МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: Вебзастосунок «Домашній менеджер»

Здобувача вищої освіти першого (бакалаврського) рівня,
IV курсу, групи КБ-21-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології (за галузями)»

Тандоган Еркан-Даніель Махмутович

Науковий керівник _____ Лаговський Володимир Вікторович, к.е.н., доцент.

Керівник освітньо-професійної програми _____ Омельчук Антон Анатолійович, к.т.н.

Допущено до захисту за рішенням кафедри комп'ютерних та інформаційних технологій і систем протокол від «13» травня 2025 № 18

Завідувач кафедри _____ Філоненко Михайло Миколайович, к.ф.-м.н., доцент

Ірпінь – 2025

Реєстрація № 50 23.05.2025 _____
(дата) (підпис відповідальної особи)

Результати перевірки до захисту 23.05.2025 _____
(до захисту, на доопрацювання) (дата) (підпис)

Результат захисту 3 (задовільно) 04.06.2025 _____
(оцінка) (дата) (підпис)

Голова комісії: _____ В.М. Франчук
(підпис)

Члени комісії: _____ М.М. Філоненко
(підпис)

_____ С.М. Грищенко
(підпис)

_____ А.А. Омельчук
(підпис)

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії,секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст випускної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	**від 0 до 30**	18		**Лаговський В.В.** Франчук В.М.
2. Дотримання вимог щодо оформлення	**від 0 до 10**	7		Філоненко М.М. Грищенко С.М.
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	**від 0 до 5**	4		Омельчук А.А.
4. Рецензент	**від 0 до 5**	3		Суховецька Л.П.
РАЗОМ БАЛІВ	**від 0 до 50**	32		

Оцінювання кваліфікаційної роботи науковим керівником – 33 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалоюЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
65	D	3 (задовільно)	04.06.2025	Франчук В.М. Філоненко М.М. Грищенко С.М. Омельчук А.А. Суховецька Л.П.

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	**A**	**5 (відмінно)**
80-89	**B**	**4 (добре)**
70-79	**C**	
60-69	**D**	**3 (задовільно)**
50-59	**E**	
35-49	**FX**	**2 (незадовільно)**

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп'ютерні науки»
Освітня програма: «Інформаційні управляючі системи і технології (за галузями)»

Затверджую
Завідувач кафедри _____ Філоненко М.М.
«14» листопада 2024

Завдання на кваліфікаційну роботу
здобувачу вищої освіти
Тандогану Еркан-Даніелю Махмутовичу

1. Тема кваліфікаційної роботи: Вебзастосунок "Домашній менеджер" Затверджена наказом ректора від «14» листопада 2024 № 1717
2. Строк здачі закінченої роботи «08» травня 2025
3. Мета роботи: розробка вебзастосунка «Домашній менеджер»

Об'єкт дослідження: технології розробки вебзастосунку «Домашній менеджер»

Предмет дослідження: вебзастосунок «Домашній менеджер»

Орієнтовна база дослідження: дослідження ґрунтуються на сучасних методиках веб-розробки, існуючих прикладах застосунків для управління домашніми завданнями, а також на практичних рекомендаціях з розробки інтерфейсів користувача.

4. Вихідні дані до роботи: технічне завдання, що містить перелік функціональних вимог,макети інтерфейсу та протитипи основних сторінок.
5. Перелік графічного матеріалу: рисунки інтерфейса, скріншоти протитипу
6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)

Аналіз предметної області "Домашній менеджер, Огляд існуючих аналогів веб застосунків для домашнього, Обґрунтування вибору функціоналу для реалізації в рамках дипломної роботи, Вибір технологічного стеку для

розробки, Проектування архітектури веб застосунка, Проектування стру
бази даних, Проектування користувацького інтерфейсу (UI) та досвіду
Реалізація серверної частини (Backend), Реалізація клієнтської ча
(Frontend), Реалізація основних функціональних модулів.

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційноїроботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	30.10.2024	30.10.2024
2	І розділ (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	27.12.2024	27.03.2025
3	ІІ розділ (аналіз експериментальної частини про методики досліджень)	17.02.2024	3.05.2025
4	ІІІ розділ (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	27.04.2025	7.05.2025
5	Висновки	01.05.2025	10.05.2025
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	09.05.2025	27.05.2025
7	Отримання відгуку керівника	27.05.2025	27.05.2025
8	Отримання зовнішньої рецензії	29.05.2025	29.05.2025

8. Дата видачі завдання «14» листопада 2024
9. Керівник кваліфікаційної роботи _____ Лаговський
10. Керівник освітньо-професійної програми _____ Омельчук А
11. Завдання до виконання отримав здобувач вищої освіти _____ Тандоган Е

АНОТАЦІЯ

Мета роботи: розробка вебзастосунка «Домашній менеджер»

Завдання роботи є:

1. Аналіз вже існуючих домашніх менеджерів
2. Вибір технологічного стеку для розробки
3. Проектування архітектури веб застосунка
4. Проектування структури бази даних.
5. Реалізація серверної частини (Backend) і клієнтської частини (Frontend)

Предметом дослідження є вебзастосунок «Домашній менеджер»

Базою дослідження є: дослідження ґрунтуються на сучасних методиках веб-розробки, існуючих прикладах застосунків для управління домашніми завданнями, а також на практичних рекомендаціях з розробки інтерфейсів користувача

За результатами дослідження розроблено вебзастосунок «Домашній менеджер».

Одержані результати можуть бути використані для управління розрахунками за комунальні послуги.

Кваліфікаційна робота складається зі 47 сторінок без додатків, 15 рисунків, 40 джерел.

Ключові слова: домашній менеджер, інтерефейс, вебзастосунок.

Рік виконання кваліфікаційної роботи: 2025.

Рік захисту роботи: 2025.

ABSTRACT

Objective of the work: Development of a web application "Home Manager"

Tasks of the work are:

1. Analysis of existing home managers
2. Selection of a technology stack for development
3. Designing the web application architecture
4. Designing the database structure.
5. Implementation of the server-side (Backend) and client-side (Frontend)

The subject of the research is the web application "Home Manager"

The basis of the research is: the research is based on modern web development methodologies, existing examples of applications for managing household tasks, as well as practical recommendations for user interface development.

According to the research results, the web application "Home Manager" was developed.

The obtained results can be used to calculate utility meter expenses.

The qualification work consists of 47 pages without appendices, 15 figures, 40 sources.

Keywords: home manager, interface, web application.

Year of completion of the qualification work: 2025.

Year of defense of the work: 2025.

ЗМІСТ

ВСТУП 3

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ 5

1.1. Аналіз предметної області «Домашній менеджер» 5

1.2. Огляд існуючих аналогів веб-застосунків для домашнього управління . 8

1.3. Обґрунтування вибору функціоналу для реалізації в рамках дипломної роботи 11

РОЗДІЛ 2. ПРОЕКТУВАННЯ АРХІТЕКТУРИ ТА БАЗИ ДАНИХ ВЕБ ЗАСТОСУНКА 12

2.1. Вибір технологічного стеку для розробки. 13

2.2. Проектування архітектури веб застосунка 16

2.3. Проектування структури бази даних 21

2.4. Проектування користувацького інтерфейсу 26

РОЗДІЛ 3. РОЗРОБКА ВЕБ ЗАСТОСУНКА «ДОМАШНІЙ МЕНЕДЖЕР» 34

3.1 Реалізація серверної частини (Backend). 34

3.2 Реалізація клієнтської частини (Frontend). 39

3.3. Реалізація основних функціональних модулів. 40

ВИСНОВОК 45

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ **46**

ДОДАТКИ **50**

ВСТУП

Сучасний розвиток інформаційних технологій кардинально змінює всі аспекти людської діяльності, зокрема – організацію повсякденного домашнього господарства. Автоматизація рутинних процесів, використання веб-технологій для централізованого управління фінансами, завданнями, запасами та календарними подіями стають критично важливими для підвищення якості життя. У цьому контексті розробка комплексного вебзастосунку «Домашній менеджер» є надзвичайно актуальною, оскільки існуючі рішення ринку часто страждають від фрагментації функціоналу та недостатньої інтегрованості, що обмежує ефективність управління побутовими процесами.

Метою даної кваліфікаційної роботи є розробка сучасного вебзастосунку, який дозволить інтегрувати ключові елементи домашнього управління в єдиний зручний інтерфейс.

Досягнення поставленої мети передбачає вирішення наступних завдань:

1 Аналіз предметної області. Визначення основних потреб користувачів домашніх господарств, аналіз існуючих аналогів веб-застосунків та формулювання як функціональних, так і нефункціональних вимог до розроблюваної системи.

2 Проектування архітектури застосунку. Вибір відповідного технологічного стеку, розробка схеми бази даних із використанням ER-діаграм та визначення структури взаємодії між клієнтською та серверною частинами.

3 Розробка інтерфейсу користувача. Створення адаптивного, інтуїтивно зрозумілого та безпечного UI/UX-рішення, що забезпечить зручне управління всіма компонентами системи.

4 Реалізація та тестування сервісу. Впровадження серверної логіки на базі Python та Django, інтеграція з базою даних і проведення всебічного тестування розробленого рішення.

У процесі дослідження застосовано методи системного аналізу, моделювання даних, розробки програмного забезпечення з використанням

сучасних технологій (Python, Django, реляційні бази даних) та інструментів для побудови адаптивних інтерфейсів.

Практична цінність роботи полягає у створенні ефективного інструменту, який дозволить оптимізувати планування домашніх ресурсів, покращити контроль за фінансовими потоками та сприятиме більш зручному розподілу завдань серед членів родини. Отже, розробка вебзастосунку «Домашній менеджер» є важливим кроком на шляху впровадження інноваційних рішень у сфері управління побутовими процесами, що сприятиме як науковому дослідженню сучасних підходів до цифрової трансформації, так і практичному удосконаленню організаційних моделей життєдіяльності родин.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

Даний розділ покликаний дослідити предметну область дипломного проекту. Аналіз проводиться з метою виявлення ключових аспектів управління домашнім господарством, окреслення потреб потенційних користувачів і формулювання як функціональних, так і нефункціональних вимог до вебзастосунку «Домашній менеджер». Окрім цього, здійснюється детальний огляд існуючих рішень із зазначенням їх сильних і слабких сторін, що дозволяє обґрунтувати вибір основного функціоналу для першої версії продукту. Застосування цього підходу сприятиме подальшому успішному етапу проектування та розробки системи на базі Python.

1.1. Аналіз предметної області «Домашній менеджер»

Поняття «домашній менеджер» являє собою комплексний підхід до організації та управління домашнім господарством, що поєднує традиційні принципи менеджменту із сучасними технологічними засобами. У сучасному світі, де динаміка та ритм повсякденного життя значно прискорилися, необхідність чіткого планування та оптимізації побутових процесів набуває особливої актуальності для кожного домогосподарства. Система управління у цьому контексті спрямована на координування різноманітних завдань, що стосуються контролю фінансових потоків, моніторингу запасів, організації щоденних справ, планування подій і ефективної комунікації між членами родини, а також інтеграції зовнішніх даних, які впливають на функціонування господарства наведених на рисинку 1.1. Домашній менеджер виступає як система, що об’єднує у собі як автономні програмні рішення, так і елементи організаційної культури. Вона забезпечує можливість систематизації інформації про доходи та витрати, ефективне відстеження запасів продуктів і побутових речей, а також своєчасне сповіщення про наближення критичних значень. Важливо, що інтегровані інтерфейси гарантують зручність

користування та адаптивність до різноманітних пристроїв, що значно полегшує взаємодію користувача із системою, сприяючи підвищенню ефективності управлінських процесів і мінімізації рутинних зусиль. Система домашнього менеджера побудована за принципом інтеграції декількох модулів, кожен з яких відповідає за конкретну функцію. До них відносяться модулі управління фінансами, інвентаризації, планування завдань і організації календарних подій. Завдяки синергетичній взаємодії цих компонентів система здатна автоматизувати рутинні операції, знижувати потребу в ручному втручанні і забезпечувати оптимальне розподілення ресурсів у господарстві. Однією з ключових складових даної системи є інтеграція зовнішніх даних. Сучасні інформаційні технології відкривають можливості для отримання актуальної інформації з різних джерел, що стосуються стану економічного середовища, валютних курсів, комерційних пропозицій і інших чинників, які безпосередньо впливають на роботу домашнього господарства. Отримані дані дозволяють оперативно реагувати на зміни ринкових умов, коригувати бюджетні плани, забезпечувати прозорість фінансових операцій і таким чином підвищувати стабільність функціонування системи. У структурі домашнього менеджера значне місце займає модуль планування завдань, який дозволяє створювати детальні розклади побутових процесів, встановлювати пріоритети, дедлайни та додаткові коментарі. Система, що використовує автоматизоване нагадування, сприяє ефективному розподілу часу, дозволяє прогнозувати навантаження на окремі домашні процеси та координувати роботу членів родини для своєчасного виконання поставлених завдань. Такий підхід сприяє оптимізації як індивідуального, так і спільного планування, що є надзвичайно важливим для безперебійного функціонування домашнього господарства. Об’єктивною метою впровадження системи домашнього менеджера є модернізація існуючих методів управління, оптимізація внутрішніх процесів та підвищення якості життя кожного члена родини. Автоматизація рутинних операцій дозволяє зменшити витрати часу на виконання щоденних завдань, забезпечити більш точний контроль за фінансами, збалансувати використання матеріальних

ресурсів та утримувати стабільність господарської діяльності. Завдяки інтегрованому підходу можливе досягнення високих стандартів ефективності управління в домашньому господарстві. Ключовим чинником успішної реалізації концепції є постійний аналіз і вдосконалення системи на основі зворотного зв'язку. Регулярне оновлення програмного забезпечення, впровадження новітніх алгоритмів і розробка адаптивних рішень, що враховують специфіку кожного дому, дозволяють підтримувати актуальність системи та ефективно реагувати на нові виклики сучасного життя. Такий підхід забезпечує можливість масштабування системи та її адаптацію до мінливих умов зовнішнього середовища. Не менш важливим аспектом є врахування соціально-психологічних чинників. Ефективно організований розподіл обов'язків у родині, оптимізація побутових процесів і налагоджена комунікація між її членами сприяють зниженню стресового навантаження, розвитку взаєморозуміння та підтримці високого рівня якості життя. Централізоване управління інформацією дозволяє забезпечити прозорість планування, стимулює колективну відповідальність і сприяє розвитку культури взаємодопомоги в родинному середовищі. Таким чином, предметна область домашнього менеджера охоплює як технічні, так і соціокультурні аспекти організації домашнього господарства. Інтеграція сучасних технологій із традиційними методами управління дозволяє автоматизувати щоденні процеси, оптимізувати використання фінансових матеріальних ресурсів і забезпечити стабільність функціонування домогосподарства. Комплексний підхід сприяє ефективному контролю за виконанням завдань, швидкій адаптації до змін і покращенню якості життя сучасних родин.



Рис 1.1. Основні компоненти домашнього менеджеру

1.2. Огляд існуючих аналогів веб-застосунків для домашнього управління

Поняття «домашній менеджер» охоплює інтеграцію традиційних принципів менеджменту з сучасними цифровими технологіями для організації, управління та контролю за побутовими процесами. Сучасне домашнє господарство - це не лише фінансовий облік і інвентаризація, а й планування завдань, координація між членами сім'ї та інтеграція зовнішніх даних, що впливають на ефективність роботи всієї системи. Основна ідея полягає в тому, щоб через спеціалізоване програмне забезпечення забезпечити чітке розподілення відповідальності, оптимізацію бюджетних витрат та своєчасне виконання завдань. Однією з важливих складових системи є модуль управління фінансами. Сучасні рішення дозволяють вести детальний облік доходів і витрат, аналізувати дані за категоріями, прогнозувати майбутні витрати на базі історичних даних та формувати комплексні звіти. Наприклад, додатки для бюджетування, такі як Money Manager або Monefy, демонструють високий рівень інтеграції з банківськими сервісами та мобільними пристроями – вони автоматично оновлюють інформацію в режимі реального часу, що суттєво полегшує планування сімейного бюджету. Ще один ключовий аспект – організація повсякденних завдань. Сучасні системи управління завданнями не обмежуються лише нагадуваннями чи календарями. Вони дозволяють

користувачам створювати інтерактивні списки справ із можливістю встановлення дедлайнів, пріоритетів та додавання коментарів або фото підтверджень виконання завдань. Для прикладу, платформи, що орієнтуються на синхронізацію сімейних розкладів, дозволяють об’єднувати як особисті, так і робочі події у єдиному інтерфейсі – це створює можливість централізованого управління родинним часом. Системи управління запасами в домашньому господарстві надають змогу автоматизувати контроль за залишками продуктів, побутових речей або медикаментів. Використання інтелектуальних алгоритмів дозволяє вести інвентаризацію, аналізувати історичні дані споживання та своєчасно формувати списки покупок. Так, деякі рішення дозволяють надсилати сповіщення при досягненні критичних рівнів запасів, що забезпечує оперативне реагування на зміни в побутовій ситуації. Щодо інтеграції зовнішніх даних, вона дозволяє домашньому менеджеру отримувати актуальну інформацію щодо валютних курсів, знижок і акцій від супермаркетів або банківських сервісів. Це значно підвищує точність бюджетного планування та допомагає коригувати фінансові стратегії в режимі реального часу. Наприклад, у деяких сучасних рішеннях інтеграція з API зовнішніх сервісів дозволяє автоматично оновлювати дані, що в свою чергу забезпечує більш адаптивну роботу системи. Щодо практичної реалізації, приклади реальних систем демонструють, як сучасна технологія може змінити підхід до управління домашнім господарством. Home Assistant — це відкрита платформа для розумного будинку, що дозволяє інтегрувати численні пристрої, керувати опаленням, освітленням та системами безпеки через єдиний інтерфейс. Google Nest забезпечує глибоку інтеграцію з мобільними пристроями, синхронізуючи календарні події, нагадування і навіть аналізуючи звички користувачів для оптимізації розкладу. FamilyWall орієнтований на спільне планування сімейних справ, ведення бюджету та координацію завдань між членами родини, що особливо важливо для родин з різними графіками роботи.

Кожен із зазначених прикладів має свої сильні сторони: інтегрованість з іншими сервісами, зручність користування, високий рівень автоматизації та гнучкість налаштувань. Наприклад, можливість використання алгоритмів машинного навчання у Home Assistant дозволяє заздалегідь прогнозувати зміни температури або надсилати автоматичні повідомлення, а Google Nest адаптує розклад завдяки аналізу поведінки користувачів. У цьому контексті перевага полягатиме не лише в технічних характеристиках, а й в здатності системи інтегрувати різноманітні сервіси в єдиний продукт, що робить його більш зручним для кінцевого користувача зображених на рисунку 1.2. Отже, аналіз існуючих аналогів показує, що сучасні вебзастосунки для домашнього управління мають низку переваг. Серед них – потужна інтеграція з банківськими та мобільними сервісами, можливість автоматизованого обліку фінансів і завдань, а також адаптивність до індивідуальних потреб користувачів. Проте на ринку все ще спостерігається фрагментація функціоналу: багато рішень орієнтовані лише на окремі аспекти управління, що створює потребу для розробки єдиного комплексного додатку, який об’єднає всі необхідні функції для ефективного управління домашнім господарством. Цей підхід забезпечить не лише оптимізацію ресурсів, але й підвищення якості життя членів родини, створюючи комфортне та злагоджене середовище побутового менеджменту.

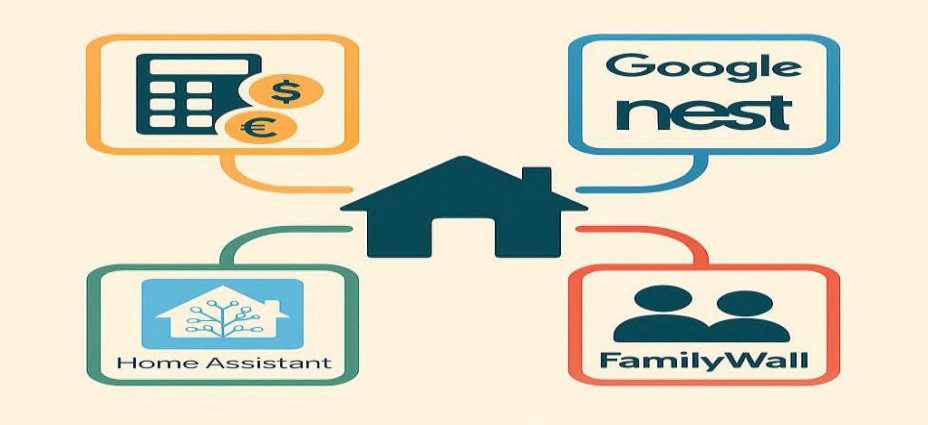


Рис. 1.2. Аналоги вебзастосунків

1.3. Обґрунтування вибору функціоналу для реалізації в рамках дипломної роботи

Вибір функціоналу для вебзастосунка «Домашній менеджер» є ключовим етапом, що визначає його конкурентоспроможність, корисність для кінцевого користувача. Обґрунтування даного вибору базується на попередньому аналізі предметної області та наявних аналогів, із фокусом на створення мінімально життєздатного продукту (MVP), який буде вирішувати найнагальніші проблеми цільової аудиторії. Проаналізувавши функціональні потреби сучасних домогосподарств, було встановлено, що найбільший запит спостерігається в напрямках контролю фінансів, організації спільних завдань, створення та управління списками покупок, а також фіксації різного роду нотаток. Існуючі аналоги часто або надто вузько спеціалізовані, або перевантажені малозрозумілим функціоналом. З урахуванням обраного стеку технологій (Python/Django), було виділено п’ять ключових функціональних модулів наведені на рисунку 1.3 реалізація яких забезпечує баланс між функціональністю та простотою використання.

1. Модуль управління фінансами

Забезпечує фіксацію доходів та витрат, дозволяє створювати бюджети, візуалізувати грошові потоки через графіки та звіти. Особливістю модуля є можливість прив’язки витрат до подій або покупок, що забезпечує цілісність у веденні домашньої бухгалтерії. Також підтримується ведення спільних рахунків.

2. Модуль завдань і календаря

Відповідає за створення списків справ, призначення завдань членам родини, встановлення дедлайнів і нагадувань. Вбудований календар синхронізує важливі події, візити та інші життєві плани.

3. Модуль списків покупок

Дозволяє створювати тематичні списки покупок, які можуть спільно редагуватись в реальному часі. Користувачі мають змогу відмічати придбані товари, що мінімізує дублювання та підвищує зручність використання.

4. Модуль нотаток

Призначений для зберігання важливої текстової інформації (рецепти, інструкції, списки контактів). Нотатки можна структурувати за категоріями та відкривати доступ іншим користувачам родини.

5. Модуль витрат на продукти

Додатковий функціонал, що сфокусований на обліку витрат саме на продукти. Дозволяє фіксувати дату, суму та категорію витрати, візуалізувати динаміку через графіки та діаграми, а також інтегрується з основним модулем фінансів для більш повного контролю бюджету.

Загалом, такий набір модулів забезпечує комплексне покриття основних потреб користувачів, зберігаючи при цьому простоту інтерфейсу та високу швидкість виконання. Всі компоненти було протестовано в умовах модульної архітектури Django, що дозволяє у майбутньому легко масштабувати застосунок та реалізовувати нові сервіси, наприклад — мобільні версії або інтеграцію зі сторонніми сервісами.

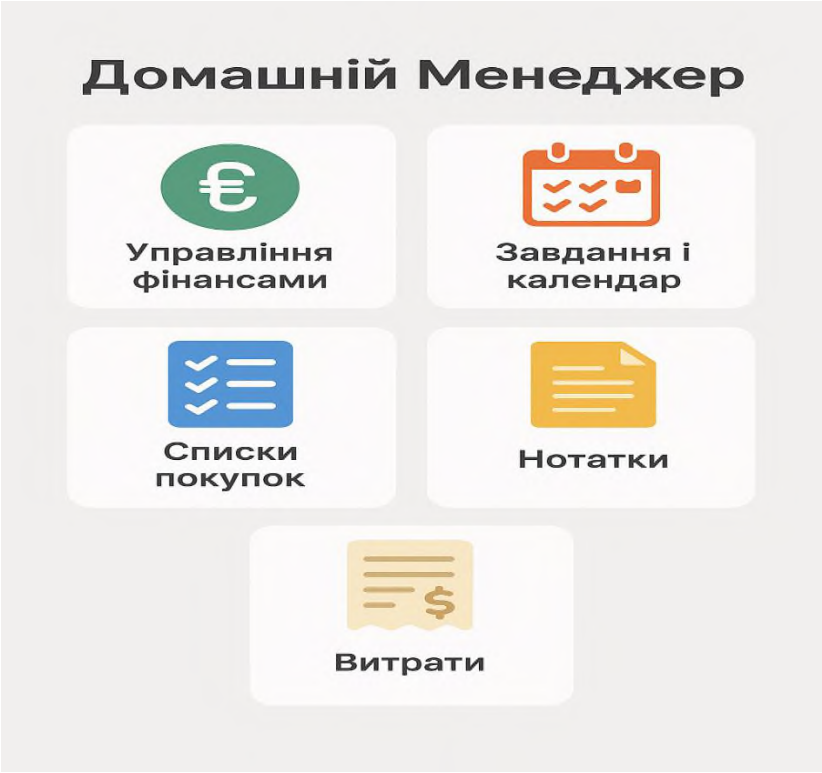


Рис 1.3. Перелік всіх модулів.

РОЗДІЛ 2. ПРОЕКТУВАННЯ АРХІТЕКТУРИ ТА БАЗИ ДАНИХ ВЕБ ЗАСТОСУНКА

У цьому розділі розкривається процес формування основної технічної бази для розробки веб застосунку «Домашній менеджер» Розглядаються стратегічні рішення щодо вибору технологічного стеку, який не лише відповідає сучасним вимогам щодо продуктивності, масштабованості та безпеки, але й гармонійно вписується у загальну концепцію системи, окреслену у вступі та в першому розділі. Обрання правильного інструментарію для розробки і розгортання системи є критично важливим для досягнення високої якості кінцевого продукту, забезпечення його гнучкості та можливості подальшого розширення функціоналу .Основною задачею є побудова архітектури, яка дозволить ефективно реалізувати логіку роботи як серверної, так і клієнтської частин застосунку, при цьому гарантувати безперебійну обробку запитів, зберігання великого обсягу структурованих даних та організацію оперативного зворотного зв’язку з користувачем. Система повинна демонструвати високу продуктивність при змінній інтенсивності використання, легко адаптуватися до майбутніх змін у вимогах та забезпечувати безперервну інтеграцію нових сервісів.

2.1. Вибір технологічного стеку для розробки.

При розробці сучасного веб-застосунку Домашній менеджер надзвичайно важливим є не тільки забезпечення стабільності, безпеки й масштабованості системи, але й вибір технологічного стеку, що відповідає потребам швидкого прототипування та подальшої розширюваності продукту. При визначенні оптимальних рішень для створення веб-застосунку ретельно аналізувалися різноманітні мови програмування та супутні фреймворки, що використовуються для реалізації серверної та клієнтської частин. Серверна частина традиційно реалізується за допомогою мов програмування, таких як

PHP, Java, C#, Ruby, JavaScript (Node.js), Go, Python. PHP протягом багатьох років залишалася популярною через свою прискорену розробку веб-сайтів і великий набір готових рішень, однак її синтаксис і підтримка сучасних парадигм часто викликають питання щодо підтримки чистоти коду. Java та C# забезпечують високу продуктивність і масштабованість, проте вимагають більше зусиль для розробки через суворий синтаксис і розлогий обсяг типового шаблонного коду. Ruby, on Rails, давно славиться своєю елегантністю та принципом «Convention over Configuration», однак його ринок не виріс так стрімко, як у випадку з іншими мовами, а підтримка фреймворку менша порівняно з альтернативами. Node.js, що базується на JavaScript, дозволяє створювати асинхронні, високопродуктивні серверні застосунки, але його модель обробки запитів може бути складною для розробників, які шукають простоту реалізації. Go демонструє чудову продуктивність і конкурентну здатність при високому навантаженні, проте екосистема бібліотек все ще розвивається.

Серед усіх перелічених мов Python вирізняється своєю читабельністю, лаконічністю та величезною кількістю бібліотек, що значно полегшує впровадження складних алгоритмів, обробку даних та взаємодію з базами даних. Python має активну спільноту та постійно отримує оновлення відповідно до сучасних стандартів, що дає змогу швидко розробляти прототипи та впроваджувати інноваційні рішення. Саме тому для серверної частини дипломного проекту було обрано Python.

Щодо фреймворків, для розробки веб-застосунків існують численні рішення: Django, Flask, Ruby on Rails, ASP.NET, Express.js, Spring Boot, Laravel тощо. Flask, як легковаговий мікрофреймворк для Python, надає високий рівень свободи, але вимагає додаткової роботи для забезпечення повноцінного набору функцій (автентифікація, валідація, адміністрування), що може бути недоцільно для комплексних систем. ASP.NET і Spring Boot, хоч і забезпечують високу продуктивність, обмежують розробку через тісну інтеграцію з платформами Microsoft або Java, що збільшує складність налаштування середовища та

підтримки. Express.js, який працює на Node.js, добре підходить для створення швидких серверних застосунків, але його асинхронна модель може призводити до складного управління кодом у великих проектах. Laravel для PHP забезпечує гарну організованість коду, проте PHP як мова має свої обмеження при роботі зі складними системами, що вимагають високої продуктивності та строгих стандартів безпеки. Обрання Python і фреймворку Django обґрунтовано наступними основними чинниками. По-перше, Python забезпечує високу читабельність коду, що дозволяє розробникам швидко засвоїти принципи роботи і зменшує витрати часу на дебаг і оновлення. По-друге, Django є потужним, «батарейки в комплекті» фреймворком, який забезпечує всі необхідні компоненти для побудови масштабованих веб-застосунків: модульну архітектуру, вбудовану систему адміністрування, інтегровану обробку запитів, високий рівень безпеки та підтримку REST API. По-третє, Django дозволяє зосередитись на розв’язанні бізнес-задач, скорочуючи кількість рутинної розробки завдяки широкій бібліотеці готових рішень. По-четверте, інтеграційні можливості Django забезпечують легке підключення до зовнішніх API, що особливо актуально для сучасних інтерактивних застосунків. До того ж, спільнота розробників Python і Django постійно працює над удосконаленням фреймворку, що гарантує його актуальність і високий рівень підтримки.

Таким чином, детальний аналіз конкурентних мов програмування та фреймворків показує, що хоча існують численні технологічні рішення, Python у поєднанні з Django є оптимальним вибором для розробки веб-застосунку Домашній менеджер,що розповідає цей рисунок 2.1. Це обгрунтування базується на таких критеріях, як читабельність і швидкість розробки, модульність архітектури, високий рівень інтеграційних можливостей і підтримка сучасних стандартів безпеки. Обраний стек дозволяє створити стабільний, масштабований і ефективний продукт, який відповідає сучасним вимогам ринку та забезпечує високий рівень задоволеності кінцевих користувачів. Завдяки цьому поєднанню Python і Django наш проект отримує міцну технологічну основу для подальшої еволюції і впровадження нових

функцій відповідно до потреб цифрової трансформації управління домашнім менеджером.



Рис. 2.1. Переваги Python над іншими мовами

2.2. Проектування архітектури веб застосунка

При проектуванні архітектури веб-застосунку «Домашній менеджер» особлива увага приділяється створенню міцного, гнучкого та масштабованого рішення, яке ефективно об’єднує взаємодію між серверною логікою, клієнтським інтерфейсом і системою зберігання даних. На цьому етапі визначаються ключові компоненти системи, обирається відповідний архітектурний патерн, продумується взаємодія між усіма модулями та впроваджуються механізми високої продуктивності і безпеки. У даному проєкті було обрано класичний патерн Model-View-Controller (MVC) як базовий підхід, що забезпечує чітке розділення обов’язків: бізнес-логіка розміщується в моделях, обробка запитів і маршрутизація – у контролерах (views), а представницький шар реалізується через шаблони. Такий підхід дозволяє не

лише спростити тестування та налагодження, а й за потреби безболісно перейти до мікросервісної архітектури за умов зростання навантаження чи спеціалізації функціональних блоків. Структура проекту організована таким чином, що файли й каталоги логічно розділено відповідно до їхньої ролі у застосунку. Проєкт містить головну папку «home_management_project/», у якій розташовано декілька підкаталогів і файлів. Каталог venv/ містить віртуальне середовище, що ізолює необхідні залежності проекту від глобальних налаштувань операційної системи. Далі слідує папка config/, яка містить конфігураційні файли для всього проєкту Django:

– файл init.py ініціалізує пакет конфігурації;

– asgi.py містить налаштування для ASGI-сумісних веб-серверів, що забезпечує підтримку асинхронної обробки запитів;

– settings.py централізує всі параметри проєкту, включаючи конфігурацію бази даних, перелік встановлених додатків, шляхи до статичних файлів і шаблонів, а також інші конфіденційні дані, такі як секретний ключ;

– urls.py виконує роль головного файлу маршрутизації, де визначається зв’язок між URL-запитами і відповідними контролерами, часто включаючи маршрути від окремих додатків;

– wsgi.py визначає конфігурацію для розгортання застосунку через WSGI-сумісні сервери, що є стандартним для виробничого середовища. Основний додаток – management/ – містить ключову функціональність системи «Домашній менеджмент». У цьому каталозі розміщено:

– init.py, що інтерпретує дану директорію як Python-пакет;

– admin.py, за допомогою якого моделі реєструються у вбудованій адмін-панелі Django для зручного управління даними;

– apps.py, який містить конфігурацію додатку та визначення його назви;

– Каталог migrations/ з файлами, що описують еволюцію схеми бази даних, дозволяє відслідковувати та впроваджувати зміни у структурі таблиць;

– models.py описує структуру бази даних – кожна модель відповідає таблиці, а атрибути відображають поля, що забезпечує роботу через ORM без необхідності написання складних SQL-запитів;

– tests.py містить набір автоматизованих тестів для перевірки функціональності додатку;

– views.py реалізує логіку обробки HTTP-запитів – контролери, які отримують дані з моделей, обробляють їх і передають у шаблони для генерації відповіді;

– каталог templates/management/ містить HTML-шаблони, специфічні для додатку «Домашній менеджер», що забезпечує динамічне формування інтерфейсу користувача та уникнення конфліктів імен між різними додатками;

– файл urls.py (для додатку) відповідає за локальну маршрутизацію запитів, що направляються до функціональних можливостей саме цього компонента.

Файл db.sqlite3 містить базу даних SQLite, що за замовчуванням використовується для зберігання інформації, забезпечуючи узгодженість і цілісність даних, що критично важливо для обліку фінансових операцій, завдань та календарних подій. Нарешті, manage.py є основним інструментом командного рядка для управління проєктом – запуск сервера, створення міграцій, виконання тестів та інші адміністративні завдання. Архітектурна модель застосунку ґрунтується на інтегрованій взаємодії трьох основних компонентів. Першим є серверна частина (backend), розроблена на Python із застосуванням фреймворків, таких як Django або Flask, що відповідає за бізнес-логіку, обробку запитів і взаємодію з базою даних через ORM. Використання Python дозволяє швидко розгортати прототипи і інтегрувати численні бібліотеки для валідації, безпеки та роботи з даними, що відповідає сучасним вимогам веб-застосунків, орієнтованих на керування домашнім господарством.

Другим компонентом є клієнтська частина (frontend), розроблена за допомогою популярних JavaScript-фреймворків, таких як React або Vue.js, що дозволяють створювати динамічний, інтуїтивно зрозумілий та адаптивний

інтерфейс. Завдяки модульному підходу ці фреймворки забезпечують розбивання інтерфейсу на малючі, повторно використовувані компоненти, що сприяють зручному управлінню кодом і швидкому перемиканню між елементами, а також забезпечують ефективну взаємодію з серверною частиною через RESTful API або GraphQL-запити. Третім компонентом є система зберігання даних, де для критично важливої інформації використовується реляційна база даних (наприклад, PostgreSQL), що забезпечує потужні можливості обробки SQL-запитів, підтримує транзакційну цілісність і зберігає дані у структурованому форматі. У випадках, коли потрібна оптимізація швидкості доступу або обробки великих обсягів неструктурованої інформації, застосовуються NoSQL-рішення, такі як MongoDB чи Redis, що дозволяє використати переваги обох типів баз даних для підвищення гнучкості та продуктивності.

Проектування архітектури здійснюється за принципами чіткого розділення відповідальностей та інтеграції між компонентами. Коли користувач взаємодіє з клієнтським інтерфейсом, його запити надсилаються через стандартизовані API до серверної частини, де контролери валідують отриману інформацію, обробляють її згідно з бізнес-логікою та виконують маніпуляції з даними через моделі. Результати обробки повертаються у вигляді HTTP-відповідей, які одразу відображаються в інтерфейсі, забезпечуючи безперебійну комунікацію між фронтендом і бекендом. Наявність чітко структурованого API сприяє моніторингу кожної транзакції, автоматизованому модульному тестуванню та легкій інтеграції нових функціональних можливостей без потреби у масштабних перебудовах інфраструктури. Особлива увага приділяється забезпеченню високого рівня безпеки і захисту даних: комунікація між клієнтом і сервером здійснюється через захищені канали за допомогою протоколів TLS/SSL, застосовується багатофакторна автентифікація, а спеціалізовані механізми логування і моніторингу допомагають оперативно реагувати на потенційні загрози. Розширена архітектурна модель враховує також аспекти масштабованості: завдяки

використанню патерну MVC кожен компонент можна розглянути окремо і, при необхідності, трансформувати в незалежний сервіс або мікросервіс що демонструє рисунок 2.2 І це дозволяє розподілити навантаження між кількома серверами чи в хмарному середовищі, зменшити ризик виникнення «вузьких місць» і створити умови для подальшого зростання функціональних можливостей. Впровадження асинхронної обробки запитів, наприклад через AJAX на фронтенді, у поєднанні з розумним кешуванням на рівні бази даних і використанням NoSQL-технологій, забезпечує оперативну реакцію системи навіть при високому навантаженні. Інтеграція з системами моніторингу, такими як Prometheus і Grafana, дозволяє постійно відстежувати стан серверів, баз даних і мережевої інфраструктури, що критично важливо для забезпечення високої відмовостійкості. Також використовуються сучасні DevOps-практики: автоматизація процесів тестування, збірки і розгортання коду за допомогою CI/CD-інструментів (Jenkins, GitHub Actions, GitLab CI) та контейнеризація (Docker) дозволяють підтримувати відтворюваність середовища та ефективно управляти ресурсами навіть у разі зростання кількості користувачів. Цей опис архітектурної моделі демонструє інтегровану екосистему, в якій кожен компонент – від серверної обробки запитів до клієнтського інтерфейсу і системи зберігання даних – взаємодіє через чітко структуровані API. Розподілення відповідальностей за допомогою патерну MVC, застосування безпечних комунікацій, асинхронна обробка запитів, гнучкість використання реляційних і NoSQL-баз, а також впровадження DevOps-практик забезпечують високий рівень продуктивності, безпеки і масштабованості системи. Таке детальне проєктування гарантує, що веб-застосунок Домашній менеджер здатний ефективно управляти всіма аспектами домашнього менеджера в умовах сучасних цифрових трансформацій і зростаючих вимог користувачів.

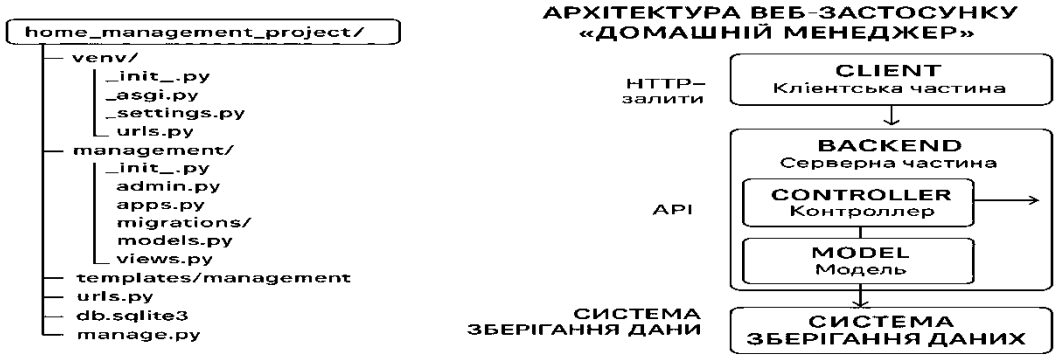


Рис 2.2. Архітектура веб-застосунку домашний менеджер

2.3. Проектування структури бази даних

При проектуванні структури бази даних для веб-застосунку Домашній менеджер основна увага приділяється розробці гнучкого, масштабованого та високопродуктивного рішення, яке дозволяє ефективно здійснювати управління домашніми фінансами, витратами, бюджетами, а також контролювати комунальні послуги.

Обрана модель базується на принципах реляційного підходу, що забезпечує цілісність, узгодженість та структурованість інформації, а також дозволяє використовувати потужні механізми SQL для обробки даних. Ретельне проектування схеми бази даних є критичним етапом у розробці систем, орієнтованих на управління домашнім господарством, оскільки правильне структуроване зберігання даних гарантує швидкий доступ до інформації, зменшує можливості помилок і спрощує масштабування системи при зростанні навантаження. Однією з базових складових системи є стандартна модель аутентифікації користувачів, що реалізується через таблицю auth_user. Ця таблиця, яка входить до стандартного функціоналу Django, містить численні поля, що відповідають за зберігання критично важливої інформації про користувачів. Серед основних полів можна виділити поле id (тип AutoField, що відповідає цілочисловому значенню з автоінкрементом та слугує первинним ключем), password (рядкове поле для збереження гешованого пароля), last_login (тип DateTimeField для реєстрації останнього часу входу), is_superuser (булеве

значення, що визначає суперкористувача), а також поля username, first_name, last_name, email, is_staff, is_active і date_joined. Окрім цього, Django реалізує додаткові механізми для обробки групових прав та дозволів через відповідні зв’язки. Структура цієї таблиці є фундаментальною для організації системи аутентифікації і авторизації, адже забезпечує найвищий рівень безпеки та контролю доступу до ресурсів застосунку.

На базі цієї моделі безпосередньо розширюється таблиця FamilyMember, яка представляє додаткову інформацію про членів домогосподарства. Ця таблиця містить поле id (PrimaryKey, AutoField), а також створює зв’язок один-до-одного з таблицею auth_user через поле user_id (тип OneToOneField, що реалізує зовнішній ключ до auth_user з поведінкою ON DELETE CASCADE). Таким чином, кожен користувач отримує унікальний профіль, який може бути розширений додатковими атрибутами, такими як роль у сім’ї (хоча, зауважується, дане поле може бути передбачене, але не реалізоване в поточній структурі). Використання моделі FamilyMember дозволяє гнучко розширювати інформацію про користувачів та адаптувати систему під специфіку обліку членів домогосподарства. Таблиця ExpenseCategory відповідає за класифікацію витрат користувача. Вона містить поле id (AutoField, первинний ключ) та name (CharField, обмеження VARCHAR(100)), яке є унікальним і визначає назву категорії.

Поле description (TextField) надає можливість зберігати детальний опис категорії витрат і може приймати значення NULL, що робить її необов’язковою. Таке розбиття інформації на категорії дозволяє більш ефективно групувати витрати та виконувати статистичний аналіз, що є суттєвим для побудови систем бюджетного планування. Надалі таблиця Expense забезпечує зберігання детальної інформації про кожну окрему витрату. Вона містить поле id (AutoField), важливе числове поле amount (DecimalField з точністю до 2 десяткових знаків), поле date (DateField із встановленням поточної дати як значення за замовчуванням) та опціональне текстове поле description (TextField, що допускає NULL). Ключовими є також зв’язки з

іншими таблицями: поле category_id (ForeignKey до ExpenseCategory, з поведінкою ON DELETE SET NULL) дозволяє приписати кожну витрату до певної категорії, а поле added_by_id (ForeignKey до FamilyMember) визначає, хто саме додав дану витрату, що також має поведінку ON DELETE SET NULL. Поля created_at та updated_at (DateTimeField з параметрами auto_now_add та auto_now) автоматично фіксують час створення та останнього оновлення запису. Така структура таблиці Expense забезпечує можливість детальної фіксації кожного витратного надходження, що є ключовою для аналізу та управління фінансовими потоками в застосунку. Таблиця Budget розроблена для зберігання бюджетних лімітів, що встановлюються для кожної категорії витрат окремо для певного місяця і року.

Основні поля таблиці включають id (AutoField) та amount_limit (DecimalField з точністю до 2 десяткових знаків), що задає ліміт для відповідного бюджету. Поля month (PositiveIntegerField) та year (PositiveIntegerField) визначають період, на який встановлено бюджет. Завдяки унікальному обмеженню (unique_together) для комбінації полів category_id, month та year база даних гарантує, що для кожної категорії витрат для заданого періоду існує лише один запис, що запобігає дублюванню даних і забезпечує консистентність бюджетної інформації. Для контролю за комунальними послугами застосунок використовує моделі UtilityService і UtilityRate. Таблиця UtilityService містить поля id (AutoField) і name (CharField з довжиною до 100 символів), що є унікальним ідентифікатором послуги, а також поле unit_of_measurement (CharField до 50 символів), яке визначає одиницю виміру (наприклад, куб. м або кВт·год). Таблиця UtilityRate зберігає інформацію про тариф на конкретну послугу: вона містить поле id (AutoField), поле service_id (ForeignKey до UtilityService з ON DELETE CASCADE), що встановлює зв’язок між тарифом і послугою, а також поле rate_per_unit (DecimalField з точністю до 4 десяткових знаків) і effective_date (DateField із значенням за замовчуванням рівним поточній даті). У цій моделі також застосовано обмеження унікальності

для комбінації service_id та effective_date, що гарантує відсутність дублювання тарифних записів для однієї послуги одночасно.

Модель MeterReading призначена для зберігання показників лічильників комунальних послуг. Таблиця включає поле id (AutoField), поле service_id (ForeignKey до UtilityService, що забезпечує зв’язок з конкретною послугою), reading_value (DecimalField з точністю до 3 десяткових знаків), а також поле reading_date (DateField, за замовчуванням поточна дата) та поле notes (TextField, що допускає NULL). У таблиці накладено унікальне обмеження для комбінації service_id та reading_date, що гарантує, що за кожну дату буде збережено лише один показник лічильника для кожної послуги. Останньою за функціональністю є таблиця UtilityPayment, яка зберігає інформацію про платежі за комунальні послуги. Цей компонент бази даних містить поле id (AutoField), поле service_id (ForeignKey до UtilityService із поведінкою ON DELETE CASCADE), що вказує на послугу, поле amount_paid (DecimalField з точністю до 2 десяткових знаків), яке фіксує суму платежу, а також payment_date (DateField із встановленням поточної дати за замовчуванням). Додатково, таблиця містить поля billing_period_start та billing_period_end (обидва DateField), які вказують на початок і кінець розрахункового періоду, поле transaction_id (CharField до 255 символів) для зберігання ідентифікатора транзакції або квитанції, а також поле notes (TextField), що може містити додаткові примітки і допускає значення NULL. Деякі коментарі до моделі свідчать про можливе додавання зв’язку з MeterReading, але це не було активовано в даній версії структури.

При розробці цієї бази даних було враховано принципи нормалізації даних, що дозволяють усунути надлишковість інформації і зменшити ризик виникнення аномалій при оновленні. Використання зовнішніх ключів, унікальних обмежень і налаштувань поведінки при видаленні (ON DELETE CASCADE, SET NULL) забезпечує підтримку цілісності даних у всіх таблицях. Такий підхід гарантує, що зміни в одному компоненті бази даних автоматично відображатимуться в залежних таблицях, що є суттєвим для динамічного управління інформацією про домашні фінанси, витрати, бюджети і комунальні

послуги. Структура бази даних, розроблена для застосунку Домашній менеджер, відзначається високим рівнем масштабованості – у разі зростання обсягу даних або у разі потреби у більш складній обробці інформації, її можна модифікувати шляхом додавання нових таблиць чи зв’язків, або ж впровадженням NoSQL-рішень для зберігання неструктурованих даних.

Окрім того, використання стандартних моделей Django, таких як auth_user, у парі з додатковими таблицями (наприклад, FamilyMember) сприяє інтегрованню системи аутентифікації й авторизації з іншими компонентами застосунку, що дозволяє ефективно керувати користувацькими правами і забезпечувати безпеку доступу до інформації. Проектування такої комплексної структури бази даних є результатом ґрунтовного аналізу потреб користувачів і технічних вимог до веб-застосунку «Домашній менеджер». Докладна специфікація таблиць, полів, зв’язків і обмежень забезпечує можливість подальшого розширення функціоналу системи, сприяє підтримці високої продуктивності при взаємодії різних компонентів і гарантує надійний захист даних. Ретельне документування структури бази є основою для успішної розробки, тестування та експлуатації системи, що відповідає стандартам сучасної цифрової трансформації.

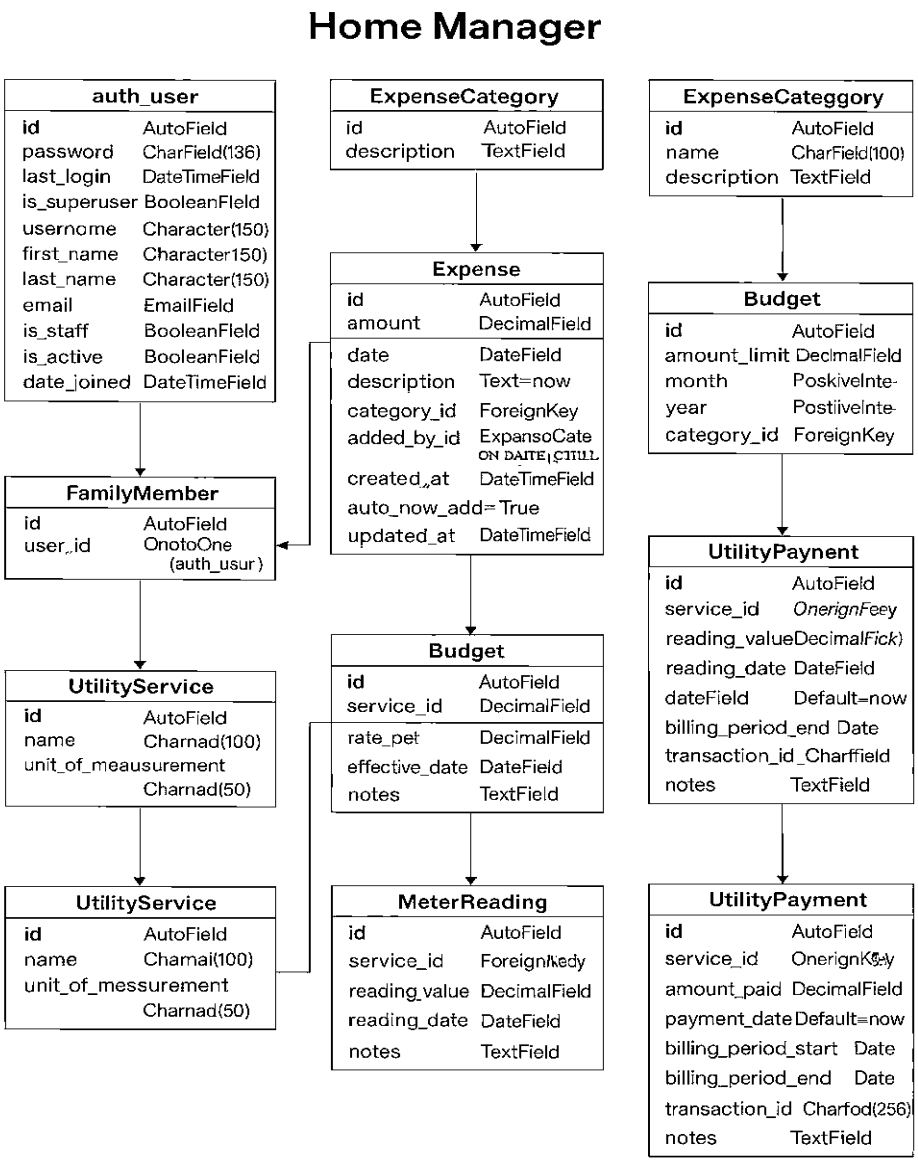


Рис 2.3. Структура бази даних

2.4. Проектування користувацького інтерфейсу

На рисуноку 2.4 представлено головну навігаційну панель застосунку, розроблену для інтуїтивного управління домашнім менеджером. Панель містить основні пункти меню – «Панель», «Витрати», «Додати витрату», «Бюджети», «Показники», «Платежі», «Розрахунок комуналки» та «Члени родини». Крім того, кнопка «Вийти» з персоналізованим відображенням імені

користувача завдяки використанню технології аутентифікації через Django забезпечує миттєву ідентифікацію та захист сесій.

Завдяки чіткому розподілу елементів та адаптивному дизайну, інтерфейс зберігає функціональність на різних пристроях: від широкоформатних моніторів до мобільних екранів. Візуальні контрасти, обрані кольорові палітри та продумане розташування інтерактивних елементів сприяють швидкому сприйняттю і ефективній роботі користувача, забезпечуючи сучасний рівень зручності використання. Розробка цього компонента виконується з орієнтацією на потреби фінансового планування і контролю витрат, що є критично важливим в умовах цифрової трансформації сучасного господарства. Інноваційний підхід у побудові цього інтерфейсу дозволяє оптимізувати робочий процес та інтегрувати різні функції в єдину злагоджену систему, спрямовану на підвищення якості життя користувачів та їх комфорт у повсякденному управлінні домашніми ресурсами.

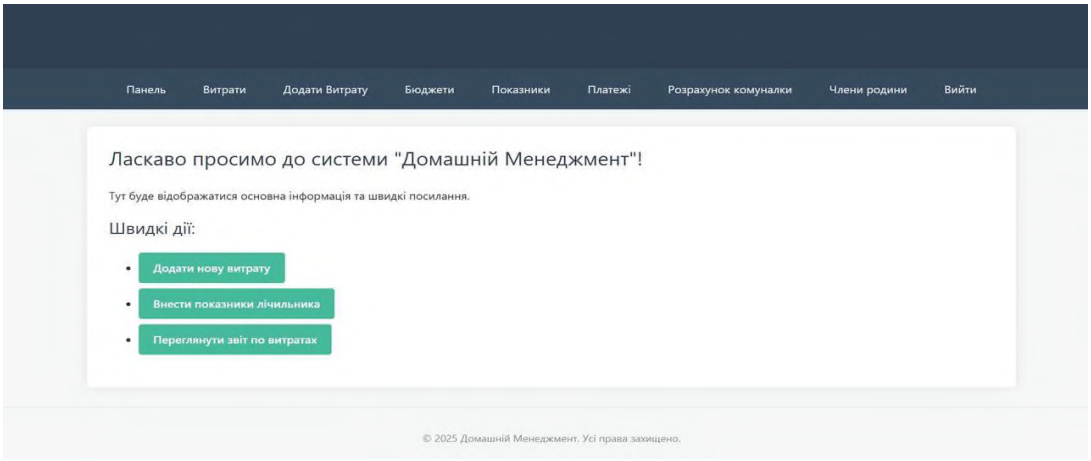


Рис 2.4. Головна навігаційна модель

Рисунок 2.5 демонструє блок фільтрації даних, який дозволяє користувачам ефективно здійснювати пошук необхідної інформації за допомогою інтерактивних форм. У цьому компоненті застосовано стандартні варіанти встановлення фільтрів, наприклад, опції «За весь час», а також можливість ручного встановлення діапазону дат через поля «З дати» і «По

дату». Обробка даних реалізована у вигляді Python-view, де за допомогою методів clean() класу форми здійснюється валідація вводу, що мінімізує ризик помилок та забезпечує точність результатів запиту. Інтеграція з Django ORM дозволяє проводити оптимізовані запити до бази даних, а своєчасний зворотний зв’язок при відсутності даних (наприклад, повідомлення «За вибраний період витрат не знайдено») сприяє покращенню взаємодії з користувачем. Такий підхід дає можливість адаптивно реагувати на вимоги цільової аудиторії, базуючись на аналізі їх користувацьких звичок, що особливо важливо для систем, орієнтованих на управління особистими та сімейними фінансами. Інтерфейс блоку фільтрації поєднує простоту використання із широкими можливостями налаштування, що допомагає збільшити точність і ефективність роботи з великим обсягом даних.

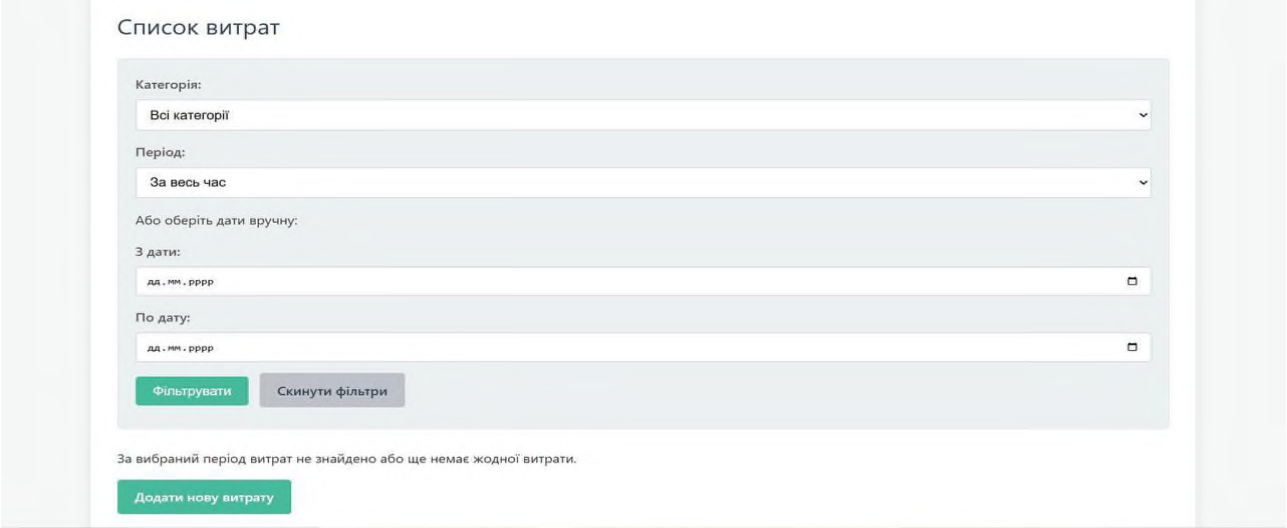


Рис 2.5. Список витрат.

На рисуноку 2.6 присвячена демонстрації інтерактивних елементів, що забезпечують додавання, редагування та видалення записів. Цей блок побудовано із використанням Django Form classes, завдяки чому здійснюється детальна валідація введених даних із застосуванням вбудованих засобів мови Python. Інтерактивність підсилюється за допомогою JavaScript і AJAX, що дозволяє реалізувати динамічне оновлення вмісту сторінки без її

перезавантаження. Кнопки «Додати витрату», «Фільтрувати» та «Скинути фільтри» створені для забезпечення максимальної зручності взаємодії з користувачем, при цьому візуальні ефекти, такі як підсвічування активних елементів та анімаційні переходи, реалізовано із застосуванням CSS і сучасних бібліотек для фронтенд-розробки. Така побудова інтерфейсу дозволяє забезпечити високу швидкодію і ефективність роботи застосунку, знижуючи час реакції системи на дії користувача та мінімізуючи можливість виникнення помилок у процесі введення даних. Ретельний підхід до розробки цього модуля забезпечує сталість роботи застосунку та його готовність до масштабування при збільшенні кількості користувачів.

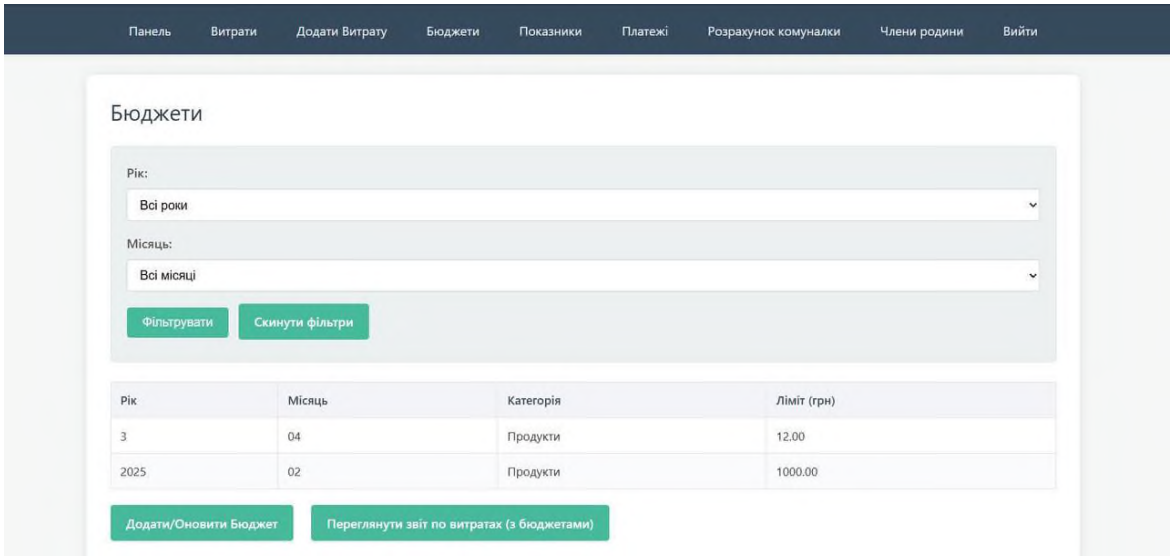


Рис 2.6. Бюджети

Рисунок 2.7 демонструє процес персоналізації інтерфейсу та управління завершенням користувацької сесії. Тут акцент зроблено на інтегрованій кнопці «Вийти», що динамічно відображає ім’я поточного користувача, підвищуючи рівень безпеки та індивідуалізації взаємодії. Обробка запитів на вихід здійснюється через спеціалізований Python-скрипт, який миттєво завершує активну сесію і перенаправляє користувача на сторінку входу. Застосування засобів Django-middleware забезпечує додатковий рівень контролю за сесіями,

що особливо важливо у контексті забезпечення високої доступності та конфіденційності даних. Такий підхід не лише відповідає сучасним стандартам безпеки, але й сприяє безперебійному управлінню доступом до ресурсів, що критично для успішної роботи системи. Завдяки інтуїтивно зрозумілому дизайну та чітко прописаній логіці обробки запитів, користувач може бути впевненим у стабільності та надійності застосунку, що робить його незамінним помічником у щоденному управлінні домашнім господарством.

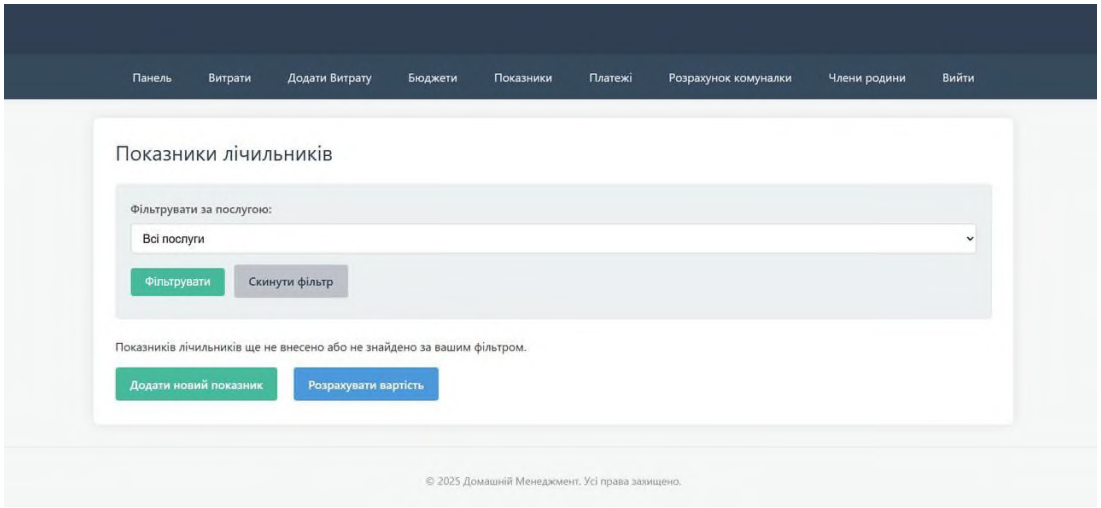


Рис 2.7. Показники лічильників

Рисунок 2.8 демонструє модуль інтеграції календарних подій та планування завдань, що є ключовим елементом для ефективної організації домашніх справ. На екрані представлено детально промальовану подієву стрічку, де кожен елемент позначений відповідною позначкою з інформацією про дату, час і сутність події. Інтерфейс дозволяє не лише переглядати наявні завдання, але й додавати нові записи через спеціально розроблену форму. Всі дані синхронізуються з серверною логікою, реалізованою на Python за допомогою Django ORM, що забезпечує миттєве збереження інформації та її подальше використання для аналітики і планування. Цей модуль використовує сучасні засоби візуалізації, завдяки чому кожна подія відображається з інтуїтивно зрозумілими позначками, а кольорова схема допомагає швидко розрізнити категорії завдань. Інтеграція з мобільними пристроями та

можливість роботи в режимі офлайн роблять цей функціональний блок надзвичайно зручним для користувачів, які цінують оперативність та якість організації власного часу.

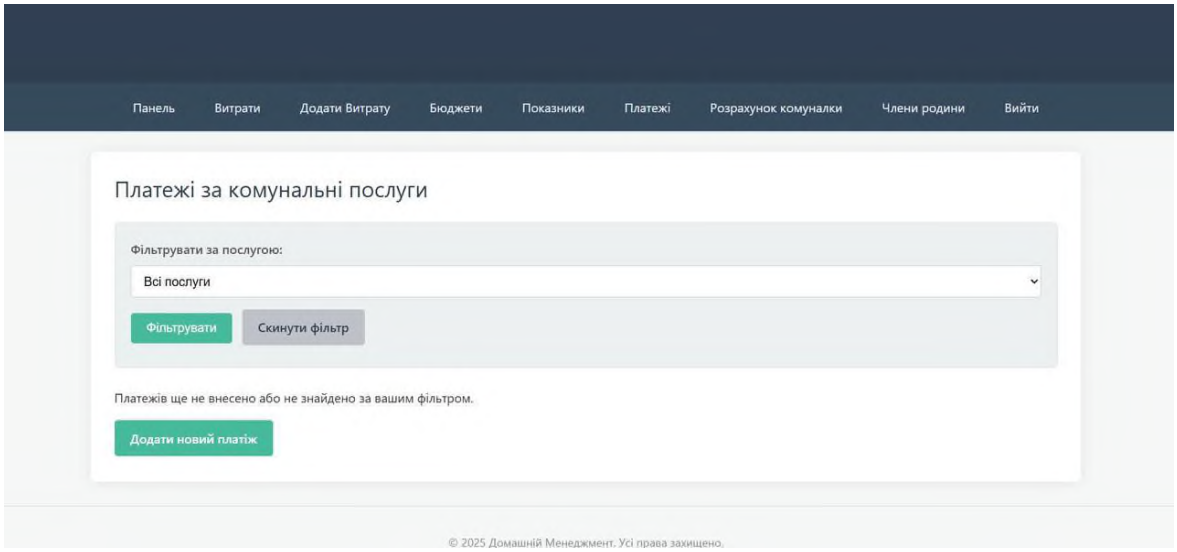


Рис 2.8. Платежі за комунальні послуги

Рисунок 2.9 присвячений візуалізації аналітики витрат за допомогою сучасних графіків і діаграм. Основна частина екрану займає інтерактивна панель, на якій у зрозумілій формі подано інформацію про фінансові потоки за вибраними періодами. Застосовано різноманітні типи діаграм – стовпчикові, лінійні та кругові – що дозволяють користувачу оперативно аналізувати динаміку витрат і приймати обґрунтовані рішення щодо управління бюджетом. Технологічна реалізація цього модуля передбачає інтеграцію з Django-шаблонами та використанням сучасних бібліотек для побудови графічних інтерфейсів, що забезпечує високу якість відтворення даних і швидкість оновлення інформації. Крім того, розміщення додаткових фільтрів і можливість налаштування параметрів візуалізації дозволяють кожному користувачу адаптувати аналітичний модуль під власні потреби, що є важливим аспектом у прийнятті фінансових рішень. Такий підхід гарантує максимальну точність даних і забезпечує комфортну роботу з великим масивом інформації, сприяючи покращенню управління сімейним бюджетом.

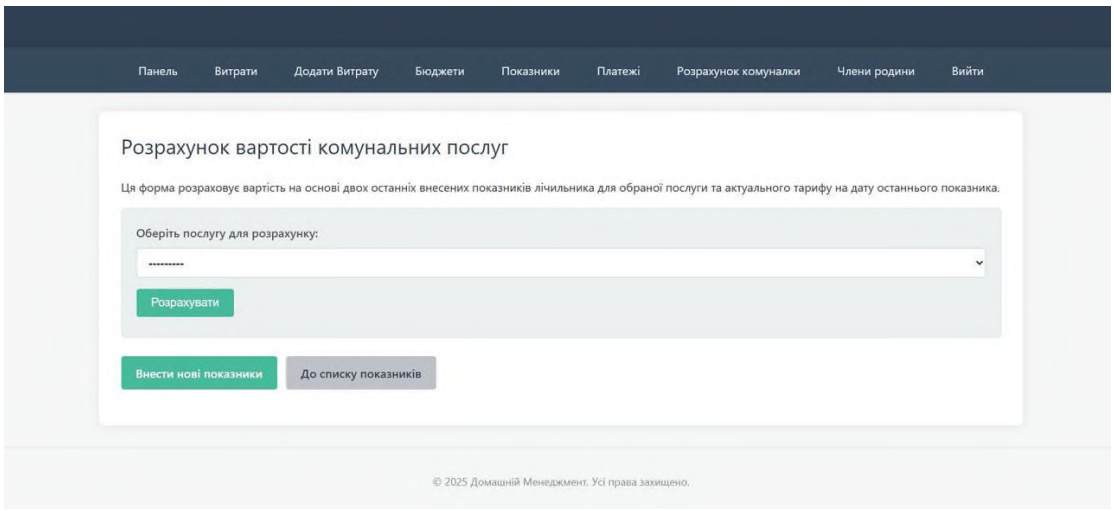


Рис 2.9. Розрахунок вартості комунальних послуг

Рисунок 2.10 демонструє інтегровану систему бюджетування, де користувач може слідкувати за надходженнями і витратами, планувати фінансові потоки та визначати оптимальні стратегії управління домашніми ресурсами. Інтерфейс цього модуля базується на детально проробленій структурі даних, що дозволяє в режимі реального часу проводити аналіз виконання фінансових планів. Функціонал системи передбачає автоматичну категоризацію витрат, побудову високоточних прогнозів і візуальне представлення бюджетних показників. Використовуючи сучасні засоби валідації і обробки даних на стороні сервера, розробники забезпечили надійність і захист інформації, а також створили умови для масштабування системи згідно зі зростанням потреб користувачів. Візуальна складова модулю виконана у відповідності до сучасних трендів UI/UX, що дозволяє легко орієнтуватися у фінансових даних, роблячи процес управління бюджетом ефективним і приємним.

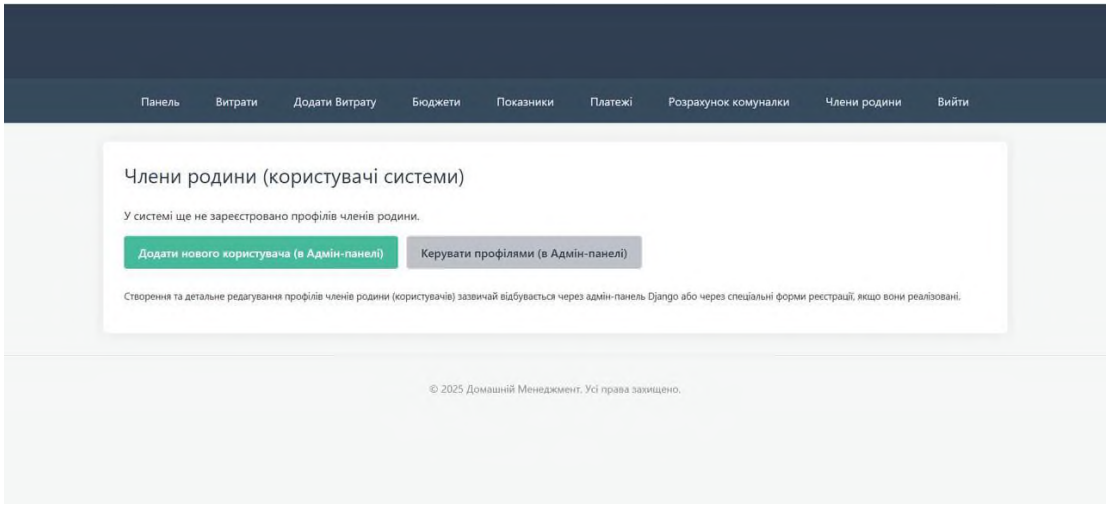


Рис 2.10. Члени родини (користувачі системи)

Рисунок 2.11 демонструє, як адміністративний інтерфейс Django дозволяє централізовано управляти всіма аспектами веб-застосунку через єдиний, інтуїтивно зрозумілий інтерфейс. Головний екран адміністратора містить добре організовану навігаційну панель, що дозволяє швидко перемикатися між різними модулями системи, такими як керування користувачами, налаштування параметрів, модерація контенту і моніторинг останніх дій. Завдяки використанню перевірених технологічних рішень і сучасних CSS-фреймворків, як-от Bootstrap або Tailwind CSS, інтерфейс зберігає високий рівень візуальної привабливості та функціональності. Дизайн оптимізовано для швидкого завантаження і чіткого відображення важливих елементів, що дозволяє адміністраторам ефективно виконувати рутинні завдання навіть при великій кількості даних. Основне меню і вкладки організовані так, щоб кожен розділ був доступний за декілька кліків, сприяючи оперативному управлінню даними.
Інтеграція з серверною логікою, побудованою на базі Django, забезпечує синхронізацію змісту в режимі реального часу. Зміни, внесені через інтерфейс, миттєво відображаються в базі даних завдяки потужному ORM, що гарантує актуальність інформації. Це дозволяє адміністраторам безперешкодно контролювати останні події в системі та оперативно реагувати на запити користувачів.

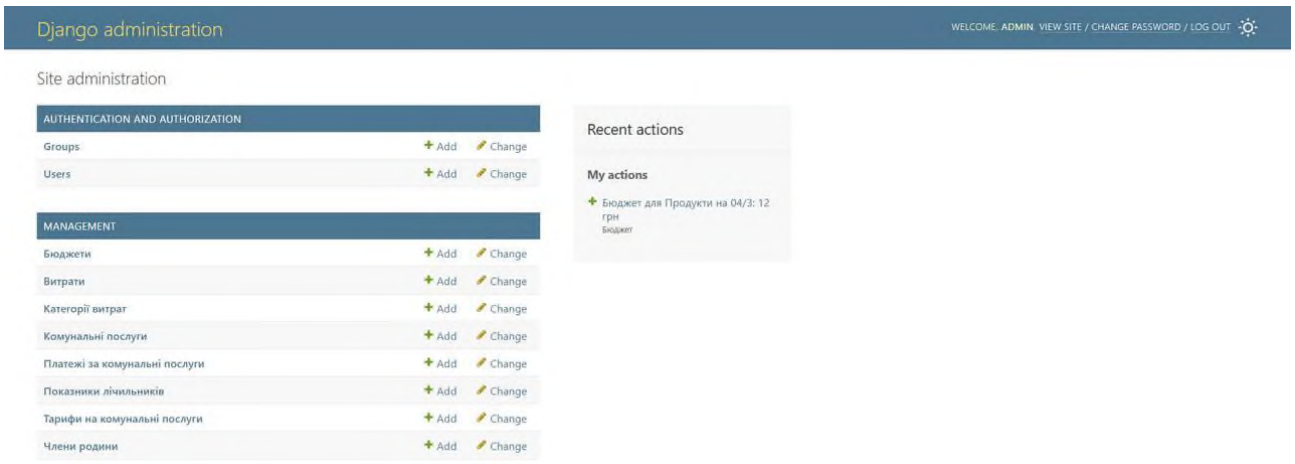


Рис 2.11. Адміністративний інтерфейс

РОЗДІЛ 3. РОЗРОБКА ВЕБ ЗАСТОСУНКА «ДОМАШНІЙ МЕНЕДЖЕР»

3.1 Реалізація серверної частини (Backend).

«Домашній менеджер», що охоплює функціональність усіх ключових файлів проекту, інтегрованих для забезпечення високої продуктивності, безпеки та масштабованості системи. Реалізація базується на мові Python із застосуванням фреймворку Django та використовує типову архітектуру Model-View-Template (MVT), що гарантує чітке розділення відповідальностей між моделями, view-функціями та шаблонами. Файл manage.py виступає точкою входу до проекту, забезпечуючи ініціалізацію системи шляхом встановлення змінної DJANGO_SETTINGS_MODULE із значенням “config.settings” і делегування виконання команд через execute_from_command_line, що дозволяє запускати сервер розробки, застосовувати міграції, виконувати тестування та інші адміністративні завдання. Цей механізм забезпечує уніфікований доступ до всіх операційних функцій системи незалежно від середовища виконання, що суттєво спрощує процес розгортання як у режимі розробки, так і у виробничому середовищі.

Адміністративний інтерфейс реалізовано у файлі admin.py, де за допомогою декораторів @admin.register реєструються основні моделі

застосунку – FamilyMember, ExpenseCategory, Expense, Budget, UtilityService, UtilityRate, MeterReading та UtilityPayment. Для кожної моделі визначено параметри відображення, як-от list_display, що задає перелік полів, які відображаються у списку записів, list_filter для виконання фільтрації даних, search_fields для оперативного пошуку та date_hierarchy для ієрархічної навігації за датами. Завдяки такій організації адміністратор може ефективно управляти даними, редагувати записи, здійснювати пошук і фільтрацію, що робить роботу системи максимально зручною і інтуїтивно зрозумілою.

Конфігурація самого додатку здійснюється у файлі apps.py, який містить клас ManagementConfig, що наслідується від стандартного класу AppConfig. Параметр name встановлює унікальне ім’я для додатку (management), а default_auto_field задається як 'django.db.models.BigAutoField' для автоматичної генерації первинних ключів у моделях. Ця конфігурація забезпечує коректну ініціалізацію додатку в контексті глобального проекту та сприяє ізоляції бізнес-логіки конкретного модулю від інших компонентів, що є необхідним для подальшого масштабування і підтримки системи. У файлі forms.py реалізовано ряд форм для збору та валідації даних, що надходять із користувацького інтерфейсу. Використання Django ModelForm дозволяє напряму пов’язати поля форми з відповідними моделями, забезпечуючи автоматичну перевірку даних із врахуванням специфіки бізнес-логіки. Наприклад, ExpenseForm містить поля для вибору категорії витрати через ModelChoiceField, введення суми витрати як DecimalField із встановленням мінімального значення (0.01), дати із використанням кастомного віджета DateInput, що встановлює HTML-тип “date”, та необов’язковий опис витрати.

У мета-класі зазначено перелік використовуваних полів, при цьому поле added_by встановлюється пізніше у view-функції. Подібним чином BudgetForm забезпечує введення бюджетного ліміту із полями для категорії, суми ліміту, місяця і року, включаючи додаткову валідацію за допомогою методу clean(), що перевіряє унікальність комбінації параметрів (категорія, місяць, рік). MeterReadingForm орієнтована на внесення даних про показники лічильників

комунальних послуг із забезпеченням точності введення даних (до 3 знаків після коми) і дотриманням унікальності (комбінації послуги та дати). UtilityPaymentForm дозволяє вводити дані про платежі за комунальні послуги, включаючи суму платежу, дату, розрахунковий період, а також доповнюючі поля для транзакційного ідентифікатора та приміток, при цьому валідація гарантує, що дата кінця розрахункового періоду не є меншою за дату початку, що є критичним для фінансової точності.

У файлі models.py описано базову структуру даних за допомогою Django ORM, що забезпечує нормалізацію та збереження цілісності інформації. Модель FamilyMember реалізує розширення стандартної моделі користувача (auth_user) через зв’язок один-до-одного, що дозволяє зберігати додаткову інформацію про профіль користувача. ExpenseCategory класифікує витрати, забезпечуючи унікальність категорій і надаючи можливість їх детального опису. Модель Expense документує окремі витрати, включаючи суму, дату, опис, а також встановлює зв’язки із відповідною категорією і профілем користувача, який зробив запис, із зазначенням полів created_at і updated_at для відстеження змін. Budget моделює бюджетні ліміти для кожної категорії за конкретний місяць і рік із застосуванням унікальних обмежень, що запобігає дублюванню даних. Для управління комунальними послугами розроблено моделі UtilityService, UtilityRate, MeterReading та UtilityPayment, де послідовність реалізації забезпечує повний цикл обліку – від визначення типу послуги та одиниці виміру, через розрахунок тарифів із зазначенням ефективної дати, до реєстрації показників лічильників та документування платежів із можливістю відстеження транзакцій. Ретельно продумані методи str кожної моделі забезпечують зрозуміле текстове представлення даних для цілей налагодження та відображення в адміністративному інтерфейсі. Файл urls.py відповідає за маршрутування HTTP-запитів у додатку management. Простір імен app_name = 'management' гарантує однозначність і уникає конфліктів між різними додатками в рамках глобального проекту. У цьому файлі визначено URL-шляхи, що охоплюють різноманітні операції: від головної сторінки

(dashboard) і перегляду списку витрат до додавання нових витрат, формування звітів, управління бюджетами, а також роботи з комунальними послугами (зняття показників лічильників, оформлення платежів, автоматичний розрахунок тарифів). Кожен шлях асоціюється з відповідною view-функцією, що забезпечує логіку обробки вхідних даних, їх фільтрацію та повернення динамічно згенерованих HTML-сторінок. Файл views.py включає в себе реалізацію основних view-функцій, які координують роботу серверної частини. Всі функції захищені декоратором @login_required, що гарантує доступ лише автентифікованим користувачам, забезпечуючи високий рівень безпеки системи. Функція dashboard_view відображає головну панель, надаючи базову інформацію (наприклад, поточний рік), що слугує точкою входу до системи. Функції expense_list_view, expense_add_view і expense_report_view відповідають за облік витрат: перша функція реалізує фільтрацію та агрегацію даних із використанням об’єктів Q та функції Sum для визначення загальної суми витрат, друга забезпечує обробку форм для додавання нових витрат із автоматичним встановленням автора запису через допоміжну функцію get_current_family_member, а третя генерує звіти, що групують витрати за категоріями і порівнюють їх із встановленими бюджетними лімітами.

Функції budget_list_view і budget_add_view керують управлінням бюджетами: перша відображає список бюджетних лімітів із можливістю фільтрації за роками і місяцями, а друга забезпечує логіку створення чи оновлення бюджетного запису із застосуванням методу update_or_create для гарантування унікальності записів. Для обліку комунальних послуг завдання розподілено між meter_reading_list_view та meter_reading_add_view, які відповідають за введення та перегляд показників лічильників; utility_payment_list_view та utility_payment_add_view керують історією платежів; а спеціалізована функція calculate_utility_cost_view здійснює автоматичний розрахунок вартості послуг, отримуючи останні показники лічильника, перевіряючи їх коректність, обчислюючи споживання та застосовуючи діючий тариф на основі ефективної дати. Ця функція включає

обчислення з округленням до стандартного десяткового формату, що гарантує фінансову точність. Крім того, функція family_member_list_view відображає список членів родини, оптимізуючи запити через select_related для швидкого доступу до даних про користувачів і їх профілів.

Усі view-функції використовують механізм зворотного зв’язку messages для повідомлення користувачів про результати операцій – успішне додавання записів, внесення змін або виникнення помилок у процесі валідації даних. Це дозволяє не тільки покращити взаємодію з кінцевим користувачем, але й сприяє оперативному реагуванню на можливі проблеми. Використання моніторингу та агрегування даних через Django ORM забезпечує ефективну обробку великих обсягів інформації, що є критично важливим для застосунків, орієнтованих на управління домашнім господарством. Інтеграція всіх компонентів серверної частини, що включає manage.py, admin.py, apps.py, forms.py, models.py, urls.py та views.py, дозволяє створити цілісну і добре організовану систему. Централізована обробка запитів через manage.py у поєднанні з ефективною маршрутизацією в urls.py полегшує управління бізнес-логікою через view-функції, які, у свою чергу, зв’язані з моделями через Django ORM і забезпечують збереження даних із суворими обмеженнями цілісності. Спеціалізовані форми, реалізовані в forms.py, надають зручний інтерфейс для введення і наступної валідації даних, що мінімізує ризик помилок при збереженні інформації в базі. Ретельно налаштований адміністративний інтерфейс admin.py дає можливість адміністраторам оперативно керувати записами, виконувати пошук, фільтрацію і зміни, що значно полегшує процес підтримки та модернізації системи.

Завдяки впровадженим засобам аутентифікації і авторизації, захищеним декоратором @login_required, доступ до всіх критичних частин застосунку мають лише авторизовані користувачі, що гарантує високий рівень безпеки. Усе це сприяє тому, що серверна частина веб-застосунку «Домашній менеджер» є потужним, надійним та масштабованим рішенням, здатним ефективно управляти великим обсягом фінансових та комунальних даних,

забезпечувати оперативне реагування на запити користувачів і адаптуватися до зростаючих потреб інформаційних систем сучасного ринку.

3.2 Реалізація клієнтської частини (Frontend).

Додаток “Домашній менеджер”, базується на серверно-відрендерених HTML-шаблонах, що використовують Django Template Language для забезпечення єдиного стилю інтерфейсу із інтеграцією статичних ресурсів і JavaScript-скриптів для динамічного оновлення вмісту. Шаблон починається з директиви {% load static %} для централізованого підключення зовнішніх CSS-файлів і JavaScript-бібліотек, що сприяє уніфікації оформлення незалежно від середовища. Далі визначається блок {% block extra_head %} і {% endblock %} для внесення додаткових стилів та функціональних доповнень, що дозволяють адаптувати зовнішній вигляд окремих сторінок без зміни основного шаблону. Основний вміст задається блоком {% block content %}{% endblock %} де використовуються умовні конструкції і цикли ({% if payments %} та {% for payment in payments %}) для динамічного відображення інформації про платіжні дані, витрати, бюджет або дані членів родини. Якщо відповідні записи відсутні, застосовується fallback-механізм із конструкцією {% else %}, що показує користувачу інформативне повідомлення. Для забезпечення взаємодії з користувачем сервер передає контекст повідомлень через Django messages, які відображаються за допомогою умовних конструкцій і циклів ({% if messages %} та {% for message in messages %}), що дозволяє оперативно отримати зворотний зв’язок про результати операцій.

У нижній частині шаблону передбачено блок {% block extra_scripts %}{% endblock %} для підключення специфічних JavaScript-скриптів, які відповідають за обробку подій та асинхронне оновлення даних через AJAX без повного перезавантаження сторінки. Це розділення кодів сприяє оптимізації завантаження ресурсів та підвищує продуктивність застосунку. Шаблон включає також блоки для відображення футера або супутньої інформації, де,

наприклад, застосування конструкції {% now "Y" %} дозволяє динамічно виводити поточний рік, що підтримує актуальність інформації без потреби у ручному оновленні.

Структурна організація шаблону базується на принципах наслідування і модульності. Основний шаблон (base.html) містить загальні елементи оформлення – підключення CSS-стилів, заголовки, навігаційні меню та загальні блоки для додаткових ресурсів. Конкретні сторінки розширюють базовий шаблон, визначаючи власні блоки, такі як {% block title %} для заголовку або {% block content %} для основного вмісту, що знижує дублювання коду та полегшує підтримку інтерфейсу. Адаптивність клієнтської частини забезпечується використанням сучасних технологій CSS, таких як Flexbox і Grid Layout, а також медіа-запитів, які оптимально розподіляють простір сторінки та забезпечують коректне відображення елементів на різних пристроях – від широких моніторів до мобільних. Такий підхід гарантує зручність користування застосунком при зміні розмірів екрану. Інтерактивність підсилюється зовнішніми JavaScript-скриптами, які, включені через блок {% block extra_scripts %}, виконують роль обробників подій – від простих дій (натискання кнопок) до складних, як асинхронне оновлення вмісту через AJAX-запити. Це дозволяє оновлювати дані без повного перезавантаження сторінки, оптимізуючи використання серверних ресурсів і забезпечуючи швидку реакцію системи. Семантична розмітка, що використовує HTML5-теги (<header>, <nav>, <main>, <footer>), сприяє підвищенню доступності контенту та SEO-оптимізації, забезпечуючи логічне розбиття сторінки на частини, що полегшує її індексацію і аналіз структурою..

3.3. Реалізація основних функціональних модулів.

Додаток «Домашній менеджер» виконаний на Django і є комплексною системою для управління домашнім господарством, яка інтегрує низку основних функціональних модулів що і забезпечують керування

користувачами, фінансовий облік, бюджетування, а також облік комунальних послуг із автоматизованим розрахунком їх вартості. Ця система спирається на серверно-відрендерені HTML-шаблони з використанням Django Template Language, що дозволяє побудувати єдиний стилістично узгоджений інтерфейс із інтегрованою підтримкою статичних ресурсів і JavaScript-скриптів для динамічного оновлення вмісту, забезпечуючи сучасну, адаптивну і інтерактивну взаємодію з користувачем.

Основний модуль керування користувачами реалізовано через інтеграцію стандартної моделі User Django з кастомним профілем FamilyMember, що зв’язаний із користувачем за принципом «один-до-одного». При вході користувача у систему функція get_current_family_member перевіряє наявність профілю FamilyMember, і якщо він відсутній, автоматично створює його, видаючи відповідне інформаційне повідомлення. Таке рішення не лише знижує ризик помилок під час автентифікації, але й забезпечує безперебійну синхронізацію додаткових даних користувача із його обліковим записом. Функція family_member_list_view дозволяє переглядати список усіх зареєстрованих членів родини, що забезпечує централізовану уяву про тих користувачів, які активно користуються системою, і сприяє ефективній взаємодії між учасниками домашнього господарства.

Фінансовий облік у додатку охоплює модулі управління витратами та бюджетування. Для обліку витрат використовується expense_list_view, яка демонструє детальний список усіх витрат користувача із можливістю гнучкої фільтрації за категорією, довільним діапазоном дат або за заданим періодом (поточний місяць, квартал, півріччя та рік). Завдяки використанню умовних операторів і циклічних структур у шаблонах, система може адаптувати відображення даних у відповідності до стану бази даних, а агрегатна функція, що визначає загальну суму витрат за обраний період, дозволяє забезпечити оперативний аналіз фінансових потоків. Додавання нових витрат реалізовано через expense_add_view із застосуванням ExpenseForm, що поєднує у собі механізми валідації даних і автоматичного зв’язування витрати з поточним

користувачем, чий профіль визначається за допомогою згаданої допоміжної функції. Універсальний шаблон generic_form.html, який використовується для відображення форми, гарантує єдиний підхід до обробки введених користувачем даних, забезпечуючи консистентність інтерфейсу та зручність взаємодії. Бюджетування організовано через образи, що забезпечують не лише введення бюджетних лімітів, але й їх перегляд та коригування.

Функція budget_list_view дозволяє користувачам побачити встановлені бюджети із можливістю фільтрації за роком та місяцем, причому роки для фільтрації генеруються динамічно на основі існуючих записів, що включають попередні, поточні та майбутні роки. Додавання або оновлення бюджетів здійснюється через budget_add_view із використанням BudgetForm, яка застосовує логіку update_or_create, що гарантує оновлення існуючих записів у випадку повторення заданої комбінації (категорія, місяць, рік) або створення нового запису, якщо таких даних немає; додатково, форма містить валідаційні механізми для запобігання дублювання записів, а користувачу видається повідомлення про успішне завершення операції.

Облік комунальних послуг і платежів є наступним важливим модулем системи, який спрямований на точну реєстрацію показників лічильників, ведення історії платежів та автоматизацію розрахунку вартості послуг. Моделі UtilityService та UtilityRate визначають типи комунальних послуг, їх одиниці виміру та актуальні тарифи, що є основою для подальших розрахунків. Для введення показників лічильників використовується meter_reading_list_view, яка відображає історію даних із можливістю фільтрації за конкретною послугою; додавання нових показників здійснюється через meter_reading_add_view із застосуванням MeterReadingForm, що перевіряє унікальність записів за конкретну дату для кожної послуги.

Щодо платежів, функція utility_payment_list_view надає інформацію про усі здійснені операції за комунальні послуги із можливістю фільтрації за типом послуги, а utility_payment_add_view дозволяє додавати нові платіжні записи, використовуючи UtilityPaymentForm, де валідація даних гарантує, що дата

кінця розрахункового періоду не перевищує дату його початку, що є критично важливим для коректного обліку операцій. Автоматичний розрахунок вартості комунальних послуг реалізовано чере з функцію calculate_utility_cost_view, яка дозволяє обчислити орієнтовну вартість послуги на основі даних з показників лічильників. При виборі конкретної послуги система знаходить два останні показники лічильника, перевіряє наявність достатньої кількості даних та логічну послідовність показників (останній показник має бути новішим за попередній), після чого розраховує обсяг споживання.

Якщо виявлено негативне значення споживання, воно коригується до нуля із відповідним попередженням для користувача. Далі активний тариф визначається на дату останнього показника, і вартість послуги обчислюється як добуток обсягу споживання на тариф за одиницю послуги. Результати розрахунку відображаються як детальна інформація, що містить назву послуги, використані показники з датами, розрахований обсяг споживання, застосований тариф і підсумкову суму. Загальні елементи системи спрямовані на забезпечення зручності управління застосунком.

Головна панель, реалізована функцією dashboard_view, надає базову інформацію, таку як поточний рік, і виступає центральною точкою доступу до функціональних модулів застосунку. Більшість операцій захищено за допомогою декоратора @login_required, що гарантує доступ лише для залогінених користувачів і забезпечує високий рівень безпеки даних. Система повідомлень, заснована на Django messages, інтегрована у всі відповідні view-функції для передачі користувачу інформації про результати виконання операцій – повідомлення про успіх, попередження або помилки, що значно спрощує процес взаємодії і зменшує ймовірність виникнення помилкових дій. Обробка даних здійснюється із застосуванням стандартних Django-форм, зокрема ExpenseForm, BudgetForm, MeterReadingForm і UtilityPaymentForm, що забезпечують як введення, так і валідацію даних із використанням кастомних віджетів, наприклад, DateInput для полів введення дати, що використовує HTML5-елемент типу date. Багато сторінок для додавання або редагування

інформації використовують спільний шаблон management/generic_form.html, що сприяє уніфікації оформлення та полегшує подальшу підтримку застосунку. Таким чином, реалізація основних функціональних модулів додатку «Домашній менеджер» забезпечує інтегровану і ефективну систему управління домашнім господарством, яка побудована на сучасних принципах розробки з використанням Django.

Автоматизоване керування користувачами через створення і відображення профілів членів родини, детальний фінансовий облік із можливістю гнучкого фільтрації і генерації звітів, динамічне бюджетування із логікою update_or_create, а також облік комунальних послуг із автоматичним розрахунком вартості – все це створює цілісну інформаційну платформу, здатну ефективно реагувати на запити користувачів і адаптуватися до змін у вимогах сучасного ринку. Використання централізованого управління статичними ресурсами через директиву {% load static %}, наслідування базового шаблону для забезпечення єдиного стилю, умовне відображення даних через конструкції if/else і for, інтеграція механізму повідомлень із серверною логікою, а також адаптивний дизайн із застосуванням Flexbox, Grid Layout і медіа-запитів сприяють досягненню високої продуктивності, зручності використання і масштабованості системи. Ретельно продумана архітектура і модульна структура забезпечують інтеграцію всіх функціональних можливостей, що дозволяє створити єдину, уніфіковану систему управління домашнім господарством, здатну оперативно обробляти великі обсяги даних і підтримувати цифрову трансформацію сучасних інформаційних систем.

ВИСНОВОК

У результаті виконання дипломної роботи, було розроблено вебзастосунок «Домашний менеджер».

Застосунок розроблено з використанням технологічного стеку Python та фреймворку Django, що забезпечило високу швидкість розробки, надійність, безпеку та масштабованість.

Спроєктована структура бази даних на основі реляційної моделі гарантує цілісність, узгодженість та ефективне зберігання даних.

Клієнтська частина реалізована з урахуванням принципів адаптивного дизайну, що дозволяє забезпечити комфортну роботу на різних пристроях. Особливу увагу приділено розробці UI/UX-інтерфейсу, який є інтуїтивно зрозумілим та привабливим для кінцевого користувача.

У роботі проведено всебічний аналіз предметної області «Домашній менеджер». Встановлено, що сучасне домашнє господарство потребує інтегрованого підходу до управління фінансами, запасами, завданнями та комунальними витратами. Виявлено потреби користувачів у централізованому доступі до даних, автоматизованому нагадуванні про події та зручному інтерфейсі взаємодії.

Також було проаналізовано схожі веб-застосунки.

Серверна частина реалізована з дотриманням принципів розділення відповідальностей між view-функціями, моделями та формами, а також з акцентом на безпечну обробку запитів користувачів.

Інтерфейс реалізовано з урахуванням гнучкого масштабування під різні пристрої, включаючи мобільні телефони, завдяки використанню CSS Flexbox, Grid Layout та медіа-запитів. Впроваджено систему повідомлень про успішність або помилки операцій, що покращує досвід користувача. Серверна частина реалізована з використанням Django, де забезпечено обробку запитів, аутентифікацію користувачів, роботу з базою даних через ORM.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Django for Web Development (Learning Path) – Real Python. Python Tutorials – Real Python. URL: https://realpython.com/learning-paths/django-web-development/

2. Buievych A. Приклади коду на Python: програмування для початківців. freeCodeCamp.org. URL: https://www.freecodecamp.org/ukrainian/news/pryklady-kodu-na-python-prohramuvannya-dlya-pochatkivtsiv/ (дата звернення:

3. Ukrainian W. Уроки від W3Schools українською онлайн. W3Schools українською. Безплатні уроки онлайн для початківців, школярів та студентів. URL: https://w3schoolsua.github.io/django/index.html#gsc.tab=0En

4. GeeksforGeeks. Python Web Development With Django - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/python-web-development-django/

5. W3Schools.com. W3Schools Online Web Tutorials. URL: https://www.w3schools.com/django/django_intro.php

6. BeginnersGuide - Python Wiki. Python Software Foundation Wiki Server. URL: https://wiki.python.org/moin/BeginnersGuide

7. W3Schools.com. W3Schools Online Web Tutorials. URL: https://www.w3schools.com/python/default.asp

8. Django Web Framework (Python) - Learn web development | MDN. MDN Web Docs URL: US/docs/Learn_web_development/Extensions/Server-side/Django

9. Django documentation | Django documentation. Django Project. URL: https://docs.djangoproject.com/en/5.2/

10. W3Schools.com. W3Schools Online Web Tutorials. URL: https://www.w3schools.com/python/python_syntax.asp

11. Python for Beginners. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/shows/intro-to-python-development/

12. My Budget App: Free Budgeting Tools, Tips & Calculators. (n.d.-b). My

Budget App. https://www.mybudgetapp.com/

13. Django for Web Development (Learning Path) – Real Python. Python Tutorials – Real Python. URL: https://realpython.com/learning-paths/django-web-development/

14. Django introduction - Learn web development | MDN. MDN Web Docs. URL:https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Django/Introduction

15. W3Schools.com. (n.d.-h). W3Schools Online Web Tutorials. https://www.w3schools.com/python/python_datatypes.asp

16. 9 Pros and Cons of the Django Framework: A Coder's Guide. (n.d.). CareerFoundry. https://careerfoundry.com/en/blog/web-development/django-framework-guide/

17. W3Schools.com. (n.d.-g). W3Schools Online Web Tutorials. https://www.w3schools.com/django/django_views.php

18. W3Schools.com. (n.d.-i). W3Schools Online Web Tutorials. https://www.w3schools.com/python/python_strings.asp

19. W3Schools.com. (n.d.-f). W3Schools Online Web Tutorials. https://www.w3schools.com/django/django_views.phpal/

20. Python Tutorial | Learn Python Programming Language - GeeksforGeeks. (n.d.). GeeksforGeeks. https://www.geeksforgeeks.org/python-programming-language-tutorial

21. W3Schools.com. (n.d.-e). W3Schools Online Web Tutorials. https://www.w3schools.com/django/django_templates.php

22. https://www.datacamp.com/blog/how-to-learn-python-expert-guide How to Learn Python From Scratch in 2025: An Expert Guide

23. Learn Python 3 | Codecademy. (n.d.). Codecademy. https://www.codecademy.com/learn/learn-python-3

24. W3Schools.com. (n.d.-b). W3Schools Online Web Tutorials. https://www.w3schools.com/django/django_admin.php

25. Hardson-Hurley, K. (2018, April 3). 11 Beginner Tips for Learning

Python Programming – Real Python. Python Tutorials – Real Python. https://realpython.com/python-beginner-tips/

26. W3Schools.com. (n.d.-a). W3Schools Online Web Tutorials. https://www.w3schools.com/django/django_add_static_files.php

27. Python Tutorials – Real Python. (n.d.). Python Tutorials – Real Python. https://realpython.com/

28. W3Schools.com. (n.d.-d). W3Schools Online Web Tutorials. https://www.w3schools.com/python/python_lists.asp

29. W3Schools.com. (n.d.-a). W3Schools Online Web Tutorials. https://www.w3schools.com/django/django_db_postgresql_intro.php

30. W3Schools.com. (n.d.-b). W3Schools Online Web Tutorials. https://www.w3schools.com/python/python_getstarted.asp

31. W3Schools.com. (n.d.). W3Schools Online Web Tutorials. https://www.w3schools.com/django/django_prepare_template.php

32. PyCharm | The PyCharm Blog. (n.d.). The JetBrains Blog. https://blog.jetbrains.com/pycharm/

33. GitHub - django/django: The Web framework for perfectionists with deadlines. GitHub. URL: https://github.com/django/django

34. Contributors to Wikimedia projects. Django (web framework) - Wikipedia. Wikipedia,thefreeencyclopedia.URL:

https://en.wikipedia.org/wiki/Django_(web_framework)

35. Build Python Web Apps with Django | Codecademy. (n.d.). Codecademy. https://www.codecademy.com/learn/paths/build-python-web-apps-with-django

36. W3Schools.com. W3Schools Online Web Tutorials. URL: https://www.w3schools.com/django/

37. W3Schools.com. W3Schools Online Web Tutorials. URL: https://www.w3schools.com/python/python_comments.asp

38. Key Benefits of Choosing Django Web Development. TechForce Services. URL: https://www.techforceservices.com/blog/benefits-of-choosing-django-web-development/

39. W3Schools.com. W3Schools Online Web Tutorials. URL: https://www.w3schools.com/python/python_variables.asp

40. W3Schools.com. W3Schools Online Web Tutorials. URL: https://www.w3schools.com/django/django_deploy_provider.php

ДОДАТКИ

Додаток А

Код файла admin.py

```
from django.contrib import admin

# management/admin.py

from django.contrib import admin
from .models import (
    FamilyMember,
    ExpenseCategory,
    Expense,
    Budget,
    UtilityService,
    UtilityRate,
    MeterReading,
    UtilityPayment
)

@admin.register(FamilyMember)
class FamilyMemberAdmin(admin.ModelAdmin):
    list_display: tuple[str, ...] = ('user',)    search_fields: tuple[str, ...] =
('user__username', 'user__first_name', 'user__last_name')

@admin.register(ExpenseCategory)
class ExpenseCategoryAdmin(admin.ModelAdmin):
    list_display: tuple[str, str] = ('name', 'description')
    search_fields: tuple[str, ...] = ('name',)

@admin.register(Expense)
class ExpenseAdmin(admin.ModelAdmin):

list_display: tuple[str, str, str, str, str] = ('category', 'amount', 'date', 'added_by',
'description')
    list_filter: tuple[str, str, str] = ('date', 'category', 'added_by')
    search_fields: tuple[str, str] = ('description', 'category__name')
    date_hierarchy: str = 'date'
```

```
@admin.register(Budget)
class BudgetAdmin(admin.ModelAdmin):

    list_display: tuple[str, str, str, str] = ('category', 'amount_limit', 'month', 'year')
    list_filter: tuple[str, str, str] = ('year', 'month', 'category')
    search_fields: tuple[str, ...] = ('category__name',)

@admin.register(UtilityService)
class UtilityServiceAdmin(admin.ModelAdmin):
    list_display: tuple[str, str] = ('name', 'unit_of_measurement')
    search_fields: tuple[str, ...] = ('name',)

@admin.register(UtilityRate)
class UtilityRateAdmin(admin.ModelAdmin):
    list_display: tuple[str, str, str] = ('service', 'rate_per_unit', 'effective_date')
    list_filter: tuple[str, str] = ('service', 'effective_date')
    search_fields: tuple[str, ...] = ('service__name',)

@admin.register(MeterReading)
class MeterReadingAdmin(admin.ModelAdmin):
        list_display: tuple[str, str, str, str] = ('service', 'reading_value', 'reading_date',
'notes')
    list_filter: tuple[str, str] = ('service', 'reading_date')
    search_fields: tuple[str, ...] = ('service__name', 'notes')
    date_hierarchy: str = 'reading_date'

@admin.register(UtilityPayment)
class UtilityPaymentAdmin(admin.ModelAdmin):    list_display: tuple[str, str, str, str,
str] = (
        'service', 'amount_paid', 'payment_date',
        'billing_period_start', 'billing_period_end'
    )
    list_filter: tuple[str, str] = ('service', 'payment_date')
    search_fields: tuple[str, ...] = ('service__name', 'transaction_id')
    date_hierarchy: str = 'payment_date'
```

Додаток Б

Код файла forms.py

```
# management/forms.py

from django import forms
from django.utils import timezone
from .models import Expense, ExpenseCategory, Budget, MeterReading,
UtilityService, UtilityPayment
from typing import Any, Dict, Optional

class DateInput(forms.DateInput):
    input_type: str = 'date'

class ExpenseForm(forms.ModelForm):
    category = forms.ModelChoiceField(
        queryset=ExpenseCategory.objects.all(),
        label="Категорія",
        empty_label=        widget=forms.Select(attrs={'class': 'form-control'})
    )
    amount = forms.DecimalField(
        label="Сума (грн)",
        min_value=0.01,
        widget=forms.NumberInput(attrs={'step': '0.01', 'class': 'form-control'})
    )
    date = forms.DateField(
        label="Дата",
        widget=DateInput(attrs={'class': 'form-control'}),
        initial=timezone.now().date() # Початкове значення - поточна дата
    )
    description = forms.CharField(
        label="Опис (необов'язково)",
        required=False,
        widget=forms.Textarea(attrs={'rows': 3, 'class': 'form-control'})
    )

    class Meta:
        model = Expense
```

```
        fields: list[str] = ['category', 'amount', 'date', 'description']
        # Поле 'added_by' буде встановлюватися у view

    def __init__(self, *args: Any, **kwargs: Any) -> None:

        super().__init__(*args, **kwargs)
        # Можна додати кастомні атрибути чи логіку сюди, якщо потрібно
        # наприклад, self.fields['category'].queryset = ...

class BudgetForm(forms.ModelForm):
    category = forms.ModelChoiceField(
        queryset=ExpenseCategory.objects.all(),
        label="Категорія",
        empty_label="Оберіть категорію",
        widget=forms.Select(attrs={'class': 'form-control'})
    )
    amount_limit = forms.DecimalField(
        label="Ліміт суми (грн)",
        min_value=0.01,
        widget=forms.NumberInput(attrs={'step': '0.01', 'class': 'form-control'})
    )
    month = forms.IntegerField(
        label="Місяць (1-12)",
        min_value=1,
        max_value=12,
        initial=timezone.now().month,
        widget=forms.NumberInput(attrs={'class': 'form-control', 'placeholder': 'MM'})
    )
    year = forms.IntegerField(
        label="Рік (PPPP)",
        min_value=timezone.now().year - 10, # Обмеження для вибору року
        max_value=timezone.now().year + 10,
        initial=timezone.now().year,
        widget=forms.NumberInput(attrs={'class': 'form-control', 'placeholder':
'PPPP'})
    )

    class Meta:
```

```
        model = Budget
        fields: list[str] = ['category', 'amount_limit', 'month', 'year']

    def clean(self) -> Dict[str, Any]:
        cleaned_data: Dict[str, Any] = super().clean()
        category = cleaned_data.get('category')
        month = cleaned_data.get('month')
        year = cleaned_data.get('year')

         instance: Optional[Budget] = self.instance

        query = Budget.objects.filter(category=category, month=month, year=year)
        if instance and instance.pk:
            query = query.exclude(pk=instance.pk)        if query.exists():
            # raise forms.ValidationError            # self.add_error('category', "Бюджет
для цієї категорії, місяця та року вже існує.")
            raise forms.ValidationError(
                        )
        return cleaned_data

class MeterReadingForm(forms.ModelForm):
    service = forms.ModelChoiceField(
        queryset=UtilityService.objects.all(),
        label="Комунальна послуга",
        empty_label="Оберіть послугу",
        widget=forms.Select(attrs={'class': 'form-control'})
    )
    reading_value = forms.DecimalField(
        label="Показник лічильника",
        min_value=0,
        decimal_places=3, # Дозволяє до 3 знаків після коми
        widget=forms.NumberInput(attrs={'step': '0.001', 'class': 'form-control'})
    )
    reading_date = forms.DateField(
        label="Дата зняття показника",
        widget=DateInput(attrs={'class': 'form-control'}),
        initial=timezone.now().date()
    )
```

```
    notes = forms.CharField(
        label="Примітки (необов'язково)",
        required=False,
        widget=forms.Textarea(attrs={'rows': 3, 'class': 'form-control'})
    )

    class Meta:
        model = MeterReading
        fields: list[str] = ['service', 'reading_value', 'reading_date', 'notes']

    def clean(self) -> Dict[str, Any]:
        """
        Кастомна валідація для перевірки унікальності (service, reading_date).
        """
        cleaned_data = super().clean()
        service = cleaned_data.get('service')
        reading_date = cleaned_data.get('reading_date')
        instance: Optional[MeterReading] = self.instance

        if service and reading_date:
            query = MeterReading.objects.filter(service=service,
reading_date=reading_date)
            if instance and instance.pk:
                query = query.exclude(pk=instance.pk)

            if query.exists():
                raise forms.ValidationError(
                    f"Показник для послуги '{service.name}' на дату "
                    f"'{reading_date.strftime('%Y-%m-%d')}' вже існує."
                )
        return cleaned_data


class UtilityPaymentForm(forms.ModelForm):
    service = forms.ModelChoiceField(
        queryset=UtilityService.objects.all(),
        label="Комунальна послуга",
        empty_label="Оберіть послугу",
```

```
        widget=forms.Select(attrs={'class': 'form-control'})
    )
    amount_paid = forms.DecimalField(
        label="Сплачена сума (грн)",
        min_value=0.01,
        widget=forms.NumberInput(attrs={'step': '0.01', 'class': 'form-control'})
    )
    payment_date = forms.DateField(
        label="Дата платежу",
        widget=DateInput(attrs={'class': 'form-control'}),
        initial=timezone.now().date()
    )
    billing_period_start = forms.DateField(
        label="Початок розрахункового періоду",
        widget=DateInput(attrs={'class': 'form-control'})
    )
    billing_period_end = forms.DateField(
        label="Кінець розрахункового періоду",
        widget=DateInput(attrs={'class': 'form-control'})
    )
    transaction_id = forms.CharField(
        label="ID транзакції/Квитанція (необов'язково)",
        required=False,
        widget=forms.TextInput(attrs={'class': 'form-control'})
    )
    notes = forms.CharField(
        label="Примітки (необов'язково)",
        required=False,
        widget=forms.Textarea(attrs={'rows': 3, 'class': 'form-control'})
    )

    class Meta:
        model = UtilityPayment
        fields: list[str] = [
            'service', 'amount_paid', 'payment_date',
            'billing_period_start', 'billing_period_end',
            'transaction_id', 'notes'
        ]
```

```
    def clean(self) -> Dict[str, Any]:
        """
        Кастомна валідація для перевірки дат розрахункового періоду.
        """
        cleaned_data = super().clean()
        start_date = cleaned_data.get('billing_period_start')
        end_date = cleaned_data.get('billing_period_end')

        if start_date and end_date and start_date > end_date:
            self.add_error('billing_period_end',           :
            # raise forms.ValidationError       return cleaned_data


# class ExpenseFilterForm(forms.Form):
#     category = forms.ModelChoiceField(queryset=ExpenseCategory.objects.all(),
required=False, label="Категорія")
#     start_date = forms.DateField(required=False, widget=DateInput(), label="З
дати")
#     end_date = forms.DateField(required=False, widget=DateInput(), label="По
дату")
#     period = forms.ChoiceField(choices=[...], required=False, label="Період")
```

Додаток В

Код для файлу setting.py

```
from pathlib import Path

BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-
)vx)g_ynlq%1&mw92s38s%z@qzd&q5$i8tq)u&=5rlr*ha0a6u'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'management', # Наш новий додаток
]

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

```
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'config.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'config.wsgi.application'


# Database
# https://docs.djangoproject.com/en/5.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}


# Password validation
# https://docs.djangoproject.com/en/5.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
```

```
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/5.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.2/howto/static-files/

STATIC_URL = 'static/'

# Default primary key field type
# https://docs.djangoproject.com/en/5.2/ref/settings/#default-auto-field
```

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

Додаток Д

Код для файлу views.py

```
# management/views.py


from django.shortcuts import render, redirect, get_object_or_404

from django.http import HttpRequest, HttpResponse, Http404 # Http404 додано

from django.contrib.auth.decorators import login_required

from django.contrib import messages

from django.db.models import Sum, Q # F, Case, When, Value, DecimalField прибрані, бо не використовуються поки

from django.utils import timezone

from datetime import datetime, date # datetime прибрано, бо не використовується явно

from decimal import Decimal

from typing import Any, Dict, List, Optional, Union # Union прибрано


from .models import (

    Expense, ExpenseCategory, Budget, FamilyMember,

    UtilityService, MeterReading, UtilityRate, UtilityPayment

)

from .forms import ( # Імпортуємо наші нові форми

    ExpenseForm, BudgetForm, MeterReadingForm, UtilityPaymentForm

)


# ----- Допоміжні функції (залишаються без змін) -----

def get_current_family_member(request: HttpRequest) -> Optional[FamilyMember]:
```

```
    """
    if request.user.is_authenticated:
        try:
            # Переконуємось, що user не є AnonymousUser перед запитом до FamilyMember
            if hasattr(request.user, 'familymember'):
                return request.user.familymember
            else:
                if not request.user.is_anonymous:
                    member, created = FamilyMember.objects.get_or_create(user=request.user)
                    if created:
                        messages.info(request, f"Автоматично створено профіль для користувача {request.user.username}.")
                    return member
        except FamilyMember.DoesNotExist: # Цей блок може бути зайвим з get_or_create
            if not request.user.is_anonymous:
                member = FamilyMember.objects.create(user=request.user)
                messages.info(request, f"Створено профіль для користувача {request.user.username}.")
                return member
    return None


# ----- Загальні Views (dashboard_view залишається без змін) -----
@login_required
```

```
def dashboard_view(request: HttpRequest) -> HttpResponse:
    """
    Відображає головну панель управління.
    """
    context: Dict[str, Any] = {
        'current_year': timezone.now().year,
    }
    return render(request, 'management/dashboard.html', context)


@login_required
def expense_list_view(request: HttpRequest) -> HttpResponse:
    """
    Відображає список витрат з можливістю фільтрації.
    Логіка фільтрації залишається такою ж, оскільки це GET-запити.
    """
    expenses_query = Expense.objects.select_related('category',
'added_by__user').all()
    categories = ExpenseCategory.objects.all()

    category_id: Optional[str] = request.GET.get('category')
    start_date_str: Optional[str] = request.GET.get('start_date')
    end_date_str: Optional[str] = request.GET.get('end_date')
    period: Optional[str] = request.GET.get('period')
```

```
    current_d: date = timezone.now().date() # перейменовано, щоб не конфліктувати з date з datetime
    q_filter = Q()

    # ... (решта логіки фільтрації залишається без змін) ...
    if category_id:
        q_filter &= Q(category_id=category_id)

    actual_start_date: Optional[date] = None
    actual_end_date: Optional[date] = None

    if period:
        if period == 'month':
            actual_start_date = current_d.replace(day=1)
            actual_end_date = current_d
        elif period == 'quarter':
            current_quarter = (current_d.month - 1) // 3 + 1
            start_month = (current_quarter - 1) * 3 + 1
            end_month = start_month + 2
            actual_start_date = current_d.replace(month=start_month, day=1)
            # Правильний розрахунок останнього дня місяця для end_month
            if end_month == 12:
                actual_end_date = current_d.replace(month=end_month, day=31)
            else:
                actual_end_date = current_d.replace(month=end_month + 1, day=1) -
```

```
timezone.timedelta(days=1)
        elif period == 'half_year':
            if current_d.month <= 6:
                actual_start_date = current_d.replace(month=1, day=1)
                actual_end_date = current_d.replace(month=6, day=30)
            else:
                actual_start_date = current_d.replace(month=7, day=1)
                actual_end_date = current_d.replace(month=12, day=31)
        elif period == 'year':
            actual_start_date = current_d.replace(month=1, day=1)
            actual_end_date = current_d.replace(month=12, day=31)

        if actual_start_date: q_filter &= Q(date__gte=actual_start_date)
        if actual_end_date: q_filter &= Q(date__lte=actual_end_date)

    elif start_date_str and end_date_str:
        try:
            actual_start_date = timezone.datetime.strptime(start_date_str, '%Y-%m-
%d').date()
            actual_end_date = timezone.datetime.strptime(end_date_str, '%Y-%m-
%d').date()
            q_filter &= Q(date__gte=actual_start_date)
            q_filter &= Q(date__lte=actual_end_date)
        except ValueError:
            messages.error(request, "Неправильний формат дати. Використовуйте
РРРР-ММ-ДД.")
```

```
    expenses = expenses_query.filter(q_filter).order_by('-date', '-created_at')
    total_expenses: Optional[Decimal] =
expenses.aggregate(total=Sum('amount'))['total'] or Decimal('0.00')


    context = {
        'expenses': expenses,
        'categories': categories,
        'total_expenses': total_expenses,
        'selected_category': int(category_id) if category_id and category_id.isdigit()
else None,
        'start_date': start_date_str if start_date_str else (actual_start_date.strftime('%Y-
%m-%d') if actual_start_date else ''),
        'end_date': end_date_str if end_date_str else (actual_end_date.strftime('%Y-
%m-%d') if actual_end_date else ''),
        'selected_period': period,
    }
    return render(request, 'management/expense_list.html', context)



@login_required
def expense_add_view(request: HttpRequest) -> HttpResponse:
    if request.method == 'POST':
        form = ExpenseForm(request.POST)
        if form.is_valid():
            expense_instance = form.save(commit=False) # Не зберігати в БД одразу
```

```
            current_member = get_current_family_member(request)
            if current_member:
                expense_instance.added_by = current_member
            else:
                # Якщо профіль не знайдено, можна або видавати помилку,
                # або дозволити додавати витрату без прив'язки (якщо модель дозволяє added_by=None)
                messages.warning(request, "Профіль члена родини не знайдено. Витрату додано без автора.")
            expense_instance.save() # Тепер зберігаємо з автором
            messages.success(request, "Витрату успішно додано!")
            return redirect('management:expense_list')
        else:
            messages.error(request, "Будь ласка, виправте помилки у формі.")
    else:
        form = ExpenseForm() # Порожня форма для GET-запиту

    context = {
        'form': form,
        'form_title': "Додати нову витрату" # Для універсального шаблону
    }
    return render(request, 'management/generic_form.html', context) #

@login_required
def expense_report_view(request: HttpRequest) -> HttpResponse:
```

```
    today: date = timezone.now().date()
    start_of_month: date = today.replace(day=1)


    expenses_this_month = Expense.objects.filter(date__gte=start_of_month,
date__lte=today)


    report_by_category = expenses_this_month.values(
        'category__name' # Групуємо по імені категорії
    ).annotate(
        total_spent=Sum('amount') # Сумуємо витрати для кожної категорії
    ).order_by('-total_spent') # Сортуємо за спаданням суми

    # Порівняння з бюджетом на поточний місяць
    budgets_this_month = Budget.objects.filter(
        year=today.year, month=today.month
    ).select_related('category')

    budget_comparison: List[Dict[str, Any]] = []
    for budget in budgets_this_month:
        spent_in_category: Decimal = expenses_this_month.filter(
            category=budget.category
        ).aggregate(total=Sum('amount'))['total'] or Decimal(0)
```

```
        remaining: Decimal = budget.amount_limit - spent_in_category

        budget_comparison.append({
            'category_name': budget.category.name,
            'limit': budget.amount_limit,
            'spent': spent_in_category,
            'remaining': remaining,
            'over_budget': spent_in_category > budget.amount_limit,
            'percentage_spent': (spent_in_category / budget.amount_limit * 100) if
budget.amount_limit > 0 else 0
        })

    context = {
        'report_by_category': report_by_category,
        'budget_comparison': budget_comparison,
        'report_period_start': start_of_month,
        'report_period_end': today,
    }
    return render(request, 'management/expense_report.html', context)


@login_required
def budget_list_view(request: HttpRequest) -> HttpResponse:
    year_filter: Optional[str] = request.GET.get('year')
```

```
    month_filter: Optional[str] = request.GET.get('month')


    budgets_query = Budget.objects.select_related('category').all()


    current_year: int = timezone.now().year
    existing_years_db = Budget.objects.values_list('year',
flat=True).distinct().order_by('year')
    available_years_set = set(existing_years_db)
    for i in range(current_year - 1, current_year + 3): # додаємо роки навколо
поточного
        available_years_set.add(i)
    available_years = sorted(list(available_years_set))


    q_filter = Q()
    selected_year_val: Optional[int] = None
    selected_month_val: Optional[int] = None


    if year_filter and year_filter.isdigit():
        selected_year_val = int(year_filter)
        q_filter &= Q(year=selected_year_val)
    if month_filter and month_filter.isdigit():
        selected_month_val = int(month_filter)
        q_filter &= Q(month=selected_month_val)


    budgets = budgets_query.filter(q_filter).order_by('year', 'month',
'category__name')
```

```
    context = {
        'budgets': budgets,
        'available_years': available_years,
        'selected_year': selected_year_val,
        'selected_month': selected_month_val,
        'months': [(i, f"{timezone.datetime(2000, i, 1).strftime('%B')} ({i:02d})") for i in range(1, 13)] # Назви місяців
    }
    return render(request, 'management/budget_list.html', context)


@login_required
def budget_add_view(request: HttpRequest) -> HttpResponse:

    # Для оновлення (якщо ми перейдемо на edit view пізніше):
    # instance = get_object_or_404(Budget, pk=budget_id) if budget_id else None
    # form = BudgetForm(request.POST or None, instance=instance)

    if request.method == 'POST':
        form = BudgetForm(request.POST)
        if form.is_valid():
            # BudgetForm.clean() вже обробляє унікальність.
            # ModelForm.save() автоматично обробляє create/update.
            # Однак, для логіки "update_or_create", потрібно це робити вручну
            # якщо ми не передаємо instance в форму.
```

```
            category = form.cleaned_data['category']
            month = form.cleaned_data['month']
            year = form.cleaned_data['year']
            amount_limit = form.cleaned_data['amount_limit']

            budget_instance, created = Budget.objects.update_or_create(
                category=category,
                month=month,
                year=year,
                defaults={'amount_limit': amount_limit}
            )

            if created:
                messages.success(request, f"Бюджет для
{budget_instance.category.name} на
{budget_instance.month:02d}/{budget_instance.year} успішно створено.")
            else:
                messages.info(request, f"Бюджет для {budget_instance.category.name}
на {budget_instance.month:02d}/{budget_instance.year} успішно оновлено.")
            return redirect('management:budget_list')
        else:
            messages.error(request, "Будь ласка, виправте помилки у формі.")
    else:
        form = BudgetForm()

    context = {
```

```
        'form': form,
        'form_title': "Додати або оновити бюджет"
    }
    return render(request, 'management/generic_form.html', context)

@login_required
def meter_reading_list_view(request: HttpRequest) -> HttpResponse:

    readings_query = MeterReading.objects.select_related('service').all()
    services = UtilityService.objects.all()

    service_id_filter: Optional[str] = request.GET.get('service')
    q_filter = Q()
    selected_service_val: Optional[int] = None

    if service_id_filter and service_id_filter.isdigit():
        selected_service_val = int(service_id_filter)
        q_filter &= Q(service_id=selected_service_val)

    readings = readings_query.filter(q_filter).order_by('service__name', '-
reading_date')

    context = {
        'readings': readings,
        'services': services,
```

```
        'selected_service': selected_service_val,
    }
    return render(request, 'management/meter_reading_list.html', context)

@login_required
def meter_reading_add_view(request: HttpRequest) -> HttpResponse:
    if request.method == 'POST':
        form = MeterReadingForm(request.POST)
        if form.is_valid():
            form.save()
            messages.success(request, "Показник лічильника успішно додано!")
            return redirect('management:meter_reading_list')
        else:
            messages.error(request, "Будь ласка, виправте помилки у формі.")
    else:
        form = MeterReadingForm()

    context = {
        'form': form,
        'form_title': "Внести новий показник лічильника"
    }
    return render(request, 'management/generic_form.html', context)


@login_required
```

```
def utility_payment_list_view(request: HttpRequest) -> HttpResponse:
    payments_query = UtilityPayment.objects.select_related('service').all()
    services = UtilityService.objects.all()

    service_id_filter: Optional[str] = request.GET.get('service')
    q_filter = Q()
    selected_service_val: Optional[int] = None

    if service_id_filter and service_id_filter.isdigit():
        selected_service_val = int(service_id_filter)
        q_filter &= Q(service_id=selected_service_val)

    payments = payments_query.filter(q_filter).order_by('-payment_date',
'service__name')

    context = {
        'payments': payments,
        'services': services,
        'selected_service': selected_service_val,
    }
    return render(request, 'management/utility_payment_list.html', context)

@login_required
def utility_payment_add_view(request: HttpRequest) -> HttpResponse:
```

```
    if request.method == 'POST':
        form = UtilityPaymentForm(request.POST)
        if form.is_valid():
            form.save()
            messages.success(request, "Платіж успішно додано!")
            return redirect('management:utility_payment_list')
        else
            messages.error(request, "Будь ласка, виправте помилки у формі.")
    else:
        form = UtilityPaymentForm()

    context = {
        'form': form,
        'form_title': "Додати новий платіж за комунальні послуги"
    }
    return render(request, 'management/generic_form.html', context)


@login_required
def calculate_utility_cost_view(request: HttpRequest) -> HttpResponse:


    services = UtilityService.objects.all()
    calculation_result: Optional[Dict[str, Union[str, Decimal, date]]] = None # date
додано до Union
    selected_service_id_str: Optional[str] = request.GET.get('service')
    selected_service_id_val: Optional[int] = None
```

```
    if selected_service_id_str and selected_service_id_str.isdigit():
        selected_service_id_val = int(selected_service_id_str)
        try:
            service = UtilityService.objects.get(pk=selected_service_id_val) # Використовуємо get замість get_object_or_404, бо це не основна дія сторінки

            latest_readings = MeterReading.objects.filter(service=service).order_by('-reading_date')[:2]

            if len(latest_readings) < 2:
                messages.warning(request, f"Недостатньо показників для розрахунку споживання для послуги '{service.name}'. Потрібно щонайменше два.")
            else:
                current_reading: MeterReading = latest_readings[0]
                previous_reading: MeterReading = latest_readings[1]

                if current_reading.reading_date <= previous_reading.reading_date: # Сувора нерівність не потрібна, якщо дати унікальні
                    messages.error(request, "Помилка даних: останній показник не новіший за попередній, або дати однакові.")
                else:
                    consumption: Decimal = current_reading.reading_value - previous_reading.reading_value
                    if consumption < Decimal(0): # Порівняння з Decimal
                        messages.warning(request, "Споживання від'ємне. Перевірте показники лічильників. Встановлено споживання 0.")
```

```
            consumption = Decimal(0)

            active_rate: Optional[UtilityRate] = UtilityRate.objects.filter(
                service=service,
                effective_date__lte=current_reading.reading_date
            ).order_by('-effective_date').first()

            if not active_rate:
                messages.error(request, f"Не знайдено діючий тариф для послуги '{service.name}' на дату {current_reading.reading_date.strftime('%Y-%m-%d')}.")
            else:
                cost: Decimal = consumption * active_rate.rate_per_unit
                calculation_result = {
                    'service_name': service.name,
                    'current_reading_value': current_reading.reading_value,
                    'current_reading_date': current_reading.reading_date, # Передаємо об'єкт date
                    'previous_reading_value': previous_reading.reading_value,
                    'previous_reading_date': previous_reading.reading_date, # Передаємо об'єкт date
                    'consumption': consumption,
                    'unit': service.unit_of_measurement,
                    'rate': active_rate.rate_per_unit,
                    'cost': cost.quantize(Decimal("0.01"))
                }
                messages.success(request, "Розрахунок виконано.")
```

```
        except UtilityService.DoesNotExist:
            messages.error(request, "Обрана послуга не знайдена.")
            selected_service_id_val = None # Скидаємо, якщо послуга не знайдена
    elif 'service' in request.GET : # Якщо параметр 'service' є, але він не валідний
        messages.info(request, "Оберіть послугу зі списку для розрахунку.")

    context = {
        'services': services,
        'calculation_result': calculation_result,
        'selected_service_id': selected_service_id_val
    }
    return render(request, 'management/calculate_utility_cost.html', context)
@login_required
def family_member_list_view(request: HttpRequest) -> HttpResponse:

    family_members = FamilyMember.objects.select_related('user').all()
    context = {
        'family_members': family_members
    }
    return render(request, 'management/family_member_list.html', context)
```

Додаток Е

Код для файлу urls.py

```
# management/urls.py

from django.urls import path
from . import views # Імпортуємо наші майбутні views

app_name: str = 'management' # Простір імен для URL-адрес цього додатка

urlpatterns: list = [
    # Головна сторінка додатку (приклад)
    path('', views.dashboard_view, name='dashboard'),

    # Фінансовий облік
    path('expenses/', views.expense_list_view, name='expense_list'),
    path('expenses/add/', views.expense_add_view, name='expense_add'),
    path('expenses/report/', views.expense_report_view, name='expense_report'),
    path('budgets/', views.budget_list_view, name='budget_list'),
    path('budgets/add/', views.budget_add_view, name='budget_add'),

    # Облік комунальних послуг
    path('utilities/readings/', views.meter_reading_list_view,
name='meter_reading_list'),
    path('utilities/readings/add/', views.meter_reading_add_view,
name='meter_reading_add'),
```

path('utilities/payments/', views.utility_payment_list_view, name='utility_payment_list'),

path('utilities/payments/add/', views.utility_payment_add_view, name='utility_payment_add'),

path('utilities/calculate/', views.calculate_utility_cost_view, name='calculate_utility_cost'), # Для розрахунку

 # Профілі членів родини (базово)

path('family/', views.family_member_list_view, name='family_member_list'),

 # Додавання/редагування профілів часто реалізується через адмінку або кастомні форми UserCreationForm

Додаток Ж

Код для файлу models.py

```
    class Meta:
        verbose_name: str = "Показник лічильника"
        verbose_name_plural: str = "Показники лічильників"
        ordering: List[str] = ['service', '-reading_date']


        unique_together: Tuple[str, str] = ('service', 'reading_date')



class UtilityPayment(models.Model):
    service: models.ForeignKey = models.ForeignKey(UtilityService, on_delete=models.CASCADE, verbose_name="Послуга")
    amount_paid: models.DecimalField = models.DecimalField(max_digits=10, decimal_places=2, verbose_name="Сплачена сума")
    payment_date: models.DateField = models.DateField(default=timezone.now, verbose_name="Дата платежу")
    billing_period_start: models.DateField = models.DateField(verbose_name="Початок розрахункового періоду")
    billing_period_end: models.DateField = models.DateField(verbose_name="Кінець розрахункового періоду")
    # Можна додати посилання на MeterReading, якщо платіж безпосередньо пов'язаний з конкретним показником
    # meter_reading = models.ForeignKey(MeterReading, on_delete=models.SET_NULL, null=True, blank=True, verbose_name="Показник лічильника")
```

transaction_id: models.CharField = models.CharField(max_length=255, blank=True, null=True, verbose_name="ID транзакції/Квитанція")

notes: models.TextField = models.TextField(blank=True, null=True, verbose_name="Примітки")

def __str__(self) -> str:

return f"Оплата {self.service.name}: {self.amount_paid} грн ({self.payment_date.strftime('%Y-%m-%d')})"

class Meta:

verbose_name: str = "Платіж за комунальні послуги"

verbose_name_plural: str = "Платежі за комунальні послуги"

ordering: List[str] = ['-payment_date', 'service']