НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

«На правах рукопису»
УДК 004.41

«До захисту допущено»
В.о. завідувача кафедри
_____ Едуард ЖАРІКОВ
«___»_____2021 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Архітектурне рішення для програмного забезпечення терміналу підтримки студентської діяльності»

Виконав (-ла):
студент (-ка) ІІ курсу, групи ІТ-з03мп
Кізюн Богдан Миколайович _____

Керівник:
доцент кафедри ІПІ, к.т.н., доц.,
Новінський Валерій Петрович _____

Рецензент:
доцент кафедри ІСТ, к.т.н., доц.,
Жданова Олена Григорівна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2021 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії
Рівень вищої освіти – другий (магістерський)
Спеціальність – 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних систем»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Едуард ЖАРІКОВ
«___»_____2021р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Кізюну Богдану Миколайовичу

1. Тема дисертації «Архітектурне рішення для програмного забезпечення терміналу підтримки студентської діяльності», науковий керівник дисертації Новінський Валерій Петрович, доцент кафедри ІПІ, к.т.н., затверджені наказом по університету від «25» жовтня 2021 р. № 3575-с

2. Термін подання студентом дисертації «6» грудня 2021 р.

3. Об'єкт дослідження – архітектурне рішення для програмного забезпечення терміналу підтримки студентської діяльності

4. Предмет дослідження – проектування та програмна реалізація архітектурного рішення системи інформаційної підтримки студентів. Для реалізації предмету дослідження був використаний підхід клієнт-серверної архітектури який використовує REST API. Що до реалізації клієнтської частини буде реалізований веб-додаток з декількома сторінками на основі JavaScript, каскадних таблиць стилів та мови розмітки HTML.

5. Перелік завдань, які потрібно розробити – Ознайомлення з різними системами інформаційної підтримки студентів, аналіз варіантів проектування на основі проведеного дослідження наявних інформаційних технологій у сфері надання інформаційних послуг студентам, проектування архітектурного рішення для зберігання та обробки даних, проектування клієнт-серверної

архітектури за допомогою різноманітних діаграм, створення веб-інтерфейсу користувача для надання інформаційних послуг студентам, розроблення серверної частини проекту на мові C# та веб-інтерфейсу за допомогою мови програмування JavaScript, мови розмітки HTML та каскадних таблиць стилів.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу – 10 слайдів.

7. Орієнтовний перелік публікацій – дві публікації.

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9.Дата видачі завдання «30» вересня 2020 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання	Примітка
1	Затвердження теми роботи	15.09.2021 -20.09.2021	
2	Вивчення та аналіз завдання	20.09.2021-30.09.2021	
3	Робота над магістерською дисертацією. Виконання та аналіз результатів експериментів для оцінки ефективності розробки	01.10.2021-05.11.2021	
4	Робота над розділом стартапу.	05.11.2021-18.11.2021	
5	Оформлення пояснювальної записки	18.11.2021-21.11.2021	
6	Подання дисертації на попередній захист	22.11.2021	
7	Подання дисертації на захист	06.12.2021	
8			
9			
10			

Студент Богдан КІЗЮН

Науковий керівник Валерій НОВІНСЬКИЙ

РЕФЕРАТ

Магістерська дисертація: 100 с., 15 рисунків, 1 таблиця, 20 джерел, 8 додатків.

Актуальність теми: в сучасному світі майже всі інформаційні послуги надаються в електронному вигляді з певною долею автоматизації, тому зникає необхідність в паперових носіях та відвідуванні певних інстанцій особисто. Саме тому, актуальним буде використання зручного веб-інтерфейсу для надання студенту інформаційних послуг без необхідності взаємодії з адміністрацією.

Мета дослідження: створення архітектурного рішення для терміналу інформаційної підтримки студентів для оптимізації навчального процесу та прискорення отримання інформації студентами. Перед проектом стала задача якомога швидше та оптимізовано передавати різноманітну інформацію, таку як: розклад занять, дані про студента, фінансову заборгованість, різноманітні повідомлення, тощо.

Завдання дослідження:

1) ознайомлення з різними системами інформаційної підтримки студентів;

2) аналіз варіантів проектування на основі проведеного дослідження наявних інформаційних технологій у сфері надання інформаційних послуг студентам;

3) проектування архітектурного рішення для зберігання та обробки даних;

4) проектування клієнт-серверної архітектури за допомогою різноманітних діаграм;

5) створення веб-інтерфейсу користувача для надання інформаційних послуг студентам;

6) розроблення серверної частини проекту на мові C# та веб-інтерфейсу за допомогою мови програмування JavaScript, мови розмітки HTML та каскадних таблиць стилів.

Об’єкт дослідження: архітектурне рішення для програмного забезпечення терміналу підтримки студентської діяльності.

Предмет дослідження: проектування та програмна реалізація архітектурного рішення системи інформаційної підтримки студентів.

Методи дослідження: під час виконання магістерської роботи були проведені наступні дослідження:

1) пошук альтернатив даному архітектурному рішенні;

2) аналіз та класифікація наявних ресурсів навчального розкладу;

3) обговорення з потенційними користувачами системи, що до функціональності необхідній в системі;

4) створення прототипу з мінімальною функціональністю;

5) емуляція високої завантаженості для тестування відмовостійкості.

Наукова новизна: наукова новизна отриманих результатів полягає в тому, що був розроблений алгоритм зберігання та доступу до даних для архітектурного рішення, який забезпечує відокремлення інформації про студентів однієї групи, безпосередньо, від їх групи, але дозволяє додавати спільні дані лише один раз для груп, а окрему інформацію — для кожного студента. Такий підхід дозволив оптимізувати процес заповнення та зберігання даних. Це було досягнуто шляхом переосмислення зв'язків між сутностями та може бути використано при розробці та покращенні існуючих систем підтримки студентської діяльності.

Публікації:

1) збірник наукових матеріалів XXIV Міжнародної інтернет-конференції «SCIENCE AND TECHNOLOGY» Польща, м. Люблін. – 2021. 11–12 жовтня 2021р. – 122 с;

2) перша Всеукраїнська науково-практична конференція молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології» (SoftTech- 2021). Секція кафедри інформатики та програмної інженерії. Матеріали конференції. – Київ. – 2021. 22–26 листопада 2021р. – 148 с.

ABSTRACT

Master's dissertation: 100 pages, 15 figures, 1 table, 20 sources, 8 appendices.

Relevance of the topic: in today's world almost all information services are provided in electronic form with a certain degree of automation, so there is no need for paper sources and visit certain instances in person. Therefore, it will be important to use a user-friendly web interface to provide students with information services without the need to interact with the administration.

The purpose of the study: to create an architectural solution for the terminal of information support for students to optimize the learning process and accelerate the receipt of information by students. The project was faced with the task of transmitting various information as quickly and optimally as possible, such as: class schedule, student data, financial debt, various messages, etc.

Objectives of the study:

1) Acquaintance with various systems of information support of students;
2) Analysis of design options based on a study of existing information technology in the field of information services to students;
3) Designing an architectural solution for data storage and processing;
4) Designing a client-server architecture using a variety of diagrams;
5) Creating a web user interface to provide information services to students;
6) Development of the server part of the project in C # language and web interface using JavaScript programming language, HTML markup language and cascading style sheets.

Object of research: architectural solution for the software of the student support terminal.

Subject of research: design and software implementation of the architectural solution of the information support system for students.

Research methods: during the master's thesis the following research was conducted:

1) search for alternatives to this architectural solution;
2) analysis and classification of available curriculum resources;
3) discuss with potential users of the system what functionality is needed in the system;
4) creating a prototype with minimal functionality;
5) high load emulation for fault tolerance testing.

Scientific novelty: the scientific novelty of the results is that an algorithm for storing and accessing data for architectural solutions was developed, which separates information about students of one group, directly from their group, but allows you to add common data only once for groups, and separate information for each student. This approach allowed to optimize the process of filling and storing data. This has been achieved by rethinking the links between entities and can be used to develop and improve existing student support systems.

Publications:

1) collection of scientific materials of the XXIV International Internet Conference "SCIENCE AND TECHNOLOGY" Poland, Lublin. - 2021. October 11–12, 2021. - 122 p;
2) the First All-Ukrainian Scientific and Practical Conference of Young Scientists and Students "Software Engineering and Advanced Information Technologies" (SoftTech-2021). Section of the Department of Informatics and Software Engineering. Conference materials. - Kyiv. - 2021. November 22-26, 2021. - 148 p.

ЗМІСТ

ВСТУП

В наші дні процес впровадження інформаційних технологій істотно необхідний в усіх сферах життя людей. Тим більше беручи до уваги останні події в світі зв’язані с пандемією коронавірусної хвороби 2019—2021(COVID-19) перед людством постала нагальна необхідність зменшити час персонального контакту та збільшити соціальну дистанцію один з одним, тому інтеграція електронного документообігу виконувалася дуже швидкими темпами. Для прикладу можна взяти сферу державних послуг: раніше подання показників лічильників, оплата податків, банківські послуги, тощо можна було виконувати в електронному вигляді в якості альтернативи, наразі ж використання сайтів та додатків для держпослуг майже необхідна умова, так як держава старається мінімізувати відвідування офісів служб та бюро для громадян в цілях зменшення кількості хворих.

В умовах адаптивного карантину були також прийняті заходи щодо освітнього процесу в школах та університетах, а саме, в першу чергу, дистанційне навчання та поширення використання електронних кабінетів студентів та школярів. Так як при навчанні вдома немає можливості відвідувати університет або школу то і необхідність в паперових носіях(розкладах занять, адміністративних документах, тощо) відпадає майже повністю за рахунок використання електронних баз. Це забезпечує впровадження електронного документообігу за рахунок виконання ретроконверсії(переведення даних з графічних образів, отриманих після сканування, в текстовий формат або формат баз даних) та імідж-каталогів(упорядкований масив оцифрованих каталожних карток одиниць зберігання).

Тому було прийнято рішення про реалізацію архітектурного рішення для програмного забезпечення терміналу підтримки студентської діяльності під час якого було створено веб-інтерфейс користувача, який допомагає користувачам-студентам отримати інформаційну підтримку в електронному вигляді.

Наявність електронного розкладу для студентів дозволяє звільнитись від необхідності наявності паперових носіїв інформації та швидко і без зайвих зусиль, при необхідності, вносити зміни в розклад. Це зменшує час затримки при взаємодії студентів та адміністрації з питань актуальності розкладу.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАДАЧІ

1.1 Проблеми розвитку інформаційних технологій в сфері надання інформаційних послуг студентам.

Беручи до уваги той факт, що за останній час процес навчання з дистанційного переходить то відвідування пар або уроків наживо, то і необхідність наявності розкладів занять та документів в паперовому вигляді, наприклад, на дошках оголошень, для інформування максимальної кількості осіб, є доволі великою. Але повернення до старого формату при існуванні оцифрованих документів є досить непрактичною ідеєю, тому що використання паперових носіїв приводить до:

– збільшення витрат на канцелярію(наприклад, папір та чорнила для друку);

– збільшення негативного впливу на екологію(виробництво паперу впливає на вирубку лісів, а при створенні чорнил залучається хімічна промисловість, що шкодить водним ресурсам);

– збільшення часу необхідного для оновлення доступної інформацій.

Тому ідея створення електронної альтернативи дошкам оголошень є досить розумною з точки зору економічної, екологічної та інформаційної вигоди. Це забезпечить швидко та ефективно покращити навчальний процес з мінімумом затрат, залученням малої кількості виконавців та без необхідності перенавчання існуючого персоналу використання налагодженого робочого процесу.

Так як варіацій використання інформаційних технологій є досить багато, тому і варіантів реалізації даної концепції є декілька, а саме: публікація інформації через мережу Інтернет для доступу з будь-якого місця,

або власні внутрішні локальні мережі з доступом з спеціально відведених робочих станцій.

1.2 Аналіз існуючих інформаційних технологій в сфері надання інформаційних послуг студентам

1.2.1 Аналіз онлайн розкладу Національного університету "Одеська юридична академія"

Офіційний сайт НУ "ОУЮ" розроблено на основі мови програмування JavaScript з використанням бібліотеки jQuery, розмітка сайту виконана за допомогою мови розмітки HTML та каскадної таблиці стилів CSS. Даний сайт призначений для надання його користувачам інформації про новини університету, розклад занять, тощо. Для переходу до розкладу необхідно обрати пункт в навігаційній панелі зверху.

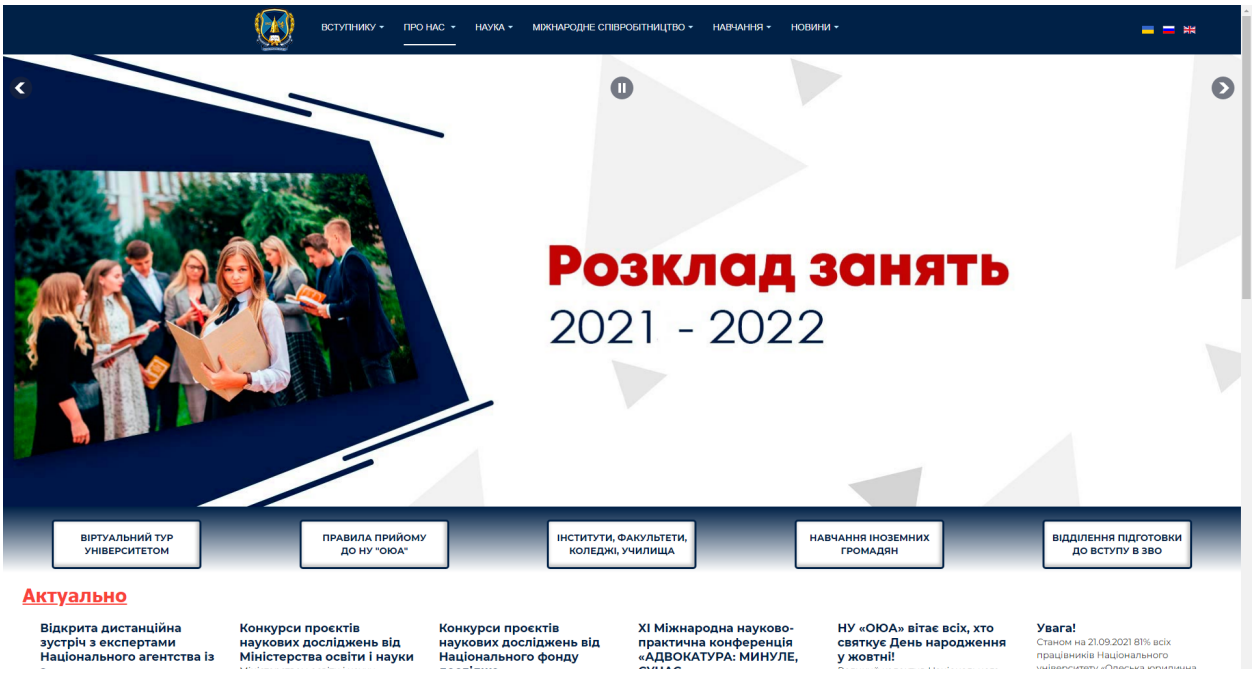


Рисунок 1.1 – “Головна” сторінка на офіційному сайт НУ "ОУЮ"

Процес доступу до актуального розкладу занять включає в себе декілька послідовних етапів:

1) вибір факультету із списку доступних;

2) вибір курсу та спеціальності;

3) доступ до відповідного документу розкладу в форматі PDF.

Основні функціональні можливості, переваги та недоліки онлайн розкладку на офіційному сайт НУ "ОУЮ":

– доступ до розкладу будь-яким користувачам використовуючи мережу Інтернет;

– документ у форматі PDF, що підтримується будь якими платформами(персональні комп’ютери, телефони, тощо);

– можливість завантажити файл документу на будь-який носій з можливістю офлайн доступу до нього;

– можливість доступу до документу з інших джерел(сторонні сайти та мобільні додатки), так як файли завантажуються за допомогою API-запитів веб-клієнта(XMLHttpRequest) за допомогою яких перегляд файлів можна реалізувати на будь-якій платформі;

– недоліком даного формату є необхідність вручну виконувати сканування роздрукованого розкладу та проводити оновлення шляхом заміни файлів;

– недоліком є відсутність можливості пошуку розкладу для конкретного студента.

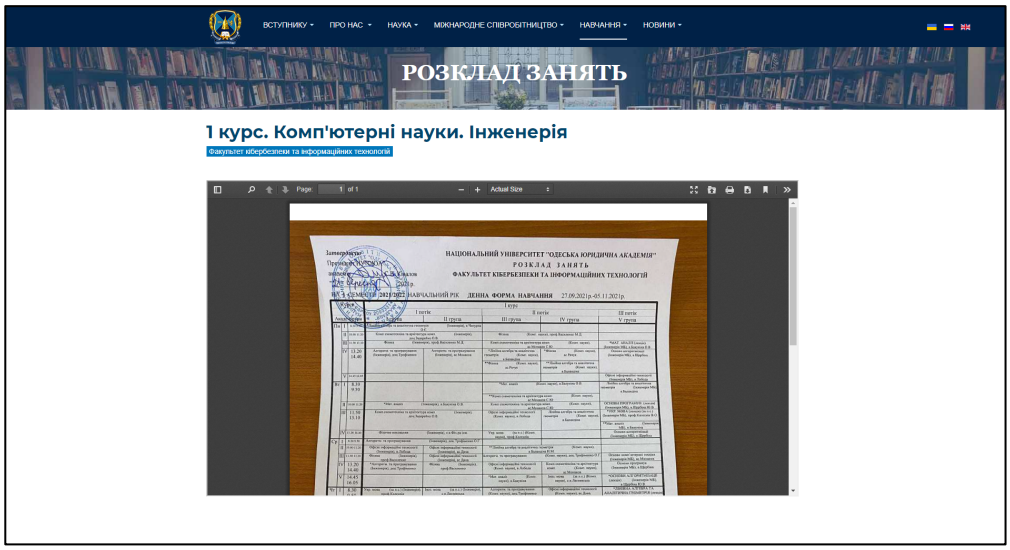


Рисунок 1.2 – Приклад онлайн розкладу на офіційному сайт НУ "ОУЮ"

1.2.2 Аналіз онлайн розкладу факультету інформаційних технологій та управління Київського університету імені Бориса Грінченка.

Даний сайт призначений для доступу до розкладу занять та новин студентам та вступникам факультету інформаційних технологій та управління в Київському університеті імені Бориса Грінченка. Сайт, як багато інших, розроблено на основі мови програмування JavaScript з використанням бібліотеки jQuery, так як це не потребує досить багато специфічних вмінь та часу, розмітка сайту виконана за допомогою мови розмітки HTML та каскадної таблиці стилів CSS. Загальний дизайн розкладу занять виконаний в досить зручному форматі.

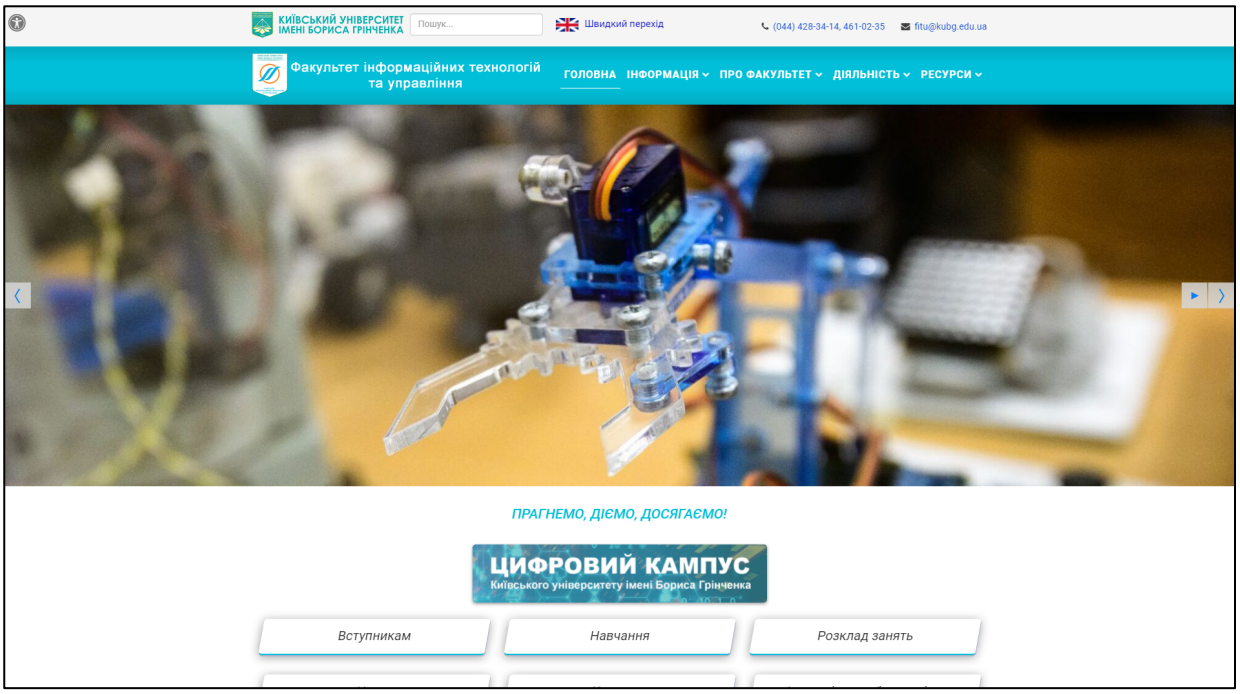


Рисунок 1.3 – “Головна” сторінка на сайті факультету інформаційних технологій та управління Київського університету імені Бориса Грінченка

Процес доступу до актуального розкладу занять включає в себе декілька послідовних етапів:

1) вибір форми навчання(денна/заочна) або доступу до дисциплін вільного вибору;

2) вибір спеціальності;

3) вибір курсу з випадаючого списку на місці обраної спеціальності;

4) доступ до відповідного документу розкладу в форматі PDF.

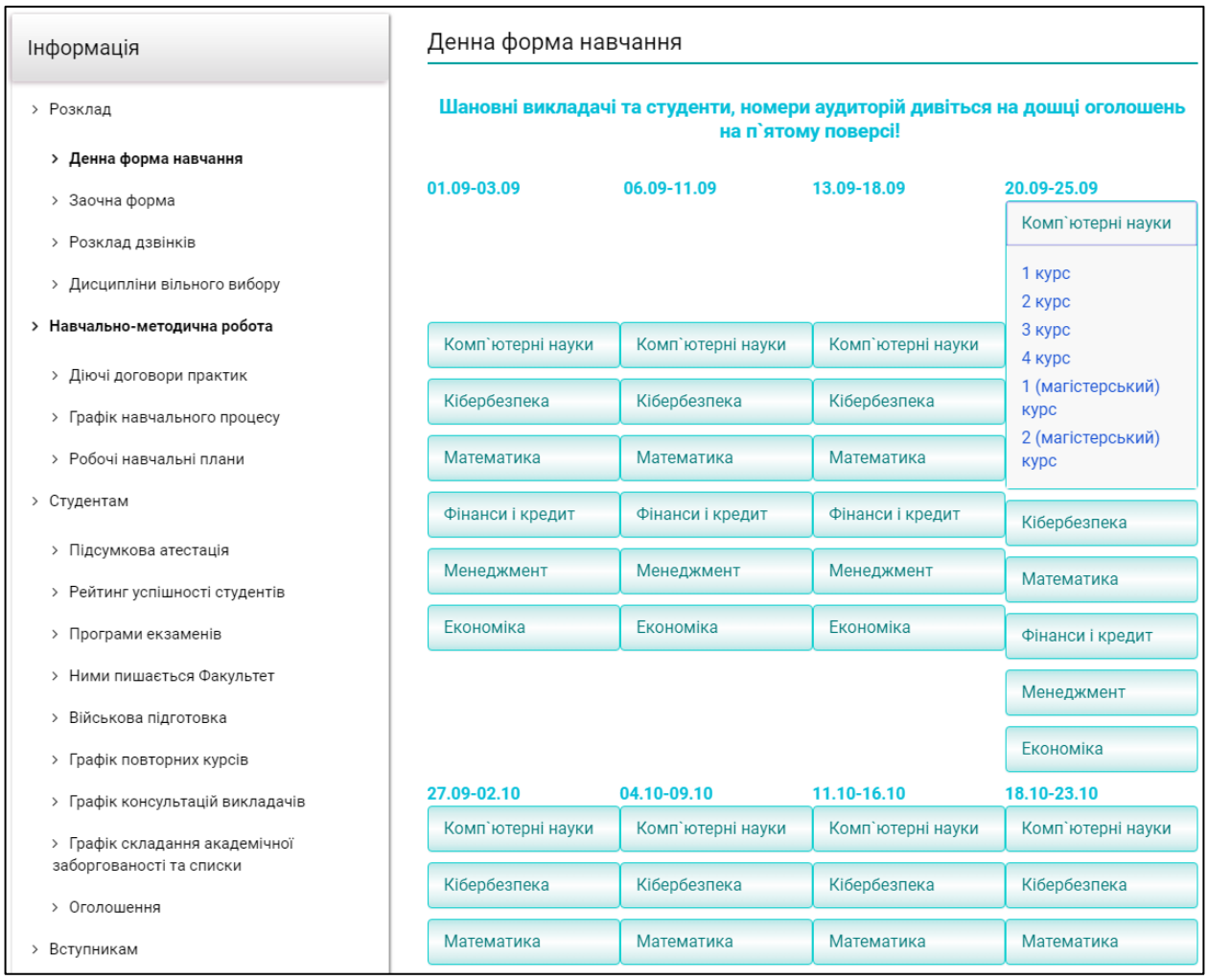


Рисунок 1.4 – Приклад вибору дати, спеціальності та курсу для доступу до розкладу на сайті

Основні функціональні можливості, переваги та недоліки онлайн розкладу факультету інформаційних технологій та управління Київського університету імені Бориса Грінченка:

– зручний інтерфейс з можливістю вибору відповідного навчального тижня;

– доступ до розкладу будь-яким користувачам використовуючи мережу Інтернет;

– документ у форматі PDF, що підтримується будь якими платформами(персональні комп’ютери, телефони, тощо);

– можливість завантажити файл документу на будь-який носій з можливістю офлайн доступу до нього;

– можливість доступу до документу з інших джерел(сторонні сайти та мобільні додатки), так як доступ до файлів виконується по посиланням за допомогою яких перегляд файлів можна реалізувати на будь-якій платформі за наявності відповідної документації;

– недоліком даного формату є необхідність вручну виконувати сканування роздрукованого розкладу та проводити оновлення шляхом заміни файлів;

– недоліком є відсутність можливості пошуку розкладу для конкретного студента.

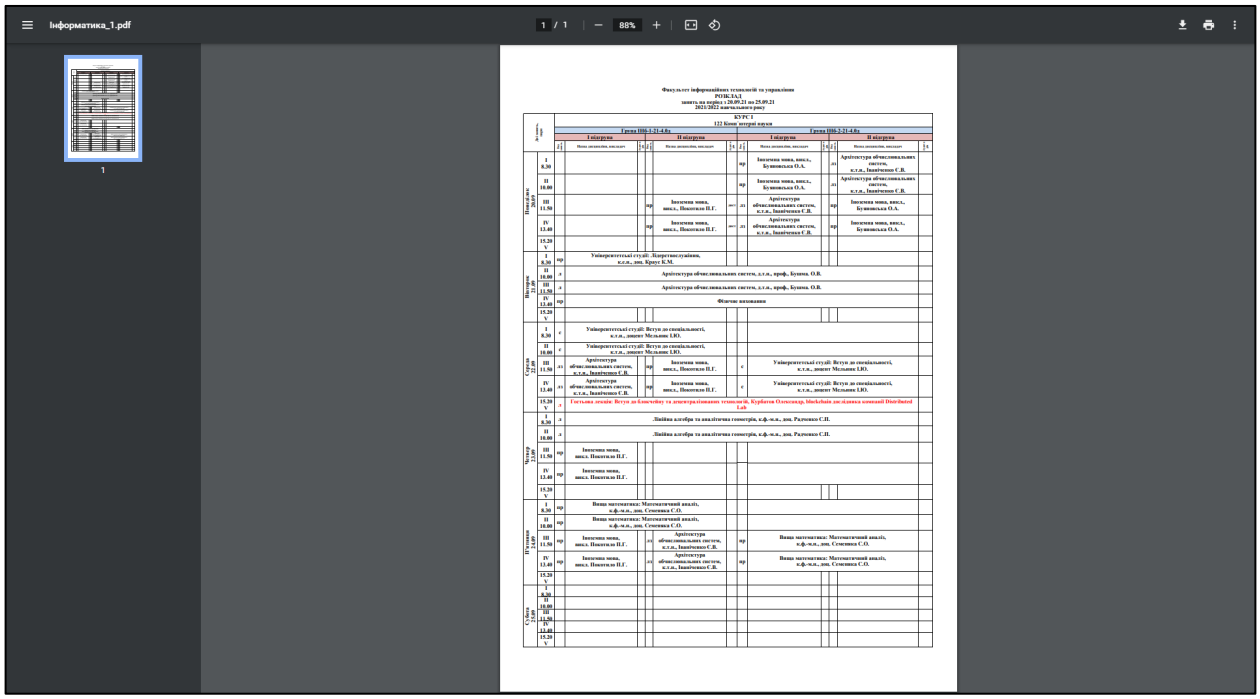


Рисунок 1.5 – Приклад документу онлайн розкладу 1 курсу факультету інформаційних технологій та управління Київського університету імені Бориса Грінченка

1.2.3 Аналіз онлайн розкладу Харківського національного університету радіоелектроніки

Даний розділ сайту призначений для доступу до розкладу занять студентам Харківського національного університету радіоелектроніки. Сайт розроблено на основі мови програмування JavaScript, розмітка сайту виконана за допомогою мови розмітки HTML та каскадної таблиці стилів CSS. Розмітка виконана в табличному стилі, що забезпечує універсальність та простоту додавання нового наповнення. Дизайн досить стриманий, що в свою чергу дозволяє витрачати менше ресурсів, нехтуючи візуальними ефектами та картинками, так як головною метою даного сайту є лише швидке отримання розкладу.

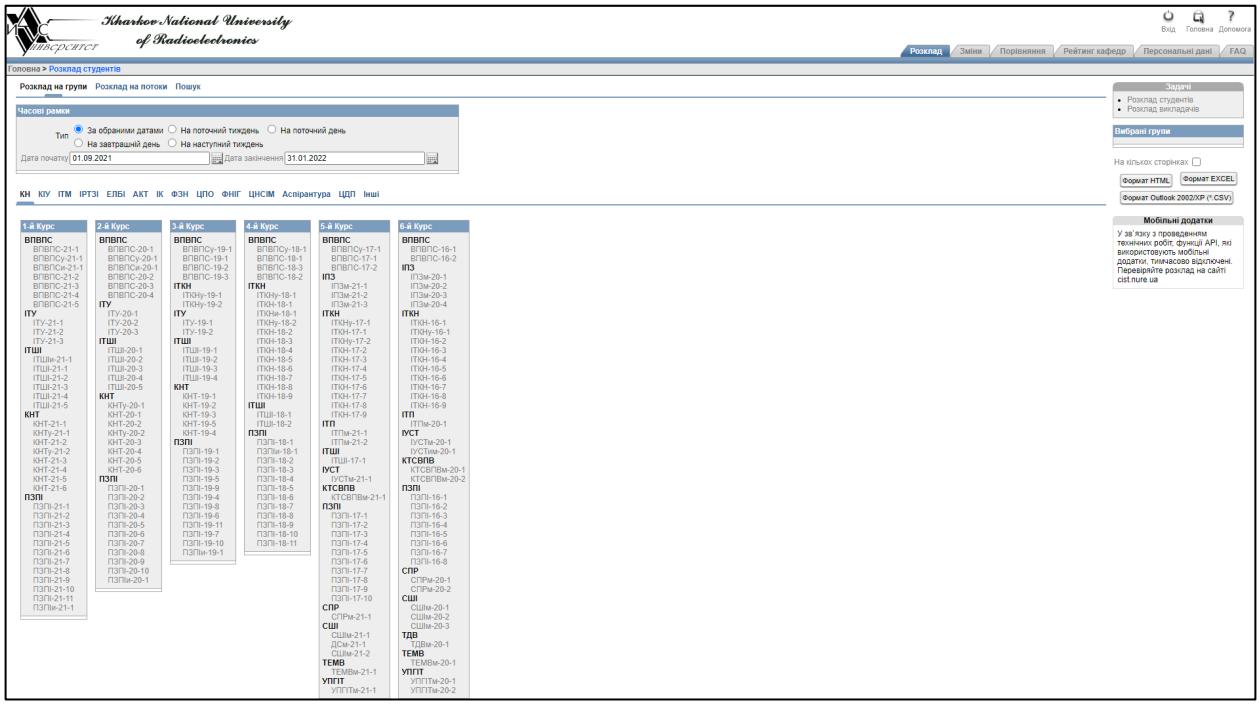


Рисунок 1.6 – “Головна” сторінка онлайн розкладу Харківського національного університету радіоелектроніки

Основні функціональні можливості, переваги та недоліки онлайн розкладу Харківського національного університету радіоелектроніки:

– доступ через мережу Інтернет забезпечує гнучку доступність ресурсу;

– вибір факультету, спеціальності та групи в одному вікні(при варіанті пошуку по групам) без необхідності переходу між сторінками, що збільшить працездатність при повільному підключенні до мережі Інтернет;

– гнучкий вибір діапазону дат;

– пошук розкладу на весь потік;

– пошук через заповнення відповідного рядку без необхідності вибору факультетів, спеціальностей та груп зі списку;

– пошук по кільком групам за один запит;

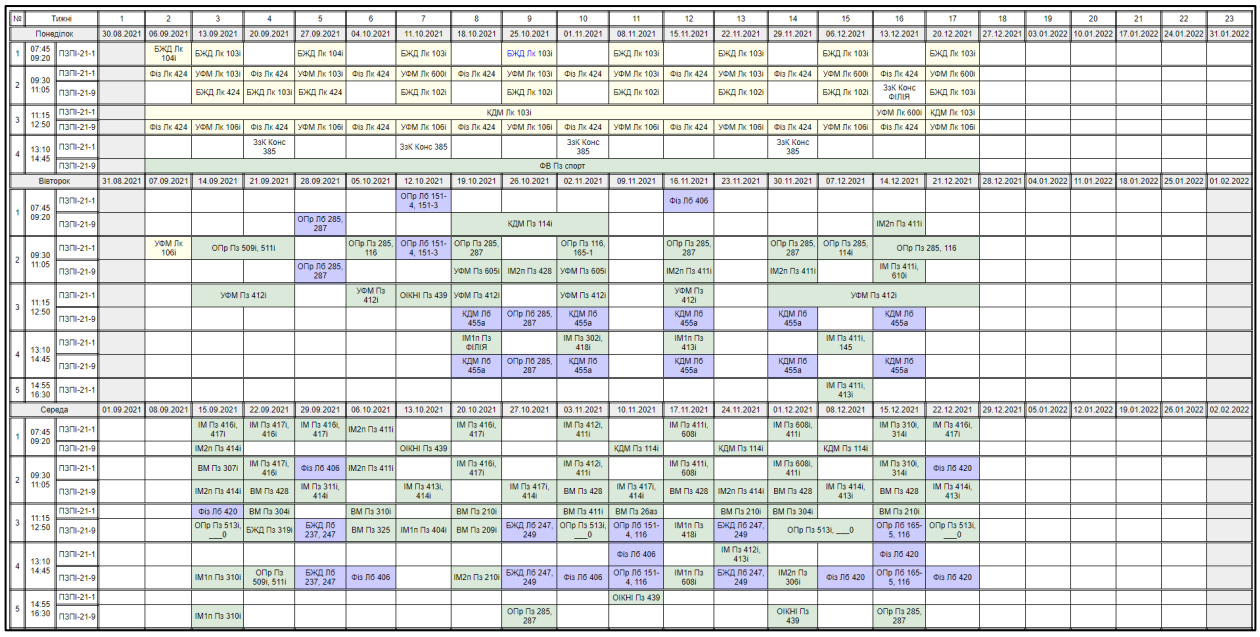
– отримання розкладу в декількох форматах, а саме: HTML(формування таблиці в новому вікні без завантаження окремих документів), Excel(завантаження файлу з можливістю відкрити його в Microsoft Excel), CSV(завантаження файлу в форматі текстової таблиці, що дозволяє його обробку в великій кількості інструментів);

– присутність API для доступу до системи з різних мобільних додатків та можливістю реалізації власних інтеграцій;

– завантаження файлів виконується шляхом запитів, отже є можливість реалізації доступу до файлів розкладу без наявності API;

– основними недоліками є завантаження картинок елементів в форматі GIF, що є неоптимальним форматом для статичних картинок, та використання в якості посиланні для переходу в інші вікна псевдо-протоколу "javascript:", що є досить застарілим підходом та може не працювати, якщо у користувача в браузері відключене використання JavaScript;

– недоліком є відсутність можливості пошуку розкладу для конкретного студента.



№	Тижні		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	Понеділок		30.08.2021	06.09.2021	13.09.2021	20.09.2021	27.09.2021	04.10.2021	11.10.2021	18.10.2021	25.10.2021	01.11.2021	08.11.2021	15.11.2021	22.11.2021	29.11.2021	06.12.2021	13.12.2021	20.12.2021	27.12.2021	03.01.2022	10.01.2022	17.01.2022	24.01.2022	31.01.2022
1	07:45 09:20	ПЗПІ-21-1		БЖД Лк 104і	БЖД Лк 103і		БЖД Лк 104і		БЖД Лк 103і		БЖД Лк 103і		БЖД Лк 103і		БЖД Лк 103і		БЖД Лк 103і		БЖД Лк 103і						
2	09:30 11:05	ПЗПІ-21-1		Фіз Лк 424	УФМ Лк 103і	Фіз Лк 424	УФМ Лк 103і	Фіз Лк 424	УФМ Лк 600і	Фіз Лк 424	УФМ Лк 103і	Фіз Лк 424	УФМ Лк 103і	Фіз Лк 424	УФМ Лк 103і	Фіз Лк 424	УФМ Лк 600і	Фіз Лк 424	УФМ Лк 600і						
		ПЗПІ-21-9			БЖД Лк 424	БЖД Лк 103і	БЖД Лк 424		БЖД Лк 102і		БЖД Лк 102і		БЖД Лк 102і		БЖД Лк 102і		БЖД Лк 102і	ЗзК Конс фізія	БЖД Лк 103і						
3	11:15 12:50	ПЗПІ-21-1		КДМ Лк 103і														УФМ Лк 600і	КДМ Лк 103і						
		ПЗПІ-21-9		Фіз Лк 424	УФМ Лк 106і	Фіз Лк 424	УФМ Лк 106і	Фіз Лк 424	УФМ Лк 106і	Фіз Лк 424	УФМ Лк 106і	Фіз Лк 424	УФМ Лк 106і	Фіз Лк 424	УФМ Лк 106і	Фіз Лк 424	УФМ Лк 106і	Фіз Лк 424	УФМ Лк 106і						
4	13:10 14:45	ПЗПІ-21-1				ЗзК Конс 385			ЗзК Конс 385			ЗзК Конс 385				ЗзК Конс 385									
		ПЗПІ-21-9		ФВ Пз спорт																					
	Вівторок		31.08.2021	07.09.2021	14.09.2021	21.09.2021	28.09.2021	05.10.2021	12.10.2021	19.10.2021	26.10.2021	02.11.2021	09.11.2021	16.11.2021	23.11.2021	30.11.2021	07.12.2021	14.12.2021	21.12.2021	28.12.2021	04.01.2022	11.01.2022	18.01.2022	25.01.2022	01.02.2022
1	07:45 09:20	ПЗПІ-21-1							ОПр Лб 151-4, 151-3					Фіз Лб 406											
		ПЗПІ-21-9					ОПр Лб 285, 287			КДМ Пз 114і								IM2n Пз 411і							
2	09:30 11:05	ПЗПІ-21-1		УФМ Лк 106і	ОПр Пз 509і, 511і			ОПр Пз 285, 116	ОПр Лб 151-4, 151-3	ОПр Пз 285, 287		ОПр Пз 116, 165-1		ОПр Пз 285, 287		ОПр Пз 285, 287	ОПр Пз 285, 114і	ОПр Пз 285, 116							
		ПЗПІ-21-9					ОПр Лб 285, 287			УФМ Пз 605і	IM2n Пз 428	УФМ Пз 605і		IM2n Пз 411і		IM2n Пз 411і		IM Пз 411і, 610і							
3	11:15 12:50	ПЗПІ-21-1			УФМ Пз 412і			УФМ Пз 412і	ОКНІ Пз 439	УФМ Пз 412і		УФМ Пз 412і		УФМ Пз 412і		УФМ Пз 412і									
		ПЗПІ-21-9								КДМ Лб 455а	ОПр Лб 285, 287	КДМ Лб 455а		КДМ Лб 455а		КДМ Лб 455а		КДМ Лб 455а							
4	13:10 14:45	ПЗПІ-21-1								IM1n Пз ФІЛЯ		IM Пз 302і, 418і		IM1n Пз 413і			IM Пз 411і, 145								
		ПЗПІ-21-9								КДМ Лб 455а	ОПр Лб 285, 287	КДМ Лб 455а		КДМ Лб 455а		КДМ Лб 455а		КДМ Лб 455а							
5	14:55 16:30	ПЗПІ-21-1															IM Пз 411і, 413і								
	Середа		01.09.2021	08.09.2021	15.09.2021	22.09.2021	29.09.2021	06.10.2021	13.10.2021	20.10.2021	27.10.2021	03.11.2021	10.11.2021	17.11.2021	24.11.2021	01.12.2021	08.12.2021	15.12.2021	22.12.2021	29.12.2021	05.01.2022	12.01.2022	19.01.2022	26.01.2022	02.02.2022
1	07:45 09:20	ПЗПІ-21-1			IM Пз 416і, 417і	IM Пз 417і, 416і	IM Пз 416і, 417і	IM2n Пз 411і		IM Пз 416і, 417і		IM Пз 412і, 411і		IM Пз 411і, 608і		IM Пз 608і, 411і		IM Пз 310і, 314і	IM Пз 416і, 417і						
		ПЗПІ-21-9			IM2n Пз 414і				ОКНІ Пз 439				КДМ Пз 114і		КДМ Пз 114і		КДМ Пз 114і								
2	09:30 11:05	ПЗПІ-21-1			ВМ Пз 307і	IM Пз 417і, 416і	Фіз Лб 406	IM2n Пз 411і		IM Пз 416і, 417і		IM Пз 412і, 411і		IM Пз 411і, 608і		IM Пз 608і, 411і		IM Пз 310і, 314і	Фіз Лб 420						
		ПЗПІ-21-9			IM2n Пз 414і	ВМ Пз 428	IM Пз 311і, 414і		IM Пз 413і, 414і		IM Пз 417і, 414і	ВМ Пз 428	IM Пз 417і, 414і	ВМ Пз 428	IM2n Пз 414і	ВМ Пз 428	IM Пз 414і, 413і	ВМ Пз 428	IM Пз 414і, 413і						
3	11:15 12:50	ПЗПІ-21-1			Фіз Лб 420	ВМ Пз 304і		ВМ Пз 310і		ВМ Пз 210і		ВМ Пз 411і	ВМ Пз 26аз		ВМ Пз 210і	ВМ Пз 304і		ВМ Пз 210і							
		ПЗПІ-21-9			ОПр Пз 513і, ___0	БЖД Пз 319і	БЖД Лб 237, 247	ВМ Пз 325	IM1n Пз 404і	ВМ Пз 209і	БЖД Лб 247, 249	ОПр Пз 513і, ___0	ОПр Лб 151-4, 116	IM1n Пз 418і	БЖД Лб 247, 249	ОПр Пз 513і, ___0		ОПр Лб 165-5, 116	ОПр Пз 513і, ___0						
4	13:10 14:45	ПЗПІ-21-1											Фіз Лб 406		IM Пз 412і, 413і			Фіз Лб 420							
		ПЗПІ-21-9			IM1n Пз 310і	ОПр Пз 509і, 511і	БЖД Лб 237, 247	Фіз Лб 406		IM2n Пз 210і	БЖД Лб 247, 249	Фіз Лб 406	ОПр Лб 151-4, 116	IM1n Пз 608і	БЖД Лб 247, 249	IM2n Пз 306і	Фіз Лб 420	ОПр Лб 165-5, 116	Фіз Лб 420						
5	14:55 16:30	ПЗПІ-21-1											ОКНІ Пз 439												
		ПЗПІ-21-9			IM1n Пз 310і						ОПр Пз 285, 287					ОКНІ Пз 439		ОПр Пз 285, 287							

Рисунок 1.7 – Приклад розкладу занять двох груп в Харківському національному університеті радіоелектроніки

1.3 Висновки по розділу

В даному розділі на прикладі аналізу окремих веб-сайтів для надання розкладу, були розглянуті основні напрямки та проблеми розвитку інформаційних технологій в сфері надання інформаційних послуг студентам.

На прикладі сайтів Національного університету "Одеська юридична академія", факультету інформаційних технологій та управління Київського університету імені Бориса Грінченка та Харківського національного університету радіоелектроніки проведений аналіз існуючих інформаційних технологій в сфері надання інформаційних послуг студентам, проведені роботи з дослідження основних функціональних можливостей зазначених автоматизованих систем, наведені їх переваги та недоліки, описаний метод реалізації окремих задач, що розглядаються в роботі.

РОЗДІЛ 2
ФОРМУВАННЯ КОНЦЕПЦІЇ АРХІТЕКТУРНОГО РІШЕННЯ СИСТЕМИ ІНФОРМАЦІЙНОЇ ПІДТРИМКИ СТУДЕНТІВ

2.1 Оцінка основних переваг та недоліків проаналізованих доступних альтернативних ресурсів в сфері надання інформаційних послуг студентам

Під час проведення аналізу аналогічних рішень для надання інформаційних послуг студентам було виділено декілька основних переваг та недоліків роботи розглянутих систем, що впливають на зручність користування та точність інформації для кінцевих споживачів.

Основними перевагами розглянутих аналогів є:

– розміщення розкладу в мережі Інтернет на сайті відповідного університету, що забезпечує легкий та постійний доступ до нього;

– отримання розкладу на сайті за допомогою запитів на сервер, що, в свою чергу, дає можливість використовувати це для зовнішніх ресурсів;

– можливість завантажити файл розкладу на носій для офлайн доступу.

Основними недоліками розглянутих аналогів є:

– досить велика кількість дій необхідних для отримання результату;

– досить перевантажений дизайн;

– файл розкладу у форматі PDF, що ускладнює процес копіювання окремих частин;

– орієнтування на групи студентів, без можливості зручно отримати список вибіркових дисциплін конкретного студента;

– необхідність ручного сканування паперових носіїв для завантаження їх на сайт кваліфікованими працівниками адміністрації.

2.2 Формування покращень на основі переваг та недоліків проаналізованих доступних альтернативних ресурсів в сфері надання інформаційних послуг студентам

На основі всіх приведених переваг та недоліків описаних в розділі 2.1 був сформований список покращень та основного функціоналу для реалізації в архітектурному рішенні, що розглядається в даній роботі:

- розміщення ресурсу в мережі Інтернет для доступу різних клієнтів;
- можливість розширення для потенційного додання нового функціоналу в майбутньому;
- уніфікований формат відповіді для можливості використання інформації в інших проектах;
- швидкий та зручний метод додавання та оновлення інформації;
- мінімальна кількість дій необхідних для досягнення бажаного результату;
- інтуїтивно зрозумілий та лаконічний дизайн для підвищення зручності користування;
- орієнтування на роботу з конкретним студентом, а не з групою, для спрощення доступу до вибіркових дисциплін;
- наявність додаткового функціоналу новин, що буде працювати схожим чином до списку занять;
- зменшення затрат на реалізацію та розширення.

2.3 Висновки по розділу

За рахунок всіх покращень результатом стане архітектурне рішення для підтримки студентської діяльності, що забезпечить комфортний та швидкий спосіб отримання розкладу та новин конкретного студента в зручному для обробки форматі з можливістю розширення та масштабування системи без великої кількості витрат фінансових та трудових ресурсів.

РОЗДІЛ 3
ПРОЕКТУВАННЯ АРХІТЕКТУРНОГО РІШЕННЯ СИСТЕМИ ІНФОРМАЦІЙНОЇ ПІДТРИМКИ СТУДЕНТІВ

3.1 Аналіз варіантів використання

Для даного архітектурного рішення, на основі аналізу існуючих систем підтримки студентської діяльності, було сформовано декілька варіантів використання, таких як:

- веб-сайт з тонким клієнтом та API в мережі Інтернет;
- чат-бот та API у месенджерах (наприклад, Telegram, Viber або Messanger);
- мобільний додаток на основі Java або Xamarin;
- стаціонарний термінал з підключенням до серверу по локальній мережі, що включає в себе сайт-клієнт, та API на сервері.

В процесі підготовки рішення був обраний останній варіант, так як всі інші варіанти мають деякі недоліки, а саме:

– веб-сайти в мережі були реалізовані на момент вибору, а також вони не дозволяють використовувати периферійні пристрої, наприклад, сканер для зчитування штрих-коду на студентському квитку;

– чат-бот має такі ж недоліки як веб-сайт, окрім того зовнішній вигляд інтерфейсу, функціональність та зручність користування бота залежить від платформи на якій він реалізований;

– створення мобільного додатку досить трудомісткий процес, потребує залучення, як мінімум, невеликої команди спеціалізованих програмістів, дизайнерів та тестувальників, тощо, також при публікації додатку виникає проблема в створенні профілю розробника в магазинах додатків Google Play та App Store, для цього необхідна одноразова оплата в розмірі 25$ та 100$ відповідно, що є додатковими економічними витратами.

Так як реалізація архітектурного рішення з підтримкою стаціонарного терміналу є найбільш придатним варіантом з функціональної та економічної точки зору тому вибір зупинився на ньому, також він стане зручною альтернативою паперовим носіям та може бути виконаний в декількох варіантах та розміщений при входах в корпуси університету, де студенти проходять кожен день.

3.2 Моделювання процесів архітектурного рішення інформаційної підтримки студентів

Так як дане архітектурне рішення передбачає використання периферійних пристроїв надалі буде розглядатися взаємодія студента-користувача з терміналом на якому встановлене програмне забезпечення, що реалізується в цій роботі.

При моделюванні процесів були взяті до уваги основні функціональні необхідності рішення, а саме:

– ідентифікація конкретного студента – реалізовано методом сканування штрих-коду, що є на кожному студентському квитку;

– отримання розкладу для студента або групи в якій він навчається на кожен день тижня;

– отримання списку актуальних новин для студента або групи в якій він навчається.

Процес взаємодії студента з терміналом зображений на діаграмі прецедентів, що представлена на рисунку 3.1, та описує його на концептуальному рівні. На ній зображені можливості описані вище в більш детальному вигляді, але без конкретизації компонентів, що взаємодіють між собою для отримання результату. Робота над нею була почата в першу чергу, так як вона допоможе зрозуміти основну функціональність системи в самому базовому вигляді.

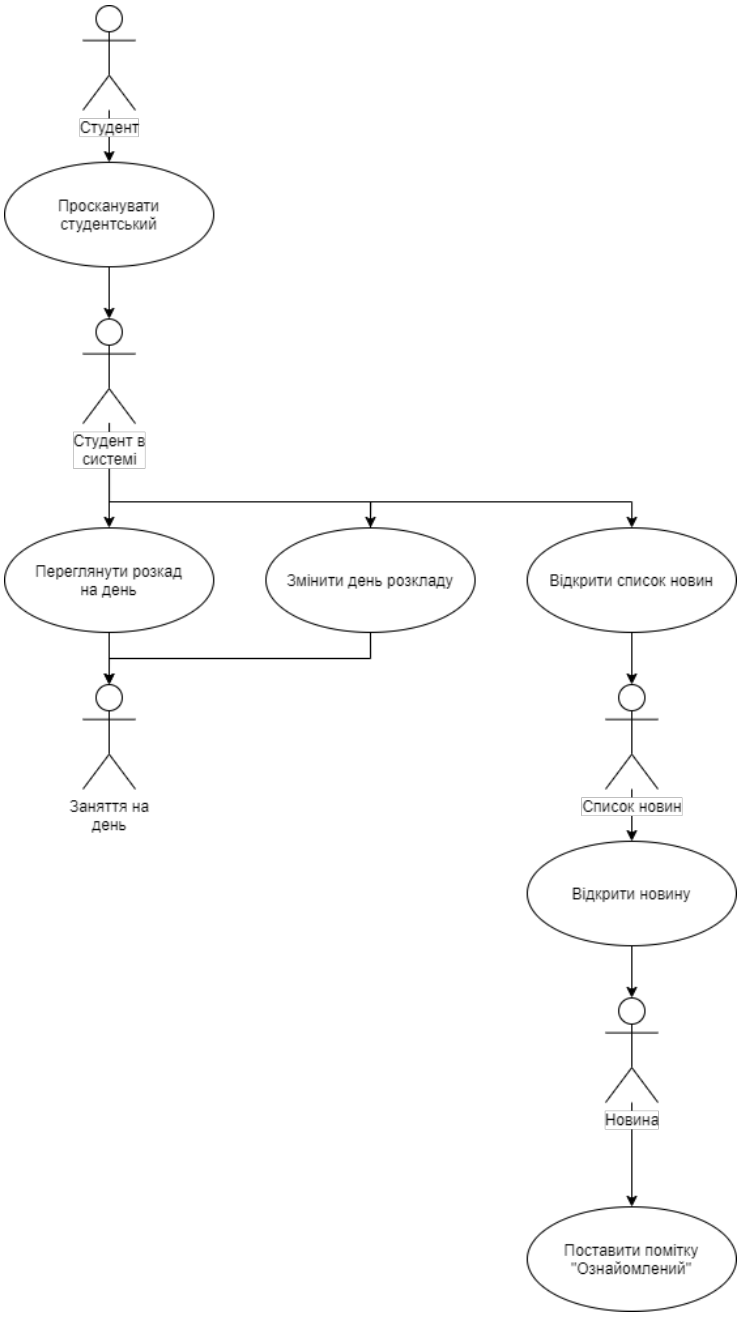


Рисунок 3.1 – Діаграма прецедентів програмного забезпечення терміналу підтримки студентської діяльності

На даній діаграмі можна побачити, що користувач-студент може отримати доступ до інформації через сканування штрих-коду на своєму студентському білеті, далі він може отримати доступ до розкладу, змінити день на який необхідно відобразити список занять, відкрити список новин зокрема і конкретну новину, а також відмітити, що він з нею ознайомився.

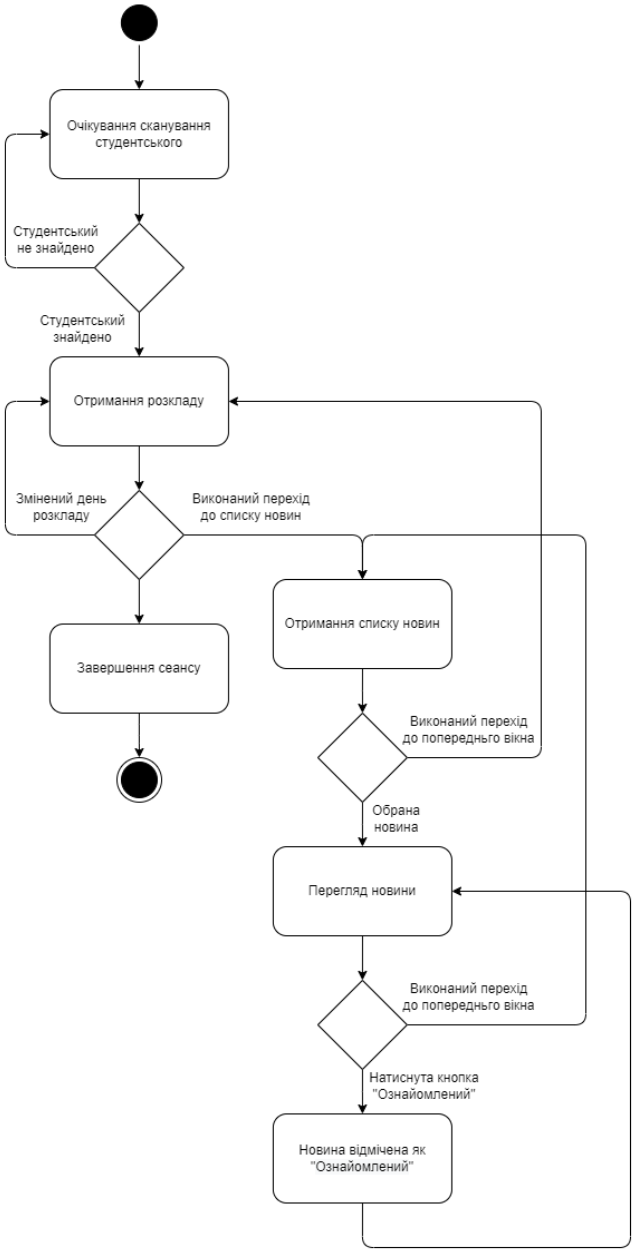


Рисунок 3.2 – Діаграма діяльності програмного забезпечення терміналу підтримки студентської діяльності

Алгоритм роботи архітектурного рішення, а також сам процес отримання розкладу та новин описаний в діаграмі діяльності, що зображена на рисунку 3.2. На ній ми можемо спостерігати, що є можливість повернутися до попередніх вікон після етапу проходження ідентифікацій (сканування штрих-коду). Також передбачений механізм, що забезпечує зберігання інформації до закінчення роботи з терміналом(вихід із системи), а при відсутності взаємодії з боку клієнта через 30 секунд буде виконане автоматичне завершення сеансу.

3.3 Моделювання даних архітектурного рішення інформаційної підтримки студентів

Для подальшої розробки постала необхідність опису всіх моделей, що використовуються для зберігання даних в даному архітектурному рішенні, які взаємодіють в його різних модулях. Тому була створена ER-діаграма з переліком всіх необхідних сутностей, що зображена на рисунку 3.3.

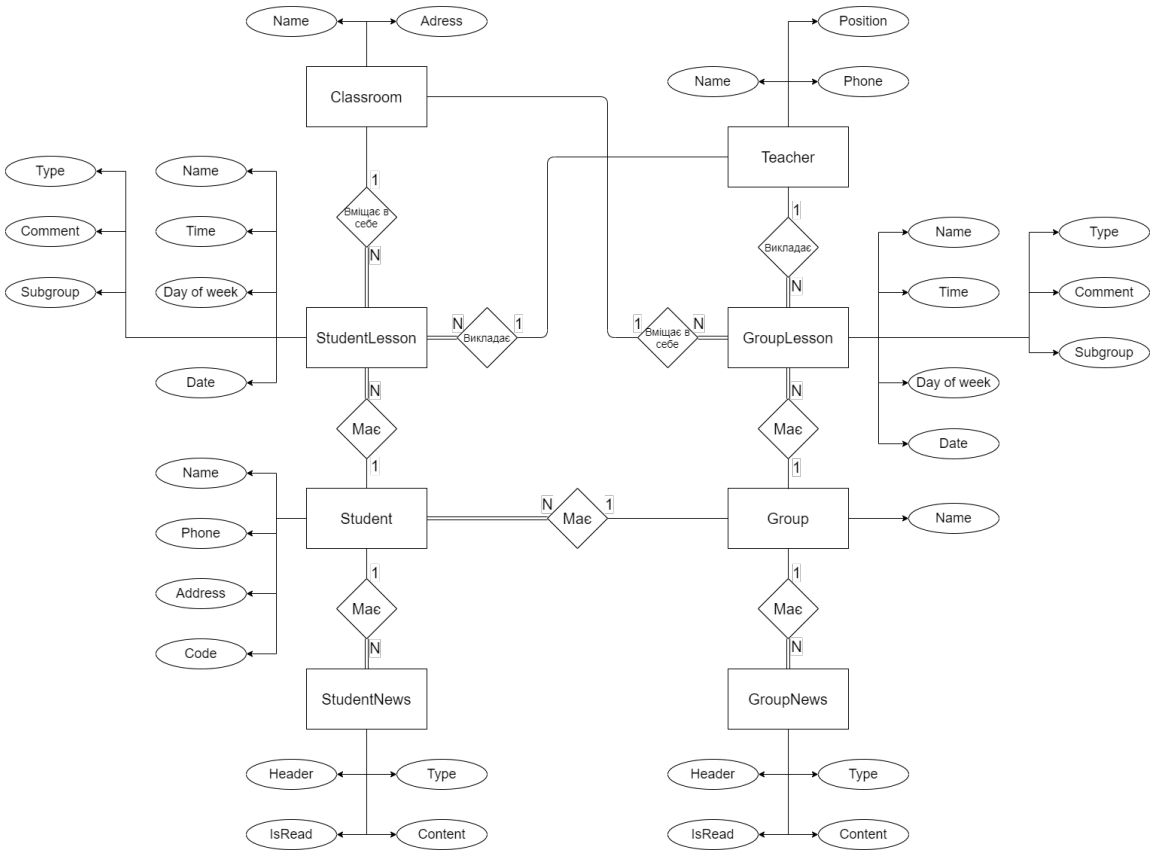


Рисунок 3.3 – ER-діаграма архітектурного рішення програмного забезпечення терміналу підтримки студентської діяльності

На даній діаграмі зображені сутності, їх зв’язки та поля, що вони включають. Найголовнішою сутністю являється студент (Student), так як вона відображає самого клієнта програмного забезпечення. Розклад зберігається в виді списку занять для груп та студентів (StudentLesson/GroupLesson). Сутність група (Group) необхідна для узагальнення даних студентів однієї групи та оптимізації зберігання даних у БД, наприклад, якщо у цілої групи

студентів в розкладі одна і та ж сама лекція, то більш розумним виходом буде зв’язати її з групою, а не з кожним окремим студентом, це забезпечить економію записів у БД та зменшить процес заповнення таблиць даними відповідальними працівниками адміністрації. Такий самий підхід використовується і для новин студентів та груп(StudentNews/GroupNews). Також для нормалізації БД інформація про викладачів(Teacher) та аудиторії(Classroom) була винесена в окремі сутності.

3.4 Проектування клієнт-серверної архітектури

В процесі проектування архітектури програмного забезпечення терміналу підтримки студентської діяльності було прийняте рішення щодо реалізації клієнт-серверної архітектури з тонким клієнтом, так як передбачається наявність великої кількості терміналів для користування студентами в різних корпусах університету. Варіант зі створенням ПЗ, що буде виконувати підключення до БД напряму був відкинутий, так як він має декілька недоліків, а саме:

– додаткові економічні витрати на термінали для підвищення їх обчислювальної потужності;

– зменшення надійності всієї системи, так як доступ до БД буде окремим для кожного клієнта та може призвести до непередбачених наслідків;

– зменшення захищеності всієї системи, так як всі операції будуть виконуватися саме клієнтом;

– збільшення затрат часу на конфігурацію кожного окремого терміналу;

– потенційний ризик несанкціонованого копіювання ПЗ.

Для реалізації була обрана трьохрівнева клієнт-серверна архітектура з тонким клієнтом, вона передбачає те, що на рівні клієнта буде відсутня будь яка бізнес-логіка, що виконує обчислення, а рівень зберігання даних буде відокремлений від прикладного рівня, тобто база даних буде знаходитися на

окремому сервері, а отже при виході з ладу серверу прикладного рівня, що виконує основну роботу системи, дані не постраждають.

Діаграма клієнт-серверної архітектури для програмного забезпечення на основі проектованого архітектурного рішення показана на рисунку 3.4.

Рисунок 2.4 – Модель трьохрівневої клієнт-серверної архітектури програмного забезпечення терміналу підтримки студентської діяльності

3.5 Проектування структури клієнтської частини

При проектуванні структури клієнтської частини були поставлені декілька вимог до реалізації:

- мінімальні вимоги до обчислювальної потужності сенсорного терміналу на якому буде розгорнута клієнтська частина;

- підтримка крос-платформеності (на випадок зміни платформи на планшетні ПК, телевізори, тощо);
- зручний та інтуїтивно зрозумілий дизайн;
- мінімальна кількість часу затрачуваного на початкову конфігурацію платформи на якій буде розгорнута клієнтська частина;
- можливість швидкого оновлення клієнта в випадку додаткових доопрацювань.

Обраним варіантом став спеціальний веб-ресурс в вигляді сайту, який доступний виключно з локальної мережі та розроблений тільки для відповідних сенсорних терміналів. Такий підхід забезпечить підвищений ступінь захисту від потенційних дій, що можуть спричинити збій в системі або втрату даних, так як для доступу до клієнта необхідно бути під'єднаним до внутрішньої мережі університету. Також швидкість з'єднання з сервером буде максимальною. Було прийняте рішення використовувати наступні інструменти:

- HTML та CSS для створення розмітки сайту;
- JavaScript, як основну мову програмування для виконання логіки доступу;
- jQuery – для спрощення написання програмного коду на JavaScript, так як ця бібліотека вміщує в себе готові функції для роботи з елементами розмітки та відправки запитів до серверу;
- Bootstrap – для доступу до більшої кількості вже готових елементів розмітки.

Так як всі дії на стороні клієнта виконуються шляхом взаємодії, по більшій частині, з кнопками або іншими схожими елементами то був обраний підхід з тонким клієнтом, що тільки передає дії на сервер, а отже відпадає необхідність створення діаграм та проектування складних підсистем для клієнтської частини.

3.6 Проектування структури серверної частини

При проектуванні структури серверної частини були поставлені декілька вимог до реалізації:

- висока надійність роботи системи;
- велика гнучкість у плані розширення системи;
- можливість швидких доопрацювань системи;
- можливість переносу програми-сервера на інші сервери;
- мінімальна кількість початкових налаштувань перед введенням в експлуатацію.

Обраним варіантом став додаток ASP.NET Core, що дає доступ до API реалізований на об’єктно-орієнтованій мові програмування C# зі строгою типізацією, що в свою чергу забезпечує високу надійність програмного коду та дуже малу кількість помилок, що можуть бути допущені на момент написання рішення.

Для реалізації була обрана Onion("цибулева")-архітектура, її концепція була запропонована в 2008 році Джеффрі Палермо, швидко набула популярності та закріпилася як найбільш застосовувана для додатків на основі ASP.NET. Ця архітектура включає в себе декілька проектів з логікою пов’язаних між собою за допомогою абстракцій та рівень моделей (рисунок 3.5), що використовуються у всьому додатку. Це забезпечить велику гнучкість кодової бази, можливість легко виконувати розширення функціоналу та виправлення помилок в майбутньому.

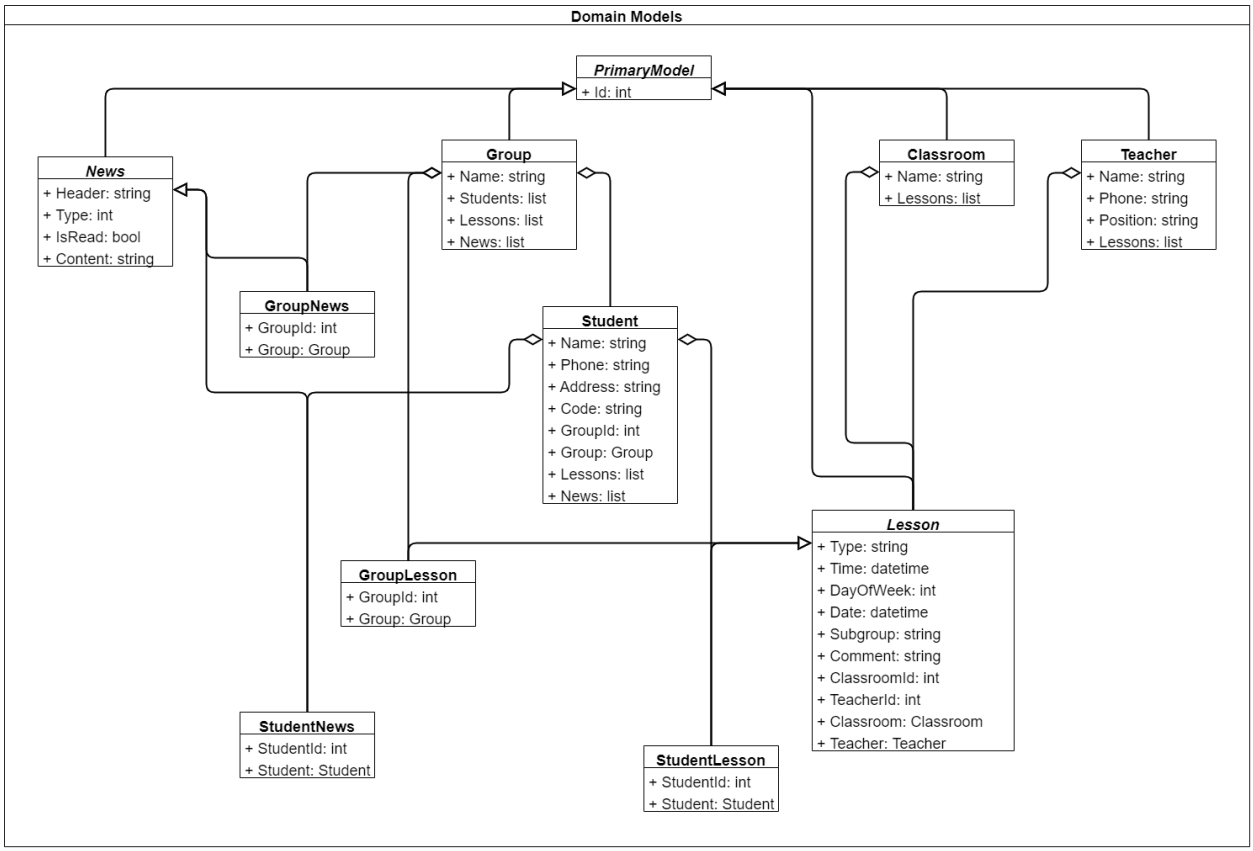


Рисунок 3.5 – Діаграма класів моделей предметної області серверної частини програмного забезпечення терміналу підтримки студентської діяльності

На даній діаграмі зображені моделі, що приймають участі в роботі серверної частини, та їх зв’язки. На ній ми можемо спостерігати, що всі моделі успадковуються від абстрактного класу PrimaryModel, а заняття та новини студентів або груп успадковуються від абстрактних класів Lesson та News відповідно, це необхідно для зменшення повторюваних полів в похідних класах та підтримки поліморфізму.

3.7 Висновки по розділу

В даному розділі розглянуті основні процеси і дані, що приймають участь під час роботи та на етапах моделювання системи. Були створені необхідні UML-діаграми для більш чіткого зрозуміння вимог до архітектурного рішення на абстрактному рівні. Був проведений аналіз наявних архітектур, що придатні для реалізації поданих вимог архітектурного рішення,

та обрана клієнт-серверна архітектура з тонким клієнтом, де є чітке розподілення між клієнтським, прикладним рівнями та рівнем даних. Також був обраний необхідний кластер технологій для реалізації клієнтської та серверної частини, сформований початковий набір моделей предметної області та їх зв’язки між собою.

РОЗДІЛ 4
РЕАЛІЗАЦІЯ АРХІТЕКТУРНОГО РІШЕННЯ ІНФОРМАЦІЙНОЇ ПІДТРИМКИ СТУДЕНТІВ

4.1 Реалізація серверної частини архітектурного рішення

4.1.1 Набір технології для розробки серверної частини

Серверна частина даного архітектурного рішення була розроблена з використанням мови програмування C#, на основі модульної платформи для розробки програмного забезпечення з відкритим вихідним кодом .NET Core 3.1. Таке рішення було прийнято, так як ця платформа дозволяє реалізовувати різноманітні рішення для різних потреб, наприклад консольні програми, додатки для робочого столу Windows, сервери API, мобільні додатки, веб-сайти, тощо.

В випадку реалізації спроектованої архітектури для серверної частини програмного забезпечення терміналу підтримки студентської діяльності був обраний варіант з реалізацією рішення де основним виконуваним модулем буде проект ASP .NET Core Web API, забезпечує інтерфейс прикладного програмування для доступу до даних, що оброблюються сервісами та репозиторіями. При створенні методів, що виконують функцію приймання запитів(ендпоінтів) був використаний підхід до архітектури мережевих протоколів REST (скор. англ. Representational State Transfer, «передача репрезентативного стану») Також будуть присутні проекти бібліотек класів .NET Core в яких будуть розміщені інші шари Onion-архітектури, що знаходяться нижче ніж шар користувацького інтерфейсу. Всі шари будуть описані в розділах нижче.

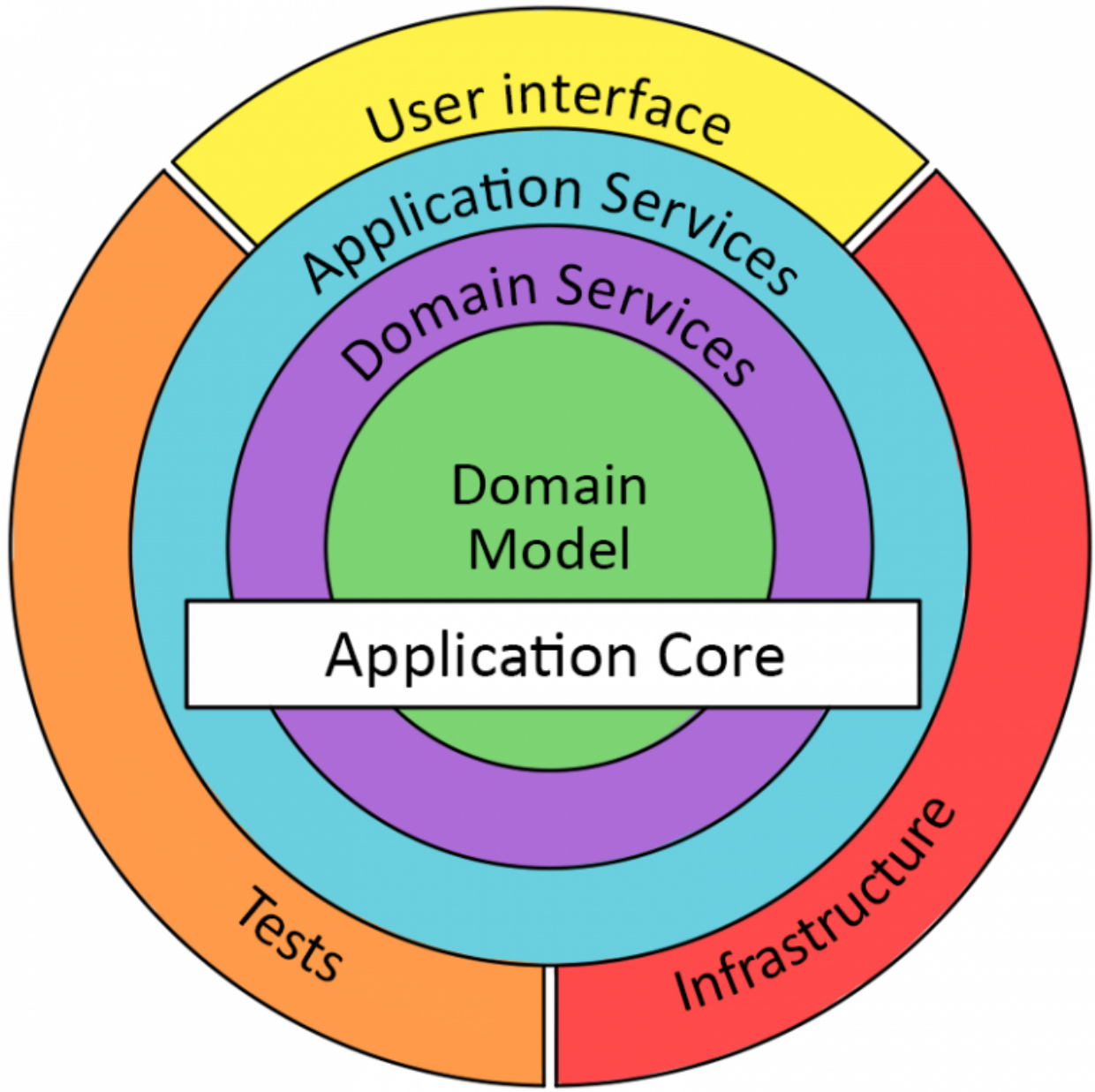


Рисунок 4.1 – Діаграма елементів Onion-архітектури

Так як базова функціональність фреймворку .NET Core 3.1 не дозволяє покрити всі потреби рішення постала необхідність використовувати пакети з вільної системи керування пакунками NuGet. Всього було використано п’ять пакетів, а саме:

– Microsoft.EntityFrameworkCore – пакет для роботи з майже будь якою БД за допомогою компоненту LINQ (англ. Language Integrated Query – запити, інтегровані в мову) на основі структури об'єктно-реляційного проекції (ORM);

– MySql.EntityFrameworkCore – пакет, що виконує роль додатку до Entity Framework Core та дозволяє виконувати підключення до БД MySQL;

– Microsoft.EntityFrameworkCore.Design – пакет, що додається до виконуваного проекту та дозволяє виконувати команди до модулю Entity Framework Core, наприклад створення міграцій (створення або модифікація БД на основі написаного коду);

– Microsoft.EntityFrameworkCore.Tools – пакет, що додається до проекту з контекстом бази даних та, також, дозволяє виконувати команди до модулю Entity Framework Core;

– Swashbuckle.AspNetCore – пакет, що надає можливість створювати інтерактивну документацію API на основі специфікації машиночитабельних файлів з інтерфейсами OpenAPI (також відома як Swagger) для проектів ASP .NET Core.

4.1.2 Опис проекту бібліотеки класів моделей предметної області (Domain.Models)

В даному проекті присутні два каталоги з класами-моделями: Database та Transaction. Кожен каталог містить моделі необхідні для певних задач, а саме: Database-моделі необхідні для відображення сутностей в БД та роботи з ними в сервісах; Transaction-моделі використовуються для відправки їх на front-end частину, в них менше властивостей та присутні деякі властивості що не зберігаються в БД.

Моделі простору імені Domain.Models.Database були реалізовані по діаграмі класів, що описана в розділі 3.6 на рисунку 3.5, але з деякими модифікаціями які необхідні для роботи технології Entity Framework Core(надалі EF):

– згідно до діаграми класів всі моделі що зберігаються в БД наслідковуються від абстрактного класу PrimaryModel, що містить властивість

Id, таке іменування необхідне так як EF сприймає властивості з такою назвою як первинні ключі та автоматично налаштовує індекси;

– для створення зв’язків "багато до одного" між таблицями в БД до моделей необхідно додати певну властивість(навігаційна властивість) в модель: для сутностей, що відображують багато елементів в зв’язку – віртуальну властивість з типом головної сутності, а для сутностей, що відображують один елемент в зв’язку – віртуальну властивість колекції типу підлеглої моделі (властивості потрібно робити обов’язково віртуальними, так як при роботі з колекціями типу DbSet<T> в EF використовуються не саме ті класи, що вказані в коді, а наслідувані з них автоматично створенні типи, що являють собою проміжні моделі з додатковими властивостями, які необхідні для відслідковування змін при виконуванні зберігання змін);

– також для забезпечення зв’язків також можна створити властивості, що будуть використовуватися як зовнішні ключі в БД, стандартом є властивість типу int, назва якої містить назву класу до якого виконується прив’язка та суфікс Id, це не обов’язкова потреба, так як EF автоматично створює такі властивості в проміжних типах та призначає їм індекси в БД, але це досить поганий підхід так як при ньому відсутній контроль за тим як між собою пов’язані сутності в базі даних.

Список всіх моделей предметної області являє собою набір з трьох абстрактних класів та восьми класів, що відображують сутності які зберігаються в БД:

– PrimaryModel – абстрактний клас загальної моделі;

– Lesson – абстрактний клас загальної моделі заняття;

– News – абстрактний клас загальної моделі новини;

– Group – клас моделі групи, що пов’язує між собою студентів та заняття/новини їх груп;

– GroupLesson – клас моделі заняття групи;

– GroupNews – клас моделі новини групи;

– Student – клас моделі студента з його основними даними;

– StudentLesson – клас моделі заняття конкретного студента;

– StudentNews – клас моделі новини конкретного студента;

– Classroom – клас моделі для опису аудиторії в якій проводиться заняття, вона винесена в окремий клас, так як різні заняття можуть проводитися в одній аудиторії;

– Teacher – клас моделі для опису викладача, що проводить заняття, він, як і аудиторія, винесений в окремий клас по причині того, що один викладач проводить заняття у декількох груп або окремих студентів.

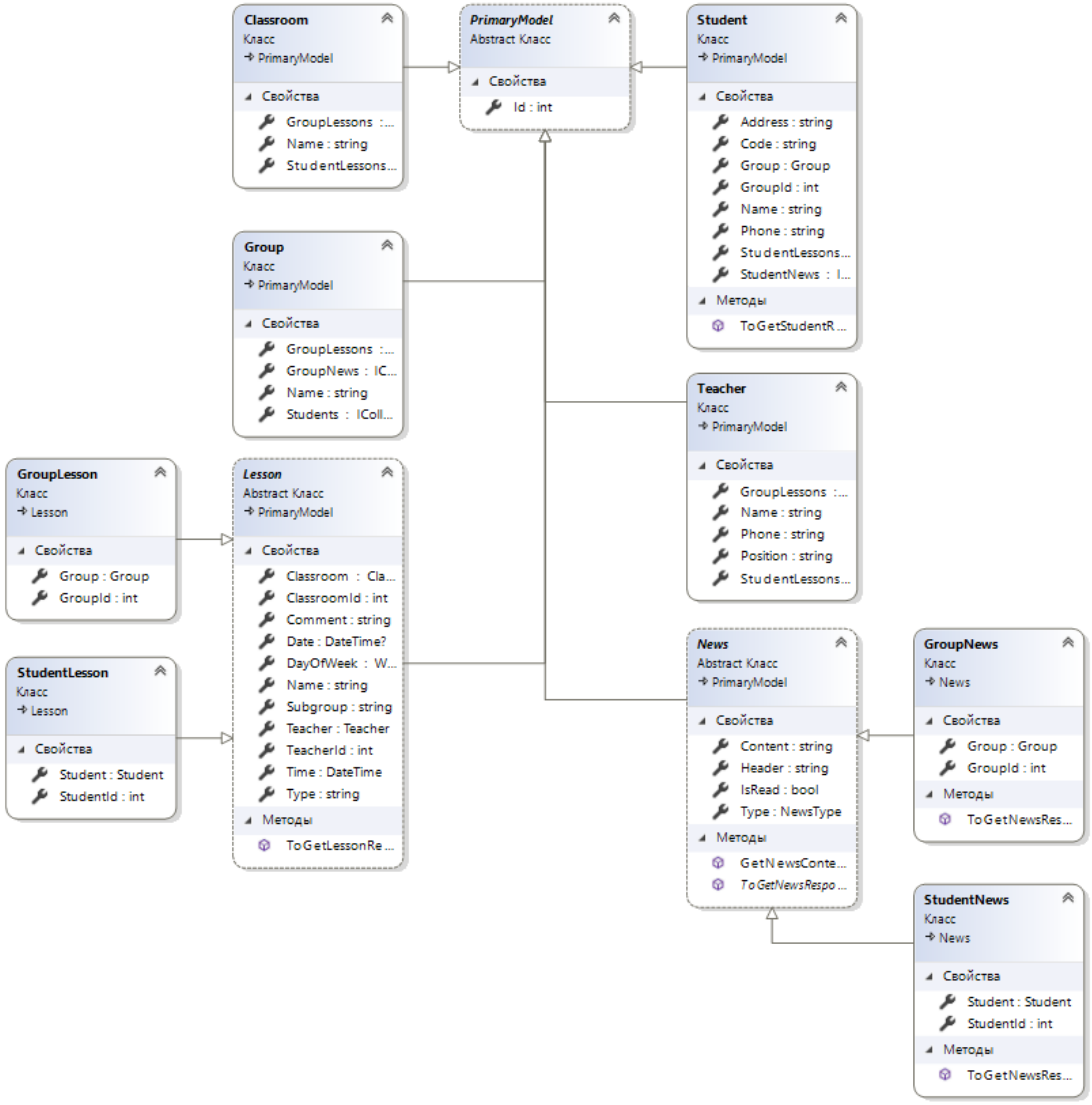


Рисунок 4.2 – Діаграма класів простору імен Domain.Models.Database в конструкторі класів Visual Studio

В свою чергу моделі простору імен Domain.Models.Transaction реалізовані як окремі класи, що мають деякі спільні з моделями предметної області властивості. Доцільність їх використання полягає в тому, що:

– деякі властивості моделей предметної області являють собою зв’язані сутності, що отримуються з БД та їх не потрібно передавати в відповіді на запит до API;

– вони вміщають деякі властивості, що вираховуються на етапі обробки в сервісі та не можуть бути отриманими напряму з БД;

– набір їх властивостей не залежить від даних з бази, за рахунок чого такий підхід додає гнучкості при роботі з API;

– конвертація Domain моделей в моделі відповідей на запити(транзакцій) виконується за рахунок методів в класах простору імен Domain.Models.Database, тому відпадає необхідність в додаткових валідаціях даних, так як дані з БД напряму, можливо, з деякими допоміжними розрахунками, транслюються в відповідь на запит.

Список всіх моделей предметної області являє собою набір з шести класів, що відображують відповідь на певний набір однотипних запитів:

– GetStudentResponse – клас моделі відповіді на запит отримання даних студента без зайвих зв’язаних сутностей;

– GetLessonResponse – клас моделі відповіді на запит отримання списку занять зі зв’язаними сутностями аудиторії та викладача;

– GetClassroomResponse – клас моделі інформації про аудиторію без зайвих зв’язаних сутностей, що в свою чергу використовується в моделі GetLessonResponse;

– GetTeacherResponse – клас моделі інформації про викладача без зайвих звязаних сутностей, що в свою чергу використовується в моделі GetLessonResponse;

– GetNewsResponse – клас моделі відповіді на запит отримання списку новин, що представляє собою загальний вид новини з розділенням на

персональні та групові, а також з відсутністю вмісту новини, це зроблено для оптимізації відповідей, так як вміст може бути досить великий в одній новині, а новин в списку знаходиться досить багато;

– GetNewsContentResponse – клас моделі відповіді на запит отримання вмісту конкретної новини, що необхідний для запобігання передачі великої кількості трафіку без необхідності.

4.1.3 Опис проекту бібліотеки класів конфігурації (Infrastructure.Configuration)

Даний проект доданий для того, щоб вміщувати в себе різні класи та інші елементи рішення, що не підходять для зберігання на інших шарах, а саме:

– перерахування (enum) – спеціальні конструкції для зберігання чітко детермінованих значень в зручному для читання вигляді;

– атрибути (attributes) – спеціальні класи для додання функціональності до класів, властивостей та методів без внесення змін до інших частин коду;

– методи розширень (extension methods) – статичні класи з набором методів для різних класів, наприклад, для обробки логіки атрибутів;

– класи з константами (constants) – класи з переліком значень визначених на моменті компіляції для зберігання "чарівних" чисел та рядків, наприклад, індексів або розділів в файлах конфігурації;

– користувацькі виключення (exceptions) – класи для опису користувацьких помилок, що можуть виникати в певних ситуаціях під час виконання програми (runtime)

В нашому випадку в проекті присутні лише два перерахування:

– WeekDay – для відображення, зберігання в БД та обробки дня тижня;

– NewsType – для відображення, зберігання в БД та обробки типу новини (PlainText, HTML, JSON), цей функціонал доданий для потенційної модифікації механізму відображення новин на сайті.

Інші типи елементів проекту, що описані вище відсутні, так як в них відпадає необхідність, адже в даному випадку це тільки збільшило б складність системи, а основна функція API тільки повертати значення що були внесені в базу даних з інших джерел, тому перевірка на помилки та збереження певних значень в константах не потрібна для коректної обробки даних.

4.1.4 Опис проекту бібліотеки класів контексту даних БД (Domain.Context)

Даний проект призначений для зберігання класів, що забезпечують роботу всього додатку з пакетом Entity Framework.

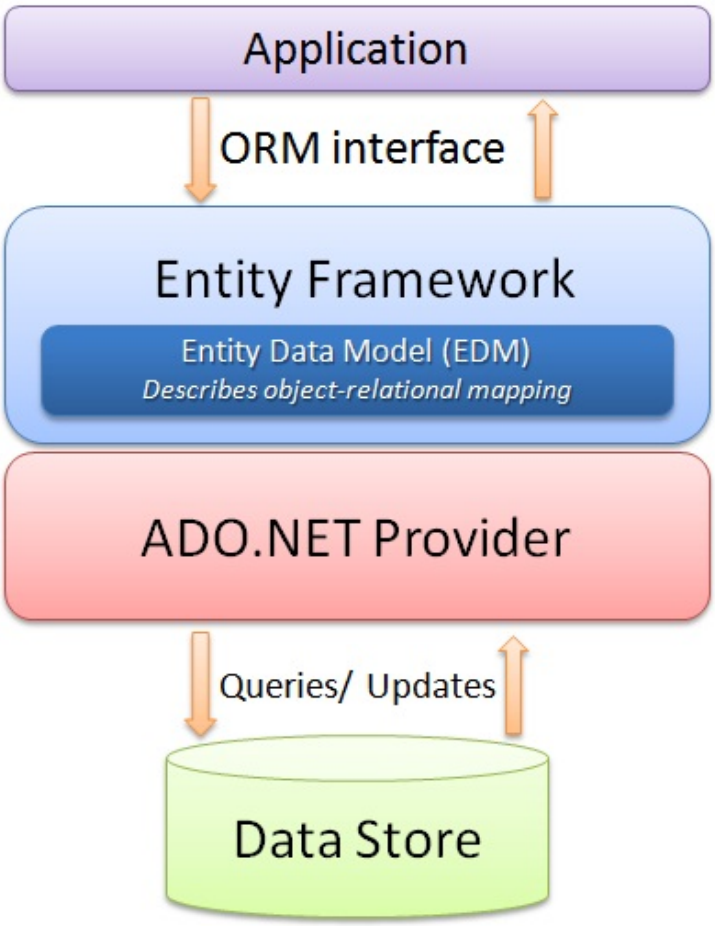


Рисунок 4.3 – Архітектура роботи Entity Framework

На рисунку вище представлена архітектура роботи EF вона розділена на п’ять частин:

– ORM interface – це інтерфейс взаємодії, що звязує базу даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних;

– Entity Framework – це внутрішня логіка роботи пакету, що включає в себе модель даних з використанням сутностей – набір основних понять, що описують структуру та забезпечують перетворення даних незалежно від форми зберігання;

– ADO.NET Provider – це постачальник даних, що забезпечує зв’язок та формування запитів для конкретного типу БД;

– Queries/Updates – це безпосередньо запити та відповіді до/від бази даних в зручному для неї вигляді;

– Data Store – це сама база даних, що знаходиться локально або на віддаленому спеціалізованому сервері.

При створенні класів для підключення до бази даних був використаний підхід Code First – це підхід, що передбачає реалізацію контексту даних та моделей в першу чергу, а вже на їх основі буде згенерована сама БД за допомогою внутрішніх команд та інструментів EF.

В коді представлено клас UserContext – це клас контексту даних, що наслідковується від класу DbContext, котрий в свою чергу являє собою сесію підключення до БД та може бути використаний для отримання, створення, зміни та видалення даних з однієї або декількох таблиць. Найчастіше клас складається з трьох елементів:

– набір властивостей DbSet<T> – це узагальнений клас, що відповідає конкретній таблиці в БД, яка, за замовчуванням, має назву як властивість та дозволяє працювати з цією таблицею як з колекцією елементів, тобто виконувати фільтрацію, сортування та групування, а вже при отриманні

результату всі ці дії за допомогою внутрішньої логіки Entity Framework транслюються в SQL-запити, деякі приклади запитів будуть описані нижче;

– метод OnConfiguring – наслідуваний віртуальний метод, який перевантажується в похідному класі та відповідає за конфігурацію підключення до бази даних(в нашому випадку використовується метод UseMySQL(), що дозволяє виконати підключення до БД MySQL, він стає доступним при підключенні пакету MySql.EntityFrameworkCore, так як за замовчуванням EF дозволяє роботу лише з серверами MS SQL і для роботи з іншими необхідно встановлення інших постачальників даних), найчастіше тут лише вказується рядок підключення до серверу БД, але є можливість налаштування внутрішньої логіки під свої потреби, наприклад, додавання логерів(класів для створення детальних заміток роботи);

– метод OnModelCreating – наслідуваний віртуальний метод, який перевантажується в похідному класі та відповідає за налаштування зіставлення класів, що використовуються для створення властивостей DbSet<T> та таблиць в БД, найчастіше використовуються наступні конструкції:

– налаштовуються обмеження щодо рядкових властивостей класів, наприклад, імен в таблицях – метод HasMaxLength(int maxLength);

– встановлюється прапорці необхідності до окремих полів таблиці в БД – метод IsRequired(bool required);

– встановлюються довільні перетворювачі значень від типів присутніх в C# до типів відповідної СУБД – метод HasConversion(ValueConverter<TProperty, TProvider> converter);

– налаштовуються зв’язки між класами та таблицями в БД (при підході Code First) – комбінація методів HasMany<TR>(Expression navigationExpression) та WithOne(Expression navigationExpression).

При Code First підході таблиці БД створюються, налаштовуються та видаляються за допомогою міграцій – спеціальних класів, що за допомогою

вбудованих в Entity Framework методів виконують скрипти до серверу бази даних та виконують там зміни. Ці файли генеруються автоматично за допомогою команди "add-migration [Назва міграції]" – при її виконанні EF виконує порівняння коду, що описаний в контексті даних(клас, що успадковується від DbContext) разом з зв’язаними класами моделей та наявної БД що описана в рядку підключення в методі OnConfiguring, а потім генерує код для внесення змін у існуючу базу даних так, щоб налаштування таблиць співпадало з налаштуваннями в контексті даних. Це можуть бути команди на створення таблиць, зміну типів або обмежень окремих колонок, створення або видалення зовнішніх ключів та створення/видалення індексів. Всі ці файли розміщуються в каталозі, що розміщується в корені проекту, під назвою Migrations, але це також можна окремо налаштувати. Виконання команди add-migration лише створює необхідну міграцію, але не застосовує її, для цього необхідно виконати команду update-database, що в свою чергу говорить Entity Framework’у виконати впровадження змін. Усі перераховані вище команди виконуються за допомогою Консолі диспетчера пакетів (Package Manager Console) (спеціального інструменту Visual Studio, що дозволяє виконувати команди що передбачені пакетами NuGet) та двох пакеті: Microsoft.EntityFrameworkCore.Design – для реалізації логіки генерації міграцій та Microsoft.EntityFrameworkCore.Tools – для безпосереднього додавання та виконання команд в консолі.

4.1.5 Опис проекту бібліотеки класів інтерфейсів репозиторіїв (Domain.Interfaces)

Так як обрана Onion-архітектура передбачає роботу з абстракціями репозиторіїв або сервісів, а не з їх реалізаціями, що в свою чергу забезпечує використання одного з принципів SOLID, а саме Принцип інверсії залежностей (Dependency inversion principle), тому репозиторії (об’єкти, що використовуються для отримання даних з різних джерел) розділені на класи та інтерфейси, які розташовані в різних проектах для забезпечення розмежування

типів. Також в даній реалізації частково дотримується Принцип розділення інтерфейсів (Interface segregation principle) – всі інтерфейси репозиторіїв працюють тільки з одною таблицею або з двома таблицями, що зберігають спільні дані і не пересікаються з іншими моделями під час обробки. Це все разом забезпечує гнучкість в отриманні даних і можливість реалізації підключень до абсолютно різних джерел, наприклад відправка запитів в сторонній API, використання декількох БД або, навіть, зчитування з файлу.

В просторі імен Domain.Interfaces наявні три інтерфейси для отримання всіх необхідних даних, а саме:

– IStudentRepository – інтерфейс для роботи з інформацією про студента, що визначає метод GetStudentByCode(string code) для отримання конкретного студента по його коду;

– ILessonRepository – інтерфейс для роботи з інформацією про заняття студентів та груп, що визначає метод GetLessonsByStudentId(int studentId, DateTime date, WeekDay dayOfWeek) для отримання переліку занять студента по його ідентифікатору за конкретну дату/день тижня;

– INewsRepository – інтерфейс для роботи з інформацією про новини студентів та груп, що визначає методи GetNewsByStudentId(int studentId), GetGroupNewsById(int groupNewsId), GetStudentNewsById(int studentNewsId), ChangeIsReadState(int studentNewsId) для отримання загального переліку новин студента по його ідентифікатору; отримання змісту новини групи по її ідентифікатору; отримання змісту новини студента по її ідентифікатору; відмічення новини з якою студент ознайомлений по її ідентифікатору відповідно.

4.1.6 Опис проекту бібліотеки класів реалізацій репозиторіїв (Infrastructure.Data)

В даному проекті представлені реалізації інтерфейсів репозиторіїв, що знаходяться в просторі імен Domain.Interfaces, та виконують роботу з БД за допомогою контексту даних, що, в свою чергу, знаходиться в просторі імен Domain.Context. Всі вони реалізують методи представлені в відповідних інтерфейсах та за допомогою звернень до конкретних об’єктів типу DbSet<T> виконують отримання або зміну даних в зв’язаних таблицях.

В усіх методах використовується ініціалізація об’єкту UserContext, що являє собою реалізацію контексту даних, в конструкції using – це забезпечує те, що при створенні об’єкту виконується підключення до БД, а після виходу із конструкції в прихованій частині коду виконується виклик методу Dispose() для контексту в якому підключення закривається та вивільняється для інших абонентів. В середині конструкції using виконується звернення до відповідних властивостей типу DbSet<T>, наприклад context.Students та за допомогою використання підходу LINQ-to-Entities(використання запитів, інтегрованих в мову, що найчастіше працюють з колекціями, для формування запитів до БД) викликаються наступні методи:

– Include(Expression navigationPropertyPath) – метод для інформування Entity Framework, що разом з об’єктами обраної колекції необхідно виконати завантаження обраної зв’язаної сутності за допомогою SQL запиту JOIN;

– ThenInclude(Expression navigationPropertyPath) – метод схожий до попереднього, викликається в комбінації з ним та дозволяє завантажити дерево зв’язаних сутностей;

– AsNoTracking() – метод для інформування Entity Framework, що отримані сутності не потрібно відслідковувати на зміни за допомогою внутрішніх механізмів та розміщувати в кеші, це забезпечує підвищену швидкість виконання запитів на отримання інформації;

– FirstOrDefault(Expression predicate) – метод доступний в звичайному LINQ, він використовується для отримання першого результату, що задовольняє вказані умови або повернення значення за замовчуванням, в випадку LINQ-to-Entities, найчастіше, повертається значення null;

– Where(Expression predicate) – метод доступний в звичайному LINQ, він використовується для отримання колекції об’єктів, що задовольняють вказані умови, або пусту колекцію якщо підходящих результатів не знайдено;

– ToList() – метод доступний в звичайному LINQ, при звичних умовах він виконує обробку LINQ-конструкції та конвертує результат в колекцію, але в випадку LINQ-to-Entities вся конструкція конвертується в SQL-запит та відправляється на сервер БД для отримання результату(тому прийнято викликати цей метод в самому кінці конструкції після фільтрації(FirstOrDefault, Where), сортування(OrderBy, OrderByDescending), групування(GroupBy, GroupJoin) та трансляції(Select), так як всі ці методи перетворюються в частини оптимізованих SQL-запитів та зменшують навантаження на БД та час виконання команд).

Також в класі NewsRepository присутній метод ChangeIsReadState який виконує внесення змін до БД. Реалізація редагування рядків в базі даних складається з декількох етапів:

1) ініціалізація об’єкту класу UserContext в конструкції using;

2) отримання конкретного обєкта за допомогою виклику методу FirstOrDefault(Expression predicate) для властивості DbSet<T>;

3) перевірка на те чи існує об’єкт за допомогою конструкції if, так як результат попереднього методу може повернути null, якщо ні один рядок не задовольняє вказані умови;

4) зміна значення однієї або декількох властивостей знайденого об’єкту, при умові, що він існує;

5) виклик методу SaveChanges() для об’єкту типу UserContext, що сигналізує механізму Entity Framework, який відслідковує модифікації об’єктів, зафіксувати зміни та відправити відповідний набір запитів до серверу БД.

Так як студент може отримувати персональні новини та розклад, а також новини/розклад доступні для всієї групи тому в відповідних методах виконується два запити до БД для отримання персональних та групових новин/занять, а отже при поверненні результату методу їх необхідно поєднати в одну спільну колекцію за допомогою LINQ-методу Concat(IEnumerable<T> second), що викликається для однієї колекції та зкріпляє її с колекцією переданою в якості параметру до цього методу, а в випадку занять ще необхідно виконати сортування по їх часу. Також необхідно виконати виклик методу ToList() для того, щоб LINQ-запит видав результат відразу.

4.1.7 Опис проекту бібліотеки класів інтерфейсів сервісів бізнес-логіки (Services.Interfaces)

Так само як і в проекті бібліотеки класів інтерфейсів репозиторіїв (Domain.Interfaces) в даному проекті знаходяться інтерфейси сервісів які відповідають за обробку бізнес-логіки на основі даних, що отримуються з репозиторіїв. Так як ми повинні дотримуватися Принципу інверсії залежностей та Принципу розділення інтерфейсів при реалізації Onion-архітектури, тому логіка обробки даних та їх конвертація(mapping) винесена в окремі інтерфейси, для кожної групи сутностей. Так як в даному рішенні відсутня досить складна логіка, що потребує додаткової реалізації, тому сервіси представлені, як і репозиторії, в кількості трьох інтерфейсів:

– IStudentService – інтерфейс для обробки інформації про студента, що визначає такий самий метод як і відповідний йому репозиторій IStudentRepository, але значення, що повертається з методу вже є моделлю відповіді на запит API, що розташовані в просторі імен Domain.Models.Transaction (модель транзакції);

– ILessonService – інтерфейс для обробки інформації про заняття студентів та груп, що визначає такий самий метод як і відповідний йому репозиторій ILessonRepository, але значення, що повертається з методу вже є моделлю транзакції;

– INewsService – інтерфейс для обробки інформації про новини студентів та груп, що визначає такий самий метод як і відповідний йому репозиторій INewsRepository, але значення, що повертається з методу вже є моделлю транзакції.

4.1.8 Опис проекту бібліотеки класів реалізацій сервісів бізнес-логіки (Infrastructure.Logic)

В даному проекті представлені реалізації інтерфейсів сервісів бізнес-логіки, що знаходяться в просторі імен Services.Interfaces, та виконують обробку даних, що отримуються з репозиторіїв, інтерфейси яких, в свою чергу, знаходяться в просторі імен Domain.Interfaces. Всі вони реалізують методи представлені в відповідних інтерфейсах сервісів та за допомогою виклику конкретних методів репозиторіїв, а потім виконання проекцій результатів забезпечують роботу бізнес-логіки.

Всі класи використовують шаблон Впровадження залежностей (англ. Dependency injection, DI) – він дозволяє передавати залежності клієнту ззовні, а не вимагає від нього створення залежностей в його власній логіці. Цей шаблон реалізує принцип Інверсії керування (англ. Inversion of Control, IoC), який в свою чергу допомагає зменшити зв'язність модулів (міра в якій компонент програми залежить від кожного іншого компоненту). Для реалізації шаблону DI використовується два елементи: властивість інтерфейсу репозиторію та конструктор, що приймає параметром об’єкт абстракції та ініціалізує властивість. Властивість може мати будь який модифікатор доступу та мати, або ж не мати блок {set;}, так як ініціалізація виконується в конструкторі. В свою чергу для забезпечення DI, без використання інших

шаблонів, сервіс повинен мати лише один конструктор в якому параметрами є абстракції всіх залежностей, що в ньому ініціалізуються.

Так як типами результатів, що повертаються з методів, є типи моделей транзакцій, а результатами роботи методів репозиторіїв є моделі предметної області, то необхідно реалізувати проекцію результатів. Це забезпечується тим, що всі моделі предметної області, що повертаються сервісами, мають спеціальні методи для перетворення(mapping) в моделі транзакцій. Отже при отриманні результату виклику методу репозиторію, якщо це один об’єкт, виконується виклик методу конвертації, якщо ж результатом, що повертається, є колекція об’єктів використовується LINQ-метод для проекції Select(Func<TSource, TResult> selector), що приймає метод для перетворення об’єкту одного типу в інший, а після нього викликається метод ToList() для виконання LINQ-запиту.

4.1.9 Опис проекту ASP .NET Core Web API прикладного програмного інтерфейсу (Interface.Api)

Для забезпечення доступу до даних від зовнішніх клієнтів постала необхідність в реалізації прикладного програмного інтерфейсу(API). Був обраний варіант використання кросплатформеного open-source фреймворку для створення веб-застосунків ASP .NET Core, який доступний на платформі .NET, а конкретніше окремий тип додатку Web API.

При початковому створенні проекту цього типу Visual Studio одразу генерує декілька класів для демонстрації роботи додатку, а саме додає модель WeatherForecast та окремий контролер в якому присутній метод-кінцева точкою(endpoint) для обробки GET-запитів. Тому у програміста завжди є можливість ознайомитися з прикладами створення API.

Для початку створення прикладного програмного інтерфейсу потрібно створити певний набір необхідних контролерів, найчастіше вони додаються у каталог Controllers. Контролер – це спеціальний клас, який визначає певний модуль в якому знаходяться спеціальні методи, які відіграють роль кінцевих

точок (endpoint), що мають певний шлях за допомогою якого відбувається спілкування клієнта з ним. Контролер має виконувати декілька вимог, щоб з ним можна було працювати клієнтам, а саме:

1) контролер повинен успадковувати спеціальний клас ControllerBase та за стандартом мати назву, що складається з імені модулю за, який він відповідає, з суфіксом Controller;

2) над визначенням класу контролеру повинен бути присутній атрибут [ApiController], це необхідно для того, щоб проміжне програмне забезпечення (middleware) ASP .NET Core та конвеєр обробки запитів чітко визначили конкретний клас як контролер та дозволили передачу запиту його кінцевим точкам;

3) над визначенням класу контролеру повинен бути присутній атрибут [Route] зі шляхом до контролеру, при умові що, відсутні абсолютні шляхи над методами-кінцевими точками та не визначений шлях за замовчуванням для всіх контролерів;

4) контролер має реалізовувати хоча б один метод зі спеціальною структурою, який буде вважатися кінцевою точкою контролеру. Сигнатура має передбачати наявність одного з семи атрибутів (HttpGet, HttpPost, HttpPut, HttpPatch, HttpDelete, HttpHead, HttpOptions), що відповідають семи типам HTTP-запитів (GET – отримання ресурсу; POST – відправка інформації на ресурс або його додавання; PUT – повна зміна ресурсу; PATCH – часткова зміна ресурсу; DELETE – видалення ресурсу; HEAD – отримання лише заголовків ресурсу, без його тіла; OPTIONS – отримання методів вказаних вище, що підтримуються сервером). До атрибуту, також, можливо додати модифікатор шляху(відносний або абсолютний), при його відсутності використовується кореневий шлях контролеру. Кращим підходом буде завжди визначати шляхи кінцевих точок, адже в ситуації наявності двох однакових методів класу з одним і тим самим атрибутом HTTP-методу та шляхом буде виникати конфлікт та помилка на етапі запуску серверу, так як конвеєр

обробки запитів не зможе чітко визначити на який метод контролеру передавати запит.

При реалізації методів, що відповідають кінцевим точкам також є декілька важливих моментів, а саме:

1) як було описано вище необхідна наявність атрибуту для визначення HTPP-методу;

2) шлях (route) до кінцевої точки може бути абсолютним (він додається безпосередньо до адреси серверу) або відносним (він додається до шляху контролеру, при умові його наявності). Для їх розмежування перед абсолютним шляхом ставиться символ "/", відносними вважаються всі інші шляхи;

3) параметри що приймаються в запиті описуються як параметри методу, але вони можуть бути отримані з різних частин запиту, тому є можливість додати атрибут до параметру, що визначить звідки слід брати інформацію для конкретного аргументу. За замовчуванням всі значення аргументів методів, що відповідають GET, HEAD або DELETE HTTP-методам, дістаються з параметрів запиту, а значення для POST, PUT або PATCH HTTP-методів дістаються з тіла запиту. Найбільш поширеними є наступні атрибути:

– [FromQuery] – атрибут, який сигналізує, що дані слід брати з переліку параметрів запиту;

– [FromRoute] – атрибут, який сигналізує, що дані слід брати з частини шляху запиту (при формуванні шляху можливо додати секцію в фігурних дужках "{}", яка визначить, що це значення для параметру методу);

– [FromBody] – атрибут, який сигналізує, що дані слід брати з тіла запиту, яке найчастіше передається в форматі JSON;

– [FromForm] – атрибут, який сигналізує, що дані слід брати з тіла запиту, якщо тіло передається у вигляді multipart/form-data, найчастіше це використовується при реалізації отримання даних від HTML-форм;

– [FromHeader] – атрибут, який сигналізує, що дані слід брати з розділу заголовків запиту.

В даному проекті реалізовані три контролера для роботи з трьома відповідними сервісами бізнес логіки, кожен з них передбачає свій набір кінцевих точок та їх шляхів.

Контролер StudentController реалізований для роботи з сервісом IStudentService та отримує його через механізм впровадження залежностей, який описаний в розділі 4.1.8. Базовим шляхом є розділ "students". Клас визначає кінцеву точку реалізовану за допомогою методу GetStudentByCode(string code), який відповідає GET-методу та приймає лише один параметр через шлях, а саме код студента, тому повний шлях даної кінцевої точки буде наступним: /students/{code}.

Контролер LessonController реалізований для роботи з сервісом IStudentService, який він отримує також через DI. Базовим шляхом є розділ "lessons". Клас визначає кінцеву точку реалізовану за допомогою методу GetLessonsByStudentId(int studentId, DateTime date, WeekDay dayOfWeek), який відповідає GET-методу та приймає три параметри: ідентифікатор студента (через шлях), дату занять (через параметр запиту) та номер дня тижня (також через параметр запиту). Повний шлях даної кінцевої точки буде наступним: /lessons{studentId}?date=[дата]&dayOfWeek=[день тижня].

Контролер NewsController реалізований для роботи з сервісом INewsService, який як і в інших контролерах отримується через впровадження залежностей. Базовим шляхом є розділ "news". Клас визначає чотири кінцевих точки:

1) GetNewsByStudentId(int studentId) - відповідає GET-методу та приймає лише один параметр через шлях, а саме ідентифікатор студента, та повертає список новин для цього студента. Повний шлях даної кінцевої точки буде наступним: /news/all/{studentId};

2) GetGroupNewsById(int groupNewsId) - відповідає GET-методу та приймає лише один параметр через шлях, а саме ідентифікатор групової

новини, та повертає зміст цієї новини. Повний шлях даної кінцевої точки буде наступним: /news/group/{groupNewsId};

3) GetStudentNewsById(int studentNewsId) - відповідає GET-методу та приймає лише один параметр через шлях, а саме ідентифікатор новини студента, та повертає зміст цієї новини. Повний шлях даної кінцевої точки буде наступним: /news/student/{studentNewsId};

4) GetStudentNewsById(int studentNewsId) - відповідає PUT-методу та приймає лише один параметр через шлях, а саме ідентифікатор новини студента, та змінює стан новини на прочитану, а також повертає змінений об’єкт новини без змісту. Повний шлях даної кінцевої точки буде наступним: /news/student/{studentNewsId}.

Так як в ситуаціях коли студента, по його коду, або конкретну новину, по її ідентифікатору, не знайдено репозиторії повертають, через сервіси, в контролери значення null, а за замовчуванням в контролерах присутня логіка, що встановлює відповідний код HTTP-статусу в залежності від змісту тіла відповіді, то в подібних випадках, для пустого тіла, клієнту передається код HTTP-статусу 204, а саме No Content, хоча мало б повертатися повідомлення, що об’єкт не знайдено (404 Not Found). Для вирішення цієї ситуації був реалізований спеціальний фільтр NotFoundResultFilterAttribute (фільтри – спеціальні класи, що вбудовуються в певні сегменти конвеєру обробки запиту та виконують певну логіку). NotFoundResultFilterAttribute являє собою аттрибут фільтру результату (Result Filter) – це фільтри які виконуються після всіх інших фільтрів та виконуються тільки тоді, коли виконання методу, що відповідає за кінцеву точку, завершено успішно. Такі фільтри можливо застосовувати для всіх контролерів одразу, для одного конкретного контролеру та для конкретної кінцевої точки. Їх зазвичай використовують для післяобробки результату запиту, наприклад, форматування або додавання користувацьких заголовків запиту. Він перевизначає один метод OnResultExecuting(ResultExecutingContext context) та в його тілі описує логіку, що перевіряє чи була відповідь з тілом і, якщо це вірне твердження, то

виконується додаткова перевірка на те, чи це значення дорівнює null, якщо обидві умови задовольняються – HTTP-статусі відповіді змінюється на 404 Not Found.

Для налаштування всього проекту прикладного програмного інтерфейсу на основі ASP .NET Core Web API використовується поєднання двох класів: Program – основний виконуваний клас, з якого починається робота API та Startup – клас, що відповідає налаштування всієї системи. Код для них генерується автоматично, але для додаткового користувацького налаштування цей код дозволяється модифікувати.

Клас Program в свою чергу складається з двох методів: Main(string[] args) – точка входу в програму (цей метод виконується в першу чергу, ініціалізує, виконує налаштування та запускає програмний інтерфейс) та CreateHostBuilder(string[] args) – метод, що викликається в середині методу Main і за допомогою вбудованих методів та класу Startup виконує конфігурування системи.

Клас Startup, як було описано вищу, призначений для налагодження всіх елементів, що присутні в проекті. Він складається, також, з двох методів: ConfigureServices(IServiceCollection services) та Configure(IApplicationBuilder app, IWebHostEnvironment env). Їх початковий код генерується автоматично та складається з виклику базових вбудованих методів, а саме:

– AddControllers() – єдиний метод, що викликається в методі ConfigureServices, а не в Configure та відповідає за інформування конвеєру обробки запитів про те, що в цьому проекті будуть використовуватися контролери;

– UseRouting () – викликається для ініціалізації конвеєру обробки запитів та повинен знаходитися перед викликом методу UseEndpoints;

– UseEndpoints(Action<IEndpointRouteBuilder> configure) – викликається для налаштування маршрутизації, в нашому випадку, всередині

нього викликається метод MapControllers(), що додає в конвеєр всі кінцеві точки без вказання шляху;

– UseAuthorization() – викликається для увімкнення базової логіки авторизації(без використання додаткових фільтрів та методів налаштування – не впливає на роботу системи).

Так як описані вище налаштування використовуються лише для того, щоб весь API запрацював в базовому вигляді, то для реалізації додаткової логіки Onion-архітектури, зручної роботи з клієнтом та створенні інтерактивної документації необхідно викликати ще декілька методів:

– AddCors() – метод, що викликається в ConfigureServices, для підключення механізму CORS (Cross-Origin Resource Sharing, спільне використання ресурсів з різних джерел – механізм безпеки браузерів, який дозволяє веб-сторінкам використовувати дані, що знаходяться на інших доменах), без виклику цього методу клієнти з інших доменів не змогли б виконувати запити на реалізований API;

– AddControllers(options=>options.Filters.Add(Type filterType)) – метод, що викликається в ConfigureServices та є перевантаженням стандартного AddControllers(), та використовується для додавання фільтрів до всіх контролерів відразу, в нашому випадку, використовується для додавання фільтру NotFoundResultFilterAttribute, що описаний вище;

– AddTransient<TService, TImplementation>() – метод, що викликається в ConfigureServices, для забезпечення функціонування механізму впровадження залежностей (DI), що використовується для співставлення певної абстракції (інтерфейсу) репозиторію або сервісу з реалізуючим його класом;

– AddSwaggerGen(Action<SwaggerGenOptions> setupAction) – метод, що викликається в ConfigureServices, який використовується для створення інтерактивної документації API на основі OpenAPI;

– UseCors(Action<CorsPolicyBuilder> configurePolicy) – метод, що викликається в Configure, для налаштування механізму CORS, в ньому вказуються параметри для налаштування дозволів, що будуть вказуватися в відповідному заголовку відповіді;

– UseSwagger() та UseSwaggerUI(Action<SwaggerUIOptions> setupAction) – це два методи, що викликаються в Configure разом, та відповідають ввімкнення інтерактивної документації OpenAPI та створенні веб-сторінки з нею за певними параметрами, що вказуються в параметрі setupAction.

4.2 Реалізація клієнтської частини архітектурного рішення

4.2.1 Набір технології для розробки клієнтської частини

Клієнтська частина даного архітектурного рішення була розроблена з використанням динамічної прототипної мови програмування JavaScript (JS), мови розмітки HTML та каскадної таблиці стилів CSS, з використанням додаткових бібліотек та інструментів, таких як jQuery та Bootstrap. Ця комбінація засобів дозволяє реалізовувати веб-додатки з декількома веб-сторінками без особливих зусиль. Перевагами даного підходу є:

– зручність в створенні HTML-розмітки за рахунок повторного використання стилів доступних в CSS та наявність вже готових елементів керування;

– швидкий та зрозумілий процес доступу до елементів сторінки внаслідок роботи з об'єктною моделлю документа (Document Object Model, DOM) через вбудовані функції jQuery;

– досить простий механізм спілкування з API з допомогою AJAX-запитів, що доступні в jQuery.

Недоліком є той факт, що робота з додатком виконується на декількох сторінках, на відміну від, наприклад, веб-сторінок на основі Angular, і переходи між ними це додаткове отримання HTML-коду з хостингу, але

беручи до уваги те, що кількість сторінок клієнтської частини досить мала, то реалізація за допомогою jQuery потребує досить мало витрат часу, тому в даному випадку такий підхід повністю задовольняє потреби клієнтської частини.

4.2.2 Опис головного вікна (index.html, script.js)

HTML-розмітка вікна з яким клієнт працює з самого початку реалізована в вигляді вікна з заголовком та підказкою про очікувані дії. Вона розподілена на дві частини: тег <head> – призначений для зберігання інших елементів, що допомагають браузеру в обробці сторінки та тег <body> – призначений для зберігання змісту веб-сторінки.

В області тегу <head> заходяться наступні елементи:

– тег <title> з назвою поточною сторінки, а саме "Студентський розклад";

– набір метатегів (<meta>) призначених для зберігання інформації виключно для браузера та пошукових систем, що складаються з імені тегу в атрибуті name, та його змісту в content;

– набір тегів <link>, що зв’язують поточний HTML-документ з зовнішніми документами (наприклад, файлами шрифтів та/або стилів), в нашому випадку виконується зв’язок з файлами стилів, що знаходяться у каталозі styles.

В свою чергу в області тегу <body> заходиться теги, що відповідають за відображення та перелік підключених скриптів:

– заголовок зображуваної сторінки виконаний з використанням тегу <nav>, що відповідає за навігацію по сайту, в нього найчастіше вміщують посилання на інші сторінки сайту або кнопки керування;

– всередині тегу <nav> знаходиться безпосередньо текст заголовку виконаний за допомогою поєднання тегів <a> (тег для створення посилань) та <p> (тег для форматування, а саме – створення абзаців), також при натисканні

на заголовок, за рахунок наявності посилання в атрибуті href тегу <a>, виконується перехід на головну сторінку;

– в випадках, коли підключення до мережі Інтернет досить повільне користувач може очікувати відповідь від серверу довгий час і якщо його не проінформувати про те, що робота виконується він може сприйняти це як збій та завершити роботу з додатком без отримання задовільного результату, тому в заголовку сторінки передбачений індикатор роботи сторінки (лоадер), він реалізований за допомогою тегу (тег блочної розмітки, призначений для виділення окремого фрагменту документу), який має атрибут class зі значенням preloader, що відповідає анімованому елементу сигналізуючому про виконання дій;

– весь основний зміст сторінки розміщений всередині тегу <main>, який відповідає за вміст основної частини документу (за виключенням елементів навігації, пошуку, тощо), та являє собою звичайну комбінацію тексту в вигляді заголовку другого рівня (тег <h2>) та схематичного зображення студентського реалізованого за допомогою тегу <img>, де в атрибуті src вказується шлях до зображення;

– окрім блоків навігації та основного змісту в тілі документу знаходяться описи всіх скриптів (тег <script> – призначений для опису скриптів безпосередньо в HTML-документі або підключенні зовнішніх файлів), що підключені до даної веб-сторінки, три з них – це використовувані бібліотеки jquery.js, popper.js, та bootstrap.js, а останній script.js – це файл JS-логіки.

Безпосередня логіка головної сторінки описана в файлі script.js, що підключається методом описаним вище, та вміщає в себе дві функції: функцію для обробки дії сканування студентського білету (вона виконується одразу після повного завантаження сторінки та побудови DOM-дерева) та загальну функцію для отримання значення параметру з посилання на поточній HTML-документ (ця функція використовується на всіх сторінках).

Алгоритм першої функції складається з наступних кроків:

1) ініціалізація змінної barcode, що відповідає за вміст штрих-коду на студентському;

2) формування блоку, що буде виконуватися при натисненні будь якої клавіші – це реалізовано таким чином, тому що майже всі сканери штрих-кодів працюють як клавіатура та емулюють натиснення клавіш;

3) в середині блоку описана логіка обробки натиснення клавіші. При умові що код натисненної клавіші не дорівнює 13 (клавіша Enter) – функція записує значення в змінну barcode. В випадку коду 13 (при закінченні сканування сканер завжди виконує "натиснення" клавіші Enter) виконується запис коду студента в локальне сховище (Local storage) браузеру та запит на сервер API для отримання інформації про студента. Запит відправляється на StudentController, а конкретніше – на метод кінцевої точки GetStudentByCode(string code), що розміщений за адресою students/{studentCode}. Відправка реалізована за допомогою функції $.ajax(), параметрами якої слугують:

– url – рядок в якому розміщена стандартизована адреса певного ресурсу (URL);

– type – тип HTTP-методу (в нашому випадку це GET);

– success – дії, що виконуються при успішному виконанні запиту (в нашому випадку це функція, що записує імя, кількість непрочитаних новин та ідентифікатор студента в локальне сховище та робить перехід на наступну сторінку);

– error – дії, що виконуються в випадку помилки при запиті (тут реалізована функція, що виконує перенаправлення користувача на сторінку з описом помилки).

4) під кінець виконання алгоритму, при будь якому результаті, користувача буде переведено на іншу сторінку.

Алгоритм загальної функції для отримання значення параметру з посилання поточної сторінки складається з наступних кроків:

1) отримання всього рядку параметрів за допомогою властивості window.location.search та отримання його без символу "?" (так як він завжди є першим) через виклик функції substring(1);

2) розділення отриманого рядку з параметрами по символу "&" за допомогою функції split('&');

3) перебір всього отриманого масиву параметрів за допомогою циклу for;

4) під час перебору, в тілі циклу, виконується отримання значення параметру запиту шляхом розділення поточного рядку через символ "=" і, якщо ім’я параметру збігається з параметром функції, виконується повернення результату функції.

4.2.3 Опис вікна з розкладом студента (scheduler.html, getSchedule.js)

Після сканування студентського та отримання інформації про нього клієнт переходить на сторінку з відображенням розкладу (scheduler.html). Вона, як і всі інші, складається з тегів <head> та <body>. Тег <head> включає в себе такі ж самі елементи як і на інших сторінках, а так як це було описано для сторінки index.html (розділ 4.2.2), то потреби описувати це не потрібно.

В області тегу <body> заходиться теги, що відповідають за відображення розкладу та скрипти:

– заголовок зображуваної сторінки виконаний з використанням тегу <nav> з логікою подібною до головної сторінки, але з деякими модифікаціями:

– текст заголовку в якому відображається ПІБ студента, яке отримано в попередньому вікні виконаний за допомогою тегу <p>;

– індикатор роботи сторінки реалізований за допомогою тегу , як і на попередній сторінці;

– кнопка керування "Завершити сеанс", реалізована за допомогою тегу <button>, що відповідає за завершення користувачем роботи з додатком та перехід на початкове вікно;

– кнопка керування "Новини", також реалізована за допомогою тегу <button>, що виконує перехід на вікно списку новин та складається з тексту написаного курсивом (за допомогою тегу <i>) і лічильника, що відображає кількість непрочитаних новин (з використанням тегу).

– весь основний зміст сторінки йде після тегу після тегу <nav> та складається з наступних елементів:

– список днів тижня для отримання розкладу за конкретний день, описаний в середині другого тегу <nav> за допомогою елементів-тегів <a> з спеціальними значеннями атрибуту date-data, посиланням на перезавантаження поточної сторінки та назвою конкретного дня тижня;

– таблиця для виведення розкладу яка реалізована за допомогою тегу <table> з переліком чітко детермінованих колонок;

– текст, про те, що занять не знайдено, для випадку отримання пустої відповіді на запит переліку занять, який за замовчуванням прихований;

– в тілі документу також знаходяться описи всіх скриптів, як і в інших документах тут присутні стандартні посилання на файли jquery.js, popper.js, та bootstrap.js, а також підключення script.js (описаного в попередньому розділі) та getSchedule.js – файлу логіки поточного вікна.

Логіка сторінки описана в файлі getSchedule.js, та включає в себе декілька функцій:

1) showLoader() – функція відображення лоадеру, що приховує текст з ПІБ студента (функція hide()) та відображує елемент з ідентифікатором "loader" (функція show());

2) hideLoader() – функція приховування лоадеру. Вона виконує ті ж самі дії, що і showLoader(), але навпаки;

3) функція, що спрацьовує при натисненні на кнопку "Завершити сеанс" та виконує перехід на початкову сторінку через встановлення значення "./index.html" властивості location.href. Вона реалізована через функцію $('body').on('click', '.exit', function);

4) getSchedule(dayOfWeek, date) – функція, що виконує запит на кінцеву точку серверу GetLessonsByStudentId(int studentId, DateTime date, WeekDay dayOfWeek), яка знаходиться в класі LessonController, за адресою "lessons/{studentId}?date=date&dayOfWeek=dayOfWeek". При помилці під час запиту – клієнта переводить на вікно з помилкою, а при успішній відповіді – викликається функція showShedule(data);

5) showShedule(scheduleObjs) – функція для запису отриманих занять в таблицю. Якщо масив занять пустий – виконується відображення тексту про їх відсутність, а в ситуації наявності хоча б одного заняття виконується перебір всіх занять, перетворення їх в HTML-розмітку з дотриманням спеціального форматування та додання цього коду до елементу з класом scheduleBody, що реалізований за рахунок тегу <tbody>;

6) при обранні дня тижня виконується перезавантаження сторінки і в цей момент виконується функція, що знаходиться на початку файлу – спочатку вона отримує поточний день тижня та за допомогою спеціального алгоритму визначає дату обраного дня тижня, виконує підсвічування обраної кнопки, затемнення інших та виклик функції getSchedule(selectedDay, selectedDate), де selectedDay – це індекс, а selectedDate – дата обраного дня.

4.2.4 Опис вікна зі списком новин студента (news.html, getNews.js)

При натисненні на кнопку "Новини" в вікні розкладу виконується перехід на сторінку з відображенням списку всіх новин студента, що працює з додатком (news.html). Опис тегу <head> ідентичний до попередніх сторінок.

В області тегу <body> заходиться наступні елементи:

– панель заголовку з розміткою подібною до вікна розкладу, але кнопка "Завершити сеанс" змінена на кнопку "Назад" (перехід на попереднє вікно), відсутні ПІБ студента та кнопка "Новини" (в ній немає необхідності, так як ми вже знаходимося на потрібному вікні);

– після тегу <nav> знаходиться таблиця, яка працює по тому ж принципу як на попередньому вікні, але з відмінним набором колонок;

– також присутній текст про відсутність новин, реалізований так само як текст про відсутність занять в документі scheduler.html;

– окрім підключень стандартних скриптів наявне підключення логіки цієї сторінки за рахунок файлу getNews.js.

Логіка сторінки описана в файлі getNews.js, та включає в себе декілька функцій:

1) showLoader() та hideLoader() – функції роботи з лоадером, з логікою відображення/приховування елементів як в файлі getSchedule.js;

2) функція, що спрацьовує при натисненні на кнопку "Назад" та виконує перехід на попередню сторінку через встановлення значення "./scheduler.html" властивості location.href. Вона реалізована через функцію $('body').on('click', '.exit', function), подібно до логіки в коді попереднього вікна;

3) showNews(studentId) – функція, яка викликається при завантаженні сторінки, що виконує запит на кінцеву точку серверу GetNewsByStudentId(int studentId), яка знаходиться в класі NewsController, за адресою "news/all/{studentId}". Параметр studentId отримується з параметрів функції. При помилці під час запиту – клієнта переводить на вікно з помилкою. В свою чергу при успішній відповіді виконується заповнення тіла таблиці в

документі news.html за методом заповнення таблиці з розкладом, але з відмінним способом форматування HTML-коду. При наявності пустого списку новин користувачу виводиться відповідне повідомлення за рахунок відображення елементу з класом no-news функцією show().

4.2.5 Опис вікна зі змістом новини студента (newsContent.html, getNewsPage.js)

При натисненні на елемент таблиці в списку новин в відповідному вікні виконується перехід на сторінку з відображенням змісту конкретної новини студента. Опис тегу <head> ідентичний до попередніх сторінок.

В області тегу <body> заходиться наступні елементи:

– панель заголовку ідентична до панелі в документі news.html;

– основна інформація про новину знаходиться в наборі вкладених тегів , що мають атрибутами різні класи;

– в одному із тегів , що знаходиться на останньому рівні вкладеності присутній текст заголовку зі значенням "Новина";

– після елементу заголовку знаходиться секція, що відповідає за зміст новини та включає в себе назву новини (тег <h5> з класом news-name), безпосередньо зміст новини (тег <p> з класом news-body) та кнопка керування "Взяв до уваги" (тег <button> з класом news-read-btn), яка призначена для відмітки факту того, що студент ознайомився з новиною;

– логіки цієї сторінки підключена за рахунок файлу getNewsPage.js разом зі стандартними скриптами.

Логіка сторінки описана в файлі getNews.js, та включає в себе декілька функцій:

1) showLoader() та hideLoader() – функції роботи з лоадером, з логікою відображення/приховування елементів як в файлах getSchedule.js та getNews.js;

2) функція, що спрацьовує при натисненні на кнопку "Назад" та виконує перехід на попередню сторінку через встановлення значення " ./news.html " властивості location.href. Вона реалізована через функцію $('body').on('click', '.exit', function), подібно до логіки в коді попереднього вікна;

3) showNews(newsId, isGroupNews) – функція, яка викликається при завантаженні сторінки, що виконує запит на одну з двох кінцевих точок серверу GetGroupNewsById(int groupNewsId) або GetStudentNewsById(int studentNewsId) в залежності від значення параметру isGroupNews, обидві кінцеві точки знаходиться в класі NewsController, за адресами "news/group/{groupNewsId}" та "news/student/{studentNewsId}" відповідно. Параметр newsId отримується з URL-параметру запиту поточної сторінки за допомогою функції getUrlParameter("id"), що описана в файлі script.js, та відповідає за ідентифікатор новини. При помилці під час запиту – клієнта, як і в випадках інших запитів, переводить на вікно з помилкою. В свою чергу при успішній відповіді виконується заповнення значень тегів з класами news-name та news-body значеннями data.header та data.content, де data – обєкт відповіді на запит, відповідно (за допомогою функції text(text)). Також при умові, що новина не є доступною для групи, а призначення тільки для відповідного студента виконується відображення кнопки "Взяв до уваги" і заповнення значення атрибуту data-news ідентифікатором новини data.id;

4) функція, яка викликається при натисненні на кнопку "Взяв до уваги" виконує запит з методом PUT на кінцеву точку серверу ChangeIsReadState(int studentNewsId) що знаходиться в класі NewsController, за адресою "news/student/{studentNewsId}" для зміни стану новини. При помилці під час запиту – перехід на вікно з помилкою. При успішному запиті – дана кнопка стає неактивною та міняє свій текст на "Прочитано" за рахунок виклику функції addClass("disabled") та text("Прочитано") відповідно.

4.2.5 Опис вікна помилок (error.html, notFound.js)

Якщо при будь якому запиті на API виникає помилка, тоді клієнт не може продовжувати роботу з додатком через певні причини(відсутність підключення до серверу, помилки або технічні роботи на сервері, не правильно введені або не існуючі дані, порушення протоколів запитів, тощо). В цьому випадку користувача переводить на сторінку, де є опис помилки (error.html), та можливі варіанти наступних дій. Опис тегу <head> ідентичний до попередніх сторінок.

В області тегу <body> заходиться наступні елементи:

– панель заголовку подібна до заголовку на сторінці scheduler.html, але без елементу з ПІБ студента (так як він може бути відсутній) та кнопки з переходом у вікно новин (так як ідентифікатор та кількість новин, також, можуть бути відсутніми);

– весь основний зміст сторінки розміщений всередині тегу <main> та являє собою перелік наступних елементів:

– зображення поламаного роботу, що сигналізує про помилку (тег <img>, джерело "img/robot-error.png");

– текст помилки, що завантажується через скрипт (тег <p>, клас "h1 errorText");

– текст підказки, щодо повторних дій (тег <p>, клас "h5 errorDescription");

– кнопка для повернення на головний екран (тег <button> з текстом "На головну");

– логіка цієї сторінки підключена за рахунок файлу notFound.js разом зі стандартними скриптами.

Логіка сторінки описана в файлі notFound.js, та включає в себе два сегменти коду: функція для повернення на головну сторінку при натисканні на відповідну кнопку (реалізовано подібно до такої ж функції в файлі getSchedule.js) та код для завантаження опису помилки в залежності від її типу.

При виконанні скрипта тип помилки отримується з URL-параметру запиту поточної сторінки за допомогою функції getUrlParameter("error "), що описана в файлі script.js, а потім за допомогою конструкції switch (error) в залежності від значення змінної error виконується встановлення заздалегіть продуманого тексту для елементів з класами errorText та errorDescription.

4.3 Дослідження ефективності реалізованого архітектурного рішення інформаційної підтримки студентів

Реалізоване архітектурне рішення було протестоване на основі даних, що вміщають в себе інформацію для 10 тисяч студентів. Також були змодельовані записи БД та запити до серверу двох інших підходів, а також виконане порівняння с результатами тестування розглянутого архітектурного рішення.

В реалізованому архітектурному рішенні використаний комбінований підхід до зберігання даних, а саме додавання занять та новин для всієї групи студентів одразу (в випадку якщо заняття та новини спільні), або ж додавання їх для окремого студента.

В якості розглянутих аналогів були розглянуті варіанти зберігання даних для всієї групи загалом в першому випадку, а також для кожного студента окремо в другому випадку.

В таблиці наведеній нижче представлені порівняльні результати за трьома критеріями: середній розмір БД в мегабайтах, середня тривалість запитів необхідних для отримання інформації, середня кількість рядків в БД для групи студентів на один навчальний тиждень.

Таблиця 4.1 – Порівняльні результати тестування

Розглядуваний підхід	Середній розмір БД (МБ)	Середня тривалість запитів (мс)	Середня кількість рядків для групи студентів на один навчальний тиждень
Комбінований підхід з заповненням груп та студентів	550	380	136
Заповнення лише груп	300	200	20
Заповнення лише студентів	900	1100	600

З даного порівняння ми можемо спостерігати те, що підхід з заповненням занять одразу для всієї групи виграє у всіх показниках у двох інших варіантів, але від не підходить по одній найголовнішій причині – такий підхід не забезпечує доступ до інформації про заняття за індивідуальним вибором, а отже перевагу варто віддавати не лише швидкодії, а й функціоналу, що дозволяє конкретний підхід.

4.4 Висновки по розділу

В даному розділі розглянуті реалізації програмних модулів серверної та клієнтської частини спроектованого архітектурного рішення, використовувані підходи та основні принципи розробки подібних додатків.

З боку серверної частини були реалізовані моделі, та взаємозв’язки між ними спроектовані в розділі 3; розглянуті сторонні бібліотеки для спрощення процесу розробки; підвищена майстерність в використанні принципів SOLID та IoC/DI; закріплені навички в роботі з базою та реалізації API.

З боку клієнтської частини були описані основні її елементи, в тому числі найбільш поширені теги та способи їх використання; використані додаткові інструменти/бібліотеки, що спрощують написання програмного коду та зменшують затрати ресурсів на реалізацію; побудований механізм

передачі даних між сторінками за допомогою локального сховища, що в свою чергу зменшило кількість непотрібного коду та зменшило час на реалізацію та тестування.

РОЗДІЛ 5
РОЗРОБКА СТАРТАП-ПРОЕКТУ

5.1 Назва проекту

Система інформаційної підтримки студентів

5.2 Короткий опис проекту

Система інформаційної підтримки студентів являє собою комплекс з одного або декількох терміналів(спеціально призначений персональний комп’ютер, або термінал на основі операційної системи Android, що розміщений в місці, через яке проходять більшість студентів, наприклад, в холі корпусу університету) та серверу (що відповідає за зберігання та обробку інформації). Термінал призначений для отримання студентом-клієнтом інформації про його розклад та доступні йому новини, шляхом сканування студентського білету за допомогою периферійного сканеру штрих-кодів. Додаванням та оновленням інформації на сервер займається відповідальний працівник адміністрації за допомогою доступних інструментів після проходження інструктажу.

5.3 Бізнес-модель

5.3.1 Цінний продукт

Цінними якостями (фактори створення цінності) системи інформаційної підтримки студентів відмінними від існуючих є:

– орієнтування на роботу з конкретним студентом;

– уніфікований формат обміну даними, що дозволить використовувати систему та її дані для інших реалізацій;

– низькі витрати на технічне обслуговування;

– низькі вимоги до технічного забезпечення;

– висока швидкість та надійність роботи;

– досить велика гнучкість та масштабованість;
– зручний дизайн.

5.3.2 Сегмент споживачів

Так як дана система має досить вузьку спеціалізацію, то основними споживачами будуть студенти ВУЗу та працівники, що виконують додавання/оновлення інформації. Також в майбутньому є можливість додати додатковий функціонал для викладачів університету.

5.3.3 Канали збуту

Каналами збуту є:

– продажі з офіційного сайту стартап-проекту;
– продажі за допомогою переговорів з представниками університетів;
– потенційний продаж за допомогою державних тендерів.

5.3.4 Взаємодія з споживачами

Підтримка користувачів здійснюється через:

– гарячу лінію;
– офіційні канали в месенджерах Telegram, Viber, WhatsApp;
– офіційну електронну пошту.

5.3.5 Дохід (монетизація)

Дохід буде отримуватися за допомогою прямих та посередницьких продажів ліцензій системи через канали збуту описані в розділі 5.3.3. Також можливий продаж додаткових модулів та оновлень системи, а також надання послуг для закупівлі апаратного забезпечення та проведення інструктажів персоналу.

5.3.6 Ключові види діяльності

Головним видом діяльності стартапу буде реалізація системи інформаційної підтримки студентів на основі терміналів та серверу, для цього була проведена:

– розробницька діяльність – закінчення розробки серверної та клієнтської частини;

– наукова діяльність – продовження дослідно-експериментальних робіт щодо розширення функціоналу системи в цілому, оптимізації серверної частини та покращення інтерфейсу сайту клієнтської частини;

– маркетингова діяльність - збільшення каналів донесення інформації до потенційних користувачів.

5.3.7 Ключові ресурси

Ключовими ресурсами системи є :

– програмні – функціонал серверної та клієнтської розроблятиметься силами кваліфікованих робітників в орендованому стартапом офісі;

– матеріальні – персональні компютери або ноутбуки, периферійні пристрої, офісне обладнання;

– інтелектуальні ресурси – будуть використані власні технічні розробки, платні та безкоштовні бібліотеки.

5.3.8 Людські ресурси

Людськими ресурсами даної системи є:

– директор – маркетолог із вищою освітою та досвідом роботи не менше 2 років;

– менеджер з продажу;

– менеджер проектів;

– Back-End розробник;

– Front-End розробник;
– тестувальник;
– дизайнер.

Всього: 7 працівників.

5.3.9 Витрати

Виготовлення та реалізація даної системи:

– повна собівартість реалізованого стартапу – 600 000 грн;
– прибуток стартапу – 3 000 000 грн;
– оптова ціна виробника – 600 000 грн.

Непрямі податки:

– податок на додану вартість – 120 000 грн;
– оптова ціна відпускна – 600 000 грн.

5.4 Споживчі властивості товару

– професійне спеціалізоване програмне забезпечення;
– надає дані в режимі реального часу;
– дозволить оптимізувати отримання розкладу студентами;
– досить простий процес додавання та оновлення інформації;
– зручний дизайн;
– велика масштабованість.

5.5 Дослідження ринку

Подібні рішення представлені внутрішніми системами на базах університетів реалізовані студентами цих ВУЗів спеціально для їх потреб. На українському ринку системи з подібним функціоналом відсутні.

5. 6 Дослідження конкурентного оточення

Так як конкурентів на ринку немає, а всі альтернативні рішення представлені внутрішніми системами університетів – необхідність в дослідженні конкурентного оточення на момент реалізації стартапу відпадає.

5.7 Маркетингова стратегія просування

Орієнтація на зручну та швидку систему інформаційної підтримки студентів та участь в державних тендерах для залучення потенційних клієнтів-університетів.

5.8 Елементи фінансового плану

5.8.1 Опис бізнес-проекту

Розробка системи інформаційної підтримки студентів з орієнтацією на роботу з конкретним студентом, його розкладом та новинами.

5.8.2 Опис товару/послуги/технології

Система інформаційної підтримки студентів – це програмний комплекс, який складається з серверу який зберігає інформацію та декількох терміналів-клієнтів, з якими працюють студенти. Основними функціями системи є надання студентам легкого та швидкого способу дізнатися власний розклад (включно з вибірковими дисциплінами) та новини за допомогою сканування студентського білету. Вся інформація заповнюється кваліфікованими робітниками адміністрації після проходження інструктажу та стає доступною в режимі реального часу.

5.8.3 Маркетинг та продаж

Дослідження ринку показало, що даний проект є першим на ринку.

Дослідження ринку показало, що даний проект є першим на ринку. Наразі університети користуються власними розробками їх студентів або працівників.

5.8.4 Фінансовий план

Витрати на організацію стартапу на період 3 місяців:

- закупівля офісного устаткування – 28 000 грн;
- закупівля технічного устаткування – 110 000 грн;
- створення робочої команди стартапу, прийом на роботу робітників, оплата заробітної плати – 500 000 грн;
- оренда приміщення – 30 000 грн;
- комунальні платежі – 8 000 грн;
- резерв, непередбачувані витрати – 50 000 грн.

5.8.5 Резюме

Система планується поширюватися за рахунок прямих продажів з офіційного сайту стартап-проекту, за допомогою переговорів з представниками університетів та участі в державних тендерах. На початку буде випущена бета-версія зі зниженою ціною для декількох цільових ВУЗів, а згодом будуть розроблятися рішення-додатки для створення екосистеми, що включає в себе мобільний додаток та бот в месенджерах.

5.9 Презентація проекту інвестору

5.9.1 Ідея проекту

Основна ідея системи полягає в орієнтуванні на роботу з конкретним студентом, а не з групами студентів, що забезпечить швидкий та зручний доступ клієнта до необхідної інформації.

5.9.2 Опис проблеми або можливості

На українському ринку не представлені комерційні аналоги системи, а всі подібні рішення розробляються силами студентів та працівників відповідних навчальних закладів, то є висока ймовірність в тому, що система матиме низький попит. Але з часом передбачене прийняття участі в державних тендерах, де система може добре себе зарекомендувати та набрати аудиторію потенційних клієнтів, що являють собою представників вищих навчальних закладів.

5.9.3 Рішення

Рішеннями описаної вище проблеми є постійні зустрічі з представниками університетів, участь в тендерах та активний розвиток системи, що в свою чергу збільшить доступний функціонал та підвищить зацікавленість клієнтів в інтегруванні системи в навчальний процес.

5.9.4 Конкуренти

На даний момент конкурентів на українському ринку не представлено, так як всі аналоги являють собою внутрішні розробки.

5.9.5 Ринок

Новий товар на новий ринок.

5.9.6 Маркетингова стратегія

Маркетингова стратегія складається через просування продукту за допомогою таких підходів:

- просування через сайт;
- просування через зустрічі з представниками навчальних закладів;
- просування через участь у державних тендерах;
- просування через прямі продажі;
- просування через участь у презентаціях та конференціях.

5.9.7 Поточна ситуація

На даний момент розробка початкової версії серверної та клієнтської частини завершена, виконується тестування системи та проектування нового функціоналу для розробки в майбутньому.

5.9.8 Команда проекту

Попередні домовленості про участь 3-х потенційних робітників, а саме: Front-End розробник, Back-end розробник, тестувальник.

5.9.9 Фінансові показники

Фінансові показники допрацьовуються. Попередньо за 2 місяці витрати складають 200 000 грн.

5.9.10 Пропозиція інвестору

Вкласти угоду на отримання 25% від продажу застосунку протягом восьми місяців та з подальшими переговорами про продовження співпраці.

5.10 Подальші кроки в проекті

5.10.1 Наукова діяльність

Продовження дослідницьких та експериментальних робіт для покращення функціоналу системи розробка нових функціональних

можливостей, аналіз наукових процесів університетів для більш точної відповідності потребам навчальних закладів.

5.10.2 Організаційна діяльність

Збільшення кількості розробників та залучення нових робітників відповідальних за організацію роботи та просування продукту.

5.10.3 Маркетингова діяльність

Збільшення каналів донесення інформації до потенційних користувачів.

Висновки по розділу

В даному розділі був поданий опис стартап-проекту "Система інформаційної підтримки студентів". Цей стартап допоможе навчальним закладам перенести фокус роботи зі здобувачами вищої освіти від груп до конкретних студентів, краще та в більш зручному форматі надавати інформацію щодо їх розкладу (разом з вибірковими дисциплінами) та новин. Заповнення даних відповідальними працівниками також передбачене швидким та зручним. Так як клієнтська частина (термінал) буде розміщено в холі університету – це залучить до взаємодії майже всіх студентів та у них завжди буде зручний розклад під рукою.

ВИСНОВКИ

На початку роботи над даним архітектурним рішенням був проведений аналіз існуючих аналогів, за допомогою яких створена концепція архітектурного рішення програмного забезпечення терміналу підтримки студентської діяльності та продумані основні його можливості.

Під час етапу проектування було продумане архітектурне рішення для програмного забезпечення, створені необхідні діаграми прецедентів, діяльності та ER-діаграма, спроектована трьохрівнева клієнт-серверна архітектура та діаграма класів, описані вимоги до серверної та клієнтської частини та обрані всі необхідні технології для реалізації цих частин системи.

На етапі реалізації проводилася розробка клієнта та сервера на основі результатів проектування архітектурного рішення. При розробці клієнта були покращені та закріплені навички в верстці веб-сторінок з допомогою HTML та Bootstrap, а також підвищена кваліфікація у області програмування на JavaScript з допомогою jQuery та використання підходу роботи з локальним сховищем.

Під час створення серверу були відточені навички в створенні серверів прикладного програмного інтерфейсу (API), впровадженні клієнт - серверної архітектури та роботи з базами даних. Підвищена кваліфікація у області розуміння та використання принципів SOLID та IoC/DI.

СПИСОК ПОСИЛАНЬ

1) Вуд Кид. Расширение библиотеки jQuery / В. Кид. – Москва: ДМК Пресс, 2014. – 400 с. – ISBN 978-5-97060-071-9.

2) Голуб Богдан Михайлович. C#. Концепція та синтаксис. / Б. М. Голуб., – Львів: Видавничий центр ЛНУ ім. Івана Франка, 2006. – 136 с.

3) Гоше Хуан Диего. HTML5. Для профессионалов. / Х. Д. Гоше. – Санкт-Петербург: Питер, 2013. – 496 с. – ISBN 978-5-496-00099-4

4) Клименко Роман. Веб-мастеринг на 100% / Клименко Р. – Санкт-Петербург: Питер, 2013. – 512 с. – ISBN 978-5-496-00079-6

5) Кузнецов Максим. MySQL 5 / М. В. Кузнецов, И. В. Симдянов. – Санкт-Петербург: БХВ-Петербург, 2010. – 1007 с. – ISBN 978-5-94157-928-0

6) Маклафлин Бретт. Изучаем AJAX / Б. Маклафлин. – Санкт-Петербург: Питер Пресс, 2008. – 424 с. – ISBN 978-5-91180-322-3

7) Мова програмування C#. Класика Computers Science. / А.Хейлсберг, М. Торгерсен, С. Вилтамут, П. Голд. – Санкт-Петербург: Питер, 2012. – 784 с. – ISBN 978-5-459-00283-6

8) Никольский А. П. JavaScript на примерах. Практика, практика и только практика / А. П. Никольский. - Санкт-Петербург: Наука и Техника, 2017. – 274 с. – ISBN 978-5-94387-735-3

9) Пилгрим Марк. Погружение в HTML5 / М. Пилгрим. – Санкт-Петербург: БХВ-Петербург, 2008. – 295 с. – ISBN 978-5-9775-0688-5

10) Пьюривал Семми. Основы разработки веб-приложений / С. Пьюривал – Санкт-Петербург: Питер, 2015. – 272 с. – ISBN 978-5-496-01226-3

11) Рихтер Джефри. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е вид. / Дж. Ріхтер – Санкт-Петербург: Питер, 2013 – 896 с. – ISBN 978-5-496-00433-6

12) Роббинс Дженнифер. HTML5, CSS3 и JavaScript. Исчерпывающее руководство / Дж. Н. Роббинс. – Москва: Эксмо, 2014. – 516 с.

13) Себеста В. Роберт. Основные концепции языков программирования. 5-е вид. / Р. В. Себеста. – Москва: Вильямс, 2001. – 672 с. – ISBN 5-8459-0192-8, 0-201-75295-6

14) Симпсон Кайл. ES6 и не только / К. Симпсон. – Санкт-Петербург: Питер, 2017. – 336 с. – ISBN 978-5-496-02445-7

15) Стиллмен Ендрю. Head First. Изучаем C#. 3-е издание / Е. Стиллмен, Д. Грін. – Санкт-Петербург: Питер, 2012. – 128 с. – ISBN 978-5-4461-1563-1, 978-1449343507

16) Фримен Эрик. Изучаем программирование на HTML5 / Э. Фримен, Э. Робсон. – Санкт-Петербург: Питер, 2011. – 594 с. – ISBN 978-5-459-00952-1

17) Херман Дэвид. Сила JavaScript. 68 способов эффективного использования JS / Д. Херман. – Санкт-Петербург: Питер, 2013. – 288 с. – ISBN 978-5-496-00524-1

18) ADO.NET Entity Framework [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/ADO.NET_Entity_Framework.

19) Data Provider for .NET Developer's Guide [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/database/121/ODPNT/intro003.htm#ODPNT131.

20) MySQL в ASP.NET MVC [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/sharp/articles/mvc/23.php.

ДОДАТОК А Публікації

1) Збірник наукових матеріалів XXIV Міжнародної інтернет-конференції «SCIENCE AND TECHNOLOGY». Реалізація методу зберігання даних в клієнт-серверній архітектурі для термінала підтримки студентської діяльності / Б. Кізюн., 2021. – 122 с.

2) Перша Всеукраїнська науково-практична конференція молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології» (SoftTech- 2021). Секція кафедри інформатики та програмної інженерії. Матеріали конференції. Підходи для аналізу та формування покращень системи інформаційної підтримки студентів. / Б. Кізюн., 2021. – 148 с.

ДОДАТОК Б Клас Startup.cs

```
public class Startup{
        public IConfiguration Configuration { get; }


        public Startup(IConfiguration configuration){
            this.Configuration = configuration;}
        public void ConfigureServices(IServiceCollection
    services){
            services.AddCors().AddControllers(options =>
    options.Filters.Add(typeof(NotFoundResultFilterAttribute)));
            services.AddTransient<IStudentRepository,
    StudentRepository>();
            services.AddTransient<ILessonRepository,
    LessonRepository>();
            services.AddTransient<INewsRepository,
    NewsRepository>();
            services.AddTransient<IStudentService,
    StudentService>();
            services.AddTransient<ILessonService,
    LessonService>();
            services.AddTransient<INewsService, NewsService>();
            services.AddSwaggerGen(options =>{
                options.SwaggerDoc("v1", new OpenApiInfo{
                    Version = "v1",
                    Title = "StudentsSchedule API",});});});}
        public void Configure(IApplicationBuilder app,
    IWebHostEnvironment env){
            if (env.IsDevelopment()){
                app.UseDeveloperExceptionPage();}
            app.UseRouting().UseCors(obj =>
    obj.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader());
            app.UseAuthorization();
            app.UseEndpoints(endpoints =>{
                endpoints.MapControllers();});
            app.UseSwagger();
            app.UseSwaggerUI(options =>{

    options.SwaggerEndpoint("/swagger/v1/swagger.json",
    "v1");});});}}
```

ДОДАТОК В Класи простору імен StudentsSchedule.Infrastructure.Data

```
public class StudentRepository : IStudentRepository{
        public Student GetStudentByCode(string code){
            using (UserContext context = new UserContext()){
                return context.Students
                    .Include(obj => obj.StudentNews)
                    .AsNoTracking()
                    .FirstOrDefault(obj => obj.Code == code);}}}


public class LessonRepository : ILessonRepository{
        public IEnumerable<Lesson> GetLessonsByStudentId(int
    studentId, DateTime date, WeekDay dayOfWeek){
            using (UserContext context = new UserContext()){
                IEnumerable<Lesson> studentLessons =
    context.StudentLessons
                    .Include(obj => obj.Classroom)
                    .Include(obj => obj.Teacher)
                    .AsNoTracking()
                    .Where(obj => obj.StudentId == studentId &&
                                    (obj.Date != null &&
    obj.Date.Value.Date == date.Date || obj.Date == null &&
    obj.DayOfWeek == dayOfWeek))
                    .ToList();
                IEnumerable<Lesson> groupLessons =
    context.GroupLessons
                    .Include(obj => obj.Classroom)
                    .Include(obj => obj.Teacher)
                    .Include(obj => obj.Group)
                        .ThenInclude(obj => obj.Students)
                    .AsNoTracking()
                    .Where(obj => obj.Group.Students.Select(obj1
    => obj1.Id).Contains(studentId) &&
                                    (obj.Date != null &&
    obj.Date.Value.Date == date.Date || obj.Date == null &&
    obj.DayOfWeek == dayOfWeek))
                    .ToList();
```

```
            return
    studentLessons.Concat(groupLessons).OrderBy(obj =>
    obj.Time).ToList();}}}


public class NewsRepository : INewsRepository{
        public IEnumerable<News> GetNewsByStudentId(int
    studentId){
            using (UserContext context = new UserContext()){
                IEnumerable<News> studentLessons =
    context.StudentNews
                    .AsNoTracking()
                    .Where(obj => obj.StudentId == studentId)
                    .ToList();
                IEnumerable<News> groupLessons =
    context.GroupNews
                    .Include(obj => obj.Group)
                        .ThenInclude(obj => obj.Students)
                    .AsNoTracking()
                    .Where(obj => obj.Group.Students.Select(obj1
    => obj1.Id).Contains(studentId))
                    .ToList();
                return
    studentLessons.Concat(groupLessons).ToList();}}


        public GroupNews GetGroupNewsById(int groupNewsId){
            using (UserContext context = new UserContext()){
                return context.GroupNews
                    .AsNoTracking()
                    .FirstOrDefault(obj => obj.Id ==
    groupNewsId);}}


        public StudentNews GetStudentNewsById(int
    studentNewsId){
            using (UserContext context = new UserContext()){
                return context.StudentNews
                    .AsNoTracking()
                    .FirstOrDefault(obj => obj.Id ==
    studentNewsId);}}
```

```
    public StudentNews ChangeIsReadState(int studentNewsId){
        using (UserContext context = new UserContext()){
            StudentNews foundStudentNews =
context.StudentNews
                .FirstOrDefault(obj => obj.Id ==
studentNewsId);
            if (foundStudentNews != null){
                foundStudentNews.IsRead = true;
                context.SaveChanges();}
            return foundStudentNews;}}}
```

ДОДАТОК Г Класи простору імен StudentsSchedule.Interface.Api.Controllers

```
[ApiController, Route("students")]
    public class StudentController : ControllerBase{
        public IStudentService StudentService { get; }

        public StudentController(IStudentService
    studentService){
            this.StudentService = studentService;}

        [HttpGet("{code}")]
        public GetStudentResponse GetStudentByCode(string code){
            return this.StudentService.GetStudentByCode(code);}}

[ApiController, Route("lessons")]
    public class LessonController : ControllerBase{
        public ILessonService LessonService { get; }

        public LessonController(ILessonService lessonService){
            this.LessonService = lessonService;}

        [HttpGet("{studentId}")]
        public IEnumerable<GetLessonResponse>
    GetLessonsByStudentId(int studentId, DateTime date, WeekDay
    dayOfWeek){
            return
    this.LessonService.GetLessonsByStudentId(studentId, date,
    dayOfWeek);}}

[ApiController, Route("news")]
    public class NewsController : ControllerBase{
        public INewsService NewsService { get; }

        public NewsController(INewsService newsService){
            this.NewsService = newsService;}
```

```
    [HttpGet("all/{studentId}")]
    public IEnumerable<GetNewsResponse>
GetNewsByStudentId(int studentId){
        return
this.NewsService.GetNewsByStudentId(studentId);}
    [HttpGet("group/{groupNewsId}")]
    public GetNewsContentResponse GetGroupNewsById(int
groupNewsId){
        return
this.NewsService.GetGroupNewsById(groupNewsId);}
    [HttpGet("student/{studentNewsId}")]
    public GetNewsContentResponse GetStudentNewsById(int
studentNewsId){
        return
this.NewsService.GetStudentNewsById(studentNewsId);}
    [HttpPut("student/{studentNewsId}")]
    public GetNewsResponse ChangeIsReadState(int
studentNewsId){
        return
this.NewsService.ChangeIsReadState(studentNewsId);}}
```

ДОДАТОК Д Розмітка вікна index.html

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-
    scale=1, shrink-to-fit=no">
    <title>Студенский розклад</title>

    <link rel="stylesheet" href="./styles/bootstrap.css">
    <link rel="stylesheet" href="./styles/style.css">
    <link rel="stylesheet" href="./styles/icons.css">
</head>

<body>
<div class="container-fluid p-0">

    <nav class="navbar navbar-expand-md navbar-dark bg-dark
    fixed-top d-flex justify-content-between">
        <div class="col-2">
            <button type="button" class="btn btn-outline-light
    float-right exit">
                <i class="fas fa-arrow-left"></i>
                Назад
            </button>
        </div>
        <div class="col-8 offset-1">
            <div class="preloader" id="loader" style="display:
    none;">
                <div></div>
                <div></div>
                <div></div>
                <div></div>
                <div></div>
```

```
                <div></div>
                <div></div>
                <div></div>
                <div></div>
                <div></div>
            </div>
        </div>
    </nav>

    <table class="table table-striped">
        <thead class="">
        <tr>
            <th class="col-1" scope="col">#</th>
            <th class="col-10" scope="col">Текст</th>
            <th class="col-1" scope="col"></th>
        </tr>
        </thead>
        <tbody class="scheduleBody">


        </tbody>
    </table>
    <div class="text-center no-news" style="display: none;">
        <h3>Новини відсутні</h3>
    </div>

</div>
<script src="scripts/jquery.js"></script>
<script src="./scripts/popper.js"></script>
<script src="./scripts/bootstrap.js"></script>
<script src="./scripts/script.js"></script>
<script src="./scripts/getNews.js"></script>
</body>

</html>
```

ДОДАТОК Е Розмітка вікна index.html

```
$(document).ready(function () {
    //Получение кода и передача дальше
    var barcode = "";
    $(document).keydown(function (e) {

        var code = (e.keyCode ? e.keyCode : e.which);
        if (code === 13) // Enter key hit{
            localStorage.studentCard = barcode;
            showLoader();
            $.ajax({
                url:  "http://176.107.180.155/students/"
+ barcode,
                type: 'GET',
                success: function (data) {
                    localStorage.studentName          =
data.name;
                    localStorage.unreadNewsCount      =
data.unreadNewsCount;
                    localStorage.studentId = data.id;
                    window.location.href              =
"./scheduler.html";},
                error: function (jqXHR) {
                    location.href                     =
'./error.html?errorCode=' + jqXHR.status;},
                timeout: 5000});
        } else {
            barcode          =          barcode          +
String.fromCharCode(code);}});
     $(".container").fadeIn();});

function showLoader() {
    $(".student-name").hide();
    $("#loader").show();}
```

```
    function hideLoader() {
        $(".student-name").show();
        $("#loader").hide();}

    //Получение параметров из GET запрса (ссылки текущей
страницы)
    function getUrlParameter(sParam) {
        let sPageURL = window.location.search.substring(1),
            sURLletiables = sPageURL.split('&'),
            sParameterName,
            i;

        for (i = 0; i < sURLletiables.length; i++) {
            sParameterName = sURLletiables[i].split('=');

            if (sParameterName[0] === sParam) {
                return sParameterName[1] === undefined ? true
: decodeURIComponent(sParameterName[1]);}}};
```

ДОДАТОК Ж Розмітка вікна scheduler.html

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-
    scale=1, shrink-to-fit=no">
    <title>Студенский розклад</title>

    <link rel="stylesheet" href="./styles/bootstrap.css">
    <link rel="stylesheet" href="./styles/style.css">
    <link rel="stylesheet" href="./styles/icons.css">
</head>

<body>
    <div class="container-fluid p-0">

    <nav class="navbar navbar-expand-md navbar-dark bg-dark
    fixed-top d-flex justify-content-between">
        <div class="">
            <button type="button" class="btn btn-outline-light
    float-right exit"><i class="fas fa-power-off"></i> Завершити
    сеанс</button>
        </div>
        <div class="">
         <p class="h3 student-name text-light" style="display:
    none;"></p>
            <div class="preloader" id="loader" style="display:
    none;">
                <div></div>
                <div></div>
                <div></div>
                <div></div>
                <div></div>
                <div></div>
```

```
            <div></div>
            <div></div>
            <div></div>
            <div></div>
        </div>
    </div>
    <div class="">
       <button type="button" class="btn btn-outline-light
float-right news"><i class="fas fa-newspaper" aria-
hidden="true"></i> Новини <span class="badge badge-light
newsCount">0</span> </button>
    </div>
</nav>


    <nav class="nav nav-pills nav-justified h5 text-align-
center">
        <a class="nav-item nav-link p-4 daySelect day-0"
date-data="0" href="#">понеділок</a>
        <a class="nav-item nav-link p-4 daySelect day-1"
date-data="1" href="#">вівторок</a>
        <a class="nav-item nav-link p-4 daySelect day-2"
date-data="2" href="#">середа</a>
        <a class="nav-item nav-link p-4 daySelect day-3"
date-data="3" href="#">четвер</a>
        <a class="nav-item nav-link p-4 daySelect day-4"
date-data="4" href="#">п’ятниця</a>
        <a class="nav-item nav-link p-4 daySelect day-5"
date-data="5" href="#">субота</a>
        <a class="nav-item nav-link p-4 daySelect day-6"
date-data="6" href="#">неділя</a>
    </nav>
    <table class="table table-striped">
        <thead class="">
            <tr>
                <th scope="col">#</th>
                <th scope="col">Дисципліна</th>
                <th scope="col">Час навчання</th>
                <th scope="col">Тип заняття</th>
                <th scope="col">Аудиторія</th>
                <th scope="col">Викладач</th>
```

```
                    <th scope="col">Підгруппа</th>
                </tr>
            </thead>
            <tbody class="scheduleBody">


            </tbody>
        </table>
     <div class="text-center no-lessons" style="display: none;">
     <h3 >В цей день заннять не найдено</h3>
     </div>

    </div>
    <script src="scripts/jquery.js"></script>
    <script src="./scripts/popper.js"></script>
    <script src="./scripts/bootstrap.js"></script>
    <script src="./scripts/script.js"></script>
    <script src="./scripts/getSchedule.js"></script>
</body>

</html>
```

ДОДАТОК И Файл скриптів getSchedule.js

```
$(function () {

    let date = new Date();
    let today = date.getDay() - 1;
    let selectedDate=new Date();
    if(today === -1){
     today = 6;}

    //Подсветим текущий день
    $('.day-' + today).addClass('active');
    //Выбор дня для просмотра расписания
    $('body').on('click', '.daySelect', function () {
     $('.daySelect').removeClass('active');
     $(this).closest('.daySelect').addClass('active');
     $(".lessonElement").remove();
     let selectedDay =
    parseInt($(".daySelect[class*='active']").attr("date-data"));
     //Разница в днях
     selectedDate.setDate(date.getDate()+(selectedDay - today));
     getSchedule(selectedDay, selectedDate);});
    getSchedule(today, date);
    $(".student-name").text(localStorage.studentName);
    $(".newsCount").text(localStorage.unreadNewsCount);});

 $('body').on('click', '.exit', function () {
      location.href = './index.html';});

function getSchedule(dayOfWeek, date) {
    showLoader();

    $('body').on('click', '.news', function () {
     window.location.href = "./news.html";});
```

```
 $.ajax({
      url: "http://176.107.180.155/lessons/" +
localStorage.studentId + "?date=" + date.toISOString() +
"&dayOfWeek=" + dayOfWeek,
      type: 'GET',
      success: function (data) {
           showShedule(data);},
      error: function (jqXHR) {
            location.href = './error.html?errorCode=' +
jqXHR.status;},});}

function showShedule(scheduleObjs) {
    $(".no-lessons").hide();

    let lessons = scheduleObjs.length;
      for (let i = 0; i < scheduleObjs.length; i++) {
           $("<tr class='lessonElement'><th scope='row'>" +
                (i + 1) + "</th><td>" +
                scheduleObjs[i].name + "</td><td>" +
                new
    Date(scheduleObjs[i].time).toLocaleTimeString() + "</td><td>"
    +scheduleObjs[i].type + "</td><td>"
    +scheduleObjs[i].classroom.name + "</td><td>"
    +scheduleObjs[i].teacher.name + "</td><td>"
    +(scheduleObjs[i].subgroup ?? "")
    +"</td></tr>").appendTo(".scheduleBody");}
    hideLoader();
    if (lessons === 0) {
     $(".no-lessons").show();}}
```

UNICHECK

Имя пользователя:
Лісовиченко Олег Іванович

ID проверки:
1009305675

Дата проверки:
23.11.2021 07:52:41 EET

Тип проверки:
Doc vs Internet + Library

Дата отчета:
23.11.2021 07:53:09 EET

ID пользователя:
76913

Название файла: **ІТ-з03мп_Кізюн_ПЗ**

Количество страниц: **53** Количество слов: **11133** Количество символов: **83662** Размер файла: **81.78 KB** ID файла: **1009331283**

1.29%
Совпадения

Наибольшее совпадение: **0.45%** с Интернет-источником (https://duan.edu.ua/science-ukr/015-profesiina-osvita-za-spets.

0.56% Источники из Интернета | 10 ..... Страница 55

0.84% Источники из Библиотеки | 103 ..... Страница 55

0% Цитат

Исключение цитат выключено

Исключение списка библиографических ссылок выключено

0% Исключений

Нет исключенных источников

Модификации

Обнаружены модификации текста. Подробная информация доступна в онлайн-отчете.

Замененные символы | 1