Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ОГЛЯД СУЧАСНИХ ФРЕЙМВОРКІВ ДЛЯ WEB РОЗРОБКИ СЕРВІСУ «ОРГАНАЙЗЕР СТУДЕНТА»

Текстова частина до курсової роботи
за спеціальністю «Комп’ютерні науки»

Виконала:
студентка 3-го курсу
Пєчкурова Е. Я.
Науковий керівник:
ст. викладач Кирієнко О. В.

Київ 2025

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри інформатики,

к. ф.-м. н. С. С. Гороховський

_____ (підпис)

“_5_” __листопада_ 2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студентки Пєчкурової Евеліни Янівни факультету інформатики 3 курсу

Тема: Огляд сучасних фреймворків для web розробки сервісу «Органайзер студента»

Зміст ТЧ до курсової роботи:

Вступ

1. Огляд веб-фреймворків

2. Опис проекту та аналіз необхідних даних для розробки веб-застосунку «Органайзер студента»

3. Реалізація проекту

Висновки

Список джерел

Дата видачі “_1_” __листопада__ 2024р.

Керівник _____(підпис) Завдання отримано _____

ЗМІСТ

КАЛЕНДАРНИЙ ПЛАН КУРСОВОЇ РОБОТИ

№ з/п	ПЕРЕЛІК РОБІТ	Термін виконання	Дата ознайомлення наукового керівника	Підпис наукового керівника	Примітки
1.	Обрання теми, її затвердження на засіданні кафедри та призначення наукового керівника	жовтень 2024р.	20 жовтня 2024р.		
2.	Погодження календарного плану виконання курсової роботи	листопад 2024р.	1 листопада 2024р.		
3.	Розробка плану курсової роботи та погодження його з науковим керівником	листопад 2024р.	10 листопада 2024р.		
4.	Проведення аналізу предметної області та вже готових застосунків подібного типу, інструментів, за допомогою яких вони були створені та відгуків на них	листопад 2024р.	25 листопада 2024р.		
5.	Визначення нефункціональних та функціональних вимог до	листопад 2024р.	25 листопада 2024р.		

	веб-сервісу, ключових вимог до його архітектури				
6.	Вибір актуальних фреймворків, які будуть включені в огляд	листопад 2024р.	30 листопада 2024р.		
7.	Архітектурне проектування сервісу, розробка його загальної концепції, опис основних компонентів та їх взаємозв’язків	грудень 2024р.	20 грудня 2024р.		
8.	Складання технічної специфікації для створення сервісу	грудень 2024р.	20 грудня 2024р.		
9.	Початок створення сервісу, програмування та розробка його компонентів	грудень 2024р.	25 грудня 2024р.		
10.	Завершення роботи над сервісом, тестування його на відповідність функціональним і нефункціональним вимогам	січень 2025р.	2 лютого 2025р.		
11.	Написання теоретичної частини курсової роботи на основі матеріалів, зібраних під час роботи над сервісом	лютий-березень 2025р.	15 березня 2025р.		

12.	Завершення написання кваліфікаційної роботи, її оформлення відповідно до вимог та передача науковому керівнику для отримання відгуку.	квітень 2025р.	30 квітня 2025р.		
13.	Передача курсової роботи на перевірку відповідності вимогам академічної доброчесності в НаУКМА	травень 2025р.	8 травня 2025р.		
14.	Підготовка до захисту курсової роботи на засіданні кафедри: підготовка доповіді та створення презентації.	травень 2025р	Травень 2025р.		
15.	Публічний захист курсової роботи перед екзаменаційною комісією	згідно з роботи ЕК	згідно з роботи ЕК		

Графік узгоджено «_1_» _листопада__ 2024р.

Науковий керівник __Кирієнко Оксана Валентинівна__(ПІБ)

Виконавець курсової роботи___Пєчкурова Евеліна Янівна__(ПІБ)

ПЕРЕЛІК ВИКОРИСТАНИХ ТЕРМІНІВ ТА УМОВНИХ ПОЗНАЧЕНЬ

Автентифікація – процес перевірки особи користувача, що підтверджує, що він є тим, за кого себе видає. Наприклад, введення паролю, який може знати лише власник акаунту.

Авторизація – процес перевірки того, чи має користувач право доступатися до певного ресурсу, або ж виконувати певну дію.

Міксін – спеціальний клас, що використовується для додавання певної функціональності до інших класів без використання механізму успадкування[1].

Опенсорс – програмне забезпечення з відкритим вихідним кодом, який користувачі можуть переглядати, модифікувати та поширювати.

Фулстек (full-stack) – характеристика, що означає здатність одночасно охоплювати створення як серверної, так і клієнтської частини програмного застосунку.

Шаблонізатор – програмне забезпечення, що надає можливість динамічно генерувати HTML-сторінки за шаблонами, які містять змінні та вирази.

API (Application Programming Interface) – набір правил та протоколів, які регулюють комунікацію між різними застосунками та їхніми компонентами[2].

Best practices – набір способів чи методів написання програмного забезпечення, що вважаються найкращими серед усіх інших відомих методів.

CSRF (Cross-Site Request Forgery) – вид атаки на веб-застосунки, який обманом змушує браузер довіреного користувача виконувати дії, вигідні зловмиснику. Атака використовує те, що сервер сприймає запити, надіслані браузером, як дії авторизованого користувача, через автоматичну передачу авторизаційних даних[3].

FK (Foreign Key) – зовнішній ключ, поле або набір полів, що встановлює зв’язок з первинним ключем іншої таблиці[4].

MVT (Model-View-Template) – архітектурний патерн, що містить рівні Model (моделі), View (представлення), Template (шаблони).

NN (Not Null) – обмеження, що вказує на обов'язковість наявності значення поля в базі даних.

N (Null) – обмеження, що вказує на необов'язковість наявності значення поля в базі даних.

on_delete=CASCADE – позначення, що вказує, що при видаленні пов'язаного об'єкта, пов'язані записи також будуть автоматично видалені[5].

on_delete=SET_NULL – позначення, що вказує, що при видаленні пов'язаного об'єкта посилання на нього буде замінене порожнім значенням[5].

ORM (Object-Relational Mapping) – технологія програмування, що пов'язує об'єкти в базі даних з об'єктами в застосунку, реалізованими за допомогою об'єктно-орієнтованого програмування, створюючи таким чином віртуальну базу даних[6].

PK (Primary Key) – первинний ключ, поле або набір полів, що однозначно ідентифікує кожен запис в базі даних[7].

SQL-ін'єкція – атака на веб-застосунки, в процесі якої через поле введення користувача впроваджується шкідливий SQL-запит з метою отримати доступ до бази даних та вкрасти або змінити дані[8].

URL (Uniform Resource Locator) – адреса ресурсу в інтернеті[9].

XSS (Cross Site Scripting) – атака на веб-застосунок, що здійснюється за рахунок вживлення шкідливого скрипту через поле введення користувача. У випадку, коли застосунок ніяк не фільтрує введені дані, браузери інших користувачів застосунку можуть сприйняти скрипт як його частину і виконати його, що дасть зловмиснику можливість викрадати дані та маніпулювати користувацьким інтерфейсом[10].

ВСТУП

Темою моєї курсової роботи є огляд сучасних фреймворків для web-розробки сервісу «Органайзер студента».

Вибір саме цієї теми пояснюється моїм бажанням опанувати професію web-розробника. Для досягнення мети необхідно більш детально ознайомитися з популярними фреймворками, що активно застосовуються для розробки бекенду та фронтенду, а також набути практичного досвіду в створенні повноцінних web-застосунків.

Сучасні web-застосунки можна створювати за допомогою безлічі різних інструментів. В цій роботі буде здійснено огляд обраних фреймворків, досліджено відмінності між ними та можливості, які вони надають, а також обмеження, які вони накладають на розробника.

Моя тема є **актуальною**, оскільки в сучасному світі, в найрізноманітніших сферах людської діяльності, існує великий попит на web-застосунки, які зазвичай розробляються за допомогою спеціальних фреймворків. До того ж обрані мною для розгляду фреймворки – популярні та сучасні, вони активно використовуються в комерційній розробці програмного забезпечення.

Мета дослідження моєї курсової роботи полягає в аналізі фреймворків, які можуть бути використані для створення веб-додатків. Проведений аналіз включатиме в себе визначення критеріїв оцінки фреймворків, дослідження реальних прикладів їх використання та формулювання рекомендацій для розробників.

Об’єктом дослідження моєї курсової роботи є сучасні web-фреймворки, які орієнтовані на написання бекенду, проте надають також можливість фулстек розробки застосунку.

Предметом дослідження є фреймворки Django та Express.js, їх принципи роботи, переваги та недоліки, сфери застосування. Я обрала саме ці фреймворки,

оскільки вони підтримують різні підходи до створення веб-застосунків та реалізовані на різних мовах програмування.

Наукове значення теми моєї курсової роботи полягає в дослідженні та порівнянні принципів роботи різних фреймворків та їхніх можливостей, а також в аналізі впливу на них тієї мови програмування, на якій вони написані. Я розглядаю підходи до написання web-застосунків, підтримувані цими фреймворками.

Практична частина роботи полягає в написанні веб-сервісу «Органайзер студента» з використанням знань, набутих в результаті теоретичного аналізу фреймворків.

Практичним значенням теми моєї курсової роботи є визначення оптимальної сфери використання кожного з проаналізованих фреймворків, огляд його переваг та недоліків, впливу на процес розробки, та надання практичних рекомендацій щодо вибору фреймворку для написання web-застосунку залежно від його функціональних та нефункціональних вимог, перспектив розвитку, масштабності і потреб розробника.

Застосування отриманих знань для вибору фреймворку і створення власного застосунку «Органайзер студента» з максимальним використанням можливостей обраного інструменту стане важливим кроком у моїй професійній діяльності.

РОЗДІЛ 1. ОГЛЯД ВЕБ-ФРЕЙМВОРКІВ
РОЗДІЛ 1.1 ОСОБЛИВОСТІ, СПІЛЬНІ РИСИ ТА КРИТЕРІЇ ПОРІВНЯННЯ

1.1.1. Що таке фреймворк

Фреймворк – це структура, що надає розробнику базове середовище з набором інструментів та компонентів, необхідне для розробки застосунку[11]. Його можна умовно назвати «скелетом» програми, або ж шаблоном, за яким відбувається її розробка[12].

В сучасній IT-індустрії використовується безліч фреймворків, які можна класифікувати за сферою їх застосування. Зокрема, існують фреймворки для веб-розробки, які в свою чергу поділяються на такі, що допомагають в розробці бекенду (Django, Express.js, FastAPI, Spring Boot), або ж фронтенду (Angular, Vue.js, Ember.js)[11][12].

Незалежно від сфери застосування, головна мета будь-якого фреймворку полягає в тому, щоб дозволити розробнику зосередитися на логіці конкретного застосунку, не відволікаючись на створення базового для даної сфери функціоналу з нуля. Таким чином пришвидшується процес розробки, код стає масштабованим, безпечним, якісним та зрозумілим для будь-кого, знайомого з фреймворком[12][13].

Однак варто додати, що використання фреймворків створює і додаткові труднощі. По-перше, лише знання мови, якою написаний фреймворк, недостатньо. Для його ефективного використання потрібно витратити час на вивчення саме фреймворку, принципів його роботи та використаних в ньому парадигм[13]. По-друге, оскільки сучасні фреймворки часто оновлюються, виникає потреба в оновленні написаних з їх допомогою застосунків. Це потрібно робити для підтримки сумісності з сучасними технологіями, а також – для підвищення рівня безпеки застосунку[14].

І все ж переваги використання фреймворків переважають недоліки, про що свідчить їх популярність у всіх сферах IT-індустрії.

1.1.2. Веб-застосунки

Веб-застосунки – це застосунки, що реалізують клієнт-серверну архітектуру, де клієнтом виступає веб-браузер користувача, а сервером – веб-сервер. До їх переваг належать простота використання та доступність. Для доступу до такого застосунку з будь-якого пристрою зазвичай достатньо мати сучасний браузер. Завдяки цьому користувачу не потрібно встановлювати додаткове програмне забезпечення, а розробнику – створювати різні версії застосунку для різних операційних систем[15].

Одним з популярних способів імплементації веб-застосунків є SPA (single-page application, односторінкові застосунки). Основна ідея SPA полягає в тому, що користувач взаємодіє з єдиною веб-сторінкою, що завантажується один раз, після чого динамічно оновлюється відповідно до його дій. Перевагою такого підходу є те, що не потрібно завантажувати нові сторінки з серверу в процесі роботи застосунку, що підвищує продуктивність, економить трафік та робить досвід користувача більш динамічним[16]. Такі популярні веб-застосунки як Gmail, Trello та Google Maps реалізовані саме як SPA.

PWA (progressive web apps, прогресивні веб-застосунки) – інший популярний підхід до розробки веб-застосунків. Створені таким чином застосунки можна локально встановлювати на пристрій. Вони мають можливість працювати в офлайн-режимі та взаємодіяти з функціями пристрою, а отже поєднують в собі переваги нативних застосунків та веб-застосунків[17].

Проте, який би підхід не було обрано, деякі завдання виникатимуть в усіх веб-застосунках: маршрутизація, взаємодія з базою даних, автентифікація та

авторизація користувачів, захист від різних типів атак… Контрпродуктивно починати побудову кожного веб-застосунку з нуля, самостійно імплементуючи базові компоненти. Тут в нагоді стають веб-фреймворки, які беруть на себе реалізацію повторюваних завдань і дозволяють розробникам зосередитися на унікальних аспектах застосунку.

1.1.3. Вплив веб-фреймворків на процес розробки

Зазвичай веб-застосунок складається з двох основних компонентів: бекенду та фронтенду. Бекенд уособлює серверну частину застосунку. Він відповідає за виконання функцій застосунку, забезпечуючи збереження, обробку і передачу даних. Фронтенд, або ж клієнтська частина, створює інтерфейс, з яким взаємодіє користувач у своєму браузері. Такий розподіл завдань в застосунках вплинув на розвиток веб-фреймворків: вони бувають орієнтовані на розробку лише бекенду, лише фронтенду, або ж обох компонентів застосунку одночасно[18].

Фреймворки, орієнтовані на розробку бекенду, надають інструменти для керування маршрутизацією, взаємодії з базами даних, обробки HTTP-запитів, валідації вхідних даних, захисту від різних видів атак та підвищення продуктивності[11][14]. А фреймворки, орієнтовані на фронтенд, допомагають створити більш інтерактивний і адаптивний інтерфейс користувача. Вони надають розробнику можливість створювати багаторазові компоненти інтерфейсу, реалізуючи таким чином компонентну архітектуру, а також контролюють обробку подій та введення даних на стороні клієнта[11][14].

Існує багато способів організації взаємодії між бекендом та фронтендом: вони можуть бути незалежними застосунками, що взаємодіють виключно через API, або ж бути тісно пов’язаними, наприклад, при використанні серверного рендерингу (SSR), коли бекенд здійснює генерацію HTML-шаблонів для фронтенду. Кожен з

цих підходів має свої переваги та недоліки і є оптимальним для розробки застосунків певного типу і масштабу[19].

Велика кількість можливих архітектурних рішень призводить до певних труднощів в категоризації фреймворків. Наприклад, Django – це фреймворк, більшість можливостей якого (робота з базами даних, обробка запитів, маршрутизація, надана за замовчуванням адмін-панель та багато інших) спрямовані на швидке створення робочого, масштабованого бекенду. Через це в деяких джерелах він фігурує як фреймворк для розробки бекенду[11].

З іншого боку, один з наданих ним інструментів – це Django Template Language, вбудований шаблонізатор, який дозволяє створювати HTML-шаблони та передавати їх клієнту. Таким чином, за допомогою Django можна реалізувати як тільки бекенд, так і весь застосунок, що робить його визначення як фулстек-фреймворку більш коректним.

З розглянутих в цій роботі фреймворків тільки Django має вбудований шаблонізатор. Він також дозволяє інтегрувати сторонні інструменти, проте ця можливість використовується рідше. Express.js не має вбудованого шаблонізатору, проте може взаємодіяти з такими сторонніми інструментами як EJS, Twig, Pug. Таким чином, ці фреймворки є орієнтованими на бекенд, але завдяки інтеграції з шаблонізаторами вони можуть реалізовувати серверний рендеринг HTML для створення фронтенду.

Усі зазначені вище переваги та недоліки фреймворків цілком справедливі й для веб-фреймворків. Проте окрім цього стандартного набору, веб-фреймворки мають ще і характерні саме для них вади. Деякі з них (наприклад, Django) нав’язують розробнику певну архітектуру застосунку, що може бути недоречним в ситуаціях, коли потрібно імплементувати проект з нестандартними вимогами, наприклад, з нестандартним підходом до обробки даних або організації взаємодії між компонентами. Також, велика кількість вбудованих функцій і залежностей

всередині веб-фреймворків призводить до того, що надані ними інструменти можуть споживати більше ресурсів серверу, ніж аналогічні кастомні рішення, реалізовані без використання фреймворків[14]. Ці недоліки не є критичними, проте їх варто враховувати, обираючи інструменти для реалізації проекту.

1.1.4. Критерії огляду фреймворків

Оскільки темою моєї курсової роботи є огляд сучасних веб-фреймворків, я вважаю необхідним визначити єдині критерії, які буде використано для їх аналізу та порівняння. Це дозволить детально співставити фреймворки, виявити їхні спільні та відмінні риси, переваги та недоліки відносно одне одного, а також сформувати цілісне уявлення про сучасну веб-розробку шляхом огляду запропонованих ними рішень. Нижче наведено перелік обраних мною критеріїв, з поясненням їхнього змісту та аргументацією їхньої практичної користі при аналізі та виборі фреймворків.

- **Походження та загальна інформація**

Для подальшого аналізу фреймворку важливо розуміти мову програмування, на якій він був написаний, та її вплив. Також слід врахувати інформацію про розробника фреймворку, його політику щодо оновлення і версій, чи є цей фреймворк з відкритим кодом і яку він використовує ліцензію.

- **Доступні інструменти**

Вбудовані в фреймворк інструменти фактично визначають його можливості та типи застосунків, які можна створити з його допомогою. Саме тому при виборі фреймворку важливо врахувати, чи задовільняє наданий ним функціонал вимоги проекту. Необхідно взяти до уваги підтримку інтеграції зі сторонніми інструментами, адже це може значно розширити можливості фреймворку.

- **Гнучкість**

Фреймворк може підтримувати певні парадигми програмування, архітектурні патерни та використання конкретних компонентів, або ж навпаки, надавати розробнику максимальну свободу вибору. Серед цих підходів немає правильного чи неправильного, є лише більш оптимальний для кожного конкретного застосунку. Здійснювати вибір фреймворку слід врахувавши його стандартний підхід до побудови застосунку, доступний рівень вибору та кастомізації компонентів, і за потреби – можливість впровадження незвичних рішень.

- **Безпека**

Веб-застосунки повинні гарантувати високий рівень безпеки для захисту власних даних і даних користувачів. Використання веб-фреймворків може значно спростити процес написання захищеного застосунку, якщо обраний фреймворк бере на себе часткове забезпечення безпеки, наприклад, впроваджує захист від різних типів атак (SQL Injection, CSRF, XSS), чи надає надійні інструменти для авторизації та автентифікації[20].

- **Практичне використання фреймворку**

Щоб завершити формування цілісного уявлення про веб-фреймворк, важливо врахувати, як він використовується на практиці. Інформація про популярність фреймворку серед розробників, сфери веб-розробки, в яких він використовується найчастіше та популярні застосунки створені з його допомогою дають розуміння перспектив роботи з фреймворком та його позиції в сучасній веб-розробці.

У наступних розділах буде детально розглянуто обрані фреймворки – Django та Express.js, опираючись на сформульовані вище критерії.

РОЗДІЛ 1.2. ОГЛЯД DJANGO

1.2.1 Походження Django та загальна інформація

Django – це популярний веб-фреймворк для фулстек-розробки, написаний на мові програмування Python. Створений у 2003 році Саймоном Віллісоном та Адріаном Головатим, американцем українського походження. Назва фреймворку відсилає до відомого джазового гітариста Джанго Рейнхардта (Django Reinhardt). У липні 2005 року відбувся перший публічний реліз Django, і з того часу ведеться безперервна робота над його покращенням. В наш час фреймворк перебуває під опікою команди волонтерів з усього світу на чолі з Django Software Foundation[21].

Регулярний випуск нових удосконалених версій фреймворку потребує чіткої стратегії їх впровадження та підтримки. Згідно з офіційною документацією Django, функціональні релізи виходять з періодичністю близько 8 місяців, а проміжні patch-релізи випускаються за потреби для виправлення багів, що були виявлені після впровадження основного релізу в продакшен. Підтримка останньої офіційно випущеної версії фреймворку передбачає виправлення як помилок в нових функціях, так і порушень роботи в раніше функціонуючих елементах, а також усунення багів, що спричиняють проблеми з безпекою та втрату даних. Так звана «розширена підтримка», що забезпечує лише вирішення проблем з втратою даних та безпекою, розповсюджується на дві попередні версії фреймворку[22].

Регулярні оновлення Django та його строга політика підтримки лише трьох останніх релізів забезпечують сучасність та постійний розвиток, однак можуть стати перешкодою при створенні великих проектів, що залежать від незмінності певного функціоналу або API фреймворку. В таких випадках корисними є релізи з LTS (long-term support, довготерміновою підтримкою). Такі версії гарантовано отримують розширену підтримку на визначений період часу, що зазвичай становить

3 роки[22]. Як наслідок, вони стають більш стабільними, і їх використання дозволяє зосередитися саме на проекті, а не на постійній інтеграції з оновленнями Django. Однак, використання LTS-релізів має і недоліки: відсутність доступу до нових функцій фреймворку та потреба в глобальному оновленні проекту після завершення їх підтримки.

Django використовує ліцензію BSD з трьома пунктами («Нову» або «Переглянуту»)[21]. Це означає, що фреймворк є з відкритим кодом (його повний вихідний код доступний за посиланням https://github.com/django/django/) і будь-хто має право використовувати та модифікувати його залежно від своїх потреб. Користувачі можуть змінювати ліцензію для похідних робіт, що робить Django привабливим вибором як для відкритих, так і для комерційних проектів. Вимагається лише зберегти текст оригінальної ліцензії в новому програмному забезпеченні та не використовувати імена оригінальних авторів для реклами свого продукту[23].

Спектр можливостей Django, його принципи роботи та філософія значною мірою були визначені мовою програмування Python. Python – інтерпретована мова, в якій код спершу компілюється в байт код, а потім виконується інтерпретатором під час запуску програми. Такий підхід до виконання відрізняється від обраного компільованими мовами, де код заздалегідь перекладається до машинного і зберігається в такому вигляді[24]. Django унаслідував від Python і переваги, і недоліки, характерні для інтерпретованих мов. З одного боку, всі зміни внесені в код проекту на основі Django, відразу ж відображаються в роботі прототипу застосунку, що полегшує процес розробки. З іншого боку, написаний на Django застосунок зазвичай працює повільніше порівняно з аналогами, створеними за допомогою компільованих мов. У процесі роботи такого застосунку, наприклад при обробці запитів у реальному часі, інтерпретатор Python щоразу буде виконувати відповідний байт код, що збільшить навантаження на сервер.

1.2.2 Доступні в Django інструменти

Django презентує себе як фреймворк, що надає можливість за кілька годин перейти від ідеї до базового робочого прототипу застосунку[25]. Необхідна для такого розгортання концепції потужність забезпечується набором вбудованих компонентів та інструментів розробки, що впроваджують високий рівень абстракції та спонукають розробника до написання модульного, якісного коду. В основі філософії Django лежить принцип розробки програмного забезпечення DRY (Don't repeat yourself, не повторюйся), що закликає уникати дублювання даних та коду шляхом створення єдиного опису кожної логічної одиниці в межах застосунку[26]. Таким чином мінімізується ризик виникнення помилок, пов'язаних з несинхронними змінами з різних частинах коду, спрощується підтримка та масштабування проекту. В цьому підрозділі детально розглянуто ключовий функціонал фреймворку, з поясненням механізмів його роботи та прикладами реалізації принципу DRY.

- **Маршрутизація**

Django надає розробникам інструменти для створення логічної системи маршрутів у веб-застосунку, передбачаючи при цьому можливість подальшого масштабування та підтримку принципу DRY. Визначення кореневого модулю маршрутизації відбувається в файлі **settings.py** через параметр **ROOT_URLCONF**. В Django діє угода про найменування, згідно з якою такий модуль реалізується в файлі **urls.py** та визначає змінну **urlpatterns**. Отримавши запит від клієнта, фреймворк послідовно перевіряє перелічені в **urlpatterns** маршрути в пошуках першого, чий паттерн відповідає заданому URL[27].

Ключовим інструментом для створення маршрутів є функція **path()**, яка надає можливість як пов'язати певний шаблон URL з обробником запитів, так і присвоїти

отриманому маршруту ім’я. Однак справжня цінність механізму маршрутизації Django полягає в низці розширених можливостей. Наприклад, динамічність оброблюваних запитів забезпечується змінними шляху, що виділяють певні частини URL-адреси, присвоюють їх значення та передають їх як аргументи до відповідних обробників. Фреймворк підтримує типізацію таких змінних, додаючи рівень валідації та підвищуючи чіткість визначення шаблонів. Для ще більш точного опису складних URL-паттернів існує функція **re_path()**, що використовує регулярні вирази для їх визначення[27].

Написання повноцінного веб-застосунку на Django передбачає створення окремих app (структурних підпрограм), відповідальних за власні логічні сегменти функціоналу. З метою підтримки масштабованості та гнучкості маршрутизації, кожна з таких app здатна визначати власну підсистему маршрутів. Об’єднання підсистем в єдину структуру відбувається за допомогою функції **include().** Подібний підхід гарантує модульність системи та можливість повторного використання коду. Наприклад, кілька екземплярів одної app можуть співіснувати в застосунку, підключаючись до системи URL в різних місцях[28].

Характерною особливістю Django, що відрізняє його від Express.js, є вбудована можливість зворотної резолюції URL. Фреймворк надає функцію **reverse(),** яка, отримавши на вхід ім’я маршруту, визначене через параметр **name** в функції **path()** або **re_path()**, та, за потреби, значення змінних в його шаблоні, генерує відповідний URL[27].

Підтримка зворотної резолюції є прикладом імплементації принципу DRY, згідно з яким структура URL повинна визначатися лише один раз в файлі маршрутизації, а не кодуватися вручну в різних частинах програми. Завдяки цьому, внесення змін до структури URL потребує редагування єдиного файлу, замість ризикованого і трудомісткого процесу пошуку необхідних посилань в коді фронтенду та бекенду.

Django пріоритезує надання інструментів для створення модульних та масштабованих застосунків. Саме тому кожна app розглядається як автономний компонент, який можна легко підключати або виключати з проекту. Ризик виникнення конфлікту імен, коли кілька маршрутів в різних app мають однакові атрибут **name** та кількість і типи змінних, суперечить цьому підходу. Така ситуація унеможливлює коректну роботу функції **reverse()**, а спроби не допустити її виникнення призведуть до підвищення рівня залежності між компонентами застосунку. Для вирішення цієї проблеми фреймворк пропонує механізм створення унікального простору імен як для app в цілому, так і для їх конкретних екземплярів, що забезпечує точну ідентифікацію маршрутів[27].

Django, на відміну від Express.js, не взаємодіє з методами HTTP на рівні маршрутизації. Тип запиту (GET, POST, PUT, DELETE та ін.) передається безпосередньо до обробника і не впливає на відповідність шаблону. Як наслідок, спрощується процес побудови схеми URL, проте виникає потреба в додатковій логіці в обробниках, що може призвести до їх перевантаження. Для побудови застосунків, де необхідна чітка взаємодія з HTTP-методами (наприклад, при створенні RESTful API), Django надає можливість використовувати Django REST Framework для компенсації відсутності вбудованого функціоналу[27].

- **ORM**

Django надає вбудований механізм ORM для взаємодії з базою даних за допомогою коду Python та парадигми об’єктно-орієнтованого програмування. Це спрощує написання модульного коду та дозволяє абстрагуватися від деталей імплементації конкретної системи управління базами даних. Django реалізує архітектурний патерн MVT (Model-View-Template), де Model – клас, відповідальний за відображення структури та поведінки сутностей бази даних в коді[29]. За конвенцією, моделі створюються в файлі **models.py** шляхом

наслідування наданого фреймворком класу **Model**. Кожна модель визначає окрему сутність, її атрибути відображають поля відповідної таблиці[30]. Вбудований механізм ORM налаштовано за замовчуванням, проте його поведінку можна змінити, використавши конфігураційний клас **Meta**[31], або перевизначивши спеціальні методи, зокрема **save()** – для зміни логіки збереження екземпляру в базі даних, або **clean()** –для встановлення спеціальних правил валідації.[32]

Визначені в коді Django моделі транслюються до бази даних за допомогою системи міграцій. Вони генеруються командою **python manage.py makemigrations** та застосовуються до бази даних за допомогою команди **python manage.py migrate**[33].

Фреймворк підтримує взаємодію з багатьма реляційними базами даних. При ініціалізації нового застосунку за замовчуванням створюється підключення до SQLite, проте також підтримуються PostgreSQL, MySQL, MariaDB[34]. Після того, як база даних належним чином налаштована, розробник може взаємодіяти з нею виключно через систему високорівневих ORM-запитів, що підтримують широкий спектр операцій.

- **Представлення**

Представлення (також відомі як views) реалізують рівень View, що відповідає за обробку запитів від користувачів та формування відповідей в підтримуваному Django MVT патерні[29]. Цей рівень реалізує логіку застосунку: отримавши запит, виражений об’єктом HttpRequest, view здійснює набір даних з моделей, опрацьовує їх та передає відповідь користувачу у вигляді html-сторінки або безпосередньо, наприклад у форматі JSON[35].

В своєму базовому варіанті view – це функція, яка приймає об’єкт запиту, формує контекст та рендерить шаблон. Обов’язковою вимогою є повернення об’єкта HttpResponse. В найпростішому варіанті view може повертати статичну

відповідь, а також – спеціальні типи відповідей, наприклад коди помилок http. Кожен view в Django прикріплюється до певного URL-маршруту і стає обробником запитів до нього [35]. Додаткове налаштування таких представлень здійснюється за допомогою вбудованих в Django декораторів. Таким чином можна обмежити доступ до певного view тільки для аутентифікованих користувачів або задати додаткові умови обробки запитів[36].

Однак в масштабних проектах зі складною логікою обробки запитів, функціонального підходу до створення представлень може виявитися недостатньо. Тому Django пропонує реалізацію views у вигляді класів (class-based views) як альтернативний варіант[37]. Цей підхід дозволяє організувати код обробки запитів у вигляді окремих методів класу (**get()**, **post()**, **put()**, тощо), що покращує його читабельність та полегшує модифікацію порівняно з функціональним варіантом, який потребує складних конструкцій керування потоком (**if request.GET** … **elif request.POST** …). Завдяки використанню класів схему представлень можна ефективно розбудовувати, використовуючи механізми наслідування і міксінів, повною мірою застосовуючи переваги методології MRO (Method Resolution Order) у Python.

Всі view-класи в Django наслідують від базового класу View, який забезпечує інтеграцію з системою маршрутизації URL, делегування обробки HTTP-методів та інші базові функції[37].

Для ще більшого спрощення процесу розробки Django надає набір готових шаблонів представлень для типових задач (generic views). Ці шаблони були створені, щоб спростити реалізацію стандартних сценаріїв, таких як перегляд списку об’єктів, перегляд деталей окремого об’єкта, створення, редагування та видалення об’єктів[38].

Функціонування generic views часто налаштовується через наслідування: використовуючи базовий шаблон як батьківський клас, розробник може

перевизначити окремі його атрибути або методи, щоб вони відповідали вимогам конкретного застосунку, зберігаючи таким чином баланс між швидкою розробкою та гнучкістю[38].

Таким чином, система представлень Django надає розробникам як прості інструменти для створення базових застосунків, так і ефективні рішення для побудови складної логіки, поступово переходячи від функціональної до об'єктно-орієнтованої парадигми програмування, максимально використовуючи переваги цих парадигм та їхню реалізацію в мові Python.

- **Django Template Language**

Django надає вбудований шаблонізатор – Django Template Language (DTL) – що відповідає його філософії «batteries included». Рушій шаблонів інтегровано безпосередньо в архітектуру Django, завдяки чому розробники можуть генерувати HTML-сторінки без додаткових налаштувань та підключення сторонніх бібліотек[39]. Таким чином реалізується Template рівень патерну MVT, відповідальний за представлення даних користувачу[29].

Основними елементами шаблонів є змінні, на місце яких в процесі рендерингу підставлятимуться значення та теги, що дозволяють використовувати базові конструкції програмування. Наприклад, умовні розгалуження (if, else), цикли (for), визначення тимчасових змінних (with). Змінні в шаблоні визначаються за допомогою подвійних фігурних дужок **{{ }}**, теги – через **{% %}**. Їх наявність допомагає визначати шаблони, значно більш лаконічні та універсальні, ніж звичайні HTML-сторінки. Приміром, тег **{% url %}** підтримує зворотню резолюцію маршруту за іменем, забезпечуючи таким чином, дотримання принципу DRY при роботі з посиланнями в шаблонах[39].

Наявність тегів **block**, **extends** та **include** надає DTL можливість встановлювати зв’язки між шаблонами через наслідування та агрегацію, виводячи

таким чином їх гнучкість, придатність до повторного використання та здатність компактно описувати структуру сторінок на якісно новий рівень. Тег extends дозволяє шаблону наслідувати інший шаблон, змінюючи при цьому вміст певних компонентів, визначених за допомогою тегу **block**[39]. Як наслідок, обов'язкові елементи HTML-сторінки – такі як **<head>**, **<body>**, **<navbar>** – можуть бути описані один раз в базовому шаблоні та успадковуватися іншими сторінками без дублювання коду. Тег **include**, зі свого боку, реалізує механізм агрегації, дозволяючи додавати шаблон як складову частину до інших шаблонів. Завдяки цьому такі елементи як sidebar чи navbar можуть бути винесені в окремі файли і багаторазово використані в різних частинах застосунку[40].

При проектуванні DTL було взято до уваги принцип розмежування повноважень (англ. separation of concerns). У патерні MVT рівень Template відповідає виключно за представлення даних користувачу, тому елементи програмування в шаблонах повинні використовуватися лише для їх коректного відображення. Уся обробка та модифікація даних повинні відбуватися на рівні Views. З цієї причини DTL навмисно обмежує можливості роботи з даними: наприклад, в шаблоні неможливо звернутися до елемента словника за змінним ключем[39].

З іншого боку, певна обмежена логіка, спрямована на підготовку даних для якіснішого їх представлення, є необхідною. Щоб позбавити розробника потреби імплементувати її через обмежений набір доступних конструкцій, DTL надає механізм фільтрів, які застосовуються до змінних в шаблоні за допомогою вертикальної риски | і дозволяють виконувати типові завдання, такі як зміна регістру тексту, перетворення форми слова (однина/множина), обмеження кількості символів або форматування дат[40]. У випадку, коли наданих Django фільтрів недостатньо, розробник має можливість визначити та використати власні, користуючись при цьому повним арсеналом можливостей мови Python. Таким

чином можна навіть обійти обмеження закладені в архітектурі DTL і додати в шаблон складну логіку обробки даних, що, однак, порушить принцип розмежування повноважень.

1.2.3 Гнучкість Django

Фреймворк Django підтримує чітко визначену структуру застосунку та орієнтується на використання архітектурного патерну MVT[29]. Надані «з коробки» компоненти та механізми потребують наявності моделей, представлень та шаблонів для коректного функціонування.

Зокрема, якщо розробник вирішує відмовитися від рівня моделей на користь «сирих» SQL-запитів, багато вбудованих в фреймворк функцій стають недоступними. І мова йде не тільки про використання ORM. Системи автентифікації та авторизації, проміжне програмне забезпечення, сесії, обробка форм також залежать від наявності моделей. У таких випадках фреймворк втрачає свою ефективність і навіть може перешкоджати розробці.

З іншого боку, Django не є абсолютно негнучким. Розробник має можливість змінювати поведінку вбудованих компонентів перевизначаючи методи або успадковуючи класи. До того ж, завдяки мові програмування Python, фреймворк підтримує розширення та адаптацію функціоналу шляхом використанням механізму міксінів. Django також здатен ефективно використовувати архітектурний патерн MVC (Model-View-Controller), розширений варіант MVT, де замість побудови шаблонів та серверного рендерингу здійснюється передача даних на фронтенд-фреймворки.

Можна зробити висновок, що Django найкраще підходить для реалізації застосунків, котрі дотримуються рекомендованої архітектури, хоч і допускає кастомізацію своїх компонентів.

1.2.4 Безпека Django

Безпека в Django імплементована відповідно до принципу «batteries included», що орієнтується на надання розробнику широкого набору вбудованих інструментів для підтримки швидкої розробки та мінімізації залежності від сторонніх бібліотек. На відміну від Express.js, котрий не надає власних безпекових рішень, Django пропонує готові потужні механізми захисту застосунку вже при створенні нового проекту.

В Django реалізовано наступну ідею: створити з його допомогою небезпечний застосунок складніше, ніж безпечний. Механізми захисту від базових типів атак – CSRF, XSS, SQL Injection – імплементовано в фреймворк та активовано за замовчуванням[41]. Замість «сирих» паролів користувачів, Django зберігає результати їх хешування потужними алгоритмами (PBKDF2, Argon2). Цей процес є незворотнім, і дізнатися пароль неможливо, навіть маючи доступ до бази даних користувачів[42]. Обробка сесій, їх створення, керування та очищення також відбувається автоматично, згідно найкращих безпекових практик. Наприклад, під час виходу користувача з Django-застосунку, стираються всі дані про його сесію, уникаючи таким чином вразливостей через браузер[43].

Було виявлено, що на відміну від Express.js, Django надає файлову структуру та готовий код при створенні нового проекту з його участю. Зокрема – вже реалізовані моделі необхідних Django сутностей, готові до міграції в базу даних. Одна з таких моделей, User, інкапсулює користувача веб-застосунку і лежить в основі всієї системи автентифікації та авторизації. З її допомогою визначаються всі можливі типи користувачів: через атрибути встановлюються права доступу, належність до певної групи та факт буття супер користувачем. Єдиним винятком є об’єкт AnonimousUser, що впроваджується для позначення неаутентифікованого користувача[44].

Систему автентифікації та авторизації Django можна кастомізувати, проте її стандартна конфігурація вже розроблена так, щоб задовольняти базові потреби широкого обсягу проектів. Мова йде про автентифікацію та авторизацію користувачів, керування групами та правами доступу.

Автентифікація користувача відбувається шляхом порівняння введеного набору облікових даних (в базовому випадку імені користувача та пароля) з валідними даними зареєстрованих користувачів через бекенд автентифікації. Django надає функції для синхронного та асинхронного виконання такого порівняння «вручну», проте розробники зазвичай працюють на вищому рівні абстракції, залишаючи імплементацію процесу автентифікації на фреймворк[43].

Що стосується авторизації, то вбудована в Django система дозволяє призначати дозволи на певні дії в застосунку як окремим користувачам, так і групам користувачів. Django автоматично створює базові дозволи для всіх моделей в базі даних. Користувач може належати до необмеженої кількості груп та успадковувати їхні дозволи, що особливо корисно в застосунках з багатьма типами користувачів, що мають різні повноваження[43].

За допомогою сесій та проміжного програмного забезпечення Django в об’єкт request, що інкапсулює запит до серверу, додається атрибут user, з допомогою якого відбувається подальші автентифікація та авторизація. Їх можна імплементувати різними способами: через ручні перевірки в коді (наприклад, перевірка **user.is_authenticated**), за допомогою вбудованих декораторів, таких як **login_required**, **user_passes_test**, а також з використанням вбудованих міксінів, що інтегруються в views-класи[36][43].

Django проектувався з метою забезпечення швидкої розробки, тому він також надає готові views для обробки автентифікації (**LoginView**, **LogoutView**) та управління паролями (**PasswordChangeView**, **PasswordResetView**), що реалізовують стандартні сценарії та легко налаштовуються за потреби[43].

1.2.5 Практичне застосування Django

За результатами опитування, проведеного у 2024-му році, Django є менш популярним, ніж Express.js[45]. Однак він все ще займає високу позицію в рейтингу. До того ж, Django займає почесне сьоме місце серед розробників-початківців, що можна пояснити кількома чинниками[45].

По-перше, розробка застосунку на Django не вимагає підбору та конфігурації окремих компонентів, оскільки фреймворк забезпечує розробника всім необхідним. Це забезпечує швидкий розвиток проектів та дозволяє зосередитися на імплементації основного функціоналу. По-друге, Django має низький поріг входу завдяки чудовій офіційній документації, яка доступно пояснює як базові концепції веб-програмування, так і їхню реалізацію в межах фреймворку.

На практиці, багато провідних компаній обирають саме Django для імплементації своїх застосунків. Наприклад, Instagram, популярна соціальна мережа, використовує фреймворк для реалізації бекенду через його високу масштабованість та здатність підтримувати модульну архітектуру[46][47]. Також деякі сервіси компанії Mozilla базуються на Django, що доводить його придатність для побудови великих, безпечних, підтримуваних застосунків з відкритим кодом[46].

РОЗДІЛ 1.3. ОГЛЯД EXPRESS.JS

1.3.1 Походження Express.js та загальна інформація

Express.js – це мінімалістичний фреймворк для розробки бекенду web-застосунків на платформі Node.js, написаний мовою програмування JavaScript[48]. Він принципово відрізняється від Django тим, що не надає переваги певним патернам проектування чи архітектурним рішенням, натомість даючи розробнику максимальну свободу дій.

Express.js був створений в травні 2010-го року Ті Джеєм Холоваучаком (TJ Holowaychuk). В наш час фреймворк перебуває під опікою OpenJS Foundation, що забезпечує його стабільний розвиток. Його код знаходиться у відкритому доступі на GitHub, за посиланням https://github.com/expressjs/express, а сам фреймворк має ліцензію MIT, що дозволяє його вільне використання, зміни та застосування в закритих комерційних застосунках[48]. Перейшовши за цим посиланням, можна побачити, що Express.js має активну команду вкладників, які невпинно працюють над його поліпшенням. Про підтримуваність фреймворку свідчить те, що його останній на момент написання курсової реліз 5.1.0 був випущений нещодавно, 31-го березня 2025-го року.

Той факт, що Express.js написаний на JavaScript, робить його привабливим вибором для імлементації web-проектів, адже з його допомогою можна реалізувати і бекенд і фронтенд застосунку однією мовою. До того ж, Express.js дає розробникам можливість скористатися неблокуючою моделлю Node.js[49], котра надзвичайно ефективна для обробки великої кількості подій.

1.3.2 Доступні в Express.js інструменти

Express.js є мінімалістичним фреймворком, тобто він надає тільки основу для створення веб-застосунку[50]. Якщо Django намагається забезпечити розробника всім необхідним для розробки додатку, то Express.js обмежується підтримкою маршрутизації і наданням інструментів для інтеграції сторонніх бібліотек, модулів та проміжного програмного забезпечення (middleware) з екосистеми Node.js.

- Маршрутизація

За допомогою Express.js можна кількома рядками коду визначити HTTP-сервер та задати обробники запитів для URL-маршрутів (Рис. 1). З усіх оглянутих фреймворків Express.js єдиний не надає попередньо визначеного коду та файлів при ініціалізації з його допомогою нового застосунку. Проекти, створені за допомогою Django потребують первинної конфігурації та певної файлової структури для створення робочого серверу, в той час як Express.js для виконання аналогічного завдання потрібні лише кілька рядків коду[48].

```
// Importing the Express library.
const express = require('express');

// Initializing the app.
const app = express();

// Getting the path request and sending the response with text.
app.get('/', (req, res) => {
    res.send('Hi, your request has been received');
});

// Listening on port 2000.
app.listen(2000, () => {
    console.log('listening at http://localhost:2000');
});
```

Рис. 1 Код для ініціалізації серверу та встановлення обробника GET-запитів на Express.js

Обробники маршрутів задаються викликами методів, наприклад **app.get()**, **app.delete()**. Таким чином визначається метод HTTP-запитів, що опрацьовуватиметься обробником при надходженні на вказаний маршрут[51].

- **Middleware**

Проміжне програмне забезпечення (далі – middleware) – це програмне забезпечення, що перехоплює запити і відповіді серверу та виконує над ними додаткові дії, перед тим як передати їх до кінцевого обробника маршруту. Exress.js містить такі вбудовані middleware як **express.static()**, **express.json()** – для надання статичних файлів та парсингу json-даних в тілі HTTP-запитів відповідно. Передбачається, що для інтеграції в реалізований за допомогою цього фреймворку застосунок додаткової функціональності – наприклад автентифікації, авторизації, керування сесіями – розробник використовуватиме стороннє проміжне програмне забезпечення[52].

Середовище Node.js містить велику кількість бібліотек та middleware, які реалізують практично всю необхідну в веб-розробці функціональність і додаються до проекту простим завантаженням npm-пакету. До того ж, Express.js надає можливість визначати власне проміжне ПЗ й підключати його до ланцюжка обробки запитів[52].

- **Шаблонізатори**

Express.js не надає власного шаблонізатору, що відрізняє його від Django, який пропонує розробникам використовувати власний Django Template Language. Фреймворк також не має єдиного оптимального сценарію взаємодії з фронтендом, з однаковою ефективністю даючи можливість реалізувати як Server Side Rendering за допомогою шаблонізаторів, так і взаємодію з сучасними фронтенд-

фреймворками. Залежно від потреб конкретного проекту, Express.js можна поєднати з великою кількістю шаблонізаторів, наприклад з Twig, EJS або LiquidJS.

1.3.3 Гнучкість Express.js

Мінімалістична філософія Express.js призводить до того, що на проекти, реалізовані з його допомогою, не накладаються архітектурні або структурні обмеження. Фреймворк не підтримує жодного патерну та не вимагає чіткої структури файлів та папок[50]. Вибір на користь певних архітектурних рішень ніяк не впливає на ефективність саме Express.js, не погіршує її, як це може статися, якщо, наприклад відмовитися від патерну MVT в Django, але і не покращує. Наприклад, Express.js однаково добре підтримує взаємодію як з реляційними, так і з нереляційними базами даних.

Express.js не містить власного ORM, шаблонізатору, системи автентифікації та авторизації чи адміністративного інтерфейсу. Розробник має самостійно обрати необхідні компоненти проекту, потурбувавшись про їх налаштування, сумісність та взаємодію. Для досвідченого розробника, котрий має чіткі побажання до технічного стеку свого застосунку, це є перевагою, проте для початківців або розробників, які реалізують проект зі стандартними вимогами, потреба аналізувати та обирати серед багатьох рішень може виявитися недоліком.

Відсутність стандартизації в Express.js може стати проблемою, якщо розробник безвідповідально ставиться до структури свого проекту. Відсутність будь-якої регуляції створює потенціал для написання неструктурованого, нечитабельного та непідтримуваного коду. З цієї причини, багато реалізованих на Express.js застосунків все ж слідують певному патерну або структурі[52].

1.3.4 Безпека Express.js

Express.js не пропонує вбудованих інструментів для підвищення рівня безпеки розроблених з його допомогою веб-застосунків. Фреймворк є надзвичайно гнучким та надає розробнику свободу в підборі інструментів, тому в ньому немає вбудованого захисту від різних типів атак, автоматичного шифрування паролів чи системи автентифікації та авторизації. Проте цю відсутність механізмів «з коробки» вдається компенсувати завдяки великій кількості перевірених сторонніх middleware. Оскільки Express.js є популярним інструментом для веб-розробки, в його спільноті вже сформувалися найкращі практики, а підключення необхідного middleware відбувається швидко та без ускладнень[53].

Наприклад, використання ORM знижує ризик SQL-injection, а можливість інтеграції з сучасними шаблонізаторами компенсує той факт, що Express.js автоматично не санітизує введені користувачем дані. Використання бібліотеки Helmet.js для встановлення HTTP-заголовків також захищає застосунок від ряду пов’язаних загроз[53].

Механізми автентифікації та авторизації слід імплементувати за допомогою сторонніх компонентів. Приміром, проміжне програмне забезпечення Passport.js підтримує всі сучасні методи автентифікації, включно з входом в систему через соціальні мережі[54].

Загалом, написані на Express.js застосунки можуть цілком відповідати сучасним стандартам безпеки. Проте імплементація відповідного рівня захисту вимагає від розробника більше зусиль та знань, ніж у випадку використання фреймворків, котрі надають вбудовані рішення, наприклад Django.

1.3.5 Практичне застосування Express.js

Фреймворк Express.js, попри свою мінімалістичність та обмежений вбудований функціонал, є популярним серед розробників і використовується провідними компаніями як основа для реалізації веб-застосунків. Основними його перевагами є висока гнучкість, ефективна обробка асинхронних запитів та продуктивність, яка зазвичай визначається характеристиками використаних компонентів.

Опитування показали, що в 2024-му році Express.js був популярнішим за Django. До того ж, він зайняв третє місце за популярністю серед розробників-початківців, що пояснюється його простотою[45]. Express.js краще ніж Django підходить для опанування базових концепцій веб-програмування, оскільки він дає можливість створювати робочі застосунки початкового рівня без додаткового навантаження у вигляді складної файлової структури.

Доказом популярності цього фреймворку є його використання на практиці. Приміром, компанія Uber[55] в своєму застосунку для пошуку, виклику та оплати таксі або приватних водіїв, для ефективного функціонування якого критично важливо своєчасно обробляти запити, використовує Express.js для імплементації взаємодії з клієнтами в реальному часі та API-запитів[46][52]. Іншим прикладом є міжнародна електронна платіжна система PayPal[56]. Ця компанія здійснила міграцію з Java до Express.js для підвищення продуктивності і досягнення вищого рівня гнучкості[46].

Ці приклади демонструють, що Express.js є ефективним, сучасним, популярним інструментом для створення веб-застосунків.

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі курсової роботи сформульовано визначення фреймворку та досліджено його роль в веб-програмуванні. Для подальшого аналізу було обрано два вживані на ринку веб-фреймворки – Django та Express.js, – а також встановлено критерії для їх подальшого порівняння та оцінки.

Проведений аналіз продемонстрував, що жоден з фреймворків не є універсально кращим за інший. Всі вони мають свої особливості, що можуть бути як перевагами, так і недоліками, залежно від потреб конкретного проекту.

Django є оптимальним рішенням для застосунків, що реалізують архітектурний патерн MVT. Цей фреймворк надає вбудовані інструменти для імплементації необхідного функціоналу, чим пришвидшує та полегшує розробку. З іншого боку, він не підходить для застосунків з нетиповою архітектурою через свою обмежену гнучкість.

Express.js дотримується протилежного підходу. В нього вбудовано необхідний для реалізації серверу та маршрутизації мінімум. Передбачається, що розробник самостійно обиратиме архітектуру, структуру та сторонні компоненти застосунку. Такий підхід надає розробнику повну свободу, але і покладає на нього велику відповідальність.

РОЗДІЛ 2. ОПИС ПРОЕКТУ ТА АНАЛІЗ НЕОБХІДНИХ ДАНИХ ДЛЯ РОЗРОБКИ ВЕБ-ЗАСТОСУНКУ «ОРГАНАЙЗЕР СТУДЕНТА»

РОЗДІЛ 2.1. ОБГРУНТУВАННЯ ВИБОРУ DJANGO

2.1.1 Первинні етапи вибору фреймворку

Першим кроком в web-розробці сервісу «Органайзер студента» став вибір фреймворку для його реалізації на основі проведеного раніше аналізу. Оскільки практичне значення цієї курсової роботи, поміж іншого, полягає в наданні рекомендацій щодо обрання фреймворку для написання web-застосунку, в даному розділі цей процес детально задокументовано, а кінцевий вибір – обґрунтовано.

Фактори, від яких залежить вибір фреймворку, можна поділити на такі, що стосуються розробника (його технічний бекграунд, час, необхідний на опанування), та такі, що визначаються особливостями майбутнього застосунку (запланована архітектура, масштабованість, рівень безпеки, швидкість розробки, функціональні та нефункціональні вимоги, сумісність ідеї застосунку з ліцензією). В контексті цієї курсової роботи, вирішальними стали початкова концепція застосунку, під оптимальну реалізацію якої підбирався інструмент, а також можливість проведення повного аналізу фреймворку в рамках курсової роботи.

Починати аналіз потенційних інструментів розробки слід з пошуку інформації про їхню ліцензію, з’ясування поточного рівня підтримуваності та дослідження обсягу документації й рівня активності спільноти. Після проведення аналізу Django та Express.js було встановлено, що обидва вони активно підтримуються своїми спільнотами, є безкоштовними, надають свій код у відкритому доступі та поширюються на умовах ліцензій, що дозволяють вільне комерційне використання[21][48]. Таким чином, ні один фреймворк не здобув

очевидної переваги на цьому етапі аналізу. Частково це пояснюється темою курсової, адже «Огляд сучасних фреймворків…» передбачає, що всі проаналізовані інструменти будуть актуальними, а отже – популярними та активно підтримуваними. Проте у реальних проектах ситуація може бути зовсім іншою – за умови відсутності попередньо сформованої вибірки фреймворків, важливо одразу відсіяти ті рішення, що не підходять через недостатню підтримку, обмеження ліцензії або брак документації. З цієї причини цей первинний етап аналізу є критично важливим, попри свою неефективність в даному випадку.

2.1.2 Порівняння підтримки ORM у фреймворках

Подальший вибір фреймворку здійснювався з урахуванням потреб веб-сервісу «Органайзер студента», які було виявлено на етапі проектування. Оскільки цінність застосунків такого типу полягає в збереженні та структуризації інформації користувачів, стала очевидною необхідність бази даних. В процесі первинного аналізу предметної області було виділено чіткі взаємопов’язані сутності з складними взаємозв’язками, що потребували комплексної валідації. Як наслідок, вибір було зроблено на користь реляційної бази даних, оскільки її використання передбачає ієрархію сутностей, чітко визначені правила їх зв'язку і контроль над цілісністю.

Використання ORM при роботі з реляційною базою даних є стандартом в сучасних застосунках. Таким чином додається рівень абстракції, що усуває залежність від особливостей роботи конкретної бази даних, спрощується процес обробки даних та зменшується вразливість до SQL-injection[57]. Коли на основі цієї інформації було прийнято рішення, що бекенд сервісу «Органайзер студента» взаємодіятиме з базою даних використовуючи ORM, саме його наявність, зручність

у використанні та функціональні можливості стали першим ключовим критерієм у виборі фреймворку.

В ході попереднього аналізу було встановлено, що Django має вбудований ORM, який підтримує велику кількість операцій з різними базами даних[34], залишаючись при цьому простим в використанні. Якби застосунок потребував активного використання «сирих» SQL-запитів чи нереляційної бази даних, можливості Django були б обмеженими, оскільки цей фреймворк найефективніше працює в рамках MVT-патерну[29]. Проте вибір на користь реляційної бази даних та ORM фактично забезпечив існування в майбутньому web-сервісі рівня Model, роблячи Django привабливим інструментом імплементації на цьому етапі аналізу.

Характерними рисами Express.js є його гнучкість та мінімалістичність[50], тому він не надає власного ORM-рішення. Як наслідок – реалізація «Органайзеру студента» з його допомогою призвела б до додаткових кроків для налаштування та інтеграції сторонніх бібліотек. Використання реляційної бази даних не надає переваги при роботі з Express.js.

В результаті, Django виявився більш доцільним вибором на даному етапі аналізу, оскільки він надає достатній для покриття потреб застосунку та простий в налаштуванні ORM-функціонал.

2.1.3 Порівняння реалізації механізмів безпеки у фреймворках

В процесі проектування сервісу «Органайзер студента» було встановлено, що для реалізації механізмів його безпеки, включно з системами автентифікації та авторизації, доцільно використати готові рішення, запропоновані обраним фреймворком або перевіреними сторонніми бібліотеками. Такий підхід дозволяє не лише пришвидшити розробку, а й знизити ризик критичних помилок, які можуть спричинити витік персональних даних або повне руйнування системи.

Було проаналізовано безпекові потреби органайзеру, зокрема типові загрози, від яких слід захиститися, та функціональність, яку має надавати система автентифікації та авторизації. Вже на етапі проектування стало зрозуміло, що взаємодія користувачів з застосунком відбуватиметься шляхом введення текстових даних. Через це він буде потенційно вразливим до типових веб-атак, наприклад XSS, SQL-injection, CSRF, а отже бажано, щоб обраний для його імплементації фреймворк надавав засоби захисту від подібних атак.

На етапі проектування було вирішено, що застосунок підтримуватиме два типи користувачів – 1) здобувачів освіти, що організовуватимуть з його допомогою свою навчальну діяльність, та 2) адміністраторів, чия задача – модерація та забезпечення коректної роботи сервісу. При цьому здобувачі освіти повинні мати можливість виконувати всі логічно змістовні дії з даними, проте лише в межах власного профілю. Разом з тим, сервіс має бути недоступним для незареєстрованих користувачів, оскільки вся інформація в ньому – персоналізована. Адміністратори, зі свого боку, повинні мати повний доступ до всіх даних з метою моніторингу та модерації. В процесі аналізу цих вимог було встановлено, що вони є типовими для сучасних веб-застосунків, а отже, не потребують створення унікальної системи безпеки. Оглянуті в цій курсовій фреймовки було розглянуто з точки зору їх здатності задовольнити виявлені вимоги.

Exress.js, через свою мінімалістичну природу[50] не має вбудованої системи безпеки, тому реалізувати безпекові вимоги застосунку він може хіба що за допомогою інтеграції з сторонніми бібліотеками. Такий підхід призводить до появи численних зовнішніх залежностей і створення системи безпеки, що складається з неоднорідних компонентів, які потрібно конфігурувати окремо. Для реалізації застосунку типу «Органайзер студента», що потребує повного стандартного набору засобів безпеки функцій, ефективніше використовувати фреймворк з уже вбудованими захисними механізмами.

Django, зі свого боку, якраз пропонує повноцінну вбудовану систему безпеки[14]. До її складу входить захист від типових веб-атак (XSS, CSRF та SQL-Injection), підтримка сесій, шифрування паролів та інтегрована система автентифікації й авторизації, що надає можливість гнучко налаштовувати доступ до компонентів застосунку на рівні представлень[43].

Таким чином, Django є найбільш оптимальним вибором для реалізації безпеки застосунку «Органайзер студента».

2.1.4 Додаткові переваги Django: адмін-інтерфейс та повноцінність

Окрім надання інструментів для роботи з базою даних та наявності системи безпеки, що найкраще підходять для реалізації відповідних компонентів «Органайзеру студента», Django вирізняється рядом додаткових переваг, аналіз яких призвів до остаточного вибору в його користь.

Першою такою перевагою є наявність адміністративного інтерфейсу[58], що автоматично генерується на основі рівня Model та забезпечує можливість управління всіма даними застосунку. В процесі проектування «Органайзеру студента» було встановлено, що для повноцінного та безпечного функціонування застосунку потрібна наявність особливого типу користувачів, адміністраторів, що матимуть повноваження, необхідні для моніторингу стану застосунку, модерації його контенту та контролю за дотриманням правил його використання.

Імплементація інтерфейсу адміністратора була визнана важливим та трудомістким завданням, виконання якого, однак, напряму не впливає на досвід кінцевих користувачів застосунку. Імплементуючи функціонал та інтерфейс користувача, автор курсової відштовхувалася зокрема від власних потреб та потреб своїх колег по навчанню, а також від досвіду користування іншими застосунками

подібного типу. Адміністрування сайтів в цей досвід не входило, до того ж, як вірно відмітили творці Django, написання адміністративного інтерфейсу зазвичай є доволі монотонним завданням, яке не потребує креативного підходу. Тому можливість делегувати це завдання Django стала ще одним аргументом на його користь.

І нарешті, коли постало запитання про спосіб, яким буде реалізовано інтерфейс користувача, було виявлено ще одну перевагу Django: можливість реалізувати повноцінний веб-застосунок виключно його силами, без використання сторонніх інструментів. Це дозволило провести чистий та об'єктивний аналіз Django як засобу для створення веб-сервісу з практичної точки зору, не відволікаючись при цьому на сумісність або взаємодію з іншими технологіями, що дозволило зосередитися виключно на особливостях і можливостях самого фреймворку.

І Django, і Express.js надають можливості як взаємодіяти з frontend-фреймворками (наприклад, через обмін JSON-даними), так і генерувати HTML-сторінки на стороні сервера з використанням шаблонізаторів. Проте, лише Django має вбудований шаблонізатор – Django Template Language[39], що дозволяє реалізувати як бекенд, так і фронтенд застосунку з його допомогою, без сторонніх інструментів. Це стало підставою для прийняття рішення про реалізацію користувацького інтерфейсу «Органайзера студента» за допомогою шаблонів. Такий підхід не лише відповідає рекомендаціям щодо ефективного використання Django (фактично реалізовано оптимальний в його випадку MVT-паттерн[29]), а й забезпечує цілісність та послідовність практичного дослідження в межах курсової роботи, оскільки воно дозволяє оцінювати ефективність фреймворку без впливу зовнішніх технологій.

Таким чином, в результаті проведеного аналізу було доведено, що вибір Django як фреймворку для імплементації «Органайзеру студента» є найбільш

логічним та ефективним рішенням. Це свідчить не про його повну перевагу над іншими оглянутими фреймворками, а лише про відповідність його інструментарію вимогам конкретного прототипу застосунку.

РОЗДІЛ 2.2 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

2.2.1 Визначення ключових сутностей предметної області

Web-розробка сервісу «Органайзер студента» розпочалася з аналізу предметної області, метою якого було визначення оптимального набору сутностей для її представлення. Основне призначення розроблюваного застосунку полягало в організації навчальної діяльності студентів: упорядкуванні та збереженні відповідної інформації, створенні на її основі ефективної системи нагадувань і зручного користувацького інтерфейсу.

На підставі власного досвіду навчання в закладі вищої освіти та шляхом його порівняння з навчальними практиками інших освітніх установ, було визначено чотири логічні сутності, що охоплюють ключові елементи навчального процесу: **предмет**, **заняття**, **форма контролю знань** та **домашнє завдання**. Окрім цього, було визначено сутність **користувач**, яка інкапсулює особу, що взаємодіє з застосунком, зберігаючи як її облікові дані, так і налаштування її профілю.

Гнучкість кінцевого застосунку та підтримка різноманітних моделей навчального процесу були пріоритетними на етапі аналізу предметної області та проектування ER-моделі. Курси, навіть в межах одного закладу вищої освіти, можуть суттєво відрізнятися за структурою: форматом і тривалістю занять, формами контролю знань, а також складністю розкладу. Застосунок мав враховувати ці відмінності без потреби жорсткого налаштування – тобто користувач отримує налаштування за замовчуванням (наприклад, тривалість заняття 90 хвилин), проте може змінювати їх відповідно до власних потреб. Як наслідок, логічна структура бази даних та зв’язки між сутностями були спроектовані з урахуванням потреб адаптації до різних форматів організації навчання.

2.2.2 Структура даних та обмеження

Наведений нижче опис розкриває структуру даних, що лежать в основі предметної області. Типи полів подано відповідно до їх реалізації в системі керування базами даних (PostgreSQL), оскільки вибір типів безпосередньо випливає з вимог до даних.

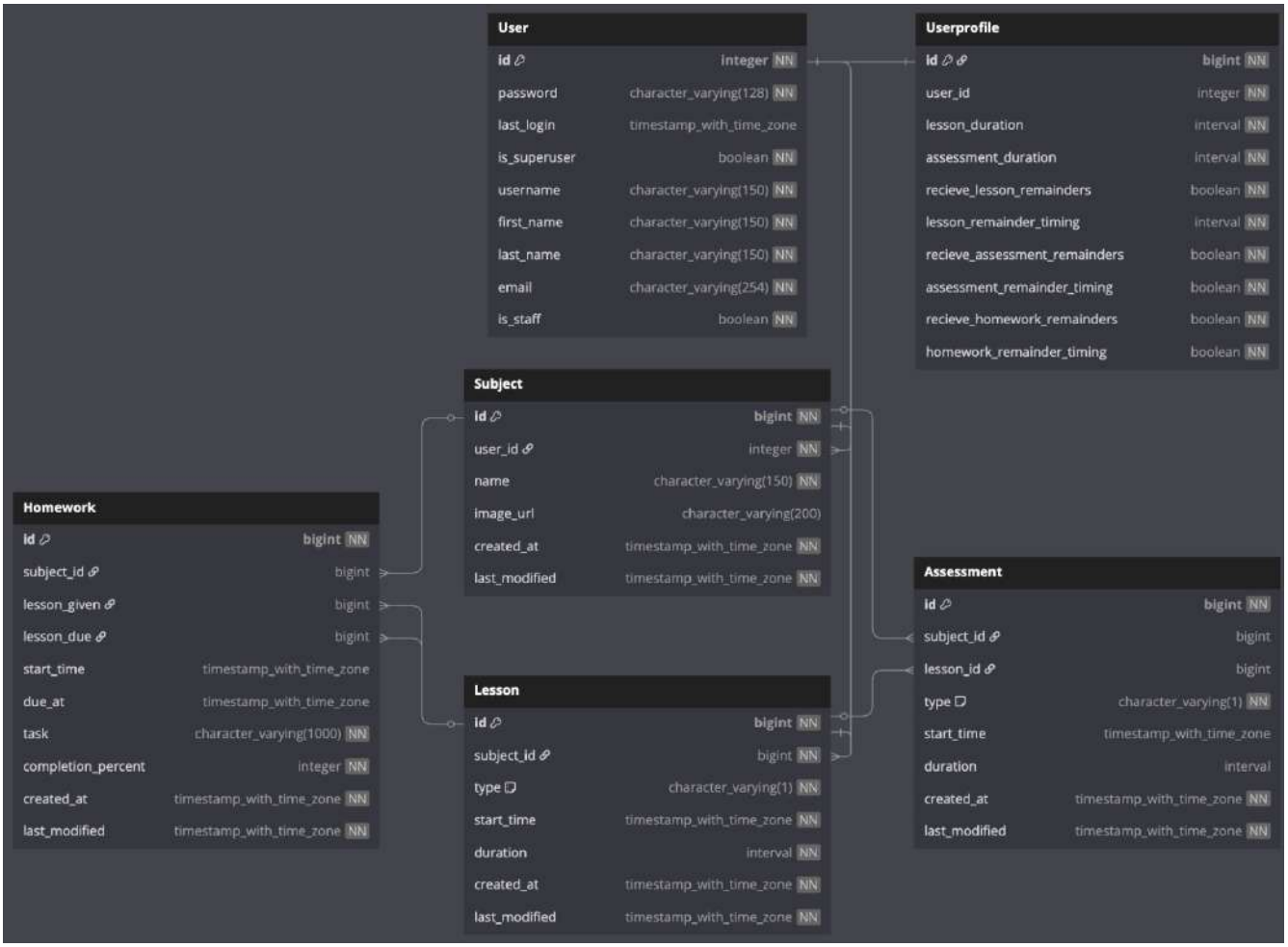

Рис. 2 ER-модель застосунку «Органайзер студента»

Сутність **Subject** відображає предмет – одиницю навчального процесу, що містить в собі певне коло знань та умінь.

Subject:

id: bigint PK NN

user_id: integer FK NN on_delete=CASCADE **(1) (2)**

name: character varying NN **(3)**

image_url: character varying(200) NN **(4)**

created_at: timestamp with time zone NN

last_modified: timestamp with time zone NN

Сутність **Lesson** відповідає уроку – окремому заняттю з певного предмету, що відбувається в конкретний момент часу та має певну тривалість.

Lesson:

id: bigint PK NN

subject_id: bigint FK NN on_delete=CASCADE

type: character varying NN **(5)**

start_time: timestamp with time zone NN **(6)**

duration: interval NN **(7)**

created_at: timestamp with time zone NN

last_modified: timestamp with time zone NN

Сутність **Assessment** уособлює форму контролю знань – процес перевірки знань здобувачів освіти з певного предмету, котрий може відбуватися як під час певного уроку, так і сам по собі.

Assessment:

id: bigint PK NN

subject_id: bigint FK N on_delete=CASCADE **(10) (11)**

lesson_id: bigint FK N on_delete= CASCADE **(9) (10) (11)**

type: character varying NN **(12)**

start_time: timestamp with time zone N **(8) (11)**

duration: interval N **(13) (4)**

description: character varying N

created_at: timestamp with time zone NN

last_modified: timestamp with time zone NN

Сутність **Homework** моделює домашнє завдання з певного предмету. Воно може бути призначене на уроці, здане під час іншого уроку або існувати цілком незалежно.

Homework:

id: bigint PK NN

subject_id: bigint FK N on_delete=CASCADE **(15)**

lesson_given: bigint FK N on_delete=SET_NULL **(15) (17) (20) (21) (27)**

lesson_due: bigint FK N on_delete=SET_NULL **(15) (16) (17) (24) (25) (26) (27)**

start_time: timestamp with time zone N **(18) (19) (26)**

due_at: timestamp with time zone N **(16) (22) (23) (26) (27)**

task: character varying NN **(28)**

completion_percent: integer NN **(29)**

has_subtasks: boolean NN

created_at: timestamp with time zone NN

last_modified: timestamp with time zone NN

Сутність **User** представляє користувача веб-сервісу «Органайзер студента» – здобувача освіти або адміністратора. Вона лежить в основі системи автентифікації

та авторизації, інкапсулюючи дані облікового запису та інформацію про права доступу.

User:

id: integer PK NN

password: character varying(128) NN

last_login: timestamp with time zone N

is_superuser: boolean NN

username: character varying(150) NN

first_name: character varying(150) NN

last_name: character varying(150) NN

email: character varying(254) NN

is_staff: boolean NN

Сутність **UserProfile** розширює модель користувача, зберігаючи додаткові налаштування профілю, що стосуються тривалості занять та переваг у нагадуваннях.

UserProfile:

id: bigint PK NN

user_id: integer FK NN on_delete=CASCADE

lesson_duration: interval NN

assessment_duration: interval NN

receive_lesson_remainders: boolean NN

lesson_remainder_timing: interval NN

receive_assessment_remainders: boolean NN

assessment_remainder_timing: interval NN

receive_homework_remainders: boolean NN

homework_remainder_timing: interval NN

Для забезпечення коректності та несуперечності даних було введено наступні **обмеженні цілісності** для сутностей**:**

Subject:

1. Кожен екземпляр сутності **User** може бути пов’язаний не більше ніж з 200-ма екземплярами сутності **Subject.**

2. Після створення екземпляру сутності **Subject** не можна модифікувати значення поля **user_id**.

3. Поле **name** повинно містити не більше 150-ти символів.

4. Поле **image_url** повинно містити валідний URL.

Lesson:

5. Поле type повинно містити 1 символ.

6. Поле **start_time** екземпляру сутності **Lesson** повинно містити майбутню дату та час.

7. Значення поля **duration** екземпляру сутності **Lesson** має знаходитися в межах від 15-ти хвилин до 8-ми годин.

Assessment:

8. Поле **start_time** екземпляру сутності **Assessment** повинно містити майбутню дату та час.

9. Поле **start_time** екземпляру сутності **Lesson**, пов’язаного з екземпляром сутності **Assessment**, повинно містити майбутню дату та час.

10. В кожному екземплярі сутності **Assessment** принаймні одне з полів **subject_id** або **lesson_id** повинно мати значення.

11. У випадку, коли в екземплярі сутності **Assessment** поле **lesson_id** не містить значення, поля **subject_id** та **start_time** обов'язково повинні мати значення.

12. Поле type повинно містити 1 символ.

13. Значення поля **duration** екземпляру сутності **Assessment** має знаходитися в межах від 5-ти хвилин до 7-ми днів.

14. Значення поля **duration** екземпляру сутності **Assessment** може перевершувати значення поля **duration** екземпляру сутності **Lesson**, з яким він пов'язаний. Таким чином забезпечується підтримка специфічних сценаріїв, наприклад ситуацій, коли студент отримує завдання для екзамену на конкретному уроці, проте має більший проміжок часу на його написання. Студент отримує повідомлення про подібну невідповідність і має підтвердити, що це саме той ефект, якого він прагнув досягти.

Homework

15. В кожному екземплярі сутності **Homework** принаймні одне з полів **subject_id** або **lesson_given** або **lesson_due** повинно мати значення.

16. В кожному екземплярі сутності **Homework** принаймні одне з полів **due_at** або **lesson_due** повинно мати значення.

17. Якщо в екземплярі сутності **Homework** обидва поля, **lesson_given** і **lesson_due**, мають значення, екземпляри сутності **Lesson**, на які вони посилаються, повинні належати одному й тому ж екземпляру сутності **Subject**.

18. Значення поля **start_time** екземпляру сутності **Homework** не може бути більше ніж на 1 рік у минулому.

19. Значення поля **start_time** екземпляру сутності **Homework** не може бути більше ніж на 1 рік в майбутньому.

20. Значення поля **start_time** екземпляру сутності **Lesson**, на який посилається поле **lesson_given**, не може бути більше ніж на 1 рік у минулому.

21. Значення поля **start_time** екземпляру сутності **Lesson**, на який посилається поле **lesson_given**, не може бути в майбутньому.

22. Значення поля **due_at** екземпляру сутності **Homework** не може бути більше ніж на 1 рік у майбутньому.

23. Значення поля **due_at** екземпляру сутності **Homework** не може бути більше ніж на 1 місяць в минулому.

24. Значення поля **start_time** екземпляру сутності **Lesson**, на який посилається поле **lesson_due**, не може бути більше ніж на 1 місяць в минулому.

25. Значення поля **start_time** екземпляру сутності **Lesson**, на який посилається поле **lesson_due**, не може бути більше ніж на 1 рік у майбутньому.

26. Значення поля **start_time** екземпляру сутності **Homework** повинно бути меншим за значенням полів **due_at** сутності **Homework** та **start_time** сутності **Lesson**, на який посилається поле **lesson_due**.

27. Значення поля **start_time** екземпляру сутності **Lesson**, на який посилається поле **lesson_given**, повинно бути меншим за значенням полів **due_at** сутності **Homework** та **start_time** сутності **Lesson**, на який посилається поле **lesson_due**.

28. Поле **task** сутності **Homework** повинно містити не більше 1000 символів.

29. Поле **completion_percent** сутності **Homework** повинно містити значення в рамках від 0 до 100 включно.

РОЗДІЛ 2.3 ФУНКЦІОНАЛ ЗАСТОСУНКУ

2.3.1 Типи користувачів застосунку

Застосунок «Органайзер студента» передбачає існування двох типів зареєстрованих користувачів, що різняться ціллю свого користування сервісом та обсягом доступних їм повноважень.

Цільова аудиторія застосунку – ***здобувачі освіти***, вони ж звичайні користувачі. Це студенти або школярі, що використовують його для організації своєї навчальної діяльності. Користувач цього типу має повний контроль над інформацією про себе, а також над сутностями, що інкапсулюють ключові елементи його навчального процесу і налаштуваннями свого профілю. З іншого боку, взаємодія звичайного користувача з «Органайзером студента» обмежується його власними даними; він не має доступу до інформації про інших користувачів. Щоб стати звичайним користувачем, достатньо зареєструватися через графічний інтерфейс застосунку, вказавши унікальне ім’я користувача та пароль, що задовольняє вимоги безпеки.

Для забезпечення стабільної роботи застосунку існують ***адміністратори*** – спеціальні користувачі, які здійснюють керування контентом та його модерацію. Адміністратор має змогу переглядати, створювати, редагувати та видаляти будь-які дані, окрім паролів користувачів, які він може скинути, але не переглянути. Ці повноваження необхідні для контролю вмісту застосунку, збору статистичної інформації, відстеження підозрілої активності та запобігання атакам. Графічний інтерфейс і функціонал бекенду не надають можливостей створення адміністратора з метою захисту від несанкціонованого доступу. Лише перевірені особи, за потреби юридично зобов’язані не розголошувати конфіденційних даних, можуть отримувати такий рівень доступу. Створити нового адміністратора можна лише

через адміністративну консоль Django за допомогою команди **python manage.py createsuperuser**. Аби ввійти в систему як адміністратор достатньо ввести відповідні облікові дані в форму для авторизації.

Незареєстровані та неавторизовані користувачі не мають доступу до жодного функціоналу «Органайзеру» окрім форм реєстрації та авторизації. Це спричинено тим, що вся інформація в застосунку персоналізована, а отже неідентифікованому користувачу неможливо надати змістовного доступу.

2.3.2 Функціональні вимоги застосунку

1) Загальні вимоги

- Існує два типи користувачів: здобувачі освіти та адміністратори.
- Здобувачі освіти та адміністратори – окремі несумісні сутності, кожен з них взаємодіє зі своєю частиною застосунку.
- Впроваджена автентифікація та авторизація користувачів.

2) Вимоги за типами користувачів

НЕЗАРЕЄСТРОВАНИЙ КОРИСТУВАЧ

- Може зареєструватися в застосунку як здобувач освіти.

ЗДОБУВАЧ ОСВІТИ

- Може переглядати створені ним предмети, уроки, форми контролю знань та домашні завдання.
- Може фільтрувати створені ним предмети, уроки, форми контролю знань та домашні завдання за ретельно визначеними параметрами.
- Може створювати нові предмети, уроки, форми контролю знань та домашні завдання.
- Може редагувати створені ним предмети, уроки, форми контролю знань та домашні завдання.

- Може видаляти створені ним предмети, уроки, форми контролю знань та домашні завдання.
- Може переглядати інформацію про себе.
- Може редагувати інформацію про себе.
- Може переглядати налаштування свого профілю.
- Може редагувати налаштування свого профілю.
- Може переглядати візуально структуровану інформацію про власні уроки, форми контролю знань та дедлайни домашніх завдань на поточний місяць за допомогою календаря.
- Може видалити свій профіль і всю інформацію про себе.

АДМІНІСТРАТОР

- Вимоги ***ЗДОБУВАЧА ОСВІТИ***, також:
- Може переглядати всі існуючі в застосунку предмети, уроки, форми контролю знань та домашні завдання.
- Може створювати нові предмети, уроки, форми контролю знань та домашні завдання, в тому числі і для інших користувачів.
- Може редагувати будь-які предмети, уроки, форми контролю знань та домашні завдання.
- Може видаляти будь-які предмети, уроки, форми контролю знань та домашні завдання.
- Може переглядати інформацію про всіх користувачів (окрім паролю).
- Може створювати нових користувачів типу «здобувач освіти».
- Може редагувати інформацію про будь-якого користувача.
- Може видаляти користувачів.

ВИСНОВОК ДО РОЗДІЛУ 2

Другий розділ курсової роботи присвячено етапу проектування веб-сервісу «Органайзер студента». В ньому було сформовано первинне бачення застосунку та обґрунтовано доцільність використання реляційної бази даних і взаємодії з нею через ORM при його реалізації. Було прийнято рішення, що сервіс підтримуватиме два типи користувачів: здобувачів освіти, які використовуватимуть його для організації навчального процесу, та адміністраторів, чиїм обов'язком буде модерація контенту. Для кожної ролі визначено рівень доступу та набір доступних дій, на основі чого було сформовано функціональні вимоги застосунку.

Також було визначено предметну область застосунку, на її основі – спроектовано структуру даних та встановлено необхідні для адекватної роботи обмеження цілісності.

Враховуючи потребу «Органайзеру студента» в ORM, розподілі ролей користувачів та захисті персональних даних, було прийнято рішення імплементувати сервіс за допомогою Django, оскільки попередній аналіз показав, що цей задовольняє поставлені вимоги, надаючи необхідні компоненти «з коробки».

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОЕКТУ

РОЗДІЛ 3.1. АРХІТЕКТУРА ЗАСТОСУНКУ

Архітектура сервісу «Органайзер студента» базується на патерні MVT[29] та спроектована навколо окремих Django Application (app)[28], кожен з яких відповідає за імплементацію замкненої на собі частини функціоналу. Застосунок складається з окремих app, що реалізують логіку, пов'язану з відповідними сутностями: **subject** відповідає за роботу з предметами, **lesson** – з уроками, тощо (Рис.3). В **dashboard** міститься код, пов'язаний з функціоналом календаря. Всі app визначають власні маршрути, моделі, представлення та шаблони.

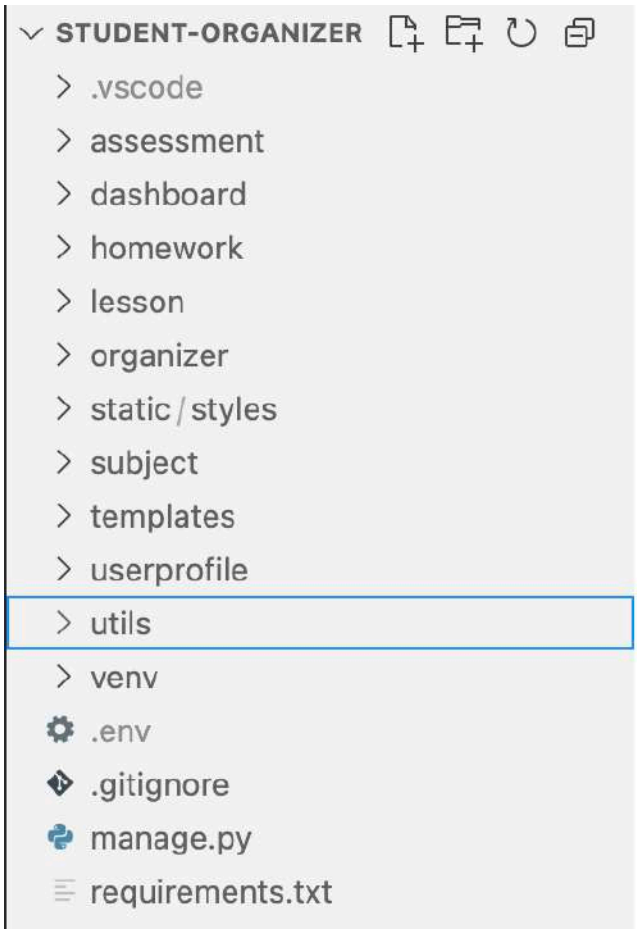

Рис. 3 Загальна структура застосунку «Органайзер студента»

Окрім основних app, в «Органайзері студента» присутні також папки з допоміжними компонентами. Зокрема, **templates** містить загальні для всього застосунку HTML-шаблони, **utils** – константи, кастомні функції та міксіни, а **static/styles** – статичні CSS-стилі, створені на основі іншого застосунку автора курсової[59]. Файл **.env** зберігає конфіденційні налаштування бази даних, **requirments.txt** містить список залежностей проекту, а **manage.py** використовується для запуску серверу, міграцій та виконання інших команд Django.

На прикладі app (Рис.4), відповідального за функціонал, пов'язаний з уроками, продемонстровано типову структуру всіх Django Application застосунку.

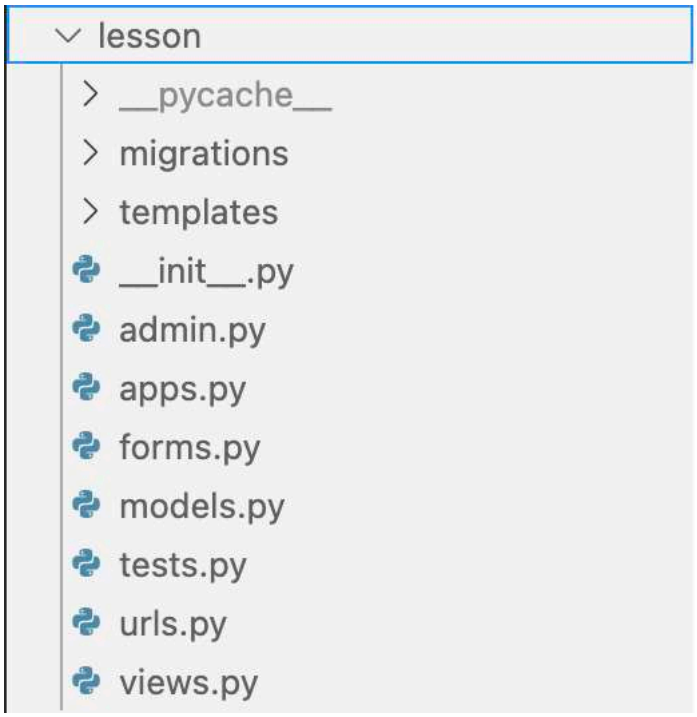

Рис. 4 Структура app відповідального за імплементацію сутності «Lesson»

Наявність файлу **__init__.py** свідчить про те, що перед нами – окремий модуль. Директорії **migrations** та **templates** містять відповідно міграції та шаблони, пов'язані з уроками. У **models.py** описано структуру моделі **Lesson** та правила її валідації, а у **forms.py** – форми для введення та редагування даних. У **urls.py** задається, які визначені в **views.py** представлення мають обробляти запити для

кожного маршруту. Файл **admin.py** дозволяє налаштувати вигляд моделі в адміністративному інтерфейсі, і, нарешті, службовий **apps.py** використовується для реєстрації всього застосунку в системі Django.

Використання Django Application дозволило створити модульний, масштабований застосунок, файлова структура та логічна архітектура якого тісно пов’язані.

РОЗДІЛ 3.2. РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ

Серверна частина застосунку «Органайзер студента» включає в себе реалізацію рівнів моделей, представлень та шаблонів відповідно до архітектурного патерну MVT[29].

База даних спеціально сконструйована таким чином, щоб її структура допускала часткове дублювання полів в різних сутностях для підтримки реальних сценаріїв використання. Наприклад, для відображення того факту, що форми контролю знань можуть як існувати незалежно, так і бути пов’язаними з певними уроками. Шляхом перевизначення методів **clean()** та **save()**, відповідальних за валідацію та збереження екземплярів сутності відповідно, на рівні моделей реалізовано логіку автоматичного заповнення або очищення полів, що дозволяє уникати суперечностей і фактичного дублювання даних.

Правила керування даними

Assessment

Екземпляр сутності **Assessment** може бути прив’язаним до певного екземпляру сутності **Lesson**, для відображення випадків, коли певна форма контролю відбувається під час запланованого уроку. В такому випадку, щоб запобігти дублюванню даних, ті поля екземплярів **Assessment,** значення яких можна видобути з екземпляру **Lesson** (а саме – **subject_id** та **start_time**), стають None. Таким чином вдається запобігти дублюванню даних, проте не унеможливити створення самостійних екземплярів

Значення поля **duration** визначається за таким алгоритмом:

- Якщо користувач ввів значення вручну, воно зберігається без змін.

- Якщо існує зв’язок з екземпляром сутності **Lesson** і немає заданого вручну значення, поле **duration** залишається порожнім, а інформація про тривалість контролю отримується з відповідного екземпляру **Lesson** (поле **duration**).
- Якщо екземпляр не має пов’язаного екземпляру сутності **Lesson** і користувач не вказав значення, поле **duration** за замовчуванням дорівнює визначеній для користувача стандартній тривалості уроку.

Homework

Екземпляр сутності **Homework** може бути прив’язаним до екземплярів сутності **Lesson**, для відображення випадків, коли домашня робота тісно пов’язана з матеріалом конкретного уроку (**lesson_given**), або її здача відбувається під час конкретного уроку (**lesson_due**). В такому випадку, щоб запобігти дублюванню даних, ті поля екземплярів **Homework,** значення яких можна видобути з екземплярів **Lesson** (а саме – **subject_id, start_time** та **due_at**), стають None. Таким чином вдається запобігти дублюванню даних, проте не унеможливити створення самостійних екземплярів.

Значення поля **start_time** визначається за таким алгоритмом:

- Якщо користувач ввів значення вручну, воно зберігається без змін.
- Якщо існує зв’язок з екземпляром сутності **Lesson** через **lesson_given** і немає заданого вручну значення, поле **start_time** залишається порожнім, а інформація про початок терміну приймання домашньої роботи отримується з відповідного екземпляру **Lesson**. (поле **start_time**).
- Якщо екземпляр не має пов’язаного з сутністю **Lesson** через **lesson_given** і користувач не вказав значення, поле **start_time** за замовчуванням встановлюється на поточний час.

Значення поля **due_at** визначається за таким алгоритмом:

- Якщо користувач ввів значення вручну, воно зберігається без змін.
- Якщо існує зв’язок з екземпляром сутності **Lesson** через **lesson_due** і немає заданого вручну значення, поле **due_at** залишається порожнім, а інформація про завершення терміну приймання домашньої роботи отримується з відповідного екземпляру **Lesson**. (поле **start_time**).

Логіка на рівні представлень реалізована за допомогою class-based generic views[37][38], поведінка яких за потреби доповнюється або адаптується за допомогою міксінів. Зокрема, саме таким чином були імплементовані обмеження за рівнем доступу. Використані в застосунку форми автоматично генеруються на основі моделей та успадковують правила їхньої валідації.

Особливості реалізації інтерфейсу користувача та використання Django Template Language розглядаються в наступному розділі, оскільки ці теми стосуються взаємодії користувача з застосунком.

РОЗДІЛ 3.3. ІНТЕРФЕЙС КОРИСТУВАЧА

3.3.1. Використання Django Template Language

Важливим етапом у розробці веб-сервісу «Органайзер студента» стало створення зручного, сучасного та добре підтримуваного інтерфейсу користувача. Проаналізувавши переваги та недоліки різних способів організації взаємодії між бекендом та фронтендом, а також враховуючи особливості Django, було прийнято рішення реалізувати інтерфейс користувача за допомогою Django Template Language (DTL) – вбудованої системи шаблонів, що дозволяє визначити як структуру HTML-сторінок, так і гнучку архітектуру самих шаблонів.

Застосування DTL дало змогу реалізувати застосунок в межах одного фреймворку, що є цінним саме для курсової роботи, оскільки дає змогу продемонструвати можливості Django як інструменту для full-stack web розробки сервісу. До того ж, обраний підхід відповідає філософії Django (opinionated, batteries included) і сприяє максимально ефективному використанню його вбудованих можливостей. Головним недоліком обраного рішення є підвищене навантаження на сервер – вся обробка шаблонів відбувається на стороні бекенду.

Шаблони майбутніх HTML-сторінок застосунку написані відповідно до best practices Django, які передбачають створення шаблонної системи з урахуванням принципів наслідування та композиції. Усі базові компоненти (наприклад навігаційна панель чи пагінація) реалізовано як окремі шаблони та вбудовано в сторінки вищого рівня через тег **{% include %}**[40]. Такий підхід дозволяє використовувати повторювані частини інтерфейсу як складові частини інших шаблонів, подібно до того, як колеса є частиною автомобіля.

Основна HTML-структура задана в базовому шаблоні, на основі якого створюються шаблони списків, форм, перегляду деталей, тощо. Наприклад,

загальний шаблон форми містить вираз **{{ form.as_p }}**, що дозволяє відобразити всі поля форми як елементи **<p>**. Шаблон base_form.html не використовується для рендерингу сторінок безпосередньо, проте всі його нащадки мають доступ до власного об’єкта form, сформованого Django для конкретної моделі, відображення якого вже прописано в батьківському шаблоні.

Усі універсальні елементи розташовані якомога вище в ієрархії, а найбільш конкретні шаблони сторінок містять лише унікальну інформацію. Такий підхід дозволяє запобігти дублюванню коду і підвищити підтримуваність застосунку, оскільки зміни до структури вносяться один раз у відповідному шаблоні.

3.3.2. Інтерфейс користувача

Зручний інтерфейс користувача є ключовим чинником, що перетворює застосунок з просто функціонального на зручний в повсякденному використанні. У веб-сервісі «Органайзер студента» інтерфейс розроблявся з ціллю забезпечити максимальний комфорт майбутніх користувачів. Саме тому проектування сторінок та розміщення на них елементів здійснювалося з урахуванням інтересів користувача, його ймовірних мотивів та очікуваних наступних дій. Приємний зовнішній вигляд застосунку досягається завдяки сучасному мінімалістичному дизайну та використанню гармонійної кольорової гами. Контрастне співвідношення тексту й фону високе, що забезпечує високу читабельність. Елементи дизайну доповнюють функціонал сайту, інтуїтивно спрощують взаємодію та не відволікають увагу на себе. При цьому назви сутностей, кнопок, гіперпосилань та повідомлення про помилки спроектовані так, щоб бути максимально недвозначними і зрозумілими навіть для користувачів без технічного бекграунду.

Взаємодія користувача з застосунком розпочинається з форми авторизації, яка надає можливість як увійти в існуючий обліковий запис (кнопка «Login»), так і створити новий (кнопка «Sign Up»).

Рис. 5 Вигляд форми авторизації користувача на початковій сторінці

Наведений нижче фрагмент коду наслідує базовий шаблон форми, реалізуючи логування користувача:

```
% extends "base/base_form.html" %}

{% block title %}Login{% endblock %}
{% block form_title %}Login{% endblock %}

{% block navbar %}{% endblock %}

{% block form_action_buttons %}
  <input type="hidden" name="next" value="{{ next }}">
  <div class="buttons">
    <input class="button-link" type="submit" value="Login">
    <a href="{% url 'user_register' %}" class="button-link">Sign Up</a>
  </div>
```

{% endblock %}

Форма для створення облікового запису підказує, як правильно заповнювати її поля, та містить поради стосовно створення безпечного паролю.

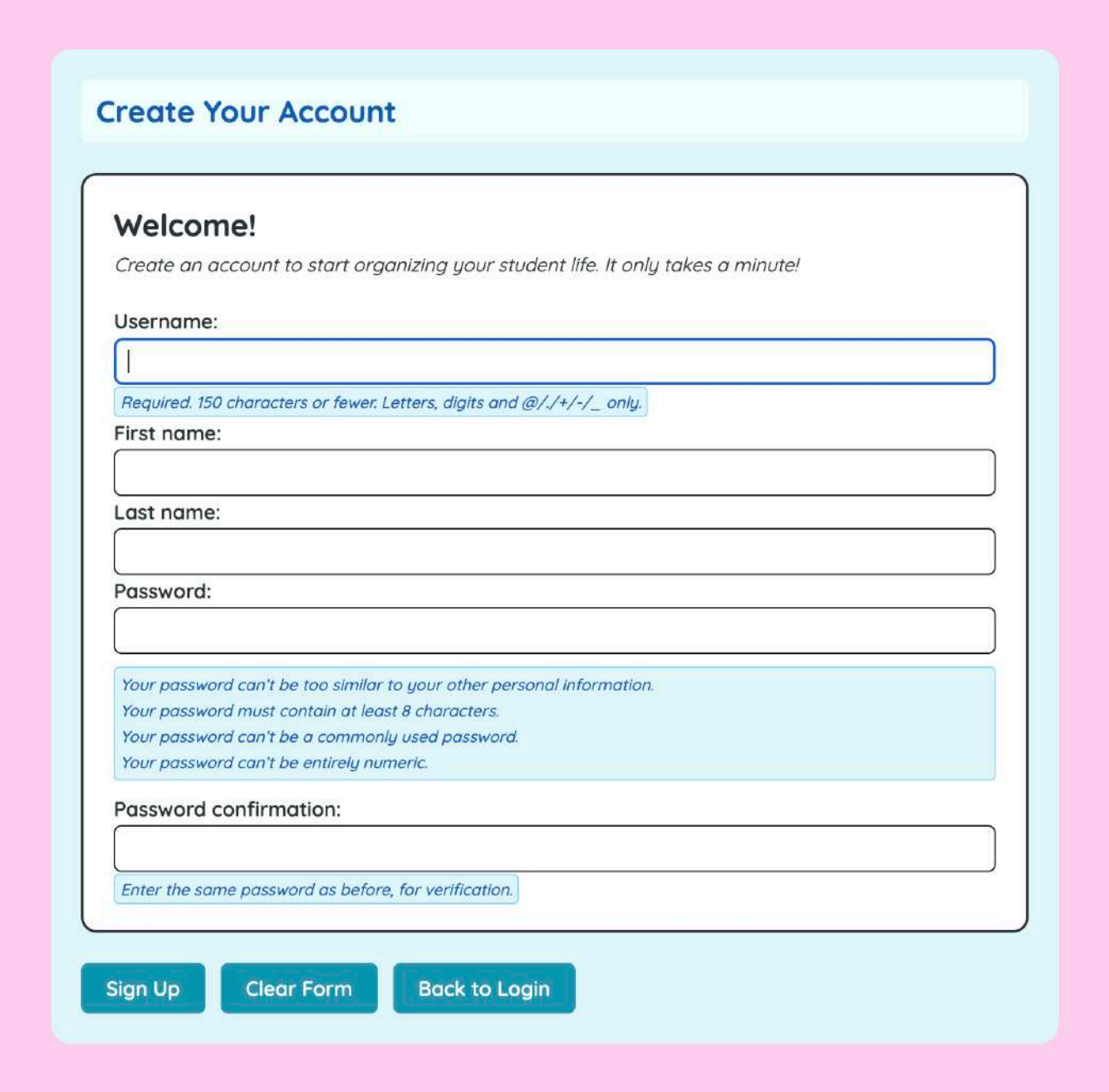

Рис. 6 Вигляд сторінки реєстрації нового користувача

Під час створення нового облікового запису користувачу автоматично присвоюються стандартні значення налаштувань тривалості активностей та параметри нагадувань про них. Надалі він може переглядати цю інформацію, а також редагувати як особисті дані, так і налаштування, скориставшись сторінкою свого профілю.

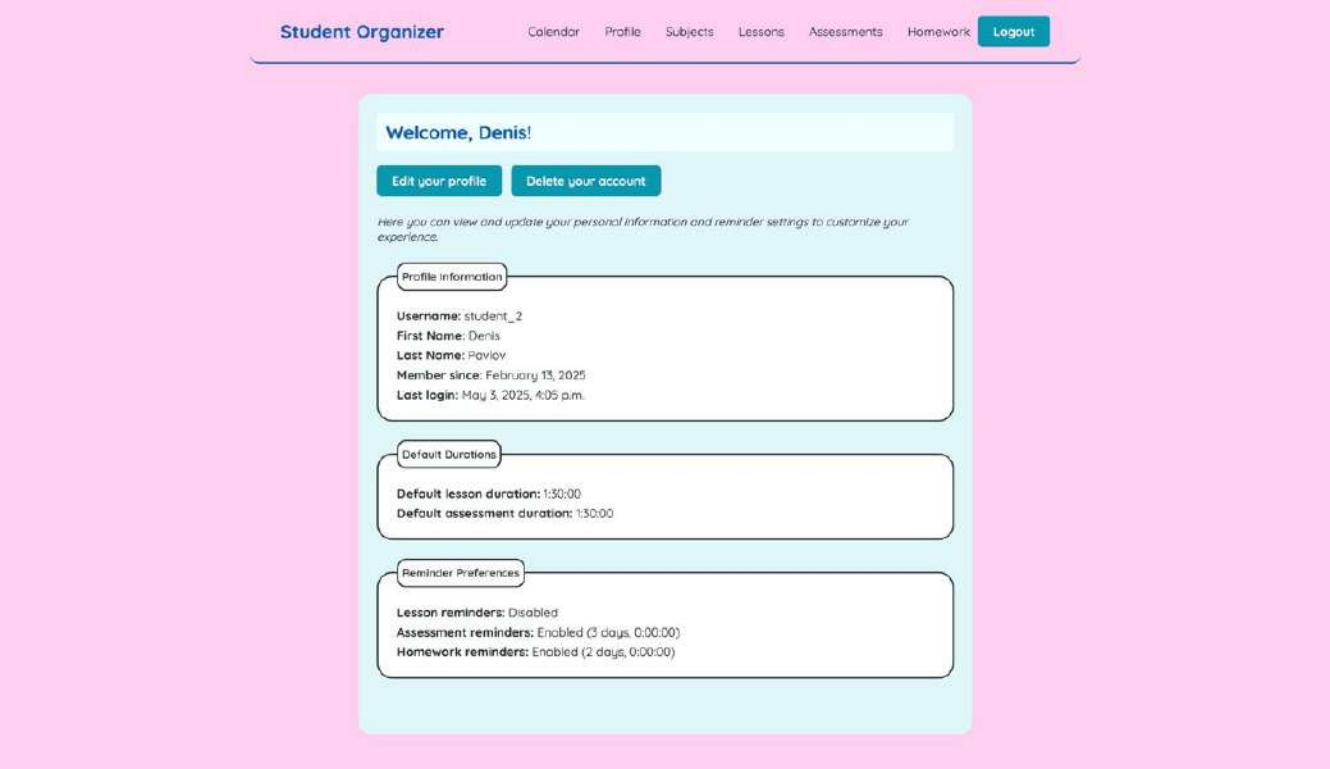

Рис. 7 Вигляд сторінки профілю користувача

Після авторизації користувач перенаправляється на головну сторінку, де за допомогою календаря відображаються події, заплановані на поточний місяць. Таким чином полегшується організація навчального процесу: користувач може швидко розібратися в своєму розкладі, оновлювати в пам’яті інформацію про минулі та майбутні події та ефективно спланувати свій час. Різні типи подій в календарі (уроки, форми контролю знань, дедлайни домашніх завдань) виділені різними кольорами для спрощення сприйняття великого обсягу інформації.

Натиснувши на певну подію, користувач переходить на сторінку з її детальним описом.

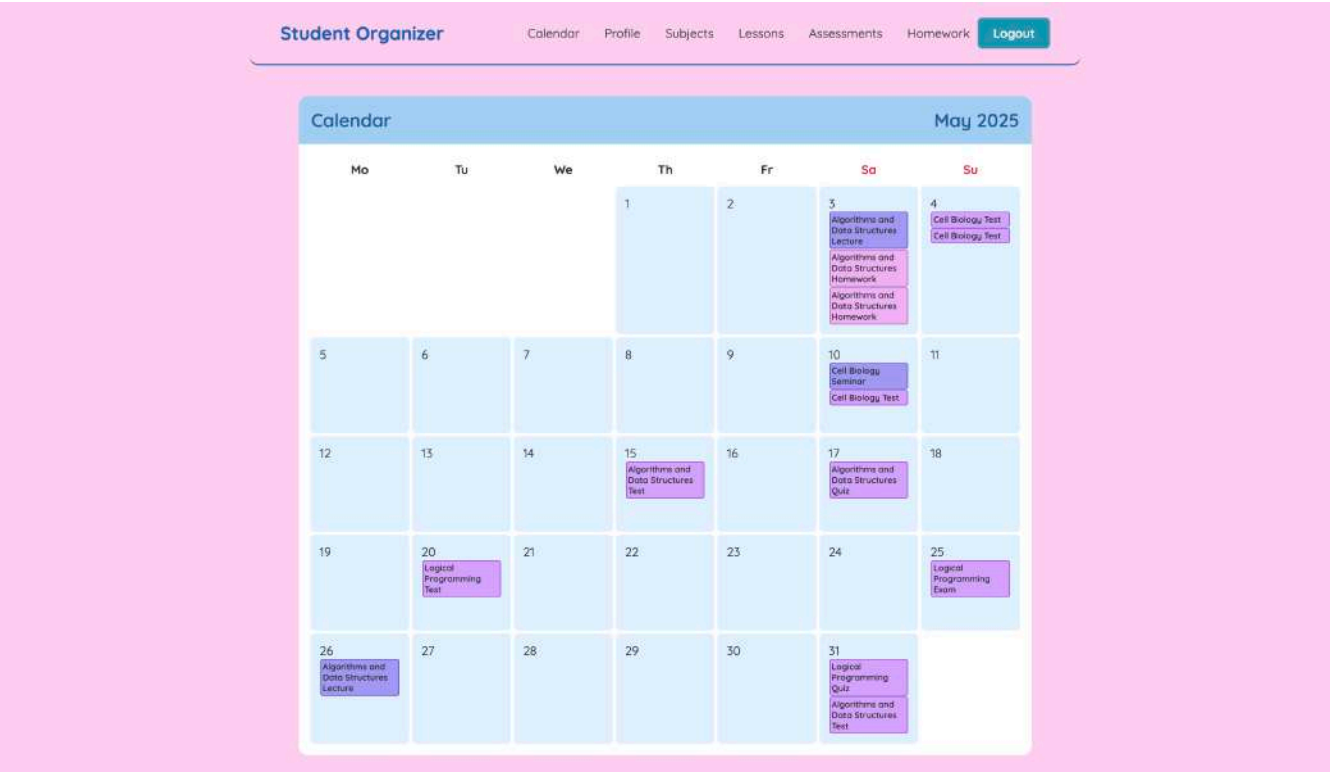

Рис. 8 Вигляд головної сторінки з календарем користувача

Наведений нижче фрагмент коду наслідує загальний шаблон сторінки, реалізуючи логіку рендеренгу календаря з усіма подіями на поточний місяць.

```
{% extends "base/base.html" %}
{% load static %}
{% load custom_tags %}

{% block title %}Calendar{% endblock title %}
{% block content_wrapper %}calendar{% endblock %}

{% block content %}
  <div class="calendar-header">
    <p>Calendar</p>
```

```
      <p>{{ month }} {{ year }}</p>
    </div>

    <div class="calendar-body">
      <div class="calendar-content">
        <div class="workweek">Mo</div>
        <div class="workweek">Tu</div>
        <div class="workweek">We</div>
        <div class="workweek">Th</div>
        <div class="workweek">Fr</div>
        <div class="weekend">Sa</div>
        <div class="weekend">Su</div>

        {% for week in calendar %}
          {% for day in week %}
            {% if day != 0 %}
              <div class="calendar-day">{{ day }}
                {% for type, pk, description in calendar_events|get_item:day %}
                  <div class="calendar-{{ type }}">
                    {% with url_name=type|add:"_detail" %}
                      <a href="{% url url_name pk %}">{{ description }}</a>
                    {% endwith %}
                  </div>
                {% endfor %}
              </div>
            {% else %}
              <div class="invisible-day">{{ day }}</div>
            {% endif %}
          {% endfor %}
        {% endfor %}
      </div>
    </div>
{% endblock content %}
```

Аби користувач мав змогу вільно переміщатися застосунком, було створено навігаційну панель, яка розташовується в верхній частині екрану та забезпечує вільне переміщення застосунком, даючи можливість доступитися до будь-якого контенту в кілька кліків.

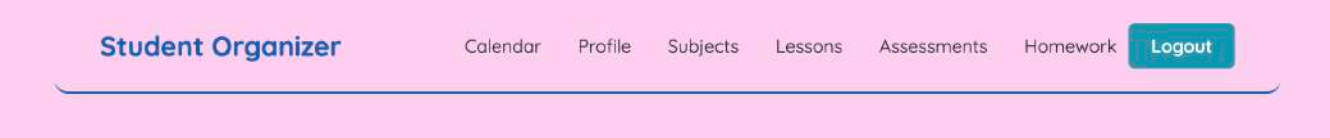

Рис. 9 Вигляд навігаційної панелі звичайного користувача

Користувач може переглядати узагальнену інформацію про всі екземпляри основних сутностей – предметів, уроків, форм контролю знань та домашніх завдань – у вигляді списку. Для ефективнішої організації інформації, полегшення процесу її аналізу та спрощення пошуку конкретних даних, було додано опцію фільтрації та сортування сутностей за ретельно підібраними параметрами, які відображені на лівій панелі вікна на Рис. 10.

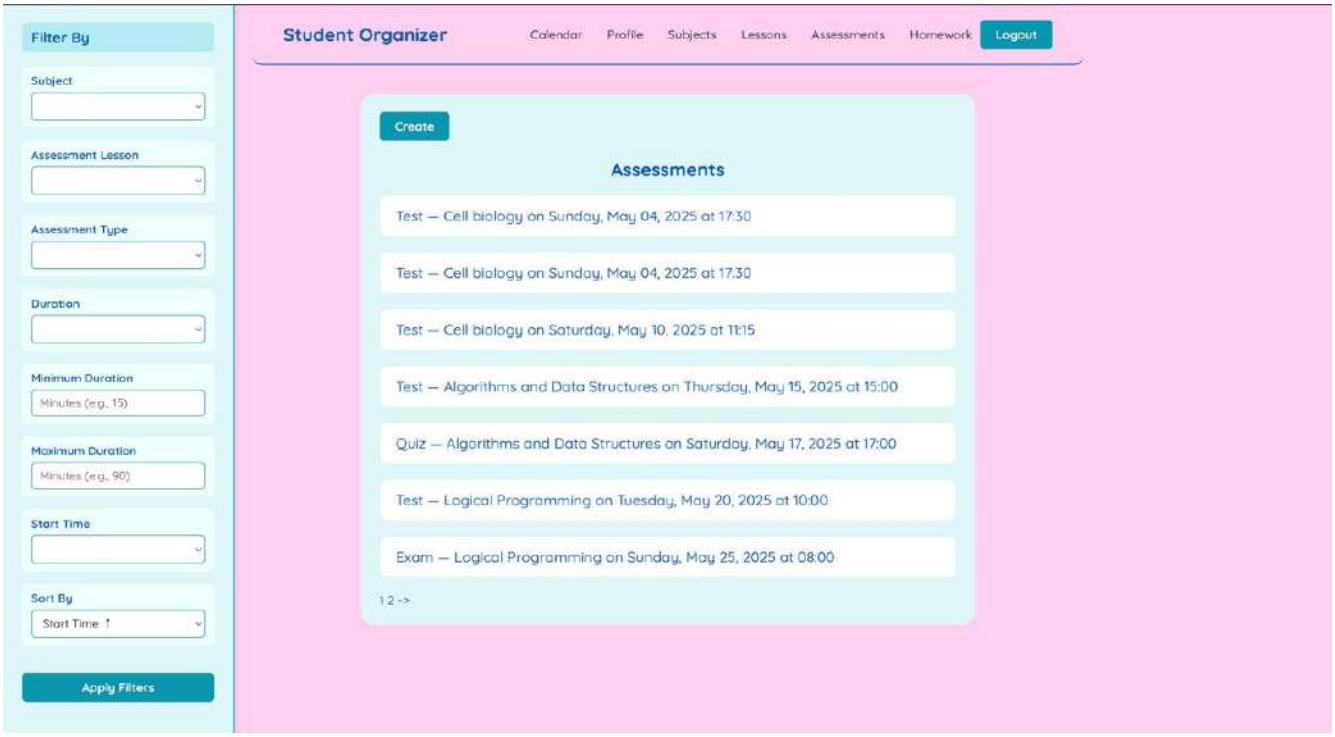

Рис. 10 Вигляд сторінки для перегляду узагальненої інформації про всі форми контролю

Натиснувши на конкретний елемент списку, користувач переходить на сторінку з детальною інформацією про обраний екземпляр сутності. Оскільки першим пріоритетом при розробці інтерфейсу була зручність кінцевого користувача, зроблено все можливе для його швидкого орієнтування в пов’язаній інформації. Наприклад, на сторінці з детальною інформацією про форму контролю знань передбачено прямі посилання на урок, під час якого вона проводитиметься.

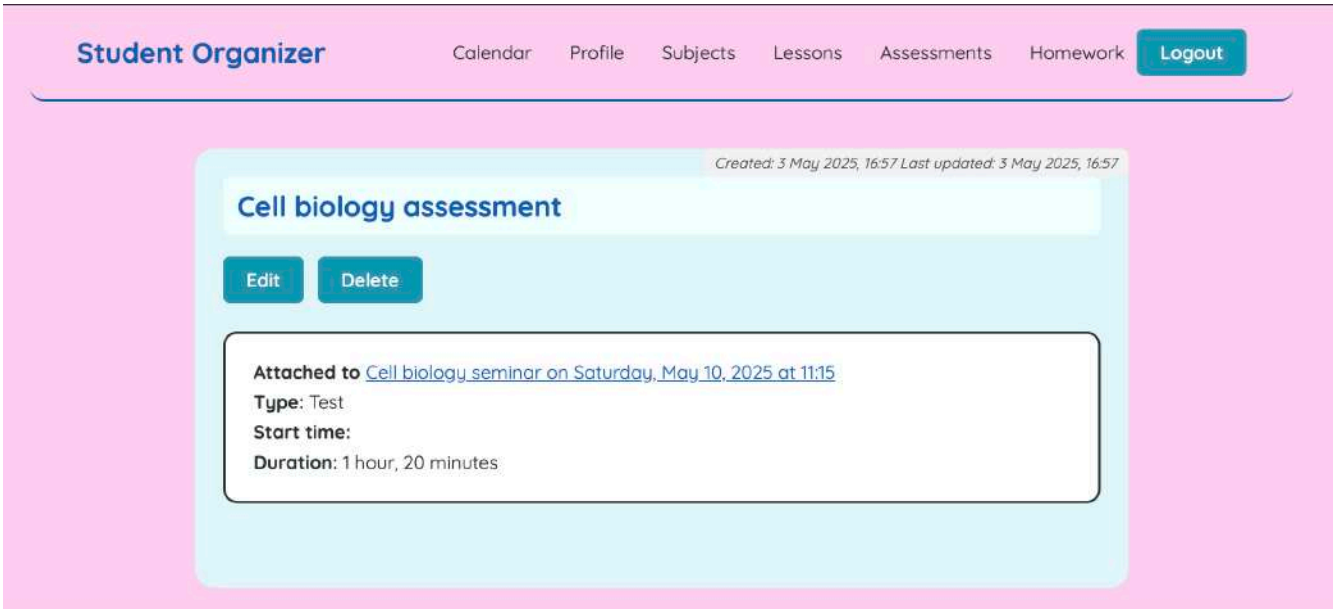

Рис. 11 Вигляд сторінки з деталями форми контролю знань

Важко переоцінити важливість цілісності даних для будь-якого застосунку, тому «Органайзер студента», окрім валідації на рівні моделей та представлень, здійснює низку перевірок для мінімізації некоректних дій та запобігання хибним крокам користувача. Приміром, видалення сутності відбувається тільки після того, як користувач підтвердить свої наміри. У випадках, коли така дія спричинить каскадне видалення пов’язаних сутностей, попередження про це виводиться на екран. Користувач має можливість переглянути ці сутності, що запобігає виникненню неочікуваних побічних ефектів.

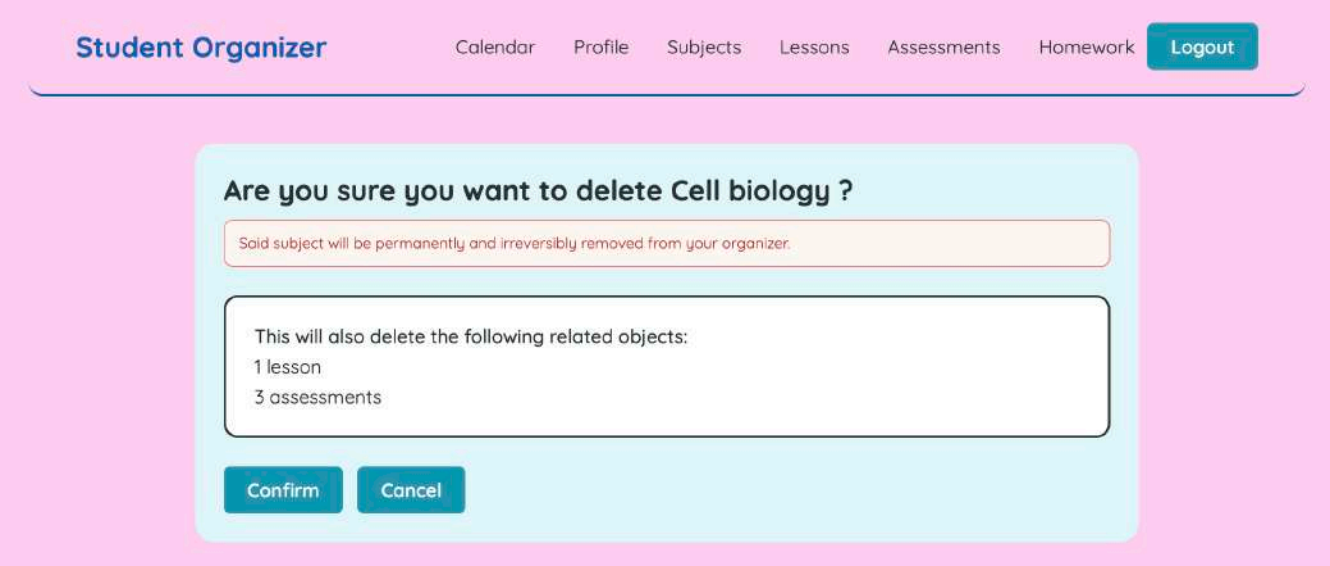

Рис. 12 Вигляд сторінки підтвердження видалення предмета

Для створення в Органайзері нового предмету, уроку, форми контролю знань або домашнього завдання, на сторінці перегляду всіх екземплярів відповідної сутності необхідно натиснути кнопку «Create», здійснивши таким чином перехід на сторінку з відповідною формою для створення.

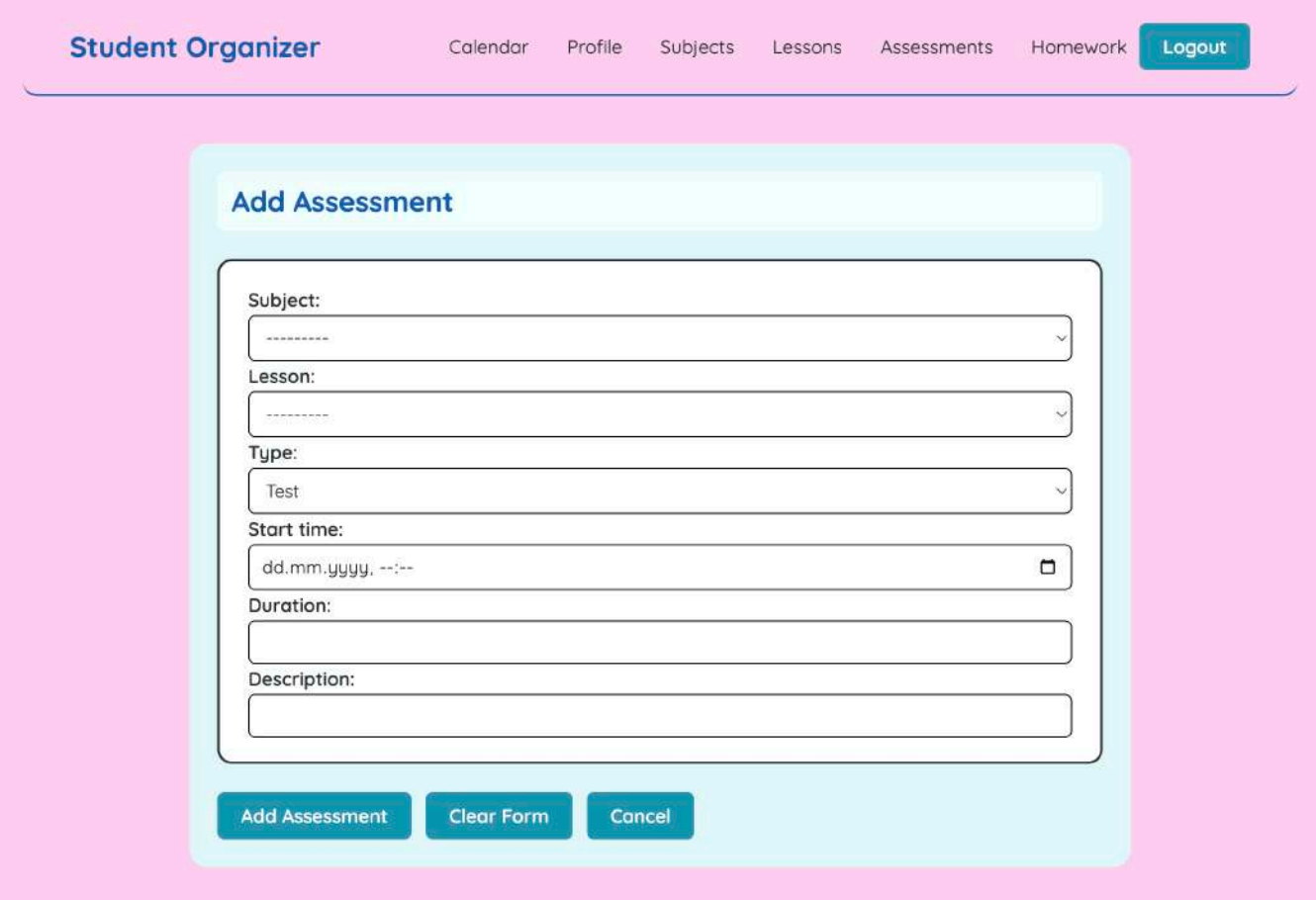

Рис. 13 Створення форми контролю

Коли, заповнюючи певну форму для створення або модифікації сутності, користувач вводить некоректні дані, біля кожного поля з такими даними виводиться повідомлення про помилку, адаптоване під конкретний ввід. В ньому пояснюється, як саме введені користувачем дані порушують правила застосунку та даються поради щодо виправлення ситуації. Такий підхід спрямований на уникнення непорозумінь між користувачем та застосунком і забезпечення швидкого вирішення можливих проблем.

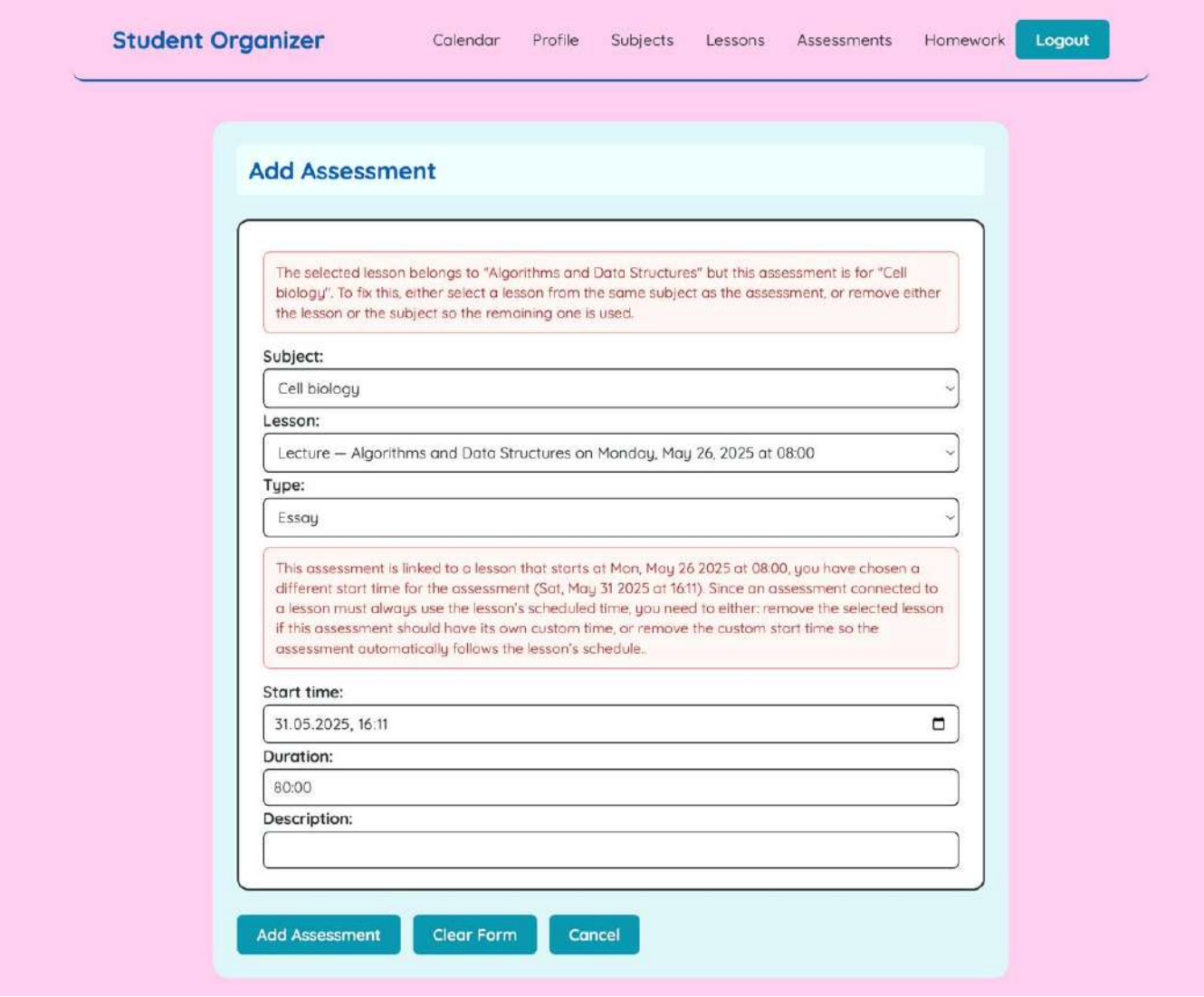

Рис. 14 Вікно форми контролю з повідомленнями про невалідні дані

Проте заповнення форми для створення нової сутності – це монотонний процес, в якому легко допустити помилку. Тому в «Органайзері студента» реалізовано автоматичне заповнення полів в випадках, коли це логічно обґрунтовано. Наприклад, якщо користувач перебуває на сторінці з деталями предмета та натискає кнопку «Add Assessment» для створення пов’язаної форми контролю, застосунок направляє його до форми з вже заповненим значенням поля subject, оскільки інформація про предмет доступна з контексту.

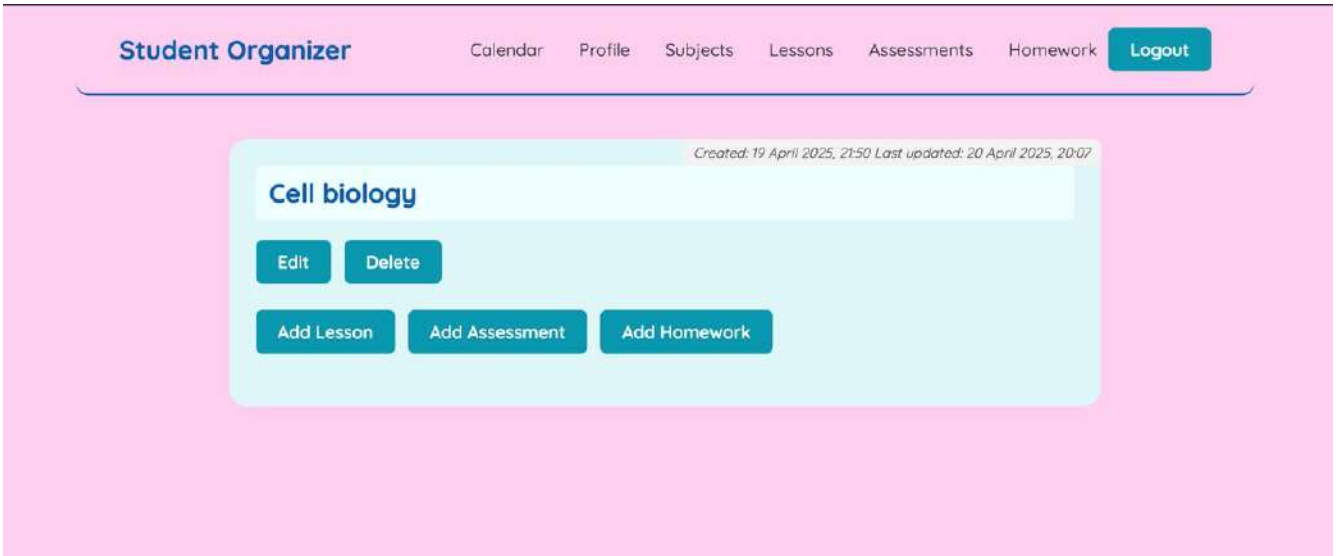

Рис. 15 Вигляд сторінки деталей предмету з кнопками для спрощеного створення пов'язаних сутностей

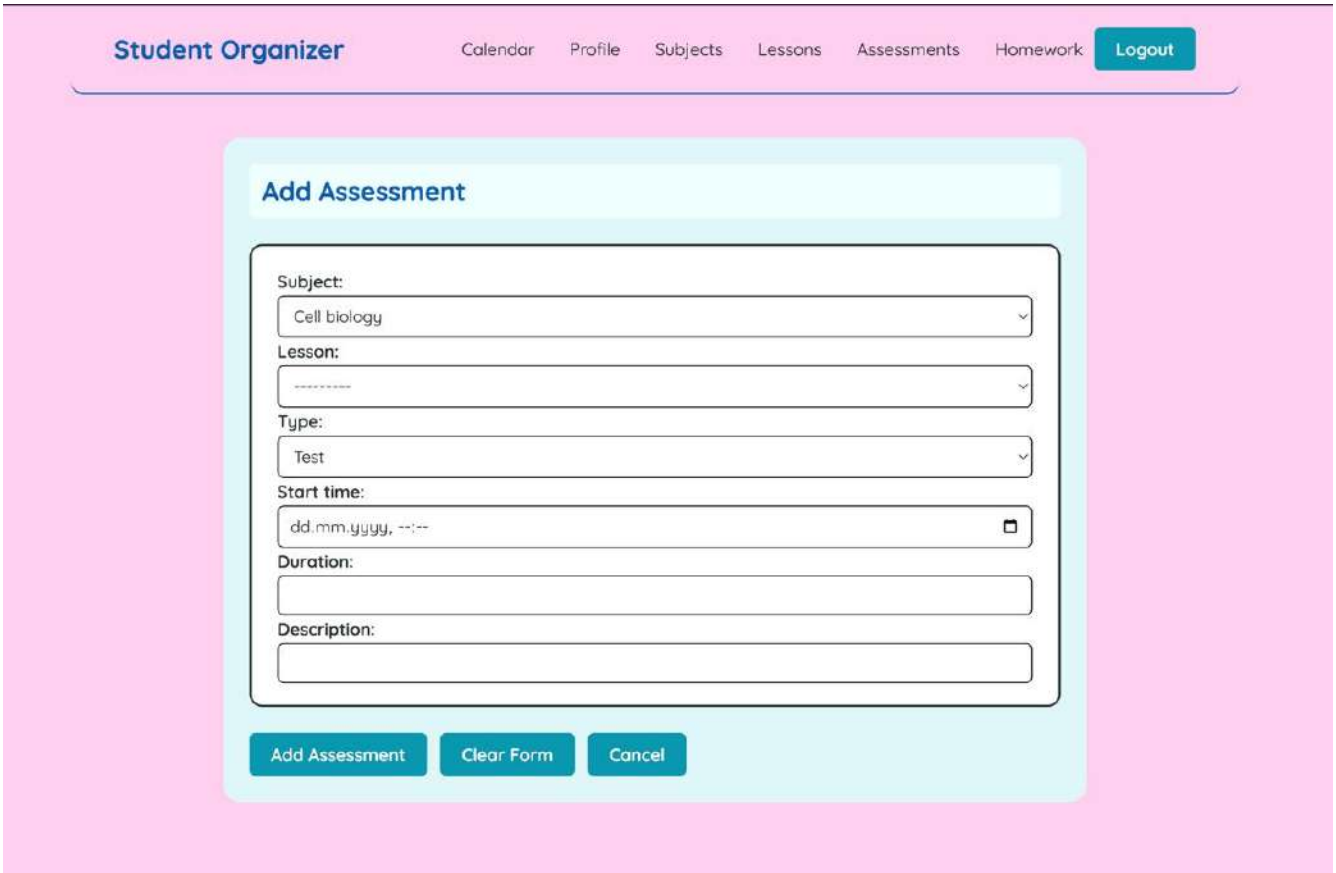

Рис. 16 Сторінка для створення форми контролю з попередньо заповненим значенням поля subject, перехід на яку здійснився після натискання кнопки Add Assessment

Принцип мінімізації помилок користувача розповсюджується на всі аспекти застосунку. Приміром, випадаючі списки містять лише валідні значення. Якщо редагуючи форму контролю знань, користувач хоче пов’язати її з певним уроком, він може обирати лише з уроків того предмету, якому належить форма і які знаходяться в допустимих часових рамках.

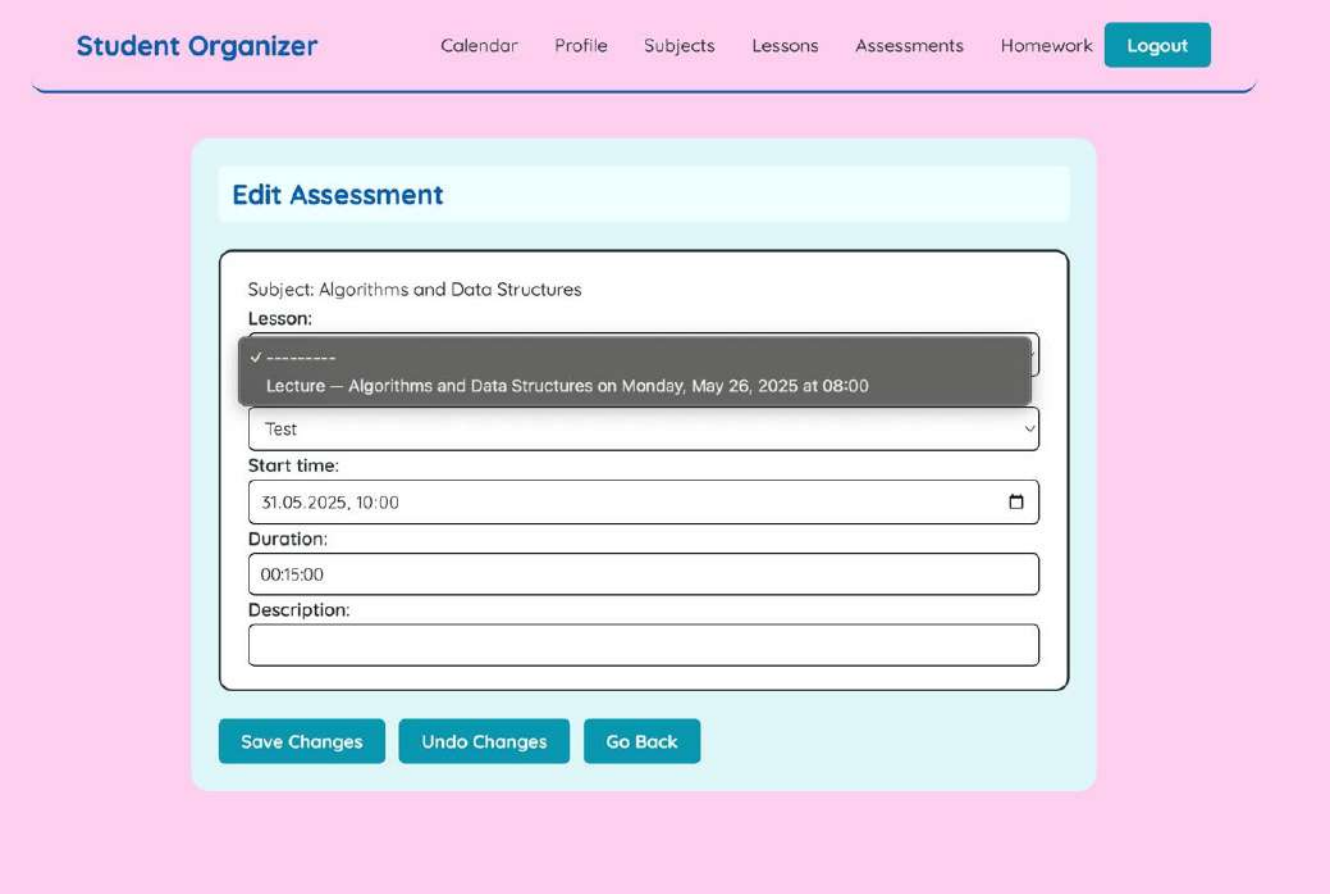

Рис. 17 Сторінка для редагування форми контролю знань

Адміністратори застосунку мають доступ до того ж графічного інтерфейсу, що і звичайні користувачі, а також – до спеціального адміністративного інтерфейсу, який надає можливості керувати вмістом застосунку. Якщо авторизований користувач розпізнаний застосунком як такий, що має повноваження адміністратора, навігаційна панель надає йому можливість перейти до адміністративного інтерфейсу, натиснувши кнопку «Admin Panel».

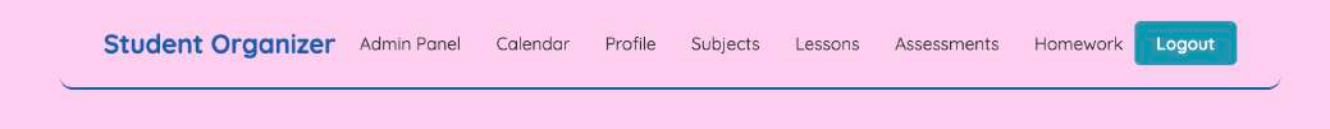

Рис. 18 Вигляд навігаційної панелі адміністратора застосунку

Головна сторінка адміністративного інтерфейсу надає зручний доступ до всіх наявних в застосунку типів сутностей.

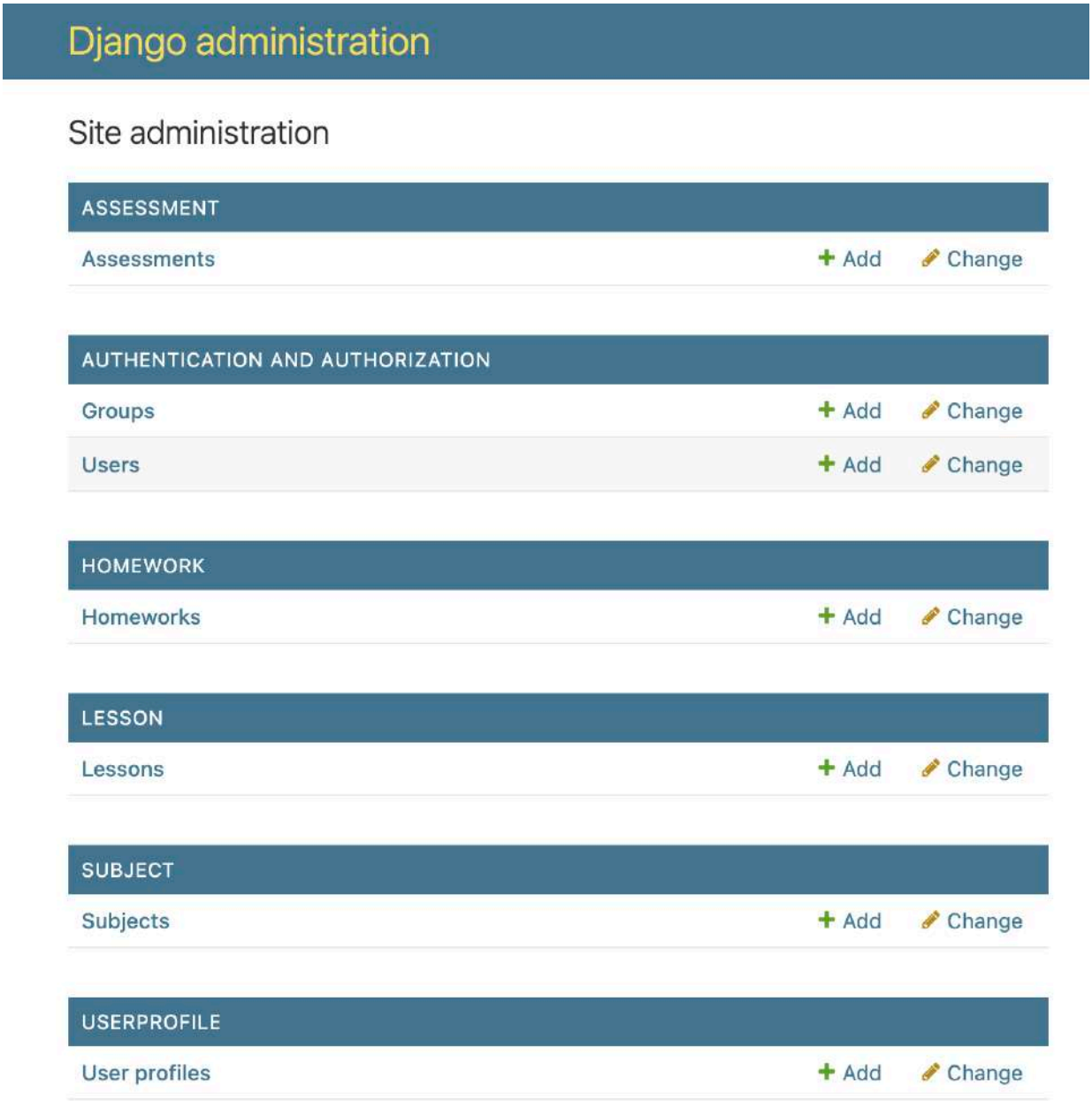

Рис. 19 Головна сторінка адміністративного інтерфейсу

Обравши конкретну сутність, адміністратор може переглянути всі її екземпляри, за потреби відфільтрувавши або відсортувавши їх.

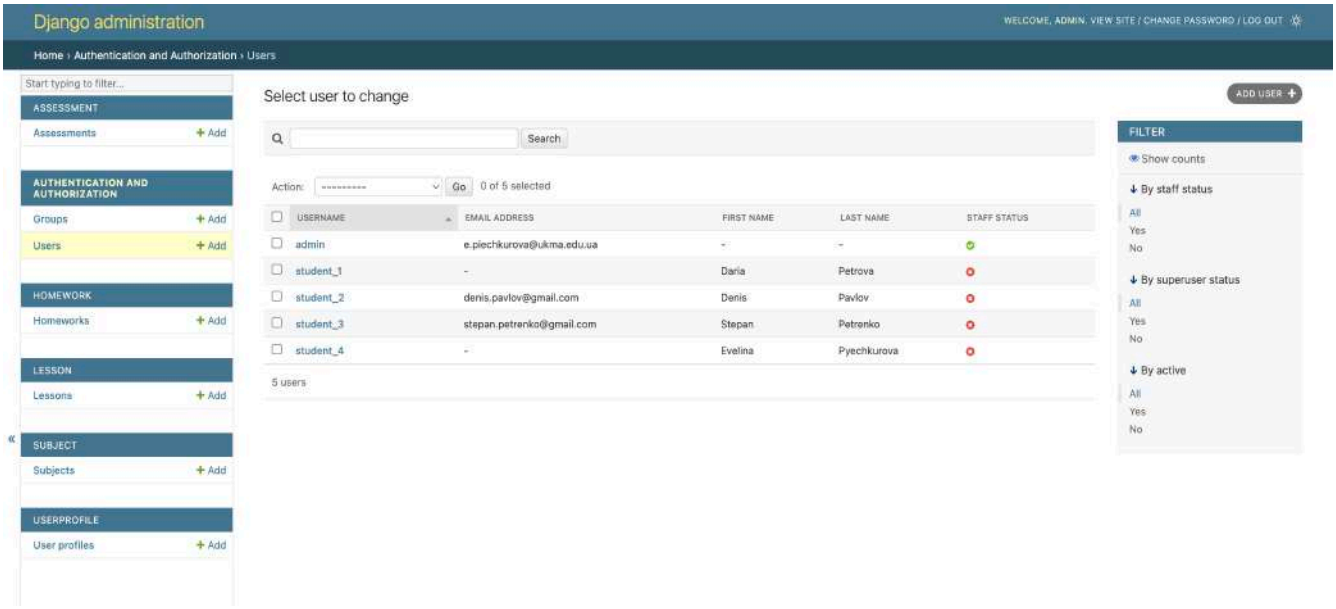

Рис. 20 Сторінка огляду інформації про всіх користувачів застосунку

Кожен екземпляр сутності доступний для перегляду в деталях, редагування або видалення.

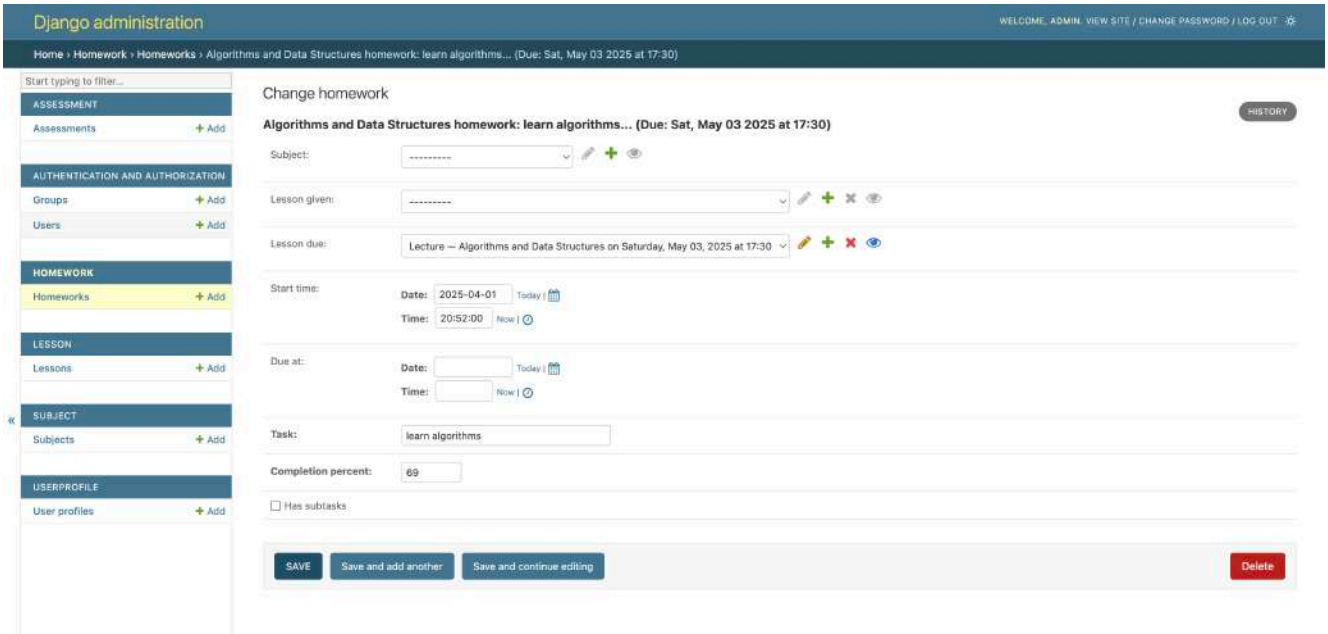

Рис. 21 Сторінка редагування інформації про екземпляр домашнього завдання

ВИСНОВОК ДО РОЗДІЛУ 3

У розділі 3 курсової роботи здійснено опис реалізації веб-додатку «Органайзер студента» за допомогою Django, з дотриманням архітектурного патерну MVT. Його код структуровано за допомогою окремих Django Application, кожен з яких реалізує окремий аспект логіки. Цей підхід забезпечив модульність, чітку структуру та масштабованість проекту.

Через те, що структура база даних була спеціально не повністю нормалізована для підтримки різних сценаріїв використання, спеціальні правила валідації та керування даними були розроблені та імплементовані шляхом перевизначення методів **clean()** та **save()** на рівні моделей. Таким чином були забезпечені як цілісність даних, так і гнучкість застосунку.

Для реалізації рівня представлень було використано надані Django class-based generic представлення, чия функціональність була розширена за допомогою механізму міксінів. Серед іншого, саме так було впроваджено механізм автентифікації.

Особливу увагу було приділено розробці ергономічного, інтуїтивно зрозумілого користувацького інтерфейсу.

Використання механізмів наслідування та агрегації Django Template Language спростило реалізацію інтерфейсу додатку.

ВИСНОВОК

Результатом виконання курсової роботи стала реалізація веб-застосунку «Органайзер студента». В процесі написання курсової роботи поставлена мета дослідження досягнута. Було проведено аналіз сучасних веб-фреймворків Django і Express.js та реалізовано додаток з використанням одного з них. Це дозволило поглибити знання сучасної веб-розробки і застосувати їх на практиці.

В процесі написання теоретичної частини курсової було визначено поняття фреймворку, досліджено його роль в розробці веб-застосунків та встановлено критерії для подальшого аналізу. При порівнянні Django та Express.js було встановлено, що вони дотримуються діаметрально протилежних підходів та філософій, однак кожен є ефективним для реалізації певного класу застосунків. Доказом цього є їх популярність серед розробників та використання в застосунках провідних компаній.

Розробка сервісу «Органайзер студента» складалася з двох етапів. На етапі проектування було визначено функціональні вимоги та необхідні компоненти майбутнього застосунку, і встановлено, що Django є оптимальним інструментом для його імплементації. Цей фреймворк підтримує архітектуру MVT та надає набір вбудованих компонентів, чим сприяє швидкій та ефективній реалізації логіки.

База даних спеціально не була повністю нормалізована для підтримки реальних сценаріїв, а узгодженість даних забезпечується за допомогою перевизначених методів clean() та save().

Реалізований застосунок підтримує два типи користувачів: здобувачів освіти та адміністраторів, а також надає необхідну їм автентифікацію та авторизацію. Користувацький інтерфейс реалізовано за допомогою Django Template Language. Він є зручним, інтуїтивно зрозумілим та візуально привабливим.

У підсумку, дана курсова робота поєднує в собі теоретичний аналіз з практичним застосуванням отриманих знань. В процесі її написання було сформоване цілісне уявлення про сучасні фреймворки та набуто цінних навичок для подальшої роботи в сфері веб-розробки.

Список використаної літератури

1. Mixin [Electronic resource]. — Mode of access: https://developer.mozilla.org/en-US/docs/Glossary/Mixin
2. API [Electronic resource] // Wikipedia. — Mode of access: https://en.wikipedia.org/wiki/API
3. Cross-site request forgery (CSRF) [Electronic resource]. — Mode of access: https://developer.mozilla.org/en-US/docs/Web/Security/Attacks/CSRF
4. What is a foreign key? (TL;DR) [Electronic resource]. — Mode of access: https://www.cockroachlabs.com/blog/what-is-a-foreign-key/
5. Difference Between ON DELETE CASCADE and ON DELETE SET NULL in DBMS [Electronic resource]. — Mode of access: https://www.geeksforgeeks.org/difference-between-on-delete-cascade-and-on-delete-set-null-in-dbms/
6. Об'єктно-реляційне відображення [Електронний ресурс] // Wikipedia. — Режим доступу: xbhttps://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D1%80%D0%B5%D0%BB%D1%8F%D1%86%D1%96%D0%B9%D0%BD%D0%B5_%D0%B2%D1%96%D0%B4%D0%BE%D0%B1%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%BD%D1%8F
7. Primary key [Electronic resource]. — Mode of access: https://www.techtarget.com/searchdatamanagement/definition/primary-key
8. SQL injection [Electronic resource] // Wikipedia. — Mode of access: https://en.wikipedia.org/wiki/SQL_injection
9. URL [Electronic resource] // Wikipedia. — Mode of access: https://en.wikipedia.org/wiki/URL

10. Cross-site scripting (XSS) [Electronic resource]. — Mode of access: https://developer.mozilla.org/en-US/docs/Web/Security/Atxtacks/XSS
11. What is a Framework? [Electronic resource] // GeeksforGeeks. — Mode of access: https://www.geeksforgeeks.org/what-is-a-framework/
12. What Is a Framework? [Electronic resource] // Codecademy Blog. — Mode of access: https://www.codecademy.com/resources/blog/what-is-a-framework/
13. What is a framework in programming and engineering? [Electronic resource] // Amazon. — Mode of access: https://aws.amazon.com/what-is/framework/
14. Web Frameworks: All You Should Know About [Electronic resource]. — Mode of access: https://www.browserstack.com/guide/web-development-frameworks
15. What is a Web Application? [Electronic resource] // Amazon. — Mode of access: https://aws.amazon.com/what-is/web-application/
16. SPA (Single-page application) [Electronic resource]. — Mode of access: https://developer.mozilla.org/en-US/docs/Glossary/SPA
17. Progressive web apps [Electronic resource]. — Mode of access: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps
18. A Guide to Web Development Frameworks [Electronic resource]. — Mode of access: https://builtin.com/articles/web-development-frameworks
19. RESTful API vs Server-Side Rendering in Web development – An in-depth comparison (with 5 use cases) [Electronic resource]. — Mode of access: https://dev.to/gem_corporation/restful-api-vs-server-side-rendering-in-web-development-an-in-depth-comparison-with-5-use-cases-2j9o
20. Common Web Security Vulnerabilities [Electronic resource]. — Mode of access: https://www.toptal.com/cybersecurity/10-most-common-web-security-vulnerabilities
21. Django (web framework) [Electronic resource] // Wikipedia. — Mode of access: https://en.wikipedia.org/wiki/Django_(web_framework)

22. Django's release process [Electronic resource] // Django. — Mode of access: https://docs.djangoproject.com/en/dev/internals/release-process/
23. BSD licenses [Electronic resource] // Wikipedia. — Mode of access: https://en.wikipedia.org/wiki/BSD_licenses#3-clause
24. Why Python is Called Interpreted Language [Electronic resource]. — Mode of access: https://www.geeksforgeeks.org/why-python-is-called-interpreted-language/
25. What does "batteries included" mean in the Django web framework? [Electronic resource]. — Mode of access: https://www.quora.com/What-does-batteries-included-mean-in-the-Django-web-framework
26. Design philosophies [Electronic resource] // Django. — Mode of access: https://docs.djangoproject.com/en/5.2/misc/design-philosophies/
27. URL dispatcher [Electronic resource] //Django. — Mode of access: https://docs.djangoproject.com/en/5.2/topics/http/urls/
28. Applications [Electronic resource] // Django. — Mode of access: https://docs.djangoproject.com/en/5.2/ref/applications/
29. How Django's MVT Architecture Works: A Deep Dive into Models, Views, and Templates [Electronic resource]. — Mode of access: https://www.freecodecamp.org/news/how-django-mvt-architecture-works/
30. Models [Electronic resource] // Django. — Mode of access: https://docs.djangoproject.com/en/5.2/topics/db/models/
31. Model Meta options [Electronic resource] // Django.— Mode of access: https://docs.djangoproject.com/en/5.2/ref/models/options/
32. Model instance reference [Electronic resource] // Django.— Mode of access: https://docs.djangoproject.com/en/5.2/ref/models/instances/
33. Migrations [Electronic resource] // Django.— Mode of access: https://docs.djangoproject.com/en/5.2/topics/migrations/

34. Databases [Electronic resource] // Django. — Mode of access: https://docs.djangoproject.com/en/5.2/ref/databases/
35. Writing views [Electronic resource] // Django. — Mode of access: https://docs.djangoproject.com/en/5.2/topics/http/views/
36. View decorators [Electronic resource] // Django. — Mode of access: https://docs.djangoproject.com/en/5.2/topics/http/decorators/
37. Class-based views [Electronic resource] // Django. — Mode of access: https://docs.djangoproject.com/en/5.2/topics/class-based-views/
38. Built-in class-based generic views [Electronic resource] // Django. — Mode of access: https://docs.djangoproject.com/en/5.2/topics/class-based-views/generic-display/
39. The Django template language: for Python programmers [Electronic resource] // Django. — Mode of access: https://docs.djangoproject.com/en/5.2/ref/templates/api/
40. Built-in template tags and filters [Electronic resource] // Django. — Mode of access: https://docs.djangoproject.com/en/5.2/ref/templates/builtins/
41. Security in Django [Electronic resource] // Django. — Mode of access: https://docs.djangoproject.com/en/5.2/topics/security/
42. Password management in Django [Electronic resource] // Django. — Mode of access: https://docs.djangoproject.com/en/5.2/topics/auth/passwords/
43. Using the Django authentication system [Electronic resource] // Django. — Mode of access: https://docs.djangoproject.com/en/5.1/topics/auth/default/
44. django.contrib.auth [Electronic resource] // Django. — Mode of access: https://docs.djangoproject.com/en/5.1/ref/contrib/auth/
45. Most popular technologies [Electronic resource]. — Mode of access: https://survey.stackoverflow.co/2024/technology#most-popular-technologies-webframe

46. Which Framework is better for you Django or Express JS? [Electronic resource]. — Mode of access: https://dev.to/zlaam/which-framework-is-better-for-you-django-or-express-js-453e#
47. Does Instagram still run on Django? [Electronic resource]. — Mode of access: https://www.reddit.com/r/django/comments/i3zyx2/does_instagram_still_run_on_django/
48. Express.js [Electronic resource] // Wikipedia. — Mode of access: https://en.wikipedia.org/wiki/
49. Overview of Blocking vs Non-Blocking [Electronic resource]. — Mode of access: https://nodejs.org/en/learn/asynchronous-work/overview-of-blocking-vs-non-blocking
50. An Intro to ExpressJS [Electronic resource]. — Mode of access: https://flexiple.com/express-js/deep-dive
51. Using middleware [Electronic resource] // Express. — Mode of access: https://expressjs.com/en/guide/using-middleware.html
52. Express vs. Django: 10 Indicators to Choose the True Backend King [Electronic resource]. — Mode of access: https://www.simform.com/blog/express-vs-django/
53. Production Best Practices: Security [Electronic resource] // Express. — Mode of access: https://expressjs.com/en/advanced/best-practice-security.html
54. Features [Electronic resource]. — Mode of access: https://www.passportjs.org/features/
55. Uber [Electronic resource] // Wikipedia. — Mode of access: https://uk.wikipedia.org/wiki/Uber
56. PayPal [Electronic resource] // Wikipedia. — Mode of access: https://uk.wikipedia.org/wiki/PayPal

57. What is ORM? Why is it used? What are its pros and cons? [Electronic resource]. — Mode of access: https://medium.com/@kadergenc/what-is-orm-why-is-it-used-what-are-its-pros-and-cons-3ed77c0e6ed2
58. The Django admin site [Electronic resource] //Django. — Mode of access: https://docs.djangoproject.com/en/5.2/ref/contrib/admin/
59. fat-client [Electronic resource]. — Mode of access: https://github.com/EvelinaPyechkurova/fat-client/blob/8d69dfd3fbf64b08a0200bd313104a62ead7aad0/src/index.css
60. How to Create/Extend Django Custom User Model [Electronic resource]. — Mode of access: https://rohitlakhotia.com/blog/django-custom-user-model/?utm_source=chatgpt.com

Додатки

Програмний код:

https://github.com/EvelinaPyechkurova/student-organizer