МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ПРИВАТНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

Кафедра інформаційних технологій

ДО ЗАХИСТУ ДОПУЩЕНА

Зав.кафедрою_____

д.е.н., доцент, Левицький С.І.

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА
ДОСЛІДЖЕННЯ ПРАКТИЧНИХ МОЖЛИВОСТЕЙ SPA ФРЕЙМВОРКІВ У ПРОЦЕСІ СТВОРЕННЯ ВЕБ-ДОДАТКУ

Виконав
ст.гр. ІПЗ–129 _____ Деркач В.А.

Науковий керівник _____ Левицький С.І.
д.е.н., доцент

Запоріжжя
2023

ПРАТ «ПВНЗ «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»
Кафедра інформаційних технологій

ЗАТВЕРДЖУЮ

Зав.кафедрою_____

д.е.н., доцент, Левицький С.І.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ

Студенту гр. ІПЗ–129, спеціальності Інженерія програмного забезпечення
Деркачу Владиславу Андрійовичу

1. Тема: Дослідження практичних можливостей SPA фреймворків у процесі створення веб-додатку
затверджена наказом по інституту № 02-10 від 27.01.2023 р.
2. Термін здачі студентом закінченої роботи: 12.06.2023 р.
3. Перелік питань, що підлягають розробці

1. Провести огляд літератури, що присвячена тематиці досліджень.

2. Розглянути і порівняти інструментарій розробки.

3. Дослідити можливості SPA фреймворків у контексті створення UI.

4. Дослідити можливості SPA фреймворків при взаємодії з іншими технологіями (API, БД, тощо).

5. Дослідити можливості SPA фреймворків у контексті SEO.

6. Протестувати розроблений додаток на стабільність роботи.

7. Оформити результати досліджень у вигляді звіту.

ЗАТВЕРДЖУЮ

Зав.кафедрою_____

д.е.н., доцент, Левицький С.І.

К А Л Е Н Д А Р Н И Й Г Р А Ф І К

підготовки бакалаврської дипломної роботи

здобувачами освіти інституту ЗІЕІТ очної форми навчання

гр. _____ П.І.Б._____

2022-2023 навчальний рік

№ етапу	Зміст	Термін виконання	Готовність по графіку (%), підпис керівника	Підпис керівника про повну готовність етапу, дата
1	Збір практичного матеріалу за темою дипломної роботи	16.01.23-11.02.23		
2	І атестація І розділ бакалаврської дипломної роботи	27.03.23-01.04.23		
3	ІІ атестація ІІ розділ бакалаврської дипломної роботи	24.04.23-29.04.23		
4	ІІІ атестація ІІІ розділ бакалаврської дипломної роботи, висновки та рекомендації, додатки, реферат	22.05.23-27.05.23		
5	Перевірка дипломної роботи на оригінальність	15.05.23-12.06.23		
6	Доопрацювання бакалаврської дипломної роботи, підготовка презентації, отримання відгуку керівника і рецензії	29.05.23-12.06.23		
7	Попередній захист бакалаврської дипломної роботи	12.06.23-18.06.23		
8	Подача бакалаврської дипломної роботи на кафедру	За 3 дні до захисту		
9	Захист бакалаврської дипломної роботи	19.06.23-24.06.23		

Керівник кваліфікаційної
бакалаврської роботи _____ Левицький С.І.

(підпис) (прізвище та ініціали)

Завдання отримав до виконання _____ Деркач В.А.

(підпис студента) (прізвище та ініціали)

РЕФЕРАТ

Кваліфікаційна бакалаврська робота містить 66 сторінки, 40 рисунків, 11 джерел та 1 додаток.

Мета даної роботи полягає дослідженні практичних можливостей SPA фреймворків у процесі створення веб-додатку.

Об’єктом дослідження є можливості SPA фреймворків.

Предметом дослідження є додаток для громадської організації «Офіс перспективного розвитку».

Здійснено детальний огляд предметної області. Виявлено, що використання можливостей SPA фреймворків в розробці веб додатків є доцільним.

Результатом виконаної роботи є проведене дослідження можливостей SPA фреймворків. Виявлення переваг та недоліків таких фреймворків, та створення веб-додатку, що представляє собою візитний сайт з блогами, статтями, презентаціями та панеллю адміністратора для громадської організації «Офіс перспективного розвитку».

Дослідження показало, що SPA фреймворки пропонують практичні можливості для створення веб-додатків, які відповідають вимогам сучасних користувачів. Використовуючи можливості цих фреймворків, розробники можуть забезпечувати багатий і захоплюючий досвід користувача, зберігаючи організацію коду і масштабованість. Водночас такі фреймворки мають проблеми з пошукової оптимізацією, часом початкового завантаження та сумісністю зі старими браузерами.

SINGLE PAGE APP, SPA, ANGULAR, SEO, ВЕБ-РОЗРОБКА

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Слово / словосполучення	Скорочення
База даних	БД
Програмне забезпечення	ПЗ
Операційна система	ОС
Система управління БД	СУБД
Мова структурованих запитів (англ. Structed Query Language)	SQL
Інтерфейс прикладного програмування (англ. Application Programming Interface)	API
Каскадні таблиці стилів (англ. Cascading Style Sheets)	CSS
Об’єктна модель документа (англ. Document Object Model)	DOM
Контролер, що оброблює усі запити до веб-сайту (англ. Front Controller)	FC
Мова розмітки гіпертексту (англ. Hyper Text Markup Language)	HTML
Система доступу до пов'язаних між собою документів на різних комп'ютерах, підключених до Інтернету	WEB
Технологія збільшення продуктивності програмних систем, шляхом компіляції байт-коду в машинний код або в інший формат безпосередньо під час роботи програми (англ. Just-in-time)	JIT

Методологія, націлена на взаємодію програмістів і системних адміністраторів для збільшення продуктивності роботи (англ. Development Operations)	DevOps
Технологія у web-розробці, яка візуально і функціонально трансформує сайт в додаток (англ. Progressive Web Application)	PWA
Система програмних засобів, яка використовується програмістами для розробки програмного забезпечення (англ. Integrated Development Environment)	IDE
Препроцесор для написання CSS коду	SCSS
Односторінковий додаток (англ. Single Page Application)	SPA
Багатосторінкові програми	MPA
Інтерфейс користувача (англ. User interface)	UI
Досвід користувача (англ. User Experience). Він описує взаємодію користувача з продуктом або послугою та охоплює всі аспекти сприйняття, які користувач відчуває у процесі використання продукту	UX
Громадська організація	ГО
Пошукова оптимізація (англ. Search Engine Optimization)	SEO

ВСТУП

Веб-додатки є невід'ємною частиною нашого сучасного цифрового життя, і їх створення потребує ефективних інструментів та підходів. Одним із найактуальніших напрямків веб-розробки є SPA фреймворки. Вони пропонують нові практичні можливості для розробників, покращуючи процес створення веб-додатків і забезпечуючи більш зручний досвід користувача.

Метою даного дослідження є вивчення практичних можливостей SPA фреймворків у контексті розробки веб-додатків. Ми зосередимося на детальному розгляді такого SPA фреймворку як Angular. Та на його впливі на процес розробки.

В ході дослідження ми розглянемо основні переваги SPA фреймворків, такі як маршрутизація на стороні клієнта, динамічне оновлення вмісту, оптимізація продуктивності та підвищення зручності веб-додатку.

Результати цього дослідження будуть корисні для розробників, керівників проектів та всіх, хто цікавиться сучасними технологіями веб-розробки. Тому тема роботи є актуальною та має практичну цінність.

Тож дослідження буде проводитися на прикладі створення веб-додатку, що представляє собою візитний сайт з блогами, статтями та презентаціями для ГО «Офіс перспективного розвитку», та включатиме в себе адміністративну панель, що дозволяє редагувати, видаляти, та додавати контент. Додаток буде побудований з використанням бази даних MySQL. Серверна частина проекту буде розроблена з використанням фреймворку FoalTS, на платформі NodeJS. Клієнтська частина проекту буде розроблена за допомогою мови програмування TypeScript, з використанням фреймворку Angular. За допомогою бібліотеки Angular Universal додаток буде оптимізований під SEO потреби. Також використовуючи можливості препроцесора SCSS буде забезпечена повна адаптивність дизайну під любі розміри девайсу на якому його буде використано.

Серверна та клієнтська частини будуть розроблені з використанням IDE - Visual Studio Code.

РОЗДІЛ 1

ІНСТРУМЕНТИ РОЗРОБКИ

1.1 Загальні положення першого розділу

Процес створення будь-якого сучасного веб-додатку потребує використання багатьох інструментів розробки. Тож у цьому розділі будуть розглянуті всі застосовані інструменти. Почнемо з загального огляду SPA технології та причин, шо призвели до її виникнення.

1.2 SPA - шлях до односторінкових додатків

У світі веб-розробки односторінкові додатки стають дедалі популярнішими, постійно вдосконалюючись із кожною новою версією, яка привертає увагу спільноти розробників.

До появи SPA багатосторінкові програми (MPA) домінували в Інтернеті. MPA були створені з використанням статичного HTML і серверних технологій, таких як PHP, ASP, Java, Ruby та Python. Цікаво, що те, що ми зараз вважаємо «звичайним» веб-сайтом, по суті є MPA.

MPA функціонували шляхом виконання кількох запитів між клієнтом і сервером. Оскільки веб-сайти ставали все складнішими, вимоги до серверів зростали в геометричній прогресії. Поява AJAX, яка дозволила оновлювати веб-сторінки без перезавантаження, частково пом’якшила цю проблему. Це також дозволило розробникам зазирнути в потенційне майбутнє SPA.

Важливою віхою на шляху до SPA став випуск jQuery наприкінці 2006 року. Хоча сам по собі не був фреймворком SPA, jQuery був першим великим гравцем у довгій лінії фреймворків JavaScript. Однак jQuery в основному

зосереджувався на інтерфейсі користувача, не маючи надійної підтримки обробки даних. Хоча це обмеження не було проблематичним для простих веб-додатків, воно створювало проблеми при роботі з динамічними сторінками або великими корпоративними додатками.

Слідом за jQuery з'явився Knockout.js, який представив прив'язку даних MVVM (Model-View-ViewModel). Цей фреймворк спростив зв'язування даних, встановивши чіткий розподіл між виглядом і даними додатку. Завдяки прив'язці моделі перегляду до HTML браузера будь-які зміни, зроблені в браузері, автоматично відстежувалися та відображалися в моделі перегляду, і навпаки.

Подорож до SPA, які ми маємо сьогодні, була далеко не лінійною. Flash і Silverlight, наприклад, представили багаті інтернет-програми (RIA), які повністю відійшли від JavaScript. Незважаючи на те, що спочатку вони були прийняті розробниками, їхня залежність від сторонніх плагінів для браузерів обмежила їх впровадження, зробивши їх застарілими фреймворками.

Повертаючись до JavaScript, Backbone.js вийшов на сцену в 2009 році, пропонуючи полегшену клієнтську структуру. Хоча було можливо створити SPA за допомогою Backbone, це потребувало значних зусиль і призвело до повторюваного коду. Лише в 2010 році всі ці фрагментовані ідеї були об'єднані у випуск Angular.js. Забезпечуючи клієнтську архітектуру Model-View-Controller (MVC), двостороннє зв'язування даних, шаблони та впровадження залежностей в єдину структуру, Angular.js став першим справжнім рішенням для розробки SPA. Його наступні версії, такі як Angular, ми більш ретельно розглянемо у наступному підрозділі.

Завдяки низці досягнень та інновацій, SPA фреймворки зробили революцію у веб-розробці, дозволивши розробникам створювати динамічні, інтерактивні та безперебійні користувальницькі умови. Оскільки технології продовжують розвиватися, SPA, ймовірно, залишаться в авангарді розробки сучасних веб-додатків, керуючи майбутнім цифровим досвідом, орієнтованим на користувача.

1.3 Angular

```
ngOnInit(): void {
  this._setInactivityTimeout();
  this._setExtendSessionInterval();
}

ngOnDestroy(): void {
  this._clearInactivityTimeout();
  this._clearExtendSessionInterval();
}

@HostListener('document:keypress')
@HostListener('document:mousedown')
@HostListener('document:mousemove')
@HostListener('document:touchstart')
@HostListener('document:touchmove')
resetInactivityTimeout() {
  this._clearInactivityTimeout();
  this._setInactivityTimeout();
}

logout(): void {
  this._showSpinner();
  this._authService
    .logout$()
    .toPromise()
    .then(() => this._showSpinner(false))
    .catch(error => {
      console.error(error);
      this._showSpinner(false);
    });
}
```

Рис. 1.1 – Приклад роботи з Angular

Angular (версія 2 і вище) - це платформа з відкритим кодом для розробки веб-додатків. Він написаний на TypeScript і розроблений командою Google разом із внесками розробників із різних компаній. Angular - це повністю переписаний фреймворк, раніше відомий як AngularJS, створений тією ж командою.[1]

Angular надає повний набір інструментів і функцій для створення сучасних, масштабованих і надійних веб-додатків. Він дотримується компонентної архітектури, де додатки створюються шляхом компонування повторно використовуваних та інкапсульованих компонентів. Ці компоненти відповідають за керування власним станом, обробку взаємодії користувача та відтворення інтерфейсу користувача.

Однією з ключових особливостей Angular є його потужна система зв’язування даних, яка дозволяє декларативну синхронізацію даних між моделлю програми та представленням. Це двостороннє зв’язування даних спрощує процес розробки та допомагає підтримувати синхронізацію інтерфейсу користувача з основними даними.

Angular також містить багатий набір вбудованих директив, таких як ngIf, ngFor і ngSwitch, які дозволяють розробникам маніпулювати DOM і контролювати рендеринг компонентів на основі різних умов.

Крім того, Angular забезпечує надійну підтримку обробки форм, та маршрутизації, що полегшує керування складними потоками додатків і залежностями.

Завдяки модульній архітектурі та розгалуженій екосистемі бібліотек та інструментів Angular пропонує розробникам структурований та ефективний підхід до створення великомасштабних програм. Він сприяє повторному використанню коду, тестуванню та легкому виправленню допущених помилок, що робить його популярним вибором серед розробників як для невеликих проектів, так і для програм корпоративного рівня.

Система модульності Angular називається NgModules. Вона пов'язує код для формування функціональних одиниць.

Різні та не пов'язані між собою модулі фреймворку, як і модулі JavaScript, можуть імпортувати функціонал з інших модулів, експортувати та використовувати їх функціональні можливості.

У кожному Angular додатку є центральний модуль, умовно названий AppModule, який забезпечує запуск додатку. Процес завантаження створює компоненти, перелічені в масиві “bootstrap”, і вставляє їх у DOM браузера. Отже, кожен завантажений компонент є базою свого власного дерева компонентів.

Як ми бачили, ми можемо мати кореневий модуль, але можемо мати те, що ми називаємо Feature Modules. Модуль функцій забезпечує згуртований набір функціональних можливостей, орієнтованих на конкретні потреби додатків. Ми могли б зробити все в кореневому модулі, але Feature Module допоможе нам розділити наш додаток на зосереджені ділянки. Однак структура модуля функцій точно така ж, як і структура кореневого модуля.[2]

Загалом, Angular - це потужний фреймворк, який дає змогу розробникам створювати багатофункціональні веб-додатки з сильним акцентом на організації коду та зручності обслуговування.

1.4 Angular Material

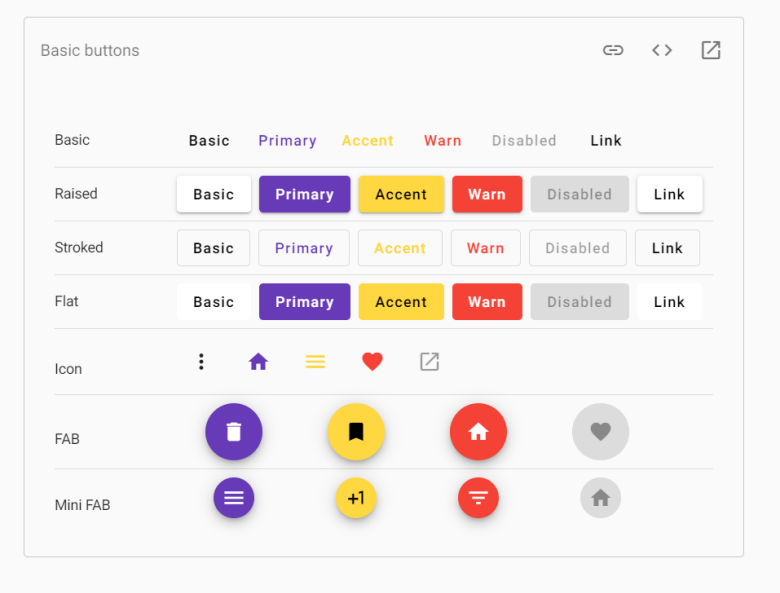

Рис. 1.2 – Приклад Angular Material

Angular Material - це бібліотека компонентів інтерфейсу користувача для програм Angular. Він надає набір попередньо створених, готових до використання компонентів інтерфейсу користувача, які відповідають принципам Material Design, який є мовою дизайну, розробленою Google. Angular Material спрощує процес створення візуально привабливих та інтерактивних інтерфейсів користувача для веб-додатків.

1.5 Angular Universal

```
import 'zone.js/dist/zone-node';

import { ngExpressEngine } from '@nguniversal/express-engine';
import * as express from 'express';
import { join } from 'path';

import { AppServerModule } from './src/main.server';
import { APP_BASE_HREF } from '@angular/common';
import { existsSync } from 'fs';

import { createProxyMiddleware } from 'http-proxy-middleware';
import { getSSRServerConf } from 'src/app/modules/core/functions/get-ssr-server-conf/get-ssr-server-conf.function';
import { ISSRServerConf } from 'src/app/modules/core/interfaces/ssr-server-conf.interface';

// The Express app is exported so that it can be used by serverless Functions.
export async function app(): Promise<express.Express> {
  const server = express();
  const distFolder = join(process.cwd(), 'dist/browser');
  const indexHtml = existsSync(join(distFolder, 'index.original.html')) ? 'index.original.html' : 'index';

  // Add proxy to other back-end
  const ssrServerConf: ISSRServerConf = await getSSRServerConf();
  const proxyMiddleware = createProxyMiddleware({
    target: ssrServerConf.proxyPrefix,
    changeOrigin: true
  });
  ssrServerConf.linksNeedRedirect.forEach((link: string) => server.use(link, proxyMiddleware));

  // Our Universal express-engine (found @ https://github.com/angular/universal/tree/master/modules/express-engine)
  server.engine('html', ngExpressEngine({
    bootstrap: AppServerModule
  }));
```

Рис. 1.3 – Приклад використання Angular Universal

Angular Universal виконується на сервері, генеруючи статичні сторінки програми, які пізніше завантажуються на клієнті. Програма зазвичай

відтворюється швидше, даючи користувачам можливість переглянути макет програми, перш ніж вона стане повністю інтерактивною.

Google, Bing, Facebook, Twitter та інші сайти соціальних мереж покладаються на веб-сканери, щоб індексувати вміст вашої програми та зробити цей вміст доступним для пошуку в Інтернеті. Ці веб-сканери можуть бути не в змозі здійснювати навігацію та індексувати ваш інтерактивний додаток Angular, як це може зробити користувач.

Angular Universal може генерувати статичну версію вашої програми, яку легко шукати. Universal також робить доступним попередній перегляд сайту, оскільки кожна URL-адреса повертає повністю відтворену сторінку.

Деякі пристрої не підтримують JavaScript або виконують JavaScript настільки погано, що досвід користувача є неприйнятним. У цих випадках вам може знадобитися версія програми, яка відтворюється на сервері без JavaScript. Ця версія, хоч і обмежена, може бути єдиною практичною альтернативою для людей, які інакше взагалі не могли б використовувати програму.

Швидке відображення першої сторінки може мати вирішальне значення для залучення користувачів. Сторінки, які завантажуються швидше, працюють краще, навіть зі змінами всього за 100 мс. Можливо, вашій програмі доведеться запуститися швидше, щоб залучити цих користувачів, перш ніж вони вирішать зробити щось інше.

За допомогою Angular Universal можливо створювати цільові сторінки для програми, які виглядають як повна програма. Сторінки є чистим HTML і можуть відображатися, навіть якщо JavaScript вимкнено. Сторінки не обробляють події браузера, але вони підтримують навігацію по сайту за допомогою routerLink.

На практиці будете показуватися статична версію цільової сторінки, щоб привернути увагу користувача. У той же час буде завантажуватися повна програма Angular. Користувач відчуває майже миттєву продуктивність з цільової сторінки та отримує повний інтерактивний досвід після повного завантаження програми.[3]

1.6 TypeScript

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

document.addEventListener('DOMContentLoaded', () => {
  platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
});
```

Рис. 1.4 – Приклад роботи з мовою програмування TypeScript

TypeScript - мова програмування, представлена Microsoft у 2012 році яка позиціонується, як інструмент для розробки веб-додатків, що розширює JavaScript.

Творцем TypeScript є Андерс Хейльсберг, який раніше створив Turbo Pascal, Delphi та C #.

Мовні специфікації відкриваються та публікуються відповідно до Угоди про специфікацію відкритого веб-сайту.

TypeScript сумісний з JavaScrip. Насправді після завершення додатків TypeScript може працювати в будь-якому сучасному браузері або використовуватися з Node.js. Код експериментального компілятора, який переводить TypeScript у JavaScript, затверджений за ліцензією Apache. Його розвиток відбувається у публічних сховищах через сервіс GitHub.

TypeScript відрізняється від JavaScript тим, що чітко призначає типи, підтримує використання високоякісних класів (як це стосується традиційних мов з об'єктно-орієнтованою мовою) та підтримує додатки, розроблені для підвищення швидкості розвитку, полегшуючи читання та повторне використання коду.

Передбачається, що завдяки повній сумісності адаптацію існуючих додатків до нової мови програмування можна здійснювати поступово, з поступовим визначенням типів.[4]

1.7 MySQL

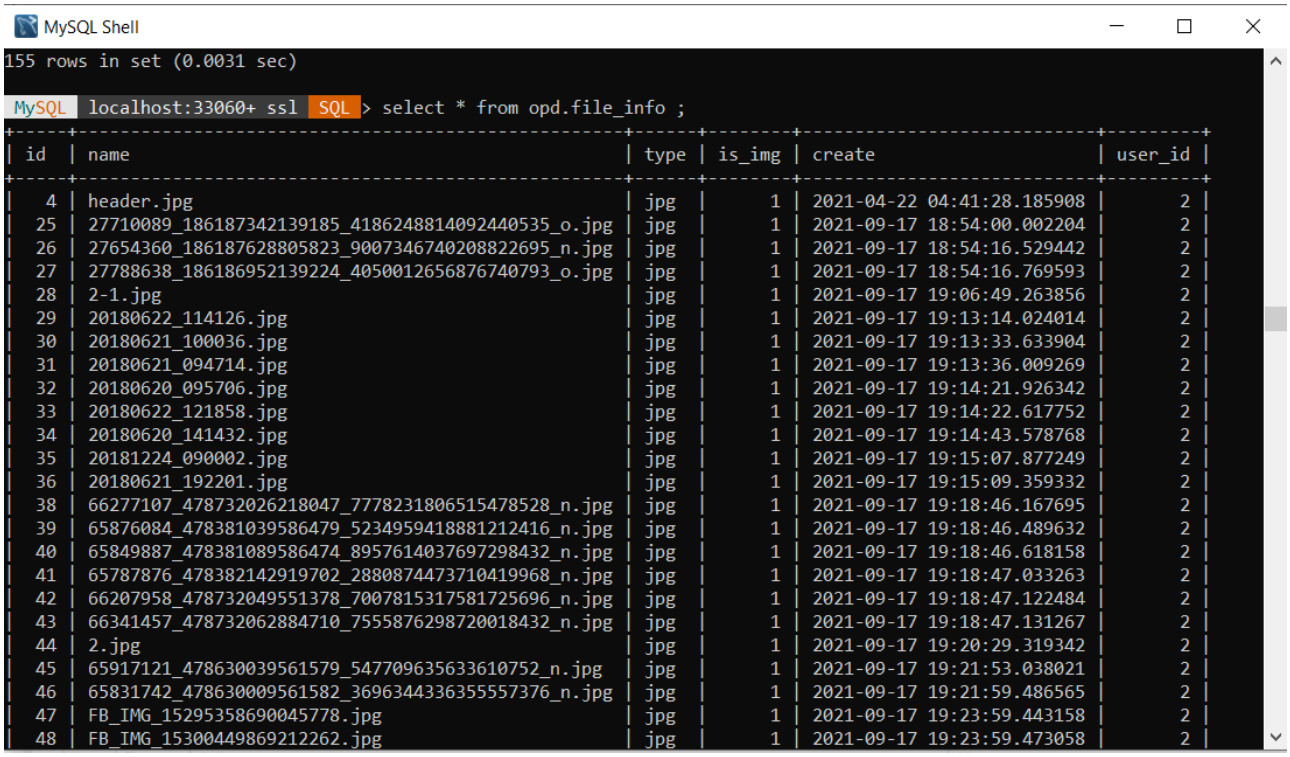

Рис. 1.5 – СУБД MySQL

MySQL — це система керування реляційною базою даних (RDBMS) із відкритим кодом, яка широко використовується для зберігання, керування та отримання даних. Його спочатку розробили Майкл Віденіус і Девід Аксмарк у

1995 році, а пізніше придбала корпорація Oracle. MySQL відомий своєю продуктивністю, масштабованістю та простотою використання, що робить його популярним вибором як для невеликих програм, так і для великих корпоративних систем.[5]

Управління реляційною базою даних: MySQL дотримується реляційної моделі, що дозволяє організовувати дані в таблиці, що складаються з рядків і стовпців. Цей структурований формат забезпечує ефективне зберігання, пошук і керування даними.

Типи даних: MySQL надає широкий спектр типів даних для обробки різних видів інформації, включаючи числову, рядкову, дату й час, логічні значення тощо. Ця гнучкість дозволяє розробникам точно представляти різноманітні набори даних і маніпулювати ними.

Мова обробки даних (DML): MySQL підтримує SQL, стандартну мову для взаємодії з реляційними базами даних. За допомогою SQL користувачі можуть виконувати різні операції, такі як вставка, оновлення, видалення та запит даних у базах даних MySQL.

Мова визначення даних (DDL): MySQL містить оператори DDL для визначення структури бази даних, таблиць та індексів. Ці оператори дозволяють користувачам створювати, змінювати та керувати схемою бази даних.

Цілісність даних: MySQL забезпечує цілісність даних за допомогою первинних ключів, зовнішніх ключів і обмежень. Первинні ключі однозначно ідентифікують кожен рядок у таблиці, а зовнішні ключі встановлюють зв’язки між таблицями. Обмеження, такі як унікальність, не нуль і перевірка, забезпечують узгодженість і дійсність даних.

Індексування та оптимізація: MySQL підтримує індексування, яке покращує продуктивність запитів шляхом створення структур даних, які забезпечують швидкий пошук даних. Він також включає такі методи оптимізації, як кешування запитів, оптимізація запитів і налаштування бази даних, покращуючи загальну продуктивність системи.

Масштабованість і висока доступність: MySQL пропонує параметри масштабованості для обробки зростаючих обсягів даних і навантажень користувачів. Він підтримує реплікацію, яка дозволяє копіювати дані на декілька серверів баз даних, підвищуючи доступність і забезпечуючи балансування навантаження.

Безпека: MySQL забезпечує надійні функції безпеки для захисту даних, включаючи автентифікацію користувачів, контроль доступу та зашифровані з’єднання. Це гарантує, що тільки авторизовані користувачі можуть отримати доступ і змінити базу даних.

Міжплатформна сумісність: MySQL сумісна з кількома операційними системами, включаючи Windows, macOS, Linux і різними варіантами Unix. Ця гнучкість дозволяє розробникам розгортати MySQL на широкому діапазоні платформ.

Розширюваність: MySQL підтримує використання визначених користувачем функцій, збережених процедур і тригерів, що дозволяє розробникам розширювати функціональні можливості бази даних. Ці функції дозволяють реалізувати індивідуальну бізнес-логіку в базі даних.

MySQL має велику спільноту розробників та широко використовується організаціями будь-якого розміру, від стартапів до великих підприємств, для різноманітних додатків, включаючи веб-розробку, програми, керовані даними, системи електронної комерції, тощо.

1.8NodeJS

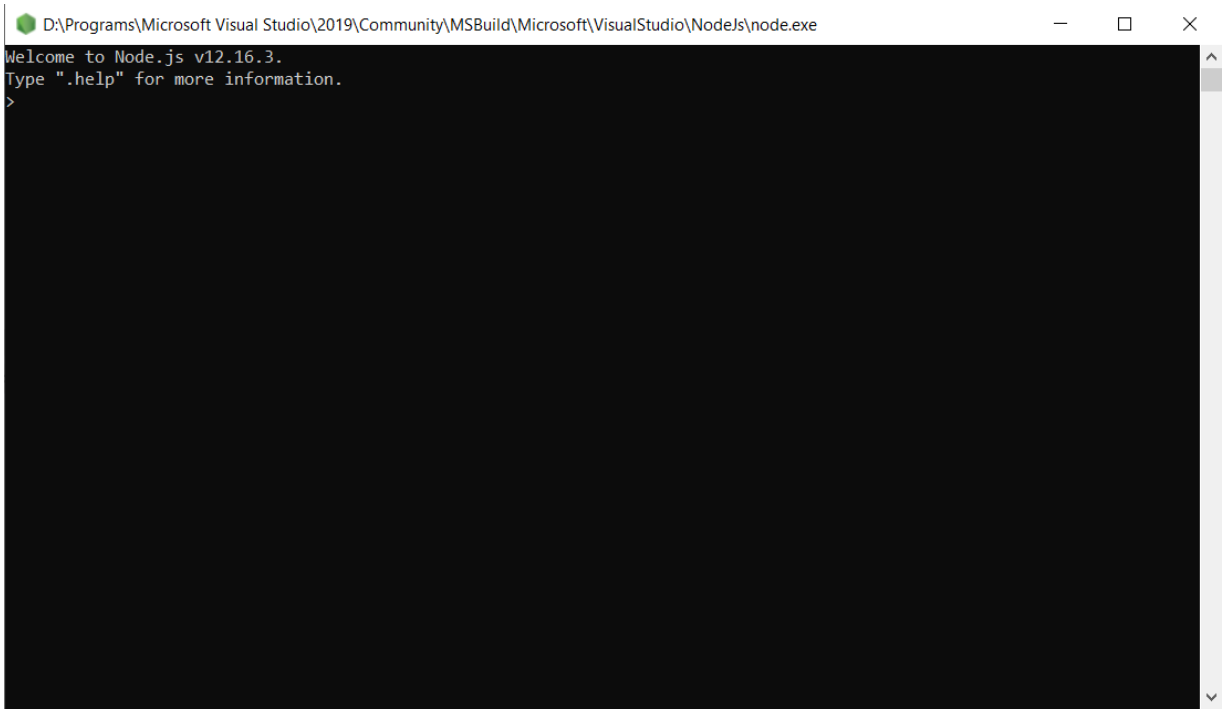

Рис. 1.6 – Консоль NodeJS

Node.js - це серверне середовище виконання JavaScript із відкритим вихідним кодом, створене на движку JavaScript V8. Це дозволяє розробникам запускати код JavaScript на сервері, дозволяючи їм створювати масштабовані та ефективні мережеві програми. Node.js був розроблений Райаном Далем у 2009 році та набув значної популярності завдяки своїм унікальним функціям і перевагам.

Node.js дозволяє розробникам використовувати JavaScript як на стороні клієнта, так і на стороні сервера, надаючи єдину мову програмування для веб-додатків. Це усуває необхідність перемикатися між різними мовами програмування, покращує та спрощує процес розробки, та сприяє спільному використанню коду.

Node.js має величезну екосистему пакетів з відкритим кодом, доступних через реєстр NPM. Розробники можуть легко знаходити та інтегрувати сторонні

бібліотеки та модулі у свої додатки, заощаджуючи час і зусилля на створенні функцій з нуля.

Node.js має яскраву та активну спільноту розробників, яка сприяє його постійному зростанню та вдосконаленню. Спільнота надає розширену документацію, навчальні посібники та ресурси, що полегшує розробникам навчання, усунення несправностей і співпрацю над проектами Node.js.

Node.js сумісний із багатьма операційними системами, включаючи Windows, macOS і різними дистрибутивами Linux, що робить його дуже портативним і придатним для розгортання на різних платформах.

Node.js зазвичай використовується для розробки веб-серверів, програм реального часу, мікросервісів, API та різноманітних інших мережевих програм. Його універсальність, продуктивність і широка екосистема зробили його популярним вибором серед розробників і організацій, які прагнуть створювати масштабовані та ефективні веб-додатки. [6]

1.9 Visual Studio Code

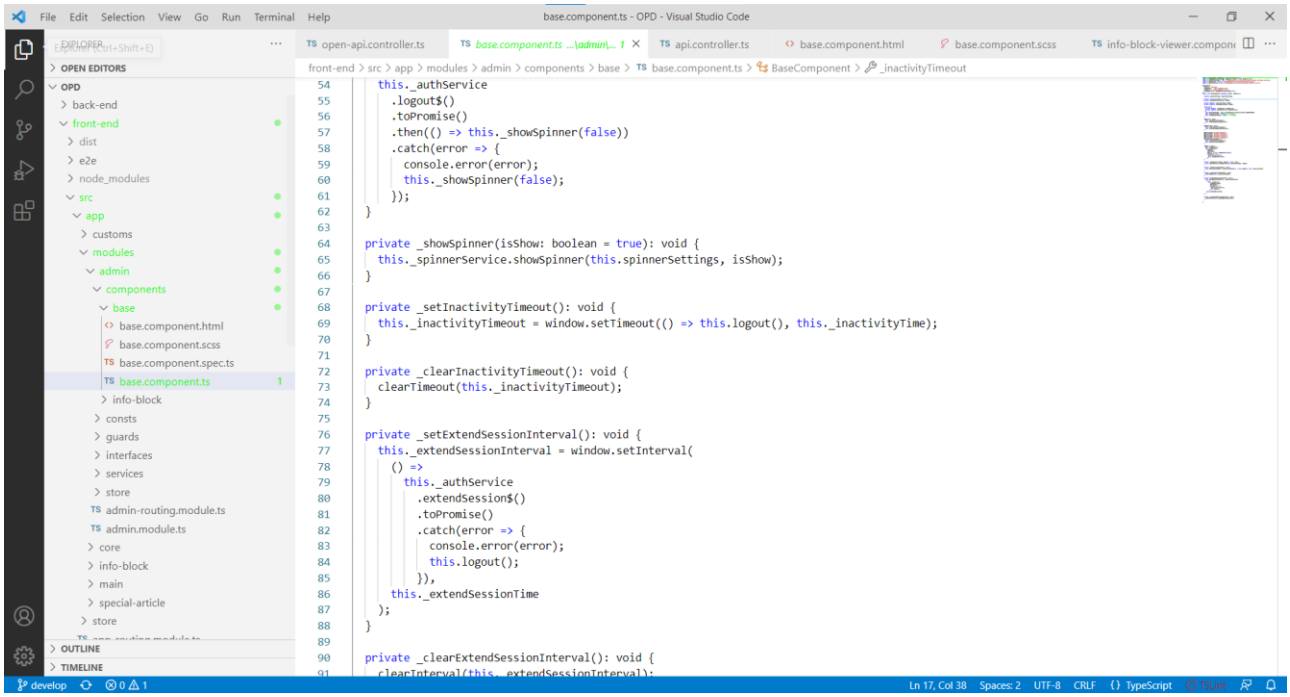

Рис. 1.7 – Редактор Visual Studio Code

Visual Studio Code - редактор коду, розроблений Microsoft. Він встановлюється як “простий” процесор коду для розробки веб-додатків. Включає інструмент виявлення помилок, інструменти Git, інструменти Intelli Sense та рефакторинг. У ньому є широкі параметри налаштування: спеціальні теми, ярлики та файли налаштувань. Він розповсюджується безкоштовно.

Visual Studio Code заснований на Electron - фреймворк, який дозволяє використовувати Node.js для розробки настільних додатків, що працюють за механізмом Blink. Хоча процесор базується на Electron, він не використовує процесор Atom. Натомість реалізований веб-процесор Monaco, призначений для Visual Studio Online.[7]

1.10 FoalTS

```
import 'source-map-support/register';

// std
import * as http from 'http';

// 3p
import { Config, createApp, displayServerURL } from '@foal/core';
import { createConnection } from 'typeorm';

// App
import { AppController } from './app/controllers/app.controller';
import { InfoBlockService } from './app/services/info-block/info-block.service';

async function main(): Promise<void> {
  await createConnection();
  const app = await createApp(AppController);
  const httpServer = http.createServer(app);
  const port = Config.get('port', 'number', 3001);
  httpServer.listen(port, () => {
    displayServerURL(port);
    new InfoBlockService().generateRobotsTxtAndSitemap();
  });
}

main()
  .catch(error => {
    console.error(error);
    process.exit(1);
  });
```

Рис. 1.8 – Приклад роботи з FoalTS

Foal (або FoalTS) — це фреймворк Node.JS для створення веб-додатків.

В одному місці у вас є повне середовище для створення веб-додатків. Це включає інтерфейс команди, інструменти тестування, утиліти інтерфейсу, сценарії, розширену автентифікацію, ORM, середовища розгортання, API GraphQL і Swagger, утиліти AWS тощо.

Але, пропонуючи всі ці функції, структура залишається простою. Складність і непотрібні абстракції відкидаються, щоб забезпечити найбільш інтуїтивно зрозумілий і виразний синтаксис. Ми віримо, що стислий та елегантний код є найкращим способом розробки програми та підтримки її в

майбутньому. Це також дозволяє витрачати більше часу на кодування, а не на спроби зрозуміти, як працює фреймворк.

Фреймворк повністю написаний на TypeScript. Ця мова пропонує додаткову статичну перевірку типів разом із найновішими функціями ECMAScript. Це дозволяє виявляти більшість безглуздих помилок під час компіляції та покращувати якість коду. Він також пропонує вам автозаповнення та добре документований API.[8]

1.11 MySQL Workbench

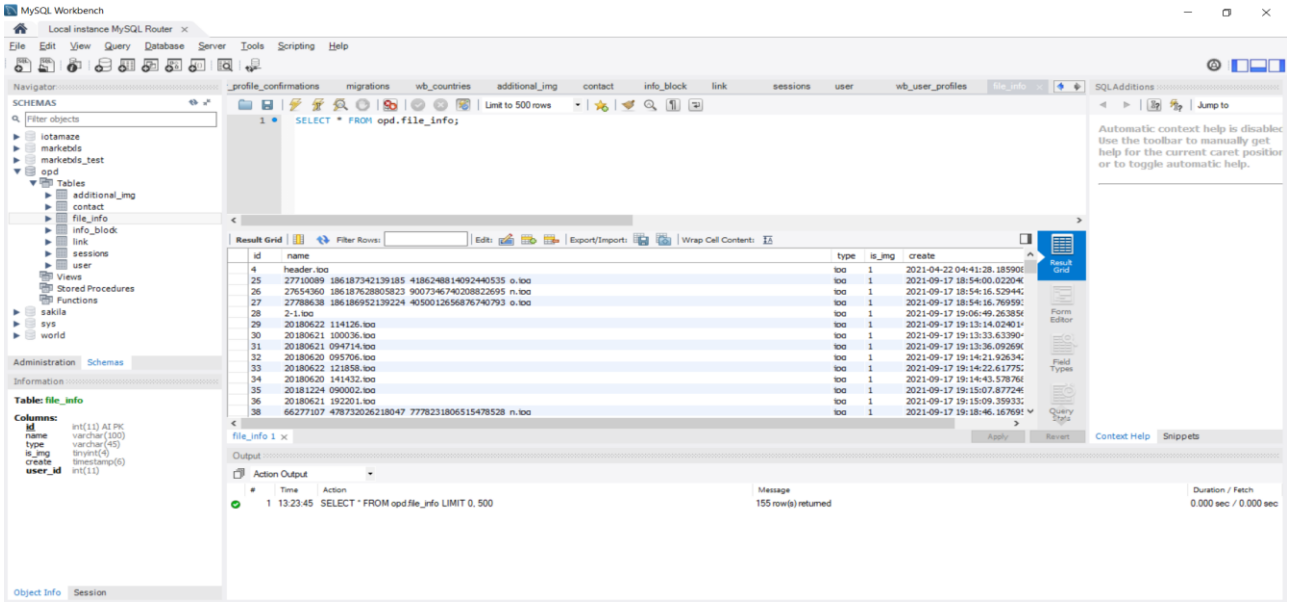

Рис. 1.9 – IDE MySQL Workbench

MySQL Workbench — це інструмент візуального проектування та адміністрування баз даних, розроблений корпорацією Oracle. Він спеціально розроблений для роботи з базами даних MySQL і надає повний набір функцій для розробки баз даних, моделювання, запитів та адміністрування.

MySQL Workbench дозволяє розробникам створювати та редагувати схеми баз даних за допомогою графічного інтерфейсу. Він підтримує моделювання

сутності та зв’язку, де таблиці, стовпці, зв’язки та обмеження можна візуально визначити. Ця функція допомагає проектувати та візуалізувати структуру бази даних перед її впровадженням.

Інструмент надає потужний редактор SQL із підсвічуванням синтаксису, доповненням коду та можливостями форматування. Розробники можуть писати та виконувати SQL-запити до своїх баз даних MySQL безпосередньо в MySQL Workbench. Інтегрований редактор також підтримує історію запитів і маніпуляції з набором результатів.

MySQL Workbench забезпечує плавну міграцію даних із різних джерел до баз даних MySQL. Він підтримує імпорт і експорт даних із різних форматів файлів, таких як CSV, JSON, XML і Excel. Ця функція спрощує процес передачі даних між різними системами баз даних.[9]

MySQL Workbench пропонує інструменти для управління та адміністрування серверів MySQL. Він забезпечує зручний інтерфейс для моніторингу стану сервера, керування екземплярами сервера, налаштування параметрів сервера та перегляду журналів сервера. Адміністратори також можуть виконувати такі завдання, як керування користувачами, резервне копіювання та відновлення бази даних і налаштування сервера.

Інструмент містить інформаційну панель продуктивності, яка надає інформацію про продуктивність баз даних MySQL у реальному часі. Він відображає такі показники, як використання ЦП, використання пам’яті та дисковий ввід-вивід, допомагаючи розробникам визначити вузькі місця продуктивності та оптимізувати операції з базою даних.

MySQL Workbench дозволяє користувачам створювати детальну документацію для своїх баз даних MySQL. Ця документація містить інформацію про таблиці, стовпці, зв’язки та інші об’єкти бази даних. Він забезпечує зручний спосіб документування та обміну проектами та структурами баз даних з іншими членами команди чи зацікавленими сторонами.

MySQL Workbench підтримує співпрацю та контроль версій через інтеграцію з популярними системами контролю версій, такими як Git. Кілька розробників можуть працювати над однією схемою бази даних одночасно, а зміни можна відстежувати, об’єднувати та скасовувати за допомогою функцій контролю версій.

MySQL Workbench підтримує розширюваність за допомогою плагінів. Розробники можуть створювати власні плагіни для покращення функціональності інструменту або інтегрувати його з іншими системами чи інструментами.

MySQL Workbench надає зручний і візуальний інтерфейс для роботи з базами даних MySQL, що робить його придатним для адміністраторів баз даних, розробників і аналітиків даних. Його комплексний набір функцій оптимізує процеси розробки та адміністрування баз даних, допомагаючи користувачам ефективно проектувати, керувати та оптимізувати свої бази даних MySQL.

1.12 GitLab

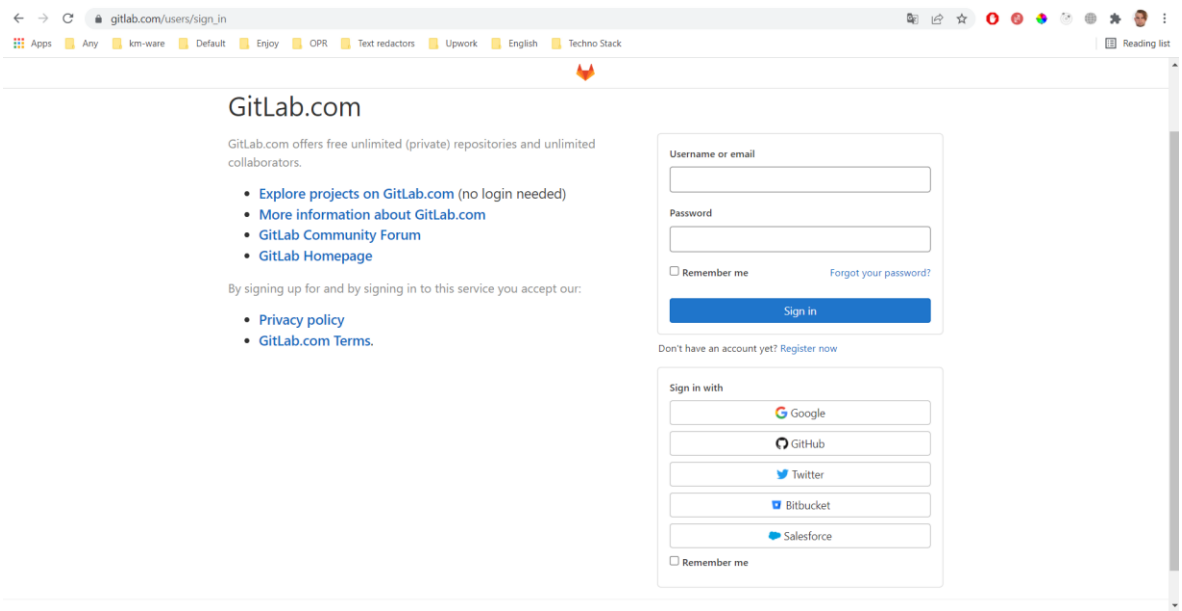

Рис. 1.10 – Сторінка входу до ресурсу GitLab

GitLab — це веб-платформа DevOps, яка надає повний набір інструментів для контролю версій, безперервної інтеграції/безперервної доставки (CI/CD), відстеження проблем, перевірки коду та керування проектами. Він пропонує інтегроване середовище для спільної роботи для команд розробників програмного забезпечення для ефективного планування, розробки, тестування, розгортання та моніторингу своїх програм.[10]

1.13 Висновки по першому розділу

У першому розділі були розглянуті всі інструменти, що використовувалися під час розробки веб-додатку. Особлива увага була приділена SPA фреймворку Angular та сумісним з ним технологіям, таким як:

1. UI бібліотеці - Angular Material;
2. Бібліотеці, що запроваджує SEO оптимізацію - Angular Universal;
3. Мові програмування - TypeScript.

РОЗДІЛ 2

ІНТЕРФЕЙС КОРИСТУВАЧА

2.1 Загальні положення другого розділу

У цьому розділі, на прикладі використання Angular Material, будуть ретельно розглянуті можливості SPA фреймворків у створенні гнучких, адаптивних та зручних у використанні інтерфейсів.

2.2 Сторінка новин

Головною сторінкою сайту є сторінка новин, розбитих по категоріям.

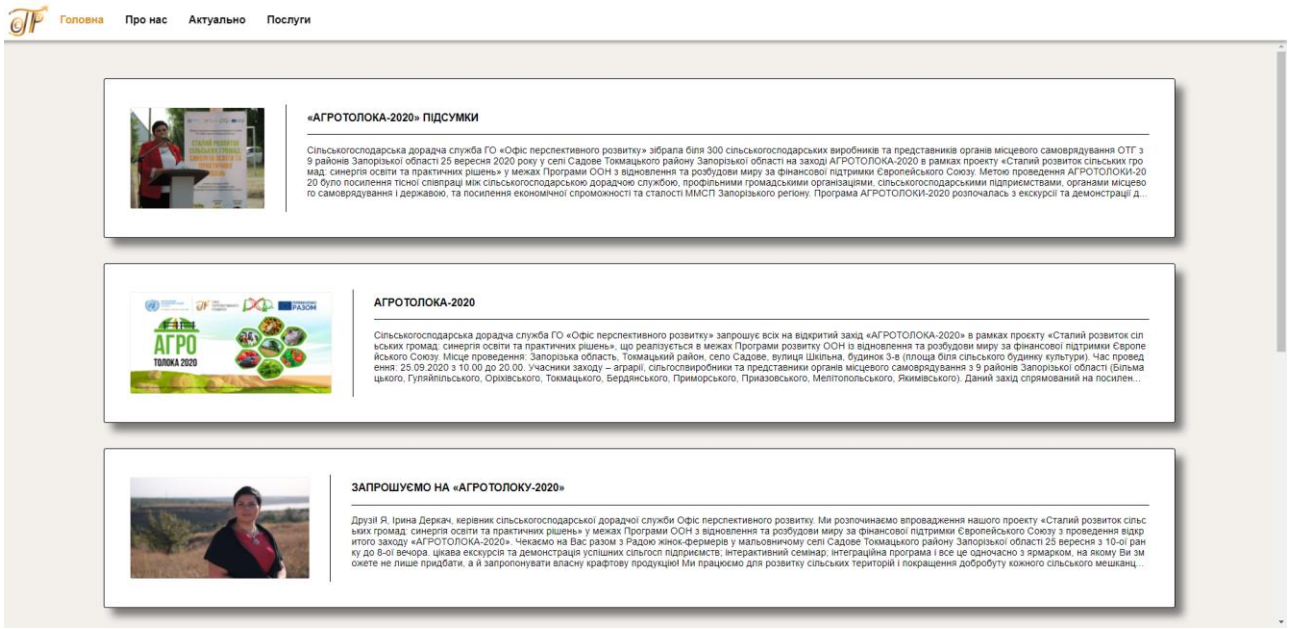

Рис. 2.1 – Загальний вигляд сторінки новин

Рис. 2.2 – Загальний вигляд сторінки новин в мобільній версії

Сторінка новин включає в себе такі основні компоненти як:

1. Хедер з навігацією по сайту;
2. Основну частину з переліком новин, та навігацією по сторінкам новин конкретної категорії;
3. Футер.

2.3 Хедер сайту

Хедер сайту представляє з себе панель з навігацією по розділам сайту.

Рис. 2.3 – Хедер сайту з вибраною вкладкою «Головна»

Нижче наведено зображення хедеру сайту с вибраною вкладкою «Про нас».

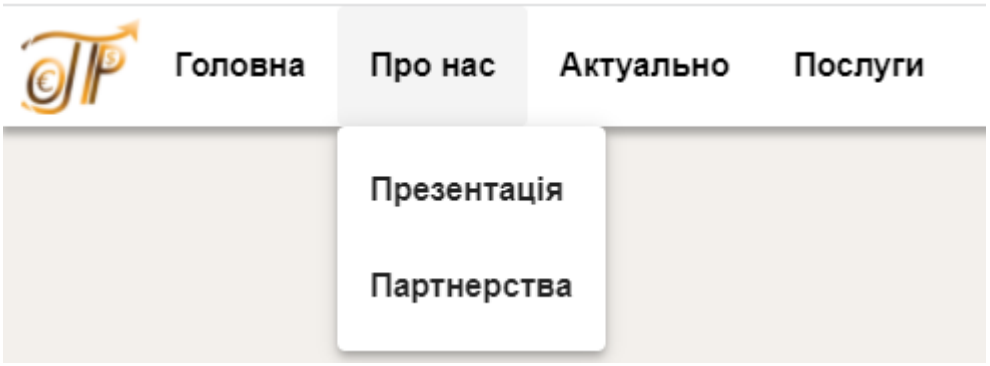

Рис. 2.4 – Хедер сайту з вибраною вкладкою «Про нас»

Нижче наведено зображення хедеру сайту с вибраною вкладкою «Актуально».

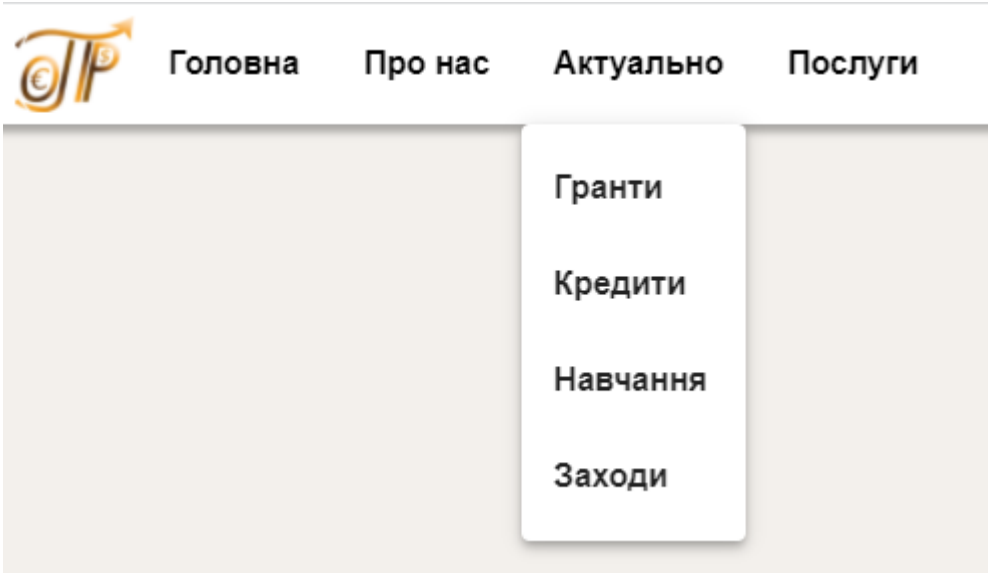

Рис. 2.5 – Хедер сайту з вибраною вкладкою «Актуально»

Нижче наведено зображення хедеру в мобільній версії.

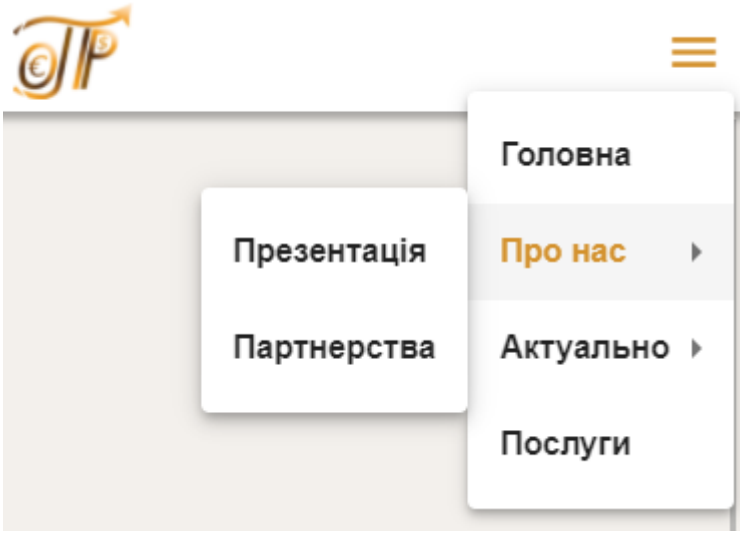

Рис. 2.6 – Хедер сайту в мобільній версії

2.4 Основна частина сторінки новин

Основна частина сторінки новин представляє з себе перелік блоків-новин, що можуть бути відкриті. Кожен з таких блоків має анімацію, що запускається при наведенні мишки на нього.

Рис. 2.7 – Перелік новин на основній частині сторінки новин

Також основна частина сторінки новин включає в себе навігацію по сторінкам новин конкретної категорії.

Рис. 2.8 – Навігація по сторінкам новин конкретної категорії

2.5 Сторінка конкретної новини

Кожна новина зі сторінки новин може бути відкрита, та переглянута.

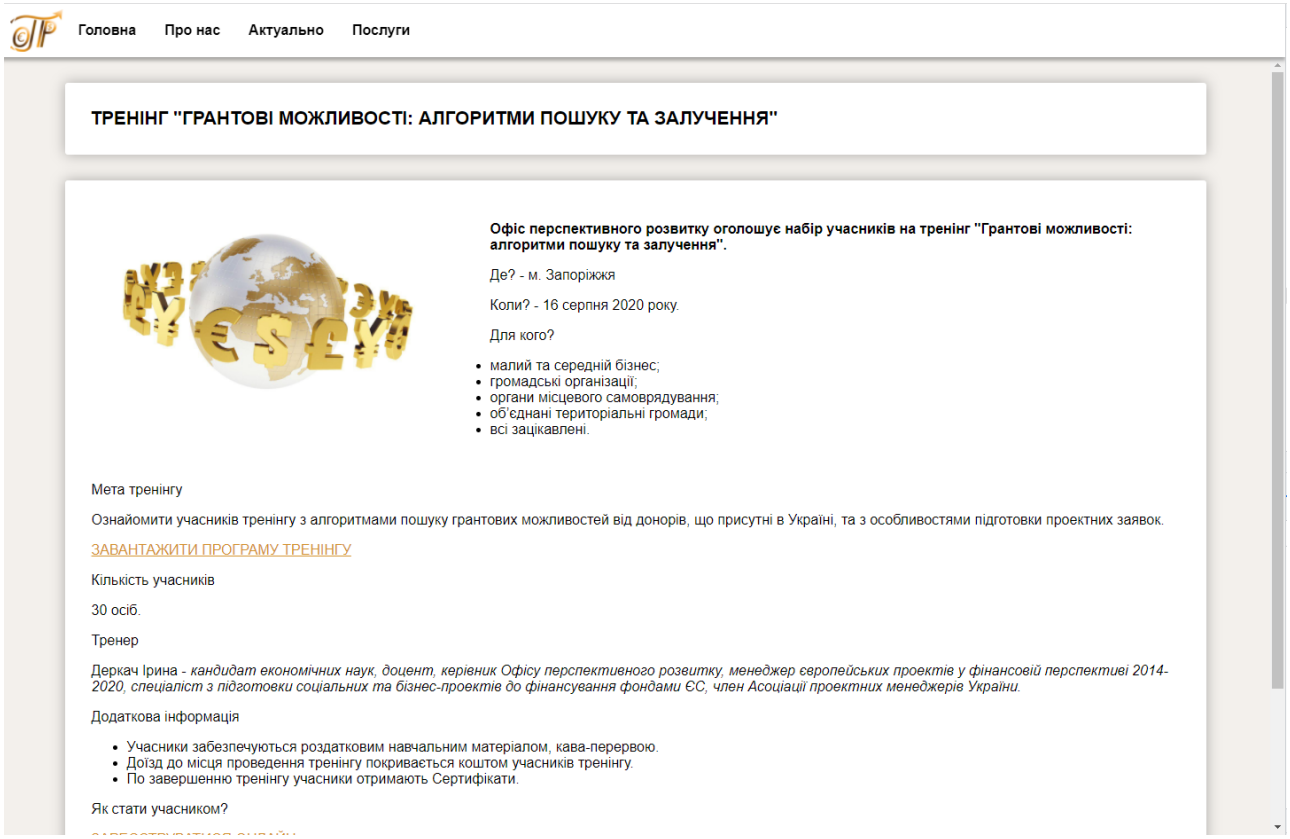

Рис. 2.9 – Загальний вигляд сторінки конкретної новини

Також така сторінка має адаптивний дизайн та може бути переглянута в мобільній версії.

Рис. 2.10 – Мобільна версія сторінки конкретної новини

Ще одним елементом сторінки конкретної новини є переглядач додаткових зображень, який з'являється у випадку, якщо такі прикріплені до новини.

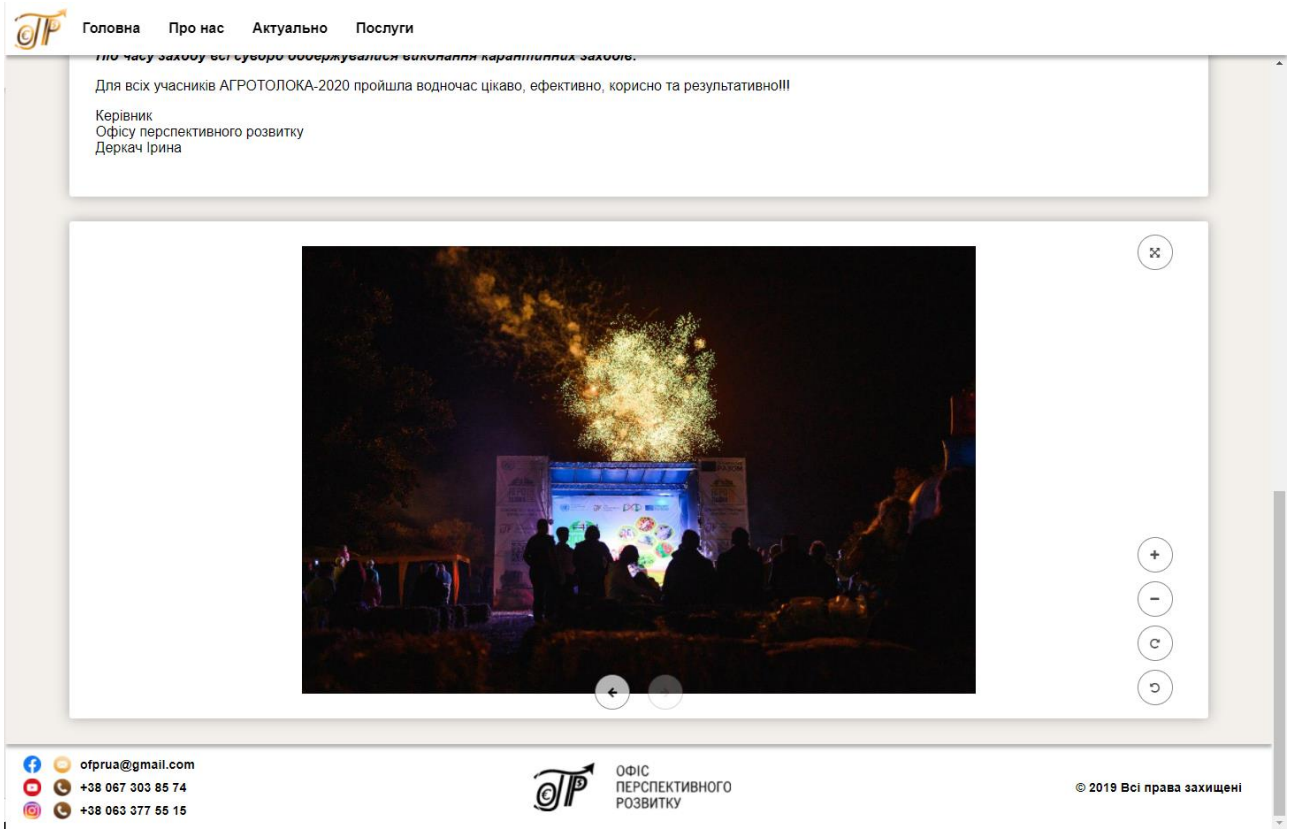

Рис. 2.11 – Переглядач додаткових зображень

Переглядач додаткових зображень також доступний у мобільній версії.

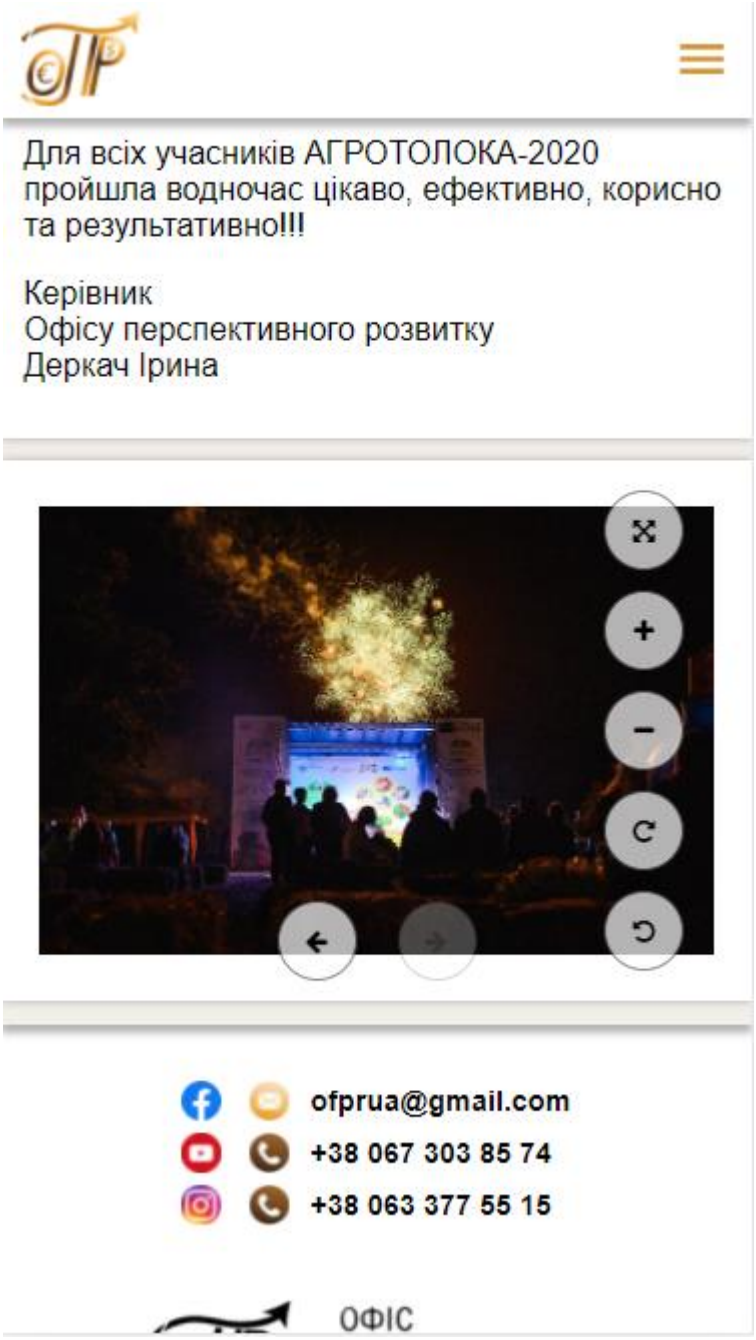

Рис. 2.12 – Переглядач додаткових зображень в мобільній версії

2.6 Футер сайту

Футер сайту включає в себе такі елементи як:

1. Перелік посилань на інші ресурси ГО «Офіс перспективного розвитку»;
2. Контакти;
3. Стилізований логотип;
4. Копірайт.

Рис. 2.13 – Футер сайту

2.7 Сторінка входу

Додаток має базу користувачів, які поділяються за ролями та правами доступу. Отже для авторизації було створено сторінку входу.

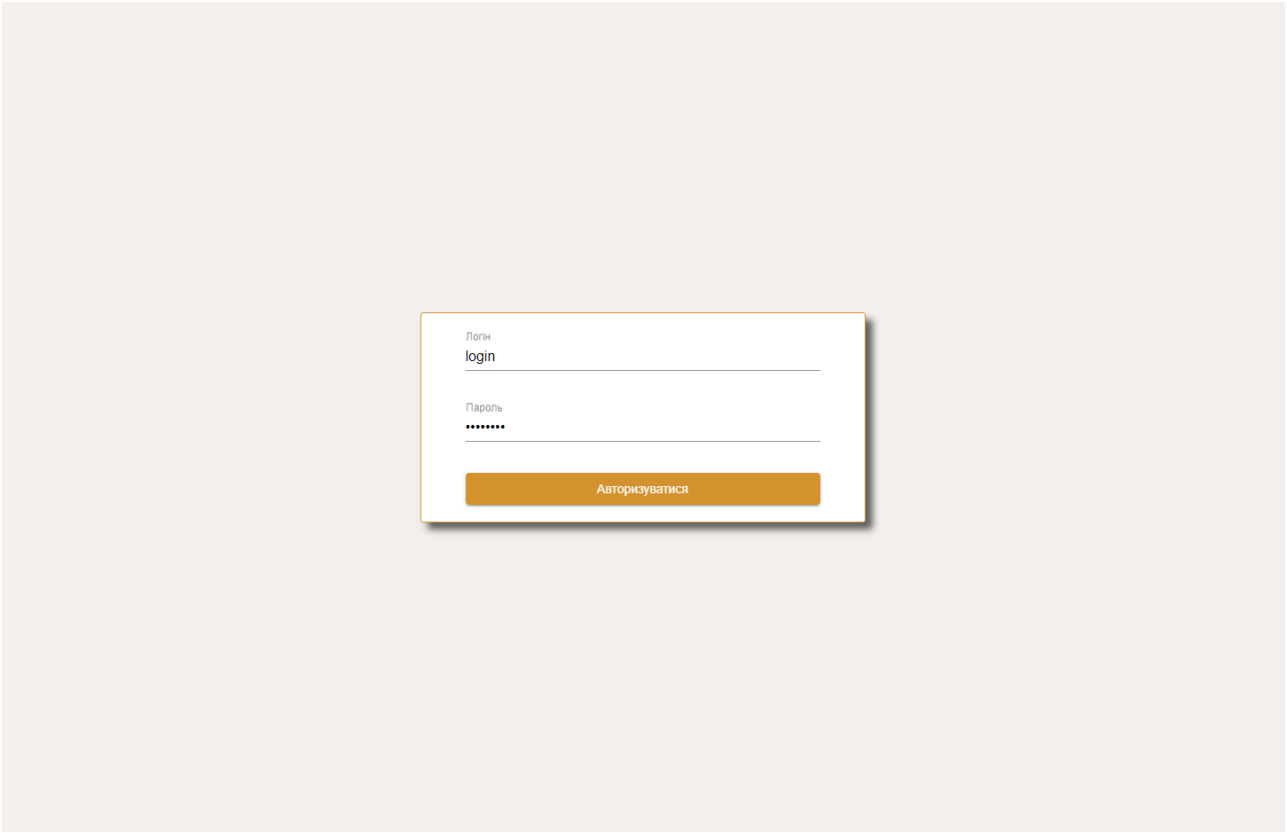

Рис. 2.14 – Сторінка входу

2.8 Адміністративна панель

Додаток має панель адміністратора, через яку користувач з відповідною роллю може додавати, видаляти та редагувати новини. А також генерувати файли robots.txt та sitemap, які використовуються пошуковими службами для коректної індексації сайту.

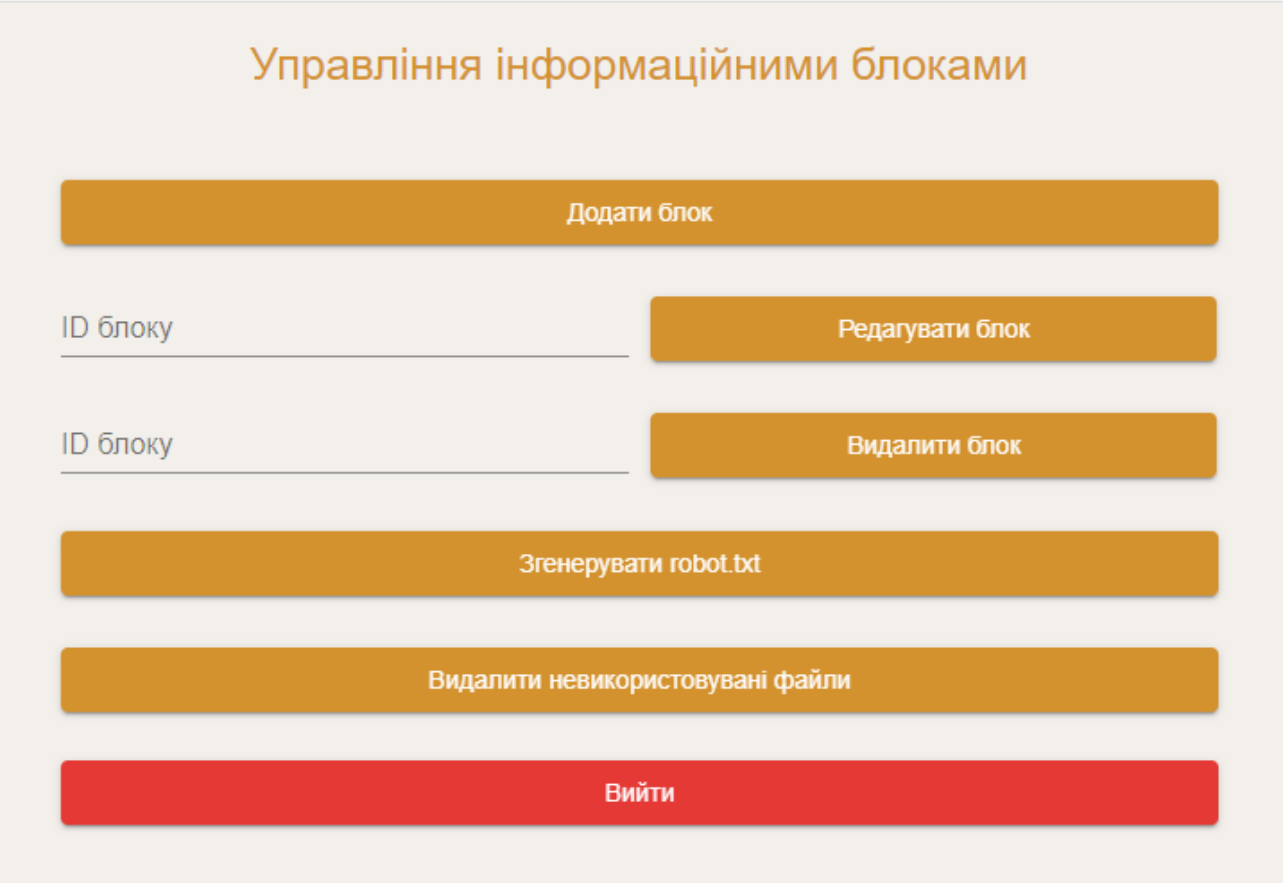

Рис. 2.15 – У правління інформаційними блоками в адміністративній панелі

Адміністративна панель включає в себе панель редагуванню новин, через яку можливо редагувати:

1. Секцію новини;
2. Заголовок новини;
3. Головне зображення;
4. HTML код статті;
5. Додаткові зображення.

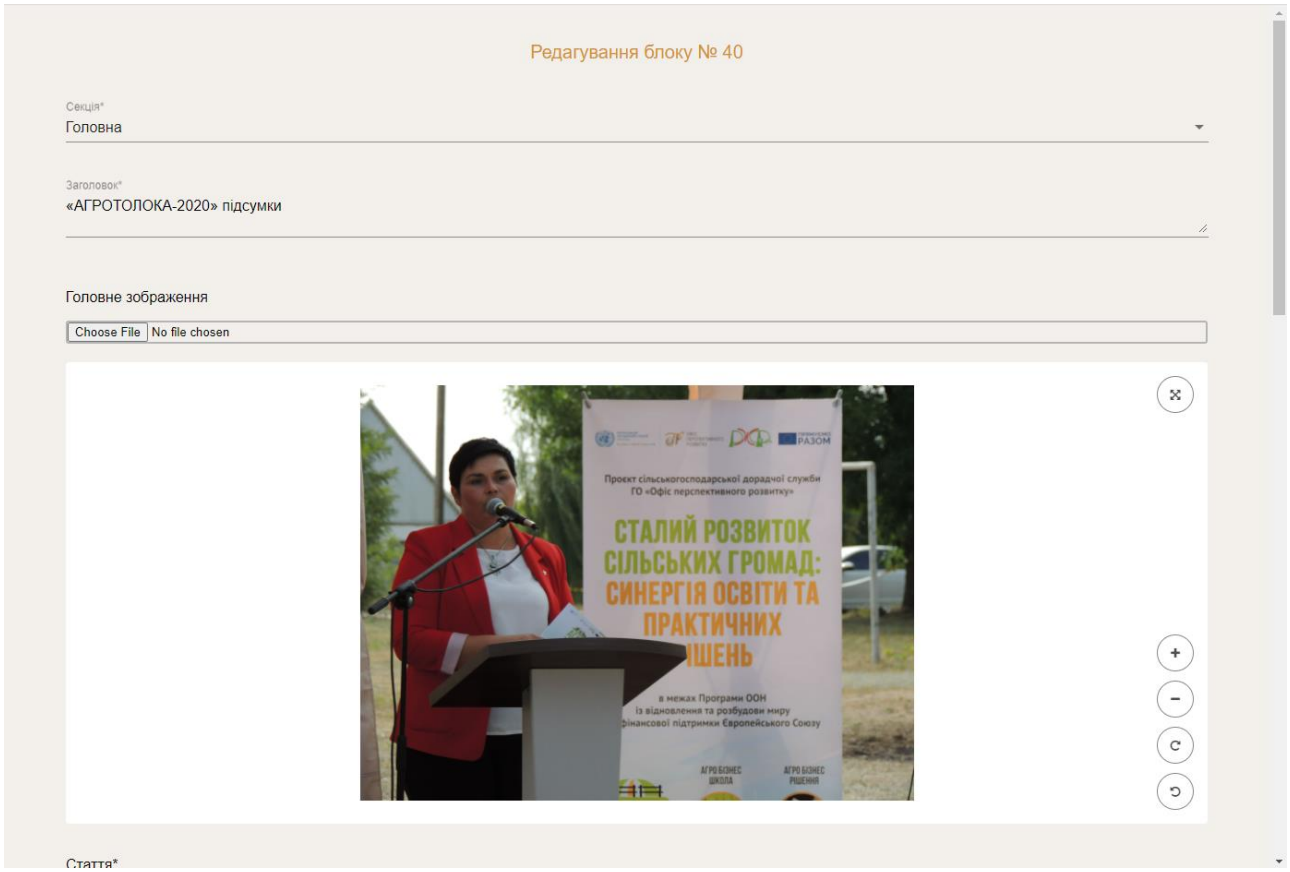

Рис. 2.16 – Елементи редагування секції, заголовку та головного зображення в панелі редагування новин

Нижче наведено елемент редагування HTML коду самої статті.

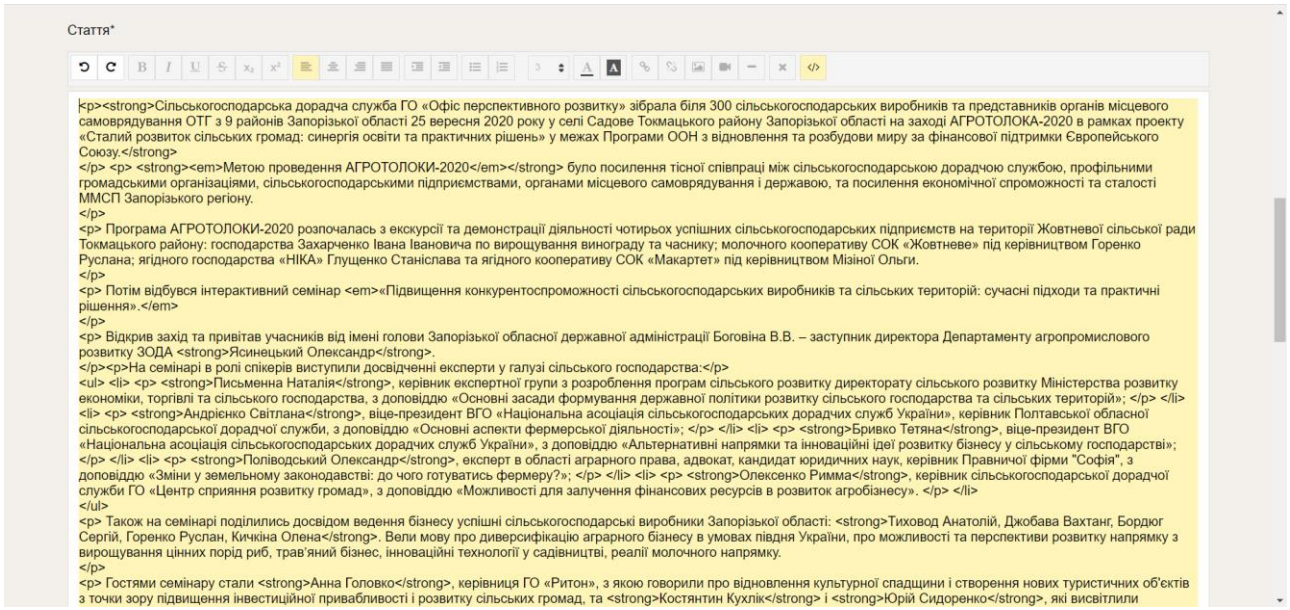

Рис. 2.17 – Елемент редагування HTML коду статті в панелі адміністратора

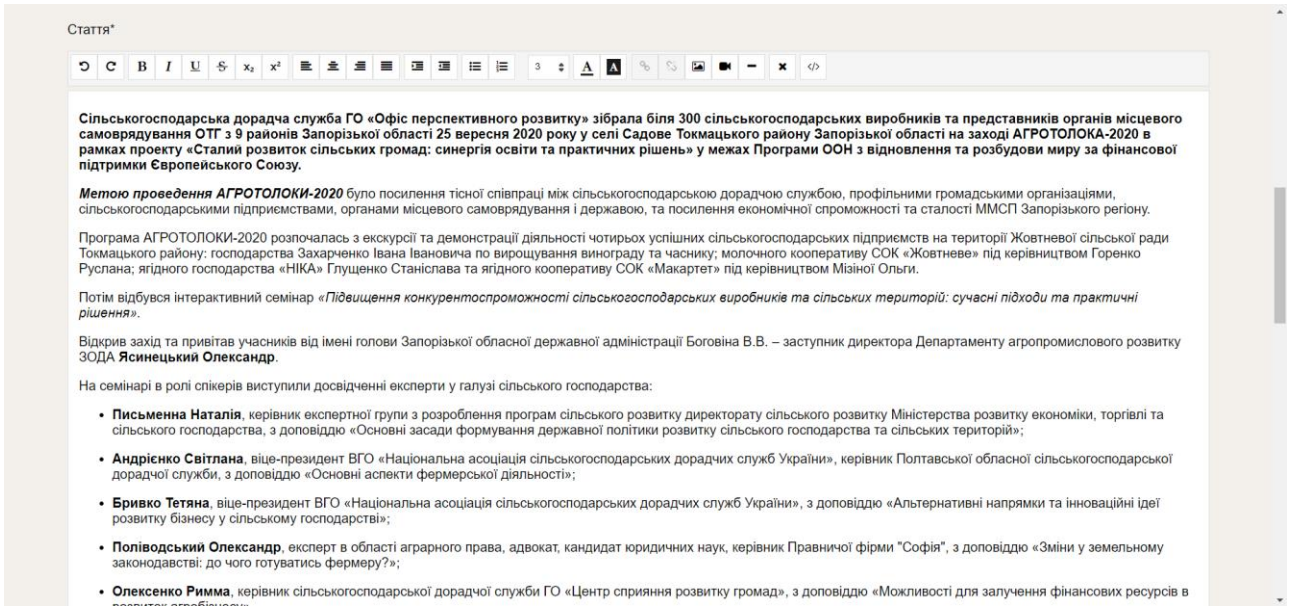

Рис. 2.18 – Елемент редагування статті в панелі адміністратора

Нижче наведено елемент редагування додаткових зображень.

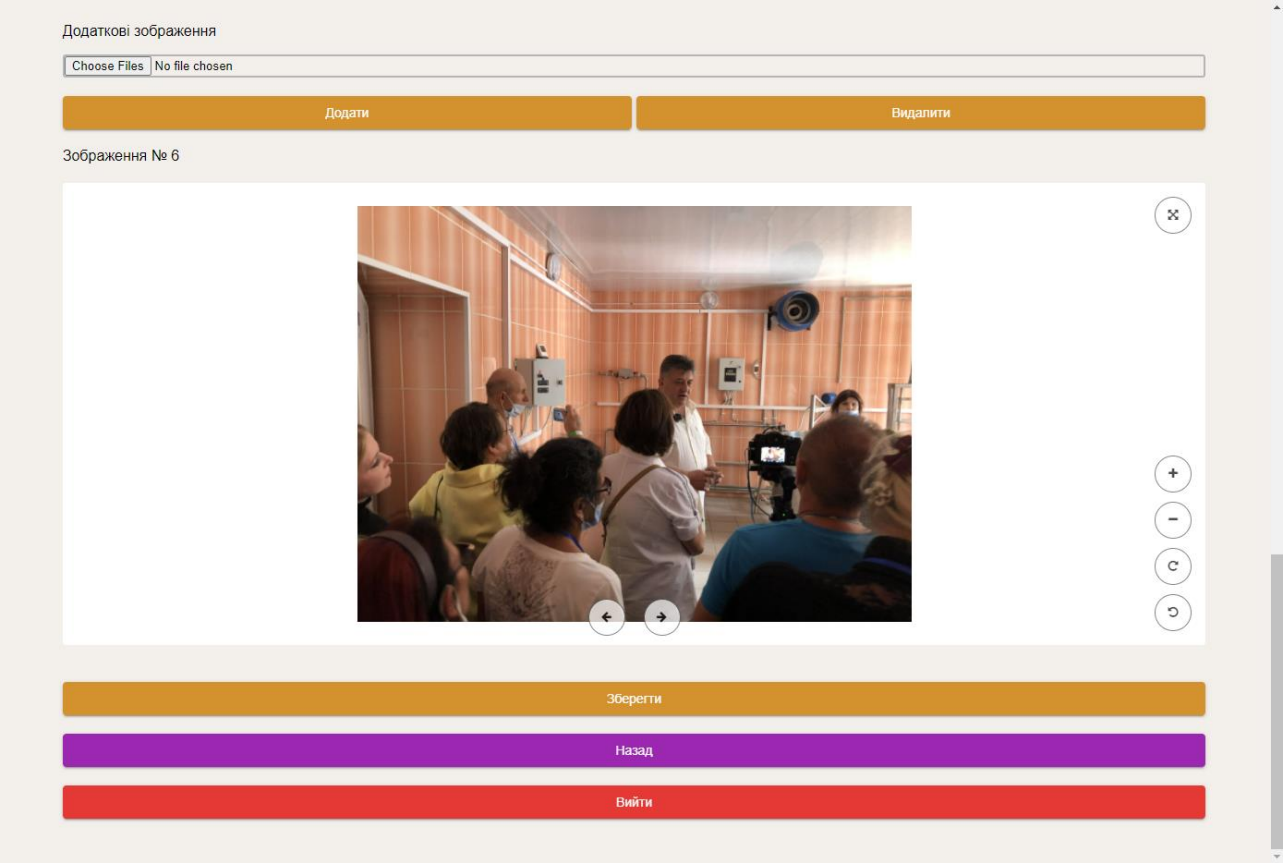

Рис. 2.19 – Елементи редагування додаткових зображень в панелі редагування новин

Також панель редагування новин має адаптивний дизайн та може використовуватися з мобільного пристрою.

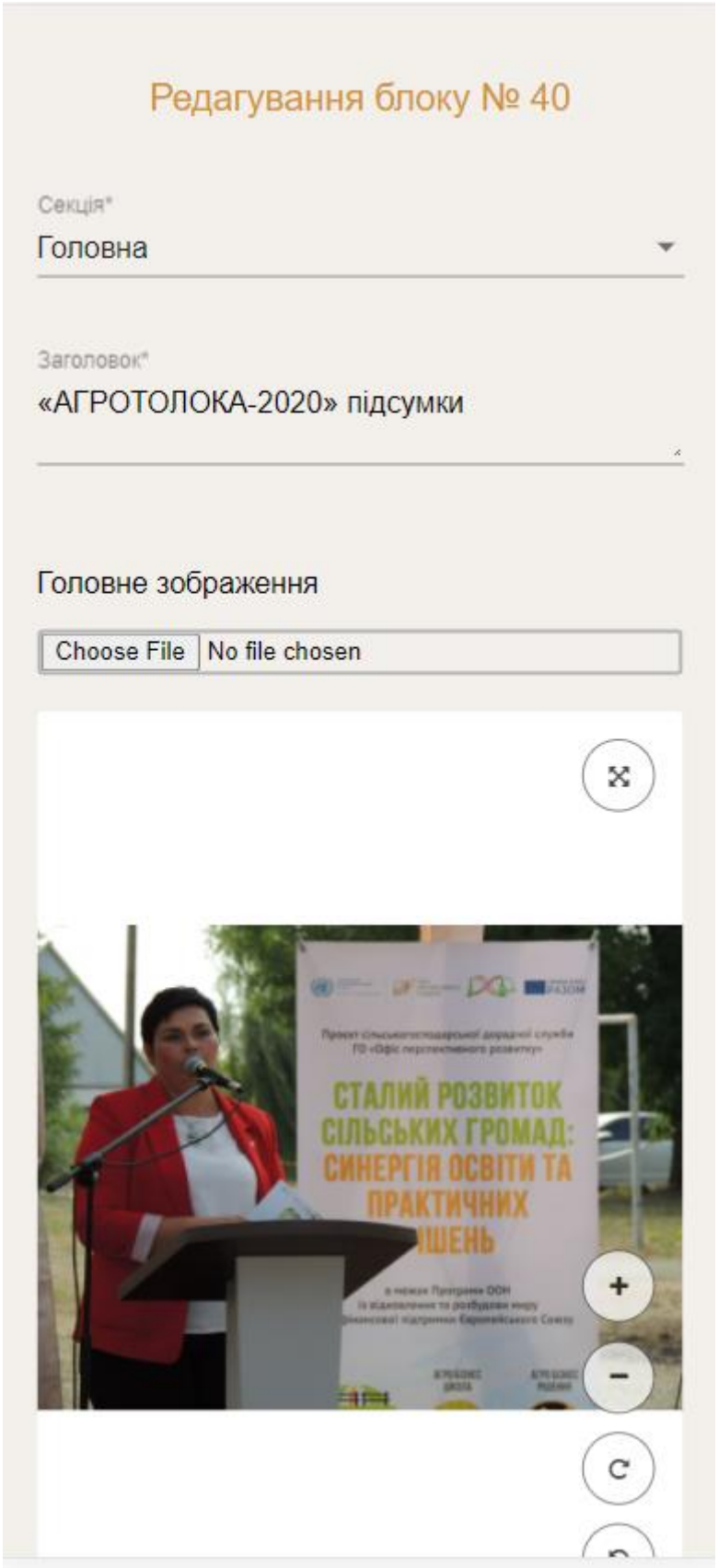

Рис. 2.20 – Мобільний дизайн панелі редагування новин

2.9 Презентація на сторінці «Про нас»

Сайт включає в себе сторінку «Про нас» на якій розміщена презентація ГО «Офіс перспективного розвитку».

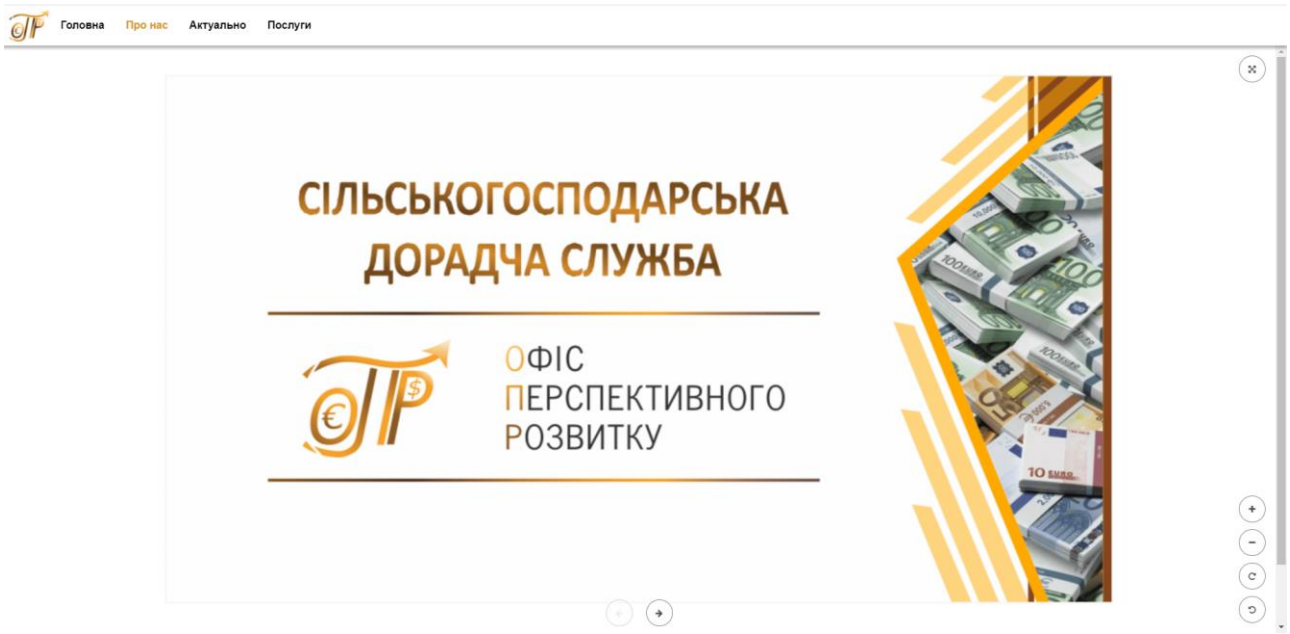

Рис. 2.21 – Презентація на сторінці «Про нас»

2.10 Висновки по другому розділу

У другому розділі, були детально розглянуті можливості SPA фреймворків у створенні гнучких, адаптивних та зручних у використанні інтерфейсів. Особлива увага була приділення можливостям Angular Material застосованого разом з препроцесором SCSS.

РОЗДІЛ 3

АРХІТЕКТУРА ПРОЕКТУ

3.1 Архітектура “клієнт - сервер”

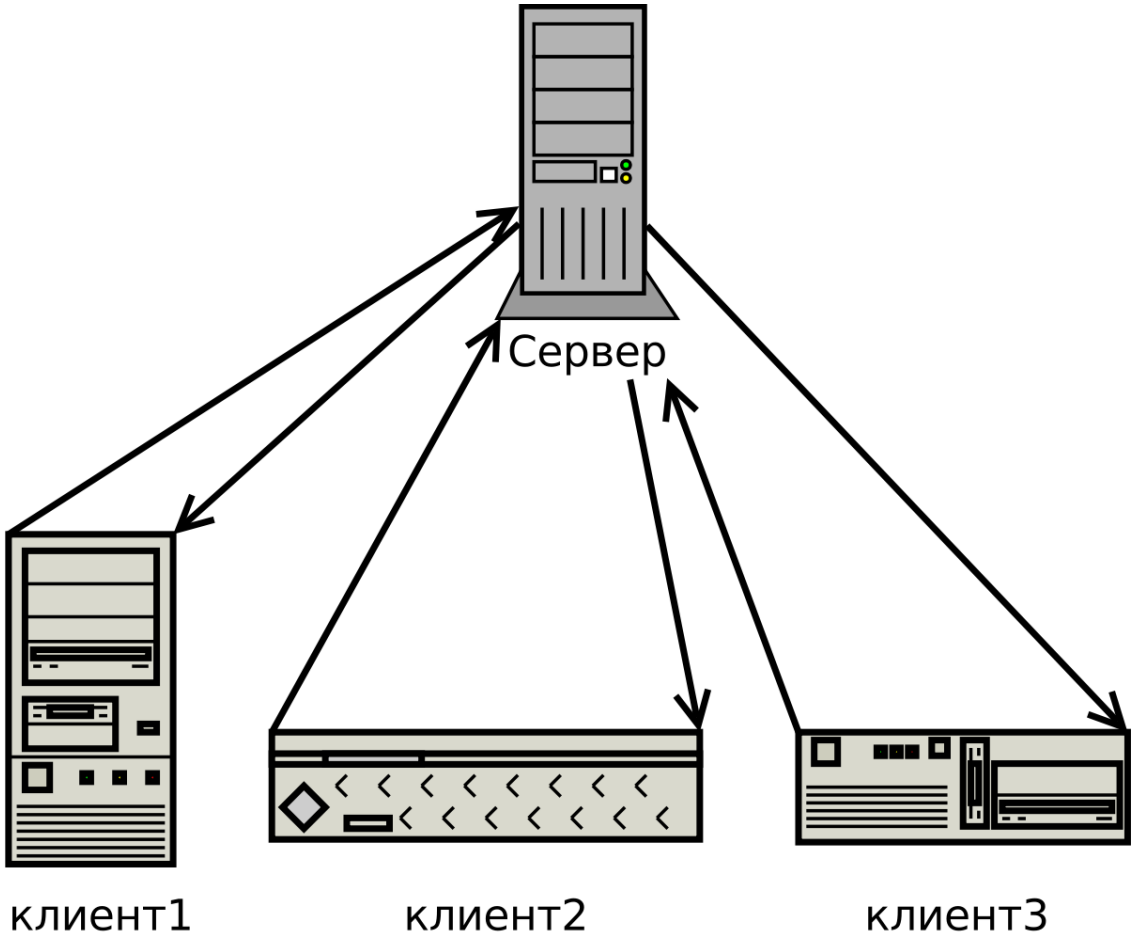

Рис. 3.1 – Умовна схема архітектури “клієнт - сервер”[11]

“Клієнт - сервер” – це обчислювальна модель, у якій обов’язки та завдання системи розподіляються між двома окремими об’єктами: клієнтом і сервером. У цій архітектурі клієнт і сервер спілкуються один з одним через мережу для виконання певних функцій і надання послуг кінцевим користувачам.

Клієнт — це програма або пристрій, який запитує служби або ресурси від сервера. Зазвичай призначений для користувача компонент, який взаємодіє з кінцевим користувачем і забезпечує графічний інтерфейс користувача. Клієнти можуть працювати на різних платформах, таких як настільні комп'ютери, ноутбуки, мобільні пристрої або вбудовані системи.

Сервер — це потужний комп'ютер або мережа комп'ютерів, які відповідають на запити клієнтів і надають необхідні послуги чи ресурси. Він відповідає за керування та розміщення даних, програм і послуг, які потрібні клієнтам. Сервери, як правило, мають більші можливості з точки зору обчислювальної потужності, сховища та підключення до мережі порівняно з клієнтами.

Клієнт і сервер спілкуються один з одним за допомогою мережевого протоколу, наприклад HTTP, HTTPS, TCP/IP або WebSocket. Клієнт ініціює зв'язок, надсилаючи запит на сервер, вказуючи бажану дію чи ресурс. Сервер отримує запит, обробляє його і надсилає відповідь із запитаними даними або підтвердженням дії.

Архітектура клієнт-сервер забезпечує масштабованість і гнучкість. Зі збільшенням кількості клієнтів можна додати більше серверів, щоб задовольнити зростаючий попит. Крім того, сервер можна оновити або замінити, не впливаючи на клієнтські програми, якщо протоколи зв'язку залишаються сумісними. Також така архітектура забезпечує централізований контроль і керування даними та ресурсами. Сервер діє як центральна точка контролю, забезпечуючи цілісність даних, дотримуючись заходів безпеки та керуючи правами доступу. Це дає змогу організаціям впроваджувати надійні заходи безпеки, механізми резервного копіювання та застосовувати узгоджені політики в усій системі.

Архітектура клієнт-сервер широко використовується в різних програмах, включаючи веб-програми, мобільні програми, корпоративні системи, хмарні обчислення та розподілені системи. Він забезпечує масштабований та

ефективний підхід до проектування та розгортання програмних систем, розділяючи питання представлення користувальницького інтерфейсу, обробки на стороні клієнта та керування та обробки даних на стороні сервера.

3.2 Загальні відомості про архітектуру проекту

Проект побудований на основі архітектури “клієнт - сервер” (описаної в главі 3.1), та включає в себе дві типові для подібної архітектури частини, а саме серверну частину – на якій відбувається робота з базою даних та файловою системою, та клієнтську частину – яка представляє з себе інтерфейс користувача, та підтримує роботу проекту.

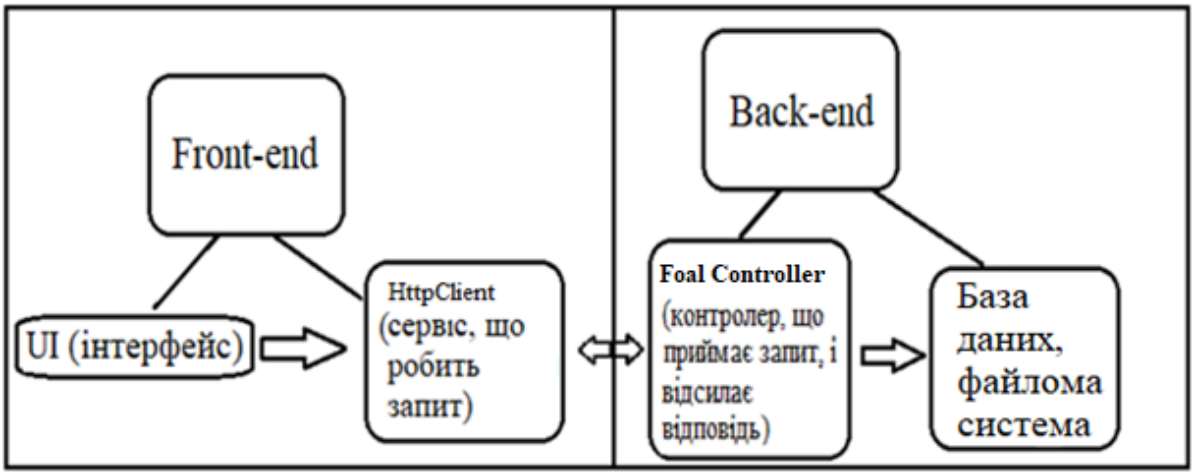

Рис. 3.2 – Схема з основними елементами архітектури проекту

3.3 Загальні відомості про клієнтську частина проекту

Клієнтська частина проекту була розроблена за допомогою мови програмування TypeScript, з використанням фреймворку Angular та бібліотеки Angular Universal, та представляє собою SPA додаток оптимізований під SEO

потреби та має адаптивний дизайн, забезпечений препроцесором таблиці стилів SCSS та UI бібліотекою Angular Material.

Основними модулями клієнтської частина додатку є MainModule та AdminModule. Повний код головного класу AdminModule наведено у додатку А. MainModule включає в себе компоненті сторінок, блогів, презентацій, новин та послуг. AdminModule включає в себе компоненти пов’язані з роботою адміністративної панелі та забезпечує додавання, видалення, редагування та повний менеджмент статей розміщених на сайті та всього контенту пов’язаного з ними.

3.4 Загальні відомості про серверну частину проекту

Серверна частина проекту була розроблена за допомогою мови програмування TypeScript, з використанням фреймворку FoalTS, на платформі NodeJS. Основними контролерами серверної частини є ApiController та OpenApiController (Swagger). ApiController являється кореневим контролером, та включає в себе всі інші суб-контролери. OpenApiController (Swagger) використовується для роботи Swagger бібліотеки, що надає можливість створення інтерактивної документації API проекту.

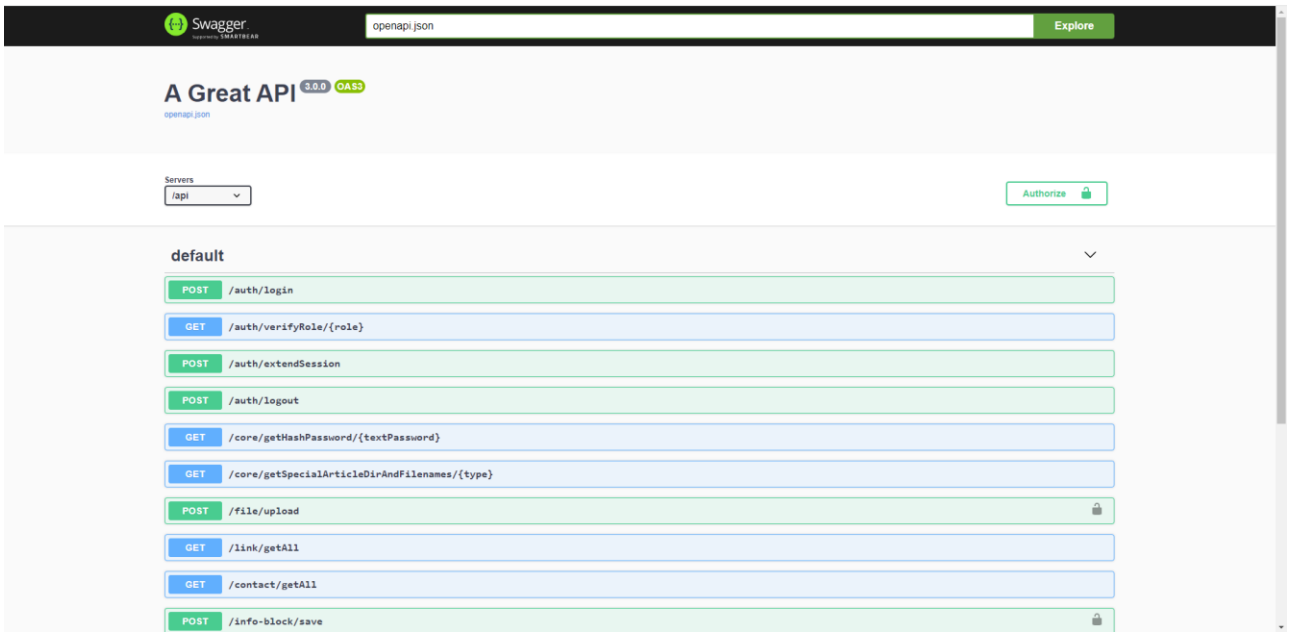

Рис. 3.3 – Інтерактивна документації згенерована завдяки Swagger бібліотеці

3.5 Опис модуля AppRoutingModule клієнтської частини

AppRoutingModule – кореневий модуль, що відповідає за навігацію на клієнтській частині, та на основі URL запита визначає які компоненти та які модулі будуть завантажені.

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { AdminGuard } from './modules/admin/guards/admin.guard';
import { LoginComponent } from './modules/core/components/login/login.component';
import { UrlRouteConst } from './modules/core/consts/route/url-route.const';

export const appRoutes: Routes = [
  {
    path: UrlRouteConst.LOGIN,
    component: LoginComponent
  },
  {
    path: UrlRouteConst.ADMIN,
    loadChildren: () => import('./modules/admin/admin.module').then(m => m.AdminModule),
    canActivate: [AdminGuard]
  },
  {
    path: '',
    loadChildren: () => import('./modules/main/main.module').then(m => m.MainModule),
  },
];

@NgModule({
  imports: [RouterModule.forRoot(appRoutes, { relativeLinkResolution: 'legacy' })],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Рис. 3.4 – Код модуля AppRoutingModule

У наведеному фрагменті коду визначено модуль Angular під назвою AppRoutingModule. Цей модуль відповідає за налаштування функціональності маршрутизації веб-додатку. Він імпортує необхідні модулі та визначає маршрути для різних компонентів.

Необхідні модулі та класи імпортуються з фреймворку Angular і файлів програми.

appRoutes - це масив конфігурацій маршрутів. Кожна конфігурація маршруту визначає шлях і відповідний компонент або модуль, які завантажуються під час доступу до цього шляху.

Перша конфігурація маршруту призначена для сторінки входу. Він відображає шлях, визначений в UrlRouteConst.LOGIN, на LoginComponent.

Друга конфігурація маршруту призначена для розділу адміністратора. Він відображає шлях, визначений в UrlRouteConst.ADMIN, до AdminModule за допомогою відкладеного завантаження. AdminGuard використовується для захисту цього маршруту та дозволяє доступ лише авторизованим адміністраторам.

Третя конфігурація маршруту — це маршрут за замовчуванням, який завантажує MainModule, коли шлях не вказано. Зазвичай це цільова сторінка або головний розділ програми.

Декоратор NgModule: декоратор @NgModule використовується для визначення модуля AppRoutingModule. Він визначає імпортовані модулі та експортує RouterModule з бібліотеки маршрутизаторів Angular.

RouterModule.forRoot(): цей метод використовується для налаштування маршрутизатора кореневого рівня для програми. Він приймає масив appRoutes і додаткові параметри конфігурації як аргументи. Параметр relativeLinkResolution: 'legacy' надається тут для зворотної сумісності.

RouterModule експортується з модуля, що дозволяє іншим модулям програми імпортувати та використовувати налаштовані маршрути.

Загалом, код налаштовує конфігурацію маршрутизації для програми Angular, вказуючи різні шляхи та компоненти/модулі, які завантажуються під час доступу до цих шляхів. Він також включає охорону для захисту певних маршрутів і гарантує, що лише авторизовані користувачі можуть отримати до них доступ.

3.6 Опис модуля MainRoutingModule клієнтської частини

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { BaseComponent } from './components/base/base.component';
import { UrlRouteConst } from '../core/consts/route/url-route.const';

const routes: Routes = [
  {
    path: '',
    component: BaseComponent,
    children: [
      {
        path: UrlRouteConst.INFO_BLOCK,
        loadChildren: () => import('../info-block/info-block.module').then(m => m.InfoBlockModule),
        data: { preload: true }
      },
      {
        path: UrlRouteConst.SPECIAL_ARTICLE,
        loadChildren: () => import('../special-article/special-article.module').then(m => m.SpecialArticleModule),
        data: { preload: true }
      },
      {
        path: '**',
        redirectTo: UrlRouteConst.INFO_BLOCK,
        pathMatch: 'full'
      },
    ]
  }
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class MainRoutingModule { }
```

Рис. 3.5 – Код модуля MainRoutingModule

У наведеному фрагменті коду визначено модуль Angular під назвою MainRoutingModule. Цей модуль відповідає за налаштування функцій маршрутизації, специфічних для модулю Main.

Routes - це масив конфігурацій маршрутів, специфічних для модулю Main.

Конфігурація основного маршруту призначена для базового компонента, який діє як макет або контейнер для дочірніх компонентів. Він відображає кореневий шлях на базовий компонент.

Усередині BaseComponent є дочірні маршрути, налаштовані за допомогою властивості «children». Кожен дочірній маршрут відповідає певній функції або модулю.

Перший дочірній маршрут призначений до модулю InfoBlock. Він відображає шлях, визначений у UrlRouteConst.INFO_BLOCK, до InfoBlockModule за допомогою відкладеного завантаження. Властивість "preload" має значення true в об'єкті даних, що вказує на те, що цей модуль має бути попередньо завантажений для кращої продуктивності.

Другий дочірній маршрут призначений для функції "SpecialArticle". Він відображає шлях, визначений в UrlRouteConst.SPECIAL_ARTICLE, до SpecialArticleModule за допомогою відкладеного завантаження. Подібно до попереднього маршруту, для властивості.

Третій маршрут — це маршрут із підстановкою (**), який відповідає будь-яким невизначеним шляхам. Він перенаправляє до функції "InfoBlock" за замовчуванням, як зазначено у властивості redirectTo. Властивість pathMatch має значення «full», щоб переконатися, що для перенаправлення збігається весь шлях.

Загалом, код налаштовує конфігурацію маршрутизації для розділу «Main» програми Angular. Він визначає базовий компонент як макет і визначає дочірні маршрути для різних функцій модулі Main. Відкладене завантаження використовується для завантаження модулів функцій на вимогу, а маршрут із шаблоном підстановки надається для обробки невизначених шляхів і перенаправлення.

3.7 Опис контролера ApiController серверної частини

ApiController является кореневим контролером, та включає в себе всі інші суб-контролери серверної частини.

```
import { ApiInfo, ApiServer, controller } from '@foal/core';
import { InfoBlockController } from './info-block/info-block.controller';
import { CoreController } from './core/core.controller';
import { AuthController } from './auth/auth.controller';
import { FileController } from './file/file.controller';
import { ContactController } from './contact/contact.controller';
import { LinkController } from './link/link.controller';

@ApiInfo({
    title: 'A Great API',
    version: '3.0.0'
})
@ApiServer({
    url: '/api'
})
export class ApiController {
    subControllers = [
        controller('/auth', AuthController),
        controller('/core', CoreController),
        controller('/file', FileController),
        controller('/link', LinkController),
        controller('/contact', ContactController),
        controller('/info-block', InfoBlockController)
    ];
}
```

Рис. 3.6 – Код контролера ApiController

ApiController використовується для визначення та організації набору контролерів API.

Декоратор @ApiInfo використовується для анотації класу ApiController метаданими про API.

Декоратор @ApiServer використовується для анотації класу ApiController і вказує, що визначені в ньому кінцеві точки API обслуговуватимуться за маршрутом “/api”.

Властивість subControllers є масивом, який містить екземпляри інших класів контролерів. Кожен контролер пов’язаний із певним маршрутом або шляхом кінцевої точки.

У прикладі показано, як декілька екземплярів контролера призначаються масиву subControllers за допомогою функції controller(). Кожен екземпляр пов’язаний із певним шляхом і відповідним класом контролера. Наприклад, controller('/auth', AuthController) пов'язує шлях маршруту '/auth' з класом AuthController. Подібним чином інші екземпляри контролера пов’язані з відповідними шляхами маршрутів.

Клас ApiController діє як контейнер або організатор для кількох субконтролерів у межах API. Він надає централізоване місце для керування та налаштування цих контролерів, включаючи пов’язані з ними маршрути, базову URL-адресу та інформацію про API. Організація контролерів таким чином допомагає підтримувати модульний і структурований підхід до створення API.

3.8 Опис контролера OpenApiController серверної частини

OpenApiController (Swagger) використовується для роботи Swagger бібліотеки, що надає можливість створення інтерактивної документації API проекту.

```
import { SwaggerController } from '@foal/swagger';
import { ApiController } from '../api/api.controller';

export class OpenApiController extends SwaggerController {
    options = { controllerClass: ApiController };
}
```

Рис. 3.7 – Код контролера OpenApiController

3.9 Базовий компонент клієнтської частини

```
<div class="base-wrapper">

    <header>
        <div class="logo">
            <a routerLink="{{logoHref}}" class="img-logo-link">
                <img [src]="opdLogoMiniPath" alt="{{oprLiteral}}" class="img-logo">
            </a>
            <div class="menu-header">
                <app-menu-header></app-menu-header>
            </div>
        </div>
    </header>

    <div id="{{mainWrapperId}}" class="main-wrapper">

        <main>
            <router-outlet></router-outlet>
        </main>

        <footer>
            <div class="contacts-and-links-wrapper">
                <div *ngIf="links$ | async as links" class="contact-rows">
                    <div *ngFor="let link of links">
                        <a *ngIf="link" [href]="link.href" target="_blank" class="contact-link"
                            [ngClass]="'background-image-' + link.type + ' ' + (!link.href ? 'not-active' : '')"> 
                        </a>
                    </div>
                </div>
                <div *ngIf="contacts$ | async as contacts" class="contact-rows">
                    <div *ngFor="let contact of contacts">
                        <a *ngIf="contact" [href]="contact.href" class="contact-link"
                            [ngClass]="'background-image-' + (contact.href?.slice(0, contact.href?.indexOf(':')) === 'tel' ? 'phone' : 'mail')">{{contact?.value}}
                        </a>
                    </div>
                </div>
            </div>
            <div class="img-logo-wrapper">
                <img [src]="opdLogoPath" alt="{{oprLiteral}}" class="img-full-logo">
            </div>
            <div class="footer-text">© 2019 - {{currentYear}} Всі права захищені</div>
        </footer>

    </div>

    <app-ngx-spinner-wrapper [settings]="spinnerSettings"></app-ngx-spinner-wrapper>
```

Рис. 3.8 – HTML код компоненту BaseComponent

Дана розмітка представляє шаблон HTML, який визначає структуру та макет всіх сторінок додатку за виключенням сторінок, що належать до адміністративної панелі.

Давайте розберемо це:

1. : це крайній елемент контейнера з класом "base-wrapper". Він служить оболонкою для всього вмісту веб-сторінки;
2. <header>: цей розділ представляє заголовок веб-сторінки;
4. : цей div містить логотип веб-сайту;

5. <a routerLink="{{logoHref}}" class="img-logo-link">: це елемент посилання з посиланням на маршрутизатор, визначеним атрибутом "routerLink". Він огортає зображення логотипу;

6. <img [src]="opdLogoMiniPath" alt="{{oprLiteral}}" class="img-logo">: це елемент зображення з джерелом, указаним в атрибуті "[src]". Він відображає логотип веб-сайту;

2. : цей div містить меню заголовка веб-сайту, яке реалізовано як спеціальний компонент під назвою "app-menu-header";

3. : цей div представляє основну оболонку вмісту веб-сторінки. Він має атрибут "id" і клас "main-wrapper";

4. <main>: цей розділ представляє основну область вмісту веб-сторінки;

5. <router-outlet></router-outlet>: це елемент-заповнювач, який буде динамічно замінено вмістом компонента на основі поточного маршруту;

6. <footer>: цей розділ представляє нижній колонтитул веб-сторінки.

3.10 SEO оптимізація клієнтської частини

SPA фреймворки як правило мають проблеми з SEO, оскільки пошукові роботи традиційно покладаються на відтворення на стороні сервера для аналізу та індексування веб-сторінок. Однак SPA фреймворки переважно відображають вміст на стороні клієнта, що ускладнює пошуковим системам точну інтерпретацію та індексацію вмісту. Але майже кожен SPA фреймворк має інструмент для вирішення цієї проблеми. У React це бібліотека Next.JS. У Angular це бібліотека Angular Universal.

```
export async function app(): Promise<express.Express> {
  const server = express();
  const distFolder = join(process.cwd(), 'dist/browser');
  const indexHtml = existsSync(join(distFolder, 'index.original.html')) ? 'index.original.html' : 'index';

  // Add proxy to other back-end
  const ssrServerConf: ISSRServerConf = await getSSRServerConf();
  const proxyMiddleware = createProxyMiddleware({
    target: ssrServerConf.proxyPrefix,
    changeOrigin: true
  });
  ssrServerConf.linksNeedRedirect.forEach((link: string) => server.use(link, proxyMiddleware));

  server.engine('html', ngExpressEngine({
    bootstrap: AppServerModule
  }));

  server.set('view engine', 'html');
  server.set('views', distFolder);

  // Serve static files from /browser
  server.get('*.*', express.static(distFolder, {
    maxAge: '1y'
  }));

  // All regular routes use the Universal engine
  server.get('*', (req, res) => {
    res.render(indexHtml, { req, providers: [{ provide: APP_BASE_HREF, useValue: req.baseUrl }] });
  });

  return server;
}

async function run() {
  const port = process.env.PORT || (await getSSRServerConf()).port;

  // Start up the Node server
  const server = await app();
  server.listen(port, () => {
    console.log(`Node Express server listening on http://localhost:${port}`);
  });
}
```

Рис. 3.9 – Код SEO серверу Angular Universal

Цей код відповідає за налаштування серверу бібліотеки Angular Universal, який забезпечує рендеринг на стороні сервера (SSR) для програми Angular.

Покроковий розбір того, що робить код:

1. Визначає асинхронну функцію app(), яка створює примірник сервера Express і налаштовує його;

2. Налаштовує проміжне програмне забезпечення проксі-сервера для обробки запитів, які потрібно перенаправляти на інший внутрішній сервер;

3. Налаштовує програму Express для відтворення шаблонів HTML;

4. Встановлює механізм перегляду на "html" і вказує папку переглядів, у якій розташовано відтворені шаблони HTML;

5. Обслуговує статичні файли з каталогу 'dist/browser';

6. Обробляє всі регулярні маршрути, відтворюючи додаток Angular за допомогою ngExpressEngine і передаючи запит і базову URL-адресу як параметри;

7. Визначає асинхронну функцію run(), яка запускає сервер Node шляхом виклику функції app() і прослуховує вказаний порт;

8. Експортує необхідні модулі та функції для візуалізації Angular Universal;

9. Виконує функцію run() для запуску сервера.

Цей код дозволяє візуалізувати додатки Angular на стороні сервера, забезпечуючи кращу продуктивність початкового завантаження сторінки та покращуючи пошукову оптимізацію (SEO).

3.11 Висновки по третьому розділу

У третьому розділі була ретельно розглянута архітектура та основні модулі як з клієнтської так із серверної частини веб-додатку. Архітектурою веб-додатку є обчислювальна модель “клієнт-сервер”. Клієнтська частина складена з динамічно маршрутизованих модулів та компонентів. Серверна частина складається з розбитих на модулі субконтролерів, кожен з яких відповідає за свою частину API.

З серверної частини були розглянуті такі модулі як:

1. OpenApiController;

2. ApiController.

З клієнтської частини були розглянуті такі модулі та компоненти як:

1. AppRoutingModule;
2. MainRoutingModule;
3. BaseComponent;
4. SEO сервер розгорнутий завдяки бібліотеці Angular Universal.

Тож як бачимо SPA фреймворки дозволяють будувати гнучку та масштабовану архітектуру з поділом коду на модулі та компоненти, а також можуть бути використані разом з другими технологіями, як то серверні API, тощо.

ВИСНОВОК

У ході виконання кваліфікаційної бакалаврської роботи, на прикладі створення веб-додатку, що представляє собою візитний сайт з блогами, статтями та презентаціями для ГО «Офіс перспективного розвитку», що включає в себе адміністративну панель, що дозволяє редагувати, видаляти, та додавати контент, було досліджено можливості SPA фреймворків.

Дослідження показало слушність використання SPA фреймворків, які пропонують сучасний та ефективний підхід до створення інтерактивних веб-додатків. Ці фреймворки надають розробникам потужні інструменти та функції для покращення процесу розробки. Вони дозволяють створювати динамічні адаптивні, зручні у використанні, зрозумілі та чисті інтерфейси користувача.

Результати дослідження свідчать про те, що SPA фреймворки сприяють покращенню взаємодії з користувачем, забезпечуючи безперебійний інтерактивний перегляд, усуваючи необхідність частого перезавантаження сторінок. Це призводить до створення швидших і більш чутливих додатків, що в кінцевому підсумку сприяє підвищенню залученості та задоволеності користувачів.

Крім того, інфраструктури SPA пропонують модульні та компонентні архітектури, що полегшує повторне використання коду та зручність обслуговування. Серед інших функціональних можливостей розробники можуть ефективно керувати станом програми, реалізовувати маршрутизацію та інтегрувати зовнішні API. Це сприяє масштабованості та розширюваності, дозволяючи розробляти складні та багатофункціональні веб-додатки.

Однак важливо визнати, що SPA фреймворки також викликають проблеми, такі як час початкового завантаження, оптимізація SEO та сумісність зі старими браузерами. Ці проблеми вимагають ретельного розгляду та впровадження відповідних методів і найкращих практик.

Загалом дослідження показує, що SPA фреймворки пропонують практичні можливості для створення веб-додатків, які відповідають вимогам сучасних користувачів. Використовуючи можливості цих фреймворків, розробники можуть забезпечувати багатий і захоплюючий досвід користувача, зберігаючи організацію коду і масштабованість.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Фреймворк програмування Angular [Електронний ресурс] – Режим доступа: URL: https://ru.wikipedia.org/wiki/Angular_(%D1%84%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) – Дата доступа: 19.03.2023 г.;

2. Фреймворк програмування Angular [Електронний ресурс] – Режим доступа: URL: https://medium.com/@mlbors/an-overview-of-angular-3ccd2950648e – Дата доступа: 20.03.2023 г.;

3. Бібліотека Angular Universal [Електронний ресурс] – Режим доступа: URL: https://angular.io/guide/universal – Дата доступа: 10.05.2023 г.;

4. Мова програмування TypeScript [Електронний ресурс] – Режим доступа: URL: https://en.wikipedia.org/wiki/TypeScript – Дата доступа: 01.05.2023 г.;

5. СУБД MySQL [Електронний ресурс] – Режим доступа: URL: https://ru.wikipedia.org/wiki/MySQL – Дата доступа: 03.03.2023 г.;

6. Платформа NodeJS [Електронний ресурс] – Режим доступа: URL: https://en.wikipedia.org/wiki/Node.js – Дата доступа: 06.04.2023 г.;

7. IDE Visual Studio Code [Електронний ресурс] – Режим доступа: https://en.wikipedia.org/wiki/Visual_Studio_Code – Дата доступа: 11.05.2023 г.

8. Фреймворк програмування FoalTS [Електронний ресурс] – Режим доступа: https://github.com/FoalTS/foal – Дата доступа: 15.05.2023 г.;

9. IDE MySQL Workbench [Електронний ресурс] – Режим доступа: https://en.wikipedia.org/wiki/MySQL_Workbench – Дата доступа: 05.04.2023 г.;

10. Система контроля версій GitLab [Електронний ресурс] – Режим доступа: https://en.wikipedia.org/wiki/GitLab – Дата доступа: 27.04.2023 г.;

11. Архітектура “клієнт-сервер” [Електронний ресурс] – Режим доступа: URL: https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D

1%82_%E2%80%94_%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80 – Дата доступа: 03.04.2023 г.

ДОДАТОК А

Повний код класу InfoBlockChangeComponent.

```
import { ChangeDetectionStrategy, ChangeDetectorRef, Component, ElementRef, OnDestroy, OnInit, ViewChild } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { Store } from '@ngrx/store';
import { first, map, mergeMap, tap } from 'rxjs/operators';
import { UrlRouteConst } from 'src/app/modules/core/consts/route/url-route.const';
import { InfoBlockSectionConst } from 'src/app/modules/core/consts/info-blocks-section.const';
import { ActionConst } from '../../../consts/action.const';
import { IInfoBlockChangeQueryParams } from '../../../interfaces/info-block/change-query-params/info-block-change-query-params.interface';
import { ISaveInfoBlock } from '../../../interfaces/info-block/save-info-block.interface';
import { IUpdateInfoBlock } from '../../../interfaces/info-block/update-info-block.interface';
import { InfoBlockAdminService } from '../../../services/info-block-admin.service';
import * as FullInfoBlockActions from '../../../store/full-info-block/full-info-block.actions';
import * as FullInfoBlockSelectors from '../../../store/full-info-block/full-info-block.selector';
import * as InfoBlockDraftActions from '../../../store/info-block-draft/info-block-draft.actions';
```

```
import { IInfoBlockDraft } from '../../../interfaces/info-block/info-block-draft.interface';
import { IOptionalInfoBlock } from '../../../interfaces/info-block/optional-info-block.interface';
import * as InfoBlockDraftSelectors from '../../../store/info-block-draft/info-block-draft.selector';
import { Observable, of } from 'rxjs';
import { IInfoBlock } from 'src/app/modules/core/interfaces/info-block/info-block.interface';
import { IOptionalAdditionalImg } from '../../../interfaces/optional-additional-img.interface';
import { IFileInfo } from 'src/app/modules/core/interfaces/file-info.interface';
import { ErrorService } from 'src/app/modules/core/services/error/error.service';
import { StringService } from 'src/app/modules/core/services/string/string.service';
import { isEqual, cloneDeep } from 'lodash';
import { AngularEditorConfig } from '@kolkov/angular-editor';

interface ISection {
  value: InfoBlockSectionConst;
  display: string;
}

const SECTIONS_CONST: ISection[] = [
  {
    value: InfoBlockSectionConst.GENERAL,
    display: 'Головна'
  },
```

```
  {
    value: InfoBlockSectionConst.ACTUAL_GRANTS,
    display: 'Актуально-Гранти'
  },
  {
    value: InfoBlockSectionConst.ACTUAL_CREDITS,
    display: 'Актуально-Кредити'
  },
  {
    value: InfoBlockSectionConst.ACTUAL_TEACHING,
    display: 'Актуально-Навчання'
  },
  {
    value: InfoBlockSectionConst.ACTUAL_ACTIVITIES,
    display: 'Актуально-Заходи'
  },
  {
    value: InfoBlockSectionConst.SERVICES,
    display: 'Послуги'
  }
];

const EDITOR_CONFIG: AngularEditorConfig = {
  editable: true,
  toolbarHiddenButtons: [['fontName', 'heading']]
};

@Component({
  selector: 'app-info-block-change',
```

```
  templateUrl: './info-block-change.component.html',
  styleUrls: ['./info-block-change.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class InfoBlockChangeComponent implements OnInit, OnDestroy {

  htmlText: string | undefined;
  errorText: string | undefined;
  headerImg: IFileInfo | undefined;
  additionalImgs: IOptionalAdditionalImg[] | undefined;
  additionalImgIndex: number | undefined;
  viewActionLiteral: string | undefined;
  chosenSection: InfoBlockSectionConst | undefined;
  uploadFiles: IFileInfo[] | undefined;

  readonly sections: ISection[];
  readonly editorConfig: AngularEditorConfig;

  private _startInfoBlock: IOptionalInfoBlock;
  private _queryParams: IInfoBlockChangeQueryParams<number>;

  @ViewChild('headerImgRef', { static: true })
  private readonly _headerImgRef: ElementRef<HTMLInputElement>;

  @ViewChild('additionalImgRef', { static: true })
  private readonly _additionalImgRef: ElementRef<HTMLInputElement>;

  @ViewChild('uploadFileRef', { static: true })
  private readonly _uploadFileRef: ElementRef<HTMLInputElement>;
```

```
@ViewChild('headerRef', { static: true })
private readonly _headerRef: ElementRef<HTMLTextAreaElement>;

constructor(
  private readonly _router: Router,
  private readonly _store: Store<IInfoBlock>,
  private readonly _errorService: ErrorService,
  private readonly _stringService: StringService,
  private readonly _activatedRoute: ActivatedRoute,
  private readonly _changeDetect: ChangeDetectorRef,
  private readonly _infoBlockAdminService: InfoBlockAdminService,
) {
  this.sections = SECTIONS_CONST;
  this.editorConfig = EDITOR_CONFIG;
}

ngOnInit(): void {
  this.uploadFiles = [];
  this.additionalImgs = [];
  this.additionalImgIndex = 0;
  this.chosenSection = InfoBlockSectionConst.GENERAL;
  this._queryParams = this._getQueryParamsFromactivatedRoute();
  this._initStore();
}

ngOnDestroy(): void {
  this._store.dispatch(FullInfoBlockActions.reset());
}
```

```
    onViewStateChange(): void {
      this.errorText = undefined;
      this._saveInfoBlockDraft();
      this._changeDetect.detectChanges();
    }

    onUploadFile(): void {
      this._uploadFileIfExist(
        this._uploadFileRef,
        (fileInfo: IFileInfo) => this.uploadFiles = this.uploadFiles.concat(fileInfo)
      );
    }

    onHeaderImgChange(): void {
      this._uploadFileIfExist(
        this._headerImgRef,
        (fileInfo: IFileInfo) => this.headerImg = fileInfo
      );
    }

    onUploadFileUrlClick(index: number): void {
      navigator.clipboard.writeText(this.uploadFiles[index].url);
    }

    // onHtmlTextChange(event: Event): void {
    //   this.onViewStateChange();
    //   if (event && event.target && (event?.target as HTMLInputElement)?.value
!= null) {
```

```
        //     const value: string = (event.target as HTMLInputElement).value;
        //     this._humanTextRef.nativeElement.innerHTML = value;
        //   }
        // }

        getAdditionalImgsForView(): string[] | undefined {
          return this.additionalImgs?.map((img: IOptionalAdditionalImg) => img?.file?.url);
        }

        addAdditionalImg(): void {
          const files: FileList = this._additionalImgRef.nativeElement.files;
          if (files && files.length > 0) {
            this._infoBlockAdminService.uploadFiles$(files)
              .toPromise()
              .then((filesInfo: IFileInfo[]) => {
                this.additionalImgs = cloneDeep(this.additionalImgs);
                this.additionalImgs.splice(
                  this.additionalImgIndex + 1, 0, ...filesInfo.map((file: IFileInfo) => ({ file: { ...file } }))
                );
                if (this.additionalImgIndex === 0) {
                  this.additionalImgIndex--;
                }
                this.additionalImgIndex += filesInfo.length;
                this.onViewStateChange();
              })
              .catch(error => this._handleUploadImgError(error));
          }
```

```
  }

  deleteAdditionalImg(): void {
    this.additionalImgs = cloneDeep(this.additionalImgs);
    this.additionalImgs.splice(this.additionalImgIndex, 1);
    if (this.additionalImgIndex > 0 && this.additionalImgIndex === this.additionalImgs.length) {
      this.additionalImgIndex--;
    }
    this.onViewStateChange();
  }

  onSave(): void {
    const infoBlock: IOptionalInfoBlock = this._getInfoBlock();
    this.errorText = undefined;
    if (
      infoBlock.section
      && infoBlock.header
      && infoBlock.htmlText
    ) {
      switch (this._queryParams.action) {
        case ActionConst.ADD: this._saveInfoBlock((infoBlock as ISaveInfoBlock)); break;
        case ActionConst.UPDATE: this._updateInfoBlock((infoBlock as ISaveInfoBlock), this._queryParams.id); break;
      }
    } else {
      this.errorText = 'Не всі обов\'язкові поля заповнені';
      this._changeDetect.detectChanges();
```

```
    }
  }

  navigateToInfoBlockOverview(): void {

this._router.navigate([UrlRouteConst.ADMIN_INFO_BLOCK_OVERVIEW]);
  }

  private _getQueryParamsFromactivatedRoute():
IInfoBlockChangeQueryParams<number> {
    const queryParamsString: IInfoBlockChangeQueryParams<string> =
      this._activatedRoute.snapshot.queryParams as
IInfoBlockChangeQueryParams<string>;
    if (queryParamsString.action === ActionConst.UPDATE) {
      const queryParams: IInfoBlockChangeQueryParams<number> =
        { ...queryParamsString, id: Number(queryParamsString.id) };
      return queryParams;
    } else {
      return queryParamsString;
    }
  }

  private _initStore(): void {
    this._store.dispatch(InfoBlockDraftActions.load({
      completed: () => {
        const draft$: Observable<IInfoBlockDraft> =
this._store.select(InfoBlockDraftSelectors.get);
        draft$
          .pipe(
```

```
            first(),
            mergeMap((draft: IInfoBlockDraft) => {
              if (draft && this._confirmToRestoreDataFromDraft()) {
                return this._restoreDataFromDraft$(draft);
              }
              this._store.dispatch(InfoBlockDraftActions.reset());
              if (this._queryParams.action === ActionConst.UPDATE) {
                return this._defineFullInfoBlock$(this._queryParams.id, true);
              }
              this._initViewActionLiteral();
              return of();
            })
          )
          .toPromise()
          .catch(error => {
            console.error(error);
            this.navigateToInfoBlockOverview();
          });
      }
    }));
  }

  private _defineFullInfoBlock$(id: number, isSetView: boolean): Observable<IInfoBlock> {
    this._store.dispatch(FullInfoBlockActions.loadByIdIfNeed({
      id, handleError: (error) => {
        console.error(error);
        this.navigateToInfoBlockOverview();
      }
```

```
      }));
      const infoBlock$: Observable<IInfoBlock> = this._store.select(FullInfoBlockSelectors.getById, id);
      return infoBlock$.pipe(
        first((infoBlock: IInfoBlock) => !!infoBlock),
        tap((infoBlock: IInfoBlock) => {
          this._startInfoBlock = infoBlock;
          if (isSetView) { this._setViews(infoBlock); }
        })
      );
    }

    private _restoreDataFromDraft$(draft: IInfoBlockDraft): Observable<void> {
      this._queryParams = draft.queryParams;
      this.uploadFiles = draft.uploadFiles || [];
      this._setViews(draft.currentInfoBlock);
      this._router.navigate([UrlRouteConst.ADMIN_INFO_BLOCK_CHANGE], { queryParams: this._queryParams });
      if (this._queryParams.action === ActionConst.UPDATE) {
        return this._defineFullInfoBlock$(this._queryParams.id, false).pipe(
          map((startInfoBlock: IInfoBlock) => {
            this._store.dispatch(InfoBlockDraftActions.set({ infoBlockDraft: { ...draft, startInfoBlock } }));
          })
        );
      } else { return of(); }
    }

    private _confirmToRestoreDataFromDraft(): boolean {
```

```
        return confirm('Ви маєте незбережені дані, бажаєте продовжити їх заповнення? У разі відмови данні будуть видалені!');
      }

      private _setViews(startInfoBlock: IOptionalInfoBlock): void {
        if (startInfoBlock.headerImg) { this.headerImg = startInfoBlock.headerImg; }
        if (startInfoBlock.section) { this.chosenSection = startInfoBlock.section; }
        if (startInfoBlock.header) { this._headerRef.nativeElement.value = startInfoBlock.header; }
        if (startInfoBlock.htmlText) {
          this.htmlText = startInfoBlock.htmlText;
          // this._htmlTextRef.nativeElement.value = startInfoBlock.htmlText;
          // this._humanTextRef.nativeElement.innerHTML = startInfoBlock.htmlText;
        }
        if (startInfoBlock.additionalImgs) {
          this.additionalImgs = startInfoBlock.additionalImgs;
        }
        this._initViewActionLiteral();
        this._changeDetect.detectChanges();
      }

      private _initViewActionLiteral(): void {
        this.viewActionLiteral = this._queryParams.action === ActionConst.ADD
          ? 'Додавання блоку'
          : `Редагування блоку № ${this._queryParams.id}`;
        this._changeDetect.detectChanges();
      }
```

```
    private async _saveInfoBlock(infoBlock: ISaveInfoBlock): Promise<void> {
      this._infoBlockAdminService.save$(infoBlock)
        .toPromise()
        .then(() => {
          this.navigateToInfoBlockOverview();
          this._store.dispatch(InfoBlockDraftActions.reset());
        })
        .catch(error => this._handleServerError(error));
    }

    private async _updateInfoBlock(infoBlock: ISaveInfoBlock, id: number): Promise<void> {
      const isChanges = this._deleteUnchangedFields(infoBlock);
      if (isChanges) {
        const updateInfoBlock: IUpdateInfoBlock = { ...infoBlock, id };
        this._infoBlockAdminService.update$(updateInfoBlock)
          .toPromise()
          .then(() => {
            this._updateStartInfoBlock(updateInfoBlock);
            this._store.dispatch(InfoBlockDraftActions.reset());
          })
          .catch(error => this._handleServerError(error));
      } else {
        this.errorText = 'Дані вже збережено!';
        this._store.dispatch(InfoBlockDraftActions.reset());
      }
    }
```

```
private _uploadFileIfExist(
  { nativeElement }: ElementRef<HTMLInputElement>,
  callback: (fileInfo: IFileInfo) => void
): void {
  const file: File | undefined = nativeElement?.files[0];
  if (file) {
    this._infoBlockAdminService.uploadFile$(file)
      .toPromise()
      .then((fileInfo: IFileInfo) => {
        callback(fileInfo);
        this.onViewStateChange();
      })
      .catch(error => this._handleUploadImgError(error));
  }
}

private _handleServerError<T>(error: T): void {
  console.error(error);
  this.errorText = 'Сталась невідома помилка!';
  this._changeDetect.detectChanges();
}

private _handleUploadImgError<T>(error: T): void {
  if (this._errorService.isBadRequest(error)) {
    console.error(error);
    this.errorText = 'Назва файлу некоректна або розмір занадто великий';
    this._changeDetect.detectChanges();
  } else {
    this._handleServerError(error);
```

```
    }
  }

  private _updateStartInfoBlock(infoBlock: IUpdateInfoBlock): void {
    this._startInfoBlock = { ...this._startInfoBlock, ...infoBlock };
  }

  private _deleteUnchangedFields(currentInfoBlock: ISaveInfoBlock): boolean
{
    let isChanges: boolean = false;
    for (const key in currentInfoBlock) {
      if (currentInfoBlock[key] == null || isEqual(currentInfoBlock[key],
this._startInfoBlock[key])) {
        delete currentInfoBlock[key];
      } else {
        isChanges = true;
      }
    }
    return isChanges;
  }

  private _getInfoBlock(): IOptionalInfoBlock {
    const id: number = this._queryParams.action === ActionConst.UPDATE ?
this._queryParams.id : undefined;
    const section: InfoBlockSectionConst = this.chosenSection;
    const header: string = this._headerRef.nativeElement.value || undefined;
    const innerHTML: string | null =
document.getElementsByClassName('angular-editor-textarea')[0]?.innerHTML;
```

```
        const htmlText: string | undefined = innerHTML ? this._stringService.deleteMultipleSpaces(innerHTML) : undefined;
        const headerImg: IFileInfo = this.headerImg;
        const additionalImgs: IOptionalAdditionalImg[] | undefined = this.additionalImgs.length > 0 ? this.additionalImgs : undefined;
        return { id, section, header, headerImg, htmlText, additionalImgs };
      }

      private async _saveInfoBlockDraft(): Promise<void> {
        const uploadFiles: IFileInfo[] | undefined = this.uploadFiles;
        const queryParams: IInfoBlockChangeQueryParams<number> = this._queryParams;
        const currentInfoBlock: IOptionalInfoBlock = this._getInfoBlock();
        const infoBlockDraft: IInfoBlockDraft = { startInfoBlock: this._startInfoBlock, currentInfoBlock, queryParams, uploadFiles };
        this._store.dispatch(InfoBlockDraftActions.set({ infoBlockDraft }));
      }}
```