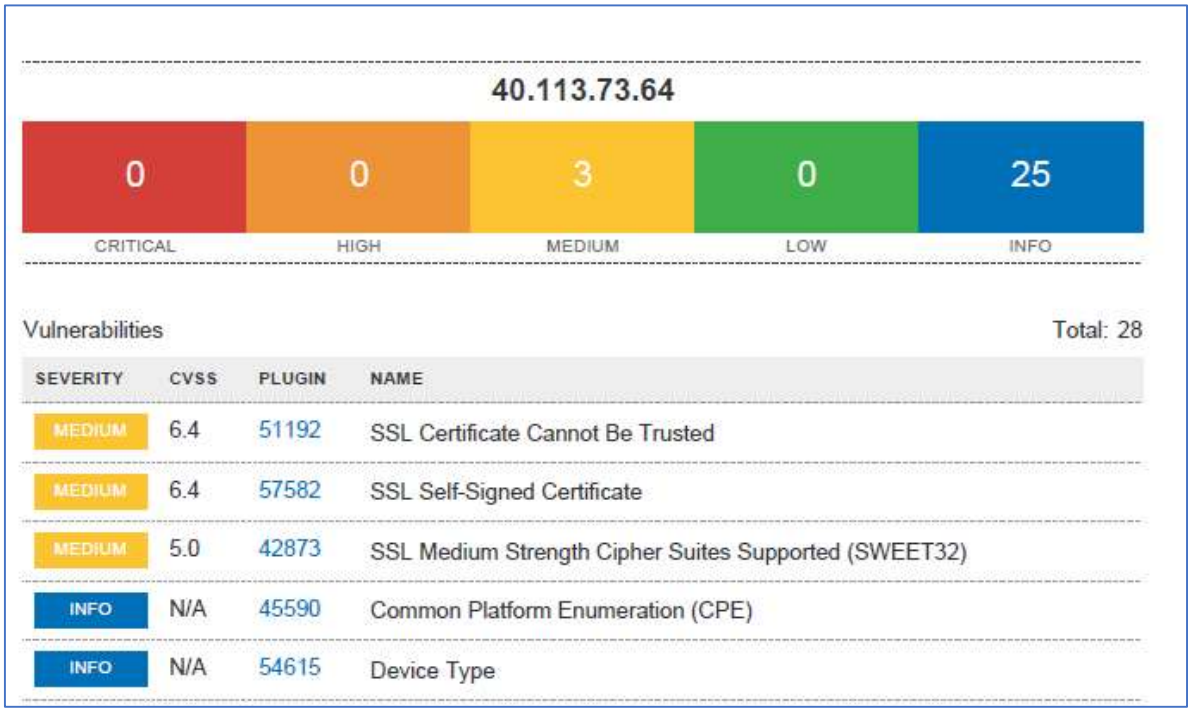


Рисунок 12 – Інтерфейс програми OWASP ZAP

Оскільки система розгорнута на віртуальній машині, яка знаходиться у відкритому доступі в Інтернеті, необхідно провести перевірку безпечності

конфігурації цієї машини. Для цього було використано сканер Nessus – програма для автоматичного пошуку відомих вад у захисті інформаційних систем [20]. Вона здатна виявити найпоширеніші види вразливостей, такі як наявність вразливих версій служб або доменів, помилки в конфігурації (наприклад, відсутність необхідності авторизації на SMTP-сервері), наявність паролів за замовчуванням, порожніх, або слабких паролів.

Фрагмент репорту з перевірки безпеки наведено на рисунку 13. Згідно з результатами сканування, критичних проблем не було знайдено, а усі Medium проблеми пов’язані із само виписаним SSL сертифікатом, що встановлено на сервері і використовується для шифрування трафіку для сайту системи електронного архіву.



40.113.73.64

0	0	3	0	25
CRITICAL	HIGH	MEDIUM	LOW	INFO

Vulnerabilities — Total: 28

SEVERITY	CVSS	PLUGIN	NAME
MEDIUM	6.4	51192	SSL Certificate Cannot Be Trusted
MEDIUM	6.4	57582	SSL Self-Signed Certificate
MEDIUM	5.0	42873	SSL Medium Strength Cipher Suites Supported (SWEET32)
INFO	N/A	45590	Common Platform Enumeration (CPE)
INFO	N/A	54615	Device Type

Рисунок 13 – Фрагмент результату перевірки конфігурації хост серверу системи електронного архіву

Таким чином, розроблена система електронного архіву відповідає поставленим нефункціональним вимогам, а протестовані конфігурації хост серверу можуть бути використані для впровадження системи і її подальшого масштабування.

6 ОЦІНКА ВПЛИВУ АРХІТЕКТУРИ І ТЕХНОЛОГІЙ НА АТРИБУТИ ЯКОСТІ СИСТЕМИ

Розроблена система електронного архіву відповідає наступним атрибутам якості програмного забезпечення:

а) Conceptual Integrity та Maintainability – система має послідовний, уніфікований дизайн, відповідає моделі предметної галуз та має гранульовану модульну структуру. Maintainability Index кодової бази системи, розрахований за допомогою Visual Studio 2019, дорівнює 86 зі 100. Даний атрибут якості був досягнений завдяки комбінації наступних шаблонів і технологій:
   1) багаторівневу тришарова архітектура у комбінації з «Цибулиною» архітектурою забезпечили виділення горизонтальних слоїв абстракцій;
   2) предметно-орієнтованого проектування дозволило розділити кодову базу на вертикальні частки, що забезпечує велику зв’язність між компонентами у межах однієї частки і практично виключає зв’язанність між компонентами різних часток, що мінімізує область ураження при внесенні змін в систему;
   3) шаблону проектування SQRS та Mediator дозволили уніфікувати стиль написання компонентів бізнес логіки. Також це дозволило зменшити кількість функціональності на однин компонент коду, оскільки на кожен варіант використання, представлений командою чи запитом, створюється свій компонент коду для обробки. Це дозволяє незалежно змінювати обробники різних варіантів використання, що відносяться до однієї функціональної частки;

б) Reusability – компоненти системи є придатними до повторного використання, завдяки використання:
   1) мови програмування високого рівня C#;
   2) принципів Inversion of Control та Dependency Inversion при проектуванні рівнів, модулів і компонентів системи;

в) Manageability – адміністрування системою є досить простим завдяки концептуальній простоті багаторівневої архітектури та невеликої кількості використаних технологій:
   1) індивідуальні компоненти – Microsoft SQL Server, Elasticsearch, Apache Tika;
   2) компоненти хостинг середовища під управлінням операційної системи Windows – IIS, Microsoft Distributed Transaction Coordinator;
   3) сторонні сервіси – Azure Blob Storage;

г) Supportability – забезпечується за рахунок запису усіх помилок у текстовий файл, використовуючи NLog бібліотеку для мови програмування C#;

д) Testability – система є легкою в тестування на усіх рівнях:
   1) Unit Testing – завдяки шаблону Dependency Inversion усі компоненти системи залежать від абстракції, а не від конкретних реалізації, тому для них легко писати модульні тести;
   2) Integration Testing – завдяки тому, що система має RESTful API із нею легко інтегруватися, оскільки у якості клієнту може виступати будь-який застосунок здатний робити HTTP запити (Postman, Curl, Rest Client extension for Visual Studio Code, JMeter, Powershell та інші);

е) Performance – система задовольняє нефункціональним вимог з продуктивності. Проведене навантажувальне тестування показало, що при мінімальному розгортанні (1 віртуальна машина з 2-х ядерним процесором з частотою 2.6 GHz і 4 гігабайт оперативної пам’яті) система виконує імпорт 25 гігабайт даних представлених 500 пакетами одночасно 4 користувачами за 7 хвилин 19 секунд. При цьому процесор є зайнятим на 90-100 відсотків, що говорить про оптимально написаний код, який використовую асинхронні операції при взаємодії з системними ресурсами

ж) Scalability – система підтримує горизонтальне масштабування, за рахунок прийнятих програмних рішень і використаних технологій:

1) MS SQL Server з підходом вертикального розділення на частки. Система має два типи інформації: інформація про системні ресурси і інформація про пакети, що знаходяться в архівах. Системна інформація має низькі темпи росту і невеликий об’єм даних, в той час як архівна інформація буде швидко рости, і може містити сотні мільйонів записів про пакети. Оскільки MS SQL Server не підтримує горизонтальне масштабування, збереження усієї інформації в одній базі даних перекриє можливість горизонтального масштабування. Тому було прийнято рішення розділити бази даних: мати 1 системну базу даних і по 1 базі даних на кожен архів. Це дозволить мати пул екземплярів MS SQL Server, на які рівномірно, використовуючи Round Robin алгоритм, розподіляти бази даних по цим серверам. Використання даного підходу надало майже лінійну масштабованість з коефіцієнтом 1.8 (відношення кількості екземплярів MS SQL Server до приросту потужності системи).
2) SMB File Share у якості сховища даних забезпечує можливість незалежного масштабувати сховища файлів за рахунок технології SMB Scale Out on Windows Failover Cluster або Azure File Share. В той же час це дозволяє додавати екземпляри програмного забезпечення системи електронного архіву, які приховані за балансером навантаження, завдяки тому що усі вони будуть працювати з єдиним сховищем даних – SMB File Share;
3) Elasticsearch – підтримує горизонтальне масштабування «з коробки»;
4) Apache Tika – не має стану, тому може бути легко масштабована горизонтально.

з) Reliability – система здатна самостійно відновлюватися після помилки, що досягається за рахунок комбінації наступних шаблонів проектування:
1) Distributed Transaction with Two-Phase Commit;
2) Transaction Log Tailing;

3) Compensating Transaction;
4) Retry with Exponential Backoff;
5) Circuit Breaker.

Так, використання патерну Retry with Exponential Backoff протягом 60 хвилин безперервного імпорту даних при симуляції виходу з ладу віртуальної машини у вигляді 5 інтервалів 2 хвилини кожен зменшило кількість помилок з 600 до 10.

и) Availability – забазпечується шляхом комбінації наступних технологій розгортання і реплікації:
   1) MS SQ Server Failover Cluster over Windows Failover Custer – у разі вихода з ладу вузла, на якому виконувався екземпляр серверу бази даних, дозволяє налаштувати автоматичний «failover» серверу бази даних до іншого активного вузла кластеру;
   2) SMB Scale Out over Windows Failover Custer;
   3) Elasticsearch реплікація – створення реплік часток, з яких складається індекс дозволяє робити репліки основними частками на активних вузлах при виході з ладу вузлів, на яких зберігалися основні частки;
к) Security – забазпечується за рахунок використання функції захисту від поширених загроз, реалізованих у Asp .Net Framework MVC and Web API 2.0 фреймворку та відповідному налаштуванні хостинг середовища.

Спроектована архітектура і технологічний стек розробки і впровадження системи забезпечують відповідність системи усім необхідним атрибутам якості.

ВИСНОВКИ

В результаті виконання атестаційної роботи було спроектовано архітектуру архівної системи та створено прототип, який відповідає OAIS моделі та задовольняє обраним атрибутам якості програмного забезпечення.

Під час виконання атестаційної роботи було проведено:

– аналізі функцій, обов'язків та моделі даних, яким має відповідати система електронного архіву;
– аналіз і визначення атрибутів якості якими має володіти система електронної архівації даних;
– аналіз існуючих архітектурних стилів і шаблонів розробки на предмет доцільності їх для набуття системою необхідних атрибутів якості;
– навантажувальне тестування системи для обробки різних об’ємів даних, використовуючи різні модифікації архітектури, технологій і стратегії розгортання системи;
– оцінку результатів навантажувального тестування з подальшим визначенням відповідної комбінації складових частин системи для отримання бажаної поведінки системи при обробці різних об’ємів даних.

Отримані результати можуть бути використані для проектування нових систем електронних архівів, а також для визначення напряму, релевантності і вартості зміни архітектури і технологій існуючих архівних систем.

Також варто відзначити, що створена система має відкриті питання для подальшого дослідження:

– розробка стратегії мінімізації витрат для розгортання системи у використання;
– дослідження меж масштабування запропонованої архітектури;
– дослідження впливу зміни технологій на атрибути якості системи;

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Шрайберг Я. Роль бібліотек у забезпеченні доступу до інформації та знань в інформаційному віці: навч. посіб. – Київ, 2007. – С. 60 –74.

2. Нежурбида Г. Г. Функциональные требования к системам долговременного хранения электронных документов / Редкол.: А. С. Онищенко и др.; Библиотеки национальных академий наук: проблемы функционирования, тенденции развития. Киев:2006. Вып. 4. С. 438.

3. X. Jin, B.W. Wah, X. Cheng and Y. Wang. Significance and challenges of big data research, Big Data Research, vol. 2, issue 2, June 2015, pp. 59-64.

4. Consultative Committee for Space Data Systems. REFERENCE MODEL FOR AN OPEN ARCHIVAL INFORMATION SYSTEM (OAIS). Washington, DC 20546-0001, USA, 2012, p. 4-1.

5. Cesar de la Torre, Bill Wagner and Mike Rousos. .NET Microservices: Architecture for Containerized .NET Applications, v2.2 ed., Redmond, Washington 98052-6399, 2019, pp.311–324.

6. S. Somasegar, Scott Guthrie, David Hill. Microsoft Application Architecture Guide 2nd Edition, Patterns & Practices, Microsoft Press, 2009, pp. 191–204.

7. Alex Homer, John Sharp, Larry Brader, Masashi Narumoto, Trent Swanson. Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications, Microsoft patterns & practices, 2014, pp.190–197.

8. Десять найбільш критичних ризиків для безпеки веб-додатків. // OWASP Топ 10 – 2013. URL: https://www.owasp.org/images/e/e3/OWASP_Top_10_-_2013_Final_Ukrainian.pdf (дата звернення: 30.10.2019 р.).

9. Martin Fowler, Kendall Scott. UML Distilled Second Edition A Brief Guide to the Standard Object Modeling Language, Addison Wesley, Second Edition August 18, 1999. – p.224.

10. Eric Evans. Domain-Driven Design: Tackling Complexity In The Heart Of Software 1st Edition, Addison-Wesley Professional, 2003. – p.560.

11. Eric van der Vlist. XML schema - the W3C's object-oriented descriptions for XML, O'Reilly, January 2002. – p.398.

12. Joseph Ingeno. Software Architect's Handbook, Packt Publishing, August 2018. – p.594.

13. Dominick Baier, Vittorio Bertocci, Keith Brown, Scott Densmore, Eugenio Pace, Matias Woloski. A Guide to Claims-Based Identity and Access Control: Authentication and Authorization for Services and the Web, Microsoft patterns & practices, Microsoft Press, 2013, pp 128–231.

14. Дудар З.В., Каук В.И., Шкиль П.И. Концепція уніфікованого доступу в росподіленому інформаційному просторі // Вестник по материалам Новая каховка. 2005.

15. The Onion Architecture: part 1 // Jeffrey Palermo (.com). URL: http://jeffreypalermo.com/blog/the-onion-architecture-part-1/ (дата звернення: 30.10.2019).

16. Dominic Betts. Exploring CQRS and Event Sourcing: A journey into high scalability, availability, and maintainability with Windows Azure, Microsoft patterns & practices, Microsoft Press, 2013. – pp 376.

17. Steven van Deursen and Mark Seemann. Dependency Injection Principles, Practices, and Patterns; Manning Publications, March 2019. – p.552.

18. Get Started with ASP.NET Web API 2 (C#) – ASP.NET 4.x // Microsoft Docs. URL: https://docs.microsoft.com/en-us/aspnet/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api (дата звернення: 30.10.2019 р.)

19. Roy Osherove. The Art of Unit Testing, Second Edition with examples in C#, Black & White, 2013 – p. 296

20. Дудар З.В., Збитнева М.В. Исследование программных средств защиты информации // Автоматизированные системы управления и приборы автоматики. 2006. № 137. С. 14- 20