МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
Кафедра Комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ Аліна САВЧЕНКО
«____» _____ 2022р.

КВАЛІФІКАЦІЙНА РОБОТА
(ДИПЛОМНА РОБОТА, ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ **"МАГІСТР"**

ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ

"ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ"

Тема: "Web-застосунок «Щоденник» для онлайн навчання мовою Python"

Виконавець: студентка групи УС-211М Кулачинська Анжеліка Олександрівна

Керівник: к.т.н., доцент Савченко Аліна Станіславівна

Нормоконтролер: Ігор РАЙЧЕВ

Київ - 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *Кібербезпеки, комп'ютерної та програмної інженерії*

Кафедра *Комп'ютерних інформаційних технологій*

Галузь знань, спеціальність, освітньо-професійна програма: 12 «Інформаційні технології», 122 «Комп'ютерні науки», «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
Завідувач випускової кафедри
_____ Аліна САВЧЕНКО
«____»_____2022р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи студентки

Кулачинської Анжеліки Олександрівни
(прізвище, ім'я, по батькові)

1. Тема роботи: «Web-застосунок «Щоденник» для онлайн-навчання мовою Python» затверджена наказом ректора від «28» вересня 2022 р. за № 1774/ст.

2. Термін виконання роботи: 26.09.2022 – 21.11.2022

3. Вихідні дані до роботи: процеси створення власного навчального плану під час онлайн навчання.

4. Зміст пояснювальної записки: вступ, аналітичний огляд та постановка задачі, проектування web-застосунку, огляд інструментів та технологій та розробка web-додатку.

5. Перелік обов'язкового графічного матеріалу: діаграма баз даних, діаграма компонентів архітектури web-застосунку.

6. Календарний план-графік

№ п/п	Завдання	Термін виконання	Підпис керівника
1.	Отримання завдання на дипломну роботу та побудова плану-графіку виконання робіт.	26.09.2022 - 28.09.2022	
2.	Проведення консультації з науковим керівником.	29.09.2022 - 30.09.2022	
3.	Аналіз предметної області та області призначення.	03.10.2022 – 05.09.2022	
4.	Огляд існуючих аналогів та постановка задачі.	06.10.2022 – 11.10.2022	
5.	Створення першого розділу кваліфікаційної роботи.	11.10.2022 – 14.10.2022	
6.	Аналіз існуючих web-архітектур та вибір оптимальної.	17.10.2022 – 19.10.2022	
7.	Огляд інструментів та технологій для створення web-додатку.	19.10.2022 – 21.10.2022	
8.	Написання другого розділу кваліфікаційної роботи.	24.10.2021 – 27.10.2022	
9.	Розробка web-додатку.	28.10.2022 – 02.11.2022	
10.	Написання третього розділу кваліфікаційної роботи.	02.11.2022 – 05.11.2022	
11.	Створення презентації, доповіді та підготовка до захисту дипломної роботи.	05.11.2022 – 21.11.2022	

7. Дата видачі завдання: «26» вересня 2022 р.

Керівник дипломної роботи _____ Аліна САВЧЕНКО
(підпис керівника)

Завдання прийняв до виконання _____ Анжеліка КУЛАЧИНСЬКА
(підпис випускниці)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Web-застосунок «Щоденник» для онлайн навчання мовою Python» складається зі вступу, трьох розділів, висновку, списку бібліографічних посилань та двох додатків і містить 101 сторінку та 58 рисунків. Список бібліографічних посилань складається з 20 найменувань.

Ключові слова: WEB-ЗАСТОСУНОК, ДИСТАНЦІЙНЕ НАВЧАННЯ, WEB-ТЕХНОЛОГІЇ, ОНЛАЙН НАВЧАННЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ЩОДЕННИК, МОВА ПРОГРАМУВАННЯ PYTHON, ФРЕЙМВОРК DJANGO.

Актуальність. Сьогодні існує гостра потреба в застосунках для дистанційного навчання, так як через повномасштабну війну в Україні більшість студентів та учнів змушені були перейти на онлайн формат навчання. Web-застосунок – це один з найзручніших способів реалізації додатку для складання власного навчального розкладу.

Метою дипломної роботи є розробка web-застосунку «Щоденник» мовою Python для автоматизація процесу складання власного навчального плану.

Перевагами web-застосунку «Щоденник» є зручна взаємодія користувача з програмним продуктом та зведення до мінімуму зусиль користувача при складанні власного навчального плану.

Для досягнення заданої мети потрібно виконати наступні **задачі**:

- розробити структуру бази даних;
- створити зручний інтерфейс для доступу до бази даних та складання розкладу;
- забезпечити доступ до даних, що зберігаються в базі;
- забезпечити можливость зміни і доповнення даних, що зберігаються в базі;
- забезпечити пошук невідповідностей у вже створеному розкладі при зміні або доповненні даних, що зберігаються в базі;

- забезпечити відображення домашніх завдань при вході в окремий обліковий запис.

Об'єктом дослідження є складання навчального плану для онлайн навчання.

Предметом дослідження є автоматизація процесу складання власного навчального плану з використанням технології та інструментів створення web-застосунку мовою Python.

Методи дослідження включають в себе:

- методи побудови web-архітектури застосунку;
- методи створення бази даних;
- методи створення інтерфейсу web-застосунку.

Теоретичною основою дипломної роботи стали вітчизняні та зарубіжні дослідження щодо забезпечення якості програмного забезпечення та публікації насайтах, присвячені питанням створення web-додатків.

Теоретична і практична значимість роботи полягає в тому, що на основі отриманих знань:

1) можна в короткий термін ознайомитись з методами побудови web-архітектури, інструментами та технологіями створення web-застосунків;
2) створити методичні матеріали для студентів і включити його в програму навчання для курсу «Web-технології»;
3) розроблено готовий web-застосунок для створення власного навчального плану.

На захист виносяться наступні положення:

1) огляд інструментів та технологій створення web-додатків;
2) процес створення бази даних з використанням цих інструментів;
3) технології створення користувацького інтерфейсу;
4) процес дизайну та публікації дизайну на сервер.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

IADIS – International Association for the Development of Information Society

SPA – Single Page Application

PWA – Progressive Web Application

HTML – Hyper Text Markup Language

XML – Extensible Markup Language

JSON – JavaScript Object Notation

URL – Uniform Resource Locator

CGI – Common Gareway Interface

DOM – Document Object Model

CSR – Certificate Signing Request

PHP – Hypertext Preprocessor

БД – база даних

CRUD – «create read update delete»

SQL – Structured Query Language

DRY – «don`t repeate yourself»

W3C – World Wide Web Consortium

MVC – Model View Controler

OS – operating system

IE – Internet Explorer

CSS – Cascading Style Sheets

CMS – Content Management System

СУБД – система управління базою даних

PaaS – «Platform as a Service»

CSR – client side render

ВСТУП

Через пандемію COVID-19, а потім і через повномасштабну війну в Україні деякі школярі та студенти вимушено або по бажанню навчаються дистанційно. Багатьом з них складно освоювати матеріал, готуватися до іспитів та контрольних робіт через такий формат навчання, тому набули популярності ресурси для проведення відеоконференцій, створення інтерактивних дошок, онлайн-тестування тощо. Хоч перехід до онлайн навчання і є вимушеним, але необхідним.

Ще на початку пандемії у 2019 році більшість вчителів, викладачів та батьків зіштовхнулися з питаннями щодо того, які інформаційні ресурси будуть зручним для проведення занять, як перевірити рівень підготовки школярів і студентів, як організувати навчальний процес максимально комфортно для всіх його учасників. В той час виникла низка наступних проблем, які можна поділити на три категорії: проблема технічного забезпечення, програмного забезпечення та інформаційної обізнанності вчителів та учнів [2].

Створення зручних додатків для навчання та організації навчальних процесів є одною з найскладніших проблем сучасної освітньої галузі.

З метою полегшення організації навчального процесу для учнів та їх батьків було вирішено створити web-додаток для організації свого навчального плану.

Метою проєкту є створення додатку «Щоденник», що полегшить процес організації навчання для учнів та студентів.

Основними задачами є:

- розробка бази даних;
- створення зручного інтерфейсу для доступу до бази даних;
- забезпечення можливості зміни, видалення і доповнення даних, що зберігаються в базі;
- забезпечення пошуку невідповідностей у вже створених завданнях при зміні або доповненні даних, що зберігаються в базі;
- забезпечення відображення завдань при вході в окремий обліковий запис.

Результатом роботи є працюючий додаток на віддаленому сервері, в якому можна переглядати своє домашнє завдання; додавати, редагувати та видаляти предмети та завдання; створювати та керувати власним обліковим записом.

РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД І ПОСТАНОВКА ЗАДАЧІ

Швидкий розвиток інформаційних технологій та необхідність їх використання в навчальному процесі визначає основні моделі та варіанти реалізації проєктованого додатку. Також необхідно сформулювати задачі та цілі, які має задовольнити застосунок, аби задовільнити всіх учасників відстежуваних процесів.

1.1. Аналіз предметної області

Школа – це навчально-виховна установа, а також будівля такої установи. Як і у багатьох установах, у школі існує свій розпорядок дня. Звичайний навчальний день у школі складається з наступних етапів: урок, перерва, позаурочна діяльність. Протягом навчального дня урок та перерва змінюють один одного кілька разів, і завершується день, як правило, позаурочною діяльністю.

Урок – це навчальний година, присвячена окремому предмету. Перерва – це вільний час між уроками. Позаурочна діяльність – це система додаткової освіти, яка забезпечує учня харчуванням та можливістю підготувати своє домашнє завдання наступного дня.

У школах, як і у всіх навчальних закладах, існують такі поняття як позначка та оцінка. В українських школах позначки виставляються за дванадцятибальною системою (мають значення від 1 до 12). Позначки від 1 до 3 є незадовільною оцінкою. Позначка 4 до 6 є задовільною, але недостатньо високою. Позначка від 7 до 9 – хороша оцінка, що показує хороші знання. Позначка від 10 до 12 – відмінна оцінка.

Існують різні типи освітніх установ, які дають загальну середню освіту: гімназії, ліцеї та загальноосвітні школи.

Вступають до загальноосвітнього закладу у 6 – 7 років, а завершують навчання у 17 – 18 років. Класична програма у школі для загального середнього освіти передбачає 11 класів і стільки ж років для навчання учнів.

Навчальний рік починається з першого вересня, закінчуватись він може або у червні, або у травні. Основний спосіб поділу навчального року – це розподіл на чотири чверті, тобто два семестри. Є чотири чверті та між кожної з них учнів відпускають на канікули: «осінні», «зимові», "весняні", "літні".

Загальна оцінка з предметів, що вивчаються, виставляється в кінці кожного семестру, або чверті. Річна оцінка виставляється (як і належить) в наприкінці кожного року. Іноді оцінка за півріччя відзначається спільно з оцінками за чверть.

Наприкінці 9 та 11 класу складаються іспити з предметів (Державна підсумкова атестація та Зовнішнє незалежне оцінювання відповідно), результати яких разом з річними оцінками заносяться до атестату про повну або середню загальну освіту. Після закінчення останнього класу, всі учні здають тест Зовнішнього незалежного оцінювання, за результатами яких вирішується їх вступ до вищого навчального закладу. За наявності предметів, з яких немає іспитів, до атестату пишуть річну оцінку.

Учень отримує на руки свідотцво про свою повну середню освіту. В Україні він носить назву «Атестат про повну загальну освіту» після успішного завершення навчання 11 класів.

У школах можливі два типи робочих тижнів. Перший варіант – це п'ятиденний навчальний тиждень. У цьому варіанті вихідними днями є субота та неділя. У другому варіанті вихідним є тільки неділя, цей тип називається шестиденний навчальний тиждень.

Щодня у школі проводиться від чотирьох до восьми уроків. У класах з першого по четвертий включно можна проводити не більше п'яти уроків, також дозволено проводити не більше шести уроків у 5 та 6 класах, а 7 – 11 класах трохи більше семи. У цій системі проводять уроки протягом 45 хвилин. Усі уроки діляться перервами, зазвичай від п'яти до двадцяти хвилин. Учні виконують домашні завдання, окрім свого навчання в класах [1].

1.2. Призначення та область застосування програмного продукту

Одна з основних складових навчального процесу – домашнє завдання – дає можливість учням повторити матеріал та краще його засвоїти. З цієї причини створення зручного додатку для запису домашнього завдання є фактором оптимізації використання обмежених ресурсів, а саме часу та творчої віддачі учнів та студентів. Оскільки інтереси учасників навчального процесу є різними, то і задача створення та виконання домашнього завдання виявляється задачею з великою кількістю критеріїв.

Задачу створення додатку для запису домашнього завдання можна розглядати як деяку програму, що може замінити звичайну рутинну роботу учнів, що зазвичай займає багато часу. Вона має бути інтуїтивно зрозуміла як для батьків та учнів, так і для будь-якого іншого користувача. Тому сама задача відслідковування виконання домашнього завдання є лише частиною складного процесу організації навчального процесу.

Створюваний програмний продукт призначений для полегшення ручної та розумової праці учнів та студентів при створенні власного навчального плану. Використання цього застосунку дозволить економити та сили, які витрачаються на виконання цього процесу.

1.3. Огляд існуючих аналогів

Разом з необхідністю переходу на дистанційне навчання, виникла потреба використання різноманітних додатків та програм для полегшення рутинних навчальних задач. Наприклад, великої популярності набули додатки для відеоконференцій, онлайн-тестування, а також застосунки для створення навчального розкладу.

1.3.1. My Study Life

Студентський онлайн-планувальник My Study Life дає змогу відстежувати всі ваші заняття, уроки, завдання та іспити – будь-де та на будь-якому пристрої [3]. My Study Life організовує академічне життя та показує нагадування про урок або домашнє завдання. Краща організація дозволить бути більш продуктивним студентом, заощаджуючи час (рис.1.1).

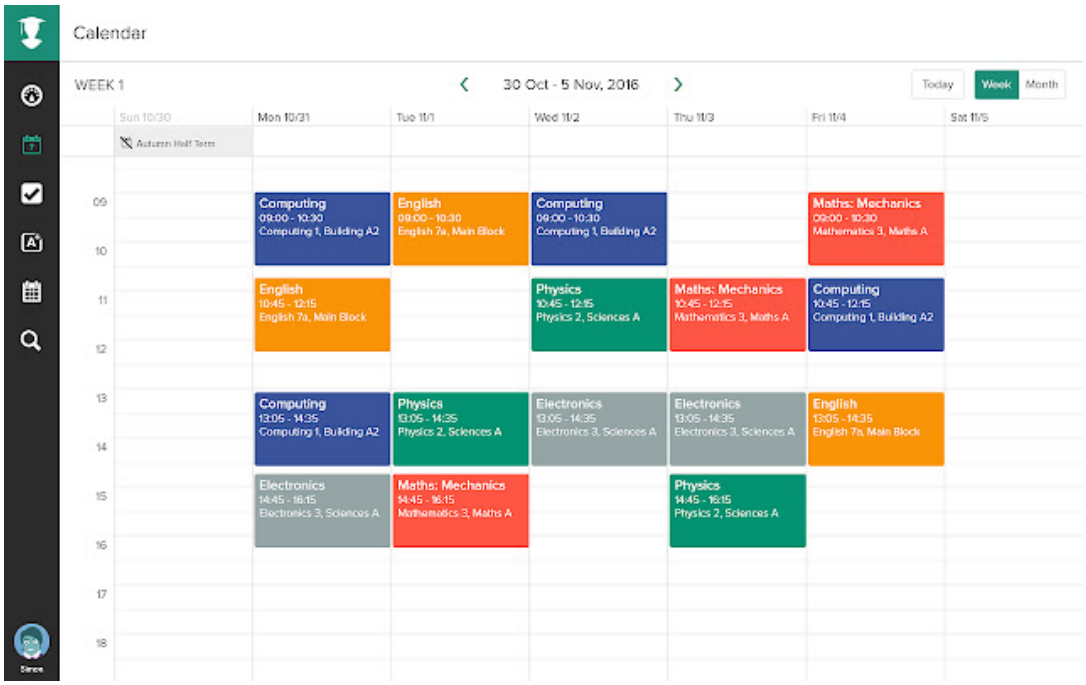

Рис.1.1. Приклад розкладу в My Study Life

My Study Life підтримує чергування розкладів, а також традиційні тижневі розклади. MSL дозволяє вводити шкільні предмети, організовувати заняття та вводити інформацію про свої уроки, надаючи можливість легко стежити за своїм розкладом.

My Study Life дозволяє створювати задачі – від найбільших до найменших. Також в цьому додатку існує функція присилання нагадувань.

До перевагами цього додатку є:

- можливість створення власного розкладу;
- можливість додавання задач;

- відправка нагадувань про урок.

Недоліками є відсутність української локалізації, додаток повністю англійською мовою.

1.3.2. Canva

Canva [4] – це застосунок для власних презентацій, інтерактивних дошок, постів та іншого контенту. Canva доступна як мобільна та веб-версія. Додаток дозволяє користуватися своєю базою зображень, шаблонів та ілюстрацій.

За допомогою шаблонів Canva можна створювати свій навчальний план, а потім зберегти його у вигляді фото. Приклад такого розкладу можна побачити на рис. 1.2.

До переваг відноситься приємне оформлення, різноманіття шрифтів та можливість вибору зручного формату відображення розкладу. З мінусів – відсутність автоматизованого створення розкладу, можливості додавання та зберігання задач, відсутність нагадувань.

Рис.1.2. Приклад розкладу в Canva

1.3.3. Google Calendar

Google Calendar – один з найпопулярніших додатків для тайм-менджменту. Він був створений у квітні 2006 року, але вийшов зі стадії бета лише у 2009. Цей web-додаток входить в пакет Google сервісів, тому для його використання необхідно мати Google акаунт [5].

Створені у Google Calendar події, задачі або нагадування зберігаються віддалено за допомоги хмарних технологій. Додаток може зберігати календар у форматі .csv (Microsoft Outlook) або .ics (Google Calendar).

Над даними в календарі можна працювати у групі, надавши доступ іншим користувачам з будь-якого куточку світу.

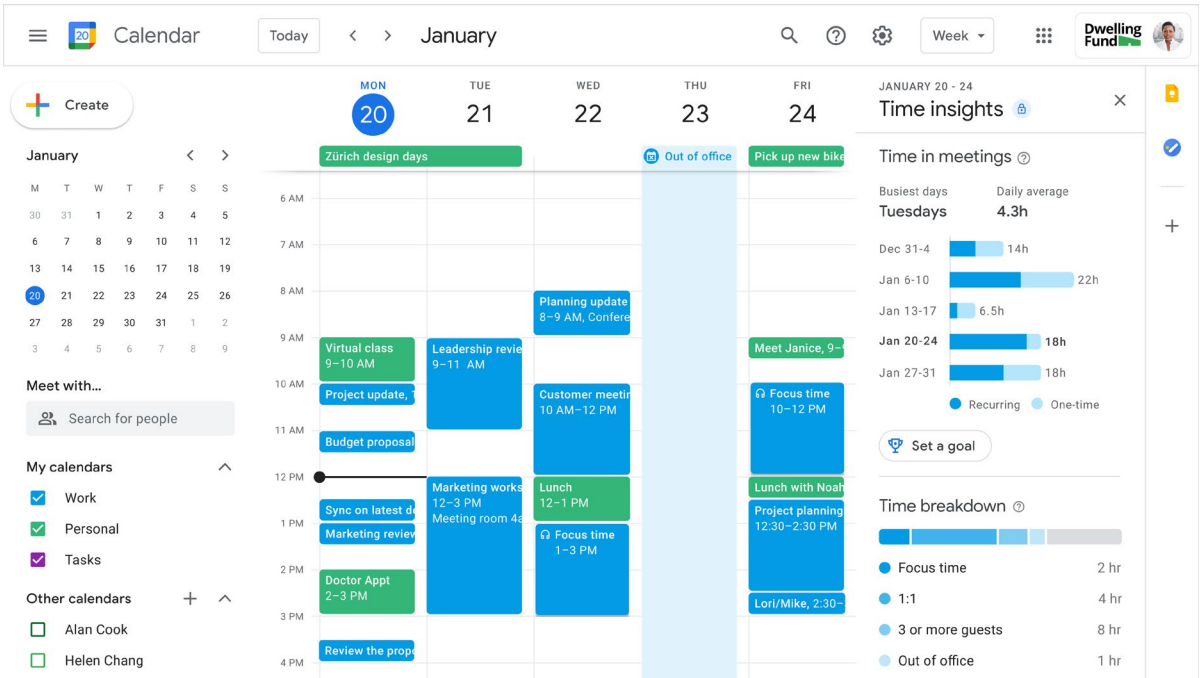

Рис.1.3. Приклад інтерфейсу Google Calendar

Плюсів у цього додатку безліч: можливість створення автоматизованого розкладу, можливість додавання завдань, інтеграція з Google Classroom, відправка нагадувань, відмітки про державні свята, можливість запрошення до подій тощо (рис.1.3). З мінусів можна назвати лише досить складний інтерфейс, в якому молодшим школярами буде непросто розібратися без допомоги дорослих.

1.3.4. Schedule Builder

Schedule Builder Online, SBO – це безкоштовна web-програма для створення тижневих або щоденних розкладів для будь-якої діяльності (наприклад, коледжу, уроків, роботи та канікул). Його розроблено з акцентом на простоту використання та персоналізацію [6].

Для роботи з цим додатком, необхідно додати заходи безпосередньо до розкладу. Розклад автоматично регулює свою тривалість. Тут існує можливість персоналізувати свій розклад, налаштувавши будь-який із наведених нижче параметрів:

- фонове зображення або колір фону;
- колір тексту;
- показати або сховати дати;
- положення позначок часу.

Після налаштування, можна надрукувати чи завантажити або експортувати розклад у бажаному форматі (PNG, PDF або iCal). Також його можна зберегти на web-сайті, створивши приватне посилання, яке можна використовувати для перегляду та редагування розкладу з будь-якого місця. Поділившись цим посиланням, є можливість запросити інших користувачів переглядати та редагувати його разом (рис.1.4).

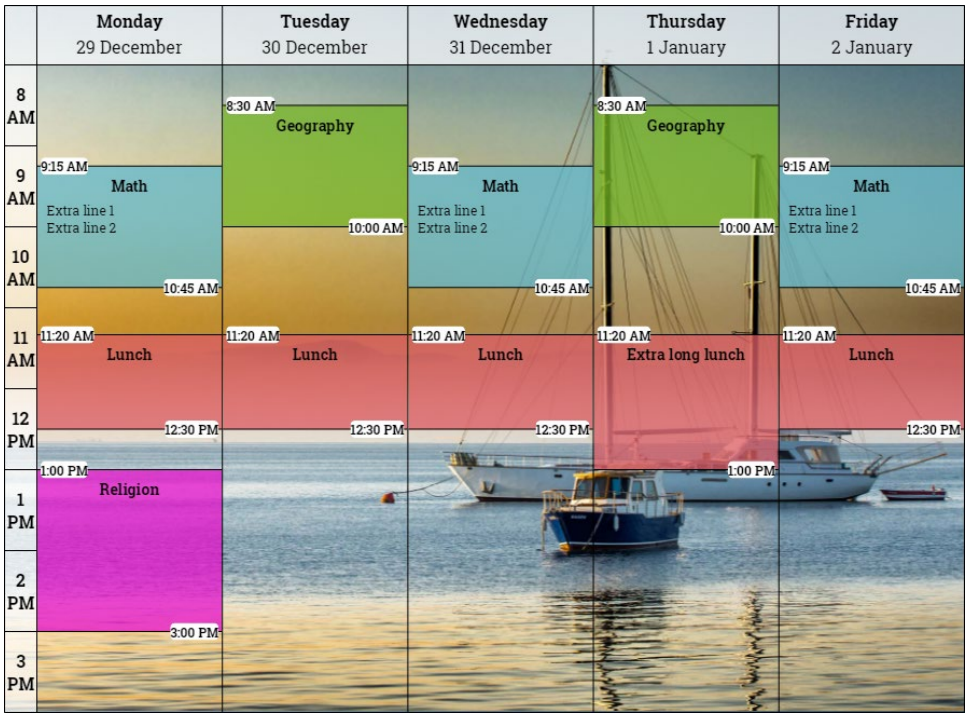

Рис.1.4. Приклад інтерфейсу Schedule Builder

З переваг цього додатку можна зазначити інтуїтивно зрозумілий інтерфейс та можливість власного оформлення розкладу. До недоліків відносяться відсутність формування задач; відсутність нагадувань; не можна додати події у вихідні дні; відсутність локалізації українською мовою.

1.3.5. School Planner

School Planner – це зручний мобільний застосунок для створення власного розкладу, запису домашнього завдання, оцінок, відслідковування графіків відвідування занять тощо. Додаток розроблений для планшетів і пристроїв, що мають браузер Google Chrome.

Перевагами цього додатку є широкий вибір функцій:

- створення планів на день, тиждень, місяць тощо;
- створення власного повторюваного розкладу;
- можливість запису оцінок та відслідковування їх динаміки;
- можливість перегляду відвіданих занять;

- можливість запису та зберігання лекцій, семінарів вбо уроків у вигляді аудіофайлу.

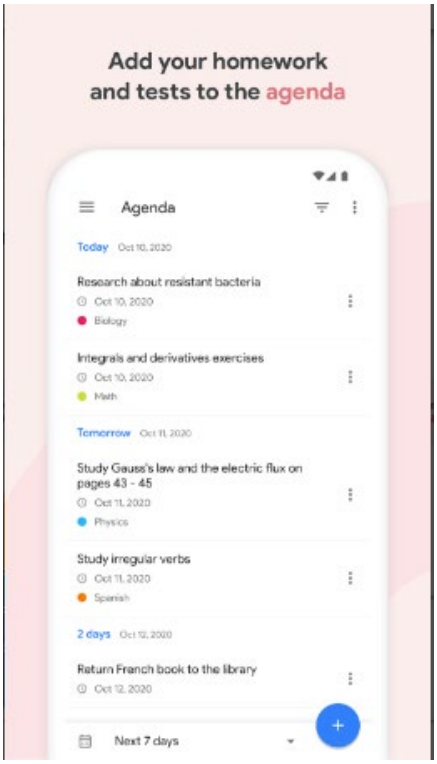

Рис. 1.5. Приклад інтерфейсу School Planner

З недоліків можна виділити те, що додаток доступний лише на мобільних пристроях з Google Chrome, не має української локалізації і нагадування про заняття іноді запізнюються.

1.4. Варіанти реалізації застосунка

Варто зазначити, що варіантів реалізації проектованого додатку існує безліч, і кожен з них має як переваги, так в недоліки. Можливість роботи на декількох пристроях в одному додатку є ключовим критерієм при розробці, але при цьому необхідно враховувати, чи можливо реалізувати задані функції на обраній платформі.

1.4.1. Десктопні застосунки

Десктопні програми зазвичай потребують операційної системи персонального комп’ютера. Такі програми встановлюються в систему за допомогою інсталятора.

Основною рисою таких програм є робота без підключення до Інтернету. Хоча багато десктопних додатків все ж таки мають з'єднання з мережею, аби автоматично встановлювати оновлення, працювати групою або ж спілкуватися з колегами.

Проте велика кількість компаній намагаються зберегти конфіденційність своїх даних, тому все ж таки утримуються від підключення до глобальної мережі. В такому випадку розробка такої програми ведеться «під ключ», а з'єднання встановлюється в закритій локальній мережі.

Загалом, серед розробників вважається, що десктопні додатки мають ширший функціонал та більшу продуктивність. Проте тут ключовим критерієм є апаратне забезпечення. Якщо встановити додаток з широким функціоналом на слабку апаратну основу, то продуктивність різко знижується.

Розробка комп’ютерних додатків справді має свої переваги.

- Функціональність.

При замовленні індивідуальної розробки, можливо досягти унікального функціоналу для кожного користувача та відповідність усім вимогам.

- Безпека та надійність.

Насправді, web-додатки досить вразливі до хакерских атак. Тому, якщо беззпека додатку понад усе, то краще обирати варіант десктоп додатку. Якщо цей додаток буде працювати в закритій локальній мережі, то ризик атак на такий застосунок майже нульовий.

- Мобільність.

Десктопні додатки мають можливість роботи у будь-якій операційній системі. При цьому функціонал такого додатку буде незмінним. Таке не можна

сказати, наприклад, про web-додатки, адже у їх випадку часто бувають випадки несумісності браузера і певних функцій застосунку.

При роботі десктопний застосунок використовує ресурси та пам'ять комп’ютера. В такому випадку, може виникнути систуація, коли застосунок не є достатньо продуктивним на старому апаратному забезпеченні. Але якщо застосунок розробляється для індивідуального користування, то такі пробеми легко вирішуються.

Десктопні застосунки допоможуть легко оптимізувати будь-які рутинні процеси, дозволять підвищити продуктивність роботи відділів та оптимізувати їх роботу. Десктопні програми мають багато можливостей:

- Робота без підключення до мережі Інтернет.

Додаток може працювати без використання мережі Інтернет. Дані для додатку будуть зберігатись у пам'яті комп'ютера, одже не потребують ніякого хмарного середовища. Це є гарантією безпеки та продуктивності.

- Робота з підключенням до мережі.

Якщо така програма має підключення до мережі Інтернет, то це дозволяє зберігати дані у хмарному середовищі, мати сумісний доступ до даних та зручно комунікувати у додатку.

- Швидкий запуск.

Так як програма не оновлюється автоматично та не підвантажує дані з Інтернету, системні файли такого додатку не перевантажуються, а отже додаток швидко запускається навіть на слабкому апаратному забезеченні.

- Якісний інтерфейс користувача.

Часто виникає потреба підлаштувати додаток під конкретного користувача, і зроботи це з десктопним додатком дуже просто.

- Використання додаткової периферії.

Десктопні застосунки мають безпосередній доступ до файлів та ресурсів комп'ютера, тому в них легко створити функції для управління різними периферійними пристроями: від принтера до джойстика.

Це далеко не всі можливості програм, які встановлюються на комп'ютер. Індивідуальна розробка дозволяє отримати ефективну програму для бізнесу, яка відповідає конкретним запитам [9].

1.4.2. Мобільні застосунки

Процес інформатизації навчання стає все більш складним та швидким. Зважаючи на популярність використання мобільних пристроїв, створення мобільних додатків для навчання є вкрай необхідним [7].

Перш ніж перейти до поняття мобільного додатка, слід зазначити той факт, що в системі освіти з'явився і розвивається новий напрямок – мобільне навчання (M-Learning). Якщо десятиліття тому був популярним термін CALL (навчання через комп'ютер), нині з'явився термін MALL (навчання через мобільні телефони).

У збірці міжнародної конференції IADIS, «Мобільне навчання - це будь-яка навчальна активність, в якій переважно або виключно використовуються портативні пристрої – телефони, смартфони, планшети, іноді ноутбуки тощо, але не звичайні настільні комп'ютери» [7].

Останнім часом все більше зростає кількість дітей, які можуть користуватися мобільними телефонами на професійному рівні. В даний час мобільні телефони є у кожного, так як це найзручніший пристрій для миттєвого доступу до будь-якої інформації, а учні використовують їх також для освітніх цілей.

У світі розвитку цифрових технологій дуже швидко розвивається і зростає кількість мобільних додатків

Мобільний додаток – це програмне забезпечення створене для мобільних пристроїв: смартфонів, планшетів тощо. App Store, Google Play, mobile market, windows phone store – це магазини, в яких можна встановити мобільні додатки безкоштовно або за плату».

Основною метою мобільних додатків було використання для швидкої перевірки електронної пошти, але високий попит даних додатків призвів до розширення їх призначень і в інших областях, таких як ігри для мобільних телефонів та GPS, спілкування, перегляд відео та користування інтернетом.

Сьогодні можна говорити і про значущість для освіти, завдяки їхній мобільності, наочності та доступності, також завдяки можливостям, які вони дають.

Використання мобільних додатків для освітніх закладів має низку переваг, як:

- мобільність;
- підвищення якості комунікації;
- різномаїття дизайнерських рішень;
- персоналізація.

Необхідно відзначити, як і всі новинки технологій, крім очевидних плюсів, використання мобільних додатків у навчанні має і ряд недоліків, наприклад:

- технічні недоліки як маленький екран, постійний заряд, обсяг пам'яті, проблеми доступу до Інтернету;
- соціальні проблеми, такі як учні не можуть придбати собі телефон з гарним апаратним забезпеченням.

Незважаючи на безліч негативних відгуків щодо використання мобільних додатків та мобільних телефонів загалом, можна з упевненістю сказати, що освіта не стоїть на одному місці і так само, як і цифрові технології, шукає нові підходи, нові методики.

1.4.3. Web-застосунки

Ставлення користувачів до web-додатків на сьогоднішній день залишається неоднозначним. Переваги браузерних програм очевидні [10].

По-перше, для роботи з web-додатком не потрібно встановлювати додаткове програмне забезпечення, достатньо мати лише браузер, який за замовчуванням встановлений в кожну операційну систему.

По-друге, при встановленні додатку на свій комп'ютер можуть виникнути помилки, які складно вирішити недосвідченому користувачеві. Часто додаток

потрібно налаштовувати або вставлювати додаткові розширення. При роботі з web-додатком такі задачі переходять на сторону сервера.

По суті, у ролі адміністраторів web-додатків виступають розробники, які працюють в одному місці. У корпоративному секторі, наприклад, це економічно набагато вигідніше і ефективніше, ніж повна команда програмістів та адміністраторів, які займаються встановленням та налаштуванням десктопних додатків на машинах користувачів.

По-третє, web-додаткам не потрібне апаратне забезпечення комп'ютера для своєї роботи. Для запуску web-додатки користуються лише браузером.

Окрім цього, немає проблем з оновленнями та старими версіями програми, адже при виході нової версії додатку він автоматично запускаєься на сервері, а отже всі користувачі автоматично переходять на цю версію. Також web-додатку привабливі тим, що запускають на більшості пристроїв, де є браузер, а отже користувачі можуть бути насправді мобільні.

На жаль, у web-додатків існують і слабкі сторони, які, звичайно, не затьмарюють переваг.

Доступ в Інтернет зараз доступний не скрізь. В багатьох районах України, навіть до війни, вартість трафіку та ширина інтернет-каналу залишалась незадовільною.

Крім того, існує безліч додатків, які не можуть бути замінені браузерними (принаймні в найближчому майбутньому). Наприклад, неможливо у браузері створювати складні тривимірні моделі.

І нарешті найголовніший мінус web-додатків – багатьох користувачів бентежить той факт, що їхні дані зберігатимуться та оброблятимуться на чужому сервері. Адже потенційно це може призвести до витоку, втрати чи спотворення інформації (і в окремих випадках, приводить).

І ще одна досить значуща деталь. Якщо всі програми будуть працювати виключно на віддаленому сервері, в роботу якого користувач не зможете ніяк втручатися, то «піратські» серії додатку будуть недоступні. За будь-яке потрібне

комерційне програмне забезпечення доведеться платити або шукати безкоштовні варіанти.

1.5. Постановка задачі

Метою створюваного проєкту є зручна взаємодія користувача з програмним продуктом та зведення до мінімуму зусиль користувача при складанні навчального плану.

Основними задачами є:

- розробка структури бази даних;
- створення зручного інтерфейсу для доступу до бази даних та складання розкладу;
- забезпечення доступу до даних, що зберігаються в базі;
- забезпечення можливості зміни і доповнення даних, що зберігаються в базі;
- забезпечення пошуку невідповідностей у вже створеному розкладі при зміні або доповненні даних, що зберігаються в базі;
- забезпечення відображення домашніх завдань при вході в окремий обліковий запис.

Зважаючи на основні цілі та задачі проєкту, можна висунути наступні вимоги до проєкту:

- створення облікових записів для кожного користувача. Це необхідно для розподілу даних серед користувачів і зручного управління цими даними;
- співставлення предмету і ряду задач до нього;
- співставлення задачі і кінцевого часу виконання цієї задачі. Це важливо для загального розуміння темпу навчального процесу та дає можливість його скорегувати;
- можливість запису додаткової інформації про предмет, викладача тощо;
- можливість відмічати виконані задачі.

Аналізуючи види реалізації програмного продукту, було визначено, що застосунок для планування начального розкладу.

Висновки до розділу 1

Через повномасштабну війну та пандемію, більшість студентів вищих навчальних закладів та школярів були змушені перейти на навчання в онлайн форматі. Як наслідок, виникла небхідність використання різноманітних додатків та програм для полегшення навчального процесу: додатків для відеоконференцій, соціальних мереж, програм онлайн тестування, інтерактивних дошок тощо.

Наразі існує безліч додатків для складання власного розкладу занять та навчального плану, і кожен має як свої переваги, так і недоліки.

Також є різні варіанти реалізації додатків, наприклад, декстоп, мобільні або ж web-застосунки. Десктопні застосунки вирізняються широкою функціональною частиною та продуктивністю, але такі застосунки необхідно встановлювати на нове апаратне забезпечення, аби ними було зручно користуватися. Мобільні застосунки зручно використовувати будь-де, але вони мають обмежену функціональність та мале розширення екрану. Web-додатки наразі найпопулярніші через відсутність потреби встановлення програмного забезпечення. Проте вони досить вразливі до атак та шахрайства, тому активно ведеться робота над вдосконалення захисту таких програм.

Зважаючи на результати аналізу предметної області та вивчення проблематики , було вирішено створити web-додаток «Щоденник» для онлайн навчання.

РОЗДІЛ 2. ПРОЕКТУВАННЯ WEB-ЗАСТОСУНКУ, ОГЛЯД ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ

Істотною перевагою використання web-додатків є їх незалежність від апаратного забезпечення, проте значущу роль при створенні такого додатку має браузер, у якому цей додаток буде використовуватись. Однак використання CSS та DOM та інших іструментів може викликати труднощі при розробці таких додатків та їх підтримки. Крім того, можливість користувача змінювати параметри браузера може заважати коректній роботі додатку.

2.1. Архітектура web-застосунків

2.1.1. Загальні архітектури web-застосунків

Web-програми розроблюються у різних цілях. Багато з них використовуються для створення електронних таблиць або презентацій, продажу або покупки товарів, групової роботи над проєктами, спілкування тощо.

За допомоги веб-додатків можна створювати будь-що: малюнки, графіки, нотатки та навіть відео і музичні файли. Наразі існують навіть хмарні середовища для розробки програмного забезпечення на різних мовах програмування.

Умовно веб-додатки можна розділити на декілька типів, в залежності від різних комбінацій його складових [11].

Backend або серверна частина додатку працює на віддаленому комп'ютері. Ця частина додатку може бути реалізована різними мовами програмування: PHP, Ruby, Python, C# та інші. Якщо створюваний додаток працює лише за допомоги серверної частини, то в результаті переходів між сторінками та відправки форм , оновлення

даних сервер буде створювати новий HTML-файл і так користувач буде бачити нову сторінку.

Frontend або клієнтська частина додатку виконується у браузері користувацького пристрою. Ця частина зазвичай реалізовується за допомоги JavaScript.

Застосунок може містити лише клієнтську частину, якщо немає необхідності зберігати дані до наступної сессії. Таким чином створюють ігри, фоторедактори і т.д.

Single page application або SPA – це односторінковий додаток. Найцікавіший варіант, коли одночасно використовується backend і frontend. Такий додаток не перезавантажує сторінки, а одразу виводить інформацію на екран.

Прогресивні веб-додатки (англ. progressive web application, PWA) – це web-застосунки, що створюються за допомогою спеціальних web-технологій та стандартних шаблонів, що дозволяє одночасно використовувати переваги і декстопних, і web-додатків.

Такі застосунки зберігають дані в клієнтській частині сайту, тому ним можна користуватись навіть без підключення до мережі.

Прогресивні застосунки мають такі переваги:

- доступні для пошуку в різних пошукових системах;
- такі додатки можна встановити на робочий комп’ютер;
- можливість створити посилання для спільної роботи над проєктом;
- не залежить від підключення до мережі;
- додаток може надсилати повідомлення про додатковий новий контент;
- такі додатки мають дизайн, що пристосований для будь-яких пристроїв: телефонів, планшетів, комп’ютерів;
- дані між користувачем та сервером захищені.

2.1.2. Принцип роботи web-застосунків

Web-застосунки містять серверну (backend) та клієнтську частини (frontend) [11]. Користувач може взаємодіяти з клієнтською частиною додатку через браузер (Safari, Chrome, Edge, Firefox тощо). Коли користувач відправляє запит до сервера через Інтернет, то сервер оброблює дані, шо надіслав користувач, та відправляє в зворотньому напрямку користувачеві відповідь (рис. 2.1)

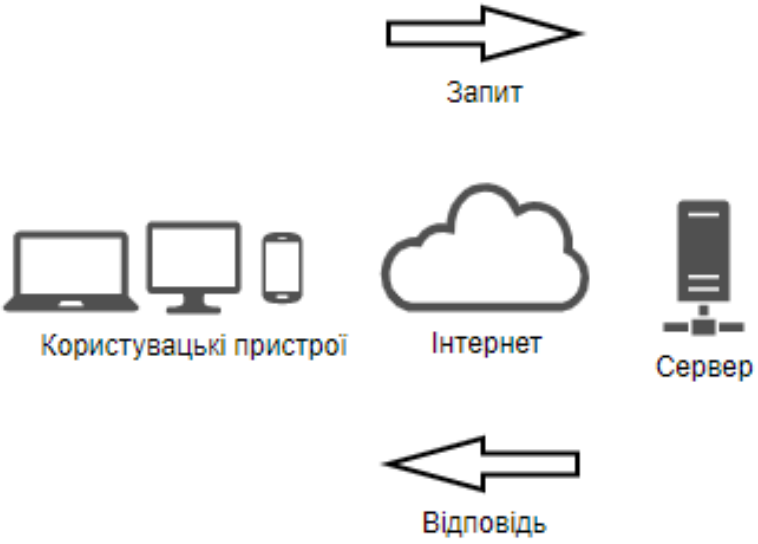

Рис. 2.1. Принцип роботи web-додатку

Відповіддю сервера може бути файл HTML, XML або JSON – це залежить від обраного типу рендерингу. Рендеринг – це процес заповнення сторінки даними від серверної частини додатку. Рендеринг може відбуватися як виключно на стороні сервера, так і на стороні клієнта, або ж різних пропорціях розподілятись між цими двома частинами.

2.1.3. Статичні та динамічні сторінки

Перші web-сторінки були виключно статичними. Дані у цих сторінках заповнювалися вручну. Звісно, вони містили вже знайомі таблиці або списки, але загалом вони були дійсно статичними.

Робота такої сторінки досить проста. Клієнт надсилає серверу запит за певним URL, а сервер відповідає користувачеві виведенням простого HTML-файлу [17]. Приклад такої сторінки наведено на рис.2.2.

Рис. 2.2. Статична web-сторінка

Щоб реалізувати кілька розділів, потрібно було використовувати кілька сторінок з однаковим кодом та різним наповненням. Наприклад, сторінка «Про сайт» буде містити такий код, як на рис.2.3.

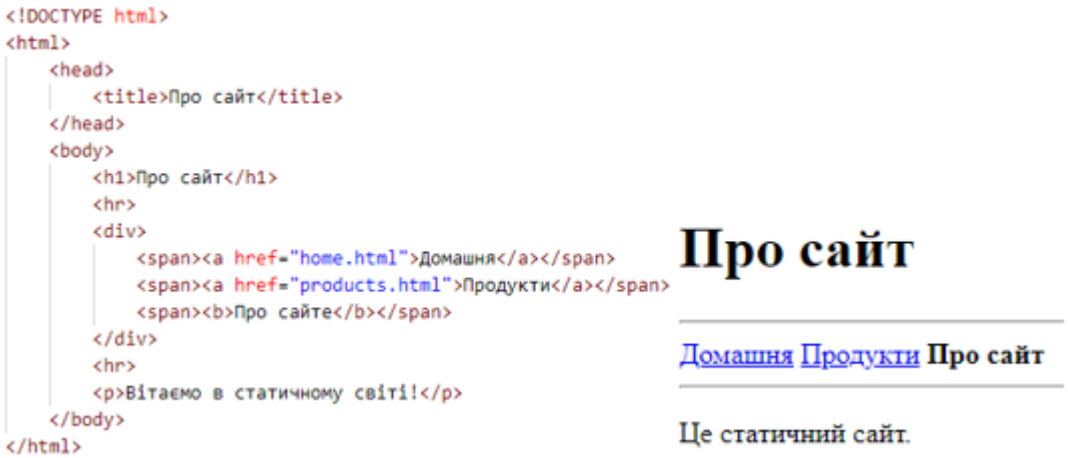

Рис. 2.3. Статична сторінка «Про сайт»

Як видно, більшість коду не змінилася. Змінився текст та оформлення меню: жирним виділено сторінку «Про сайт», а «Домашня» сторінка тепер — посилання. Ці зміни вносились вручну.

Для сторінки "Продукти", на якій наведено перелік продуктів із посиланнями на сторінки окремих продуктів, при змінах необхідно було б змінювати список, інформацію про наявність продукту, ціну тощо.

У 1993 році було створено Common Gateway Interface (CGI) – інтерфейс, що пов’язує виконувану програму (шлюз) з сервером. Шлюз можу бути написаний будь-якою мовою програмування – C++, Visual Basic, AppleScript тощо. Найчастіше використовують Perl.

Такі сценарії дозволяють наповнювати певний шаблон різними даними, що значно полегшило роботу програмістів, адже тепер не потрібно було вручну змінювати дані у статичній сторінці. Але з боку користувача сторінка все одно залишалась статичною.

У 1995 році виник JavaScript, що дозволив додатку реагувати на дії користувача та виводити на екран повідомлення. Цього ж року було створено мову PHP, яка змогла поєднати HTML-код з логікою програми.

Для домашньої сторінки можна було б створити такий динамічний код (рис.2.4):

```
<!DOCTYPE html>
<html>
    <head>
        <title><?php echo $title ?></title>
    </head>
    <body>
        <h1><?php echo $title ?></h1>
        <hr>
        <div>
            <?php
            $i = 0;
            foreach($menu as $menu_item) {
                if($menu_item == $current){
            ?>
<span><b><?php echo $menu_item ?></b></span>
            <?php
                }else{
            ?>
<span><a href="<?php echo $menu_links[$i] ?>"><?php echo $menu_item ?></a></span>
            <?php

                }
                $i++;
            }
?>

        </div>
        <hr>
        <p>Вітаємо в динамічному світі!</p>
    </body>
</html>
```

Рис.2.4. Динамічна web-сторінка

При запуску серверу ця сторінка вставляє в код значення змінних, за допомоги циклу for перебираються елементи сторінки та виводяться на екран.

Сьогодні для написання коду для web-застосунків використовується безліч мов програмування та спеціальних фреймворків. Найпопулярніші з них:

- Node.js;
- PHP + Laravel;
- Python + Django;
- Java (з використанням Java Servlet API);
- мови платформи .NET (C#, VB) + ASP.NET,
- Ruby + Ruby on Rails;
- Go.

2.2. Технології та інструменти, використані у web-застосунку

В розробці фреймворк – це набір готових бібліотек та інструментів, які допомагають створювати web-додаток. В створюваному додатку буде використовуватись фреймворк Django, що написаний на мові Python.

Першим етапом користувацький запит переходить у роутер (URL dispatcher), який вирішує, яку функцію необхідно викликати. Функція, яка викликається цим роутером називається видом (view). Всередині може міститись будь-яка бізнес-логіка, але начастіше це або з бази даних отримують дані, обробляють їх та надають користувацькій стороні, або дані надійшли з користувацької форми, тому вони перевіряються, оброблюються та зберігаються у базу даних.

Дані програми зберігаються у базі даних (БД). Найчастіше використовуються реляційні бази даних.

Дані в базі даних можна створювати, читати, змінювати та видаляти. Іноді для позначення цих дій можна зустріти абревіатуру CRUD (Create Read Update Delete). Для запиту до даних у базі даних використовується спеціальна мова SQL (structured query language).

У Django для роботи з базами даних використовуються моделі (model). Вони дозволяють описувати таблиці і робити запити на звичному форматі розробнику Python, що набагато зручніше. За цю зручність доводиться платити: такі запити повільніші та обмежені у можливостях порівняно з використанням SQL.

Отримані з бази даних дані готуються до відправки до користувацького інтерфейсу. Вони можуть бути підставлені як шаблон (template) та надіслані у вигляді HTML-файлу. Але у випадку односторінкової програми це відбувається лише один раз, коли генерується HTML-сторінка, на яку підключаються всі JS-скрипти. В інших випадках дані серіалізуються та відправляються у JSON-форматі.

2.2.1. Мова програмування Python. Модуль Django

Мова програмування Python наразі є одною з найпопулярніших мов програмування, і на це є низка причин. По-перше, вона неймовірно ефективна: порівняно з програмами на інших мовах, програми на мові Python будуть виконувати більший об’єм задачі за меншу кількість рядків коду. Також синтаксис Python допомагає писати «чистий» код, який легко читається, виправляється та розширюється [13].

Мова програмування Python використовується для різних завдань: створення ігор і web-застосунків, вирішення бізнес-задач, написання допоміжних внутрішніх програм для конкретної предметної області та інше. Python широко використовується в науці, наприклад, в академічних дослідження, так і в прикладних сферах.

Стандартна бібліотека Python – це набір модулів, що ідуть у комплекті при завантаженні середовища розробки Python. Маючи загальне уявлення про роботу функцій і класів, можна використовувати модулі стандартної бібліотеки Python.

Модулі можна завантажувати із зовнішніх джерел за допомогою команди pip install <package>. Ця команда завжди шукає останню версію пакета та встановлює її. Він також шукає залежності, перераховані в метаданих пакета, і встановлює їх, щоб переконатися, що пакет відповідає всім вимогам, які йому потрібні [12].

У Python web-застосунки створюють за допомогою модулів Django та Flask.

Django – це web-фреймворк на мові програмування Python, який дозволяє швидко створювати безпечні та підтримувані web-сайти. Створений досвідченими розробниками, Django бере на себе більшу частину клопоту web-розробки, тому можна зосередитися на написанні свого web-додатку. Він безкоштовний і з відкритим вихідним кодом, має зростаючу та активну спільноту, відмінну документацію та безліч варіантів як безкоштовної, так і платної підтримки [14].

Django допомагає писати програмне забезпечення, яке буде:

- Повним.

Django слідує філософії «all inclusive» і надає майже все, що розробники можуть захотіти зробити. Оскільки все, що потрібно, є частиною єдиного продукту, проєкти з його використанням відповідають послідовним принципам проєктування і має велику і актуальну документацію.

- Різностороннім.

Django може бути (і був) використаний для створення практично будь-якого типу web-додатків– від соціальних мереж та новинних сайтів до блогів та систем керування. Також він може користуватись рендерингом для файлів різних форматів (включаючи HTML, JSON, XML тощо). За необхідності Django може бути розширеним для будь-якого функціоналу.

- Безпечним.

Django дозволяє розробникам захистити свй проєкт наскільки це можливо. Наприклад, при роботі з додатком через панель адміністратора, Django зашифровує паролі користувачів за допомоги геш.

Геш пароль – це значення сталої довжини, що створене шляхом обробки пароля через геш-функцію. Django може перевірити правильність користувацького паролю, використовуючи геш-функцію та порівняти введене користувачем значення із збереженим значенням геш. Геш-функція є односторонньою, тому потенційному злодію важко буде визначити справжній пароль.

Django за замовчуванням забезпечує захист від багатьох атак, включаючи міжсайтовий скриптинг, SQL-ін'єкцію, клікджекінг та підробку міжсайтових запитів.

- Масштабується.

Django застосовує "shared-nothing" архітектуру (кожна частина цієї архітектури може бути замінена або змінена у разі необхідності). Розподіл цих частин означає, що Django-застосунок масштабується при збільшенні трафіку додаванням обладнання: серверами кешування, серверами баз даних або серверами застосунків. Деякі найбільш завантажені трафіком додатки успішно масштабували за допомоги Django (наприклад, Instagram та Disqus).

- Зручний у супроводі.

Код Django керується принципами та шаблонами проектування, які заохочують створення повторно використовуваного коду. У цьому фреймворку активно застосовується принцип "Don't Repeat Yourself" (DRY, "не повторюйся"),то при створенні коду немає непотрібного дублювання, що значно скорочує обсяг написаного коду. Django дозволяє групувати додатки у модулі (відповідно до шаблону Model View Controller (MVC)).

- Переносним.

Django написано на Python – кросплатформеній мові програмування.Це означає, що застосунок не буде прив’язаний до певної операційної системи, тобто Linux, Windows та Mac OS. Авжеж, Django підтримується більшістю web-хостингів, які часто мають власні інструменти для роботи з Django.

2.2.2. Інструменти створення бази даних

База даних – це сховище даних, яке дозволяє зручно організовувати та безпечно зберігати дані. Для роботи з web-застосунками база даних є невід’ємною частиною, адже дозволяє зберігати дані користувача та управляти ними.

Код додатку за допомогою серверу взамодіє з базою даних. Сервер може видобувати дані, читати їх, обробляти та повертати до клієнта. У різних мовах

програмування існує безліч пакунків та засобів роботи з базами даних, такими як Oracle, MySQL, PostgreSQL тощо.

У розглянутому вище випадку на сторінці «Продукти» були показані продукти, список яких отримано з бази даних. Можливо, цей список буде відсортований або відфільтрований за будь-яким критеріям, заданим користувачем. Такий відбір можна здійснювати при виправленні запиту до бази даних. У системі адміністрування такого web-ресурсу можна додавати продукти, змінювати інформацію про них і видаляти їх з бази даних.

Будь-який проєкт Django – це група індивідуальних застосунків, які взаємодіють всередині нього.

Django дозволяє працювати з такими базами даних:

- SQLite;
- Oracle;
- MySQL;
- PostgreSQL.

Існує також ряд database backends, передбачених третіми частинами. Django намагається підтримувати якнайбільше функцій на всіх бекендах баз даних. Проте, не всі бази даних однакові, і можливо доведеться приймати проектні рішення про те, які функції підтримувати та які припущення зробити безпечними [15].

MySQL – це система управління реляційними базами. Її вихідний код (СУБД) відкрито. Використовується модель клієнт-сервер. Комплекс програм або ціла служба, яка використовується для роботи з базами на основі реляційної моделі, називається СУБД [12].

База MySQL – це реляційна база даних. Такі бази даних працюють на основі пов’язаних між таблиць, кожна яких реалізує яку-небудь сутність.

Комп’ютери, які можуть запускати СУБД (систему управління базами даних) називають комп’ютерами-клієнтами. Система «клієнт-сервер» є системою з використанням сервера MySQL (рис.2.1).

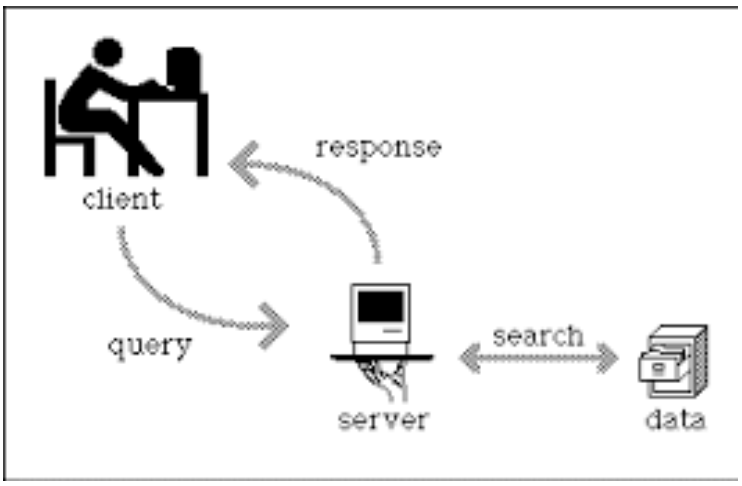

Рис. 2.5. Система «клієнт-сервер»

У 1984 році шведська компанія розробила MySQL. У 2008 році цією системою управління почала керувати компанія Sun Microsystems (США). Oracle у 2010 році викупив її знову.

Наразі ця система працює на операційних системах Microsoft Windows, Linux, macOS тощо. Інформаційні гіганти Facebook, YouTube, Twitter та Google, використовують для роботи з БД саме MySQL.

MySQL має безліч переваг, а саме:

- простота. Можна змінювати вихідний код, навіть не користуючись платною версією;
- продуктивність. Ця система підтримує безліч серверів з різною конфігурацією;
- стандартизація. Для розробників існує безліч ресурсів та вони отримують замовлене програмне забезпечення .швидко та якісно;
- безпека. Всі користувацькі записи мають високий рівень захисту. Пароль шифруються та перевіряються хостом.

Як вже було зазначено, web-сайт без використання бази даних створити складно. Аби спростити роботу розробників ще більше, MySQL дозволяє працювати з такими CMS як Wordpress та Joomla.

Розробка бази даних за допомоги MySQL включає такі етапи:

- створення база даних MySQL, визначення відношень між таблицями сутностей;
- створення запитів до SQL клієнтом;
- відповідь за надісланим запитом клієнту.

MySQL Workbench – найвідоміший інтерфейс для роботи з MySQL. Існує і багат**о інших, що працюють на різних операційних платформах.**

2.2.3. Інструменти створення шаблонів

HTML-сторінка може відображати контент з web-застосунку на сторінку користувача. В додатку, що користується тільки клієнтським інтерфейсом для відображення даних використовуються теги. Наприклад, тег дозволяє виділити блок всередині сторінки для відображення певного контенту [17].

Кінцевого виду сторінка набуває після застосування всіх стилів та відображення всього контенту. Але окрім цього, сторінка може динамічно оновлюватися, якщо, наприклад, користувач натискає на елемент меню або гіперпосилання.

Шаблон описує, який вигляд матиме сторінка, а Django наповнює її відповідними даними щоразу, як на неї приходить запит. Запит дозволяє отримати доступ до даних у виді. Функція виду приймає інформацію з запиту, готує дані, необхідні для генерації, а тоді надсилає їх назад браузеру. Часто вона використовується як шаблон. Для створенння шаблонів зазвичай використовується мова гіпертекстової розмітки HTML.

HTML (HyperText Markup Language) – це мова гіпертекстової розмітки, що застосовується для створення статичних та динамічних web-сторінок в Інтернеті. У 1990-х роках HTML був примітивною мовою для створення web-сторінок, проте наразі майже жоден сайт не створюється без використання HTML. Переважна більшість сайтів так чи інакше його використовують [17].

2014 року офіційно було запущено новий стандарт - HTML5, що зробив абсолюну революцію, додавши у вже існуючий HTML безліч нових функцій.

HTML5 починає використовувати DOM, додаються нові елементи та теги, як наприклад, теги video, audio, що дозволяють зробити HTML-сторінку більш динамічною та заповнити контентом.

Додані функції та теги в HTML5 вивели мову розмітки на новий рівень, що і стало причиною такої його великої популярності. Сьогодні HTML5 використовується не тільки для створення web-сторінок, а для створення мобільних додатків для операційних систем Android, iOS, Windows Mobile і для створення десктопних програм для настільних комп'ютерів.

У результаті, як правило, HTML 5 застосовується переважно у двох значеннях:

- HTML5 як оновлена мова розмітки гіпертексту, деякий розвиток попередньої версії HTML4;
- HTML5 як потужна платформа для створення веб-додатків, яка включає не лише безпосередньо мову розмітки гіпертексту, оновлену HTML, але й мову програмування JavaScript та каскадні таблиці стилів CSS3.

World Wide Web Consortium (скорочено W3C – Консорціум Всесвітньої Павутини) створює нові стандарти HTML5 у вигляді специфікацій. І варто зазначити, що ця версія HTML постійно розвивається та доповнюється новими функціями.

Між специфікацією HTML5 та використанням цієї технології у web-браузерах завжди був розрив. Більшість браузерів стали впроваджувати стандарти HTML5 ще до її офіційної публікації. І на сьогодні більшість останніх версій браузерів підтримують більшість функціональностей HTML5 (Google Chrome, Firefox, Opera, Internet Explorer 11, Microsoft Edge). У той самий час багато старих браузерів, як, наприклад, Internet Explorer 8 і молодші версії, не підтримують ці стандарти, а IE 9, 10 підтримують лише в деякій мірі.

При цьому навіть ті браузери, які загалом підтримують стандарти, можуть не підтримувати якісь окремі функції.

Для того, аби генерувати інформацію, яка буде відображатися на сторінці використовують шаблонні теги. Шаблонний тег відокремлюється фігурними дужками та знаками відсотка {%%}. Для виклику шаблонних функцій, Django містить цілу низку вбудованих шаблонних тегів та фільтрів шаблону. Щоб надрукувати значення змінної в шаблоні, необхідно використовувати подвійні фігурні дужки {{ім'я_змінної}}.

2.2.4. Вибір інструментів для дизайну

Інтерфейс користувача може містити інформаційні блоки та елементи керування. До прикладу, інформаційними блоками у Facebook є дописи в стрічці записів, історії, рекомендації, а елементи керування – це кнопки посилань, меню, поля пошуку, вкладок та введення вмісту для дописів, коментарів тощо.

Якщо розглянути Google Slides, то є такі елементи керування: меню, панель інструментів, панель структури, документ. Інформаційний блок тут служить орієнтиром. Загалом, це інтерактивний інтерфейс (рис.2.6).

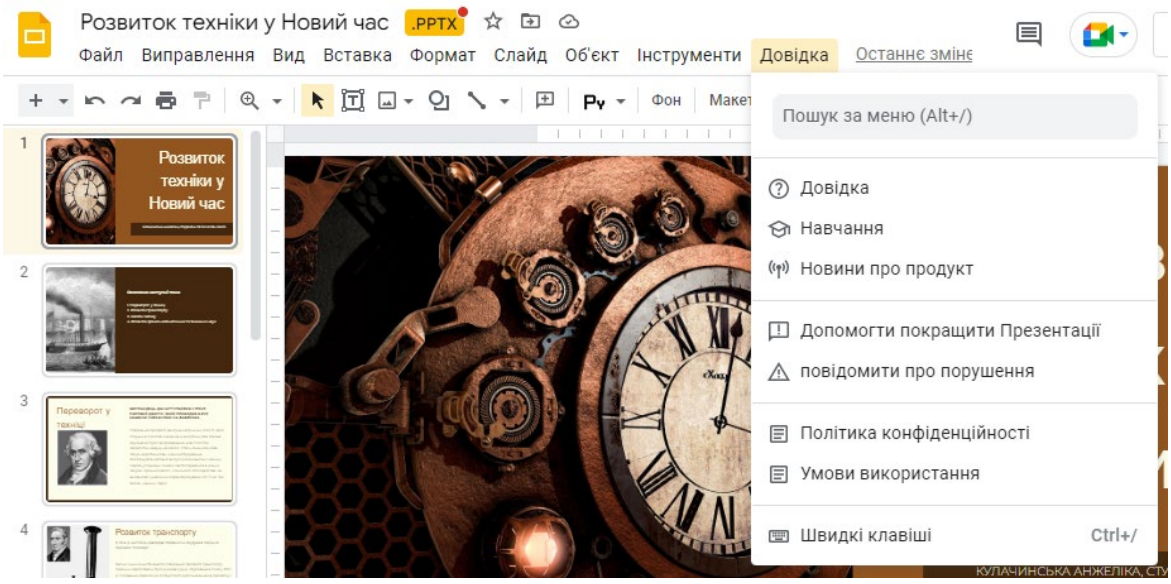

Рис.2.6. Інтерфейс Google Презентації

Інтерактивність сторінки забезпечує JavaScript, який виконує свій код в браузері. Для більш складних програм використовуються бібліотеки мови

JavaScript для спрощення написання коду. До прикладу, Google Документи і багато інших web-додатків Google використовують Closure Library, тоді як Facebook використовує бібліотеку React. В React часто використовуюється бібліотека Redux для управління станом додатків. Широкої популярності набули біблотеки для web-додатків, такі як Express.js (порівняно з Node.js), Angular, Vue.js.

Завдяки використанню цих бібліотек можна використовувати клієнтський рендеринг (рендеринг на стороні клієнта, CSR). В односторонньому прикладі рендеринг, логіка і завантаження відображаються на клієнтській стороні.

Bootstrap – це величезна колекція інструментів оформлення, що працює на основі HTML і CSS. Він містить стилі для основних елементів, які будуть застосовуватися при верстці. Використання такого фреймворку значно спрощує процес створення web-сторінок застосунку. Цей фрейморк містить велику бібліотеку стилів та дозволяє використовувати макети свої застосунків [18]. Приклад оформлення веб-сторінки за допомогою фреймворку Bootstrap наведено на рис.2.7.

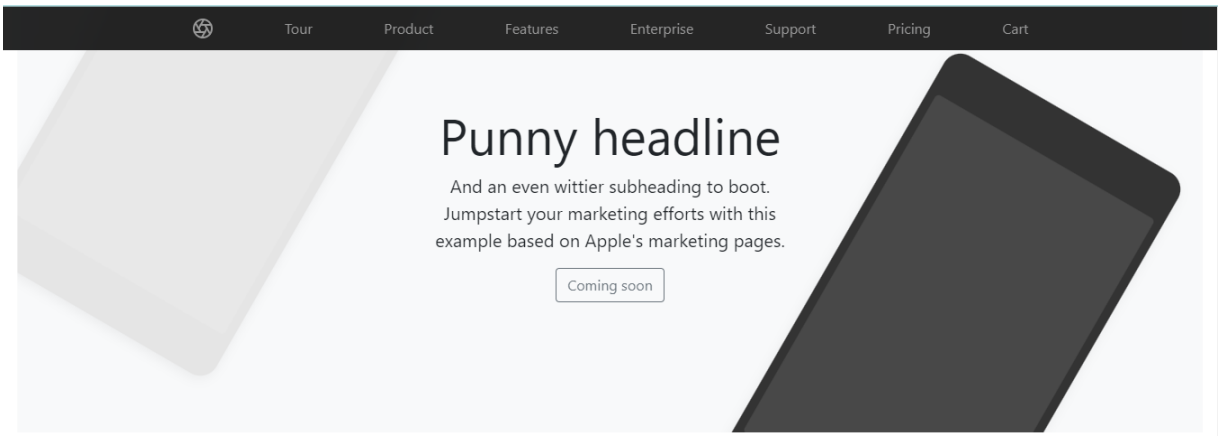

Рис.2.7. Приклад оформлення сторінки з Bootstrap4

Bootstrap ефективно масштабує веб-сайти та додатки з єдиною базою коду, від телефонів до планшетів та персональних комп’ютерів.

Основними інструментами Bootstrap є:

- сітки;
- шаблони;

- типографіка;
- медіа;
- таблиці;
- форми;
- навігація;
- алерти.

Bootstrap сучасні розробки в області HTML і CSS, тому необхідно бути уважним при підтримці старих версій web-браузерів.

2.2.5. Вибір програмного забезпечення для публікації на сервер

Хостинг Heroku – це відмінний майданчик для розміщення проекту на сервер. Більшість функцій та можливостей в Heroku безкоштовні, але продовження часу роботи додатку або налаштування його специфікацій можна отримати за плату.

Heroku є хмарною платформою, що використовує принцип PaaS («платформа як послуга») та підтримує багато мов програмування. На початку своєї роботи Heroku підтримував Ruby, але наразі до цього списку додались мови Node.js, Java, Python, Clojure, Scala і PHP. Сервери Heroku для своєї роботи користуються системами Debian та Ubuntu [20].

GitHub і Dropbox – це одні з найпопулярніших сервісів для зберігання вихідного коду програм для їх запуску на сервері. Для коректної публікації застосунку на сервер бажано використовувати Heroku Toolbelt, так як він дозволяє працювати в командному рядку для створення проектів, їх конфігурації, завантаження коду та перегляду журналу логів.

GitHub та Dropbox використовують частіше для завантаження коду на сервер Heroku, але інструментів для керування додатком вони не мають. Кожен з цих сервісів по різному створює відбиток коду: Github послуговується комітами для запису вихідного коду, а Dropbox для такого відбитку використовує лише web-інтерфейс.

Heroku Toolbelt можна встановити з спеціальної сторінки https://devcenter.heroku.com/articles/heroku-cli на настільний комп'ютер.

Git – це система контролю версій, що створює копію коду проєкту після успішно втіленого етапу розробки. Кожна така копія називається комітом [19].

Ця система була створена в 2005 році Лінусом Торвальдсом, засновником Linux, аби інші програмісти мали можливість внеску в ядро Linux. Система Git набула популярності через свою швидкість роботи, простий дизайн, повну децентралізацією та можливість сумісної роботи над великими проектами.

За допомогою Git можна працювати над новим функціоналом проєкту без страху зробити помилки в коді програми. Викладаючи проєкт на сервер, обов'язково необхідно впевнитись, що проєкт робочий.

Перевагами системи контролю версій Git є :

- Безкоштовний open-source. Можна переглядати будт-який вихідни код, зберігати будь-який вихідний код та редагувати його;
- Невеликий та швидкий. Всі операції виконуються на комп'ютері, а репозиторій зберігається у невеликому файлі;
- Резервне копіювання. Git зберігає останні версії проєктів дуже ефективно;
- Просте розгалуження. Інші системи котролю версій теж використовують дерева, але там процес їх створення – це складний і трудомісткий процес. В Git створення таких дерев виконується без затримок;
- Для запису даних Git використовує онлайн-хостинг GitHub, що має всі функції системи контролю версій і навіть більше.

Створений в Github Git-репозиторій буде доступний за допомогою командної строки Git та Git-команд.

Висновки до розділу 2

Архітектура web-застосунків – складний та багатоетапний процес. Умовно будь-який сайт можна розділити на дві частини: backend та frontend. Backend відповідає за роботу з базами даних, обробку даних, їх запис та видобування. Frontend – це клієнтська частина сайту, він визначає, яким буде інтерфейс користувача.

Також існують додатки, які містять спеціальні технології, що дозволяють працювати з ними навіть без підключення до Інтернету, їх називають прогресивними застосунками (PWA). Також є односторінкові додатки (SPA), що не перезавантажують сторінки для виведення даних, а одразу виводять їх на екран.

Було розглянуто відмінність між статичними та динамічними сторінками та визначено, що робота з динамічними сторінками набагато спрощує роботу програміста та позбавляє рутинної роботи.

Протягом аналізу застосованих інструментів, було обрано мову програмування для майбутнього застосунку – Python. Це об'єктно-орієнтована мова з багатьма вбудованими модулями та простим синтаксисом. Для побудови застосунку буде використовуватись модуль Django – фреймворк для створення web-застосунків. Він відрізняється своєю повнотою, безпекою та широким функціоналом. Django надає інструменти для створення баз даних, але було визначено, що в застосунку буде використовуватись база даних на основі мови SQL.

Також для створення сторінок застосунку буде використано мову гіпертекстової розмітки HTML. Для дизайну застосунку обрано фреймворк Bootstrap: в ньому міститься безліч шаблонів, які можна комбінувати між собою. Для публікації застосунку на сервер оберемо хостинг Heroku, а для полегшення роботи з ним – систему контролю версій Git.

РОЗДІЛ 3. РОЗРОБКА WEB-ЗАСТОСУНКУ

Як було зазначено в попередніх розділах, за лаштунками сучасних веб-додатків прихована складні інформаційні процеси. Сьогодні майже кожен веб-додаток має три складові: серверну, клієнтську сторони та базу даних. У Python дя створення web-застосунку використовують фреймворк Django. Користуючись Django та іншими фреймворками, створено додаток «Щоденник» для запису предметів, завдань до них та відслідковування дат їх виконання.

Спочатку буде створено специфікацію до цього проєкту, визначено модель даних, з якими буде працювати застосунок. Через систему адміністрування Django введено початкові дані. З використанням шаблонів та видів можна побудувати сторінки сайту.

Django відповідає на запити користувача та полегшує процес читання та запису даних до БД, допомагає адмініструвати користувачів тощо. Пізніше застосунок буде викладений на робочий сервер, аби всі бажаючі могли ним користуватися.

3.1. Створення проєкту та встановлення необхідних модулів

Аби працювати з Django необхідно створити віртуальне середовище. Віртуальне середовище – це простір системи, куди можна встановити бібліотеки ізольовано від решти бібліотек Python. Відокремлення Django від інших бібліотек знадобиться на етапі публікації на сервер.

Віртуальне середовище створюється в окремій папці за допомоги команди в терміналі (рис.3.1).

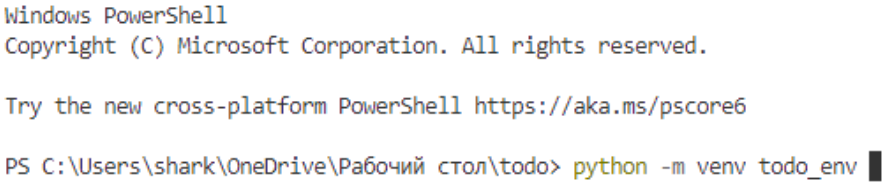

Рис. 3.1. Створення віртуального середовища

Активація віртуального середовища відбувається за допомоги команди todo_env\Scripts\activate. Таким чином активується файл activate.bat (рис.3.2).

```
C:\Users\shark\OneDrive\Рабочий стол\todo>todo_env\Scripts\activate

(todo_env) C:\Users\shark\OneDrive\Рабочий стол\todo>
```

Рис. 3.2. Активація віртуального середовища

Аби припинити роботу у віртуальному середовищі пишуть команду deactivate.

Активувавши віртуальне середовище, неохідно встановити наступні пакунки:

- Django;
- Django-bootstrap4 для створення дизайну застосунку;
- Psycopg 2.7.* для користування базою даних, якою послуговується хостинг Heroku;
- Django-heroku виконує майже всю конфігурацію застосунку для запуску на сервері Heroku – це контроль над базою даних, збереження статичних файлів. Статичні файли містять правила оформлення та файли JavaScript.
- Gunicorn - це сервер, що зможе надати доступ до застосунків на сервері Heroku.

Встановлення модулів у Python відбувається за допомоги команди pip install <package>. Приклад встановлення пакунку Django показано на рис.3.3. Так як, в

створюваному проєкті він був встановлений попередньо, то виведе повідомлення «Requirements already satisfied». Встановлення інших модулів є в Додатку А.

```
(todo_env) C:\Users\shark\OneDrive\Рабочий стол\todo>pip install django
Requirement already satisfied: django in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages (4.1.2)
Requirement already satisfied: sqlparse>=0.2.2 in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages (
from django) (0.4.3)
Requirement already satisfied: tzdata in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages (from djan
go) (2022.5)
Requirement already satisfied: asgiref<4,>=3.5.2 in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages
 (from django) (3.5.2)
WARNING: You are using pip version 22.0.4; however, version 22.3 is available.
You should consider upgrading via the 'C:\Users\shark\OneDrive\Рабочий стол\todo\todo_env\Scripts\python.exe -m pip inst
all --upgrade pip' command.
```

Рис. 3.3. Встановлення модуля Django

Також перед створенням бази даних та сторінок, необхідно створити проєкт у віртуальному середовищі – todos (рис.3.4).

```
(todo_env) C:\Users\shark\OneDrive\Рабочий стол\todo>django-admin startproject todo .

(todo_env) C:\Users\shark\OneDrive\Рабочий стол\todo>cd todo\

(todo_env) C:\Users\shark\OneDrive\Рабочий стол\todo\todo>dir
 Volume in drive C has no label.
 Volume Serial Number is E455-67C8

 Directory of C:\Users\shark\OneDrive\Рабочий стол\todo\todo

14.10.2022  19:21    <DIR>          .
14.10.2022  19:21    <DIR>          ..
14.10.2022  19:21               401 asgi.py
27.10.2022  21:43             3 414 settings.py
24.10.2022  22:02               858 urls.py
14.10.2022  19:21               401 wsgi.py
14.10.2022  19:21                 0 __init__.py
27.10.2022  21:57    <DIR>          __pycache__
               5 File(s)          5 074 bytes
               3 Dir(s)  140 142 088 192 bytes free
```

Рис. 3.4. Створення проєкту

Папка todo містить чотири файли, найважливіші з них – це setting.py, urls.py, wsgi.py. Файл setting.py контролює взаємодію Django з системою та керує проєктом. В цьому файлі буде додано назви додаткових застосунків, налаштування безпеки тощо. Файл urls.py описує, що має повертати додаток у відповідь на певні запити браузера. Файл wsgi.py дозволяє керувати створеними файлами.

Проєкт Django складається з набору застосунків, що взаємодіють один з одним. Таких застосунків два – один з них буде визначати загальну логіку проєкту

та оброблювати інформацію основних сутностей бази дани., інший – необхідний для створення користувацьких облікових записів. Створити новий застосунок можна за допомоги команди startapp.

```
(todo_env) C:\Users\shark\OneDrive\Рабочий стол\todo>python manage.py startapp todos

 Directory of C:\Users\shark\OneDrive\Рабочий стол\todo\todos

14.10.2022  20:27    <DIR>          .
14.10.2022  20:27    <DIR>          ..
14.10.2022  19:43               154 admin.py
14.10.2022  19:22               148 apps.py
23.10.2022  13:01               696 forms.py
27.10.2022  21:20    <DIR>          migrations
27.10.2022  21:19               798 models.py
14.10.2022  19:51    <DIR>          templates
14.10.2022  19:22                63 tests.py
19.10.2022  10:49               754 urls.py
27.10.2022  22:34             4 328 views.py
14.10.2022  19:22                 0 __init__.py
27.10.2022  22:34    <DIR>          __pycache__
               8 File(s)          6 941 bytes
               5 Dir(s)  139 955 855 360 bytes free
```

Рис. 3.5. Створення нового застосунку

Найважливішими файлами є models.py, admin.py, views.py.

3.2. Проектування бази даних

Перед визначенням сутностей бази даних та їх зв’язків, необхідно ініціалізувати її у віртуальному сеедовищі. В модулі Django за створення бази даних відповідає команда python manage.py migrate. Результати виконання команди на рис. 3.6.

```
 Directory of C:\Users\shark\OneDrive\Рабочий стол\todo

01.11.2022  13:35    <DIR>          .
01.11.2022  13:35    <DIR>          ..
01.11.2022  13:35           147 456 db.sqlite3
14.10.2022  19:21               682 manage.py
14.10.2022  19:21    <DIR>          todo
14.10.2022  20:27    <DIR>          todos
14.10.2022  19:20    <DIR>          todo_env
24.10.2022  22:02    <DIR>          users
               2 File(s)        148 138 bytes
               6 Dir(s)  139 886 841 856 bytes free
```

Рис. 3.6. Створення бази даних

Міграція – це зміна в структурі бази даних. Якщо запустити команду вперше, то можна помітити, що структура бази даних віповідає поточному стану проєкту. Якщо вперше запустити цю команду, то Django створить нову базу даних за допомоги SQLite. Детальніше про MySQL та загалом SQL було згадано у розділі 2. Так як в створюваному проєкті база даних вже готова, то команда дала віповідь, що міграції вже застосовані.

Команда dir вивела на екран, що в проєкті з'явився новий файл db.sqlite3. Це база даних, яка використовує лише один файл, так як дозволяє уникнути проблем з адмініструванням.

Обміркувавши цілі та задачі проєкту, тепер на їх основі необхідно створити базу даних з сутностями, які будуть повністю задовольняти вимогам. На рис.3.7. запропонована діаграма класів проєктованої бази даних.

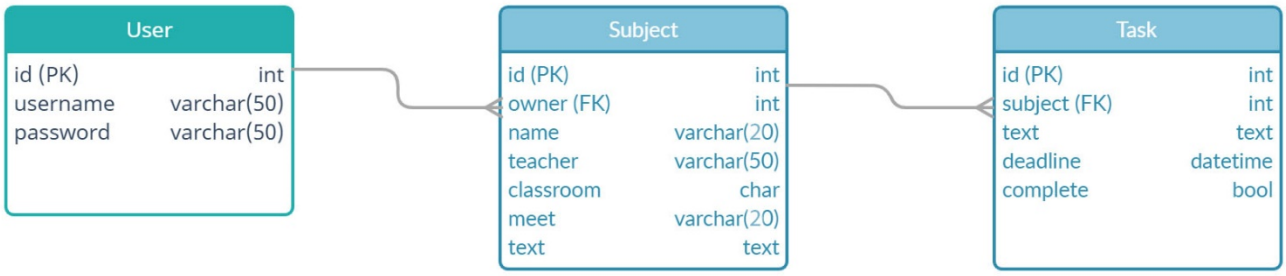

Рис.3.7. Діаграма класів

Так як проєкт буде містити облікові записи, потрібно створити сутність Користувач (User), аби в подальшому була можливість зробити зв'язок між даними користувача та його обліковим записом. При цьому, користувач може мати дві ролі: звичайний користувач або адміністратор. Користувач може виконувати наступні дії:

- реєструватись в застосунку, входити та виходити зі свого облікового запису;

- додавати нові предмети та інформацію про нього: ім’я викладача, кабінет, в якому буде проходити заняття, посилання на Google Meet та примітки;
- додавати завдання до кожного предмету та інформацію про нього: текст завдання, дату виконання та вкзати, виконане воно чи ні;
- редагувати та видаляти предмети та інформацію про них;
- редагувати та видаляти інформацію про завдання ;

Django дозволяє створити адміністратора (або суперкористувача), тобто такого користувача, який має всі можливості управління додатком. Можиливо налаштувати застосунок так, аби відвідувачі сайту могли лише читати відкриту інформацію, зарєєстровані користувачі можуть читати лише власні записи. Аби ефективно працювати з даними, адміністратор повинен мати до них повний доступ. Створення адміністратора відбувається так, як на рис.3.8.

```
(todo_env) C:\Users\shark\OneDrive\Рабочий стол\todo>python manage.py createsuperuser
Username:
```

Рис. 3.8. Створення адміністратора

Деяку інформацію від адміністратора Django приховує. Наприклад, в панелі управління неможливо побачити пароль від облікового запису користувача, замість цього видає так званий геш. При авторизації система гешує ввід паролю, а потім порівнює його з тим гешем, який вже збережений. Якщо геші збігаються, це означає, що користувач ввів правильний пароль. При правильному налаштуванні сайту, навіть якщо сторонній користувач увійде в обліковий запис адміна, він не зможе розшифрувати геші паролів.

На відміну від користувача, адміністратор має більший спектр можливостей. Адміністрування в таких проєктах відбувається не безпосередньо в проєктованому додатку, а через спеціальну панель Django Administration (рис.3.9).

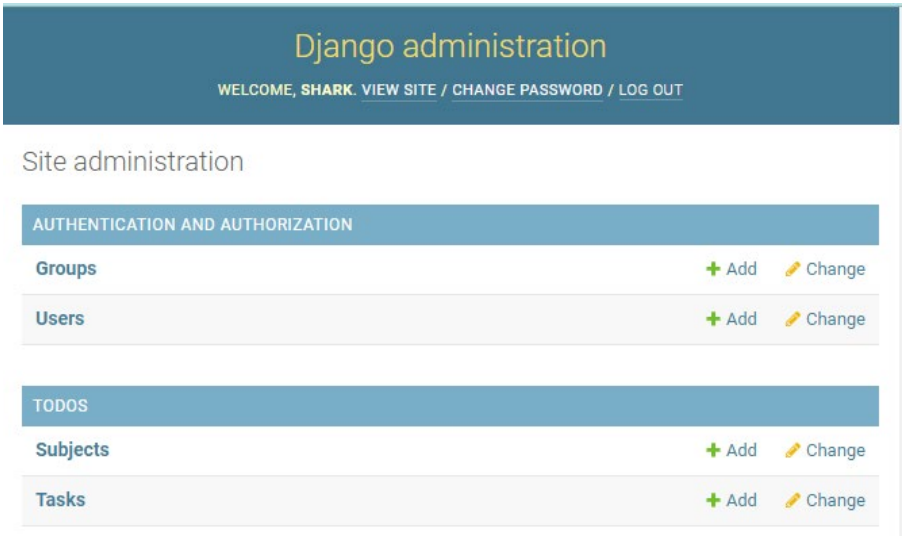

Рис. 3.9. Панель управління адміністратора

Django Адміністратор може виконувати наступні дії, окрім тих, що і Користувач:

- змінювати інформацію про користувачів, їх електронну пошту, імена та прізвища;
- надати користувачу привілеї, наприклад, зробити модератором або новим адміністратором;
- обирати, до яких функцій може мати доступ користувач;
- видалити його запис у базі даних;
- додавати нові предмети та завдання, використовуючи панель управління.

На основі досліджених можливостей користувача та адміністратора, створено діаграму варіантів використання (рис.3.10).

Рис.3.10. Діаграма варіантів використання

Створення сутності Користувач є автоматичним, так як за замовчуванням зберігається у базі даних Django. Для цієї сутності модель створювати не потрібно.

Кожен користувач може додавати у базу даних власний список предметів. Сутність Предмет (Subject) повинна містити наступні поля:

- назва предмету;
- ім’я, прізвище та по-батькові викладача;
- номер кабінету, якщо заняття проводиться в аудиторії;
- посилання на відеоконференцію у Google Meet;
- додаткові примітки.

Програмно така сутність створюється як клас (Subject), що наслідує батьківський клас Model – він визначає базові функції будь-якої моделі. До цього класу буде додано name, teacher, classroom, meet, text та owner (рис.3.11).

```python
class Subject(models.Model):
    name = models.CharField(max_length = 200)
    teacher = models.CharField(max_length = 500)
    classroom = models.CharField(max_length = 200)
    meet = models.CharField(max_length = 500)
    text = models.TextField()
    owner = models.ForeignKey(User, on_delete = models.CASCADE)

    def __str__(self):
        return self.name
```

Рис.3.11. Реалізація моделі «Предмет»

Атрибут name – це об’єкт класу CharField, тобто такий елемент, який складається з символьних значень. Такий клас можна використовувати для коротких записів: назва міста, імені або заголовка. Застосувавши такий клас, необхідно в його конструкторі визначити максимально допустиму кількість символів. Для поля з іменем викладача максимально допустимою кількістю символів буде 500, для всіх інших полів – 200.

Окрім цього, в моделі застосовано поле типу TextField. В такому полі можна записати необмежену кількість символів .

Також до поля owner застосовано клас ForeignKey (зовнішній ключ). Зовнішні ключі – це поняття, властиве базам даних. За допомоги зовнішніх ключів можна зв’язати декілька моделей один з одним. У випадку проєктованого застосунку, кожен користувач може мати декілька предметів, але всі створювані користувачем предмети повинні бути доступні лише в одному обліковому записі. Такий вид зв’язку називається «один-до-багатьох». В конструкторі класу ForeignKey вказано, з яким класом буде встановлено такий зв’язок, а атрибут on_delete визначає, що у випадку видалення облікового запису користувача, всі його дані теж втрачаються.

Метод __str__() створює просте представлення моделі, повертаючи рядком назву предмету.

Аби користувач міг записувати домашнє завдання з кожного предмету, необхідно створити модель, яка б могла зберігати ці записи у базі даних. Ця модель, так само, як і модель Subject і User, буде пов’язана з Subject зв’язком

«один-до-багатьох». Це пояснюється тим, що один предмет може мати безліч завдань, але одне завдання не може стосуватися декількох предметів. Приклад реалізації моделі Task показано на рис.3.12.

```
class Task(models.Model):
    subject = models.ForeignKey(Subject, on_delete = models.CASCADE)
    text = models.TextField()
    deadline = models.DateTimeField()
    complete = models.BooleanField(default = False)

    def __str__(self):
        return f"{self.text[:50]}..."
```

Рис. 3.12. Реалізація моделі «Завдання»

Так само, як і в моделі Subject, модель Task вставлює зовнішній ключ за допомоги класу ForeignKey. Атрибут on_delete = models.CASCADE визначає, що при видаленні предмету, завдання, що його стосувались, також видаляться.

Атрибут deadline є екземпляром класу DateTimeField(). Цей клас створює елемент даних для запису дати та часу.

Об’єкт complete належить класу BooleanField(). Він створює поле для вибору логічних значень: True або False. Якщо завдання було виконано, то змінна complete приймає значення True, інакше False. За замовчуванням атрибуту надано значення False.

Метод __str__() повертає рядкове значення перших 50 символів тексту завдання.

Після визначення моделей застосунку, необхідно помістити їх у файл налаштувань загального проєкту setting.py, а потім зафіксувати зміни у базі даних за допомоги команди makemigrations (рис.3.13).

```
INSTALLED_APPS = [
    'todos',
    'users',
    'bootstrap4',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

```
(todo_env) C:\Users\shark\OneDrive\Рабочий стол\todo>python manage.py makemigrations todos
No changes detected in app 'todos'
```

Рис. 3.13. Застосування змін бази даних

Аби адміністратор тепер мав доступ до цих моделей та міг їх переглядати, необхідно зареєструвати їх у інтерфейсі адміністратора так, як показано на рис.3.14.

```
from django.contrib import admin
from .models import Subject, Task
# Register your models here.
admin.site.register(Subject)
admin.site.register(Task)
```

Рис.3.14. Реєстрація моделей для адміністратора

3.3. Створення сторінок

Загалом, створення сторінок у Django має три етапи. По-перше, необхідно визначити URL, за допомогою якого будуть звертатись до певної сторінки. Для створення URL використовують регулярні вирази, що описують структуру URL. По-друге, необхідно створити вид (view) - функцію, що буде видобувати та обробляти дані з бази даних. По-третє, створюється шаблон, який містить загальну структуру та виводить дані, оброблені видом.

3.3.1. Головна сторінка

Головна сторінка застосунку буде містити інформацію про завдання на сьогодні, на завтра та на найближчі 3 дні. Для того, аби створити таку сторінку, спочатку необхідно визначити URL, за яким будемо до неї звертатись.

URL домашньої сторінки – це базовий URL застосунку http://127.0.0.1:8000, і наразі за цим посиланням відкривається сторінка Django. Необхідно перенести базовий URL на головну сторінку.

Для цього змінимо список основних URL застосунку. Другим елементом списку urlpatterns при заході на домашню сторінку, буде викликатись модуль urls з застосунку todos (рис.3.15).

Сюди ж можна додати посилання на започаткований застосунок users, в якому пізніше буде створено сторінки авторизації та реєстрації.

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('users/', include('users.urls')),
    path("", include('todos.urls'))
]
```

Рис. 3.15. Основний файл urls.py

Необхідно створити ще один файл urls.py у застосунку todos. В ньому будуть визначатися URL головної сторінки та всіх подальших сторінок. Код, який потрібен для переходу до посилання головної сторінки показаний на рис.3.16.

```python
from django.urls import path

from . import views

app_name = 'todos'

urlpatterns = [
    path('', views.index, name = "index"),

]
```

Рис.3.16. Файл todos/urls.py

В цьому файлі імпортовано модуль views та функцію path з модуля django.urls. Змінна app_name дозволяє відрізняти цей файл urls.py від інших файлів з такою самою назвою у цьому проєкті. Змінна urlpatterns визначає список індивідуальних сторінок, до яких можна звернутися у застосунку.

Функція path() містить два аргументи: перший аргумент дозволяє правильно маршрутизувати поточний запит, другий описує, яку функцію необхідно викликати з файлу views.py, і третій – дає ім'я цьому регулярному виразу, аби в подальшому звертатися до нього в інших фрагментах коду.

Другим етапом є створення виду для цієї сторінки. Вид приймає інформацію за надісланим запитом, обробляє дані та генерує шаблон, який буде виводитись на екран.

У файлі views.py потрібно імпортувати функцію render() з модуля django.shortcuts, для того, щоб відображати відповідь на основі отриманого запиту. Запит URL має відповідати заданому регулярному виразу, тоді викликається функція index() , якому передається об’єкт запиту. Функція index() представлена на рис.3.17.

```python
def index(request):
    subjects = Subject.objects.filter(owner = request.user)
    tasks=[]
    for subject in subjects:
        task = subject.task_set.order_by('deadline')
        tasks.append(task)
    week = ['Понеділок', 'Вівторок', 'Середа', 'Четвер', 'П\'ятниця', 'Субота', 'Неділя']
    today = datetime.datetime.today()
    day = week[calendar.weekday(today.year, today.month, today.day)]
    tomorrow = today + datetime.timedelta(days =  1)
    tomorrow_day = week[calendar.weekday(tomorrow.year, tomorrow.month, tomorrow.day)]
    near_days = [today + datetime.timedelta(days=2), today + datetime.timedelta(days=3),
    context = {'near_days': near_days,
               'subjects':subjects,
               'tasks': tasks,
               'today': today,
               'day': day,
               'tomorrow_day': tomorrow_day,
               'tomorrow': tomorrow}
    return render(request, 'todos/index.html', context)
```

Рис. 3.17. Функція виду головної сторінки

Функція index() приймає запит від користувача та зберігає в змінну subjects всі предмети, які належать саме цьому користувачеві. Перебираючи всі предмети за допомоги циклу for, видобуваємо всі завдання, що стосуються всіх предметів, сортуючи їх по кінцевій даті виконання та додаємо до списку tasks. Тобто список tasks буде зберігати інформацію про всі завдання з усіх предметів. Щоб дізнатися сьогоднішню дату, застосуємо функцію today() з модуля datetime. Datetime – це уставний модуль в Python, що дозволяє спросити роботу з часом та датою. Проте, школярам (особливо молодшим) іноді буває складно орієнтуватися по датах, тому, аби спростити сприйняття кінцевої дати виконання, створено список днів тижня. Аби дізнатися сьогоднішній день тижня застосовано функцію weekday з модуля calendar.

Модуль datetime та calendar дуже схожі, але в той же час виконують зовсім різні завдання. Datetime застосовується у випадку, коли, наприклад, необхідно видобути системні дату або час чи дізнатись, яка дата буде через кілька днів (як це, наприклад, відбувається зі змінними tomorrow та near_days). В цей час модуль calendar працює тільки з датами, місяцями, днями тижня тощо. Знаючи точну дату, модуль calendar може передбачити, якому дню тижня буде належати ця дата або чи існує вона в календарі у цьому році (як це відбувається з 29 лютого, ця дата з’являється тільки у високосному році).

Тому при створенні змінних tomorrow та near_days використовувалась функція timedelta(), аби передбачити, яка дата буде завтра або найближчі три дні. А якому дню тижня буде належати завтрашній день, дізнаємось за допомоги знову ж функції weekday().

Далі створюється словник context – він існує лише для того, аби передати у шаблон назви змінних, що зберігають корисну інформацію з виду.

У функції render() записується три параметри: перший – на який запит було дано відповідь, другий – яка сторінка шаблону при цьому має бути виведена, і третій – які значення необхідні для виведення результату у шаблоні.

Шаблон – це сформована сторінка, призначена для виведення результату обробки даних видом. Зазвичай шаблони створюють за допомоги HTML (Hyper Text Markup Language), а також додатково використують шаблонні теги та шаблонні змінні. Шаблонний тег – це спеціальна команда, що записується у {%%} та має на меті вивести дані на сторінку у певному форматі. Наприклад, є шаблонні теги {%for%}, {% if %}, {% block content %} тощо. Шаблонні змінні – це і є ті змінні, що були передані в шаблон за допомоги словника context.

Шаблон головної сторінки має велику кількість коду, тому розглянемо лише його частину на рис.3.18.

```
<p>
    <h4> {{day}}, {{today|date:'M d, Y'}} треба виконати: </h4>
    <ul>
        {% for task in tasks %}
        {% if task.deadline.year == today.year and task.deadline.month == today.month and task.deadline.day
        <li>
            {% if task.complete %}
            <s><p> {{task.deadline|date:'H:i'}} <br>  {{task.subject}} - {{task.text}} </p></s>
            {% else %}
            <p> {{task.deadline|date:'H:i'}} <br> {{task.subject}} - {{task.text}}  </p>
            {% endif %}
            <p>
                <a href="{% url 'todos:edit_task' task.id%}">Редагувати завдання</a>
            </p>
            <p>
                <a href="{% url 'todos:delete_task' task.id %}">Видалити завдання</a>
            </p>
        </li>
        {% endif %}
        {% endfor %}
        <li>Завдань більше немає</li>
    </ul>
</p>
```

Рис.3.18. Шаблон головної сторінки

Як видно, на початку шаблону застосовано тег <h4> для створення заголовку з сьогоднішньою датою та днем тижня. До речі, тут використовується так званий шаблонний фільтр date (він записаний поряд зі змінною today), тобто функція, що корегує значення шаблонної змінної та виводить дані у певному форматі. Далі необхідно перебрати всі завдання з усіх предметів за допомоги двох циклів for: перший видобуває всі завдання з конкретного предмету та зберігає у змінну task, а другий – перебирає всі завдання по одному для цього предмету. Перевіряється, чи завдання належить сьогоднішній даті та чи воно виконане. Якщо завдання виконане, то його текст та дата будуть відображатись у шаблоні як перекреслені, інакше ні. Тут також додано посилання на сторінки редагування та видалення, але детальніше їх розглянемо в наступних пунктах. Загалом, головна сторінка буде мати наступний вигляд (рис.3.19).

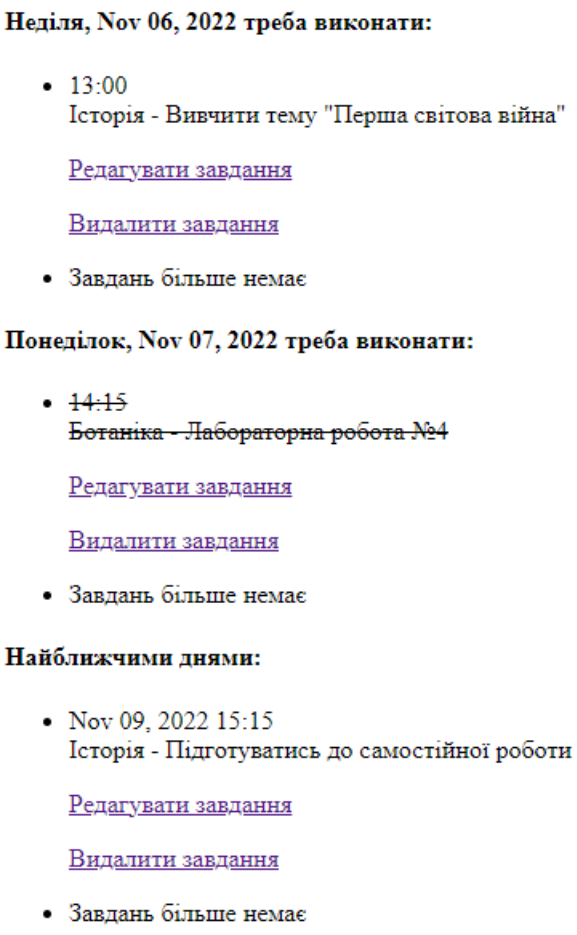

Рис.3.19. Головна сторінка застосунку

3.3.2. Побудова подальших сторінок

Наступні сторінки, як і головна, будуть будуватись за допомоги визначення URL, створення видів та шаблонів. Необхідно створити базовий шаблон, від якого будуть наслідуватися всі інші. Також буде створено сторінки для перегляду усіх предметів та завдань до них.

Побудова батьківського шаблону – один з важливих етапів розробки, так як всі подальші сторінки повинні мати навігаційну панель для зручного переміщення по додатку. Такий підхід дозволяє не створювати безліч однакових сторінок, а просто виводити на екран вже існуючі результати. Базова сторінка буде мати такий вигляд, як на рис.3.20. Код базового шаблону надано в додатку Б.

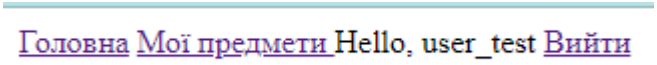

Рис. 3.20. Створення батьківського шаблону

Як бачимо, розділи в батьківському шаблоні сформовані за допомоги тегу <a>, тобто вони є посиланнями. Тут замість стандартного регулярного виразу URL викликається шаблонний тег {% url %}, що в свою чергу викликає функцію index().

Те саме відбувається, коли переходимо на сторінку з предметами. Тут необхідно створити новий регулярний вираз у файлі urls.py. Також можна створити регулярний вираз для відображення завдань. Ці регулярні вирази представлені на рис.3.21.

```
urlpatterns = [
    path('', views.index, name = "index"),
    path('subjects', views.subjects, name = "subjects"),
    path('subjects/<int:subject_id>', views.subject, name = "subject"),
```

Рис. 3.21. Регулярні вирази для сторінок предметів та завдань

В регулярному виразі для виведення сторінки з завданнями використовується рядок subjects/<int:subject_id>. Мається на увазі, що при переході на сторінку з предметами, кожен предмет матиме власний ідентифікатор. Для того, аби відкрити сторінку з завданнями до цього предмету, необхідно знати його порядковий номер. Тут і використовується рядок <int:subject_id>, що перенаправляє на сторінку конкретного завдання.

Створення виду для сторінки предметів та завдань має такий вигляд, як на рис.3.22.

```
def subjects(request):
    subjects = Subject.objects.filter(owner = request.user).order_by('name')
    context = {'subjects': subjects}
    return render(request, 'todos/subjects.html', context)

def subject(request, subject_id):
    subject = Subject.objects.get(id = subject_id)
    if subject.owner != request.user:
        raise Http404
    tasks = subject.task_set.order_by('deadline')
    context = {'subject': subject, 'tasks': tasks}
    return render(request, 'todos/subject.html', context)
```

Рис. 3.22. Функції видів subjects() та subject()

Функція subjects() призначена для видобування всіх існуючих предметів для певного користувача, при чому ці предмети маються бути відсортовані за ім’ям. У функції subject() видобуваються всі завдання для предмету з певним id та всі ці завдання записуються у змінну tasks, відсортовані по даті виконання.

Шаблони для сторінок предметів та завдань мають схожу структуру, як і головна сторінка, окрім того, що дані там виводяться у вигляді списку (рис.3.23, рис.3.24). Переглянути повний код цих шаблонів можна в Додатку Б.

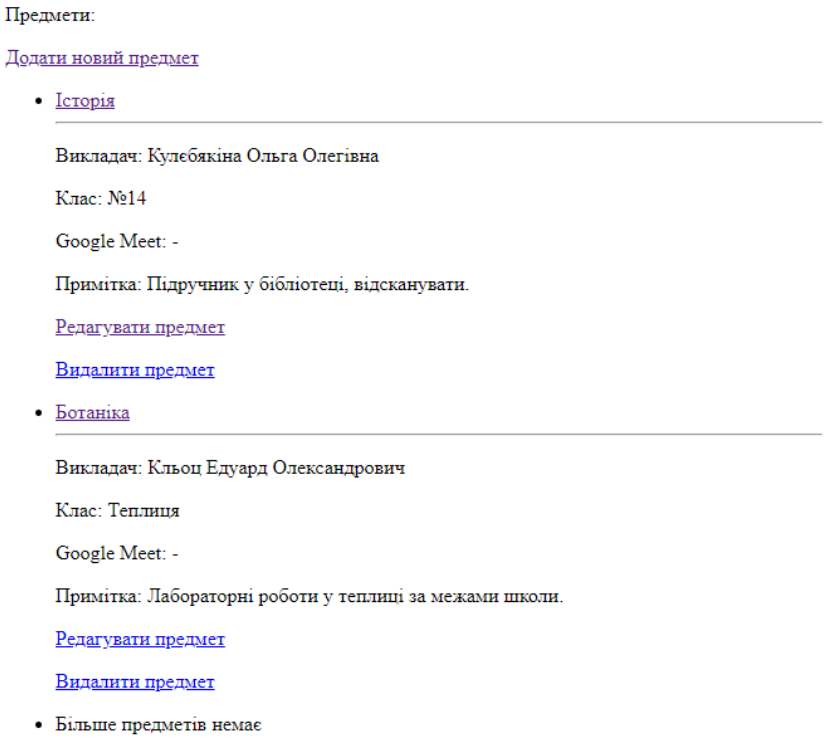

Рис. 3.23. Сторінка предметів

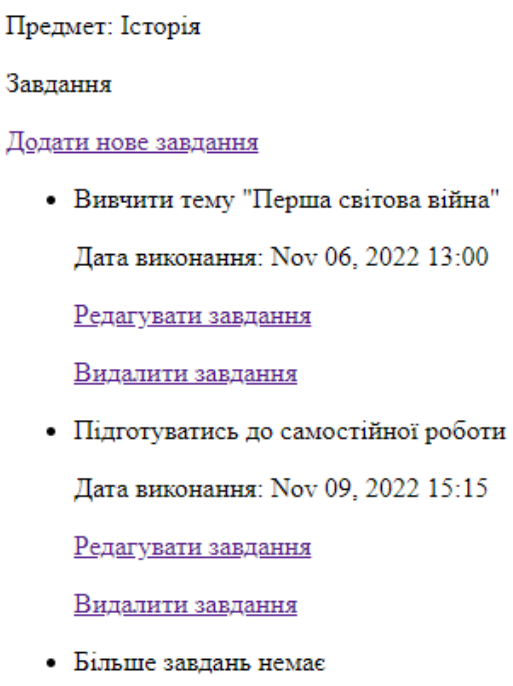

Рис. 3.24. Сторінка завдань

3.3.3. Користувацький ввід даних

Користувацький ввід даних передбачає, що зареєстрований користувач буде мати змогу додавати, редагувати та видаляти завдання в самому додатку. Сторінка, що містить форму, будується за схожим алгоритмом, як і сторінки завдань та предметів. Відмінним буде лише наявість модуля forms.py, що і буде містити власне форми (рис.3.25).

```
from django import forms
from .models import Subject, Task

class SubjectForm(forms.ModelForm):
    class Meta:
        model = Subject
        fields = ['name', 'teacher', 'classroom', 'meet', 'text']
        labels = {'name': 'Назва предмету', 'teacher': 'Викладач', 'classroom':'Клас', 'meet': 'Google Meet', 'text' : ""}

class TaskForm(forms.ModelForm):
    class Meta:
        model =  Task
        fields = ['text', 'deadline', 'complete']
        labels = {'text': 'Завдання', 'deadline':'Дата виконання', 'complete' : "Виконано"}
        widgets = {'deadline': forms.DateTimeInput(attrs = {'type': 'datetime-local'})}
```

Рис. 3.25. Файл forms.py

Форма – це будь-яка сторінка або частина сторінки, що дозволяє користувачеві вводити та відправляти дані. Перед записом даних з форми до бази даних, необхідно перевірити правильність вводу, потім обробити та записати ці дані. Більшість рутинних задач при цьому Django виконує самостійно.

Один з найпростійших способів створити форму - це скористатись класом ModelForm. Від нього власне і наслідуються класи SubjectForm та TaskForm. Найпростіша версія ModelForm містить вкладений клас Meta. Цей клас описує, на основі якої моделі буде створено форму, які імена будуть мати поля форми, надписи для кожного них, а також, за необхідності, перевизначити уставні поля форми за допомоги атрибуту widgets. У HTML віджетами називають текстові поля, спадний список, перемикачі тощо. Наприклад, у класі TaskForm змінено уставний віджет для поля deadline, так як за замовчуванням цьому віджету створювалось

однорядкове текстове поле. Замість цього поля встановлено віджет DateTimeInput, що дозволяє вводити дані за допомогою календаря.

Для виведення сторінок створення, редагування та видалення завдань потрібно створити нові регулярні вирази URL (рис.3.26).

```
path('new_subject/', views.new_subject, name = 'new_subject'),
path('new_task/<int:subject_id>', views.new_task, name = 'new_task'),
path('edit_subject/<int:subject_id>', views.edit_subject, name = "edit_subject"),
path('edit_task/<int:task_id>', views.edit_task, name='edit_task'),
path('delete_subject/<int:subject_id>',views.delete_subject, name = 'delete_subject'),
path('delete_task/<int:task_id>', views.delete_task, name = "delete_task")
```

Рис. 3.26. URL сторінок форм

Для прикладу розглянемо сторінки для створення нового завдання, його редагування та видалення. Повний лістинг коду для інших сторінок форм буде надано в Додатку Б. Функція виду new_task() показана на рис.3.27.

```
def new_task(request, subject_id):
    subject = Subject.objects.get(id = subject_id)
    if request.method !='POST':
        form = TaskForm()
    else:
        form = TaskForm(data=request.POST)
        if form.is_valid():
            new_task = form.save(commit=False)
            new_task.subject = subject
            new_task.save()
            return redirect('todos:subject', subject_id = subject_id)
    context = {'subject': subject, 'form': form}
    return render( request, 'todos/new_task.html', context)
```

Рис. 3.27. Функція new_task()

Ця функція має обробляти декілька випадків: коли користувач вперше потрапляє на цю сторінку, він побачить пусту форму, при цьому браузер надсилає GET-запит. Щойно користувач заповнив та відправив форму, браузер надішле POST-запит. Також необхідно перевірити, чи дані у формі коректні, якщо так, то спочатку присвоюємо значення предмету для атрибуту subject, а потім зберагіємо

дані з форми у базу даних та перенаправляємо користувача на сторінку предмету за допомоги функції redirect().

Сторінка для створення нового завдання має вигляд простої форми, яку часто використовують у HTML. Для захисту даних користувача, що вводить дані у форми, використовуєється шаблонний тег {% csrf_token%}. Цей тег має завадити атакам, що спрямовані на використання форми для незаконного доступу до сервера. Такий вид атаки називається міжсайтове підроблення запиту. Вигляд сторінки для додавання завдань показано на рис.3.28.

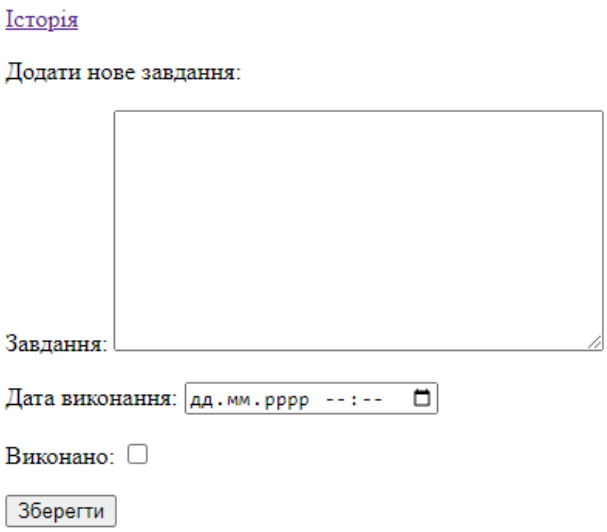

Рис. 3.28. Сторінка створення завдання

Форма редагування дуже схожа на форму створення нового завдання, окрім того, що у ній поля вже будуть заповнені даними. Для того, аби заповнити форму вже снуючими даними при створенні об’єкту класу TaskForm використовують вираз instance = task. Переглянути код функції цього виду можна у файлі views.py Додатку Б. Видалення даних за бази даних відбувається за допомоги функції delete(), що остаточно видаляє дані з БД (рис.3.29).

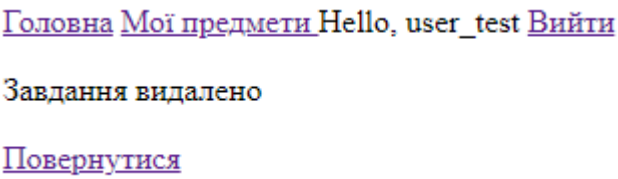

Рис. 3.29. Видалення завдань

Шаблон з повідомлення про видалення завдання виводить просте повідомлення та має посилання на сторінку завдань.

3.3.4. Користувацькі облікові записи

Створення користувацьких облікових записів буде складатись з трьох основних сторінок:

- сторінка реєстрації;
- сторінка авторизації;
- сторінка виходу з облікового запису.

Для створення сторінок авторизації та виходу з облікового запису необов'язково створювати функції виду. Справа в тому, що в Django вже існують уставні URL для авторизації та виходу з облікового запису (http://127.0.0.1:8000/users/login/ та http://127.0.0.1:8000/users/logout/ відповідно). Тобто Django зчитує ці URL і викликає уставні функції login() або logout(). Проте, все одно необхідно створити шаблони для цих сторінок. Їх структура схожа з іншими сторінками форм. Тому повний код шаблонів буде надано у Додатку Б. Власне, сторінка автентифікації виглядає, як на рис.3.30.

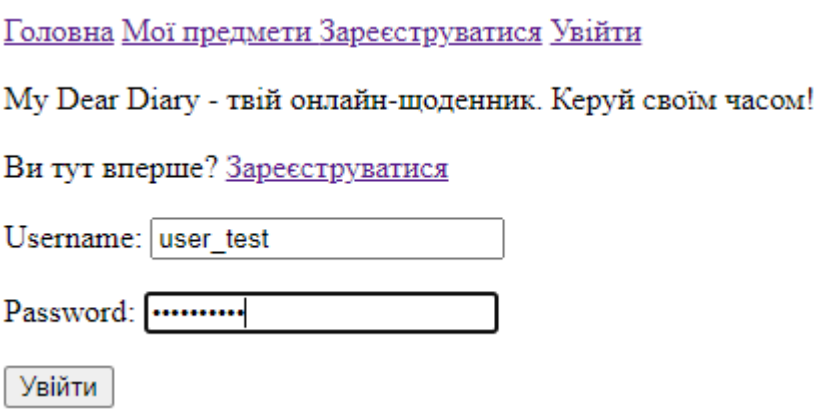

Рис. 3.30. Сторінка автенфікації

Сторінка виходу з облікового запису показана на рис.3.31.

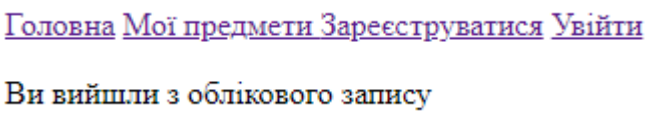

Рис. 3.31. Сторінка виходу з облікового запису

На жаль, у Django не існує уставного виду для реєстрації користувачів, тому необідно створити такий вид. Його мета – створити форму для заповнення даних, перевірити правильність введених даних та увійти в створений обліковий запис. Створення такого виду схоже на створення виду для сторінок форм для предметів та завдань, тому повний код буде надано в додатку Б. Основна відмінність – наявність класу UserCreationForm() для створення стандартної форми реєстрації та використання функції login(), що дозволяє користувачеві одразу увійти в обліковий запис. Вигляд сторінки реєстрації показано на рис.3.32.

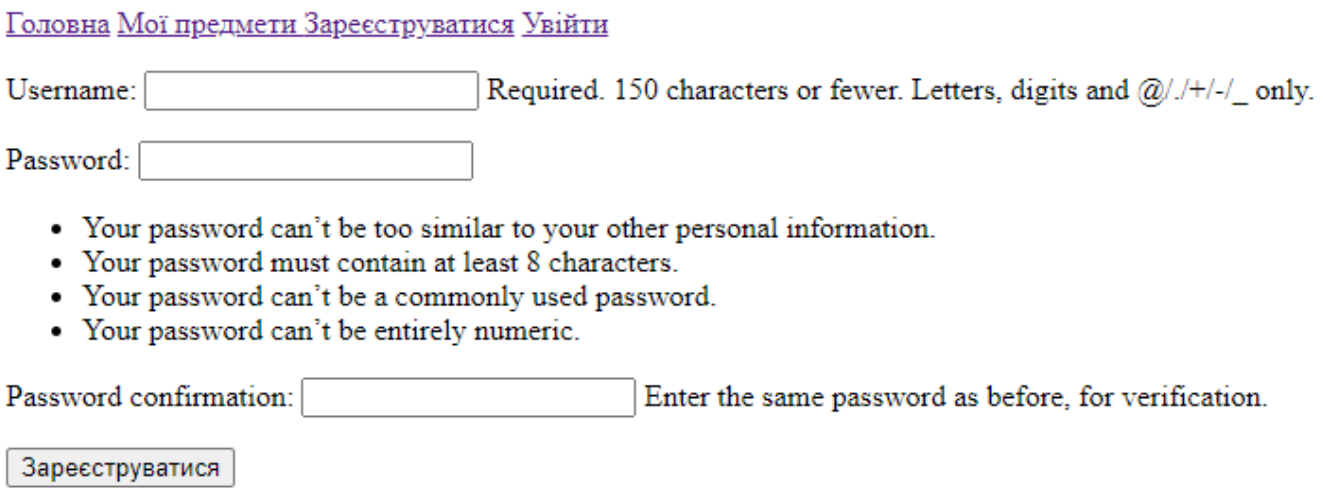

Рис.3.32. Сторінка реєстрації

Система доволяє створювати одному користувачеві безліч облікових записів. Деякі системи змушують користувачів підтверджувати свою особистість для подальшого користування, що дозволяє оптимізувати роботу бази даних та попереджують створенню спаму.

3.4. Дизайн застосунку

Звичайно, будь-який застосунок корисний тоді, коли він добре функціонує, але без належного дизайну, користувачам буде незручно працювати з додатком. Загалом, не є важливим занадто яскравий та барвистий дизайн. Можна скористатись вже готовими інструментами, що нададуть мінімалістичний та приємний дизайн.

Використання фреймворку Bootstrap спрощує процес дизайну застосунку, мінімізуючи витрати часу на розробку CSS та JavaScript файлів. Цей фреймворк містить безліч шаблонів для різних проєктів. В створюваному застосунку буде використано шаблон під назвою Navbar static, що містить просту навігаційну стрічку та контейнер для вмісту сторінки.

Створення приємного дизайну застосунку можна почати з навігаційного рядку. Необхідно внести зміни у файл base.html. Сюди завантажується колекція шаблоних тегів, що доступна Django-bootstrap4. Сторінка HTML зазвичай

складається з двох частин: шапка сторінки та її тіло. У шапки сторінки треба додати всі файли зі стилями Bootstrap, використовуючи шаблонний тег {% bootstrap_css %}. За наявність у стрінки інтерактивної поведінки відповідає тег {% bootstrap_javascript jquery = 'full' %}. Код для створення навігаційного рядку наведено на рис.3.33.

```
<nav class="navbar navbar-expand-md navbar-light bf-light mb-4 border">
    <a class="navbar-brand" href="{% url 'todos:index' %}">Головна</a>
    <div class="collappse navbar-collapse" id="navbarCollapse">
        <ul class="navbar-nav mr-auto">
            <li class="nav-item">
                <a class = "nav-link" href="{% url 'todos:subjects' %}"> Мої предмети </a>
            </li>
        </ul>
        <ul class="navbar-nav ml-auto">
            {% if user.is_authenticated %}
            <li class="nav-item">
                <span class="navbar-text">Hello, {{ user.username }}!</span>
            </li>
            <li class="nav-item">
                <a class = "nav-link" href="{% url 'users:log_out' %}">Вийти</a>
            </li>
            {% else %}
            <li class="nav-item">
                <a class = "nav-link" href="{% url 'users:register' %}">Зареєструватися</a>
            </li>
            <li class="nav-item">
                <a class = "nav-link" href="{% url 'users:login' %}">Увійти</a>
            </li>
            {% endif %}
        </ul>
    </div>
</nav>
```

Рис. 3.33. Шаблон навігаційного рядка

Тег <nav> позначає посилання на навігаційний рядок. Всередині цього тегу записаний атрибут class, що буде визначати, які класи з CSS-файлів будуть застосовані до цього тегу. В Bootstrap вже є готові класи, тому аби знати, який клас використовувати у певному випадку, необхідно ознайомитись з документацією цього фреймворку. Тут застосовуються селектори navbar, navbar-expand-md. Navbar-light та bg-light визначає, що навігаційна стрічка буде мати світлий задній фон. Mb-4 - це скорочення від margin-bottom (створити відступ в 4 пікселі знизу по зовнішній межі, а border створює лінію між навігаційною стрічкою та основним вмістом сторінки.

Тег <a> створю посилання на головну сторінку та надаючи цьому надпису селектор navbar-brand. Тег створює пустий контейнер. Посилання на сторінку «Мої предмети» буде знаходитись у лівому кутку екран, поряд з посиланням на головну сторінку, так як у тезі <ul> визначено ml-auto (margin-left:auto, автоматичне вирівнювання по лівому краю. Текст з вітанням для користувача та посилання на сторінки авторизації, реєстрації та виходу з акаунту будуть знаходитись зліва. Якщо екран буде занадто вузький, то посилання будуть вибудовуватись один за одним донизу. Це передбачено за допомоги селектору collapse. Навігаційний рядок в різних станах показаний на рис.3.34.

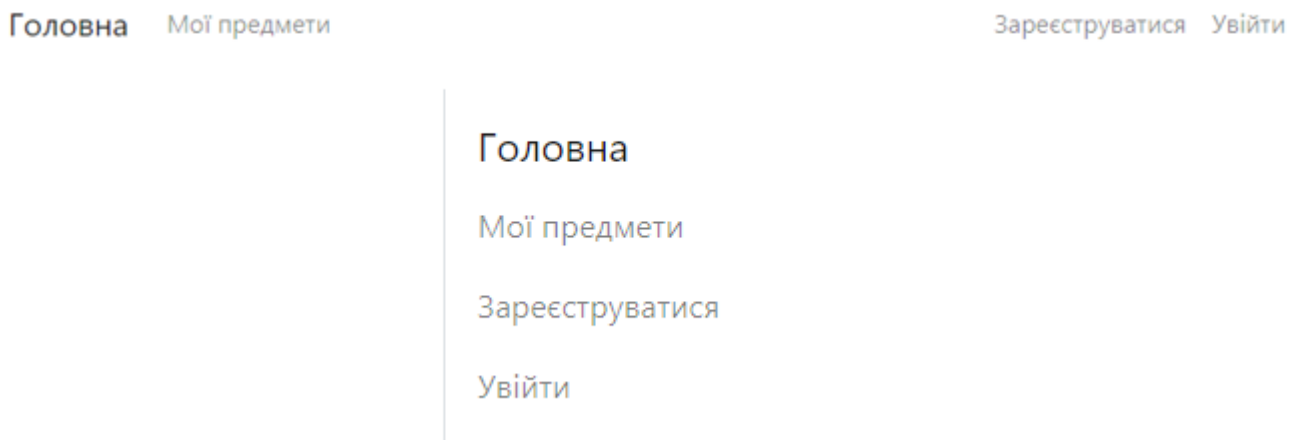

Рис. 3.34. Навігаційний рядок

Наступним кроком треба створити дизайн для торінок з формами. До прикладу візьмемо сторінку авторизації. Для початку завантажимо до цієї сторінки дані з файлу bas.html, адже саме там підключено CSS та JavaScript файли за допомоги тегу {% load_bootstrap4 %}. Шаблон цієї сторінки показаний на рис.3.35.

```
<p>Введіть логін та пароль</p>
{% if form.errors %}
    <p>Ваш логін і пароль не співпадають</p>
{% endif %}
<form method = "post" action = "{% url 'users:login' %}" class="form">
    {% csrf_token %}
    {% bootstrap_form form %}
    {% buttons %}
    <button name="submit" class="btn btn-primary">Увійти</button>
    {% endbuttons %}
    <input type="hidden" name="next" value="{% url 'todos:index' %}"/>
</form>
```

Рис. 3.35. Шаблон сторінки авторизації

Для того, аби поля форми займали всю ширину екрану, застосовано селектор form для тегу <form>. Також для кнопки увійти були обрані селектори btn та btn-primary, таким чином кнопка стане синього кольору (рис.3.36). Дизайн інших форм буде дуже схожим, повний їх код можна переглянути в додатку Б.

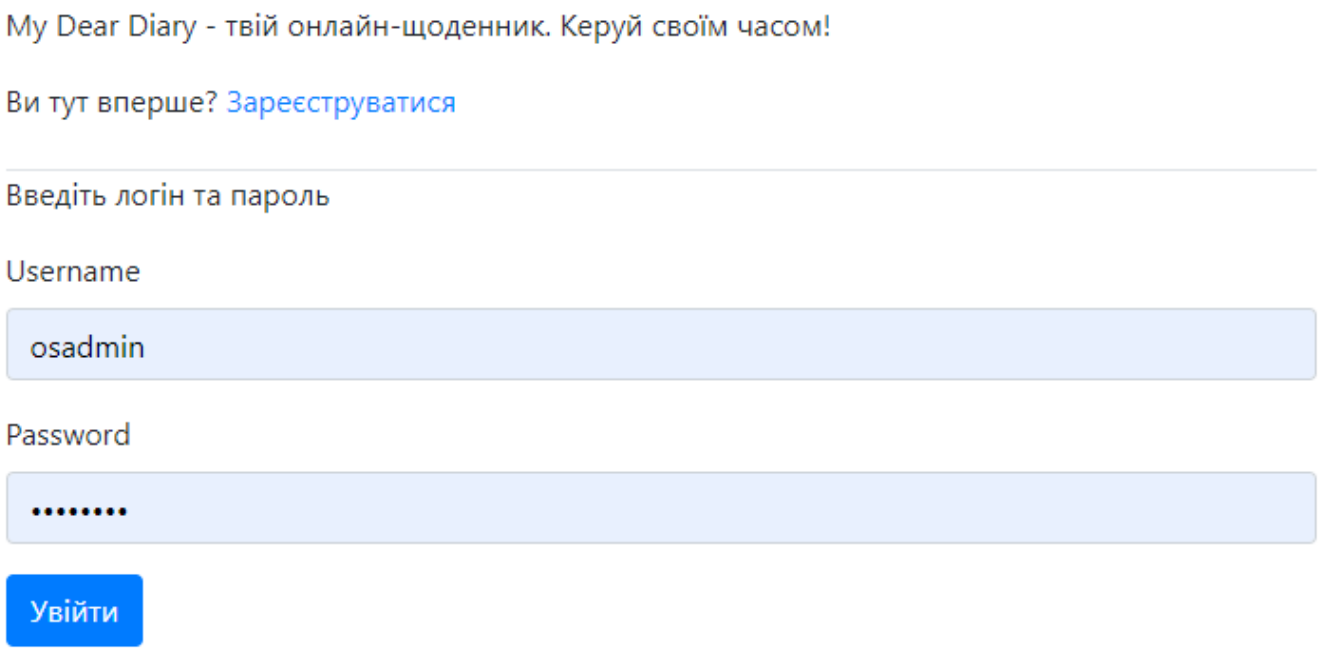

Рис. 3.36. Сторінка авторизації

Останнім етапом буде дизайн головної сторінки. Для створення зручного інтерфейсу, було створено картки з завданнями. В заголовку цієї картки

відображатиметься час та дата виконанння завдання, посилання на сторінки редагування та видалення завдання, а в тілі картки – сам текст (рис.3.37).

```
<div class="card mb-3">
{% if t.complete %}
    <h4 class="card-header"> <s>{{t.deadline|date:'H:i'}}</s>
        <small><a href="{% url 'todos:edit_task' t.id%}">Редагувати</a>
            <a href="{% url 'todos:delete_task' t.id %}">Видалити</a> </small></h4>
    <div class="card-body"><s> {{t.subject}} - {{t.text}}</s></div>
    {% else %}
    <h4 class="card-header">{{t.deadline|date:'H:i'}}
        <small><a href="{% url 'todos:edit_task' t.id%}">Редагувати</a>
            <a href="{% url 'todos:delete_task' t.id %}">Видалити</a> </small></h4>
    <div class="card-body"> {{t.subject}} - {{t.text}}</div>
    {% endif %}
</div>
```

Рис. 3.37. Шаблон картки з завданням

Для створення картки використовується тег з селектором card. Для створення заголовку картки використовується селектор card-header. Текст завдання розміщено у з селектором card-body. Загалом, основна сторінка матиме вигляд, як на рис.3.38.

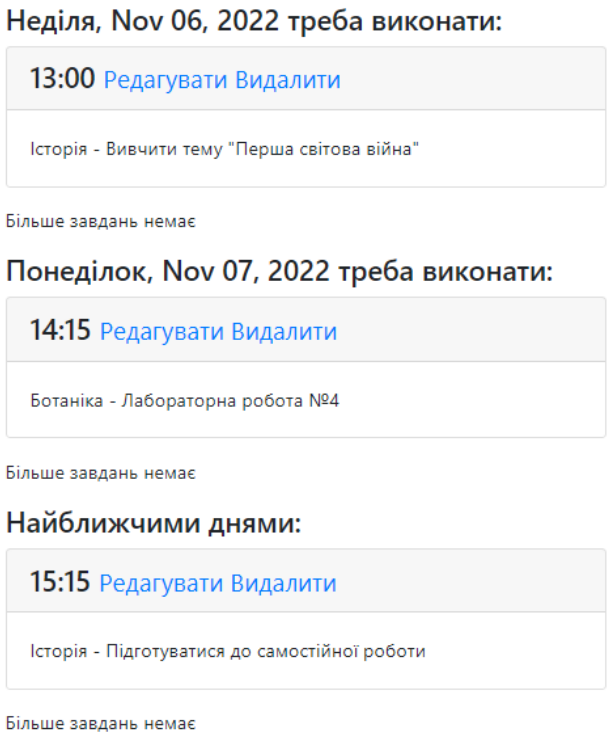

Рис. 3.38. Головна сторінка

Всі інші стрінки будуть мати схожий дизайн, тому тут вони не розглядаються. Повний лістинг коду для інших сторінок знаходиться в додатку Б.

3.5. Публікація застосунку на сервер

Фінальним етапом розробки будь-якого додатку є його публікація на сервер. У другому розділі дипломної роботи було визначено, за допомогою якого програмного забезпечення буде забезпечено публікацію на сервер. Для цього обрали хостинг Heroku та систему контролю версій Git. Перед початком процесу публікації на сервер необхідно встановити Heroku CLI, зареєструватися на хостингу Heroku та створити акаунт у Github.

Перед деплоєм застосунку, необхідно також підготувати файли для коректної роботи серверу. Heroku потрібно знати, які модулі використовуються у додатки, аби додатково їх встановити перед початком роботи застосунку (рис.3.39).

```
asgiref==3.5.2
beautifulsoup4==4.11.1
dj-database-url==1.0.0
Django==4.1.2
django-bootstrap4==22.2
django-heroku==0.3.1
gunicorn==20.1.0
psycopg2==2.9.5
psycopg2-binary==2.9.5
soupsieve==2.3.2.post1
sqlparse==0.4.3
tzdata==2022.5
whitenoise==6.2.0
```

Рис. 3.39. Вміст файлу requirements.txt

Команда pip freeze зберігає назви всіх модулів, що використовувались у проєкті у файл requirements.txt.

Якщо не вказати версію Python, то при публікації на сервер Heroku обере версію за замовчуванням. Але в такому випадку деякі функції можуть некоректно

працювати, тому краще в такому випадку створити файл runtime.txt та вказати там версію Python , що використовується в додатку.

Наступним кроком необхідно конфігурувати певні налаштування, що повинні отримати специфічні значення, яких потребує Heroku. Такі налаштування проводяться у файлі setting.py. Переглянути код можна в додатку Б.

Далі створюємо файл Procfile. Його мета - вказати на процесі, які потрібно запустити, щоб проєкт запрацював Рядок в цьому файлі вказує, що необхідно використати gunicorn як сервер та взяти налаштування з файлу todo/wsgi.py.

Наразі, при запуску застосунку, коли стається помилка, то на екран виводиться розширена інформація про неї. Це досить небезпечно, так як потенційні злодії можуть скористатись цим у власних цілях. Тому, аби такого не трапилось, маємо зробити наступне:

- Усунути розгорнуте виведення помилок, скориставшись змінною середовища DEBUG та встановити їй значення False;
- Створити сторінки для виведення клієнтських та серверних помилок (файли 404.html та 500.html).

Зміни у файлі settings.py показані на рис.3.40.

```
if os.environ.get('DEBUG') == 'TRUE':
    DEBUG = True
if os.environ.get('DEBUG') == 'FALSE':
    DEBUG = False
```

Рис. 3.40. Вміст файлу settings.py

Наступним кроком необхідно створити шаблони для виведення клієнтських та серверних помилок. Результат роботи цих шаблонів показані на рис.3.41.

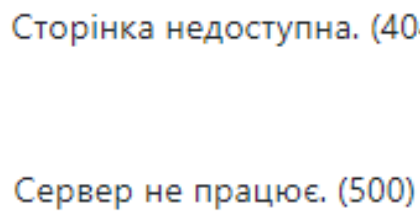

Рис. 3.41. Сторінки помилок

Коли налаштування безпеки готові, можна викладати застосунок на сервер. Для цього потрібно скористатись системою контролю версій Git . Він служить для створення комітів (відбитків коду), які в подальшому запускають у сервері. За допомоги комітів можна легко працювати над проєктом, при цьому робоча його версія завжди буде збережена у попередньому коміті.

Перед тим , як створити коміт, потрібно створити файл .gitignore , що буде вказувати, які файли не потрібно завантажувати у коміт.

Створення коміту відбувається за допомоги таких команд:

- git init – створює порожнє середовище у папці;
- git add . – додає всі файли, які не потрібно ігнорувати, до нового репозиторію;
- git commit –am “Ready for deployment” – вказує додати всі змінені файли в коміт та додати запис у журнал комітів;
- git status – виводить повідомлення про те, що коміт знаходиться у гілці master і робоче дерево чисте.

```
(todo_env) C:\Users\shark\OneDrive\Рабочий стол\todo>git init
Initialized empty Git repository in C:/Users/shark/OneDrive/Рабочий стол/todo/.git/

(todo_env) C:\Users\shark\OneDrive\Рабочий стол\todo>git add .

(todo_env) C:\Users\shark\OneDrive\Рабочий стол\todo>git commit -am "Ready for deployment"
[master (root-commit) 1bd2f78] Ready for deployment
 44 files changed, 831 insertions(+)
```

Рис. 3.42. Створення коміту

Можна завантажити проєкт на хостинг. Для цього потрібно записати наступні команди:

- heroku login – для авторизації на хостингу Heroku;
- heroku create – для створення нового застосунку;
- git push heroku master – завантажує головну гілку репозиторію Heroku, яку було створено в коміті;
- heroku ps – показує, скільки часу безкоштовно буде працювати застосунок (550 годин).

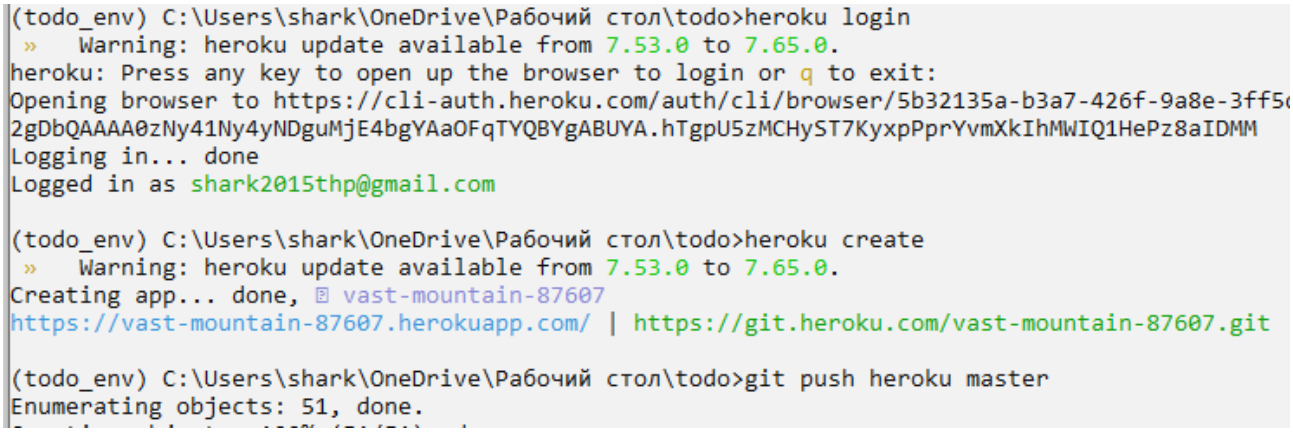

Рис. 3.43. Публікація застосунку на сервер

Аби база даних працювала коректно , необхідно мігрувати її вже на сервері Heroku. Команди для Python запускаються на сервері Heroku за допомоги команди heroku run (рис.3.44).

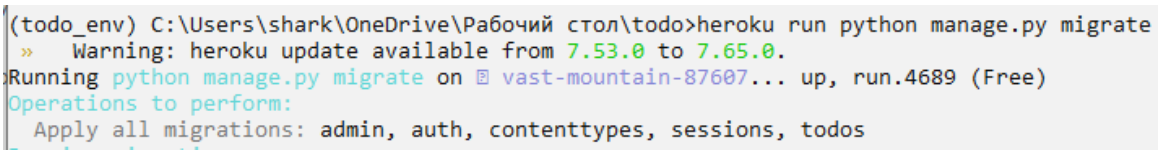

Рис. 3.44. Міграція баз даних на сервері

Для того, щоб мати доступ до панелі адміністратора так само, як і в неопублікованому застосунку, потрібно створити адміністратора в проєкті, що вже викладений на сервер. Для цього можна скористатися командою heroku run

bash. Скриптова мова bash дозволяє відкрити інтерактивну консоль Python та записувати туди команди . Як створювати суперкористувача було пказано у пункті 3.2.

Останній етап – вибір зручного URL для доступу до застосунку. Вибрати URL можна за допомоги команди heroku apps:rename (рис.3.45).

```
(todo_env) C:\Users\shark\OneDrive\Рабочий стол\todo>heroku apps:rename my-dearr-diaryy
 »   Warning: heroku update available from 7.53.0 to 7.65.0.
Renaming vast-mountain-87607 to my-dearr-diaryy... done
https://my-dearr-diaryy.herokuapp.com/ | https://git.heroku.com/my-dearr-diaryy.git
 !    Don't forget to update git remotes for all other local checkouts of the app.
Git remote heroku updated
```

Рис. 3.45. Перейменування застосунку

При запуску додатку на екран виводиться сторінка авторизації (рис.3.46).

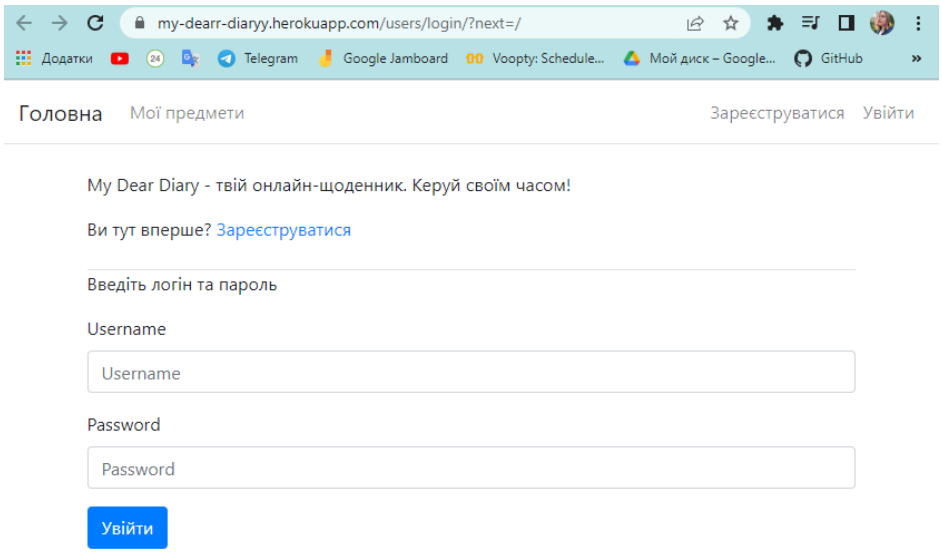

Рис. 3.46. Готовий застосунок

Перейти до цього застосунку можна за посиланням https://my-dearr-diaryy.herokuapp.com/. Якщо в застосунку необхідні будуть зміни, то потрібно дотримуватись наступного алгоритму : зробити необхідні зміни у локальному проєкті; додати нові файли до репозиторію, якщо вони є; зробити нову міграцію, якщо була зміна у базі даних; створити новий коміт та завантажити зміни за допомоги git push heroku master.

Висновки до розділу 3

Створення застосунку включає безліч процесів, таких як створення проєкту та встановлення необхідних модуів, створення база, створення шаблонів та форм, дизайн застосунку, а також його публікація на сервер. В цьому розділі було детально розглянуто кожен з цих етапів.

Створення проєкту у Django – один з найважливіших етапів, адже від правильної конфігурації файлів залежить подальша робота застосунку. Було створено віртуальне середовище, куди встановили всі необхідні пакунки: django, bootstrap, psycopg, gunicorn та django-heroku. Також було створено два вкладених застосунки: todos – для управління основними даними проєкту та users – для створення користувацьких облікових записів.

Також було розроблено базу даних, в які зберігається інформація про користувачів, їх предмети та завдання до них. Кожна з цих сутностей має атрибути, що дозволяють легко зберігати всю необхідну інформацію. Окрім цього, створено адміністратора, що має певні привілеї, такі як видалення або створення облікових записів, перегляд інформації про користувачів тощо.

Основна сторінка додатку виводить завдання на сьогодні, завтра та на найближчі три дні. Сторінка «Мої предмети» зберігає інформацію про всі предмети, які додав користувач. Натиснувши на назву предмету, також можна переглянути завдання, які стосуються цього предмету. Завдання та предмети можна редагувати, додавати та видаляти за допомоги створених форм.

Користувацькі обліков записи дозволяють розділити інформацію між кожним користувачем, аби кожному користувачеві було видно лие ті записи, які він створював. Було розроблено сторінки реєстрації, авторизації та виходу з облікового запису.

Дизайн застосунку дозволяє зробити його приємним на вигляд, так полегшує користувачам роботу з застосунком. Тут було використано фреймворк Bootstrap та його селектори. Завдяки цому інтерфейс користувача став набагато привабливішим.

Публікація застосунку на сервер потрібна для віддаленого доступу до додатку без використання локального комп’ютера. Створивши зручний URL, тепер кожен бажаючий може використовувати застосунок у васних цілях.

ВИСНОВКИ

Зважаючи на події в Україні в період з 2019 року по 2022 рік, більшість дітей шкільного віку, а також студентів вищих навчальних закладів тепер вимушено або за бажанням перейшли на навчання онлайн. В результаті переходу на такий формат навчання, виникла гостра потреба у великій кількості застосунків та програм для вирішення найрізноманітныших задач: від створення зручного розкладу до онлайн-тестування або систем для оцінювання учнів.

Проаналізувавши існуючі аналоги, видно, що багато додатків або не пристосовані до маленьких користувачів, або ж мають скромний функціонал, що не завжди може задовільнити всі потреби. А більшість загалом корисних та зручних додатків просто не мають української локалізації, що є вкрай важливим.

Сукупністю всіх цих факторів було вирішено створити застосунок «Щоденник» для ведення власного розкладу та запису домашніх завдань.

Для створення додатку було обрано фрейморк Django. Він дозволяє зручно управляти базами даних та створювати функції для обробки даних з неї. Також цей фреймворк має зручну панель адміністратора, що може керувати процесами в додатку, створювати та видаляти облікові записи.

Для створення шаблонів застосунку було обрано мову гіпертекстової розмітки HTML, так як він дозволяє створити основну структуру сторінки. При цьому в HTML-файлі можна використовувати шаблонні теги та змінні, що полегшує виведення даних на сторінку. За допомоги HTML також було створено сторінки для користувацького вводу даних та сторінки реєстрації, авторизації та виходу з облікового запису.

З використанням фреймворку Bootstrap зробили привабливий та зручний інтерфейс для кожної сторінки. З його допомогою тепер можна користуватись додатком з будь-якого пристрою, адже навігаційна панель згортається, а сторінки приймають розмір будь-якого екрану.

За допомоги системи контролю Git створено відбиток коду та завантажено до репозиторію Github. Використовуючи цей репозиторій, хостинг Heroku може

виконувати код застосунку та надавати доступ до нього за допомоги зручного URL https://my-dearr-diaryy.herokuapp.com/.

При подальшій розробці до застосунку можна додати наступні функції:

- створити регулярні нагадування про дату виконання кожного завдання, наприклад, на завтра або в найближчий час;
- виконати яскравий, барвистий дизайн, який буде захоплювати дітей;
- створити розклад уроків, аби можна було вибирати не час виконання завдання, а номер уроку;
- додати можливість додавання своїх оцінок та відстежувати відвідуваність уроків;
- додавати до кожного уроку файли з теоретичними матеріалами, посиланнями на інтерактивні дошки тощо.

Загалом, розроблений проєкт дозволяє кожному користувачеві додавати власні предмети та завдання у віддаленому застосунку, використовуючи зручний інтерфейс.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ

1. Закон України «Про освіту» від 05 вер. 2017р. / Верховна Рада України. – Офіц. вид. – Київ: Парлам. вид-во, 2017. – с.380.
2. Дистанційка – не причина проблем в освіті, а їх індикатор. Автор: Тарас Павлов [Електронний ресурс]. Режим доступу: https://zn.ua (дата звернення: 28.09.2022р).
3. My Study Life – Online Study Planner. [Електронний ресурс]. Режим доступу: https://www.mystudylife.com/ (дата звернення: 28.09. 2022р).
4. Canva – Вікіпедія. [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/Canva (дата звернення: 28.09.2022р).
5. The Ultimate Guide To Google Calendar – Calendar. [Електронний ресурс]. Режим доступу: https://www.calendar.com/google-calendar-guide/ (дата звернення: 28.09.2022р).
6. Free Online Schedule Maker | Plan weekly activities. [Електронний ресурс]. Режим доступу: https://schedulebuilder.org/ (дата звернення: 28.09.2022р).
7. Типи мобільних додатків: нативні та гібридні. [Електронний ресурс]. Режим доступу: https://smile-ukraine.com/ (дата звернення: 12.10.2022р).
8. Типи і приклади додатків, де брати та качати додатки на мобільні пристрої. [Електронний ресурс]. Режим доступу: http://ipkey.com.ua/ (дата звернення: 12.10.2022р).
9. Desktop програми: що це, навіщо потрібні, які у них переваги. [Електронний ресурс]. Режим доступу: https://wezom.com.ua/ (дата звернення: 12.10.2022р).
10. «Особливості веб-додатків». Конспект лекцій «Сучасні методи веб-програмування». Запорізький національний університет. [Електронний ресурс]. Режим доступу: http://sites.znu.edu.ua/ (дата звернення: 12.10.2022р).
11. Common web application architectures | Microsoft Learn. [Електронний ресурс]. Режим доступу: https://learn.microsoft.com/ (дата звернення: 06.11.2022 р).

12. Олександр Мізюк – Путівник мовою програмування Python. Вивчення основ програмування для початківців. [Електронний ресурс]. Режим доступу: https://pythonguide.rozh2sch.org.ua/ (дата звернення: 06.11.2022р).
13. Пришвидшений курс Python. Практичний, проєктно-орієнтований вступ до програмування [Текст] : Ерік Маттес, перекл. з англ. Ольги Бєлової. – Львів : Видавництво Старого Лева, 2021. – 600 с.
14. Django – PyPi. [Електронний ресурс]. Режим доступу: https://pypi.org/project/Django (дата звернеййня 21.10.2022р).
15. Django Documentation. [Електронний ресурс]. Режим доступу: https://docs.djangoproject.com/en/4.0/ (дата звернення 14.10.2022р).
16. MySQL. [Електронний ресурс]. Режим доступу: https://www.mysql.com (дата звернення 15.10.2022р).
17. What is HTML? Hypertext Markup Language Basics Explained. [Електронний ресурс]. Режим доступу: https://www.hostinger.com/tutorials/what-is-html (дата звернення 21.10.2022р).
18. Bootstrap. [Електронний ресурс]. Режим доступу: https://getbootstrap.com/ (дата звернення: 01.11.2022р).
19. Git – Про системи контролю версій. [Електронний ресурс]. Режим доступу: https://git-scm.com/book/ru/v2/ (дата звернення 21.10.2022р).
20. Cloud Application Platform | Heroku. [Електронний ресурс]. Режим доступу: https://blog.heroku.com/data-in-functions (дата звернення 22.10.2022р).

ДОДАТКИ

Додаток А

Інсталяція модуля Django у віртуальне середовище:

```
(todo_env) C:\Users\shark\OneDrive\Рабочий стол\todo>pip install django
Requirement already satisfied: django in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages (4.1.2)
Requirement already satisfied: tzdata in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages (from djan
go) (2022.5)
Requirement already satisfied: asgiref<4,>=3.5.2 in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages
 (from django) (3.5.2)
Requirement already satisfied: sqlparse>=0.2.2 in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages (
from django) (0.4.3)
```

Рис. А.1.

Інсталяція модуля Django-bootstrap4 у віртуальне середовище:

```
(todo_env) C:\Users\shark\OneDrive\Рабочий стол\todo>pip install django-bootstrap4
Requirement already satisfied: django-bootstrap4 in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages
 (22.2)
Requirement already satisfied: beautifulsoup4>=4.8.0 in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-pack
ages (from django-bootstrap4) (4.11.1)
Requirement already satisfied: Django>=3.2 in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages (from
 django-bootstrap4) (4.1.2)
Requirement already satisfied: soupsieve>1.2 in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages (fr
om beautifulsoup4>=4.8.0->django-bootstrap4) (2.3.2.post1)
Requirement already satisfied: asgiref<4,>=3.5.2 in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages
 (from Django>=3.2->django-bootstrap4) (3.5.2)
Requirement already satisfied: tzdata in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages (from Djan
go>=3.2->django-bootstrap4) (2022.5)
Requirement already satisfied: sqlparse>=0.2.2 in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages (
from Django>=3.2->django-bootstrap4) (0.4.3)
```

Рис. А.2.

Інсталяція модуля psycopg2 у віртуальне середовище:

```
(todo_env) C:\Users\shark\OneDrive\Рабочий стол\todo>pip install psycopg2-binary
Requirement already satisfied: psycopg2-binary in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages (
2.9.5)
```

Рис. А.3.

Інсталяція модуля Django-heroku у віртуальне середовище:

```
(todo_env) C:\Users\shark\OneDrive\Рабочий стол\todo>pip install django-heroku
Requirement already satisfied: django-heroku in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages (0.
3.1)
Requirement already satisfied: whitenoise in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages (from
django-heroku) (6.2.0)
Requirement already satisfied: django in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages (from djan
go-heroku) (4.1.2)
Requirement already satisfied: dj-database-url>=0.5.0 in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-pac
kages (from django-heroku) (1.0.0)
Requirement already satisfied: psycopg2 in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages (from dj
ango-heroku) (2.9.5)
Requirement already satisfied: tzdata in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages (from djan
go->django-heroku) (2022.5)
Requirement already satisfied: sqlparse>=0.2.2 in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages (
from django->django-heroku) (0.4.3)
Requirement already satisfied: asgiref<4,>=3.5.2 in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages
 (from django->django-heroku) (3.5.2)
```

Рис. А.4.

Інсталяція модуля gunicorn у віртуальне середовище:

```
(todo_env) C:\Users\shark\OneDrive\Рабочий стол\todo>pip install gunicorn
Requirement already satisfied: gunicorn in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages (20.1.0)

Requirement already satisfied: setuptools>=3.0 in c:\users\shark\onedrive\рабочий стол\todo\todo_env\lib\site-packages (
from gunicorn) (58.1.0)
```

Рис. А.5.

todo/setting.py

```
from pathlib import Path
import os

BASE_DIR = Path(__file__).resolve().parent.parent
SECRET_KEY = 'django-insecure-9--p(o4_(7py!z_*hl&=j$ol0+@d827*_@$xqz$!w^3*!(hwzy'
DEBUG = False
ALLOWED_HOSTS = []
INSTALLED_APPS = [
    'todos',
    'users',
    'bootstrap4',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'todo.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
```

```
]

WSGI_APPLICATION = 'todo.wsgi.application'
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_L10N = True

USE_TZ = True

STATIC_URL = 'static/'

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

LOGIN_URL = 'users:login'
import django_heroku
django_heroku.settings(locals())
if os.environ.get('DEBUG') == 'TRUE':
    DEBUG = True
if os.environ.get('DEBUG') == 'FALSE':
    DEBUG = False
```

todo/urls.py

```
from django.contrib import admin
from django.urls import path, include
```

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('users/', include('users.urls')),
    path("", include('todos.urls'))
]
```

todos/admin.py

```
from django.contrib import admin
from .models import Subject, Task
# Register your models here.
admin.site.register(Subject)
admin.site.register(Task)
```

todos/forms.py

```
from django import forms
from .models import Subject, Task

class SubjectForm(forms.ModelForm):
    class Meta:
        model = Subject
        fields = ['name', 'teacher', 'classroom', 'meet', 'text']
        labels = {'name': 'Назва предмету', 'teacher': 'Викладач', 'classroom':'Клас',
'meet': 'Google Meet', 'text' : ""}

class TaskForm(forms.ModelForm):
    class Meta:
        model =  Task
        fields = ['text', 'deadline', 'complete']
        labels = {'text': 'Завдання', 'deadline':'Дата виконання', 'complete':
"Виконано"}
        widgets = {'deadline': forms.DateTimeInput(attrs = {'type': 'datetime-
local'})}
```

todos/models.py

```
from email.policy import default
from django.db import models
from django.contrib.auth.models import User

# Create your models here.

class Subject(models.Model):
    name = models.CharField(max_length = 200)
    teacher = models.CharField(max_length = 500)
    classroom = models.CharField(max_length = 200)
    meet = models.CharField(max_length = 500)
    text = models.TextField()
    owner = models.ForeignKey(User, on_delete = models.CASCADE)
```

```
    def __str__(self):
        return self.name

class Task(models.Model):
    subject = models.ForeignKey(Subject, on_delete = models.CASCADE)
    text = models.TextField()
    deadline = models.DateTimeField()
    complete = models.BooleanField(default = False)

    def __str__(self):
        return f"{self.text[:50]}..."
```

todos/urls.py

```
from django.urls import path

from . import views

app_name = 'todos'

urlpatterns = [
    path('', views.index, name = "index"),
    path('subjects', views.subjects, name = "subjects"),
    path('subjects/<int:subject_id>', views.subject, name = "subject"),

    path('new_subject/', views.new_subject, name = 'new_subject'),
    path('new_task/<int:subject_id>', views.new_task, name = 'new_task'),
    path('edit_subject/<int:subject_id>', views.edit_subject, name = "edit_subject"),
    path('edit_task/<int:task_id>', views.edit_task, name='edit_task'),
    path('delete_subject/<int:subject_id>',views.delete_subject, name =
'delete_subject'),
    path('delete_task/<int:task_id>', views.delete_task, name = "delete_task")

]
```

todos/views.py

```
from django.shortcuts import render, redirect
from .models import Subject, Task
from .forms import SubjectForm, TaskForm
import datetime
import calendar
from django.contrib.auth.decorators import login_required
from django.http import Http404

@login_required
def index(request):
    subjects = Subject.objects.filter(owner = request.user)
    tasks=[]
    for subject in subjects:
        task = subject.task_set.order_by('deadline')
```

```
            tasks.append(task)
        week = ['Понеділок', 'Вівторок', 'Середа', 'Четвер', 'П\'ятниця', 'Субота',
'Неділя']
        today = datetime.datetime.today()
        day = week[calendar.weekday(today.year, today.month, today.day)]
        tomorrow = today + datetime.timedelta(days =  1)
        tomorrow_day = week[calendar.weekday(tomorrow.year, tomorrow.month, tomorrow.day)]
        near_days = [today + datetime.timedelta(days=2), today +
datetime.timedelta(days=3), today + datetime.timedelta(days=4)]
        context = {'near_days': near_days,
                   'subjects':subjects,
                   'tasks': tasks,
                   'today': today,
                   'day': day,
                   'tomorrow_day': tomorrow_day,
                   'tomorrow': tomorrow}
        return render(request, 'todos/index.html', context)

@login_required
def subjects(request):
    subjects = Subject.objects.filter(owner = request.user).order_by('name')
    context = {'subjects': subjects}
    return render(request, 'todos/subjects.html', context)


@login_required
def subject(request, subject_id):
    subject = Subject.objects.get(id = subject_id)
    if subject.owner != request.user:
        raise Http404
    tasks = subject.task_set.order_by('deadline')
    context = {'subject': subject, 'tasks': tasks}
    return render(request, 'todos/subject.html', context)

@login_required
def new_subject(request):
    if request.method != 'POST':
        form = SubjectForm()
    else:
        form = SubjectForm(data = request.POST)
        if form.is_valid():
            new_subject = form.save(commit=False)
            new_subject.owner = request.user
            new_subject.save()
            return redirect('todos:subjects')
    context = {'form': form }
    return render (request, 'todos/new_subject.html', context)

@login_required
def new_task(request, subject_id):
    subject = Subject.objects.get(id = subject_id)
```

```
    if request.method !='POST':
        form = TaskForm()
    else:
        form = TaskForm(data=request.POST)
        if form.is_valid():
            new_task = form.save(commit=False)
            new_task.subject = subject
            new_task.save()
            return redirect('todos:subject', subject_id = subject_id)
    context = {'subject': subject, 'form': form}
    return render( request, 'todos/new_task.html', context)

@login_required
def edit_subject(request, subject_id):
    subject = Subject.objects.get(id = subject_id)
    if subject.owner != request.user:
        raise Http404
    if request.method != "POST":
        form = SubjectForm(instance=subject)
    else:
        form = SubjectForm(instance=subject, data = request.POST)
        if form.is_valid():
            form.save()
            return redirect('todos:subjects')
    context = {'subject': subject, 'form': form}
    return render(request, 'todos/edit_subject.html', context)

@login_required
def edit_task(request, task_id):
    task = Task.objects.get(id = task_id)
    subject = task.subject
    if request.method != "POST":
        form = TaskForm(instance = task)
    else:
        form = TaskForm(instance=task, data = request.POST)
        if form.is_valid():
            form.save()
            return redirect('todos:subject', subject_id = subject.id)
    context = {'task': task, 'subject': subject, 'form': form}
    return render(request, 'todos/edit_task.html', context)

@login_required
def delete_subject(request, subject_id):
    subject = Subject.objects.get(id = subject_id)
    subject.delete()
    context = {'subject': subject}
    return render(request, 'todos/delete_subject.html', context)

@login_required
def delete_task(request, task_id):
    task = Task.objects.get(id = task_id)
```

```
    subject = task.subject
    task.delete()
    context = {'task': task, 'subject': subject}
    return render(request, 'todos/delete_task.html', context)
```

todos/templates/todos/base.html

```
{% load bootstrap4 %}
    <!doctype html>
    <html lang="uk">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
fit=no">
        <title>My Dear Diary</title>

        {% bootstrap_css %}
        {% bootstrap_javascript jquery='full' %}
    </head>
    <body>

        <nav class="navbar navbar-expand-md navbar-light bf-light mb-4 border">
            <a class="navbar-brand" href="{% url 'todos:index' %}">Головна</a>
            <div class="collappse navbar-collapse" id="navbarCollapse">
                <ul class="navbar-nav mr-auto">
                    <li class="nav-item">
                        <a class = "nav-link" href="{% url 'todos:subjects' %}"> Мої
предмети </a>
                    </li>
                </ul>
                <ul class="navbar-nav ml-auto">
                    {% if user.is_authenticated %}
                    <li class="nav-item">
                        <span class="navbar-text">Hello, {{ user.username }}!</span>
                    </li>
                    <li class="nav-item">
                        <a class = "nav-link" href="{% url 'users:log_out'
%}">Вийти</a>
                    </li>
                    {% else %}
                    <li class="nav-item">
                        <a class = "nav-link" href="{% url 'users:register'
%}">Зареєструватися</a>
                    </li>
                    <li class="nav-item">
                        <a class = "nav-link" href="{% url 'users:login'
%}">Увійти</a>
                    </li>
                    {% endif %}
                </ul>
```

```
            </div>
        </nav>
        <main role="main" class="container">
            <div class="pb-2 border-bottom">
                {% block page_header %}{% endblock page_header %}
            </div>
            <div>
                {% block content %}{% endblock content %}
            </div>
        </main>

    </body>
</html>
```

todos/templates/todos/index.html

```
{% extends 'todos/base.html' %}
{% block content %}
<p>
    <h4> {{day}}, {{today|date:'M d, Y'}} треба виконати: </h4>
        {% for task in tasks %}
            {% for t in task %}

                {% if t.deadline.year == today.year and t.deadline.month ==
today.month and t.deadline.day == today.day %}
                <div class="card mb-3">
                {% if t.complete %}
                    <h4 class="card-header"> <s>{{t.deadline|date:'H:i'}}</s>
                        <small><a href="{% url 'todos:edit_task'
t.id%}">Редагувати</a>
                            <a href="{% url 'todos:delete_task' t.id %}">Видалити</a>
</small></h4>
                    <div class="card-body"><s> {{t.subject}} - {{t.text}}</s></div>
                    {% else %}
                    <h4 class="card-header">{{t.deadline|date:'H:i'}}
                        <small><a href="{% url 'todos:edit_task'
t.id%}">Редагувати</a>
                            <a href="{% url 'todos:delete_task' t.id %}">Видалити</a>
</small></h4>
                    <div class="card-body"> {{t.subject}} - {{t.text}}</div>
                    {% endif %}
                </div>
                {% endif %}
            {% endfor %}
        {% endfor %}
        <p>Більше завдань немає</p>
</p>
<p>
    <h4> {{tomorrow_day}}, {{tomorrow|date:'M d, Y'}} треба виконати: </h4>
        {% for task in tasks %}
```

```
            {% for t in task%}
                {% if t.deadline.year == tomorrow.year and t.deadline.month ==
tomorrow.month and t.deadline.day == tomorrow.day %}
                <div class="card mb-3">
                {% if t.complete %}
                <h4 class="card-header"> <s>{{t.deadline|date:'H:i'}}</s>
                    <small><a href="{% url 'todos:edit_task' t.id%}">Редагувати</a>
                        <a href="{% url 'todos:delete_task' t.id %}">Видалити</a>
</small></h4>
                <div class="card-body"><s> {{t.subject}} - {{t.text}}</s></div>
                    {% else %}
                <h4 class="card-header">{{t.deadline|date:'H:i'}}
                    <small><a href="{% url 'todos:edit_task' t.id%}">Редагувати</a>
                        <a href="{% url 'todos:delete_task' t.id %}">Видалити</a>
</small></h4>
                <div class="card-body"> {{t.subject}} - {{t.text}}</div>
                    {% endif %}
                </div>
                {% endif %}
            {% endfor %}
        {% endfor %}
        <p>Більше завдань немає</p>
</p>
<p>
    <h4> Найближчими днями:</h4>
        {% for task in tasks %}
            {% for t in task %}
                {% for day in near_days%}
                {% if t.deadline.year == day.year and t.deadline.month == day.month
and t.deadline.day == day.day %}
                <div class="card mb-3">
                    {% if t.complete %}

                    <h4 class="card-header"> <s>{{t.deadline|date:'H:i'}}</s>
                        <small><a href="{% url 'todos:edit_task'
t.id%}">Редагувати</a>
                            <a href="{% url 'todos:delete_task' t.id %}">Видалити</a>
</small></h4>
                    <div class="card-body"><s> {{t.subject}} - {{t.text}}</s></div>
                    {% else %}
                    <h4 class="card-header"> {{t.deadline|date:'H:i'}}
                        <small><a href="{% url 'todos:edit_task'
t.id%}">Редагувати</a>
                            <a href="{% url 'todos:delete_task' t.id %}">Видалити</a>
</small></h4>
                    <div class="card-body"> {{t.subject}} - {{t.text}}</div>
                    {% endif %}
                    </div>
                {% endif %}
            {% endfor %}
        {% endfor %}
```

```
        {% endfor%}
        <p>Більше завдань немає</p>
</p>
{% endblock content %}
```

todos/templates/todos/subjects.html

```
{% extends 'todos/base.html' %}
{% block page_header %}
 <h2>Предмети: </h2>
 {% endblock page_header %}
 {% block content %}
 <ul>
    {% for subject in subjects%}
        <li> <h3><a style="text-decoration:none;" href="{% url 'todos:subject'
subject.id %}">{{subject}}</a></h3>
            <a href="{% url 'todos:edit_subject' subject.id %}">Редагувати</a>
            <a href="{% url 'todos:delete_subject' subject.id %}">Видалити</a>
            <p>Викладач: {{subject.teacher}} </p>
            <p>Клас: {{subject.classroom}} </p>
            <p>Google Meet: {{subject.meet}} </p>
            <p>Примітка: {{subject.text}} </p>
        </li>
    {% endfor %}
    <li>Більше предметів немає</li>
 </ul>
 <h4><a style="text-decoration:none;" href = "{% url 'todos:new_subject' %}">Додати
новий предмет</a></h4>

{% endblock content %}
```

todos/templates/todos/subject.html

```
{% extends 'todos/base.html' %}
{% block page_header %}
 <h2>Предмет: {{subject}}</h2>
{% endblock page_header %}
{% block content %}
 <p>
    <a href="{% url 'todos:new_task' subject.id %}">Додати нове завдання</a>
 </p>
    {% for task in tasks%}
        <div class="card mb-3">

            {% if task.complete %}
            <h4 class="card-header"> <s>Дата виконання: {{task.deadline|date:'M d, Y
H:i'}}</s>
                <small> <a href="{% url 'todos:edit_task' task.id%}">Редагувати</a>
                    <a href="{% url 'todos:delete_task' task.id %}">Видалити</a>
</small></h4>
```

```
            <div class="card-body"><s>{{task.text|linebreaks}}</s></div>

            {% else %}
            <h4 class="card-header"> {{task.deadline|date:'M d, Y H:i'}} <small> <a
href="{% url 'todos:edit_task' task.id%}">Редагувати</a>
                <a href="{% url 'todos:delete_task' task.id %}">Видалити</a> </small>
</h4>

                <div class="card-body">{{task.text|linebreaks}}</div>
            {% endif %}
            </div>
    {% endfor %}
    <p>Більше завдань немає</p>
{% endblock content %}
```

todos/templates/todos/new_subject.html

```
{% extends 'todos/base.html'%}
{% load bootstrap4 %}
{%block content%}
<p>Додати новий предмет: </p>
<form action="{% url 'todos:new_subject' %}" method="post" class="form">
    {%csrf_token%}
    {% bootstrap_form form %}
    {% buttons %}
    <button name = "submit" class="btn btn-primary">Зберегти</button>
    {% endbuttons %}
</form>
{%endblock content%}
```

todos/templates/todos/new_task.html

```
{% extends 'todos/base.html'%}
{% load bootstrap4 %}
{% block page_header %}
<a href = "{% url 'todos:subject' subject.id %}">{{subject}}</a>
{% endblock page_header %}
{%block content%}
<p>Додати нове завдання: </p>
<form action="{% url 'todos:new_task' subject.id %}" method="post" class="form">
    {%csrf_token%}
    {% bootstrap_form form %}
    {% buttons %}
    <button name = "submit" class="btn btn-primary">Зберегти</button>
    {% endbuttons%}
</form>
{%endblock content%}
```

todos/templates/todos/edit_subject.html

```
{% extends 'todos/base.html'%}
{% load bootstrap4 %}
{% block page_header %}
<p><a href="{% url 'todos:subject'  subject.id %}">{{subject}}</a></p>
{% endblock page_header %}
{%block content%}
<p>Редагувати предмет: </p>
<form action="{% url 'todos:edit_subject' subject.id %}" method="post" class="form">
    {%csrf_token%}
    {% bootstrap_form form %}
    {% buttons %}
    <button name = "submit" class="btn btn-primary">Зберегти</button>
    {% endbuttons %}
</form>
{%endblock content%}
```

todos/templates/todos/edit_task.html

```
{% extends 'todos/base.html'%}
{% load bootstrap4 %}
{% block page_header %}
<p><a href="{% url 'todos:subject' subject.id %}">{{subject}}</a></p>
{% endblock page_header %}
{%block content%}
<p>Редагувати завдання: </p>
<form action="{% url 'todos:edit_task' task.id %}" method="post" class="form">
    {%csrf_token%}
    {% bootstrap_form form %}
    {% buttons %}
    <button name = "submit" class="btn btn-primary">Зберегти</button>
    {% endbuttons %}
</form>
{%endblock content%}
```

todos/templates/todos/delete_subject.html

```
{% extends 'todos/base.html' %}
{% block content %}
<p>Предмет видалено</p>
<a href="{% url 'todos:subjects'%}">Повернутися</a>
{% endblock content %}
```

todos/templates/todos/delete_task.html

```
{% extends 'todos/base.html' %}
{% block content %}
<p>Завдання видалено</p>
```

```
<a href="{% url 'todos:subject' subject.id %}">Повернутися</a>
{% endblock content %}
```

users/urls.py

```
from django.urls import path,  include
from . import views
app_name = "users"
urlpatterns = [
    path('', include('django.contrib.auth.urls')),
    path('register/', views.register, name="register"),
    path('log_out/', views.log_out, name ="log_out")
]
```

users/views.py

```
from django.shortcuts import render, redirect
from django.contrib.auth import login, logout
from django.contrib.auth.forms import UserCreationForm
# Create your views here.
def log_out(request):
    logout(request)
    return render(request, 'registration/log_out.html')
def register(request):
    if request.method != 'POST':
        form = UserCreationForm()
    else:
        form = UserCreationForm(data = request.POST)
        if form.is_valid():
            new_user = form.save()
            login(request, new_user)
            return redirect('todos:index')
    context = {'form': form}
    return render(request, 'registration/register.html', context)
```

users/templates/registration/register.html

```
{% extends 'todos/base.html' %}
{% load bootstrap4 %}
{% block content %}
    <form method = "post" action = "{% url 'users:register' %}"class= "form">
        {% csrf_token %}
        {% bootstrap_form form %}
        {% buttons %}
        <button name="submit" class="btn btn-primary">Зареєструватися</button>
        {% endbuttons %}
        <input type="hidden" name="text" value="{% url 'todos:index' %}"/>
    </form>
{% endblock content %}
```

users/templates/registration/login.html

```
{% extends 'todos/base.html' %}
{% load bootstrap4 %}
{% block page_header %}
    <p> My Dear Diary - твій онлайн-щоденник. Керуй своїм часом! </p>
    <p> Ви тут вперше? <a href="{% url 'users:register' %}">Зареєструватися</a></p>
{% endblock page_header %}
{% block content %}
    <p>Введіть логін та пароль</p>
    {% if form.errors %}
        <p>Ваш логін і пароль не співпадають</p>
    {% endif %}
    <form method = "post" action = "{% url 'users:login' %}" class="form">
        {% csrf_token %}
        {% bootstrap_form form %}
        {% buttons %}
        <button name="submit" class="btn btn-primary">Увійти</button>
        {% endbuttons %}
        <input type="hidden" name="next" value="{% url 'todos:index' %}"/>
    </form>
{% endblock content %}
```

users/templates/registration/log_out.html

```
{% extends 'todos/base.html' %}
{% block content %}
    <p> Ви вийшли з облікового запису </p>
{% endblock content %}
```

todo/templates/404.html

```
{% extends 'todos/base.html' %}
{% block page_header %}
<p>Сторінка недоступна. (404)</p>
{% endblock page_header %}
```

todo/templates/500.html

```
{% extends 'todos/base.html' %}
{% block page_header %}
<p>Сервер не працює. (500)</p>
{% endblock page_header %}
```

todo/Procfile

```
web: gunicorn todo.wsgi --log-file -
```

todo/runtime.txt

python-3.9.13

todo/.gitignore

todo_env/

__pycache__/

*.sqlite3