

Міністерство освіти і науки України
Одеський національний політехнічний університет
Інститут комп'ютерних систем
Кафедра інформаційних систем

КУРСОВА РОБОТА
з дисципліни «Технології створення програмних продуктів»
за темою
«Лапка допомоги»
Частина №3

Виконав(ла):
студентка 3-го курсу
групи АІ-183
Усова А.В.
Перевірив:
Блажко О. А.

Одеса-2020

Анотація

В курсовій роботі розглядається процес створення програмного продукту «Лапка допомоги». Робота виконувалась в команді з декількох учасників: Поліщук Катерина Андріївна, Соловйова Діана Володимирівна, Усова Анастасія Володимирівна. Тому в пояснівальній записці у розділах «Проектування» та «Конструювання» детальніше описано лише одну частину з урахуванням планів проведених робіт з розділу «Планування» з описом особливостей конструювання:

- структур даних моделі MVP в системі керування базами даних PostgreSQL;
- програмних модулів в інструментальному середовищі IntelliJ IDEA з використанням фреймворку Java Spring та мови програмування Java.

Результати роботи розміщено на *github*-репозиторії за адресою:
<https://github.com/staceyDragon/ShelterProject>.

Перелік скорочень

ОС – операційна система

ІС – інформаційна система

БД – база даних

СКБД – система керування базами даних

ПЗ – програмне забезпечення

ПП – програмний продукт

UML – уніфікована мова моделювання

Зміст

1 Вимоги до програмного продукту	7
1.1 Визначення потреб споживача	7
1.1.1 Ієрархія потреб споживача	7
1.1.2 Деталізація матеріальної потреби	8
1.2 Бізнес-вимоги до програмного продукту	8
1.2.1 Опис проблеми споживача	8
1.2.1.1 Концептуальний опис проблеми споживача	8
1.2.1.2 Метричний опис проблеми споживача	9
1.2.2 Мета створення програмного продукту	10
1.2.2.1 Проблемний аналіз існуючих програмних продуктів	10
1.2.2.2 Мета створення програмного продукту	12
1.2.3 Назва програмного продукту	12
1.2.3.1 Гасло програмного продукту	12
1.2.3.2 Логотип програмного продукту	12
1.3 Вимоги користувача до програмного продукту	13
1.3.1 Історія користувача програмного продукту	13
1.3.2 Діаграма прецедентів програмного продукту	14
1.3.3 Сценаріїв використання прецедентів програмного продукту	15
1.4 Функціональні вимоги до програмного продукту	20
1.4.1. Багаторівнева класифікація функціональних вимог	20
1.4.2 Функціональний аналіз існуючих програмних продуктів	23
1.5 Нефункціональні вимоги до програмного продукту	23
1.5.1 Опис зовнішніх інтерфейсів	23
1.5.1.1 Опис інтерфейса користувача	23
1.5.1.1.1 Опис INPUT-інтерфейса користувача	23
1.5.1.1.2 Опис OUTPUT-інтерфейса користувача	24

1.5.1.2 Опис інтерфейсу із зовнішніми пристроями	31
1.5.1.3 Опис програмних інтерфейсів	32
1.5.1.4 Опис інтерфейсів передачі інформації	32
1.5.1.5 Опис атрибутів продуктивності	32
2 Планування процесу розробки програмного продукту	33
2.1 Планування ітерацій розробки програмного продукту	33
2.2 Концептуальний опис архітектури програмного продукту	33
2.3 План розробки програмного продукту	34
2.3.1 Оцінка трудомісткості розробки програмного продукту	36
2.3.2 Визначення дерева робіт з розробки програмного продукту	40
2.3.3 Графік робіт з розробки програмного продукту	45
2.3.3.1 Таблиця з графіком робіт	45
2.3.3.2 Діаграма Ганта	46
3 Проектування програмного продукту	47
3.1 Концептуальне та логічне проектування структур даних програмного продукту	47
3.1.1 Концептуальне проектування на основі UML-діаграми концептуальних класів	47
3.1.2 Логічне проектування структур даних	48
3.2 Проектування програмних класів	49
3.3 Проектування алгоритмів роботи методів програмних класів	50
3.4 Проектування тестових наборів методів програмних класів	53
4 Конструювання програмного продукту	62
4.1 Особливості конструювання структур даних	62
4.1.1 Особливості інсталяції та роботи з СУБД	62
4.1.2 Особливості створення структур даних	62
4.2 Особливості конструювання програмних модулів	64
4.2.1 Особливості роботи з інтегрованим середовищем розробки	64

4.2.2 Особливості створення програмної структури з урахуванням спеціалізованого Фреймворку	65
4.2.3 Особливості створення програмних класів	66
4.2.4 Особливості розробки алгоритмів методів програмних класів або процедур/функцій	68
4.3 Модульне тестування програмних класів	72
5 Розгортання та валідація програмного продукту	81
5.1 Інструкція з встановлення програмного продукту	81
5.2 Інструкція з використання програмного продукту	81
5.3 Результати валідації програмного продукту	83
Висновки до курсової роботи	84

1 Вимоги до програмного продукту

1.1 Визначення потреб споживача

1.1.1 Ієрархія потреб споживача

Відомо, що в теорії маркетингу потреби людини можуть бути представлені у вигляді ієрархії потреб ідей американського психолога Абрахама Маслоу включають рівні:

- фізіологія (вода, їжа, житло, сон);
- безпека (особиста, здоров'я, стабільність),
- принадлежність (спілкування, дружба, любов),
- визнання (повага оточуючих, самооцінка),
- самовираження (вдосконалення, персональний розвиток).

На рисунку 1.1 представлено рівень потреби споживача, який хотілося б задоволити, використовуючи майбутній програмний продукт.



Рисунок. 1.1.1 – Ієрархія потреби споживача. Рівень потреби – принадлежність

Був обраний рівень «Принадлежність», тому що, використовуючи програмний продукт «Лапка допомоги», споживач задовольняє таки потреби, як потреба у любові, дружбі та спілкуванні.

1.1.2 Деталізація матеріальної потреби

Для деталізації матеріальної потреби можна скористатися ментальними картами (MindMap). При створенні ментальних карт матеріальна потреба розташовується в центрі карти. Асоціативні гілки можна швидко створити, припускаючи, що в загальному вигляді з об'єктом пов'язані три потоки даних / інформації: вхідний, внутрішній, вихідний. Кожен потік - це асоціативна група, що включає можливі п'ять гілок, що відповідають на п'ять питань: Хто? Що? Де? Коли? Як?

Відповідно до рекомендацій по створенню ментальних карт кожна гілка-асоціація може бути розділена на додаткові асоціативні гілки, які деталізують відповіді на поставлені питання.

Потреба, яка була визначена при аналізі матеріальних проблем споживача, основні та додаткові асоціативні гілки зображені на Рис 1.2

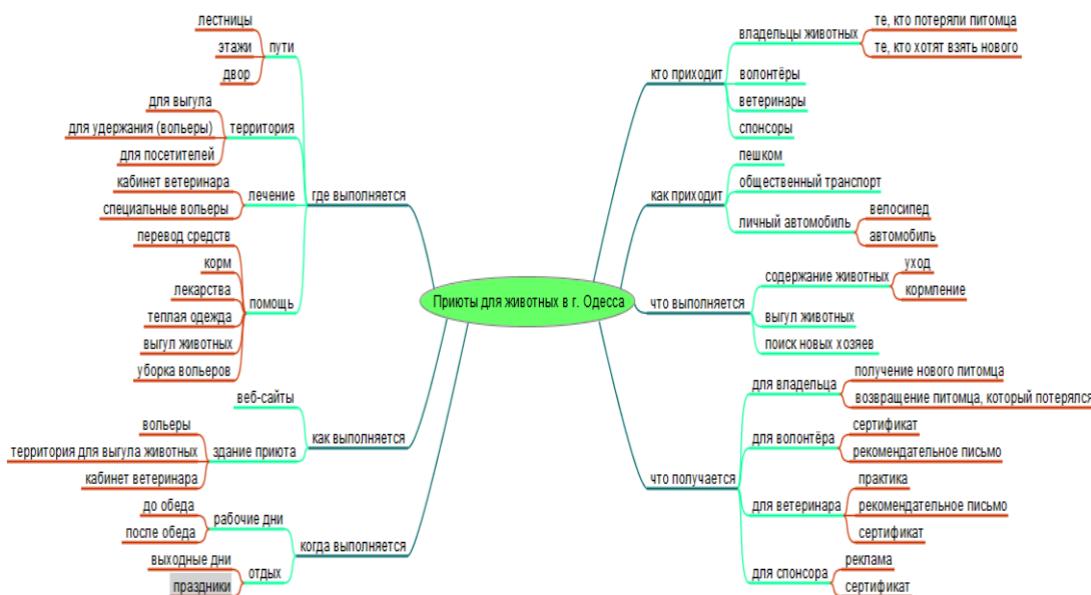


Рисунок - 1.1.2 – Деталізація матеріальної потреби

1.2 Бізнес-вимоги до програмного продукту

1.2.1 Опис проблеми споживача

1.2.1.1 Концептуальний опис проблеми споживача

Концептуальний опис проблеми споживача відображені у табл. 1.1

Таблиця – 1.2.1.1

№	Загальний опис проблеми
1	Довгий пошук інформації про загублену чи знайдену тарину
2	Недостатнє фінансування притулків для тварин.
3	Відсутність відстеження кількості бездомних тварин.
4	Недостатня кількість волонтерів для догляду за тваринами
5	Довгий пошук нового вихованця при виборі його з безлічі притулків міста.

1.2.1.2 Метричний опис проблеми споживача

Метричний опис проблем споживача відображені у табл.1.2.1 – 1.2.5

Таблиця – 1.2.1.2.1

№	Загальний опис проблеми	Метричні показники незадоволеності споживача
1	Довгий пошук інформації про загублену чи знайдену тарину	Низький рівень доступності до інформації про домашніх тварин.

Рівень доступності $AL = NA / N$, де

NA - кількість власників, які зможуть обійти всі притулки або перейти на всі існуючі веб- сайти про зниклих тварин.

N - загальна кількість власників тварин, які загубилися.

$AL \rightarrow 0$

Таблиця – 1.2.1.2.2

2	Недостатнє фінансування притулків для тварин.	Низький рівень доступності змісту і лікування тварин, які перебувають у притулку.
---	---	---

Рівень доступності $AL = NA / N$, де

NA - кількість притулків з фінансуванням достатнім для утримання тварин.

N - загальна кількість притулків.

$AL \rightarrow 0.$

Таблиця – 1.2.1.2.3

3	Відсутність відстеження кількості бездомних тварин.	Низький рівень актуальності інформації про кількість бездомних тварин.
---	---	--

Рівень актуальності $AL = NA / N$, де

NA - кількість актуальної інформації про бездомних тварин.

N - загальна кількість інформації про бездомних тварин.

$AL \rightarrow 0.$

Таблиця – 1.2.1.2.4

4	Недостатня кількість волонтерів для догляду за тваринами	Низький рівень доступності лікування та догляду за вихованцями, які перебувають у притулку
---	--	--

Рівень доступності $AL = NA / N$, де

NA - кількість волонтерів, які забезпечують догляд та лікування.

N - загальна кількість необхідних волонтерів для утримання притулку.

$AL \rightarrow 0.$

Таблиця – 1.2.1.2.5

5	Довгий пошук нового вихованця при виборі його з безлічі притулків міста.	Низький рівень доступності отримання нового вихованця.
---	--	--

Рівень доступності $AL = NA / N$, де

NA - кількість майбутніх власників, які виберуть вихованця з першого притулку.

N - загальна кількість майбутніх власників, які шукають нового вихованця.

$AL \rightarrow 0.$

1.2.2 Мета створення програмного продукту

1.2.2.1 Проблемний аналіз існуючих програмних продуктів

Проблемний аналіз існуючих програмних продуктів відображені у табл.2.1

Таблиця 1.2.2.1

№	Назва продукту	Вартість	Ступінь готовності	Примітка
1	Animal ID	безкоштовно	3	немає можливості розміщувати оголошення, немає можливості шукати пункти перетримки тварин і не локально по Одесі
2	Ковчег Одесса	безкоштовно	2	немає можливості шукати пункти перетримки тварин, відсутність оголошень про пропажу, незручний для користувача інтерфейс
3	Кошкин Дом	безкоштовно	2	не є веб-сервісом, ніякого взаємодії з користувачем, тільки галерея
4	Помощник РЭЙ	безкоштовно	4	немає можливості шукати пункти перетримки тварин, відсутня можливість

				викласти оголошення для будь- якого користувача
--	--	--	--	---

1.2.2.2 Мета створення програмного продукту

Підвищення рівня доступності до інформації про втрачених, знайдених і безпритульних тварин в м Одеса, поширення інформації про можливі варіанти допомоги притулкам на підставі створення веб-сайту для об'єднання необхідної інформації.

1.2.3 Назва програмного продукту

Назва «Лапка допомоги» відображає мету та гасло програмного продукту. Головна задача створення веб-сервісу – це допомога тваринам та людям, які мають тварин. «Допомога» - ключове слово, на основі якого будується принцип роботи веб-сайту.

1.2.3.1 Гасло програмного продукту

Гасло — лаконічна фраза, що впадає в око, добре запам'ятовується та висловлює суть продукту.

На початковому етапі розробки було вигадано гасло, яке найкращим чином відображає мету та суть роботи веб-сервісу та пов'язано з логотипом та назвою.

Гасло - «Простягни тваринному світу лапу допомоги!»

1.2.3.2 Логотип програмного продукту

Відмінним способом представлення назви програмного продукту є його логотип, що поєднує зорові образи.

На рис 1.2.3.2 можна побачити розроблений командою логотип, який відображає назву веб-сервісу.



Рисунок - 1.2.3.2 - Логотип

1.3 Вимоги користувача до програмного продукту

1.3.1 Історія користувача програмного продукту

User-stories

Як хазяїн тварини, я хочу викласти оголошення на сайт про втрату тваринного, щоб знайти його.

Як відвідувач, я хочу переглянути інформацію про загублених тварин, щоб допомогти господарям знайти їх вихованців.

Як відвідувач / потенційний хазяїн, я хочу переглянути інформацію про тварин з притулків, щоб вибрати зацікавив мене.

Як відвідувач / потенційний хазяїн, я хочу зберегти зацікавила мене анкету вихованця на майбутнє, щоб не шукати її потім заново.

Як хазяїн тварини, я хочу переглянути інформацію про можливі пунктах перетримки тварини, щоб мені було, де його залишити, коли я пойду.

Як відвідувач / потенційний хазяїн / хазяїн тварини, я хочу увійти в свій аккаунт, де будуть збережені мої дані, щоб не вводити кожен раз їх заново.

Як відвідувач / потенційний хазяїн /хазяїн тварини, я хочу переглянути інформацію про притулок, щоб зробити пожертвування.

Як відвідувач / потенційний хазяїн / хазяїн тварини, я хочу подивитися на відгуки про роботу компанії і результат виконання їхніх обіцянок, щоб змогти довіряти їм.

Як представник притулку, я хочу викласти інформацію про притулок на платформу, щоб ймовірність знаходження допомоги для закладу і знаходження господарів для тварин була вищою.

Як адміністратор / модератор, я хочу модерувати всю опубліковану інформацію, щоб здійснювати фільтрацію даних на платформі.

1.3.2 Діаграма прецедентів програмного продукту

Діаграма прецедентів (Use Case UML-діаграма) включає:

- актори (зацікавлені особи і зовнішні системи зі своїм API);
- прецеденти як основні функції ПП;
- зв'язки між прецедентами і акторами як множиною зацікавлених осіб;
- можливі зв'язки-узагальнення між акторами.

Діаграма прецедентів програмного продукту зображена на рис. 1.3.1

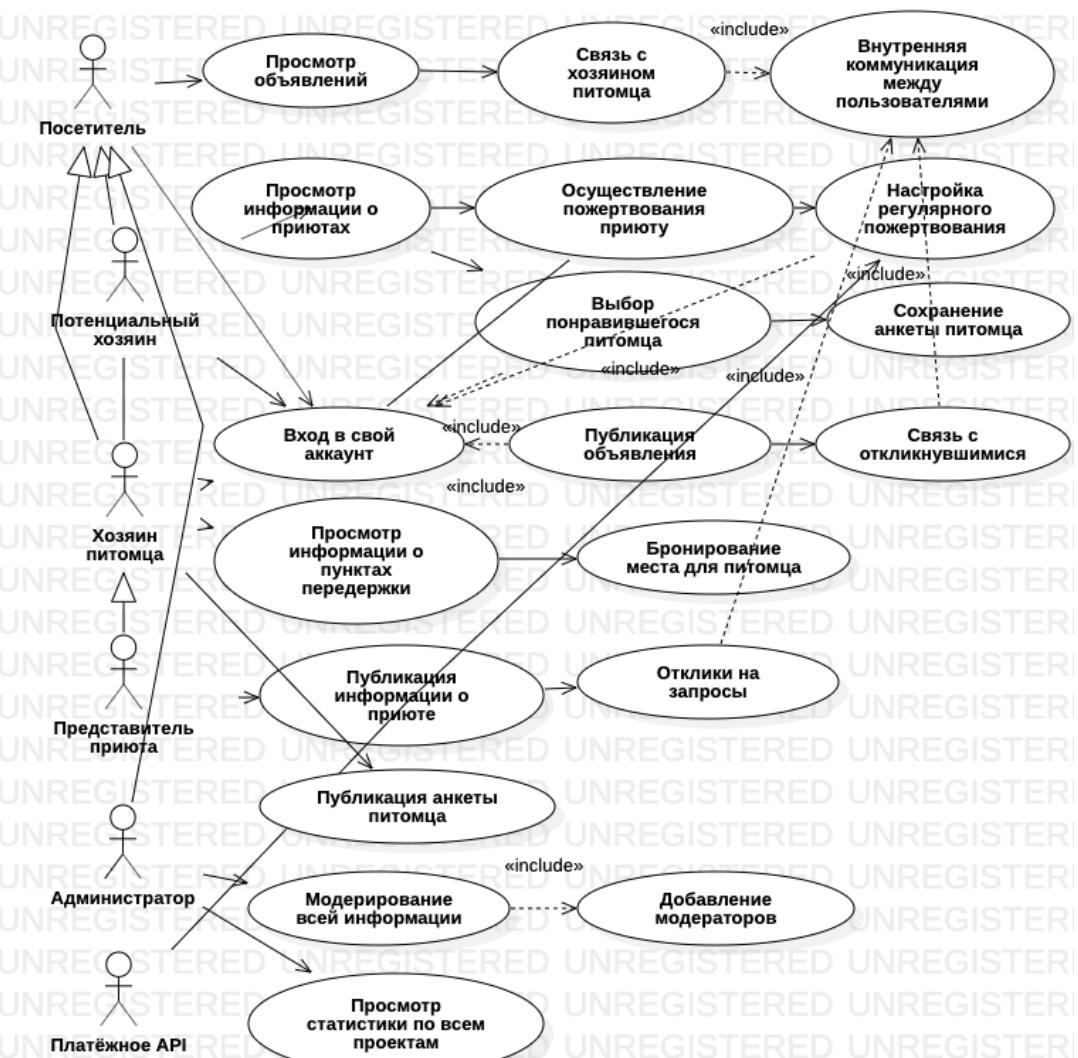


Рисунок - 1.3.2. Діаграма прецедентів програмного продукту

1.3.3 Сценарій використання прецедентів програмного продукту

Для кожного прецедента описан сценарій використання з урахуванням пунктів:

- назва прецеденту;
- передумови початку виконання прецеденту;
- актори як зацікавлені особи у виконанні прецеденту;
- актор-основна зацікавлена особа як ініціатор початку прецеденту;
- гарантії успіху (що отримають актори у разі успішного завершення прецеденту);

- основний успішний сценарій;
- альтернативні сценарії, прив'язані до кроків основного успішного сценарію.

Опис сценарію “Авторизація”

Умова початку прецеденту: Дія з боку Відвідувач.

Актори, що зацікавлені в виконанні даного прецеденту:

Відвідувач.

Актор - основа початку: Відвідувач

Гарантії успіху: успішна авторизація на веб-сайті

Успішний сценарій:

- ПП запрошує у користувача параметри авторизації: логін/пошту та пароль
- Користувач передає свої параметри
- ПП надає користувачу доступ до інших прецедентів ПП

Альтернативний сценарій:

- ПП виявляє помилку в значенні переданих користувачем параметрів
- ПП видає повідомлення про помилку та переходить до кроку а успішного сценарію

Опис сценарію “Публікація об’яви”

Умова початку прецеденту: здійснення авторизації

Актори, що зацікавлені в виконанні даного прецеденту:

Відвідувач.

Актор - основа початку: Відвідувач

Гарантії успіху: успішна публікація на веб-сайті

Успішний сценарій

- ПП запрошує у користувача параметри об’яви: фото, опис, називу

- б) користувач передає параметри
- в) об'ява опублікована, ПП надає користувачу доступ до наступних прецедентів

Альтернативний сценарій:

- г) ПП виявляє помилку в значенні переданих користувачем параметрів (або модератор)

д) ПП видає повідомлення про помилку – об'ява не пройшла модерацію, та переходить до кроку а успішного сценарію

Опис сценарію “Перегляд об’яви”

Умова початку прецеденту: здійснення авторизації.

Актори, що зацікавлені в виконанні даного прецеденту:

Відвідувач.

Аktor - основа початку: Відвідувач

Гарантії успіху: успішний перегляд об’яви

Успішний сценарій

а) користувач переходить до об’яви

б) ПП надає йому доступ до зв’язку з автором

Альтернативний сценарій

а) ПП виявляє помилку в значенні переданих користувачем параметрів (або модератор)

б) ПП видає повідомлення про помилку – перегляд об’яви неможливий, та переходить до кроку а успішного сценарію

Опис сценарію “Модерація об’яви”

Умова початку прецеденту: публікація об’яви.

Актори, що зацікавлені в виконанні даного прецеденту:

moderator, відвідувач, представник притулку

Аktor - основа початку: moderatop

Гарантії успіху: успішна модерація об’яви

Успішний сценарій

- а) модератор входить в аккаунт
- б) ПП надає йому доступ до неопублікованих змін або опублікованих нових
- в) модератор робить свої правки
- г) ПП збережує та оновлює інформацію

Альтернативний сценарій

- а) ПП виявляє помилку
- б) ПП видає повідомлення про помилку

Опис сценарію “Здійснення пожертвування”

Умова початку прецеденту: авторизація відвідувача

Актори, що зацікавлені в виконанні даного прецеденту:

представник приюту

Актори, задіяні в сценарії: відвідувач, платіжне API

Актор-основа початку: відвідувач

Гарантії успіху: успішне здійснення пожертвування

Успішний сценарій прецедента

- а) API запрошує у користувача параметри картки для оплати рахунку
- б) користувач передає параметри
- в) параметри вірні, оплата пройшла успішно (API повідомляє про оплату)
- г) ПП надає повідомлення про успішну оплату, та з часом – звіт

Альтернативний сценарій:

- д) API виявляє помилку в значенні переданих користувачем параметрів
- е) ПП видає користувачеві повідомлення про помилку та запитує щодо повторної спроби, якщо так - пункт е) ні - кінець сценарію
- е) перехід до пункту а)

Опис сценарію “Перегляд анкети тварини”

Умова початку прецеденту: авторизація відвідувача

Актори, що зацікавлені в виконанні даного прецеденту:

відвідувач

Актор - основа початку: відвідувач

Гарантії успіху: успішний перегляд тварин

Успішний сценарій прецедента

а) користувач переходить до анкети

б) ПП надає йому доступ до інформації про тварину

Альтернативний сценарій:

а) ПП виявляє помилку в значенні переданих користувачем параметрів

б) ПП видає повідомлення про помилку

Опис сценарію “Публікація інформації про притулок”

Умова початку прецеденту: авторизація відвідувача

Актори, що зацікавлені в виконанні даного прецеденту:

Представник притулку

Актор - основа початку: представник притулку

Гарантії успіху: успішна публікація

1. Успішний сценарій

а) представник притулку переходить до дії - публікації

б) ПП запрошує у користувача параметри, необхідні для заповнення полей об'єкту “притулок”: назва, фото, опис, контакти

в) представник притулку передає параметри

г) інформація опублікована, ПП надає користувачу доступ до наступних прецедентів

Альтернативний сценарій:

д) ПП виявляє помилку в значенні переданих користувачем параметрів (або модератор)

е) ПП видає повідомлення про помилку – притулок не пройшов модерацію, та переходить до кроку б успішного сценарію

Опис сценарію “Заповнення анкети для тварини”

Умова початку прецеденту: авторизація відвідувача

Актори, що зацікавлені в виконанні даного прецеденту:

Відвідувач

Аktor - основа початку: відвідувач

Гарантії успіху: успішно заповнена та збережена на сайті анкета

1. Успішний сценарій

а) представник притулку переходить до дії - заповнення анкети для тварини

б) ПП запрошує у користувача параметри, необхідні для заповнення полей об'єкту “тварина”: фото, ім'я, опис

в) відвідувач передає параметри

г) інформація збережена та опублікована, ПП надає користувачу доступ до наступних прецедентів

2. Альтернативний сценарій:

д) ПП виявляє помилку в значенні переданих користувачем параметрів (або модератор)

е) ПП видає повідомлення про помилку

1.4 Функціональні вимоги до програмного продукту

1.4.1. Багаторівнева класифікація функціональних вимог

Таблиця – 1.4.1. Багаторівнева класифікація функціональних вимог

Ідентифікатор функції	Назва функції
-----------------------	---------------

FR0	Перегляд сайту без авторизації
FR1	Авторизація користувача (реєстрація)
FR1.1	Створення запиту у користувача на відправку його параметрів ідентифікації і аутентифікації
FR2.1	Авторизація користувача (вхід в систему)
FR2.2	Створення сторінки «Мій акаунт»
FR3	Публікація оголошення
FR4	Перегляд оголошень
FR4.1	Показ всіх доступних оголошень на сайті
FR4.1.1	Показ обраного доступного оголошення
FR5	Модерування оголошень
FR5.1	Показ всіх оголошень для модерування
FR5.2	Вибір, перегляд оголошень для модерування
FR5.3	Публікація оголошень
FR6	Пожертвування
FR6.1	Виклик форми для введення платіжних даних за допомогою API платіжної системи
FR7	Перегляд анкет тварин
FR7.1	Перегляд анкети тварини
FR8	Створення притулку
FR8.1	Валідація полів форми
FR8.1.1	Збереження притулку в БД і публікація притулку
FR9	Перегляд списку притулків
FR10	Перегляд інформації про притулок

FR10.1	Вибір притулку зі списку
FR11	Відновлення паролю
FR11.1	Створення форми відновлення пароля
FR11.2	Валідація форми відновлення пароля
FR11.2.1 (Success)	Відправлення листа на вказаний користувачем email для підтвердження пароля
FR11.2.2 (Failure)	Показ помилки(ок) валідації форми
FR12	Підтвердження зміни пароля

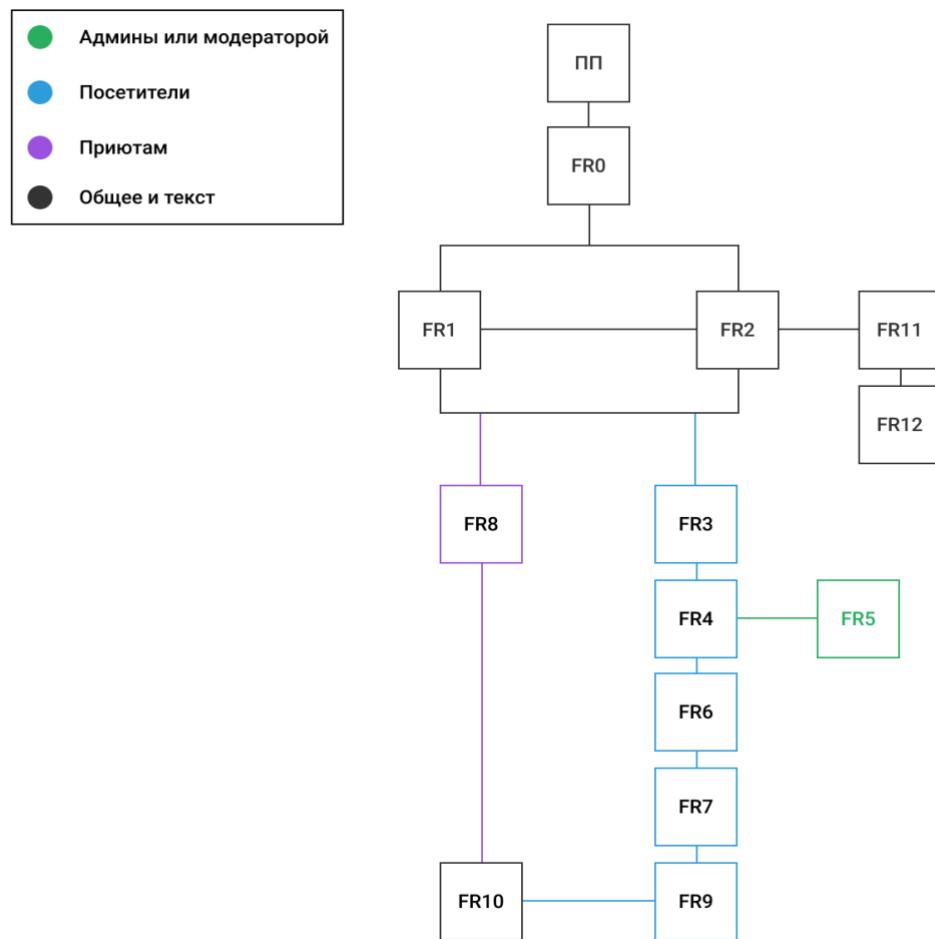


Рисунок – 1.4.1. WBS-структурата багаторівневої класифікації функціональних вимог

1.4.2 Функціональний аналіз існуючих програмних продуктів

Таблиця – 4.2. Функціональний аналіз існуючих програмних продуктів

Ідентифікатор функції	Кошkin Дом	Помощник Рей	Ковчег	Animal ID
FR0	+	+	+	+
FR1	+	-	-	+
FR2	+	-	-	+
FR3	+	-	+	-
FR4	+	+	+	+
FR5	+	+	+	+
FR6	+	+	+	+
FR7	+	+	+	+
FR8	+	+	+	+
FR9	-	-	+	+
FR10	+	-	-	+
FR11	+	-	+	-
FR12	+	-	+	-

1.5 Нефункціональні вимоги до програмного продукту

1.5.1 Опис зовнішніх інтерфейсів

1.5.1.1 Опис інтерфейса користувача

1.5.1.1.1 Опис INPUT-інтерфейса користувача

Таблиця – 1.5.1.1.1. Опис INPUT-інтерфейса користувача

Ідентифікатор функції	Засіб INPUT-потоку	Особливості використання
FR0	Маніпулятор типу миша, колесо миši	Колесо миši для повного перегляду всієї сторінки
FR1	Маніпулятор типу миша	Введення даних
FR2	Маніпулятор типу миша, клавіатура	-
FR3	Маніпулятор типу миша, клавіатура	-
FR4	Маніпулятор типу миша, клавіатура	-
FR5	Маніпулятор типу миша	-

FR6	Маніпулятор типу миша, клавіатура	-
FR7	Маніпулятор типу миша, клавіатура	-
FR8	Маніпулятор типу миша, клавіатура	-
FR9	Маніпулятор типу миша, клавіатура	-
FR10	Маніпулятор типу миша, клавіатура	-
FR11	Маніпулятор типу миша, клавіатура	-
FR12	Маніпулятор типу миша, клавіатура	-

1.5.1.1.2 Опис OUTPUT-інтерфейса користувача

Для функції «FR0» OUTPUT-інтерфейса користувача.

О нас

Товарищи! Укрепление и развитие структуры требуют от нас анализа модели развития. Задача организаций в особенности же реализации намеченных плановых заданий требуют от нас анализа систем массового участия. Значимость этих проблем настолько очевидна, что укрепление и развитие структуры в значительной степени обуславливает создание систем массового участия. Задача организации, в особенности же дальнейшее развитие различных форм деятельности влечет за собой процесс внедрения и модернизации форм развития.

Наши приюты

Слайдер с пятью фотографиями приютов.

Наши волонтеры

Слайдер с восемью фотографиями волонтеров.

Наши партнеры

Слайдер с восемью фотографиями партнеров.

Сотрудничество **Вопросы и ответы** **Животные - братья наши меньшие**

Помощь **Донаты**

Волонтерство **Другое**

О нас **Одесса**

Оплата

НАШИ ПРИЮТЫ

ДОБРОЕ СЕРДЦЕ, **КОТЕНОК**, **ЛУЧШИЙ ДРУГ**

НАШИ ВОЛОНТЕРЫ

МАРИЯ ЧИКОВА, **ИРИНА МАСЛОВА**, **ДМИТРИЙ ПОЛЯКОВ**

Сотрудничество **Вопросы и ответы** **Оплата**

Помощь **Донаты** **МастерCard**, **VISA**

Волонтерство **Другое**

О нас **Одесса**

Оплата

M

Рисунок 1.5.1.1.2.1 – Прототип та фінальний дизайн функції FR0
Для функції «FR1» OUTPUT-інтерфейса користувача.

Назад О нас Помощь Рус ▾

Регистрация

У меня уже есть аккаунт

Имя

Фамилия

E-mail или телефон

Пароль

Подтверждение пароля

Запомнить меня в течении 30 дней

Регистрация

Рисунок 1.5.1.1.2.2 – Прототип функції FR1

Назад О нас Помощь

РЕГИСТРАЦИЯ

У меня уже есть аккаунт

Имя

Фамилия

E-mail или телефон

Пароль

Подтверждение пароля

Запомнить меня в течении 30 дней

Регистрация

Рисунок 1.5.1.1.2.3 – Фінальний дизайн функції FR1

Для функції «FR2» OUTPUT-інтерфейса користувача.

Назад О нас Помощь Руc ▾

Вход

У меня нет аккаунта

E-mail или телефон

Пароль

Я не помню свой пароль

Запомнить меня в течении 30 дней

Регистрация

Рисунок 1.5.1.1.2.4 – Прототип функції FR2

Назад О нас Помощь

ВХОД

У меня нет аккаунта

E-mail или телефон

Пароль

Я не помню свой пароль

Запомнить меня в течении 30 дней

Регистрация

Рисунок 1.5.1.1.2.5 – Фінальний дизайн функції FR2

Для функції «FR7.1» OUTPUT-інтерфейса користувача.

Left Version (Wireframe):

- Header: Животные - братья наши меньшие, Рус, Авторизация
- Navigation: Приюты, Объявления, Пожертвования, Отели, О нас, Вопросы и ответы, Помощь
- Page Title: < Главная < Приюты < Приют "Название 1" < Просмотр питомца
- Section: Вася
- Buttons: Помочь, Забрать
- Text: Товарищи! Укрепление и развитие структуры требуют от нас анализа модели развития. Задача организации, в особенности же реализации начальных плановых заданий требует от нас анализа систем массового участия. Значимость этих проблем настолько очевидна, что укрепление и развитие структуры в значительной степени обуславливает создание систем массового участия. Задача организации, в особенности же дальнейшее развитие различных форм деятельности влечет за собой процесс внедрения и модернизации форм развития.
- Table: Вес: 2,5 кг, Животное: Кот, Возраст: 3 года, Особенности: Очень игрив, Парода: Уличная, Статус: Мечтает о любящем хозяине
- Section: Другие питомцы
- Image Placeholder: Веснушка, Небо, Туся, Маня
- Footer: Сотрудничество, Вопросы и ответы, Помощь, Волонтерство, О нас, Оплата: монобанк, Visa, MasterCard, МИР, Одесса

Right Version (Final Design):

- Header: Животные - братья наши меньшие, Авторизация
- Navigation: Приюты, Объявления, Пожертвования, Отели, О нас, Вопросы и ответы, Помощь
- Page Title: < Главная < Приюты < Приют "Название 1" < Просмотр питомца
- Section: Вася
- Image: A close-up photo of a puppy's face.
- Buttons: Помочь, Забрать
- Text: Товарищи! Укрепление и развитие структуры требуют от нас анализа модели развития. Задача организации, в особенности же реализации начальных плановых заданий требует от нас анализа систем массового участия. Значимость этих проблем настолько очевидна, что укрепление и развитие структуры в значительной степени обуславливает создание систем массового участия. Задача организации, в особенности же дальнейшее развитие различных форм деятельности влечет за собой процесс внедрения и модернизации форм развития.
- Table: Вес: 2,5 кг, Животное: Кот, Возраст: 3 года, Особенности: Очень игрив, Парода: Уличная, Статус: Мечтает о любящем хозяине
- Section: Другие питомцы
- Image: Photos of other pets: ВЕСНУШКА (a rabbit), НЕБО (a small dog), ТУСЯ (a cat), МАНЯ (a dog).
- Footer: Сотрудничество, Вопросы и ответы, Помощь, Волонтерство, О нас, Оплата: монобанк, Visa, MasterCard, МИР, Одесса

Рисунок – 1.5.1.1.2.6. Прототип та фінальний дизайн функції FR7.1

Для функції «FR8» OUTPUT-інтерфейса користувача.

Создание приюта

Название приюта
ФОП
Контакты
Контакты
Подтверждение пароля
+ +

Обратите внимание, что подтверждение реальности приюта подтверждается лично нашими волонтерами в течении 3х рабочих дней. Спасибо за внимание!

Готово

Сотрудничество
Помощь
Волонтерство
О нас
Вопросы и ответы
Донаты
Другое
Оплата
Мопо
Оплата
McDonald's
Оплата
Одесса

Животные - братья наши меньшие

Создание приюта

Название приюта
ФОП
Контакты
Контакты
Прикрепите документы
+ +

Обратите внимание, что подтверждение реальности приюта подтверждается лично нашими волонтерами в течении 3х рабочих дней. Спасибо за внимание!

Готово

Сотрудничество
Помощь
Волонтерство
О нас
Вопросы и ответы
Донаты
Другое
Оплата
Одесса

Рисунок – 1.5.1.1.2.7. Прототип та фінальний дизайн функції FR8
Для функції «FR9» OUTPUT-інтерфейса користувача.

Название 1

О нас:
Текущий устраним и развитие структуры, требует от нас анализа модели развития. Задачи и симптомы в настоящий момент не являются актуальными для нас, но это не означает, что мы не можем помочь вам. Значимость этих проблем настолько очевидна, что устранение и развитие структуры в значительной степени облегчает задачу. Важно отметить, что устранение и развитие структуры в значительной степени облегчает задачу.

Наши постояльцы

Волчак	Небо	Туса	Макс
Рыжик	Бася	Бобби	Таня

Помощь

Пожертвовать
Подробнее

Сотрудничество
Помощь
Волонтерство
О нас
Вопросы и ответы
Донаты
Другое
Оплата
Мопо
Оплата
McDonald's
Оплата
Одесса

НАЗВАНИЕ ПРИЮТА

О НАС:
Текущий устраним и развитие структуры, требует от нас анализа модели развития. Задачи и симптомы в настоящий момент не являются актуальными для нас, но это не означает, что мы не можем помочь вам. Значимость этих проблем настолько очевидна, что устранение и развитие структуры в значительной степени облегчает задачу. Важно отметить, что устранение и развитие структуры в значительной степени облегчает задачу.

НАШИ ПОСТОЯЛЬЦЫ

Белочка	Небо	Туса	Макс
Рыжик	Бася	Бобби	Таня

Помощь

Пожертвовать
Подробнее

Сотрудничество
Помощь
Волонтерство
О нас
Вопросы и ответы
Донаты
Другое
Оплата
Мопо
Оплата
McDonald's
Оплата
Одесса

Рисунок – 1.5.1.1.2.8. Прототип та фінальний дизайн функції FR9

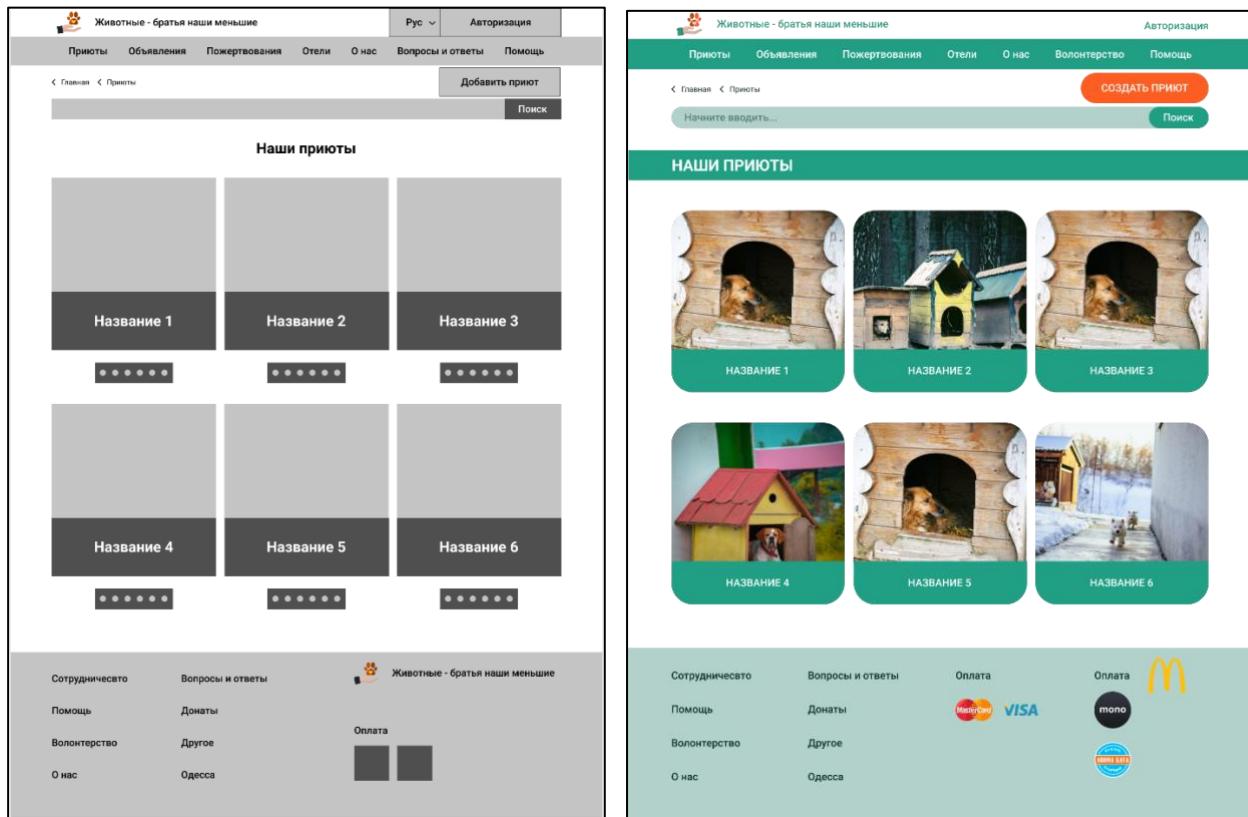


Рисунок – 1.5.1.1.2.9. Для функції «FR4.1» OUTPUT-інтерфейса користувача.

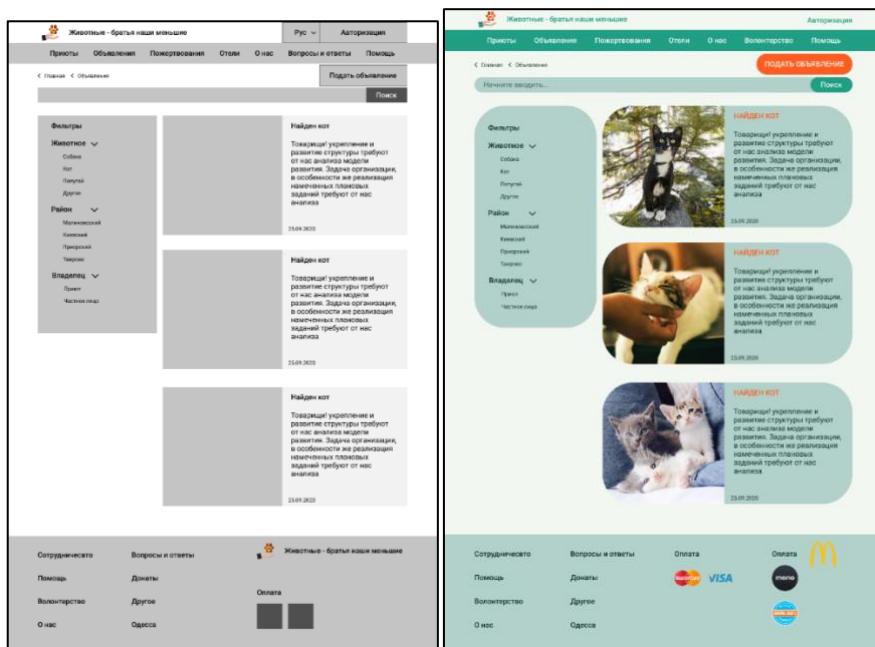


Рисунок – Рисунок – 1.5.1.1.2.9. Прототип та фінальний дизайн функції FR4.1 для функції «FR4.1.1» OUTPUT-інтерфейса користувача.

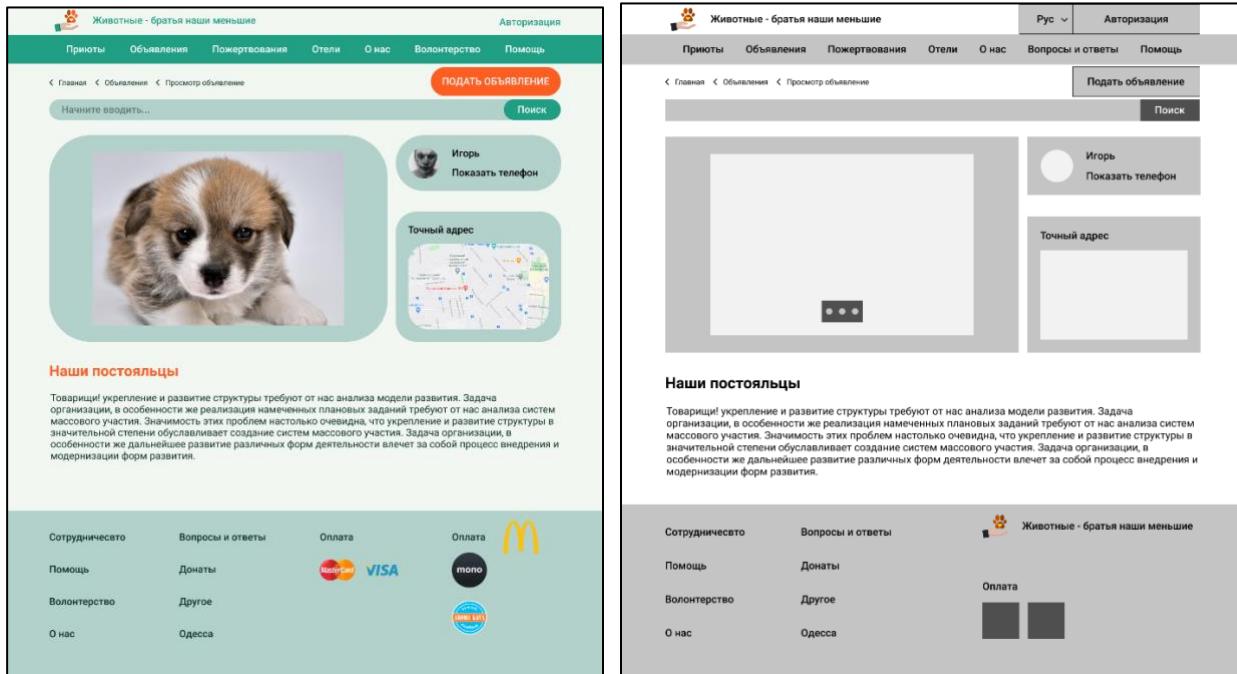


Рисунок – Рисунок – 1.5.1.1.2.10. Прототип та фінальний дизайн функції FR4.1.1

1.5.1.2 Опис інтерфейсу із зовнішніми пристроями

F0 - Desktop, смартфон, принтер (можливість друку)

FR1 – Desktop, смартфон

FR2 – Desktop, смартфон

FR3 – Desktop, смартфон

FR4 – Desktop, смартфон, принтер (можливість друку)

FR5 – Desktop

FR6 – Desktop, смартфон, принтер (можливість друку)

FR7 – Desktop, смартфон, принтер (можливість друку)

FR8 – Desktop, смартфон

FR9 – Desktop, смартфон, принтер (можливість друку)

FR10 – Desktop, смартфон, принтер (можливість друку)

FR11 – Desktop, смартфон

FR12 – Desktop, смартфон

1.5.1.3 Опис програмних інтерфейсів

В даному ПП можна використовувати будь-які операційні системи, не є принциповим наявність певної конкретної версії (ПП адоптовано під усі).

Зовнішні програмні продукти, з якими буде взаємодіяти ПП:

- Google (для авторизації / реєстрації)

- Facebook (для авторизації / реєстрації)

Зовнішні API-сервіси, що використовуються в ПП:

- API платіжної системи LiqPay для здійснення пожертвування

1.5.1.4 Опис інтерфейсів передачі інформації

Інтерфейси передачі інформації для реалізації більшості функцій ПП:

○ провідні:

- Ethernet
- GigabitEthernet
- Інше

○ безпровідні:

- Wi-Fi

1.5.1.5 Опис атрибутів продуктивності

Ідентифікатор функції	Максимальний час реакції ПП на дії користувача, секунди
FR0	5 сек
FR1	4 сек
FR2	3 сек
FR3	1,5 сек
FR4	4 сек

FR5	3 сек
FR6	2 сек
FR7	1 сек
FR8	2 сек
FR9	5 сек
FR10	3,5 сек
FR11	4 сек
FR12	2 сек

2 Планування процесу розробки програмного продукту

2.1 Планування ітерацій розробки програмного продукту

З метою забезпечення для вимог таких рекомендацій IEEE-стандарту, як необхідність, корисність при експлуатації, здійсненність функціональних вимог до ПП, були визначені функціональні пріоритети, які будуть використані при плануванні ітерацій розробки ПП.

2.2 Концептуальний опис архітектури програмного продукту

Концептуальний опис архітектури програмного продукту зображене на рис. 2.2

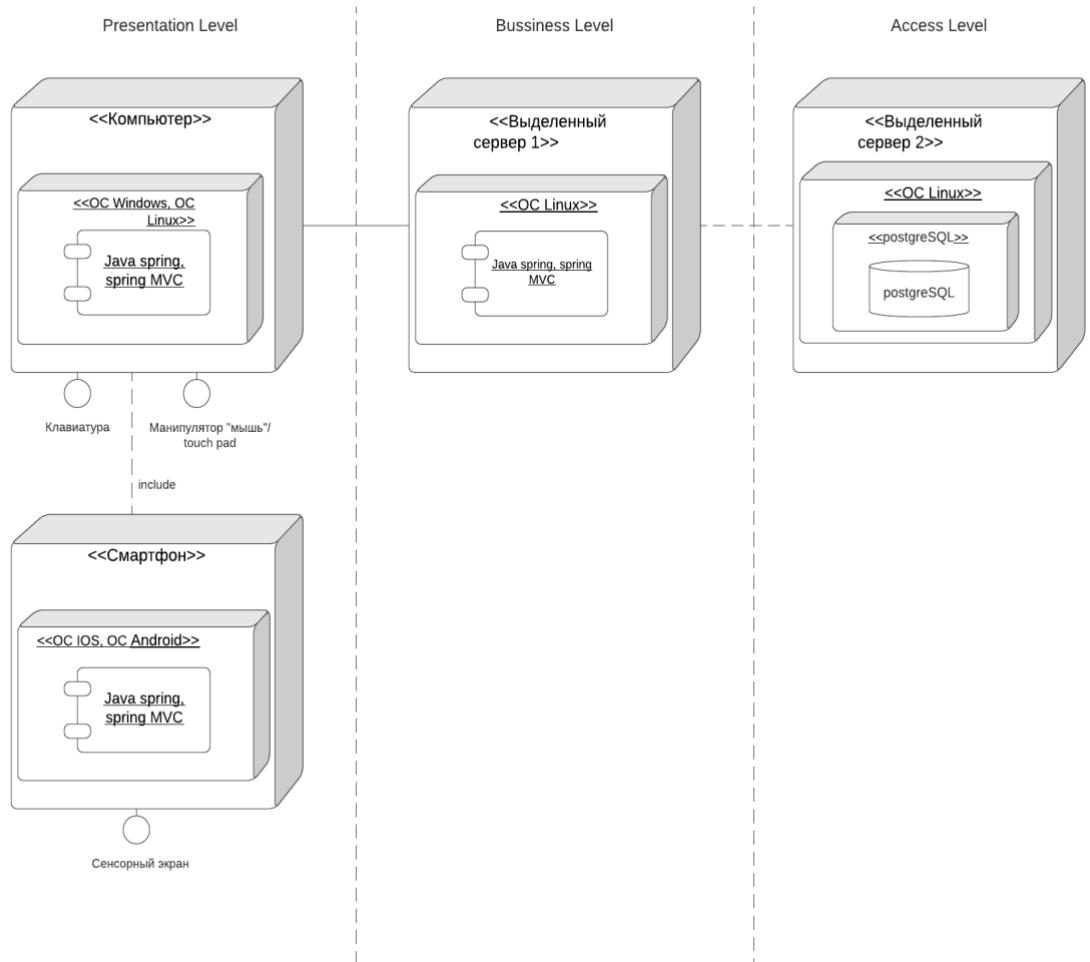


Рисунок – 2.2. Концептуальний опис архітектури програмного продукту

2.3 План розробки програмного продукту

З метою забезпечення для вимог таких рекомендацій IEEE-стандарту, як необхідність, корисність при експлуатації, здійсненність функціональних вимог до ПП, було визначено функціональні пріоритети, які будуть використані при плануванні ітерацій розробки ПП.

При створенні пріоритетів необхідно врахувати:

- сценарні залежності між прецедентами, до яких належать функції, на основі аналізу пунктів передумов початку роботи прецедентів, вказаних в описі сценаріїв роботи прецедентів;
- вплив роботи прецеденту, до якого належить функція, на досягнення

мети ПП, наприклад у відсotках, на основі аналізу пунктів гарантій успіху, вказаних в описі сценаріїв роботи прецедентів.

Сценарні залежності будуть перетворені у відповідні функціональні залежності. Вплив роботи прецеденту буде поширено на всі підлеглі функції ієрархії. При визначенні пріоритетів рекомендується використовувати наступні позначки:

- M (Must) – функція повинна бути реалізованою у перших ітераціях за будь-яких обставин;
- S (Should) – функція повинна бути реалізованої у перших ітераціях, якщо це взагалі можливо;
- C (Could) – функція може бути реалізованої, якщо це не вплине негативно на строки розробки;
- W (Want) – функція може бути реалізованої у наступних ітераціях.

Опису представлено в таблиці 2.3

Таблиця 2.3 – Опис функціональних пріоритетів

Назва	Функціональні залежності	Вплив на досягнення мети, %	Пріоритет функції
FR0	-	100%	M
FR1	FR0	25%	W
FR2	FR0	25%	W
FR3	FR0	75%	M
FR4	FR0, FR1	75%	M
FR5	FR3,FR4	0%	C
FR6	FR0	50%	S

FR7	FR0	75%	M
FR8	FR1, FR2,	75%	M
FR9	FR8	50%	S
FR10	FR8, FR9	50%	S
FR11	FR1	0%	C
FR12	FR11	0%	C

2.3.1 Оцінка трудомісткості розробки програмного продукту

Визначення нескорегованого показника UUCP (Unadjusted Use Case Points)

1. Визначення вагових показників акторів А

Всі актори діляться на три типи: прості, середні і складні.

Простий актор представляє зовнішню систему з чітко визначеним Програмним інтерфейсом.

Середній актор представляє або зовнішню систему, що взаємодіє з ПП за допомогою мережевих протоколів, або особистість, що користується текстовим інтерфейсом (наприклад, алфавітно-цифровим терміналом).

Складний актор представляє особистість, що користується графічним інтерфейсом. Загальна кількість акторів кожного типу помножується на відповідний ваговий коефіцієнт, потім обчислюється загальний ваговий показник (табл.2.3.1.1)

Таблиця - 2.3.1.1. Вагові коефіцієнти акторів

Тип актора	Ваговий коефіцієнт
Простий	1
Середній	2
Складний	3

Визначення UUCP

$$A = 1*1 + 0*2 + 2*3 = 7$$

$$UC = 5*5 + 0*10 + 0*15 = 25$$

$$UUCP = A + UC = 32$$

Технічна складність проекту (TCF - Technical Complexity Factor) обчислюється з урахуванням показників технічної складності.

Кожному показнику присвоюється значення ST_i в діапазоні від 0 до 5:

- 1) 0 означає відсутність значимості показника для даного проекту;
- 2) 5 - високу значимість.

Показники технічної складності проекту TCF представлено в таблиці

2.3.1.2

Таблиця – 2.3.1.2. Показники технічної складності проекту

Показник	Опис	Вага
T1	Розподілена система	3
T2	Висока продуктивність (пропускна здатність)	3
T3	Робота кінцевих користувачів в режимі он-лайн	1
T4	Складна обробка даних	2
T5	Повторне використання коду	3
T6	Простота установки	1
T7	Простота використання	2
T8	Переносимість	2
T9	Простота внесення змін	3

T10	Паралелізм	1,5
T11	Спеціальні вимоги до безпеки	1,5
T12	Безпосередній доступ до системи з боку зовнішніх користувачів	1
T13	Спеціальні вимоги до навчання користувачів	1

Значення TCF

$$TCF1 = 0,6 + (0,01 * (3 * 3)) = 0,69$$

$$TCF2 = 0,6 + (0,01 * (4 * 3)) = 0,72$$

$$TCF3 = 0,6 + (0,01 * (3 * 1)) = 0,63$$

$$\text{TCF4} = 0,6 + (0,01 * (3 * 2)) = 0,66$$

$$TCF5 = 0,6 + (0,01 * (1 * 3)) = 0,63$$

$$TCF6 = 0,6 + (0,01 * (5 * 1)) = 0,65$$

$$TCF7 = 0,6 + (0,01 * (5 * 2)) = 0,7$$

$$\text{TCF8} \equiv 0.6 + (0.01 * (3 * 2)) \equiv 0.66$$

$$TCF9 = 0.6 + (0.01 * (5 * 3)) = 0.75$$

$$\text{TCF10} = 0.6 + (0.01 * (3 * 1.5)) = 0.645$$

$$TCF11 = 0.6 \pm (0.01 * (5 * 1.5)) = 0.675$$

Рівень кваліфікації розробників (EF - Environmental Factor) обчислюється з урахуванням наступних показників (табл. 2.3.1.3).

Таблиця – 2.3.1.3. Показники рівня кваліфікації розробників

Показник	Опис	Вага
F1	Знайомство з технологією	1
F2	Досвід розробки додатків	2
F3	Досвід застосування об'єктно-орієнтованого підходу	3
F4	Наявність ведучого аналітика	2
F5	Мотивація	1
F6	Стабільність вимог	2
F7	Часткова зайнятість	2
F8	Складні мови програмування	2

TCF13 =

Обчислення значення EF:

$$EF1 = 1,4 + (-0,03 * (3 * 1)) = 1,31$$

$$EF2 = 1,4 + (-0,03 * (3 * 2)) = 1,22$$

$$EF3 = 1,4 + (-0,03 * (5 * 3)) = 0,95$$

$$EF4 = 1,4 + (-0,03 * (3 * 2)) = 1,22$$

$$EF5 = 1,4 + (-0,03 * (5 * 1)) = 1,25$$

$$EF_6 = 1,4 + (-0,03 * (5 * 2)) = 1,1$$

$$EF_7 = 1,4 + (-0,03 * (3 * 2)) = 1,22$$

$$EF_8 = 1,4 + (-0,03 * (3 * 2)) = 1,22$$

Остаточне значення UCP (Use Case Points)

$$UCP = UUCP * TCF * EF = 32 * 8.66 * 9.49 = 2629$$

2.3.2 Визначення дерева робіт з розробки програмного продукту

При створенні дерева робіт (Work BreakDown Structure- WBS)

використовується дерево функцій, яке було створено раніше.

Кожна функція 1-го рівня ієрархії перетворюється в Work Package (WP)

Кожна функція 2-го рівня ієрархії перетворюється в Work Task (WT).

Для кожної задачі визначаються підзадачі - Work SubTask (WST) з урахуванням базових процесів розробки програмних модулів: проектування, конструювання, модульне тестування, збірка та системне тестування.

WBS представлено на рисунку 2.3.2 .1 – 2.3.2.5.

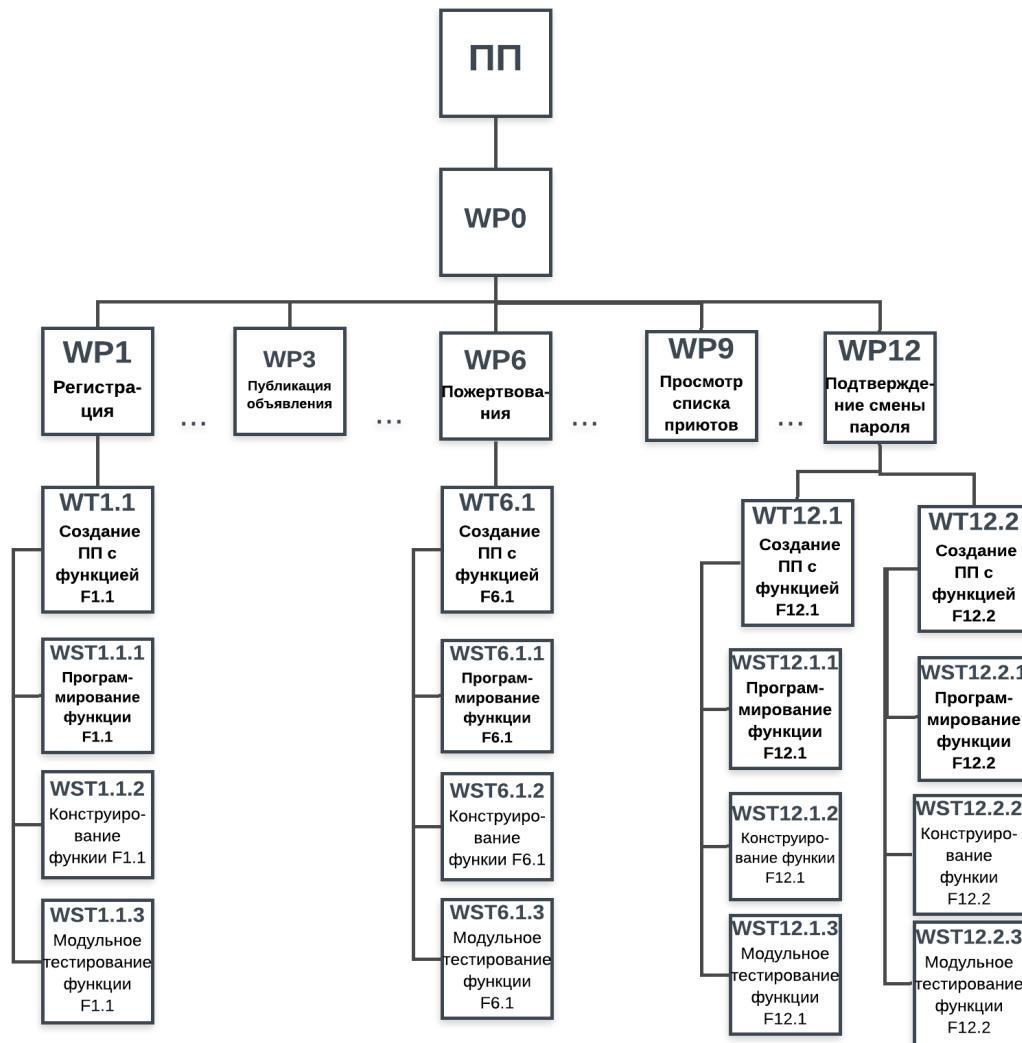


Рисунок – 2.3.2.1

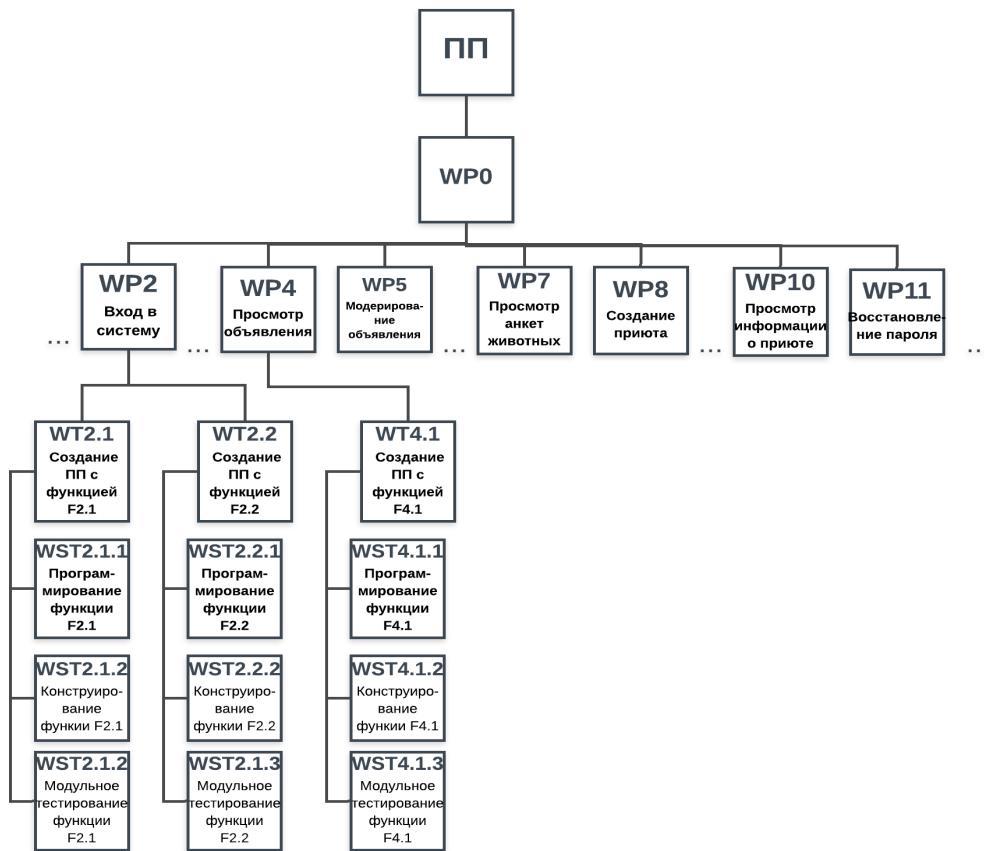


Рисунок – 2.3.2.2

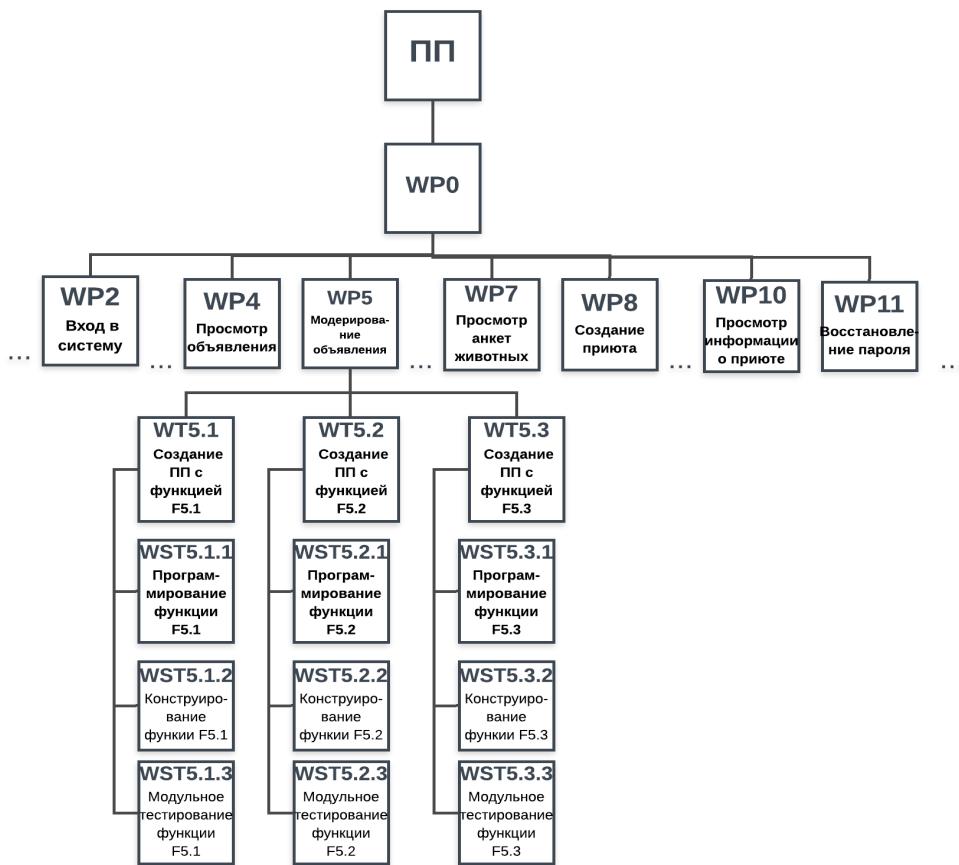


Рисунок – 2.3.2.3

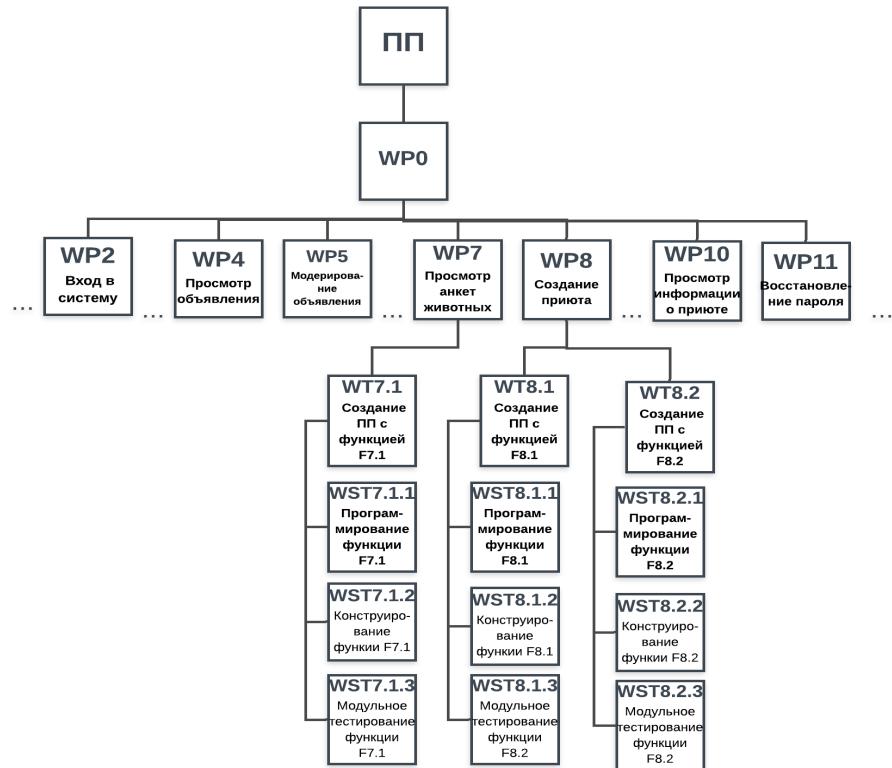


Рисунок – 2.3.2.4

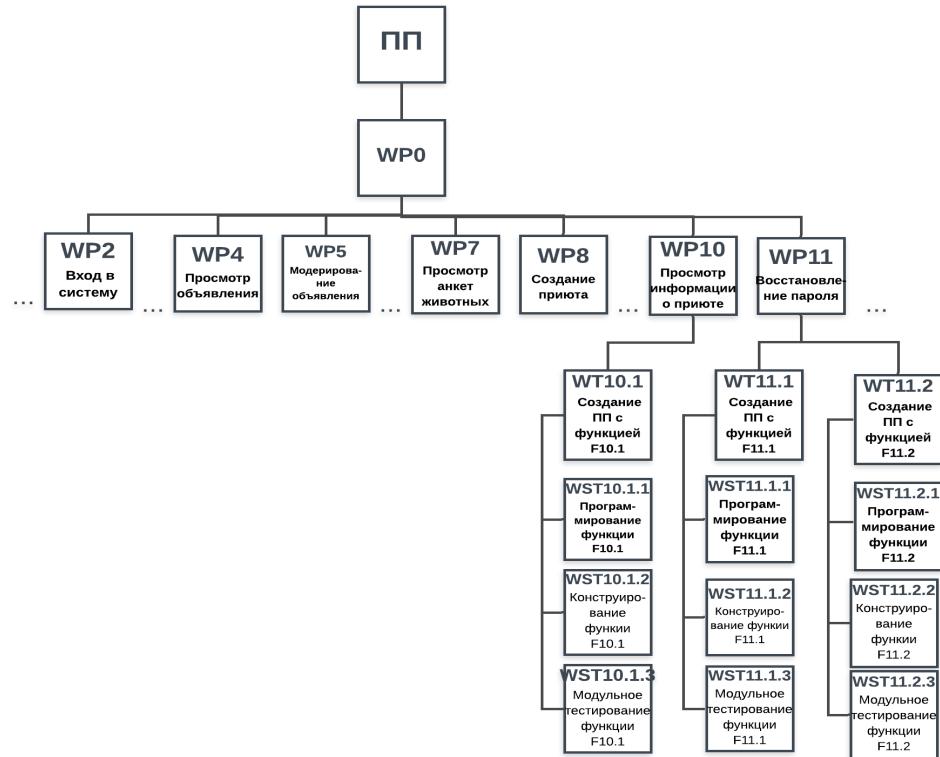


Рисунок – 2.3.2.5

2.3.3 Графік робіт з розробки програмного продукту

2.3.3.1 Таблиця з графіком робіт

Для кожної підзадачі визначається виконавець, що фіксується у вигляді таблиці, яка представлена в таблиці 2.3.3.1

Таблиця – 2.3.3.1

WST	Дата початку	Дні	Дата завершення	Виконувач
WST1.1.1	15.10.2020	5	19.10.2020	Соловйова Д.В.

WST1.1.2	20.10.2020	4	24.10.2020	Соловйова Д.В.
WST1.1.3	25.10.2020	5	30.10.2020	Соловйова Д.В.
WST2.1.1	15.10.2020	5	20.10.2020	Поліщук К.А.
WST2.1.2	21.10.2020	5	26.10.2020	Поліщук К.А.
WST2.1.3	27.10.2020	4	1.11.2020	Поліщук К.А.
WST8.1.1	15.10.2020	6	21.10.2020	Усова А.В.
WST8.1.2	22.10.2020	4	26.10.2020	Усова А.В.
WST8.1.3	27.10.2020	4	2.11.2020	Усова А.В.
WST7.1.1	25.10.2020	7	2.11.2020	Соловйова Д.В.
WST7.1.2	3.11.2020	4	7.11.2020	Соловйова Д.В.
WST7.1.3	7.11.2020	5	12.11.2020	Соловйова Д.В.
WST6.1.1	27.10.2020	7	4.11.2020	Поліщук К.А.
WST6.1.2	5.11.2020	4	9.11.2020	Поліщук К.А.
WST6.1.3	10.11.2020	3	12.11.2020	Поліщук К.А.
WST2.2. 1	3.11.2020	5	8.11.2020	Усова А.В.
WST2.2 .2	9.11.2020	3	11.11.2020	Усова А.В
WST2.2 .3	11.11.2020	2	12.11.2020	Усова А.В.

2.3.3.2 Діаграма Ганта

Приклад діаграми Ганта рис 2.3.3.2

WST	Дата початку	Дні	Дата завершення	Виконувач	15.10.2020	16.10.2020	17.10.2020	18.10.2020	19.10.2020	20.10.2020	21.10.2020	22.10.2020	23.10.2020	2
WST1.1.1	15.10.2020	5	19.10.2020	Соловіова Д.В.										
WST1.1.2	20.10.2020	4	24.10.2020	Соловіова Д.В.										
WST1.1.3	25.10.2020	5	30.10.2020	Соловіова Д.В.										
WST2.1.1	15.10.2020	5	20.10.2020	Поліщук К.А.										
WST2.1.2	21.10.2020	5	26.10.2020	Поліщук К.А.										
WST2.1.3	27.10.2020	4	1.11.2020	Поліщук К.А.										
WST8.1.1	15.10.2020	6	21.10.2020	Усова А.В.										
WST8.1.2	22.10.2020	4	26.10.2020	Усова А.В.										
WST8.1.3	27.10.2020	4	2.11.2020	Усова А.В.										
WST7.1.1	25.10.2020	7	2.11.2020	Соловіова Д.В.										
WST7.1.2	3.11.2020	4	7.11.2020	Соловіова Д.В.										
WST7.1.3	7.11.2020	5	12.11.2020	Соловіова Д.В.										
WST6.1.1	27.10.2020	7	4.11.2020	Поліщук К.А.										
WST6.1.2	5.11.2020	4	9.11.2020	Поліщук К.А.										
WST6.1.3	10.11.2020	3	12.11.2020	Поліщук К.А.										
WST2.2.1	3.11.2020	5	8.11.2020	Усова А.В.										
WST2.2.2	9.11.2020	3	11.11.2020	Усова А.В.										
WST2.2.3	11.11.2020	2	12.11.2020	Усова А.В.										

Рисунок 2.3.3.2

Посилання на діаграму

https://docs.google.com/spreadsheets/d/1UOYx_j5HAKSoqu-RJuk9WXBN0j_Wv7Z8MYIEHCa1NQM/edit?usp=sharing

3. Проектування програмного продукту

3.1 Концептуальне та логічне проектування структур даних програмного продукту

3.1.1 Концептуальне проектування на основі UML-діаграми концептуальних класів

Використовуючи кроки основного успішного та альтернативного сценаріїв роботи прецедентів ПП, було спроектовано UML-діаграми концептуальних класів.

Моделювання проведено з урахуванням наступної послідовності кроків.

1. Визначити імена класів.
2. Визначити імена атрибутів класів.
3. Визначити зв'язку між класами (узагальнення, іменовані асоціації, агреговані асоціації).
4. Для зв'язків-асоціацій визначити назви, кратність і можливість агрегації.

5. Створити UML-діаграму класів в будь-якому графічному редакторі.

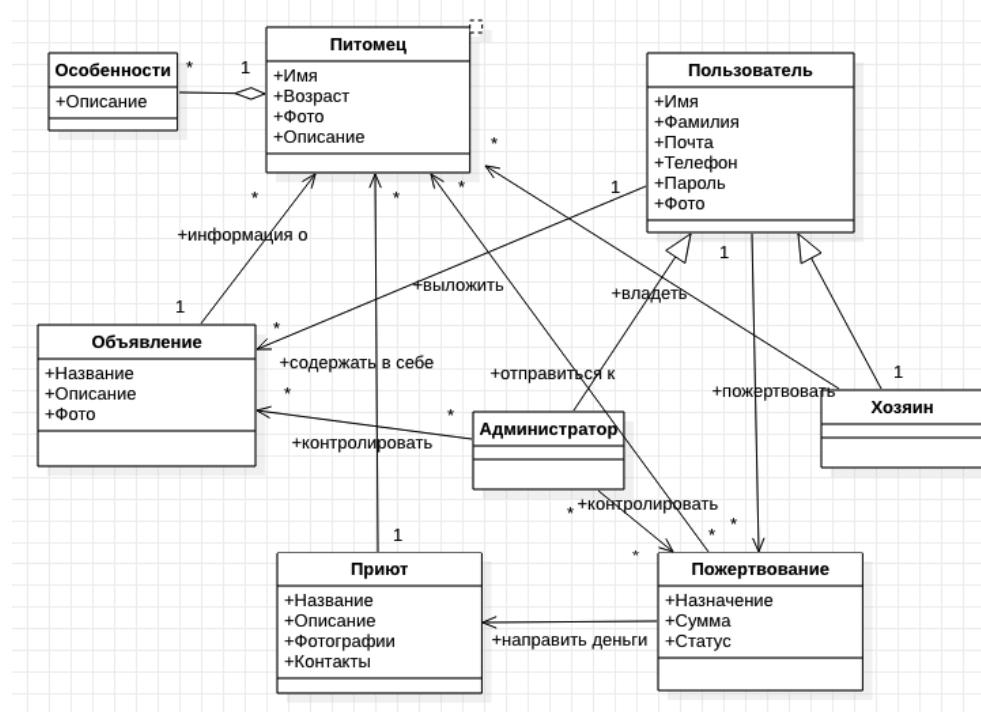


Рисунок - 3.1.1. UML-діаграма класів

3.1.2 Логічне проектування структур даних

UML-діаграма концептуальних класів була перетворена в опис структур даних з використанням моделі, яка була обрана в концептуальному описі архітектури ПП,

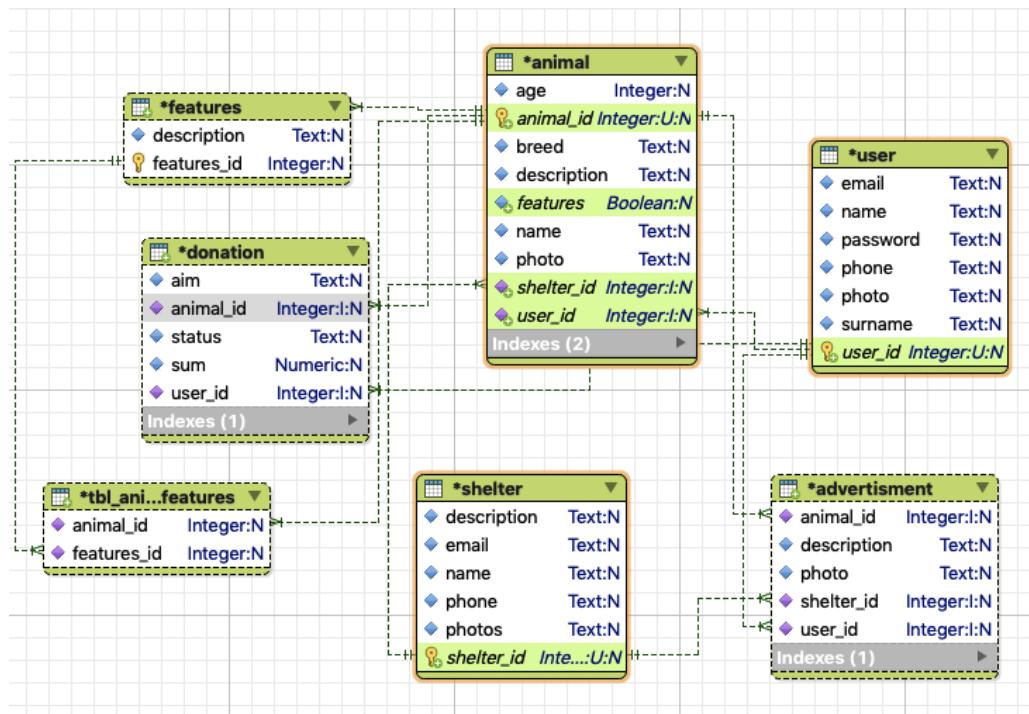
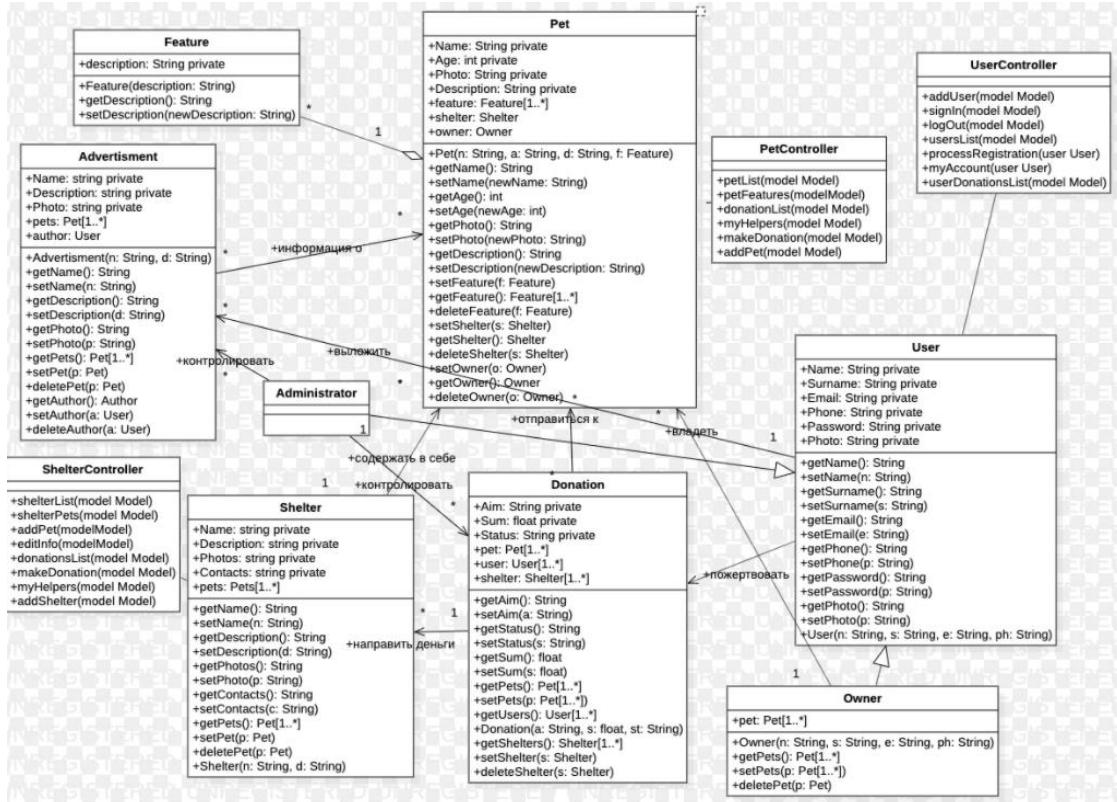


Рисунок – 3.1.2. Схема БД

3.2 Проектування програмних класів

- На основі UML-діаграми концептуальних класів були спроектовані програмні класи:
 - англійські або транслітерацію україномовних назви класів та їх атрибутів;
 - абстрактні класи, їх класи-нащадки та інші класи;
 - зв`язки між класами (наслідування, іменована асоціація, агрегатна асоціація, або агрегація, композитна асоціація або композиція) та їх кратності;
 - атрибути класів з типами даних (цілий, дійсний, логічний, перелічуваний, символний з урахуванням розміру), та типом видимості (публічний, захищений, приватний);
 - методи-конструктори ініціалізації екземплярів об'єктів класу, set-методи та get-методи для доступу до атрибутів класу

2. На основі ієрархії функцій додайте до класів методи доступу, які будуть реалізовувати алгоритм роботи функцій.



3.3 Проектування алгоритмів роботи методів програмних класів

Процес розробки алгоритмів:

1. З усіх методів виділили ті, що мають операції доступу до БД або містять керуючі умови (if-then-else, while).
 2. Опишіть алгоритм роботи у вигляді UML-діаграми активності:
 - кожний опис алгоритму представили в текстовому файлі на мові

PlantUML, використовуючи веб-редактор.

Алгоритм роботи методу «Shelter» мовою PlantUML:

@startuml

start

:Создание формы для регистрации приюта;
:Создание тестов;
repeat
repeat
:Отправка запроса на обработку данных;
:Тестирование данных на корректность;
repeat while(Данные не прошли проверку)
:Проверка существования приюта;
backward:Информирование пользователя об ошибке;
repeat while (Данные не прошли проверку)
:Информирование пользователя об успешной проверке;
:Передача данных о приюте в БД;
:Сохранение данных в БД;
note left
Insert into shelter values(передаваемые параметры)
insert into images_shelter values (1, lo_import('/path/to/image.jpg'));
end note
:Информирование пользователя об успешной регистрации приюта;
stop
@enduml

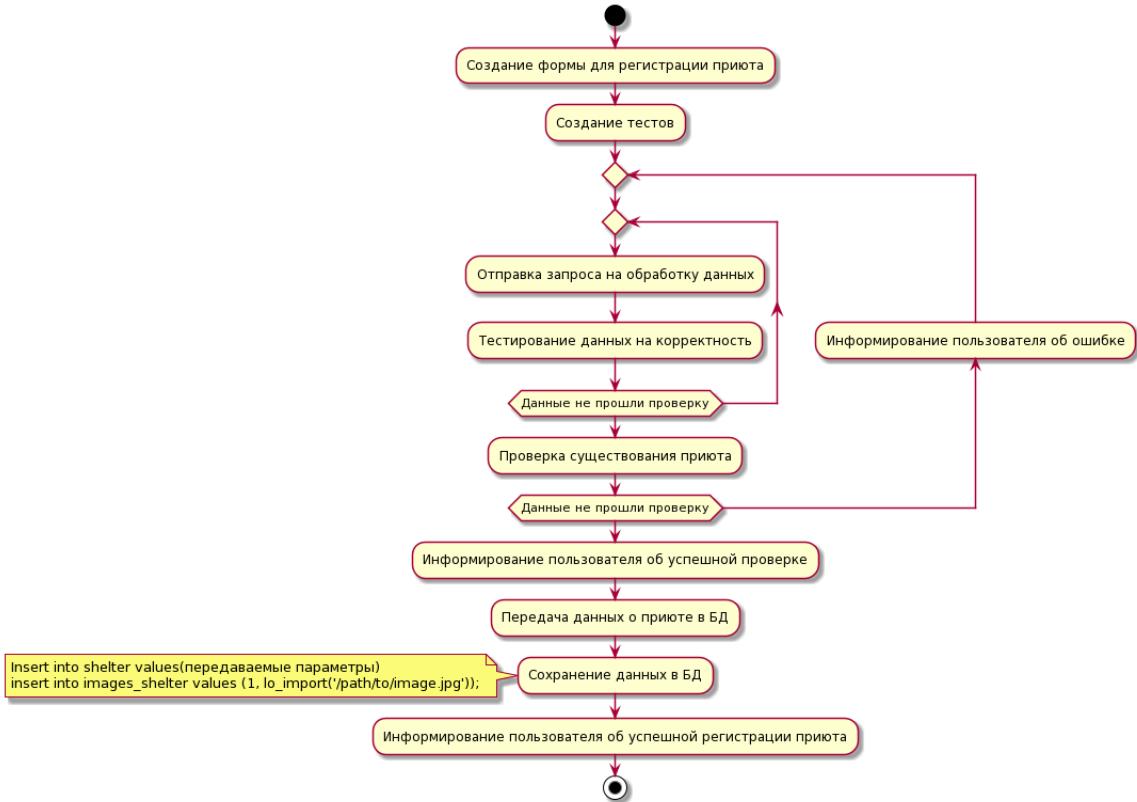


Рисунок – 3.3.1. Алгоритм методу «Shelter»

Алгоритм работы методу «User» мовою PlantUML:

```

@startuml

start
    :Создание страницы "Аккаунт";
    :Вывод на странице данных, введённых при регистрации;
    note left
        Вывод данных из БД по логину;
    end note
    :Создание полей для редактирования;
    note left
        Изменение данных в БД на те, которые были введены пользователем;
    end note
stop

```


@enduml



Рисунок – 3.3.1. Алгоритм методу «User»

3.4 Проектування тестових наборів методів програмних класів

Назва функції	№ тесту	Вхідні дані	Очікуваний результат тесту
AddShelter	1	{}	{ "status":404, "message":"Errors", "errors":[{ "errorCode":404009, "errorMessage":"shelter name not found" }, { "errorCode":404000, "errorMessage":"email not found" }, { "errorCode":404011, "errorMessage":"phone not found" }] }
	2	{ "name":"Sheler1" }	{ "status":404, "message":"Errors", "errors":[{ "errorCode":404000, "errorMessage":"email not found" }] }

			<pre> }, { "errorCode":404011, "errorMessage":"phone not found" }] } } } </pre>
3	<pre> { "name":"Sheler1", "email":"john.watson@mail.com" } </pre>	<pre> { "status":404, "message":"Errors", "errors":[{ "errorCode":404011, "errorMessage":"phone not found" }] } </pre>	
4	<pre> { "name":"1", "email":"john.watson@mail.com", "phone":"+380551112211" } </pre>	<pre> { "status":409, "message":"Errors", "errors":[{ "errorCode":409004, "errorMessage":"name is too short" }] } </pre>	
5	<pre> { "name":"Sheler1", "email":"john.watson@mail.com", "phone":"+380551112211" } </pre>	<pre> { "status":200, "message":"OK", "data":{ "shelterId":"_id" } } </pre>	
ChangeShelter	1	{}	<pre> { "status":404, "message":"Errors", "errors":[{ "errorCode":404009, "errorMessage":"shelter name not found" }, { "errorCode":404011, "errorMessage":"phone not found" }] } </pre>

			<pre>{ "errorCode":404000, "errorMessage":"email not found" }, { "errorCode":404011, "errorMessage":"phone not found" }]</pre>
	2	<pre>{ "name":"Sheler1" }</pre>	<pre>{ "status":404, "message":"Errors", "errors":[{ "errorCode":404000, "errorMessage":"email not found" }, { "errorCode":404011, "errorMessage":"phone not found" }] }</pre>
	3	<pre>{ "name":"Sheler1", "email":"john.watson@mail.com" }</pre>	<pre>{ "status":404, "message":"Errors", "errors":[{ "errorCode":404011, "errorMessage":"phone not found" }] }</pre>
	4	<pre>{ "name":"1", "email":"john.watson@mail.com", "phone":"+380551112211" }</pre>	<pre>{ "status":409, "message":"Errors", "errors":[{ "errorCode":409004, "errorMessage":"name is too short" }] }</pre>

	5	<pre>{ "name": "Sheler1", "email": "john.watson@mail.com", "phone": "+380551112211" }</pre>	<pre>{ "status": 200, "message": "OK", "data": { "shelterId": "_id" } }</pre>
ProfileChange	1	<pre>{}</pre>	<pre>{ "status": 404, "message": "Errors", "errors": [{ "errorCode": 404002, "errorMessage": "name not found" }, { "errorCode": 404003, "errorMessage": "surname not found" }, { "errorCode": 404000, "errorMessage": "email not found" }, { "errorCode": 404001, "errorMessage": "password not found" }] }</pre>
	2	<pre>{ "name": "John" }</pre>	<pre>{ "status": 404, "message": "Errors", "errors": [{ "errorCode": 404003, "errorMessage": "surname not found" }, { "errorCode": 404000, "errorMessage": "email not found" }, { "errorCode": 404001, "errorMessage": "password not found" }] }</pre>

			<pre>] } } </pre>
	3	<pre> { "surname":"Watson" } </pre>	<pre> { "status": 404, "message": "Errors", "errors": [{ "errorCode": 404002, "errorMessage": "name not found" }, { "errorCode": 404000, "errorMessage": "email not found" }, { "errorCode": 404001, "errorMessage": "password not found" }] } </pre>
	4	<pre> { "email":"john.watson@mail.com" } </pre>	<pre> { "status": 404, "message": "Errors", "errors": [{ "errorCode": 404002, "errorMessage": "name not found" }, { "errorCode": 404003, "errorMessage": "surname not found" }, { "errorCode": 404001, "errorMessage": "password not found" }] } </pre>
	5	<pre> { "password":"johnwatson11" } </pre>	<pre> { "status": 404, "message": "Errors", "errors": [{ "errorCode": 404002, "errorMessage": "name not found" }] } </pre>

			<pre> }, { "errorCode": 404003, "errorMessage": "surname not found" }, { "errorCode": 404000, "errorMessage": "email not found" }] } } } </pre>
	6	<pre> { "name": "John", "surname": "Watson", "email": "email@mail.com", "password": "johnwatson11" } </pre>	<pre> { "status": 403, "message": "Errors", "errors": [{ "errorCode": 403000, "errorMessage": "email already exists" }] } </pre>
	7	<pre> { "name": "John", "surname": "Watson", "email": "john.watson@mail.com", "password": "john" } </pre>	<pre> { "status": 409, "message": "Errors", "errors": [{ "errorCode": 409000, "errorMessage": "password is too short" }] } </pre>
	8	<pre> { "name": "John", "surname": "Watson", "email": "john.watson@mail.com", "password": "johnwatson1111111111" } </pre>	<pre> { "status": 409, "message": "Errors", "errors": [{ "errorCode": 409001, "errorMessage": "password is too long" }] } </pre>
	9	<pre> { "name": "John", "surname": "Watson", } </pre>	<pre> { "status": 409, "message": "Errors", } </pre>

		<pre> "email":"john.watson@mail.com", "password":"johnwatson" } </pre>	<pre> "errors": [{ "errorCode":409002, "errorMessage":"the password must contain at least one digit" }] } </pre>
	10	<pre> { "name":"John", "surname":"Watson", "email":"john@mail", "password":"johnwatson11" } </pre>	<pre> { "status":409, "message":"Errors", "errors": [{ "errorCode":409003, "errorMessage":"invalid email" }] } </pre>
	11	<pre> { "name":"John", "surname":"Watson", "email":"john.watson@mail.com", "password":"johnwatson11" } </pre>	<pre> { "status":200, "message":"OK" } </pre>

4 Конструювання програмного продукту

4.1 Особливості конструювання структур даних

Архітектура програмного забезпечення – це представлення системи програмного забезпечення, яке дає інформацію про складові компоненти системи, про взаємозв'язки між цими компонентами і правила, що регламентують ці взаємозв'язки. Це процес, що передбачає послідовність дій для створення або зміни архітектури системи, і проекту системи по цій архітектурі, з врахуванням безлічі обмежень.

4.1.1 Особливості інсталяції та роботи з СУБД

Розглянуто створення бази даних для роботи рекламного агентства з використанням СУБД PostgreSQL.

PostgreSQL називають найбільш досконалою з наявних сьогодні СУБД з відкритим вихідним кодом. Таку репутацію вона завоювала завдяки зусиллям розробників протягом десятиліть. Будучи повнофункціональної реляційної СУБД з відкритим вихідним кодом, PostgreSQL володіє багатьма якостями, необхідними для підтримки найважливіших додатків з великим числом транзакцій.

Діаграма баз даних є структурою частиною системи баз даних описаною формальною мовою, яка підтримується системою керування баз даних і відноситься до організації даних з розподілом на таблиці.

4.1.2 Особливості створення структур даних

Розглянемо SQL-запити для створення таблиці тварин:

```
CREATE SEQUENCE IF NOT EXISTS animals_ids; CREATE
TABLE IF NOT EXISTS pets (id int PRIMARY KEY
DEFAULT nextval('animals_ids'), animal_id int,
age int NOT NULL CHECK (salary > 0),
breed varchar,
description varchar,
name varchar NOT NULL,
photo varchar,
FOREIGN KEY (shelter_id) REFERENCES shelters (id),
FOREIGN KEY (user_id) REFERENCES users (id);
```

Таблиця об'єднує інформацію про тварину, притулки, та користувачів через додавання первісних ключів цих таблиць як зовнішніх для таблиці вакансій.

Для зберігання ключів використовується тип даних integer, бо операції порівняння та пошуку швидше за все працюють з цифровими типами. Таким

чином кожен елемент таблиці може бути знайдений за лінійний час роботи алгоритму пошуку. Формування унікальних ідентифікаторів робиться через елемент sequence відповідно до архітектури PostgreSQL.

База даних складається з наступних таблиць: Тварини, Користувачі, Об'яви, Притулки, Пожертвування, Ознаки та допоміжної таблиці Тварина_Ознака, що забезпечує зв'язок «багато до багатьох» між таблицями «Тварини» та «Ознаки».

Таблиця «Користувачі» містить інформацію про усіх користувачів системи. У ній присутні такі поля, як унікальний ідентифікатор анкети (первинний ключ), ім'я, прізвище, номер телефону, фото, пароль та пошта (ім'я користувача) для входу у систему.

Таблиця «Тварини» містить інформацію про анкету тварини. У ній присутні такі поля, як унікальний ідентифікатор анкети, ім'я тварини, її вік, код анкети (первинний ключ), ознаки, фото тварини, код притулку (вторинний ключ), код користувача (вторинний ключ).

Таблиця «Пожертвування» містить інформацію про пожертвування дітям. У ній присутні такі поля, як ім'я та прізвище спонсора, напрямок пожертвування, сума пожертвування або інша його фіксація, статус виконання пожертвування організацією, дата надходження пожертвування та дата виконання організацією, код тварини (вторинний ключ), ідентифікатор користувача (вторинний ключ).

Таблиця «Притулки» містить інформацію по конкретному притулку. У ній присутні такі поля, як унікальний ідентифікатор притулку(первинний ключ), опис, пошта, телефон та фото.

Таблиця «Об'ява» зберігає в собі інформацію про об'яву: унікальний ідентифікатор (первинний ключ), фото, опис, ідентифікатор користувача (вторинний ключ), ідентифікатор тварини (вторинний ключ), ідентифікатор притулку (вторинний ключ).

4.2 Особливості конструювання структур даних

Для реалізації структур, описаних у розділі вище, було обрано створення веб-застосунку, де і буде реалізовано структури даних за допомогою програмних класів.

4.2.1 Особливості роботи з інтегрованим середовищем розробки

Інформаційна система «Лапка допомоги» для допомоги притулкам представляє собою клієнт-серверне застосування.

Програмне забезпечення розроблялося у середовищі IntelliJ IDEA. Там було створено проект Spring Boot з модулем Thymeleaf.

Для реалізації серверної частини спроектованої системи була обрана мова об'єктно-орієнтованого програмування Java із використанням фреймворку Spring MVC. Спираючись на шаблон модель-уявлення-контролер, фреймворк Spring MVC допомагає будувати веб-додатки, так само гнучкі і слабо пов'язані, як сам фреймворк Spring.

Фронт-енд системи реалізований з використанням фреймворку Bootstrap, який включає в себе HTML- і CSS- шаблони оформлення різних компонентів веб-інтерфейсу.

Для реалізації аутентифікації і авторизації в проекті використаний фреймворк Spring Security, який забезпечує безпеку інформації, надає можливість декларативного управління безпекою додатків на основі фреймворку Spring.

Всі дані в створеній системі зберігаються в базі даних PostgreSQL, яка розгорнута на хмарної PaaS-платформі Heroku.

4.2.2 Особливості створення програмної структури з урахуванням спеціалізованого Фреймворку

Моделлю у даній системі служить клас AnimalService, який дозволяє працювати з базою даних за допомогою використання Spring Data JPA.


```

package com.example.demo.animal;

@Service
public class AnimalService {

    private AnimalRepository repository;

    @Autowired
    public void setRepository(AnimalRepository repository) {
        this.repository = repository;
    }

    public void saveAnimal(Animal animal) {
        repository.save(animal);
    }

    public void addAnimal(Animal animal) {
        repository.save(animal);
    }

    Public Animal getChildById(long id)
    {
        return repository.getOne(id);
    }

    public List<Animal> getAllAnimals () {
        return repository.findAll();
    }

    public void deleteAnimal (long id) {
        repository.deleteById(id);
    }
}

```

4.2.3 Особливості створення програмних класів

Сутність (Entity) – це Java-клас, який представляє бізнес-логіку програми та визначає дані, які будуть зберігатися в базі даних і вилучатись з неї. Зазвичай клас суті представляє таблицю в базі даних, поля або властивості класу являють собою колонки в таблиці, а об'єкт суті являє собою один запис у таблиці.

Нижче описана сутність Animal – клас, який містить поля для описання тварини та методи, для роботи з ними (getters/setters).

```

package com.example.demo.animal;

@Entity
@Table(name = "animal")
public class Child {

    public enum Gender {

```

```

        male ("мужской"),
        female ("женский");

    private String name;
    Gender(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return this.name;
    }

}

@Id
@GeneratedValue
private long id;

@Column(name = " name")
@Size(min = 2, max= 50, message = "First name should be from 2 to 50
characters")
private String name;

@Column(name = "year_of_birth")
@Range(min = 2000, max= 2020, message = "Year of birth should be from
2000 to 2020")
private int yearOfBirth;

@Column(name = "gender")
private Gender gender;

@Size(min = 4, max= 6, message = "Code should be from 4 to 6 characters")
@Column(name = "code")
private String code;

@Size(min = 2, max= 50, message = "Hobby should be from 2 to 50
characters")

@Column(name = "age")
private int age;

@Column (name = "photo_name")
private String photoName;

public long getId() {
    return id;
}

public void setId(long id) {
    this.id = id;
}

public String getName() {
    return name;
}

```

```

        public int getYearOfBirth() {
            return yearOfBirth;
        }

        public void setYearOfBirth(int yearOfBirth) {
            this.yearOfBirth = yearOfBirth;
        }

        public Gender getGender() {
            return gender;
        }

        public void setGender(Gender gender) {
            this.gender = gender;
        }

        public String getCode() {
            return code;
        }

        public void setCode(String code) {
            this.code = code;
        }

        public String getPhotoName() {
            return photoName;
        }

        public void setPhotoName(String photoName) {
            this.photoName = photoName;
        }
    }
}

```

4.2.4 Особливості розробки алгоритмів методів програмних класів або процедур/функцій

Усі необхідні функції реалізовано через класи-контролери, які являють собою набор функцій для класів. Контролер - це компонент Spring, що обробляє запит. Він дозволяє компонентам Spring MVC впроваджуватися і вільно взаємодіяти шляхом запуску і обробки подій.

Нижче описаний контролер AnimalController, що містить безпосередньо весь функціонал роботи з запитами щодо анкет тварин. Так як контролер – це обробник клієнтських запитів, він містить такі методи, як отримання списку анкет, додання нової анкети, видалення та редагування існуючих та інші. У своїй реалізації він працює з сервісами.

```
package com.example.demo.admin;
```


```

@Controller
public class AnimalController {

    private AnimalService service;
    private UserService userService;
    private DonationService donationService;
    private MyUserDetailsService userDetailsService;
    private FavouriteService favouriteService;
    private CompanyService companyService;

    @Autowired
    public void setUserDetailsService(MyUserDetailsService userDetailsService) {
        this.userDetailsService = userDetailsService;
    }

    @Autowired
    public void setService(AnimalService service) {
        this.service = service;
    }

    @Autowired
    public void setUserService(UserService service) {
        this.userService = service;
    }

    @Autowired
    public void setFavouriteService(FavouriteService service) {
        this.favouriteService = service;
    }

    @GetMapping("/admin/child_list")
    public String adminChildList(Model model) {
        model.addAttribute("children", service.getAllChildren());
        return "admin/child_list";
    }

    @GetMapping("/admin/child")
    public String processAddStudentForm(Model model, @RequestParam int id) {
        model.addAttribute("child", service.getChildById(id));
        return "admin/child";
    }

    @PostMapping("/admin/child")
    public String processAddFavourite(Model model, @RequestParam long id,
    Principal principal) {
        Child child = service.getChildById(id);
        Favourite favourite = new Favourite(child.getId(),
        child.getFirstName(), child.getYearOfBirth(),

```

```

        child.getGender(),    child.getCode(),    child.getHobby(),
child.getPhotoName(), child.getIllness());
final MyUserDetails userDetails
userDetailsService.loadUserByUsername(principal.getName());
long user_id = userDetails.getUser_id();
User user = userService.getUserById(user_id);
favouriteService.addFavourite(favourite, user_id);
userService.saveUser(user);
model.addAttribute("favourites", user.getFavourites());
return "admin/favourites";
}

@GetMapping("/index")
public String reindex(Model model) {
    model.addAttribute("children", service.getAllChildren());
    return "index";
}
@GetMapping("/details")
public String childDetails(Model model) {
    model.addAttribute("children", service.getAllChildren());
    return "details";
}

@GetMapping("/donation")
public String donation(Model model) {
    model.addAttribute("company", companyService.getCompanyById(1));
    return "donation";
}

@GetMapping("/donations_list")
public String donationsList(Model model) {
    model.addAttribute("donations",
donationService.getAllDonations());
    return "donations_list";
}

@GetMapping("/about_project")
public String aboutProject() {
    return "about_project";
}

@GetMapping("/contacts")
public String contacts(Model model) {
    model.addAttribute("company", companyService.getCompanyById(1));
    return "contacts";
}

// normalize the file path
String fileName
StringUtil.cleanPath(file.getOriginalFilename());

// save the file on the local file system
try {
    Path path
Paths.get("/Users/disvik/Documents/ONPU/2course2sem/oop/course_work/src/main/resources/static/images/" + fileName);

```

```

        Files.copy(file.getInputStream(),
path,
StandardCopyOption.REPLACE_EXISTING);
        child.setPhotoName("../images/" + fileName);
    } catch (IOException e) {
        e.printStackTrace();
    }
    if(child.getIllness().length()<5)
        child.setIllness(null);
service.saveChild(child);
return "redirect:/admin/index";
}

// normalize the file path
String fileName
StringUtils.cleanPath(file.getOriginalFilename());

// save the file on the local file system
try {
    Path path
Paths.get("/Users/disvik/Documents/ONPU/2course2sem/oop/course_work/src/main/resources/static/images/" + fileName);
    Files.copy(file.getInputStream(),
path,
StandardCopyOption.REPLACE_EXISTING);
    child.setPhotoName("../images/" + fileName);
} catch (IOException e) {
    e.printStackTrace();
}
if(child.getIllness().length()<5)
    child.setIllness(null);
service.saveChild(child);
return "redirect:/admin/animal_list";
}

```

4.2.5 Особливості використання спеціалізованих бібліотек та API

Для реалізації пожертвування треба застосувати стороннє платіжне API.

Буде використано API FONDY 1.0.

Взаємодія з API відбувається наступним шляхом:

1. Клієнт на сайті вводить суму та інші ознаки пожертвування.
2. ПП на сайті відображає кнопку оплати, при натисканні на яку буде сформована HTML форма для відправки на платіжний шлюз FONDY
3. На сайті користувач натискає кнопку оплати, і торговець перенаправляє покупця в браузері методом редиректу на url <https://api.fondy.eu/api/checkout/redirect/>, передаючи параметри методом HTTPS POST

4. Користувач вводить платіжні реквізити на сайті платіжного шлюзу FONDY, платіжний шлюз здійснює списання коштів через зовнішню платіжну систему (банк-еквайєр)
5. Платіжний шлюз FONDY, отримавши від зовнішньої платіжної системи відповідь з результатом платежу, перенаправляє покупця в браузері на сайт торговця методом редиректу, передаючи параметри з результатом оплати методом HTTPS POST. ПП на сайті відображає сторінку з результатом оплати.
6. Для більшої надійності доставки відповіді про результат платежу на бік торговця, платіжний шлюз FONDY також дублює відповідь з результатами оплати на сервер сайту торговця методом host-to-host HTTPS POST.

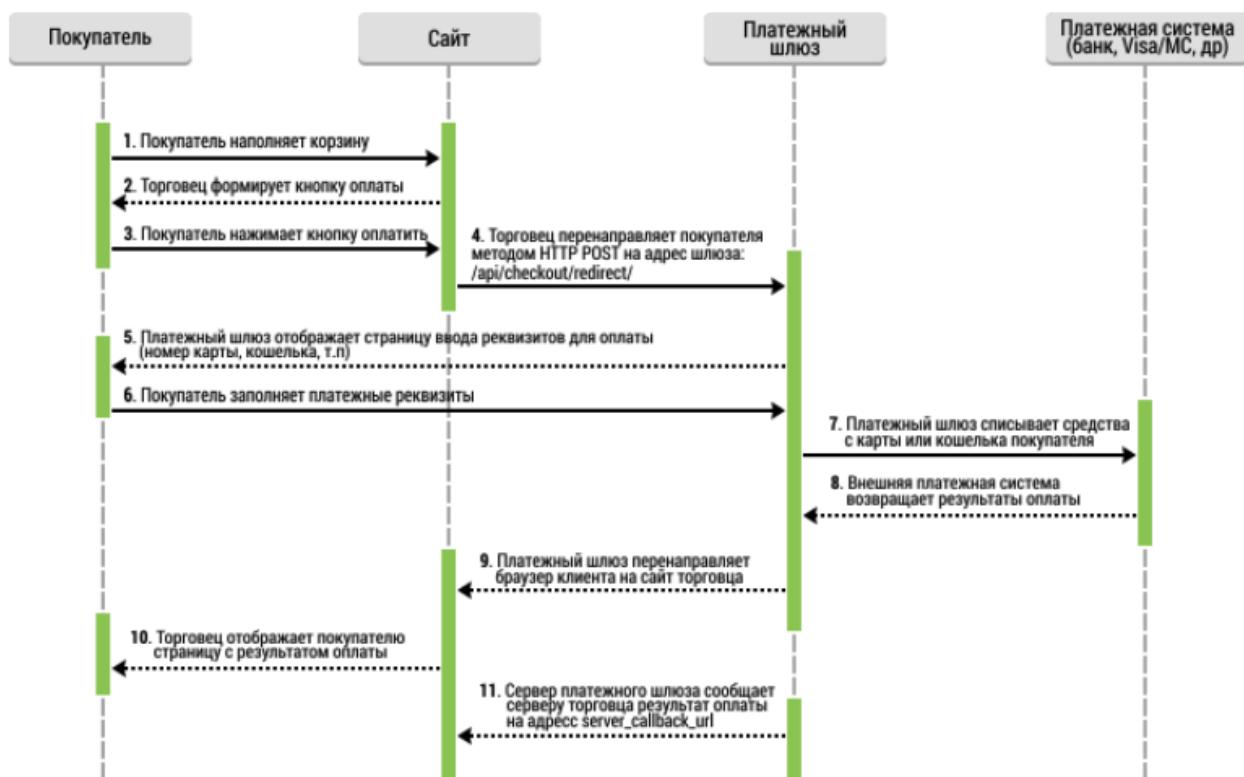


Рис. 4.2.5 – Схема взаємодії застосунку з зовнішнім API

4.3 Тестування програмних модулів

4.3.1 Тестування методу EditUser()

EditUser() являє собою метод, що перевіряє введені користувачем під час реєстрації поля та проводить їхню валідацію. Метод виглядає наступним образом:

```
@PostMapping("/edit_user")
public String editUser(@Valid User user, BindingResult bindingResult) {
    if (bindingResult.hasErrors()) {
        return "registration_form";
    }
    user.setActive(true);
    user.setRoles("ROLEUSER");
    user.setPassword(passwordEncoder.encode(user.getPassword()));
    userService.saveUser(user);
    return "redirect:/";
}
```

Тобто введені значення полей перевіряються відповідно до обмежень, що накладено на них у певному класі. Якщо введенне значення відповідає обмеженню, то значення полей класів додаються у відповідні поля таблиць в базу даних (оновляються).

1. Розглянемо випадок, коли на вхід поступає **пустий об'єкт**.

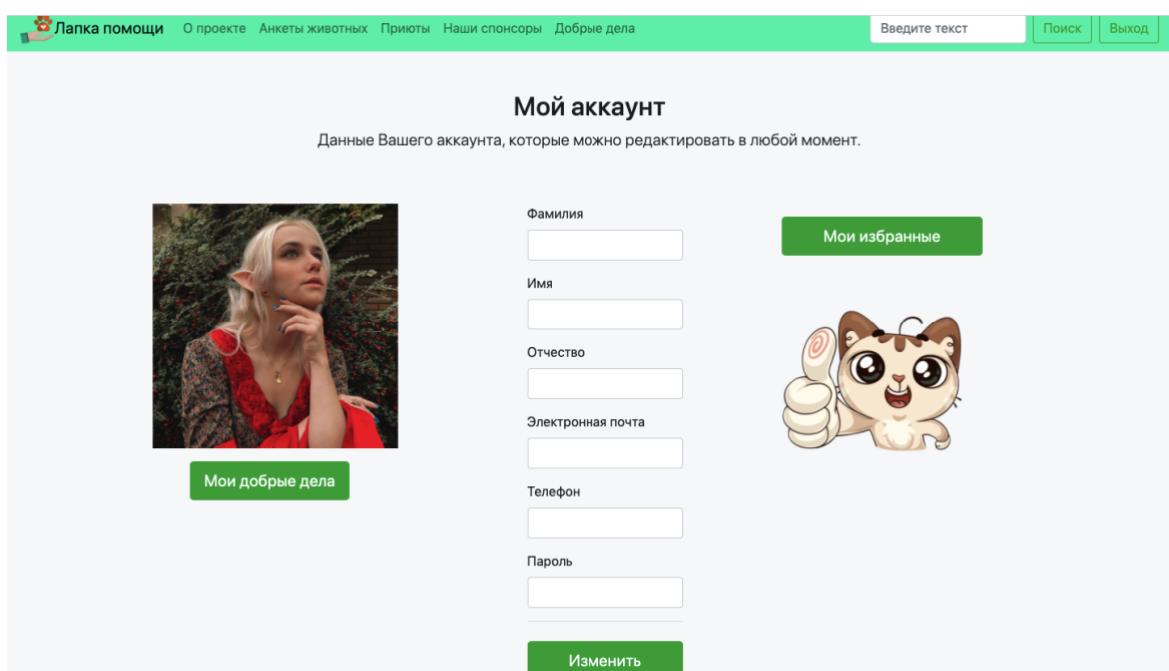


Рис. 4.3.1.1 – Пустий об'єкт під час редагування

Система працює таким чином, що повідомлення про помилку передається прямо на екран за допомогою змінної *bindingResult*. Та, звичайно, у консолі нема повідомлення про додавання нового об'єкту. Користувача перенаправлено знову на сторінку аккаунта, виведено повідомлення про помилки та об'єкт не поновлено.

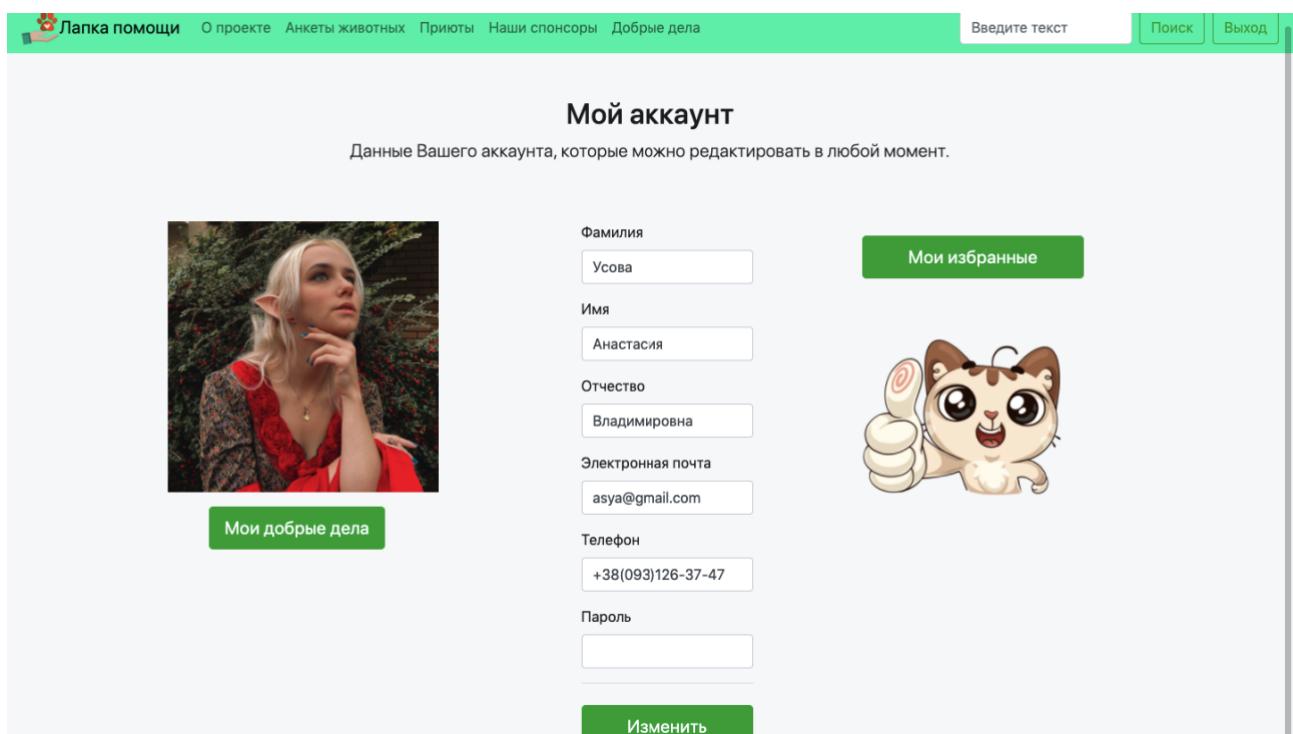


Рис. 4.3.1.2 – Повернення користувача на попередню сторінку

2. Розглянемо випадок, коли на вход поступає об'єкт з усіма правильними полями {Усова, Ася, Владимирівна, asya@gmail.com, +380634564576, 1234}

Система працює таким чином, у консолі виводиться повідомлення про редагування об'єкту. Бачимо наступне:

```
Hibernate: select user0_.user_id as user_id1_8_, user0_.active as active2_8_, user0_.city as city3_8_, user0_.first_name as first_na4_8_, user0_.last_name as last_nam5_8_, user0_.passwd as passwd6_8_, user0_.patronymic as patronym7_8_, user0_.phone as phone8_8_, user0_.photo_name as photo_na9_8_,
```


```
user0_.roles as roles10_8_, user0_.user_name as user_name11_8_ from users user0_
where user0_.user_name=?
```

Отже, об'єкт поновлено успішно та користувача перенаправлено у його особистий кабінет, що містить щойно введені дані.

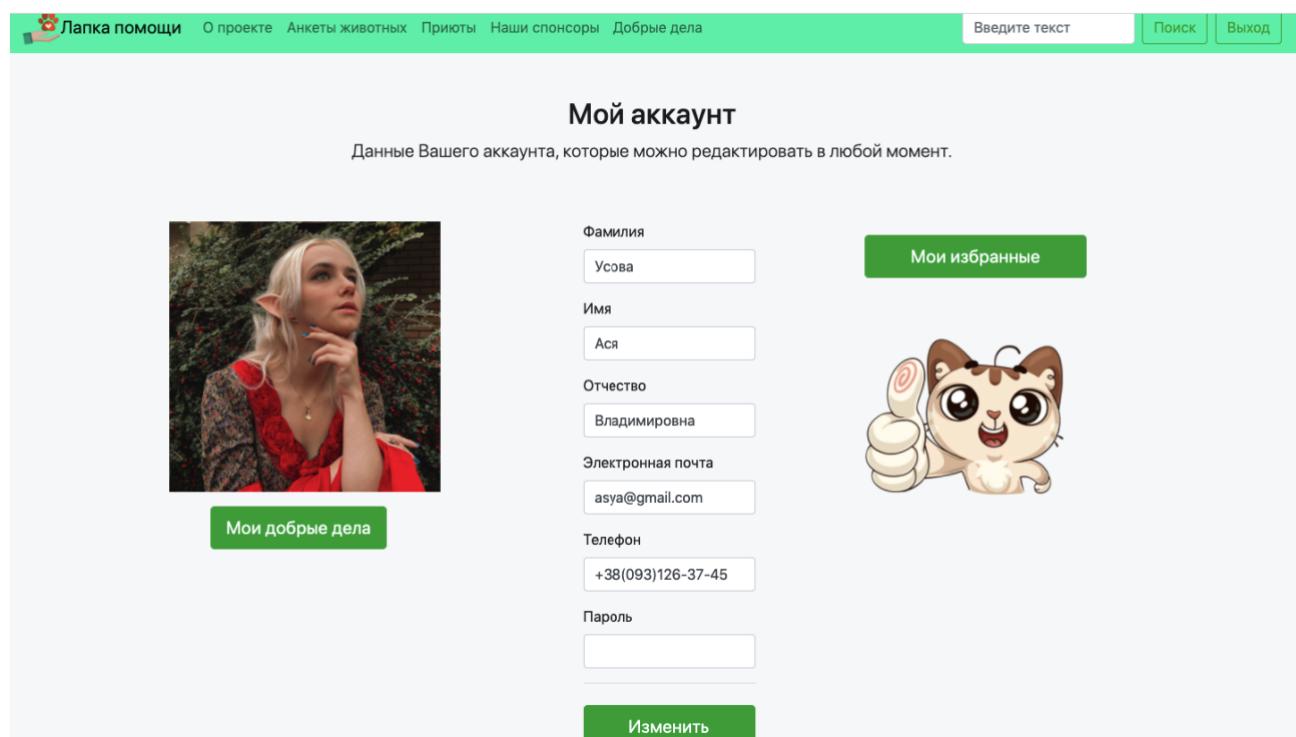


Рис. 4.3.1.3 – Успішне поновлення даних

4.3.2 Тестування методу AddShelter()

AddAnimal() являє собою метод, що перевіряє введені користувачем під час додавання нового притулку поля та проводить їхню валідацію. Цей метод є доступним лише у адмінській версії сайту. Метод виглядає наступним образом:

```
@PostMapping("/admin/add_shelter")
public String processAddShelterForm(@Valid Shelter shelter,
BindingResult bindingResult, @RequestParam("file") MultipartFile file) {
    if (bindingResult.hasErrors()) {
        return "admin/add_shelter";
    }
    // normalize the file path
    String fileName =
StringUtils.cleanPath(file.getOriginalFilename());
    // save the file on the local file system
```

```

        shelterService.saveShelter(shelter);
        return "redirect:/admin/shelter_list";
    }

```

Тобто введені значення полей перевіряються відповідно до обмежень, що накладено на них у певному класі. Якщо введене значення відповідає обмеженню, то фото зберігається на відповідний ресурс у базі даних та значення полей класів додаються у відповідні поля таблиць в базу даних.

1. Розглянемо випадок, коли на вхід поступає **пустий об'єкт**.

Добавление нового приюта
Заполните поля и нажмите кнопку 'Добавить приют'

Поля для заполнения

Название	Фотография <input type="button" value="Выберите файл"/> Файл не выбран
Электронная почта	Телефон
Описание	
Реквизиты	

Добавить приют

Рис. 4.3.2.1 – Пустий об'єкт під час додавання нового притулку

Система працює таким чином, що повідомлення про помилку передається прямо на екран за допомогою змінної *bindingResult*. Та, звичайно, у консолі нема повідомлення про додавання нового об'єкту. Користувача (адміністратора) перенаправлено знову на сторінку додавання анкети знову, виведено повідомлення про помилки та об'єкт не створено.

Добавление нового приюта

Заполните поля и нажмите кнопку 'Добавить приют'

Поля для заполнения

<p>Название</p> <input type="text"/> <p>Name should be from 2 to 50 characters</p>	<p>Фотография</p> <input type="file" value="Выберите файл"/> <p>Файл не выбран</p>
<p>Электронная почта</p> <input type="text"/> <p>Invalid email format</p>	<p>Телефон</p> <input type="text"/> <p>invalid phone number</p>
<p>Описание</p> <input type="text"/> <p>About should be from 2 to 50 characters</p>	
<p>Реквизиты</p> <input type="text"/> <p>Card number should be from 16 to 20 characters</p>	

Рис. 4.3.2.2 – Повідомлення про помилку під час тесту №1

3. Розглянемо випадок, коли на вхід поступає **об'єкт з неправильним полем електронна адреса** {Доброе сердце, dobro, 4441 4456 7543 6574}

Система працює таким чином, що повідомлення про помилку передається прямо на екран за допомогою змінної *bindingResult*. На пошту накладено регулярне вираження *regexp* = "^{^([a-zA-Z0-9_-]+\\.)*[a-zA-Z0-9_-]+@[a-zA-Z0-9_-]+(\\.[a-zA-Z0-9_-]+)*\\.[a-zA-Z]{2,6}\$}". Та, звичайно, у консолі нема повідомлення про додавання нового об'єкту. Користувача перенаправлено знову на сторінку реєстрації, виведено повідомлення про помилки та об'єкт не створено

Поля для заполнения

Название	Фотография
Доброе сердце	<input type="button" value="Выберите файл"/> Файл не выбран
Электронная почта	Телефон
dobro	+38(093)126-37-41
Invalid email format	
Описание	
Находится в Одессе по адресу Пушкинская, 5	
Реквизиты	
4441 1144 5542 6654	

Рис. 4.3.2.3 – Повідомлення про помилку під час тесту №2

4. Розглянемо випадок, коли на вхід поступає **об'єкт з усіма правильними полями** {Доброе сердце, dobro@gmail.com, 4441 4456 7543 6574}

Система працює таким чином, у консолі виводиться повідомлення про додавання нового об'єкту. Бачимо наступне:

```
Hibernate: select nextval ('hibernate_sequence')
Hibernate: insert into animal (age, code, first_name, gender,
description, illness, photo_name, year_of_birth, id) values (?, ?, ?, ?, ?, ?,
?, ?, ?, ?)
```

Отже, об'єкт створено успішно та користувача (адміністратора) перенаправлено на сторінку з переліком тварин, що вже містить щойно введену інформацію.

Поля для заполнения

Название	Фотография
Доброе сердце	Выберите файл 0101010.jpg
Электронная почта	Телефон
dobro@gmail.com	+38(093)126-37-41
Описание	
Находится в Одессе по адресу Пушкинская, 5	
Реквизиты	
4441 1144 5542 6654	

Добавить приют

Рис. 4.3.2.4 – Успішне введення даних

Приюты нашего города

Тут представлен список приютов, которые нуждаются в вашей помощи и поддержке.

[Подробнее...](#) [Добавить приют](#)



Доброе сердце

Рис. 4.3.2.5 – Демонстрація успішного додавання анкети

							IC KP 122 AI-183 ПЗ	

5 Розгортання та валідація програмного продукту

пояснювальної записки курсової роботи

5. 1 Інструкція з встановлення програмного продукту

Розроблене програмне забезпечення підтримується усіма веб-браузерами, усіма версіями як на ОС Windows, Mac OS , так и на Linux ОС.

Здійснювати дії на веб-сервісі та користуватися ним користувач може за допомогою маніпулятора «миша» та клавіатури. За допомогою маніпулятора «миша» користувач може нажати на кнопку/текст, а за допомогою клавіатури – вводити дані у різні поля та форми.

Система також має підтримуватися у всіх веб-браузерах мобільних пристрій та усі дії будуть реалізовані користувачем за допомогою сенсора. Але, на жаль, адаптування мобільної версії ще не розроблено та вона виглядає так же само, як і на десктопній версії, що є незручним до користувача. Але у подальшому адаптація для мобільних пристрій теж буде розроблена.

5.2 Інструкція з використання програмного продукту

5.2.1 Додавання притулку

ПП надає можливість користувачу «адміністратор» ввести параметри додавання нового притулку (назва, пошта, телефон, опис, реквізити) (для цього треба нажати на кнопку «додати притулок» (рис.5.2.1.1) , заповнювати – як показано на рисунку 5.2.1.2.

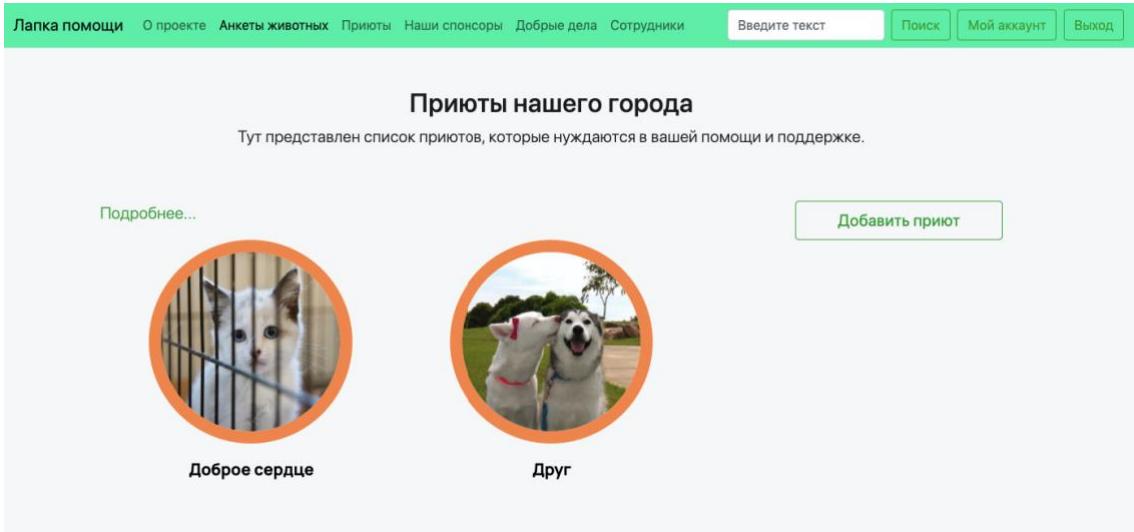


Рисунок 5.2.1.1 – Приклад еcrannoї форми списку притулків

Рисунок 5.2.1.2 – Приклад еcrannoї форми додавання притулку
Для додавання анкети користувач повинен ввести значення, як показано на рисунку 5.2.1.3.

При цьому необхідно пам'ятати про обмеження значення назви та опису – строки від 2 до 50 символів, , на пошту накладено регулярне вираження $regexp = "^(a-zA-Z0-9_-)+\.\)*[a-zA-Z0-9_-]+@[a-zA-Z0-9_-]+\(\.\.[a-zA-Z0-9_-]+\)*\.\{2,6}$, введена пошта має йому відповідати (масці

xxx@xxx.xxx). На телефон – }\$""^|\|+\|\d{2}\|\(\|\d{3}\|\)\|\d{3}-\d{2}-\d{2}\$.
Тобто маска +(xxx)-xxx-xx-xx.

Добавление нового приюта
Заполните поля и нажмите кнопку 'Добавить приют'

Поля для заполнения

Название <input type="text" value="Котики"/>	Фотография <input type="file" value="Выберите файл cats.jpeg"/>
Электронная почта <input type="text" value="cats@gmail.com"/>	Телефон <input type="text" value="+38(093)126-37-40"/>
Описание <input type="text" value="Приют для котов. Одесса, Преображенская, 5"/>	
Реквизиты <input type="text" value="4441 1144 5542 6654"/>	

Добавить приют

Рисунок 5.2.1.3 – Приклад екранної форми заповлення даних під час додавання нового притулку

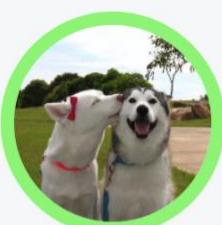
Після додавання анкети адміністратор потрапляє до списку притулків, куди додано новий (рис. 5.2.1.4).

Лапка помощи О проекте Анкеты животных Приюты Наши спонсоры Добрые дела Введите текст Помощь Вход

Приюты нашего города
Тут представлен список приютов, которые нуждаются в вашей помощи и поддержке.
[Подробнее...](#)



Доброе сердце



Друг



Котики

Рисунок 5.2.1.4 – Приклад екранної форми зі списком притулків

5.2.2 Редагування аккаунту та додавання тварини в обране

Після входу користувач потрапляє на головну сторінку (рис. 5.2.2.1), та може увійти, наприклад, у свій кабінет (рис. 5.2.2.2). Там він може редагувати усі дані за такою ж самою схемою, як і процес реєстрації. Також користувач може переглянути добрі справи, які зробив для нашого проекта та тварин, яких додав до обраних (рис. 5.2.2.3), щоб повернутися к ним після.

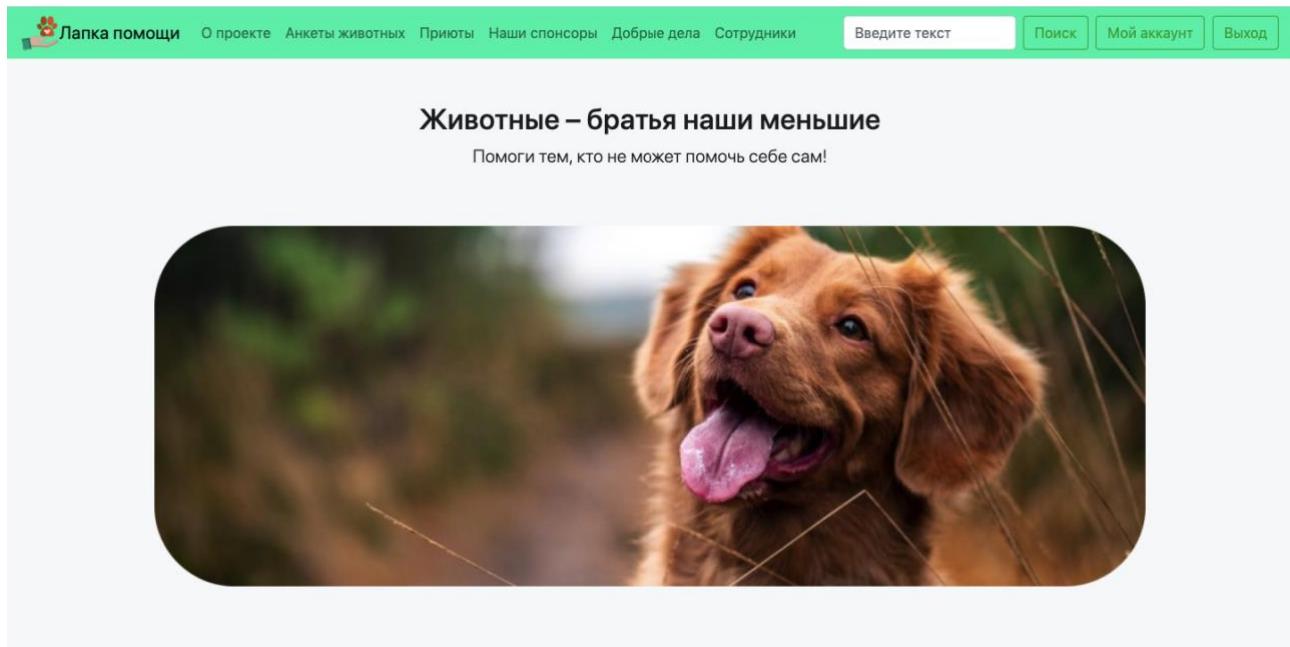


Рисунок 5.2.2.1 – Приклад екранної форми головної сторінки авторізованого користувача

 **Лапка помощи** [О проекте](#) [Анкеты животных](#) [Приюты](#) [Наши спонсоры](#) [Добрые дела](#) [Поиск](#) [Выход](#)

Мой аккаунт

Данные Вашего аккаунта, которые можно редактировать в любой момент.



[Мои добрые дела](#)

Фамилия

Имя

Отчество

Электронная почта

Телефон

Пароль

Мои избранные



[Изменить](#)

Рисунок 5.2.2.2 – Приклад еcrannoї форми аккаунта користувача

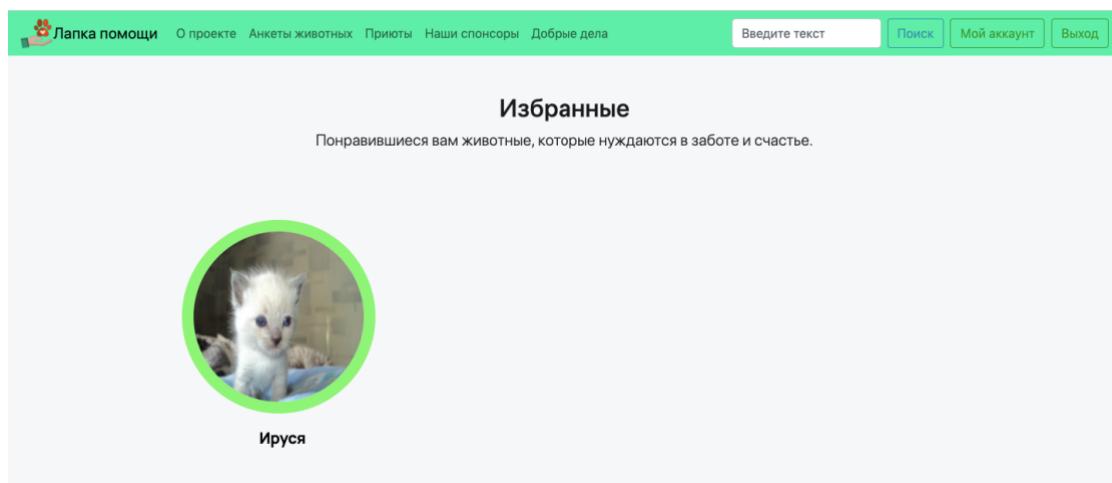


Рисунок 5.2.2.3 – Приклад еcrannoї форми обраних тварин користувача

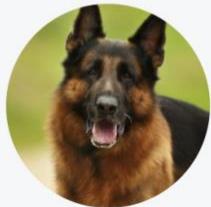
Щоб додати тварину до обраних, треба перейти на сторінку «анкети тварин» (рис. 5.2.2.4) та обрати конкретну анкету (рис. 5.2.2.5), де нажати на «додати до обраних».

Ищу хозяев!

Тут представлены анкеты животных из приютов и просто брошенных питомцев, которые нуждаются в вашей заботе.

[Подробнее...](#)

Персик

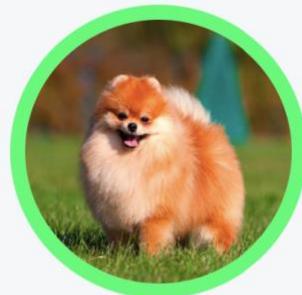


Мухтар

Рисунок 5.2.2.4 – Приклад экранної форми зі списком тварин

Ищу хозяев!

Тут представлена анкета питомца, который очень нуждается в заботе и помощи.

**Имя:**

Персик

Дата рождения:

2019 год

О питомце:

Персик – прелестный и добрый шпиц.

Болезни:

Отсутствуют.

Мои спонсоры[В избранные](#)[Сделать добро](#)

Рисунок 5.2.2.5 – Приклад экранної форми з анкетою конкретної

тварини

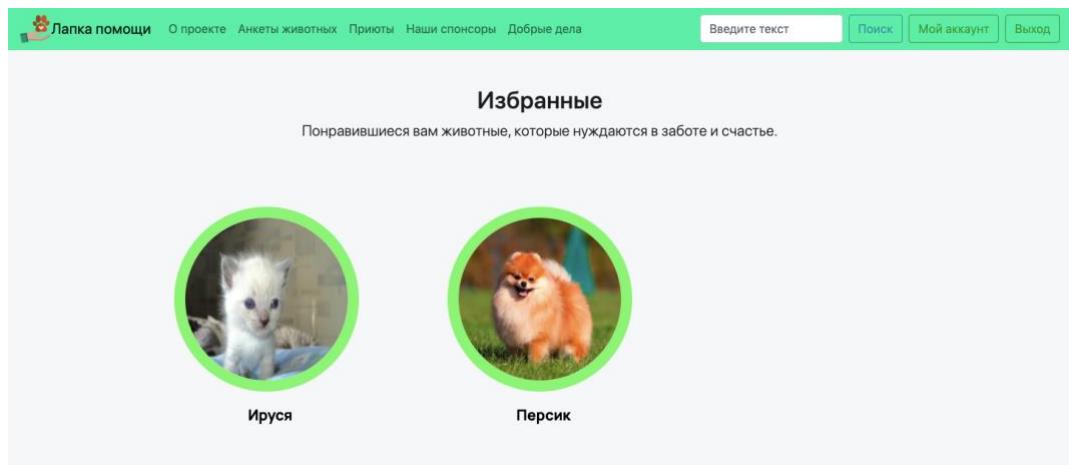


Рисунок 5.2.2.6 – Результат після додавання ще одної тварини до обраних

Коли на анкеті тварини користувач натискає на «зробити добро», йому відкривається вікно з реквізитами (рис. 5.2.2.7).

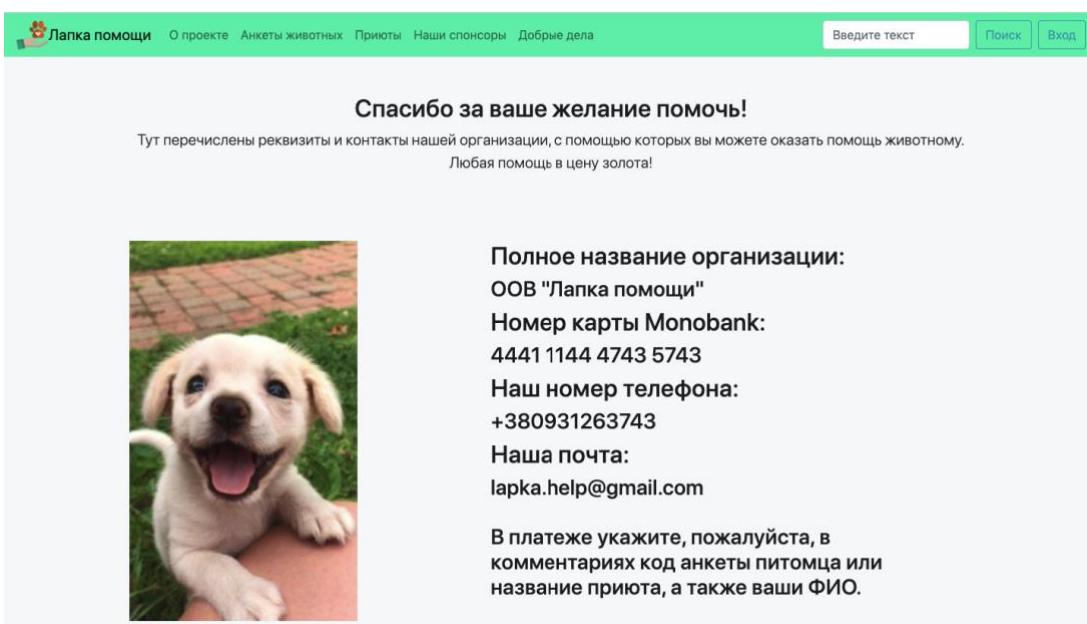


Рисунок 5.2.2.7 – Приклад екранної форми зі здійсненням пожертвування

5.3 Результати валідації програмного продукту

Метою програмного продукту було підвищення рівня доступності до інформації про втрачених, знайдених і безпритульних тварин в м Одеса,

поширення інформації про можливі варіанти допомоги притулкам на підставі створення веб-сайту для об'єднання необхідної інформації.

В даний період часу проект знаходиться на ранній стадії свого розвитку, але внаслідок буде розвинений більше. Його буде глибше впроваджено в процес пожертвувань Одеському притулку для тварин №1, що дозволить в майбутньому наочно побачити зміни в статистиці пожертвувань і в принципі фінансового становища закладу.

У розробленому ПП доступна інформація про безпритульних тварин, притулки, способи пожертвування, отже мета підвищення рівня доступності інформації на цю тему виконана.

Розробка подібних інформаційних систем може залучити громадянське суспільство до проблеми притулків для тварин, потенційно зменшити кількість безпритульних тварин. Створена інформаційна система «Лапка допомоги» робить пожертвування доступним, зручним і легким для будь-якого користувача. Її широкий функціонал надає користувачеві безліч можливостей дізнатися про проблему і прийняти участь в її вирішенні. Вона повинна вплинути на фінансове становище притулків, на які спрямована і благополуччя тварин в ньому.

Висновки

В результаті створення програмного продукту була досягнута наступна мета його споживача: «Підвищення рівня доступності до інформації про втрачених, знайдених і безпритульних тварин в м Одеса, поширення інформації про можливі варіанти допомоги притулкам на підставі створення веб-сайту для об'єднання необхідної інформації».

Доказом цього є наступні факти. Програмний продукт «Лапка допомоги» об'єднує притулки для тварин, оголошення, пов'язані з пошуком загубленої

тварини або пошуком для неї нового хазяїна та готелі для перетримки домашніх улюблениців.

«Лапка допомоги» задовольняє такі потреби споживача:

- 1) швидкий пошук необхідної інформації;
- 2) своєчасне знаходження загубленої тварини;
- 3) швидкий пошук нового домашнього улюбленця;
- 4) зменшення кількості безпритульних тварин;
- 5) великий вибір готелів для перетримки тварин;
- 6) фінансування притулків;
- 7) пошук волонтерів для притулків.

В процесі створення програмного продукту виникли такі труднощі:

- 1) організаційні труднощі роботи у команді;
- 2) брак часу;
- 3) відсутність досвіду у front-end розробці.

Через вищеописані непередбачені труднощі, а також через обмежений час на створення програмного продукту, залишилися нереалізованими такі прецеденти або їх окремі кроки роботи:

- 1) зв'язок з хазяїнами тварин напряму;
- 2) публікація об'яв та їхня модерація;
- 3) здійснення пожертвування напряму через платіжне API;
- 4) здійснення пожертвування (всьому притулку, а не лише конкретній тварині);
- 5) налаштування регулярного пожертвування;
- 6) комунікація (тих, кто публікує об'яви та відгукається на них).

Зазначені недоробки планується реалізувати в майбутніх курсових роботах з урахуванням тем дисциплін наступних семестрів.
