КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра комп'ютерної інженерії

До захисту допущено: «На правах рукопису»
Завідувач кафедри _____ Юрій Бойко
« _ » _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему:

«РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВЕБ-ПЛАТФОРМИ З ПОШИРЕННЯ КУЛЬТУРНИХ ІНІЦІАТИВ»

Виконав:
студент 4-го курсу бакалаврату
денної форми навчання
спеціальності 123 Комп'ютерна інженерія
ОНП «_____»
Бакум Антон Владиславович _____

Науковий керівник:
кандидат фізико-математичних наук, доцент
Моторна Оксана Віталіївна _____

Рецензент:
_____

Засвідчую, що у цій бакалаврській роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент _____

Робота допущена до захисту в ЕК рішенням кафедри _____
від «__»_____ 2023 р., протокол № __.

Завідувач кафедри _____,
кандидат фізико-математичних наук, доцент
Бойко Юрій Володимирович

(підпис)

Київ – 2023

РЕФЕРАТ

Випускна кваліфікаційна робота за об’ємом складає 57 сторінок, містить 23 рисунки, 1 таблицю, 6 додатків, використано 13 інформаційних джерел.

Розглянуто: приклади та проекти платформ, що мають за мету створення громадських ініціатив (подій) і реалізацію інституту прямої демократії, підходи до комплексної розробки програмного забезпечення, включаючи архітектурні патерни, питання безпеки програмного забезпечення, підходи до роботи з даними, обробку помилок та виключних ситуацій.

Зроблено: реалізовано програмне забезпечення для перегляду та долучення до культурних ініціатив (подій), перегляду новин в області культури та створення власних новин, а також розроблено програмне забезпечення для авторизації та автентифікації користувача

Ключові слова: ГРОМАДСЬКІ ІНІЦІАТИВИ, АВТЕНТИФІКАЦІЯ, АВТОРИЗАЦІЯ, ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, “ ЧИСТА АРХІТЕКТУРА”

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ 4

ВСТУП 5

1. ПОСТАНОВКА ЗАДАЧІ І ФОРМУВАННЯ ВИМОГ 7

1.1 Кейс вимог до програмного забезпечення 7

1.2 Вимоги до забезпечення безпеки, авторизації та аутентифікації 7

1.3 Вимоги до розділу новин 8

1.4 Вимоги до стрічки культурних подій (ініціатив) 9

1.5 Вимоги до аутентифікації користувача та його дій у системі 9

2. ОБРАНІ ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТИ 10

2.1 Серверна частина застосунку 10

2.2 Клієнтська частина застосунку 11

3. АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 14

3.1 Чиста архітектура 14

3.2 Паттерн “модель-представлення-контроллер” 16

4. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 18

4.1 База даних та рівень доступу до даних 18

4.2 Сервіси авторизації та аутентифікації користувача, JWT 21

4.3 Сервіс завантаження фото 25

4.4 Сервіс розсилки електронних листів 26

4.5 Сервіс обробки помилок та ведення журналу 27

5. РЕЗУЛЬТАТИ ТА ЇХ АНАЛІЗ 30

ВИСНОВКИ 38

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 39

ДОДАТКИ 41

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

API - набір програмно реалізованих методів, які описують логіку взаємодії одного типу програмного забезпечення з іншим.

CRUD - Create Update Read Delete

DOM - Document Object Model

IDE - integrated development environment

JS - JavaScript

JSON - JavaScript object notation

JWT - Json Web Token, закодована та піддана хешуванню строка, яка використовується у ролі ключа доступу до ресурсів певного API.

HTTP(S) - Hypertext Transportation Protocol (Secured)

MVC – паттерн проєктування Model-View-Controller

MSSQL – Microsoft Server SQL

ORM - object-relational mapping, технологія програмування, що пов’язує бази даних з концепціями ООП, відображуючи певні таблиці баз даних на реалізовані високорівневі класи.

PDF - Portable Document Format

REST - representational state transfer

JIT - just-in-time compilation

SPA - Single-Page Application

SQL – Structured Query Language

СУБД – система управління базами даними

TS - TypeScript

VDOM - Virtual Document Object Model

Web-API - web application programming interface

ВСТУП

Сьогодні Україна стала на шлях створення сучасної цифрової держави, і великий об’єм роботи було виконано у сфері винесення багатьох державних послуг, які раніше були піддані критиці за сильну забюрократизованість, в цифровий формат. Зараз досвід нашої держави переймають багато сучасних постіндустріальних економік світу. Одним із дієвих і перевірених цифрових інституцій в нашій державі, є інституція прямої демократії, яка реалізована у вигляді механізму петицій до голови держави, які може створити будь-який громадянин України. Цією послугою ще більш активно почали користуватись під час повномасштабної війни, щоб звернути увагу на ті наявні проблеми, що турбують населення.

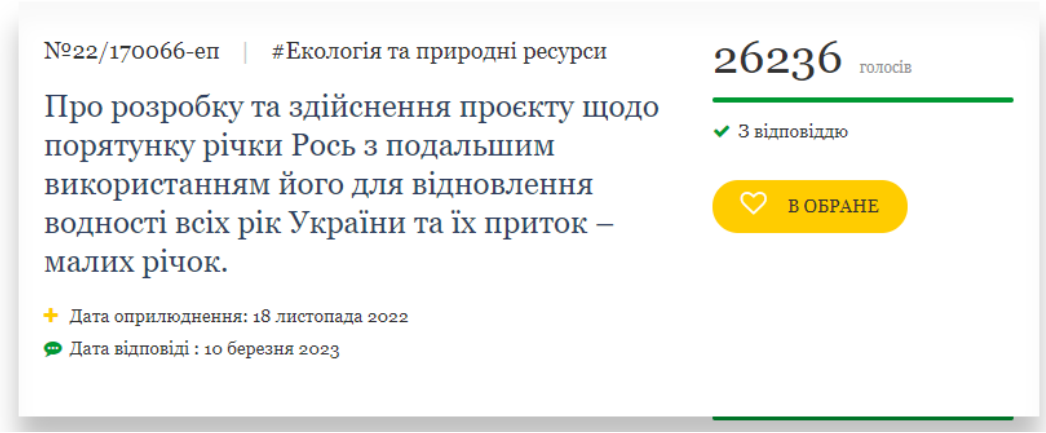


Рис. 1 Приклад електронної петиції на сайті Президента України [1]

На думку автора роботи, позитивні зрушення у системі державного управління мають стимулювати подібні перетворення і у інших сферах функціонування громадянського суспільства. Важливою частиною сучасного українського суспільства є розвиток культурної сфери, популяризація культури серед молоді, просування різноманітних прогресивних ініціатив в суспільстві. Навіть попри війну, в Україні проходять різноманітні культурні заходи: фотовиставки, спектаклі, працюють покази відомих світових та українських митців. Створення веб-застосунку, що дозволяє створювати та модерувати культурні події, підвищувати рівень обізнаності населення про культурне життя України, має підтримати вектор цифровізації українського громадянського

суспільства. Дана платформа може бути розгорнута і в майбутньому розвиватись як проект з відкритим вихідним кодом.

Мета даної випускної кваліфікаційної роботи: розробити програмне забезпечення для можливості переглядати останні новини української мистецької спільноти, мати доступ до створених ініціатив в області культури, підтримувати їх, створювати та поширювати власні події.

ПОСТАНОВКА ЗАДАЧІ ТА ФОРМУВАННЯ ВИМОГ

1.1 Список вимог до програмного забезпечення

Для вирішення задач розробки повнофункціонального веб-застосунку, необхідно чітко визначити увесь кейс вимог до програмного забезпечення, яке має бути реалізоване в процесі роботи над даною кваліфікаційною роботою. Дані вимоги до ПЗ автором роботи було розділено відповідно до підходу ***user-first,*** основою якого є використання підходу до моделювання дій користувача, використовуючи основні можливі патерни поведінки під час взаємодії користувача та даного комплексу прикладного програмного забезпечення. У сучасних процесах створення програмних продуктів, даний підхід активно використовується manual QA-інженерами. Користувацькі вимоги подані у вигляді списку нижче:

1) вимоги до забезпечення безпеки даних користувача та його доступу до ресурсів та можливостей додатку (процеси аутентифікації та авторизації);
2) вимоги до перегляду сторінки новин та взаємодії з її елементами;
3) вимоги до перегляду сторінки із культурними подіями, підтримки певних культурних ініціатив, отримання корисної та необхідної інформації про події;
4) вимоги до можливості редагувати власні ініціативи, змінювати дані про користувача, унікально ідентифікувати кожного користувача в системі.

1.2 Вимоги до забезпечення безпеки, авторизації та аутентифікації

Є кілька перевірених способів забезпечення захисту користувацьких даних від використання сторонніми особами. Основна частина інформації про користувача знаходиться у відповідних таблицях бази даних. Вимогою до ПЗ в контексті безпекових підходів для даної кваліфікаційної роботи є необхідність реалізації комбінованого методу для зберігання чутливих даних, таких як пароль користувача, а саме використання алгоритмів хешування у поєднанні зі створенням секретного ключа, який генерується сервером для кожного унікального користувача системи.

Цей ключ не має бути відомим стороннім додаткам, які представляють небезпеку для чутливих даних веб-застосунку, тому загальна складність процесів проникнення до баз даних за допомогою таких методів, як використання “райдужних таблиць” має бути максимально невигідним для зловмисників. Після успішної авторизації для проведення авторизації користувачу має видаватись пара унікальних токенів, що мають бути використані для доступу до ресурсів додатку, таких як маршрути отримання даних з сервера. Авторизація має статус успішної, коли користувач вперше створює аккаунт в системі, або коли здійснено перевірки пари “логін-пароль” під час повторної ідентифікації користувача у системі на рівні бізнес-логіки.

У вікнах реєстрації та входу користувача у його акаунт має бути наявна клієнтська валідація, що при можливому вводі некоректних даних відобразить повідомлення про помилку користувачу додатку та зробить неможливим потрапляння на сервер застосунку тих даних, які не відповідають правилам безпеки або не є частиною бізнес-логіки.

1.3 Вимоги до розділу новин

Розділ новин у додатку, що має бути розроблений у даній кваліфікаційній роботі, має коректно обслуговуватись програмним забезпеченням додатку, що відповідає за навігацію. Користувач має отримати доступ до сторінки новин, навіть якщо він не є активним користувачем платформи, отже, сторінка новин має бути загально доступною для всіх користувачів мережі Інтернет, що будуть користуватись веб-додатком.

На сторінці новин мають бути представлені список новин із можливістю завантажити додаткові новини з серверної частини додатку. Кожна новина є унікально ідентифікованою в системі, на сторінці новини має відображатись дата створення новини, її заголовок і текст. Користувач має отримати можливість переглянути повний текст новини, шляхом відкриття її в новій вкладці браузера. Додатково у цьому пункті вимог до ПЗ необхідно включити вимогу до реалізації інтерфейсу пошуку новини за її заголовком.

1.4 Вимоги до стрічки культурних подій (ініціатив)

Користувач має отримати можливість переглядати ініціативи та культурні події, створені іншими користувачами. Кожна подібна подія має бути унікально ідентифікована в системі. На картці, що відображає подію, має бути її опис, дата створення та кнопка, що дозволяє користувачу підтримати ініціативу.

Якщо користувач не пройшов етапи аутентифікації та авторизації, дана кнопка не має бути активною, а користувач має отримати інформацію про необхідність авторизації для підтримки чи підпису на дану подію. Додаток має забезпечувати логіку, що попереджає ситуації, коли користувач підписується на подію більше ніж один раз. Подібні ситуації не мають здійснювати негативний вплив на функціональність як користувацького інтерфейсу, так і важливих системних даних. Після того, як користувач підтримав ініціативу, він має отримати на пошту повідомлення з інформацією про подію, дана інформація має бути ідентичною до тієї, яка зображена на картці ініціативи. Повідомлення відправляється на ту пошту, що була використана користувачем для реєстрації.

На сторінці, яка відображає список ініціатив, має бути присутній графік, що відображає список найбільш популярних подій серед користувачів.

1.5 Вимоги до ідентифікації користувача та його дій у системі

Дана вимога до програмного забезпечення, що має бути розроблене у роботі, включає необхідність унікально ідентифікувати користувача в системі. Після успішної аутентифікації, додаток має отримати інформацію про користувача, що має бути використана у тих частинах бізнес-логіки веб-застосунку, де наявна прив’язка до особистості людини, що взаємодіє з системою. Кожен користувач має отримати можливість змінювати дані власного облікового запису, переглядати створені ним ініціативи та вносити певні корективи за необхідності. Також необхідність у даних користувача очевидна під час процесів створення нових культурних подій та доєднання до вже існуючих ініціатив, створених іншими користувачами, а також створенні новин для стрічки.

ОБРАНІ ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТИ

2.1 Серверна частина застосунку

Для реалізації back-end було обрано платформу .NET, та одну з найбільш популярних мов програмування C#. C# - високорівнева компільована мова програмування, призначена для розробки прикладного програмного забезпечення, що може включати різноманітні веб-додатки, консольні додатки, застосунки для персональних комп'ютерів тощо, з можливостями використовувати як підходи функціонального, так і більш класичного об'єктно-орієнтованого програмування.

Серверна частина застосунку розроблялась на платформі .NET версії шість, а саме з використанням легковісного і гнучкого веб-фреймворка ASP.NET Core. Дана платформа представляє такі можливості та переваги:

1) **Автоматична система збору сміття (Garbage Collection)** - загальна система виконання додатку, що керує створенням програмних модулів, компіляцією та виконанням додатку, бере на себе функції очистки пам'ятів від об'єктів, що не використовуються додатком, відповідного до заданого алгоритму. Це дає можливість не відволікати сили на мануальну чистку пам'яті, як наприклад під час розробки на мові C++ [2];
2) **Кросплатформеність:** додатки, створені за допомогою фреймфорку ASP.NET Core, можуть бути розгорнуті як на Windows, так і на Linux-серверах, або на Mac-OS платформах [3].
3) **JIT-компіляція та профільна оптимізація:** під час компіляції програмного коду у executable файл, компілятор CLR враховує особливості конкретного процесора, щоб максимально оптимізувати код та пришвидшити його виконання на сервері, а також пропускає код, на який немає посилань у різних частинах додатку. Профільна оптимізація розширює даний потенціал, оскільки компілятор реального часу створює оптимізованих код на базі тих шляхів коду, які використовуються найчастіше. Дані фрагменти коду групуються і є розміщеними поряд із

результуючими бінарними файлами для пришвидшення доступу до них [2].

4) **Проміжне програмне забезпечення:** за допомогою проміжного програмного забезпечення запит, що надходить до сервера, комплексно обробляється перед та після взаємодії з шарами MVC-моделі, що дозволяє організувати гнучкі механізми авторизації, аутентифікації, ведення журналу, обробки помилок тощо [3].
5) **Бібліотека базових класів:** набір готового функціоналу, що полегшує розробки додатків, наприклад BSL включає в себе класи для роботи з файловою системою, відправкою email-повідомлень, обробкою помилок, роботою з колекціями даних та базою даних, тощо [2].

В якості додаткових бібліотек, що не входять до бібліотеки базових класів, було використано бібліотеку Dapper, що є об’єктно-реляційним маппером, що ретранслює результати SQL-запитів до бази даних додатку, на класи моделі, створені розробником, створюючи об’єкти або колекції об’єктів в залежності від типу запитів [4]. Це дозволяє також виконувати більш складні запити, що задіюють різні таблиці за допомогою зв’язків “один до багатьох” чи “багато до багатьох”. Через використання вручну написаних запитів, є можливість значно пришвидшити швидкодію роботи прошарку доступу до даних. Механізм динамічних параметрів, що передаються в метод Dapper, дозволяє захистити дані додатку від атак з використанням SQL-ін'єкцій.

2.2 Клієнтська частина застосунку

Для розробки front-end було обрано платформу React. Дана платформа була створена та підтримується компанією Facebook, і є однією з найбільш популярних бібліотек для розробки користувацьких інтерфейсів в веб-додатках. На відміну від Angular, React не є повноцінним фреймворком, а працює як розширення класичного JavaScript + HTML/CSS. Згідно офіційної філософії React, основними його перевагами є:

1) **Декларативність:** розробник описує не те, як має працювати компонент, а описує, як виглядає компонент у тому чи іншому стані додатку, це дуже сильно відрізняє React від класичних JavaScript додатків.
2) **Модульність:** можливість створювати інкапсульовані модулі, будувати з них складні інтерфейси, і використовувати один компонент у різних місцях, слідуючи принципу DRY.
3) **Кросплатформеність:** знаючи базу платформи React, з'являється можливість розробляти як веб, так і мобільні додатки. Бібліотека React Native майже на 100% сумісна із класичним React, що дозволяє забезпечити за необхідності експорт коду зі зменшенням операційних витрат на розробку [5].

Робота бібліотек грунтується на використанні VDOM. Віртуальна DOM (VDOM) - це концепція програмування, в якій ідеальне або "віртуальне" представлення користувацького інтерфейсу зберігається в пам'яті та синхронізується зі "справжнім" DOM за допомогою бібліотеки, такої як ReactDOM. Цей процес називається узгодженням. Такий підхід і робить API React декларативним: розробник вказує, в якому стані має перебувати користувальницький інтерфейс, а React домагається, щоб DOM відповідав цьому стану [5].

Окрім самого класичного React, було використано декілька супутніх бібліотек, які значно розширюють можливості базової бібліотеки, і дозволяють створювати адаптивні та здатні до розширення користувацькі інтерфейси. У ході розробки клієнтської частини додатку необхідно забезпечити можливості роботи зі збереження складного стану, та клієнтської маршрутизації, якщо UI має не одну, а декілька повноцінних сторінок. Для реалізації першої задачі було використано бібліотеку Redux.

За допомогою Redux, можна створити центральне клієнтське сховище даних, яке об'єднує декілька заявлених бізнес-логікою модулів [6]. Наприклад це можуть бути модулі для збереження даних про новини, події. Кожен із модулів має набір методів для зміни стану сховища, або отримання даних з сервера із супутнім їх збереженням на стороні клієнта. Ці дані залишаються доступними

користувачу під час усього періоду життєвого циклу додатку, на відміну від локального стану компоненту, який втрачається під час розмонтування [6]. Оскільки дані зберігаються централізовано, вони можуть бути використані у декількох абсолютно різних частинах додатку (кожен компонент, який потребує даних зі сховища, має підписатись на оновлення за допомогою спеціальних методів).

Друга важлива розширююча бібліотека, це бібліотека React-Router, що дозволяє забезпечити клієнтську маршрутизацію. Для реалізації даного підходу, необхідно створити роутер для маршрутів, і обернути кожен відповідний компонент сторінки у компонент маршруту. Після цього при переході за певним посиланням, бібліотека будує необхідний маршрут у строці URL-браузера [7]. Також React-Router представляє можливості для створення параметрів запиту, методи для переадресації, отримання посилань із рядка пошуку і передача її в стан певного компоненту тощо. Бібліотека підтримує можливості “лінивого завантаження”, до дозволяє пришвидшити роботу додатку та оптимізувати швидкість рендерингу програмних модулів [7].

АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Чиста архітектура

Концепція чистої архітектури - не нова, але перевірена концепція підходів до побудови систем автоматизації та розробки прикладного програмного забезпечення. Сама по собі ***“clean architecture”*** не є конкретним паттерном, а скоріше набором підходів для проектування систем для підвищення можливостей їх розвитку та розширення [8]. Це пов’язано з тим, що кожен додаток вже на початковому етапі процесу розробки матиме певну межу свого розвитку, після досягнення якої наступає так званий ***“параліч”***. Це явище означає, що компоненти системи вже не мають можливостей для адаптації та розширення функціоналу. На цьому етапі вже неможливо внести зміни в архітектуру без значних вкладень в переробку усього програмного коду / баз даних/серверів розгортання тощо.

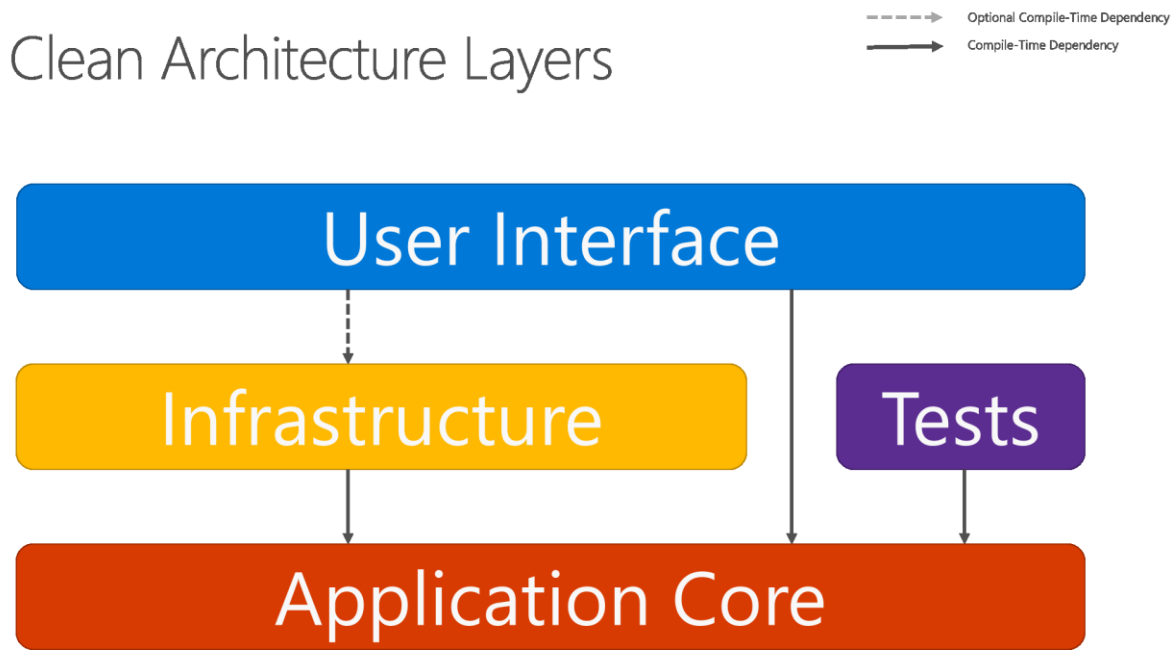


Рис. 3.1 Горизонтальна схема “чистої архітектури”

Чисту архітектуру ще називають “цибулевою архітектурою”, оскільки вона має три основний рівні, що знаходять один над одним, і мають односторонній зв’язок один з одним, з найвищого рівня, до найнижчого [9]. Перший рівень у чистій архітектурі це рівень представлення або рівень користувацького інтерфейсу. Це та частина додатку, з якою безпосередньо

взаємодіє користувач. Також до цього рівня відносять контроллери для маршрутизації запитів (у веб-додатках) [9]. Другий рівень - це рівень бізнес-логіки, або інфраструктури. Основна задача цього рівня - забезпечити функціональність бізнес-логіки застосунку, а саме у цей прошарок може бути включена логіка взаємодії з базою даних, сервіси для відправлення електронних листів, завантаження фото, певних математичних обчислень, роботи з платежами, тощо.

Важливою особливістю рівня інфраструктури є активно використання абстракцій та патерну інверсії залежностей. Суть даного патерну полягає в тому, щоб знизити зв’язаність компонентів додатку, і працювати не з конкретними класами та реалізаціями, а з їх абстракціями. Користувачу сервісу не важливо, як саме реалізована логіка, що необхідна для функціонування того чи іншого коду. Тому під час роботи із певним модулем системи, необхідно забезпечити використання абстракції замість конкретної реалізації [9]. Це значно знижує зв’язаність між різними компонентами додатку, а також при зміні логіки у класі, що реалізує інтерфейс, немає необхідності змінювати увесь програмний код, де використано службу, що була піддана модифікації. У платформі .NET (фреймворк ASP.NET Core) патерн ***“інверсія контролю”*** реалізовано у вигляді контейнера ін’єкції залежностей [3], а доступ до сервісів, що реалізують інтерфейси, відбувається через конструктори класів

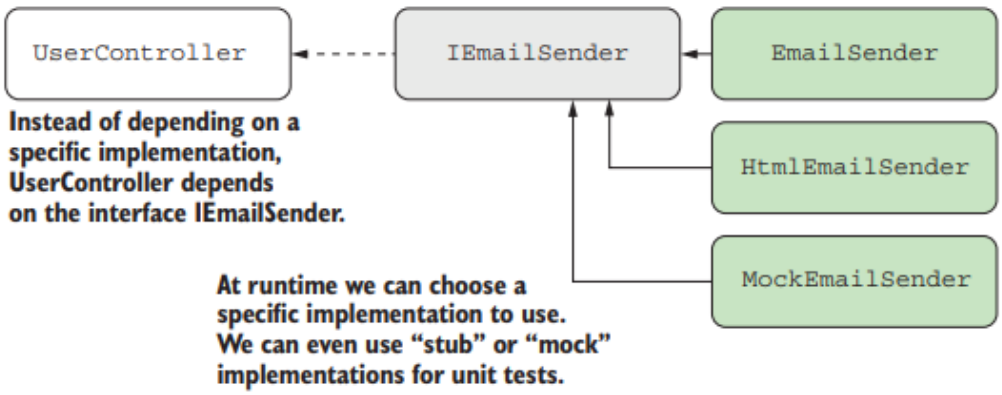


Рис. 3.2 Використання ***“IoC”*** в програмному коді додатку

Третій рівень в концепції “чистої архітектури” - це доменний рівень, або рівень ядра. До цього рівня входять такі елементи додатку, як сутності бази

даних, які використовуються для об'єктно-реляційного мапінгу, а також можуть бути реалізовані абстракції, що реалізують правила для бізнес-логіки [9]. Рівень ядра є найглибшим рівнем у ***"чистій архітектурі"***, і напряму взаємодіє лише з рівнем інфраструктури, та опціонально з фреймворком для тестування.

3.2 Паттерн "модель-представлення-контроллер"

На відміну від концепції ***"чистої архітектури"***, даний паттерн є конкретним архітектурним підходом до проектування веб-додатків, а саме їх серверних частин. MVC є найбільш класичним, перевіреним і досі актуальним підходом до організації архітектури застосунків, що був адаптованих як для мережевої REST-API, так і для класичних веб-застосунків, що відповідають за повний рендеринг усієї веб-сторінки на стороні браузера.

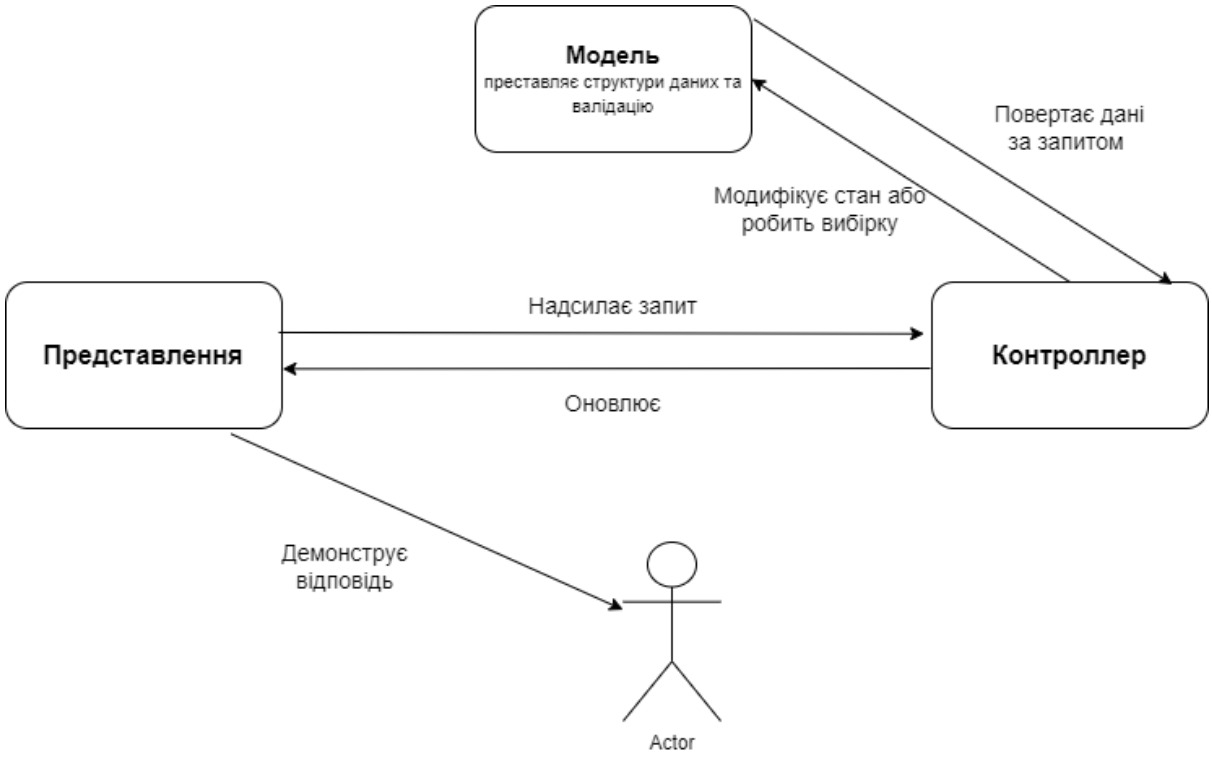


Рис. 3.3 Реалізація паттерну MVC в Web-API

Необхідно розглянути реалізацію MVC для сучасних незалежних від стану веб-служб, які є активними постачальниками даних для односторінкових додатків. У фреймворку ASP. NET Core є спеціальний проект, під час створення якого формується з використанням технології MVC максимально простий, але функціональний додаток. Схема роботи застосунку, що побудовано з використанням MVC, інтуїтивно зрозумілий.

Під час надходження HTTP-запиту з клієнтської частини застосунку, він проходить по всім позиціям проміжного програмного забезпечення, і на кінцевому етапі надходить на перший із функціональних компонентів додатку, на контролер. У застосунку може бути кілька контролерів, які відповідають за роботу з відповідними областями бізнес-логіки (наприклад Initiatives Controller відповідає за роботу з логікою, пов’язаною з ініціативами) [10]. Для отримання доступу до кінцевих точок контроллера використовується маршрутизація. У випадку із REST-API, фреймворк ASP.NET Core дозволяє використати атрибути, у яких ініціалізується маршрут і тип запиту, наприклад GET, PUT або PATCH (маршрутизація з використанням рефлексії) [3].

Після отримання запиту, контроллер з використанням механізму ін’єкції залежностей, викликає ті служби, які задекларовані у правилах бізнес-логіки, або ж методи, що взаємодіють з базою даних. У свою чергу, кожна із служб взаємодіє з моделями, що описані у відповідній функціональній складовій патерну, та змінює стан додатку відповідно до типу і даних у запиті, що надійшов на контроллер, або повертають необхідні контролеру та відповідно клієнтському застосунку дані.

У MVC модель виступає доволі абстрактним поняттям, оскільки до неї можна віднести як конкретні сутності моделі (класи), так і методи роботи з ними, а також логіку валідації моделі. Валідація - це важлива частина функціонування додатку, що відповідає за попередження зміни стану потенційно некоректними даними [10]. Наприклад, поле паролю в базі даних не може бути пустим, оскільки потенційно це порушить як безпекові політики, так і коректну роботу бізнес-логіки та логіки авторизації.

РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 База даних та рівень доступу до даних

База даних - основоположна частина будь-якого додатку, як веб-застосунку, так і мобільних або застосунків для персональних комп'ютерів. Більша частина програмного коду так чи інакше стосується саме роботи з даних, їх отримання, модифікації та обробки відповідно до заявленої бізнес-логіки. Для розробки веб-платформи було обрано підхід із використанням класичної реляційної бази даних, оскільки необхідно забезпечити логічний зв'язок між таблицями з використанням відношень "один-до-багатьох" або "багато-до-багатьох". База даних включає в себе п'ять таблиць:

1) **Події (ініціативи)**: дана таблиця включає в себе культурні події та активності, які створюють користувачі;
2) **Новини**: таблиця містить новини, що створені різними користувачами, має зв'язок з таблицею користувачів характеру "один-до-багатьох" за допомогою зовнішнього ключа UserId (один користувач може створити декілька подій);
3) **Користувачі**: дана таблиця є центральним елементом системи, оскільки вона містить дані про користувачів, включаючи чутливі, наприклад пароль. Пароль зберігається у системі у зашифрованому вигляді (хешування з додаванням унікальної для кожного користувача "солі", яку генерує відповідний сервіс рівня інфраструктури");
4) **Рефреш-токени**: дана таблиця містить довготермінові токени, що надаються користувачу для оновлення токену доступу, коли термін його дії спливає. Має зв'язок з таблицею користувачів характеру "один-до-багатьох" за допомогою зовнішнього ключа UserId, оскільки під час закінчення терміну дії токену доступу, створюється новий токен для оновлення.
5) **КористувачіІніціативи**: дана таблиця використовується для ідентифікації того, чи користувач вже підтримав ініціативу, чи ні. Це можливо забезпечити використовуючи зв'язок між таблицями

користувачів та ініціатив через дану проміжну таблицю, що зберігає унікальну пару із ідентифікаторів ініціативи та користувача, що її підтримав.

6) **Версії:** дана таблиця є службовою, і зберігає в собі інформацію про останні міграції та версії бази даних на різних етапах її функціонування

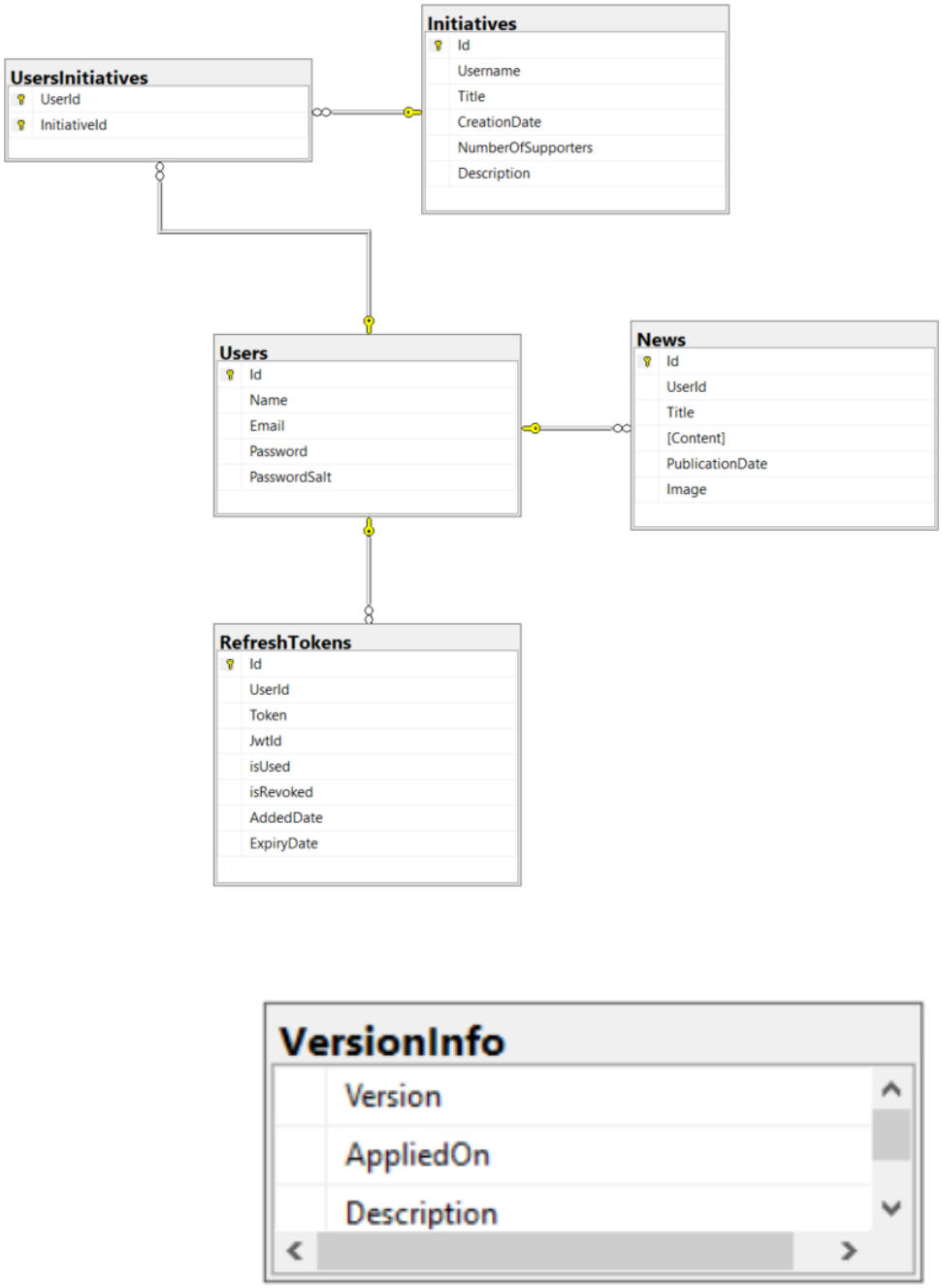


Рис. 4.1 Діаграма бази даних

Як було відмічено у розділі архітектури, основоположну роль у роботі додатків грає рівень доступу до даних, що є частиною шару “інфраструктури”. У програмному забезпеченні, що було розроблено у даній випускній кваліфікаційній роботі, рівень доступу до даних містить чотири репозиторії, що відповідають за роботу з відповідними таблицями бази даних, а також програмну логіку для реалізації міграцій з використанням підходу ***“code-first”*** [1].

1) Репозиторій ініціатив (подій): даний клас репозиторію включає методи для отримання списку усіх створених користувачами ініціатив, отримання топ десяти ініціатив за рівнем підтримки, логіку для підтримки ініціативи та перевірки чи була конкретна ініціатива підтримана даним користувачем та метод для отримання електронної пошти для відправки шаблону повідомлення підписанту ініціативи;
2) Репозиторій новин: включає в себе методи для створення та отримання списку ініціатив (з використанням механізмів пошуку новини за заголовком та пагінації новин). Для отримання інформації про пагінацію використовується відповідний клас параметрів;
3) Репозиторій токенів оновлення: включає логіку для отримання токена оновлення для його валідації, та код, що записує новостворений токен доступу до відповідної бази даних;
4) Репозиторій користувачів: містить логіку для створення нового користувача в базі даних, зміни даних про користувача, отримання інформації про юзера за адресою його пошти та за унікальним ідентифікатором в базі даних.

Друга частина рівня доступу до даних відповідає за логіку створення міграцій в базі даних та збереження інформації про стан підключення. Підхід ***“code-first”*** означає створення та зміну стану БД за допомогою спеціальних API на основі задекларованих класів шару моделі [1]. До механізму міграції відносяться відповідний метод розширення, що отримує дані про модифікації з таблиці версіонування, та проводить міграцію, а також класи міграцій, до яких включена інформація про таблиці, що мають бути створені в БД, та логіка для

відкату міграції. У файлі ***Program.cs*** проводиться сканування бази даних на наявність нових міграцій, та запускається метод розширення, що піднімає описані розробником модифікації бази. Строка підключення до сховища може бути отримана із відповідного файлу конфігурації ***appsettings. json.*** Під час створення веб-вузла, дана строка передається до екземпляру службового класу, що відповідає за підключення, та у службовий метод мігратора. Після створення екземпляру класу контексту бази даних, він використовується репозиторіями рівня доступу до даних під час усього життєвого циклу Web-API. Приклад репозиторію подано в Додатку А.

4.2 Сервіси авторизації та аутентифікації користувача, JWT

Авторизація та аутентифікація - дуже важливі і необхідні сервіси, що забезпечують як унікальну ідентифікацію користувача в системі, так і можливості для доступу юзера до тих чи інших функцій в додатку. Під час автентифікації юзер підтверджує свою особистість, а під час авторизації - свої права доступу до тих чи інших модулів додатку.

Процес аутентифікації відбувається наступним чином: користувач входить у свій аккаунт, використовуючи пару “логін-пароль”, або при відсутності інформації про користувача в базі даних, отримує можливість створити новий акаунт, ввівши ім’я або юзернейм, електронну пошту та пароль.

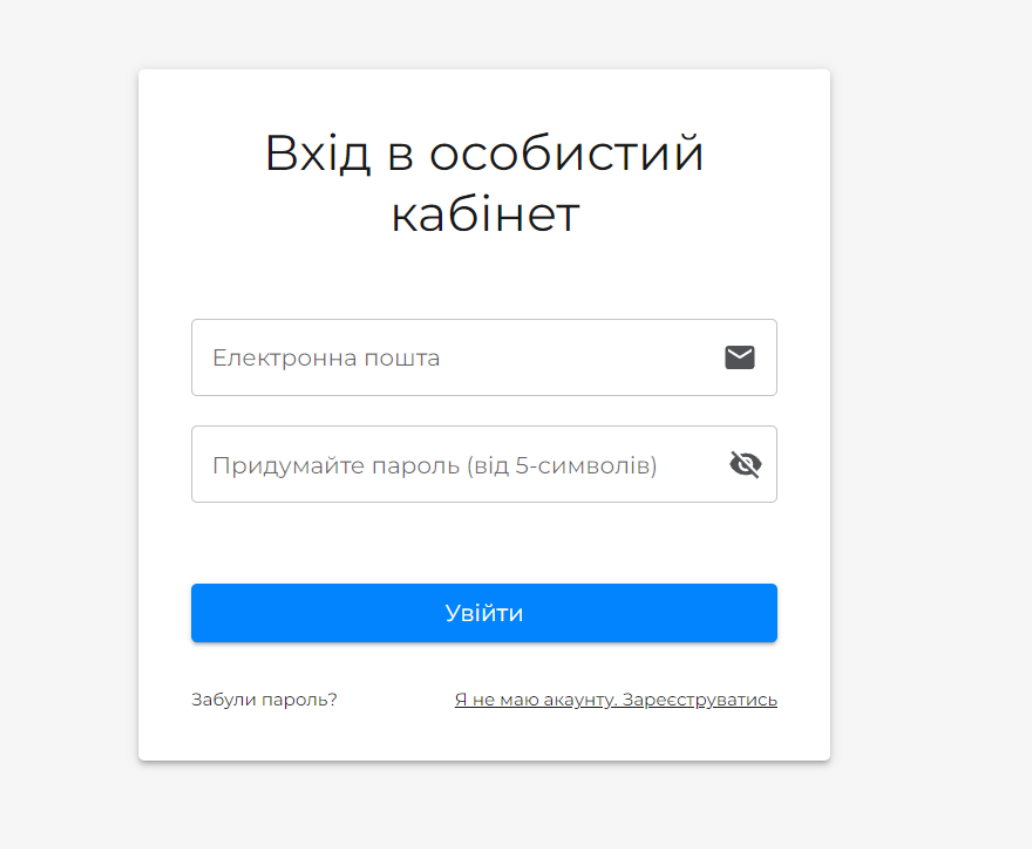


Рис. 4.3 Вікно аутентифікації користувача

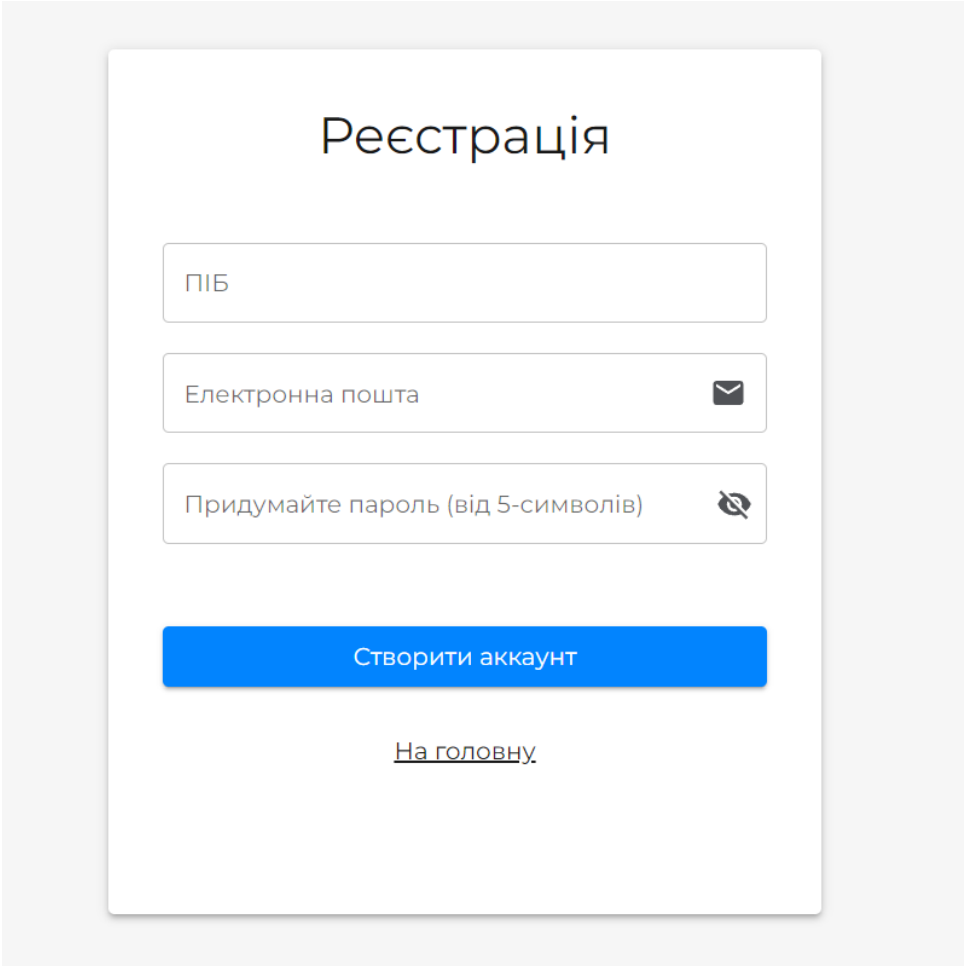


Рис. 4.4 Вікно реєстрації акаунту для нового користувача

Ці дані передаються на сервер, де служба аутентифікації-авторизації створює новий запис в сховищі, в якому містяться чутливі дані, у формі захешованої строки. У відповідь на успішну аутентифікацію, або після створення нового акаунту, користувачу повертається пара із токену доступу та токену оновлення. Для створення хеша для паролю користувача використовується алгоритм ***SHA-512***, у якому використовується унікально згенерований для кожного нового користувача ключ, відомий лише серверу. Функція хешування **SHA-512** є продовженням розвитку алгоритмів **SHA-1** та ще більш застарілого MD-4. Вона здатна генерувати 512-бітне хеш значення, і вважається більш швидкою, і більш захищеною функцією [12]. У таблиці нижче подано таблицю, що порівнює основні алгоритми хешування.

Алгоритм	Довжина хеш-суми (дайджесту повідомлення) у бітах	Розмір блоку повідомлення	Наявність колізій
MD4	128	512	Так
MD5	128	512	Так
SHA-1	160	512	Майже неймовірна, потребує складних обчислень
SHA-512	512	1024	Ні

Таблиця 4.1 Порівняльна характеристика хеш-функцій [12]

Для перевірки паролю із бази даних служба витягає даний ключ, та створює хеш для паролю, що прийшов із HTTP-запиту. Якщо згенерований хеш співпадає з тим, що вже збережено у базі даних, аутентифікація вважається пройденою.

Після успішної аутентифікації користувач отримує можливість взаємодіяти із тими частинами додатку, бізнес-правила яких вимагають унікальної ідентифікації. До них відносяться сторінки для створення ініціатив та новин, редагування особистих даних користувача тощо. Також користувач, що не пройшов аутентифікацію, не має права підтримати та підписатись на

культурну подію. Для забезпечення доступу до цих програмних модулів імплементовано систему авторизації за допомогою JWT-токенів доступу. ***JWT-токен*** - це JSON-об’єкт, що складається з трьох частин: заголовку, що містить тип токену та алгоритм хешування, корисного навантаження, що складається з так званих “заявок”: певних полів що характеризують токен та зачасту містять інформацію про користувача, необхідну клієнтській частині застосунку, а також із підпису, що створюється за допомогою ключа, що знаходиться на сервері. Після отримання токена, його необхідно передавати в заголовок запиту, щоб сервер міг валідувати токен, і перевірити чи є користувач авторизованим для отримання прав доступу. У ПЗ користувацького інтерфейсу за це відповідають методи мережевих запитів, для яких створено додаткові обгортки, що мають назву axios-перехоплювачів.

Задля підвищення рівня безпеки, користувач отримує і другий токен, рефреш-токен, або токен оновлення. Він використовується для оновлення токена доступу, оскільки останній має короткий термін життя. Це дозволяє не повторювати процеси аутентифікації декілька разів задля отримання нового JWT, і підвищує рівень безпеки, оскільки зловмисник, навіть отримавши доступ до access-токена, не зможе довго їм користуватись. Рефреш-токен є випадковою строкою чітко заданого розміру, що є унікальним для кожного користувача. Тому у сервісі авторизації-автентифікації за створення “солі” як ключа хешування та рефреш-токена може відповідати ідентична програмна логіка. Код сервісу авторизації та аутентифікації подано у Додатку Б.

Для для генерації та валідації JWT було створено окрему службу, оскільки програмний код, що використовується для цих цілей, є досить об’ємним та багаторівневим, тому його було винесено в окрему службу, яка включає в себе три окремих методи. Перший метод використовується у процесі генерації токена, для цього в ньому створюється окремий об’єкт для обробки токена, за допомогою якого створюється токен, а також серіалізується у формат String, для коректного декодування токена на стороні клієнтського інтерфейсу. Оскільки корисне навантаження токена завжди включає в себе певну кількість так званих ***“заявок”***, що містять важливу для додатка інформацію, необхідно

створити об’єкт заявок, куди записати той список заявок, які необхідно знати front-end частині застосунку [13]. Також в об’єкт дескриптора токена включається час, через який токен не буде придатний для авторизації, а також формат хешування з відповідним секретним ключем, що зберігається у файлах конфігурації сервера. Оскільки сервер використовує схему авторизації і з використанням рефреш-токенів, то у даному методі створюється відповідний токен для користувача.

Метод валідації за допомогою об’єкту параметрів валідації токена, проводить багатоетапну перевірку токена на відповідність вимогам сервера авторизації. У випадку ПЗ, що розроблялось в роботі, токен перевіряється на:

1) чи токен доступу відповідає сигнатурі JWT;
2) чи використовує токен доступу заявлений сервером авторизації алгоритм хешування;
3) чи закінчився час життя даного токена доступу;
4) чи є для токена оновлення, що прийшов із HTTP-запиту, відповідний збережений в базі даних аналог;
5) чи є токен оновлення активно використовуваним, чи можливо відмічений як той, що було викрадено зловмисниками;
6) чи не закінчився термін дії токена оновлення, що прийшов з відповідного HTTP-запиту.

Код для сервісу генерації та валідації пари токенів ***“Access-Refresh”*** подано у Додатку В.

4.3 Сервіс завантаження фото

Оскільки користувач може створити певну подію або актуальну новину власноруч, то згідно до вимог до програмного забезпечення було розроблено backend-сервіс, який дозволяє завантажити фото, що буде прикріплено до картки ініціативи в розділі “Деталі про ініціативу”. Фреймворк ASP.NET Core дозволяє використати інтерфейс IFormFile, для Blob-файлів, які завантажуються на стороні клієнта. Щоб правильно і коректно обробляти дані, що надходять, у параметрах методу контролера важливо відмітити, що дані мають бути

отримані із форми (атрибут ***[FromForm]***). В клієнтському React-додатку зображення та інші дані, які завантажує користувач, мають бути передані в створений об’єкт класу ***FormData.***

Після надходження даних із запитом типу POST, контролер викликає відповідні методи сервісу, і завантажує зображення до папки статичних файлів. У ASP.NET Core за збереження статичних файлів відповідає папка wwwroot. Для того щоб сховище файлів працювало коректно, необхідно активувати відповідне проміжне програмне забезпечення в файлі **Program.cs** (даний модуль відповідає за підключення сервісів та запуск проміжного програмного забезпечення, та формує веб-вузол для роботи всього API).

Сам сервіс містить в собі три основні елементи: метод, що записує файл з використанням байтового потоку в пам’яті, публічний метод, що формує URL, за допомогою якого клієнтський застосунок зможе завантажити зображення із статичних файлів API, а також приватний метод класу служби, що відповідає за створення шляху, за яким у постійну пам’ять комп’ютера буде записано файл. Щоб записати файл у пам’ять, необхідно створити об’єкт спеціального байтового потоку. Директива using мови C# дозволяє не очищати пам’ять від некерованих ресурсів мануально, оскільки клас FileStream, що наслідує абстрактному класу Stream, реалізує спеціальний інтерфейс IDisposable, який відповідає за процеси очищення тих некерованих ресурсів, які не можуть бути очищені збирачем сміття. Код для сервісу завантаження фото подано у додатку Г.

4.4 Сервіс розсилки електронних листів

Після того, як аутентифікований та авторизований користувач активує дію для підтримки певної культурної події, фактично підписуючись на неї, згідно до вимог до ПЗ, від має отримати на електронну пошту, вказану при реєстрації, повідомлення, що містить контактну інформацію про користувача, що створив подію, та докладний опис цієї події. Для реалізації цих вимог на

рівні інфраструктури було створено сервіс, що автоматично відправляє дане повідомлення підписанту.

Сервіс email-розсилки складається з таких службових методів:

1) перший метод, що є приватним (лише даний клас служби має до нього доступ), відповідає за створення шляху до того файлу шаблону повідомлення, який було розроблено для реакції на підпис на подію;
2) другий приватний метод відповідає за створення самого шаблону повідомлення. Він зчитує html-файл у байтовий потік за допомогою вбудованих засобів бібліотеки базових класів .NET з того шляху, який було створено попереднім методом, а потім записує його у нову строку тіла повідомлення та заміняє певні частини тексту на інформацію про підтриману ініціативу;
3) основний публічний метод, що отримує тіло повідомлення з відповідної службової функції, формує тему повідомлення, встановлює з'єднання з SMTP-сервером та за допомогою логіну і паролю до сервера, що зчитуються із файлів конфігурації додатку, проводиться аутентифікація. Якщо автентифікація успішна, повідомлення відправляється підписанту. Опціонально до повідомлення можуть бути додані певні файли, що є частиною опису ініціативи.

Процес аутентифікації до SMTP-сервера Google відбувається лише коли для вибраного для розсилки акаунту була встановлена двоетапна аутентифікація. Це пов'язано із останніми політиками безпеки Google, реалізація котрих потребує встановлення окремих унікальних паролів та дозволів для кожного додатку, що використовує цільовий акаунт для підключення до SMTP. Програмний код сервісу подано у Додатку Д.

4.5 Обробка помилок та ведення журналу

Окрім сервісів, що безпосередньо взаємодіють з користувацьким інтерфейсом і відповідають за реалізацію тих чи інших вимог бізнес-логіки, у серверах наявні певні самостійні служби, що включають програмне забезпечення, основною метою якого є виконання таких функцій, як ведення

журналу та реакція на помилки. У ASP.NET Core, це ПЗ можливо реалізувати за допомогою проміжного програмного забезпечення, що як вказувалось вище, обробляє HTTP-запити і відповіді сервера конвеєрним методом. Зв'язок між елементами конвеєра створюється за допомогою механізму виклику делегатів. Під час розробки Web-API є можливість використовувати як вже готове проміжне ПЗ, що є частиною бібліотеки класів фреймворку, так і створювати власне. У серверній частині додатку, що було розроблено у даній випускній роботі, використовується два види створеного розробником проміжного програмного забезпечення: для ведення журналу та обробки помилок.

Middleware для ведення журналу використовує декілька службових класів, створених для реалізації власного механізму ведення журналу у текстові файли, що створюються кожен день. Такий підхід необхідний, оскільки базовий провайдер для ведення журналу є консольним, тому дані журналу втрачаються після завершення життєвого циклу додатку. Щоб реалізувати власний провайдер логування, необхідно створити файл логгера, який реалізує відповідний інтерфейс (включає в себе метод-прапор для демонстрації можливості записати інформацію в журнал, та сам метод для запису), клас для створення власного провайдера логування, який створює екземпляри об'єктів класу ведення журналу, а також клас з методом для реєстрації власного провайдера у файлі створення веб-вузла.

Саме проміжне програмне забезпечення після його реєстрації, записує кожен запит до сервера в файл, який щоденно створюється під час першого HTTP-запиту від клієнтського застосунку.

```
18.04.2023 12:38:11 ; OPTIONS ; /api/auth/login ; 204
18.04.2023 12:38:13 ; POST ; /api/auth/login ; 200
18.04.2023 12:38:13 ; GET ; /api/initiatives ; 200
18.04.2023 12:38:13 ; GET ; /api/initiatives/top-initiatives ; 200
18.04.2023 12:38:13 ; GET ; /api/initiatives/top-initiatives ; 200
18.04.2023 12:38:13 ; GET ; /api/initiatives ; 200
18.04.2023 12:44:00 ; GET ; /api/news ; 200
18.04.2023 12:44:00 ; GET ; /api/news ; 200
```

Рис. 4.5 Запити до сервера в файлі журналу

Middleware для обробки помилок, в веб-додатках переважно розміщується першим в конвеєрі, оскільки це дає можливість охопити максимальну кількість можливих місць виникнення виключних ситуацій. При виникненні подібної ситуації, програмне забезпечення повертає відповідь сервера з кодом 500, що означає внутрішню помилку сервера, та додає відповідний запис в журнал, що дозволяє ідентифікувати причину помилки, і значно швидше вирішити її.

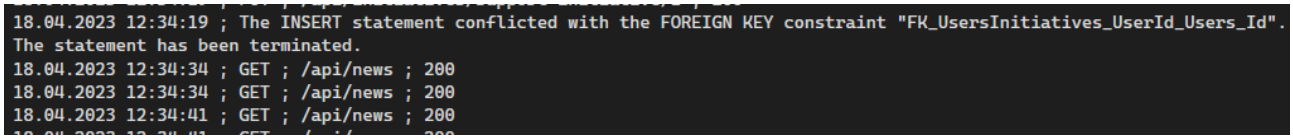


Рис. 4.6 Приклад можливої помилки в файлі журналу

Програмний код службових класів та проміжного програмного забезпечення для ведення журналу і обробки помилок подано у додатку Е.

РЕЗУЛЬТАТИ ТА ЇХ АНАЛІЗ

Після реалізації усіх вище вказаних сервісів, було створено веб-застосунок, що дозволяє поширювати останні новини культури, створювати власні новини та події, редагувати їх, а також підтримувати ініціативи інших користувачів. Оскільки проєкт має дві повнофункціональні складові частини, автором роботи представлено структуру клієнтського веб-застосунку та Web-API (сервер), що обслуговує даного клієнта.

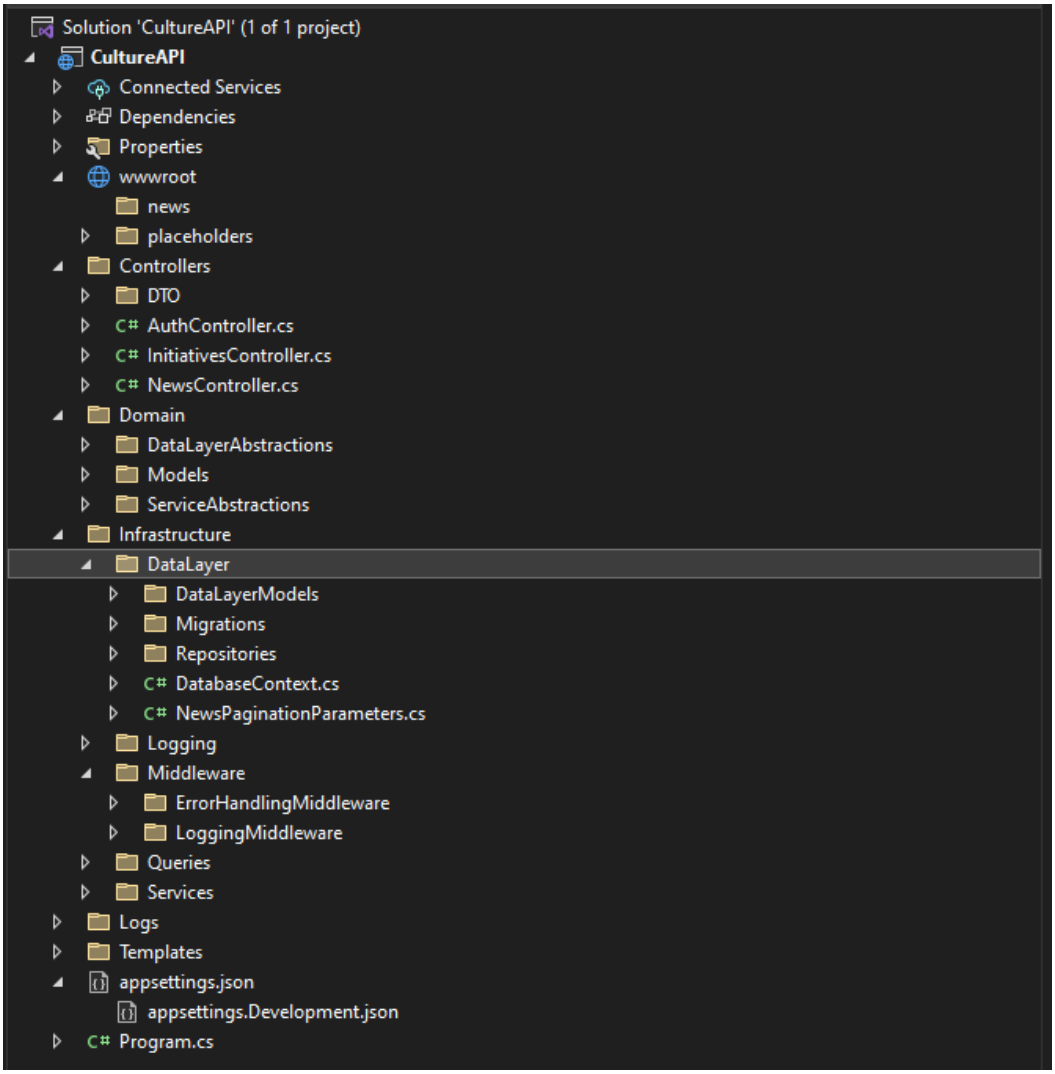


Рис. 5.1 Структура проєкту Web-API

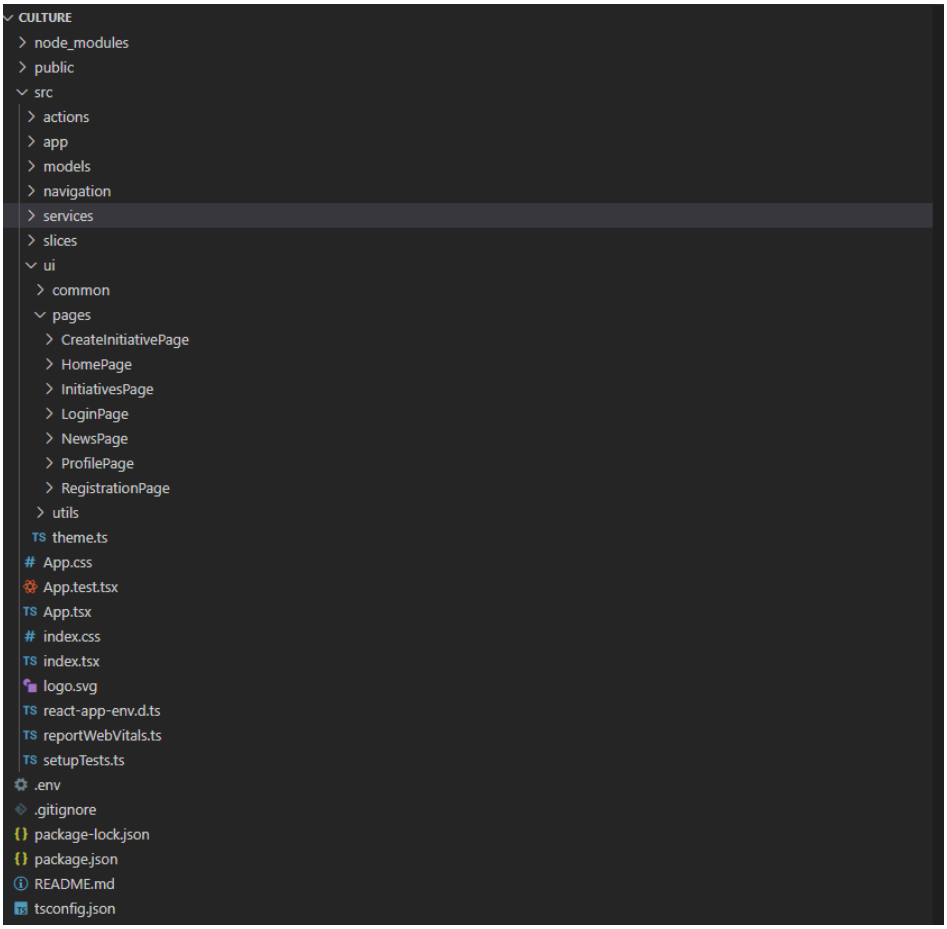


Рис. 5.2 Структура React-клієнта

Структура серверної частини використовує принципи чистої архітектури, клієнт було розділено на логічні модулі: виділено модуль користувацького інтерфейсу, сервіси для отримання даних з сервера, модуль асинхронних функцій для зміни стану додатку, модуль центрального сховища, роутер маршрутизації тощо.

Головна сторінка додатку представляє користувачу можливість навігації до різних сторінок додатку. Якщо користувач був авторизований, він має відмінну від стандартної домашню сторінку. Для ідентифікаційного користувача існує можливість перейти на сторінку реєстрації.

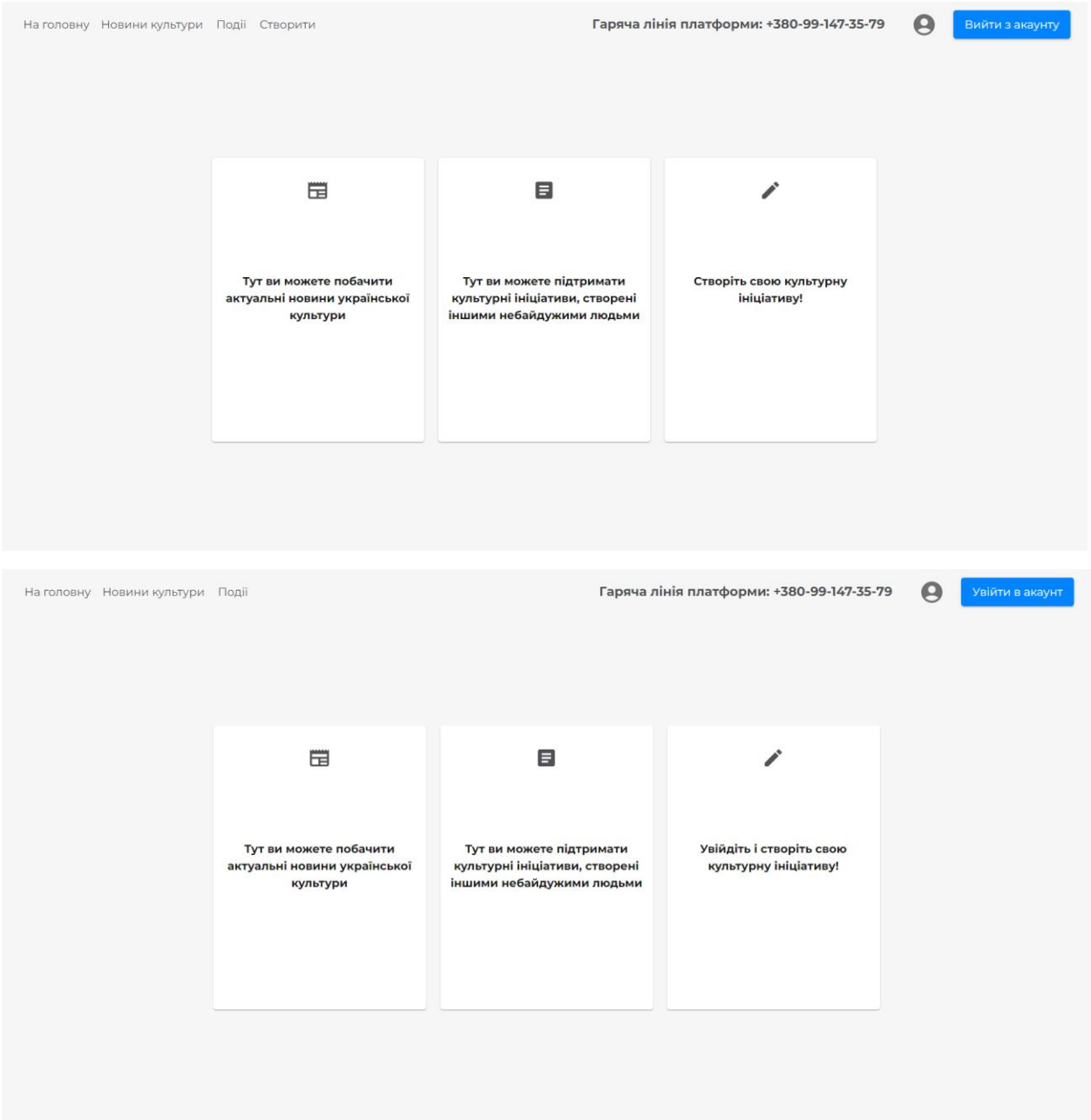


Рис. 5.3 - 5.4 Домашня сторінка користувача, що залогінився або не залогінився в систему

Вікно авторизації та створення акаунту згідно до вимог має клієнтську валідацію. Користувач не має можливості ввести пароль менше п’яти символів, або відправити на сервер запит з пустим тілом. Наявна валідація для електронної пошти, що перевіряє формат на коректність. Неможливо створити пароль розміром менше п’яти символів, ця умова може бути легко змінена на іншу, якщо дана політика для пароля користувача буде недостатньою. При реєстрації користувача наявна умова, що ПІБ має містити в собі лише символи алфавіту.

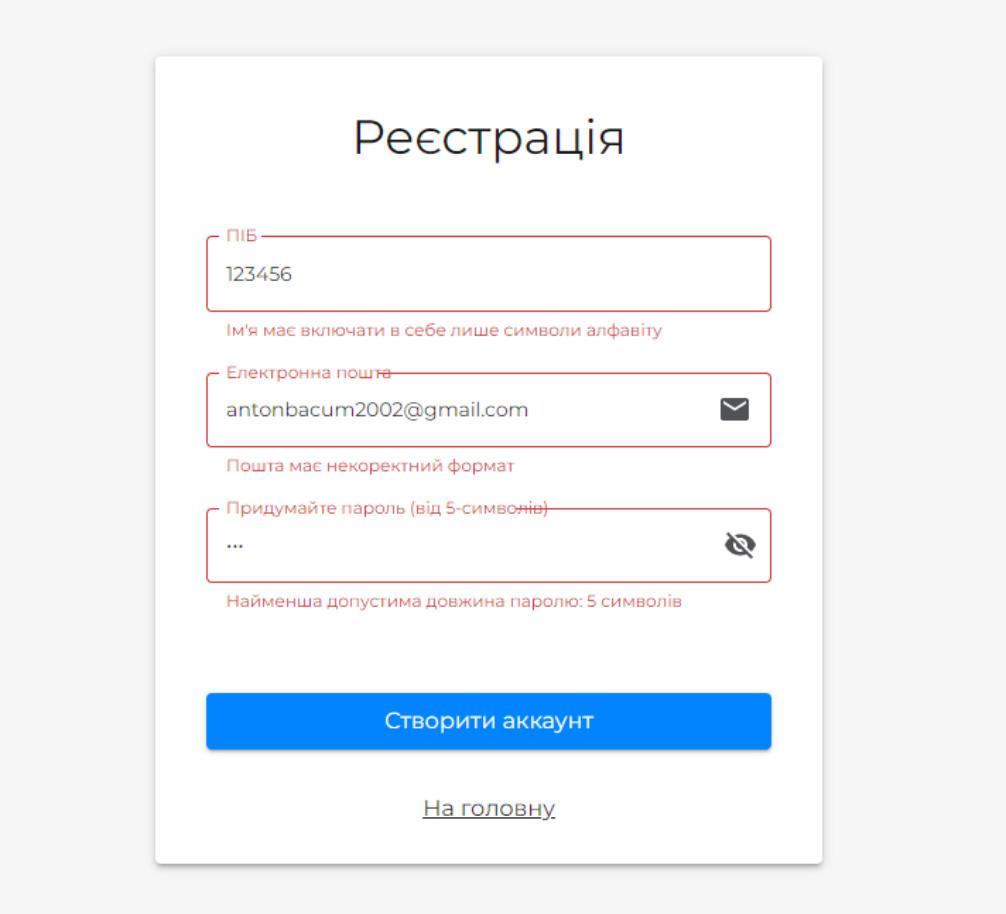


Рис. 5.5 Приклад клієнтської валідації

На сторінці ініціатив користувач має можливість отримати список створених іншими користувачами подій, та підписатись на певну подію, і потім отримати на пошту повідомлення з вичерпною інформацією про подію, на яку було оформлено підписку. Розглянемо даний користувацький кейс на прикладі тестової події, поданої нижче:

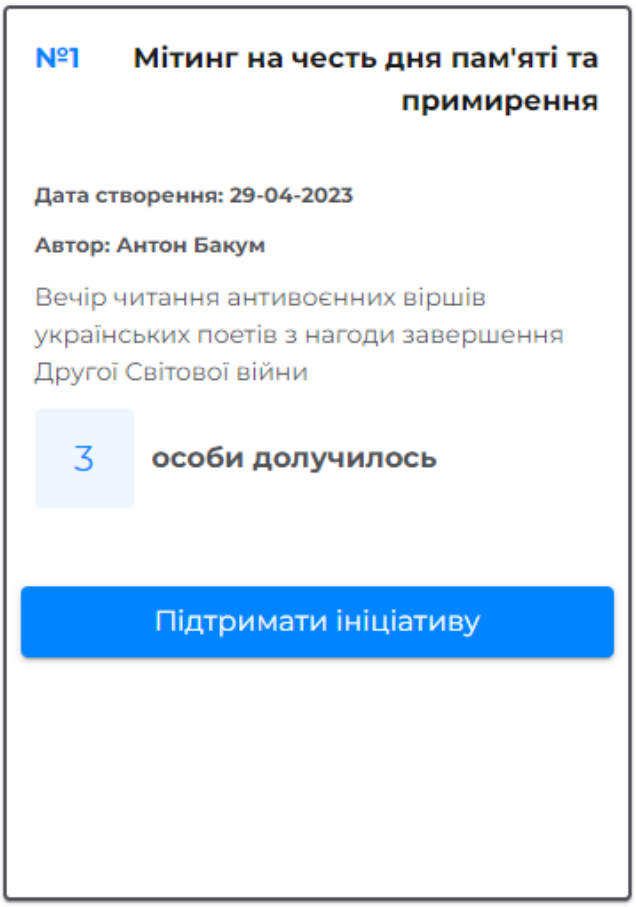
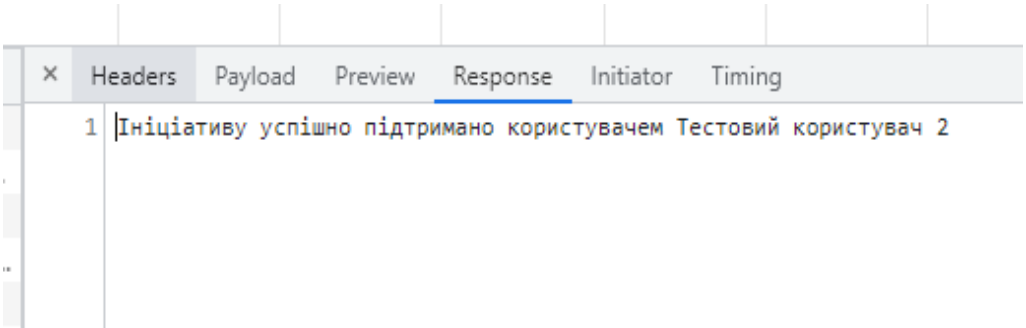
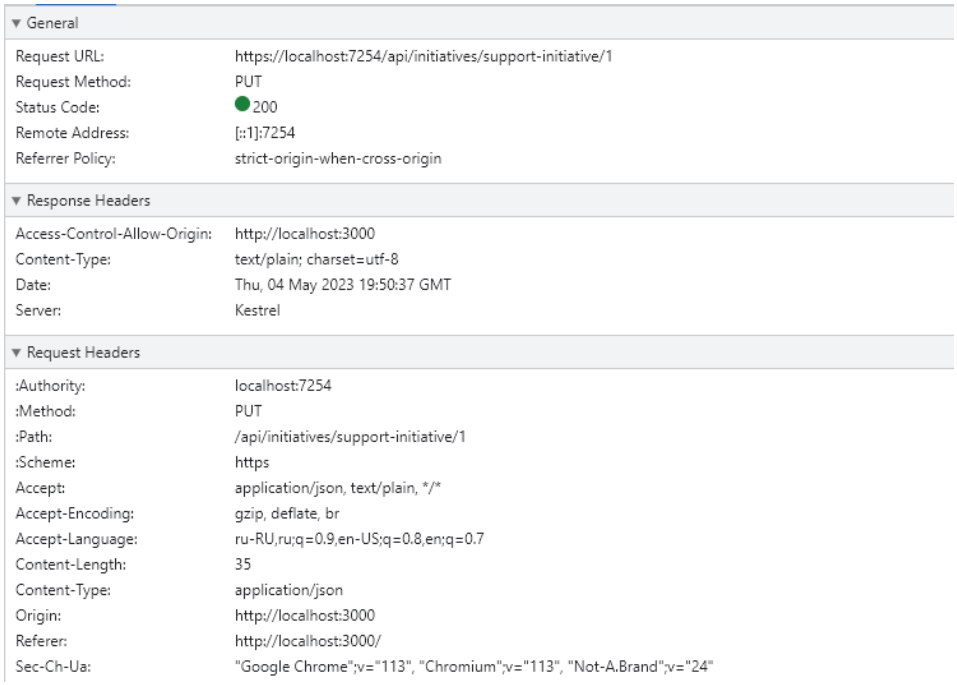


Рис. 5.6 Тестова картка ініціативи

Картка ініціативи включає в себе інформацію про ініціативу, її опис, дату створення, автора та короткий заголовок культурної події, а також кількість людей що долучились до події. Якщо ініціатива успішно підтримана, кількість підписантів збільшується на одиницю, що можна перевірити за допомогою аналізу відповіді сервера:





▼ General

Request URL:	https://localhost:7254/api/initiatives/support-initiative/1
Request Method:	PUT
Status Code:	200
Remote Address:	[::1]:7254
Referrer Policy:	strict-origin-when-cross-origin

▼ Response Headers

Access-Control-Allow-Origin:	http://localhost:3000
Content-Type:	text/plain; charset=utf-8
Date:	Thu, 04 May 2023 19:50:37 GMT
Server:	Kestrel

▼ Request Headers

:Authority:	localhost:7254
:Method:	PUT
:Path:	/api/initiatives/support-initiative/1
:Scheme:	https
Accept:	application/json, text/plain, */*
Accept-Encoding:	gzip, deflate, br
Accept-Language:	ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
Content-Length:	35
Content-Type:	application/json
Origin:	http://localhost:3000
Referer:	http://localhost:3000/
Sec-Ch-Ua:	"Google Chrome";v="113", "Chromium";v="113", "Not-A.Brand";v="24"

Рис. 5.7-5.8 Результат запиту на підтримку ініціативи

Якщо користувач матиме бажання відправити запит знову, сервер повертає помилку з поясненням, що дана ініціатива вже була підтверджена користувачем

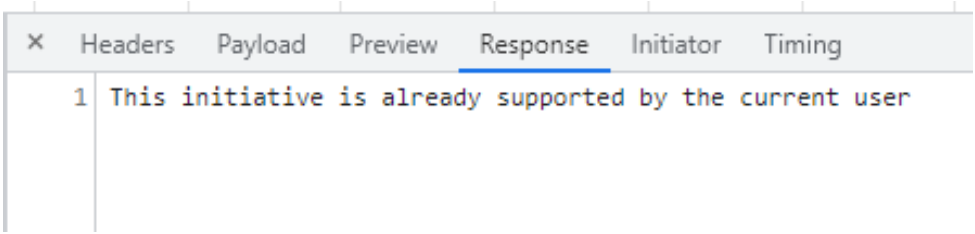


Рис. 5.9 Результат повторної спроби підтримати ініціативу

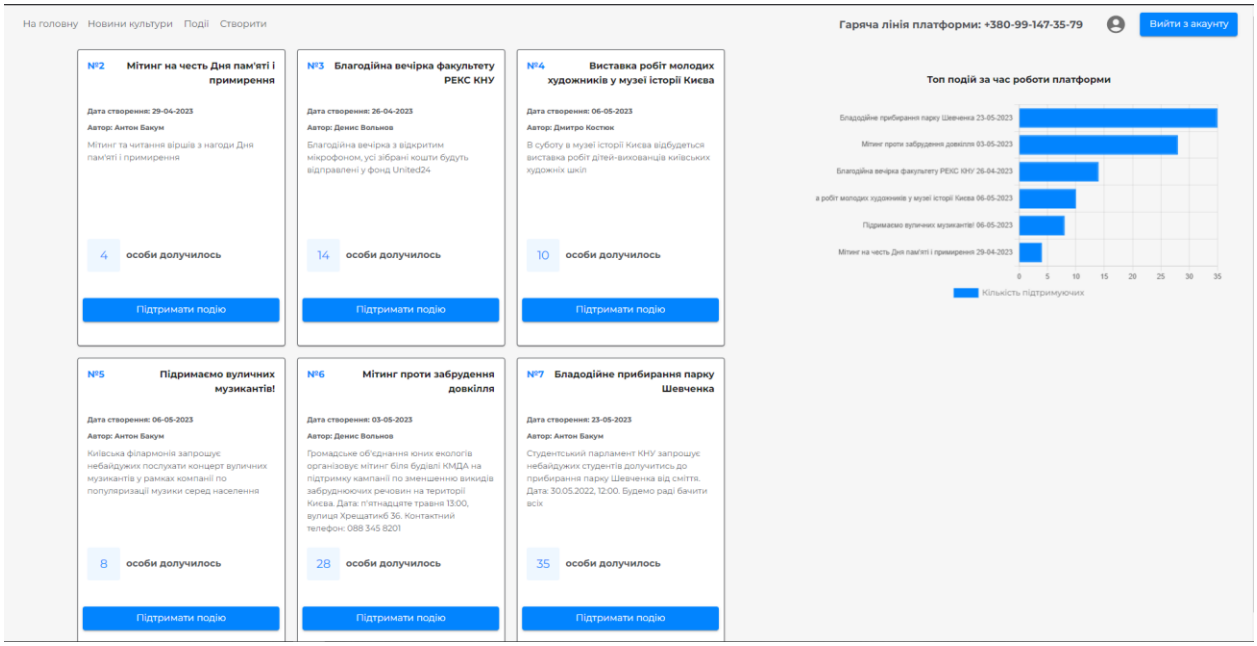


Рис 5.10 Загальний вигляд сторінки ініціатив

Користувач може побачити створені ним ініціативи на сторінці свого профілю, який зберігає основні дані про користувача в системі: юзернейм, прізвище та ім’я, телефон та електронну пошту користувача, а також фото його профілю. Оскільки при реєстрації телефон вказувати не потрібно, то за замовчуванням відповідне поле є пустим, і юзер може в майбутньому додати контактний телефон до своїх особистих даних.

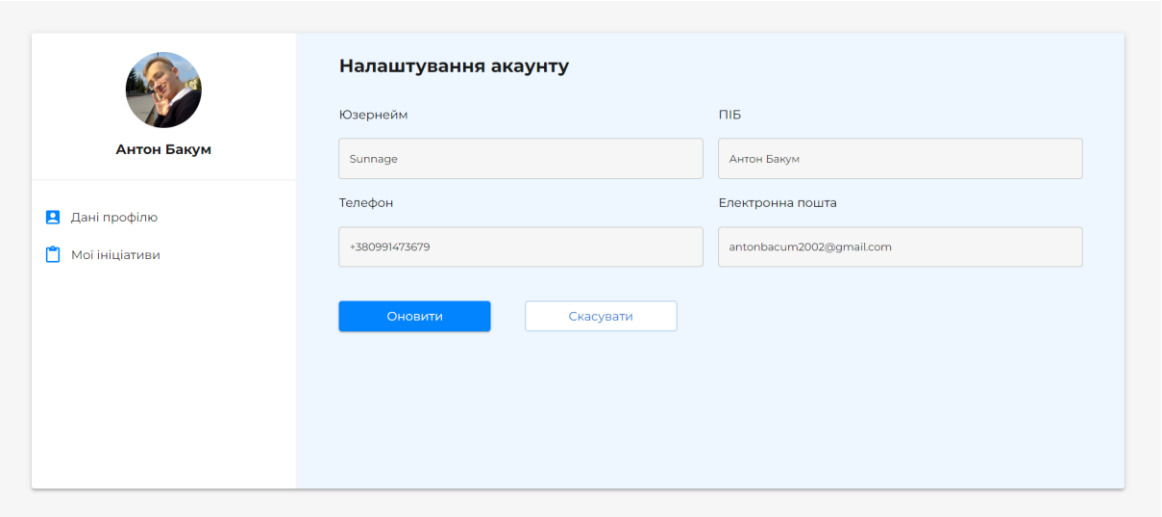


Рис. 5.11 Налаштування профілю користувача

Користувач також має доступ до історії власних ініціатив, які розміщені у вигляді карток на відповідній вкладці сторінки профілю. Він може видалити створену власноруч подію, або редагувати її натисканням на відповідну кнопку.

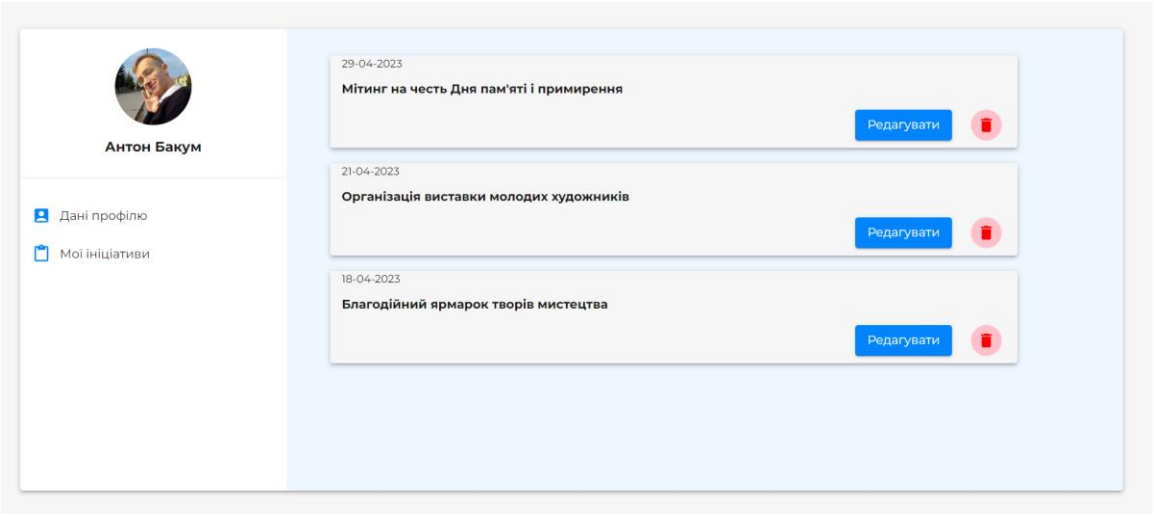


Рис 5.12 Список ініціатив, створених користувачем

Сторінка новин представляє собою інтуїтивно зрозумілий інтерфейс, де користувач може побачити список наявних новин та здійснювати пошук новини за її заголовком, також є можливість підвантажити додаткові новини. За

замовчуванням при завантаженні сторінки з сервера відвантажуються вісім новин. Сторінка новин платформи має такий вигляд (були додані новини із стандартним зображенням для тестування, у майбутньому при використанні платформи користувачі матимуть можливість створювати новини на основі власних фото):

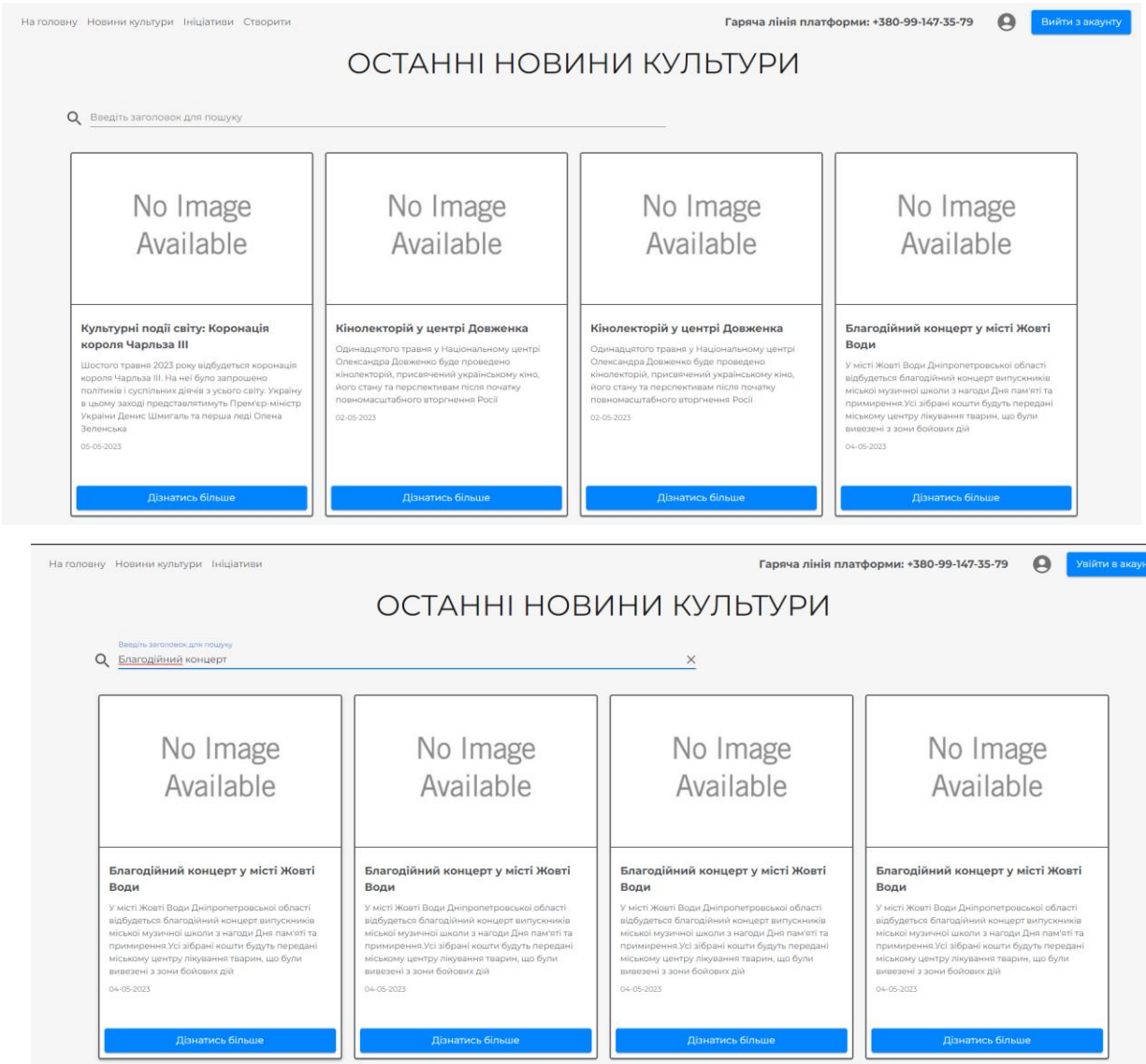


Рис. 5.13 - 5.14 Сторінка новин та механізм пошуку

Механізм пошуку дозволяє знайти новину за її заголовком, для цього створене спеціальне поле пошуку.

ВИСНОВКИ

В ході виконання даної дипломної роботи було проаналізовано наявні приклади розробленого програмного забезпечення, яке забезпечує обслуговування потреб населення у вираженні громадської думки, враховано даний досвід, на основі якого розроблено прототип веб-застосунку, що вирішує подібні задачі, але не в сфері громадського управління, а забезпечує реалізацію потреб громадянського суспільства в області культури та мистецтва.

Було проаналізовано найбільш поширені архітектурні підходи для реалізації даного прототипу, та відібрано ті, що найкраще підходять під ті вимоги, що були розроблені для даного програмного забезпечення.

Відповідно до заявлених вимог було розглянуто сучасні методи для захисту даних та безпеки інформації, і вибрано ті, які найкраще відповідають необхідним вимогам до зберігання чутливих даних користувачів даного прототипу.

Було розроблено програмне забезпечення для даного прототипу веб-застосунку, з урахуванням обраної архітектури та створено сервіси, що є необхідними для коректної роботи програмного забезпечення. Було проведено його тестування на основі технічних даних, що створені з метою відпрацювання кейсів користувацьких вимог. Отримані результати показують, що створений прототип веб-застосунку виконує покладені на нього задачі, а його функціонал має здатність до розвитку у майбутньому.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] Про розробку та здійснення проєкту щодо порятунку річки Рось з подальшим використанням його для відновлення водності всіх рік України та їх приток – малих річок. Петиція до Президента України № 22/170066-еп [Електронний ресурс] - Режим доступу до ресурсу: https://petition.president.gov.ua/petition/170066.

[2] CLR via C# (Developer Reference) 4th Edition / J.Richter - Microsoft Press, 2012 (4th edition). - с. 554-606;

[3] ASP.NET Core in Action. Second Edition/A.Lock - Manning, 2021.

[4] Getting started with Dapper [Електронний ресурс] - Режим доступу до ресурсу: https://www.learndapper.com/.

[5] React tutorial and official documentation [Електронний ресурс] - Режим доступу до ресурсу: https://ru.legacy.reactjs.org/docs/getting-started.html.

[6] Redux Essentials, Part 1: Redux Overview and Concepts [Електронний ресурс] - Режим доступу до ресурсу: https://redux.js.org/tutorials/essentials/part-1-overview-concepts.

[7] React Router main concepts [Електронний ресурс] - Режим доступу до ресурсу: https://reactrouter.com/en/main/start/concepts.

[8] Martin, R.C. Clean Architecture: A Craftsman's Guide to Software Structure and Design/ R.C. Martin. - Robert C. Martin Series, 2017.

[9] Architecting Modern Web Applications with ASP.NET Core and Microsoft Azure/S. “Ardalis” Smith - Microsoft Developer Division, Redmond, Washington 98052-6399, 2022. - с. 23-27;

[10] Pro ASP.NET Core MVC 6th edition/A.Freeman - Apress, 2016. - с.56-65;

[11] Programming Entity Framework: Code First/ J. Lerman, R. Miller - O`Reilly, 2011. - с. 2-4.

[12] Analysis of Secure Hash Algorithm (SHA) 512 for Encryption Process on Web Based Application/M.Sumagita, I.Riadi - International Journal of Cyber-Security and Digital Forensics (Vol.7, Issue 4), 2018.

[13] JSON Web Token Structure [Електронний ресурс] - Режим доступу до ресурсу: https://auth0.com/docs/secure/tokens/json-web-tokens/json-web-token-structure

Додаток А

Репозиторій подій рівня доступу до даних

```
public class InitiativesRepository: IInitiativesReporitory
  {
    private readonly IDatabaseContext _context;
    private readonly ILogger<InitiativesRepository> _logger;

    public InitiativesRepository (IDatabaseContext connection, ILogger<InitiativesRepository> logger)
    {
      _context = connection;
      _logger = logger;
    }
    public async Task<IEnumerable<Initiative>> GetAllInitiatives()
    {
      return await _context.SqlConnection.QueryAsync<Initiative>(InitiativesQueries.getAllInitiatives);
    }

    public async Task<int> CreateInitiative(InitiativeDTO initiative)
    {
      int newId = await _context.SqlConnection.ExecuteScalarAsync<int>(InitiativesQueries.addInitiative, new
        {
          initiative.Username,
          initiative.Title,
          initiative.CreationDate,
          initiative.NumberOfSupporters,
          initiative.Description,
        });
      return newId;
    }

    public async Task<IEnumerable<Initiative>> GetTopInitiatives()
    {
```

```
        return await _context.SqlConnection.QueryAsync<Initiative>(InitiativesQueries.getTopInitiatives);
    }

    public async Task SupportInitiative(int id, SupportInitiativeDTO supportInitiative)
    {
        try
        {
            using (TransactionScope scope = new TransactionScope(TransactionScopeAsyncFlowOption.Enabled))
            {
                await _context.SqlConnection.ExecuteAsync(
                    UsersInitiativesQueries.
                    AddSupportedInitiative, new { InitiativeId = id, supportInitiative.UserId }).
                    ConfigureAwait(false);
                await _context.SqlConnection.
                     ExecuteAsync(InitiativesQueries.updateNumberOfSupporters, new { Id = id, supportInitiative.NumberOfSupporters })
                     .ConfigureAwait(false);
                scope.Complete();
            }
        }
        catch (TransactionAbortedException ex)
        {
            _logger.LogError($"{DateTime.Now} ; {ex.Message}");
            throw;
        }
    }

    public async Task<int> CheckSupportedInitiative(int initiativeId, int userId)
    {
        return await _context.SqlConnection.QuerySingleAsync<int>(UsersInitiativesQueries.CheckSupportedInitiative,
            new { InitiativeId = initiativeId, UserId = userId });
    }
```

```
    public async Task<Initiative> GetInitiativeForEmail(int initiativeId)
    {
        return await _context.SqlConnection.QuerySingleAsync<Initiative>(InitiativesQueries.getInitiativeForEmail, new { Id = initiativeId });
    }

    public async Task<int> UpdateInitiative(int initiativeId, InitiativeUpdateDTO initiative)
    {
        return await _context.SqlConnection.ExecuteAsync(InitiativesQueries.updateInitiative,
            new { Id = initiativeId, initiative.Username, initiative.Title, initiative.Description, });
    }
  }
```

Додаток Б

Сервіс аутентифікації та авторизації

```
public class AuthService: IAuthService
  {
    private static Random random = new Random();

    public string CreatePasswordHash(string userPassword, string userSalt)
```

```
    {
        using (var hmac = new HMACSHA512())
        {
            hmac.Key = Encoding.UTF8.GetBytes(userSalt);
            byte[] hash = hmac.ComputeHash(Encoding.UTF8.GetBytes(userPassword));
            return Convert.ToBase64String(hash);
        }
    }
    public bool CheckPasswordHash(string incomingPassword, string userHash, string userSalt)
    {
        using (var hmac = new HMACSHA512())
        {
            hmac.Key = Encoding.UTF8.GetBytes(userSalt);
            byte[] hashForCheck =
hmac.ComputeHash(Encoding.UTF8.GetBytes(incomingPassword));
            return userHash.SequenceEqual(Convert.ToBase64String(hashForCheck));
        }
    }

    DateTime UnixTimeStampToDateTime (long unixTimeStamp)
    {
        var dateTimeValue = new DateTime(1970, 1, 1, 0, 0 ,0, 0, DateTimeKind.Utc);
        dateTimeValue.AddSeconds(unixTimeStamp).ToUniversalTime();
        return dateTimeValue;
    }
    public string GenerateRandomString(int length)
    {
        const string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
        return new string(Enumerable.Repeat(chars, length)
            .Select(s => s[random.Next(s.Length)]).ToArray());
    }
}
```

Додаток В

Сервіс валідації та генерації токенів доступу та оновлення

```
public class JwtGenerationService: IJwtGenerationService
  {
    private readonly IAuthService _authService;
    private readonly IConfiguration _configuration;
    private readonly IUsersRepository _usersRepository;
    private readonly IRefreshTokensRepository _refreshTokensRepository;
    //private readonly TokenValidationParameters _tokenValidationParameters;

    public JwtGenerationService(IConfiguration configuration, IAuthService authService,
      IRefreshTokensRepository refreshTokensRepository, IUsersRepository usersRepository)
    {
      _configuration = configuration;
```

```
        //_tokenValidationParameters = validationParameters;
        _authService = authService;
        _refreshTokensRepository = refreshTokensRepository;
        _usersRepository = usersRepository;
    }
    public async Task<AuthResult> GenerateJwtToken(string email, int id)
    {
        var jwtTokenHandler = new JwtSecurityTokenHandler();
        var secretTokenKey =
Encoding.ASCII.GetBytes(_configuration.GetSection("JwtConfig:Secret").Value);

        var tokenDescriptor = new SecurityTokenDescriptor()
        {
            Subject = new ClaimsIdentity(
             new[]
             {
                new Claim("Id", id.ToString()),
                new Claim(JwtRegisteredClaimNames.Sub, email),
                new Claim(JwtRegisteredClaimNames.Email, email),
                new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
                new Claim(JwtRegisteredClaimNames.Iat,
DateTime.Now.ToUniversalTime().ToString()),
             }),

            Expires =
DateTime.UtcNow.Add(TimeSpan.Parse(_configuration.GetSection("JwtConfig:ExpiryTimeFrame
").Value)),
            SigningCredentials = new SigningCredentials(new
SymmetricSecurityKey(secretTokenKey), SecurityAlgorithms.HmacSha256)
        };
        var token = jwtTokenHandler.CreateToken(tokenDescriptor);
        string serializedToken = jwtTokenHandler.WriteToken(token);

        RefreshToken refreshToken = new RefreshToken()
        {
            JwtId = token.Id,
```

```
            Token = _authService.GenerateRandomString(20),
            AddedDate = DateTime.UtcNow,
            ExpiryDate = DateTime.UtcNow.AddMonths(6),
            isRevoked = false,
            isUsed = false,
            UserId = id
        };

        await _refreshTokensRepository.AddRefreshToken(refreshToken);

        return new AuthResult()
        {
            Token = serializedToken,
            RefreshToken = refreshToken.Token,
            Result = true,
        };
    }

    public async Task<AuthResult> ValidateAndGenerateToken(TokenRequestDTO
tokenRequest)
    {
        var jwtTokenHandler = new JwtSecurityTokenHandler();
        var secretTokenKey =
Encoding.ASCII.GetBytes(_configuration.GetSection("JwtConfig:Secret").Value);
        var tokenValidationParameters = new TokenValidationParameters()
        {
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new SymmetricSecurityKey(secretTokenKey),
            ValidateIssuer = false,
            ValidateAudience = false,
            RequireExpirationTime = true,
            ValidateLifetime = false,
        };
        try
        {
            var tokenInVerification = jwtTokenHandler.
```

```
            ValidateToken(tokenRequest.Token, tokenValidationParameters, out var
validatedToken);
        if (validatedToken is JwtSecurityToken jwtSecurityToken)
        {
            var result = jwtSecurityToken.Header.Alg.Equals(SecurityAlgorithms.HmacSha256,
                StringComparison.InvariantCultureIgnoreCase);

            if (!result)
            {
                return null;
            }
        }

        var utcExpiryDate = long.Parse(tokenInVerification.Claims.
            FirstOrDefault(x => x.Type == JwtRegisteredClaimNames.Exp).Value);

        var expiryDate = UnixTimeStampToDateTime(utcExpiryDate);
        if (expiryDate > DateTime.Now)
        {
            return new AuthResult()
            {
                Result = false,
                Errors = new List<string>()
                {
                    "Tokens are expired"
                }
            };
        }

        RefreshToken storedToken = await _refreshTokensRepository.GetToken(tokenRequest);

        if (storedToken == null)
        {
            return new AuthResult()
            {
                Result = false,
```

```
            Errors = new List<string>()
            {
                "Invalid tokens, token = null",
            }
        };
    }

    if (storedToken.isUsed)
    {
        return new AuthResult()
        {
            Result = false,
            Errors = new List<string>()
            {
                "Invalid tokens",
            }
        };
    }

    if (storedToken.isRevoked)
    {
        return new AuthResult()
        {
            Result = false,
            Errors = new List<string>()
            {
                "Invalid tokens",
            }
        };
    }

    var jti = tokenInVerification.Claims.FirstOrDefault(claim => claim.Type ==
JwtRegisteredClaimNames.Jti).Value;

    if (storedToken.JwtId != jti)
    {
```

```
                return new AuthResult()
                {
                    Result = false,
                    Errors = new List<string>()
                    {
                        "Invalid tokens",
                    }
                };
            }

            if (storedToken.ExpiryDate < DateTime.UtcNow)
            {
                return new AuthResult()
                {
                    Result = false,
                    Errors = new List<string>()
                    {
                        "Tokens are expired",
                    }
                };
            }

            TokenUserModel tokenUser = await
_usersRepository.GetUserById(storedToken.UserId);
            await _refreshTokensRepository.UpdateTokenStatus(storedToken.Token, true);

            return  await GenerateJwtToken(tokenUser.Email, tokenUser.Id);
        }
        catch (Exception e)
        {

            return new AuthResult()
            {
                Result = false,
                Errors = new List<string>()
                {
```

```
                "ServerError"
            }
        };
    }
}

DateTime UnixTimeStampToDateTime(long unixTimeStamp)
{
    var dateTimeValue = new DateTime(1970, 1, 1, 0, 0, 0, 0, DateTimeKind.Utc);
    dateTimeValue.AddSeconds(unixTimeStamp).ToUniversalTime();
    return dateTimeValue;
}

}
```

Додаток Г

Сервіс завантаження фото

```
public class FileService: IFileService
  {
     private readonly IWebHostEnvironment _environment;

     private readonly IConfiguration _configuration;

     public FileService(IWebHostEnvironment environment, IConfiguration configuration)
     {
        _environment = environment;
        _configuration = configuration;
     }

     private string CreateFilePath(string title, string id)
     {
        string staticFilesPath = _environment.WebRootPath;
        string formattedString = title.Replace(" ", "-");
        string uniqueIdentifier = Guid.NewGuid().ToString();
        return Path.Combine(staticFilesPath, $"news/{id}-{uniqueIdentifier}-
{formattedString}.png");
     }
     public async Task UploadFile (IFormFile file, string title, string id)
     {
        using (var fileStream = new FileStream(CreateFilePath(title, id), FileMode.Create))
        {
           await file.CopyToAsync(fileStream);
        }
     }

     public string CreateUrl(string title, string id)
     {
        string staticFilesPath = _configuration.GetSection("AppURLs:BaseUrl").Value;
```

```
            string formattedString = title.Replace(" ", "-");
            string uniqueIdentifier = Guid.NewGuid().ToString();
            return Path.Combine(staticFilesPath, $"news/{id}-{uniqueIdentifier}-
{formattedString}.png");
        }
    }
```

Додаток Д

Сервер розсилки електронних листів

```
public class MailService : IMailService
   {
      private readonly MailSettings _mailSettings;
      public MailService(IOptions<MailSettings> mailSettings)
      {
         _mailSettings = mailSettings.Value;
      }

      private string GetMailTemplate(string templatePath, InitiativeRequest request)
      {
         StreamReader templateReader = new StreamReader(templatePath);
         string messageText = templateReader.ReadToEnd();
         templateReader.Close();
         messageText.Replace("[Username]", request.Username)
            .Replace("[Description]", request.Description)
            .Replace("[Title]", request.Title);
         return messageText;
      }

      private string TemplatePathBuilder(string template) =>
Path.Combine(Directory.GetCurrentDirectory(), $"Templates//{template}.html");

      public async Task SendMailAsync(Email mailRequest, InitiativeRequest request)
      {
         var email = new MimeMessage();
         email.Sender = MailboxAddress.Parse(_mailSettings.Mail);
         email.To.Add(MailboxAddress.Parse(mailRequest.ToEmail));
         email.Subject = $"{mailRequest.Template} {request.Title} {request.Username}";
         var builder = new BodyBuilder();
         if (mailRequest.Attachments != null)
         {
            byte[] fileBytes;
            foreach (var file in mailRequest.Attachments)
            {
               if (file.Length > 0)
```

```
                {
                    using (var ms = new MemoryStream())
                    {
                        file.CopyTo(ms);
                        fileBytes = ms.ToArray();
                    }
                    builder.Attachments.Add(file.FileName, fileBytes,
ContentType.Parse(file.ContentType));
                }
            }
        }
        string test = TemplatePathBuilder(mailRequest.Template);
        builder.HtmlBody = GetMailTemplate(TemplatePathBuilder(mailRequest.Template),
request);
        email.Body = builder.ToMessageBody();
        using var smtp = new SmtpClient();
        smtp.Connect(_mailSettings.Host, _mailSettings.Port, SecureSocketOptions.StartTls);
        smtp.Authenticate(_mailSettings.Mail, _mailSettings.Password);
        await smtp.SendAsync(email);
        smtp.Disconnect(true);
    }
}
```

Додаток Е

Проміжне програмне забезпечення для обробки помилок та ведення журналу

```
public class ErrorHandlingMiddleware
{
    private readonly RequestDelegate _next;
    private readonly ILogger<ErrorHandlingMiddleware> _logger;
```

```
    public ErrorHandlingMiddleware(RequestDelegate next, ILogger<ErrorHandlingMiddleware>
logger)
    {
        _next = next;
        _logger = logger;
    }

    public async Task InvokeAsync(HttpContext context)
    {
        try
        {
            await _next(context);
        }
        catch (Exception exp)
        {
            _logger.LogError($"{DateTime.Now} ; {exp.Message}");
            await HandleExceptionAsync(context, exp);
        }
    }
    private async Task HandleExceptionAsync(HttpContext context, Exception exception)
    {
        context.Response.ContentType = "application/json";
        context.Response.StatusCode = (int)HttpStatusCode.InternalServerError;
        await context.Response.WriteAsync(new ErrorDetails
        {
            StatusCode = context.Response.StatusCode,
            Message = "Internal Server Error."
        }.ToString());
    }
  }

public class LoggingMiddleware
  {
    private readonly RequestDelegate _next;
    private readonly ILogger<LoggingMiddleware> _logger;
```

```
    public LoggingMiddleware(RequestDelegate next, ILogger<LoggingMiddleware> logger)
    {
        _next = next;
        _logger = logger;
    }

    public async Task InvokeAsync(HttpContext context)
    {
        try
        {
            await _next(context);
        }
        finally
        {
            _logger.LogInformation($"{DateTime.Now} ; {context.Request?.Method} ; " +
                $"{context.Request?.Path.Value} ; {context.Response?.StatusCode}");
        }
    }
}
```