

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавра

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПОШУКУ РОБІТНИКІВ ДЛЯ
ВЕДЕННЯ ДОМАШНЬОГО ГОСПОДАРСТВА
HOUSEKEEPING STAFF SEARCH SOFTWARE

Виконав: студент(ка) 4 курсу, групи КНТ-138
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

Марков К.А.

(прізвище та ініціали)

Керівник Скрупський С.Ю.

(прізвище та ініціали)

Рецензент Голуб Т.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “_____” _____ 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Маркова Кирила Андрійовича
 (прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення пошуку робітників для
ведення домашнього господарства
Housekeeping Staff Search Software

керівник проєкту (роботи) Скрупський Степан Юрійович, к.т.н., доцент,
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від «27» квітня 2022 року № 102

2. Строк подання студентом проєкту (роботи) червень 2022 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми.
3. Розробка програми. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Скрупський С.Ю., доцент		
Нормоконтролер	Камінська Ж.К., асистент		

7. Дата видачі завдання «4» квітня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2 тиждень	Розділ 1
3	Розробка архітектури програми	3 тиждень	Розділ 2
4	Розробка програми	4-5 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	6 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	7 тиждень	Додатки
7	Захист роботи	8 тиждень	

Студент(ка)

(підпис)

Марков К.А.

(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Скрупський С.Ю.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 81 с., 23 рис., 1 табл., 3 дод., 9 джерел.

ДОМАШНЄ ГОСПОДАРСТВО, РОБІТНИК, ГОСПОДАР, ЗАМОВЛЕННЯ, ВЕБЗАСТОСУНОК.

Об'єкт дослідження – процес пошуку робітників для ведення домашнього господарства. Предмет дослідження – програмне забезпечення пошуку робітників для ведення домашнього господарства. Мета дослідження – розробити програмне забезпечення пошуку робітників для ведення домашнього господарства для підтримки планування цих процесів за допомогою зовнішніх працівників.

Матеріали, методи та технічні засоби: структурне, об'єктно-орієнтоване програмування, мова Python, фреймворк веброзробки Django, персональний комп'ютер під керування операційної системи Windows 10.

Результати. Результатом виконання роботи є створена програма у вигляді вебзастосунку, включаючи створені на основі використання фреймворку веброзробки Django і шаблону проєктування Model View Template відповідні класи, що визначають структуру програми, розроблена структура база даних.

Висновки. Виконання роботи дозволило створити програму, яка забезпечує підтримку людей, які знаходяться в пошуку робітників для ведення домашнього господарства у себе вдома, в будинках, які їм належать або ними керуються, або які планують таку роботу в подальшому, та підтримку людей, які готові виконувати роботу щодо ведення домашнього господарства в будинках інших людей.

Галузь використання. Ведення домашніх справ за неможливості їх реалізації силами сім'ї.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 81 pages, 23 figures, 1 table, 3 appendixes, 9 sources.

HOUSEHOLD, WORKER, HOST, ORDER, WEBSITE.

The object of research is the process of finding workers to run a household. The subject of the research is software for finding workers for housekeeping. The purpose of the study is to develop software for finding workers for housekeeping to support the planning of these processes with the help of external workers.

Materials, methods and technical means: structural, object-oriented programming, Python language, Django web development framework, personal computer running Windows 10.

Results. The result of the work include a program created in the form of a web application, including the appropriate classesw, which were created on the basis of the Django web development framework and the Model View Template design template and which define the structure of the program, as well as the developed database structure.

Conclusions. The program, which provides support for people who are looking for workers to run the household at home, in homes that they own or manage, or who plan to do so in the future, and provides support for people who are willing to do the job. regarding housekeeping in other people's homes, work has created in the work.

Field of use. Doing housekeeping acitivities when it is impossible to do such activities by the family members.

ЗМІСТ

	С.
Перелік скорочень та умовних познач	7
Вступ	8
1 Аналіз предметної області	9
1.1 Проблема пошуку робітників для ведення домашнього господарства ...	9
1.2 Огляд програмних аналогів	10
1.3 Висновки за розділом 1 та завдання роботи	13
2 Розробка архітектури програми	14
2.1 Визначення функціональних вимог до програми	14
2.2 Вибір засобів розробки	17
2.3 Проектування бази даних	18
2.4 Висновки за розділом 2	28
3 Розробка програми	30
3.1 Алгоритм основи функціонування програми	30
3.2 Опис реалізованих класів програми	32
3.3 Висновки за розділом 3	37
4 Експлуатація та тестування програми	38
4.1 Призначення програми	38
4.2 Умови виконання програми	38
4.3 Експлуатація програми	38
4.4 Тестування програми	43
Висновки	44
Перелік джерел посилання	45
Додаток А Технічне завдання	46
Додаток Б Текст програми	50
Додаток В Слайди презентації	75

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

MVT – Model-View-Template;

БД – база даних.

ВСТУП

Роботи з ведення домашнього господарства – це достатньо широкий набір робіт, які включають, наприклад, достатньо популярні клірингові роботи, а також певні садові роботи, можливо навіть деякі роботи, які можна віднести до будівельних або ремонтних [1]-[2]. Тобто під поняттям робіт з ведення домашнього господарства розглядається широкий клас завдань, який пов'язаний з виконанням робіт як всередині приміщення будинку, так і на території прилеглої ділянки. Такі роботи часто виконуються сім'єю, яка проживає в будинку, квартирі або окремими її членами, проте не кожна сім'я може або має час виконувати такі роботи. Високе навантаження на роботі, високий рівень заробітної платні в країні, що характеризує стан розвиненої економіки зокрема та частково економіки країн, що розвиваються, для певної частини їх громадян, сприяють тому, щоб перекласти відповідальність за виконання таких робіт на робітників, які до сім'ї відношення не мають.

У такому випадку виникає потреба в винайманні робітників з чіткими завданнями щодо ведення домашнього господарства. Звичайно, що за сучасного розвитку інформаційних технологій, для широкого охоплення взаємодії власників будинків з такими робітниками потрібне відповідне програмне забезпечення. Таке програмне забезпечення має бути представлене вебзастосунком з реалізацією відповідних функцій.

Метою роботи в підсумку було сформульовано виконання розробки програмного забезпечення пошуку робітників для ведення домашнього господарства для підтримки планування цих процесів за допомогою зовнішніх працівників.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Проблема пошуку робітників для ведення домашнього господарства

Знайти і винайняти робітника для ведення домашнього господарства може бути вирішенням проблеми, яка стоїть перед людиною, проте для того, щоб це було ефективне рішення проблеми, потрібно враховувати певні питання перед цим:

- по-перше, важливо знайти когось, кому можна довіряти, і це звичайно стосується багатьох сфер, а особливо, якщо ця людина буде прибирати в будинку, поки власник буде на роботі або виконуватиме певні інші доручення, тому важливо поцікавитися рекомендаціями цього робітника, подумати про тонкощі використання його здібностей;

- по-друге, деякі такі працівники працюють як приватні власники власного бізнесу, але власнику може бути зручніше працювати з більшою клінінговою компанією, яка наймає власних працівників, проте варто розуміти, що у обох цих варіантів є плюси і мінуси: перевага найму клінінгової компанії полягає в тому, що вони відповідають за перевірку співробітників, щоб переконатися, що результат був саме таким, як очікувалось, з іншого боку, якщо власник хоче, щоб одна і та ж людина прибирала будинок щотижня, потрібно вибрати приватного підприємця – оборот у клінінгових компаній, як правило, вищий, і вони можуть щоразу надсилати когось іншого;

- деякі завдання з прибирання будинку є стандартними, наприклад, підмітання, миття душових та туалетів, однак, можливо, доведеться домовитися про будь-які додаткові завдання, такі як прання або посуд, які потрібно виконувати регулярно;

- у будь-якому випадку з новим робітником бажано почати з пробного періоду в два-чотири тижні: це дасть робітникам час звикнути до очікувань власника, а власнику дасть можливість дозволити робітникам влаштуватися в

тому, чого потребує будинок, обмежуючи додатково початковий період, можна не отримати справедливую оцінку того, на робітники справді здатні, проте якщо власник не задоволений після кількох відвідувань і чітких очікувань, то є велика ймовірність, що ці стосунки не спрацюють, випробувальний період захищає власника і робітника;

– потрібно чітко уявляти, що саме робітник буде, а що не буде робити у будинку, розглядаючи можливість спільного створення переліку завдань, які будуть виконуватися з будь-якими конкретними інструкціями, які можуть бути, обговорення цих питань до початку роботи допоможе, можливо, власник також захоче домовитися про спосіб запиту та оплати додаткових робіт, які виходять за межі звичайного прибирання або якоїсь іншої процедури з ведення домашнього господарства, обов'язково потрібно встановити певні межі того, що не відбудуватиметься у будинку [3]-[4].

1.2 Огляд програмних аналогів

Вивчивши особливості реалізації взаємодії між власниками будинків та робітниками, які займаються домашнім господарством у них, важливо тепер зрозуміти, якими можливостями повинна володіти програма, яка буде підтримувати таку взаємодію. Саме для цього і було виконано огляд програмних аналогів.

Програмним аналогом розробки зокрема є програма MaidsApp (рис. 1.1) [5], зважаючи на відповідну направленість її роботи та наявні можливості. Програма, по-перше, дозволяє працювати з нею зокрема через вебсайт, що дозволяє отримувати доступ всім користувачам фактично без обмежень. Проте вона має достатньо обмежену процедуру реєстрації, адже потрібно саме зареєструватися окремо в застосунку. Це характерно для даної предметної області, адже потрібно чітко ідентифікувати користувача і мати всі необхідні дані про нього в застосунку, адже це питання довіри між власником будинку і робітником.

The screenshot displays the MaidsApp website. At the top, there is a navigation bar with links: About, Find Maids (highlighted), Reviews, Blogs, and Contact. A user profile icon labeled 'Jeko Rets' and a 'Book a Cleaning' button are also present. The main header features a woman cleaning a white cabinet, with the text 'Set Your Location' and 'Tell us where you are, and we will send you a maid your way'. Below this is a 'Search for Maids' section. It includes a location input field with 'New York, NY, USA' and a search icon. A note states: 'Some of property information was not detected and will need to be entered manually.' There are two tabs: 'Residence' (selected) and 'Business'. A square footage slider is set to '220 Sqr.Ft' on a scale from 0 to 6000. Below the slider is an 'Apt / Unit No.' input field. At the bottom of the search section, there are fields for 'Bathrooms' (set to 1) and 'Bedrooms' (set to 1), each with minus and plus buttons. A large blue 'Find Your Maid' button is at the bottom.

Рисунок 1.1 – Інтерфейс програми MaidsApp

В програмі фактично виконується пошук самих робітників. Під час реалізації пошуку потрібно вказати адресу будинку, задати відстань, кількість апартаментів, ванних кімнат, спальних кімнат.

Результати пошуку відображаються достатньо обмежено (рис. 1.2) [5], адже вони містять прізвище та ім'я працівника, адресу працівника, відстань, інші ж параметри відображають те, яка робота буде виконуватися за параметрами пошуку, надаються варіанти виконання прибирання та вартість цих варіантів. Власнику надається можливість забронювати послугу. Після цього у нього буде доступ до всіх власних замовлень, який також підкріплений можливостями внесення власних адрес для швидкого доступу.

Search for Maids

Pasadena, CA, USA

Maid Name :Ana Mayora Distance : 23.95 Mile

Address : Location : Resident

Alondra Blvd & Avalon Blvd, West Rancho Sqr.Ft : 1500

Dominguez, CA 90220, USA Bathrooms : 1

Bedrooms : 1

Maid Schedule : Choose a Cleaning Package :

☒ ASAP ☐ Tomorrow at 8:00 AM

☐ Schedule for Later

Quick Clean - \$105.36

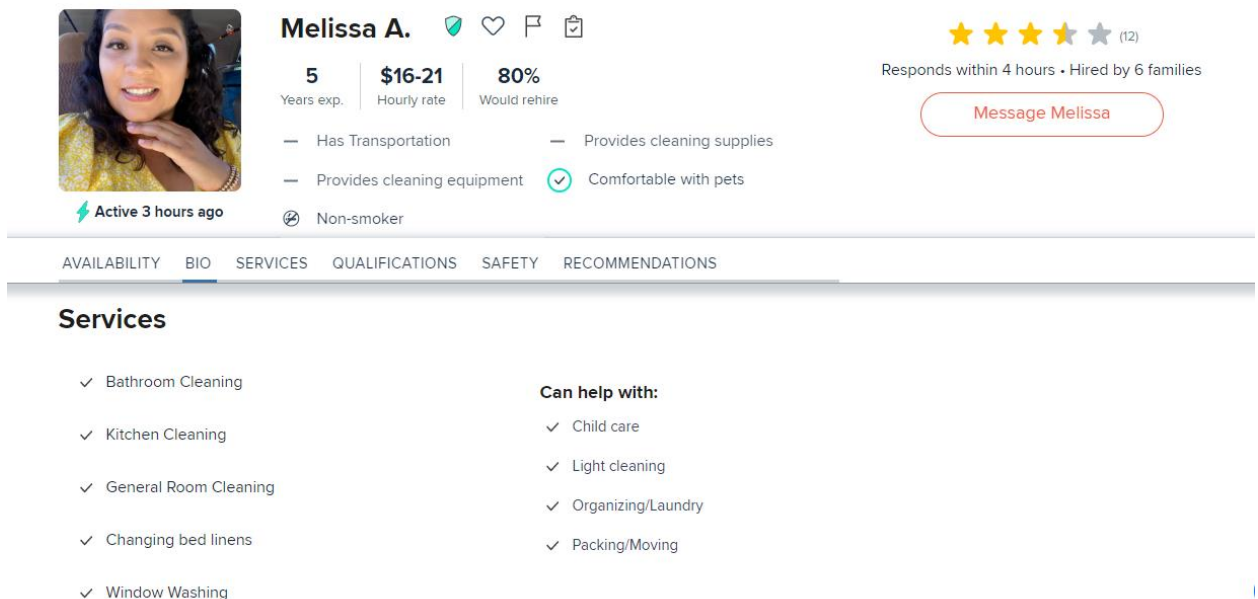
Enter Promocode Apply Payable : \$ 105.36

Confirm Your Maid

Рисунок 1.2 – Інтерфейс результатів пошуку програми MaidsApp

В програмі при цьому відсутні можливості перегляду детальної інформації про працівника, хоча і подається повне ім'я, окрім того є фіксований набір робіт, і зокрема таким чином знайти робітника для роботи в саду з обрізання дерев не вдасться. Це є її обмеженнями. Їх необхідно подолати в програмі, яка має бути розроблена в роботі.

Ще одним аналогом є вебсайт care.com (рис. 1.3) [6]. Він дозволяє переглядати перелік робіт, які здатен виконувати працівник. Окрім того працівнику виставляються оцінки, що дозволяє таким чином сприймати одразу незнайомого робітника. Окрім того можна переглянути відгуки одразу. Проте відгуків за окремим процедурами немає. Також варто зазначити, що програма повністю працює з боку власника будинку, наприклад, оцінювати можна саме робітників, не навпаки. Але з іншого боку параметри, зокрема та ж сама вартість, встановлюються самим робітником, а власник на них фактично не впливає, тобто він не може сформувати якусь особливу пропозицію.



Melissa A.    

5 Years exp. | **\$16-21** Hourly rate | **80%** Would rehire

★ ★ ★ ★ ★ (12)
Responds within 4 hours • Hired by 6 families

[Message Melissa](#)

⚡ Active 3 hours ago

— Has Transportation — Provides cleaning supplies
— Provides cleaning equipment ✓ Comfortable with pets
🚭 Non-smoker

AVAILABILITY BIO SERVICES QUALIFICATIONS SAFETY RECOMMENDATIONS

Services

- ✓ Bathroom Cleaning
- ✓ Kitchen Cleaning
- ✓ General Room Cleaning
- ✓ Changing bed linens
- ✓ Window Washing

Can help with:

- ✓ Child care
- ✓ Light cleaning
- ✓ Organizing/Laundry
- ✓ Packing/Moving

Рисунок 1.3 – Інтерфейс вебсайту care.com

1.3 Висновки за розділом 1 та завдання роботи

Вивчено особливості реалізації взаємодії між власниками будинків та робітниками, фактори, які при цьому приймаються до уваги. Розглянуто і проаналізовано можливості програмних аналогів, виділено їх позитивні та негативні особливості. Програма, яка розробляється, повинна надавати можливість створення відгуків про окремі навички, про замовників, вносити план робіт, розділяючи його за частинами будинку, створювати пропозиції з особливими умовами робіт.

Завдання роботи:

- визначення вимог до програми;
- визначення мови програмування і засобів, за допомогою яких має бути розроблена програма;
- проєктування бази даних;
- проєктування програми створення персональних відгуків про страви;
- реалізація програми пошуку робітників для ведення домашнього господарства.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Визначення функціональних вимог до програми

Проведений аналіз аналогів визначив основні функціональні вимоги до програми та її особливості, що в результаті сформували наступний перелік вимог для робітника (рис. 2.1), власника житла і адміністратора:

- реєстрація в якості працівника і в якості власника;
- авторизація;
- змінювання даних власного профілю;
- визначення переліку будинків, якими керує власник;
- визначення складових будинку;
- перегляд власних будинків;
- визначення помічників у керуванні власними будинками;
- додавання, редагування категорій процедур;
- перегляд існуючих категорій процедур;
- перегляд існуючих процедур за категорією;
- пошук процедур;
- пошук варіантів обслуговування будинку;
- визначення процедур, які виконує працівник;
- перегляд переліку процедур, які виконує працівник;
- визначення варіантів виконання процедур працівником;
- замовлення процедур;
- скасування замовлення;
- взяття замовлення на виконання;
- підтвердження виконання замовлення;
- змінювання складу замовлення;
- перегляд переліку власних замовлень;
- перегляд переліку адресованих замовлень;
- створення повторюваного обслуговування;
- визначення плану робіт за частинками будинку в замовленні;

- визначення власного графіку роботи працівника;
- перегляд графіку робіт працівника;
- перегляд доступного часу у працівника;
- створення відгуку про працівника;
- створення відгуку про процедуру;
- перегляд переліку відгуків про працівника;
- перегляд переліку відгуків за процедурою, яку виконує працівник;
- створення запиту;
- прийняття запиту на виконання;
- скасування запиту;
- перегляд переліку адресованих запитів;
- перегляд переліку власних запитів;
- керування доступом користувачів.

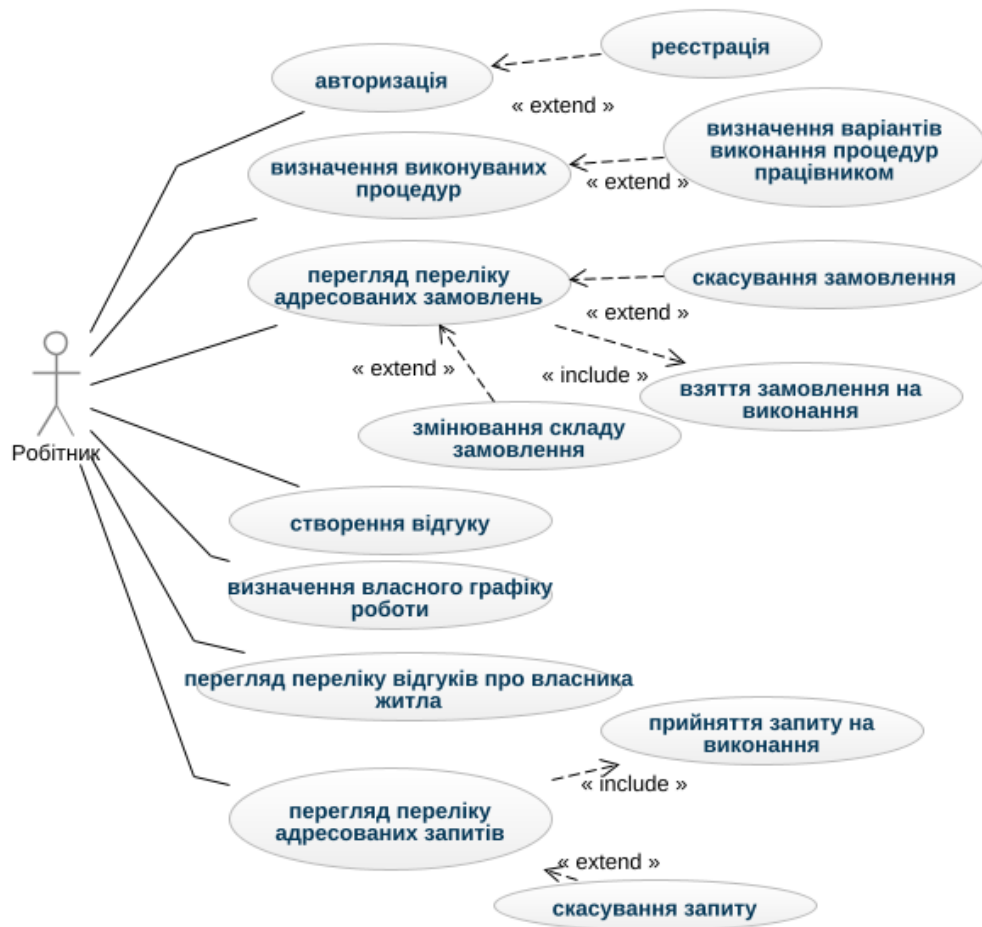


Рисунок 2.1 – Діаграма варіантів використання системи працівником

Набір сценаріїв використання програми власником житла приведено на рис. 2.2, а для адміністратора на рис. 2.3.

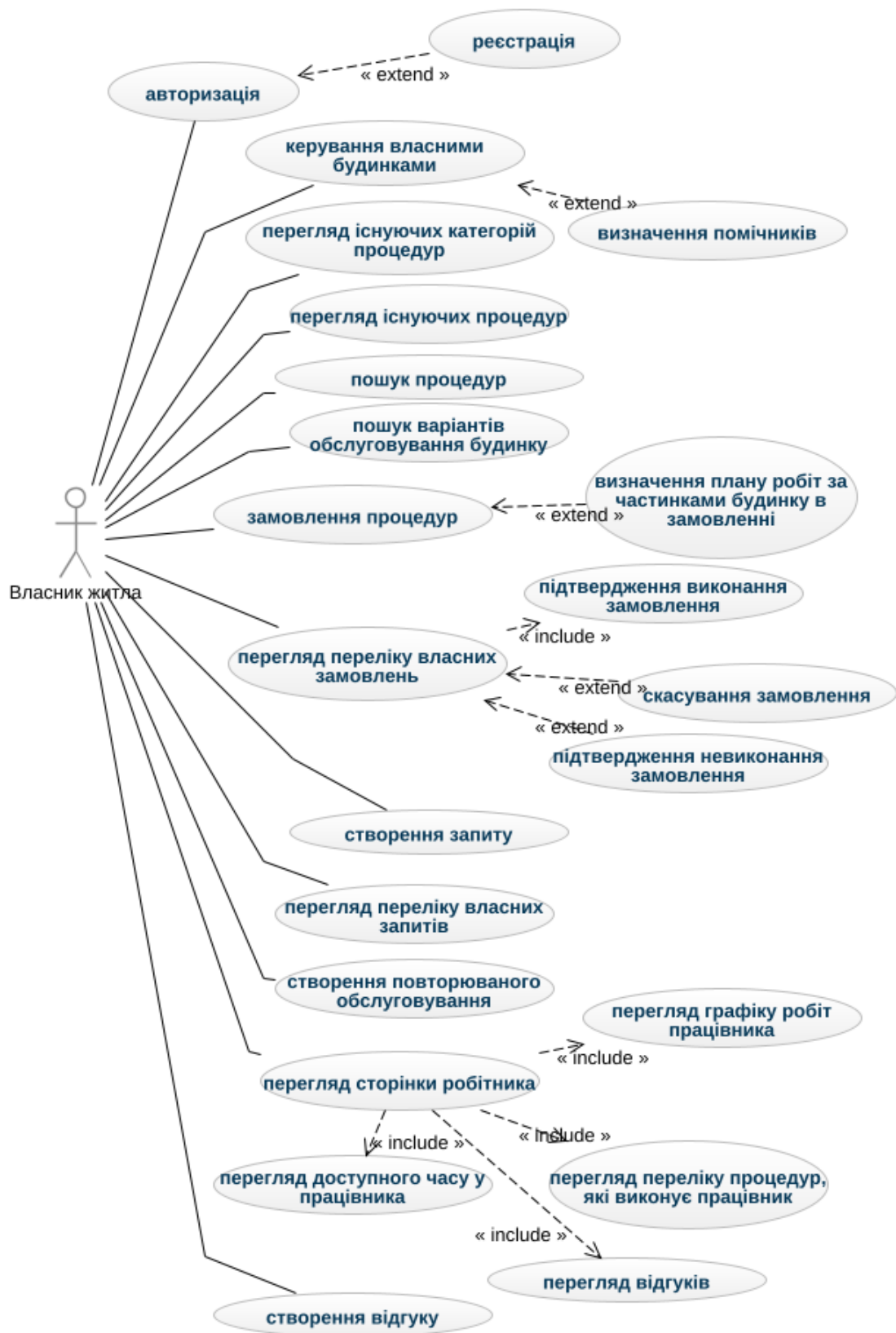


Рисунок 2.2 – Діаграма варіантів використання системи власником житла



Рисунок 2.3 – Діаграма варіантів використання системи адміністратором

2.2 Вибір засобів розробки

Виконувався вибір саме засобів розробки, оскільки вебробка на даний момент виконується, використовуючи не просто чисту мову програмування, а цілий набір засобів. Вибір (табл. 2.1) був між фреймворком Django, що ґрунтується на мові Python [7], і платформою Node.JS, в основі якої мова JavaScript. Обидва засоби є дуже розповсюдженими, а вибір зроблений на користь Django [8] і Python через те, що таке рішення дозволяє швидше розробити проєкт, вимагаючи менше додаткових дій.

Таблиця 2.1 – Вибір засобів розробки

Показник	Django	Node.JS
Мова для розробки в основі рішення	Python	JavaScript
Розробка серверної сторони	+	+
Вільне використання	+	+
Підтримка розробників	+	+
Ефективність самого процесу розроблення програми	+	+—
Шаблони створення вебсторінок	+	—
Захист даних	+	+

Враховуючи це, програма в результаті була побудована на основі патерну проєктування Model View Template (MVT) [9].

2.3 Проєктування бази даних

У даному підрозділі виконано проєктування бази даних (БД) і представлено виділені в результаті таблиці і їх зв'язки.

Дані працівника, який займається веденням домашнього господарства, зберігаються в таблиці Housekeeper, а характеристики цього працівника представлені наступним чином:

- id – ідентифікатор працівника, який займається веденням домашнього господарства;
- username – логін, який використовує користувач, що є працівником, для авторизації в програмі;
- password – пароль користувач для авторизації працівника;
- email – електронна пошта працівника;
- phone – номер телефону працівника;
- address – адреса, за якою працівник проживає або з якої стандартно розпочинає свою роботу;
- address_long – довгота координати цієї адреси;
- address_lat – широта координати цієї адреси;
- distance – відстань, за якою від власної адреси готовий працювати робітник;
- first_name – справжнє ім'я працівника;
- last_name – справжнє прізвище працівника;
- details – опис працівником себе;
- response – середня оцінка, яку отримав працівник за свою роботу від замовників.

Дані власника будинку, який бажає знайти працівника для ведення домашнього господарства або користується послугами вже знайденого

раніше працівника, представлені таблицею Owner. Це фактично сутність замовника в цій системі. Характеристики цього власника представлені наступним чином:

- id – ідентифікатор власника будинку (або замовника);
- username – логін власника для авторизації ним в програмі;
- password – пароль власника для авторизації в програмі;
- email – електронна пошта власника;
- phone – актуальний номер телефону власника;
- details – опис власника з деталізацією особливостей його вимог;
- first_name – справжнє ім'я власника;
- last_name – справжнє прізвище власника.

Дані будинку зберігаються в таблиці House, а характеристики цього працівника представлені наступним чином:

- id – ідентифікатор будинку;
- address – адреса будинку;
- apartment – номер апартаментів у будинку;
- address_long – довгота координати адреси будинку;
- address_lat – широта координати адреси будинку;
- area – площа будинку або апартаментів у будинку;
- owner – власник цього будинку;
- name – псевдонім, який використовується власником для назви будинку.

Поле owner з'єднує таблицю House з таблицею Owner через зв'язок один до багатьох: апартаменти можуть мати тільки одного власника – фактично це людина, яка внесла дані про них в програму, а власник як користувач програми може зберігати в ній дані про різні апартаменти, де він веде господарство.

Дані складової частини будинку зберігаються в таблиці HousePart, а характеристики цієї частини представлені наступним чином:

- id – ідентифікатор частини будинку (це може бути кімната, сад тощо);
- name – псевдонім, який використовується для цієї частини будинку для зручного доступу;
- area – площа цієї частини будинку;
- specification – опис особливостей цієї частини будинку;
- house – будинок, частиною якого є дана.

Поле house з'єднує таблицю HousePart з таблицею House через зв'язок один до багатьох: один будинок може мати багато частин, а кожна частина належить тільки одному будинку.

Дані категорії процедур з ведення домашнього господарства зберігаються в таблиці ProcedureCategory, а характеристики цієї категорії представлені наступним чином:

- id – ідентифікатор категорії процедур;
- name – назва категорії процедур.

Дані процедури з ведення домашнього господарства зберігаються в таблиці Houseprocedure, а характеристики цієї процедури представлені наступним чином:

- id – ідентифікатор процедури;
- name – назва процедури;
- procedurecategory – категорія процедури;
- specification – опис деталей процедури.

Поле procedurecategory з'єднує таблицю Houseprocedure з таблицею ProcedureCategory через зв'язок один до багатьох: процедура з ведення домашнього господарства належить тільки одній категорії, а ось кожна категорія процедур представляється багатьма процедурами.

Дані процедури з ведення домашнього господарства, яку здатен виконувати працівник, зберігаються в таблиці Housekeeperprocedure, а характеристики цієї процедури працівника представлені наступним чином:

- id – ідентифікатор процедури працівника;

- houseprocedure – процедура з ведення домашнього господарства;
- housekeeper – працівник, який реалізує цю процедуру;
- specification – особливості виконання цієї процедури даним працівником;
- state – стан процедури у робітника, тобто чи вона ще надається;
- score – середня оцінка процедури.

Поле housekeeper з'єднує таблицю Housekeeperprocedure з таблицею Housekeeper через зв'язок один до багатьох: одна процедура з ведення домашнього господарства може виконуватися різними працівниками, заради чого фактично і створюється програма, а окремий запис процедури працівника логічно, що виконується одним конкретним працівником.

Поле houseprocedure з'єднує таблицю Housekeeperprocedure з таблицею Houseprocedure через зв'язок один до багатьох: одна процедура з ведення домашнього господарства може мати багато реалізацій у виконанні різних працівників, а окремий запис процедури працівника стосується тільки однієї конкретної процедури.

Дані варіанту процедури з ведення домашнього господарства, яку здатен виконувати працівник, (поверхнєве прибирання, глибоке прибирання) зберігаються в таблиці HousekeeperprocedureVariant, а характеристики цього варіанту представлені наступним чином:

- id – ідентифікатор варіанту;
- housekeeperprocedure – процедура з ведення домашнього господарства;
- name – назва варіанту, встановлена працівником;
- specification – опис деталей варіанту;
- price – вартість варіанту;
- duration – тривалість виконання цього варіанту процедури;
- area – максимальна площа приміщення, за якої можна використати цей варіант.

Поле `housekeeperprocedure` з'єднує таблицю `HousekeeperprocedureVariant` з таблицею `Housekeeperprocedure` через зв'язок один до багатьох: один варіант процедури з ведення домашнього господарства працівником стосується тільки однієї конкретної процедури з ведення домашнього господарства працівником, а кожна така процедура у виконанні певного працівника може мати потенційно багато варіантів реалізації.

Дані замовлення для ведення домашнього господарства в будинку зберігаються в таблиці `KeepingOrder`, а характеристики цього замовлення представлені наступним чином:

- `id` – ідентифікатор замовлення;
- `housekeeper` – працівник, роботу якого замовлено для виконання;
- `house` – будинок, в якому потрібно виконати цю роботу;
- `need_at` – час, починаючи з якого потрібно виконувати цю роботу;
- `ended_at` – час, до якого мають бути завершені роботи;
- `created_at` – час створення замовлення;
- `specification` – особливості замовлення;
- `price` – сумарна вартість замовлення;
- `state` – стан замовлення.

Поле `housekeeper` з'єднує таблицю `Housekeeper` з таблицею `KeepingOrder` через зв'язок один до багатьох: одне замовлення може виконувати тільки одним працівником, а один працівник може мати багато замовлень щодо своєї роботи.

Поле `house` з'єднує таблицю `House` з таблицею `KeepingOrder` через зв'язок один до багатьох: одне замовлення стосується тільки одного будинку, але за весь час один будинок може обслуговуватися в різних замовленнях.

Дані процедури в замовленні зберігаються в таблиці `OrderProcedure`, а характеристики цієї складової замовлення представлені наступним чином:

- `id` – ідентифікатор складової замовлення;
- `keepingorder` – замовлення, до якого належить ця складова;

- `housekeeperprocedurevariant` – варіант процедури з ведення домашнього господарства, яку виконує деякий працівник;
- `specification` – особливі вимоги до виконання цієї процедури;
- `price` – вартість процедури в замовленні;
- `duration` – тривалість виконання процедури.

Поле `keepingorder` з'єднує таблицю `OrderProcedure` з таблицею `KeepingOrder` через зв'язок один до багатьох: одне замовлення складається з багатьох складових, якими є окремі процедури, але кожна така складова належить тільки до одного замовлення.

Поле `housekeeperprocedurevariant` з'єднує таблицю `OrderProcedure` з таблицею `HousekeeperprocedureVariant` через зв'язок один до багатьох: одна складова замовлення визначається одним варіантом процедури з ведення домашнього господарства, яку виконує деякий працівник, а кожен такий варіант у виконанні певного працівника може декілька разів замовлятися в різних замовленнях для виконання.

Дані повторюваної процедури зберігаються в таблиці `RepeatProcedure`, а характеристики цього повторення представлені наступним чином:

- `id` – ідентифікатор повторюваної процедури;
- `orderprocedure` – процедура з замовлення (складова замовлення), яка використана як основа для повторення;
- `beginning` – час початку повторювання;
- `ending` – час завершення повторювання;
- `interval` – інтервал повторювання процедури (кількість діб, що має пройти для виконання процедури знову);
- `state` – стан повторювання.

Поле `orderprocedure` з'єднує таблицю `RepeatProcedure` з таблицею `OrderProcedure` через зв'язок один до багатьох: одне повторювання може стосуватися тільки однієї процедури в замовленні, а на основі однієї процедури в замовленні може бути побудовано декілька повторювань, що

пояснюється тим, що таке повторювання може встановлюватися в різні періоди часу заново.

Дані особливостей виконання процедури в частині будинку зберігаються в таблиці `OrderProcedurePart`, а характеристики цієї особливості представлені наступним чином:

- `id` – ідентифікатор особливостей виконання процедури в частині будинку;
- `orderprocedure` – процедура в замовленні (складова замовлення);
- `housepart` – частина будинку, в якій процедура має бути виконана, якщо таке розділення виконується.

Таблиця `OrderProcedurePart` є реалізацією зв'язку багато до багатьох між таблицями `OrderProcedure` і `HousePart`: одна процедура в замовленні може реалізовуватися в різних частинах будинку, а в одній частині будинку може виконуватися багато різних процедур з різних замовлень.

Дані розкладу роботи працівника зберігаються в таблиці `HousekeeperTimetable`, а характеристики розкладу представлені наступним чином:

- `id` – ідентифікатор запису розкладу;
- `housekeeper` – працівник;
- `beginning` – час початку роботи в цей день;
- `ending` – час завершення роботи в цей день;
- `weekday` – день тижня, якого стосується цей розклад.

Поле `housekeeper` з'єднує таблицю `HousekeeperTimetable` з таблицею `Housekeeper` через зв'язок один до багатьох: один працівник може визначати розклад на декілька робочих днів, а кожен такий розклад стосується одного працівника.

Дані особливого розкладу роботи працівника на конкретні дати зберігаються в таблиці `HousekeeperSpecTimetable`, а характеристики цього особливого розкладу представлені наступним чином:

- `id` – ідентифікатор особливого розкладу роботи працівника;

- `housekeeper` – працівник, для якого визначено особливий розклад;
- `beginning` – початок роботи в цю дату;
- `ending` – завершення роботи в цю дату;
- `day` – дата застосування такого розкладу.

Поле `housekeeper` з'єднує таблицю `HousekeeperSpecTimetable` з таблицею `Housekeeper` через зв'язок один до багатьох: один працівник може визначати спеціальний розклад на різні дати, а кожен такий розклад стосується одного працівника.

Дані відгуку про процедуру зберігаються в таблиці `ResponseProcedure`, а характеристики відгуку представлені наступним чином:

- `id` – ідентифікатор відгуку про процедуру;
- `owner` – власник, який створив відгук;
- `housekeeperprocedure` – процедура, якої стосується відгук;
- `created_at` – час створення відгуку;
- `score` – оцінка відгуку.

Поле `owner` з'єднує таблицю `ResponseProcedure` з таблицею `Owner` через зв'язок один до багатьох: один власник може мати багато відгуків про процедури, а кожен відгук стосується одного власника.

Поле `housekeeperprocedure` з'єднує таблицю `ResponseProcedure` з таблицею `Housekeeperprocedure` через зв'язок один до багатьох: одна процедура у виконанні працівника може мати багато відгуків, а один відгук стосується тільки однієї конкретної процедури працівника, про яку він написаний.

Дані відгуку про працівника загалом зберігаються в таблиці `ResponseHousekeeper`, а характеристики відгуку представлені наступним чином:

- `id` – ідентифікатор відгуку про працівника;
- `owner` – власник, який створив цей відгук;
- `housekeeper` – працівник, який створив цей відгук;
- `created_at` – час створення відгуку;

– score – оцінка відгуку.

Поле housekeeper з'єднує таблицю ResponseHousekeeper з таблицею Housekeeper через зв'язок один до багатьох: один працівник може мати багато відгуків, а один відгук стосується тільки одного працівника.

Дані запиту від власника до працівника зберігаються в таблиці Ask, а характеристики запиту представлені наступним чином:

- id – ідентифікатор запиту;
- housekeeper – працівник, якому адресовано запит;
- house – будинок, якого стосується запит;
- need_at – час, коли має початися обслуговування;
- created_at – час створення запиту;
- specification – особливості запиту;
- proposed_price – запропонована плата за виконання цієї роботи;
- state – стан запиту.

Поле housekeeper з'єднує таблицю Ask з таблицею Housekeeper через зв'язок один до багатьох: один запит стосується тільки одного працівника, але працівник може мати багато таких запитів.

Поле house з'єднує таблицю Ask з таблицею House через зв'язок один до багатьох: один будинок може мати багато запитів про його обслуговування, а один запит стосується тільки одного конкретного будинку.

Дані складової запиту від власника до працівника зберігаються в таблиці AskProcedure, а характеристики цієї складової представлені наступним чином:

- id – ідентифікатор запиту від власника до працівника;
- ask – запит, якого стосується ця процедура як складова;
- housekeeperprocedurevariant – варіант процедури у виконанні працівника;
- specification – особливості виконання цієї процедури;
- proposed_price – запропонована плата за виконання цієї процедури.

Створена схема структури БД показана на рис. 2.4.

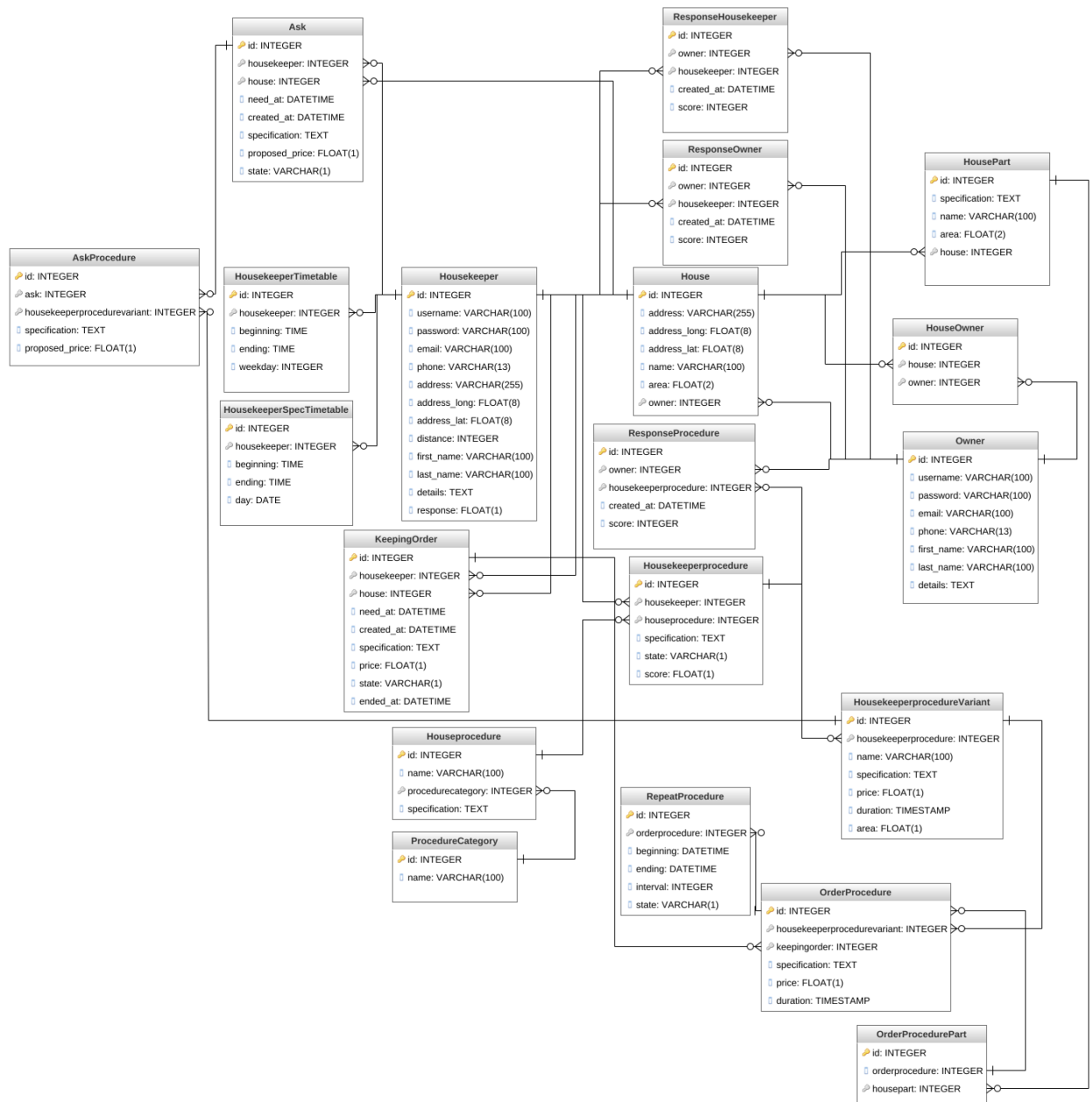


Рисунок 2.4 – Схема структури БД

Поле `ask` з'єднує таблицю `AskProcedure` з таблицею `Ask` через зв'язок один до багатьох: один запит може мати багато процедур у своєму складі, але кожний запис про процедуру стосується одного запиту.

Поле `housekeeperprocedurevariant` з'єднує таблицю `AskProcedure` з таблицею `HousekeeperprocedureVariant` через зв'язок один до багатьох: один варіант процедури у виконанні працівника може бути використаний як складова в різних запитах, але кожен запис стосується тільки одного варіанту процедури у виконанні деякого працівника.

Дані про можливість роботи з замовленнями по будинку або апартаментам зберігаються в таблиці HouseOwner, а характеристики цієї можливості представлені наступним чином:

- id – ідентифікатор зв'язування власника як помічника з будинком в програмі;
- house – будинок, для якого виконується зв'язування;
- owner – власник, який буде допомагати з керуванням справами по цьому будинку.

Таблиця HouseOwner є реалізацією зв'язку багато до багатьох між таблицями House і Owner: один будинок може керуватися різними власниками, а один власник може мати доступ до різних будинків.

Дані про відгук про власника житла зберігаються в таблиці ResponseOwner, а характеристики цієї можливості представлені наступним чином:

- id – ідентифікатор відгуку;
- owner – власник житла, про якого відгук;
- housekeeper – робітник, який створив відгук;
- created_at – час створення відгуку;
- score – оцінка відгуку.

Поле housekeeper з'єднує таблицю ResponseOwner з таблицею Housekeeper через зв'язок один до багатьох: один робітник може мати багато відгуків про власників, а кожен відгук створений конкретним робітником.

Поле owner з'єднує таблицю ResponseOwner з таблицею Owner через зв'язок один до багатьох: один власник житла може мати багато відгуків про себе, але кожен відгук стосується одного конкретного власника житла.

2.4 Висновки за розділом 2

У цьому розділі виконано моделювання сценаріїв роботи власника житла, робітника та адміністратора з програмою, за результатами створено

діаграми використання. Виконано порівняння фреймворку веброзробки Django і платформи Node.JS для розробки програми, в підсумку обрано Django. Реалізовано проектування структури роботи з даними через БД.

3 РОЗРОБКА ПРОГРАМИ

3.1 Алгоритм основи функціонування програми

Основні функції програми у відповідності з функціональними вимогами до програми є достатньо типовими, оскільки Django має достатню кількість засобів для їх виконання з встановленням основних налаштувань. Звичайно, що в деяких з них реалізуються додаткові дії, але загалом це достатньо типові процеси. У той же час процес пошуку варіантів процедур з ведення домашнього господарства є достатньо особливим. Тому саме його було визначено окремо, використовуючи його під час реалізації для забезпечення відповідного алгоритму. Подібним чином побудовано і процес пошуку варіантів процедур для створення запитів від власників, але при цьому враховується можливість більшої відстані, яку зазначить власник, а не відстань встановлена самим робітником, а також не враховується робочий час робітника.

Процес пошуку процедур за варіантами виконання (рис. 3.1) розпочинається з формування базового переліку варіант процедур за вказаною назвою, потім він зменшується за рахунком фільтрування за встановленою категорією, потім за мінімальною оцінкою, яку повинні перевищувати всі варіанти, що залишаються, та за площею приміщення, яка має бути доступною у відповідному варіанті процедури.

Далі реалізуються більш складні процеси, кожен з яких передбачає певний порядок окремих дій всередині себе. Спочатку виконується перевірка відстані між адресою робітника і встановленою ним максимальною відстанню від неї та адресою пошуку власником, де знаходиться його будинок. Далі перевіряється, чи не має робітник на цей час уже замовлення, після цього перевіряється графік роботи працівника на цей день. А в підсумку формується і впорядковується перелік вже не варіантів, а самих процедур. Початково використовується саме перелік варіантів, адже в них встановлено вартість, площу приміщення.



Рисунок 3.1 – Схема виконання процесу пошуку процедур за варіантами

3.2 Опис реалізованих класів програми

На структуру і реалізацію програми безпосередньо вплинув вибір технологій розробки. Оскільки класи моделей відповідають фактично структурі таблиць БД, що детально була розглянута в розділі 2, а шаблони фактично є розміткою вебсторінок, то в подальшому тут приділено основну увагу саме класам представлень для роз'яснення того, як вони забезпечують ті функції, які були в розділі 2 визначені.

Спочатку детально розглянемо те, як було реалізовано пошук варіантів робіт з ведення домашнього господарства. Для цього було створено клас представлення `SearchView`. Він побудований на основі моделі `Housekeeperprocedure`, а шаблоном для роботи над пошуком є `search.html`.

Оскільки цей клас побудовано на основі стандартного класу `ListView`, що дозволяє працювати з переліками даних, то основою реалізації його функціональності є стандартний метод `get_queryset`, який було перевизначено наступним чином.

Метод спочатку розпізнає параметри, передані в нього, що відносяться до фільтрів пошуку. Ці фільтри охоплюють:

- максимальну вартість робіт, що відповідає параметру `price`;
- мінімальну оцінку за таку процедуру, що відповідає параметру `score`;
- назву процедури або її частину, що відповідає параметру `name`;
- номер категорії процедури, якщо її було обрано з переліку користувачем, що відповідає параметру `category`;
- площа приміщення (фактично максимальна площа), що відповідає параметру `area`;
- координати адреси, де необхідно виконати роботу, включаючи довготу (відповідає параметру `address_long`) та широту (відповідає параметру `address_lat`);

– дату і час, коли найраніше потрібно почати роботи, що відповідає параметру `startdate`;

– дату і час, коли найпізніше потрібно завершити роботи, що відповідає параметру `enddate`.

Далі відбувається розбір всіх цих параметрів задля виконання безпосередньо самого пошуку.

Для цього спочатку у відповідності зі схемою, показаною на рис. 3.1, виконується відбір тих варіантів, в імені процедур яких міститься фраза, яку було в якості назви задано власником, а далі перевіряється також категорія процедур, чи відповідає вона у відсіяного набору встановленому номеру категорії, якщо такий номер було встановлено.

Далі реалізується відсіювання варіантів процедур, які мають середню оцінку нижче заданої, всі вони відкидаються, а ті, що мають оцінку, яка рівна або вище заданої, залишаються.

Далі множина результатів пошуку звужується на основі відкидання тих варіантів, які передбачають роботи на меншій площі, ніж задана під час пошуку. Одразу після цього реалізується відсіювання тих варіантів, вартість яких перевищує встановлену власником межу:

```
variants =
HousekeeperprocedureVariant.objects.filter(housekeeperprocedure__house
procedure__name__icontains = name)
variants =
variants.filter(housekeeperprocedure__houseprocedure__procedurecategor
y__id = category)
variants = variants.filter(housekeeperprocedure__score__gte =
score)
variants = variants.filter(area__gte = area)
variants = variants.filter(price__lte = price)
```

Далі відбувається перевірка віддаленості робітників, які готові виконувати роботи, від адреси, за якою власник хоче такі роботи виконати.

Під час такої перевірки обчислюється дистанція між цими двома адресами за широтою та за довготою. Обчислена відстань порівнюється з допустимою відстанню, встановленою в профілі робітника. Якщо обидва виміри відстані відповідають встановленій межі, то такий варіант процедури залишається в результатах пошуку, а якщо ні, то він видаляється:

```
proc_variants = []
for variant in variants:
    hkeeper = variant.housekeeperprocedure.housekeeper
    addr_long = hkeeper.address_long
    addr_lat = hkeeper.address_lat
    km_long = haversine.haversine(address_long, addr_long)
    km_lat = haversine.haversine(address_lat, addr_lat)
    radius = hkeeper.distance
    if km_long <= radius and km_lat <= radius:
        proc_variants.append(variant)
```

Після цього реалізується окремий етап пошуку, який передбачає перевірку того, чи має робітник на цей час вже існуюче замовлення. Такі замовлення мають стосуватися будь-якої з процедур цього робітника, а також перетинатися за часом виконання з заданим в параметрах пошуку часом. Якщо існуючі замовлення перетинаються, то такий варіант процедури вилучається з результатів пошуку.

Перевірка перетинання інтервалів реалізується за допомогою чотирьох окремих перевірок. Це перевірка перетинання з лівого краю:

```
op1 =
OrderProcedure.objects.filter(housekeeperprocedurevariant__housekeeper
procedure__housekeeper = hkeeper, keepingorder__need_at__lte =
startdate, keepingorder__ended_at__gte = startdate,
keepingorder__ended_at__lte = enddate)
```

Якщо ця перевірка не призвела до отримання результату, то перевіряється перетин з правого краю:

```

    if not op1:
        op2 =
OrderProcedure.objects.filter(housekeeperprocedurevariant__housekeeper
procedure__housekeeper = hkeeper, keepingorder__need_at__gte =
startdate, keepingorder__ended_at__gte = startdate,
keepingorder__ended_at__gte = enddate)

```

Якщо не відпрацювали обидві перевірки, то перевіряється, чи не міститься один варіант в іншому:

```

    if not op2:
        op3 =
OrderProcedure.objects.filter(housekeeperprocedurevariant__housekeeper
procedure__housekeeper = hkeeper, keepingorder__need_at__gte =
startdate, keepingorder__need_at__lte = enddate,
keepingorder__ended_at__gte = startdate, keepingorder__ended_at__lte =
enddate)

```

І в останньому випадку перевіряється зворотнє входження одного інтервалу в інший:

```

    if not op3:
        op4 =
OrderProcedure.objects.filter(housekeeperprocedurevariant__housekeeper
procedure__housekeeper = hkeeper, keepingorder__need_at__lte =
startdate, keepingorder__ended_at__gte = enddate)
        if not op4:
            proc_variants.append(variant)

```

Після цього реалізується окремий етап, що передбачає співставлення графіку роботи працівника. Для цього спочатку перевіряється, чи не відноситься день, в який потрібно виконати роботи, до днів з особливим графіком. Якщо відноситься, то виконується співставлення часу роботи в цей день з часом, який вимагається в замовленні. Якщо це допустимий варіант, то такий варіант пошуку залишається в результатах, якщо ні, то вилючається.

Якщо спеціального розкладу на цей день не заплановано, то перевіряється розклад за днем тижня. Якщо час пошуку не потрапляє в робочий час для такого дня тижня, то такий варіант також з пошуку вилучається.

Після відсіювання всіх зайвих варіантів виконується останній крок – на основі всіх відібраних варіантів процедур створюється перелік процедур, до яких ці варіанти належать. Створений таким чином перелік повертається для заповнення даними відповідного шаблону.

Клас `HouseCreateView` реалізує визначення будинку у власному керуванні користувачем, який авторизувався в системі як власник.

Клас `HousePartCreateView` реалізує визначення частини будинку (кімнати, саду) до самого будинку, який вже знаходиться у власному керуванні користувачем, який авторизувався в системі як власник, і номер будинку переданий через параметр `house`.

Клас `HouseUpdateView` реалізує редагування даних будинку, який знаходиться в керуванні власника. Під час редагування використовується параметр `owners`, який визначає перелік користувачів, яким також доступна інформація про всі замовлення процедур з ведення домашнього господарства за цим будинком. Цей перелік кожного разу перевіряється, і якщо виявляється, що такого помічника немає для даного будинку, то він вноситься, а якщо виявляється, що він зник з переліку в версій, переданій в параметрі, то відповідно вилучається:

```
for owner in sep_owners:
    owner = Owner.objects.get(id = owner)
    ready_owners.append(owner)
    house_owner = HouseOwner.objects.get (house = house, owner =
owner)
    if not house_owner:
        HouseOwner.objects.create(house = house, owner =
owner)
for owner in start_owners:
```

```
if owner not in ready_owners:  
    HouseOwner.objects.filter(id = owner.id).delete()
```

Клас `HousePartU pateView` реалізує редагування даних заданої частини будинку, змінюючи при цьому за необхідності назву, площу, особливості.

Клас `HousePartDeleteView` реалізує підтримку видалення відповідної частини будинку з керування. Клас `HouseDeleteView` реалізує відповідні дії для будинку.

Клас `HouseDetailView` дозволяє отримати повну інформацію про будинок власнику з додаванням до контексту даних одразу переліку всіх частин цього будинку та всіх замовлень, які стосуються цього будинку, а також всіх запитів, які були сформовані для цього будинку.

За аналогічними принципами створені і інші класи застосунку, забезпечуючи відповідне внесення, редагування і відображення даних.

3.3 Висновки за розділом 3

У цьому розділі розглянуто порядок роботи програми на прикладі основного процесу – процесу пошуку процедур з ведення домашнього господарства за варіантами, окремо приведено схему виконання даного процесу. Описано класи, які було створено під час реалізації програми, з визначенням принципів і деталей їх створення.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Призначенням програми є надання можливості знайти працівників для ведення домашнього господарства для зменшення витрат власних часових ресурсів у результаті. Програма також експлуатується самими робітниками для пошуку замовників та формування власного розкладу виконання робіт.

4.2 Умови виконання програми

Мінімальний рівень технічних засобів для забезпечення роботи програмної системи включає процесор не менш ніж 4 ядра і частотою 2,9 ГГц, оперативну пам'ять у 8 Гб, жорсткий диск у 1 Тб.

4.3 Експлуатація програми

Експлуатація програми трьома типами користувачів відбувається різним способом.

Після звернення до програми через веббраузер відкривається сторінка пошуку варіантів процедур для ведення домашнього господарства, яка буде розглянута нижче. Дана сторінка окрім всього іншого має пункт меню «Авторизація», який дозволяє перейти до сторінки авторизації користувача в програмі. На цій же сторінці доступна кнопка «Реєстрація» для реєстрації у випадку відсутності облікового запису та посилання «Я власник житла», після натискання на яке відкривається сторінка авторизації для власників житла. У випадку авторизації таким чином користувач буде працювати з програмою саме як власник житла. Це буде розглянуто нижче.

Адміністратор для входу в систему має використовувати свої облікові дані, а авторизація йому доступна через додавання розділу admin до повної адреси вебсайту, за якою розташовано програмне забезпечення.

Робітник взаємодіє з програмою через меню, з якого йому доступні розділ «Процедури» для відображення всіх процедур, які він надає, розділ «Замовлення» для роботи з адресованими йому замовленнями процедур з ведення домашнього господарства, розділ «Запити» з відображенням запитів від власників житла, які пропонують виконати роботу за адресами, що віддалені більше, ніж дозволяє робітник, або в час, який не встановлений як робочий у робітника, за підвищену плату. У випадку підтвердження запиту він стає замовленням, а всі відповідні дані переносяться в замовлення. Робітник може також скасувати такий запит. З розділу «Графік» робітнику доступна таблиця з тривалістю робочого дня в кожен день тижня, звідки він може відредагувати ці дані, а також за допомогою кнопки «Виключення» перейти до внесення окремих дат, в які графік роботи може бути іншим.

З розділу «Моя сторінка» є можливість переглянути основні дані про себе, що відображаються власникам житла, а також є можливість перейти до редагування даних власного профілю, зокрема і встановити власну адресу, встановити дистанцію, на яку готовий виїжджати працювати робітник.

У випадку переходу на сторінку однієї з власних процедур робітнику відкривається інформація (рис. 4.1) про назву процедури, про власний її опис, про виставлену клієнтами (власниками житла) середню оцінку. За допомогою кнопки «Редагувати» можна перейти до редагування відповідної процедури. За допомогою кнопки «Відгуки» можна за необхідності відобразити всі відгуки, що стосуються саме цієї процедури.

На сторінці відображається перелік всіх варіантів цієї процедури. За кожним варіантом відображається іконка, яка дозволяє перейти до редагування цього варіанта, а вище знаходиться кнопка «Додати», яка дозволяє додати новий варіант процедури. Кнопка додавання самої процедури доступна зі сторінки перегляду переліку власних процедур.

За кожним варіантом показується назва, опис, максимальна площа приміщення, для якої доступний цей варіант, вартість його та приблизна тривалість.

HOUSEKEEPER

Процедури

Замовлення

Запити

Графік

Моя Сторінка

Вийти

— ВЛАСНА ПРОЦЕДУРА

Загальне прибирання будинку

Редагувати

Категорія: Прибирання

Особливості:

Використовується професійне обладнання. Надається стандартний набір дій. Доступні також окремі дії за потребою замість пакету загалом, але їх потрібно обирати з моєї сторінки окремо в такому випадку.

Оцінка клієнтів: 4.5

Відгуки

Варіанти процедури

Додати

Поверхнєве прибирання

Прибирання пилу та підлоги пилососом

Максимальна площа: 100 кв.м

Приблизна тривалість: 1:30

€600

Стандартне прибирання

Прибирання пилу і миття поверхонь, прибирання підлоги пилососом та миття підлоги

Максимальна площа: 100 кв.м

Приблизна тривалість: 2:30

€1500

Детальне прибирання

Прибирання пилу всюди, миття всіх поверхонь, в тому числі усунення тривалих забруднень, миття вікон

Максимальна площа: 100 кв.м

Приблизна тривалість: 4:00

€3000

Рисунок 4.1 – Власна процедура в інтерфейсі робітника

Розділ «Замовлення» для робітника організований таким чином, щоб він фактично міг з нього і керувати замовленнями, які йому адресовані, і переглядати одразу графік робіт (рис. 4.2). Для цього користувач має задати спочатку дату з календаря, потім натиснути на кнопку «Показати». Весь графік робіт на день включає всі відповідні замовлення. За кожним замовленням показується час його виконання, фотографія власника, за допомогою якої можна перейти до сторінки цього користувача, де можна побачити його оцінку, відгуки про нього тощо. При натисканні на час виконання робіт відкривається повна сторінка замовлення з усіма деталями, в тому числі планом прибирання кімнат, якщо це було встановлено.

Робітник може скасувати замовлення, може за допомогою кнопки «Взятись» зберегти замовлення для свого виконання, або може відмовитись від замовлення. Ці кнопки відображаються тільки для замовлень, які ще не були підтверджені. Для таких замовлень також показується кнопка «Редагувати» для редагування доступних процедур.

HOUSEKEEPER
Процедури
Замовлення
Запити
Графік
Моя Сторінка
Вийти

План і пропозиції робіт

15.06.2022

Показати



15.06.22 9:00 – 11:00
Миття вікон
Примітка: двоповерховий будинок, вимити всі вікна

Скасувати

€2500



15.06.22 12:00 – 15:00
Миття кухонної техніки
Чищення застарілих забруднень

Скасувати

€3000

Рисунок 4.2 – План робіт в інтерфейсі робітника

Власник житла має окремий інтерфейс, який включає відповідний набір пунктів меню. Звичайно, що для власника першочерговим є пошук необхідних йому процедур. Він доступний через пункт меню «Знайти». На сторінці одразу потрібно визначити всі параметри пошуку, розглянуті в розділі 3, а також потрібно задати спосіб відображення результатів. Може

відображатися або перелік всіх процедур від різних робітників, або потрібно обрати варіант відображення за робітниками, тоді відобразяться робітники, які можуть надати за вказаними фільтрами відповідні процедури з ведення домашнього господарства, а вже за ними відповідні доступні процедури. На рис. 4.3 показаний приклад такого пошуку за категорією «Сад».

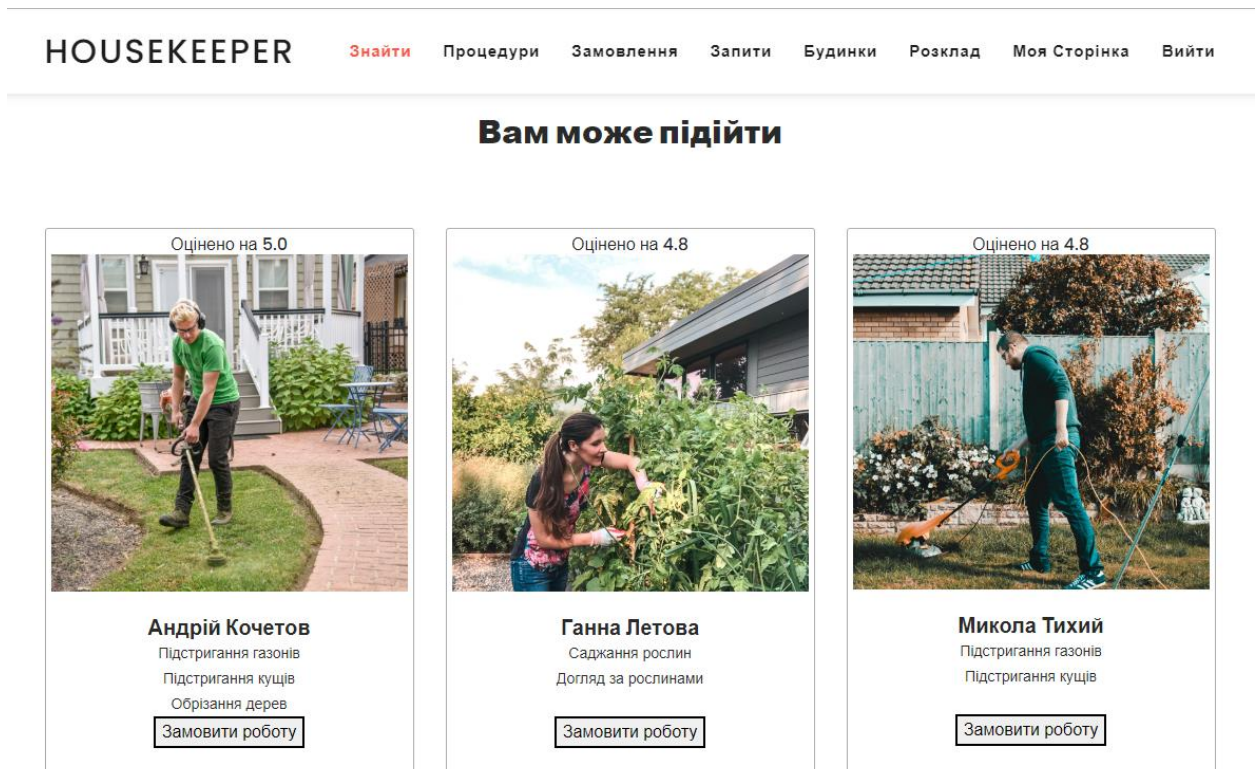


Рисунок 4.3 – Пошук потрібних власнику житла процедур

За кожним робітником надається його середня оцінка, ім'я та прізвище, перелік не всіх його процедур, а тільки тих, які доступні за параметрами пошуку. Кнопка «Замовити роботу» дозволяє перейти до замовлення робіт від цього робітника. Натисканням на ім'я та прізвище можна перейти до сторінки цього робітника. Замовлення можна виконати і звідти. Але тільки звідти можна створити запити.

Власник житла взаємодіє з програмою через меню, з якого йому доступні розділ «Процедури» для відображення переліку категорій процедур, звідки можна перейти до сторінок відповідних процедур для ознайомлення з ними, розділ «Замовлення» для роботи з власними замовленнями процедур з

ведення домашнього господарства, розділ «Запити» дозволяє працювати з власними запитами, розділ «Будинки» дозволяє керувати власними будинками, додавати інших користувачів, яким будуть відображатися замовлення за відповідною адресою, розділ «Розклад» дозволяє встановити процедури, які власник житла бажає виконувати повторно, та вказати період такого повторення.

4.4 Тестування програми

Тестування програми було реалізовано перевіркою роботи програми з відповідними функціями. Організовано тестування було після завершення реалізації програми. Програма відпрацювала коректно з визначеними в функціональних вимогах функціями.

ВИСНОВКИ

Досліджуючи предметну область, було виявлено, що з розвитком економіки люди менше часу витрачають на ведення домашнього господарства, тому з'являється потреба в тих працівниках, які зможуть таку роботу виконувати. Саме тому метою роботи було визначено розробити програмне забезпечення пошуку робітників для ведення домашнього господарства для підтримки планування цих процесів за допомогою зовнішніх працівників.

Для досягнення цієї мети було виконано аналіз проблеми, результатом якого стали сформульовані у вигляді набору сценаріїв роботи користувачів функціональні вимоги до програми разом з іншими вимогами до програми, та структура БД. Розробку було виконано мовою Python за допомогою фреймворку веброзробки Django. Усі ці результати були використані під час структуризації програми, яка з врахуванням прийнятих рішень щодо засобів підтримки розробки була побудована на основі шаблону MVT, при безпосередній реалізації класів програми у відповідності з цим обраним шаблоном і визначеною структурою БД, яка була впроваджена на основі використання підходу «код спочатку», тобто автоматично згенерована на основі розроблених спочатку класів моделей програми.

Виконання роботи дозволило створити програму, яка забезпечує підтримку людей, які знаходяться в пошуку робітників для ведення домашнього господарства у себе вдома, в будинках, які їм належать або ними керуються, або які планують таку роботу в подальшому, та підтримку людей, які готові виконувати роботу щодо ведення домашнього господарства в будинках інших людей. Програма є простою у використанні і достатньо простою в налаштуванні на сервері за рахунок стандартного налаштування, що визначається для застосунків Django.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Housekeeper job description. Workable [Electronic resource]. – Access mode : <https://resources.workable.com/housekeeper-job-description>.
2. Housekeeper Job Description [Electronic resource]. – Access mode : <https://www.betterteam.com/housekeeper-job-description>.
3. Aguirre, S. What to Know Before You Hire a Housekeeper [Electronic resource] / S. Aguirre. – Access mode : <https://www.thespruce.com/before-you-hire-a-housekeeper-1900773>.
4. Mogelonsky, L. Top issues and solutions for your housekeeping department [Electronic resource] / L. Mogelonsky. – Access mode : <https://www.todayshotelier.com/2018/06/11/top-issues-and-solutions-for-your-housekeeping-department/>.
5. Maidsapp [Electronic resource]. – Access mode : <https://www.maidsapp.com/>.
6. Care.com [Electronic resource]. – Access mode : <https://www.care.com/>.
7. Ramalho, L, Fluent Python [Text] : Clear, Concise, and Effective Programming / L. Ramalho. – Newton, Massachusetts : O'Reilly Media, 2022. – 1012 p.
8. The web framework for perfectionists with deadlines. Django [Electronic resource]. – Access mode : <https://www.djangoproject.com/>.
9. Django MVT Architecture – AskPython [Electronic resource]. – Access mode : <https://www.askpython.com/django/django-mvt-architecture>.

ДОДАТОК А
Технічне завдання

Вступ

Люди потребують допомоги з власним домашнім господарством доволі часто, хоча це звичайно залежить також від графіку і способу життя людини. Сучасний світ доволі активний, тому велика кількість людей потребує допомоги в домашніх справах, при цьому допомоги не членів сім'ї, а допомоги працівників ззовні. Для підтримки такої потреби розробляється програмне забезпечення.

A.1 Підстави для розробки

Розробка в межах дипломної кваліфікаційної роботи реалізується на підставі наказу № 102 від 27 квітня 2022 р. за Національним університетом «Запорізька політехніка» за темою «Програмне забезпечення пошуку робітників для ведення домашнього господарства».

A.2 Призначення розробки

Призначенням розробки є надання можливості знайти працівників для ведення домашнього господарства для зменшення витрат власних часових ресурсів у результаті.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Функціональні характеристики, які повинна мати програма, охоплюють:

- реєстрація в якості працівника і в якості власника;
- авторизація;
- змінювання даних власного профілю;

- визначення переліку будинків, якими керує власник;
- визначення складових будинку;
- перегляд власних будинків;
- визначення помічників у керуванні власними будинками;
- додавання, редагування категорій процедур;
- додавання, редагування процедур;
- перегляд існуючих категорій процедур;
- перегляд існуючих процедур за категорією;
- пошук процедур;
- пошук варіантів обслуговування будинку;
- визначення процедур, які виконує працівник;
- перегляд переліку процедур, які виконує працівник;
- визначення варіантів виконання процедур працівником;
- замовлення процедур;
- скасування замовлення;
- взяття замовлення на виконання;
- підтвердження виконання замовлення;
- змінювання складу замовлення;
- перегляд переліку власних замовлень;
- перегляд переліку адресованих замовлень;
- створення повторюваного обслуговування;
- визначення плану робіт за частинками будинку в замовленні;
- визначення власного графіку роботи працівника;
- перегляд графіку робіт працівника;
- перегляд доступного часу у працівника;
- створення відгуку про працівника;
- створення відгуку про процедуру;
- перегляд переліку відгуків про працівника;
- перегляд переліку відгуків за процедурою, яку виконує працівник;
- створення запиту;

- прийняття запиту на виконання;
- скасування запиту;
- перегляд переліку адресованих запитів;
- перегляд переліку власних запитів;
- керування доступом користувачів.

A.3.2 Умови експлуатації

Експлуатація програми має забезпечуватися з одного боку звичайними користувачами, які працюють з даними, наповнюють їх, а з іншого боку – адміністраторами системи та БД.

A.3.3 Вимоги до складу та параметрів технічних засобів

Мінімальний рівень технічних засобів для забезпечення роботи програмної системи включає процесор не менш ніж 4 ядра і частотою 2,9 ГГц, оперативну пам'ять у 8 Гб, жорсткий диск у 1 Тб.

A.4 Порядок контролю та приймання

Для контролю виконання роботи використовується календарний план, а приймання роботи виконується під час проведення процедури захисту роботи перед екзаменаційною комісією.

ДОДАТОК Б
Текст програми

Б.1 Текст файла houseview.py

```

from django.contrib.auth.mixins import LoginRequiredMixin
from django.urls import reverse_lazy
from django.views.generic.edit import CreateView
from django.views.generic import ListView
from django.views.generic import UpdateView
from django.views.generic.detail import DetailView
from django.views.generic.edit import DeleteView
from .models import House
from .models import HousePart
from .models import HouseOwner
from .models import Owner

class HouseCreateView (LoginRequiredMixin, CreateView):
    model = House
    fields = ['address', 'address_long', 'address_lat', 'area',
'name']
    template_name = 'add_your_house.html'

    def get_success_url(self):
        return reverse_lazy('houseview', kwargs={'pk':
self.object.pk})

    def form_valid(self, form):
        me = self.request.user
        owner = Owner.objects.get(id = me.id)
        self.owner = owner
        return super().form_valid(form)

class HousePartCreateView (LoginRequiredMixin, CreateView):
    model = House
    fields = ['name', 'area', 'specification']
    template_name = 'add_your_house_part.html'

    def get_success_url(self):

```

```

        return reverse_lazy('houseview', kwargs={'pk':
self.object.house.pk})

    def get_context_data(self, **kwargs):
        house = get_object_or_404(House,
id=self.kwargs['house'])
        kwargs['house'] = house
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        house = get_object_or_404(House,
id=self.kwargs['house'])
        self.house = house
        return super().form_valid(form)

class HouseUpdateView (LoginRequiredMixin, UpdateView):
    model = House
    fields = ['address', 'address_long', 'address_lat', 'area',
'name']
    template_name = 'edit_your_house.html'

    def get_success_url(self):
        return reverse_lazy('houseview', kwargs={'pk':
self.object.pk})

    def form_valid(self, form):
        housef = get_object_or_404(House,
id=self.kwargs['house'])
        house = self.object.pk
        if house:
            self.house = housef
            house = housef
        owners = self.kwargs['owners']
        sep_owners = owners.split(',')
        ready_owners = []
        start_owners = self.owners
        for owner in sep_owners:
            owner = Owner.objects.get(id = owner)
            ready_owners.append(owner)

```

```

        house_owner = HouseOwner.objects.get (house =
house, owner = owner)
        if not house_owner:
            HouseOwner.objects.create(house = house,
owner = owner)
        for owner in start_owners:
            if owner not in ready_owners:
                HouseOwner.objects.filter(id =
owner.id).delete()
        return super().form_valid(form)

class HousePartUpdateView (LoginRequiredMixin, UpdateView):
    model = HousePart
    fields = ['name', 'area', 'specification']
    template_name = 'edit_your_house_part.html'

    def get_success_url(self):
        return reverse_lazy('houseview', kwargs={'pk':
self.object.pk})

class HousePartDeleteView (DeleteView):
    model = HousePart

class HouseDeleteView (DeleteView):
    model = House

class HouseDetailView (DetailView):
    model = House
    template_name = 'house.html'

    def get_context_data(self, **kwargs):
        data_about_house = super().get_context_data(**kwargs)
        rooms = HousePart.objects.filter(house =
self.object.pk)
        rooms = rooms.order_by('name')
        data_about_house ['parts'] = rooms
        orders = KeepingOrder.objects.filter(house =
self.object.pk)
        orders = orders.order_by('-need_at')
```

```

        data_about_house ['orders'] = orders
        asks = Ask.objects.filter(house = self.object.pk)
        asks = asks.order_by('-need_at')
        data_about_house ['asks'] = asks
        return data_about_house

class HousePartDetailView (DetailView):
    model = HousePart
    template_name = 'housepart.html'

    def get_context_data(self, **kwargs):
        data_about_house = super().get_context_data(**kwargs)
        part = HousePart.objects.get(id = self.object.pk)
        procedures = part.orderprocedureparts
        orders = []
        for procedure in procedures:
            keep_order =
procedure.orderprocedure.keepingorder
            if keep_order not in orders:
                orders.append(keep_order)
        orders = orders.order_by('-need_at')
        data_about_house ['orders'] = orders
        return data_about_house

class AuthHousesView (ListView):
    model = House
    template_name = 'your_houses.html'

    def get_queryset(self):
        me = self.request.user
        owner = Owner.objects.get(id = me.id)
        houses = House.objects.filter(owner = owner)
        houses_add = HouseOwner.objects.filter(owner = owner)
        for house in houses_add:
            next_house = house.house
            houses.append(next_house)
        return houses

```

Б.2 Текст файла `procedureview.py`

```

from django.contrib.auth.mixins import LoginRequiredMixin
from django.urls import reverse_lazy
from django.views.generic.edit import CreateView
from django.views.generic import ListView
from django.views.generic import UpdateView
from django.views.generic.detail import DetailView
from django.views.generic.edit import DeleteView
import datetime
import haversine
from .models import Houseprocedure
from .models import Housekeeperprocedure
from .models import HousekeeperprocedureVariant
from .models import ProcedureCategory
from .models import OrderProcedure
from .models import KeepingOrder
from .models import HousekeeperTimetable
from .models import HousekeeperSpecTimetable
from datetime import date

class HouseprocedureCreateView (LoginRequiredMixin, CreateView):
    model = Houseprocedure
    fields = ['specification', 'name']
    template_name = 'add_houseprocedure.html'

    def get_success_url(self):
        return reverse_lazy('houseprocedureview',
kwargs={'pk': self.object.pk})

    def get_context_data(self, **kwargs):
        proc_categ = get_object_or_404(ProcedureCategory,
id=self.kwargs['category'])
        kwargs['category'] = proc_categ
        return super().get_context_data(**kwargs)

```

```

    def form_valid(self, form):
        proc_categ = get_object_or_404(ProcedureCategory,
id=self.kwargs['category'])
        self.procedurecategory = proc_categ
        return super().form_valid(form)

```

```

class ProcedureCategoryCreateView (LoginRequiredMixin,
CreateView):

```

```

    model = ProcedureCategory
    fields = ['name']
    template_name = 'add_procedure_category.html'

```

```

    def get_success_url(self):
        return reverse_lazy('categoriesview')

```

```

class HousekeeperprocedureCreateView (LoginRequiredMixin,
CreateView):

```

```

    model = Housekeeperprocedure
    fields = ['specification', 'state']
    template_name = 'add_housekeeper_procedure.html'

```

```

    def get_success_url(self):
        return reverse_lazy('housekeeperproceduresview',
kwargs={'pk': self.object.housekeeper.pk})

```

```

    def get_context_data(self, **kwargs):
        procedure = get_object_or_404(Houseprocedure,
id=self.kwargs['procedure'])
        kwargs['procedure'] = procedure
        return super().get_context_data(**kwargs)

```

```

    def form_valid(self, form):
        procedure = get_object_or_404(Houseprocedure,
id=self.kwargs['procedure'])
        self.houseprocedure = procedure
        hkeeper = self.request.user
        housekeeper = Housekeeper.objects.get(id = hkeeper)
        self.housekeeper = housekeeper
        return super().form_valid(form)

```

```

class HousekeeperprocedureVariantCreateView (LoginRequiredMixin,
CreateView):
    model = HousekeeperprocedureVariant
    fields = ['name', 'specification', 'price', 'duration',
'area']
    template_name = 'add_housekeeper_procedure_variant.html'

    def get_success_url(self):
        return reverse_lazy('housekeeperprocedurereview',
kwargs={'pk': self.object.housekeeperprocedure.pk})

    def get_context_data(self, **kwargs):
        housekeeperprocedure =
get_object_or_404(Housekeeperprocedure, id=self.kwargs['procedure'])
        kwargs['procedure'] = housekeeperprocedure
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        housekeeperprocedure =
get_object_or_404(Housekeeperprocedure, id=self.kwargs['procedure'])
        self.housekeeperprocedure = housekeeperprocedure
        return super().form_valid(form)

class ProcedureCategoryU pateView (LoginRequiredMixin,
UpdateView):
    model = ProcedureCategory
    fields = ['name']
    template_name = 'edit_category.html'

    def get_success_url(self):
        return reverse_lazy('categoriesview')

class HouseprocedureU pateView (LoginRequiredMixin, UpdateView):
    model = Houseprocedure
    fields = ['specification', 'name']
    template_name = 'edit_houseprocedure.html'

```

```

    def get_success_url(self):
        return reverse_lazy('houseprocedurereview',
                               kwargs={'pk': self.object.pk})

    def get_context_data(self, **kwargs):
        proc_categ = get_object_or_404(ProcedureCategory,
                                         id=self.kwargs['category'])
        kwargs['category'] = proc_categ
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        proc_categ = get_object_or_404(ProcedureCategory,
                                         id=self.kwargs['category'])
        self.procedurecategory = proc_categ
        return super().form_valid(form)

class HousekeeperprocedureUpdateView (LoginRequiredMixin,
                                       UpdateView):
    model = Housekeeperprocedure
    fields = ['specification', 'state', 'area']
    template_name = 'edit_housekeeperprocedure.html'

    def get_success_url(self):
        return reverse_lazy('housekeeperproceduresview',
                               kwargs={'pk': self.object.housekeeper.pk})

    def get_context_data(self, **kwargs):
        procedure = get_object_or_404(Houseprocedure,
                                         id=self.kwargs['procedure'])
        kwargs['procedure'] = procedure
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        procedure = get_object_or_404(Houseprocedure,
                                         id=self.kwargs['procedure'])
        self.houseprocedure = procedure
        return super().form_valid(form)

```

```

class HousekeeperprocedureVariantU pateView (LoginRequiredMixin,
UpdateView):
    model = HousekeeperprocedureVariant
    fields = ['name', 'specification', 'price', 'duration']
    template_name = 'edit_housekeeperprocedure_variant.html'

    def get_success_url(self):
        return reverse_lazy('housekeeperprocedurereview',
kwargs={'pk': self.object.housekeeperprocedure.pk})

    def get_context_data(self, **kwargs):
        housekeeperprocedure =
get_object_or_404(Housekeeperprocedure, id=self.kwargs['procedure'])
        kwargs['procedure'] = housekeeperprocedure
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        housekeeperprocedure =
get_object_or_404(Housekeeperprocedure, id=self.kwargs['procedure'])
        self.housekeeperprocedure = housekeeperprocedure
        return super().form_valid(form)

class AuthProceduresView (ListView):
    model = Housekeeperprocedure
    template_name = 'your_procedures.html'

    def get_queryset(self):
        me = self.request.user
        keeper = Housekeeper.objects.get(id = me.id)
        procedures =
Housekeeperprocedure.objects.filter(keeper = owner)
        return procedures

class ProceduresView (ListView):
    model = Housekeeperprocedure
    template_name = 'all_procedures.html'

    def get_queryset(self):
        me = self.request.user

```

```

        keeper = Housekeeper.objects.get(id = me.id)
        procedures = Housekeeperprocedure.objects.all()
        procedures = procedures.order_by('-
houseprocedure__name')
        return procedures

```

```

class SearchView (ListView):
    model = Housekeeperprocedure
    template_name = 'search.html'

    def get_queryset(self):
        price = self.request.GET.get('price')
        price = int(price)
        score = self.request.GET.get('score')
        score = float(score)
        name = self.request.GET.get('name')
        category = self.request.GET.get('category')
        area = self.request.GET.get('area')
        area = float(area)
        address_long = self.request.GET.get('address_long')
        address_long = float(address_long)
        address_lat = self.request.GET.get('address_lat')
        address_lat = float(address_lat)
        startdate = self.request.GET.get('start')
        enddate = self.request.GET.get('end')
        startdate = datetime.datetime.strptime(startdate,
"%d.%m.%Y %H:%M")
        enddate= datetime.datetime.strptime(enddate,
"%d.%m.%Y %H:%M")
        variants =
HousekeeperprocedureVariant.objects.filter(housekeeperprocedure__house
procedure__name__icontains = name)
        variants =
variants.filter(housekeeperprocedure__houseprocedure__procedurecategor
y__id = category)
        variants =
variants.filter(housekeeperprocedure__score__gte = score)
        variants = variants.filter(area__gte = area)
        variants = variants.filter(price__lte = price)

```

```

        proc_variants = []
        for variant in variants:
            hkeeper =
variant.housekeeperprocedure.housekeeper
                addr_long = hkeeper.address_long
                addr_lat = hkeeper.address_lat
                km_long = haversine.haversine(address_long,
addr_long)

                km_lat = haversine.haversine(address_lat,
addr_lat)

                radius = hkeeper.distance
                if km_long <= radius and km_lat <= radius:
                    proc_variants.append(variant)
        variants = proc_variants
        proc_variants = []
        for variant in variants:
            hkeeper =
variant.housekeeperprocedure.housekeeper
                op1 =
OrderProcedure.objects.filter(housekeeperprocedurevariant__housekeeper
procedure__housekeeper = hkeeper, keepingorder__need_at__lte =
startdate, keepingorder__ended_at__gte = startdate,
keepingorder__ended_at__lte = enddate)
                if not op1:
                    op2 =
OrderProcedure.objects.filter(housekeeperprocedurevariant__housekeeper
procedure__housekeeper = hkeeper, keepingorder__need_at__gte =
startdate, keepingorder__ended_at__gte = startdate,
keepingorder__ended_at__gte = enddate)
                    if not op2:
                        op3 =
OrderProcedure.objects.filter(housekeeperprocedurevariant__housekeeper
procedure__housekeeper = hkeeper, keepingorder__need_at__gte =
startdate, keepingorder__need_at__lte = enddate,
keepingorder__ended_at__gte = startdate, keepingorder__ended_at__lte =
enddate)

                        if not op3:
                            op4 =
OrderProcedure.objects.filter(housekeeperprocedurevariant__housekeeper

```

```

procedure__housekeeper = hkeeper, keepingorder__need_at__lte =
startdate, keepingorder__ended_at__gte = enddate)
        if not op4:

    proc_variants.append(variant)
        variants = proc_variants
        procedures = []
        for variant in variants:
            if variant.housekeeperprocedure not in
procedures:

        procedures.append(variant.housekeeperprocedure)
        res_procedures = []
        for procedure in procedures:
            start = startdate.date()
            keeper = procedure.housekeeper
            spec_timetable =
HousekeeperSpecTimetable.objects.get(housekeeper = keeper, day =
start)

            if spec_timetable:
                if spec_timetable.beginning <= startdate and
spec_timetable.ending >= enddate:
                    res_procedures.append(procedure)
            else:
                timetable = HousekeeperTimetable(housekeeper
= keeper, weekday = start.weekday())
                if timetable.beginning <= startdate and
timetable.ending >= enddate:
                    res_procedures.append(procedure)
        res_procedures = res_procedures.order_by('-score')
        return res_procedures

class ProcedureVariantsView (ListView):
    model = HousekeeperprocedureVariant
    template_name = 'procedure_variants.html'

    def get_queryset(self):
        procedure = self.request.GET.get('procedure')

```

```

        procedures =
HousekeeperprocedureVariant.objects.get(housekeeperprocedure__id =
procedure)

```

```

        procedures = procedures.order_by('name')
        return procedures

```

```

class SearchProceduresView (ListView):
    model = Houseprocedure
    template_name = 'search_procedures.html'

    def get_queryset(self):
        category = self.request.GET.get('category')
        name = self.request.GET.get('name')
        procedures =
Houseprocedure.objects.filter(procedurecategory__id = category,
name__icontains = name)
        procedures = procedures.order_by('name')
        return procedures

```

```

class ProcedureCategoryDetailView (DetailView):
    model = ProcedureCategory
    template_name = 'procedure_category.html'

    def get_context_data(self, **kwargs):
        data_about_category =
super().get_context_data(**kwargs)
        procedures =
Houseprocedure.objects.filter(procedurecategory = self.object.pk)
        procedures = procedures.order_by('name')
        data_about_category ['procedures'] = procedures
        return data_about_category

```

```

class ProcedureDetailView (DetailView):
    model = Houseprocedure
    template_name = 'house_procedure.html'

```

```

class ProcedureCategoryDeleteView (DeleteView):
    model = ProcedureCategory

```

```

class HousekeeperprocedureDetailView (DetailView):
    model = Housekeeperprocedure
    template_name = 'housekeeperprocedure.html'

    def get_context_data(self, **kwargs):
        data_about_procedure =
super().get_context_data(**kwargs)
        variants =
HousekeeperprocedureVariant.objects.get(housekeeperprocedure =
self.object.pk)
        variants = variants.order_by('name')
        data_about_procedure ['variants'] = variants
        return data_about_procedure

```

Б.3 Текст файла keepingorderview.py

```

from django.contrib.auth.mixins import LoginRequiredMixin
from django.urls import reverse_lazy
from django.views.generic.edit import CreateView
from django.views.generic import ListView
from django.views.generic import UpdateView
from django.views.generic.detail import DetailView
from django.views.generic.edit import DeleteView
from .models import KeepingOrder
from .models import HouseOwner
from .models import House
from .models import Owner
from .models import Housekeeper
from .models import OrderProcedure

class KeepingOrderCreateView (LoginRequiredMixin, CreateView):
    model = KeepingOrder
    fields = ['need_at', 'created_at', 'specification', 'price',
'ended_at']
    template_name = 'add_keepingorder.html'
    def get_success_url(self):

```

```

        return reverse_lazy('keepingorderview', kwargs={'pk':
self.object.pk})

```

```

    def get_context_data(self, **kwargs):
        housekeeper = get_object_or_404(Housekeeper,
id=self.kwargs['housekeeper'])
        kwargs['housekeeper'] = housekeeper
        house = get_object_or_404(House,
id=self.kwargs['house'])
        kwargs['house'] = house
        me = self.request.user
        owner = Owner.objects.get(id = me.id)
        houses = House.objects.filter(owner = owner)
        houses_add = HouseOwner.objects.filter(owner = owner)
        for house in houses_add:
            next_house = house.house
            houses.append(next_house)
        kwargs['houses'] = houses
        return super().get_context_data(**kwargs)

```

```

    def form_valid(self, form):
        housekeeper = get_object_or_404(Housekeeper,
id=self.kwargs['housekeeper'])
        house = get_object_or_404(House,
id=self.kwargs['house'])
        self.housekeeper = housekeeper
        self.house = house
        self.state = 'A'
        procedures = self.kwargs['procedures']
        specifications = self.kwargs['specifications']
        sep_procedures = procedures.split(',')
        sep_specifications = specifications.split(':', '-')
        curr = -1
        price = 0
        endtime = self.needed_at
        for procedure in sep_procedures:
            hkprocedure =
HousekeeperprocedureVariant.objects.get(id = procedure)
            price = price + hkprocedure.price

```

```

        endtime = endtime + hkprocedure.duration
        curr = curr + 1
        OrderProcedure.objects.create(keepingorder =
self.object.pk, housekeeperprocedurevariant = hkprocedure,
specification = sep_specifications[curr], price = hkprocedure.price
duration = hkprocedure.duration)
        self.price = price
        self.ended_at = endtime
        return super().form_valid(form)

```

```

class KeepingOrderUpdateView (LoginRequiredMixin, UpdateView):
    model = KeepingOrder
    fields = ['need_at', 'specification', 'price', 'ended_at']
    template_name = 'edit_keepingorder.html'

    def get_success_url(self):
        return reverse_lazy('keepingorderview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        house = get_object_or_404(House,
id=self.kwargs['house'])
        kwargs['house'] = house
        me = self.request.user
        owner = Owner.objects.get(id = me.id)
        houses = House.objects.filter(owner = owner)
        houses_add = HouseOwner.objects.filter(owner = owner)
        for house in houses_add:
            next_house = house.house
            houses.append(next_house)
        kwargs['houses'] = houses
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        house = get_object_or_404(House,
id=self.kwargs['house'])
        self.house = house
        procedures = self.kwargs['procedures']
        specifications = self.kwargs['specifications']

```

```

        sep_procedures = procedures.split(',')
        sep_specifications = specifications.split(':', '-')
        procedures_delete =
self.kwargs['procedures_to_delete']
        sep_procedures_to_delete =
procedures_delete.split(',')
        price = self.price
        endtime = self.ended_at
        curr = -1
        for procedure in sep_procedures:
            hkprocedure =
HousekeeperprocedureVariant.objects.get(id = procedure)
            curr = curr + 1
            OrderProcedure.objects.create(keepingorder =
self.object.pk, housekeeperprocedurevariant = hkprocedure,
specification = sep_specifications[curr], price = hkprocedure.price,
duration = hkprocedure.duration)
            price = price + hkprocedure.price
            endtime = endtime + hkprocedure.duration
        for procedure in sep_procedures_to_delete:
            res = OrderProcedure.objects.filter(id =
procedure.id).price
            restime = OrderProcedure.objects.filter(id =
procedure.id).duration
            endtime = endtime - restime
            OrderProcedure.objects.filter(id =
procedure.id).delete()
            price = price - res
        self.price = price
        self.ended_at = endtime
        return super().form_valid(form)

class KeepingOrdersView (ListView):
    model = KeepingOrder
    template_name = 'your_keepingorders.html'

    def get_queryset(self):
        me = self.request.user
        owner = Owner.objects.get(id = me.id)

```

```

houses = House.objects.filter(owner = owner)
houses_add = HouseOwner.objects.filter(owner = owner)
for house in houses_add:
    next_house = house.house
    houses.append(next_house)
orders = KeepingOrder.objects.filter(house__in =
houses)

orders = orders.order_by('need_at')
return orders

```

```

class HouseKeepingOrderView (ListView):
    model = KeepingOrder
    template_name = 'housekeeper_orders.html'

    def get_queryset(self):
        me = self.request.user
        housekeep = Housekeeper.objects.get(id = me.id)
        orders = KeepingOrder.objects.filter(housekeeper =
housekeep)

        orders = orders.order_by('need_at')
        return orders

```

```

class HouseKeeperOrder (ListView):
    model = KeepingOrder
    template_name = 'housekeeper_slots.html'

    def get_queryset(self):
        about = self.request.GET.get('keeper')
        keeper = Housekeeper.objects.get(id = about.id)
        housekeep = Housekeeper.objects.get(id = keeper.id)
        current = date.today()
        orders = KeepingOrder.objects.filter(housekeeper =
housekeep, ended_at__gte = current, state = 'D')
        orders = orders.order_by('need_at')
        for order in orders:
            order.house = None
            order.created_at = None
            order.specification = None
            order.price = 0

```

```

        return orders

    def get_context_data(self, **kwargs):
        data = super(HouseKeeperOrder,
self).get_context_data(**kwargs)
        keeper = Housekeeper.objects.get(id =
self.request.GET.get('keeper'))
        data ['keeper'] = keeper
        return data

class KeepingOrderCancel (LoginRequiredMixin, UpdateView):
    model = KeepingOrder
    template_name = 'keepingorder_cancel.html'

    def get_success_url(self):
        return reverse_lazy('keepingorderview', kwargs={'pk':
self.object.pk})

    def form_valid(self, form):
        state = self.state
        if state == 'A' or state == 'D':
            self.state = 'C'
        return super().form_valid(form)

class KeepingOrderReady (LoginRequiredMixin, UpdateView):
    model = KeepingOrder
    template_name = 'keepingorder_ready.html'

    def get_success_url(self):
        return reverse_lazy('keepingorderview', kwargs={'pk':
self.object.pk})

    def form_valid(self, form):
        state = self.state
        if state == 'A':
            self.state = 'D'
        return super().form_valid(form)

```

```

class KeepingOrderDone (LoginRequiredMixin, UpdateView):
    model = KeepingOrder
    template_name = 'keepingorder_done.html'

    def get_success_url(self):
        return reverse_lazy('keepingorderview', kwargs={'pk':
self.object.pk})

    def form_valid(self, form):
        state = self.state
        if state == 'D':
            self.state = 'O'
        return super().form_valid(form)

class KeepingOrderCrash (LoginRequiredMixin, UpdateView):
    model = KeepingOrder
    template_name = 'keepingorder_crash.html'

    def get_success_url(self):
        return reverse_lazy('keepingorderview', kwargs={'pk':
self.object.pk})

    def form_valid(self, form):
        state = self.state
        if state == 'D':
            self.state = 'N'
        return super().form_valid(form)

```

Б.4 Текст файла responseview.py

```

from django.contrib.auth.mixins import LoginRequiredMixin
from django.urls import reverse_lazy
from django.views.generic.edit import CreateView
from django.views.generic import ListView
from django.views.generic import UpdateView
from django.views.generic.detail import DetailView
from django.views.generic.edit import DeleteView
from .models import ResponseOwner

```

```

from .models import ResponseHousekeeper
from .models import ResponseProcedure
from .models import Housekeeperprocedure
from .models import Housekeeper
from .models import Owner

class ResponseProcedureCreateView (LoginRequiredMixin,
CreateView):
    model = ResponseProcedure
    fields = ['score']
    template_name = 'add_procedure_response.html'

    def get_success_url(self):
        return reverse_lazy('procresponsesview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        housekeeperprocedure =
get_object_or_404(Housekeeperprocedure, id=self.kwargs['procedure'])
        kwargs['procedure'] = housekeeperprocedure
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        me = self.request.user
        owner = Owner.objects.get(id = me.id)
        self.owner = owner
        housekeeperprocedure =
get_object_or_404(Housekeeperprocedure, id=self.kwargs['procedure'])
        self.housekeeperprocedure = housekeeperprocedure
        curr = date.today()
        self.created_at = curr
        return super().form_valid(form)

class ResponseHousekeeperCreateView (LoginRequiredMixin,
CreateView):
    model = ResponseHousekeeper
    fields = ['score']
    template_name = 'add_housekeeper_response.html'

```

```

    def get_success_url(self):
        return reverse_lazy('housekeeperresponsesview',
kwargs={'pk': self.object.pk})

```

```

    def get_context_data(self, **kwargs):
        housekeeper = get_object_or_404(Housekeeper,
id=self.kwargs['housekeeper'])
        kwargs['housekeeper'] = housekeeper
        return super().get_context_data(**kwargs)

```

```

    def form_valid(self, form):
        me = self.request.user
        owner = Owner.objects.get(id = me.id)
        self.owner = owner
        housekeeper = get_object_or_404(Housekeeper,
id=self.kwargs['housekeeper'])
        self.housekeeper = housekeeper
        curr = date.today()
        self.created_at = curr
        return super().form_valid(form)

```

```

class ResponseOwnerCreateView (LoginRequiredMixin, CreateView):
    model = ResponseOwner
    fields = ['score']
    template_name = 'add_owner_response.html'

    def get_success_url(self):
        return reverse_lazy('ownerresponsesview',
kwargs={'pk': self.object.pk})

```

```

    def get_context_data(self, **kwargs):
        owner = get_object_or_404(Owner,
id=self.kwargs['owner'])
        kwargs['owner'] = owner
        return super().get_context_data(**kwargs)

```

```

    def form_valid(self, form):
        me = self.request.user
        housekeeper = Housekeeper.objects.get(id = me.id)

```

```

        self.housekeeper = housekeeper
        owner = get_object_or_404(Owner,
id=self.kwargs['owner'])
        self.owner = owner
        curr = date.today()
        self.created_at = curr
        return super().form_valid(form)

class HousekeeperResponsesView (ListView):
    model = ResponseHousekeeper
    template_name = 'housekeeper_responses.html'

    def get_queryset(self):
        keeper = get_object_or_404(Housekeeper,
id=self.kwargs['keeper'])
        responses =
ResponseHousekeeper.objects.filter(housekeeper = keeper)
        responses = responses.order_by('created_at')
        return responses

class ProcedureResponsesView (ListView):
    model = ResponseProcedure
    template_name = 'procedure_responses.html'

    def get_queryset(self):
        procedure = get_object_or_404(Housekeeperprocedure,
id=self.kwargs['procedure'])
        responses =
ResponseHousekeeper.objects.filter(housekeeperprocedure = procedure)
        responses = responses.order_by('created_at')
        return responses

class OwnerResponsesView (ListView):
    model = ResponseOwner
    template_name = 'owner_responses.html'

    def get_queryset(self):
        owner = get_object_or_404(ResponseOwner,
id=self.kwargs['owner'])

```

```
owner) responses = ResponseOwner.objects.filter(owner =  
responses = responses.order_by('created_at')  
return responses
```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Дипломна кваліфікаційна робота
**ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ
ПОШУКУ РОБІТНИКІВ ДЛЯ
ВЕДЕННЯ ДОМАШНЬОГО
ГОСПОДАРСТВА**

Виконав студент групи КНТ-138

К.А. Марков

Керівник

С.Ю. Скрупський

2022

Рисунок В.1 – Слайд № 1

Об’єкт дослідження – процес пошуку робітників для ведення домашнього господарства.

Предмет дослідження – програмне забезпечення пошуку робітників для ведення домашнього господарства.

Мета дослідження – розробити програмне забезпечення пошуку робітників для ведення домашнього господарства для підтримки планування цих процесів за допомогою зовнішніх працівників.

2

Рисунок В.2 – Слайд № 2

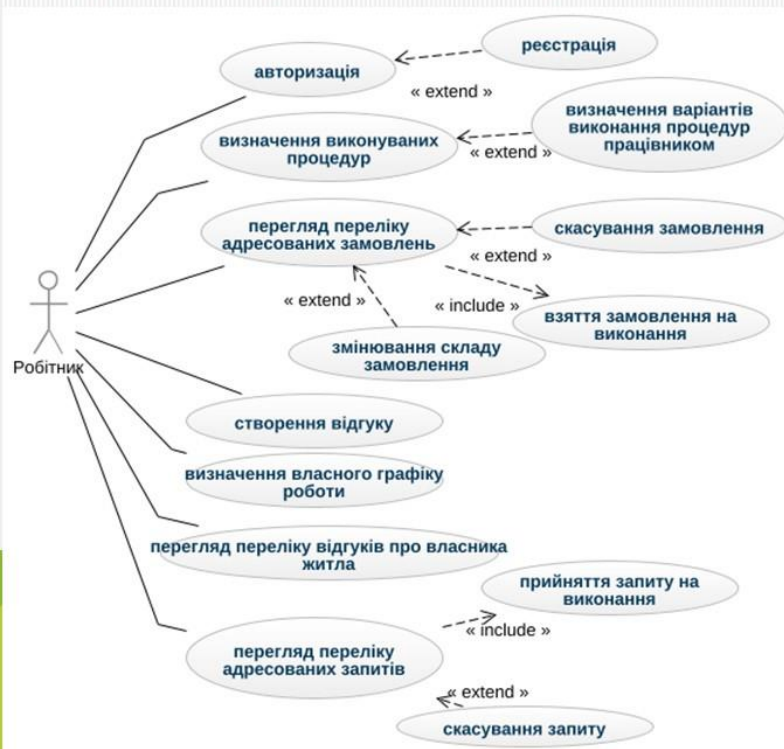
Завдання роботи

- ✓ визначення вимог до програми;
- ✓ визначення мови програмування і засобів, за допомогою яких має бути розроблена програма;
- ✓ проєктування бази даних;
- ✓ проєктування програми створення персональних відгуків про страви;
- ✓ реалізація програми пошуку робітників для ведення домашнього господарства.

3

Рисунок В.3 – Слайд № 3

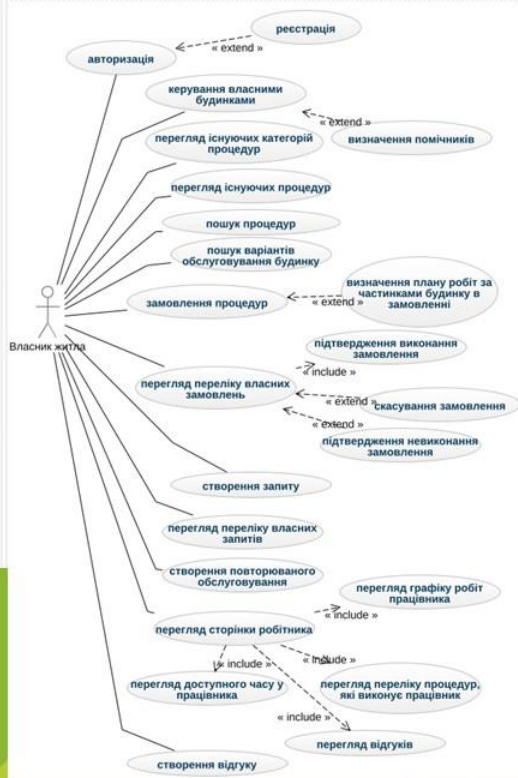
Діаграма варіантів використання



4

Рисунок В.4 – Слайд № 4

Діаграми варіантів використання



5

Рисунок В.5 – Слайд № 5

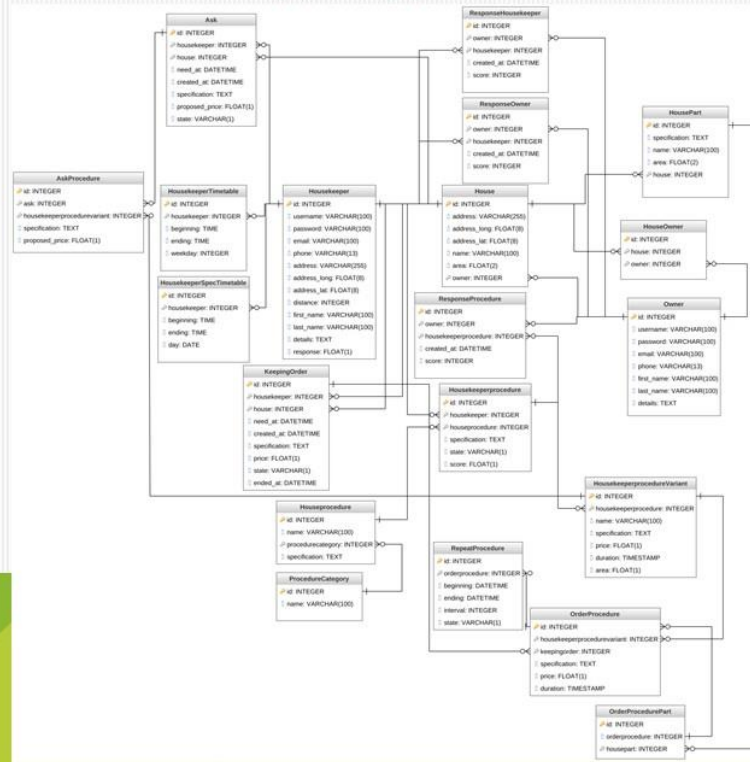
Вибір засобів розробки

Показник	Django	Node_JS
Мова для розробки в основі рішення	Python	JavaScript
Розробка серверної сторони	+	+
Вільне використання	+	+
Підтримка розробників	+	+
Ефективність самого процесу розроблення програми	+	+ –
Шаблони створення вебсторінок	+	–
Захист даних	+	+

6

Рисунок В.6 – Слайд № 6

Схема структури бази даних



7

Рисунок В.7 – Слайд № 7

Схема виконання процесу пошуку процедур за варіантами



8

Рисунок В.8 – Слайд № 8

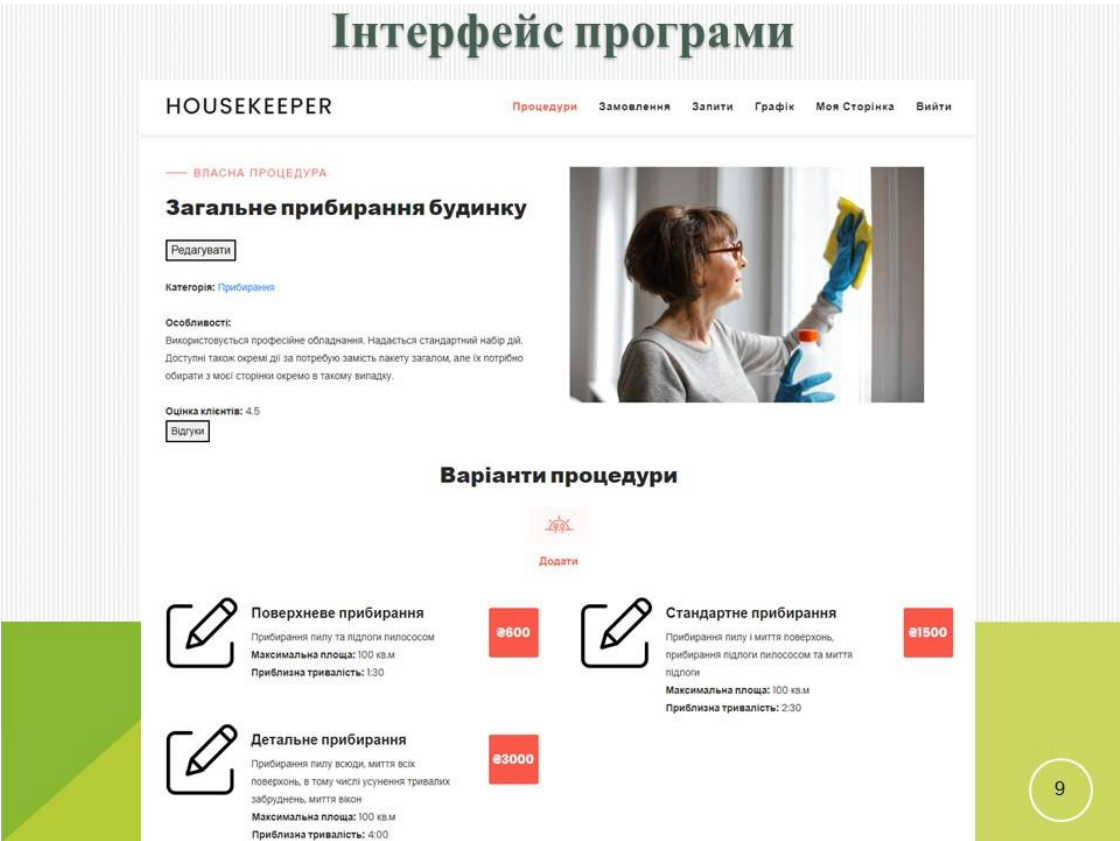


Рисунок В.9 – Слайд № 9

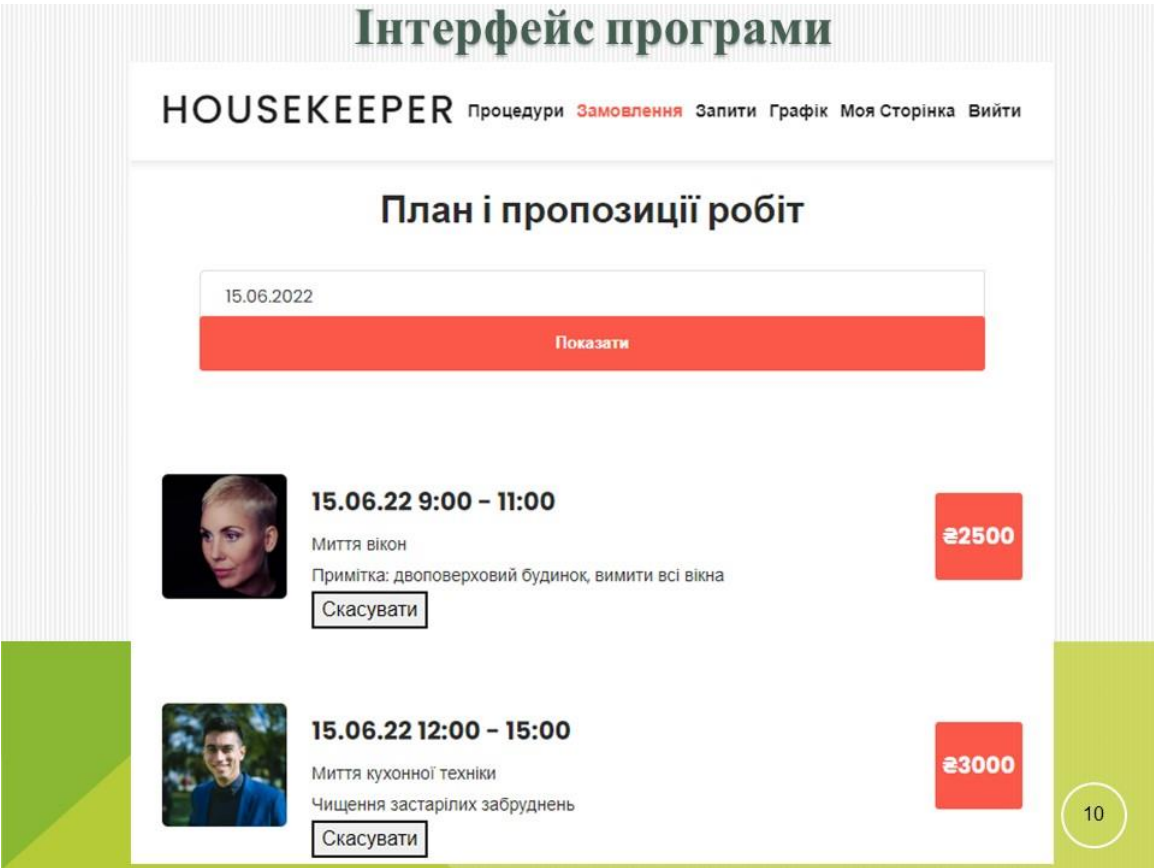


Рисунок В.10 – Слайд № 10

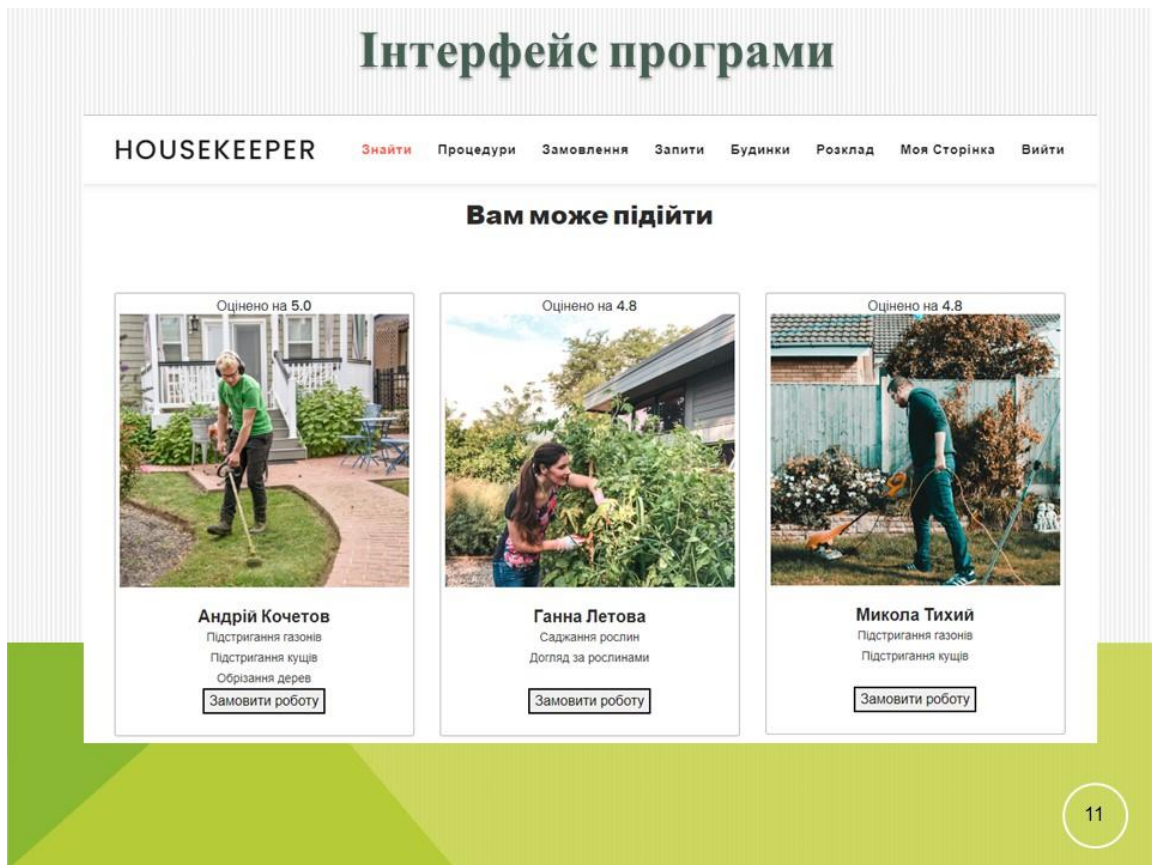


Рисунок В.11 – Слайд № 11

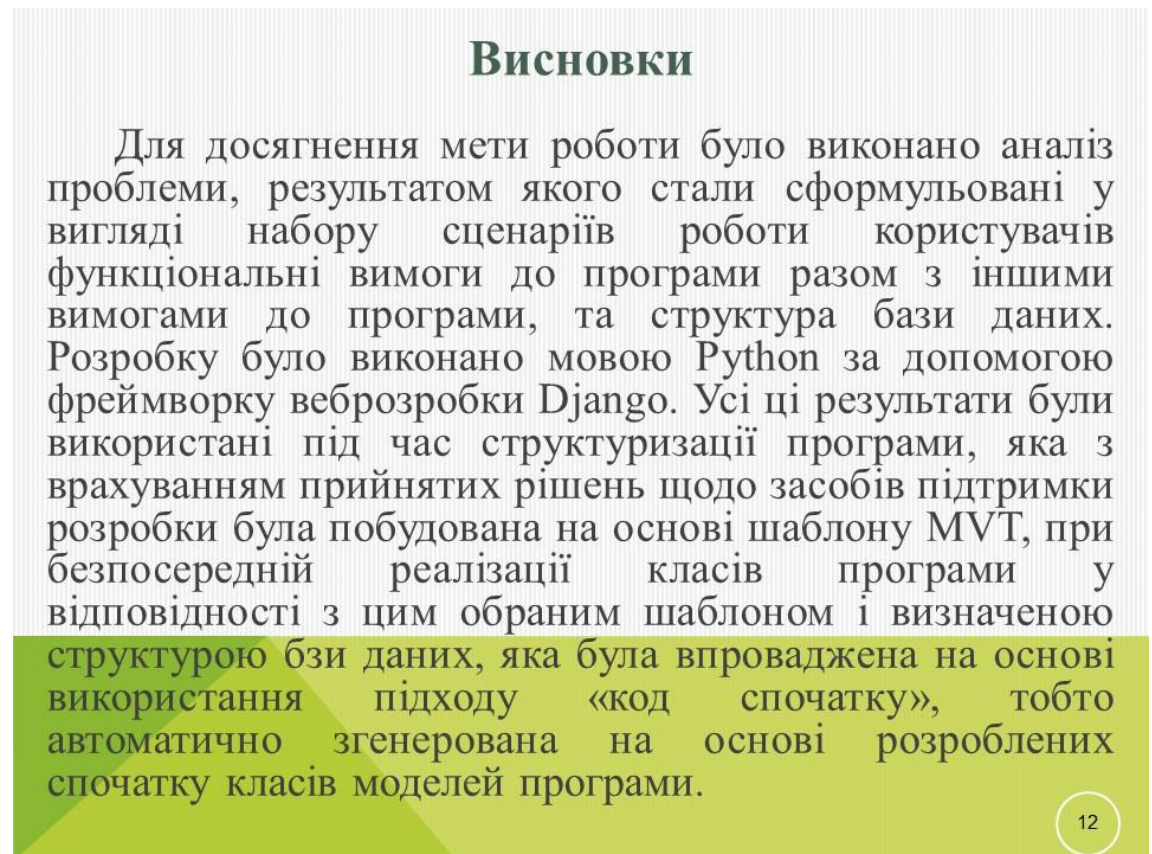


Рисунок В.12 – Слайд № 12